# MagicBox Handoff

Handoff by Stanford Code the Change of Magicbox App and Mobility Pipeline to UNICEF

2019-05-30

# Magicbox App

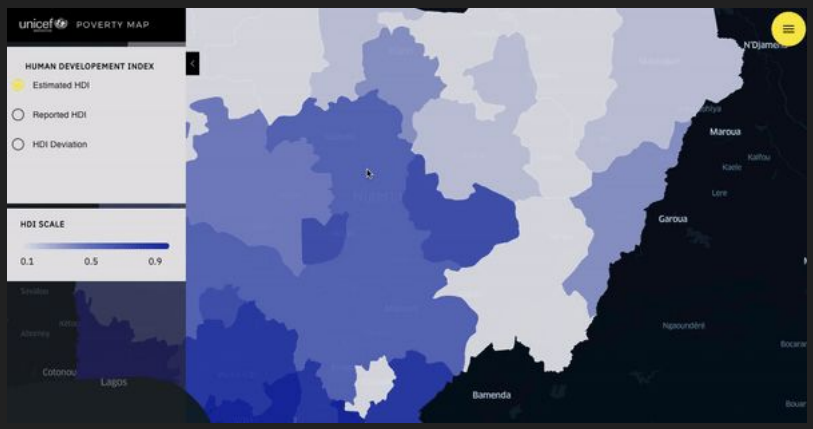
https://github.com/codethechange/magicbox-app

# Purpose

Change the map colors when clicking on provinces

Generate layer information on mobility for province clicked

# What we have


unicef
POVERTY MAP
HUMAN DEVELOPEMENT INDEX
Estimated HDI
Reported HDI
HDI Deviation
HDI SCALE
0.1
0.5
0.9
Maroua
Garoua
Bamenda
Cotonou
Lagos

# Where to go from here

Update the rest of the data store when province clicked

Condition click on specific province selected

Generate layers dynamically from CSV files

# Suggestions

For UNICEF

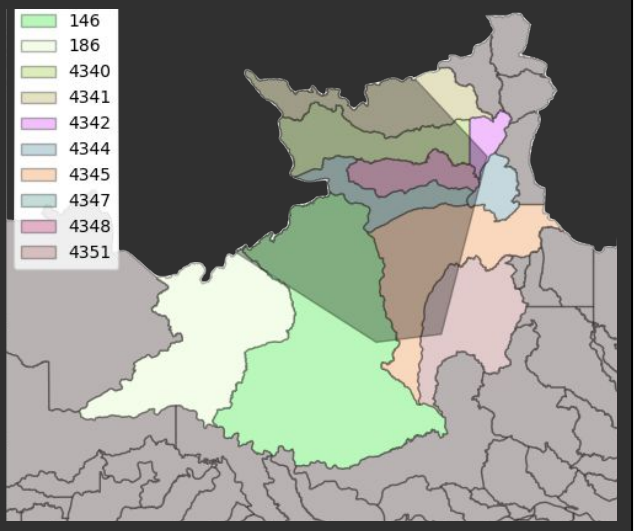
- More documentation and comments on code

For us

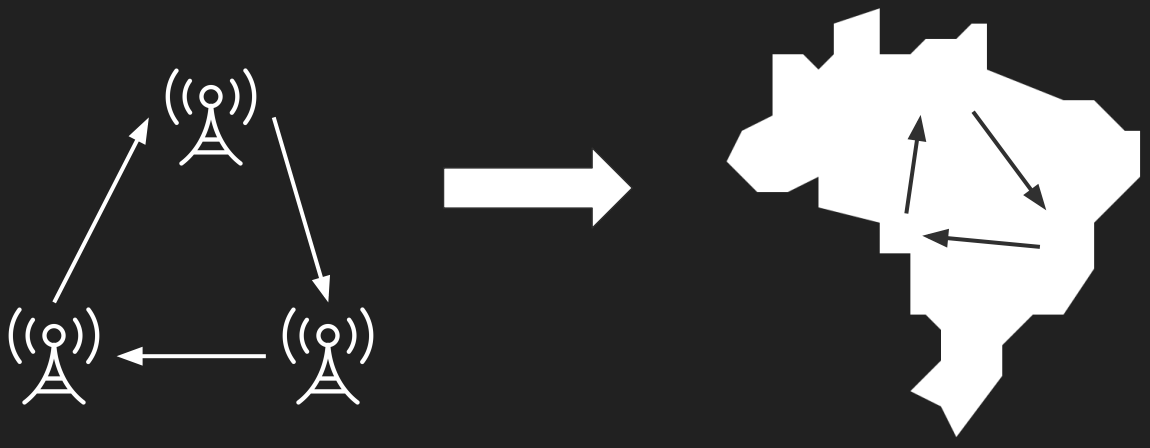
- Better understanding of Redux

# Mobility Pipeline

github.com/codethechange/mobility_pipeline



## Purpose



tower cell by Maxim Kulikov from the Noun Project, Brazil by Hea Poh Lin from the Noun Project

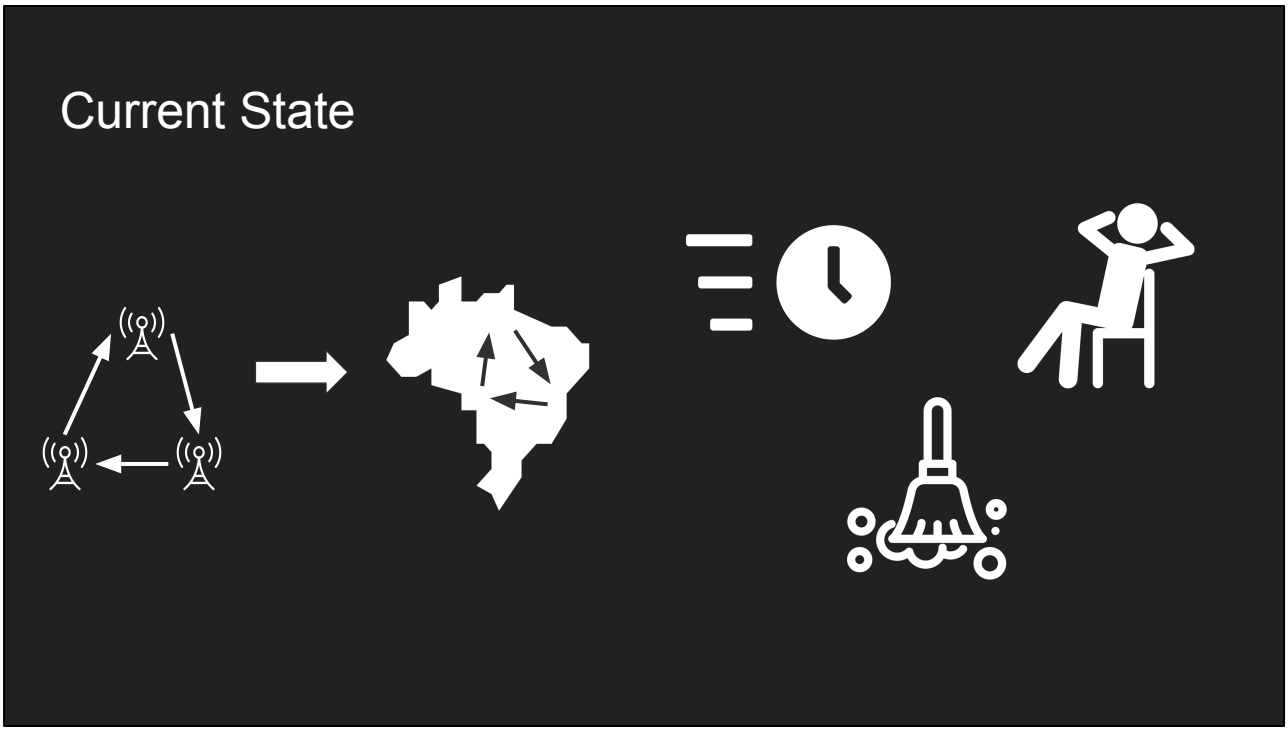
You can get data on how cell phones move between cell tower coverage areas. We can assume that this represents people moving between those areas, but we would like to know how people are moving between administrative regions like provinces within a country. In other words, we want to translate the data into the language of political regions.

The overall strategy we use to do this is detailed here in Mike's article: https://medium.com/@mikefabrikant/cell-towers-chiefdoms-and-anonymized-call-detail-records-a-guide-to-creating-a-mobility-matrix-d2d5c1bafb68

## Goals

fast by Andrejs Kirma from the Noun Project, Clean by Rudez Studio from the Noun Project, convenient by John Francis T from the Noun Project

We wanted to make sure our code was reasonably fast, even on large countries like Brazil. We also tried to keep our code clean and clear so that it would be easy to maintain. Finally, we wanted to keep the user experience as simple and easy as possible.




Right now, our code successfully transforms the mobility data. It runs reasonably quickly (2 mins for Brazil level 2) because of our reliance on matrix operations and a geospatial data structure called an RTree. Our code is, I think, pretty clean and readable, and we have written comprehensive tests and documentation to facilitate easy maintenance. The user experience is also simple--just run the script!

# How Our Code Works

# Overview of Mobility Pipeline

- **Function:** Creates the tower-admin matrix, tower-tower matrix, and admin-tower matrix, and multiplies them together (in the respective order)
- **Input:** JSON Shape Files for Administrative Region (Brazil), Voronoi Shape Files, Mobility Data (provided by UNICEF)
- **Output:** The admin-admin matrix, depicting movement of people from one admin region to another, in a `.csv` file

# Computing the Tower-to-Tower Matrix

| From | To | # ppl |
|---|---|---|
| 0 | 0 | 105 |
| 0 | 1 | 53 |
| 1 | 1 | 76 |

| From | To | # ppl |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

| From | To | # ppl |
|---|---|---|
| 0 | 0 | 105 |
| 0 | 1 | 53 |
| 1 | 0 | 0 |
| 1 | 1 | 76 |

| | From | | |
|---|---|---|---|
| To | | | |
| | | 105 | 0 |
| | | 53 | 76 |

We get mobility data in this format, where each row is for a unique pair of towers. The rows are ordered in from-major order, so we first see all the pairs with a from tower 0, and then those with from tower 1. All tower numbers are in strictly increasing order. You can see how we might be able to re-shape the right-most row into the tower-to-tower matrix on the far right, but there's a problem because any rows with a number of people equal to 0 are missing.

# Computing the Tower-to-Tower Matrix

| From | To | # ppl |
|---|---|---|
| 0 | 0 | 105 |
| 0 | 1 | 53 |
| 1 | 1 | 76 |

| From | To | # ppl |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

| From | To | # ppl |
|---|---|---|
| 0 | 0 | 105 |
| 0 | 1 | 53 |
| 1 | 0 | 0 |
| 1 | 1 | 76 |

| | From | | |
|---|---|---|---|
| To | | | |
| | | 105 | 0 |
| | | 53 | 76 |

To fix this problem, we create a matrix with every row, but no mobility data. We can generate this very efficiently with Pandas.

# Computing the Tower-to-Tower Matrix

| From | To | # ppl |
|---|---|---|
| 0 | 0 | 105 |
| 0 | 1 | 53 |
| 1 | 1 | 76 |

| From | To | # ppl |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

| From | To | # ppl |
|---|---|---|
| 0 | 0 | 105 |
| 0 | 1 | 53 |
| 1 | 0 | 0 |
| 1 | 1 | 76 |

| | From | | |
|---|---|---|---|
| To | | | |
| | | 105 | 0 |
| | | 53 | 76 |

Then, we can efficiently merge these two matrices using pandas. After filling in the empty cells with zeroes, we get a matrix with all our mobility data and no missing rows.

# Computing the Tower-to-Tower Matrix

| From | To | # ppl |
|---|---|---|
| 0 | 0 | 105 |
| 0 | 1 | 53 |
| 1 | 1 | 76 |

| From | To | # ppl |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

| From | To | # ppl |
|---|---|---|
| 0 | 0 | 105 |
| 0 | 1 | 53 |
| 1 | 0 | 0 |
| 1 | 1 | 76 |

| | From | | |
|---|---|---|---|
| To | | 0 | 1 |
| | 0 | 105 | 0 |
| | 1 | 53 | 76 |

Finally, we can reshape the number of people column into a square matrix that shows all our mobility data. This is the tower-to-tower matrix. Notice how the “to” and “from” tower indices are just the row and column indices of each cell.

# Tower-Admin Matrix

*RTree Algorithm*: determines whether two rectangles overlap

**Tower-Admin Matrix:**

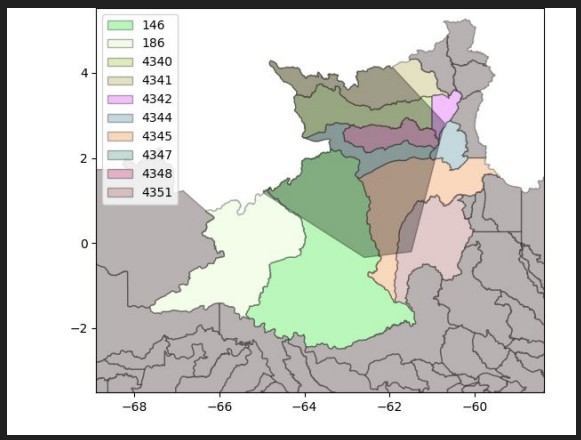
(1) Load RTree with list of admin region polygons and their bounding boxes
(2) Input rectangular bounding box of Voronoi cell into RTree
(3) Output a list of polygons whose bounding boxes overlap with that of the Voronoi
(4) (Hash Map) …
(5) Compute the percentage of overlap between each polygon and Voronoi cell
(6) Repeat (2)-(5) for all Voronoi cells

## Recovering Polygon Indices

```
dict = {}  # dict = {polygon_key: index}

for every admin MultiPolygon mpol with index i:

    polygon_key = Tuple(

        [ Tuple(pol.exterior.coors) for pol in mpol ]

    )

    dict[polygonKey] = i
```

One complication is that the RTree returns the polygon itself, so we need a way to get its index. As an aside, we were actually considering using another library that would give us the indices, but that required installing non-Python dependencies, and we wanted to keep things simple. Anyway, we created a dictionary from polygons back to indices. Polygons aren't hashable, though, so we couldn't use them as keys. Instead, we took the coordinates of the exterior edge of the polygon and made them into a tuple. We did this for every polygon in the MultiPolygon, and but all of those tuples into another tuple. Tuples are hashable, so we use them as the key.

# Tower-Admin Matrices




| Region | Percent Overlap |
|---|---|
| 146 | 0.3438314729373602 |
| 186 | 0.005168094316897268 |
| 4340 | 0.9791906016675118 |
| 4341 | 0.7525484088364074 |
| 4342 | 0.24378993279982075 |
| 4344 | 0.07680992472243754 |
| 4345 | 0.6068501217581207 |
| 4347 | 1.0 |
| 4348 | 1.0 |
| 4351 | 0.14599840959396204 |

# Admin-Tower Matrix

Constructed using a similar algorithm as the Tower-Admin matrix (namely, the RTree), except the input/outputs are “flipped”:

- Load RTree with a list of Voronoi cells and their bounding boxes
- Input the bounding box of an admin region and find the percentage overlap with each Voronoi cell

# Code Organization

## Code Structure

```
mobility_pipeline/
    data/actual/
        br_admin2.json, ...
    docs/
        *.rst documentation files for Sphinx
        _build/
            *.html generated documentation files from Sphinx
    mobility_pipeline/
        lib/
            make_matrix.py, ...
        data_interface.py
        mobility_matrix.py, ...
    tests/src/
        test_make_matrix.py, ...
```

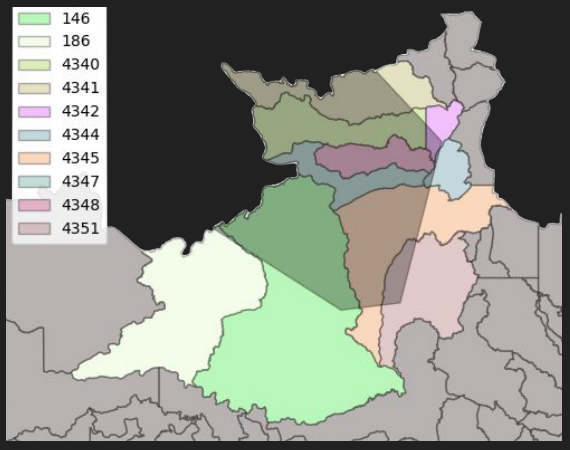
In our repository, you can see at the top here our data files and documentation. The orange files here don't have any code. Then we have library files like make_matrix. The code in these files doesn't care about the format of the data files, and they mostly have unit tests. All the code dealing with the format of the data files lives in data_interface. Then we have the script files, in green, which load the data using the data interface and perform operations using the library files. The idea here is that the library files could be re-used in other contexts, while the rest of the code is tightly linked to this particular application. Finally we have the test files, with one test file for each library file.

Everything Else Included

# Visualization Tools

Note: For `matplotlib` to work on macOS, you have to create the virtual environment with `python -m venv` instead of `virtualenv`.

`python plot_voronoi.py`




`python visualize_overlaps.py`

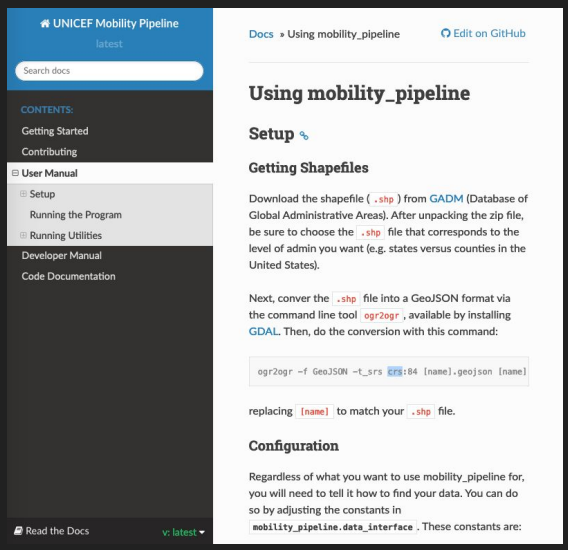
We created some visualization tools along the way to sanity-check our work, and we thought you might find them useful for seeing what the data really mean. Plot_voronoi shows the Voronoi tessellation with the towers overlayed.

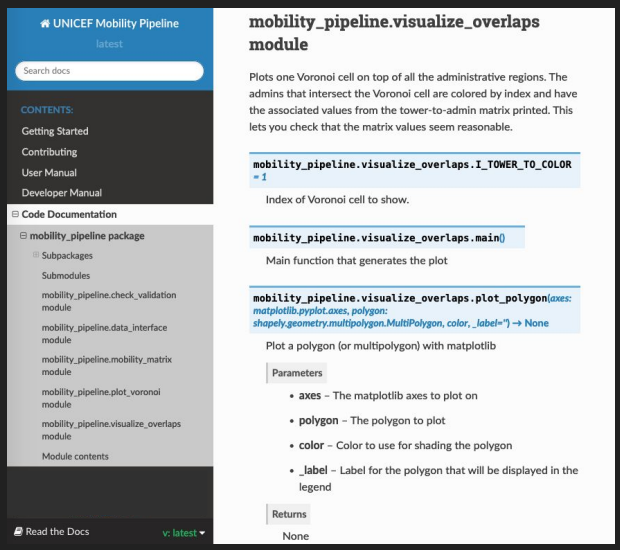
visualize_overlaps shows part of the computation behind the tower-to-admin matrix. It shows which admin regions might overlap with a Voronoi tessellation (the gray shaded region), and it prints out the matrix values computed for each admin region. The number labels are indices, which are also printed alongside the matrix values.

Finally, we created a script to validate data files. This script is slow, but it runs a variety of check to ensure that the data files look reasonable.

# Documentation with Sphinx

https://unicef-mobility-pipeline.readthedocs.io/




We have detailed code comments for all functions, modules, and constants. We also wrote introductory documentation for both users and developers. These forms of documentation are combined by Sphinx, which can be built into local HTML. It is also hosted online by readthedocs.

# Unit Tests with PyTest

./test.sh

Test by rajakumara from Noun Project

```
$ python -m pytest                                                                                                    3.99s
============================= test session starts ==============================
platform linux -- Python 3.7.1, pytest-4.4.0, py-1.8.0, pluggy-0.9.0
hypothesis profile 'default' -> database=DirectoryBasedExampleDatabase('/home/travis/build/codethechange/mobility_pipeline
/.hypothesis/examples')
rootdir: /home/travis/build/codethechange/mobility_pipeline, inifile: pytest.ini
plugins: cov-2.6.1, hypothesis-4.16.0
collected 23 items

tests/src/test_make_matrix.py ......                                    [ 26%]
tests/src/test_overlap.py                                               [ 30%]
tests/src/test_validate.py ..............                              [ 91%]
tests/src/test_voronoi.py ..                                            [100%]

---------- coverage: platform linux, python 3.7.1-final-0 ----------
Name                                                Stmts   Miss  Cover
-----------------------------------------------------------------------
mobility_pipeline/lib/__init__.py                       0      0   100%
mobility_pipeline/lib/make_matrix.py                   36      0   100%
mobility_pipeline/lib/overlap.py                        5      0   100%
mobility_pipeline/lib/validate.py                      97     45    54%
mobility_pipeline/lib/voronoi.py                       19      1    95%
-----------------------------------------------------------------------
TOTAL                                                 157     46    71%


========================== 23 passed in 1.54 seconds ===========================
The command "python -m pytest" exited with 0.

$ find mobility_pipeline -name "*.py" | xargs pylint                                                                 12.79s

--------------------------------------
Your code has been rated at 10.00/10

The command "find mobility_pipeline -name "*.py" | xargs pylint" exited with 0.

$ find tests/src -name "*.py" | xargs pylint                                                                          10.73s

--------------------------------------
Your code has been rated at 10.00/10

The command "find tests/src -name "*.py" | xargs pylint" exited with 0.

$ python -m mypy mobility_pipeline                                                                                    8.98s
The command "python -m mypy mobility_pipeline" exited with 0.

$ liccheck -s strategy.ini -r requirements.txt                                                                        0.85s
gathering licenses...
64 packages and dependencies.
check authorized packages...
64 packages.
The command "liccheck -s strategy.ini -r requirements.txt" exited with 0.

$ bash <(curl -s https://codecov.io/bash)                                                          after_success      1.98s

Done. Your build exited with 0.
```

Excluding the data validation code, data interface, and scripts, we have unit tests for almost all of our codebase (98%). We don’t comprehensively test edge cases though. Executing the test.sh script runs the unit tests, checks style with a linter, uses type annotations to type-check our code, and checks license compatibility. These tests are run by Travis-CI, which gives output like that shown on the right.