# Data cleaning and exploratory analysis

A/B TESTING IN PYTHON

**Moe Lotfy, PhD**
Principal Data Science Manager

# Cleaning missing values

- **Missing values**
  - Drop, ignore, impute

```
# Calculate the mean order value
checkout.order_value.mean()
```

```
30.0096
```

```
# Replace missing values with zeros and get mean
checkout['order_value'].fillna(0).mean()
```

```
25.3581
```

# Cleaning duplicates

- **Duplicates**
  - Identical rows should be dropped

```
# Check for duplicate rows due to logging issues
print(len(checkout))
print(len(checkout.drop_duplicates(keep='first')))
```

```
9000
9000
```

# Cleaning duplicates

- Duplicates
  - Duplicate users should be handled with care.

```
# Unique users in group B
print(checkout[checkout['checkout_page'] == 'B']['user_id'].nunique())
# Unique users who purchased at least once
print(checkout[checkout['checkout_page'] == 'B'].groupby('user_id')['purchased'].max().sum())
# Total purchase events in group B
print(checkout[checkout['checkout_page'] == 'B']['purchased'].sum())
```

```
2938
2491.0
2541.0
```

# EDA summary stats

- Mean, count, and standard deviation summary

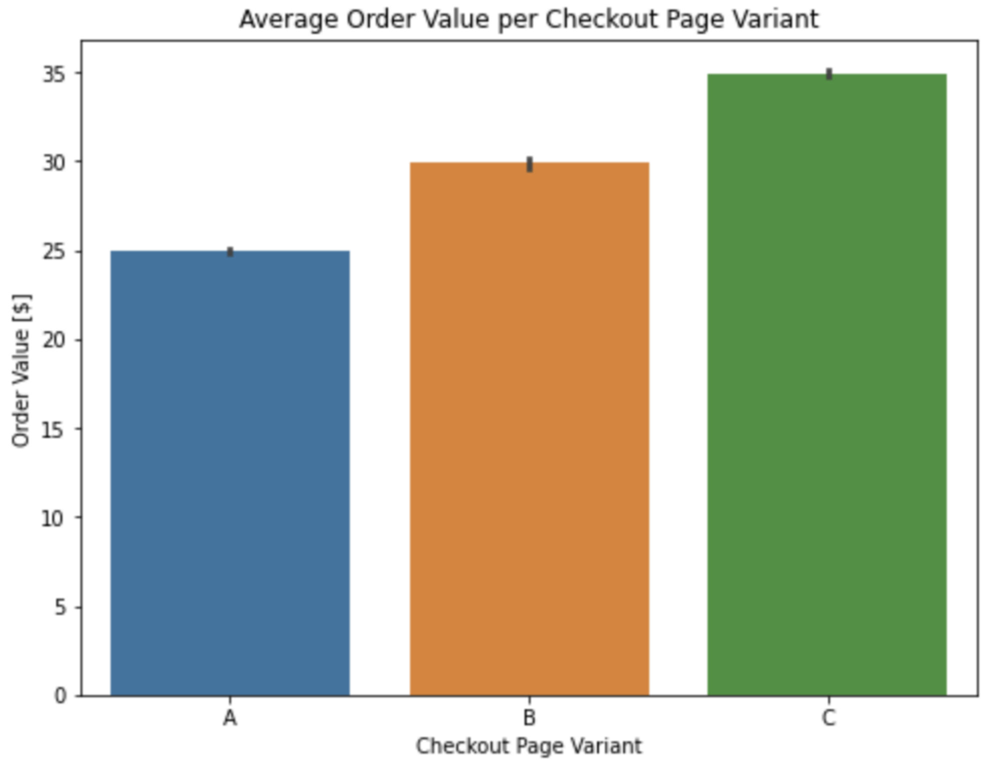
```
checkout.groupby('checkout_page')['order_value'].agg({'mean','std','count'})
```

```
                     mean      count     std
checkout_page
        A         24.956437     2461     2.418837
        B         29.876202     2541     7.277644
        C         34.917589     2603     4.869816
```

# EDA plotting

- **Bar plots**

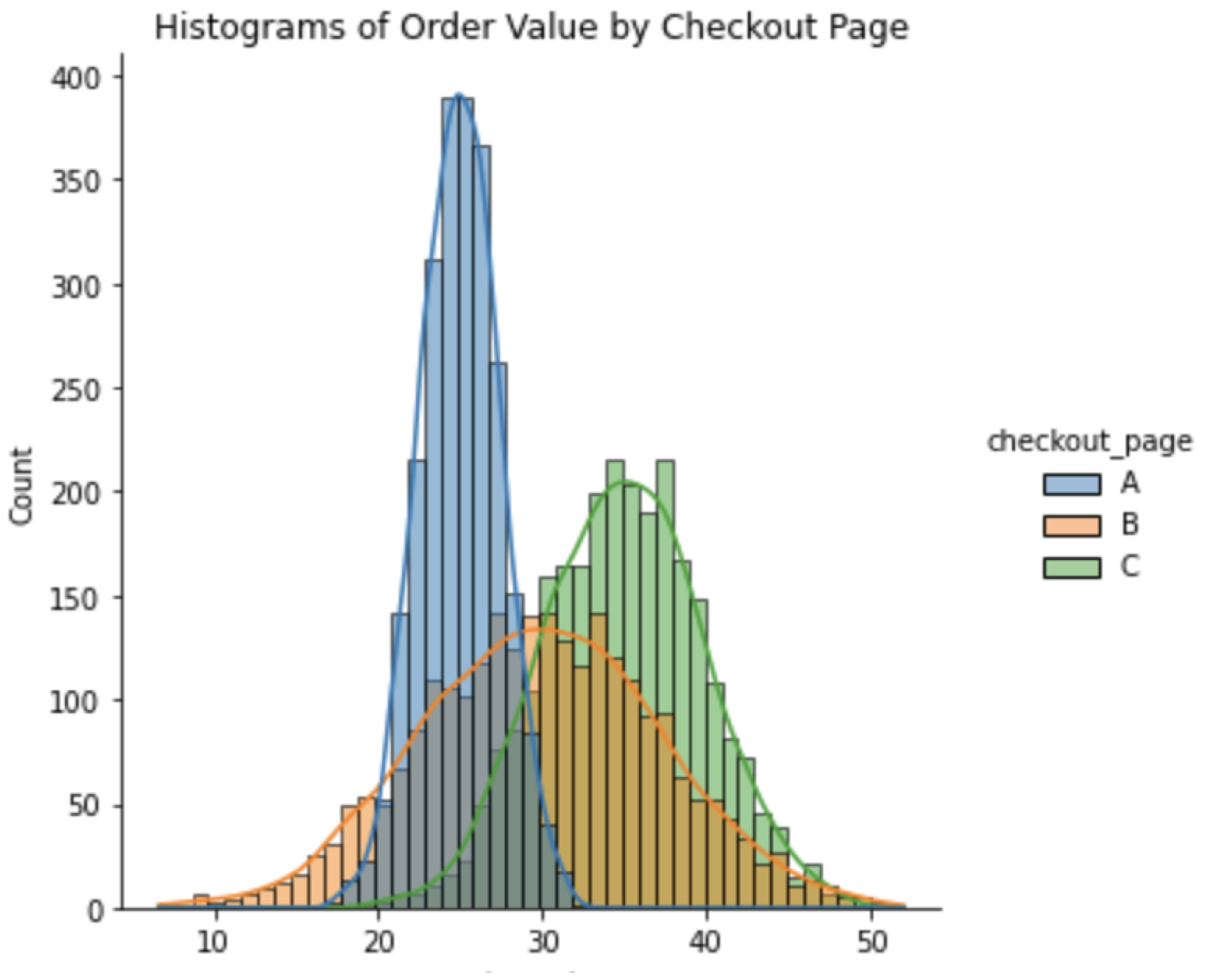
```
sns.barplot(x=checkout['checkout_page'], y=checkout['order_value'], estimator=np.mean)
plt.title('Average Order Value per Checkout Page Variant')
plt.xlabel('Checkout Page Variant')
plt.ylabel('Order Value [$]')
```



# EDA plotting

- **Histograms**

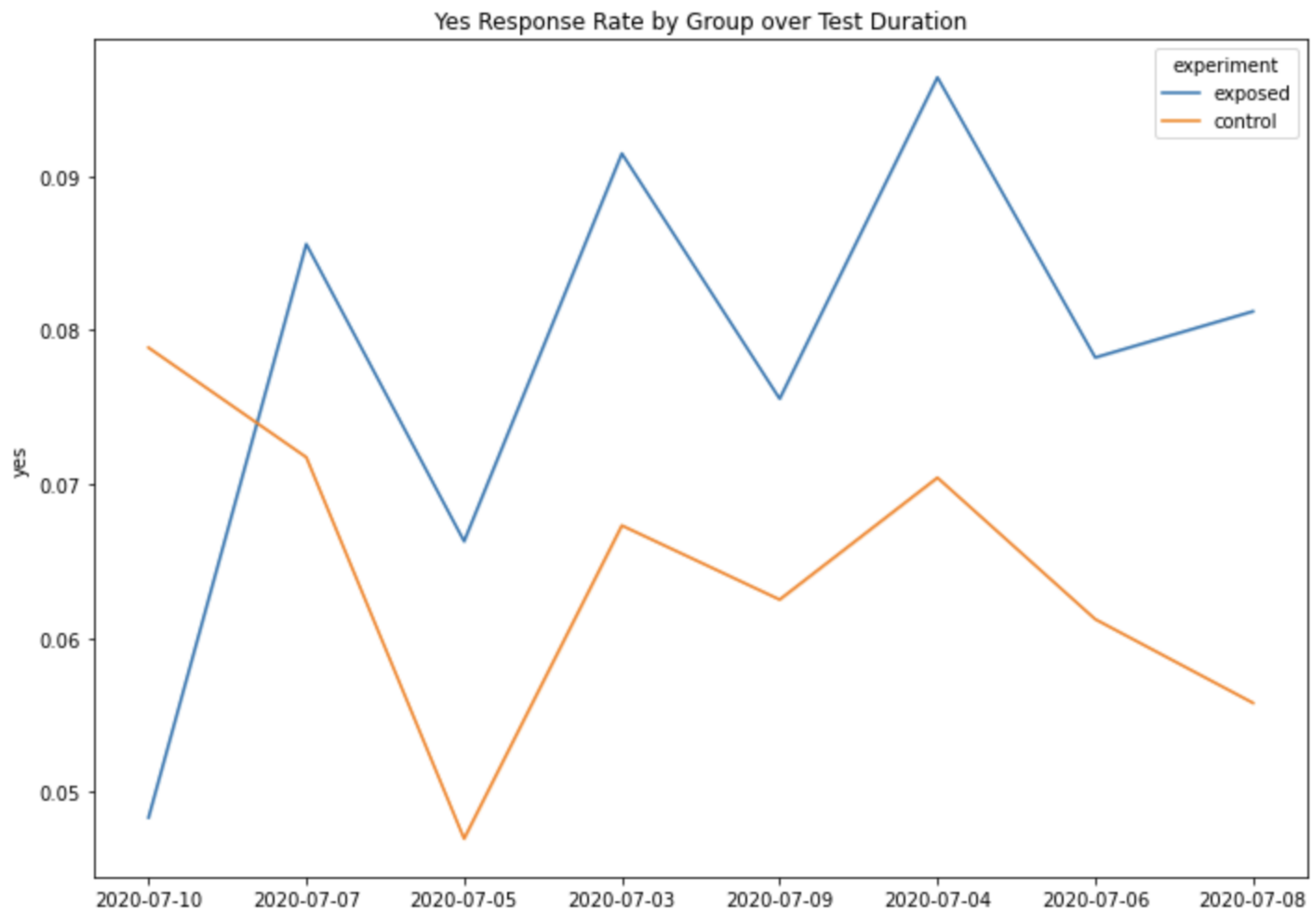
```
sns.displot(data=checkout, x='order_value', hue = 'checkout_page', kde=True)
```



# EDA plotting

- **Time series (line plots)**

```
sns.lineplot(data=AdSmart,x='date', y='yes', hue='experiment', errorbar=None)
```




[1] Adsmart Kaggle dataset: https://www.kaggle.com/datasets/osuolaleemmanuel/ad-ab-testing

# Let's practice!

A/B TESTING IN PYTHON

# Sanity checks: Internal validity

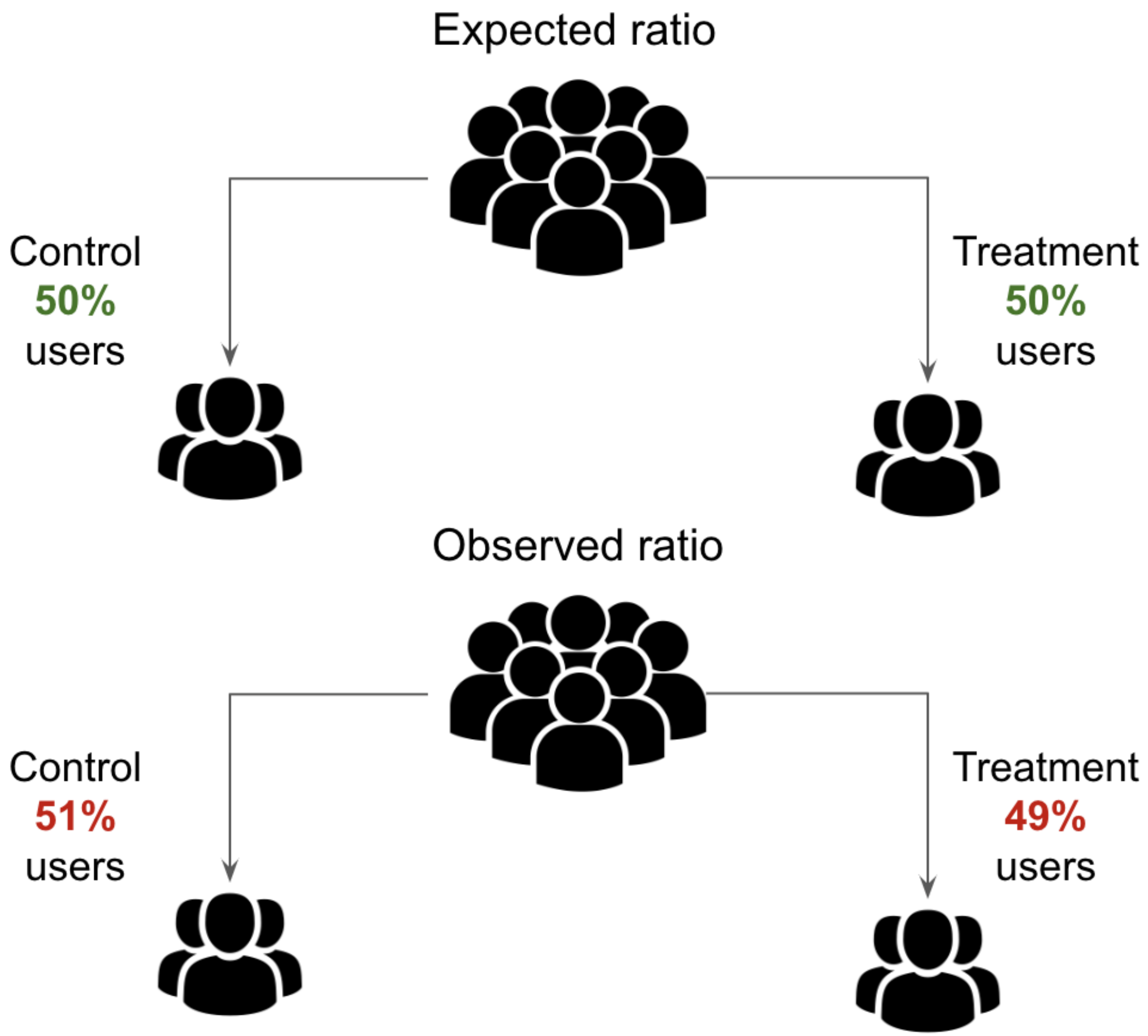
A/B TESTING IN PYTHON

Moe Lotfy, PhD
Principal Data Science Manager

# Sample Ratio Mismatch (SRM)

- Sample Ration Mismatch (SRM)
  - Allocation across variants deviates from design
- Chi-square goodness of fit test

$$\chi^2 = \sum \frac{(\text{Observed - Expected})^2}{\text{Expected}}$$



# SRM python example

```python
# Calculate the unique IDs per variant
AdSmart.groupby('experiment')['auction_id'].nunique()
```

```
experiment
control     4071
exposed     4006
```

```python
# Assign the unqiue counts to each variant
control_users=AdSmart[AdSmart['experiment']=='control']['auction_id'].nunique()
exposed_users=AdSmart[AdSmart['experiment']=='exposed']['auction_id'].nunique()
total_users=control_users+exposed_users
# Calculate allocation ratios per variant
control_perc = control_users / total_users
exposed_perc = exposed_users / total_users
print("Percentage of users in the Control group:",100*round(control_perc,5),"%")
print("Percentage of users in the Exposed group:",100*round(exposed_perc,5),"%")
```

```
Percentage of users in the Control group: 50.402 %
Percentage of users in the Exposed group: 49.598 %
```

[1] Adsmart Kaggle dataset: https://www.kaggle.com/datasets/osuolaleemmanuel/ad-ab-testing

# SRM python example

```python
# Creat lists of observed and expected counts per variant
observed = [ control_users, exposed_users ]
expected = [ total_users/2, total_users/2 ]
# Import chisquare from scipy library
from scipy.stats import chisquare
# Run chisquare test on observed and expected lists
chi = chisquare(observed, f_exp=expected)
# Print test results and interpretation
print(chi)
if chi[1] < 0.01:
    print("SRM may be present")
else:
    print("SRM likely not present")
```

```
Power_divergenceResult(statistic=0.5230902562832735, pvalue=0.4695264353014863)
SRM likely not present
```

[1] Adsmart Kaggle dataset: https://www.kaggle.com/datasets/osuolaleemmanuel/ad-ab-testing

# SRM root-causing

**Common causes of SRM:**[1]

- Assignment: incorrect bucketing or faulty randomization functions
- Execution: delayed variants starting time or ramp up rates
- Data logging: logging delays or bot filtering
- Interference: experimenter pausing a variant

[1] Diagnosing Sample Ratio Mismatch in Online Controlled Experiments: A Taxonomy and Rules of Thumb for Practitioners

# A/A tests

- A/A test
  - Presents an identical experience to two groups of users
  - Reveals bugs in experimental setup
  - No statistically significance differences between the metrics
  - False positives can still happen at the specified $\alpha$ (5% of the time)
  - Reveals imbalances in distributions across groups (e.g. browsers, devices, etc.)

# Distributions balance Python example

- Balanced browsers distribution
- Valid test

```
checkout.groupby('checkout_page')['browser'].value_counts(normalize=True)
```

```
checkout_page  browser
A              chrome     0.341333
               safari     0.332000
               firefox    0.326667
B              safari     0.352000
               firefox    0.325000
               chrome     0.323000
C              safari     0.346000
               chrome     0.330000
               firefox    0.324000
```

- Imbalanced browsers distribution
- Invalid test

```
AdSmart.groupby('experiment')['browser'].value_counts(normalize=True)
```

```
experiment  browser
control     Chrome Mobile                 0.591992
            Facebook                      0.137804
            Samsung Internet              0.120855
            Chrome Mobile WebView         0.071727
            Mobile Safari                 0.060427
            Chrome Mobile iOS             0.008352
            Mobile Safari UI/WKWebView    0.007369
exposed     Chrome Mobile                 0.535197
            Chrome Mobile WebView         0.298802
            Samsung Internet              0.082876
            Facebook                      0.050674
            Mobile Safari                 0.022716
            Chrome Mobile iOS             0.004244
```

[1] Adsmart Kaggle dataset: https://www.kaggle.com/datasets/osuolaleemmanuel/ad-ab-testing

# Let's practice!

A/B TESTING IN PYTHON

# Sanity checks: external validity

A/B TESTING IN PYTHON

**Moe Lotfy, PhD**
Principal Data Science Manager

# Simpson's paradox

**Simpson's Paradox:** a statistical phenomenon where certain trends between variables emerge, disappear or reverse when the population is divided into segments.

```
print(simp_imbalanced.groupby('Variant').mean())
```

```
Variant     Conversion
A              0.80
B              0.64
```

```
print(simp_imbalanced.groupby(['Variant','Device']).mean())
```

```
Variant Device    Conversion
A       Phone        0.875
        Tablet       0.500
B       Phone        0.900
        Tablet       0.575
```

# Simpson's paradox

```
simp_imbalanced.groupby(['Variant','Device'])\
                            ['Device'].count()
```

```
Variant  Device
A        Phone     40
         Tablet    10
B        Phone     10
         Tablet    40
```

| Device | A | B |
|---|---|---|
| Phone | 35/40 | 9/10 |
| Tablet | 5/10 | 23/40 |
| Total | 40/50 | 32/50 |

| Device | A | B |
|---|---|---|
| Phone | 0.875 | 0.9 |
| Tablet | 0.5 | 0.575 |
| Total | 0.8 | 0.64 |

# Simpson's paradox

```
simp_balanced.groupby(['Variant','Device'])\
                                ['Device'].count()
```

```
Variant  Device
A        Phone     40
         Tablet    10
B        Phone     40
         Tablet    10
```

```
print(simp_balanced.groupby('Variant').mean())
```

```
Variant     Conversion
A              0.70
B              0.52
```

```
print(simp_balanced.groupby(['Variant','Device']).mean())
```

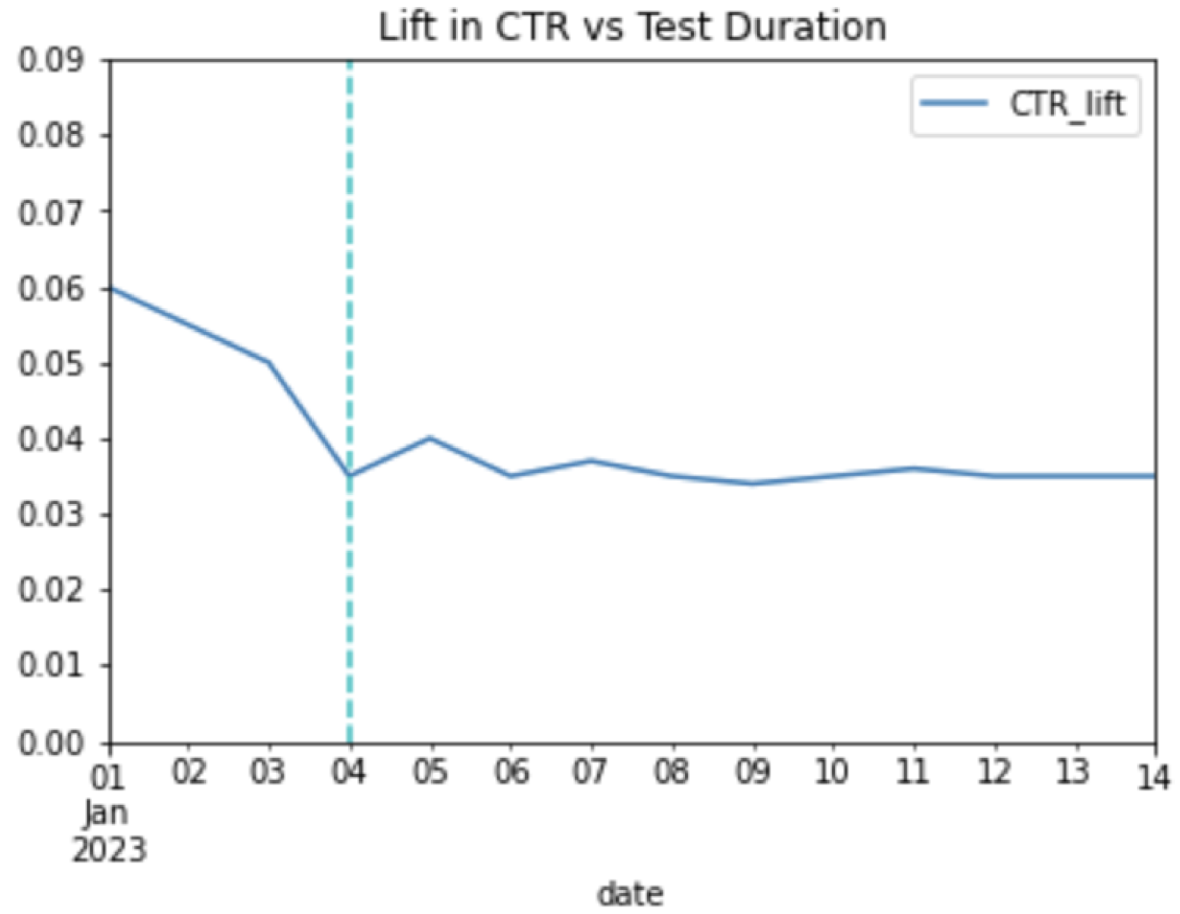
```
Variant Device     Conversion
A       Phone        0.750
        Tablet       0.500
B       Phone        0.575
        Tablet       0.300
```

# Novelty effect

- **Novelty effect**
  - A short-lived improvement in metrics caused by users' curiosity about a new feature.
- **Change aversion**
  - The opposite of novelty effect.
  - Users avoiding trying a new feature due to familiarity with the old one.

# Novelty effect visual inspection

```
# Plot Lift in CTR vs test days
novelty.plot('date', 'CTR_lift')
plt.ylim([0, 0.09])
plt.title('Lift in CTR vs Test Duration')
plt.show()
```



# Correcting for novelty effects

- Increasing the test duration
  - Start including data after treatment effect stabilizes.
- Examine new and returning user cohorts
  - New users are by default less likely to experience novelty effects.
  - Old users compare consider their old experiences.

# Let's practice!

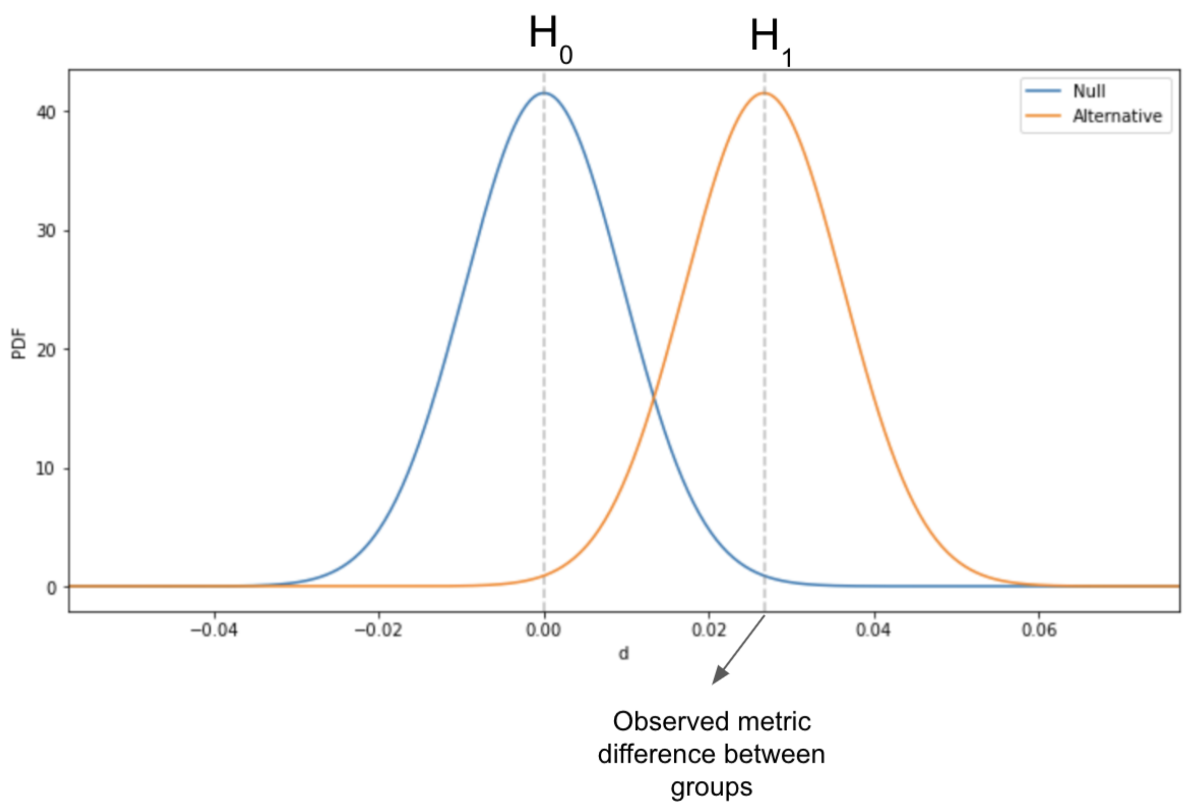
A/B TESTING IN PYTHON

# Analyzing difference in proportions A/B tests

A/B TESTING IN PYTHON


**Moe Lotfy, PhD**
Principal Data Science Manager

# Framework for difference in proportions




$$d = p_B - p_A$$

$$H_0 : d = p_B - p_A = 0$$

$$H_1 : d = p_B - p_A \neq 0$$

- If p-value < $\alpha$
  - Reject Null hypothesis
- If p-value > $\alpha$
  - Fail to reject Null hypothesis
- Confidence intervals
  - 95% CI is the range that captures the true difference 95% of the time
  - Like fishing with a net instead of a spear
  - Centered around the observed difference between the treatment and the control

# Two sample proportions z-test

```python
from statsmodels.stats.proportion import proportions_ztest, proportion_confint
# Calculate the number of users in groups A and B
n_A = checkout[checkout['checkout_page'] == 'A']['user_id'].nunique()
n_B = checkout[checkout['checkout_page'] == 'B']['user_id'].nunique()
print('Group A users:',n_A)
print('Group B users:',n_B)
```

```
Group A users: 2940
Group B users: 2938
```

```python
# Compute unique purchasers in each group
puchased_A = checkout[checkout['checkout_page'] == 'A'].groupby('user_id')['purchased'].max().sum()
purchased_B = checkout[checkout['checkout_page'] == 'B'].groupby('user_id')['purchased'].max().sum()
# Assign groups lists
purchasers_abtest = [puchased_A, purchased_B]
n_abtest = [n_A, n_B]
```

# Two sample proportions z-test

```python
# Calculate p-value and confidence intervals
z_stat, pvalue = proportions_ztest(purchasers_abtest, nobs=n_abtest)
(A_lo95, B_lo95), (A_up95, B_up95) = proportion_confint(purchasers_abtest, nobs=n_abtest, alpha=0.05)
# Print the p-value and confidence intervals
print(f'p-value: {pvalue:.4f}')
print(f'Group A 95% CI : [{A_lo95:.4f}, {A_up95:.4f}]')
print(f'Group B 95% CI : [{B_lo95:.4f}, {B_up95:.4f}]')
```

```
p-value: 0.0058
Group A 95% CI : [0.8072, 0.8349]
Group B 95% CI : [0.8349, 0.8608]
```

# Confidence intervals for proportions

```
# Set random seed for repeatability
np.random.seed(34)
# Calculate the average purchase rate for group A
pop_mean = checkout[checkout['checkout_page'] == 'B']['purchased'].mean()
print(pop_mean)
```

```
0.847
```

# Confidence intervals for proportions

```python
# Calculate 20 90% confidence intervals for 20 random samples of size 100 each
for i in range(20):
    confidence_interval = proportion_confint(
        count = checkout[checkout['checkout_page'] == 'B'].sample(100)['purchased'].sum(),
        nobs = 100,
        alpha = (1 - 0.90))
    print(confidence_interval)
```

```
(0.7912669777384846, 0.9087330222615153)
(0.8385342148455946, 0.9414657851544054)
(0.8265485838585659, 0.9334514161414341)
(0.7568067872454262, 0.8831932127545737)
(0.8506543911914558, 0.9493456088085442)*
(0.8385342148455946, 0.9414657851544054)
(0.7230037568938057, 0.8569962431061944)
(0.8146830076144598, 0.9253169923855402)
(0.8029257122801267, 0.9170742877198733)
(0.8146830076144598, 0.9253169923855402)
(0.8506543911914558, 0.9493456088085442)*
(0.7454722433688197, 0.8745277566311804)
...
```

# Let's practice!

A/B TESTING IN PYTHON