

# Package: grizbayr (via r-universe)

September 3, 2024

**Type** Package

**Title** Bayesian Inference for A/B and Bandit Marketing Tests

**Version** 1.3.5

**Author** Ryan Angi

**Maintainer** Ryan Angi <angi.ryan@gmail.com>

**Description** Uses simple Bayesian conjugate prior update rules to calculate the win probability of each option, value remaining in the test, and percent lift over the baseline for various marketing objectives. References: Fink, Daniel (1997) ``A Compendium of Conjugate Priors" <<https://www.johndcook.com/CompendiumOfConjugatePriors.pdf>>. Stucchio, Chris (2015) ``Bayesian A/B Testing at VWO" <[https://vwo.com/downloads/VWO\\_SmartStats\\_technical\\_whitepaper.pdf](https://vwo.com/downloads/VWO_SmartStats_technical_whitepaper.pdf)>.

**Depends** R (>= 2.10)

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Imports** purrr, dplyr, tidyr (>= 1.0.0), magrittr, tibble, rlang

**Suggests** spelling, knitr, testthat (>= 2.1.0), rmarkdown

**Language** en-US

**VignetteBuilder** knitr

**URL** <https://github.com/rangi513/grizbayr>

**BugReports** <https://github.com/rangi513/grizbayr/issues>

**Repository** <https://rangi513.r-universe.dev>

**RemoteUrl** <https://github.com/rangi513/grizbayr>

**RemoteRef** HEAD

**RemoteSha** 8102c9f759419932fa4b7de1ad2b5b967a75d45f

## Contents

calculate_multi_rev_per_session . . . . .	3
calculate_total_cm . . . . .	3
estimate_all_values . . . . .	4
estimate_lift . . . . .	5
estimate_lift_vs_baseline . . . . .	6
estimate_loss . . . . .	7
estimate_value_remaining . . . . .	8
estimate_win_prob . . . . .	9
estimate_win_prob_given_posterior . . . . .	10
estimate_win_prob_vs_baseline . . . . .	10
estimate_win_prob_vs_baseline_given_posterior . . . . .	11
find_best_option . . . . .	12
impute_missing_options . . . . .	13
is_prior_valid . . . . .	13
is_winner_max . . . . .	14
rdirichlet . . . . .	14
sample_cm_per_click . . . . .	15
sample_conv_rate . . . . .	15
sample_cpa . . . . .	16
sample_cpc . . . . .	17
sample_ctr . . . . .	18
sample_from_posterior . . . . .	19
sample_multi_rev_per_session . . . . .	20
sample_page_views_per_session . . . . .	21
sample_response_rate . . . . .	22
sample_rev_per_session . . . . .	22
sample_session_duration . . . . .	24
sample_total_cm . . . . .	25
update_beta . . . . .	26
update_dirichlet . . . . .	26
update_gamma . . . . .	27
validate_data_values . . . . .	28
validate_input_column . . . . .	28
validate_input_df . . . . .	29
validate_posterior_samples . . . . .	29
validate_priors . . . . .	30
validate_wrt_option . . . . .	30

---

calculate\_multi\_rev\_per\_session  
*Calculate Multi Rev Per Session*

---

**Description**

Calculate Multi Rev Per Session

**Usage**

```
calculate_multi_rev_per_session(conv_rates, inverse_rev_A, inverse_rev_B)
```

**Arguments**

conv\_rates      Dirichlet samples containing a tibble with columns alpha\_1, alpha\_2, and alpha\_0  
inverse\_rev\_A    Vector of inverse revenue samples from A conversion type  
inverse\_rev\_B    Vector of inverse revenue samples from B conversion type

**Value**

Vector of samples (dbl)

---

calculate\_total\_cm      *Calculate Total CM*

---

**Description**

Calculate Total CM

**Usage**

```
calculate_total_cm(rev_per_click, cost_per_click, expected_clicks)
```

**Arguments**

rev\_per\_click    vector of rev per click samples  
cost\_per\_click   vector of cost per click (cpc) samples  
expected\_clicks      vector of expected clicks (expected CTR \* fixed impressions)

**Value**

vector of CM estimates (dbl)

---

estimate\_all\_values     *Estimate All Values*

---

### Description

Efficiently estimates all values at once so the posterior only need to be sampled one time. This function will return as a list win probability, value remaining, estimated percent lift with respect to the provided option, and the win probability of the best option vs the provided option.

### Usage

```
estimate_all_values(
  input_df,
  distribution,
  wrt_option_lift,
  priors = list(),
  wrt_option_vr = NULL,
  loss_threshold = 0.95,
  lift_threshold = 0.7,
  metric = "lift"
)
```

### Arguments

input_df	Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.
distribution	String of the distribution name
wrt_option_lift	String: the option lift and win probability is calculated with respect to (wrt). Required.
priors	Optional list of priors. Defaults will be use otherwise.
wrt_option_vr	String: the option against which loss (value remaining) is calculated. If NULL the best option will be used. (optional)
loss_threshold	The confidence interval specifying what the "worst case scenario" should be. Defaults to 95%. (optional)
lift_threshold	The confidence interval specifying how likely the lift is to be true. Defaults to 70%. (optional)
metric	string the type of loss. absolute will be the difference, on the outcome scale. 0 when best = wrt_option lift will be the (best - wrt_option) / wrt_option, 0 when best = wrt_option relative_risk will be the ratio best/wrt_option, 1 when best = wrt_option

### Details

TODO: Add high density credible intervals to this output for each option.

**Value**

A list with 4 named items: Win Probability, Value Remaining, Lift vs Baseline, and Win Probability vs Baseline.

**Examples**

```
## Not run:
input_df <- data.frame(option_name = c("A", "B", "C"),
  sum_clicks = c(1000, 1000, 1000),
  sum_conversions = c(100, 120, 110), stringsAsFactors = FALSE)
estimate_all_values(input_df, distribution = "conversion_rate", wrt_option_lift = "A")

## End(Not run)
```

---

estimate_lift	<i>Estimate Lift Distribution</i>
---------------	-----------------------------------

---

**Description**

Estimates lift distribution vector from posterior samples.

**Usage**

```
estimate_lift(posterior_samples, distribution, wrt_option, metric = "lift")
```

**Arguments**

posterior_samples	Tibble returned from <code>sample_from_posterior</code> with 3 columns ‘option_name’, ‘samples’, and ‘sample_id’.
distribution	String of the distribution name
wrt_option	string the option lift is calculated with respect to (wrt). Required.
metric	string the type of lift. ‘absolute’ will be the difference, on the outcome scale. 0 when best = wrt_option ‘lift’ will be the (best - wrt_option) / wrt_option, 0 when best = wrt_option ‘relative_risk’ will be the ratio best/wrt_option, 1 when best = wrt_option

**Value**

numeric, the lift distribution

**Examples**

```
# Requires posterior_samples dataframe. See `sample_from_posterior()`
# for an example.

## Not run:
estimate_lift(posterior_samples = posterior_samples,
              distribution = "conversion_rate",
              wrt_option = "A",
              metric = "lift")

## End(Not run)
```

---

```
estimate_lift_vs_baseline
```

*Estimate Lift vs Baseline*

---

**Description**

Estimate Lift vs Baseline

**Usage**

```
estimate_lift_vs_baseline(
  input_df,
  distribution,
  priors = list(),
  wrt_option,
  metric = "lift",
  threshold = 0.7
)
```

**Arguments**

input_df	Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.
distribution	String of the distribution name
priors	Optional list of priors. Defaults will be use otherwise.
wrt_option	string the option loss is calculated with respect to (wrt). Required.
metric	string the type of loss. absolute will be the difference, on the outcome scale. 0 when best = wrt_option lift will be the (best - wrt_option) / wrt_option, 0 when best = wrt_option relative_risk will be the ratio best/wrt_option, 1 when best = wrt_option
threshold	Lift percentage threshold between 0 and 1. (0.7 threshold is "at least 70% lift"). Defaults to 0.7.

**Value**

numeric value remaining at the specified threshold

**Examples**

```
input_df <- tibble::tibble(option_name = c("A", "B", "C"),
  sum_clicks = c(1000, 1000, 1000),
  sum_conversions = c(100, 120, 110))
estimate_lift_vs_baseline(input_df, distribution = "conversion_rate", wrt_option = "A")
```

---

estimate\_loss

*Estimate Loss*

---

**Description**

Estimate Loss

**Usage**

```
estimate_loss(
  posterior_samples,
  distribution,
  wrt_option = NULL,
  metric = c("absolute", "lift", "relative_risk")
)
```

**Arguments**

posterior_samples	Tibble: returned from sample_from_posterior with 3 columns 'option_name', 'samples', and 'sample_id'.
distribution	String: the name of the distribution
wrt_option	String: the option loss is calculated with respect to (wrt). If NULL, the best option will be chosen.
metric	String: the type of loss. absolute will be the difference, on the outcome scale. 0 when best = wrt_option lift will be the (best - wrt_option) / wrt_option, 0 when best = wrt_option relative_risk will be the ratio best/wrt_option, 1 when best = wrt_option

**Value**

numeric, the loss distribution

**Examples**

```
# Requires posterior_samples dataframe. See `sample_from_posterior()`
# for an example.

## Not run:
estimate_loss(posterior_samples = posterior_samples, distribution = "conversion_rate")

## End(Not run)
```

---

```
estimate_value_remaining
      Estimate Value Remaining
```

---

**Description**

Estimates value remaining or loss (in terms of percent lift, absolute, or relative).

**Usage**

```
estimate_value_remaining(
  input_df,
  distribution,
  priors = list(),
  wrt_option = NULL,
  metric = "lift",
  threshold = 0.95
)
```

**Arguments**

input_df	Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.
distribution	String of the distribution name
priors	Optional list of priors. Defaults will be use otherwise.
wrt_option	string the option loss is calculated with respect to (wrt). If NULL, the best option will be chosen.
metric	string the type of loss. absolute will be the difference, on the outcome scale. 0 when best = wrt_option lift will be the (best - wrt_option) / wrt_option, 0 when best = wrt_option relative_risk will be the ratio best/wrt_option, 1 when best = wrt_option
threshold	The confidence interval specifying what the "worst case scenario should be. Defaults to 95%. (optional)

**Value**

numeric value remaining at the specified threshold



**Examples**

```
input_df <- tibble::tibble(option_name = c("A", "B", "C"),
  sum_clicks = c(1000, 1000, 1000),
  sum_conversions = c(100, 120, 110))
estimate_value_remaining(input_df, distribution = "conversion_rate")
estimate_value_remaining(input_df,
  distribution = "conversion_rate",
  threshold = 0.99)
estimate_value_remaining(input_df,
  distribution = "conversion_rate",
  wrt_option = "A",
  metric = "absolute")
```

---

estimate_win_prob	<i>Estimate Win Probability</i>
-------------------	---------------------------------

---

**Description**

Creates a tibble of win probabilities for each option based on the data observed.

**Usage**

```
estimate_win_prob(input_df, distribution, priors = list())
```

**Arguments**

input_df	Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.
distribution	String of the distribution name
priors	Optional list of priors. Defaults will be use otherwise.

**Value**

tibble object with 2 columns: 'option\_name' and 'win\_probability' formatted as a percent

**Examples**

```
input_df <- tibble::tibble(
  option_name = c("A", "B"),
  sum_clicks = c(1000, 1000),
  sum_conversions = c(100, 120)
)
estimate_win_prob(input_df, "conversion_rate")
```

---

 estimate\_win\_prob\_given\_posterior

*Estimate Win Probability Given Posterior Distribution*


---

### Description

Estimate Win Probability Given Posterior Distribution

### Usage

```
estimate_win_prob_given_posterior(posterior_samples, winner_is_max = TRUE)
```

### Arguments

posterior\_samples

Tibble of data in long form with 2 columns 'option\_name' and 'samples'

winner\_is\_max

Boolean. This should almost always be TRUE. If a larger number is better then this should be TRUE. This should be FALSE for metrics such as CPA or CPC where a higher cost is not necessarily better.

### Value

Tibble of each option\_name and the win probability expressed as a percentage and a decimal 'raw'

### Examples

```
# Requires posterior_samples dataframe. See `sample_from_posterior()`
# for an example.
## Not run:
estimate_win_prob_given_posterior(posterior_samples = posterior_samples)
estimate_win_prob_given_posterior(
  posterior_samples = posterior_samples,
  winner_is_max = TRUE
)

## End(Not run)
```

---

 estimate\_win\_prob\_vs\_baseline

*Estimate Win Probability vs. Baseline*


---

### Description

Calculates the win probability of the best option compared to a single other option given an input\_df

**Usage**

```
estimate_win_prob_vs_baseline(
  input_df,
  distribution,
  priors = list(),
  wrt_option
)
```

**Arguments**

input_df	Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.
distribution	String of the distribution name
priors	Optional list of priors. Defaults will be use otherwise.
wrt_option	string the option win prob is calculated with respect to (wrt). Required.

**Value**

Tibble of each option\_name and the win probability expressed as a percentage and a decimal 'raw'

**Examples**

```
input_df <- tibble::tibble(
  option_name = c("A", "B", "C"),
  sum_clicks = c(1000, 1000, 1000),
  sum_conversions = c(100, 120, 110)
)
estimate_win_prob_vs_baseline(input_df = input_df,
  distribution = "conversion_rate",
  wrt_option = "B")
```

---

```
estimate_win_prob_vs_baseline_given_posterior
```

*Estimate Win Probability vs. Baseline Given Posterior*

---

**Description**

Calculates the win probability of the best option compared to a single other option given a posterior distribution.

**Usage**

```
estimate_win_prob_vs_baseline_given_posterior(
  posterior_samples,
  distribution,
  wrt_option
)
```

**Arguments**

posterior\_samples  
Tibble returned from `sample_from_posterior` with 3 columns 'option\_name', 'samples', and 'sample\_id'.

distribution String: the distribution name

wrt\_option String: the option to compare against the best option.

**Value**

Tibble of each option\_name and the win probability expressed as a percentage and a decimal 'raw'

**Examples**

```
# Requires posterior_samples dataframe. See `sample_from_posterior()`
# for an example.
## Not run:
estimate_win_prob_vs_baseline_given_posterior(
  posterior_samples = posterior_samples,
  distribution = "conversion_rate",
  wrt_option = "A")

## End(Not run)
```

---

find_best_option	<i>Find Best Option</i>
------------------	-------------------------

---

**Description**

Samples from posterior, calculates win probability, and selects the best option. Note: this can be inefficient if you already have the win probability dataframe. Only use this if that has not already been calculated.

**Usage**

```
find_best_option(posterior_samples, distribution)
```

**Arguments**

posterior\_samples  
Tibble returned from `sample_from_posterior` with 3 columns 'option\_name', 'samples', and 'sample\_id'.

distribution String: name of the distribution

**Value**

String: the best option name

**Examples**

```
# Requires posterior distribution
## Not run:
find_best_option(posterior_samples = posterior_samples, distribution = "conversion_rate")

## End(Not run)
```

---

```
impute_missing_options
      Impute Missing Options
```

---

**Description**

When win probability is calculated

**Usage**

```
impute_missing_options(posterior_samples, wp_raw)
```

**Arguments**

```
posterior_samples      Tibble of data in long form with 2 columns 'option_name' and 'samples'
wp_raw                  Tibble of win probabilities with the columns: 'option_name' and 'win_prob_raw'
```

**Value**

wp\_raw table with new rows if option names were missing.

---

```
is_prior_valid      Is Prior Valid
```

---

**Description**

Checks if a single valid prior name is in the list of prior values and if that prior value from the list is greater than 0.

**Usage**

```
is_prior_valid(priors_list, valid_prior)
```

**Arguments**

```
priors_list    A list of valid priors
valid_prior    A character string
```

**Value**

Boolean (TRUE/FALSE)

---

is_winner_max	<i>Is Winner Max</i>
---------------	----------------------

---

**Description**

Determines if the max or min function should be used for win probability. If CPA or CPC distribution, lower is better, else higher number is better.

**Usage**

```
is_winner_max(distribution)
```

**Arguments**

distribution    String: the name of the distribution

**Value**

Boolean TRUE/FALSE

---

rdirichlet	<i>Random Dirichlet</i>
------------	-------------------------

---

**Description**

Randomly samples a vector of length n from a dirichlet distribution parameterized by a vector of alphas PDF of Gamma with scale = 1 :  $f(x) = 1/(\Gamma(a)) x^{(a-1)} e^{-x}$

**Usage**

```
rdirichlet(n, alphas_list)
```

**Arguments**

n                    integer, the number of samples  
 alphas\_list        Named List of Integers: parameters of the dirichlet, interpreted as the number of success of each outcome

**Value**

n x length(alphas) named tibble representing the probability of observing each outcome

**Examples**

```
rdirichlet(100, list(a = 20, b = 15, c = 60))
```

---

sample_cm_per_click	<i>Sample CM Per Click</i>
---------------------	----------------------------

---

**Description**

Adds 4 new nested columns to the input\_df: ‘beta\_params’, ‘gamma\_params\_rev’, ‘gamma\_params\_cost’ and ‘samples’

**Usage**

```
sample_cm_per_click(input_df, priors, n_samples = 50000)
```

**Arguments**

input_df	Dataframe containing option_name (str), sum_conversions (dbl), sum_revenue (dbl), and sum_clicks (dbl).
priors	Optional list of priors alpha0, beta0 for Beta, k0, theta0 for Gamma Inverse Revenue, and k01, theta01 for Gamma Cost (uses alternate priors so they can be different from Revenue). Default <i>Beta</i> (1,1) and <i>Gamma</i> (1,250) will be use otherwise.
n_samples	Optional integer value. Defaults to 50,000 samples.

**Details**

‘beta\_params’ and ‘gamma\_params\_rev’ in each row should be a tibble of length 2 ( $\alpha$  and  $\beta$  parameters and  $k$  and  $\theta$  parameters) ‘samples’ in each row should be a tibble of length ‘n\_samples’

See update\_rules vignette for a mathematical representation.

$$CMPerClick = ConversionsPerClick * RevPerConversion - CostPerClick$$

**Value**

input\_df with 4 new nested columns ‘beta\_params’, ‘gamma\_params\_rev’, ‘gamma\_params\_cost’, and ‘samples’

---

sample_conv_rate	<i>Sample Conversion Rate</i>
------------------	-------------------------------

---

**Description**

Adds 2 new nested columns to the input\_df: ‘beta\_params’ and ‘samples’ ‘beta\_params’ in each row should be a tibble of length 2 ( $\alpha$  and  $\beta$  parameters) ‘samples’ in each row should be a tibble of length ‘n\_samples’

**Usage**

```
sample_conv_rate(input_df, priors, n_samples = 50000)
```

**Arguments**

`input_df` Dataframe containing `option_name` (str), `sum_conversions` (dbl), and `sum_clicks` (dbl).

`priors` Optional list of priors `alpha0` and `beta0`. Default  $Beta(1, 1)$  will be use otherwise.

`n_samples` Optional integer value. Defaults to 50,000 samples.

**Details**

See `update_rules` vignette for a mathematical representation.

$$conversion_i \text{ Bernoulli}(\phi)$$

$$\phi \text{ Beta}(\alpha, \beta)$$

Conversion Rate is sampled from a Beta distribution with a Binomial likelihood of an individual converting.

**Value**

`input_df` with 2 new nested columns ‘`beta_params`’ and ‘`samples`’

---

sample\_cpa

*Sample Cost Per Activation (CPA)*

---

**Description**

Adds 3 new nested columns to the `input_df`: ‘`beta_params`’, ‘`gamma_params`’, and ‘`samples`’. ‘`beta_params`’ and ‘`gamma_params`’ in each row should be a tibble of length 2 ( $\alpha$  and  $\beta$  parameters and  $k$  and  $\theta$  parameters) ‘`samples`’ in each row should be a tibble of length ‘`n_samples`’

**Usage**

```
sample_cpa(input_df, priors, n_samples = 50000)
```

**Arguments**

`input_df` Dataframe containing `option_name` (str), `sum_conversions` (dbl), `sum_cost` (dbl), and `sum_clicks` (dbl).

`priors` Optional list of priors `alpha0`, `beta0` for Beta and `k0`, `theta0` for Gamma. Default  $Beta(1, 1)$  and  $Gamma(1, 250)$  will be use otherwise.

`n_samples` Optional integer value. Defaults to 50,000 samples.



**Details**

See `update_rules` vignette for a mathematical representation. This is a combination of a Beta-Bernoulli update and a Gamma-Exponential update.

$$conversion_i \sim \text{Bernoulli}(\phi)$$

$$cpc_i \sim \text{Exponential}(\lambda)$$

$$\phi \sim \text{Beta}(\alpha, \beta)$$

$$\lambda \sim \text{Gamma}(k, \theta)$$

$$cpa_i \sim 1/(\text{Bernoulli}(\phi) * \text{Exponential}(\lambda))$$

$$\text{averageCPA} \sim 1/(\phi\lambda)$$

Conversion Rate is sampled from a Beta distribution with a Binomial likelihood of an individual converting.

Average CPC is sampled from a Gamma distribution with an Exponential likelihood of an individual cost.

**Value**

input\_df with 3 new nested columns ‘beta\_params’, ‘gamma\_params’, and ‘samples’

---

sample\_cpc

*Sample Cost Per Click*

---

**Description**

Adds 2 new nested columns to the input\_df: ‘gamma\_params’ and ‘samples’. ‘gamma\_params’ in each row should be a tibble of length 2 ( $k$  and  $\theta$  parameters) ‘samples’ in each row should be a tibble of length ‘n\_samples’

**Usage**

```
sample_cpc(input_df, priors, n_samples = 50000)
```

**Arguments**

input_df	Dataframe containing option_name (str), sum_clicks (dbl), sum_cost (dbl).
priors	Optional list of priors k0, theta0 for Gamma. Default $\text{Gamma}(1, 250)$ will be use otherwise.
n_samples	Optional integer value. Defaults to 50,000 samples.

**Details**

See `update_rules` vignette for a mathematical representation.

$$cpc_i \sim \text{Exponential}(\lambda)$$

$$\lambda \sim \text{Gamma}(k, \theta)$$

Average CPC is sampled from a Gamma distribution with an Exponential likelihood of an individual cost.

**Value**

`input_df` with 2 new nested columns ‘`gamma_params`’ and ‘`samples`’

---

<code>sample_ctr</code>	<i>Sample Click Through Rate</i>
-------------------------	----------------------------------

---

**Description**

This is an alias for `sample_conv_rate` with 2 different input columns. This function calculates posterior samples of  $CTR = \text{clicks}/\text{impressions}$ . Adds 2 new nested columns to the `input_df`: ‘`beta_params`’ and ‘`samples`’. ‘`beta_params`’ in each row should be a tibble of length 2 ( $\alpha$  and  $\beta$  parameters) ‘`samples`’ in each row should be a tibble of length ‘`n_samples`’

**Usage**

```
sample_ctr(input_df, priors, n_samples = 50000)
```

**Arguments**

<code>input_df</code>	Dataframe containing <code>option_name</code> (str), <code>sum_clicks</code> (dbl), and <code>sum_impressions</code> (dbl).
<code>priors</code>	Optional list of priors <code>alpha0</code> and <code>beta0</code> . Default $Beta(1, 1)$ will be use otherwise.
<code>n_samples</code>	Optional integer value. Defaults to 50,000 samples.

**Details**

See `update_rules` vignette for a mathematical representation.

$$click_i \sim \text{Bernoulli}(\phi)$$

$$\phi \sim \text{Beta}(\alpha, \beta)$$

Click Through Rate is sampled from a Beta distribution with a Binomial likelihood of an individual Clicking

**Value**

`input_df` with 2 new nested columns ‘`beta_params`’ and ‘`samples`’

---

sample\_from\_posterior *Sample From Posterior*

---

## Description

Selects which function to use to sample from the posterior distribution

## Usage

```
sample_from_posterior(  
  input_df,  
  distribution,  
  priors = list(),  
  n_samples = 50000  
)
```

## Arguments

input_df	Dataframe containing option_name (str) and various other columns depending on the distribution type. See vignette for more details.
distribution	String of the distribution name
priors	Optional list of priors. Defaults will be use otherwise.
n_samples	Optional integer value. Defaults to 50,000 samples.

## Value

A tibble with 2 columns: option\_name (chr) and samples (dbl) [long form data].

## Examples

```
input_df <- tibble::tibble(  
  option_name = c("A", "B"),  
  sum_clicks = c(1000, 1000),  
  sum_conversions = c(100, 120),  
  sum_sessions = c(1000, 1000),  
  sum_revenue = c(1000, 1500)  
)  
sample_from_posterior(input_df, "conversion_rate")  
sample_from_posterior(input_df, "rev_per_session")
```

---

sample\_multi\_rev\_per\_session  
*Sample Multiple Revenue Per Session*

---

### Description

Adds 5 new nested columns to the input\_df: 'dirichlet\_params', 'gamma\_params\_A', 'gamma\_params\_B', and 'samples'. This samples from multiple revenue per session distributions at once.

### Usage

```
sample_multi_rev_per_session(input_df, priors, n_samples = 50000)
```

### Arguments

input_df	Dataframe containing option_name (str), sum_conversions (dbl), sum_sessions (dbl), sum_revenue (dbl), sum_conversion_2 (dbl), sum_sessions_2 (dbl), sum_revenue_2 (dbl).
priors	Optional list of priors alpha0 and beta0. Default $Beta(1, 1)$ will be use otherwise.
n_samples	Optional integer value. Defaults to 50,000 samples.

### Details

See update\_rules vignette for a mathematical representation.

$$conversion_i \text{ MultiNomial}(\phi_1, \phi_2, \dots, \phi_k)$$

$$\phi_k \text{ Dirichlet}(\alpha, \beta)$$

Conversion Rate is sampled from a Dirichlet distribution with a Multinomial likelihood of an individual converting.

### Value

input\_df with 4 new nested columns 'dirichlet\_params', 'gamma\_params\_A', 'gamma\_params\_B', and 'samples'. 'samples' in each row should be a tibble of length 'n\_samples'.

---

 sample\_page\_views\_per\_session

*Sample Page Views Per Session (Visit)*


---

## Description

Adds 2 new nested columns to the input\_df: ‘gamma\_params‘ and ‘samples‘ ‘gamma\_params‘ in each row should be a tibble of length 2 ( $k$  and  $\theta$  parameters) ‘samples‘ in each row should be a tibble of length ‘n\_samples‘

## Usage

```
sample_page_views_per_session(input_df, priors, n_samples = 50000)
```

## Arguments

input_df	Dataframe containing option_name (str), sum_sessions (dbl), and sum_page_views_per_session (dbl).
priors	Optional list of priors $k_0$ and $\theta_0$ . Default $Gamma(1, 250)$ will be use otherwise. $Gamma(1, 1)$ might also be a good choice for this distribution if you only have a few page views per session.
n_samples	Optional integer value. Defaults to 50,000 samples.

## Details

See update\_rules vignette for a mathematical representation.

$$page\_views_i \sim Poisson(\lambda)$$

$$\lambda \sim Gamma(k, \theta)$$

Page Views Per Visit is sampled from a Gamma distribution with a Poisson likelihood of an individual having  $n$  page views by the end of their session.

This is not always the case, so verify your data follows the shape of an Poisson distribution before using this.

## Value

input\_df with 2 new nested columns ‘gamma\_params‘ and ‘samples‘

---

sample\_response\_rate    *Sample Response Rate*

---

### Description

This is an alias for `sample_conv_rate` with a different input column. Adds 2 new nested columns to the `input_df`: `'beta_params'` and `'samples'` `'beta_params'` in each row should be a tibble of length 2 ( $\alpha$  and  $\beta$  parameters) `'samples'` in each row should be a tibble of length `'n_samples'`

### Usage

```
sample_response_rate(input_df, priors, n_samples = 50000)
```

### Arguments

<code>input_df</code>	Dataframe containing <code>option_name</code> (str), <code>sum_conversions</code> (dbl), and <code>sum_sessions</code> (dbl).
<code>priors</code>	Optional list of priors <code>alpha0</code> and <code>beta0</code> . Default <code>Beta(1, 1)</code> will be use otherwise.
<code>n_samples</code>	Optional integer value. Defaults to 50,000 samples.

### Details

See `update_rules` vignette for a mathematical representation.

$$conversion_i \sim \text{Bernoulli}(\phi)$$

$$\phi \sim \text{Beta}(\alpha, \beta)$$

Response Rate is sampled from a Beta distribution with a Binomial likelihood of an individual converting.

### Value

`input_df` with 2 new nested columns `'beta_params'` and `'samples'`

---

sample\_rev\_per\_session  
*Sample Rev Per Session*

---

### Description

Adds 3 new nested columns to the `input_df`: `'beta_params'`, `'gamma_params'`, and `'samples'` `'beta_params'` and `'gamma_params'` in each row should be a tibble of length 2 ( $\alpha$  and  $\beta$  parameters and  $k$  and  $\theta$  parameters) `'samples'` in each row should be a tibble of length `'n_samples'`

**Usage**

```
sample_rev_per_session(input_df, priors, n_samples = 50000)
```

**Arguments**

input_df	Dataframe containing option_name (str), sum_conversions (dbl), sum_revenue (dbl), and sum_clicks (dbl).
priors	Optional list of priors alpha0, beta0 for Beta and k0, theta0 for Gamma. Default <i>Beta</i> (1, 1) and <i>Gamma</i> (1, 250) will be use otherwise.
n_samples	Optional integer value. Defaults to 50,000 samples.

**Details**

See update\_rules vignette for a mathematical representation.

$$RevPerSession = RevPerOrder * OrdersPerClick$$

This is a combination of a Beta-Bernoulli update and a Gamma-Exponential update.

$$conversion_i \text{ Bernoulli}(\phi)$$

$$revenue_i \text{ Exponential}(\lambda)$$

$$\phi \text{ Beta}(\alpha, \beta)$$

$$\lambda \text{ Gamma}(k, \theta)$$

$$revenue_i \text{ Bernoulli}(\phi) * \text{Exponential}(\lambda)^{-1}$$

$$RevPerSession \phi/\lambda$$

Conversion Rate is sampled from a Beta distribution with a Binomial likelihood of an individual converting.

Average Rev Per Order is sampled from a Gamma distribution with an Exponential likelihood of Revenue from an individual order. This function makes sense to use if there is a distribution of possible revenue values that can be produced from a single order or conversion.

**Value**

input\_df with 3 new nested columns 'beta\_params', 'gamma\_params', and 'samples'

---

sample\_session\_duration

*Sample Session Duration*

---

### Description

Adds 2 new nested columns to the input\_df: ‘gamma\_params‘ and ‘samples‘ ‘gamma\_params‘ in each row should be a tibble of length 2 ( $k$  and  $\theta$  parameters) ‘samples‘ in each row should be a tibble of length ‘n\_samples‘

### Usage

```
sample_session_duration(input_df, priors, n_samples = 50000)
```

### Arguments

input_df	Dataframe containing option_name (str), sum_sessions (dbl), and sum_duration (dbl).
priors	Optional list of priors $k_0$ and $\theta_0$ . Default $Gamma(1, 250)$ will be use otherwise.
n_samples	Optional integer value. Defaults to 50,000 samples.

### Details

See update\_rules vignette for a mathematical representation.

$$duration_i \sim Exponential(\lambda)$$

$$\lambda \sim Gamma(k, \theta)$$

Session Duration is sampled from a Gamma distribution with a Exponential likelihood of an individual leaving the site or ending a session at time  $t$ .

This is not always the case, so verify your data follows the shape of an exponential distribution before using this.

### Value

input\_df with 2 new nested columns ‘gamma\_params‘ and ‘samples‘



---

sample_total_cm	<i>Sample Total CM (Given Impression Count)</i>
-----------------	---

---

### Description

Adds 4 new nested columns to the input\_df: 'beta\_params\_ctr', 'beta\_params\_conv', 'gamma\_params\_rev', 'gamma\_params\_cost' and 'samples'.

### Usage

```
sample_total_cm(input_df, priors, n_samples = 50000)
```

### Arguments

input_df	Dataframe containing option_name (str), sum_conversions (dbl), sum_revenue (dbl), and sum_clicks (dbl).
priors	Optional list of priors alpha0, beta0 for Beta, k0, theta0 for Gamma Inverse Revenue, and k01, theta01 for Gamma Cost (uses alternate priors so they can be different from Revenue). Default <i>Beta</i> (1,1) and <i>Gamma</i> (1,250) will be use otherwise.
n_samples	Optional integer value. Defaults to 50,000 samples.

### Details

'beta\_params' and 'gamma\_params' in each row should be a tibble of length 2 ( $\alpha$  and  $\beta$  params and  $k$  and  $\theta$  params). 'samples' in each row should be a tibble of length 'n\_samples'.

One assumption in this model is that sum\_impressions is not stochastic. This assumes that Clicks are stochastically generated from a set number of Impressions. It does not require that the number of impressions are equal on either side. Generally this assumption holds true in marketing tests where traffic is split 50/50 and very little variance is observed in the number of impressions on either side.

See update\_rules vignette for a mathematical representation.

$$TotalCM = Impr * ExpectedCTR * (RevPerOrder * OrdersPerClick - ExpectedCPC)$$

### Value

input\_df with 5 new nested columns 'beta\_params\_conv', 'beta\_params\_ctr', 'gamma\_params\_rev', 'gamma\_params\_cost', and 'samples'

---

update_beta	<i>Update Beta</i>
-------------	--------------------

---

**Description**

Updates Beta Distribution with the Beta-Bernoulli conjugate prior update rule

**Usage**

```
update_beta(alpha, beta, priors = list())
```

**Arguments**

alpha	Double value for alpha (count of successes). Must be 0 or greater.
beta	Double value for beta (count of failures). Must be 0 or greater.
priors	An optional list object that contains alpha0 and beta0. Otherwise the function with use Beta(1,1) as the prior distribution.

**Value**

A tibble object that contains 'alpha' and 'beta'

**Examples**

```
update_beta(alpha = 1, beta = 5, priors = list(alpha0 = 2, beta0 = 2))
update_beta(alpha = 20000, beta = 50000)
```

---

update_dirichlet	<i>Update Dirichlet Distribution</i>
------------------	--------------------------------------

---

**Description**

This function updates the Dirichlet distribution with the Dirichlet-Multinomial conjugate prior update rule.

**Usage**

```
update_dirichlet(alpha_0, alpha_1, alpha_2, priors = list())
```

**Arguments**

alpha_0	Double value for alpha_0 (count of failures). Must be 0 or greater.
alpha_1	Double value for alpha_1 (count of successes side 1). Must be 0 or greater.
alpha_2	Double value for alpha_2 (count of successes side 2). Must be 0 or greater.
priors	An optional list object that contains alpha00, alpha01, and alpha02. Otherwise the function with use <i>Dirichlet</i> (1, 1, 1) as the prior distribution.

**Details**

TODO: This function currently only works in 3 dimensions. Should be extended into N dimensions in the future. Can use ... notation.

**Value**

tibble with columns alpha\_0, alpha\_1, and alpha\_2

**Examples**

```
update_dirichlet(alpha_0 = 20, alpha_1 = 5, alpha_2 = 2)
sample_priors_list <- list(alpha00 = 2, alpha01 = 3, alpha02 = 5)
update_dirichlet(alpha_0 = 20, alpha_1 = 5, alpha_2 = 2, priors = sample_priors_list)
```

---

update_gamma	<i>Update Gamma</i>
--------------	---------------------

---

**Description**

Updates Gamma Distribution with the Gamma-Exponential conjugate prior update rule. Parameterized by  $k$  and  $\theta$  (not  $\alpha, \beta$ )

**Usage**

```
update_gamma(k, theta, priors = list(), alternate_priors = FALSE)
```

**Arguments**

k	Double value for $k$ (total revenue generating events). Must be 0 or greater.
theta	Double value for $\theta$ (sum of revenue). Must be 0 or greater.
priors	An optional list object that contains k0 and theta0. Otherwise the function will use $Gamma(1, 250)$ as the prior distribution. If a second gamma distribution is used k01 and theta01 can be defined as separate priors when alternate_priors is set to TRUE.
alternate_priors	Boolean Defaults to FALSE. Allows a user to specify alternate prior names so the same prior isn't required when multiple gamma distributions are used.

**Value**

A list object that contains 'k' and 'theta'

**Examples**

```
update_gamma(k = 1, theta = 100, priors = list(k0 = 2, theta0 = 1000))
update_gamma(k = 10, theta = 200)
```

---

validate\_data\_values *Validate Data Values*

---

### Description

Validates data values are all greater than 0.

### Usage

```
validate_data_values(data_values)
```

### Arguments

data\_values List of named data values

### Value

None

---

validate\_input\_column *Validate Input Column*

---

### Description

Validates the input column exists in the dataframe, is of the correct type, and that all values are greater than or equal to 0.

### Usage

```
validate_input_column(column_name, input_df, greater_than_zero = TRUE)
```

### Arguments

column\_name String value of the column name

input\_df Dataframe containing option\_name (str) and various other columns depending on the distribution type. See vignette for more details.

greater\_than\_zero Boolean: Do all values in the column have to be greater than zero?

### Value

None

---

validate\_input\_df      *Validate Input DataFrame*

---

**Description**

Validates the input dataframe has the correct type, correct required column names, that the distribution is valid, that the column types are correct, and that the column values are greater than or equal to 0 when they are numeric.

**Usage**

```
validate_input_df(input_df, distribution)
```

**Arguments**

input\_df      Dataframe containing option\_name (str) and various other columns depending on the distribution type. See vignette for more details.

distribution      String of the distribution name

**Value**

Bool TRUE if all checks pass.

**Examples**

```
input_df <- tibble::tibble(  
  option_name = c("A", "B"),  
  sum_clicks = c(1000, 1000),  
  sum_conversions = c(100, 120)  
)  
validate_input_df(input_df, "conversion_rate")
```

---

validate\_posterior\_samples      *Validate Posterior Samples Dataframe*

---

**Description**

Function fails if posterior is not shaped correctly.

**Usage**

```
validate_posterior_samples(posterior_samples)
```

**Arguments**

posterior\_samples  
Tibble of data in long form with 2 columns 'option\_name' and 'samples'

**Value**

None

---

validate_priors	<i>Validate Priors</i>
-----------------	------------------------

---

**Description**

Validates list of priors against a vector of valid priors and if the values are not valid, default priors are returned.

**Usage**

```
validate_priors(priors, valid_priors, default_priors)
```

**Arguments**

priors           List of named priors with double values.  
valid\_priors    A character vector of valid prior names.  
default\_priors  A list of default priors for the distribution.

**Value**

A named list of valid priors for the distribution.

---

validate_wrt_option	<i>Validate With Respect To Option</i>
---------------------	--

---

**Description**

Verify that the option provided is in the poster\_samples dataframe 'option\_name' column. Raises error if not TRUE

**Usage**

```
validate_wrt_option(wrt_option, posterior_samples)
```

**Arguments**

wrt\_option      string name of the option

posterior\_samples

Tibble returned from `sample_from_posterior` with 3 columns 'option\_name', 'samples', and 'sample\_id'.

**Value**

None

# Index

calculate\_multi\_rev\_per\_session, 3  
calculate\_total\_cm, 3

estimate\_all\_values, 4  
estimate\_lift, 5  
estimate\_lift\_vs\_baseline, 6  
estimate\_loss, 7  
estimate\_value\_remaining, 8  
estimate\_win\_prob, 9  
estimate\_win\_prob\_given\_posterior, 10  
estimate\_win\_prob\_vs\_baseline, 10  
estimate\_win\_prob\_vs\_baseline\_given\_posterior, 11

find\_best\_option, 12

impute\_missing\_options, 13  
is\_prior\_valid, 13  
is\_winner\_max, 14

rdirichlet, 14

sample\_cm\_per\_click, 15  
sample\_conv\_rate, 15  
sample\_cpa, 16  
sample\_cpc, 17  
sample\_ctr, 18  
sample\_from\_posterior, 19  
sample\_multi\_rev\_per\_session, 20  
sample\_page\_views\_per\_session, 21  
sample\_response\_rate, 22  
sample\_rev\_per\_session, 22  
sample\_session\_duration, 24  
sample\_total\_cm, 25

update\_beta, 26  
update\_dirichlet, 26  
update\_gamma, 27

validate\_data\_values, 28  
validate\_input\_column, 28  
validate\_input\_df, 29  
validate\_posterior\_samples, 29  
validate\_priors, 30  
validate\_wrt\_option, 30