Using Python to read data files and explore their contents

This notebook demonstrates using the Pandas (http://pandas.pydata.org) data processing library to read a dataset into Python, and obtain a basic understanding of its contents.

Note that Python by itself is a general-purpose programming language and does not provide high-level data processing capabilities. The Pandas library was developed to meet this need. Pandas is the most popular Python library for data manipulation, and we will use it extensively in this course.

In addition to Pandas, we will also make use of the following Python libraries

- Numpy (http://www.numpy.org) is a library for working with arrays of data
- Matplotlib (https://matplotlib.org) is a library for making graphs
- Seaborn (https://seaborn.pydata.org) is a higher-level interface to Matplotlib that can be used to simplify many graphing tasks
- Statsmodels (https://www.statsmodels.org/stable/index.html) is a library that implements many statistical techniques
- Scipy (https://www.scipy.org) is a library of techniques for numerical and scientific computing

Importing libraries

When using Python, you must always begin your scripts by importing the libraries that you will be using. After importing a library, its functions can then be called from your code by prepending the library name to the function name. For example, to use the '`dot`' function from the '`numpy`' library, you would enter '`numpy.dot`'. To avoid repeatedly having to type the libary name in your scripts, it is conventional to define a two or three letter abbreviation for each library, e.g. '`numpy`' is usually abbreviated as '`np`'. This allows us to use '`np.dot`' instead of '`numpy.dot`'. Similarly, the Pandas library is typically abbreviated as '`pd`'.

The following statement imports the Pandas library, and gives it the abbreviated name 'pd'.

```
In [1]: import pandas as pd
```

Reading a data file

We will be working with the NHANES (National Health and Nutrition Examination Survey) data from the 2015-2016 wave, which has been discussed earlier in this course. The raw data for this study are available here:

https://wwwn.cdc.gov/nchs/nhanes/Default.aspx (https://wwwn.cdc.gov/nchs/nhanes/Default.aspx)

As in many large studies, the NHANES data are spread across multiple files. The NHANES files are stored in SAS transport (https://v8doc.sas.com/sashtml/files/z0987199.htm) (Xport) format. This is a somewhat obscure format, and while Pandas is perfectly capable of reading the NHANES data directly from the xport files, accomplishing this task is a more advanced topic than we want to get into here. Therefore, for this course we have prepared some merged datasets in text/csv format.

Pandas is a large and powerful library. Here we will only use a few of its basic features. The main data structure that Pandas works with is called a "data frame". This is a two-dimensional table of data in which the rows typically represent cases (e.g. NHANES subjects), and the columns represent variables. Pandas also has a one-dimensional data structure called a `Series` that we will encounter occasionally.

Pandas has a variety of functions named with the pattern '`read_xxx`' for reading data in different formats into Python. Right now we will focus on reading '`csv`' files, so we are using the '`read_csv`' function, which can read csv (and "tsv") format files that are exported from spreadsheet software like Excel. The '`read_csv`' function by default expects the first row of the data file to contain column names.

Using '`read_csv`' in its default mode is fairly straightforward. There are many options to '`read_csv`' that are useful for handling less-common situations. For example, you would use the option `sep='\t'` instead of the default `sep=','` if the fields of your data file are delimited by tabs instead of commas. See here (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html) for the full documentation for '`read_csv`'.

Pandas can read a data file over the internet when provided with a URL, which is what we will do below. In the Python script we will name the data set '`da`', i.e. this is the name of the Python variable that will hold the data frame after we have loaded it.

The variable '`url`' holds a string (text) value, which is the internet URL where the data are located. If you have the data file in your local filesystem, you can also use '`read_csv`' to read the data from this file. In this case you would pass a file path instead of a URL, e.g. `pd.read_csv("my_file.csv")` would read a file named `my_file.csv` that is located in your current working directory.

```
In [2]: url = "nhanes_2015_2016.csv"
        da = pd.read_csv(url)
```

To confirm that we have actually obtained the data the we are expecting, we can display the shape (number of rows and columns) of the data frame in the notebook. Note that the final expression in any Jupyter notebook cell is automatically printed, but you can force other expressions to be printed by using the '`print`' function, e.g. '`print(da.shape)`'.

Based on what we see below, the data set being read here has 5735 rows, corresponding to 5735 people in this wave of the NHANES study, and 28 columns, corresponding to 28 variables in this particular data file. Note that NHANES collects thousands of variables on each study subject, but here we are working with a reduced file that contains a limited number of variables.

```
In [3]: da.shape
Out[3]: (5735, 28)
```

Exploring the contents of a data set

Pandas has a number of basic ways to understand what is in a data set. For example, above we used the '`shape`' method to determine the numbers of rows and columns in a data set. The columns in a Pandas data frame have names, to see the names, use the '`columns`' method:

```
In [4]: da.columns
```

```
Out[4]: Index(['SEQN', 'ALQ101', 'ALQ110', 'ALQ130', 'SMQ020', 'RIAGENDR', 'RIDAGEYR',
               'RIDRETH1', 'DMDCITZN', 'DMDEDUC2', 'DMDMARTL', 'DMDHHSIZ', 'WTINT2YR',
               'SDMVPSU', 'SDMVSTRA', 'INDFMPIR', 'BPXSY1', 'BPXDI1', 'BPXSY2',
               'BPXDI2', 'BMXWT', 'BMXHT', 'BMXBMI', 'BMXLEG', 'BMXARML', 'BMXARMC',
               'BMXWAIST', 'HIQ210'],
              dtype='object')
```

These names correspond to variables in the NHANES study. For example, `SEQN` is a unique identifier for one person, and `BMXWT` is the subject's weight in kilograms ("BMX" is the NHANES prefix for body measurements). The variables in the NHANES data set are documented in a set of "codebooks" that are available on-line. The codebooks for the 2015-2016 wave of NHANES can be found by following the links at the following page:

https://wwwn.cdc.gov/nchs/nhanes/continuousnhanes/default.aspx?BeginYear=2015 (https://wwwn.cdc.gov/nchs/nhanes/continuousnhanes/default.aspx?BeginYear=2015)

For convenience, direct links to some of the code books are included below:

- Demographics code book (https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/DEMO_I.htm)
- Body measures code book (https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/BMX_I.htm)
- Blood pressure code book (https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/BPX_I.htm)
- Alcohol questionaire code book (https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/ALQ_I.htm)
- Smoking questionaire code book (https://wwwn.cdc.gov/Nchs/Nhanes/2015-2016/SMQ_I.htm)

Every variable in a Pandas data frame has a data type. There are many different data types, but most commonly you will encounter floating point values (real numbers), integers, strings (text), and date/time values. When Pandas reads a text/csv file, it guesses the data types based on what it sees in the first few rows of the data file. Usually it selects an appropriate type, but occasionally it does not. To confirm that the data types are consistent with what the variables represent, inspect the '`dtypes`' attribute of the data frame.

```
In [5]: da.dtypes
```

```
Out[5]: SEQN          int64
        ALQ101      float64
        ALQ110      float64
        ALQ130      float64
        SMQ020        int64
        RIAGENDR      int64
        RIDAGEYR      int64
        RIDRETH1      int64
        DMDCITZN    float64
        DMDEDUC2    float64
        DMDMARTL    float64
        DMDHHSIZ      int64
        WTINT2YR    float64
        SDMVPSU       int64
        SDMVSTRA      int64
        INDFMPIR    float64
        BPXSY1      float64
        BPXDI1      float64
        BPXSY2      float64
        BPXDI2      float64
        BMXWT       float64
        BMXHT       float64
        BMXBMI      float64
        BMXLEG      float64
        BMXARML     float64
        BMXARMC     float64
        BMXWAIST    float64
        HIQ210      float64
        dtype: object
```

As we see here, most of the variables have floating point or integer data type. Unlike many data sets, NHANES does not use any text values in its data. For example, while many datasets would use text labels like "F" or "M" to denote a subject's gender, this information is represented in NHANES with integer codes. The actual meanings of these codes can be determined from the codebooks. For example, the variable `RIAGENDR` contains each subject's gender, with male gender coded as `1` and female gender coded as `2`. The `RIAGENDR` variable is part of the demographics component of NHANES, so this coding can be found in the demographics codebook.

Variables like `BMXWT` which represent a quantitative measurement will typically be stored as floating point data values.

Slicing a data set

As discussed above, a Pandas data frame is a rectangular data table, in which the rows represent cases and the columns represent variables. One common manipulation of a data frame is to extract the data for one case or for one variable. There are several ways to do this, as shown below.

To extract all the values for one variable, the following three approaches are equivalent ("DMDEDUC2" here is an NHANES variable containing a person's educational attainment). In these four lines of code, we are assigning the data from one column of the data frame `da` into new variables `w`, `x`, `y`, and `z`. The first three approaches access the variable by name. The fourth approach accesses the variable by position (note that `DMDEDUC2` is in position 9 of the `da.columns` array shown above -- remember that Python counts starting at position zero).

```
In [9]: w = da["DMDEDUC2"]
        x = da.loc[:, "DMDEDUC2"]
        y = da.DMDEDUC2
        z = da.iloc[:, 9]  # DMDEDUC2 is in column 9
```

Another reason to slice a variable out of a data frame is so that we can then pass it into a function. For example, we can find the maximum value over all `DMDEDUC2` values using any one of the following four lines of code:

```
In [7]: print(da["DMDEDUC2"].max())
        print(da.loc[:, "DMDEDUC2"].max())
        print(da.DMDEDUC2.max())
        print(da.iloc[:, 9].max())
```

```
9.0
9.0
9.0
9.0
```

Every value in a Python program has a type, and the type information can be obtained using Python's '`type`' function. This can be useful, for example, if you are looking for the documentation associated with some value, but you do not know what the value's type is.

Here we see that the variable `da` has type 'DataFrame', while one column of `da` has type 'Series'. As noted above, a Series is a Pandas data structure for holding a single column (or row) of data.

```
In [8]: print(type(da)) # The type of the variable
        print(type(da.DMDEDUC2)) # The type of one column of the data frame
        print(type(da.iloc[2,:])) # The type of one row of the data frame
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

It may also be useful to slice a row (case) out of a data frame. Just as a data frame's columns have names, the rows also have names, which are called the "index". However many data sets do not have meaningful row names, so it is more common to extract a row of a data frame using its position. The `iloc` method slices rows or columns from a data frame by position (counting from 0). The following line of code extracts row 3 from the data set (which is the fourth row, counting from zero).

```
In [9]: x = da.iloc[3, :]
```

Another important data frame manipulation is to extract a contiguous block of rows or columns from the data set. Below we slice by position, in the first case taking row positions 3 and 4 (counting from 0, which are rows 4 and 5 counting from 1), and in the second case taking columns 2, 3, and 4 (columns 3, 4, 5 if counting from 1).

```
In [10]: x = da.iloc[3:5, :]
         y = da.iloc[:, 2:5]
```

Missing values

When reading a dataset using Pandas, there is a set of values including 'NA', 'NULL', and 'NaN' that are taken by default to represent a missing value. The full list of default missing value codes is in the '`read_csv`' documentation here (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html). This document also explains how to change the way that '`read_csv`' decides whether a variable's value is missing.

Pandas has functions called `isnull` and `notnull` that can be used to identify where the missing and non-missing values are located in a data frame. Below we use these functions to count the number of missing and non-missing `DMDEDUC2` values.

```
In [11]: print(pd.isnull(da.DMDEDUC2).sum())
         print(pd.notnull(da.DMDEDUC2).sum())
```

```
261
5474
```

As an aside, note that there may be a variety of distinct forms of missingness in a variable, and in some cases it is important to keep these values distinct. For example, in case of the DMDEDUC2 variable, in addition to the blank or NA values that Pandas considers to be missing, three people responded "don't know" (code value 9). In many analyses, the "don't know" values will also be treated as missing, but at this point we are considering "don't know" to be a distinct category of observed response.