CS-432/532 Introduction to Web Science: Assignment #2

Due on Thursday, February 11, 2016

Dr. Michael L. Nelson

Plinio Vargas
pvargas@cs.odu.edu

Contents

List of Figures

List of Tables

Preface

Since the course material will become more complex as the semester progresses, and knowing that many solutions to new problems can be built on previous ones, I decided to create a library to access and re-utilize code.

Some of these functions can be really small (4-7 lines), however they solve a specific problem, so it is better to separate them from the rest of the code.

The location, interaction and functionality of this library, and the directory structure for this assignment are presented in more detail in the README file within the same folder hierarchy where this file was found.

Problem 1

Write a Python program that extracts 1000 unique links from Twitter. You might want to take a look at:

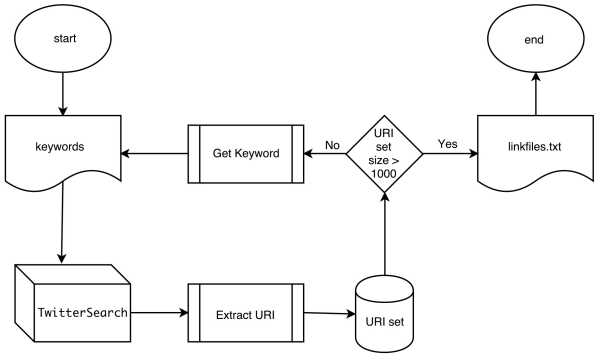
urlhttp://thomassileo.com/blog/2013/01/25/using-twitter-rest-api-v1-dot-1-with-python/

But there are many other similar resources available on the web. Note that only Twitter API 1.1 is currently available; version 1 code will no longer work.

Also note that you need to verify that the final target URI (i.e., the one that responds with a 200) is unique. You could have many different shortened URIs for www.cnn.com (t.co, bit.ly, goo.gl, etc.).

1.1 Approach

Figure 1: Flowchart diagram for *TwitterURI.py*



TwitterURI.py is the Python program solution for problem 1. **keywords** is a text file document which contains subject of interest, such as baseball, "Dominican, Republic", etc.; **TwitterSeach** is an API that provides a set of tweets from Twitter given a particular keyword; **Extract URI** filters any URI within the tweet; **URI set** is a unique collection of URIs; **Get Keyword** gets the next keyword from the text file; and finally ***linkfiles.txt*** is a collection of 1000 unique URIs.

Since we do not know the number of tweets returned from a particular keyword search, a file was created

Problem 1 continued on next page...

(*keywords.txt*) as input for ***TwitterURI.py*** (lines 70-75). Then, the API **TwitterSeach** in line (28-34) is used, which returns n number of tweets. From the tweet object we can extract the URI using regular expression (line 44):

```
url = re.findall(r'(https?://[^\s]+)', tweet['text'])
```

The extracted URI is validated and placed in a set: *all_links_set* lines (49-54):

```
# check if url has been already used
if uri not in root_url_array:
    root_url_array.append(uri)

    if validators.url(uri):
        AddURI(all_links_set, uri)
```

The last line of the portion code above uses a function *AddURI* imported from the self developed library *WEBlib.py* (a collection of functions that may have use for future applications). The function takes as a parameter a set object and a string (uri). *AddURI* takes care of all details regarding request response, connection timeouts, resource redirects, etc. ONLY if final URI provides a response 200 it gets added to the set.

Finally, if the set size is not greater than or not equal to 1000, we continue to get more keywords and repeat the process, otherwise the deliverable (1000 unique URIs) is written in *linkfiles.txt*.

1.2 Solution

See attached embedded file: ***linkfiles.txt***
Below is the entire code ***TwitterURI.py***, which is also attached to this document.

TwitterURI.py

```
1  import re
2  import sys
3  import os
4  import requests
5  import validators
6  import urllib3
7
8  PACKAGE_PARENT = '..'
9  SCRIPT_DIR = os.path.dirname(os.path.realpath(os.path.join(os.getcwd(),
       os.path.expanduser(__file__))))
10 sys.path.append(os.path.normpath(os.path.join(SCRIPT_DIR, PACKAGE_PARENT)))
11
12 from lib.WEBlib import AddURI
13 from TwitterSearch import *
14
15 __author__ = 'Plinio H. Vargas'
16 __date__ = 'Wed, Feb 03, 2016 at 22:27:02'
17 __email__ = 'pvargas@cs.odu.edu'
18
19 """
20 Given a set of key words this program makes use of TwitterSearch API to search for any tweets
```

```
related to the key words, and then extract the URI within in the tweet.
"""
root_url_array = []
all_links_set = set()
final_link = ''


def add_links(ts, key_words):
    global final_link
    try:
        tso = TwitterSearchOrder() # create a TwitterSearchOrder object
        tso.set_keywords(key_words) # let's define all words we would like to have a look for
        tso.set_language('en') # we want to see English tweets only
        tso.set_include_entities(False) # and don't give us all those entity information

         # get all tweet feeds with parameter above
        for tweet in ts.search_tweets_iterable(tso):
            if len(all_links_set) >= 1000:
                break

            tweet_text = tweet['text']

            # extract url from tweet
            url = re.findall(r'(https?://[^\s]+)', tweet['text'])

            # analyze url
            for uri in url:

                # check if url has been already used
                if uri not in root_url_array:
                    root_url_array.append(uri)

                    if validators.url(uri):
                        AddURI(all_links_set, uri)

    except TwitterSearchException as e: # take care of all those ugly errors if there are some
        print(e)

    return

# it's about time to create a TwitterSearch object with our secret tokens
ts = TwitterSearch(
    consumer_key='mysconsumerkey',
    consumer_secret='myconsumer_secret',
    access_token='myaccess_tocken',
    access_token_secret='myaccess_token_secret'
)

# find tweets for a particular key word
with open('keywords.txt', 'r') as file:
    for record in file:
        keys = record.strip().split(',')

```

```
        key_words = []
        for words in keys:
            key_words.append(words.strip())

        add_links(ts, key_words)
        print(len(all_links_set))

# save unique URIs to a file
with open('linkfiles.txt', 'w') as out:
    for link in all_links_set:
        out.write(link + '\n')
```

Problem 2

Download the TimeMaps for each of the target URIs. We'll use the ODU Memento Aggregator, so for example:

URI-R = http://www.cs.odu.edu/

URI-T = http://mementoproxy.cs.odu.edu/aggr/timemap/link/1/http://www.cs.odu.edu/

Create a histogram* of URIs vs. number of Mementos (as computed from the TimeMaps). For example, 100 URIs with 0 Mementos, 300 URIs with 1 Memento, 400 URIs with 2 Mementos, etc.

* = https://en.wikipedia.org/wiki/Histogram

2.1 Approach

A Python code < ***MementosURI.py*** > was developed to generate the histogram data, and the Histogram Graph (Figure 2) was generated using R-code < *histogram.r* >. Both files are attached to this document. Since we will be using the data collected later on in the course, a folder **a2/mementodata** was created to hold all TimeMaps for URIs included in < *linkfiles.txt* > (solution to problem 1).

URIs in < *linkfiles.txt* > are written sequentially and only separated by a new-line character ('\n'). In order to make a relation TimeMap-URI, the following file name format was established:

date_created.relative_position

For example if file *20160204.21* is located in folder **a2/mementodata**, then it references the TimeMap for URI located in position 21 out of 1000 in file < *linkfiles.txt* >. If *20160204.1* is not found in the directory, then there is no TimeMap for the first URI in < *linkfiles.txt* >.

Given the schema to solve Problem 2, below is the pseudo-code used for its implementation:

Problem 2 continued on next page...

Listing 1: MementosURI.py Pseudo Code

```
input: (I) file linkfiles.txt
output: (O) file histogramdata
Let A be an empty array where index i is number of mementos.
Let D be an stack containing all filenames in directory a2/mementodata
Let S be an integer containing number of URIs in I

# Initialize URIs with zero memento to number of URIs in I
A[0] ← S

begin
   while file in D
      A[0] ← A[0] - 1  # decrease number URIs with zero mementos
      i ← number of memento in file
      D.pop()
      if i ∈ A then
         A[i] ← A[i] + 1
      else
         A[i] ← 1

      for k := 1 → A.size
         write into O (k, A[k])
end
```

The python implementation found in < ***MementosURI.py*** > and pseudo-code above are very similar, and straightforward if we remove all exception handling required for the implementation. The input file I is used to get the number of URIs in our sample S (1000). O is the output file where histogram data will be saved. D contains all file names that have TimeMaps. Array A contains the data that will be written as a tuple in O. A starts as an empty array, so ONLY the indexes with values will be saved.

Different from the pseudo-code will be using instead of an array A a dictionary object [1] in the Python implementation.

Then, the only worthwhile explanation is how to find the value of index i or the number of mementos in any particular file. This is performed by the function *GetNumberMementos(filename)* located in our developed library *WEBlib.py*:

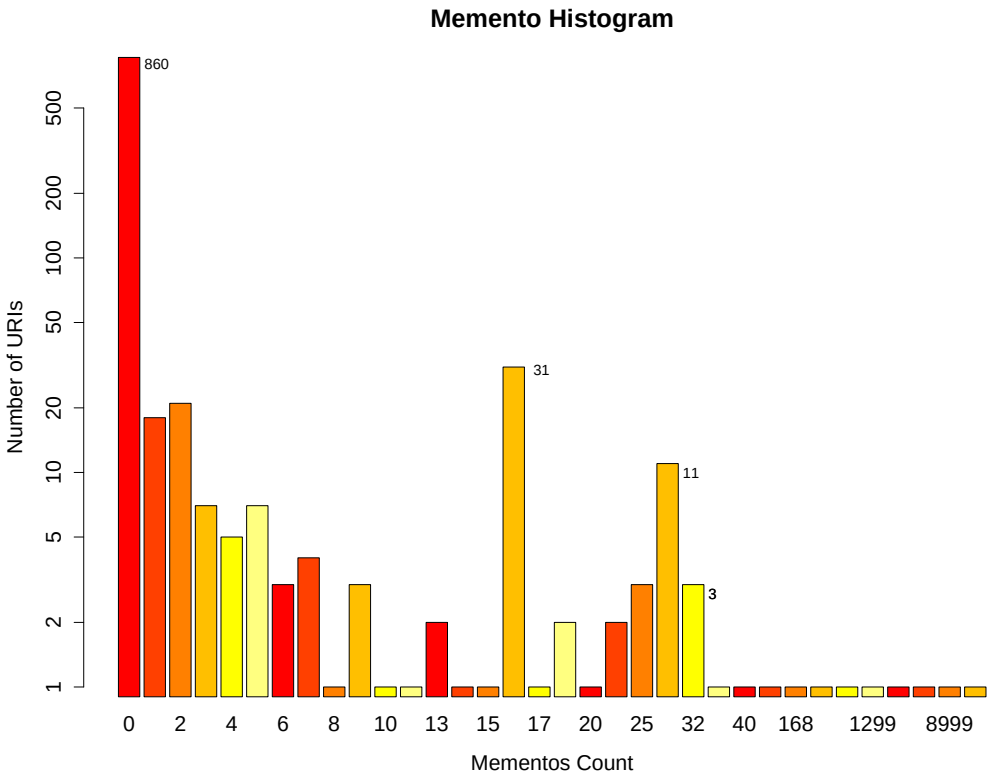
```
def GetNumberMementos(filename):
    """
    :param filename: Name of file to count number of mementos
    :return: number of mementos
    """
    with open(filename, 'r') as file:
        text = file.read()

    return len(re.findall(r'rel=\"memento\"', text))
```

As we may notice, function *GetNumberMementos()* simple takes as an argument a file name. It opens the file for reading and it uses regular expression to count and return the number of rel="memento" occurrences in the file.

Problem 2 continued on next page...

2.2 Solution

Figure 2: Memento Histogram for 1000 URIs Extracted from Twitter



The y-axis or Number of URIs is in a log scale. The data used to plot this histogram is contained in Table 1. The histogram resembles characteristics of a power-law distribution: (80/20) values. In our case 860 URIs out of a 1,000 had 0 mementos. At the tail end of the histogram we can notice that the memento counts gets higher as the number of URIs get smaller.

Problem 3

Estimate the age of each of the 1000 URIs using the "Carbon Date" tool:

http://ws-dl.blogspot.com/2014/11/2014-11-14-carbon-dating-web-version-20.html

Note: you'll should download the library and run it locally; don't try to use the web service.

For URIs that have > 0 Mementos and an estimated creation date, create a graph with age (in days) on one axis and number of mementos on the other.

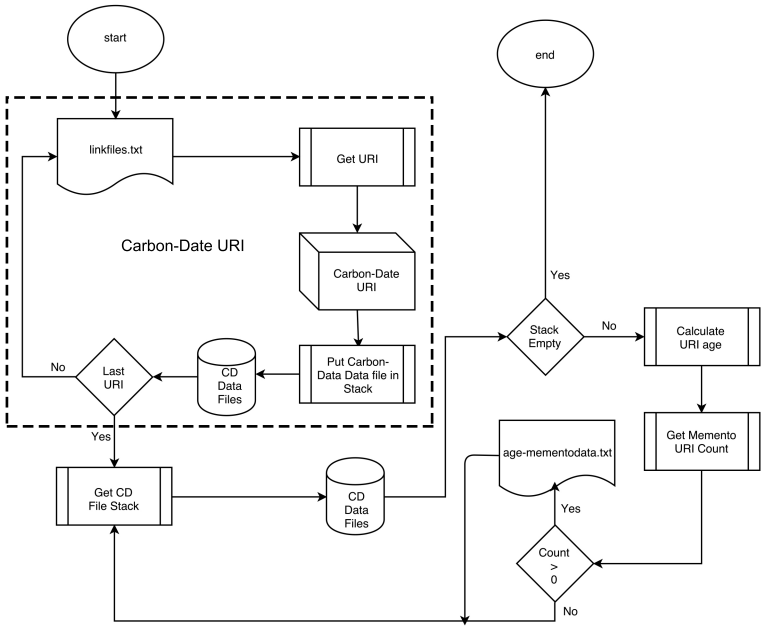
Not all URIs will have Mementos, and not all URIs will have an estimated creation date. State how many fall into either categories.

3.1 Approach

We will be using the same nomenclature file schema to associate the Carbon Date data with an URI's age. However, the Carbon Date files will be stored in a different directory: **a2/uriagedata**. Similar to problem 2 approach, python was used to generate the data, and R was utilized to create the graphs.

Shown below is a flowchart diagram for a python program used to generate the data:

Figure 3: Flowchart diagram for *AgeURI.py*

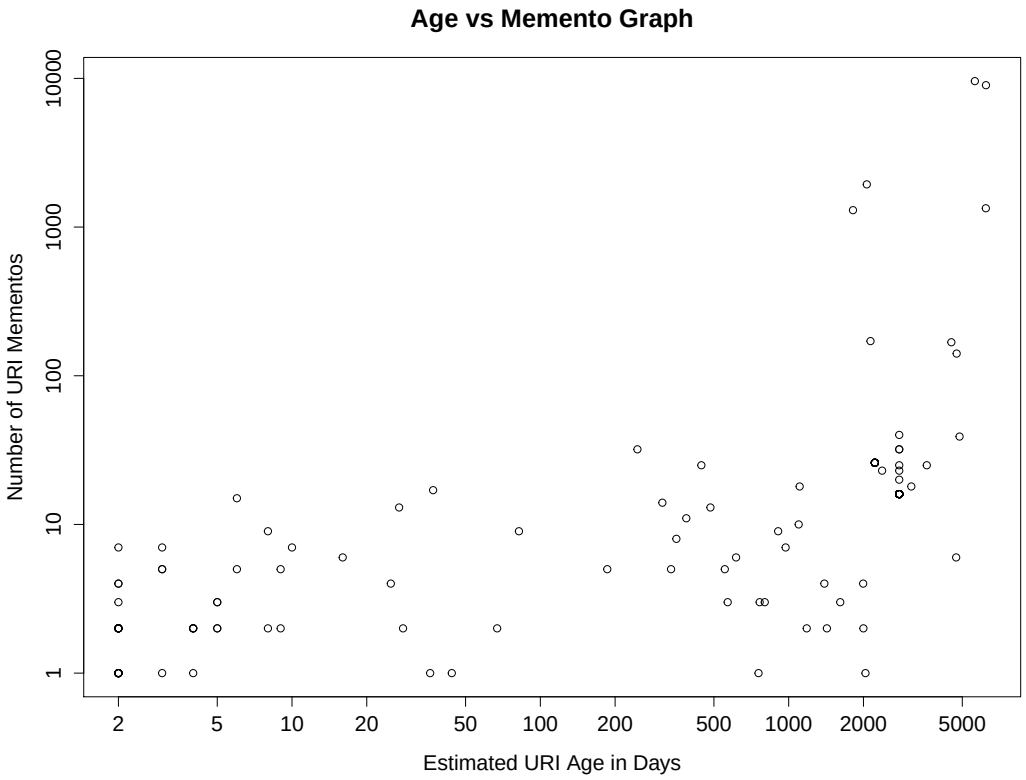


AgeURI.py is the Python implementation for problem 3. The section in the diagram labeled **Carbon-Date URI** takes all URIs generated to solve problem 1 and uses the **Carbon Date Tool** obtained from `http://ws-dl.blogspot.com/2014/11/2014-11-14-carbon-dating-web-version-20.html`. The HTTP request output is saved into **uriagedata** directory. This directory is labeled in the diagram **CD Data Files**. The remaining part of the flowchart explains that **CD Data Files** result gets merged with the result from data file solution from problem 2 (**Get Memento URI count** to produce our deliverable: **age-memento-data.txt** file.

Problem 3 continued on next page...

3.2 Solution Part-1

Figure 4: Age-Memento Graph

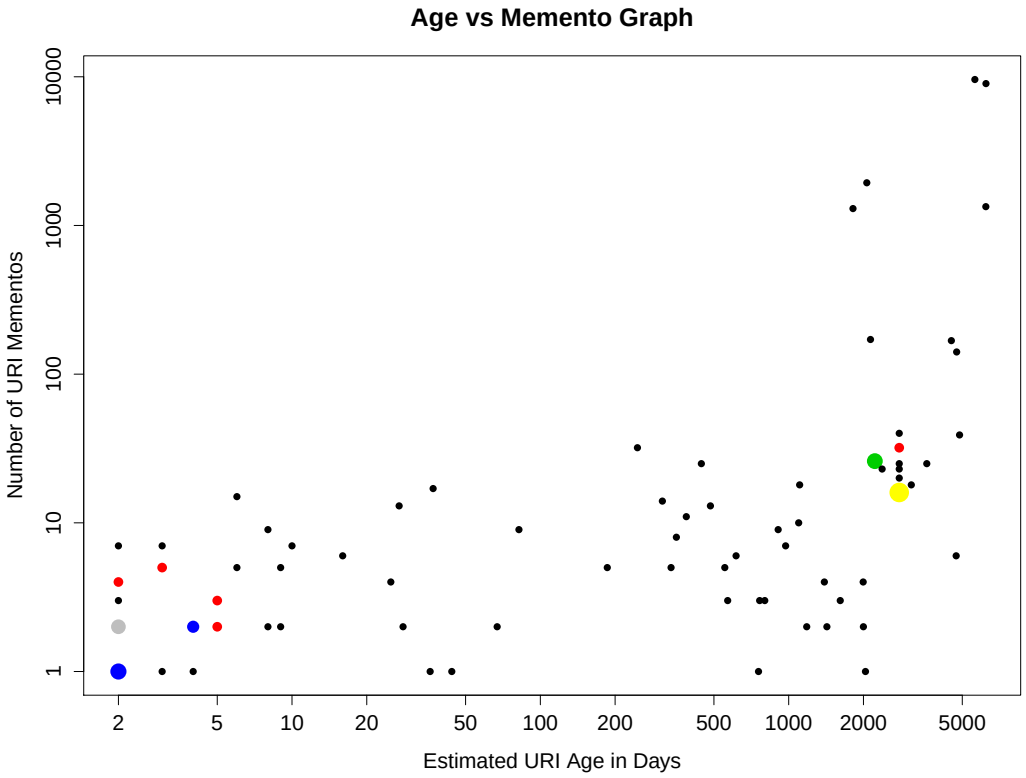


The x-axis and y-axis are in a log scale. The data used to plot this graph is contained in Table 2. Number of plotting points do not correspond to number of data set in Table 2 due to tuple value repetition with different URIs. It is difficult to appreciate repeating plots with darker plotting points on this graph. Then, it is a need to capture plot point repetition: Figure 5.

From the graph above we can notice a relationship between an URI's age and its Mementos count. In general, the younger the URI the less count of mementos it has, and vice versa. Figure 5 captures much better this inference, since the plot density is more representative of the collected data.

Problem 3 continued on next page...

Figure 5: Age-Memento Graph with Various Plot Size



The x-axis and y-axis are in a log scale. The data used to plot this graph is contained in Table 3. The plot point size corresponds to column labeled *Freq* from the same table utilizing the formula: $log\ Freq/2 + 1$. The larger the plot point size, the greater the number of occurrences for any particular data set.

The analysis of Figure 5 is similar to Figure 4. In general, the younger the URI the less count of mementos it has, and vice versa. However, the plotting is more representative of its data. For example, we can appreciate that a great deal of pages 2 days old have 1 memento. This information is not capture on Figure 4.

3.3 Solution Part-2

Not all URIs will have Mementos, and not all URIs will have an estimated creation date. ***AgeURI.py*** calculated this information:

Out of 1000 URIs 382 or 38.20% have no estimated creation date.
Out of 1000 URIs 860 or 86.00% have no mementos.

Additionally, the number of URIs with an estimated creation date and a memento is 140 or 14.00%.

Table 1: Histogram Memento Data for 1000 URIs Extracted from Twitter

No-URI	Mementos
860	0
18	1
21	2
7	3
5	4
7	5
3	6
4	7
1	8
3	9
1	10
1	11
2	13
1	14
1	15
31	16
1	17
2	18
1	20
2	23
3	25
11	26
3	32
1	39
1	40
1	141
1	168
1	171
1	240
1	1299
1	1338
1	1937
1	8999
1	9586

Histogram data was generated on 02/04/2016 utilizing timeMaps from `http://mementoproxy.cs.odu.edu/aggr/timemap/link/1/` concatenated to the URIs contained in the attached file *linkfiles.txt*. Any particular tuple (No-URI, Memento) represents number of URIs that have a specific number of mementos. For example the first tuple translate that from our 1,000 URIs sample 860 of them had 0 mementos.

Table 2: Age and Memento Data for 1000 URIs Extracted from Twitter

Age (days)	Memento Qty
2	2
2	4
2	1
2	2
2	2
2	2
2	3
2	2
2	1
2	1
2	1
2	2
2	2
2	1
2	1
2	1
2	1
2	1
2	1
2	2
2	1
2	4
2	7
2	1
3	5
3	1
3	5
3	7
4	1
4	2
4	2
4	2
4	2
5	2
5	3
5	2
5	3
6	15
6	5
8	9
8	2
9	5
9	2
10	7
16	6
25	4
27	13

Age (days)	Memento Qty
28	2
36	1
37	17
44	1
67	2
82	9
186	5
246	32
310	14
336	5
353	8
387	11
445	25
484	13
553	5
568	3
614	6
756	1
764	3
801	3
907	9
972	7
1097	10
1107	18
1182	2
1392	4
1425	2
1613	3
1815	1299
1995	4
1999	2
2038	1
2063	1937
2135	171
2223	26
2223	26
2223	26
2223	26
2223	26
2223	26
2223	26
2223	26
2223	26
2223	26
2223	26
2378	23
2787	16

Age (days)	Memento Qty
2787	16
2787	16
2787	16
2787	25
2787	16
2787	16
2787	32
2787	20
2787	16
2787	16
2787	16
2787	16
2787	16
2787	16
2787	16
2787	32
2787	16
2787	16
2787	16
2787	16
2787	16
2787	16
2787	16
2787	16
2787	16
2787	40
2787	16
2787	23
2787	16
2787	16
2787	16
2787	16
2787	16
2787	16
2787	16
3117	18
3598	25
4519	168
4725	6
4746	141
4876	39
5619	9586
6221	1338
6229	8999

The Age in days for all URIs were calculated on 02/05/2016 utilizing the Carbon-Dating tool found at `http://ws-dl.blogspot.com/2014/11/2014-11-14-carbon-dating-web-version-20.html`. The same methodology utilized on Table 2 to find mementos count was utilize here to find Mementos Qty.

Table 3: Data Grouped by Age and Memento from Table 2

Age (days)	Memento Qty	Freq
2	1	12
2	2	8
2	3	1
2	4	2
2	7	1
3	1	1
3	5	2
3	7	1
4	1	1
4	2	4
5	2	2
5	3	2
6	5	1
6	15	1
8	2	1
8	9	1
9	2	1
9	5	1
10	7	1
16	6	1
25	4	1
27	13	1
28	2	1
36	1	1
37	17	1
44	1	1
67	2	1
82	9	1
186	5	1
246	32	1
310	14	1
336	5	1
353	8	1
387	11	1
445	25	1
484	13	1
553	5	1

Age (days)	Memento Qty	Freq
568	3	1
614	6	1
756	1	1
764	3	1
801	3	1
907	9	1
972	7	1
1097	10	1
1107	18	1
1182	2	1
1392	4	1
1425	2	1
1613	3	1
1815	1299	1
1995	4	1
1999	2	1
2038	1	1
2063	1937	1
2135	171	1
2223	26	11
2378	23	1
2787	16	31
2787	20	1
2787	23	1
2787	25	1
2787	32	2
2787	40	1
3117	18	1
3598	25	1
4519	168	1
4725	6	1
4746	141	1
4876	39	1
5619	9586	1
6221	1338	1
6229	8999	1

Same data set that resulted in generating the data for Table 2 was utilized to generate values for this table. The grouping of (Age, Memento Qty) from Table 2 added a new column: Freq.

References

[1] Lutz, Mark (2013). List and Dictionaries. *Learning Python* (5th ed.). (pp. 262-263). Sebastopol, CA: O'Reilly Media.