

# Designing Stimuli (or Anything) Reproducibly

Jack E. Taylor

SIPS 2021

<https://jackedtaylor.github.io/SIPS2021/>

- How can we design lists of stimuli reproducibly?
- Focusing on “sample” stimuli
- Explain methods, then show example code
- Can implement solutions in other languages (e.g., Python, MATLAB, Julia, Javascript, heck even Excel formulas)
- No programming knowledge necessary to understand what’s going on
- Some programming knowledge necessary to apply the methods yourself

Website with slides and example code:

<https://jackedtaylor.github.io/SIPS2021/>

# Today’s Session

## Designing Stimuli (or Anything) Reproducibly

1)

- What do we mean by stimuli?
- What do we mean by reproducibility?
- How can we generate stimuli reproducibly?
- Item-wise versus distribution-wise approaches

2)

- Distribution-wise matching
- Assumption-free distribution-wise matching

3)

- Item-wise matching
- LexOPS
- Propensity score matching

4)

- Matching continuous independent variables
- Reporting matching in Methods sections
- Applications to things other than designing stimuli
- Summary, Questions, and Answers

# This Session

*15-20 minute sections*

*5ish minute breaks*

## Part 1/4

# Introductions

- PhD Candidate at University of Glasgow
- Psycholinguistics: Orthography and Semantics
- EEG Research
- Word nerd
- Cat Person
- One of those people who will spend 2 hours writing reproducible functions if it means I don't have to spend 20 minutes doing something by hand
- Currently looking for Post-Doctoral positions



Sara Sereno



Guillaume Rousselet

What is a  
Stimulus?

---

**Anything you present  
to a participant to  
elicit a response**

---

**One stimulus =  
Something presented  
on a single trial**

## “PARAMETRIC” STIMULI

- Can use any combination of multiple parameters to generate novel items
- Potentially infinite in number
- Parameters defining the stimulus known exactly
- E.g., pseudowords, procedurally generated faces

spinkle



## “SAMPLE” STIMULI

- Sampled from a population of existing items
- Only a finite number of candidates
- Parameters defining the stimulus have to be inferred
- E.g., real words, images of real faces

orange



# What is a control variable?

---

A potentially confounding variable which you don't want to pollute your effect of interest

---

*parametric stimuli*: just create items which do not differ in the control variable

---

*sample stimuli*: identify and select items such that the control variable doesn't confound your effect of interest

# Reproducibility

Can other people reproduce the method by which you generated your stimuli?

- For parametric stimuli, you can just share the code
- Are sample stimuli reproducible?

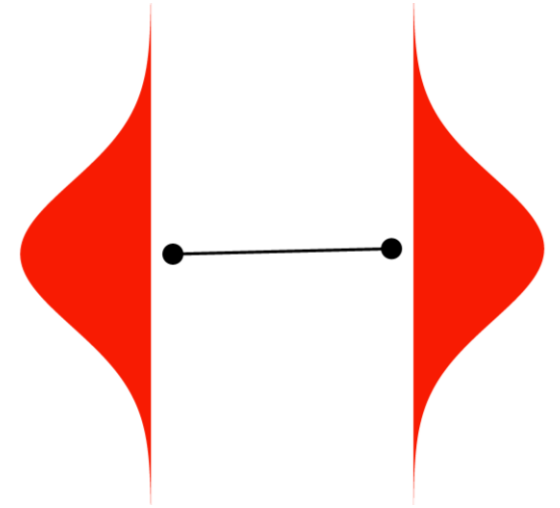


How do people  
report picking  
their “sample”  
stimuli?

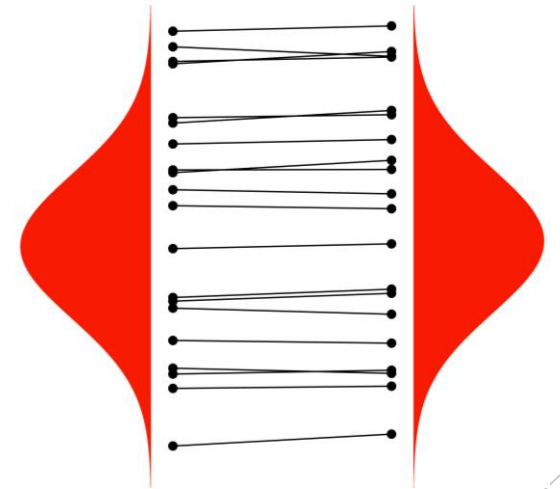
- “The two conditions were matched in terms of word length, word frequency, and age of acquisition. Independent sample *t* tests confirmed that the conditions were suitably controlled (all *p* values > .05).”  
- Foo et al. (2005): *Revisiting the Effect of Bar*
- “Items in the two conditions were matched pairwise in terms of word length, word frequency, and age of acquisition. Paired sample *t* tests confirmed that the conditions were suitably controlled (all *p* values > .05).”  
- Foo et al. (2003): *The Effect of Bar on Word Recognition*

How are people  
picking their  
“sample” stimuli?

**Distribution-Wise  
Matching**

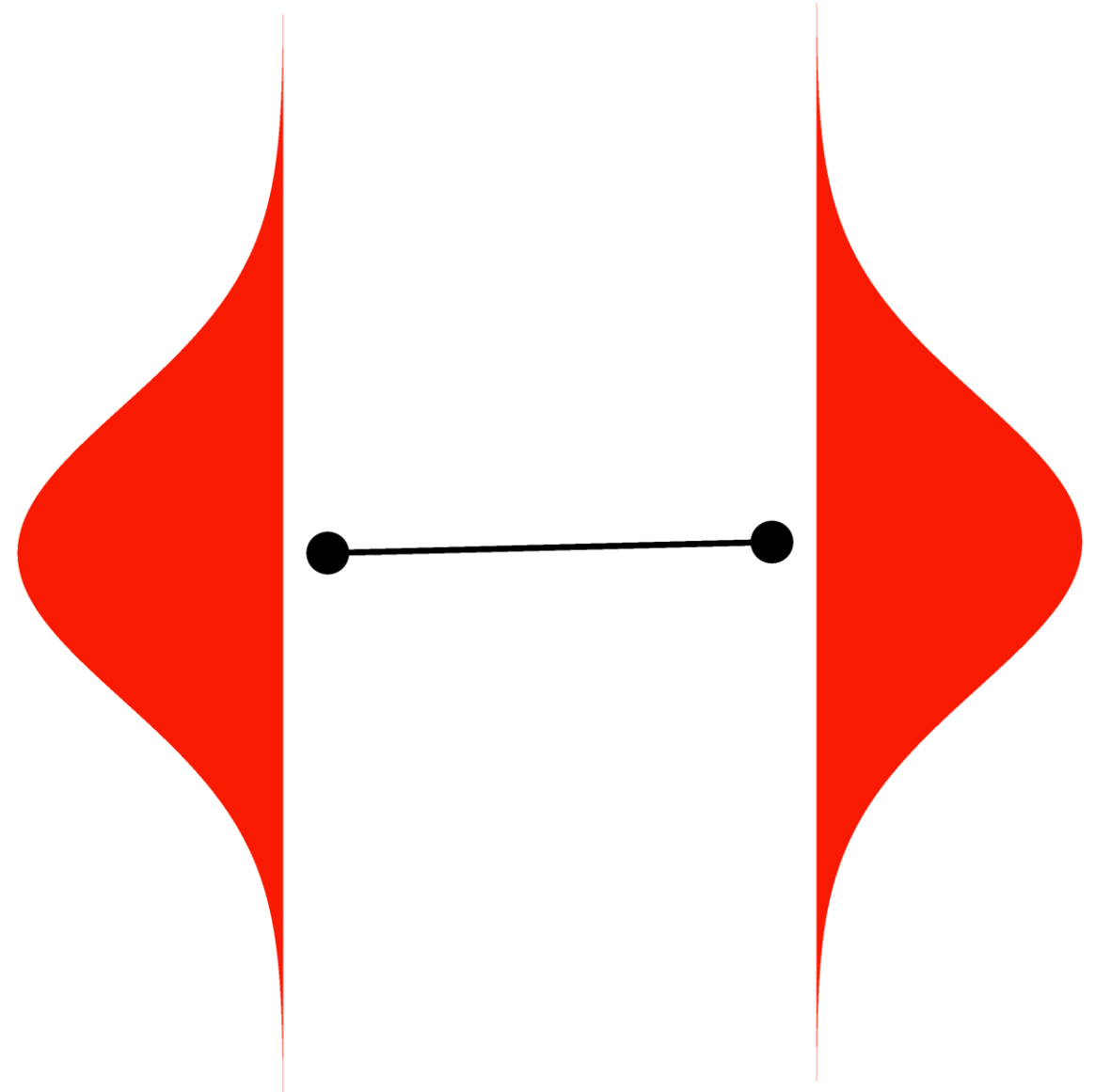


**Item-Wise  
Matching**

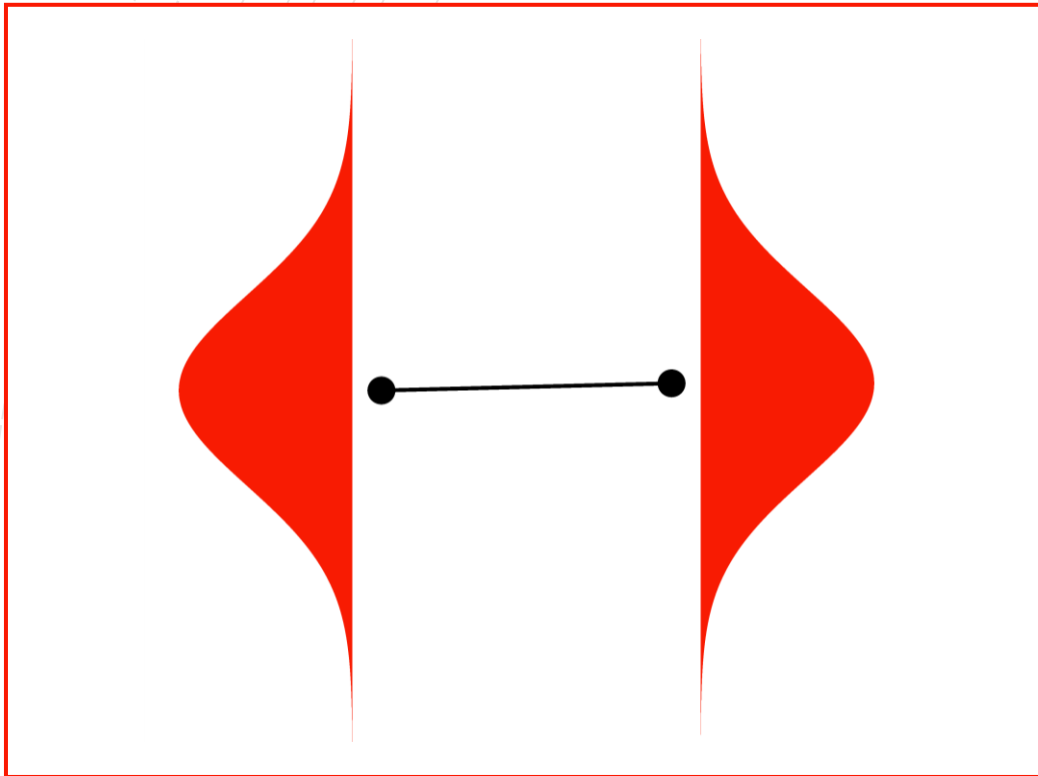


## Part 2/4

# Distribution-Wise Approaches to Matching



# Distribution-Wise Matching



- The full distribution in one condition is matched with the full distribution in the other.
- Results in independent conditions which are similarly distributed on control variables.
- Can control how precise this matching is.

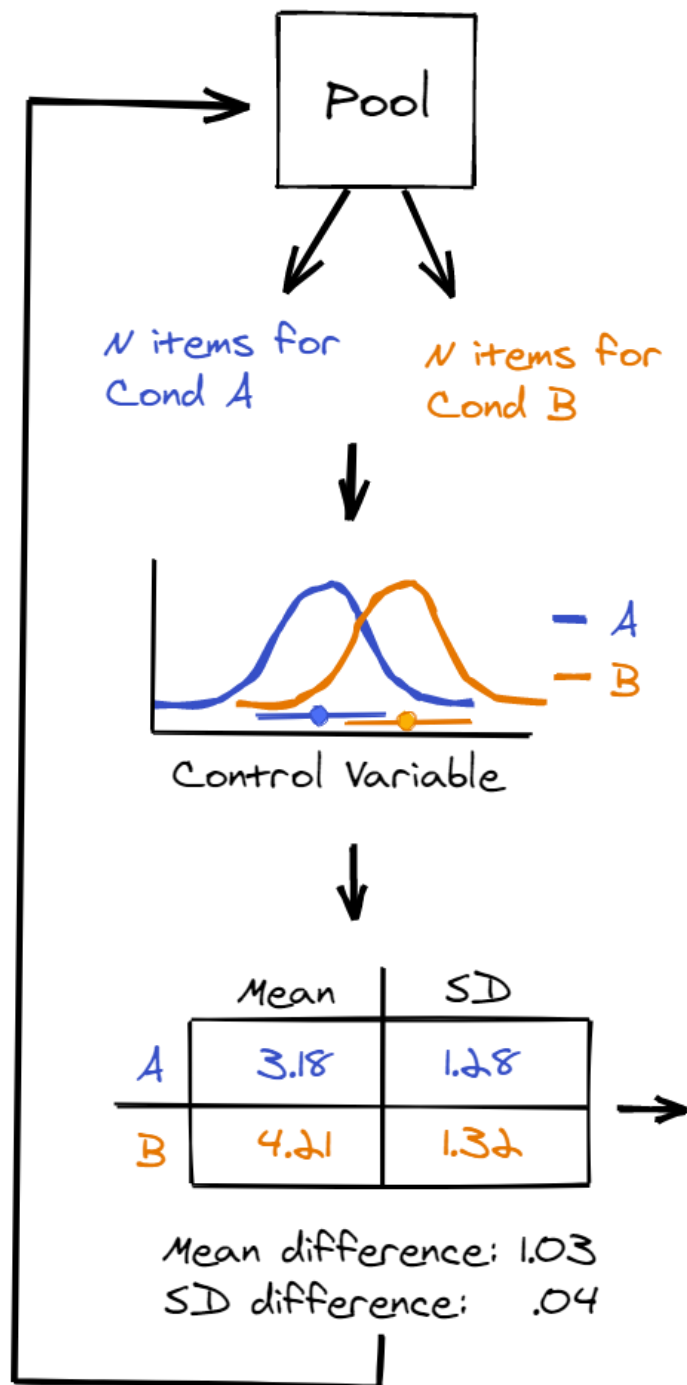
# Distribution- Wise Matching Manually

How are people currently matching distributions?

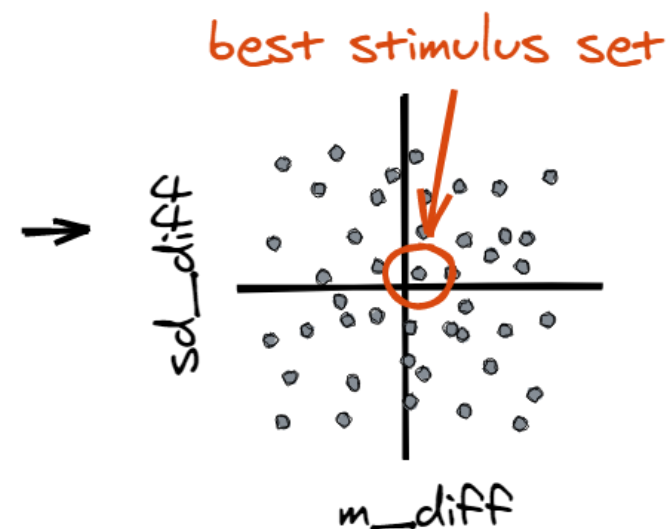
1. Identify/create databases with the variables you need.
2. Identify filters and apply.
3. Identify conditions, and get lists of items that fit each condition.
4. Try to pick items which balance each other out across the conditions.
5. Check the conditions are similarly distributed using a statistical test or comparing means and SDs.

## DISTRIBUTION-WISE MATCHING WITH CODE

- Randomly pick a sample of items which fits the conditions.
- Calculate a summary statistic for how close the samples are
- Store the result and a seed to recreate the stimuli
- Repeat the process many times
- Select the best stimulus (e.g., smallest difference in mean and SD)



iter	seed	m_diff	sd_diff
1	42	1.03	.04
2	49	<del>1.03</del>	<del>.04</del>
3	45	<del>1.03</del>	<del>.04</del>
...	...	...	...

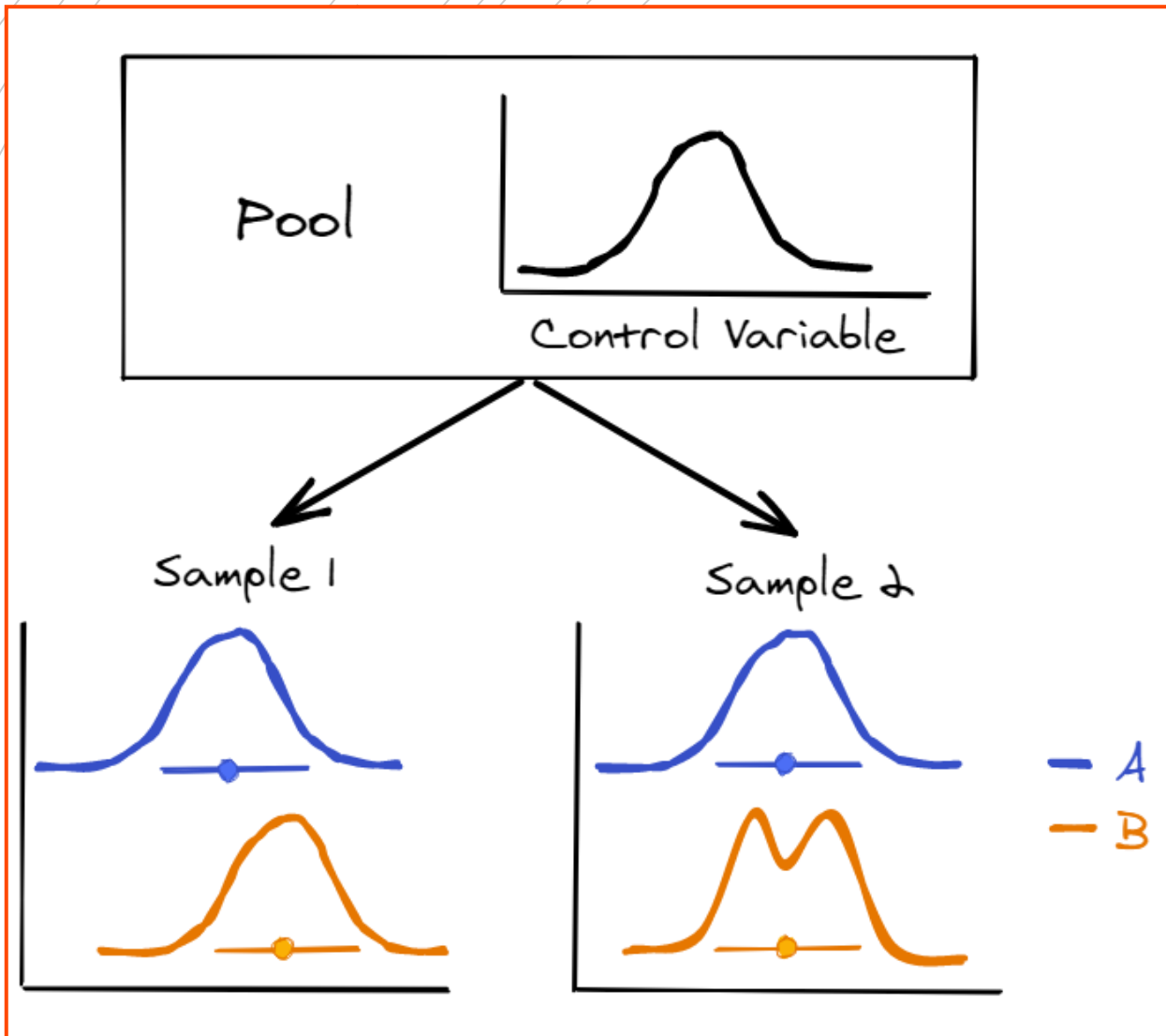


What's wrong with matching on distributional parameters?

- Need to specify a distribution that describes the variable well
- Variable might not have that distribution in a random sample anyway
- Could add a check for appropriateness of the distribution assumptions

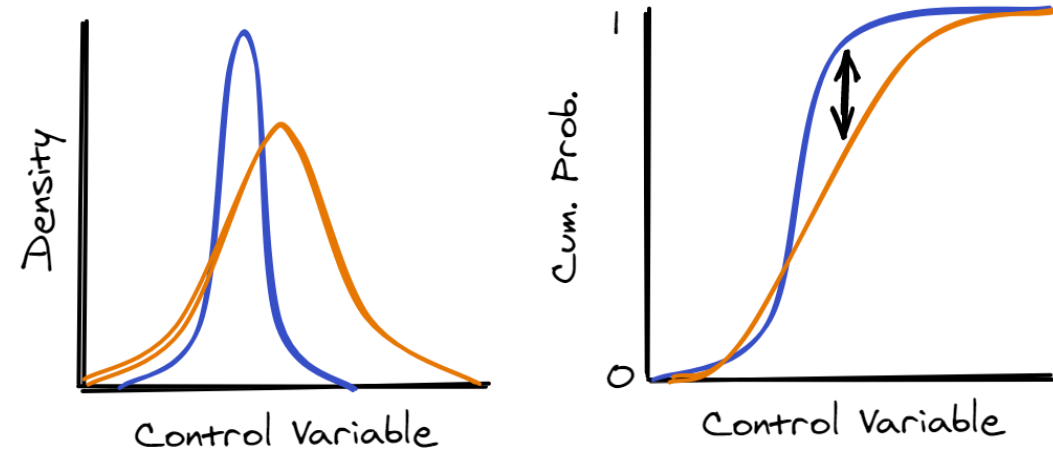
Or...

Could use an assumption-free measure of distributional similarity

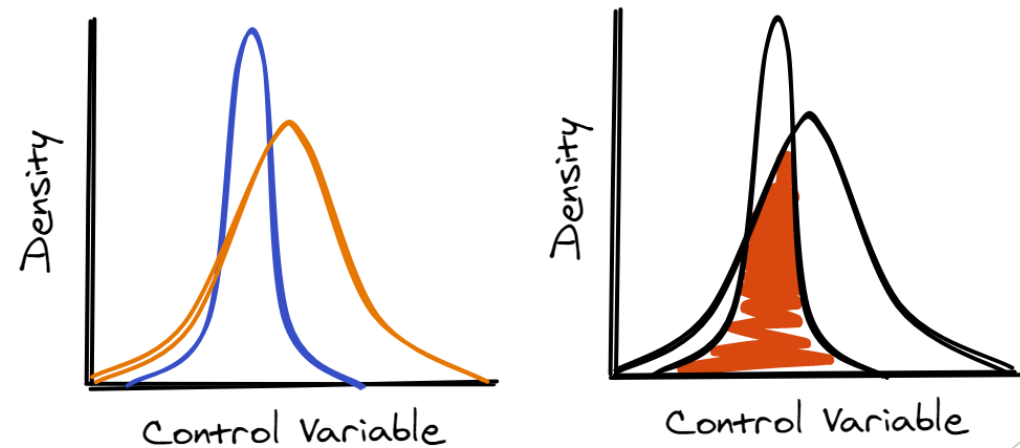


# Assumption-Free Measures of Distributional Similarity

Kolmogorov-Smirnov Statistic (Kolmogorov, 1933; Smirnov, 1948)



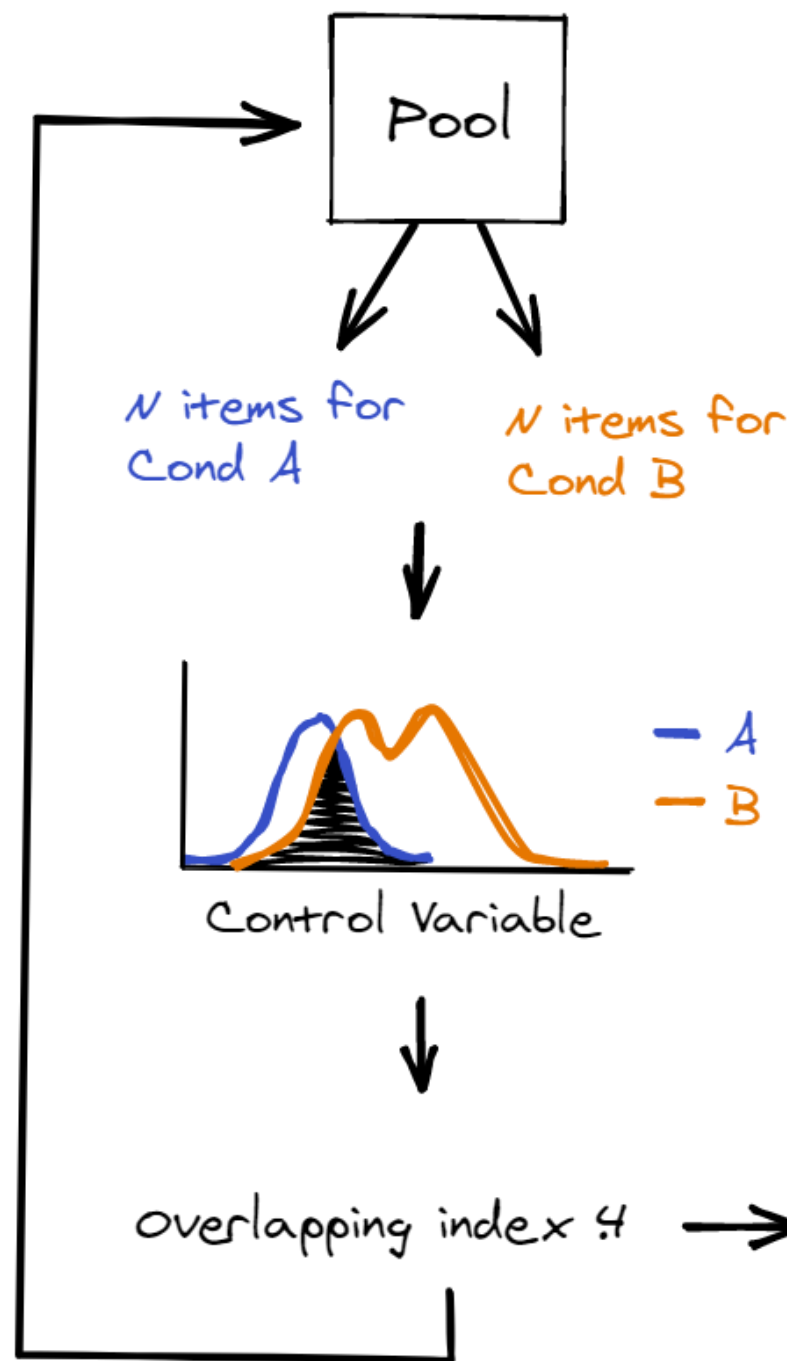
Overlapping Index (Pastore & Calcagni, 2019)



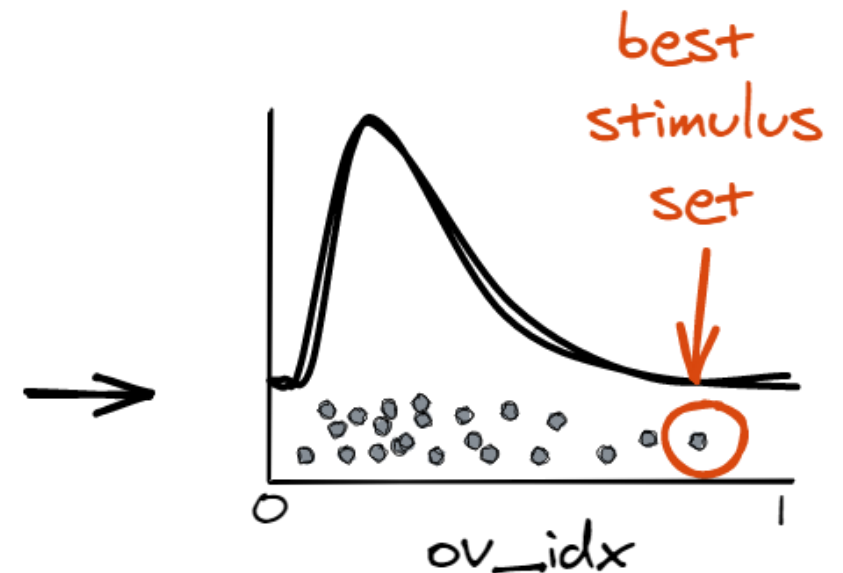


## ASSUMPTION-FREE DISTRIBUTION-WISE MATCHING

- Randomly pick a sample of items which fits the conditions.
- Calculate a summary statistic for distributional similarity
- Store the result and a seed to recreate the stimuli
- Repeat the process many times
- Select the best stimulus (e.g., largest overlapping index)



iter	seed	ov_idx
1	42	4
2	49	<del>4</del>
3	45	<del>4</del>
...	...	...



▼ A distribution-wise  
example with code

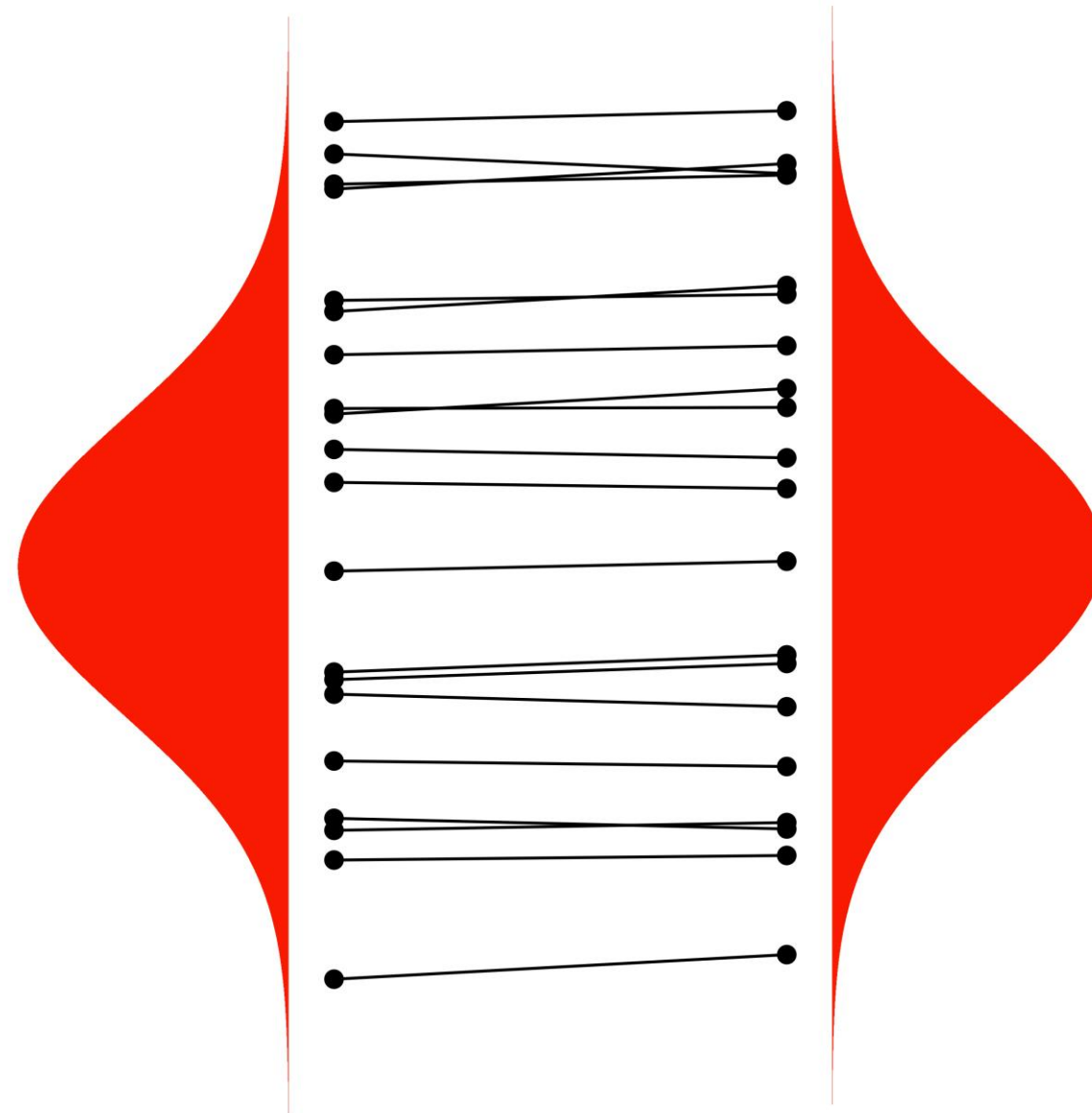


What else could we implement with distribution-wise matching?

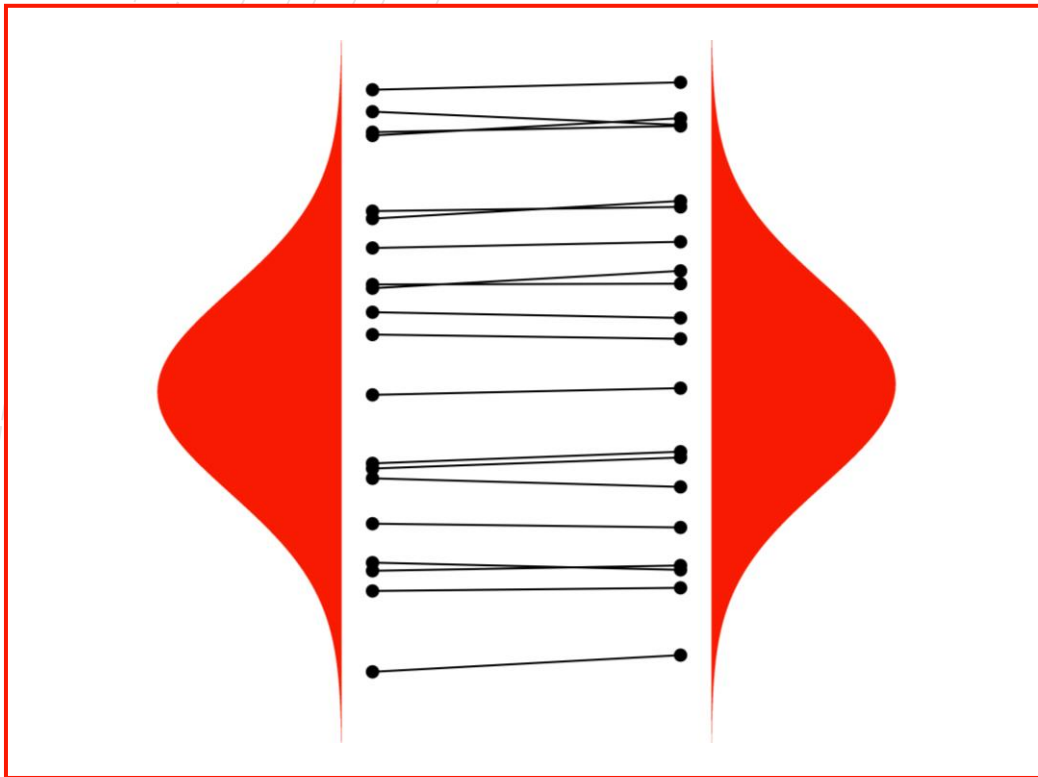
- Filtering the pool
- Matching multiple variables simultaneously
- Weighting distributional similarity
- Maximising representativeness as distributional similarity to the pool

Part 3/4

# Item-Wise Approaches to Matching



# Item-Wise Matching



- Each item is matched with one item in every other condition.
- Results in items across conditions that are directly comparable to one another.
- Can control how precise this matching is.
- Distributional similarity is a necessary by-product of item-wise matching.
- Degree of distributional similarity is defined by the precision of the matching.

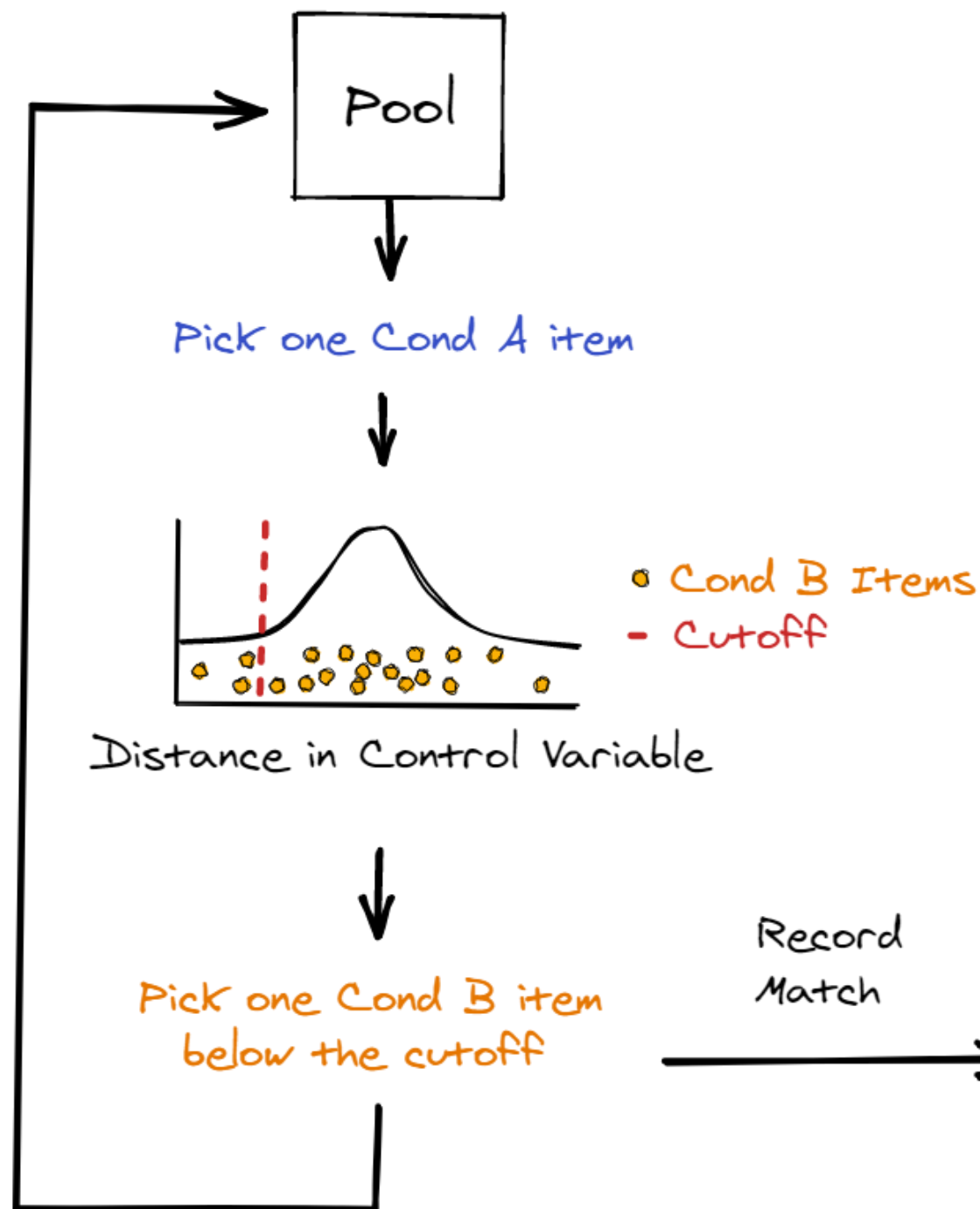
## Item-Wise Matching Manually

How are people currently matching items?

1. Identify/create databases with the variables you need.
2. Identify filters and apply.
3. Identify conditions, and get lists of items that fit each condition.
4. Pick an item in either condition.
5. Try find an item in the other condition which is closely matched in the control variables.
6. If you find a match, make note of it and remove both items from the pool.
7. Repeat steps 4 – 6 until you have as many stimuli as you want.

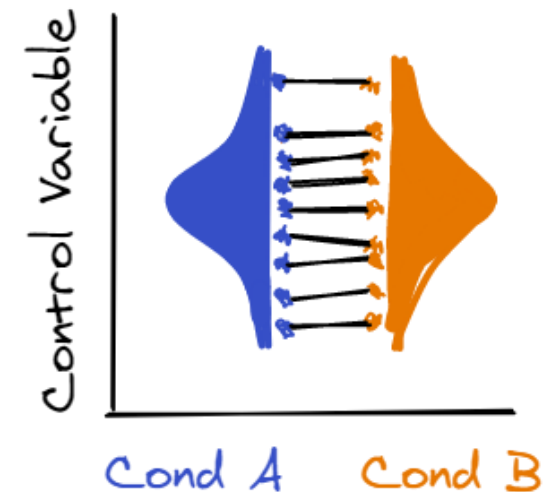
## ITEM-WISE MATCHING WITH CODE

- Randomly pick an item from one condition.
- Try find one item from every other condition which is suitably close in the control variables.
- If you find a match, record it.



Record  
Match

A	B
20	56
<del>20</del>	<del>49</del>
<del>20</del>	<del>49</del>
...	...





<https://github.com/JackEdTaylor/LexOPS>





- An R package for generating item-wise matches in a reproducible way.
- Similar in style to the tidyverse, focusing on readability.
- Some R knowledge required, but there is a Shiny app for users less familiar with R.

# Three main functions in LexOPS

1

`split_by()`

specify  
independent  
variables

2

`control_for()`

specify control  
variables

3

`generate()`

run the matching  
algorithm

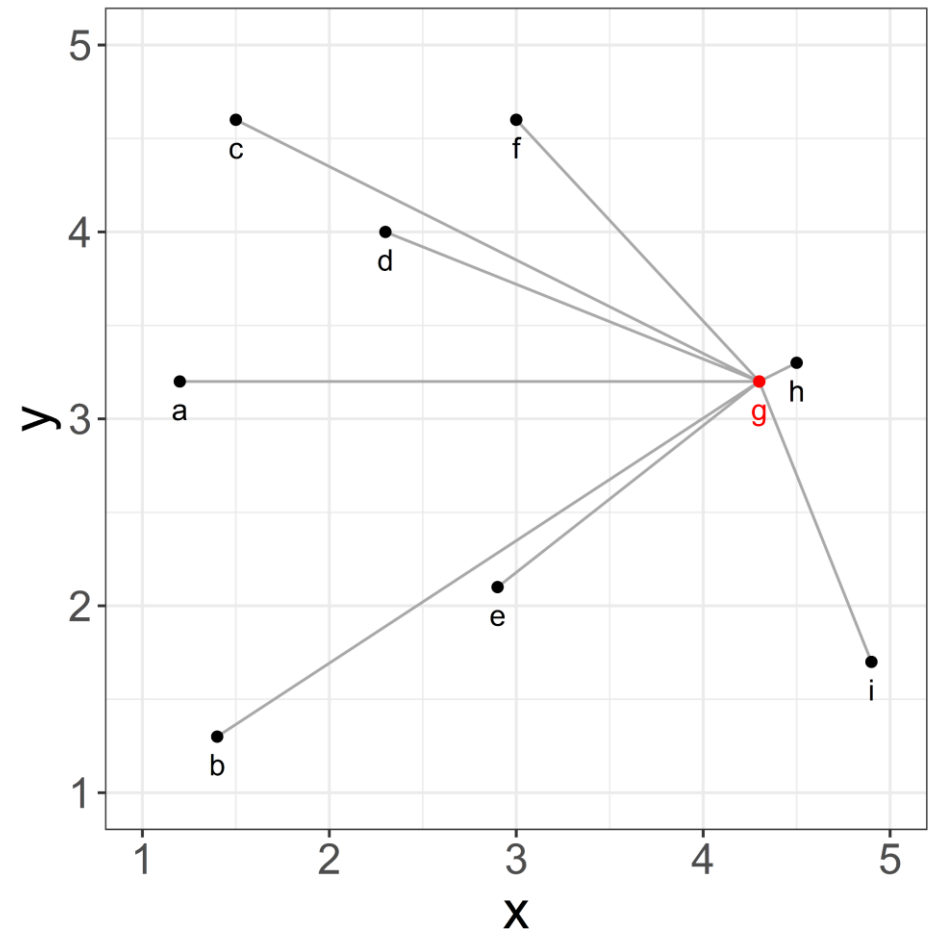
▼ An item-wise example  
with code



# EUCLIDEAN DISTANCE

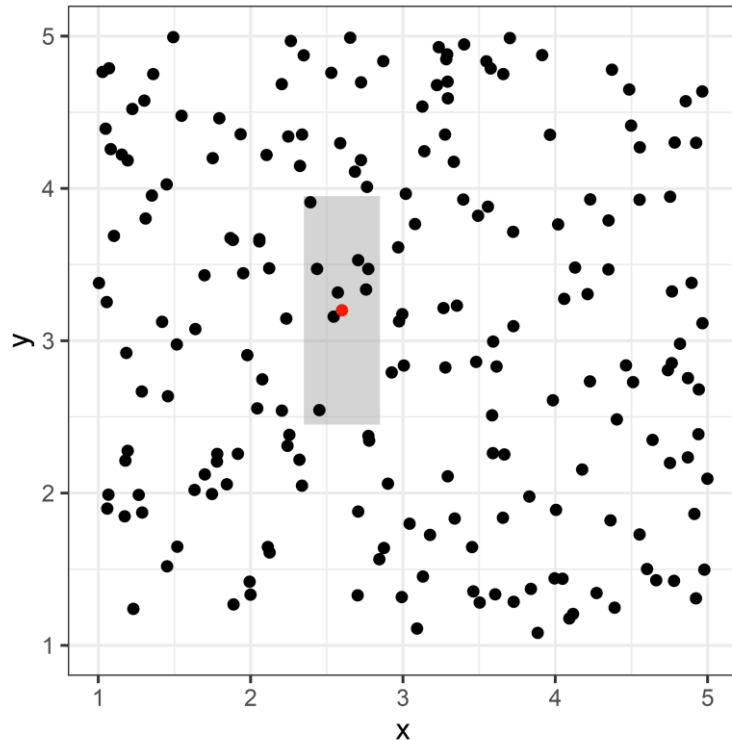
- Can define an  $n$ -dimensional space, where each dimension is an individual numeric variable you want to control.
- Categorical variables can be recoded into  $k-1$  dummy-coded variables (e.g., to values of 0 and 1), where  $k$  is the number of categories.
- Scale variables to have equal weighting.
- Can weight dimensions by the relative importance of their variables being matched,  $w_i$ .
- Can then identify items from each condition which are suitably close to each other in this space.

$$d(a, b) = \sqrt{\sum_{i=1}^n (w_i \cdot (a_i - b_i))^2}$$



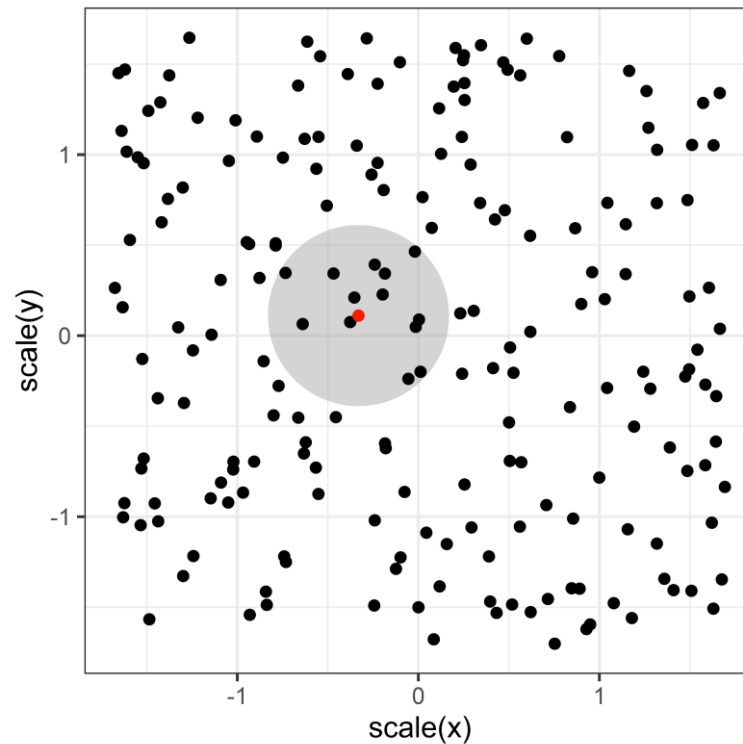
## VARIABLE-SPECIFIC TOLERANCES

```
df %>%  
  split_by(iv, "a" ~ "b") %>%  
  control_for(x, -0.25:0.25) %>%  
  control_for(y, -0.75:0.75) %>%  
  generate(1)
```



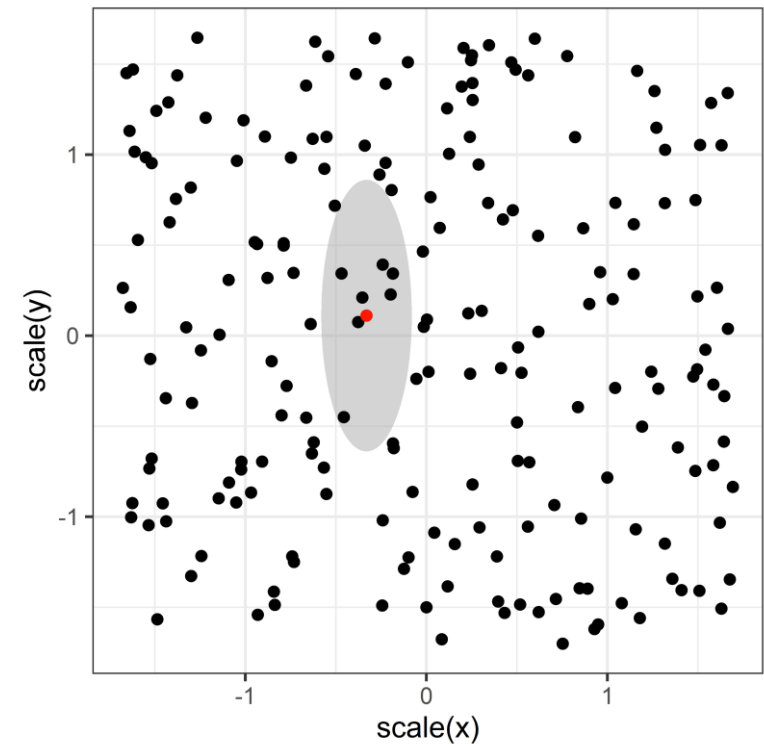
## UNWEIGHTED EUCLIDEAN TOLERANCES

```
df %>%  
  split_by(iv, "a" ~ "b") %>%  
  control_for_euc(  
    c(x, y), 0:0.5  
  ) %>%  
  generate(1)
```



## WEIGHTED EUCLIDEAN TOLERANCES

```
df %>%  
  split_by(iv, "a" ~ "b") %>%  
  control_for_euc(  
    c(x, y), 0:0.5,  
    weights = c(3, 1)  
  ) %>%  
  generate(1)
```



▼ An example with code



## Other Useful LexOPS Functions

- `control_for_map()` – Define higher-order controls which are calculated each iteration with a function. Lets you control for more complicated variables like semantic similarity.
- `split_random()` – Perform a random split rather than splitting on a variable. This is useful for within subjects designs which alter things other than stimuli, e.g., task effects.
- `plot_sample()` – See how representative the generated stimuli are on the variables in your design.
- `plot_iterations()` – Diagnose the algorithm's results. Shows the cumulative count of items generated over all iterations.
- `cite_design()` – List the variables you've used which probably need describing and citing when you write up your methods.
- `run_shiny()` – Run the LexOPS shiny app. This is a friendly GUI with lots of useful visualisation, which can translate options to reproducible LexOPS code.



<https://github.com/JackEdTaylor/LexOPS>



## SOFTWARE PACKAGES FOR ITEM-WISE MATCHING

Package/Library	Language	No. Conditions	Type
designmatch	R	2	Optimal propensity score matching
Matching	R	2	Optimal propensity score matching
MatchIt	R	2	Optimal propensity score matching
LexOPS	R	2- $\infty$	Tolerance matching
optmatch	R	2	Optimal propensity score matching
causalinference	Python	2	Optimal propensity score matching
matchpairs() function	MATLAB/OCTAVE	2	Optimal pairwise distance

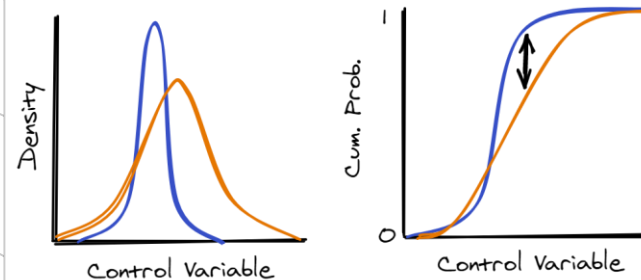
# Propensity Score Matching

- **Propensity Score:** conditional probability of being assigned to a treatment (i.e., binary outcome), given a vector of covariates
- Pairwise matching of propensity score across treatment and control ensures conditions do not differ in covariates
- Can be used as:
  - Method for generating samples before an experiment
  - Method for accurately estimating effect size in observational study

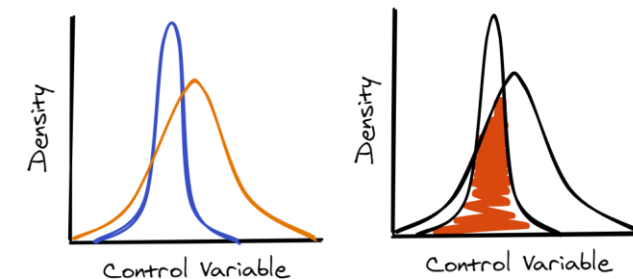
Part 4/4

# Rounding Up

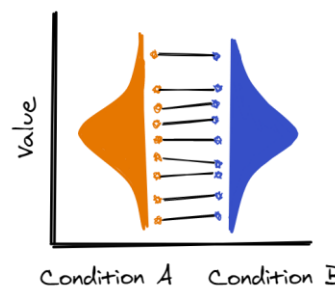
Kolmogorov-Smirnov Statistic



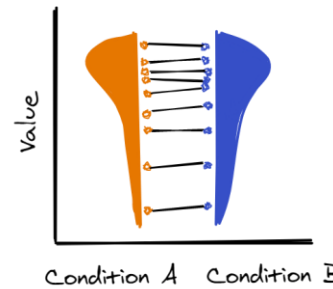
Overlapping Index



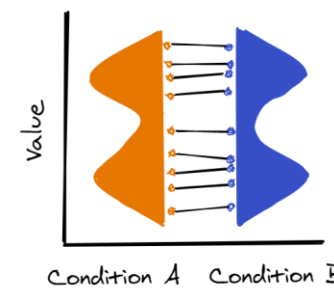
Control 1



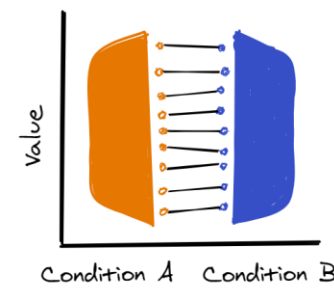
Control 2



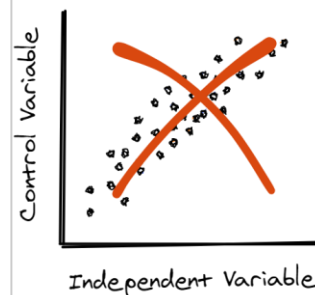
Control 3



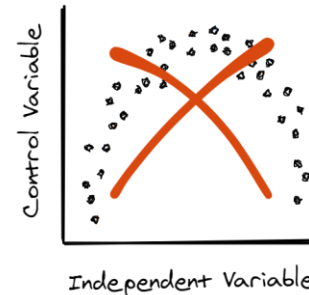
Control 4



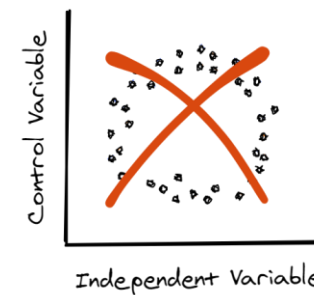
Sample 1



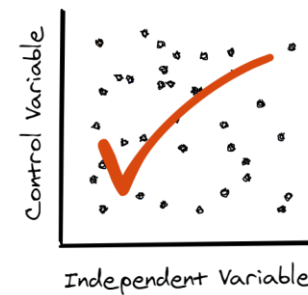
Sample 2



Sample 3

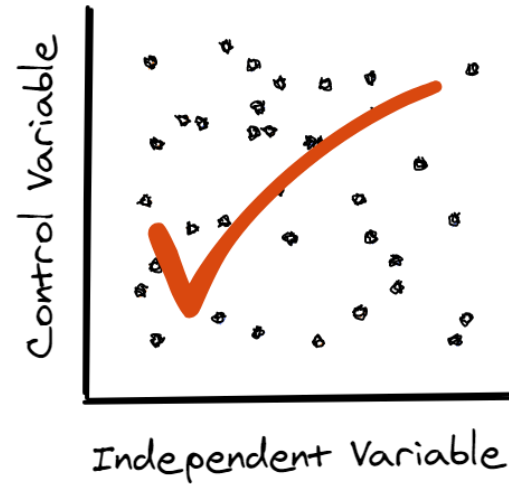
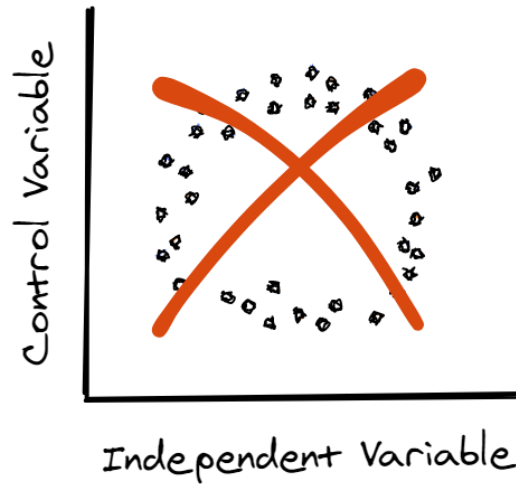
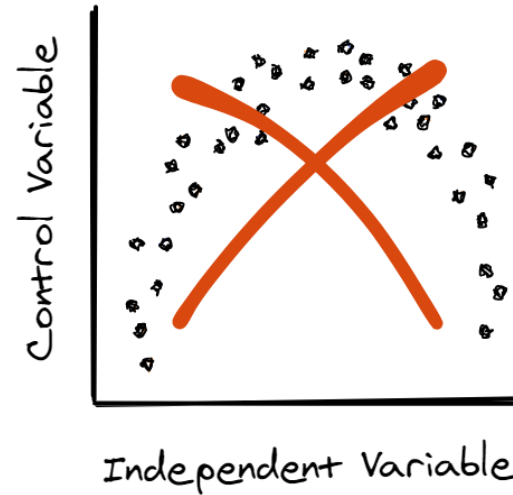
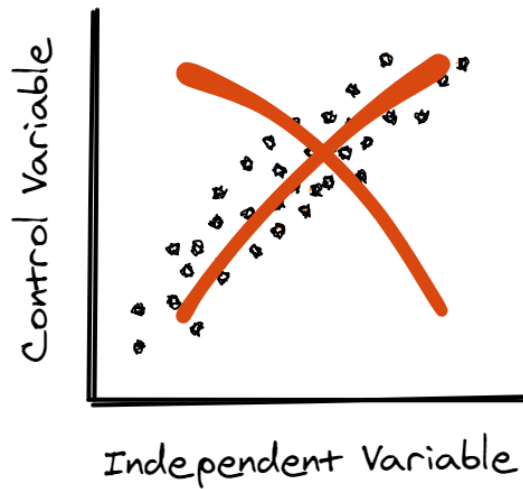


Sample 4



# Controlling for Variables when IVs are Continuous





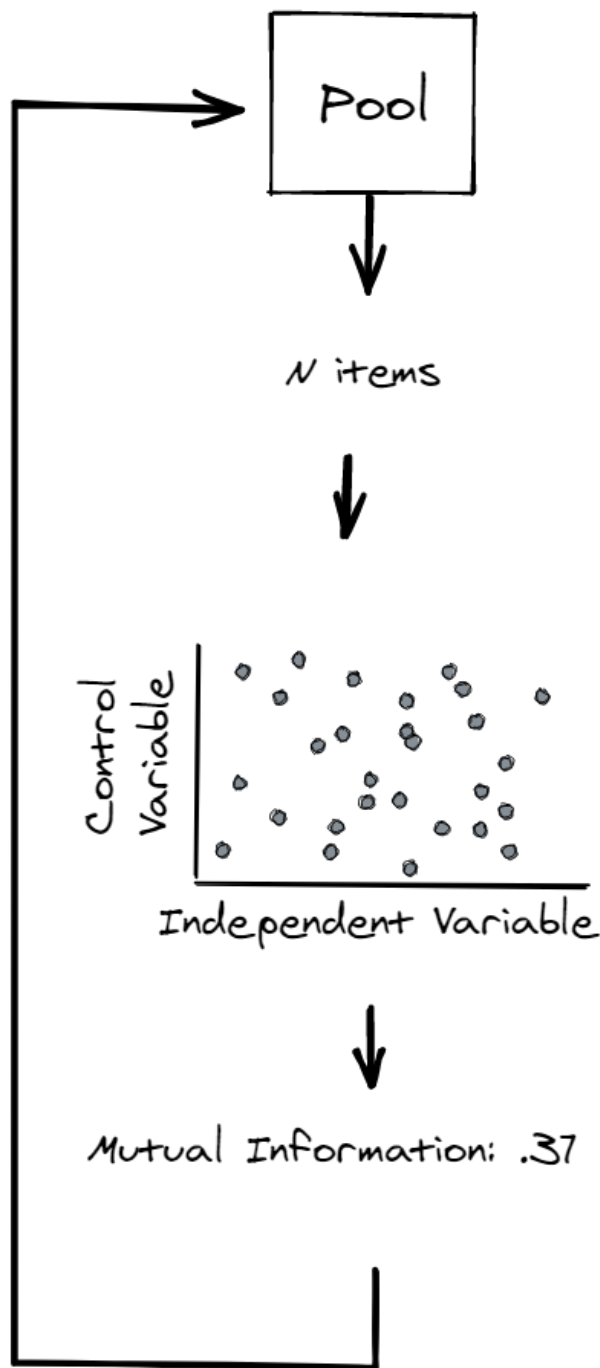
## Controlling for Variables when IVs are Continuous

- Unnecessary binning of continuous variables can reduce sensitivity to effects
- Can just use continuous predictors
- Would still want to minimise the influence of confounding variables

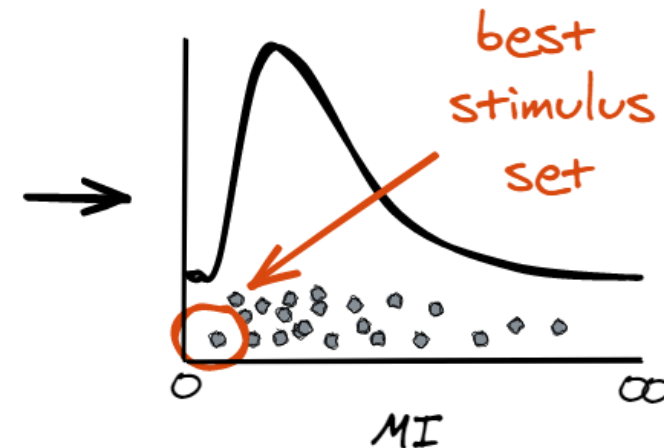
One possible solution:  
**minimise mutual information**

# MINIMISING MUTUAL INFORMATION WITH CODE

- Randomly pick a sample of items.
- Calculate the mutual information between the predictor and control variable
- Store the result and a seed to recreate the stimuli
- Repeat the process many times
- Select the best stimulus set (i.e., smallest MI)



iter	seed	MI
1	42	.37
2	49	<del>.37</del>
3	45	<del>.37</del>
...	...	...



# Reporting Methods of Designing Stimuli



## Reporting Methods of Designing Stimuli

- Where did your pool of candidates come from?
- Where did you get the data for the variables from?
- What kind of matching/controlling did you use (item/distribution/continuous)?
- How many iterations did you run?
- How closely are items matched?
- How many items did you create?
- Can you visualise it?
- Where is the code?
- Even without the code, could someone reproduce your methods just from your description?



## What not to do when reporting stimulus design

---

“Stimuli comprised 100 male faces and 100 female faces. Conditions were matched in terms of face width, face height, and age. Independent sample  $t$  tests confirmed that the conditions were suitably controlled (all  $p$  values  $> .05$ ).”

- *Foo et al. (2005): Revisiting the Effect of Bar*

---

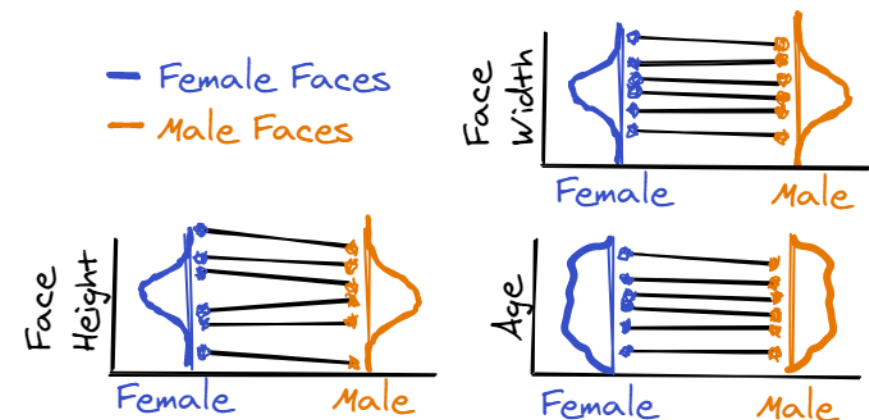
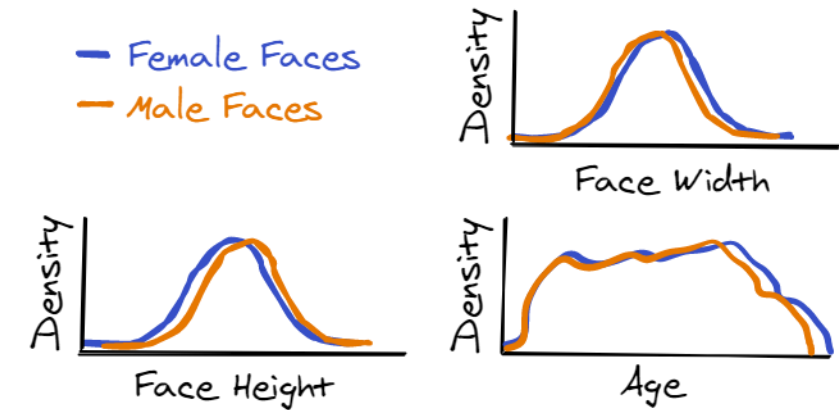
“Stimuli comprised 100 male faces and 100 female faces. Items were matched pairwise in terms of face width, face height, and age. Paired sample  $t$  tests confirmed that the conditions were suitably controlled (all  $p$  values  $> .05$ ).”


- *Foo et al. (2003): The Effect of Bar on Face Recognition*

# What to do instead

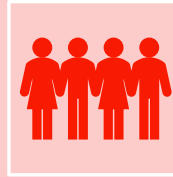
“Stimuli comprised 100 male faces and 100 female faces from the Chicago Face Database (Ma et al., 2015). The two conditions were matched on the distributions of face width, face height, and age by maximising the overlapping index (Pastore & Calcagni, 2019) between the two conditions. Overlapping index values were calculated using the overlapping package (Pastore & Calcagni, 2019) for R (R Core Team, 2020). Of 30000 random samples, the sample with the maximum overlap in all variables, between the two conditions, was selected. The full list of stimuli, and the code to generate it (with a seed for reproducibility) is available at [osf.io/wdbiw](https://osf.io/wdbiw). The selected stimuli are presented in Figure 1.”

“Stimuli comprised 100 male faces and 100 female faces from the Chicago Face Database (Ma et al., 2015). Faces were matched pairwise on the variables of face width (within  $\pm 1$  pixel), face height (within  $\pm 5$  pixels), and age (within  $\pm 0.5$  years), using the LexOPS package (Taylor et al., 2020) for R (R Core Team, 2020). The full list of stimuli, and the code to generate it (with a seed for reproducibility) is available at [osf.io/wdbiw](https://osf.io/wdbiw). The selected stimuli are presented in Figure 1.”

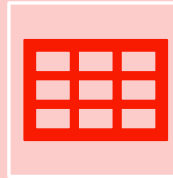


Applications to  
things other   
than stimuli

# Applications to things other than stimuli



Selecting matched participants from a pool for a between-subjects design



Analysis of megastudy data where you only analyse matched subsets of the data (common use of propensity score matching)



Dating?

# Summary, Questions, Answers

- Design stimuli with code, and share the code, to be reproducible and save time
- Report what you did specifically, and show the data with a visualisation



Sara  
Sereno



Guillaume  
Rousselet



Alistair  
Beith