Intro to Data Analysis in Python

April 1, 2019

```
In [1]: %matplotlib inline
```

1 Intro to Data Analysis in Python

This tutorial provides an overview of common data anlysis tasks and packages in Python. "Intro to Programming in Python" is a pre-requisite, or working knowledge of data types and structures, loops and conditional statements, functions, reading and writing data. In this tutorial we will: - Import the packages needed. - Read in the data we're using as an example. - Clean and proces the data using different functions in the Pandas package. - Compute descriptive statistics for the variables of interest. - Display frequency and proportions tables. - Calculate correlations. - Conduct hypothesis tests. - Conduct regression analysis and regression diagnostics checks. - Visualize relations in the data.

1.1 Importing modules

Everything that we've done so far was based on functions from base Python. However, we will often need to import other packages which can handle more complex or specific tasks. For example, we may want to use a module that is able to better read and write csv data, such as the 'csv' module. To do that, we have to first import the module. For packages that are already installed, you can simply do that by typing 'import' and the name of the package. Many of the useful packages are already installed in Anaconda.

```
In [2]: import csv
```

But how do you know which packages are installed? If you open Anaconda Prompt, and you type "conda list", it will list all installed package. You can do this in any terminal/command prompt.

If a package is not installed, you can install it in the Anaconda Prompt with: `conda install csv`

1.2 Reading data in different formats

The 'csv' module is already installed in Anaconda, so we can go ahead and import it. Let's read the file in csv format, recode missing values as NA, and write it out as a new clean.csv. The 'csv' module is very useful for manipulating large files that contain long text fields.

```
In [3]: import csv
        with open("clean.csv", "w") as outfile:
            writer=csv.writer(outfile)
            with open("mydataset.csv", "r") as infile:  # open the file for writing
                reader=csv.reader(infile)
                writer.writerow(next(reader))
                for row in reader:
                    writer.writerow(row[0:6]+[(row[6].replace("missing", "NaN"))])
```

You can also read csv files, as well as other file formats using Pandas. Pandas is one of the main libraries for data analysis in Python. For those of you familiar with R, the data frames structure and Pandas will make it very easy to use. Let's see what we can do, by importing the clean.csv file that you just saved.

```
In [4]: import pandas as pd # we import it as pd because it's easier to type
        df=pd.read_csv("clean.csv")
        df
```

Out[4]:

	id	region	party	chamber	spent	raised	reelected
0	1	East	Centre	H	285937	411847	0.0
1	2	East	Centre	H	308530	1301546	1.0
2	3	East	Centre	H	435962	629768	4.0
3	4	East	Centre	H	685526	737446	3.0
4	5	East	Centre	H	242312	370557	1.0
5	6	East	Centre	H	149546	432485	3.0
6	7	East	Centre	H	618818	850163	2.0
7	8	East	Centre	H	354655	364555	2.0
8	9	East	Centre	H	147248	165364	0.0
9	10	East	Centre	H	306052	360675	3.0
10	11	East	Centre	H	746673	1025318	0.0
11	12	East	Centre	H	1265171	4205366	5.0
12	13	East	Centre	H	54084	100084	0.0
13	14	East	Centre	H	260806	457341	1.0
14	15	East	Centre	H	71157	237351	4.0
15	16	East	Centre	H	261123	373064	3.0
16	17	East	Centre	H	136185	571276	1.0
17	18	East	Centre	H	218830	320998	2.0
18	19	East	Centre	H	251084	700724	2.0
19	20	East	Centre	H	200186	588016	2.0
20	21	East	Centre	H	455185	557252	0.0
21	22	East	Centre	H	322763	712911	3.0
22	23	East	Centre	H	360835	539662	NaN
23	24	East	Centre	H	245872	280121	5.0
24	25	East	Centre	H	316460	370905	0.0
25	26	East	Centre	H	253641	937422	1.0
26	27	East	Centre	H	286431	834714	4.0
27	28	East	Centre	H	237073	296125	3.0

```
28    29   East  Centre       H   210111  1159071        1.0
29    30   East  Centre       S    66606    48704        3.0
..   ...    ...     ...     ...      ...      ...        ...
509  510   West   Right       H   212548   218840        3.0
510  511   West   Right       H    96166   251470        2.0
511  512   West   Right       H   187035   324529        2.0
512  513   West   Right       H   272139   454256        0.0
513  514   West   Right       H   433568   843939        3.0
514  515   West   Right       H   421667   691779        0.0
515  516   West   Right       H   297943   250585        5.0
516  517   West   Right       H   208831   380948        0.0
517  518   West   Right       H   226642   420613        1.0
518  519   West   Right       H   299537   664525        4.0
519  520   West   Right       H   361325   250110        3.0
520  521   West   Right       H   170404   244851        1.0
521  522   West   Right       H   230738   455582        3.0
522  523   West   Right       H   242817   263879        2.0
523  524   West   Right       H   199311   255132        2.0
524  525   West   Right       H   372081   449664        0.0
525  526   West   Right       H   526982   990676        3.0
526  527   West   Right       H   266526   720413        0.0
527  528   West   Right       H    72911   148110        5.0
528  529   West   Right       H   291832   613410        0.0
529  530   West   Right       H  1417682  2122890        1.0
530  531   West   Right       H   154252   561509        4.0
531  532   West   Right       H   495108  1261714        3.0
532  533   West   Right       H   396456   967929        1.0
533  534   West   Right       H   139957   257002        3.0
534  535   West   Right       H   187541   469759        2.0
535  536   West   Right       S   112443    78720        2.0
536  537   West   Right       S   157211  1073163        0.0
537  538   West   Right       S  1692394  4631824        3.0
538  539   West   Right       S   424965   474590        0.0

[539 rows x 7 columns]
```

1.3 Processing data with Pandas

```
In [5]: df.head(5) # first 5 rows
```

```
Out[5]:    id region   party chamber   spent   raised  reelected
       0   1   East  Centre       H  285937   411847        0.0
       1   2   East  Centre       H  308530  1301546        1.0
       2   3   East  Centre       H  435962   629768        4.0
       3   4   East  Centre       H  685526   737446        3.0
       4   5   East  Centre       H  242312   370557        1.0
```

```
In [6]: df.tail(3) # last 3 rows
```

```
Out[6]:       id region  party chamber    spent   raised  reelected
        536  537   West  Right       S   157211  1073163        0.0
        537  538   West  Right       S  1692394  4631824        3.0
        538  539   West  Right       S   424965   474590        0.0

In [7]: df.shape # how many rows and columns

Out[7]: (539, 7)

In [8]: df.columns # the column names

Out[8]: Index(['id', 'region', 'party', 'chamber', 'spent', 'raised', 'reelected'], dtype='obj

In [9]: df["reelected"][0:5] # select a column, and a slice within it

Out[9]: 0    0.0
        1    1.0
        2    4.0
        3    3.0
        4    1.0
        Name: reelected, dtype: float64

In [10]: df.region.unique() # Unique values in a column

Out[10]: array(['East', 'North', 'North-East', 'South', 'West'], dtype=object)

In [11]: # Subsetting data: create another data frame that only includes obswervations from th
         value_list=["South", "East"]
         df_SE=df[df.region.isin(value_list)] # Replace this with df[~df.region...] to keep on
         df_SE.count()

Out[11]: id           216
         region       216
         party        216
         chamber      216
         spent        216
         raised       216
         reelected    214
         dtype: int64

In [12]: # Select only dataframes that meet multiple conditions:
         df_restricted=df[(df['region']=="South") & (df["chamber"]=="S") & (df["reelected"]==0)
         df_restricted.head(5)

Out[12]:       id region   party chamber    spent   raised  reelected
         356  357  South  Centre       S   199176   436192        0.0
         358  359  South  Centre       S   221402   424304        0.0
         392  393  South    Left       S  1768956  4699994        0.0
         394  395  South    Left       S  1972873    48947        0.0
         428  429  South   Right       S   719563  3231786        0.0
```

```
In [13]: # Group and aggregate
         grouped=df.groupby(["region", "chamber"])
         aggregated=grouped.agg({"spent":['sum','mean', 'min'],
                                 'raised':['sum', 'mean', 'max']})
         aggregated
```

Out[13]:

		spent			raised		
		sum	mean	min	sum	mean	max
region	chamber						
East	H	29446708	320072.913043	0	55481753	6.030625e+05	4205366
	S	7259432	453714.500000	0	12413611	7.758507e+05	3959212
North	H	21778335	259265.892857	0	39890400	4.748857e+05	1773323
	S	12482798	520116.583333	0	27547225	1.147801e+06	6263060
North-East	H	27511352	348244.962025	0	53487404	6.770557e+05	4091159
	S	13722168	490077.428571	0	43154780	1.541242e+06	9790929
South	H	29046467	330073.488636	0	56751507	6.449035e+05	3020933
	S	11140983	557049.150000	40206	22286751	1.114338e+06	4699994
West	H	30186789	331722.956044	43175	59857901	6.577791e+05	5169778
	S	6797473	399851.352941	0	15904129	9.355370e+05	4631824

1.4 Descriptive statistics

Mean, median, 25th and 75th quartiles, min, max, number of missing observations, etc.

```
In [14]: # For numeric variables in the dataset
         df.describe()
```

Out[14]:

	id	spent	raised	reelected
count	539.000000	5.390000e+02	5.390000e+02	537.000000
mean	270.000000	3.513405e+05	7.175797e+05	2.005587
std	155.740168	3.440426e+05	9.499696e+05	1.573440
min	1.000000	0.000000e+00	0.000000e+00	0.000000
25%	135.500000	1.721955e+05	2.760930e+05	1.000000
50%	270.000000	2.563070e+05	4.558610e+05	2.000000

```
75%    404.500000  3.935870e+05  7.554350e+05    3.000000
max    539.000000  3.489592e+06  9.790929e+06    5.000000
```

```
In [15]: #Let's turn off scientific notation:
         pd.options.display.float_format = '{:.2f}'.format
         df.describe()
```

Out[15]:

	id	spent	raised	reelected
count	539.00	539.00	539.00	537.00
mean	270.00	351340.45	717579.71	2.01
std	155.74	344042.64	949969.55	1.57
min	1.00	0.00	0.00	0.00
25%	135.50	172195.50	276093.00	1.00
50%	270.00	256307.00	455861.00	2.00
75%	404.50	393587.00	755435.00	3.00
max	539.00	3489592.00	9790929.00	5.00

```
In [16]: # For all variables in the dataset
         df.describe(include='all')
```

Out[16]:

	id	region	party	chamber	spent	raised	reelected
count	539.00	539	539	539	539.00	539.00	537.00
unique	nan	5	3	2	nan	nan	nan
top	nan	East	Right	H	nan	nan	nan
freq	nan	108	180	434	nan	nan	nan
mean	270.00	NaN	NaN	NaN	351340.45	717579.71	2.01
std	155.74	NaN	NaN	NaN	344042.64	949969.55	1.57
min	1.00	NaN	NaN	NaN	0.00	0.00	0.00
25%	135.50	NaN	NaN	NaN	172195.50	276093.00	1.00
50%	270.00	NaN	NaN	NaN	256307.00	455861.00	2.00
75%	404.50	NaN	NaN	NaN	393587.00	755435.00	3.00
max	539.00	NaN	NaN	NaN	3489592.00	9790929.00	5.00

You can compute summary statistics on a single variable:

```
In [17]: df['spent'].describe()
```

```
Out[17]: count       539.00
         mean     351340.45
         std      344042.64
         min           0.00
         25%      172195.50
         50%      256307.00
         75%      393587.00
         max     3489592.00
         Name: spent, dtype: float64
```

You can also calculate one value of interest at a time:

```
In [18]: #The sum of all the money raised
         df['raised'].sum()

Out[18]: 386775461

In [19]: #Cumulative sum of the money raised
         df['raised'].cumsum()

Out[19]: 0          411847
         1         1713393
         2         2343161
         3         3080607
         4         3451164
         5         3883649
         6         4733812
         7         5098367
         8         5263731
         9         5624406
         10        6649724
         11       10855090
         12       10955174
         13       11412515
         14       11649866
         15       12022930
         16       12594206
         17       12915204
         18       13615928
         19       14203944
         20       14761196
         21       15474107
         22       16013769
         23       16293890
         24       16664795
         25       17602217
         26       18436931
         27       18733056
         28       19892127
         29       19940831
                    ...
         509    366201890
         510    366453360
         511    366777889
         512    367232145
         513    368076084
         514    368767863
         515    369018448
         516    369399396
         517    369820009
```

```
518    370484534
519    370734644
520    370979495
521    371435077
522    371698956
523    371954088
524    372403752
525    373394428
526    374114841
527    374262951
528    374876361
529    376999251
530    377560760
531    378822474
532    379790403
533    380047405
534    380517164
535    380595884
536    381669047
537    386300871
538    386775461
Name: raised, Length: 539, dtype: int64
```

```
In [20]: #Count the number of non-NA values
         df['reelected'].count()
```

```
Out[20]: 537
```

```
In [21]: #Minimum spending
         df['spent'].min()
```

```
Out[21]: 0
```

```
In [22]: #Maximum spending
         df['spent'].max()
```

```
Out[22]: 3489592
```

```
In [23]: #Mean spending
         df['spent'].mean()
```

```
Out[23]: 351340.45454545453
```

```
In [24]: #Median spending
         df['spent'].median()
```

```
Out[24]: 256307.0
```

```
In [25]: #Skewness of spending values
         df['spent'].skew()
```

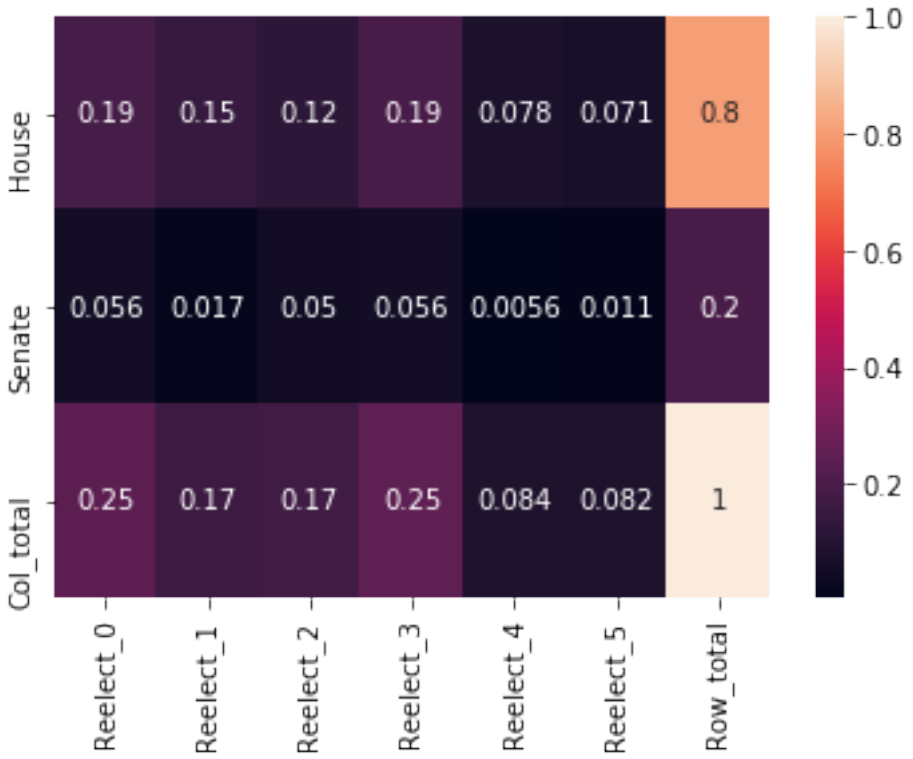
```
Out[25]: 3.625230614314503

In [26]: #Sample variance of spending
         df['spent'].var()

Out[26]: 118365339902.01813

In [27]: #Sample standard deviation of spending
         df['spent'].std()

Out[27]: 344042.6425634156

In [28]: #Kurtosis of spending values
         df['spent'].kurt()

Out[28]: 19.984062718467175
```

1.5 Frequency and proportions tables

You can make al types of frequency tables in Pandas using the *crosstab* function.

```
In [29]: # One way table of frequencies
         pd.crosstab(index=df["chamber"], columns="count")

Out[29]: col_0    count
         chamber
         H          434
         S          105

In [30]: # One way table of proportions
         pd.crosstab(index=df["chamber"],  # Make a crosstab
                     columns="count",  # Name the count column
                     normalize="all")  # Display as percentages. Options are all, index(row), c

Out[30]: col_0    count
         chamber
         H         0.81
         S         0.19

In [31]: # Two way table of frequencies
         pd.crosstab(df.chamber, df.reelected)

Out[31]: reelected  0.00  1.00  2.00  3.00  4.00  5.00
         chamber
         H           102    80    66   104    42    38
         S            30     9    27    30     3     6

In [32]: # Two way table of frequencies
         pd.crosstab(df.chamber, df.reelected, normalize="all") #proportion of total
```

```
Out[32]: reelected  0.00  1.00  2.00  3.00  4.00  5.00
         chamber
         H          0.19  0.15  0.12  0.19  0.08  0.07
         S          0.06  0.02  0.05  0.06  0.01  0.01

In [33]: # Two way table of frequencies
         pd.crosstab(df.chamber, df.reelected, normalize="index") #proportion of rows

Out[33]: reelected  0.00  1.00  2.00  3.00  4.00  5.00
         chamber
         H          0.24  0.19  0.15  0.24  0.10  0.09
         S          0.29  0.09  0.26  0.29  0.03  0.06

In [34]: # Two way table of frequencies
         pd.crosstab(df.chamber, df.reelected, normalize="columns") #proportion of columns

Out[34]: reelected  0.00  1.00  2.00  3.00  4.00  5.00
         chamber
         H          0.77  0.90  0.71  0.78  0.93  0.86
         S          0.23  0.10  0.29  0.22  0.07  0.14

In [35]: # Two way table of frequencies
         pd.crosstab(df.chamber, df.reelected, normalize="all", margins=True) # add row and co

Out[35]: reelected  0.0  1.0  2.0  3.0  4.0  5.0  All
         chamber
         H         0.19 0.15 0.12 0.19 0.08 0.07 0.80
         S         0.06 0.02 0.05 0.06 0.01 0.01 0.20
         All       0.25 0.17 0.17 0.25 0.08 0.08 1.00

In [36]: # Label the rows and columns
         mytab = pd.crosstab(df.chamber, df.reelected, normalize="all", margins=True)   # Incl
         mytab.columns = ["Reelect_0","Reelect_1","Reelect_2","Reelect_3","Reelect_4","Reelect_
         mytab.index= ["House","Senate","Col_total"]

         mytab

Out[36]:            Reelect_0  Reelect_1  Reelect_2  Reelect_3  Reelect_4  Reelect_5  \
         House           0.19       0.15       0.12       0.19       0.08       0.07
         Senate          0.06       0.02       0.05       0.06       0.01       0.01
         Col_total       0.25       0.17       0.17       0.25       0.08       0.08

                     Row_total
         House            0.80
         Senate           0.20
         Col_total        1.00

In [37]: import seaborn
         heat_plot=seaborn.heatmap((mytab), annot=True)
         heat_plot.get_figure().savefig("heatmap_output.png", bbox_inches='tight')
```



1.6 Correlation and covariance

You can compute correlations in Pandas using the *corr* and *cov* functions.

```
In [38]: df.corr()
```

```
Out[38]:
             id  spent  raised  reelected
id         1.00   0.03    0.03      -0.01
spent      0.03   1.00    0.76       0.00
raised     0.03   0.76    1.00       0.04
reelected -0.01   0.00    0.04       1.00
```

```
In [39]: df.cov()
```

```
Out[39]:
                    id          spent          raised  reelected
id           24255.00      1491398.86      4634556.29      -2.35
spent      1491398.86 118365339902.02 249439228563.81    1719.37
raised     4634556.29 249439228563.81 902442151544.64   64861.30
reelected       -2.35         1719.37        64861.30       2.48
```

For only two variables:

```
In [40]: df["spent"].corr(df["raised"])
```

```
Out[40]: 0.7632077887016929

In [41]: df["spent"].cov(df["raised"])

Out[41]: 249439228563.8146
```

1.7 Hypothesis testing

For the next section we'll import the *scipy.stats* sub-module of *scipy*.

```
In [42]: import scipy.stats as stats
```

We can test if the population mean of data is likely to be equal to a given value (if observations are drawn from a Gaussian distributions of given population mean) by using the scipy.stats.ttest_1samp()function. It returns the T statistic, and the p-value.

```
In [43]: stats.ttest_1samp(df['spent'], 320000)

Out[43]: Ttest_1sampResult(statistic=2.11488812544832, pvalue=0.03489865446172435)
```

Is the mean spending in the House and Senate different, and it the difference statistically significant? We can do a 2-sample t-test with scipy.stats.ttest_ind():

```
In [44]: house_spent = df[df['chamber'] == 'H']['spent']
         senate_spent = df[df['chamber'] == 'S']['spent']

         print("House mean:", house_spent.mean(), "Senate mean:", senate_spent.mean(), stats.t

House mean: 317902.4216589862 Senate mean: 489550.99047619046 Ttest_indResult(statistic=-4.675
```

1.8 Regression analysis

We can use the *ols* function from the *statsmodels* module to conduct OLS regression analysis.

```
In [45]: from statsmodels.formula.api import ols
```

1.8.1 Simple linear regression.

```
In [46]: model1 = ols("spent ~ raised", df).fit()
         print(model1.summary())

                            OLS Regression Results
==============================================================================
Dep. Variable:                  spent   R-squared:                       0.582
Model:                            OLS   Adj. R-squared:                  0.582
Method:                 Least Squares   F-statistic:                     749.2
Date:                Mon, 01 Apr 2019   Prob (F-statistic):          6.36e-104
Time:                        04:51:56   Log-Likelihood:                -7400.4
No. Observations:                 539   AIC:                         1.480e+04
```

```
Df Residuals:                      537   BIC:                         1.481e+04
Df Model:                            1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    1.53e+05    1.2e+04     12.734      0.000    1.29e+05    1.77e+05
raised         0.2764      0.010     27.371      0.000       0.257       0.296
==============================================================================
Omnibus:                      303.175   Durbin-Watson:                   1.973
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            11094.855
Skew:                           1.819   Prob(JB):                         0.00
Kurtosis:                      24.927   Cond. No.                     1.49e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.49e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Saveing the models to txt or csv:

```
In [47]: with open('model1_summary.txt', 'w') as f:
             f.write(model1.summary().as_text())

         with open('model1_summary.csv', 'w') as f:
             f.write(model1.summary().as_csv())
```

Regression with dummy variables.

```
In [48]: model2 = ols("spent ~ chamber", df).fit()
         print(model2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  spent   R-squared:                       0.039
Model:                            OLS   Adj. R-squared:                  0.037
Method:                 Least Squares   F-statistic:                     21.86
Date:                Mon, 01 Apr 2019   Prob (F-statistic):           3.71e-06
Time:                        04:51:59   Log-Likelihood:                -7625.0
No. Observations:                 539   AIC:                         1.525e+04
Df Residuals:                     537   BIC:                         1.526e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    3.179e+05   1.62e+04     19.619      0.000    2.86e+05     3.5e+05
```

```
chamber[T.S]  1.716e+05   3.67e+04      4.676      0.000    9.95e+04    2.44e+05
==============================================================================
Omnibus:                      416.348   Durbin-Watson:                   2.146
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7844.034
Skew:                           3.262   Prob(JB):                         0.00
Kurtosis:                      20.513   Cond. No.                         2.64
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
In [49]: model3 = ols("spent ~ region", df).fit()
         print(model3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  spent   R-squared:                       0.005
Model:                            OLS   Adj. R-squared:                 -0.002
Method:                 Least Squares   F-statistic:                    0.6717
Date:                Mon, 01 Apr 2019   Prob (F-statistic):              0.612
Time:                        04:52:00   Log-Likelihood:                -7634.4
No. Observations:                 539   AIC:                         1.528e+04
Df Residuals:                     534   BIC:                         1.530e+04
Df Model:                           4
Covariance Type:            nonrobust
===============================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------------
Intercept                3.399e+05   3.31e+04     10.254      0.000    2.75e+05    4.05e+05
region[T.North]         -2.264e+04   4.69e+04     -0.483      0.629   -1.15e+05    6.94e+04
region[T.North-East]     4.549e+04    4.7e+04      0.968      0.333   -4.68e+04    1.38e+05
region[T.South]          3.223e+04   4.69e+04      0.688      0.492   -5.98e+04    1.24e+05
region[T.West]           2575.2037   4.69e+04      0.055      0.956   -8.95e+04    9.47e+04
==============================================================================
Omnibus:                      453.621   Durbin-Watson:                   2.082
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            10432.723
Skew:                           3.628   Prob(JB):                         0.00
Kurtosis:                      23.295   Cond. No.                         5.82
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

1.8.2 Multiple regression

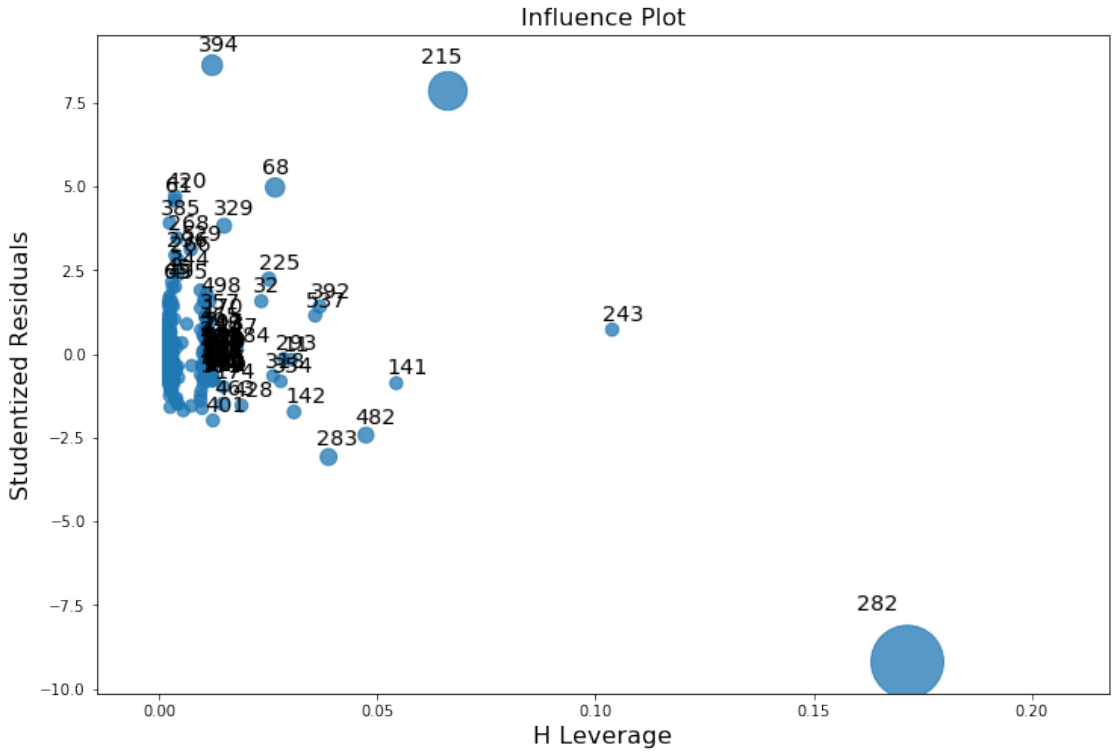
```
In [50]: model4 = ols("spent ~ raised+chamber", df).fit()
         print(model4.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  spent   R-squared:                       0.583
Model:                            OLS   Adj. R-squared:                  0.582
Method:                 Least Squares   F-statistic:                     374.9
Date:                Mon, 01 Apr 2019   Prob (F-statistic):          1.45e-102
Time:                        04:52:01   Log-Likelihood:                -7400.0
No. Observations:                 539   AIC:                         1.481e+04
Df Residuals:                     536   BIC:                         1.482e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     1.501e+05   1.24e+04     12.084      0.000    1.26e+05    1.75e+05
chamber[T.S]  2.255e+04   2.49e+04      0.907      0.365   -2.63e+04    7.14e+04
raised           0.2743      0.010     26.447      0.000       0.254       0.295
==============================================================================
Omnibus:                      299.233   Durbin-Watson:                   1.974
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            10766.758
Skew:                           1.788   Prob(JB):                         0.00
Kurtosis:                      24.602   Cond. No.                     3.12e+06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.12e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

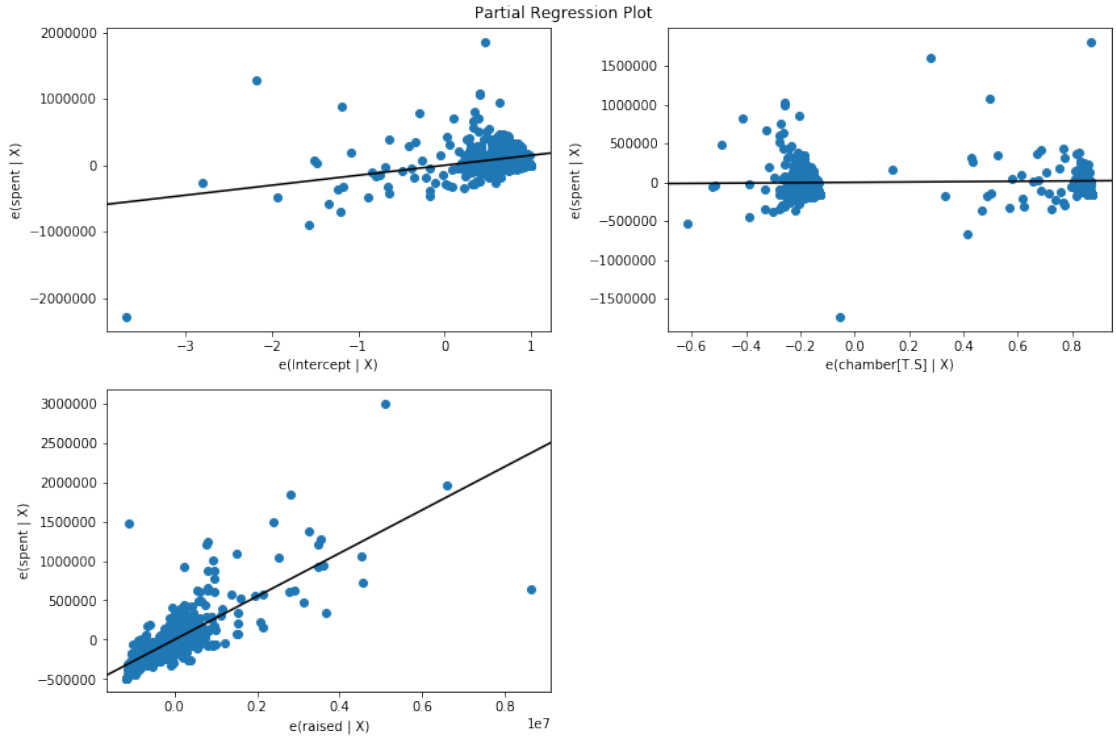
1.8.3 Regression diagnostics.

You can compute regression diagnostics using different funcions in *statsmodels*. You can read more about these here: http://www.statsmodels.org/dev/diagnostic.html. For example:

```
In [56]: #Cook's distance
         import statsmodels.graphics.regressionplots as regplots
         import matplotlib.pyplot as plt
         fig, ax = plt.subplots(figsize=(12,8))
         fig = regplots.influence_plot(model4, ax=ax, criterion="cooks")
```

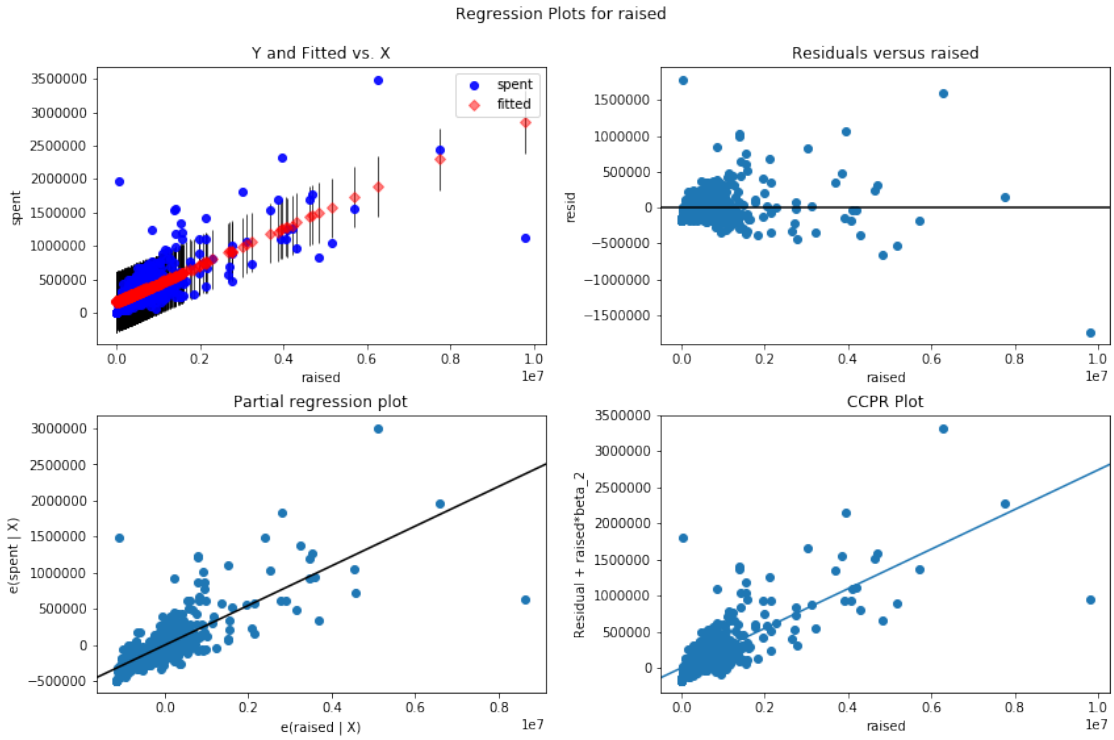


In [57]:
```
fig = plt.figure(figsize=(12,8))
fig = statsmodels.graphics.regressionplots.plot_partregress_grid(model4, fig=fig)
```



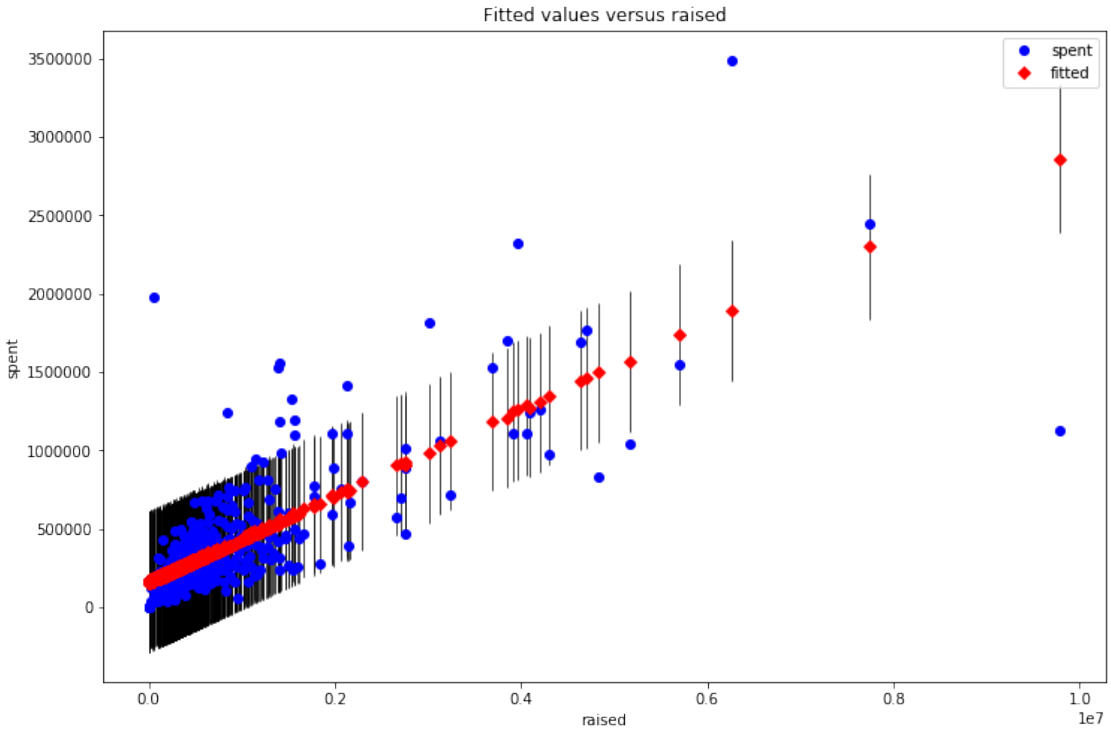
Plot the dependent variable and fitted values with confidence intervals vs. an independent variable, the residuals of the model vs. the chosen independent variable, a partial regression plot, and a CCPR plot. Use function to quickly checking modeling assumptions with respect to a single regressor.

```
In [58]: fig = plt.figure(figsize=(12,8))
         fig = statsmodels.graphics.regressionplots.plot_regress_exog(model4, "raised", fig=fig
```



Fitted values vs a chosen independent variable:

```
In [59]: fig, ax = plt.subplots(figsize=(12, 8))
         fig = statsmodels.graphics.regressionplots.plot_fit(model4, "raised", ax=ax)
```



1.9 Logistic regression

What is the probability that

```
In [60]: import statsmodels.api as sm
         from statsmodels.formula.api import glm
         from pandas.core import datetools
         model5 = glm("chamber~ raised+spent+reelected", df, family=sm.families.Binomial()).fi
         print(model5.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:     ['chamber[H]', 'chamber[S]']   No. Observations:                  537
Model:                                     GLM   Df Residuals:                      533
Model Family:                         Binomial   Df Model:                            3
Link Function:                           logit   Scale:                          1.0000
Method:                                   IRLS   Log-Likelihood:                -252.90
Date:                         Mon, 01 Apr 2019   Deviance:                       505.81
Time:                                 04:55:49   Pearson chi2:                     554.
No. Iterations:                              4   Covariance Type:             nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1.6800      0.211      7.977      0.000       1.267       2.093
```

```
raised      -4.001e-07   1.74e-07     -2.294       0.022   -7.42e-07   -5.83e-08
spent       -3.324e-07   4.67e-07     -0.712       0.476   -1.25e-06    5.82e-07
reelected       0.0949      0.073      1.302       0.193      -0.048       0.238
==============================================================================

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: FutureWarning: The pandas.
  This is separate from the ipykernel package so we can avoid doing imports until
```

1.10 Visualization

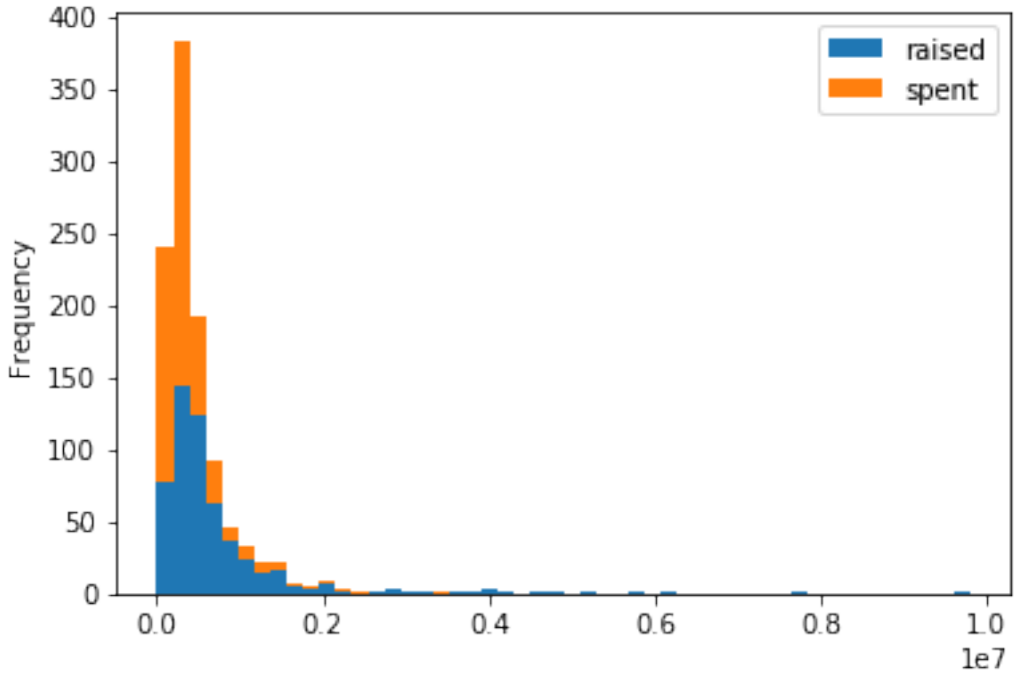
To display graphs inline in Jupyter notebooks make sure you add "%matplotlib inline" in the first cell.

```
In [63]: %matplotlib inline
         #%matplotlib notebook
```

1.10.1 Histograms, comparing two distributions.

```
In [64]: import matplotlib.pyplot as plt
         df_money=df[["raised","spent"]]
         df_money.plot.hist(stacked=True, bins=50)
```

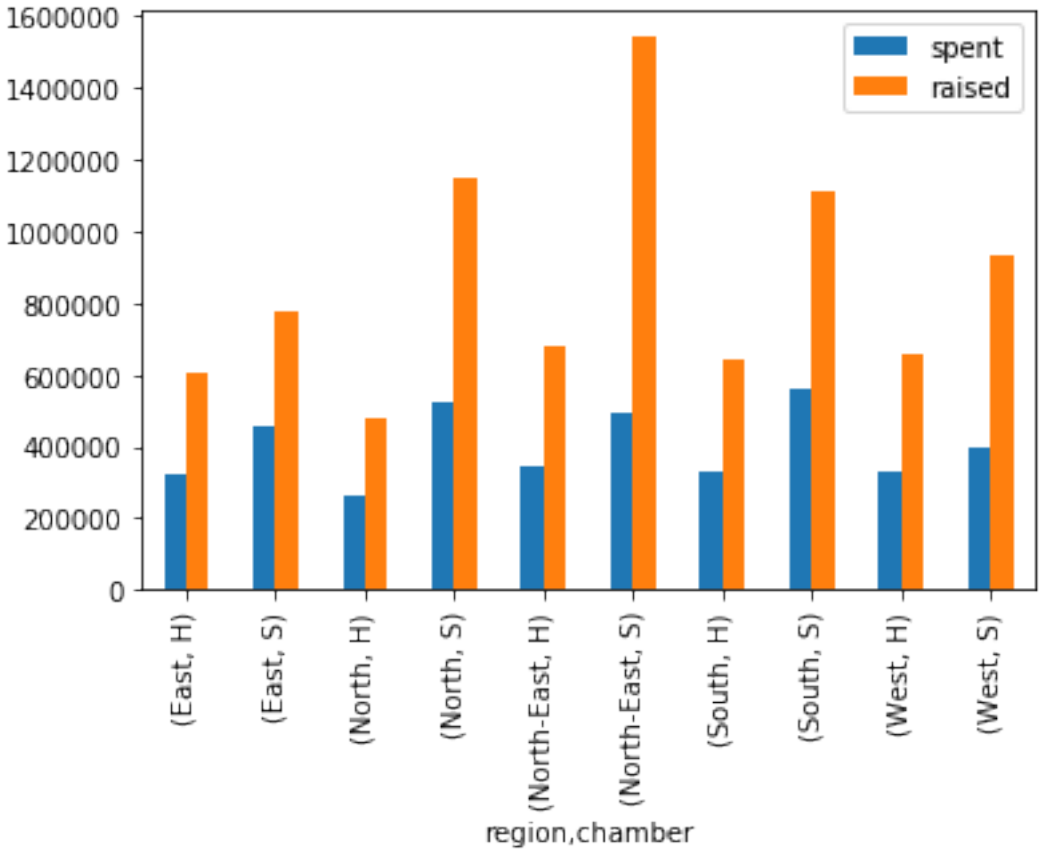
```
Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x1f2f1c63208>
```



1.10.2 Barplots

```
In [65]: agg2=grouped.agg({"spent":"mean",
                           "raised":"mean"})
         agg2.plot.bar()
```

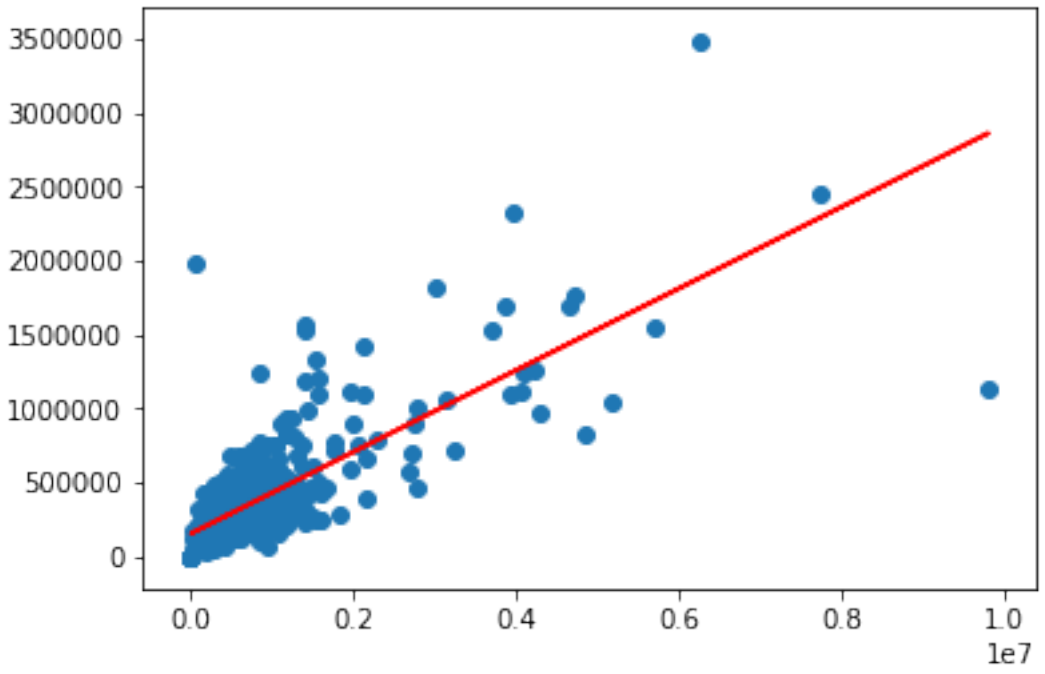
```
Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x1f2f1d20fd0>
```



1.10.3 Scatterplot with linear fit line

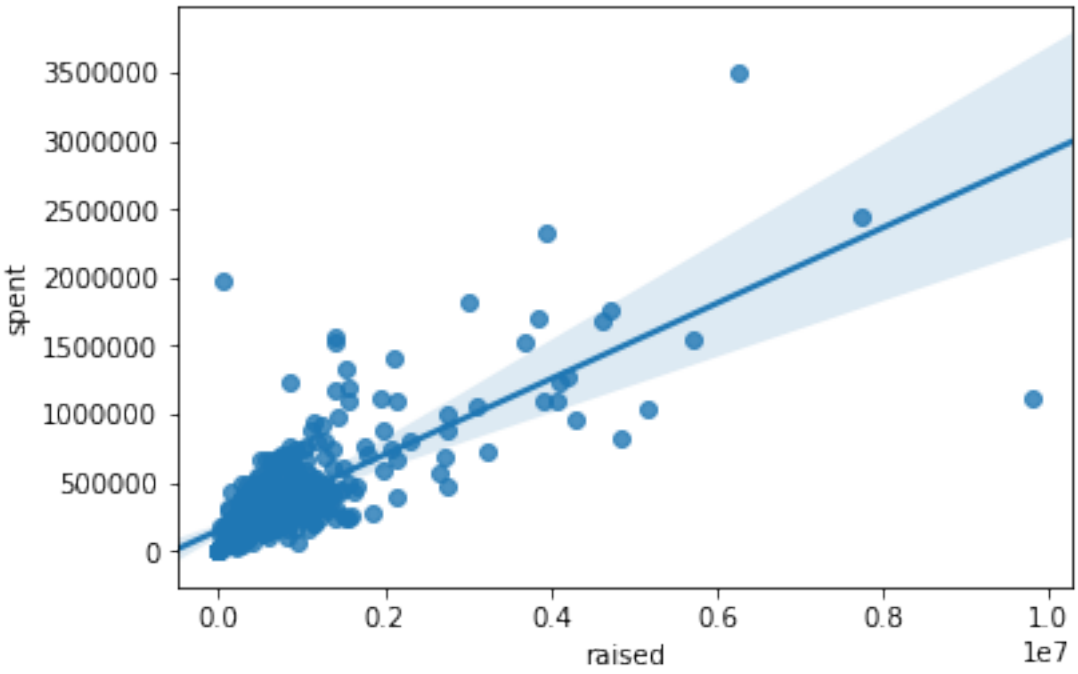
```
In [66]: import numpy as np

         x=df_money.raised.values
         y=df_money.spent.values
         fig, ax = plt.subplots()
         fit = np.polyfit(x, y, deg=1)
         ax.plot(x, fit[0] * x + fit[1], color='red')
         ax.scatter(x, y)
```

```
Out[66]: <matplotlib.collections.PathCollection at 0x1f2eff58240>
```

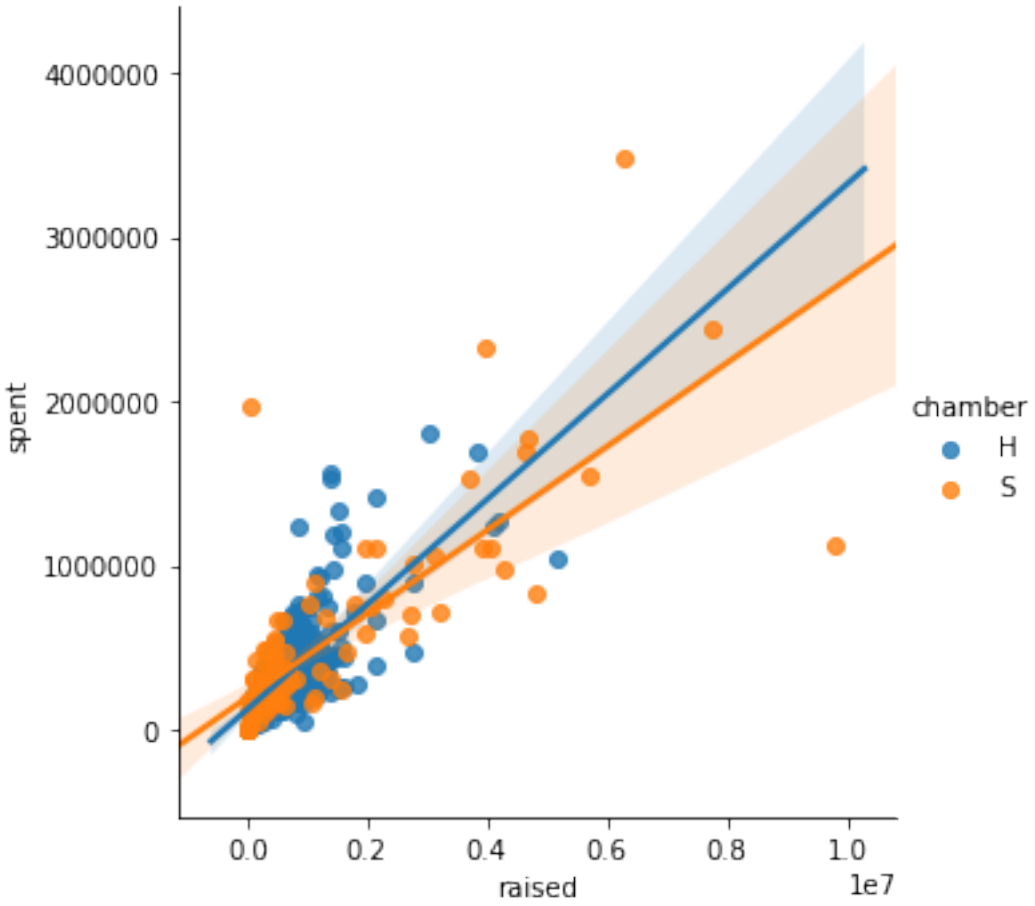


Or using Seaborn:

```
In [68]: seaborn.regplot(x="raised", y="spent", data=df)

Out[68]: <matplotlib.axes._subplots.AxesSubplot at 0x1f2f0c90080>
```

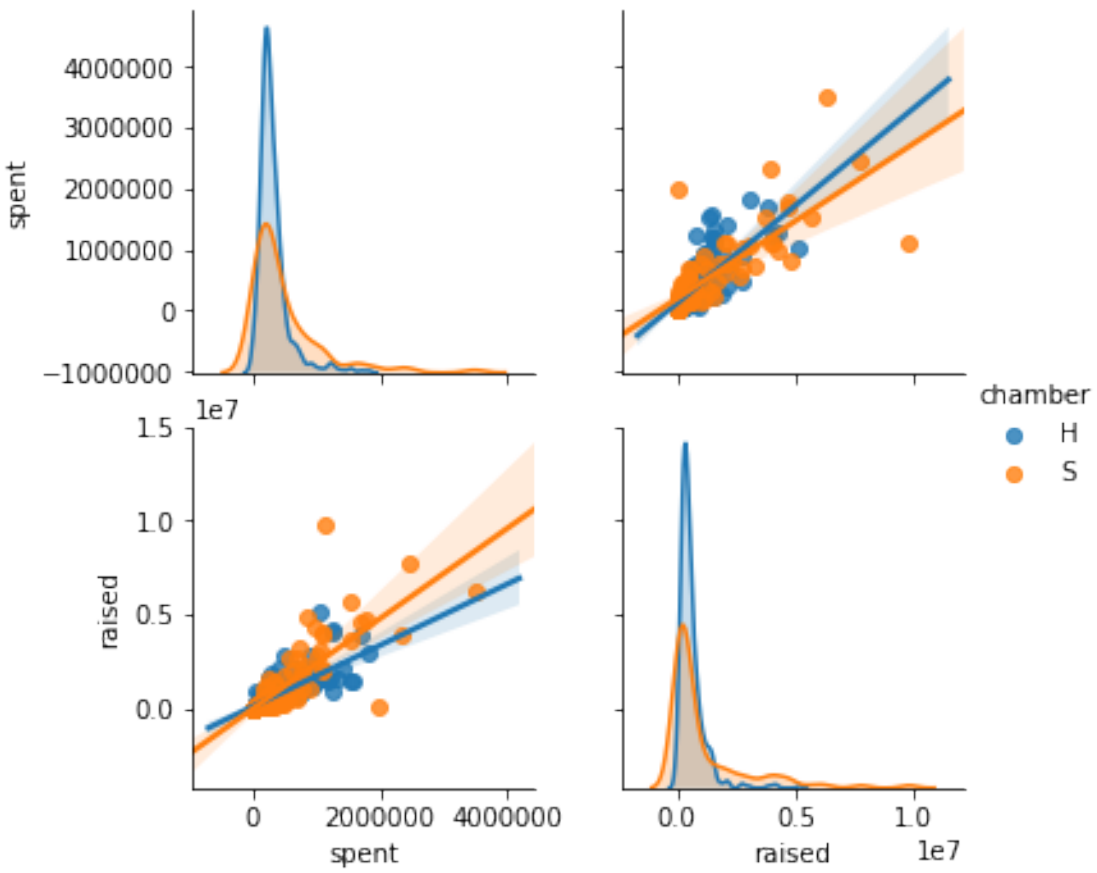


```
In [70]: seaborn.lmplot(x="raised", y="spent", hue="chamber", data=df)

Out[70]: <seaborn.axisgrid.FacetGrid at 0x1f2f0e50cc0>
```



Explore multiple regression relations with the module *seaborn*.

```
In [71]: seaborn.pairplot(df, vars=['spent', 'raised'], kind='reg', hue="chamber")

Out[71]: <seaborn.axisgrid.PairGrid at 0x1f2f0e89908>
```



1.11 Additional resources

Pandas documentation

Statsmodels documentation

Seaborn examples