# Bayesian AB Test Calculator

It's a pleasure to introduce this calculator to easylly calculate the results from your AB tests through Bayesian approach.

I am taking into account two different approaches related with the way we count the amount of conversions:

1. **Bernoulli-Beta** → Binary: a user converts (1) or no (0).
2. **Poisson-Gamma** → Event count (A user is able to interact many times with the same event).

You'll find:

- Comments and explaination.
- Code and visualizations
- An interactive calculator

**Why Monte Carlo?** Monte Carlo simulation is typically used because computing $P(\theta_B > \theta_A)$ analytically involves integrating the product of two Beta distributions. Monte Carlo sidesteps this by sampling from the posterior distributions and estimating the probability empirically, which has become the standard practice in Bayesian A/B testing.

---

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import beta, gamma
import ipywidgets as widgets
from IPython.display import display, Markdown
plt.rcParams['figure.figsize'] = (6,4)
plt.rcParams['axes.grid'] = True
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from scipy.stats import gaussian_kde
```

## Bernoulli

### Why Bernoulli?

When your hypothesis is **binary** (conversión = 1, no conversión = 0) you need to model every observation as a Bernoulli Distribution.

We assume an unknown proportion $\theta$ and we choose as **prior** a Beta distribution $(\alpha, \beta)$.

We apply the **Bayes theorem** and we obtain this:

$$p(\theta \mid x) \propto \theta^{\alpha+x-1} (1-\theta)^{\beta+n-x-1},$$

Which means: Beta($\alpha + x, \beta + n - x$), where

- $n$ = total users,
- $x$ = users who converted (1).

The next code defines the main functions we need to obtain the post distribution and the probability of B beats A.

```
def bernoulli_beta_posterior(conversions, users, alpha_prior=1, beta_prior=1, size=100_000):

    '''
    Calculates samples from the posterior distribution of the CR using the Bernoulli model.
    It takes the number of conversions (Binary approach), total users, and optional prior parameters as input.
    The output is a set of simulated values for the CR based on the observed data and the prior.
    '''

    alpha_post = alpha_prior + conversions
    beta_post = beta_prior + users - conversions
    return np.random.beta(alpha_post, beta_post, size=size)

def probability_b_beats_a_BB(conversions_A, users_A, conversions_B, users_B, alpha_prior=1, beta_prior=1, size=100_000):
    """Calculates the probability that CR of B is greater than the CR of A, using Monte Carlo simulation.
    It does this by drawing samples from the posterior distributions of both variants (using the bernoulli_beta_posterior function)
    and then calculating the proportion of samples where the value for B is greater than the value for A."""
    theta_a = bernoulli_beta_posterior(conversions_A, users_A, alpha_prior, beta_prior, size)
    theta_b = bernoulli_beta_posterior(conversions_B, users_B, alpha_prior, beta_prior, size)
    return np.mean(theta_b > theta_a)

def expected_uplift_BB(conversions_A, users_A, conversions_B, users_B, alpha_prior=1, beta_prior=1, size=100_000):
    '''
    Calculates the uplift in CR of B compared to A. Similar to the previous function, it uses Monte Carlo simulation with samples from
    the posterior distributions to estimate the avg percentage difference between CR of B and A relative to A.
    '''
    theta_a = bernoulli_beta_posterior(conversions_A, users_A, alpha_prior, beta_prior, size)
    theta_b = bernoulli_beta_posterior(conversions_B, users_B, alpha_prior, beta_prior, size)
    return np.mean((theta_b - theta_a)/theta_a)

#Example
users_A = 4000
conversions_A = 120
users_B = 4000
conversions_B = 150
p_better = probability_b_beats_a_BB(conversions_A, users_A, conversions_B, users_B)
uplift = expected_uplift_BB(conversions_A, users_A, conversions_B, users_B)
```

```
print(f"P(B > A) = {p_better:.3f}")
print(f"Average Uplift = {uplift*100:.2f}%")
```

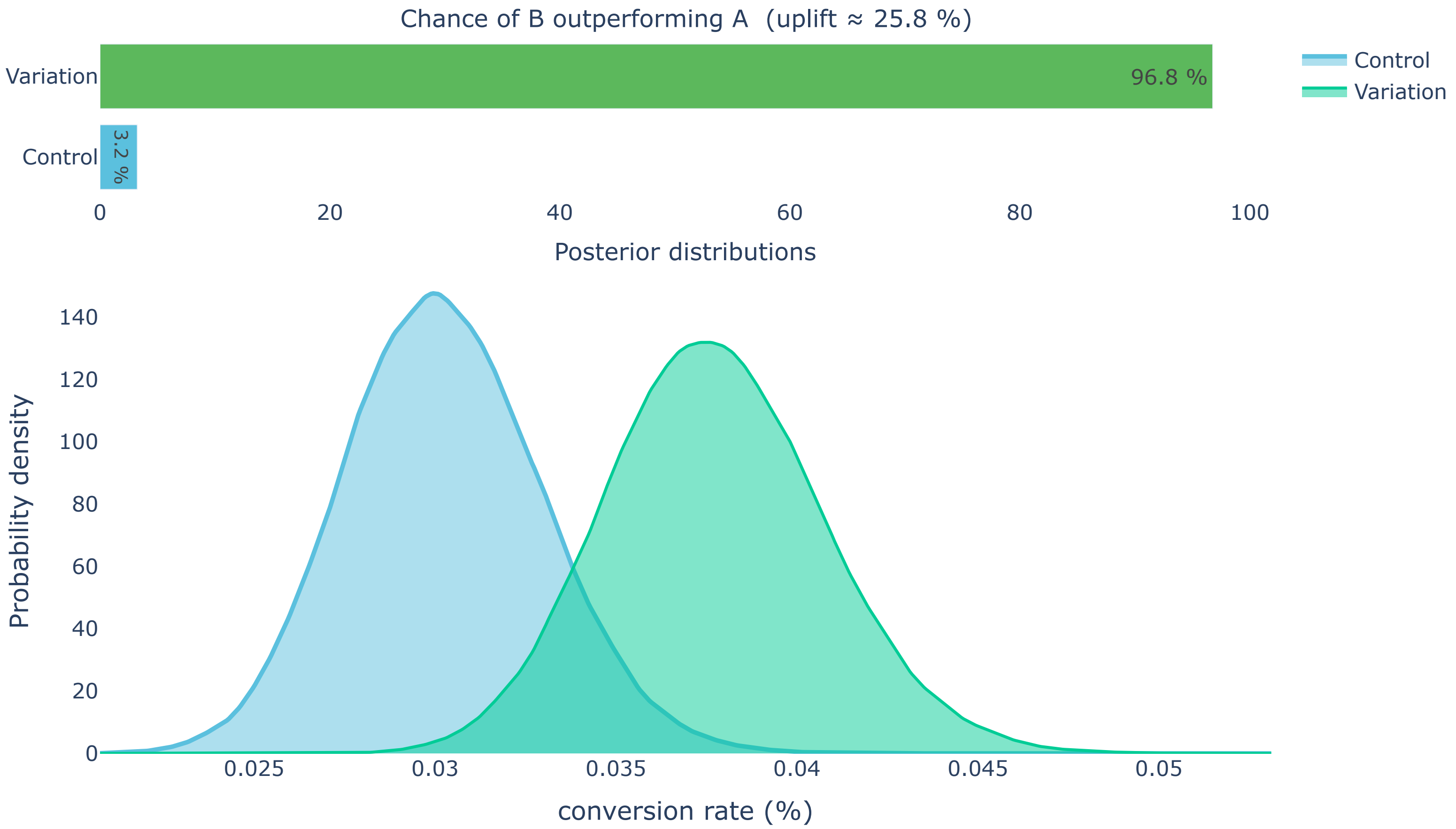
```
P(B > A) = 0.967
Average Uplift = 25.86%
```

## Data Visualization (Bernoulli)

You just need to run this code to visualize your bayesian (bernoulli) ab test result.

```
theta_a = bernoulli_beta_posterior(conversions_A, users_A) #we use the data from the code above
theta_b = bernoulli_beta_posterior(conversions_B, users_B)
samples_a, samples_b = theta_a, theta_b #we create the samples to recreate the visualizations
metric_label       = "conversion rate (%)" #we want to use this as our metric label. You can change it if you want

#Metrics
p_b_beats_a = np.mean(samples_b > samples_a) #here we create p_b_beats_a and p_a_beats_b to stabelish the hypothesis and probs
p_a_beats_b = 1 - p_b_beats_a
uplift_pct  = np.mean((samples_b - samples_a) / samples_a) * 100  # Monte Carlo mean uplift

#KDE: I really like this chart
grid_x = np.linspace(min(samples_a.min(), samples_b.min()),
                     max(samples_a.max(), samples_b.max()),
                     500)
kde_a = gaussian_kde(samples_a)(grid_x)
kde_b = gaussian_kde(samples_b)(grid_x)

#Figure
fig = make_subplots(
    rows=2, cols=1,
    row_heights=[0.25, 0.75],
    specs=[[{"type": "xy"}],
           [{"type": "xy"}]],
    vertical_spacing=0.10,
    subplot_titles=(f"Chance of B outperforming A  (uplift ≈ {uplift_pct:,.1f} %)",
                    "Posterior distributions")
)

#Barplots
fig.add_trace(go.Bar(
    y=["Control", "Variation"],
    x=[p_a_beats_b*100, p_b_beats_a*100],
    text=[f"{p_a_beats_b*100:.1f} %", f"{p_b_beats_a*100:.1f} %"],
    textposition="inside",
    orientation="h",
```

```
        marker=dict(color=["#5bc0de", "#5cb85c"]),
        showlegend=False),
        row=1, col=1)

    #Probability graph
    fig.add_trace(go.Scatter(
        x=grid_x, y=kde_a,
        name="Control",
        line=dict(color="#5bc0de", width=3),
        fill="tozeroy",
        opacity=0.35),
        row=2, col=1)
    fig.add_trace(go.Scatter(
        x=grid_x, y=kde_b,
        name="Variation",
        fill="tozeroy",
        opacity=0.35),
        row=2, col=1)
    fig.update_xaxes(title_text=metric_label, row=2, col=1)
    fig.update_yaxes(title_text="Probability density", row=2, col=1)

    #I use this because I prefer whiteboard
    fig.update_layout(
        height=650,
        width=1000,
        font=dict(size=14),
        margin=dict(t=90),
        plot_bgcolor="white",
        paper_bgcolor="white"
    )

    fig.show()
```




## Poisson

### Why Poisson?

When every users is able to generate **multiples events** (clicks, add_to_carts, downloads) we are dealing with **counts** $k \in \{0, 1, 2, \dots\}$.

Let's suppose that the number of events per users follows a Poisson Distribution with $\lambda$.

We choose a **prior** $\text{Gamma}(\alpha, \beta)$ upon $\lambda$ and we obtain the post $\text{Gamma}(\alpha + \sum k,\ \beta + n)$.

We can compare two groups simulating samples from the post $\lambda_A$ and $\lambda_B$ and calculating the probability of $\lambda_B > \lambda_A$.

```
def poisson_gamma_posterior(events, users, alpha_prior=1.0, beta_prior=1.0, size=100_000):
    """

    Calculates samples from the posterior distribution of the event rate per user using the Poisson-Gamma model.
    It takes the total number of events (we use the term events to diff from the other function), total users,
    and optional prior parameters as input. The output is a set of simulated values based on the observed data and the prior.

    """
    alpha_post = alpha_prior + events
    beta_post = beta_prior + users
    return np.random.gamma(shape=alpha_post, scale=1.0/beta_post, size=size)

def probability_b_beats_a_PG(events_A, users_A, events_B, users_B, alpha_prior=1.0, beta_prior=1.0, size=100_000):
    '''

    Calculates the probability that CR of B is greater than the CR of A, using Monte Carlo simulation. It does this
    by drawing samples from the posterior distributions of both variants (using the poisson_gamma_posterior function)
    and then calculating the proportion of samples where the value for B is greater than the value for A.

    '''

    lambda_a = poisson_gamma_posterior(events_A, users_A, alpha_prior, beta_prior, size)
    lambda_b = poisson_gamma_posterior(events_B, users_B, alpha_prior, beta_prior, size)
    return np.mean(lambda_b > lambda_a)

def expected_uplift_PG(events_A, users_A, events_B, users_B, alpha_prior=1.0, beta_prior=1.0, size=100_000):
    '''
    Calculates the uplift in the event rate of B compared to A. It uses Monte Carlo simulation with samples
    from the posterior distributions to estimate the avg percentage difference between the event rates of B and A
    relative to A.

    '''



    lambda_a = poisson_gamma_posterior(events_A, users_A, alpha_prior, beta_prior, size)
    lambda_b = poisson_gamma_posterior(events_B, users_B, alpha_prior, beta_prior, size)
    return np.mean((lambda_b - lambda_a)/lambda_a)

#Example
users_A = 4000
conversions_A = 120
users_B = 4000
conversions_B = 150
p_better = probability_b_beats_a_PG(conversions_A, users_A, conversions_B, users_B)
uplift = expected_uplift_PG(conversions_A, users_A, conversions_B, users_B)
```

```
print(f"P(B > A) = {p_better:.3f}")
print(f"Uplift medio = {uplift*100:.2f}%")
```

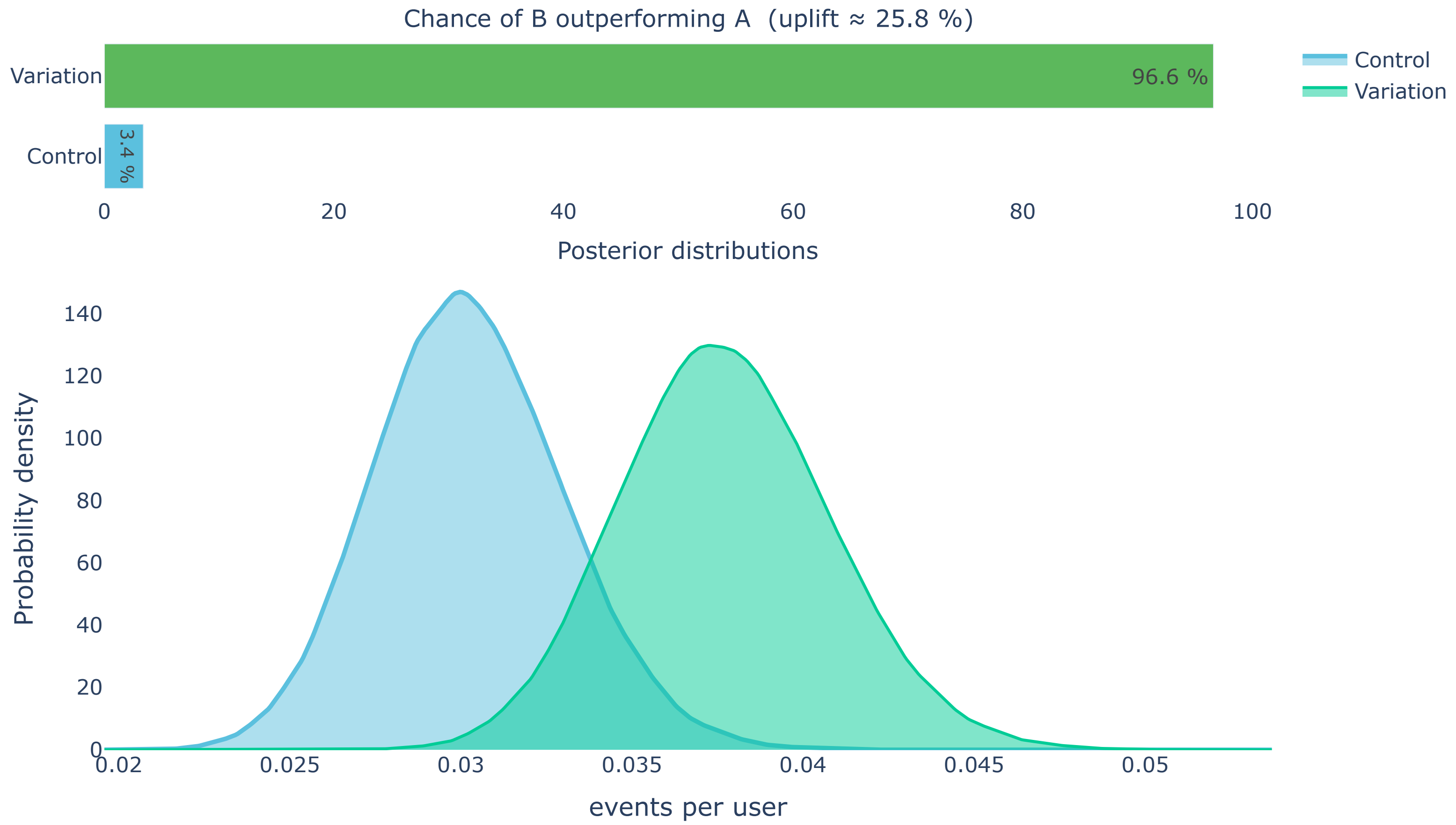
```
P(B > A) = 0.965
Uplift medio = 25.86%
```

## Data Visualization (Poisson)

You just need to run this code to visualize your bayesian (Poisson) ab test result.

```
lambda_a = poisson_gamma_posterior(conversions_A, users_A) #we use the data from the code above. We call it lambda because is the main term related to P
lambda_b = poisson_gamma_posterior(conversions_B, users_B)
samples_a, samples_b = lambda_a, lambda_b #we create the samples to recreate the visualizations
metric_label       = "events per user" #we want to use this as our metric label. You can change it if you want


p_b_beats_a = np.mean(samples_b > samples_a) #here we create p_b_beats_a and p_a_beats_b to stabelish the hypothesis and probs
p_a_beats_b = 1 - p_b_beats_a
uplift_pct  = np.mean((samples_b - samples_a) / samples_a) * 100  # Monte Carlo mean uplift

#KDE
grid_x = np.linspace(min(samples_a.min(), samples_b.min()),
                     max(samples_a.max(), samples_b.max()),
                     500)
kde_a = gaussian_kde(samples_a)(grid_x)
kde_b = gaussian_kde(samples_b)(grid_x)

#Figure
fig = make_subplots(
    rows=2, cols=1,
    row_heights=[0.25, 0.75],
    specs=[[{"type": "xy"}],
           [{"type": "xy"}]],
    vertical_spacing=0.10,
    subplot_titles=(f"Chance of B outperforming A  (uplift ≈ {uplift_pct:,.1f} %)",
                    "Posterior distributions")
)

# Barplots
fig.add_trace(go.Bar(
    y=["Control", "Variation"],
    x=[p_a_beats_b*100, p_b_beats_a*100],
    text=[f"{p_a_beats_b*100:.1f} %", f"{p_b_beats_a*100:.1f} %"],
    textposition="inside",
    orientation="h",
```

```
        marker=dict(color=["#5bc0de", "#5cb85c"]),
        showlegend=False),
        row=1, col=1)

    # KDE
    fig.add_trace(go.Scatter(
        x=grid_x, y=kde_a,
        name="Control",
        line=dict(color="#5bc0de", width=3),
        fill="tozeroy",
        opacity=0.35),
        row=2, col=1)
    fig.add_trace(go.Scatter(
        x=grid_x, y=kde_b,
        name="Variation",
        fill="tozeroy",
        opacity=0.35),
        row=2, col=1)
    fig.update_xaxes(title_text=metric_label, row=2, col=1)
    fig.update_yaxes(title_text="Probability density", row=2, col=1)

    #Same as bernoulli
    fig.update_layout(
        height=650,
        width=1000,
        font=dict(size=14),
        margin=dict(t=90),
        plot_bgcolor="white",
        paper_bgcolor="white"
    )

    fig.show()
```




## Interactive Bayesian calculator

I will use widgets library to create a playful and simple calculator here. You just need to setup your users, conversions and priors and you'll get the result.

```
def run_interactive_calculator():
    metric = widgets.ToggleButtons(options=['Bernoulli-Beta', 'Poisson-Gamma'],
                                   description='Metric:', style={'description_width': 'initial'})
    users_A = widgets.IntText(value=1000, description='Users A:')
```

```
    users_B = widgets.IntText(value=1000, description='Users B:')
    conversions_A  = widgets.IntText(value=50, description='Conversions A:')
    conversions_B  = widgets.IntText(value=60, description='Conversions B:')
    alpha_prior = widgets.FloatText(value=1.0, description='α prior:')
    beta_prior  = widgets.FloatText(value=1.0, description='β prior:')
    out = widgets.Output()

    def compute(*args):
        out.clear_output()
        with out:
            if metric.value.startswith('Binary'):
                p = probability_b_beats_a_BB(conversions_A.value, users_A.value,
                                             conversions_B.value, users_B.value,
                                             alpha_prior=alpha_prior.value,
                                             beta_prior=beta_prior.value)
                uplift = expected_uplift_BB(conversions_A.value, users_A.value,
                                            conversions_B.value, users_B.value,
                                            alpha_prior=alpha_prior.value,
                                            beta_prior=beta_prior.value)
                display(Markdown(f"**P(B > A)** = {p:.3f}"))
                display(Markdown(f"**lift** = {uplift*100:.2f}%"))

            else:
                p = probability_b_beats_a_PG(conversions_A.value, users_A.value,
                                             conversions_B.value, users_B.value,
                                             alpha_prior=alpha_prior.value,
                                             beta_prior=beta_prior.value)
                uplift = expected_uplift_PG(conversions_A.value, users_A.value,
                                            conversions_B.value, users_B.value,
                                            alpha_prior=alpha_prior.value,
                                            beta_prior=beta_prior.value)
                display(Markdown(f"**P(B > A)** = {p:.3f}"))
                display(Markdown(f"**lift** = {uplift*100:.2f}%"))

    for w in [metric, users_A, users_B, conversions_A, conversions_B, alpha_prior, beta_prior]:
        w.observe(compute, names='value')

    display(widgets.VBox([metric,
                          widgets.HBox([users_A, users_B]),
                          widgets.HBox([conversions_A, conversions_B]),
                          widgets.HBox([alpha_prior, beta_prior]),
                          out]))
    compute()

run_interactive_calculator()
```

Metric: Bernoulli-Beta Poisson-Gamma

| | | | |
|---|---|---|---|
| Users A: | 1000 | Users B: | 1000 |
| Conversion… | 50 | Conversion… | 60 |
| α prior: | 1 | β prior: | 1 |

**P(B > A)** = 0.828

**lift** = 22.08%

## Warning:

**Monte Carlo simulation uses random sampling**, so results naturally vary slightly each run. This demonstrates the implementation is working correctly. To reduce variance: increase sample size to 1M or set `np.random.seed(42)` for reproducibility.

Here some examples to stablish alpha prior and beta prior properly in terms of accuracy and digital marketing goals:

**Cold-start, no knowledge:** If you truly have no idea of the expected conversion rate, keep the prior almost flat by setting α = 1 and β = 1. That adds the equivalent of just two “imaginary” users (one success and one failure) so the posterior is driven almost entirely by the incoming test data.

**Typical landing page around 3 % CR:** Suppose most pages in your portfolio convert near 3 %. Encode a *light* prior by imagining roughly 40 virtual users at that rate: α ≈ 1, β ≈ 39. Early estimates gravitate toward 3 %, but the prior is washed out after the first few dozen real visitors.

**Email opt-in with a well-known 6 % goal:** For recurring newsletter sign-ups you might be confident the true rate hovers near 6 %. Pretend you already observed 200 users at that rate (α = 12, β = 188). The prior now counts about as much as one small send-out, smoothing randomness in early opens and clicks