index

January 17, 2022

1 Statistical Power - Lab

1.1 Introduction

In this lesson, you'll practice doing a power-analysis during experimental design. As you've seen, power analysis allows you to determine the sample size required to detect an effect of a given size with a given degree of confidence. In other words, it allows you to determine the probability of detecting an effect of a given size with a given level of confidence, under-sample size constraints.

The following four factors have an intimate relationship:

- Sample size
- Effect size
- Significance level = P (Type I error) = probability of finding an effect that is not there
- **Power = 1 - P (Type II error)** = probability of finding an effect that is there

Given any three of these, we can easily determine the fourth.

1.2 Objectives

In this lab you will:

- Describe the impact of sample size and effect size on power
- Perform power calculation using SciPy and Python
- Demonstrate the combined effect of sample size and effect size on statistical power using simulations

1.3 Let's get started!

To start, let's import the necessary libraries required for this simulation:

```
[1]: import numpy as np
     import scipy.stats as stats
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     sns.set_style('darkgrid')
```

1.4 Scenario

A researcher wants to study how daily protein supplementation in the elderly population will affect baseline liver fat. The study budget will allow enrollment of 24 patients. Half will be randomized to a placebo group and half to the protein supplement treatment group and the trial will be carried out over one month. It is desired to see whether the mean change in percentage of liver fat from baseline to the end of the study differs between the two groups in the study.

With this, the researcher writes the null hypothesis:

`There is no difference between experimental and control group mean change in percentage of live`

$$\mu_1 = \mu_2$$

And the alternative Hypothesis:

`There is a difference between experimental and control group mean change in percentage of live`

$$\mu_1 \neq \mu_2$$

The researcher needs to know what power will be obtained under the sample size restrictions to identify a change in mean percent liver fat of 0.17. Based on past results, a common standard deviation of 0.21 will be used for each treatment group in the power analysis.

To determine the practicality of this experimental design, you'll run a power analysis simulation:

[2]:
```
# Number of patients in each group
sample_size = 12

# Control group
control_mean = 0
control_sd = 0.21

# Experimental group
experimental_mean = 0.17
experimental_sd = 0.21

# Set the number of simulations for our test = 1000
n_sim = 1000
```

You can now start running simulations to run an independent t-test with above data and store the calculated p-value in our `p` array. Perform following tasks:

- Initialize a numpy array and fill it with `NaN` values for storing the results (p_value) of the independent t-test

- For a defined number of simulations (i.e., 1000), do the following:
  - Generate a random normal variable with control mean and sd
  - Generate a random normal variable with experimental mean and sd

    – Run and independent t-test using control and experimental data
    – Store the p value for each test

- Calculate the total number and overall proportion of simulations where the null hypothesis is rejected

[3]:
```python
alpha = 0.05

# For reproducibility
np.random.seed(10)

# Initialize array to store results
p = (np.empty(n_sim))
p.fill(np.nan)
p1 = []  # My Way
#  Run a for loop for range of values in n_sim
for i in range(n_sim):
    control = np.random.normal(control_mean, control_sd, sample_size)
    experiment = np.random.normal(experimental_mean, experimental_sd,
                                  sample_size)
    t_test = stats.ttest_ind(control, experiment)
    p[i] = t_test[1]
    p1.append(t_test[1]) # My Way

# number of null hypothesis rejections
num_null_rejects = np.sum(p < alpha)
num_null_rejects1 = np.sum(np.array(p1) < alpha) # My Way

power = num_null_rejects / n_sim
power1 = num_null_rejects1 / n_sim

print(power)
assert power == power1
# 0.495
```

```
0.495
```

These results indicate that using 12 participants in each group and with given statistics, the statistical power of the experiment is 49%. This can be interpreted as follows:

> **If a large effect (0.17 or greater) is truly present between control and experimental groups, then the null hypothesis (i.e. no difference with alpha 0.05) would be rejected 49% of the time.**

1.5 Sample size requirements for a given effect size

Often in behavioral research 0.8 is accepted as a sufficient level of power.

Clearly, this is not the case for the experiment as currently designed. Determine the required sample size in order to identify a difference of 0.17 or greater between the group means with an

80% power.

```
[4]: # Required power
     target = 0.8
```

```
[5]: from statsmodels.stats.power import TTestIndPower
     power = TTestIndPower()
```

```
[6]: # Determine the sample size
     power.solve_power(effect_size = 0.17/0.21, power = target, alpha = 0.05)
```

```
[6]: 24.951708908275144
```

```
[7]: # Minimum sample size to start the simulations
     sample_size = 12
     null_rejected = 0
     n_sim = 10000
```

As above, perform the following

- Initialize an empty array for storing results
- initialize a list for storing sample size x power summary
- While current power is less than the target power
  - Generate distributions for control and experimental groups using given statistics (as before)
  - Run a t-test and store results
  - Calculate current power
  - Output current sample size and power calculated for inspection
  - Store results: Sample size, power
  - increase the sample size by 1 and repeat

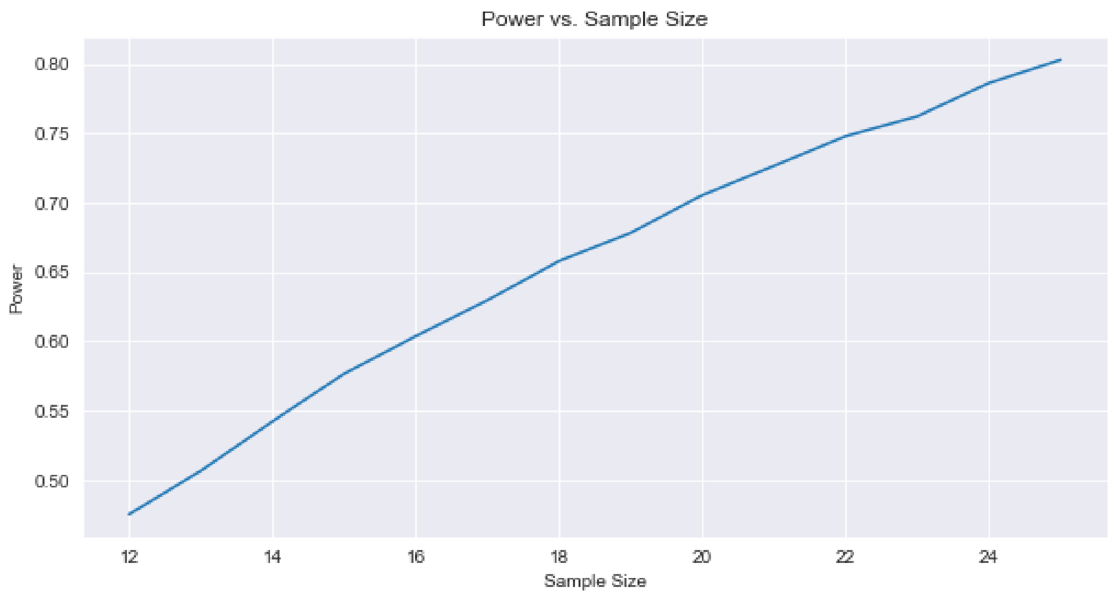
```
[8]: np.random.seed(10)

     p = (np.empty(n_sim))
     p.fill(np.nan)

     power_sample = []

     while null_rejected < target:
         ctrl = np.random.normal(control_mean, control_sd,
                                 size=[n_sim, sample_size])
         exptl = np.random.normal(experimental_mean, experimental_sd,
                                  size=[n_sim, sample_size])

         results = stats.ttest_ind(ctrl, exptl, axis = 1)
         pval = results[1]
         null_rejected = np.sum(pval < 0.05)/n_sim
```

```
    power_sample.append([null_rejected, sample_size])
    sample_size += 1



# Keep iterating as shown above until desired power is obtained


##### From GitHub Solution

# np.random.seed(10)

# p = (np.empty(n_sim))
# p.fill(np.nan)

# # Keep iterating until desired power is obtained

# power_sample = []
# while null_rejected < target:

#     data = np.empty([n_sim, sample_size, 2])
#     data.fill(np.nan)

#     # For control group
#     # Here we specify size=[n_sim, sample_size] which creates an array of␣
↪n_sim number of arrays,
#     # each containing sample_size number of elements.
#     # This is equivalent to manually looping n_sim times like we did above␣
↪but is much faster.
#     data[:,:,0] = np.random.normal(loc=control_mean, scale=control_sd,␣
↪size=[n_sim, sample_size])

#     # For experimental group
#     data[:,:,1] = np.random.normal(loc=experimental_mean,␣
↪scale=experimental_sd, size=[n_sim, sample_size])

#     result = stats.ttest_ind(data[:, :, 0],data[:, :, 1],axis=1)

#     p_vals = result[1]

#     # Since you know that all simulations are from a different distribution \
#     # all those that rejected the null-hypothesis are valid
#     null_rejected = np.sum(p_vals < 0.05) / n_sim

#     print('Number of Samples:', sample_size,', Calculated Power =',␣
↪null_rejected)
#     power_sample.append([sample_size, null_rejected])
#     # increase the number of samples by one for the next iteration of the loop
#     sample_size += 1
```

You can also plot the calculated power against sample size to visually inspect the effect of increasing sample size.

[9]:
```
# Plot a sample size X Power line graph
plt.figure(figsize=(10,5))
plt.title('Power vs. Sample Size')
plt.xlabel('Sample Size')
plt.ylabel('Power')
df = pd.DataFrame(power_sample)
df.head()
sns.lineplot(df[1], df[0]);


plt.show()
```

```
/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(
```



This output indicates that in order to get the required power (80%) to detect a difference of 0.17, you would need a considerably higher number of patients.

1.6 BONUS: Investigating the relationship between Power, Sample Size, and Effect Size

You've seen how to calculate power given alpha, sample size, and effect size. To further investigate this relationship, it is interesting to plot the relationship between power and sample size for various effect sizes.

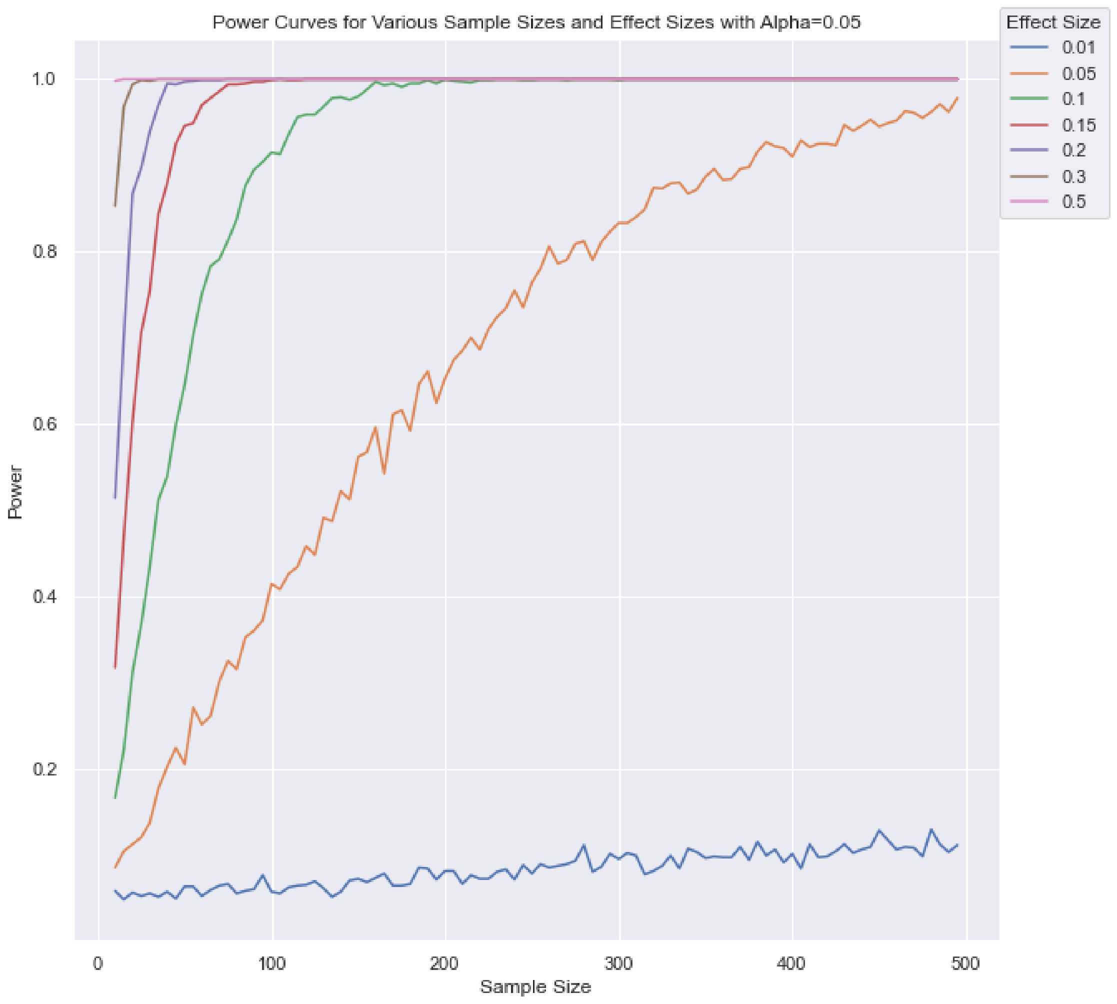
To do this, run multiple simulations for varying parameters. Then store the parameters and plot the resulting dataset. Specifically:

1. Use a value of $\alpha = 0.05$ for all of your simulations
2. Use the following effect sizes: [0.01, 0.05, 0.1, 0.15, 0.2, 0.3, 0.5]
3. Use the sample sizes from 10 to 500
4. For each effect size sample size combination, calculate the accompanying power
5. Plot a line graph of the power vs sample size relationship. You should have 7 plots; one for each of the effect sizes listed above. All 7 plots can be on the same graph but should be labeled appropriately. Plot the power on the y-axis and sample size on the x-axis.

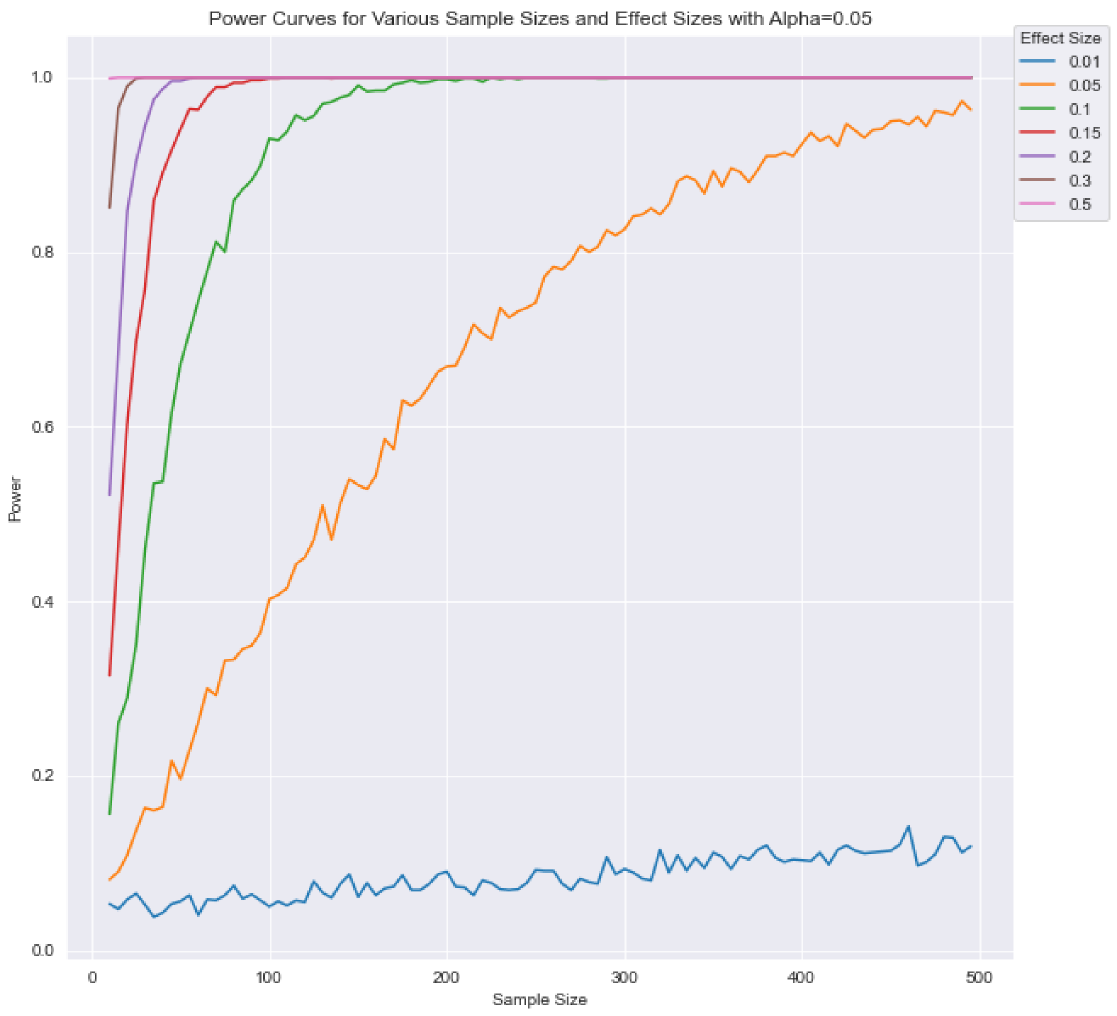
[81]:
```python
def get_powers(experimental_mean, experimental_sd = 0.21,
               control_mean = 0, control_sd = 0.21, n_sim = 10**3,
               sam_siz_min = 10, sam_siz_max = 500, step = 5):

    ctrl = np.array([])


    exptl = np.array([])
    power_sample ={}

    for i in list(range(10, 500, 5)):

        ctrl = np.random.normal(control_mean, control_sd,
                                 size=[n_sim, i])
        exptl = np.random.normal(experimental_mean, experimental_sd,
                                  size=[n_sim, i])

        results = stats.ttest_ind(ctrl, exptl, axis = 1)

        pval = results[1]

        null_rejected = np.sum(pval < 0.05)/n_sim

        power_sample[i]=null_rejected

    return power_sample
```

```
[81]: {10: 0.046,
       15: 0.06,
       20: 0.057,
       25: 0.06,
       30: 0.052,
       35: 0.058,
       40: 0.055,
       45: 0.056,
       50: 0.051,
       55: 0.047,
       60: 0.055,
       65: 0.072,
       70: 0.053,
       75: 0.059,
       80: 0.057,
       85: 0.062,
       90: 0.061,
       95: 0.056,
       100: 0.073,
       105: 0.064,
       110: 0.068,
       115: 0.062,
       120: 0.064,
       125: 0.072,
       130: 0.066,
       135: 0.07,
       140: 0.067,
       145: 0.069,
       150: 0.065,
       155: 0.057,
       160: 0.068,
       165: 0.064,
       170: 0.077,
       175: 0.074,
       180: 0.063,
       185: 0.074,
       190: 0.067,
       195: 0.068,
       200: 0.082,
       205: 0.06,
       210: 0.085,
       215: 0.068,
       220: 0.075,
       225: 0.073,
       230: 0.081,
       235: 0.082,
       240: 0.084,
       245: 0.077,
       250: 0.078,
       255: 0.081,
       260: 0.089,
       265: 0.097,
       270: 0.09,
       275: 0.082,
       280: 0.085,
       285: 0.074,
       290: 0.085,
       295: 0.084,
       300: 0.091,
       305: 0.091,
       310: 0.104,
       315: 0.095,
       320: 0.09,
       325: 0.094,
       330: 0.098,
       335: 0.092,
       340: 0.089,
       345: 0.1,
       350: 0.104,
       355: 0.092,
       360: 0.089,
       365: 0.092,
       370: 0.109,
       375: 0.107,
       380: 0.11,
       385: 0.099,
       390: 0.105,
       395: 0.097,
       400: 0.101,
       405: 0.089,
       410: 0.107,
       415: 0.084,
       420: 0.102,
       425: 0.109,
       430: 0.112,
       435: 0.114,
       440: 0.103,
       445: 0.122,
       450: 0.117,
       455: 0.099,
       460: 0.104,
       465: 0.099,
       470: 0.106,
       475: 0.125,
       480: 0.126,
       485: 0.116,
       490: 0.093,
       495: 0.124}
```

[91]:
```
experimental_mean = [0.01, 0.05, 0.1, 0.15, 0.2, 0.3, 0.5]
sns.set(rc={"figure.figsize":(10, 10)})

for item in experimental_mean:
    power = get_powers(experimental_mean = item)
    sns.lineplot(x = power.keys(), y = power.values())
plt.legend(labels = experimental_mean, title='Effect Size',loc=(1,0.8));


plt.title('Power Curves for Various Sample Sizes and Effect Sizes with Alpha=0.
 ↪05')
plt.xlabel('Sample Size')
plt.ylabel('Power');
```



[25]:
```python
#### FROM GitHub Solution

def power_curve(min_sample_size = 10, max_sample_size=500, n_sim = 1000,
                control_mean = 0, control_sd = 0.21, experimental_mean = 0.17,
                experimental_sd = 0.21):

    p = (np.empty(n_sim))
    p.fill(np.nan)

    # Keep iterating until desired power is obtained

    power_sample = []
    for sample_size in range(min_sample_size, max_sample_size, 5):

        data = np.empty([n_sim, sample_size, 2])
        data.fill(np.nan)

        # For control group
        data[:,:,0] = np.random.normal(loc=control_mean, scale=control_sd,
                                       size=[n_sim, sample_size])

        # For experimental group
        data[:,:,1] = np.random.normal(loc=experimental_mean,
                                       scale=experimental_sd,
                                       size=[n_sim, sample_size])

        result = stats.ttest_ind(data[:, :, 0],data[:, :, 1],axis=1)

        p_vals = result[1]

        # Since you know that all simulations are from a different distribution␣
 ↪\
        # all those that rejected the null-hypothesis are valid
        null_rejected = np.sum(p_vals < 0.05) / n_sim

        power_sample.append(null_rejected)

    return power_sample
cols = {}

for exp_mean in [0.01, 0.05, 0.1, 0.15, 0.2, 0.3, 0.5]:
    col = power_curve(experimental_mean=exp_mean)
    cols[exp_mean] = col
df = pd.DataFrame.from_dict(cols)
df.index = list(range(10,500,5))
df.plot(figsize=(10,10))
plt.legend(title='Effect Size',loc=(1,0.8))
plt.title('Power Curves for Various Sample Sizes and Effect Sizes with Alpha=0.
  ↪05')
plt.xlabel('Sample Size')
plt.ylabel('Power');
```



1.7 Summary

In this lesson, you gained further practice with "statistical power" and how it can be used to analyze experimental design. You ran a simulation to determine the sample size that would provide a given value of power (for a given alpha and effect size). Running simulations like this, as well as further investigations regarding required sample sizes for higher power thresholds or smaller effect sizes is critical in designing meaningful experiments where one can be confident in the subsequent conclusions drawn.

[ ]:
```

```