Generalizing CDR toolbox for new data types

A study on the Python toolbox bandicoot

Benjamin Holm Glaas, s123862
Department of Applied Mathematics and Computer Science
Technical University of Denmark
Lyngby, Denmark
s123862@student.dtu.dk

ABSTRACT

The research field within computational social science is rapidly expanding. This calls for better computational research tools, that provide fast analysis of large amounts of social data. Currently, a number of different tools exist, among which the most promising is the Python toolbox *bandicoot*. *bandicoot* relies on Call Detail Records (CDR) from mobile phone users to compute a wide range of so-called indicators. This project provides a generalization of *bandicoot*, allowing researchers to input any type of data into the toolbox, rather than just Call Detail Records. The power of the generalized framework is being demonstrated using data from social encounters measured from Bluetooth signals between smart phones, as well as time-series data of screen on/off events for a large set of users. Furthermore, software validation is provided using Python test functions to show that the implemented generalizations are robust for any type of time series data.

Keywords

Python; bandicoot; CDR data; smartphone data

1. INTRODUCTION

Some years back, however not many, all we used our cell phones for was to keep in contact. We did this by calling or sending text messages. Information about calls and text messages are stored in metadata records called Call Detail Records (CDR). As the development of cell phones rapidly increased, cell phones (or smart phones) yielded access to the internet and therefore providing a vast amount of new types of data. As a simple example, these types of data could hold information on where the user has been, that is GPS locations. For now, a general framework for processing behavioral smart phone data or other kinds of social data does not exist. A framework for processing CDR metadata does however exist, which is called *bandicoot*[3]. *bandicoot* provides what is called *indicators* based on CDR data, which gives a simple and rich overview of the behaviors of a phone user.

What is intended in this project is to generalize *bandicoot* such that these indicators are available for any given data type. It should therefore be possible to provide *bandicoot* with other data types such as proximity events or other behavioral data.
In the following section, the implementation process will be outlined, where it will be discussed in detail how the original *bandicoot* works and how the modified *bandicoot* differs. The implementation section is followed by a section discussing the results of providing new data into the modified version of the toolbox, including a subsection on validation. Finally, a section is provided discussing further work and a section concluding the project as a whole.

2. IMPLEMENTATION

In Section 2.1 it will be discussed how the original *bandicoot* works and in Section 2.2 it will be discussed how the modified version of *bandicoot* works.
The source code for this project can be found on Github [1].

2.1 bandicoot AS-IS

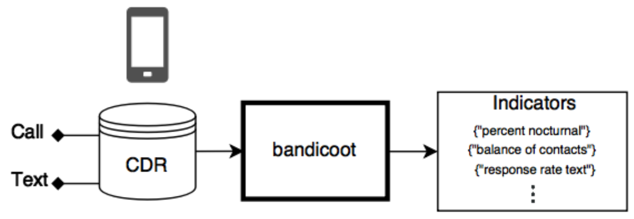


Figure 1: Schematic overview of the original bandicoot for the *individual* indicators.

bandicoot is a Python toolbox that handles Call Detail Records (CDR) metadata [3], which is data holding information about calls and text messages. In the data, attributes like "datetime" (timestamp), "correspondent id", "call duration", "direction" and "interaction" (call or text) are included, yielding information on the specific interaction. The unique thing about *bandicoot* is the *indicators*, which also can be adressed as "features", of which this project will attend the so-called *individual indicators*, called indicators from now on for simplicity. Some of these indicators include the percentage that the user is nocturnal based on the interactions, the entropy of contacts and the percentage of

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

initiated conversations. Indicators are by default grouped weekly and provides statistical measurements such as mean and standard deviation, or higher order statistics if specified. A schematic overview of the toolbox can be seen in Figure 1.

Spatial indicators are also available, if the user provides information on locations of the antennas that are used and social indicators, also addressed as network indicators. *bandicoot* can provide more than 1400 different indicators in total based on the three different types of indicators discussed. The high number of available indicators is based on the fact that the user can choose to get an extended summary of the statistics or group the records differently, that is daily, weekly, monthly or over all time. Finally, a visualization part is available for a quick overview of the phone user with the opportunity to interact with the visualization.

2.2 Generalization of bandicoot

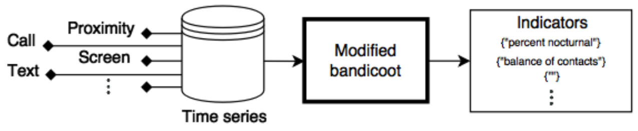


Figure 2: Schematic overview of the modified bandicoot for the *individual* indicators. It can be seen that some of the indicators are left out, if the data holds different interaction types than "call" or "text".

To modify *bandicoot*, the software was cloned from the GitHub repository [2]. From this point, the source code was first modified focusing on the import of the data, since new attributes and other interaction types are now available for import. As seen in Figure 2, the only requirement for the data set is that it contains a time series. The import should accept all of the data, except if the data contains attributes that do not make sense for the original *bandicoot* attributes. Also, if a new attribute is stored in the data, the accepted type of the attribute is the most common type, and a warning is issued for observations not having this type. The user should then attend to the data set and either clean it or separate it into multiple data sets. Next, the focus is attended on the indicator functions, which holds the uniqueness of *bandicoot*. These indicator functions should find the indicators, based on what is available in the dataset. This means, that if an attribute such as "datetime", which is a time stamp in **datetime** format, is present in the dataset, *bandicoot* now computes all indicators based on the "datetime" attribute. Functions that can't return anything returns "None", and the user has the option to filter them out. This functionality can be seen in Figure 2, where the indicator "response rate text" returned "None", as the new data set might not contain interactions of text messages, as opposed to Figure 1. In the modified version, the user also has the option to provide *bandicoot* with an indicator function that they themselves made, meaning they can customize indicator functions based on their specific data type. Along with the new indicator function, the user should also provide the name of the attributes that the function uses, such that *bandicoot* can skip the records that have faulty or missing data for the given attributes. Otherwise the records that are faulty will be included in the computation, which may yield an error or wrong results.

The user can in the modified version choose the interaction type on which to group the data. If a dataset for example contains interactions such as "physical" and "call", the user can give the keyword *interaction* equal to the grouping they want. If the user wants to group "physical" and "call" together, the keyword should be equal to a list of these, that is *interaction* = ["physical","call"], and single groupings should be separate, that is *interaction* = "physical". The user can also group separate and combined at the same time, that is *interaction* = [["physical","call"],"physical","call"].
Additionally, more user feedback has been incorporated. The user is now informed on the number of unique contacts overall and the number of contacts for the different interaction types. Furthermore, the user is also warned on the functions that return "None", since it can be relevant for the user to see what *bandicoot* couldn't quantify.
Examples of how the modified *bandicoot* works can be seen in Appendix A.

3. RESULTS

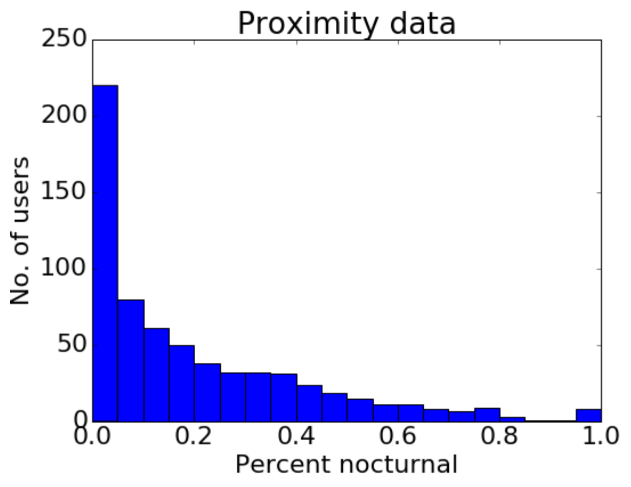
With the new software, it was possible to obtain indicators for a new and unknown data type. The new datasets that were tested included

- Data holding information on screen turned on/off
- Data holding proximity events via Bluetooth connection
- Data holding stop locations for a user

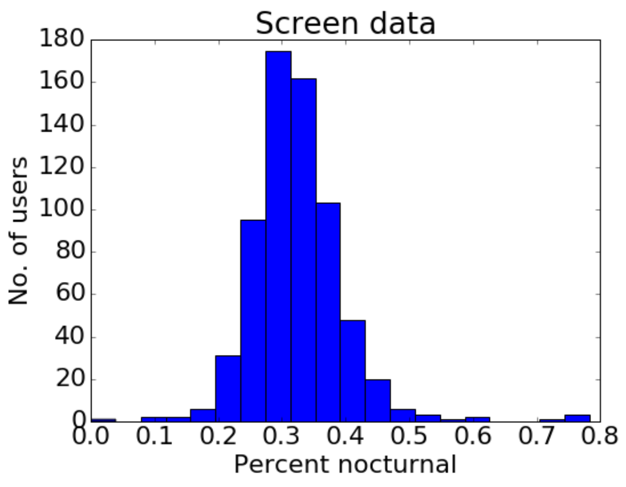
All of these different types of data are obtained from the Sensible DTU project [5], which is a large study on the behavioral patterns using the data of smart phones.
In Section 3.1, examples of the new datatypes will be processed in the new *bandicoot*, and in Section 3.2 a discussion of the validation of the indicators will be provided.

3.1 Examples with new data

To see if we can get some interesting results with the new *bandicoot*, phone data from 661 test subjects, all of which are students, from the Sensible DTU project has been used. The data which is used is within the month of February 2014, where two subsets of data has been used. The subsets that has been used are regarding the on/off switch of the test subject's screen and proximity data of the test subjects, which has been recorded via Bluetooth when two test subjects are in proximity of each other. The indicator *percent nocturnal*, which indicates how many percent of the test subject's interactions are between 7 pm. and 7 am., has been computed. The results can be seen in Figure 3. In Figure 3a, the number of test subjects tends to decrease as the percentage is increased. This makes sense, since students usually meet during the working hours, that is before 7 pm. It can however be seen, that some students only meet other students during the night, but since the data only comes from a sample of a month, this is definitely plausible. Looking at Figure 3b we can see that the distribution tends to be almost normally distributed, with a mean around 0.32. This means that generally test subjects spend more time looking at their screens in the day hours than in the night hours,



(a) Proximity data for users



(b) Screen data for users

Figure 3: The indicator *percent nocturnal* computed for 661 users for two different sub data sets from the Sensible DTU data set. The data sets are within the period of February 2014. In Figure 3a the data set consists of proximity events of users in the study and in Figure 3b the data set consists of observations of the user's screen turned on/off.

which makes sense since most people sleep at night. Therefore, the results make sense intuitively.
The two data sets have been processed separately, but could also have been incorporated into *bandicoot* as a whole, that is a file containing both proximity and screen data for each test subject. A new indicator could also have been provided. An idea for a new indicator could for example be "the number of seconds it took for a test subject to look at their screen after starting a physical meeting (conversation)". However, due to the time limit of the project this has not been done, but could make a great example of what the modified *bandicoot* is capable of.

3.2 Validation

To validate whether or not the indicators make sense for new data types, one should start out by looking at them qualitatively. This means that if a test subject e.g. has an unusual large amount of time spent looking at their screens, maybe there is a problem. Qualitative inspection of the results all yield plausible outcomes, but quantitative testing should be made as well. This could be by taking a *very* simple dataset and calculating each indicator by hand. This has been done, and test functions have been written using *nosetest* [4], testing whether the new dataset holds the same indicators as the ones computed by hand. Running the nosetest gives no failures, which concludes that the indicators are correctly computed. The code for the nose testing can be seen in Appendix B.

4. DISCUSSION

Through the configuration of *bandicoot*, it was possible to obtain the primary goal of the project, which was to let *bandicoot* process a new and unknown data type, that is different from the original CDR data type. However, some aspects of the toolbox were not configured, such as the network and visualization parts. The network part holds a lot of potential, especially if one looks at data sets with proximity events, similar to the one discussed in this project. However, it seems like the network functions should be radically changed, which therefore potentially could be another project similar to this one.
As it holds now, the visualization part does not work, but this is not of great importance, as this project was about configuring *bandicoot* such that it could be useful for researchers who would use *bandicoot* for other data types. Further work therefore includes to modify the visualization part as well and maybe include more interactions in the visualization.
The spatial indicators have not been attended either, since these at the moment handle information on the antennas that the user has been connected to. It could however be interesting to modify this part of the toolbox as well to see if raw spatial data could be processed through *bandicoot*.
Even though the testing worked for a very simple data set with a few observations, it could be argued that more test functions should be written for a more complicated or larger data set. This would further state the validity of the indicators, however time has not permitted to create more test functions, which therefore has been excluded in this project.

5. CONCLUSIONS

The *bandicoot* toolbox yields a powerful processing of CDR data, but it cannot be used for any other types of data. In this project it was managed to modify the toolbox such that new data types can be processed in the toolbox, yielding indicators for any other time series data type. The modified *bandicoot* therefore gives the user the ability to find many new interesting results, and to customize their own indicators to their specific data set. The primary goal was therefore achieved, with the future work including modification of the visualization part and to modify the network part of the toolbox.

6. ACKNOWLEDGMENTS

A great thank you to Ulf Aslak, who has helped a lot during the project, both on the implementation part and feedback on the report, as well as always being able to answer any questions on the matters. Also thank you to Sune Lehmann who has helped setting up this project and laying the base for a masters thesis.

7. REFERENCES

[1] B. H. GLAAS. Modified bandicoot source code. https://github.com/benjaminglaas/bandicoot.

[2] Y.-A. D. MONTJOYE. bandicoot source code. https://github.com/yvesalexandre/bandicoot.

[3] Y.-A. D. MONTJOYE, L. ROCHER, and A. PENTLAND. bandicoot: a python toolbox for mobile phone metadata. *Journal of Machine Learning Research*, 17(175):1–5, 2016.

[4] J. PELLERIN. nosetest documentation. http://nose.readthedocs.io/en/latest/.

[5] A. Stopczynski, V. Sekara, P. Sapiezynski, A. Cuttone, M. M. Madsen, J. E. Larsen, and S. Lehmann. Measuring large-scale social networks with high resolution. *PLoS ONE*, 9(4), 2014.

APPENDIX

A. EXAMPLES

Examples of how the modified *bandicoot* works can be seen on the following website https://benjaminglaas.github.io/examples/. The examples are also given on the following pages.

Example notebook

December 4, 2016

0.1 Examples on how to use the modified version of *bandicoot*

This notebook is used for showing examples of how the modified version of *bandicoot* works.

```
In [1]: import bc_dev as bc #Import the modified bandicoot
        import pandas as pd #For showing the header of the files
```

```
In [2]: path = '../data/user0/'
```

First we will take a look at data regarding whether the user turned their screen on/off

```
In [3]: data = pd.read_csv(path + "screen.csv")
```

```
In [4]: data[:10]
```

```
Out[4]:                datetime  duration interaction
        0  2014-01-01 01:16:52         6      screen
        1  2014-01-01 01:21:16        60      screen
        2  2014-01-01 01:30:59        41      screen
        3  2014-01-01 01:31:42       110      screen
        4  2014-01-01 01:36:29        97      screen
        5  2014-01-01 02:31:23        91      screen
        6  2014-01-01 02:35:46         2      screen
        7  2014-01-01 02:40:45        31      screen
        8  2014-01-01 02:57:58        31      screen
        9  2014-01-01 03:04:56         0      screen
```

We will now import the data into *bandicoot*

```
In [7]: U_screen = bc.io.read_csv('screen',path)
```

```
WARNING:root:100.00% of the records are missing a location.
         No antennas file was given and records are using antennas for position.
Warning: 100.00% of the records are missing a location.

No antennas file was given and records are using antennas for position.
WARNING:root:19 record(s) are duplicated.
```

```
[x] 41602 records from 2014-01-01 01:16:52 to 2014-12-30 23:52:34
[ ] No contacts for screen
[ ] No attribute stored
[ ] No antenna stored
[ ] No recharges
[ ] No home
[ ] No texts
[ ] No calls
[ ] No network
[x] Has screen


Warning: 19 record(s) are duplicated.
```

We will now compute the indicators for the data set using the **all** function. Using the keyword *show_all* = ***False*** yields all the indicators that are not returning **None**. If the user gives the keyword *warnings* = **True**, and there is at least one function that returns **None**, then the user will be notified of which functions that returned **None**, as seen below.

```
In [8]: bc.utils.all(U_screen,show_all=False,warnings=True)

WARNING:root:16 functions that only returned 'None'
number_of_contacts
percent_initiated_conversations
percent_initiated_interactions
response_delay_text
response_rate_text
entropy_of_contacts
balance_of_contacts
interactions_per_contact
percent_pareto_interactions
percent_pareto_durations
number_of_interaction_in
number_of_interaction_out
entropy_of_antennas
percent_at_home
radius_of_gyration
churn_rate

Warning: 16 functions that only returned 'None'number_of_contactspercent_initiated_

Out[8]: {
            "name": "screen",
            "reporting": {
                "antennas_path": None,
                "attributes_path": None,
                "recharges_path": None,
                "version": "0.5.3",
                "code_signature": "900718f0bed6a9c5dac2806eab9cccfc91d3cec9",
        "groupby": "week",
        "split_week": false,
        "split_day": false,
        "start_time": "2014-01-01 01:16:52",
        "end_time": "2014-12-30 23:52:34",
        "night_start": "19:00:00",
        "night_end": "07:00:00",
        "weekend": [
            6,
            7
        ],
        "number_of_records": 41602,
        "number_of_antennas": 0,
        "number_of_recharges": 0,
        "bins": 53,
        "bins_with_data": 53,
        "bins_without_data": 0,
        "has_call": false,
        "has_text": false,
        "has_home": false,
        "has_recharges": false,
        "has_attributes": false,
        "has_network": false,
        "percent_records_missing_location": 1.0,
        "antennas_missing_locations": 0,
        "percent_outofnetwork_calls": 0,
        "percent_outofnetwork_texts": 0,
        "percent_outofnetwork_contacts": 0,
        "percent_outofnetwork_durations": 0,
        "ignored_records": {
            "position": 0,
            "interaction": 0,
            "all": 0,
            "datetime": 0,
            "duration": 0,
            "direction": 0
        }
    },
    "active_days": {
        "allweek": {
            "allday": {
                "screen": {
                    "mean": 6.867924528301887,
                    "std": 0.7278226989428646
                }
            }
        }
    },
"duration": {
    "allweek": {
        "allday": {
            "screen": {
                "mean": {
                    "mean": 103.9061871276594,
                    "std": 23.243153464972842
                },
                "std": {
                    "mean": 308.8849635253289,
                    "std": 76.33759783133397
                }
            }
        }
    }
},
"percent_nocturnal": {
    "allweek": {
        "allday": {
            "screen": {
                "mean": 0.5433778644663688,
                "std": 0.15430907301059496
            }
        }
    }
},
"interevent_time": {
    "allweek": {
        "allday": {
            "screen": {
                "mean": {
                    "mean": 772.8691010960384,
                    "std": 118.89793635094766
                },
                "std": {
                    "mean": 2110.73734966351,
                    "std": 383.90293304277935
                }
            }
        }
    }
},
"number_of_interactions": {
    "allweek": {
        "allday": {
            "screen": {
                "mean": 784.9433962264151,
                "std": 153.04819996652674
```

```
                    }
                }
            }
        },
        "number_of_antennas": {
            "allweek": {
                "allday": {
                    "mean": 1.0,
                    "std": 0.0
                }
            }
        },
        "frequent_antennas": {
            "allweek": {
                "allday": {
                    "mean": 1.0,
                    "std": 0.0
                }
            }
        },
        "none_functions": [
            "number_of_contacts",
            "percent_initiated_conversations",
            "percent_initiated_interactions",
            "response_delay_text",
            "response_rate_text",
            "entropy_of_contacts",
            "balance_of_contacts",
            "interactions_per_contact",
            "percent_pareto_interactions",
            "percent_pareto_durations",
            "number_of_interaction_in",
            "number_of_interaction_out",
            "entropy_of_antennas",
            "percent_at_home",
            "radius_of_gyration",
            "churn_rate"
        ]
    }
```

A function like **entropy_of_contacts** has returned **None** since it depends on the attribute *correspondent_id*, which is not present in the data set.
We can now take a look at a data set for proximity events, that is a meeting is recorded when test subjects are near each other, recorded via Bluetooth.

```
In [16]: data2 = pd.read_csv(path+"proximity_original.csv")
```

```
In [17]: data2[:10]
```

```
Out[17]:                datetime interaction  correspondent_id
         0  2014-01-06 08:53:32    physical               371
         1  2014-01-06 09:38:41    physical               285
         2  2014-01-06 09:53:32    physical               371
         3  2014-01-06 09:54:52    physical               371
         4  2014-01-06 10:03:33    physical               371
         5  2014-01-06 10:13:38    physical               371
         6  2014-01-06 10:38:38    physical               285
         7  2014-01-06 10:43:39    physical               285
         8  2014-01-06 10:59:05    physical               285
         9  2014-01-06 11:04:15    physical               285
```

Here we have a *correspondent_id* attribute, which means that some of the functions that returned **None** for the other data sets will not return **None** for this data set, for example the **entropy_of_contacts** function.

```
In [19]: U_proximity = bc.io.read_csv("proximity_original",path)

WARNING:root:100.00% of the records are missing a location.
         No antennas file was given and records are using antennas for position.
Warning: 100.00% of the records are missing a location.

No antennas file was given and records are using antennas for position.
WARNING:root:4 record(s) are duplicated.

[x] 13621 records from 2014-01-06 08:53:32 to 2014-06-20 19:43:22
[x] 222 contacts physical
[ ] No attribute stored
[ ] No antenna stored
[ ] No recharges
[ ] No home
[ ] No texts
[ ] No calls
[ ] No network
[x] Has physical


Warning: 4 record(s) are duplicated.

In [20]: bc.utils.all(U_proximity,show_all=False)

Out[20]: {
             "name": "proximity_original",
             "reporting": {
                 "antennas_path": None,
                 "attributes_path": None,
                 "recharges_path": None,
                 "version": "0.5.3",
                 "code_signature": "900718f0bed6a9c5dac2806eab9cccfc91d3cec9",
        "groupby": "week",
        "split_week": false,
        "split_day": false,
        "start_time": "2014-01-06 08:53:32",
        "end_time": "2014-06-20 19:43:22",
        "night_start": "19:00:00",
        "night_end": "07:00:00",
        "weekend": [
            6,
            7
        ],
        "number_of_records": 13621,
        "number_of_antennas": 0,
        "number_of_recharges": 0,
        "bins": 24,
        "bins_with_data": 21,
        "bins_without_data": 3,
        "has_call": false,
        "has_text": false,
        "has_home": false,
        "has_recharges": false,
        "has_attributes": false,
        "has_network": false,
        "percent_records_missing_location": 1.0,
        "antennas_missing_locations": 0,
        "percent_outofnetwork_calls": 0,
        "percent_outofnetwork_texts": 0,
        "percent_outofnetwork_contacts": 0,
        "percent_outofnetwork_durations": 0,
        "ignored_records": {
            "position": 0,
            "interaction": 0,
            "all": 0,
            "direction": 0,
            "datetime": 0,
            "correspondent_id": 0
        }
    },
    "active_days": {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": 5.857142857142857,
                    "std": 1.551825784457174
                }
            }
        }
    },
"number_of_contacts": {
    "allweek": {
        "allday": {
            "physical": {
                "mean": 25.142857142857142,
                "std": 13.694926812807848
            }
        }
    }
},
"percent_nocturnal": {
    "allweek": {
        "allday": {
            "physical": {
                "mean": 0.3065686567099935,
                "std": 0.2612915508448955
            }
        }
    }
},
"entropy_of_contacts": {
    "allweek": {
        "allday": {
            "physical": {
                "mean": 1.7251441042734903,
                "std": 0.8241087308035341
            }
        }
    }
},
"interactions_per_contact": {
    "allweek": {
        "allday": {
            "physical": {
                "mean": {
                    "mean": 32.6769004336772,
                    "std": 24.92742551303872
                },
                "std": {
                    "mean": 74.48857191537824,
                    "std": 54.55078509944941
                }
            }
        }
    }
},
"interevent_time": {
    "allweek": {
```

```
                "allday": {
                    "physical": {
                        "mean": {
                            "mean": 853.9451543850575,
                            "std": 351.32139431039144
                        },
                        "std": {
                            "mean": 6471.311892414895,
                            "std": 3353.9400850469583
                        }
                    }
                }
            }
        },
        "percent_pareto_interactions": {
            "allweek": {
                "allday": {
                    "physical": {
                        "mean": 0.05724867363863627 6,
                        "std": 0.21100374717808576
                    }
                }
            }
        },
        "number_of_interactions": {
            "allweek": {
                "allday": {
                    "physical": {
                        "mean": 648.6190476190476,
                        "std": 294.36217992554947
                    }
                }
            }
        },
        "number_of_antennas": {
            "allweek": {
                "allday": {
                    "mean": 1.0,
                    "std": 0.0
                }
            }
        },
        "frequent_antennas": {
            "allweek": {
                "allday": {
                    "mean": 1.0,
                    "std": 0.0
                }
            }
        },
        "none_functions": [
            "duration",
            "percent_initiated_conversations",
            "percent_initiated_interactions",
            "response_delay_text",
            "response_rate_text",
            "balance_of_contacts",
            "percent_pareto_durations",
            "number_of_interaction_in",
            "number_of_interaction_out",
            "entropy_of_antennas",
            "percent_at_home",
            "radius_of_gyration",
            "churn_rate"
        ]
    }
```

Finally, it will be shown that the toolbox still works for CDR data. This data set has been extracted via the *bandicoot* App provided in *Google Play Store*.

```
In [24]: path2 = '../'

In [25]: data3 = pd.read_csv(path2+"metadata.csv")

In [27]: data3[:10]

Out[27]:   interaction direction                            correspondent_id  \
         0        call       out  1ca3e79268249930e7b5b20f70531e22233d2c33
         1        text        in  b22f56514f8ef784f4e9f6aebd7b98f96942d400
         2        text        in  b22f56514f8ef784f4e9f6aebd7b98f96942d400
         3        text        in  b22f56514f8ef784f4e9f6aebd7b98f96942d400
         4        text        in  562869765f146fff44b0303842323a49b5b95cff
         5        text        in  13d37a5097bb63214a6e3da479ee7ca8ce04caff
         6        text       out  13d37a5097bb63214a6e3da479ee7ca8ce04caff
         7        call        in  1ca3e79268249930e7b5b20f70531e22233d2c33
         8        text        in  aab513e40e81aee7ef073cf092834e1f49f87c07
         9        text       out  13d37a5097bb63214a6e3da479ee7ca8ce04caff

                       datetime  call_duration  antenna_id
         0  2016-05-13 19:03:38            NaN         NaN
         1  2016-05-13 21:06:17            NaN         NaN
         2  2016-05-13 21:17:53            NaN         NaN
         3  2016-05-13 21:21:23            NaN         NaN
         4  2016-05-13 22:02:47            NaN         NaN
         5  2016-05-13 22:34:48            NaN         NaN
         6  2016-05-13 23:36:49            NaN         NaN
         7  2016-05-14 15:48:34          199.0         NaN
```

```
        8  2016-05-16 09:37:21            NaN         NaN
        9  2016-05-16 13:17:12            NaN         NaN
```

```
In [28]: U = bc.io.read_csv("metadata",path2)
```

```
WARNING:root:100.00% of the records are missing a location.
         No antennas file was given and records are using antennas for position.
Warning: 100.00% of the records are missing a location.

No antennas file was given and records are using antennas for position.
WARNING:root: 2 record(s) are duplicated.

[x] 1541 records from 2016-05-13 19:03:38 to 2016-09-28 18:35:25
[x] 88 unique contacts
[x] 49 contacts call
[x] 50 contacts text
[ ] No attribute stored
[ ] No antenna stored
[ ] No recharges
[ ] No home
[x] Has texts
[x] Has calls
[ ] No network


Warning: 2 record(s) are duplicated.
```

We can now group the data in a different way. We will look at the data grouped separately, that is *call* and *text* separately, and also group the *call* and *text* together. This type of grouping can be done for any dataset, yielding many different and interesting indicators.

```
In [30]: bc.utils.all(U,show_all=False,interaction=["call","text",["call","text"]])
```

```
Out[30]: {
             "name": "metadata",
             "reporting": {
                 "antennas_path": None,
                 "attributes_path": None,
                 "recharges_path": None,
                 "version": "0.5.3",
                 "code_signature": "900718f0bed6a9c5dac2806eab9cccfc91d3cec9",
                 "groupby": "week",
                 "split_week": false,
                 "split_day": false,
                 "start_time": "2016-05-13 19:03:38",
                 "end_time": "2016-09-28 18:35:25",
                 "night_start": "19:00:00",
                 "night_end": "07:00:00",
                 "weekend": [
            6,
            7
        ],
        "number_of_records": 1541,
        "number_of_antennas": 0,
        "number_of_recharges": 0,
        "bins": 21,
        "bins_with_data": 21,
        "bins_without_data": 0,
        "has_call": true,
        "has_text": true,
        "has_home": false,
        "has_recharges": false,
        "has_attributes": false,
        "has_network": false,
        "percent_records_missing_location": 1.0,
        "antennas_missing_locations": 0,
        "percent_outofnetwork_calls": 0,
        "percent_outofnetwork_texts": 0,
        "percent_outofnetwork_contacts": 0,
        "percent_outofnetwork_durations": 0,
        "ignored_records": {
            "position": 0,
            "all": 0,
            "direction": 0,
            "correspondent_id": 0,
            "interaction": 0,
            "antenna_id": 0,
            "duration": 0,
            "datetime": 0
        }
    },
    "active_days": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": 5.428571428571429,
                    "std": 1.3652639225553762
                },
                "text": {
                    "mean": 5.857142857142857,
                    "std": 1.5208304212067918
                },
                "callandtext": {
                    "mean": 6.380952380952381,
                    "std": 1.32651316925563
                }
            }
    }
},
"number_of_contacts": {
    "allweek": {
        "allday": {
            "call": {
                "mean": 7.380952380952381,
                "std": 3.169440246151188
            },
            "text": {
                "mean": 9.904761904761905,
                "std": 3.57111109700051
            },
            "callandtext": {
                "mean": 14.380952380952381,
                "std": 4.61339860753633
            }
        }
    }
},
"duration": {
    "allweek": {
        "allday": {
            "call": {
                "mean": {
                    "mean": 77.9903615828616,
                    "std": 34.780686278490236
                },
                "std": {
                    "mean": 69.54787333586637,
                    "std": 30.240834209681015
                }
            },
            "text": {
                "mean": {
                    "mean": None,
                    "std": None
                },
                "std": {
                    "mean": None,
                    "std": None
                }
            },
            "callandtext": {
                "mean": {
                    "mean": 77.9903615828616,
                    "std": 34.780686278490236
                },
                    "std": {
                        "mean": 69.54787333586637,
                        "std": 30.240834209681015
                    }
                }
            }
        }
    },
    "percent_nocturnal": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": 0.16253114229374674,
                    "std": 0.16227761924157175
                },
                "text": {
                    "mean": 0.3751208670651482,
                    "std": 0.21122962711789556
                },
                "callandtext": {
                    "mean": 0.32776544579766864,
                    "std": 0.17911754998674098
                }
            }
        }
    },
    "percent_initiated_conversations": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": 0.0,
                    "std": 0.0
                },
                "text": {
                    "mean": 0.425321710044081,
                    "std": 0.10141579329959363
                },
                "callandtext": {
                    "mean": 0.4282067888067688,
                    "std": 0.10365188343423812
                }
            }
        }
    },
    "percent_initiated_interactions": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": 0.481364275453553,
                    "std": 0.1832121929950095
                },
                "text": {
                    "mean": 0.45839353379228626,
                    "std": 0.07529970828460919
                },
                "callandtext": {
                    "mean": 0.46229202036110034,
                    "std": 0.0759008114135604
                }
            }
        }
    },
    "response_delay_text": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": {
                        "mean": None,
                        "std": None
                    },
                    "std": {
                        "mean": None,
                        "std": None
                    }
                },
                "text": {
                    "mean": {
                        "mean": 474.1348872066148,
                        "std": 217.3510563565674
                    },
                    "std": {
                        "mean": 652.4912430446403,
                        "std": 272.3378615224165
                    }
                },
                "callandtext": {
                    "mean": {
                        "mean": 467.00246962419726,
                        "std": 227.65122198530315
                    },
                    "std": {
                        "mean": 631.6882967850754,
                        "std": 296.13913750132696
                    }
                }
            }
        }
    },
    "response_rate_text": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": None,
                    "std": None
                },
                "text": {
                    "mean": 0.30171934004861917,
                    "std": 0.15620899760771112
                },
                "callandtext": {
                    "mean": 0.30171934004861917,
                    "std": 0.15620899760771112
                }
            }
        }
    },
    "entropy_of_contacts": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": 1.647054481535061,
                    "std": 0.5838641496261422
                },
                "text": {
                    "mean": 1.7063719301105151,
                    "std": 0.40135487899527533
                },
                "callandtext": {
                    "mean": 2.010157484653839,
                    "std": 0.37429964422404854
                }
            }
        }
    },
    "balance_of_contacts": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": {
                        "mean": 0.0888911367821485,
                        "std": 0.10328055829431501
                    },
                    "std": {
                        "mean": 0.05902708555265434,
                    "std": 0.03991047746822355
                }
            },
            "text": {
                "mean": {
                    "mean": 0.05316908384712266,
                    "std": 0.024438770366315293
                },
                "std": {
                    "mean": 0.07124378421887713,
                    "std": 0.022984158275078843
                }
            },
            "callandtext": {
                "mean": {
                    "mean": 0.03572734436483614,
                    "std": 0.01246758043818512
                },
                "std": {
                    "mean": 0.054635975612160076,
                    "std": 0.015309094338676986
                }
            }
        }
    }
},
"interactions_per_contact": {
    "allweek": {
        "allday": {
            "call": {
                "mean": {
                    "mean": 2.2136191850477562,
                    "std": 0.7643746533995858
                },
                "std": {
                    "mean": 1.5767931631558592,
                    "std": 1.2277023859653053
                }
            },
            "text": {
                "mean": {
                    "mean": 5.757601459387173,
                    "std": 1.8523874352682312
                },
                "std": {
                    "mean": 6.786246957462101,
                    "std": 3.8746678478493677
                }
                },
                "callandtext": {
                    "mean": {
                        "mean": 5.069779038266434,
                        "std": 1.5308733914546122
                    },
                    "std": {
                        "mean": 6.733737332854262,
                        "std": 3.3226687742593417
                    }
                }
            }
        }
    },
    "interevent_time": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": {
                        "mean": 37792.92943234367,
                        "std": 18906.05640420461
                    },
                    "std": {
                        "mean": 39370.690935675535,
                        "std": 16015.052229904284
                    }
                },
                "text": {
                    "mean": {
                        "mean": 9540.774388325444,
                        "std": 5482.766431827406
                    },
                    "std": {
                        "mean": 21585.986263684852,
                        "std": 9558.724766872087
                    }
                },
                "callandtext": {
                    "mean": {
                        "mean": 7974.015344105529,
                        "std": 3605.8999596073472
                    },
                    "std": {
                        "mean": 17764.32704676006,
                        "std": 5355.4269700925815
                    }
                }
            }
        }
    },
    "percent_pareto_interactions": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": 0.36236005389311476,
                    "std": 0.16242260702929603
                },
                "text": {
                    "mean": 0.10221040455847577,
                    "std": 0.06818605667106467
                },
                "callandtext": {
                    "mean": 0.10286564601766933,
                    "std": 0.07212061530088389
                }
            }
        }
    },
    "percent_pareto_durations": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": 0.35012293262293265,
                    "std": 0.19467331509625
                },
                "text": {
                    "mean": None,
                    "std": None
                },
                "callandtext": {
                    "mean": 0.35012293262293265,
                    "std": 0.19467331509625
                }
            }
        }
    },
    "number_of_interactions": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": 16.285714285714285,
                    "std": 8.547585886658053
                },
                "text": {
                    "mean": 57.095238095238095,
                    "std": 24.189538789486512
                },
                "callandtext": {
                    "mean": 73.38095238095238,
                    "std": 27.668920488478268
                }
            }
        }
    },
    "number_of_interaction_in": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": 8.238095238095237,
                    "std": 4.689448763470544
                },
                "text": {
                    "mean": 29.904761904761905,
                    "std": 12.235408265337988
                },
                "callandtext": {
                    "mean": 38.142857142857146,
                    "std": 13.67404811446613
                }
            }
        }
    },
    "number_of_interaction_out": {
        "allweek": {
            "allday": {
                "call": {
                    "mean": 8.047619047619047,
                    "std": 4.725575706687725
                },
                "text": {
                    "mean": 27.19047619047619,
                    "std": 12.19140621452375
                },
                "callandtext": {
                    "mean": 35.23809523809524,
                    "std": 14.631683696706048
                }
            }
        }
    },
    "number_of_antennas": {
        "allweek": {
            "allday": {
                "mean": 1.0,
                        "std": 0.0
                    }
                }
            },
            "frequent_antennas": {
                "allweek": {
                    "allday": {
                        "mean": 1.0,
                        "std": 0.0
                    }
                }
            },
            "none_functions": [
                "entropy_of_antennas",
                "percent_at_home",
                "radius_of_gyration",
                "churn_rate"
            ]
        }
```

It can be seen that the functions that only returned **None** are spatial functions, which makes sense since we provided CDR data but not data about the antennas.

```
In [ ]:
```

B. TEST FUNCTIONS

The test functions provided with nosetest are shown on the following website https://benjaminglaas.github.io/testing/. The code can also be seen on the following pages.

nosetesting

December 6, 2016

0.1 nosetesting

The code provided here is the code for testing the modified bandicoot, source code given here. The code tests the indicators provided by the "utils.all()" function. The tests are run with the "nosetest" library.

For testing, the following data set has been synthesed. This data set is very simple and all the indicators can be computed by hand.

```
In [3]: import pandas as pd

In [4]: pd.read_csv("../data/test_indicators.csv")

Out[4]:                datetime interaction  correspondent_id  duration
        0  2014-01-06 08:00:00    physical               371         1
        1  2014-01-06 08:00:10    physical               285         1
        2  2014-01-06 08:00:20    physical               371         1
        3  2014-01-06 08:00:30    physical               371         1
        4  2014-01-06 08:00:40    physical               371         1
        5  2014-01-06 08:00:50    physical               371         1
        6  2014-01-06 08:01:00    physical               371         1
        7  2014-01-06 08:01:10    physical               371         1
        8  2014-01-06 08:01:20    physical               371         1
        9  2014-01-06 08:01:30    physical               371         1
```

One then runs the following code by putting it in a script, e.g. called "test_all.py" and run the command "nosetests test_all.py" in a terminal.

```
"""
Test to see if indicators for modified bandicoot are computed correctly.
Indicators are computed from the following functions in "all"
"""

import unittest
from bc_dev.utils import all
from bc_dev.io import read_csv

class TestAll(unittest.TestCase):
```

```
def setUp(self):
    self.user = read_csv("test_indicators","../data/",
                        warnings=False,describe=False)
    self.r = all(self.user,show_all=True,interaction="physical")
    #Grouping is per default per week.

def test_active_days(self):
    "User is only present the 6th of January 2014."
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": 1.0,
                    "std": 0.0
                },
            }
        }
    }
    self.assertEqual(out,self.r["active_days"])

def test_number_of_contacts(self):
    "Contacts are 285 and 371."
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": 2.0,
                    "std": 0.0
                },
            }
        }
    }
    self.assertEqual(out,self.r["number_of_contacts"])

def test_duration(self):
    "All durations are 1 second."
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": {
                        "mean": 1.0,
                        "std": 0.0
                    },
                    "std": {
                        "mean": 0.0,
                        "std": 0.0
                    }
                },
            }
        }
    }
    self.assertEqual(out,self.r["duration"])

def test_percent_nocturnal(self):
    "All interactions are within 1 minute and 30 seconds of 8 am."
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": 0.0,
                    "std": 0.0
                },
            }
        }
    }
    self.assertEqual(out,self.r["percent_nocturnal"])

def test_percent_initiated_conversations(self):
    "There is no 'direction' attribute, therefore 0."
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": 0.0,
                    "std": 0.0
                },
            }
        }
    }
    self.assertEqual(out,self.r["percent_initiated_conversations"])

def test_percent_initiated_interactions(self):
    "There is no 'direction' attribute, therefore 0."
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": 0.0,
                    "std": 0.0
                },
            }
        }
    }
    self.assertEqual(out,self.r["percent_initiated_interactions"])
```

```
def test_response_delay_text(self):
    "Interaction types are not equal to 'text', so None."
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": {
                        "mean": None,
                        "std": None
                    },
                    "std": {
                        "mean": None,
                        "std": None
                    }

                },
            }
        }
    }
    self.assertEqual(out,self.r["response_delay_text"])

def test_response_rate_text(self):
    "Interaction types are not equal to 'text', so None."
    out = out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": None,
                    "std": None
                },
            }
        }
    }
    self.assertEqual(out,self.r["response_rate_text"])

def test_entropy_of_contacts(self):
    """Shannon entropy is defined as (from entropy function)

    if len(data) == 0:
        return None

    n = sum(data)

    _op = lambda f: f * math.log(f)
    return - sum(_op(float(i) / n) for i in data)

    As there are the correspondent_id's
```

```
        285 with 1 interaction
        371 with 9 interactions

        the entropy will be
        -((1/10)*ln(1/10)+(9/10)*ln(9/10)) = 0.3250829733914482,
        """

        out = {
            "allweek": {
                "allday": {
                    "physical": {
                        "mean": 0.3250829733914482,
                        "std": 0.0
                    },
                }
            }
        }
        self.assertEqual(out,self.r["entropy_of_contacts"])

    def test_balance_of_contacts(self):
        "Only computed for records having a 'direction'."
        out = {
            "allweek": {
                "allday": {
                    "physical": {
                        "mean": {
                            "mean": None,
                            "std": None
                        },
                        "std": {
                            "mean": None,
                            "std": None
                        }

                    },
                }
            }
        }
        self.assertEqual(out,self.r["balance_of_contacts"])

    def test_interactions_per_contacts(self):
        """
        The mean is 10/2 = 5.
        The variance is defined as
        ((1-5)^2 + (9-5)^2)/2 = 16.0
        The std is therefore
        std = sqrt(16.0) = 4.0
        """
        out = {
            "allweek": {
                "allday": {
                    "physical": {
                        "mean": {
                            "mean": 5.0,
                            "std": 0.0
                        },
                        "std": {
                            "mean": 4.0,
                            "std": 0.0
                        }

                    },
                }
            }
        }
        self.assertEqual(out,self.r["interactions_per_contact"])

    def test_interevent_time(self):
        "All of the events are 10 seconds apart."
        out = {
            "allweek": {
                "allday": {
                    "physical": {
                        "mean": {
                            "mean": 10.0,
                            "std": 0.0
                        },
                        "std": {
                            "mean": 0.0,
                            "std": 0.0
                        }

                    },
                }
            }
        }
        self.assertEqual(out,self.r["interevent_time"])

    def test_percent_pareto_interactions(self):
        """There is 1 contact that contributes to 90 % of the interactions,
        and the other 10 %. 1 contact contributes to more than 80 %,
        divided by 10 observations give 0.1."""

        out = {
            "allweek": {
                "allday": {
```

```
                "physical": {
                    "mean": 0.1,
                    "std": 0.0
                },
            }
        }
    }
    self.assertEqual(out,self.r["percent_pareto_interactions"])

def test_percent_pareto_durations(self):
    """Function only looks at 'call' interactions,
    therefore the result is 0.0. Could however be
    modified to look at all interactions,
    but this will be saved for another time."""
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": 0.0,
                    "std": 0.0
                },
            }
        }
    }
    self.assertEqual(out,self.r["percent_pareto_durations"])

def test_number_of_interactions(self):
    """10 records."""
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": 10.0,
                    "std": 0.0
                },
            }
        }
    }
    self.assertEqual(out,self.r["number_of_interactions"])

def test_number_of_interactions_in(self):
    "There is no 'direction' attribute, therefore 0."
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": 0.0,
                    "std": 0.0
                },
            }
        }
    }
    self.assertEqual(out,self.r["number_of_interaction_in"])

def test_number_of_interactions_out(self):
    "There is no 'direction' attribute, therefore 0."
    out = {
        "allweek": {
            "allday": {
                "physical": {
                    "mean": 0.0,
                    "std": 0.0
                },
            }
        }
    }
    self.assertEqual(out,self.r["number_of_interaction_out"])

def test_number_of_antennas(self):
    """Scalar type datatype. Even though no antennas are provided,
    the output of the 'positions' list is
    [None]. Therefore, taking the len(set(positions)) gives 1."""
    out = {
        "allweek": {
            "allday": {
                    "mean": 1.0,
                    "std": 0.0
                },
            }
        }

    self.assertEqual(out,self.r["number_of_antennas"])

def test_entropy_of_antennas(self):
    "No antennase, therefore 0."
    out = {
        "allweek": {
            "allday": {
                    "mean": 0.0,
                    "std": 0.0
                },
            }
        }
    self.assertEqual(out,self.r["entropy_of_antennas"])

def test_percent_at_home(self):
    "No antennas, therefore no home."
    out = {
        "allweek": {
            "allday": {
                    "mean": None,
                    "std": None
                },
            }
        }
    self.assertEqual(out,self.r["percent_at_home"])

def test_radius_of_gyration(self):
    "No antennas, therefore None."
    out = {
        "allweek": {
            "allday": {
                    "mean": None,
                    "std": None
                }
            }
        }
    self.assertEqual(out,self.r["radius_of_gyration"])

def test_frequent_antennas(self):
    """ Since the positions list is [None],
    there is 1 observation. This yields 'mean' = 1.0, since 1 location
    takes up more than 80 % (which is 0.8
    observations)of the total number of positions."""
    out = {
        "allweek": {
            "allday": {
                    "mean": 1.0,
                    "std": 0.0
                },
            }
        }
    self.assertEqual(out,self.r["frequent_antennas"])

def test_churn_rate(self):
    """ Taking 'statistics' of an empty list.
    Statistics yields {'mean': , 'std', },
    which here gives 'None' for both."""
    out = {
            "mean": None,
            "std": None
        }

    self.assertEqual(out,self.r["churn_rate"])
```