Technical documentation

Data-driven similarity scoring in matching refugees and native Dutch people.

-

Joost van der Meulen - 1661839

MA Data-Driven Design

HU University of Applied Sciences

-

October 25th, 2020

Introduction

In this documentation, the decisions that have been made that led to the prototype can be found. The prototype will have a back- and a front side, which splits the document into the technical section and the section of screen design. For the actual working of the algorithm and creating the model, there has been worked in Jupyter Notebook and the program language Python. The creation of the prototype of the app has been done with the help of Invision Studio.

Table of Contents

Functioning of the app

For the problem that has been treated in the academic paper, there has been thought of a solution in which both target groups are involved. A prototype of an app will be made where the goal is to find clusters of people, based on common interest and personality aspects. A split up between the two target groups is made after the landing screen. After that, they go through several statements where they give an indication to what extent they agree by sliding the value indicator. This creates a profile where the participants have rated statements about interests, hobbies and personality aspects. When this is completed, an algorithm will help to find people that have answered similarly and then be linked to other people. Refugees will be linked with Dutch people, and Dutch people will be linked with refugees. After the statements have been rated, for every statement, there is a number between 1 and 10 saved. When the match is made, the user will see the persons that are in his cluster and can swipe between different people in the cluster. Hobbies and interested that are connected to a place where these activities can be done (think of going to the cinema) are clickable, and a map will open where this can be done. Also, the user can view their profile add more aspects such as work, a short biography and a profile picture can be added. Research has been done to make the decisions for the content and what variables to include in the matching procedure en what not.

Terminology

The two target groups that are involved are refugees and native Dutch people. This target group has been divided after the following question (Figure 1):

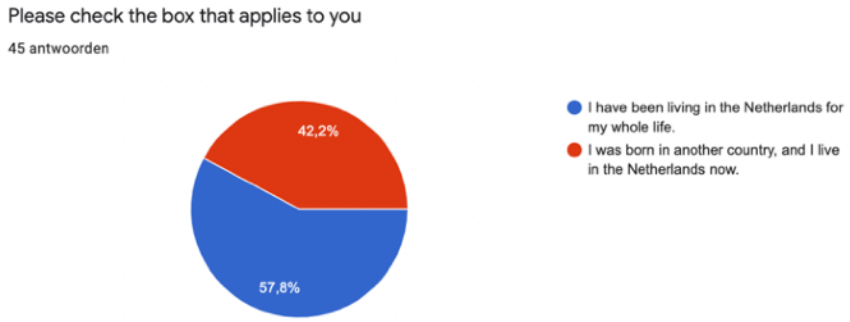


Figure 1. Target group division

To describe the target groups more easily in this documentation, the following terms will be used:

- *I have been living in the Netherlands for my whole life* - Native Dutch people.
- *I was born in another country, and live in the Netherlands now* - Refugees.

It can be that a person that has filled in the survey (and answered B) is not a refugee, but since the survey has been spread in a groups chat in which the most prominent part had to flee to the Netherlands, this term is used.

Data supply

The functioning of the app dependents on a group of people that have used the application before; otherwise, there is not enough data in the 'pool'. The finding of a fine match for the first flow of people will be a difficult task. Also, the more questions, the harder it is to connect people on several fronts. For this reason, first, a model is built where only two questions are asked (A and B), and 45 people that have supplied their data. Because the app is not realized, this data is not available (yet). For this reason, the data has to be randomly generated. After building the model for 45 participants and two questions (which can be applied in a more nearby future), a model will be built for 750 participants and ten questions, which is more applicable when the app has grown. The decision of which algorithm to choose will be made after building the model for 45 participants.

For the clustering of data entries, there are several clustering methods. The two algorithms that are used the most for this sort of cases are K-means clustering and Hierarchical Clustering (Seif, 2019). These two algorithms will be compared with each other, and a decision will be made. To work with data before the actual data is provided, the names of the participants will be generated by a Python package called Faker (Faker documentation, n.d.). With this function, the front names of the participants will be provided. The answers of the participants will be generated with the random integer function. In order to gain more insights into how both algorithms work, the functioning will now be explained.

Algorithms

K-means Clustering

K-means is a type of unsupervised machine learning where unlabelled data is used. The goal of the algorithm is to find groups in the data. The number of groups must be determined before applying the algorithm; the K represents the number of groups. In order to process the data, the K-means algorithm starts with a group of randomly selected centroids, which can be seen at the beginning point of every cluster. In the next stage, every data point allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The process is finished when no weight point changes all centroids together. By then, the clusters are formed, and the clusters can be

labelled (if wanted). If the decision is made to split the group into three, the K is set to 3. These groups score on similarity in their features. K-means can help analyze data before looking at the data; it allows us to discover groups that have formed organically (Amelia, 2018).

Agglomerative Hierarchical Clustering

Agglomerative clustering is a type of Hierarchical Clustering that groups objects based on their similarity. Agglomerative is a type of clustering that works bottom-up. The benefit of the algorithm is that the group can be made as specific as wanted, but also as big as wanted. The agglomerative clustering is good at identifying small clusters, while divisive clustering works better for identifying large clusters. The algorithm treats every object as a single cluster, which means that in the beginning, there will be as many clusters as objects (bottom-up). This results in a dendrogram, which represents all the objects. The position of the objects in the cluster is based on a method that calculates the similarity between two objects. This distance function can compute the distance between every pair of objects in a data set, and will determine what the position of the data points will be on the y-axis (Datanovia, 2018).

After calculating the distance between the objects, the *linkage* function can be used. This function groups pairs of objects into clusters and distinguishes the clusters from each other. A problem with Hierarchical Clustering is that it does not tell us how many clusters there are; the decision of how many clusters (K) still needs to be made, and then the dendrogram will be cut (Datanovia, 2018).

Clustering experiments

Clustering with 2 variables (Hierarchical)

The decision needs to be made whether the Hierarchical Clustering model is going to be used or the K-Means algorithm. A decision will be made which algorithm fits the best for the app. To compare the two algorithms, they have been tested with simulative data. The first models have been created with two answers of the participants, the first answer would be placed on the x-axis and the second answer on the y-axis. The plan is to first get grip on what is done by the algorithm and later on scale the amount of variables on the x-axis. In this case, there are 45 participants that answered two questions with moving the value indicator from 0 to 10 on the bar. The participants will only see 0 to 10, but the exact number of the indicator is being saved in a more specific number between 0 and 100. Every cell will receive their own (randomly) assigned number.

First, the libraries that will be recalled are imported (Figure 2). Not all libraries are standard installed with Anaconda, but can be installed with pip.

Agglomerative hierarchical clustering

Agglomerative clustering is a type of hierarchical clustering to group objects based on their similarity. The benefit of the algorithm is that the group can be made as specific as wanted, but also as big as wanted. Agglomerative clustering will say that the clustering has a bottom-up approach, and not a latter-down. This means that all datapoints first are clustered in one big cluster and thereafter be assigned to smaller clusters.

Libraries needed

First off, the libraries that are needed will be imported. Not all of these libraries are installed standard and installed by pip.

[379]:
```
import pandas as pd #Pandas is used for analysis and helps with mathematics. It offers structure in data and and numerical tables.
import numpy as np #Creates the arrays
import random
import string
import names #Names consists of a library of thousands of first- and last names
from faker.providers.person.en import Provider #This provides the 'fake names'
```

Figure 2. Libraries imported

The dataframe is created with a size of 45 (Figure 3). Also the target group is added (t), which is randomly generated. 1 stands for Native Dutch people, 0 for Refugees.

To split up the dataset into the Dutch people and the refugees, a 1 and a 0 is assigned to the row. 1 will stand for the Dutch people, 0 for refugee.

[398]:
```
df1 = pd.DataFrame(np.random.randint(1,101,size=(45, 2)), columns=list('AB')) #Every data entry will have a number between 1 and 100  assigned.
target_group = pd.DataFrame(np.random.randint(0,2,size=(45, 1)), columns=list('t')) #Target group
df2 = pd.concat([df1, target_group], axis=1)
df2.head(5) #Shows the first 5
```

[398]:

	A	B	t
0	23	12	1
1	39	42	1
2	85	32	0
3	82	24	0
4	64	98	1

Figure 3. First dataframe with answers and the target group

Next to the two results from two questions, the rows will have their own name assigned, which is the person who stated these two answers. A function is created and runs every time a new name is being inserted in a cell. Attaching a name to the row, will help distinguish the rows later on in the process. Now, the participants have a name and two data points to cluster them.

Next up, the data entries will be provided with a name. This is done because it helps recognizing the rows and brings it closer to the case.

[399]:
```
def random_names(name_type, size): #A function is being created and later will be recalled
    names = getattr(Provider, name_type)
    return np.random.choice(names, size=size) #size will be provided later on
```

[401]:
```
size = 45 #45 names will be generated.
df3 = pd.DataFrame(columns=['name']) #A new dataframe is created and later be merged
df3 ['name'] = random_names('first_names', size) #This fills in all the cells with a generated name
df = pd.concat([df3, df2], axis=1)
df.head(5)
```

[401]:

	name	A	B	t
0	Madeline	23	12	1
1	Cindy	39	42	1
2	Caddie	85	32	0
3	Deante	82	24	0
4	Kole	64	98	1

Figure 4. The addition of the names in the dataframe

Because there will be experimented with more algorithms, the Dataframe will now be exported to a CSV file that is saved in the same folder (Figure 5). Every time this cell runs, the file will be overwritten. This is being done because otherwise there would be generated new data and the clusters can not be compared to each other.

To test with the same data, this dataframe will be exported to a csv file. It is saved in the same folder and will be overwritten everytime it runs.

[24]:
```
df.to_csv ('participant_data.csv', index = False, header=True)
```

Figure 5. The dataframe is saved in a CSV file

Predicting the cluster

Now the the class is imported for clustering.

[9]:
```
import matplotlib.pyplot as plt #To plot the people
df.shape #It has 750 rows and 3 columns
```

[9]:
```
(45, 4)
```

[10]:
```
data = df.iloc[:, 1:4].values #Names and target group will be excluded for now
```

The affinity is set to euclidean, this means it will look at the distance between the datapoints; the length of the line between two data points. Ward is a linkage method that minimizes the variant between the clusters and is close (by it properties and efficiency) to K-means clustering; they share the same objective function.

[12]:
```
from sklearn.cluster import AgglomerativeClustering #The function that will cluster the data entries

cluster = AgglomerativeClustering(n_clusters=15, affinity='euclidean', linkage='ward')
cluster.fit_predict(data)
```

[12]:
```
array([12, 11,  3,  1, 11,  2,  5, 13,  7, 11, 14, 11,  8,  4, 10, 13, 11,
       14,  7,  2,  4,  2,  6,  9,  0, 13, 12,  1,  4,  0,  2,  8, 11, 12,
        1,  3,  2,  0,  6,  0,  9,  7,  6,  5,  0])
```

Figure 6. The participants will have a cluster assigned

Now, the participants will be assigned to a cluster by the function AgglomerativeClustering (Figure 6). Because the goal is to find a match between people that are very near to each other, the number of clusters is set to a (relatively) high number of 15. If more people would participate, the number of clusters can also be higher.

The participants are plotted and assigned to a cluster (Figure 7). The decision making for the plots can be redirected in the graph. Because the data is generated randomly, there are no real outliers. It can not be seen which data points belong to which person, but plotting the Hierarchical Dendogram will help in this.

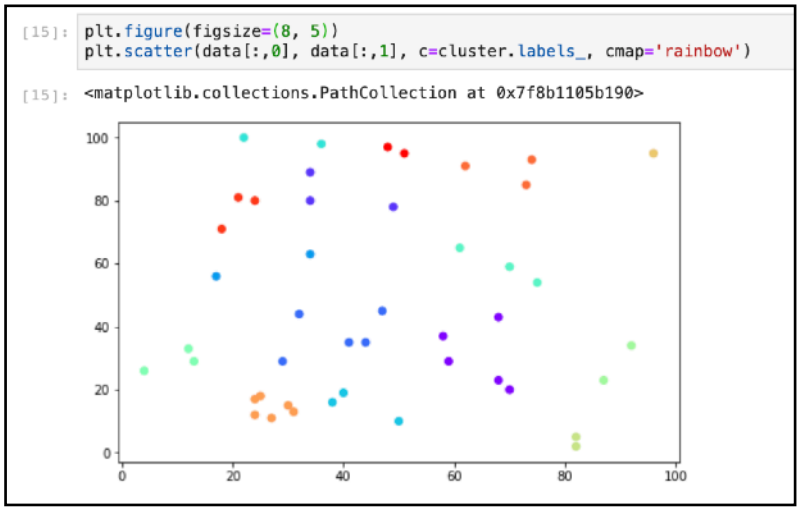


Figure 7. Scatterplot of the participants

As you can see in the plot below, the clustering process goes top-down (Figure 8). All data points are in one cluster (the blue lines), and the more they go down, the more specific the clusters will become. At this moment there are three big clusters that are separated by their color (orange, green, red). The benefit of hierarchical clustering is that also smaller clusters can be seen in the same graph.

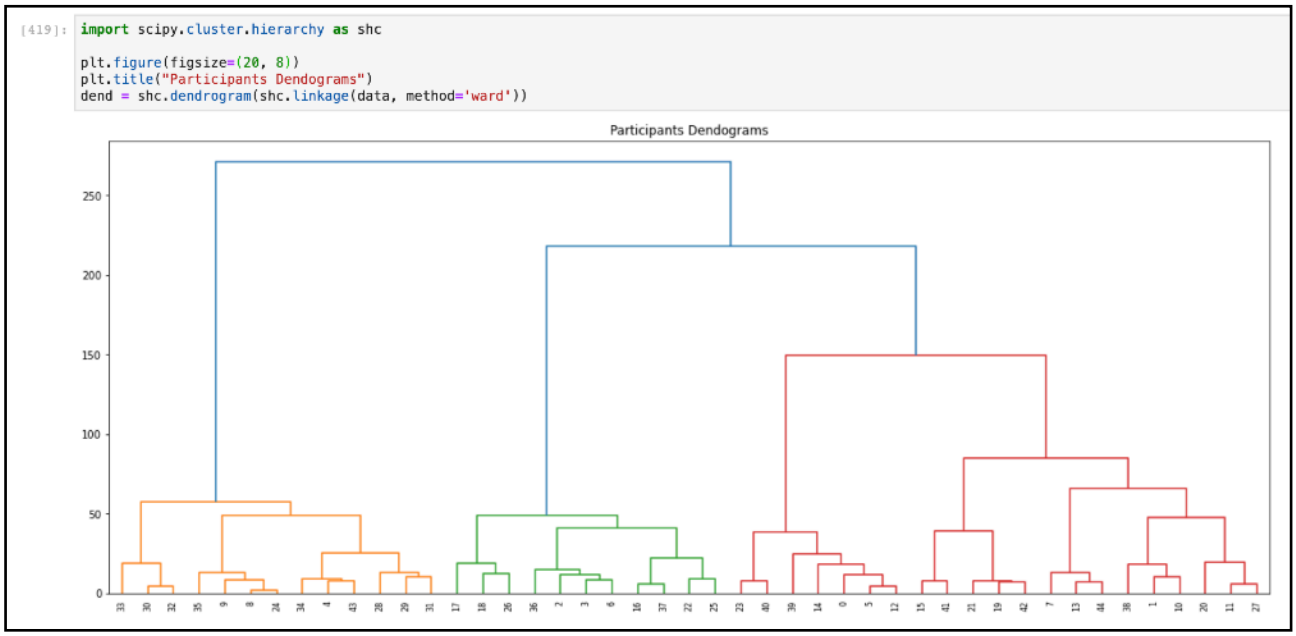
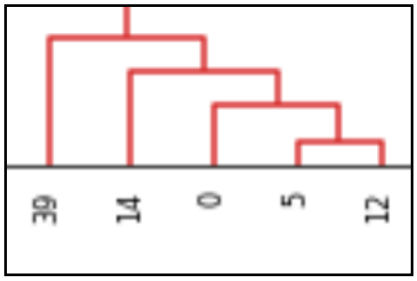


Figure 8. Participants dendrogram



The numbers 39, 14, 0, 5 and 12 can also be considered as one cluster, or 0, 5 and 12; as small as wanted (Figure 8.1) . The position of the entry that is placed on the x-axis is based on the distance between the data points in the plot.

Figure 8.1. a cluster with five people

Now, the column of the cluster will be added to the dataframe and sorted on the value 'cluster'. The clusters are understandable; Marybeth and Elma answered the first question very low and the second one around the 50. For example, if Kian (cluster 0) would completed the questions, she is able to swipe with Neppie and Lyric. They both scored high on the first question and not that high on the second question.

	name	A	B	t	cluster
22	Kian	94	31	1	0
37	Sherrill	98	14	1	0
25	Neppie	94	22	0	0
16	Lyric	96	9	0	0

Figure 9. an example of a cluster

Clustering with 2 variables (K-means)

For the clustering with K-means, the same dataframe is used as for Hierarchical clustering. Some libraries are similar as the libraries before, but also new libraries are imported (Figure.

K-means clustering

With K-means clustering it is possible to group similar data points together and discover underlying patterns. K-means looks for a set number of clusters (K) in a dataset.

The target number k will be defined, which refers to the number of centroids that are needed in the dataset. A centroid is the location that represents the center of the cluster. The centroid of every cluster will be a red dot.

Libraries needed

[3]:
```
import tkinter as tk #This will help creating a root where a file can be opened
from tkinter import filedialog
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

[4]:
```
Data = pd.read_csv("participant_data.csv")
Data.head(10)
```

[4]:

	name	A	B
0	Sonji	81	5
1	Mariann	28	5
2	Regan	30	52
3	Doss	54	87
4	Les	31	10
5	Tera	47	68
6	Mansfield	13	4
7	Eustace	62	59
8	Darin	59	94
9	Aditya	30	25

[5]:
```
df = DataFrame(Data,columns=['A','B'])
```

Figure 10. The same dataframe is imported

Now the centroids will be determined, every centroid stands for the middle of a cluster. Because the amount of clusters is set to 15 (K=15), the x- and y-axis will be printed of 15 centroids (Figure 11). A new column is added with the cluster they are assigned to.

[9]:
```
from pandas import DataFrame
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans #The KMeans algorithm

kmeans = KMeans(n_clusters=15).fit(df)
centroids = kmeans.cluster_centers_
print(centroids) #The X- and Y axis of the centroids
```

```
[[83.33333333 88.        ]
 [35.25        6.5       ]
 [17.         75.75      ]
 [31.5        48.5       ]
 [78.5         5.        ]
 [64.75       54.25      ]
 [44.2        65.        ]
 [ 4.5        51.5       ]
 [28.66666667 19.        ]
 [86.8        69.6       ]
 [13.         6.        ]
 [55.33333333 87.33333333]
 [28.         99.        ]
 [43.         33.        ]
 [59.66666667  9.66666667]]
```

Figure 11. The centroids

[14]:
```
df['cluster'] = kmeans.fit_predict(df[['A', 'B']])
df.head(8)
```

[14]:

	A	B	cluster
0	81	5	12
1	28	5	4
2	30	52	13
3	54	87	8
4	31	10	4
5	47	68	2
6	13	4	9
7	62	59	3

Figure 12. Clusters they have been assigned to

A plot is created where all the participants are plotted and assigned to a cluster, also the centroids (red dots) are plotted which shows the middle of the cluster (Figure 13).

```
[10]: plt.scatter(df['A'], df['B'], c= kmeans.labels_.astype(float), s=50, alpha=0.5) #This plots all the datapoints
      plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=40)
      plt.show()
```

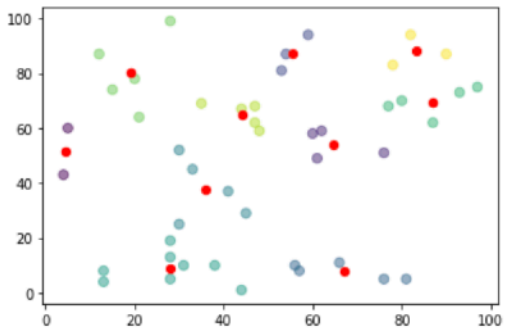


Figure 13. The participants and the clusters they belong to

The ultimate goal of the application would be a 'live' connection between the app and the model, but this is not being realised. What comes near this 'input-output' procedure, is a small tool that is created, also known as a root. If a participant has completed the questions in the app, his data is added to a row of a file where all the participants are in. By doing this manually, this file with participants and their 0 - 100 answer to the statements, is exported in an Excel file. To quickly plot the participants and determine the amount of clusters, there is a root created from where an Excel file can be opened.

```
[6]: import tkinter as tk
from tkinter import filedialog
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

root= tk.Tk()

canvas1 = tk.Canvas(root, width = 400, height = 300,  relief = 'raised')
canvas1.pack()
label1 = tk.Label(root, text='Matched Clustering')
label1.config(font=('avenir', 14, 'bold'))
canvas1.create_window(200, 25, window=label1)

label2 = tk.Label(root, text='Enter the number of clusters:')
label2.config(font=('avenir', 10, 'bold'))
canvas1.create_window(200, 120, window=label2)
entry1 = tk.Entry (root)
canvas1.create_window(200, 140, window=entry1)

def getExcel ():

    global df
    import_file_path = filedialog.askopenfilename()
    read_file = pd.read_excel (import_file_path)
    df = DataFrame(read_file,columns=['x','y'])

browseButtonExcel = tk.Button(text=" Import Excel File ", command=getExcel, bg='green', fg='blue', font=('avenir', 10, 'bold'))
canvas1.create_window(200, 70, window=browseButtonExcel)

def getKMeans ():
    global df
    global numberOfClusters
    numberOfClusters = int(entry1.get())

    kmeans = KMeans(n_clusters=numberOfClusters).fit(df)
    centroids = kmeans.cluster_centers_

    label3 = tk.Label(root, text= centroids)
    canvas1.create_window(200, 250, window=label3)

    figure1 = plt.Figure(figsize=(4,3), dpi=100)
    ax1 = figure1.add_subplot(111)
    ax1.scatter(df['x'], df['y'], c= kmeans.labels_.astype(float), s=50, alpha=0.5)
    ax1.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
    scatter1 = FigureCanvasTkAgg(figure1, root)
    scatter1.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH)

processButton = tk.Button(text=' View Matched Clusters ', command=getKMeans, bg='brown', fg='blue', font=('avenir', 12, 'bold'))
canvas1.create_window(200, 170, window=processButton)

root.mainloop()
```

Figure 14. The code with the settings for the root

By running the cell, a second screen opens and the possibility is there to open an Excel file and fill in the amount of clusters that is wanted.

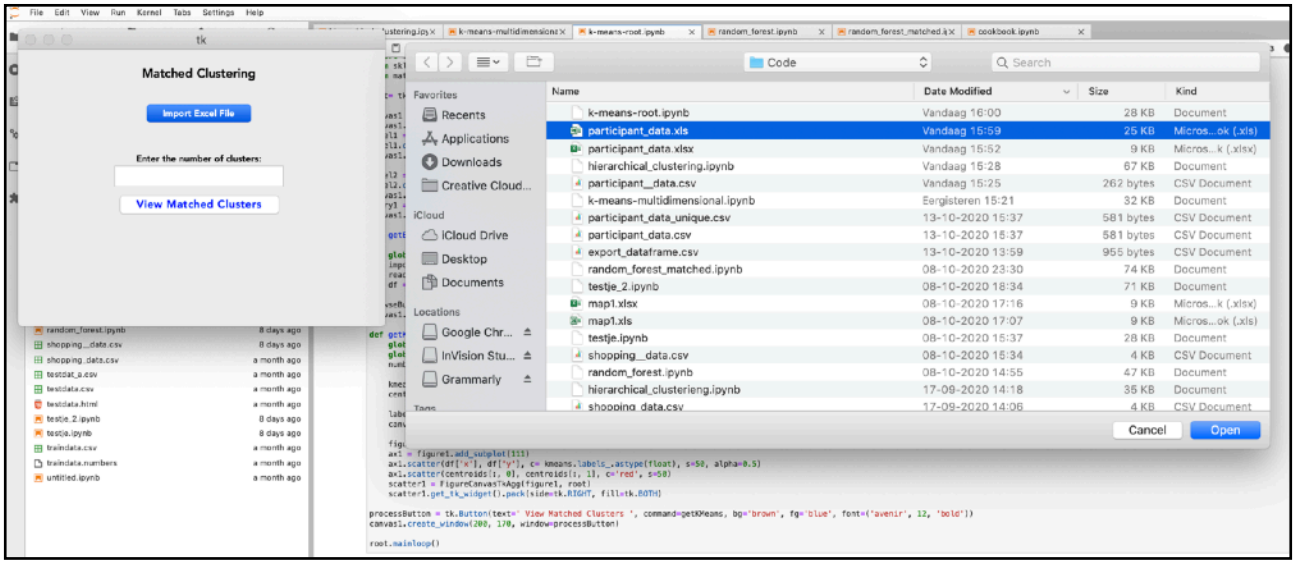


Figure 15. The screen that will open

The centroids will be calculated and plotted in the model together with the data points. If it seems like the amount of the clusters is too less or too many, the amount of clusters can be changed in the same screen and the next graph will be viewed next to the first graph. In this way there can be experimented with the amount of clusters (Figure 16.1 & 16.2).

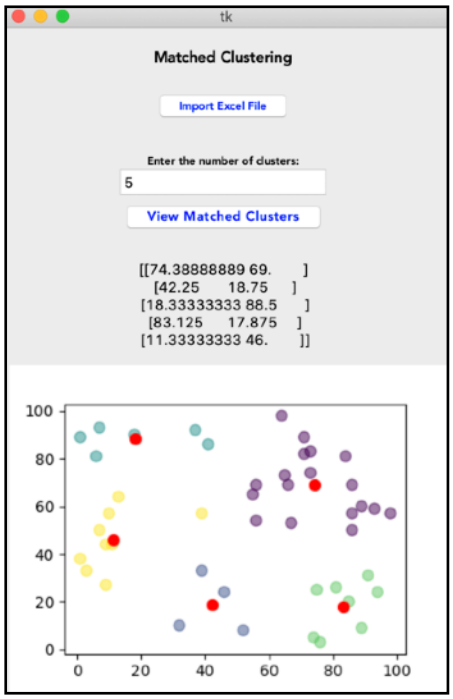


Figure 16.1 plotting with 5 clusters

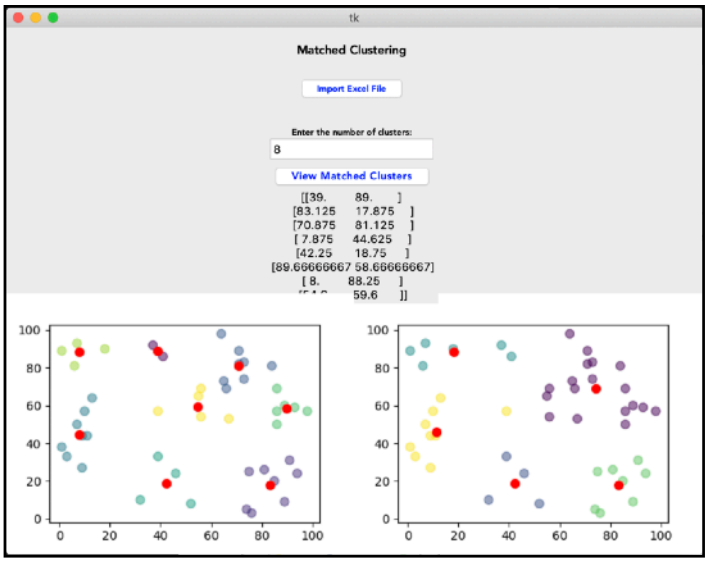


Figure 16.2 comparison with 8 clusters

Because the clustering of the participants between the two algorithms does not show significant differences (the ward-method of Hierarchical Clustering is close to K-means), for the early stage of the application the K-means algorithm will be used, also because the addition of the root is helpful.

Clustering with 10 variables

When the number of participants will grow in the future, the people can be clustered on more variables. In this model, the people have filled in their answer to 10 statements. The same function is used to fill in the rows with the 'number of agreement' that is collected after every answer. For the functioning of this model and the optimization of the app, the use of only 45 participants' data is not enough for finding a fine match between two people. For 2 questions, this was fine but when clustering 45 people with 8 questions there are too many differences between the matches. That is why the amount of participants will significantly increase and now will be 750. For this clustering proces, there will not be a number saved between 1 and 101 but between 1 and 11. This decision is made because in this proces the participants will not actually be plotted in a graph, so the specific numbers of 1 to 100 to have on the x- and y-axis are not needed anymore.

To split up the dataset into the Dutch people and the refugees, a 1 and a 0 is assigned to the row. 1 will stand for the Dutch people, 0 for refugee.

[335]:
```
df1 = pd.DataFrame(np.random.randint(1,11,size=(750, 10)), columns=list('ABCDEFGHIJ')) #Every data entry will have a number between 1 and 11
target_group = pd.DataFrame(np.random.randint(0,2,size=(750, 1)), columns=list('t')) #Target group
df2 = pd.concat([df1, target_group], axis=1)
df2.head(5) #Shows the first 5
```

[335]:

	A	B	C	D	E	F	G	H	I	J	t
0	9	3	8	5	9	7	6	3	10	1	0
1	10	8	1	5	2	1	1	10	5	8	1
2	9	7	8	9	8	8	3	10	7	4	1
3	5	9	4	2	10	10	9	8	4	10	1
4	10	6	10	7	9	1	6	3	10	7	0

Next up, the data entries will be provided with a name. This is done because it helps recognizing the rows and brings it closer to the case.

[336]:
```
def random_names(name_type, size): #A function is being created and later will be recalled
    names = getattr(Provider, name_type)
    return np.random.choice(names, size=size) #size will be provided later on
```

[348]:
```
size = 750 #750 names will be generated.
df3 = pd.DataFrame(columns=['name']) #A new dataframe is created and later be merged
df3 ['name'] = random_names('first_names', size) #This fills in all the cells with a generated name
df = pd.concat([df3, df2], axis=1)
df.head(5)
```

[348]:

	name	A	B	C	D	E	F	G	H	I	J	t
0	Izayah	9	3	8	5	9	7	6	3	10	1	0
1	Clytie	10	8	1	5	2	1	1	10	5	8	1
2	Dionicio	9	7	8	9	8	8	3	10	7	4	1
3	Pascal	5	9	4	2	10	10	9	8	4	10	1
4	Wiley	10	6	10	7	9	1	6	3	10	7	0

Figure 17. The dataframe is created with the names, answers and the target group

The dataset has 750 participants now, they will be clustered in 125 clusters, to aim for around 6 people in one cluster and avoid 1 people being alone in one cluster. The amount of 6 is per cluster is for now realistic because the data is generated random and with this normally divided. The array is printed with the clusters that belong to the participants.

Predicting the cluster

Now the the class is imported for clustering.

```
[361]: import matplotlib.pyplot as plt #To plot the people
       df.shape #It has 751 rows and 3 columns

[361]: (750, 12)

[362]: data = df.iloc[:, 1:11].values #Names and target group will be excluded for now
```

The affinity is set to euclidean, this means it will look at the distance between the datapoints; the length of the line between two data points. Ward is a linkage method that minimizes the variant between the clusters and is close (by it properties and efficiency) to K-means clustering; they share the same objective function.

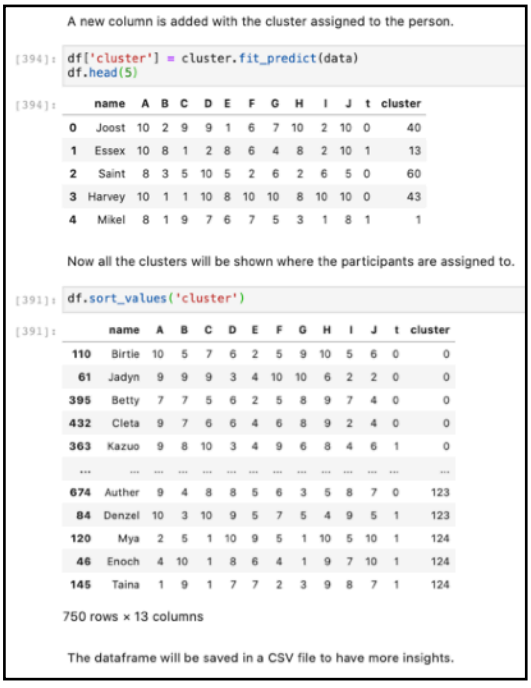
```
[364]: from sklearn.cluster import AgglomerativeClustering #The function that will cluster the data entries

       cluster = AgglomerativeClustering(n_clusters=125, affinity='euclidean', linkage='ward')
       cluster.fit_predict(data)

[364]: array([ 17, 119,  30,  94,  39, 105,   0,  13, 105,  35,  73,  12,  49,
               75, 110,  92,   1,  49,  96,   3,  56,   2,  46, 112,   6,  53,
               80,  40,  48,  58,  62,  60,  44,  47,  67,  27,  14,  68,   7,
                1,  65,  59,  32,  26,  10,  73,  31,  87,  11,  94,  23,  34,
               85, 100,  22,  55,  25,  82,  18,  65, 124,  33,  25, 103,  37,
               26, 111,  70,  38, 102,  65,  57,  26, 120,  98,  49,  62,  79,
               19,  45,  43,  79,   8,  32,   6,  79,  20,  49,   7,  97,   8,
                0,  45,  22, 105,  30,  70,  71,  23, 124,   7,  90,  55,   7,
              113,   6,  66,  16,  24,  79,   1,  78,  67,  49,  42, 124,  80,
               95,  29,  24,  83,  20,   4,  11,  30,  66,  51,  42,  68, 106,
```

Figure 18. The clusters that belong to the participants

The clusters are now added in a new column, and to see who is in which cluster, the dataframe will be sorted by cluster. The dataframe will be exported to a CSV file to gain more insights and scroll through the people that have been matched with each other (Figure 19). Cluster 31 is an example of seven people that have been matched (4 refugees and 3 native Dutch people). As you can see in the answers, the participants are all close to each other in their answering. In this case, Delton would be able to swipe with Clarnce, Wilfrid, Reina and Annalise.



A new column is added with the cluster assigned to the person.

```
[394]: df['cluster'] = cluster.fit_predict(data)
       df.head(5)
```

[394]:

	name	A	B	C	D	E	F	G	H	I	J	t	cluster
0	Joost	10	2	9	9	1	6	7	10	2	10	0	40
1	Essex	10	8	1	2	8	6	4	8	2	10	1	13
2	Saint	8	3	5	10	5	2	6	2	6	5	0	60
3	Harvey	10	1	1	10	8	10	10	8	10	10	0	43
4	Mikel	8	1	9	7	6	7	5	3	1	8	1	1

Now all the clusters will be shown where the participants are assigned to.

```
[391]: df.sort_values('cluster')
```

[391]:

	name	A	B	C	D	E	F	G	H	I	J	t	cluster
110	Birtie	10	5	7	6	2	5	9	10	5	6	0	0
61	Jadyn	9	9	9	3	4	10	10	6	2	2	0	0
395	Betty	7	7	5	6	2	5	8	9	7	4	0	0
432	Cleta	9	7	6	6	4	6	8	9	2	4	0	0
363	Kazuo	9	8	10	3	4	9	6	8	4	6	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
674	Auther	9	4	8	8	5	6	3	5	8	7	0	123
84	Denzel	10	3	10	9	5	7	5	4	9	5	1	123
120	Mya	2	5	1	10	9	5	1	10	5	10	1	124
46	Enoch	4	10	1	8	6	4	1	9	7	10	1	124
145	Taina	1	9	1	7	7	2	3	9	8	7	1	124

750 rows × 13 columns

The dataframe will be saved in a CSV file to have more insights.

Figure 19. The result

Delton	8	3	6	3	8	1	10	4	5	4	1	31
Evelyn	7	3	3	3	3	1	6	3	4	6	1	31
Kalen	9	5	5	1	4	5	9	1	9	4	1	31
Clarnce	10	3	5	1	5	2	9	4	6	4	0	31
Wilfrid	8	2	6	1	3	1	9	1	5	5	0	31
Reina	8	2	4	1	7	1	7	1	7	5	0	31
Annalise	10	3	2	1	2	3	9	1	7	2	0	31

Figure 20. Cluster 31

Next to the matching of generated participants, it is also possible to add a new (real) participant to the dataframe and thereafter cluster with the people that already are in the participants. In this functionality it is possible to match a new participant, with real data with the participants that are in the dataset with simulative data. The name and the answers can be inserted and the new participant will be taken in the clustering process.

```
[31]: new_participant = pd.DataFrame({'name':'Joost',
                                      'A': 10,
                                      'B': 2,
                                      'C': 9,
                                      'D': 9,
                                      'E': 6,
                                      'F': 7,
                                      'G': 10,
                                      'H': 2,
                                      'I': 10,
                                      'J': 9,
                                      't': 1 }, #New dataframe with one participant
                                                                    index =[0])
      df = pd.concat([new_participant, df]).reset_index(drop = True) #Merged with the original
      df.head(3)
```

[31]:

	name	A	B	C	D	E	F	G	H	I	J	t
0	Joost	10	2	9	9	6	7	10	2	10	9	1
1	Katlyn	1	8	10	9	1	9	3	3	3	2	0
2	Leatrice	9	8	9	5	6	5	7	5	5	1	0

Figure 21. The addition of a new participant

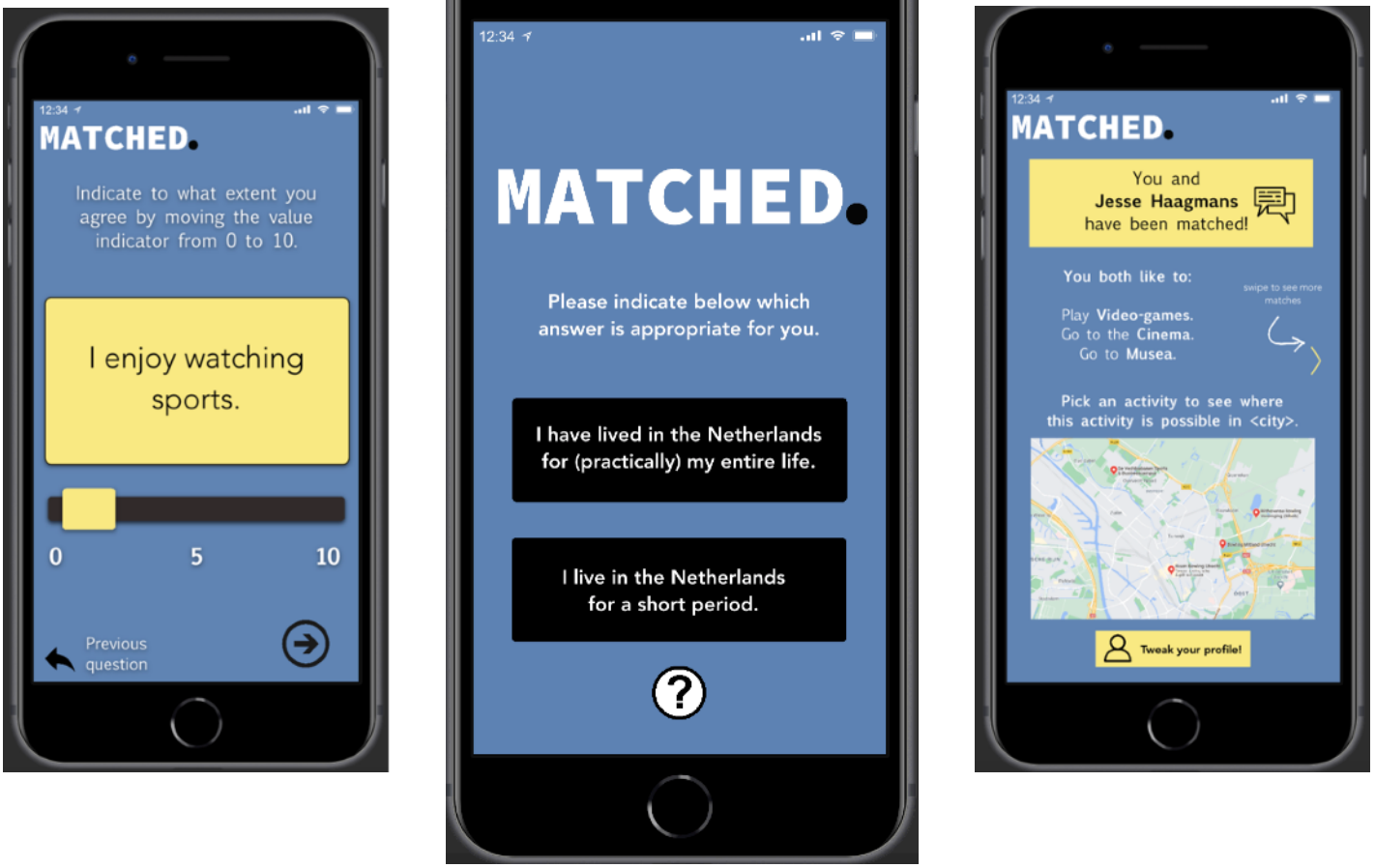
The new participant is clustered in cluster 34 and he (as a 1) is able to swipe through all participants with a 0 (refugees) (Figure 22). So this would be Takisha, Vicky, Kasie, Nikita and Ashlyn. Brandy, Sheridan, Lettie, Adonis and Hilliard are matched with the same refugees.

Joost	10	2	9	9	6	7	9	2	10	9	1	34
Takisha	10	2	10	3	4	7	3	9	9	1	0	34
Vicky	9	2	9	7	3	10	9	2	6	5	0	34
Adonis	10	1	10	5	10	4	10	4	7	5	1	34
Kasie	9	1	10	1	4	9	7	8	7	7	0	34
Hilliard	10	1	9	9	4	8	1	9	3	5	1	34
Nikita	9	2	9	8	2	1	7	7	6	1	0	34
Ashlyn	10	2	9	2	9	4	10	9	5	4	0	34
Brandy	9	2	10	10	7	7	3	6	5	8	1	34
Sheridan	9	1	9	5	2	7	4	1	2	9	1	34
Lettie	9	1	10	4	5	7	6	7	6	9	1	34

Figure 22. A new participant and his cluster

Screen design of the prototype

The clickable prototype has been made with Invision Studio. The focus of the research had more of a focus on what should be the content of the app, and finding a suitable algorithm, than a focus on user experience of the app. This means that not all design considerations, such as use of color, navigation, fonts and other aspects of user experience have been tested with the target group. In this chapter you will read more about the process that led to the final prototype of the app Matched.



Design iterations

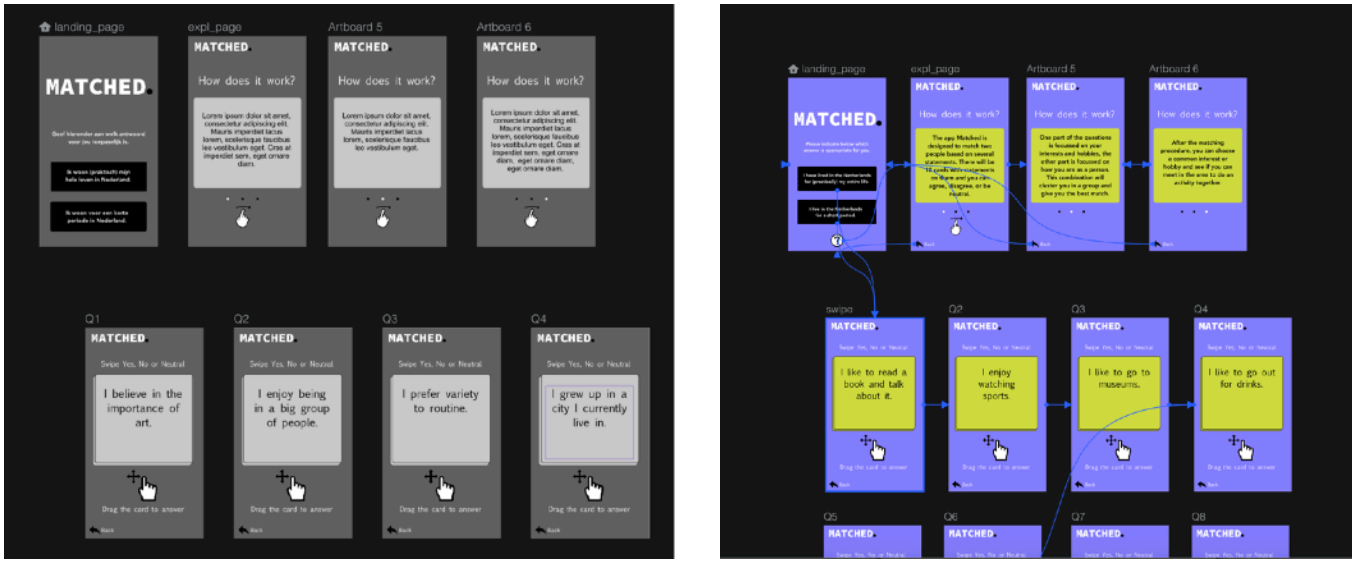


Figure 23. Creation of the wireframes and color decisions

The designing of the screens started with creating the wireframes and the components. At this moment, swiping the cards was still the way how the statements would be rated. A color combination of yellow/light purple was used (Figure 23). After this resulted in a low contrast ratio (Figure 24) with white text, this color combination was adjusted to colors with a higher contrast ratio (HexNaw, n.d.)

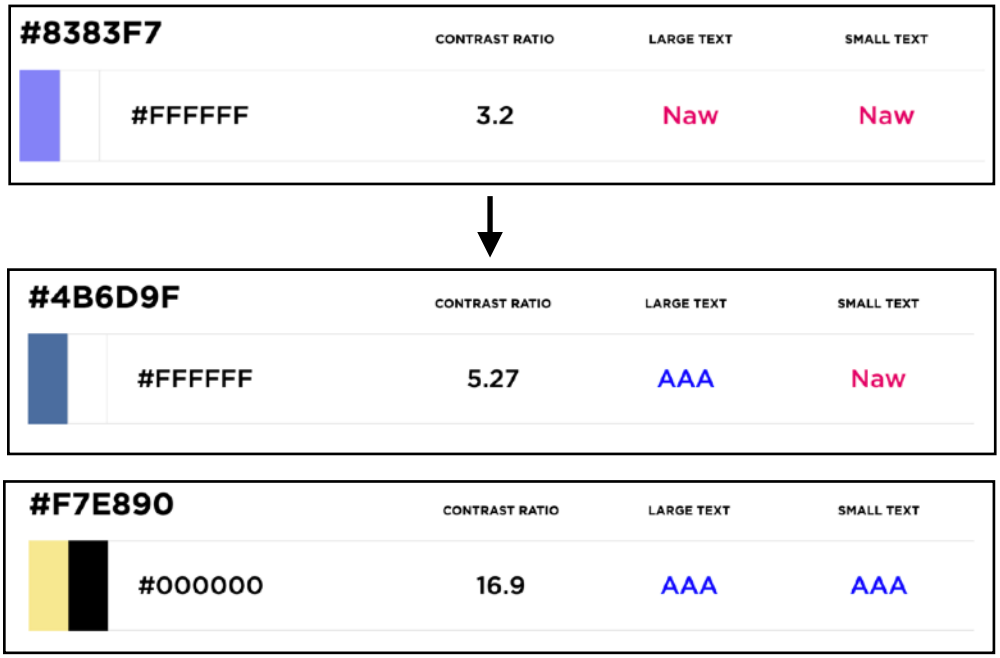
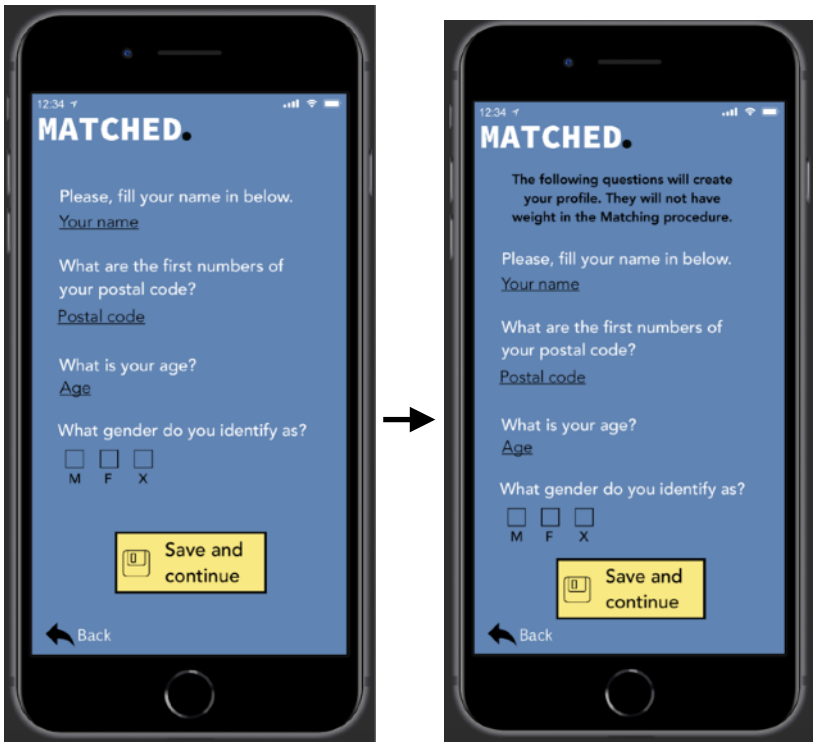


Figure 24. Improvement of contrast ratio



Demographical questions

In the first iteration of the form where demographical questions are being asked, the questions were supposed to have weight in the matching procedure. After doing research it can be concluded that little people thought this mattered for making friends. The questions do not disappear, but a message is shown that it will not but this number is not taken with in the clustering process (Figure 25).

Figure 25. Changes after the research has been done

Cards or moving bar

The decision to not choose for swiping cards but for the moving bar has been made because it delivers a more precise indication in what extent the participants agrees with the statements, which helps clustering more precise (Figure 26).

Figure 26. From swiping cards to a value indicator on a bar

Final prototype

As described earlier, the prototype has a front- and a backside. The clickable prototype can be visited by a link and the code will be uploaded on Github. In the prototype it is possible to drag the value indicator to the right, but due to the limitations of Invision it is not possible to leave it on the location you want to leave it on.

A clickable version of the app is available on:
https://projects.invisionapp.com/prototype/ckg6fxfjr004vq401pw47f2no/play

In the code there can be experimented with adding new users, in the fifth cell, the name of the participant, numbers of agreement, and target group (t) can be filled in.

name: ‘’
A: I like to play video games. (0-10)
B: I like to go on a bicycle trip. (0-10)
C: I like to go out for drinks. (0-10)
D: I enjoy watching sports. (0-10)
E: I like to go for a walk. (0-10)
F: I enjoy going to the cinema. (0-10)
G: I like to listen to music. (0-10)
H: I try to always look at the bright side of life. (0-10)
I: I always keep my promises. (0-10)
J: I am determined to be successful at what I do. (0-10)
t: 0 / 1 (0 = Refugee, 1 = Native Dutch person)

If all cells are runned[1], a CSV file will be saved in the same map as the *.ipynb* file, you can sort by cluster, look up the name of the new participant and see the people he or she have been matched with. Also the code of the earlier phase (45 parcan be found on Github. For the earlier phase (less participants, less questions), the K-means model with the Tkinter can be used.

The code can be found on Github:
https://github.com/vdmeulenjoost/Matched

[1] The following are not installed standard and need to be installed to run the code: pip install Faker - pip install names

References

Amelia, A. (2018, September 27). K-Means Clustering: From A to Z. Retrieved from https://towardsdatascience.com/k-means-clustering-from-a-to-z-f6242a314e9a

Datanovia. (2018, October 20). Agglomerative Hierarchical Clustering. Retrieved from https://www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/

Hex Naw | A color accessibility tool built by The Scenery. (n.d.). Hex Naw. https://hexnaw.com/

Seif, G. (2019, September 14). The 5 Clustering Algorithms Data Scientists Need to Know. Towards Data Science. https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68\

Faker's documentation — Faker 4.14.0 documentation. (n.d.). https://faker.readthedocs.io/en/master/. https://faker.readthedocs.io/en/master/