

Lahti2014_solution

January 23, 2017

1 Solution of Lahti *et al.* 2014

1.0.1 Write a function that takes as input a dictionary of constraints (i.e., selecting a specific group of records) and returns a dictionary tabulating the BMI group for all the records matching the constraints. For example, calling:

```
get_BMI_count({'Age': '28', 'Sex': 'female'})
```

1.0.2 should return:

```
{'NA': 3, 'lean': 8, 'overweight': 2, 'underweight': 1}
```

```
In [4]: import csv # Import csv module for reading the file
```

For each row in the file, you need to make sure all the constraints are matching the desired ones. If so, keep count of the BMI group using a dictionary.

```
In [5]: def get_BMI_count(dict_constraints):
        """ Take as input a dictionary of constraints
            for example, {'Age': '28', 'Sex': 'female'}
            And return the count of the various groups of BMI
        """
        # We use a dictionary to store the results
        BMI_count = {}
        # Open the file, build a csv DictReader
        with open('../data/Lahti2014/Metadata.tab') as f:
            csvr = csv.DictReader(f, delimiter = '\t')
            # For each row
            for row in csvr:
                # check that all conditions are met
                matching = True
                for e in dict_constraints:
                    if row[e] != dict_constraints[e]:
                        # The constraint is not met. Move to the next record
                        matching = False
                        break
                # matching is True only if all the constraints have been met
                if matching == True:
```

```

        # extract the BMI_group
        my_BMI = row['BMI_group']
        if my_BMI in BMI_count.keys():
            # If we've seen it before, add one record to the count
            BMI_count[my_BMI] = BMI_count[my_BMI] + 1
        else:
            # If not, initialize at 1
            BMI_count[my_BMI] = 1
    return BMI_count

```

```
In [6]: get_BMI_count({'Nationality': 'US', 'Sex': 'female'})
```

```
Out[6]: {'lean': 12, 'obese': 3, 'overweight': 5, 'severeobese': 1, 'underweight': 3}
```

1.0.3 Write a function that takes as input the constraints (as above), and a bacterial "genus". The function returns the average abundance (in logarithm base 10) of the genus for each group of BMI in the sub-population. For example, calling:

```
get_abundance_by_BMI({'Time': '0', 'Nationality': 'US'}, 'Clostridium difficile et rel.')
```

1.0.4 should return:

```
-----
Abundance of Clostridium difficile et rel. In sub-population:
-----
```

```
Nationality -> US
```

```
Time -> 0
```

```
-----
3.08      NA
3.31      underweight
3.84      lean
2.89      overweight
3.31      obese
3.45      severeobese
-----
```

```
In [7]: import scipy # For log10
```

```

def get_abundance_by_BMI(dict_constraints, genus = 'Aerococcus'):
    # We use a dictionary to store the results
    BMI_IDs = {}
    # Open the file, build a csv DictReader
    with open('../data/Lahti2014/Metadata.tab') as f:
        csvr = csv.DictReader(f, delimiter = '\t')
        # For each row
        for row in csvr:
            # check that all conditions are met
            matching = True
            for e in dict_constraints:

```

```

        if row[e] != dict_constraints[e]:
            # The constraint is not met. Move to the next record
            matching = False
            break
    # matching is True only if all the constraints have been met
    if matching == True:
        # extract the BMI_group
        my_BMI = row['BMI_group']
        if my_BMI in BMI_IDs.keys():
            # If we've seen it before, add the SampleID
            BMI_IDs[my_BMI] = BMI_IDs[my_BMI] + [row['SampleID']]
        else:
            # If not, initialize
            BMI_IDs[my_BMI] = [row['SampleID']]
# Now let's open the other file, and keep track of the abundance of the genus for each
# BMI group
abundance = {}
with open('../data/Lahti2014/HITChip.tab') as f:
    csvr = csv.DictReader(f, delimiter = '\t')
    # For each row
    for row in csvr:
        # check whether we need this SampleID
        matching = False
        for g in BMI_IDs:
            if row['SampleID'] in BMI_IDs[g]:
                if g in abundance.keys():
                    abundance[g][0] = abundance[g][0] + float(row[genus])
                    abundance[g][1] = abundance[g][1] + 1
                else:
                    abundance[g] = [float(row[genus]), 1]
                # we have found it, so move on
                break
# Finally, calculate means, and print results
print("-----")
print("Abundance of " + genus + " In sub-population:")
print("-----")
for key, value in dict_constraints.items():
    print(key, "->", value)
print("-----")
for ab in ['NA', 'underweight', 'lean', 'overweight',
           'obese', 'severeobese', 'morbidobese']:
    if ab in abundance.keys():
        abundance[ab][0] = scipy.log10(abundance[ab][0] / abundance[ab][1])
        print(round(abundance[ab][0], 2), '\t', ab)
print("-----")
print("")

```

```
In [8]: get_abundance_by_BMI({'Time': '0', 'Nationality': 'US'},
                             'Clostridium difficile et rel.')
```

```
-----
Abundance of Clostridium difficile et rel. In sub-population:
```

```
-----
Nationality -> US
Time -> 0
```

```
-----
3.08      NA
3.31      underweight
3.84      lean
2.89      overweight
3.31      obese
3.45      severeobese
-----
```

1.0.5 Repeat this analysis for all genera, and for the records having Time = 0.

A function to extract all the genera in the database:

```
In [10]: def get_all_genera():
          with open('../data/Lahti2014/HITChip.tab') as f:
              header = f.readline().strip()
              genera = header.split('\t')[1:]
          return genera
```

Testing:

```
In [7]: get_all_genera()[:6]
```

```
Out[7]: ['Actinomycetaceae',
         'Aerococcus',
         'Aeromonas',
         'Akkermansia',
         'Alcaligenes faecalis et rel.',
         'Allistipes et rel.']
```

Now use this function to print the results for all genera at Time = 0:

```
In [8]: for g in get_all_genera()[:5]:
          get_abundance_by_BMI({'Time': '0'}, g)
```

```
-----
Abundance of Actinomycetaceae In sub-population:
```

```
-----
Time -> 0
```

1.98	NA
1.95	underweight
1.98	lean
1.97	overweight
1.93	obese
1.95	severeobese
1.9	morbidobese

Abundance of Aerococcus In sub-population:

Time -> 0

1.66	NA
1.63	underweight
1.66	lean
1.66	overweight
1.61	obese
1.62	severeobese
1.6	morbidobese

Abundance of Aeromonas In sub-population:

Time -> 0

1.68	NA
1.68	underweight
1.69	lean
1.69	overweight
1.66	obese
1.66	severeobese
1.63	morbidobese

Abundance of Akkermansia In sub-population:

Time -> 0

3.53	NA
4.0	underweight
3.65	lean
3.71	overweight
3.52	obese

3.48	severeobese
3.35	morbidobese

Abundance of *Alcaligenes faecalis* et rel. In sub-population:

Time -> 0

2.32	NA
2.26	underweight
2.36	lean
2.37	overweight
2.49	obese
2.43	severeobese
2.26	morbidobese
