DataCleaningInPython

June 18, 2018

1 Data Cleaning in python

This practical was based on Section 2 of "An Introduction ot data cleaning with R" from Statistics Netherlands, and adapted for python. * Available at https://cran.r-project.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf

1.1 Logging in and getting started

1.1.1 Data files to be set up

unnamed.txt

```
21,6.0
42,5.9
18,5.7*
21,NA
```

daltons.txt

```
%% Data on the Dalton Brothers
Gratt,1861,1892
Bob,1892
1871,Emmet,1937
% Names, birth and death dates
```

1.1.2 On Your Own Laptop

Prerequisites Python 2.7 and Conda.

Command line install:

```
condo create --name pythonData
conda install -n pythonData Jupyter pandas
source activate pythonData
jupyter notebook
```

Open http://localhost:8888 in browser.

1.1.3 On EPCC's RDF (Research Data Facility)

- Open a terminal window and run the following commands: *NB Replace port number 8889 with another number between 8000-9000

```
# Login
ssh username@login.rdf.ac.uk
# Load python modules
module load python
module load anaconda
# Create working directory
mkdir dataCleaning
cd dataCleaning
# create the input data files
nano unnamed.txt
nano daltons.txt
ipython notebook --no-browser --port=8889
```

Mac or Unix connection to RDF

- Open another terminal window (mac or Unix) and run this command:

```
ssh -N -f -L localhost:8888:localhost:8889 username@login.rdf.ac.uk
```

- Go to http://localhost:8888 in your browser. Open the dataCleaning directory and create a new notebook to work in.

Windows connection to RDF Open putty set ssh connection: `Host Name: user@IP port: 22 set putty/connections/SSH/tunnels source: local port:8888 Destination: remote server: localhost:8889`

2 Part 1: Read csv data into a data frame

Start with a file called unnamed.txt containing the following text:

```
21,6.0
42,5.9
18,5.7*
21,NA
```

2.0.1 Start Python Notebook

As described above.

2.0.2 read_csv

- Import the pandas module as pd
- Read this with pd.read_csv()
- What has happened to the first row? - it's a header.

```
In [65]: import pandas as pd
         pd.read_csv("unnamed.txt")

Out[65]:    21   6.0
         0  42   5.9
         1  18  5.7*
         2  21   NaN
```

2.0.3 Header row

- Read this again with header=None as an argument
- What has happened now?

```
In [66]: pd.read_csv("unnamed.txt", header=None)

Out[66]:    0     1
         0  21   6.0
         1  42   5.9
         2  18  5.7*
         3  21   NaN
```

2.0.4 Setting the column names

```
In [67]: pd.read_csv("unnamed.txt", header=None, names=('age','height'))

Out[67]:    age height
         0   21    6.0
         1   42    5.9
         2   18   5.7*
         3   21    NaN

In [68]: person = pd.read_csv("unnamed.txt", header=None, names=('age','height'))
         person

Out[68]:    age height
         0   21    6.0
         1   42    5.9
         2   18   5.7*
         3   21    NaN

In [69]: person.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
age       4 non-null int64
height    3 non-null object
dtypes: int64(1), object(1)
memory usage: 136.0+ bytes
```

- Let's convert the height column into numeric values What happened to 5.7*?

```
In [70]: person.height = pd.to_numeric(person.height)
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-70-ffe88830763a> in <module>()
----> 1 person.height = pd.to_numeric(person.height)


/Users/eilidhtroup/anaconda/envs/pythonData/lib/python2.7/site-packages/pandas/core/to
124             coerce_numeric = False if errors in ('ignore', 'raise') else True
125             values = lib.maybe_convert_numeric(values, set(),
--> 126                                                coerce_numeric=coerce_numeric)
127
128     except Exception:


pandas/_libs/src/inference.pyx in pandas._libs.lib.maybe_convert_numeric (pandas/_libs/


ValueError: Unable to parse string "5.7*" at position 2
```

- What happens to 5.7* when you coerce to a numeric value?

```
In [71]: person.height = pd.to_numeric(person.height, errors='coerce')
         person
```

Out[71]:

	age	height
0	21	6.0
1	42	5.9
2	18	NaN
3	21	NaN

Let's check the structure. It's a data frame containing: * an age column of ints * a height columns of floats.

```
In [72]: person.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
age       4 non-null int64
height    2 non-null float64
dtypes: float64(1), int64(1)
memory usage: 136.0 bytes
```

3 Part 2 Dealing with unstructured data

3.1 Step 1 Readlines

```
In [73]: with open("daltons.txt") as f:
             txt = f.readlines()
```

```
In [74]: txt
```

```
Out[74]: ['%% Data on the Dalton Brothers\r\n',
          'Gratt,1861,1892\r\n',
          'Bob,1892\r\n',
          '1871,Emmet,1937\r\n',
          '% Names, birth and death dates\r\n']
```

3.1.1 Pattern matching

Several example datasets come as part of Python's scikit-learn package. http://scikit-learn.org/stable/tutorial/basic/tutorial.html#loading-example-dataset

However, here we are going to download a csv file.

```
In [77]: iris = pd.read_csv('https://github.com/pandas-dev/pandas/raw/master/pandas/tests/data/
```

```
In [78]: iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
SepalLength    150 non-null float64
SepalWidth     150 non-null float64
PetalLength    150 non-null float64
PetalWidth     150 non-null float64
Name           150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

```
In [79]: names = iris.columns.tolist()  # Alternatively list(iris)
```

```
In [80]: names
```

```
Out[80]: ['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', 'Name']
```

Using list comprehension

- Python's list comprehension applys a function to each element in a list.

```
In [81]: numbers = [4,5,6]
         [x*2 for x in numbers]

Out[81]: [8, 10, 12]

In [82]: 'Petal' in 'PetalLength'

Out[82]: True

In [83]: ["Petal" in name for name in names]

Out[83]: [False, False, True, True, False]

In [84]: [name for name in names if 'Petal' in name]

Out[84]: ['PetalLength', 'PetalWidth']
```

Regular expressions

- As above, using regular expressions

```
In [85]: import re
         [name for name in names if re.search("Petal", name)]

Out[85]: ['PetalLength', 'PetalWidth']
```

^ matches pattern at start

```
In [86]: [name for name in names if re.search("^P", name)]

Out[86]: ['PetalLength', 'PetalWidth']
```

$ matches pattern at end

```
In [87]: [name for name in names if re.search("th$", name)]

Out[87]: ['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth']
```

[] character class, match characters enclosed in []

```
In [88]: [name for name in names if re.search("[g][t][h]", name)]

Out[88]: ['SepalLength', 'PetalLength']
```

For more see help(re) for full explanation.

3.1.2 An aside on Simple string matching alternatives (regular expression are more powerful, and return position of match)

```
In [89]: [name for name in names if name.startswith("P")]

Out[89]: ['PetalLength', 'PetalWidth']

In [90]: [name.startswith("P") for name in names]

Out[90]: [False, False, True, True, False]

In [91]: [name for name in names if name.endswith("th")]

Out[91]: ['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth']

In [92]: [name for name in names if "gth" in name]

Out[92]: ['SepalLength', 'PetalLength']
```

3.1.3 Subsetting and Logicals

- Logical AND &

```
In [93]: iris[(iris.Name == "Iris-versicolor") & (iris.PetalWidth >= 1.7)]

Out[93]:     SepalLength  SepalWidth  PetalLength  PetalWidth            Name
         70          5.9         3.2          4.8         1.8  Iris-versicolor
         77          6.7         3.0          5.0         1.7  Iris-versicolor
```

- Logical OR |

```
In [94]: iris[(iris.SepalLength == 4.3) | (iris.SepalLength == 7.9)]

Out[94]:      SepalLength  SepalWidth  PetalLength  PetalWidth            Name
         13           4.3         3.0          1.1         0.1     Iris-setosa
         131          7.9         3.8          6.4         2.0  Iris-virginica
```

- Note == for comparison not =
- Logical NOT ~

```
In [95]: iris[~(iris.SepalLength > 4.3)]

Out[95]:     SepalLength  SepalWidth  PetalLength  PetalWidth         Name
         13          4.3         3.0          1.1         0.1  Iris-setosa
```

3.1.4 Selecting rows and columns

- Pandas filter() command selects columns
- Can filter by regular expression

```
In [96]: iris.filter(regex='^P').columns
```

```
Out[96]: Index([u'PetalLength', u'PetalWidth'], dtype='object')
```

- Select columns and rows at the same time

```
In [97]: iris.filter(regex='^P')[~(iris.SepalLength > 4.3)]
```

```
Out[97]:     PetalLength  PetalWidth
         13          1.1         0.1
```

3.2 Step 2 (cont) Selecting lines only with data

Find lines starting with a % sign

```
In [98]: [name for name in txt if re.search("^%", name)]
```

```
Out[98]: ['%% Data on the Dalton Brothers\r\n', '% Names, birth and death dates\r\n']
```

```
In [99]: dat = [name for name in txt if not re.search("^%", name)]
         dat
```

```
Out[99]: ['Gratt,1861,1892\r\n', 'Bob,1892\r\n', '1871,Emmet,1937\r\n']
```

3.3 Step 3 - split lines into fields

- For each line, we now want to extract the content for each field.
- We now need to know about splitting lines and learn about lists in Python.

```
In [100]: L = [1,2, "three", [3,3]]
```

[] retrieves an object from the list. Indexing starts at zero.

```
In [101]: L[0]
```

```
Out[101]: 1
```

Can select a range of values

```
In [102]: L[0:3]
```

```
Out[102]: [1, 2, 'three']
```

Use - to count from end e.g. last 2 items

```
In [103]: L[-2]
```

```
Out[103]: 'three'
```

From second last to end

```
In [104]: L[-2:]
```

```
Out[104]: ['three', [3, 3]]
```