Lecture: Linear Regression in Python

March 9, 2020

Contents

In addition to what's in Anaconda, this lecture will need the following libraries:

In [1]:
```
!pip install linearmodels
```

```
Requirement already satisfied: linearmodels in /Users/wasin_siwasarit/anaconda3/
lib/python3.7/site-packages (4.17)
Requirement already satisfied: numpy>=1.15 in /Users/wasin_siwasarit/anaconda3/l
ib/python3.7/site-packages (from linearmodels) (1.15.4)
Requirement already satisfied: scipy>=1 in /Users/wasin_siwasarit/anaconda3/lib/
python3.7/site-packages (from linearmodels) (1.1.0)
Requirement already satisfied: patsy in /Users/wasin_siwasarit/anaconda3/lib/pyt
hon3.7/site-packages (from linearmodels) (0.5.1)
Requirement already satisfied: statsmodels>=0.9 in /Users/wasin_siwasarit/anacon
da3/lib/python3.7/site-packages (from linearmodels) (0.9.0)
Requirement already satisfied: mypy-extensions>=0.4 in /Users/wasin_siwasarit/an
aconda3/lib/python3.7/site-packages (from linearmodels) (0.4.3)
Requirement already satisfied: property-cached>=1.6.3 in /Users/wasin_siwasarit/
anaconda3/lib/python3.7/site-packages (from linearmodels) (1.6.4)
Requirement already satisfied: pandas>=0.23 in /Users/wasin_siwasarit/anaconda3/
lib/python3.7/site-packages (from linearmodels) (0.25.3)
Requirement already satisfied: Cython>=0.29.14 in /Users/wasin_siwasarit/anacond
a3/lib/python3.7/site-packages (from linearmodels) (0.29.15)
Requirement already satisfied: six in /Users/wasin_siwasarit/anaconda3/lib/pytho
n3.7/site-packages (from patsy->linearmodels) (1.12.0)
Requirement already satisfied: python-dateutil>=2.6.1 in /Users/wasin_siwasarit/
anaconda3/lib/python3.7/site-packages (from pandas>=0.23->linearmodels) (2.7.5)
Requirement already satisfied: pytz>=2017.2 in /Users/wasin_siwasarit/anaconda3/
lib/python3.7/site-packages (from pandas>=0.23->linearmodels) (2018.7)
```

In [ ]:
```
!pip install --upgrade pip
```

Overview

Linear regression is a standard tool for analyzing the relationship between two or more variables.

In this lecture, we'll use the Python package `statsmodels` to estimate, interpret, and visualize linear regression models.

Along the way, we'll discuss a variety of topics, including

- simple and multivariate linear regression
- visualization
- endogeneity and omitted variable bias
- two-stage least squares

As an example, we will replicate results from Acemoglu, Johnson and Robinson's seminal paper [AJR01] (https://python.quantecon.org/zreferences.html#acemoglu2001).

- You can download a copy here (https://economics.mit.edu/files/4123).

In the paper, the authors emphasize the importance of institutions in economic development.

The main contribution is the use of settler mortality rates as a source of *exogenous* variation in institutional differences.

Such variation is needed to determine whether it is institutions that give rise to greater economic growth, rather than the other way around.

Let's start with some imports:

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import statsmodels.api as sm
from statsmodels.iolib.summary2 import summary_col
#from linearmodels.iv import IV2SLS
```

```
/Users/wasin_siwasarit/anaconda3/lib/python3.7/site-packages/statsmodels/compat/
pandas.py:49: FutureWarning: The Panel class is removed from pandas. Accessing i
t from the top-level namespace will also be removed in the next version
  data_klasses = (pandas.Series, pandas.DataFrame, pandas.Panel)
```

Prerequisites

This lecture assumes you are familiar with basic econometrics.

For an introductory text covering these topics, see, for example, [Woo15] (https://python.quantecon.org/zreferences.html#wooldridge2015).

Simple Linear Regression

[AJR01] (https://python.quantecon.org/zreferences.html#acemoglu2001) wish to determine whether or not differences in institutions can help to explain observed economic outcomes.

How do we measure *institutional differences* and *economic outcomes*?

In this paper,

- economic outcomes are proxied by log GDP per capita in 1995, adjusted for exchange rates.
- institutional differences are proxied by an index of protection against expropriation on average over 1985-95, constructed by the Political Risk Services Group (https://www.prsgroup.com/).

These variables and other data used in the paper are available for download on Daron Acemoglu's webpage (https://economics.mit.edu/faculty/acemoglu/data/ajr2001).

We will use pandas' `.read_stata()` function to read in data contained in the `.dta` files to dataframes

In [2]:
```
df1 = pd.read_stata('https://github.com/QuantEcon/lecture-source-py/blob/master/s
df1.head()
x = df1[['logpgp95','avexpr']]
x
```

Out[2]:

	logpgp95	avexpr
0	NaN	NaN
1	7.770645	5.363636
2	9.804219	7.181818
3	9.133459	6.386364
4	7.682482	NaN
...	...	...
158	NaN	6.318182
159	8.885994	6.863636
160	6.866933	3.500000
161	6.813445	6.636364
162	7.696213	6.000000

163 rows × 2 columns

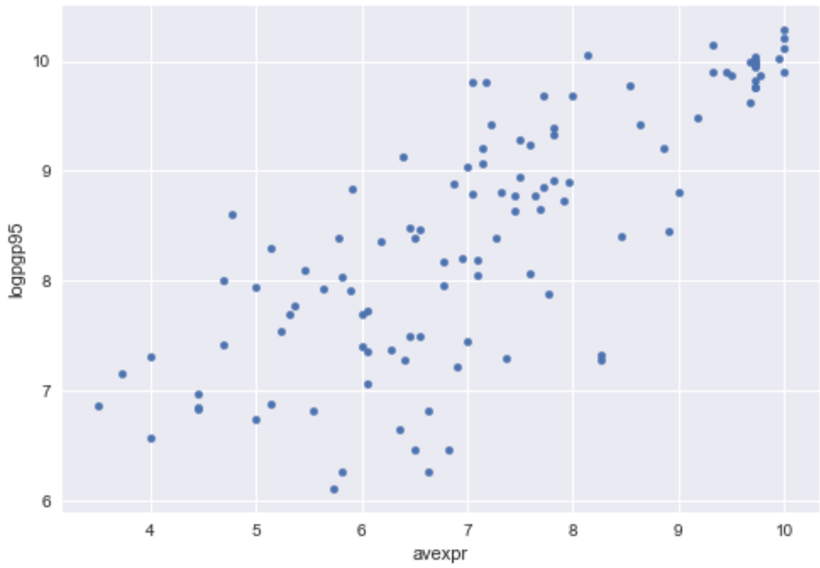
In [3]:
```
df2 = pd.read_stata('maketable1.dta')
df2.head()
```

Out[3]:

	shortnam	euro1900	excolony	avexpr	logpgp95	cons1	cons90	democ00a	cons00a	extmort4	log
0	AFG	0.000000	1.0	NaN	NaN	1.0	2.0	1.0	1.0	93.699997	4.54
1	AGO	8.000000	1.0	5.363636	7.770645	3.0	3.0	0.0	1.0	280.000000	5.63
2	ARE	0.000000	1.0	7.181818	9.804219	NaN	NaN	NaN	NaN	NaN	
3	ARG	60.000004	1.0	6.386364	9.133459	1.0	6.0	3.0	3.0	68.900002	4.23
4	ARM	0.000000	0.0	NaN	7.682482	NaN	NaN	NaN	NaN	NaN	

Let's use a scatterplot to see whether any obvious relationship exists between GDP per capita and the protection against expropriation index

In [5]:
```
plt.style.use('seaborn')

df1.plot(x='avexpr', y='logpgp95', kind='scatter')
plt.show()
```



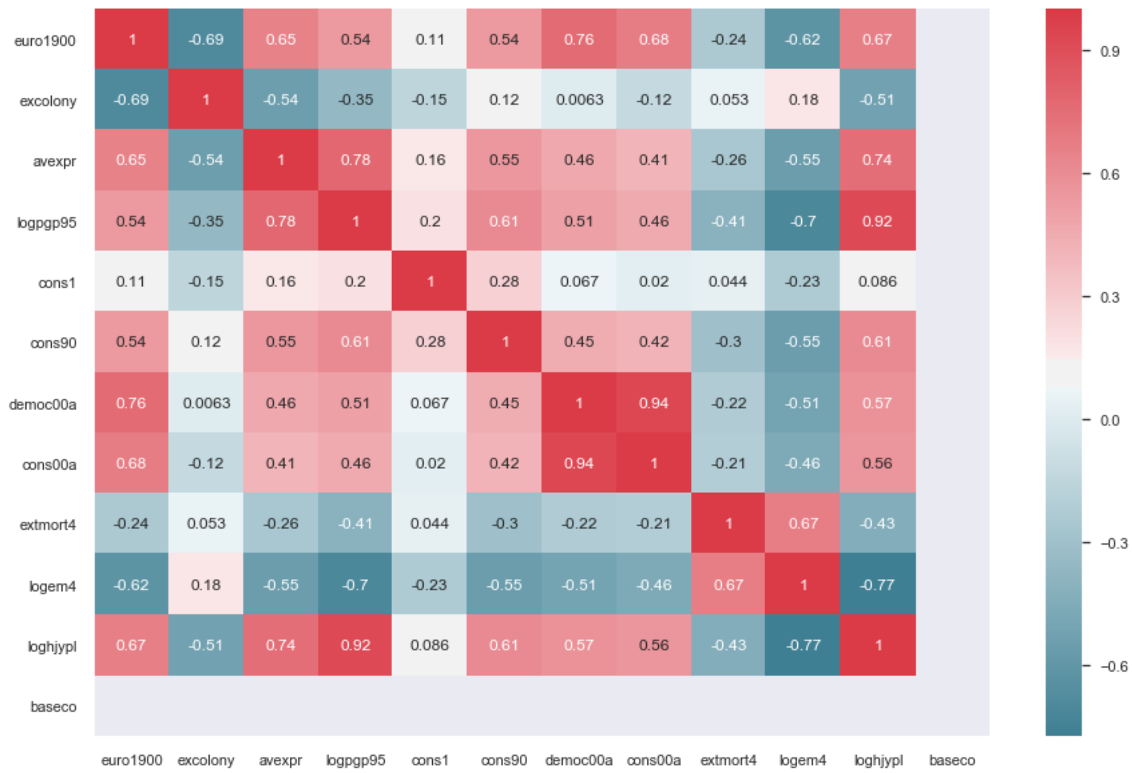
In [6]:
```
correlation_data = df1.corr()
correlation_data
```

Out[6]:

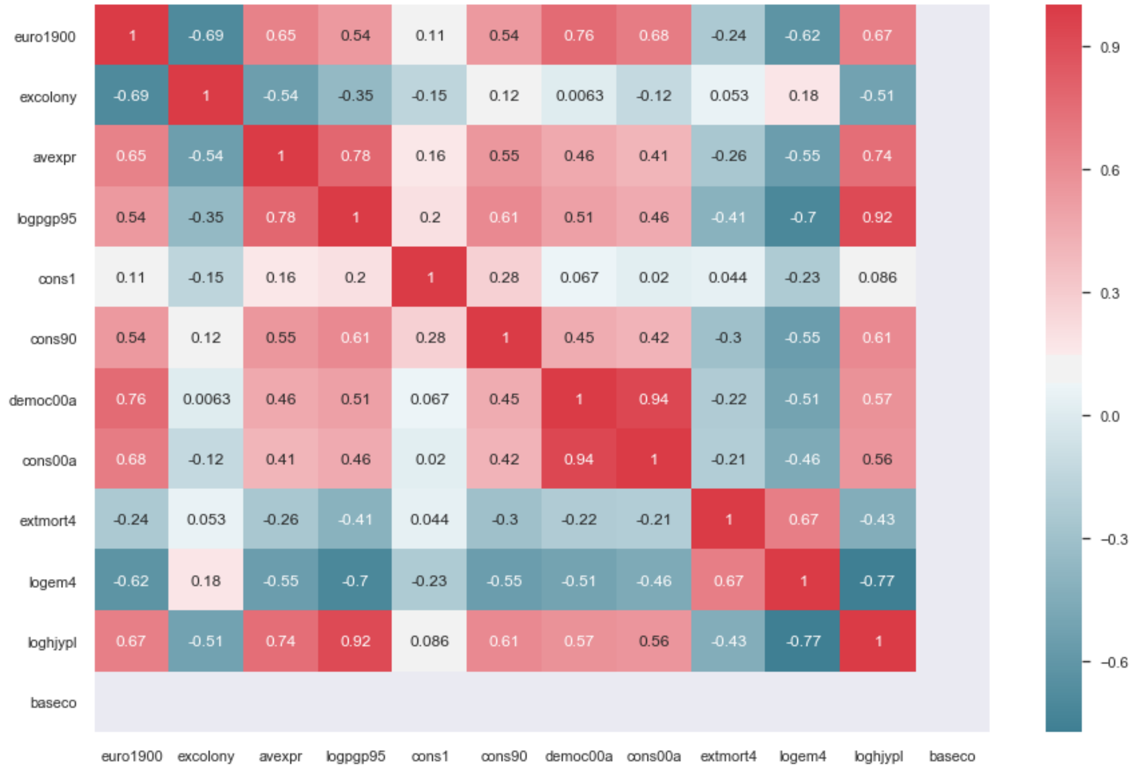
	euro1900	excolony	avexpr	logpgp95	cons1	cons90	democ00a	cons00a	extmort4
euro1900	1.000000	-0.689273	0.651259	0.543301	0.105453	0.539373	0.760907	0.681130	-0.240488
excolony	-0.689273	1.000000	-0.544850	-0.351238	-0.152230	0.121484	0.006289	-0.124190	0.052674
avexpr	0.651259	-0.544850	1.000000	0.781871	0.164761	0.550720	0.464887	0.405621	-0.258020
logpgp95	0.543301	-0.351238	0.781871	1.000000	0.202709	0.614515	0.507646	0.455448	-0.406272
cons1	0.105453	-0.152230	0.164761	0.202709	1.000000	0.279989	0.067338	0.020381	0.043582
cons90	0.539373	0.121484	0.550720	0.614515	0.279989	1.000000	0.448136	0.418775	-0.302129
democ00a	0.760907	0.006289	0.464887	0.507646	0.067338	0.448136	1.000000	0.936181	-0.217778
cons00a	0.681130	-0.124190	0.405621	0.455448	0.020381	0.418775	0.936181	1.000000	-0.208006
extmort4	-0.240488	0.052674	-0.258020	-0.406272	0.043582	-0.302129	-0.217778	-0.208006	1.000000
logem4	-0.619421	0.175785	-0.551757	-0.704763	-0.230706	-0.550856	-0.514465	-0.461551	0.674897
loghjypl	0.667556	-0.514670	0.744358	0.921999	0.086061	0.613374	0.571446	0.558489	-0.433068
baseco	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [6]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
_,ax=plt.subplots(figsize=(15,10))
colormap=sns.diverging_palette(220,10,as_cmap=True)
sns.heatmap(df1.corr(),annot=True,cmap=colormap)
```

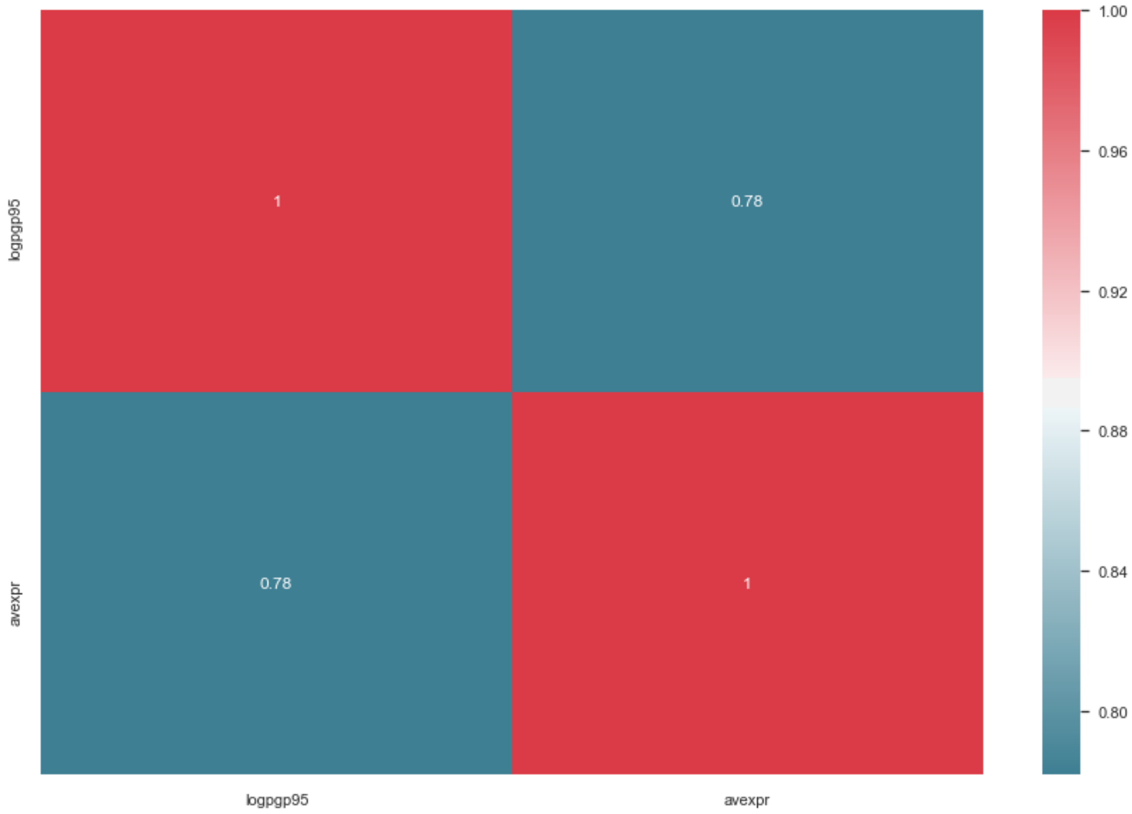
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1c21bd8cc0>



In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
def correlation_heatmap(wasin):
    _,ax=plt.subplots(figsize=(15,10))
    colormap=sns.diverging_palette(220,10,as_cmap=True)
    sns.heatmap(wasin.corr(),annot=True,cmap=colormap)

correlation_heatmap(df1)
plt.savefig("heat_map.png")
```



In [19]:
```
correlation_heatmap(x)
```



The plot shows a fairly strong positive relationship between protection against expropriation and log GDP per capita.

Specifically, if higher protection against expropriation is a measure of institutional quality, then better institutions appear to be positively correlated with better economic outcomes (higher GDP per capita).

Given the plot, choosing a linear model to describe this relationship seems like a reasonable assumption.

We can write our model as

$$logpgp95_i = \beta_0 + \beta_1 avexpr_i + u_i$$

where:

- β_0 is the intercept of the linear trend line on the y-axis
- β_1 is the slope of the linear trend line, representing the *marginal effect* of protection against risk on log GDP per capita
- u_i is a random error term (deviations of observations from the linear trend due to factors not included in the model)

Visually, this linear model involves choosing a straight line that best fits the data, as in the following plot (Figure 2 in [AJR01] (https://python.quantecon.org/zreferences.html#acemoglu2001))

In [9]:
```
df1_subset = df1.dropna(subset=['logpgp95', 'avexpr'])
df1_subset
df1_subset = df1_subset[df1_subset['baseco'] == 1]
df1_subset
```

Out[9]:

	shortnam	euro1900	excolony	avexpr	logpgp95	cons1	cons90	democ00a	cons00a	extmort4
1	AGO	8.000000	1.0	5.363636	7.770645	3.0	3.0	0.0	1.0	280.000000
3	ARG	60.000004	1.0	6.386364	9.133459	1.0	6.0	3.0	3.0	68.900002
5	AUS	98.000000	1.0	9.318182	9.897972	7.0	7.0	10.0	7.0	8.550000
11	BFA	0.000000	1.0	4.454545	6.845880	3.0	1.0	0.0	1.0	280.000000
12	BGD	0.000000	1.0	5.136364	6.877296	7.0	2.0	0.0	1.0	71.410004
...	...	...	...	...	...	...	...	...	...	...
153	USA	87.500000	1.0	10.000000	10.215740	7.0	7.0	10.0	7.0	15.000000
155	VEN	20.000000	1.0	7.136364	9.071078	1.0	3.0	1.0	3.0	78.099998
156	VNM	0.000000	1.0	6.409091	7.279319	1.0	3.0	0.0	1.0	140.000000
159	ZAF	22.000000	1.0	6.863636	8.885994	3.0	7.0	3.0	3.0	15.500000
160	ZAR	8.000000	1.0	3.500000	6.866933	1.0	1.0	0.0	1.0	240.000000

64 rows × 13 columns

In [22]:
```python
# Dropping NA's is required to use numpy's polyfit
df2_subset = df1.dropna(subset=['logpgp95', 'loghjypl'])
df2_subset = df2_subset[df2_subset['baseco'] == 1]
labels=df2_subset['shortnam']
X=df2_subset['loghjypl']
y=df2_subset['logpgp95']
# Replace markers with country labels
fig, ax = plt.subplots()
ax.scatter(X, y, marker='')

for i, label in enumerate(labels):
    ax.annotate(label, (X.iloc[i], y.iloc[i]))

# Fit a linear trend line

ax.plot(np.unique(X),
         np.poly1d(np.polyfit(X, y, 1))(np.unique(X)),
         color='black')

ax.set_xlim([-5,2.5])
ax.set_ylim([6,12])
ax.set_xlabel('Average Expropriation Risk 1985-95')
ax.set_ylabel('Log GDP per capita, PPP, 1995')
ax.set_title('Figure 2: OLS relationship between expropriation risk and income')
plt.show()
```

The most common technique to estimate the parameters (β's) of the linear model is Ordinary Least Squares (OLS).

As the name implies, an OLS model is solved by finding the parameters that minimize *the sum of squared residuals*, i.e.

$$\min_{\hat{\beta}} \sum_{i=1}^{N} \hat{u}_i^2$$

where $\hat{u}_i$ is the difference between the observation and the predicted value of the dependent variable.

To estimate the constant term β_0, we need to add a column of 1's to our dataset (consider the equation if β_0 was replaced with $\beta_0 x_i$ and $x_i = 1$)

In [23]:
```python
df1['const'] = 1
df1
```

Out[23]:

	shortnam	euro1900	excolony	avexpr	logpgp95	cons1	cons90	democ00a	cons00a	extmort4
0	AFG	0.000000	1.0	NaN	NaN	1.0	2.0	1.0	1.0	93.699997
1	AGO	8.000000	1.0	5.363636	7.770645	3.0	3.0	0.0	1.0	280.000000
2	ARE	0.000000	1.0	7.181818	9.804219	NaN	NaN	NaN	NaN	NaN
3	ARG	60.000004	1.0	6.386364	9.133459	1.0	6.0	3.0	3.0	68.900002
4	ARM	0.000000	0.0	NaN	7.682482	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...
158	YUG	100.000000	0.0	6.318182	NaN	NaN	NaN	NaN	NaN	NaN
159	ZAF	22.000000	1.0	6.863636	8.885994	3.0	7.0	3.0	3.0	15.500000
160	ZAR	8.000000	1.0	3.500000	6.866933	1.0	1.0	0.0	1.0	240.000000
161	ZMB	3.000000	1.0	6.636364	6.813445	3.0	1.0	0.0	1.0	NaN
162	ZWE	7.200000	1.0	6.000000	7.696213	7.0	3.0	0.0	1.0	NaN

163 rows × 14 columns

Now we can construct our model in `statsmodels` using the OLS function.

We will use `pandas` dataframes with `statsmodels`, however standard arrays can also be used as arguments

In [30]:
```python
reg1 = sm.OLS(endog=df1['logpgp95'], exog=df1[['const','avexpr']], \
    missing='drop')
type(reg1)
```

Out[30]:
```
statsmodels.regression.linear_model.OLS
```

So far we have simply constructed our model.

We need to use `.fit()` to obtain parameter estimates $\hat{\beta}_0$ and $\hat{\beta}_1$

In [31]:
```python
results = reg1.fit()
type(results)
```

Out[31]:
```
statsmodels.regression.linear_model.RegressionResultsWrapper
```

We now have the fitted regression model stored in `results`.

To view the OLS regression results, we can call the `.summary()` method.

Note that an observation was mistakenly dropped from the results in the original paper (see the note located in maketable2.do from Acemoglu's webpage), and thus the coefficients differ slightly.

In [32]:
```
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               logpgp95   R-squared:                       0.611
Model:                            OLS   Adj. R-squared:                  0.608
Method:                 Least Squares   F-statistic:                     171.4
Date:                Wed, 11 Mar 2020   Prob (F-statistic):           4.16e-24
Time:                        18:09:16   Log-Likelihood:                -119.71
No. Observations:                 111   AIC:                             243.4
Df Residuals:                     109   BIC:                             248.8
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          4.6261      0.301     15.391      0.000       4.030       5.222
avexpr         0.5319      0.041     13.093      0.000       0.451       0.612
==============================================================================
Omnibus:                        9.251   Durbin-Watson:                   1.689
Prob(Omnibus):                  0.010   Jarque-Bera (JB):                9.170
Skew:                          -0.680   Prob(JB):                       0.0102
Kurtosis:                       3.362   Cond. No.                         33.2
==============================================================================

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

From our results, we see that

- The intercept $\hat{\beta}_0 = 4.63$.
- The slope $\hat{\beta}_1 = 0.53$.
- The positive $\hat{\beta}_1$ parameter estimate implies that. institutional quality has a positive effect on economic outcomes, as we saw in the figure.
- The p-value of 0.000 for $\hat{\beta}_1$ implies that the effect of institutions on GDP is statistically significant (using p < 0.05 as a rejection rule).
- The R-squared value of 0.611 indicates that around 61% of variation in log GDP per capita is explained by protection against expropriation.

Using our parameter estimates, we can now write our estimated relationship as

$$\widehat{logpgp95}_i = 4.63 + 0.53\,\text{avexpr}_i + 0.89\,\text{excolony}_i$$

This equation describes the line that best fits our data, as shown in Figure 2.

We can use this equation to predict the level of log GDP per capita for a value of the index of expropriation protection.

For example, for a country with an index value of 7.07 (the average for the dataset), we find that their predicted level of log GDP per capita in 1995 is 8.38.

In [33]:
```
mean_expr = np.mean(df1_subset['avexpr'])
mean_expr
```

Out[33]: 6.515625

In [34]:
```
predicted_logpdp95 = 4.63 + 0.53 * 6.516
predicted_logpdp95
```

Out[34]: 8.08348

An easier (and more accurate) way to obtain this result is to use `.predict()` and set $constant = 1$ and $avexpr_i = mean\_expr$

In [35]:
```
results.predict(exog=[1, mean_expr])
```

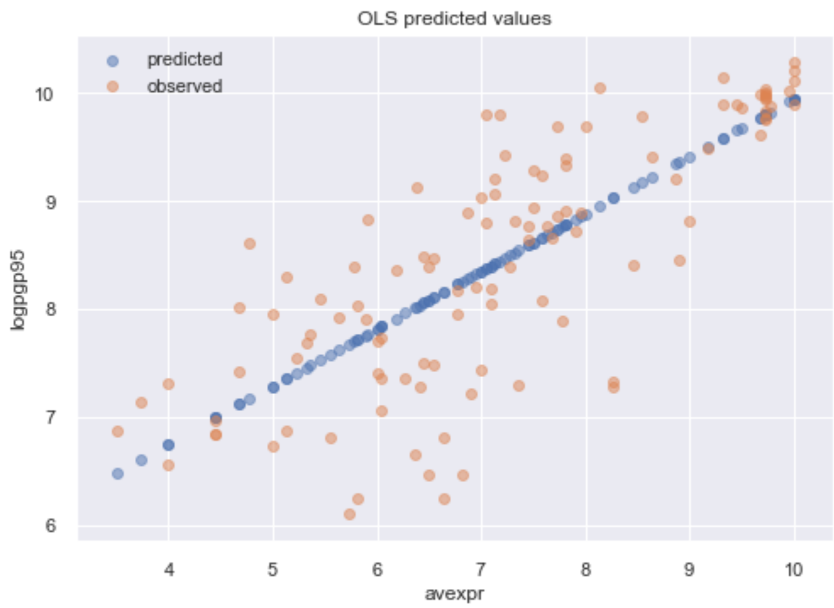
Out[35]:
```
array([8.09156367])
```

We can obtain an array of predicted $logpgp95_i$ for every value of $avexpr_i$ in our dataset by calling `.predict()` on our results.

Plotting the predicted values against $avexpr_i$ shows that the predicted values lie along the linear line that we fitted above.

The observed values of $logpgp95_i$ are also plotted for comparison purposes

In [36]:
```
# Drop missing observations from whole sample

df1_plot = df1.dropna(subset=['logpgp95', 'avexpr'])

# Plot predicted values

fix, ax = plt.subplots()
ax.scatter(df1_plot['avexpr'], results.predict(), alpha=0.5,
        label='predicted')

# Plot observed values

ax.scatter(df1_plot['avexpr'], df1_plot['logpgp95'], alpha=0.5,
        label='observed')

ax.legend()
ax.set_title('OLS predicted values')
ax.set_xlabel('avexpr')
ax.set_ylabel('logpgp95')
plt.show()
```



Extending the Linear Regression Model

So far we have only accounted for institutions affecting economic performance - almost certainly there are numerous other factors affecting GDP that are not included in our model.

Leaving out variables that affect $logpgp95_i$ will result in **omitted variable bias**, yielding biased and inconsistent parameter estimates.

We can extend our bivariate regression model to a **multivariate regression model** by adding in other factors that may affect $logpgp95_i$.

[AJR01] (https://python.quantecon.org/zreferences.html#acemoglu2001) consider other factors such as:

- the effect of climate on economic outcomes; latitude is used to proxy this
- differences that affect both economic performance and institutions, eg. cultural, historical, etc.; controlled for with the use of continent dummies

Let's estimate some of the extended models considered in the paper (Table 2) using data from `maketable2.dta`

In [4]:
```
df2 = pd.read_stata('https://github.com/QuantEcon/lecture-source-py/blob/master/s

# Add constant term to dataset
df2['const'] = 1

# Create lists of variables to be used in each regression
X1 = ['const', 'avexpr']
X2 = ['const', 'avexpr', 'lat_abst']
X3 = ['const', 'avexpr', 'lat_abst', 'asia', 'africa', 'other']

# Estimate an OLS regression for each set of variables
reg1 = sm.OLS(df2['logpgp95'], df2[X1], missing='drop').fit()
reg2 = sm.OLS(df2['logpgp95'], df2[X2], missing='drop').fit()
reg3 = sm.OLS(df2['logpgp95'], df2[X3], missing='drop').fit()
```

```
/Users/wasin_siwasarit/anaconda3/lib/python3.7/site-packages/statsmodels/base/da
ta.py:480: FutureWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ix-indexer-
is-deprecated (http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.h
tml#ix-indexer-is-deprecated)
  if hasattr(x, 'ix'):
```

Now that we have fitted our model, we will use `summary_col` to display the results in a single table (model numbers correspond to those in the paper)

In [13]:
```
info_dict={'Adj-R-squared' : lambda x: f"{x.rsquared_adj:.2f}",
           'No. observations' : lambda x: f"{int(x.nobs):d}"}

results_table = summary_col(results=[reg1,reg2,reg3],
                            float_format='%0.2f',
                            stars = True,
                            model_names=['Model 1',
                                         'Model 3',
                                         'Model 4'],
                            info_dict=info_dict,
                            regressor_order=['const',
                                             'avexpr',
                                             'lat_abst',
                                             'asia',
                                             'africa'])

results_table.add_title('Table 2 - OLS Regressions')

print(results_table)
```

```
        Table 2 - OLS Regressions
=========================================
                 Model 1 Model 3 Model 4
-----------------------------------------
const            4.63*** 4.87*** 5.85***
                 (0.30)  (0.33)  (0.34)
avexpr           0.53*** 0.46*** 0.39***
                 (0.04)  (0.06)  (0.05)
lat_abst                 0.87*   0.33
                         (0.49)  (0.45)
asia                             -0.15
                                 (0.15)
africa                           -0.92***
                                 (0.17)
other                            0.30
                                 (0.37)
Adj-R-squared    0.61    0.62    0.70
No. observations 111     111     111
=========================================
Standard errors in parentheses.
* p<.1, ** p<.05, ***p<.01
```

In [25]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
boston = load_boston()
boston.DESCR
boston.feature_names
x = boston.data
y = boston.target
```

In [26]:
```
boston.DESCR
boston.feature_names
```

Out[26]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [27]:
```
boston.feature_names
```

Out[27]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [28]:
```
print(boston.data.shape)
```

```
(506, 13)
```

In [29]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attr
ibute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.
ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 ot
herwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by tow
n
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://arch
ive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that a
ddress regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Dat
a and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Pr
oceedings on the Tenth International Conference of Machine Learning, 236-243, Un
iversity of Massachusetts, Amherst. Morgan Kaufmann.
```

In [30]:
```
boston.data
type(boston.data)
```

Out[30]: numpy.ndarray

In [31]:
```
bos = pd.DataFrame(boston.data)
print(bos)
type(bos)
```

```
           0     1      2    3      4      5     6       7    8      9    10  \
0    0.00632  18.0   2.31  0.0  0.538  6.575  65.2  4.0900  1.0  296.0  15.3
1    0.02731   0.0   7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0  17.8
2    0.02729   0.0   7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0  17.8
3    0.03237   0.0   2.18  0.0  0.458  6.998  45.8  6.0622  3.0  222.0  18.7
4    0.06905   0.0   2.18  0.0  0.458  7.147  54.2  6.0622  3.0  222.0  18.7
..       ...   ...    ...  ...    ...    ...   ...     ...  ...    ...   ...
501  0.06263   0.0  11.93  0.0  0.573  6.593  69.1  2.4786  1.0  273.0  21.0
502  0.04527   0.0  11.93  0.0  0.573  6.120  76.7  2.2875  1.0  273.0  21.0
503  0.06076   0.0  11.93  0.0  0.573  6.976  91.0  2.1675  1.0  273.0  21.0
504  0.10959   0.0  11.93  0.0  0.573  6.794  89.3  2.3889  1.0  273.0  21.0
505  0.04741   0.0  11.93  0.0  0.573  6.030  80.8  2.5050  1.0  273.0  21.0

         11    12
0    396.90  4.98
1    396.90  9.14
2    392.83  4.03
3    394.63  2.94
4    396.90  5.33
..      ...   ...
501  391.99  9.67
502  396.90  9.08
503  396.90  5.64
504  393.45  6.48
505  396.90  7.88

[506 rows x 13 columns]
```

Out[31]: pandas.core.frame.DataFrame

In [32]:
```
bos.columns = boston.feature_names
print(bos.head())
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

In [34]:
```
bos['PRICE'] = boston.target
bos['const'] = 1
bos
```

Out[34]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 15 columns

In [ ]:
```
X2=bos[['CRIM','TAX']]
X2
```

In [ ]:
```
print(bos.describe())
```

Task 1

ให้นักศึกษา ทำตารางเปรียบเทียบผลการประมาณ ค่า เปรียบเทียบ 3 สมการถดถอยต่อไปนี้

$$price = \beta_1 + \beta_2 CRIM + \epsilon$$

$$price = \beta_1 + \beta_2 CRIM + \beta_3 RM + \epsilon$$

$$price = \beta_1 + \beta_2 CRIM + \beta_3 RM + \beta_4 NOX + \epsilon$$

In [16]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
boston = load_boston()
boston.DESCR
boston.feature_names
x = boston.data
y = boston.target

# Fitting a model is trivial: call the ``fit`` method in LinearRegression:
lr = LinearRegression()
reg=lr.fit(x, y)
bos = pd.DataFrame(boston.data)
bos.columns = boston.feature_names
```

In [19]:
```
bos['PRICE'] = boston.target
print(bos.head())
X2=bos[['CRIM','TAX']]
Y = bos['PRICE']
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  PRICE
0     15.3  396.90   4.98   24.0
1     17.8  396.90   9.14   21.6
2     17.8  392.83   4.03   34.7
3     18.7  394.63   2.94   33.4
4     18.7  396.90   5.33   36.2
```

In [20]:
```
lr = LinearRegression()
reg=lr.fit(X2, Y)
```

In [21]:
```
#To retrieve the intercept:
print(reg.intercept_)
```

```
31.379119420390783
```

In [22]:
```
#For retrieving the slope:
print(reg.coef_)
```

```
[-0.18661658 -0.0200177 ]
```

In [24]:
```
y_pred = reg.predict(X2)
y_pred
```

Out[24]:
```
array([25.45270022, 26.52973904, 26.52974278, 26.9291488 , 26.92230371,
       26.92961908, 25.13713773, 25.12663868, 25.11419322, 25.12188182,
       25.1116459 , 25.13169226, 25.1361132 , 25.11616126, 25.114631  ,
       25.11660354, 25.0370041 , 25.08734019, 25.08388592, 25.0982386 ,
       25.00008014, 25.07468012, 25.00368558, 25.04922749, 25.09367396,
       25.07682621, 25.10829537, 25.05532238, 25.08943216, 25.04661112,
       25.02265702, 24.9808717 , 24.97466297, 25.01875487, 24.93270596,
       25.78220538, 25.77599665, 25.77922512, 25.76151334, 26.32950231,
       26.32839007, 26.69121244, 26.68858861, 26.68525564, 26.69209887,
       26.68300505, 26.67984376, 26.67220928, 26.66761851, 26.67398213,
       26.49825935, 26.50672428, 26.50481519, 26.50552247, 21.98827921,
       26.85267223, 25.10974373, 26.25191536, 25.66526913, 25.6748183 ,
       25.66622647, 25.66204813, 25.67351385, 25.67048506, 27.0516549 ,
       24.62646552, 24.62498191, 24.462209  , 24.44771823, 24.44909546,
       25.25724954, 25.24409307, 25.25661877, 25.2372573 , 23.39733879,
       23.39432306, 23.39312685, 23.39582533, 23.40153766, 23.3964225 ,
       25.74646962, 25.74581833, 25.74731686, 25.74751841, 26.4253061 ,
       26.42404457, 26.42506536, 26.42140208, 25.96377739, 25.96444548,
       25.96559877, 25.96700212, 25.96649639, 25.96897466, 25.96632657,
       25.83145899, 25.8327653 , 25.83168479, 25.83895538, 25.84143178,
       23.66457944, 23.67098785, 23.64963145, 23.65283193, 23.66627019,
       23.66757277, 23.6603731 , 23.66784336, 23.66843121, 23.64312413,
       23.67218033, 22.71265375, 22.70846421, 22.69002089, 22.70491476,
       22.69949728, 22.70691716, 22.70329679, 22.70710377, 22.70445755,
       27.60291677, 27.60242037, 27.59843797, 27.58772805, 27.59741158,
       27.58424951, 27.54350552, 22.58302197, 22.57065302, 22.46692779,
       22.56792282, 22.40876127, 22.52127054, 22.56983378, 22.44921415,
       22.52729266, 22.57117368, 22.56563304, 22.58476683, 22.5297672 ,
       22.57709689, 22.32745243, 22.69222253, 22.54734275, 22.79323995,
       22.86796123, 22.90981746, 22.86996176, 22.87698414, 22.80178139,
       23.0028365 , 23.0327474 , 23.10174702, 22.9109129 , 23.04813767,
       22.65229405, 22.85539447, 23.08364521, 23.06138931, 23.04605316,
       23.07433677, 23.03889828, 22.96977364, 23.02851121, 22.89352397,
       22.76631864, 22.93685074, 22.97602343, 22.88269274, 22.85486261,
       23.08666093, 22.88017342, 25.4279138 , 25.43675197, 25.43811613,
       25.44144351, 25.44077542, 25.44375569, 25.44148456, 27.5049165 ,
       27.50340864, 27.50284879, 27.49871523, 27.49702635, 27.50019883,
       27.50441823, 27.50524868, 23.39737798, 23.38859953, 23.39645423,
       23.39515164, 23.39917696, 23.39590557, 26.07034709, 26.07174298,
       26.27202824, 24.78581028, 24.78458794, 24.78626376, 23.3261248 ,
       23.32868518, 24.40653019, 24.40889649, 26.88860394, 26.89140505,
       25.80875774, 25.79135201, 25.78719046, 25.80886038, 25.75290526,
       25.80165884, 25.76408919, 25.79368472, 25.80799261, 25.78018114,
       25.79726216, 25.84572396, 25.84114625, 25.83357709, 25.83291273,
       25.16685938, 25.15759947, 25.11731828, 25.1189717 , 25.17483911,
       25.13535104, 25.16237125, 25.15672797, 25.17803772, 25.15124144,
       25.13347181, 25.1472889 , 25.12632626, 25.17182712, 25.15009748,
       25.17201746, 25.13653605, 25.13816895, 25.35842416, 25.35654306,
       25.35266703, 25.35400508, 25.35460598, 25.35000215, 24.73481982,
       24.73757242, 24.70985986, 24.73659455, 24.74259987, 24.73768439,
       24.74709546, 24.73332502, 24.75793602, 24.70442745, 25.06455024,
       25.06692214, 26.49192997, 25.98032259, 25.97062413, 25.97190432,
       25.99365262, 25.99477045, 25.99737935, 25.9404389 , 25.99179392,
       25.95231518, 25.94782145, 25.98651827, 25.99357984, 26.89825509,
       26.85934366, 26.88491947, 26.89378562, 26.87376539, 26.28409048,
       26.27670046, 26.27508623, 26.28318912, 26.27973485, 27.0160354 ,
       27.04861865, 27.04838165, 27.04385806, 27.41281331, 25.67238361,
       25.37176351, 26.55118623, 25.50670881, 25.50536704, 25.50591383,
       26.46824712, 26.46006585, 26.46803625, 25.57857969, 25.57870286,
       25.56987029, 25.58397851, 25.56768501, 24.20071548, 24.20240436,
       24.20453926, 24.78669484, 24.77600358, 24.77463381, 26.92489768,
       26.92496486, 26.92118774, 26.92598565, 25.20173978, 25.22853419,
       24.80191375, 25.14623441, 25.24490233, 25.24346724, 25.22483918,
       25.24641952, 25.23434356, 25.2479759 , 25.21871442, 25.20500743,
       25.60276201, 25.60015125, 25.56851041, 25.58105477, 25.5703859 ,
       25.5982347 , 25.57740642, 25.58905876, 22.75915915, 22.75895947,
```

In [25]:
```
df = pd.DataFrame({'Actual': y.flatten(), 'Predicted': y_pred.flatten()})
df
```

Out[25]:

	Actual	Predicted
0	24.0	25.452700
1	21.6	26.529739
2	34.7	26.529743
3	33.4	26.929149
4	36.2	26.922304
...	...	...
501	22.4	25.902599
502	20.6	25.905839
503	23.9	25.902948
504	22.0	25.893835
505	11.9	25.905439

506 rows × 2 columns

In [26]:
```
from sklearn import metrics
```

In [27]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(y, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y, y_pred)))
```

```
Mean Absolute Error: 5.789776955289234
Mean Squared Error: 64.18906314885675
Root Mean Squared Error: 8.011807732893791
```

In [28]:
```
from sklearn.model_selection import KFold
```

In [29]:
```
kf = KFold(n_splits=10)
```

In [30]:
```
err = 0
for train,test in kf.split(x):
    reg=lr.fit(x[train],y[train])
    y_pred =reg.predict(x[test])
    e = y[test]-y_pred
    err += np.sum(e*e)
rmse_10cv = np.sqrt(err/len(x))
print('RMSE on 10-fold CV: {}'.format(rmse_10cv))
```

```
RMSE on 10-fold CV: 5.877045136800733
```

Task2

We can extend our bivariate regression model to a multivariate regression model by adding in other factors that may affect $logpgp95i$.

[AJR01] consider other factors such as:

```
the effect of climate on economic outcomes; latitude is used to proxy thi
s
differences that affect both economic performance and institutions, eg. c
ultural, historical, etc.; controlled for with the use of continent dummi
es
```

Create lists of variables to be used in each regression

X1 = ['const', 'avexpr'] X2 = ['const', 'avexpr', 'lat_abst'] X3 = ['const', 'avexpr', 'lat_abst', 'asia', 'africa', 'other']

Compare the rmse of these models by using the 10-Kfold

Endogeneity

As [AJR01] (https://python.quantecon.org/zreferences.html#acemoglu2001) discuss, the OLS models likely suffer from **endogeneity** issues, resulting in biased and inconsistent model estimates.

Namely, there is likely a two-way relationship between institutions and economic outcomes:

- richer countries may be able to afford or prefer better institutions
- variables that affect income may also be correlated with institutional differences
- the construction of the index may be biased; analysts may be biased towards seeing countries with higher income having better institutions

To deal with endogeneity, we can use **two-stage least squares (2SLS) regression**, which is an extension of OLS regression.

This method requires replacing the endogenous variable $avexpr_i$ with a variable that is:

1. correlated with $avexpr_i$
2. not correlated with the error term (ie. it should not directly affect the dependent variable, otherwise it would be correlated with u_i due to omitted variable bias)

The new set of regressors is called an **instrument**, which aims to remove endogeneity in our proxy of institutional differences.

The main contribution of [AJR01] (https://python.quantecon.org/zreferences.html#acemoglu2001) is the use of settler mortality rates to instrument for institutional differences.

They hypothesize that higher mortality rates of colonizers led to the establishment of institutions that were more extractive in nature (less protection against expropriation), and these institutions still persist today.

Using a scatterplot (Figure 3 in [AJR01] (https://python.quantecon.org/zreferences.html#acemoglu2001)), we can see protection against expropriation is negatively correlated with settler mortality rates, coinciding with the authors' hypothesis and satisfying the first condition of a valid instrument.

In [ ]:
```
# Dropping NA's is required to use numpy's polyfit
df1_subset2 = df1.dropna(subset=['logem4', 'avexpr'])

X = df1_subset2['logem4']
y = df1_subset2['avexpr']
labels = df1_subset2['shortnam']

# Replace markers with country labels
fig, ax = plt.subplots()
ax.scatter(X, y, marker='')

for i, label in enumerate(labels):
    ax.annotate(label, (X.iloc[i], y.iloc[i]))

# Fit a linear trend line
ax.plot(np.unique(X),
         np.poly1d(np.polyfit(X, y, 1))(np.unique(X)),
         color='black')

ax.set_xlim([1.8,8.4])
ax.set_ylim([3.3,10.4])
ax.set_xlabel('Log of Settler Mortality')
ax.set_ylabel('Average Expropriation Risk 1985-95')
ax.set_title('Figure 3: First-stage relationship between settler mortality \
    and expropriation risk')
plt.show()
```

The second condition may not be satisfied if settler mortality rates in the 17th to 19th centuries have a direct effect on current GDP (in addition to their indirect effect through institutions).

For example, settler mortality rates may be related to the current disease environment in a country, which could affect current economic performance.

[AJR01] (https://python.quantecon.org/zreferences.html#acemoglu2001) argue this is unlikely because:

- The majority of settler deaths were due to malaria and yellow fever and had a limited effect on local people.
- The disease burden on local people in Africa or India, for example, did not appear to be higher than average, supported by relatively high population densities in these areas before colonization.

As we appear to have a valid instrument, we can use 2SLS regression to obtain consistent and unbiased parameter estimates.

First stage

The first stage involves regressing the endogenous variable ($avexpr_i$) on the instrument.

The instrument is the set of all exogenous variables in our model (and not just the variable we have replaced).

Using model 1 as an example, our instrument is simply a constant and settler mortality rates $logem4_i$.

Therefore, we will estimate the first-stage regression as

$$avexpr_i = \delta_0 + \delta_1 logem4_i + v_i$$

The data we need to estimate this equation is located in `maketable4.dta` (only complete data, indicated by `baseco = 1`, is used for estimation)

In [ ]:
```
# Import and select the data
df4 = pd.read_stata('https://github.com/QuantEcon/lecture-source-py/blob/master/s
df4 = df4[df4['baseco'] == 1]

# Add a constant variable
df4['const'] = 1

# Fit the first stage regression and print summary
results_fs = sm.OLS(df4['avexpr'],
                    df4[['const', 'logem4']],
                    missing='drop').fit()
print(results_fs.summary())
```

Second stage

We need to retrieve the predicted values of $avexpr_i$ using `.predict()` .

We then replace the endogenous variable $avexpr_i$ with the predicted values $\widehat{avexpr}_i$ in the original linear model.

Our second stage regression is thus

$$logpgp95_i = \beta_0 + \beta_1 \widehat{avexpr}_i + u_i$$

In [ ]:
```
df4['predicted_avexpr'] = results_fs.predict()

results_ss = sm.OLS(df4['logpgp95'],
                    df4[['const', 'predicted_avexpr']]).fit()
print(results_ss.summary())
```

The second-stage regression results give us an unbiased and consistent estimate of the effect of institutions on economic outcomes.

The result suggests a stronger positive relationship than what the OLS results indicated.

Note that while our parameter estimates are correct, our standard errors are not and for this reason, computing 2SLS ‘manually’ (in stages with OLS) is not recommended.

We can correctly estimate a 2SLS regression in one step using the linearmodels (https://github.com/bashtage/linearmodels) package, an extension of `statsmodels`

Note that when using `IV2SLS` , the exogenous and instrument variables are split up in the function arguments (whereas before the instrument included exogenous variables)

In [ ]:
```
iv = IV2SLS(dependent=df4['logpgp95'],
            exog=df4['const'],
            endog=df4['avexpr'],
            instruments=df4['logem4']).fit(cov_type='unadjusted')

print(iv.summary)
```

Given that we now have consistent and unbiased estimates, we can infer from the model we have estimated that institutional differences (stemming from institutions set up during colonization) can help to explain differences in income levels across countries today.

[AJR01] (https://python.quantecon.org/zreferences.html#acemoglu2001) use a marginal effect of 0.94 to calculate that the difference in the index between Chile and Nigeria (ie. institutional quality) implies up to a 7-fold difference in income, emphasizing the significance of institutions in economic development.

Summary

We have demonstrated basic OLS and 2SLS regression in `statsmodels` and `linearmodels`.

If you are familiar with R, you may want to use the [formula interface (http://www.statsmodels.org/dev/example_formulas.html)](http://www.statsmodels.org/dev/example_formulas.html) to `statsmodels`, or consider using [r2py (https://rpy2.bitbucket.io/)](https://rpy2.bitbucket.io/) to call R from within Python.

Exercises

Exercise 1

In the lecture, we think the original model suffers from endogeneity bias due to the likely effect income has on institutional development.

Although endogeneity is often best identified by thinking about the data and model, we can formally test for endogeneity using the **Hausman test**.

We want to test for correlation between the endogenous variable, $avexpr_i$, and the errors, u_i

$$H_0 : Cov(avexpr_i, u_i) = 0 \quad (no\ endogeneity)$$
$$H_1 : Cov(avexpr_i, u_i) \neq 0 \quad (endogeneity)$$

This test is running in two stages.

First, we regress $avexpr_i$ on the instrument, $logem4_i$

$$avexpr_i = \pi_0 + \pi_1 logem4_i + \upsilon_i$$

Second, we retrieve the residuals $\hat{\upsilon}_i$ and include them in the original equation

$$logpgp95_i = \beta_0 + \beta_1 avexpr_i + \alpha\hat{\upsilon}_i + u_i$$

If α is statistically significant (with a p-value < 0.05), then we reject the null hypothesis and conclude that $avexpr_i$ is endogenous.

Using the above information, estimate a Hausman test and interpret your results.

Exercise 2

The OLS parameter β can also be estimated using matrix algebra and `numpy` (you may need to review the [numpy (https://lectures.quantecon.org/py/numpy.html)](https://lectures.quantecon.org/py/numpy.html) lecture to complete this exercise).

The linear equation we want to estimate is (written in matrix form)

$$y = X\beta + u$$

To solve for the unknown parameter β, we want to minimize the sum of squared residuals

$$\min_{\hat{\beta}} \hat{u}'\hat{u}$$

Rearranging the first equation and substituting into the second equation, we can write

$$\min_{\hat{\beta}} \ (Y - X\hat{\beta})'(Y - X\hat{\beta})$$

Solving this optimization problem gives the solution for the $\hat{\beta}$ coefficients

$$\hat{\beta} = (X'X)^{-1}X'y$$

Using the above information, compute $\hat{\beta}$ from model 1 using `numpy` - your results should be the same as those in the `statsmodels` output from earlier in the lecture.

Solutions

Exercise 1

In [ ]:

```
# Load in data
df4 = pd.read_stata('https://github.com/QuantEcon/lecture-source-py/blob/master/s

# Add a constant term
df4['const'] = 1

# Estimate the first stage regression
reg1 = sm.OLS(endog=df4['avexpr'],
              exog=df4[['const', 'logem4']],
              missing='drop').fit()

# Retrieve the residuals
df4['resid'] = reg1.resid

# Estimate the second stage residuals
reg2 = sm.OLS(endog=df4['logpgp95'],
              exog=df4[['const', 'avexpr', 'resid']],
              missing='drop').fit()

print(reg2.summary())
```

The output shows that the coefficient on the residuals is statistically significant, indicating $avexpr_i$ is endogenous.

Exercise 2

In [ ]:
```
# Load in data
df1 = pd.read_stata('https://github.com/QuantEcon/lecture-source-py/blob/master/s
df1 = df1.dropna(subset=['logpgp95', 'avexpr'])

# Add a constant term
df1['const'] = 1

# Define the X and y variables
y = np.asarray(df1['logpgp95'])
X = np.asarray(df1[['const', 'avexpr']])

# Compute β_hat
β_hat = np.linalg.solve(X.T @ X, X.T @ y)

# Print out the results from the 2 x 1 vector β_hat
print(f'β_0 = {β_hat[0]:.2}')
print(f'β_1 = {β_hat[1]:.2}')
```

It is also possible to use `np.linalg.inv(X.T @ X) @ X.T @ y` to solve for β, however `.solve()` is preferred as it involves fewer computations.