HarvardX Data Science: Capstone - Predicting Food Waste Diversion Potential Across 50 US States

s-kakad (GitHub)

2024-05-13

Introduction & Executive Summary

Food loss and waste, defined as food intended for human consumption that never gets consumed, represents an social, environmental, and economic burden on already stretched food systems. In 2015, as part of its Sustainable Development Goals (SDGs) framework, the United Nations (UN) introduced the target SDG 12.3 to halve per capita food waste and reduce food loss by 2030 globally (UN, 2015). To align with SDG 12.3, the United States (US) set a national goal to divert the amount of food waste landfilled or incinerated by 50% by 2030, equivalent to diverting 164 lbs (74 kg) food waste/capita (EPA, 2024). To achieve such a goal, a number of strategies, or 'solutions' across four broad policy categories (prevention, rescue, animal feed and recycling), are needed (e.g. education campaigns to prevent food waste, repurposing food waste into animal feed, home and industrial composting, etc.).

Given that states are responsible for implementing this federal goal through state policies and that knowledge on the effectiveness of these policies remains limited, this Capstone project seeks to develop several machine learning algorithms to predict the food waste diversion potential by solution and state, for the year 2022. Earlier diversion potential values (from 2021), population data, and policy scoring metrics were sourced from a publicly available database and used as predictors.

The following steps, typical of data science projects in R, were adopted:

(1) Setting up the working environment by downloading essential packages;

(2) Downloading the dataset(s) of interest and perform preliminary data wrangling;

(3) Performing some preliminary data exploration and visualization;

(4) Building several machine learning algorithms;

(5) Assessing their predictive power to select the best algorithm, and;

(6) Evaluating the best model on the test set

> Note: This project uses publicly available data from the US non-profit organization ReFED (www.refed.com) - for ease and speed of ease, smaller pre-processed datasets are available on GitHub.

Methodology

First, the directory of choice was set as working directory.

```r
setwd("~/projects/edX-CYOP")
# Note this absolute path may be change to reflect individual users' folder system
```

The following R packages were first uploaded (and installed if not already available):

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(ggplot2)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: knitr
```

```r
if(!require(tinytex)) install.packages("tinytex", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tinytex
```

```r
library(tidyverse)
library(caret)
library(ggplot2)
library(knitr)
library(tinytex)
```

Downloading Datasets

The datasets listing the food waste diversion potential values for each solution and state, as well as population and/or policy metrics were downloaded from GitHub (see note above). Note there are two separate files for diversion potential data, one for 2021 and one for 2022, so we will merge them together.

```
fw_df_solutions_2021 <- read.csv("edX_ReFED_2021.csv")
fw_df_solutions_2022 <- read.csv("edX_ReFED_2022.csv")
fw_df_policies <- read.csv("edX_ReFED_policies.csv")

# Merge 2021 and 2022 dataframes into single 'mother' solutions dataframe
fw_df_solutions <- merge(x = fw_df_solutions_2021, y = fw_df_solutions_2022[, c("state", "solution_name
      by = c("state", "solution_name"), all.x = TRUE)
remove(fw_df_solutions_2021, fw_df_solutions_2022)

# Check structure
str(fw_df_solutions)
```

```
## 'data.frame':    697 obs. of  6 variables:
##  $ state                              : chr  "Alabama" "Alabama" "Alabama" "Alabama" ...
##  $ solution_name                      : chr  "Centralized Anaerobic Digestion" "Centralized Compo
##  $ solution_policy_category           : chr  "Recycling" "Recycling" "Recycling" "Recycling" ...
##  $ solution_intervention_category     : chr  "Recycle Anything Remaining" "Recycle Anything Remai
##  $ solution_diversion_potential_tons_2021: int  91193 266978 86575 36149 51699 4839 14360 4568 12169
##  $ solution_diversion_potential_tons_2022: int  78040 228423 70797 33316 50919 2436 9358 2489 6809 8
```

```
str(fw_df_policies)
```

```
## 'data.frame':    200 obs. of  7 variables:
##  $ state                   : chr  "AL" "AL" "AL" "AL" ...
##  $ solution_policy_category: chr  "Prevention" "Rescue" "Animal Feed" "Recycling" ...
##  $ population_2021         : int  5049846 5049846 5049846 5049846 734182 734182 734182 734182 726487
##  $ population_2022         : int  5074296 5074296 5074296 5074296 733583 733583 733583 733583 735919
##  $ total_fw_tons_2021      : int  1100735 1100735 1100735 1100735 135146 135146 135146 135146 885568
##  $ total_fw_tons_2022      : int  918036 918036 918036 918036 119604 119604 119604 119604 628779 628
##  $ policy_score            : chr  "Negative" "Weak" "Negative" "None" ...
```

The structure of the solutions dataframe shows that each solution is associated with a distinct diversion potential for each state (i.e. how much food waste a given policy has the potential to divert in a given state, in US short tons, for the years 2021 and 2022), a policy category (e.g. education campaigns fall under prevention, while composting falls under recycling) and a corresponding intervention type (e.g. education campaigns specifically aim to reshape consumer environments).

The policies dataframe provides additional variables, including 2021 and 2022 population census data and policy scores for each policy category and state.

Processing

In the next steps, we will perform some data wrangling to convert variables to their appropriate class and ensure both dataframes are compatible before merging them.

```
# Convert dataframes to appropriate class
fw_df_solutions$solution_policy_category <- as.factor(fw_df_solutions$solution_policy_category)
fw_df_solutions$solution_intervention_category <- as.factor(fw_df_solutions$solution_intervention_categ
fw_df_solutions$solution_name <- as.factor(fw_df_solutions$solution_name)
fw_df_solutions$state <- as.factor(fw_df_solutions$state)
fw_df_solutions$solution_diversion_potential_tons_2021 <- as.numeric(fw_df_solutions$solution_diversion
fw_df_solutions$solution_diversion_potential_tons_2022 <- as.numeric(fw_df_solutions$solution_diversion

fw_df_policies$state <- as.factor(fw_df_policies$state)
fw_df_policies$solution_policy_category <- as.factor(fw_df_policies$solution_policy_category)
fw_df_policies$population_2021 <- as.numeric(fw_df_policies$population_2021)
fw_df_policies$population_2022 <- as.numeric(fw_df_policies$population_2022)
fw_df_policies$total_fw_tons_2021 <- as.numeric(fw_df_policies$total_fw_tons_2021)
fw_df_policies$total_fw_tons_2022 <- as.numeric(fw_df_policies$total_fw_tons_2022)
fw_df_policies$policy_score <- as.factor(fw_df_policies$policy_score)

# Reassign policy type for solution Livestock Feed from Recycling to Animal Feed
# Note: this is because Livestock Feed is considered as a separate policy category in the policy datafr
fw_df_solutions <- fw_df_solutions %>%
  mutate(solution_policy_category = ifelse(solution_name == "Livestock Feed", "Animal Feed", as.characte
fw_df_solutions$solution_policy_category <- as.factor(fw_df_solutions$solution_policy_category)

# Reorder factor order for policy categories
fw_df_solutions$solution_policy_category <- factor(fw_df_solutions$solution_policy_category, levels = c
order(levels(fw_df_solutions$solution_policy_category))
```

```
## [1] 3 1 4 2
```

```
fw_df_policies$solution_policy_category <- factor(fw_df_policies$solution_policy_category, levels = c("
```

Earlier, we saw that states are named differently across dataframes, where the the 'solutions' dataframe spells out state names, while the 'policy' dataframes adopts the two-letter state abbreviation. After ensuring that their respective state order follows the same order as the built-in state.name and state.abb datasets, we will adopt the full name spelling across both dataframes.

```
sum(levels(fw_df_solutions$state) == state.name) # TRUE for all states
```

```
## [1] 50
```

```
sum(levels(fw_df_policies$state) == state.abb) # TRUE only for 26 states
```

```
## [1] 26
```

```
# Reorder the two-letter state abbreviations according to the built-in state.abb data set
fw_df_policies$state <- factor(fw_df_policies$state, levels = state.abb)
sum(levels(fw_df_policies$state) == state.abb) # now TRUE for all states
```

```
## [1] 50
```

```r
# Rename the factors using the state.abb data set (same order as state.abb)
levels(fw_df_solutions$state) <- state.abb

# Check that levels for both "state" and "solution_policy_category" align
sum(levels(fw_df_solutions$state) == levels(fw_df_policies$state))
```

```
## [1] 50
```

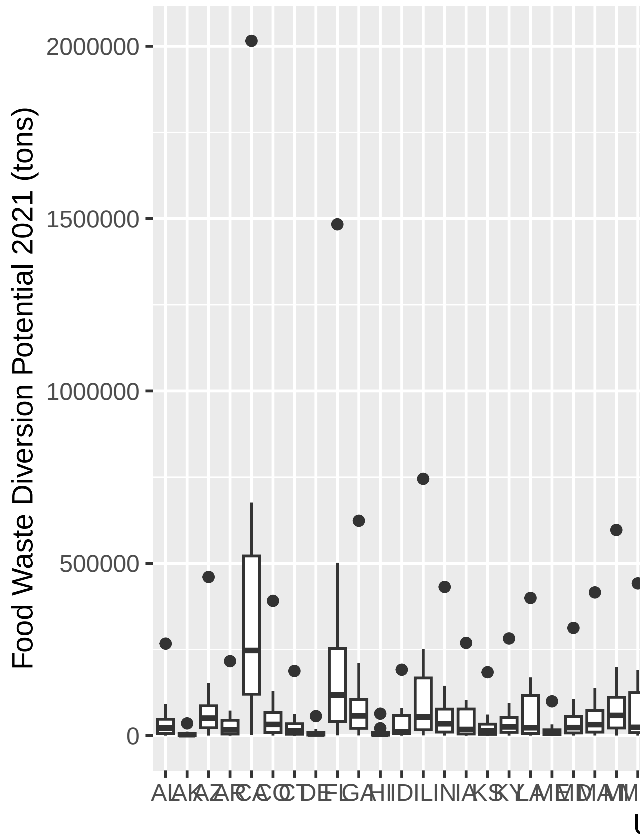
```r
levels(fw_df_solutions$solution_policy_category) == levels(fw_df_policies$solution_policy_category)
```

```
## [1] TRUE TRUE TRUE TRUE
```

```r
# Combine data frames
combined_df <- left_join(fw_df_solutions, fw_df_policies, by = c("state", "solution_policy_category"))
remove(fw_df_policies, fw_df_solutions)
```

Data Exploration & Visualization

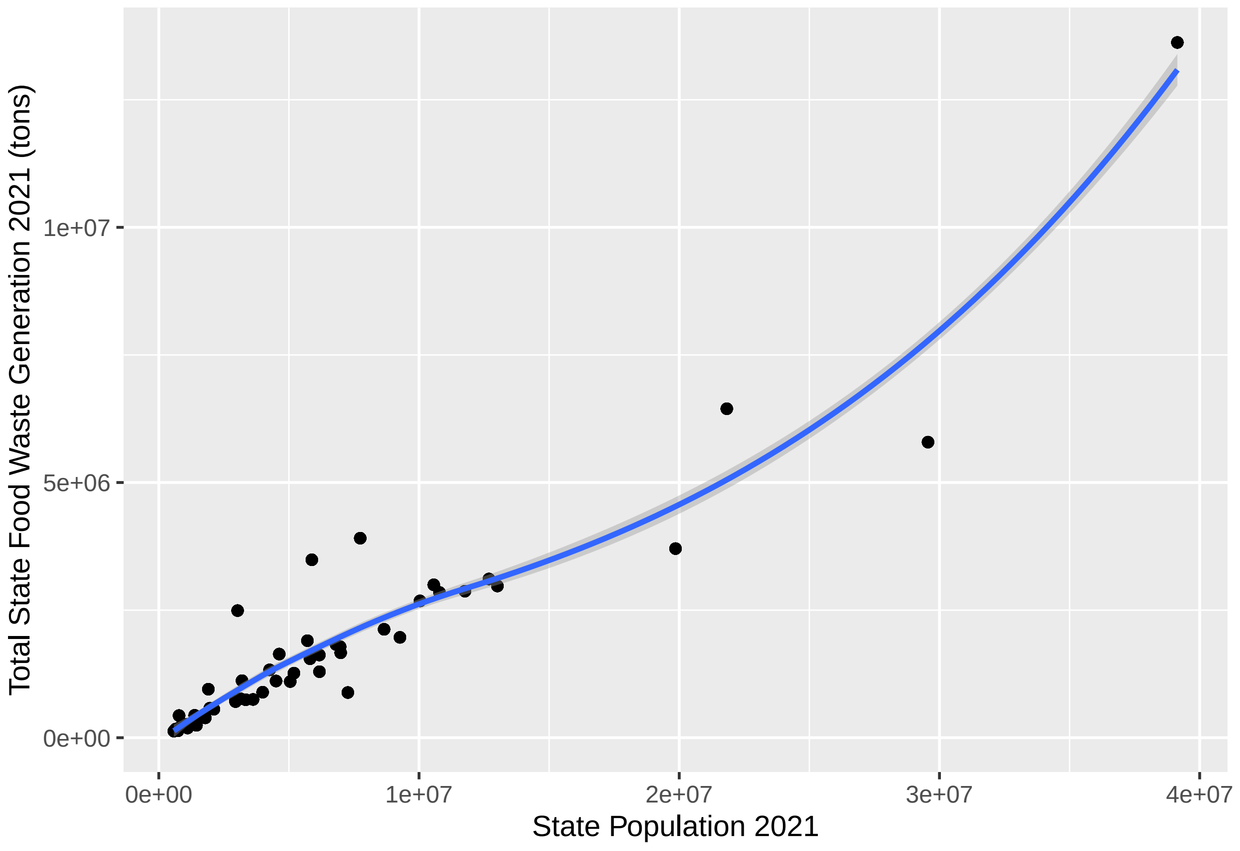
Plotting the food waste diversion potential for 2021 across the 50 states shows these is a high



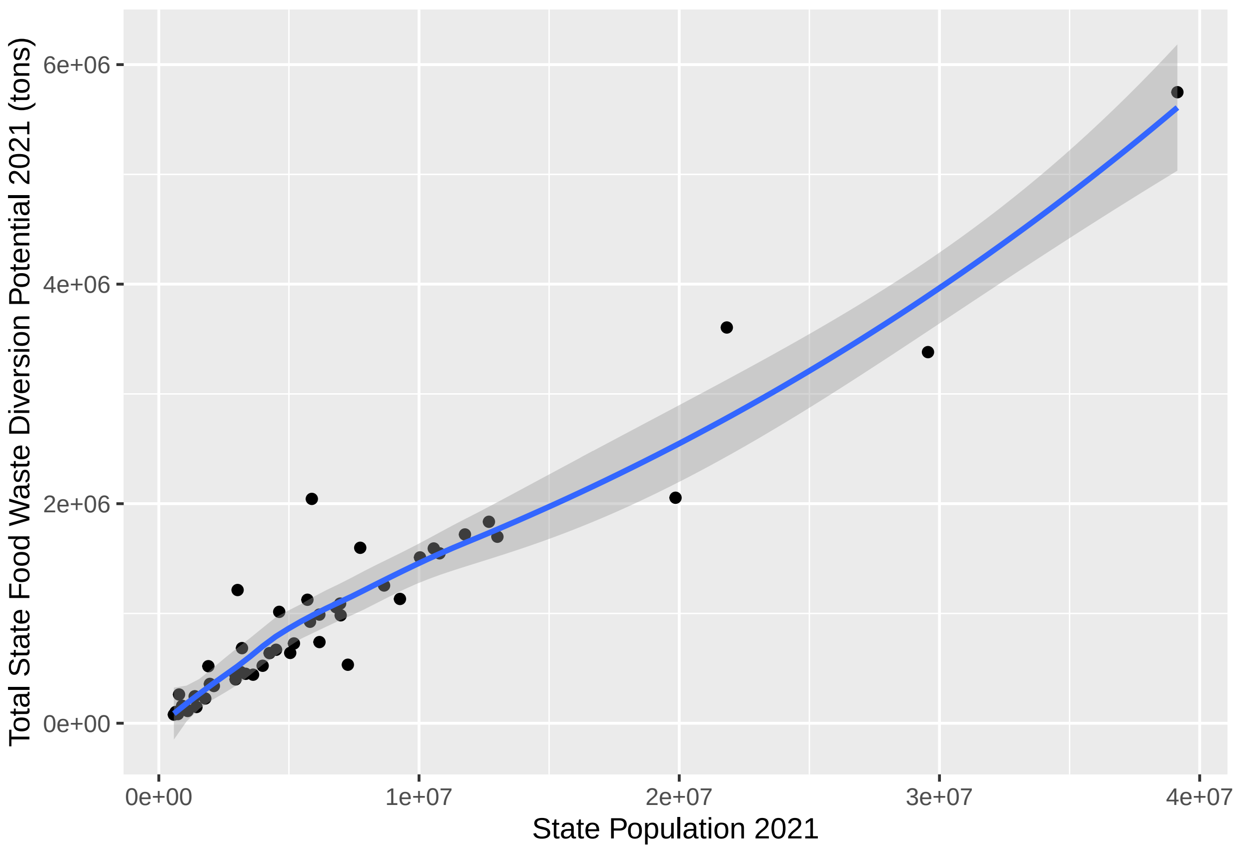
variability in the range of diversion potential achievable.

One reason for this is that the bigger the population, the more food waste is generated, and thus the bigger the food waste diversion potential, as visualized in the plots below. The trends are similar, since the food waste diversion potential is derived from the food waste generation for a given state and solution (for an in-depth explanation, please consult ReFED's website on www.refed.org).

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

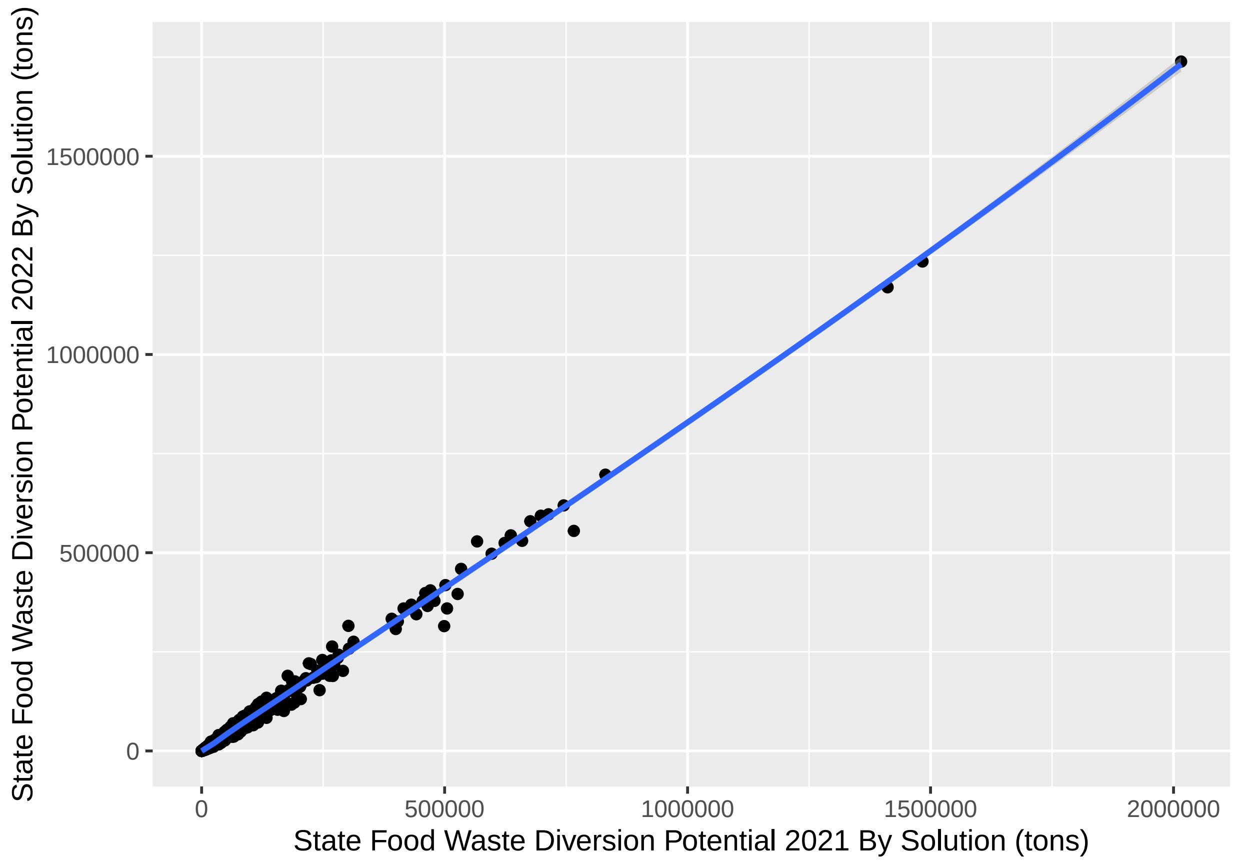


```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



In addition, food waste diversion potential data from 2021 and 2022 are highly correlated, with cor(2021, 2022 = 0.9967):

```
## ‘geom_smooth()‘ using method = ’loess’ and formula = ’y ~ x’
```



Building Train & Test Sets

Because datapoints are limited, the combined dataset is split 80/20 to ensure enough variability in the dataset is captured in the machine learning step, while leaving enough data to assess the performance of the algorithms across a range of solutions and states.

```
# Create test and train set
set.seed(1, sample.kind = "Rounding") # using R 3.6 or later (R 4.1.1)
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_index <- createDataPartition(combined_df$total_fw_tons_2022, times = 1, p = 0.2, list = FALSE)
test_temp <- combined_df[test_index, ] # this is the temporary test set
train_temp <- combined_df[-test_index, ] # this is the temporary train set

# Ensure combinations of state x policy_category in the test set are also in the train set (if not, mov
test_set <- test_temp %>%
  semi_join(train_temp, by = "state") %>%
  semi_join(train_temp, by = "solution_policy_category") # this is now the updated test set

removed <- anti_join(test_temp, test_set)
```

```
## Joining with 'by = join_by(state, solution_name, solution_policy_category,
## solution_intervention_category, solution_diversion_potential_tons_2021,
## solution_diversion_potential_tons_2022, population_2021, population_2022,
## total_fw_tons_2021, total_fw_tons_2022, policy_score)‘
```

```
train_set <- rbind(train_temp, removed) # this is now the updated train set
remove(test_index, test_temp, train_temp, removed)

options(digits = 4)
```

Results

Here, given that food waste forecasting remains limited, a prediction 20% above or below the real value was considered as acceptable (i.e. a ‘hit’ for a given algorithm).

Just The Average

First, a static approach waste adopted, whereby the 2022 average across all solutions and states was used, as a baseline.

```
solutions_avg <- mean(combined_df$solution_diversion_potential_tons_2022)
avg_acc <- mean(solutions_avg >= 0.80*test_set$solution_diversion_potential_tons_2022 & solutions_avg <
avg_acc
```

```
## [1] 0.0922
```

Linear Regression Models

Then, a simple linear regression model was considered, using 2021 data as sole predictor, followed by a more comprehensive model building on all variables.

```
train_glm <- train(solution_diversion_potential_tons_2022 ~ solution_diversion_potential_tons_2021, met
glm_preds <- predict(train_glm, test_set)
glm_acc <- mean(glm_preds >= 0.80*test_set$solution_diversion_potential_tons_2022 & glm_preds <= 1.20*t
glm_acc
```

```
## [1] 0.4894
```

```
train_glm_all <- train(solution_diversion_potential_tons_2022 ~ ., method = "glm", data = train_set)
glm_all_preds <- predict(train_glm_all, test_set)
glm_all_acc <- mean(glm_all_preds >= 0.80*test_set$solution_diversion_potential_tons_2022 & glm_all_pre
glm_all_acc
```

```
## [1] 0.5177
```

More Developed Models: kNN, Cross-validation, Classification Tree, Random Forest & Ensemble

In a third step, more sophisticated models were built, including kNN, cross-validation, classification tree, random forest, and an ensemble model based on the best performing model (classification tree and random forest).

```
# kNN model
train_knn <- train(solution_diversion_potential_tons_2022 ~ .,
                   method = "knn",
                   data = train_set,
                   tuneGrid = data.frame(k = seq(1, 50, 1)))
train_knn$bestTune
```

```
##   k
## 1 1
```

```
knn_preds <- predict(train_knn, test_set)
knn_acc <- mean(knn_preds >= 0.80*test_set$solution_diversion_potential_tons_2022 & knn_preds <= 1.20*te
knn_acc
```

```
## [1] 0.3121
```

```
# Cross-validation model
train_knn_cv <- train(solution_diversion_potential_tons_2022 ~ .,
                      method = "knn",
                      data = train_set,
                      tuneGrid = data.frame(k = seq(1, 50, 1)),
                      trControl = trainControl(method = "cv", number = 10, p = 0.1))
train_knn_cv$bestTune
```

```
##   k
## 1 1
```

```
knn_cv_preds <- predict(train_knn_cv, test_set)
knn_cv_acc <- mean(knn_cv_preds >= 0.80*test_set$solution_diversion_potential_tons_2022 & knn_cv_preds 
knn_cv_acc
```

```
## [1] 0.3121
```

```
# Classification tree model
train_rpart <- train(solution_diversion_potential_tons_2022 ~ .,
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0, 0.05, 0.002)),
                     data = train_set)
train_rpart$bestTune
```

```
##   cp
## 1  0
```

```r
rpart_preds <- predict(train_rpart, test_set)
rpart_acc <- mean(rpart_preds >= 0.80*test_set$solution_diversion_potential_tons_2022 & rpart_preds <=
rpart_acc
```

```
## [1] 0.6596
```

```r
# Random forest model
train_rf <- train(solution_diversion_potential_tons_2022 ~ .,
                  data = train_set,
                  method = "rf",
                  ntree = 150,
                  tuneGrid = data.frame(mtry = seq(1, 100, 5)))
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
train_rf$bestTune
```

```
##    mtry
## 14   66
```

```
rf_preds <- predict(train_rf, test_set)
rf_acc <- mean(rf_preds >= 0.80*test_set$solution_diversion_potential_tons_2022 & rf_preds <= 1.20*test_
rf_acc
```

```
## [1] 0.844
```

```
# Ensemble model (attempt with rpart and rf, the best performing models)
ensemble_preds <- (rpart_preds + rf_preds)/2
ensemble_acc <- mean(ensemble_preds >= 0.80*test_set$solution_diversion_potential_tons_2022 & ensemble_
ensemble_acc
```

```
## [1] 0.7092
```

Conclusion

The table below provides a summary of all models developed:

```
# Collate accuracies and present in table
accuracies <- c(avg_acc, glm_acc, glm_all_acc, knn_acc, knn_cv_acc, rpart_acc, rf_acc, ensemble_acc)
models <- c("mean", "glm-single", "glm-all", "kNN", "kNN cross-validation", "classification tree", "ranc

final_table <- data.frame(models, accuracies) %>%
  rename("model" = models, "accuracy (+/- 20%)" = accuracies)
final_table
```

```
##                                         model accuracy (+/- 20%)
## 1                                        mean             0.0922
## 2                                  glm-single             0.4894
## 3                                     glm-all             0.5177
## 4                                         kNN             0.3121
## 5                        kNN cross-validation             0.3121
## 6                         classification tree             0.6596
## 7                               random forest             0.8440
## 8 ensemble (classification tree + random forest)          0.7092
```

Despite a high correlation between 2021 and 2022 data, the simplest linear model was only able to achieve around 50% accuracy. The random forest algorithm yielded the highest predictive power (considering an accuracy of +/- 20%). This makes intuitive sense, given that a given solution will yield a highly variable diversion potential, based on its associated state and corresponding population data and policy metrics. Thus, a clustering approach (kNN) may have failed to capture patterns in the data that may be easier to model using a tree, or classification, approach (e.g. if a given state has a population higher than X, then food waste diversion will likely be higher than Y (say, the average), etc.).

It is important to note that accuracy was defined as a value 20% below or above average, in part to reflect variability in real-life trends. This also enabled the development of classification models on a continuous variable. Nonetheless, it is a somewhat subjective measure of accuracy and represents an non-trivial limitation. Lastly, increasing the number of years for which food waste generation and/or food waste diversion potential were reported represents an area for future research.

References

ReFED (2024). ReFED Insights Engine. Available from: https://insights.refed.org/ [Accessed May 12, 2024]. United Nations General Assembly (2015). Transforming our world: the 2030

Agenda for Sustainable Development. A/RES/70/1, 21 October 2015. United Nations, New York, USA. United States Environmental Protection Agency (EPA) (2024). United States 2030 Food Loss and Waste Reduction Goal. Available from: https://www.epa.gov/sustainable-management-food/united-states-2030-food-loss-and-waste-reduction-goal [Accessed May 12, 2024].

All files need to perform this analysis are available on the following GitHub page: https://github.com/s-kakad/edX-CYOP/tree/main