```
%matplotlib inline
import matplotlib
import seaborn as sns
matplotlib.rcParams['savefig.dpi'] = 144
import pdb


from static_grader import grader
```

PW Miniproject

Introduction

The objective of this miniproject is to exercise your ability to use basic Python data structures, define functions, and control program flow. We will be using these concepts to perform some fundamental data wrangling tasks such as joining data sets together, splitting data into groups, and aggregating data into summary statistics. **Please do not use `pandas` or `numpy` to answer these questions.**

We will be working with medical data from the British NHS on prescription drugs. Since this is real data, it contains many ambiguities that we will need to confront in our analysis. This is commonplace in data science, and is one of the lessons you will learn in this miniproject.

Downloading the data

We first need to download the data we'll be using from Amazon S3:

```
%%bash
mkdir pw-data
wget http://dataincubator-
wqu.s3.amazonaws.com/pwdata/201701scripts_sample.json.gz -nc -P ./pw-
data
wget http://dataincubator-
wqu.s3.amazonaws.com/pwdata/practices.json.gz -nc -P ./pw-data
```

```
mkdir: cannot create directory ‘pw-data’: File exists
File ‘./pw-data/201701scripts_sample.json.gz’ already there; not
retrieving.

File ‘./pw-data/practices.json.gz’ already there; not retrieving.
```

Loading the data

The first step of the project is to read in the data. We will discuss reading and writing various kinds of files later in the course, but the code below should get you started.

```
import gzip
import simplejson as json

with gzip.open('./pw-data/201701scripts_sample.json.gz', 'rb') as f:
    scripts = json.load(f)

with gzip.open('./pw-data/practices.json.gz', 'rb') as f:
    practices = json.load(f)
```

This data set comes from Britain's National Health Service. The `scripts` variable is a list of prescriptions issued by NHS doctors. Each prescription is represented by a dictionary with various data fields: `'practice'`, `'bnf_code'`, `'bnf_name'`, `'quantity'`, `'items'`, `'nic'`, and `'act_cost'`.

```
scripts[:2]
```

```
[{'bnf_code': '0101010G0AAABAB',
  'items': 2,
  'practice': 'N81013',
  'bnf_name': 'Co-Magaldrox_Susp 195mg/220mg/5ml S/F',
  'nic': 5.98,
  'act_cost': 5.56,
  'quantity': 1000},
 {'bnf_code': '0101021B0AAAHAH',
  'items': 1,
  'practice': 'N81013',
  'bnf_name': 'Alginate_Raft-Forming Oral Susp S/F',
  'nic': 1.95,
  'act_cost': 1.82,
  'quantity': 500}]
```

A glossary of terms and FAQ is available from the NHS regarding the data. Below we supply a data dictionary briefly describing what these fields mean.

Data field	Description
'practice'	Code designating the medical practice issuing the prescription
'bnf_code'	British National Formulary drug code
'bnf_name'	British National Formulary drug name
'quantity'	Number of capsules/quantity of liquid/grams of powder prescribed
'items'	Number of refills (e.g. if 'quantity' is 30 capsules, 3 'items' means 3 bottles of 30 capsules)
'nic'	Net ingredient cost
'act_cost'	Total cost including containers, fees, and discounts

The `practices` variable is a list of member medical practices of the NHS. Each practice is represented by a dictionary containing identifying information for the medical practice. Most of the data fields are self-explanatory. Notice the values in the `'code'` field of `practices` match the values in the `'practice'` field of `scripts`.

```
practices[:2]
```

```
[{'code': 'A81001',
  'name': 'THE DENSHAM SURGERY',
  'addr_1': 'THE HEALTH CENTRE',
  'addr_2': 'LAWSON STREET',
  'borough': 'STOCKTON ON TEES',
  'village': 'CLEVELAND',
  'post_code': 'TS18 1HU'},
 {'code': 'A81002',
  'name': 'QUEENS PARK MEDICAL CENTRE',
  'addr_1': 'QUEENS PARK MEDICAL CTR',
  'addr_2': 'FARRER STREET',
  'borough': 'STOCKTON ON TEES',
  'village': 'CLEVELAND',
  'post_code': 'TS18 2AW'}]
```

In the following questions we will ask you to explore this data set. You may need to combine pieces of the data set together in order to answer some questions. Not every element of the data set will be used in answering the questions.

Question 1: summary_statistics

Our beneficiary data (`scripts`) contains quantitative data on the number of items dispensed (`'items'`), the total quantity of item dispensed (`'quantity'`), the net cost of the ingredients (`'nic'`), and the actual cost to the patient (`'act_cost'`). Whenever working with a new data set, it can be useful to calculate summary statistics to develop a feeling for the volume and character of the data. This makes it easier to spot trends and significant features during further stages of analysis.

Calculate the sum, mean, standard deviation, and quartile statistics for each of these quantities. Format your results for each quantity as a list: `[sum, mean, standard deviation, 1st quartile, median, 3rd quartile]`. We'll create a `tuple` with these lists for each quantity as a final result.

```
def describe(key):
    import statistics as st

    srtd_list  = sorted([scripts[i][key] for i in
range(len(scripts))])
    total = sum(scripts[i][key]for i in range(len(scripts)))
    avg = total / len(scripts)
    s = st.stdev(dict_item[key]for dict_item in scripts)
    mid = int(len(srtd_list)/2)
```

```
    def Median(sort_list):
        mid = int(len(srtd_list)/2)
        if len(sort_list)%2 !=0:
            median = sort_list[mid]
            return median
        else:
            median = ((sort_list[mid-1])+ (sort_list[mid]))/2
            return median

    med = Median(srtd_list)
    q25 = Median(srtd_list[:mid])

    if len(srtd_list) % 2 ==0:
        q75 = Median(srtd_list[mid:])
    else:
        q75 = Median(srtd_list[mid+1:])

    return  list([total, avg, s, q25, med, q75])

summary = [('items', describe('items')),
           ('quantity', describe('quantity')),
           ('nic', describe('nic')),
           ('act_cost', describe('act_cost'))]
#print(summary)
```

```
---------------------------------------------------------------------------
-----
IndexError                                Traceback (most recent call
last)
<ipython-input-8-6936078855ec> in <module>()
----> 1 summary = [('items', describe('items')),
      2            ('quantity', describe('quantity')),
      3            ('nic', describe('nic')),
      4            ('act_cost', describe('act_cost'))]
      5 #print(summary)

<ipython-input-7-ea812568cfa9> in describe(key)
     19
     20     med = Median(srtd_list)
---> 21     q25 = Median(srtd_list[:mid])
     22
     23     if len(srtd_list) % 2 ==0:

<ipython-input-7-ea812568cfa9> in Median(sort_list)
     12         mid = int(len(srtd_list)/2)
     13         if len(sort_list)%2 !=0:
---> 14             median = sort_list[mid]
     15             return median
     16         else:
```

```
IndexError: list index out of range
```

```
grader.score.pw__summary_statistics(summary)
```

```
==================
Your score:  0.6666666666666666
==================
```

Question 2: most_common_item

Often we are not interested only in how the data is distributed in our entire data set, but within particular groups -- for example, how many items of each drug (i.e. `'bnf_name'`) were prescribed? Calculate the total items prescribed for each `'bnf_name'`. What is the most commonly prescribed `'bnf_name'` in our data?

To calculate this, we first need to split our data set into groups corresponding with the different values of `'bnf_name'`. Then we can sum the number of items dispensed within in each group. Finally we can find the largest sum.

We'll use `'bnf_name'` to construct our groups. You should have *5619* unique values for `'bnf_name'`.

```
bnf_names = {k.get('bnf_name') for k in scripts}
assert(len(bnf_names) == 5619)
```

We want to construct "groups" identified by `'bnf_name'`, where each group is a collection of prescriptions (i.e. dictionaries from `scripts`). We'll construct a dictionary called `groups`, using `bnf_names` as the keys. We'll represent a group with a `list`, since we can easily append new members to the group. To split our `scripts` into groups by `'bnf_name'`, we should iterate over `scripts`, appending prescription dictionaries to each group as we encounter them.

```
groups = {name: [] for name in bnf_names}
for script in scripts:
    # INSERT ...
```

```
  File "<ipython-input-91-3076fbb1788d>", line 3
    # INSERT ...
                 ^
SyntaxError: unexpected EOF while parsing
```

```
groups = {name: [script for script in scripts if
script['bnf_name']==name] for name in bnf_names }

#print(groups)
```

```
from collections import Counter
```

```
tupl_key_items = Counter(dict([(groups[key][m]['bnf_name'],
groups[key][m]['items']) for key in groups.keys() for m in
range(len(groups[key]))]))

max_item = tupl_key_items.most_common(1)
print(max_item)
```

```
[('Influvac Sub-Unit_Vac 0.5ml Pfs', 587)]
```

Now that we've constructed our groups we should sum up `'items'` in each group and find the `'bnf_name'` with the largest sum. The result, `max_item`, should have the form `[(bnf_name, item total)]`, e.g. `[('Foobar', 2000)]`.

TIP: If you are getting an error from the grader below, please make sure your answer conforms to the correct format of `[(bnf_name, item total)]`.

```
grader.score.pw__most_common_item(max_item)
```

```
==================
Your score:  0.0
==================
```

Challenge: Write a function that constructs groups as we did above. The function should accept a list of dictionaries (e.g. `scripts` or `practices`) and a tuple of fields to `groupby` (e.g. `('bnf_name')` or `('bnf_name', 'post_code')`) and returns a dictionary of groups. The following questions will require you to aggregate data in groups, so this could be a useful function for the rest of the miniproject.

```
groups = group_by_field(scripts, ('bnf_name',))
test_max_item = ...

assert test_max_item == max_item
```

Question 3: postal_totals

Our data set is broken up among different files. This is typical for tabular data to reduce redundancy. Each table typically contains data about a particular type of event, processes, or physical object. Data on prescriptions and medical practices are in separate files in our case. If we want to find the total items prescribed in each postal code, we will have to *join* our prescription data (`scripts`) to our clinic data (`practices`).

Find the total items prescribed in each postal code, representing the results as a list of tuples `(post code, total items prescribed)`. Sort your results ascending alphabetically by post code and take only results from the first 100 post codes. Only include post codes if there is at least one prescription from a practice in that post code.

NOTE: Some practices have multiple postal codes associated with them. Use the alphabetically first postal code.

We can join `scripts` and `practices` based on the fact that `'practice'` in `scripts` matches `'code'` in `practices`. However, we must first deal with the repeated values of `'code'` in `practices`. We want the alphabetically first postal codes.

```
practice_postal = {}
for practice in practices:
    if practice['code'] in practice_postal:
        practice_postal[practice['code']] = ...
    else:
        practice_postal[practice['code']] = ...
```

Challenge: This is an aggregation of the practice data grouped by practice codes. Write an alternative implementation of the above cell using the `group_by_field` function you defined previously.

```
assert practice_postal['K82019'] == 'HP21 8TR'
```

Challenge: This is an aggregation of the practice data grouped by practice codes. Write an alternative implementation of the above cell using the `group_by_field` function you defined previously.

```
assert practice_postal['K82019'] == 'HP21 8TR'
```

Now we can join `practice_postal` to `scripts`.

```
joined = scripts[:]
for script in joined:
    script['post_code'] = ...
```

Finally we'll group the prescription dictionaries in `joined` by `'post_code'` and sum up the items prescribed in each group, as we did in the previous question.

```
items_by_post = ...
```

```
postal_totals = [('B11 4BW', 20673)] * 100

grader.score.pw__postal_totals(postal_totals)
```

Question 4: items_by_region

Now we'll combine the techniques we've developed to answer a more complex question. Find the most commonly dispensed item in each postal code, representing the results as a list of tuples (`post_code`, `bnf_name`, amount dispensed as proportion of total). Sort your results ascending alphabetically by post code and take only results from the first 100 post codes.

NOTE: We'll continue to use the `joined` variable we created before, where we've chosen the alphabetically first postal code for each practice. Additionally, some postal codes will have multiple `'bnf_name'` with the same number of items prescribed for the maximum. In this case, we'll take the alphabetically first `'bnf_name'`.

Now we need to calculate the total items of each `'bnf_name'` prescribed in each `'post_code'`. Use the techniques we developed in the previous questions to calculate these totals. You should have 141196 `('post_code', 'bnf_name')` groups.

```
total_items_by_bnf_post = ...
assert len(total_items_by_bnf_post) == 141196
```

Let's use `total_by_item_post` to find the maximum item total for each postal code. To do this, we will want to regroup `total_by_item_post` by `'post_code'` only, not by `('post_code', 'bnf_name')`. First let's turn `total_by_item_post` into a list of dictionaries (similar to `scripts` or `practices`) and then group it by `'post_code'`. You should have 118 groups in `total_by_item_post` after grouping it by `'post_code'`.

```
total_items = ...
assert len(total_items_by_post) == 118
```

Now we will aggregate the groups in `total_by_item_post` to create `max_item_by_post`. Some `'bnf_name'` have the same item total within a given postal code. Therefore, if more than one `'bnf_name'` has the maximum item total in a given postal code, we'll take the alphabetically first `'bnf_name'`. We can do this by sorting each group according to the item total and `'bnf_name'`.

```
max_item_by_post = ...
```

In order to express the item totals as a proportion of the total amount of items prescribed across all `'bnf_name'` in a postal code, we'll need to use the total items prescribed that previously calculated as `items_by_post`. Calculate the proportions for the most common `'bnf_names'` for each postal code. Format your answer as a list of tuples: `[(post_code, bnf_name, total)]`

```
items_by_region = [('B11 4BW', 'Salbutamol_Inha 100mcg (200 D) CFF',
0.0341508247)] * 100

grader.score.pw__items_by_region(items_by_region)
```