Northeastern University
DS2001 – Programming with Data — Social Science Practicum
Fall 2019, Mattsson & McCabe

Practicum 6: Summarization

In this week's practicum we will be asking Python to summarize large amounts of data in a way that we could use for multiple datasets. We will write functions that make exploring data more tractable, and apply them to the datasets we are already familiar with.

1 Summarizing data from multiple datasets

We will be considering the same three datasets as we have in previous exercises:

1920 Census, Nevada The US Census is conducted every 10 years and is used in many aspects of public administration. This dataset covers the population of Nevada.

Boston Blue Bikes, August 2019 The Boston Blue Bikes system keeps track of the trips that users take using their system, for the purposes of billing, real-time monitoring, inventory management, and data science. This dataset covers all trips taken in August, 2019.

Baseball Baseball lends itself to statistical analysis, especially in that performance is readily quantifiable and such data has been collected in the same manner since the 1930s. This dataset covers batting performances since 1969 from the well-known Lahman database.

1.1 Download

Please download `pr06.zip` from the practicum website (or Blackboard), unzip it, and move the directory/folder to where you want it in your file system. Use Atom's `File > "Add Project Folder..."` to open this folder and view both `summarize.py` and `data.py`.

Built-in modules When programming tasks get bigger, it becomes important to organize your code. Python does this using `modules`. To use a function from a built-in module, such as the `math` module, you would use the following syntax:

```
import MODULE
result = MODULE.specific_function(ARGUMENT_1,ARGUMENT_2)
```

Custom modules It can also be useful to create modules for your own functions. In fact, any Python file can be a module! As long as `MODULE.py` is in the same folder as the code you are running, then the same statement would let you access these functions.

1.2 Load the datasets

The file `data.py` defines a module with several functions that will be useful to you. Please write the `import` statement at the top of your code in `summarize.py`, and use the functions provided in `data.py` to read in the three datasets and their headers from the following files:

- `"census.csv"`
- `"bluebikes.csv"`
- `"baseball.csv"`

Make sure to give each dataset a good name!

1.3 View the datasets

These files are large. Thankfully, `data.py` also includes the `head` function that we implemented last week. Use this function to print out a few rows of each dataset and confirm that they are what you expect.

2 Maximum distance of Blue Bikes, revisited

2.1 Looping to store a new list

Previously, we calculated the maximum distance of Blue Bike #5561 like so:

```
max_distance = 0

for trip in bluebike_data:
    trip_distance = trip[7]
    if trip_distance > max_distance:
        max_distance = trip_distance

print("max_distance:", max_distance)
```

This works perfectly well, but what if we want to know also the minimum? The mean? It becomes cumbersome to write out every variable we want to track. Please implement this same functionality using a for loop that builds up a list of all the distances of all the trips in the bluebikes data. Use the `min`, `mean`, and `max` functions to calculate the desired summary statistics.

2.2 List comprehension

Looping over a list to build up another list is *such* a common thing to do that Python has a special syntax for it: `list comprehensions`. It is important to remember that they are *exactly* like the `for` loop you just wrote. Syntax that looks effortless is great, but harder to debug.

```
new_list = [ <expression> for ELEMENT in LIST ]
```

Please use a `list comprehension` to create the same list as before. Again, use the `min`, `mean`, and `max` functions to summarize the trip distances in the Blue Bikes dataset. Confirm that you get the same answer.

3 Summarizing numerical data

Finding the `min`, `mean`, and `max` of a numerical column is useful for many columns of the data in many of our three datasets. In this section we will generalize the previous section, building up a function that summarizes the numerical columns of any dataset we give it.

3.1 Get the i^{th} column

Please write a function that takes as input a `dataset` and a `col_index`, then returns that column of the data. Use a list comprehension.

3.2 Identify a numeric column

Please write a function that takes as input a column (i.e., a list) and returns `True` if the list contains `int`s or `float`s, and `False` otherwise. You may assume from the structure of the dataset that every element of a column will be of the same type (so you'd only need to test one element of the column list).

3.3 Print the names and indices of every column

Please write a function that takes as input a `header`, and that prints out both the name and the index of every column. This function should not return anything. Hint: use `range(len(header))`.

3.4 Print the name, min, mean, and max of the numerical columns

Please write a function that takes as input a `header` and a `dataset`, and that summarizes each of the numerical columns. This function should not return anything, but it should:

1. Loop over the columns
2. Select the columns
3. Print out the name, max, min, and mean of the numerical columns
4. Ignore the non-numerical columns

3.5 Summarize the three datasets

Use your function to summarize the three available datasets.

4 Filtering datasets, revisited

If you look in `data.py` you'll see that it includes also the filtering functions that we implemented last week. If you look closely, you'll see that these *also* follow the pattern of looping over a list to build up another list. The for loop doesn't *change* the entires of the old list as it is loops over them, but it does *filter out* some of the original entries.

List comprehension with a condition A `list comprehension` can both *change* and *filter out* the entries of an old list to build up a new one. These so the same thing as a `for` loop with an `if` statement.

```
new_list = [ <expression> for ELEMENT in LIST if <condition> ]
```

4.1 Implement another version of the filter functions

Re-write the `for` loops in the `data.filter` functions using list comprehension instead. Use these functions to find the following subsets of data:

- Residents of Pahrump, Nevada (Hint: "PAHRUMP PRECINCT")
- Boston Blue Bike #5561
- Washington Nationals (WAS) players

Use the `head` function to make sure it's working!

5 (extra) Summarize also the non-numeric columns

In the main exercise we summarized the numerical columns. We would also like to be able to summarize the other columns.

5.1 Counters

Within Python's `collections` module there is a data structure that will be useful to you: a `Counter`. Look up the documentation for `Counters` and understand what they do.

5.2 Top 3 strings

Write a function that uses a `Counter` to find the top three most common strings in a list of strings.

5.3 Modified summary function

Modify your summary function to summarize both numerical and string columns, using min, mean, max for the former and the top 3 for the latter.

6 Baseball codebook

Index	Header	Detail
0	first_name	
1	last_name	
2	year	Season
3	stint	The nth team someone played for this season
4	team	Team name (short ID)
5	league	League (AL/NL)
6	AB	Times at bat
7	H	Hits
8	HR	Home runs
9	SB	Stolen bases
10	BB	Walks
11	SO	Strikeouts
12	HBP	Times hit by pitch
13	PA	Plate appearances