Analyze A/B Test Results

Table of Contents

Introduction

The goal of the project is to understand the results of an A/B test run by an e-commerce website and to help the company understand if they should implement the new page, keep the old page, or perhaps run the experiment longer to make their decision.

I used Python:

- to test the probability of conversion for the new page and old page users (treatment group and control group)
- to run A/B tests
- confirmed the results of A/B tests by regression approach

Part I - Probability

In [1]:
```
#import Python libraries
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
%matplotlib inline
#setting the seed
random.seed(42)
```

In [2]:
```
#read the data in the csv file and store it in df
df = pd.read_csv('ab_data.csv')
df.head()
```

Out[2]:

	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

	user_id	timestamp	group	landing_page	converted
4	864975	2017-01-21 01:52:26.210827	control	old_page	1

In [5]:
```
#number of rows in the dataset
df.shape
```

Out[5]:
```
(294478, 5)
```

In [6]:
```
#he number of unique users in the dataset.
df.nunique()
```

Out[6]:
```
user_id         290584
timestamp       294478
group                2
landing_page         2
converted            2
dtype: int64
```

In [7]:
```
#The proportion of users converted
#two ways of counting the proportion of users converted:
#df['converted'].sum() / len(df)
df.converted.mean()
```

Out[7]:
```
0.11965919355605512
```

In [8]:
```
# The number of times the new_page and treatment don't line up
#we can count it in two ways.
#First solution: is to use two queries, each giving us a a combination when new_page and treatment don't line up
#Second solution: is by using one query which will group df to give us all the combinations, including the demanded one.
#First solution:
#combination 1
df.query ('group == "treatment" and landing_page != "new_page"')
```

Out[8]:

	user_id	timestamp	group	landing_page	converted
308	857184	2017-01-20 07:34:59.832626	treatment	old_page	0
327	686623	2017-01-09 14:26:40.734775	treatment	old_page	0
357	856078	2017-01-12 12:29:30.354835	treatment	old_page	0
685	666385	2017-01-23 08:11:54.823806	treatment	old_page	0
713	748761	2017-01-10 15:47:44.445196	treatment	old_page	0
...	...	...	...	...	...
293773	688144	2017-01-16 20:34:50.450528	treatment	old_page	1
293817	876037	2017-01-17 16:15:08.957152	treatment	old_page	1

	user_id	timestamp	group	landing_page	converted
293917	738357	2017-01-05 15:37:55.729133	treatment	old_page	0
294014	813406	2017-01-09 06:25:33.223301	treatment	old_page	0
294252	892498	2017-01-22 01:11:10.463211	treatment	old_page	0

1965 rows × 5 columns

In [9]:
```
#First soultion
#combination2
df.query ('group == "control" and landing_page != "old_page"')
#summed result from both combinations = 1965 + 1928 = 3893 = number of times when treatment and ne_page don't match
```

Out[9]:

	user_id	timestamp	group	landing_page	converted
22	767017	2017-01-12 22:58:14.991443	control	new_page	0
240	733976	2017-01-11 15:11:16.407599	control	new_page	0
490	808613	2017-01-10 21:44:01.292755	control	new_page	0
846	637639	2017-01-11 23:09:52.682329	control	new_page	1
850	793580	2017-01-08 03:25:33.723712	control	new_page	1
...	...	...	...	...	...
293894	741581	2017-01-09 20:49:03.391764	control	new_page	0
293996	942612	2017-01-08 13:52:28.182648	control	new_page	0
294200	928506	2017-01-13 21:32:10.491309	control	new_page	0
294253	886135	2017-01-06 12:49:20.509403	control	new_page	0
294331	689637	2017-01-13 11:34:28.339532	control	new_page	0

1928 rows × 5 columns

In [10]:
```
#Second solution:
# 1928 + 1965 = 3893
df.groupby(['group', 'landing_page'])['landing_page'].count()
```

Out[10]:
```
group      landing_page
control    new_page          1928
           old_page        145274
treatment  new_page        145311
           old_page          1965
Name: landing_page, dtype: int64
```

In [11]:
```
df.info() #check for null values
```

```
<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
```

In [12]:
```
# Remove the inaccurate rows, and store the result in a new dataframe df2
#1st solution:
df2 = df.drop(df[((df.group == 'control') & (df.landing_page == 'new_page')) | \
                 ((df.group == 'treatment') & (df.landing_page == 'old_page'))].index)
```

In [13]:
```
#2nd solution:
# Remove the inaccurate rows, and store the result in a new dataframe df2
# first, create a new df with combination: 'treatment' and 'new_page'
#df2_t = df.query('group == "treatment" and landing_page == "new_page"')
```

In [14]:
```
#second, create a new df with combination: 'control' and 'old_page'
#df2_c = df.query('group == "control" and landing_page == "old_page"')
```

In [15]:
```
#third, merge both new dfs into one properly aligned df2
#df2 = df2_t.merge(df2_c, how = 'outer')
```

In [16]:
```
#check the first rows of the new df2
df2.head()
```

Out[16]:

	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

In [17]:
```
#check number of rows and columns in d2
df2.shape
```

Out[17]:
```
(290585, 5)
```

In [18]:
```
df2.describe()
```

Out[18]:

	user_id	converted
count	290585.000000	290585.000000
mean	788004.825246	0.119597
std	91224.582639	0.324490
min	630000.000000	0.000000
25%	709035.000000	0.000000
50%	787995.000000	0.000000
75%	866956.000000	0.000000
max	945999.000000	1.000000

In [19]:
```
# Double Check all of the correct rows were removed - this should be 0
df2[((df2['group'] == 'treatment') == (df2['landing_page'] == 'new_page')) == False].shape[0]
```

Out[19]:
```
0
```

In [20]:
```
#number of unique user_ids are in df2
df2.user_id.nunique()
```

Out[20]:
```
290584
```

In [21]:
```
#check if there is a duplicated user_id in df2
sum(df2.user_id.duplicated())
```

Out[21]:
```
1
```

In [22]:
```
# display the row information for the repeat 'user_id'
df2[df2.duplicated(['user_id'], keep=False)]
```

Out[22]:

	user_id	timestamp	group	landing_page	converted
1899	773192	2017-01-09 05:37:58.781806	treatment	new_page	0
2893	773192	2017-01-14 02:55:59.590927	treatment	new_page	0

In [113…
```
# Remove one of the rows with a duplicate user_id
# The dataframe.drop_duplicates() may not work in this case because the rows with duplicate user_id are not entirely identical.
```

```
# the solution found on stackoverflow:
# https://stackoverflow.com/questions/13035764/remove-pandas-rows-with-duplicate-indices
df2 = df2[~df2.user_id.duplicated(keep = 'first')]

#second solution that also works:
#df2.drop_duplicates(['user_id'], inplace=True)


## Check again if the row with a duplicate user_id is deleted or not
#df.shape
sum(df2.user_id.duplicated())
```

Out[113… 0

In [23]:
```
#the probability of an individual converting regardless of the page they receive
df2.converted.mean()
```

Out[23]: 0.11959667567149027

In [24]:
```
#Given that an individual was in the control group, what is the probability they converted
df2_control_group = df2.query('group == "control"')
df2_control_group.converted.mean()
```

Out[24]: 0.1203863045004612

In [25]:
```
#Given that an individual was in the treatment group, what is the probability they converted
df2_treatment_group = df2.query('group == "treatment"')
df2_treatment_group.converted.mean()
```

Out[25]: 0.11880724790277405

In [26]:
```
# Calculate the actual difference (obs_diff) between the conversion rates for the two groups.
obs_diff = (df2_control_group.converted.mean()) - (df2_treatment_group.converted.mean())
print(obs_diff)
```

```
0.0015790565976871451
```

In [27]:
```
# the probability that an individual received the new page
# number of users (indexes) in treatment group divided by total number of users (indexes)
len(df2_treatment_group.index)/len(df2.index)
```

Out[27]: 0.5000636646764286

-The treatment group has a conversion rate of 11.88%, while the control group has a conversion rate of 12.04%. -The observed difference in conversion rate is small: 0.16%. -The probability of conversion in the treatment group is slightly lower than in control group. Based on these results, we can assume that the treatment group will not lead to more conversions than the control group. It would need more investigation to check if the result is unbiased.

Part II - A/B Test

Because of the time stamp associated with each event, I could technically run a hypothesis test continuously as each observation was observed.

However, then the hard question is should we stop as soon as one page is considered significantly better than another or does it need to happen consistently for a certain amount of time? How long should I run to render a decision that neither page is better than another?

1. For this project, I consider that I need to make the decision just based on all the data provided.

If we assume that the old page is better unless the new page proves to be definitely better at a Type I error rate of 5%, then: -null hypothesis is that the converted rate of the old page is greater or equal to the converted rate of the new page p_old >= p_new

-alternative hypothesis is that the converted rate of the new age is greater than the converted rate of the old page p_new > p_old

In [28]:
```
#conversion rate for  p_new  under the null hypothesis
p_new = df2.converted.mean()
print(p_new)
```

```
0.11959667567149027
```

In [29]:
```
#conversion rate for p_old under the null hypothesis
p_old = df2.converted.mean()
print(p_old)
```

```
0.11959667567149027
```

In [30]:
```
#n_new the number of individuals in the treatment group
n_new = len(df2_treatment_group.index)
print(n_new)
```

```
145311
```

In [31]:
```
# n_old the number of individuals in the control group
n_old = len(df2_control_group.index)
print(n_old)
```

```
145274
```

In [32]:
```
#Simulate  n_new transactions with a convert rate of  p_new under the null. Store these  n_new 1's
# and 0's in new_page_converted.
new_page_converted = np.random.choice([1, 0], size=len(df2_treatment_group.index), p=[df2.converted.mean(), (1-(df2.converted.mean()))])
```

f. Simulate n_{old} transactions with a convert rate of p_{old} under the null. Store these n_{old} 1's and 0's in **old_page_converted**.

In [33]:
```
# Simulate a Sample for the control Group (simulate n_old transactions with a convert rate of  p_old under the null.
#Store these  n_old  1's and 0's in old_page_converted.)
old_page_converted = np.random.choice([1, 0], size=len(df2_control_group.index), p=[df2.converted.mean(), (1-(df2.converted.mean()))])
```

In [34]:
```
 #the difference in the "converted" probability  (p_new - p_old)
#for the above simulated samples
samples_p_diff = new_page_converted.mean() - old_page_converted.mean()
samples_p_diff
```

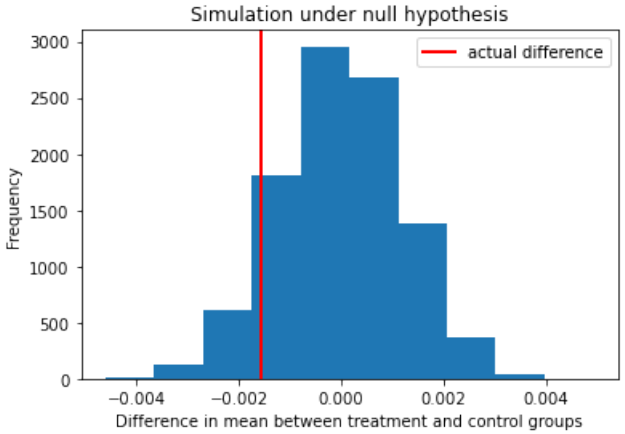
Out[34]:
```
0.0021033485955119363
```

h. Simulate 10,000 p_{new} - p_{old} values using this same process similarly to the one you calculated in parts **a. through g.** above. Store all 10,000 values in a numpy array called **p_diffs**.

In [35]:
```
#Simulate 10,000  p_new -  p_old values using this same process similarly to the one you calculated
#in parts a. through g. above. Store all 10,000 values in a numpy array called p_diffs.
p_diffs = []
for _ in range(10000):
    new_page_converted = np.random.choice([1, 0], size=len(df2_treatment_group.index), p=[df2.converted.mean(), (1-(df2.converted.mean()))])
    old_page_converted = np.random.choice([1, 0], size=len(df2_control_group.index), p=[df2.converted.mean(), (1-(df2.converted.mean()))])
    p_diffs.append(new_page_converted.mean() - old_page_converted.mean())
```

In [36]:
```
#Plot a histogram of the p_diffs.
obs_diff = (df2_treatment_group.converted.mean()) - (df2_control_group.converted.mean())
plt.hist(p_diffs);
plt.title('Simulation under null hypothesis')
plt.xlabel('Difference in mean between treatment and control groups')
plt.ylabel('Frequency')
plt.axvline(x=obs_diff, color='r', linewidth=2, label='actual difference')
plt.legend()
```

Out[36]:
```
<matplotlib.legend.Legend at 0x21887200a90>
```



In [61]:
```
#proportion of the p_diffs are greater than the actual difference observed in ab_data.csv
p_value = (p_diffs > obs_diff).mean()
print(p_value)
#it worked in Udacity terminal ??? and gave the result: 0.0906
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_19976/2843395513.py in <module>
----> 1 p_value = (p_diffs > obs_diff).mean()
      2 print(p_value)

TypeError: '>' not supported between instances of 'list' and 'float'
```

Results: The proportion of the p_diffs that are greater than the actual difference observed in the df2 data is called the p-value.

The p-value is the probability of observing your statistic or - in other words - the probability of observing a difference as extreme as the one observed if the null hypothesis is true.

Our null hypothesis was that the difference in means would be equal or less than 0. Our alternative hypothesis was that the difference would be greater than 0. The p-value of 0.0906 is realtively large with the significance level set at 0.05 (or - in other words - with Type I error rate (0.05)). We do not have evidence to reject the null hypotesis.

(ad.Type I error it is also known as a "false positive," occurs in hypothesis testing when we reject the null hypothesis even though it is actually true. In other words, it is an error that happens when we conclude there is a significant effect or difference when, in reality, there is none).

In [64]:

```
#We could also use a built-in to achieve similar results. Though using the built-in might be easier to code,
#the above portions are a walkthrough of the ideas that are critical to correctly thinking about statistical
#significance.
#Calculate the number of conversions for each page, as well as the number of individuals who received each page.
#Let n_old and n_new refer the the number of rows associated with the old page and new pages, respectively.
import statsmodels.api as sm
import statsmodels.api as sm


# number of conversions with the old_page
convert_old = len(df2_control_group[df2_control_group['converted'] == 1])
print(convert_old)

# number of conversions with the new_page
convert_new = len(df2_treatment_group[df2_treatment_group['converted'] == 1])
print(convert_new)

# number of individuals who were shown the old_page
n_old = len(df2_control_group.index)
print(n_old)

# number of individuals who received new_page
n_new = len(df2_treatment_group.index)
print(n_new)
```

```
17489
17264
145274
145310
```

The syntax is:

proportions_ztest(count_array, nobs_array, alternative='larger') where,

count_array = represents the number of "converted" for each group nobs_array = represents the total number of observations (rows) in each group alternative = choose one of the values from ['two-sided', 'smaller', 'larger'] depending upon two-tailed, left-tailed, or right-tailed respectively.

It's a two-tailed if you defined $H1$ as $(pnew=pold)$. It's a left-tailed if you defined $H1$ as $(pnew<pold)$. It's a right-tailed if you defined $H1$ as $(pnew>pold)$.

The built-in function above will return the z_score, p_value.

About the two-sample z-test I have plotted a distribution p_diffs representing the difference in the "converted" probability $(p'new-p'old)$ for my two simulated samples 10,000 times.

Another way for comparing the mean of two independent and normal distribution is a two-sample z-test. We can perform the Z-test to calculate the Z_score, as shown in the equation below:

$$Zscore=(p'new-p'old)-(pnew-pold)\sigma 2newnnew+\sigma 2oldnold\sqrt{}$$

where,

p' is the "converted" success rate in the sample $pnew$ and $pold$ are the "converted" success rate for the two groups in the population. σnew and σnew are the standard deviation for the two groups in the population. $nnew$ and $nold$ represent the size of the two groups or samples (it's same in our case) Z-test is performed when the sample size is large, and the population variance is known. The z-score represents the distance between the two "converted" success rates in terms of the standard error.

Next step is to make a decision to reject or fail to reject the null hypothesis based on comparing these two values:

$Zscore$

$Z\alpha$ or $Z0.05$, also known as critical value at 95% confidence interval. $Z0.05$ is 1.645 for one-tailed tests, and 1.960 for two-tailed test. You can determine the $Z\alpha$ from the z-table manually. I need to decide if my hypothesis is either a two-tailed, left-tailed, or right-tailed test. Accordingly, reject OR fail to reject the null based on the comparison between $Zscore$ and $Z\alpha$. I determine whether or not the $Zscore$ lies in the "rejection region" in the distribution. In other words, a "rejection region" is an interval where the null hypothesis is rejected iff the $Zscore$ lies in that region.

For a right-tailed test, reject null if $Zscore$

> $Z\alpha$. For a left-tailed test, reject null if $Zscore < Z\alpha$.

Reference:

Example 9.1.2 on this page, courtesy www.stats.libretexts.org

In [65]:
```
#using stats.proportions_ztest to compute your test statistic and p-value
import statsmodels.api as sm
# ToDo: Complete the sm.stats.proportions_ztest() method arguments
z_score, p_value = sm.stats.proportions_ztest([convert_old, convert_new], [n_old, n_new], alternative = 'smaller')
print(z_score, p_value)
```

```
1.3109241984234394 0.9050583127590245
```

-Critical value at 95% confidence interval ($Z\alpha$) is 1.960 for two-tailed test (our test is a two tailed test as we are testing for the difference) what means that a z-score past -1.96 or 1.96 would be significant. The convertion rate of the new landing page is 1.3109 standard deviations from the convertion rate of the old landing page. 1.3109 is less than critical value of 1.96. so we failed to reject the null hypothesis. -The p-value was calculated for null hypothesis that the new page would convert more than the old page. The alternative was that the old page converted more than

or equal to the new page. The p-value of 0.9051 is much greater than a significance level of 0.05. so it supports the conclusion that we failed to reject the null hypothesis. The conclusion agrees with the findings from j.k.

Part III - A regression approach

1. In this final part, I will check the result acheived in the previous A/B test by by performing regression.

We shoud perform linear regression. "In statistics, linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables)[1]. Particularly, the case when there is only one explanatory variable is known as Simple Linear Regression." (https://medium.com/@alexandre.hsd/an-introduction-to-linear-regression-13527642f49#:~:text=In%20statistics%2C%20linear%20regression%20is,known%20as%20Simple%20Linear%20Regression.)

b. The goal is to use **statsmodels** to fit the regression model you specified in part **a.** to see if there is a significant difference in conversion based on which page a customer receives. However, I first need to create a column for the intercept, and create a dummy variable column for which page each user received. I will add an **intercept** column, as well as an **ab_page** column, which is 1 when an individual receives the **treatment** and 0 if **control**.

In [139…

```
df2['intercept'] = 1
df2['ab_page'] = pd.get_dummies(df2['group'])['treatment']
df2.head()
```

Out[139…

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

In [140…

```
#another way to do the same:
df2['intercept'] = 1
df2['ab_page'] = 0
ab_page_index = df2[df2['group'] == 'treatment'].index
df2.loc[ab_page_index, "ab_page"] = 1
df2.head()
```

Out[140…

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

	user_id	timestamp	group	landing_page	converted	intercept	ab_page
4	864975	2017-01-21 01:52:26.210827	control	old_page	1	1	0

c. I use **statsmodels** to import my regression model. I will instantiate the model, and fit the model using the two columns I created in part **b.** to predict whether or not an individual converts.

In [141…

```
lm = sm.OLS(df2['converted'], df2[['intercept', 'ab_page']])
results=lm.fit()
```

d. I will provide the summary of my model below, and use it as necessary to answer the following questions.

In [78]:

```
results.summary()
```

Out[78]:

OLS Regression Results

Dep. Variable:	converted	**R-squared:**	0.000
Model:	OLS	**Adj. R-squared:**	0.000
Method:	Least Squares	**F-statistic:**	1.719
Date:	Mon, 14 Aug 2023	**Prob (F-statistic):**	0.190
Time:	16:07:35	**Log-Likelihood:**	-85267.
No. Observations:	290584	**AIC:**	1.705e+05
Df Residuals:	290582	**BIC:**	1.706e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
intercept	0.1204	0.001	141.407	0.000	0.119	0.122
ab_page	-0.0016	0.001	-1.311	0.190	-0.004	0.001

Omnibus:	125553.456	**Durbin-Watson:**	2.000
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	414313.355
Skew:	2.345	**Prob(JB):**	0.00
Kurtosis:	6.497	**Cond. No.**	2.62

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

e. What is the p-value associated with **ab_page**? Why does it differ from the value you found in **Part II**?

1.The p-value associated with ab_page is 0.190 With a p-value of 0.190, and the significance level 0.05, since the p-value is greater than 0.05, we fail to reject the null hypothesis. This suggests that there is not enough evidence to conclude that the difference between the pages is statistically significant.

2.In Part II, the p-value was calculated where: -null hypothesis = new page would convert more than the old page, -alternative hypothesis = the old page converted more than or equal to the new page.

3.In Part III, I used variables and a linear model to calculate the p-value where: -the null hypothesis = the difference between the pages is equal to 0, -the alternative hypothesis = difference between the pages is greater or less than 0.

f. Other things that might influence whether or not an individual converts.

It is a good idea to add other factors as they might enrich the result. For example: -by adding the type of platform that users used to enter the page, might tell us if platform choice has an impact on conversion, -by adding the time of the daywhen users enter the page, we can check if the time has an influence on conversion. However, we should be careful while adding additional terms to our regression model, as some variables might affect others and disturb the result.

g. Now, I will go along with testing if the conversion rate changes for different pages, also I will add an effect based on which country a user lives. I will need to read in the **countries.csv** dataset and merge together my datasets on the approporiate rows.

I will check if country had an impact on conversion.

In [143…
```
df_countries = pd.read_csv('countries.csv')
df_new = df_countries.set_index('user_id').join(df2.set_index('user_id'), how='inner')
df_new.head()
```

Out[143…

user_id	country	timestamp	group	landing_page	converted	intercept	ab_page
834778	UK	2017-01-14 23:08:43.304998	control	old_page	0	1	0
928468	US	2017-01-23 14:44:16.387854	treatment	new_page	0	1	1
822059	UK	2017-01-16 14:04:14.719771	treatment	new_page	1	1	1
711597	UK	2017-01-22 03:14:24.763511	control	old_page	0	1	0
710616	UK	2017-01-16 13:14:44.000513	treatment	new_page	0	1	1

In [164…
```
### Create the necessary dummy variables
dummy_countries = pd.get_dummies(df_new['country'])
df3 = dummy_countries.join(df_new, how = 'inner')
df3.head()
```

Out[164…

user_id	CA	UK	US	country	timestamp	group	landing_page	converted	intercept	ab_page
834778	0	1	0	UK	2017-01-14 23:08:43.304998	control	old_page	0	1	0
928468	0	0	1	US	2017-01-23 14:44:16.387854	treatment	new_page	0	1	1

user_id	CA	UK	US	country	timestamp	group	landing_page	converted	intercept	ab_page
822059	0	1	0	UK	2017-01-16 14:04:14.719771	treatment	new_page	1	1	1
711597	0	1	0	UK	2017-01-22 03:14:24.763511	control	old_page	0	1	0
710616	0	1	0	UK	2017-01-16 13:14:44.000513	treatment	new_page	0	1	1

h. Though I have now looked at the individual factors of country and page on conversion, I would now like to look at an interaction between page and country to see if there significant effects on conversion. I will create the necessary additional columns, and fit the new model.

I will provide the summary results, and my conclusions based on the results.

In [152…

```
### Fit Linear Model And Obtain the Results
lm2 = sm.Logit(df3['converted'], df3[['intercept', 'UK', 'CA']])
results = lm2.fit()
results.summary()
# in case of getting AttributeError: 'LogitResults' object has no attribute 'chisq', try results.sumary2()
#soultions found on stackoverflow
```

```
Optimization terminated successfully.
         Current function value: 0.366116
         Iterations 6
```

Out[152…

Logit Regression Results

Dep. Variable:	converted	**No. Observations:**	290584
Model:	Logit	**Df Residuals:**	290581
Method:	MLE	**Df Model:**	2
Date:	Wed, 16 Aug 2023	**Pseudo R-squ.:**	1.521e-05
Time:	11:53:04	**Log-Likelihood:**	-1.0639e+05
converged:	True	**LL-Null:**	-1.0639e+05
Covariance Type:	nonrobust	**LLR p-value:**	0.1984

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-1.9967	0.007	-292.314	0.000	-2.010	-1.983
UK	0.0099	0.013	0.746	0.456	-0.016	0.036
CA	-0.0408	0.027	-1.518	0.129	-0.093	0.012

In [153…

```
#another way:
#lm = sm.OLS(df3['converted'], df3[['intercept', 'UK', 'CA']])
#results = lm.fit()
#results.summary()
#if you got an AttributeError: 'LogitResults' object has no attribute 'chisq' try this:
#results.summary2()
```

Out[153…

OLS Regression Results

Dep. Variable:	converted	**R-squared:**	0.000
Model:	OLS	**Adj. R-squared:**	0.000
Method:	Least Squares	**F-statistic:**	1.605
Date:	Wed, 16 Aug 2023	**Prob (F-statistic):**	0.201
Time:	12:10:12	**Log-Likelihood:**	-85267.
No. Observations:	290584	**AIC:**	1.705e+05
Df Residuals:	290581	**BIC:**	1.706e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
intercept	0.1195	0.001	166.244	0.000	0.118	0.121
UK	0.0010	0.001	0.746	0.455	-0.002	0.004
CA	-0.0042	0.003	-1.516	0.130	-0.010	0.001

Omnibus:	125552.384	**Durbin-Watson:**	1.996
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	414306.036
Skew:	2.345	**Prob(JB):**	0.00
Kurtosis:	6.497	**Cond. No.**	4.84

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

-The model above uses only the country as the explanatory variable. -The p-values are relatively large what means that country variable doesn't significantly affect the conversion rate (or - in other words - the conversion rate doesn't depend on which country the user comes from). -We failed to reject the null hypothesis

-We can check if adding another variable will change anything:

In [178…

```
#check if adding another variable will change anything:
lm2 = sm.Logit(df3['converted'], df3[['intercept', 'ab_page', 'UK', 'CA']])
results = lm2.fit()
results.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.366113
         Iterations 6
```

Out[178…

Logit Regression Results

Dep. Variable:	converted	**No. Observations:**	290584
Model:	Logit	**Df Residuals:**	290580
Method:	MLE	**Df Model:**	3
Date:	Wed, 16 Aug 2023	**Pseudo R-squ.:**	2.323e-05
Time:	14:49:33	**Log-Likelihood:**	-1.0639e+05
converged:	True	**LL-Null:**	-1.0639e+05
Covariance Type:	nonrobust	**LLR p-value:**	0.1760

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-1.9893	0.009	-223.763	0.000	-2.007	-1.972
ab_page	-0.0149	0.011	-1.307	0.191	-0.037	0.007
UK	0.0099	0.013	0.743	0.457	-0.016	0.036
CA	-0.0408	0.027	-1.516	0.130	-0.093	0.012

Adding ab_page as a variable didn't change anything significantly - we still cannot reject the null hypothesis. Let's check if adding other variables will make any difference

In [179…

```
UK_newpage = df3['ab_page'] * df3['UK']
df3['UK_newpage'] = UK_newpage
```

In [180…

```
CA_newpage = df3['ab_page'] * df3['CA']
df3['CA_newpage'] = CA_newpage
df3.head()
```

Out[180…

user_id	CA	UK	US	country	timestamp	group	landing_page	converted	intercept	ab_page	UK_newpage	UK_oldpage	CA_newpage
834778	0	1	0	UK	2017-01-14 23:08:43.304998	control	old_page	0	1	0	0	0	0
928468	0	0	1	US	2017-01-23 14:44:16.387854	treatment	new_page	0	1	1	0	0	0
822059	0	1	0	UK	2017-01-16 14:04:14.719771	treatment	new_page	1	1	1	1	1	0
711597	0	1	0	UK	2017-01-22 03:14:24.763511	control	old_page	0	1	0	0	0	0
710616	0	1	0	UK	2017-01-16 13:14:44.000513	treatment	new_page	0	1	1	1	1	0

In [182…

```
#fit the model and summarize the results
lm4 = sm.Logit(df3['converted'], df3[['intercept', 'ab_page', 'UK', 'CA', 'UK_newpage', 'CA_newpage']])
results = lm4.fit()
results.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.366109
         Iterations 6
```

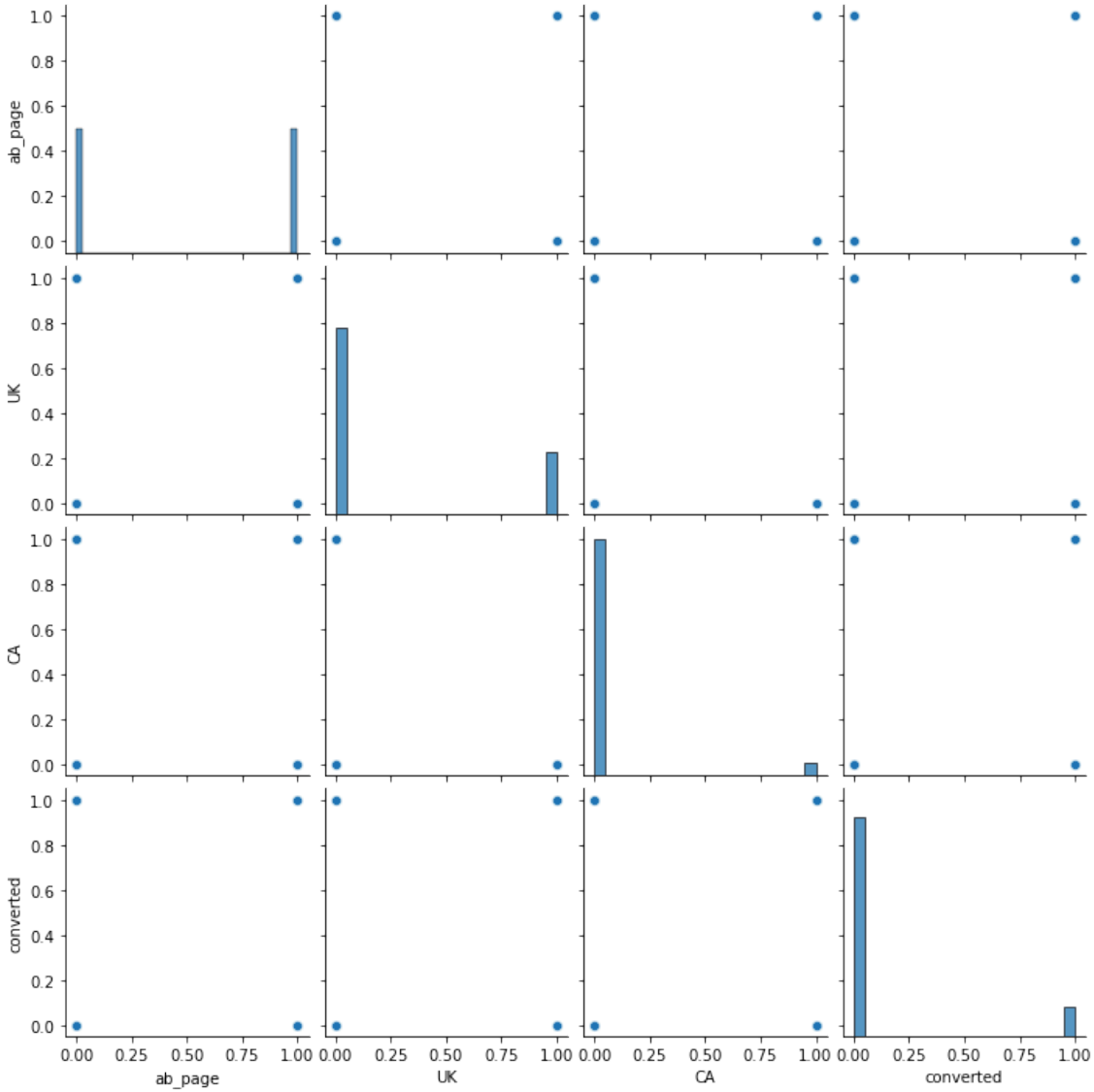
Out[182…

Model:	Logit	Pseudo R-squared:	0.000
Dependent Variable:	converted	AIC:	212782.6602
Date:	2023-08-16 15:08	BIC:	212846.1381
No. Observations:	290584	Log-Likelihood:	-1.0639e+05
Df Model:	5	LL-Null:	-1.0639e+05
Df Residuals:	290578	LLR p-value:	0.19199
Converged:	1.0000	Scale:	1.0000
No. Iterations:	6.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
intercept	-1.9865	0.0096	-206.3440	0.0000	-2.0053	-1.9676
ab_page	-0.0206	0.0137	-1.5052	0.1323	-0.0473	0.0062
UK	-0.0057	0.0188	-0.3057	0.7598	-0.0426	0.0311
CA	-0.0175	0.0377	-0.4652	0.6418	-0.0914	0.0563
UK_newpage	0.0314	0.0266	1.1807	0.2377	-0.0207	0.0835
CA_newpage	-0.0469	0.0538	-0.8718	0.3833	-0.1523	0.0585

In [170…

```
#we can additionaly use a paiplot to check for correlations between variables
import seaborn as sns
sns.pairplot(df3[['ab_page', 'UK', 'CA', 'converted']]);
```



We can see that number of users with the new page is the same as with the old page

FINAL CONCLUSIONS:

I conducted multiple tests to check if the new page will effectively increase conversion rates. All tests' results suggest that the new page won't increase conversion rate, I failed to reject the null hypothesis. That means that new page won't be better than the old page. It might be reasonable to conduct more tests (with more variables). However, for now - I would recommend keeping the old page.