Causal Inference Assignment 2

Creator: Junjie Lei

CreationDate: Feb_7/2020

Title: *Random Sampling*

---

1. Notebook from class

1.1 Simulating basic distributions class

1.

```
n = 10000
x_i = np.random.binomial(n=1, p = 0.25, size = n )

print("mean for the bionomial = ", x_i.mean())
print('p = 0.25')
print("std for the binomial = ", x_i.std())
```

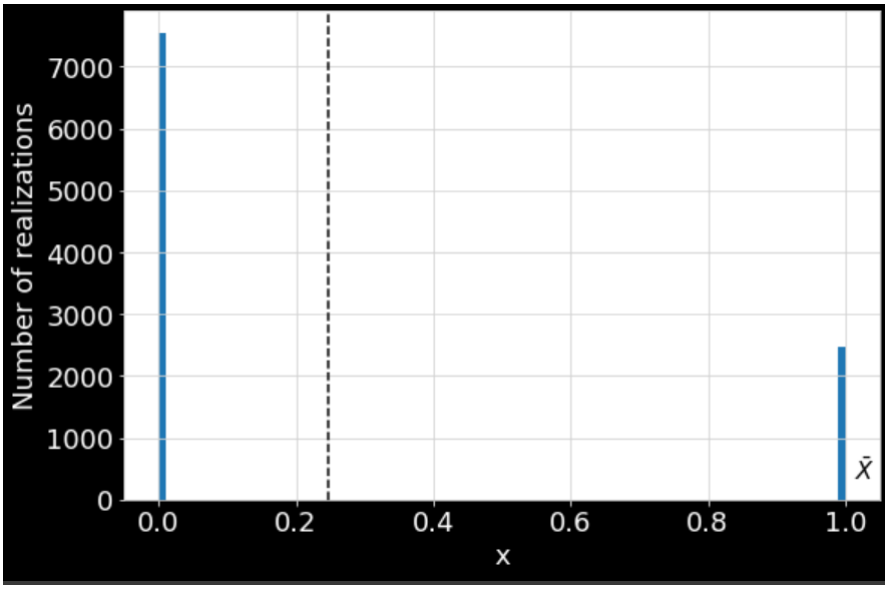
output:

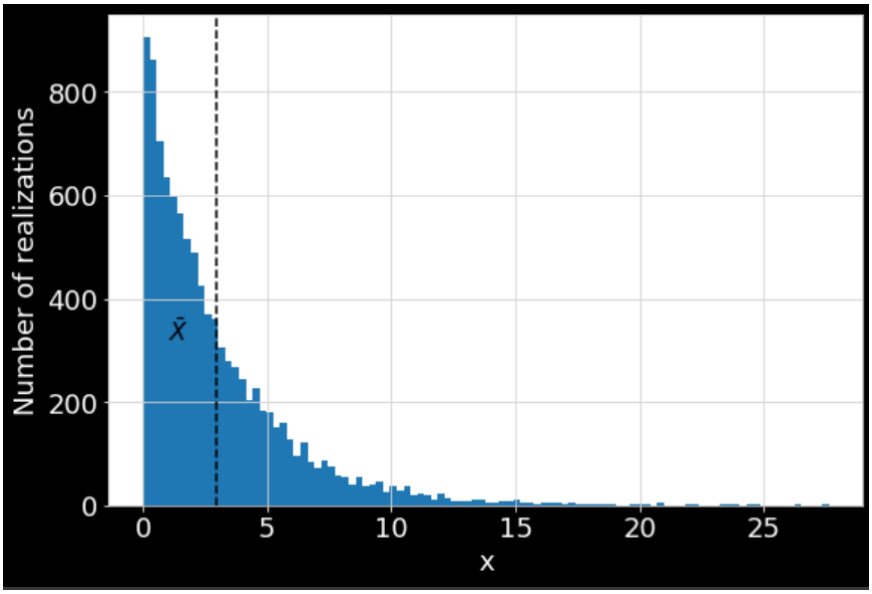
```
mean for the bionomial =  0.246
p = 0.25
std for the binomial =  0.4306785344082056 4
```

```
n = 10000
x_i = np.random.exponential(scale=3.0, size= n )

print("mean for the exp function = ", x_i.mean())
print('p = 0.25')
print("std for the binomial = ", x_i.std())
```

output:

```
mean for the exp function =  2.954894135354989
p = 0.25
std for the binomial =  2.9647568759686362
```

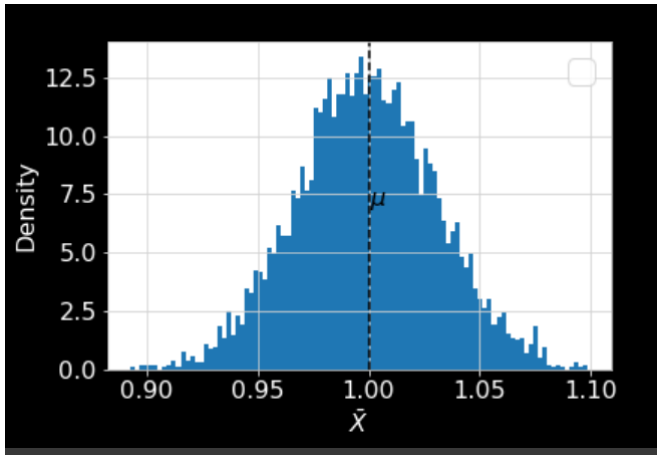




2. plot the underlying mean on the graph

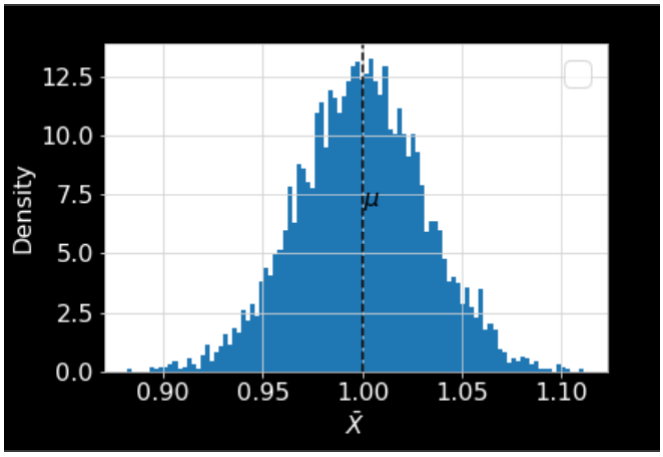
   for binomial plot

```
ax = sample_means_bi.hist(bins=100, density=True)


ax.axvline(mu, color='black', linestyle='--')
ax.text(x=mu, 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 mean", fontsize=16)
ax.legend(fontsize=22)
ax.tick_params(labelsize=16)
```



for exponential

```
ax = sample_means_exp.hist(bins=100, density=True)
'''and other formatting code'''
```



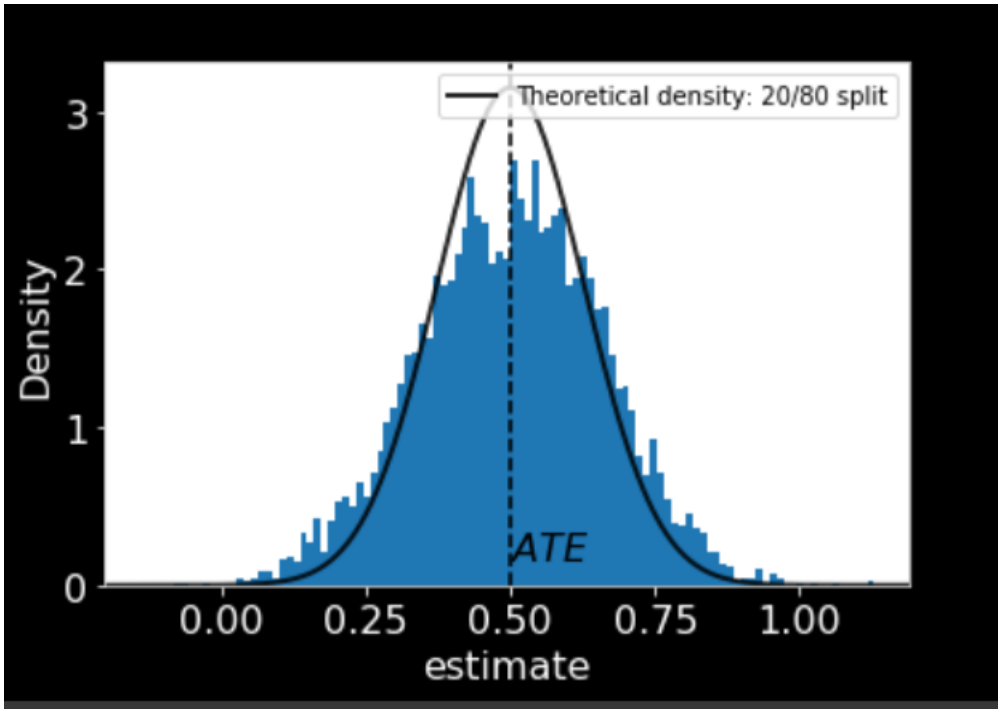
1.2 Simulating experiment statistics

1. the code `sample_results.describe()` showed the standard deviation is 0.127811
2.
    - change the simulation so that the split is always $20\%\&80\%$

      parameter `share_treatment` indicates the the proportion of N in the treatment group

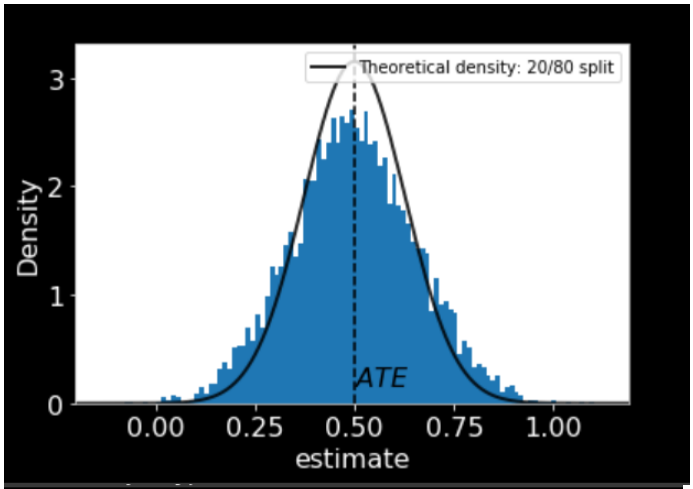
      ```
      share_treatment = 0.2
      ```

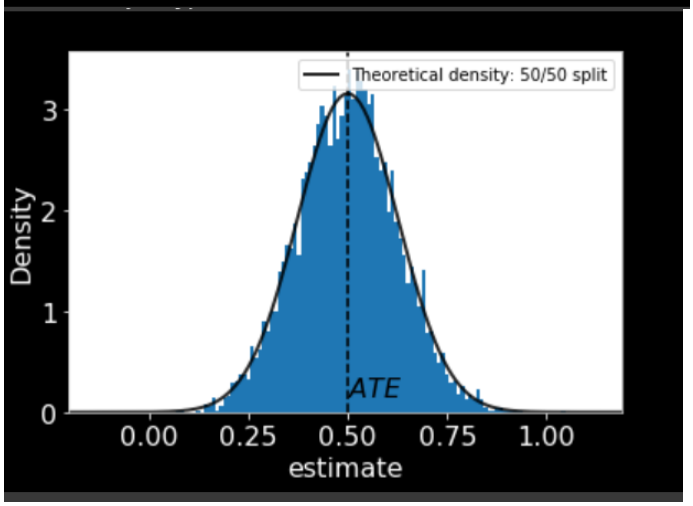
      the plot of the 20% treatment and 80% control



the standard deviation is 0.158808

- the noise `standard deviation` becomes smaller when estimating the `ATE` if we use the $20\%\&80\%$ to split the treatment and control group.





- the best way to split is to do the $50/50$, because the std.ev is the minimum when the shared treatment is split into $50/50$

3. the estimated `ATE` is greater than 0.7

   I used the following to calculate the

```
count_ate = sample_results.groupby(['diff']).apply(lambda x : (x >= 0.7))
count_ate.groupby(['diff']).count().reset_index()
```

   the percentage is `6.06%`

```
1 count_ate = sample_results.groupby(['diff']).apply(lambda x : (x >= 0.7))
```

```
1 count_ate.groupby(['diff']).count().reset_index()
```

	diff	N	N_treatment	mean_treatment	mean_control
0	False	4697	4697	4697	4697
1	True	303	303	303	303

4. 2 standard deviation would be $2 \times 0.128988 = 0.257976$ away from the μ

   hence we can calculate the percentage $\geq 2std$

```
count_ate_1 = sample_results.groupby(['diff']).apply(lambda x : (x >=
0.757976 ))
count_ate_1.groupby(['diff']).count().reset_index()
```

	diff	N	N_treatment	mean_treatment	mean_control
0	False	4891	4891	4891	4891
1	True	109	109	109	109

   if larger than 2 $std.ev()$ it would be `2.18%`

   if lower than 2 $std.ev()$ it would be `2.61%`

```
1 count_ate_1 = sample_results.groupby(['diff']).apply(lambda x : (x <= 0.24202400000000002 ))
2 count_ate_1.groupby(['diff']).count().reset_index()
```

	diff	N	N_treatment	mean_treatment	mean_control
0	False	4869	4869	4869	4869
1	True	131	131	131	131

2. Simulating t-tests under the null

1. $False\text{-}postive\ rate \rightarrow$ the probability that when given the null hypothesis is true and we still reject the null;

when it is refers to the process of generating sample data, it is the probability that this process rejects H_0 when the underlying H_0 simulated is true.

2.

```
# Simulation results over a grid of parameter values
B = 1000  # Total number of *samples* to generate
N = 5000  # The number of data points in each sample

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 = 0.50, 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 = 0.90, 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())
```

---

sample output:

```
# Simulation results over a grid of parameter values
B = 1000  # Total number of *samples* to generate
N = 5000  # The number of data points in each sample

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 = 0.50, 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 = 0.90, 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% (1000 of 1000) |####################| Elapsed Time: 0:00:03 Time:  0:00:03
100% (1000 of 1000) |####################| Elapsed Time: 0:00:03 Time:  0:00:03
100% (1000 of 1000) |####################| Elapsed Time: 0:00:03 Time:  0:00:03
100% (1000 of 1000) |####################| Elapsed Time: 0:00:03 Time:  0:00:03
100% (1000 of 1000) |####################| Elapsed Time: 0:00:03 Time:  0:00:03
100% (1000 of 1000) |####################| Elapsed Time: 0:00:03 Time:  0:00:03
100% (1000 of 1000) |####################| Elapsed Time: 0:00:03 Time:  0:00:03
100% (1000 of 1000) |####################| Elapsed Time: 0:00:03 Time:  0:00:03
100% (1000 of 1000) |####################| Elapsed Time: 0:00:03 Time:  0:00:03
100% (1000 of 1000) |####################| Elapsed Time: 0:00:03 Time:  0:00:03
```

```
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	0.000846	0.047195	0.017554	1.002443	0.05	0.047	1000	5000.0	500.0	0.10	3	1	0.0
1	0.000888	0.033210	0.027412	1.017646	0.05	0.060	1000	5000.0	1250.0	0.25	3	1	0.0
2	0.000458	0.027788	0.016094	0.982130	0.05	0.043	1000	5000.0	2500.0	0.50	3	1	0.0
3	-0.000603	0.032184	-0.018435	0.985299	0.05	0.049	1000	5000.0	3750.0	0.75	3	1	0.0
4	0.001235	0.046500	0.026211	0.986394	0.05	0.050	1000	5000.0	4500.0	0.90	3	1	0.0

- diff becomes more scattered when the shared_treatment goes to 0.75 and 0.9
- rejection rate wave above and down the 0.05.
- the changes of t's distribution is not that significant, no pattern can be observed.

3.

in the following code I changed the n

```
# Simulation results over a grid of parameter values
B = 5000
ATE = 0.0
parameter_grid = [
    dict(B = B, n = 100, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.50, sigma = 1, alpha = 0.05),
    dict(B = B, n = 1000, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.50, sigma = 1, alpha = 0.05),
    dict(B = B, n = 10000, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.50, 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: 0:00:12 Time:  0:00:12
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
100% (5000 of 5000) |####################| Elapsed Time: 0:00:20 Time:  0:00:20
100% (5000 of 5000) |####################| Elapsed Time: 0:00:19 Time:  0:00:19
```

```
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	-0.002460	0.201860	-0.013404	1.017506	0.05	0.0522	5000	100.0	50.0	0.5	3	1	0.0
1	-0.000550	0.064695	-0.009456	1.023659	0.05	0.0582	5000	1000.0	500.0	0.5	3	1	0.0
2	-0.000257	0.019867	-0.012974	0.993154	0.05	0.0496	5000	10000.0	5000.0	0.5	3	1	0.0

4.

in the following code I changed the `sigma` & $E[Y^0]$

```
# Simulation results over a grid of parameter values
B = 5000
parameter_grid = [
    dict(B = B, n = 1000, expectation_Y0 = 10, ATE = 0, share_treatment = 0.50, sigma = 20, alpha = 0.05)
]

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

```
100% (5000 of 5000) |####################| Elapsed Time: 0:00:12 Time:  0:00:12
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
```

```
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	-0.000605	1.222237	-0.000518	0.966573	0.05	0.039	5000	1000.0	500.0	0.5	10	20	0

- yes, the rejection rate changed

5.

- in the code chunk, the α determines the *false-positive rate*

  hence in the following code I changed the `n (sample size)` and the *rejection rate* at the same time

```
# Simulation results over a grid of parameter values
B = 5000
ATE = 0.0
parameter_grid = [
    dict(B = B, n = 100, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.50, sigma = 1, alpha = 0.10),
    dict(B = B, n = 1000, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.50, sigma = 1, alpha = 0.10),
    dict(B = B, n = 10000, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.50, sigma = 1, alpha = 0.10)
]

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: 0:00:12 Time:  0:00:12
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
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
```

ble-click (or enter) to edit

```
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	3.392370e-03	0.196623	0.017549	0.995949	0.1	0.0922	5000	100.0	50.0	0.5	3	1	0.0
1	-6.721616e-04	0.063538	-0.010018	1.004942	0.1	0.1024	5000	1000.0	500.0	0.5	3	1	0.0
2	-8.617276e-07	0.019945	0.000063	0.997285	0.1	0.1022	5000	10000.0	5000.0	0.5	3	1	0.0

6. the false positive rate is the threshold analysts set to determine whether we should reject a null hypothesis or not.

   in the simulation results we got above, we can see that the distribution changes while we change some of the parameters. so we need to determine upfront a threshold to evaluate the distribution or the test results.

3. Simulating t-tests when there is an ATE

1. Power is the probability when we rejects a hypothesis test with effect size when, in fact, it is false. That is, Power is the probability of making a correct decision (to reject the null hypothesis) when the null hypothesis is false. It is often affected by sample size, effect size, and alpha. But not necessarily under the control of analyst.

2. here is the code chunk that I modified; it takes really long time to simulate the process;

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

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

---

sample Input/Output:

```
[109]
B = 5000
N = 1000
ATE = 0.15
expectation_Y0 = 3
sigma = 1
share_treatment = 0.5
parameter_grid = [
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = share_treatment, sigma = sigma, alpha = 0.01),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = share_treatment, sigma = sigma, alpha = 0.05),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = share_treatment, sigma = sigma, alpha = 0.10),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = share_treatment, sigma = sigma, alpha = 0.15),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = share_treatment, sigma = sigma, alpha = 0.25),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = share_treatment, sigma = sigma, alpha = 0.35),
]

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: 0:00:16 Time:  0:00:16
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
```

```
[110]
simulation_results
```

	diff_mean	diff_std	t_mean	t_std	alpha	p_reject	B	N	N_treatment	share_treatment	expectation_Y0	sigma	ATE
0	0.151559	0.062807	2.397434	0.994210	0.01	0.4248	5000	1000.0	500.0	0.5	3	1	0.15
1	0.150658	0.064350	2.384004	1.020261	0.05	0.6618	5000	1000.0	500.0	0.5	3	1	0.15
2	0.148645	0.062799	2.351433	0.994394	0.10	0.7618	5000	1000.0	500.0	0.5	3	1	0.15
3	0.149827	0.063112	2.369287	0.998810	0.15	0.8234	5000	1000.0	500.0	0.5	3	1	0.15
4	0.149837	0.063490	2.371402	1.006367	0.25	0.8912	5000	1000.0	500.0	0.5	3	1	0.15
5	0.149808	0.063507	2.371854	1.007592	0.35	0.9254	5000	1000.0	500.0	0.5	3	1	0.15

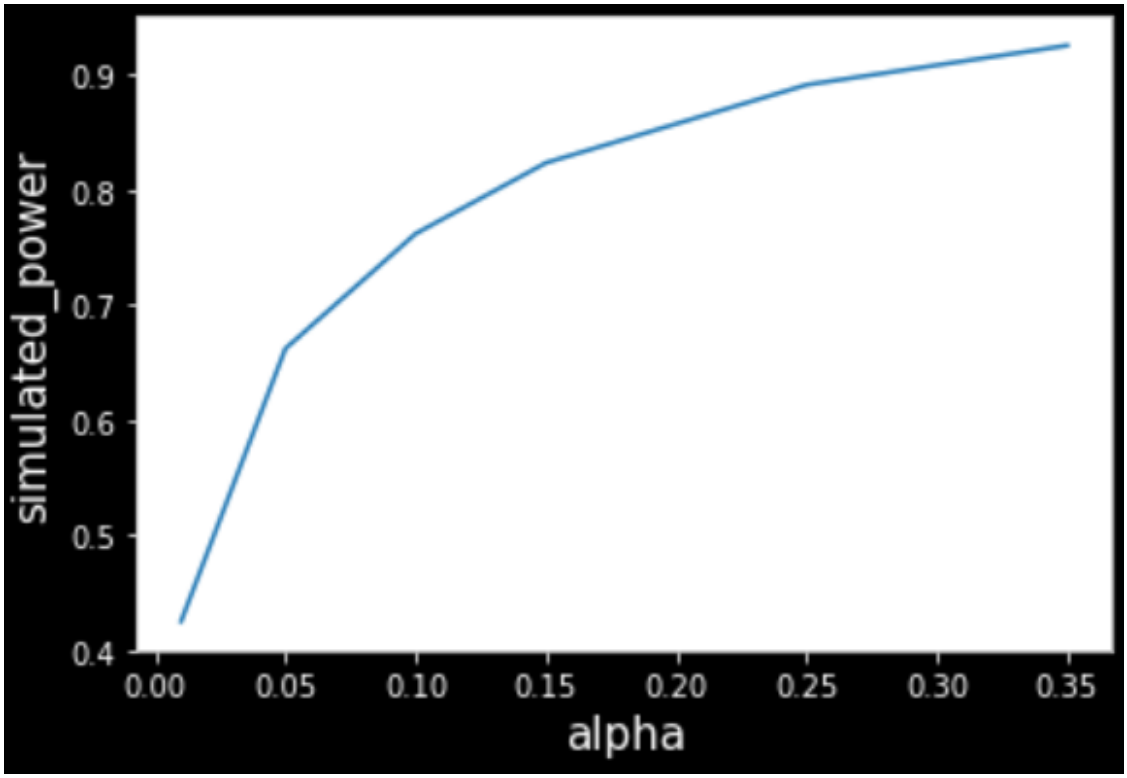
as the α goes up, we can see that the P_{reject} parameter increase but at a decrease rate;

- simulated power is p_reject; that is the mean of probability of reject the null hypothesis
- false positive rate is set as alpha

here is the plot between the relationship between the alpha and power

code and output

```
plt.plot(simulation_results['alpha'], simulation_results['p_reject'])
font2 = {'family' : 'Times New Roman',
'weight' : 'normal',
'size'   : 15,
}
plt.xlabel('alpha',font2)
plt.ylabel('simulated_power',font2)

plt.show()
```



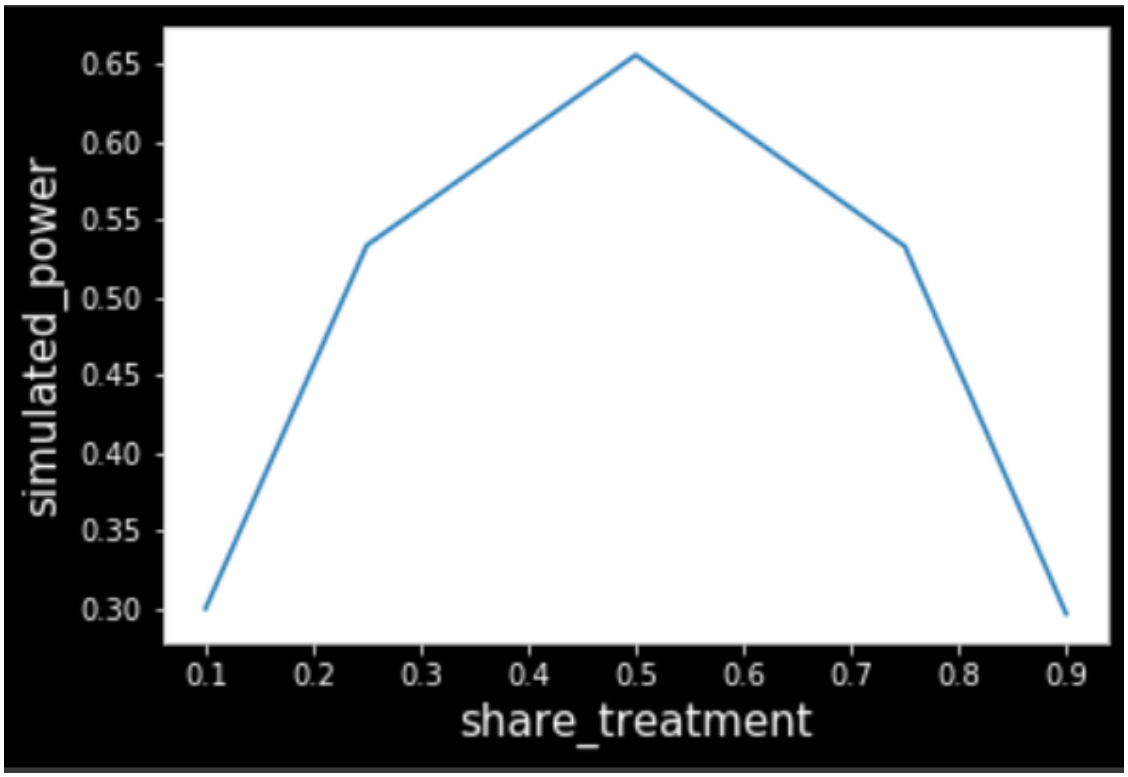
3. in this task, I vary the treatment group percentage

```
B = 5000
N = 1000
ATE = 0.15
expectation_Y0 = 3
sigma = 1
alpha = 0.05
parameter_grid = [
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE,
share_treatment = 0.1, sigma = sigma, alpha = alpha),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE,
share_treatment = 0.25, sigma = sigma, alpha = alpha),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE,
share_treatment = 0.5, sigma = sigma, alpha = alpha),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE,
share_treatment = 0.75, sigma = sigma, alpha = alpha),
    dict(B = B, n = N, expectation_Y0 = expectation_Y0, ATE = ATE,
share_treatment = 0.9, sigma = sigma, alpha = alpha),
]

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

---

and the code for plot & plot shows that 50/50 still the best in this circumstance;

```
plt.plot(simulation_results['share_treatment'],
simulation_results['p_reject'])
font2 = {'family' : 'Times New Roman',
'weight' : 'normal',
'size'   : 15,
}
plt.xlabel('share_treatment',font2)
plt.ylabel('simulated_power',font2)

plt.show()
```



4.

- Under Scenario 1

Because the false positive rate is the probability that rejects the null hypothesis in favor of a false alternative hypothesis, that is the probability of a Type I error. The power is the probability that rejects the null hypothesis when, in fact, it is false. That is, the probability of denying the Type II error. Therefore Power equals to (1-probability(Type II error)). According to the question, a full launch have a very large up-front cost, hence, the product manager's desire outcome must be the most utility of this A/B Test. Therefore, we want both the probability of Type I error and of Type II error as small as possible. It means that power should be bigger and alpha should be smaller.

- However, as we plot above, power gets smaller while alpha gets smaller. So we have to find the balance between this two statistics. We should notice that margin value when alpha changes from 0.01 to 0.05 is the max.
- also, in statistical history, the default value of alpha is 0.05.

In summary, alpha = 0.05 is our best choice.

- Under Scenario 2

in this circumstance, the false positive rate I recommend is 0.01.

As we discussed in (i), we want false positive rate to to the smallest and the power to be biggest.

Because we can launch without any additional costs, that is, we can expand the size of population that participant in this A/B Test (in this case is N and B) to enlarge the value of power. Therefore, we do not have to sacrifice false positive rate to get higher power. We can choose the smallest value of the false positive rate to get the max utility.

5.

1.

```
simulated_power = simulation_results.query('alpha==0.05')['p_reject']
[1]
# Example of how to calculate the power for a given point.
calculated_power = power.tt_ind_solve_power(
    effect_size=ATE / sigma,
    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)
```

---

output:

```
Rejection rate from power calculator =  0.6589069549458351
Simulated rejection rate using p-values =  0.5334
```

2.

```


# Other way to increase power without changing the false positive rate
is to increasing the size of the samples.
# Or we can also decreasing the effect size to enlarge the value of
power.
# nobs means the size of the treatment group = the control group
(share_treatment=0.5)
nobs = power.tt_ind_solve_power(effect_size = ATE/sigma, alpha =0.05,
power=0.8 )

print (nobs)

B = 5000
N = 1500
ATE = 0.15
expectation_Y0 = 3
sigma = 1
alpha = 0.05
# expectation_Y0, ATE, share_treatment, n, B, sigma, alpha
```

```
simulation_results_new = get_simulation_results(B = B, n = N,
expectation_Y0 = expectation_Y0, ATE = ATE, share_treatment = 0.5,
sigma = sigma, alpha = alpha)

simulated_power = simulation_results_new['p_reject']
# Example of how to calculate the power for a given point.
calculated_power = power.tt_ind_solve_power(
    effect_size=ATE / sigma,
    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)
```

output

```
  1% (64 of 5000) |                    | Elapsed Time: 0:00:00 ETA:   0:00:12698.6381616833869
100% (5000 of 5000) |####################| Elapsed Time: 0:00:13 Time:  0:00:13
100% (5000 of 5000) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
Rejection rate from power calculator =  0.8271373121535427
Simulated rejection rate using p-values =  0.829
```

- Other way to increase power without changing the false positive rate is to increasing the size of the samples or we can also decreasing the effect size to enlarge the value of power.

Bonus

1.

- need the `math` package to run the following chunk of code

```
# nobs means the size of the treatment group therefore the control group =
9*nobs (share_treatment=0.1)
import math

nobs = power.tt_ind_solve_power(effect_size = ATE/sigma, alpha =0.05, power=0.8,
ratio=9 )
print(nobs)
N = math.ceil(nobs/0.1)
print(N)
# because nobs = size of treatment group

B = 5000
ATE = 0.15
expectation_Y0 = 3
sigma = 1
alpha = 0.05
# expectation_Y0, ATE, share_treatment, n, B, sigma, alpha
simulation_results_new = get_simulation_results(B = B, n = N, expectation_Y0 =
expectation_Y0, ATE = ATE, share_treatment = 0.1, sigma = sigma, alpha = alpha)

simulated_power = simulation_results_new['p_reject']
```

```
# Example of how to calculate the power for a given point.
calculated_power = power.tt_ind_solve_power(
    effect_size=ATE / sigma,
    alpha=0.05,
    nobs1=N/10,
    ratio=9,
)

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

output:

```
  1% (64 of 5000) |                    | Elapsed Time: 0:00:00 ETA:   0:00:13387.7902228571874
3878
100% (5000 of 5000) |####################| Elapsed Time: 0:00:15 Time:  0:00:15
100% (5000 of 5000) |####################| Elapsed Time: 0:00:17 Time:  0:00:17
Rejection rate from power calculator =  0.8000098930331093
Simulated rejection rate using p-values =  0.8046
```

2.

```
alpha = 0.01
nobs = power.tt_ind_solve_power(effect_size = ATE/sigma, alpha =alpha,
power=0.8, ratio=9 )
print(nobs)
N = math.ceil(nobs/0.1)
print(N)
# because nobs = size of treatment group

B = 5000
ATE = 0.15
expectation_Y0 = 3
sigma = 1
# expectation_Y0, ATE, share_treatment, n, B, sigma, alpha
simulation_results_new = get_simulation_results(B = B, n = N, expectation_Y0 =
expectation_Y0, ATE = ATE, share_treatment = 0.1, sigma = sigma, alpha = alpha)

simulated_power = simulation_results_new['p_reject']
# Example of how to calculate the power for a given point.
calculated_power = power.tt_ind_solve_power(
    effect_size=ATE / sigma,
    alpha=alpha,
    nobs1=N/10,
    ratio=9,
)

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

output

```
  0% (27 of 5000) |                       | Elapsed Time: 0:00:00 ETA:   0:00:19577.071013987634
5771
100% (5000 of 5000) |####################| Elapsed Time: 0:00:20 Time:  0:00:20
100% (5000 of 5000) |####################| Elapsed Time: 0:00:21 Time:  0:00:21
Rejection rate from power calculator =  0.8000240413810171
Simulated rejection rate using p-values =  0.7978
```

The key trade-offs we are making with the product manager is that our aim is to make the A/B test successfully, we have to maintain a power larger than 0.8. In order to make it, we have to choose between false positive rate, size of the samples or effect size.