index

January 18, 2022

1 Effect Size, P-Values and Power - Lab

1.1 Introduction

In this lab, you'll run simulations to continue to investigate the relationship between effect size, p-values, power, and sample size!

1.2 Objectives

You will be able to: * Run a simulation that creates a visualization to demonstrate the interaction between power, sample size, and effect size

1.3 Philosophical review

Remember that the underlying question behind all hypothesis tests is:

> "What is the probability I would see this effect due to random fluctuations if there was actually no effect?"

This is exactly what a p-value represents: the chance that the observed data would satisfy the null hypothesis. As such, if the p-value is sufficiently low, you can declare the results statistically significant and reject the null hypothesis. Recall that this threshold is defined as α, and is also the rate of type I errors. In this lab, you'll investigate the robustness of p-values and their relation with effect-size and sample-size.

1.4 Import starter functions

To start, import the functions stored in the `flatiron_stats.py` file. It contains the stats functions that you previously coded in the last lab: `welch_t(a,b)`, `welch_df(a, b)`, and `p_value(a, b, two_sided=False)`. You'll then use these functions below to further investigate the relationship between p-values and sample size.

[2]:
```
# Your code here; import the contents from flatiron_stats.py
# You may also wish to open up flatiron_stats.py in a text editor to preview␣
↪its contents

from flatiron_stats import *
```

1.5 Generate random samples

Before you start running simulations, it will be useful to have a helper function that will generate random samples. Write such a function below which generates 2 random samples from 2 normal distributions. The function should take 6 input parameters:

- m1 - The underlying population mean for sample 1
- s1 - The underlying population standard deviation for sample 1
- n1 - The sample size for sample 1
- m2 - The underlying population mean for sample 2
- s2 - The underlying population standard deviation for sample 2
- n2 - The sample size for sample 2

[3]:
```
import numpy as np
def generate_samples(m1, s1, n1, m2, s2, n2):
    sample1 = np.random.normal(m1, s1, n1)
    sample2 = np.random.normal(m2, s2, n2)
    # Your code here; have the function create two random samples using the␣
 ↪input parameters
    return sample1, sample2
```

1.6 Run a simulation

For your first simulation, you're going to investigate how the p-value of an experiment relates to sample size when both samples are from identical underlying distributions. To do this, use your `generate_samples()` function along with the `p_value_welch_ttest()` function defined in the `flatiron_stats` file. Use sample sizes from 5 to 750. For each sample size, simulate 100 experiments. For each of these experiments, generate 2 samples of the given sample size. Each sample should have a standard deviation of 1. The first sample should have a mean of 5 and the second should have a mean of 5 plus the effect size, you hope to detect. Calculate the corresponding p-values for a Welch's t-test for each of these sample pairs. Finally, use the p-values to calculate the power of the test. Remember that for all of the simulations where the effect size does not equal zero, the null hypothesis is not true. Store the overall power from the 100 simulations along with the corresponding sample size and effect size. Use varying effect sizes such as [0, 0.01, 0.1, 0.2, 0.5, 1, 2]. You'll then plot power vs sample size for various effect sizes.

[4]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('darkgrid')
```

[18]:
```
# Your code here
sample_size = list(range(5, 750, 5))
n_sim = 100
effect_size = [0, 0.01, 0.1, 0.2, 0.5, 1, 2]
```

```
p_es = {}
alpha = 0.05

for es in effect_size:
    p_s = {}
    for i in sample_size:

        p_val = []

        for j in range(100):

            a, b = generate_samples(5, 1, i, 5+es, 1, i)
            p_val.append(p_value_welch_ttest(a, b, two_sided=False))

        p_s[i] = np.sum(np.array(p_val) < alpha) / 100


    p_es[es] = p_s


# Pseudo code outline
# for effect size:
#     for sample_size:
#         perform 100 simulations
#         calculate power
#         store effect_size, sample_size, power for simulations
```

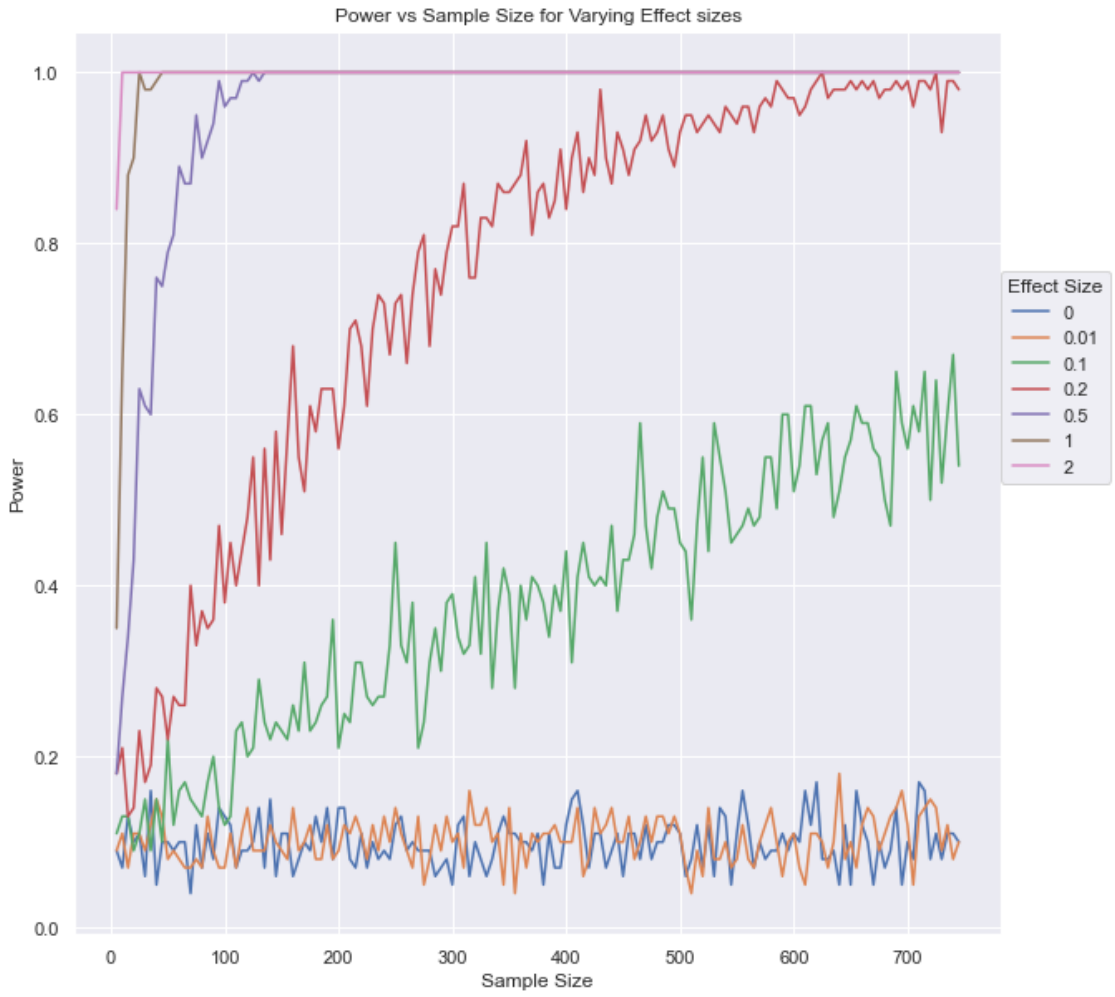
Now that you've simulated the data, go ahead and graph it! Label the x-axis sample size, the y-axis power, and be sure to include a legend for the various effect sizes.

[26]:
```
# Your code here
sns.set(rc={"figure.figsize":(10, 10)})
for item in p_es:
#     sns.lineplot(p_es[item].values(), p_es[item].keys());
    sns.lineplot(p_es[item].keys(), p_es[item].values());
plt.legend(labels = p_es.keys(), title='Effect Size', loc=(1, 0.5));
plt.title('Power vs Sample Size for Varying Effect sizes')
plt.xlabel("Sample Size")
plt.ylabel("Power")
```

```
/opt/anaconda3/envs/learn-env/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/opt/anaconda3/envs/learn-env/lib/python3.8/site-
```

```
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/opt/anaconda3/envs/learn-env/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/opt/anaconda3/envs/learn-env/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/opt/anaconda3/envs/learn-env/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/opt/anaconda3/envs/learn-env/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/opt/anaconda3/envs/learn-env/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[26]:
```
Text(0, 0.5, 'Power')
```



As you can see, it's also typically incredibly difficult (if not impossible) to accurately detect effect sizes below 0.1!

1.7 Summary

This lesson summarizes and further builds upon the ideas that we saw in the previous labs. We learned how p-value can be described as a function of effect size and for a given effect size, the p-value may get lower if we increase the sample size considerably. We also saw how p-value alone can not be used in order to identify some results as truly significant, as this can be achieved when there is not a significant effect size.