OLS Return Predictability Analysis

In this notebook we will analyze some of the more popular variables used to predict equity market returns.

The packages I will use are mostly the same as before. Regressions are done with statsmodels.api. This is slightly different from statsmodels.formula.api. They each have an OLS regression function, but they are used in different ways. Both are useful in certain situations.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

The data we're going to read in comes from Amit Goyal's website. It includes a number of variables that are useful in predicting future returns. It also includes the returns themselves.

```
predictors = pd.read_excel('PredictorData2023.xlsx',sheet_name='Monthly',index_col=[0])
predictors.head()
```

```
C:\Users\shaneshe\Anaconda3\lib\site-packages\openpyxl\worksheet\header_footer.py:48: UserWarning: Cannot parse header or fo
  warn("""Cannot parse header or footer so it will be ignored""")
```

	price	d12	e12	ret	retx	AAA	BAA	lty	ltr	corpr	...	ygap	rdsp	rsvix	gpce	gip	tchi	house	avgcor	shtint	d
yyyymm																					
187101	4.44	0.26	0.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
187102	4.50	0.26	0.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
187103	4.61	0.26	0.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
187104	4.74	0.26	0.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
187105	4.86	0.26	0.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

5 rows × 56 columns

The data go back a long way! Let's focus on a more modern sample to try to make this more relevant.

```
predictors = predictors.loc[195001:]
predictors.head()
```

	price	d12	e12	ret	retx	AAA	BAA	lty	ltr	corpr	...	ygap	rdsp	rsvix	gpce	gip	tchi	hou
yyyymm																		
195001	17.05	1.15	2.33667	0.019703	0.018245	0.0257	0.0324	0.0215	-0.0061	0.0037	...	NaN	0.046476	NaN	NaN	NaN	NaN	N
195002	17.22	1.16	2.35333	0.019603	0.009975	0.0258	0.0324	0.0214	0.0021	0.0007	...	NaN	0.035501	NaN	NaN	NaN	NaN	N
195003	17.29	1.17	2.37000	0.008185	0.003542	0.0258	0.0324	0.0215	0.0008	0.0022	...	NaN	0.033742	NaN	NaN	NaN	NaN	N
195004	18.07	1.18	2.42667	0.045887	0.044493	0.0260	0.0323	0.0214	0.0030	-0.0008	...	NaN	0.050808	NaN	NaN	NaN	NaN	N
195005	18.78	1.19	2.48333	0.046902	0.037590	0.0261	0.0325	0.0213	0.0033	-0.0008	...	NaN	0.029955	NaN	NaN	NaN	NaN	N

5 rows × 56 columns

The predictors dataset represents my entire sample. If I am running a backtest, I want to pretend that today is sometime during that sample and perform the data analysis that would have been feasible at that time.

Before we get into a full analysis, let's look at an example. Suppose it is the very end of 2012. I will create a data frame pred2012 that includes this data. NOte that this includes everything from 1950 up until 2012. We could also use a rolling window -- say, the last 10 years. In that case we would select the data from [200212:201212].

```
pred2012 = predictors.loc[:201212]
```

Next, I will add a few variables that might be interesting. One is the ratio of earnings to prices. The other is the 12-month MA of inflation:

```
pred2012['ep']     = pred2012['e12']/pred2012['price']
pred2012['infl12'] = pred2012['infl'].rolling(12).mean()
```

```
<ipython-input-7-614a8ee66560>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  pred2012['ep']     = pred2012['e12']/pred2012['price']
<ipython-input-7-614a8ee66560>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  pred2012['infl12'] = pred2012['infl'].rolling(12).mean()
```

Now I will just keep the data that I care about, which includes these columns, a few other predictors, and the market return:

```
pred2012 = pred2012[['ep','infl12','lty','ntis','ret']]
```

Predictive regressions are of the form

$$R_{t+1} = \alpha + \beta X_t + \epsilon_{t+1}$$

It is important that the dependent variable is later than the independent variable. We are trying to predict returns with X, so we need to be able to see X before the investment is made.

I can either *lag* the X variables or *lead* the returns. Either way is fine. I usually do the former, but this time I'll do the latter, which is accomplished with shift(-1):

```
pred2012['ret']=predictors['ret'].shift(-1)
```

Since shift(-1) will create some missing values, I will use dropna:

```
pred2012.dropna(inplace=True)
```

Let's run a regression. I'll use statsmodels.formula.api, which allows you to specify your regression equation. Also, smf.ols automatically adds a constant to the regression, which is usually convenient.

In my regression, I will regress future returns on the current E/P ratio:

```
results = smf.ols('ret ~ ep + infl12', data=pred2012).fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    ret   R-squared:                       0.029
Model:                            OLS   Adj. R-squared:                  0.027
Method:                 Least Squares   F-statistic:                     11.17
Date:                Tue, 29 Oct 2024   Prob (F-statistic):           1.66e-05
Time:                        23:39:23   Log-Likelihood:                 1312.8
No. Observations:                 745   AIC:                            -2620.
Df Residuals:                     742   BIC:                            -2606.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.0025      0.004     -0.605      0.545      -0.011       0.006
ep             0.3453      0.076      4.517      0.000       0.195       0.495
infl12        -3.7207      0.909     -4.093      0.000      -5.505      -1.936
==============================================================================
Omnibus:                       36.967   Durbin-Watson:                   1.916
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               75.296
Skew:                          -0.305   Prob(JB):                     4.46e-17
Kurtosis:                       4.433   Cond. No.                         598.
==============================================================================

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

The t-stats indicate strong statistical significance, but the R-square is not super high. The predictability here is not huge.

Maybe we will find more predictability using the other two predictors, lty (long-term bond yield) and ntis (net share issuance).

```
results = smf.ols('ret ~ ep + infl12 + lty + ntis', data=pred2012).fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    ret   R-squared:                       0.032
Model:                            OLS   Adj. R-squared:                  0.026
Method:                 Least Squares   F-statistic:                     6.018
Date:                Tue, 29 Oct 2024   Prob (F-statistic):           9.08e-05
Time:                        23:39:29   Log-Likelihood:                 1313.7
No. Observations:                 745   AIC:                            -2617.
Df Residuals:                     740   BIC:                            -2594.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.0049      0.005     -0.922      0.357      -0.015       0.005
ep             0.3516      0.077      4.573      0.000       0.201       0.503
infl12        -4.2485      1.041     -4.082      0.000      -6.292      -2.205
lty            0.0701      0.072      0.974      0.330      -0.071       0.211
ntis          -0.0581      0.085     -0.680      0.496      -0.226       0.109
==============================================================================
Omnibus:                       39.389   Durbin-Watson:                   1.919
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               81.257
Skew:                          -0.323   Prob(JB):                     2.27e-18
Kurtosis:                       4.483   Cond. No.                         686.
==============================================================================

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

The same OLS regression can also be run using sm.OLS. (The capitalization in "sm.OLS" is required.) This time, we give the function our dependent variable and our independent variables as separate arguments. In addition, the function does not automatically assume that we want to include a constant. Instead, we have to use the sm.add_constant function to add a constant to our predictors.

```
Y = pred2012['ret']
X = pred2012[['ep','infl12','lty','ntis']]
X.head()
```

yyyymm	ep	infl12	lty	ntis
195012	0.139147	0.004825	0.0224	0.031358
195101	0.130964	0.006512	0.0221	0.030045
195102	0.129969	0.007496	0.0228	0.031120
195103	0.132243	0.007465	0.0241	0.032687
195104	0.124535	0.007465	0.0248	0.032092

```
sm.add_constant(X).head()
```

yyyymm	const	ep	infl12	lty	ntis
195012	1.0	0.139147	0.004825	0.0224	0.031358
195101	1.0	0.130964	0.006512	0.0221	0.030045
195102	1.0	0.129969	0.007496	0.0228	0.031120
195103	1.0	0.132243	0.007465	0.0241	0.032687
195104	1.0	0.124535	0.007465	0.0248	0.032092

We can run the regression using sm.OLS as follows:

```
results2 = sm.OLS(Y, sm.add_constant(X)).fit()
print(results2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    ret   R-squared:                       0.032
Model:                            OLS   Adj. R-squared:                  0.026
Method:                 Least Squares   F-statistic:                     6.018
Date:                Tue, 29 Oct 2024   Prob (F-statistic):           9.08e-05
Time:                        23:39:44   Log-Likelihood:                 1313.7
No. Observations:                 745   AIC:                            -2617.
Df Residuals:                     740   BIC:                            -2594.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0049      0.005     -0.922      0.357      -0.015       0.005
ep             0.3516      0.077      4.573      0.000       0.201       0.503
infl12        -4.2485      1.041     -4.082      0.000      -6.292      -2.205
lty            0.0701      0.072      0.974      0.330      -0.071       0.211
ntis          -0.0581      0.085     -0.680      0.496      -0.226       0.109
==============================================================================
Omnibus:                       39.389   Durbin-Watson:                   1.919
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               81.257
Skew:                          -0.323   Prob(JB):                     2.27e-18
Kurtosis:                       4.483   Cond. No.                         686.
==============================================================================

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

The results turn out to be the same, of course.

Let's now consider briefly how you would run a regression on a subset of the variables in the dataframe, for example those that have a higher correlation with the dependent variable. This is in line with what Hull and Qiao recommend.

We can compute the correlation between Y and each column of X with

```
rho=X.corrwith(Y)
print(rho)
```

```
ep        0.085510
infl12   -0.050415
lty       0.001298
ntis     -0.016634
dtype: float64
```

Note that rho is a series:

```
type(rho)
```

```
pandas.core.series.Series
```

We can therefore use the loc property to subset the rows of rho, for example to find rows that have a high enough absolute correlation. (I'm using .05 here instead of .1 since there aren't any correlations above .1 in absolute value.)

```
rhokeep = rho.loc[np.abs(rho)>.05]
print(rhokeep)
```

```
ep        0.085510
infl12   -0.050415
dtype: float64
```

If we want to know what the indexes are of the high correlation variables, we can see them by typing

```
rhokeep.index
```

```
Index(['ep', 'infl12'], dtype='object')
```

It looks like we might want to drop lty and ntis. We can pull out the desired columns out of the X dataframe using the standard method:

```
Xsub=X[rhokeep.index]
Xsub.head()
```

	ep	infl12
yyyymm		
195012	0.139147	0.004825
195101	0.130964	0.006512
195102	0.129969	0.007496
195103	0.132243	0.007465
195104	0.124535	0.007465

With the subset of predictors that we want to keep, we can come up with a slightly more parsimonious regression.

The nice thing about sm.OLS, as opposed to smf.ols, is that we can just give it the entire Xsub dataframe without having to write out the model describing which independent variables we want it to include:

```
results4 = sm.OLS(Y, sm.add_constant(Xsub)).fit()
print(results4.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    ret   R-squared:                       0.029
Model:                            OLS   Adj. R-squared:                  0.027
Method:                 Least Squares   F-statistic:                     11.17
Date:                Tue, 29 Oct 2024   Prob (F-statistic):           1.66e-05
Time:                        23:40:53   Log-Likelihood:                 1312.8
No. Observations:                 745   AIC:                            -2620.
Df Residuals:                     742   BIC:                            -2606.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0025      0.004     -0.605      0.545      -0.011       0.006
ep             0.3453      0.076      4.517      0.000       0.195       0.495
infl12        -3.7207      0.909     -4.093      0.000      -5.505      -1.936
==============================================================================
Omnibus:                       36.967   Durbin-Watson:                   1.916
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               75.296
Skew:                          -0.305   Prob(JB):                     4.46e-17
Kurtosis:                       4.433   Cond. No.                         598.
==============================================================================

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

Now let's Look at the larger set of predictors. We won't do anything too fancy but will consider a "kitchen sink" approach.

I'll pull the data through 2014 and drop the fields where there are no monthly values.

```
pred2014 = predictors.loc[:201214]
pred2014.drop(['gpce','gip','house','eqis','cay','i/k','pce','govik','skew','crdstd','accrul','cfacc'],axis=1,inplace=True)
pred2014['infl12'] = pred2014['infl'].rolling(12).mean()
pred2014['ret']=predictors['ret'].shift(-1)
pred2014.dropna(inplace=True)
print(pred2014.columns)
```

```
Index(['price', 'd12', 'e12', 'ret', 'retx', 'AAA', 'BAA', 'lty', 'ltr',
       'corpr', 'tbl', 'Rfree', 'd/p', 'd/y', 'e/p', 'd/e', 'b/m', 'tms',
       'dfy', 'dfr', 'infl', 'ntis', 'svar', 'csp', 'vp', 'impvar', 'vrp',
       'lzrt', 'ogap', 'wtexas', 'sntm', 'ndrbl', 'skvw', 'tail', 'fbm',
       'dtoy', 'dtoat', 'ygap', 'rdsp', 'rsvix', 'tchi', 'avgcor', 'shtint',
       'disag', 'infl12'],
      dtype='object')
<ipython-input-72-92fe26c2d59b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  pred2014.drop(['gpce','gip','house','eqis','cay','i/k','pce','govik','skew','crdstd','accrul','cfacc'],axis=1,inplace=True
<ipython-input-72-92fe26c2d59b>:3: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  pred2014['infl12'] = pred2014['infl'].rolling(12).mean()
<ipython-input-72-92fe26c2d59b>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  pred2014['ret']=predictors['ret'].shift(-1)
<ipython-input-72-92fe26c2d59b>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  pred2014.dropna(inplace=True)
```

```
Y=pred2014['ret']
X=pred2014.drop(['retx','ret','price'],axis=1)
```

```
Y.head()
```

```
yyyymm
199601    0.010088
199602    0.009585
199603    0.015133
199604    0.025257
199605    0.004128
Name: ret, dtype: float64
```

This gives us a pretty good r-squared. But we have some problems still. First, this is not a true walk-forward analysis. This is using data all the way through 2014 to estimate the coefficients, and the r-squared applies those coefficients to predictions made at every month along the way. But we don't know the information used to estimate the coefficients yet. A true walk-forward analysis will reestimate the coefficients every month along the way.

```
results5 = sm.OLS(Y, sm.add_constant(X)).fit()
print(results5.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    ret   R-squared:                       0.771
Model:                            OLS   Adj. R-squared:                  0.569
Method:                 Least Squares   F-statistic:                     3.805
Date:                Wed, 30 Oct 2024   Prob (F-statistic):           1.43e-05
Time:                        00:08:41   Log-Likelihood:                 192.87
No. Observations:                  84   AIC:                            -305.7
Df Residuals:                      44   BIC:                            -208.5
Df Model:                          39
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -1.1658      3.516     -0.332      0.742      -8.252       5.920
d12            0.1126      0.054      2.077      0.044       0.003       0.222
e12           -0.0221      0.012     -1.925      0.061      -0.045       0.001
AAA           -3.6836      3.697     -0.996      0.324     -11.134       3.766
BAA            4.6432      3.302      1.406      0.167      -2.012      11.299
lty           -2.9300      3.381     -0.867      0.391      -9.744       3.884
ltr           -0.1457      0.203     -0.718      0.476      -0.555       0.263
corpr          0.1774      0.368      0.482      0.632      -0.565       0.920
tbl           -3.6251      2.801     -1.294      0.202      -9.270       2.020
Rfree         -6.7424     18.897     -0.357      0.723     -44.826      31.341
d/p           -8.8914     41.725     -0.213      0.832     -92.983      75.200
d/y           13.0454     11.088      1.177      0.246      -9.300      35.391
e/p           21.3598     17.831      1.198      0.237     -14.577      57.297
d/e           -1.4401      0.996     -1.446      0.155      -3.447       0.566
b/m           -0.3362      0.321     -1.048      0.300      -0.982       0.310
tms            0.6951      1.657      0.419      0.677      -2.645       4.035
dfy            8.3267      4.764      1.748      0.087      -1.275      17.929
dfr            0.3231      0.463      0.697      0.489      -0.611       1.257
infl           1.2797      2.551      0.502      0.618      -3.861       6.420
ntis          -2.4468      1.619     -1.511      0.138      -5.710       0.816
svar          -9.8715      6.929     -1.425      0.161     -23.836       4.093
csp           56.9846     12.208      4.668      0.000      32.382      81.587
vp            -0.0017      0.001     -1.189      0.241      -0.004       0.001
impvar         4.1172      4.450      0.925      0.360      -4.851      13.086
vrp           -0.0010      0.001     -0.961      0.342      -0.003       0.001
lzrt           0.0651      0.105      0.623      0.537      -0.146       0.276
ogap           1.7825      1.105      1.614      0.114      -0.444       4.009
```

```
wtexas      -0.0048      0.054     -0.088      0.930      -0.114       0.104
sntm         0.0232      0.056      0.418      0.678      -0.089       0.135
ndrbl       -0.3293      0.225     -1.460      0.151      -0.784       0.125
skvw        -0.1136      0.160     -0.712      0.480      -0.435       0.208
tail         0.1452      0.510      0.285      0.777      -0.882       1.173
fbm          0.0515      0.129      0.399      0.692      -0.208       0.311
dtoy        -2.4401      0.925     -2.639      0.011      -4.303      -0.577
dtoat        1.7198      1.000      1.720      0.092      -0.296       3.735
ygap        -0.3177      0.868     -0.366      0.716      -2.068       1.433
rdsp         0.0744      0.441      0.169      0.867      -0.814       0.963
rsvix        1.3226      1.328      0.996      0.325      -1.354       3.999
tchi        -0.0062      0.011     -0.547      0.587      -0.029       0.017
avgcor      -0.0923      0.126     -0.735      0.466      -0.345       0.161
shtint       0.0375      0.057      0.664      0.510      -0.076       0.151
disag       -0.0330      0.034     -0.983      0.331      -0.101       0.035
infl12      -7.3446     28.083     -0.262      0.795     -63.943      49.253
==============================================================================
```

Also note that there is strong multicollinarity problems -- we have a bunch of redundant regressors in here. That's shown through the correlation of the x-variables with one another.

First let's thin down the list of regressors a little bit more

```
rho=X.corrwith(Y)
print(rho)
```

```
d12      -0.107335
e12       0.062273
AAA       0.023921
BAA      -0.081108
lty       0.134196
ltr      -0.067603
corpr    -0.114048
tbl       0.158413
Rfree     0.144447
d/p       0.218231
d/y       0.205059
e/p       0.283114
d/e      -0.143479
b/m       0.040495
tms      -0.121085
dfy      -0.183962
dfr      -0.047609
infl     -0.041661
ntis      0.151262
svar      0.074783
csp       0.292732
vp        0.218441
impvar    0.326652
vrp       0.230221
lzrt      0.064488
ogap     -0.090766
wtexas   -0.137603
sntm      0.305761
ndrbl     0.047307
skvw     -0.213490
tail      0.044648
fbm       0.070604
dtoy     -0.074653
dtoat     0.020276
ygap      0.289543
rdsp     -0.040399
rsvix     0.198180
tchi      0.205758
avgcor    0.188565
shtint    0.017044
disag    -0.156096
infl12   -0.070773
dtype: float64
```

```
rhokeep = rho.loc[np.abs(rho)>.15]
Xsub=X[rhokeep.index]
Xsub.head()
```

	tbl	d/p	d/y	e/p	dfy	ntis	csp	vp	impvar	vrp	sntm	skvw	ygap	rsvi
yyyymm														
199601	0.0500	0.021844	0.022557	0.053436	0.0066	0.016128	-0.000053	3.609275	0.005112	6.2807	0.526699	0.045017	-2.984223	0.01336
199602	0.0483	0.021858	0.022010	0.053110	0.0064	0.016801	-0.000228	8.057655	0.007465	15.4671	0.428522	0.036731	-2.991862	0.02652
199603	0.0496	0.021850	0.022023	0.052734	0.0068	0.016892	0.000031	16.592991	0.005754	18.9536	0.462212	-0.016471	-3.003302	0.02985
199604	0.0495	0.021646	0.021937	0.052479	0.0069	0.021122	0.000068	10.918662	0.007574	14.1619	0.312246	0.032744	-3.010416	0.02170
199605	0.0502	0.021247	0.021732	0.051740	0.0068	0.026419	-0.000255	9.828647	0.006154	14.2560	0.336702	0.040774	-3.026758	0.02277

```
results6 = sm.OLS(Y, sm.add_constant(Xsub)).fit()
print(results6.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    ret   R-squared:                       0.422
Model:                            OLS   Adj. R-squared:                  0.274
Method:                 Least Squares   F-statistic:                     2.840
Date:                Wed, 30 Oct 2024   Prob (F-statistic):            0.00126
Time:                        00:26:10   Log-Likelihood:                 153.97
No. Observations:                  84   AIC:                            -271.9
Df Residuals:                      66   BIC:                            -228.2
Df Model:                          17
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.9465      1.288      0.735      0.465      -1.625       3.518
tbl           -2.4692      2.007     -1.230      0.223      -6.477       1.538
d/p           -4.8623     12.920     -0.376      0.708     -30.657      20.933
d/y           -5.5342      9.120     -0.607      0.546     -23.742      12.674
e/p           -0.2822      9.155     -0.031      0.976     -18.561      17.997
dfy           10.6841      5.340      2.001      0.050       0.023      21.346
ntis          -0.0433      1.079     -0.040      0.968      -2.197       2.111
csp           49.6890     11.196      4.438      0.000      27.336      72.042
vp             0.0009      0.001      0.675      0.502      -0.002       0.003
impvar         6.1648      4.424      1.393      0.168      -2.669      14.998
vrp        -1.779e-05      0.001     -0.026      0.979      -0.001       0.001
sntm           0.0350      0.041      0.861      0.392      -0.046       0.116
skvw          -0.2356      0.176     -1.339      0.185      -0.587       0.116
ygap           0.1946      0.289      0.672      0.504      -0.383       0.772
rsvix         -0.4904      0.666     -0.737      0.464      -1.819       0.839
tchi          -0.0087      0.010     -0.897      0.373      -0.028       0.011
avgcor         0.0511      0.103      0.495      0.622      -0.155       0.257
disag         -0.0251      0.026     -0.985      0.328      -0.076       0.026
==============================================================================
Omnibus:                        1.554   Durbin-Watson:                   1.893
Prob(Omnibus):                  0.460   Jarque-Bera (JB):                0.991
Skew:                          -0.226   Prob(JB):                        0.609
Kurtosis:                       3.282   Cond. No.                     1.34e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.34e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
Xsub.corr()
```

	tbl	d/p	d/y	e/p	dfy	ntis	csp	vp	impvar	vrp	sntm	skvw	yga
tbl	1.000000	-0.080460	-0.038278	0.590961	-0.843705	-0.161503	0.755480	-0.241114	0.296421	-0.068618	0.331612	0.097674	0.64143
d/p	-0.080460	1.000000	0.972416	0.733903	-0.102859	0.518388	-0.019255	-0.084143	-0.163573	0.004912	0.590043	-0.173147	0.67111
d/y	-0.038278	0.972416	1.000000	0.739394	-0.145176	0.502968	0.046338	-0.136974	-0.216103	-0.070829	0.630698	-0.081861	0.67719
e/p	0.590961	0.733903	0.739394	1.000000	-0.626858	0.230568	0.461814	-0.240550	0.038001	-0.078795	0.641248	-0.067955	0.98924
dfy	-0.843705	-0.102859	-0.145176	-0.626858	1.000000	0.069354	-0.697828	0.114044	-0.257314	-0.106860	-0.544701	-0.024202	-0.66319
ntis	-0.161503	0.518388	0.502968	0.230568	0.069354	1.000000	-0.058960	0.110921	0.103933	0.149049	0.580857	-0.037260	0.20353
csp	0.755480	-0.019255	0.046338	0.461814	-0.697828	-0.058960	1.000000	-0.397064	0.033457	-0.199712	0.288028	0.239324	0.45239
vp	-0.241114	-0.084143	-0.136974	-0.240550	0.114044	0.110921	-0.397064	1.000000	0.711592	0.825376	0.054688	-0.514328	-0.18629
impvar	0.296421	-0.163573	-0.216103	0.038001	-0.257314	0.103933	0.033457	0.711592	1.000000	0.622557	0.201547	-0.381874	0.11844
vrp	-0.068618	0.004912	-0.070829	-0.078795	-0.106860	0.149049	-0.199712	0.825376	0.622557	1.000000	0.197748	-0.609329	-0.05010
sntm	0.331612	0.590043	0.630698	0.641248	-0.544701	0.580857	0.288028	0.054688	0.201547	0.197748	1.000000	-0.028057	0.64880
skvw	0.097674	-0.173147	-0.081861	-0.067955	-0.024202	-0.037260	0.239324	-0.514328	-0.381874	-0.609329	-0.028057	1.000000	-0.06372
ygap	0.641430	0.671119	0.677199	0.989249	-0.663197	0.203536	0.452391	-0.186297	0.118444	-0.050100	0.648800	-0.063729	1.00000
rsvix	-0.279658	-0.091430	-0.195913	-0.252469	0.218963	0.109682	-0.459774	0.892326	0.724563	0.697528	-0.053685	-0.544357	-0.19247
tchi	0.615944	0.270018	0.340885	0.557505	-0.660371	0.401921	0.631220	-0.248314	0.098662	0.024245	0.768039	0.088223	0.55391
avgcor	-0.534876	0.306679	0.292811	-0.100326	0.329597	0.372811	-0.505751	0.532261	0.217109	0.330822	0.267868	-0.246655	-0.08388
disag	0.116341	-0.830206	-0.830116	-0.553597	0.168047	-0.413365	0.086957	0.071716	0.196712	-0.084189	-0.684709	0.124599	-0.50170