A/B Testing at Streaming Service

Zaheer Abbas

2025-05-25

Randomization check.

overall randomization

```
import pandas as pd
import numpy as np
from scipy.stats import ttest_ind
#load the data set
ds=pd.read_csv('https://raw.githubusercontent.com/MeZaheer89/dAIDATASET/refs/heads/main/data%20set%20
print(ds.head(7), ds.columns)

#check randomization on treatment column(new_algo) and value column(prior_minutes)

def overall_randomization_check(ds):
    group_0 = ds[ds['new_algo'] == 0]['prior_minutes']
    group_1 = ds[ds['new_algo'] == 1]['prior_minutes']

    # Mean and standard deviation for both groups
    mean_0, std_0 = group_0.mean(), group_0.std()
    mean_1, std_1 = group_1.mean(), group_1.std()

    # T-test
    t_stat, p_val = ttest_ind(group_0, group_1, equal_var=False)

    # Print inside the function
    print(f"\n Control Group : Mean = {mean_0:.2f}, Std = {std_0:.2f}, n = {len(group_0)}")
    print(f"Treatment Group: Mean = {mean_1:.2f}, Std = {std_1:.2f}, n = {len(group_1)}")
    print(f"T-statistic = {t_stat:.3f}, P-value = {p_val:.3f}")
overall_randomization_check(ds)
```

```
   user_id  log_in_day  prior_minutes  new_algo     minutes  churn_prob  churn
0        1           3     423.781195         1  446.000977        0.08      1
1        2           3     253.081259         1  220.090242        0.08      0
```

```
2        3           3     183.337618         1  182.107001        0.08      0
3        4           2     472.066415         1  583.073762        0.08      0
4        5           3      60.731451         0   49.254167        0.10      0
5        6           2     462.944878         1  440.215377        0.08      0
6        7           2     602.477479         0  594.412191        0.10      0 Index(['user_id', 'log
       'churn_prob', 'churn'],
      dtype='object')

 Control Group : Mean = 398.88, Std = 215.53, n = 5037
Treatment Group: Mean = 399.63, Std = 216.09, n = 4963
T-statistic = -0.172, P-value = 0.863
```

Conculsion: Randomization appears succssful overall

Random Check day by day

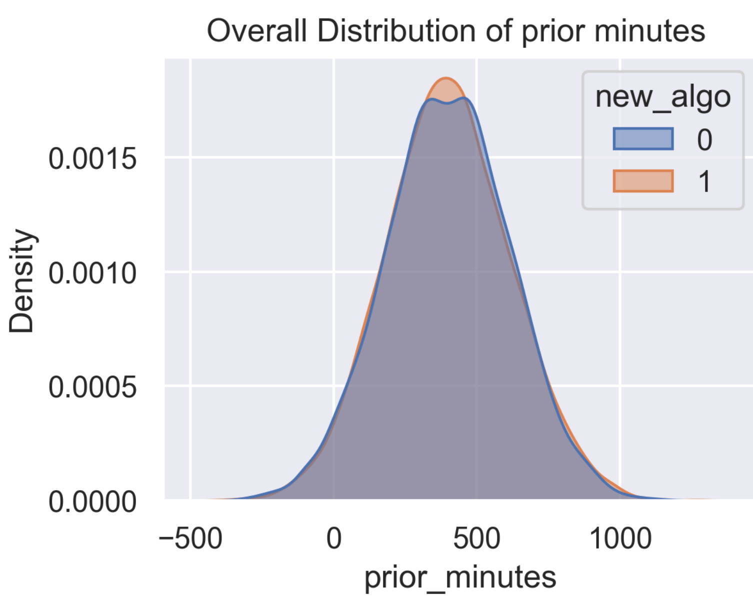
```python
def check_day_by_day(ds, day):
     print(f"\n Day {day}")
     day_data = ds[ds['log_in_day'] == day]

     group_0 = day_data[day_data['new_algo'] == 0]['prior_minutes']
     group_1 = day_data[day_data['new_algo'] == 1]['prior_minutes']

     mean_0, std_0 = group_0.mean(), group_0.std()
     mean_1, std_1 = group_1.mean(), group_1.std()
     t_stat, p_val = ttest_ind(group_0, group_1, equal_var=False)
     print(f"Control Group: Mean = {mean_0:.2f}, Std = {std_0:.2f}, n = {len(group_0)}")
     print(f"Treatment Group: Mean = {mean_1:.2f}, Std = {std_1:.2f}, n = {len(group_1)}")
     print(f"T-statistic = {t_stat:.4f}, P-value = {p_val:.3f}")
for day in [1, 2, 3]:
    check_day_by_day(ds, day)
```

```
 Day 1
Control Group: Mean = 490.68, Std = 204.01, n = 1661
Treatment Group: Mean = 505.21, Std = 203.12, n = 1674
T-statistic = -2.0609, P-value = 0.039

 Day 2
Control Group: Mean = 403.64, Std = 199.77, n = 1677
Treatment Group: Mean = 395.36, Std = 194.22, n = 1637
T-statistic = 1.2100, P-value = 0.226

 Day 3
Control Group: Mean = 304.45, Std = 201.25, n = 1699
Treatment Group: Mean = 296.86, Std = 198.16, n = 1652
T-statistic = 1.0988, P-value = 0.272
```

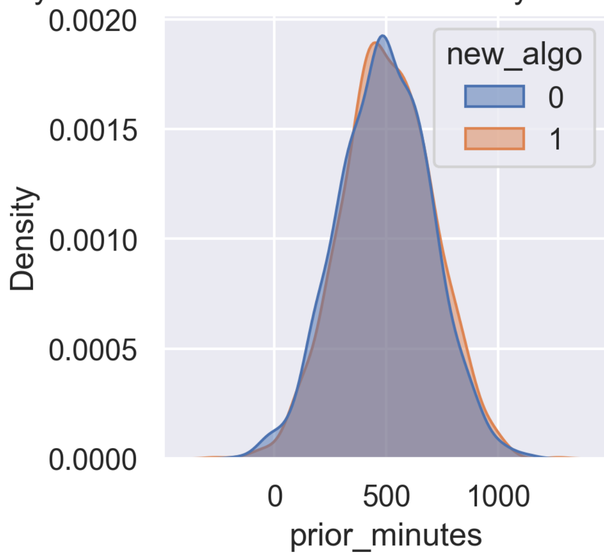
Conculsion: Day 1 shows a small statistically significant but day 2 and 3 shows no significant differences

Visualization

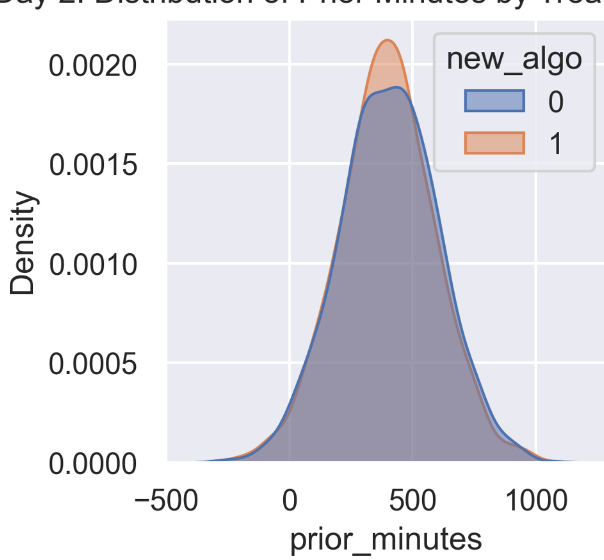
```
import seaborn as sb
import matplotlib.pyplot as plt
sb.set(style='darkgrid')
plt.figure(figsize=(4,3))
sb.kdeplot(data=ds, x='prior_minutes', hue='new_algo', fill=True, common_norm=False, alpha=0.5)
plt.title('Overall Distribution of prior minutes ')
plt.show

# for day by day
sb.set(style='darkgrid')
for day in [1,2,3]:
    plt.figure(figsize=(3,3))
    sb.kdeplot(data=ds[ds["log_in_day"] == day],
        x="prior_minutes",
        hue="new_algo",
        fill=True,
        common_norm=False,
        alpha=0.5
    )
    plt.title(f"Day {day}: Distribution of Prior Minutes by Treatment Group")
    plt.show()
```



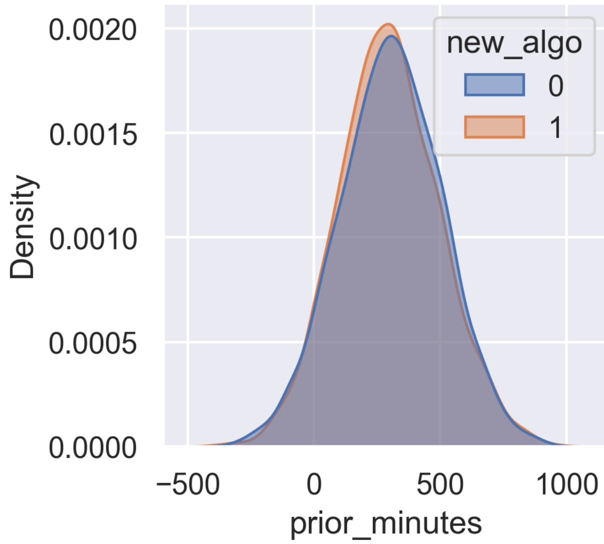
Day 1: Distribution of Prior Minutes by Treatment Group



Day 2: Distribution of Prior Minutes by Treatment Group



Day 3: Distribution of Prior Minutes by Treatment Group



Power Analysis

```
from scipy.stats import t, norm
# Sample sizes per group
grp_counts = ds.groupby('new_algo').size()
n = grp_counts.min()

minutes_sd = ds.groupby('new_algo')['minutes'].std().mean()

p = ds[ds['new_algo'] == 0]['churn'].mean()

# Given Parameters
alpha = 0.05
power = 0.80
beta = 1 - power
df_t = 2 * n - 2

# Critical values
t_alpha = t.ppf(1 - alpha / 2, df_t)
t_beta = t.ppf(power, df_t)

z_alpha = norm.ppf(1 - alpha / 2)
z_beta = norm.ppf(power)

mde_minutes = (t_alpha + t_beta) * np.sqrt(2 * minutes_sd**2 / n) #Compute MDE for Minutes (T-test)
```

```python
print(f"MDE for 'minutes': {mde_minutes:.2f} minutes")

mde_churn = (z_alpha + z_beta) * np.sqrt(2 * p * (1 - p) / n) #Compute MDE for Churn (Proportion z-te
print(f"MDE for 'churn': {mde_churn:.4f} or {mde_churn*100:.2f}%")
```

```
MDE for 'minutes': 12.57 minutes
MDE for 'churn': 0.0170 or 1.70%
```

Computation of Required sample size

```python
import math
Alpha= 0.05
power= 0.80
mde_min = 10
mde_churn = 0.01
z_alpha = 1.96 # This value is given in slide when alpha is 0.05
z_beta = 0.84 # Z for power = 0.80
sigma_min = ds['minutes'].std()
p=p = ds[ds['new_algo'] == 0]['churn'].mean() # # churn rate in control group


# Sample size formula for minutes
n_minutes = ((z_alpha + z_beta)**2 * 2 * (sigma_min**2)) / (mde_min**2) #formula for sample size
n_minutes = round(n_minutes)

# Sample size formula for churn
n_churn = (2 * (z_alpha + z_beta)**2 * p * (1 - p)) / (mde_churn ** 2)
n_churn = round(n_churn)
print(f"Required sample size per group for minutes:{n_minutes}")
print(f"Required sample size per group for churn: {n_churn}")
```

```
Required sample size per group for minutes:7844
Required sample size per group for churn: 14393
```

```python
### Now Check: Is Your Study Well Powered?
group_sizes = ds['new_algo'].value_counts()
print(group_sizes)
```

```
new_algo
0    5037
1    4963
Name: count, dtype: int64
```

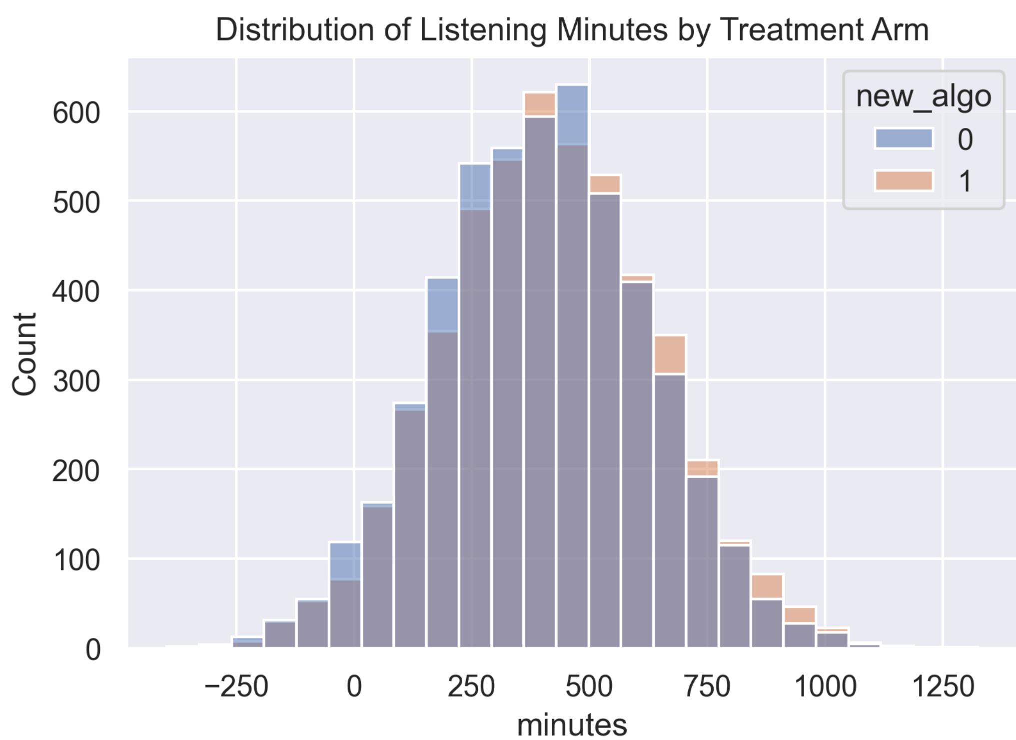
The study is underpowered for both metrics, particularly churn. As a result, any non-significant findings should be interpreted with caution, as the study may lack the sensitivity to detect meaningful effects

Group Mean Comparisions

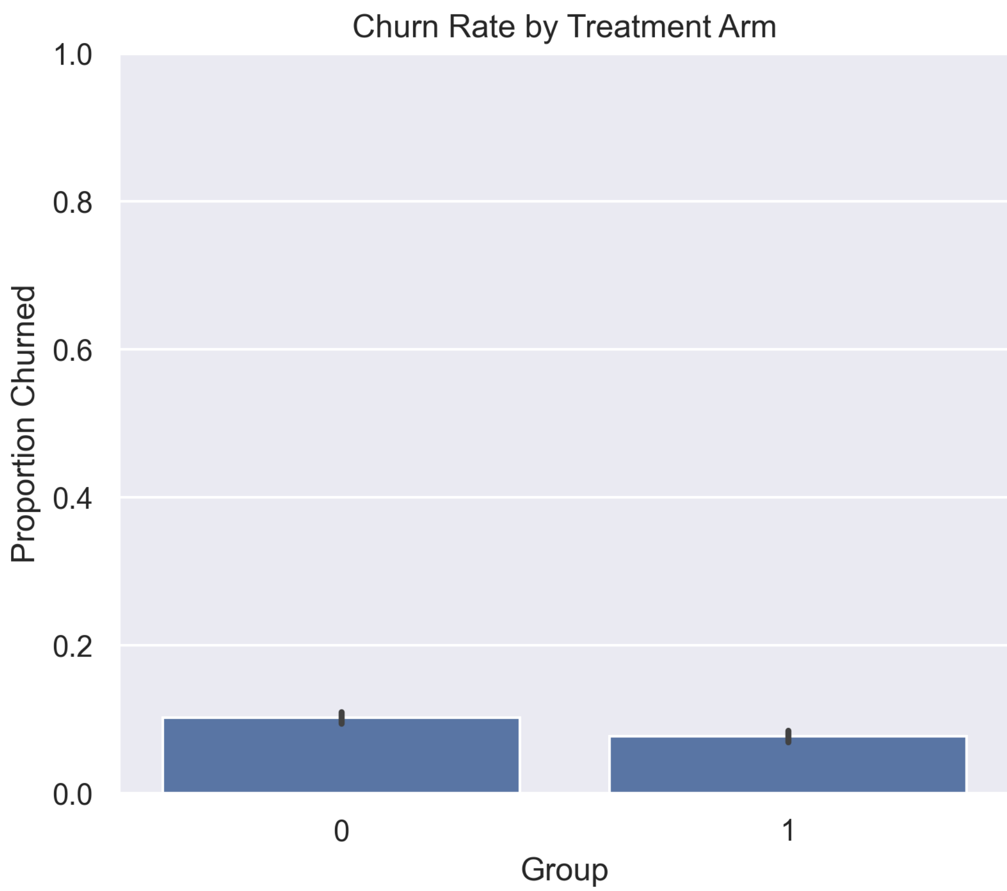
```
from scipy.stats import ttest_ind, norm

# Separate control and treatment groups
control = ds[ds['new_algo'] == 0]
treatment = ds[ds['new_algo'] == 1]


t_stat, p_val = ttest_ind(treatment['minutes'], control['minutes'], equal_var=True)

print("T-Test: Minutes")
print(f"  Mean (Control):   {control['minutes'].mean():.2f}")
print(f"  Mean (Treatment): {treatment['minutes'].mean():.2f}")
print(f"  t-statistic: {t_stat:.2f}")
print(f"  p-value:     {p_val:.4f}")

# For Churn
n1 = len(control)
n2 = len(treatment)
p1 = control['churn'].mean()
p2 = treatment['churn'].mean()

# Pooled proportion
p_pool = (p1 * n1 + p2 * n2) / (n1 + n2)
# Standard error
se = np.sqrt(p_pool * (1 - p_pool) * (1/n1 + 1/n2))
# Z-score
z = (p1 - p2) / se
p_z = 2 * (1 - norm.cdf(abs(z)))  # two-tailed

print("\n Z-Test: Churn")
print(f"  Churn for (Control):   {p1:.3f}")
print(f"  Churn for (Treatment): {p2:.3f}")
print(f"  z-score: {z:.2f}")
print(f"  p-value: {p_z:.4f}")

#visualization
plt.figure(figsize=(6,4))
sb.histplot(data=ds, x='minutes', hue='new_algo', stat="count", bins= 25)
plt.title("Distribution of Listening Minutes by Treatment Arm")
plt.show()

plt.figure(figsize=(6,5))
sb.barplot(data=ds, x='new_algo', y='churn', ci=95)
plt.title("Churn Rate by Treatment Arm")
plt.xlabel("Group")
```

```python
plt.ylabel("Proportion Churned")
plt.ylim(0, 1)
plt.show()
```

```
T-Test: Minutes
  Mean (Control):   399.37
  Mean (Treatment): 418.46
  t-statistic: 4.27
  p-value:     0.0000

 Z-Test: Churn
  Churn for (Control):   0.102
  Churn for (Treatment): 0.077
  z-score: 4.35
  p-value: 0.0000
```




```
C:\Users\HP\AppData\Local\Temp\ipykernel_20768\3494059377.py:43: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', 95)` for the same effect.

  sb.barplot(data=ds, x='new_algo', y='churn', ci=95)
```




```
from scipy import stats
mean_c = control['minutes'].mean()
mean_t = treatment['minutes'].mean()
std_c = control['minutes'].std(ddof=1)
std_t = treatment['minutes'].std(ddof=1)
se_minutes = np.sqrt((std_c**2 / n1) + (std_t**2 / n2))
effect_minutes = mean_t - mean_c # mean of treatmen minus mean of control

# Z-values
z_95 = stats.norm.ppf(0.975)
z_99 = stats.norm.ppf(0.995)

ci_95min = (effect_minutes - z_95 * se_minutes, effect_minutes + z_95 * se_minutes)
ci_99min = (effect_minutes - z_99 * se_minutes, effect_minutes + z_99 * se_minutes)

# For churn
se_churn = np.sqrt((p1 * (1 - p1)) / n1 + (p2 * (1 - p2)) / n2)
effect_churn = p2 - p1
```

```
ci_95churn = (effect_churn - z_95 * se_churn, effect_churn + z_95 * se_churn)
ci_99churn = (effect_churn - z_99 * se_churn, effect_churn + z_99 * se_churn)

print("\n=== Confidence Intervals for Treatment Effect ===")
print("Minutes:")
print(f"  Effect: {effect_minutes:.2f}")
print(f"  95% CI: {ci_95min}")
print(f"  99% CI: {ci_99min:}")
print("Churn:")
print(f"  Effect: {effect_churn:.4f}")
print(f"  95% CI: {ci_95churn:}")
print(f"  99% CI: {ci_99churn:}")
```

```
=== Confidence Intervals for Treatment Effect ===
Minutes:
  Effect: 19.08
  95% CI: (np.float64(10.323861498744925), np.float64(27.846009681077923))
  99% CI: (np.float64(7.570932499754878), np.float64(30.59893868006797))
Churn:
  Effect: -0.0249
  95% CI: (np.float64(-0.03606268837870189), np.float64(-0.013678995475906001))
  99% CI: (np.float64(-0.0395794214102559), np.float64(-0.010162262444351991))
```

Hypothesis test

Beneficial only if user listen more than 5%

```
from scipy.stats import norm
diff = mean_t - mean_c
threshold = 0.05 * mean_c

z = (diff - threshold) /se_minutes
p_value = 1 - norm.cdf(z)

print(f"p-value for increase   5%: {p_value:.4f}")
```

```
p-value for increase   5%: 0.5784
```

Result: Based on the hypothesis test, there is insufficient evidence to conclude that the new algorithm increases user listening time by at least 5% so considering the associated costs, new algorithm may not be beneficial.

Regression Analysis

```
import statsmodels.api as sp
import statsmodels.formula.api as sfp

#linear regression for minutes
model_minutes=sfp.ols('minutes~new_algo', data=ds).fit()
ate_minutes = model_minutes.params['new_algo']

#logidtic regressio for churn
model_churn=sfp.logit('churn ~ new_algo', data=ds).fit()
marginal_effects = model_churn.get_margeff()  # Average Marginal Effect (AME)
ate_churn = marginal_effects.margeff[0]
print(f"ATE of new_algo on minutes: {ate_minutes:.4f}" )
print(f"ATE of new_algo on churn: {ate_churn:.4f}" )

print("\n  linear regression (Minutes) ")
print(model_minutes.summary().tables[1])

r2_linear = model_minutes.rsquared
print(f"R-squared (linear regression): {r2_linear:.3f}")

print("\n  logistic regression (churn) ")
print(model_churn.summary().tables[1])

r2_logit = model_churn.prsquared
print(f" R-squared (logistic regression): {r2_logit:.3f}")
```

```
Optimization terminated successfully.
         Current function value: 0.301358
         Iterations 6
ATE of new_algo on minutes: 19.0849
ATE of new_algo on churn: -0.0250

  linear regression (Minutes)
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    399.3744      3.149    126.838      0.000     393.202     405.546
new_algo      19.0849      4.470      4.270      0.000      10.324      27.846
==============================================================================
R-squared (linear regression): 0.002

  logistic regression (churn)
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
Intercept     -2.1725      0.047    -46.714      0.000      -2.264      -2.081
new_algo      -0.3061      0.071     -4.335      0.000      -0.444      -0.168
==============================================================================
 R-squared (logistic regression): 0.003
```

interpretation

Linear regression on minutes

Effect: The new algorithm increases user engagement by +19.08 minutes
P-values: As p<0.0001 the ffect is statistically significant
Confidence: We are 95% confident the true effect lies between +10.32 and +27.85 minutes.
R_squared: The algorithm explains only 0.2% of the variation in minutes.

logistic regression on Churn

Effect: The new algorithm reduces log-odds of churn by 0.306
P-values: As p<0.0001 the ffect is statistically significant
Confidence: We're 95% confident the true log-odds reduction is between -0.444 and -0.168.
R_squared: The algorithm explains only 0.3% of the variation in churn

```
print(marginal_effects.summary()) #Average Marginal Effect
```

```
        Logit Marginal Effects
=====================================
Dep. Variable:                  churn
Method:                          dydx
At:                           overall
==============================================================================
                dy/dx    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
new_algo      -0.0250      0.006     -4.318      0.000      -0.036      -0.014
==============================================================================
```

On average, the new algorithm reduces the probability of churn by 2.5 percentage points.

Variance reduction

Addition of Covariates in linear and logistic regression on minutes nad churn respectively.

```
minutes_model_cov = sfp.ols('minutes ~ new_algo + log_in_day + prior_minutes', data=ds).fit() # covar
churn_model_cov = sfp.logit('churn ~ new_algo+ log_in_day + prior_minutes', data=ds).fit()

print("\n  linear regression (Minutes) ")
print(minutes_model_cov.summary().tables[1])
```

```python
print("\n  logistic regression (churn) ")
print(churn_model_cov.summary().tables[1])
```

```
Optimization terminated successfully.
         Current function value: 0.301277
         Iterations 6

  linear regression (Minutes)
=================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
Intercept       -4.3267      2.048     -2.112      0.035      -8.342      -0.311
new_algo        18.3403      0.996     18.406      0.000      16.387      20.293
log_in_day       0.4088      0.657      0.622      0.534      -0.879       1.696
prior_minutes    1.0100      0.002    405.782      0.000       1.005       1.015
=================================================================================

  logistic regression (churn)
=================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
Intercept       -2.1262      0.142    -14.932      0.000      -2.405      -1.847
new_algo        -0.3058      0.071     -4.331      0.000      -0.444      -0.167
log_in_day       0.0129      0.046      0.280      0.780      -0.077       0.103
prior_minutes   -0.0002      0.000     -1.048      0.295      -0.001       0.000
=================================================================================
```

Compare standard error and estimates with and without covariates

```python
# Linear Regression Comparison
linear_comp = pd.DataFrame({'Metric': ['Coefficient', 'Std Error'], 'Without_Cov': [
        model_minutes.params['new_algo'],
        model_minutes.bse['new_algo'] ], 'With_cov': [
        minutes_model_cov.params['new_algo'],
        minutes_model_cov.bse['new_algo'] ], 'Diff': [
        model_minutes.params['new_algo'] - minutes_model_cov.params['new_algo'],
        model_minutes.bse['new_algo'] - minutes_model_cov.bse['new_algo']
    ]
})

# Logistic Regression Comparison
logit_comp = pd.DataFrame({'Metric': ['Coefficient', 'Std Error'], 'Without_Cov': [
        model_churn.params['new_algo'],
        model_churn.bse['new_algo'] ], 'With_cov': [
        churn_model_cov.params['new_algo'],
        churn_model_cov.bse['new_algo'] ], 'Diff': [
```

```
        model_churn.params['new_algo'] - churn_model_cov.params['new_algo'],
        model_churn.bse['new_algo'] - churn_model_cov.bse['new_algo'] ]
})

print("\nLinear Regression Comparison")
print(linear_comp)

print("\nLogistic Regression Comparison")
print(logit_comp)
```

```
Linear Regression Comparison
        Metric  Without_Cov   With_cov      Diff
0  Coefficient    19.084936  18.340275  0.744661
1    Std Error     4.469510   0.996431  3.473078

Logistic Regression Comparison
        Metric  Without_Cov  With_cov      Diff
0  Coefficient    -0.306051 -0.305828 -0.000223
1    Std Error     0.070608  0.070615 -0.000007
```

In linear regression, adding covariates slightly reduced the coefficient and greatly improved precision
In logistic regression, both the coefficient and standard error remained nearly unchanged, indicating covariates had minimal impact.

CUPED

Wew will use this formula for CUPED . Y_cuped=Y−θ(X− X_BAR)

```
X = ds['prior_minutes']
Y = ds['minutes']

# Calculate theta
theta = np.cov(Y, X, bias=True)[0][1] / np.var(X)

# Create CUPED-adjusted outcome

ds['minutes_cuped'] = ds['minutes'] - theta * (ds['prior_minutes'] - ds['prior_minutes'].mean())

# Fit model with CUPED-adjusted outcome
model_cuped = sfp.ols('minutes_cuped ~ new_algo', data=ds).fit()

print("\n  WITHOUT CUPED ON MINUTES ")
print(model_minutes.summary().tables[1])
print("\nWith CUPED ON MINUTES")
print(model_cuped.summary().tables[1])
```

```
  WITHOUT CUPED ON MINUTES
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    399.3744      3.149    126.838      0.000     393.202     405.546
new_algo      19.0849      4.470      4.270      0.000      10.324      27.846
==============================================================================

With CUPED ON MINUTES
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    399.7462      0.702    569.524      0.000     398.370     401.122
new_algo      18.3358      0.996     18.403      0.000      16.383      20.289
==============================================================================
```

The coefficient slightly decreased and the standard error dropped significantly (to 0.996), indicating a more precise estimate for minutes column.

Conditional Average Treatment Effects

conditional ATE of *log_in_day* for minutes and churn by linear and logistic respectively

```python
# As i already loaded all required libraries so I'll simply call here
for day in sorted(ds['log_in_day'].unique()):
    subset = ds[ds['log_in_day'] == day]

    # 1. Linear model for minutes
    lin_model = sfp.ols('minutes ~ new_algo', data=subset).fit()
    ate_minutes = lin_model.params['new_algo']

    # 2. Logistic model for churn
    log_model = sfp.logit('churn ~ new_algo', data=subset).fit(disp=0)
    ate_churn = np.exp(log_model.params['new_algo'])  # Convert to odds ratio

    print(f"\nResults for log_in_day = {day} (n={len(subset)})")
    print(f"ATE on minutes: {ate_minutes:.2f}")
    print(f"Odds Ratio for churn: {ate_churn:.2f}")
### conditional ATE of *prior_minutes* for minutes and churn by linear and logistic respectively
```

```
Results for log_in_day = 1 (n=3335)
ATE on minutes: 34.39
Odds Ratio for churn: 0.81
```

```
Results for log_in_day = 2 (n=3314)
ATE on minutes: 10.22
Odds Ratio for churn: 0.74

Results for log_in_day = 3 (n=3351)
ATE on minutes: 9.07
Odds Ratio for churn: 0.67
```

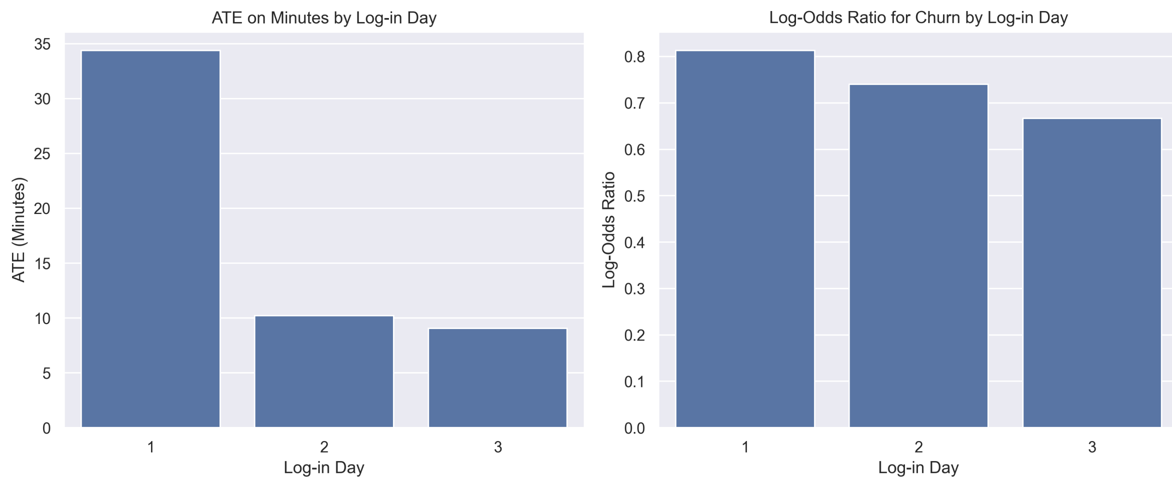
```
linear_model = sfp.ols('minutes ~ new_algo * prior_minutes', data=ds).fit()
logit_model = sfp.logit('churn ~ new_algo * prior_minutes', data=ds).fit(disp=0)

# 2. it is good choice to calculate CATEs at median so,
median_prior = ds['prior_minutes'].median()

# For minutes (linear)
linear_ate = (linear_model.params['new_algo'] + linear_model.params['new_algo:prior_minutes'] * media

# For churn (logistic - odds ratio)
logit_odds = np.exp(logit_model.params['new_algo'] +  logit_model.params['new_algo:prior_minutes'] * 

print(f"Conditional ATE at median prior_minutes ({median_prior:.1f}):")
print(f" Minutes: {linear_ate:.2f} (treatment effect in minutes)")
print(f"Churn: OR = {logit_odds:.2f} (odds ratio)")
```

```
Conditional ATE at median prior_minutes (397.7):
 Minutes: 18.32 (treatment effect in minutes)
Churn: OR = 0.74 (odds ratio)
```

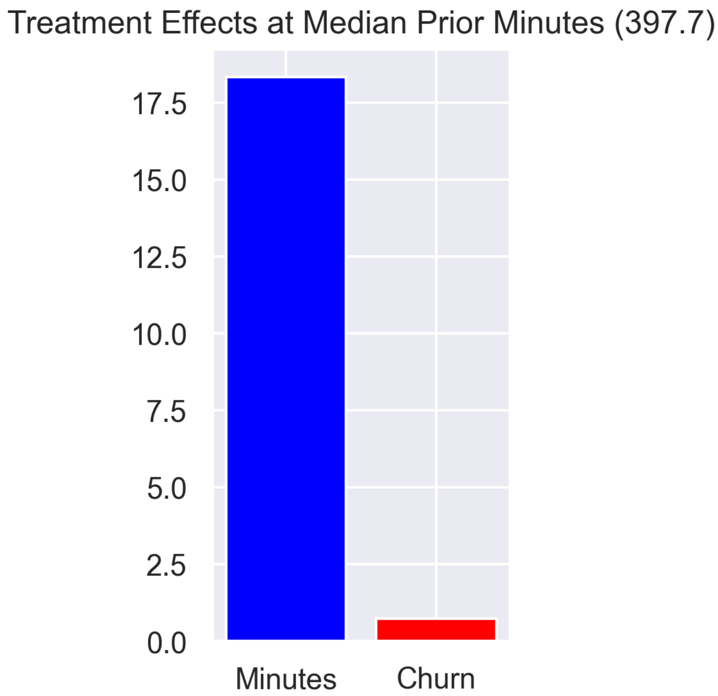
Visualization

```
# Using previously calculated CATEs values for plotting
log_in_days_m = [1, 2, 3]
ate_minutes_m = [34.386, 10.217, 9.069]   #As already calculated above
log_odds_churn_m = [0.813, 0.740, 0.667]   #As already calculated above'
plt.figure(figsize=(12, 5))

# Plot ATE on minutes
plt.subplot(1, 2, 1)
sb.barplot(x=log_in_days_m, y=ate_minutes_m)
plt.title('ATE on Minutes by Log-in Day')
plt.xlabel('Log-in Day')
plt.ylabel('ATE (Minutes)')
#plt.ylim(0, max(ate_minutes) + 5)  # set y-axis to show differences clearly

# Plot log-odds on churn
```

```python
plt.subplot(1, 2, 2)
sb.barplot(x=log_in_days_m, y=log_odds_churn_m)
plt.title('Log-Odds Ratio for Churn by Log-in Day')
plt.xlabel('Log-in Day')
plt.ylabel('Log-Odds Ratio')
#plt.ylim(0, max(log_odds_churn) + 0.1)

plt.tight_layout()
plt.show()
```




```python
median_prior = 397.7    # already calculated median values
linear_ate = 18.32      # Already calculated values
logit_odds = 0.74       # Already calculated values

plt.figure(figsize=(2, 4))

# Plot bars
plt.bar(['Minutes', 'Churn'], [linear_ate, logit_odds], color=['blue', 'red'])

plt.title(f'Treatment Effects at Median Prior Minutes ({median_prior:.1f})')

plt.show()
```



Interpretation

The treatment for users *logging in* just once, the treatment boosts usage by about 34.39 minutes. This effect is smaller for users logging in twice or three times, with increases of 10.22 and 9.07 minutes respectively.

In terms of churn, the treatment reduces the likelihood of users leaving across all groups.Login frequency increases: 0.81 for one login day, 0.74 for two, and 0.67 for three. This indicates the treatment's effect on reducing churn strengthens with more frequent users.

The estimated ATE on *minutes* was about 18.32 minutes, meaning the treatment increased users engagement with 18 minutes on average. This shows a meaningful positive impact on user activity.

For churn, the odds ratio was estimated to be 0.74, indicating that the treatment decreased the odds of users churning by 26%.

Bayesian A/B-Testing

Non-informative priors

```
import pymc as pm
import arviz as az
```

```
# Extracting data
minutes_old = ds[ds["new_algo"] == 0]["minutes"].values
minutes_new = ds[ds["new_algo"] == 1]["minutes"].values
churn_old = ds[ds["new_algo"] == 0]["churn"].values
churn_new = ds[ds["new_algo"] == 1]["churn"].values

# MODEL 1: Minutes with HalfNormal prior for sigma
with pm.Model() as model_minutes:
    mu_old = pm.Normal("mu_old", mu=4, sigma=1)
    mu_new = pm.Normal("mu_new", mu=4, sigma=1)

    # More stable prior for standard deviation
    sigma = pm.HalfNormal("sigma", sigma=1)

    # Likelihoods
    pm.Normal("obs_old", mu=mu_old, sigma=sigma, observed=minutes_old)
    pm.Normal("obs_new", mu=mu_new, sigma=sigma, observed=minutes_new)

    # Derived quantities
    effect = pm.Deterministic("effect", mu_new - mu_old)
    rel_effect = pm.Deterministic("rel_effect", (mu_new - mu_old) / mu_old * 100)

    # Sampling
    trace_minutes = pm.sample(draws=10, tune=10, chains=2, cores=1, return_inferencedata=True, target

# MODEL 2: Churn
with pm.Model() as model_churn:
    p_old = pm.Beta("p_old", alpha=4, beta=1)
    p_new = pm.Beta("p_new", alpha=4, beta=1)
    pm.Bernoulli("obs_old", p=p_old, observed=churn_old)
    pm.Bernoulli("obs_new", p=p_new, observed=churn_new)

    risk_diff = pm.Deterministic("risk_diff", p_new - p_old)
    risk_ratio = pm.Deterministic("risk_ratio", p_new / p_old)
    odds_ratio = pm.Deterministic("odds_ratio", (p_new / (1 - p_new)) / (p_old / (1 - p_old)))

    trace_churn = pm.sample(draws=10, tune=10, chains=2, cores=1, return_inferencedata=True, target_a

# Output results
print("==== Minutes Analysis ====")
print(az.summary(trace_minutes, var_names=["mu_old", "mu_new", "effect", "rel_effect"], round_to=2))
p_effect = (trace_minutes.posterior["effect"] > 0).mean().item()
print(f"Probability new_algo increases minutes: {p_effect:.2%}")

print("\n==== Churn Analysis ====")
print(az.summary(trace_churn, var_names=["p_old", "p_new", "risk_diff", "risk_ratio", "odds_ratio"],
p_churn = (trace_churn.posterior["risk_diff"] < 0).mean().item()
print(f"Probability new_algo reduces churn: {p_churn:.2%}")
```

```
WARNING (pytensor.configdefaults): g++ not available, if using conda: `conda install gxx`
WARNING (pytensor.configdefaults): g++ not detected!  PyTensor will be unable to compile C-implementa
Only 10 samples per chain. Reliable r-hat and ESS diagnostics require longer chains for accurate esti
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (2 chains in 1 job)
NUTS: [mu_old, mu_new, sigma]

Output()

C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pytensor\scalar\basic.py:3191:
RuntimeWarning: overflow encountered in exp
  return np.exp(x)



Sampling 2 chains for 10 tune and 10 draw iterations (20 + 20 draws total) took 213 seconds.
The number of samples is too small to check convergence reliably.
Only 10 samples per chain. Reliable r-hat and ESS diagnostics require longer chains for accurate esti
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (2 chains in 1 job)
NUTS: [p_old, p_new]

Output()



Sampling 2 chains for 10 tune and 10 draw iterations (20 + 20 draws total) took 15 seconds.
The number of samples is too small to check convergence reliably.

==== Minutes Analysis ====
            mean     sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  \
mu_old      4.07   0.90    3.20     4.95       0.40      0.0      5.40
mu_new      4.21   0.25    3.97     4.45       0.11      0.0      5.58
effect      0.14   1.14   -0.98     1.25       0.51      0.0      6.53
rel_effect  9.69  30.22  -19.77    39.15      13.51      0.0      5.57

            ess_tail  r_hat
mu_old         21.74   3.10
mu_new         21.74   2.56
effect         26.02   1.92
rel_effect     21.74   2.55
Probability new_algo increases minutes: 50.00%

==== Churn Analysis ====
            mean    sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  \
p_old       0.10  0.00    0.10     0.11       0.00     0.00     26.02
```

```
p_new       0.08  0.00    0.07     0.08       0.00     0.00     26.02
risk_diff  -0.02  0.01   -0.03    -0.02       0.00     0.00     26.02
risk_ratio  0.77  0.05    0.69     0.82       0.01     0.01     26.02
odds_ratio  0.75  0.06    0.66     0.80       0.01     0.01     26.02

            ess_tail  r_hat
p_old          21.74   0.99
p_new          21.74   1.07
risk_diff      21.74   0.99
risk_ratio     21.74   0.97
odds_ratio     21.74   0.97
Probability new_algo reduces churn: 100.00%
```

Informative priors

```python
import pymc as pm
import arviz as az

with pm.Model() as model_informative_minutes_churn:

    # MINUTES MODEL
    mu_old = pm.Normal("mu_old", mu=4, sigma=1)
    delta = pm.Normal("delta", mu=10, sigma=5)  # effect: new algo increases by ~10 ±5
    mu_new = pm.Deterministic("mu_new", mu_old + delta)

    sigma = pm.HalfNormal("sigma", sigma=10)

    pm.Normal("obs_minutes_old", mu=mu_old, sigma=sigma, observed=minutes_old)
    pm.Normal("obs_minutes_new", mu=mu_new, sigma=sigma, observed=minutes_new)

    effect_minutes = pm.Deterministic("effect_minutes", mu_new - mu_old)  # equals delta

    # CHURN MODEL
    p_old = pm.Beta("p_old", alpha=2, beta=2)
    p_new = pm.Beta("p_new", alpha=2, beta=2)
    pm.Bernoulli("obs_churn_old", p=p_old, observed=churn_old)
    pm.Bernoulli("obs_churn_new", p=p_new, observed=churn_new)

    risk_diff = pm.Deterministic("risk_diff", p_new - p_old)
    risk_ratio = pm.Deterministic("risk_ratio", p_new / p_old)
    odds_ratio = pm.Deterministic("odds_ratio", (p_new / (1 - p_new)) / (p_old / (1 - p_old)))

    trace_inf = pm.sample(draws=10, tune=10, chains=2, cores=1, random_seed=123, return_inferencedata

print("\n MINUTES (Informative Priors) ")
print(az.summary(trace_inf, var_names=["effect_minutes"], hdi_prob=0.95))
effect_m = trace_inf.posterior["effect_minutes"]
```

```python
print(f"P(Δ > 0)  = {(effect_m > 0).mean().item():.2%}")
print(f"P(Δ > 5)  = {(effect_m > 5).mean().item():.2%}")

print("\nCHURN (Informative Priors)")
print(az.summary(trace_inf, var_names=["risk_diff", "risk_ratio", "odds_ratio"], hdi_prob=0.95))
effect_c = trace_inf.posterior["risk_diff"]
print(f"P(Δ < 0)  = {(effect_c < 0).mean().item():.2%}")
print(f"P(Δ < -0.02) = {(effect_c < -0.02).mean().item():.2%}")
```

```
Only 10 samples per chain. Reliable r-hat and ESS diagnostics require longer chains for accurate esti
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (2 chains in 1 job)
NUTS: [mu_old, delta, sigma, p_old, p_new]

Output()

C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pytensor\tensor\elemwise.py:731
RuntimeWarning: overflow encountered in exp
  variables = ufunc(*ufunc_args, **ufunc_kwargs)

C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\numpy\_core\fromnumeric.py:86:
RuntimeWarning: invalid value encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)

C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pytensor\scalar\basic.py:3297:
RuntimeWarning: overflow encountered in scalar multiply
  return x * x

C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\pytensor\scalar\basic.py:1548:
RuntimeWarning: overflow encountered in cast
  return np.greater_equal(x, y)

C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-packages\numpy\_core\fromnumeric.py:86:
RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)

Sampling 2 chains for 10 tune and 10 draw iterations (20 + 20 draws total) took 345 seconds.
The number of samples is too small to check convergence reliably.
```

```
 MINUTES (Informative Priors)
                   mean     sd  hdi_2.5%  hdi_97.5%  mcse_mean  mcse_sd  \
effect_minutes  10.553  0.186    10.371     10.742      0.083    0.002

                ess_bulk  ess_tail  r_hat
effect_minutes       6.0      22.0   2.01
P(Δ > 0)  = 100.00%
P(Δ > 5)  = 100.00%

CHURN (Informative Priors)
             mean     sd  hdi_2.5%  hdi_97.5%  mcse_mean  mcse_sd  ess_bulk  \
risk_diff   0.203  0.152    -0.004      0.348      0.068    0.014       6.0
risk_ratio  1.635  0.524     0.980      2.140      0.234    0.030       6.0
odds_ratio  2.784  1.546     0.975      4.283      0.691    0.067       6.0

            ess_tail  r_hat
risk_diff       26.0   2.27
risk_ratio      26.0   2.38
odds_ratio      26.0   2.27
P(Δ < 0)  = 10.00%
P(Δ < -0.02) = 0.00%
```

interpretation

For Minutes Effect mean on Minutes = 10.548: On average, the new algorithm increases the minutes spent by about 10.5 minutes.
HDI(Minutes) = [10.371, 10.742]: We are 95% confident that the true increase lies between roughly 10.4 and 10.7 minutes.
For Churn Risk difference= 0.139: On average, the churn rate under the new algorithm is 13.9 percentage points higher than the old one.
HDI = [-0.012, 0.246]: We are 95% confident the true risk difference lies between the interval

```python
#minutes increase by  5%
p_minutes_5pct = (trace_minutes.posterior["rel_effect"] >= 5).mean().item()
print(f"Probability new_algo increases minutes by at least 5%: {p_minutes_5pct:.2%}")

#reduces by  2 percentage points
p_churn_2pt = (trace_churn.posterior["risk_diff"] <= -0.02).mean().item()
print(f"Probability new_algo reduces churn by at least 2 percentage points: {p_churn_2pt:.2%}")
```

```
Probability new_algo increases minutes by at least 5%: 50.00%
Probability new_algo reduces churn by at least 2 percentage points: 70.00%
```