ABacus

Release 0.0.1

Data Sapience. Advanced Analytics team

Feb 08, 2024

USER GUIDE

ABacus is a Python library developed for A/B experimentation and testing. It includes versatile instruments for different experimentation tasks like experiment design, sample size determination, results evaluation, visualisation and reporting.

CHAPTER

ONE

IMPORTANT FEATURES

- Experiment design: type I/II errors, effect size, sample size simulations.
- Groups splitting.
- A/A test and evaluation of splitter accuracy.
- Evaluation of experiment results with various statistical tests and approaches.
- Sensitivity increasing techniques like CUPED, CUPAC.

Note: This project is under active development.

1.1 Installation

You can use **pip** to install **ABacus** from Github and use it for your projects:

```
pip install pip+https://github.com/kolmogorov-lab/abacus
```

Later the package will be published in **PyPI** and will be able to be installed with

```
pip install kolmogorov-abacus
```

Note: ABacus requires Python 3.11+.

1.2 Experiment Initialization

Before actual analysis, you have to define your experiment. Here is how you can do it:

```
from abacus.auto_ab.abtest import ABTest
from abacus.auto_ab.params import ABTestParams, DataParams, HypothesisParams

df = pd.read_csv('./data/ab_data.csv')

data_params = DataParams(
    id_col='user_id',
    group_col='groups',
    control_name='control',
    treatment_name='treatment',
```

(continues on next page)

(continued from previous page)

```
    target='check_rub_campaign',
)

hypothesis_params = HypothesisParams(
    alpha=0.01,
    beta=0.2,
    alternative='greater',
    metric_type='solid',
    metric_name='95th quantile',
    metric=lambda x: np.quantile(x, 0.95)
)

ab_params = ABTestParams(data_params, hypothesis_params)
ab_test = ABTest(df, ab_params)
```

As you can see, you just need to describe data and your hypothesis.

For data, you have to define columns and their purposes. Required attributes are:

- `id_col` is observation id. It can be user_id or any other id for your rows. Note that if your observations are somehow dependent (e.g. several checks per user), they must have the same id_col.
- `group_col` contains group names. If your data have two groups, then there mush be only two unique values in this column.
- `control_name` and `treatment_name` are group names e.g. 'control', 'treatment', 'A', 'B', 'control group', 'send sms', 'do not send sms', etc.
- `target` is obviously target column containing metric of interest.

Hypothesis is described with:

- `alpha` — type I error.
- `beta` — type II error.
- `alternative` — alternative of hypothesis (two-sided, less, or greater.
- `metric_type` — metric type. There are three of them: continuous, binary, and ratio.
- `metric_name` — metric name, either default ('mean' or 'median') or customer (e.g. '95th percentile').
- `metric` — function for metric calculation if `metric_name` is not default.

1.3 Experiment Evaluation

After the initialization of experiment, we are ready to dive into the analysis.

You have the following options for analysis:

- Statistical Inference
- Metric Transformations
- Increasing Sensitivity (Variance Reduction)
- Visualizations
- Reporting

1.3.1 Statistical Inference

ABacus supports three types of metrics: continuous, binary, and ratio. Each of these types requires its own particular methods to conduct statistical analysis of experiment.

ABacus has the following statistical tests for each type of metric:

1. For continuous metrics: `Welch t-test`, `Mann-Whitney U-test`, `bootstrap`.
2. For binary metrics: `chi-squared test`, `Z-test`.
3. For ratio metrics: `delta method`, `Taylor method`.

To get the result of a test, just call the appropriate statistical method on your ABTest instance:

```
ab_test = ABTest(...)

ab_test.test_welch()
{'stat': 5.172, 'p-value': 0.312, 'result': 0}

# or

ab_test.test_mannwhitney()
{'stat': 0.12, 'p-value': 0.67, 'result': 0}
```

As a result, you'll get dictionary with

- statistic of the test,
- p-value of this empirical statistic,
- result in binary form: 0 - H0 is not rejected, 1 - H0 is not accepted.

1.3.2 Metric Transformations

Sometimes experiment data cannot be analyzed directly due to different limitations such as presense of outliers or form of distribution. Metric transformation techniques available in ABacus are:

- **Outliers removal**: direct exclusion of outliers according to some algorithm. There are two methods implemented in **ABacus**: remove `top 5%` observations and `isolation forest`.

```
hypothesis_params = HypothesisParams(..., filter_method='isolation_forest')

ab_test = ABTest(...)
ab_test_2 = ab_test.filter_outliers()

print(ab_test.params.data_params.control)
# 200 000

print(ab_test_2.params.data_params.control)
# 198 201
```

- **Functional transformation**: application of any function to your target metric in order to make it more normal or remove outliers. The following example includes functional transformation with `sqrt` function:

```
hypothesis_params = HypothesisParams(..., metric_transform=np.sqrt)

ab_test = ABTest(...)
ab_test_2 = ab_test.metric_transform()
```

- **Bucketing**: aggregation of target metric into buckets in order to obtain smaller number of points for analysis and from initial distribution to distributions of means.

```
hypothesis_params = HypothesisParams(..., n_buckets=1500)

ab_test = ABTest(...)
ab_test_2 = ab_test.bucketing()
```

- **Linearization**: remove dependence of observations (and move from ratio target) using linearization approach.

```
data_params = DataParams(..., is_grouped=False)

ab_test = ABTest(...)
ab_test_2 = ab_test.linearization()
```

1.3.3 Increasing Sensitivity (Variance Reduction)

As you want to make your metrics more sensitive, you will mostly likely want to use some sensitivity increasing techniques. **ABacus** supports the following options for increasing sensitivity of your experiments:

- **CUPED (Controlled experiment Using Pre-Experiment Data)** uses information about covariate independent from experiment.

```
data_params = DataParams(..., covariate='pre_experiment_metric')

ab_test = ABTest(...)
ab_test_2 = ab_test.cuped()
```

- **CUPAC (Control Using Predictions as Covariate)** predicts variable that can be used as a covariate.

```
data_params = DataParams(..., predictors_prev=['pre_pred_1', 'pre_pred_2'],
                              predictors_now=['now_pred_1', 'now_pred_2'],
                              target_prev='pre_experiment_metric')

ab_test = ABTest(...)
ab_test_2 = ab_test.cupac()
```

- **Stratification** allows you to remove variance using not sample random sampling, but stratified sampling.

```
data_params = DataParams(..., strata_col='city')
hypothesis_params = HypothesisParams(..., strata='city',
                                          strata_weights={
```

(continues on next page)

(continued from previous page)

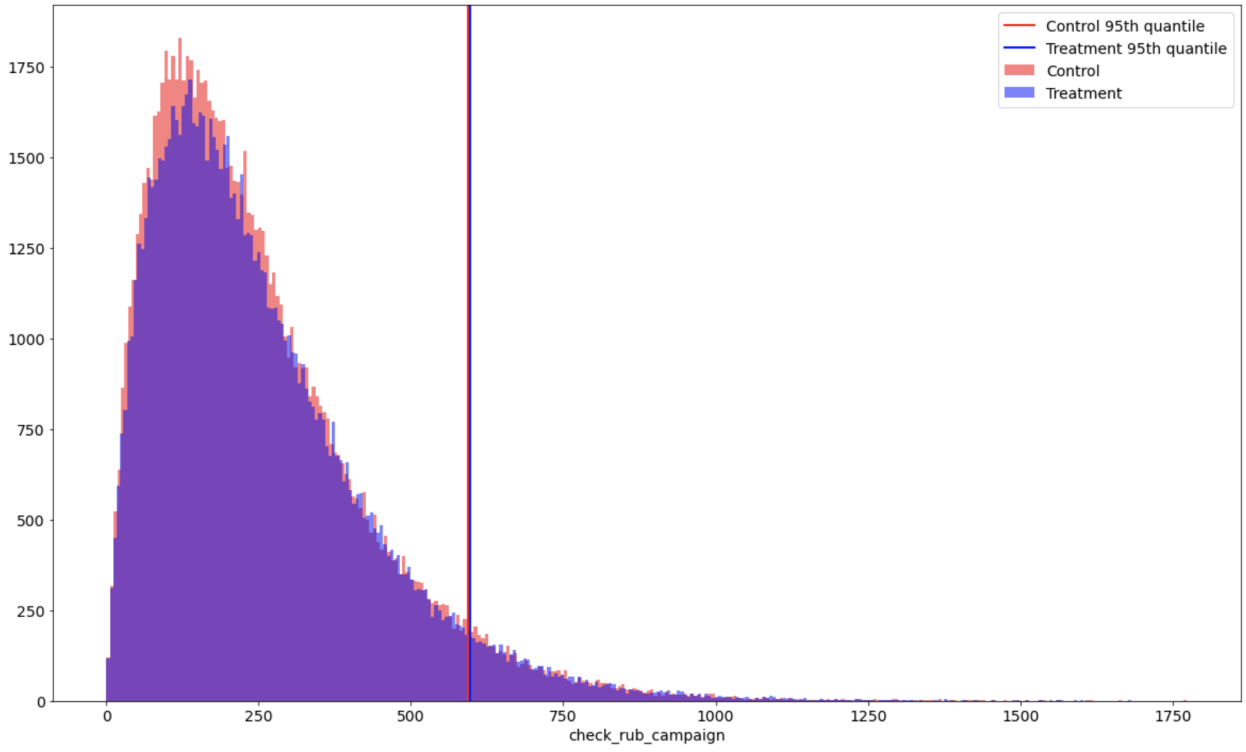
```
                                                'Moscow': 0.6,
                                                'Voronezh': 0.1,
                                                'Samara': 0.3
                    })

ab_test = ABTest(...)
ab_test_2 = ab_test.test_strat_confint()
```

1.3.4 Visualizations

A picture is worth a thousand words. No doubt that you want to visually explore your experiment.

You can plot experiments with continuous and binary variables. Continuous plots illustrates not only distributions of desired targe variable, but also a desired metric of a distribution. You can also plot a bootstrap distribution of differences if you want to estimate your experiment with bootstrap approach.

Here is the output of `ab_test.plot()` method:



1.3.5 Reporting

As you may wish to get some sort of report with information of your experiment, you can definitely do it with ABacus.

You just need to call `ab_test.report()` and get information about preprocessing steps and results of statistical tests:

```
Parameters of experiment:
- Metric: mean.
- Errors: alpha = 0.05, beta = 0.2.
- Alternative: two-sided.

Control group:
- Observations: 50
- Mean: 0.0000
- Median: 0.0070
- 25th quantile: -0.0522
- 75th quantile: 0.0416
- Minimum: -0.1635
- Maximum: 0.1263

Treatment group:
- Observations: 50
- Mean: -0.0423
- Median: -0.0395
- 25th quantile: -0.0916
- 75th quantile: 0.0058
- Minimum: -0.1769
- Maximum: 0.0789

Transformations applied: linearization -> cuped -> bucketing.
Number of bootstrap iterations: 100.
Number of buckets: 50

Following statistical tests are used:
- Welch's t-test: -3.39, p-value = 0.0010, H0 is rejected.
- Mann Whitney's U-test: 1693.00, p-value = 0.0023, H0 is rejected.
- Bootstrap test: H0 is rejected.

All three stat. tests showed that H0 is rejected.
```

Report is available for any metric type. On each metric type, you will get a bit different results.

1.3.6 Everything at once

You can freely mix everything you saw above using **chaining**.

```
ab_test = ABTest(...).filter_outliers().metric_transform().cuped().bucketing()
ab_test.test_welch()
```

As you can see, you just need to call methods one by one. `ab_test.report()` will show information about all applied transformations:

```
Parameters of experiment:
- Metric type: continuous.
- Metric: median.
- Errors: alpha = 0.01, beta = 0.2.
- Alternative: greater.

Control group:
- Observations: 1000
- Mean: 0.1971
- Median: 0.2156
- 25th quantile: 0.0800
- 75th quantile: 0.3312
- Minimum: -0.6130
- Maximum: 0.7552
- St.deviation: 0.1878
- Variance: 0.0353

Treatment group:
- Observations: 1000
- Mean: 0.3136
- Median: 0.3288
- 25th quantile: 0.2071
- 75th quantile: 0.4403
- Minimum: -0.9597
- Maximum: 0.7536
- St.deviation: 0.1878
- Variance: 0.0353

Transformations applied: metric transform -> filter outliers -> linearization -> cuped -> bucketing.

Number of bootstrap iterations: 1000.
Number of buckets: 1000.
Metric transformation applied: sqrt.
Outliers filtering method applied: top_5.

Following statistical tests are used:
- Welch's t-test: 13.78, p-value = 0.0000, H0 is rejected.
- Mann Whitney's U-test: 322535.00, p-value = 1.0000, H0 is not rejected.
- Bootstrap test: H0 is rejected.

Two out of three stat. tests showed that H0 is rejected.
```

1.4 Splitter

Splitter is a core instrument that allows you to get 'equal' groups for your experiment. Groups of an experiment are equal in the sense of users' desired characteristic of experiment are equal.

It is a crucial part of any experiment design - to get approximately equal groups. Splitter in **ABacus** not only allows you to split your observations into groups, but also assesses the quality of this split.

```
df = pd.read_csv('./data/ab_data.csv')

split_builder_params = SplitBuilderParams(
    map_group_names_to_sizes={
        'control': 20_000,
        'target': 30_000
    },
    main_strata_col = "city",
    split_metric_col = "check_rub_campaign",
    id_col = "user_id",
    cols = ["check_rub_pre_campaign"],
    cat_cols=["gender"],
```

(continues on next page)

(continued from previous page)

```
    pvalue=0.05,
    n_bins = 6,
    min_cluster_size = 500
)

split_builder = SplitBuilder(df, split_builder_params)
split = split_builder.collect()

split.head()
```

After the application of splitter to your data, you will see two additional columns to your data — **strata** and **group_name**:

	user_id	gender	age	city	check_rub_campaign	check_rub_pre_campaign	has_transaction	clicks	session_duration	strata	group_name
0	189963	0.351604	64	Moscow	337.9	322.498048	0	12	91	Moscow4-1	target
1	42186	0.648396	44	Perm	84.8	80.795804	0	18	7	Perm0-1	control
2	3038	0.648396	35	Moscow	428.9	410.870342	0	13	12	Moscow5-1	control
3	125577	0.351604	49	St.Petersburg	401.4	416.752732	0	3	10	St.Petersburg4-1	control
4	10191	0.351604	53	St.Petersburg	122.8	108.475735	0	29	24	St.Petersburg1-1	target

- `strata`: strata of observation created by clustering algorithm (HDBSCAN).
- `group_name`: groups of experiment. Control group have the same group name - `control`, and the treatment is called `target`.

1.5 MDE Researcher

MDE Researcher makes experimental design in order to get all the information about your experiment. The main purpose of its usage is calculation of samples size needed to detect particular effect size based on type I and II errors, directionality of hypothesis and other parameters.

There are three components in experimental design:

- data and hypothesis parameters;
- splitter parameters;
- actual experimental design parameters (this section).

This is an example of everything at once for experimental design:

```
from abacus.splitter.split_builder import SplitBuilder
from abacus.splitter.params import SplitBuilderParams
from abacus.mde_researcher.params import MdeParams
from abacus.mde_researcher.mde_research_builder import MdeResearchBuilder
from abacus.mde_researcher.multiple_split_builder import MultipleSplitBuilder

# data, data params and hypothesis
df = pd.read_csv('./data/ab_data.csv')
data_params = DataParams(
    id_col='user_id',
    group_col='groups',
    control_name='control',
    treatment_name='treatment',
```

(continues on next page)

(continued from previous page)

```
    is_grouped=True,
    target='check_rub_campaign'
)
hypothesis_params = HypothesisParams(
    alpha=0.01,
    beta=0.2,
    alternative='greater',
    metric_type='continuous',
    metric_name='mean',
)
ab_params = ABTestParams(data_params, hypothesis_params)

# splitter params
split_builder_params = SplitBuilderParams(
    map_group_names_to_sizes={
        'control': None,
        'target': None
    },
    main_strata_col = "city",
    split_metric_col = "check_rub_campaign",
    id_col = "user_id",
    cols = ["check_rub_pre_campaign"],
    cat_cols=["gender"],
    pvalue=0.05,
    n_bins = 6,
    min_cluster_size = 500
)

# design params
experiment_params = MdeParams(
    metrics_names=['check_rub_campaign'],
    injects=[1.010, 1.013, 1.015, 1.018, 1.02, 1.030],
    min_group_size=5_000,
    max_group_size=30_000,
    step=5_000,
    variance_reduction=None,
    use_buckets=False,
    stat_test=ABTest.test_welch,
    iterations_number=10,
    max_beta_score=0.9,
    min_beta_score=0.2,
)

# simulation of experimental design
design = MdeResearchBuilder(df,
                            ab_params,
                            experiment_params,
                            split_builder_params)
beta, alpha = design.collect()
```

As a result, you will see something similar to the following tables:

- for type II error (β)

	split_rate	(5000, 5000)	(10000, 10000)	(15000, 15000)	(20000, 20000)	(25000, 25000)	(30000, 30000)
metric	MDE						
check_rub_campaign	1.0%	>=0.9	>=0.9	>=0.9	>=0.9	>=0.9	>=0.9
	1.3%	>=0.9	>=0.9	>=0.9	>=0.9	0.8	0.4
	1.5%	>=0.9	>=0.9	>=0.9	0.5	<=0.2	<=0.2
	1.8%	>=0.9	>=0.9	0.6	0.3	<=0.2	<=0.2
	2.0%	>=0.9	0.8	0.3	<=0.2	<=0.2	<=0.2
	3.0%	>=0.9	<=0.2	<=0.2	<=0.2	<=0.2	<=0.2

Table should be read as follow: if you think that effect size of the experiment will be 1.8% and you want to constraint type II error by 20%, then the minimum number of observations in each group must be at least 25 000.

- for type I error (α)

split_rate	(5000, 5000)	(10000, 10000)	(15000, 15000)	(20000, 20000)	(25000, 25000)	(30000, 30000)
metric						
check_rub_campaign	0.0	0.0	0.0	0.0	0.0	0.0

1.6 Auto A/B

1.6.1 ABTest

1.6.2 VarianceReduction

1.6.3 Graphics

1.6.4 Params

1.7 Splitter

1.7.1 Split Builder

1.7.2 Params

1.8 Resplitter

1.8.1 Resplit Builder

1.8.2 Params

1.9 MDE Researcher

1.9.1 Abstract MDE Experiment

1.9.2 Experiment Structures

1.9.3 MDE Research Builder

1.9.4 Multiple Split Builder

1.9.5 Params

CHAPTER

TWO

EXAMPLES

For more details, see the examples.

CHAPTER

THREE

COMMUNICATION

Developers and authors:

- Vadim Glukhov
- Egor Shishkovets
- Dmitry Zabavin