ECON626_Lab 2

Qingwen Wang

1.1 Simulating basic distributions

1. For Bernoulli distribution with p=0.25, the sample mean(=0.2516) matches the theoretical mean(=0.25);

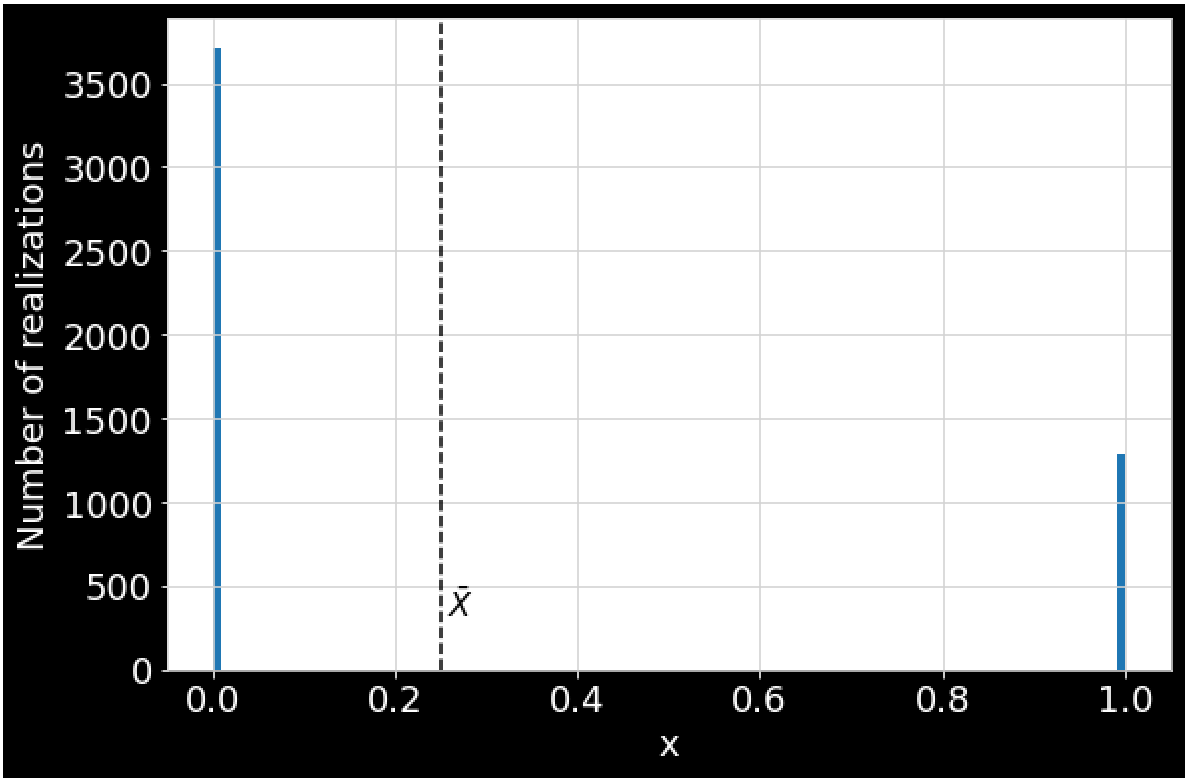
```
p = 0.25
n = 5000
Yi = np.random.binomial(n=1, p=p, size=n)
print("Shape = ", Yi.shape)
print("Average of 5000 draws = {}".format(Yi.mean()))
```

```
Shape =  (5000,)
Average of 500 draws = 0.2516
```

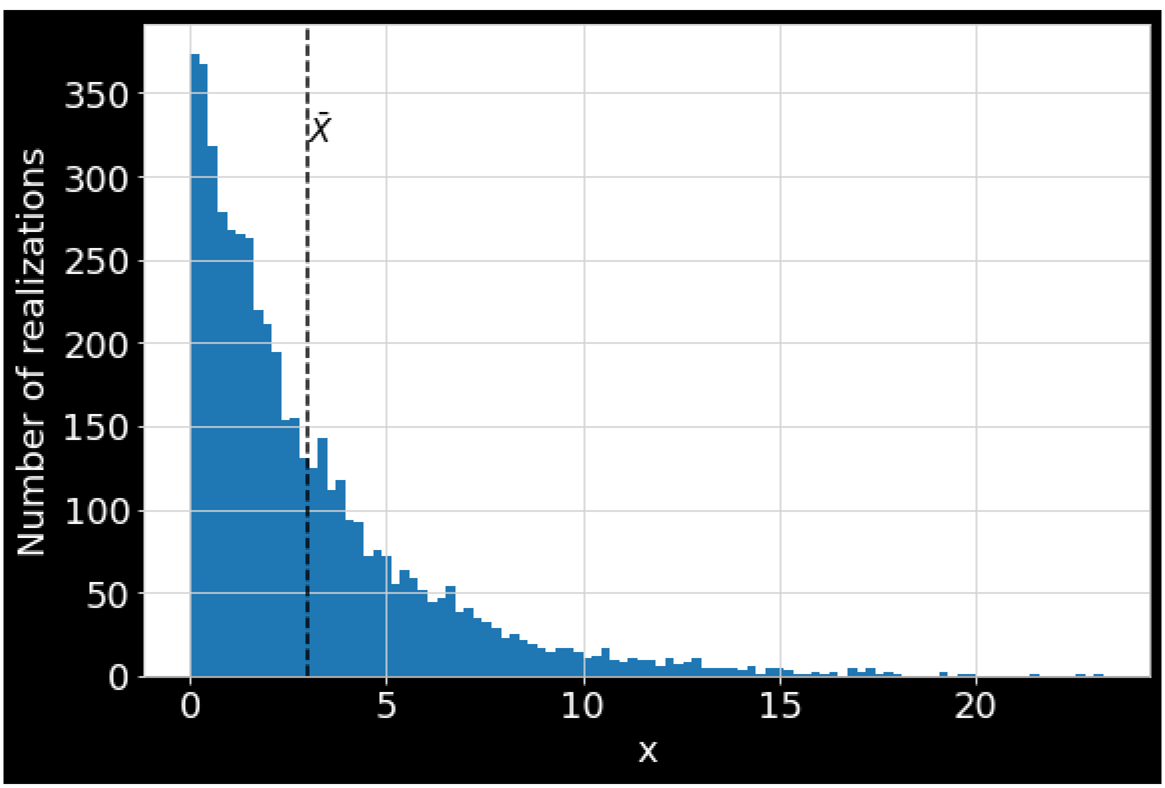
For exponential distribution with scale=3, the sample mean is 2.95 matches the theoretical mean

```
n = 5000
Yi = np.random.exponential(scale = 3,size = n)
print("Exponetial distribution mean = {}".format(Yi.mean()))
```

```
Exponetial distribution mean = 2.953656692339521
```



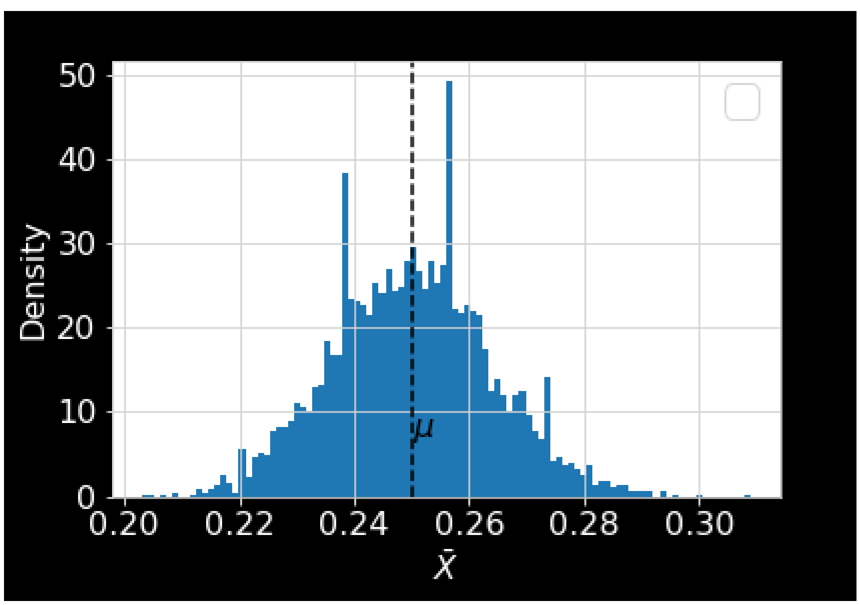
Bernoulli distribution



Exponential distribution

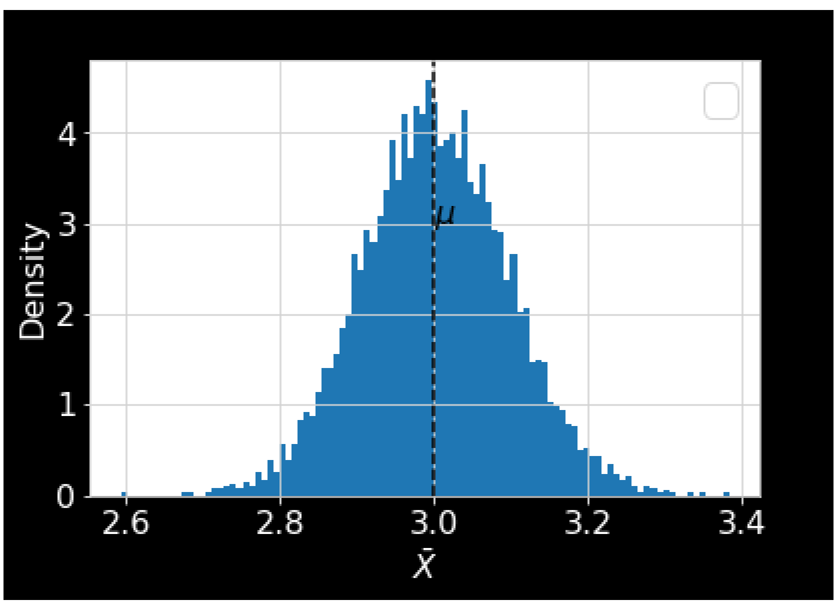
2. For Bernoulli one:

```
sample_means_bi = []
list_of_samples = [np.random.binomial(n = 1, p = 0.25, size=10
00) for _ in progressbar(range(0, 5000))]
sample_means_bi = [sample.mean() for sample in list_of_sample
s]
sample_means_bi = pd.Series(sample_means_bi)


ax = sample_means_bi.hist(bins=100, density=True)
ax.axvline(0.25, color='black', linestyle='--')
ax.text(x=0.25, y=7, s=r'$\mu$', fontsize=16)
ax.set_xlabel(r'$\bar{X}$', fontsize=16)
ax.set_ylabel('Density', fontsize=16)
ax.set_title("Simulating the sampling distribution of the mea
n", fontsize=16)
ax.legend(fontsize=22)
ax.tick_params(labelsize=16)
```



For exponential one:

```
sample_means_exp = []
list_of_samples = [np.random.exponential(scale=3, size=1000) for _ in progressbar(range(0, 5000))]
sample_means_exp = [sample.mean() for sample in list_of_samples]
sample_means_exp = pd.Series(sample_means_exp)


ax = sample_means_exp.hist(bins=100, density=True)
ax.axvline(3, color='black', linestyle='--')
ax.text(x=3, y=3, s=r'$\mu$', fontsize=16)
ax.set_xlabel(r'$\bar{X}$', fontsize=16)
ax.set_ylabel('Density', fontsize=16)
ax.set_title("Simulating the sampling distribution of the mean", fontsize=16)
ax.legend(fontsize=22)
ax.tick_params(labelsize=16)
```



1.2 Simulation experiment statistics

1. sd = 0.12537020597118997

```
sample_results['diff'].std()
```

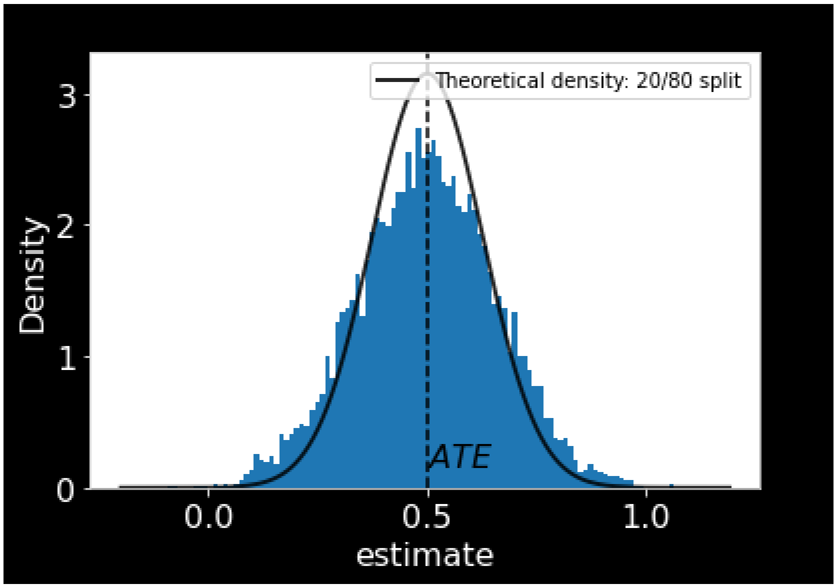
2.

i) 20% treatment, 80% control

```
share_treatment = 0.2
```

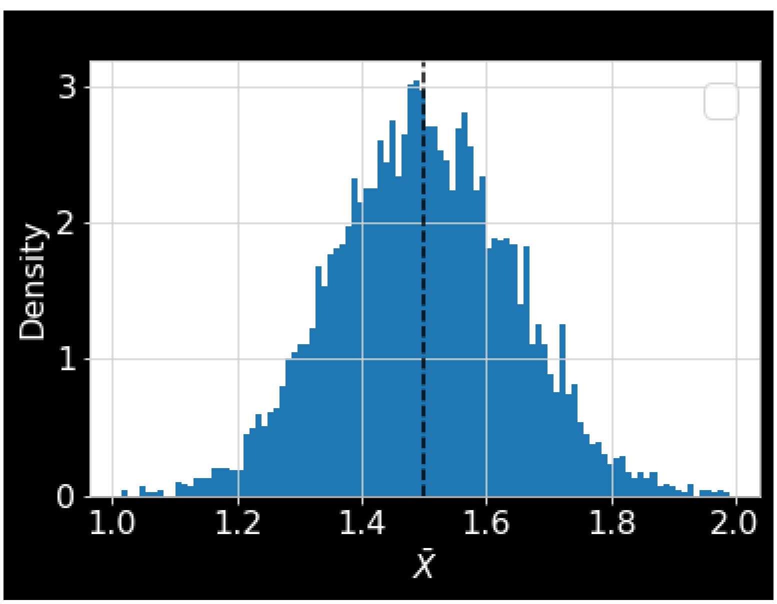
new sd = 0.1592574947996461

```
sample_results['diff'].std()
```

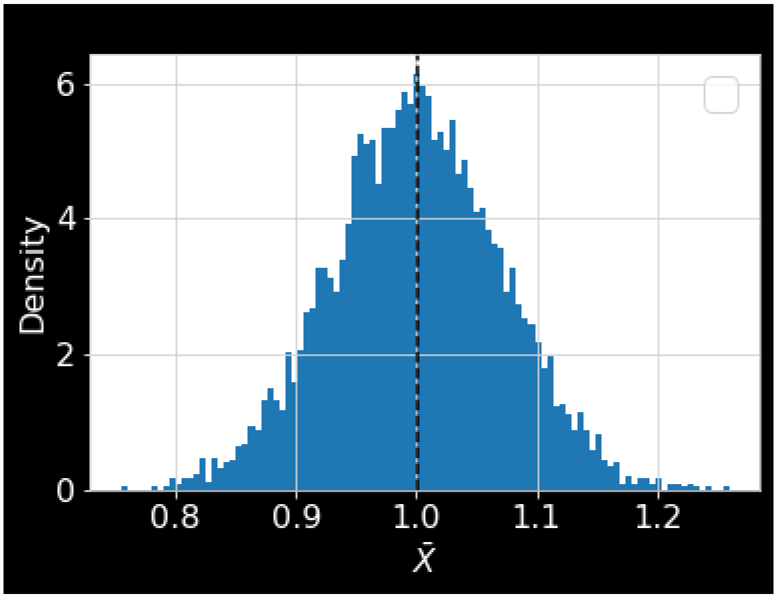


ii)

Histogram of sample means of treatment group:



Histogram of sample means of control group:



iii)
After split dataset in 2:8 ratio, the standard deviation goes up by 0.03. so the noise become larger.
iv)
The best way is to do 50/50, because the standard deviation is the minimum when they are equal-sized treatment and control groups.

3. Theoretically, the probability P(=0.7) is expected to be between $P[\mu + 1.5\sigma]$ and $P[\mu + 1.6\sigma]$, $P \in [0.0548, 0.06681]$
   In this case, the probability is 6.06%, which matches the theory.

```
sum(sample_results["diff"].apply(lambda x: x>0.7))/sample_results.shape[0]
```
```
0.0606
```

4. Theoretically, it should be expected to be around 0.05 (2.37% above + 2.37% below).
   In this case, the probability is 0.0492(0.0248 above + 0.0244 below).

```
p_right = sum(sample_results["diff"].apply(lambda x: x>0.501696 + 2* 0.128776))/sample_results.shape[0]
p_left = sum(sample_results["diff"].apply(lambda x: x<0.501696 - 2* 0.128776))/sample_results.shape[0]
p = p_left + p_right
print(p_right)
print(p_left)
print(p)
```

```
0.0248
0.0244
0.0492
```

2 Simulating t-tests under the null

1. False positive rate is the probability of that when the null hypothesis is true, we still reject the null. When it refers to data-generating process, the false positive rate is the probability of generating a data set which causes the test to reject.
2.

```
B = 5000
N = 1000
ATE = 0.0
expectation_Y0 = 3
parameter_grid = [
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.10, sigma = 1, alpha = 0.05),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.25, sigma = 1, alpha = 0.05),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.05),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.75, sigma = 1, alpha = 0.05),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.9, sigma = 1, alpha = 0.05),
]
simulation_results =  pd.DataFrame([get_simulation_results(**params) for params in parameter_grid])
# We can also test that the simulations worked as expected.
# For example:
```

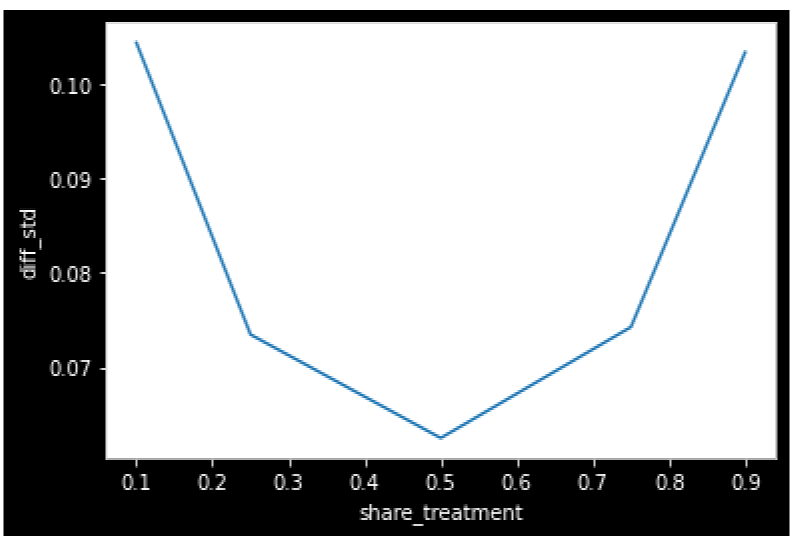
```
assert((simulation_results['share_treatment'] == simulation_results.eval("N_treatment / N")).all())
```

```
100% (5000 of 5000) |####################| Elapsed Time: 0:00:14 Time:  0:00:14
100% (5000 of 5000) |####################| Elapsed Time: 0:00:18 Time:  0:00:18
100% (5000 of 5000) |####################| Elapsed Time: 0:00:14 Time:  0:00:14
100% (5000 of 5000) |####################| Elapsed Time: 0:00:18 Time:  0:00:18
100% (5000 of 5000) |####################| Elapsed Time: 0:00:14 Time:  0:00:14
100% (5000 of 5000) |####################| Elapsed Time: 0:00:18 Time:  0:00:18
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:18 Time:  0:00:18
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
```

```
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	0.001095	0.104408	0.009950	0.991391	0.05	0.0482	5000	1000.0	100.0	0.10	3	1	0.0
1	0.001034	0.073447	0.014726	1.006775	0.05	0.0496	5000	1000.0	250.0	0.25	3	1	0.0
2	0.000826	0.062468	0.013783	0.988779	0.05	0.0474	5000	1000.0	500.0	0.50	3	1	0.0
3	0.000721	0.074246	0.009568	1.017340	0.05	0.0554	5000	1000.0	750.0	0.75	3	1	0.0
4	0.000400	0.103388	0.003127	0.982135	0.05	0.0434	5000	1000.0	900.0	0.90	3	1	0.0

The difference becomes more closer to zero and always near to zero; t is near to zero; p_reject is around 0.05
The standard deviation seems to be affected by the treatment portion; it reaches minimum when treatment share is 0.5, and get bigger as the share increase to 0.9 or decrease to 0.1.



3. change n to 100, 1000, 10000, and set share_treatment = 0.5

```
B = 5000
ATE = 0.0
expectation_Y0 = 3
parameter_grid = [
    dict(B = B, n = 100, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.05),
    dict(B = B, n = 1000, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.05),
    dict(B = B, n = 10000, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.05),

]

simulation_results =  pd.DataFrame([get_simulation_results(**params) for params in parameter_grid])

# We can also test that the simulations worked as expected.
# For example:
assert((simulation_results['share_treatment'] == simulation_results.eval("N_treatment / N")).all())
```

```
100% (5000 of 5000) |####################| Elapsed Time: 0:00:14 Time:  0:00:14
100% (5000 of 5000) |####################| Elapsed Time: 0:00:19 Time:  0:00:19
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
100% (5000 of 5000) |####################| Elapsed Time: 0:00:21 Time:  0:00:21
100% (5000 of 5000) |####################| Elapsed Time: 0:00:20 Time:  0:00:20
```

```
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	-0.002906	0.204803	-0.015613	1.035187	0.05	0.0598	5000	100.0	50.0	0.5	3	1	0.0
1	0.000323	0.063941	0.005378	1.011839	0.05	0.0526	5000	1000.0	500.0	0.5	3	1	0.0
2	-0.000254	0.019703	-0.012768	0.984979	0.05	0.0428	5000	10000.0	5000.0	0.5	3	1	0.0

4. change Y0 to 10, sigma to 20;

```
B = 5000
ATE = 0.0
expectation_Y0 = 10
parameter_grid = [
    dict(B = B, n = 1000, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 20, alpha = 0.05),

]

simulation_results =  pd.DataFrame([get_simulation_results(**params) for params in parameter_grid])

# We can also test that the simulations worked as expected.
# For example:
assert((simulation_results['share_treatment'] == simulation_results.eval("N_treatment / N")).all())
```

```
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
```

```
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	-0.014943	1.266107	-0.011992	1.000725	0.05	0.0508	5000	1000.0	500.0	0.5	10	20	0.0

The rejection rate is changed.

5. by setting alpha to 0.1, p_reject is now close to 0.1 , as shown in the results.

```python
B = 5000
ATE = 0.0
expectation_Y0 = 3
parameter_grid = [
    dict(B = B, n = 100, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.1),
    dict(B = B, n = 1000, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.1),
    dict(B = B, n = 10000, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.1),

]

simulation_results =  pd.DataFrame([get_simulation_results(**params) for params in parameter_grid])

# We can also test that the simulations worked as expected.
# For example:
assert((simulation_results['share_treatment'] == simulation_results.eval("N_treatment / N")).all())
```

```
100% (5000 of 5000) |####################| Elapsed Time: 0:00:14 Time:  0:00:14
100% (5000 of 5000) |####################| Elapsed Time: 0:00:18 Time:  0:00:18
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:18 Time:  0:00:18
100% (5000 of 5000) |####################| Elapsed Time: 0:00:19 Time:  0:00:19
100% (5000 of 5000) |####################| Elapsed Time: 0:00:20 Time:  0:00:20
```

```python
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	0.001483	0.199598	0.006851	1.009888	0.1	0.1012	5000	100.0	50.0	0.5	3	1	0.0
1	0.001165	0.063247	0.018460	1.000462	0.1	0.0980	5000	1000.0	500.0	0.5	3	1	0.0
2	-0.000085	0.019895	-0.004181	0.994973	0.1	0.1018	5000	10000.0	5000.0	0.5	3	1	0.0

6.
the false positive rate is the threshold set by the analyst to determine whether we should reject the null hypothesis or not. It does not depends on sample size.
The simulation results show that in our simulation about 5% of the data sets causes the *t*-test to reject if we set alpha to 0.05; and about 10% of the data sets causes the *t*-test to reject if we set alpha to 0.1. Therefore, our test is performing as expected.

3 Simulating t-test when there is an ATE

1. Power is the probability of making a correct decision (to reject the null hypothesis) when the null hypothesis is false. It is not necessarily under control of analyst, instead, it is determined by sample size and effect size.
2. Set N = 1000, E[Y 0] = 3, AT E = 0.15, σ = 1, share_treatment = 0.5, and six α values = 0.01, 0.05, 0.10, 0.15, 0.25, 0.35

```python
B = 5000
N = 1000
ATE = 0.15
expectation_Y0 = 3
parameter_grid = [
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.01),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.05),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.10),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.15),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.25),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.35),

]

simulation_results =  pd.DataFrame([get_simulation_results(**params) for params in parameter_grid])
```

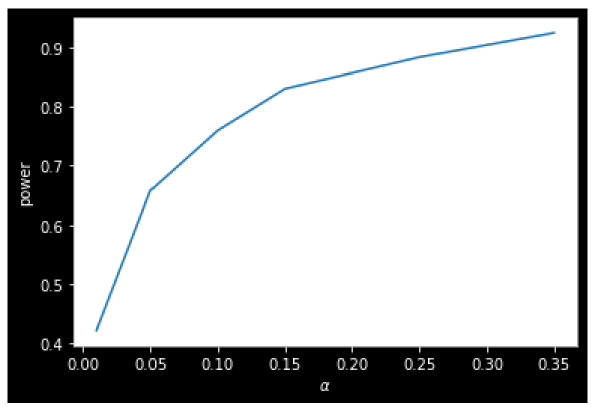
```
100% (5000 of 5000) |####################| Elapsed Time: 0:00:14 Time:  0:00:14
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
100% (5000 of 5000) |####################| Elapsed Time: 0:00:14 Time:  0:00:14
100% (5000 of 5000) |####################| Elapsed Time: 0:00:18 Time:  0:00:18
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:18 Time:  0:00:18
100% (5000 of 5000) |####################| Elapsed Time: 0:00:14 Time:  0:00:14
100% (5000 of 5000) |####################| Elapsed Time: 0:00:18 Time:  0:00:18
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:18 Time:  0:00:18
```

```python
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	0.150338	0.063655	2.378397	1.007294	0.01	0.4210	5000	1000.0	500.0	0.5	3	1	0.15
1	0.150916	0.063530	2.388018	1.006459	0.05	0.6580	5000	1000.0	500.0	0.5	3	1	0.15
2	0.149540	0.063630	2.364903	1.007474	0.10	0.7600	5000	1000.0	500.0	0.5	3	1	0.15
3	0.151888	0.063340	2.402936	1.002928	0.15	0.8306	5000	1000.0	500.0	0.5	3	1	0.15
4	0.150415	0.063971	2.379998	1.014304	0.25	0.8838	5000	1000.0	500.0	0.5	3	1	0.15
5	0.149297	0.063020	2.361851	0.998286	0.35	0.9250	5000	1000.0	500.0	0.5	3	1	0.15

```python
[16] df_power = simulation_results.iloc[:,[4,5]]
     df_power
```

	alpha	p_reject
0	0.01	0.4210
1	0.05	0.6580
2	0.10	0.7600
3	0.15	0.8306
4	0.25	0.8838
5	0.35	0.9250



As the alpha increases, the simulated power increases at a decreasing rate.

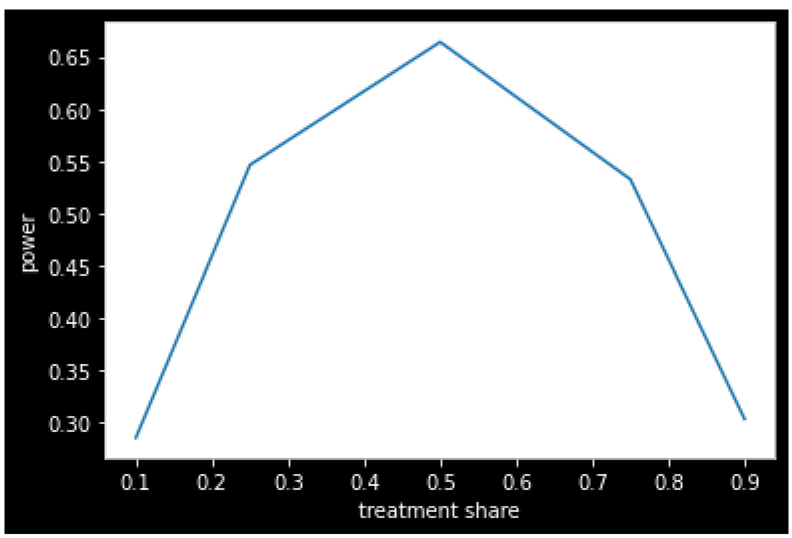
3.

```
[36] B = 5000
     N = 1000
     ATE = 0.15
     expectation_Y0 = 3
     parameter_grid = [
         dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.10, sigma = 1, alpha = 0.05),
         dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.25, sigma = 1, alpha = 0.05),
         dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5, sigma = 1, alpha = 0.05),
         dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.75, sigma = 1, alpha = 0.05),
         dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.9, sigma = 1, alpha = 0.05),
     ]

     simulation_results =  pd.DataFrame([get_simulation_results(**params) for params in parameter_grid])
```

```
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
```

```
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	0.147775	0.104081	1.402181	0.988245	0.05	0.2858	5000	1000.0	100.0	0.10	3	1	0.15
1	0.150782	0.073031	2.066063	1.001771	0.05	0.5464	5000	1000.0	250.0	0.25	3	1	0.15
2	0.149854	0.063161	2.372688	1.001878	0.05	0.6640	5000	1000.0	500.0	0.50	3	1	0.15
3	0.149534	0.072900	2.049740	1.001026	0.05	0.5326	5000	1000.0	750.0	0.75	3	1	0.15
4	0.151238	0.106745	1.435114	1.013905	0.05	0.3038	5000	1000.0	900.0	0.90	3	1	0.15



50/50 is the best share in this case.

4.

False positive ("Type I Error"): The test said there is an ATE ("rejected the null") but really ATE=0. False negative ("Type II Error"): The test said ATE=0 ("did not reject') but really ATE!=0. The inverse of the false negative rate is power. Thus, to reaches the maximum utility, our main rule is to lower typer I rule, and higher power.

Scenario 1: alpha = 0.01

According to the question, a full launch have a very large up-front cost, hence, we want to make false positive rate as small as possible, which means when our experiment shows there exists ATE and it does exist. So set alpha = 0.01 indicates that we can reject H0 at 99% confidence, which is a very strong evidence to reject the null.
Scenario 2: alpha = 0.05
Since we do not have additional cost(beyond the cost of the A/B test), we are not very sensitive to make type I error. So we can use higher alpha to get higher power and reduce required sample size.

5.
i) the simulated power matches with the calculated power = 0.66

```
[56] simulated_power = simulation_results.query('share_treatment==0.50')['p_reject'].iloc[0]
     # Example of how to calculate the power for a given point.
     calculated_power = power.tt_ind_solve_power(
         effect_size=ATE / 1,
         alpha=0.05,
         nobs1=N / 2,
         ratio=1.0,
     )

     # Compare these two!
     print("Rejection rate from power calculator = ", calculated_power)
     print("Simulated rejection rate using p-values = ", simulated_power)
```

```
Rejection rate from power calculator =  0.6589069549458351
Simulated rejection rate using p-values =  0.664
```

ii) we can increase the sample size to increase power.
example: keep all the parameters same as previous(ate= 0.3, sigma = 1, alpha = 0.05, ratio = 1.0), except changes N from 1000 to 1400, then we can make the power reaches 0.8

```
calculated_power = power.tt_ind_solve_power(
    effect_size=ATE / 1,
    alpha=0.05,
    nobs1=1400 / 2,
    ratio=1.0,
)
calculated_power
```

```
0.8007642341642731
```

Bonus 1:

if 10%treatment, 90%control, we can still get 0.8power requirement when sample size reaches 6981.

```
[74] for N in range(6500,8000):
       calculated_power = power.tt_ind_solve_power(
           effect_size=ATE / 1,
           alpha=0.05,
           nobs1=N / 2,
           ratio=1/9,
     )
       if calculated_power >= 0.8:
         print('when N = {}, calculated_power = {}, reaches the 0.8 requirement'.format(N, calculated_power))
         break
```

```
when N = 6981, calculated_power = 0.8000436150483817, reaches the 0.8 requirement
```

Bonus 2:

if we lower our false positive rate to 0.01, we will need much bigger sample size(nearly 1.5 times) to secure the power. Here we need sample size to be at least 10388.

```
[84] for N in range(8000,12000):
       calculated_power = power.tt_ind_solve_power(
           effect_size=ATE / 1,
           alpha=0.01,
           nobs1=N / 2,
           ratio=1/9,
     )
       if calculated_power >= 0.8:
         print('when N = {}, calculated_power = {}, reaches the 0.8 requirement'.format(N, calculated_power))
         break
```

```
when N = 10388, calculated_power = 0.8000332564507875, reaches the 0.8 requirement
```