# UCF STA4102 Lecture 3

Alexander V. Mantzaris

UCF | Department of Statistics

UNIVERSITY OF CENTRAL FLORIDA

# Overview

1. an interesting read
2. Running SAS
3. Running our first SAS programs
4. General SAS
5. DATA STEP example
6. data ex
    - LongleyData
7. DATA SCIENCE TO THE RESCUE!

# *PROBABILITY, STATISTICS AND TRUTH* by Richard von Mises (This was before the age of BIG DATA)

Considerable difficulties arise, however, when we are asked to give an exact explanation or even more, a definition of what we mean by 'probability'. The Latin term 'probabilis' was at one time translated by 'like truth' or 'with an appearance of truth'.

•'The probable is something which lies midway between truth and error' (Thomasius 1688).

•'That which, if it were held as truth, would be more than half certain, is called probable (Kant).

# current section

1. an interesting read
2. Running SAS
3. Running our first SAS programs
4. General SAS
5. DATA STEP example
6. data ex
   - LongleyData
7. DATA SCIENCE TO THE RESCUE!

# Starting SAS! *my.apps.ucf.edu*

I am interested now in trying SAS, how should I go about doing this?

There are a couple of ways to do this as a UCF student. There is the University SAS edition which you can download and install. It requires you to run VMware and from that environment work with SAS. I expect that you might encounter problems with this approach.

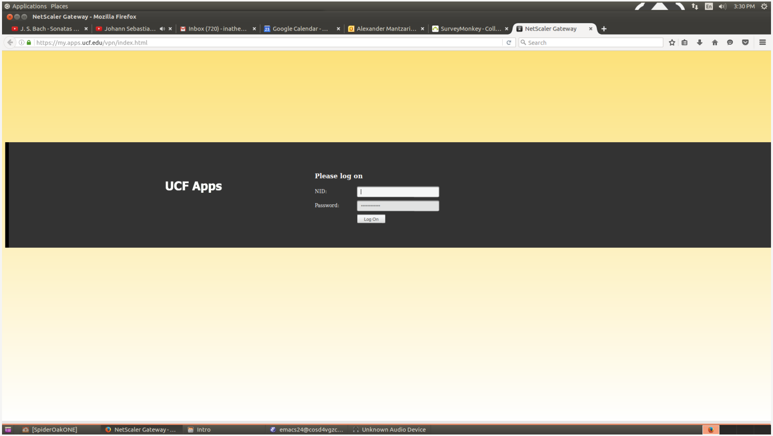
Well, I don't know what VMware is

If you haven't, it is not too challenging and the computer support can assist you through challenges, but I would recommend you use the web interface the university has for SAS:

my.apps.ucf.edu

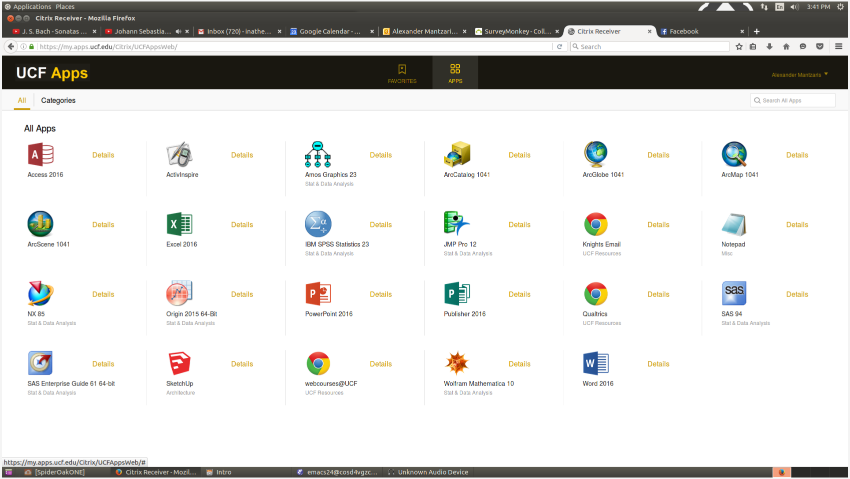
# Starting SAS! *my.apps.ucf.edu*

In your browser go to: my.apps.ucf.edu



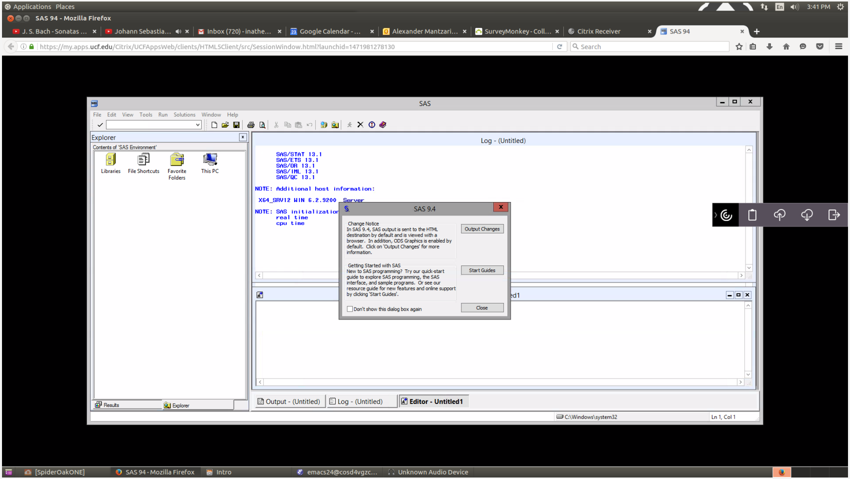
# Starting SAS! *my.apps.ucf.edu*

Click on SAS 94



# Starting SAS! *my.apps.ucf.edu*

Click close



```
      SAS/STAT 13.1
      SAS/ETS 13.1
      SAS/OR 13.1
      SAS/IML 13.1
      SAS/QC 13.1

NOTE: Additional host information:

 X64_SRV12 WIN 6.2.9200  Server

NOTE: SAS initializatior
      real time
      cpu time
```

SAS 9.4

Change Notice
In SAS 9.4, SAS output is sent to the HTML destination by default and is viewed with a browser. In addition, ODS Graphics is enabled by default. Click on 'Output Changes' for more information.

Output Changes

Getting Started with SAS
New to SAS programming? Try our quick-start guide to explore SAS programming, the SAS interface, and sample programs. Or see our resource guide for new features and online support by clicking 'Start Guides'.

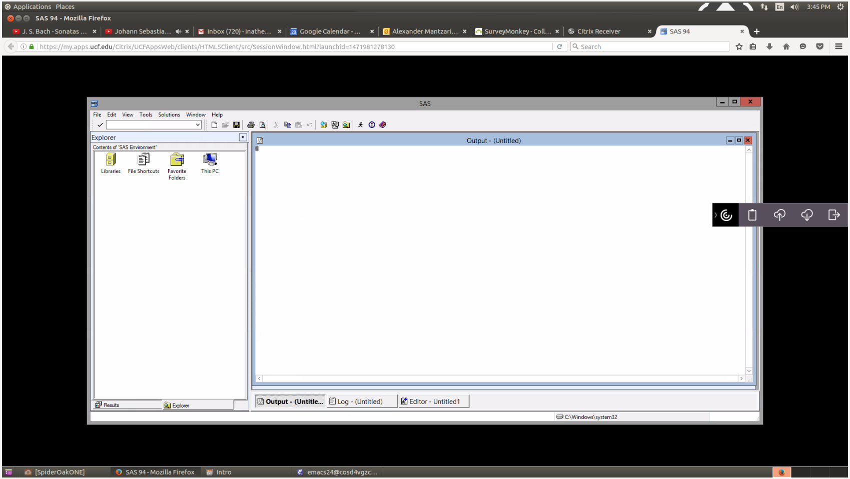
Start Guides

☐ Don't show this dialog box again

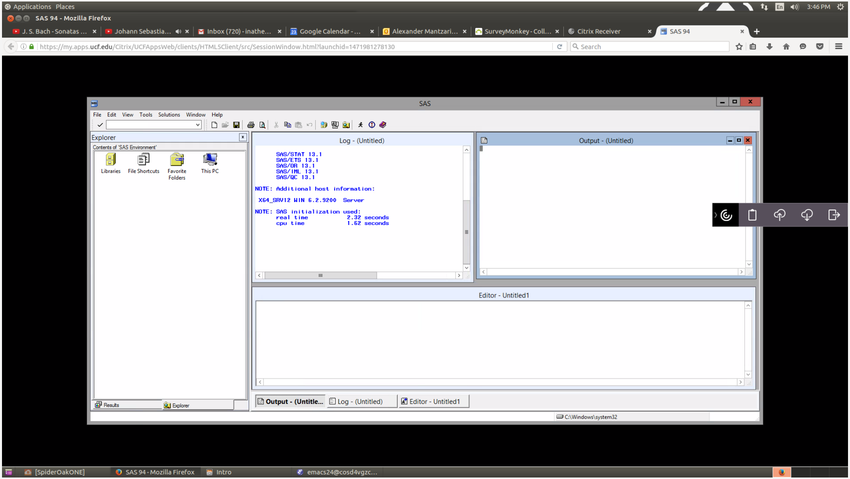
Close

# Starting SAS! *my.apps.ucf.edu*

Resize the windows to fit the 'Output window/log window/Editor window' inside



# Starting SAS! *my.apps.ucf.edu*



```
      SAS/STAT 13.1
      SAS/ETS 13.1
      SAS/OR 13.1
      SAS/IML 13.1
      SAS/QC 13.1

NOTE: Additional host information:

 X64_SRV12 WIN 6.2.9200  Server

NOTE: SAS initialization used:
      real time           2.32 seconds
      cpu time            1.62 seconds
```

# This is the basic view

What am I looking at? What are all these window panes for?

Do not get confused by complexity. That is a common pitfall. The strategy to handle complex things thrown at you is to focus on your immediate goal and build on it.

What is our immediate goal?

To become familiar with the DATA-PROC steps.

I've heard that this interface allows us to do a wide range of tasks without any programming necessary. Can we avoid using commands and stick to the tools available from the point and click?

We could, but end up not utilising the full extent of SAS's capability. It is best to get through the tedious parts first.

# current section

1. an interesting read
2. Running SAS
3. Running our first SAS programs
4. General SAS
5. DATA STEP example
6. data ex
   - LongleyData
7. DATA SCIENCE TO THE RESCUE!

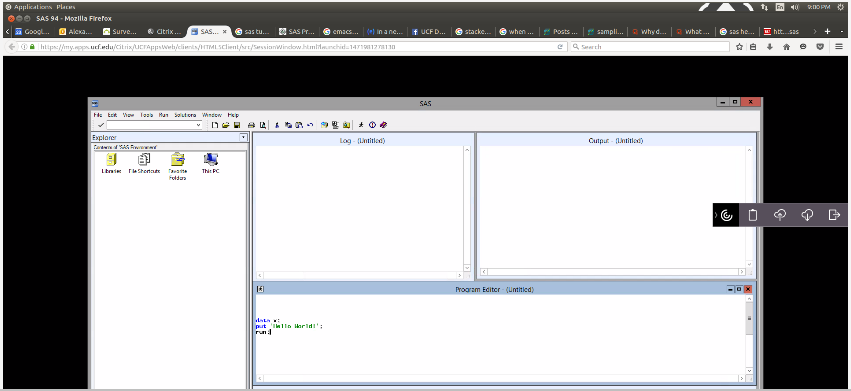
# The age long duty of printing out *Hello World!*

really? Yes.

```
data x;
put 'Hello World!';
run;
```

# The age long duty of printing out *Hello World!*

```
data x;
put 'Hello World!';
run;
```



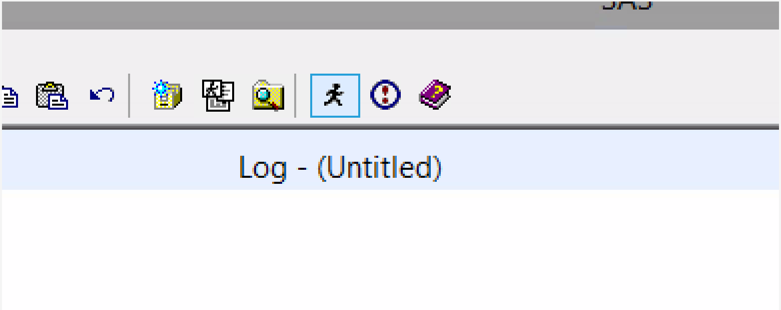
```
data x;
put 'Hello World!';
run;
```

Applications Places
9:00 PM
SAS 94 - Mozilla Firefox
https://my.apps.ucf.edu/Citrix/UCFAppsWeb/clients/HTML5Client/src/SessionWindow.html?launchid=1471981278130
Search
SAS
File Edit View Tools Run Solutions Window Help
Explorer
Contents of 'SAS Environment'
Libraries
File Shortcuts
Favorite Folders
This PC
Log - (Untitled)
Output - (Untitled)
Program Editor - (Untitled)

# The age long duty of printing out *Hello World!*

```
data x;
put 'Hello World!';
run;
```

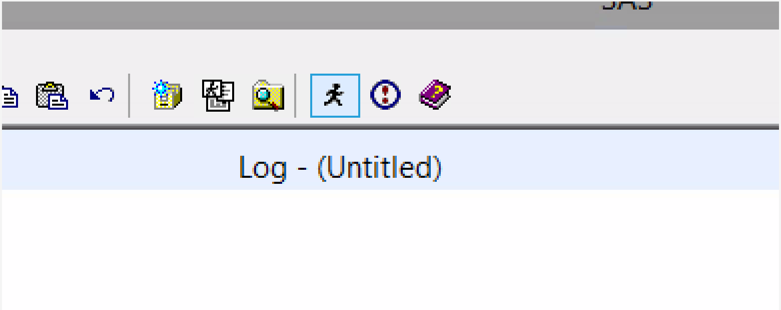
Now push the button on the Toolbar of a person running to execute the script (*Submit* button)



# The age long duty of printing out *Hello World!*

```
data x;
put 'Hello World!';
run;
```

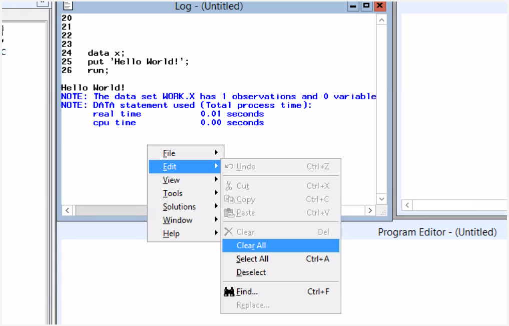
Now push the button on the Toolbar of a person running to execute the script (*Submit* button)



# The age long duty of printing out *Hello World!*

```
data x;
put 'Hello World!';
run;
```

Clearing the Log screen:

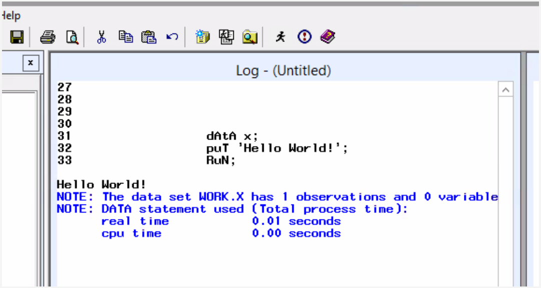


```
20
21
22
23
24      data x;
25      put 'Hello World!';
26      run;

Hello World!
NOTE: The data set WORK.X has 1 observations and 0 variable
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

# The age long duty of printing out *Hello World!*

```
dAtA x;
puT 'Hello World!';
RuN;
```

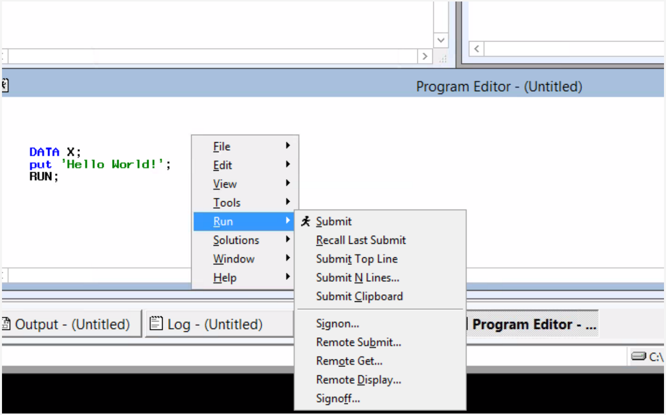
Changing the case from lower to upper does not change results:



```
                    dAtA x;
                    puT 'Hello World!';
                    RuN;

Hello World!
NOTE: The data set WORK.X has 1 observations and 0 variable
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

# The age long duty of printing out *Hello World!*

You can right click and choose the option to run the commands:



```
DATA X;
put 'Hello World!';
RUN;
```

# current section

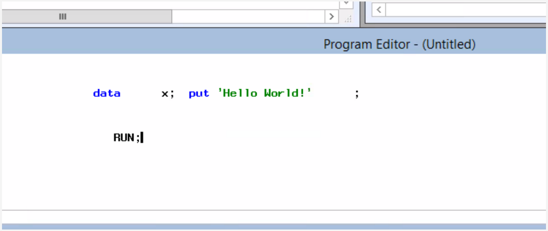
1. an interesting read
2. Running SAS
3. Running our first SAS programs
4. General SAS
5. DATA STEP example
6. data ex
   - LongleyData
7. DATA SCIENCE TO THE RESCUE!

# SAS Statements

- SAS statements can be chained onto the same line if they have a semi-colon to separate them.
- Statements require no particular alignment in the editor. The interpreter will assemble a correct statement form as long as the end of the line of commands has a semicolon, (;).
- Spaces can be included as you wish.

# SAS Statements

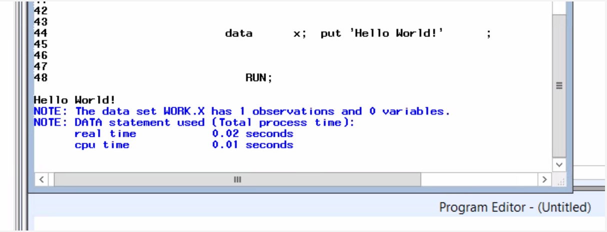
- SAS statements can be chained onto the same line if they have a semi-colon to separate them.
- Statements require no particular alignment in the editor. The interpreter will assemble a correct statement form as long as the end of the line of commands has a semicolon, (;).
- Spaces can be included as you wish.



```
        data      x;  put 'Hello World!'      ;

          RUN;
```

# SAS Statements

- SAS statements can be chained onto the same line if they have a semi-colon to separate them.
- Statements require no particular alignment in the editor. The interpreter will assemble a correct statement form as long as the end of the line of commands has a semicolon, (;).
- Spaces can be included as you wish.



```
                              data      x;  put 'Hello World!'      ;



                                 RUN;

Hello World!
NOTE: The data set WORK.X has 1 observations and 0 variables.
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.01 seconds
```

# SAS variable names

Some rules about the names.

- Maximum 32 characters long. (Sounds like something you would never do but names like *dataSetRetrialAugust23of2003goodSample* is not uncommon for you to want to remember the contents' contexts)
- No blanks. ( variable 'TEMP' cannot be written as 'TE MP').
- Variables can start with a letter or underscore(_).
- Numbers can be included in the name but not at the start.

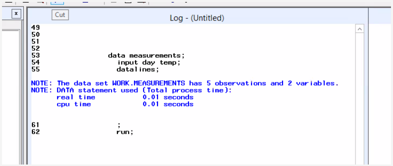
# current section

1. an interesting read
2. Running SAS
3. Running our first SAS programs
4. General SAS
5. DATA STEP example
6. data ex
   - LongleyData
7. DATA SCIENCE TO THE RESCUE!

# Another simple example

```
data measurements;
input day temp;
datalines;
1 50
2 52
3 49
4 48
8 55
;
run;
```

```
data measurements;
input day temp;
datalines;
1 50
2 52
3 49
4 48
8 55
;
run;
```

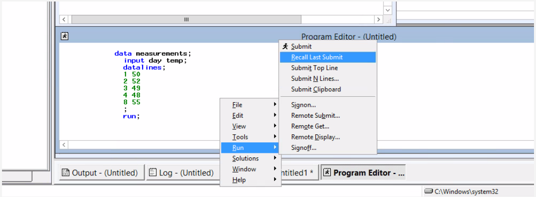


```
49
50
51
52
53                data measurements;
54                  input day temp;
55                  datalines;

NOTE: The data set WORK.MEASUREMENTS has 5 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


61                  ;
62                  run;
```

# Recall Last Submit

(Very important to not waste time!) Because the submit button sends things to the interpreter and clears the Program Editor pane on default options, you might end up retyping things, which is a waste.

*bullet*You can right click and select from Run-Recall Last Submit to bring back the commands you had.

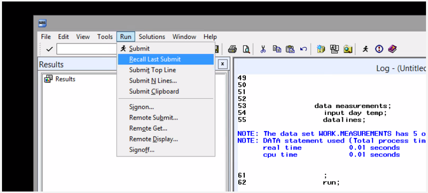


```
data measurements;
  input day temp;
  datalines;
  1 50
  2 52
  3 49
  4 48
  8 55
  ;
  run;
```

# Recall Last Submit

(Very important to not waste time!) Because the submit button sends things to the interpreter and clears the Program Editor pane on default options, you might end up retyping things, which is a waste.

- Use the drop down menues to access the Recall of the previous command set.



```
49
50
51
52
53                     data measurements;
54                       input day temp;
55                       datalines;

NOTE: The data set WORK.MEASUREMENTS has 5 o
NOTE: DATA statement used (Total process tim
      real time               0.01 seconds
      cpu time                0.01 seconds


61                       ;
62                       run;
```

# Another simple example

We can recall the previous submission and then add manually another variable of humidity.

```
data measurements;
input day temp humidity;
datalines;
1 50 88
2 52 84
3 49 84
4 48 88
8 55 99
;
run;
```

# Another simple example

THIS FAILS:

```
data measurements;
input day temp humidity rained;
datalines;
1 50 88 Y
2 52 84 Y
3 49 84 N
4 48 88 Y
8 55 99 Y
;
run;
```

WHY?

# Another simple example

THIS Works:

```
data measurements;
input day temp humidity rained $;
datalines;
1 50 88 Y
2 52 84 Y
3 49 84 N
4 48 88 Y
8 55 99 Y
;
run;
```

Because it has the ($) after the variable which is for character data. If SAS expects numerical input for a variable, and receives a character instead, it will fail.

# Another simple example

I just included an arbitrary '$' and it works:

```
data measurements;
input day $ temp humidity rained $;
datalines;
1 50 88 Y
2 52 84 Y
3 49 84 N
4 48 88 Y
8 55 99 Y
;
run;
```

Not so arbitrary in this case as the day can be used as an ID, which is unique, not like the temp variable. Using it for character data makes sense since you don't want to do arithmetic on it.

# Another simple example

Why is it so important?:

```
data measurements;
input day $ temp humidity rained $;
datal5ines;
1 50 88 Y
2 52 84 Y
3 49 84 N
4 48 88 Y
8 55 99 Y
;
run;
```

Character data does not fit into the numerical input.

# Another simple example

Why don’t we have everything in character variables?
For a couple of reasons. The system can do speed ups knowing if a datastructure is only comprised of numerical data. If you look at the speed ups the language Julia has over Python and Ruby, it is because it requires like many other languages for data types to be predefined and will throw errors if it is violated. The other is to be able to do arithmetic operation between variables known to contain only numerical values.

# Comments

I would like to add some notes to the code to remember what it is that I am doing. To remember my track of thinking in the future

What you want to do is add comments to your code in SAS, especially if you plan to save the commands in a '.sas' file. Comments can be added in two ways:

## Comments in SAS

- *Commented text;
- /*Commented text*/

# Commenting code-simple example

```
data measurements;
input day $ temp humidity rained $;/*Observed days and the temp/humidity
and whether it rained*/
datalines;
1 50 88 Y
2 52 84 Y
3 49 84 N *No rain this day;
4 48 88 Y
8 55 99 Y
;
run;
```

Yes, it is good to describe the data, because in the future numerical columns might be forgotten what they measure.

# Some points on the DATA step statement

If you only use the 'data' statement keyword, a temporary placeholder is created without memory usage or the allocation of a namespace after the session. You can make the name persistent if you attach it to a library name variable.

- DATA tempData;
- MyDataLib.temp1;

DATA <data-set-name-1 <(data-set-options-1)>>

# SAS Variables

I understand a bit more the data keyword, but in the example what is the meaning of the 'input' keyword I saw for variables in the manually inserted dataset?

Well, you just said it from your understanding of reading the log file summarising the set up of dataset structures. The 'INPUT' allows you to define the number/name/type of variables collected in each observation. Traditionally and most of the time, this corresponds to the column names because each observation is recorded into a row in the matrix/spreadsheet.

# SAS Variables

Syntax for NUMERIC variables:

```
INPUT VAR1 VAR2 VAR3;
```

Syntax for CHARACTER variables:

```
INPUT VAR1 $ VAR2 $ VAR3 $;
```

DATE variables as well which can be done in various ways:

```
INPUT VarID $ VarDATE MMDDYY10.;
```

Date variable formatting is quite extensive and varied according to the purpose. More on that in the near future.

# Important point to remember

Writing code in any language is not about remembering the syntax, it is about the style which code is organised, data is organised, nature of the datastructures and operations on them. Eg. Matlab using for-loops loses its benefits, understanding the nature of vector operations is its strength and knowing that allows you to do things the *Matlab way*. There is also the *Python* way.

# Python ZEN *https://www.python.org/dev/peps/pep-0020/*

***A selection of my favourite*:**

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
There should be one– and preferably only one –obvious way to do it.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.e of those!

# SAS has its own way

SAS has its own way of doing things although it is not stated as explicitly as it is for Python. You could one day write a similar set of guidelines for it as there as a vacancy.
I mention this here merely to not focus too hard on the individual aspects of syntax which can always be looked up, but for the general style which outlines the experience with SAS or any language and system.

# current section

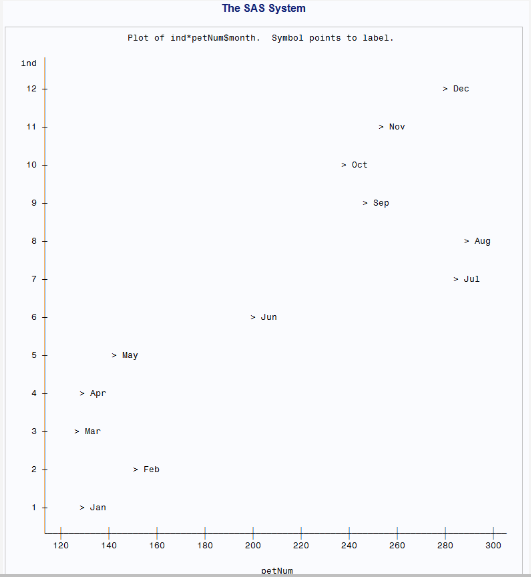
1. an interesting read
2. Running SAS
3. Running our first SAS programs
4. General SAS
5. DATA STEP example
6. data ex
   - LongleyData
7. DATA SCIENCE TO THE RESCUE!

# King County, Washington PetData 2016 Pet Adoptions

```
data one;
input ind month $ petNum;
datalines;
1 Jan 129
2 Feb 151
3 Mar 126
4 Apr 128
5 May 143
6 Jun 200
7 Jul 285
8 Aug 288
9 Sep 247
10 Oct 238
11 Nov 253
12 Dec 279
;
run;
```

```
proc plot data = one;
plot ind*petNum $month;
by month; run;
```

# output of petplot, *conclusions?*



The SAS System

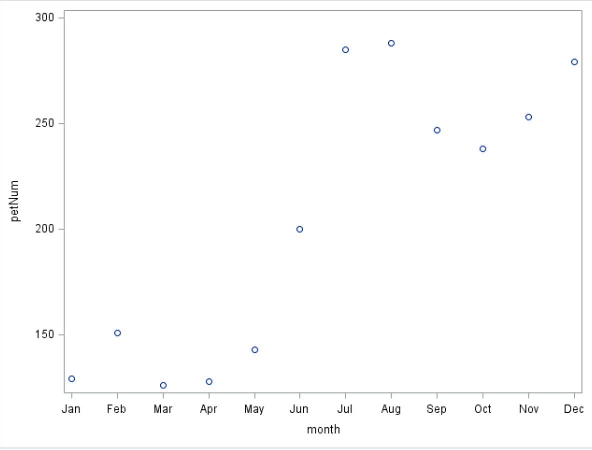
Plot of ind*petNum$month.  Symbol points to label.

# King County, Washington PetData 2016 Pet Adoptions

```
data one;
input ind month $ petNum;
datalines;
1 Jan 129
2 Feb 151
3 Mar 126
4 Apr 128
5 May 143
6 Jun 200
7 Jul 285
8 Aug 288
9 Sep 247
10 Oct 238
11 Nov 253
12 Dec 279
;
run;
```

```
proc sgplot data = one;
scatter x = month y = petNum;
run;
```

# output of petplot 'sgplot', *different conclusions?*

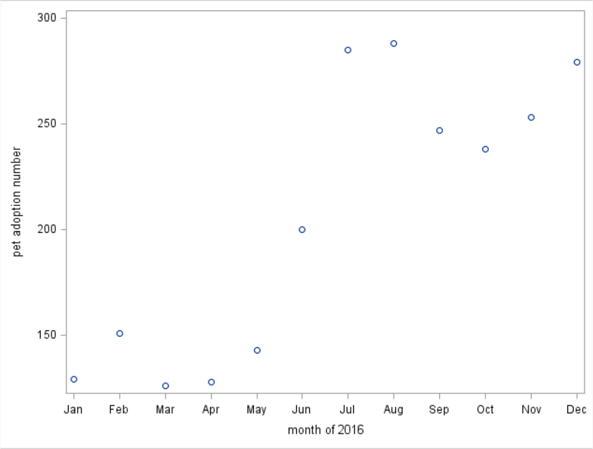


# King County, Washington PetData 2016 Pet Adoptions

```
data one;
input ind month $ petNum;
datalines;
1 Jan 129
2 Feb 151
3 Mar 126
4 Apr 128
5 May 143
6 Jun 200
7 Jul 285
8 Aug 288
9 Sep 247
10 Oct 238
11 Nov 253
12 Dec 279
;
run;
```

```
proc sgplot data = one;
xaxis label = "month of 2016";
yaxis label = "pet adoption
number";
scatter x = month y = petNum;
run;
```

# output of petplot 'sgplot', *with better labels*

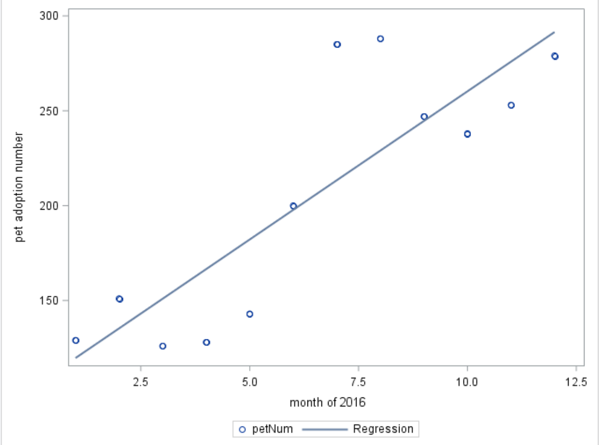


# PetData 2016 scatter plot with a *linear regression* line

```
data one;
input ind month $ petNum;
datalines;
1 Jan 129
2 Feb 151
3 Mar 126
4 Apr 128
5 May 143
6 Jun 200
7 Jul 285
8 Aug 288
9 Sep 247
10 Oct 238
11 Nov 253
12 Dec 279
;
run;
```

```
proc sgplot data = one;
xaxis label = ”month of 2016”;
yaxis label = ”pet adoption
number”;
scatter x = ind y = petNum;
reg x = ind y = petNum;
run;
```

# output of petplot 'sgplot', *with regression line*

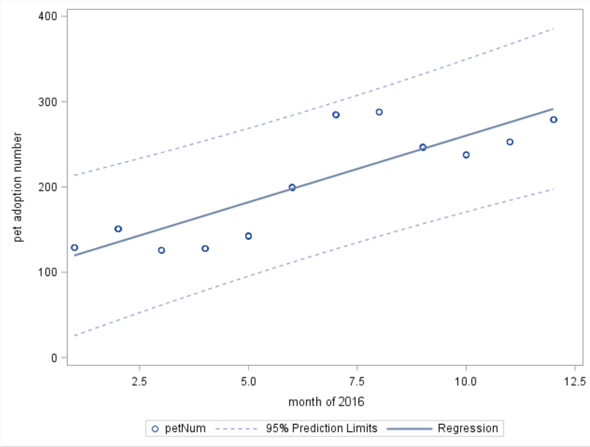


# PetData with *confidence limits for individual predicted observations*

```
data one;
input ind month $ petNum;
datalines;
1 Jan 129
2 Feb 151
3 Mar 126
4 Apr 128
5 May 143
6 Jun 200
7 Jul 285
8 Aug 288
9 Sep 247
10 Oct 238
11 Nov 253
12 Dec 279
;
run;
```

```
proc sgplot data = one;
xaxis label = ”month of 2016”;
yaxis label = ”pet adoption
number”;
scatter x = ind y = petNum;
reg x = ind y = petNum / CLI;
run;
```

# output of petplot 'sgplot', *with CLI*

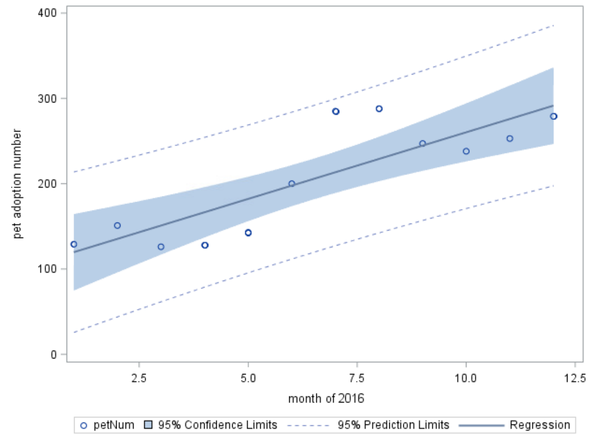


# PetData with *confidence limits for predicted mean of observations*

```
data one;
input ind month $ petNum;
datalines;
1 Jan 129
2 Feb 151
3 Mar 126
4 Apr 128
5 May 143
6 Jun 200
7 Jul 285
8 Aug 288
9 Sep 247
10 Oct 238
11 Nov 253
12 Dec 279
;
run;
```

```
proc sgplot data = one;
xaxis label = ”month of 2016”;
yaxis label = ”pet adoption
number”;
scatter x = ind y = petNum;
reg x = ind y = petNum / CLI
CLM;
run;
```

# output of petplot 'sgplot', *with CLI and CLM*



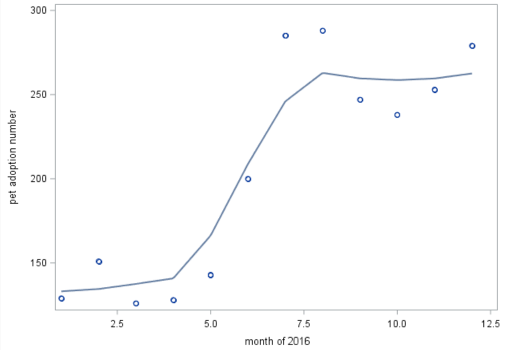
# LOESS/LOWESS model

## Locally weighted scatterplot smoothing

Works on subsetting the data to produce a smooth curve via weighted least squares regression using linear or some degree of polynomial function. It goes through each of the data points and using a ’local view’ from that point (not from setting Euclidean distance threshold but order, remember K-NN). This k is an alpha parameter that determines the ’locality’ measures. The more ’local’ the less smooth it will be. As you can imagine, if the data is not dense in some parts, we will be overfitting in that region.

# pet data with LOESS model

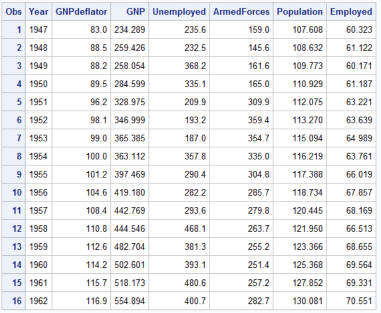
```
proc sgplot data=one;
xaxis label = ”month of 2016”;
yaxis label = ”pet adoption number”;
scatter x = ind y = petNum;
loess x=ind y=petNum;
run;
```



# Longley economic data: US dept of labor March1963

```
data econ;
infile datalines delimiter=’,’;
input Year $ GNPdeflator GNP
Unemployed ArmedForces
Population Year Employed;
datalines;
"1947",83,234.289,235.6,159,107.608,1947,60.323
"1948",88.5,259.426,232.5,145.6,108.632,1948,61.122
"1949",88.2,258.054,368.2,161.6,109.773,1949,60.171
"1950",89.5,284.599,335.1,165,110.929,1950,61.187
"1951",96.2,328.975,209.9,309.9,112.075,1951,63.221
"1952",98.1,346.999,193.2,359.4,113.27,1952,63.639
"1953",99,365.385,187,354.7,115.094,1953,64.989
"1954",100,363.112,357.8,335,116.219,1954,63.761
"1955",101.2,397.469,290.4,304.8,117.388,1955,66.019
"1956",104.6,419.18,282.2,285.7,118.734,1956,67.857
"1957",108.4,442.769,293.6,279.8,120.445,1957,68.169
"1958",110.8,444.546,468.1,263.7,121.95,1958,66.513
"1959",112.6,482.704,381.3,255.2,123.366,1959,68.655
"1960",114.2,502.601,393.1,251.4,125.368,1960,69.564
"1961",115.7,518.173,480.6,257.2,127.852,1961,69.331
"1962",116.9,554.894,400.7,282.7,130.081,1962,70.551

;run;
```

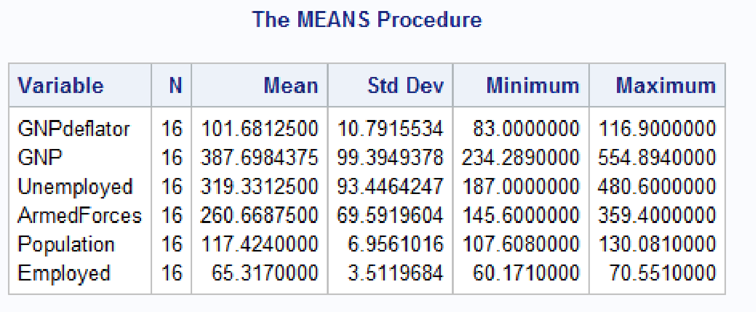
```
proc print data = econ;
run;
```



| Obs | Year | GNPdeflator | GNP | Unemployed | ArmedForces | Population | Employed |
|---|---|---|---|---|---|---|---|
| 1 | 1947 | 83.0 | 234.289 | 235.6 | 159.0 | 107.608 | 60.323 |
| 2 | 1948 | 88.5 | 259.426 | 232.5 | 145.6 | 108.632 | 61.122 |
| 3 | 1949 | 88.2 | 258.054 | 368.2 | 161.6 | 109.773 | 60.171 |
| 4 | 1950 | 89.5 | 284.599 | 335.1 | 165.0 | 110.929 | 61.187 |
| 5 | 1951 | 96.2 | 328.975 | 209.9 | 309.9 | 112.075 | 63.221 |
| 6 | 1952 | 98.1 | 346.999 | 193.2 | 359.4 | 113.270 | 63.639 |
| 7 | 1953 | 99.0 | 365.385 | 187.0 | 354.7 | 115.094 | 64.989 |
| 8 | 1954 | 100.0 | 363.112 | 357.8 | 335.0 | 116.219 | 63.761 |
| 9 | 1955 | 101.2 | 397.469 | 290.4 | 304.8 | 117.388 | 66.019 |
| 10 | 1956 | 104.6 | 419.180 | 282.2 | 285.7 | 118.734 | 67.857 |
| 11 | 1957 | 108.4 | 442.769 | 293.6 | 279.8 | 120.445 | 68.169 |
| 12 | 1958 | 110.8 | 444.546 | 468.1 | 263.7 | 121.950 | 66.513 |
| 13 | 1959 | 112.6 | 482.704 | 381.3 | 255.2 | 123.366 | 68.655 |
| 14 | 1960 | 114.2 | 502.601 | 393.1 | 251.4 | 125.368 | 69.564 |
| 15 | 1961 | 115.7 | 518.173 | 480.6 | 257.2 | 127.852 | 69.331 |
| 16 | 1962 | 116.9 | 554.894 | 400.7 | 282.7 | 130.081 | 70.551 |

# Longley economic data: US dept of labor March1963

```
proc means data=econ;
run;
```

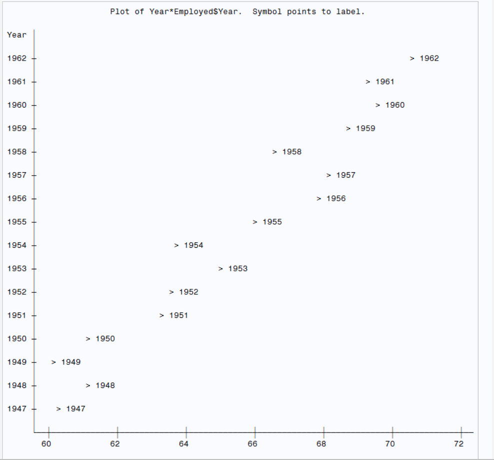


The MEANS Procedure

| Variable | N | Mean | Std Dev | Minimum | Maximum |
|---|---|---|---|---|---|
| GNPdeflator | 16 | 101.6812500 | 10.7915534 | 83.0000000 | 116.9000000 |
| GNP | 16 | 387.6984375 | 99.3949378 | 234.2890000 | 554.8940000 |
| Unemployed | 16 | 319.3312500 | 93.4464247 | 187.0000000 | 480.6000000 |
| ArmedForces | 16 | 260.6687500 | 69.5919604 | 145.6000000 | 359.4000000 |
| Population | 16 | 117.4240000 | 6.9561016 | 107.6080000 | 130.0810000 |
| Employed | 16 | 65.3170000 | 3.5119684 | 60.1710000 | 70.5510000 |

# Longley economic data: US dept of labor March1963

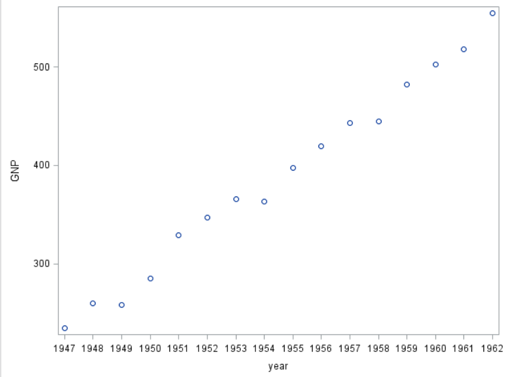
```
proc plot data=econ;
plot Year*Employed $Year;
run;
```



Plot of Year*Employed$Year.  Symbol points to label.

# Longley data sgplot years and GNP

```
proc sgplot data=econ;
xaxis label = “year”;
scatter x = Year y = GNP;
run;
```



# Longley data sgplot years and GNP and GNPdeflator

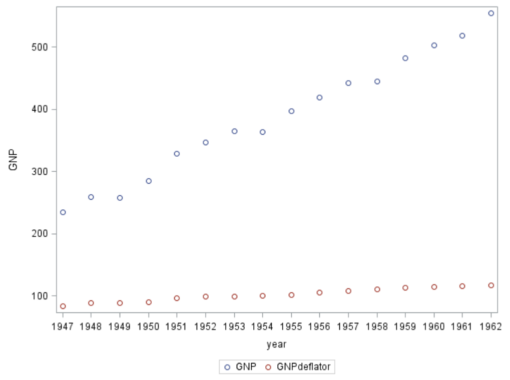
proc sgplot data=econ;

xaxis label = “year”;

scatter x = Year y = GNP;

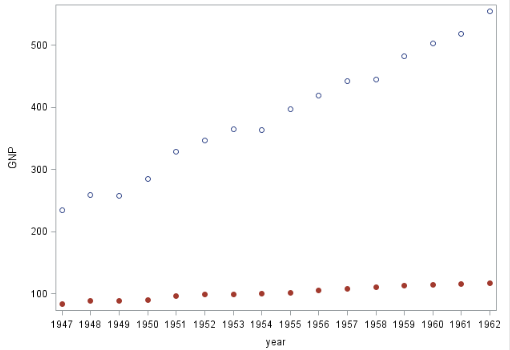
scatter x = Year y = GNPdeflator;

run;



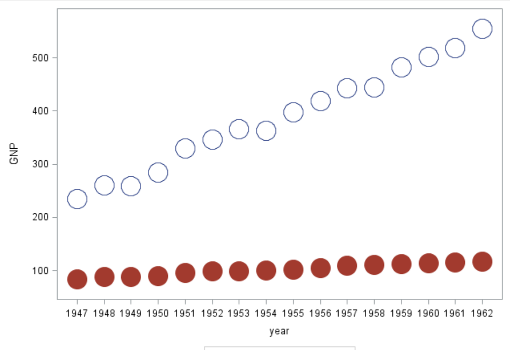
# Longley data sgplot years and GNP and GNPdeflator

```
proc sgplot data=econ;
xaxis label = “year”;
scatter x = Year y = GNP;
scatter x = Year y = GNPdeflator /
markerattrs=(symbol=CircleFilled);
run;
```



# Longley data sgplot years and GNP and GNPdeflator

```
proc sgplot data=econ;
xaxis label = "year";
scatter x = Year y = GNP / markerattrs=(size=25);
scatter x = Year y = GNPdeflator /
markerattrs=(symbol=CircleFilled size= 25);
run;
```



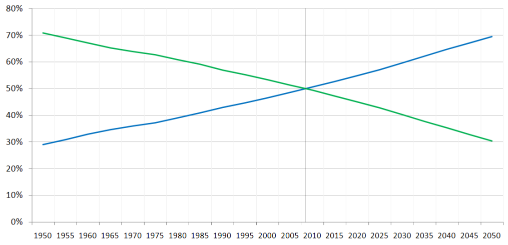
# current section

1. an interesting read
2. Running SAS
3. Running our first SAS programs
4. General SAS
5. DATA STEP example
6. data ex
    - LongleyData
7. DATA SCIENCE TO THE RESCUE!

# The housing price crisis

In the UK, there is a very large problem with the amount of available housing. This is especially relevant as a problem in cities such as London, but now even other cities as well.
Are houses that difficult to construct? Why is it so difficult to build panel based houses? They look simple to set up. Well, we can have a crisis merely from relocation. It also creates a large amount of debt.

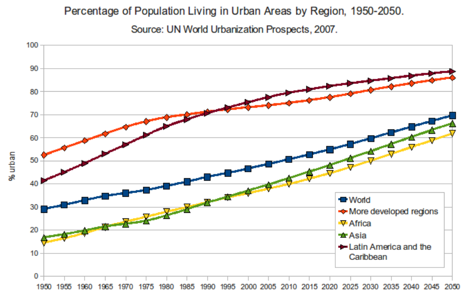
The global trend of urbanisation is on the increase.



'United Nations, Department of Economic and Social Affairs'

# The Urbanisation across our world

This is not just a trend from the 'developed' world, the developing nations are experiencing it at an even faster rate. It is the rate which out runs the supply, which in cities is slower than in rural areas, and the demand raises prices.

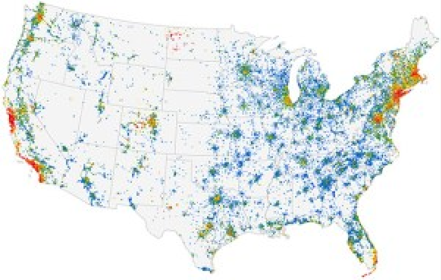


Percentage of Population Living in Urban Areas by Region, 1950-2050.

Source: UN World Urbanization Prospects, 2007.

# Recent article using Craigslist data

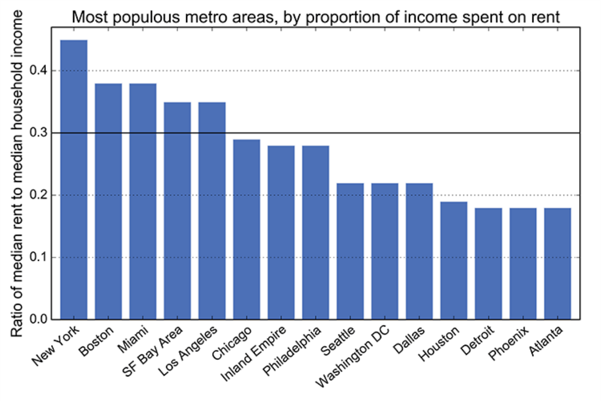
*http://geoffboeing.com*
The UC Berkeley Urban Analytics Lab collected house prices from Craigslist.



Rentals make up a significant portion of the U.S. housing market, but much of this market activity is poorly understood due to its informal characteristics and historically minimal data trail.
Analyzed 11 million Craigslist rental listings to discover fine-grained patterns across metropolitan housing markets in the United States

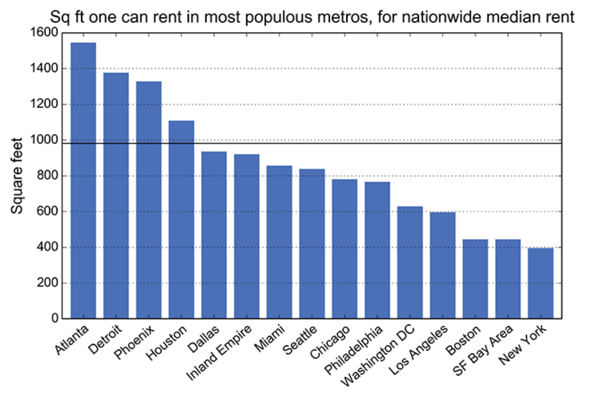
# Rent Burden

The standard definition of 'rent burden' is rent exceeding 30% of household income. This chart shows the share of its income that a typical (i.e., median) household would spend on a typical Craigslist rent, in each of the 15 most populous metro areas:



# Rental Power

A 'rental power' indicator represents an estimate of how many square feet someone can rent on Craigslist in each metro area for the nationwide median rent of \$1,145. It simply divides the nationwide median rent by each regional median rent/ft$^2$


Sq ft one can rent in most populous metros, for nationwide median rent
Square feet
Atlanta
Detroit
Phoenix
Houston
Dallas
Inland Empire
Miami
Seattle
Chicago
Philadelphia
Washington DC
Los Angeles
Boston
SF Bay Area
New York