CS1010S

Tutorial 8: Data Analysis

Nicholas Russell Saerang (russellsaerang@u.nus.edu)

Table of contents

1

Review

2

Tutorial 8

Dictionaries

Like a phone book, but faster

Dictionaries

Known as associative arrays or maps.

- Contains key-value pairs
- Orders of keys guaranteed (in python 3.7 onward)
- Very FAST lookup, add, remove

Very widely used in databases, caches, memoization, etc.

- Keep track of properties, e.g. the frequency of a particular score.

```
print(score in score_list)
print(score in score_dict)
```

Dictionaries

Known as associative arrays or maps.

- Contains key-value pairs
- Orders of keys guaranteed (in python 3.7 onward)
- Very FAST lookup, add, remove

Very widely used in databases, caches, memoization, etc.

- Keep track of properties, e.g. the frequency of a particular score.

```
print(score in score_list) # Slower
print(score in score_dict) # Faster
```

Dictionaries

Known as associative arrays or maps.

- Contains key-value pairs
- Orders of keys guaranteed (in python 3.7 onward)
- Very FAST lookup, add, remove

Very widely used in databases, caches, memoization, etc.

- Keep track of properties, e.g. the frequency of a particular score.

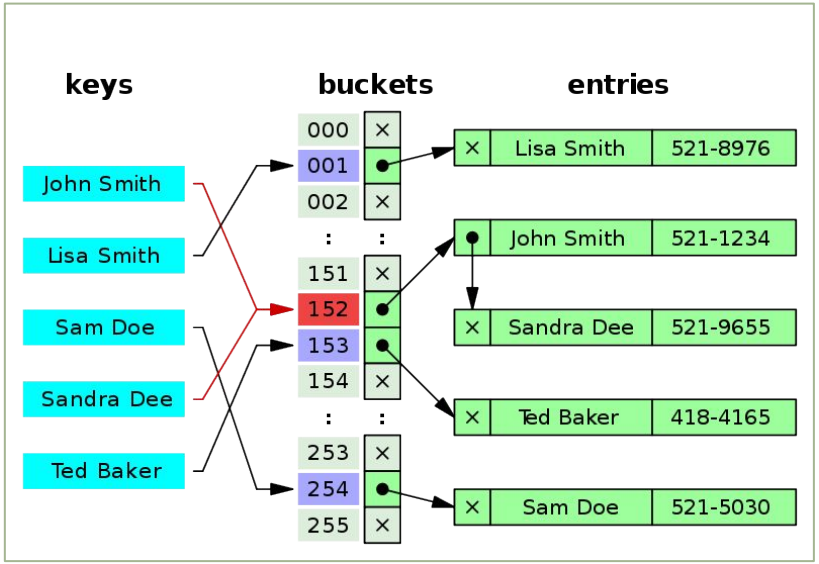
```
print(score in score_list) # Slower
print(score in score_dict) # Faster
```

Wonder why??
You will learn in CS2040, no longer in CS1010S syllabus

How do they work? (Optional)

Many different ways to implement:

- Separate chaining or open addressing
- Self-balancing binary trees



Optional reading:
https://en.wikipedia.org/wiki/Associative_array

Creating Dictionaries

```
# Initializing
d = {1:2, 3:4}
d = dict([(3.14, 1592),(1, 2)])

# Updating
d[("hello",)] = "world"  # cannot use list as key
d[3.14] = d # replaces 1592
d[1] += 1

# Removing
del d[1]
d.pop(3.14) # {("hello",): "world"}
```

Dictionary Method

`dct.clear()`	Removes all the elements in the dictionary dct
`dct.copy()`	Returns a copy of the dictionary with all the (key, value) pairs of dct
`dct.keys()`	Returns a sequence of all the keys of the dictionary dct
`dct.values()`	Returns a sequence of all the values of the dictionary dct
`dct.items()`	Returns a sequence of all the (key, value) pairs of the dictionary dct
`dct.get(<key>, <def>)`	Returns the value for the given key from `dct`, if the value is not in `dct`, returns None or the default value def if specified.

Iterating Dictionaries

```
# Check existence
if "key" not in d:
    d["key"] = {}

# Prints dictionary values
for key, value in d.items(): # d.items() => ((k1, v1), ...)
    print(value)

for key in d: # same as for key in d.keys():
    print(d[key])

for value in d.values():
    print(value)
```

Data Analysis

General workflow

This is very important for your PE too!

Step 1: Read the data

Step 2.1: Pre-process the data (filter, type conversion, etc)

Step 2.2: Process your goal

Step 3: Return the goal

Choosing the suitable data type is crucial.
`map` and `filter` makes your life easier.

Top-k example

This is very important for your PE too!

Step 1: Read the data

Step 2: Filter the data

Step 3: Sort the data
Sort the data according to what you need (Top IPPT Taker by score, we have to calculate the score, then use that to sort)

Step 4: Consider different cases of k, then return

Top-k

Step 4: Consider different cases of k

1) if k == 0 or k > (length of filtered and sorted data):
return data[0:k:1]

2) Tied cases! E.g. k = 3 and k < len(data)

```
# Calculate the k-th value (cutoff value)
top_k_value = data[k-1][2] # some index - 2 is arbitrary
# Filter the data!
output = list(filter(lambda k:k[2] >= top_k_value, data))
# Depending on question, transform the output
return output # (transformed)
```

Recap

Reading in CSV Files!

Basic CSV Reading function:

```
filename = 'test.csv'

def read_csv(csvfilename): # this will be given
    rows = ()
    with open(csvfilename) as csvfile:
        file_reader = csv.reader(csvfile)
        for row in file_reader:
            rows += (tuple(row), )
    return rows
```

(Recap) Using your shell interpreter

Step 1: Run the py file with the **read_csv** function
Step 2: In the shell interpreter try to interpret your data

```
>>> read_csv('test.csv')[0:5:1] # The slice is to show the
first 5 rows inclusive of the header row
(('Name', 'Age', 'Height'), ('Russell', '21', '190'),
('Daren', '35', '160'), ('Adi', '21', '177'), ('Markus',
'24', '185'))
```

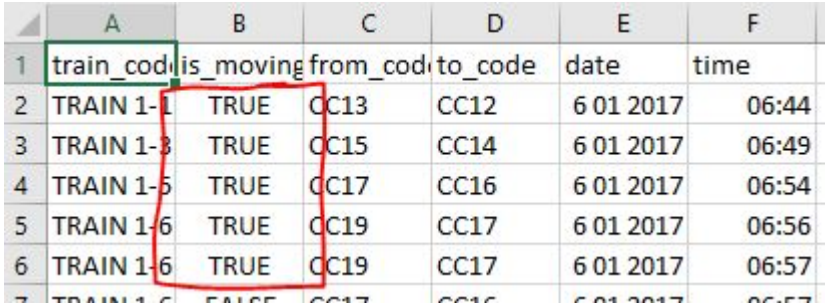
This is what your data looks like!

Step 3: Implement the requirements that the function has stated

(Recap) Using your shell interpreter

Don't use excel please… you won't have it during the PE

Also, take note that all values in read_csv are strings.



	A	B	C	D	E	F
1	train_cod	is_moving	from_cod	to_code	date	time
2	TRAIN 1-1	TRUE	CC13	CC12	6 01 2017	06:44
3	TRAIN 1-3	TRUE	CC15	CC14	6 01 2017	06:49
4	TRAIN 1-5	TRUE	CC17	CC16	6 01 2017	06:54
5	TRAIN 1-6	TRUE	CC19	CC17	6 01 2017	06:56
6	TRAIN 1-6	TRUE	CC19	CC17	6 01 2017	06:57

```
(('train_code', 'is_moving', 'from_code', 'to_code', 'date', 'time'),
 ('TRAIN 1-1', 'True', 'CC13', 'CC12', '06/01/2017', '06:44'),
 ('TRAIN 1-3', 'True', 'CC15', 'CC14', '06/01/2017', '06:49'),
 ('TRAIN 1-5', 'True', 'CC17', 'CC16', '06/01/2017', '06:54'),
 ('TRAIN 1-6', 'True', 'CC19', 'CC17', '06/01/2017', '06:56'))
```

Tutorial 8

PE
https://comp.nus.edu.sg/~cs1010s/glide

Question Introduction

Your friend recently made a lot of money buying Bitcoins, only to lose it all during the recent China ICO panic. Being wary of the returns, you obtained a price history the top few crypto- currencies to perform some analysis. The first line of the data file is a header which describes each column of data.

Year	Month	Day	Currency	Components...

You should only assume that **the first four columns are fixed** and that the rows are unique. The remaining columns are the components, which are not fixed, i.e. different data files can have different components. Your code should take this into account.

Hint: The method `List.index(item)` returns the index of the first matching item in the list.

Quick look

Take note that only the first four columns are fixed!
Year, Month, Day, Currency

```
>>> for row in read_csv('crypto.csv')[:5]:
        print(row)


['Year', 'Month', 'Day', 'Currency', 'Open', 'High', 'Low', 'Close', 'Volume', 'Market Cap']
['2017', 'Nov', '7', 'BTC', '7023.10', '7253.32', '7023.10', '7144.38', '2326340000', '117056000000']
['2017', 'Nov', '6', 'BTC', '7403.22', '7445.77', '7007.31', '7022.76', '3111900000', '123379000000']
['2017', 'Nov', '5', 'BTC', '7404.52', '7617.48', '7333.19', '7407.41', '2380410000', '123388000000']
['2017', 'Nov', '4', 'BTC', '7164.48', '7492.86', '7031.28', '7379.95', '2483800000', '119376000000']
```

Question 1: Data Analysis (PE 2a)

1. Implement the function `monthly_avg` takes as inputs a filename (`str`), currency (`str`), year (`int`) and component (`str`). It returns a dictionary where the keys are months, and the values are the monthly average of the given component of the given currency in the given year, rounded to 4 decimal places. You may assume that all the values in the requested component are floats.

   Note, if there are months with no data for the given inputs, then the month is not included in the returned dictionary.

   Hint: You can use the function `round(n, d)` to round n to d decimal places.

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    # Step 1: Read in the csv file

    # Step 2: Do some filtering based on conditions

    # Step 3: Do some processing to shape it into
    #         the desired output format.
    #         Either with for loops or map
```

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    # Step 1: Read in the csv file
    data = read_csv(fname)
    col = data[0].index(component) # find the column to index
    …
```

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    …
    # Step 2: Do some filtering based on conditions
    # Condition 1: correct year


    # Condition 2: correct currency
```

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    …
    # Step 2: Do some filtering based on conditions
    # Condition 1: correct year
    data = list(filter(lambda x:int(x[0]) == year, /
                data[1:len(data):1]))

    # data[1:] to exclude header row
    # int(x[0]) because x[0] is still a string (not int)
    # list typecasting in the end, don't forget!

    # data = list of rows where the year is correct
```

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    …
    # Step 2: Do some filtering based on conditions
    # Condition 1: correct year
    data = list(filter(lambda x:int(x[0]) == year, /
                data[1:len(data):1]))

    # Condition 2: correct currency
```

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    …
    # Step 2: Do some filtering based on conditions
    # Condition 1: correct year
    data = list(filter(lambda x:int(x[0]) == year, /
                data[1:len(data):1]))

    # Condition 2: correct currency
    data = list(filter(lambda x:x[3] == currency, data))

    # data is now a list of rows with the correct year
    # and currency (but all values are still strings!)
```

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    …
    # data is now a list of rows with the correct year
    # and currency (but all values are still strings!)

    # Step 3: Do some processing to shape it into
    #         the desired output format.
    #         Either with for loops or map
```

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    …
    # data is now a list of rows with the correct year
    # and currency (but all values are still strings!)

    # Step 3.1: group by month, collect all values to a list
    # Step 3.2: for each month, convert the list to a single
    # average
```

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    …
    # Step 3.1: group by month, collect all values to a list
    d = {}
    for row in data:
        month = row[1]
        if month not in d:
            d[month] = []
        d[month] += [float(row[col])]
```

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    …
    # Step 3.1: group by month, collect all values to a list
    d = {}
    for row in data:
        month = row[1]
        if month not in d:
            d[month] = []
        d[month] += [float(row[col])]
    # Step 3.2: for each month, convert the list to a single
    # average
```

Question 1: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def monthly_avg(fname, currency, year, component):
    # Step 3.1: group by month, collect all values to a list
    d = {}
    for row in data:
        month = row[1]
        if month not in d:
            d[month] = []
        d[month] += [float(row[col])]
    # Step 3.2: for each month, convert the list to a single
    # average
    for key, val in d.items():
        d[key] = round(sum(val)/len(val), 4) # reassign
    return d # done!
```

Question 1: Data Analysis (PE 2a)

```
# Putting it all together!
def monthly_avg(fname, currency, year, component):
    # Read
    data = read_csv(fname)
    col = data[0].index(component)
    # Filter
    data = list(filter(lambda x:int(x[0]) == year, data[1:len(data):1]))
    data = list(filter(lambda x:x[3] == currency, data))
    # Shape
    d = {}
    for row in data:
        month = row[1]
        if month not in d:
            d[month] = []
        d[month] += [float(row[col])]
    for key, val in d.items():
        d[key] = round(sum(val)/len(val), 4)
    return d
```

Question 2: Data Analysis (PE 2a)

2. We are now interested in computing the gain for each component in a month. The gain is calculated by taking the highest value of the component in the month and dividing it by the lowest value in the month. Since the result should be displayed as a percentage gain, it should be subtracted by 1 then multiplied by 100.

   For each month in a given year, we want to know which currency had the highest gain for a particular component amongst all the currencies.

   Implement the function `highest_gain` takes as input a filename (`str`), a year (`int`), and a component (`str`), and returns a dictionary where the keys are the months and the values are a tuple of two elements: the currency and the gain (rounded to 2 decimal places).

   You may assume that the values for the components are either integers or floats. Note that it is possible for some values to be missing, in which case it is denoted by a `'-'`. Such rows should be ignored.

Question 2: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def highest_gain(fname, year, component):
    # Step 1: Read in the csv file

    # Step 2: Do some filtering based on conditions

    # Step 3: Do some processing to shape it into
    #         the desired output format.
    #         Either with for loops or map
```

Question 2: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def highest_gain(fname, year, component):
    # Step 1: Read in the csv file
    data = read_csv(fname)
    col = data[0].index(component) # find the column to index
    …
```

Question 2: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def highest_gain(fname, year, component):
    …
    # Step 2: Do some filtering based on conditions
    data = list(filter(lambda x:int(x[0]) == year, /
                data[1:len(data):1]))
```

Question 2: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def highest_gain(fname, year, component):
    …
    # Step 3: Do some processing to shape it into
    #         the desired output format.
```

Question 2: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def highest_gain(fname, year, component):
    …
    # Step 3: Do some processing to shape it into
    #         the desired output format.
    # Step 3.1: Group by month and among
    # one month group by currency, collect values to a list
    # Step 3.2: For each list, convert to the gain %-age
    # Step 3.3: For each month, get the maximum %-age
```

Question 2: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def highest_gain(fname, year, component):
    …
    # Step 3.1: Group by month and among
    # one month group by currency, collect values to a list
    d = {}
    for row in data:
        if row[col] == '-':
            continue
        month, curr = row[1], row[3]
        if month not in d:
            d[month] = {}  # new dict for currency
        if curr not in d[month]:
            d[month][curr] = []
        d[month][curr] += [float(row[col])]
```

Question 2: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def highest_gain(fname, year, component):
    …
    # Step 3.2: For each list, convert to the gain %-age
    for month, val in d.items():
        for curr, vol in val.items():
            val[curr] = round((max(vol)/min(vol)-1)*100, 2)
```

Question 2: Data Analysis (PE 2a)

```
# Standard structure of a PE question
def highest_gain(fname, year, component):
    …
    # Step 3.2: For each list, convert to the gain %-age
    for month, val in d.items():
        for curr, vol in val.items():
            val[curr] = round((max(vol)/min(vol)-1)*100, 2)
        # Step 3.3: For each month, get the maximum %-age
        d[month] = max(val.items(), key=lambda x:x[1])
    return d # done
```

Question 2: Data Analysis (PE 2a)

```
# Putting it all together!
def highest_gain(fname, year, component):
    data = read_csv(fname)
    col = data[0].index(component)
    data = list(filter(lambda x:x[0] == str(year), data[1:len(data):1]))
    d = {}
    for row in data:
        if row[col] == '-':
            continue
        month, curr = row[1], row[3]
        if month not in d:
            d[month] = {}
        if curr not in d[month]:
            d[month][curr] = []
        d[month][curr] += [float(row[col])]
    for month, val in d.items():
        for curr, vol in val.items():
            val[curr] = round((max(vol)/min(vol)-1)*100, 2)
        d[month] = max(val.items(), key=lambda x:x[1])
    return d
```

The End