Bachelor's Thesis

Grau en Enginyeria en Tecnologies Industrials

Design and implementation of an App for the analysis of kansei engineering data

REPORT

Author: Ferran Gebellí Guinjoan
Supervisor: Lluís Marco Almagro

Date: June 2017

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Abstract

The main objective of the project is to create a web App to interactively display the analysis of data from kansei engineering studies. Kansei engineering is a tool used in emotional product design and aims to discover which product features convey a specific set of emotions and feelings.

The project will include the development of analytical tools and the creation of graphics to visualize results. The application will be programmed with the language R together with the extension Shiny, which will create an easy to use interactive platform. The application is primarily directed to the sector of product design, where statistical knowledge is often scarce, so results are displayed in a visually appealing and easy to interpret manner.

The App obtained has been modularised, pursuing a structured code and the possibility to expand it easily in the future. After the upload of data from any kansei engineering study, it helps the user to analyse it. It gives all information necessary to know which product features affect which emotions and gives a measure of how much.

The App has a sequential workflow, and the last step gives the user a suggestion of the features that the product should have, according to his or her requirements. This has been achieved through a multicriteria optimization model that has been specially adapted for kansei engineering studies. The App has also some further settings that can be modified, for example it offers the possibility to stratify all the results (say, by gender), if this information was given in the preliminary data set.

INDEX

1. Introduction

This first chapter will briefly explain how I got involved with the project, but also sets its basis and define both goals and limits.

1.1. Origin of the project and motivation

Curiously, I had my first approach to this project not by its topic, kansei engineering (of which I had never heard), but by the tools and methodology used: R programming language and its package Shiny, capable of creating interactive apps. In my last semester, I had been enrolled in the course *Project II*, which consisted in exactly that: creating a Shiny app. This was my link to the Department of Statistics and Operational Research but also my link to start this project.

Once I came in contact with this project, there were various reasons that made me decide that it would be my bachelor's thesis.

First of all, I had found really interesting the *Project II* subject. I already knew that I liked programming, but discovering R was "surprising" for me. Together with RStudio, I found out that it can be very user friendly, and that both deep and really good looking results could be obtained in a short time.

Moreover, R language is specially designed for data science, a topic that I also really like. Indeed, the two subjects that I had done about statistics in my bachelor's degree had fascinated me. How real information can be revealed from a bunch of data that seems random at first sight really made an impression on me. This knowing out of nothing appeared as something that I find exciting. And one of the techniques to achieve that is design of experiments (DOE), one of the tools that I especially liked, because it goes straight forward to reality, taking some measurements of variables to obtain the best combination that optimizes the solution and giving information on how each variable affects a response. Since kansei Engineering is highly related with DOEs, this was translated into more interest on the subject of this project.

Finally, there was another factor that gave me extra motivation: obtaining a scholarship to collaborate with the Department of Statistics and Operational Research. That meant that I could dedicate more hours to this project, make it complete and for certain with better results.

As previous requirements, it was crucial for me to know the basics of R programming, and the usage of its Shiny package. That prevented me to be stuck for too much time in the first stage of any project: the search of information and learning of the basics of the topic and tools to use.

Of course my statistical background was also important, indeed it was crucial for this project. It helped me to figure out what should be displayed on the App in order to lead to a correct analysis and interpretation, but it also provided me the tools to implement all the calculations that lay behind the results.

1.2. Objectives of the project

It is very important for any project to have clear purposes from the beginning and know what is expected to be accomplished, in order to prevent losing efforts while working in directions that may lead to unnecessary or unwanted results.

This is obviously a study on kansei engineering, but it still bears a great difference from other works in this field. While most of kansei engineering research try to find out which statistical methods are better suited or which is the best methodology to obtain and treat the data, this project is focused on creating a tool that enables a clearer analysis of the results.

Some discussion and bibliographic research to determinate which are proper methods and models to obtain results from the raw data will be carried out in Chapter 3.3. But the focus will be most of the time on what is the main target of this thesis: the displaying of results, how can that help to a better interpretation and the way to implement it. Objectives have been especially designed for that. They have been divided between general (concerning the project, Table 1) and specific (directly related to the App, Table 2):

Table 1. General objectives for this project

General objectives	Provide a tool to analyse interactively data from kansei engineering studies
	Display clear results difficult to misunderstand
	Write a clear code structure easy to read and extend
	Determinate a multicriteria optimization method for kansei engineering

Table 2. Specific objectives for this project (App requirements)

Specific objectives	Stratification possible
	User friendly
	Real time response
	Allow to modify main variables
	Multiplatform

1.3. Scope

Already knowing that the main goal is not finding the best models and methods to treat the data, but the creation of an interface to analyse the results, other limits must be set, to have the project perfectly delimitated.

First of all, the App will not be designed for the data acquisition stage. That means that the user will have to upload the data, and this data needs to be in a certain format. Otherwise the App will not work, unable to understand the data provided.

The format required for the data, and also the limitations that this implies, will be explained in detail in Chapter 5.1.

The main potential users of the App must also be stated: those are people working in the design field, more precisely in the design of consumer goods or in the marketing areas. Now the reason for displaying results that cannot be misinterpreted seems clearer, since those users do not necessarily have statistical knowledge.

For this reason, although all the results will be calculated with advanced statistical tools, the app will only show the most straightforward interpretation of them. As a contra, this won't give the user a full control to change all parameters and to do a deep statistical analysis, but guarantees that the target public can understand the results of the App.

2. Kansei engineering: introduction and model

This section introduces the concept of kansei engineering, following the model proposed by Lluís Marco in the doctoral thesis *Statistical Methods in Kansei Engineering studies* [1].

2.1. An introduction to kansei engineering

Over the last decades, users of products and services have become more and more demanding. People do not only want products that satisfy our needs and work fine, but also products they like.

For many years, designers in general did not pay much attention to the emotional needs of customers, and only translated technical functionalities into parameters. However, this emotional aspects related to products or services cannot be eliminated from the equation. The massive success of some emotional products (such as the iPod, for example) has confirmed this tendency.

Once it has been decided and proved that the emotional part of a product should be taken into account, the problems arise when thinking how to incorporate it to the design of a product. Relying on the designer's intuition and creativity has been the most used option traditionally, but there are qualitative and quantitative methods to find information on how users perceive and use products and services. Most of these methods are grouped under what is called "emotional design".

Qualitative methods for emotional design, although they can give a lot of information, normally have several difficulties:

- Results from qualitative approaches depend a lot on the person leading the focus group or performing the interview.
- Qualitative approaches, especially interviews, require a lot of time. So usually only a few interviews are performed and conclusions are derived from asking a small amount of people.
- It can be difficult to obtain product design guidelines due to the fact that users are not typically thinking in a designer's paradigm.

Quantitative approaches such as questionnaires eliminate those difficulties but, obviously, have others. The main one, probably, is that they are by definition reductionists and, therefore, limited.

A quantitative method used in emotional design and mainly based in questionnaires is kansei engineering (KE). Kansei engineering's most important features are:

- In a kansei engineering study, the aim is to connect the physical properties of an object with emotions.
- In kansei engineering, there is an attempt to describe the whole range of emotions a product can convey. Not a unique response is modelled (such as the elements that make people prefer a watch over the others, for instance), but several responses (such as the elements that provoke that people perceive a watch as being modern, and elegant, and reliable, and so on). There are as many responses as necessary concepts to cover the whole range of expected emotions.
- Another important feature of KE studies is that they are based on collecting and analyzing quantitative data. Statistical or computational methodologies are usually used to find the relationship between properties and emotions.

2.2. Kansei engineering study model

A model to carry on a KE study on a product can be divided in several steps:

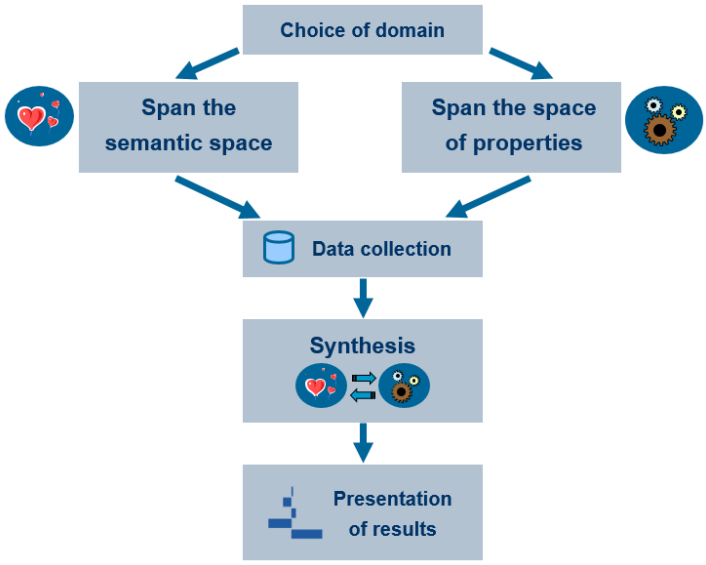
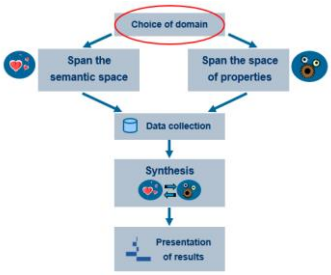


Figure 1. Graphic of the workflow of a Kansei study

- In the first place, the product to be analyzed has to be described, as well as the people to whom it is addressed and the market situation.
- Secondly, the semantic space has to be defined. This means collecting words that can describe emotionally the product. The initial amount of words (normally relatively large) is then reduced, containing what is called the kansei words.
- After that, it is necessary to define the space of product properties. Some design attributes are chosen, and several possible values are considered for each attribute. Prototypes (the products for the KE study) are then built (either physically or in a graphical representation)
- Data is then collected, normally asking a group of people using questionnaires.
- In the synthesis stage, statistical methodologies are used to relate the space of properties to the semantic space. For every kansei word, product properties that affect it are found.
- Results are then presented in a visual manner, easily palatable for designers and technicians with no statistical skills.

Next sections of this chapter will provide a practical example of a kansei engineering model. Each of the steps shown in Figure 1 will be clarified using a real kansei study example (the kansei study done in Lluís Marco's thesis [1]).

2.2.1. Choice of domain



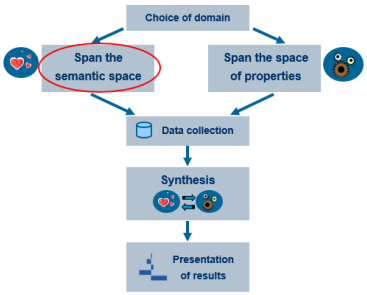
Choosing the domain obviously includes deciding which product is the protagonist of the study. In this example, the chosen product for the KE study are fruit juices, and specifically its presentation just before being drunk. But this is not the only task, it also requires:

- Defining the target group to which the product is addressed. The example had a rather heterogeneous scope: from 24 participants, there were 13 women and 11 men, with ages ranging from 17 to 59, and different educational levels (from high school to post-graduate studies).
- Defining the kind of presentation for the product. Photographs were employed to gather the ratings on several kansei words. It was thought it would be enough since the interest was in the visual impression, but one could argue that this only gives a

narrow affective channel, as no smell or taste is involved.

- Defining the context for presentation. The atmosphere of the place where the experiment is conducted can have an effect on the emotions elicited by the product, but in this case this fact was not considered, and probably presenting photographs makes the study more robust to potential effects derived by the context of presentation.

2.2.2. Span of the semantic space



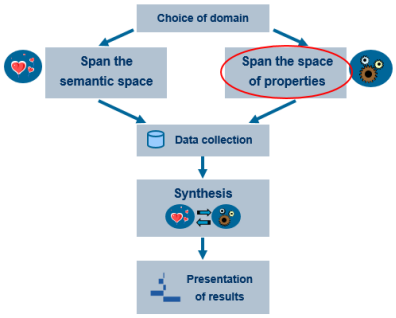
Explained briefly, spanning the semantic space means choosing which kansei words are going to be used in the study. That is the same as finding the exact words to express the emotions to be observed, which is not always an easy task. It comprises three steps:

1. Setting an initial list of kansei words. In the juices experiment the 134 words shown in Table 3 were proposed. These words constitute the initial semantic space.
2. Reducing the initial list of kansei words. First, an affinity diagram was used to group kansei words. Each word was written on a yellow post-it, which was grouped with all words similar in meaning obtaining 33 groups. Each group was then labeled choosing the most representative word in the group. This word was written on a blue post-it and again, groups were made with all blue post-its to further reduce the number of words, and a representative word was chosen for each group and written on a pink post-it. The initial semantic space was reduced to only 14 words. Later, a cluster analysis was conducted to decide on the final kansei words. This required the collection of data, and 6 people were asked to give a rating for 4 juices on the 33 kansei words obtained. A k-means clustering was then performed to modify the groups obtained with the blue post-its. Finally one word from each of the final clusters was chosen to name the cluster.
3. Proposing the final reduced list of kansei words, which in the study were: refreshing, healthy, exotic, seductive, natural, relaxing and tasty.

Table 3. Original list of kansei words in Catalan and English.

Catalan	English	Catalan	English	Catalan	English
àcid	acidic	excel·lent	excellent	refrescant	refreshing
divertit	amusing	excitant	exciting	regenerador	regenerative
antiestresant	anti-stressing	exòtic	exotic	enfortidor	reinforcing
antifatiga	anti-fatigue	explosiu	explosive	relaxant	relaxing
antioxidant	antioxidant	exquisit	exquisite	remineralitzant	remineralising
aphrodisiac	aphrodisiac	fabulós	fabulous	renovador	renovative
atraient	appealing	fashion	fashionable	reconstituent	restorative
aperitiu	appetizer	festiu	festive	restaurador	restorer
aromatic	aromatic	fibrós	fibrous	revitalitzant	revitalizing
artificial	artificial	vistós	flamboyant	ric en ferro	rich in iron
astringent	astringent	floral	flowery	romàntic	romantic
atractiu	attractive	espumós	foamy	saludable	salutary
dolent	bad	fresc	fresh	saciant	satiable
equilibrat	balanced	futurista	futuristic	saborós	savory
beneficiós	benefical	golós	gluttonous	seductor	seductive
amarg	bitter	bo	good	sensual	sensual
tonificant	bracing	gratificant	gratifying	sedós	silky
genial	brilliant	sa	healthy	senzill	simple
calmant	calming	casolà	home-made	aprimant	slimming
caribeny	caribbean	ideal	ideal	suau	soft
nadalenc	christmas spirit	infantil	infantile	sofisticat	sophisticated
clàssic	classical	intens	intense	agre	sour
fred	cold	vigoritzant	invigorating	picant	spicy
colorit	colorful	irresistible	irresistible	estimulant	stimulating
combinable	combinable	sucós	juicy	reforçant	strengthening
còmode	comfortable	juvenil	youthful	fort	strong
concentrat	concentrated	laxant	laxative	substitutiu	substitutive
consistent	consistent	lleuger	light	sabor subtil	subtle taste
corpulent	corpulent	luxós	luxurious	ensucrat	sugary
cremós	creamy	madur	mature	estiuenc	summery
curatiu	curative	embafador	mawkish	dolç	sweet
refinat	dainty	mediocre	mediocre	silvestre	sylvan
decorat	decorated	hidratant	moisturizing	gustós	tasty
profund	deep	natural	natural	temptador	tempting
delicat	delicate	bonic	nice	tropical	tropical
deliciós	delicious	nutritiu	nutritional	lleig	ugly
espès	dense	apassionat	passionate	vellutat	velvety
desintoxicant	detoxifying	agradable	pleasant	versàtil	versatile
digestiu	digestive	popular	popular	vibrant	vibrant
diürètic	diuretic	potent	powerful	vital	vital
diví	divine	preventiu	preventative	vitamínic	vitamin-rich
sec	dry	proteic	protein-rich	càlid	warm
fàcil de digerir	easy to digest	pur	pure	salvatge	wild
energetic	energetic	depuratiu	purifying	amb alcohol	with alcohol
erotic	erotic	fi	refined		

2.2.3. Span of the space of properties



The space of properties lists the physical properties of the product that can have an effect in the elected kansei words. It is similar to choosing factors in a design of experiments. In an experiment conducted in an industrial environment, previous knowledge from the process is used to select the variables (factors) that are more prone to have an effect in the response.

The values that those factors take in the experiment are also carefully chosen to maximize the probability of detecting the factors' effects, if these effects really exist. In this context, each value a factor takes in the experiment is called a level. It comprises the following steps:

1. Making a list of all possible physical product properties and selecting the ones that apparently have the largest impact in the users. In the juices example, a brainstorming session was conducted to produce as many factors as possible for juices and the properties that could impact in the visual perception of the juices. The ones that were easy to modify when taking the photographs were selected, shown in Table 4.

Table 4. Factors and levels in the juices experiment.

Factor	Levels
Straw	Yes
	No
Decoration	Yes
	No
Ice	Yes
	No
Container	Glass
	Goblet
Color	Yellow
	Orange

2. Preparing the design matrix, a matrix that defines how many products will be used for the experiment, and establishes the level of each factor for each one of the products. In the juice experiment, the prototypes were prepared according to the design matrix (in other studies only already existing products are used). The design matrix (Table 5)

is a 2^{5-1} factorial design, which has resolution V. This means that main effects are confounded with interactions of order four and higher, which are normally not relevant.

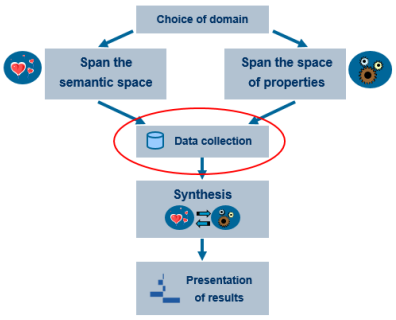
Table 5. Design matrix for the fruit juice experiment

	Straw	Decoration	Ice	Container	Color
1	No	No	No	Glass	Orange
2	Yes	No	No	Glass	Yellow
3	No	Yes	No	Glass	Yellow
4	Yes	Yes	No	Glass	Orange
5	No	No	Yes	Glass	Yellow
6	Yes	No	Yes	Glass	Orange
7	No	Yes	Yes	Glass	Orange
8	Yes	Yes	Yes	Glass	Yellow
9	No	No	No	Goblet	Yellow
10	Yes	No	No	Goblet	Orange
11	No	Yes	No	Goblet	Orange
12	Yes	Yes	No	Goblet	Yellow
13	No	No	Yes	Goblet	Orange
14	Yes	No	Yes	Goblet	Yellow
15	No	Yes	Yes	Goblet	Yellow
16	Yes	Yes	Yes	Goblet	Orange

3. Selecting the products (or producing product prototypes) according to the design matrix. The photos in Figure 2 were created.

Figure 2. The 16 stimulus created according to the design matrix.

2.2.4. Data collection



The data collection phase is very important in a KE study, because if it is not done properly, the results obtained will give false interpretations. For this reason, time needed to complete the survey must be short, in order to maintain the focus of the participants. This can only be achieved by having a maximum number of prototypes scored by each participant

Before starting the actual data collection, a definition of each kansei word was given. This has the problem of rationalizing the procedure even more, but is better than discovering that some words were unclear when data collection is over.

To collect the data, there are two main options: each participant is presented with a product and rates it on all the kansei words or each participant is presented with a kansei word and rates all products on it. The first option was elected for the juices example, because the respondent can concentrate better on the product.

After randomizing all products and words, a three dimensional matrix of data was obtained (Figure 3). Usually, this matrix is collapsed following the grey arrow: the subjects' dimension is lost, and the average for all participants on each stimulus and kansei word is used. However, this is not what we will do in our application, where the idea will be using the raw data coming from each participant, and including the participant as a random factor in a regression model.

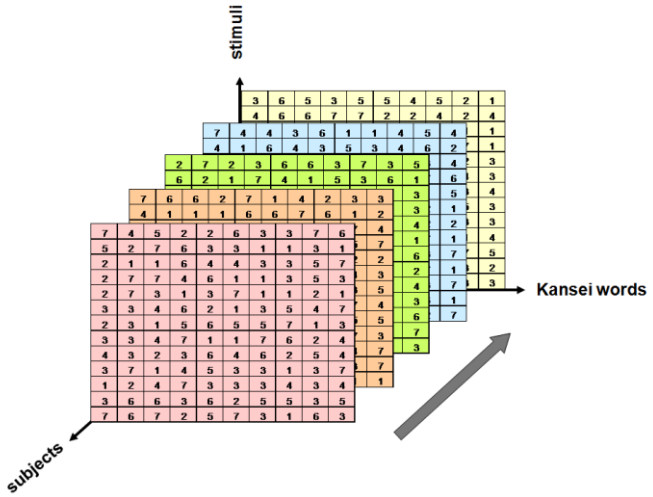
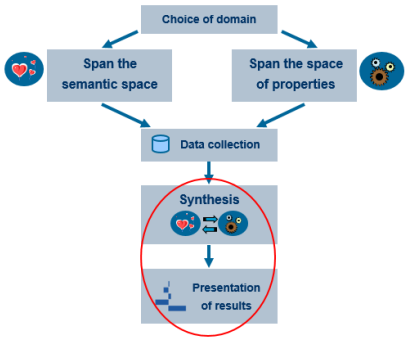


Figure 3. A three dimensional matrix with all the data from a kansei engineering Study

2.2.5. Synthesis and presentation of results



All steps viewed until now were in fact beyond the scope of this work, and were presented to give the full view of a kansei engineering study. The App will just perform the two final steps, the synthesis and the presentation of results. As it has already been stated in the scope of the project, the main target of this project is only the presentation of results.

Regarding the synthesis stage, there are lots of studies that try to determinate the best statistical methods to be used to obtain results from the raw data. Anyway, all methods have the same objective: knowing which factors affect each word and give a quantitative measure of how much. Furthermore, in almost all experiments all methods lead to similar results and conclusions.

Since it is not the goal of the project to find the best model, the ones proposed by Lluís Marco in his thesis [1] will be used (or minor modifications of those proposals), except in the creation of a multicriteria optimization, that was not included there and is going to be developed entirely. Section 3.3 explains how all the statistical models work out.

So, as it has already been repeated, the presentation of the results will be the focus of this thesis, being indeed everything that will be seen in the App. Chapter 3 (except section 3.3) is included later to explain how the results should be displayed and the arrangement in the App's layout. Chapters 4 and 5 also refer to the App, but from the programming perspective.

3. Design of the application and statistical methods

This chapter will go through the conception and design of the application. At first, the general philosophy will be stated, collecting what guidelines should follow the App. Afterwards a mock-up of the App will be proposed. Finally, a section about statistical tools used in the App is included.

3.1. General philosophy and functions of the App

This App must follow what have already been said in the objectives, mainly in the specific ones. There are three main issues that should be accomplished:

Interactive App: Users must perceive it and be able to exploit it. To achieve this, the App must have real time updates. Everything in the App must show always the results obtained with the current selected inputs, with no need of clicking on any refresh button. Other interactive tools should be used, to reinforce this idea in the users: tooltips that display extra information, possibility to select or deselect parts in the plots, and many inputs to play with.

Sequential workflow: The way to use the App is going to be sequential, step by step. That means having a specially designed layout that shows the user the order to follow, but that also allows him or her to go back to any of the previous steps, or even to jump forward if he wishes. Moreover, changes on each of the steps must affect all the other ones (for example, if the user decides to stratify by gender, everything must be changed according to that). The different steps must have memory, that is, they do not reinitialize every time the user goes through them.

Easy to interpret results: Conclusions must be clearly obtained from the displayed visualizations, which must have straightforward interpretations that cannot be misleading. All technical complexities are transparent to the user: though the results sometimes come from sophisticated statistical algorithms, the user does not need to know these details. The interface must be user-friendly, which means that everything is well organised and that styles used

are appealing, in other words, it looks nice.

After describing the general properties of the App, it is time to decide its functions and how these are going to be implemented and grouped. Below there is a list of everything that the App should perform (calculate or display):

- Acquire data
- Calculate and display design matrix and confusion indexes
- Give a summary of the data
- Show participants' information
- Calculate and display significant factors
- Calculate and display possible outliers
- Exclude desired participants
- Calculate and display the influence of the factors for each kansei word
- Enable to weight desired responses
- Calculate and display the best factor combinations
- Compare results with manual entries
- Provide stratification options
- Provide security level options

Any other minor functions can be included into the previous ones, and the ones that do not appear in the list will not be implemented. The next step is to group or split all these functions, decide where they are going to be visualized in the app, if something must appear twice…

3.2. Wireframe of the application

As it has already been said, the App will consist in steps allowing a sequential usage. In this section a mock-up of each one of the steps, which are going to be panels, will be explained. This means defining which functions are going to be implemented, how they are going to be arranged, and which are the best plots or other visualisations to show the results.

Apart from what is going to be shown in the wireframe, there should be a place in each panel for extra options. One is the possibility to stratify by gender, age or any other factor stored in the loaded data. The other extra option could be a ”security” indicator, which would change the results obtained by being more or less strict when detecting significant factors. This is the

same as changing the significance level in the statistical tests.

3.2.1. First step, Data

In this panel (Figure 4) there must be some input that enables uploading data, the one that will be used for the analysis. It must have a specific format. When this data is loaded, some initial basic information appears: a summary of participants, including its number for each stratification options, and also information that indicates how good the data provided is.

To do that, information about the confusion among factors is given, and it is proposed to do so in two ways: a matrix that relates them in pairs, and a global indicator. Following the kansei juices experiment, a bad global indicator would mean that the prototypes showed were too similar to each other, so it could not be determined which the real factors that affect the response are. If a cell in the matrix showed a bad index, it would mean that those factors would be very confused, revealing for example that the effect produced by having a straw could not be distinguished from the one produced by having ice. The calculation of the confusion values is explained in section 3.3.1.

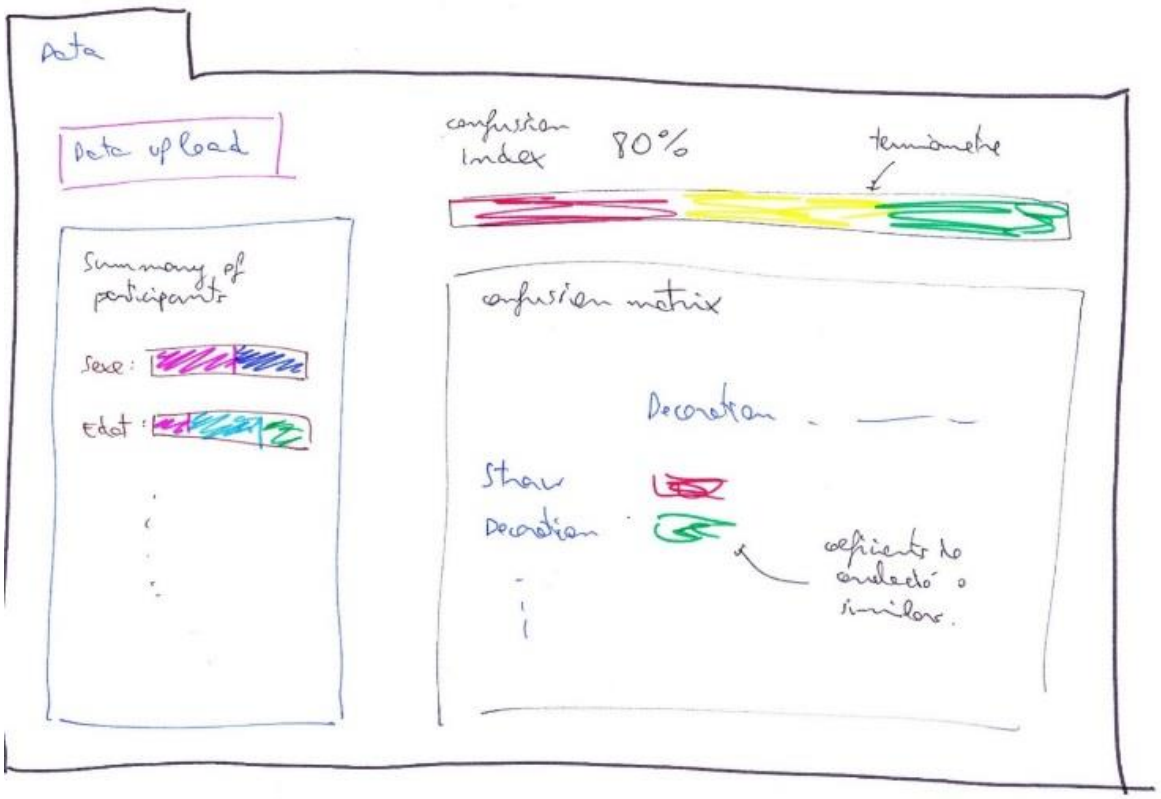


Figure 4. Wireframe of the first step, data acquisition

3.2.2. Second step, participants

This is a panel (Figure 5) conceived to allow the user to obtain the raw data in order to examine any specific participant. In the proposed design, this can be easily done by just clicking the corresponding dot in the scatterplot, which will result in a summary of the participant plus a table with all his or her scores.

But this scatterplot with the participants provides further information. The X axis, emotionality, shows the average of the punctuations given, which is a measure of how "emotional" the participants are (high emotionality indicates that the participant gave mainly high scores). On the other hand, the Y axis, called diversity, represents the variety in the scores: for example, a low diversity would mean that the participant gave the same scores to almost all prototypes. Finally, some dots will have a different shape, those are participants detected as outliers, which means that they scored very rare (in a very different way from the rest of participants) and for this reason including them in the study could give misleading results. The algorithm for automatically detecting outliers is detailed in section 3.3.2.

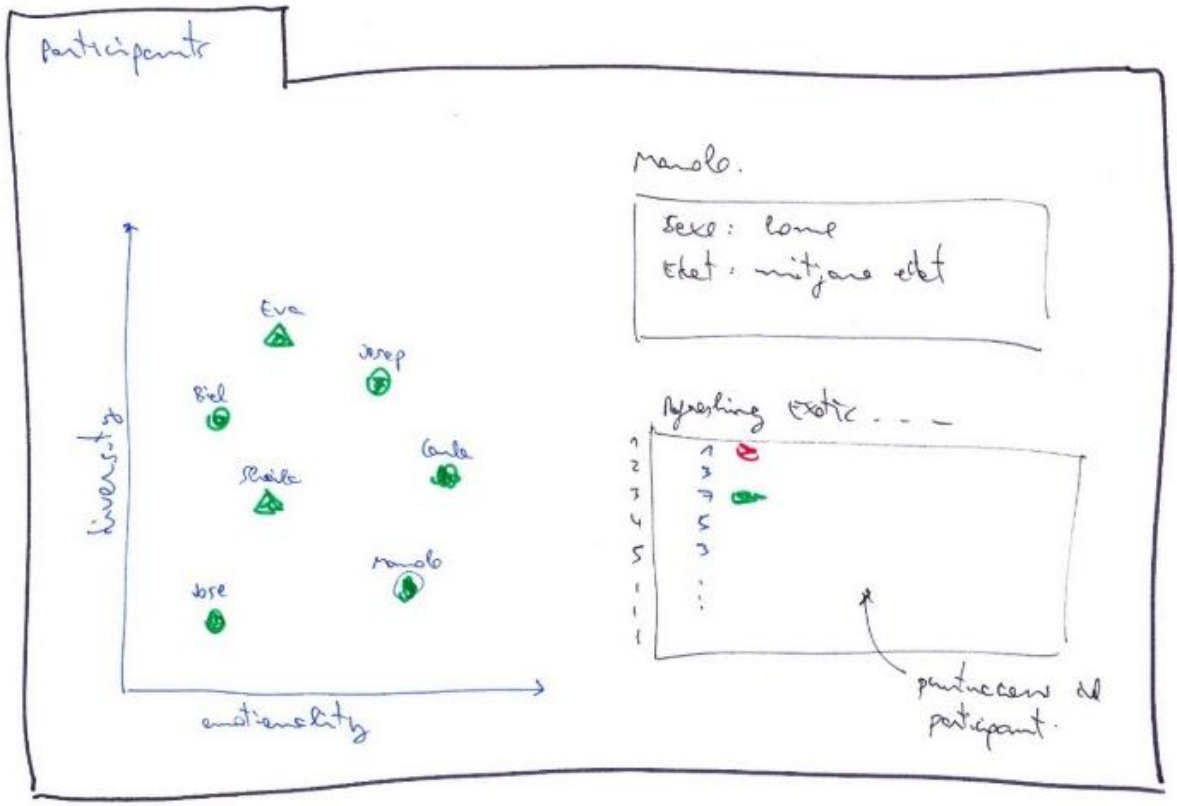


Figure 5. Wireframe of the second step, participants

3.2.3. Third step, Summary

In this panel (Figure 6) there will be two plots. The one on the left will be the same as the previous scatterplot, and the one on the right will be a summary of the results. This last one will be some kind of matrix (heatmap) with axis that are going to represent the kansei words and the factors. If a cell is coloured, that will mean that the corresponding factor affects the kansei word, the response. There should be some colour intensity that indicates the importance of each factor in each kansei word, giving an idea of which factors affect the most and the less for each word. The calculation of the results for the plot on the right is explained in section 3.3.3.

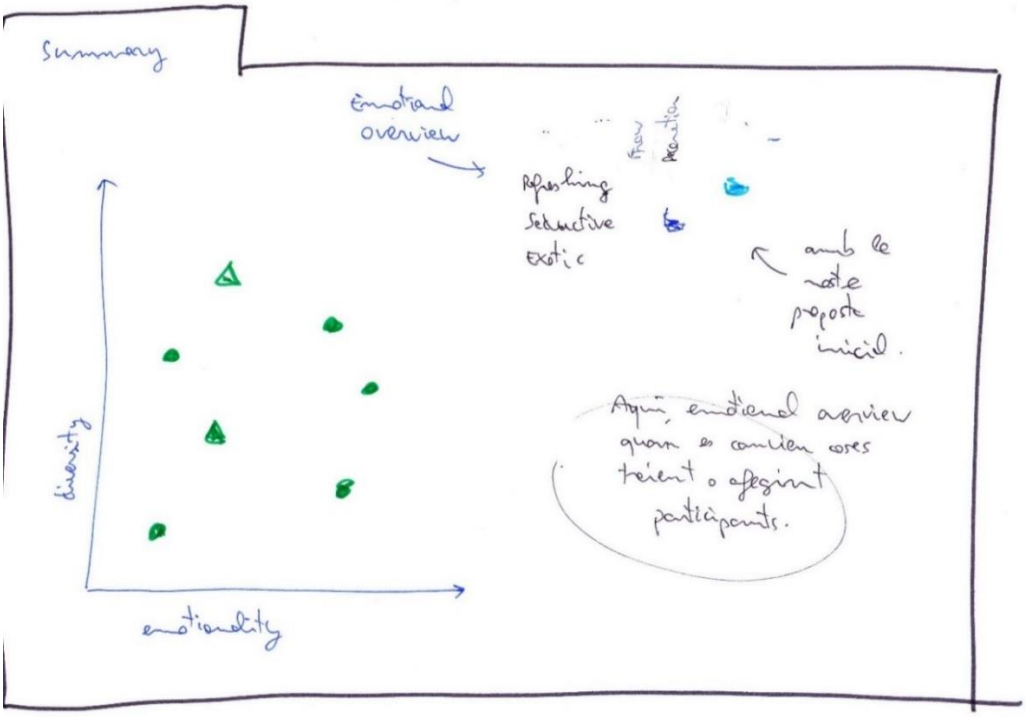


Figure 6. Wireframe of the third step, a summary of the results

3.2.4. Fourth step, learning

The aim of this panel (Figure 7) is to provide a more exhaustive analysis of the results already seen in the previous step. There will be a tab panel, where each tab will correspond to a kansei word. In each of these tabs, there will be two plots. One will show the variation produced in the response by changing the levels for each factor. Obviously only the factors that in the previous panel where stated as significant will have a repercussion here. The second plot will display the relative importance of each factor for each kansei word. This information has

already been given in the previous step with the colour intensity in the heatmap, but here it can be presented with more precision, and in a graphical manner.

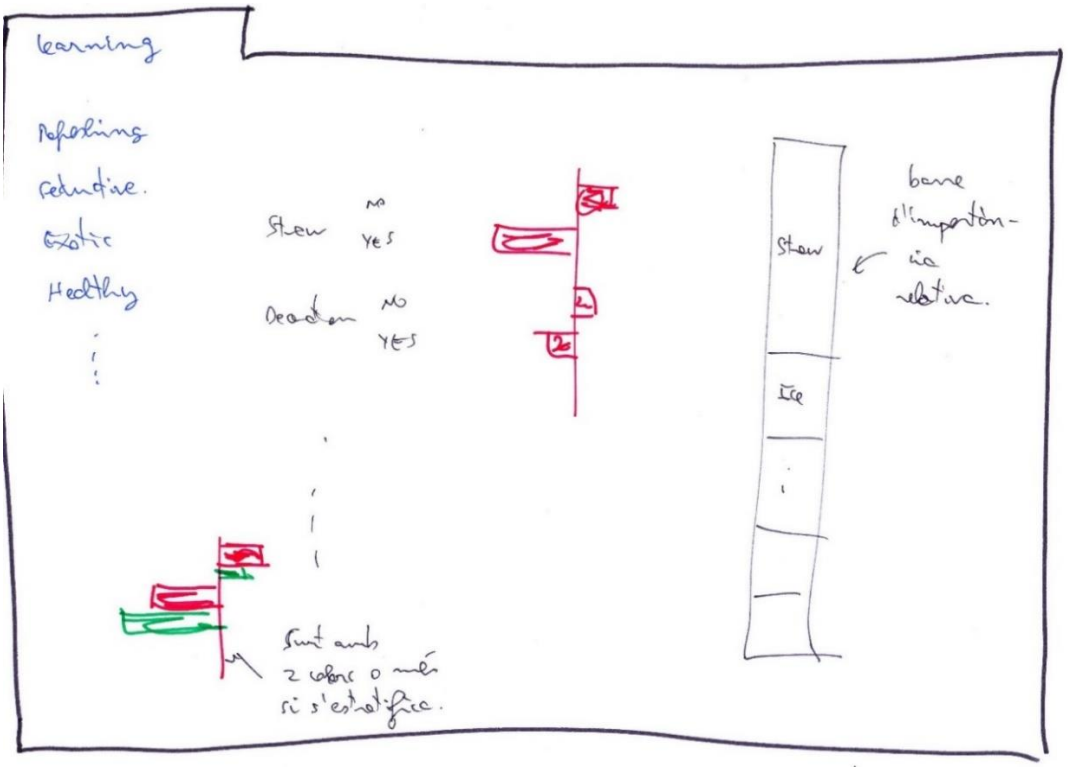


Figure 7. Wireframe of the fourth step, an analysis

3.2.5. Fifth step, optimizing

This is going to be the final panel (Figure 8), and also the one that summarizes everything to give a final and clear result that can be useful for the user to design a new product. The idea is to have one input for each kansei word which will be changed to give each of them the desired importance in comparison with the other words. Then, the optimization will be calculated (details are offered in section 3.3.4) and the best factor combinations will be displayed.

The 3 best options will be shown in a radar plot, which will enable an easy comparison to see the strengths and weak points of each proposed solution. It is important to remark that the suggested prototypes may be a combination of factor levels that have never been created or shown to the participants. This panel could have a further option (or maybe it could be implemented with a new panel) that would ask the user to manually enter a factor level combination, and show that new prototype in the radar plot next to the best ones.

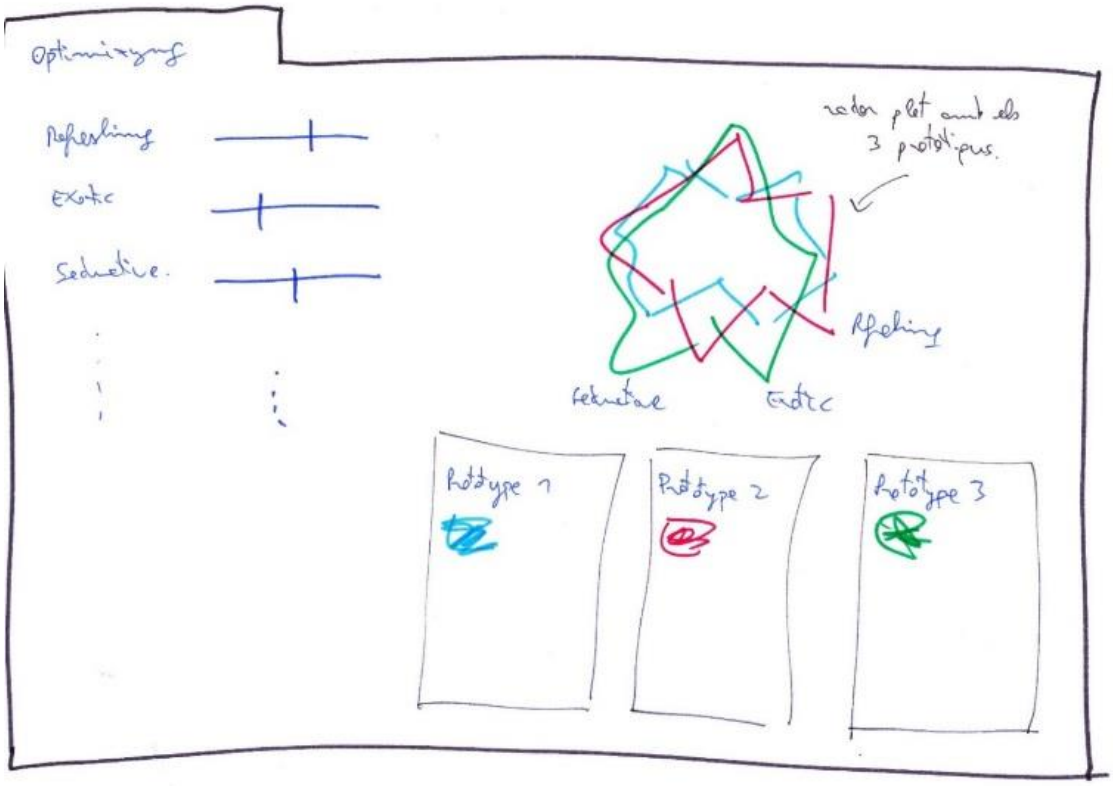


Figure 8. Wireframe of the fifth step, optimization

3.3. Statistical methods used in the application

3.3.1. Assessment of confusion in the design matrix

Here the statistical method to find the confusion indexes required for 3.2.1 is going to be explained. First, an index that will relate factors one by one will be obtained, and afterwards a global indicator.

The first index will go from 0 (two columns in the design matrix have the same sequence of levels, the main effects of those two factors are totally confounded, this is the worst situation), to 1 (levels are all different in the design matrix, they are completely independent, the best situation). Values in between will give an idea of how far the design is from that extreme situations.

It will be computed using Cramer's V [2], a statistic that measures the strength of association between two nominal variables in a contingency table. It is named after the Swedish mathematician Harald Cramér.

Suppose X and Y are two factors. X has M different levels, labeled $X_1, \dots, X_M$. Y has N different levels, labeled $Y_1, \dots, Y_N$. X and Y can be arranged as a contingency table, as shown in Figure 9.

$X \backslash Y$	Y_1	Y_2	$\cdots$	Y_N
X_1	n_{11}	n_{12}	$\cdots$	n_{1N}
X_2	n_{21}	n_{22}	$\cdots$	n_{2N}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
X_M	n_{M1}	n_{M2}	$\cdots$	n_{MN}

Figure 9. A contingency table with factors X and Y

In this contingency table, cell (i, j) contains the count n_{ij} of occurrences of level X_i in X and level Y_j in Y. n is the total number of pairs that can be done, and $n = \sum n_{ij}$.

The chi-squared statistic χ^2 can be computed from this contingency table. Then, Cramer's V is defined as:

$$V = \sqrt{\frac{\chi^2}{n \, min(M-1, N-1)}}$$

In a design matrix with r factors, there are $k = \binom{r}{2}$ pairs of factors. For each of these pairs of factors pf_i, with $i = 1, \dots, k$, its correlation V_{pf_i} (by means of its Cramer's V statistic) can be computed. In fact, what is going to be shown in the matrix that relate factors by pairs is the complimentary of V_{pf_i}, which will be named M_{pf_i}:

$$M_{pf_i} = (1 - V_{pf_i})$$

Finally, a global index is computed, to have a general indicator that take into account the index of all factors' pairs, the global independence coefficient (GIC). This GIC is the complementary of the geometric mean of the Cramer's V correlation coefficient of all pairs of factors.

$$GIC = \left(1 - \sqrt[k]{\prod_{i=1}^{k} V_{pf_i}}\right) \cdot 100$$

The geometric is suitable in this case, because if only one pair of factors are completely correlated, the GIC will give a value of 0% (meaning the structure of the design matrix is completely unsuitable).

3.3.2. Automatic detection of outliers

Simply looking at the data, it is sometimes possible to detect participants in a KE study who give ratings in a weird way, but the multidimensional nature of KE data (several participants rate different stimuli on a number of kansei words) makes the manual detection of outliers an almost impossible task.

Although studies with large number of participants are quite robust to the presence of outliers, many KE studies are done with few participants. Therefore an automatic detection of outliers is necessary to avoid them, since they often change the results in the synthesis phase. Excluding outliers can give protection from reaching wrong conclusions.

The method to detect outliers will be based on what is explained in H. R. Álvarez doctoral thesis [3], which is about detecting outliers in KE studies. This work presents a methodology based on robust principal component analysis and distinguish to phases.

- Detecting outliers in the so-called simple kansei tables (only considering one response, one kansei word).
- Detecting outliers in the so-called multiple kansei tables (considering all responses, all kansei words, at the same time).

Detecting outliers in these simple kansei tables can be done using the ROBPCA algorithm proposed by Hubert and Rousseeuw [4]. ROBPCA stands for Robust Principal Component Analysis. Although this method require working with continuous variables and data from KE studies is usually ordinal, the procedure can be safely used when using the common 7-point scale response. ROBPCA purpose is twofold: it allows the calculation of principal components resistant to outliers and it gives a diagnostic plot that allows the detection of outliers.

The details of the ROBPCA algorithm are complicated, but the procedure can be summarized in three steps. Consider n subjects rate m stimuli on a kansei word. A simple kansei table $\boldsymbol{X} = \{x_{ij}\}$, with $i = 1, \dots, n$ and $j = 1, \dots, m$ can be created (the rows are subjects and the columns are stimuli). x_{ij} is the rating given by participant i to stimuli j:

1. The data is preprocessed by reducing their space to the subspace spanned by the n observations. This is done by singular value decomposition of the original matrix $\boldsymbol{X}$.

The transformed data are then lying in a subspace with a dimension that is, at most, $n-1$.

2. A measure of outlyingness is calculated for each data point: the data points are projected on many univariate directions, each time the univariate estimator of location and scale is computed and the standardized distance to the center is measured. The largest of these distances is the outlyingness measure of that data point. The data points with smallest outlyingness measures are used to compute a preliminary covariance matrix $\boldsymbol{S}_0$. This covariance matrix $\boldsymbol{S}_0$ is used for selecting the number of components p that will be retained.
3. The data points are finally projected on the p-dimensional subspace where their location and scatter matrix are robustly estimated.

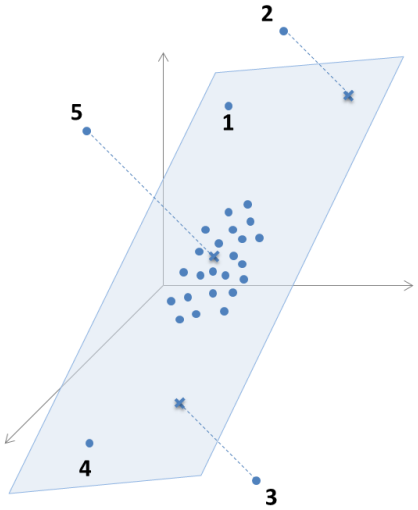


Figure 10. 3-dimensional dataset projected on a robust 2-dimensional PCA subspace

An extremely useful output from a ROBPCA is the diagnostic plot. The diagnostic plot allows the detection of outliers and the determination of its type. Consider the graph in Figure 10, where $m = 3$ and $p = 2$. Four types of observations can be described:

- Regular observations: they form a homogenous group close to the PCA subspace.
- Good leverage points: they fall close to the PCA subspace, but far from the regular observations (points 1 and 4 in Figure 10).
- Orthogonal outliers: they have a large orthogonal distance to the PCA subspace, but cannot be detected as outliers if we just look at their projection on the PCA subspace (point 5 in Figure 10).
- Bad leverage points: they have a large orthogonal distance and its projection on the PCA subspace is far from the typical projections (points 2 and 3 in Figure 10).

The diagnostic plot (Figure 11) places each observation in a scatterplot where the horizontal

axis shows a robust score distance and the vertical axis shows an orthogonal distance. To classify the observations, two lines are drawn that divide the plot in four quadrants.

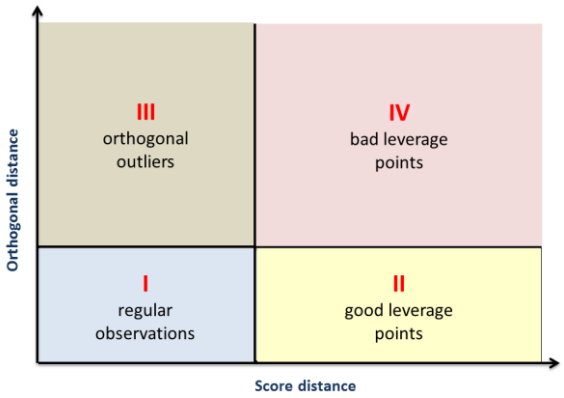


Figure 11. The diagnostic plot for detecting outliers according to the ROBPCA method

The diagnostic plot can then be used to detect participants in a kansei engineering study that give ratings on a kansei word in a manner very different from the others.

Although it could be possible to detect outlier participants for each kansei word, it would be interesting having a procedure to detect which participants can be flagged as outliers considering all kansei words at the same time. This is done using a multiple kansei table. The multiple kansei table is created juxtaposing m tables (one for each stimuli in the study). Each table has n rows (participants) and r columns (kansei words). Its structure is shown in Figure 12.

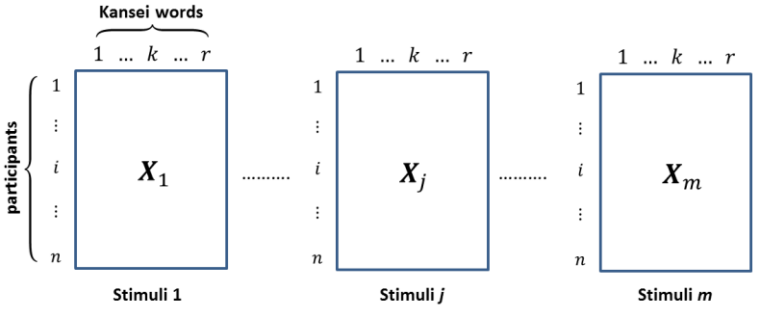


Figure 12. Structure of a multiple kansei table.

The method for detecting global outliers combines a multiple factor analysis like the one explained by Escofier and Pagès [5], and the ROBPCA method. Data in the multiple kansei table is centered (columns' means are subtracted to each element) before starting. These are the two steps of the procedure:

1. The multiple kansei table is treated as in a multiple factor analysis. A subtable could have a contribution in the first factorial axis much higher than the others. To avoid this, they should be weighted. This is done through a principal component analysis (with all observations, including outliers) in each subtable. They are then "normalized" by dividing all its elements by the square root of the first eigenvalue from its PCA:

$$\left({}^{1}\!\Big/\!{\sqrt{\lambda_1^1}}\, X_1 \;\middle|\; \dots \;\middle|\; {}^{1}\!\Big/\!{\sqrt{\lambda_1^k}}\, X_k \;\middle|\; \dots \;\middle|\; {}^{1}\!\Big/\!{\sqrt{\lambda_1^r}}\, X_r \right)$$

2. A ROBPCA is applied to the weighted juxtaposed table. This new table is considered as a simple table with n rows and $m + m + \cdots + m = m \cdot r$ columns. The diagnostic plot from ROBPCA allows then the detection of global outliers.

Following all that has been explained in this section, the outliers can be detected. However, it is not always recommendable to exclude them directly. It should be seen how much they change the results. Maybe it could be interesting to know more about those participants to take a better decision. This is why this App will detect outliers, but will give the opportunity to include them back to the study at the user needs.

3.3.3. Mixed effects regression analysis for the synthesis phase

In statistics, regression analysis can be used to model the relationship between a dependent variable and one or more independent variables. In a kansei engineering study, each kansei word acts as a response, whereas each factor is an independent variable. All the well-known techniques from regression analysis can be used for facing the synthesis phase of the study.

So the aim in the synthesis phase is estimating the main effects of all factors for each one of the kansei words. The particularity in kansei studies is that the independent variables are categorical factors (having two or more levels), and not quantitative. So all factors must be translated into dummy variables to perform the regression [6].

Dummy variables are built in the following way:

$$\delta_{i(jk)} = \begin{cases} 1, & \text{if product i has category k in item j} \\ 0, & \text{otherwise} \end{cases}$$

i = 1, ..., n (with *n* the number of prototypes)

j = 1, ..., R (with *R* the number of factors)

k = 1, ..., C_j (with C_j the number of levels in factor j).

As an example, we can imagine we have a kansei study with only one kansei word, the word colourful. The prototypes are T-shirts, and the design matrix is the one shown in Table 6.

Table 6. Design matrix and responses for the T-shirts example

	Color	Sleeves	Printing	Carla	Joan	Marc	Maria	MEAN	
1	Red	Long	Picture	5	6	7	5	5.75	1
2	White	Long	Picture	3	4	5	3	3.75	2
3	Red	Short	Picture	7	4	5	6	5.50	3
4	White	Short	Picture	4	4	4	3	3.75	4
5	Red	Long	Plain	1	4	5	1	2.75	5
6	White	Long	Plain	1	3	2	1	1.75	6
7	Red	Short	Plain	4	5	5	2	4.00	7
8	White	Short	Plain	2	4	5	1	3.00	8
9	Red	Long	Text	5	6	6	4	5.25	9
10	White	Long	Text	1	2	5	1	2.25	10
11	Red	Short	Text	5	6	7	4	5.50	11
12	White	Short	Text	2	4	3	2	2.75	12

For this example, the response will be the mean of the ratings given by all 4 participants in the study. The following nomenclature will be used to represent each factor level:

x_1: Color: x_{11} = white; x_{12} = red.

x_2: Sleeves: x_{21} = long-sleeved; x_{22} = short-sleeved.

x_3: Printing: x_{31} = picture; x_{32} = plain; x_{33} = text.

The design matrix with dummy variables can be seen in Table 7.

Table 7. Design matrix for the T-shirts example with dummy variables

	Color	Sleeves	Printing	x11 Red	x12 White	x21 Long	x22 Short	x31 Picture	x32 Plain	x33 Text	MEAN
1	Red	Long	Picture	1	0	1	0	1	0	0	5.75
2	White	Long	Picture	0	1	1	0	1	0	0	3.75
3	Red	Short	Picture	1	0	0	1	1	0	0	5.50
4	White	Short	Picture	0	1	0	1	1	0	0	3.75
5	Red	Long	Plain	1	0	1	0	0	1	0	2.75
6	White	Long	Plain	0	1	1	0	0	1	0	1.75
7	Red	Short	Plain	1	0	0	1	0	1	0	4.00
8	White	Short	Plain	0	1	0	1	0	1	0	3.00
9	Red	Long	Text	1	0	1	0	0	0	1	5.25
10	White	Long	Text	0	1	1	0	0	0	1	2.25
11	Red	Short	Text	1	0	0	1	0	0	1	5.50
12	White	Short	Text	0	1	0	1	0	0	1	2.75

If a linear regression analysis if performed with this dataset, the equation will not have all levels for each factor. One level for each factor will be missing, as this level acts as a reference level. For instance, the regression equation coming from the T-shirts example is the following:

$$\hat{Y} = 3.2292 + 1.9167\, x_{11} - 0.5000\, x_{21} + 0.7500\, x_{31} - 1.0625\, x_{32}$$

This equation is obviously the same as this one, which makes the reference levels for each factor explicit:

$$\hat{Y} = 3.2292 + 1.9167\, x_{11} + 0\, x_{12} - 0.5000\, x_{21} + 0\, x_{22} + 0.7500\, x_{31} - 1.0625\, x_{32} + 0\, x_{33}$$

Although having reference levels is something common in regression, it makes interpretation of results somewhat more complex to people not accustomed to it.

Quantification theory type I (QT1) is a variation of linear regression analysis first proposed by Chikio Hayashi [7], with the aim of facilitating the interpretation of results from a regression analysis with categorical variables.

The idea behind QT1 is having a regression equation that has the mean of the response as the constant, and coefficients for all the levels of the factors in the equation (no reference level). This coefficients are called, in the context of QT1, category scores (CS). So, in the T-shirts example, the final equation should have this form:

$$\hat{Y} = b'_0 + b'_{11}x_{11} + b'_{12}x_{12} + b_{21}{}' x_{21} + b_{22}{}' x_{22} + b_{31}{}'x_{31} + b_{32}{}' x_{32} + b_{33}{}' x_{33}$$

How are these CS computed? One should work factor by factor. Consider factor *j*. The coefficients in the transformed regression are calculated from the following equation:

$$b'_{jk} = b_{jk} + Q_j$$

Q_j is calculated from the formula:

$$\sum_{k=1}^{c_j} P_k\left(b_{jk} + Q_j\right) = 0$$

Each factor *j* will have a different Q_j constant. P_k is the proportion of appearance of level *k* from factor *j* in the sample.

An example focusing on the item Printing (*j*=3) from the T-shirts example will illustrate the procedure. The item Printing has 3 categories (Picture, Plain and Text). The coefficients in the transformed regression are calculated from the following equation:

$$b'_{3k} = b_{3k} + Q_3$$

Where Q_3 is a constant computed from:

$$0.333(0.7500 + Q_3) + 0.333(-1.0625 + Q_3) + 0.333(0 + Q_3) = 0 \Rightarrow Q_3 = 0.1042$$

Table 8 summarizes all calculations for the factor Printing:

Table 8. Calculation of category scores for factor Printing in the T-shirts example

Category	Coefficients in original regression	Proportion in design matrix	Coefficients in transformed regression
Picture (x_{31})	b_{31} = 0.7500	4/12 = 0.333	b_{31}' = 0.7500 + 0.1042 = 0.8542
Plain (x_{32})	b_{32} = -1.0625	4/12 = 0.333	b_{32}' = –1.0625 + 0.1042 = –0.9583
Text (x_{33})	b_{33} = 0	4/12 = 0.333	b_{33}' = 0 + 0.1042 = 0.1042

The final transformed equation for the T-shirts example is:

$$\hat{Y} = 3.8333 + 0.9583\, x_{11} - 0.9583\, x_{12} - 0.2500\, x_{21} + 0.2500\, x_{22} + 0.8542\, x_{31} - 0.9583\, x_{32} + 0.1042\, x_{33}$$

The great advantage of QT1 is that it allows a graphical representation of the results very easy to interpret (Figure 13).

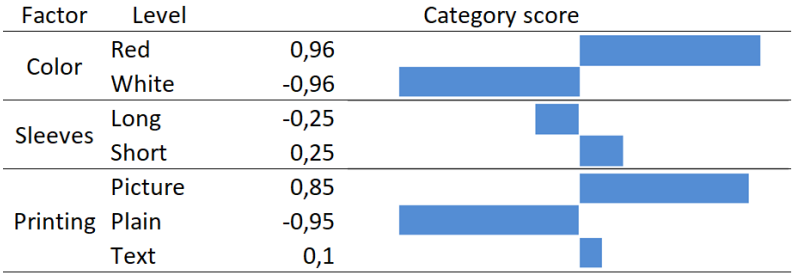


Figure 13. Representation of the category scores in the T-shirts example

The App uses an improved version of the QT1 algorithm that takes into account the following issues:

1. The response is not summarized with the mean of all participants. On the contrary, each participant raw rating is used (although ratings from 1 to 7, the most common

scale used in kansei engineering studies, are obviously not continuous, they can be directly used as the response with no further consequences). However, the participant is introduced in the regression model as a random effect. The reason for doing this is that each participant represents a cluster in the data, and analyzing all the ratings without introducing the participant as a random effect will clearly violate the assumptions of the linear regression. Furthermore, the variability among participants is correctly extracted when considering participants as a random effect.

2. A global p-value is computed for each one of the factors in the regression analysis, based on the common likelihood ratio used in statistics. In this way, we can determine if a factor is significant or not with respect to a certain significance level (the default in the App is 0,05, called medium safety level). When a factor is not significant, its category scores are set to 0, in order to give the correct impression that the factor has no effect in that kansei word.

The tab analysis of the App exploits the results from this QT1 analysis for each kansei word in detail.

3.3.4. Multicriteria optimization to detect the best prototypes

In this section the model for optimizing the response will be explained. A multicriteria optimization method will be required, where each of the responses (each word of each stratification level) will have a different weight but also a direction (minimize or maximize).

Before going deeper into this topic, it is important to say that the results of a kansei experiment are not as exact as the ones obtained in an experiment in the industry, where responses are measured and not scored subjectively by people. This is the reason why the exact implementation of this optimization model is not as important as in other cases, and only a gross indicator that takes into account the provided weights will be enough. Following this premise, the clearer and simpler methods will be employed, to provide a better understanding of the code and lower computation times.

After some research, the desirability functions are selected, which were first introduced by Harrington [8] and have the functional forms described by Derringer and Suich [9] shown in Figure 14. It is a very widespread method in design of experiments, robust and easy to use.

$$d_i(\hat{Y}_i) = \begin{cases} 0 & \text{if } \hat{Y}_i(x) < L_i \\ \left(\frac{\hat{Y}_i(x) - L_i}{T_i - L_i}\right)^s & \text{if } L_i \leq \hat{Y}_i(x) \leq T_i \\ 1.0 & \text{if } \hat{Y}_i(x) > T_i \end{cases} \qquad d_i(\hat{Y}_i) = \begin{cases} 1.0 & \text{if } \hat{Y}_i(x) < T_i \\ \left(\frac{\hat{Y}_i(x) - U_i}{T_i - U_i}\right)^s & \text{if } T_i \leq \hat{Y}_i(x) \leq U_i \\ 0 & \text{if } \hat{Y}_i(x) > U_i \end{cases}$$

Figure 14. Desirability functions. Left to maximize and right to minimize

Using the formulas in Figure 14 for this application, each of the kansei words response will become a desirability function *d*, that will give an idea of how far is that response from the best one, the target. The one on the left is to be used when maximizing, and the one on the right for minimizing. Variables are obtained as following:

- $\hat{Y}_i$: It is a response for each kansei word and for each prototype. It is calculated as the "mean" for that word plus the scores of its factors levels.
- L_i : Is the lower limit, the minimum possible punctuation, usually 1.
- U_i : Is the upper limit, the minimum possible punctuation, usually 7.
- T_i : Is the target value. When maximizing it is going to be equal to U_i and to L_i when minimizing.
- *s* : It is an exponent that can be set to different values, producing the effect seen in Figure 15. This variable could be changed to give more importance to values that are closer to the target, for example. Long discussion could be hold to determinate its value, but as it has already been said, this project only aims to get a simple approach. For this reason the linear desirability function will be employed, with *s*=1.

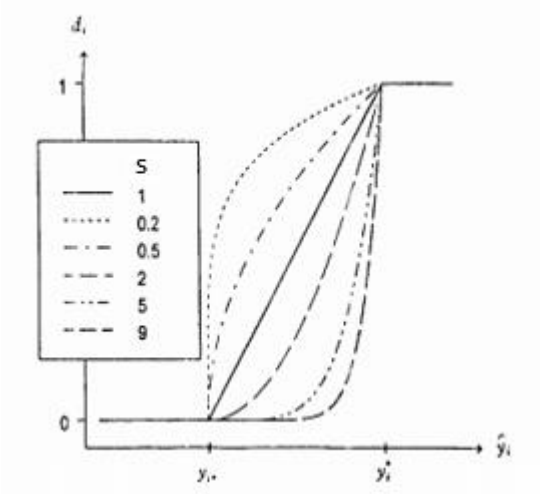


Figure 15. Change of the shape of d function depending on s coefficient

Although the desirability functions in Figure 14 are generally defined by parts, in this particular use only the central definition will be employed, because it is always true that:

$$L_i < \hat{Y}_i < U_i \text{ and } L_i < T_i < U_i$$

The app will also allow to weight the different kansei words, so each desirability function must also contain that information. The weights used for each kansei word are captured with the W_i values. Afterwards some "mean" must be performed with all desirability functions for each prototype. All this is done with the formula in Figure 16, obtaining one *D* function for each prototype. Finally, all the *D* functions are compared, and the best ones will be the best and optimized prototypes. The value of *D* itself, can be seen as a percentage of how good that prototype is in front of the best possible one.

$$D = (d_1^{W_1}(Y_1) d_2^{W_2}(Y_2) \cdots d_k^{W_k}(Y_k))^{1/\sum W_i}$$

Figure 16. Formula for the D function

One last important thing to consider is what happens if the user stratifies for example by gender. Then, a *d* function will be obtained for each prototype and kansei word, but now also for each level. Again all those functions will be weighted and grouped by prototype in the formula already seen in Figure 16. For example, when only a *d* function for the word refreshing of the prototype number one was to be computed without a stratification, after stratifying by gender there will be two *d* functions instead, one for men and one for woman.

4. Programming tools

This chapter is a brief explanation of the R Programming Language characteristics and its extensions used for this project.

4.1. R Programming language

R [10] is an open source programming language and software environment for statistical computing and graphics. It is an interpreted language and also highly object-oriented, this last as a consequence of being an evolution of the S language.

The R language (Figure 17) is widely used in the data science field for developing statistical software and data analysis, because it can implement a wide variety of statistical and graphical techniques, including linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, and many others. One of the main differences it has with other languages is its capability of showing a wide range of user configurable graphics in a very professional way.

Figure 17. R logo

R is easily extensible through functions and extensions, which are called packages. The R community is noted to be very active, and now there are packages for lots of specific uses. Packages can be easily obtained downloading them through an online repository named CRAN. Many of R's standard functions and packages are written in R itself, which makes it easy for users to follow the algorithmic choices made, but other languages as C, C++, Fortran, Java or Python can be used for a more advance control (and sometimes faster executions).

Another big difference that R holds is the implementation of data frames, an object specially designed for statistical analysis. Data frames are used for storing data tables, being a list of vectors of equal length. The top line of the table, called the header, contains the column names and each horizontal line afterward denotes a data row, which begins with the name of the row, and then followed by the actual data. Each data member of a row is called a cell. This concept, similar to a matrix, enables an easy access to the data that allows a flexible subsetting in multiple ways by indexes or names.

Finally, there are different integrated development environments (IDE) for R that provide the users an interface to program more efficiently. The most commonly used, Rstudio [11], includes a console, syntax-highlighting editor that supports direct code execution, integrated R help and documentation as well as tools for plotting, history, debugging and workspace management. It really makes programming in R much easier and it has been a key for its success.

4.2. Shiny

Shiny [12] is a package that creates an interactive App from R. To achieve that, Shiny generates the necessary *HTML*, *CSS* and JavaScript code (Figure 18) producing in this way a whole website. Thus, the Shiny user does not need to know anything about web programming, although it can help him or her to implement advanced features.

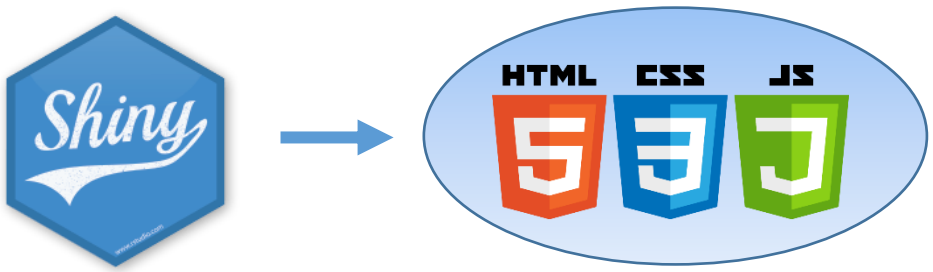


Figure 18. Shiny package from R will generate all necessary code to create a website

Shiny applications have always both inputs, which users can set and change, and outputs, that change interactively according to the inputs' state.

As inputs, Shiny already provides a wide range of widgets including sliders, radio buttons, check boxes, text, file uploaders, action buttons, dates and select boxes. It also provides different layouts, so tabs and navigation menus can be implemented.

The outputs can be anything R alone could output, like tables, plots or plain text. However, more and more packages are appearing which extend the range of possible outputs. Some of them, called wrappers, make JavaScript implementations work in R, achieving that now the user can also interact with the outputs (for example, selecting a point in a plot), converting outputs in inputs as well.

An example of a very simple Shiny App is shown in Figure 19 . There is just one input, a slider that controls the number of observations. There is also just one output, a plot of a histogram that automatically changes to match always the number of observations set in the input.

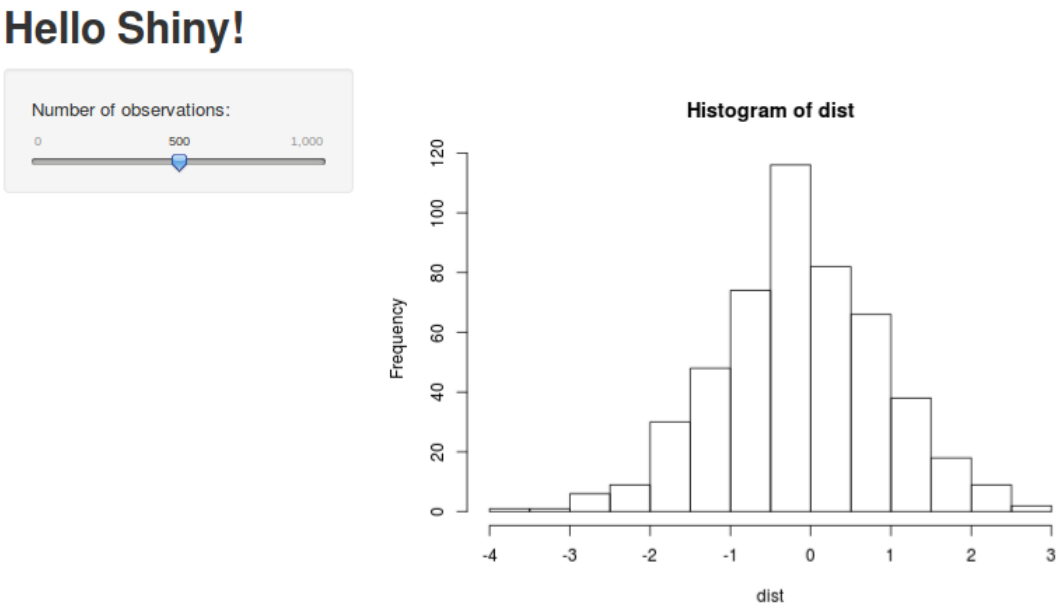


Figure 19. Example of a Shiny App

Furthermore, if the programmer has knowledge of *HTML*, *CSS* and *JavaScript*, he will also be able to customize all these inputs and outputs, or even to create new ones. When this happens, the new output or input is normally wrapped in a package that anyone will be able to use, and this way Shiny is growing more and more.

Shiny applications have two main components: a user-interface definition and a server script. The first one is a function named *ui* that contains the layout of the page, saying which elements (menus, panels, inputs and outputs) the App will contain and where they are going to be placed. The second one, a function named *server*, contains the code with the logic that computes what need to be outputted according to the state of the inputs. Those two functions can be in one file called App, or in two .r files.

4.2.1. Reactivity

As said by Joe Cheng [13], this is the heart feature of Shiny, the one that allows its interactivity. As a rule, code in R executes just once (except from loops), but this way detecting when an input is changed would be impossible. A new conception is needed, which will make some parts of the code to recalculate when inputs are changed: that is exactly what reactivity offers. That implies that any part of the server code which needs to rerun due to interaction, will have to be within a reactive object.

What is interesting about reactive objects is that whenever they execute, they automatically keep track of what reactive objects they read. If those "dependencies" become out of date, then they know that their own return value has also become out of date. Then the reactive object will recalculate and update its return value, and this will automatically instruct all reactive objects that depended on that value to re-execute.

The simplest structure (Figure 20) involves just a source (typically a user input, for example a slider) and an endpoint (something that appears, such as a plot or a table of values):

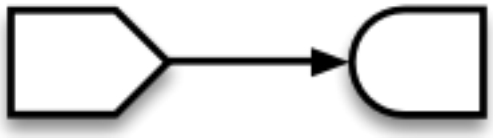


Figure 20. Representation of a reactive source and endpoint

When the source changes, the output invalidates. Once the source is again ready, the endpoint detects it and reruns to produce an updated output.

Just with reactive sources and reactive endpoints, only the simplest examples could be implemented. For more complex Apps, it is necessary to put reactive components in between the sources and endpoints. These components are called reactive conductors.

A conductor can both be a dependent and have dependents, it can be both a parent and a child. Sources can only be parents (they can have dependents), and endpoints can only be children (they can be dependents) in the reactive graph (Figure 21). Reactive conductors can be useful to encapsulate computationally expensive operations that are used more than once. As many conductors as necessary can be created to bind one endpoint with the sources.

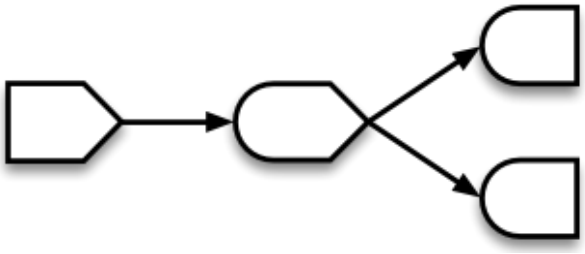


Figure 21. Reactive representation with a reactive conductor

Shiny has one class of objects for reactive sources (reactive values), one for reactive conductors (reactive expressions, which always return a value), and one for reactive endpoints (observers, which can access reactive sources and reactive expressions and are used for their side effects, they do not return any values). These class objects are represented in Figure 22.

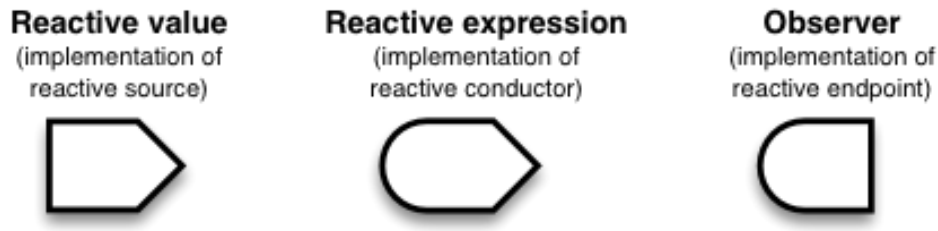


Figure 22. Summary of the 3 reactive elements

4.2.2. Bootstrap and CSS

To configure the layout and distribution of all the App panels, Shiny uses Bootstrap grid. It consists of rows which in turn include columns. Rows make sure their elements appear on the same line, and columns define how much horizontal space they occupy within a 12-unit wide scale. It is a responsive design, so their components will always fit in real time the available browser width. They can also be nested, creating other fluid rows (which can be divided into 12 columns as well) in one of the previous columns, as shown in Figure 23.

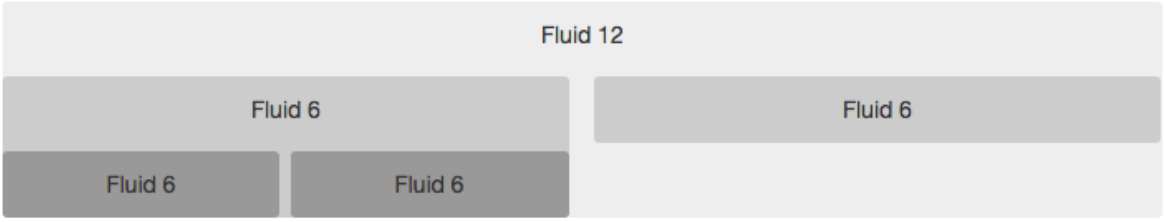


Figure 23. Nested bootstrap fluid rows

Regarding the styles (font, sizes, colours…) they can be configured from zero, or an already existing theme from Bootstrap can be used. For this application, *Yeti* theme has been used, which contains all the information in a .css file. Afterwards, a personal customisation has been done by modifying some global parameters (like the background colour of the panels) in this file and some specific ones (like the size of just one header) in the R App code.

4.3. Modules

Modularizing a Shiny App is a new and powerful feature that allows the programmer to divide an App in different parts. It can be said that an App can have other Aps inside which are independent one from each other. Following Garrett Grolemund [14] definition, a module is a self-contained and composable component of a Shiny App. Modules can also be nested, reaching as many levels as necessary.

A great benefit of a modularized App is that variables inside one module do not affect the other ones, so there will not be any naming conflict. Variables can be passed from one module to another, but it must be explicit.

Another advantage is that one module can be called more than once, so if some part of the App is repeated, there is no need to write the same code more than once, just to call the same module the desired times.

Finally, modules help to have a better organized code, because the App is going to be clearly divided also in the code. Modules can even be saved in separate files (one UI and one server file for each module), which allow an even tidier App.

Our App is going to be a modularized App, each module containing one of the 6 panels. That means that there will be one *server.R* and *ui.R* file pair, which are very empty and basically just call all the modules. Modules are stored in the *server* and *UI* folders pair. Anyway, the *UI* file will be quite empty in comparison with its analogue. The reason is that it will only output another *UI* that will be created in the server through a *renderUI* command. It is the only way to do it, because the *UI* depends on the data loaded, and it must also be wrapped in a reactive environment.

There is also a *global.R* file which imports all the necessary packages, tells the App where to find the modules and contains some constant variables that can be always accessed. Finally,

there is also a www folder that contains some special statistical functions and also the *CSS* file. Figure 24 is a scheme of the App files distribution.

4.4. Tidyverse

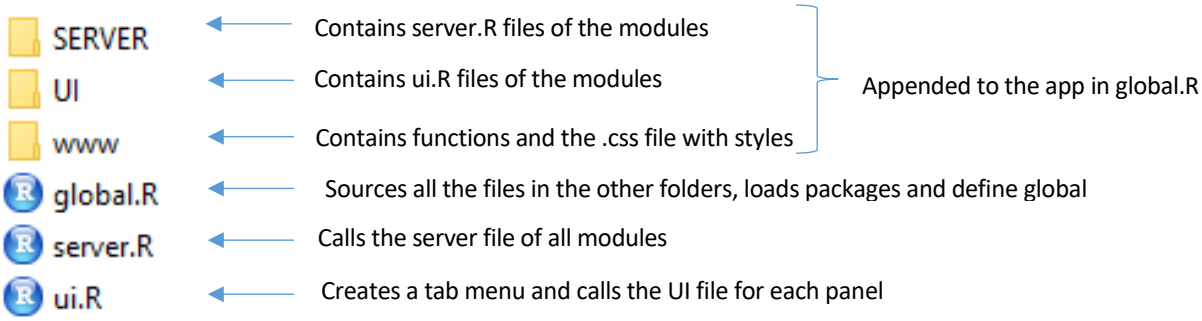


Figure 24. Distribution of the App files

Tidyverse is a wide set of R packages which offer functions that make data manipulation and visualisation easier, tidier and simpler. It has been developed by Rstudio, and follows the rule the simplier, the better. Hadley Wickham and Garrett Grolemund explain in detail its philosophy in the *R for Data Science* [15] book. The packages used for this project are listed below:

4.4.1. dplyr, tidyr, tibbles and pipes

Those are packages used for having a tidier data through manipulating the dataframes.

Tibbles are a new realisation of data frames that allow a clearer way of subsetting and a compacter display when printed. Although mainly tibbles will be used in the coding, they will always be referred as data frames, as this is the general case. *Dplyr* allows to subset, filter or select parts of a dataframe, but also to compute new data frames or to expand the existing ones. *Tidyr* is used to change the way the data is saved in the dataframe, being able to spread or gather it depending on the needs. Finally, *pipes* allow to link one operation to another for the same data frame, resulting in a shorter and clearer code.

4.4.2. ggplot

Ggplot is the main package used in R for data visualization, created by Hadley Wickham, now considered an eminency in R programming. It is a very flexible and versatile tool that can produce almost every plot that can be imagined.

It requires a data frame with the information, and then creates a *ggplot* object which can be later plotted in many different ways, depending on the extra functions or parameters added to it. Those are all the necessary parts to make and customize a plot, and were named by Hadley Wickham the "grammar of graphics", inspired by Leland Wilkinson book [16]:

- Aesthetics: Roles that the variables play in each graph. A variable may control where points appear, the color or shape of a point, the height of a bar…
- Geoms: The geometric objects to be plotted: bars, points, lines…
- Statistics: Functions like linear regression
- Scales and themes: Sizes, colours, axis, labels, background…
- Facets : Allow to make multiple graphics according one variable

Below, an example of usage is provided:

First, the plot object is created with the command below, but a plot completely blank will be shown, as no aesthetics or geometry have been told.

```
plotobject<-ggplot(diamonds)
```

Notice that the data frame for the plot is already given, which is called diamonds, which is indeed a tibble already incorporated in R, shown in Figure 25.

```
> diamonds
# A tibble: 53,940 x 10
   carat       cut color clarity depth table price     x     y     z
   <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
 2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
 3  0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
 4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
 5  0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
 6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
 7  0.24 Very Good     I    VVS1  62.3    57   336  3.95  3.98  2.47
 8  0.26 Very Good     H     SI1  61.9    55   337  4.07  4.11  2.53
 9  0.22      Fair     E     VS2  65.1    61   337  3.87  3.78  2.49
10  0.23 Very Good     H     VS1  59.4    61   338  4.00  4.05  2.39
# ... with 53,930 more rows
```

Figure 25. Diamonds tibble

So the next step is to decide the aesthetics, which is the same as deciding which variables are going to be mapped and how. It is chosen that X variable is going to be the carat and Y variable will represent the price, while the cut will be shown by a color. The geom also needs to be chosen, in this case a point, which produce the scatterplot in Figure 26 .

```
> plotobject<-plotobject+geom_point(aes(x=carat,y=price,color=cut))
```

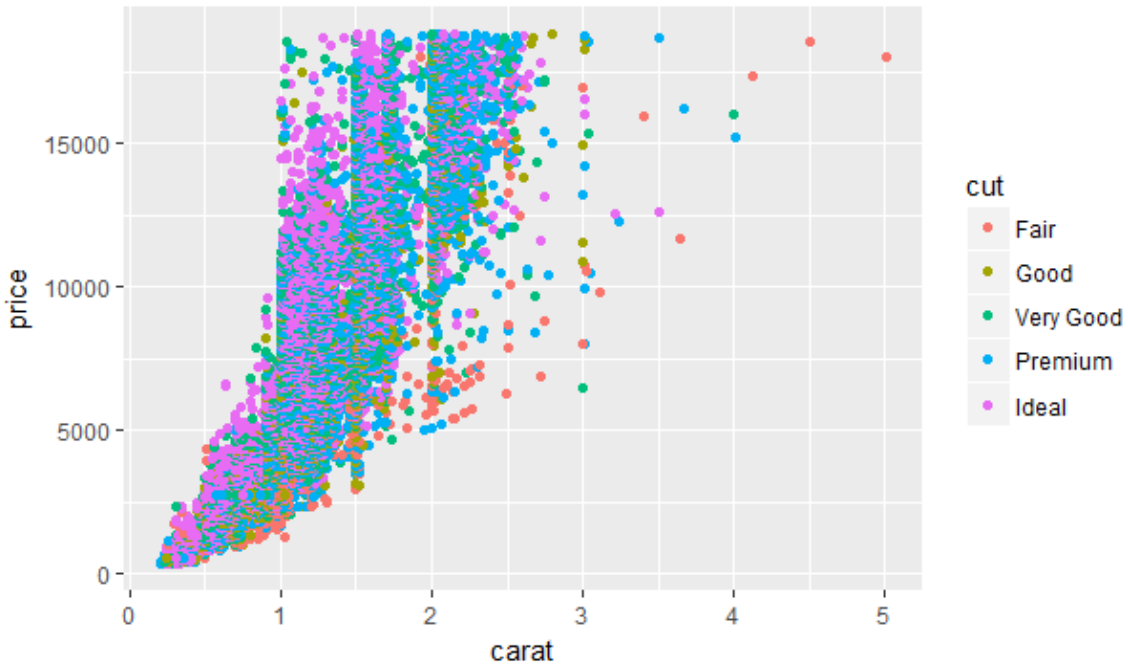


Figure 26. Scatter plot produced with ggplot2

Other layers of the grammar of graphics can be changed, like the theme or the faceting (Figure 27). Here just some basic options were used, and the statistics layer is still missing. However, it already provides an idea of how powerful this tool is, since it really gives full control of what

needs to be mapped, how, and also helps the user to make it look nice. *R Graphics Cookbook* [17] and *ggplot2: Elegant Graphics for Data Analysis* [18] provide all the necessary information to master visualization of data with ggplot, and are written by the developers of the package themselves.

```
> plotobject<-plotobject+theme_dark()+facet_wrap(~clarity)
```

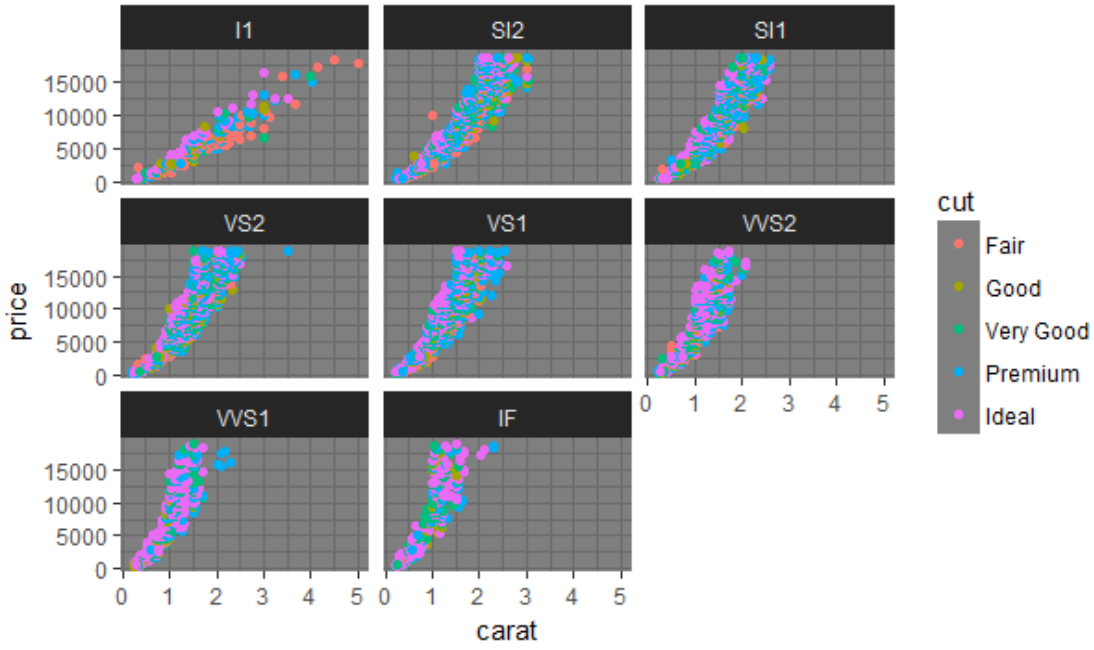


Figure 27. Final plot after changing the theme and faceting

Here the changes in the plot have been introduced sequentially, but normally everything is plotted in just one command, using the "+" operator:

```
>ggplot(diamonds)+geom_point(aes(x=carat,y=price,color=cut))+theme_dark()+
facet_wrap(~clarity)
```

If more interactivity is required, another package called *ggiraph*, developed by David Gohel [19], can be used on top of *ggplot*. This package allows the selection of the geom elements (points, bars…) and also to display a tooltip when the mouse is on top of them.

5. The App

This chapter will describe the App implemented in this project. The general layout of the app will be set in the *UI* file, which will create a page with a top navigation menu (each tab of the menu will link to the correspondent panel), a footer, and will also call the *CSS* file to import the theme. In the *server* all the logic will happen. But these two files will be very short, since all the App has been modularised following the structure explained in section 4.3

5.1. Data acquisition panel

5.1.1. Data import

As default, this is going to be the panel showed when initializing the App. At first, it will display just one possible input, that will ask the user to upload the necessary data for the study. All the other tabs are disabled, showing a message to the user that indicates that data needs to be uploaded (Figure 28).

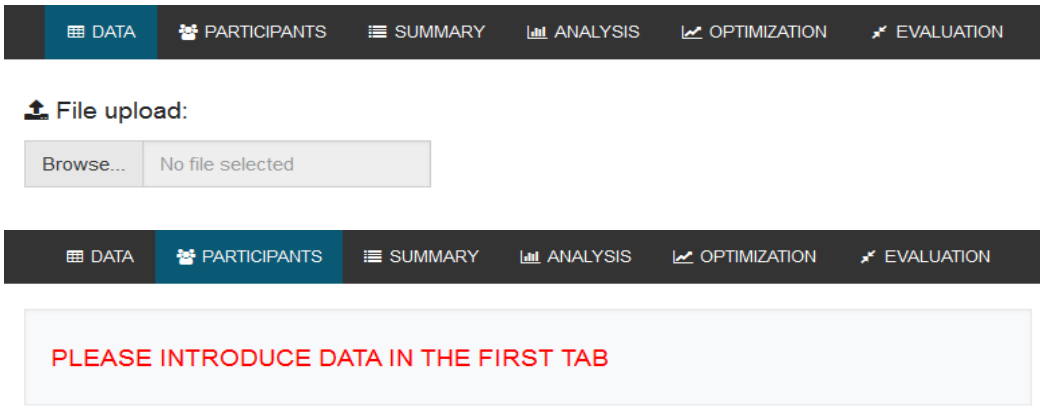


Figure 28. Display of the tabs before uploading data

As it has already been said, the data to be uploaded needs to have a special format, otherwise nothing would work. The required format is a *Rdata* file, which is a special R file that saves all the objects in its enviroment. In the *RData* file required for this App, only two objects are needed, which have to be data frames (or tibbles).

One must be called *data*, and should contain all the information about the scores given by the participants, in the following way:

- One column for each factor. The column name must be the name of the factor, whereas the elements contained must be its levels.
- One column named *Person*, containig the names of the participants. It is important that no special characters are included there, except from accents.
- One column named *KW*, containing the kansei words.
- One column called *Rating*, with the punctuation given by the correspondent person to the kansei word in the prototype defined by the factor columns.

An example of how this data frame looks like in the juices example is shown in Figure 29.

```
> data
# A tibble: 2,688 x 8
    Straw Decoration    Ice Container  Color Person         KW Rating
   <fctr>     <fctr> <fctr>    <fctr> <fctr> <fctr>      <chr>  <dbl>
 1     no         no     no     glass orange   Imma Refreshing      4
 2    yes         no     no     glass yellow   Imma Refreshing      1
 3     no        yes     no     glass yellow   Imma Refreshing      1
 4    yes        yes     no     glass orange   Imma Refreshing      1
 5     no         no    yes     glass yellow   Imma Refreshing      1
 6    yes         no    yes     glass orange   Imma Refreshing      2
 7     no        yes    yes     glass orange   Imma Refreshing      3
 8    yes        yes    yes     glass yellow   Imma Refreshing      4
 9     no         no     no    goblet yellow   Imma Refreshing      2
10    yes         no     no    goblet orange   Imma Refreshing      1
# ... with 2,678 more rows
```

Figure 29. Tibble containing data

The second data frame to be included must be named participants, and must have the following format:

- A column called *Name*, containing participants names (here it is also important to have just conventional characters).
- One extra column for each stratification option.

```
> participants
# A tibble: 24 x 3
      Name Gender       Age
    <fctr> <fctr>    <fctr>
 1 Antonio   Male    23-30
 2   Belén Female below 22
 3   Núria Female below 22
 4   Carla Female    23-30
 5   David   Male    23-30
 6 Enrique   Male    23-30
 7    Erli Female    23-30
 8     Eva Female below 22
 9 Guillem   Male below 22
10  Héctor   Male  over 40
# ... with 14 more rows
```

Figure 30. Tibble containing participants

All this data is loaded into a local enviroment inside an observer. It must be wrapped in a reactive context, because file upload is an input that can always change. Inside this observer, data is processed and prepared for the App needs, creating some extra data frames.

All these new data frames, which will be necessary for the other modules, are saved into a *reactiveValues* object called *datav*, wich is a source in the reactive graph. It is indeed a list where its elements are reactive, being the best way to transfer at once all the uploaded data to any part of the App (remember that modules do not acces the variables from outside, unless you explicitly pass them while calling the module). This observer, which will fullfill the *datav* object, will be the only code in the main server script (apart from the modules calling).

The *datav* contains the following elements:

- *outliers*: list of the detected outliers after using the function explained in section 3.3.2.
- *df*: the same data frame as data, but with the Person column arranged alphabetically and without accents.
- *vectorWords*: a vector with the words.
- *vectorFactors*: a vector with the factors.
- *stratOptions*: a vector with the stratification options.
- *stratList*: A list where every element is named as a stratification option and contains all the possible levels
- *prototypes*: a data frame containing all the possible factor combinations (each one is called prototype, even if it has never been shown to the participants), and with a last column giving a number to each prototype.
- *participants*: apart from the columns already contained in the participants source object, 3 extra columns are included: emotionality, with the mean of the participants' punctuations, diversity, with its standard deviation, and outlier, which is going to be true if the participant is in the outliers vector.

It is important to say that if the way of importing the data wants to be changed or the format in which it is loaded, there would be no problem if a *datav* object is still created with all the previous elements (those do need to have the specific format).

From now on in this chapter, everything that will be shown is what is seen in the app, after loading the data of the juices experiment.

5.1.2. Summary of participants and independence indexes

The first panel, once data is loaded, displays all the information previously stated in the wireframe of the aplication. The summary of the participants is going to be done with a bar chart for every stratification option, which is created from the participants data frame. The colours for the levels of each option are set at the beginning of the *global.R* file, and will be the same along all the panels.

The implementation of the two confusion indexes has changed a little bit. Instead of confusion, where the best value is a zero, the independence is plotted, because people normally tend to see with better eyes larger numbers. As seen in Figure 31, the global indicator is shown as a *ggplot* object made of a bar coloured with a gradient plus a vertical bar that shows the index, while the matrix is a simple heatmap (*geom_tile* in *ggplot*). The data necessary to plot those graphics is provided by the function explained in section 3.3.1

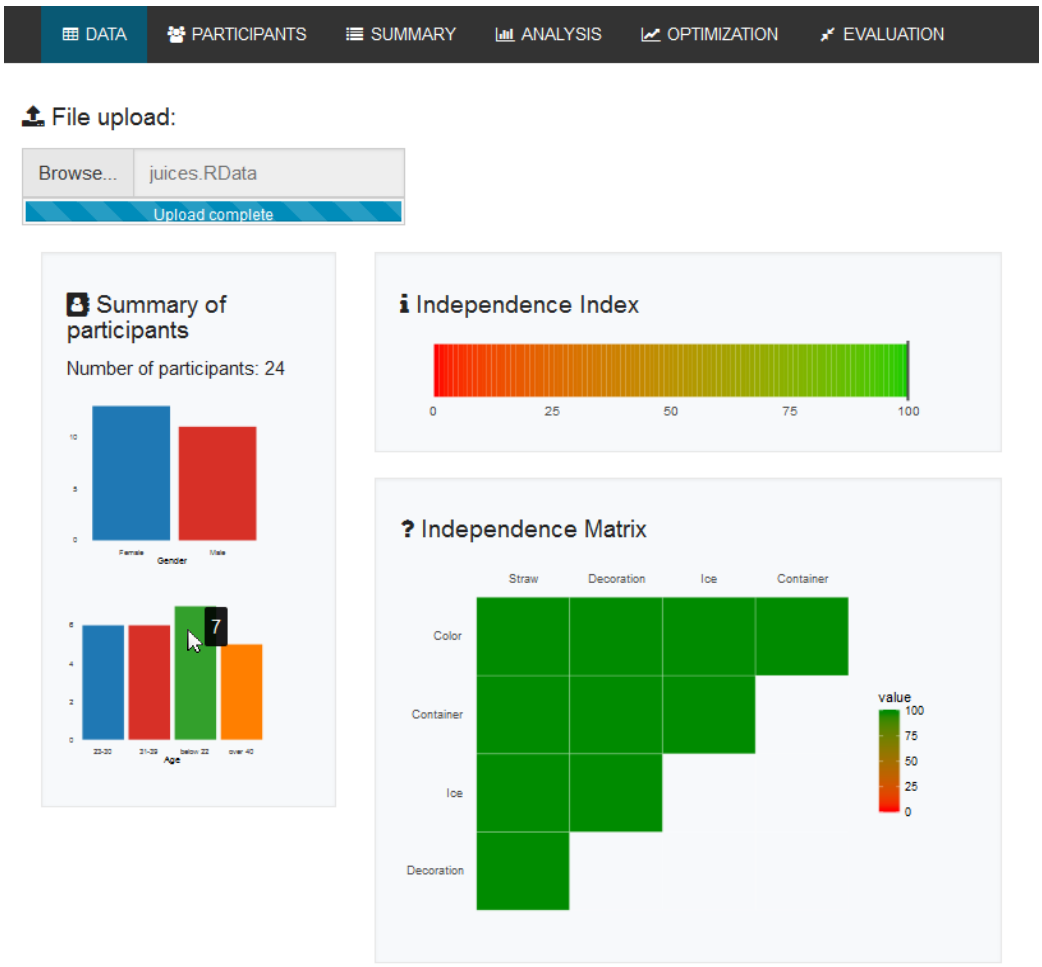


Figure 31. Screen capture of the first panel

5.2. Participants information panel

This panel has 3 sub panels plus a footer, as seen in Figure 32. The footer is common for all the tabs, except in the data panel, where it is not needed. It is composed by two collapsible elements, but in the participants tab just one is shown, because the other one is not required yet. Anyway, the performance of the footer will be explained now, because the second element works as the first one.

The footer functionalities are all defined in one module called *panel0*, which is also called from the *server*, and placed in the *UI* position specially designed for a footer. This module requires as arguments only the *datav* reactive list and the current tab (this last to know when to hide the footer or a part of it). From *datav*, only the *stratOptions* and *stratList* objects are used to create the radio buttons input plus the legend. Those are placed into a panel, which is initially collapsed, till the user clicks on it (Figure 32).

Figure 32. Screen capture of the participants tab

This *panel0* module returns to the server a reactive expression (an intermediate conductor in the reactive graph) containing the actual value of the stratification called *stratValue*, which will be necessary for all the other modules.

From now, all other functionalities of this section have been programed inside the *panel2* module, which requires the *datav* variable as well, but also the *stratValue* we have been talking right now.

Regarding the participants compass, it is a scatterplot produced by the participants data frame from the *datav* object, where *Emotionality* column is plotted in the X axis, *Diversity* in the Y and the shape differentiates the outliers. If a stratification would be required (like in Figure 32), then the filling of the dots must be done according to the column of the corresponding stratification option. This requires a different formulation of the plot, which is coded through an *if* condition that distinguishes when there is stratification and when not.

Furthermore, this is an interactive graphic, which is achieved thanks to the *ggiraph* package. As a consequence, the dots can be clicked, resulting in a reduced color intensity (parameter alpha), and a tooltip with the value of the *Person* column. This value is also saved in a reactive expression called selected, which will be used by the other sub panels.

In the participants information there is an initial message asking to click on one participant to see its information. Once that is done, it shows in a text output its characteristics, obtained also from *datav*'s participants data frame.

In the scores sub panel, some manipulation of the *datav*'s *df* dataframe is done, with the help of *dplyr*, *pipes* and *tidyr*. A filter by participant is done, only the desired columns (*Prototype*, *KW* and *Rating*) are selected and the *KW* column is spread to one column for each word, containing the corresponding scores. Then this is displayed in a *formattable* [20], a special package that can give format to data frames. In this case just the functionality to display a bar of length proportional to the score is used.

Finally, this formattable is converted to a *datatable*, another class from the *DT* package [21], an interface for R to the *DataTable* JavaScript library. It allows a more interactive display of the data frame. In this case, only two functionalities have been used, the possibility to scroll horizontally and to order by a column.

5.3. Summary panel

This is the panel where a first overview of the results is implemented. As it can be seen in Figure 33, it consists in a scatterplot very similar to the previous one, another heatmap and the already existing footer, this time the whole of it.

Apart from what was already shown in the footer, the stratification options, another collapsible appears here. Inside there will be always 3 radio buttons, which allow the user to choose the significance level and this way change slightly the results by being more or less strict with the evidences. This second collapsible is also produced in the *panel0* module, and its current value is returned to the *server* file as the *safetyLevel* variable. This variable will be passed then to all modules that need it.

The other two sub panels are both created in the *panel3* module, which needs the variables *datav*, *stratValue* and *safetyLevel*.

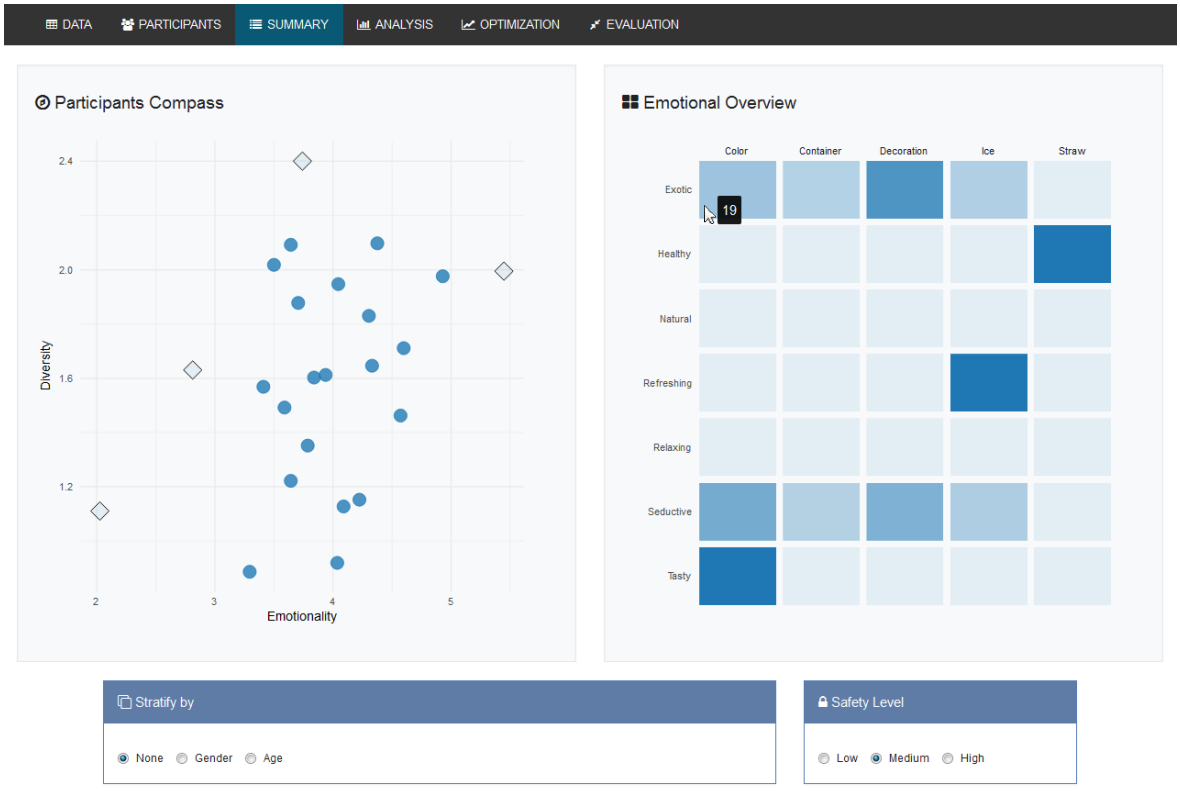


Figure 33. Screen capture of the summary tab

The scatterplot is created similarly as the one in the previous tab. But one tiny characteristic in the display demands a quite bigger change in the code. First, a reactive expression called

realSelected is created, which contains all the selected dots. Initially this expression contains the participants detected as outliers. Then, the scatterplot is produced using ggplot as it was done in the previous tab. But this time, 4 layers of *geom_point* must be employed, to achieve the desired result:

- One layer to plot interactively the participants in *realSelected*, in a colour that has a lower alpha.
- Two extra layers, one for the outliers and one for the no-outliers, that plots for the participants in *realSelected* the contour in black. If this layer did not exist, the alpha in the previous layer would also affect the contour and the point could not be distinguished properly. Two layers are needed, one for each different shape.
- A final layer, showing all the points that are not selected in the same way as they were in the other scatterplot.

To draw the heatmap, results from the study must be obtained. Of course this is done by the *mQT1* function explained in section 3.3.3, but the format must be changed. Furthermore, this is an expensive computational calculation, and requires quite a lot of time. For this reason, a list is designed to include the results in the appropriate format, for each stratification option and security level. This list is going to be fulfilled with a new item each time this item needs to be computed because the user had required it.

This way, only the first time when a combination of stratification option and security level is chosen, the calculations will be performed. However, each time a participant is included or excluded, everything will be erased, since there are numerous selected participants' combinations, and it would be difficult to define a proper element naming and know which list elements should be accessed. Moreover, including and excluding participants is an action that is expected to be seldom performed.

To obtain this list, which will be called *resultList*, several steps are needed:

- Create a *reactiveValues* object (a source in the reactive graph), named *memory*, that will keep track of the stratification and security options combinations already computed.
- Create the *resultList* object, a reactive expression (an intermediate conductor in the reactive graph). Inside this object, the following is done:
    - The already computed combinations stored in *memory* are copied into a

temporary list, named *tempList*, which is initialized at zero if memory is empty.

- If the current stratification-security options combination is not yet stored, a new empty data frame is created. This data frame is going to be filled while going through all levels, kansei words and factors, to obtain the data in the required shape. Finally, this new data frame is added to the *tempList*, and then this list is returned by *resultList,* the reactive object and stored in the *memory* as well.

This *resultList* object will be returned by the *panel3* module, and this way all the other modules will be able to access the information of the results in the proper format, a list that contains for each stratification and security options combination, a dataframe as the one seen in Figure 34. It contains for each level of each design factor, for each kansei word, and for each stratification level of the corresponding stratification option, the category score (*CS*), the importance and the *Basis* (this last common for each kansei word).

```
# A tibble: 140 x 7
       Factor  Level       CS StratLevel         KW Importance   Basis
       <fctr> <fctr>    <dbl>      <chr>      <chr>      <dbl>   <dbl>
 1      Color orange  0.00000       Male Refreshing          0 4.58125
 2      Color yellow  0.00000       Male Refreshing          0 4.58125
 3  Container  glass  0.00000       Male Refreshing          0 4.58125
 4  Container goblet  0.00000       Male Refreshing          0 4.58125
 5 Decoration     no  0.00000       Male Refreshing          0 4.58125
 6 Decoration    yes  0.00000       Male Refreshing          0 4.58125
 7        Ice     no -0.64375       Male Refreshing        100 4.58125
 8        Ice    yes  0.64375       Male Refreshing        100 4.58125
 9      Straw     no  0.00000       Male Refreshing          0 4.58125
10      Straw    yes  0.00000       Male Refreshing          0 4.58125
# ... with 130 more rows
```

Figure 34. Data frame of the results

Finally, the heatmap must also be created. It will need the recently explained *resultList* reactive expression. A new column containing factor and level combinations is included, and then a ggplot plot object of that column versus the *KW* is done, where the alpha parameter is the Importance (it is scaled before to adapt its range for better visualizations). Finally, the fill parameter is set to match the *StratLevel* column, and a faceting is done by *Factors* in the columns. The results of this stratified version can be seen in Figure 35. Since an interactive *geom* from *ggiraph* extension was used, the tooltip is set to show the importance.

Figure 35. Screen capture of the summary panel once stratified by age

5.4. Detailed analysis panel

This panel will need the *datav*, stratValue, *safetyLevel* and *resultList* objects, which will be provided when it is called. Its particularity is that it must have one tab for each kansei word. In each tab, the layout and logic required will be the same, but using a different slice of the corresponding *resultList* data frame. That is why at first a list with all the tabs is crated, called *myTabs*, and then a tabset panel is generated out of it. Each of the elements of *myTabs* generates a *tabPanel*, with its layout and its two outputs, which look as in Figure 36. They are generated as explained below:

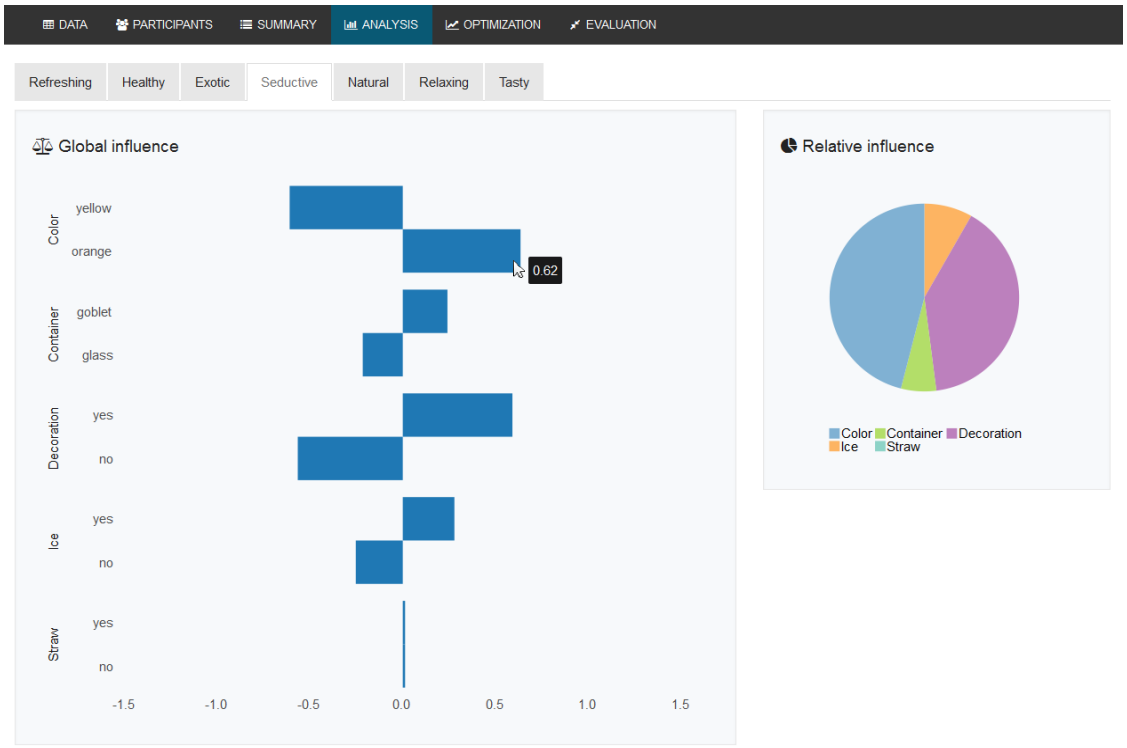


Figure 36. Screen capture of the analysis panel

- Global influence: The right *resultList* element is filtered by the corresponding kansei word, and this new data frame is used in a ggplot object. The *CSplot* column is created out of *CS*, to make zeros a 1% of the total range, and this way a very short bar appears, as with the factor Straw in Figure 36. This *CSplot* column will be the y, the *Level* the *x*, and *StratValue* the *fill* and *group*. After making a coordinates flip, the *geom_bar* with position "dodge" will be employed (this places the bars when stratifying side by side, as seen in Figure 37), and a row facet through factors will produce the final plot. As the *ggiraph* extension is also used here, the tooltip is set to show the initial *CS* column.

- Relative influence: To create this plot, the same dataframe is used as by the previous plot, with slight extra filtering. Then, a ggplot object is created, where the *y* is mapped by *Importance*, and the *Factor* takes account for the *fill,* while *x* is left empty. Later, the *geom_bar* is used again, but afterwards coordinates are changed to polar, having the *Y* as the angle. Finally, a facet through the *StratLevel* is done, which creates a pie chart for each level, as seen in Figure 37. The interactivity is used again, having a tooltip displaying *Importance*.

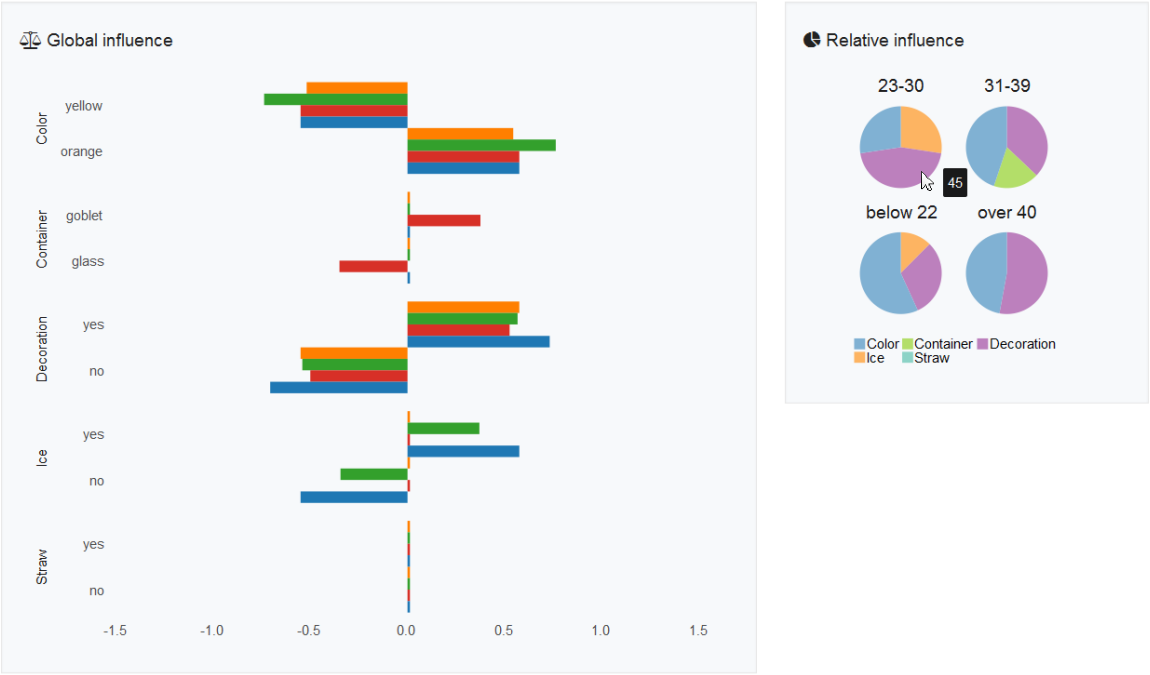


Figure 37. Screen capture of the analysis panel after stratifying by age

5.5. Best prototype panel

This panel is all implemented in the *panel5* module that generates the radar plot, the podium tables and the side bar as seen in Figure 38 (the footer is in *panel0* as always). However, it has a particularity; it is the only module that contains another module nested inside. This other module is employed just to make the sidebar. The *panel5* module requires from *server* the *datav*, *stratValue*, *safetyLevel* and *resultList* objects, and must be called in a reactive environment, because the module needs to use directly *datav* values outside of observers.

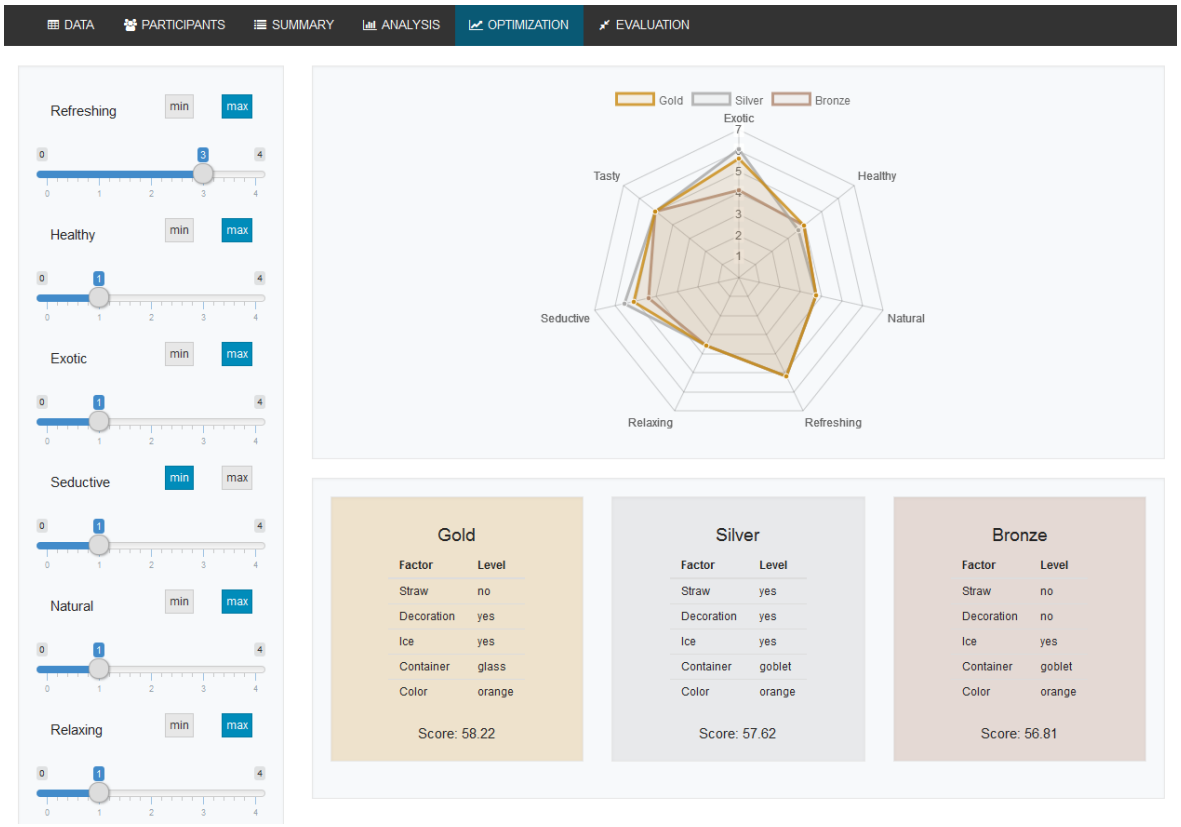


Figure 38. Screen capture of the optimization tab

The nested module, responsible for creating the sidebar content, is called *sliders* and it will be the first thing to be explained. The decision to make a module have been taken because if a stratification is required, the creation of the sliders must be repeated for each level. Creating a simple list as in the previous tab was not possible here, due to more required complexity.

And why is this module so complex? The main reason is because it must keep track of all the input values (*slider*, and *min* or *max*), for all words and for all levels of each stratification option (this last stage is achieved through the calling of different modules). And all this must be done within a reactive context. The only way to store so many reactive variables is to create a

reactiveValues list (a source in the reactive graph). This object is going to be called *v*, from values, and will contain for each word:

- An indicator of the *css* class to use to show the max and min buttons (one class for active, and one for inactive). It will also serve as an indicator to know which button is active.
- The value of the slider.

Afterwards, an easy layout distribution of one slider and the *max* and the *min* buttons is done in a *fluidpage*, which is then included into a list (one element for each word). Out of this list, the complete sidebar column is produced, adding the *reset* and *inactivate* buttons at the end. In the module there are also observers which change the correspondent *v* value if any input is changed. There is also one observer that sets all sliders values to one if *reset* button is pressed or to zero if it is the *inactivate* button.

It is important to understand that here the use of a module is crucial, meaning that a function could not have been employed instead. The reason is that the *v* value for a certain word (for example refreshing) and for one input (for example, the slider value) is going to be named equally in all the levels in all stratification options. But as a module has been called for each of them, they do not share the same namespace and they are automatically differentiated.

In the sliders module one last thing is left to be done, the creation of a reactive expression containing all the information with the inputs current values. This information is going to be taken from *v* and from *resultList* but is going to be prepared in a format ready to use for the parent module, resulting into a dataframe as the one seen in Figure 39.

```
# A tibble: 224 × 8
           KW    Score   Max Weight maxPunct minPunct Prototype Level
        <chr>    <dbl> <lgl>  <dbl>    <dbl>    <dbl>     <int> <chr>
1      Exotic 2.343750  TRUE      1        7        1         1  Male
2     Healthy 3.875000  TRUE      1        7        1         1  Male
3     Natural 3.750000  TRUE      1        7        1         1  Male
4  Refreshing 4.046875  TRUE      1        7        1         1  Male
5    Relaxing 4.046875  TRUE      1        7        1         1  Male
6   Seductive 3.398438  TRUE      1        7        1         1  Male
7       Tasty 5.359375  TRUE      1        7        1         1  Male
8      Exotic 2.343750  TRUE      1        7        1         2  Male
9     Healthy 3.875000  TRUE      1        7        1         2  Male
10    Natural 3.750000  TRUE      1        7        1         2  Male
# ... with 214 more rows
```

Figure 39. Dataframe returned for each level by the sliders module

It directly contains for each one of the prototypes of the *datav* object, a *Score* (the *Basis* plus the sum of all *CS*, if it is to be maximized and the weight). The best and worst score (7 and 1 respectively) are also there, as well as the corresponding level.

As it has already been said, the *sliders* module is called once for each level (not only for the levels of the current stratValue, but for all of them). Apart from the level itself, the *datav*, *resultList* and *safetyLevel* are also provided in the call of the module, and the reactive expression with the dataframe in Figure 39 is stored in a list called *radarData* that has one element per level. For the *UI* part of the module, a tabset (or none, if no stratification is selected) is created in a *renderUI* and placed in the sidebar, which will show in each tab the content of the corresponding module.

Now, it must be explained how from the *radarData* list the desired plots are produced. First of all a new *reactiveValues* object is defined, called *bestList*. Then, following the algorithm explained in the section 3.3.4, which is going to be implemented through chained *dplyr* operations, the best prototypes are obtained in a data frame (Figure 40) stored as a *Prototypes* entry to *bestList*.

```
# A tibble: 32 x 2
   Prototype OverallPunct
       <int>        <dbl>
1         15     60.48555
2         16     59.44940
3          7     58.76475
4          8     57.75808
5         11     54.89553
6         13     54.18876
7         12     53.95515
8         14     53.26047
9          3     53.10898
10        31     53.08367
# ... with 22 more rows
```

Figure 40. Prototypes object

The three best are saved in a *bestPrototypes* entry. The *bestList* object will also include the *relevantLevels* vector, with all the levels that have not been inactivated (all sliders to zero). This last element is later used to know which levels must be hidden in the radar chart (if a level was inactivated it is not shown anymore, as it means that it is not of the user's interest).

Now, the radar chart is ready to be produced. For each level, a dataframe as the one in Figure 41 is created out of operations with *pipes* and *dplyr* and the *bestPrototypes* and *radarData* objects. This dataframe is stored also in *bestList* as *dataPlot*. This is done because its information, together with the already present in *bestList*, is going to be returned by *panel5* module since the *panel6* module will need it. This *radarPlot* dataframe, combined with the *radarchart* package [22], which is a wrapper for the *JavaScript Chart* library, produces an output for each level with the radar plot already seen in Figure 39.

```
# A tibble: 7 x 4
          KW `Prototype 16` `Prototype 7` `Prototype 11`
       <chr>          <dbl>         <dbl>          <dbl>
1     Exotic           6.09          5.62           5.45
2    Healthy           3.63          3.96           3.96
3    Natural           3.77          3.77           3.77
4 Refreshing           5.19          5.19           3.76
5   Relaxing           3.58          3.58           3.58
6  Seductive           5.56          5.10           5.03
7      Tasty           5.04          5.04           5.04
```

Figure 41. Dataframe used to make the radar charts

Finally, the three tables containing the factor level combination for the three best prototypes is to be coded. A normal *renderTable* function is used to show the levels stored in the *datav*'s prototype dataframe for the corresponding prototypes. Below, the score (*OverallPunct* column from *Prototypes* dataframe) is also displayed.

In Figure 42 it can be seen how the whole panel looks like if stratification is done. The sidebars turn to another tabPanel, and the radar chart is doubled. Notice that the best prototypes are the same for Male and Female, because the App returns always the best overall prototype.

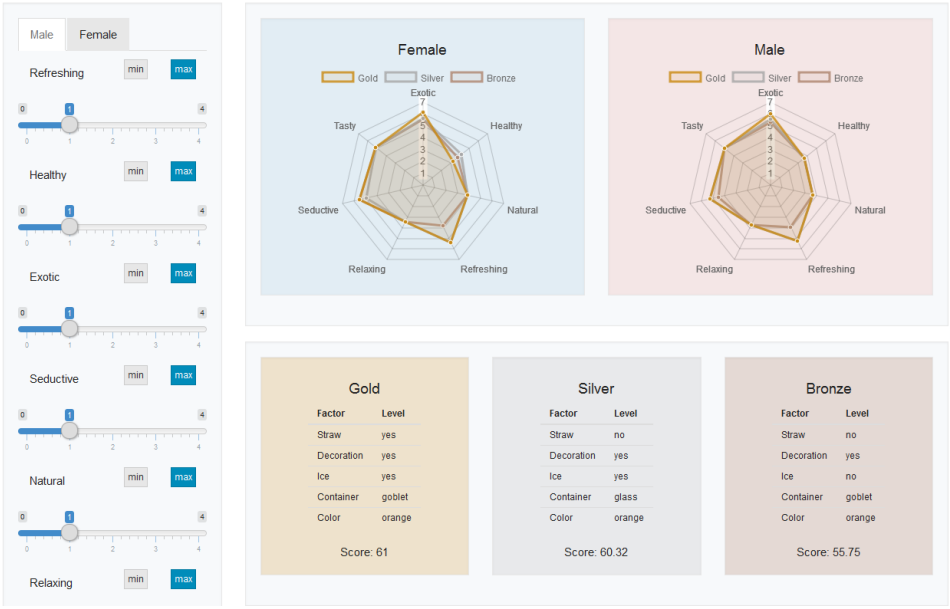


Figure 42. Screen capture of the optimization tab after stratifying

However, this prototypes perform different in each level and 2 charts are required. If a level was inactivated, its radar plot would disappear, as it has already been said. This panel plays a role in limiting the number of maximum levels for a stratification option, because plotting more than 4 radar charts would cause them to be too small or to have a too large panel.

5.6. Evaluation panel

This last panel is coded similarly as the previous one. The outputs are the same (a sidebar, a radarchart and some tables, as can be seen in Figure 43). The code where the general layout is defined will change only a little bit, but the subpanels will have to be modified in depth. Anyway, everything is implemented through the *panel6* module, which as the previous one it has to be called inside an observer. It needs as arguments the usual *datav*, *resultList*, *stratvalue* and *safetyLevel* plus the *bestList reactiveValues* object returned by *panel5* module.

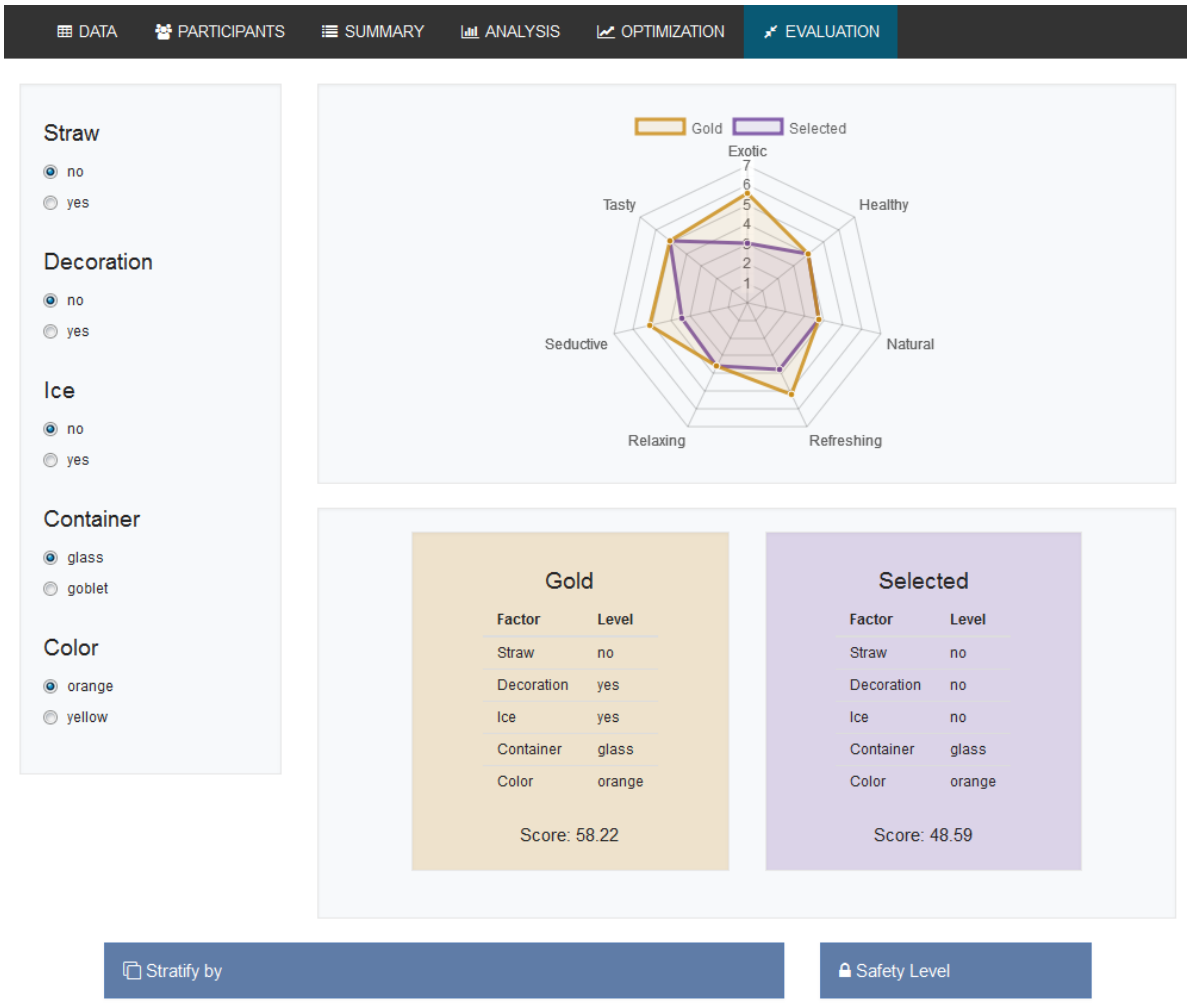


Figure 43. Screen capture of the evaluation panel

The sidebar is the part that mostly changes. To create it, a loop goes through the *vectorFactors* from *datav*, takes its levels and display them as radio buttons options.

To generate the radar chart, the *dataPlot* object is subseted to maintain only the *gold*

prototype, and then the *Selected* prototype performance is calculated from *resultList* similarly as it was done inside the *sliders* module.

To plot the tables, the gold one is created as in the previous panel, using the *bestPrototypes* element of *bestList*. The other one is easily created by making one column with all the factors, and a second column that contains the corresponding input value of that factor.

Here a stratification only changes the radar charts, which replicate itselves one time for each level. If a level was inactivated in the previous panel, as this information is included in the *bestList* object, it would not appear here as well.

5.7. Initial testing and feedback

An extensive testing is needed to guarantee both the correct performance and usability of the App. Taking into account the limited time framework for developing this work, this testing procedure has just started. However, the initial results look very promising.

The App has been tested with 4 different datasets coming from real kansei engineering studies. Some minor bugs have been corrected thanks to this tests (basically due to problems with character encodings).

The App has also been shown to 2 experts in product design from ELISAVA School of Design and Engineering (a professor and a PhD student). Both the user interface of the App and outputs obtained got a very positive feedback. Interpretation of the statistical results obtained were correct in spite of the fact that these two experts have a very scarce statistical background. This was one of the main objectives when creating the App. The two experts from ELISAVA suggested the use of softer colors in the graphs (not primary colors) and sans-serif fonts (such as Helvetica) for the texts.

6. Economic cost

Studying of the economic view is necessary for any engineering project. This section details the cost analysis.

The elaboration of this project has involved the usage of different pieces of software, but most of them had a free license. For the one that was not open software, the annual cost has been proportionally included, only taking into account the 5 months of the duration of the project.

Design and development costs include all costs related to the creation of the actual program: the previous investigation and market studies, the conceptual design, the UI design, the actual programming and the elaboration of the documentation. Indirect costs are computed as a 10% of the total cost.

All the costs can be visualized in Table 9.

Table 9. Costs of the project

App Project Costs				
Software Costs	**Annual cost [€]**	**5 month cost [€]**	**Usage**	**Cost [€]**
R (includes Rstudio and all packages)	0	0	100%	0
Microsoft Office Professional	600	250	80%	200
TOTAL				**200**
Design and development Costs	**Hours [h]**	**Cost per hour [€/h]**		**Cost [€]**
Initial documentation and investigation	50	30		1500
Conceptual and theoretical design	25	30		750
UI Design	10	30		300
R programming	180	30		5400
Project documentation elaboration	40	30		1200
TOTAL				**9.150**
Indirect Costs				**Cost [€]**
10% SW, design and development costs				**935**
TOTAL COST				**10.285**

7. Conclusions

After completing this project and testing the created App, a review of the objectives that were set at the beginning must be done to check if they have been accomplished.

Looking at first to the general objectives (already shown in Table 1 in page 6), it can be seen that all of them have been achieved at its whole. A tool to analyse interactively data from KE studies have been created, the App itself. It can be assured that this general objective has been completed, because the App has been tested with different datasets with different characteristics, and has performed as expected in all of them.

The objective to provide results that are easily understood, especially by not statistically skilled people (like product designers) has also been reached. It has been checked after asking some experts of that branch. One key aspect to achieve it are the best prototypes that the App automatically display according to the user needs. This has been done thanks to the multicriteria optimization method that has been adapted to kansei engineering studies, another general objective.

The last objective of this block was to create a structured code, which has been accomplished through creating a modularised app.

Regarding the specific objectives (which were explained in Table 2 in page 7), it is clear that they have also been achieved. The App makes stratification possible in all panels and it also facilitates changing from one option to another while keeping all the information. This was one specific objective, as well as to give control to the user to change the main variables, which have been implemented with the possibility to exclude outliers or to change the security level.

Another specific objective was to get real-time response. In this case the target has been reached, but not as well as it would be desired. In most panels the App computes everything that must be rendered very fast, and the user perceives it instantly. However, there are two calculations that require more computational power, which lead to some seconds of delay: when the data is loaded for the first time and when a new part of that data is used for the study (excluding participants or changing the options with new values).

This dims a little bit the real-time perception, and should be one think to be taken into account

to improve. This can be left as a recommendation for the future, in case that more investigation on this project is done. The way to do it is trying to diminish the number of loops in the code and to check all functions from other packages, to find the most efficient alternative. This last must be done specially in the functions used to calculate the statistical models, which are the ones that slow at most the App.

The app is also user friendly and multiplatform (the two last specific objectives), due to the use of a responsive theme and matching color palettes. However, it is true that the product designers consulted have said that the colours could be changed slightly to improve the visual impact in the user. For this reason, if this project was to be continued or expanded, it would be recommended to check that part as well and maybe also change a little bit the styles of some parts of it.

Some other recommendations to consider if this project is to be continued can be given. One would be adding another feature in the App. This would create an automatic report, prepared to be printed or download in a common format (pdf, PowerPoint…), which would contain a summary of the results showed in the App with the current selected options.

Another suggestion would be to find a way to publish it on the internet, because until now it just exists locally. This should not be very difficult since *Shiny* already provides a server, but still some research should be done to find the best option and the final implementation.

After knowing all that have been stated in this chapter, it is clear that there are obviously both things that can be polished and new functionalities that could be added to the App. Anyway, it can be definitely concluded that this project has achieved successfully its goals, accomplishing what was pursued.

8. Bibliography

[1] L. Marco, *Statistical Methods in Kansei Engineering studies (Doctoral Thesis)*, Barcelona: Universitat Politècnica de Catalunya, 2011.

[2] H. Cramér, *Mathematical Methods of Statistics, Princeton University Press*, 1999.

[3] H. Àlvarez, *Análisis de valoraciones atípicas en los estudios de ingeniería Kansei (Doctoral Thesis)*, 2009.

[4] M. Hubert and P. J. Rousseeuw, *ROBPCA: a new approach to robust principal*, Technometrics, 2005.

[5] B. Escofier and J. Pagès, *Multiple factor analysis (AFMULT package)*, Computational Statistics & Data Analysis, vol. 18, 1994.

[6] N. Draper and H. Smith, *Applied regression analysis*, New York: Wiley-Interscience, 1998.

[7] C. Hayashi, *On the prediction of phenomena from qualitative data and the quantification of qualitative data from the mathematico-statistical point of view*, Annals of the Institute of Statistical Mathematics, vol. 3, no. 2, p. 69, 1952.

[8] J. Harrrington, *The Desirability Function*, Industrial Quality Control, 1965.

[9] G. Derringer and S. R. , *Simultaneous Optimization of Several Response Variables*, Journal of Quality Technology, 1980.

[10] R project, [Online]. Available: https://www.r-project.org/about.html. [Accessed May 2017].

[11] R Studio, [Online]. Available: https://www.rstudio.com/.

[12] Rstudio, *Shiny*, [Online]. Available: http://shiny.rstudio.com/. [Accessed April 2017].

[13] J. Cheng, *Rstudio*, 2016. [Online]. Available: https://www.rstudio.com/resources/videos/effective-reactive-programming/. [Accessed April 2017].

[14] G. Grolemund, Rstudio, 2016. [Online]. Available: https://www.rstudio.com/resources/videos/modularizing-shiny-app-code/. [Accessed April 2017].

[15] H. Wickham and G. Grolemund, *R for Data science*, O'Reilly, 2017.

[16] L. wilkinson, *The Grammar of graphics*, Springer, 1999.

[17] W. Chang, *R Graphics Cookbook*, O'Reilly, 2012.

[18] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*, Springer, 2010.

[19] D. Gohel, *ggiraph*, ArData, [Online]. Available: https://davidgohel.github.io/ggiraph/index.html. [Accessed May 2017].

[20] K. Ren, *CRAN. Formattable*, [Online]. Available: https://cran.r-project.org/web/packages/formattable/vignettes/formattable-data-frame.html. [Accessed June 2017].

[21] Y. Xie, *CRAN. DT*, [Online]. Available: https://cran.r-project.org/web/packages/DT/index.html. [Accessed June 2017].

[22] D. Ashton, *CRAN. radarchart*, [Online]. Available: https://cran.r-project.org/web/packages/radarchart/index.html. [Accessed June 2017].