main

September 29, 2024

1 Midterm 1, Spring 2022: USDA Food Labels

Version 1.2

Solution

Changes:

- 1.1 - Clarified instructions for exercise 6
- 1.2 - Corrected typos in exercise 2 and exercise 4

This problem builds on your knowledge of nested data structures, string processing, and implementation of mathematical functions. It has 7 exercises, numbered 0 to 6. There are 14 available points. However, to earn 100% the threshold is 11 points. (Therefore, once you hit 11 points, you can stop. There is no extra credit for exceeding this threshold.)

Each exercise builds logically on previous exercises, but you may solve them in any order. That is, if you can't solve an exercise, you can still move on and try the next one. Use this to your advantage, as the exercises are **not** necessarily ordered in terms of difficulty. Higher point values generally indicate more difficult exercises.

Code cells starting with the comment `### Loading results` load the result of applying the **correct** solution as described in the markdown cell above it. These can be used to follow along with the overall analysis, but they are not required to be run to move on. Most of these cells are loading rather large data structures (at least in terms of what's human-readable). As such, the data is only loaded - not printed. You are free to use Python to explore it further.

The point values of individual exercises are as follows:

- Exercise 0: 1 point (This one is a freebie!)
- Exercise 1: 1 point
- Exercise 2: 4 points
- Exercise 3: 1 point
- Exercise 4: 2 points
- Exercise 5: 3 points
- Exercise 6: 2 points

We will be working with some data about food. The foods you see in the grocery store are required to have nutrition labels, which provide information to consumers about which nutrients are present in a food, the amounts present, and the ingredients in that food (listed in order of content). The USDA maintains all of this information and makes it available to the public on their website - we are using the "Global Branded Foods" data from October 2021 in this notebook.

The file linked on the website may change (hopefully this link will work in the future) - https://fdc.nal.usda.gov/fdc-datasets/FoodData_Central_branded_food_json_2021-10-28.zip.

Note to keep the runtimes shorter, we are working with a *sample* of the source data. This will not affect any of the functionality we will be developing in this notebook.

1.1 Exercise 0 (1 Points):

After downloading and unzipping the file, you will find the data stored in a **JSON** format. You may or may not have seen this type of data format before, so we will take care of reading the data into our Python environment. The result of the code below is that `food_lod` will load the serialized contents of the json file into Python objects - in this case, a `list` of `dicts`.

Run the test cell below to load the data. We are treating this as a "test" cell, so you will get one point for just submitting. How generous!

```
In [1]: ###
        ### AUTOGRADER TEST - DO NOT REMOVE
        ###

        import json                                                # import the json module
        path = './resource/asnlib/publicdata/food_data.json' # path to the data file
        with open(path, 'r') as file:                              # open the path and keep the file
            food_lod = json.load(file)                             # load the file into a variable i
```

1.2 Exploring the data

Let's start by taking a look at some of the basic attributes about this data.

```
In [2]: {
            'type': type(food_lod),
            'length': len(food_lod),
            'value_types': {type(v) for v in food_lod}
        }

Out[2]: {'type': list, 'length': 30000, 'value_types': {dict}}
```

Well... `food_lod` is a `list`, with 30,000 entries, and each of those are of type `dict`. Let's take a look at some of the keys in one of the `dicts`.

```
In [3]: food_lod[0].keys()

Out[3]: dict_keys(['foodClass', 'description', 'foodNutrients', 'foodAttributes', 'modifiedDat
```

1.3 Exercise 1 (1 Points):

These look like some promising candidates for extracting information about individual foods. There appear to be some "category" related keys, which may be useful for grouping foods and comparing between groups as well. For further analysis, we want to know if the dicts are all of similar structure to the first one. A good start to analyzing this is determining which keys are common to all of them...

Given an input lod, which is a list of dicts complete the function common_keys(lod) to return a Python set of the keys which are common to all of the dicts in lod.

```
In [4]: ### Define common_keys
        def common_keys(lod):
            ###
            common_keys = set.intersection(*map(set, lod)) #*map unpacks the map
            return common_keys
            ###
```

The demo cell below should display the following output:

```
{'bar', 'qux'}
```

```
In [5]: ### define demo inputs
        demo_key_list_ex0 = ['foo', 'bar', 'baz', 'qux', 'tav', 'wot']
        demo_lod_ex0 = [
            {k: 1 for k in demo_key_list_ex0[1:]},
            {k: 1 for k in demo_key_list_ex0[:-1]},
            {k: 1 for k in demo_key_list_ex0[1::2]}
        ]
```

```
In [6]: ### call demo funtion
        common_keys(demo_lod_ex0)

Out[6]: {'bar', 'qux'}
```

The cell below will test your solution for Exercise 1. The testing variables will be available for debugging under the following names in a dictionary format. - input_vars - Input variables for your solution. - original_input_vars - Copy of input variables from prior to running your solution. These *should* be the same as input_vars - otherwise the inputs were modified by your solution. - returned_output_vars - Outputs returned by your solution. - true_output_vars - The expected output. This *should* "match" returned_output_vars based on the question requirements - otherwise, your solution is not returning the correct output.

```
In [7]: ### test_cell_ex1


        ###
        ### AUTOGRADER TEST - DO NOT REMOVE
        ###

        from tester_fw.testers import Tester_ex1
        tester = Tester_ex1()
        for _ in range(20):
            try:
                tester.run_test(common_keys)
                (input_vars, original_input_vars, returned_output_vars, true_output_vars) = tes
            except:
                (input_vars, original_input_vars, returned_output_vars, true_output_vars) = tes
                raise

        ###
        ### AUTOGRADER TEST - DO NOT REMOVE
        ###
        print('Passed! Please submit.')
```

```
initializing tester_fw.tester_6040
Passed! Please submit.
```

Even if your solution was incorrect or you skipped this exercise, run this cell to see the expected output of a call to `common_keys(food_lod)`.

```
In [8]: ### Loading results
        import pickle
        import os
        path = './resource/asnlib/publicdata/ex1.pkl'
        if not os.path.exists(path):
            with open(path, 'wb') as file:
                pickle.dump(common_keys(food_lod), file)
        with open(path, 'rb') as file:
            food_keys = pickle.load(file)
```

1.4 Exercise 2 (4 Points):

For our analysis, we are interested in the nutritional content and ingredients contained in each food. Additionally we would like to group foods by the categories given. The keys of interest are 'description', 'ingredients', 'labelNutrients', and 'brandedFoodCategory'.

```
In [9]: keys_of_interest = {'description', 'ingredients', 'labelNutrients', 'brandedFoodCatego
        {k:v for k, v in food_lod[0].items() if k in keys_of_interest}
```

```
Out[9]: {'description': 'KETTLE COOKED POTATO CHIPS, PINK HIMALAYAN SALT & RED WINE VINEGAR',
         'ingredients': 'POTATOES, VEGETABLE OIL (CONTAINS ONE OR MORE OF THE FOLLOWING: CANOL
         'labelNutrients': {'fat': {'value': 7.0},
          'saturatedFat': {'value': 0.501},
          'transFat': {'value': 0.0},
          'cholesterol': {'value': 0.0},
          'sodium': {'value': 140},
          'carbohydrates': {'value': 17.0},
          'fiber': {'value': 1.01},
          'sugars': {'value': 0.0},
          'protein': {'value': 2.0},
          'calcium': {'value': 0.0},
          'iron': {'value': 0.4},
          'potassium': {'value': 319},
          'addedSugar': {'value': 0.0},
          'calories': {'value': 140}},
         'brandedFoodCategory': 'Chips, Pretzels & Snacks'}
```

Define `extract_basic_data` to meet the following requirements. Given a `list` of `dicts`, `lod`, create a **new** `list` of `dicts` called `basic_data`. For each `dict` in `lod`, there should be a corresponding `dict` in `basic_data` with the following key/value pairs: - `'description'` - `str` associated with `'description'` in the `lod` `dict`. - `'list_of_ingredients'` - `list` of all the ingredients associated with `'ingredients'` in the `lod` `dict`. *See "Notes on ingredients" below.* - `'raw_nutrients'` - `dict` mapping the nutrient name (`str`) to it's amount (`float`). *See "Notes on nutrients" below.* - `'category'` - `str` associated with `'brandedFoodCategory'` in the `lod` `dict`.

Notes on ingredients - For each `dict`, `d` in `lod` the **ingredients** are stored as a `str` associated with the `'ingredients'` key.
- Sometimes there is extra information wrapped in parentheses. We do not want to include this information in our analysis, so any text wrapped in `()` (and the parentheses themselves) should be left out of further processing. There may be **multiple** sets of parentheses in an `ingredients` string. - You can assume that there are not **nested** parentheses. For example strings of this form **will not** occur - `'item, item1 (level 1 (another, level)), item2.'` - The `re` module may be helpful here. - Note that there can be *anything* in between the parentheses and all of that text should be discarded. For example `'ingredient 1, ingredient 2 (ingredient 2.1, ingredient 2.2, [ingredient 2.2.1, ingredient 2.2.2]), ingredient 3.'` should result in just `['ingredient 1', 'ingredient 2', 'ingredient 3']` as it's associated `'list_of_ingredients'`. - The ingredient string ends in `'.'`, which should also be left out of further processing.
- The individual ingredients are separated by `', '`. - The ingredients in `basic_data[i]['list_of_ingredients']` should not have any leading or trailing whitespace.

Notes on nutrients - For each `dict`, `d`, in `lod`, the nutrients are associated with the `'labelNutrients'` key. - `d['labelNutrients']` is a dictionary of the form `{'protein': {'value': 10}, 'riboflavin': {'value': 2}}`, i.e. mapping the nutrient name to a dictionary with one key (`'value'`) which is mapped to the amount of that nutrient present in a particular food.

```
In [10]: ### Define extract_basic_data
         def extract_basic_data(lod):
             ###
             #What are we returning?
             basic_data = []

             #loop through each dict in lod
             for label in lod:
                 label_dict = {}
             #Extract the keys
```

```
        label_dict['description'] = label['description']
        label_dict['category'] = label['brandedFoodCategory']
    #Ingredients -- turns into a list
        clean_ingr = clean_ingredients(label['ingredients']).split(", ")
        label_dict['list_of_ingredients'] = [i.strip() for i in clean_ingr]
    #Nutrients -- just get the value
        label_dict['raw_nutrients'] = {k:v['value'] for k, v in label['labelNutrients
    #Append the dicts with the extracted keys to basic_data
        basic_data.append(label_dict)

    return basic_data

#Clean function for cleaning ingredients string
def clean_ingredients(ing_str):
    #import re (reg ex)
    import re
    #Remove text within ()
    print(ing_str)
    clean = re.sub("\(.*?\)", "", ing_str)#re.sub(pattern, replacement, target_string
    print(clean)
#Strip whitespace
    clean = clean.strip()

#remove the final period
    clean = clean[:-1]

    return clean

    ###
```

The demo cell below should display the following output:

```
[{'description': 'KETTLE COOKED POTATO CHIPS, PINK HIMALAYAN SALT & RED WINE VINEGAR',
  'list_of_ingredients': ['POTATOES',
   'VEGETABLE OIL',
   'MALTODEXTRIN',
   'HIMALAYAN SALT',
   'RED WINE VINEGAR',
   'CITRIC ACID',
   'SUGAR',
   'WHITE DISTILLED VINEGAR',
   'NATURAL FLAVOR'],
  'raw_nutrients': {'fat': 7.0,
   'saturatedFat': 0.501,
   'transFat': 0.0,
   'cholesterol': 0.0,
   'sodium': 140.0,
   'carbohydrates': 17.0,
   'fiber': 1.01,
   'sugars': 0.0,
   'protein': 2.0,
   'calcium': 0.0,
   'iron': 0.4,
   'potassium': 319.0,
   'addedSugar': 0.0,
   'calories': 140.0},
  'category': 'Chips, Pretzels & Snacks'},
 {'description': 'TOMATO BASIL PASTA SAUCE',
  'list_of_ingredients': ['TOMATO PUREE',
   'TOMATOES',
   'SUGAR',
   'SOYBEAN OIL',
   'SALT',
   'DRIED ONIONS',
   'DRIED GARLIC',
   'SPICES',
   'LEMON JUICE CONCENTRATE',
   'ROMANO CHEESE'],
  'raw_nutrients': {'fat': 2.0,
   'sodium': 580.0,
   'carbohydrates': 17.0,
   'fiber': 2.94,
   'sugars': 10.0,
   'protein': 3.0,
   'calcium': 29.4,
   'iron': 0.998,
   'potassium': 750.0,
   'addedSugar': 2.05,
   'calories': 89.6},
  'category': 'Prepared Pasta & Pizza Sauces'}]
```

```
In [11]: ### define demo inputs
         keys_of_interest = {'description', 'ingredients', 'labelNutrients', 'brandedFoodCatego
         demo_lod_ex1 = [{k:v for k, v in d.items() if k in keys_of_interest} for d in food_lod
```

```
In [12]: ### call demo funtion
         extract_basic_data(demo_lod_ex1)
```

```
POTATOES, VEGETABLE OIL (CONTAINS ONE OR MORE OF THE FOLLOWING: CANOLA OIL, SAFFLOWER OIL AND/
POTATOES, VEGETABLE OIL , MALTODEXTRIN, HIMALAYAN SALT, RED WINE VINEGAR, CITRIC ACID, SUGAR, 
TOMATO PUREE (WATER, TOMATO PASTE), TOMATOES, SUGAR, SOYBEAN OIL, SALT, DRIED ONIONS, DRIED GA
TOMATO PUREE , TOMATOES, SUGAR, SOYBEAN OIL, SALT, DRIED ONIONS, DRIED GARLIC, SPICES, LEMON J
```

```
Out[12]: [{'description': 'KETTLE COOKED POTATO CHIPS, PINK HIMALAYAN SALT & RED WINE VINEGAR'
           'category': 'Chips, Pretzels & Snacks',
           'list_of_ingredients': ['POTATOES',
            'VEGETABLE OIL',
            'MALTODEXTRIN',
            'HIMALAYAN SALT',
            'RED WINE VINEGAR',
            'CITRIC ACID',
            'SUGAR',
            'WHITE DISTILLED VINEGAR',
            'NATURAL FLAVOR'],
           'raw_nutrients': {'fat': 7.0,
            'saturatedFat': 0.501,
            'transFat': 0.0,
            'cholesterol': 0.0,
            'sodium': 140,
            'carbohydrates': 17.0,
            'fiber': 1.01,
            'sugars': 0.0,
            'protein': 2.0,
            'calcium': 0.0,
            'iron': 0.4,
            'potassium': 319,
            'addedSugar': 0.0,
            'calories': 140}},
          {'description': 'TOMATO BASIL PASTA SAUCE',
           'category': 'Prepared Pasta & Pizza Sauces',
           'list_of_ingredients': ['TOMATO PUREE',
            'TOMATOES',
            'SUGAR',
            'SOYBEAN OIL',
            'SALT',
            'DRIED ONIONS',
            'DRIED GARLIC',
            'SPICES',
            'LEMON JUICE CONCENTRATE',
            'ROMANO CHEESE'],
           'raw_nutrients': {'fat': 2.0,
            'sodium': 580,
            'carbohydrates': 17.0,
            'fiber': 2.94,
            'sugars': 10.0,
            'protein': 3.0,
            'calcium': 29.4,
            'iron': 0.998,
            'potassium': 750,
            'addedSugar': 2.05,
            'calories': 89.6}}]
```

The cell below will test your solution for Exercise 2. The testing variables will be available for debugging under the following names in a dictionary format. - `input_vars` - Input variables for your solution. - `original_input_vars` - Copy of input variables from prior to running your solution. These *should* be the same as `input_vars` - otherwise the inputs were modified by your solution. - `returned_output_vars` - Outputs returned by your solution. - `true_output_vars` - The expected output. This *should* "match" `returned_output_vars` based on the question requirements - otherwise, your solution is not returning the correct output.

```
In [13]: ### test_cell_ex2

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###

         from tester_fw.testers import Tester_ex2
         tester = Tester_ex2()
         for _ in range(20):
             try:
                 tester.run_test(extract_basic_data)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                 raise

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
         print('Passed! Please submit.')
```

```
initializing tester_fw.tester_6040
XGUNQYBHYDBFWV, D, AQFAPSO, GPJBNDSWOUNERY.
XGUNQYBHYDBFWV, D, AQFAPSO, GPJBNDSWOUNERY.
OEGWZEZ, PUOIWUASFW, PIUSLDKGFGFO.
OEGWZEZ, PUOIWUASFW, PIUSLDKGFGFO.
I, UIVCRP (ZMNPFTMC), OJLRJBWKPROAHW (MJQHDFHLEDBZPQY, HH).
I, UIVCRP , OJLRJBWKPROAHW .
YSMBQKHHVPFRNVH, DXZQVVY, ELPPYTLNPMD, ZPOZMLOVEDLBSOQ, LSCKBEOGFUJTLGS (LINQIBODNU).
YSMBQKHHVPFRNVH, DXZQVVY, ELPPYTLNPMD, ZPOZMLOVEDLBSOQ, LSCKBEOGFUJTLGS .
RJBIBD, JJSKDQBYBVUPGXS (D, PSZKBCBVSTPP, EGGUTELBOQCV), AGCPTGPXFRDDCO, WBSFNHSHNBUDVPT, PR.
RJBIBD, JJSKDQBYBVUPGXS , AGCPTGPXFRDDCO, WBSFNHSHNBUDVPT, PR.
N, GFBBNX, BJARQC.
N, GFBBNX, BJARQC.
ZWNIBFZZQZAXVON, KRZ, SRNVZRKWG, SINZFKCS, YEDVKNJSP (MOKWK, M).
ZWNIBFZZQZAXVON, KRZ, SRNVZRKWG, SINZFKCS, YEDVKNJSP .
HWHLAJOLWQ, WUHYAKIN, UYVIV (PU, W, VAOX), VYBMVWBNYT, BVTNJLMG.
HWHLAJOLWQ, WUHYAKIN, UYVIV , VYBMVWBNYT, BVTNJLMG.
YWHOAZLOGC, THTPEKLTQCMREO, KTH.
YWHOAZLOGC, THTPEKLTQCMREO, KTH.
XIKABJHRBSA, OYSRCJS, D, BS (UETKU), MLH.
XIKABJHRBSA, OYSRCJS, D, BS , MLH.
```

SMLKTYPV, RHELP, LJ, SRYLRUSTB.
SMLKTYPV, RHELP, LJ, SRYLRUSTB.
HDXTTJIYJHFBQF, HMHB (ZLLUTUCDPXLJGS, L), HKBFWSYPIRN, ADDUFZ, YXO.
HDXTTJIYJHFBQF, HMHB , HKBFWSYPIRN, ADDUFZ, YXO.
SP, VWWDSW, UABYCOAXEHLRVLHY.
SP, VWWDSW, UABYCOAXEHLRVLHY.
CXMAO (NXYDKIP, KGBXZYXNCVYL, SP), BEWDNHOSWBNZEWG, YYQ, YYD, TJPOWOOI.
CXMAO , BEWDNHOSWBNZEWG, YYQ, YYD, TJPOWOOI.
OCKT, QGZNUNUWORYURR, HSGH, LBZ.
OCKT, QGZNUNUWORYURR, HSGH, LBZ.
WQUTDLPCPXXI, AWYWPZPAVRYKGTGV, QFP, EQYPOSAGOTP.
WQUTDLPCPXXI, AWYWPZPAVRYKGTGV, QFP, EQYPOSAGOTP.
XGZBESVOF (XHIU, GRX), ERWQMWNCUBHEYPPX, UF.
XGZBESVOF , ERWQMWNCUBHEYPPX, UF.
RSSHUOEJOP, YRIIMZREMQCQ, XNMNMSCNIYAJBNJ, ML.
RSSHUOEJOP, YRIIMZREMQCQ, XNMNMSCNIYAJBNJ, ML.
LLBHQPGSRI, YCTUTNGA, GOCVRZHOLH, INSUCBJHDS.
LLBHQPGSRI, YCTUTNGA, GOCVRZHOLH, INSUCBJHDS.
BPEZBS, RVGALJBADHRDXPLI, TRSXC, AGD.
BPEZBS, RVGALJBADHRDXPLI, TRSXC, AGD.
TRJF, B, IJP, VXCNOYLS (NRJCRUG, RGRP), EHGC (DPWOEXMIMI, FBE).
TRJF, B, IJP, VXCNOYLS , EHGC .
FOFTOIWF, PNKTHSRJ, AKITQKNW.
FOFTOIWF, PNKTHSRJ, AKITQKNW.
JMQ, RXEAVFPVPNTZGCL, DJPGXS, LNZZEFHCZHEMSB.
JMQ, RXEAVFPVPNTZGCL, DJPGXS, LNZZEFHCZHEMSB.
AVDOON, DVDJNUZSGYAPZCON, NIJCXJYHIRN, JGKWHVSIK, TKSCOLOTJUL.
AVDOON, DVDJNUZSGYAPZCON, NIJCXJYHIRN, JGKWHVSIK, TKSCOLOTJUL.
FFFOXW (CRQOVVJY, X), WAGVPKXQUWZFD, VFCOFXOQKFRFOP, EJ.
FFFOXW , WAGVPKXQUWZFD, VFCOFXOQKFRFOP, EJ.
YMTMTWPSPAF, QGLOHOD, IKMMTDMJRE, NXRZSBXJQWHUTFM.
YMTMTWPSPAF, QGLOHOD, IKMMTDMJRE, NXRZSBXJQWHUTFM.
BZELCVAQ (QYLNBX), KEMQIYJQATKR, XUFJ, FZDRG, DBJHHCPXCGIR.
BZELCVAQ , KEMQIYJQATKR, XUFJ, FZDRG, DBJHHCPXCGIR.
ZGQFZ, WUEC, YRMAY.
ZGQFZ, WUEC, YRMAY.
J, NWNULZMNNWXWOJN, IRKFZEVBRRDTJ, N.
J, NWNULZMNNWXWOJN, IRKFZEVBRRDTJ, N.
FKFDMTEOFPNWRHVU, INHQSW, OY.
FKFDMTEOFPNWRHVU, INHQSW, OY.
NXUFYRHJPKXRMFA, RW, JCIQ, GLPIEESADTWVUG, M.
NXUFYRHJPKXRMFA, RW, JCIQ, GLPIEESADTWVUG, M.
SMLKTYPV, RHELP, LJ, SRYLRUSTB.
SMLKTYPV, RHELP, LJ, SRYLRUSTB.
HDXTTJIYJHFBQF, HMHB (ZLLUTUCDPXLJGS, L), HKBFWSYPIRN, ADDUFZ, YXO.
HDXTTJIYJHFBQF, HMHB , HKBFWSYPIRN, ADDUFZ, YXO.
SP, VWWDSW, UABYCOAXEHLRVLHY.
SP, VWWDSW, UABYCOAXEHLRVLHY.

CXMAO (NXYDKIP, KGBXZYXNCVYL, SP), BEWDNHOSWBNZEWG, YYQ, YYD, TJPOWOOI.
CXMAO , BEWDNHOSWBNZEWG, YYQ, YYD, TJPOWOOI.
OCKT, QGZNUNUWORYURR, HSGH, LBZ.
OCKT, QGZNUNUWORYURR, HSGH, LBZ.
GMKRGVVVNIDPXOUR, MQTFWSDMAV, SLOJCRM, PZ, LLRYMOYEGCR.
GMKRGVVVNIDPXOUR, MQTFWSDMAV, SLOJCRM, PZ, LLRYMOYEGCR.
VZ, ESKPVKQJCDJE, XPK, HVQXNZNJDHBAZUJH.
VZ, ESKPVKQJCDJE, XPK, HVQXNZNJDHBAZUJH.
RDNJUKG, PADQTCO (RQKLFJGOCZUFU), NPQWPWRMWTPM, LBEBDADGORNEKQI, GFFMRCA.
RDNJUKG, PADQTCO , NPQWPWRMWTPM, LBEBDADGORNEKQI, GFFMRCA.
AWKJFVARJBOQPZY, FGEMPRZBEVCUIZ, WMWDIMPSCOI, AGUAOE, MGCF.
AWKJFVARJBOQPZY, FGEMPRZBEVCUIZ, WMWDIMPSCOI, AGUAOE, MGCF.
HDBFZPXLR, OHOYEB (OIMASRKJEI, KTUXPYFVENCSIB, AXVCPFQM), SPDTSFMRS.
HDBFZPXLR, OHOYEB , SPDTSFMRS.
HJZBKLM, KEKQPPZWWIFDXA, IPUQUUDK (PVKUE, PEF).
HJZBKLM, KEKQPPZWWIFDXA, IPUQUUDK .
ENFAJYDVFSGOTEYX, ZYOCZ, VZKDFC, HOZMAAUUAJAU.
ENFAJYDVFSGOTEYX, ZYOCZ, VZKDFC, HOZMAAUUAJAU.
IOLF, RGJZQ, WEUYQNGVI, ULEJVMDQSUWORXUN.
IOLF, RGJZQ, WEUYQNGVI, ULEJVMDQSUWORXUN.
ZQ, MLARI, OMDGYFOKPWTGCE, DEMNCVEXTFSERTL, EJPJCKK (EUAJIJKFKHXKOR).
ZQ, MLARI, OMDGYFOKPWTGCE, DEMNCVEXTFSERTL, EJPJCKK .
EBMQMQNKNAOEIOD (WJGMNJQAI, UROUUPGRBFM), BHEFMODATXAOD, ACYJPKGKLMHOJML, UDNZID.
EBMQMQNKNAOEIOD , BHEFMODATXAOD, ACYJPKGKLMHOJML, UDNZID.
W, KSZ, CHGKU, UDNZALEPUGDWSJ, UGBAIOUWSCJTL.
W, KSZ, CHGKU, UDNZALEPUGDWSJ, UGBAIOUWSCJTL.
YBWREDG, Q, UMDR.
YBWREDG, Q, UMDR.
ADUZFAB, BEDUGBKXDHPEV, EJLZFW, I, UQKVOLOADDQIY.
ADUZFAB, BEDUGBKXDHPEV, EJLZFW, I, UQKVOLOADDQIY.
MEVT (PYHPJTHF), OIMQDTGVFZALKEK, FLYGSRLHFYXXLQ.
MEVT , OIMQDTGVFZALKEK, FLYGSRLHFYXXLQ.
SQYJY, SLLEBYLQSSNWXRJ, JBXKWPIHML, Q.
SQYJY, SLLEBYLQSSNWXRJ, JBXKWPIHML, Q.
V, FUBYGIVIX (ORQRW, MNGEEVKZRXD), ZFEZHWIIXBCDYQH.
V, FUBYGIVIX , ZFEZHWIIXBCDYQH.
FITPIWJETWJMCDWI, MO, DPNYQFLVJCFUODQ, SPWGRNMEMPWSUEO (FWCEZIUZPYU, SYIGDEQMXW).
FITPIWJETWJMCDWI, MO, DPNYQFLVJCFUODQ, SPWGRNMEMPWSUEO .
KA, MIF, LBPVGHKMMQNRDSTL, JFQDSCE, SELLNVU.
KA, MIF, LBPVGHKMMQNRDSTL, JFQDSCE, SELLNVU.
GSU (SLXNEIZ), PMC, FPZJCGLA, IWLPNJTKRQKBB.
GSU , PMC, FPZJCGLA, IWLPNJTKRQKBB.
NSTQMASBQSMJ, UQAKGFXJGS, BHUVVD, WKFTOK.
NSTQMASBQSMJ, UQAKGFXJGS, BHUVVD, WKFTOK.
TKUONDFMRIORJPOU, MVRVQJFPPWDOP, BXKYDL (MSWDAGNTMIEVDHPV, DCM), CMNF.
TKUONDFMRIORJPOU, MVRVQJFPPWDOP, BXKYDL , CMNF.
SFYKEOHGVQSHD, VLLBXFDVV, VRPYUQO.
SFYKEOHGVQSHD, VLLBXFDVV, VRPYUQO.

IJMLDAQRFJHFX, WYTVBAUMYVVMX, A (VZZUECSHNHE).
IJMLDAQRFJHFX, WYTVBAUMYVVMX, A .
P, WQUIFN, ADJYRBKDWVOYSFV.
P, WQUIFN, ADJYRBKDWVOYSFV.
GOMYVJVRHF, JCWTASQJBVR, ANYVXOMKVXZHAIPR.
GOMYVJVRHF, JCWTASQJBVR, ANYVXOMKVXZHAIPR.
YTLYX, PDMJKMNKHNDXCLGZ, QW, IMNZFGXGAWDSBJG (CGTQIFJDAMDBZK).
YTLYX, PDMJKMNKHNDXCLGZ, QW, IMNZFGXGAWDSBJG .
XEBV, WHONOTLWNAAV (LUTYYGICCTWIAX), QZXYAMPEJVVLX.
XEBV, WHONOTLWNAAV , QZXYAMPEJVVLX.
GASKEJEXTSL, JWHGPXGVMGRTC, YRXBPZMRRX, HWNRC.
GASKEJEXTSL, JWHGPXGVMGRTC, YRXBPZMRRX, HWNRC.
IFPAFHUSUD, QTG (VOMBE, XP), LY, NQXSJSAGFSDUG.
IFPAFHUSUD, QTG , LY, NQXSJSAGFSDUG.
IFYJWXYBHQHXIRR, TMZSVWBBTCZF, VAGOLCIQMZBDSE, PMKXXSCYKW, QGPGHOYIRF.
IFYJWXYBHQHXIRR, TMZSVWBBTCZF, VAGOLCIQMZBDSE, PMKXXSCYKW, QGPGHOYIRF.
G, DBMWVC, UYTCWOFHAIYKZ (VRYSGGJFCL, FKPGULEBMXDPPBRA).
G, DBMWVC, UYTCWOFHAIYKZ .
EUXOKOSUUWTVXSOY (BXFBXXE, NZ, OMGAT), VIJHLS, SJSGDAVKFOK.
EUXOKOSUUWTVXSOY , VIJHLS, SJSGDAVKFOK.
MYYXMLCXNCUN, TLPUDVCLTYGYKP, OGNPKNTXREINCW, NYZLCXTGTBISVLF (NQMEXYNPARNHNN, OQXBDW, AHJWAKKW
MYYXMLCXNCUN, TLPUDVCLTYGYKP, OGNPKNTXREINCW, NYZLCXTGTBISVLF .
SBKZ, UKUVPSOWQOAZRWNU, EMQOTWERAVPGHLFP, CZO (AYXOYPUPSXI), SKYFIAVK.
SBKZ, UKUVPSOWQOAZRWNU, EMQOTWERAVPGHLFP, CZO , SKYFIAVK.
BXMUSYALHM (PW, A, XDSFO), KHRHCSRSNRS, CPAKJORPEHUOP (V), QK.
BXMUSYALHM , KHRHCSRSNRS, CPAKJORPEHUOP , QK.
EKOAYNW, WQJJPWNSUBGHJEVZ, SSZNHJQG, S.
EKOAYNW, WQJJPWNSUBGHJEVZ, SSZNHJQG, S.
EXPNVLVJZA (DVWIUBGHGVKW, ZYJR), ALWHHUIMZVUR, IPSA.
EXPNVLVJZA , ALWHHUIMZVUR, IPSA.
NIYH, I, JYDVCBO, HKIOYGZIBEEHSBQA, K.
NIYH, I, JYDVCBO, HKIOYGZIBEEHSBQA, K.
UWL, BIJ, HNQAJPPQERVN.
UWL, BIJ, HNQAJPPQERVN.
OSUGKSJQ (LZMML), DWM, TMDRFWXLCEREDWXR.
OSUGKSJQ , DWM, TMDRFWXLCEREDWXR.
PAZPL, UTZTCDXMDODA, DZCGCISCUAIM, YTC, ITSACOAKH.
PAZPL, UTZTCDXMDODA, DZCGCISCUAIM, YTC, ITSACOAKH.
FENLQW, MYZISQMGMZRGGLRD, EHHXDR.
FENLQW, MYZISQMGMZRGGLRD, EHHXDR.
GYAHBUNLEMQRE (VLUHGCNHS, ILKGUOLNMTGUDIO), NPN, APIUR.
GYAHBUNLEMQRE , NPN, APIUR.
TFYRAJSUQROGM, FR, CSGXBOFDYUACB, HTTPOCPDULKQQARI, QBJACCRIQFTDTUJ.
TFYRAJSUQROGM, FR, CSGXBOFDYUACB, HTTPOCPDULKQQARI, QBJACCRIQFTDTUJ.
M (URLKMWQYDSDEDX, YZGCCY), QCWHUAYTMVBNYL (BDBDLLPPXUCPF), SFEPXYRIJJYXV, CSUXZYCMKZKZH.
M , QCWHUAYTMVBNYL , SFEPXYRIJJYXV, CSUXZYCMKZKZH.
MBENHVRYHUH, HAPNI, GHPGOVI.
MBENHVRYHUH, HAPNI, GHPGOVI.

```
WLRNEMUTRTVHJXLT, BZUTADFVRVAW, PGRFAHRAKJTARS, RSYVTATD, ERJX.
WLRNEMUTRTVHJXLT, BZUTADFVRVAW, PGRFAHRAKJTARS, RSYVTATD, ERJX.
Passed! Please submit.
```

Even if your solution was incorrect or you skipped this exercise, run this cell to see the expected output of a call to `extract_basic_data(food_lod)`.

```
In [14]: ### Loading results
         import pickle
         import os
         path = './resource/asnlib/publicdata/ex2.pkl'
         if not os.path.exists(path):
             with open(path, 'wb') as file:
                 pickle.dump(extract_basic_data(food_lod), file)
         with open(path, 'rb') as file:
             basic_data = pickle.load(file)
```

1.5 Exercise 3 (1 Points):

Our analysis requires that the foods have at least 3 listed ingredients and have amounts for nutrients: 'fat', 'protein', 'sodium', and 'carbohydrates'.

Given `data`, a `list` of `dict`s, define `filter_basic_data(data)` to filter out unwanted records. - This function should return a **new** list containing the same `dicts` as `data` with the following exceptions. - You can assume that each `dict` in `data` will have `'list_of_ingredients` and `'raw_nutrients'` as keys and that the respective values for those keys are of type `list` and `dict`. - Any `dict` with fewer than 3 items in it's `'list_of_ingredients'` should not be included. - Any `dict`, `d`, where `d['raw_nutrients']` does not have **all** of `{'fat', 'protein', 'sodium', 'carbohydrates'}` as keys should not be included.

```
In [15]: ### Define filter_basic_data
         def filter_basic_data(data):
             ###
             new_data = []
             for recipe in data:
                 to_include = {'ingredients':0,
                              'fat': False,
                               'protein': False,
                               'sodium': False,
                              'carbohydrates': False}
                 for nutrient in recipe['raw_nutrients']:
                     to_include[nutrient] = True
                 for ingredient in recipe['list_of_ingredients']:
                     to_include["ingredients"] += 1
                     if to_include["fat"] and to_include["protein"] and to_include["sodium"] a
                         new_data.append(recipe)
                         break
             return new_data
             ###
```

The demo cell below should display the following output:

```
[{'list_of_ingredients': [1, 2, 'this ok', 'milk'],
  'raw_nutrients': {'fat': 22,
   'protein': 'caterpillar',
   'carbohydrates': 5,
   'sodium': 100,
   'awesome sauce': 'this one should be kept'}}]
```

```
In [16]: ### define demo inputs
         demo_data_ex3 = [
             {
                 'list_of_ingredients': [1, 2, 'this ok', 'milk'],
                 'raw_nutrients': {
                     'fat':22,
                     'protein': 'caterpillar',
                     'carbohydrates': 5,
                     'sodium': 100,
                     'awesome sauce': 'this one should be kept'}
             },
             {
                 'list_of_ingredients': ['catfish', 2, 'cse6040', 'bicycle'],
                 'raw_nutrients': {
                     'fat':12,
                     'carbohydrates': 35,
                     'sodium': 70,
                     'awesome sauce': 'this one should be rejected - no protein'
                 }
             },
             {
                 'list_of_ingredients': ['marble', 2.5],
                 'raw_nutrients': {
                     'fat':12,
                     'carbohydrates': 35,
                     'protein': 7,
                     'sodium': 70,
                     'awesome sauce': 'this one should be rejected too - not enough ingredient
                 }
             }
         ]
```

```
In [17]: ### call demo funtion
         filter_basic_data(demo_data_ex3)
```

```
Out[17]: [{'list_of_ingredients': [1, 2, 'this ok', 'milk'],
           'raw_nutrients': {'fat': 22,
            'protein': 'caterpillar',
            'carbohydrates': 5,
            'sodium': 100,
            'awesome sauce': 'this one should be kept'}}]
```

The cell below will test your solution for Exercise 3. The testing variables will be available for debugging under the following names in a dictionary format. - `input_vars` - Input variables for your solution. - `original_input_vars` - Copy of input variables from prior to running your solution. These *should* be the same as `input_vars` - otherwise the inputs were modified by your solution. - `returned_output_vars` - Outputs returned by your solution. - `true_output_vars` - The expected output. This *should* "match" `returned_output_vars` based on the question requirements - otherwise, your solution is not returning the correct output.

```
In [18]: ### test_cell_ex3

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###

         from tester_fw.testers import Tester_ex3
         tester = Tester_ex3()
         for _ in range(20):
             try:
                 tester.run_test(filter_basic_data)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                 raise

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
         print('Passed! Please submit.')
```

```
initializing tester_fw.tester_6040
Passed! Please submit.
```

Even if your solution was incorrect or you skipped this exercise, run this cell to see the expected output of a call to `filter_basic_data(basic_data)`.

```
In [19]: ### Loading results
         import pickle
         import os
         path = './resource/asnlib/publicdata/ex3.pkl'
         if not os.path.exists(path):
             with open(path, 'wb') as file:
                 pickle.dump(filter_basic_data(basic_data), file)
         with open(path, 'rb') as file:
             filtered_data = pickle.load(file)
```

1.6 Exercise 4 (2 Points):

We want to compute summary statistics on the nutrients present in each food. While you might be able to find formulas for these statistics and implement them yourselves - there is no need to reinvent the wheel. Feel free to use the `statistics` module, but you will have to `import` it yourself.

Given a `list` of `dicts`, `data` structured as `basic_data`, define `make_summary(data, key)` to generate a dictionary of summary statistics. - We will assume that each `dict` in `data` has a `'raw_nutrients'` key mapped to a dictionary which maps nutrients to amounts. - Extract the amount of the nutrient given by `key` for each `dict`, `d` in `data` - we will call these the observations. I.e. `data[0]['raw_nutrients'][key]` is one observation. - Note: each entry in `data` counts as an observation and for all `dicts` `d` in `data`, `d['raw_nutrients'][key]` is **not** guaranteed to exist. In such cases, we will interpret the observation as a 0. - Compute statistics on the observations. Store the results in a dictionary with the following mapping: - `'mean'` - (`float`) mean of all observations - `'median'` - (`float`) median of all observations - `'stdev'` - (`float`) **population** standard deviation of all observations - check your stats notes and documentation to make sure you're computing this correctly - `'min'` - (`float`) minimum - `'max'` - (`float`) maximum

```
In [20]: ### Define make_summary
         def make_summary(data, key):
             import statistics
             new_list = []
             for d in data:
                 if key in d['raw_nutrients'].keys():
                     temp = d['raw_nutrients'][key]
                     new_list.append(temp)
                 else:
                     new_list.append(0)
             xmean = statistics.mean(new_list)
             xmedian = statistics.median(new_list)
             xstdv = statistics.pstdev(new_list)
             xmin = min(new_list)
             xmax = max(new_list)

             new_dict = {}
             new_dict['mean'] = float(xmean)
             new_dict['median'] = float(xmedian)
             new_dict['stdev'] = float(xstdv)
             new_dict['min'] = float(xmin)
             new_dict['max'] = float(xmax)


             return new_dict
```

```
###
#d = data
#print(d)
#new_list =
#nutrient_dict = d[0]['raw_nutrients'][key] #, d[1]['raw_nutrients'][key]
#print(nutrient_dict)

###
```

The demo cell below should display the following output:

```
key: foo
{'mean': 18.714285714285715, 'median': 17.0, 'stdev': 12.75835060672349, 'min': 5.0, 'max': 48

key: bar
{'mean': 23.571428571428573, 'median': 33.0, 'stdev': 14.907880397936646, 'min': 0.0, 'max': 33

key: baz
{'mean': 100.0, 'median': 100.0, 'stdev': 0.0, 'min': 100.0, 'max': 100.0}
```

```
In [21]: ### define demo inputs
         demo_data_ex4 = [
             {'raw_nutrients': {'foo': 12,   'bar': 33,  'baz': 100}},
             {'raw_nutrients': {'foo': 48,   'bar': 33,  'baz': 100}},
             {'raw_nutrients': {'foo': 17,   'bar': 33,  'baz': 100}},
             {'raw_nutrients': {'foo': 5,                'baz': 100}},
             {'raw_nutrients': {'foo': 18,   'bar': 33,  'baz': 100}},
             {'raw_nutrients': {'foo': 12,   'bar': 33,  'baz': 100}},
             {'raw_nutrients': {'foo': 19,               'baz': 100}},
         ]
         demo_keys_ex4 = ['foo', 'bar', 'baz']
```

```
In [22]: ### call demo funtion
         for k in demo_keys_ex4:
             print(f'key: {k}')
             print(make_summary(demo_data_ex4, k))
             print()
```

```
key: foo
{'mean': 18.714285714285715, 'median': 17.0, 'stdev': 12.75835060672349, 'min': 5.0, 'max': 48

key: bar
{'mean': 23.571428571428573, 'median': 33.0, 'stdev': 14.907880397936646, 'min': 0.0, 'max': 33

key: baz
{'mean': 100.0, 'median': 100.0, 'stdev': 0.0, 'min': 100.0, 'max': 100.0}
```

The cell below will test your solution for Exercise 4. The testing variables will be available for debugging under the following names in a dictionary format. - `input_vars` - Input variables for your solution. - `original_input_vars` - Copy of input variables from prior to running your solution. These *should* be the same as `input_vars` - otherwise the inputs were modified by your solution. - `returned_output_vars` - Outputs returned by your solution. - `true_output_vars` - The expected output. This *should* "match" `returned_output_vars` based on the question requirements - otherwise, your solution is not returning the correct output.

```
In [23]: ### test_cell_ex4

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###

         from tester_fw.testers import Tester_ex4
         tester = Tester_ex4()
         for _ in range(20):
             try:
                 tester.run_test(make_summary)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                 raise

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
         print('Passed! Please submit.')
```

```
initializing tester_fw.tester_6040
Passed! Please submit.
```

```
In [24]: input_vars
```

```
Out[24]: {'data': [{'description': 'clrlenpl',
            'raw_nutrients': {'ubb': 17, 'w': 47, 'oooeulpilsgbur': 17}},
           {'description': 'mmefpcucwgulyxqd',
            'raw_nutrients': {'ubb': 47, 'w': 16, 'oooeulpilsgbur': 32}},
           {'description': 'kpvxexqyuyxol',
            'raw_nutrients': {'ubb': 49, 'w': 35, 'oooeulpilsgbur': 4}},
           {'description': 'wmretynrbnfi', 'raw_nutrients': {'ubb': 6, 'w': 27}},
           {'description': 'kswfxzhkg',
            'raw_nutrients': {'ubb': 6, 'w': 5, 'oooeulpilsgbur': 41}},
           {'description': 'bsoslhxgfdpwgtmb', 'raw_nutrients': {'ubb': 31, 'w': 18}},
           {'description': 'e', 'raw_nutrients': {'w': 27, 'oooeulpilsgbur': 9}},
           {'description': 'hgpevnrn',
            'raw_nutrients': {'ubb': 12, 'w': 26, 'oooeulpilsgbur': 33}},
           {'description': 'ynmkgk',
            'raw_nutrients': {'ubb': 48, 'oooeulpilsgbur': 19}},
           {'description': 'dv',
            'raw_nutrients': {'ubb': 38, 'w': 35, 'oooeulpilsgbur': 33}}],
          'key': 'ubb'}
```

```
In [25]: returned_output_vars

Out[25]: {'summary': {'mean': 25.4,
           'median': 24.0,
           'stdev': 18.364095403803585,
           'min': 0.0,
           'max': 49.0}}
```

```
In [26]: true_output_vars

Out[26]: {'summary': {'mean': 25.4,
           'median': 24.0,
           'stdev': 18.364095403803585,
           'min': 0.0,
           'max': 49.0}}
```

Even if your solution was incorrect or you skipped this exercise, run this cell. You would get the same result as `summary_dict` if you were to run the code below with a correct implementation of `make_summary`.

```
keys = ('fat', 'protein', 'carbohydrates', 'sodium')
{key:make_summary(filtered_data, key) for key in keys}
```

```
In [27]: ### Loading results
         import pickle
         import os
         path = './resource/asnlib/publicdata/ex4.pkl'
         if not os.path.exists(path):
             with open(path, 'wb') as file:
                 pickle.dump({key:make_summary(filtered_data, key) for key in ('fat', 'protein
         with open(path, 'rb') as file:
             summary_dict = pickle.load(file)
```

1.7 Exercise 5 (3 Points):

We are interested in whether the amount of one particular nutrient in a food has any relationship with the amounts of other nutrients in the food. For this, we will compare the observations of multiple nutrients and compute the correlation between them.

Given `data`, a `list` of `dicts`, and `keys`, a `list` of `strings`, complete the function `create_cor_dict(data, key)` to find the correlation between each nutrient listed and all of the other nutrients listed. Return the result as a `dict` which maps each key to a dictionary mapping the other keys to the correlation between the parent key and the child key. For example, if `keys=['fat', 'protein', 'carbohydrates']` then the result would look something like this:

```
{'fat': {                                        # parent key is 'fat'
    'protein': 0.1854653535334078,              # parent key is 'fat' --> correlation between
    'carbohydrates': -0.6720362432582452        # correlation between 'fat' and 'carbohydrates
    },
 'protein': {                                    # 'protein'
     'fat': 0.1854653535334078,                  # 'protein' correlation w/ 'fat'
     'carbohydrates': -0.3814834566078096        # 'protein' correlation w/ 'carbohydrates'
     },
 'carbohydrates': {                              # 'carbohydrates'
     'fat': -0.6720362432582452,                 # 'carbohydrates' correlation w/ 'fat'
     'protein': -0.3814834566078096}             # 'carbohydrates' correlation w/ 'protein'
     }
```

You can assume that if `d` is a `dict` in `data`, then `d` will have `'raw_nutrients'` as a key which is mapped to a `dict` which itself maps strings to integers. For example:

```
[
    {'raw_nutrients': {'foo': 17,   'bar': 33,  'baz': 150}},
    {'raw_nutrients': {'foo': 5,                'baz': 35}},
    {'raw_nutrients': {'foo': 18,   'bar': 33,  'baz': 200}}
]
```

Each dictionary in `data` should be treated as a single observation. You can compute the correlation with the following formulas. - n is the number of observations. - $\bar{x}, \bar{y}$ - Means nutrient x, and nutrient y. - $\bar{xy} = \frac{1}{n}\sum_{i=0}^{n-1} x_i y_i$ - $\sigma_x =$ **population** standard deviaiton - check your stats notes and documentation to make sure that you are calculating this correctly - Correlation:

$$c = \frac{\bar{xy} - (\bar{x})(\bar{y})}{\sigma_x \sigma_y}$$

```
In [29]: ### Define make_correlations
         def make_correlations(data, keys):
             ###
             def correlation(x, y):
                 from statistics import mean, pstdev
                 xy = [x_*y_ for x_, y_ in zip(x, y)]
                 return (mean(xy) - mean(x)*mean(y)) / pstdev(x) / pstdev(y)

             def key2list(key):
                 return [record['raw_nutrients'].get(key, 0.0) for record in data]

             return {k1:{k2: correlation(key2list(k1), key2list(k2)) for k2 in keys if k2 != k
             ###
```

The demo cell below should display the following output:

```
{'foo': {'bar': 0.3328398218980465, 'baz': 0.983194888209125},
 'bar': {'foo': 0.33283982189804656, 'baz': 0.31688680340974},
 'baz': {'foo': 0.983194888209125, 'bar': 0.31688680340974007}}
```

```
In [30]: ### define demo inputs
         ### use naming convention demo_varname_ex_* to name demo variables
         demo_data_ex5 = [
             {'raw_nutrients': {'foo': 12,   'bar': 33,  'baz': 100}},
             {'raw_nutrients': {'foo': 48,   'bar': 33,  'baz': 400}},
             {'raw_nutrients': {'foo': 17,   'bar': 33,  'baz': 150}},
             {'raw_nutrients': {'foo': 5,                'baz': 35}},
             {'raw_nutrients': {'foo': 18,   'bar': 33,  'baz': 200}},
             {'raw_nutrients': {'foo': 12,   'bar': 33,  'baz': 105}},
             {'raw_nutrients': {'foo': 19,               'baz': 195}},
         ]
         demo_keys_ex5 = ['foo', 'bar', 'baz']
```

```
In [31]: ### call demo funtion
         make_correlations(demo_data_ex5, demo_keys_ex5)
```

```
Out[31]: {'foo': {'bar': 0.3328398218980465, 'baz': 0.983194888209125},
          'bar': {'foo': 0.33283982189804656, 'baz': 0.31688680340974},
          'baz': {'foo': 0.983194888209125, 'bar': 0.31688680340974007}}
```

The cell below will test your solution for Exercise 5. The testing variables will be available for debugging under the following names in a dictionary format. - `input_vars` - Input variables for your solution. - `original_input_vars` - Copy of input variables from prior to running your solution. These *should* be the same as `input_vars` - otherwise the inputs were modified by your solution. - `returned_output_vars` - Outputs returned by your solution. - `true_output_vars` - The expected output. This *should* "match" `returned_output_vars` based on the question requirements - otherwise, your solution is not returning the correct output.

```
In [32]: ### test_cell_ex5

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###

         from tester_fw.testers import Tester_ex5
         tester = Tester_ex5()
         for _ in range(20):
             try:
                 tester.run_test(make_correlations)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                 raise

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
         print('Passed! Please submit.')
```

```
initializing tester_fw.tester_6040
Passed! Please submit.
```

Even if your solution was incorrect or you skipped this exercise, run this cell. You would get the same result as `corr_dict` if you were to run the code below with a correct implementation of `make_correlations`.

```
make_correlations(filtered_data, ('fat', 'carbohydrates', 'protein'))
```

```
In [33]: ### Loading results
         import pickle
         import os
         path = './resource/asnlib/publicdata/ex5.pkl'
         if not os.path.exists(path):
             with open(path, 'wb') as file:
                 pickle.dump(make_correlations(filtered_data, ('fat', 'carbohydrates', 'protei
         with open(path, 'rb') as file:
             corr_dict = pickle.load(file)
         corr_dict
```

```
Out[33]: {'fat': {'carbohydrates': 0.42945936057100154, 'protein': 0.5300501948181573},
          'carbohydrates': {'fat': 0.42945936057100154, 'protein': 0.4185373785303934},
          'protein': {'fat': 0.5300501948181573, 'carbohydrates': 0.4185373785303934}}
```

1.8 Exercise 6 (2 Points):

We are interested in the most common ingredients listed for foods. Instead of gathering this information on the whole data set, we want it on a category level. A good strategy for drilling down could be useful for generating category level summaries and correlations as well. The function below will transform our `basic_data` structure (a list of dictionaries) into a dictionary mapping each category to a list of dictionaries which have that category. Each of these lists will have the same structure as `basic_data`. You may (or may not) find it useful in completing exercise 6.

```
In [34]: def group_by_category(data):
             from collections import defaultdict
             g = defaultdict(list)
             for d in data:
                 c = d['category']
                 g[c].append(d)
             return dict(g)
```

Complete the function `top_ingredients` to accomplish the following: - Parameters - `data` - `list of dicts`. You can assume that `d['list_of_ingredients']` is a `list` of strings, and `d['category']` is a string - for any `d` in `data`. Each of these `dicts` contains data on a single food. - `n` - `int` - number of ingredients to list - We will say that an ingredient's "strength" within a category is given by the following:

$$x_i = \text{number of times ingredient x has been listed in position i}$$

$$\text{Strength}_x = 3x_0 + 2x_1 + x_2$$

- For each unique category (value of `d['category']`) compute the strength of all ingredients present in that category. - Return a dictionary mapping each category to a list containing the top `n` ingredients in that category, ranked by strength in descending order. Only include ingredients which have strength greater than 0. - In the instance of ties (two ingredients having the same strength in a category), break the tie by ranking ingredients alphabetically. - If there are fewer than `n` ingredients - all of the ingredients should be included. There should always be `n` or fewer ingredients listed for each category.

```
In [35]: ### Define top_ingredients
         def top_ingredients(data, n=3):
             ###
             from collections import defaultdict
             strength = defaultdict(lambda: defaultdict(int))
             for item in data:
                 for rank, ingredient in enumerate(item['list_of_ingredients'][:min([3, len(ite
                     strength[item['category']][ingredient] += 3-rank
             return {k:sorted(v, key=lambda x: (-v.get(x), x))[:min([len(v), n])] for k, v in s
             ###
```

The demo cell below should display the following output:

```
{'cat0': ['bar', 'foo', 'baz', 'tux'],
 'cat1': ['bax', 'rak', 'foo'],
 'cat2': ['rah']}
```

```
In [36]: ### define demo inputs
         demo_data_ex6 = [
             {'category': 'cat0', 'list_of_ingredients':['foo', 'bar', 'baz', 'tux', 'rak']},
             {'category': 'cat0', 'list_of_ingredients':['bar', 'foo', 'baz', 'tux', 'baz']},
             {'category': 'cat0', 'list_of_ingredients':['bar', 'foo', 'tux']},
             {'category': 'cat0', 'list_of_ingredients':['bar', 'baz', 'tux',]},

             {'category': 'cat1', 'list_of_ingredients':['rak', 'foo', 'bax']},
             {'category': 'cat1', 'list_of_ingredients':['rak', 'bax']},
             {'category': 'cat1', 'list_of_ingredients':['bax', 'rak', 'foo']},
             {'category': 'cat1', 'list_of_ingredients':['bax', 'foo', 'rak']},

             {'category': 'cat2', 'list_of_ingredients':['rah']},
             {'category': 'cat2', 'list_of_ingredients':['rah']},
             {'category': 'cat2', 'list_of_ingredients':['rah']},
         ]
```

```
In [37]: ### call demo funtion
         top_ingredients(demo_data_ex6, n=5)
```

```
Out[37]: {'cat0': ['bar', 'foo', 'baz', 'tux'],
          'cat1': ['bax', 'rak', 'foo'],
          'cat2': ['rah']}
```

The cell below will test your solution for Exercise 6. The testing variables will be available for debugging under the following names in a dictionary format. - `input_vars` - Input variables for your solution. - `original_input_vars` - Copy of input variables from prior to running your solution. These *should* be the same as `input_vars` - otherwise the inputs were modified by your solution. - `returned_output_vars` - Outputs returned by your solution. - `true_output_vars` - The expected output. This *should* "match" `returned_output_vars` based on the question requirements - otherwise, your solution is not returning the correct output.

```
In [38]: ### test_cell_ex6

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###

         from tester_fw.testers import Tester_ex6
         tester = Tester_ex6()
         for _ in range(20):
             try:
                 tester.run_test(top_ingredients)
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
             except:
                 (input_vars, original_input_vars, returned_output_vars, true_output_vars) = te
                 raise

         ###
         ### AUTOGRADER TEST - DO NOT REMOVE
         ###
         print('Passed! Please submit.')
```

```
initializing tester_fw.tester_6040
Passed! Please submit.
```

Even if your solution was incorrect or you skipped this exercise, run this cell. You would get the same result as `leaders` if you were to run the code below with a correct implementation of `top_ingredients`.

```
top_ingredients(filtered_data, 3)
```

```
In [39]: ### Loading results
         import pickle
         import os
         path = './resource/asnlib/publicdata/ex6.pkl'
         if not os.path.exists(path):
             with open(path, 'wb') as file:
                 pickle.dump(top_ingredients(filtered_data, 3), file)
         with open(path, 'rb') as file:
             leaders = pickle.load(file)
```

Fin. This is the end of the exam. If you haven't already, submit your work.