

Experimentation Fundamentals



Brief introduction and intuitions

Sebastian Daza



- Introduction & Causal Inference Fundamentals
- Experimental Design & Randomization
- Power Analysis & Sample Size
- Sampling & Covariate Balance
- AB Analysis + Confounding & Non-compliance Simulations

Python Package: experiment-utils-pd



A Python package for designing, analyzing, and validating experiments with advanced causal inference capabilities



Key components:

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

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

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** treatment condition
- Compare **different** units: Group A vs Group B
- Example: 50% users see version A, 50% see version B
- **Key limitation:** More participants needed, lower power

Within-Subjects Design:

- Each unit receives **all** treatment conditions (at different times)
- Compare **same** unit against itself
- Example: All users see version A in week 1, version B in week 2
- **Key limitation:** Order/carryover effects, not irreversible treatments

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 → 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, regression adjustment, stratified randomization
2. **Use sensitive metrics**: surrogate/proxy metrics
3. **Redesign experiment**: within-subjects, responsive subpopulations, longer duration
4. **Accumulate evidence**: meta-analysis
(combine_effects)
5. **Go quasi-experimental**: DiD, Synthetic Control, Geo-experiments, Interrupted Time Series



Problem: Testing m metrics at $\alpha = 0.05$ expects 1 false positive per 20 tests

$$P(\text{at least 1 false positive}) = 1 - (1 - \alpha)^m$$

Solutions:

- Multiple comparison corrections
- Bonferroni, Holm-Bonferroni, Sidak, Benjamini-Hochberg (FDR)

Find Sample Size & Power



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

```
from experiment_utils.power_sim 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 17202 (driven by (0, 1))

Optimized sample sizes: {'control': 1455, 'variant_1': 7873, 'variant_2': 7873}

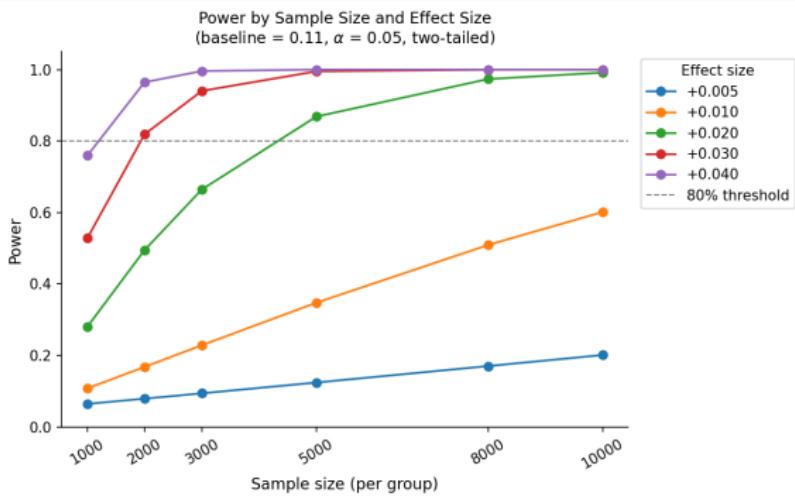
Achieved power: {'(0, 1)': 0.907, '(0, 2)': 1.0, '(1, 2)': 0.942}



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],  
    plot=True)
```





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.utils import balanced_random_assignment

# simulated data with covariates (3000 rows):
age previous_purchases days_since_signup treatment conversion
0 29.143694 5 464.228095 treatment 0
1 49.973454 2 577.496020 control 0
2 42.829785 6 806.725758 treatment 1
3 24.937053 6 1324.618460 control 0
4 34.213997 5 415.036894 treatment 0

# random allocation
treatment = balanced_random_assignment(
    df,
    variants=['control', 'treatment'],
    allocation_ratio=0.5,
    check_balance_covariates=['age', 'previous_purchases', 'days_since_signup'],
    # balance_covariates=['age', 'previous_purchases', 'days_since_signup']
)

-----
covariate mean_treatment mean_control smd balanced
age 40.046056 40.415916 -0.036537 y
previous_purchases 3.088000 3.004000 0.048583 y
days_since_signup 365.495181 358.738388 0.018646 y

Summary: 3/3 covariates balanced (|SMD| < 0.1)
Mean |SMD|: 0.0346
Max |SMD|: 0.0486
=====
```

Blocking / Stratified Sampling 🤖

```
# stratified allocation
treatment = balanced_random_assignment(
    df,
    variants=['control', 'treatment'],
    allocation_ratio=0.5,
    check_balance_covariates=['age', 'previous_purchases', 'days_since_signup'],
    balance_covariates=['age', 'previous_purchases', 'days_since_signup']
)
```

covariate	mean_control	mean_treatment	smd	balanced
age	40.167258	40.297051	-0.012818	y
previous_purchases	3.062868	3.028513	0.019869	y
days_since_signup	362.615133	361.600167	0.002801	y

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

Mean $|SMD|: 0.0118$

Max $|SMD|: 0.0199$



Even with perfect randomization, regression adjustment helps:

- Reduced variance → narrower confidence intervals, higher precision
- Increased power to detect effects (especially with correlated covariates)
- Corrects for chance imbalances in small samples

When to use:

- Pre-treatment covariates strongly correlated with outcome (most of the time)
- Sample size < 1000 per group (chance imbalances more likely)

Regression Adjustment



```
from experiment_utils.experiment_analyzer import ExperimentAnalyzer

# no adjustment
analyzer_simple = ExperimentAnalyzer(
    df,
    treatment_col='treatment',
    outcomes='conversion'
)
analyzer_simple.get_effects()
analyzer_simple.results['standard_error']
0.0173

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

analyzer_adjusted.get_effects()
analyzer_adjusted.results['standard_error']
0.0122
```

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.

Winner's Curse 💀

```
# reduce the sample size (~500 per group)
sdf = df.sample(1000)
analyzer_simple = ExperimentAnalyzer(
    sdf,
    treatment_col='treatment',
    outcomes='conversion'
)
analyzer_simple.get_effects()
analyzer_simple.results[['absolute_effect', 'pvalue']]
   absolute_effect      pvalue
0           0.076136  0.011805

# let's assess the potential bias (we know the MDE is 0.045)
analyzer_simple.calculate_retrodesign(true_effect=0.045)
cols = ['power', 'type_s_error', 'type_m_error', 'relative_bias', 'trimmed_abs_effect']
print(analyzer_simple.calculate_retrodesign(true_effect=true_effect)[cols])

   power  type_s_error  type_m_error  relative_bias  trimmed_abs_effect
0  0.336        0.0006       1.7265       1.7248          0.04414
```

More on 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 🧑

```
from experiment_utils import ExperimentAnalyzer

analyzer = ExperimentAnalyzer(
    df,
    treatment_col='treatment',
    outcomes='conversion',
    bootstrap=True,
    exp_sample_ratio_col='expected_sample_ratio',
    outcome_models={'conversion': ['ols', 'logistic']},
    # pvalue_adjustment='sidak',
)
analyzer.get_effects()
print(analyzer.results.round(3)[['model_type', 'absolute_effect', 'relative_effect',
    'rel_effect_lower', 'rel_effect_upper', 'srmpvalue']])
```

model_type	absolute_effect	relative_effect	rel_effect_lower	rel_effect_upper	srmpvalue
0 ols	0.056	0.179	0.075	0.309	0.320
1 logistic	0.056	0.179	0.074	0.307	0.320

Distribution Assumptions: Don't Overthink It 🧠

Why not useful:

- **CLT:** 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!)

What If the Experiment Is Broken? 🤯

Common problems:

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

What can we do?

- **ITT** - Preserves randomization, underestimates effect
- **Regression adjustment / IPW** - Correct for imbalances
- **IV** - Assignment as instrument (ITT → LATE)



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 Approach



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
```

Intend-to-Treat (ITT)

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',  
    covariates=['engagement', 'account_age_months', 'monthly_usage'],  
    adjustment='balance', balance_method='ps-logistic', target_effect='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  
  
# what about just regression adjustment?  
reg_adj = ExperimentAnalyzer(  
    cdf,  
    treatment_col='attended', outcomes='bookings',  
    covariates=['engagement', 'account_age_months', 'monthly_usage'],  
    regression_covariates=['engagement', 'account_age_months', 'monthly_usage'])  
  
reg_adj.get_effects()  
reg_adj.results[['absolute_effect', 'abs_effect_lower', 'abs_effect_upper']]  
    absolute_effect  abs_effect_lower  abs_effect_upper      pvalue  
0            5.086338          4.766525          5.406151  2.597419e-213
```

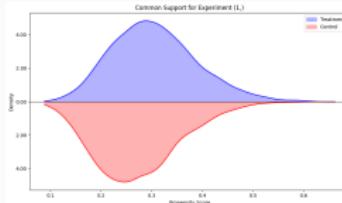


Why it matters:

- 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: Assignment as instrument → One-sided non-compliance →
LATE \equiv ATT (compliers = attenders)

Since always-takers don't exist, every attender is a complier.

True effect = 5 bookings

```
iv = ExperimentAnalyzer(  
    cdf,  
    treatment_col='attended',  
    outcomes='bookings',  
    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
```

We only want to use the part of attendance variation that came from the random assignment. I'm throwing away self-selection.

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)

Fixed-Effects Meta-Analysis



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 = 0.1397    <-- 2.8x 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        0.0758       0.0200  0.0001
    exp_2        0.0319       0.0221  0.1494
    exp_3        0.0333       0.0247  0.1774
    exp_4        0.0800       0.0342  0.0193
    exp_5        0.0203       0.0443  0.6464

print(pooled[["outcome", "experiments", "absolute_effect", "standard_error", "pvalue"]])
  outcome experiments absolute_effect standard_error pvalue
  conversion           5.0        0.0514         0.0115  0.000008

# True effect = 0.05
# Naive pooled = 0.14 <-- 2.8x overestimate
# Meta-analysis = 0.05 <-- 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

Code

- <https://github.com/sdaza/experiment-utils-pd>