# Solution Design

# **Downloading the USDA Food Data**

Food and nutrient information is available on the US Department of Agriculture website.

The USDA SR25 data shows different foods with a number of identifying attributes along with two lists of nutrients and portion sizes. Data in this form is not particularly pliable to analysis, therefore, there is need to do some work to wrangle the data into a better form. In order, to improve the formatting and presentation of the data, there is need to first convert the data to the JSON format before proceeding. Programmer Schirinos made available a version of the USDA database for easy exporting to in JSON format or mongo database format through his git files. The files are to be downloaded from the nutrient-db python utility from GitHub from <https://github.com/schirinos/nutrient-db.git>

# **Converting the USDA SR25 data to a JSON file**

After downloading and extracting the data to a folder on the desktop, the data is converted to a json file using the Anaconda3 command prompt. I used Window 10 Pro computer, and installed Anaconda3-5.0.1-Windows-x86\_64, which comes with Spyder 3.2.4 and Python 3.6.3. Since, the python version used is python 3, there was need to convert the nutrientdb.py code written in python 2.7 to python 3 compatible code. The following codes were used in converting the nutrientdb.py to nutrients. json:

There is need to first install the pymongo required to Schrinos’ nutrientdb.py file in python. Secondly, there is need to use the 2to3 script from Anaconda3 scripts as follows:

python 2to3-script.py -w nutrientdb.py

The code outputs a python 3 coded nutrientdb.py that was then used to create the JSON file. The code importing pymongo, which need to be fetched using:

pip install pymongo

After, the installation the pymongo the JSON file is created by the following code:

python nutrientdb.py -e > nutrients.json

3.Loading the JSON file into Python

Loading the JSON file into python can be done with any JSON library. However, for this program, the Pandas built-in json class was recommended and used.

df = pd.read\_json('nutrients.json', orient='records', typ='frame',

dtype=False, convert\_axes=True, convert\_dates=True,

keep\_default\_dates=True, numpy=False, precise\_float=True,

date\_unit=None, encoding='utf-8', lines=True)

The pd.read\_json is blackbox command that was giving a trailing error, which proved difficult to address. Therefore, I resorted to using the pandas’ documentation from

<http://pandas.pydata.org/pandas-docs/version/0.21/generated/pandas.read_json.html?highlight=read_json#pandas.read_json>

Using the pandas documentation as guide, the pd.read\_json() was used on the nutrients.json file. Orient string value of ‘column’ was used to indicate the expected JSON string format. The column orient produces a dictionary like json presented as:

'columns’: dict like {column -> {index -> value}}

The value of the typ parameter was set to ‘series’. While, the convert the axes was left at True to ensure that the axes were converted to proper dtypes. While, numpy Boolean was left at False, since the JSON ordering may not be the same for each term. The lines Boolean was set True, which returns the JsonReader object for iteration through line-delimitation. Line delimitation is a relatively new function in pandas, which was introduced in version 0.21.0. The code for loading the json file into python was as follows:

df = pd.read\_json('nutrients.json', orient='column', typ='series', dtype=True,

convert\_axes=True, numpy=False, precise\_float=True,

encoding='utf-8', lines=True)

The result is a nested json file, which is flattened by importing it with json\_normalize from panda.io.json. This results in more readable dataframe, which increases the columns from 6 in df to 17 in data, thus making it easier to extract values from fields such as food names, group, id, and manufacturer. This was done as follows:

First we json normalize the data, with the following code, the logic behind the code is based on the example given in pandas documentation. The example is given in the readme file.

df\_normalized = json\_normalize(df, 'nutrients',

['group', 'manufacturer',

['meta', 'ndb\_no'],

['name', 'long']])

df\_normalized[:10]

Second, we need to select the columns that wll appear in the data frame table.To do this we have to create an info\_keys that will ensure that the right food data dictionary is included in our DataFrame. The code is as follows:

info\_keys = ['meta.ndb\_no', 'name.long', 'group', 'manufacturer',

'name', 'value']

info = pd.DataFrame(df\_normalized, columns=info\_keys)

info.keys()

For clarity the columns of the DataFrame objects are renamed using column mapping

col\_mapping = {'name.long': 'food', 'group': 'fgroup',

'meta.ndb\_no': 'id', 'name': 'nutrient',

'value': 'nutrient.value'}

usda\_data = info.rename(columns=col\_mapping, copy=False)

After creating our dataframe we need to check if the right information has been included and we will also use the info to get the range index

usda\_data.info()

Having seen the range index, we pick 30000, which is a point within our range. We then use the output information for analysis

usda\_data.iloc[30000]

# 4. For the ‘Amino Acids’ nutrient group, an output a table is generated showing the different constituents of the group.(Alanine, Glycine, Histidine etc) and the foods in which they are present (Gelatins, dry powder, beluga, meat...etc).

If everything is were it should be, then the next stage would be to present the table of Amino Acid group nutrients by nutrients and food. However, the data used was the SR25 and the data did not have a Nutrient group. After, a futile effort to create the group. The table presented is for nutrient group by food groups with nutrient values

nutrient fgroup

10:0 American Indian/Alaska Native Foods 0.0000

Baby Foods 0.0010

Baked Products 0.0000

Beef Products 0.0050

Beverages 0.0000

Breakfast Cereals 0.0000

Cereal Grains and Pasta 0.0000

Dairy and Egg Products 0.0755

Fast Foods 0.0470

Fats and Oils 0.0000

Finfish and Shellfish Products 0.0000

Fruits and Fruit Juices 0.0000

Lamb, Veal, and Game Products 0.0120

Legumes and Legume Products 0.0000

Meals, Entrees, and Side Dishes 0.0080

Nut and Seed Products 0.0000

Pork Products 0.0060

Poultry Products 0.0000

Restaurant Foods 0.0080

Sausages and Luncheon Meats 0.0200

Snacks 0.0100

Soups, Sauces, and Gravies 0.0010

Spices and Herbs 0.0000

Sweets 0.0140

Vegetables and Vegetable Products 0.0000

12:0 American Indian/Alaska Native Foods 0.0000

Baby Foods 0.0070

Baked Products 0.0010

Beef Products 0.0060

Beverages 0.0000

Water Snacks 4.2200

Soups, Sauces, and Gravies 85.9000

Spices and Herbs 9.3100

Sweets 9.0400

Vegetables and Vegetable Products 89.2100

Zinc, Zn American Indian/Alaska Native Foods 1.0450

Baby Foods 0.4900

Baked Products 0.6000

Beef Products 5.1900

Beverages 0.0300

Breakfast Cereals 2.7000

Cereal Grains and Pasta 1.0900

Dairy and Egg Products 0.8300

Fast Foods 1.2400

Fats and Oils 0.0000

Finfish and Shellfish Products 0.6600

Fruits and Fruit Juices 0.0900

Lamb, Veal, and Game Products 3.5650

Legumes and Legume Products 1.0000

Meals, Entrees, and Side Dishes 0.7500

Nut and Seed Products 3.3100

Pork Products 2.2900

Poultry Products 2.0400

Restaurant Foods 0.7400

Sausages and Luncheon Meats 2.1100

Snacks 1.4200

Soups, Sauces, and Gravies 0.2800

Spices and Herbs 2.8400

Sweets 0.3600

Vegetables and Vegetable Products 0.3300

Name: nutrient.value, Length: 3553, dtype: float64

# 5. For all the different nutrient group (beef Products, Pork Products, dairy and egg products etc.), The median of the zinc content in all the foods that constitute the nutrient group is then calculated and presented.

n\_med = usda\_data.groupby(['nutrient',

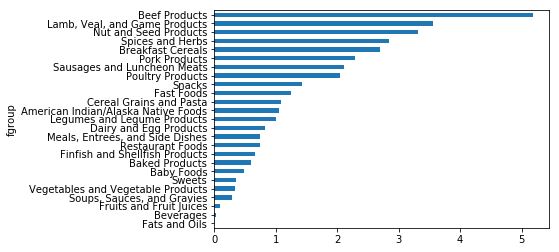
'fgroup'])['nutrient.value'].quantile(0.5)

6. Lastly, the program plots the distribution of median Zinc Content for different nutrient groups as a bar chart.

The plot the median Zinc Values is coded as follows:

n\_med['Zinc, Zn'].sort\_values().plot(kind='barh')

print("----Median zinc values by nutrient group-----")



The output files are with inline graphs and tables are attached.

The code is json\_project.py is also attached.

The readme file is also attached. The project relied mostly on pandas documentation and the read me file includes procedures for the using python 3 on python 2 data, and installation of additional modules.

The code was done in Windows 10 Pro, using Anacoda3 for Python 3.6 with Spyder as the IDLE. Please take note that most of the methods and classes used in the Pandas require versions later than 0.21.0.