Choosing the right statistical test

EXPERIMENTAL DESIGN IN PYTHON

James Chapman
Curriculum Manager, DataCamp

datacamp

Selecting the right test

- Dataset's features
  - Data types
  - Distributions → **assumed to be normal in many tests!**
  - Number of variables
- Hypotheses

Result: accurate and dependable conclusions!

- t-tests, ANOVA, and chi-square

The dataset: athletic performance

- Impact of training programs and diets on athletic performance

```
athletic_perf.sample(n=5)
```

Athlete_ID	Training_Program	Diet_Type	Initial_Fitness	Performance_Inc
167	Endurance	Plant-Based	High	9.113040
289	Endurance	Keto	Low	11.039744
164	Endurance	Plant-Based	Medium	11.614835
30	Strength	Keto	Medium	7.384686
186	HIIT	High-Protein	Low	6.776078

Independent samples t-test

- Comparing means of *two groups*
- **Assumptions**: normal distribution, equal variances

```
from scipy.stats import ttest_ind
group1 = athletic_perf[athletic_perf['Training_Program'] == 'HIIT']['Performance_Inc']
group2 = athletic_perf[athletic_perf['Training_Program'] == 'Endurance']['Performance_Inc']

t_stat, p_val = ttest_ind(group1, group2)
print(f"T-statistic: {t_stat}, P-value: {p_val}")
```

```
T-statistic: 0.20671020082911742, P-value: 0.8364563849070663
```

`p_val` > α → insufficient evidence of a difference in means

One-way ANOVA

- Comparing means across *multiple (>2) groups*
- **Assumption**: equal variances among groups

```
from scipy.stats import f_oneway
program_types = ['HIIT', 'Endurance', 'Strength']
groups = [athletic_perf_data[athletic_perf_data['Training_Program'] == program]
['Performance_Increase'] for program in program_types]
f_stat, p_val = f_oneway(*groups)
print(f"F-statistic: {f_stat}, P-value: {p_val}")
```

```
F-statistic: 1.5270022393256704, P-value: 0.2188859009050602
```

`p_val` > α → insufficient evidence of a difference in means

Chi-square test of association

- Testing relationships between *categorical variables*
- No assumptions about distributions

```
from scipy.stats import chi2_contingency
import pandas as pd
contingency_table = pd.crosstab(athletic_perf['Training_Program'],
                                athletic_perf['Diet_Type'])
```

```
Diet_Type          High-Protein  Keto  Plant-Based
Training_Program
Endurance                    33    28           33
HIIT                         27    32           40
Strength                     38    29           40
```

Chi-square test of association

```
chi2_stat, p_val, dof, expected = chi2_contingency(contingency_table)
print(f"Chi2-statistic: {chi2_stat}, P-value: {p_val}")
```

```
Chi2-statistic: 2.154450885821988, P-value: 0.7073764021451127
```

`p_val` > α → insufficient evidence of an association

Let's practice!

EXPERIMENTAL DESIGN IN PYTHON

datacamp

Post-hoc analysis following ANOVA

EXPERIMENTAL DESIGN IN PYTHON

James Chapman
Curriculum Manager, DataCamp

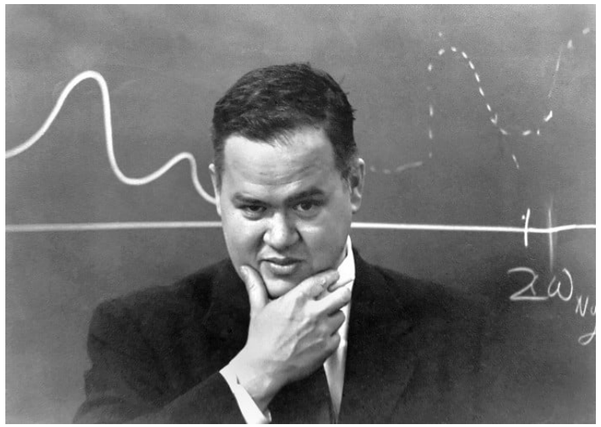
datacamp

When to use post-hoc tests

- After significant ANOVA results
- To explore pairwise group differences

Key post-hoc methods

- **Tukey's HSD** (Honest Significant Difference)
  - Robust for multiple comparisons
  - Best for *broader comparisons*
- **Bonferroni Correction**
  - Adjusts p-values to control for Type I errors
  - Best for *focused tests*



[1] https://www.amphilsoc.org/item-detail/photograph-john-wilder-tukey [2] https://en.wikipedia.org/wiki/Carlo_Emilio_Bonferroni

The dataset: marketing ad campaigns

```
ad_campaigns
```

```
              Ad_Campaign  Click_Through_Rate
1300    Seasonal Discount          2.1659547732
1661          New Arrival          2.9409657365
2762       Loyalty Reward          3.2476777154
571     Seasonal Discount          3.3382186561
775     Seasonal Discount          1.7148876401
```

Data organization with pivot tables

```
pivot_table = ad_campaigns.pivot_table(values='Click_Through_Rate',
                                       index='Ad_Campaign',
                                       aggfunc="mean")
print(pivot_table)
```

```
                  Click_Through_Rate
Ad_Campaign
Loyalty Reward              2.792716
New Arrival                 3.013843
Seasonal Discount           2.518917
```

Performing ANOVA

```
from scipy.stats import f_oneway
campaign_types = ['Seasonal Discount', 'New Arrival', 'Loyalty Reward']
groups = [ad_campaigns[ad_campaigns['Ad_Campaign'] == campaign]
['Click_Through_Rate'] for campaign in campaign_types]

f_stat, p_val = f_oneway(*groups)
print(p_val)
```

```
4.4841244969406 93e-134
```

Tukey's HSD test

```
from statsmodels.stats.multicomp import pairwise_tukeyhsd
tukey_results = pairwise_tukeyhsd(ad_campaigns['Click_Through_Rate'],
                                  ad_campaigns['Ad_Campaign'],
                                  alpha=0.05)
print(tukey_results)
```

```
         Multiple Comparison of Means - Tukey HSD, FWER=0.05
=============================================================================
       group1             group2  meandiff  p-adj   lower    upper  reject
<hr />-----------------------------------------------------------------------
Loyalty Reward        New Arrival    0.2211   0.0    0.176   0.2663    True
Loyalty Reward  Seasonal Discount   -0.2738   0.0  -0.3189  -0.2287    True
   New Arrival  Seasonal Discount   -0.4949   0.0  -0.5401  -0.4498    True
<hr />-----------------------------------------------------------------------
```

Bonferroni correction set-up

```
from scipy.stats import ttest_ind
from statsmodels.sandbox.stats.multicomp import multipletests
p_values = []

comparisons = [('Seasonal Discount', 'New Arrival'),
               ('Seasonal Discount', 'Loyalty Reward'),
               ('New Arrival', 'Loyalty Reward')]

for comp in comparisons:
    group1 = ad_campaigns[ad_campaigns['Ad_Campaign'] == comp[0]]['Click_Through_Rate']
    group2 = ad_campaigns[ad_campaigns['Ad_Campaign'] == comp[1]]['Click_Through_Rate']
    t_stat, p_val = ttest_ind(group1, group2)
    p_values.append(p_val)
```

Performing Bonferroni correction

```
p_adjusted = multipletests(p_values, alpha=0.05, method='bonferroni')
print(f"Adjusted P-values: {p_adjusted[1]}")
```

```
Adjusted P-values: [5.33634403e-133 2.17627991e-043 5.62590083e-029]
```

Let's practice!

EXPERIMENTAL DESIGN IN PYTHON

P-values, alpha, and errors

EXPERIMENTAL DESIGN IN PYTHON

James Chapman
Curriculum Manager, DataCamp

datacamp

P-values and alpha

- **P-values**: probability of observing our data if the null hypothesis was true
- α: threshold at which we consider our results *statistically significant*
- P-value $\leq \alpha$: Reject null in favor of alternative hypothesis

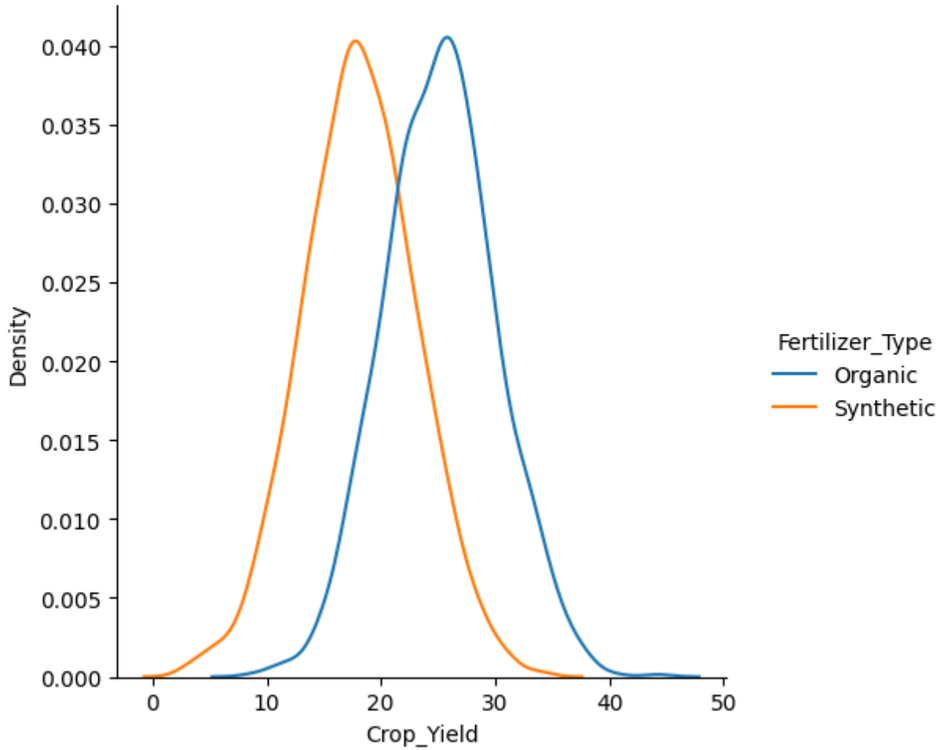
The dataset: crop yields

```
crop_yields.head()
```

```
  Fertilizer_Type  Crop_Yield
0         Organic      26.225
1       Synthetic      17.452
2         Organic      22.283
3       Synthetic      23.548
4       Synthetic      24.138
```

Visualizing the data

```
import seaborn as sns
sns.displot(data=crop_data, x="Crop_Yield", hue="Fertilizer_Type", kind="kde")
```



Conducting an independent samples t-test

$\alpha = 0.05$

```
from scipy.stats import ttest_ind
organic_yield = crop_yields[crop_yields['Fertilizer_Type'] == 'Organic']
['Crop_Yield']
synthetic_yield = crop_yields[crop_yields['Fertilizer_Type'] == 'Synthetic']
['Crop_Yield']
t_stat, p_val = ttest_ind(organic_yield, synthetic_yield)
print(p_val)
```

```
1.8496748715743899e-209
```

Exploring experimental errors

		Reality	
		H_0 False	**H_0 True**
Test	**Reject H_0**	**Correct Reject**	**Type I Error**
	Accept H_0	**Type II Error**	**Correct Accept**

- **Type I Error:** False positive, rejecting a true null hypothesis
- **Type II Error:** False negative, failing to reject a false null hypothesis

More on alpha

- Common values: 0.05, 0.01, and 0.10
  - Reflecting a 5%, 1%, and 10% probability of making a Type I error
- Choosing an α
  - Based on the context of the study
  - Balancing a tolerance for a Type I error
- Conventions:
  - 0.05 (5%): Most common, used as a convention
  - 0.01 (1%): More stringest testing, where cost of Type I error is high
  - 0.10 (10%): Sometimes in preliminary studies, where higher tolerance for Type I error

Let's practice!

EXPERIMENTAL DESIGN IN PYTHON

datacamp

Power analysis: sample and effect size

EXPERIMENTAL DESIGN IN PYTHON

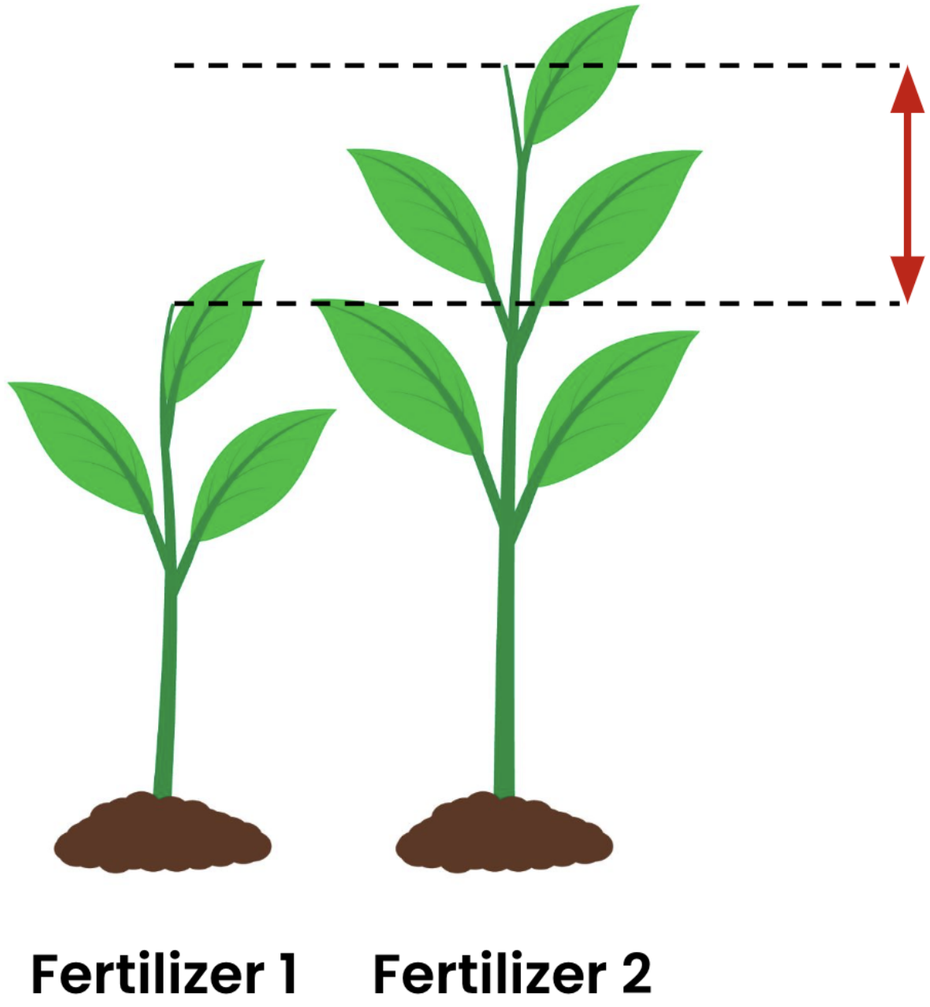
James Chapman
Curriculum Manager, DataCamp

datacamp

A primer on effect size

- **Effect size:** Quantifies the difference between two groups

- **Cohen's d:** Standard measure for effect size



The dataset: video game engagement

- 60 participants
  - 30 assigned randomly to *Action*
  - 30 assigned randomly to *Puzzle*

```
video_game_data.head()
```

```
   Game_Genre  Engagement_Time
0      Action             5.1
1      Puzzle             4.4
2      Action             7.2
3      Action             5.3
4      Puzzle             2.7
```

Calculating power overview

- **Power**: the probability of correctly rejecting a false null hypothesis: $(1 - \beta)$
  - Ranges between `0` and `1` (certainty in ability to detect a true effect)
- Assume `effect_size=1` from historical data

```
from statsmodels.stats.power import TTestIndPower
power_analysis = TTestIndPower()
power = power_analysis.solve_power(effect_size=1, nobs1=30, alpha=0.05)
print(power)
```

```
0.9677082519951168
```

Cohen's d formulation

```
def cohens_d(group1, group2):
    diff = group1.mean() - group2.mean()
    n1, n2 = len(group1), len(group2)
    var1, var2 = group1.var(), group2.var()
    pooled_std = np.sqrt(((n1 - 1) * var1 + (n2 - 1) * var2) / (n1 + n2 - 2))
    d = diff / pooled_std
    return d
```

Pooled Standard Deviation: $\sigma_p = \sqrt{\frac{(n_1-1)\times \text{var}_1 + (n_2-1)\times \text{var}_2}{n_1+n_2-2}}$

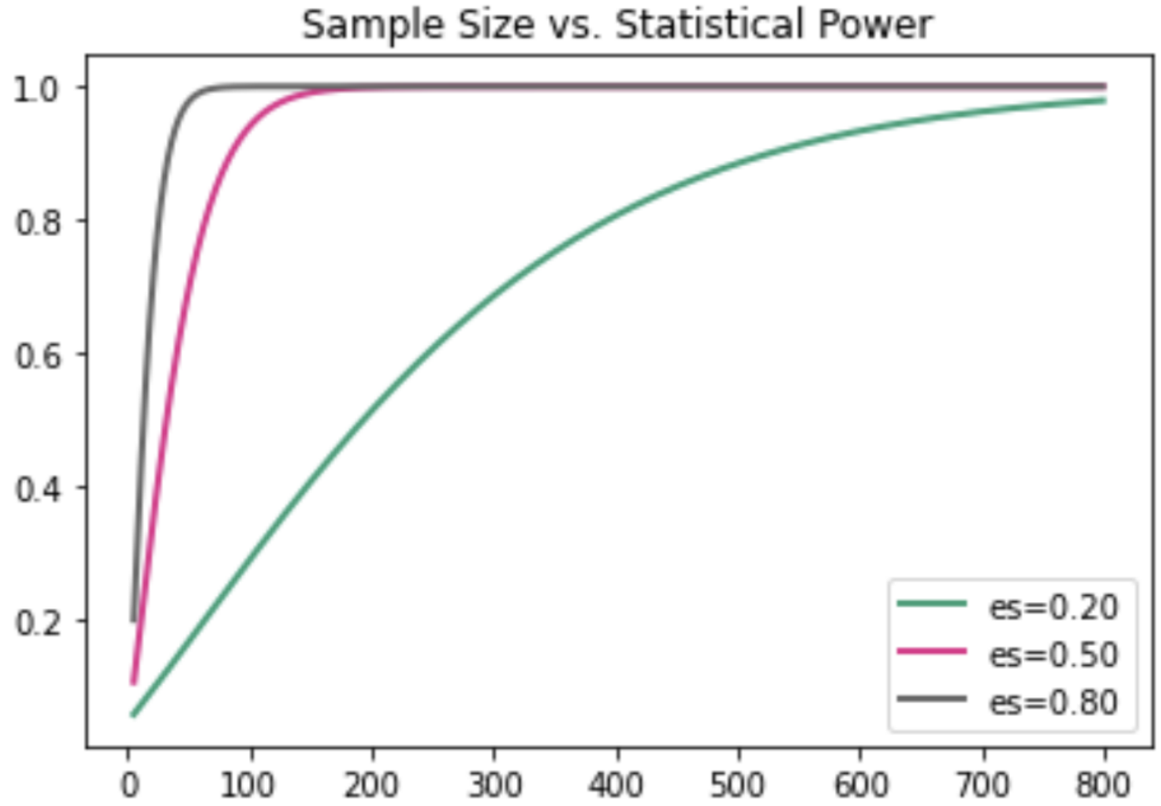
Cohen's d for video game data

```
action_times = video_game_data[video_game_data['Game_Genre'] == 'Action']['Engagement_Time']
puzzle_times = video_game_data[video_game_data['Game_Genre'] == 'Puzzle']['Engagement_Time']

d = cohens_d(action_times, puzzle_times)

print(f"Cohen's d: {d}")
```

```
Cohen's d: 1.161524633221452
```

Understanding sample size and power

- *Trade-off* in balancing power and sample size
- Larger sample sizes increase study power



[1] https://grabngoinfo.com/power-analysis-for-sample-size-using-python/

Sample size calculation in context

- Response variable: `engagement_time`

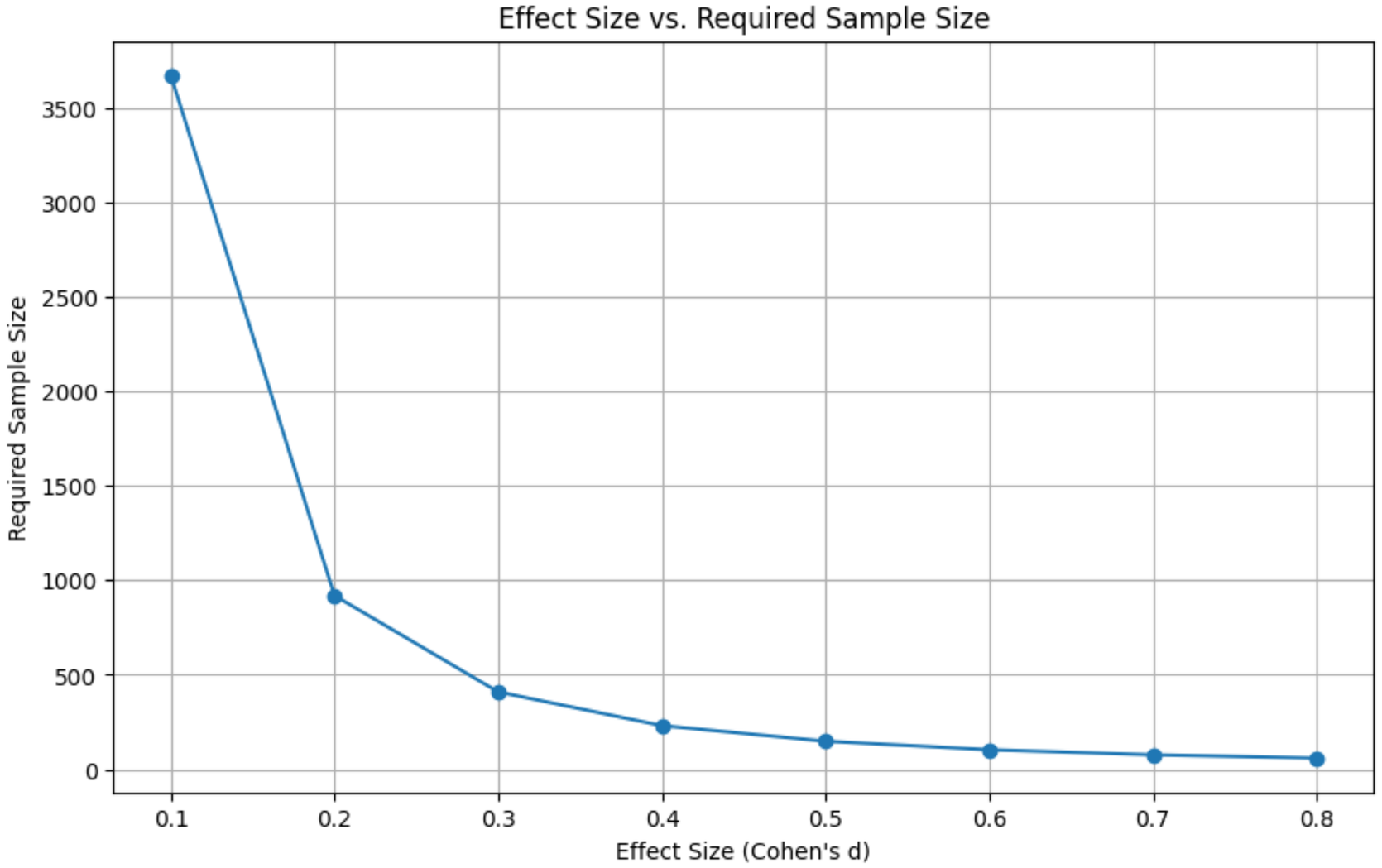
```
from statsmodels.stats.power import TTestIndPower
power_analysis = TTestIndPower()
required_n = power_analysis.solve_power(effect_size=d, alpha=0.05,
                                        power=0.99, ratio=1)
print(required_n)
```

```
28.237827708942007
```

Visualizing sample size requirements

```
import numpy as np
import matplotlib.pyplot as plt
effect_sizes = np.linspace(0.1, 0.8, 8)
sample_sizes = [power_analysis.solve_power(effect_size=es, alpha=0.05, power=0.99,
                                           ratio=1) for es in effect_sizes]

plt.figure(figsize=(10, 6))
plt.plot(effect_sizes, sample_sizes, 'o-')
plt.title('Effect Size vs. Required Sample Size')
plt.xlabel('Effect Size (Cohen\'s d)')
plt.ylabel('Required Sample Size')
plt.grid(True)
plt.show()
```

Visualizing sample size requirements



Let's practice!

EXPERIMENTAL DESIGN IN PYTHON