

## ▼ Project Introduction

The main objective of this study is to predict the nutrition score of several food items by utilizing the nutrient makeup provided in the Open Food Facts dataset, followed by testing against the nutrition scores and grades already provided. Since not all food items in the dataset obtain a nutritional score, these can be used as a testing set, while the ones with scores can be used to actually train a model in predicting nutritional scores based off the nutritional makeup of different foods. These nutrition scores will also be compared to health outcomes for the different regions those foods are associated with, which will help in obtaining a better understanding for any correlation between nutrition score and health outcomes, which therefore also ties back and relates to the nutritional makeup of such foods.

This is a supervised learning problem, so we will fit a regression model, allowing us to extract feature importance and understand how these nutritional scores change as our inputs change. Predicting the nutrition score allows for unique insights into which nutrients are more impactful for overall nutrition and allows us to predict nutrition scores for unscored data items.

Below, the steps taken will be displayed, including plenty of data preprocessing, data analysis (which primarily dealt with handling an overabundance of null values), and finally, the development and fine tuning of a machine learning algorithm.

## ▼ DATA PREPROCESSING AND EXPLORATORY DATA ANALYSIS

*\*In this section, both datasets (open food facts and health and nutrition population outcomes were downloaded, cleaned, irrelevant columns were dropped, and data was explored with some figures presented below. Ultimately columns that would be relevant features for modeling were determined. There was not found to be a correlation between nutrition scores and health outcomes.\**

## ▼ Importing Datasets

---

**The first step was to load all data and libraries to be used**

```
1 # Sklearn and Pandas Setup
2 import json
3 import glob
4 import pandas as pd
5 import numpy as np
6 import datetime as dt
7 import re
8 import os
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11 from matplotlib import cm
12 from google.colab import drive
13 from sklearn.model_selection import train_test_split
```

```
1 # Run this cell to mount your drive (you will be prompted to sign in)
2 from google.colab import drive
3 drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
1 !pip install kaggle
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kaggle in /usr/local/lib/python3.8/dist-packages (1.5.12)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.8/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from kaggle) (2.23.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.8/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.8/dist-packages (from kaggle) (2022.9.24)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from kaggle) (4.64.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.8/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.8/dist-packages (from kaggle) (7.0.0)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.8/dist-packages (from python-slugify->kaggle) (1.3)
```

```
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->kaggle) (3.7.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->kaggle) (2.10)
```

```
1 !mkdir ~/.kaggle

1 # Read the uploaded kaggle.json file
2 !cp /content/drive/MyDrive/kaggle.json ~/.kaggle/

1 !!kaggle datasets download -d theworldbank/health-nutrition-and-population-statistics

['Downloading health-nutrition-and-population-statistics.zip to /content',
'',
' 0% 0.00/14.0M [00:00<?, ?B/s]',
' 36% 5.00M/14.0M [00:00<00:00, 32.6MB/s]',
'',
'100% 14.0M/14.0M [00:00<00:00, 68.6MB/s]']

1 # Unzip folder in Colab content folder
2 !unzip /content/health-nutrition-and-population-statistics.zip
3
4

Archive: /content/health-nutrition-and-population-statistics.zip
  inflating: data.csv

1 # TO-DO: Read the csv file and save it to a dataframe
2 HNP_df = pd.read_csv('data.csv')
3
4 # Check out the first five rows
5 HNP_df.head()

1 !!kaggle datasets download -d openfoodfacts/world-food-facts

1 !unzip /content/world-food-facts.zip

1 OFF_df=pd.read_csv("en.openfoodfacts.org.products.tsv",sep='\t')
2
3 OFF_df.head()
```

## ▼ Open Food Facts EDA

**Next, we started to clean the Open Food Facts dataset in more depth. This included learning about what the columns were and whether they were relevant, and which ones to clean and drop. Exploratory figures shown too**

```
1 print(OFF_df.shape)

(356027, 163)
```

Shape: (356027 rows, 163 columns)

**List of All Columns-** will be going through these in categories to explore and dropping irrelevant ones as we do so

```
1 columns_list_OFF=OFF_df.columns.to_list()
2 print(columns_list_OFF)

['code', 'url', 'creator', 'created_t', 'created_datetime', 'last_modified_t', 'last_modified_datetime', 'product_na
```

### Date/Time Related Columns

```

1 #all date related columns
2
3 date_related=OFF_df[["created_t","created_datetime","last_modified_t","last_modified_datetime"]]
4 date_related.head(5)
5

```

	created_t	created_datetime	last_modified_t	last_modified_datetime
0	1474103866	2016-09-17T09:17:46Z	1474103893	2016-09-17T09:18:13Z
1	1489069957	2017-03-09T14:32:37Z	1489069957	2017-03-09T14:32:37Z
2	1489069957	2017-03-09T14:32:37Z	1489069957	2017-03-09T14:32:37Z
3	1489055731	2017-03-09T10:35:31Z	1489055731	2017-03-09T10:35:31Z
4	1489055653	2017-03-09T10:34:13Z	1489055653	2017-03-09T10:34:13Z

Based on the date related-columns: we should use "last\_modified\_datetime" as date.

```

1 drop_columns=["created_t","created_datetime","last_modified_t"]
2 OFF_df.drop(drop_columns, axis=1, inplace=True)

```

### Location Related Columns

```

1 #location_related_columns
2
3 location_related=OFF_df[["countries","countries_tags","countries_en","stores","purchase_places","cities_tags","cities",
4
5 #find column values without nulls
6 location_related_nonulls = pd.DataFrame()
7
8 for column in location_related.columns:
9     location_related_nonulls[[column]]=location_related[[column]].dropna().reset_index(drop=True)
10
11 location_related_nonulls.head(5)

```

	countries	countries_tags	countries_en	stores	purchase_places	cities_tags	cities	first_packaging_code_geo	ma
0	en:FR	en:france	France	Costco	Roissy,France	brignemont-haute-garonne-france	b	43.783333,0.983333	
1	US	en:united-states	United States	Costco	Roissy,France	saint-didier-au-mont-d-or-rhone-france	b	45.8,4.8	
2	US	en:united-states	United States	Waitrose	Brossard Québec	donzere-drome-france	c	44.45,4.716667	
3	US	en:united-states	United States	Costco	Brent,UK	molay-littry-calvados-france	c	49.25,-0.883333	
4	US	en:united-states	United States	Costco	Brossard Québec	clecycalvados-france	a	48.916667,-0.483333	

After going through each of the location-related columns, the ones relevant to our analysis are the countries and origins. The other columns focus on cities, stores, and manufacturing which is irrelevant for our analysis.

```

1 drop_columns=["stores","purchase_places","cities_tags","cities","first_packaging_code_geo","manufacturing_places","manu
2 OFF_df.drop(drop_columns, axis=1,inplace=True)
3

```

```

1 OFF_df["origins"].isna().sum()

```

330977

Origins column could relate to country, but high amount of nulls and not relevant to modeling in future-drop

```
1 drop_columns=["origins","origins_tags"]
2 OFF_df.drop(drop_columns, axis=1,inplace=True)
3
```

Which Country Column Should be Used? Drop Other Columns

```
1 #make sure the countries column in OFF and HNP match format for most part
2 print(OFF_df[["countries","countries_tags","countries_en"]])
3 HNP_countries=HNP_df[['Country Name', 'Country Code']]
4 print(HNP_countries[HNP_countries["Country Name"]=="United States"])
```

	countries	countries_tags	countries_en
0	en:FR	en:france	France
1	US	en:united-states	United States
2	US	en:united-states	United States
3	US	en:united-states	United States
4	US	en:united-states	United States
...	...	...	...
356022	US	en:united-states	United States
356023	China	en:china	China
356024	France	en:france	France
356025	en:FR	en:france	France
356026	US	en:united-states	United States

```
[356027 rows x 3 columns]
   Country Name Country Code
85215  United States      USA
85216  United States      USA
85217  United States      USA
85218  United States      USA
85219  United States      USA
...
85555  United States      USA
85556  United States      USA
85557  United States      USA
85558  United States      USA
85559  United States      USA
```

```
[345 rows x 2 columns]
```

countries and countries\_tags is not the column we want- will use countries\_en and drop other two since countries\_en matches the syntax of country names in the health and nutrition population dataset.

```
1 drop_columns=["countries","countries_tags"]
2 OFF_df.drop(drop_columns, axis=1,inplace=True)
```

```
1 OFF_df["countries_en"].unique()
```

Countries column appears that some instances have multiple countries listed. To clean, use explode to separate countries into multiple rows (repeat instances).

```
1 #first drop any rows without countries
2 OFF_df.dropna(subset=["countries_en"],inplace=True)
3
4
5
6
7
1 from ast import literal_eval
2 #use explode to separate countries into multiple rows since some instances have multiple countries
3 country_list=OFF_df["countries_en"].apply(lambda x:x.split(","))
4 OFF_df["countries_final"]=country_list
5 OFF_df["countries_final"]=OFF_df["countries_final"].to_list()
6 OFF_df["countries_final"] = OFF_df["countries_final"].apply(lambda x: literal_eval(str(x)))
7 exploded_df_OFF = OFF_df.explode('countries_final', ignore_index=True)
```

Make sure the countries match the countries in the health and nutrition population dataset.

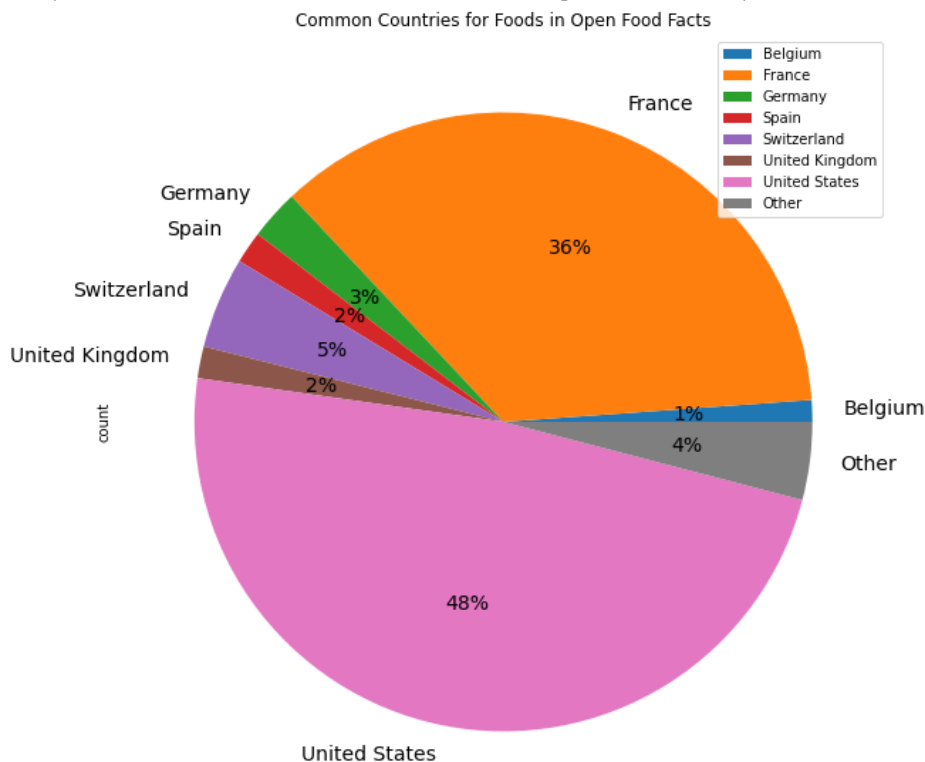
```
1 list_HNP=HNP_df["Country Name"].to_list()
2 matched_countries_OFF=exploded_df_OFF[exploded_df_OFF["countries_final"].isin(list_HNP)]

1 #drop the countries_en column
2 OFF_df=matched_countries_OFF.drop("countries_en", axis=1)
```

What are the Most Common Countries?

```
1 country = OFF_df.groupby(['countries_final'])['countries_final'].count().reset_index(name="count") #groupby countries &
2 other_threshold=int(country["count"].sum()*0.01) #if less than one percent, group into other category
3 other_countries=country[country["count"]<other_threshold]
4
5 other_val=other_countries["count"].sum() #find count of "other" category
6
7 #add "other" back to countries list
8 common_countries=country[country["count"]>=other_threshold]
9 common_countries
10 df2={"countries_final":"Other","count":other_val}
11 common_countries = common_countries.append(df2, ignore_index = True)
12 CC=common_countries.set_index("countries_final")
13
14 #make pie chart
15 CC.plot(kind="pie",y="count", autopct='%0f%%', figsize=(10,10),textprops={'fontsize': 14})
16 plt.title("Common Countries for Foods in Open Food Facts")
17
18
19
```

Text(0.5, 1.0, 'Common Countries for Foods in Open Food Facts')



Food products in Open Food Facts were most commonly present in the United States and France

Status Related Columns

```
1 status_related_col=["states","states_tags","states_en"]
2 OFF_df[status_related_col].head(5)
```

States columns- have to do with whether it is completed or not- should look into how many rows are actually completed. 1068 different tags for states- each of them talk about what is "to-be-completed" and what is completed. This shows that many items on the list are still to be completed: especially based on the number of nulls in columns as shown below. Drop these columns.

```
1 drop_columns=status_related_col
2 OFF_df.drop(drop_columns, axis=1, inplace=True)
```

### What are the nutrition score related columns?

```
1 #grade/score related columns
2
3 grade_related=OFF_df[["nutrition_grade_uk","nutrition-score-uk_100g","nutrition_grade_fr","nutrition-score-fr_100g"]]
4
5 grade_related.head(5)
```

	<b>nutrition_grade_uk</b>	<b>nutrition-score-uk_100g</b>	<b>nutrition_grade_fr</b>	<b>nutrition-score-fr_100g</b>
0	NaN	NaN	NaN	NaN
1	NaN	14.0	d	14.0
2	NaN	0.0	b	0.0
3	NaN	12.0	d	12.0
4	NaN	NaN	NaN	NaN

### How are Nutrition Scores for France and UK Related?

```
1 import seaborn as sns
2 scores = OFF_df[['nutrition-score-uk_100g', 'nutrition-score-fr_100g']].dropna(how='any') #drop nulls
3 correlation = scores.corr()
4 ax = plt.axes()
5 sns.heatmap(correlation, annot=True, vmin=0, vmax=1,ax=ax) #plot heatmap of correlation
6 ax.set_title('Correlation Between UK and FR Nutrition Scores')
7
8 means = scores.mean()
9 stds = scores.std()
10
11
12 print("\n\nMeans:")
13 print(means)
14 print("\n\nStandard Deviations:")
15 print(stds)
```

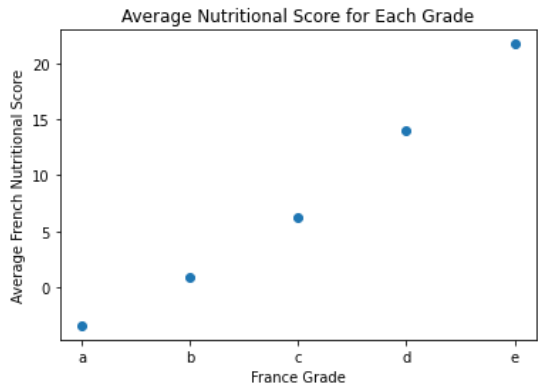
```
Means:
nutrition-score-uk_100g    9.010944
nutrition-score-fr_100g    9.204557
dtype: float64
```

Both the UK and France scores are highly correlated and have similar means and standard deviations. Therefore, don't need to use both, can use one as a nutrition score.

```
nutrition-score-fr_100g    9.013342
```

How Does Nutrition Score Relate to Nutrition Grade?

```
1 france_scores = OFF_df[['nutrition_grade_fr', 'nutrition-score-fr_100g']].dropna(how='any') #drop nulls so every score
2 means = france_scores.groupby('nutrition_grade_fr', as_index=False).mean() #group scores by grade and average
3 plt.scatter(means['nutrition_grade_fr'], means['nutrition-score-fr_100g'])
4 plt.title('Average Nutritional Score for Each Grade')
5 plt.xlabel('France Grade')
6 plt.ylabel('Average French Nutritional Score')
7 plt.show()
```



There is a clear relationship between grade and nutrition score as seen here where a grade of a is associated with a lower score whereas a grade of e is associated with a higher score. Thus, we can replace grades with the average score for that grade to utilize scores in the modeling for regression.

A "better" grade is associated with something of higher nutrition. So lower nutrition scores are associated with healthier food.

```
1 OFF_df[['product_name', 'nutrition_grade_fr', 'nutrition-score-fr_100g']].dropna(how='any')
2
```

	product_name	nutrition_grade_fr	nutrition-score-fr_100g
1	Banana Chips Sweetened (Whole)	d	14.0
2	Peanuts	b	0.0
3	Organic Salted Nut Mix	d	12.0
7	Organic Muesli	c	7.0
12	Zen Party Mix	d	12.0
...	...	...	...
363553	Neszt Cochon Con	d	17.0
363556	Natural Cassava	a	-1.0
363576	Tartines craquantes bio au sarrasin	a	-4.0
363580	Amandes	b	0.0
363592	Mint Melange Tea A Blend Of Peppermint, Lemon ...	b	0.0

257472 rows x 3 columns

Exploring relevance of other category-related columns:

```
1 #exploring what other columns are:
2 category=OFF_df[['categories', 'categories_tags', 'categories_en',"main_category","main_category_en","pnns_groups_1","pnns_groups_2"]]
3
```

```
1 category[~category["main_category"].isnull()].head(5)
```

	categories	categories_tags	categories_en	main_category	main_category_en
47	Filet de bœuf	fr:filet-de-boeuf	fr:Filet-de-boeuf	fr:filet-de-boeuf	fr:Filet-de-boeuf
178	Légumes-feuilles	en:plant-based-foods-and-beverages,en:plant-based-foods-and-beverages	Plant-based foods and beverages,Plant-based foods and beverages	en:plant-based-foods-and-beverages	Plant-based foods and beverages
179	Snacks sucrés,Biscuits et gâteaux,Pâtisseries	en:sugary-snacks,en:biscuits-and-cakes,en:pastries	Sugary snacks,Biscuits and cakes,Pastries	en:sugary-snacks	Sugary snacks
181	Plant-based foods and beverages,Plant-based foods and beverages	en:plant-based-foods-and-beverages,en:plant-based-foods-and-beverages	Plant-based foods and beverages,Plant-based foods and beverages	en:plant-based-foods-and-beverages	Plant-based foods and beverages
184	Snacks sucrés,Biscuits et gâteaux,Pâtisseries	en:sugary-snacks,en:biscuits-and-cakes,en:pastries	Sugary snacks,Biscuits and cakes,Pastries	en:sugary-snacks	Sugary snacks

All of these columns are describing the category of food but pnns\_groups\_1 and pnns\_groups\_2 seem to be the easiest to clean so these will be the focus.

```
1 OFF_df["pnns_groups_1"].unique()

array([nan, 'unknown', 'Fruits and vegetables', 'Sugary snacks',
       'Cereals and potatoes', 'Beverages', 'Composite foods',
       'Fish Meat Eggs', 'Fat and sauces', 'Milk and dairy products',
       'fruits-and-vegetables', 'Salty snacks', 'sugary-snacks',
       'cereals-and-potatoes', 'salty-snacks'], dtype=object)

1 OFF_df["pnns_groups_2"].unique()

array([nan, 'unknown', 'Vegetables', 'Biscuits and cakes', 'Bread',
       'Legumes', 'Sweetened beverages', 'Pizza pies and quiche', 'Meat',
       'Sweets', 'Non-sugared beverages', 'Alcoholic beverages',
       'Dressings and sauces', 'Ice cream', 'Cheese', 'One-dish meals',
       'vegetables', 'Appetizers', 'Chocolate products', 'Soups',
       'Fruits', 'Sandwich', 'Cereals', 'Milk and yogurt', 'Fats',
       'Artificially sweetened beverages', 'Fruit juices', 'Nuts',
       'Breakfast cereals', 'Eggs', 'Fish and seafood', 'Dried fruits',
       'Processed meat', 'Potatoes', 'pastries', 'Dairy desserts',
       'Fruit nectars', 'Tripe dishes', 'fruits',
       'Salty and fatty products', 'cereals', 'legumes', 'nuts'],
      dtype=object)
```

Groups PNNS 2 has more specific categories compared to pnns\_groups\_1, and both are relatively easy to clean with some repeated values like "Vegetables" and "vegetables".

```
1 drop_columns=["emb_codes_tags","emb_codes","categories", 'categories_tags', 'categories_en',"main_category","main_category_en"]
2 OFF_df.drop(drop_columns, axis=1, inplace=True)
```

## CLEANING pnns columns

```
1 OFF_df["pnns_groups_1"]=OFF_df["pnns_groups_1"].apply(lambda x:"Vegetables" if x=="vegetables" or x=="Vegetables" else x)
2 OFF_df["pnns_groups_1"]=OFF_df["pnns_groups_1"].apply(lambda x:"Cereals" if x=="Cereals" or x=="Breakfast cereals" or x=="Cereals and potatoes" else x)
3 OFF_df["pnns_groups_1"]=OFF_df["pnns_groups_1"].apply(lambda x:"Nuts" if x=="nuts" or x=="Nuts" else x)
4 OFF_df["pnns_groups_1"]=OFF_df["pnns_groups_1"].apply(lambda x:"Legumes" if x=="Legumes" or x=="legumes" else x)
5 OFF_df["pnns_groups_1"]=OFF_df["pnns_groups_1"].apply(lambda x:"Fruits" if x=="Fruits" or x=="fruits" else x)
6 print(OFF_df["pnns_groups_1"].unique())

[nan 'unknown' 'Fruits and vegetables' 'Sugary snacks'
 'Cereals and potatoes' 'Beverages' 'Composite foods' 'Fish Meat Eggs'
 'Fat and sauces' 'Milk and dairy products' 'fruits-and-vegetables'
 'Salty snacks' 'sugary-snacks' 'cereals-and-potatoes' 'salty-snacks']
```

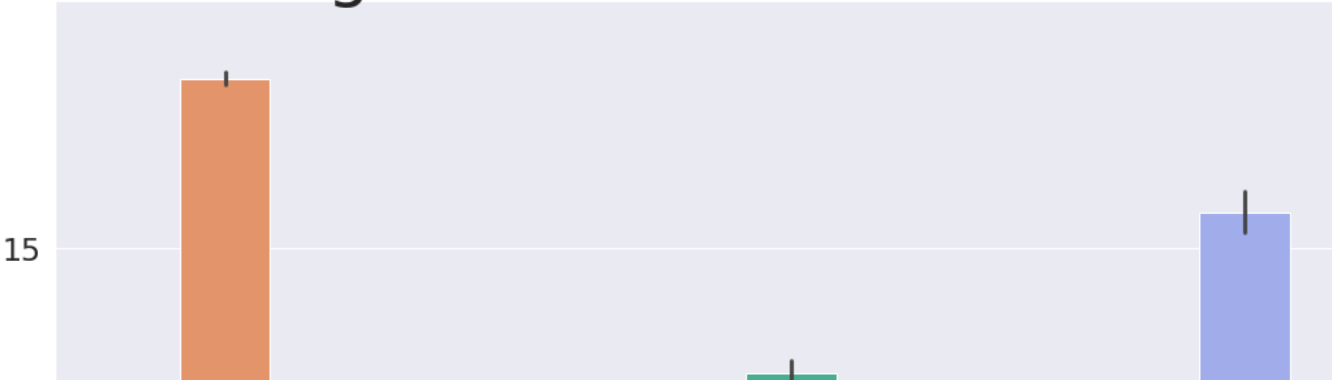


*How Does Nutrition Score Differ By Category*

```
1 #drop rows with NaN in nutrition score and pnns_groups_1 and make bar plot
2 categories_food = OFF_df[["pnns_groups_1", 'nutrition-score-fr_100g']].dropna(axis=0,how="any")
3 fig, ax = plt.subplots(figsize=(20, 20))
4 sns.set(font_scale = 4)
5 chart=sns.barplot(x="pnns_groups_1",y="nutrition-score-fr_100g",data=categories_food,ax=ax)
6 chart.set_xticklabels(chart.get_xticklabels(), rotation=90)
7 plt.title("Categories of Food and Nutrition Scores")
8
```

Text(0.5, 1.0, 'Categories of Food and Nutrition Scores')

# Categories of Food and Nutrition S



From this barplot, it can be seen that potatoes, eggs, vegetables, fruits, and legumes are the healthiest.  
The highest nutrition scores (unhealthiest)= chocolate product, biscuits and cake.



## Exploring Columns Related to Additives and Traces and Palm Oil



Additives



1 OFF\_df[["additives\_n", "additives\_tags", "additives\_en"]].dtypes

```
additives_n      float64
additives_tags    object
additives_en      object
dtype: object
```

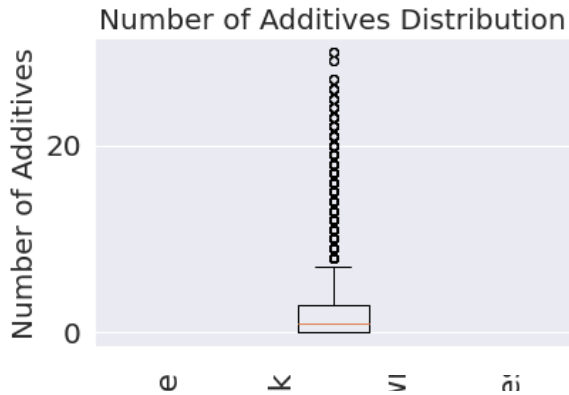


N means number of additives, tags are abbreviations here, additives listed. The number of additives seems the most relevant here, although conducting one-hot encoding on the additives themselves could be useful.



```
1 plt.boxplot(OFF_df["additives_n"].dropna())
2 plt.ylabel("Number of Additives",fontsize=20)
3 plt.title("Number of Additives Distribution",fontsize=20)
4 plt.xticks([])
5 plt.yticks(fontsize=20)
6
```

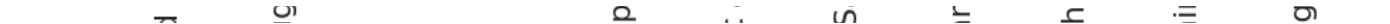
(array([-20., 0., 20., 40.]), <a list of 4 Text major ticklabel objects>)



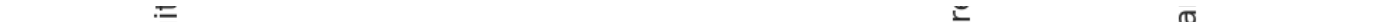
It appears that most products have 0-few additives added with a handful of outliers. May not be a useful feature



Traces



```
1 traces=OFF_df[["traces","traces_tags","traces_en"]]
2
```



```

1 frac_nan_perCol_OFF = []
2
3 for i in range(traces.shape[1]):
4     # calculate fraction of NaN values in the column
5     frac_nan = len(traces[traces[[traces.columns[i]]].isna()[traces.columns[i]] == True]) / len(traces[[traces.columns[i]]
6
7
8     # append the fraction of NaN values for the column to the list
9     frac_nan_perCol_OFF.append(frac_nan)
10
11 columns_OFF = traces.columns.tolist()
12
13 dictionary_fracNulls_OFF = dict(zip(columns_OFF, frac_nan_perCol_OFF))
14 print(dictionary_fracNulls_OFF)

{'traces': 0.9158322026797184, 'traces_tags': 0.9158349772898163, 'traces_en': 0.9158349772898163}

```

traces have high number of nulls and not much relevance to nutrition score, will drop

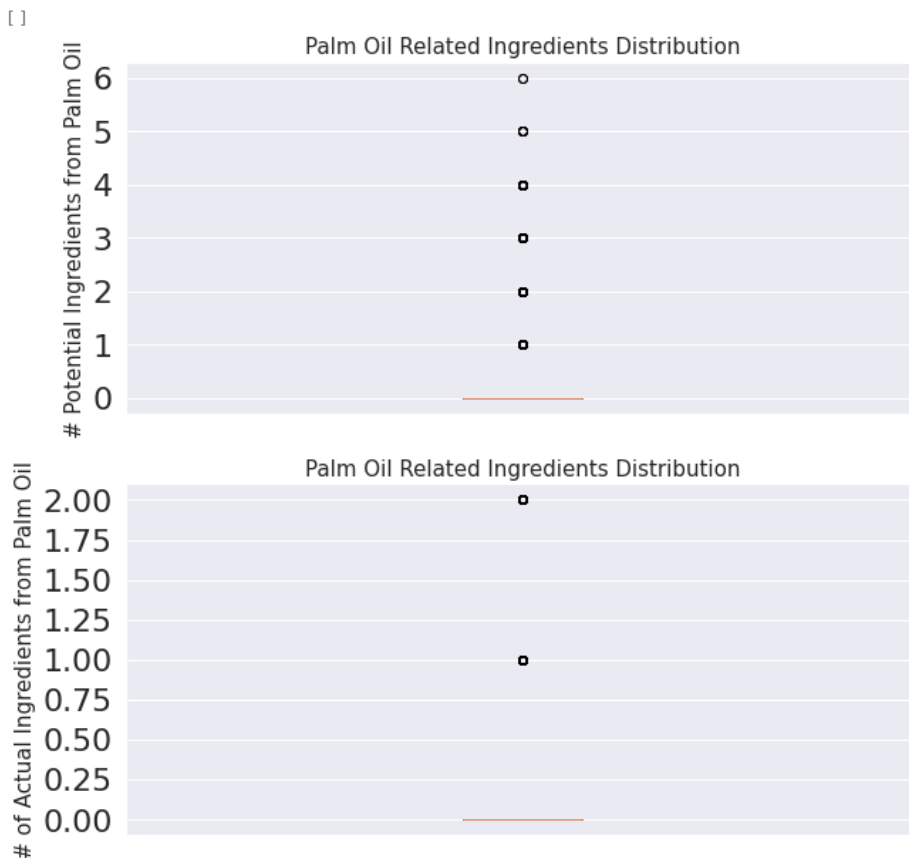
```
1 OFF_df.drop(traces, axis=1, inplace=True)
```

Palm Oil

```

1 fig,ax=plt.subplots(2,figsize=(10,10))
2 sns.set(font_scale = 1)
3 ax[0].boxplot(OFF_df["ingredients_that_may_be_from_palm_oil_n"].dropna())
4 ax[0].set_ylabel("# Potential Ingredients from Palm Oil",fontsize=15)
5 ax[0].set_title("Palm Oil Related Ingredients Distribution",fontsize=15)
6 ax[0].set_xticks([])
7 ax[1].boxplot(OFF_df["ingredients_from_palm_oil_n"].dropna())
8 ax[1].set_ylabel("# of Actual Ingredients from Palm Oil",fontsize=15)
9 ax[1].set_title("Palm Oil Related Ingredients Distribution",fontsize=15)
10 ax[1].set_xticks([])
11

```

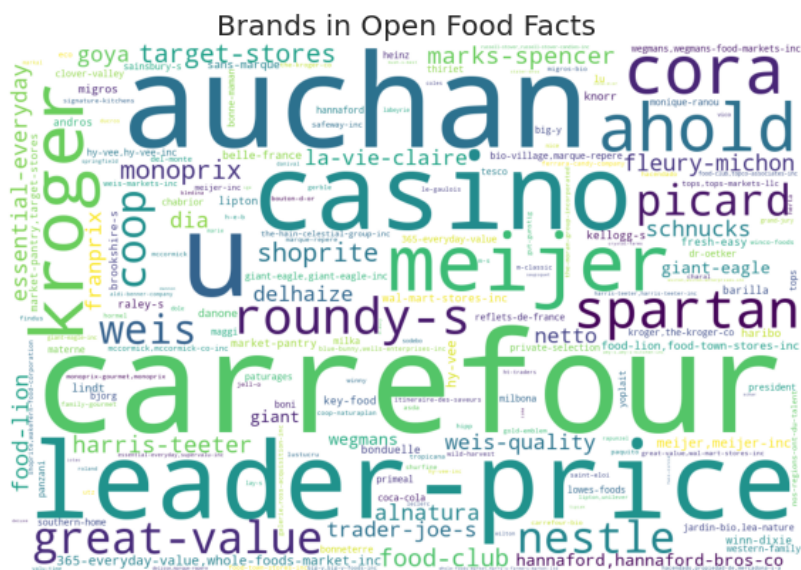


It appears that Ingredients related to Palm Oil might not be very relevant as most of the products have zero ingredients with a few outliers.

## Brands

What Brands Were The Foods From?

```
1 #Most Common Brands
2 brand_frequencies = OFF_df['brands_tags'].dropna().value_counts()
3 brand_frequencies.head(10)
4
5 # Word Cloud of Brands (OFF)
6 import wordcloud
7
8 plt.figure(figsize=(10,10))
9 wc = wordcloud.WordCloud(background_color='white', width = 1200, height = 800).generate_from_frequencies(frequencies=brand_frequencies)
10 plt.imshow(wc)
11 plt.axis("off")
12 plt.title("Brands in Open Food Facts",fontsize=20)
13 plt.show()
```



## Dropping More Irrelevant Columns

Next, we started deciding which feature columns for the Open Food Facts dataset we wanted to drop. This involved deciding on which feature columns seemed irrelevant, which also included eliminating some columns that had too many null values (a high percentage of nulls in the column).

```
1 #creating list of columns to drop based on above and also dropping columns that are not in English- choosing en columns
2 drop_columns=["code","url","creator","packaging","packaging_tags","additives","allergens","image_url","image_small_url"]
3 OFF_df_rel=OFF_df.drop(drop_columns, axis=1,inplace=True)
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2
```

```
10
11
en:vegetarian,en:green-dot,en:pure-butter
15457
298521
E415 - Xanthan gum
39851
187244
```

Should pick the columns with tags for analysis if we want to use these columns:standardized.

## Standardizing Columns Across HNP and OFF Through Cleaning

*In order to compare how nutrition scores relate to outcomes, it is important to make sure that columns in common for both datasets are formatted the same way/have the same values for a future join. This includes columns related to the year, country, and indication.*

### Year

Make sure same years present in HNP and OFF. First extracted years from date modified column

```
1 #extract all the years from date modified and make it a new column and drop the date modified column
2 def year(x):
3     yr=x[0:4]
4     return str(yr)
5 OFF_df["year"]=OFF_df["last_modified_datetime"].apply(lambda x: year(x))
6 OFF_df=OFF_df.drop("last_modified_datetime", axis=1)

1 OFF_df["year"].unique()

array(['2016', '2017', '2015', '2014', '2013', '2012'], dtype=object)

1 HNP_df.columns

Index(['Country Name', 'Country Code', 'Indicator Name', 'Indicator Code',
      '1960', '1961', '1962', '1963', '1964', '1965', '1966', '1967', '1968',
      '1969', '1970', '1971', '1972', '1973', '1974', '1975', '1976', '1977',
      '1978', '1979', '1980', '1981', '1982', '1983', '1984', '1985', '1986',
      '1987', '1988', '1989', '1990', '1991', '1992', '1993', '1994', '1995',
      '1996', '1997', '1998', '1999', '2000', '2001', '2002', '2003', '2004',
      '2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013',
      '2014', '2015', 'Unnamed: 60'],
      dtype='object')

1 #filter columns of HNP to include only the years in OFF
2 HNP_rel=HNP_df[["Country Name","Indicator Name","2012","2013","2014","2015"]]
3 HNP_rel.head(5)
```

	Country Name		Indicator Name	2012	2013	2014	2015
0	Arab World	% of females ages 15-49 having comprehensive c...		NaN	NaN	NaN	NaN
1	Arab World	% of males ages 15-49 having comprehensive cor...		NaN	NaN	NaN	NaN
2	Arab World	Adolescent fertility rate (births per 1,000 wo...		49.383745	48.796558	48.196418	NaN
3	Arab World	Adults (ages 15+) and children (0-14 years) li...		NaN	NaN	NaN	NaN
4	Arab World	Adults (ages 15+) and children (ages 0-14) new...		NaN	NaN	NaN	NaN

no data for 2016 and 2017 in HNP, so use 2015 data for that. will change year in OFF to reflect this

```
1 def change_year(x):
2     if x=="2017":
3         y="2015"
```

```

4     elif x=="2016":
5         y="2015"
6     else:
7         y=x
8     return y
9 OFF_df["year_changed"]=OFF_df["year"].apply(lambda x:change_year(x))

1 OFF_df["year_changed"].unique()

array(['2015', '2014', '2013', '2012'], dtype=object)

1 OFF_df["year_changed"].value_counts()

2015    353635
2014     5014
2013     1460
2012       302
Name: year_changed, dtype: int64

```

### Make Sure No Countries in HNP not in OFF

```

1 #filter all rows in HNP for the countries in OFF
2 list_OFF_countries=OFF_df["countries_final"].to_list()
3 HNP_df=HNP_df[HNP_df["Country Name"].isin(list_OFF_countries)]
4

```

### Removing Irrelevant Indications in HNP

```

1 #go through HNP and determine relevant rows of indications (use unique)
2 HNP_d=HNP_rel.dropna(axis=0,how='all',subset=("2012","2013","2014","2015")) #first drop rows without year
3 print(HNP_d.shape)
4
5 HNP_d["Indicator Name"].unique()

```

keep indications associated with nutrition/relevant to nutrition:

```

1 #keep indications associated with nutrition/relevant to nutrition:
2 row_list_keep=['Consumption of iodized salt (% of households)','Low-birthweight babies (% of births)',
3               'Infant and young child feeding practices, all 3 IYCF (% children ages 6-23 months)',
4               'Malnutrition prevalence, height for age, female (% of children under 5)',
5               'Malnutrition prevalence, height for age, male (% of children under 5)',
6               'Malnutrition prevalence, weight for age, female (% of children under 5)',
7               'Malnutrition prevalence, weight for age, male (% of children under 5)',
8               'Prevalence of overweight, female (% of children under 5)',
9               'Prevalence of overweight, male (% of children under 5)',
10              'Prevalence of severe wasting, weight for height, female (% of children under 5)',
11              'Prevalence of severe wasting, weight for height, male (% of children under 5)',
12              'Prevalence of wasting, female (% of children under 5)',
13              'Prevalence of wasting, male (% of children under 5)',
14              'Prevalence of overweight (% of adults)',
15              'Prevalence of overweight, female (% of female adults)',
16              'Prevalence of overweight, male (% of male adults)',
17              'Vitamin A supplementation coverage rate (% of children ages 6-59 months)',
18              'Prevalence of overweight (% of children under 5)',
19              'Prevalence of severe wasting, weight for height (% of children under 5)',
20              'Prevalence of undernourishment (% of population)',
21              'Prevalence of wasting (% of children under 5)',
22              'Cause of death, by communicable diseases and maternal, prenatal and nutrition conditions (% of total)',
23              'Diabetes prevalence (% of population ages 20 to 79)',
24              'Malnutrition prevalence, height for age (% of children under 5)'
25              'Malnutrition prevalence, weight for age (% of children under 5)']
26
27 #less repetitive (ie. don't care about male vs female)
28 row_list_keep=['Consumption of iodized salt (% of households)','Low-birthweight babies (% of births)',
29               'Prevalence of overweight (% of adults)',
30               'Vitamin A supplementation coverage rate (% of children ages 6-59 months)',
31               'Prevalence of overweight (% of children under 5)',

```

```

32     'Prevalence of severe wasting, weight for height (% of children under 5)',
33     'Prevalence of undernourishment (% of population)',
34     'Prevalence of wasting (% of children under 5)',
35     'Cause of death, by communicable diseases and maternal, prenatal and nutrition conditions (% of total)',
36     'Diabetes prevalence (% of population ages 20 to 79)',
37     'Malnutrition prevalence, height for age (% of children under 5)'
38     'Malnutrition prevalence, weight for age (% of children under 5)']
39
40 HNP_rel_nutrition=HNP_d[HNP_d["Indicator Name"].isin(row_list_keep)]
41

```

## ▼ EDA for Health and Nutrition Population Outcomes

***This section explores the health outcomes in the HNP dataset and includes various exploratory figures comparing health outcomes and nutrition scores***

*How Have Health Outcomes Changed From 2012-2015 Across the World*

```

1 #line plot with a different color line for each health outcome where x axis is year and y axis is prevalence
2 outcomes_2012=HNP_rel_nutrition[["Indicator Name", "2012"]].dropna(axis=0,how="any")
3 outcomes_2013=HNP_rel_nutrition[["Indicator Name", "2013"]].dropna(axis=0,how="any")
4 outcomes_2014=HNP_rel_nutrition[["Indicator Name", "2014"]].dropna(axis=0,how="any")
5 outcomes_2015=HNP_rel_nutrition[["Indicator Name", "2015"]].dropna(axis=0,how="any")
6
7 outcomes_2012=outcomes_2012.groupby(by="Indicator Name").mean()
8 outcomes_2013=outcomes_2013.groupby(by="Indicator Name").mean()
9 outcomes_2014=outcomes_2014.groupby(by="Indicator Name").mean()
10 outcomes_2015=outcomes_2015.groupby(by="Indicator Name").mean()

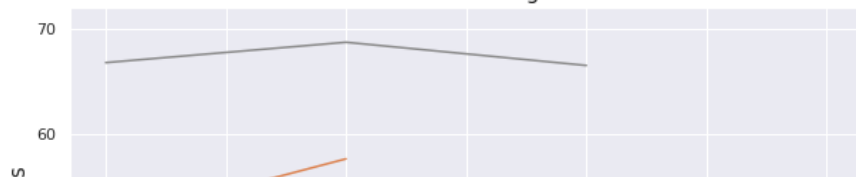
1 m1=outcomes_2012.merge(outcomes_2013,how="left", left_on='Indicator Name',right_on="Indicator Name")
2 m2=m1.merge(outcomes_2014,how="left", left_on='Indicator Name',right_on="Indicator Name")
3 yearly_outcomes=m2.merge(outcomes_2015, how="left",left_on='Indicator Name',right_on="Indicator Name")
4 yearly_outcomes

1 yearly_outcomes.T.plot(figsize=(10,10))
2 plt.xlabel("Year",fontsize=15)
3 plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))
4 plt.ylabel("Average Prevalence of Nutrition Outcomes",fontsize=15)
5 plt.title("How Nutrition Outcomes Change from 2012-2015",fontsize=15)
6

```

```
Text(0.5, 1.0, 'How Nutrition Outcomes Change from 2012-2015')
```

### How Nutrition Outcomes Change from 2012-2015



Nutrition Outcomes remain relatively the same across year, so can just keep whichever column has the numbers.

```
ut =
```

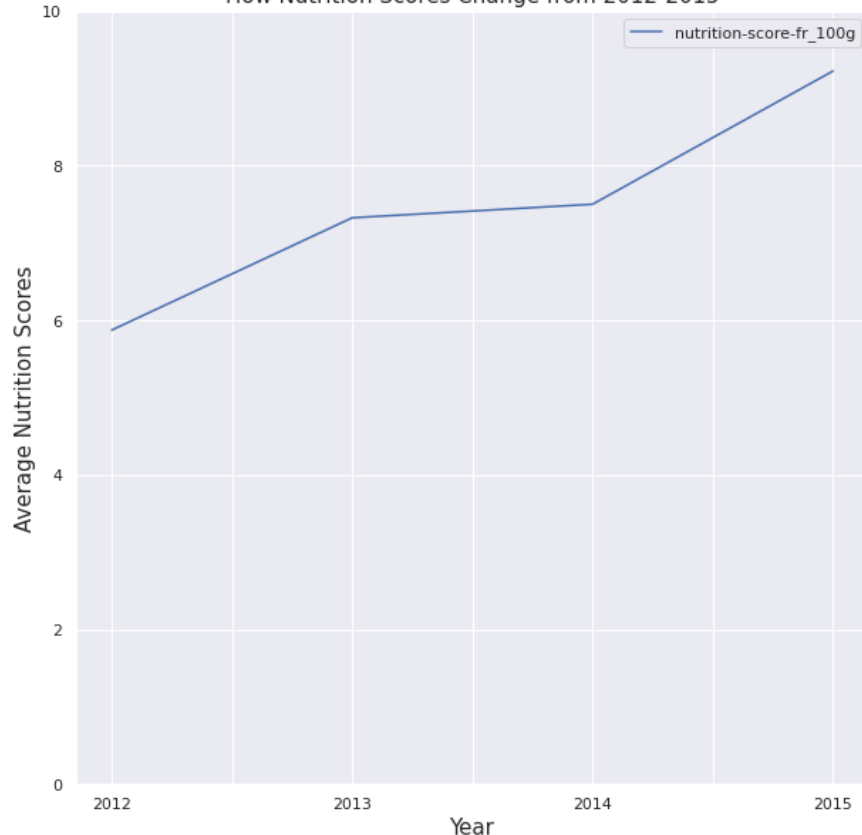
How Have Nutrition Scores Changed Across the Years- OFF

```
ni =
```

```
1 scores_years=OFF_df[["year_changed", "nutrition-score-fr_100g"]].groupby(by="year_changed").mean()
2 scores_years.plot(figsize=(10,10))
3 plt.xlabel("Year",fontsize=15)
4 plt.ylabel("Average Nutrition Scores",fontsize=15)
5 plt.title("How Nutrition Scores Change from 2012-2015",fontsize=15)
6 plt.ylim([0,10])
7
```

```
(0.0, 10.0)
```

### How Nutrition Scores Change from 2012-2015



While health outcomes did not change much over 2012-2015, nutrition scores of food did, increasing in score, becoming unhealthier.

What are Nutrition Related Health Outcomes in France and USA- most common countries in Open Food Facts?

```
1 HNP_fr=HNP_rel_nutrition[(HNP_rel_nutrition["Country Name"]=="France")]
2 HNP_us=HNP_rel_nutrition[(HNP_rel_nutrition["Country Name"]=="United States")]
3 HNP_fr.merge(HNP_us,left_on="Indicator Name",right_on="Indicator Name")
```

```
1 HNP_fr_us=pd.DataFrame(data=[["Cause of death, by communicable diseases and maternal, prenatal and nutrition conditions"]])
```

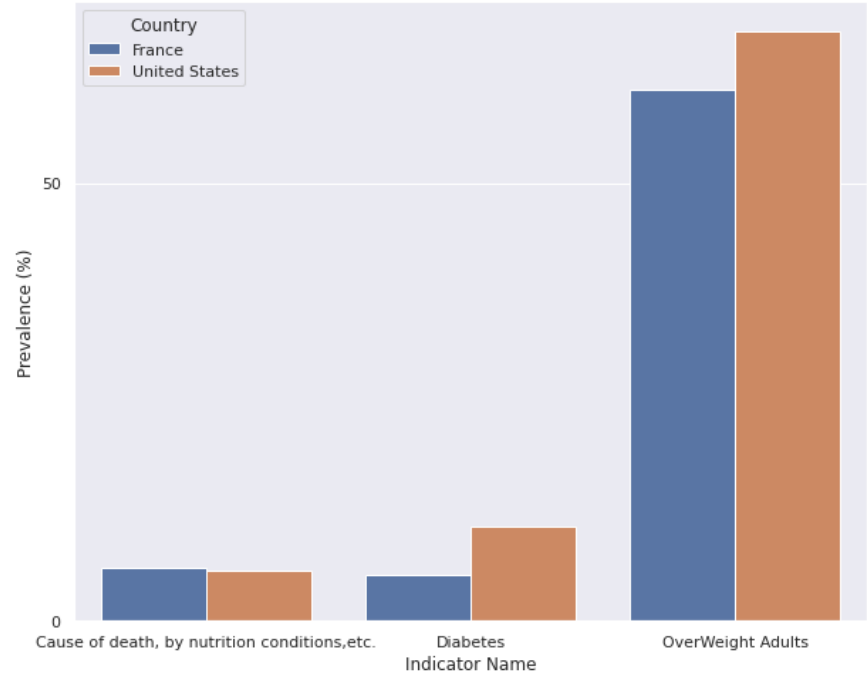


```
1 HNP_fr_us=pd.DataFrame(data=[["Cause of death, by nutrition conditions,etc.",6.1,"France"],
2                               ["Cause of death, by nutrition conditions,etc.",5.7,"United States"]
3                               ,["Diabetes",5.3,"France"],["Diabetes",10.8,"United States"],
4                               ["OverWeight Adults",60.7,"France"],["OverWeight Adults",67.3,"United States"]],columns=[

1 plt.figure()
2 sns.set(rc={'figure.figsize':(10,8)})
3 chart=sns.barplot(x="Indicator Name",y="Prevalence (%)",hue="Country",data=HNP_fr_us)
4 sns.set(font_scale=8)
5 plt.title("Prevalence of Nutrition Related Health Outcomes in France vs USA",fontsize=20)
6
```

Text(0.5, 1.0, 'Prevalence of Nutrition Related Health Outcomes in France vs USA')

Prevalence of Nutrition Related Health Outcomes in France vs USA



```
1 countries_scores = OFF_df[["countries_final", 'nutrition-score-fr_100g']].dropna(axis=0,how="any")
2 countries_scores = countries_scores.groupby(by="countries_final",as_index=False).mean()
3 fr_us_scores=countries_scores[(countries_scores['countries_final']=="France") | (countries_scores['countries_final']=="United States")]
4 fr_us_scores
```

	countries_final	nutrition-score-fr_100g
31	France	8.861481
90	United States	9.450814

Based on the above plot, for the two main countries that Open Food Facts has products from, France has slightly better healthcare outcomes related to nutrition than the United States with lower prevalence of diabetes and overweight adults. This makes sense with France having a lower ("healthier") average nutrition score for the products consumed in that country.

### Compare Health Outcomes with Nutrition Scores

Ultimately, in this section, the OFF and HNP datasets were joined and relationships between scores, countries, and outcomes were explored. No relationship between outcomes and average scores for countries was found

```
1 #use OFF, just groupby country and average nutrition score. Then do bar chart!
2 OFF_df['avg_nutrition_score'] = (OFF_df['nutrition-score-fr_100g'] + OFF_df['nutrition-score-uk_100g']) / 2
3 OFF_df[['product_name', 'countries_final', 'avg_nutrition_score']]
4
5 # before grouping by country and averaging nutrition score per country, deal with null nutrition scores:
```

```
6 countries_nutritionScores = OFF_df[OFF_df['avg_nutrition_score'].notna()]\n7\n8 countries_nutritionScores = countries_nutritionScores.groupby('countries_final').mean()\n9 countries_nutritionScores.head(5)
```

	no_nutriments	additives_n	ingredients_from_palm_oil_n	ingredients_from_palm_oil	ingredients_that
countries_final					
Albania	NaN	0.500000		0.000000	NaN
Algeria	NaN	1.692308		0.230769	NaN
Andorra	NaN	0.428571		0.000000	NaN
Argentina	NaN	2.666667		0.000000	NaN
Armenia	NaN	0.000000		0.000000	NaN

5 rows x 108 columns

```
1 ordered_country_nutritionScores = countries_nutritionScores.sort_values('avg_nutrition_score', ascending=False)\n2 print("Country with highest average nutrition score:")\n3 display(ordered_country_nutritionScores[0:1])\n4 print("Country with lowest average nutrition score:")\n5 display(ordered_country_nutritionScores[-1:])
```

Country with highest average nutrition score:

	no_nutriments	additives_n	ingredients_from_palm_oil_n	ingredients_from_palm_oil	ingredients_that
countries_final					
Pakistan	NaN	NaN		NaN	NaN

1 rows x 108 columns

Country with lowest average nutrition score:

	no_nutriments	additives_n	ingredients_from_palm_oil_n	ingredients_from_palm_oil	ingredients_that
countries_final					
Vanuatu	NaN	1.0		0.0	NaN

1 rows x 108 columns

```
1 # simplify this dataframe to contain only the country name and nutrition score:\n2\n3 countries_nutritionScores = countries_nutritionScores[['avg_nutrition_score']]\n4 countries_nutritionScores.reset_index(inplace=True)\n5 countries_nutritionScores\n6
```

	countries_final	avg_nutrition_score
0	Albania	6.285714
1	Algeria	14.416667
2	Andorra	8.125000
3	Argentina	13.166667
4	Armenia	22.000000
...	...	...
88	United Arab Emirates	11.388889

Now, the health outcomes dataframe will be merged with the dataframe containing nutritional scores and countries.

```

1 #less repetitive (ie. don't care about male vs female)
2 row_list_keep=['Consumption of iodized salt (% of households)', 'Low-birthweight babies (% of births)',
3               'Prevalence of overweight (% of adults)',
4               'Vitamin A supplementation coverage rate (% of children ages 6-59 months)',
5               'Prevalence of overweight (% of children under 5)',
6               'Prevalence of severe wasting, weight for height (% of children under 5)',
7               'Prevalence of undernourishment (% of population)',
8               'Prevalence of wasting (% of children under 5)',
9               'Cause of death, by communicable diseases and maternal, prenatal and nutrition conditions (% of total)',
10              'Diabetes prevalence (% of population ages 20 to 79)',
11              'Malnutrition prevalence, height for age (% of children under 5)'
12              'Malnutrition prevalence, weight for age (% of children under 5)']
13
14 HNP_rel_nutrition=HNP_d[HNP_d["Indicator Name"].isin(row_list_keep)]
15 len(np.unique(HNP_rel_nutrition['Indicator Name']))

10

```

```

1 join_df = countries_nutritionScores.merge(HNP_rel_nutrition, left_on='countries_final', right_on='Country Name')
2 join_df

```

	countries_final	avg_nutrition_score	Country Name	Indicator Name	2012	2013	2014
0	Albania	6.285714	Albania	Cause of death, by communicable diseases and m...	5.1	NaN	NaN
1	Albania	6.285714	Albania	Diabetes prevalence (% of population ages 20 t...	NaN	NaN	NaN
2	Albania	6.285714	Albania	Prevalence of overweight (% of adults)	NaN	NaN	52.700000
3	Algeria	14.416667	Algeria	Cause of death, by communicable diseases and m...	14.5	NaN	NaN
4	Algeria	14.416667	Algeria	Diabetes prevalence (% of population ages 20 t...	NaN	NaN	NaN
...	...	...	...	...	...	...	...
425	World	1.700000	World	Diabetes prevalence (% of population ages 20 t...	NaN	NaN	NaN
426	World	1.700000	World	Prevalence of overweight (% of children under 5)	NaN	NaN	6.107095
427	World	1.700000	World	Prevalence of severe wasting, weight for heigh...	NaN	NaN	2.449456
428	World	1.700000	World	Prevalence of undernourishment (% of population)	11.4	11.2	11.000000
429	World	1.700000	World	Prevalence of wasting (% of children under 5)	NaN	NaN	7.502322

430 rows x 8 columns

```

1 # average the prevalence of different indicators:
2 join_df = join_df.fillna(0)
3 join_df['avg_prevalence'] = join_df[['2012', '2013', '2014', '2015']].mean(axis=1)
4 join_df.drop(columns=['2012', '2013', '2014', '2015'], inplace = True)
5 join_df

```

	countries_final	avg_nutrition_score	Country Name	Indicator Name	avg_prevalence
0	Albania	6.285714	Albania	Cause of death, by communicable diseases and m...	1.275000
1	Albania	6.285714	Albania	Diabetes prevalence (% of population ages 20 t...	2.575000
2	Albania	6.285714	Albania	Prevalence of overweight (% of adults)	13.175000
3	Algeria	14.416667	Algeria	Cause of death, by communicable diseases and m...	3.625000
4	Algeria	14.416667	Algeria	Diabetes prevalence (% of population ages 20 t...	1.875000
...	...	...	...	...	...
425	World	1.700000	World	Diabetes prevalence (% of population ages 20 t...	2.131485
426	World	1.700000	World	Prevalence of overweight (% of children under 5)	1.526774
427	World	1.700000	World	Prevalence of severe wasting, weight for heigh...	0.612364
428	World	1.700000	World	Prevalence of undernourishment (% of population)	11.100000

With the dataframe above, one can see how the average nutrition scores for certain countries correlate to the amount and types of health outcomes present in that country over the span of 4 years.

Below, plots will be generated to display how the prevalence of certain health outcomes relates to nutrition scores.

```
1 join_df.groupby(['Indicator Name', 'avg_nutrition_score']).mean()
```

		2012	2013
	Indicator Name	avg_nutrition_score	
Cause of death, by communicable diseases and maternal, prenatal and nutrition conditions (% of total)		1.000000	17.600000
		1.583333	10.600000
		1.700000	23.010006
		2.818182	4.900000
		3.000000	16.300000
...		...	...
Vitamin A supplementation coverage rate (% of children ages 6-59 months)		16.000000	90.000000
		17.000000	88.000000
		20.000000	73.000000
		20.666667	88.000000
		30.500000	99.000000

420 rows x 4 columns

```
1 len(np.unique(join_df['Indicator Name']))
10

1 join_df['Indicator Name'][0]
'Cause of death, by communicable diseases and maternal, prenatal and nutrition conditions (% of total)'

1 join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[0]]
```

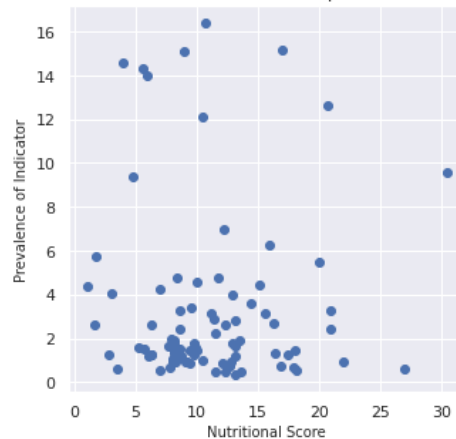
	countries_final	avg_nutrition_score	Country Name	Indicator Name	2012	2013
0	Albania	6.285714	Albania	Cause of death, by communicable diseases and m...	5.100000	NaN
3	Algeria	14.416667	Algeria	Cause of death, by communicable diseases and m...	14.500000	NaN
12	Argentina	13.166667	Argentina	Cause of death, by communicable diseases and m...	11.300000	NaN
16	Armenia	22.000000	Armenia	Cause of death, by communicable diseases and m...	3.800000	NaN
21	Australia	8.268206	Australia	Cause of death, by communicable diseases and m...	3.600000	NaN
...	...	...	...	...	...	...

```

1 nut_scores_0 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[0]]['avg_nutrition_score']
2 prevalence_0 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[0]]['avg_prevalence']
3
4 sns.set(font_scale=1)
5 plt.figure(figsize=(5,5))
6 plt.scatter(nut_scores_0, prevalence_0)
7 plt.xlabel('Nutritional Score',fontsize=10)
8 plt.ylabel('Prevalence of Indicator',fontsize=10)
9 plt.title(np.unique(join_df['Indicator Name'])[0])
10 plt.show()
11

```

Cause of death, by communicable diseases and maternal, prenatal and nutrition conditions (% of total)



```

1 nut_scores_1 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[1]]['avg_nutrition_score']
2 prevalence_1 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[1]]['avg_prevalence']
3
4 sns.set(font_scale=1)
5 plt.figure(figsize=(5,5))
6 plt.scatter(nut_scores_1, prevalence_1)
7 plt.xlabel('Nutritional Score')
8 plt.ylabel('Prevalence of Indicator')
9 plt.title(np.unique(join_df['Indicator Name'])[1])
10 plt.show()
11

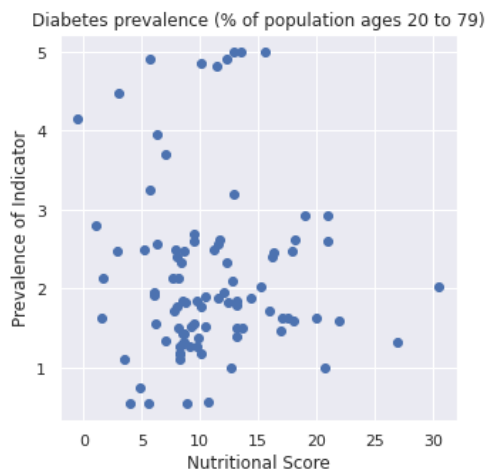
```

## Consumption of iodized salt (% of households)

```

1 nut_scores_2 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[2]]['avg_nutrition_score']
2 prevalence_2 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[2]]['avg_prevalence']
3
4 sns.set(font_scale=1)
5 plt.figure(figsize=(5,5))
6 plt.scatter(nut_scores_2, prevalence_2)
7 plt.xlabel('Nutritional Score')
8 plt.ylabel('Prevalence of Indicator')
9 plt.title(np.unique(join_df['Indicator Name'])[2])
10 plt.show()
11

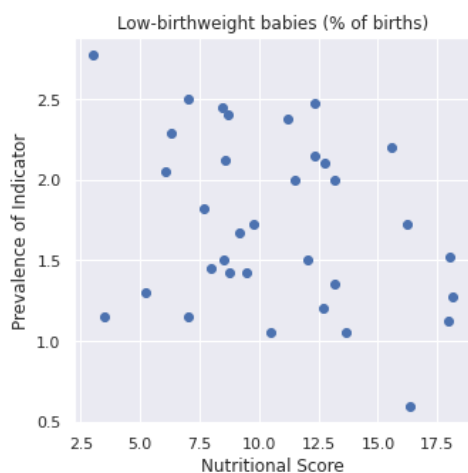
```



```

1 nut_scores_3 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[3]]['avg_nutrition_score']
2 prevalence_3 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[3]]['avg_prevalence']
3
4 sns.set(font_scale=1)
5 plt.figure(figsize=(5,5))
6 plt.scatter(nut_scores_3, prevalence_3)
7 plt.xlabel('Nutritional Score')
8 plt.ylabel('Prevalence of Indicator')
9 plt.title(np.unique(join_df['Indicator Name'])[3])
10 plt.show()
11

```



```

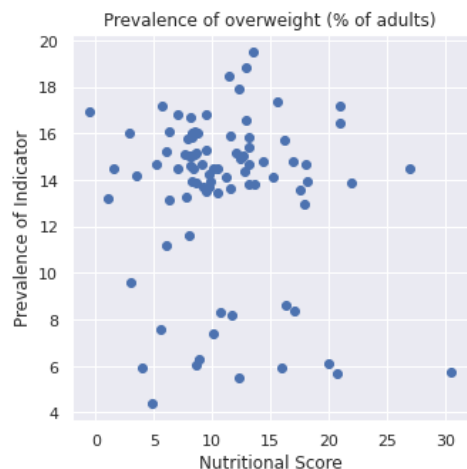
1 nut_scores_4 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[4]]['avg_nutrition_score']
2 prevalence_4 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[4]]['avg_prevalence']
3
4 sns.set(font_scale=1)
5 plt.figure(figsize=(5,5))
6 plt.scatter(nut_scores_4, prevalence_4)
7 plt.xlabel('Nutritional Score')

```

```

8 plt.ylabel('Prevalence of Indicator')
9 plt.title(np.unique(join_df['Indicator Name'])[4])
10 plt.show()
11

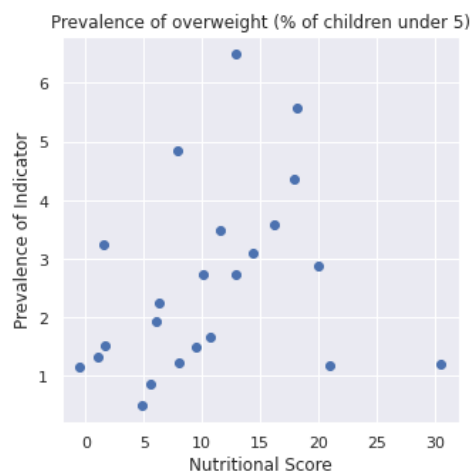
```



```

1 nut_scores_5 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[5]]['avg_nutrition_score']
2 prevalence_5 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[5]]['avg_prevalence']
3
4 sns.set(font_scale=1)
5 plt.figure(figsize=(5,5))
6 plt.scatter(nut_scores_5, prevalence_5)
7 plt.xlabel('Nutritional Score')
8 plt.ylabel('Prevalence of Indicator')
9 plt.title(np.unique(join_df['Indicator Name'])[5])
10 plt.show()
11

```

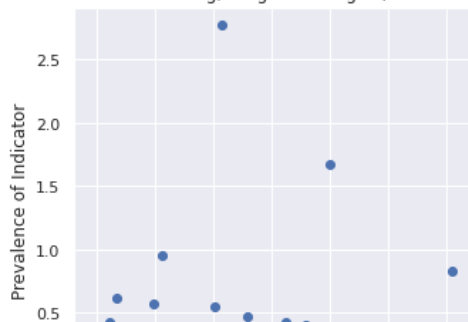


```

1 nut_scores_6 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[6]]['avg_nutrition_score']
2 prevalence_6 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[6]]['avg_prevalence']
3
4 sns.set(font_scale=1)
5 plt.figure(figsize=(5,5))
6 plt.scatter(nut_scores_6, prevalence_6)
7 plt.xlabel('Nutritional Score')
8 plt.ylabel('Prevalence of Indicator')
9 plt.title(np.unique(join_df['Indicator Name'])[6])
10 plt.show()
11

```

Prevalence of severe wasting, weight for height (% of children under 5)

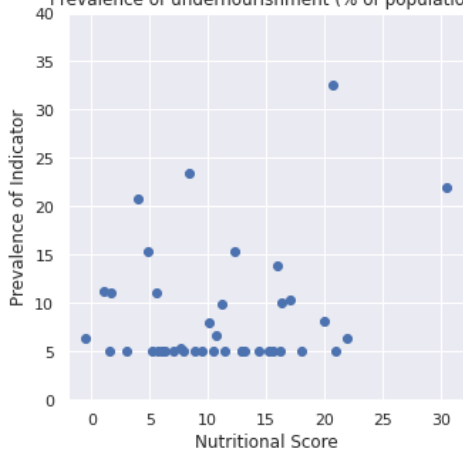


```

1 nut_scores_7 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[7]]['avg_nutrition_score']
2 prevalence_7 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[7]]['avg_prevalence']
3
4 sns.set(font_scale=1)
5 plt.figure(figsize=(5,5))
6 plt.scatter(nut_scores_7, prevalence_7)
7 plt.xlabel('Nutritional Score')
8 plt.ylabel('Prevalence of Indicator')
9 plt.title(np.unique(join_df['Indicator Name'])[7])
10 plt.ylim([0,40])
11 plt.show()
12

```

Prevalence of undernourishment (% of population)



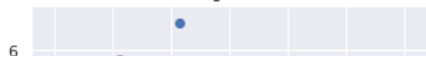
```

1 nut_scores_8 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[8]]['avg_nutrition_score']
2 prevalence_8 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[8]]['avg_prevalence']
3 sns.set(font_scale=1)
4 plt.figure(figsize=(5,5))
5 plt.scatter(nut_scores_8, prevalence_8)
6 plt.xlabel('Nutritional Score')
7 plt.ylabel('Prevalence of Indicator')
8 plt.title(np.unique(join_df['Indicator Name'])[8])
9 plt.show()
10

```



Prevalence of wasting (% of children under 5)

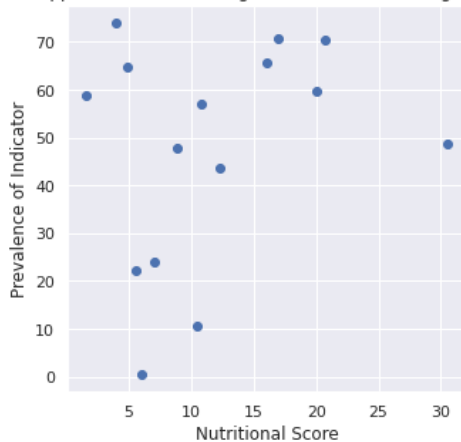


```

1 nut_scores_9 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[9]]['avg_nutrition_score']
2 prevalence_9 = join_df[join_df['Indicator Name'] == np.unique(join_df['Indicator Name'])[9]]['avg_prevalence']
3
4 sns.set(font_scale=1)
5 plt.figure(figsize=(5,5))
6 plt.scatter(nut_scores_9, prevalence_9)
7 plt.xlabel('Nutritional Score')
8 plt.ylabel('Prevalence of Indicator')
9 plt.title(np.unique(join_df['Indicator Name'])[9])
10 plt.show()
11

```

Vitamin A supplementation coverage rate (% of children ages 6-59 months)



There were no clear trends spotted between outcomes and nutrition score.

### What Countries Have the Best and Worst Nutrition Scores?

```

1 countries_scores = OFF_df[["countries_final", 'nutrition-score-fr_100g']].dropna(axis=0,how="any")
2 countries_scores = countries_scores.groupby(by="countries_final",as_index=False).mean()
3 best_scores=countries_scores.sort_values(by="nutrition-score-fr_100g",ascending=True).head(10)
4 worst_scores=countries_scores.sort_values(by="nutrition-score-fr_100g",ascending=False).head(10)
5
6
7
8
9
10
11
12
13
14

```

```

1 fig, ax = plt.subplots(figsize=(10, 10))
2 sns.set(font_scale = 1)
3 chart=sns.barplot(x="countries_final",y="nutrition-score-fr_100g",data=worst_scores,ax=ax)
4 chart.set_xticklabels(chart.get_xticklabels(), rotation=25)
5 plt.title("Countries with Worst Nutrition Scores",fontsize=20)
6 plt.xlabel("Country",fontsize=20)
7 plt.ylabel("Nutrition Score",fontsize=20)
8 plt.yticks(fontsize=20)
9 plt.xticks(fontsize=16)
10
11
12
13
14

```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 <a list of 10 Text major ticklabel objects>)
```



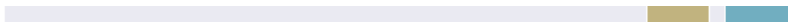
```
1 fig, ax = plt.subplots(figsize=(10, 10))
2 sns.set(font_scale = 1)
3 chart=sns.barplot(x="countries_final",y="nutrition-score-fr_100g",data=best_scores,ax=ax)
4 chart.set_xticklabels(chart.get_xticklabels(), rotation=25)
5 plt.title("Countries with Best Nutrition Scores",fontsize=20)
6 plt.xlabel("Country",fontsize=20)
7 plt.ylabel("Nutrition Score",fontsize=20)
8 plt.yticks(fontsize=20)
9 plt.xticks(fontsize=16)
10
11
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
<a list of 10 Text major ticklabel objects>)
```

## Countries with Best Nutrition Scores



### Dealing with Categorical Features



Finally, one of the last steps involved determining how we wanted to handle the categorical features in the dataset. For the Open Food Facts dataset, for example, this included the following feature columns: `categories_tags`, `additives_en`, `pnns_groups_1`, `pnns_groups_2`, `labels_tags`, and `ingredients_text`. These could either be one hot encoded or removed entirely from the dataset. At first attempt of one hot encoding, we realized that our dataset was so large that the session would crash when trying to one hot encode. We had the option to try using Apache Spark in order to handle "big data".

However, in this case, we decided to first use the dataset that contained only numeric features, move forward in machine learning model development, and test the model's performance. If our model did not perform well enough using purely numerical data, then we would go back and perform one hot encoding in Apache Spark in order to improve any possible issues of underfitting the data.

Luckily, as you will see later in the notebook, our model performed fairly well using purely numeric data, so one hot encoding was not performed.



## MODELING WITH MACHINE LEARNING



```
1 df_open_food_facts=pd.read_csv("en.openfoodfacts.org.products.tsv",sep='\t')
2
```

...atu ...ana ...iati...rlo ...alta ...cia ...ania ...cas...iba ...ania

```
1 #Extract Feature and Y Columns
2 other_cols = ['product_name', 'ingredients_text']
3 y = ['nutrition-score-uk_100g']
4 features = ['energy_100g',
5 'energy-from-fat_100g',
6 'fat_100g',
7 'saturated-fat_100g',
8 '-butyric-acid_100g',
9 '-caproic-acid_100g',
10 '-caprylic-acid_100g',
11 '-capric-acid_100g',
12 '-lauric-acid_100g',
13 '-myristic-acid_100g',
14 '-palmitic-acid_100g',
15 '-stearic-acid_100g',
16 '-arachidic-acid_100g',
17 '-behenic-acid_100g',
18 '-lignoceric-acid_100g',
19 '-cerotic-acid_100g',
20 '-montanic-acid_100g',
21 '-melissic-acid_100g',
22 'monounsaturated-fat_100g',
23 'polyunsaturated-fat_100g',
24 'omega-3-fat_100g',
25 '-alpha-linolenic-acid_100g',
26 '-eicosapentaenoic-acid_100g',
27 '-docosahexaenoic-acid_100g',
28 'omega-6-fat_100g',
29 '-linoleic-acid_100g',
30 '-arachidonic-acid_100g',
31 '-gamma-linolenic-acid_100g',
32 '-dihomo-gamma-linolenic-acid_100g',
33 'omega-9-fat_100g',
34 '-oleic-acid_100g',
35 '-elaidic-acid_100g',
36 '-gondoic-acid_100g',
37 '-mead-acid_100g',
38 '-erucic-acid_100g',
```

```

39 '-nervonic-acid_100g',
40 'trans-fat_100g',
41 'cholesterol_100g',
42 'carbohydrates_100g',
43 'sugars_100g',
44 '-sucrose_100g',
45 '-glucose_100g',
46 '-fructose_100g',
47 '-lactose_100g',
48 '-maltose_100g',
49 '-maltodextrins_100g',
50 'starch_100g',
51 'polyols_100g',
52 'fiber_100g',
53 'proteins_100g',
54 'casein_100g',
55 'serum-proteins_100g',
56 'nucleotides_100g',
57 'salt_100g',
58 'sodium_100g',
59 'alcohol_100g',
60 'vitamin-a_100g',
61 'beta-carotene_100g',
62 'vitamin-d_100g',
63 'vitamin-e_100g',
64 'vitamin-k_100g',
65 'vitamin-c_100g',
66 'vitamin-b1_100g',
67 'vitamin-b2_100g',
68 'vitamin-pp_100g',
69 'vitamin-b6_100g',
70 'vitamin-b9_100g',
71 'folates_100g',
72 'vitamin-b12_100g',
73 'biotin_100g',
74 'pantothenic-acid_100g',
75 'silica_100g',
76 'bicarbonate_100g',
77 'potassium_100g',
78 'chloride_100g',
79 'calcium_100g',
80 'phosphorus_100g',
81 'iron_100g',
82 'magnesium_100g',
83 'zinc_100g',
84 'copper_100g',
85 'manganese_100g',
86 'fluoride_100g',
87 'selenium_100g',
88 'chromium_100g',
89 'molybdenum_100g',
90 'iodine_100g',
91 'caffeine_100g',
92 'taurine_100g',
93 'ph_100g']
94
95 cols_to_keep = other_cols + features + y
96 cols_to_keep
97 df_open_food_facts = df_open_food_facts[cols_to_keep]

1 #Drop cols with all NA
2 df_open_food_facts = df_open_food_facts.dropna(axis=1, how='all')
3 print("Dropped cols:")
4 set(cols_to_keep) - set(df_open_food_facts.columns)

Dropped cols:
{'-behenic-acid_100g',
'-butyric-acid_100g',
'-capric-acid_100g',
'-caproic-acid_100g',
'-caprylic-acid_100g',
'-cerotic-acid_100g',
'-dihomo-gamma-linolenic-acid_100g',

```

```
'-elaidic-acid_100g',
'-erucic-acid_100g',
'-gondoic-acid_100g',
'-lignoceric-acid_100g',
'-mead-acid_100g',
'-melissic-acid_100g',
'-montanic-acid_100g',
'-myristic-acid_100g',
'-nervonic-acid_100g',
'-palmitic-acid_100g',
'-stearic-acid_100g'}
```

```
1 #Update features list
2 features = list(set(features) - (set(cols_to_keep) - set(df_open_food_facts.columns)))
```

```
1 #Drop any NAN y values
2 df_open_food_facts = df_open_food_facts.dropna(subset=y, how='any')
```

```
1 #Fill NA with 0
2 df_open_food_facts = df_open_food_facts.fillna(0)
```

## Checking Correlations

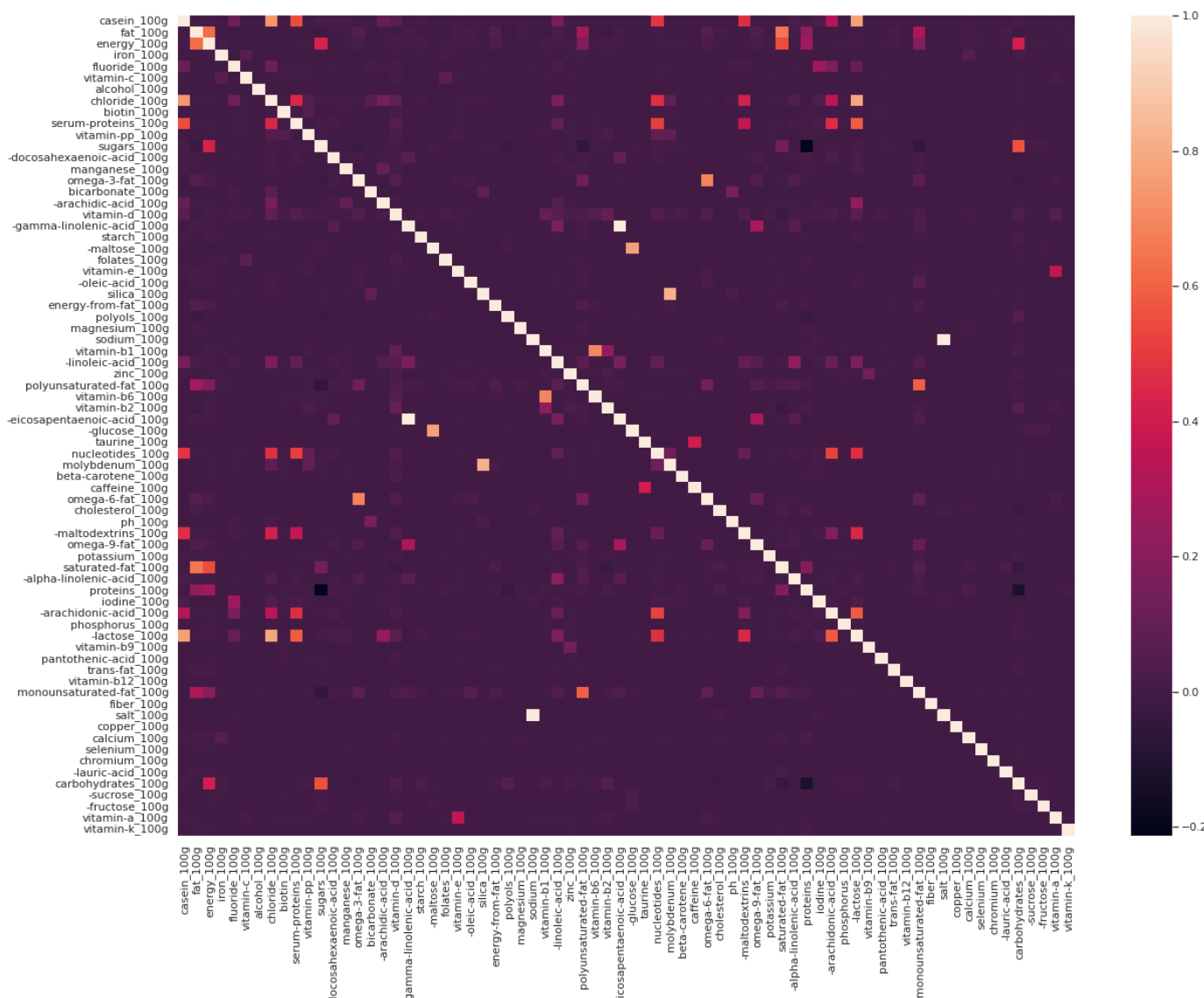
Before performing any modeling, we need to check for any autocorrelations in the features. This is good practice because correlated features have equal prediction power, and we always want models with fewer parameters. But more importantly, if we are performing feature analysis and comparing the magnitude of coefficients, we absolutely don't want correlated features. The coefficients of correlated features will steal magnitude from one another, masking the true prediction power of any one individually.

```
1 #Plot correlation matrix
2 corrs = df_open_food_facts[features].corr()
3 corrs
```

	casein_100g	fat_100g	energy_100g	iron_100g	fluoride_100g	vitamin- c_100g	alcohol_100g	chloride_100g
casein_100g	1.000000	0.004290	0.007135	0.000310	0.120008	0.001572	-0.000051	0.734942
fat_100g	0.004290	1.000000	0.628529	0.000351	-0.001845	-0.005962	-0.001503	0.002406
energy_100g	0.007135	0.628529	1.000000	0.005556	-0.003355	-0.004990	-0.000651	0.002904
iron_100g	0.000310	0.000351	0.005556	1.000000	0.000014	0.070298	-0.000074	0.000368
fluoride_100g	0.120008	-0.001845	-0.003355	0.000014	1.000000	0.000302	-0.000029	0.124918
...	...	...	...	...	...	...	...	...
carbohydrates_100g	0.007427	-0.012993	0.417215	0.012036	-0.002400	0.001425	0.004156	0.004024
-sucrose_100g	0.000629	-0.000727	0.002194	0.000986	-0.000016	-0.000007	-0.000022	0.000260
-fructose_100g	-0.000067	-0.004419	0.001200	-0.000095	-0.000039	0.000002	-0.000046	-0.000100
vitamin-a_100g	0.001697	0.000863	-0.004458	0.000658	0.000216	0.015217	-0.000263	0.002110
vitamin-k_100g	-0.000014	0.001014	0.001459	0.000032	-0.000009	0.000173	-0.000012	-0.000021

72 rows x 72 columns

```
1 #Plot correlation matrix heatmap
2 sns.set(rc={'figure.figsize':(20,15)})
3 sns.heatmap(corrs)
4 plt.show()
```



From above, the most correlated features are:

- Salt and sodium
- Eicosapentaenoic-acid and gamma-linolenic-acid

These make sense:

- Salt is made up of sodium
- Eicosapentaenoic acid and gamma-linolenic acid are similar fatty acids

```
1 #Finding Columns with correlation greater than c
2 c = 0.7
3 corrs_upper_triangle = corrs.where(np.triu(np.abs(corrs), 1).astype(bool))
4 corr_cols = [column for column in corrs_upper_triangle.columns if any(corrs_upper_triangle[column] > c)]
5 corr_cols

['chloride_100g',
'-eicosapentaenoic-acid_100g',
'-glucose_100g',
'molybdenum_100g',
'-lactose_100g',
'salt_100g']

1 #Update features list and drop correlated cols
2 features = list(set(features) - set(corr_cols))
```

12/30/22, 11:13 AM

Final\_Annotated\_Notebook.ipynb - Colaboratory

```
3 df_open_food_facts_clean = df_open_food_facts.drop(columns = corr_cols)

1 #Normalize features
2 df_open_food_facts_clean[features]=(df_open_food_facts_clean[features]-df_open_food_facts_clean[features].min())/(df_o

1 df_open_food_facts_clean.head(3)
```

	product_name	ingredients_text	energy_100g	energy- from- fat_100g	fat_100g	saturated- fat_100g	-lauric- acid_100g	- arachidic- acid_100g	monounsaturated- fat_100g
0	Banana Chips Sweetened (Whole)	Bananas, vegetable oil (coconut oil, corn oil ...	0.009702	0.0	0.14285	0.051945	0.0	0.0	0.0
1	Peanuts	Peanuts, wheat flour, sugar, rice flour, tapio...	0.008395	0.0	0.08930	0.000000	0.0	0.0	0.0
2	Organic Salted Nut Mix	Organic hazelnuts, organic cashews, organic wa...	0.010986	0.0	0.28570	0.009745	0.0	0.0	0.0

3 rows x 69 columns

Modeling EDA and Train/Test Split

```
1 #Plotting histogram of nutrition scores
2 plt.hist(df_open_food_facts_clean['nutrition-score-uk_100g'], bins=20)
3 plt.title('Histogram of Nutrition Scores')
4 plt.xlabel('Nutrition Score')
5 plt.ylabel('Count')
6 plt.show()
```



```
1 #Splitting training and testing sets
2 x_train, x_test, y_train, y_test = train_test_split(df_open_food_facts_clean[features], df_open_food_facts_clean[y], te
```

## Linear Regression

We will first start with a basic linear regression model.

```
8
1 #Train Linear Regression Model
2 from sklearn.linear_model import LinearRegression
3
4 reg = LinearRegression().fit(x_train, y_train)
5
6 train_pred = reg.predict(x_train)
7 test_pred = reg.predict(x_test)
8
9 train_score = reg.score(x_train, y_train)
10 test_score = reg.score(x_test, y_test)
11 train_mse = mean_squared_error(train_pred, y_train)
12 test_mse = mean_squared_error(test_pred, y_test)
```

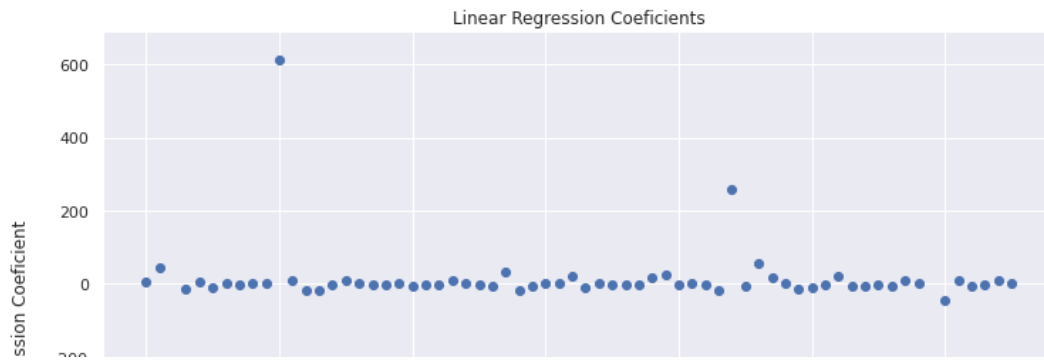
```
1 print("Train R^2: " + str(train_score))
2 print("Test R^2: " + str(test_score))
3 print("Train MSE: " + str(train_mse))
4 print("Test MSE: " + str(test_mse))
```

```
Train R^2: 0.5998875791317069
Test R^2: 0.4175771369443373
Train MSE: 33.49340763591776
Test MSE: 48.88369130004823
```

This model's performance is frankly terrible. The Test  $R^2$  is well below 0.5, and the test MSE is a whopping 49. This means our model, on average, predicts a score that is 6.99 away from the true score. Given that the majority of scores lie between 0 and 20, this margin of error is not acceptable.

```
1 #Coefficients for Feature Importance
2 fig = plt.gcf()
3 fig.set_size_inches(12, 8)
4 plt.scatter(np.arange(len(reg.coef_[0])), reg.coef_[0])
5 plt.xlabel('Feature Number')
6 plt.ylabel('Linear Regression Coefficient')
7 plt.title('Linear Regression Coefficients')
8 plt.show()
```





```
1 #Back out the names of the most powerful features
2 ind = np.where((reg.coef_[0] > 100) | (reg.coef_[0] < -100))
3 df_open_food_facts_clean[features].columns[ind]

Index(['energy_100g', 'sugars_100g', 'saturated-fat_100g', 'chromium_100g'], dtype='object')
```

We can perform feature importance on the model by comparing each feature's coefficient and seeing which features have the most weight in the calculated nutritional score. We can do this because we normalized all features to be on the same scale before training.

Sugars and saturated fat are strong positive factors in nutrition score, and chromium and energy are strong negative factors. These intuitively make sense, but it does not excuse the model's poor performance.

The fact that there are a few very large coefficients and the rest are near zero indicates that we should add regularization to the regression model.

## Ridge Correlations

We will attempt to regularize the model by using ridge regression.

```
1 #Train Ridge Regression model
2 from sklearn.linear_model import Ridge
3
4 reg_ridge = Ridge(alpha=10).fit(x_train, y_train)
5
6 ridge_train_pred = reg_ridge.predict(x_train)
7 ridge_test_pred = reg_ridge.predict(x_test)
8
9 ridge_train_score = reg_ridge.score(x_train, y_train)
10 ridge_test_score = reg_ridge.score(x_test, y_test)
11 ridge_train_mse = mean_squared_error(ridge_train_pred, y_train)
12 ridge_test_mse = mean_squared_error(ridge_test_pred, y_test)
```

```
1 print("Train R^2: " + str(ridge_train_score))
2 print("Test R^2: " + str(ridge_test_score))
3 print("Train MSE: " + str(ridge_train_mse))
4 print("Test MSE: " + str(ridge_test_mse))
```

```
Train R^2: 0.5501210607509373
Test R^2: 0.5600458937065264
Train MSE: 37.65936250212822
Test MSE: 36.926058509113446
```

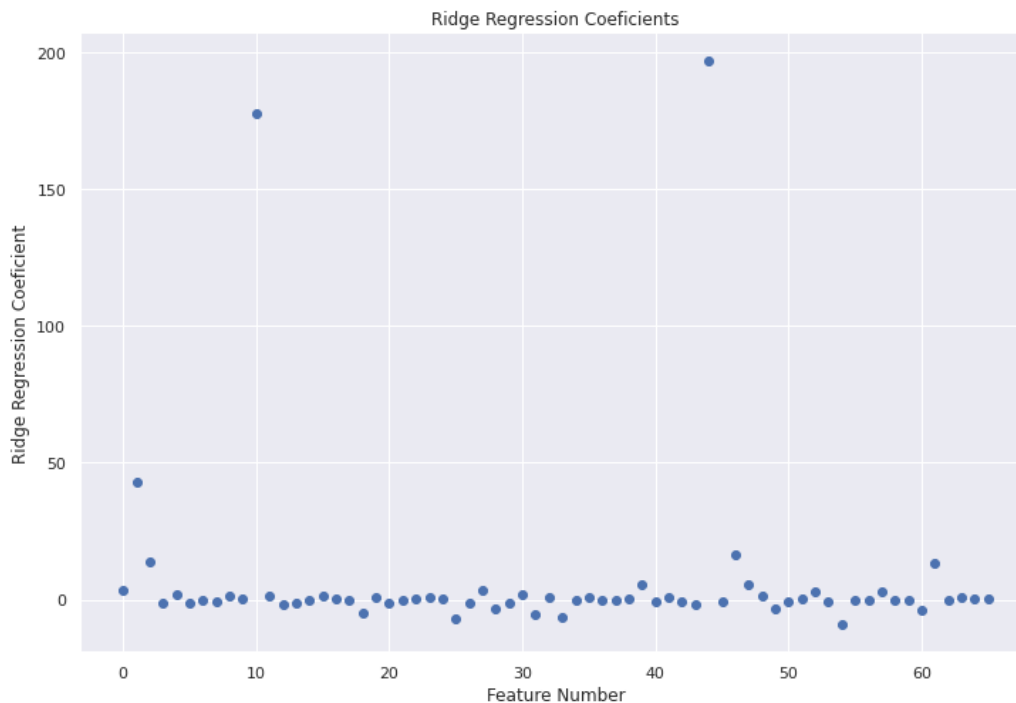
Using Ridge Regression improves the model over a linear baseline. The Test  $R^2$  improves to below 0.56, and the test MSE lowers to 36.9. The model is improving, but perhaps nutritional scores are not formed using linear combinations of the features. We will next explore a different class of models.

```
1 #Coefficients for Feature Importance
2 fig = plt.gcf()
3 fig.set_size_inches(12, 8)
4 plt.scatter(np.arange(len(reg_ridge.coef_[0])), reg_ridge.coef_[0])
5 plt.xlabel('Feature Number')
```

```

6 plt.ylabel('Ridge Regression Coefficient')
7 plt.title('Ridge Regression Coefficients')
8 plt.show()

```



```

1 #Back out the names of the most powerful features
2 ind_ridge = np.where((reg_ridge.coef_[0] > 20 | (reg_ridge.coef_[0] < -20)))
3 df_open_food_facts_clean[features].columns[ind_ridge]

```

```
Index(['fat_100g', 'sugars_100g', 'saturated-fat_100g'], dtype='object')
```

The Ridge Regression model shrinks the coefficients more towards zero with all coefficients being less than 200. The most predictive features for nutritional score in this model are saturated fat, sugar, and fat. Interestingly enough, these three features all have positive coefficients.

## Decision Tree

The model class we will explore next is the decision tree model class. In the case of our problem description, a decision tree model can be expected to perform well. A nutritional score built by thresholding certain nutritional components is a method that makes sense, and one we will explore for our final nutritional model.

```

1 #Training Decision Tree model
2 from sklearn.tree import DecisionTreeRegressor
3
4 dectree = DecisionTreeRegressor(criterion = 'squared_error', random_state=0).fit(x_train, y_train)
5
6 dectree_train_pred = dectree.predict(x_train)
7 dectree_test_pred = dectree.predict(x_test)
8
9 dectree_train_score = dectree.score(x_train, y_train)
10 dectree_test_score = dectree.score(x_test, y_test)
11 dectree_train_mse = mean_squared_error(dectree_train_pred, y_train)
12 dectree_test_mse = mean_squared_error(dectree_test_pred, y_test)

```

```

1 print("Tree Depth: " + str(dectree.get_depth()))
2 print("Number of Leaves: " + str(dectree.get_n_leaves()))

```

```

Tree Depth: 46
Number of Leaves: 65293

```

The trained model has a depth of 46 and 65247 leaves.

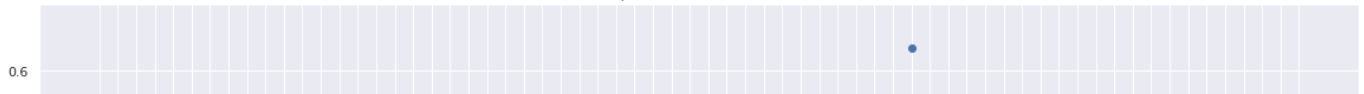
```
1 print("Train R^2: " + str(dectree_train_score))
2 print("Test R^2: " + str(dectree_test_score))
3 print("Train MSE: " + str(dectree_train_mse))
4 print("Test MSE: " + str(dectree_test_mse))
```

```
Train R^2: 0.9994021829076537
Test R^2: 0.9552507310969749
Train MSE: 0.05004326414616448
Test MSE: 3.755878393031536
```

The Decision Tree model performs considerably better compared to our linear models. On the holdout set, the model achieves a high  $R^2$  of 0.955 and an MSE of only 3.8. This means the model, on average, predicts a nutrition score only 1.94 away from the original.

```
1 importances = dectree.feature_importances_
2 plt.scatter(df_open_food_facts_clean[features].columns, importances)
3 plt.xticks(rotation='vertical')
4 plt.title("Feature Importance for Decision Tree Model")
5 plt.ylabel("Mean Decrease in Impunity")
6 plt.xlabel("Feature")
7 plt.show()
```

Feature Importance for Decision Tree Model



We can run a feature importance algorithm on the Decision Tree in order to determine which features factor the most in that data point's nutrition score. The algorithm functions by calculating the weighted probability of a data point reaching that feature's node, and the more data passing through a certain node, the more that node is contributing to the prediction. After running this algorithm, the most important features when calculating nutrition score are shown above. Like the ridge regression model, the decision tree weighs saturated fat and sugars highly, but it also determines salt as an important factor.

## Random Forest (Final Model)

Decision Trees are notorious to overfitting to training data and being unable to generalize to new data coming in. In order to produce more robustness, we will do two things.

1. Use a Random Forest Model with 64 trees. Ensemble methods can help reduce overfitting because they reduce the power that a single tree's specific structure can have.
2. Use cross validation in order to set the hyperparameter of max depth.

```
1 from sklearn.model_selection import KFold
2 from sklearn.ensemble import RandomForestRegressor
3
4 x_rf = df_open_food_facts_clean[features].to_numpy()
5 y_rf = df_open_food_facts_clean[y].to_numpy().transpose()[0]
6
7 num_folds = 5
8 num_trees = 64
9 train_mse_array = []
10 test_mse_array = []
11
12 for max_depth in range(1, 50):
13     print("Starting iteration " + str(max_depth))
14     kfolds = KFold(n_splits=num_folds)
15     fold_train_mse_cum = 0
16     fold_test_mse_cum = 0
17
18     for s in kfolds.split(x_train):
19         train_indices = s[0]
20         test_indices = s[1]
21         rf = RandomForestRegressor(n_estimators = num_trees, criterion = 'squared_error', random_state=0, max_depth=max_depth)
22
23         rf_train_pred = rf.predict(x_rf[train_indices])
24         rf_test_pred = rf.predict(x_rf[test_indices])
25
26         rf_train_mse = mean_squared_error(rf_train_pred, y_rf[train_indices])
27         rf_test_mse = mean_squared_error(rf_test_pred, y_rf[test_indices])
28
29         fold_train_mse_cum += rf_train_mse
30         fold_test_mse_cum += rf_test_mse
31
32 train_mse_array.append(fold_train_mse_cum / num_folds)
33 test_mse_array.append(fold_test_mse_cum / num_folds)
34
```

```
Starting iteration 1
Starting iteration 2
Starting iteration 3
Starting iteration 4
Starting iteration 5
Starting iteration 6
Starting iteration 7
Starting iteration 8
Starting iteration 9
Starting iteration 10
Starting iteration 11
Starting iteration 12
```

```
Starting iteration 13
Starting iteration 14
Starting iteration 15
Starting iteration 16
Starting iteration 17
Starting iteration 18
Starting iteration 19
Starting iteration 20
Starting iteration 21
Starting iteration 22
Starting iteration 23
Starting iteration 24
Starting iteration 25
Starting iteration 26
Starting iteration 27
Starting iteration 28
Starting iteration 29
Starting iteration 30
Starting iteration 31
Starting iteration 32
Starting iteration 33
Starting iteration 34
Starting iteration 35
Starting iteration 36
Starting iteration 37
Starting iteration 38
Starting iteration 39
Starting iteration 40
Starting iteration 41
Starting iteration 42
Starting iteration 43
Starting iteration 44
Starting iteration 45
Starting iteration 46
Starting iteration 47
Starting iteration 48
Starting iteration 49
```

```
1 #Plotting Training and Testing Error
2 fig = plt.gcf()
3 fig.set_size_inches(12, 8)
4 plt.plot(np.arange(1, 50), train_mse_array, color='blue', label='Train MSE')
5 plt.plot(np.arange(1, 50), test_mse_array, color='red', label='Test MSE')
6 plt.title("Max Depth vs. Average MSE Over 5-Fold Cross Validation")
7 plt.xlabel("Max Depth")
8 plt.ylabel("Average MSE Across Folds")
9 plt.legend()
10 plt.show()
```

## Max Depth vs. Average MSE Over 5-Fold Cross Validation

The test MSE bottoms out when the max depth reaches about 25. In order to prevent overfitting, we want to choose the smallest max depth that gives us the best performance on our testing data. Thus we should choose a max depth of 25 in our final model.

```

35
1 #Train final Random Forest model
2 from sklearn.ensemble import RandomForestRegressor
3
4 y_train_1d = y_train.to_numpy().transpose()[0]
5 rf = RandomForestRegressor(n_estimators = 64, criterion = 'squared_error', random_state=0, max_depth=25).fit(x_train, y_train)
6
7 rf_train_pred = rf.predict(x_train)
8 rf_test_pred = rf.predict(x_test)
9
10 rf_train_score = rf.score(x_train, y_train)
11 rf_test_score = rf.score(x_test, y_test)
12 rf_train_mse = mean_squared_error(rf_train_pred, y_train)
13 rf_test_mse = mean_squared_error(rf_test_pred, y_test)

1 print("Train R^2: " + str(rf_train_score))
2 print("Test R^2: " + str(rf_test_score))
3 print("Train MSE: " + str(rf_train_mse))
4 print("Test MSE: " + str(rf_test_mse))

Train R^2: 0.9949094948768022
Test R^2: 0.9759388647997611
Train MSE: 0.42612614423213113
Test MSE: 2.019489927449483

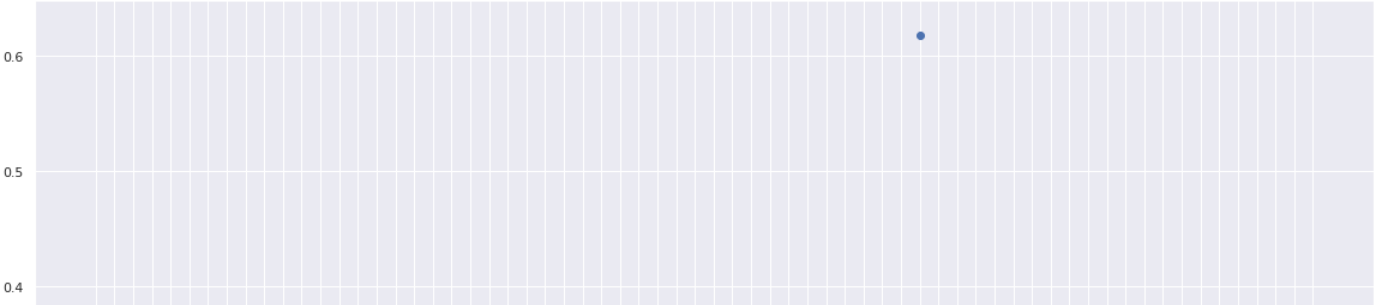
```

The final random forest model has improved on all our previous models. The  $R^2$  on the test set is 0.98, and the MSE on the testing set is down to 2.02, meaning the average predicted score for a particular food item is merely 1.42 away from its actual score.

```

1 rf_importances = rf.feature_importances_
2 fig = plt.gcf()
3 fig.set_size_inches(20, 12)
4 plt.scatter(df_open_food_facts_clean[features].columns, rf_importances)
5 plt.xticks(rotation='vertical')
6 plt.show()

```



With this final model, we can extrapolate the most important features and learn what factors into the UK Nutritional Score calculation. Saturated fat remains the most important factor, followed by sugars and sodium. Energy, proteins, and carbohydrates make up the last three most significant factors

