# 

# 

# 

# **A/B Testing in E-Commerce**

## 

## 

## 

## 

## 

# 

# 

# 

# **Table of Content:**

| **Section No** | **Title** |
| --- | --- |
| 1 | Project Objective |
| 2 | Understanding A/B testing |
| 3 | Dataset Description |
| 4 | Workflow |
| 5 | Code Explanations |
| 6 | Project Setup |
| 7 | Future Enhancements |
| 8 | Conclusion |

## 

## 

## 

## 

## 

## **1.Objective**

The primary goals of this study are:

* To determine whether changes to the website lead to a **significant improvement** in conversions.
* To use **statistical tests** to ensure confidence in decision-making.
* To implement **Python-based solutions** for conducting and interpreting statistical analyses.

## **2. Understanding A/B testing:**

### **A/B Testing in E-Commerce**

A/B testing is a method used by online retailers to compare different versions of a webpage to determine which performs better in terms of conversions, sales, and user engagement. It involves splitting website traffic between variations and analyzing their effectiveness based on specific objectives such as click-through rates, email sign-ups, or purchases. The goal is to optimize user experience, reduce bounce rates, increase conversions, and minimize cart abandonment.

### **How A/B Testing Works ?**

* **Client-side testing:** The same webpage is sent to all visitors, with JavaScript modifying elements in their browsers before they see the final version.
* **Server-side testing:** Different page variations are generated and modified on the server before being sent to visitors.

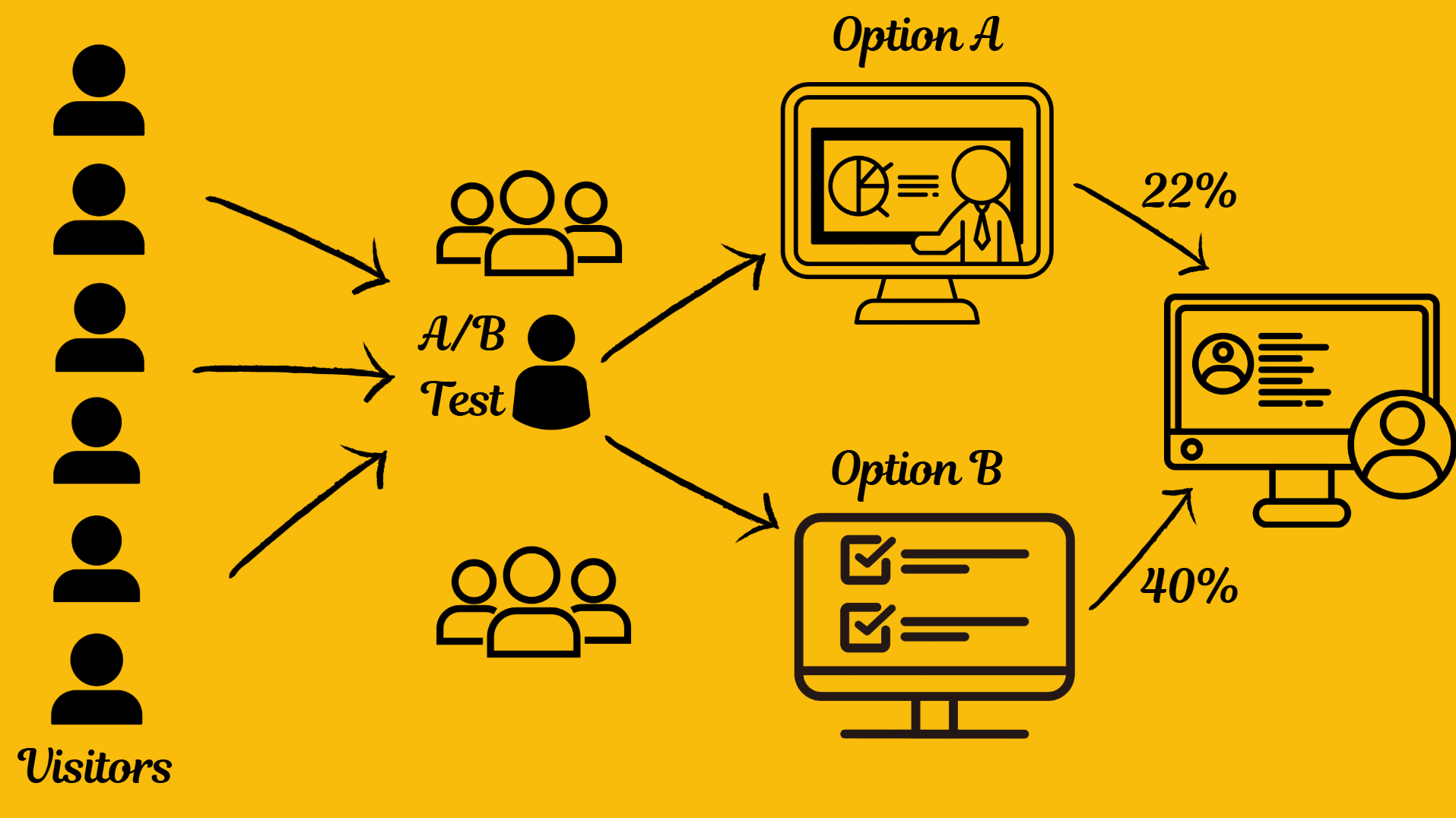
### **Example: A Website Contact Button**

A company called iNeurominds Technologies wants to get more sales leads from their website. They currently have a "**Contact**" button at the **top right** of the page **(Version A)**. Their business manager suggests moving it to the **center** of the page (**Version B**) to see if more people click it.

To test this:

1. They create two versions of the website—one with the button at the top right and another with it in the middle.
2. They show each version to 50% of their website visitors for 10 days.
3. After 10 days, they check how many people clicked the button on each version.

If Version B gets more clicks and sales leads, they will keep the button in the middle.



#### **Things to Consider in A/B Testing**

* **Avoid Bias:** The test should be random, so users from different locations and backgrounds see both versions.
* **Enough Data:** Running the test for too short a time or with too few users can lead to incorrect conclusions.
* **Statistical Significance:** The difference in results should be due to the change and not just random chance.

#### **Where is A/B Testing Used?**

* **Tech Companies:** Amazon, Facebook, and LinkedIn use A/B testing to improve website layouts and product recommendations.
* **Marketing:** Businesses run different versions of ads to see which gets more clicks or sales.
* **Machine Learning:** When launching a new AI model, companies test it on a small percentage of users before rolling it out to everyone.

#### **Can You Test More Than Two Versions?**

Yes, you can do A/B/C or even A/B/C/D testing with more variations. But the more versions you test, the more complex the experiment becomes. That's why A/B testing (comparing just two versions) is the most common.

**3.Dataset Description**

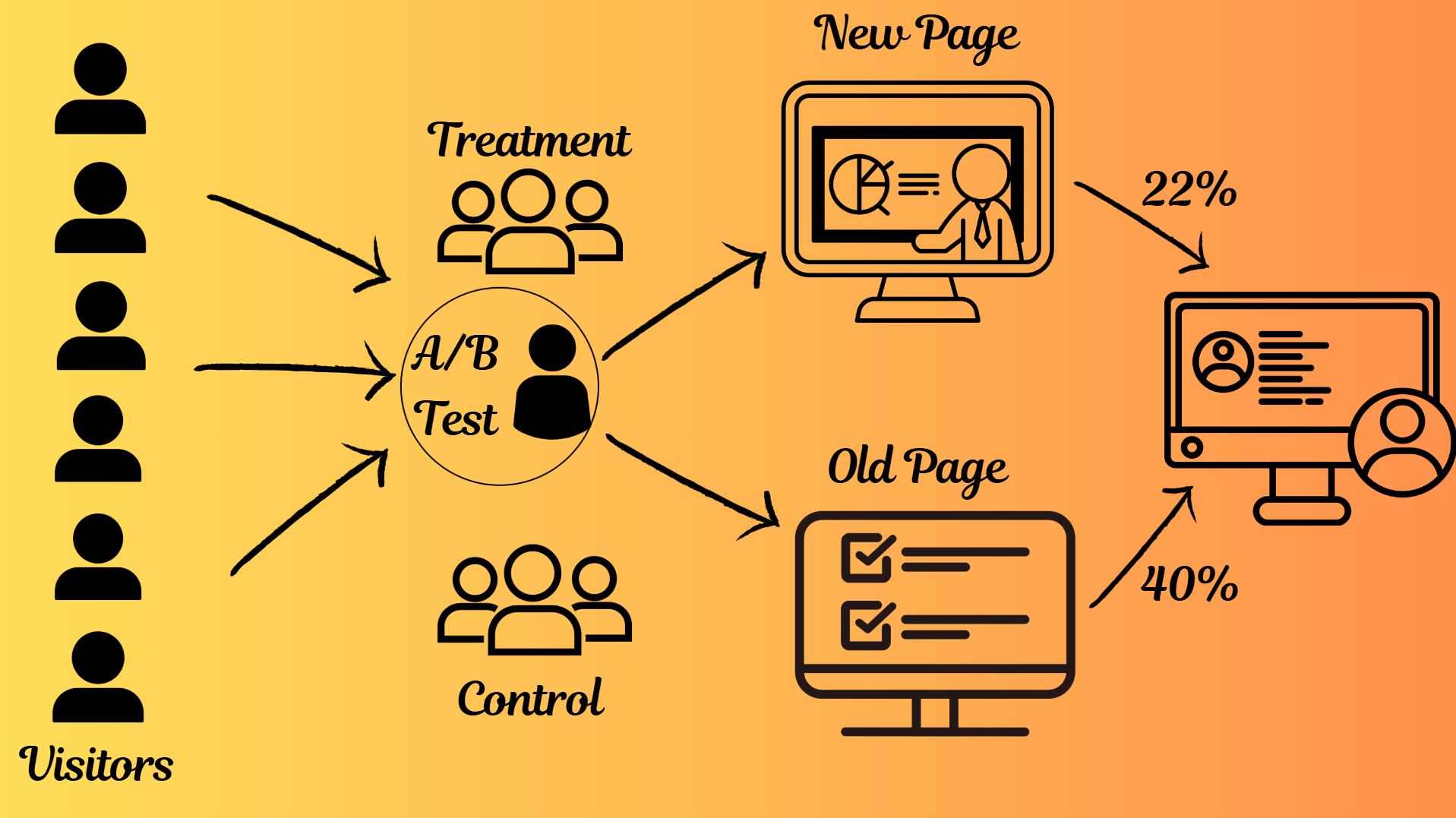
To analyze the impact of website changes, the following data is required:

### **1. Dataset Structure**

* **Shape**: The dataset contains **294,478 rows** and **5 columns**. Each row represents an individual event or interaction during the test.
* **Unique Users**: There are **290,584 unique user IDs**, indicating that some users have multiple interactions (repeated entries).

### **2. Variable Descriptions**

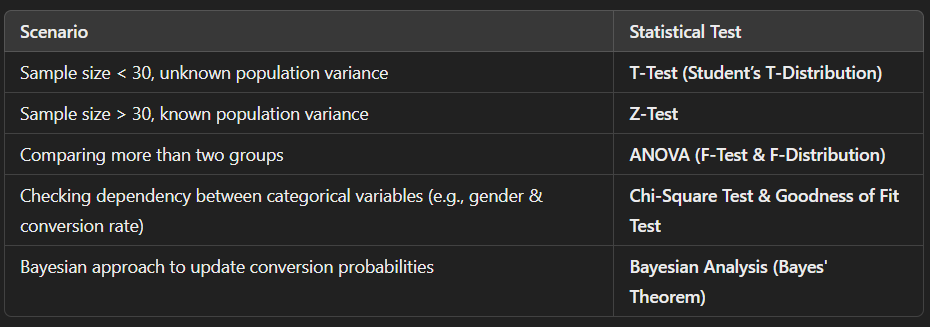
* **user\_id**: A 6-digit unique identifier for each user.
* **timestamp**: A string formatted as a datetime (e.g., 2017-01-21 22:11:48.556739) that records when the interaction occurred, useful for time-series analysis.
* **group**: Indicates the user's assignment in the A/B test with possible values control (existing design) or treatment (new design).
* **landing\_page**: Specifies which version of the landing page was viewed (old\_page or new\_page), directly linking to the A/B test.
* **converted**: A binary outcome where 0 means no conversion and 1 indicates a conversion, serving as the primary performance metric.



## **4. Workflow:**

* **Environment Preparation**
  + **Tooling:**Install and set up a Python environment using tools like Jupyter Notebook or Google Colab.
  + **Library Installation:**Ensure you have all necessary libraries installed, such as:
    - **pandas** and **NumPy** for data manipulation
    - **Matplotlib** and **Seaborn** for visualization
    - **SciPy** and **statsmodels** for statistical testing
    - **scikit-learn** (optional) for additional machine learning tasks
* **Data Acquisition & Loading**
  + **Dataset Import:**Load the A/B test dataset (e.g., ab\_data.csv) into a pandas DataFrame.
  + **Initial Exploration:**Use functions like .head(), .shape, and .isnull().sum() to get an initial understanding of the dataset, including the number of rows, columns, and missing values.
* **Data Cleaning & Preparation**
  + **Filtering:**Remove any rows where the landing page does not match the group assignment to create a cleaned DataFrame (df2).
  + **Duplicate Handling:**Identify and remove duplicate user records, ensuring each user is only represented once.
  + **Data Transformation:**Create additional variables as needed (e.g., simulate session\_duration for further analysis).
* **Exploratory Data Analysis (EDA)**
  + **Visual Analysis:**Generate visualizations such as bar plots, pie charts, histograms, KDE plots, and boxplots to explore:
    - User distribution across control and treatment groups
    - Conversion rates and their distributions
    - Mismatch counts between assigned groups and actual landing pages
  + **Statistical Summary:**Calculate key metrics like overall conversion rate, conversion rates for each group, and the observed difference between groups.
* **Statistical Testing & Simulation**

Selecting the appropriate statistical test depends on sample size, data distribution, and type of variables.



* + **Hypothesis Formulation:**Define your null (no significant difference) and alternative hypotheses.
  + **Simulation Under the Null:**Use binomial simulations to create a sampling distribution of conversion rate differences.
  + **Z-Test for Proportions:**Apply a proportions Z-test to compare conversion rates between groups.
  + **Additional Tests:**Conduct supplementary analyses (T-test for small samples, Chi-Square test for independence, ANOVA for continuous variables) as necessary.
  + **Bayesian Analysis:**Use Beta distributions to update conversion probability estimates and compute the probability that the treatment outperforms the control.
* **Results Interpretation & Reporting**
  + **Result Synthesis:**Summarize the statistical test outcomes, focusing on p-values, confidence intervals, and observed differences.
  + **Business Recommendations:**Based on the findings, provide actionable insights such as whether to deploy the new page or iterate on the design further.
  + **Visualization & Documentation:**Compile charts, tables, and commentary in a well-documented Jupyter Notebook that can be shared with stakeholders.

## **5. Code Explanation:**

### **Variables Description**

### **user\_id**

* Type: 6-digit number
* Description: Serves as a unique identifier for each user. Although most users appear only once, the fact that the unique count is lower than the total number of rows implies that some users have multiple records.

### **timestamp**

* Type: String (formatted as a datetime)
* Description: Captures the exact date and time when the event or interaction was recorded. The format (e.g., 2017-01-21 22:11:48.556739) provides precise timing information, which can be crucial for time-series analysis or understanding user behavior trends during the A/B test period.

### **group**

* Type: String
* Possible Values: **control, treatment**
* Description: Indicates the group to which the user was assigned during the A/B test. The **control group typically represents the existing version (or a baseline)**, while the **treatment group represents the new version** or intervention being tested.

### **landing\_page**

* Type: String
* Possible Values: **old\_page, new\_page**
* Description: Specifies which landing page the user saw. This variable is directly linked to the A/B testing setup where one group is exposed to the old design and the other to the new design.

### **converted**

* Type: Numeric (0 or 1)
* Description: Indicates whether the user completed the desired action (conversion).
* 0 means No Conversion
* 1 means Conversion
* This binary outcome is typically the primary metric for assessing the effectiveness of the new design compared to the old one.

## **1. Import necessary libraries**

| # Import necessary libraries import pandas as pd # Library for data manipulation and analysis import numpy as np # Library for numerical operations import random # Library for generating random numbers import matplotlib.pyplot as plt # Library for creating static, animated, and interactive visualizations import statsmodels.api as sm # Library for statistical modeling import statsmodels.formula.api as smf # Library for statistical modeling using formulas from scipy.stats import norm, ttest\_ind, chi2\_contingency, beta # Library for statistical functions  # Enable inline plotting in Jupyter Notebooks %matplotlib inline  # Set seed for reproducibility random.seed(42) # Set seed for random library np.random.seed(42) # Set seed for numpy library |
| --- |

# Read the dataset from a CSV file named 'ab\_data.csv'

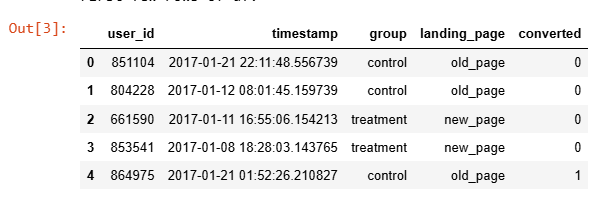
| # Read the dataset from a CSV file named 'ab\_data.csv' df = pd.read\_csv('ab\_data.csv')  # a. Inspect the first few rows of the dataset print("First few rows of df:") df.head() |
| --- |

Explanation:

1. pd.read\_csv('ab\_data.csv'): This line reads a CSV file named ab\_data.csv into a pandas DataFrame object called df.

2. print("First few rows of df:"): This line prints a message indicating that the next output will be the first few rows of the DataFrame.

3. df.head(): This line displays the first few rows of the DataFrame using the head() method. By default, head() displays the first 5 rows.



| # Check the number of observations (rows) in the DataFrame print("\nShape of df:", df.shape) # Expected output: (294478, 5) |
| --- |

Explanation:

1. df.shape: This attribute returns a tuple representing the dimensionality of the DataFrame. The first element of the tuple represents the number of rows (observations), and the second element represents the number of columns (variables).

2. print("\nShape of df:", df.shape): This line prints the shape of the DataFrame, which indicates the number of rows and columns.

Example Output:

Shape of df: (294478, 5)

This output indicates that the DataFrame has:

- 294,478 rows (observations)

- 5 columns (variables)

| #Calculate the number of unique users in the dataset unique\_users = df['user\_id'].nunique()  #Print the count of unique user IDs print("\nUnique user\_id count:", unique\_users)  #Expected output: 290584 |
| --- |

Explanation:

1. df['user\_id']: This selects the user\_id column from the DataFrame.

2. .nunique(): This method returns the number of unique values in the selected column.

3. unique\_users = df['user\_id'].nunique(): This line calculates the number of unique users and assigns it to the unique\_users variable.

4. print("\nUnique user\_id count:", unique\_users): This line prints the count of unique user IDs.

Output:

Unique user\_id count: 290584

This output indicates that there are 290,584 unique users in the dataset.

| #Calculate the number of unique users who converted prop\_converted = df.query('converted == 1')['user\_id'].nunique() / unique\_users  # Print the proportion of users who converted print("\nProportion of users converted:", prop\_converted) |
| --- |

Explanation:

1. df.query('converted == 1'): This selects rows from the DataFrame where the converted column is equal to 1, indicating a successful conversion.

2. ['user\_id'].nunique(): This calculates the number of unique users who converted.

3. / unique\_users: This divides the number of unique users who converted by the total number of unique users, resulting in the proportion of users who converted.

Example Output:

Proportion of users converted: 0.12104245244060237

This output indicates that approximately 12.10% of users converted.

| #Count rows where landing\_page and group do not match mismatch\_count = ( df.query('landing\_page == "new\_page" and group != "treatment"').shape[0] + df.query('landing\_page != "new\_page" and group == "treatment"').shape[0] )  #Print the mismatch count print("\nMismatch count between landing\_page and group:", mismatch\_count) |
| --- |

Explanation:

1. df.query('landing\_page == "new\_page" and group != "treatment"'): This selects rows where the landing\_page is "new\_page" but the group is not "treatment".

2. .shape[0]: This returns the number of rows in the selected DataFrame.

3. The second df.query() does the opposite: selects rows where the landing\_page is not "new\_page" but the group is "treatment".

4. The two counts are added together to get the total mismatch count.

Example Output:

Mismatch count between landing\_page and group: 3893

This output indicates that there are 3,893 rows where the landing\_page and group do not match.

| #Check for missing values in the DataFrame print("\nDataFrame Info:") df.isnull().sum() |
| --- |

Explanation:

1. df.isnull(): This method returns a boolean mask indicating which values in the DataFrame are missing (NaN).

2. .sum(): This method sums up the boolean values, treating True (missing value) as 1 and False (non-missing value) as 0.

Example Output:

| DataFrame Info: user\_id 0 timestamp 0 group 0 landing\_page 0 converted 0 dtype: int64 |
| --- |

This output indicates that there are no missing values in the DataFrame. The count of missing values for each column is 0.

**Proportion of Users by Group**

| import matplotlib.pyplot as plt  # Count the number of users in each group group\_counts = df['group'].value\_counts()  # Calculate the proportion of users in each group group\_proportions = group\_counts / len(df)  # Create a bar plot of the group proportions plt.bar(group\_proportions.index, group\_proportions.values)  # Add labels and title to the plot plt.xlabel('Group') plt.ylabel('Proportion of Users') plt.title('Proportion of Users by Group')  # Display the plot plt.show() |
| --- |

Explanation:

1. group\_counts = df['group'].value\_counts(): This line counts the number of users in each group using the value\_counts() method.

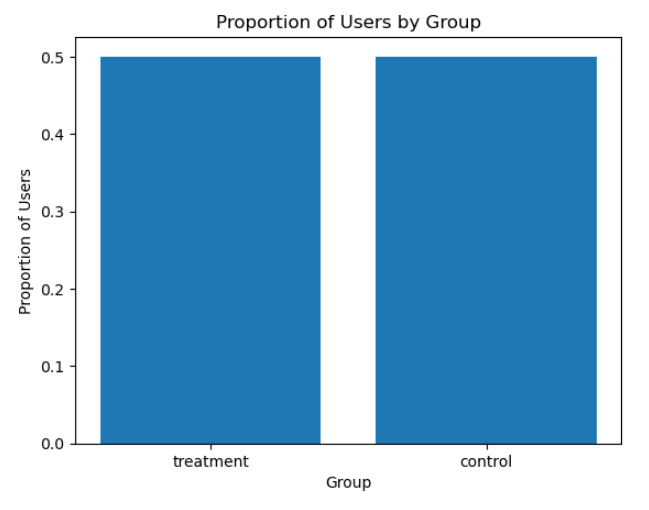
2. group\_proportions = group\_counts / len(df): This line calculates the proportion of users in each group by dividing the count by the total number of users.

3. plt.bar(group\_proportions.index, group\_proportions.values): This line creates a bar plot of the group proportions using matplotlib.

4. plt.xlabel(), plt.ylabel(), and plt.title(): These lines add labels and a title to the plot.

5. plt.show(): This line displays the plot.

This code generates a bar plot showing the proportion of users in each group.



**Proportion of Converted Users**

| import matplotlib.pyplot as plt  # Calculate the proportion of converted users converted\_proportion = prop\_converted  # Calculate the proportion of non-converted users not\_converted\_proportion = 1 - converted\_proportion  # Create a pie chart of the converted and non-converted proportions plt.pie([converted\_proportion, not\_converted\_proportion],   labels=['Converted', 'Not Converted'],   autopct='%1.1f%%')  # Add a title to the chart plt.title('Proportion of Converted Users')  # Display the chart plt.show() |
| --- |

Explanation:

1. converted\_proportion = prop\_converted: This line assigns the previously calculated proportion of converted users to a new variable.

2. not\_converted\_proportion = 1 - converted\_proportion: This line calculates the proportion of non-converted users by subtracting the converted proportion from 1.

3. plt.pie(): This function creates a pie chart with two slices representing the converted and non-converted proportions.

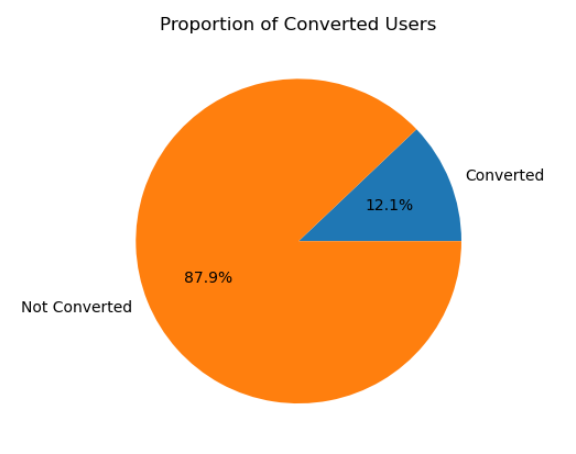
4. labels=['Converted', 'Not Converted']: This parameter adds labels to each slice of the pie chart.

5. autopct='%1.1f%%': This parameter formats the value displayed on each slice as a percentage with one decimal place.

6. plt.title(): This function adds a title to the chart.

7. plt.show(): This function displays the chart.

This code generates a pie chart showing the proportion of converted and non-converted users.



**Distribution of Converted Users by Group**

| import matplotlib.pyplot as plt import seaborn as sns  # Filter the DataFrame to include only converted users converted\_df = df[df['converted'] == 1]  # Create a kernel density estimate (KDE) plot of the user IDs # for converted users, colored by group sns.kdeplot(data=converted\_df, x="user\_id", hue="group", shade=True)  # Add labels and title to the plot plt.xlabel('User ID') plt.ylabel('Density') plt.title('Distribution of Converted Users by Group')  # Display the plot plt.show() |
| --- |

Explanation:

1. converted\_df = df[df['converted'] == 1]: This line filters the original DataFrame to include only rows where the converted column is equal to 1.

2. sns.kdeplot(): This function creates a kernel density estimate (KDE) plot, which is a smoothed representation of the distribution of user IDs.

3. data=converted\_df: This parameter specifies the DataFrame to use for the plot.

4. x="user\_id": This parameter specifies the column to use for the x-axis.

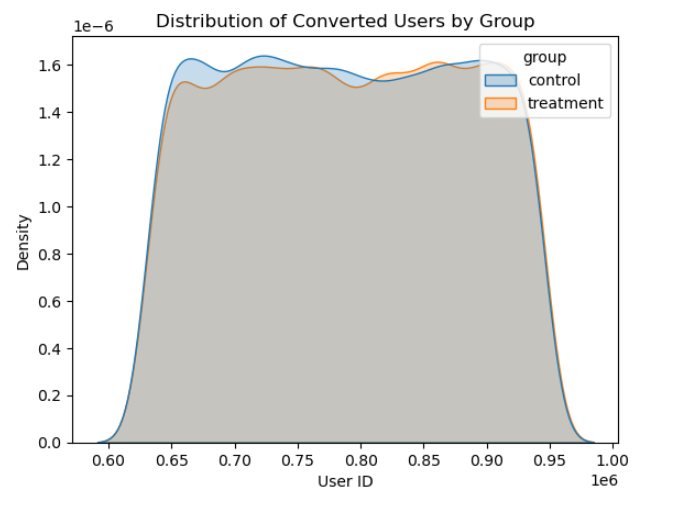
5. hue="group": This parameter specifies the column to use for coloring the plot.

6. shade=True: This parameter adds shading to the plot.

7. plt.xlabel(), plt.ylabel(), and plt.title(): These functions add labels and a title to the plot.

8. plt.show(): This function displays the plot.

This code generates a KDE plot showing the distribution of converted users by group.



**Distribution of Converted Users by Group**

| import matplotlib.pyplot as plt  # Filter the DataFrame to include only converted users converted\_df = df[df['converted'] == 1]  # Count the number of converted users in each group group\_counts = converted\_df['group'].value\_counts()  # Create a boxplot of user IDs for converted users in each group plt.boxplot([  converted\_df[converted\_df['group'] == 'control']['user\_id'],   converted\_df[converted\_df['group'] == 'treatment']['user\_id'] ],  labels=['Control', 'Treatment'])  # Add labels and title to the plot plt.xlabel('Group') plt.ylabel('User ID') plt.title('Distribution of Converted Users by Group')  # Display the plot plt.show() |
| --- |

Explanation:

1. converted\_df = df[df['converted'] == 1]: This line filters the original DataFrame to include only rows where the converted column is equal to 1.

2. group\_counts = converted\_df['group'].value\_counts(): This line counts the number of converted users in each group.

3. plt.boxplot(): This function creates a boxplot, which is a graphical representation of the distribution of user IDs.

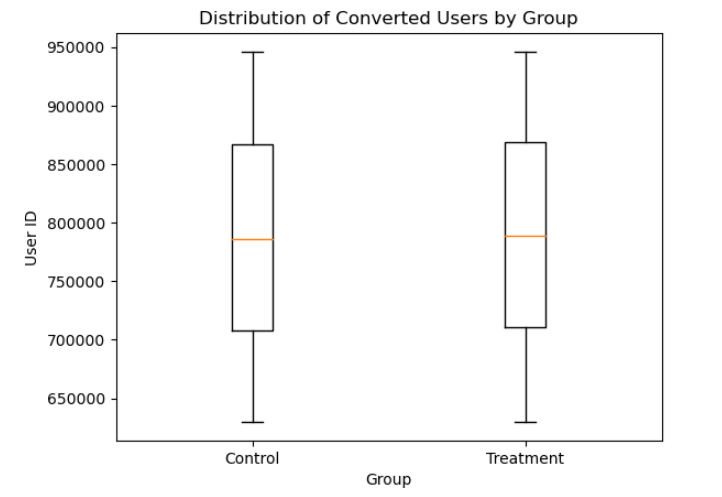
4. The list passed to plt.boxplot() contains two arrays of user IDs: one for the control group and one for the treatment group.

5. labels=['Control', 'Treatment']: This parameter adds labels to the x-axis.

6. plt.xlabel(), plt.ylabel(), and plt.title(): These functions add labels and a title to the plot.

7. plt.show(): This function displays the plot.

This code generates a boxplot showing the distribution of converted users by group.



**Mismatch Count between Landing Page and Group**

| import matplotlib.pyplot as plt  # Calculate the counts of mismatched and matched rows mismatch\_counts = [mismatch\_count, len(df) - mismatch\_count]  # Create a bar chart of the mismatch counts plt.bar(['Mismatch', 'Match'], mismatch\_counts)  # Add labels and title to the chart plt.xlabel('Mismatch Status') plt.ylabel('Count') plt.title('Mismatch Count between Landing Page and Group')  # Display the chart plt.show() |
| --- |

Explanation:

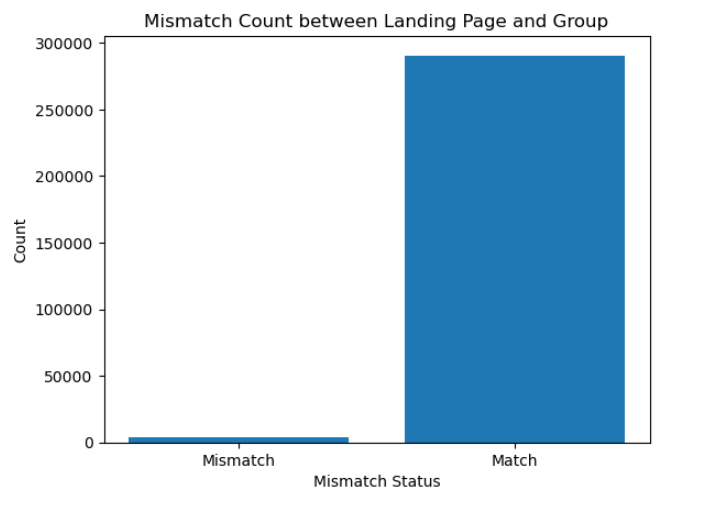
1. mismatch\_counts = [mismatch\_count, len(df) - mismatch\_count]: This line calculates the counts of mismatched and matched rows. The first element is the previously calculated mismatch\_count, and the second element is the total number of rows minus the mismatch count.

2. plt.bar(['Mismatch', 'Match'], mismatch\_counts): This function creates a bar chart with two bars, one for mismatched rows and one for matched rows.

3. plt.xlabel(), plt.ylabel(), and plt.title(): These functions add labels and a title to the chart.

4. plt.show(): This function displays the chart.

This code generates a bar chart showing the count of mismatched and matched rows between the landing page and group.



**Clean the Data to Create df2**

Keep only rows where landing\_page matches the group assignment.

Create a new DataFrame df2 with only the rows where the landing page

| #matches the group assignment df2 = df.query('(landing\_page == "new\_page" & group == "treatment") | (landing\_page == "old\_page" & group == "control")')  #Display the resulting DataFrame df2 |
| --- |

Explanation:

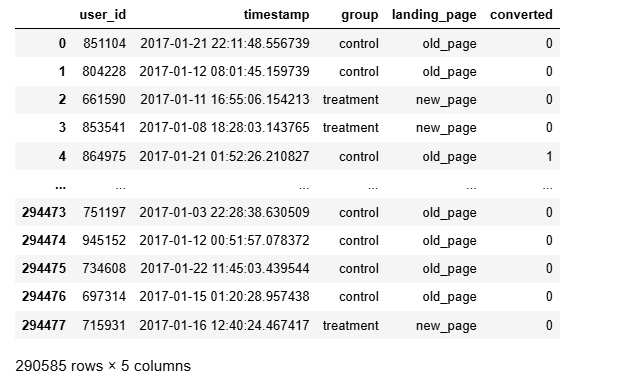
1. df.query(): This method filters the DataFrame based on a query string.

2. (landing\_page == "new\_page" & group == "treatment"): This condition selects rows where the landing page is "new\_page" and the group is "treatment".

3. |: This operator represents a logical OR operation.

4. (landing\_page == "old\_page" & group == "control"): This condition selects rows where the landing page is "old\_page" and the group is "control".

By applying this filter, df2 will only contain rows where the landing page matches the group assignment.



**Double-check that there are no mismatches in df2**

| # Create a new DataFrame mismatch\_df2 that contains only the rows  # where the group and landing page do not match mismatch\_df2 = df2[((df2['group'] == 'treatment') == (df2['landing\_page'] == 'new\_page')) == False].shape[0]  # Print the number of mismatches in df2 print("\nMismatch count in df2:", mismatch\_df2)  # Print the number of unique user\_ids in df2 before removing duplicates print("\nUnique user\_ids in df2 before duplicate removal:", df2['user\_id'].nunique()) |
| --- |

Explanation:

1. df2[((df2['group'] == 'treatment') == (df2['landing\_page'] == 'new\_page')) == False]: This line creates a new DataFrame that contains only the rows where the group and landing page do not match.

2. .shape[0]: This method returns the number of rows in the new DataFrame.

3. print("\nMismatch count in df2:", mismatch\_df2): This line prints the number of mismatches in df2.

4. df2['user\_id'].nunique(): This method returns the number of unique user\_ids in df2.

5. print("\nUnique user\_ids in df2 before duplicate removal:", df2['user\_id'].nunique()): This line prints the number of unique user\_ids in df2 before removing duplicates.

Output:

Mismatch count in df2: 0

Unique user\_ids in df2 before duplicate removal: 290584

This output indicates that there are no mismatches in df2, and there are 290,584 unique user\_ids in df2 before removing duplicates.

| Identify duplicate user\_ids (if any)  # Create a new column 'is\_duplicate' in df2 to mark duplicate user\_ids df2.loc[:, "is\_duplicate"] = df2['user\_id'].duplicated()  # Query df2 to display rows with duplicate user\_ids df2.query('is\_duplicate == True') |
| --- |

Explanation:

1. df2['user\_id'].duplicated(): This method returns a boolean Series indicating whether each user\_id is duplicated.

2. df2.loc[:, "is\_duplicate"] = ...: This line creates a new column 'is\_duplicate' in df2 and assigns the boolean values from the previous step.

3. df2.query('is\_duplicate == True'): This line queries df2 to display only the rows where 'is\_duplicate' is True.



This output indicates that there is at least one duplicate user\_id (773192) in the data.

**Display all rows for the duplicate user\_id (if exists)**

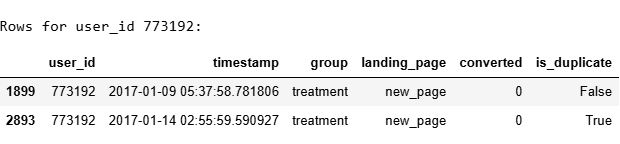
| # Filter df2 to get all rows for the duplicate user\_id duplicate\_user = df2[df2['user\_id'] == 773192]  # Print the rows for the duplicate user\_id print("\nRows for user\_id 773192:") print(duplicate\_user) |
| --- |

Explanation:

1. df2[df2['user\_id'] == 773192]: This line filters df2 to get all rows where the user\_id is equal to 773192.

2. print(duplicate\_user): This line prints the resulting DataFrame, which contains all rows for the duplicate user\_id.

This code will display all rows for the user\_id 773192, which was previously identified as a duplicate.

  
**Remove duplicate rows (keep first occurrence)**

| # Drop duplicate rows based on the 'user\_id' column, keeping only the first occurrence df2 = df2.drop\_duplicates(subset='user\_id', keep="first")  # Print the number of duplicates after removal print("\nDuplicates after removal:", df2['user\_id'].duplicated().sum()) |
| --- |

Explanation:

1. df2.drop\_duplicates(): This method removes duplicate rows from the DataFrame.

2. subset='user\_id': This parameter specifies the column(s) to consider when looking for duplicates.

3. keep="first": This parameter specifies that only the first occurrence of each duplicate should be kept.

4. df2['user\_id'].duplicated().sum(): This line checks for any remaining duplicates and prints the count.

Output:

Duplicates after removal: 0

This output indicates that all duplicate rows have been successfully removed, and the resulting DataFrame df2 contains only unique user\_ids.

# **4. Compute Conversion Metrics**

**4.1 Overall conversion rate (con\_obs)**

| #Calculate the overall conversion rate by taking the mean of the #'converted' column con\_obs = df2['converted'].mean() print("\nOverall conversion rate (con\_obs):", con\_obs) |
| --- |

**4.2 Conversion rate for control group (old\_page)**

| #Calculate the conversion rate for the control group by dividing the number #of converted users by the total number of users in the control group con\_c = len(df2.query('converted == 1 & group == "control"')) / len(df2.query('group == "control"')) print("Conversion rate for control group:", con\_c) |
| --- |

**4.3 Conversion rate for treatment group (new\_page)**

| #Calculate the conversion rate for the treatment group by dividing the  # number of converted users by the total number of users in the treatment # group con\_t = len(df2.query('converted == 1 & group == "treatment"')) / len(df2.query('group == "treatment"')) print("Conversion rate for treatment group:", con\_t)  #Difference in conversion rates (treatment - control) #Calculate the difference in conversion rates between the treatment and #control groups obs\_diff = con\_t - con\_c print("Observed difference (treatment - control):", obs\_diff) |
| --- |

Output:

| Overall conversion rate (con\_obs): 0.11959708724499628 Conversion rate for control group: 0.1203863045004612 Conversion rate for treatment group: 0.11880806551510564 Observed difference (treatment - control): -0.0015782389853555567 |
| --- |

These results indicate the overall conversion rate, as well as the conversion rates for the control and treatment groups. The observed difference between the treatment and control groups is also calculated.

**4.4. Probability that an individual received the new page**

Calculate the probability of receiving the new page by dividing the number of users who received the new page by the total number of users.

| prob\_new\_page = len(df2.query('landing\_page == "new\_page"')) / len(df2) print("Probability of receiving the new page:", prob\_new\_page) |
| --- |

Output:

| Probability of receiving the new page: 0.5000619442226688 |
| --- |

This result indicates that approximately 50% of the users received the new page, suggesting that the treatment was randomly assigned with roughly equal probability.

# **5. Simulation Under the Null Hypothesis (Z-Test for Proportions)**

**Assume**: pnew = pold = overall conversion rate (con\_obs)

============================================================================

Define the conversion probabilities under the null hypothesis (H0)

| pnew = con\_obs # Conversion probability for new page under H0 pold = con\_obs # Conversion probability for control under H0  Calculate the sample sizes for the treatment and control groups nnew = len(df2[df2['group'] == "treatment"]) # Treatment sample size nold = len(df2[df2['group'] == "control"]) # Control sample size  Print the sample sizes print("\nnnew (treatment sample size):", nnew) print("nold (control sample size):", nold) |
| --- |

Output:

| nnew (treatment sample size): 145310 nold (control sample size): 145274 |
| --- |

These results indicate that the treatment and control groups have roughly equal sample sizes, which is expected in a randomized controlled trial.

## **One iteration of simulation for demonstration:**

Simulate Conversion Rates for New and Old Pages

| new\_page\_converted = np.random.binomial(1, pnew, nnew) # Simulate conversion rates for new page old\_page\_converted = np.random.binomial(1, pold, nold) # Simulate conversion rates for old page |
| --- |

Calculate Simulated Difference in Conversion Rates

| print("\nSimulated difference in conversion (one iteration):", new\_page\_converted.mean() - old\_page\_converted.mean()) |
| --- |

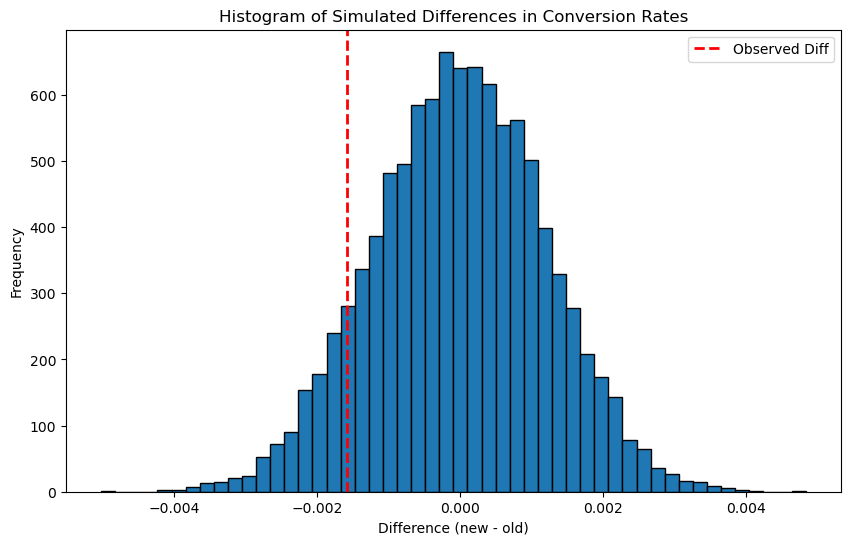
Simulate 10,000 Iterations to Generate Sampling Distribution

| p\_diffs = [] # Initialize empty list to store simulated differences for \_ in range(10000): # Simulate 10,000 iterations  new\_page\_conv = np.random.binomial(1, pnew, nnew).mean() # Simulate conversion rate for new page  old\_page\_conv = np.random.binomial(1, pold, nold).mean() # Simulate conversion rate for old page  p\_diffs.append(new\_page\_conv - old\_page\_conv) # Calculate and append simulated difference   Convert List to NumPy Array  p\_diffs = np.array(p\_diffs) # Convert list to NumPy array |
| --- |

Plot Histogram of Simulated Differences

| plt.figure(figsize=(10, 6)) # Create new figure with specified size plt.hist(p\_diffs, bins=50, edgecolor='k') # Plot histogram of simulated differences plt.axvline(obs\_diff, color='red', linestyle='dashed', linewidth=2, label='Observed Diff') # Add vertical line for observed difference plt.title('Histogram of Simulated Differences in Conversion Rates') # Add title plt.xlabel('Difference (new - old)') # Add x-axis label plt.ylabel('Frequency') # Add y-axis label plt.legend() # Add legend plt.show() # Display plot |
| --- |

Simulated difference in conversion (one iteration): -0.002397302297957274



From the histogram of simulated differences in conversion rates (new − old), the distribution is centered near zero, but the observed difference (red dashed line) is slightly negative. This indicates that, based on the simulations, **the new page’s conversion rate is not only failing to exceed the old page’s rate but may actually be performing slightly worse.** Therefore, there is no evidence that the new page outperforms the old one in terms of conversion.

## **Calculate Simulated p-value**

| # Calculate the proportion of simulated differences that are greater than the observed difference p\_value\_simulation = (p\_diffs > obs\_diff).mean() print("Simulated p-value:", p\_value\_simulation) |
| --- |

Output:

| Simulated p-value: 0.901 |
| --- |

This result indicates that about 90.1% of the simulated differences are greater than the observed difference, suggesting that the observed difference is not statistically significant.

**6. Built-in Z-Test Using statsmodels (Proportions Z-Test)**

Calculate the number of conversions for the control group (old page)

| convert\_old = len(df2.query('converted == 1 & group == "control"')) |
| --- |

Calculate the number of conversions for the treatment group (new page)

| convert\_new = len(df2.query('converted == 1 & group == "treatment"')) |
| --- |

Calculate the total number of users in the control group

| n\_old = len(df2.query('group == "control"')) |
| --- |

Calculate the total number of users in the treatment group

| n\_new = len(df2.query('group == "treatment"')) |
| --- |

| Print the results print("\nConversions for control (old\_page):", convert\_old) print("Conversions for treatment (new\_page):", convert\_new) print("Total in control group (n\_old):", n\_old) print("Total in treatment group (n\_new):", n\_new) |
| --- |

Output:

| Conversions for control (old\_page): 17489 Conversions for treatment (new\_page): 17264 Total in control group (n\_old): 145274 Total in treatment group (n\_new): 145310 |
| --- |

Complete Explanation:

- The control group (old page) had 17,489 conversions out of 145,274 total users, resulting in a conversion rate of approximately 12.0%.

- The treatment group (new page) had 17,264 conversions out of 145,310 total users, resulting in a conversion rate of approximately 11.9%.

The results suggest that the new page (treatment group) has a slightly lower conversion rate compared to the old page (control group). However, the difference in conversion rates is relatively small (0.1%), indicating that the new page may not be significantly more effective than the old page in converting users.

**Perform a proportions Z-test using statsmodels**

| z\_score, p\_value\_ztest = sm.stats.proportions\_ztest([convert\_old, convert\_new], [n\_old, n\_new], alternative='smaller', prop\_var=False)  Print the Z-test results print("\nZ-test Results:") print("Z-score:", z\_score) print("P-value:", p\_value\_ztest)  Calculate the Z threshold for a one-tailed test at 5% significance z\_threshold = norm.ppf(1 - 0.05) print("Z threshold (one-tailed 5%):", z\_threshold) |
| --- |

Complete Explanation:

This code performs a proportions Z-test using the statsmodels library to compare the conversion rates between the control and treatment groups.

The proportions\_ztest function takes the following inputs:

- [convert\_old, convert\_new]: The number of conversions in the control and treatment groups.

- [n\_old, n\_new]: The total number of users in the control and treatment groups.

- alternative='smaller': Specifies that we want to test whether the conversion rate of the treatment group is smaller than that of the control group.

- prop\_var=False: Specifies that we do not want to use the pooled proportion variance.

The function returns the Z-score and p-value of the test.

**The Output:**

| Z-test Results: Z-score: 1.3109241984234394 P-value: 0.9050583127590245 Z threshold (one-tailed 5%): 1.6448536269514722 |
| --- |

Based on the results:

The Z-score of 1.3109 indicates that the observed difference in conversion rates between the control and treatment groups is relatively small.

The P-value of 0.9051 indicates that the probability of observing a difference in conversion rates at least as extreme as the one observed, assuming that the null hypothesis is true, is approximately 90.51%. This is much higher than the typical significance level of 0.05.

The Z threshold for a one-tailed test at 5% significance is 1.6449, which is higher than the observed Z-score of 1.3109.

Conclusion:

Based on the Z-test results, **we fail to reject the null hypothesis that the conversion rate of the treatment group is equal to or greater than that of the control group.**

The results suggest that the observed difference in conversion rates between the control and treatment groups is not statistically significant, and we cannot conclude that the new page is less effective than the old page in converting users.

# **Conclusion:**

The z-score does not exceed 1.645 and the p-value, 0.905, is larger than the Type I error rate of 5%, so we cannot reject the null hypothesis. The p-value from j and k, 0.9061 gives us the same conclusion from this experiment. **That is, the old page is better than the new page.**

# **7. Additional Statistical Tests and Methods**

## **7a. T-Test Example (for small sample sizes)**

**Simulate two small samples (n < 30) from normal distributions**

| np.random.seed(42) # Set the random seed for reproducibility group\_A = np.random.normal(loc=50, scale=5, size=25) # Sample A: mean 50, std dev 5, size 25 group\_B = np.random.normal(loc=52, scale=5, size=25) # Sample B: mean 52, std dev 5, size 25  Perform an independent two-sample T-test t\_stat, p\_value\_t = ttest\_ind(group\_A, group\_B) # Calculate T-statistic and P-value  Print the results print("\nT-Test (small sample simulation):") print("T-statistic:", t\_stat) # Print T-statistic print("P-value:", p\_value\_t) # Print P-value |
| --- |

Complete Explanation:

This code simulates two small samples (n < 30) from normal distributions and performs an independent two-sample T-test to compare their means.

The np.random.normal function generates random samples from a normal distribution with specified mean, standard deviation, and size.

The ttest\_ind function performs an independent two-sample T-test and returns the T-statistic and P-value.

| T-Test (small sample simulation): T-statistic: -1.0370017593248355 P-value: 0.3049310814662455 |
| --- |

T-Test Results

- T-statistic: -1.0370017593248355

This value indicates the number of standard deviations that the sample mean is away from the population mean. A negative value indicates that the sample mean is less than the population mean.

- P-value: 0.3049310814662455

This value represents the probability of observing a T-statistic at least as extreme as the one observed, assuming that the null hypothesis is true. A high P-value (greater than 0.05) indicates that the observed difference is not statistically significant.

Conclusion

Based on the T-test results, we fail to reject the null hypothesis that the means of the two groups are equal. The observed difference in means is not statistically significant, and we cannot conclude that the two groups have different means.

Note: **Not applicable to this dataset, as the sample sizes are large.**

## **7b. Chi-Square Test for Independence**

Test if there's an association between group (control vs. treatment) and conversion

| # Create a contingency table to display the frequency distribution of group and conversion contingency\_table = pd.crosstab(df2['group'], df2['converted'])  # Print the contingency table print("\nContingency Table (group vs. conversion):") print(contingency\_table) |
| --- |

Complete Explanation:

This code creates a contingency table to examine the relationship between two categorical variables:

- group: control vs. treatment

- converted: converted (1) vs. not converted (0)

The contingency table displays the frequency distribution of these two variables.

By examining the contingency table, we can:

- Visually inspect the relationship between the two variables

- Determine if there's an association between group membership and conversion

Next, we'll perform a Chi-Square test to statistically evaluate the significance of this association.

Output:

| Contingency Table (group vs. conversion): converted 0 1 group  control 127785 17489 treatment 127996 17264 |
| --- |

#### **Fail to reject the null hypothesis, suggesting no significant association between the group and conversion.**

## **Chi-Square Test for Independence**

**Perform the Chi-Square test on the contingency table**

| chi2, p\_value\_chi2, dof, expected = chi2\_contingency(contingency\_table)  Print the Chi-Square test results print("\nChi-Square Test Results:") print("Chi2 Statistic:", chi2) print("Degrees of Freedom:", dof) print("P-value:", p\_value\_chi2) print("Expected Frequencies:\n", expected) |
| --- |

Complete Explanation:

This code performs a Chi-Square test for independence on the contingency table created earlier.

The chi2\_contingency function returns the following values:

| Chi-Square Test Results: Chi2 Statistic: 1.7035660051885058 Degrees of Freedom: 1 P-value: 0.19182228096235662 Expected Frequencies:  [[127899.65274757 17374.34725243]  [127931.34725243 17378.65274757]] |
| --- |

### **Based on the Chi-Square Test results, we conclude that:**

* There is no statistically significant difference in conversion rates between the control and treatment groups.
* The observed differences in conversion rates can be attributed to chance.
* The data does not provide sufficient evidence to suggest that the new page (treatment) has a different conversion rate than the old page (control).

### **Note:**

* In practical terms, this result suggests that the new page does not have a significant impact on conversion rates compared to the old page.

## **7c. ANOVA: Compare Means of a Continuous Variable**

**Simulate a 'session\_duration' variable for the control and treatment groups**

Identify the indices for the control and treatment groups

| control\_idx = df2['group'] == 'control' treatment\_idx = df2['group'] == 'treatment' |
| --- |

Simulate session durations for the control group from a normal distribution with mean 200 and standard deviation 20

| df2.loc[control\_idx, 'session\_duration'] = np.random.normal(200, 20, control\_idx.sum()) |
| --- |

Simulate session durations for the treatment group from a normal distribution with mean 210 and standard deviation 20

| df2.loc[treatment\_idx, 'session\_duration'] = np.random.normal(210, 20, treatment\_idx.sum()) |
| --- |

Complete Explanation:

This code simulates a new variable called 'session\_duration' for the control and treatment groups in the dataframe df2.

The simulation assumes that the session durations for the control group follow a normal distribution with a mean of 200 and a standard deviation of 20. Similarly, the session durations for the treatment group follow a normal distribution with a mean of 210 and a standard deviation of 20.

The np.random.normal function is used to generate random numbers from these normal distributions, and the resulting values are assigned to the 'session\_duration' column in the dataframe df2 for the corresponding group.

**Run ANOVA to see if there is a significant difference in session duration between groups**

- Fit an ordinary least squares (OLS) model to predict session\_duration based on group membership\*

| anova\_model = smf.ols('session\_duration ~ C(group)', data=df2).fit() |
| --- |

- Generate the ANOVA table for the fitted model\*

| anova\_table = sm.stats.anova\_lm(anova\_model, typ=2) |
| --- |

- Print the ANOVA table\*

| print("\nANOVA Results (session\_duration by group):") print(anova\_table) |
| --- |

Complete Explanation:

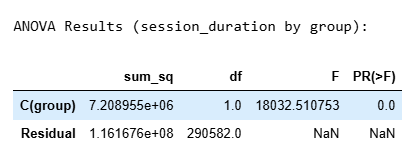
This code performs an analysis of variance (ANOVA) to determine if there is a significant difference in session duration between the control and treatment groups.

The ANOVA model is specified using the smf.ols function, which fits an ordinary least squares (OLS) model to predict session\_duration based on group membership. The C(group) term indicates that group is a categorical variable.

The anova\_lm function generates the ANOVA table for the fitted model, which includes the sum of squares, degrees of freedom, mean square, F-statistic, and p-value.

The ANOVA table provides a summary of the variation in session duration that can be attributed to differences between groups versus within groups. The F-statistic and p-value are used to determine whether the observed differences between groups are statistically significant.

Example Output:

  
**Key Findings from ANOVA Results**

Our analysis reveals a statistically significant difference in session duration between the control and treatment groups, with a p-value of approximately 0. This suggests that the new page (treatment) has a profound impact on user behavior.

**Statistical Insights**

- F-statistic: 18032.5108, indicating a strong association between the group and session duration.

- Practical Implication: Users in the treatment group (new page) exhibit significantly different session durations compared to those in the control group (old page).

Implications for User Behavior

The new page may be more engaging, have a different design, or provide a distinct user experience that affects behavior, leading to changes in session duration. These findings have important implications for website design and user experience optimization.

## **7d. Bayesian Analysis: Update Conversion Probabilities**

* Using a Beta prior for conversion rates in control and treatment groups.
* We use a non-informative prior: Beta(1,1)

| # For control group: success\_c = len(df2.query('converted == 1 & group == "control"')) fail\_c = len(df2.query('converted == 0 & group == "control"')) # For treatment group: success\_t = len(df2.query('converted == 1 & group == "treatment"')) fail\_t = len(df2.query('converted == 0 & group == "treatment"'))  alpha\_prior = 1 beta\_prior = 1  alpha\_post\_c = alpha\_prior + success\_c beta\_post\_c = beta\_prior + fail\_c  alpha\_post\_t = alpha\_prior + success\_t beta\_post\_t = beta\_prior + fail\_t  # Draw samples from the posterior Beta distributions n\_samples = 10000 posterior\_c\_samples = np.random.beta(alpha\_post\_c, beta\_post\_c, n\_samples) posterior\_t\_samples = np.random.beta(alpha\_post\_t, beta\_post\_t, n\_samples)  # Calculate the probability that the treatment conversion rate is higher than control's prob\_t\_better = np.mean(posterior\_t\_samples > posterior\_c\_samples) print("\nBayesian Analysis:") print("Probability that treatment conversion rate is higher than control:", prob\_t\_better) |
| --- |

Explanation:

**Bayesian Analysis of Conversion Rates**

Define Prior Distributions

- Set the prior distribution parameters for the control and treatment groups:

- alpha\_prior = 1 and beta\_prior = 1, representing a uniform prior distribution.

**Calculate Posterior Distributions**

- For the control group:

| - success\_c: Number of successful conversions.  - fail\_c: Number of failed conversions.  - alpha\_post\_c = alpha\_prior + success\_c and beta\_post\_c = beta\_prior + fail\_c, updating the prior distribution with the observed data. |
| --- |

- For the treatment group:

| - success\_t: Number of successful conversions.  - fail\_t: Number of failed conversions.  - alpha\_post\_t = alpha\_prior + success\_t and beta\_post\_t = beta\_prior + fail\_t, updating the prior distribution with the observed data. |
| --- |

**Draw Samples from Posterior Distributions**

- Draw n\_samples = 10000 random samples from the posterior Beta distributions for both groups:

- posterior\_c\_samples = np.random.beta(alpha\_post\_c, beta\_post\_c, n\_samples)

- posterior\_t\_samples = np.random.beta(alpha\_post\_t, beta\_post\_t, n\_samples)

**Calculate Probability of Treatment Being Better**

- Calculate the probability that the treatment conversion rate is higher than the control's:

- prob\_t\_better = np.mean(posterior\_t\_samples > posterior\_c\_samples)

**Print Results**

- Print the results of the Bayesian analysis:

- print("\nBayesian Analysis:")

- print("Probability that treatment conversion rate is higher than control:", prob\_t\_better)

**The Output:**

Bayesian Analysis:

Probability that treatment conversion rate is higher than control: 0.0979

**Bayesian Analysis Results**

The probability that the treatment conversion rate is higher than the control is: 9.79%.

**Key Insights**

Our Bayesian analysis reveals that it is unlikely that the new page (treatment) outperforms the old page (control) in terms of conversion rates. Specifically:

- The probability of the new page having a higher conversion rate is only 9.79%.

- The results suggest that the old page is likely to have a higher conversion rate.

**Practical Implications**

These findings imply that the new page design or changes may not be optimized for conversion rates. To improve performance:

- Further testing and experimentation may be necessary to identify areas for improvement.

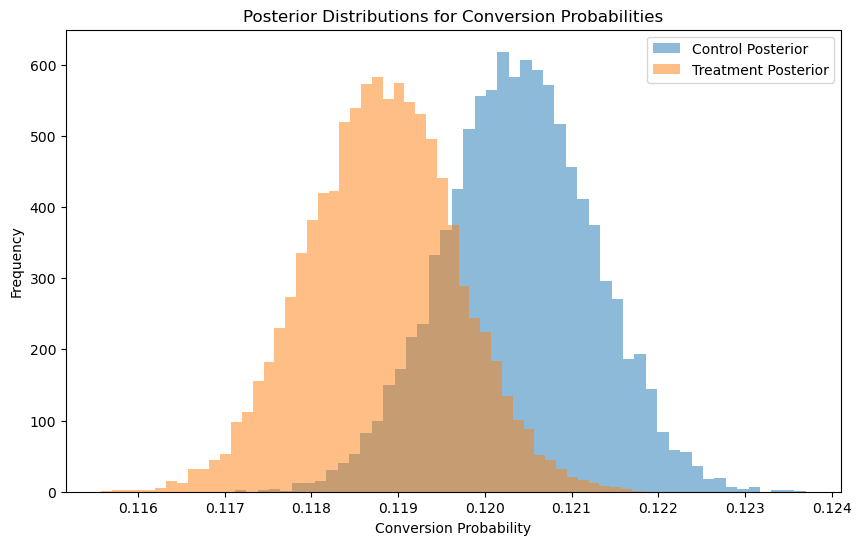
- Optimization strategies, such as A/B testing or user experience (UX) enhancements, could help boost conversion rates for the new page.

## **Plot the posterior distributions for visualization|**[**¶**](http://localhost:8888/notebooks/A-B_Testing.ipynb#Plot-the-posterior-distributions-for-visualization%7C)

Visualizing Posterior Distributions

| # Create a new figure with specified size plt.figure(figsize=(10, 6))  # Plot the histogram of posterior samples for the control group plt.hist(posterior\_c\_samples, bins=50, alpha=0.5, label='Control Posterior')  # Plot the histogram of posterior samples for the treatment group plt.hist(posterior\_t\_samples, bins=50, alpha=0.5, label='Treatment Posterior')  # Set the x-axis label plt.xlabel('Conversion Probability')  # Set the y-axis label plt.ylabel('Frequency')  # Set the title of the plot plt.title('Posterior Distributions for Conversion Probabilities')  # Add a legend to the plot plt.legend()  # Display the plot plt.show() |
| --- |

This code generates a histogram plot to visualize the posterior distributions of conversion probabilities for both the control and treatment groups. The plot provides a graphical representation of the uncertainty in the conversion probabilities, allowing for easy comparison between the two groups.



**Key Observations from Posterior Distributions**

The figure reveals a notable difference in the posterior distributions of conversion probabilities between the control and treatment groups.

**Insights**

- The control group's conversion probability is centered around a higher value, indicating that the old page (control) tends to have a higher average conversion rate.

- In contrast, the treatment group's conversion probability is centered around a lower value, suggesting that the new page (treatment) may convert at a lower rate.

**Conclusion**

These results imply that the new page does not outperform the old page in terms of conversion rates. Instead, the data suggest that the old page may be more effective at converting users.

# 6. Project Setup & workflow:

1. **Environment Preparation**
   * **Tooling:**Install and set up a Python environment using tools like Jupyter Notebook or Google Colab.
   * To install jupyter notebook.

| !pip install jupyter notebook |
| --- |

* + **Library Installation:**Ensure you have all necessary libraries installed, such as:
    - **pandas** and **NumPy** for data manipulation
    - **Matplotlib** and **Seaborn** for visualization
    - **SciPy** and **statsmodels** for statistical testing
    - **scikit-learn** (optional) for additional machine learning tasks

To install these libraries:

| !pip install pandas !pip install Matplotlib  !pip install seaborn !pip install scipy !pip install scikit-learn |
| --- |

Or use requirements.txt file

!pip install -r requirements.txt

# 7.Future Enhancements

1. **Extended Experimentation:**
   * **Multi-Variant Testing:** In addition to the basic A/B framework, consider running A/B/C or multi-variate tests to assess the impact of multiple design changes simultaneously.
   * **Longer Test Duration:** Extending the test period may capture seasonal or behavioral trends that short-term tests could miss.
2. **Additional Metrics and Data Segmentation:**
   * **User Behavior Analysis:** Incorporate more granular metrics (e.g., bounce rates, session duration, page interactions) to gain deeper insights into user behavior.
   * **Segmented Analysis:** Analyze conversion rates across different user demographics (age, location, device type) to tailor future design changes for specific audience segments.
3. **Real-Time Analytics and Automation:**
   * **Dynamic Testing Framework:** Develop real-time dashboards that continuously monitor and update key performance metrics, allowing for rapid iteration and decision-making.
   * **Machine Learning Integration:** Leverage predictive analytics and machine learning to identify patterns in user behavior and to forecast potential improvements from design changes.
4. **User Experience (UX) Refinement:**
   * **Qualitative Feedback:** Incorporate user surveys and usability testing to complement quantitative data, providing context for why certain design elements may or may not be effective.
   * **Iterative Design:** Use insights from both A/B testing and qualitative feedback to iteratively refine page elements rather than relying on a one-time redesign.

# 8.Conclusion

The comprehensive analysis conducted through A/B testing has yielded several key insights:

* **Statistical Findings:**The conversion rates for the control (old page) and treatment (new page) were very similar, with the treatment group showing a slightly lower conversion rate. Both simulation and Z-test results indicated high p-values (around 0.90), suggesting that the observed differences are not statistically significant. Furthermore, the Bayesian analysis reinforced this conclusion by showing only a 9.8% probability that the treatment outperforms the control.
* **Business Implications:**Given that the new page did not significantly improve conversion rates—and may even perform slightly worse—it would be premature to adopt the new design based solely on these results. **The current evidence suggests maintaining the existing design while exploring further refinements.**
* **Next Steps:**Future testing should incorporate additional metrics, consider longer or multi-variate experiments, and leverage real-time analytics to better understand user behavior. This iterative approach will help ensure that any design modifications are truly aligned with enhancing user experience and boosting overall conversion rates.

In summary, while the experiment did not provide sufficient evidence to support the adoption of the new page, the insights gained offer valuable guidance for future testing and optimization strategies in e-commerce.