Python Workshop

April 15, 2021

1 Python Coding Workshop

4/15/21

1.1 Overview

Graduate Student Instructor: Kayleigh Barnes

Email: kayleighnb@berkeley.edu

1.1.1 Goals for today

This session is intended to guide you through the practical implementation of basic analytic techniques in Python in Jupyter notebooks. Python is an open-source statistical computing software used to analyze data (among many, many other things). A Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. This workshop will be focused on interactive demonstration in Python, but also include time for additional questions and guidance in working through the sample code. We will cover some fundamental coding techniques that will help you in Econ 140, basic data science classes, or research assistant positions. This workshop is for *beginners* that have little or no coding experience.

1.1.2 Important notes

- One attendee from today's workshop will be randomly selected to win a 20 dollar gift card to Amazon
- Attendance to this workshop comes with free access to datacamp through July. Datacamp offers online courses in both R and Python so that you can continue learning after today's workshop
- Link to join Berkeley Econ's datacamp group with @berkeley.edu ID: here (make sure you're signed out of datacamp before clicking this - otherwise the sign-up gets screwed and you'll be asked to pay after the first chapter of any course)

1.2 Jupyter and Python Basics

- To create a new notebook, click the "New" button and select Python 3
- Write Python script by selecting the option "Code" from the dropdown list, or write text by selecting "Markdown"
- Select "Insert" to add a block of text or code
- Run code by highlighting and selecting "Run"
- Use the # symbol to add comments to the script, or to add headlines to text selections
- To clear your coding output, select Cell=>All Output=>Clear

User written open-source libraries are needed for specific functionality in python (e.g. nice graphics, data analysis). However, we need to manually install these libraries (once) and load them at the beginning of every script. Libraries have been pre-installed in Jupyter notebooks. If you are wondering why a command you've used before is no longer working, it may be because you haven't loaded the library.

[44]:
```
import numpy as np #Numeric Python, allows us to sort and index data mong many␣
 ↪many other things (next two packages use it as a base)
import scipy as sp #Scientific Python, similar to numpy but with more linear␣
 ↪algebra capabilities
from scipy import stats
import pandas as pd #this is the library that enables interacting with data␣
 ↪very important!
import os #Operating system, will use to manage working directory
import matplotlib.pyplot as plt #for making nice graphs
plt.style.use('seaborn-whitegrid')
```

[45]:
```
# The help function, using help() before a command will bring up information on␣
 ↪what the command does
help(print)
```

```
Help on built-in function print in module builtins:

print(…)
    print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

[46]:
```
#The working directory is the location that R will look for data in
    # this is the same as telling your computer to look in a documents folder␣
 ↪when uploading soemthing
os.getcwd()
```

```
#os.chdir('/home/jovyan/my-work') #remove the first # from this line to run␣
 ↪code that changes the working directory
```

[46]:
```
'/home/jovyan/my-work'
```

1.3 Loading in data and summary statistics

Now let's load in the data set. Make sure you have uploaded the data to Jupyter before running the next line of code. We are going to use data on a set of households in Mexico in the 1990's. The data includes a village ID, a household ID, and demogrpahic variables like income, household size, age and gender of the head of household and a poverty indicator.

[47]:
```
MyFirstData = pd.read_csv('Data/MyFirstData.csv')
```

Notice that there is no ouput from the code that reads in the data. Unlike excel, R stores the data in the background and we need to use specific comands to interact with it. Once it's read in, we can use several commands to describe the data.

[48]:
```
# Information about the structure of the data
MyFirstData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   villid     1200 non-null   int64
 1   hogid      1200 non-null   object
 2   D_HH       1183 non-null   float64
 3   IncomeLab  176 non-null    float64
 4   famsize    1200 non-null   int64
 5   agehead    1199 non-null   float64
 6   sexhead    1200 non-null   object
 7   pov_HH     1199 non-null   object
dtypes: float64(3), int64(2), object(3)
memory usage: 75.1+ KB
```

[49]:
```
# summary statistics for the data
MyFirstData.describe(include='all')
```

[49]:

	villid	hogid	D_HH	IncomeLab	famsize	\
count	1.200000e+03	1200	1183.000000	176.000000	1200.000000	
unique	NaN	1200	NaN	NaN	NaN	
top	NaN	MQ3991411149	NaN	NaN	NaN	
freq	NaN	1	NaN	NaN	NaN	
mean	5.951112e+06	NaN	0.810651	2242.840909	4.800833	
std	2.291954e+06	NaN	0.391951	2386.681593	2.236142	

```
min     1.001106e+06           NaN     0.000000    160.000000     1.000000
25%     7.011004e+06           NaN     1.000000   1200.000000     3.000000
50%     7.011019e+06           NaN     1.000000   1550.000000     5.000000
75%     7.015003e+06           NaN     1.000000   2800.000000     6.000000
max     7.015038e+06           NaN     1.000000  27000.000000    16.000000

            agehead sexhead pov_HH
count   1199.000000    1200   1199
unique          NaN       2      2
top             NaN    Male  pobre
freq            NaN     943    996
mean      49.124270     NaN    NaN
std       15.578989     NaN    NaN
min       16.000000     NaN    NaN
25%       37.000000     NaN    NaN
50%       47.000000     NaN    NaN
75%       60.000000     NaN    NaN
max       96.000000     NaN    NaN
```

[50]:
```
# print the names of the columns of the data
MyFirstData.columns
```

[50]:
```
Index(['villid', 'hogid', 'D_HH', 'IncomeLab', 'famsize', 'agehead', 'sexhead',
       'pov_HH'],
      dtype='object')
```

[51]:
```
# number of rows and number of columns
MyFirstData.shape
```

[51]:
```
(1200, 8)
```

[52]:
```
# first X rows of the data
MyFirstData.head(6)
```

[52]:
```
    villid           hogid  D_HH  IncomeLab  famsize  agehead sexhead  \
0  1001106  0101103050.0539   1.0        NaN        6     29.0  Female
1  1001106  0101103052.0540   1.0        NaN        6     43.0    Male
2  1001106  0101103054.0541   1.0        NaN        6     43.0    Male
3  1001106  0101103056.0542   1.0     3200.0        5     25.0  Female
4  1001106  0101103058.0543   1.0        NaN        5     40.0    Male
5  1001106  0101103060.0544   1.0     4320.0        5     40.0    Male

     pov_HH
0     pobre
1  no pobre
2  no pobre
3     pobre
```

```
4  no pobre
5  no pobre
```

[53]:
```
# display values and counts of categorical data
MyFirstData['sexhead'].value_counts()
```

[53]:
```
Male      943
Female    257
Name: sexhead, dtype: int64
```

1.4 Basic Data Cleaning and Formatting

1.4.1 Category Variable

Right now, we have two categorical variables: sexhead, which indicates the sex of the head of household and pov_HH, which indicates whether a household is below the poverty line. The data entries for these variables are text rather than numbers (we call these string variables in the data science world). Often when doing data analysis, it is easier to map categorical text variables to numbers, particularly 0 and 1. These variables that contain only 0's and 1's are called dummy variables.

Now, suppose we want to create a poor_male variable, which will be defined as 1 if the household is categorized as poor (pov_HH = pobre) and the head of the household is male (sexhead is Male), and 0 otherwise.

[54]:
```
# first, lets create dummy variables out of sexhead and pov_HH using the map␣
 ↪function
MyFirstData['sexhead_male'] = MyFirstData['sexhead'].map({'Male':1, 'Female':0})
MyFirstData['pov_HH_pobre'] = MyFirstData['pov_HH'].map({'pobre':1, 'no pobre':
 ↪0})

# compare this output to the output above to make sure it worked correctly
MyFirstData['sexhead_male'].value_counts()
```

[54]:
```
1    943
0    257
Name: sexhead_male, dtype: int64
```

[55]:
```
MyFirstData['poor_male']=MyFirstData['pov_HH_pobre']*MyFirstData['sexhead_male']
MyFirstData['poor_male'].value_counts()
```

[55]:
```
1.0    786
0.0    413
Name: poor_male, dtype: int64
```

1.4.2 Numerical Variable

We can use regular mathematical operations to create numerical variables from other variables.

[56]:
```
MyFirstData['agehead2'] = MyFirstData['agehead']**2
MyFirstData['agehead2'].describe()
```

[56]:
```
count    1199.000000
mean     2655.696414
std      1647.055309
min       256.000000
25%      1369.000000
50%      2209.000000
75%      3600.000000
max      9216.000000
Name: agehead2, dtype: float64
```

[57]:
```
MyFirstData['constant'] = 1
MyFirstData['constant'].describe()
```

[57]:
```
count    1200.0
mean        1.0
std         0.0
min         1.0
25%         1.0
50%         1.0
75%         1.0
max         1.0
Name: constant, dtype: float64
```

New Datasets We may also want to create a new data that summarizes the old, or is a subset of the original dataset.

[58]:
```
#Subset of only observations with male head of hh
data_males=MyFirstData.loc[MyFirstData['sexhead_male']==1]
data_males.describe(include='all')
```

[58]:
```
              villid         hogid        D_HH     IncomeLab     famsize  \
count   9.430000e+02           943  928.000000    136.000000  943.000000
unique           NaN           943         NaN           NaN         NaN
top              NaN  MQ3991411149         NaN           NaN         NaN
freq             NaN             1         NaN           NaN         NaN
mean    5.954876e+06           NaN    0.825431   2276.617647    5.022269
std     2.289004e+06           NaN    0.379803   2499.042252    2.217603
min     1.001106e+06           NaN    0.000000    160.000000    1.000000
25%     7.011004e+06           NaN    1.000000   1387.500000    4.000000
50%     7.011019e+06           NaN    1.000000   1600.000000    5.000000
75%     7.015003e+06           NaN    1.000000   2850.000000    6.000000
```

```
max     7.015038e+06           NaN    1.000000  27000.000000   16.000000

           agehead sexhead pov_HH  sexhead_male  pov_HH_pobre   poor_male  \
count   942.000000     943    942         943.0    942.000000  942.000000
unique         NaN       1      2           NaN           NaN         NaN
top            NaN    Male  pobre           NaN           NaN         NaN
freq           NaN     943    786           NaN           NaN         NaN
mean     49.049894     NaN    NaN           1.0      0.834395    0.834395
std      15.238684     NaN    NaN           0.0      0.371923    0.371923
min      18.000000     NaN    NaN           1.0      0.000000    0.000000
25%      36.250000     NaN    NaN           1.0      1.000000    1.000000
50%      47.000000     NaN    NaN           1.0      1.000000    1.000000
75%      60.000000     NaN    NaN           1.0      1.000000    1.000000
max      94.000000     NaN    NaN           1.0      1.000000    1.000000

           agehead2  constant
count    942.000000     943.0
unique          NaN       NaN
top             NaN       NaN
freq            NaN       NaN
mean    2637.863057       1.0
std     1605.688200       0.0
min      324.000000       1.0
25%     1314.250000       1.0
50%     2209.000000       1.0
75%     3600.000000       1.0
max     8836.000000       1.0
```

[59]:
```
meandata = MyFirstData.groupby('villid').agg({'IncomeLab':['mean'],
                            'famsize':['mean'],
                            'agehead':['mean']}).reset_index()
meandata.columns = ['villid', 'meanIncomeLab', 'meanfamsize', 'meanagehead']
meandata.describe(include='all')
```

[59]:
```
             villid  meanIncomeLab  meanfamsize  meanagehead
count  2.400000e+01      21.000000    24.000000    24.000000
mean   2.506884e+06    2595.022822     5.316248    45.506405
std    2.657087e+06     847.909016     0.648130     4.909733
min    1.001106e+06    1000.000000     4.503846    36.000000
25%    1.002032e+06    1960.000000     4.720771    41.026515
50%    1.008514e+06    2357.142857     5.348485    45.991667
75%    2.509100e+06    3200.000000     5.687500    50.170794
max    7.015038e+06    4350.000000     6.800000    52.266667
```

1.5 Making comparisons - T-Tests

A main goal of working with data is to make inferences about the population we are interested in. Much of Econ 140 will be focused on methods to make these inferences: What is the relationship between two variables? Did an experiment have a significant treatment effect?

If you have taken Stats 20, you are likely already familiar with a t-test. T-tests compare the difference in the means of a variable between two groups. The test statistic tells us whether the difference is *significant*, that is we can confidently say that the two groups are different.

[60]:
```
MyFirstData.groupby('pov_HH').mean()
```

[60]:
```
                villid      D_HH    IncomeLab   famsize    agehead  \
pov_HH
no pobre  5.118856e+06  0.783920  2610.714286  4.064039  45.389163
pobre     6.119674e+06  0.816887  2127.537313  4.950803  49.885542

          sexhead_male  pov_HH_pobre  poor_male     agehead2  constant
pov_HH
no pobre      0.768473           0.0   0.000000  2183.103448       1.0
pobre         0.789157           1.0   0.789157  2752.018072       1.0
```

[61]:
```
cat1 = MyFirstData[MyFirstData['pov_HH']=='pobre']
cat2 = MyFirstData[MyFirstData['pov_HH']=='no pobre']

stats.ttest_ind(cat1['famsize'], cat2['famsize'])
```

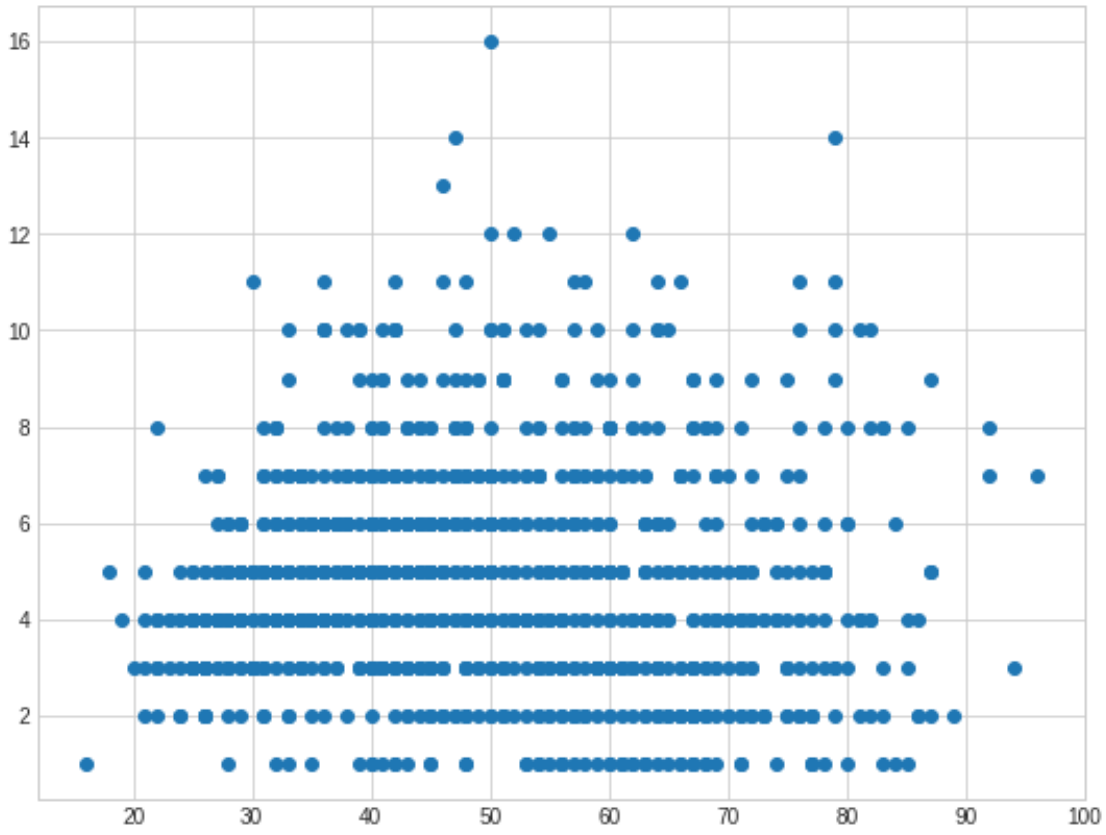
[61]:
```
Ttest_indResult(statistic=5.20321999319838, pvalue=2.3027229962108083e-07)
```

1.6 Visualizing Data

We will use the library matplotlib to make some graphs

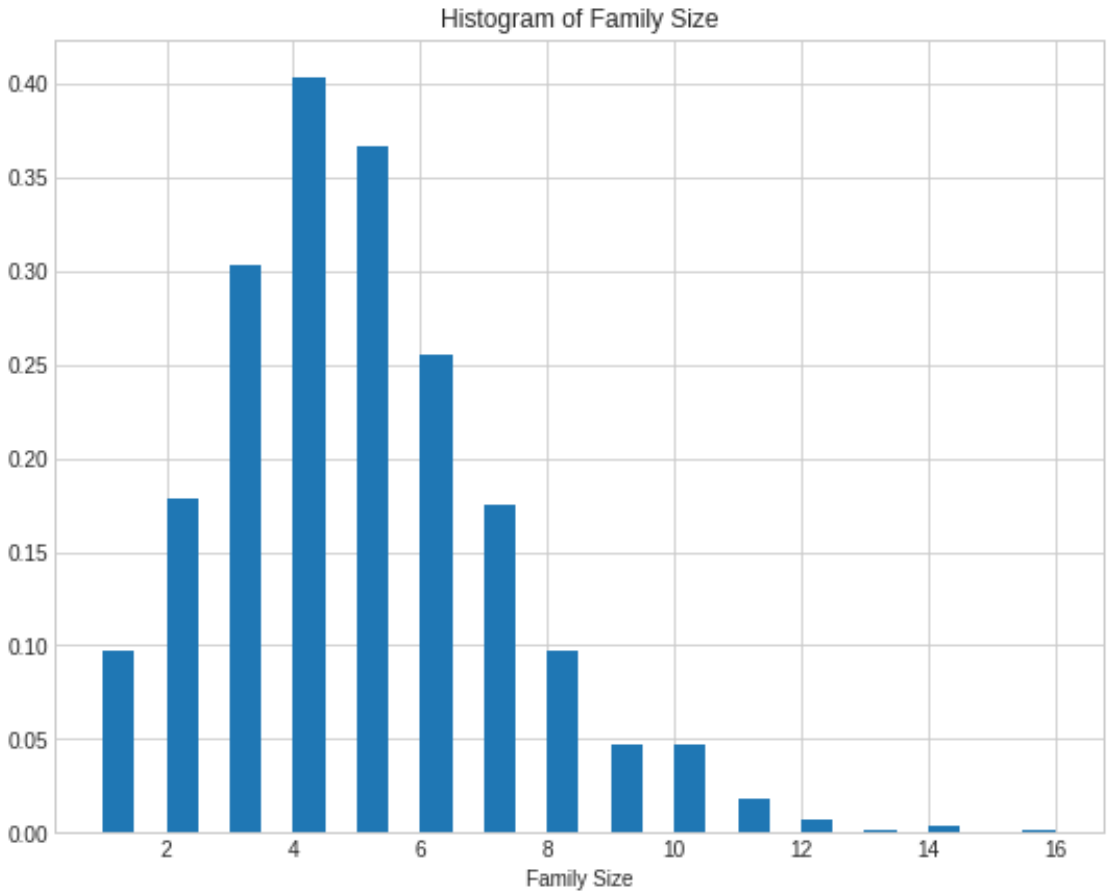
[62]:
```
plt.figure(figsize=(9,7))
plt.scatter(MyFirstData['agehead'], MyFirstData['famsize'])
```

[62]:
```
<matplotlib.collections.PathCollection at 0x7f910c0dcf50>
```



[63]:
```
plt.figure(figsize=(9,7))
plt.hist(MyFirstData['famsize'], density=True, bins=30)
plt.xlabel('Family Size')
plt.title('Histogram of Family Size')
```

[63]: Text(0.5, 1.0, 'Histogram of Family Size')


Histogram of Family Size
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
2
4
6
8
10
12
14
16
Family Size

[ ]: