Open Food Facts - Supervised Learning Analysis

Judah Axelrod

Part I - Data Preparation and EDA

Data Source and Research Questions

https://www.kaggle.com/openfoodfacts/world-food-facts
The raw data has 314,496 observations with 163 variables. I am interested in answering 2 main questions:

1. Can I build a model to accurately predict the nutrition score that France assigns to each food?
2. What are some of the underlying variables that drive the nutrition score?

Data Cleaning

I take the following steps to prepare the data for modeling:

1. First, limit to only observations that have a France nutrition score value, as this will be the outcome variable of interest.

2. Compute and plot the number of missing values for each column. To be conservative, I only discard variables that have less than 100,000 non-missing values, as there seems to be an inflection point around 100,000 when I plot the sorted number of non-missing observations for every variable. After eliminating a few irrelevant variables, I am left with 19 predictors.

3. I had to drop the 'Country' variable because the remaining data were so heavily skewed toward the United States that there was little variation to exploit.

4. Finally, remove all missing values for the remaining variables. This leaves me with a dataset of 96,333 observations and 20 total variables. Here is the list of candidate predictors after data cleaning:

Predictors			
Energy per 100g	Saturated Fat per 100g	Sugars per 100g	Proteins per 100g
Salt per 100g	Sodium per 100g	Calcium per 100g	Fat per 100g
Fiber per 100g	Cholesterol per 100g	Number of Additives	Trans Fat per 100g
Iron per 100g	Vitamin A per 100g	Vitamin C per 100g	Carbohydrates per 100g
Ingredients from Palm Oil	Ingredients that may be from Palm Oil	Brand(s) associated with the food	

Brand Cleaning

- There are over 16,000 unique brands in the dataset, which is far too many values of a categorical variable to be computationally feasible. I create a brands lookup file to merge on a cleaned brands value.
- Any brands associated with less than 150 unique foods are grouped into a large 'Other' category.
- For foods with multiple brands, choose just the first one listed for simplicity.

- After cleaning, I am left with just 44 brands, a more feasible number.

Highlights of Further Data Exploration

- Taking a look at the distribution for nutrition score, the values range from -10 to 36, and they seem to have two peaks, roughly around 0 and 14. The overlay of a normal density line onto the histogram confirms that the scores are not normally distributed, as does a QQ-plot.
- I also generate a correlation heat map to get a sense of what predictors may be most important. Judging by this tile plot, the predictors most correlated with nutrition score are a food's energy, saturated fat, sugar, fat, and fiber content.

Part II - Modeling France's Nutrition Score

First, I divide the dataset into a $\frac{2}{3}$ training/validation set used to tune each model, and a $\frac{1}{3}$ test set, which will be used to compare the final forms of each type of model.

Variable Selection and Linear Regression

I first run a LASSO regression to perform feature selection. Because the LASSO uses an L_1 penalty, it can choose a sparser model with some of the 19 possible independent variables being shrunk to zero. After fitting a LASSO to the training set and performing 10-fold cross validation, only 4 of the variables have nonzero coefficients: Energy per 100g, Saturated Fat per 100g, Sugars per 100g, and Fiber per 100g. These overlap nicely with the most highly correlated variables found during EDA, so the result looks more trustworthy.

The LASSO shrinks the magnitudes of all coefficients, even the nonzero ones, and therefore introduces bias into the estimation. Because of this, I fit the following linear regression model to the training data using just the selected variables. I will calculate a test error at the end of the analysis:

$$nutritionScore_i = \beta_0 + \beta_1 Energy_i + \beta_2 SatFat_i + \beta_3 Sugars_i + \beta_4 Fibers_i$$

KNN Regression

I want to compare the Linear Regression results with a non-parametric alternative, KNN Regression, to see if a more flexible method will perform better. I use just the 4 variables selected above by the LASSO in order to directly compare the two statistical learning methods. Again, I run 10-fold cross validation in order to select an optimal number of neighbors k to consider, finding the lowest CV-error to occur at $k = 5$. Finally, I refit the 5-NN model and use it compute a test error on the test dataset. Since 5 is a relatively small number of neighbors for such a large dataset, this should yield a far more flexible fit than the Linear Regression above.

Generalized Additive Model

The GAM serves as a compromise between the inflexibility of the Linear Regression which could lead to higher bias, and the very flexible, nonparametric KNN Regression which may be more prone to overfitting. Because I am interested not just in prediction, but also in better understanding the drivers of nutrition score, a GAM allows us to retain the additive, parametric structure of linear regression while utilizing splines to account for the likely non-linearities inherent in the data.

To perform variable selection here, I use the group LASSO (implemented by using the 'select = TRUE' option in the mgcv package). The group LASSO uses an L_2 penalty, which means that either the entire group is selected or none of the smoothing parameters corresponding to a given predictor are selected (i.e. there is within-group sparsity).

Interestingly, I can see from the summary and plots of each smoothing function that almost none of the predictors are shrunk to exactly zero; in fact, only Iron per 100g looks to be zero from the plot. The plots of several other variables (like Trans Fat per 100g, Vitamin A per 100g, etc.) look heavily shrunken toward zero but not entirely. This suggests that almost all the numeric variables play at least a small role in predicting

nutrition score. I will later calculate a test error on this fitted GAM model, which I note performed far better on the test set than a GAM fitted without the group LASSO shrunken coefficients.

One final note on the GAM: I can see from the *gam.check* diagnostic that many of the p-values corresponding to the k-indices are extremely low even though the corresponding coefficients were shrunk to zero. This is likely because the dimension k of the basis was too low. Had I been able to increase k, the GAM may have led to stabler fits - and indeed in the plots I can see some unstable behavior at the tails for several predictors. However, to increase the basis size for all of the candidate predictors was too computationally intensive for this analysis.

Tree-Based Methods

Regression Tree

Next, I move on to tree-based methods, beginning with just a single regression tree. Including all predictors (besides the brand), I use 10-fold cross validation on the training/validation set to select the optimal number of terminal nodes through cost-complexity pruning, which I find to be 9. I then trim the original tree to obtain our final subtree with 9 terminal nodes.

Plotting the tree, note that the first split occurs using Saturated Fat per 100g, while further splits occur on Salt, Sugars, and Fiber per 100g. Again, there is significant overlap between the predictors used for splitting and those obtained from variable selection above, a nice sanity check of the results.

Random Forest

A single decision tree, as will become clear, tends to perform poorly in terms of prediction and suffer from low robustness. By using the bootstrap approach to repeatedly sample from our data, I can obtain a more powerful result through Bagging. However, to ensure that the same variable is not selected at the first split for all B trees, I use the Random Forest approach to only consider a subset m of all p predictors as candidates at every split. Specifically, the RandomForest package recommends choosing $\frac{p}{3}$ predictors for a regression specification, so I choose $m = 6$.

To select the number of trees, I picked as large a number as was computationally feasible, in this case 250 trees. Fitting on the training set, I calculated an out-of-bag MSE for each number of trees and plotted the result. Beyond roughly 150 trees, the decrease in MSE is negligible, and I therefore use a 150-tree random forest as our preferred model on the test dataset.

Looking at the variable importance plot, I again see Saturated Fat, Sugars, and Fiber per 100g as among the predictors that led to (1) the largest increase in MSE when excluded and (2) the largest increase in node purity when included.

Boosting

Finally, I make use of boosting, which grows each new tree based on the residuals of previous trees and then updates the model by adding a shrunken version of that tree to the previous results. In addition to the number of trees B, I also need to choose the shrinkage parameter λ and the interaction depth d. To do this, I create a parameter grid and fit a boosting model for each combination of candidate parameter values. Due to computational limitations, I don't use CV for this; rather, I simply use a 70% training set and 30% validation set, as well as $B = 5,000$ trees.

The validation error would keep decreasing for more trees, but it was computationally impractical to use a higher number - plus, unlike a random forest, boosting can be prone to eventual overfitting. The results of tuning yield $B = 5,000$, $\lambda = 0.010$, $d = 5$.

Again, when I summarize the model, I see Saturated Fat, Salt, Sugars, and Fiber per 100g as the predictors with the highest relative importance, yet another result consistent with the other models.

Results

Now that I have fit and tuned all of our models on the training/validation set, I calculate a Root Mean Squared Error and Mean Absolute Error for each one on the $\frac{1}{3}$ test set:

Model	Root Mean Squared Error	Mean Absolute Error
Linear Regression	4.92	3.78
5-NN Regression	3.24	2.13
Generalized Additive Model	1.76	1.05
Decision Tree	3.63	2.76
Random Forest	**0.69**	**0.35**
Boosting	0.71	0.41

Based on these test errors, the Random Forest is my model of choice, slightly beating out the boosting model. This is to be expected, as these are two of the highest-performing statistical learning methods. The GAM also performed very well, suggesting that it is a good compromise between predictive power and interpretability. As expected, the decision tree and linear regression models did not perform as well. Here, I really consider these only as baselines for their more advanced counterparts. Overall, it looks like this is a highly nonlinear regression problem, and the random forest provides the best way to predict nutrition score.

The second part of the analysis was to better understand the drivers of nutrition score, and fortunately, the results were pretty similar among all the models. While nearly all of the numeric variables corresponding to nutrition information play at least some role in the score, Saturated Fat, Sugars, Energy, Salt, and Fiber per 100g are generally the most important predictors of nutrition score.

Limitations and Future Directions

1. One of the major limitations of this analysis is computational power. At several points, I had to make concessions. Here are some examples:
    a. Cleaning the brands in the data to reduce the number of unique values. Maybe brands would have had more predictive power if I had not transformed it or dropped it altogether in many of the later model specifications.
    b. While the GAM performed very well as is, I would have liked to improve upon the variable selection procedure by being able to experiment with different basis dimensions for the various smoothing terms to obtain stabler results.
    c. In the boosting model, I was forced to use a training/validation structure instead of performing cross validation to select the optimal parameters. This means the obtained errors likely have higher variance as they are more dependent on the random split of data.

2. The data itself also forced some compromising:
    a. I was left with just over 30% of the original data and roughly $\frac{1}{8}$ of the original predictors after running diagnostics to identify missing values.
    b. There were other variables I would have liked to use that just required too much cleaning for the scope of this analysis. One example was Serving Size, which varied in units and format across foods without a clear way to standardize the measurement.

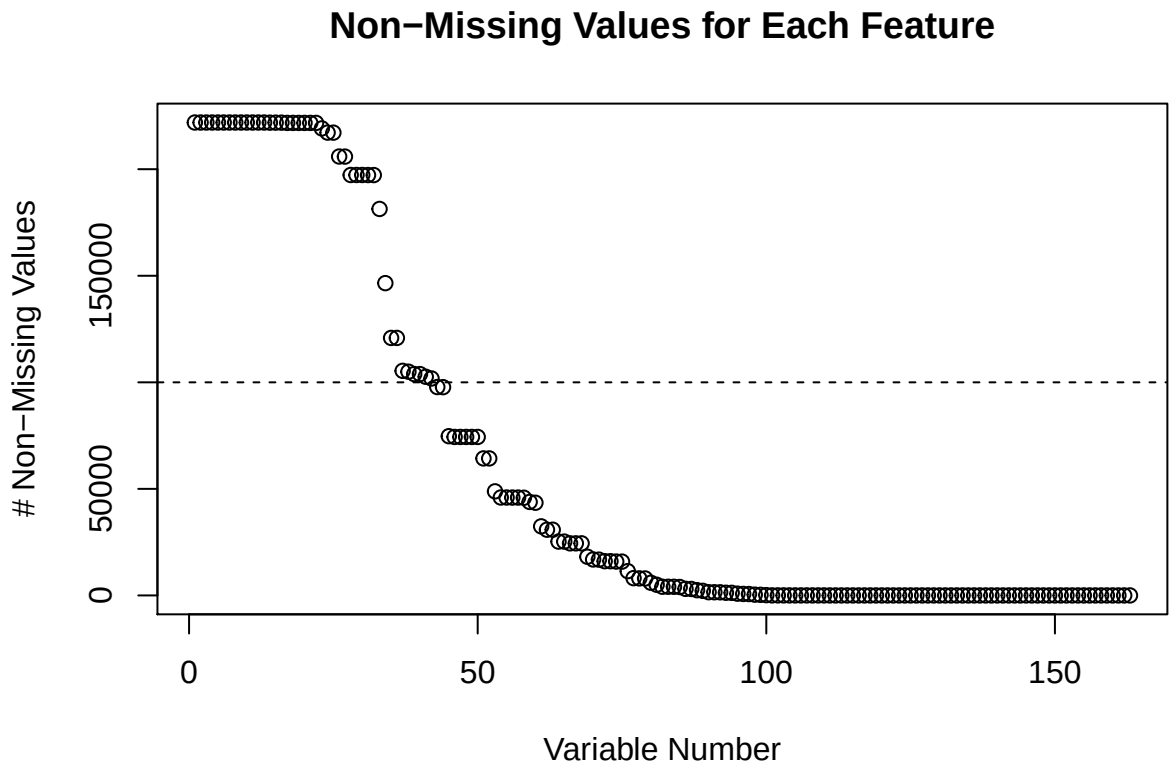
3. This is a dynamic dataset, and it would be interesting to return to the analysis when more missing values are filled in, especially for foods in countries besides the United States.

4. France also computes a nutrition score (A, B, C, D, or E) which is directly based on the nutrition grade. Future work could explore the predictive accuracy of multi-class classification models applied to this research question.

Appendix

Data Cleaning

```
food_lim <- food %>%
  filter(!is.na(`nutrition-score-fr_100g`)) %>%
  rename(nutrition_score_fr = `nutrition-score-fr_100g`)


missing_vals <- data.frame(missing = map_dbl(food_lim, function(x) sum(is.na(x))),
                           nonmissing = map_dbl(food_lim, function(x) sum(!is.na(x)))) %>%
  arrange(missing)

plot(missing_vals$nonmissing, main = 'Non-Missing Values for Each Feature',
     xlab = 'Variable Number', ylab = '# Non-Missing Values')
abline(h = 100000, lty = 'dashed')
```




```
rel_vars <- rownames(filter(missing_vals, nonmissing > 100000))


food_lim <- food_lim %>%
  select(nutrition_score_fr, country = countries_en, brands, energy_100g,
         sat_fat_100g = `saturated-fat_100g`, sugars_100g, proteins_100g,
         salt_100g, sodium_100g, fat_100g, carbs_100g = carbohydrates_100g,
         additives_n, from_palm = ingredients_from_palm_oil_n,
         maybe_palm = ingredients_that_may_be_from_palm_oil_n,
         fiber_100g, cholest_100g = cholesterol_100g, calcium_100g,
         trans_fat_100g = `trans-fat_100g`, iron_100g,
         vit_a_100g = `vitamin-a_100g`, vit_c_100g = `vitamin-c_100g`) %>%
  na.omit()
```

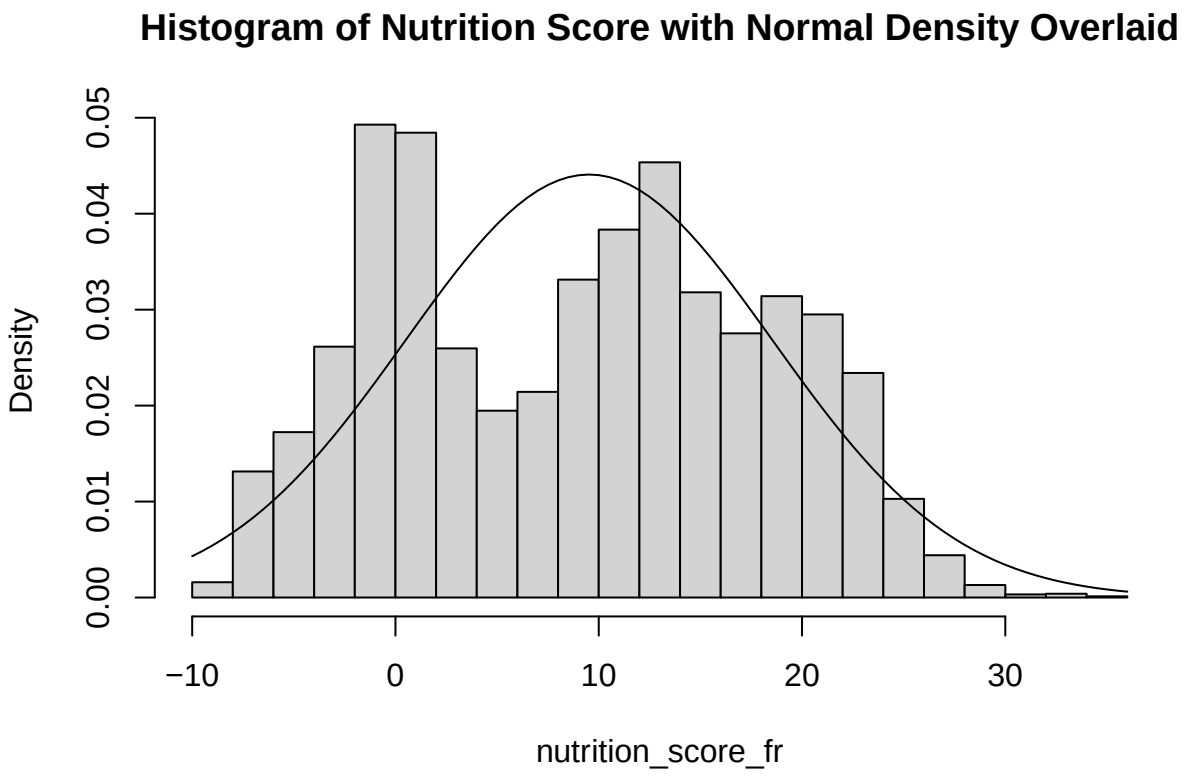
Brands Lookup

- Generate counts of brand names to facilitate data cleaning in Excel
- Notes on brand cleaning:
  1. If multiple brands listed, went with the first listed name for simplicity
  2. Classified brands with < 150 observations as 'Other" to avoid too many distinct values
- Re-import the lookup 'brands_lookup.csv' after cleaning the brands
- Merge the clean brand onto the rest of the food data

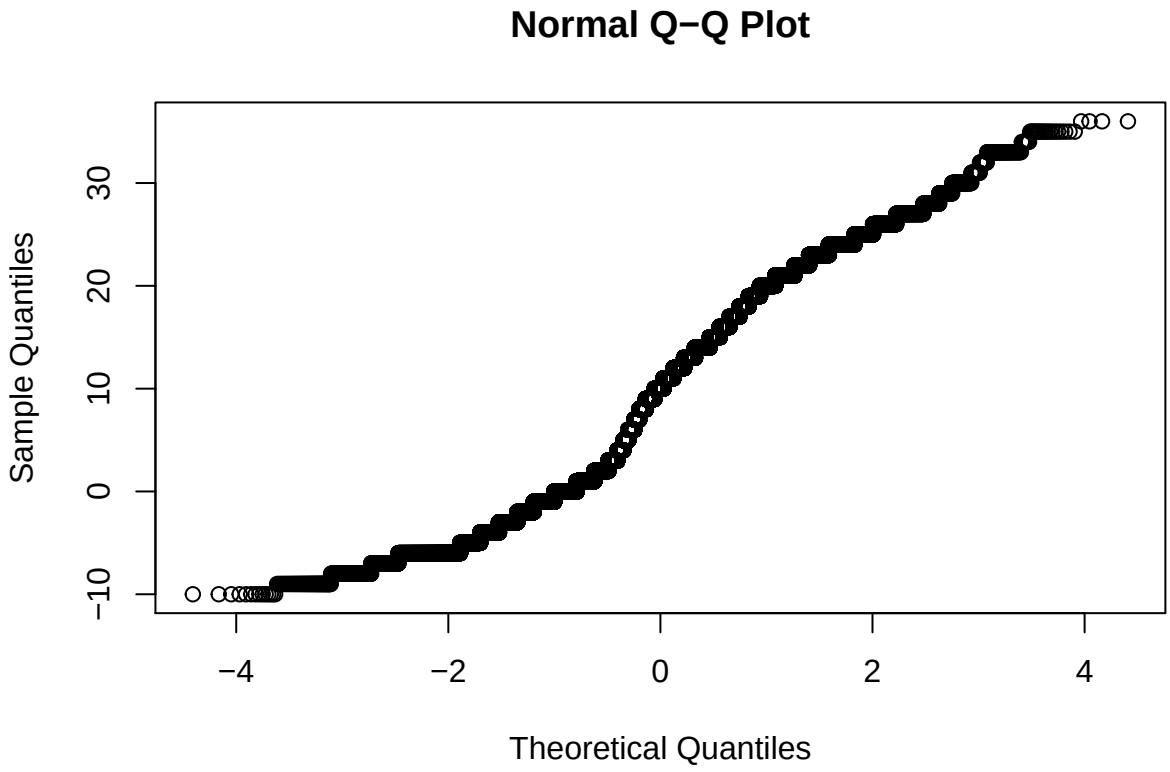
```
food_lim <- left_join(food_lim, brands_lookup, by='brands') %>%
  mutate(brands = as.factor(ifelse(is.na(clean_brand), 'Other', clean_brand))) %>%
  select(-count,-clean_brand)
```

EDA Plots

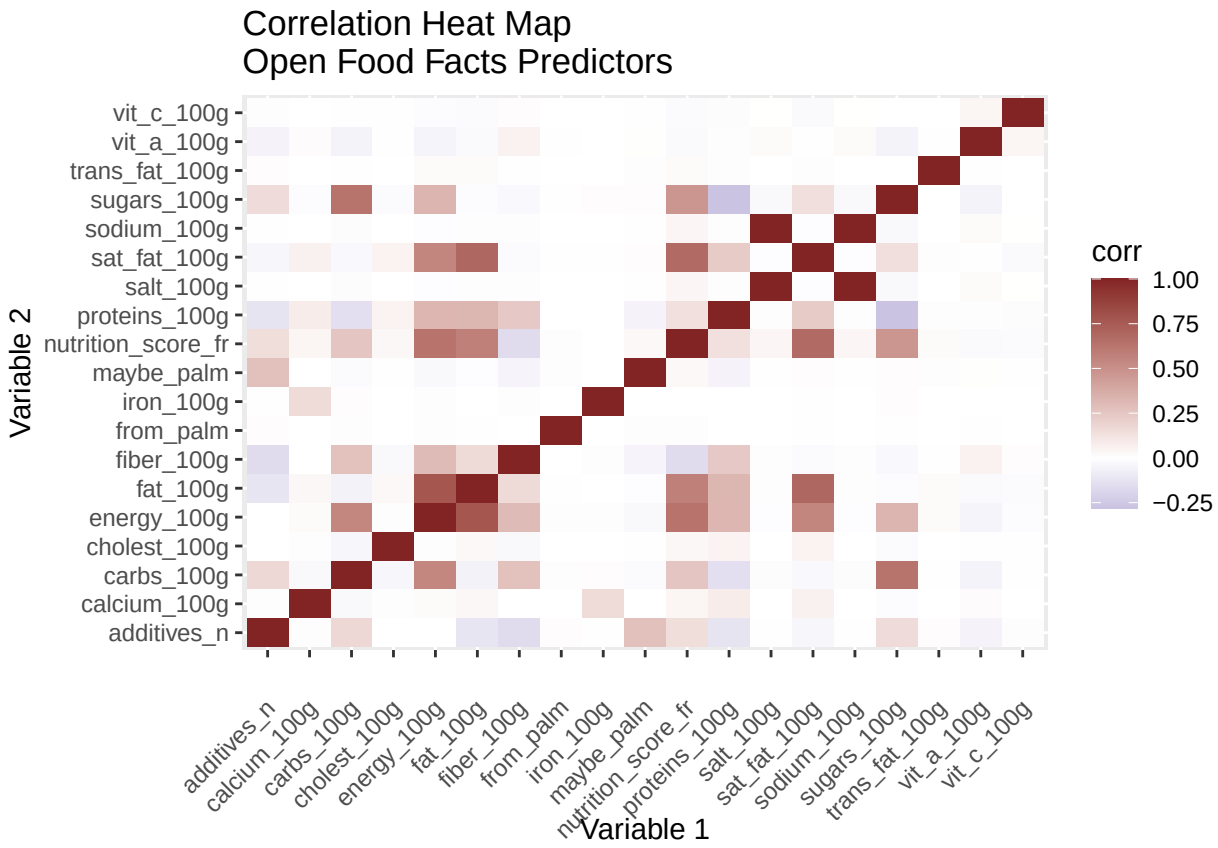
```
attach(food_lim)
hist(nutrition_score_fr,probability = T,main =
         'Histogram of Nutrition Score with Normal Density Overlaid')
x <- seq(min(nutrition_score_fr),max(nutrition_score_fr),0.5)
lines(x,dnorm(x,mean(nutrition_score_fr),sd(nutrition_score_fr)))
```




```
#Doesn't seem to match normal distribution
qqnorm(nutrition_score_fr)
```

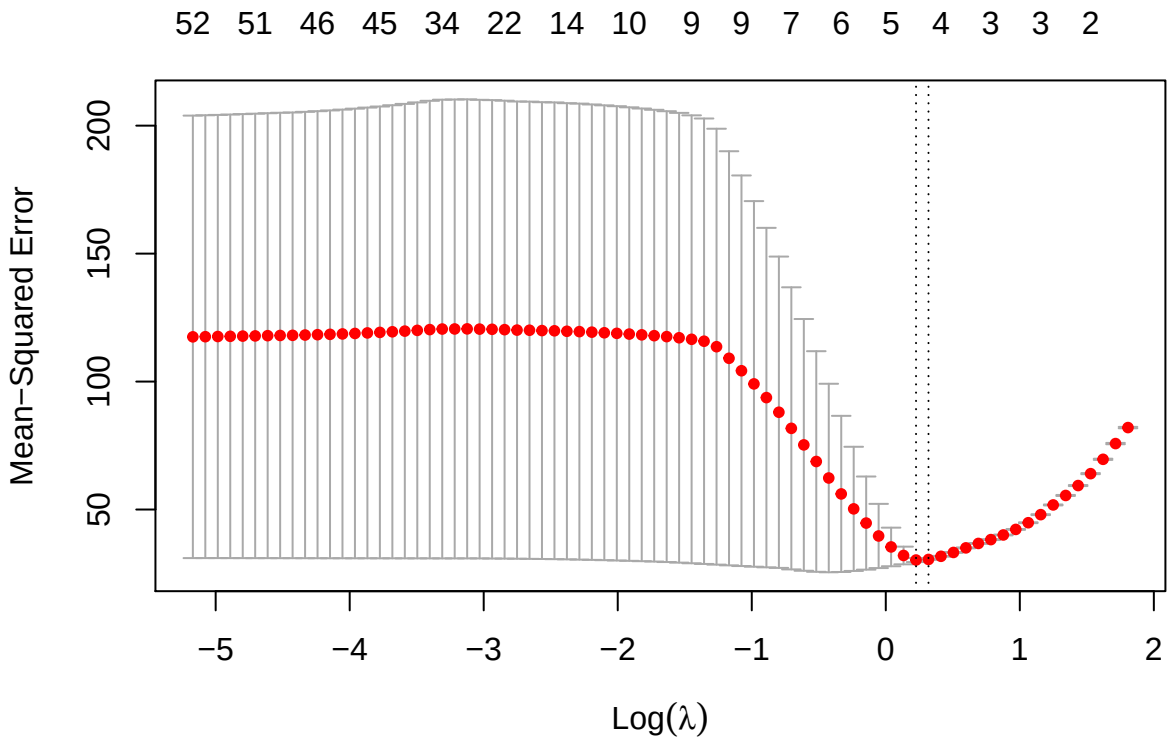



```
unique_vars <- unique(select(food_lim, -country, -brands) %>% colnames())
cor_matrix <- data.frame(cor(select(food_lim, -country, -brands))) %>%
  gather() %>%
  mutate(var2 = rep(unique_vars, length(unique_vars))) %>%
  select(var1 = key, var2, corr = value)

ggplot(cor_matrix, aes(var1,var2,fill=corr)) + geom_tile() +
    theme(axis.text.x = element_text(angle = 45, hjust=1,vjust=0.75)) +
    ggtitle('Correlation Heat Map \nOpen Food Facts Predictors') +
  scale_fill_gradient2(low = scales::muted('blue'),
                       mid = 'white',
                       high = scales::muted('red')) +
  xlab('Variable 1') + ylab('Variable 2')
```



Variable Selection via LASSO

```
food_lim <- dplyr::select(food_lim, -country) #Excluding country due to most being US
set.seed(443)
x <- model.matrix(nutrition_score_fr~., data=food_lim)[,-1]
y <- food_lim$nutrition_score_fr
train <- sample(1:nrow(x),2*nrow(x)/3)
y_test <- y[-train]
cv_feature <- cv.glmnet(x[train,], y[train], alpha = 1)
plot(cv_feature)
```




```
cat('Best Lambda =', cv_feature$lambda.min)
```

```
## Best Lambda = 1.253259
```

```
cat('1 SE Lambda =', cv_feature$lambda.1se,'\n')
```

```
## 1 SE Lambda = 1.37545
```

```
coef(cv_feature)
```

```
## 57 x 1 sparse Matrix of class "dgCMatrix"
##                                      1
## (Intercept)                2.69276750
## brandsAhold                .
## brandsBig Y                .
## brandsBrookshire's         .
## brandsClover Valley        .
## brandsCrystal Farms        .
## brandsDel Monte            .
## brandsDole                 .
## brandsFamily Gourmet       .
## brandsFood Club            .
## brandsFood Lion            .
## brandsFresh & Easy         .
## brandsGiant                .
## brandsGiant Eagle          .
## brandsGreat Value          .
## brandsHarris Teeter        .
## brandsHormel               .
## brandsHy-Vee               .
## brandsJohnvince Foods      .
## brandsKey Food             .
```

```
## brandsKroger               .
## brandsLowes Foods          .
## brandsMarket Pantry        .
## brandsMeijer               .
## brandsOther                .
## brandsPrivate Selection    .
## brandsRoundy's             .
## brandsShurfine             .
## brandsSignature Kitchens   .
## brandsSouthern Home        .
## brandsSpartan              .
## brandsTarget Stores        .
## brandsTops                 .
## brandsTrader Joe's         .
## brandsWegmans              .
## brandsWestern Family       .
## brandsWild Harvest         .
## brandsWinco Foods          .
## brandsWinn-Dixie           .
## energy_100g                0.00305504
## sat_fat_100g               0.46843188
## sugars_100g                0.08442593
## proteins_100g              .
## salt_100g                  .
## sodium_100g                .
## fat_100g                   .
## carbs_100g                 .
## additives_n                .
## from_palm                  .
## maybe_palm                 .
## fiber_100g                -0.15752128
## cholest_100g               .
## calcium_100g               .
## trans_fat_100g             .
## iron_100g                  .
## vit_a_100g                 .
## vit_c_100g                 .
```

```
rm(x) #Memory reasons
```

Linear Regression

```
lm.fit <- lm(nutrition_score_fr ~ energy_100g + sat_fat_100g + sugars_100g + fiber_100g,
             data = food_lim, subset = train)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = nutrition_score_fr ~ energy_100g + sat_fat_100g +
##     sugars_100g + fiber_100g, data = food_lim, subset = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -49.617  -3.069  -0.272   3.103  50.148
```

```
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.112e-01  3.800e-02   16.08   <2e-16 ***
## energy_100g   4.887e-03  3.579e-05  136.52   <2e-16 ***
## sat_fat_100g  5.386e-01  3.497e-03  153.99   <2e-16 ***
## sugars_100g   1.257e-01  1.070e-03  117.48   <2e-16 ***
## fiber_100g   -5.587e-01  4.873e-03 -114.65   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.943 on 64217 degrees of freedom
## Multiple R-squared:  0.7024, Adjusted R-squared:  0.7024
## F-statistic: 3.789e+04 on 4 and 64217 DF,  p-value: < 2.2e-16
```

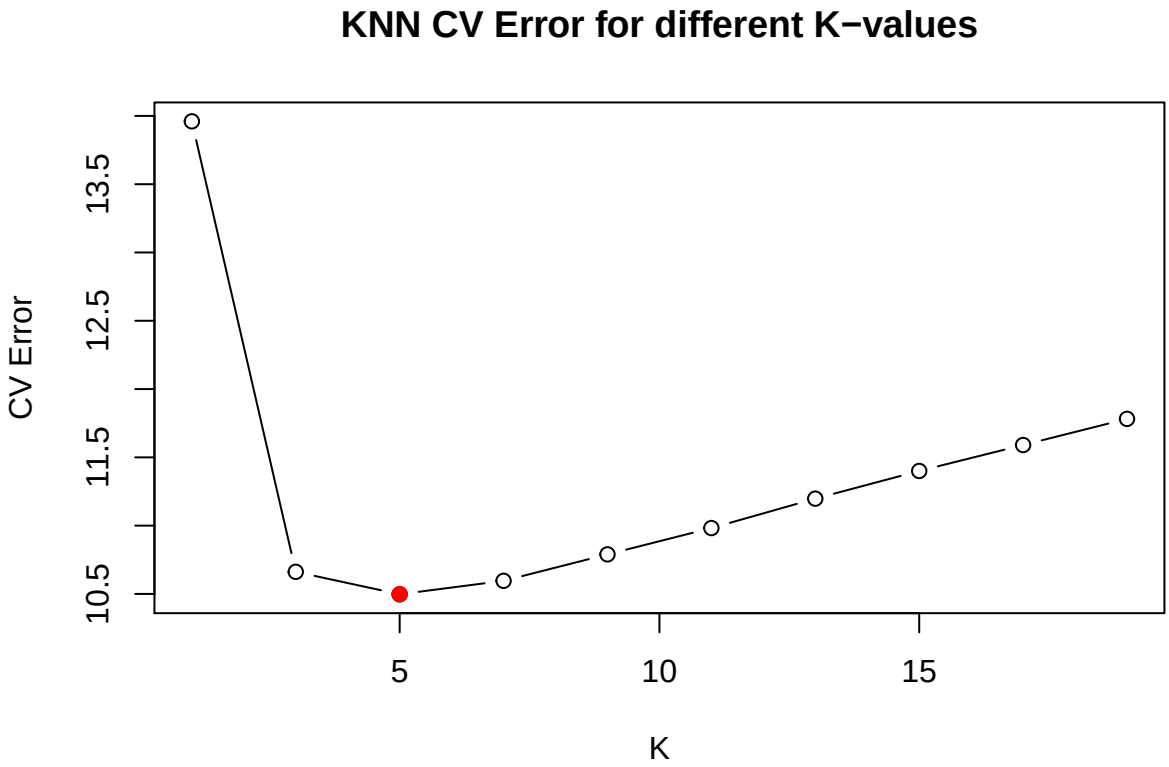
```
lm.preds <- predict(lm.fit, newdata = food_lim[-train,])
```

KNN Regression

```
X_train <- food_lim[train,] %>% select(energy_100g, sat_fat_100g, sugars_100g, fiber_100g)
X_test <- food_lim[-train,] %>% select(energy_100g, sat_fat_100g, sugars_100g, fiber_100g)
y_train <- y[train]
#Derive own 10-fold CV to select optimal k
set.seed(443)
folds <- sample(rep(1:10, length=nrow(X_train))) #Check folds are of equal size
table(folds)
```

```
## folds
##    1    2    3    4    5    6    7    8    9   10
## 6423 6423 6422 6422 6422 6422 6422 6422 6422 6422
```

```
cv_errors <- matrix(0,10,10)
for (i in 1:10){
  for (j in seq(1,20,2)){
    knn_pred <- knn.reg(X_train[folds!=i,], X_train[folds==i,],
                        y_train[folds!=i], k=j)$pred
    cv_errors[i,j%/%2+1] <- mean((knn_pred-y_train[folds==i])^2,na.rm=T)
  }
}
MSE <- colMeans(cv_errors)
plot(seq(1,20,2), MSE,xlab = 'K',ylab='CV Error',type = 'both',
     main = 'KNN CV Error for different K-values')
points(5,min(MSE),col='red',pch=19)
```




```
knn.preds <- knn.reg(X_train, X_test, y_train, k=5)$pred
rm(X_train, X_test, y_train) #Memory reasons
```

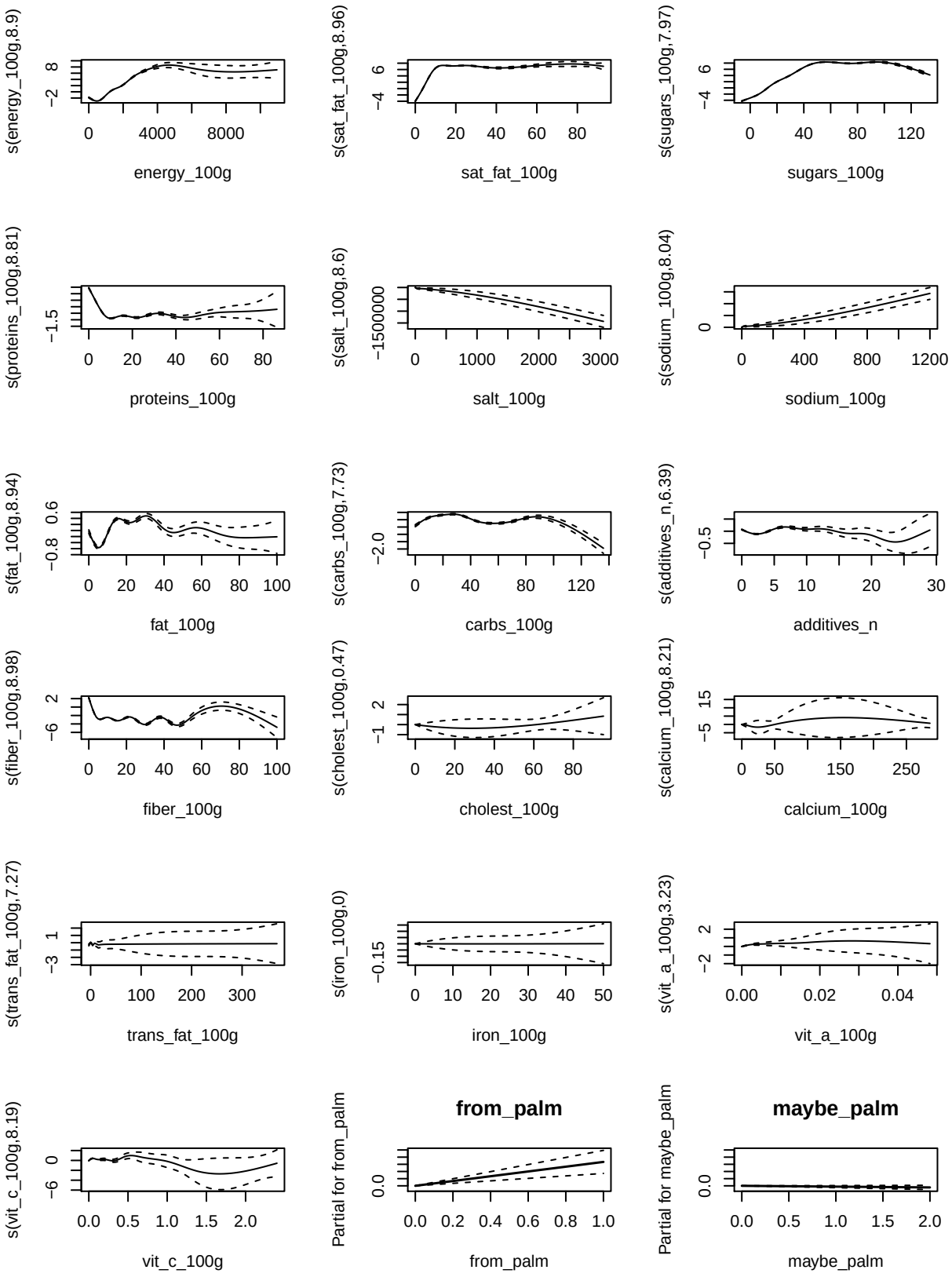
GAM

```
#Individually smooth each numeric variable and use select = TRUE for group lasso
set.seed(443)
gam.fit <- bam(nutrition_score_fr ~ from_palm + maybe_palm + s(energy_100g) +
                   s(sat_fat_100g) + s(sugars_100g) + s(proteins_100g) +
                   s(salt_100g) + s(sodium_100g) + s(fat_100g) + s(carbs_100g) +
                   s(additives_n) + s(fiber_100g) + s(cholest_100g) +
                   s(calcium_100g) + s(trans_fat_100g) + s(iron_100g) +
                   s(vit_a_100g) + s(vit_c_100g), select = TRUE,
               method = 'REML', data = food_lim[train,])
summary(gam.fit)
```

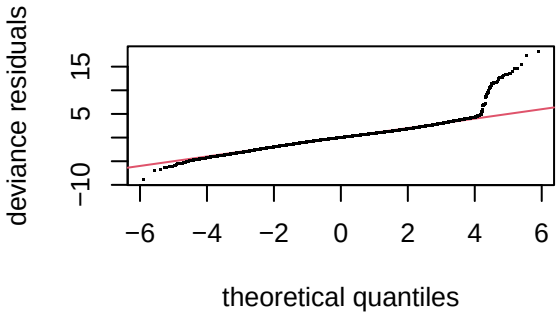
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## nutrition_score_fr ~ from_palm + maybe_palm + s(energy_100g) +
##     s(sat_fat_100g) + s(sugars_100g) + s(proteins_100g) + s(salt_100g) +
##     s(sodium_100g) + s(fat_100g) + s(carbs_100g) + s(additives_n) +
##     s(fiber_100g) + s(cholest_100g) + s(calcium_100g) + s(trans_fat_100g) +
##     s(iron_100g) + s(vit_a_100g) + s(vit_c_100g)
##
## Parametric coefficients:
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept)  9.465062   0.005506 1719.053  < 2e-16 ***
```

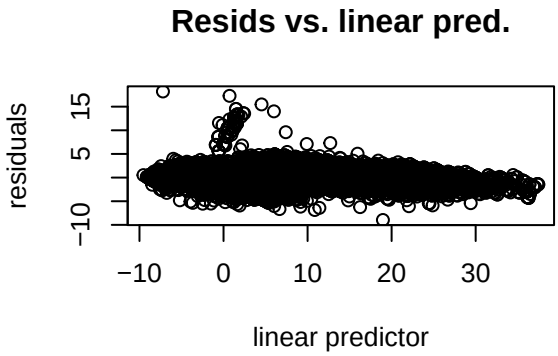
```
## from_palm    1.667076   0.401720    4.150 3.33e-05 ***
## maybe_palm  -0.056087   0.037122   -1.511    0.131
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                         edf Ref.df         F  p-value
## s(energy_100g)    8.8997658      9   968.205  < 2e-16 ***
## s(sat_fat_100g)   8.9587367      9 15232.858  < 2e-16 ***
## s(sugars_100g)    7.9673035      9 20641.797  < 2e-16 ***
## s(proteins_100g)  8.8056623      9  1022.828  < 2e-16 ***
## s(salt_100g)      8.6011824      9  1363.635  < 2e-16 ***
## s(sodium_100g)    8.0358600      9  1341.098  < 2e-16 ***
## s(fat_100g)       8.9417797      9   155.233  < 2e-16 ***
## s(carbs_100g)     7.7331904      9   106.992  < 2e-16 ***
## s(additives_n)    6.3892700      9    26.312  < 2e-16 ***
## s(fiber_100g)     8.9805089      9  7765.232  < 2e-16 ***
## s(cholest_100g)   0.4740900      9     0.097    0.175
## s(calcium_100g)   8.2089211      9    50.234  < 2e-16 ***
## s(trans_fat_100g) 7.2731967      9     5.700 4.21e-09 ***
## s(iron_100g)      0.0009598      9     0.000    0.448
## s(vit_a_100g)     3.2322010      9     4.018 6.84e-09 ***
## s(vit_c_100g)     8.1886954      9    34.015  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.977   Deviance explained = 97.7%
## -REML = 1.1155e+05  Scale est. = 1.863     n = 64222
```

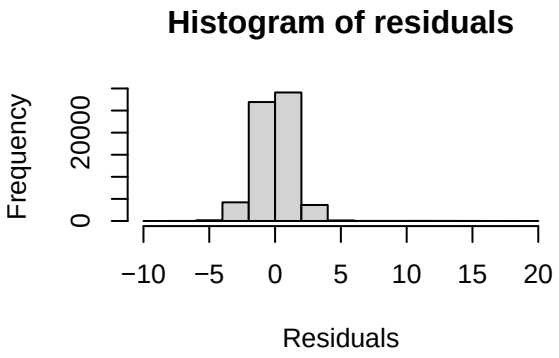
```
gam.preds <- predict(gam.fit, newdata = food_lim[-train,])
plot.gam(gam.fit, scale = 0, pages = 2, all.terms = TRUE)
```

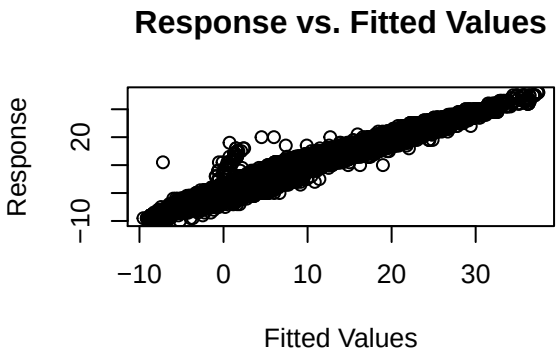



```
gam.check(gam.fit)
```










```
##
## Method: REML   Optimizer: outer newton
## full convergence after 63 iterations.
## Gradient range [-0.0006365422,0.01105392]
## (score 111551.3 & scale 1.863048).
## eigenvalue range [-0.01103198,32109.51].
## Model rank =  147 / 147
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                        k'     edf k-index p-value
## s(energy_100g)    9.00000 8.89977    0.92  <2e-16 ***
## s(sat_fat_100g)   9.00000 8.95874    0.96  <2e-16 ***
## s(sugars_100g)    9.00000 7.96730    0.92  <2e-16 ***
## s(proteins_100g)  9.00000 8.80566    0.95  <2e-16 ***
## s(salt_100g)      9.00000 8.60118    0.81  <2e-16 ***
## s(sodium_100g)    9.00000 8.03586    0.81  <2e-16 ***
## s(fat_100g)       9.00000 8.94178    0.93  <2e-16 ***
## s(carbs_100g)     9.00000 7.73319    0.94  <2e-16 ***
## s(additives_n)    9.00000 6.38927    1.04   1.000
## s(fiber_100g)     9.00000 8.98051    0.95  <2e-16 ***
## s(cholest_100g)   9.00000 0.47409    0.99   0.220
## s(calcium_100g)   9.00000 8.20892    1.00   0.460
## s(trans_fat_100g) 9.00000 7.27320    0.99   0.280
## s(iron_100g)      9.00000 0.00096    0.97   0.025 *
## s(vit_a_100g)     9.00000 3.23220    0.97   0.015 *
## s(vit_c_100g)     9.00000 8.18869    0.99   0.140
## ---
```

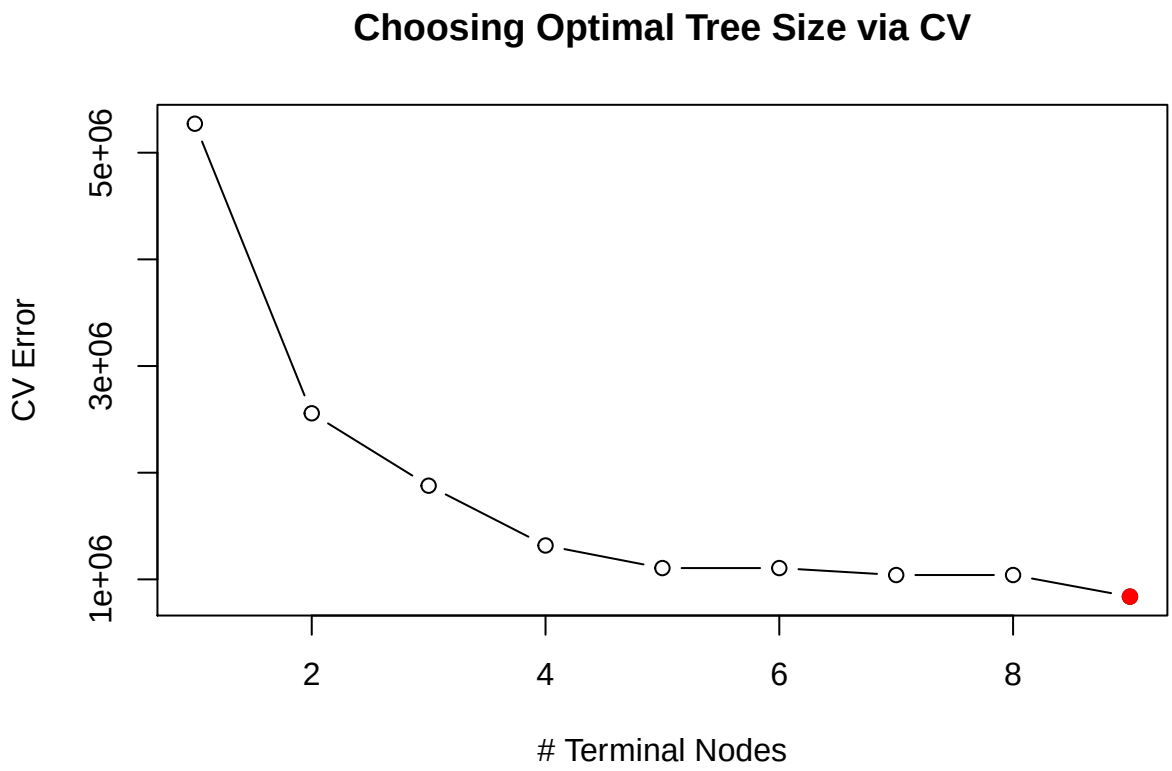
```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Regression Tree

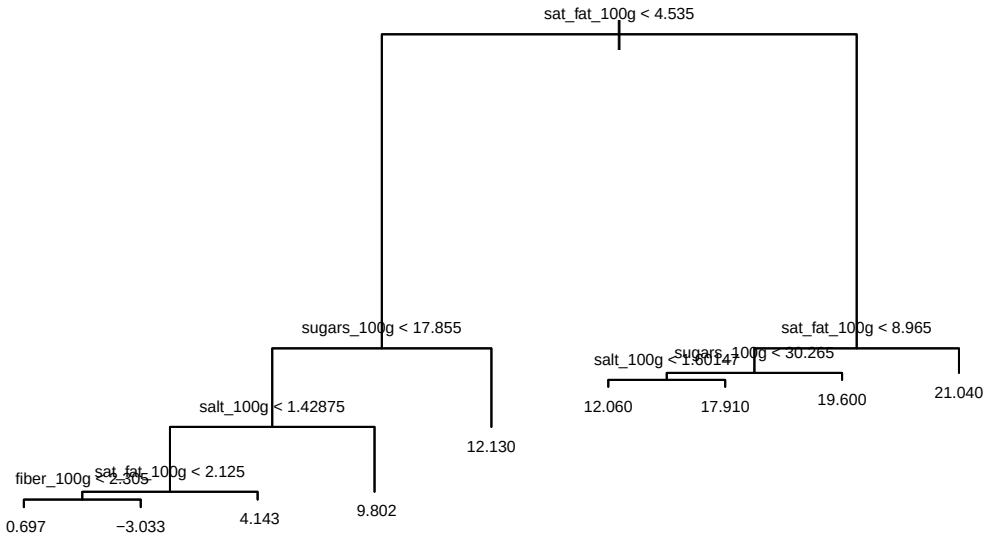
```
set.seed(443)
tree.fit <- tree(nutrition_score_fr~.-brands, data=food_lim, subset=train)
cv.tree.fit <- cv.tree(tree.fit)
cv.tree.fit
```

```
## $size
## [1] 9 8 7 6 5 4 3 2 1
##
## $dev
## [1]  838327.4 1040747.7 1040747.7 1105895.2 1105895.2 1317840.2 1878348.5
## [8] 2557031.6 5271665.1
##
## $k
## [1]       -Inf   60435.79   60467.51   65755.90   67721.32  211985.86  560735.67
## [8]  678797.90 2714628.80
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv.tree.fit$size, cv.tree.fit$dev, type="b",xlab = '# Terminal Nodes',
     ylab = 'CV Error', main = 'Choosing Optimal Tree Size via CV')
points(9,min(cv.tree.fit$dev),col='red',pch=19)
```



```
prune.fit <- prune.tree(tree.fit, best=9)
plot(prune.fit)
text(prune.fit, pretty=0, cex=0.5)
```

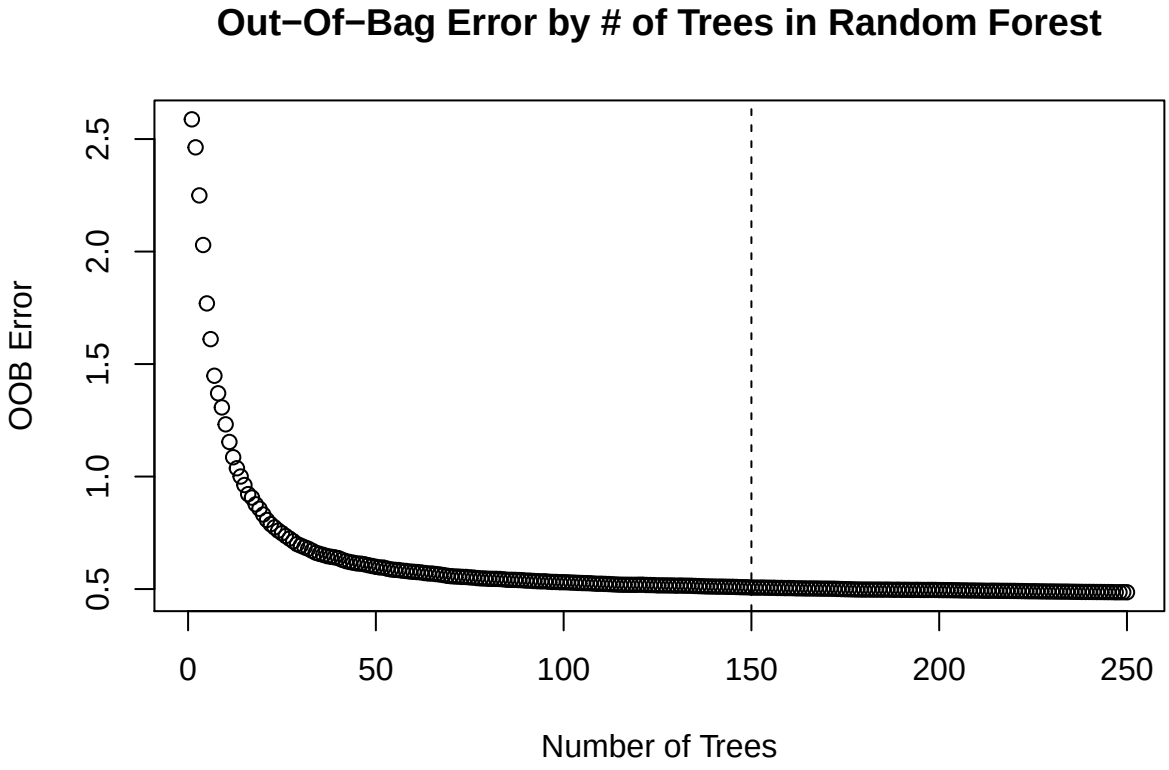



```
tree.preds <- predict(prune.fit, newdata = food_lim[-train,])
```

```
set.seed(443)
rf.tune <- randomForest(nutrition_score_fr~.-brands, data=food_lim,
                        subset=train, mtry=6,
                        importance=TRUE, ntree = 250)
rf.tune
```

```
##
## Call:
##  randomForest(formula = nutrition_score_fr ~ . - brands, data = food_lim,      mtry = 6, importance =
##                Type of random forest: regression
##                      Number of trees: 250
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 0.4858816
##                     % Var explained: 99.41
```

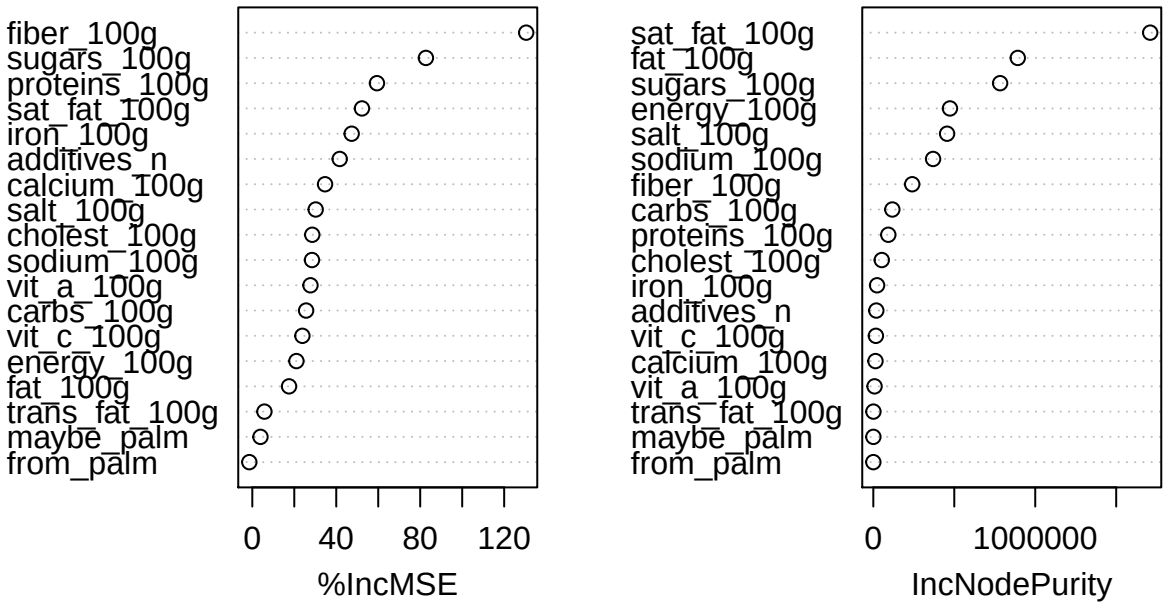
```
plot(rf.tune$mse, xlab = 'Number of Trees', ylab = 'OOB Error',
     main = 'Out-Of-Bag Error by # of Trees in Random Forest')
abline(v=150, lty='dashed')
```




```
#Refit tree with B = 150
rm(rf.tune) #Memory reasons
rf.fit <- randomForest(nutrition_score_fr~.-brands, data=food_lim, subset=train, mtry=6,
                       importance=TRUE, ntree = 150)
importance(rf.fit)
```

```
##                   %IncMSE IncNodePurity
## energy_100g     21.012316  4.732797e+05
## sat_fat_100g    52.239202  1.710013e+06
## sugars_100g     82.717935  7.831220e+05
## proteins_100g   59.386732  9.254199e+04
## salt_100g       30.199264  4.555839e+05
## sodium_100g     28.425740  3.690246e+05
## fat_100g        17.507917  8.921553e+05
## carbs_100g      25.632616  1.173568e+05
## additives_n     41.648320  1.843511e+04
## from_palm       -1.421467  5.074381e+00
## maybe_palm       3.878550  2.787136e+02
## fiber_100g     130.497510  2.400477e+05
## cholest_100g    28.558314  5.215665e+04
## calcium_100g    34.654648  1.330166e+04
## trans_fat_100g   5.768172  5.714309e+02
## iron_100g       47.344281  2.314076e+04
## vit_a_100g      27.713567  7.537509e+03
## vit_c_100g      23.857315  1.568834e+04
```

```
varImpPlot(rf.fit)
```



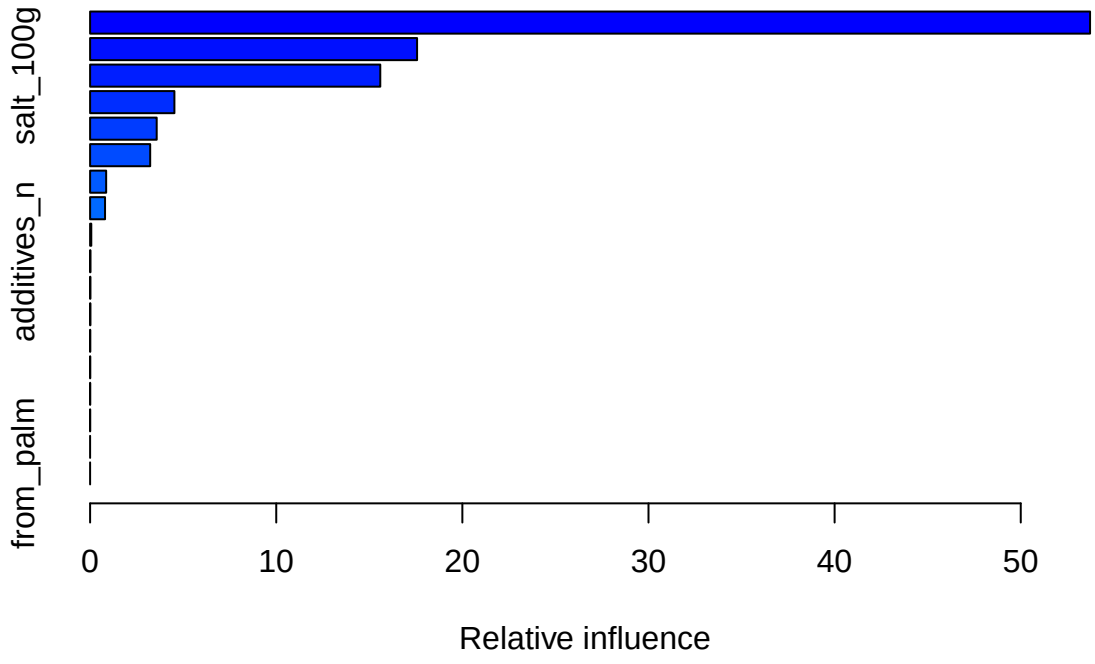

```
rf.preds <- predict(rf.fit, newdata = food_lim[-train,])
```

```
#Grid of values for d and lambda
param_grid <- expand.grid(shrinkage = c(.001, 0.005, 0.01),
                          interaction.depth = 1:5)

set.seed(443)
for(i in 1:nrow(param_grid)){
  boost.tune <- gbm(nutrition_score_fr~.-brands, data = food_lim[train,],
                    distribution = 'gaussian', n.trees = 5000,
                    interaction.depth = param_grid$interaction.depth[i],
                    shrinkage = param_grid$shrinkage[i],
                    n.cores = 4,
                    train.fraction = 0.7)

  param_grid$opt_trees[i] <- which.min(boost.tune$valid.error)
  param_grid$opt_MSE[i] <- sqrt(min(boost.tune$valid.error))
}
param_grid
```

```
##    shrinkage interaction.depth opt_trees   opt_MSE
## 1      0.001                 1      5000 3.3369623
## 2      0.005                 1      5000 1.3594449
## 3      0.010                 1      5000 1.2501295
## 4      0.001                 2      5000 2.0277766
## 5      0.005                 2      5000 1.0579561
## 6      0.010                 2      5000 0.9301868
## 7      0.001                 3      5000 1.7691130
## 8      0.005                 3      5000 0.9886520
```

```
## 9      0.010                3      5000 0.8422313
## 10     0.001                4      5000 1.5934369
## 11     0.005                4      5000 0.8887423
## 12     0.010                4      5000 0.7600765
## 13     0.001                5      5000 1.4463162
## 14     0.005                5      5000 0.8252681
## 15     0.010                5      5000 0.7088986
```

```
#Refit tuned model to entire training data
boost.fit <- gbm(nutrition_score_fr~.-brands, data = food_lim[train,],
                 distribution = 'gaussian', n.trees = 5000, interaction.depth = 5,
                 shrinkage = 0.01, n.cores = 4)
summary(boost.fit)
```




```
##                              var      rel.inf
## sat_fat_100g         sat_fat_100g 53.723931435
## sugars_100g           sugars_100g 17.567154367
## salt_100g               salt_100g 15.590245738
## fiber_100g             fiber_100g  4.529137002
## fat_100g                 fat_100g  3.577309630
## energy_100g           energy_100g  3.227749486
## proteins_100g       proteins_100g  0.862666479
## sodium_100g           sodium_100g  0.801857191
## carbs_100g             carbs_100g  0.062552028
## additives_n           additives_n  0.016347838
## cholest_100g         cholest_100g  0.013103849
## calcium_100g         calcium_100g  0.012387933
## vit_c_100g             vit_c_100g  0.006289467
## iron_100g               iron_100g  0.003972443
## vit_a_100g             vit_a_100g  0.003396562
## trans_fat_100g   trans_fat_100g  0.001483813
## maybe_palm             maybe_palm  0.000414741
## from_palm               from_palm  0.000000000
```

```r
boost.preds <- predict(boost.fit, newdata = food_lim[-train,])
```

```r
models <- c('Linear Regression', 'KNN Regression', 'GAM',
            'Decision Tree', 'Random Forest', 'Boosting')
preds <- list(lm.preds, knn.preds, gam.preds, tree.preds, rf.preds, boost.preds)
names(preds) <- models
(RMSEs <- lapply(preds, function(x) sqrt(mean((x - y_test)^2))))
```

```
## $`Linear Regression`
## [1] 4.924694
##
## $`KNN Regression`
## [1] 3.23773
##
## $GAM
## [1] 1.761583
##
## $`Decision Tree`
## [1] 3.62616
##
## $`Random Forest`
## [1] 0.6890062
##
## $Boosting
## [1] 0.7081739
```

```r
(MAEs <- lapply(preds, function(x) mean(abs(x - y_test))))
```

```
## $`Linear Regression`
## [1] 3.776618
##
## $`KNN Regression`
## [1] 2.126891
##
## $GAM
## [1] 1.053227
##
## $`Decision Tree`
## [1] 2.76173
##
## $`Random Forest`
## [1] 0.3546263
##
## $Boosting
## [1] 0.4145442
```