

Hypothesis formulation and distributions

A/B TESTING IN PYTHON



Moe Lotfy, PhD

Principal Data Science Manager

Defining hypotheses

- A **hypothesis** is:
 - a statement explaining an event
 - a starting point for further investigation
 - an idea we want to test
- A strong hypothesis:
 - is testable, declarative, concise, and logical
 - enables systematic iteration
 - is easier to generalize and confirm understanding
 - results in actionable/focused recommendations

Hypothesis format

- **General framing format:**
 - Based on X, we believe that if we do Y
 - Then Z will happen
 - As measured by metric(s) M
- **Example of the alternative hypothesis:**
 - Based on user experience research, we believe that if we update our checkout page design
 - Then the percentage of purchasing customers will increase
 - As measured by purchase rate
- **Null hypothesis:** ...the percentage of purchasing customers will not change...

Calculating sample statistics

```
# Calculate the number of users in groups A and B  
n_A = checkout[checkout['checkout_page'] == 'A']['purchased'].count()  
n_B = checkout[checkout['checkout_page'] == 'B']['purchased'].count()  
print('Group A users:', n_A)  
print('Group B users:', n_B)
```

Group A users: 3000

Group B users: 3000

```
# Calculate the mean purchase rates of groups A and B  
p_A = checkout[checkout['checkout_page'] == 'A']['purchased'].mean()  
p_B = checkout[checkout['checkout_page'] == 'B']['purchased'].mean()  
print('Group A mean purchase rate:', p_A)  
print('Group B mean purchase rate:', p_B)
```

Group A mean purchase rate: 0.820

Group B mean purchase rate: 0.847

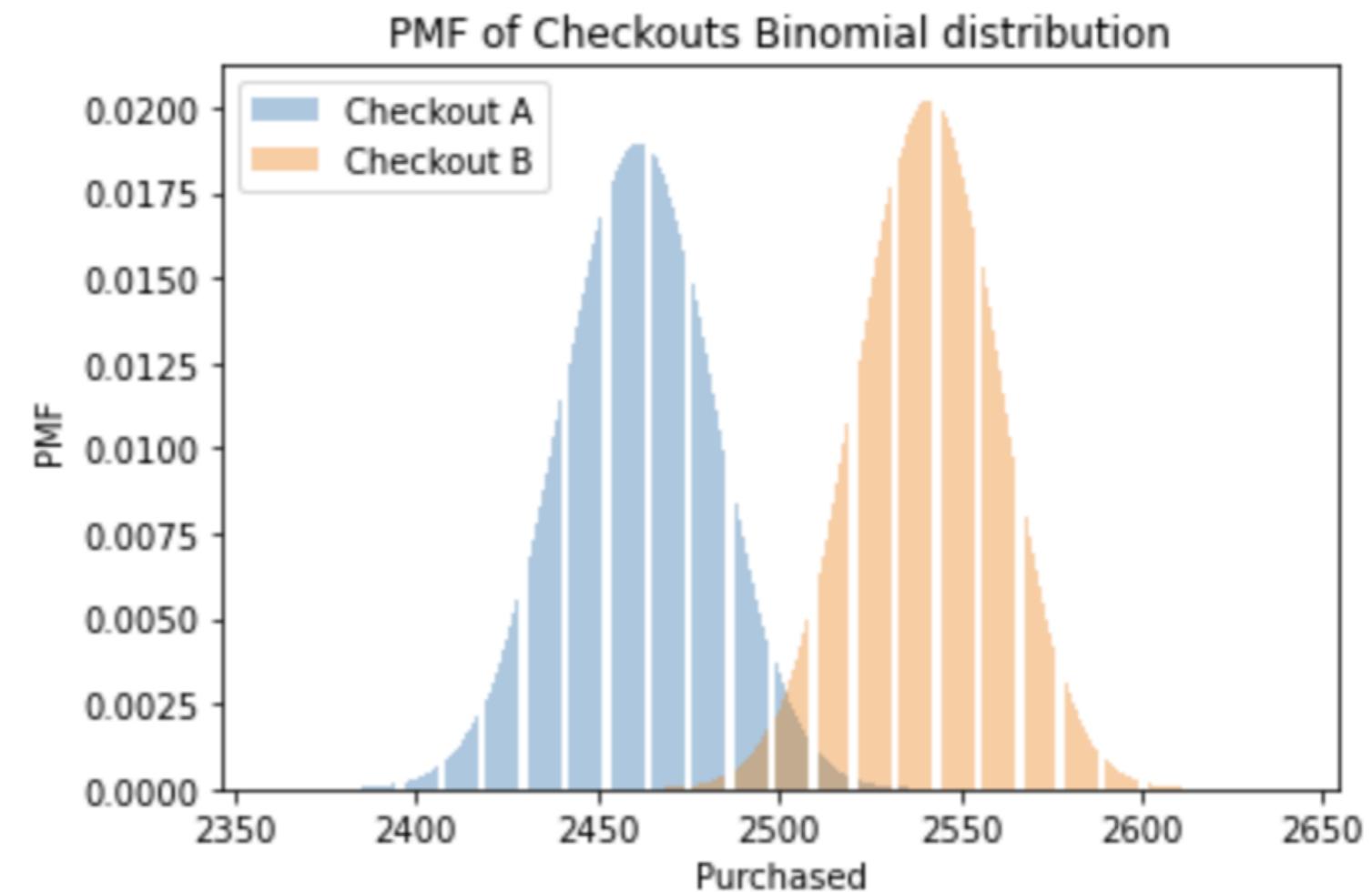
Simulating and plotting distributions

The number of purchasers in n trials with purchasing probability p is Binomially distributed.

```
# Import binom from scipy library
from scipy.stats import binom

# Create x-axis range and Binomial distributions A and B
x = np.arange(n_A*p_A - 100, n_B*p_B + 100)
binom_a = binom.pmf(x, n_A, p_A)
binom_b = binom.pmf(x, n_B, p_B)

# Plot Binomial distributions A and B
plt.bar(x, binom_a, alpha=0.4, label='Checkout A')
plt.bar(x, binom_b, alpha=0.4, label='Checkout B')
plt.xlabel('Purchased')
plt.ylabel('PMF')
plt.title('PMF of Checkouts Binomial distribution')
plt.show()
```



Central limit theorem

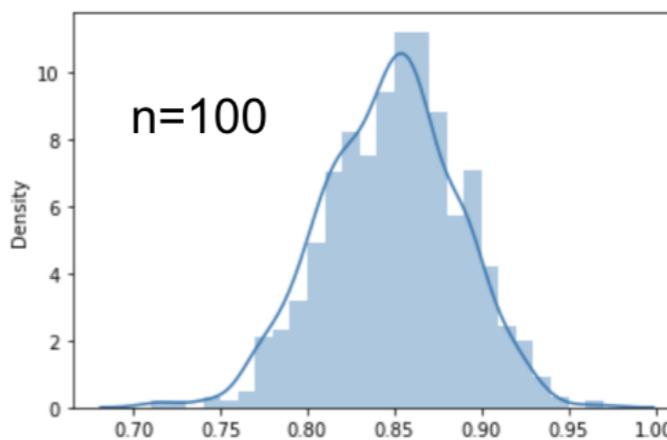
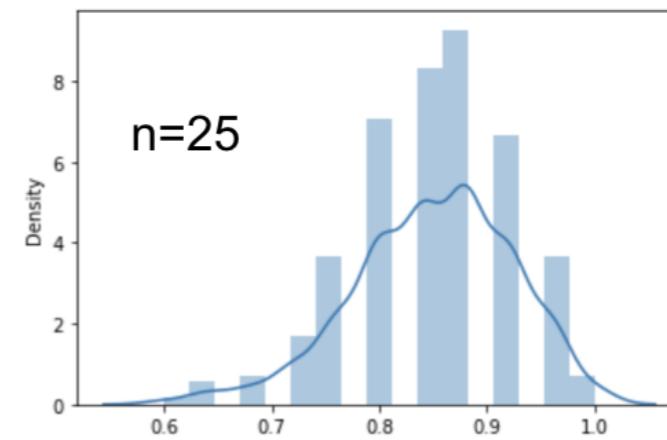
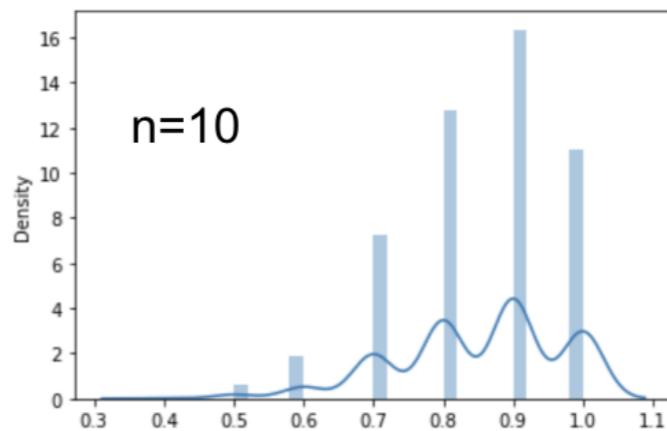
For a sufficiently large sample size, the distribution of the sample means, \bar{p} , will be

- normally distributed around the true population mean
- with a standard deviation = standard error of the mean
- irrespective of the distribution of the underlying data

$$\bar{p} \sim \text{Normal} \left(\mu=p, \sigma=\frac{\sqrt{p(1-p)}}{\sqrt{n}} \right)$$

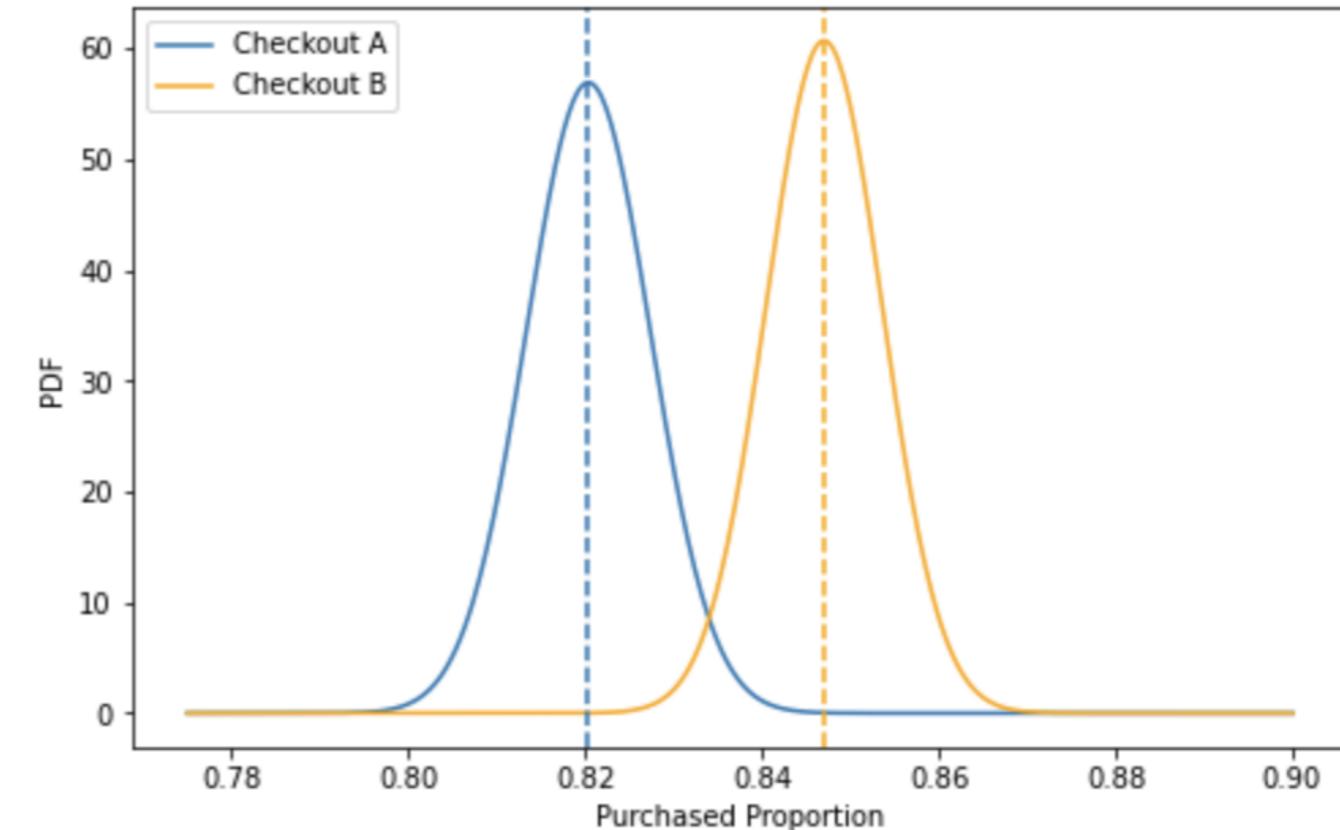
Central limit theorem in python

```
# Set random seed for repeatability
np.random.seed(47)
# Create an empty list to hold means
sampled_means = []
# Create loop to simulate 1000 sample means
for i in range(1000):
    # Take a sample of n=100
    sample = checkout['purchased'].sample(100,replace=True)
    # Get the sample mean and append to list
    sample_mean = np.mean(sample)
    sampled_means.append(sample_mean)
# Plot distribution
sns.displot(sampled_means, kde=True)
plt.show()
```



Hypothesis mathematical representation

```
# Import norm from scipy library
from scipy.stats import norm
# Create x-axis range and normal distributions A and B
x = np.linspace(0.775, 0.9, 500)
norm_a = norm.pdf(x, p_A, np.sqrt(p_A*(1-p_A) / n_A))
norm_b = norm.pdf(x, p_B, np.sqrt(p_B*(1-p_B) / n_B))
# Plot normal distributions A and B
sns.lineplot(x=x, y=norm_a, ax=ax, label='Checkout A')
sns.lineplot(x=x, y=norm_b, color='orange', \
             ax=ax, label= 'Checkout B')
ax.axvline(p_A, linestyle='--')
ax.axvline(p_B, linestyle='--')
plt.xlabel('Purchased Proportion')
plt.ylabel('PDF')
plt.legend(loc="upper left")
plt.show()
```



$$d = p_B - p_A$$

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

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

Let's practice!

A/B TESTING IN PYTHON

Experimental design: setting up testing parameters

A/B TESTING IN PYTHON



Moe Lotfy, PhD

Principal Data Science Manager

Distribution parameters

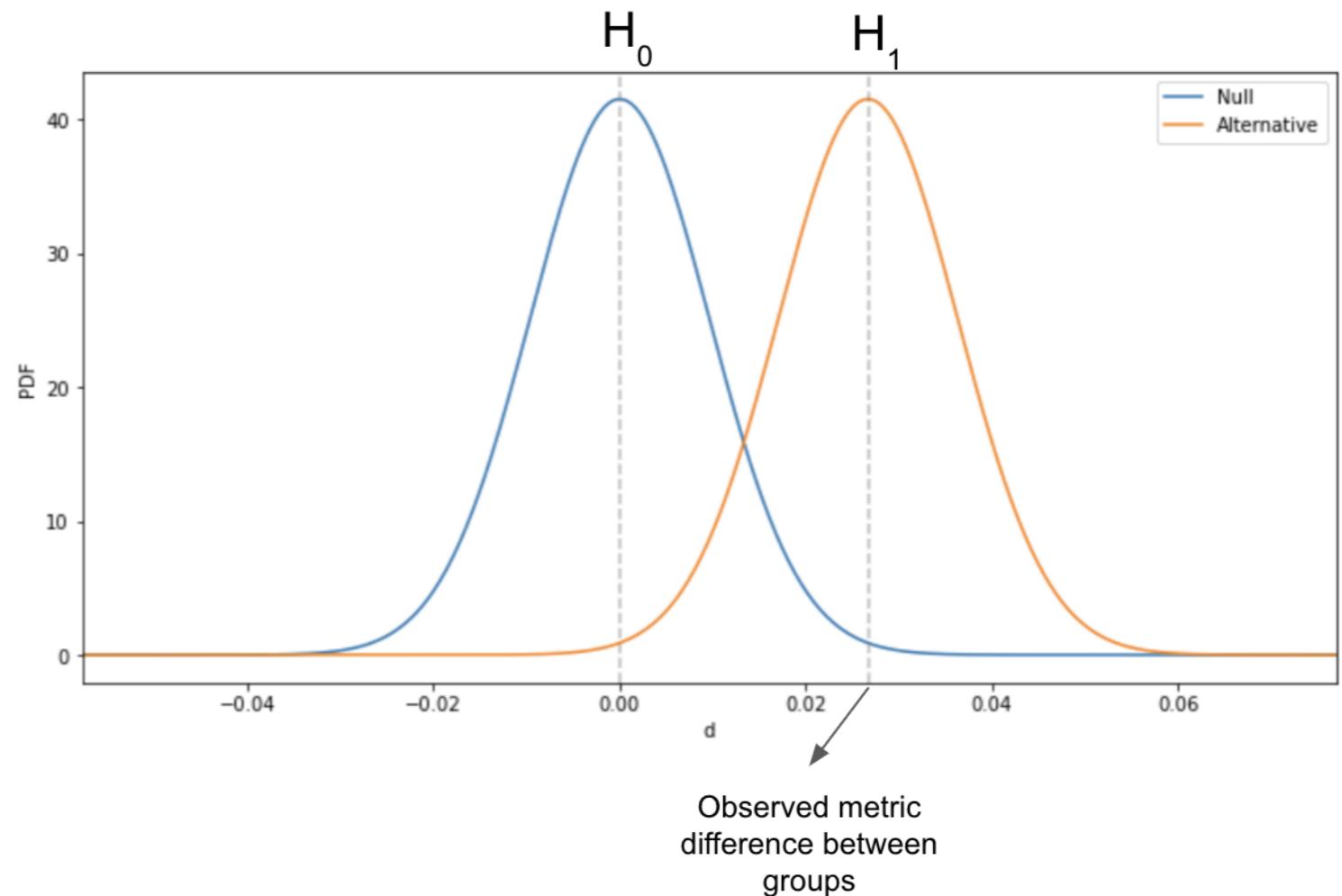
- d follows a normal distribution
- Null vs alternative hypothesis distributions

$$d = p_B - p_A$$

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

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

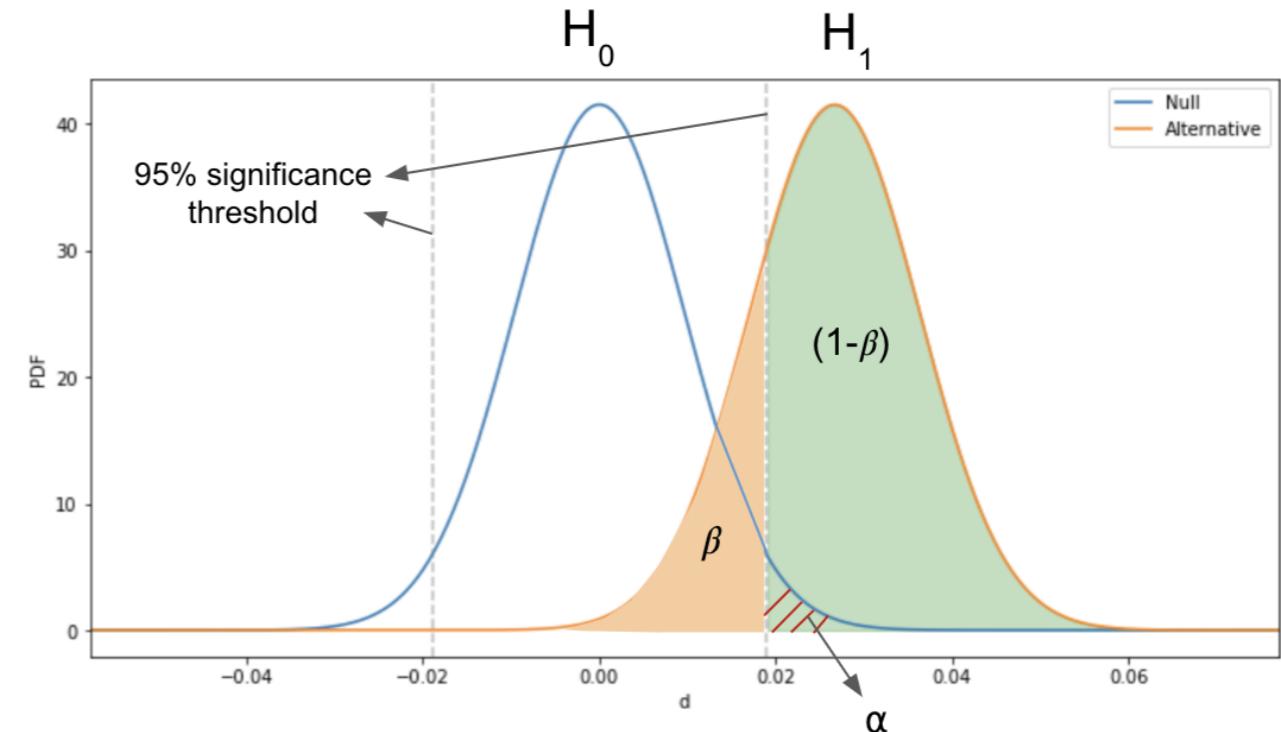
- If observed difference ' d ' is unlikely:
 - reject the Null hypothesis



Design parameters and error types

- Power ($1 - \beta$)
 - β = Type II error = False negative
 - Commonly set at 80%
- Minimum Detectable Effect (MDE)
 - Smallest difference we care to capture

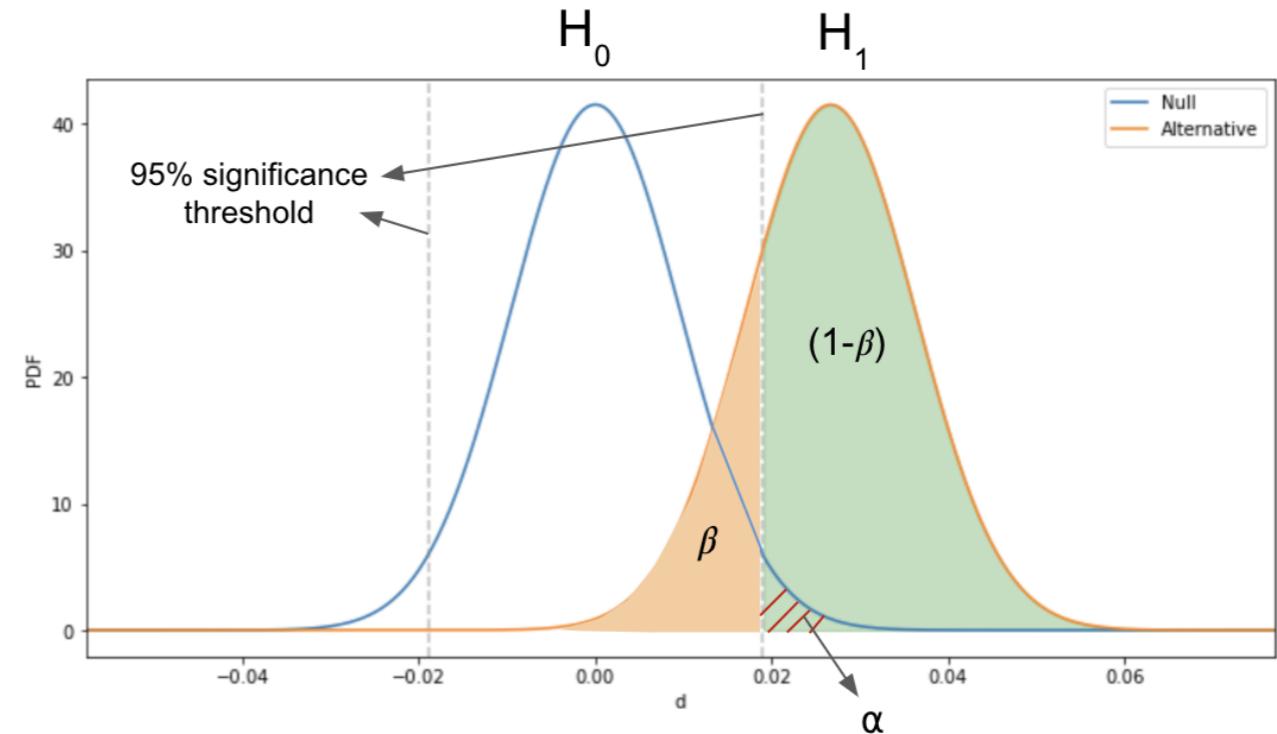
| | | Truth about our users |
|------------------------|--|--|
| Decision based on data | H_0 is true | H_0 is false |
| Fail to reject H_0 | Correct Decision (probability = $1 - \alpha$) | Type II Error - False Negative (probability = β) |
| Reject H_0 | Type I Error - False Positive (probability = α) | Correct Decision (probability = $1 - \beta$) |



Design parameters and error types

- Significance level α
 - α = Type I error = False positive
 - Commonly set at 5%
- P-value
 - Probability of obtaining a result assuming the Null hypothesis is true.
 - If p-value < α
 - Reject Null hypothesis
 - If p-value > α
 - Fail to reject Null hypothesis

| | | Truth about our users |
|------------------------|--|--|
| Decision based on data | H_0 is true | H_0 is false |
| Fail to reject H_0 | Correct Decision (probability = $1 - \alpha$) | Type II Error - False Negative (probability = β) |
| Reject H_0 | Type I Error - False Positive (probability = α) | Correct Decision (probability = $1 - \beta$) |



Experiment parameters analogy

Analogy for explaining statistical power and parameters:

1. Time at store = sample size/experiment duration
2. Bag of chips size = effect size/MDE
3. Store cleanliness/organization = data variance

Low power



High power

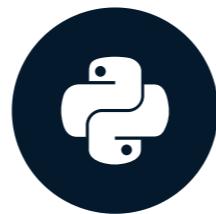


Let's practice!

A/B TESTING IN PYTHON

Experimental design: power analysis

A/B TESTING IN PYTHON



Moe Lotfy, PhD

Principal Data Science Manager

Effect size

- Cohen's d for differences in means

$$\text{Cohen's d} = \frac{\mu_B - \mu_A}{SD_{\text{pooled}}}$$

- Cohen's h for differences in proportions

$$\text{Cohen's h} = 2 \arcsin \sqrt{p_1} - 2 \arcsin \sqrt{p_2}$$

- Rule of thumb

- Small effect = 0.2
- Medium effect = 0.5
- Large effect = 0.8

```
# Calculate standardized effect size
from statsmodels.stats.proportion import proportion_effectsize
effect_size_std = proportion_effectsize(.33, .3)
print(effect_size_std)
```

0.0645

```
# Calculate standardized effect size
from statsmodels.stats.proportion import proportion_effectsize
effect_size_std = proportion_effectsize(p_B, p_A)
print(effect_size_std)
```

0.0716

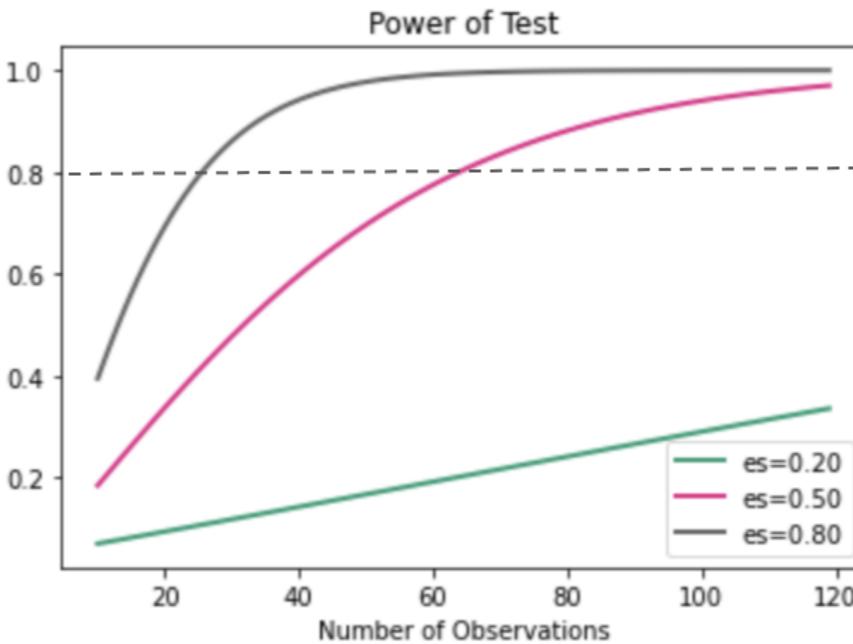
Sample size estimation for proportions

```
# Import power module
from statsmodels.stats import power
# Calculate sample size
sample_size = power.TTestIndPower().solve_power(effect_size=effect_size_std,
                                                power=.80,
                                                alpha=.05,
                                                nobs1=None)
print(sample_size)
```

3057.547

Effect of sample size and MDE on power

```
# Import t-test power package
from statsmodels.stats.power import TTestIndPower
# Specify parameters for power analysis
sample_sizes = array(range(10, 120))
effect_sizes = array([0.2, 0.5, 0.8])
# Plot power curves
TTestIndPower().plot_power(nobs=sample_sizes, effect_size=effect_sizes)
plt.show()
```



Sample size estimation for means

```
# Calculate the baseline mean order value  
mean_A = checkout[checkout['checkout_page']=='A']['order_value'].mean()  
print(mean_A)
```

24.9564

```
std_A = checkout[checkout['checkout_page']=='A']['order_value'].std()  
print(std_A)
```

2.418

```
# Specify the desired minimum average order value  
mean_new = 26
```

```
# Calculate the standardized effect size  
std_effect_size=(mean_new-mean_A)/std_A
```

Sample size estimation for means

```
sample_size = power.TTestIndPower().solve_power(effect_size=std_effect_size,  
                                              power=.80,  
                                              alpha=.05,  
                                              nobs1=None)  
  
print(sample_size)
```

85.306

Let's practice!

A/B TESTING IN PYTHON

Multiple comparisons tests

A/B TESTING IN PYTHON

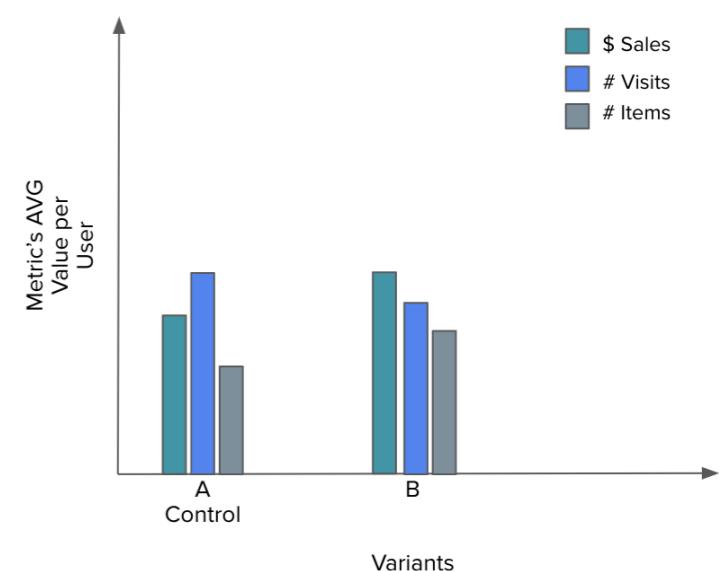
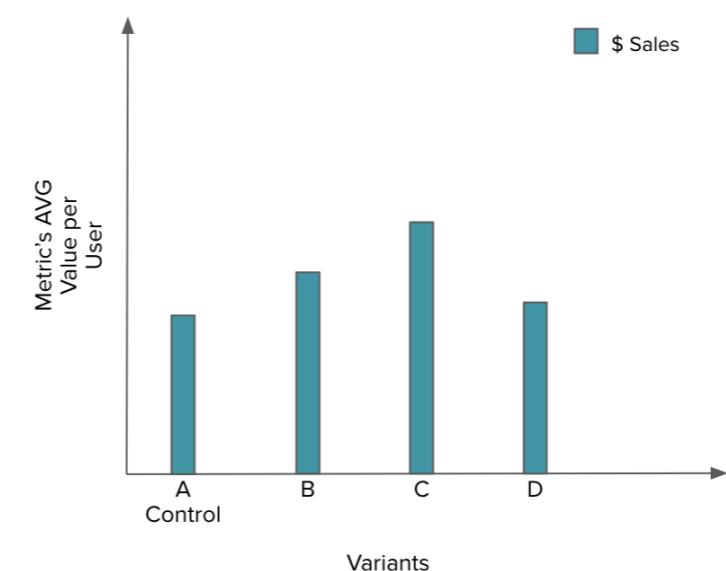
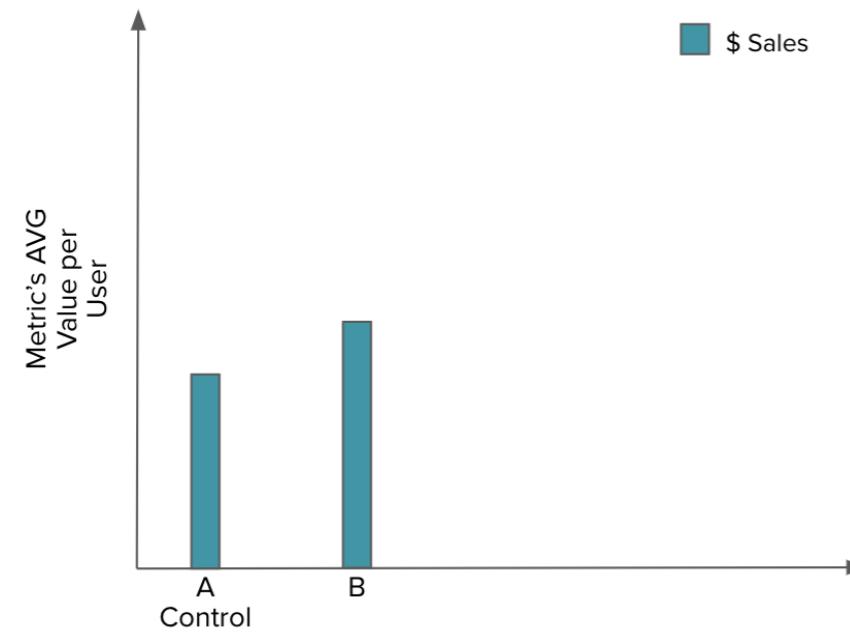


Moe Lotfy, PhD

Principal Data Science Manager

Introduction to the multiple comparisons problem

- Single comparison:
 - Control (A) versus Treatment (B)
 - One metric
 - No subcategories
- Multiple comparisons:
 - Multiple variants (A/B/n tests)
 - Multiple metrics
 - Granular categories



Family-wise error rate

- $P(\text{making Type I error}) = \alpha = 0.05$
- $P(\text{not making Type I error}) = 1 - \alpha$
- $P(\text{not making Type I error in } m \text{ tests}) = (1 - \alpha)^m$
- $P(\text{making at least one Type I error in } m \text{ tests}) = 1 - (1 - \alpha)^m = \text{FWER}$

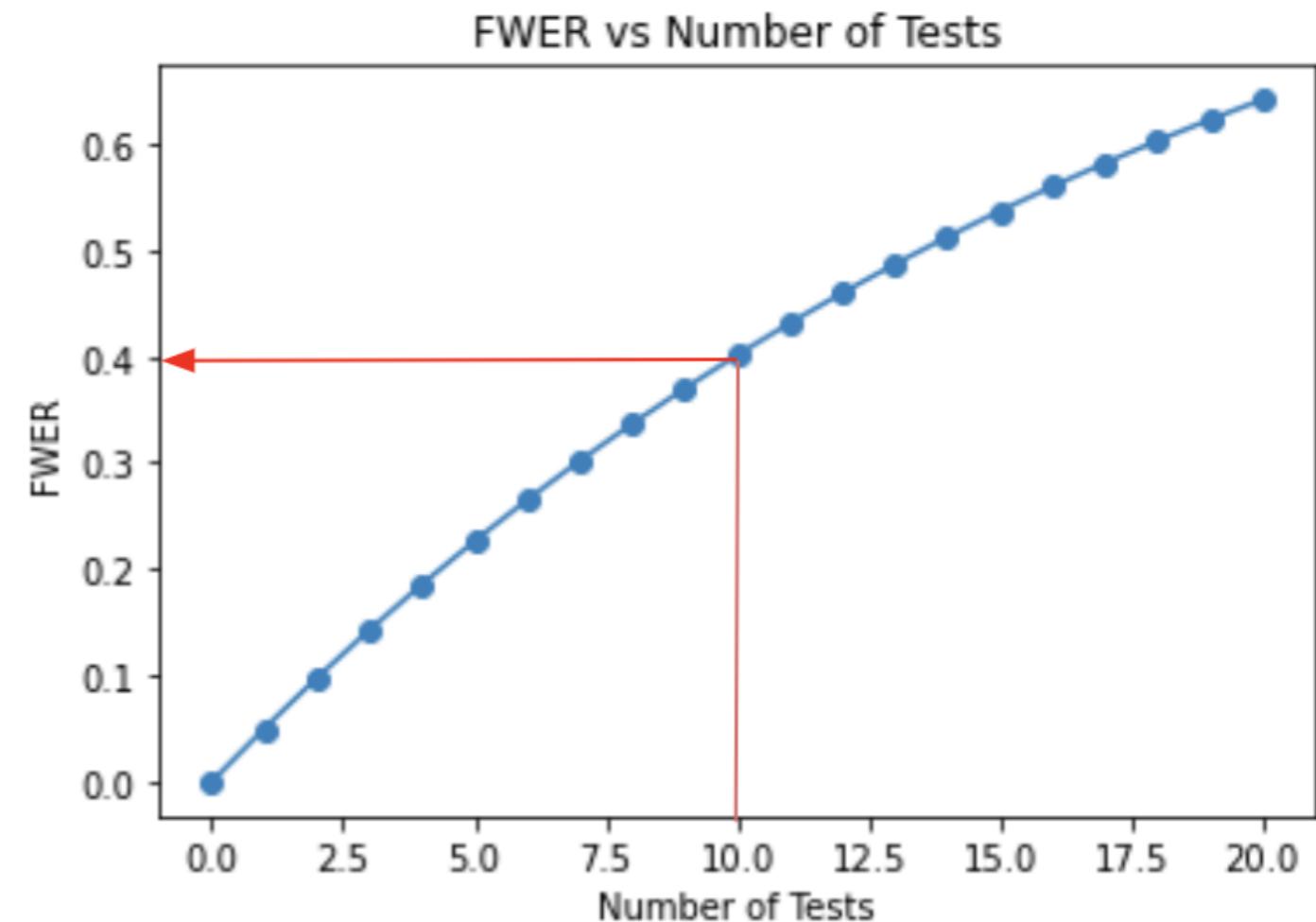
Family-wise Error Rate (FWER): the probability of making one or more type I errors when performing multiple hypothesis tests.

- For a single test, $\text{FWER} = 1 - (1 - \alpha)^1 = \alpha = 0.05$
- **But what if we perform more than one test?**

Family-wise error rate

```
import matplotlib.pyplot as plt  
import numpy as np  
  
alpha = 0.05  
  
x = np.linspace(0, 20, 21)  
y = 1-(1-alpha)**x  
  
plt.plot(x,y, marker='o')  
plt.title('FWER vs Number of Tests')  
plt.xlabel('Number of Tests')  
plt.ylabel('FWER')  
plt.show()
```

- $\text{FWER} = 1 - (1 - \alpha)^{10}$
- FWER for 10 tests = 40%



Correction methods

- The simplest and most popular approach is the Bonferroni Correction
- Set the adjusted α^* to the individual test α divided by the number of tests m

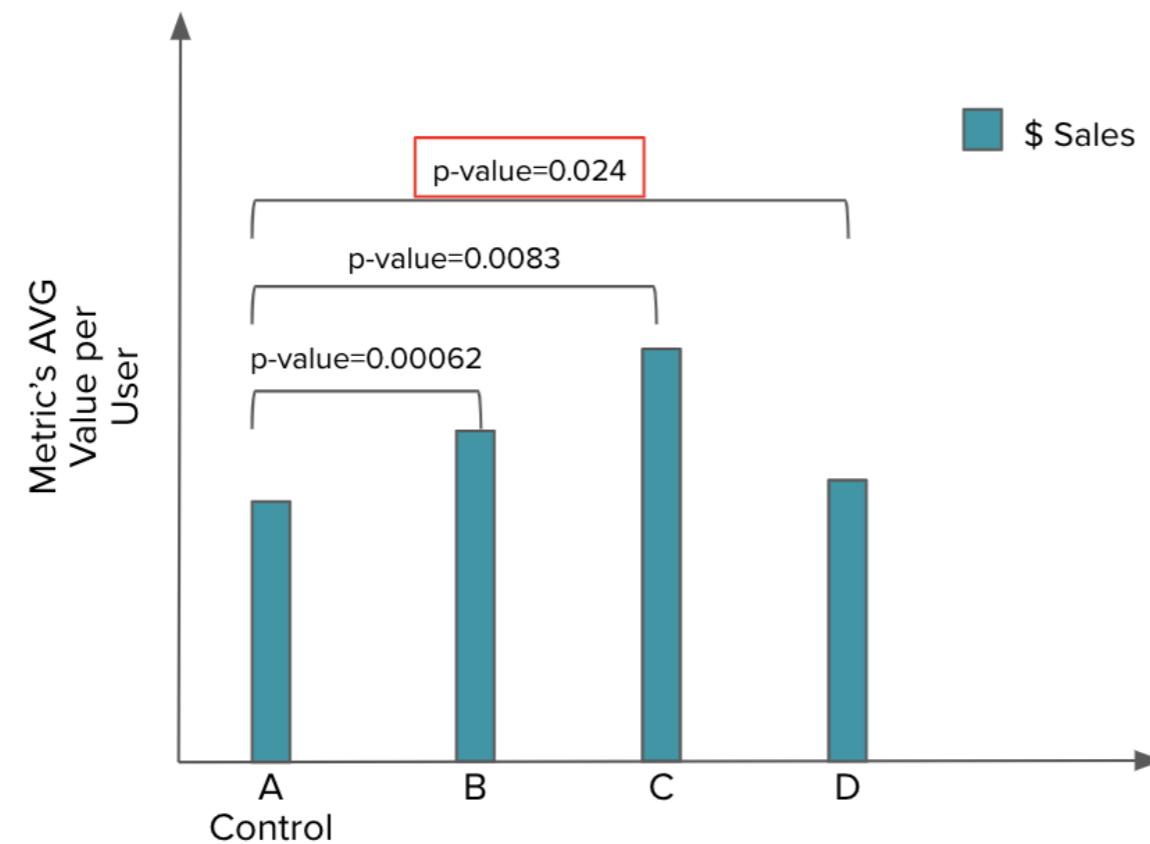
$$\text{Bonferroni } \alpha^* = \frac{\alpha}{m}$$

- Less stringent Sidak correction
- Set FWER to desired α , then solve for α_s

$$\alpha_s = 1 - (1 - \alpha)^{1/m}$$

Bonferroni correction example

- Without correction, all three tests are considered significant
 - but the probability of making a type I error is inflated at 14%
- With a Bonferroni Correction, A versus D is no longer significant, but FWER is controlled at 0.049



$$FWER = 1 - (1 - 0.05)^3 = 0.143$$

$$FWER = 1 - \left(1 - \frac{0.05}{3}\right)^3 = 0.049$$

$$\text{Bonferroni } \alpha^* = \frac{\alpha}{m} = \frac{0.05}{3} = 0.0167$$

statsmodels multipletests method

```
import statsmodels.stats.multitest as smt  
pvals = [0.023, 0.0005, 0.0004]
```

```
corrected = smt.multipletests(pvals, alpha=0.05, method='bonferroni')
```

```
print("Significant Test:", corrected[0])  
print("Corrected P-values:", corrected[1])  
print("Bonferroni Corrected alpha: {:.4f}".format(corrected[3]))
```

```
Significant Test: [False  True  True]  
Corrected P-values: [0.069  0.0015  0.00012]  
Bonferroni Corrected alpha: 0.0167
```

Let's practice!

A/B TESTING IN PYTHON