Introduction to Open Source Conjoint

John Paul Helveston, Ph.D.

The George Washington University | Dept. of Engineering Management and Systems Engineering

June 15, 2023

Target audience

You are familiar with:

- Conjoint analysis / discrete choice experiments
- Choice modeling / utility models
- R / programming in general

Install Software!

https://jhelvy.github.io/2023-qux-conf-conjoint/software

Hello World!

John Helveston, Ph.D.

Assistant Professor, Engineering Management & Systems Engineering

- 2016-2018 Postdoc at Institute for Sustainable Energy, Boston University
- 2016 PhD in Engineering & Public Policy at Carnegie Mellon University
- 2015 MS in Engineering & Public Policy at Carnegie Mellon University
- 2010 BS in Engineering Science & Mechanics at Virginia Tech
- Website: www.jhelvy.com

Technology Change Lab

I study how consumers, firms, markets, and policy affect technological change, with a focus on accelerating the transition to low-carbon technologies

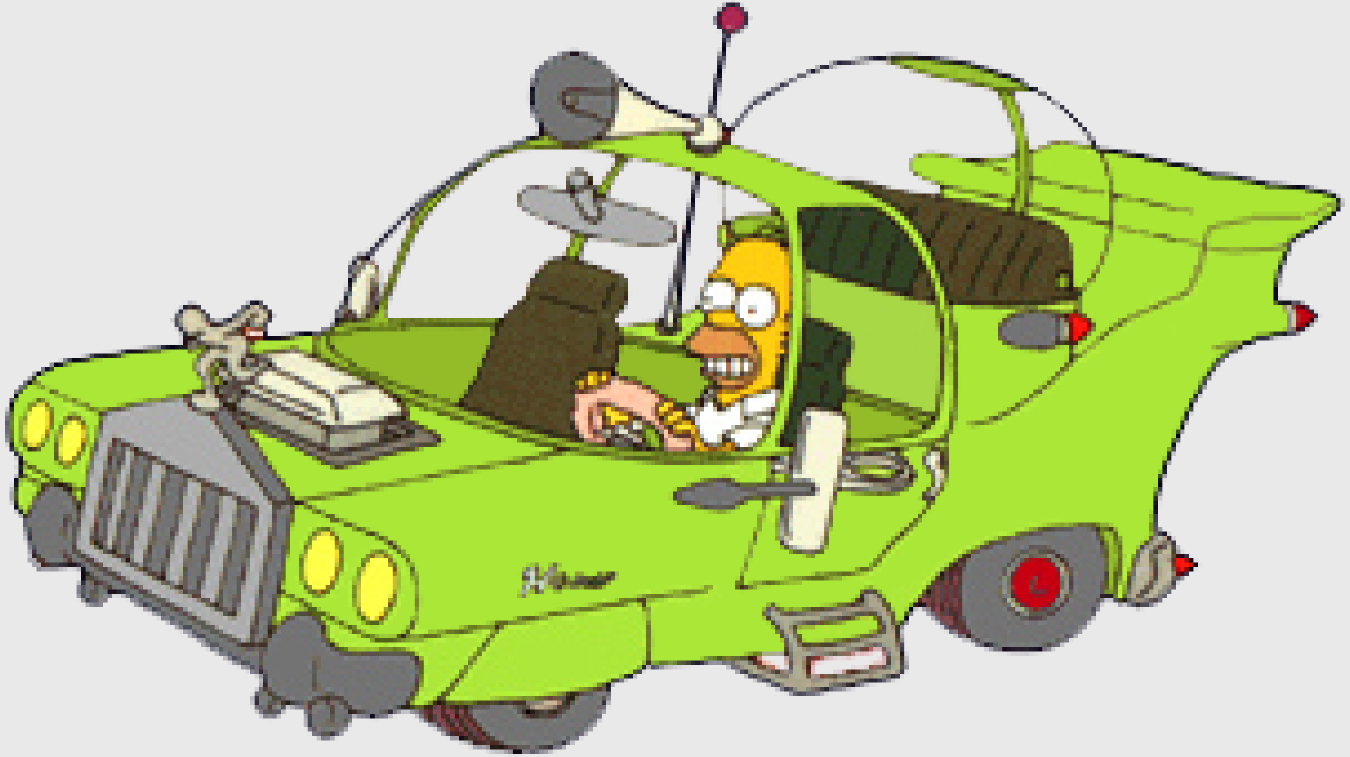
Electric & Sustainable Vehicle Technologies

Market & Policy Analysis

How can you find out know what people want?

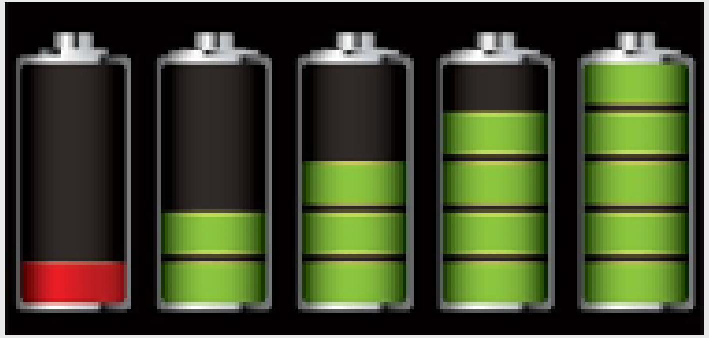
Directly asking people what they want isn't always helpful

(People want everything)



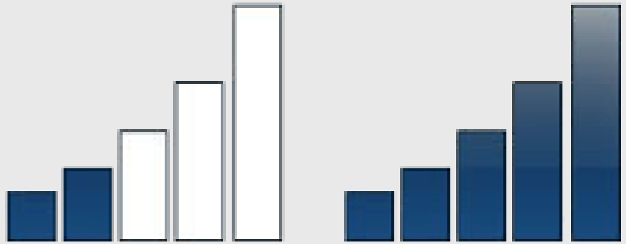
Which feature do you care more about?

Battery Life?



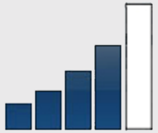
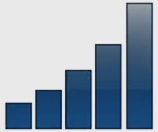
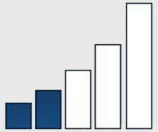
Brand?

Signal quality?



Conjoint Analysis:

Use choice data to model preferences

Attribute	Phone 1	Phone 2	Phone 3
Price	$400	$450	$350
Brand		LG	SAMSUNG
Battery Life			
Signal Quality			

Use random utility framework to predict probability of choosing phone *j*

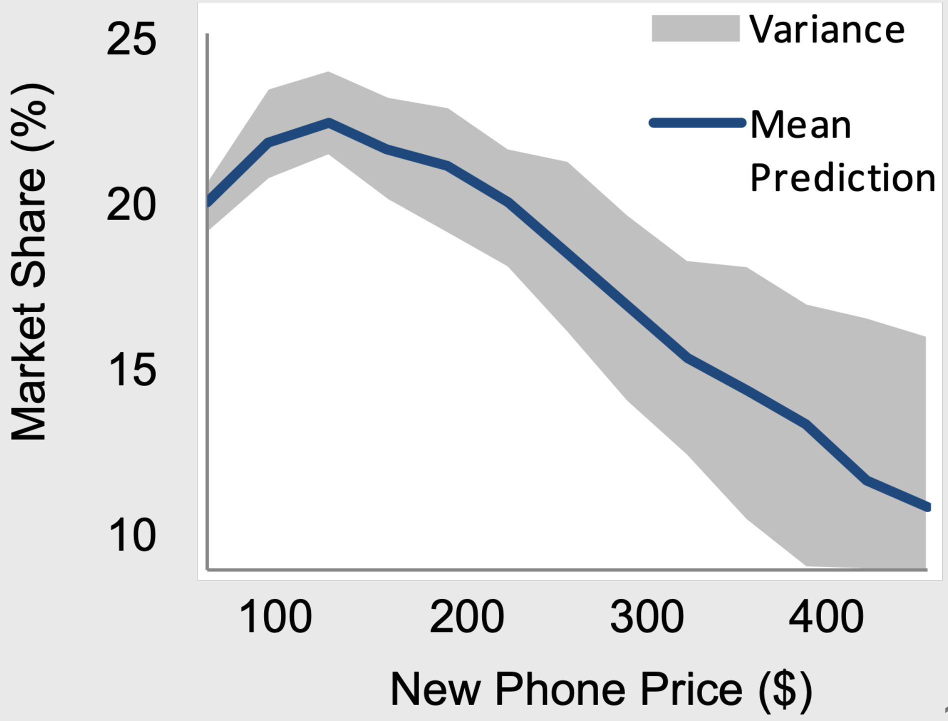
1. $u_j = \beta_1 \mathbf{price}_j + \beta_2 \mathbf{brand}_j + \beta_3 \mathbf{battery}_j + \beta_4 \mathbf{signal}_j + \varepsilon_j$
2. Assume $\varepsilon_j \sim$ iid Gumbel distribution
3. Probability of choosing phone *j*: $P_j = \frac{e^{\beta' x_j}}{\sum_k^J e^{\beta' x_k}}$
4. Estimate β_1, β_2, β_3, β_4 via maximum likelihood estimation

Willingness to Pay

Respondents on average are willing to pay $XX to improve battery life by XX%

Make predictions

$$P_j = \frac{e^{\hat{\beta}' x_j}}{\sum_k^J e^{\hat{\beta}' x_k}}$$



Choice-Based Conjoint Analysis Steps

1. Design a survey (design of experiment)
2. Implement it online
3. (Collect data) <- not covering this today
4. Estimate models

Software for Choice-Based Conjoint Analysis

	Sawtooth Software	jmp STATISTICAL DISCOVERY	STATA	Ngene
Experiment Design	✅	✅	✅	✅
Online Surveys	✅			
Model Estimation	✅	✅	✅	

- **Licenses cost $$$**
- **Not reproducible**

FOSS for Choice-Based Conjoint Analysis

Experiment Design

R:

- {cbcTools}
- {ExpertChoice}
- {support.CEs}
- {idefix}
- {choiceDes}

Online Surveys

R:

- formr

Model Estimation

R:

- {logitr}
- {apollo}
- {mlogit}
- {gmnl}
- {mixl}

Other:

- Python: {xlogit}
- Stan

FOSS for Choice-Based Conjoint Analysis

Experiment Design

cbcTools

by John Paul Helveston

Online Surveys

form{`r}

by Ruben C. Arslan and Cyril S. Tata

Conjoint adaptation by John Paul Helveston

Model Estimation

logitR

by John Paul Helveston

Back to workshop website:

https://jhelvy.github.io/2023-qux-conf-conjoint/

@JohnHelveston
@jhelvy
@jhelvy
jhelvy.com
jph@gwu.edu

Designing Conjoint Surveys with {cbcTools}

John Paul Helveston

The George Washington University | Dept. of Engineering Management and Systems Engineering

June 15, 2023

Designing a Choice-Based Conjoint Survey is Hard

Design Parameters

- What are my attributes and levels?
- Sample size (# respondents)
- Choice questions per respondent
- Alternative per choice question
- Labeled or unlabeled design?

Designing a Choice-Based Conjoint Survey is Hard

Design Parameters

- What are my attributes and levels?
- Sample size (# respondents)
- Choice questions per respondent
- Alternative per choice question
- Labeled or unlabeled design?

Design of Experiment

- Orthogonality
- Balance
- Overlap

Designing a Choice-Based Conjoint Survey is Hard

Design Parameters

- What are my attributes and levels?
- Sample size (# respondents)
- Choice questions per respondent
- Alternative per choice question
- Labeled or unlabeled design?

Design of Experiment

- Orthogonality
- Balance
- Overlap

User Experience

- Implausible combinations
- Respondent fatigue

Many R packages for design of experiment

- {cbcTools}
- {ExpertChoice}
- {support.CEs}
- {idefix}
- {choiceDes}

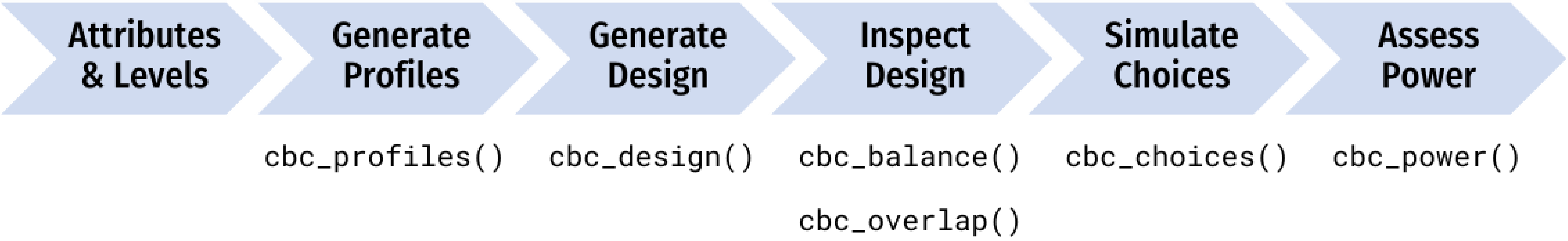
Many R packages for design of experiment

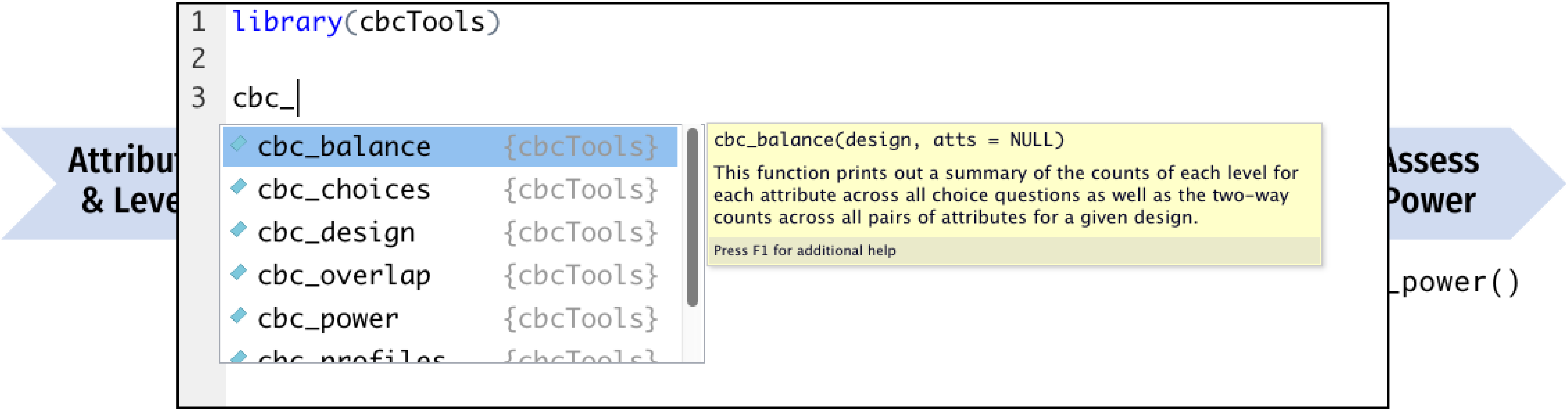
- {cbcTools} ← Does a lot more than just DOE!
- {ExpertChoice}
- {support.CEs}
- {idefix}
- {choiceDes}

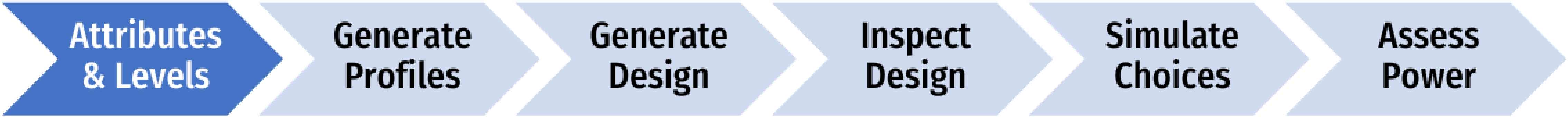
A systematic workflow for designing a CBC experiment

A systematic workflow for designing a CBC experiment

A systematic workflow for designing a CBC experiment



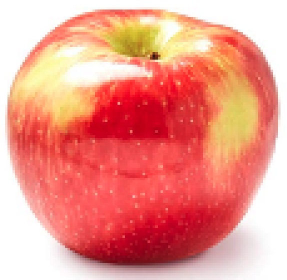

Attribu
& Leve
Assess
Power
_power()
1 library(cbcTools)
2
3 cbc_
cbc_balance {cbcTools}
cbc_choices {cbcTools}
cbc_design {cbcTools}
cbc_overlap {cbcTools}
cbc_power {cbcTools}
cbc_balance(design, atts = NULL)
This function prints out a summary of the counts of each level for each attribute across all choice questions as well as the two-way counts across all pairs of attributes for a given design.
Press F1 for additional help



`cbc_profiles()` `cbc_design()` `cbc_balance()` `cbc_choices()` `cbc_power()`

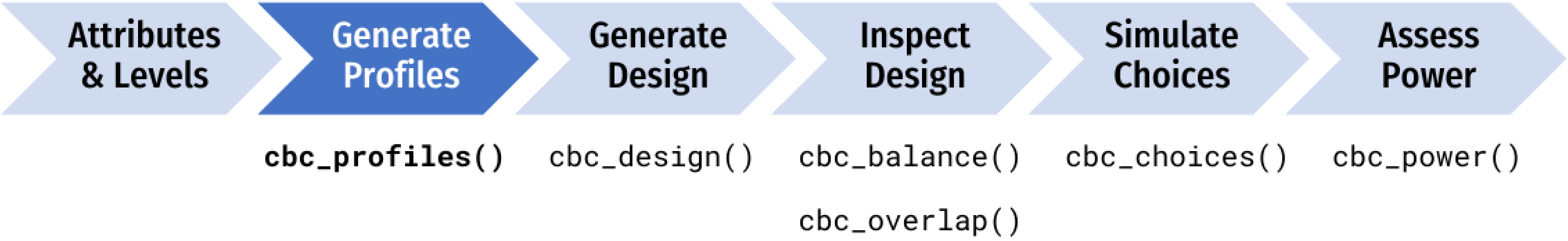
`cbc_overlap()`

Example CBC question about apples

Option 1	Option 2	Option 3
		
Type: Pink Lady **Price**: $ 2 / lb **Freshness**: Average	**Type**: Pink Lady **Price**: $ 1.5 / lb **Freshness**: Excellent	**Type**: Honeycrisp **Price**: $ 2 / lb **Freshness**: Average

Define the attributes and levels

- **Price ($/lb)**: 1.00, 1.50, 2.00, 2.50, 3.00, 3.50, 4.00
- **Type**: Fuji, Gala, Honeycrisp
- **Freshness**: Excellent, Average, Poor


Attributes & Levels
Generate Profiles
Generate Design
Inspect Design
Simulate Choices
Assess Power
cbc_profiles()
cbc_design()
cbc_balance()
cbc_overlap()
cbc_choices()
cbc_power()

Generate all possible profiles

```r
profiles <- cbc_profiles(
  price     = seq(1, 4, 0.5), # $ per pound
  type      = c('Fuji', 'Gala', 'Honeycrisp'),
  freshness = c('Poor', 'Average', 'Excellent')
)
```

```r
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
```

```r
tail(profiles)
```

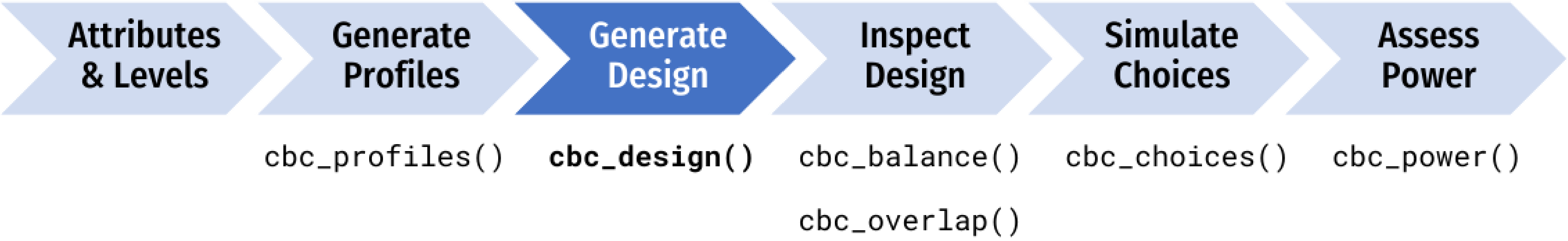
```
#>    profileID price       type freshness
#> 58        58   1.5 Honeycrisp Excellent
#> 59        59   2.0 Honeycrisp Excellent
#> 60        60   2.5 Honeycrisp Excellent
#> 61        61   3.0 Honeycrisp Excellent
#> 62        62   3.5 Honeycrisp Excellent
#> 63        63   4.0 Honeycrisp Excellent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

Generate a restricted set of profiles?

CAUTION: including restrictions in your designs can substantially reduce the statistical power of your design, so use them cautiously (and avoid them if possible).

```r
restricted_profiles <- cbc_restrict(
    profiles,
    type == "Gala" & price %in% c(1.5, 2.5, 3.5),
    type == "Honeycrisp" & price < 2,
    type == "Fuji" & freshness == "Poor"
)

dim(restricted_profiles)
```

```
#> [1] 41  4
```


Attributes & Levels
Generate Profiles
Generate Design
Inspect Design
Simulate Choices
Assess Power
cbc_profiles()
cbc_design()
cbc_balance()
cbc_overlap()
cbc_choices()
cbc_power()

Generate a survey design

```r
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6    # Number of questions per respondent
)
```

```r
head(design)
```

```
#>   profileID respID qID altID obsID price       type freshness
#> 1        53      1   1     1     1   2.5       Gala Excellent
#> 2        45      1   1     2     1   2.0       Fuji Excellent
#> 3        33      1   1     3     1   3.0       Gala   Average
#> 4        19      1   2     1     2   3.0 Honeycrisp      Poor
#> 5        14      1   2     2     2   4.0       Gala      Poor
#> 6        28      1   2     3     2   4.0       Fuji   Average
```

Include a "no choice" option

```
design <- cbc_design(
  profiles  = profiles,
  n_resp    = 300, # Number of respondents
  n_alts    = 3,   # Number of alternatives per question
  n_q       = 6,    # Number of questions per respondent
  no_choice = TRUE
)
```

```
head(design)
```

```
#>   profileID respID qID altID obsID price type_Fuji type_Gala type_Honeycrisp freshness_
#> 1         6      1   1     1     1   3.5         1         0               0
#> 2         1      1   1     2     1   1.0         1         0               0
#> 3        27      1   1     3     1   3.5         1         0               0
#> 4         0      1   1     4     1   0.0         0         0               0
#> 5        48      1   2     1     2   3.5         1         0               0
#> 6         1      1   2     2     2   1.0         1         0               0
```

Make a labeled design

(aka "alternative-specific design")

```r
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6,    # Number of questions per respondent
  label    = "type"
)
```

```r
head(design)
```

```
#>   profileID respID qID altID obsID price       type freshness
#> 1        22      1   1     1     1   1.0       Fuji   Average
#> 2        55      1   1     2     1   3.5       Gala Excellent
#> 3        63      1   1     3     1   4.0 Honeycrisp Excellent
#> 4        28      1   2     1     2   4.0       Fuji   Average
#> 5        54      1   2     2     2   3.0       Gala Excellent
#> 6        57      1   2     3     2   1.0 Honeycrisp Excellent
```

Make a Bayesian D-efficient design

(Uses the `idefix` package to generate a design)

```
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6,   # Number of questions per respondent
  priors = list(
    price     = -0.1, # Numeric, modeled as continuous
    type      = c(0.1, 0.2), # Reference level: "Fuji"
    freshness = c(0.1, 0.2) # Reference level: "Poor"
  )
)
```

Priors are defining the following model:

$$u_j = -0.1p_j + 0.1t_j^{Gala} + 0.2t_j^{Honeycrisp} + 0.1f_j^{Ave} + 0.2f_j^{Excellent} + \varepsilon_j$$

Import a design: Sawtooth → CSV → R

```r
library(readr)

design <- read_csv('design.csv')

head(design)
```

```
#>   respID qID altID obsID price       type freshness
#> 1      1   1     1     1   1.0       Fuji   Average
#> 2      1   1     2     1   3.5       Gala Excellent
#> 3      1   1     3     1   4.0 Honeycrisp Excellent
#> 4      1   2     1     2   4.0       Fuji   Average
#> 5      1   2     2     2   3.0       Gala Excellent
#> 6      1   2     3     2   1.0 Honeycrisp Excellent
```

Attributes & Levels | Generate Profiles | Generate Design | **Inspect Design** | Simulate Choices | Assess Power

`cbc_profiles()` `cbc_design()` **`cbc_balance()`** `cbc_choices()` `cbc_power()`

`cbc_overlap()`

Check design **balance**

```
cbc_balance(design)
```

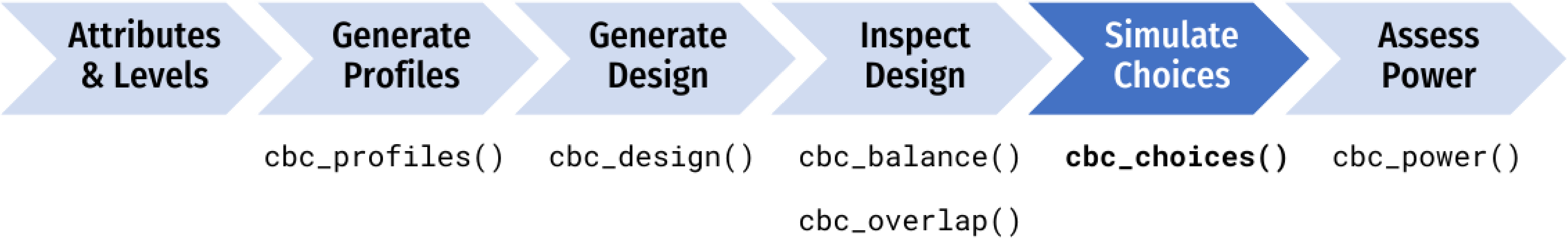
```
Individual attribute level counts

price:

  1 1.5   2 2.5   3 3.5   4
784 755 759 741 776 827 758

type:

      Fuji       Gala Honeycrisp
      1800       1800       1800

freshness:

     Poor   Average Excellent
     1845      1767      1788
```

```
Pairwise attribute level counts

price x type:

        Fuji Gala Honeycrisp
     NA 1800 1800       1800
1   784  260  256        268
1.5 755  248  254        253
2   759  259  240        260
2.5 741  239  254        248
3   776  263  286        227
3.5 827  264  258        305
4   758  267  252        239
```

Check design **overlap**

```r
cbc_overlap(design)
```

```
Counts of attribute overlap:
(# of questions with N unique levels)

price:

     1    2    3
    31  630 1139

type:

     1    2    3
   156 1248  396

freshness:

     1    2    3
   175 1189  436
```


Attributes & Levels
Generate Profiles
Generate Design
Inspect Design
Simulate Choices
Assess Power
cbc_profiles()
cbc_design()
cbc_balance()
cbc_overlap()
cbc_choices()
cbc_power()

Simulate random choices

```
data <- cbc_choices(
  design = design,
  obsID  = "obsID"
)
```

```
head(data)
```

```
#>   profileID respID qID altID obsID price       type freshness choice
#> 1        22      1   1     1     1   1.0       Fuji   Average      0
#> 2        55      1   1     2     1   3.5       Gala Excellent      0
#> 3        63      1   1     3     1   4.0 Honeycrisp Excellent      1
#> 4        28      1   2     1     2   4.0       Fuji   Average      1
#> 5        54      1   2     2     2   3.0       Gala Excellent      0
#> 6        57      1   2     3     2   1.0 Honeycrisp Excellent      0
```

Simulate choices according to a prior

(Fixed coefficients)

```
data <- cbc_choices(
  design = design,
  obsID = "obsID",
  priors = list(
    price     = -0.1,
    type      = c(0.1, 0.2),
    freshness = c(0.1, -0.2)
  )
)
```

Attribute	Level	Utility
Price	Continuous	-0.1
Type	Fuji	0
	Gala	0.1
	Honeycrisp	0.2
Freshness	Average	0
	Excellent	0.1
	Poor	-0.2

Simulate choices according to a prior

(Random coefficients...currently supports Normal & Log-normal)

```
data <- cbc_choices(
  design = design,
  obsID = "obsID",
  priors = list(
    price = -0.1,
    type = randN(
      mu    = c(0.1, 0.2),
      sigma = c(0.5, 1)
    ),
    freshness = c(0.1, -0.2)
  )
)
```

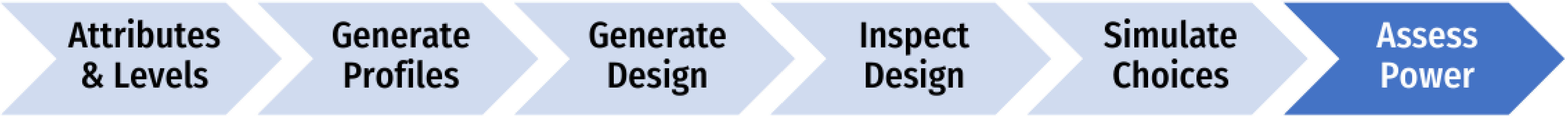
Attribute	Level	Utility
Price	Continuous	-0.1
Type	Fuji	0
	Gala	N(0.1, 0.5)
	Honeycrisp	N(0.2, 1)
Freshness	Average	0
	Excellent	0.1
	Poor	-0.2

Simulate choices according to a prior

(Models with interactions)

```
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)
  )
)
```

Attribute	Level	Utility
Price	Continuous	-0.1
Type	Fuji	0
	Gala	0.1
	Honeycrisp	0.2
Freshness	Average	0
	Excellent	0.1
	Poor	-0.2
Price x Type	Fuji	0
	Gala	0.1
	Honeycrisp	0.5



`cbc_profiles()` `cbc_design()` `cbc_balance()` `cbc_choices()` **`cbc_power()`**

`cbc_overlap()`

Conduct a power analysis

```r
power <- cbc_power(
    nbreaks = 10,
    n_q     = 6,
    data    = data,
    obsID   = "obsID",
    outcome = "choice",
    pars    = c("price", "type", "freshness")
)
```

```r
head(power)
```

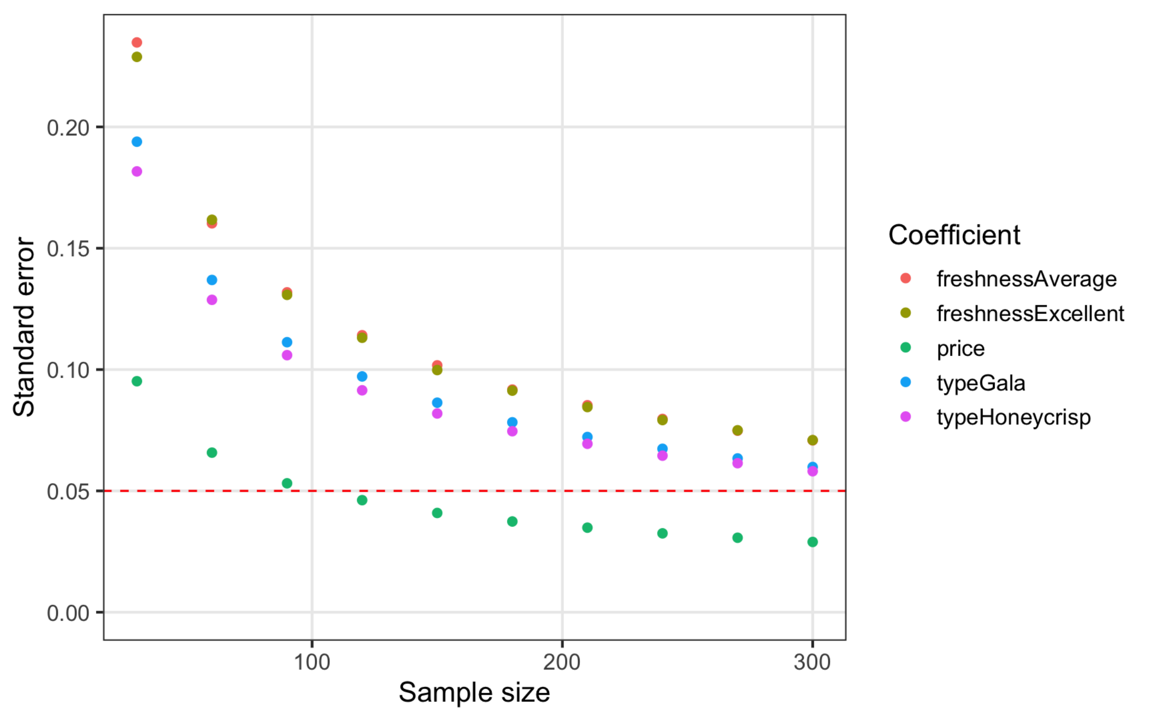
```
#>   sampleSize                coef          e
#> 1         30               price -0.189699
#> 2         30            typeGala -0.030741
#> 3         30      typeHoneycrisp  0.199566
#> 4         30    freshnessAverage  0.467130
#> 5         30 freshnessExcellent  0.477121
#> 6         60               price -0.131852
```

```r
tail(power)
```

```
#>    sampleSize                coef
#> 45        270 freshnessExcellent -0.14791
#> 46        300              price -0.11983
#> 47        300           typeGala  0.08577
#> 48        300     typeHoneycrisp  0.22142
#> 49        300   freshnessAverage  0.17092
#> 50        300 freshnessExcellent -0.11784
```

Conduct a power analysis

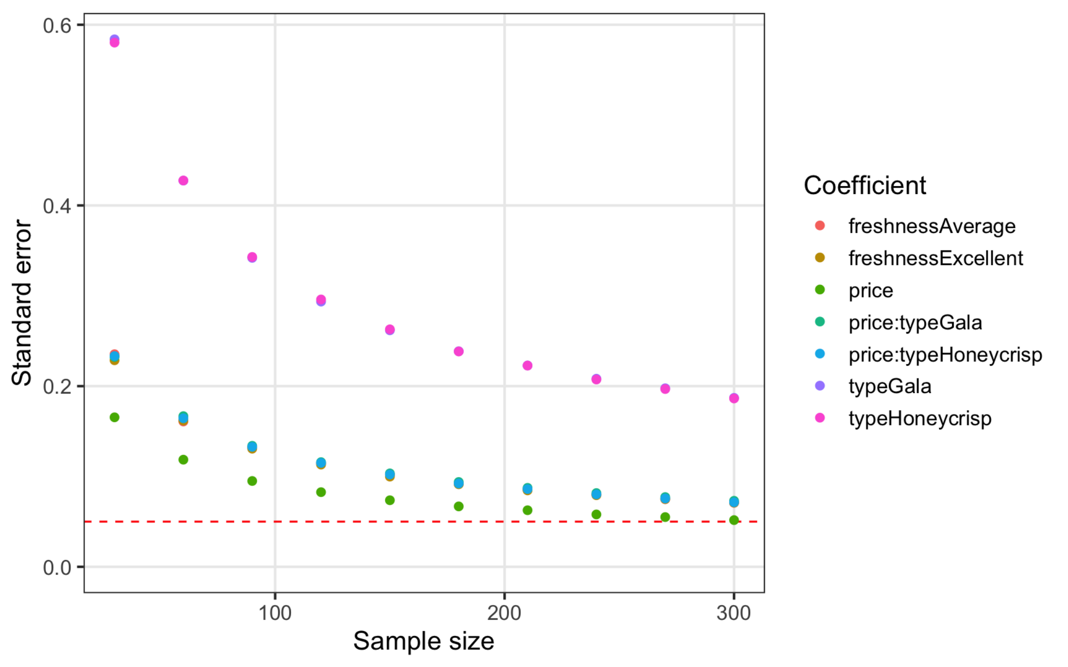
```
plot(power)
```



Conduct a power analysis

```r
power_int <- cbc_power(
    nbreaks = 10,
    n_q     = 6,
    data    = data,
    pars    = c(
      "price",
      "type",
      "freshness",
      "price*type"
    ),
    outcome = "choice",
    obsID   = "obsID"
)
```

```r
plot(power_int)
```



cbcTools

Attributes & Levels

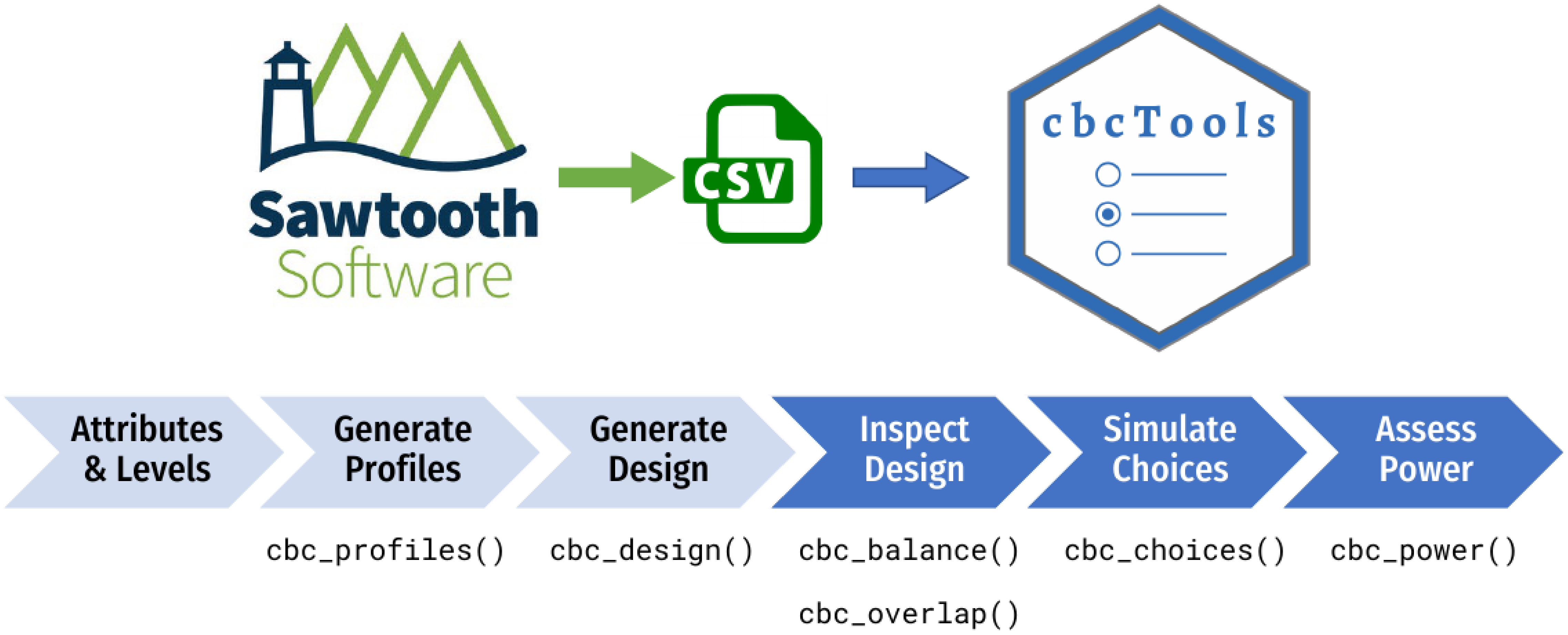
Generate Profiles
`cbc_profiles()`

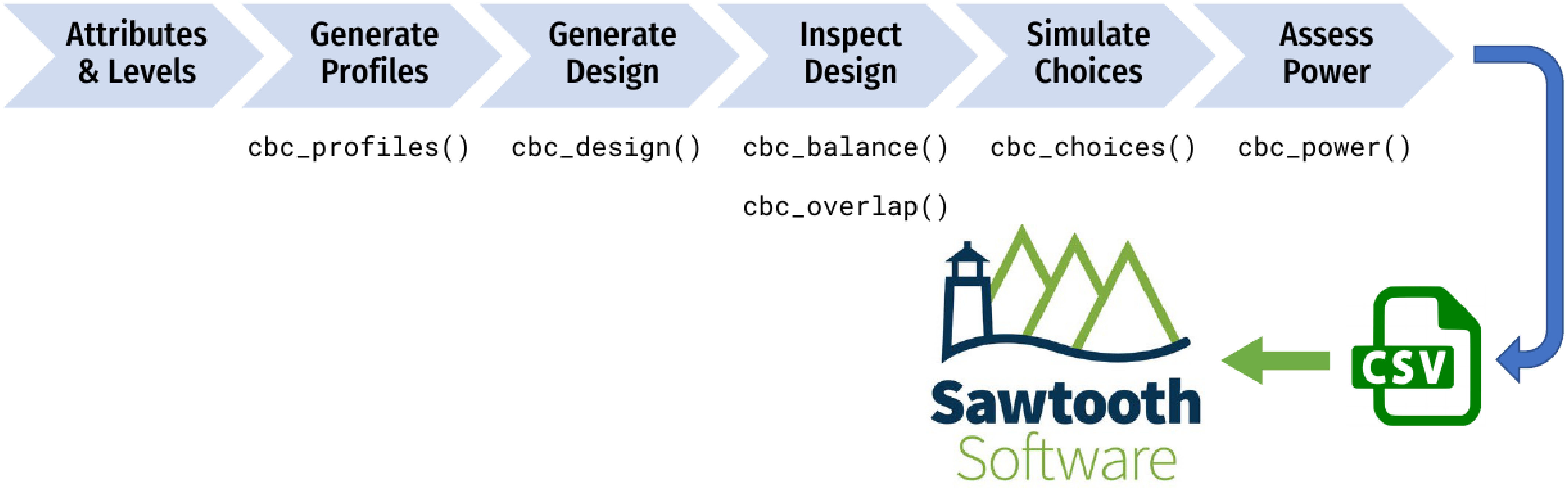
Generate Design
`cbc_design()`

Inspect Design
`cbc_balance()`
`cbc_overlap()`

Simulate Choices
`cbc_choices()`

Assess Power
`cbc_power()`


Sawtooth
Software
CSV
cbcTools
Attributes & Levels
Generate Profiles
Generate Design
Inspect Design
Simulate Choices
Assess Power
cbc_profiles()
cbc_design()
cbc_balance()
cbc_overlap()
cbc_choices()
cbc_power()


Attributes & Levels
Generate Profiles
Generate Design
Inspect Design
Simulate Choices
Assess Power
cbc_profiles()
cbc_design()
cbc_balance()
cbc_overlap()
cbc_choices()
cbc_power()
CSV
Sawtooth Software

Your turn

10:00

- Be sure to have downloaded and unzipped the practice code.
- Open the `2023-qux-conf-conjoint.Rproj` file to open RStudio.
- In RStudio, open the `designing-surveys.R` file.
- Experiment with different design options, then examine the power:
    - What if you modify the quesitons per respondent?
    - What if you use a labeled design?
    - What if you include a "no choice" option?
    - What if you use a Bayesian D-efficient design?

Back to workshop website:

https://jhelvy.github.io/2023-qux-conf-conjoint/

@JohnHelveston
@jhelvy
@jhelvy
jhelvy.com
jph@gwu.edu

Estimating Models with {logitr}

John Paul Helveston

The George Washington University | Dept. of Engineering Management and Systems Engineering

June 15, 2023

Many FOSS options model estimation

R packages:

- {logitr}: Fastest, mixed logit, WTP space.
- {apollo}: Most flexible, great documentation.
- {mlogit}: The OG R package.
- {gmnl}: Generalized logit model (though slow).
- {mixl}: Good for big datasets (uses C for speed).

Python packages:

- {xlogit}: Basically Python version of {logitr}.

Stan: For the Bayesians.

Many FOSS options model estimation

R packages:

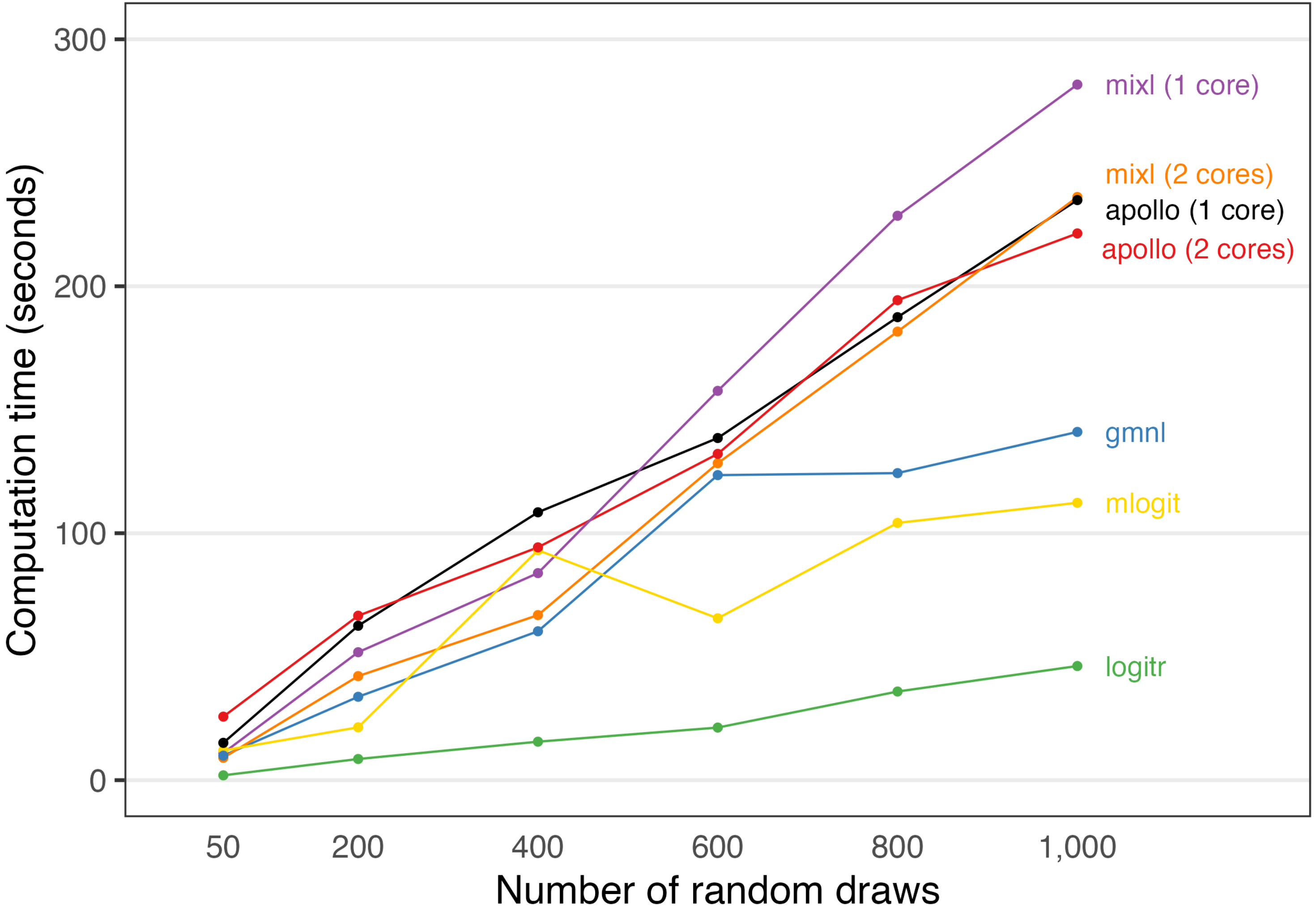
- {logitr}: Fastest, mixed logit, WTP space. ← I wrote this one, so I'm showcasing it!
- {apollo}: Most flexible, great documentation.
- {mlogit}: The OG R package.
- {gmnl}: Generalized logit model (though slow).
- {mixl}: Good for big datasets (uses C for speed).

Python packages:

- {xlogit}: Basically Python version of {logitr}.

Stan: For the Bayesians.

{logitr} is fast!



{logitr} supports two common forms of utility models

Preference Space	WTP Space
$u_j = \boldsymbol{\beta}'\mathbf{x}_j + \alpha p_j + \varepsilon_j$	$u_j = \lambda\left(\boldsymbol{\omega}'\mathbf{x}_j - p_j\right) + \varepsilon_j$

{logitr} has a similar UI with {cbcTools}

({cbcTools} uses {logitr} to simulate choices and assess power)

{cbcTools}

```
power <- cbc_power(
    nbreaks = 10,
    n_q     = 6,
    data    = data,
    obsID   = "obsID",
    outcome = "choice",
    pars    = c("price", "type", "freshness"
)
```

{logitr}

```
model <- logitr(
    data    = data,
    obsID   = "obsID",
    outcome = "choice",
    pars    = c("price", "type", "freshness"
)
```

Utility model refresher

Which would you choose?

$2.49

$2.99

$1.99

$3.99

Estimate marginal utilities

$$u_j = \boldsymbol{\beta}'\mathbf{x}_j + \alpha p_j + \varepsilon_j, \qquad \varepsilon_j \sim \text{Gumbel}\left(0, \frac{\pi^2}{6}\right)$$

```
#>                  Estimate Std. Error   z-value Pr(>|z|)
#> price          -0.3886257 0.02426923 -16.01311        0
#> brandhiland    -3.1167063 0.14496806 -21.49926        0
#> brandyoplait    1.4463603 0.08869767  16.30663        0
#> branddannon    0.6440868 0.05435965  11.84862        0
```

Convert marginal *utilities* to marginal *WTPs*

$$\hat{\omega} = \frac{\hat{\beta}}{-\hat{\alpha}}$$

```
#>                Estimate Std. Error  z-value  Pr(>|z|)
#> brandhiland  -8.01982    0.46096 -17.3980 < 2.2e-16 ***
#> brandyoplait  3.72173    0.15890  23.4214 < 2.2e-16 ***
#> branddannon   1.65734    0.16832   9.8463 < 2.2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Alternative approach: **Estimate a WTP-Space Model**

Substitutions:

$$\boldsymbol{\omega} = \frac{\boldsymbol{\beta}}{-\alpha}$$

$$\lambda = -\alpha$$

"Preference Space"

$$u_j = \boldsymbol{\beta}'\mathbf{x}_j + \alpha p_j + \varepsilon_j$$

"WTP Space"

$$u_j = \lambda\left(\boldsymbol{\omega}'\mathbf{x}_j - p_j\right) + \varepsilon_j$$

What's the difference?

Preference Space

$$u_j = \boldsymbol{\beta}'\mathbf{x}_j + \alpha p_j + \varepsilon_j$$

$$\downarrow$$

$$\hat{\boldsymbol{\omega}} = \frac{\hat{\boldsymbol{\beta}}}{-\hat{\alpha}}$$

WTP Space

$$u_j = \lambda\left(\boldsymbol{\omega}'\mathbf{x}_j - p_j\right) + \varepsilon_j$$

Mixed logit:

Unreasonably large WTP variance across population

$$u_j = \boldsymbol{\beta}'\mathbf{x}_j + \alpha p_j + \varepsilon_j$$

$$\downarrow$$

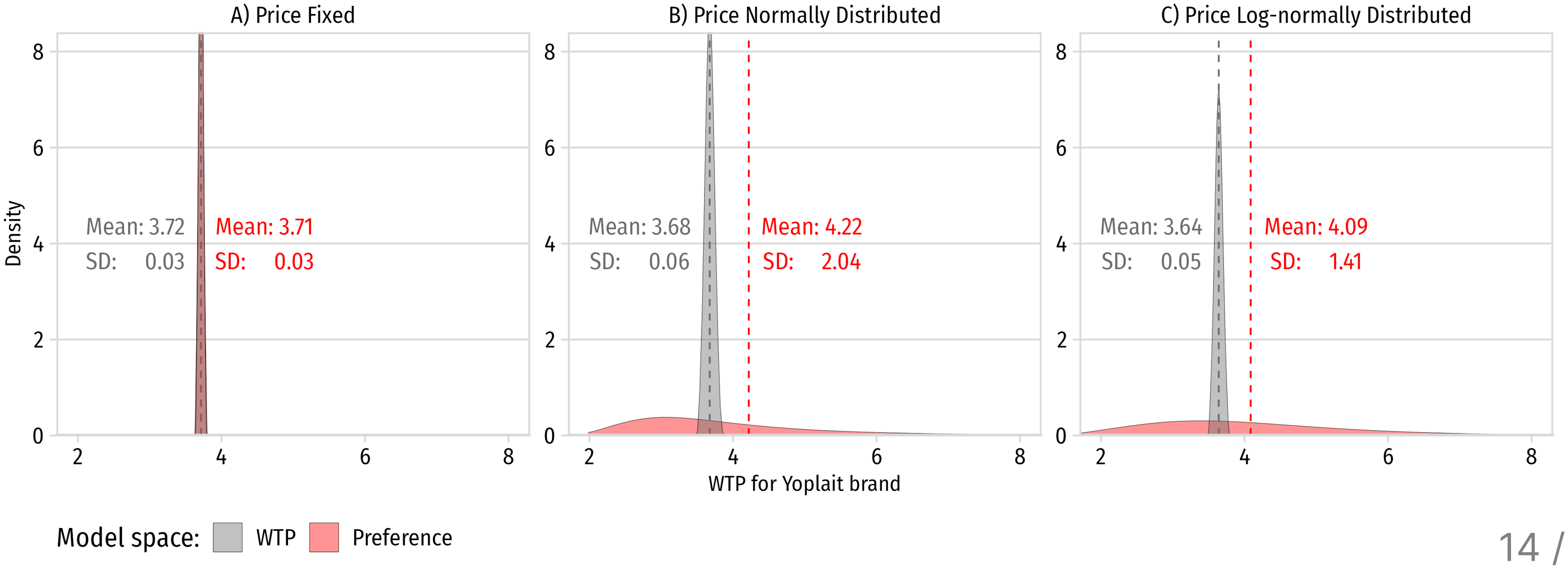
$$\hat{\boldsymbol{\omega}} = \frac{\hat{\boldsymbol{\beta}}}{-\hat{\alpha}}$$

$$\hat{\boldsymbol{\beta}} \sim \mathcal{N}\left(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}\right)$$

$$\hat{\alpha} \sim \mathcal{N}\left(\hat{\mu}, \hat{\sigma^2}\right)$$

Preference space model produces unreasonably large variance in WTP

Preference Space	**WTP Space**
$\hat{\beta} \sim \mathcal{N}\left(\hat{\mu}, \hat{\Sigma}\right)$	$\hat{\omega} \sim \mathcal{N}\left(\hat{\mu}, \hat{\Sigma}\right)$



Practical Considerations

Practical Considerations

WTP space models produce immediately interpretable results

Unit: "Utility" (relative)

$$u_j = \boldsymbol{\beta}'\mathbf{x}_j + \alpha p_j + \varepsilon_j$$

```
#>                 Estimate Std. Error   z-value Pr(>|z|)
#> price         -0.3886257 0.02426923 -16.01311        0
#> brandhiland   -3.1167063 0.14496806 -21.49926        0
#> brandyoplait   1.4463603 0.08869767  16.30663        0
#> branddannon   0.6440868 0.05435965  11.84862        0
```

Units: $ (absolute)

$$u_j = \lambda\left(\boldsymbol{\omega}'\mathbf{x}_j - p_j\right) + \varepsilon_j$$

```
#>               Estimate Std. Error  z-value  Pr(>|z|)
#> scalePar      0.388626   0.024399  15.9280 < 2.2e-16 ***
#> brandhiland  -8.019815   0.460961 -17.3980 < 2.2e-16 ***
#> brandyoplait  3.721731   0.158903  23.4214 < 2.2e-16 ***
#> branddannon   1.657345   0.168321   9.8463 < 2.2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Practical Considerations

WTPs can be directly compared across different models
(even estimates from different data sets)

$$u_j^* = \boldsymbol{\beta}^{*'} \mathbf{x}_j + \alpha^* p_j + \varepsilon_j^*, \qquad \varepsilon_j^* \sim \text{Gumbel}\left(0, \sigma^2 \frac{\pi^2}{6}\right)$$

Preference Space
Parameters proportional to σ

$$\left(\frac{u_j^*}{\sigma}\right) = \left(\frac{\boldsymbol{\beta}^*}{\sigma}\right)' \mathbf{x}_j + \left(\frac{\alpha^*}{\sigma}\right) p_j + \left(\frac{\varepsilon_j^*}{\sigma}\right)$$

$$u_j = \boldsymbol{\beta}' \mathbf{x}_j + \alpha p_j + \varepsilon_j$$

WTP Space
Parameters independent of σ

$$\left(\frac{u_j^*}{-\alpha^*}\right) = \left(\frac{\boldsymbol{\beta}^*}{-\alpha^*}\right)' \mathbf{x}_j + \left(\frac{\alpha^*}{-\alpha^*}\right) p_j + \left(\frac{\varepsilon_j^*}{-\alpha^*}\right)$$

$$u_j = \lambda \left(\boldsymbol{\omega}' \mathbf{x}_j - p_j\right) + \varepsilon_j$$

Practical Considerations

Neither space systematically predicts choice better

- **Train and Weeks (2005)** and **Sonnier et al. (2007)** found preference space model fit data better.
- **Das et al. (2009)** found nearly identical model fit on out-of-sample predictions with each model specification.

...but most software is built for

$$u_j = \boldsymbol{\beta}'\mathbf{x}_j + \alpha p_j + \varepsilon_j$$

not

$$u_j = \lambda\left(\boldsymbol{\omega}'\mathbf{x}_j - p_j\right) + \varepsilon_j$$

`logitr` to the rescue!

The logitr Package

Estimation of multinomial and mixed logit models in with "Preference" space or "Willingness-to-pay" (WTP) space utility parameterizations.

- Multinomial logit (MNL) models
- Mixed logit (MXL) models with normal and log-normal parameter distributions.
- Preference space and WTP space utility parameterizations.
- Weighted models to differentially weight individual observations.
- Uncorrelated or correlated heterogeneity covariances for mixed logit models.
- Functions for computing WTP from preference space models.
- Functions for predicting expected probabilities and outcomes for sets of alternatives based on an estimated model.
- A parallelized multistart optimization loop that uses different random starting points in each iteration to search for different local minima (useful for non-convex problems like MXL models or models with WTP space parameterizations).

Data format

Data must be arranged in a "long" format:

- Each row is an alternative from a choice observation.
- Choice observations do *not* have to be symmetric.

Required variables:

- `outcome`: A dummy variable for the chosen alternative (`1` or `0`).
- `obsID`: A sequence of repeated numbers identifying each unique choice observation, e.g. `1, 1, 2, 2, 3, 3`.
- `pars`: Any other variables to use as model covariates.

Data format

```r
head(yogurt, 10)
```

```
#>    choice obsID alt price   brand
#> 1       0     1   1   8.1  dannon
#> 2       0     1   2   6.1  hiland
#> 3       1     1   3   7.9  weight
#> 4       0     1   4  10.8 yoplait
#> 5       1     2   1   9.8  dannon
#> 6       0     2   2   6.4  hiland
#> 7       0     2   3   7.5  weight
#> 8       0     2   4  10.8 yoplait
#> 9       1     3   1   9.8  dannon
#> 10      0     3   2   6.1  hiland
```

- `outcome = "choice"`
- `obsID = "obsID"`
- `pars = c("price", "brand")`

Multinomial logit in **Preference Space**

```r
mnl_pref <- logitr(
  data    = yogurt,
  outcome = "choice",
  obsID   = "obsID",
  pars    = c("price", "brand")
)

summary(mnl_pref)
```

$$u_j = \boldsymbol{\beta}'\mathbf{x}_j + \alpha p_j + \varepsilon_j$$

```
#> =================================================
#>
#> Model estimated on: Fri Jun 09 10:12:27 2023
#>
#> Using logitr version: 1.1.0
#>
#> Call:
#> logitr(data = yogurt, outcome = "choice", obsID = "obsID", pars = c("p
#>     "brand"))
#>
#> Frequencies of alternatives:
#>        1        2        3        4
#> 0.402156 0.029436 0.229270 0.339138
#>
#> Exit Status: 3, Optimization stopped because ftol_rel or ftol_abs was
#>
#> Model Type:    Multinomial Logit
#> Model Space:          Preference
#> Model Run:                1 of 1
#> Iterations:                   20
#> Elapsed Time:        0h:0m:0.01s
#> Algorithm:        NLOPT_LD_LBFGS
#> Weights Used?:             FALSE
#> Robust?                    FALSE
#>
#> Model Coefficients:
#>               Estimate Std. Error z-value  Pr(>|z|)
#> price        -0.388626   0.024269 -16.013 < 2.2e-16 ***
#> brandhiland  -3.116706   0.144968 -21.499 < 2.2e-16 ***
#> brandyoplait  1.446360   0.088698  16.307 < 2.2e-16 ***
#> branddannon   0.644087   0.054360  11.849 < 2.2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
```

Multinomial logit in **WTP Space**

```r
library(logitr)

mnl_wtp <- logitr(
  data     = yogurt,
  outcome  = "choice",
  obsID    = "obsID",
  pars     = "brand",
  scalePar = "price"
)

summary(mnl_wtp)
```

$$u_j = \lambda \left(\boldsymbol{\omega}' \mathbf{x}_j - p_j \right) + \varepsilon_j$$

```
#> =================================================
#>
#> Model estimated on: Fri Jun 09 10:12:41 2023
#>
#> Using logitr version: 1.1.0
#>
#> Call:
#> logitr(data = yogurt, outcome = "choice", obsID = "obsID", pars = "bra
#>     scalePar = "price")
#>
#> Frequencies of alternatives:
#>        1        2        3        4
#> 0.402156 0.029436 0.229270 0.339138
#>
#> Exit Status: 3, Optimization stopped because ftol_rel or ftol_abs was
#>
#> Model Type:    Multinomial Logit
#> Model Space:   Willingness-to-Pay
#> Model Run:                 1 of 1
#> Iterations:                    40
#> Elapsed Time:          0h:0m:0.02s
#> Algorithm:          NLOPT_LD_LBFGS
#> Weights Used?:              FALSE
#> Robust?                     FALSE
#>
#> Model Coefficients:
#>               Estimate Std. Error z-value  Pr(>|z|)
#> scalePar      0.388633   0.024269  16.013 < 2.2e-16 ***
#> brandhiland  -8.019717   0.455549 -17.605 < 2.2e-16 ***
#> brandyoplait  3.721711   0.157655  23.607 < 2.2e-16 ***
#> branddannon  1.657290   0.165712  10.001 < 2.2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
```

Caution

Log-likelihood function for WTP space models is **non-convex** 😔

Use a Multistart

```r
mnl_wtp <- logitr(
  data      = yogurt,
  outcome   = "choice",
  obsID     = "obsID",
  pars      = "brand",
  scalePar = "price",
  numMultiStarts = 10
)

summary(mnl_wtp)
```

$$u_j = \lambda \left(\boldsymbol{\omega}' \mathbf{x}_j - p_j \right) + \varepsilon_j$$

```
#> =================================================
#>
#> Model estimated on: Fri Jun 09 10:13:43 2023
#>
#> Using logitr version: 1.1.0
#>
#> Call:
#> logitr(data = yogurt, outcome = "choice", obsID = "obsID", pars = "bra
#>     scalePar = "price", numMultiStarts = 10)
#>
#> Frequencies of alternatives:
#>        1        2        3        4
#> 0.402156 0.029436 0.229270 0.339138
#>
#> Summary Of Multistart Runs:
#>    Log Likelihood Iterations Exit Status
#> 1        -2665.11         40           3
#> 2        -2665.11         39           3
#> 3        -2665.11         43           3
#> 4        -2665.11         47           3
#> 5        -2665.11         54           3
#> 6        -2665.11         42           3
#> 7        -2665.11         39           3
#> 8        -2665.11         44           3
#> 9        -2665.11         39           3
#> 10       -2665.11         38           3
#>
#> Use statusCodes() to view the meaning of each status code
#>
#> Exit Status: 3, Optimization stopped because ftol_rel or ftol_abs was
#>
#> Model Type:     Multinomial Logit
#> Model Space:   Willingness-to-Pay
```

Mixed logit in **Preference Space**

```r
mxl_pref <- logitr(
  data      = yogurt,
  outcome   = "choice",
  obsID     = "obsID",
  pars      = c("price", "brand"),
  randPars = c(brand = "n"),
  numMultiStarts = 10
)
```

$$u_j = \boldsymbol{\beta}'\mathbf{x}_j + \alpha p_j + \varepsilon_j$$

$$\hat{\boldsymbol{\beta}} \sim \mathcal{N}\left(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}\right)$$

Mixed logit in **WTP Space**

```r
mxl_wtp <- logitr(
  data      = yogurt,
  outcome   = "choice",
  obsID     = "obsID",
  pars      = "brand",
  scalePar = "price",
  randPars = c(brand = "n"),
  randScale = "ln",
  numMultiStarts = 10
)
```

$$u_j = \lambda\left(\boldsymbol{\omega}'\mathbf{x}_j - p_j\right) + \varepsilon_j$$

$$\hat{\boldsymbol{\omega}} \sim \mathcal{N}\left(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}\right)$$

Convenient helper functions

`predict()`: Expected shares for a set of alternatives

Define a set of alternatives

```r
data <- subset(
  yogurt, obsID == 42,
  select = c('price', 'brand', 'obsID'))

data
```

```
#>   price   brand obsID
#> 1   6.3  dannon    42
#> 2   6.1  hiland    42
#> 3   7.9  weight    42
#> 4  11.5 yoplait    42
```

Predict probabilities

```r
predict(
  mnl_pref,
  newdata = data,
  obsID = "obsID",
  returnData = TRUE
)
```

```
#>   obsID predicted_prob price   brand
#> 1    42     0.62391435   6.3  dannon
#> 2    42     0.01568877   6.1  hiland
#> 3    42     0.17593683   7.9  weight
#> 4    42     0.18446005  11.5 yoplait
```

Your turn

10:00

- Be sure to have downloaded and unzipped the practice code.
- Open the `2023-qux-conf-conjoint.Rproj` file to open RStudio.
- In RStudio, open the `estimating-models.R` file.
- Experiment with estimating different models (use either one of the example datasets included in the package, or simulate your own data!)

{logitr} documentation:
https://jhelvy.github.io/logitr/

Back to workshop website:
https://jhelvy.github.io/2023-qux-conf-conjoint/

@JohnHelveston
@jhelvy
@jhelvy
jhelvy.com
jph@gwu.edu

Fielding Surveys with formr

John Paul Helveston, Ph.D.

The George Washington University | Dept. of Engineering Management and Systems Engineering

June 15, 2023

Making a survey in formr

Building a survey in formr

- Use *RMarkdown* / *html* to create survey elements

- Copy elements to a *Google Sheet*

- Import Google Sheets into formr *surveys*

- Link surveys together in formr *runs*

My Recommentation:
Draft your survey in RMarkdown

Survey content in `demoSurvey.Rmd`

Google sheet

Live survey

formr row types (more here)

Type	Description
`note`	Display content in `label` column
`submit`	Next page button
`mc`	Multiple choice question (single choice)
`mc_multiple`	Multiple choice question (multiple choices)
`mc_button`	Multiple choice question (large buttons)
`select_one`	Drop down menu (choose one)
`text`	Open text, single row
`textarea`	Open text, block

Some Guidelines

- Be sure that any data / images are hosted somewhere on the web
- Consider each new page a **New R Session** (reload libraries, etc.)

Embedding images

I recommend just writing html code, like this

```
<img src="https://github.com/jhelvy/2023-qux-conf-conjoint/blob/main/images/logo.png?raw=t
```

Centered image

I recommend just writing html code, like this

```
<center>
<img src="https://github.com/jhelvy/2023-qux-conf-conjoint/blob/main/images/logo.png?raw=t
</center>
```

Check your urls carefully!

This is the link to the **Github page** with the image:

https://github.com/jhelvy/2023-qux-conf-conjoint/blob/main/images/logo.png

This is the link to the **actual image**:

https://github.com/jhelvy/2023-qux-conf-conjoint/blob/main/images/logo.png?raw=true

Two ways to define choice options

Add "choice" columns

H	I	J	K
choice1	choice2	choice3	value
Yes!	Kind of	No :(	

]

Use `choices` tab
(when you have a lot of choices)

Example: "Year of birth" in this demo

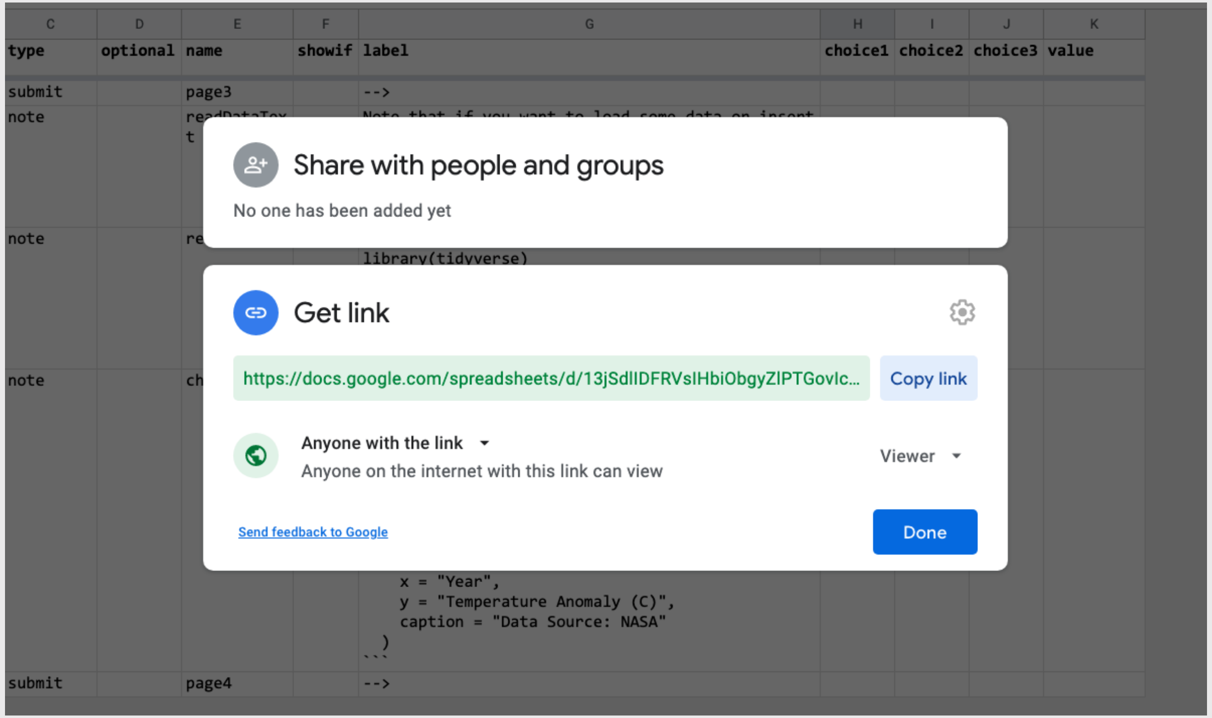
Control the way things look in `class` column

(options here)

Importing survey into formr

formr.org --> Admin --> Surveys --> Create new survey

(Make sure your Google Sheet is visible!)



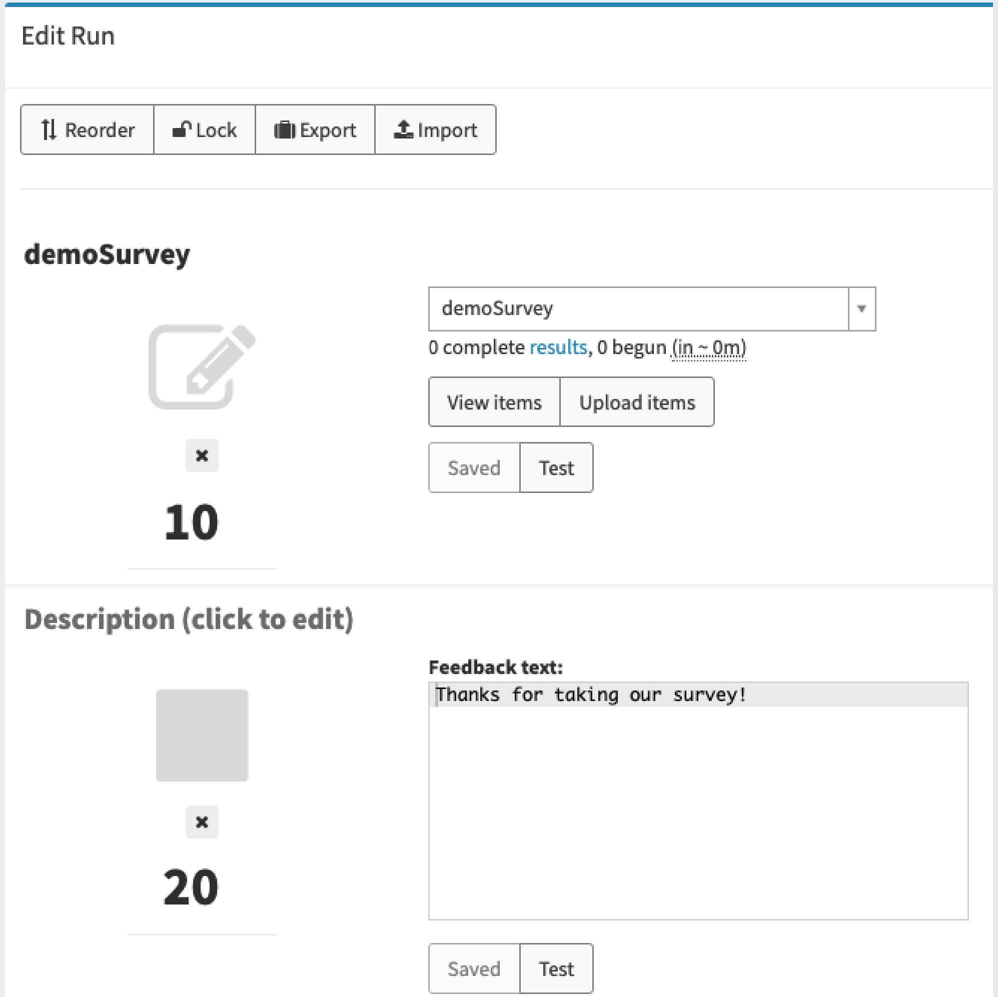
Make a run

formr.org --> Admin --> Runs --> Create new run

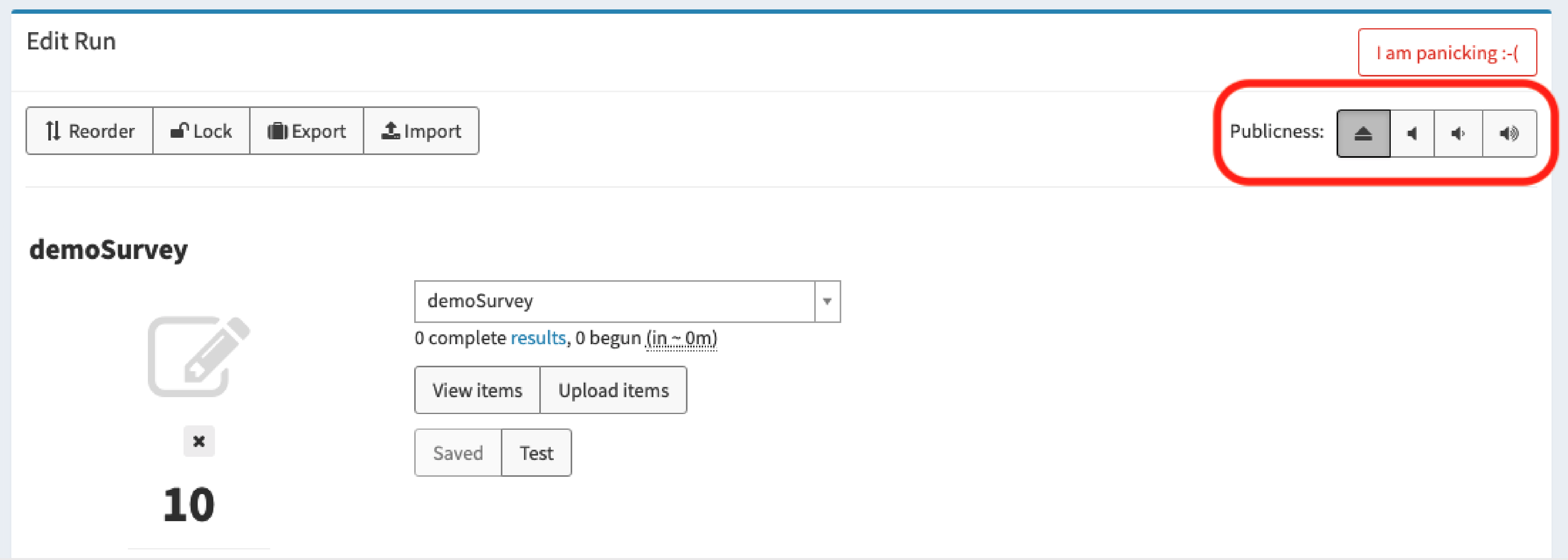
Insert survey with

Insert stop with

Change order by adjusting numbers & clicking "Reorder"



Make it "live" with the volume buttons



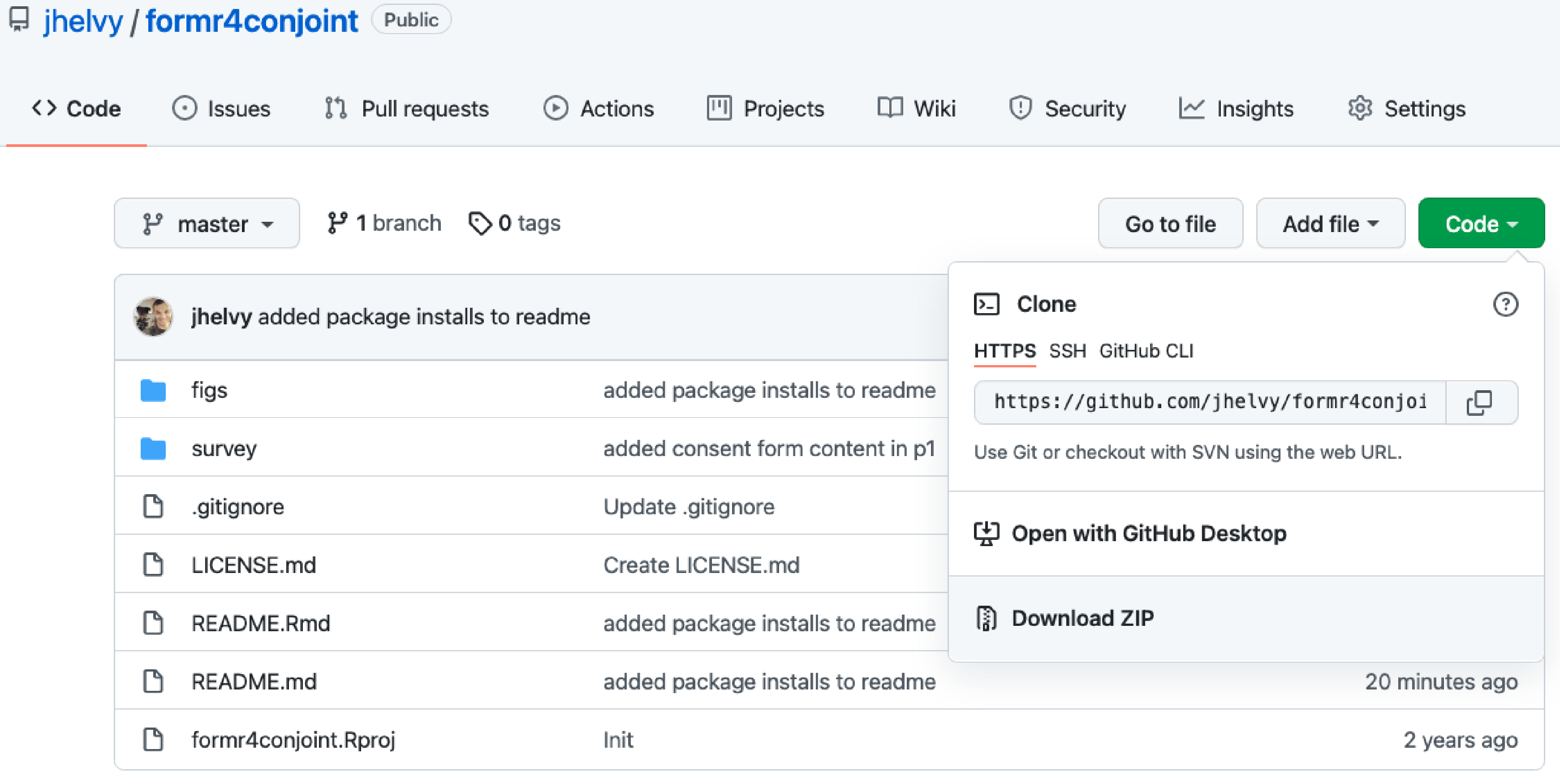
Fine tune look & feel in "Settings"

Making a *conjoint* survey in formr

(Detailed demo in [this blog post])

Full demo in the formr4conjoint repo from GitHub

(code used in the related blog post)



3 Parts

- **Part 1**: Intro
- **Part 2**: Conjoint questions
- **Part 3**: Other / demographic questions

3 Parts

- **Part 1**: Intro --> screen for target population
- **Part 2**: Conjoint questions --> screen for random answers
- **Part 3**: Other / demographic questions

Displaying your choice questions online

(See example in part two demo google sheet)

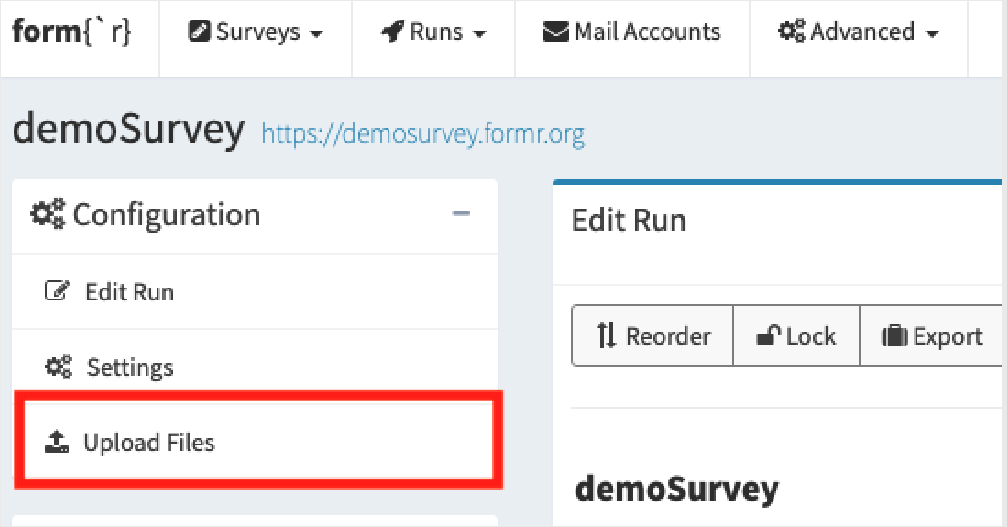
1. Export your choice questions as a .csv file
2. Upload your .csv file somewhere (e.g. GitHub)
3. Use R code to extract the values to display
4. Use RMarkdown to display the values

1. Export your experiment design (from {cbcTools}) as a .csv file

```
write_csv(design, here('choice_questions.csv'))
```

2. Upload your .csv file somewhere

Inside a formr run (private)



github.com (public)

apples example

Serialize the experiment design

Converts a data frame to one long string

```
df
```

```
#>   profileID altID price fuelEconomy accelTime powertrain
#> 1         9     1    25          30         6   Gasoline
#> 2        11     2    20          20         7   Gasoline
#> 3        23     3    20          25         8   Gasoline
```

```
df_json <- jsonlite::serializeJSON(df)
df_json
```


```
#> {"type":"list","attributes":{"names":{"type":"character","attributes":{},"value":["prof
```

Using the `calculate` type (example sheet)

RMarkdown

```r
# Read in the choice questions
library(tidyverse)
design <- read_csv("https://raw.githubuserc

# Define the respondent ID
respondentID <- sample(design$respID, 1)

# Create the subset of rows for that respond
df <- design %>%
    filter(respID == respondentID) %>%
    mutate(image = paste0("https://raw.githu

# Convert df to json
df_json <- jsonlite::serializeJSON(df)
```

Google sheet

C	D	E	K
type	**optional**	**name**	**value**
calculate		time3	Sys.time()
calculate		survey	library(tidyverse) read_csv("https://raw.githubusercontent.com/jhelvy/for
calculate		respondentID	sample(survey$respID, 1)
calculate		df	survey %>% filter(respID == respondentID) %>% mutate(image = paste0("https://raw.githubusercontent.com/jhelvy/formr4conjoint/master/survey/images/", image))
calculate		df_json	jsonlite::toJSON(df)

Random choice questions as **buttons**

Use the `mc_button` question type

`label`

- Show your question text
- Insert a code chunk to create one-row data frame for each alternative

`choice` columns

- Insert RMarkdown code to display each alternative

Random choice questions as **buttons**

Create separate data frames for each alternative

```
library(dplyr)

alts <- jsonlite::unserializeJSON(df_json)
alt1 <- alts %>% filter(altID == 1)
alt2 <- alts %>% filter(altID == 2)
alt3 <- alts %>% filter(altID == 3)
```

Use RMarkdown formatting to display content in each alternative

```
**Option 1**

**Price**: $ `r alt1$price`
**Powertrain**: $ `r alt1$powertrain`
**Fuel Economy**: `r alt1$fuelEconomy` mpg
**0-60 Accel. Time**: `r alt1$accelTime` s
```

Option 1

Price: $ 25
Powertrain: $ Gasoline
Fuel Economy: 30 mpg
0-60 Accel. Time: 6 s

Random choice questions as **table**

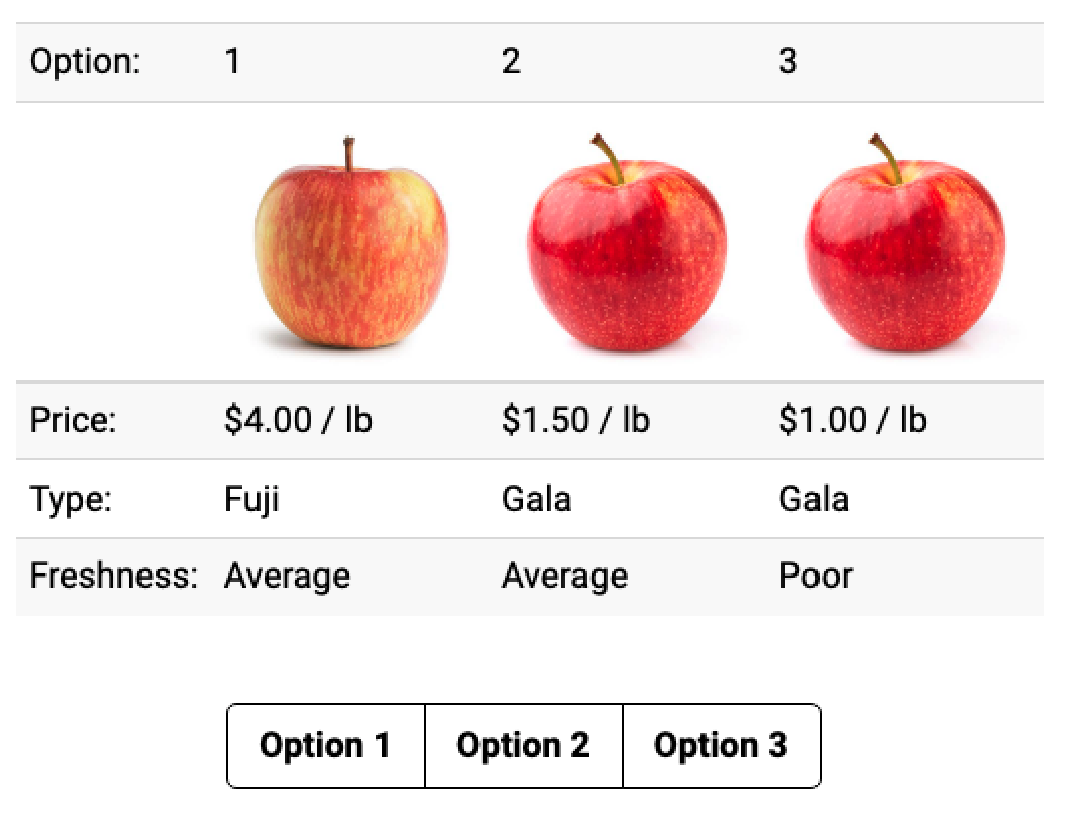
- Use the `mc_button` question type

`label`

- Show your question text
- Insert a code chunk to modify `alts` data frame & display it using `kable()`
- Use kableExtra to control table styling

`choice` columns

- Simple text / number for each option



Option:	1	2	3
Price:	$4.00 / lb	$1.50 / lb	$1.00 / lb
Type:	Fuji	Gala	Gala
Freshness:	Average	Average	Poor

Random choice questions as **table**

```r
library(dplyr)

alts <- jsonlite::unserializeJSON(df_json) %
  # Add $ sign to price
  mutate(price = scales::dollar(price)) %>%
  # Make nicer attribute labels
  select(
    `Option:`              = altID,
    `Powertrain:`          = powertrain,
    `Price:`               = price,
    `Fuel Economy (mpg):`  = fuelEconomy,
    `Accel. Time (s):`     = accelTime)

# Drop row names
row.names(alts) <- NULL
```

Display the *transpose*, `t(alts)`

```r
kable(t(alts))
```

Option:	1	2	3
Powertrain:	Gasoline	Gasoline	Gasoline
Price:	$25	$20	$20
Fuel Economy (mpg):	30	20	25
Accel. Time (s):	6	7	8

Back to workshop website:

https://jhelvy.github.io/2023-qux-conf-conjoint/

@JohnHelveston
@jhelvy
@jhelvy
jhelvy.com
jph@gwu.edu