The cbcTools Package: Tools for Designing and Testing Choice-Based Conjoint Surveys in R

John Paul Helveston, Ph.D.[[1]](#footnote-1)

While many open source packages and applications exist for estimating choice models using data from choice-based conjoint experiments, few packages exist for generating and evaluating experiment designs prior to collecting data. Furthermore, existing tools for designing choice-based conjoint survey experiments often focus on optimizing the design for statistical power under ideal conditions, but they rarely provide guidance on important design decisions for less ideal conditions, such as when preference heterogeneity or strong interactions between certain attributes may be expected in respondent choices. The cbcTools R package was developed to provide researchers tools for creating and evaluating experiment designs and sample size requirements under a variety of different conditions prior to fielding an experiment. The package contains functions for generating experiment designs, examining attribute balance and overlap, simulating choice data, and conducting power analyses. The package data format matches that of experiment designs exported from Sawtooth Software, enabling a smooth integration with the Sawtooth Software workflow. Detailed package documentation can be found at <https://jhelvy.github.io/cbcTools/>.

# Introduction

Designing a choice-based conjoint survey is almost never a simple, straightforward process. Designers must consider multiple trade offs between design parameters (e.g., which attributes and levels to include, how many choice questions to ask each respondent, and how many alternatives per choice question) and the design outcomes in terms of the user experience and the statistical power available for identifying effects. The process is typically highly iterative (R. M. Johnson and Orme 2002).

As a quick example, consider a simple conjoint experiment about cars with just two attributes with three levels each:

* **Price**: $20,000, 40,000, & 100,000
* **Brand**: GM, BMW, & Ferrari

A common starting point is to first generate the full factorial set of all possible combinations of brand and price. These profiles can then be selected into a survey design from which respondents will make choices. Without any prior knowledge or experience about car prices and brands, a designer may simply choose to create a survey design by randomly selecting profiles from the full factorial set of profiles. This often results in a less efficient design, but it is a starting point for illustrating trade offs in making a design.

Once a survey design is created, one of the first things designers examine is the count of how often each level of each attribute is shown, or the design “balance” (Hauser and Toubia 2005; Huber and Zwerina 1996). The table below shows an example of the counts from a design with just 9 choice sets of 3 alternatives per question, each of which were randomly chosen from the full factorial set of profiles:

**Table 1: Individual and pairwise counts of attributes.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Price*:** | $20,000 | $40,000 | $100,000 |
| ***Brand*** |  | 9 | 9 | 9 |
| GM | 10 | 3 | 0 | 7 |
| BMW | 11 | 4 | 5 | 2 |
| Ferrari | 6 | 2 | 4 | 0 |

Based on the balance alone, it is clear that this design has several problems. First, while the price levels are perfectly balanced (each level is shown 9 times), the brand levels are not – GM and BMW are shown 10 and 11 times, respectively, whereas Ferrari is only shown 6 times. And the pairwise counts are particularly troubling. The Ferrari brand is only shown with a price of $20,000 and $40,000 and never at the $100,000 level (the most logical level for a Ferrari!). Likewise, GM brand is shown with a price of $100,000 in 7 out of 10 times and never with a price of $40,000.

This rather poor design is a common outcome when generating a design by randomly selecting profiles from the full factorial with only a limited number of choice sets. One way to improve the outcome is to simply increase the number of choice sets used, which often results in a much better balance (Kuhfeld 2002). For example, if we increase the number of choice sets from 9 to 90, we obtain the following counts:

**Table 2: Individual and pairwise counts of attributes.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Price*:** | $20,000 | $40,000 | $100,000 |
| ***Brand*** |  | 91 | 84 | 95 |
| GM | 92 | 31 | 31 | 30 |
| BMW | 80 | 25 | 25 | 30 |
| Ferrari | 98 | 35 | 30 | 35 |

This design has a much better balance across the attribute levels than the one created from just 9 choice sets, but it too has issues. Consider, for example, that about 1/3 of the time the Ferrari brand is shown it has a price of just $20,000. This is obviously an unrealistic scenario, and if a user saw such a profile multiple times they may not take the choice exercise seriously.

As this simple example illustrates, randomly selecting profiles (even with a larger number of choice sets) can still produce a flawed design[[2]](#footnote-2). An alternative approach is to attempt to identify a statistically optimal design, which is now a rather well-established practice in the literature on discrete choice experiments (F. R. Johnson et al. 2013). Optimal designs maximize the expected Fisher information, which depends on the parameters of the assumed choice model. As a result, a design’s efficiency is related to how accuracy the prior guess is to the true parameters. Using a “Bayesian” D-efficient design where the designer can specify prior expected coefficients for each attribute has been proven to outperform other design approaches when sufficient information on the utility model is known (Kessels, Goos, and Vandebroek 2006; Walker et al. 2018).

**Table 3: Example conjoint attributes and levels.**

|  |  |  |
| --- | --- | --- |
| Attribute | Level | Prior |
| Price | $20,000 | 0 |
|  | $40,000 | -1 |
|  | $100,000 | -4 |
| Brand | GM | 0 |
|  | BMW | 1 |
|  | Ferrari | 2 |

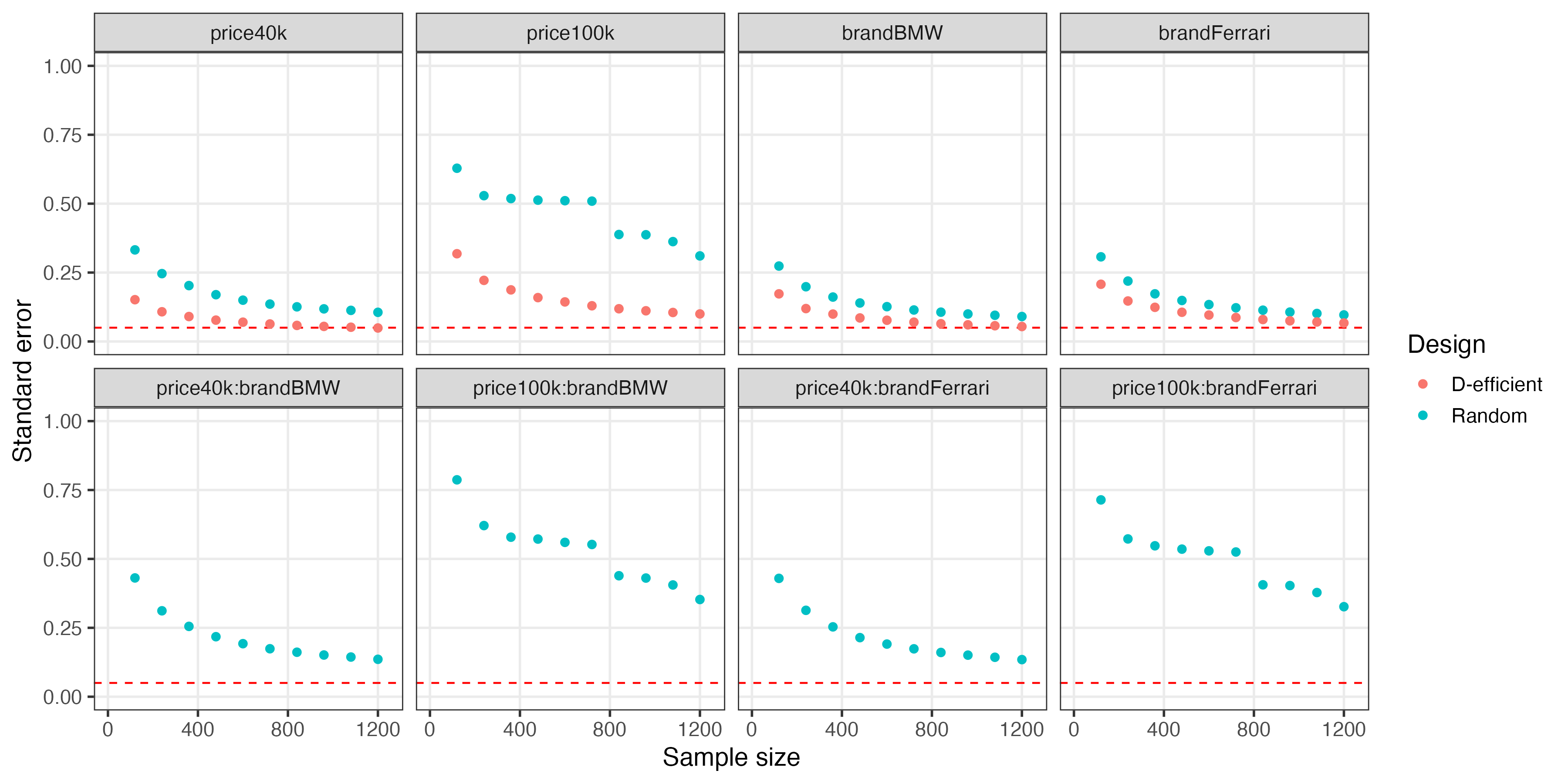
Using a Bayesian D-efficient design with 90 choice sets and the prior utilities in Table 3, the counts become much more reasonable (see Table 4). Now the individual attribute counts are relatively balanced, and the pairwise combinations of attributes are much more plausible. For example, the GM brand is only shown at price levels of $20,000 and $40,000, and about half of the time Ferrari is shown at the $100,000 price level.

**Table 4: Individual and pairwise counts of attributes.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Price*:** | $20,000 | $40,000 | $100,000 |
| ***Brand*** |  | 97 | 93 | 78 |
| GM | 93 | 52 | 41 | 0 |
| BMW | 90 | 30 | 30 | 30 |
| Ferrari | 86 | 15 | 22 | 49 |

Of course, even Bayesian D-efficient designs have their downsides. In particular, they can become problematic if there are significant interaction effects between any of the attributes that the designer did not anticipate.

Figure 1 highlights this trade off. The plots show the standard errors of each coefficients from the same model estimated using increasing subsets of two simulated data sets: a randomized design and a Bayesian D-efficient design. In the first row, it is clear that the D-efficient design produces lower standard errors for the same sample size compared to the randomized design. However, the second row shows the standard errors on interaction effects, which are not identifiable using the D-efficient design. This is because interaction effects can become confounded with main effects in D-efficient designs where the designer did not consider interactions in their priors. Thus, the accuracy of the designer’s prior model used in obtaining a D-efficient design is an important factor in the overall performance of the design. If little to no prior information is known and interactions may be present, a fully randomized design may be a better choice, though of course this would require larger sample sizes to obtain significant parameter estimates.

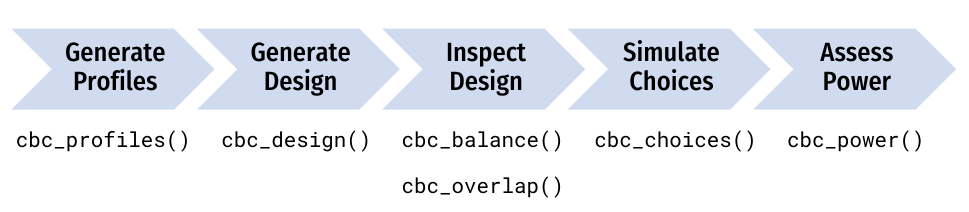


**Figure 1**: Standard errors with increasing sample size for a multinomial logit model with interaction effects estimated using a randomized design versus a Bayesian D-efficient design.

# The cbcTools Package

As the previous example illustrates, designing a choice-based conjoint experiment is an iterative process requiring careful consideration of a variety of factors. The cbcTools package was designed as an open source tool to help designers navigate this process and to understand the impacts that different design decisions could have on the outcomes of a choice-based conjoint experiment prior to fielding the survey (Helveston 2022).

The package provides a set of functions (each starting with cbc\_) for designing surveys and conducting power analyses for choice-based conjoint survey experiments in R. Each function is associated with a process in the typical design workflow (see Figure 2). The rest of this paper explains each of these steps using a detailed example.



**Figure 2**: Diagram of the choice-based conjoint survey design process and associated package function(s).

## Generating profiles

One of the first steps in designing a conjoint experiment is to define the attributes and levels and then generate all of the profiles for each combination of those attributes and levels, often referred to as the “full factorial” design. For example, consider designing a conjoint experiment about apples with the following three attributes: price, type, and freshness. The full factorial set of profiles for these attributes can be obtained using the cbc\_profiles() function:

profiles <- cbc\_profiles(  
 price = seq(1, 4, 0.5), # $ per pound  
 type = c('Fuji', 'Gala', 'Honeycrisp'),  
 freshness = c('Poor', 'Average', 'Excellent')  
)  
  
head(profiles)

#> profileID price type freshness  
#> 1 1 1.0 Fuji Poor  
#> 2 2 1.5 Fuji Poor  
#> 3 3 2.0 Fuji Poor  
#> 4 4 2.5 Fuji Poor  
#> 5 5 3.0 Fuji Poor  
#> 6 6 3.5 Fuji Poor

tail(profiles)

#> profileID price type freshness  
#> 58 58 1.5 Honeycrisp Excellent  
#> 59 59 2.0 Honeycrisp Excellent  
#> 60 60 2.5 Honeycrisp Excellent  
#> 61 61 3.0 Honeycrisp Excellent  
#> 62 62 3.5 Honeycrisp Excellent  
#> 63 63 4.0 Honeycrisp Excellent

The resulting data frame, profiles, has 63 rows, each defining a unique profile. Depending on the context of the survey, some profiles may need to be eliminated (e.g., some profile combinations may be illogical or unrealistic).[[3]](#footnote-3) To do so, each level of an attribute can be defined as a list defining those constraints. In the example below, the type attribute has constraints such that only certain price levels will be shown for each level. In addition, for the "Honeycrisp" level of the type attribute, only two of the three freshness levels are included: "Excellent" and "Average". Note that both the other attributes (price and freshness) should contain all of the possible levels. With these constraints, only 30 profiles are available compared to 63 without constraints.

profiles <- cbc\_profiles(  
 price = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),  
 freshness = c('Poor', 'Average', 'Excellent'),  
 type = list(  
 "Fuji" = list(  
 price = c(2, 2.5, 3)  
 ),  
 "Gala" = list(  
 price = c(1, 1.5, 2)  
 ),  
 "Honeycrisp" = list(  
 price = c(2.5, 3, 3.5, 4, 4.5, 5),  
 freshness = c("Average", "Excellent")  
 )  
 )  
)  
  
head(profiles)

#> profileID price freshness type  
#> 1 1 2.0 Poor Fuji  
#> 2 2 2.5 Poor Fuji  
#> 3 3 3.0 Poor Fuji  
#> 4 4 2.0 Average Fuji  
#> 5 5 2.5 Average Fuji  
#> 6 6 3.0 Average Fuji

tail(profiles)

#> profileID price freshness type  
#> 25 25 2.5 Excellent Honeycrisp  
#> 26 26 3.0 Excellent Honeycrisp  
#> 27 27 3.5 Excellent Honeycrisp  
#> 28 28 4.0 Excellent Honeycrisp  
#> 29 29 4.5 Excellent Honeycrisp  
#> 30 30 5.0 Excellent Honeycrisp

## Generating designs

### Randomized designs

Once a set of profiles is obtained, a randomized conjoint survey can then be generated using the cbc\_design() function:

design <- cbc\_design(  
 profiles = profiles,  
 n\_resp = 900, # Number of respondents  
 n\_alts = 3, # Number of alternatives per question  
 n\_q = 6 # Number of questions per respondent  
)  
  
dim(design) # View dimensions

#> [1] 16200 8

head(design) # Preview first 6 rows

#> respID qID altID obsID profileID price type freshness  
#> 1 1 1 1 1 52 2.0 Gala Excellent  
#> 2 1 1 2 1 34 3.5 Gala Average  
#> 3 1 1 3 1 25 2.5 Fuji Average  
#> 4 1 2 1 2 10 2.0 Gala Poor  
#> 5 1 2 2 2 11 2.5 Gala Poor  
#> 6 1 2 3 2 36 1.0 Honeycrisp Average

For now, the cbc\_design() function only generates a randomized design. Other packages, such as the [idefix](https://github.com/traets/idefix) package, are able to generate other types of designs, including Bayesian D-efficient designs, and future versions of the cbcTools package will add a wrapper around the idefix package to integrate some of these features. The randomized design simply samples from the set of profiles while ensuring that no two profiles are the same in any choice question.

The structure of the resulting design data frame is such that each row specifies the levels shown for one alternative in a choice question. In addition to the attribute levels, the design data frame also includes the following variables for identifying different aspects of the survey:

* respID: Identifies each survey respondent.
* qID: Identifies the choice question answered by the respondent.
* altID:Identifies the alternative in any one choice observation.
* obsID: Identifies each unique choice observation across all respondents.
* profileID: Identifies the profile in profiles.

### Labeled designs (a.k.a. “alternative-specific” designs)

A “labeled” design (also known as an “alternative-specific” design) can also be generated. In labeled designs, the levels of one attribute are used as the labels for each alternative. To do so, the label argument can be set to one of the attributes. This by definition also sets the number of alternatives in each question to the number of levels in the chosen attribute, so the n\_alts argument is overridden. In the example below, the type attribute is used as the label:

design\_labeled <- cbc\_design(  
 profiles = profiles,  
 n\_resp = 900, # Number of respondents  
 n\_alts = 3, # Number of alternatives per question  
 n\_q = 6, # Number of questions per respondent  
 label = "type" # Set the "type" attribute as the label  
)  
  
dim(design\_labeled)

#> [1] 16200 8

head(design\_labeled)

#> respID qID altID obsID profileID price type freshness  
#> 1 1 1 1 1 44 1.5 Fuji Excellent  
#> 2 1 1 2 1 54 3.0 Gala Excellent  
#> 3 1 1 3 1 42 4.0 Honeycrisp Average  
#> 4 1 2 1 2 44 1.5 Fuji Excellent  
#> 5 1 2 2 2 29 1.0 Gala Average  
#> 6 1 2 3 2 17 2.0 Honeycrisp Poor

In the above example, the type attribute is now fixed to be the same order for every choice question, ensuring that each level in the type attribute will always be shown in each choice question.

### Adding a “no choice” option (a.k.a. “outside good”)

Often times designers may wish to allow respondents to opt out from choosing any of the alternatives shown in any one choice question. Such a “no choice” (or “outside good”) option can be included by setting no\_choice = TRUE. If included, all categorical attributes will be dummy-coded to appropriately dummy-code the “no choice” alternative.

design\_nochoice <- cbc\_design(  
 profiles = profiles,  
 n\_resp = 900, # Number of respondents  
 n\_alts = 3, # Number of alternatives per question  
 n\_q = 6, # Number of questions per respondent  
 no\_choice = TRUE  
)  
  
dim(design\_nochoice)

#> [1] 21600 13

head(design\_nochoice)

#> respID qID altID obsID profileID price type\_Fuji type\_Gala type\_Honeycrisp  
#> 1 1 1 1 1 48 3.5 1 0 0  
#> 2 1 1 2 1 11 2.5 0 1 0  
#> 3 1 1 3 1 42 4.0 0 0 1  
#> 4 1 1 4 1 0 0.0 0 0 0  
#> 5 1 2 1 2 52 2.0 0 1 0  
#> 6 1 2 2 2 7 4.0 1 0 0  
#> freshness\_Poor freshness\_Average freshness\_Excellent no\_choice  
#> 1 0 0 1 0  
#> 2 1 0 0 0  
#> 3 0 1 0 0  
#> 4 0 0 0 1  
#> 5 0 0 1 0  
#> 6 1 0 0 0

## Inspecting designs

The package includes some functions to quickly inspect some basic metrics of a design. The cbc\_balance() function prints out a summary of the counts of each level for each attribute across all choice questions as well as the two-way counts across all pairs of attributes for a given design:

cbc\_balance(design)

#> ==============================  
#> price x type   
#>   
#> Fuji Gala Honeycrisp  
#> NA 5326 5425 5449  
#> 1 2318 757 777 784  
#> 1.5 2325 787 727 811  
#> 2 2314 776 791 747  
#> 2.5 2416 816 793 807  
#> 3 2278 717 787 774  
#> 3.5 2278 745 772 761  
#> 4 2271 728 778 765  
#>   
#> price x freshness   
#>   
#> Poor Average Excellent  
#> NA 5379 5435 5386  
#> 1 2318 746 772 800  
#> 1.5 2325 753 787 785  
#> 2 2314 755 788 771  
#> 2.5 2416 840 798 778  
#> 3 2278 788 747 743  
#> 3.5 2278 747 788 743  
#> 4 2271 750 755 766  
#>   
#> type x freshness   
#>   
#> Poor Average Excellent  
#> NA 5379 5435 5386  
#> Fuji 5326 1759 1765 1802  
#> Gala 5425 1787 1817 1821  
#> Honeycrisp 5449 1833 1853 1763

Similarly, the cbc\_overlap() function prints out a summary of the amount of “overlap” across attributes within the choice questions. For example, for each attribute, the count under "1" is the number of choice questions in which the same level was shown across all alternatives for that attribute (because there was only one level shown). Likewise, the count under "2" is the number of choice questions in which only two unique levels of that attribute were shown, and so on:

cbc\_overlap(design)

#> ==============================  
#> Counts of attribute overlap:  
#> (# of questions with N unique levels)  
#>   
#> price:  
#>   
#> 1 2 3   
#> 78 1812 3510   
#>   
#> type:  
#>   
#> 1 2 3   
#> 524 3627 1249   
#>   
#> freshness:  
#>   
#> 1 2 3   
#> 573 3587 1240

## Simulating choices

Choices for a given design can be simulated using the cbc\_choices() function. By default, random choices are simulated:

data <- cbc\_choices(  
 design = design,  
 obsID = "obsID"  
)  
  
head(data)

#> respID qID altID obsID profileID price type freshness choice  
#> 1 1 1 1 1 52 2.0 Gala Excellent 0  
#> 2 1 1 2 1 34 3.5 Gala Average 1  
#> 3 1 1 3 1 25 2.5 Fuji Average 0  
#> 4 1 2 1 2 10 2.0 Gala Poor 0  
#> 5 1 2 2 2 11 2.5 Gala Poor 1  
#> 6 1 2 3 2 36 1.0 Honeycrisp Average 0

Choices can also be simulated according to an assumed prior model. The default model used is a multinomial logit model with fixed parameters. In the example below, the choices are simulated using a utility model with the following parameters:

* 1 continuous parameter for price
* 2 categorical parameters for type ("Gala" and "Honeycrisp")
* 2 categorical parameters for freshness ("Average" and "Excellent")

Note that for categorical variables (type and freshness in this example), the first level defined when using cbc\_profiles() is set as the reference level. The example below defines the following utility model for simulating choices for each alternative *j*:

+

data <- cbc\_choices(  
 design = design,  
 obsID = "obsID",  
 priors = list(  
 price = 0.1,  
 type = c(0.1, 0.2),  
 freshness = c(0.1, 0.2)  
 )  
)

The prior model used for simulating choices can also include other, more complex models, such as models that include interaction terms or mixed logit models. For example, the example below is the same as the previous example but with an added interaction between price and type:

data <- cbc\_choices(  
 design = design,  
 obsID = "obsID",  
 priors = list(  
 price = 0.1,  
 type = c(0.1, 0.2),  
 freshness = c(0.1, 0.2),  
 `price\*type` = c(0.1, 0.5)  
 )  
)

To simulate choices according to a mixed logit model where parameters follow a normal or log-normal distribution across the population, the randN() and randLN() functions can be used inside the priors list. The example below models the type attribute with two random normal parameters using a vector of means (mean) and standard deviations (sd) for each level of type:

data <- cbc\_choices(  
 design = design,  
 obsID = "obsID",  
 priors = list(  
 price = 0.1,  
 type = randN(mean = c(0.1, 0.2), sd = c(1, 2)),  
 freshness = c(0.1, 0.2)  
 )  
)

## Analyzing power

The simulated choice data can be used to conduct a power analysis by estimating the same model multiple times with incrementally increasing sample sizes. As the sample size increases, the estimated coefficient standard errors will decrease (i.e. coefficient estimates become more precise), allowing the designer to identify the sample size required to achieve a desired level of precision.

The cbc\_power() function achieves this by partitioning the choice data into multiple sizes (defined by the nbreaks argument) and then estimating a user-defined choice model on each data subset. In the example below, 10 different sample sizes are used. All models are estimated using the [logitr](https://jhelvy.github.io/logitr) package (Helveston 2021), and any arguments for estimating models with the logitr package can be passed through the cbc\_power() function (see the logitr documentation for more details at <https://jhelvy.github.io/logitr/>)

power <- cbc\_power(  
 data = data,  
 pars = c("price", "type", "freshness"),  
 outcome = "choice",  
 obsID = "obsID",  
 nbreaks = 10,  
 n\_q = 6  
)  
  
head(power)

#> sampleSize coef est se  
#> 1 90 price -0.03180537 0.05266167  
#> 2 90 typeGala -0.15433384 0.13022498  
#> 3 90 typeHoneycrisp 0.04192555 0.12766024  
#> 4 90 freshnessAverage 0.04817604 0.12976462  
#> 5 90 freshnessExcellent -0.25984441 0.12922654  
#> 6 180 price -0.01600508 0.03742382

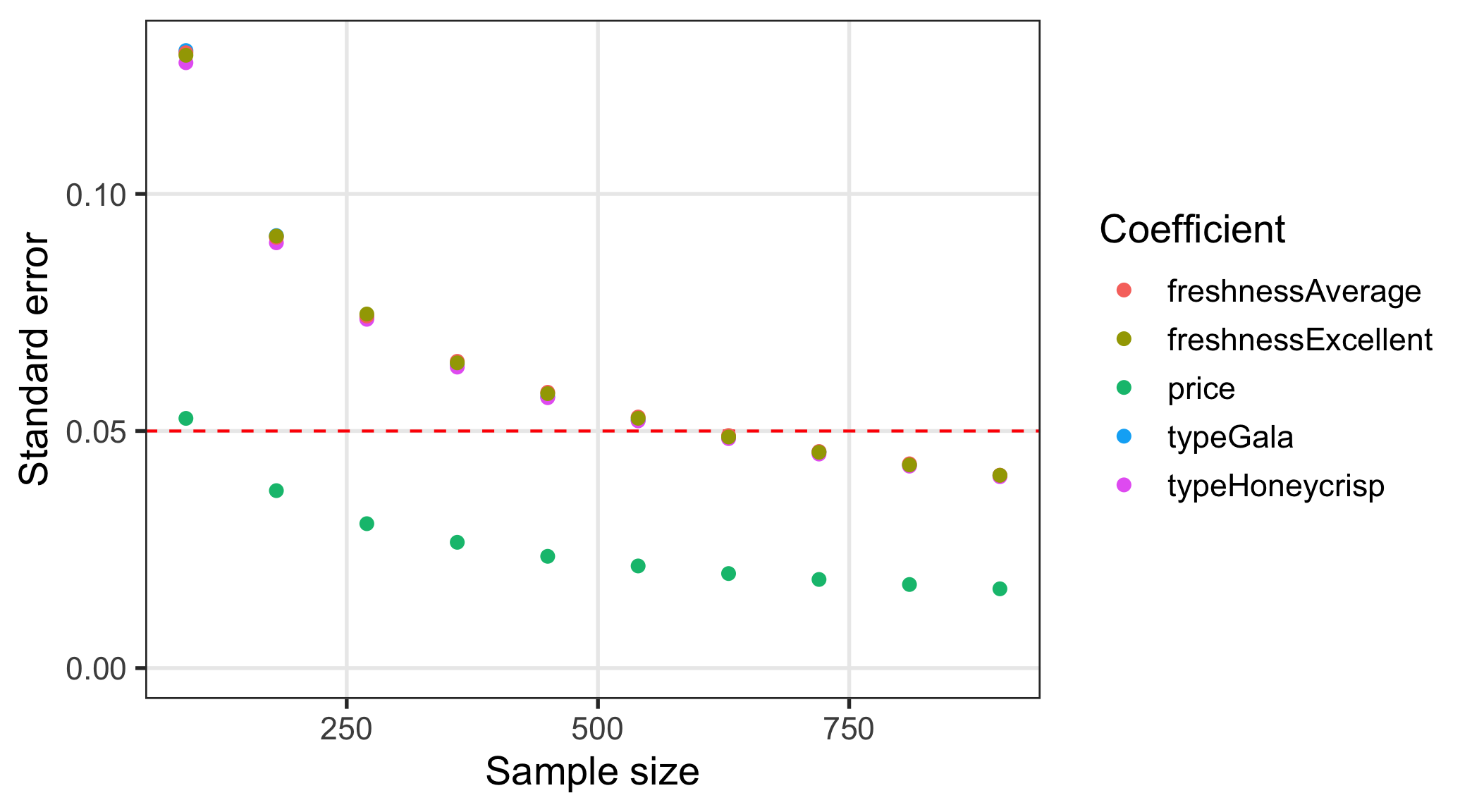
tail(power)

#> sampleSize coef est se  
#> 45 810 freshnessExcellent -0.017125538 0.04284571  
#> 46 900 price -0.021522096 0.01672266  
#> 47 900 typeGala -0.089114429 0.04061808  
#> 48 900 typeHoneycrisp -0.065052536 0.04036081  
#> 49 900 freshnessAverage -0.008200304 0.04068266  
#> 50 900 freshnessExcellent -0.031254919 0.04065947

The power data frame contains the coefficient estimates and standard errors for each sample size. In the example above, it is clear that the standard errors for a sample size of 900 are much lower than those for a sample size of just 90.

Visualizing the results of the power analysis can be particularly helpful for identifying sample size requirements. Since the cbc\_power() function returns a data frame in a “tidy” (or “long”) format (Wickham 2014), the results can be conveniently plotted using the popular ggplot2 package (Wickham, Chang, and Wickham 2016). A plot() method is already included in cbcTools to create a simple ggplot of the power curves:

plot(power)



**Figure 3:** Standard errors of each model coefficient with increasing sample size.

Of course, designers may be interested in aspects other than standard errors. By setting return\_models = TRUE, the cbc\_power() function will return a list of estimated models (one for each sample size increment), which can then be used to examine other model objects. The example below prints a summary of the last model in the list of returned models.

library(logitr)  
  
models <- cbc\_power(  
 data = data,  
 pars = c("price", "type", "freshness"),  
 outcome = "choice",  
 obsID = "obsID",  
 nbreaks = 10,  
 n\_q = 6,  
 return\_models = TRUE  
)  
  
summary(models[[10]])

#> =================================================  
#>   
#> Model estimated on: Tue Jun 07 15:59:19 2022   
#>   
#> Using logitr version: 0.6.0   
#>   
#> Call:  
#> FUN(data = X[[i]], outcome = ..1, obsID = ..2, pars = ..3, randPars = ..4,   
#> panelID = ..5, clusterID = ..6, robust = ..7, predict = ..8)  
#>   
#> Frequencies of alternatives:  
#> 1 2 3   
#> 0.33500 0.33333 0.33167   
#>   
#> Exit Status: 3, Optimization stopped because ftol\_rel or ftol\_abs was reached.  
#>   
#> Model Type: Multinomial Logit  
#> Model Space: Preference  
#> Model Run: 1 of 1  
#> Iterations: 7  
#> Elapsed Time: 0h:0m:0.04s  
#> Algorithm: NLOPT\_LD\_LBFGS  
#> Weights Used?: FALSE  
#> Robust? FALSE  
#>   
#> Model Coefficients:   
#> Estimate Std. Error z-value Pr(>|z|)   
#> price -0.0215221 0.0167227 -1.2870 0.19809   
#> typeGala -0.0891144 0.0406181 -2.1940 0.02824 \*  
#> typeHoneycrisp -0.0650525 0.0403608 -1.6118 0.10701   
#> freshnessAverage -0.0082003 0.0406827 -0.2016 0.84025   
#> freshnessExcellent -0.0312549 0.0406595 -0.7687 0.44207   
#> ---  
#> Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
#>   
#> Log-Likelihood: -5.928754e+03  
#> Null Log-Likelihood: -5.932506e+03  
#> AIC: 1.186751e+04  
#> BIC: 1.190048e+04  
#> McFadden R2: 6.324431e-04  
#> Adj McFadden R2: -2.103710e-04  
#> Number of Observations: 5.400000e+03

## Piping it all together!

One of the convenient features of how the package is written is that the object generated in each step is used as the first argument to the function for the next step. Thus, just like in the overall program diagram (see Figure 2), the functions can be piped together using either the Base R |> operator or the popular %>% operator from the magrittr package (Bache et al. 2022). The example below will generate the same power analysis plot as in Figure 3 by sequentially evaluating each of the main design steps.

cbc\_profiles(  
 price = seq(1, 4, 0.5), # $ per pound  
 type = c('Fuji', 'Gala', 'Honeycrisp'),  
 freshness = c('Poor', 'Average', 'Excellent')  
) |>  
cbc\_design(  
 n\_resp = 900, # Number of respondents  
 n\_alts = 3, # Number of alternatives per question  
 n\_q = 6 # Number of questions per respondent  
) |>  
cbc\_choices(  
 obsID = "obsID",  
 priors = list(  
 price = 0.1,  
 type = c(0.1, 0.2),  
 freshness = c(0.1, 0.2)  
 )  
) |>  
cbc\_power(  
 pars = c("price", "type", "freshness"),  
 outcome = "choice",  
 obsID = "obsID",  
 nbreaks = 10,  
 n\_q = 6  
) |>  
plot()

One benefit of this piped structure is that it enables the designer to quickly observe the implications of different design choices on statistical power. For example, consider the following “what if” design questions:

* What if one more choice question was added to each respondent?
* What if the number of alternatives per choice question was decreased to two?
* What if a labeled design were used for the type attribute?
* What if there was an interaction effect between price and type?

Obtaining an answer to each of these questions requires only small modifications to the arguments in one or more functions inside the analysis “pipeline”. Once the change is made, the entire analysis can be quickly re-executed and the results compared to those of an alternative design. Because cbcTools leverages the logitr package (which is highly optimized for speed) for simulating choices and estimating models, re-executing each analysis should only seconds to compute the entire process on most modern computers and laptops.

# Conclusions

This paper introduces an open source R package, cbcTools, for generating and evaluating choice-based conjoint experiment designs. The package contains functions for generating experiment designs, examining attribute balance and overlap, simulating choice data, and conducting power analyses. In addition to being open source, one of its primary advantages over alternatives is the ability to examine the statistical power of different designs under a variety of conditions, such as when preference heterogeneity or strong interactions between certain attributes may be expected in respondent choices. Users can simulate choice data for designs according to multinomial and mixed logit models and examine sample size requirements to achieve a desired level of precision for multinomial and mixed logit models. The package is designed such that each function along the analysis process generates an object for the next function, enabling the ability to conveniently pipe together a full analysis from defining attributes and levels, generating an experiment design, simulating choice data, and examining statistical power. Detailed package documentation can be found at <https://jhelvy.github.io/cbcTools/>.

# References

Bache, Stefan Milton, Hadley Wickham, Lionel Henry, and Maintainer Lionel Henry. 2022. “Package ‘Magrittr’.”

Hauser, John R., and Olivier Toubia. 2005. “The Impact of Utility Balance and Endogeneity in Conjoint Analysis.” *Marketing Science* 24 (3): 498–507. <https://doi.org/10.1287/mksc.1040.0108>.

Helveston, John Paul. 2021. *logitr: Fast Estimation of Multinomial and Mixed Logit Models with Preference Space and Willingness to Pay Space Utility Parameterizations*. <https://jhelvy.github.io/logitr/>.

Helveston, John Paul. 2022. *cbcTools: Tools for Designing Choice-Based Conjoint Survey Experiments*. <https://jhelvy.github.io/cbcTools/>.

Huber, Joel, and Klaus Zwerina. 1996. “The Importance of Utility Balance in Efficient Choice Designs.” *Journal of Marketing Research* 33 (3): 307–17.

Johnson, F Reed, Emily Lancsar, Deborah Marshall, Vikram Kilambi, Axel Mühlbacher, Dean A Regier, Brian W Bresnahan, Barbara Kanninen, and John FP Bridges. 2013. “Constructing Experimental Designs for Discrete-Choice Experiments: Report of the ISPOR Conjoint Analysis Experimental Design Good Research Practices Task Force.” *Value in Health* 16 (1): 3–13.

Johnson, Richard M, and Bryan K Orme. 2002. “Sawtooth Software How Many Questions Should You Ask in Choice-Based Conjoint Studies ?” 360. Vol. 98382.

Kessels, Roselinde, Peter Goos, and Martina Vandebroek. 2006. “A Comparison of Criteria to Design Efficient Choice Experiments.” *Journal of Marketing Research* 43 (3): 409–19.

Kuhfeld, Warren F. 2002. “Efficient Experimental Designs Using Computerized Searches” 98382 (360).

Walker, Joan L, Yanqiao Wang, Mikkel Thorhauge, and Moshe Ben-Akiva. 2018. “D-Efficient or Deficient? A Robustness Analysis of Stated Choice Experimental Designs.” *Theory and Decision* 84 (2): 215–38.

Wickham, Hadley. 2014. “Tidy Data.” *Journal of Statistical Software* 59 (10): 1–23. <https://doi.org/10.18637/jss.v059.i10>.

Wickham, Hadley, Winston Chang, and Maintainer Hadley Wickham. 2016. “Package ‘Ggplot2’.” *Create Elegant Data Visualisations Using the Grammar of Graphics. Version* 2 (1): 1–189.

1. Engineering Management and Systems Engineering, George Washington University, Washington, D.C. USA [↑](#footnote-ref-1)
2. Note that randomly selecting profiles is different from the kinds of “randomized” designs Sawtooth Software generates, where respondents are randomly assigned to blocks in the experiment which later are *not* randomly constructed. [↑](#footnote-ref-2)
3. Note that including hard constraints in your designs can substantially reduce the statistical power of your design, so use them cautiously and avoid them if possible. [↑](#footnote-ref-3)