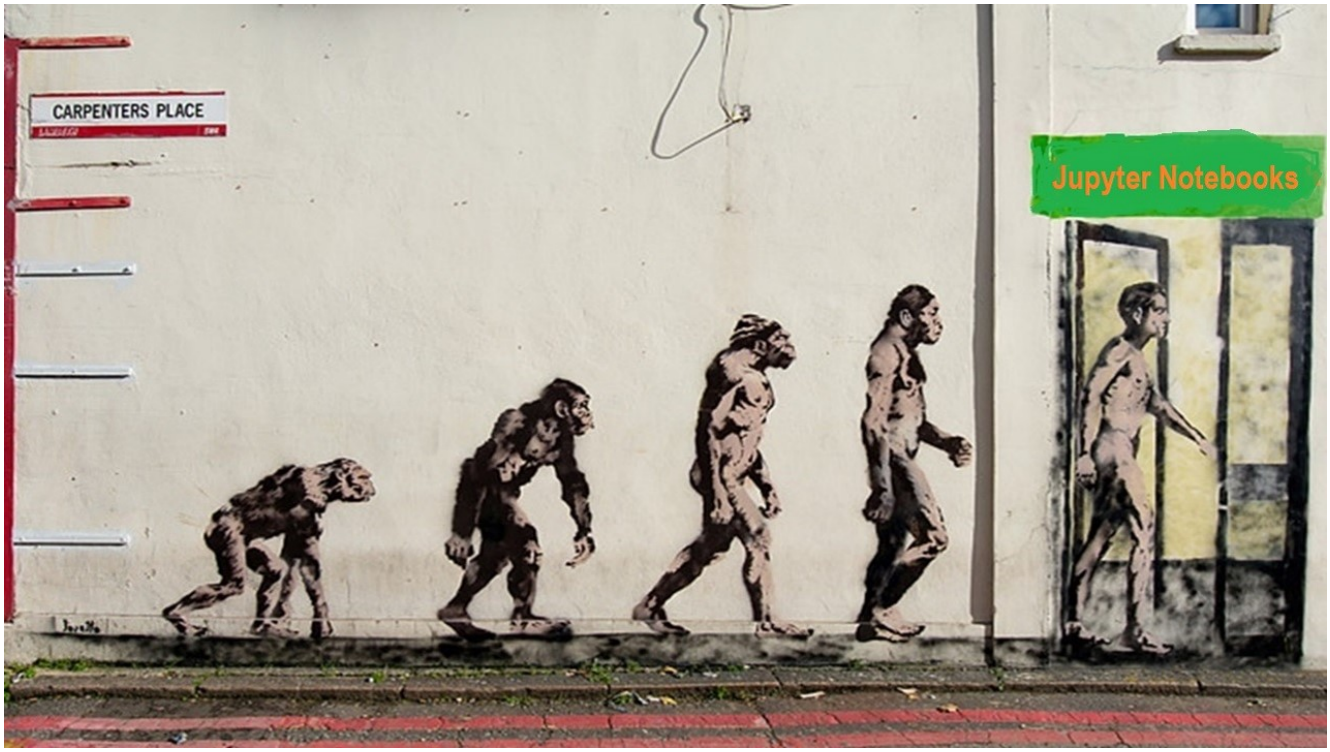
Jupyter Notebooks for Social Research



Authors

Professor Vernon Gayle, University of Edinburgh (UK)

Dr Roxanne Connelly, University of York (UK)

The work is very exploratory.

Positive comments are always appreciated, but brickbats improve work.

or @profbigvern
or @ConnellyRoxanne

Next Actions: Demonstrate at Univesity of Luxembourg Meeting Oct 2018

Author: Professor Vernon Gayle

Project: Reproducible Research with Social Science Data

Sub-project: Jupyter Notebooks / Social Science Data Analysis with Python

Latest Update: 30th September 2018 Stirling Scotland

Previous Updates: 25th January 2017 University of Edinburgh 26th January 2017 Demonstrated at Q-Step Edinburgh Meeting 24th January 2017 (back on Scotrail - delay at Linlithgow) 23rd January 2017 (on Scotrail) 22nd January 2017 University of Edinburgh

---

My General Motivation

Can we use Jupyter Notebooks to support reproducible social science research?

Are there insights we can gain from computer science & e-Research?

Are there insights from Open Science?

Overview

The Jupyter Notebook is an open source web application that facilitates the creation and sharing of documents that contain live code and supporting commentary in the form of explanatory text.

It is platform that can be used throughout the research process to organise and articulate elements of the social science workflow.

The Jupyter Notebook is open source and currently supports interactive data science and scientific computing across over 40 programming languages.

Obvious uses include data enabling and management, statistical analyses, simulations, and machine learning, but it is sufficiently flexible to be used more widely.

Jupyter Notebooks & Big Science

The detection of gravitational waves is heralded as a major scientific discovery.

Here is a video of Fernado Perez demonstrating a Jupyter Notebook that includes data and analyses of the first observed gravitational waves which are from the LIGO collaborations.

click on picture to run video

For more information on how this important work was rendered openly accessible by the LIGO collaboration see https://www.gw-openscience.org/tutorials/ .

Counting Galaxies in the Hubble Deep Field

This example uses the image-processing library scikit-image to identify galaxies in an image of the sky provided by the Hubble Space Telescope using a blob feature detection algorithm (an approach known as the determinant of Hessian).

After running the cell, we can explore the parameters of the detection algorithm to find galaxies of different sizes and prominences:

```
The max_sigma parameter determines the maximum size of the objects that will be identified

The threshold parameter can be reduced to detect less prominent objects
```

This example was provided in a live example hosted by the journal Nature

http://www.nature.com/news/ipython-interactive-demo-7.21492?article=1.16261

The live demo is no longer running (see below) but we can run it here.

A note from their site

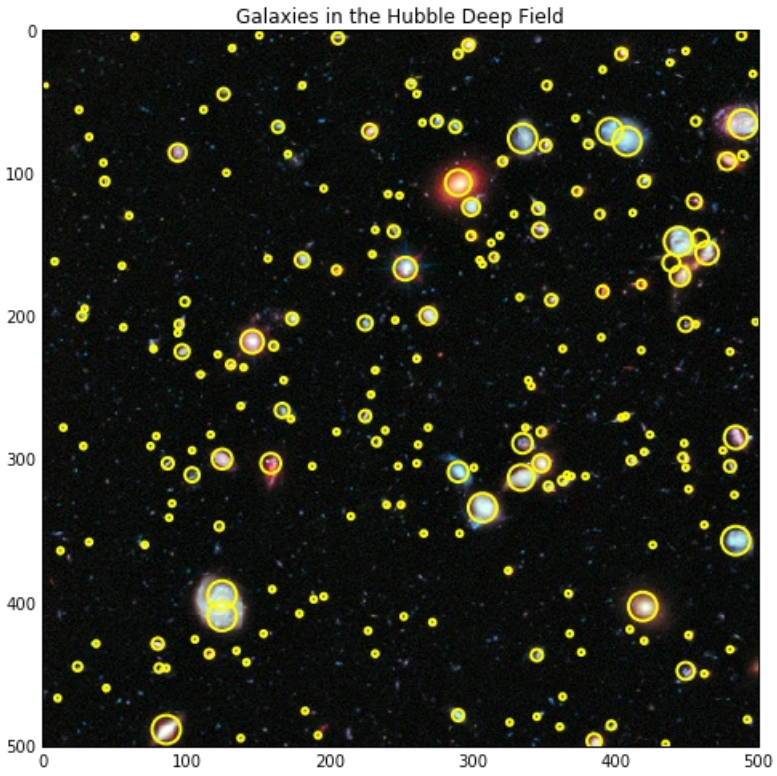
Attention! This is no longer a live demo. What follows is the original text from the live Nature demo that ran from November 2014 to March 2017. For up to date installation and usage of the Jupyter Notebook (née IPython notebook), visit jupyter.org

---

In [1]:

```
# Import matplotlib (plotting), skimage (image processing) and interact (user interfaces)
# This enables their use in the Notebook.
%matplotlib inline
from matplotlib import pyplot as plt

from skimage import data
from skimage.feature import blob_doh
from skimage.color import rgb2gray

from IPython.html.widgets import interact, fixed

# Extract the first 500px square of the Hubble Deep Field.
image = data.hubble_deep_field()[0:500, 0:500]
image_gray = rgb2gray(image)

def plot_blobs(max_sigma=30, threshold=0.1, gray=False):
    """
    Plot the image and the blobs that have been found.
    """
    blobs = blob_doh(image_gray, max_sigma=max_sigma, threshold=threshold)

    fig, ax = plt.subplots(figsize=(8,8))
    ax.set_title('Galaxies in the Hubble Deep Field')

    if gray:
        ax.imshow(image_gray, interpolation='nearest', cmap='gray_r')
        circle_color = 'red'
    else:
        ax.imshow(image, interpolation='nearest')
        circle_color = 'yellow'
    for blob in blobs:
```

```
    for blob in blobs:
        y, x, r = blob
        c = plt.Circle((x, y), r, color=circle_color, linewidth=2, fill=False)
        ax.add_patch(c)

# Use interact to explore the galaxy detection algorithm.
interact(plot_blobs, max_sigma=(10, 40, 2), threshold=(0.005, 0.02, 0.001));

# The max_sigma parameter determines the maximum size of the objects that will be identified

# The threshold parameter can be reduced to detect less prominent objects
```



Literate Computing

Fernando Perez says

Literate Computing is the weaving of a narrative directly into a live computation, interleaving text with code and results to construct a complete piece that relies equally on the textual explanations and the computational components, for the goals of communicating results in scientific computing and data analysis.

http://blog.fperez.org/

Why is it call Jupyter?

The computer languages Julia Python and R almost spell *JuPyteR*

The original authors Fernando Perez and Brian Granger are physicists.

Galileo's notebooks are an inportant early milestone (or deliverable) in open science.

Galileo's notes directly integrated his data (drawings of Jupiter and its moons), key metadata (timing of each observation, weather, telescope properties), and text (descriptions of methods, analysis, and conclusions) see https://www.authorea.com/users/3/articles/6316/_show_article .

Two obvious examples are *Sidereus Nuncius* and *la vacchetta* (which literally means the small cow or calf, because of the leather of its binding).

Translations of these notebooks can be viewed here

http://www.dioi.org/galileo/galileo.htm

http://homepages.wmich.edu/~mcgrew/Siderius.pdf

Three Galilean moons of Jupiter are visible on the logo.

Some people may have come accross IPython Notebooks which are from an earlier stage in the evolutionary process.

Jupyter Notebooks Social Science Research

Jupyter Notebooks have a number of attractive features

1. Easy documentation alongside research code
2. 'Language agnostic' 40+ languages
3. Rich visual outputs
4. Big data tools e.g. python
5. Teaching and training
6. Collaborative work
7. Portability (publication) easy to share

These features are demonstrated in this Jupyter Notebook.

Structure of a Jupyter Notebook

A Jupyter Notebook is made up of cells.

A cell can contain either

i. live research code (e.g. Stata syntax) that can be executed

ii. text comments that form the documentation of the research workflow

iii. cells that contain the results of data analyses

In addition to running your code the Notebook frontend stores code and output, together with markdown notes, in an editable document called a notebook. When you save it, this is sent from your browser to the notebook server, which saves it on disk as a JSON file with a .ipynb extension.

Documentation in the Jupyter Notebook

Writing comments and documenting research code in a Jupyter Notebook is relatively easy.

Notebooks use *Markdown*, which is explained below.

Comments can also be included within research code. For example a comment within Stata code can be written by starting with the familiar asterisk or star symbol.

Markdown

Markdown is an easy way to write documents.

It is written in what computer geeks like to call 'plaintext'. It is the sort of text that we are used to writing and seeing.

Plaintext is just the regular alphabet plus a few other familiar symbols (for example the asterisk *).

Plaintext is just the regular alphabet plus a few other familiar symbols (for example the asterisk *).

Unlike cumbersome word processing applications, text written in Markdown can be easily shared between computers.

It's quickly becoming the writing standard in some academic areas and in science.

Websites like GitHub and reddit use Markdown to style their comments.

Here is a summary of *Markdown* codes https://en.wikipedia.org/wiki/Markdown#Example .

If you have half an hour you can learn *Markdown* here http://www.markdowntutorial.com/ (you might need to use Internet Explorer).

Examples

Example 1 'Sitting in the Swivel Chair' (undertaking data analyses in different languages)

Example 2 'Rich Visual Output'

Example 3 'Widgets'

Example 4 'Using Magic Cells'

Example 5 'A Gnarly Example in Stata with a Complex Model'

Example 1 'Sitting in the Swivel Chair'

The Notebook can work with over 40 programming languages.

This includes data analysis software such as Stata and R and languages such as Python that are popular in some areas of data science.

Moving between languages with ease has been likened to sitting in an office chair and swivelling between three different computers.

The example below demonstrates some social science data analysis (estimating a logistic regression model) within the Jupyter Notebook.

To illustrate that the Jupyter Notebook is 'language agnostic', the analysis is undertaken using Stata, R and Python.

Jupyter Notebooks use different 'kernels' to work in different languages.

Stata Analysis

Stata is a proprietary software package and you MUST have access to the software to run it within a Jupyter Notebook.

Here is a link to the Stata Corporation http://www.stata.com/ .

This is an example of running Stata from within the Jupyter Notebook.

The live research code in each cell is Stata syntax.

Comments are written in cells using markdown.

WARNING

We would usually use a Stata Kernel to run Stata.

We would usually use a Stata Kernel to run Stata.

In this current example we will use MAGIC - which will be explained later in this notebook.

This is because some users with Windows 10 machines have experienced problems installing the Stata Kernel.

MAGIC cells are explained later in the notebook.

In [3]:

```
import ipystata
```

The cell above imports ipystata to run Stata

see http://dev-ii-seminar.readthedocs.io/en/latest/notebooks/Stata_in_jupyter.html

---

The cell below imports the Stata data file from my website vernongayle.com.

%%stata
makes the whole cell a Stata cell

-o wemp_df
slots the data into a data frame - remember we are working in Python here

use "http://www.vernongayle.com/uploads/2/2/3/0/22304498/wemp.dta", clear codebook, compact
Is the usual Stata code for using a dataset and producing a compact codebook

In [4]:

```
%%stata -o wemp_df
use "http://www.vernongayle.com/uploads/2/2/3/0/22304498/wemp.dta", clear
codebook, compact
```

```
Variable     Obs Unique      Mean  Min   Max  Label
-------------------------------------------------------------------------------
case        1580    155  517.7411    1  1003
femp        1580      2  .6455696    0     1
mune        1580      2  .0740506    0     1
time        1580     14       7.2    0    13
und1        1580      2  .0746835    0     1
und5        1580      2  .2974684    0     1
age         1580     43  36.01013   18    60
-------------------------------------------------------------------------------
```

The data mirror a real example of data analysed in Davies et al. (1992).

The dataset is a panel of 155 married women.

Davies, Richard B., Peter Elias, and Roger Penn. "The relationship between a husband's unemployment and his wife's participation in the labour force." Oxford Bulletin of Economics and Statistics 54.2 (1992): 145-171.

Estimating a simple logit (logistic regression) model using Stata

The outcome variable is femp (female unemployed 0 or 1)

The explanatory variables are mune (husband unemployed 0,1) und5 (couple have a child under age 5)

In [5]:

```
%%stata -o wemp_df
logit femp mune und5
```

```
Iteration 0:   log likelihood = -1027.2309
Iteration 1:   log likelihood = -879.88806
Iteration 2:   log likelihood = -878.68101
Iteration 3:   log likelihood = -878.67998
```

```
Iteration 4:   log likelihood = -878.67998

Logistic regression                               Number of obs     =      1,580
                                                  LR chi2(2)        =     297.10
                                                  Prob > chi2       =     0.0000
Log likelihood = -878.67998                       Pseudo R2         =     0.1446

------------------------------------------------------------------------------
       femp |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
------------+-----------------------------------------------------------------
       mune |  -1.703308   .2358489    -7.22   0.000    -2.165563   -1.241053
       und5 |  -1.733521   .1221909   -14.19   0.000    -1.973011   -1.494031
      _cons |   1.306829   .0744154    17.56   0.000     1.160978    1.452681
------------------------------------------------------------------------------
```

R Analysis

WARNING

You must have R installed on your machine.

You must have installed the R kernel (See https://anaconda.org/r/r-irkernel).

You must have installed the R libraries foreign and survey

for example run the code *install.packages("foreign","survey")*

(see https://cran.r-project.org/web/packages/survey/index.html ; https://cran.r-project.org/web/packages/foreign/index.html).

Reminder *Switch Kernel to R < Menu kernel - change kernel>*

Getting the libraries for R

In [1]:

```
library(foreign)
library(survey)
```

```
Loading required package: grid
Loading required package: Matrix
Loading required package: survival
Warning message:
: package 'survival' was built under R version 3.2.5
Attaching package: 'survey'

The following object is masked from 'package:graphics':

    dotchart
```

If you have an error message here it is possibly because you have not switched to the _R Kernel_

Getting the Stata data file (.dta) into an object called "mydata" then summarizing the dataset

In [2]:

```
mydata <- read.dta("http://www.vernongayle.com/uploads/2/2/3/0/22304498/wemp.dta")
summary(mydata)
```

Out[2]:

```
      case             femp             mune              time       
 Min.   :   1.0   Min.   :0.0000   Min.   :0.00000   Min.   : 0.0  
 1st Qu.: 274.0   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.: 4.0  
 Median : 538.0   Median :1.0000   Median :0.00000   Median : 8.0  
 Mean   : 517.7   Mean   :0.6456   Mean   :0.07405   Mean   : 7.2  
 3rd Qu.: 753.0   3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:11.0  
 Max.   :1003.0   Max.   :1.0000   Max.   :1.00000   Max.   :13.0  
      und1             und5              age       
```

```
      und1              und5              age       
 Min.   :0.00000   Min.   :0.0000   Min.   :18.00  
 1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:29.00  
 Median :0.00000   Median :0.0000   Median :35.00  
 Mean   :0.07468   Mean   :0.2975   Mean   :36.01  
 3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:43.00  
 Max.   :1.00000   Max.   :1.0000   Max.   :60.00  
```

Estimating the logit model and sending it to the object "mylogit".

In [3]:

```
str(mydata)
```

```
'data.frame': 1580 obs. of  7 variables:
 $ case: num  1 1 1 1 6 6 6 6 6 6 ...
 $ femp: num  1 0 0 0 1 1 0 0 1 1 ...
 $ mune: num  0 0 0 0 0 0 0 0 0 0 ...
 $ time: num  10 11 12 13 0 1 2 3 4 5 ...
 $ und1: num  1 0 0 0 0 0 0 0 0 0 ...
 $ und5: num  1 1 1 1 0 0 0 0 0 0 ...
 $ age : num  23 24 25 26 42 43 44 45 46 47 ...
 - attr(*, "datalabel")= chr ""
 - attr(*, "time.stamp")= chr "22 Jun 2006 18:18"
 - attr(*, "formats")= chr  "%9.0g" "%9.0g" "%9.0g" "%9.0g" ...
 - attr(*, "types")= int  254 254 254 254 254 254 254
 - attr(*, "val.labels")= chr  "" "" "" "" ...
 - attr(*, "var.labels")= chr  "" "" "" "" ...
 - attr(*, "version")= int 8
```

In [4]:

```
mylogit <- glm(femp ~ mune + und5, data = mydata, family = "binomial")

summary(mylogit)
```

Out[4]:

```
Call:
glm(formula = femp ~ mune + und5, family = "binomial", data = mydata)

Deviance Residuals:
    Min       1Q   Median       3Q      Max  
-1.7586  -1.0024   0.6922   0.6922   2.1177  

Coefficients:
            Estimate Std. Error z value Pr(>|z|)    
(Intercept)  1.30683    0.07442  17.561  < 2e-16 ***
mune        -1.70331    0.23585  -7.222 5.12e-13 ***
und5        -1.73352    0.12219 -14.187  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2054.5  on 1579  degrees of freedom
Residual deviance: 1757.4  on 1577  degrees of freedom
AIC: 1763.4

Number of Fisher Scoring iterations: 4
```

The R results are the same as the Stata results - thankfully

Python Analysis

WARNING *Switch Kernel to Python < Menu kernel - change kernel>*

Switch Kernel to Python 3 < Menu kernel - change kernel>

In [1]:

```
import pandas as pd
```

Construct the data frame "df" reading in the data from an Excel (xlsx) file. It could also be read in from a csv file.

In [2]:

```
df = pd.read_excel("http://www.vernongayle.com/uploads/2/2/3/0/22304498/wemp.xlsx")
df.head()
```

Out[2]:

	case	femp	mune	time	und1	und5	age
0	1	1	0	10	1	1	23
1	1	0	0	11	0	1	24
2	1	0	0	12	0	1	25
3	1	0	0	13	0	1	26
4	6	1	0	0	0	0	42

Python is more general purpose and not primarily orientated towards social science data analysis. Therefore some things are a little more fiddly. For example we must set a constant for all case (int=1).

In [3]:

```
df['Int']=1
```

Examining the data in the data frame "df".

In [4]:

```
df.head()
```

Out[4]:

	case	femp	mune	time	und1	und5	age	Int
0	1	1	0	10	1	1	23	1
1	1	0	0	11	0	1	24	1
2	1	0	0	12	0	1	25	1
3	1	0	0	13	0	1	26	1
4	6	1	0	0	0	0	42	1

Import the package "statsmodels".

In [5]:

```
import statsmodels.api as sm
```

Estimate a logistic regression model the independent variables are "mune" "und5" and "int" the outcome variable is "femp".

In [6]:

```
independentVar = ['mune', 'und5', 'Int']
logReg = sm.Logit(df['femp'] , df[independentVar])
answer = logReg.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.556127
         Iterations 5
```

Iterations 5

The results are in the object "answer".

In [7]:

```
answer.summary()
```

Out[7]:

Logit Regression Results

Dep. Variable:	femp	**No. Observations:**	1580
Model:	Logit	**Df Residuals:**	1577
Method:	MLE	**Df Model:**	2
Date:	Sun, 30 Sep 2018	**Pseudo R-squ.:**	0.1446
Time:	14:33:26	**Log-Likelihood:**	-878.68
converged:	True	**LL-Null:**	-1027.2
		LLR p-value:	3.056e-65

	coef	**std err**	**z**	**P>|z|**	**[95.0% Conf. Int.]**
mune	-1.7033	0.236	-7.222	0.000	-2.166 -1.241
und5	-1.7335	0.122	-14.187	0.000	-1.973 -1.494
Int	1.3068	0.074	17.561	0.000	1.161 1.453

The result are the same in Python, R and Stata - Phew! ****

Summary

This example was designed to demonstrate that Jupyter Notebooks are language agnostic.

The language agnostic aspects of Jupyter Notebooks mean that they could be an appropriate unified environment in which to undertake research analyses using alternative software packages and languages.

This feature is especially attractive in some collaborative endeavours. For example although Stata is the primary data analysis software package in many sociology departments, from time to time there may be a requirement to use other data analysis tools.

Example 2 'Rich Visual Output'

There are several visual features of Jupyter Notebooks that make them attractive as environments in which to undertake analyses of social science data and which might also enrich teaching, training and kwonledge exchange and research capacity buidling activities.

Here is a smart wee example that shows the flexibility of graphing within the Jupyter Notebook.

This wee example runs in Python. You MUST have the Python kernel running.

Make sure you have the wemp data in the the "df" data frame.

In [8]:

```
df.head()
```

Out[8]:

	case	femp	mune	time	und1	und5	age	lnt
0	1	1	0	10	1	1	23	1
1	1	0	0	11	0	1	24	1
2	1	0	0	12	0	1	25	1
3	1	0	0	13	0	1	26	1
4	6	1	0	0	0	0	42	1

If you can't see the wemp data you need to go back up and reload it in the Python part of the Example 1.

We will now construct a new variable called *age2* which is the woman's age squared.

In [9]:

```
df['age2'] = df.age * df.age
```

This is an example of a graph in the style of a xkcd comic.

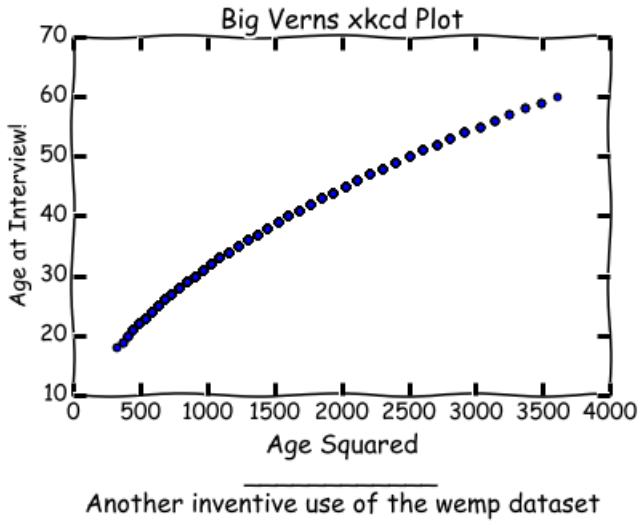
In [10]:

```
%pylab inline
plt.xkcd()  # Yes...
fig1 = plt.figure(1)
ax1 = fig1.add_subplot(111)
ax1.scatter(df['age2'], df['age'],)
plt.title('Big Verns xkcd Plot')
plt.ylabel('Age at Interview!')
plt.xlabel("Age Squared\n"
           "_____________\n"
            "Another inventive use of the wemp dataset", size=16)
```

```
Populating the interactive namespace from numpy and matplotlib
```

Out[10]:

```
<matplotlib.text.Text at 0x2aed724b00>
```



see http://nbviewer.jupyter.org/url/jakevdp.github.com/downloads/notebooks/XKCD_plots.ipynb

Inserting a picture into the notebook (within a markdown cell).

Donald Knuth

Inserting a video into the notebook (within a markdown cell).

In [11]:

```
from IPython.display import YouTubeVideo
YouTubeVideo('p47tetYy7co')
```

Out[11]:

To render cell contents as LaTeX.

In [12]:

```
%%latex
\begin{align}
a = \frac{1}{2} && b= \frac{1}{2} && c = \frac{1}{4} \\
\end{align}
```

$$a = \frac{1}{2} \qquad b = \frac{1}{2} \qquad c = \frac{1}{4}$$

In [13]:

```
%%latex
$e^{i\pi} + 1 = 0
$
```

$e^{i\pi} + 1 = 0$

Code can produce rich output such as images, videos, LaTeX, and JavaScript.

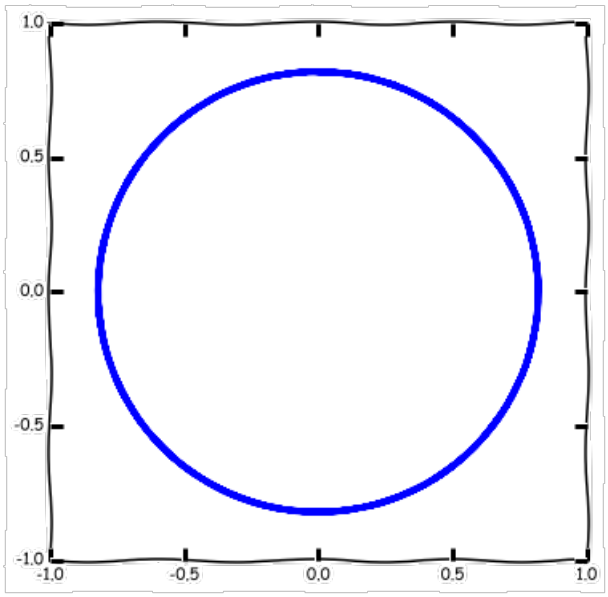
Example 3 'Widgets'

Widgets are clever devices that can be included in notebooks to help users visualize and control changes in the data. Widgets may be useful in teaching and training because users can easily see how changing inputs to something impacts on the results.

An Interesting Wee Widget

In [14]:

```
%matplotlib inline
import matplotlib.pyplot as plt
from numpy import pi, exp, real, imag, linspace
from ipywidgets import interact

def f(t,a=1,b=6,c=-14,d=0):
    return exp(a*1j*t) - exp(b*1j*t)/2 + 1j*exp(c*1j*t)/3 + exp(d*1j*t)/4

def plot_swirly(a=1,b=6,c=-14,d=0):
    t = linspace(0, 2*pi, 1000)
    ft = f(t,a,b,c,d)
    plt.plot(real(ft), imag(ft))

    # These two lines make the aspect ratio square
    fig = plt.gcf()
    fig.set_size_inches(6, 6, forward='True')

interact(plot_swirly,a=(-20,20),b=(-20,20),c=(-20,20),d=(-20,20));
```



The Sine Wave Example

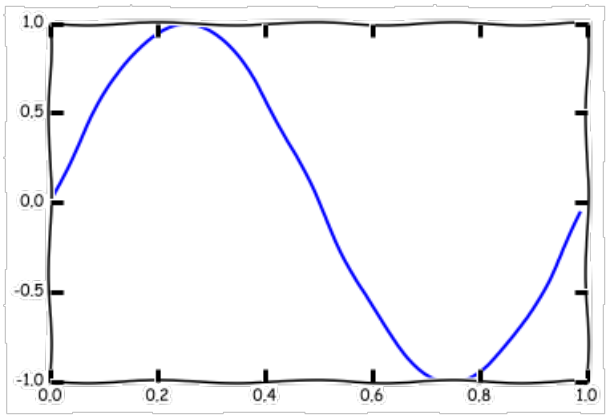
In [15]:

```
from ipywidgets import widgets
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display
from numpy import arange, sin, pi
%matplotlib inline
```

In [16]:

```
from IPython.html.widgets import *
t = arange(0.0, 1.0, 0.01)

def pltsin(f):
    plt.plot(t,sin(2*pi*t*f))
    plt.show()
```

```
interact(pltsin, f=(1,10,0.1))
```



Example 4 'Using Magic Cells'

Any cell that starts with a % symbol or two symbols %% is a 'magic' cell (honestly that is what they are called).

Here are a list of some of the standard 'magics'.

In [17]:

```
%lsmagic
```

Out[17]:

```
Available line magics:
%alias  %alias_magic  %autocall  %automagic  %autosave  %bookmark  %cd  %clear  %cls  %colors  %cor
fig  %connect_info  %copy  %ddir  %debug  %dhist  %dirs  %doctest_mode  %echo  %ed  %edit  %env  %g
ui  %hist  %history  %install_default_config  %install_ext  %install_profiles  %killbgscripts  %ldi
r  %less  %load  %load_ext  %loadpy  %logoff  %logon  %logstart  %logstate  %logstop  %ls  %lsmagic
%macro  %magic  %matplotlib  %mkdir  %more  %notebook  %page  %pastebin  %pdb  %pdef  %pdoc  %pfile
%pinfo  %pinfo2  %popd  %pprint  %precision  %profile  %prun  %psearch  %psource  %pushd  %pwd  %py
cat  %pylab  %qtconsole  %quickref  %recall  %rehashx  %reload_ext  %ren  %rep  %rerun  %reset  %re
set_selective  %rmdir  %run  %save  %sc  %set_env  %store  %sx  %system  %tb  %time  %timeit  %unal
ias  %unload_ext  %who  %who_ls  %whos  %xdel  %xmode

Available cell magics:
%%!  %%HTML  %%SVG  %%bash  %%capture  %%cmd  %%debug  %%file  %%html  %%javascript  %%latex  %%per
l  %%prun  %%pypy  %%python  %%python2  %%python3  %%ruby  %%script  %%sh  %%svg  %%sx  %%system  %
%time  %%timeit  %%writefile

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

In Example 1 we used a non-standard magic to run Stata which was written by Ties de Kok

In [18]:

```
%time print ("How long does this cell take to run?")
```

```
How long does this cell take to run?
Wall time: 0 ns
```

Example 5 'A Really Gnarly Example in Stata with a Complex

Model

Multilevel mixed-effects generalized linear model in Stata (meglm)

Multilevel modeling of survey data is a little different from standard modeling in that weighted sampling can take place at multiple levels in the model, resulting in multiple sampling weights.

Most survey datasets, regardless of the design, contain one overall inclusion weight for each observation in the data. This weight reflects the inverse of the probability of ultimate selection (i.e. it factors in all levels of clustered sampling, corrections for noninclusion and oversampling, poststratification).

Rabe-Hesketh, S. and Skrondal, A., 2006. Multilevel modelling of complex survey data. Journal of the Royal Statistical Society: Series A (Statistics in Society), 169(4), pp.805-827.

Rabe-Hesketh and Skrondal (2006) analyzed data from the 2000 Programme for International Student Assessment (PISA) study on reading proficiency among 15-year-old American students, as performed by the Organisation for Economic Co-operation and Development (OECD).

In [20]:

```
import ipystata
```

In [32]:

```
%%stata -o pisa_df
use http://www.stata-press.com/data/r14/pisa2000, clear
```

```
(Programme for International Student Assessment (PISA) 2000 data)
```

In [33]:

```
%%stata -o pisa_df
codebook, compact
```

```
Variable      Obs Unique      Mean       Min       Max  Label
-------------------------------------------------------------------------------
female       2069      2  .5432576         0         1  1 if female
isei         2069     58   47.0435        16        90  International socio-...
w_fstuwt     2069    224  843.3753  130.3095  3858.783  Student-level weight
wnrschbw     2069    147  167.8744     15.41   3352.32  School-level weight
high_school  2069      2  .3412276         0         1  1 if highest level b...
college      2069      2   .578057         0         1  1 if highest level b...
one_for      2069      2  .0560657         0         1  1 if one parent fore...
both_for     2069      2   .117448         0         1  1 if both parents ar...
test_lang    2069      2  .9134848         0         1  1 if English (the te...
pass_read    2069      2  .3881102         0         1  1 if passed reading ...
id_school    2069    148  73.65974         1       151  School ID
-------------------------------------------------------------------------------
```

For student i in school j, where the variable id_school identifies the schools, the variable w_fstuwt is a student-level overall inclusion weight (wij , not wi|j) adjusted for non-inclusion and non-participation of students, and the variable wnrschbw is the school-level weight wj adjusted for oversampling of schools with more students from minority groups.

Rabe-Hesketh and Skrondal (2006) fit a two-level logistic model for passing a reading proficiency threshold. We will do the same using meglm, but first we must adjusted the weight variables that were used. We need adjustment scales for the first-level weights so that they sum to the effective sample size of their corresponding second-level cluster.

In [34]:

```
%%stata -o pisa_df
sort id_school
generate sqw = w_fstuwt * w_fstuwt
by id_school: egen sumw = sum(w_fstuwt)
by id_school: egen sumsqw = sum(sqw)
generate pst1s1 = w_fstuwt*sumw/sumsqw
```

The new variable pst1s1 holds the adjusted first-level weights.

Rabe-Hesketh and Skrondal (2006) also included the school mean socioeconomic index as a covariate in their analysis. We reproduce this new variable mn_isei.

In [36]:

```
%%stata -o pisa_df
by id_school: egen mn_isei = mean(isei)
```

Estimation a multi-level logistic regression model with complex survey data.

In [37]:

```
%%stata -o pisa_df
meglm pass_read female isei mn_isei high_school college test_lang one_for both_for [pw=pst1s1], fa
mily(bernoulli) link(logit)|| id_school:, pweight(wnrschbw)
```

```
> h_for [pw=pst1s1], family(bernoulli) link(logit)|| id_school:, pweight(wnrsch
> bw)

Fitting fixed-effects model:

Iteration 0:   log likelihood = -199789.29
Iteration 1:   log likelihood = -199575.17
Iteration 2:   log likelihood = -199574.45
Iteration 3:   log likelihood = -199574.45

Refining starting values:

Grid node 0:   log likelihood = -200169.61

Fitting full model:

Iteration 0:   log pseudolikelihood = -200169.61  (not concave)
Iteration 1:   log pseudolikelihood = -198196.62
Iteration 2:   log pseudolikelihood = -197524.63
Iteration 3:   log pseudolikelihood = -197398.31
Iteration 4:   log pseudolikelihood = -197395.98
Iteration 5:   log pseudolikelihood = -197395.98

Mixed-effects GLM                               Number of obs     =      2,069
Family:              Bernoulli
Link:                    logit
Group variable:      id_school                  Number of groups  =        148

                                                Obs per group:
                                                              min =          1
                                                              avg =       14.0
                                                              max =         28

Integration method: mvaghermite                 Integration pts.  =          7

                                                Wald chi2(8)      =      88.30
Log pseudolikelihood = -197395.98               Prob > chi2       =     0.0000
                          (Std. Err. adjusted for 148 clusters in id_school)
------------------------------------------------------------------------------
             |               Robust
   pass_read |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      female |   .6221369   .1540088     4.04   0.000     .3202852    .9239887
        isei |    .018215   .0048057     3.79   0.000     .0087959     .027634
     mn_isei |   .0682472   .0164337     4.15   0.000     .0360378    .1004566
 high_school |   .1028108    .477141     0.22   0.829    -.8323683     1.03799
     college |   .4531688   .5053447     0.90   0.370    -.5372885    1.443626
   test_lang |   .6251822   .3821182     1.64   0.102    -.1237557     1.37412
     one_for |  -.1089314   .2739724    -0.40   0.691    -.6459075    .4280447
    both_for |   .2804038   .3264681     0.86   0.390    -.3593696     .920177
```

```
    both_lor |  -.2804038   .3264681    -0.86   0.390    -.9202696     .359462
       _cons |  -5.877565    .954525    -6.16   0.000      -7.7484   -4.006731
-------------+------------------------------------------------------------------
id_school    |
   var(_cons)|   .2955769   .1243375                      .1295996    .6741201
------------------------------------------------------------------------------
```

Here we have specified a multi-level logit model (or a random effects logit model).

The survey data are complex.

Pupils are nested in schools and we are adjusting the model for both school-level and pupil-level weights.

1] We specified the level-one weights using standard Stata weight syntax [pw=pst1s1].

2] We specified the level-two weights via the pweight(wnrschbw) option as part of the random effects specification for the id_school level using the syntax || id_school:, pweight(wnrschbw). The weight wnrschbw needs to be constant within schools.

3] The standard errors in the presence of sampling weights are robust.

4] Robust standard errors are clustered at the top level of the model, and this will always be true unless you specify vce(cluster clustvar), where clustvar identifies an even higher level of grouping.

This illustration is drawn from Example 5 of the meglm section of Stata Manual (pp.81-82).

A Little Light Relief

My Jupyter Limerick

A researcher with time to fritter

Decided he didn't need Jupyter

His results he would show

Without a traceable workflow

Could a researcher be any stupider?

Remarks and Conclusion

Jupyter Notebooks and Social Research

Jupyter Notebooks have a number of attractive features

1. Easy documentation alongside research code
2. 'Language agnostic' 40+ languages
3. Rich visual outputs
4. Big data tools e.g. python
5. Teaching and training
6. Collaborative work
7. Portability (publication) easy to share

These features are demonstrated in this Jupyter Notebook.

Finally converting this Jupyter Notebook into Portable Formats

see http://nbconvert.readthedocs.io/en/latest/

1. At the cmd prompt *conda install nbconvert*
2. Change directory (for example my directory is *C:\Users\Vernon*

3. Type *jupyter nbconvert --to html mynotebook.ipynb*

The work is very exploratory.

Positive comments are always appreciated, but brickbats improve work.

or @profbigvern

or @ConnellyRoxanne

END OF FILE