Intro to Jupyter Notebook

May 15, 2018

1 Introduction to the Basics of Anaconda & Jupyter Notebook

Michelle Hoogenhout

1.1 A note on terminology

Conda is the package manager (e.g. conda list displays all installed packages in the environment), whereas Anaconda and Miniconda are distributions. A software distribution is a pre-build and pre-configured collection of packages that can be installed and used on a system, while a package manager is a tool that automates the process of installing, updating, and removing packages.

Anaconda is a full distribution of the central software in the PyData ecosystem, and includes Python itself along with the binaries for several hundred third-party open-source projects.

Miniconda is essentially an installer for an empty conda environment, containing only Conda, its dependencies, and Python. Once Conda is installed, you can then install whatever package you need from scratch along with any desired version of Python.

We will be using the Jupyter Notebook IDE (Integrated Development Environment) for projects.

1.2 Installation

Install miniconda instead of anaconda when space is limited. If installing miniconda, you should also install the packages you want to use (e.g. numpy) as it does not come with pre-installed packages like anaconda does.

When installing python and anaconda/miniconda, make sure to add python and conda to the path list.

1.3 To open jupyter notebook:

1. Open anaconda prompt
2. Navigate to directory where you want the notebook, e.g. cd Dropbox/Umuzi
3. Type jupyter notebook

You could also do 2 &3 in one step with the following code:
jupyter notebook --notebook-dir=C:/Users/Michelle/Dropbox/consulting/Umuzi

1.4 You can access bash commands directly from notebook!

```
In [1]: %pwd

Out[1]: 'C:\\Users\\Michelle\\Dropbox\\consulting\\Umuzi'

In [2]: # print working directory
        %pwd

Out[2]: 'C:\\Users\\Michelle\\Dropbox\\consulting\\Umuzi'

In [1]: #list files in current directory
        %ls

 Volume in drive C is OS
 Volume Serial Number is 3C50-45C9

 Directory of C:\Users\Michelle\Dropbox\consulting\Umuzi

14/05/2018  09:00    <DIR>          .
14/05/2018  09:00    <DIR>          ..
09/05/2018  11:07    <DIR>          .ipynb_checkpoints
09/05/2018  12:50        20,865,282 [Jake_VanderPlas]_Python_Data_Science_Handbook.pdf
09/05/2018  12:50        10,190,392 [John_Mueller,_Zanab_Hussain,_Luca_Massaron]_Python_for_data
09/05/2018  12:51         5,924,775 [Madhavan_S.]_Mastering_Python_for_Data_Science.pdf
08/05/2018  17:22    <DIR>          application test
08/05/2018  17:22    <DIR>          cheatsheets
01/05/2018  23:24           202,745 cohens_d_confidence_interval_vs_sample_size.png
01/05/2018  23:19            14,043 conf_int_graph.png
01/05/2018  23:22            30,153 conf_int_sizes.jpg
01/05/2018  23:19             7,864 conf1.gif
11/05/2018  12:59    <DIR>          CrimeStats
07/05/2018  08:48         5,477,638 everyday stats.pdf
09/05/2018  12:54        90,007,117 Guttag_Introduction_to_Computation_and_Python.pdf
14/05/2018  09:00            96,437 Intro to Jupyter Notebook.ipynb
08/05/2018  17:22    <DIR>          invoices
17/02/2017  16:36             2,396 mtcars.csv
09/05/2018  11:08            30,889 notebook.tex
09/05/2018  11:08             4,672 output_10_0.png
09/05/2018  11:08             7,355 output_11_1.png
09/05/2018  11:08            11,854 output_12_0.png
09/05/2018  11:08             8,052 output_13_1.png
09/05/2018  13:02    <DIR>          python lessons
11/05/2018  12:54            19,818 Report instructions. Crime and crime perception in Gauteng.d
13/05/2018  09:12         5,232,017 The Stats Sidekick.pdf
09/05/2018  07:45               555 Untitled.ipynb
13/05/2018  09:12           914,796 Uri Stats Sidekick Hypothesis Testing.pdf
              20 File(s)    139,048,850 bytes
               8 Dir(s)  236,534,550,528 bytes free
```

In [4]:
```
#Get all line magics
%lsmagic
```

Out[4]:
```
Available line magics:
%alias  %alias_magic  %autocall  %automagic  %autosave  %bookmark  %cd  %clear  %cls  %c

Available cell magics:
%%!  %%HTML  %%SVG  %%bash  %%capture  %%cmd  %%debug  %%file  %%html  %%javascript  %%j

Automagic is ON, % prefix IS NOT needed for line magics.
```

A cell with text is created by changing the format from code to Markdown

2 You can do all the usual things with notebook that you can do with python

2.1 Graphs

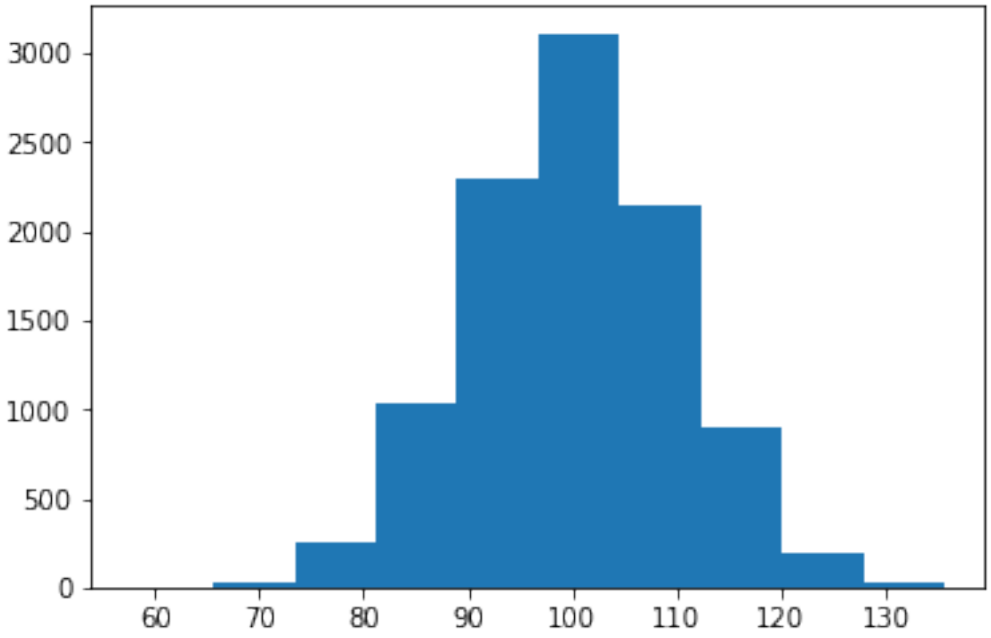
In [5]:
```
#import numpy and matplotlib
import numpy as np
import matplotlib.pyplot as plt

#make new random variable
x = np.random.normal(100,10,10000)

#print(x)
```

In [6]:
```
#this command lets us make plots as we go along
%matplotlib inline
```

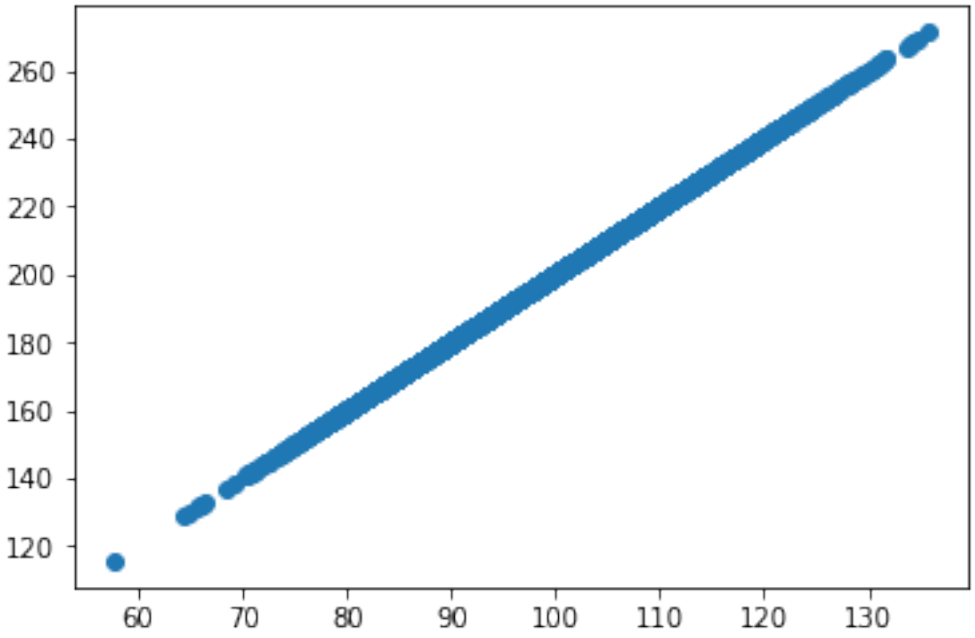
In [7]:
```
#make histogram
plt.hist(x, bins = 10)
plt.show()
```



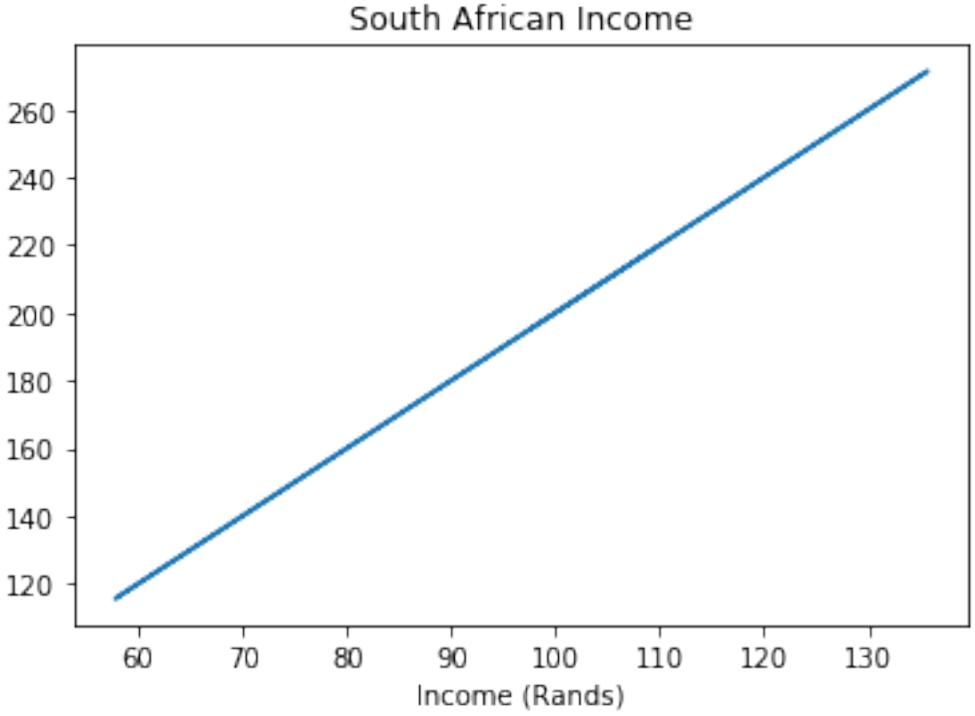

```
In [8]: y = x*2
        print(y)

        #make scatterplot
        plt.scatter(x,y)
        plt.show()
```

```
[195.64561027 203.55422482 179.12247876 ... 182.09128771 191.04134353
 219.21998865]
```



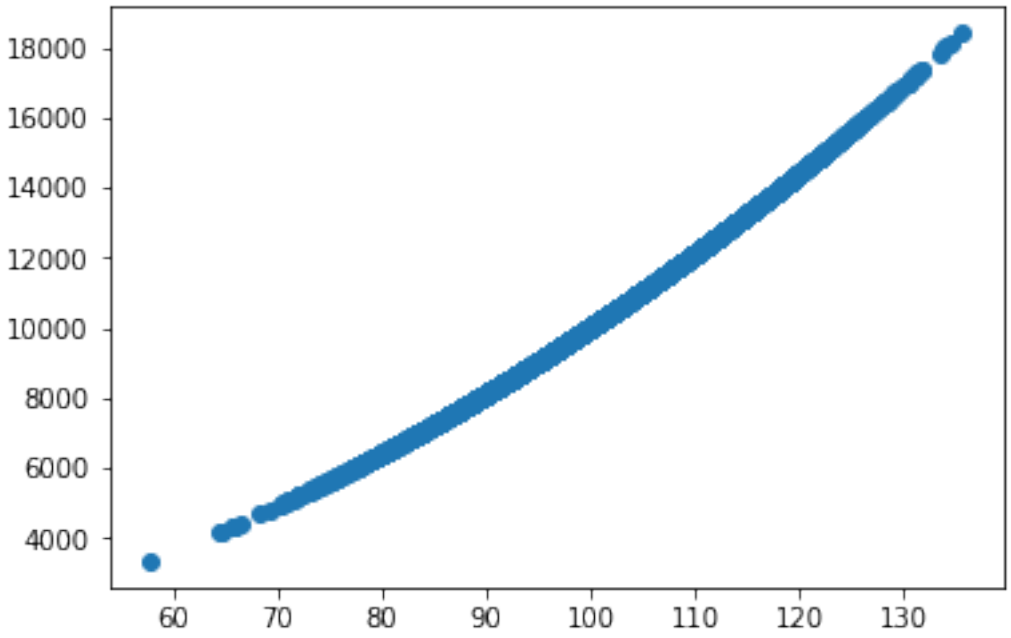
In [9]:
```
#make line plot with x-axis label and title
plt.plot(x,y)
plt.xlabel("Income (Rands)")
plt.title("South African Income")
plt.show()
```



```
In [10]: #if we change the value of y, our subsequent plot will change to show the new values of
         y = x**2
         print(y)

         plt.scatter(x,y)
         plt.show()
```

```
[ 9569.30120458 10358.58061098  8021.21559888 ...  8289.30926494
  9124.19873479 12014.35085611]
```



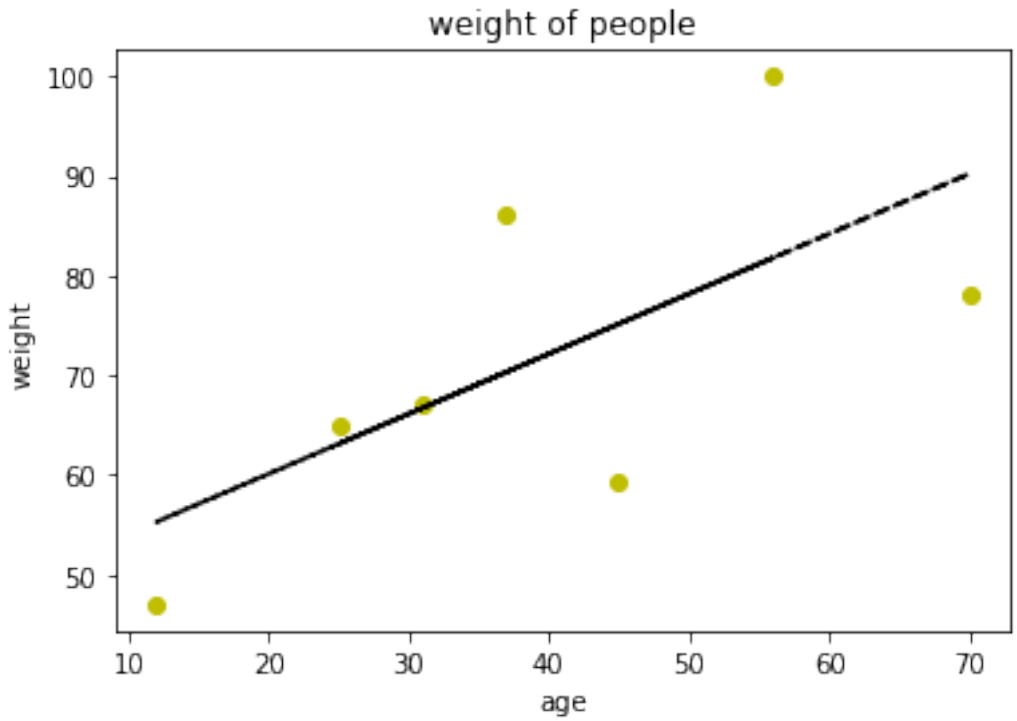
2.1.1 Create graph with regression line

```
In [22]: #create example variables
         age  = [37, 25, 70, 45, 12, 56, 31]
         weight = [86.2, 65, 78, 59.4, 47, 100, 67]

         #fit linear regression
         fit = np.polyfit(age,weight,1)
         fit_fn = np.poly1d(fit)
         # fit_fn is now a function which takes in x and returns an estimate for y

         #plot regression line (fit_fn) scatter dots
```

```
plt.plot(age,weight, 'yo', age, fit_fn(age), '--k')
plt.xlabel("age")
plt.ylabel("weight")
plt.title("weight of people")
plt.show()
```



2.2 Work with data frames

```
In [21]: import pandas
         cars = pandas.read_csv('CrimeStats/mtcars.csv', skiprows = 1, index_col = 0)
         cars
```

Out[21]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
2	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
3	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
4	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
6	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
8	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
9	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
10	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
11	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4

```
12  16.4    8  275.8  180  3.07  4.070  17.40   0   0     3     3
13  17.3    8  275.8  180  3.07  3.730  17.60   0   0     3     3
14  15.2    8  275.8  180  3.07  3.780  18.00   0   0     3     3
15  10.4    8  472.0  205  2.93  5.250  17.98   0   0     3     4
16  10.4    8  460.0  215  3.00  5.424  17.82   0   0     3     4
17  14.7    8  440.0  230  3.23  5.345  17.42   0   0     3     4
18  32.4    4   78.7   66  4.08  2.200  19.47   1   1     4     1
19  30.4    4   75.7   52  4.93  1.615  18.52   1   1     4     2
20  33.9    4   71.1   65  4.22  1.835  19.90   1   1     4     1
21  21.5    4  120.1   97  3.70  2.465  20.01   1   0     3     1
22  15.5    8  318.0  150  2.76  3.520  16.87   0   0     3     2
23  15.2    8  304.0  150  3.15  3.435  17.30   0   0     3     2
24  13.3    8  350.0  245  3.73  3.840  15.41   0   0     3     4
25  19.2    8  400.0  175  3.08  3.845  17.05   0   0     3     2
26  27.3    4   79.0   66  4.08  1.935  18.90   1   1     4     1
27  26.0    4  120.3   91  4.43  2.140  16.70   0   1     5     2
28  30.4    4   95.1  113  3.77  1.513  16.90   1   1     5     2
29  15.8    8  351.0  264  4.22  3.170  14.50   0   1     5     4
30  19.7    6  145.0  175  3.62  2.770  15.50   0   1     5     6
31  15.0    8  301.0  335  3.54  3.570  14.60   0   1     5     8
32  21.4    4  121.0  109  4.11  2.780  18.60   1   1     4     2

    fuel.consumption  car.weight  engine.size
1          11.190476       2.620        160.0
2          11.190476       2.875        160.0
3          10.307018       2.320        108.0
4          10.981308       3.215        258.0
5          12.566845       3.440        360.0
6          12.983425       3.460        225.0
7          16.433566       3.570        360.0
8           9.631148       3.190        146.7
9          10.307018       3.150        140.8
10         12.239583       3.440        167.6
11         13.202247       3.440        167.6
12         14.329268       4.070        275.8
13         13.583815       3.730        275.8
14         15.460526       3.780        275.8
15         22.596154       5.250        472.0
16         22.596154       5.424        460.0
17         15.986395       5.345        440.0
18          7.253086       2.200         78.7
19          7.730263       1.615         75.7
20          6.932153       1.835         71.1
21         10.930233       2.465        120.1
22         15.161290       3.520        318.0
23         15.460526       3.435        304.0
24         17.669173       3.840        350.0
25         12.239583       3.845        400.0
```

```
26          8.608059       1.935         79.0
27          9.038462       2.140        120.3
28          7.730263       1.513         95.1
29         14.873418       3.170        351.0
30         11.928934       2.770        145.0
31         15.666667       3.570        301.0
32         10.981308       2.780        121.0
```

In [16]:

```
Help on function read_csv in module pandas.io.parsers:

read_csv(filepath_or_buffer, sep=',', delimiter=None, header='infer', names=None, index_col=None
    Read CSV (comma-separated) file into DataFrame

    Also supports optionally iterating or breaking of the file
    into chunks.

    Additional help can be found in the `online docs for IO Tools
    <http://pandas.pydata.org/pandas-docs/stable/io.html>`_.

    Parameters
    ----------
    filepath_or_buffer : str, pathlib.Path, py._path.local.LocalPath or any object with a read()
        The string could be a URL. Valid URL schemes include http, ftp, s3, and
        file. For file URLs, a host is expected. For instance, a local file could
        be file ://localhost/path/to/table.csv
    sep : str, default ','
        Delimiter to use. If sep is None, the C engine cannot automatically detect
        the separator, but the Python parsing engine can, meaning the latter will
        be used and automatically detect the separator by Python's builtin sniffer
        tool, ``csv.Sniffer``. In addition, separators longer than 1 character and
        different from ``'\s+'`` will be interpreted as regular expressions and
        will also force the use of the Python parsing engine. Note that regex
        delimiters are prone to ignoring quoted data. Regex example: ``'\r\t'``
    delimiter : str, default ``None``
        Alternative argument name for sep.
    delim_whitespace : boolean, default False
        Specifies whether or not whitespace (e.g. ``' '`` or ``'    '``) will be
        used as the sep. Equivalent to setting ``sep='\s+'``. If this option
        is set to True, nothing should be passed in for the ``delimiter``
        parameter.

        .. versionadded:: 0.18.1 support for the Python parser.

    header : int or list of ints, default 'infer'
        Row number(s) to use as the column names, and the start of the
        data.  Default behavior is to infer the column names: if no names
```

```
    are passed the behavior is identical to ``header=0`` and column
    names are inferred from the first line of the file, if column
    names are passed explicitly then the behavior is identical to
    ``header=None``. Explicitly pass ``header=0`` to be able to
    replace existing names. The header can be a list of integers that
    specify row locations for a multi-index on the columns
    e.g. [0,1,3]. Intervening rows that are not specified will be
    skipped (e.g. 2 in this example is skipped). Note that this
    parameter ignores commented lines and empty lines if
    ``skip_blank_lines=True``, so header=0 denotes the first line of
    data rather than the first line of the file.
names : array-like, default None
    List of column names to use. If file contains no header row, then you
    should explicitly pass header=None. Duplicates in this list will cause
    a ``UserWarning`` to be issued.
index_col : int or sequence or False, default None
    Column to use as the row labels of the DataFrame. If a sequence is given, a
    MultiIndex is used. If you have a malformed file with delimiters at the end
    of each line, you might consider index_col=False to force pandas to _not_
    use the first column as the index (row names)
usecols : array-like or callable, default None
    Return a subset of the columns. If array-like, all elements must either
    be positional (i.e. integer indices into the document columns) or strings
    that correspond to column names provided either by the user in `names` or
    inferred from the document header row(s). For example, a valid array-like
    `usecols` parameter would be [0, 1, 2] or ['foo', 'bar', 'baz'].

    If callable, the callable function will be evaluated against the column
    names, returning names where the callable function evaluates to True. An
    example of a valid callable argument would be ``lambda x: x.upper() in
    ['AAA', 'BBB', 'DDD']``. Using this parameter results in much faster
    parsing time and lower memory usage.
as_recarray : boolean, default False
    .. deprecated:: 0.19.0
       Please call `pd.read_csv(...).to_records()` instead.

    Return a NumPy recarray instead of a DataFrame after parsing the data.
    If set to True, this option takes precedence over the `squeeze` parameter.
    In addition, as row indices are not available in such a format, the
    `index_col` parameter will be ignored.
squeeze : boolean, default False
    If the parsed data only contains one column then return a Series
prefix : str, default None
    Prefix to add to column numbers when no header, e.g. 'X' for X0, X1, ...
mangle_dupe_cols : boolean, default True
    Duplicate columns will be specified as 'X.0'...'X.N', rather than
    'X'...'X'. Passing in False will cause data to be overwritten if there
    are duplicate names in the columns.
```

```
dtype : Type name or dict of column -> type, default None
    Data type for data or columns. E.g. {'a': np.float64, 'b': np.int32}
    Use `str` or `object` to preserve and not interpret dtype.
    If converters are specified, they will be applied INSTEAD
    of dtype conversion.
engine : {'c', 'python'}, optional
    Parser engine to use. The C engine is faster while the python engine is
    currently more feature-complete.
converters : dict, default None
    Dict of functions for converting values in certain columns. Keys can either
    be integers or column labels
true_values : list, default None
    Values to consider as True
false_values : list, default None
    Values to consider as False
skipinitialspace : boolean, default False
    Skip spaces after delimiter.
skiprows : list-like or integer or callable, default None
    Line numbers to skip (0-indexed) or number of lines to skip (int)
    at the start of the file.

    If callable, the callable function will be evaluated against the row
    indices, returning True if the row should be skipped and False otherwise.
    An example of a valid callable argument would be ``lambda x: x in [0, 2]``.
skipfooter : int, default 0
    Number of lines at bottom of file to skip (Unsupported with engine='c')
skip_footer : int, default 0
    .. deprecated:: 0.19.0
       Use the `skipfooter` parameter instead, as they are identical
nrows : int, default None
    Number of rows of file to read. Useful for reading pieces of large files
na_values : scalar, str, list-like, or dict, default None
    Additional strings to recognize as NA/NaN. If dict passed, specific
    per-column NA values.  By default the following values are interpreted as
    NaN: '', '#N/A', '#N/A N/A', '#NA', '-1.#IND', '-1.#QNAN', '-NaN', '-nan',
    '1.#IND', '1.#QNAN', 'N/A', 'NA', 'NULL', 'NaN', 'n/a', 'nan',
    'null'.
keep_default_na : bool, default True
    If na_values are specified and keep_default_na is False the default NaN
    values are overridden, otherwise they're appended to.
na_filter : boolean, default True
    Detect missing value markers (empty strings and the value of na_values). In
    data without any NAs, passing na_filter=False can improve the performance
    of reading a large file
verbose : boolean, default False
    Indicate number of NA values placed in non-numeric columns
skip_blank_lines : boolean, default True
    If True, skip over blank lines rather than interpreting as NaN values
```

```
parse_dates : boolean or list of ints or names or list of lists or dict, default False

    * boolean. If True -> try parsing the index.
    * list of ints or names. e.g. If [1, 2, 3] -> try parsing columns 1, 2, 3
      each as a separate date column.
    * list of lists. e.g.  If [[1, 3]] -> combine columns 1 and 3 and parse as
      a single date column.
    * dict, e.g. {'foo' : [1, 3]} -> parse columns 1, 3 as date and call result
      'foo'

    If a column or index contains an unparseable date, the entire column or
    index will be returned unaltered as an object data type. For non-standard
    datetime parsing, use ``pd.to_datetime`` after ``pd.read_csv``

    Note: A fast-path exists for iso8601-formatted dates.
infer_datetime_format : boolean, default False
    If True and `parse_dates` is enabled, pandas will attempt to infer the
    format of the datetime strings in the columns, and if it can be inferred,
    switch to a faster method of parsing them. In some cases this can increase
    the parsing speed by 5-10x.
keep_date_col : boolean, default False
    If True and `parse_dates` specifies combining multiple columns then
    keep the original columns.
date_parser : function, default None
    Function to use for converting a sequence of string columns to an array of
    datetime instances. The default uses ``dateutil.parser.parser`` to do the
    conversion. Pandas will try to call `date_parser` in three different ways,
    advancing to the next if an exception occurs: 1) Pass one or more arrays
    (as defined by `parse_dates`) as arguments; 2) concatenate (row-wise) the
    string values from the columns defined by `parse_dates` into a single array
    and pass that; and 3) call `date_parser` once for each row using one or
    more strings (corresponding to the columns defined by `parse_dates`) as
    arguments.
dayfirst : boolean, default False
    DD/MM format dates, international and European format
iterator : boolean, default False
    Return TextFileReader object for iteration or getting chunks with
    ``get_chunk()``.
chunksize : int, default None
    Return TextFileReader object for iteration.
    See the `IO Tools docs
    <http://pandas.pydata.org/pandas-docs/stable/io.html#io-chunking>`_
    for more information on ``iterator`` and ``chunksize``.
compression : {'infer', 'gzip', 'bz2', 'zip', 'xz', None}, default 'infer'
    For on-the-fly decompression of on-disk data. If 'infer' and
    `filepath_or_buffer` is path-like, then detect compression from the
    following extensions: '.gz', '.bz2', '.zip', or '.xz' (otherwise no
    decompression). If using 'zip', the ZIP file must contain only one data
```

```
    file to be read in. Set to None for no decompression.

    .. versionadded:: 0.18.1 support for 'zip' and 'xz' compression.

thousands : str, default None
    Thousands separator
decimal : str, default '.'
    Character to recognize as decimal point (e.g. use ',' for European data).
float_precision : string, default None
    Specifies which converter the C engine should use for floating-point
    values. The options are `None` for the ordinary converter,
    `high` for the high-precision converter, and `round_trip` for the
    round-trip converter.
lineterminator : str (length 1), default None
    Character to break file into lines. Only valid with C parser.
quotechar : str (length 1), optional
    The character used to denote the start and end of a quoted item. Quoted
    items can include the delimiter and it will be ignored.
quoting : int or csv.QUOTE_* instance, default 0
    Control field quoting behavior per ``csv.QUOTE_*`` constants. Use one of
    QUOTE_MINIMAL (0), QUOTE_ALL (1), QUOTE_NONNUMERIC (2) or QUOTE_NONE (3).
doublequote : boolean, default ``True``
   When quotechar is specified and quoting is not ``QUOTE_NONE``, indicate
   whether or not to interpret two consecutive quotechar elements INSIDE a
   field as a single ``quotechar`` element.
escapechar : str (length 1), default None
    One-character string used to escape delimiter when quoting is QUOTE_NONE.
comment : str, default None
    Indicates remainder of line should not be parsed. If found at the beginning
    of a line, the line will be ignored altogether. This parameter must be a
    single character. Like empty lines (as long as ``skip_blank_lines=True``),
    fully commented lines are ignored by the parameter `header` but not by
    `skiprows`. For example, if comment='#', parsing '#empty\na,b,c\n1,2,3'
    with `header=0` will result in 'a,b,c' being
    treated as the header.
encoding : str, default None
    Encoding to use for UTF when reading/writing (ex. 'utf-8'). `List of Python
    standard encodings
    <https://docs.python.org/3/library/codecs.html#standard-encodings>`_
dialect : str or csv.Dialect instance, default None
    If provided, this parameter will override values (default or not) for the
    following parameters: `delimiter`, `doublequote`, `escapechar`,
    `skipinitialspace`, `quotechar`, and `quoting`. If it is necessary to
    override values, a ParserWarning will be issued. See csv.Dialect
    documentation for more details.
tupleize_cols : boolean, default False
    .. deprecated:: 0.21.0
       This argument will be removed and will always convert to MultiIndex
```

```
    Leave a list of tuples on columns as is (default is to convert to
    a MultiIndex on the columns)
error_bad_lines : boolean, default True
    Lines with too many fields (e.g. a csv line with too many commas) will by
    default cause an exception to be raised, and no DataFrame will be returned.
    If False, then these "bad lines" will dropped from the DataFrame that is
    returned.
warn_bad_lines : boolean, default True
    If error_bad_lines is False, and warn_bad_lines is True, a warning for each
    "bad line" will be output.
low_memory : boolean, default True
    Internally process the file in chunks, resulting in lower memory use
    while parsing, but possibly mixed type inference.  To ensure no mixed
    types either set False, or specify the type with the `dtype` parameter.
    Note that the entire file is read into a single DataFrame regardless,
    use the `chunksize` or `iterator` parameter to return the data in chunks.
    (Only valid with C parser)
buffer_lines : int, default None
    .. deprecated:: 0.19.0
       This argument is not respected by the parser
compact_ints : boolean, default False
    .. deprecated:: 0.19.0
       Argument moved to ``pd.to_numeric``

    If compact_ints is True, then for any column that is of integer dtype,
    the parser will attempt to cast it as the smallest integer dtype possible,
    either signed or unsigned depending on the specification from the
    `use_unsigned` parameter.
use_unsigned : boolean, default False
    .. deprecated:: 0.19.0
       Argument moved to ``pd.to_numeric``

    If integer columns are being compacted (i.e. `compact_ints=True`), specify
    whether the column should be compacted to the smallest signed or unsigned
    integer dtype.
memory_map : boolean, default False
    If a filepath is provided for `filepath_or_buffer`, map the file object
    directly onto memory and access the data directly from there. Using this
    option can improve performance because there is no longer any I/O overhead.

Returns
-------
result : DataFrame or TextParser
```

In [27]:
```
countries = pandas.read_csv("https://raw.githubusercontent.com/cs109/2014_data/master/c
```

```
countries
```

```
Out[27]: 0     Algeria
         1      Angola
         2       Benin
         3    Botswana
         4     Burkina
         Name: Country, dtype: object
```

```
In [ ]: #subset column 'fuel consumption'
        fuel = cars['fuel.consumption']

        #get first 10 rows of fuel consumption data
        fuel[0:10]
```

2.3 We can also render HTML and other files directly in Jupyter Notebook!

Use the YouTube embedded link as the source when inserting videos.

```
In [13]: %%HTML
         <iframe width="420" height="345" src="https://youtube.com/embed/q_BzsPxwLOE">
         </iframe>
```

```
<IPython.core.display.HTML object>
```

Finally, download your files in HTML format to use on your blog, or export to PDF for printable report.