Analyze A/B Test Results

Table of Contents

Introduction

This is an A/B tests to determine if using a new website will help convert the users from free users to paying members. 3 different concepts will be used to reach a conclusion if the new website design is more effective than the original design at converting users to paid members.

Part I - Probability

To get started, let's import our libraries.

In [3]:
```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
%matplotlib inline
#We are setting the seed to assure you get the same answers on quizzes as we set up
random.seed(42)
```

`1.` Now, read in the `ab_data.csv` data. Store it in `df`. **Use your dataframe to answer the questions in Quiz 1 of the classroom.**

a. Read in the dataset and take a look at the top few rows here:

In [4]:
```
df = pd.read_csv('ab_data.csv')
df.head()
```

Out[4]:

	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

b. Use the cell below to find the number of rows in the dataset.

In [5]:
```
df.shape
```

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

c. The number of unique users in the dataset.

In [6]:
```
df.nunique()
```

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

d. The proportion of users converted.

In [7]:
```
df['converted'].mean()
```

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

e. The number of times the `new_page` and `treatment` don't match.

In [8]:
```
# create a dataframe where control and new page match. Then check how many rows are i
del1 = df.query('group == "control" and landing_page == "new_page" ')
del1.count()
```

Out[8]:
```
user_id         1928
timestamp       1928
group           1928
landing_page    1928
converted       1928
dtype: int64
```

In [9]:
```
# create a dataframe where treatment and old page match. Then check how many rows are
del2 = df.query('group == "treatment" and landing_page == "old_page"')
del2.count()
```

Out[9]:
```
user_id         1965
timestamp       1965
group           1965
landing_page    1965
converted       1965
dtype: int64
```

f. Do any of the rows have missing values?

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
user_id         0
timestamp       0
group           0
landing_page    0
converted       0
dtype: int64
```

`2.` For the rows where **treatment** does not match with **new_page** or **control** does not match with **old_page**, we cannot be sure if this row truly received the new or old page. Use **Quiz 2** in the classroom to figure out how we should handle these rows.

a. Now use the answer to the quiz to create a new dataset that meets the specifications from the quiz. Store your new dataframe in **df2**.

In [11]:
```
# drop the dataframe where control and new page match from the original dataframe and
df1 = df.drop(del1.index)
df1.shape
```

Out[11]:
```
(292550, 5)
```

In [12]:
```
# now from this newly created df1 drop the datafram where treatment and old page matc
df2 = df1.drop(del2.index)
df2.shape
```

Out[12]:
```
(290585, 5)
```

In [13]:
```
# Double Check all of the correct rows were removed - this should be 0
df2[((df2['group'] == 'treatment') == (df2['landing_page'] == 'new_page')) == False].s
```

Out[13]:
```
0
```

`3.` Use **df2** and the cells below to answer questions for **Quiz3** in the classroom.

a. How many unique **user_id**s are in **df2**?

In [14]:
```
df2.nunique()
```

Out[14]:
```
user_id         290584
timestamp       290585
group                2
landing_page         2
converted            2
dtype: int64
```

b. There is one **user_id** repeated in **df2**. What is it?

In [15]:
```
df2['user_id'].duplicated().sum()
```

Out[15]:
```
1
```

c. What is the row information for the repeat **user_id**?

In [16]:
```
duplicate = df2[df2.duplicated(['user_id'])]
duplicate
```

Out[16]:

	user_id	timestamp	group	landing_page	converted
2893	773192	2017-01-14 02:55:59.590927	treatment	new_page	0

In [17]:
```
# find the shape of df2
df2.shape
```

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

d. Remove **one** of the rows with a duplicate **user_id**, but keep your dataframe as **df2**.

In [18]:
```
df2 = df2.drop(duplicate.index)
```

In [19]:
```
# check the shape again to make sure the duplicated row has been removed
df2.shape
```

Out[19]:
```
(290584, 5)
```

`4.` Use **df2** in the cells below to answer the quiz questions related to **Quiz 4** in the classroom.

a. What is the probability of an individual converting regardless of the page they receive?

In [20]:
```
df2['converted'].mean()
```

Out[20]:
```
0.11959708724499628
```

b. Given that an individual was in the `control` group, what is the probability they converted?

In [21]:
```
df2.query('group=="control"')['converted'].mean()
```

Out[21]:
```
0.1203863045004612
```

c. Given that an individual was in the `treatment` group, what is the probability they converted?

In [22]:
```
df2.query('group=="treatment"')['converted'].mean()
```

Out[22]:
```
0.11880806551510564
```

d. What is the probability that an individual received the new page?

In [23]:
```
new = df2.query("landing_page == 'new_page'")
len(new)/len(df2)
```

Out[23]:
```
0.5000619442226688
```

e. Consider your results from parts (a) through (d) above, and explain below whether you think there is sufficient evidence to conclude that the new treatment page leads to more conversions.

0.1203863045004612 - 0.11880806551510564 = 0.0015782389853555567 This is less than a 1 % change in conversion rate. infact the control group seems to have a better conversion rate than the new treatment page. so far there is no evidence that the new treatment page leads to more conversions

Part II - A/B Test

Notice that because of the time stamp associated with each event, you could technically run a hypothesis test continuously as each observation was observed.

However, then the hard question is do you 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 do you run to render a decision that neither page is better than another?

These questions are the difficult parts associated with A/B tests in general.

`1.` For now, consider you need to make the decision just based on all the data provided. If you want to assume that the old page is better unless the new page proves to be definitely better at a Type I error rate of 5%, what should your null and alternative hypotheses be? You can state your hypothesis in terms of words or in terms of p_{old} and p_{new}, which are the converted rates for the old and new pages.

null: $p\_old >= p\_new$

alternate: $p\_old < p\_new$

`2.` Assume under the null hypothesis, p_{new} and p_{old} both have "true" success rates equal to the **converted** success rate regardless of page - that is p_{new} and p_{old} are equal. Furthermore, assume they are equal to the **converted** rate in **ab_data.csv** regardless of the page.

Use a sample size for each page equal to the ones in **ab_data.csv**.

Perform the sampling distribution for the difference in **converted** between the two pages over 10,000 iterations of calculating an estimate from the null.

Use the cells below to provide the necessary parts of this simulation. If this doesn't make complete sense right now, don't worry - you are going to work through the problems below to complete this problem. You can use **Quiz 5** in the classroom to make sure you are on the right track.

a. What is the **conversion rate** for p_{new} under the null?

In [24]:
```
p_new = df2['converted'].mean()

p_new
```

Out[24]:
```
0.11959708724499628
```

b. What is the **conversion rate** for p_{old} under the null?

In [25]:
```
p_old = p_new
p_old
```

Out[25]:
```
0.11959708724499628
```

c. What is n_{new}, the number of individuals in the treatment group?

In [26]:
```
# user_id is arbitrarily selected as one of the columns used for count. any of the ot
# would have given the same result
n_new = df2.query('group == "treatment"')['user_id'].count()
n_new
```

Out[26]:
```
145310
```

d. What is n_{old}, the number of individuals in the control group?

In [27]:
```
# user_id is arbitrarily selected as one of the columns used for count. any of the ot
# would have given the same result
n_old = df2.query('group == "control"')['user_id'].count()
n_old
```

Out[27]:
```
145274
```

e. Simulate n_{new} transactions with a conversion rate of p_{new} under the null. Store these n_{new} 1's and 0's in **new_page_converted**.

In [28]:
```
new_page_converted = np.random.choice([0,1], size=n_new, p = [1-p_new, p_new])
new_page_converted
```

Out[28]:
```
array([0, 0, 1, ..., 0, 0, 0])
```

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

In [29]:
```
old_page_converted = np.random.choice([0,1], size=n_old, p = [1-p_old, p_old])
old_page_converted
```

Out[29]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

g. Find p_{new} - p_{old} for your simulated values from part (e) and (f).

In [30]:
```
p_new - p_old
```

Out[30]:
```
0.0
```

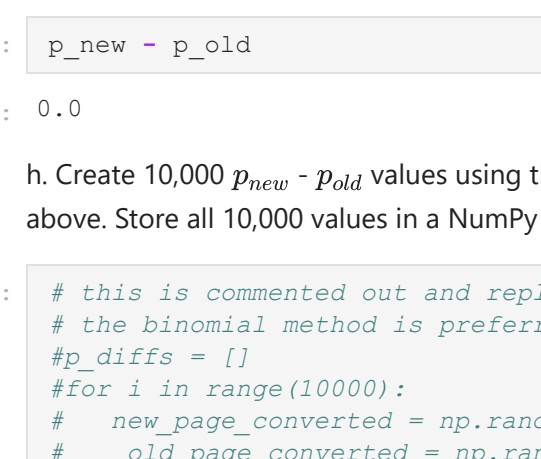
h. Create 10,000 p_{new} - p_{old} values using the same simulation process you used in parts (a) through (g) above. Store all 10,000 values in a NumPy array called **p_diffs**.

In [31]:
```
# this is commented out and replaced with code below. this is an alternative method t
# the binomial method is preferred as it avoids using a python loop which runs much s
#p_diffs = []
#for i in range(10000):
#   new_page_converted = np.random.choice([0,1], size=n_new, p = [1-p_new, p_new]).me
#    old_page_converted = np.random.choice([0,1], size=n_old, p = [1-p_old, p_old]).m
#    diff = new_page_converted - old_page_converted
#    p_diffs.append(diff)
```

In [32]:
```
new_converted_simulation = np.random.binomial(n_new, p_new, 10000)/n_new
old_converted_simulation = np.random.binomial(n_old, p_old, 10000)/n_old
p_diffs = new_converted_simulation - old_converted_simulation
```

i. Plot a histogram of the **p_diffs**. Does this plot look like what you expected? Use the matching problem in the classroom to assure you fully understand what was computed here.

In [33]:
```
plt.hist(p_diffs)
plt.axvline(x=np.mean(p_diffs), color = 'r');
```



j. What proportion of the **p_diffs** are greater than the actual difference observed in **ab_data.csv**?

In [34]:
```
obs_diff = df2.query('group=="treatment"').converted.mean() - df2.query('group=="cont

p_val = (p_diffs > obs_diff).mean()
p_val
```

Out[34]:
```
0.9022
```

k. Please explain using the vocabulary you've learned in this course what you just computed in part **j.** What is this value called in scientific studies? What does this value mean in terms of whether or not there is a difference between the new and old pages?

Our p-value is 11.87%. Since we are using a 95% confidence interval our alpha is 5%. This is the probability of rejecting the null hypothesis when its true, also known as type 1 error.

Our p-value is greater than our alpha so we would fail to reject the null hypothesis. This means there is no statistically signicant conversion rate difference between the old and new page.

l. 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. Fill in the below to 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.

In [35]:
```
import statsmodels.api as sm

convert_old = df2.query('landing_page=="old_page" and converted==1')['converted'].cou
convert_new = df2.query('landing_page=="new_page" and converted==1')['converted'].cou
n_old = df2.query('landing_page=="old_page"')['converted'].count()
n_new = df2.query('landing_page=="new_page"')['converted'].count()
```

m. Now use `stats.proportions_ztest` to compute your test statistic and p-value. Here is a helpful link on using the built in.

In [36]:
```
from statsmodels.stats.proportion import proportions_ztest
stat,pval = proportions_ztest([convert_old,convert_new], [n_old,n_new], alternative='
stat, pval
```

Out[36]:
```
(1.3109241984234394, 0.9050583127590245)
```

n. What do the z-score and p-value you computed in the previous question mean for the conversion rates of the old and new pages? Do they agree with the findings in parts **j.** and **k.**?

The Z score tells us that our data is at 1.31 standard deviations away from the mean. A p value of 0.905 is way above our alpha of 0.05. Therefore suggesting the data is not statistically significant so we fail to reject the null.

Since we fail to reject the null, this is in agreement with the answers in parts j and k

Part III - A regression approach

`1.` In this final part, you will see that the result you achieved in the A/B test in Part II above can also be achieved by performing regression.

a. Since each row is either a conversion or no conversion, what type of regression should you be performing in this case?

Logistic Regression

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, you first need to create in df2 a column for the intercept, and create a dummy variable column for which page each user received. 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 [37]:
```
df2['intercept'] = 1

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

Out[37]:

	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

c. Use **statsmodels** to instantiate your regression model on the two columns you created in part b., then fit the model using the two columns you created in part **b.** to predict whether or not an individual converts.

In [38]:
```
logit_mod = sm.Logit(df2['converted'], df2[['intercept', 'ab_page']])
results = logit_mod.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.366118
         Iterations 6
```

d. Provide the summary of your model below, and use it as necessary to answer the following questions.

In [39]:
```
results.summary2()
```

Out[39]:

Model:	Logit	Pseudo R-squared:	0.000
Dependent Variable:	converted	AIC:	212780.3502
Date:	2021-04-06 01:32	BIC:	212801.5095
No. Observations:	290584	Log-Likelihood:	-1.0639e+05
Df Model:	1	LL-Null:	-1.0639e+05
Df Residuals:	290582	LLR p-value:	0.18988
Converged:	1.0000	Scale:	1.0000
No. Iterations:	6.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
intercept	-1.9888	0.0081	-246.6690	0.0000	-2.0046	-1.9730
ab_page	-0.0150	0.0114	-1.3109	0.1899	-0.0374	0.0074

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

Hint: What are the null and alternative hypotheses associated with your regression model, and how do they compare to the null and alternative hypotheses in **Part II**?

The p value is 0.1899.

the null hypothesis from the regression model states that the parameter is equal to zero. the alternate states that it does not equal zero which makes it a 2 sided hypothesis test.

the null and alternate hypothesis in part II are one sided tests. therefore the resulting p value from the regression is double the answer in part II

f. Now, you are considering other things that might influence whether or not an individual converts. Discuss why it is a good idea to consider other factors to add into your regression model. Are there any disadvantages to adding additional terms into your regression model?

another variable to consider is the timestamp. it would be a good to add to the model to see if it can increase the accuracy of the model for predicting if a user will convert or not. the disadvantage of this is that it could introduce multicolinearity which needs to be tested if it should be removed. and it can introduce the ideas of change aversion and the novelty effect of the newly designed website

In [40]:
```
results.params
```

Out[40]:
```
intercept   -1.988777
ab_page     -0.014989
dtype: float64
```

In [41]:
```
1/np.exp(_)
```

Out[41]:
```
intercept    7.306593
ab_page      1.015102
dtype: float64
```

interpretation: for each 1 unit decrease in ab_page, conversion is 1.015 times as likely holding all else constant. further interpreted as: if ab_page is 0, which is the old control page, then the chance of converion goes up by 1.015 times.

g. Now along with testing if the conversion rate changes for different pages, also add an effect based on which country a user lives in. You will need to read in the **countries.csv** dataset and merge together your datasets on the appropriate rows. Here are the docs for joining tables.

Does it appear that country had an impact on conversion? Don't forget to create dummy variables for these country columns - **Hint: You will need two columns for the three dummy variables.** Provide the statistical output as well as a written response to answer this question.

In [42]:
```
# read in the countries CSV file
# create a new dataframe and name it df3 that joins df2 and the countries CSV file.

countries = pd.read_csv('countries.csv')
df3 = df2.join(countries.set_index('user_id'), on='user_id')
df3.head()
```

Out[42]:

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

In [43]:
```
# get the dummy values for the countries read in and give them descriptive column nam

df3[['CA', 'UK', 'US']] = pd.get_dummies(df3['country'])
df3.tail()
```

Out[43]:

	user_id	timestamp	group	landing_page	converted	intercept	ab_page	country	CA	UK	U
294473	751197	2017-01-03 22:28:38.630509	control	old_page	0	1	0	US	0	0	
294474	945152	2017-01-12 00:51:57.078372	control	old_page	0	1	0	US	0	0	
294475	734608	2017-01-22 11:45:03.439544	control	old_page	0	1	0	US	0	0	
294476	697314	2017-01-15 01:20:28.957438	control	old_page	0	1	0	US	0	0	
294477	715931	2017-01-16 12:40:24.467417	treatment	new_page	0	1	1	UK	0	1	

In [44]:
```
# build and fit a logistic model using the country, US, as a baseline
df3['intercept'] = 1

logit_mod = sm.Logit(df3['converted'], df3[['intercept', 'ab_page', 'UK', 'CA']])
results = logit_mod.fit()
results.summary2()
```

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

Out[44]:

Model:	Logit	Pseudo R-squared:	0.000
Dependent Variable:	converted	AIC:	212781.1253
Date:	2021-04-06 01:32	BIC:	212823.4439
No. Observations:	290584	Log-Likelihood:	-1.0639e+05
Df Model:	3	LL-Null:	-1.0639e+05
Df Residuals:	290580	LLR p-value:	0.17599
Converged:	1.0000	Scale:	1.0000
No. Iterations:	6.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
intercept	-1.9893	0.0089	-223.7628	0.0000	-2.0067	-1.9718
ab_page	-0.0149	0.0114	-1.3069	0.1912	-0.0374	0.0075
UK	0.0099	0.0133	0.7433	0.4573	-0.0162	0.0359
CA	-0.0408	0.0269	-1.5161	0.1295	-0.0934	0.0119

In [45]:
```
# find the coeffecients of the logistic model
results.params
```

Out[45]:
```
intercept   -1.989272
ab_page     -0.014943
UK           0.009883
CA          -0.040757
dtype: float64
```

In [46]:
```
# for coeffeceints over 1, we can interpret them by raising as the power of an expone
np.exp(results.params)
```

Out[46]:
```
intercept    0.136795
ab_page      0.985168
UK           1.009932
CA           0.960062
dtype: float64
```

In [47]:
```
# for coefficents below 1 we can interpret them more easily when we take the inverse
1/np.exp(results.params)
```

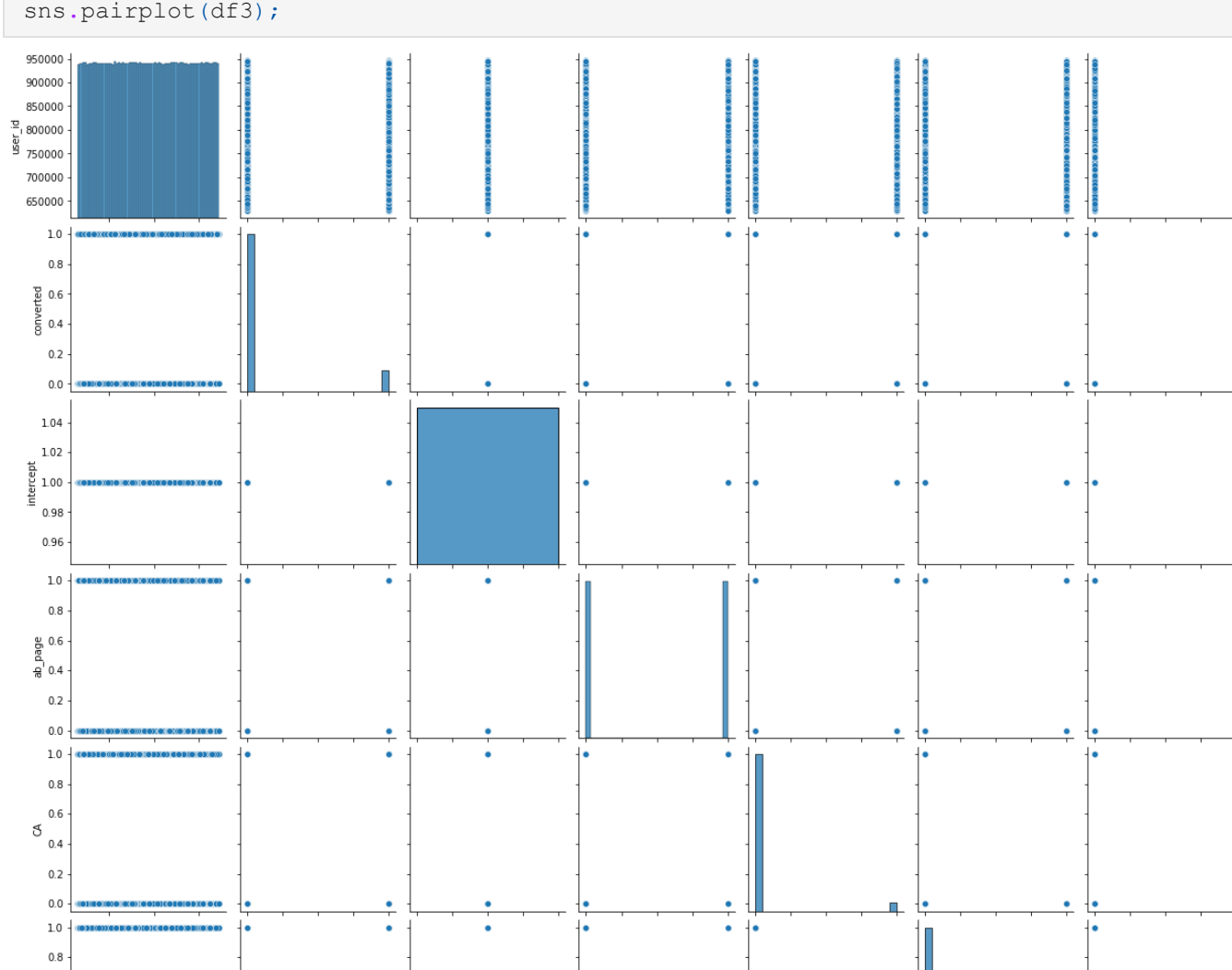
Out[47]:
```
intercept    7.310207
ab_page      1.015056
UK           0.990165
CA           1.041599
dtype: float64
```

the country doesnt seem to have a major impact on the conversion rate. none of the p values for any of the variables is below 0.05, therefore we fail to reject the null hypothesis.

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

Provide the summary results, and your conclusions based on the results.

In [48]:
```
#run a scatter plot to see if the data has any multicolinearity
import seaborn as sns
sns.pairplot(df3);
```



In [49]:
```
# no multicolinearity is seen so we can assume it is safe to keep all the variables u
```

In [50]:
```
#create an interaction of page and country
df3['UK_page'] = df3['UK']*df3['ab_page']
df3['CA_page'] = df3['CA']*df3['ab_page']
df3.head()
```

Out[50]:

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

In [51]:
```
# build and fit a logistic model using the country, US, and the interaction of US_pag
logit_mod = sm.Logit(df3['converted'], df3[['intercept', 'ab_page', 'UK', 'CA', 'UK_p
results = logit_mod.fit()
results.summary2()
```

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

Out[51]:

Model:	Logit	Pseudo R-squared:	0.000
Dependent Variable:	converted	AIC:	212782.6602
Date:	2021-04-06 01:33	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_page	0.0314	0.0266	1.1807	0.2377	-0.0207	0.0835
CA_page	-0.0469	0.0538	-0.8718	0.3833	-0.1523	0.0585

In [52]:
```
# find the coefficients of the model
results.params
```

Out[52]:
```
intercept   -1.986478
ab_page     -0.020564
UK          -0.005744
CA          -0.017528
UK_page      0.031398
CA_page     -0.046885
dtype: float64
```

In [53]:
```
# interpret coefficents over 1 by raising them to the power of an exponent
np.exp(results.params)
```

Out[53]:
```
intercept    0.137178
ab_page      0.979646
UK           0.994272
CA           0.982625
UK_page      1.031896
CA_page      0.954198
dtype: float64
```

In [54]:
```
# interpret coefficents under 1 by raising them to the power of an exponent
1/np.exp(results.params)
```

Out[54]:
```
intercept    7.289813
ab_page      1.020776
UK           1.005761
CA           1.017682
UK_page      0.969090
CA_page      1.048001
dtype: float64
```

all the coefficents have a p value abpve 0.05 which tells us we fail to reject the null. So no statistical significance is seen in converting between the new page and old page

In [55]:
```
df3.head()
```

Out[55]:

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

In [56]:
```
# create new column called date and convert timestamp column into datetime format
import datetime
df3['time'] = pd.to_datetime(df3['timestamp'])
```

In [57]:
```
# check to see that the newly created column, time, is of type datetime
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 290584 entries, 0 to 294477
Data columns (total 14 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   user_id       290584 non-null  int64
 1   timestamp     290584 non-null  object
 2   group         290584 non-null  object
 3   landing_page  290584 non-null  object
 4   converted     290584 non-null  int64
 5   intercept     290584 non-null  int64
 6   ab_page       290584 non-null  uint8
 7   country       290584 non-null  object
 8   CA            290584 non-null  uint8
 9   UK            290584 non-null  uint8
 10  US            290584 non-null  uint8
 11  UK_page       290584 non-null  uint8
 12  CA_page       290584 non-null  uint8
 13  time          290584 non-null  datetime64[ns]
dtypes: datetime64[ns](1), int64(3), object(4), uint8(6)
memory usage: 21.6+ MB
```

In [58]:
```
# find out how long the experiment has run for
max = df3.time.max()
min = df3.time.min()
max-min
```

Out[58]:
```
Timedelta('21 days 23:59:49.081927')
```

In [59]:
```
# this is a relatively short run experiment.
# it can be insightful to see if the day of the week has an impact on the conversion
# create a new column that shows what day of the week an instance occurred on
df3['day_of_week'] = df3['time'].dt.day_name()
```

In [60]:
```
# get the dummy variables and save it in a new dataframe called df4
df4 = df3.join(pd.get_dummies(df3['day_of_week']))
df4.head()
```

Out[60]:

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

5 rows × 22 columns

In [61]:
```
# build a logistic model and use country US, interaction, US_page and day of week, Mo
logit_mod = sm.Logit(df4['converted'], df4[['intercept', 'ab_page', 'CA', 'UK', 'CA_p
results = logit_mod.fit()
results.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.366099
         Iterations 6
```

Out[61]:

Model:	Logit	Pseudo R-squared:	0.000
Dependent Variable:	converted	AIC:	212789.1230
Date:	2021-04-06 01:33	BIC:	212916.0787
No. Observations:	290584	Log-Likelihood:	-1.0638e+05
Df Model:	11	LL-Null:	-1.0639e+05
Df Residuals:	290572	LLR p-value:	0.29688
Converged:	1.0000	Scale:	1.0000

No. Iterations:	6.0000

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
intercept	-1.9719	0.0163	-120.9419	0.0000	-2.0039	-1.9400
ab_page	-0.0205	0.0137	-1.4998	0.1337	-0.0473	0.0063
CA	-0.0174	0.0377	-0.4617	0.6443	-0.0913	0.0565
UK	-0.0056	0.0188	-0.2965	0.7668	-0.0424	0.0313
CA_page	-0.0470	0.0538	-0.8743	0.3819	-0.1524	0.0584
UK_page	0.0314	0.0266	1.1795	0.2382	-0.0208	0.0835
Tuesday	-0.0153	0.0202	-0.7551	0.4502	-0.0549	0.0243
Wednesday	-0.0076	0.0211	-0.3604	0.7186	-0.0490	0.0338
Thursday	-0.0110	0.0212	-0.5212	0.6022	-0.0525	0.0305
Friday	-0.0425	0.0212	-1.9995	0.0456	-0.0841	-0.0008
Saturday	-0.0030	0.0211	-0.1437	0.8857	-0.0443	0.0383
Sunday	-0.0254	0.0211	-1.2026	0.2291	-0.0668	0.0160

In [62]:
```
# get coefficents of logistic model
results.params
```

Out[62]:
```
intercept   -1.971926
ab_page     -0.020491
CA          -0.017398
UK          -0.005572
CA_page     -0.047021
UK_page      0.031366
Tuesday     -0.015257
Wednesday   -0.007607
Thursday    -0.011035
Friday      -0.042489
Saturday    -0.003028
Sunday      -0.025414
dtype: float64
```

In [63]:
```
# interpret coefficeints greater than 1
np.exp(results.params)
```

Out[63]:
```
intercept    0.139188
ab_page      0.979718
CA           0.982753
UK           0.994444
CA_page      0.954067
UK_page      1.031863
Tuesday      0.984858
Wednesday    0.992422
Thursday     0.989026
Friday       0.958401
Saturday     0.996976
Sunday       0.974906
dtype: float64
```

In [64]:
```
# interpret coefficents below 1
1/np.exp(results.params)
```

Out[64]:
```
intercept    7.184503
ab_page      1.020702
CA           1.017550
UK           1.005587
CA_page      1.048144
UK_page      0.969121
Tuesday      1.015374
Wednesday    1.007636
Thursday     1.011096
Friday       1.043404
Saturday     1.003033
Sunday       1.025740
dtype: float64
```

From these coefficents it seems that the day of the week has no significant impact on conversion rates.

At best relative to monday, students using the website on Friday are more liekly to convert by a factor of 1.043.

This seems like a practically insignificant difference.

All p values in this model are above 0.05 so we fail to reject the null hypothesis

However this experiment has only been run for 21 days so there just might not be enough data.

I would recoomend to run the experiment for a longer time

Extra models are created below to test if accuracy increases by removing certain variables

In [65]:
```
# build identical model however remove interaction
logit_mod = sm.Logit(df4['converted'], df4[['intercept', 'ab_page', 'CA', 'UK', 'Tues
results = logit_mod.fit()
results.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.366103
         Iterations 6
```

Out[65]:

Model:	Logit	Pseudo R-squared:	0.000
Dependent Variable:	converted	AIC:	212787.5901
Date:	2021-04-06 01:33	BIC:	212893.3866
No. Observations:	290584	Log-Likelihood:	-1.0638e+05
Df Model:	9	LL-Null:	-1.0639e+05
Df Residuals:	290574	LLR p-value:	0.31314
Converged:	1.0000	Scale:	1.0000
No. Iterations:	6.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
intercept	-1.9747	0.0159	-124.2803	0.0000	-2.0058	-1.9436
ab_page	-0.0149	0.0114	-1.3018	0.1930	-0.0373	0.0075
CA	-0.0407	0.0269	-1.5138	0.1301	-0.0934	0.0120
UK	0.0100	0.0133	0.7551	0.4502	-0.0160	0.0361
Tuesday	-0.0153	0.0202	-0.7558	0.4498	-0.0549	0.0243
Wednesday	-0.0076	0.0211	-0.3594	0.7193	-0.0490	0.0338
Thursday	-0.0111	0.0212	-0.5239	0.6004	-0.0526	0.0304
Friday	-0.0425	0.0212	-2.0002	0.0455	-0.0842	-0.0009
Saturday	-0.0031	0.0211	-0.1457	0.8841	-0.0444	0.0382
Sunday	-0.0254	0.0211	-1.2021	0.2293	-0.0668	0.0160

In [66]:
```
# get coefficents of logistic model
results.params
```

Out[66]:
```
intercept   -1.974700
ab_page     -0.014885
CA          -0.040695
UK           0.010040
Tuesday     -0.015270
Wednesday   -0.007586
Thursday    -0.011091
Friday      -0.042502
Saturday    -0.003070
Sunday      -0.025405
dtype: float64
```

In [67]:
```
# interpret coefficeints greater than 1
np.exp(results.params)
```

Out[67]:
```
intercept    0.138803
ab_page      0.985225
CA           0.960122
UK           1.010090
Tuesday      0.984846
Wednesday    0.992443
Thursday     0.988970
Friday       0.958388
Saturday     0.996934
Sunday       0.974915
dtype: float64
```

In [68]:
```
# interpret coefficents below 1
1/np.exp(results.params)
```

Out[68]:
```
intercept    7.204456
ab_page      1.014997
CA           1.041535
UK           0.990010
Tuesday      1.015387
Wednesday    1.007615
Thursday     1.011153
Friday       1.043419
Saturday     1.003075
Sunday       1.025730
dtype: float64
```

In [69]:
```
# build similar model further simplifying by removing countires
logit_mod = sm.Logit(df4['converted'], df4[['intercept', 'ab_page', 'Tuesday','Wednesd
results = logit_mod.fit()
results.summary2()
```

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

Out[69]:

Model:	Logit	Pseudo R-squared:	0.000
Dependent Variable:	converted	AIC:	212786.8299
Date:	2021-04-06 01:33	BIC:	212871.4671
No. Observations:	290584	Log-Likelihood:	-1.0639e+05
Df Model:	7	LL-Null:	-1.0639e+05
Df Residuals:	290576	LLR p-value:	0.40444
Converged:	1.0000	Scale:	1.0000
No. Iterations:	6.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
intercept	-1.9742	0.0155	-127.7522	0.0000	-2.0045	-1.9439
ab_page	-0.0149	0.0114	-1.3058	0.1916	-0.0373	0.0075
Tuesday	-0.0152	0.0202	-0.7523	0.4519	-0.0548	0.0244
Wednesday	-0.0076	0.0211	-0.3620	0.7173	-0.0490	0.0337
Thursday	-0.0111	0.0212	-0.5264	0.5986	-0.0526	0.0303
Friday	-0.0425	0.0212	-1.9991	0.0456	-0.0841	-0.0008
Saturday	-0.0030	0.0211	-0.1446	0.8850	-0.0443	0.0383
Sunday	-0.0253	0.0211	-1.1986	0.2307	-0.0667	0.0161

In [70]:
```
# get coefficents of logistic model
results.params
```

Out[70]:
```
intercept   -1.974176
ab_page     -0.014931
Tuesday     -0.015200
Wednesday   -0.007641
Thursday    -0.011145
Friday      -0.042479
Saturday    -0.003047
Sunday      -0.025329
dtype: float64
```

In [71]:
```
# interpret coefficeints greater than 1
np.exp(results.params)
```

Out[71]:
```
intercept    0.138876
ab_page      0.985180
Tuesday      0.984915
Wednesday    0.992388
Thursday     0.988917
Friday       0.958411
Saturday     0.996958
Sunday       0.974989
dtype: float64
```

In [72]:
```
# interpret coefficents below 1
1/np.exp(results.params)
```

Out[72]:
```
intercept    7.200685
ab_page      1.015043
Tuesday      1.015316
Wednesday    1.007670
Thursday     1.011207
Friday       1.043394
Saturday     1.003051
Sunday       1.025652
dtype: float64
```

We can see removing the countries and interactions has no effect on conversion rates for day of week as th ecoeffecients remain unchanged.

Final recommendation is to run experiment for a longer period of time. As for the data collected here, we have failed to reject the null and see no statistically significant difference between the old and new websites when it comes to converting users

In [73]:
```
from subprocess import call
call(['python', '-m', 'nbconvert', 'Analyze_ab_test_results_notebook.ipynb'])
```

Out[73]: 1