Situation

A company provides web-based services (e.g., LinkedIn, Yelp, Amazon, ...) and strives to increase active users. Accordingly, the company attempts to have fancy UI designs on its website. We anticipate (and hope) that the new design positively impacts increasing user interactions. To statistically analyze its impact, we do data analysis!

We collect user data for 1000 users over the last 14 days (Actually, we will generate data) and conduct a randomized controlled trial (also known as, A/B testing). That is, we divide users into two groups, treat them differently, and see their outcomes. For more details, let's first see the data generation part.

References

- Prof. Wayne Lee's the **Calculating summary statistics** and **Functions and loops** notebooks. Jupyter notebooks are available at Ed Platform.

Data Generation

- We will generate two 2D `numpy.ndarrays` called `views` and `clicks`. Both arrays have `n=1000` rows and `d=14` columns. Each row and column correspond to a user and a day, respectively.
- We will also create one 1D `numpy.ndarrays` called `assigns` indicating whether the corresponding user has been assigned to a treatment or control group.

Assignments

- Only $p_{\text{assign}} = 5\%$ of the users should be assigned to a treatment group. (We implicitly assume that a user should be assigned to the same treatment/control group across the days!).

Views

- Each user has a $p_{\text{visit}} = 50\%$ chance of visiting each day. If they don't visit, no views are created. For users who visit the website, a user's distribution for views follows the geometric distribution with the parameter $p_{\text{view}} = 0.1$. We here assume that the views are i.i.d. across users and days.

Clicks

- A user's distribution for clicks C_{ij} given their views V_{ij} and their assignment information A_i follows a binomial distribution with the size parameters V_{ij} and the success probability p_{A_i}.

So, p_0 and p_1 indicate the probability of a click given a view for a control group and a treatment group, respectively. We set $p_0 = 0.05$ and $p_1 = 0.1$.

In [1]:
```
import numpy as np
```

[Step 1] Let's create necessary variables called `n`, `d`, `p_assign`, `p_visit`, `p_view`, `p_0`, and `p_1`.

- Their values should be set as in description.

In [2]:
```
n, d=1000, 14
p_assign, p_visit, p_view=0.05, 0.5, 0.1
```

[Step 2] Creata a 1D Numpy array `assign`.

- We can denote assignments by

$$A_1, \ldots, A_n \sim \mathrm{Bern}(p_{\text{assign}}),$$

where $A_i = 1$ indiciates that i-th user has been assigned to a treatment group. Hint: Bernoulli distribution is a specical case of Binomial distribution with the one size parameter. `numpy.random.binomial` [Check official doc.].

In [3]:
```
assign = np.random.binomial(n=1, p=p_assign, size=n)
assign[:10]
```

Out[3]:
```
array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0])
```

[Step 3] Creata a 2D Numpy array `views`.

- We can formalize this as follows. First, we denote the visit indicator by for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots 14\}$,

$$I_{ij} \sim \mathrm{Bern}(p_{\text{visit}}),$$

and the views by

$$\begin{aligned} V_{ij} \mid I_{ij} = 0 &\stackrel{d}{=} 0 \\ V_{ij} \mid I_{ij} = 1 &\sim \mathrm{Geometric}(p_{\text{view}}). \end{aligned}$$

Hint: `numpy.random.geometric`. [Check official doc.].

In [4]:
```
I_mat = np.random.binomial(n=1, p=p_visit, size=(n, d))
views = np.zeros(shape=(n, d))
for i in range(n):
    for j in range(d):
        if I_mat[i,j] != 0:
            views[i,j] = np.random.geometric(p_view)
print(views)
```

```
[[ 0. 17.  0. ...  6.  1. 45.]
 [ 0.  2.  0. ...  2.  4.  3.]
 [ 0.  1.  4. ...  9.  0. 17.]
 ...
 [ 1.  0. 17. ...  0.  0.  0.]
 [ 0.  3.  0. ...  0.  0. 21.]
 [ 0.  0.  0. ...  0.  7.  0.]]
```

[Step 4] Creata a 2D Numpy array `clicks`.

- We can denote it by

$$C_{ij} \mid V_{ij}, A_i \sim \mathrm{Bin}(V_{ij}, p_{A_i}).$$

  - Or more explicitly,

$$C_{ij} \mid V_{ij}, A_i = 0 \sim \mathrm{Bin}(V_{ij}, p_0)$$
$$C_{ij} \mid V_{ij}, A_i = 1 \sim \mathrm{Bin}(V_{ij}, p_1).$$

  Hint: `numpy.random.binomial` [Check official doc.].

In [5]:
```
p_0, p_1 = 0.05, 0.1
p_vector=[p_0, p_1]

clicks = np.zeros(shape=(n, d))
for i in range(n):
    for j in range(d):
        if I_mat[i,j] != 0:
            clicks[i,j] = np.random.binomial(n=views[i,j], p=p_vector[assign[i]], size

clicks, views
```

Out[5]:
```
(array([[0., 3., 0., ..., 0., 0., 4.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 2., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 1.],
        [0., 0., 0., ..., 0., 0., 0.]]),
 array([[ 0., 17.,  0., ...,  6.,  1., 45.],
        [ 0.,  2.,  0., ...,  2.,  4.,  3.],
        [ 0.,  1.,  4., ...,  9.,  0., 17.],
        ...,
        [ 1.,  0., 17., ...,  0.,  0.,  0.],
        [ 0.,  3.,  0., ...,  0.,  0., 21.],
        [ 0.,  0.,  0., ...,  0.,  7.,  0.]]))
```

- Can we combine the two for loops in [Step 3] and [Step 4] into one for loop?

In [6]:
```
I_mat = np.random.binomial(n=1, p=p_visit, size=(n, d))
views = np.zeros(shape=(n, d))
clicks = np.zeros(shape=(n, d))
for i in range(n):
    for j in range(d):
        if I_mat[i,j] != 0:
            views[i,j] = np.random.geometric(p_view)
            clicks[i,j] = np.random.binomial(n=views[i,j], p=p_vector[assign[i]], size
```

```
# clicks, views
```

[Step 5] Let's wrap everything into one function.

- A function called `generate_ab_testing_dataset` takes as input `(n, d, p_assign, p_visit, p_view, p_0, p_1)`.
- Set their default values as the one we used above except `n`. We want to create a function that forces users to enter the `n` value.
- return three Numpy arrays `assign`, `views`, and `clicks`.

In [7]:

```
def generate_ab_testing_dataset(n, d=14,
                                p_assign=0.05,
                                p_visit=0.5,
                                p_view=0.1,
                                p_0=0.05,
                                p_1=0.1):
  # assign = np.random.binomial(n=1, p=p_assign, size=n)
  # I_mat = np.random.binomial(n=1, p=p_visit, size=(n, d))

  assign = np.zeros(shape=(n))
  I_mat = np.zeros(shape=(n, d))
  views = np.zeros(shape=(n, d))
  clicks = np.zeros(shape=(n, d))
  p_vector=[p_0, p_1]

  for i in range(n):
      assign[i] = np.random.binomial(n=1, p=p_assign, size=1)
      for j in range(d):
          I_mat[i,j] = np.random.binomial(n=1, p=p_visit, size=1)
          if I_mat[i,j] != 0:
              views[i,j] = np.random.geometric(p_view)
              clicks[i,j] = np.random.binomial(n=views[i,j], p=p_vector[int(assign[i])

  return assign, views, clicks
```

- [Step 5+] Sanity check
  - `views` and `clicks` should have the same size.
  - For every element, `views` should be greater than or equal to `clicks` (Hint: `np.all`).
  - mean of assignments should be close to `p_assign`.
  - ...

In [8]:

```
n=1000
assign, views, clicks = generate_ab_testing_dataset(n)
print(f'Q1: are they same size? Answer: {views.shape == clicks.shape}')
print(f'Q2: views >= clicks? Answer: {np.all(views >= clicks)}')
print(f'Q3: mean of assign? and true value? Answer: {np.mean(assign)}, {p_assign}')
```

```
Q1: are they same size? Answer: True
Q2: views >= clicks? Answer: True
Q3: mean of assign? and true value? Answer: 0.049, 0.05
```

Data analysis

- To see the impact of a new UI, we will look at **Clickthrough rate (CTR)**, a probability of clicking over 14 days. For $i \in \{1, \ldots, 1000\}$, we define CTR as follows.

$$\text{CTR}_i = \frac{\sum_{j=1}^{14} C_{ij}}{\sum_{j=1}^{14} V_{ij}}$$

- [Step 6] Compute CTRs for every user and create 1D Numpy array called `CTR_array`. Each value `CTR_array` is user's CTR value.
  - Generate datasets using `generate_ab_testing_dataset` with $n = 1000$.
  - Also, compute each group's average CTR and print results as follows. Which group has a higher CTR? treatment or control?

In [9]:
```
assign, views, clicks = generate_ab_testing_dataset(n=1000)
# assert np.all(np.sum(views, axis=1) != 0), 'Sum of views is zero from some user!'
CTR_array = np.sum(clicks, axis=1)/np.sum(views, axis=1)
CTR_array[:10]
```

Out[9]:
```
array([0.02459016, 0.03703704, 0.04587156, 0.04705882, 0.04444444,
       0.01123596, 0.13636364, 0.072     , 0.04255319, 0.        ])
```

In [10]:
```
avg_ctr_treatment=np.mean(CTR_array[assign == 1])
avg_ctr_control=np.mean(CTR_array[assign == 0])
print(f'Average CTR of a treatment group: {avg_ctr_treatment:.3f}')
print(f'Average CTR of a control group: {avg_ctr_control:.3f}')
print(f'Does a treatment group have a higher average CTR?: ', avg_ctr_treatment>avg_ct
```

```
Average CTR of a treatment group: 0.107
Average CTR of a control group: 0.049
Does a treatment group have a higher average CTR?:  True
```

- [Step 7] Identifying Outliers.
  - CTR could be sensitive if the denominator part is small.
  - We define outliers if $\sum_{j=1}^{14} V_{ij}$ is smaller than or equal to 5. (Hint: `np.sum(XXXX, axis= XXX)`
  - print a list of outlier's index (Hint: `np.where()`)

In [11]:
```
# np.sum(views, axis=1) <= 5
np.sum(views, axis=1)[:20]
```

Out[11]:
```
array([122.,  54., 109.,  85.,  90.,  89.,  44., 125.,  94.,  55.,  77.,
        79., 123.,  86.,  99.,  63., 112.,  80.,  76.,  71.])
```

In [12]:
```
outlier_users = np.where(np.sum(views, axis=1) <= 5)[0]
print(outlier_users)
outlier_users, len(outlier_users)
```

```
[]
```

Out[12]:
```
(array([], dtype=int64), 0)
```

- [Step 8] Going back to our original question, we will conduct t-test between a control group and a treatment group. The hypothesis we are interested in can be expressed as

$$H_0 : \mathrm{CTR}_{treatment} = \mathrm{CTR}_{control}, \quad H_1 : \mathrm{CTR}_{treatment} \neq \mathrm{CTR}_{control}.$$

- Write a function called `compute_t_statistics` that outputs `t-statistics` and its `p_value` under the same variance assumption. The function `compute_t_statistics` takes as input two sets $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_m\}$ (you can assume that they are `numpy.ndarray`). Hint: Use `t.cdf` with `from scipy.stats import t` for p-value [Official manual].

- For two sets of samples $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_m\}$, the `t-statistics` is given as

$$T := \frac{\bar{x} - \bar{y}}{\sqrt{s_p^2 \left(\frac{1}{n} + \frac{1}{m}\right)}},$$

  where $\bar{x}$ and $\bar{y}$ is sample average of x and y, and s_p^2 is a pooled variance defined as

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2 + \sum_{i=1}^{m}(y_i - \bar{y})^2}{n + m - 2}.$$

  The t-statistics T follows t-distribution with $n + m - 2$ degrees of freedom under the null hypothesis.

T is a T-distribution with n+m-2 degrees of freedom, t is a t-statistics. P(T > |t|)

In [13]:
```python
def compute_t_statistics(list_a, list_b):
  from scipy.stats import t
  n_a, n_b = len(list_a), len(list_b)
  N = n_a + n_b
  df = N-2
  numerator=np.mean(list_a) - np.mean(list_b)

  pooled_var=(np.sum((list_a-np.mean(list_a))**2) + np.sum((list_b-np.mean(list_b))**2
  normalizing_constant=(1/n_a + 1/n_b)
  t_statistics = numerator / (pooled_var * normalizing_constant)** (0.5)

  cdf_t_statistics = t.cdf(t_statistics, df=df)
  p_value= 2*min(cdf_t_statistics, 1-cdf_t_statistics)
  return t_statistics, p_value
```

In [14]:
```python
assign, views, clicks = generate_ab_testing_dataset(n=1000, p_1=0.1)
CTR_array = np.sum(clicks, axis=1)/np.sum(views, axis=1)
compute_t_statistics(CTR_array[assign == 1], CTR_array[assign == 0])
```

Out[14]:
```
(9.831224624443328, 0.0)
```

- [Step 9] **Sanity Check:** Compare your results with the `from scipy.stats import ttest_ind` [official manual]. Python package `scipy` offers many useful functions.

In [18]:
```
assign, views, clicks = generate_ab_testing_dataset(n=1000, p_1=0.051)
CTR_array = np.sum(clicks, axis=1)/np.sum(views, axis=1)
print(f'Our implementation (t-statistics, p_value): ', compute_t_statistics(CTR_array[

from scipy.stats import ttest_ind
ttest_ind(CTR_array[assign == 1], CTR_array[assign == 0])

CTR_array.shape
```

```
Our implementation (t-statistics, p_value):  (0.5760006335451319, 0.564744616843329)
```

Out[18]:
```
(1000,)
```

In [16]:
```
assign, views, clicks = generate_ab_testing_dataset(n=1000, p_1=0.01)
CTR_array = np.sum(clicks, axis=1)/np.sum(views, axis=1)
print(f'Our implementation (t-statistics, p_value): ', compute_t_statistics(CTR_array[

from scipy.stats import ttest_ind
ttest_ind(CTR_array[assign == 1], CTR_array[assign == 0])
```

```
Our implementation (t-statistics, p_value):  (-8.960148056261772, 1.5606971380042412e
-18)
```

Out[16]:
```
Ttest_indResult(statistic=-8.960148056261772, pvalue=1.5606971380042412e-18)
```

Advanced questions

- Here, we used CTR as the evaluation metric.
  - What would be other good evalutoin metrics? How do you convince others that a new metric is better?
  - What if we use different evaluation metrics?
- We consider t-test to examine the impact of assignment.
  - What could be the potential problem of this approach? If there is, how can we fix this problem?

In [16]:
```

```