Prediction Project: Life Expectancy

R notebook using data from [Private Datasource] · 104 views · 6mo ago

In [1]:

```r
# This R environment comes with many helpful analytics packages install
ed
# It is defined by the kaggle/rstats Docker image: https://github.com/k
aggle/docker-rstats
# For example, here's a helpful package to load

library(tidyverse) # metapackage of all tidyverse packages

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) w
ill list all files under the input directory

list.files(path = "../input")

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save & R
un All"
# You can also write temporary files to /kaggle/temp/, but they won't b
e saved outside of the current session
```

```
── Attaching packages ─────────────────────────────────────── tidy
verse 1.3.0 ──

✔ ggplot2 3.3.3     ✔ purrr   0.3.4
✔ tibble  3.0.6     ✔ dplyr   1.0.4
✔ tidyr   1.1.2     ✔ stringr 1.4.0
✔ readr   1.4.0     ✔ forcats 0.5.0

── Conflicts ────────────────────────────────────────── tidyverse_co
nflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

'life-expectancy'

By: Devin Bunch and Claire Field

Which US socioeconomic factors have an impact on life expectancy?

To answer this question, we conduct a series of predictive tests to determine if economic factors like total schooling completed or average income do in fact predict life expectancy. Before running these tests, we must load the following packages, the data we are going to be using (which we created using IPUMS and IHME), and split the data into train and test data.

In [2]:

```
#Load packages for linear regression, decision trees, elasticnet logist
ic regression, boosting, and random forests
library(pacman)

p_load(tidyverse, tidymodels, skimr, janitor, magrittr,
       datasets, rpart.plot, baguette, glmnet, tune, haven, ranger, d
ata.table, parallel)
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

also installing the dependency ‘butcher’



baguette installed
```

In [3]:

```
#Load the data created from IPUMS and IHME
life_df <- read.csv("../input/life-expectancy/DataSet.csv")

life_df %<>% clean_names()

#Look to see what we have going on
#Reader beware: the output here is very long because of the large numbe
r of predictors
life_df %>% skim
```

```
Name                       Piped data
Number of rows             3143
Number of columns          156
_______________________
Column type frequency:
  factor                   5
  numeric                  151
________________________
Group variables            None

── Variable type: factor ─────────────────────────────────────────────
──────────
  skim_variable n_missing complete_rate ordered n_unique
1 gisjoin               0             1 FALSE       3143
2 year                  0             1 FALSE          1
3 state                 0             1 FALSE         51
4 county                0             1 FALSE       1877
5 area_name             0             1 FALSE       3143
  top_counts
1 G01: 1, G01: 1, G01: 1, G01: 1
2 200: 3143
3 Tex: 254, Geo: 159, Vir: 134, Ken: 120
4 Was: 30, Jef: 25, Fra: 24, Jac: 23
5 Abb: 1, Aca: 1, Acc: 1, Ada: 1

── Variable type: numeric ────────────────────────────────────────────
──────────
    skim_variable
  1 female_life_expectancy_2010
  2 male_life_expectancy_2010
  3 avg_life_expectancy
  4 total_means_of_transportation_to_work
  5 car_truck_or_van
  6 car_truck_or_van_drove_alone
  7 car_truck_or_van_carpooled
  8 car_truck_or_van_carpooled_in_2_person_carpool
  9 car_truck_or_van_carpooled_in_3_person_carpool
 10 car_truck_or_van_carpooled_in_4_person_carpool
 11 car_truck_or_van_carpooled_in_5_or_6_person_carpool
 12 car_truck_or_van_carpooled_in_7_or_more_person_carpool
 13 public_transportation_excluding_taxicab
 14 public_transportation_excluding_taxicab_bus_or_trolley_bus
 15 public_transportation_excluding_taxicab_streetcar_or_trolley_car
 16 public_transportation_excluding_taxicab_subway_or_elevated
 17 public_transportation_excluding_taxicab_railroad
 18 public_transportation_excluding_taxicab_ferryboat
 19 taxicab
 20 motorcycle
 21 bicylce
 22 walked
```

In [4]:
```
#Set a seed to always grab the same subset of data

set.seed(123)

#Split the data into train (80%) and test (20%) using initial_split()

train_test_split = life_df %>% initial_split(prop=0.8)

train_df = train_test_split %>% training()

test_df = train_test_split %>% testing()
```

Model 1: Linear Regression

Model 1 Score: 610/628

For this linear regression, we decided to look at life expectancy for males and females separately, as we have data for each sex as well as general data. We wanted to see if the following factors were good predictors for life expectancy: having no school, having a doctorate degree, living in a household with a single parent, the median income of the household, working in law enforcement, median value of a house, working from home, or taking public transportation.

In [5]:
```
#Fit the lin reg model to the training data


est_reg_f =
    linear_reg() %>%
    set_engine("lm") %>%
    fit(female_life_expectancy_2010 ~ female_no_schooling_completed +
female_doctorate_degree +
        family_households_other_family_female_householder_no_husband_
present +
        median_household_income_in_past_12_months +
        female_service_occupations_protective_service_occupations_law
_enforcement_workers_including_supervisors +
        median_value_of_owner_occupied_housing_units_dollars +
        worked_at_home + public_transportation_excluding_taxicab, dat
a = train_df)
```

```r
est_reg_f

est_reg_m =
    linear_reg() %>%
    set_engine("lm") %>%
    fit(male_life_expectancy_2010 ~ male_no_schooling_completed + mal
e_doctorate_degree +
        family_households_other_family_male_householder_no_wife_prese
nt +
        median_household_income_in_past_12_months +
        male_natural_resources_construction_and_maintenance_occupatio
ns +
        median_value_of_owner_occupied_housing_units_dollars +
        motorcycle + worked_at_home, data = train_df)

est_reg_m
```

```
parsnip model object

Fit time:  10ms

Call:
stats::lm(formula = female_life_expectancy_2010 ~ female_no_schooling_
completed +
    female_doctorate_degree + family_households_other_family_female_ho
useholder_no_husband_present +
    median_household_income_in_past_12_months + female_service_occupat
ions_protective_service_occupations_law_enforcement_workers_including_
supervisors +
    median_value_of_owner_occupied_housing_units_dollars + worked_at_h
ome +
    public_transportation_excluding_taxicab, data = data)

Coefficients:

(Intercept)

7.591e+01

female_no_schooling_completed

7.356e-06

female_doctorate_degree
```

```
In [6]:
#Predict the lin reg model onto the test data

y_hat_f = predict(object=est_reg_f, new_data=test_df)

y_hat_m = predict(object=est_reg_m, new_data=test_df)

skim(y_hat_f)

skim(y_hat_m)
```

```
── Data Summary ────────────────────────
                           Values
Name                       y_hat_f
Number of rows             628
Number of columns          1
_______________________
Column type frequency:
  numeric                  1
________________________
Group variables            None

── Variable type: numeric ────────────────────────────────────────────
──────────
  skim_variable n_missing complete_rate  mean    sd    p0   p25   p50
p75
1 .pred                 1         0.998  79.8  1.31  76.3  79.0  79.6
80.2
   p100 hist
1  89.0 ▁▇▁▁▁
── Data Summary ────────────────────────
                           Values
Name                       y_hat_m
Number of rows             628
Number of columns          1
_______________________
Column type frequency:
  numeric                  1
________________________
Group variables            None

── Variable type: numeric ────────────────────────────────────────────
──────────
  skim_variable n_missing complete_rate  mean    sd    p0   p25   p50
p75
```

```
    p100 hist
1   83.6 ▅▇▂▁▁
```

In [7]:

```
#Now we want to see if our predictions match our testing data by subtra
cting our
#control values from the male and female predictions we've made to see
how "off"
#we are with zero being the exact right prediction

#give me a data frame with just avg_life_expectancy from our testing da
ta for both male and female

#select out only the average life expectancies for men and women in the
tesing data set

ale_only_f <- test_df[5]

ale_only_m <- test_df[6]
```

In [8]:

```
#Here, we will combine our four colummns of data

#Now we want to see if our predictions for male and female life expecta
ncy fit our testing data

#to do this, let's create a new column of data where our predictions ar
e subtracted
#from our test data to see how much error we have. The quantity of "zer
o" in this new dataframe
#represents an exact match of prediction to test data.

#create one big data set all together with our male and female test and
trainging datatsets
```

In [9]:

```
#compare the estimates on the test data with the actual data averages f
rom the test data

#numbers from the test data that we selected
a1 <- ale_only_f

a2 <- ale_only_m

#average predicted ages someone will die from the test data that we fou
nd
```

```
a3 <- y_hat_m

a4 <- y_hat_f

#This is our new data set with all relevant results

#the last two columns are the difference of the traing model's best pre
diction
#result from our actual data for men and women

df1 <- cbind(a1, a2, a3, a4, a5 <- (a2-a3), a6<- (a1-a4))

df1
```

A data.frame: 628 × 6

	female_life_expectancy_2010	male_life_expectancy_2010	.pred	.pred
	<dbl>	<dbl>	<dbl>	<dbl>
8	75.80	70.80	73.68222	79.01479
10	77.07	72.23	74.16294	79.31016
13	77.35	72.29	72.50358	78.21935
16	78.36	74.05	74.34376	79.48087
24	75.44	68.57	72.25671	77.96408
31	78.06	72.17	73.32305	78.75857
41	78.75	76.34	74.14006	79.34072
49	77.81	71.72	73.74922	78.85365
52	77.85	72.89	74.50605	79.61143
71	80.06	75.80	75.67634	80.36285
76	80.06	75.80	75.40505	80.20383
79	80.06	75.80	76.47328	80.98579
81	80.06	75.80	76.79865	81.14582
83	80.06	75.80	77.67919	81.89860
88	80.06	75.80	74.92855	79.86793
94	80.06	75.80	75.54544	80.26550
95	80.06	75.80	77.30788	81.44469
108	82.42	78.10	75.73409	79.84283
117	77.19	72.73	72.89582	78.41215
120	76.41	69.79	71.77831	77.68608
127	78.61	72.66	74.13412	79.30289
134	79.73	74.39	74.69867	79.74715
137	78.46	72.24	73.75477	79.08443
141	78.65	71.84	73.67759	78.97047
151	77.99	72.87	73.51661	78.78875

In [10]:

```
#Now I want to create a means of comparing our four models

#To do this, we can use a 95% confidence interval, and count how
#many successful predictions we get for each model.

#so for this first model here, let's calculate how many predictions com
e within 95% of the real test value.

#this will give us the MALE +/- column for the amount we will need to b
e within 0.05 % of the real.

m_error_allowance <- a1 %>% mutate(a1 <- 0.05*(a1[]))

#print output to check
#I hid this output so it doesn't print when it runs, I ran it before to
make sure it worked

#m_error_allowance

#now for the FEMALE
f_error_allowance <- a2 %>% mutate(a5 <- 0.05*(a2[]))

#print output to check
#I hid this output so it doesn't print when it runs, I ran it before to
make sure it worked
#f_error_allowance


#We can also see our data frame is still correct at 628 by 1.

#now let's make a data frame with these two new columns and square them
so we don't have to deal with neagtives and keep their same names

df3 <- cbind(a5 <- a5*a5, a6<- a6*a6, m_error_allowance <- m_error_al
lowance*m_error_allowance, f_error_allowance <- f_error_allowance*f_e
rror_allowance)

df3

#okay, now we want to count how many predictions for
#male are less than m_error_allowance


# counting occurrences in a column range checking
successes <- a5[] <= m_error_allowance[]
```

```
#now to count how many predictions are statistically significant
table(successes)
```

A data.frame: 628 × 4

	male_life_expectancy_2010	female_life_expectancy_2010	female_life_expectancy
	<dbl>	<dbl>	<dbl>
8	8.30719944	1.033490e+01	14.36410
10	3.73627027	5.018327e+00	14.84946
13	0.04561761	7.557727e-01	14.95756
16	0.08629787	1.256350e+00	15.35072
24	13.59186119	6.370975e+00	14.22798
31	1.32953367	4.879955e-01	15.23341
41	4.83972985	3.489519e-01	15.50391
49	4.11774457	1.089214e+00	15.13599
52	2.61161859	3.102640e+00	15.15156
71	0.01529167	9.171699e-02	16.02401
76	0.15598872	2.068625e-02	16.02401
79	0.45330247	8.570889e-01	16.02401
81	0.99730972	1.178995e+00	16.02401
83	3.53137080	3.380457e+00	16.02401
88	0.75942386	3.689163e-02	16.02401
94	0.06479829	4.222940e-02	16.02401
95	2.27369413	1.917372e+00	16.02401
108	5.59753002	6.641826e+00	16.98264
117	0.02749721	1.493653e+00	14.89574
120	3.95339643	1.628386e+00	14.59622
127	2.17303873	4.800919e-01	15.44883
134	0.09528023	2.941101e-04	15.89218
137	2.29453852	3.899134e-01	15.38993
141	3.37672363	1.026996e-01	15.46456
151	0.41810584	6.379974e-01	15.20610
153	6.37461215	1.476607e+00	15.15156
160	0.39302079	2.209526e-01	15.37816
165	25.82893803	1.365814e+01	13.85328
166	0.23215633	2.186773e-03	15.45669
170	5.99303185	1.543743e+00	15.06604
⋮	⋮	⋮	⋮
3004	0.7209824	0.0005557507	15.36640
3009	2.2293071	2.3592094380	16.09614
3021	13.5730791	2.6528215028	14.81095
3022	0.2803019	0.5525421201	15.89617
3027	6.3037250	0.4658560108	15.37424

```
successes
FALSE  TRUE 
   17   610 
```

In [11]:
```
#Now that all of my messy work is done and I have my True and False val
ues
#I can give our first model a score of the number of trues out of our 6
28 observation test.

#from our table function, we see that we get 610 Trues, and 17 Falses

model_1_score <- 610/628
```

Set up for classification models

Linear regression requires the outcome variable to be numeric, while classification requires the outcome variable to be a factor and takes a little more cleaning. The following section removes variables that could bias the estimate, those with values that are hard to clean, and those that serve no real purpose. We turned our outcome variable, average life expectancy, into a dummy by having it return a value of 1 if the observation was in the 75th percentile of life expectancy or higher. This age was 78.8, so if the observation was less than 78.8 the model returned a 0.

In [12]:
```
#Let's remove male and female life expectancy and other pesky variables
#We need to remove them from the test and train data since we already s
plit them

train_df %<>% select(-"female_life_expectancy_2010")
test_df %<>% select(-"female_life_expectancy_2010")
train_df %<>% select(-"male_life_expectancy_2010")
test_df %<>% select(-"male_life_expectancy_2010")
train_df %<>% select(-"year")
test_df %<>% select(-"year")
train_df %<>% select(-"gisjoin")
test_df %<>% select(-"gisjoin")
train_df %<>% select(-"area_name")
test_df %<>% select(-"area_name")

#For classification, we need the outcome variable to be a factor
```

```
#First, create a dummy variable that returns a 1 if the observation has
a life expectancy of 78.8 or above
#It will return a 0 if not
#Then turn the new dummy variable into a factor

life_df$dummy_life_expectancy <- ifelse(life_df$avg_life_expectancy >
= 78.8, 1, 0)

life_df %<>% mutate_at(c("dummy_life_expectancy"),
                    as.factor)

#Let's do the above steps for the train and test data too, since we alr
eady split our data above

train_df$dummy_life_expectancy <- ifelse(train_df$avg_life_expectancy
>= 78.8, 1, 0)

test_df$dummy_life_expectancy <- ifelse(test_df$avg_life_expectancy >
= 78.8, 1, 0)

train_df %<>% mutate_at(c("dummy_life_expectancy"),
                    as.factor)

test_df %<>% mutate_at(c("dummy_life_expectancy"),
                    as.factor)

#Check to make sure the necessary change have taken place
#We hid this output so it doesn't print out long output
#skim(train_df)
#skim(test_df)
```

[13]:
```
# Define the recipe
#This is an important step that will be used in all of the following se
ctions

life_recipe =
    recipe(dummy_life_expectancy ~ ., data = train_df) %>%
    #Remove state and county as they are factor variables, and average
life expectancy because it will bias the results
    step_rm(contains("avg_life_expectancy")) %>%
    step_rm(contains("state")) %>%
    step_rm(contains("county")) %>%
    #Mode imputation for all nominal predictors
    step_modeimpute(all_predictors() & all_nominal()) %>%
    # Mean imputation for numeric predictors
    step_meanimpute(all_predictors() & all_numeric()) %>%
    # Standardize
    step_normalize(all_predictors() & all_numeric()) %>%
    # Remove low-variance, highly correlated, or linearly dependent pre
dictors
    step_nzv(all_predictors() & all_numeric()) %>%
    step_corr(all_predictors() & all_numeric()) %>%
    step_lincomb(all_predictors() & all_numeric())

life_recipe
#prep and juice our recipe

life_clean <- life_recipe %>% prep() %>% juice()
life_clean

#You won't want to use life_clean in some of the following sections
```

```
Data Recipe

Inputs:

      role #variables
   outcome          1
 predictor        151

Operations:

Delete terms contains("avg_life_expectancy")
Delete terms contains("state")
Delete terms contains("county")
Mode Imputation for all_predictors() & all_nominal()
Mean Imputation for all_predictors() & all_numeric()
Centering and scaling for all_predictors() & all_numeric()
Sparse, unbalanced variable filter on all_predictors() & all_numeric()
Correlation filter on all_predictors() & all_numeric()
Linear combination filter on all_predictors() & all_numeric()
```

A tibble: 2515 × 16

car_truck_or_van_carpooled_in_7_or_more_person_carpool	public_transportation_excluding
<dbl>	<dbl>
-0.26965208	-0.09066107
0.53592978	-0.08262626
0.02298786	-0.09265789
-0.28280444	-0.09170702
-0.27622826	-0.09346612
-0.28280444	-0.09408418

In [14]:
```
#Create 5 fold cross validation split
#Set a new seed that will hold values

set.seed(1291115)

life_cv = train_df %>% vfold_cv(v=5)
```

Method 2: Decision Trees

Decision trees are algorithms that split the data up by certain metrics. In this case, we will use accuracy to see how close our predictions got to the true values. Decision trees use cross validation for prediction.

Model score: 492/628

In [15]:
```
#Define the model for a decision tree with classification
#Tune the cost complexity so the model has more flexibility

life_tree = decision_tree(
    mode = "classification",
    cost_complexity = tune(),
    tree_depth = tune(),
    min_n = 5) %>% set_engine("rpart")
```

In [16]:
```
#Define the workflow

tree_workflow = workflow() %>%
    add_model(life_tree) %>% add_recipe(life_recipe)
```

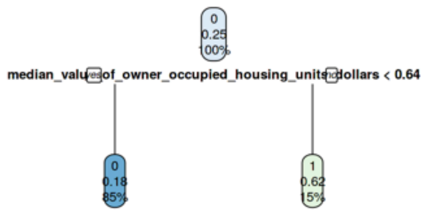
In [17]:
```
#Tune and run, using a simple model so it doesn't take hours to run
#We are using accuracy here to see how accurate our predictions were

tree_cv_life = tree_workflow %>% tune_grid(
    life_cv,
    grid = expand_grid(
        cost_complexity = seq(0, 0.2, by = 0.1),
        tree_depth = 1
    ),
    metrics = metric_set(accuracy)
)
```

In [18]:
```
#Finalize the workflow for plotting and predicting
best_life_flow =
    tree_workflow %>%
    finalize_workflow(select_best(tree_cv_life, metric="accuracy")) %
>%
    fit(data = life_df)
```

In [19]:
```
#Extract fitted model
best_life_tree = best_life_flow %>% pull_workflow_fit()
```

In [20]:
```
#Plot the decision tree
best_life_tree$fit %>% rpart.plot(roundint=F)
```



This classification tree split on the predictor, "Median Value of Owner Occupied Housing Units in Dollars". What this means is that if the median value of the owner occupied housing unit is less than 0.64 (this odd number comes from mean imputation and I interpret it to mean that this value is 64% higher than the mean), 85% of the observations will have a life expectancy less than 78.8 and 15% of the observations will have a life expectancy greater than or equal to 78.8.

In [21]:
```
#Predict best tree onto test data

y_hat = best_life_flow %>% predict(new_data = test_df)

#y_hat
```

In [22]:
```
#Create confusion matrix
cm_life_tree = conf_mat(
    data = tibble(
        y_hat = y_hat %>% unlist(),
        y = test_df$dummy_life_expectancy
    ),
    truth = y, estimate = y_hat
)

#View the matrix
cm_life_tree
```

```
          Truth
Prediction   0   1
         0 421  94
         1  42  71
```

Based on this confusion matrix, our decision tree accurately predicted life expectancy to be less than 78.8 a total of 421 of the 628 observations (approximately 67.0% accuracy). This method predicted life expectancy to be greater than or equal to 78.8 a total of 71 of the 628 times (approximately 11.3% accuracy). The decision tree had false negatives 94 of the 628 times (approximately 15.0%) and false positives 42 of the 628 times (approximately 6.7%). This means that the accuracy of this method was 492/628 or 78.3%.

Method 3: Elasticnet

For the elasticnet method, we will be using logistic regression and the accuracy metric to see how close our predicted values are to the actual values. Elasticnet uses cross validation for prediction.

Model score: 496/628

In [23]:
```
#Define model for elasticnet

model_life_en = logistic_reg(penalty = tune(), mixture = tune()) %>%
    set_engine("glmnet")
```

In [24]:

```r
#Define the workflow

workflow_life_en = workflow() %>%
    add_model(model_life_en) %>%
    add_recipe(life_recipe)
```

In [25]:

```r
#Cross validation with range of penalty and mixture
#Use accuracy as the metric just like in the decision tree

cv_life_en = workflow_life_en %>%
    tune_grid(
        life_cv,
        grid = grid_regular(mixture(), penalty(), levels = 3:3),
        metrics = metric_set(accuracy)
    )
```

In [26]:

```r
#Finalize the workflow to use in prediction

final_life_en =
    workflow_life_en %>%
    finalize_workflow(select_best(cv_life_en, 'accuracy'))
```

In [27]:

```r
#Fit the final model

life_ff_en = final_life_en %>% fit(data = train_df)
```

In [28]:

```r
#Predict onto test data

y_hat1 = life_ff_en %>% predict(new_data = test_df, type = "class")


#head(y_hat1)
```

In [29]:

```
#Confusion matrix for elasticnet

cm_life_en = conf_mat(
    data = tibble(
        y_hat1= y_hat1 %>% unlist(),
        y1 = test_df$dummy_life_expectancy %>% as.factor()
    ),
    truth = y1, estimate = y_hat1
)

cm_life_en
```

```
          Truth
Prediction   0   1
         0 443 112
         1  20  53
```

Based on this confusion matrix, our elasticnet logistic regression accurately predicted life expectancy to be less than 78.8 a total of 443 of the 628 observations (approximately 70.5% accuracy). The model predicted life expectancy to be greater than or equal to 78.8 a total of 53 of the 628 times (approximately 8.4% accuracy). The elasticnet logisitic regression had false negatives 112 of the 628 times (approximately 17.8%) and false positives 20 of the 628 times (approximately 3.2%). This means that the accuracy of this method was 496/628 or 78.9%.

Method 4: Bagging

Bagging is an ensemble classification method that uses bootstrapping to make predictions. It is different from the previous two methods in that it doesn't use cross validation in favor of bootstrapping.

Model score: 610/628

In [30]:

```r
#Bagging method

#Create the bagging model
life_bag_mod = bag_tree(
    mode = "classification",
    cost_complexity = 0,
    tree_depth = NULL,
    min_n = 2,
    class_cost = NULL
) %>% set_engine(
    engine = "rpart",
    times = 5
)
```

In [31]:

```r
#Define the workflow
life_bag_wf = workflow() %>%
    add_model(life_bag_mod) %>% add_recipe(life_recipe)
```

In [32]:

```r
#Fit the model to the training data

bag_life = life_bag_wf %>%
    fit(data = life_df)

bag_life
```

```
══ Workflow [trained] ══════════════════════════════════════════════════
══════════
Preprocessor: Recipe
Model: bag_tree()

── Preprocessor ────────────────────────────────────────────────────────
──────────
9 Recipe Steps
```

```
• step_rm()
• step_rm()
• step_rm()
• step_modeimpute()
• step_meanimpute()
• step_normalize()
• step_nzv()
• step_corr()
• step_lincomb()

── Model ─────────────────────────────────────────────────────────────
──────────

Bagged CART (classification with 5 members)

Variable importance scores include:

# A tibble: 16 x 4
   term                                                     value std.e
rror  used
   <chr>                                                    <dbl>     <
dbl> <int>
 1 median_household_income_in_past_12_months                 312.      2
0.6      5
 2 median_value_of_owner_occupied_housing_units_dollars      295.      1
5.4      5
 3 public_transportation_excluding_taxicab                   211.
7.48     5
 4 walked                                                    203.      1
1.3      5
 5 female_nursery_to_4th_grade                               193.      1
3.3      5
 6 other_means                                               191.      1
0.7      5
 7 bicylce                                                   186.      1
2.1      5
 8 public_transportation_excluding_taxicab_bus_or_trolley… 182.      1
3.8      5
 9 female_management_business_science_and_arts_occupation… 175.      1
3.5      5
10 male_management_business_science_and_arts_occupations_… 165.      1
5.8      5
11 female_natural_resources_construction_and_maintenance_… 162.
```

In [33]:
```
#Predict onto test data
y_hat2 = bag_life %>%
    predict(new_data = test_df, type = "class")
```

In [34]:
```
#Create the confusion matrix

cm_life_bag = conf_mat(
    data = tibble(
        y_hat2 = y_hat2 %>% unlist(),
        y2 = test_df$dummy_life_expectancy
    ),
    truth = y2, estimate = y_hat2
)

#View the confusion matrix for the bagging method

cm_life_bag
```

```
          Truth
Prediction   0   1
         0 456  11
         1   7 154
```

Based on this confusion matrix, our bagging method accurately predicted life expectancy to be less than 78.8 a total of 456 of the 628 observations (approximately 72.6% accuracy). This method predicted life expectancy to be greater than or equal to 78.8 a total of 154 of the 628 times (approximately 24.5% accuracy). The bagging method had false negatives 11 of the 628 times (approximately 1.8%) and false positives 7 of the 628 times (approximately 1.1%). This means that the accuracy of this method was 610/628 or 97.1%.

Method 5: Random Forest

This is an extra method that we did not have to do, but we used it to get practice using random forest. Random forest is an ensemble method similar to bagging, but uses randomly selected predictors and a bootstrapped sample. Just like bagging, it use bootstrapping instead of cross validation.

Model score: 513/628

In [35]:
```
#Set a small mtry value here, otherwise it will take a long time to run
#There are technically 16 predictors to use, but that takes much longer
#I used at least 10 observations to split since there are so many obser
vations in the data
#life_rf_grid helps us tune the random forest

life_rf_grid = expand_grid(
  mtry = 1:5,
  min_n = 1:10
)
```

In [36]:
```
#This function will cycle through the grid from above, then we define t
he random forest

rf_i = function(i) {
  #Define the random forest model, use only 50 trees so it runs faster
  life_rf_i = rand_forest(
    mode = "classification",
    mtry = life_rf_grid$mtry[i],
    trees = 50,
    min_n = life_rf_grid$min_n[i]
    #Use ranger as the engine for random forests
  ) %>% set_engine(engine = "ranger", splitrule = "gini")

  #Define workflow

  life_rf_wf_i =
    workflow() %>% add_model(life_rf_i) %>% add_recipe(life_recipe)

  #Fit the model to our dataset (don't use the clean dataset)

  life_rf_fit_i = life_rf_wf_i %>% fit(life_df)

  #Return DF w/ OOB error and hyperparameters - we want to know which h
as the lowest error rate

  tibble(
    mtry = life_rf_grid$mtry[i],
    min_n = life_rf_grid$min_n[i],
    error_oob = life_rf_fit_i$fit$fit$fit$prediction.error
  )
}
```

In [37]:

```
#Fit the RFs on the grid (since one of us is working on a Windows machi
ne, we need mc.cores=1)

life_rf_tuning = mclapply(
  X = 1:nrow(life_rf_grid),
  FUN = rf_i,
  mc.cores = 1
) %>% rbindlist()

#Arrange by smallest oob error - this is essential for printing out our
confusion matrix
#More can be done with oob errors, but for now, we just need it to dete
rmine accuracy

life_rf_tuning_arranged = life_rf_tuning %>% arrange(error_oob)

#life_rf_tuning_arranged

#Run the best model with only 50 trees so it doesn't take hours to run

life_best_rf_model = rand_forest(
  mode = "classification",
  mtry = life_rf_tuning_arranged[1,1],
  trees = 50,
  min_n = life_rf_tuning_arranged[1,2]
) %>% set_engine(engine = "ranger", splitrule = "gini")

#Define workflow

life_rf_wf =
  workflow() %>% add_model(life_best_rf_model) %>% add_recipe(life_re
cipe)

#Finalize workflow to use for predicting

best_life_flow =
  life_rf_wf %>%
  finalize_workflow(life_best_rf_model) %>%
  fit(data = train_df)

#Predict onto test data

y_hat3 = best_life_flow %>% predict(new_data = test_df, type = "class
")

#Create a confusion matrix
```

```
cm_life_rf = conf_mat(
  data = tibble(
    y_hat3 = y_hat3 %>% unlist(),
    y3 = test_df$dummy_life_expectancy
  ),
  truth = y3, estimate = y_hat3
)

#View the confusion matrix

cm_life_rf
```

```
          Truth
Prediction   0   1
         0 434  86
         1  29  79
```

Based on this confusion matrix, our random forest accurately predicted life expectancy to be less than 78.8 a total of 434 of the 628 observations (approximately 69.1% accuracy). This method predicted life expectancy to be greater than or equal to 78.8 a total of 79 of the 628 times (approximately 12.6% accuracy). The random forest had false negatives 86 of the 628 times (approximately 13.7%) and false positives 29 of the 628 times (approximately 4.6%). This means that the accuracy of this method was 513/628 or 81.7%.

In [38]:
```
#Compare the matrices; which model did the best?
cm_life_rf
cm_life_bag
cm_life_tree
cm_life_en
```

```
          Truth
Prediction   0   1
         0 434  86
         1  29  79

          Truth
Prediction   0   1
         0 456  11
         1   7 154

          Truth
Prediction   0   1
         0 421  94
         1  42  71
```

Based on the output of the confusion matrices, the most accurate method for this data is the bagging method (97.1% accuracy). The decision tree had the worst performance, with only 78.3% accuracy. It appears that ensemble methods do a better job with cross validation than simpler classification methods (this is not too surprising).