

Experimentation Fundamentals

Brief introduction and intuitions

Sebastian Daza

- Causal Inference Fundamentals
- Experimental Design & Randomization
- Power Analysis & Sample Size
- Sampling & Covariate Balance
- Analysis
- Non-Compliance & Confounding

Python Package: `experiment-utils-pd`



Key components:

- **ExperimentAnalyzer** - AB test analysis, IV, IPW, regression adjustment, retrodesign
- **PowerSim** - Sample size & power calculations, retrodesign
- **utils** - Balanced random assignment

Links: [PyPI](#) | [GitHub](#)

Causal Inference Fundamentals

What problem are we trying to solve? 🤔

Goal: Measure the **causal effect** of a treatment/intervention

The Fundamental Problem of Causal Inference:

- For any individual, we can only observe **one** potential outcome
- Example: Did the medication work for patient i ?
 - We observe: Patient took medication → recovered in 5 days
 - We **cannot** observe: Same patient without medication → ?
- **Individual causal effects are fundamentally unobservable**
- We need a **counterfactual**: What would have happened without treatment?

Solution: **Randomized experiments** (RCTs/A/B tests) create valid comparisons

Why Randomization Works

Without randomization:

- Treatment and control groups may differ in many ways
- Differences in outcomes could be due to pre-existing differences, not treatment
- Example: Sicker patients seek treatment → worse outcomes (confounding)

With randomization:

- Treatment assignment is **independent** of all other characteristics
- Groups are **exchangeable**: same distribution of characteristics
- Any difference in outcomes can be attributed to treatment

1. Independence (Randomization)

- Random treatment assignment creates exchangeable groups

2. Stable Unit Treatment Value Assumption (SUTVA)

- **No interference**: Units don't affect each other
- **Consistency**: Treatment uniformly defined
- Violations: Network effects, marketplace spillovers

3. Compliance

- Treatment received matches treatment assigned

Internal vs External Validity

Internal Validity: Can we trust the causal inference **within** our experiment?

- **Threats:** Selection bias, implementation errors (SRM), measurement errors, SUTVA violations

External Validity: Can we generalize **beyond** our experiment?

- **Threats:** Non-representative samples, novelty effects, seasonal/context factors

Between-Subjects Design:

- Each unit receives **only one** condition; compare **different** units (A vs B)
- *E.g.*, 50% see version A, 50% see version B
- **Limitation:** More participants needed, lower power

Within-Subjects Design:

- Each unit receives **all** conditions at different times; compare **same** unit
- *E.g.*, All users see A in week 1, B in week 2
- **Limitation:** Order/carryover effects; not for irreversible treatments

Estimand What we want to know — the target quantity

- *E.g.*, ATE: $\mathbb{E}[Y(1) - Y(0)]$; also ATT (on the treated), LATE (on compliers)

Estimator How we compute it — the method or formula

- *E.g.*, difference in means, OLS, IPW — same estimand can have multiple estimators

Estimate What we get — the actual number from our data

- *E.g.*, $\hat{\tau} = +0.056$ (5.6 pp lift); always comes with a standard error

Power Analysis & Sample Size

Statistical Power = Probability of detecting an effect when it exists

Key components:

- α (Type I error): False positive rate, typically 0.05
- β (Type II error): False negative rate, typically 0.20
- **Power** = $1 - \beta$, typically 0.80 (80%)
- **Effect size**: Magnitude of difference
- **Sample size**: Number of units per group

For more details look at: [blog's post on power analysis](#)

Minimum Detectable Effect (MDE)

MDE = Smallest effect size reliably detected at 80% power, $\alpha = 0.05$, given sample size

How to define MDE:

1. **Business:** Minimum effect for ROI (e.g., 2% revenue lift)
2. **Resource-constrained:** Achievable with sample (e.g., 50K users \rightarrow 3% MDE)
3. **Historical:** Based on past experiments (typically 1-5%)

If MDE unreasonable & limited sample:

- Avoid running under-powered experiments, risk of winner's curse (exaggerated estimates)
- Use quasi-experimental methods instead!

Small Effects + Limited Traffic: What To Do? 💡

1. **Reduce variance**: CUPED (pre-experiment covariate adjustment), regression adjustment (control for pre-treatment covariates), stratified randomization
2. **Use sensitive metrics**: surrogate/proxy metrics
3. **Redesign experiment**: within-subjects, responsive subpopulations, longer duration
4. **Accumulate evidence**: meta-analysis
5. **Go quasi-experimental**: DiD (Difference-in-Differences), Synthetic Control, Geo-experiments, Interrupted Time Series

Multiple Comparisons Problem

Problem: Every test has a 5% false positive rate by chance.
Testing m metrics simultaneously inflates this:

$$P(\text{at least 1 false positive}) = 1 - (1 - \alpha)^m \xrightarrow{m=10} \approx 40\%$$

Two correction strategies:

- FWER** Control probability of *any* false positive
Bonferroni, Holm-Bonferroni — stricter, fewer discoveries
- FDR** Control the *proportion* of false positives among significant results
Benjamini-Hochberg — less strict, better when testing many metrics

PowerSim uses a simulation approach, more useful for more complex designs and metrics (compliance, multiple variants, etc.)

```
from experiment_utils import PowerSim

p = PowerSim(metric='proportion',
             relative_effect=False,
             variants=2,
             nsim=1000,
             alpha=0.05,
             alternative='two-tailed',
             comparisons=[(1, 0), (2, 0), (2, 1)],
             correction='holm')
```


Find Sample Size & Power 🧑

```
result = p.find_sample_size(  
    target_power=0.80,  
    baseline=0.10,  
    effect=[0.03, 0.05],  
    # compliance=0.80,  
    optimize_allocation=True)
```

Using sample size 12926 (driven by (0, 1))

Optimized sample sizes: {'control': 1093, 'variant_1': 5916, 'variant_2': 5916}

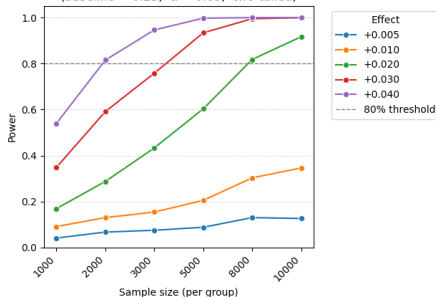
Achieved power: {'(0, 1)': 0.839, '(0, 2)': 0.986, '(1, 2)': 0.873}

1. **Always explore multiple scenarios** — not just one power calculation
 - Jointly optimize: 80% power + **realistic MDE** + acceptable duration
 - In 1–2% MDE territory: proxy metrics, CUPED
2. **MDE is the most constrained parameter** — rarely inflated to compensate
 - Should reflect **minimum business-relevant effect**, not statistical convenience
 - If MDE is unrealistic, the experiment probably shouldn't run (quasi-exp instead)
3. **Heuristics as smart defaults**
 - Keep power $\geq 80\%$ (most respected threshold)
 - Cap MDE at 5%; two-tailed tests preferred unless strong prior

Exploring the Power Surface

```
p = PowerSim(metric='proportion',  
             relative_effect=False,  
             variants=1,  
             nsim=1000)  
  
rr = p.grid_sim_power(baseline_rates=0.25,  
                     effects=[0.005, 0.01, 0.02, 0.03, 0.04],  
                     sample_sizes=[1000, 2000, 3000, 5000, 8000, 10000],  
                     hue='effect',  
                     threads=16,  
                     plot=True)
```

Power by Sample Size and Effect Size [comparison=(0, 1)]
(baseline = 0.25, $\alpha = 0.05$, two-tailed)



Sampling & Covariate Balance

Why sampling matters:

- Limited resources (budget, time, traffic)
- **Reduces variance** - Better precision in estimates
- **Increases credibility** - Shows randomization worked properly

Two main approaches:

- **Random Sampling** - Simple random selection
 - Easy to implement, may result in imbalanced small segments
- **Stratified Sampling** - Sample within dimensions
 - Ensures representation across strata, better for heterogeneous populations
 - Can use proportional or equal allocation

Blocking / Stratified Sampling 🤖

```
from experiment_utils import balanced_random_assignment

# random allocation
treatment_unblock = balanced_random_assignment(
    df,
    variants=['treatment', 'control'],
    allocation_ratio=1/2,
    balance_covariates=['age', 'previous_purchases', 'days_since_signup'],
    seed=4321
)
```

Balance Check After Assignment

Comparison: control (n=1,500) vs treatment (n=1,500)

	covariate	n_control	n_treatment	mean_control	mean_treatment	smd	balanced
	age	1500	1500	39.629457	39.578661	0.005167	y
	previous_purchases	1500	1500	3.072000	2.958000	0.065706	y
	days_since_signup	1500	1500	367.491830	345.775354	0.060344	y

Summary: 3/3 covariates balanced ($|SMD| < 0.1$)

Mean $|SMD|$: 0.0437

Max $|SMD|$: 0.0657

Blocking / Stratified Sampling 🧑💻

```
# stratified allocation
treatment_block = balanced_random_assignment(
  df,
  variants=['treatment', 'control'],
  allocation_ratio=1/2,
  stratification_covariates=['age', 'previous_purchases', 'days_since_signup'],
  seed=4321
)
```

Balance Check After Assignment

Comparison: treatment (n=1,500) vs control (n=1,500)

covariate	n_treatment	n_control	mean_treatment	mean_control	smd	balanced
age	1500	1500	39.656748	39.551370	0.010718	y
previous_purchases	1500	1500	3.015333	3.014667	0.000384	y
days_since_signup	1500	1500	358.103009	355.164175	0.008163	y

Summary: 3/3 covariates balanced ($|SMD| < 0.1$)

Mean $|SMD|$: 0.0064

Max $|SMD|$: 0.0107

Distribution Assumptions: Don't Overthink It

Why not useful:

- **CLT (Central Limit Theorem):** Sample means \approx normal for large n ($>30-50$), regardless of distribution
- Normality tests too sensitive; **sampling distribution** matters, not population

What to do:

- **Large samples:** Use standard t-tests / z-tests (rely on CLT)
- **Small samples/complex metrics:** Use **bootstrapping**
- **Extreme outliers:** Robust statistics (better models!)

Analysis

Key components:

- **Primary metrics**
 - Pre-specified metrics (e.g., revenue, conversion)
 - Multiple tests?
- **Guardrail metrics**
 - Safety checks (e.g., load time, error rates)
 - Implementation metrics
- **Sample Ratio Mismatch (SRM)**
 - Does split match expectation?
 - SRM signals implementation issues or bias

Simple analysis 🧑 (1,500 users/group, 5 pp lift)

```
from experiment_utils import ExperimentAnalyzer

analyzer_simple = ExperimentAnalyzer(
    df,
    treatment_col='treatment',
    outcomes='conversion',
    bootstrap=True,
    exp_sample_ratio=0.50,
    outcome_models={'conversion':['ols', 'logistic']},
    # pvalue_adjustment='sidak',
)
analyzer_simple.get_effects()
print(analyzer_simple.results.round(3)[['model_type', 'absolute_effect', 'relative_effect', 'srp_pvalue']])
```

	model_type	absolute_effect	relative_effect	srp_pvalue
0	ols	0.051	0.153	1.0
1	logistic	0.051	0.153	1.0

Regression adjustment controls for pre-treatment covariates to partial out noise unrelated to treatment.

Benefits (even with perfect randomization):

- **Reduced variance** → narrower confidence intervals, higher precision
- **Increased power** (especially when covariates correlate with outcome)
- **Corrects chance imbalances** in small samples ($n < 1000/\text{group}$)
- **Use when:** pre-treatment covariates available — almost always worth it

Regression Adjustment

```
from experiment_utils import ExperimentAnalyzer

# no adjustment
analyzer_simple = ExperimentAnalyzer(
    df,
    treatment_col='treatment',
    outcomes='conversion',
)
analyzer_simple.get_effects()
print(analyzer_simple.results[['absolute_effect', 'pvalue', 'standard_error']])

    absolute_effect    pvalue  standard_error
0          0.051333    0.003411          0.017531

# covariate adjustment
analyzer_adjusted = ExperimentAnalyzer(
    df,
    treatment_col='treatment',
    outcomes='conversion',
    regression_covariates=['age', 'previous_purchases', 'days_since_signup']
)

analyzer_adjusted.get_effects()
print(analyzer_adjusted.results[['absolute_effect', 'pvalue', 'standard_error']])

    absolute_effect    pvalue  standard_error
0          0.050259    0.000039          0.012212

final_effect = analyzer_adjusted.results.loc[0, 'absolute_effect']
```

Winner's Curse (Subsample of 500 users/group)

Problem: Significant results often **overestimate** true effect

Why?

- Selection bias: Only "winners" reported
- Small samples + low power → worse exaggeration (2-3x for power < 0.50)

$$\text{Exaggeration (Type M)} = \left| \frac{\hat{\tau}}{\tau} \right|$$

$$\text{Relative Bias} = \frac{\hat{\tau}}{\tau}$$

When an underpowered experiment reports a significant effect, divide the estimate by the exaggeration ratio to get a more realistic sense of the true effect size.

```
# we hunted for significant effect using a smaller sample (n=500/group)
print(analyzer_retro.results[['absolute_effect', 'pvalue', 'standard_error']])
      absolute_effect      pvalue  standard_error
0          0.080322    0.007778          0.030179

print(f'True effect: {final_effect:.4f}')
True effect: 0.0503

cols = ['power', 'type_s_error', 'type_m_error', 'relative_bias', 'trimmed_abs_effect']
print(analyzer_retro.calculate_retrodesign(true_effect=final_effect)[cols])
      power  type_s_error  type_m_error  relative_bias  trimmed_abs_effect
0  0.3996           0.0         1.5829         1.5829         0.050743
```

Non-Compliance & Confounding

What If the Experiment Is Broken? 🤖

Common problems:

- **Implementation issues** - Bugs in assignment or logging
- **SRM (Sample Ratio Mismatch)** - Observed split differs from expected
- **Non-compliance** - Users don't receive assigned treatment

What can we do?

- **ITT (Intent to Treat)** - Preserves randomization, underestimates effect
- **Regression adjustment / IPW (Inverse Probability Weighting)** - Correct for imbalances
- **IV (Instrumental Variables)** - Assignment as instrument (ITT \rightarrow LATE)

Simulation: Non-Compliance + Confounding 🧑💻

CS farming meeting scenario:

- Only a % of treated users **attend** (one-sided non-compliance)
- Attenders are more engaged, higher baseline revenue (**confounding**)

How to recover the true effect on bookings?

ITT As assigned → **underestimates** effect

Regression Covariate adjustment → reduces bias

IPW Inverse probability weighting → corrects imbalance

IV Assignment as instrument → **LATE** \equiv **ATT**
(one-sided non-compliance)

Naive analysis: Compare attenders vs non-attenders

True effect = 5 bookings

```
naive = ExperimentAnalyzer(  
    cdf,  
    treatment_col='attended',  
    outcomes='bookings',  
)  
  
naive.get_effects()  
  
naive.results[['absolute_effect', 'abs_effect_lower', 'abs_effect_upper', 'pvalue']]
```

	absolute_effect	abs_effect_lower	abs_effect_upper	pvalue
0	6.316905	5.921429	6.712381	3.823925e-215

ITT: Compare all treated vs control, regardless of attendance

True effect = 5 bookings

```
itt = ExperimentAnalyzer(  
    cdf,  
    treatment_col='assigned',  
    outcomes='bookings',  
)  
itt.get_effects()  
  
itt.results[['absolute_effect', 'abs_effect_lower', 'abs_effect_upper', 'pvalue']]
```

absolute_effect	abs_effect_lower	abs_effect_upper	pvalue
0	2.743512	2.359612	3.127413 1.418363e-44

IPW: Adjust for compliance selection using covariates

True effect = 5 bookings

```
ipw = ExperimentAnalyzer(  
    cdf,  
    treatment_col='attended', outcomes='bookings',  
    balance_covariates=['engagement', 'account_age_months', 'monthly_usage'],  
    adjustment='balance', balance_method='ps-logistic', estimand='ATT', overlap_plot=True)  
  
ipw.get_effects()  
ipw.results[['absolute_effect', 'abs_effect_lower', 'abs_effect_upper', 'pvalue']]  
    absolute_effect  abs_effect_lower  abs_effect_upper      pvalue  
0          5.056775          4.652374          5.461177  1.212354e-132
```

Why it matters:

- IPW requires **common support**: every unit needs a non-zero probability of either treatment
- Without it, weights become extreme → **high variance and bias**

What to do:

- **Diagnose**: Inspect propensity score distributions by group
- **Trim**: Drop or down-weight units outside common support
- **Target**: Shift estimand from ATE → ATT

Instrumental Variables (IV)

IV: Use assignment as instrument to isolate the effect on compliers (attenders)

One-sided non-compliance: $LATE \equiv ATT$, since there are no always-takers.

True effect = 5 bookings

```
iv = ExperimentAnalyzer(  
    cdf,  
    treatment_col='attended',  
    outcomes='bookings',  
    regression_covariates=['engagement', 'account_age_months', 'monthly_usage'],  
    instrument_col='assigned',  
    adjustment='IV'  
)  
iv.get_effects()  
print(iv.results[['absolute_effect', 'abs_effect_lower', 'abs_effect_upper', 'pvalue']])
```

	absolute_effect	abs_effect_lower	abs_effect_upper	pvalue
0	4.870595	4.381111	5.36008	0.0

Goal: Pool effect estimates across multiple experiments into a single estimate

Inverse-variance weighting:

- Each experiment's estimate is weighted by $w_i = 1/\sigma_i^2$
- More precise experiments (smaller SE) get **more weight**

When to use it:

- Multiple experiments testing the **same intervention**
- Any single experiment underpowered on its own
- Experiments run across different **regions / segments**

Key assumption: All experiments share the same true effect (homogeneity)

Setup: 5 experiments, **baselines correlated with allocation** →
naive pooling is biased

```
# high-alloc -> low baseline; low-alloc -> high baseline
experiments = [
  {"name": "exp_1", "n": 3000, "alloc": 0.20, "baseline": 0.20},
  {"name": "exp_2", "n": 2000, "alloc": 0.30, "baseline": 0.25},
  {"name": "exp_3", "n": 1500, "alloc": 0.50, "baseline": 0.30},
  {"name": "exp_4", "n": 1000, "alloc": 0.70, "baseline": 0.40},
  {"name": "exp_5", "n": 800, "alloc": 0.80, "baseline": 0.50},
]
# true_effect = 0.05 in all experiments

# naive pooled analysis: ignores experiment structure
naive = ExperimentAnalyzer(data=meta_df, treatment_col="treatment", outcomes=["conversion"])
naive.get_effects()
# absolute_effect = 0.1431  <-- 2.9x the true effect!

# correct: fixed-effects meta-analysis
analyzer = ExperimentAnalyzer(
  data=meta_df, treatment_col="treatment",
  outcomes=["conversion"], experiment_identifier="experiment",
)
analyzer.get_effects()
pooled = analyzer.combine_effects(grouping_cols=["outcome"])
```

Fixed-Effects Meta-Analysis

```
# per-experiment results
print(analyzer.results[["experiment", "absolute_effect", "standard_error", "pvalue"]].round(4)
  experiment absolute_effect standard_error pvalue
  exp_1      0.0558         0.0196  0.0043
  exp_2      0.0621         0.0225  0.0057
  exp_3      0.0707         0.0244  0.0038
  exp_4      0.0586         0.0341  0.0856
  exp_5      0.0406         0.0443  0.3596

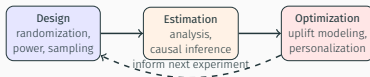
print(pooled[["outcome", "experiments", "absolute_effect", "standard_error", "pvalue"]])
  outcome experiments absolute_effect standard_error pvalue
conversion      5.0      0.0600         0.0114  1.54e-07

# True effect = 0.05
# Naive pooled = 0.14 <-- 2.9x overestimate
# Meta-analysis = 0.06 <-- recovers true effect
```

Key takeaways ✨

- Power analysis is crucial for experiment design; MDE should reflect business relevance, not statistical convenience
- Sampling methods (random vs stratified) can improve balance and precision
- Regression adjustment reduces variance and corrects for imbalances, even with perfect randomization
- Non-compliance and confounding can be addressed with ITT, IPW, and IV methods
- Meta-analysis allows pooling across multiple experiments

The Inference Cycle



Other techniques to explore:

- **Quasi-experimental:** DiD, Synthetic Control, Geo-experiments, ITS
- **Heterogeneous effects:** Uplift modeling, causal forests (HTE)
- **Sequential testing:** Always-valid inference, alpha-spending (peeking)
- **Bayesian experimentation:** Prior incorporation, posterior decision rules

- Simulation code
- `experiment-utils-pd`