AB Testing

Welcome! In this assignment you will be presented with two cases that require an AB test to choose an action to improve an existing product. You will perform AB test for a continuous and a proportion metric. For this you will define functions that estimate the relevant information out of the samples, compute the relevant statistic given each case and take a decision on whether to (or not) reject the null hypothesis.

Let's get started!

```
import math
import numpy as np
import pandas as pd
import scipy.stats as stats
from dataclasses import dataclass
import utils
```

Section 1: Continuous Metric - Average Session Duration

Suppose you have a website that provides machine learning content in a blog-like format. Recently you saw an article claiming that similar websites could improve their engagement by simply using a specific color palette for the background. Since this change seems pretty easy to implement you decide to run an AB test to see if this change does in fact drive your users to stay more time in your website.

The metric you decide to evaluate is the `average session duration`, which measures how much time on average your users are spending on your website. This metric currently has a value of 30.87 minutes.

Without further considerations you decide to run the test for 20 days by randomly splitting your users into two segments:

- `control`: These users will keep seeing your original website.
- `variation`: These users will see your website with the new background colors.

Run the next cell to load the data from the test:

```
# Load the data from the test
data = utils.run_ab_test_background_color(n_days=20)

# Print the first 10 rows
data.head(10)
```

```
      user_id  user_type  session_duration
0  TUI2UNIQL5  variation         15.528769
1  ST76J20B6H  variation         32.287590
2  00K2M0ZRY0  variation         43.718217
3  FIVYVPV4A9  variation         49.519702
4  6TUC4TRA5S    control         61.709028
5  JE0PZKM91P  variation         71.779283
6  PS8DSIW714  variation         23.291835
7  9LUCKQC58C    control         25.219461
8  01QJP4T5W1    control         26.240482
9  QE139J2L7I  variation         20.780244
```

The data shows for every user the average session duration and the version of the website they interacted with. To separate both segments for easier computations you can slice the dataframe by running the following cell:

```
# Separate the data from the two groups (sd stands for session duration)
control_sd_data = data[data["user_type"]=="control"]["session_duration"]
variation_sd_data = data[data["user_type"]=="variation"]["session_duration"]

print(f"{len(control_sd_data)} users saw the original website with an average duration of {control_sd_data.mean():.2f} minutes\n")
print(f"{len(variation_sd_data)} users saw the new website with an average duration of {variation_sd_data.mean():.2f} minutes")

2069 users saw the original website with an average duration of 32.92 minutes

2117 users saw the new website with an average duration of 33.83 minutes
```

Notice that the split is not perfectly balanced. This is common in AB testing as there is randomness associated with the way the users are assigned to each group.

At first glance it looks like the change to the background did in fact drive users to stay longer on your website. However you know better than driving conclusions at face value out of this data so you decide to perform a hypothesis test to know if there is a significant difference between the **means** of these two segments. You can do this by computing the t-statistic and using the null hypothesis that there is **not** a statistically significant difference between the means of the two samples:

$$t=\frac{(\acute{x}_1-\acute{x}_2)-(\mu_1-\mu_2)}{\sqrt{\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2}}}$$

Notice that by computing the metric at a user level you ensure that the independence criteria is met since each user is independent of one another. Also, although the data is not strictly normal you have a large enough sample size to justify the use of the t-test.

But before doing so you will need to compute all the necessary metrics for every group. For this you decide to use a dataclass that holds this information:

```
@dataclass
class estimation_metrics_cont:
    n: int
    xbar: float
    s: float

    def __repr__(self):
        return f"sample_params(n={self.n}, xbar={self.xbar:.3f},
s={self.s:.3f})"
```

This class will hold the information for n, $\acute{x}$ and s.

Exercise 1: compute_continuous_metrics

Now that you have a container for all these metrics it is your job to code a function that given some data will compute them for that particular set of data. To do this complete the `compute_continuous_metrics` below.

Hints:

- np.mean, np.std and len functions are useful.
- When computing the sample standard deviation be sure to use the s estimation: $s=\frac{1}{N-1}\sum_{i=1}^{n}\left(x_i-\overline{x}\right)^2$. To accomplish this you can set the parameter `ddof=1` within the np.std function. This ensures that the denominator of the expression for the standard deviation is `N-1` rather than `N`.

```
def compute_continuous_metrics(data):
    """Computes the relevant metrics out of a sample for continuous
data.

    Args:
        data (pandas.core.series.Series): The sample data. In this
case the average session duration for each user.

    Returns:
        estimation_metrics_cont: The metrics saved in a dataclass
instance.
    """

    ### START CODE HERE ###
    metrics = estimation_metrics_cont(
        n=len(data),
```

```
        xbar=np.mean(data),
        s=np.std(data, ddof=1)
    )
    ### END CODE HERE ###

    return metrics

# Test your code

cm = compute_continuous_metrics(np.array([1,2,3,4,5]))
print(f"n={cm.n}, xbar={cm.xbar:.2f} and s={cm.s:.2f} for example
array\n")

control_metrics = compute_continuous_metrics(control_sd_data)
print(f"n={control_metrics.n}, xbar={control_metrics.xbar:.2f} and
s={control_metrics.s:.2f} for control data\n")

variation_metrics = compute_continuous_metrics(variation_sd_data)
print(f"n={variation_metrics.n}, xbar={variation_metrics.xbar:.2f} and
s={variation_metrics.s:.2f} for variation data")
```

```
n=5, xbar=3.00 and s=1.58 for example array

n=2069, xbar=32.92 and s=17.54 for control data

n=2117, xbar=33.83 and s=18.24 for variation data
```

Expected Output

```
n=5, xbar=3.00 and s=1.58 for example array

n=2069, xbar=32.92 and s=17.54 for control data

n=2117, xbar=33.83 and s=18.24 for variation data
```

Exercise 2: degrees_of_freedom

Another important piece of information when performing a t-test is the degrees of freedom, which can be computed as follows:

$$\text{Degrees of freedom} = \frac{\left[\dfrac{s_1^2}{n_1} + \dfrac{s_2^2}{n_2}\right)^2}{\dfrac{\left(s_1^2/n_1\right)^2}{n_1 - 1} + \dfrac{\left(s_2^2/n_2\right)^2}{n_2 - 1}}$$

Complete the `degrees_of_freedom` function below so that given two samples it will return the degrees of freedom. Notice that this value does not necessarily need to be an integer and can be a float.

Hints:

- Use the `compute_continuous_metrics` function you previously coded to compute the metrics for each sample.
- You can use np.square to get the square of a value.
- In this context the suffix 1 denotes the control data, while 2 denotes the variation data (this applies to all functions in this assignment).
- To retrieve information from the metrics dataclass you can use the dot (.) notation. For example if you have defined the following variable `control_metrics = compute_continuous_metrics(control_sample)`, you can get s by using the expression `control_metrics.s`
- You can assign multiple values in Python in the same line. For example if you have a class with attributes a, b and c you can do something like `a1, b1, c1 = class.a, class.b, class.c`. This sometimes can make code easier to read.

```
def degrees_of_freedom(control_metrics, variation_metrics):
    """Computes the degrees of freedom for two samples.

    Args:
        control_metrics (estimation_metrics_cont): The metrics for the 
control sample.
        variation_metrics (estimation_metrics_cont): The metrics for 
the variation sample.

    Returns:
        numpy.float: The degrees of freedom.
    """

    ### START CODE HERE ###

    n1, s1 = control_metrics.n, np.square(control_metrics.s)
    n2, s2 = variation_metrics.n, np.square(variation_metrics.s)

    dof = (np.square((s1/n1) + (s2/n2))) / ((np.square(s1/n1) / (n1 - 
1)) + (np.square(s2/n2) / (n2 - 1)))


    ### END CODE HERE ###


    return dof

# Test your code
test_m1, test_m2 = compute_continuous_metrics(np.array([1,2,3])), 
compute_continuous_metrics(np.array([4,5]))
dof = degrees_of_freedom(test_m1, test_m2)
print(f"DoF for example arrays: {dof:.2f}\n")

dof = degrees_of_freedom(control_metrics, variation_metrics)
print(f"DoF for AB test samples: {dof:.2f}")
```

```
DoF for example arrays: 2.88

DoF for AB test samples: 4182.97
```

Expected Output

```
DoF for example arrays: 2.88

DoF for AB test samples: 4182.97
```

Exercise 3: t_statistic_diff_means

Now you have everything you need to perform the hypothesis testing. Complete the `t_statistic_diff_means` which given two samples should return the t-statistic, which should be computed like this:

$$t = \frac{(\acute{x}_1 - \acute{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

Hints:

- You can use np.sqrt to compute the squared root of a value.
- The value for the difference of $(\mu_1 - \mu_2)$ should be replaced by the value of this difference under the null hypothesis.

```python
def t_statistic_diff_means(control_metrics, variation_metrics):
    """Compute the t-statistic for the difference of two means.

    Args:
        control_metrics (estimation_metrics_cont): The metrics for the 
control sample.
        variation_metrics (estimation_metrics_cont): The metrics for 
the variation sample.

    Returns:
        numpy.float: The value of the t-statistic.
    """

    ### START CODE HERE ###

    n1, xbar1, s1 = control_metrics.n, control_metrics.xbar, 
control_metrics.s
    n2, xbar2, s2 = variation_metrics.n, variation_metrics.xbar, 
variation_metrics.s

    t = ((xbar1 - xbar2)) / (np.sqrt((np.square(s1) / n1) + 
(np.square(s2) / n2)))
    ### END CODE HERE ###
```

```
    return t

# Test your code

t = t_statistic_diff_means(test_m1, test_m2)
print(f"t statistic for example arrays: {t:.2f}\n")

t = t_statistic_diff_means(control_metrics, variation_metrics)
print(f"t statistic for AB test: {t:.2f}")
```

```
t statistic for example arrays: -3.27

t statistic for AB test: -1.64
```

Expected Output

```
t statistic for example arrays: -3.27

t statistic for AB test: -1.64
```

Exercise 4: reject_nh_t_statistic

With the ability to calculate the t-statistic now you need a way to determine if you should reject (or not) the null hypothesis. Complete the `reject_nh_t_statistic` function below. This function should return whether to reject (or not) the null hypothesis by using the p-value method given the value of the observed statistic, the degrees of freedom and a level of significance. **This should be a two-sided test.**

In this case the p-value represents the probability of obtaining a value of the t-statistic as extreme as or more extreme than the observed value under the null hypothesis. You can use the fact that the `CDF` of a distribution provides the probability of getting a value less than or equal to the one provided, so the probability that the statistic is greater than the observed value can be computed as `1 - CDF(observed_value)`. If you are conducting a two-sided test, then "more extreme" means that the absolute value of the statistic is greater than the absolute value of the observed statistic. In other words, "more extreme" happens when the statistic being too big or too small. Since the distribution under H_0 is symmetric around 0, the probabilities at each tail of the distribution are the same, and you can get the p-value by multiplying said `1-CDF(observed_value)` by 2, so you can compare against α rather than against $\frac{\alpha}{2}$.

Hints:

- You can use the `cdf` method from the stats.t class to compute the p-value given the degrees of freedom. Don't forget to multiply your result by 2 since this is a two-sided test.
- When passing the value of the t-statistic to the `cdf` function, you should provide the absolute value. You can achieve this by using Python's built-in `abs` function.
- If the p-value is lower to `alpha` then you should reject the null hypothesis.

```python
def reject_nh_t_statistic(t_statistic, dof, alpha=0.05):
    """Decide whether to reject (or not) the null hypothesis of the t-test.

    Args:
        t_statistic (numpy.float): The computed value of the t-statistic for the two samples.
        dof (numpy.float): The computed degrees of freedom for the two samples.
        alpha (float, optional): The desired level of significancy. Defaults to 0.05.

    Returns:
        bool: True if the null hypothesis should be rejected. False otherwise.
    """

    reject = False
    ### START CODE HERE ###
    p_value = 2 * (1 - stats.t.cdf(abs(t_statistic), df=dof))

    if p_value < alpha:
        reject = True
    ### END CODE HERE ###

    return reject

# Test your code

alpha = 0.05
reject_nh = reject_nh_t_statistic(t, dof, alpha)

print(f"The null hypothesis can be rejected at the {alpha} level of significance: {reject_nh}\n")

msg = "" if reject_nh else " not"
print(f"There is{msg} enough statistical evidence against H0.\nIt can be concluded that there is{msg} a statistically significant difference between the means of the two samples.")
```

```
The null hypothesis can be rejected at the 0.05 level of significance: False

There is not enough statistical evidence against H0.
It can be concluded that there is not a statistically significant difference between the means of the two samples.
```

Expected Output

```
The null hypothesis can be rejected at the 0.05 level of significance: False
```

```
There is not enough statistical evidence against H0.
It can be concluded that there is not a statistically significant
difference between the means of the two samples.
```

Given the initial values for each group it looked like the change in the background could be having the positive impact it was initially thought. However after performing the hypothesis testing you can conclude that there is not enough statistical evidence to reject the null hypothesis at a significance level of 0.05, so you can't confirm that the average session duration was affected by the change and the slight increase you saw at first may be due to randomness.

Section 2: Proportions - Conversion Rate (CVR)

After the experience with your own website you decided to work as a full time Data Analyst helping other companies run their AB tests. Currently you are working for a food delivery app to determine if a new feature (which provides custom suggestions to each user based on their preferences) will increase the `conversion rate` of the app. This rate measures the rate of users who "converted" or placed an order using the app. By now you know that most companies use proportion-based metrics to measure their AB tests since these are typically well understood by stakeholders and an economic value is usually predefined for them.

One thing you missed in your first AB test was to take into account the sample size required to get a significant result out of your test. Luckily now you have the experience to compute this before starting the test. **The current CVR of the app is 12% and the stakeholders would like the new feature to increase it up to a 14%**. Given this expectation you can compute the required sample size like this:

Sample size needed to compare two binomial proportions using a two-sided test with significance level α and power $1-\beta$ where one sample (n_2) is k times as large as the other sample (n_1) (independent-sample case)

To test the hypothesis $H_0: p_1 = p_2$ vs. $H_1: p_1 \neq p_2$ for the specific alternative $|p_1 - p_2| = \Delta$, with significance level α and power $1-\beta$, for the following sample size is required

$$n_1 = \frac{\left[\sqrt{\acute{p}\acute{q}\left(1+\frac{1}{k}\right)} z_{1-\alpha/2} + \sqrt{p_1 q_1 + \frac{p_2 q_2}{k}} z_{1-\beta}\right)^2}{\Delta^2}$$

$$n_2 = k n_1$$

where $p_1,p_2 = $ projected true probabilities of success in the two groups and

$$q_1, q_2 = 1 - p_1, 1 - p_2$$

$$\Delta = |p_2 - p_1|$$

$$\overline{p}=\frac{p_1+k\,p_2}{1+k}$$

This is already provided for you and can be determined by running the following cell:

```
# Compute the sample size required to compare the actual vs desired CVR
required_sample_size = utils.sample_size_diff_proportions(0.12, 0.14)
required_sample_size
```

```
4438
```

You would need around 4400 users per group to be able to detect a difference between the current CVR and the expected one with a level of significance of 0.05 and a power of 0.8.

In case you are wondering about this computation but for the continuous metrics case (section 1). Click the robot to see the formula:

Since the app has 1038 daily active users you will need to determine for how long should you run the experiment to get the desired number of users. Assuming you will split your users 50-50 between the original app and the version with the feature you would have:

```
daily_active_users = 1038

n_days = math.ceil((required_sample_size*2)/daily_active_users)

print(f"AB test should run for {n_days} days to gather enough data")
```

```
AB test should run for 9 days to gather enough data
```

This is a very important step in AB testing because you want to have a big enough sample size so you can trust the results but you don't want to run the experiment forever because this increases the chances of any external factor messing up the effect of the feature you want to capture. Also you don't know if the new feature will even be beneficial so keeping the experiment short minimizes the risk of damaging the overall conversion rate. Run the experiment by running the cell below:

```
data = utils.run_ab_test_personalized_feed(n_days)

data.head(5)
```

	user_id	user_type	converted
0	MC9Y9OFKMI	variation	0
1	QFJ7IEMBF0	control	1
2	89MIZXHCAF	variation	0
3	KJ0EGWWYG2	control	0
4	R4QX16CNJO	control	0

Similarly to the data in section 1, you have the information of the type of group and whether or not the user converted, for every user. Separate the two groups by running the next cell:

```
control_data = data[data["user_type"]=="control"]["converted"]
variation_data = data[data["user_type"]=="variation"]["converted"]

print(f"{len(control_data)} users saw the original app with an average CVR of {control_data.mean():.4f}\n")
print(f"{len(variation_data)} users saw the app with the new feature with an average CVR of {variation_data.mean():.4f}")
```

```
4632 users saw the original app with an average CVR of 0.1244

4728 users saw the app with the new feature with an average CVR of 0.1519
```

The split is not perfectly balanced but you have enough data for each group to reach a conclusion.

At first glance it looks like the new feature did in fact improve the user experience and drived more users to convert. However you already know you must perform a hypothesis test to know if there is a significant difference between the **rates (proportions)** of these two segments. You can do this by computing the z-statistic:

$$z=\frac{\hat{p}_1-\hat{p}_2}{\sqrt{\hat{p}\left(1-\hat{p}\right)\left(\frac{1}{n_1}+\frac{1}{n_2}\right)}}$$

where $\hat{p}$ is the pooled proportion: $\hat{p}=\frac{x_1+x_2}{n_1+n_2}$

The next step is to compute all the necessary metrics for every group. For this you decide to use a dataclass that holds this information:

```
@dataclass
class estimation_metrics_prop:
    n: int
    x: int
    p: float

    def __repr__(self):
        return f"sample_params(n={self.n}, x={self.x}, p={self.p:.3f})"
```

This class will hold the information for n, x and p.

Exercise 5: compute_proportion_metrics

Now that you have a container for all these metrics it is your job to code a function that given some data will compute them for that particular set of data. To do this complete the `compute_proportion_metrics` below.

Hints:

- n stands for the number of users in the data
- x stands for the number of users who converted in the data
- p stands for CVR (users who converted/total users
- `compute_proportion_metrics` expects a Pandas series as a parameter. You can sum all the values of a Pandas series by using the `sum()` method like this: `series.sum()`.

```
def compute_proportion_metrics(data):
    """Computes the relevant metrics out of a sample for proportion-
like data.

    Args:
        data (pandas.core.series.Series): The sample data. In this
case 1 if the user converted and 0 otherwise.

    Returns:
        estimation_metrics_prop: The metrics saved in a dataclass
instance.
    """

    ### START CODE HERE ###
    metrics = estimation_metrics_prop(
        n=len(data),
        x=data.sum(),
        p=data.mean(),
    )
    ### END CODE HERE ###

    return metrics

# Test your code
cm = compute_proportion_metrics(np.array([1,0,0,1]))
print(f"n={cm.n}, x={cm.x} and p={cm.p:.4f} for sample array\n")

control_metrics = compute_proportion_metrics(control_data)
print(f"n={control_metrics.n}, x={control_metrics.x} and
p={control_metrics.p:.4f} for control data\n")

variation_metrics = compute_proportion_metrics(variation_data)
print(f"n={variation_metrics.n}, x={variation_metrics.x} and
p={variation_metrics.p:.4f} for variation data")
```

```
n=4, x=2 and p=0.5000 for sample array

n=4632, x=576 and p=0.1244 for control data

n=4728, x=718 and p=0.1519 for variation data
```

Expected Output

```
n=4, x=2 and p=0.5000 for sample array

n=4632, x=576 and p=0.1244 for control data

n=4728, x=718 and p=0.1519 for variation data
```

Exercise 6: pooled_proportion

Now that you have a way of computing all necessary metrics for each sample it is time to create a way to compute the pooled proportion. For this fill the `pooled_proportion` function below. Notice that this function will receive two instances of the `estimation_metrics_prop` class.

Remember that the pooled proportion can be computed like this:

$$\hat{p} = \frac{x_1 + x_2}{n_1 + n_2}$$

```
def pooled_proportion(control_metrics, variation_metrics):
    """Compute the pooled proportion for the two samples.

    Args:
        control_metrics (estimation_metrics_prop): The metrics for the 
control sample.
        variation_metrics (estimation_metrics_prop): The metrics for 
the variation sample.

    Returns:
        numpy.float: The pooled proportion.
    """

    ### START CODE HERE ###

    x1, n1 = control_metrics.x, control_metrics.n
    x2, n2 = variation_metrics.x, variation_metrics.n

    pp = (x1 + x2) / (n1 + n2)

    ### END CODE HERE ###

    return pp

# Test your code

test_m1, test_m2 = compute_proportion_metrics(np.array([1,0,1])), 
compute_proportion_metrics(np.array([1,1,1,0]))
pp = pooled_proportion(test_m1, test_m2)
print(f"pooled proportion for example arrays: {pp:.4f}\n")
```

```
pp = pooled_proportion(control_metrics, variation_metrics)
print(f"pooled proportion for AB test samples: {pp:.4f}")
```

```
pooled proportion for example arrays: 0.7143

pooled proportion for AB test samples: 0.1382
```

Expected Output

```
pooled proportion for example arrays: 0.7143

pooled proportion for AB test samples: 0.1382
```

Exercise 7: z_statistic_diff_proportions

Now you have everything you need to calculate the z-statistic for the difference between proportions. Remember that this statistic can be computed like this:

$$z = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1-\hat{p})\left(\frac{1}{n_1}+\frac{1}{n_2}\right)}}$$

where $\hat{p}$ is the pooled proportion: $\hat{p} = \frac{x_1 + x_2}{n_1 + n_2}$

Hints:

- Remember to use the `pooled_proportion` function you coded earlier.

```
def z_statistic_diff_proportions(control_metrics, variation_metrics):
    """Compute the z-statistic for the difference of two proportions.

    Args:
        control_metrics (estimation_metrics_prop): The metrics for the
control sample.
        variation_metrics (estimation_metrics_prop): The metrics for
the variation sample.

    Returns:
        numpy.float: The z-statistic.
    """

    ### START CODE HERE ###

    pp = pooled_proportion(control_metrics, variation_metrics)

    n1, p1 = control_metrics.n, control_metrics.p
    n2, p2 = variation_metrics.n, variation_metrics.p

    z = (p1 - p2) / np.sqrt((pp * (1 - pp)) * ((1 / n1) + (1 / n2)))
```

```
    ### END CODE HERE ###

    return z

# Test your code

z = z_statistic_diff_proportions(test_m1, test_m2)
print(f"z statistic for example arrays: {z:.4f}\n")

z = z_statistic_diff_proportions(control_metrics, variation_metrics)
print(f"z statistic for AB test: {z:.4f}")
```

```
z statistic for example arrays: -0.2415

z statistic for AB test: -3.8551
```

Expected Output

```
z statistic for example arrays: -0.2415

z statistic for AB test: -3.8551
```

Exercise 8: reject_nh_z_statistic

Complete the `reject_nh_z_statistic` function below. This function should return whether to reject (or not) the null hypothesis by using the p-value method given the value of the z-statistic and a level of significance. **This should be a two-sided test.**

Hints:

- You can use the `cdf` method from the stats.norm class to compute the p-value. Don't forget to multiply your result by 2 since this is a two-sided test.
- When passing the value of the z-statistic to the `cdf` function, you should provide the absolute value. You can achieve this by using Python's built-in `abs` function.
- If the p-value is lower than `alpha` then you should reject the null hypothesis.

```
def reject_nh_z_statistic(z_statistic, alpha=0.05):
    """Decide whether to reject (or not) the null hypothesis of the z-test.

    Args:
        z_statistic (numpy.float): The computed value of the z-statistic for the two proportions.
        alpha (float, optional): The desired level of significancy. Defaults to 0.05.

    Returns:
        bool: True if the null hypothesis should be rejected. False otherwise.
```

```
    """

    reject = False
    ### START CODE HERE ###
    p_value = 2 * (1 - stats.norm.cdf(z_statistic))

    if alpha < p_value:
        reject = True
    ### END CODE HERE ###

    return reject

# Test your code

alpha = 0.05
reject_nh = reject_nh_z_statistic(z, alpha)

print(f"The null hypothesis can be rejected at the {alpha} level of significance: {reject_nh}\n")

msg = "" if reject_nh else " not"
print(f"There is{msg} enough statistical evidence against H0.\nThus it can be concluded that there is{msg} a statistically significant difference between the two proportions.")
```

```
The null hypothesis can be rejected at the 0.05 level of significance:
True

There is enough statistical evidence against H0.
Thus it can be concluded that there is a statistically significant
difference between the two proportions.
```

Expected Output

```
The null hypothesis can be rejected at the 0.05 level of significance:
True

There is enough statistical evidence against H0.
Thus it can be concluded that there is a statistically significant
difference between the two proportions
```

In this case the new feature did in fact increased the CVR. The conclusion of the AB test is that you should release the new feature to all users as there is strong statistical evidence that this will result in a better CVR.

Exercise_9: confidence_interval_proportion

Finally you would like to create confidence intervals for the CVRs of each one of the two groups. You can compute such interval for a proportion like this:

$$\hat{p} \pm z_{1-\alpha/2}\sqrt{\frac{\hat{p}\left(1-\hat{p}\right)}{n}}$$

Complete the `confidence_interval_proportion` function below. This function will receive the metrics of one of the groups and should return the lower and upper values of the confidence interval.

Hints:

- You can use the `ppf` method from the stats.norm class to compute the value of z

```
def confidence_interval_proportion(metrics, alpha=0.05):
    """Compute the confidende interval for a proportion-like sample.

    Args:
        metrics (estimation_metrics_prop): The metrics for the sample.
        alpha (float, optional): The desired level of significance.
Defaults to 0.05.

    Returns:
        (numpy.float, numpy.float): The lower and upper bounds of the
confidence interval.
    """

    ### START CODE HERE ###
    n, p = metrics.n, metrics.p

    distance = stats.norm.ppf(1 - alpha / 2) * np.sqrt((p * (1 - p)) /
n)

    lower = p - distance
    upper = p + distance
    ### END CODE HERE ###

    return lower, upper

# Test your code

c_lower, c_upper = confidence_interval_proportion(control_metrics)
print(f"Confidence interval for control group: [{c_lower:.3f},
{c_upper:.3f}]\n")

v_lower, v_upper = confidence_interval_proportion(variation_metrics)
print(f"Confidence interval for variation group: [{v_lower:.3f},
{v_upper:.3f}]")
```

```
Confidence interval for control group: [0.115, 0.134]

Confidence interval for variation group: [0.142, 0.162]
```

Expected Output

```
Confidence interval for control group: [0.115, 0.134]

Confidence interval for variation group: [0.142, 0.162]
```

As you can see the intervals for the two groups do not overlap, which alligns with the conclusion that you found earlier that there is indeed a statistically significant difference between the two proportions.

Bonus Widget: AB test calculator

If you use any web search engine you will find a lot of AB test calculators online but they usually just provide a result with no real explanation of how these computations are made. After finishing this assignment you know what is going on behind the scenes so you decide to create your own AB test calculator for future uses. This can be accomplished by using some python widgets and the functions you just coded.

Run the next cell to render the calculator with your functions (`z_statistic_diff_proportions` and `reject_nh_z_statistic`) as backend:

```
utils.AB_test_dashboard(z_statistic_diff_proportions,
reject_nh_z_statistic)
```

{"model_id":"52c8fe2135de483ca6f443045d22bed1","version_major":2,"version_minor":0}

Congratulations on finishing this assignment!

Now you have created all the required steps to perform an AB test for continuous and proportion-based metrics.

This is the last assignment of the course and the specialization so give yourself a pat on the back for such a great accomplishment! Nice job!!!!