Intro to Programming for Public Policy Week 8 Statistics, Regression, and Visualizations

Eric Potash

May 17, 2018

Writing files

Reading

Recall that we could read a file as follows:

```
file = open('filename.txt')
contents = file.readlines()
```

Writing

Similarly, we can write:

```python
file = open('filename.txt', 'w')
file.write('First line')
file.write('Second line')
file.write('Still second line\n')
file.close()
```

- Note the format argument `'w'`, indicating that we want to write to the specified file.
    - `'w'` will overwrite any existing file
    - To append, use `'a'`
- If you don't close the file when you're done, bad things can happen.

with block

To avoid remembering to close the file, you can wrap your witing in a `with` block. Python will automatically close the file when the block is finished.

```python
with open('filename.txt', 'w') as file:
    file.write('First line')
    file.write('Second line')
    file.write('Still second line\n')
```

Reshaping/Pivoting

Hierarchical index

When we groupby with two keys the result has what's called a
multiindex or a hierarchical index:

```
In [1]: import pandas as pd
        %matplotlib inline
```

```
In [2]: crimes = pd.read_csv('Crimes_-_2001_to_present.csv')
        crimes['Date'] = pd.to_datetime(crimes.Date)
```

```
In [10]: crimes['Day'] = crimes.Date.dt.date
         crimes['Month'] = crimes.Date.dt.month
```

```
In [16]: crime_months = crimes.groupby(['Community Area', 'Month']).size()
         crime_months
```

```
Out[16]: Community Area  Month
         0               5          1
         1               1        289
                         2        295
                         3        289
                         4        336
                         5        369
                         6        347
                         7        410
                         8        384
                         9        355
                         10       406
                         11       360
                         12       291
         2               1        307
```

Simpler example

```
>>> df = pd.DataFrame([[0, 1, 2], [3, 4, 5]],
                      index=['Ohio', 'Colorado'],
                      columns=['one', 'two', 'three'])
>>> df
          one   two   three
Ohio        0     1       2
Colorado    3     4       5
```

To *reshape* into a hierarchical index use `stack()`:

```
>>> s = df.stack()
>>> s
Ohio      one       0
          two       1
          three     2
Colorado  one       3
          two       4
          three     5
dtype: int64
```

unstack()

The inverse to this operation is unstack():

```
>>> s.unstack()
          one  two  three
Ohio        0    1      2
Colorado    3    4      5
```

Unstack crimes

`crime_months.unstack('Community Area')`

Community Area	0	1	2	3	4	5	6	7	8	9	...	68	69	70	71	72	73
Month																	
1	NaN	289.0	307.0	265.0	163.0	95.0	447.0	367.0	887.0	18.0	...	490.0	557.0	269.0	620.0	70.0	278.0
2	NaN	295.0	236.0	255.0	136.0	100.0	385.0	305.0	795.0	32.0	...	392.0	451.0	226.0	572.0	68.0	199.0
3	NaN	289.0	284.0	242.0	132.0	100.0	386.0	349.0	812.0	20.0	...	443.0	486.0	240.0	559.0	59.0	229.0
4	NaN	336.0	267.0	290.0	118.0	93.0	470.0	308.0	908.0	17.0	...	487.0	502.0	209.0	677.0	66.0	239.0
5	1.0	369.0	298.0	304.0	174.0	147.0	485.0	371.0	937.0	15.0	...	608.0	568.0	184.0	697.0	81.0	280.0
6	NaN	347.0	299.0	299.0	160.0	138.0	551.0	387.0	1085.0	21.0	...	558.0	550.0	194.0	621.0	110.0	291.0
7	NaN	410.0	327.0	345.0	171.0	138.0	555.0	401.0	1150.0	35.0	...	553.0	607.0	186.0	721.0	82.0	269.0
8	NaN	384.0	329.0	334.0	214.0	158.0	544.0	382.0	1237.0	35.0	...	520.0	604.0	191.0	683.0	80.0	257.0
9	NaN	355.0	315.0	339.0	198.0	115.0	512.0	397.0	1123.0	27.0	...	509.0	554.0	196.0	609.0	67.0	282.0
10	NaN	406.0	306.0	331.0	184.0	119.0	570.0	389.0	1165.0	16.0	...	476.0	512.0	199.0	628.0	76.0	272.0
11	NaN	360.0	295.0	263.0	172.0	99.0	375.0	405.0	1062.0	16.0	...	458.0	550.0	211.0	534.0	70.0	257.0
12	NaN	291.0	320.0	281.0	168.0	115.0	432.0	397.0	1104.0	21.0	...	418.0	516.0	185.0	579.0	59.0	217.0

Time series

`crime_months.unstack(level='Community Area')[[1,2,3, 4]].plot(figsize=(10,5))`

<matplotlib.axes._subplots.AxesSubplot at 0x7ff97ba02410>
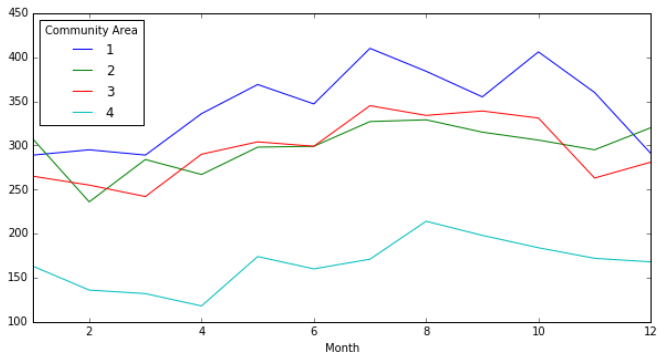
Visualizations

Matplotlib

- Matplotlib is the core of all plotting in python
- When you use pandas (or seaborn) to plot, it is using matplotlib.
- As a result, matplotlib settings will affect pandas plots

Matplotlib rcParams

```
In [75]:  import pandas as pd
          import matplotlib.pyplot as plt
          %matplotlib inline

          plt.rcParams['figure.figsize'] = 9,5
          plt.rcParams['font.family'] = 'sans-serif'
```

Axes

Plot functions like DataFrame.plot() or Series.hist() return matplotlib Axes objects:

```
>>> s = pd.Series([1,2,3,4])
>>> ax = s.plot()
>>> ax
<matplotlib.axes._subplots.AxesSubplot at 0x7fa917dac410>
```

You can use these Axes objects to customize the plot.

Axes customization

```
ax = s.plot()
ax.set_xlabel('X axis', fontsize=14)
ax.set_ylabel('Y axis', fontsize=14)

ax.set_title('Title of plot', fontsize=20)
```

Out[10]: <matplotlib.text.Text at 0x7f9d0b7363d0>
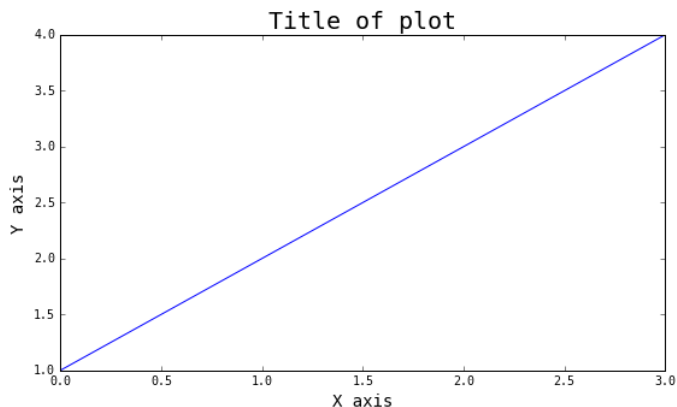
Wages data

```
In [5]: import pandas as pd
        %matplotlib inline
```

```
In [23]: df = pd.read_csv('wages.csv')
         df
```

Out[23]:

	EDUCATION	SOUTH	SEX	EXPERIENCE	UNION	WAGE	AGE	RACE	OCCUPATION	SECTOR	MARR
0	8	NORTH	FEMALE	21	False	5.10	35	Hispanic	Other	Manufacturing	True
1	9	NORTH	FEMALE	42	False	4.95	57	White	Other	Manufacturing	True
2	12	NORTH	MALE	1	False	6.67	19	White	Other	Manufacturing	False
3	12	NORTH	MALE	4	False	4.00	22	White	Other	Other	False
4	12	NORTH	MALE	17	False	7.50	35	White	Other	Other	True
5	13	NORTH	MALE	9	True	13.07	28	White	Other	Other	False
6	10	SOUTH	MALE	27	False	4.45	43	White	Other	Other	False
7	12	NORTH	MALE	9	False	19.47	27	White	Other	Other	False
8	16	NORTH	MALE	11	False	13.28	33	White	Other	Manufacturing	True
9	12	NORTH	MALE	9	False	8.75	27	White	Other	Other	False
10	12	NORTH	MALE	17	True	11.35	35	White	Other	Other	True

From 1985 CPS, CSV here.

Boxplot

`df.boxplot('WAGE', by='OCCUPATION')`

`<matplotlib.axes._subplots.AxesSubplot at 0x7faf9368ca90>`
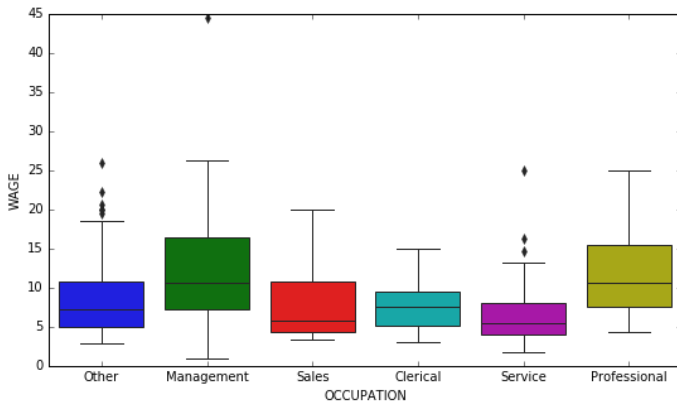

Boxplot grouped by OCCUPATION
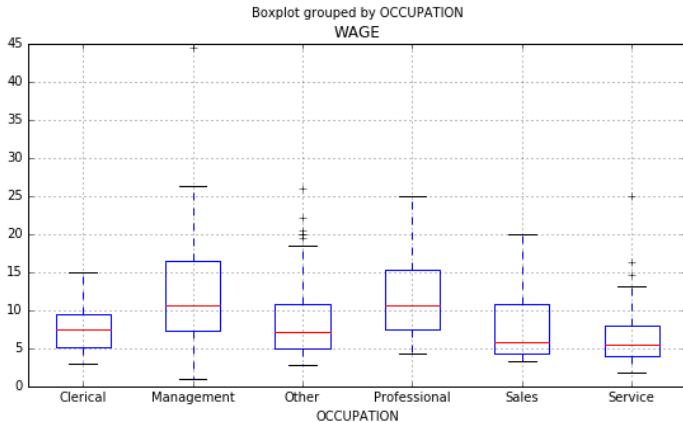
Seaborn

- Seaborn is a visualization module that builds on matplotlib
  - So what we know about matplotlib Axes and rcParams still applies
- Provides a lot of very useful plots and options

Seaborn boxplot

In [5]:
```python
import seaborn as sns
sns.boxplot(x='OCCUPATION', y='WAGE', data=df)
```
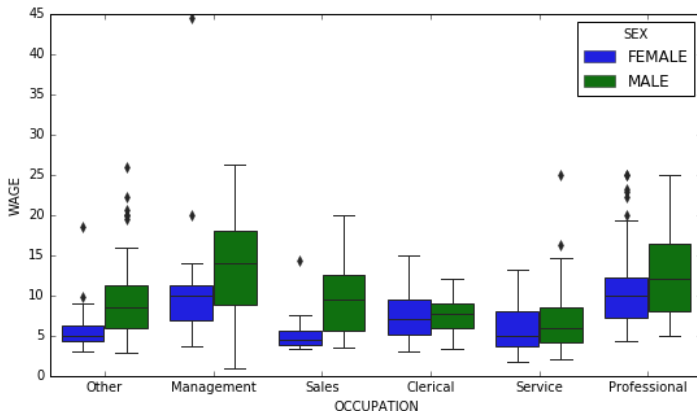
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7faf934e1a10>

Seaborn boxplot hue

```
In [6]: sns.boxplot(x='OCCUPATION', y='WAGE', hue='SEX', data=df)

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7faf8b018410>
```
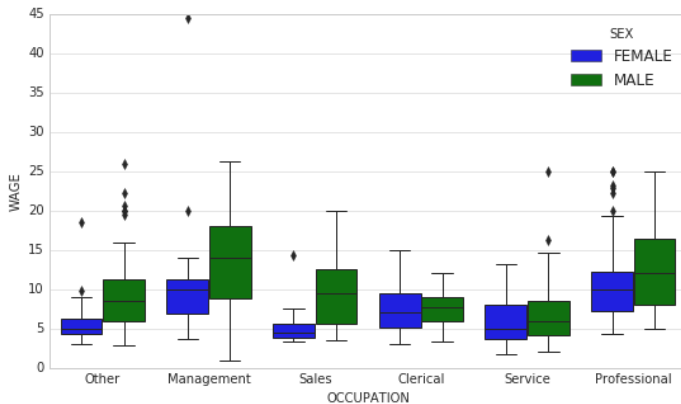
Seaborn aesthetics

```python
sns.set_style('whitegrid')
sns.boxplot(x='OCCUPATION', y='WAGE', hue='SEX', data=df)
```
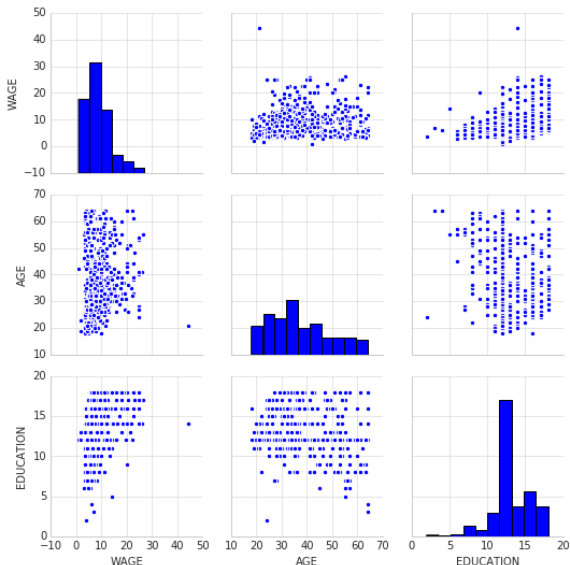
`<matplotlib.axes._subplots.AxesSubplot at 0x7faf8a1fdd10>`

Seaborn `pairplot()`

```
In [12]: sns.pairplot(df, vars=['WAGE', 'AGE', 'EDUCATION'])

Out[12]: <seaborn.axisgrid.PairGrid at 0x7faf8a97cf50>
```
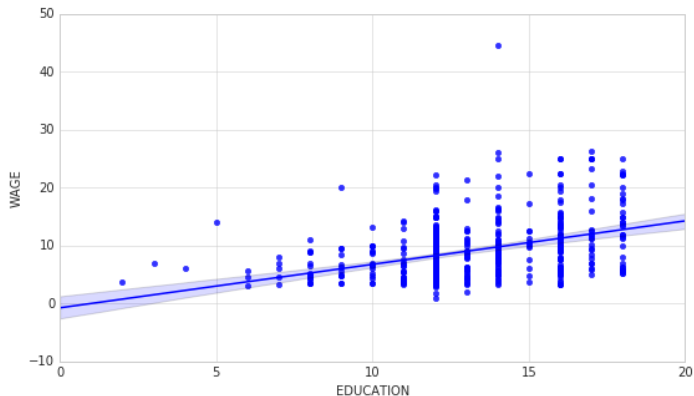
Seaborn regplot()

```
In [21]: sns.regplot('EDUCATION', 'WAGE', df)
```

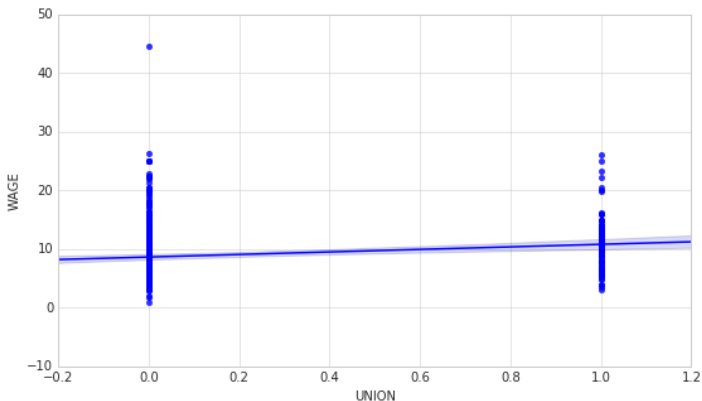Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7faf82ad7f90>

Another `regplot()`

Jittering

```
In [26]: sns.regplot('UNION', 'WAGE', df, x_jitter=.1)

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d02a13250>
```
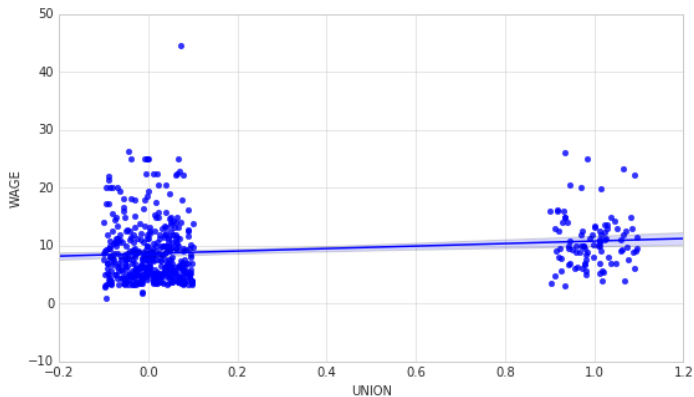
pairplot kind='reg'

In [13]: `sns.pairplot(df, vars=['WAGE', 'AGE', 'EDUCATION'], kind='reg')`

Out[13]: `<seaborn.axisgrid.PairGrid at 0x7faf8a97cc10>`
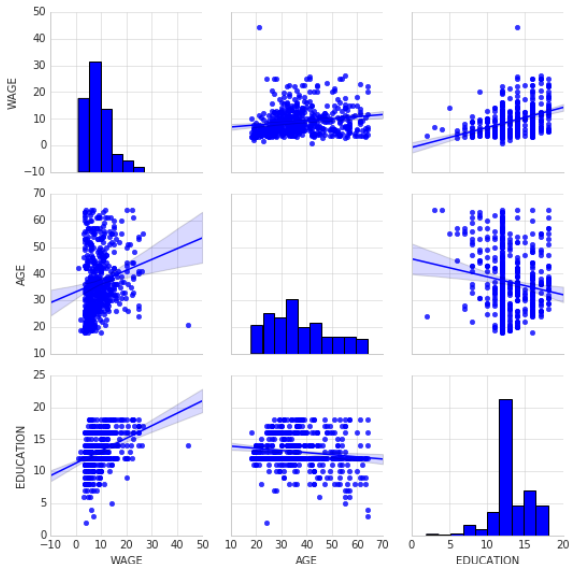
pairplot hue

`sns.pairplot(df, vars=['WAGE', 'AGE', 'EDUCATION'], kind='reg', hue='SEX')`

`<seaborn.axisgrid.PairGrid at 0x7faf8367b910>`
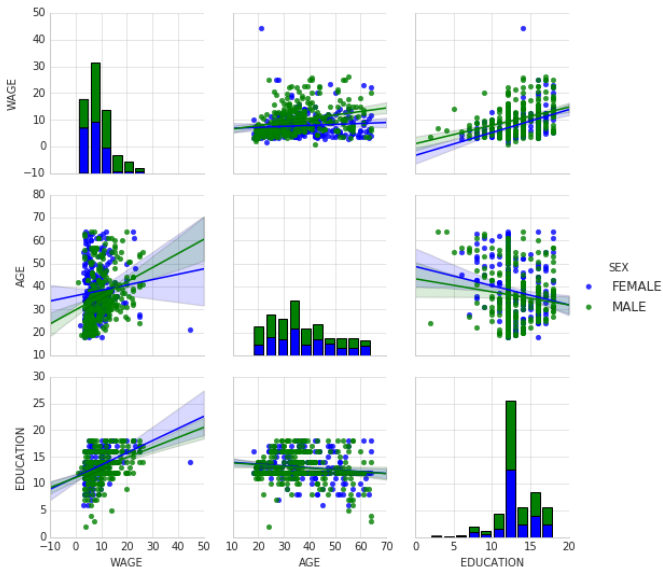
Regression in Python

Statsmodels

- Statsmodels is an econometrics/statistics library for python.
- It provides OLS, logistic and other regression functions.
- There are other modules that can perform regression (e.g. `scipy` and `sklearn`)
  - But statsmodels has the best output

X,y

One way to run a regression with statsmodels is to provide a
exogenous design matrix and an endogenous outcome vector:

```python
In [26]: from statsmodels.regression.linear_model import OLS
```

```python
In [28]: X = df[['AGE', 'EDUCATION']]
         y = df['WAGE']
```

statsmodels regression

```
In [32]:  res = OLS(y, X)
          type(res)

Out[32]:  statsmodels.regression.linear_model.OLS


In [34]:  fit = res.fit()
          type(fit)

Out[34]:  statsmodels.regression.linear_model.RegressionResultsWrapper
```

Regression `summary()`

In [36]: `fit.summary()`

Out[36]:

OLS Regression Results

Dep. Variable:	WAGE	R-squared:	0.798
Model:	OLS	Adj. R-squared:	0.797
Method:	Least Squares	F-statistic:	1051.
Date:	Thu, 17 May 2018	Prob (F-statistic):	1.73e-185
Time:	00:05:00	Log-Likelihood:	-1580.3
No. Observations:	534	AIC:	3165.
Df Residuals:	532	BIC:	3173.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
AGE	0.0595	0.014	4.309	0.000	0.032 0.087
EDUCATION	0.5351	0.040	13.311	0.000	0.456 0.614

Omnibus:	230.878	Durbin-Watson:	1.789
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1396.245
Skew:	1.805	Prob(JB):	6.45e-304
Kurtosis:	10.052	Cond. No.	8.47

Add a constant

You could add a constant like this and then rerun the regression:

```
In [52]: X['Intercept'] = 1
         fit = OLS(y,X).fit()
         fit.summary()
```

Out[52]:

OLS Regression Results

Dep. Variable:	WAGE	R-squared:	0.202
Model:	OLS	Adj. R-squared:	0.199
Method:	Least Squares	F-statistic:	67.21
Date:	Thu, 17 May 2018	Prob (F-statistic):	9.57e-27
Time:	00:08:41	Log-Likelihood:	-1571.1
No. Observations:	534	AIC:	3148.
Df Residuals:	531	BIC:	3161.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
AGE	0.1050	0.017	6.112	0.000	0.071 0.139
EDUCATION	0.8211	0.077	10.657	0.000	0.670 0.972
Intercept	-5.5342	1.279	-4.326	0.000	-8.047 -3.021

Fitted values

```
In [53]: fit.fittedvalues
```

```
Out[53]: 0      4.710601
         1      7.842321
         2      6.314583
         3      6.629667
         4      7.995029
         5      8.080941
         6      7.193038
         7      7.154806
         8     11.069402
         9      7.154806
         10     7.995029
         11     8.205085
         12     5.340768
         13     6.581986
         14     6.476958
         15    10.095586
         16     6.200107
         17     8.940280
         18     6.963866
         19     9.675475
```

Residuals

```
In [54]: fit.resid

Out[54]: 0        0.389399
         1       -2.892321
         2        0.355417
         3       -2.629667
         4       -0.495029
         5        4.989059
         6       -2.743038
         7       12.315194
         8        2.210598
         9        1.595194
         10       3.354971
         11       3.294915
         12       1.159232
         13      -0.331986
         14      13.503042
         15      -2.795586
         16       1.799893
         17      13.259720
```

Coefficients

```
In [55]: fit.params

Out[55]: AGE            0.105028
         EDUCATION      0.821107
         Intercept     -5.534232
         dtype: float64
```

Formulas

It's more convenient to run a regression using formulas:

```
In [56]: import statsmodels.formula.api as smf
```

```
In [57]: fit = smf.ols('WAGE ~ AGE + EDUCATION', df).fit()
         fit.summary()
```

Out[57]:

OLS Regression Results

Dep. Variable:	WAGE	R-squared:	0.202
Model:	OLS	Adj. R-squared:	0.199
Method:	Least Squares	F-statistic:	67.21
Date:	Thu, 17 May 2018	Prob (F-statistic):	9.57e-27
Time:	00:09:02	Log-Likelihood:	-1571.1
No. Observations:	534	AIC:	3148.
Df Residuals:	531	BIC:	3161.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]

Formula anatomy

```
WAGE ~ AGE + EDUCATION
```

- ▶ The variable on the left of the ~ is the outcome
- ▶ The variables on the right are separated by plus signs

Formula intercepts

- Formulas include an intercept term on the right-hand side by default. To remove it use

```
WAGE ~ EDUCATION + AGE - 1
```

- You can also specify the intercept explicitly (same as not specifying it):

```
WAGE ~ EDUCATION + AGE + 1
```

Formulas with `numpy` functions

With formulas you can apply functions to the covariates or
dependent variable:

```
np.log(WAGE) ~ EDUCATION + AGE + 1
```

Formula interactions

Formulas allow you to play around with different specifications easily.

- To add the interaction of two variables use `a:b`
- To add the variables and their interaction use `a*b`
    - So `a*b` is equivalent to `a + b + a*b`

Formulas with categorical variables

By default, variables with string values become dummy variables (fixed effects) in the regression:

```
In [59]: fit = smf.ols('WAGE ~ AGE + EDUCATION + OCCUPATION', df).fit()
         fit.summary()
```

Out[59]:

OLS Regression Results

Dep. Variable:	WAGE	**R-squared:**	0.272
Model:	OLS	**Adj. R-squared:**	0.262
Method:	Least Squares	**F-statistic:**	28.09
Date:	Thu, 17 May 2018	**Prob (F-statistic):**	7.28e-33
Time:	00:14:41	**Log-Likelihood:**	-1546.5
No. Observations:	534	**AIC:**	3109.
Df Residuals:	526	**BIC:**	3143.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	-4.6162	1.566	-2.949	0.003	-7.692 -1.541
OCCUPATION[T.Management]	3.9994	0.764	5.235	0.000	2.499 5.500
OCCUPATION[T.Other]	2.0634	0.589	3.503	0.000	0.906 3.221

Formula C()

For numeric columns to be interpreted as categories use `C()`:

```
In [68]: fit = smf.ols('WAGE ~ AGE + C(EDUCATION) + OCCUPATION + SECTOR + UNION', df).fit()
         fit.summary()
```

Out[68]:

OLS Regression Results

Dep. Variable:	WAGE	R-squared:	0.320
Model:	OLS	Adj. R-squared:	0.286
Method:	Least Squares	F-statistic:	9.560
Date:	Thu, 17 May 2018	Prob (F-statistic):	2.88e-29
Time:	00:18:46	Log-Likelihood:	-1528.4
No. Observations:	534	AIC:	3109.
Df Residuals:	508	BIC:	3220.
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	2.6712	4.533	0.589	0.556	-6.235 11.578
C(EDUCATION)[T.3]	-2.9981	6.223	-0.482	0.630	-15.225 9.229
C(EDUCATION)[T.4]	-1.2337	6.180	-0.200	0.842	-13.375 10.908
C(EDUCATION)[T.5]	5.2968	6.252	0.847	0.397	-6.985 17.579
C(EDUCATION)[T.6]	-5.1672	5.088	-1.015	0.310	-15.164 4.830
C(EDUCATION)[T.7]	-2.1600	4.800	-0.450	0.653	-11.589 7.269

Other models

- Logit
- Weighted least squares

Statistical testing

SciPy

SciPy (Scientific Python) is a library of scientific tools for python.
It includes, among many other things, a submodule `scipy.stats`
with a many statistical test functions.

Log wages by sex

```
In [40]:  male_wages = df['WAGE'][df.SEX == 'MALE']
          female_wages = df['WAGE'][df.SEX=='FEMALE']
```

```
In [46]:  np.log(male_wages).hist(alpha=.5)
          np.log(female_wages).hist(alpha=.5)
```

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9d02327f90>
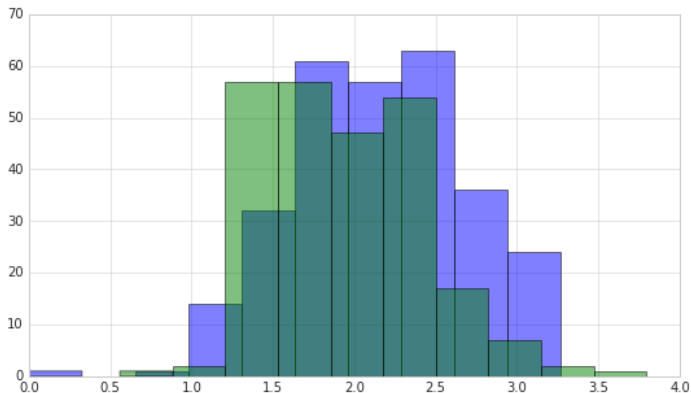
t-test

```
In [47]:  from scipy.stats import ttest_ind
          ttest_ind(np.log(male_wages), np.log(female_wages))

Out[47]:  Ttest_indResult(statistic=5.1658066558780682, pvalue=3.3904092752343325e-07)
```

Other stats

- 1 sample t-test
- 1- and 2-sample z-test
- Spearman's rank correlation