# Analyze_ab_test_results_notebook

June 22, 2022

## 1 Analyze A/B Test Results

This project will assure you have mastered the subjects covered in the statistics lessons. We have organized the current notebook into the following sections:

- Section **??**
- Section **??**
- Section **??**
- Section **??**
- Section **??**
- Section **??**

Specific programming tasks are marked with a **ToDo** tag.
## Introduction
A/B tests are very commonly performed by data analysts and data scientists. For this project, you will be working to understand the results of an A/B test run by an e-commerce website. Your goal is to work through this notebook to help the company understand if they should: - Implement the new webpage, - Keep the old webpage, or - Perhaps run the experiment longer to make their decision.

Each **ToDo** task below has an associated quiz present in the classroom. Though the classroom quizzes are **not necessary** to complete the project, they help ensure you are on the right track as you work through the project, and you can feel more confident in your final submission meeting the rubric specification.

> **Tip**: Though it's not a mandate, students can attempt the classroom quizzes to ensure statistical numeric values are calculated correctly in many cases.

## Part I - Probability
To get started, let's import our libraries.

```
In [1]: 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)
```

### 1.0.1 ToDo 1.1

Now, read in the `ab_data.csv` data. Store it in `df`. Below is the description of the data, there are a total of 5 columns:

| Data columns | Purpose | Valid values |
|---|---|---|
| user_id | Unique ID | Int64 values |
| timestamp | Time stamp when the user visited the webpage | - |
| group | In the current A/B experiment, the users are categorized into two broad groups. The `control` group users are expected to be served with `old_page`; and `treatment` group users are matched with the `new_page`. However, **some inaccurate rows** are present in the initial data, such as a `control` group user is matched with a `new_page`. | `['control', 'treatment']` |
| landing_page | It denotes whether the user visited the old or new webpage. | `['old_page', 'new_page']` |
| converted | It denotes whether the user decided to pay for the company's product. Here, `1` means yes, the user bought the product. | `[0, 1]` |

Use your dataframe to answer the questions in Quiz 1 of the classroom.

**Tip**: Please save your work regularly.

**a.** Read in the dataset from the `ab_data.csv` file and take a look at the top few rows here:

```
In [2]: 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
        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 [3]: df.shape[0]
```

```
Out[3]: 294478
```

**c.** The number of unique users in the dataset.

```
In [4]: df.nunique()['user_id']
```

```
Out[4]: 290584
```

**d.** The proportion of users converted.

```
In [5]: df.query('converted ==1')['converted'].count() / df.shape[0]
```

```
Out[5]: 0.11965919355605512
```

**e.** The number of times when the "group" is `treatment` but "landing_page" is not a `new_page`.

```
In [6]: df.query('group == "treatment" and landing_page != "new_page"').shape[0] + df.query('gro
```

```
Out[6]: 3893
```

**f.** Do any of the rows have missing values?

```
In [7]: df.isnull().sum()
```

```
Out[7]: user_id         0
        timestamp       0
        group           0
        landing_page    0
        converted       0
        dtype: int64
```

### 1.0.2 ToDo 1.2

In a particular row, the **group** and **landing_page** columns should have either of the following acceptable values:

| user_id | timestamp | group | landing_page | converted |
|---|---|---|---|---|
| XXXX | XXXX | `control` | `old_page` | X |
| XXXX | XXXX | `treatment` | `new_page` | X |

It means, the `control` group users should match with `old_page`; and `treatment` group users should matched with the `new_page`.

However, for the rows where `treatment` does not match with `new_page` or `control` does not match with `old_page`, we cannot be sure if such rows truly received the new or old wepage.

Use **Quiz 2** in the classroom to figure out how should we handle the rows where the group and landing_page columns don't match?

**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 [8]: # Remove the inaccurate rows, and store the result in a new dataframe df2
        drop1 = df.query('group == "treatment" and landing_page != "new_page"').index
        drop2 = df.query('group == "control" and landing_page == "new_page"').index
        df_1= df.drop(drop1)
        df2= df_1.drop(drop2)
        df2.head()
```

```
Out[8]:    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 [9]: # Double Check all of the incorrect rows were removed from df2 -
        # Output of the statement below should be 0
        df2[((df2['group'] == 'treatment') == (df2['landing_page'] == 'new_page')) == False].sha
```

```
Out[9]: 0
```

### 1.0.3 ToDo 1.3

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

```
In [10]: df2.nunique()
```

```
Out[10]: 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 [11]: df2[df2.user_id.duplicated()]
```

```
Out[11]:       user_id                   timestamp      group landing_page  converted
         2893   773192  2017-01-14 02:55:59.590927  treatment     new_page          0
```

**c.** Display the rows for the duplicate **user_id**?

```
In [12]: duplicate= df2[df2.duplicated(['user_id'],keep=False)]
         print("Duplicated Rows Based On user_id:")
         duplicate

Duplicated Rows Based On user_id:


Out[12]:       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
```

**d.** Remove **one** of the rows with a duplicate **user_id**, from the **df2** dataframe.

```
In [13]: # Remove one of the rows with a duplicate user_id..
         # Hint: The dataframe.drop_duplicates() may not work in this case because the rows with
         df2.drop(2893, inplace= True)
         # Check again if the row with a duplicate user_id is deleted or not
         df2.user_id.duplicated().sum()

Out[13]: 0
```

### 1.0.4 ToDo 1.4

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

> **Tip**: The probability you'll compute represents the overall "converted" success rate in the population and you may call it $p_{population}$.

```
In [14]: df2.query('converted ==1')['converted'].count()/ df2.shape[0]

Out[14]: 0.11959708724499628
```

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

```
In [15]: df2.query('group == "control" and converted == 1').shape[0] / df2.query('group=="contro

Out[15]: 0.1203863045004612
```

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

```
In [16]: len(df2.query('group == "treatment" and converted == 1'))/ len(df2.query('group == "tre

Out[16]: 0.11880806551510564

In [17]: # Calculate the actual difference (obs_diff) between the conversion rates for the two g
         df2.query('group == "control" and converted == 1').shape[0] / df2.query('group=="contro
Out[17]: 0.0015782389853555567
```

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

```
In [18]: df2.query('landing_page == "new_page"').shape[0]/df2.shape[0]
```

```
Out[18]: 0.5000619442226688
```

**e.** Consider your results from parts (a) through (d) above, and explain below whether the new `treatment` group users lead to more conversions.

> **The probability of an individual converting regardless of the page they receive is 11.9%**
> **and we notice that the probability that an individual received the new page is about 50% >P(control to treatment) = 12% P(treatment to control) = 11.8% , almost the same so the new page doesn't like it leads to more conversions that the old page**

## Part II - A/B Test
Since a timestamp is associated with each event, you could run a hypothesis test continuously as long as you observe the events.
However, then the hard questions would be: - 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.

### 1.0.5 ToDo 2.1

For now, consider you need to make the decision just based on all the data provided.

> Recall that you just calculated that the "converted" probability (or rate) for the old page is *slightly* higher than that of the new page (ToDo 1.4.c).

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 be your null and alternative hypotheses ($H_0$ and $H_1$)?
You can state your hypothesis in terms of words or in terms of $p_{old}$ and $p_{new}$, which are the "converted" probability (or rate) for the old and new pages respectively.

> **if the p-value is less than your Type I error(0.05), then we have evidence to reject the null and choose the alternative[new page is better than the old page]. Otherwise, you fail to reject the null hypothesis[we don't need to create a new page]**

### 1.0.6 ToDo 2.2 - Null Hypothesis $H_0$ Testing

Under the null hypothesis $H_0$, assume that $p_{new}$ and $p_{old}$ are equal. Furthermore, assume that $p_{new}$ and $p_{old}$ both are equal to the **converted** success rate in the `df2` data regardless of the page. So, our assumption is:
$p_{new} = p_{old} = p_{population}$
In this section, you will:

- Simulate (bootstrap) sample data set for both groups, and compute the "converted" probability $p$ for those samples.
- Use a sample size for each group equal to the ones in the `df2` data.
- Compute the difference in the "converted" probability for the two samples above.
- Perform the sampling distribution for the "difference in the converted probability" between the two simulated-samples over 10,000 iterations; and calculate an estimate.

Use the cells below to provide the necessary parts of this simulation. 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 hypothesis?

```
In [19]: p_new = df2.query('converted ==1')['converted'].count()/ df2.shape[0]
         p_new

Out[19]: 0.11959708724499628
```

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

```
In [20]: p_old = df2.query('converted ==1')['converted'].count()/ df2.shape[0]
         p_old

Out[20]: 0.11959708724499628
```

**c.** What is $n_{new}$, the number of individuals in the treatment group? *Hint*: The treatment group users are shown the new page.

```
In [21]: n_new = df2.query('group == "treatment"').shape[0]
         n_new

Out[21]: 145310
```

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

```
In [22]: n_old = df2.query('group == "control"').shape[0]
         n_old

Out[22]: 145274
```

**e. Simulate Sample for the `treatment` Group** Simulate $n_{new}$ transactions with a conversion rate of $p_{new}$ under the null hypothesis.

```
In [23]: # Simulate a Sample for the treatment Group

         new_page_converted = np.random.choice([0,1], size= n_new , p = [(1-p_new), (p_new)])
         new_page_converted.mean()

Out[23]: 0.11979216846741449
```

**f. Simulate Sample for the `control` Group** Simulate $n_{old}$ transactions with a conversion rate of $p_{old}$ under the null hypothesis. Store these $n_{old}$ 1's and 0's in the `old_page_converted` numpy array.

```
In [24]: # Simulate a Sample for the control Group

         old_page_converted = np.random.choice([0,1], size= n_old , p = [(1-p_old), (p_old)])
         old_page_converted.mean()

Out[24]: 0.11990445640651459
```

**g.** Find the difference in the "converted" probability $(p'_{new} - p'_{old})$ for your simulated samples from the parts (e) and (f) above.

```
In [25]: new_page_converted.mean() - old_page_converted.mean()

Out[25]: -0.00011228793910010582
```

**h. Sampling distribution** Re-create `new_page_converted` and `old_page_converted` and find the $(p'_{new} - p'_{old})$ value 10,000 times using the same simulation process you used in parts (a) through (g) above.

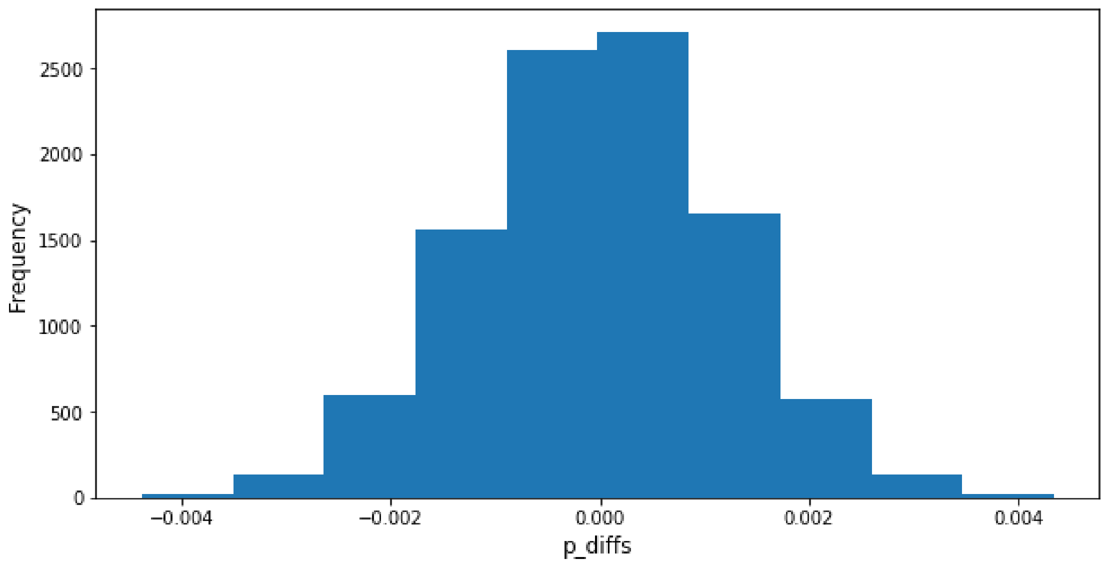
Store all $(p'_{new} - p'_{old})$ values in a NumPy array called `p_diffs`.

```
In [26]: # Sampling distribution
         p_diffs = []

         for _ in range (10000):
             new_page_converted = np.random.choice([0,1], size= n_new , p = [(1-p_new), (p_new)]
             old_page_converted = np.random.choice([0,1], size= n_old , p = [(1-p_old), (p_old)]
             # append the info
             p_diffs.append(new_page_converted.mean() - old_page_converted.mean())

         p_diffs = np.array(p_diffs)
```

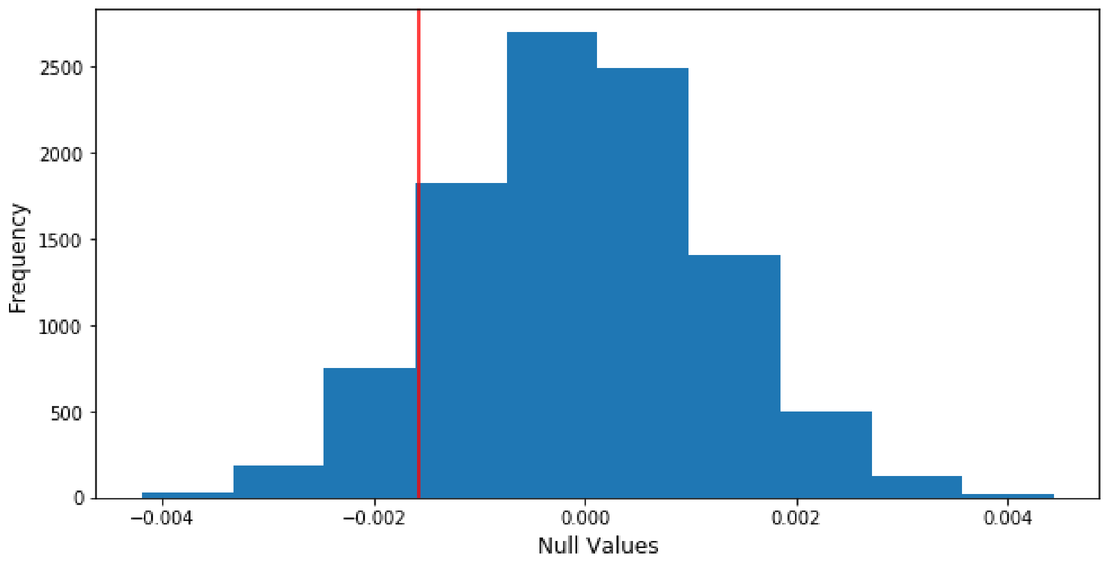
**i. Histogram** Plot a histogram of the **p_diffs**. Does this plot look like what you expected? Use the matching problem in the classroom to assure you fully understand what was computed here.

Also, use `plt.axvline()` method to mark the actual difference observed in the `df2` data (recall `obs_diff`), in the chart.

```
In [27]: plt.figure(figsize=[10,5])
         plt.hist(p_diffs)
         plt.xlabel('p_diffs', fontsize = 12)
         plt.ylabel('Frequency' , fontsize = 12);
```



```
In [28]: #Get the standard deviation:
         np.std(p_diffs)

Out[28]: 0.0012075896296745265

In [29]: # obs_diff(actual_diff) = p_treatment - p_control :

         obs_diff = (len(df2.query('group == "treatment" and converted == 1'))/ len(df2.query('g
         obs_diff

Out[29]: -0.0015782389853555567

In [30]: # Create a random sample for the null values:
         null_values = np.random.normal(0, np.std(p_diffs), 10000)

         #plot the null values:
         plt.figure(figsize=[10,5])
         plt.hist(null_values)
         plt.axvline(obs_diff, color='r')
         plt.xlabel('Null Values', fontsize = 12)
         plt.ylabel('Frequency' , fontsize = 12);
```



**j.** What proportion of the **p_diffs** are greater than the actual difference observed in the `df2` data?

```
In [31]: (p_diffs > obs_diff).mean()

Out[31]: 0.9011000000000001

In [32]: # Or
         (null_values> obs_diff).mean()

Out[32]: 0.9006999999999995
```

**k.** Please explain in words what you have just computed in part **j** above.
- What is this value called in scientific studies?
- What does this value signify in terms of whether or not there is a difference between the new and old pages? *Hint*: Compare the value above with the "Type I error rate (0.05)".

**This value is called (P_value) in scientific studies**

**We can notice that "p_value (0.907)" is greater than "Type I error rate (0.05)", So we fail to reject the null hypothesis**

**l. Using Built-in Methods for Hypothesis Testing** 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 walk-through of the ideas that are critical to correctly thinking about statistical significance.

Fill in the statements below to calculate the: - `convert_old`: number of conversions with the old_page - `convert_new`: number of conversions with the new_page - `n_old`: number of individuals who were shown the old_page - `n_new`: number of individuals who were shown the new_page

```
In [33]: import statsmodels.api as sm
```

```
/opt/conda/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56: FutureWarning: The panda
  from pandas.core import datetools
```

```
In [34]: # number of conversions with the old_page
         convert_old = df2.query('converted == 1 and landing_page == "old_page"').shape[0]

         # number of conversions with the new_page
         convert_new = df2.query('converted == 1 and landing_page == "new_page"').shape[0]

         # number of individuals who were shown the old_page
         n_old = df2.query('landing_page == "old_page"').shape[0]

         # number of individuals who received new_page
         n_new = df2.query('landing_page == "new_page"').shape[0]

         convert_old, convert_new, n_old, n_new
```

```
Out[34]: (17489, 17264, 145274, 145310)
```

**m.** Now use `sm.stats.proportions_ztest()` to compute your test statistic and p-value. Here is a helpful link on using the built in.
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. >**Hint**: It's a two-tailed if you defined $H_1$ as $(p_{new} = p_{old})$. It's a left-tailed if you defined $H_1$ as $(p_{new} < p_{old})$. It's a right-tailed if you defined $H_1$ as $(p_{new} > p_{old})$.
The built-in function above will return the z_score, p_value.

> **Tip**: You don't have to dive deeper into z-test for this exercise. **Try having an overview of what does z-score signify in general.**

```
In [45]: import statsmodels.api as sm
         # ToDo: Complete the sm.stats.proportions_ztest() method arguments
         z_score, p_value = sm.stats.proportions_ztest([convert_new, convert_old], [n_new, n_old
         print(z_score, p_value)
```

```
-1.31092419842 0.905058312759
```

**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.**?

> **Tip**: Notice whether the p-value is similar to the one computed earlier. Accordingly, can you reject/fail to reject the null hypothesis? It is important to correctly interpret the test statistic and p-value.

**The result of z-test = - 1.31 which is lower than the ctritical value 1.96**

**Also p_value = 0.905 which is still greater than the (type i error rate) = 0.05**

**So the result of z-score and p-value agree with the findings in part j and k**

**We fail to reject the null hypothesis**

### Part III - A regression approach

### 1.0.7 ToDo 3.1

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 in the `df2` data is either a conversion or no conversion, what type of regression should you be performing in this case?

**We will use logistic regression in this case because there are only two possible outcomes [0, 1] and each row in the df2 data is either a conversion or no conversion**

**b.** The goal is to use **statsmodels** library to fit the regression model you specified in part **a.** above to see if there is a significant difference in conversion based on the page-type a customer receives. However, you first need to create the following two columns in the `df2` dataframe: 1. `intercept` - It should be `1` in the entire column. 2. `ab_page` - It's a dummy variable column, having a value `1` when an individual receives the **treatment**, otherwise `0`.

```
In [36]: df2['intercept'] =1
         df2[['ab_page','new']]= pd.get_dummies(df2['group'])
         df2 = df2.drop('new', axis =1)
         df2.head()
```

```
Out[36]:    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

            intercept  ab_page
         0          1        1
         1          1        1
         2          1        0
         3          1        0
         4          1        1
```

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

```
In [37]: logit_mod = sm.Logit(df2['converted'], df2[['intercept', 'ab_page']])


         #Fit the model:
         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 [38]: results.summary2()
```

```
Out[38]: <class 'statsmodels.iolib.summary2.Summary'>
         """
                                   Results: Logit
         ==================================================================
         Model:              Logit            No. Iterations:   6.0000
         Dependent Variable: converted        Pseudo R-squared: 0.000
         Date:               2022-04-24 11:34 AIC:              212780.3502
         No. Observations:   290584           BIC:              212801.5095
         Df Model:           1                Log-Likelihood:   -1.0639e+05
         Df Residuals:       290582           LL-Null:          -1.0639e+05
         Converged:          1.0000           Scale:            1.0000
         -------------------------------------------------------------------
                       Coef.   Std.Err.      z      P>|z|    [0.025   0.975]
         -------------------------------------------------------------------
         intercept    -2.0038    0.0081  -247.1457  0.0000  -2.0197  -1.9879
         ab_page       0.0150    0.0114     1.3109  0.1899  -0.0074   0.0374
         ==================================================================

         """
```

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

**p-value associated with ab_page is 0.1899 and it differs from the results in Part II beacause in part II null hypothesis is what we assuemed to be true before collecting any data [p_old = p_new] , in regression p-values are testing if the parameter for the intercept or ab_page(slope) are equal to zero and the null hypothesis , and alternative hypothesis by default is not equal to zero, while in part II the null values were computing the difference between the new and old pages**

**p-value (0.1899) is greater than the Type I error rate(0.05) so we fail to reject null hypothesis**

**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?

**It is a good idea to consider other things that might influence whether or not an individual converts as till now we failed to reject the new pages and we gonna see if other things could change or mind or support our decision, but using additional irrelevant terms in our regression may mislead us and screw up our anaysis**

**g. Adding countries** Now along with testing if the conversion rate changes for different pages, also add an effect based on which country a user lives in.

1. You will need to read in the **countries.csv** dataset and merge together your `df2` datasets on the appropriate rows. You call the resulting dataframe `df_merged`. Here are the docs for joining tables.
2. Does it appear that country had an impact on conversion? To answer this question, consider the three unique values, `['UK', 'US', 'CA']`, in the `country` column. Create dummy variables for these country columns. >**Hint:** Use `pandas.get_dummies()` to create dummy variables. **You will utilize two columns for the three dummy variables.**

Provide the statistical output as well as a written response to answer this question.

```
In [39]: # Read the countries.csv
         df3 = pd.read_csv('countries.csv')
         df3.head()
```

```
Out[39]:    user_id country
         0   834778      UK
         1   928468      US
         2   822059      UK
         3   711597      UK
         4   710616      UK
```

```
In [40]: # Join with the df2 dataframe
         frames = [df2.set_index('user_id'), df3.set_index('user_id')]
         df_merged = pd.concat(frames, axis =1 , join= 'inner')
         df_merged.head()
```

```
Out[40]:                          timestamp      group landing_page  converted  \
         user_id
         851104   2017-01-21 22:11:48.556739    control     old_page          0
         804228   2017-01-12 08:01:45.159739    control     old_page          0
         661590   2017-01-11 16:55:06.154213  treatment     new_page          0
         853541   2017-01-08 18:28:03.143765  treatment     new_page          0
         864975   2017-01-21 01:52:26.210827    control     old_page          1

                  intercept  ab_page country
         user_id
         851104           1        1      US
         804228           1        1      US
         661590           1        0      US
         853541           1        0      US
         864975           1        1      US
```

```
In [41]: # Create the necessary dummy variables
         #dummy variable columns, having a value 1 when an individual receives the treatment, ot
         df_merged[['UK','US','CA']] = pd.get_dummies(df_merged['country'])
         df_merged = df_merged.drop('country', axis=1)
         df_merged.head()
```

```
Out[41]:                           timestamp      group landing_page  converted  \
         user_id
         851104   2017-01-21 22:11:48.556739    control     old_page          0
         804228   2017-01-12 08:01:45.159739    control     old_page          0
         661590   2017-01-11 16:55:06.154213  treatment     new_page          0
         853541   2017-01-08 18:28:03.143765  treatment     new_page          0
         864975   2017-01-21 01:52:26.210827    control     old_page          1

                  intercept  ab_page  UK  US  CA
         user_id
         851104           1        1   0   0   1
         804228           1        1   0   0   1
         661590           1        0   0   0   1
         853541           1        0   0   0   1
         864975           1        1   0   0   1
```

**h. Fit your model and obtain the results** 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 are there significant effects on conversion. **Create the necessary additional columns, and fit the new model.**

Provide the summary results (statistical output), and your conclusions (written response) based on the results.

```
In [42]: #Adding the necessary columns:
         df_merged['ab_page_UK'] = df_merged['ab_page'] * df_merged['UK']
         df_merged['ab_page_US'] = df_merged['ab_page'] * df_merged['US']
         df_merged['ab_page_CA'] = df_merged['ab_page'] * df_merged['CA']
         df_merged.head()
```

```
Out[42]:                           timestamp      group landing_page  converted  \
         user_id
         851104   2017-01-21 22:11:48.556739    control     old_page          0
         804228   2017-01-12 08:01:45.159739    control     old_page          0
         661590   2017-01-11 16:55:06.154213  treatment     new_page          0
         853541   2017-01-08 18:28:03.143765  treatment     new_page          0
         864975   2017-01-21 01:52:26.210827    control     old_page          1

                  intercept  ab_page  UK  US  CA  ab_page_UK  ab_page_US  ab_page_CA
         user_id
         851104           1        1   0   0   1           0           0           1
         804228           1        1   0   0   1           0           0           1
         661590           1        0   0   0   1           0           0           0
         853541           1        0   0   0   1           0           0           0
         864975           1        1   0   0   1           0           0           1
```

```
In [46]: # Fit your model, and summarize the results
         logit = sm.Logit(df_merged['converted'], df_merged[['intercept','ab_page', 'ab_page_UK'
         results = logit.fit()
         results.summary2()
```

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

```
Out[46]: <class 'statsmodels.iolib.summary2.Summary'>
         """
                                  Results: Logit
         ==================================================================
         Model:              Logit            No. Iterations:   6.0000
         Dependent Variable: converted        Pseudo R-squared: 0.000
         Date:               2022-04-24 11:50 AIC:              212782.6602
         No. Observations:   290584           BIC:              212846.1381
         Df Model:           5                Log-Likelihood:   -1.0639e+05
         Df Residuals:       290578           LL-Null:          -1.0639e+05
         Converged:          1.0000           Scale:            1.0000
         -------------------------------------------------------------------
                       Coef.   Std.Err.      z      P>|z|    [0.025   0.975]
         -------------------------------------------------------------------
         intercept    -2.0070    0.0097  -207.0454  0.0000  -2.0260  -1.9880
         ab_page       0.0206    0.0137     1.5052  0.1323  -0.0062   0.0473
         ab_page_UK    0.0469    0.0538     0.8718  0.3833  -0.0585   0.1523
         ab_page_US   -0.0314    0.0266    -1.1807  0.2377  -0.0835   0.0207
         UK           -0.0644    0.0384    -1.6788  0.0932  -0.1396   0.0108
         US            0.0257    0.0188     1.3634  0.1728  -0.0112   0.0625
         ==================================================================

         """
```

**We can notice that p-value is always greater than 0.05 even after considering new factors**

**So we fail to reject the null hypothesis**

**There is no need to create the new page**

```
In [44]: from subprocess import call
         call(['python', '-m', 'nbconvert', 'Analyze_ab_test_results_notebook.ipynb'])
```

```
Out[44]: 0
```