* **Import your packages**

import wooldridge as woo

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**import statsmodels.formula.api as smf**

**from scipy.optimize import minimize**

* **Graphs**

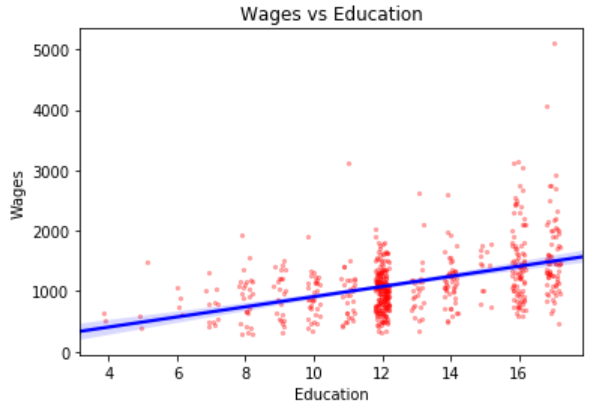
# Plot ***wages as a function of education***

#use line\_kws= {'color':'blue'} to change color of the line

# scatter\_kws={'color':'red', 's':5} to change color and size of dots

**sns.regplot**(x="education", y="wage", data=PSID1982,line\_kws= {'color':'blue'}, marker='o', scatter\_kws={'color':'red', 's':5, 'alpha':0.3}, x\_jitter = 0.2). set(title= 'Wages vs Education',

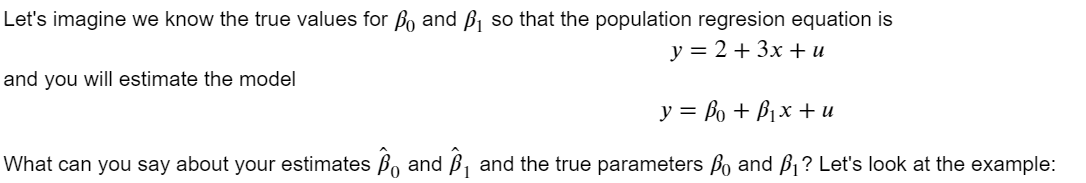
xlabel="Education", ylabel = "Wages");

****

**#*Plot wages as a function of education, separated by gender***

sns.lmplot(x="education", y="wage", **col="gender"**, scatter\_kws={'color':'red','alpha':0.3},

sharex=False, data=PSID1982).add\_legend();

****

# Set the seed to generate reproducible results

np.random.seed(123)

# generates a random from the normal distribution between 0 and 10

*x* = np.random.normal(0, 10, 1000)

# generates u a random sample of 500 from the normal distribution

*u* = np.random.normal(0, 36, 1000)

# You know the actual function that relates x to y and you generate y

*y* = 2+(3\*x)+u

**# Put all this in a dataframe/dictionary**

*data* = {"x": x, "u": u, "y": y}

**# Put all vectors into a data frame to work with it later**

*datas* = **pd.DataFrame**(data)

* **estimate a model:**

reg = **smf.ols(formula='y ~ x', data=datas)**

results = **reg.fit()**

b = results**.params**

print(f'b: \n{b}\n')

datas['yhat'] = results

* **Graph the results; compare the regression model with true values**

plt.rcParams['figure.figsize'] = [12, 8]

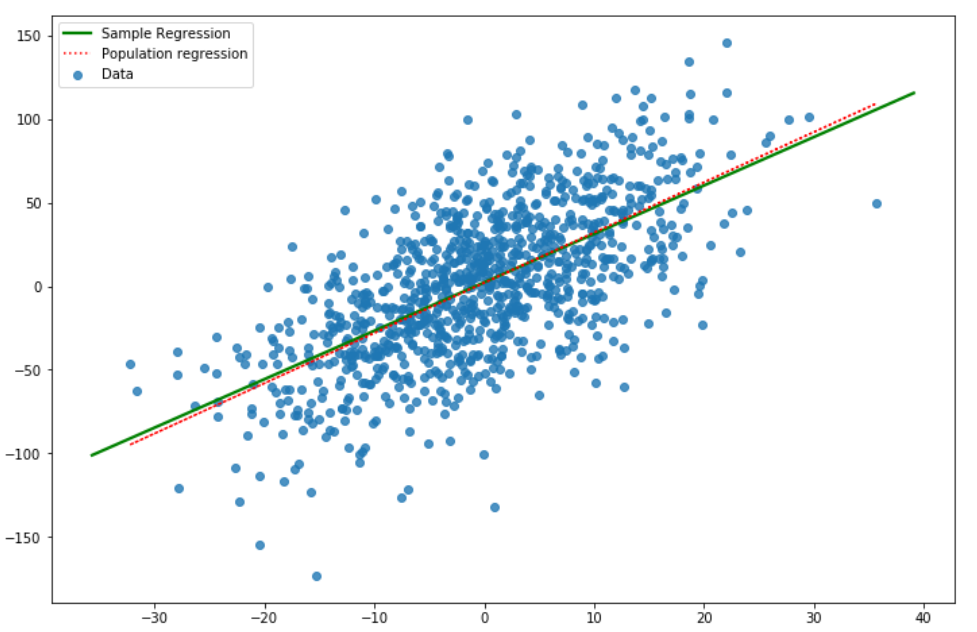
#Regression Model

**sns.regplot**(*"x"*,*"y"*, data=datas, line\_kws={'color':'green'}, ci=None);

#True y

**plt.plot**(*datas["x"]*, *2+3\*datas["x"]*, ':r');

**plt.legend**(["Sample Regression",'Population regression',"Data"],loc='upper left');



* The **smf.ols()** command

Python has a module to make your life easier to allow you to do more sophisticated regressions. The statsmodels module. This command will be the workforce of our class during the semester; it provides you all the information you need for your regression results. Today we will only see how it estimates the parameters. We will learn all about this command in future classes.

Code to estimate the equation: 1613967416(1)

# estimate log-level model:

reg = **smf.ols**(formula = 'y ~ x', data = datas) -> data is a data frame

results = **reg.fit()**

# print results using summary:

print(f'results.summary(): \n{results.summary()}\n')

# **print regression table**:

table = pd.DataFrame({'b': round(results.params, 3),

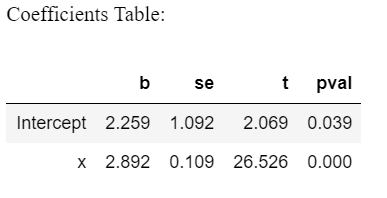
'se': round(results.bse, 3),

't': round(results.tvalues, 3),

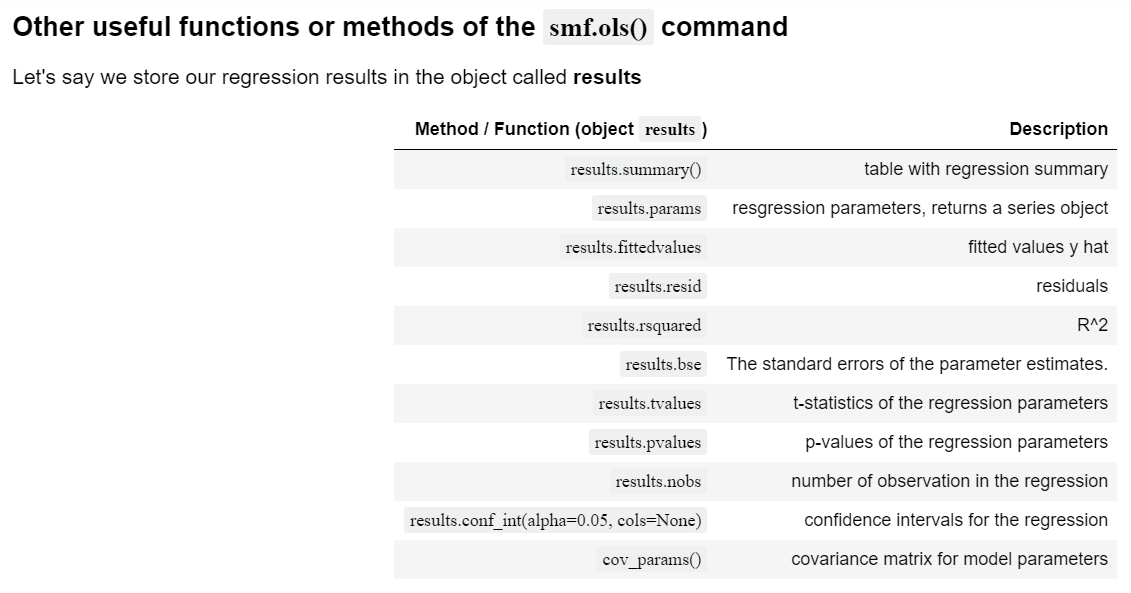
'pval': round(results.pvalues, 3)})

print(f'Coefficients Table:')

display(table)



* Functions Associated With ols() command



#Find regression line

reg = smf.ols(formula='y ~ x', data=mydata)

results = reg.fit()

results.summary()

#look at parameters

results.**params**

#confidence interval

results.**conf\_int**(**alpha=0.05**, cols=None)

#Show average of predicted values

print('Model1 \n Average predicted y is mean of yhat=', round(*np.mean*(**results.fittedvalues**),5))

* **Show** **predicted values of y given specific values of x**

# ***If you want to predict for the mean of dependent variables***

means = mydata[['x']].mean()

predict1 = results**.predict(**means**)[0]**

print('Model1 \n Predicted y given a value of x \n yhat given x=', round( predict1,5))

# ***If you want to predict given x = 0.5***

values = {'x': [0.5]}

results**.predict(**values**)[0]**

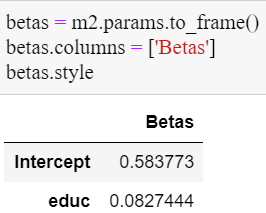
#Call the dataset wage1 and save as wage1

wage1 = woo.dataWoo('wage1')

m1 = smf.ols(formula='wage ~ educ', data=wage1)

m1= m1.fit()

#Show results in table

****

#*Calculate average of the dependent variable y from model 1 and the average for the fitted values*

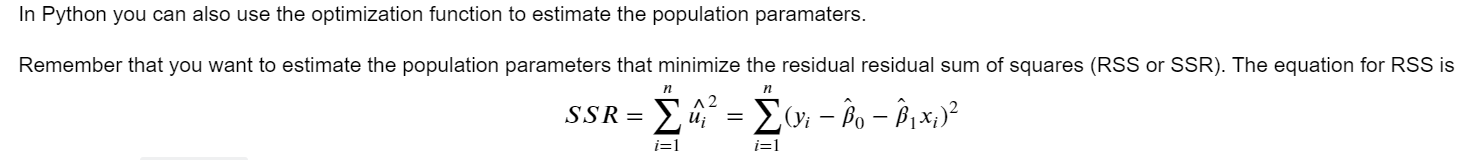
print('Model1 \nMean of y=', round(*np.mean*(**wage1['wage']**),3), 'mean of yhat=', round*(np.mean*(**m1.fittedvalues**),3))

#*Show that* ***mean of residuals*** *is equal to zero*

u\_hat = m2**.resid**

print('Mean u\_hat =', round(np.mean(u\_hat),3))

* Using the **minimize()** function **from scipy.optimize**

****

#Create initial values for the Beta parameters

b = np.array([0.5,0.5])

#create a column of constants

k= np.ones(len(x))

# Create function for Sum of square residuals (SSR)

def sse(b,y,x,k):

return np.sum((y-b[0]\*k-b[1]\*x)\*\*2)

#Use minimize from scipy.optimize to find the parameters beta that minimize the SSR

resa = minimize(fun = sse, args = (y, x, k), x0 = b)

