19-Regression-III-More-Linear

November 15, 2016

1 More on Linear Regression

Today, we'll look at some additional aspects of Linear Regression.

Let's look at a standard regression.

Here are the pieces of a standard linear model.

```
In [2]: nsample = 100
        x = np.linspace(0, 10, 100)
        #print(x)
        X = np.column_stack((x, x**2))
        print(X)
        beta = np.array([1, 0.1, 10])
        e = np.random.normal(size=nsample)
```

```
[[  0.00000000e+00   0.00000000e+00]
 [  1.01010101e-01   1.02030405e-02]
 [  2.02020202e-01   4.08121620e-02]
 [  3.03030303e-01   9.18273646e-02]
 [  4.04040404e-01   1.63248648e-01]
 [  5.05050505e-01   2.55076013e-01]
 [  6.06060606e-01   3.67309458e-01]
 [  7.07070707e-01   4.99948985e-01]
 [  8.08080808e-01   6.52994592e-01]
 [  9.09090909e-01   8.26446281e-01]
 [  1.01010101e+00   1.02030405e+00]
 [  1.11111111e+00   1.23456790e+00]
 [  1.21212121e+00   1.46923783e+00]
 [  1.31313131e+00   1.72431385e+00]
 [  1.41414141e+00   1.99979594e+00]
 [  1.51515152e+00   2.29568411e+00]
 [  1.61616162e+00   2.61197837e+00]
 [  1.71717172e+00   2.94867871e+00]
 [  1.81818182e+00   3.30578512e+00]
 [  1.91919192e+00   3.68329762e+00]
 [  2.02020202e+00   4.08121620e+00]
 [  2.12121212e+00   4.49954086e+00]
 [  2.22222222e+00   4.93827160e+00]
```

```
[  2.32323232e+00   5.39740843e+00]
[  2.42424242e+00   5.87695133e+00]
[  2.52525253e+00   6.37690032e+00]
[  2.62626263e+00   6.89725538e+00]
[  2.72727273e+00   7.43801653e+00]
[  2.82828283e+00   7.99918376e+00]
[  2.92929293e+00   8.58075707e+00]
[  3.03030303e+00   9.18273646e+00]
[  3.13131313e+00   9.80512193e+00]
[  3.23232323e+00   1.04479135e+01]
[  3.33333333e+00   1.11111111e+01]
[  3.43434343e+00   1.17947148e+01]
[  3.53535354e+00   1.24987246e+01]
[  3.63636364e+00   1.32231405e+01]
[  3.73737374e+00   1.39679625e+01]
[  3.83838384e+00   1.47331905e+01]
[  3.93939394e+00   1.55188246e+01]
[  4.04040404e+00   1.63248648e+01]
[  4.14141414e+00   1.71513111e+01]
[  4.24242424e+00   1.79981635e+01]
[  4.34343434e+00   1.88654219e+01]
[  4.44444444e+00   1.97530864e+01]
[  4.54545455e+00   2.06611570e+01]
[  4.64646465e+00   2.15896337e+01]
[  4.74747475e+00   2.25385165e+01]
[  4.84848485e+00   2.35078053e+01]
[  4.94949495e+00   2.44975003e+01]
[  5.05050505e+00   2.55076013e+01]
[  5.15151515e+00   2.65381084e+01]
[  5.25252525e+00   2.75890215e+01]
[  5.35353535e+00   2.86603408e+01]
[  5.45454545e+00   2.97520661e+01]
[  5.55555556e+00   3.08641975e+01]
[  5.65656566e+00   3.19967350e+01]
[  5.75757576e+00   3.31496786e+01]
[  5.85858586e+00   3.43230283e+01]
[  5.95959596e+00   3.55167840e+01]
[  6.06060606e+00   3.67309458e+01]
[  6.16161616e+00   3.79655137e+01]
[  6.26262626e+00   3.92204877e+01]
[  6.36363636e+00   4.04958678e+01]
[  6.46464646e+00   4.17916539e+01]
[  6.56565657e+00   4.31078461e+01]
[  6.66666667e+00   4.44444444e+01]
[  6.76767677e+00   4.58014488e+01]
[  6.86868687e+00   4.71788593e+01]
[  6.96969697e+00   4.85766758e+01]
[  7.07070707e+00   4.99948985e+01]
```

```
 [  7.17171717e+00   5.14335272e+01]
 [  7.27272727e+00   5.28925620e+01]
 [  7.37373737e+00   5.43720029e+01]
 [  7.47474747e+00   5.58718498e+01]
 [  7.57575758e+00   5.73921028e+01]
 [  7.67676768e+00   5.89327620e+01]
 [  7.77777778e+00   6.04938272e+01]
 [  7.87878788e+00   6.20752984e+01]
 [  7.97979798e+00   6.36771758e+01]
 [  8.08080808e+00   6.52994592e+01]
 [  8.18181818e+00   6.69421488e+01]
 [  8.28282828e+00   6.86052444e+01]
 [  8.38383838e+00   7.02887460e+01]
 [  8.48484848e+00   7.19926538e+01]
 [  8.58585859e+00   7.37169677e+01]
 [  8.68686869e+00   7.54616876e+01]
 [  8.78787879e+00   7.72268136e+01]
 [  8.88888889e+00   7.90123457e+01]
 [  8.98989899e+00   8.08182838e+01]
 [  9.09090909e+00   8.26446281e+01]
 [  9.19191919e+00   8.44913784e+01]
 [  9.29292929e+00   8.63585348e+01]
 [  9.39393939e+00   8.82460973e+01]
 [  9.49494949e+00   9.01540659e+01]
 [  9.59595960e+00   9.20824406e+01]
 [  9.69696970e+00   9.40312213e+01]
 [  9.79797980e+00   9.60004081e+01]
 [  9.89898990e+00   9.79900010e+01]
 [  1.00000000e+01   1.00000000e+02]]
```

We add one more column of ones (1) so that we can estimate the intercept.
We then go on to create data that comes from a particular model.

```
In [3]: X = sm.add_constant(X)
        y = np.dot(X, beta) + e
```

Now we have the explanatory variable X (predictor, regressor, input variable) and the response variable y (predicted, regressand, output).
We are ready to fit the model.

```
In [4]: model = sm.OLS(y, X)
        results = model.fit()
        print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
```

```
Method:                 Least Squares   F-statistic:                 4.020e+06
Date:                Tue, 15 Nov 2016   Prob (F-statistic):          2.83e-239
Time:                        10:16:58   Log-Likelihood:                -146.51
No. Observations:                 100   AIC:                             299.0
Df Residuals:                      97   BIC:                             306.8
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          1.3423      0.313      4.292      0.000         0.722     1.963
x1            -0.0402      0.145     -0.278      0.781        -0.327     0.247
x2            10.0103      0.014    715.745      0.000         9.982    10.038
==============================================================================
Omnibus:                        2.042   Durbin-Watson:                   2.274
Prob(Omnibus):                  0.360   Jarque-Bera (JB):                1.875
Skew:                           0.234   Prob(JB):                        0.392
Kurtosis:                       2.519   Cond. No.                         144.
==============================================================================

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

Let's look at the various aspects of the fitted model.

```
In [5]: #dir(results)
        print('Parameters: ', results.params)
        print('R2: ', results.rsquared)
        print('Predicted values: ', results.predict())
```

```
Parameters:  [  1.34233516  -0.04024948  10.01025357]
R2:  0.999987936503
Predicted values:  [    1.34233516     1.44040458     1.74274404     2.24935355
     3.87538271     4.99480235     6.31849204     7.84645178     9.57868156
    11.51518139    13.65595126    16.00099118    18.55030114    21.30388115
    24.2617312     27.4238513     30.79024145    34.36090164    38.13583187
    42.11503215    46.29850248    50.68624285    55.27825327    60.07453373
    65.07508424    70.27990479    75.68899539    81.30235603    87.11998672
    93.14188746    99.36805824   105.79849906   112.43320993   119.27219085
   126.31544181   133.56296282   141.01475387   148.67081497   156.53114611
   164.5957473    172.86461853   181.33775981   190.01517113   198.8968525
   207.98280392   217.27302538   226.76751688   236.46627843   246.36931003
   256.47661167   266.78818336   277.30402509   288.02413687   298.94851869
   310.07717056   321.41009247   332.94728443   344.68874643   356.63447848
   368.78448058   381.13875272   393.69729491   406.46010714   419.42718941
   432.59854174   445.9741641    459.55405651   473.33821897   487.32665148
   501.51935402   515.91632662   530.51756926   545.32308194   560.33286467
```

```
575.54691745   590.96524027   606.58783314   622.41469605   638.445829
654.68123201   671.12090505   687.76484815   704.61306128   721.66554447
738.9222977    756.38332097   774.04861429   791.91817766   809.99201107
828.27011452   846.75248802   865.43913157   884.33004516   903.4252288
922.72468248   942.22840621   961.93639998   981.8486638   1001.96519767]
```

Now we'll load a standard dataset as an example.

The Longley dataset contains various US macroeconomic variables that are known to be highly collinear. It has been used to appraise the accuracy of least squares routines.

```
In [6]: from statsmodels.datasets.longley import load_pandas
        y = load_pandas().endog
        X = load_pandas().exog
        X = sm.add_constant(X)
```

```
In [7]: ols_model = sm.OLS(y, X)
        ols_results = ols_model.fit()
        print(ols_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 TOTEMP   R-squared:                       0.995
Model:                            OLS   Adj. R-squared:                  0.992
Method:                 Least Squares   F-statistic:                     330.3
Date:                Tue, 15 Nov 2016   Prob (F-statistic):           4.98e-10
Time:                        10:16:58   Log-Likelihood:                -109.62
No. Observations:                  16   AIC:                             233.2
Df Residuals:                       9   BIC:                             238.6
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const      -3.482e+06    8.9e+05     -3.911      0.004      -5.5e+06 -1.47e+06
GNPDEFL       15.0619     84.915      0.177      0.863      -177.029   207.153
GNP           -0.0358      0.033     -1.070      0.313        -0.112     0.040
UNEMP         -2.0202      0.488     -4.136      0.003        -3.125    -0.915
ARMED         -1.0332      0.214     -4.822      0.001        -1.518    -0.549
POP           -0.0511      0.226     -0.226      0.826        -0.563     0.460
YEAR        1829.1515    455.478      4.016      0.003       798.788  2859.515
==============================================================================
Omnibus:                        0.749   Durbin-Watson:                   2.559
Prob(Omnibus):                  0.688   Jarque-Bera (JB):                0.684
Skew:                           0.420   Prob(JB):                        0.710
Kurtosis:                       2.434   Cond. No.                     4.86e+09
==============================================================================

Warnings:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
[2] The condition number is large, 4.86e+09. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
/Users/crovella/anaconda3/lib/python3.5/site-packages/scipy/stats/stats.py:1327: Us
  "anyway, n=%i" % int(n))
```

What does this mean?

> In statistics, multicollinearity (also collinearity) is a phenomenon in which two or more predictor variables in a multiple regression model are highly correlated, meaning that one can be linearly predicted from the others with a substantial degree of accuracy.

(Wikipedia)
The condition number being referred to is the condition number of the design matrix.
That is the X in $X\beta = y$.
If X did not have full column rank, then there would be multiple solutions to the normal equations:

$$X^T X\beta = X^T y.$$

The solutions are in a sense all equivalent in that they yield the same value of $\|X\beta - y\|$.
However, the actual values of β can vary tremendously and so it is not clear how best to interpret the case when X does not have full column rank.
This happens when the columns of X are not linearly independent, ie, one column can be expressed as a linear combination of the other columns.
Condition number is a measure of whether X is **nearly** lacking full column rank.
In other words, whether some column is **close to** being a linear combination of the other columns.
Even in this case, the actual values of β can vary a lot due to the limitations of numerical precision in the computer.
This does not happen too often in practice, but there are some things that can be done if it does happen.
We can try to find one of the offending predictor variables, and leave it out of the regression.

1.1 Influence

Also, it can happen that dropping a single observation can have a dramatic effect on the coefficient estimates.
Restricting ourselves to the first 14 observations:

```
In [8]: ols_results2 = sm.OLS(y.ix[:14], X.ix[:14]).fit()
```

Let us see how much each parameter changes if we compare to all 16 observations:

```
In [9]: print("Percentage change %4.2f%%\n"*7 % tuple([i for i in (ols_results2.par
```

```
Percentage change -13.35%
Percentage change -236.18%
Percentage change -23.69%
Percentage change -3.36%
Percentage change -7.26%
Percentage change -200.46%
Percentage change -13.34%
```

Formal statistics for this such as the DFBETAS – a standardized measure of how much each coefficient changes when that observation is left out.

An observation is considered troublesome if its DFBETA is greater than $2/\sqrt{\text{number of observations}}$.

```
In [10]: infl = ols_results.get_influence()
         ## significant influence
         2./len(X)**.5
```

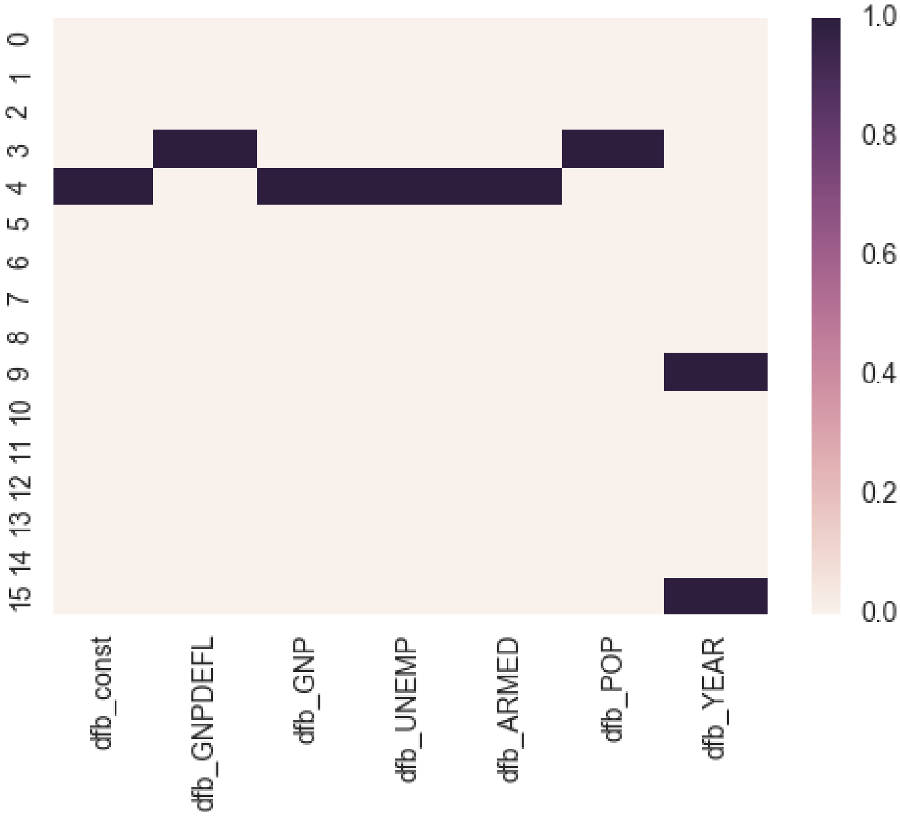
```
Out[10]: 0.5
```

```
In [11]: print(infl.summary_frame().filter(regex="dfb"))
```

	dfb_const	dfb_GNPDEFL	dfb_GNP	dfb_UNEMP	dfb_ARMED	dfb_POP	dfb_YEAR
0	-0.016406	-0.234566	-0.045095	-0.121513	-0.149026	0.211057	0.013388
1	-0.020608	-0.289091	0.124453	0.156964	0.287700	-0.161890	0.025958
2	-0.008382	0.007161	-0.016799	0.009575	0.002227	0.014871	0.008103
3	0.018093	0.907968	-0.500022	-0.495996	0.089996	0.711142	-0.040056
4	1.871260	-0.219351	1.611418	1.561520	1.169337	-1.081513	-1.864186
5	-0.321373	-0.077045	-0.198129	-0.192961	-0.430626	0.079916	0.323275
6	0.315945	-0.241983	0.438146	0.471797	-0.019546	-0.448515	-0.307517
7	0.015816	-0.002742	0.018591	0.005064	-0.031320	-0.015823	-0.015583
8	-0.004019	-0.045687	0.023708	0.018125	0.013683	-0.034770	0.005116
9	-1.018242	-0.282131	-0.412621	-0.663904	-0.715020	-0.229501	1.035723
10	0.030947	-0.024781	0.029480	0.035361	0.034508	-0.014194	-0.030805
11	0.005987	-0.079727	0.030276	-0.008883	-0.006854	-0.010693	-0.005323
12	-0.135883	0.092325	-0.253027	-0.211465	0.094720	0.331351	0.129120
13	0.032736	-0.024249	0.017510	0.033242	0.090655	0.007634	-0.033114
14	0.305868	0.148070	0.001428	0.169314	0.253431	0.342982	-0.318031
15	-0.538323	0.432004	-0.261262	-0.143444	-0.360890	-0.467296	0.552421

```
In [12]: sns.heatmap(infl.summary_frame().filter(regex="dfb") > 0.5 )
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x11aa379e8>
```



1.2 Flexible Modeling

The Guerry dataset is a collection of historical data used in support of Andre-Michel Guerry's 1833 "Essay on the Moral Statistics of France."

> Andre-Michel Guerry's (1833) Essai sur la Statistique Morale de la France was one of the foundation studies of modern social science. Guerry assembled data on crimes, suicides, literacy and other "moral statistics," and used tables and maps to analyze a variety of social issues in perhaps the first comprehensive study relating such variables.

Wikipedia

> Guerry's results were startling for two reasons. First he showed that rates of crime and suicide remained remarkably stable over time, when broken down by age, sex, region of France and even season of the year; yet these numbers varied systematically across departements of France. This regularity of social numbers created the possibility to conceive, for the first time, that human actions in the social world were governed by social laws, just as inanimate objects were governed by laws of the physical world.

Source: "A.-M. Guerry's Moral Statistics of France: Challenges for Multivariable Spatial Analysis", Michael Friendly. Statistical Science 2007, Vol. 22, No. 3, 368–399.

```
In [13]: # Lottery is per-capital wager on Royal Lottery
         df = sm.datasets.get_rdataset("Guerry", "HistData").data
         df = df[['Lottery', 'Literacy', 'Wealth', 'Region']].dropna()
         df.head()
```

```
Out[13]:    Lottery  Literacy  Wealth Region
         0       41        37      73      E
         1       38        51      22      N
         2       66        13      61      C
         3       80        46      76      E
         4       79        69      83      E
```

We can use another version of the module that can directly type formulas and expressions in the functions of the models.

We can specify the name of the columns to be used to predict another column, remove columns, etc.

```
In [14]: mod = smf.ols(formula='Lottery ~ Literacy + Wealth + Region', data=df)
         res = mod.fit()
         print(res.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Lottery   R-squared:                       0.338
Model:                            OLS   Adj. R-squared:                  0.287
Method:                 Least Squares   F-statistic:                     6.636
Date:                Tue, 15 Nov 2016   Prob (F-statistic):           1.07e-05
Time:                        10:16:59   Log-Likelihood:                -375.30
No. Observations:                  85   AIC:                             764.6
Df Residuals:                      78   BIC:                             781.7
Df Model:                           6
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [95.0% Conf. Int.]
-------------------------------------------------------------------------------
Intercept      38.6517      9.456      4.087      0.000        19.826    57.478
Region[T.E]   -15.4278      9.727     -1.586      0.117       -34.793     3.938
Region[T.N]   -10.0170      9.260     -1.082      0.283       -28.453     8.419
Region[T.S]    -4.5483      7.279     -0.625      0.534       -19.039     9.943
Region[T.W]   -10.0913      7.196     -1.402      0.165       -24.418     4.235
Literacy       -0.1858      0.210     -0.886      0.378        -0.603     0.232
Wealth          0.4515      0.103      4.390      0.000         0.247     0.656
==============================================================================
Omnibus:                        3.049   Durbin-Watson:                   1.785
Prob(Omnibus):                  0.218   Jarque-Bera (JB):                2.694
Skew:                          -0.340   Prob(JB):                        0.260
Kurtosis:                       2.454   Cond. No.                         371.
==============================================================================
```

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

Categorical variables

Patsy is the name of the interpreter that parses the formulas.

Looking at the summary printed above, notice that patsy determined that elements of Region were text strings, so it treated Region as a categorical variable. patsy's default is also to include an intercept, so we automatically dropped one of the Region categories.

Removing variables

The "-" sign can be used to remove columns/variables. For instance, we can remove the intercept from a model by:

```
In [15]: res = smf.ols(formula='Lottery ~ Literacy + Wealth + C(Region) -1 ', data=
         print(res.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Lottery   R-squared:                       0.338
Model:                            OLS   Adj. R-squared:                  0.287
Method:                 Least Squares   F-statistic:                     6.636
Date:                Tue, 15 Nov 2016   Prob (F-statistic):           1.07e-05
Time:                        10:17:00   Log-Likelihood:                -375.30
No. Observations:                  85   AIC:                             764.6
Df Residuals:                      78   BIC:                             781.7
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [95.0% Conf. Int.]
--------------------------------------------------------------------------------
C(Region)[C]    38.6517      9.456      4.087      0.000        19.826    57.478
C(Region)[E]    23.2239     14.931      1.555      0.124        -6.501    52.949
C(Region)[N]    28.6347     13.127      2.181      0.032         2.501    54.769
C(Region)[S]    34.1034     10.370      3.289      0.002        13.459    54.748
C(Region)[W]    28.5604     10.018      2.851      0.006         8.616    48.505
Literacy        -0.1858      0.210     -0.886      0.378        -0.603     0.232
Wealth           0.4515      0.103      4.390      0.000         0.247     0.656
==============================================================================
Omnibus:                        3.049   Durbin-Watson:                   1.785
Prob(Omnibus):                  0.218   Jarque-Bera (JB):                2.694
Skew:                          -0.340   Prob(JB):                        0.260
Kurtosis:                       2.454   Cond. No.                         653.
==============================================================================

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

Functions

We can also apply vectorized functions to the variables in our model:

```
In [16]: res = smf.ols(formula='Lottery ~ np.log(Literacy)', data=df).fit()
         print(res.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Lottery   R-squared:                       0.161
Model:                            OLS   Adj. R-squared:                  0.151
Method:                 Least Squares   F-statistic:                     15.89
Date:                Tue, 15 Nov 2016   Prob (F-statistic):           0.000144
Time:                        10:17:00   Log-Likelihood:                -385.38
No. Observations:                  85   AIC:                             774.8
Df Residuals:                      83   BIC:                             779.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                       coef    std err          t      P>|t|      [95.0% Conf. Int.
------------------------------------------------------------------------------
Intercept          115.6091     18.374      6.292      0.000        79.064   152.15
np.log(Literacy)   -20.3940      5.116     -3.986      0.000       -30.570   -10.21
==============================================================================
Omnibus:                        8.907   Durbin-Watson:                   2.019
Prob(Omnibus):                  0.012   Jarque-Bera (JB):                3.299
Skew:                           0.108   Prob(JB):                        0.192
Kurtosis:                       2.059   Cond. No.                         28.7
==============================================================================

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

1.3 Understanding Problematic Observations

```
In [17]: from statsmodels.formula.api import ols
         from statsmodels.compat import lzip
         prestige = sm.datasets.get_rdataset("Duncan", "car", cache=True).data
```

This is data on the "prestige" and other characteristics of 45 U. S. occupations in 1950.

- type: Type of occupation. A factor with the following levels: `prof`, professional and managerial; `wc`, white-collar; `bc`, blue-collar.
- income: Percent of males in occupation earning USD 3500 or more in 1950.
- education: Percent of males in occupation in 1950 who were high-school graduates.
- prestige: Percent of raters in NORC study rating occupation as excellent or good in prestige.

```
In [18]: prestige.head()
```

```
Out[18]:             type  income  education  prestige
         accountant  prof      62         86        82
         pilot       prof      72         76        83
         architect   prof      75         92        90
         author      prof      55         90        76
         chemist     prof      64         86        90
```

```
In [19]: prestige_model = ols("prestige ~ income + education", data=prestige).fit()
         print(prestige_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               prestige   R-squared:                       0.828
Model:                            OLS   Adj. R-squared:                  0.820
Method:                 Least Squares   F-statistic:                     101.2
Date:                Tue, 15 Nov 2016   Prob (F-statistic):           8.65e-17
Time:                        10:17:00   Log-Likelihood:                -178.98
No. Observations:                  45   AIC:                             364.0
Df Residuals:                      42   BIC:                             369.4
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept     -6.0647      4.272     -1.420      0.163       -14.686     2.556
income         0.5987      0.120      5.003      0.000         0.357     0.840
education      0.5458      0.098      5.555      0.000         0.348     0.744
==============================================================================
Omnibus:                        1.279   Durbin-Watson:                   1.458
Prob(Omnibus):                  0.528   Jarque-Bera (JB):                0.520
Skew:                           0.155   Prob(JB):                        0.771
Kurtosis:                       3.426   Cond. No.                         163.
==============================================================================

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

Influence plots

To ask about the impact of individual observations on the overall model, we can calculate the `influence` of each observation.

`leverage` is a measure of how far away the independent variable values of an observation are from those of the other observations.

The `residual` is the error obtained when predicting the observation.

Influence plots show the studentized residuals vs. the leverage of each observation as measured by the projection matrix.

Externally studentized residuals are residuals that are scaled by their standard deviation where

$$\mathrm{var}(\hat{\epsilon}_i) = \hat{\sigma}^2(1 - h_{ii}).$$

h_{ii} is the i-th diagonal element of the projection matrix

$$H = X(X^T X)^{-1} X^T$$

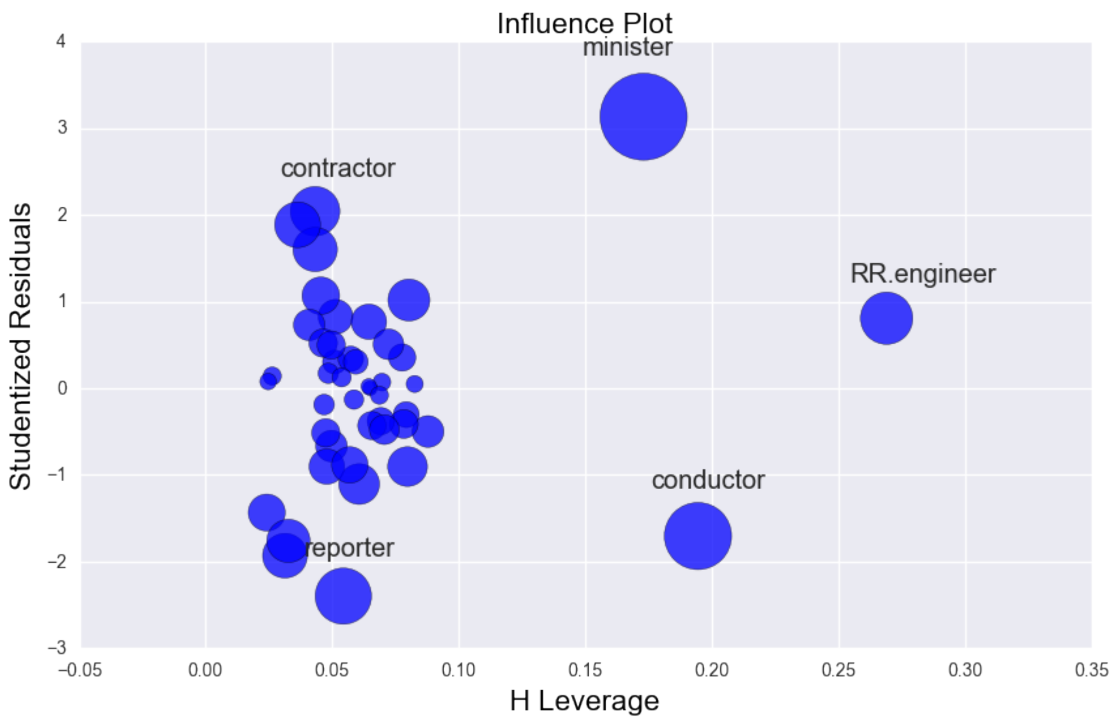
where X is the design matrix.

It can be shown that

$$h_{ii} = \frac{\partial \hat{y}_i}{\partial y_i}.$$

The influence of each point can be visualized by the criterion keyword argument. Options are Cook's distance and DFFITS, two measures of influence. Basically these methods combine leverage and residual.

```
In [20]: fig, ax = plt.subplots(figsize=(10,6))
         fig = sm.graphics.influence_plot(prestige_model, ax=ax, criterion="cooks")
```

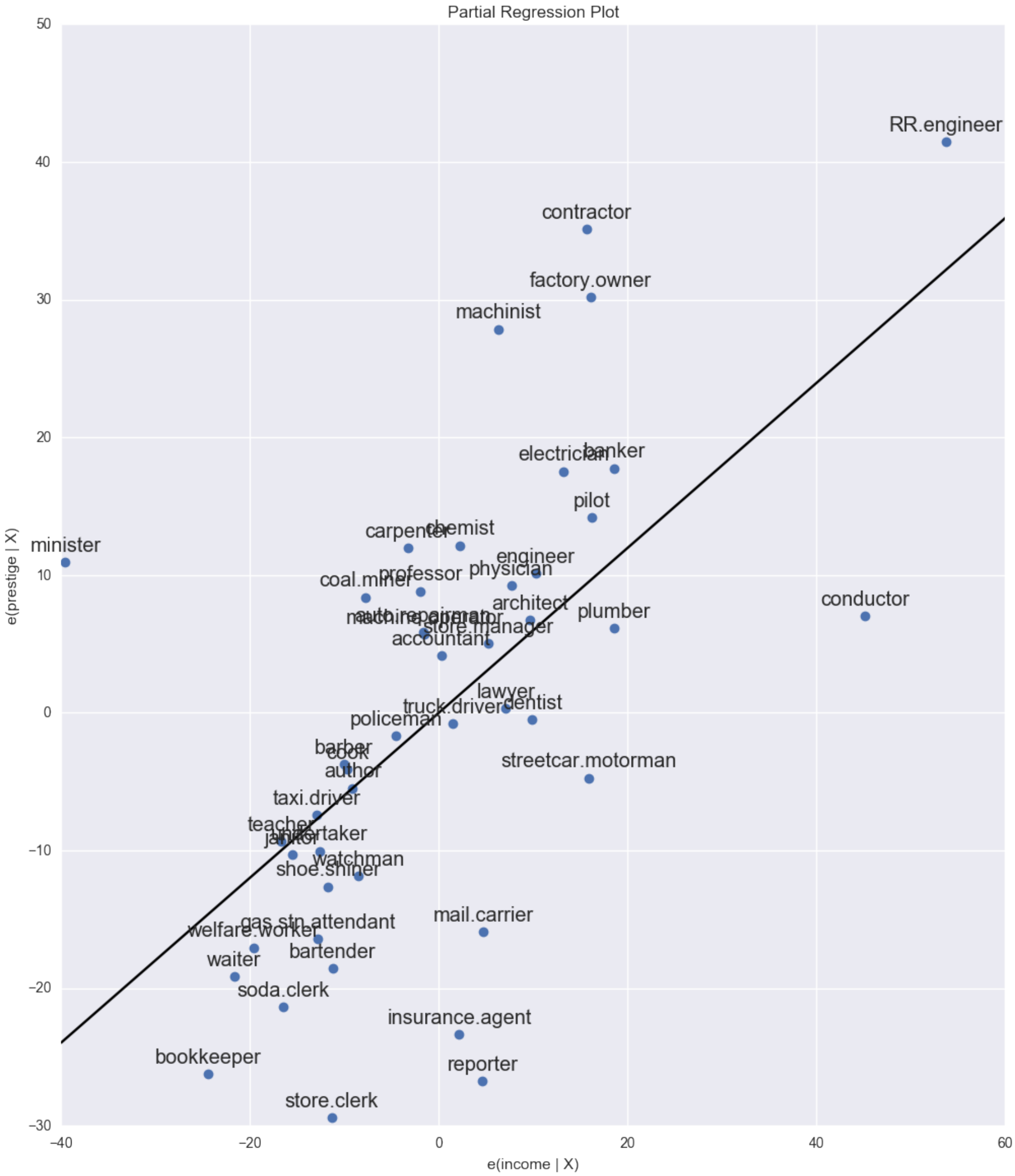


As you can see there are a few worrisome observations. Both contractor and reporter have low leverage but a large residual. RR.engineer has small residual and large leverage. Conductor and minister have both high leverage and large residuals, and, therefore, large influence.

Partial Regression Plots are a principled way of looking at individual independent variables versus the dependent variable, for visualization.

In a partial regression plot, to discern the relationship between the response variable and the k-th variable, we compute the residuals by regressing the response variable versus the independent variables excluding X_k. We can denote this by $X_{\sim k}$. We then compute the residuals by regressing X_k on $X_{\sim k}$. The partial regression plot is the plot of the former versus the latter residuals.

```
In [21]: fix, ax = plt.subplots(figsize=(12,14))
         fig = sm.graphics.plot_partregress("prestige", "income", ["education"], da
```



As you can see the partial regression plot confirms the influence of conductor, minister, and RR.engineer on the partial relationship between income and prestige. The cases greatly decrease the effect of income on prestige. Dropping these cases confirms this.

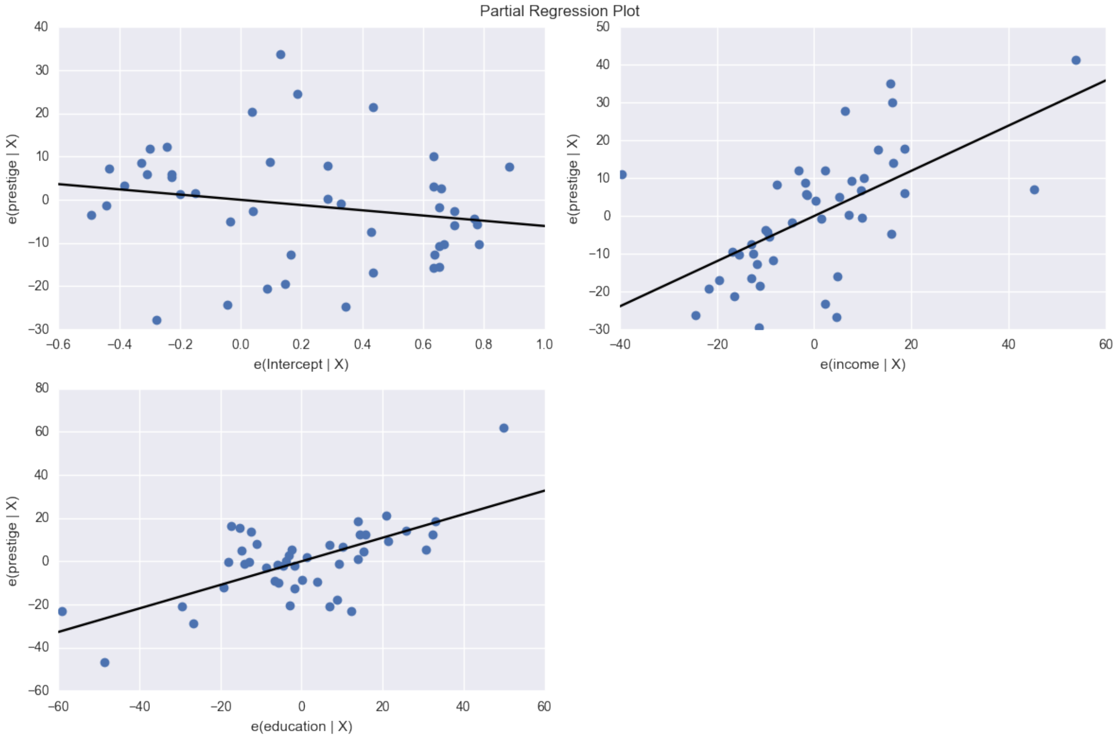
```
In [22]: subset = ~prestige.index.isin(["conductor", "RR.engineer", "minister"])
         prestige_model2 = ols("prestige ~ income + education", data=prestige, subs
         print(prestige_model2.summary())

                            OLS Regression Results
==============================================================================
```

```
Dep. Variable:                prestige   R-squared:                       0.876
Model:                             OLS   Adj. R-squared:                  0.870
Method:                  Least Squares   F-statistic:                     138.1
Date:                 Tue, 15 Nov 2016   Prob (F-statistic):           2.02e-18
Time:                         10:17:01   Log-Likelihood:                -160.59
No. Observations:                   42   AIC:                             327.2
Df Residuals:                       39   BIC:                             332.4
Df Model:                            2
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept     -6.3174      3.680     -1.717      0.094       -13.760     1.125
income         0.9307      0.154      6.053      0.000         0.620     1.242
education      0.2846      0.121      2.345      0.024         0.039     0.530
==============================================================================
Omnibus:                         3.811   Durbin-Watson:                   1.468
Prob(Omnibus):                   0.149   Jarque-Bera (JB):                2.802
Skew:                           -0.614   Prob(JB):                        0.246
Kurtosis:                        3.303   Cond. No.                         158.
==============================================================================

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

```
In [23]: fig = plt.figure(figsize=(12,8))
         fig = sm.graphics.plot_partregress_grid(prestige_model, fig=fig)
```


Partial Regression Plot
e(prestige | X)
e(Intercept | X)
e(income | X)
e(education | X)