

★ Member-only story

A/B testing: A step-by-step guide in Python

From experimental design to hypothesis testing



Renato Fillinich · [Follow](#)

Published in Towards Data Science

9 min read · May 28, 2020

Listen

Share

More

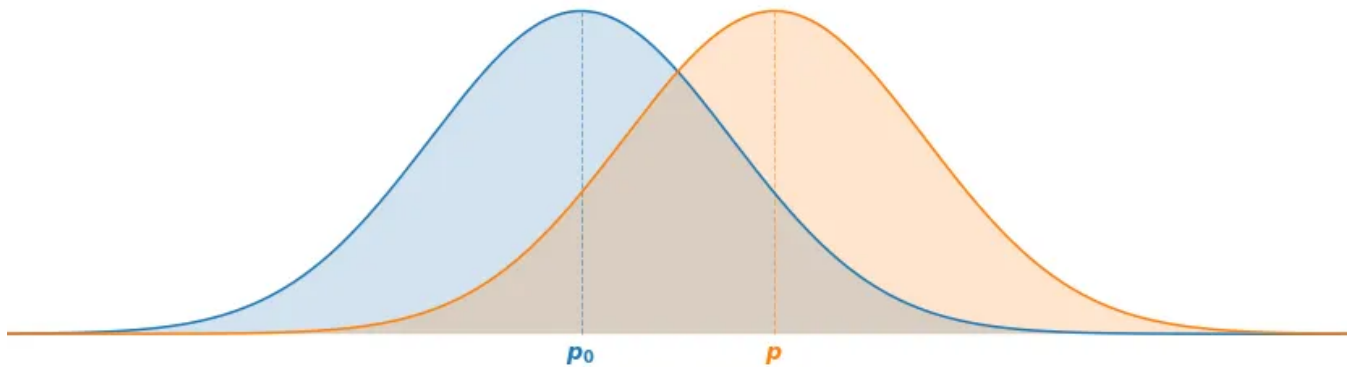


Image by author

In this article we'll go over the process of analysing an A/B experiment, from formulating a hypothesis, testing it, and finally interpreting results. For our data, we'll use a [dataset from Kaggle](#) which contains the results of an A/B test on what seems to be 2 different designs of a website page (old_page vs. new_page). If you want to follow along with the code I used, feel free to [download the jupyter notebook](#) at my GitHub page.

Here's what we'll do:

1. [Designing our experiment](#)
2. [Collecting and preparing the data](#)

3. Visualising the results

4. Testing the hypothesis

5. Drawing conclusions

To make it a bit more realistic, here's a potential **scenario** for our study:

*Let's imagine you work on the product team at a medium-sized **online e-commerce business**. The UX designer worked really hard on a new version of the product page, with the hope that it will lead to a higher conversion rate. The product manager (PM) told you that the **current conversion rate** is about 13% on average throughout the year, and that the team would be happy with an **increase of 2%**, meaning that the new design will be considered a success if it raises the conversion rate to 15%.*

Before rolling out the change, the team would be more comfortable testing it on a small number of users to see how it performs, so you suggest running an A/B test on a subset of your user base users.

1. Designing our experiment

Formulating a hypothesis

First things first, we want to make sure we formulate a hypothesis at the start of our project. This will make sure our interpretation of the results is correct as well as rigorous.

Given we don't know if the new design will perform better or worse (or the same?) as our current design, we'll choose a **two-tailed test**:

$$H_0: p = p_0$$

$$H_a: p \neq p_0$$

where p and p_0 stand for the conversion rate of the new and old design, respectively. We'll also set a **confidence level of 95%**:

$$\alpha = 0.05$$

The α value is a threshold we set, by which we say “if the probability of observing a result as extreme or more (*p-value*) is lower than α , then we reject the Null hypothesis”. Since our $\alpha=0.05$ (indicating 5% probability), our confidence ($1 - \alpha$) is 95%.

Don't worry if you are not familiar with the above, all this really means is that whatever conversion rate we observe for our new design in our test, we want to be 95% confident it is statistically different from the conversion rate of our old design, before we decide to reject the Null hypothesis H_0 .

Choosing the variables

For our test we'll need **two groups**:

- A `control` group - They'll be shown the old design
- A `treatment` (or experimental) group - They'll be shown the new design

This will be our *Independent Variable*. The reason we have two groups even though we know the baseline conversion rate is that we want to control for other variables that could have an effect on our results, such as seasonality: by having a `control` group we can directly compare their results to the `treatment` group, because the only systematic difference between the groups is the design of the product page, and we can therefore attribute any differences in results to the designs.

For our *Dependent Variable* (i.e. what we are trying to measure), we are interested in capturing the `conversion rate`. A way we can code this is by each user session with a binary variable:

- `0` - The user did not buy the product during this user session
- `1` - The user bought the product during this user session

This way, we can easily calculate the mean for each group to get the conversion rate of each design.

Choosing a sample size

It is important to note that since we won't test the whole user base (our population), the conversion rates that we'll get will inevitably be only *estimates* of the true rates.

The number of people (or user sessions) we decide to capture in each group will have an effect on the precision of our estimated conversion rates: **the larger the**

sample size, the more precise our estimates (i.e. the smaller our confidence intervals), **the higher the chance to detect a difference** in the two groups, if present.

On the other hand, the larger our sample gets, the more expensive (and impractical) our study becomes.

So how many people should we have in each group?

The sample size we need is estimated through something called Power analysis, and it depends on a few factors:

- **Power of the test** ($1 - \beta$) — This represents the probability of finding a statistical difference between the groups in our test when a difference is actually present. This is usually set at 0.8 by convention (here's more info on statistical power, if you are curious)
- **Alpha value** (α) — The critical value we set earlier to 0.05
- **Effect size** — How big of a difference we expect there to be between the conversion rates

Since our team would be happy with a difference of 2%, we can use 13% and 15% to calculate the effect size we expect.

Luckily, **Python takes care of all these calculations for us:**

```
# Packages imports
import numpy as np
import pandas as pd
import scipy.stats as stats
import statsmodels.stats.api as sms
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from math import ceil

%matplotlib inline

# Some plot styling preferences
plt.style.use('seaborn-whitegrid')
font = {'family' : 'Helvetica',
        'weight' : 'bold',
        'size'   : 14}

mpl.rc('font', **font)
```

```
effect_size = sms.proportion_effectsize(0.13, 0.15)    # Calculating
effect size based on our expected rates

required_n = sms.NormalIndPower().solve_power(
    effect_size,
    power=0.8,
    alpha=0.05,
    ratio=1
)                                                       # Calculating
sample size needed

required_n = ceil(required_n)                          # Rounding up
to next whole number

print(required_n)
```

4720

We'd need at least 4720 observations for each group.

Having set the `power` parameter to 0.8 in practice means that if there exists an actual difference in conversion rate between our designs, assuming the difference is the one we estimated (13% vs. 15%), we have about 80% chance to detect it as statistically significant in our test with the sample size we calculated.

2. Collecting and preparing the data

Great stuff! So now that we have our required sample size, we need to collect the data. Usually at this point you would work with your team to set up the experiment, likely with the help of the Engineering team, and make sure that you collect enough data based on the sample size needed.

However, since we'll use a dataset that we found online, in order to simulate this situation we'll:

1. Download the [dataset from Kaggle](#)
2. Read the data into a pandas DataFrame
3. Check and clean the data as needed
4. Randomly sample `n=4720` rows from the DataFrame for each group *

***Note:** Normally, we would not need to perform step 4, this is just for the sake of the exercise

Since I already downloaded the dataset, I'll go straight to number 2.

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

df.head()
```

	user_id	timestamp	group	landing_page	converted
0	851104	2017-01-21 22:11:48.556739	control	old_page	0
1	804228	2017-01-12 08:01:45.159739	control	old_page	0
2	661590	2017-01-11 16:55:06.154213	treatment	new_page	0
3	853541	2017-01-08 18:28:03.143765	treatment	new_page	0
4	864975	2017-01-21 01:52:26.210827	control	old_page	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 294478 entries, 0 to 294477
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   user_id         294478 non-null  int64
1   timestamp       294478 non-null  object
2   group           294478 non-null  object
3   landing_page    294478 non-null  object
4   converted       294478 non-null  int64
dtypes: int64(2), object(3)
memory usage: 11.2+ MB
```

```
# To make sure all the control group are seeing the old page and
viceversa
```

```
pd.crosstab(df['group'], df['landing_page'])
```

	new_page	old_page
group		
control	1928	145274
treatment	145311	1965

There are **294478 rows** in the DataFrame, each representing a user session, as well as **5 columns** :

- `user_id` - The user ID of each session
- `timestamp` - Timestamp for the session
- `group` - Which group the user was assigned to for that session { `control`, `treatment` }
- `landing_page` - Which design each user saw on that session { `old_page`, `new_page` }
- `converted` - Whether the session ended in a conversion or not (binary, `0` =not converted, `1` =converted)

We'll actually only use the `group` and `converted` columns for the analysis.

Before we go ahead and sample the data to get our subset, let's make sure there are no users that have been sampled multiple times.

```
session_counts = df['user_id'].value_counts(ascending=False)
multi_users = session_counts[session_counts > 1].count()

print(f'There are {multi_users} users that appear multiple times in
the dataset')
```

There are 3894 users that appear multiple times in the dataset

There are, in fact, 3894 users that appear more than once. Since the number is pretty low, we'll go ahead and remove them from the DataFrame to avoid sampling the same users twice.

```
users_to_drop = session_counts[session_counts > 1].index

df = df[~df['user_id'].isin(users_to_drop)]
print(f'The updated dataset now has {df.shape[0]} entries')
```

The updated dataset now has 286690 entries

Sampling

Now that our DataFrame is nice and clean, we can proceed and sample $n=4720$ entries for each of the groups. We can use pandas' `DataFrame.sample()` method to do this, which will perform Simple Random Sampling for us.

Note: I've set `random_state=22` so that the results are reproducible if you feel like following on your own Notebook: just use `random_state=22` in your function and you should get the same sample as I did.

```
control_sample = df[df['group'] == 'control'].sample(n=required_n,
random_state=22)
treatment_sample = df[df['group'] ==
'treatment'].sample(n=required_n, random_state=22)

ab_test = pd.concat([control_sample, treatment_sample], axis=0)
ab_test.reset_index(drop=True, inplace=True)

ab_test
```

	user_id	timestamp	group	landing_page	converted
0	763854	2017-01-21 03:43:17.188315	control	old_page	0
1	690555	2017-01-18 06:38:13.079449	control	old_page	0
2	861520	2017-01-06 21:13:40.044766	control	old_page	0
3	630778	2017-01-05 16:42:36.995204	control	old_page	0
4	656634	2017-01-04 15:31:21.676130	control	old_page	0
...
9435	908512	2017-01-14 22:02:29.922674	treatment	new_page	0
9436	873211	2017-01-05 00:57:16.167151	treatment	new_page	0
9437	631276	2017-01-20 18:56:58.167809	treatment	new_page	0
9438	662301	2017-01-03 08:10:57.768806	treatment	new_page	0
9439	944623	2017-01-19 10:56:01.648653	treatment	new_page	1

9440 rows × 5 columns

```
ab_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9440 entries, 0 to 9439
Data columns (total 5 columns):
```



```
# Column Non-Null Count Dtype
---  ---
0 user_id 9440 non-null int64
1 timestamp 9440 non-null object
2 group 9440 non-null object
3 landing_page 9440 non-null object
4 converted 9440 non-null int64
dtypes: int64(2), object(3)
memory usage: 368.9+ KB
```

```
ab_test['group'].value_counts()
```

```
control 4720
treatment 4720
Name: group, dtype: int64
```

Great, looks like everything went as planned, and we are now ready to analyse our results.

3. Visualising the results

The first thing we can do is to calculate some **basic statistics** to get an idea of what our samples look like.

```
conversion_rates = ab_test.groupby('group')['converted']

std_p = lambda x: np.std(x, ddof=0)           # Std. deviation of
the proportion
se_p = lambda x: stats.sem(x, ddof=0)         # Std. error of the
proportion (std / sqrt(n))

conversion_rates = conversion_rates.agg([np.mean, std_p, se_p])
conversion_rates.columns = ['conversion_rate', 'std_deviation',
'std_error']

conversion_rates.style.format('{:.3f}')
```

	conversion_rate	std_deviation	std_error
group			
control	0.123	0.329	0.005
treatment	0.126	0.331	0.005

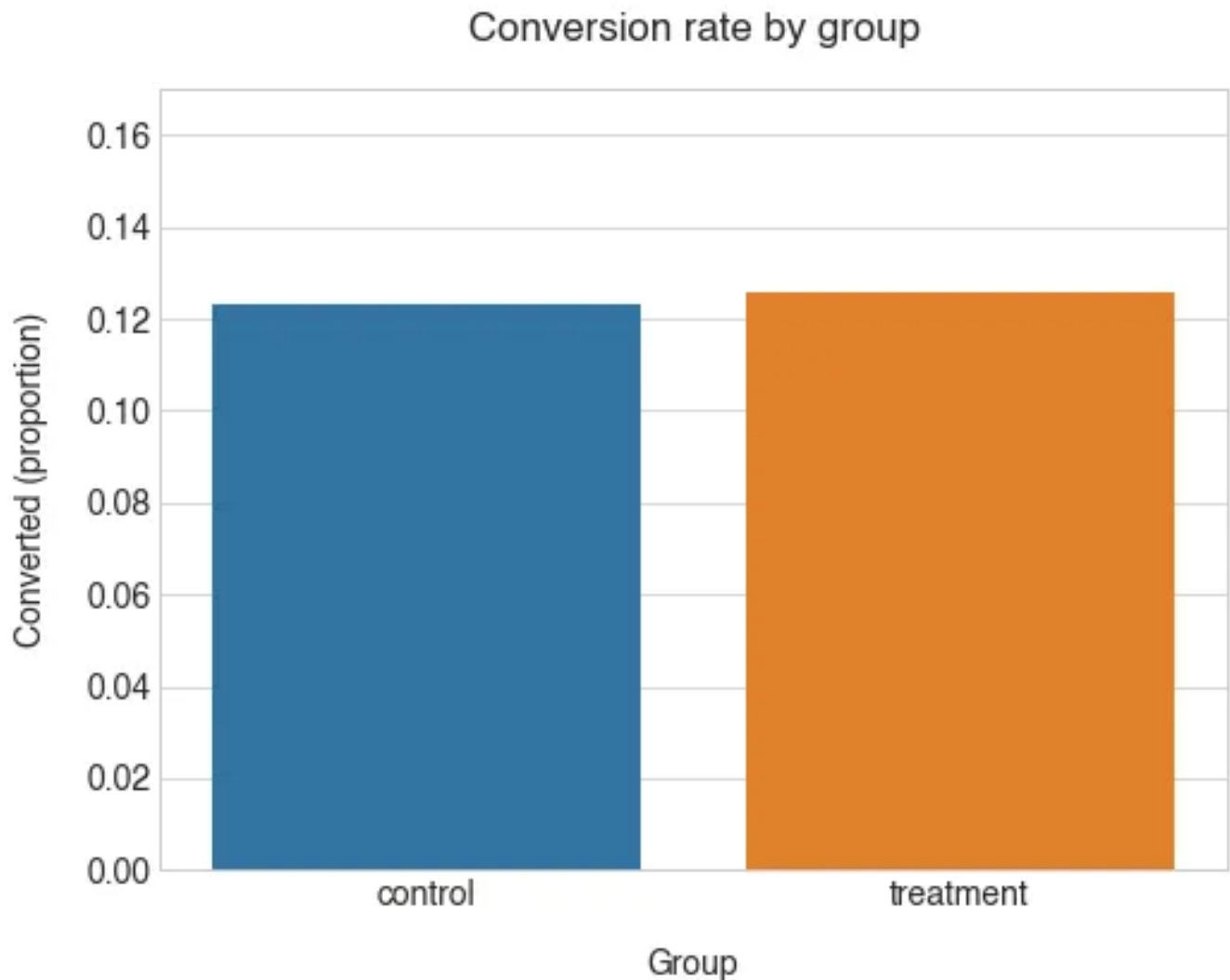
Judging by the stats above, it does look like **our two designs performed very similarly**, with our new design performing slightly better, approx. **12.3% vs. 12.6% conversion rate**.

Plotting the data will make these results easier to grasp:

```
plt.figure(figsize=(8,6))

sns.barplot(x=ab_test['group'], y=ab_test['converted'], ci=False)

plt.ylim(0, 0.17)
plt.title('Conversion rate by group', pad=20)
plt.xlabel('Group', labelpad=15)
plt.ylabel('Converted (proportion)', labelpad=15);
```



The conversion rates for our groups are indeed very close. Also note that the conversion rate of the `control` group is lower than what we would have expected given what we knew about our avg. conversion rate (12.3% vs. 13%). This goes to show that there is some variation in results when sampling from a population.

So... the `treatment` group's value is higher. Is this difference *statistically significant*?

4. Testing the hypothesis

The last step of our analysis is testing our hypothesis. Since we have a very large sample, we can use the normal approximation for calculating our p -value (i.e. z-test).

Again, Python makes all the calculations very easy. We can use the `statsmodels.stats.proportion` module to get the p -value and confidence intervals:

```

from statsmodels.stats.proportion import proportions_ztest,
proportion_confint

control_results = ab_test[ab_test['group'] == 'control']
['converted']
treatment_results = ab_test[ab_test['group'] == 'treatment']
['converted']

n_con = control_results.count()
n_treat = treatment_results.count()
successes = [control_results.sum(), treatment_results.sum()]
nobs = [n_con, n_treat]

z_stat, pval = proportions_ztest(successes, nobs=nobs)
(lower_con, lower_treat), (upper_con, upper_treat) =
proportion_confint(successes, nobs=nobs, alpha=0.05)

print(f'z statistic: {z_stat:.2f}')
print(f'p-value: {pval:.3f}')
print(f'ci 95% for control group: [{lower_con:.3f},
{upper_con:.3f}]')
print(f'ci 95% for treatment group: [{lower_treat:.3f},
{upper_treat:.3f}]')

```

Open in app ↗



5. Drawing conclusions

Since our $p\text{-value}=0.732$ is way above our $\alpha=0.05$ threshold, we cannot reject the Null hypothesis H_0 , which means that **our new design did not perform significantly different (let alone better) than our old one** :(

Additionally, if we look at the confidence interval for the `treatment` group ([0.116, 0.135], or 11.6-13.5%) we notice that:

1. It includes our baseline value of 13% conversion rate
2. It does not include our target value of 15% (the 2% uplift we were aiming for)

What this means is that it is more likely that the true conversion rate of the new design is similar to our baseline, rather than the 15% target we had hoped for. This is further proof that our new design is not likely to be an improvement on our old design, and that unfortunately we are back to the drawing board!

Did you like my story? Please let me know!

And please feel free to [download the jupyter notebook](#) at my [GitHub page](#).

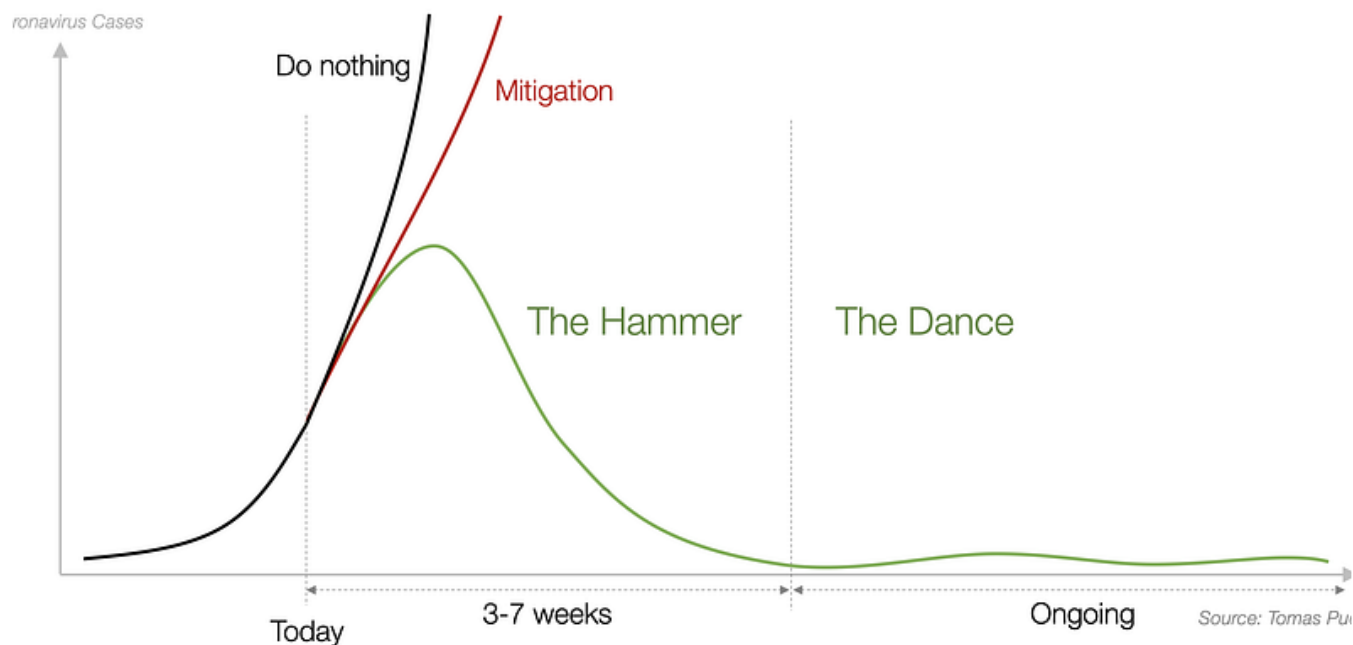
[User Research](#)[Ab Testing](#)[UX](#)[Python](#)[Hypothesis Testing](#)[Follow](#)

Written by Renato Fillinich

291 Followers · Writer for Towards Data Science

UX Researcher @ Google, Data science and math enthusiast

More from Renato Fillinich and Towards Data Science



Renato Fillinich

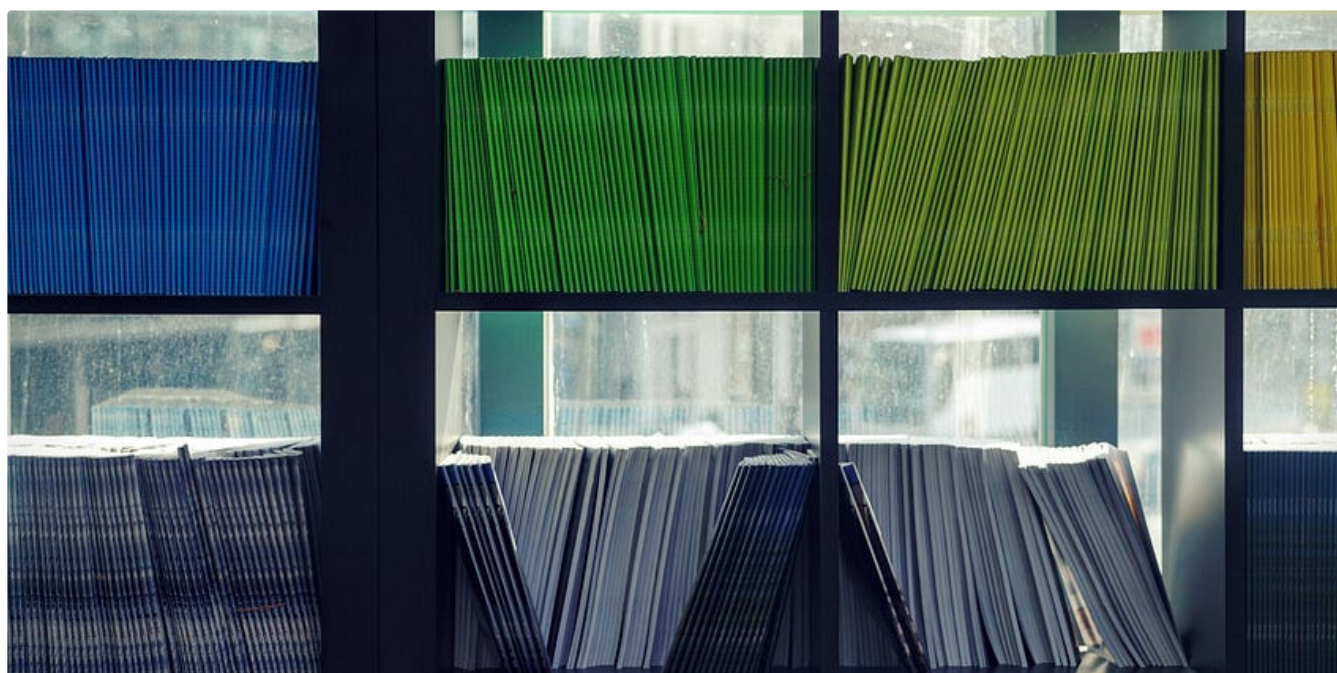
Coronavirus: Il Martello e la Danza

Come potranno essere i prossimi 18 mesi se i leader riusciranno a guadagnare tempo

28 min read · Mar 25, 2020



4



Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company's Docs into a Searchable Database with OpenAI

And how you can do the same with your docs

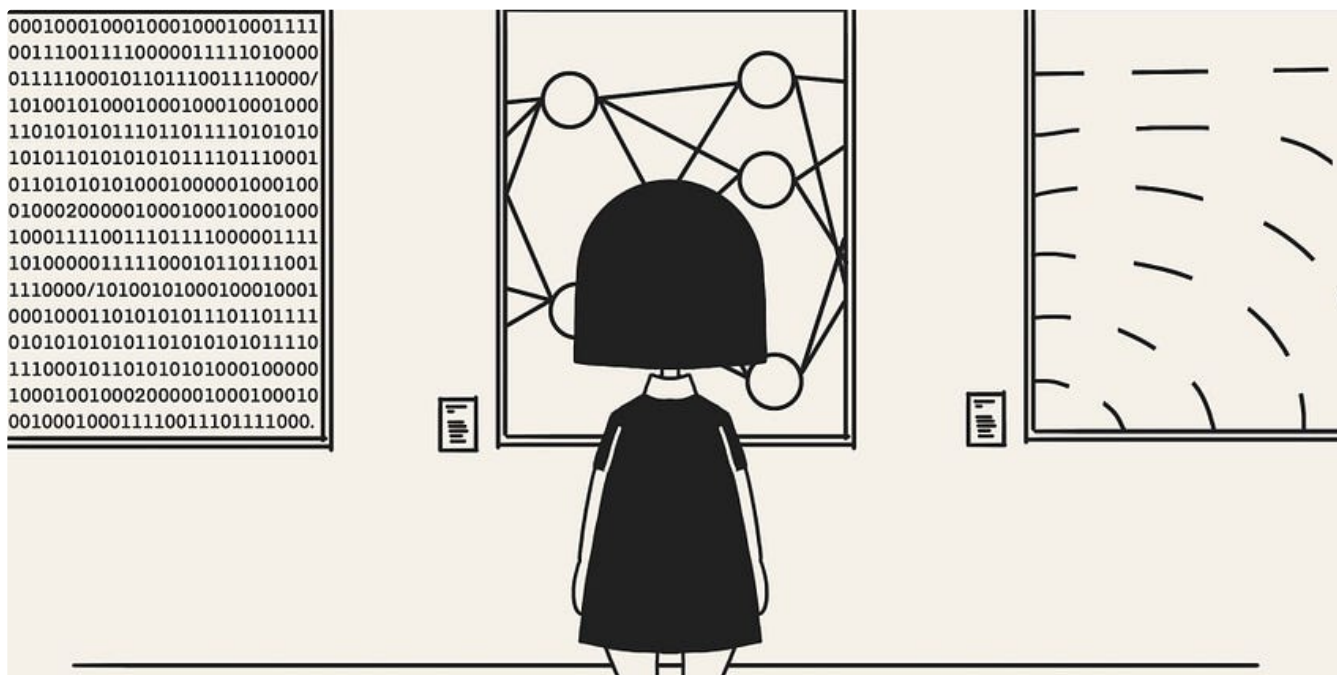
15 min read · Apr 25



3.7K



48



Leonie Monigatti in Towards Data Science

10 Exciting Project Ideas Using Large Language Models (LLMs) for Your Portfolio

Learn how to build apps and showcase your skills with large language models (LLMs). Get started today!



· 11 min read · May 15



1.2K



7






 Leonie Monigatti in Towards Data Science

Getting Started with LangChain: A Beginner’s Guide to Building LLM-Powered Applications

A LangChain tutorial to build anything with large language models in Python

🌟 · 12 min read · Apr 25

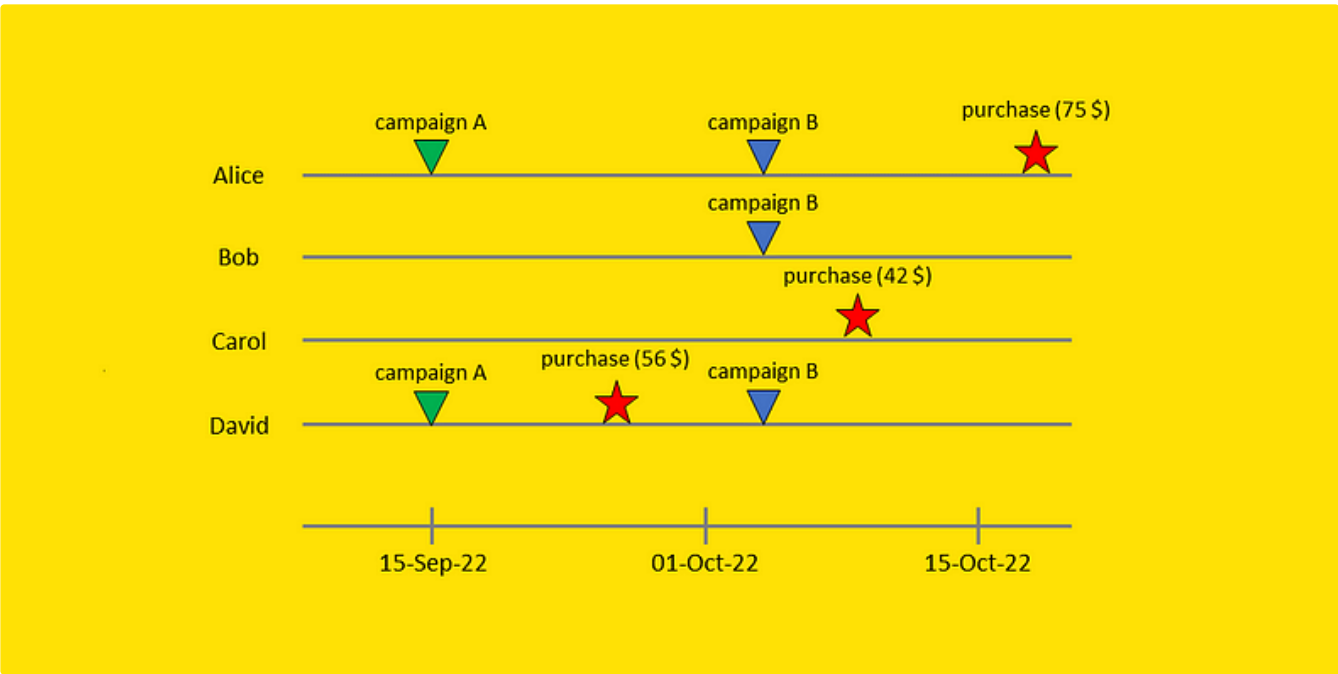
 2.8K  19

See all from Renato Fillinich

See all from Towards Data Science

Recommended from Medium



 Samuele Mazzanti in Towards Data Science

Using Causal ML Instead of A/B Testing

In complex environments, Causal ML is a powerful tool because it is more flexible than A/B Testing, and it doesn't require strong...

★ · 9 min read · Nov 29, 2022


 927

 15







 Pararawendy Indarjo in Bukalapak Data

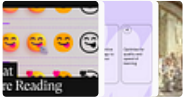
Understanding Proportion Z-test

Why using Z (normal) distribution to compare two proportions?

★ · 6 min read · Jan 9

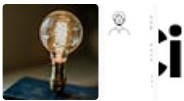


Lists



Stories to Help You Grow as a Designer

11 stories · 70 saves



Good Product Thinking

11 stories · 117 saves



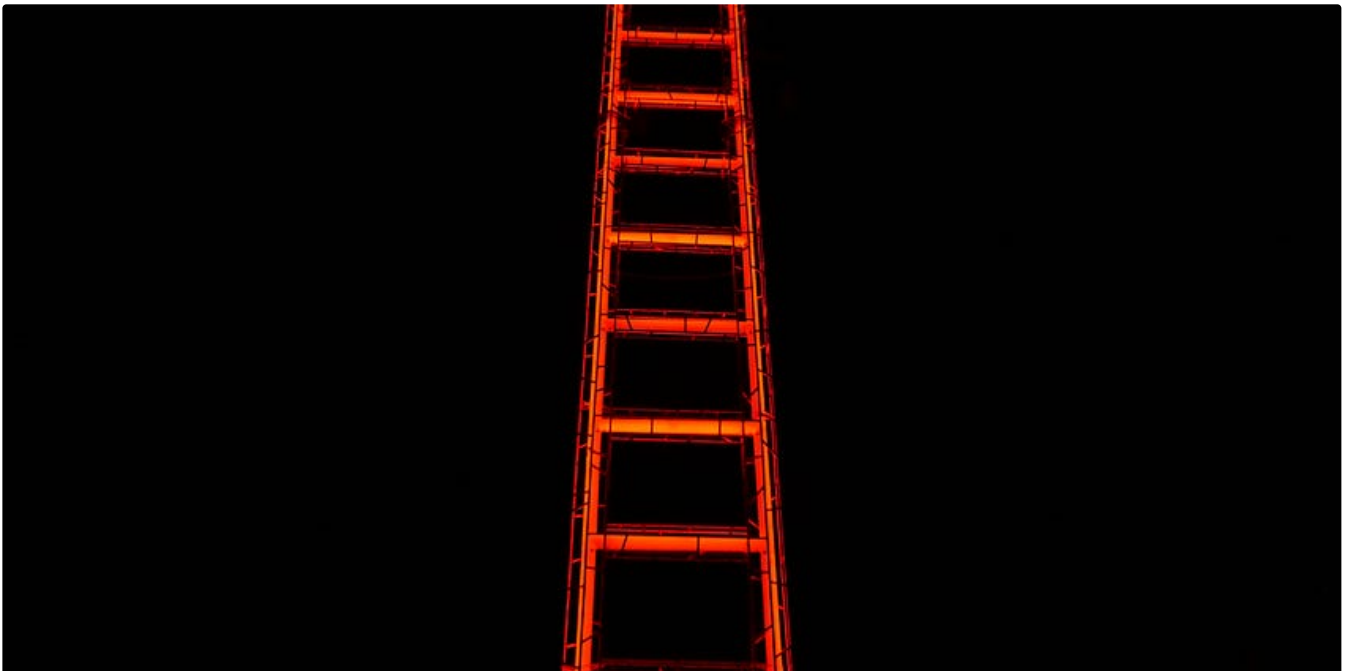
A Guide to OKRs – Objectives and Key Results

10 stories · 60 saves



Staff Picks

345 stories · 106 saves



Leihua Ye, PhD

Why Data Scientists Should Learn Causal Inference

Climb up the ladder of causation

★ · 7 min read · Jul 5, 2022



803



9



Vered Shapovalov in Riskified Tech

The Significance of A/B Testing and Power Analysis in Fraud Detection

Model replacement evaluation in a multi-model and dynamic environment

8 min read · Feb 22



395





Santiago Rodrigues Manica in Epidence

Sample Size Matters

How to calculate sample size using R and Python

★ · 4 min read · Feb 5



7



Rahul Sikder

Netflix's Engagement Metric Has Dropped By 20%. Root Cause Analysis

Before we begin, allow me to introduce myself briefly. Over the past 4–5 months, I have been immersing myself in the field of Product...

9 min read · Apr 9

 374

 5



See more recommendations