from scipy.stats import norm

from scipy.stats import t

import scipy.stats as stats

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import wooldridge as woo

import statsmodels.formula.api as smf

from stargazer.stargazer import Stargazer

from IPython.core.display import HTML

* **CV for alpha=5% and 1% using the t distribution with 137 d.f.**:

alpha = np.array([0.05, 0.01])

cv\_t = stats.t.ppf(1 - alpha / 2, 137)

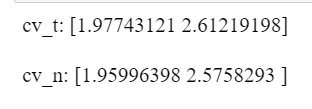
print(f'cv\_t: {cv\_t}\n')

* CV for alpha=5% and 1% using the normal approximation:

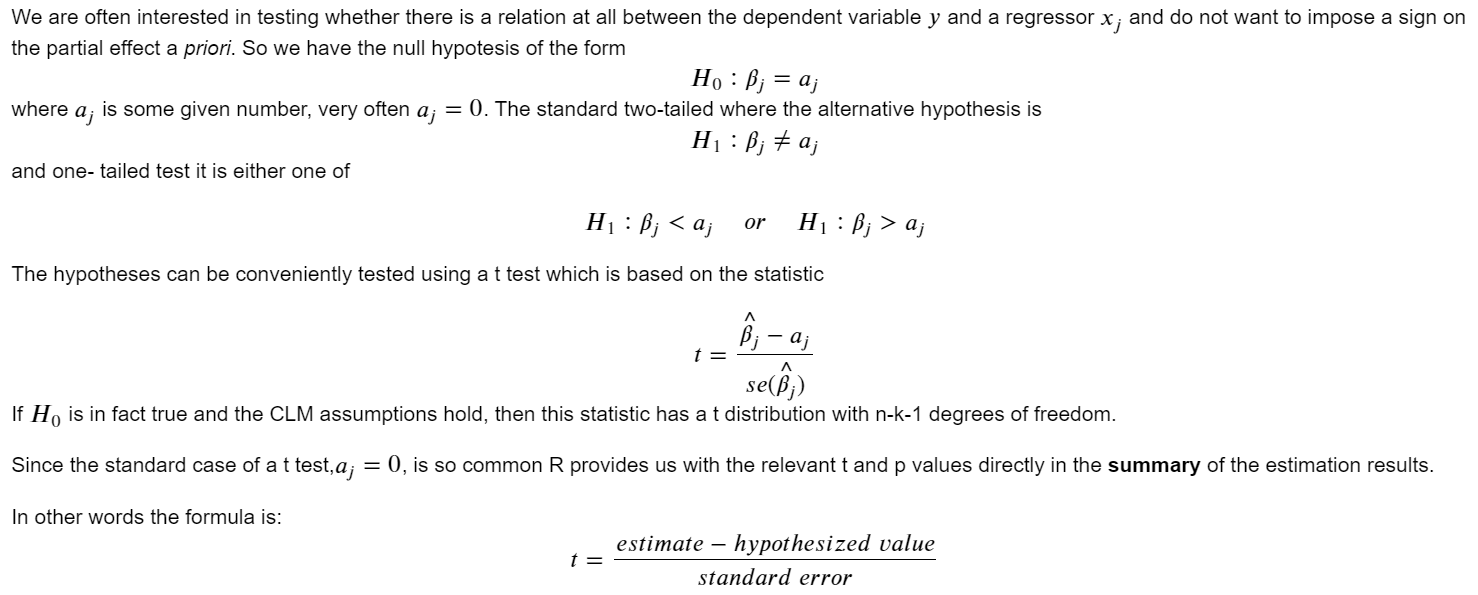
cv\_n = stats.norm.ppf(1 - alpha / 2)

print(f'cv\_n: {cv\_n}\n')

*Output:*



*Recap:*



* **t-test** step-by-step

gpa1 = woo.dataWoo('gpa1')

# *store results*

reg = smf.ols('colGPA ~ hsGPA +ACT +skipped', data=gpa1)

results =reg.fit()

# manually confirm the formulas, i.e. extract coefficients and SE:

**b = results.params**

**se = results.bse**

# *reproduce t statistic*

*tstat = b/se*

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

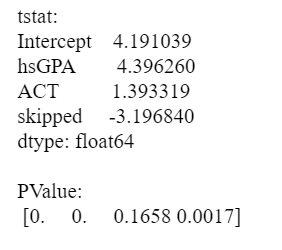
**df = results.nobs - 3-1**

# *reproduce p value*

*pval* = 2\* stats.t.cdf(-abs(tstat), df)

print(f'PValue: \n {np.around(pval,4)}\n')

*Output:*



* **t-test in the regression results**

**results.summary()**

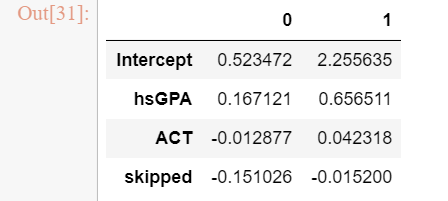
# shows the 5% confidence interval

****

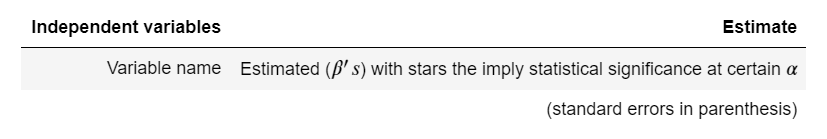
**-Interpretation:** the critical value for alpha=0.05 was 1.97 the t-stats for High School GPA and skipped are bigger than this number. Showing the H0: Bj = 0is rejected in favor of Ha: Bj 0. Thus, the betas for High School GPA and skipped are statistically significant; while the beta for ACT is not.

**# 99% CI: see if 0 is in the confidence interval. If in, fail to reject null -> don’t use this coefficient**

results.**conf\_int**(alpha=0.01, cols=None)

****

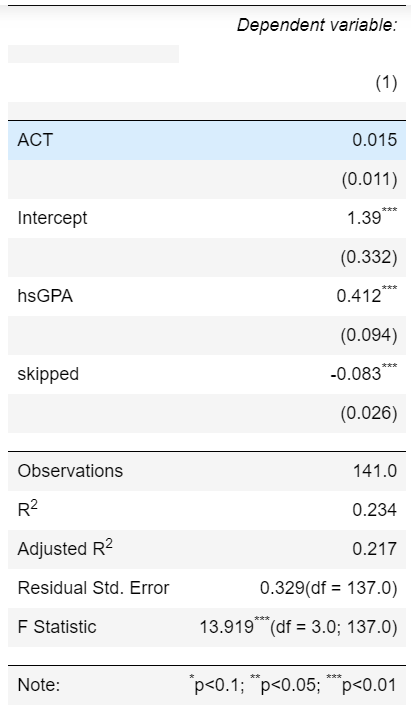
* **Stargazer provides a table with one column per model**

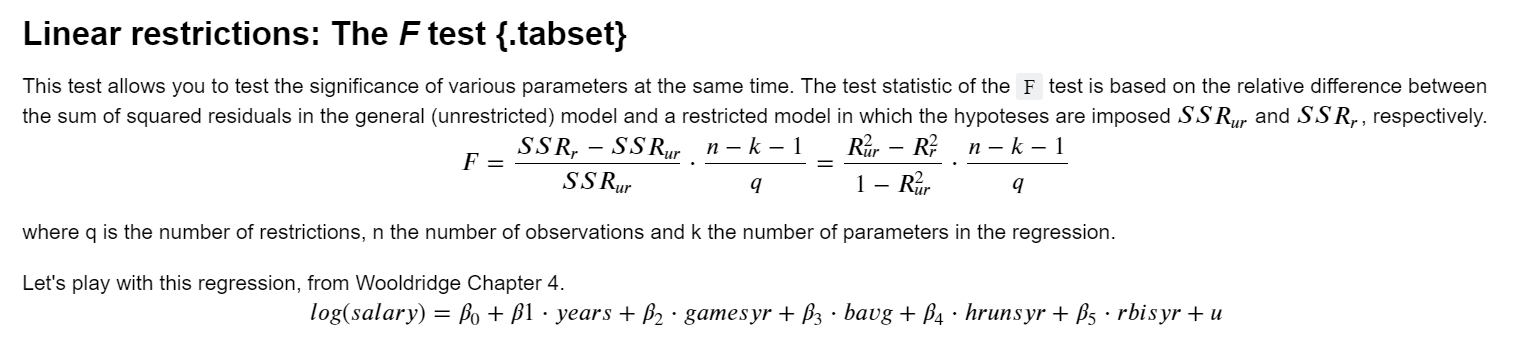
****

stargazer = **Stargazer([results])**

HTML(stargazer.render\_html())

# **Three-star indicates statistical significance (we want to use this coefficient) because the null hypothesis is rejected**.

****

****

* **F test using f\_test**

# *Import dataset mlb1*

mlb1 = woo.dataWoo('mlb1')

# ***OLS regression***

reg = smf.ols(

formula='np.log(salary) ~ years + gamesyr + bavg + hrunsyr + rbisyr',

data=mlb1)

results = reg.fit()

# ***automated F test***

**hypotheses = ['bavg = 0', 'hrunsyr = 0', 'rbisyr = 0']**

ftest = results**.f\_test(hypotheses)**

fstat = ftest**.statistic[0][0]**

fpval = ftest**.pvalue**

# *Beacuse fstat comes form an array we need .around() to round it*

print(f'Fstat: {np.around(fstat, 3)}\n')

print(f'Fpval: {np.around(fpval,3)}\n')

* **F test using compare\_f\_test()**

*#You can just use .compare\_f\_test() to compare both models. Run both \*unrestricted\* and \*restricted\* and apply the method:* ***unrestricted.compare\_f\_test(restricted)****.*

*#The results provide an array with the fstat, p-value and degrees of freedom.*

**fit\_ur.compare\_f\_test(fit\_r)**

* Testing of other hypothesis

#We can perform a more complicated hypothesis like that there is a relation between two variables: beta\_i =c\*beta\_i where c is a constant.

# automated F test:

**hypotheses = ['bavg = 0', 'hrunsyr = 2\*rbisyr']**

ftest = results.f\_test(hypotheses)

fstat = ftest.statistic[0][0]

fpval = ftest.pvalue

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

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

* **F test step-by-step**

#*Import dataset mlb1*

mlb1 = woo.dataWoo('mlb1')

#***Get the number of observations in the dataset using .shape[0]***

n = mlb1.**shape[0]**

#***R2 of unrestricted OLS regression***

reg\_ur = smf.ols(formula='np.log(salary) ~ years + gamesyr + bavg + hrunsyr + rbisyr',data=mlb1)

fit\_ur = reg\_ur.fit()

r2\_ur = fit\_ur.rsquared

print(f' R2 of unrestricted OLS regression: {r2\_ur}\n')

#***R2 of restricted OLS regression (here we do not include the variables bavg , hrunsyr , rbisyr)***

reg\_r = smf.ols(formula='np.log(salary) ~ years + gamesyr', data=mlb1)

fit\_r = reg\_r.fit()

r2\_r = fit\_r.rsquared

print(f' R2 of restricted OLS regression:: {r2\_r}\n')

#***Calculate the F statistic***

fstat = (r2\_ur - r2\_r) / (1 - r2\_ur) \* (n - 6) / 3

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

#***CV for alpha=1% using the F distribution with 3 and 347 d.f.***

cv = stats.f.ppf(1 - 0.01, 3, 347)

print(f'Critical value at 1% with 3 and 347 df= {cv}')

#***p value = 1-cdf of the appropriate F distribution***

fpval = 1 - stats.f.cdf(fstat, 3, 347)

print(f'fpval: {round(fpval,4)}\n')