DSApps 2023 @ TAU: Final Project

Prediction

Sofia Prahia- 314710567, Noa Shaya – 318455961

2023-08-20

Install and load the required libraries

```
if (!require("tidyverse")) {install.packages("tidyverse") }
library(tidyverse)
if (!require("ggplot2")) {install.packages("ggplot2") }
library(ggplot2)
if (!require("stringr")) {install.packages("stringr") }
library(stringr)
if (!require("tidytext")) {install.packages("tidytext") }
library(tidytext)
if (!require("purrr")) {install.packages("purrr") }
library(purrr)
if (!require("dplyr")) {install.packages("dplyr") }
library(dplyr)
if (!require("tidyr")) {install.packages("tidyr") }
library(tidyr)
if (!require("tidymodels")) {install.packages("tidymodels") }
library(tidymodels)
if (!require("naniar")) {install.packages("naniar") }
library(naniar)
if (!require("caret")) {install.packages("caret") }
library(caret)
if (!require("keras")) {install.packages("keras") }
library(keras)
if (!require("tensorflow")) {install.packages("tensorflow") }
library(tensorflow)
if (!require("reticulate")) {install.packages("reticulate") }
library(reticulate)
```

Load the data:

```
food_train <- read_csv("data/food_train.csv")
food_test <- read_csv("data/food_test.csv")
nutrients <- read_csv("data/nutrients.csv")
food_nutrients <- read_csv("data/food_nutrients.csv")
```

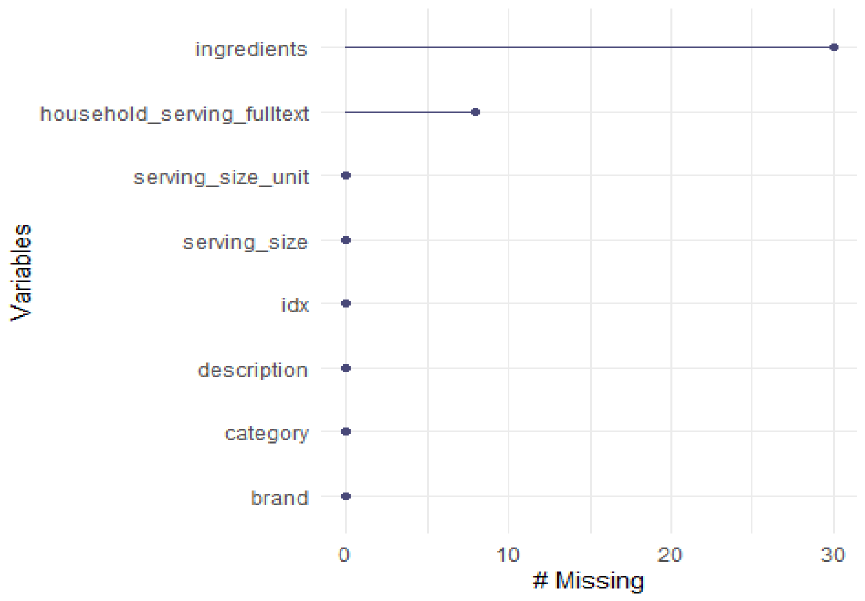
#Split the train data to validation, tuning and engineering sets

```
tidymodels_prefer()
#split the data to validation (10%) and train (90%) sets
set.seed(42)
food_split <- initial_split(food_train, strata=category, prop = 0.1 )
```

```r
validation <- training(food_split)
train <- testing(food_split)
#take 25% of the train data for checking the tuning
set.seed(42)
tuning_split <- initial_split(train, prop = (1/4))
tuning <- training(tuning_split)
engineering <- testing(tuning_split) # use for feature engineering
```

Dealing with missing values:

```r
gg_miss_var(engineering)
```



```r
sum(is.na(food_nutrients)) #no NA

## [1] 0

sum(is.na(nutrients)) #no NA

## [1] 0

#The dataset contains a minimal number of NA values.
100 * colMeans(is.na(engineering))

##                        idx                      brand
##                 0.00000000                 0.00000000
##                description                ingredients
##                 0.00000000                 0.13996454
##               serving_size          serving_size_unit
##                 0.00000000                 0.00000000
## household_serving_fulltext                   category
##                 0.03732388                 0.00000000

#Dealing with the NA in the data, by replacing them with "Unknown"
category.
NA_fix <- function(data) {
  data <- data %>%
    mutate(ingredients = ifelse(is.na(ingredients), "Unknown",
ingredients),
           household_serving_fulltext =
ifelse(is.na(household_serving_fulltext), "Unknown",
household_serving_fulltext))
  return(data)
}
engineering <- NA_fix(engineering)
```

In our data analysis, we found that there are very few instances where data is missing. These missing values are only present in the ingredients and household_serving_fulltext columns. Given the minor number of missing values, it's unlikely that their absence would significantly influence our analysis. Therefore, we've chosen to handle these missing values by assigning them an "Unknown" category with the function "NA_fix". This decision is based on the understanding that the low number of missing values makes it unnecessary to extensively explore advanced methods of filling in missing data, such as using resampling techniques.

FEATURE ENGINEERING

We will begin by addressing each of the columns:

BRAND Column:

Cleaning the brand column

```
#clean and standardizes brand names by removing special characters,
replacing spaces with underscores, changing ampersands to "and," and
converting names to lowercase.
brand_cleaning <- function(data) {
  data <- data %>%
    mutate(brand = str_replace_all(brand, "[^A-Za-z0-9& ]", ""),
           brand = str_replace_all(brand, "\\s", "_"),
           brand = str_replace_all(brand, "&", " and "),
           brand = str_to_lower(brand))
  return(data)
}

engineering <- brand_cleaning(engineering)
```

We are now going to add for each database 6 additional columns, each corresponding to a specific category. These columns will receive a value of 1 if the snack's brand is exclusively associated with the corresponding category. We are going to create a vector containing all brands assigned to *only one* category and related to more than 10 different snacks. #Subsequently, we will examine each item to determine if it is associated with any of those brands. If such a connection exists, the respective column will be assigned a value of 1, while the remaining columns will retain a value of 0:

```
brand_one_category <- engineering %>%
  group_by(brand) %>%
  mutate(num_snacks = n(), num_categories = n_distinct(category)) %>%
  select(brand, category, num_snacks, num_categories) %>%
  filter(num_categories==1,num_snacks>=10) %>%
  arrange(desc(num_snacks)) %>%
  distinct() %>%
  select(brand, category)

add_category <- function(data, brand_one_category) {
  data %>%
    mutate(popcorn_peanuts_seeds_related_snacks_brand = ifelse(data$brand
%in% brand_one_category$brand[brand_one_category$category ==
"popcorn_peanuts_seeds_related_snacks"], 1, 0)) %>%
    mutate(candy_brand = ifelse(data$brand %in%
brand_one_category$brand[brand_one_category$category == "candy"], 1, 0))
%>%
    mutate(chocolate_brand = ifelse(data$brand %in%
brand_one_category$brand[brand_one_category$category == "chocolate"], 1,
0)) %>%
      mutate(cookies_biscuits_brand = ifelse(data$brand %in%
brand_one_category$brand[brand_one_category$category ==
"cookies_biscuits"], 1, 0)) %>%
      mutate(cakes_cupcakes_snack_cakes_brand = ifelse(data$brand %in%
brand_one_category$brand[brand_one_category$category ==
"cakes_cupcakes_snack_cakes"], 1, 0)) %>%
      mutate(chips_pretzels_snacks_brand = ifelse(data$brand %in%
brand_one_category$brand[brand_one_category$category ==
"chips_pretzels_snacks"], 1, 0))
}
#Add a column for each category brands that are related to one category.
To sum of, there will be 6 additional columns
engineering <- add_category(engineering,brand_one_category)
```

INGREDIENTS, HOUSEHOLD SERVING FULLTEXT,and DESCRIPTION Columns:

We will clean and identify the top appeared keywords for each of these three columns The cleaning is based on correcting misspelled words, replacing special characters with underscores, handling variations like singular/plural forms, and eliminating non-indicative words. This enhanced standardization and clarity in the column’s content.

```
#Find the most famous ingredients, and enter their name to a vector called
keywords_ingredients
nutrient_names <- nutrients %>% select(name)
keywords_ingredients <- engineering %>%
  mutate(ingredients = strsplit(ingredients, ", ")) %>%
  unnest(ingredients) %>%
  mutate(ingredients = str_replace(ingredients, "\\s*\\(.*", "")) %>%
  mutate(ingredients = str_replace(ingredients, "\\s*\\).*", "")) %>%
  mutate(ingredients = str_replace_all(ingredients, " ", "_")) %>%
  mutate(ingredients = ifelse(str_detect(ingredients, "salt"), "salt",
ingredients)) %>%
  mutate(ingredients = ifelse(str_detect(ingredients, "corn syrup"), "corn
syrup", ingredients)) %>%
  mutate(ingredients = ifelse(ingredients ==
"natural_and_artificial_flavors", c("natural_flavor",
"artificial_flavor"), ingredients)) %>%
  mutate(ingredients = ifelse(ingredients == "cornstrach", "corn_strach",
ingredients)) %>%
  mutate(ingredients = ifelse(ingredients == "modified_corn_starch",
"corn_strach", ingredients)) %>%
  group_by(ingredients) %>%
  filter(!ingredients %in% nutrient_names) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  head(50)

keywords_ingredients <- keywords_ingredients$ingredients
```

Household serving fulltext - alter some names,find the most popular ones, and add them to a vector called keywords_household

```
household_serving_fulltext_cleaning <- function (data){
data <- data %>%
  mutate(household_serving_fulltext =
str_to_lower(household_serving_fulltext)) %>%
  mutate(household_serving_fulltext = gsub("[0-9.]", "",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "cookie"), "cookie",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "ckooies"), "cookie",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "cracker"), "cracker",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "waffle"), "waffle",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "wafel"), "waffle",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "cake"),
"cake",household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "pretzel"), "pretzel",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "petzel"), "pretzel",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "prtzl"), "pretzel",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "candy"), "candy",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "candies"), "candy",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "wafer"), "wafer",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "biscuit"), "biscuit",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "nut"),"nut",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "chocolate"), "chocolate",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "chips"), "chips",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "crisps"), "chips",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "brownie"), "brownie",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "bar"), "bar",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "danish"), "danish",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "beans"), "beans",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "cup"), "cup",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "onz"), "onz",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "piece"), "piece",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "of pie"), "pie",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "packages"), "package",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "squares"), "square",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "bunnies"), "bunny",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "slices"), "slice",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "eggs"), "egg",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "pops"), "pop",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "truffles"), "truffle",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(str_detect(household_serving_fulltext, "macaroons"), "macaroon",
household_serving_fulltext)) %>%
  mutate(household_serving_fulltext =
ifelse(substr(household_serving_fulltext, 1, 1) == " ",
substring(household_serving_fulltext, 2), household_serving_fulltext))
}

engineering <- household_serving_fulltext_cleaning (engineering)
size_and_structure_keywords_household <- c("onz", "pieces", "cup", "grm",
"piece", "package" , "pcs", "tbsp","slice","pouch", "bag","pack",
"container", "grahams", "pc", "pc's", "pkg","packs", "tsp", "serving",
"kit", "box" , "sections", "hearts", "ounce","stick", "blocks", "packet",
"loaf", "cane", "bunny", "chicks", "rolls", "tube", "sticks", "snack
pack","pouches", "dots", "bites", "pcs | about", "balls", "roll")

keywords_household <- engineering %>%
  group_by(household_serving_fulltext) %>%
  filter(!household_serving_fulltext %in%
size_and_structure_keywords_household)%>%
  mutate(count= n()) %>%
  select(household_serving_fulltext, count) %>%
  arrange(desc(count)) %>%
  distinct() %>%
  head(22)
```

cleaning the description column, and finding the most popular keywords and adding them to a vector called keywords_description

```
descreption_cleaning <- function(data){
data <- data %>%
  mutate(description = str_replace_all(description, "[^A-Za-z0-9 ]", ""))
%>%
  mutate(description = str_replace_all(description, "( |&|and)+", "_"))
%>%
  mutate(description = ifelse(str_detect(description, "salted"), "salt",
description)) %>%
  mutate(description = ifelse(str_detect(description, "chocolates"),
"chocolate", description)) %>%
  mutate(description = ifelse(str_detect(description, "bake"), "bake",
description)) %>%
  mutate(description = ifelse(str_detect(description, "peanut"), "peanut",
description)) %>%
  mutate(description = ifelse(str_detect(description, "creme"), "cream",
description)) %>%
  mutate(description = ifelse(str_detect(description, "creamy"),"cream",
description)) %>%
  mutate(description = ifelse(str_detect(description, "gummi"), "gummy",
description)) %>%
  mutate(description = ifelse(str_detect(description, "mix"), "mix",
description)) %>%
  mutate(description = ifelse(str_detect(description, "nut"), "nut",
description)) %>%
  mutate(description = ifelse(str_detect(description, "cake"), "cake",
description)) %>%
  mutate(description = ifelse(str_detect(description, "caramel"),
"caramel", description)) %>%
  mutate(description = ifelse(str_detect(description, "almond"), "almond",
description)) %>%
  mutate(description = ifelse(str_detect(description, "pretzel"),
"pretzel", description))
}

engineering <- descreption_cleaning(engineering)
keywords_description <- engineering %>%
  mutate(description = strsplit(description, "_")) %>%
  unnest(description) %>%
  group_by(description) %>%
  mutate(count = n()) %>%
  select(description, count) %>%
  filter(! description %in% c("c", "y", "with", "mini", "trail","cashews",
"s", "original", "kettle", "ies", "assorted", "gourmet", "whole", "soft",
"the", "premium", "wich", "filled","bites","in","pieces", "foods",
"style","raw","of", "covered", "hearts", "free","halves",
"shortbread","classic", "sticks", "hard", "food","blue", "double",
"market", "graham", "slices", "hot", "black" ))%>%
  arrange(desc(count)) %>%
  distinct() %>%
  head(90)
```

Create a column for each keyword in keywords, and detect if the keywords exists in the three columns - "description", "ingredients", "household_serving_fulltext". If so, put the number of shows in the relevant column

```
#unit all the keywords to one vector
keywords <-
unique(c(keywords_description$description,keywords_household$household_ser
ving_fulltext, keywords_ingredients))

#function that creates a new column in the data frame representing keyword
detections
create_keyword_detection_column <- function(data, column_name, keyword) {
  data %>%
    mutate(!!keyword := rowSums(across(c("description", "ingredients",
"household_serving_fulltext"), ~ str_detect(.x, keyword))))
}
```

```r
#function to add multiple keyword detection columns to the data frame
add_keyword_detection_columns <- function(data, keywords) {
  keywords %>%
    reduce(~ create_keyword_detection_column(.x, "description", .y), .init
= data)
}
engineering <- add_keyword_detection_columns(engineering,keywords)
```

SERVING SIZE column:

Converting the serving sizes to grams, but only for entries that are currently in milliliters

```r
change_to_grams <- function(data) {data %>% mutate(serving_size =
ifelse(serving_size_unit=="ml",serving_size*0.789,serving_size))}
engineering <- change_to_grams(engineering)
```

Full Features *Alter validation and tuning sets as we did to the engineering set*

```r
full_features <- function(data) {
  data <- data %>%
    NA_fix() %>%
    brand_cleaning() %>%
    add_category(brand_one_category) %>%
    household_serving_fulltext_cleaning() %>%
    descreption_cleaning() %>%
    add_keyword_detection_columns(keywords) %>%
    change_to_grams() %>%
    select(-c(brand, serving_size_unit, description, ingredients,
household_serving_fulltext))

  return(data)
}

validation <- full_features(validation)
tuning_engd <- full_features(tuning)
```

Model Tuning - GBT

GBT is great for classifying things and dealing with tricky data because it uses a bunch of decision trees that work together step by step. This helps it figure out complicated patterns and relationships that might be hard for other methods to spot. Therefore, we decided to use GBT model.

In the beginning, we splitted the original train_food data to validation, engineering and tuning sets, and now we are using the tuning set to determine the hyperparameters of the model.

```r
tuning_engd_ready <- tuning_engd %>% select(-idx)
cv_splits <- vfold_cv(tuning_engd_ready, v=5)
rec <- recipe(category ~., data=tuning_engd_ready) %>%
step_dummy(all_nominal_predictors())
mod_01 <- boost_tree(mode="classification", engine="xgboost",learn_rate =
tune(), sample_size = tune())
param_grid <- expand_grid(learn_rate=c(0.7, 0.75, 0.8), sample_size=c(0.7,
0.75, 0.8))
tune_res <- tune_grid(object=mod_01, preprocessor=rec,
resamples=cv_splits, grid=param_grid,
                        metrics=metric_set(accuracy),
control=control_grid(verbose=TRUE))
collected_metrics <- collect_metrics(tune_res)
show_best(tune_res, n=6, metric="accuracy")

## # A tibble: 6 x 8
##   learn_rate sample_size .metric  .estimator  mean     n std_err .config
##        <dbl>       <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1       0.7         0.8  accuracy multiclass 0.896     5 0.00361 Preprocessor1_~
## 2       0.75        0.8  accuracy multiclass 0.894     5 0.00298 Preprocessor1_~
## 3       0.8         0.8  accuracy multiclass 0.893     5 0.00326 Preprocessor1_~
## 4       0.7         0.75 accuracy multiclass 0.893     5 0.00262 Preprocessor1_~
## 5       0.7         0.7  accuracy multiclass 0.891     5 0.00499 Preprocessor1_~
## 6       0.8         0.7  accuracy multiclass 0.891     5 0.00204 Preprocessor1_~
```

The best hyperparameters are : learn rate = 0.75 , sample_size = 0.70

Gather the data sets In the end, it's smarter to use all the data sets together to build a more accurate model. That's why we're combining the tuning and engineering sets. Moreover, we prepared the sets for prediction.

```
engineering <- engineering %>% select(-
c(brand,serving_size_unit,description,ingredients,household_serving_fullte
xt))
train_ready <- bind_rows(engineering, tuning_engd) %>% select(-idx)
train_ready$serving_size <- as.numeric(train_ready$serving_size)
validation$serving_size <- as.numeric(validation$serving_size)
rec <- recipe(category ~., data=train_ready) %>%
  step_dummy(all_nominal_predictors()) %>%
  prep(train_ready)
train_ready <- rec %>% bake(train_ready)
val_ready <- rec %>% bake(validation)
```

Model after tuning the hyperparameters

```
set.seed(42)
mod_01 <- boost_tree(mode = "classification", engine = "xgboost",
learn_rate = 0.75, sample_size = 0.70) %>%
  fit(category ~ ., data = train_ready)
```

Assess model performance

```
predictions <- mod_01 %>% predict(new_data=val_ready)
accuracy_calc <- sum(predictions == validation$category) /
length(predictions$.pred_class)
print(paste("Accuracy on the validation set:", round(accuracy_calc, 4)))

## [1] "Accuracy on the validation set: 0.9114"
```

Ultimately, we'll merge all the data as we learned in class, which should improve our performance on the test data. *Gather all the train data*

```
train_ready <- bind_rows(engineering, tuning_engd,validation) %>% select(-
idx)
train_ready$serving_size <- as.numeric(train_ready$serving_size)
rec <- recipe(category ~., data=train_ready) %>%
  step_dummy(all_nominal_predictors()) %>%
  prep(train_ready)
train_ready <- rec %>% bake(train_ready)
```

Final first Model

```
set.seed(42)
mod_01 <- boost_tree(mode = "classification", engine = "xgboost",
learn_rate = 0.75, sample_size = 0.70) %>%
  fit(category ~ ., data = train_ready)
```

Predict on test data

```
test_ready <- full_features(food_test)
test_ready <- rec %>% bake(test_ready)
```

FIRST TRY- Result:

```
pred_cat <- mod_01 %>% predict(new_data=test_ready)
write_csv(cbind(food_test$idx, pred_cat), "model01.csv")
```

SECOND TRY:

Lets add the nutrients data to the model: We will find the most important nutrients and create a column for them. The value will be 0 if the substance doesn't exists in the snack, and the amount otherwise: *Modify the name column in nutrients - converting it to lowercase and replacing spaces with underscores, while also removing commas, making the column's values more consistent and easier to work with.*

```
nutrients <- nutrients %>%
  mutate(name = str_replace_all(tolower(name), " ", "_"),
         name = str_replace_all(name, ",", ""))
```

Identifying important nutrients and constructing a keyword vector for them

```
merged_food_data <- food_nutrients %>% left_join(nutrients, by =
"nutrient_id") %>% inner_join(engineering, by = "idx")
nutrient_stats <- merged_food_data %>%
   mutate(name = ifelse(str_detect(name, "fatty_acids"), "fatty_acids",
              ifelse(str_detect(name, "carbohydrate"), "carbohydrate",
              ifelse(str_detect(name, "fiber"), "fiber",
              ifelse(str_detect(name, "vitamin_e"), "vitamin_e",
              ifelse(str_detect(name, "vitamin_d"), "vitamin_d",
name)))))) %>%
  group_by(name) %>%
  mutate(num_snacks=n(), num_categories = n_distinct(category),
mean_amount = mean(amount)) %>%
  select(name, num_snacks, num_categories, mean_amount) %>%
  arrange(desc(num_snacks)) %>%
  distinct() %>%
  filter(num_categories<6, num_snacks>18) #by filtering nutrients with
```

```
less than 6 categories and present in more than 18 snacks, we aim to
identify more indicative ingredients for predicting snacks categories.
nutrients_keywords <- nutrient_stats$name
```

Add important nutrients as columns - with each column indicating the quantity of the respective nutrient present in the snack

```
get_nutrient_amount <- function(data, nutrient_keyword) {
 relevant_nutrient_id <- nutrients %>% filter(name==nutrient_keyword) %>%
select(nutrient_id) %>% distinct() # find nutrient_id
 relevant_data <- data %>% left_join(food_nutrients %>% filter(nutrient_id
%in% relevant_nutrient_id) %>% select(idx, amount), by = "idx")
 relevant_data <- relevant_data  %>% mutate(amount =
ifelse(is.na(amount),0,amount)) # replace NA with 0
 # aggregate nutrient amounts for each snack
 nutrient_amounts <- relevant_data %>% group_by(idx) %>% summarize(amount
= sum(amount))
 return(nutrient_amounts$amount)
}
create_nutrient_columns <- function(data, nutrient_keywords) {
  nutrient_data <- map_dfc(nutrient_keywords, ~get_nutrient_amount(data,
.x))
  names(nutrient_data) <- nutrient_keywords
  return(cbind(data, nutrient_data))
}
engineering <- create_nutrient_columns(engineering,nutrients_keywords)
tuning_engd <- create_nutrient_columns(tuning_engd,nutrients_keywords)
validation <- create_nutrient_columns(validation,nutrients_keywords)
```

Model Tuning - GBT:

We will continue with selecting the GBT model.

```
tuning_engd_ready <- tuning_engd %>% select(-idx)
cv_splits <- vfold_cv(tuning_engd_ready, v=5)
rec <- recipe(category ~., data=tuning_engd_ready) %>%
step_dummy(all_nominal_predictors())
mod_02 <- boost_tree(mode="classification", engine="xgboost",learn_rate =
tune(), sample_size = tune())
param_grid <- expand_grid(learn_rate=c(0.5, 0.55, 0.6, 0.65, 0.7),
sample_size=c(0.7, 0.75, 0.8, 0.85))
tune_res <- tune_grid(object=mod_02, preprocessor=rec,
resamples=cv_splits, grid=param_grid,
                        metrics=metric_set(accuracy),
control=control_grid(verbose=TRUE))
collected_metrics <- collect_metrics(tune_res)
show_best(tune_res, n=5, metric="accuracy")

## # A tibble: 5 x 8
##   learn_rate sample_size .metric  .estimator  mean     n std_err
.config
##        <dbl>       <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1        0.6         0.8  accuracy multiclass 0.899     5 0.00555 Preprocessor1_~
## 2        0.65        0.85 accuracy multiclass 0.898     5 0.00540 Preprocessor1_~
## 3        0.7         0.7  accuracy multiclass 0.897     5 0.00372 Preprocessor1_~
## 4        0.65        0.7  accuracy multiclass 0.897     5 0.00590 Preprocessor1_~
## 5        0.6         0.75 accuracy multiclass 0.897     5 0.00469 Preprocessor1_~
```

The best hyperparameters are : learn rate = 0.7 , sample_size = 0.85

Gather the tuning and engineering data sets

```
train_ready <- bind_rows(engineering, tuning_engd) %>% select(-idx)
train_ready$serving_size <- as.numeric(train_ready$serving_size)
validation$serving_size <- as.numeric(validation$serving_size)
rec <- recipe(category ~., data=train_ready) %>%
  step_dummy(all_nominal_predictors()) %>%
  prep(train_ready)
train_ready <- rec %>% bake(train_ready)
val_ready <- rec %>% bake(validation)
```

Model after tuning the hyperparameters

```
set.seed(42)
mod_02 <- boost_tree(mode = "classification", engine = "xgboost",
learn_rate = 0.7, sample_size = 0.85) %>%
  fit(category ~ ., data = train_ready)
```

Assess model performance

```
predictions <- mod_02 %>% predict(new_data=val_ready)
accuracy_calc <- sum(predictions == validation$category) /
length(predictions$.pred_class)
print(paste("Accuracy on the validation set:", round(accuracy_calc, 4)))

## [1] "Accuracy on the validation set: 0.9061"
```

Gather all the train data

```
train_ready <- bind_rows(engineering, tuning_engd,validation) %>% select(-
idx)
train_ready$serving_size <- as.numeric(train_ready$serving_size)
rec <- recipe(category ~., data=train_ready) %>%
  step_dummy(all_nominal_predictors()) %>%
  prep(train_ready)
train_ready <- rec %>% bake(train_ready)
```

Final second Model

```
set.seed(42)
mod_02 <- boost_tree(mode = "classification", engine = "xgboost",
learn_rate = 0.7, sample_size = 0.85) %>%
  fit(category ~ ., data = train_ready)
```

Predict on test data

```
test_ready <- full_features(food_test) %>% create_nutrient_columns(.,
nutrients_keywords)
test_ready <- rec %>% bake(test_ready)
```

SECOND TRY - Result

```
pred_cat <- mod_02 %>% predict(new_data=test_ready)
write_csv(cbind(food_test$idx, pred_cat), "model02.csv")
```

Based on our evaluation, the first model outperformed the second one. Consequently, we've chosen to proceed with the first model and enhance it by incorporating image processing analysis.

THIRD TRY:

Now we will combined the image processing, and add it to the prediction model.

Load the data:

```
zip_file_path <- "C:/Users/sofi1/Desktop/Final_Project-
Sophiep1103/foods_final.zip"
destination_folder <- "C:/Users/sofi1/Desktop/Final_Project-
Sophiep1103/images_final"
unzip(file.path(zip_file_path, "images.zip"), exdir = zip_file_path)
train_folder <- file.path(zip_file_path, "train")
test_folder <- file.path(zip_file_path, "test")
```

Prepare the images and split the engineering data to train and validation sets.

```
engineering_idx <- engineering$idx
validation_idx <- validation$idx
tuning_idx <- tuning$idx

class_names <- c("cakes_cupcakes_snack_cakes", "candy",
"chips_pretzels_snacks", "chocolate", "cookies_biscuits",
"popcorn_peanuts_seeds_related_snacks")

train_datagen <- image_data_generator(
  rotation_range = 20,
  width_shift_range = 0.1,
  height_shift_range = 0.1,
  shear_range = 0.2,
  zoom_range = 0.2,
  horizontal_flip = TRUE,
  fill_mode = 'nearest'
)

train_df_all <- tibble(
  path = list.files(
    "C:/Users/sofi1/Desktop/Final_Project-Sophiep1103/images_final/train",
    full.names = TRUE,
    recursive = TRUE
  ),
  idx = as.numeric(gsub("[^0-9]", "", basename(path)))
)

test_df <- tibble(
  path = list.files(
    "C:/Users/sofi1/Desktop/Final_Project-Sophiep1103/images_final/test",
    full.names = TRUE,
    recursive = TRUE
  ),
  idx = as.numeric(gsub("[^0-9]", "", basename(path)))
)

train_df_all <- train_df_all %>%
  mutate(is_in_engineering_idx = idx %in% engineering_idx) %>%
  mutate(is_in_validation_idx = idx %in% validation_idx) %>%
  mutate(is_in_tuning_idx = idx %in% tuning_idx) %>%
  mutate(category = str_extract(path, "(?<=train\\/).*?(?=\\/)"))

engineering_df <-  train_df_all %>%
  filter (is_in_engineering_idx) %>%
  select(c(path, idx, category))

validation_df <-  train_df_all %>%
  filter (is_in_validation_idx) %>%
  select(c(path, idx, category))

tuning_df <- train_df_all %>%
  filter (is_in_tuning_idx)%>%
  select(c(path, idx, category))

engineering_set <- engineering_df %>%
  flow_images_from_dataframe(
    dataframe = .,
    directory = "C:/Users/sofi1/Desktop/Final_Project-
Sophiep1103/images_final/train",
    generator = train_datagen,
    target_size = c(64, 64),
    batch_size = 32,
    class_mode = 'categorical',
    x_col = 'path',
    y_col = 'category'
  )

validation_set <- validation_df %>%
  flow_images_from_dataframe(
    dataframe = .,
    directory = "C:/Users/sofi1/Desktop/Final_Project-
Sophiep1103/images_final/train",
    generator = train_datagen,
    target_size = c(64, 64),
    batch_size = 32,
    class_mode = 'categorical',
    shuffle = FALSE,
    x_col = 'path',
    y_col = 'category'
  )

tuning_set <- tuning_df %>%
  flow_images_from_dataframe(
    dataframe = .,
    directory = "C:/Users/sofi1/Desktop/Final_Project-
Sophiep1103/images_final/train",
    generator = train_datagen,
    target_size = c(64, 64),
    batch_size = 32,
    class_mode = 'categorical',
    shuffle = FALSE,
    x_col = 'path',
    y_col = 'category'
  )

test_datagen <- image_data_generator(rescale = 1/255)

test_set <- test_df %>%
  flow_images_from_dataframe(
    dataframe = .,
    directory = "C:/Users/sofi1/Desktop/Final_Project-
Sophiep1103/images_final/test",
    generator = test_datagen,
    target_size = c(64, 64),
    batch_size = 32,
    class_mode = NULL,  # No labels in the test set
    x_col = 'path'
  )

#split the engineering data to validation and train for the training of
the cnn on the images.
train_ratio <- 0.9

num_samples <- nrow(engineering_df)
num_train <- round(num_samples * train_ratio)
num_validation <- num_samples - num_train

indices <- sample(num_samples)

train_indices <- indices[1:num_train]
validation_indices <- indices[(num_train + 1):num_samples]

engineering_train_df <- engineering_df[train_indices, ]
engineering_validation_df <- engineering_df[validation_indices, ]

engineering_train_set <- engineering_train_df %>%
  flow_images_from_dataframe(
    dataframe = .,
    directory = "C:/Users/sofi1/Desktop/Final_Project-
Sophiep1103/images_final/train",
    generator = train_datagen,
    target_size = c(64, 64),
    batch_size = 32,
    class_mode = 'categorical',
    x_col = 'path',
    y_col = 'category'
  )

engineering_validation_set <- engineering_validation_df %>%
  flow_images_from_dataframe(
    dataframe = .,
    directory = "C:/Users/sofi1/Desktop/Final_Project-
Sophiep1103/images_final/train",
    generator = train_datagen,
    target_size = c(64, 64),
    batch_size = 32,
    class_mode = 'categorical',
    x_col = 'path',
    y_col = 'category'
  )

## Found 21434 validated image filenames belonging to 6 classes.
## Found 3173 validated image filenames belonging to 6 classes.
## Found 7144 validated image filenames belonging to 6 classes.
## Found 3525 validated image filenames.
## Found 19291 validated image filenames belonging to 6 classes.
## Found 2143 validated image filenames belonging to 6 classes.
```

Build the CNN model : We chose the CNN model for image processing because it is a good model for learning detailed patterns and features in images. Images has many pixels, and CNN works well on large inputs. This helps accurately understand complex visual information, like classifying images.

```
model_03 <- keras_model_sequential()
model_03 %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
input_shape = c(64, 64, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 100, activation = "relu") %>%
  layer_dense(units = length(class_names), activation = "softmax")

model_03 %>% compile(
  optimizer = optimizer_adam(),
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)

history <- model_03 %>% fit_generator(
  engineering_train_set,
  steps_per_epoch = nrow(engineering_train_df) / 32,
  epochs = 10,
  validation_data = engineering_validation_set,
  validation_steps = nrow(engineering_validation_df) / 32,
  verbose =0
)
```

```
images_engineering <- model_03 %>% predict(engineering_set)
images_validation <- model_03 %>% predict(validation_set)
images_tuning <- model_03 %>% predict(tuning_set)
images_test <- model_03 %>% predict(test_set)
```

Model Tuning - GBT

Now, we're going to integrate the output from the image processing step (the probabilities assigned to each category) into the first model.

```
#combining the tuning set to the output from the image processing
tuning_engd_ready <- cbind(tuning_engd, images_tuning) %>% select(-idx)
cv_splits <- vfold_cv(tuning_engd_ready, v=5)
rec <- recipe(category ~., data=tuning_engd_ready) %>%
step_dummy(all_nominal_predictors())
mod_03 <- boost_tree(mode="classification", engine="xgboost",learn_rate =
tune(), sample_size = tune())
param_grid <- expand_grid(learn_rate=c(0.6, 0.65, 0.7, 0.75),
sample_size=c(0.7, 0.75, 0.8))
tune_res <- tune_grid(object=mod_03, preprocessor=rec,
resamples=cv_splits, grid=param_grid,
                        metrics=metric_set(accuracy),
control=control_grid(verbose=TRUE))
collected_metrics <- collect_metrics(tune_res)
show_best(tune_res, n=6, metric="accuracy")

## # A tibble: 5 x 8
##   learn_rate sample_size .metric  .estimator  mean     n std_err .config
##        <dbl>       <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1       0.6         0.8  accuracy multiclass 0.892     5 0.00660 Preprocessor1_~
## 2       0.65        0.8  accuracy multiclass 0.891     5 0.00562 Preprocessor1_~
## 3       0.7         0.8  accuracy multiclass 0.891     5 0.00641 Preprocessor1_~
## 4       0.6         0.7  accuracy multiclass 0.891     5 0.00739 Preprocessor1_~
## 5       0.65        0.75 accuracy multiclass 0.890     5 0.00700 Preprocessor1_~
## 6       0.6         0.75 accuracy multiclass 0.889     5 0.00613 Preprocessor1_~
```

The best hyperparameters are : learn rate = 0.6 , sample_size = 0.8

Gather data:

```
#combining the engineering set to the output from the image processing
engineering <- cbind(engineering,images_engineering) %>% select(-
c(brand,serving_size_unit,description,ingredients,household_serving_fullte
xt))
tuning_engd <- cbind(tuning_engd, images_tuning)
train_ready <- bind_rows(engineering, tuning_engd) %>% select(-idx)

train_ready$serving_size <- as.numeric(train_ready$serving_size)
validation$serving_size <- as.numeric(validation$serving_size)
#combining the validation set to the output from the image processing
validation <- cbind(validation, images_validation)


rec <- recipe(category ~., data=train_ready) %>%
  step_dummy(all_nominal_predictors()) %>%
  prep(train_ready)
train_ready <- rec %>% bake(train_ready)
val_ready <- rec %>% bake(validation)
```

Model after tuning the hyperparameters

```
set.seed(42)
mod_03 <- boost_tree(mode = "classification", engine = "xgboost",
learn_rate = 0.6, sample_size = 0.8) %>%
  fit(category ~ ., data = train_ready)
```

Assess model performance

```
predictions <- mod_03 %>% predict(new_data=val_ready)
accuracy_calc <- sum(predictions == validation$category) /
length(predictions$.pred_class)
print(paste("Accuracy on the validation set:", round(accuracy_calc, 4)))

## [1] "Accuracy on the validation set: 0.9089"
```

Gather all the train data

```
train_ready <- bind_rows(engineering, tuning_engd,validation) %>% select(-
idx)
train_ready$serving_size <- as.numeric(train_ready$serving_size)
rec <- recipe(category ~., data=train_ready) %>%
  step_dummy(all_nominal_predictors()) %>%
  prep(train_ready)
train_ready <- rec %>% bake(train_ready)
```

Final third Model

```
set.seed(42)
mod_03 <- boost_tree(mode = "classification", engine = "xgboost",
learn_rate = 0.6, sample_size = 0.8) %>%
  fit(category ~ ., data = train_ready)
```

Predict on test data

```
test_ready <- full_features(food_test)
test_ready <- cbind(test_ready,images_test)
test_ready <- rec %>% bake(test_ready)
```

Third try - Result

```
pred_cat <- mod_03 %>% predict(new_data=test_ready)
write_csv(cbind(food_test$idx, pred_cat), "model03.csv")
```

SUMMARY

Missing values- there are not many missing values. Therefore, we decided not to focus on finding the best solution. Instead, we simply filled in the gaps with a new “Unknown” category.

Next Steps: Moving on, we proceeded with feature engineering. To begin, we followed the process of cleaning the “Brand” column as outlined earlier. Given our observation (in the exploratory data analysis) that many brands are linked to just one category, we took the decision to introduce six new columns – one for each category. These columns serve as indicators, showing whether a brand is connected to a particular category or not.

Further, we extended the cleaning process to other categories such as Ingredients, Household Serving Fulltext, and Description, following the previously outlined approach. For each column, we formed a vector of noteworthy keywords and introduced corresponding columns for these keywords. We counted the occurrence of each keyword across all these columns and assigned the count as a value in the respective keyword column.

Serving Size: During our analysis, we identified a few items listed in milliliters (ml) instead of grams (grm). To address this, we performed conversions to ensure consistency in serving size units.

Full features: Similar to our classroom approach, we developed a "full_feature" function encompassing all the data engineering steps we undertook. This function serves to maintain reproducibility. We applied this function on both the tuning and validation sets to ensure consistency.

Once we used the full_feature function, models 2 and 3 included few more steps that we'll discuss further.

Tuning and model selection : we use our tuning data to find the best hyperparameters of the model, and decided to use GBT model.

After tuning, we evaluated the model's performance using the validation set. Subsequently, we merged the validation, engineered, and tuned sets. We then trained the model on this combined dataset and employed it to make predictions on the test set.

model 1: Includes only analysis on food_train data model 2: Includes only analysis on food_train, nutrients and food_nutrients data- We cleaned the nutrients data and identified the most important nutrients. Subsequently, we generated columns for these essential nutrients, recording the respective amounts present in each item. model 3: Includes only analysis on food_train and the images- We developed a Convolutional Neural Network to analyze the images. The end outcome comprises six probabilities for each snack, representing the likelihood of it belonging to a specific category. We formulated a new model that incorporates both the six columns we generated and all the data utilized in the first model.

The best model according to our assessments is model_01. However, due to minor differences, the ultimate best test model might vary.

REFRANCES to additional material:

1. https://srdas.github.io/DLBook/DeepLearningWithR.html
2. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
3. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
4. https://www.geeksforgeeks.org/training-vs-testing-vs-validation-sets/
5. https://www.machinelearningplus.com/machine-learning/an-introduction-to-gradient-boosting-decision-trees/#google_vignette