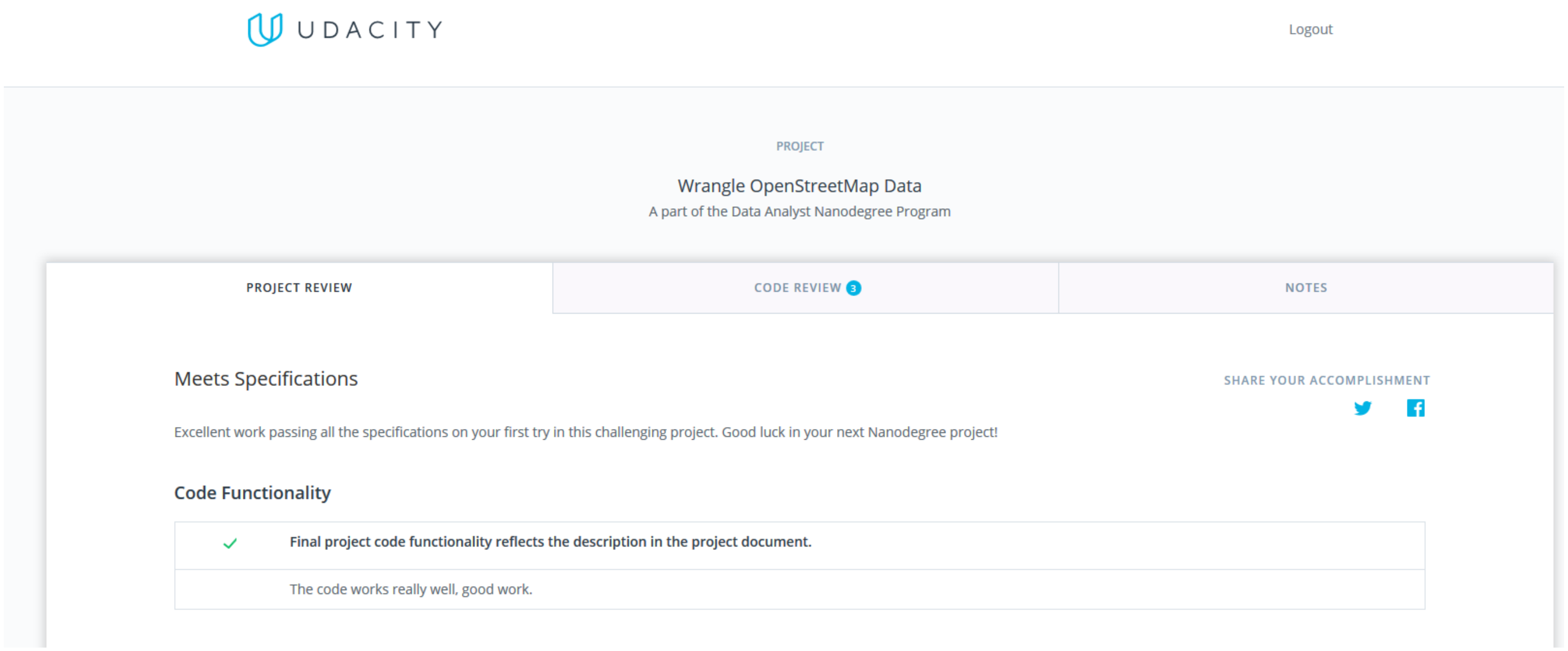


PROJECT

Wrangle OpenStreetMap Data

A part of the Data Analyst Nanodegree Program

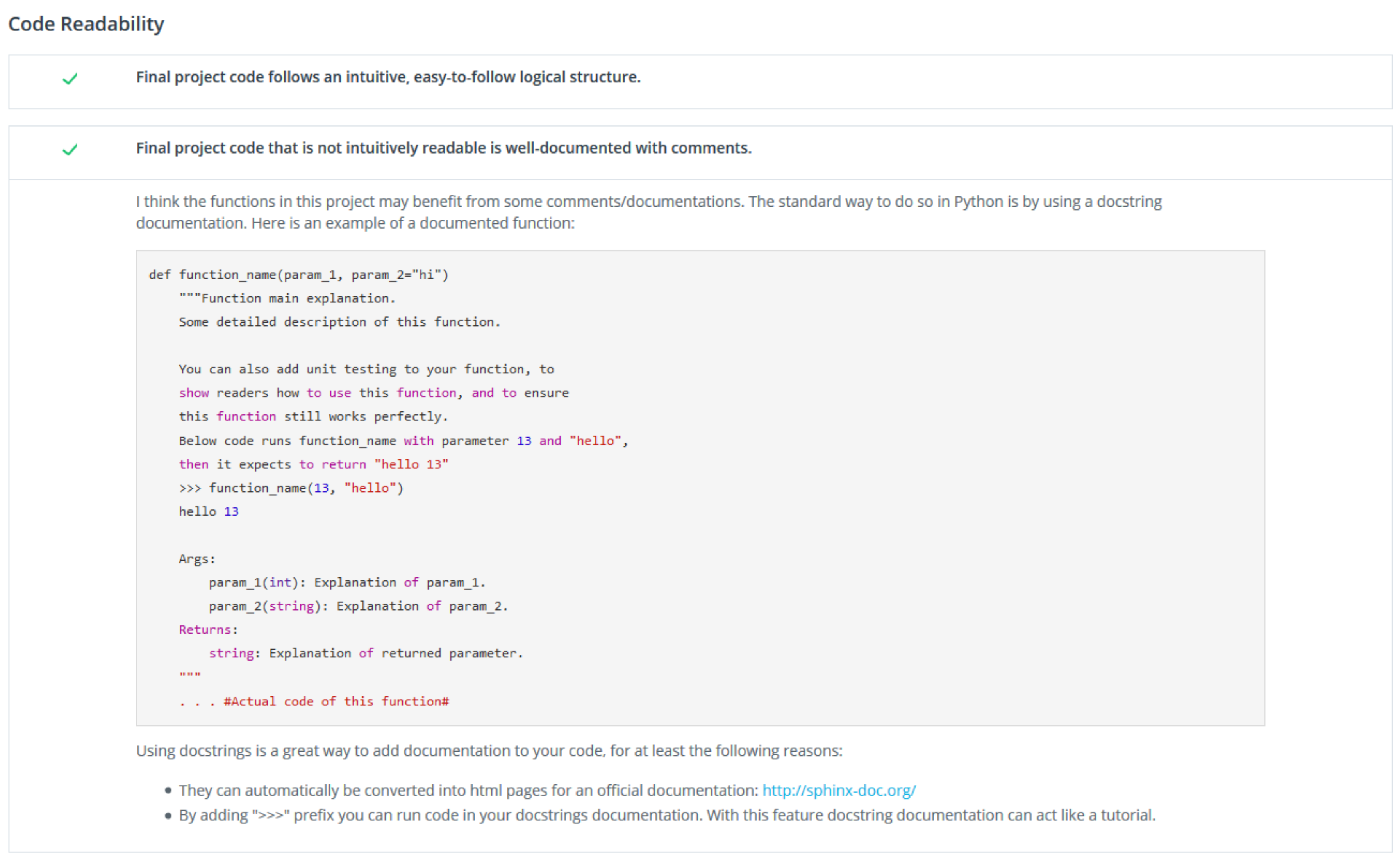
Meets Specifications

Excellent work passing all the specifications on your first try in this challenging project. Good luck in your next Nanodegree project!

Code Functionality

✓ **Final project code functionality reflects the description in the project document.**

The code works really well, good work.



Code Readability

✓ **Final project code follows an intuitive, easy-to-follow logical structure.**

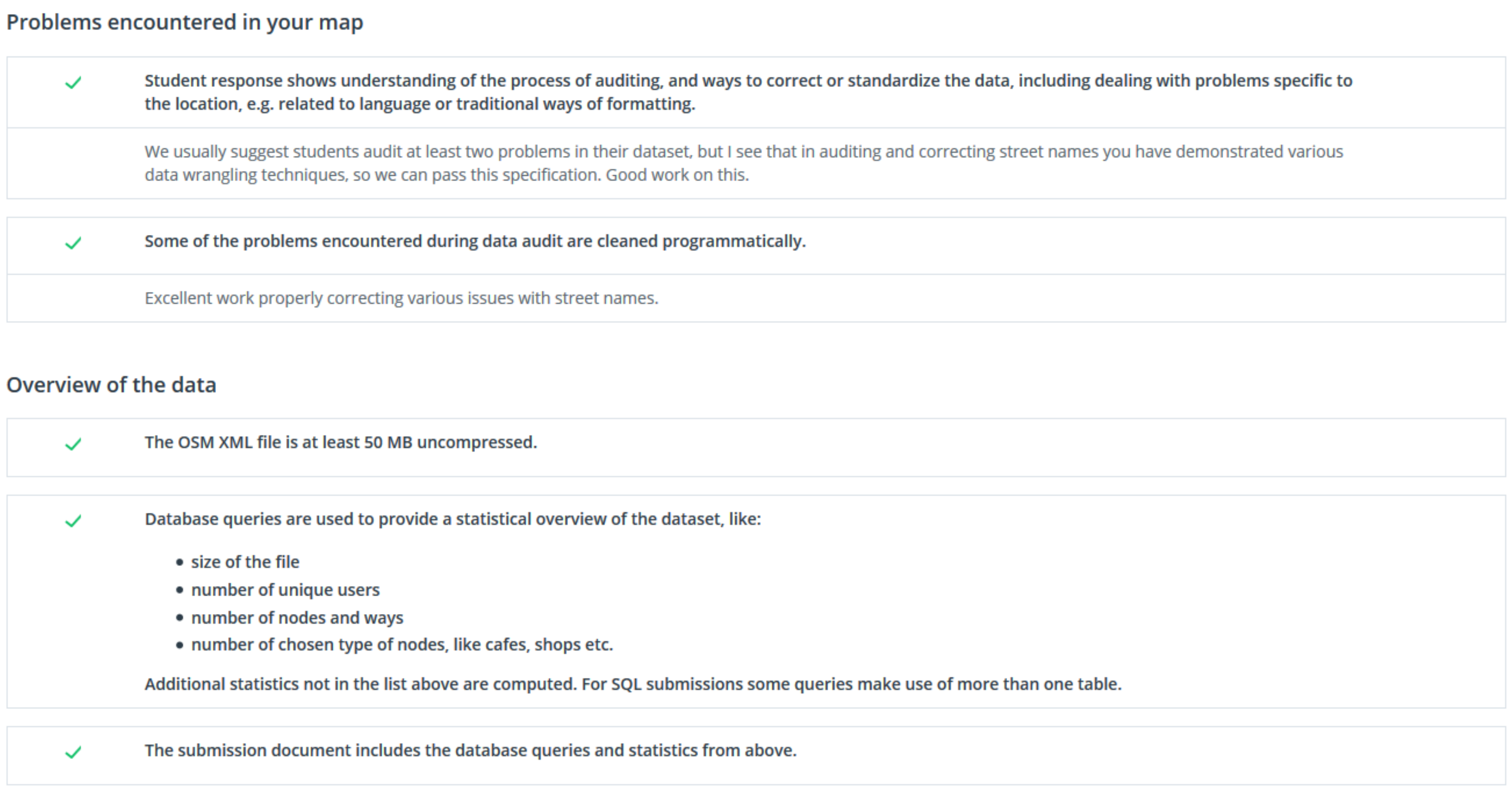
✓ **Final project code that is not intuitively readable is well-documented with comments.**

I think the functions in this project may benefit from some comments/documentations. The standard way to do so in Python is by using a docstring documentation. Here is an example of a documented function:

```
def function_name(param_1, param_2="hi")
    """Function main explanation.
    Some detailed description of this function.

    You can also add unit testing to your function, to
    show readers how to use this function, and to ensure
    this function still works perfectly.
    Below code runs function_name with parameter 13 and "hello",
    then it expects to return "hello 13"
    >>> function_name(13, "hello")
    hello 13

    Args:
        param_1(int): Explanation of param_1.
        param_2(string): Explanation of param_2.
    Returns:
        string: Explanation of returned parameter.
    """
    . . . #Actual code of this function#
```

Using docstrings is a great way to add documentation to your code, for at least the following reasons:

- They can automatically be converted into html pages for an official documentation: http://sphinx-doc.org/
- By adding ">>>" prefix you can run code in your docstrings documentation. With this feature docstring documentation can act like a tutorial.



Problems encountered in your map

✓ **Student response shows understanding of the process of auditing, and ways to correct or standardize the data, including dealing with problems specific to the location, e.g. related to language or traditional ways of formatting.**

We usually suggest students audit at least two problems in their dataset, but I see that in auditing and correcting street names you have demonstrated various data wrangling techniques, so we can pass this specification. Good work on this.

✓ **Some of the problems encountered during data audit are cleaned programmatically.**

Excellent work properly correcting various issues with street names.

Overview of the data

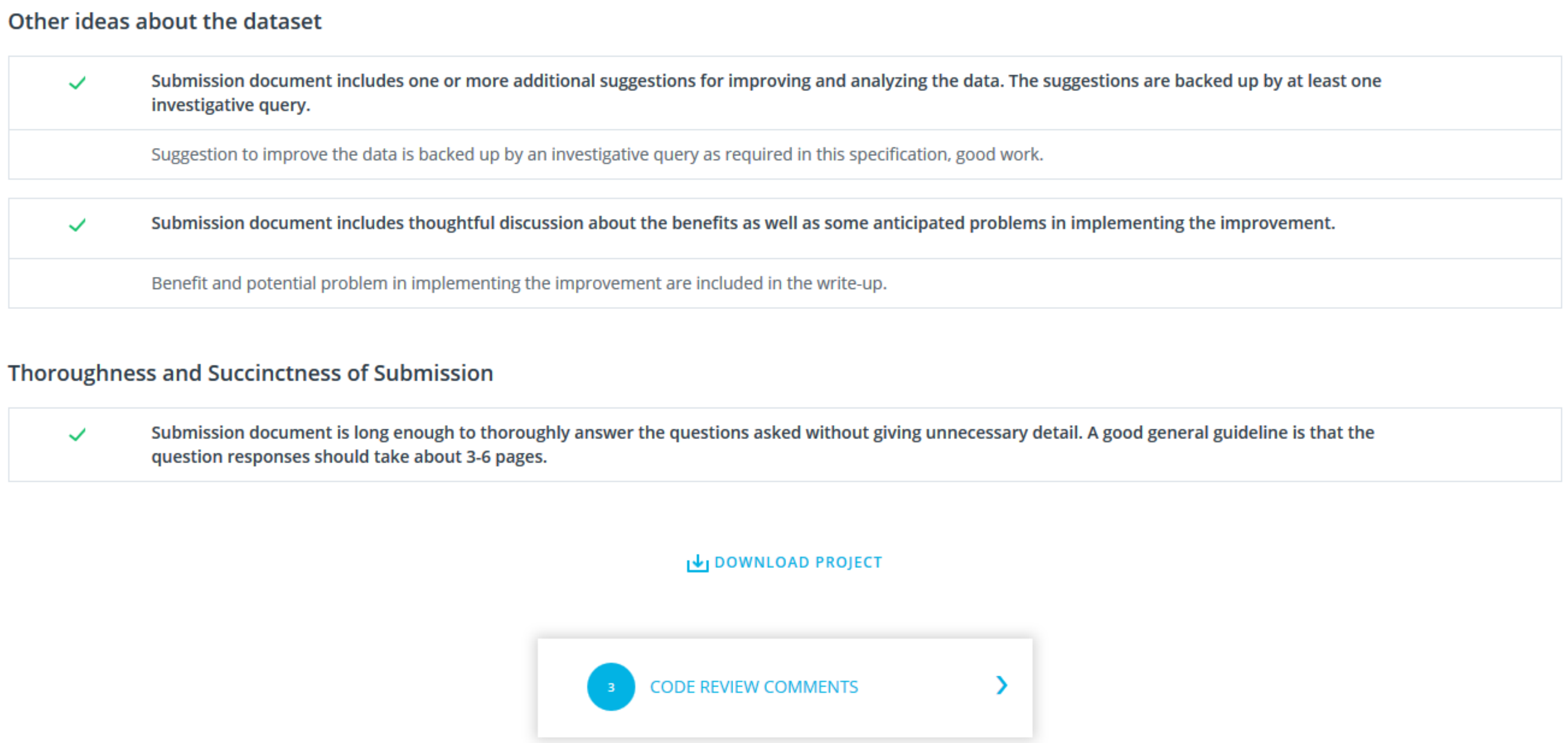
✓ **The OSM XML file is at least 50 MB uncompressed.**

✓ **Database queries are used to provide a statistical overview of the dataset, like:**

- size of the file
- number of unique users
- number of nodes and ways
- number of chosen type of nodes, like cafes, shops etc.

Additional statistics not in the list above are computed. For SQL submissions some queries make use of more than one table.

✓ **The submission document includes the database queries and statistics from above.**



Other ideas about the dataset

✓ **Submission document includes one or more additional suggestions for improving and analyzing the data. The suggestions are backed up by at least one investigative query.**

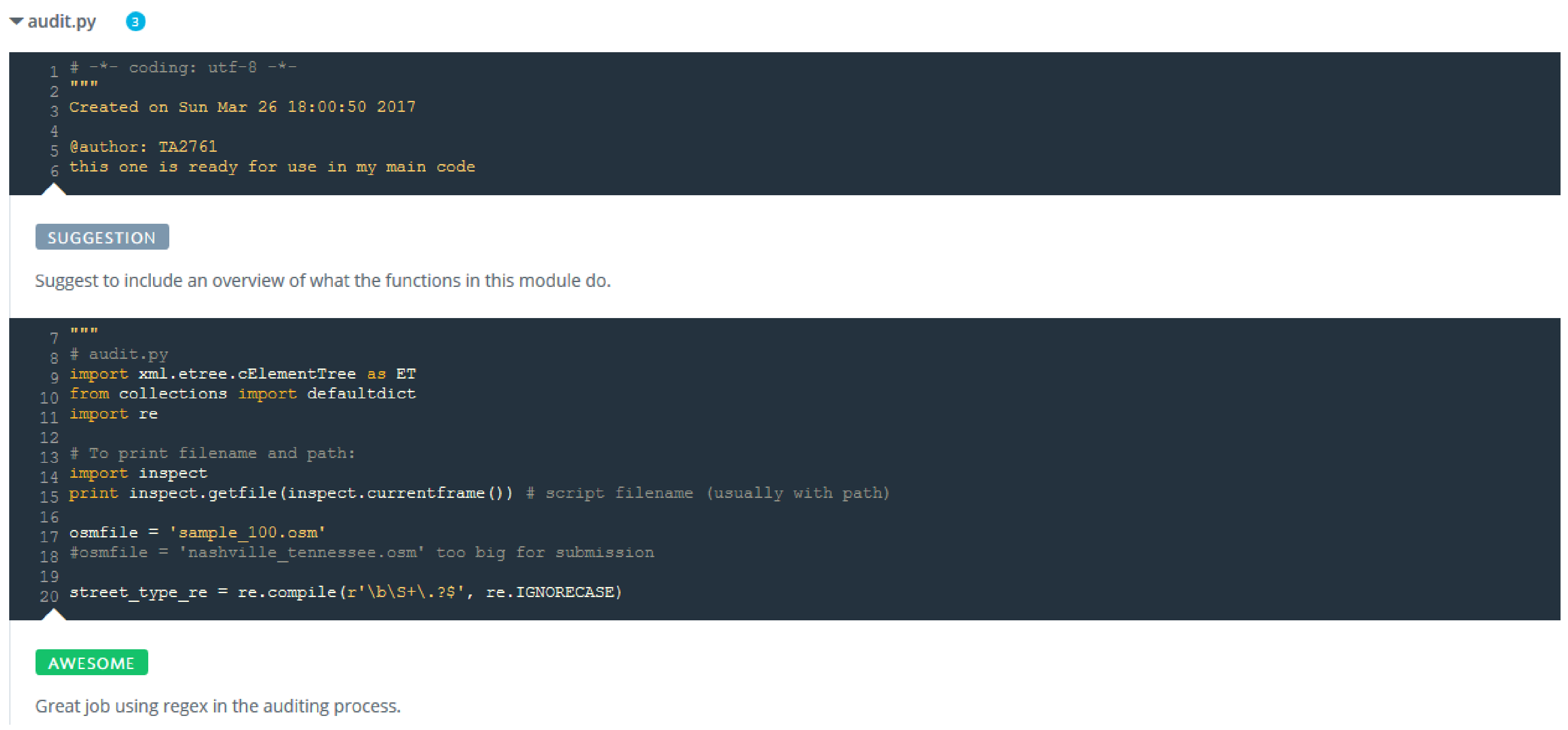
Suggestion to improve the data is backed up by an investigative query as required in this specification, good work.

✓ **Submission document includes thoughtful discussion about the benefits as well as some anticipated problems in implementing the improvement.**

Benefit and potential problem in implementing the improvement are included in the write-up.

Thoroughness and Succinctness of Submission

✓ **Submission document is long enough to thoroughly answer the questions asked without giving unnecessary detail. A good general guideline is that the question responses should take about 3-6 pages.**



audit.py 3

```
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 26 18:00:50 2017

@author: TA2761
this one is ready for use in my main code
```

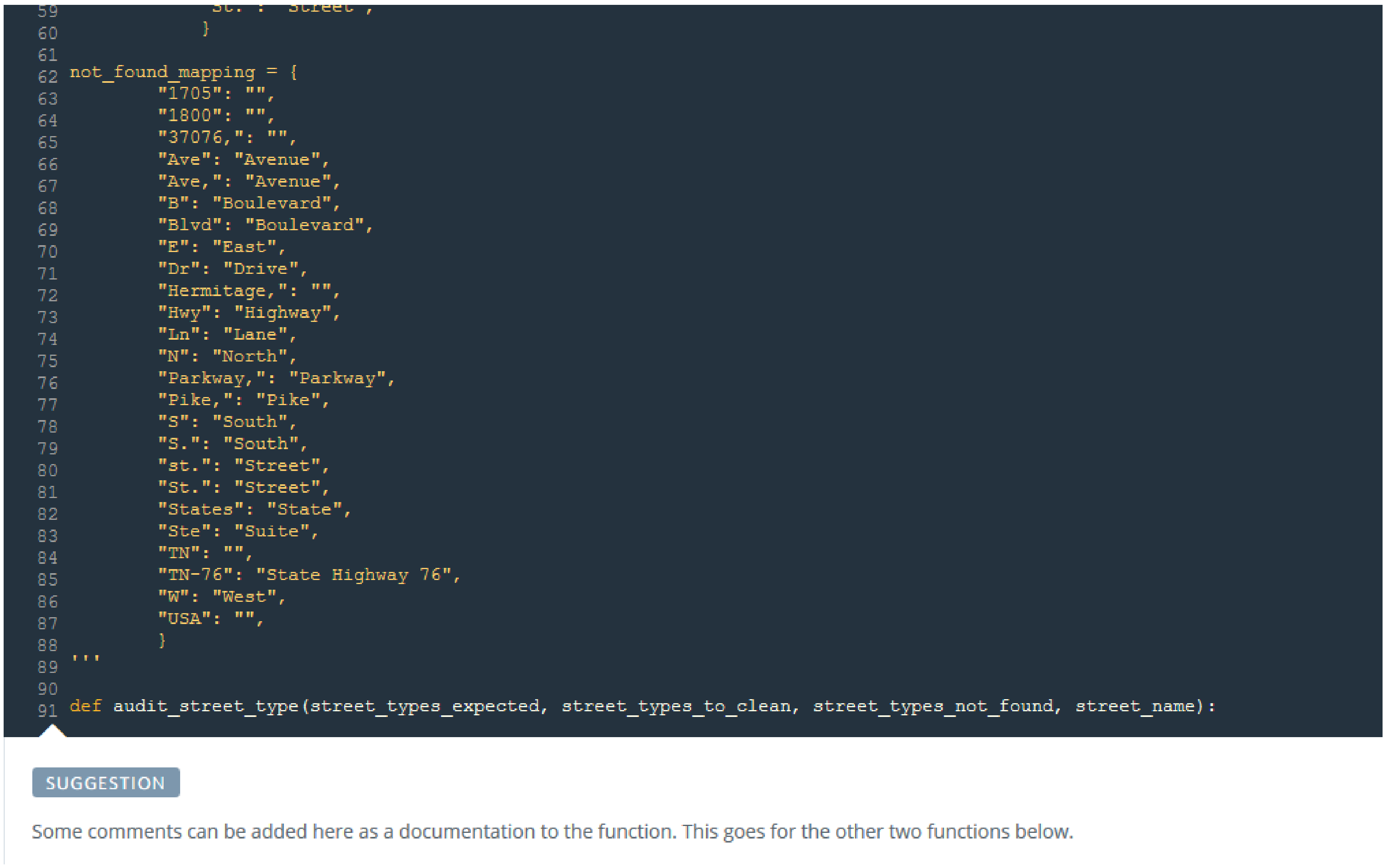
SUGGESTION

Suggest to include an overview of what the functions in this module do.

```
"""
# audit.py
import xml.etree.cElementTree as ET
from collections import defaultdict
import re

# To print filename and path:
import inspect
print inspect.getfile(inspect.currentframe()) # script filename (usually with path)

osmfile = 'sample_100.osm'
#osmfile = 'nashville_tennessee.osm' too big for submission

street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
```

AWESOME

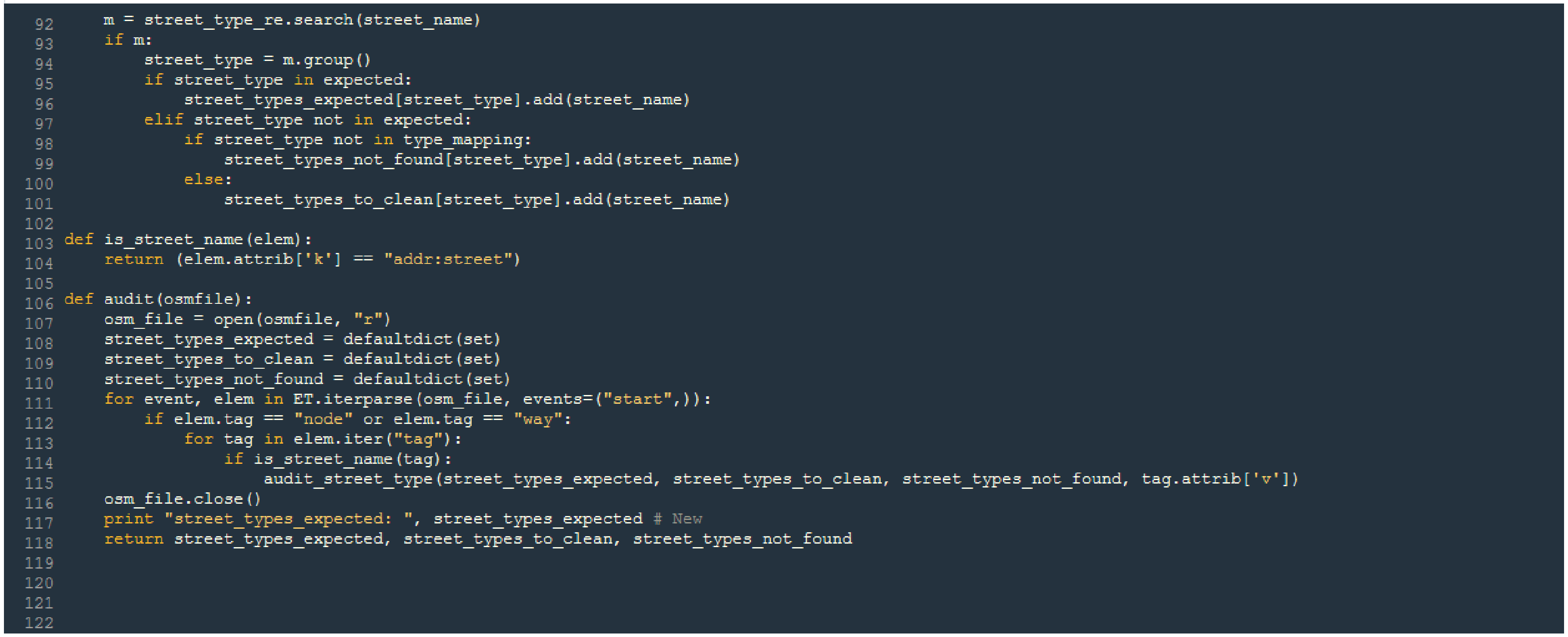
Great job using regex in the auditing process.



```
             "St.": "Street",
            }

not_found_mapping = {
        "1705": "",
        "1800": "",
        "37076,": "",
        "Ave": "Avenue",
        "Ave,": "Avenue",
        "B": "Boulevard",
        "Blvd": "Boulevard",
        "E": "East",
        "Dr": "Drive",
        "Hermitage,": "",
        "Hwy": "Highway",
        "Ln": "Lane",
        "N": "North",
        "Parkway,": "Parkway",
        "Pike,": "Pike",
        "S": "South",
        "S.": "South",
        "st.": "Street",
        "St.": "Street",
        "States": "State",
        "Ste": "Suite",
        "TN": "",
        "TN-76": "State Highway 76",
        "W": "West",
        "USA": "",
        }
'''

def audit_street_type(street_types_expected, street_types_to_clean, street_types_not_found, street_name):
```

SUGGESTION

Some comments can be added here as a documentation to the function. This goes for the other two functions below.



```python
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type in expected:
            street_types_expected[street_type].add(street_name)
        elif street_type not in expected:
            if street_type not in type_mapping:
                street_types_not_found[street_type].add(street_name)
            else:
                street_types_to_clean[street_type].add(street_name)

def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

def audit(osmfile):
    osm_file = open(osmfile, "r")
    street_types_expected = defaultdict(set)
    street_types_to_clean = defaultdict(set)
    street_types_not_found = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):
        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
                    audit_street_type(street_types_expected, street_types_to_clean, street_types_not_found, tag.attrib['v'])
    osm_file.close()
    print "street_types_expected: ", street_types_expected # New
    return street_types_expected, street_types_to_clean, street_types_not_found
```