In this project I am planning to build up an interactive facial feature analysis platform as an extension on the previous project. The main tools would be used are Shiny packages in R for UI design and the Microsoft Oxford API for connection with Microsoft facial detection and recognition system.

The basic idea is to first scrape through the image database websites to download existing facial images of different genders, ages and races , send them to the Microsoft system for further analysis and get back all the generated features to form the final local facial features database as multiple large RDS files. And then, after the initiation of the whole database, when the user begin exploring the Shiny App, he or she can upload any picture from local system to the app. The app, in turn, send the chosen picture to the Microsoft Oxford for this particular analysis. Finally, when the app get back the analytic data, it will automatically conduct deep descriptive analysis based on the whole distribution of the facial features and that of the uploaded one.

1.Initiate the local database

Since we don't have an existing database with the right features recorded, we need to generate one first. In order to do that, we first need to scrape on some websites which contain some pretty long lists of images with well structured framework. I choose the IMDB site for reference: http://www.imdb.com/. This website contains many lists of famous celebrities, which is very convenient for us to scrape through and download for further modification. I finally chose the list of 1000 celebrities(http://www.imdb.com/list/ls058011111/) as the data source.

The screenshot of the website

Using the inspect function of Chrome or other browsers, we can have a clear view of the HTML structure of the image list. The very information we want is the exact url links of every image:

```
▼<div id="content-2-wide" class="redesign">
  ▼<div id="main">
    ▼<div class="article listo list_people" data-list-class="LIST"
    data-list-type="People" data-list-id="ls058011111" data-items-
    per-page="100" data-author-id="ur39602325">
      ▶<div class="disable-pane">…</div>
      ▶<div class="rightcornerlink">…</div>
       <a name="page_top"></a>
       <h1 class="header">Top 1000 Actors and Actresses</h1>
      ▶<div class="byline">…</div>
       <div class="clear"></div>
      ▶<div class="header">…</div>
      ▼<div class="list detail">
        ▼<div class="list_item odd">
          ▼<div class="image">
            ▼<a href="/name/nm0000134/">
              ▶<div class="hover-over-image zero-z-index" data-const=
               "nm0000134">…</div>
               </a>
             </div>
             <div class="number">1.</div>
            ▶<div class="info">…</div>
             <div class="clear"> </div>
           </div>
        ▶<div class="list_item even">…</div>
```

the inner structure of the image elements

Next step, we can continue to write codes to get all the links from the website. We need library "rvest" for scraping. And since there are totally 10 pages with 100 persons per page. A small iteration loop is needed to let the program go over the ten pages one by one:

```
library(rvest)
linksactresses = 'http://www.imdb.com/list/ls058011111/'
out = read_html(linksactresses)
images = html_nodes(out, '.zero-z-index')
imglinks = html_nodes(out, xpath = "//img[@class='zero-z-index']/@src") %>% html_text()
for (i in c(1:9))
{
  infix=paste('?start=',i,'01&view=detail&sort=listorian:asc',sep="")
  linksactresses=paste('http://www.imdb.com/list/ls058011111/',infix,sep="")
  out = read_html(linksactresses)
  images = html_nodes(out, '.zero-z-index')
  templink=html_nodes(out, xpath = "//img[@class='zero-z-index']/@src") %>% html_text()
  imglinks=c(imglinks,templink)
}
```

After retrieving, we can print out the imglinks variable to see the link list we just got from the internet:

```
 [1] "http://ia.media-imdb.com/images/M/MV5BMjAwNDU3MzcyOV5BMl5BanBnXkFtZTcwMjc0MTIxMw@@._V1._SY209_CR9,0,140,209_.jpg"
 [2] "http://ia.media-imdb.com/images/M/MV5BMTQ3OTY0ODk0M15BMl5BanBnXkFtZTYwNzE4Njc4._V1._SY209_CR5,0,140,209_.jpg"
 [3] "http://ia.media-imdb.com/images/M/MV5BMTQ2MjMwNDA3Nl5BMl5BanBnXkFtZTcwMTA2NDY3NQ@@._V1._SY209_CR2,0,140,209_.jpg"
 [4] "http://ia.media-imdb.com/images/M/MV5BMTg3MDYyMDE5OF5BMl5BanBnXkFtZTcwNjgyNTEzNA@@._V1._SY209_CR65,0,140,209_.jpg"
 [5] "http://ia.media-imdb.com/images/M/MV5BMjI0MTg3MzI0M15BMl5BanBnXkFtZTcwMzQyODU2Mw@@._V1._SY209_CR7,0,140,209_.jpg"
 [6] "http://ia.media-imdb.com/images/M/MV5BMTIyOTE3MDM5Ml5BMl5BanBnXkFtZTYwMzA2MTM2._V1._SY209_CR10,0,140,209_.jpg"
 [7] "http://ia.media-imdb.com/images/M/MV5BMTM0ODU5Nzk2OV5BMl5BanBnXkFtZTcwMzI2ODgyNQ@@._V1._SY209_CR3,0,140,209_.jpg"
 [8] "http://ia.media-imdb.com/images/M/MV5BMTQzMzg1ODAyNl5BMl5BanBnXkFtZTYwMjAxODQ1._V1._SX140_CR0,0,140,209_.jpg"
 [9] "http://ia.media-imdb.com/images/M/MV5BMjE5NDU2Mzc3MV5BMl5BanBnXkFtZTcwNjAwNTE5OQ@@._V1._SY209_CR8,0,140,209_.jpg"
[10] "http://ia.media-imdb.com/images/M/MV5BMTkwNjYwNDE5M15BMl5BanBnXkFtZTYwNzg0MDQ2._V1._SY209_CR13,0,140,209_.jpg"
[11] "http://ia.media-imdb.com/images/M/MV5BMjA1MjE2MTQ2MV5BMl5BanBnXkFtZTcwMjE5MDY0Nw@@._V1._SX140_CR0,0,140,209_.jpg"
[12] "http://ia.media-imdb.com/images/M/MV5BMjE2NDY2NDc1Ml5BMl5BanBnXkFtZTcwNjAyMjkwOQ@@._V1._SY209_CR9,0,140,209_.jpg"
[13] "http://ia.media-imdb.com/images/M/MV5BMTk1MjM3NTU5M15BMl5BanBnXkFtZTcwMTMyMjAyMg@@._V1._SY209_CR9,0,140,209_.jpg"
[14] "http://ia.media-imdb.com/images/M/MV5BNzYyODM4NDU1MV5BMl5BanBnXkFtZTYwMjI1ODM2._V1._SY209_CR11,0,140,209_.jpg"
[15] "http://ia.media-imdb.com/images/M/MV5BMTc3NzU0ODczMF5BMl5BanBnXkFtZTcwODEyMDY5Mg@@._V1._SY209_CR8,0,140,209_.jpg"
[16] "http://ia.media-imdb.com/images/M/MV5BMjE1NDY5MjM5Ml5BMl5BanBnXkFtZTYwNTU1OTQ2._V1._SY209_CR10,0,140,209_.jpg"
[17] "http://ia.media-imdb.com/images/M/MV5BMTc1NjMzMjY3NF5BMl5BanBnXkFtZTcwMzkxNjQzMg@@._V1._SY209_CR1,0,140,209_.jpg"
[18] "http://ia.media-imdb.com/images/M/MV5BMTkxMzk4MjQ4MF5BMl5BanBnXkFtZTcwMzExODQxOA@@._V1._SX140_CR0,0,140,209_.jpg"
[19] "http://ia.media-imdb.com/images/M/MV5BMjA0ODc0NTE2NF5BMl5BanBnXkFtZTYwNjYyMjQ2._V1._SX140_CR0,0,140,209_.jpg"
```

After we restore all the urls into one vector, we could continue to send every element inside a vector into the Microsoft Oxford API. We should iterate the vector from beginning to the end to send every web link directly to the APIs. Here the two important APIs are functions getFaceResponseURL and getEmotionResponseURL, which are in charge of sending picture links to corresponding interfaces and get back responses. The variables facekey and emotionkey are the accessing keys used for successfully getting connected. They can both be got by simply registering on the website. The two functions can get back the facial features and facial emotion analytic results.

```
for (i in imglinks)
{
  if (i %in% finalframe[['link']])
  {
    next
  }
  tempface=getFaceResponseURL(i, facekey)
  if (length(tempface)!=14)
  {
    next
  }
  tempemotion=getEmotionResponseURL(i, emotionkey)
  if (length(tempemotion)!=12)
  {
    next
  }
  tempemotion=data.frame(tempemotion)
  tempface=data.frame(tempface)
  tempface['link']=i
  tempemotion=tempemotion[grep('scores',names(tempemotion))]
  newframe=cbind(tempface,tempemotion)
  finalframe=rbind(finalframe,newframe)
}
```

One thing worth mentioning is that since the accessing key is provided for a free account, there is a limitation of how many requests can be sent every minute. Microsoft currently limit the number to 20 and the rest requests that exceeds 20 would be just ignored. So I actually tried to run this loop three times to get as many responses as possible. The structure and content of the final result is as follows:

```
> names(finalframe)
 [1] "faceId"                          "faceRectangle.top"               "faceRectangle.left"              "faceRectangle.width"
 [5] "faceRectangle.height"            "faceAttributes.smile"            "faceAttributes.headPose.pitch"   "faceAttributes.headPose.roll"
 [9] "faceAttributes.headPose.yaw"     "faceAttributes.gender"           "faceAttributes.age"              "faceAttributes.facialHair.moustache"
[13] "faceAttributes.facialHair.beard" "faceAttributes.facialHair.sideburns" "link"                        "scores.anger"
[17] "scores.contempt"                 "scores.disgust"                  "scores.fear"                     "scores.happiness"
[21] "scores.neutral"                  "scores.sadness"                  "scores.surprise"
```

the structure of the final result frame

```
> head(finalframe)
                                  faceId faceRectangle.top faceRectangle.left faceRectangle.width faceRectangle.height faceAttributes.smile
1 c85063be-87b2-4b00-9732-57a3227a721a               54                38               74                74               0.862
2 34c1f037-a89b-474a-92e2-b064dcbd208a               46                39               65                65                   1
3 099aaaf5-cd8b-4a96-9021-d5a7ac50eb97               53                53               67                67                   1
4 e5f75a12-94f6-4aac-ba7c-632ad633e6b3               73                22               92                92                   0
5 1261cc9f-e87e-44e6-af61-7b6e237815e5               58                15               76                76               0.157
6 b195e610-40cb-49a6-91f7-07aaee962c2f               48                27               93                93               0.018
  faceAttributes.headPose.pitch faceAttributes.headPose.roll faceAttributes.headPose.yaw faceAttributes.gender faceAttributes.age
1                             0                          2.7                          6.8                 male              52.5
2                             0                          1.6                          6.3                 male              82.5
```

the head content of the final result frame

At last, I stored this large data frame into local directory as an RDS file in case of loss. However, this is not the end of the initialization. For another function of face matching, we need more works to be done before shifting our focus onto the implementation of Shiny. If we go back to look at the structure of the final frame, we would find there is a feature called "faceId" at the first column. This is a unique ID generated automatically as we sent the pictures to APIs. But they can just last for 24 hours. For a permanent ID for each facial picture, we would need to build a face list where a permanent ID is stored for each picture. To do this, we first send the request of initiating several new face lists:

```
library(dplyr)
facelist_male = "https://api.projectoxford.ai/face/v1.0/facelists/list_gender_male"
facelist_female= "https://api.projectoxford.ai/face/v1.0/facelists/list_gender_female"
facelist_hair_y="https://api.projectoxford.ai/face/v1.0/facelists/list_hair_y"
facelist_hair_n="https://api.projectoxford.ai/face/v1.0/facelists/list_hair_n"
facelist_age_2="https://api.projectoxford.ai/face/v1.0/facelists/list_age_2"
facelist_age_3="https://api.projectoxford.ai/face/v1.0/facelists/list_age_3"
facelist_age_4="https://api.projectoxford.ai/face/v1.0/facelists/list_age_4"
facelist_list=c(facelist_male,facelist_female,facelist_hair_y,facelist_hair_n,facelist_age_2,facelist_age_3,facelist_age_4)
for (i in facelist_list)
{
  mybody=list(name=i)
  faceLIST = PUT(
    url = i,
    content_type('application/json'), add_headers(.headers = c('Ocp-Apim-Subscription-Key' = facekey)),
    body = mybody,
    encode = 'json'
  )
  print(content(faceLIST))
}
```

And again we need to send all pictures this API to add them into corresponding face list according to the groups they in:

```
facelist_image_male=unique(select(filter(finalframe,faceAttributes.gender=='male'),link))[[1]]
facelist_image_female=unique(select(filter(finalframe,faceAttributes.gender=='female'),link))[[1]]
facelist_image_hair_y=unique(select(filter(finalframe,faceAttributes.facialHair.beard==0),link))[[1]]
facelist_image_hair_n=unique(select(filter(finalframe,faceAttributes.facialHair.beard!=0),link))[[1]]
finalframe['faceAttributes.age']=as.numeric(as.vector(finalframe[['faceAttributes.age']]))
facelist_age_2=unique(select(filter(finalframe,(faceAttributes.age>=20) & (faceAttributes.age<30)),link))[[1]]
facelist_age_3=unique(select(filter(finalframe,(faceAttributes.age>=30) & (faceAttributes.age<40)),link))[[1]]
facelist_age_4=unique(select(filter(finalframe,(faceAttributes.age>=40)),link))[[1]]
```

```
image_list=list(facelist_image_male,facelist_image_female,facelist_image_hair_y,facelist_image_hair_n,facelist_age_2,facelist_age_3,facelist_age_4)
for (i in c(1:length(facelist_list)))
{
  URL.face = facelist_list[i]
  userdata=0
  for (j in image_list[[i]])
  {
    face.uri = paste(
      URL.face,'/persistedFaces?userData=',
      userdata,
      sep = ""
    )
    face.uri = URLencode(face.uri)
    mybody = list(url = j )
    faceLISTadd = POST(
      url = face.uri,
      content_type('application/json'), add_headers(.headers = c('Ocp-Apim-Subscription-Key' = facekey)),
      body = mybody,
      encode = 'json'
    )
    userdata=userdata+1
  }
}
```

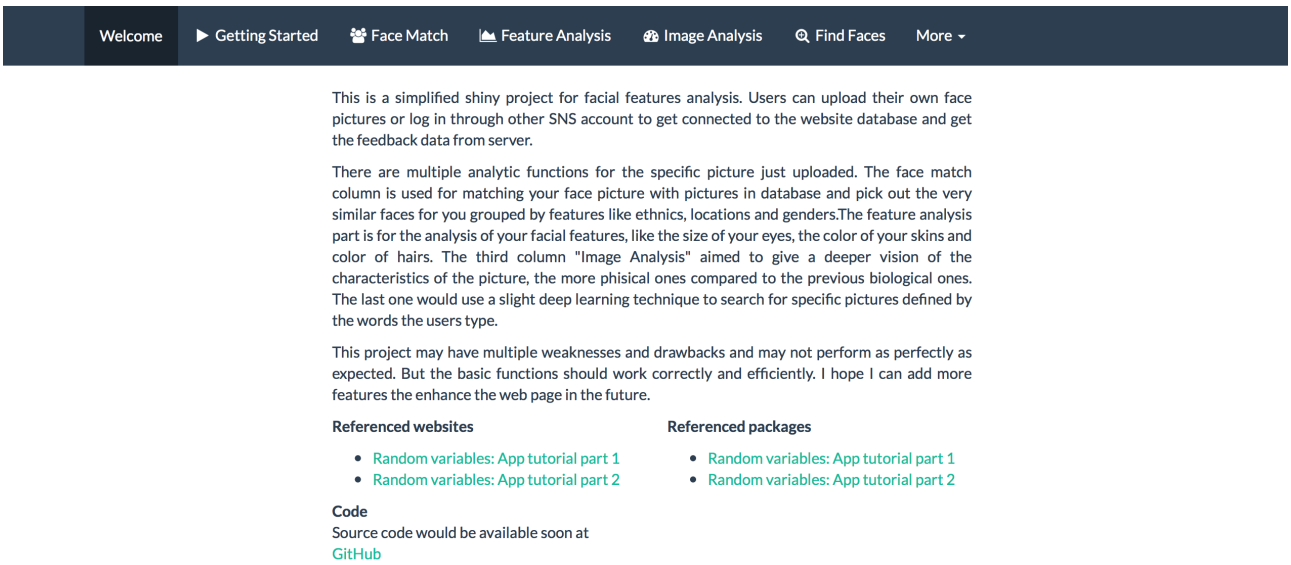
This is even a much longer procedure which takes nearly half an hour. In case to keep all the results, another RDS file is needed as "dataframe_list.rds" in local path.

2. Building The Shiny Interactive APP

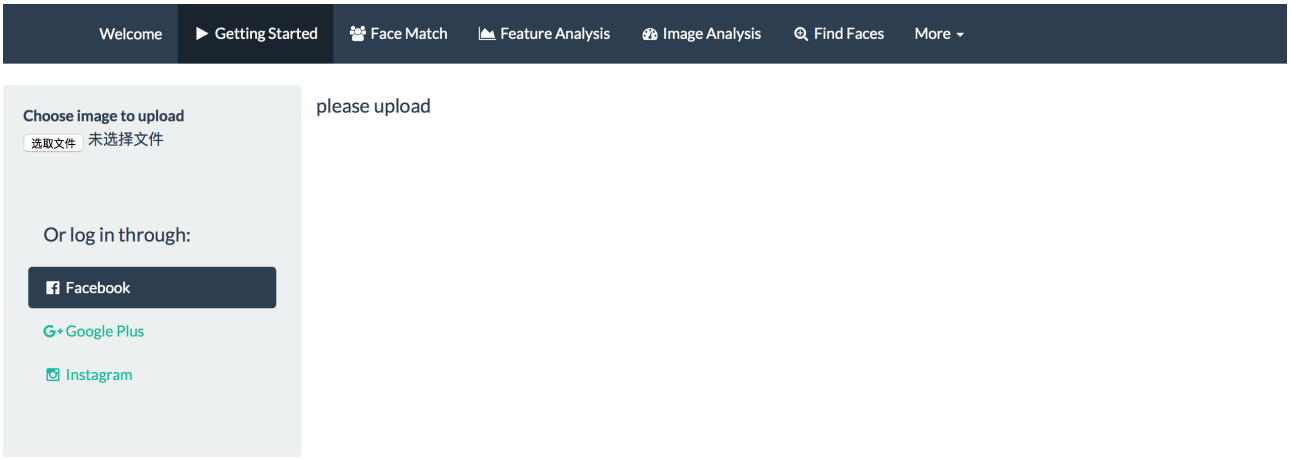
After the works of all preparations, we can move on to build our interactive UI, Shiny. Different from that of the first section, in this section I would not go into details of the designs of the whole shiny framework, which contains hundreds of lines of codes in multiple script files as ui.r, server.r and global.r. Instead, I would focus on the final visual result of the Shiny. All my source codes can be found inside my Github directory:

Before browsing the web page, there are a few things need to be made clear in advance due to some minor drawbacks of the interactive system.

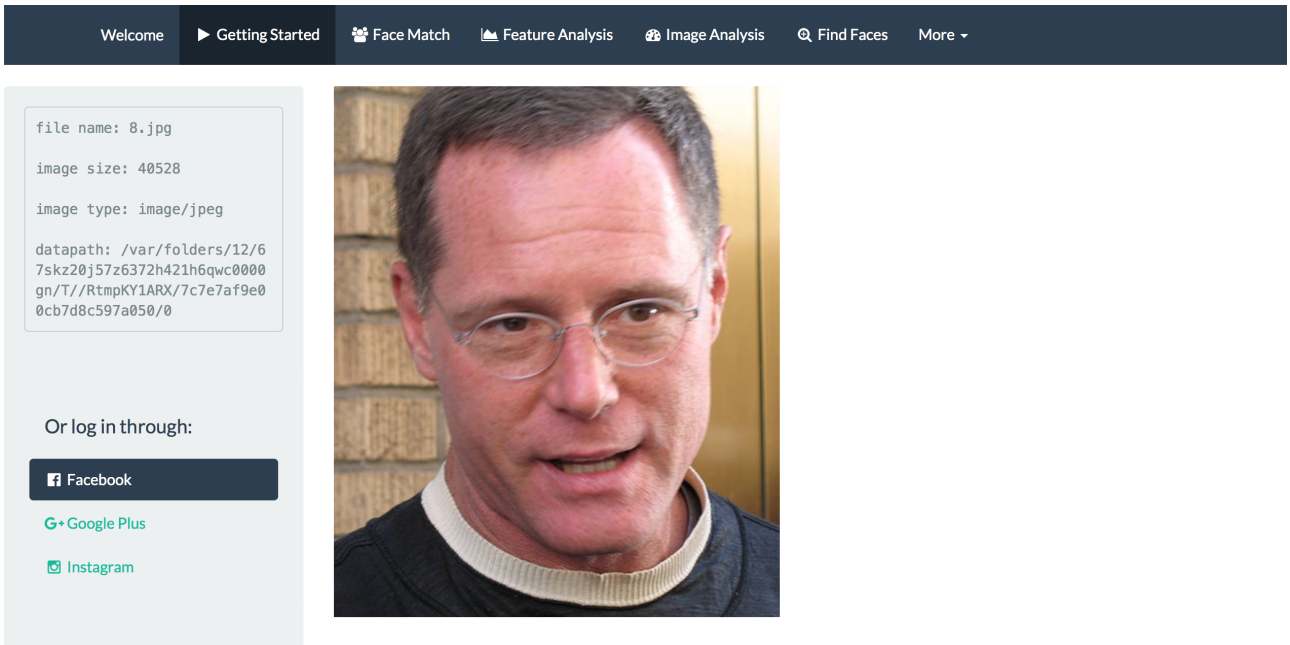
The first page comes in our sight is the introduction page, we can see a navigation bar on the top while a brief introduction passage is located in the middle of the page:



At the second page, when we click the "Getting Started" button, we will see a sidebar with an upload function asking us to upload a local picture:

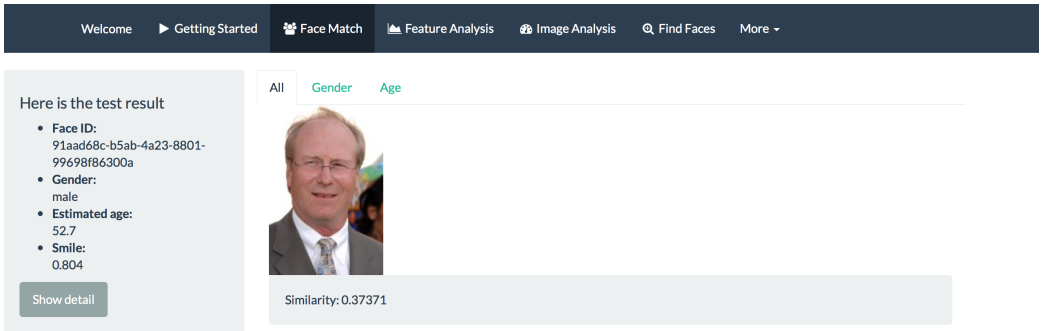


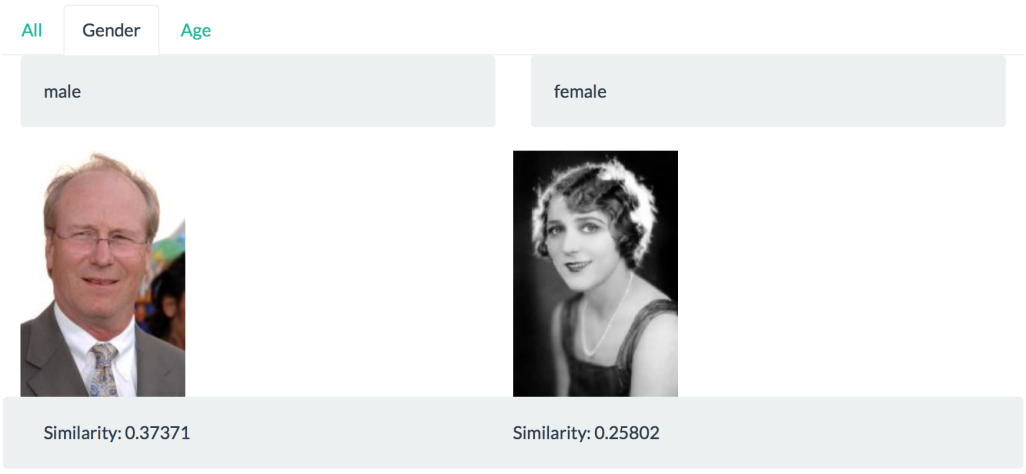
The default label language turns into Chinese in my computer browser. But it should be English in others. After successfully uploading the file, we can immediately see the result with a brief piece of information on the left and the original picture shows up on the right:



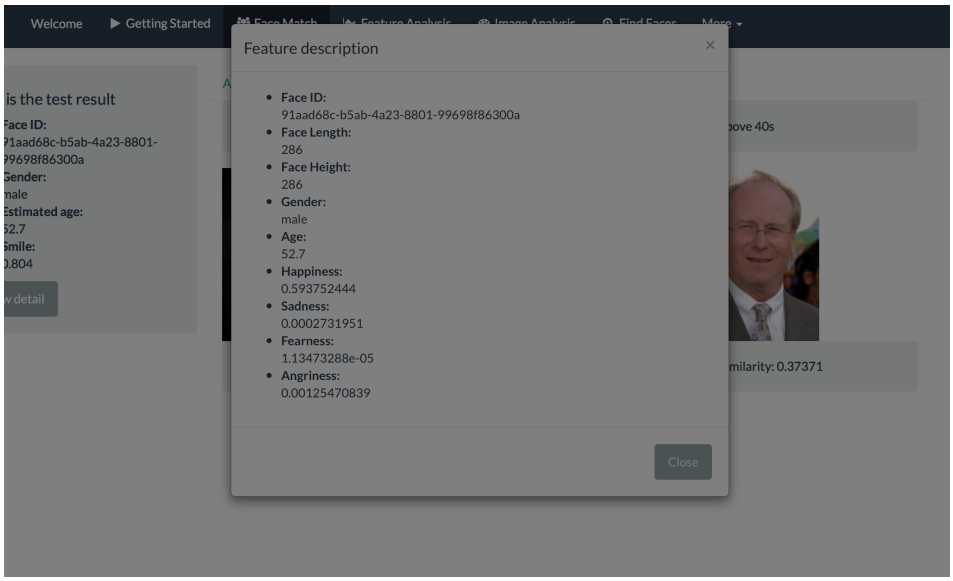
There are a few things need to be noticed in advance. First, I still don't have a full comprehension of how to get the real local path of the file uploaded, whether a relative one or an absolute one. We can see at the left side there is a parameter called "datapath". However I haven't figured out how to translate it into the real path name. Thus so far I have set the default path as the uploading path, which is the shiny directory containing the ui.r and server.r files. All the files need to be uploaded would first be moved into that directory for further processing. Secondly, the other functions attached to the navigation bars after "Getting Started" could only be triggered after an uploading of a file. So in order to experience other functions we should first finish this step.

After successfully uploading, we could continue to the next function of face matching. This is really a magic functional page where we can get a detailed facial features analysis result as well as a list of most matched faces inside the image database:

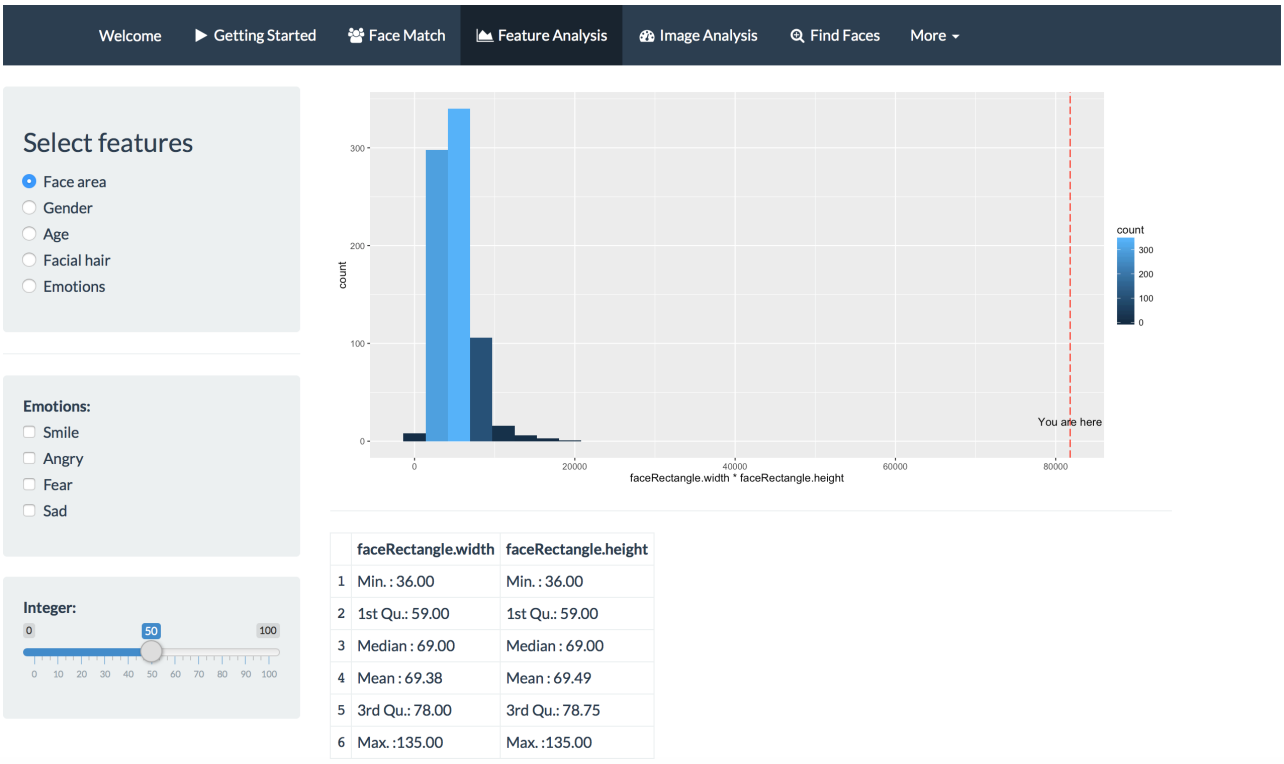




When we click the "show detail" button, we could get a float window showing all the detailed information we get from the API:



After viewing the second main function of the web page, we can move on to the third one, which contains the basic facial feature distribution along with the location of the user's own feature against the overall distribution. The discrete values would be shown as pie chart or bar plot while the continuous value would be shown as histogram diagrams. For the emotion analysis, we would use density distribution and offer the choice to overlap one with another. Additionally, we would also have specific ways to mark the user's own location:



	faceRectangle.width	faceRectangle.height
1	Min. : 36.00	Min. : 36.00
2	1st Qu.: 59.00	1st Qu.: 59.00
3	Median : 69.00	Median : 69.00
4	Mean : 69.38	Mean : 69.49
5	3rd Qu.: 78.00	3rd Qu.: 78.75
6	Max. :135.00	Max. :135.00

Select features

- ○ Face area
- ◉ Gender
- ○ Age
- ○ Facial hair
- ○ Emotions

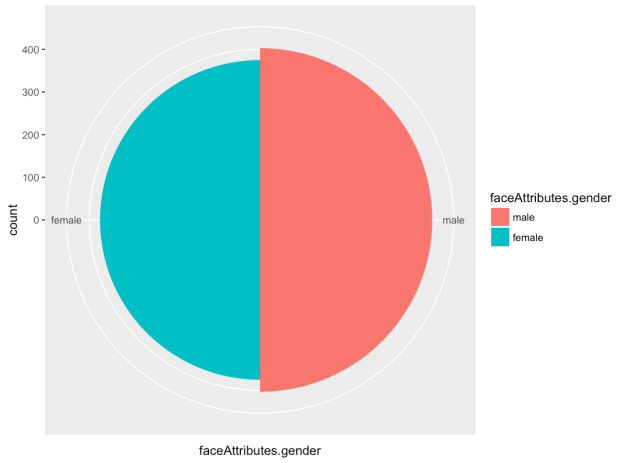
Emotions:

- ☐ Smile
- ☐ Angry
- ☐ Fear
- ☐ Sad

Integer:

0 — 50 — 100

0 10 20 30 40 50 60 70 80 90 100



	faceAttributes.gender
1	male :403
2	female:375

Select features

- ○ Face area
- ○ Gender
- ○ Age
- ◉ Facial hair
- ○ Emotions

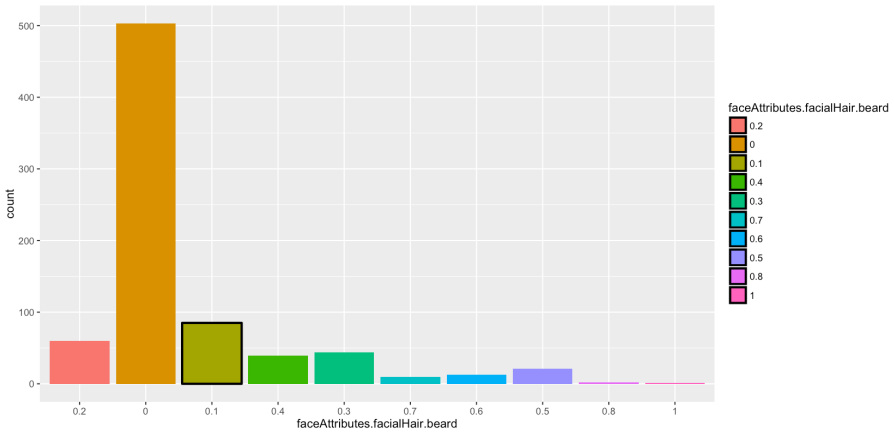
Emotions:

- ☐ Smile
- ☐ Angry
- ☐ Fear
- ☐ Sad

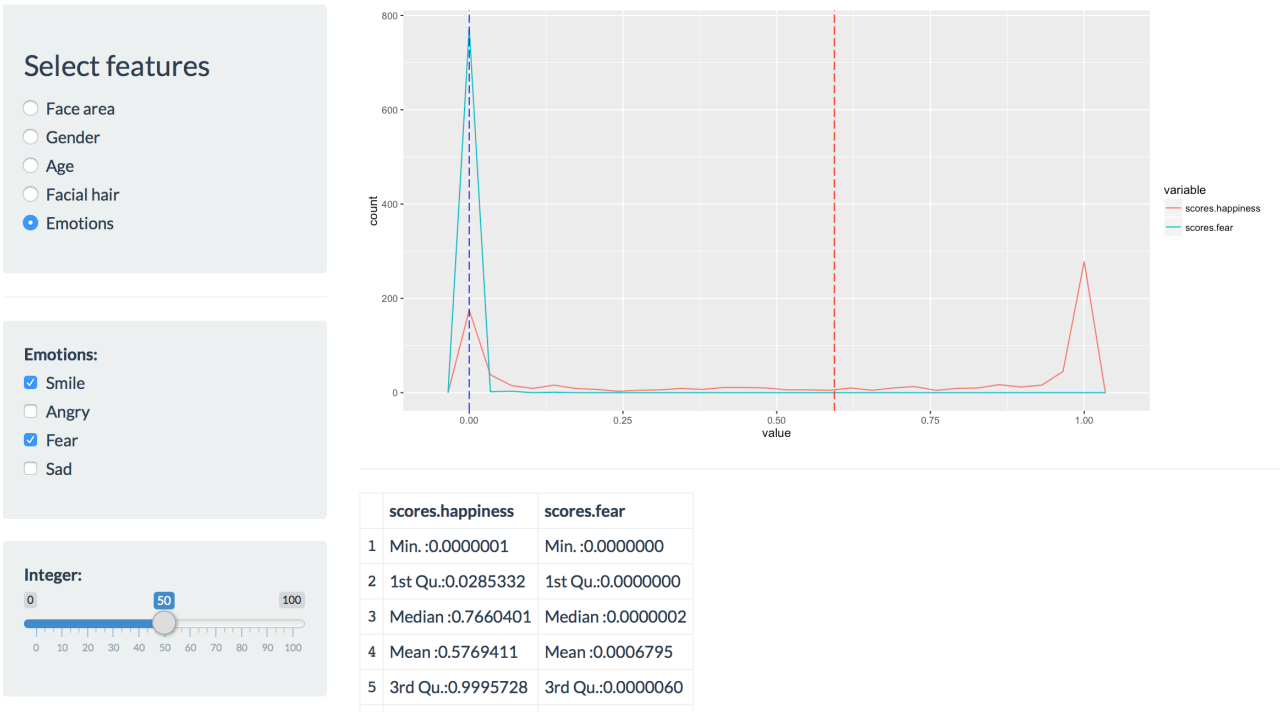
Integer:

0 — 50 — 100

0 10 20 30 40 50 60 70 80 90 100

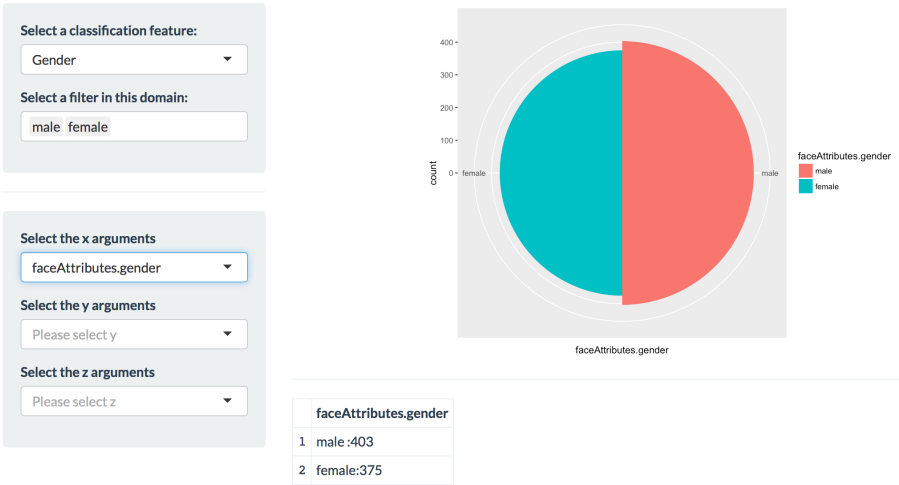


	faceAttributes.facialHair.beard
1	0 :503
2	0.1 : 85
3	0.2 : 60
4	0.3 : 44
5	0.4 : 39
6	0.5 : 21
7	(Other): 26

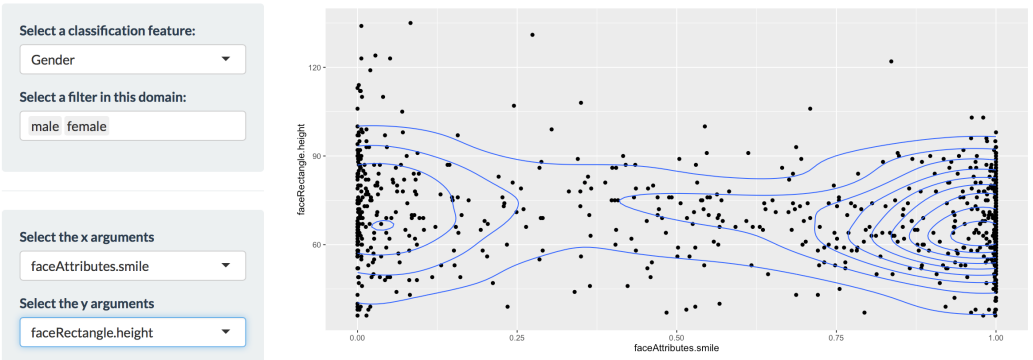


	scores.happiness	scores.fear
1	Min. :0.0000001	Min. :0.0000000
2	1st Qu.:0.0285332	1st Qu.:0.0000000
3	Median :0.7660401	Median :0.0000002
4	Mean :0.5769411	Mean :0.0006795
5	3rd Qu.:0.9995728	3rd Qu.:0.0000060

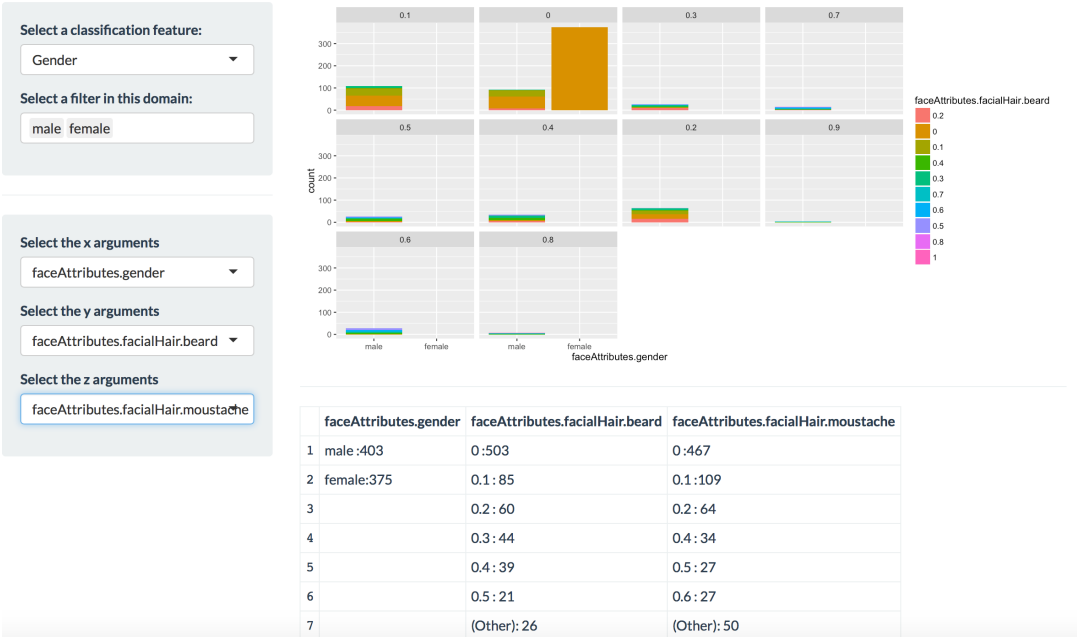
The second to last part is the multidimensional analysis. This part two first offer the user a simple filter to let users to choose the analytic data in a specific range. Then users could arbitrarily set the x, y and z axis to conduct multidimensional analysis. As the type of every axis variable varies from discrete to continuous, the type of plot would also be different from one to another:



	faceAttributes.gender
1	male :403
2	female:375



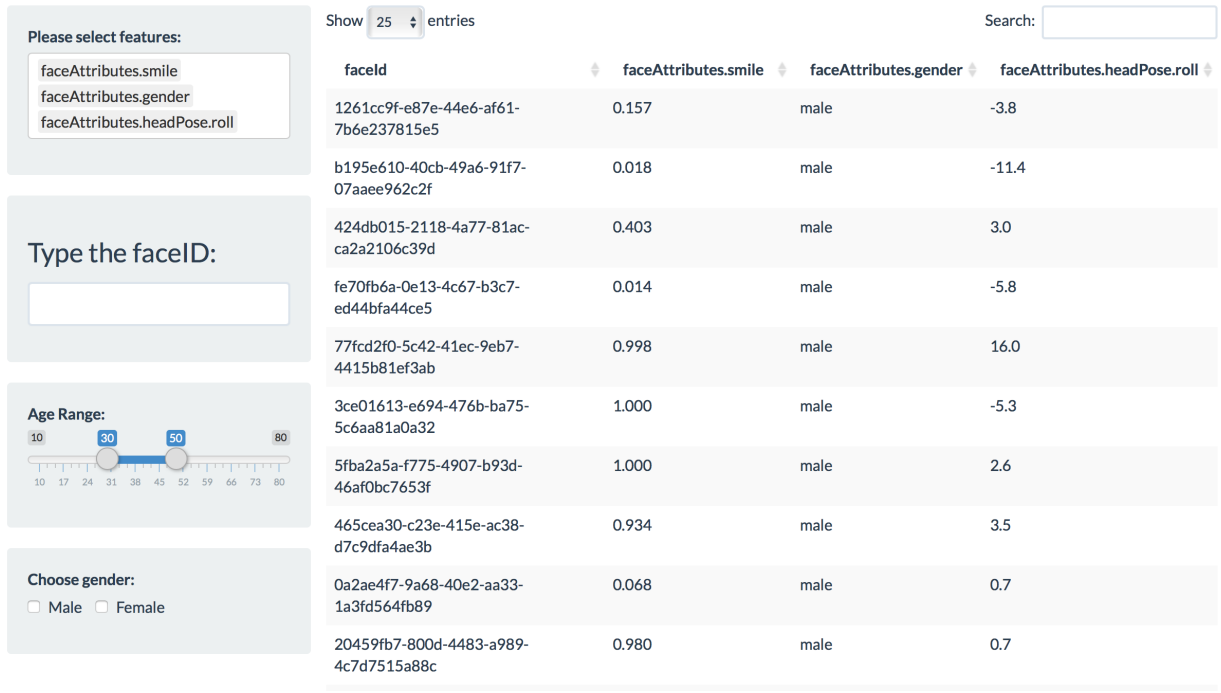
	faceAttributes.smile	faceRectangle.height
1	Min. :0.00000	Min. : 36.00
2	1st Qu.:0.08575	1st Qu.: 59.00
3	Median :0.80550	Median : 69.00
4	Mean :0.60807	Mean : 69.49
5	3rd Qu.:0.99900	3rd Qu.: 78.75
6	Max. :1.00000	Max. :135.00

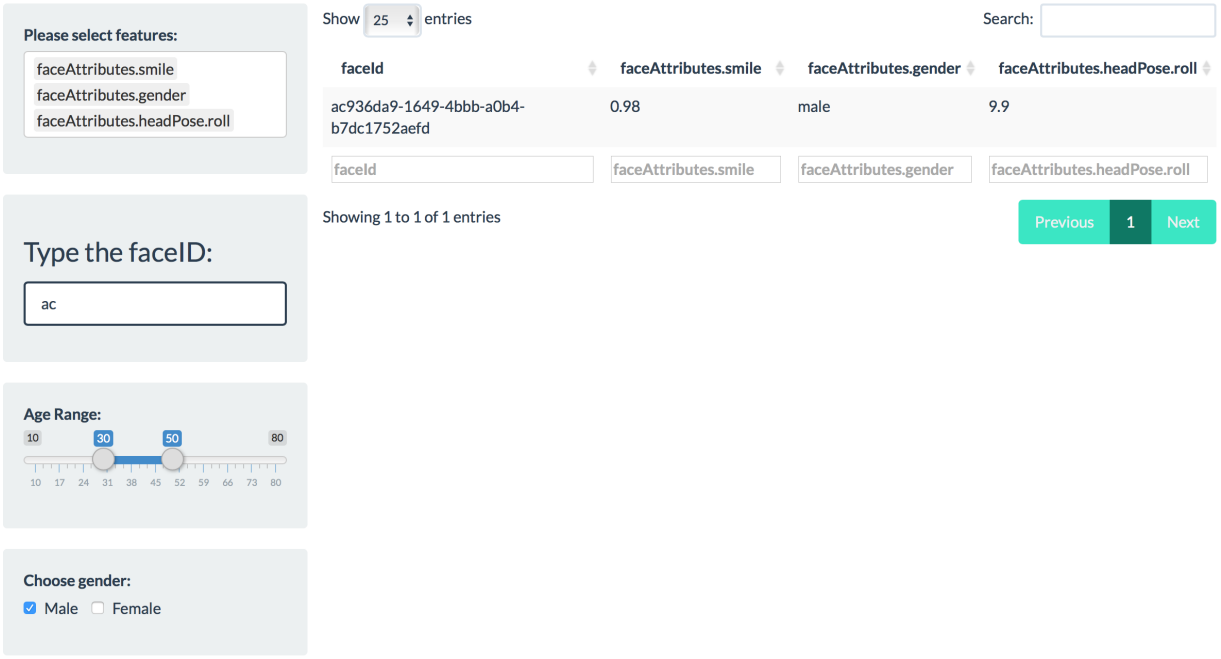


	faceAttributes.gender	faceAttributes.facialHair.beard	faceAttributes.facialHair.moustache
1	male :403	0 :503	0 :467
2	female:375	0.1 : 85	0.1 :109
3		0.2 : 60	0.2 : 64
4		0.3 : 44	0.4 : 34
5		0.4 : 39	0.5 : 27
6		0.5 : 21	0.6 : 27
7		(Other): 26	(Other): 50

	faceAttributes.smile	faceRectangle.height	faceAttributes.headPose.roll
1	Min. :0.00000	Min. : 36.00	Min. :-45.2000
2	1st Qu.:0.08575	1st Qu.: 59.00	1st Qu.: -4.7000
3	Median :0.80550	Median : 69.00	Median : -0.4000
4	Mean :0.60807	Mean : 69.49	Mean : -0.8139
5	3rd Qu.:0.99900	3rd Qu.: 78.75	3rd Qu.: 3.2000
6	Max. :1.00000	Max. :135.00	Max. : 35.6000

The last part is the simple implementation of the searching function through a large table. On the top of the side bar, we could select the columns we want to show up. And then we could type in a sub string to search for a specific group of IDs through faceId. Although a datatable object has its own search function, it aims to search with the functionality of "contain" while the search function in the side bar aims to search by "start with":





3.Conclusion

By far I have implemented a bunch of functions for a Shiny interactive app, which includes many basic Shiny components and interactive logics. However, this is not the end, I will continue to enhance and modify it, to add new functions and to adjust the old ones. And in the near future, I wish I could enlarge the current database dynamically and make it able to update itself by periodically crawling on other SNS platforms such as Facebook or Instagram.