Python Results:

1. For example 3, the output shows a data frame with Fertilizer, Row, and Column as categorical variables. The "Yield" values reflect the specified overall effect (7), fertilizer effects (1, 2, 3), row effects (2, 2, 1), and column effects (3, 3, 2), along with normally distributed errors (mean = 3, standard deviation = 2)

3. Simulation 1 (normal errors, no fertilizer effects), ~ 5% of p-values were <= 0.05. This supports the null hypothesis and a uniform distribution. Simulations 2 - 4 (normal errors, non-zero effects) have the p-values cluster down near 0 (high power). Simulations 5 - 7 (exponential errors, non-zero effects) have the p-value distribution deviate from expectations. This means that the histogram shows a higher proportion of significant p-values, indicating a stable ANOVA to non-normal errors in this case.

R Results:

The histograms show the distribution of p-values for the test of significant fertilizer effect in each simulation. The red vertical line represents the typical significance threshold of p = 0.05.

<u>Key observations:</u>

- In Simulation 1, where there is no significance in fertilizer effects, we expect about 5% of p-values to be below 0.05.

- In Simulations 2-7, where there is significance in fertilizer effects, we expect to see more p-values below 0.05, indicating that the test is able to identify these effects.

- Comparing simulations with normal errors (2-4) to those with exponential errors (5-7) shows how different error distributions might affect the test's Behavior.

4. The simulations between R and Python showed minimal differences, mainly in Simulation 5-7, which had non-normal errors:

- Simulations 1-4 (with normal errors) for both R and Python produced similar p-value distributions.
- Simulation 1, which was the null case, ~ 5% of p-values that were <= 0.05 were observed for both languages. This aligns with the expected Type I error rate.
- Simulations 5-7 (with exponential errors):
    - Python kept producing slightly fewer significant p-values than R. For example, Simulation 5 in R had 22% of p-values <= 0.05, but only 17% in Python.
    - The simulation histograms in Python showed more uniform distributions for Simulations 5-7. However, R's histograms were marginally skewed toward lower p-values, so they were more skewed right.

- These differences between R's and Python's simulation distributions must've been caused by the random number generation methods between R and Python. Python's "numpy" and R work differently and so they default to different results in RNG. To be more specific, R uses Type I ANOVA from using "lm()," while Python uses Type II ANOVA from "statsmodels."

5a. I noticed that Python has more uniqueness in its syntax with what we learned in comparison to R. What I mean by this is that with R, the function syntax was by far the most predominantly used, from what I remember at least. I remember learning the syntax for functions, dictionaries, series, data frames, and arrays much more vividly than for R, probably because we finished off learning Python. Coding the problems with R felt smooth and understandable in that the syntax better matched the material, compared to Python, if that makes sense. I wish we got to spend more time with Python in class to make a better judgment on the two programming languages, but I feel like with what I have learned so far with Python, I will grow into it and its simplistic syntax and feel.

b. I prefer R because I noticed that R felt more natural and easier to comprehend with what we learned throughout this semester in comparison to Python. Then again, I feel like we spent more time with R and got to learn it more in-depth, rather than Python. However, I much prefer the layout and structure of Python because of its simplicity compared to R. There isn't a console or environment with Python like there is for R, where numerous sections are opened up at once while you're coding. Another small preference I have, this time for R instead of Python, is setting up the language and saving your files. I found it to be easier and less time-consuming for R than

for Python. With coding and syntax, Python may be simpler and more direct, but R felt more easier to grasp with almost everything being a function and how it's a matter of being consistent with how you set up the syntax. I don't know if Python has this feature as well, but if you never know how to code something in R, you could always write "?..." or "help…" I think it was, in the console, which was incredibly useful.

```
#Example 1: Default Parameters
```{r}
example1 <- simCorn()
example1
```
```

Fertilizer <fctr>	Row <fctr>	Column <fctr>	Yield <dbl>
A	1	1	-0.2814269
B	1	2	0.7170131
C	1	3	-0.7382318
C	2	1	0.1785897
A	2	2	-0.5100990
B	2	3	0.1858384
B	3	1	-0.3464843
C	3	2	-1.9102800
A	3	3	1.6613584

9 rows

#Example 2: Custom Error Distribution

```{r}
example2 <- simCorn(overallEffect = 10, seed = 2123, dist = rgamma,
shape = 2)
example2
```

Fertilizer <fctr>	Row <fctr>	Column <fctr>	Yield <dbl>
A	1	1	14.83727
B	1	2	10.45424
C	1	3	13.13900
C	2	1	10.47095
A	2	2	10.90779
B	2	3	13.44940
B	3	1	10.77832
C	3	2	11.49251
A	3	3	10.62710

9 rows

#Example 3: Custom Parameters

```{r}
mu <- 7
alpha <- c(1, 2, 3)
beta <- c(2, 2, 1)
gamma <- c(3, 3, 2)
example3 <- simCorn(overallEffect = mu,
                    fertilizerEffect = alpha,
                    rowEffect = beta,
                    colEffect = gamma,
                    seed = 29429,
                    dist = rnorm,
                    mean = 3,
                    sd = 2)
example3
```

Fertilizer <fctr>	Row <fctr>	Column <fctr>	Yield <dbl>
A	1	1	19.97551
B	1	2	16.08501
C	1	3	17.82301
C	2	1	15.00537
A	2	2	15.31535
B	2	3	13.89126
B	3	1	15.24435
C	3	2	16.98722
A	3	3	17.10382

9 rows