NK_SS_decomp_sim

August 30, 2024

```
[ ]: #The notebook was getting too long to scroll through lol

# show_plots = True
show_plots = False
# show_prints = True
show_prints = False
```

```
[ ]: import numpy as np
import pandas as pd
from plotnine import *

from sklearn.linear_model import LinearRegression
from scipy.stats import f

# Set the seed for reproducibility
np.random.seed(42)
```

0.1 Exercises:

1. Modify the data generation function to take the slope, intercept, and sample size as arguments.
2. Modify the SS decomposition function to conduct the F-test for testing whether the true slope is zero or not.

```
[ ]: def generate_data(sigma=2, beta0=3, beta1=3, sample_size=10):

    #Array w/ 1 column, the -1 means 'infer the # of rows'
    X = np.linspace(start=1, stop=sample_size, num=sample_size).reshape(-1, 1)

    # Generate epsilon as a 1D array of N(0, sigma^2) RVs
    epsilon = sigma * np.random.randn(sample_size)

    # Generate y using SLR model
    y = beta0 + beta1 * X.flatten() + epsilon

    return X, y
```

```
[ ]: def ss_decomp(X, y):
```

```python
    # Fit a linear regression model
    model = LinearRegression()
    model.fit(X, y)

    # Make predictions
    y_pred = model.predict(X)

    # Calculate the mean of y
    y_mean = np.mean(y)

    #calculate SS quantities
    SST = np.sum((y - y_mean) ** 2).round(4)
    SSR = np.sum((y_pred - y_mean) ** 2).round(4)
    SSE = np.sum((y - y_pred) ** 2).round(4)

    if show_prints:
        # Output the sum of squares decomposition
        print(f"SST (Total Sum of Squares): {SST}")
        print(f"SSR (Regression Sum of Squares): {SSR}")
        print(f"SSE (Error Sum of Squares): {SSE}")
        print(f"SST = SSR + SSE: {np.isclose(SST, SSR + SSE)}")
        print(f"Coefficient of Determination, R^2 is: {np.round(1-SSE/SST, 4)}")

    #F Test
    n = len(y)
    MSR = SSR / 1
    MSE = SSE / (n - 2)
    F_stat = MSR / MSE

    #make a ggplot
    df = pd.DataFrame({
        'X': X.flatten(),
        'y': y,
        'predicted': y_pred,
        'y_mean':  y_mean
    })

    gg1 = (
        ggplot(df, aes(x = 'X', y = 'y')) +
        geom_point() +
        #geom_smooth(method = "lm", formula = "y ~ x", se = False) +
        geom_hline(yintercept=y_mean, linetype='dashed') +
        geom_segment(aes(xend = 'X', yend = 'y_mean'), color = "black") + #SST␣
↪components
        ggtitle(f"R^2 is {np.round(1-SSE/SST, 4)}")
        )
```

```
    gg2 = (
        ggplot(df, aes(x = 'X', y = 'y')) +
        geom_point() +
        geom_smooth(method = "lm", formula = "y ~ x", se = False) +
        geom_hline(yintercept=y_mean, linetype='dashed') +
        geom_segment(aes(xend = 'X', y = 'predicted', yend = 'y_mean'), color =␣
↪"blue", linetype = "dashed") + #SSR components
        geom_segment(aes(xend = 'X',  yend = 'predicted'), color = "red",␣
↪linetype = "dashed") + #SSE components
        ggtitle(f"R^2 is {np.round(1-SSE/SST, 4)}")
        )

    return gg1, gg2, F_stat
```

[ ]:
```
def hypothesis_test(F_stat, n, alpha=0.05):

    # Degrees of freedom
    dfn = 1
    dfd = n - 2
    # Calculate the p-value using the cumulative distribution function (CDF) of␣
↪the F-distribution
    p_value = 1 - f.cdf(F_stat, dfn, dfd)

    # Decision rule

    if p_value < alpha:
        decision = f"Reject the null hypothesis at alpha = {alpha}. There is␣
↪evidence of a linear relationship."
    else:
        decision = f"Fail to reject the null hypothesis at alpha = {alpha}. No␣
↪evidence of a linear relationship."

    if show_prints:
        print(f"F-statistic: {F_stat}")
        print(f"p-value: {p_value}")
        print(decision)

    return p_value, decision
```

3. Holding the intercept, sample size, and error variance constant, compare how different values of the slope affect:

    a. the SS decomposition,

    b. the plots,

    c. the coefficient of determination, and

d. the hypothesis test results.
   What do you find?

As the slope increases, the model generally explains a larger portion of the variance in resulting in higher SSR, higher r2, and increased F-statistic. The hypothesis test becomes more likely to reject the null hypothesis at higher slopes and more likely to show evidence of a linear relationship.

When the slope is small, the effect of the random error is more pronounced because the changes in y due to X are subtle, making it harder for the model to distinguish the signal (the linear relationship) from the noise (random error).

[ ]:
```
alpha = 0.05
sigma = 5
beta1 = 3
beta0 = 3
n = 10
for i in range(1, 12):
    beta1 = i/2 #iterate slope from 0.5 to range-1
    X, y = generate_data(sigma = sigma, beta0=beta0, beta1=beta1, sample_size=n)
    if show_prints:
        print(f"alpha: {alpha}")
        print(f"sigma: {sigma}")
        print(f"Slope: {beta1}")
        print(f"Intercept: {beta0}")
        print(f"Sample Size: {n}")
    gg1, gg2, F_stat = ss_decomp(X, y)
    hypothesis_test(F_stat=F_stat, n=n, alpha=0.05)

    if show_plots:
        gg1.show()
        gg2.show()

if show_prints:
    print("\n ----------------- \n")
```

4. Holding the slope, sample size, and error variance constant, compare how different values of the intercept affect:

   a. the SS decomposition,

   b. the plots,

   c. the coefficient of determination, and

   d. the hypothesis test results.
      What do you find?

Increasing the intercept also leads to a better-fitting model. This is reflected in higher SSR, R2 and F-statistic values, and lower SSE. The hypothesis test becomes more likely to reject the null hypothesis, indicating stronger evidence of a linear relationship.

Since the baseline level of y increases,

[ ]:
```
alpha = 0.05
sigma = 5
beta1 = 3
beta0 = 3
n = 10
for i in range(1, 12):
    beta0 = i/2 #iterate intercept from 0.5 to range-1
    X, y = generate_data(sigma = sigma, beta0=beta0, beta1=beta1, sample_size=n)
    if show_prints:
        print(f"alpha: {alpha}")
        print(f"sigma: {sigma}")
        print(f"Slope: {beta1}")
        print(f"Intercept: {beta0}")
        print(f"Sample Size: {n}")
    gg1, gg2, F_stat = ss_decomp(X, y)
    hypothesis_test(F_stat=F_stat, n=n, alpha=0.05)

    if show_plots:
        gg1.show()
        gg2.show()
if show_prints:
    print("\n ----------------- \n")
```

5. Holding the intercept, slope, and error variance constant, compare how different values of the sample size affect:

   a. the SS decomposition,

   b. the plots,

   c. the coefficient of determination, and

   d. the hypothesis test results.
      What do you find?

Higher sample size means larger variances average out and the model gets very close with a high R2. The model is highly significant as random noise dissapears.

[ ]:
```
alpha = 0.05
sigma = 5
beta1 = 3
beta0 = 3
n = 10
for i in range(1, 4):
    n = 10**i #iterate sample size from 10 to 1000
    X, y = generate_data(sigma = sigma, beta0=beta0, beta1=beta1, sample_size=n)
```

```
    if show_prints:
        print(f"alpha: {alpha}")
        print(f"sigma: {sigma}")
        print(f"Slope: {beta1}")
        print(f"Intercept: {beta0}")
        print(f"Sample Size: {n}")
    gg1, gg2, F_stat = ss_decomp(X, y)
    hypothesis_test(F_stat=F_stat, n=n, alpha=0.05)

    if show_plots:
        gg1.show()
        gg2.show()
if show_prints:
    print("\n ----------------- \n")
```

6. Write a function which calculates the t-statistic for testing whether the true slope is zero or not. For each simulation you run, calculate both the t-stat and the F-stat. Repeat this \$B=100 times, then plot the ordered pairs $\{(t_b, F_b)\}_{b=1}^B$ as a scatter plot. What relationship do you observe between them? Can you prove this relation?

It looks like there is an exponential relationship between the F and t statistics.

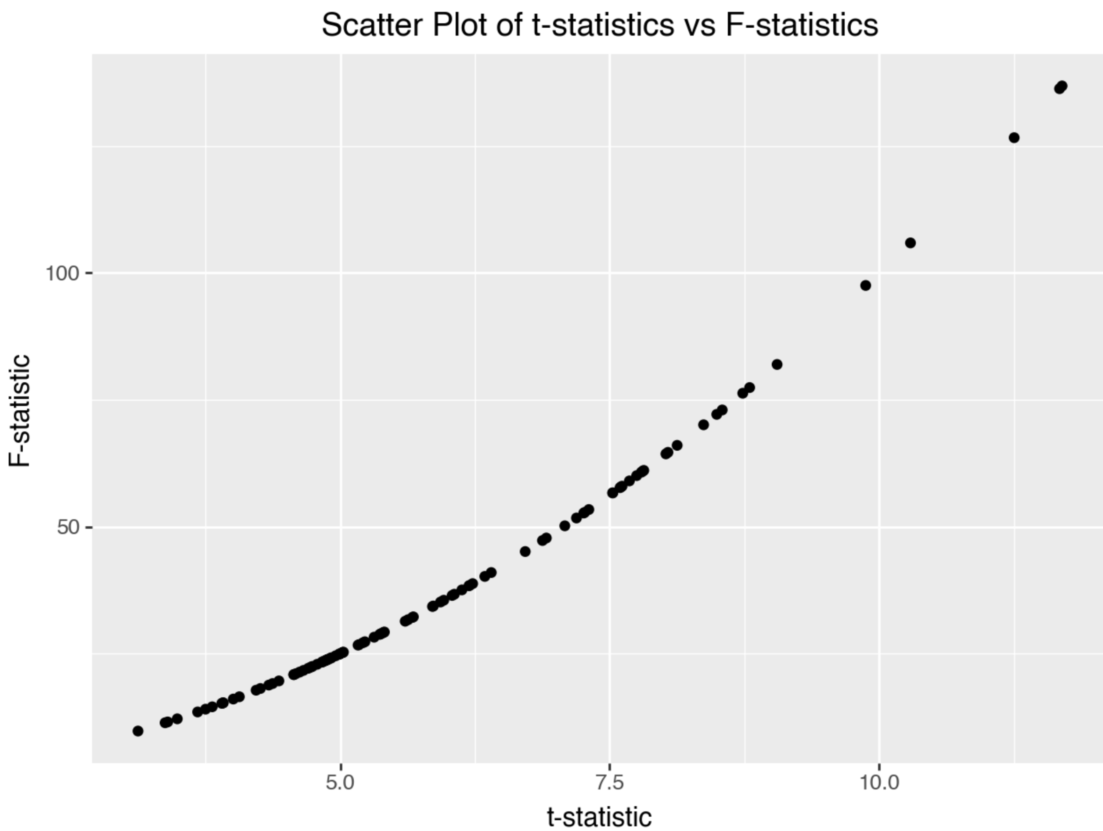
I showed it in code that F is apporox t^2 for each value.

I didn't try to prove it mathematically but with some googling I found - https://stats.stackexchange.com/questions/55236/prove-f-test-is-equal-to-t-test-squared

[ ]:
```
from scipy.stats import t

def t_statistic_slope(X, y):
    model = LinearRegression()
    model.fit(X, y)

    n = len(y)

    y_pred = model.predict(X)

    X_mean = np.mean(X)

    SSX = np.sum((X - X_mean) ** 2).round(4)

    residual_variance = np.sum((y - y_pred) ** 2).round(4) / (n - 2)

    se = np.sqrt(residual_variance / SSX)

    beta1_hat = model.coef_[0]

    t_stat = beta1_hat / se
```

```
    if show_prints:
        print(f"SSX (Sum of Squares of X): {SSX}")
        print(f"t_statistic: {t_stat}")

    df = n - 2

    p_value = 2 * (1 - t.cdf(np.abs(t_stat), df))

    return t_stat, p_value
```

[ ]:
```
alpha = 0.05
sigma = 5
beta1 = 3
beta0 = 3
n = 10

list1 = []
flag1 = True
for i in range(1, 101):
    X, y = generate_data(sigma=sigma, beta0=beta0, beta1=beta1, sample_size=n)
    t_stat, _ = t_statistic_slope(X, y)
    _, _, F_stat = ss_decomp(X, y)
    list1.append({'t_stat': t_stat, 'F_stat': F_stat})
    #Lets show F = t^2
    flag1 = np.isclose(t_stat**2, F_stat)

df2 = pd.DataFrame(list1)

gg3 = (
    ggplot(df2, aes(x='t_stat', y='F_stat')) +
    geom_point() +
    ggtitle('Scatter Plot of t-statistics vs F-statistics') +
    xlab('t-statistic') +
    ylab('F-statistic')
)

gg3.show()

print(f"F = t^2 for all rows?\n{flag1}")
```




```
F = t^2 for all rows?
True
```