ipprl_tools Documentation: Corrupting Existing Data v1.0

August 2019

Contents

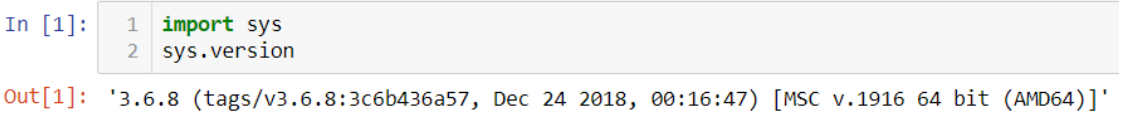
1 Compatibility

The *ipprl_tools* package was written using Python 3.6, but should be compatible with any version of Python 3 (Python 3.x).

2 Required Dependencies

2.1 Language

ipprl_tools requires Python 3.x to be installed prior to using the package. To check which version of Python you are using, run the following commands in your interpreter:



```
In [1]:  import sys
          sys.version

Out[1]: '3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)]'
```

Figure 1: Checking the installed version of Python.

To install Python, visit `https://www.python.org/` and download the installer for your Operating System.

2.2 Python Package Dependencies

The following packages are required dependencies for the *ipprl_tools* package. If you installed *ipprl_tools* through PIP, these dependencies should be installed automatically.

- **Pandas** $\geq$ v0.23

  - https://pandas.pydata.org

- **NumPy** $\geq$ v1.16
  - https://www.numpy.org
- **SciPy** $\geq$ v1.2
  - https://www.scipy.org

3 Optional Dependencies

The following packages are optional dependencies for the *ipprl_tools* package. These dependencies will not be installed automatically when installing *ipprl_tools* with PIP, so they must be installed manually if needed.

- **Fuzzy** $\geq$ v1.2.2
  - This package is required for the Soundex corruption method. For more information about the package, visit https://pypi.org/project/Fuzzy/.
- **Jupyter** $\geq$ v1.0.0
  - This package is required to view and run the tutorial Jupyter notebook. For more information about Jupyter, visit https://jupyter.org/

4 Installation

4.1 PIP Method (Recommended)

To install the package via PIP run the command:

```
pip install git+git://github.com/cu-recordlinkage/ipprl_tools
```

through a command-line interface.

This command will install the *ipprl_tools* package into your default Python environment. This command will also install the required dependencies (Pandas, NumPy, SciPy, etc) if they are not already installed.

4.2 GitHub Method

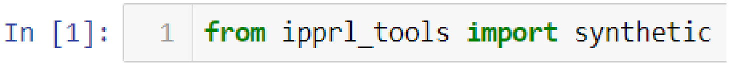
: The source code can also be cloned directly from GitHub using the following command from a command-line interface.

```
git clone https://github.com/cu-recordlinkage/ipprl_tools
```

5 Usage

5.1 Importing the Package

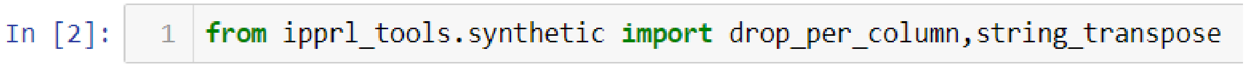
To use *ipprl_tools*, first import the `synthetic` submodule.



```
from ipprl_tools import synthetic
```

Figure 2: Importing *ipprl_tools*

This command will import all of the functions defined in the `synthetic` submodule. If you only need a subset of the functions, you can specify the exact functions to import as well:



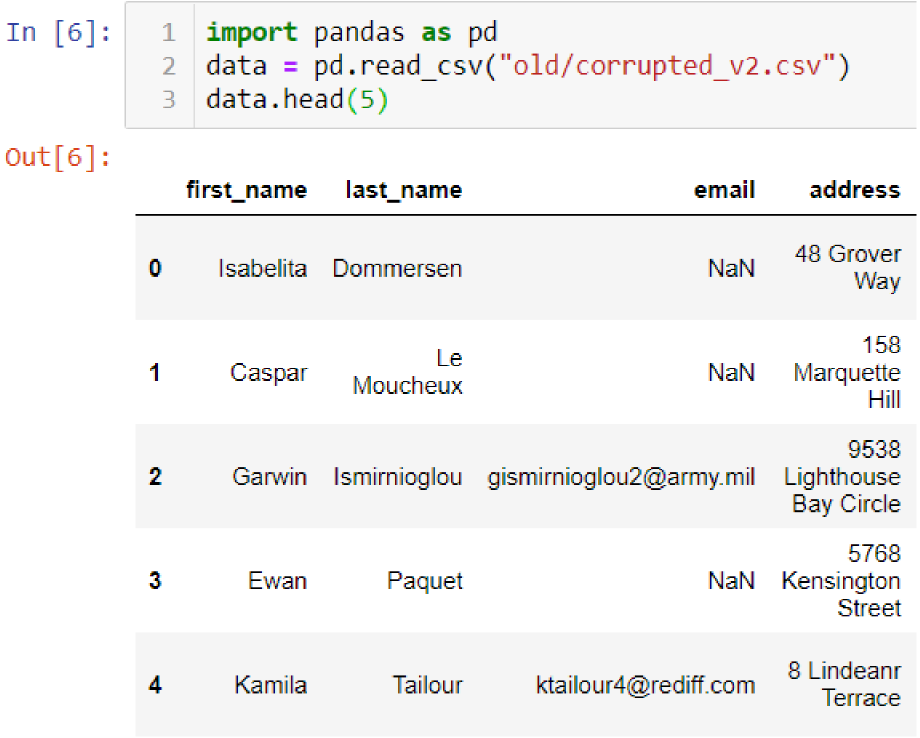
```
from ipprl_tools.synthetic import drop_per_column,string_transpose
```

Figure 3: Importing specific functions

5.2 Data Prerequisites

The synthetic data functions are designed to operate on Pandas DataFrame objects. Pandas DataFrame objects are data structures, similar to R dataframes, which contain data organized in named columns.

Before using the synthetic data methods, first read the raw data in as a Pandas DataFrame. The example below shows reading a CSV file in using Pandas.
For additional ways to import data using Pandas, refer to the Pandas Documentation here:
Pandas Documentation: IO



```
In [6]: import pandas as pd
        data = pd.read_csv("old/corrupted_v2.csv")
        data.head(5)
Out[6]:
```

	first_name	last_name	email	address
0	Isabelita	Dommersen	NaN	48 Grover Way
1	Caspar	Le Moucheux	NaN	158 Marquette Hill
2	Garwin	Ismirnioglou	gismirnioglou2@army.mil	9538 Lighthouse Bay Circle
3	Ewan	Paquet	NaN	5768 Kensington Street
4	Kamila	Tailour	ktailour4@rediff.com	8 Lindeanr Terrace

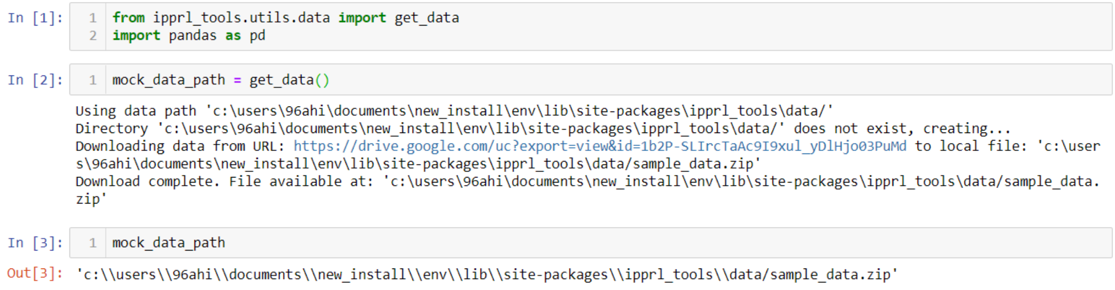
Figure 4: Reading in a CSV file with Panda

6 Corrupting Existing Data

In order to make generating synthetic datasets easier, *npprl_tools* provides a file a pre-made synthetic data, generated using Mockaroo (Link: `https://mockaroo.com/`).
Using *ipprl_tools*, the user can automatically download this data, import it into their Python interpreter, and perform corruption using the corruption methods provided in *ipprl_tools*.

6.1 Downloading the Data

To download the data, first import the `get_data()` function from the `utils.data` submodule of *ipprl_tools*.

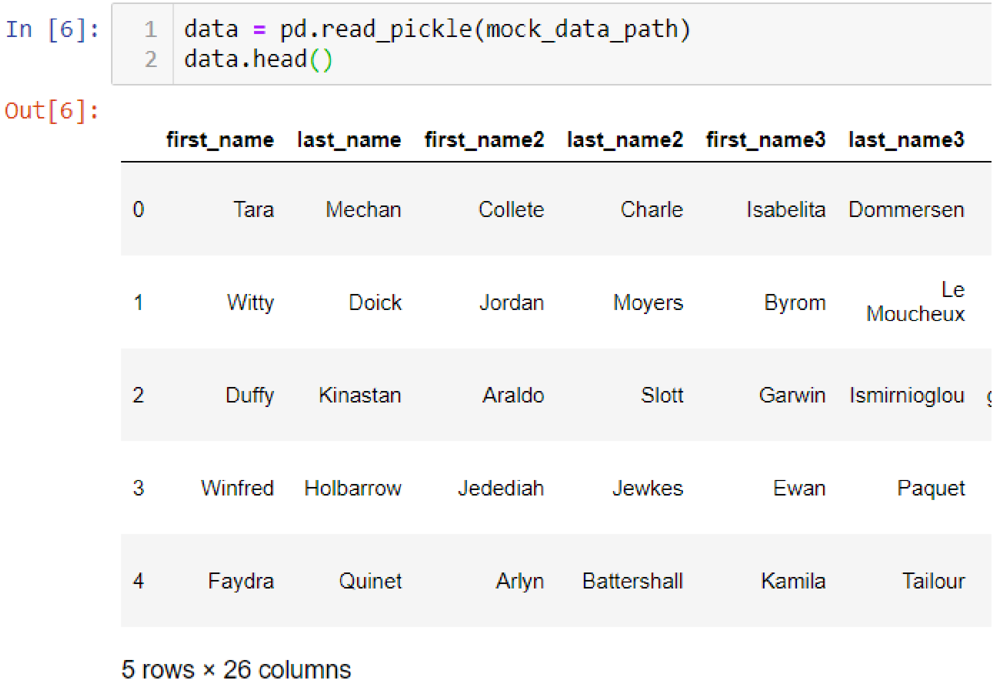


```
In [1]: from ipprl_tools.utils.data import get_data
        import pandas as pd

In [2]: mock_data_path = get_data()

        Using data path 'c:\users\96ahi\documents\new_install\env\lib\site-packages\ipprl_tools\data/'
        Directory 'c:\users\96ahi\documents\new_install\env\lib\site-packages\ipprl_tools\data/' does not exist, creating...
        Downloading data from URL: https://drive.google.com/uc?export=view&id=1b2P-SLIrcTaAc9I9xul_yDlHjo03PuMd to local file: 'c:\user
        s\96ahi\documents\new_install\env\lib\site-packages\ipprl_tools\data/sample_data.zip'
        Download complete. File available at: 'c:\users\96ahi\documents\new_install\env\lib\site-packages\ipprl_tools\data/sample_data.
        zip'

In [3]: mock_data_path

Out[3]: 'c:\\users\\96ahi\\documents\\new_install\\env\\lib\\site-packages\\ipprl_tools\\data/sample_data.zip'
```

Figure 5: Importing the *ipprl_tools* submodule, along with Pandas (So we can read the file in as a DataFrame).

In the above figure, get_data() determines that the pre-made data bundle has not been downloaded before, so it creates a new directory and downloads the data. get_data() returns the file path where the data bundle was downloaded. In future calls to get_data(), the data path will be returned immediately without re-downloading the data unless the file is moved or deleted.

After downloading the data bundle, we can read it in to a DataFrame using the read_pickle() method from Pandas.



In [6]:

```
data = pd.read_pickle(mock_data_path)
data.head()
```

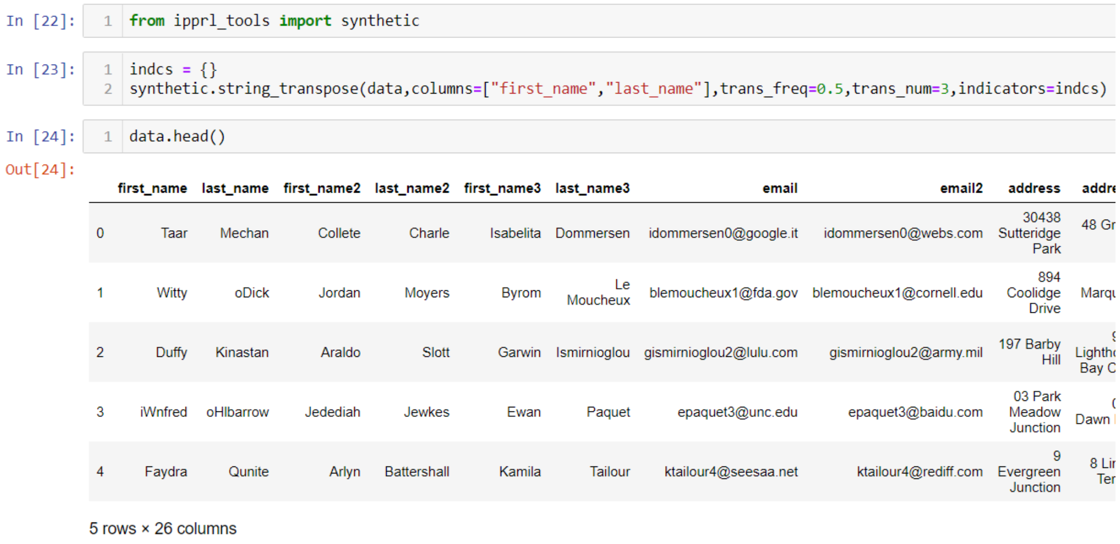
Out[6]:

	first_name	last_name	first_name2	last_name2	first_name3	last_name3
0	Tara	Mechan	Collete	Charle	Isabelita	Dommersen
1	Witty	Doick	Jordan	Moyers	Byrom	Le Moucheux
2	Duffy	Kinastan	Araldo	Slott	Garwin	Ismirnioglou
3	Winfred	Holbarrow	Jedediah	Jewkes	Ewan	Paquet
4	Faydra	Quinet	Arlyn	Battershall	Kamila	Tailour

5 rows × 26 columns

Figure 6: Reading the data into a DataFrame using Pandas.

At this point, we are able to apply any corruption method desired to the DataFrame.



```
In [22]: from ipprl_tools import synthetic

In [23]: indcs = {}
         synthetic.string_transpose(data,columns=["first_name","last_name"],trans_freq=0.5,trans_num=3,indicators=indcs)

In [24]: data.head()

Out[24]:
```

	first_name	last_name	first_name2	last_name2	first_name3	last_name3	email	email2	address	addre
0	Taar	Mechan	Collete	Charle	Isabelita	Dommersen	idommersen0@google.it	idommersen0@webs.com	30438 Sutteridge Park	48 Gr
1	Witty	oDick	Jordan	Moyers	Byrom	Le Moucheux	blemoucheux1@fda.gov	blemoucheux1@cornell.edu	894 Coolidge Drive	Marqu
2	Duffy	Kinastan	Araldo	Slott	Garwin	Ismirnioglou	gismirnioglou2@lulu.com	gismirnioglou2@army.mil	197 Barby Hill	9 Lightho Bay C
3	iWnfred	oHlbarrow	Jedediah	Jewkes	Ewan	Paquet	epaquet3@unc.edu	epaquet3@baidu.com	03 Park Meadow Junction	(Dawn
4	Faydra	Qunite	Arlyn	Battershall	Kamila	Tailour	ktailour4@seesaa.net	ktailour4@rediff.com	9 Evergreen Junction	8 Lir Ter

5 rows × 26 columns

Figure 7: Applying a String Transpose corruption to the mock data.

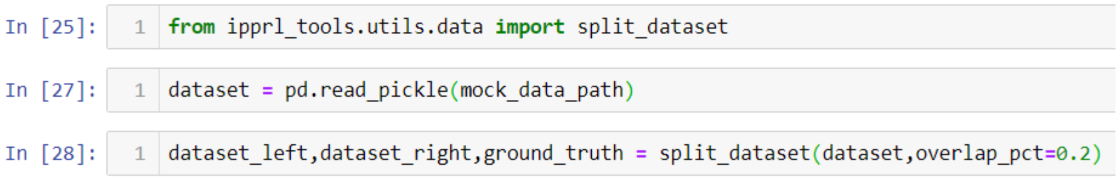
In the following three cells we import the `synthetic` sub-module of *ipprl_tools*, which contains all of the methods for data corruption. We then apply a String Transpose corruption to the `first_name` and `last_name` columns of the data.

After viewing, we can see that some of the values in `first_name` and `last_name` have been randomly transposed.

For a complete overview of the synthetic data corruption methods and their usage, refer to the ***ipprl_tools* Documentation: Synthetic Data Corruption Tools** document, which contains information about the `synthetic` sub-module of *ipprl_tools*.

6.2 Splitting Data for Linkage

One common use case for mock patient record data is testing the performance of linkage methods. To assist with this use case, *ipprl_tools* also provides a utility for splitting the data bundle (or your own dataset) into two equal-sized groups, with a user-specified amount of overlapping records.

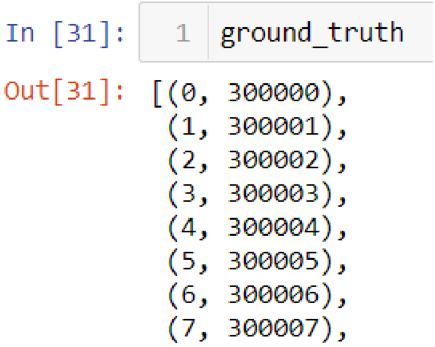


```
In [25]: from ipprl_tools.utils.data import split_dataset

In [27]: dataset = pd.read_pickle(mock_data_path)

In [28]: dataset_left,dataset_right,ground_truth = split_dataset(dataset,overlap_pct=0.2)
```

Figure 8: Importing the *split_dataset()* function, and applying it on a DataFrame.

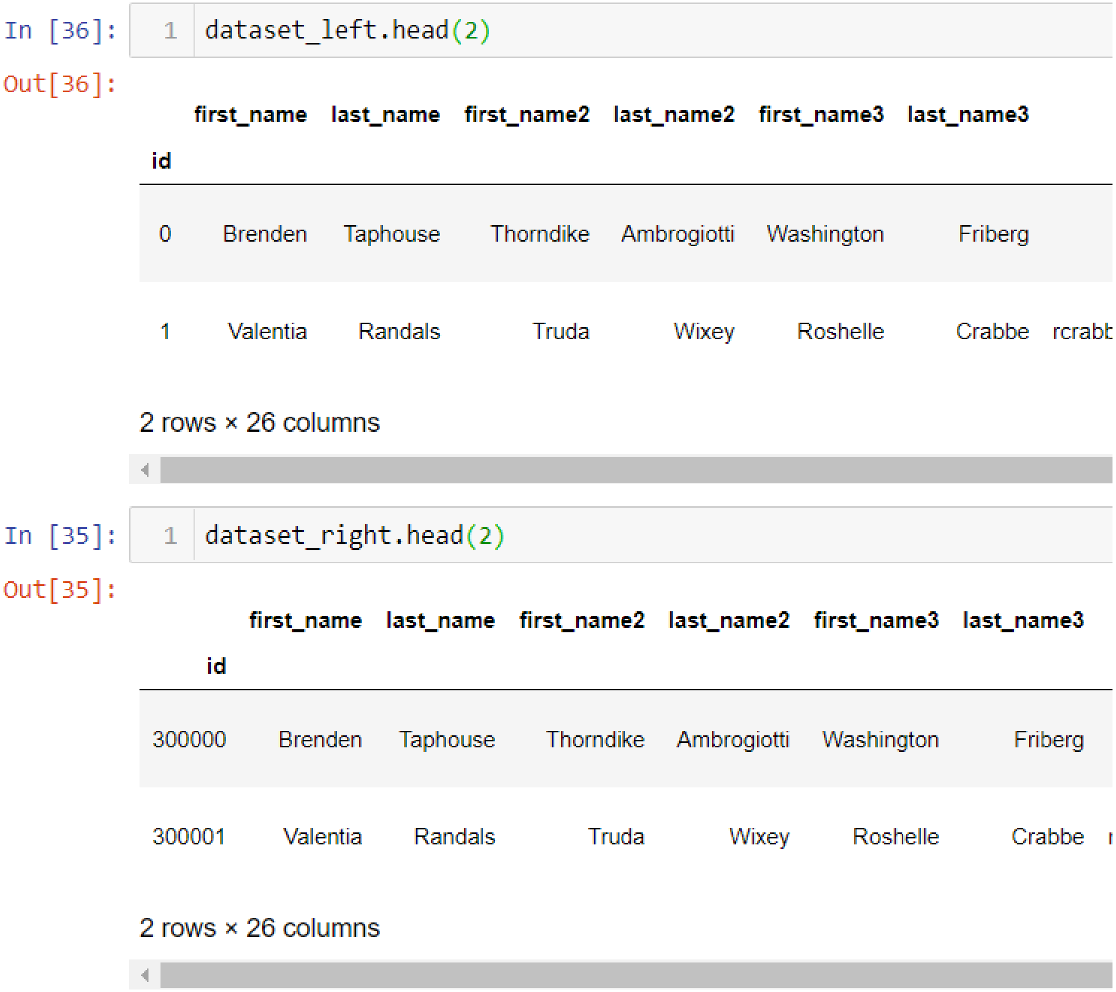
In the above example, we read mock data path into a new DataFrame called `dataset`, then call `split_dataset(dataset,overlap_pct=0.2)` on the DataFrame. This call will split the data bundle into two equal-sized datasets, with 20% of the records from `dataset` appearing in both new datasets.

This function call returns two DataFrames (referred to in the example code as `dataset_left` and `dataset_right`). In addition, this function call returns `ground_truth`, which is a list of tuples mapping IDs in `dataset_left` to IDs in `dataset_right`. These IDs are generated by the function, and are contained in the `id` column of both `dataset_left` and `dataset_right`.



```
In [31]:  ground_truth

Out[31]: [(0, 300000),
          (1, 300001),
          (2, 300002),
          (3, 300003),
          (4, 300004),
          (5, 300005),
          (6, 300006),
          (7, 300007),
```

Figure 9: View of *ground_truth*, which is a list of tuples of integers, mapping IDs from *dataset_left* to *dataset_right*.



In [36]:

```
dataset_left.head(2)
```

Out[36]:

	first_name	last_name	first_name2	last_name2	first_name3	last_name3	
id							
0	Brenden	Taphouse	Thorndike	Ambrogiotti	Washington	Friberg	
1	Valentia	Randals	Truda	Wixey	Roshelle	Crabbe	rcrabt

2 rows × 26 columns

In [35]:

```
dataset_right.head(2)
```

Out[35]:

	first_name	last_name	first_name2	last_name2	first_name3	last_name3	
id							
300000	Brenden	Taphouse	Thorndike	Ambrogiotti	Washington	Friberg	
300001	Valentia	Randals	Truda	Wixey	Roshelle	Crabbe	ı

2 rows × 26 columns

Figure 10: Views of *dataset_left* and *dataset_right*. The *id* is visible as the left-most column in both DataFrames.

As is visible in Figures 9 and 10, the `ground_truth` variable links the `IDs` of matching records between `dataset_left` and `dataset_right`.

After this process is complete, `dataset_left` and `dataset_right` can be corrupted and shuffled independently, then used as inputs for record linkage. When the linkage process is complete, the user can compare the results against the `ground_truth` list to determine the performance of the linkage on the synthetic data.