Lesson 3: Problem set3: Testing Data Quality

Testing Data Quality

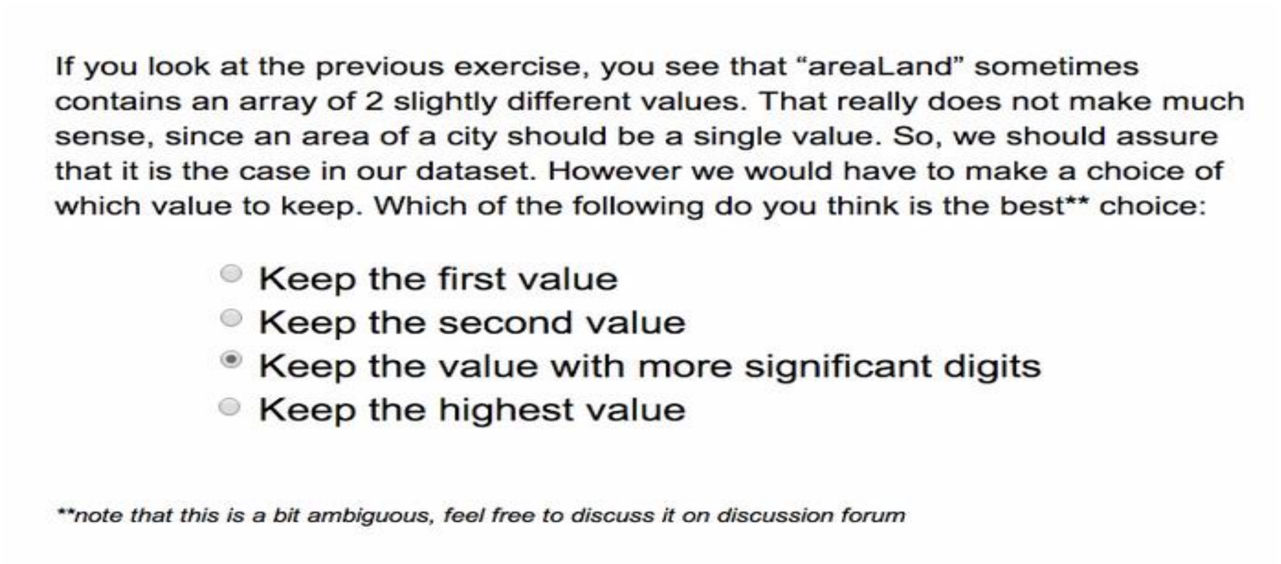
audit.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
In this problem set you work with cities infobox data, audit it, come up with a cleaning idea and then
clean it up. In the first exercise we want you to audit the datatypes that can be found in some
particular fields in the dataset.
The possible types of values can be:
- 'NoneType' if the value is a string "NULL" or an empty string ""
- 'list', if the value starts with "{"
- 'int', if the value can be cast to int
- 'float', if the value can be cast to float, but is not an int
- 'str', for all other values

The audit_file function should return a dictionary containing fieldnames and a set of the datatypes
that can be found in the field.
All the data initially is a string, so you have to do some checks on the values first.
"""
import codecs
import csv
import json
import pprint

CITIES = 'cities.csv'

FIELDS = ["name", "timeZone_label", "utcOffset", "homepage", "governmentType_label",
"isPartOf_label", "areaCode", "populationTotal",
        "elevation", "maximumElevation", "minimumElevation", "populationDensity", "wgs84_pos#lat",
"wgs84_pos#long",
        "areaLand", "areaMetro", "areaUrban"]

def int_type(n):
   try:
      return int(n)
   except ValueError:
      return False

def float_type(n):
   try:
      return float(n)
   except ValueError:
      return False

def audit_file(filename, fields):
    fieldtypes = {}

    # YOUR CODE HERE
    for eachfield in fields:
        fieldtypes[eachfield] = []
    with open(filename, "r") as f:
        reader = csv.DictReader(f)
        count = 0
        for eachline in reader:
            count += 1
            if count < 4:
                continue
            for eachfield in fields:
                if eachline[eachfield] == 'NULL' or eachline[eachfield] == '':
                    #print 'found a null'
                    fieldtypes[eachfield].append(type(None))
                elif eachline[eachfield][0] == '{':
                    #print 'found a list'
                    fieldtypes[eachfield].append(type([]))
                elif int_type(eachline[eachfield]):
                    #print 'found an INT'
                    fieldtypes[eachfield].append(type(1))
                elif float_type(eachline[eachfield]):
                    #print 'found a FLOAT'
                    fieldtypes[eachfield].append(type(1.1))
                else:
                    fieldtypes[eachfield].append(type('str'))

    for key, value in fieldtypes.iteritems():
        fieldtypes[key] = set(fieldtypes[key])

    return fieldtypes

def test():
    fieldtypes = audit_file(CITIES, FIELDS)

    pprint.pprint(fieldtypes)

    assert fieldtypes["areaLand"] == set([type(1.1), type([]), type(None)])
    assert fieldtypes['areaMetro'] == set([type(1.1), type(None)])

if __name__ == "__main__":
    test()
```

audit.py Output:

```
{'areaCode': set([<type 'NoneType'>, <type 'str'>, <type 'int'>]),
 'areaLand': set([<type 'NoneType'>, <type 'list'>, <type 'float'>]),
'areaMetro': set([<type 'NoneType'>, <type 'float'>]),
'areaUrban': set([<type 'NoneType'>, <type 'float'>]),
'elevation': set([<type 'NoneType'>,
             <type 'list'>,
             <type 'str'>,
             <type 'float'>]),
'governmentType_label': set([<type 'NoneType'>, <type 'str'>]),
'homepage': set([<type 'NoneType'>, <type 'str'>]),
'isPartOf_label': set([<type 'NoneType'>, <type 'list'>, <type 'str'>]),
'maximumElevation': set([<type 'NoneType'>]),
'minimumElevation': set([<type 'NoneType'>]),
'name': set([<type 'NoneType'>, <type 'list'>, <type 'str'>]),
'populationDensity': set([<type 'NoneType'>, <type 'list'>, <type 'float'>]),
'populationTotal': set([<type 'NoneType'>, <type 'int'>]),
'timeZone_label': set([<type 'NoneType'>, <type 'str'>]),
'utcOffset': set([<type 'NoneType'>,
             <type 'list'>,
             <type 'str'>,
             <type 'int'>]),
'wgs84_pos#lat': set([<type 'float'>]),
'wgs84_pos#long': set([<type 'float'>])}
```

Making Choice:



If you look at the previous exercise, you see that "areaLand" sometimes contains an array of 2 slightly different values. That really does not make much sense, since an area of a city should be a single value. So, we should assure that it is the case in our dataset. However we would have to make a choice of which value to keep. Which of the following do you think is the best** choice:

- Keep the first value
- Keep the second value
- [✓ Keep the value with more significant digits]
- Keep the highest value

***note that this is a bit ambiguous, feel free to discuss it on discussion forum*

Yes! Keeping the value with most significant digits makes the most sense, since it most likely comes from the most reliable source.

Fixing the Area

area.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
In this problem set you work with cities infobox data, audit it, come up with a cleaning idea and then
clean it up.

Since in the previous quiz you made a decision on which value to keep for the "areaLand" field,
you now know what has to be done.

Finish the function fix_area(). It will receive a string as an input, and it has to return a float
representing the value of the area or None.
You have to change the function fix_area. You can use extra functions if you like, but changes to
process_file
will not be taken into account.
The rest of the code is just an example on how this function can be used.
"""
import codecs
import csv
import json
import pprint

CITIES = 'cities.csv'

def fix_area(area):

    # YOUR CODE HERE
    if area == "NULL":
        return None
    elif area[0] == "{":
        l = area.strip("{}").split("|")
        # Get string without 0's
        l1, l2 = str(l[0]).replace("e+", "").replace("0", ""), str(l[1]).replace("e+", "").replace("0", "")
        # Compare length of non-zero "significant" digits
        # Then return original which has more as a float
        if len(l1) > len(l2):
            return float(l[0])
        else:
            return float(l[1])
    return float(area)

def process_file(filename):
    # CHANGES TO THIS FUNCTION WILL BE IGNORED WHEN YOU SUBMIT THE EXERCISE
    data = []

    with open(filename, "r") as f:
        reader = csv.DictReader(f)
        #skipping the extra metadata
        for i in range(3):
            l = reader.next()

        # processing file
        for line in reader:
            # calling your function to fix the area value
            if "areaLand" in line:
                line["areaLand"] = fix_area(line["areaLand"])
            data.append(line)

    return data

def test():
    data = process_file(CITIES)

    print "Printing three example results:"
    for n in range(5,8):
        pprint.pprint(data[n]["areaLand"])

    assert data[8]["areaLand"] == 55166700.0
    assert data[3]["areaLand"] == None

if __name__ == "__main__":
    test()
```

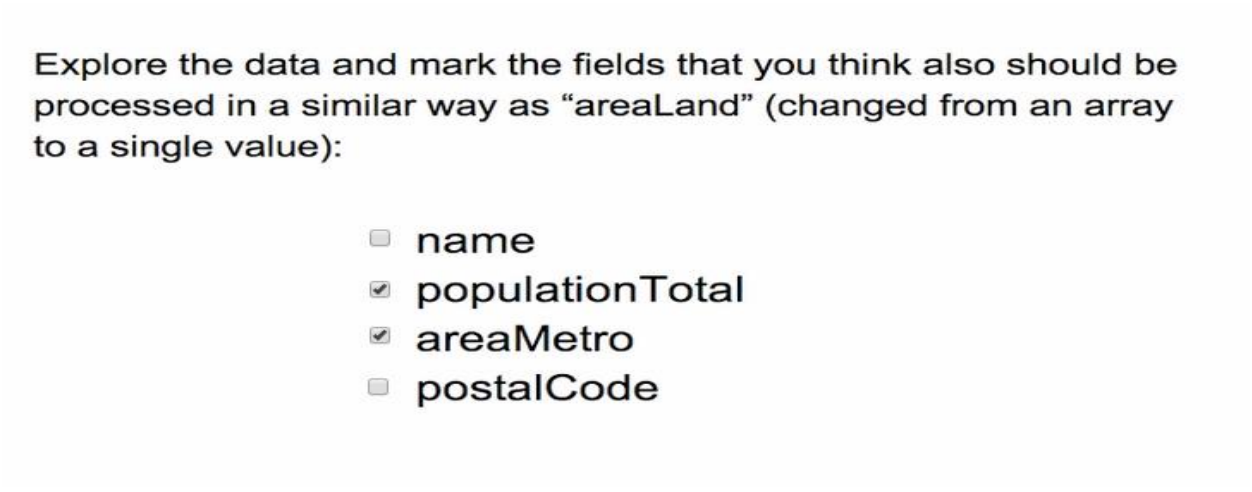
area.py Output:

```
Printing three example results:
None
101787000.0
31597900.0
```

```
Nice work fixing the area!
```

Other Fields

Answers:



Explore the data and mark the fields that you think also should be processed in a similar way as “areaLand” (changed from an array to a single value):

- ☐ name
- [✓ populationTotal]
- [✓ areaMetro]
- ☐ postalCode

<u>Fixing Name</u>

<u>name.py</u>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
In this problem set you work with cities infobox data, audit it, come up with a cleaning idea and then
clean it up.
In the previous quiz you recognized that the "name" value can be an array (or list in Python terms).
It would make it easier to process and query the data later, if all values for the name
would be in a Python list, instead of being just a string separated with special characters, like now.
Finish the function fix_name(). It will recieve a string as an input, and it has to return a list
of all the names. If there is only one name, the list with have only one item in it, if the name is "NULL",
the list should be empty.
The rest of the code is just an example on how this function can be used
"""
import codecs
import csv
import pprint

CITIES = 'cities.csv'

def fix_name(name):

  if name[0] == '{':
    name = name.replace('{','').replace('}','')
    name = name.split('|')
  elif name == 'NULL':
    name = []
  else:
    name = [name]

  return name
```

```
def process_file(filename):
    data = []
    with open(filename, "r") as f:
        reader = csv.DictReader(f)
        #skipping the extra matadata
        for i in range(3):
            l = reader.next()
        # processing file
        for line in reader:
            # calling your function to fix the area value
            if "name" in line:
                line["name"] = fix_name(line["name"])
            data.append(line)
    return data

def test():
    data = process_file(CITIES)

    print "Printing 20 results:"
    for n in range(20):
        pprint.pprint(data[n]["name"])

    assert data[14]["name"] == ['Negtemiut', 'Nightmute']
    assert data[3]["name"] == ['Kumhari']

if __name__ == "__main__":
    test()
```

Name.py Output:

```
Printing 20 results:
['Kud']
['Kuju']
['Kumbhraj']
['Kumhari']
['Kunigal']
['Kurgunta']
['Athens']
['Demopolis']
['Chelsea Alabama']
['Pell City Alabama']
['City of Northport']
['Sand Point']
['Unalaska Alaska']
['City of Menlo Park']
```

```
['Negtemiut', 'Nightmute']
['Fairbanks Alaska']
['Homer']
['Ketchikan Alaska']
['Nuniaq', 'Old Harbor']
['Rainier Washington']
```

```
Great! All city name values are in a list.

The 'name' value for the tested single city name is correct!

The 'name' value for the tested array is correct!

You correctly processed the tested 'NULL' name!
```

Crossfield Auditing:

Location.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
In this problem set you work with cities infobox data, audit it, come up with a cleaning idea and then
clean it up.
If you look at the full city data, you will notice that there are couple of values that seem to provide
the same information in different formats: "point" seems to be the combination of "wgs84_pos#lat" and
"wgs84_pos#long".
However we do not know if that is the case and should check if they are equivalent.
Finish the function check_loc(). It will recieve 3 strings, first will be the combined value of "point" and
then the
"wgs84_pos#" values separately. You have to extract the lat and long values from the "point" and
compare
to the "wgs84_pos# values and return True or False.
Note that you do not have to fix the values, just determine if they are consistent. To fix them in this case
you would need more information. Feel free to discuss possible strategies for fixing this on the
discussion forum.
The rest of the code is just an example on how this function can be used.
Changes to "process_file" function will not be take into account.
"""
import csv
import pprint

CITIES = 'cities.csv'

def check_loc(point, lat, longi):
    points = point.split(' ')
    same = False
    if points[0] == lat and points[1] == longi:
        same = True

    return same

def process_file(filename):
    data = []
    with open(filename, "r") as f:
        reader = csv.DictReader(f)
        #skipping the extra matadata
        for i in range(3):
            l = reader.next()
        # processing file
        for line in reader:
            # calling your function to check the location
            result = check_loc(line["point"], line["wgs84_pos#lat"], line["wgs84_pos#long"])
            if not result:
                print "{}: {} != {} {}".format(line["name"], line["point"], line["wgs84_pos#lat"],
line["wgs84_pos#long"])
            data.append(line)

    return data

def test():
    assert check_loc("33.08 75.28", "33.08", "75.28") == True
    assert check_loc("44.57833333333333 -91.21833333333333", "44.5783", "-91.2183") == False

if __name__ == "__main__":
    test()
```

location.py output:

```
Congratulations, you correctly tested the location values!
```