Market Analytics-HW1

November 10, 2020

```
[1]: import pandas as pd
     import math
     import numpy as np
     import scipy
     import scipy.stats as st
```

```
[2]: booking = pd.read_csv('AB_test_data.csv')
```

0.1 1. Conduct an A/B test to determine whether Alternative B improved conversion rates (site users book the property) over alternative A.

```
[3]: booking.head()
```

```
[3]:    purchase_TF Variant        date        id
     0        False       A  2019-11-08  0x25b44a
     1        False       B  2020-08-27  0x46271e
     2        False       A  2020-06-11  0x80b8f1
     3        False       B  2020-08-22  0x8d736d
     4        False       A  2020-08-05  0x96c9c8
```

```
[4]: booking_A = booking[booking['Variant'] == "A"]
     booking_B = booking[booking['Variant'] == "B"]
```

```
[5]: booking_B
```

```
[5]:         purchase_TF Variant        date        id
     1             False       B  2020-08-27  0x46271e
     3             False       B  2020-08-22  0x8d736d
     59            False       B  2020-08-19  0x3ff83f
     74            False       B  2020-08-10  0x138d19
     103           False       B  2020-08-04  0x966e6a
     ...             ...     ...         ...       ...
     129805        False       B  2020-08-27  0x7d95d0
     129827        False       B  2020-08-10  0x38a6e8
     129879        False       B  2020-08-30  0x6a711a
     129910        False       B  2020-08-13  0x13175a
```

```
129960        False        B  2020-08-02  0x8d0674

[5000 rows x 4 columns]
```

[6]:
```python
#calculate 'True' percentage in A group: p = 0.149616

p = booking_A[(booking_A['purchase_TF'] == True)].id.count()/booking_A.id.
 ↪count()
print(p)
```

```
0.149616
```

[7]:
```python
#calculate 'True' percentage in B group: p_head = 0.1766

p_head = booking_B[(booking_B['purchase_TF'] == True)].id.count()/booking_B.id.
 ↪count()
print(p_head)
```

```
0.1766
```

[8]:
```python
#null hypo: p head = p
#alternative hypo: p head > p
```

[9]:
```python
#Calculate z-score = 5.349273094732516

n = booking_B.shape[0]

numerator = p_head-p
denominator = math.sqrt((p*(1-p))/n)

numerator / denominator
```

[9]: 5.349273094732516

[10]:
```python
# calculate z critical value = 1.6448536269514722

scipy.stats.norm.ppf(1-.05)


#z-score > z critical value, reject null hypothesis.
#There is enough evidence (alpha = 5%) to support the claim that B improves
 ↪conversion rates.
```

[10]: 1.6448536269514722

0.2 2. Calculate the optimal sample size for a 95% confidence rate and test with 80% power. Conduct the test 10 times using samples of the optimal size. Report results.

```
[11]: #confidence rate = 95% --> probability of type I error: 5%
      #power = 80% --> probability of type II error: 20%
```

```
[12]: #delta: difference between the two means. We use 1%.

      p_bar = (p + p_head)/2

      #difference of conversion rate
      optimal_sample = (st.norm.ppf(0.975)*math.sqrt(2*p_bar*(1-p_bar)) + st.norm.
       ⌄ppf(0.8)*math.sqrt(p*(1-p)+p_head*(1-p_head)))**2 /((p_head-p)**2)
      optimal_sample = int(optimal_sample)

      print("The optimal size should be:", optimal_sample)
```

```
The optimal size should be: 2941
```

```
[13]: test1a = booking_A.sample(n = optimal_sample)
      test2a = booking_A.sample(n = optimal_sample)
      test3a = booking_A.sample(n = optimal_sample)
      test4a = booking_A.sample(n = optimal_sample)
      test5a = booking_A.sample(n = optimal_sample)
      test6a = booking_A.sample(n = optimal_sample)
      test7a = booking_A.sample(n = optimal_sample)
      test8a = booking_A.sample(n = optimal_sample)
      test9a = booking_A.sample(n = optimal_sample)
      test10a = booking_A.sample(n = optimal_sample)

      test1b = booking_B.sample(n = optimal_sample)
      test2b = booking_B.sample(n = optimal_sample)
      test3b = booking_B.sample(n = optimal_sample)
      test4b = booking_B.sample(n = optimal_sample)
      test5b = booking_B.sample(n = optimal_sample)
      test6b = booking_B.sample(n = optimal_sample)
      test7b = booking_B.sample(n = optimal_sample)
      test8b = booking_B.sample(n = optimal_sample)
      test9b = booking_B.sample(n = optimal_sample)
      test10b = booking_B.sample(n = optimal_sample)

      a_sample_list = [test1a, test2a, test3a, test4a, test5a, test6a, test7a,
       ⌄test8a, test9a, test10a]
      b_sample_list = [test1b, test2b, test3b, test4b, test5b, test6b, test7b,
       ⌄test8b, test9b, test10b]
```

```
[14]: a_sample_list[5]
```

```
[14]:         purchase_TF Variant        date          id
      114113        False       A  2020-06-13  0x7d03c2
      35995          True       A  2020-03-09  0x57be29
      88795          True       A  2020-02-22  0x337198
      108132        False       A  2020-01-24  0x2c6370
      68647         False       A  2020-05-27  0x1b05ff
      ...             ...     ...         ...       ...
      120578        False       A  2020-01-08  0x47ae2c
      48686         False       A  2019-12-31  0x4561c4
      40935         False       A  2019-09-15  0x781411
      30658         False       A  2019-08-28  0x2c5574
      38313         False       A  2019-08-14  0x6e825d

      [2941 rows x 4 columns]
```

```
[15]: a_mean_list = []
      b_mean_list = []

      for i in a_sample_list:
          a_mean_list.append(float(np.mean(i["purchase_TF"])))

      for i in b_sample_list:
          b_mean_list.append(float(np.mean(i["purchase_TF"])))

      mean_diff_list = []
      for i in range(10):
          mean_diff_list.append(b_mean_list[i] - a_mean_list[i])
```

```
[16]: mean_diff_list
```

```
[16]: [0.020401224073444413,
       0.01700102006120366,
       0.017341040462427737,
       0.022441346480788826,
       0.027201632097925865,
       0.03230193811628698,
       0.03060183611016662,
       0.01394083645018701,
       0.026521591295477737,
       0.021421285277116647]
```

```
[17]: sigma_list = []

      for i in range(10):
          p = a_mean_list[i]
```

```
        sigma_list.append(math.sqrt((p*(1-p))/n))
```

[18]:
```
sigma_list
```

[18]:
```
[0.005123348154191923,
 0.005109586931587035,
 0.0051551578387677675,
 0.005001444675672144,
 0.005049045360393573,
 0.005104982555093356,
 0.0050110379014435885,
 0.005141576198164069,
 0.0050443613294714,
 0.005146111887704192]
```

[19]:
```
z_score_list = []
for i in range(10):
    z_score_list.append(mean_diff_list[i]/sigma_list[i])
```

[20]:
```
z_score_list
```

[20]:
```
[3.9820100956348505,
 3.327278758309167,
 3.36382337937741,
 4.4869728520535785,
 5.387480237611789,
 6.327531537607041,
 6.10688578135695,
 2.7113935324278464,
 5.257707490848245,
 4.162615532767441]
```

[21]:
```
significance_list = []
for i in z_score_list:
    if i > scipy.stats.norm.ppf(1-.05):
        significance_list.append(True)
    else:
        significance_list.append(False)
```

[22]:
```
significance_list
```

[22]:
```
[True, True, True, True, True, True, True, True, True, True]
```

[23]:
```
for i in significance_list:
    if i == True:
        print('WOW!!significant improvement!')
    else:
```

```
        print('oops')
```

```
WOW!!significant improvement!
WOW!!significant improvement!
WOW!!significant improvement!
WOW!!significant improvement!
WOW!!significant improvement!
WOW!!significant improvement!
WOW!!significant improvement!
WOW!!significant improvement!
WOW!!significant improvement!
WOW!!significant improvement!
```

0.3   3. Conduct a sequential test for the 10 samples. For any of the samples, were you able to stop the test prior to using the full sample? What was the average number of iterations required to stop the test?

```
[24]: upper_bound = np.log(1/(1-0.95))
      lower_bound = np.log(1-0.8)
      rounds_ran = []
      reason = []

      for i in range(10):
          print(i)
          log_gamma = 0
          rounds = 0

          while (log_gamma > lower_bound) & (log_gamma < upper_bound):
              if rounds < optimal_sample:
                  if b_sample_list[i]['purchase_TF'].values[rounds]:
                      log_gamma = log_gamma + math.log(p_head / p)
                  else:
                      log_gamma = log_gamma + math.log((1-p_head) / (1-p))
                  rounds += 1
              else:
                  reason.append('Did not stop early')
                  break

          rounds_ran.append(rounds)
          if log_gamma < lower_bound:
              reason.append('Lower bound')
          elif log_gamma > upper_bound:
              reason.append('Higher bound')
```

```
0
1
```

```
2
3
4
5
6
7
8
9
```

[25]: `rounds_ran`

[25]: `[2622, 2941, 1353, 1052, 705, 554, 1233, 542, 1587, 195]`

[26]: `reason`

[26]:
```
['Higher bound',
 'Did not stop early',
 'Higher bound',
 'Higher bound',
 'Lower bound',
 'Higher bound',
 'Higher bound',
 'Higher bound',
 'Higher bound',
 'Lower bound']
```

[35]:
```python
from statistics import mean
mean(rounds_ran)
```

[35]: `1278.4`

[37]:
```python
#(mean(rounds_ran[0:1]+rounds_ran[2:]))
```

[37]: `1093.6666666666667`

[29]:
```python
#We're able to stop the test 9 out of 10 times, with an average of stopping at
 ↪~1278 rounds/iterations.
#Or if not including the one time not stopping early and going to 2317 rounds,
 ↪then an average of ~1094 rounds/iterations

#Note, that this varies greatly when rerunning with different seeds/
 ↪randomizations.
#This run is a pretty balanced one in terms of having just one non-stopping
 ↪early,
#and 2 lower bounds as well as 7 upper bound reasons for stopping
```

[ ]:

```
[30]: print(upper_bound)
      print(lower_bound)
```

```
2.99573227355399
-1.6094379124341005
```

[ ]: