

Predictive Beer Analytics

Joshua Weinstein, Jim Sundkvist, Marek Kühn

Abstract—This project aims to apply various concepts of Python programming as well as data mining and processing. Based on a large amount of user and beer data, the application uses varying techniques including data mining, natural language processing, and color analysis to predict user-defined beers' desirability in the world through a web application.

I. INTRODUCTION

The Predictive Beer Analytics project has been initiated in the Fall of 2014 as a part of "Data Mining Using Python" course at Denmark Technical University. Its purpose is not only exercise the study material, but also to give users some useful, informative results. To satisfy this, we used our previous experience with web development and created a lightweight web-service with visual representations of the results. The users can design their own beer by defining its style, alcohol by volume, keywords and dominant label color. This information is then used to generate a desirability map (ref).

The application to retrieve and analyse data along with our web application (without the database) can be found on our GitHub account. The web-service can be run locally with an installation of [Django](#), [MySQL](#), and a copy of the database.

II. DATA MINING

Being the main aspect of any data mining project, retrieving the data, and a lot of it, is the first course of action. All data used by Predictive Beer Analytics was gathered through the [Untappd API](#), a beer review website which provides access to the data it collects. Unfortunately, the service limited our access to their data to 100 calls an hour. This significantly increased the amount of time required to gather data. The script, `untappd.py` provides data structures to store Untappd data as well as uses the `Requests` module to make calls to Untappd's servers. Due to the available requests through the API service, it was necessary to first retrieve a list of active users and then later use their username to gather information about beer they have reviewed in the past. Over the course of roughly two weeks of mining, a reasonable amount of information was stored. Seen in Figure 1, over 200.000 user reviews are used to predict the desirability of a beer based on location. To keep this information up-to-date, `dataQuantity.py` script has been included in the library.

After data has been gathered, updated, and normalized with methods in the main script `predictiveBeerAnalytics.py`, it is split up into users and beers where users contain a location, using the [Google Geocoding API](#), and their ratings and beers are composed of descriptive keywords, alcohol content, label's url, style, and other unused information. These are then

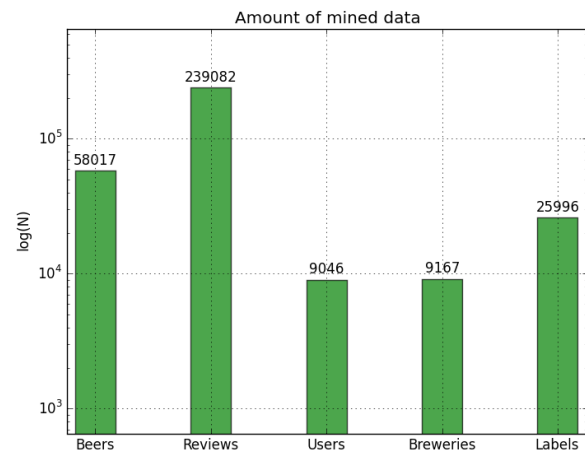


Fig. 1. Amount of used data (by the time of presentation)

used in conjunction with other scripts we have written to analyse label colors, determine keywords from descriptions, find relationships between alcohol content and location, and determining relationships between a user's location and the style of beer they enjoy.

III. MACHINE LEARNING

There are two applications of machine learning:

- **Natural language processing**
- **Image color analysis**

With its extraordinary lexical resources and easy-to-use interface, **Natural Language Toolkit (NLTK)** was an easy choice. It is used to tokenize the sentence in order to classify the words according to its type. After various filtering (usage threshold, invalid characters etc.), we calculate the average rating of each keyword. The best, worst and the most used keywords are presented on the website. The relevance of the results is to be discussed later. The actual **keyword extraction** function is defined in `keywordExtractor.py`, making it easy to reuse in other projects. It is possible to introduce regular expressions in order to improve the keyword recognition, but the results are reasonable even without using it.

While thinking about all aspects that can possibly affect the beer desirability, we introduced also the visual point of view. We have no information about the color of the bottle. Nevertheless, more than half of the beers from untappd have a link to the beer label picture attached. All we had to do is determine how the beers are rated in relation to the

label color. The **dominant colors clustering** is done with `sklearn.cluster` module, built on `NumPy`, `SciPy`, and `matplotlib`, well-known open-source libraries. All of which are also used regularly in other context in this project. Currently, we use **K-means clustering** to determine five of the most used colors in each label. The class responsible for the calculations `labels.py:Image` is designed such as the parameters like number of clustered colors are easy to change in the future.

To check if the result is correct, we use the clustered colors to rebuild the picture as in the figure 2. Since we observed a lot of labels not having rectangular shape, we implemented a **custom masking algorithm** based on the color differences. It seeks a border at which certain threshold is overrode and generates `numpy.mask`. This is covered by `labels.py:Mask`.

Having these colors extracted, they need to be associated with ratings. Otherwise there would be no measurable impact on the total beer desirability. We couldn't match those color directly to the user input, because that way its rating would have to be calculated every time the request is made, which is not acceptable given the amount of data. So we come up with a solution by using template color palette. It serves as a fixed categorizing root. None of the available functions were suitable for comparing the RGB values of the colors, so we designed our own classification algorithm. It uses the well-known euclidean distance formula to determine which color from the template color palette is the closest. However, simple distance in RGB colorspace doesn't correspond to human perception. A change of one parameter results in much more different color than the same change distributed between multiple parameters. It was possible to overcome this by converting the colors to YUV colorspace (Fig. 3), which takes human perception into account, making the response more or less linear.

IV. WEB SERVICE

The web-service uses one of the most popular Python frameworks - Django. This allowed us to create elegant web

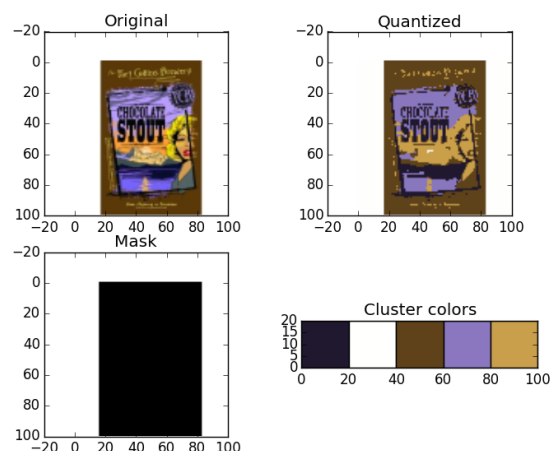


Fig. 2. Image color clustering



Fig. 3. YUV linear colorspace sample for $Y = 0$

application while still being able to focus on the server-side python scripts rather than web-design.

TODO add more

The data that the webapp uses are stored in a MySQL database. The data itself is generated from the datamining scripts and saved to the webapps database by importing them as a scv file. The webapp uses Django's database-abstraction API to perform the basic CRUD functionality to the datamodel SE FIGURE ?.

When it comes to security Django has some great features that will bring about basic security with minimum effort. Django's database-abstraction API automatically sanitizes any input so SQL-injection is not a problem. Cross-site-scripting is dealt with by using csrf tokens. Responses automatically has csrf tokens inside so when doing POST you only need to add it. In a Form simply having Django also have some predefined user authentication and authorization functionality. In the webapp we do not use any access control so this part is unnecessary until a time arrive when we do need it.

The decision to use these technologies is pretty straight forward: The technologies were familiar, well used and documented, and easy to use. This lead to quite a fast development of the pba GUI.

V. DISCUSSION

Keywords - Regular expression.
Hardcoded web color palette

VI. FURTHER DEVELOPMENT

Collecting data
Machine learning
Website
Including brewery information

VII. RESULTS

At the beginning of Predictive Beer Analytics, our goal was to find relationships between the alcohol content or style of a beer and the location of a user's rating to determine what areas enjoy the most. As the project progressed, ambitions rose and

we wanted to see if the color of the label or the description of beers made a significant impact on the desirability of a beer. Using `PBAMap.py`, `keywordClassifier.py` we are able to graphically visualize the effects of specific words, alcohol content, and beer style on a user's rating. Figure 4 shows the consistent and seemingly global idea that people typically enjoy stronger beers. While there is a clear correlation between user reviews and a beer's alcohol content, the type of beers an area likes are typically varied and relatively unpredictable. Though it does seem that the United States enjoys IPAs more than ciders and fruitier beers.

It was also interesting to see that certain words used in a beer's description had a large impact on a user's review of the beer. For example, beer descriptions containing words like "inexpensive", "alcohol-free", and "grandma" all had an average rating of below three out of five stars. Whereas "sought-after", "chocolate-covered", and "fabled" all had ratings above four stars.

VIII. DISCUSSION

IX. CONCLUSION

APPENDIX A

CODE LISTINGS

LISTINGS

../machine_learning/lib/predictiveBeerAnalytics.py	5
../machine_learning/lib/untappd.py	10
../machine_learning/lib/keywordExtractor.py	11
../machine_learning/lib/keywordClassifier.py	12
../machine_learning/lib/dataPoints.py	14
../machine_learning/lib/PBAMap.py	14
../machine_learning/lib/labels.py	17
../machine_learning/lib/fileReader.py	21
../machine_learning/lib/dataQuantity.py	22

```

1  """
2  This is the main script for Predictive Beer Analytics project.
3
4  Use it to mine and process new data.
5  """
6
7  import argparse
8  import jsonpickle as jpickle
9  import os
10 import operator
11 import csv
12 # import cPickle
13 from time import sleep
14 import untappd as UT
15 import PBAMap
16 import keywordExtractor as extract
17 import dataPoints as dp
18 import labels
19 import fileReader as files
20
21 parser = argparse.ArgumentParser(prog='PBA')
22 group = parser.add_mutually_exclusive_group(required=True)
23 group.add_argument('--users', action='store_true',
24                   help='Add to the list of users')
25 group.add_argument('--reviews', action='store_true',
26                   help='Add to the list of users, beers, and breweries')
27 group.add_argument('--normalizeData', action='store_true',
28                   help='Alter Untappd data for privacy')
29 group.add_argument('--keywords', action='store_true',
30                   help='Extract keywords from beer descriptions and attach to beer')
31 group.add_argument('--dataPoints', action='store_true',
32                   help='Create list of data points from user locations, ratings, \
33                        beer alcohol content, and beer style')
34 group.add_argument('--styles', action='store_true',
35                   help='Create csv file of allowable beer styles to make maps with')
36 group.add_argument('--abvMap', type=float,
37                   help='Create map of ratings using data points on maps \
38                        provided alcohol level. Data is split into abv ranges: \
39                        0, 0.1-3.9, 4.0-4.9, 5.0-5.9, 6.0-6.9, 7.0-7.9, 8.0-8.9, \
40                        9.0-9.9, 10.0-10.9, 11.0+ Requires GEOS Library and \
41                        mpl_toolkits.basemap')
42 group.add_argument('--styleMap', type=str,
43                   help='Create map of ratings using data points on maps \
44                        provided beer style. Styles limited to those listed in \
45                        styles.csv. Requires GEOS Library and mpl_toolkits.basemap')
46 group.add_argument('--colorPalette', action='store_true',
47                   help='Download label images, clusterize colors, \
48                        generate global color rating palette of N colors.')
49 args = parser.parse_args()
50
51 # set the api settings and create an Untappd object
52 untappd = UT.Untappd()
53 untappd.settings('../apiConfig.ini')
54
55
56
57 def writeJSONFile(path, data):
58     """Write JSON file."""
59     with open(path, 'wb') as jsonFile:
60         json = jpickle.encode(data)
61         jsonFile.write(json)
62
63
64 def usersList():
65     """
66     Parse through data from / thepub to get unique usernames, user ids,
67     and locations. Stores this information in a csv file to be used in later api

```

```

68 requests_limited_to_100_api_calls_per_hour_requiring_sleep_method.
69 May_be_run_multiple_times_to_retrieve_continuously_run_until_user_stops_script.
70 """
71
72 usersList = files.readUsers()
73 apiCount = 0
74 userNameCountAdditions = 0
75 while (True):
76     # get 25 most recent updates
77     data = untappd.getPubFeed()
78     apiCount += 1
79     print 'apiCount:' + str(apiCount)
80     checkins = data['response']['checkins']['items']
81     # each response has 25 items, each with a username
82     for checkin in checkins:
83         userId = checkin['user']['uid']
84         username = checkin['user']['user_name']
85         userLocation = checkin['user']['location']
86         if hash(str(userId)) not in usersList:
87             if userLocation != '':
88                 userNameCountAdditions += 1
89                 userAttribs = {'uid': str(userId), 'username': username,
90                               'location': {'name': unicode(userLocation).encode("utf-8")}, 'ratings': {}}
91                 user = UT.UntappdUser(userAttribs)
92                 usersList[hash(str(userId))] = user
93     writeJSONFile('../data/users.json', usersList)
94     userCount = len(usersList)
95     print 'Total Users:' + str(userCount)
96     # Untappd only allows 100 api requests per hour. Sleep for 38
97     # seconds between requests
98     sleep(37)
99
100
101 def userReviews():
102     """
103     Parse through user reviews /user/ beers/{username}
104     Retrieves at most 50 reviews per user, retains review, beer, and
105     brewery information. After querying the api, remove username to
106     lessen privacy concerns with untappd data.
107     """
108     usersList = files.readUsers()
109     beersList = files.readBeers()
110     breweryList = files.readBreweries()
111     breweryToBeers = files.readBreweryToBeers()
112
113     total = 0
114     totalUsersComplete = 0
115     for userHash, user in usersList.iteritems():
116         totalUsersComplete += 1
117         # if the data has been normalized, old data will not
118         # have usernames. Ignore older users which may have
119         # already gotten reviews
120         if user.username:
121             userId = user.uid
122             username = user.username
123             user.username = None
124             userReviewCount = 0
125             offsetTotal = 0
126             ratings = {}
127
128             print 'Processing' + str(userId) + ':' + username
129             # each response returns at most 25 reviews. To get more user
130             # reviews, call again with an offset get at most 50 reviews
131             # from the same user
132             while (userReviewCount < 2):
133                 print username + ':' + str(userReviewCount + 1)
134                 data = untappd.getUserReviewData(username, offsetTotal)
135                 offset = data['response']['beers']['count']
136                 offsetTotal += offset
137                 reviews = data['response']['beers']['items']
138                 for review in reviews:
139                     userRating = review['rating_score']
140                     if userRating > 0:
141                         beerInfo = review['beer']
142                         breweryInfo = review['brewery']
143                         # fill in beer information
144                         if hash(str(beerInfo['bid'])) not in beersList:
145                             stylesList = []
146                             style = unicode(beerInfo['beer_style']).encode("utf-8")
147                             styles = style.lower().title().split('/')
148                             for style in styles:
149                                 style = style.strip()
150                                 stylesList.append(style)
151                             beerAttribs = {
152                                 'bid': str(beerInfo['bid']),
153                                 'name': unicode(beerInfo['beer_name']).encode("utf-8"),

```

```

154         'label': beerInfo['beer_label'],
155         'abv': beerInfo['beer_abv'],
156         'ibu': beerInfo['beer_ibu'],
157         'style': stylesList,
158         'description': unicode(beerInfo['beer_description']).encode("utf-8"),
159         'rating': beerInfo['rating_score'],
160         'numRatings': 1,
161         'brewery': str(breweryInfo['brewery_id'])
162     }
163     beer = UT.UntappdBeer(beerAttribs)
164     beersList[hash(beer.bid)] = beer
165 else:
166     beersList[hash(str(beerInfo['bid']))].numRatings += 1
167 # fill in brewery information
168 if hash(str(breweryInfo['brewery_id'])) not in breweryList:
169     breweryAttribs = {
170         'breweryId': str(breweryInfo['brewery_id']),
171         'name': unicode(breweryInfo['brewery_name']).encode("utf-8"),
172         'label': breweryInfo['brewery_label'],
173         'country': unicode(breweryInfo['country_name']).encode("utf-8"),
174         'location': unicode(breweryInfo['location']).encode("utf-8")
175     }
176     brewery = UT.UntappdBrewery(breweryAttribs)
177     breweryList[hash(brewery.breweryId)] = brewery
178
179 # map brewery_id to a list of beers produced there
180 if hash(str(breweryInfo['brewery_id'])) not in breweryToBeers:
181     # store the current beer in a list of beers of
182     # the brewery
183     breweryToBeers[hash(str(breweryInfo['brewery_id']))] = {str(breweryInfo['brewery_id']): [str(beerInfo['bid'])]}
184 else:
185     # add current beer to brewery's list of beers
186     breweryToBeers[hash(str(breweryInfo['brewery_id']))][str(breweryInfo['brewery_id']).append(str(beerInfo['bid']))]
187
188 # add list of beer ratings to user
189 ratings[str(beerInfo['bid'])] = userRating
190 userReviewCount += 1
191 user.ratings = ratings
192
193 # store the dictionaries after new data so user doesn't kill process before writing
194 # with open('../data/users.json', 'wb') as usersFile:
195 #     json = jpickle.encode(usersList)
196 #     usersFile.write(json)
197 # with open('../data/beers.json', 'wb') as beersFile:
198 #     json = jpickle.encode(beersList)
199 #     beersFile.write(json)
200 # with open('../data/breweries.json', 'wb') as breweriesFile:
201 #     json = jpickle.encode(breweryList)
202 #     breweriesFile.write(json)
203 # with open('../data/breweryToBeers.json', 'wb') as breweryToBeersFile:
204 #     json = jpickle.encode(breweryToBeers)
205 #     breweryToBeersFile.write(json)
206
207 # if the offset is less than 25, then there are no more reviews to retrieve
208 if offset < 25:
209     break
210 writeJSONFile('../data/users.json', usersList)
211 writeJSONFile('../data/beers.json', beersList)
212 writeJSONFile('../data/breweries.json', breweryList)
213 writeJSONFile('../data/breweryToBeers.json', breweryToBeers)
214
215 total += len(ratings)
216 print str(userId) + ': ' + username + ', _Processed: ' + str(len(ratings)) + ' _reviews'
217 print 'Total _Reviews: ' + str(total)
218 print 'Total _Users _Completed: ' + str(totalUsersComplete)
219 sleep(37 * (userReviewCount))
220 else:
221     total += len(user.ratings)
222
223
224 def normalizeUsers():
225     """
226     Change the user ids so the information can be made public and
227     use the googlemaps module to determine the user's location.
228     """
229     usersList = files.readUsers()
230     newUsersList = {}
231
232     i = 1
233     newUid = 1
234     for hashId, user in usersList.iteritems():
235         uid = user.uid
236         user.uid = str(newUid)
237         location = user.location
238         if location['name'] != "" and 'lat' not in location:
239             if isinstance(location['name'], unicode):

```

```

240         location = location['name'].encode('utf-8')
241     else:
242         location = location['name']
243
244     mapInfo = PBAMap.getLatLong(location, i)
245     i += 1
246     if mapInfo == 'apiLimit':
247         print str(i) + "\tAt daily API limit. Update script and repeat tomorrow"
248     elif mapInfo != '':
249         user.location = {
250             'name': location,
251             'lat': mapInfo['lat'],
252             'lng': mapInfo['lng'],
253         }
254         if 'country' in mapInfo:
255             user.location['country'] = mapInfo['country']
256         print str(i), user.location
257     else:
258         print str(i), "checked: none"
259         user.location = {'name': ''}
260     newUid += 1
261     newUsersList[hash(str(uid))] = user
262
263     writeJSONFile('../data/users.json', newUsersList)
264     print "User ids, usernames, and locations updated\n"
265
266
267 def beerKeywords():
268     """Extract keywords from beer descriptions and rate it."""
269     beersList = files.readBeers()
270     print 'beers.json loaded...'
271
272     # List of keywords generation
273     keywordsList = {}
274     position = 0
275
276     for hashId, beer in beersList.iteritems():
277         beer.keywords = []
278         beer.keywords = extract.extractKeywords(beer.description)
279         for keyword in beer.keywords:
280             if keyword in keywordsList:
281                 keywordsList[keyword][0] += beer.rating
282                 keywordsList[keyword][1] += 1
283             else:
284                 keywordsList[keyword] = [beer.rating, 1]
285         position += 1
286         if (position % 100) == 0:
287             print 'Processed' + str(position) + '/' + str(len(beersList)) + ' beers.'
288
289     writeJSONFile('../data/beers.json', beersList)
290     writeJSONFile('../data/keywords.json', keywordsList)
291
292
293 def createDataPoints():
294     """Make the data points of user locations for the map generation."""
295     usersList = files.readUsers()
296     beersList = files.readBeers()
297     points = []
298     i = 1
299     for hashId, user in usersList.iteritems():
300         if 'lat' in user.location and user.ratings:
301             for bid, rating in user.ratings.iteritems():
302                 country = None
303                 if 'country' in user.location:
304                     country = user.location['country']
305                 pointAttribs = {'lat': user.location['lat'], 'lng': user.location['lng'],
306                                'country': country, 'abv': beersList[str(hash(bid))].abv, 'rating': rating,
307                                'style': beersList[str(hash(bid))].style}
308                 point = dp.dataPoint(pointAttribs)
309                 points.append(point)
310                 if i % 1000 == 0:
311                     print "Points added:" + str(i)
312                 i += 1
313     data = dp.dataPoints(points)
314     writeJSONFile('../data/dataPoints.json', data)
315
316
317 def createABVMap():
318     """Make a color map of specific alcohol by volume."""
319     print "Drawing user rating maps of beers with an alcohol concentration of" + str(args.abvMap) + '%'
320     dataPoints = files.readDataPoints()
321     if len(dataPoints) > 0:
322         abv = int(args.abvMap)
323         points = []
324         for point in dataPoints:
325             if int(point.abv) == abv:

```

```

326         points.append(point)
327     elif (abv in [1, 2, 3]) and (int(point.abv) in [1, 2, 3]):
328         points.append(point)
329     elif (abv > 11 and int(point.abv) > 11):
330         points.append(point)
331     print str(len(points)) + "_points_of_data"
332     PBAMap.drawMap(points, abv)
333 else:
334     print "No_data_points_found"
335
336
337 def createStyleMap():
338     print "Drawing_user_rating_maps_of_beers_with_a_style_of_" + str(args.styleMap)
339     dataPoints = files.readDataPoints()
340     if len(dataPoints) > 0:
341         style = args.styleMap
342         points = []
343         for point in dataPoints:
344             if style in point.style:
345                 points.append(point)
346         print str(len(points)) + "_points_of_data"
347         PBAMap.drawMap(points, style)
348     else:
349         print "No_data_points_found"
350
351
352 def createCommonStyles():
353     """Generate common beer styles and save it to csv file."""
354     beersList = files.readBeers()
355     allStyles = {}
356     for hashId, beer in beersList.iteritems():
357         styles = beer.style
358         for style in styles:
359             numRatings = beer.numRatings if (hasattr(beer, 'numRatings')) else 0
360             if style in allStyles:
361                 allStyles[style] += numRatings
362             else:
363                 allStyles[style] = numRatings
364
365     sorted_styles = sorted(allStyles.items(), key=operator.itemgetter(1))[-20:]
366     with open('../data/styles.csv', 'wb') as stylesCSV:
367         csvwriter = csv.writer(stylesCSV, delimiter=',',
368                                quotechar='"')
369         csvwriter.writerow(["id", "style", "numRatings"])
370         i = 1
371         for style in sorted_styles:
372             csvwriter.writerow([i, unicode(style[0]).encode("utf-8"), style[1]])
373             i += 1
374
375
376 def processLabels():
377     """
378     Prediction of how dominant label color affects the beer rating.
379
380     Download beer bottle labels, extract dominant colors,
381     make the color palette, flag each color and calculate
382     average rating of that color.
383     """
384     beersList = files.readBeers()
385     beerColorsDict = files.readBeerColors()
386
387     # Path for saving the images
388     path = "../data/labels/"
389
390     fileList = os.listdir(path)
391     fileList = [item for item in fileList
392                 if item.split(".")[-1] in ('jpeg', 'jpg', 'png')]
393
394     # Download and save images
395     labels.download(beersList, path, fileList)
396
397     # Number of label colors to cluster
398     nColors = 5
399     i = 0
400     # stop = 6 # Then use the whole list.
401
402     # Loop over images in the folder
403     for file in fileList:
404         i += 1
405         bid = unicode(file.split('.')[0])
406         if (bid in beerColorsDict and
407             len(beerColorsDict[bid].colorPaletteFlags) == nColors):
408             continue
409
410     print ("Processing_image_" + file +
411           "_" + str(i - 1) + "/" + str(stop) + ")")

```



```

412         beerLabel = labels.Image(path + file)
413
414         beerLabel.preprocess()
415         beerColor = beerLabel.clusterize(nColors)
416         beerColorsDict[bid] = beerColor
417
418         # Only for presentation
419         # beerLabel.quantizeImage()
420         # beerLabel.showResults()
421
422         # Generate the color palette with ratings - Classification
423         colorPalette = labels.ColorPalette()
424         colorPalette.build(beerColorsDict, beersList)
425
426         # Write the colorsFile - dict{ 'bid': beerColor{RGB,intensity}}
427         writeJSONFile('../data/beerColors.json', beerColorsDict)
428         writeJSONFile('../data/colorPalette.json', colorPalette.palette)
429
430         print 'Color palette saved.'
431
432     if args.users:
433         usersList()
434     elif args.reviews:
435         userReviews()
436     elif args.normalizeData:
437         normalizeUsers()
438     elif args.keywords:
439         beerKeywords()
440     elif args.styles:
441         createCommonStyles()
442     elif args.colorPalette:
443         processLabels()
444     elif args.dataPoints:
445         createDataPoints()
446     elif args.abvMap >= 0:
447         createABVMap()
448     elif args.styleMap:
449         if args.styleMap in files.readBeerStyles():
450             createStyleMap()
451         else:
452             print "No support for style:" + args.styleMap
453             print "See styles.csv for supported styles."

```

```

1  import requests
2  import json
3  from ConfigParser import SafeConfigParser
4
5
6  class UntappdUser:
7      """
8      Representation of an untappd user
9      """
10     def __init__(self, attrs):
11         self.uid = attrs['uid']
12         self.username = attrs['username']
13         self.location = attrs['location']
14         self.ratings = attrs['ratings']
15
16
17     class UntappdBeer:
18         """
19         Representation of an untappd beer
20         """
21         def __init__(self, attrs):
22             self.bid = attrs['bid']
23             self.name = attrs['name']
24             self.label = attrs['label']
25             self.abv = attrs['abv']
26             self.ibu = attrs['ibu']
27             self.style = attrs['style']
28             self.description = attrs['description']
29             self.rating = attrs['rating']
30             self.numRatings = attrs['numRatings']
31             self.brewery = attrs['brewery']
32
33
34     class UntappdBrewery:
35         """
36         Representation of an untappd brewery
37         """
38         def __init__(self, attrs):
39             self.breweryId = attrs['breweryId']
40             self.name = attrs['name']
41             self.label = attrs['label']
42             self.country = attrs['country']
43             self.location = attrs['location']

```

```

44
45
46 class Untappd:
47     """
48     Untappd object which handles everything related to untappd
49     and the data obtained through untappd's api
50     """
51     def __init__(self):
52         self.client_id = ''
53         self.client_secret = ''
54         self.endpoint = ''
55         self.request_header = {}
56         self.users = {}
57         self.beers = {}
58         self.breweries = {}
59
60     def settings(self, filename):
61         config = SafeConfigParser()
62         config.read(filename)
63         client_id = config.get('untappd', 'clientId')
64         client_secret = config.get('untappd', 'clientSecret')
65         endpoint = config.get('untappd', 'endpoint')
66         request_header = {'User-Agent': config.get('untappd', 'header')}
67
68         self.client_id = client_id
69         self.client_secret = client_secret
70         self.endpoint = endpoint
71         self.request_header = request_header
72
73     def createUrl(self, method):
74         """
75         Creates the api url for the GET request
76         """
77         return self.endpoint + '/' + method
78
79     def getPubFeed(self):
80         """
81         Retrieves information which includes the usernames of
82         active members of the site
83         """
84         method = 'thepub'
85         url = self.createUrl(method)
86         parameters = {'client_id': self.client_id,
87                     'client_secret': self.client_secret}
88         response = requests.get(url, headers=self.request_header,
89                               params=parameters)
90         data = json.loads(response.text)
91         return data
92
93     def getUserReviewData(self, username, offset):
94         method = 'user/beers/' + username
95         url = self.createUrl(method)
96         parameters = {'offset': offset, 'client_id': self.client_id,
97                     'client_secret': self.client_secret}
98         response = requests.get(url, headers=self.request_header,
99                               params=parameters)
100         data = json.loads(response.text)
101         return data

```

```

1  """
2  Extract keywords from beer descriptions obtained from Untappd.
3  Use NLTK natural language processing tool.
4  """
5
6  import nltk
7  from nltk.corpus import wordnet as wn
8  import re
9  import jsonpickle as jpickle
10 import untappd as UT
11
12
13 def extractKeywords(text):
14     """ Extract the keywords from the given text. """
15     keywords = []
16     sentences = nltk.sent_tokenize(text)
17     words = [nltk.word_tokenize(sent) for sent in sentences]
18     words = [nltk.pos_tag(sent) for sent in words]
19
20     # Regex for more precise extraction
21     # grammar = "NP: {<DT>?<JJ>*<NN>}"
22     # cp = nltk.RegexpParser(grammar)
23
24     for w in words:
25         for ww in w:
26             # Pick conditions based on word types from nltk
27             if (ww[1] == 'NN' or ww[1] == 'JJ') and len(ww[0]) > 3:

```

```

28         keywords.append(ww[0])
29     return keywords

```

```

1     """
2     Show the usage of keywords in various ways.
3     Write sorted csv and text files for export to database and graphics..
4     """
5
6     import jsonpickle as jpickle
7     import sys
8     import numpy as np
9     import matplotlib.pyplot as plt
10    import csv
11
12
13    # Load keywords
14    try:
15        keywordsFile = open('../data/keywords.json', 'rb')
16    except:
17        print 'Keywords.json not found.'
18        sys.exit()
19
20    try:
21        f = keywordsFile.read()
22        keywordsRawDict = jpickle.decode(f)
23    except:
24        print 'Keywords list corrupted'
25        sys.exit()
26    keywordsFile.close()
27
28    # Filters - minimum usage and lowercase conversion
29    votesThreshold = 50
30    keywordsDict = {k.lower(): v for (k, v) in
31                    keywordsRawDict.iteritems()
32                    if (v[1] >= votesThreshold)}
33
34    # Dictionary of sorted lists
35    SortedByRating = {}
36    SortedByRating['keywords'] = []
37    SortedByRating['ratings'] = []
38    SortedByRating['usage'] = []
39    for word in sorted(keywordsDict.items(),
40                      key=lambda k: (k[1][0] / k[1][1]), reverse=True):
41        ratingSum = word[1][0]
42        usage = word[1][1]
43        if usage > 2:
44            SortedByRating['keywords'].append(word[0])
45            SortedByRating['ratings'].append(ratingSum / usage)
46            SortedByRating['usage'].append(usage)
47
48    # Sorting by keywords usage
49    SortedByUsage = {}
50    SortedByUsage['keywords'] = []
51    SortedByUsage['ratings'] = []
52    SortedByUsage['usage'] = []
53    for word in sorted(keywordsDict.items(),
54                      key=lambda k: (k[1][1]), reverse=True):
55        ratingSum = word[1][0]
56        usage = word[1][1]
57        SortedByUsage['keywords'].append(word[0])
58        SortedByUsage['ratings'].append(ratingSum / usage)
59        SortedByUsage['usage'].append(usage)
60
61
62    def plotBestKeywords(n=10):
63        """
64        Plot a graph of keywords associated with the best rated beers.
65        param n: Amount of keywords.
66        """
67
68        fig, ax = plt.subplots()
69        index = np.arange(n)
70
71        bar_width = 0.35
72        opacity = 0.4
73        error_config = {'ecolor': '0.3'}
74
75        bars1 = plt.barh(index, SortedByRating['ratings'][0:n],
76                         bar_width,
77                         alpha=opacity,
78                         color='b',
79                         error_kw=error_config,
80                         label='Rating_avg.')
81
82        bars2 = plt.barh(index + bar_width, SortedByRating['usage'][0:n],
83                         bar_width,

```

```

84         alpha=opacity ,
85         color='r' ,
86         error_kw=error_config ,
87         label='Usage')
88
89     plt.title('Beer-rating-keywords')
90     plt.yticks(index + bar_width, SortedByRating['keywords'][0:n])
91     plt.legend()
92     plt.tight_layout()
93     plt.show()
94
95
96 def plotWorstKeywords(n=10):
97     """
98     Plot a graph of keywords associated with the worst rated beers.
99     :param n: Amount of keywords.
100     """
101
102     fig, ax = plt.subplots()
103     index = np.arange(n)
104
105     bar_width = 0.35
106     opacity = 0.4
107     error_config = {'ecolor': '0.3'}
108
109     bars1 = plt.barh(index, SortedByRating['ratings'][-n:],
110                      bar_width,
111                      alpha=opacity,
112                      color='b',
113                      error_kw=error_config,
114                      label='Rating-avg. ')
115
116     bars2 = plt.barh(index + bar_width, SortedByRating['usage'][-n:],
117                      bar_width,
118                      alpha=opacity,
119                      color='r',
120                      error_kw=error_config,
121                      label='Usage')
122
123
124     plt.title('Beer-rating-keywords')
125     plt.yticks(index + bar_width, SortedByRating['keywords'][-n:])
126     plt.legend()
127     plt.tight_layout()
128     plt.show()
129
130
131 def plotMostUsed(n=10):
132     """
133     Plot a graph of most used keywords in the beer description.
134     :param n: Amount of keywords.
135     """
136
137     fig, ax = plt.subplots()
138     index = np.arange(n)
139
140     bar_width = 0.35
141     opacity = 0.4
142     error_config = {'ecolor': '0.3'}
143
144     bars1 = plt.barh(index, [x*1000 for x in SortedByUsage['ratings'][0:n]],
145                      bar_width,
146                      alpha=opacity,
147                      color='b',
148                      error_kw=error_config,
149                      label='Rating-avg. (x1000)')
150
151     bars2 = plt.barh(index + bar_width, SortedByUsage['usage'][0:n],
152                      bar_width,
153                      alpha=opacity,
154                      color='r',
155                      error_kw=error_config,
156                      label='Usage')
157
158     plt.title('Beer-rating-keywords')
159     plt.yticks(index + bar_width, SortedByUsage['keywords'][0:n])
160     plt.legend()
161     plt.tight_layout()
162     plt.show()
163
164
165 def writeFiles():
166     """Export CSV and text files export."""
167     f = open('../data/sortedByRating.csv', 'wt')
168     writer = csv.writer(f)
169     writer.writerow(('id', 'Keyword', 'Rating', 'Usage'))

```

```

170     for i, word in enumerate(SortedByRating['keywords']):
171         try:
172             writer.writerow((str(i),
173                             word, SortedByRating['ratings'][i],
174                             SortedByRating['usage'][i]))
175         except:
176             pass
177     f.close()
178
179     f = open('../data/sortedByUsage.csv', 'wt')
180     writer = csv.writer(f)
181     writer.writerow(('id', 'Keyword', 'Rating', 'Usage'))
182     for i, word in enumerate(SortedByUsage['keywords']):
183         try:
184             writer.writerow((str(i), word,
185                             SortedByUsage['ratings'][i],
186                             SortedByUsage['ratings'][i]))
187         except:
188             pass
189     f.close()
190
191     # Files for graphic export to http://www.wordle.net/
192     f = open('../data/mostUsedKeywords.txt', 'wt')
193     for i, word in enumerate(SortedByUsage['keywords'][0:100]):
194         f.write(word + " : " + str(SortedByUsage['usage'][i]) + "\n")
195     f.close()
196
197     f = open('../data/bestKeywords.txt', 'wt')
198     for i, word in enumerate(SortedByRating['keywords'][0:100]):
199         f.write(word + " : " + str(SortedByRating['ratings'][i]*1000) + "\n")
200     f.close()
201
202     f = open('../data/worstKeywords.txt', 'wt')
203     for i, word in enumerate(SortedByRating['keywords'][100:]):
204         f.write(word + " : " + str(SortedByRating['ratings'][i]*1000) + "\n")
205     f.close()
206
207     plotBestKeywords(10)
208     plotWorstKeywords(10)
209     plotMostUsed(40)
210
211     writeFiles()
212
213     print 'done'

```

```

1 class dataPoint:
2     def __init__(self, attribs):
3         self.lat = attribs['lat']
4         self.lng = attribs['lng']
5         self.country = attribs['country']
6         self.abv = attribs['abv']
7         self.rating = attribs['rating']
8         self.style = attribs['style']
9
10
11 class dataPoints:
12     def __init__(self, points):
13         self.points = points

```

```

1 import urllib
2 import json
3 from ConfigParser import SafeConfigParser
4 from mpl_toolkits.basemap import Basemap
5 import matplotlib.pyplot as plt
6 import numpy
7
8 config = SafeConfigParser()
9 config.read('../apiConfig.ini')
10 # set the api setting values
11 baseUrl = "https://maps.googleapis.com/maps/api/geocode/json?"
12 apiKey = config.get('googleMaps', 'apiKey')
13 apiKey2 = config.get('googleMaps', 'apiKey2')
14 apiKey3 = config.get('googleMaps', 'apiKey3')
15 apiKey4 = config.get('googleMaps', 'apiKey4')
16
17 abvMapName = {'0': '0', '1': '.1-3.9', '2': '.1-3.9', '3': '.1-3.9',
18              '4': '4.0-4.9', '5': '5.0-5.9', '6': '6.0-6.9', '7': '7.0-7.9',
19              '8': '8.0-8.9', '9': '9.0-9.9', '10': '10.0-10.9', '11': '11+'}
20
21
22 def getLatLong(address, calls):
23     # Google Geocoding api only allows 2500 api calls a day
24     # Update code to handle number of users and api keys
25     url = ''
26     if calls < 2450:

```

```

27     url = baseUrl + "address=%s&sensor=false&key=%s" % (urllib.quote(address.replace('_', '+')), apiKey)
28 elif calls > 2450 and calls < 4950:
29     url = baseUrl + "address=%s&sensor=false&key=%s" % (urllib.quote(address.replace('_', '+')), apiKey2)
30 elif calls > 4950 and calls < 7450:
31     url = baseUrl + "address=%s&sensor=false&key=%s" % (urllib.quote(address.replace('_', '+')), apiKey3)
32 elif calls > 7450 and calls < 9950:
33     url = baseUrl + "address=%s&sensor=false&key=%s" % (urllib.quote(address.replace('_', '+')), apiKey4)
34
35 if (url != ''):
36     data = urllib.urlopen(url).read()
37     info = json.loads(data).get("results")
38     if info:
39         location = info[0].get("geometry").get("location")
40         address_components = info[0].get("address_components")
41         for component in address_components:
42             if "country" in component["types"]:
43                 country = component["short_name"]
44                 print country
45                 location["country"] = country
46         else:
47             location = ""
48         return location
49     else:
50         return "apiLimit"
51
52
53 def createMap(lats, lngs, parallels, meridians, states=False):
54     # create polar stereographic Basemap instance.
55     m = Basemap(projection='stere', lat_0=lats[0], lon_0=lngs[0],
56                 llcrnrlat=lats[1], urcrnrlat=lats[2],
57                 llcrnrlon=lngs[1], urcrnrlon=lngs[2],
58                 rsphere=6371200, resolution='l', area_thresh=10000)
59     # draw coastlines, state and country boundaries, edge of map.
60     m.drawcoastlines()
61     m.drawcountries()
62     if states:
63         m.drawstates()
64
65     # draw parallels.
66     m.drawparallels(parallels, labels=[1, 0, 0, 0], fontsize=10)
67     # draw meridians
68     m.drawmeridians(meridians, labels=[0, 0, 0, 1], fontsize=10)
69
70     return m
71
72
73 def saveMap(param, numPoints, filePrefix):
74     if type(param) is int:
75         if str(param) in abvMapName:
76             abvRange = abvMapName[str(param)]
77         else:
78             abvRange += abvMapName['11']
79
80         title = 'Average_Beer_Ratings_of_Beer_with_Alcohol_Concentration_of_' + \
81             abvRange + '%' + '\n' + str(numPoints) + "_Reviews_Used"
82
83         filename = filePrefix + 'ABV' + abvRange + '.png'
84     else:
85         title = 'Average_Beer_Ratings_of_Beer_with_a_Style_of_' + param + '\n' + \
86             str(numPoints) + "_Reviews_Used"
87         filename = filePrefix + 'Style' + param.replace('_', '') + '.png'
88     plt.suptitle(title)
89     plt.savefig(filename)
90     plt.close()
91
92
93 def drawUSMap(points, param):
94     print "Drawing_US_map_with_" + str(len(points)) + "_data_points."
95     lats = []
96     lngs = []
97     ratings = []
98
99     for point in points:
100         lats.append(point.lat)
101         lngs.append(point.lng)
102         ratings.append(point.rating)
103
104     # US middle, lower left, and upper right
105     # latitude and longitude coordinates
106     usLat = [38, 22, 48]
107     usLng = [-97, -125, -59]
108     parallels = [30, 40]
109     meridians = [280, 270, 260, 250, 240]
110
111     # create polar stereographic Basemap instance.
112     m = createMap(usLat, usLng, parallels, meridians, True)

```

```

113     xs, ys, average = createHistogram(m, lats, lngs, ratings, True)
114
115
116     # overlay the averages histogram over map
117     plt.pcolormesh(xs, ys, average, vmin=0, vmax=5)
118     plt.colorbar(orientation='horizontal')
119     saveMap(param, len(points), '../graphics/US')
120
121
122 def drawEUMap(points, param):
123     print "Drawing EU map with " + str(len(points)) + " data points."
124     lats = []
125     lngs = []
126     ratings = []
127
128     for point in points:
129         lats.append(point.lat)
130         lngs.append(point.lng)
131         ratings.append(point.rating)
132
133     # EU middle, lower left, and upper right
134     # latitude and longitude coordinates
135     euLat = [51, 27, 71]
136     euLng = [20, -16, 45]
137     parallels = [30, 40, 50, 60, 70]
138     meridians = [350, 0, 10, 20, 30, 40]
139
140     # create polar stereographic Basemap instance.
141     m = createMap(euLat, euLng, parallels, meridians)
142     xs, ys, average = createHistogram(m, lats, lngs, ratings)
143     #####
144     # xs, ys, average data will be stored in db as lists. Need to
145     # change back to ndarray later
146     # xs = numpy.array(xs.tolist())
147     # ys = numpy.array(ys.tolist())
148     # average = numpy.array(average.tolist())
149     #####
150
151     # overlay the averages histogram over map
152     plt.pcolormesh(xs, ys, average, vmin=0, vmax=5)
153     plt.colorbar(orientation='horizontal')
154     saveMap(param, len(points), '../graphics/EU')
155
156
157 def inEU(lat, lng):
158     return lat >= 27 and lat <= 71 and lng >= -16 and lng <= 45
159
160
161 def inUS(lat, lng):
162     return lat >= 22 and lat <= 48 and lng >= -125 and lng <= -59
163
164
165 def createHistogram(m, lats, lngs, ratings, us=False):
166     nx, ny = 10, 10
167     if us:
168         nx, ny = 20, 20
169     # compute appropriate bins to histogram the data into
170     lng_bins = numpy.linspace(min(lngs), max(lngs), nx + 1)
171     lat_bins = numpy.linspace(min(lats), max(lats), ny + 1)
172
173     # Histogram the lats and lngs to produce an array of frequencies in each box.
174     frequency, _, _ = numpy.histogram2d(lats, lngs, [lat_bins, lng_bins])
175
176     # Histogram the lats and lngs to produce an array of frequencies weighted by
177     # ratings in each box.
178     weighted, _, _ = numpy.histogram2d(lats, lngs, [lat_bins, lng_bins], weights=ratings)
179
180     # divide the weighted bins by the frequency bins to create bins of average
181     # beer ratings in each bin
182     with numpy.errstate(invalid='ignore'):
183         average = numpy.divide(weighted, frequency)
184         average = numpy.nan_to_num(average)
185
186     # Turn the lng/lat bins into 2 dimensional arrays ready
187     # for conversion into projected coordinates
188     lng_bins_2d, lat_bins_2d = numpy.meshgrid(lng_bins, lat_bins)
189
190     # convert the xs and ys to map coordinates
191     xs, ys = m(lng_bins_2d, lat_bins_2d)
192
193     return xs, ys, average
194
195
196 def drawMap(points, param):
197     euPoints = []
198     usPoints = []

```

```

199
200     for point in points:
201         if inUS(point.lat, point.lng):
202             usPoints.append(point)
203         elif inEU(point.lat, point.lng):
204             euPoints.append(point)
205     drawEUMap(euPoints, param)
206     drawUSMap(usPoints, param)

```

```

1  """
2  Download, process label images for all beers.
3
4  * Assign N most used colors based on K-Means clustering algorithm
5  * Classify these colors to fit the palette
6  * Rate these palette colors based on previously downloaded UNTAPPD beer ratings.
7  """
8
9  import os
10 import numpy as np
11 import numpy.ma as ma # Masked array
12 import matplotlib.pyplot as plt
13 import matplotlib.cm as cm # Color map
14 from sklearn.cluster import KMeans
15 from scipy import misc
16 from math import sqrt
17 import requests
18 import jsonpickle as jpickle
19 import csv
20
21
22 class Image:
23
24     """Object to manipulate image data and cluster the colors."""
25
26     def __init__(self, imgFilePath):
27         self.filePath = imgFilePath
28         self.raw_data = misc.imread(imgFilePath)
29
30         # Normalizing RGB to be in the range [0-1]. Assuming 8bit integer coding.
31         self.original_data = np.array(self.raw_data, dtype=np.float64) / 255
32         #self.original_data = np.array(self.raw_data, dtype=np.float64)
33
34         # Transform to a 2D numpy array [100x100px - no need to crop].
35         self.w, self.h, self.d = self.shape = tuple(self.raw_data.shape)
36         self.shaped1d = (self.w, self.h, 1)
37         assert self.d == 3 # [R,G,B] tuple
38
39         # Arrays for further manipulation
40         self.pixels = np.reshape(self.original_data, (self.w * self.h, self.d))
41         self.kmeans = None
42         self.nColors = 0
43         self.mask = Mask(self)
44
45         # Convert to grayscale for analysis (cropping etc.)
46         self.grayscale = np.zeros(self.shaped1d)
47         for i, row in enumerate(self.original_data):
48             for j, color in enumerate(row):
49                 r, g, b = tuple(color)
50                 # Uses luminosity method - better match with human perception
51                 self.grayscale[i][j] = 0.21 * r + 0.72 * g + 0.07 * b
52
53     def preprocess(self):
54         """Prepare image for the clustering."""
55         # Check if the picture has white background [sample 5x5], otherwise do not crop
56         offsetFromWhite = sqrt(sum(np.array([x * x for x in (1 - self.original_data[0:5, 0:5])]).flatten()))
57         if offsetFromWhite > 0.05:
58             return
59
60         # Cropping based on luminiscence derivative
61         # looks for the first jump from the sides and makes flags for generating mask
62         lightDifference = np.zeros(self.shaped1d)
63         for r, row in enumerate(self.grayscale):
64             borderDetected = False
65             for c, col in enumerate(row):
66                 if c < (self.w - 1):
67                     diff = self.grayscale[r][c + 1][0] - self.grayscale[r][c][0]
68                     lightDifference[r, c] = diff
69
70                 # Border detection from the left side
71                 if abs(diff) > 0.05 and borderDetected == False:
72                     self.mask.boundaries[r, c] = 1
73                     borderDetected = True
74
75                 # Border detection from the right side
76                 c = self.w - 1
77                 borderDetected = False

```



```

78         while c > 0:
79             diff = lightDifference[r, c]
80             if abs(diff) > 0.05 and borderDetected == False:
81                 self.mask.boundaries[r, c] = 1
82                 borderDetected = True
83             c -= 1
84
85     self.mask.genMatrix()
86
87     def clusterize(self, n):
88         """Cluster the image into [n] of RGB Clusters using Scikit_KMeans class."""
89         self.nColors = n
90
91         # Apply cropping mask if exists
92         try:
93             pixels = np.reshape(ma.masked_array(self.pixels, mask=self.mask.matrix), (self.w * self.h, self.d))
94         except:
95             pixels = self.pixels
96
97         # Calculate the clusters
98         self.kmeans = KMeans(init='k-means++', n_clusters=n, random_state=0).fit(pixels)
99
100        return BeerColor(self.kmeans)
101
102    def quantizeImage(self):
103        """Plot the image using only clustered colors."""
104        codeBook = self.kmeans.cluster_centers_
105
106        # Determine to which cluster the pixel belongs
107        labels = self.kmeans.predict(self.pixels)
108
109        newImage = np.zeros((self.w, self.h, self.d))
110        label_idx = 0
111        for i in range(self.w):
112            for j in range(self.h):
113                newImage[i][j] = codeBook[labels[label_idx]]
114                label_idx += 1
115        self.pixels = newImage
116
117    def showResults(self):
118        """Plot the original picture, processed picture and clustered colors."""
119        plt.figure(1)
120        plt.clf()
121
122        plt.subplot(2, 2, 1)
123        plt.title('Original')
124
125        plt.imshow(self.original_data)
126        plt.axis('scaled')
127
128        plt.subplot(2, 2, 2)
129        plt.title('Quantized')
130        plt.imshow(self.pixels)
131        plt.axis('scaled')
132
133        plt.subplot(2, 2, 3)
134        plt.title('Mask')
135        plt.imshow(self.mask.matrix)
136        plt.axis('scaled')
137
138        plt.subplot(2, 2, 4)
139        plt.title('Cluster colors')
140        for i, color in enumerate(self.kmeans.cluster_centers_):
141            rectangleHeight = self.h / self.nColors
142            rectangleWidth = rectangleHeight
143            rectangle = plt.Rectangle((i * rectangleWidth, 0), rectangleWidth, rectangleHeight, fc=color)
144            plt.gca().add_patch(rectangle)
145        plt.axis('scaled')
146        plt.show()
147
148
149    class Mask:
150
151        """Object to handle non-rectangular color-difference based cropping."""
152
153        def __init__(self, img):
154            self.w, self.h, self.d = self.shape = img.shape
155            self.boundaries = np.zeros(self.shape)
156            self.matrix = np.zeros(self.shape)
157
158        def genMatrix(self):
159            """Generate Numpy matrix mask matrix based on the boundaries."""
160            for r, row in enumerate(self.boundaries):
161                # From the left
162                c = 0
163                while row[c][0] == 0 and c < self.w - 1:

```

```

164         self.matrix[r][c] = (1, 1, 1)
165         c += 1
166     # From the right
167     c = self.w - 1
168     while row[c][0] == 0 and c > 0:
169         self.matrix[r][c] = (1, 1, 1)
170         c -= 1
171
172
173 class ColorPalette:
174
175     """Object to create color palette from already processed beer labels."""
176
177     def __init__(self):
178         # self.nColors = nPaletteColors
179         self.palette = dict()
180
181     def build(self, beerColorsDict, beersList):
182         """
183         Generate color rating palette based on beer ratings.
184
185         Palette colors correspond to those on the web color picker.
186         """
187         print 'Generating the color palette ...'
188
189         webPaletteRGB = [[254, 82, 9],
190                         [251, 153, 2],
191                         [247, 189, 1],
192                         [255, 254, 53],
193                         [209, 233, 51],
194                         [102, 177, 49],
195                         [2, 145, 205],
196                         [9, 68, 253],
197                         [63, 1, 164],
198                         [134, 2, 172],
199                         [168, 24, 75],
200                         [254, 38, 18],
201                         [255, 255, 255],
202                         [0, 0, 0]]
203
204         # Normalize
205         webPaletteRGB = (np.array(webPaletteRGB, dtype=np.float64)/255).tolist()
206
207         # Construct the Palette
208         for i in range(len(webPaletteRGB)):
209             paletteColor = self.palette[i] = dict()
210             paletteColor['RGB'] = webPaletteRGB[i]
211             paletteColor['RatingSum'] = 0
212             paletteColor['Votes'] = 0
213             paletteColor['Rating'] = 0
214
215         progress = Progress(max=len(beerColorsDict), msg="Rating the colors from palette ...")
216         for bid, beerColor in beerColorsDict.iteritems():
217             beer = getBeer(bid, beersList)
218             if beer:
219                 for colorId, beerColorValue in enumerate(beerColor.colors):
220                     closestPaletteColorId = self.classifyColor(beerColorValue)
221
222                     # Update classification flags
223                     beerColor.colorPaletteFlags[colorId] = closestPaletteColorId
224
225                     # Update color rating
226                     self.palette[closestPaletteColorId]['RatingSum'] += beer.rating
227                     self.palette[closestPaletteColorId]['Votes'] += 1
228             else:
229                 print 'Not found bid_' + bid
230
231         # Show the classified color.
232         # plt.figure(1)
233         # plt.title(' Palette colors ')
234         # color = [beerColor.colors[colorId], self.palette[closestPaletteColorId]['RGB']]
235         # for i in range(2):
236         #     rectangleHeight = 20
237         #     rectangleWidth = 20
238         #     rectangle = plt.Rectangle((i * rectangleWidth, 0), rectangleWidth, rectangleHeight, fc=color[i])
239         #     plt.gca().add_patch(rectangle)
240         # plt.axis('scaled')
241         # plt.show()
242
243         progress.tick()
244
245         for color in self.palette.values():
246             if color['Votes'] != 0:
247                 color['Rating'] = color['RatingSum']/color['Votes']
248
249

```

```

250 def classifyColor(self, inputColor):
251     """Custom classification of colors to fit the palette."""
252     closestDistance = sqrt(3)
253     inputColorYUV = RGBtoYUV(inputColor)
254     for paletteColorId, paletteColor in self.palette.iteritems():
255         # Find maximum Euclidean distance of linear colorspace (YUV) values
256         paletteColorYUV = RGBtoYUV(paletteColor['RGB'])
257         distance = sqrt((paletteColorYUV[0] - inputColorYUV[0])**2 +
258                         (paletteColorYUV[1] - inputColorYUV[1])**2 +
259                         (paletteColorYUV[2] - inputColorYUV[2])**2)
260         if distance < closestDistance:
261             closestDistance = distance
262             closestPaletteColorId = paletteColorId
263
264     return closestPaletteColorId
265
266 def readFromFile(self, path):
267     """Get the palette from file instead of building it again."""
268     try:
269         paletteFile = open(path, 'rb')
270     except IOError:
271         paletteFile = open(path, 'wb')
272
273     try:
274         f = paletteFile.read()
275         loadedColorPalette = jpickle.decode(f)
276     except:
277         print "Palette file corrupted."
278         return 0
279     paletteFile.close()
280
281     self.palette = loadedColorPalette
282     return 1
283
284 def genCSV(self):
285     """Save the csv file of the color palette."""
286     f = open('../data/colorPalette.csv', 'wt')
287     writer = csv.writer(f)
288     writer.writerow(('id', 'HEX', 'Rating', 'Votes'))
289     for id, item in self.palette.iteritems():
290         try:
291             writer.writerow((id, rgbToHex(tuple(np.array(item['RGB'])*255)), item['Rating'], item['Votes']))
292         except:
293             pass
294     f.close()
295
296
297 class BeerColor:
298     """Object to save dominant colors of beer along with the color palette flags."""
299
300     def __init__(self, kmeans):
301         self.colors = kmeans.cluster_centers_.tolist() # array of RGB values
302         self.presence = [(float(sum(kmeans.labels_ == i)) / kmeans.labels_.size)
303                          for i in range(len(kmeans.cluster_centers_))]
304         self.colorPaletteFlags = [0] * len(self.colors)
305
306
307 class BeerColorsDict(dict):
308
309     """Container to hold colors of each beer."""
310
311     def __init__(self, *arg, **kw):
312         super(BeerColorsDict, self).__init__(*arg, **kw)
313
314     def getColors(self):
315         """Return non-nested color RGB values for K-Means processing."""
316         return np.concatenate([x for x in [i.colors for i in self.values()]])
317
318
319 class Progress:
320
321     """Print percentage of done work."""
322
323     def __init__(self, max, msg="Processing...", freq=100):
324         """Construct the object, determine frequency of printing."""
325         self.max = max
326         self.msg = msg
327         self.printFrequency = max/freq # Each percent
328         self.i = 0
329
330     def tick(self):
331         """Call in every cycle."""
332         if (self.i % self.printFrequency) == 0:
333             print self.msg + str(100*self.i/self.max) + "%"
334         if self.i == self.max:

```

```

336         print "Done."
337         self.i += 1
338
339
340     def download(beersList, imgPath, fileList):
341         """Get the beer labels based on the Untapped beer list."""
342         progress = Progress(max=len(beersList), msg="Downloading images...")
343         for hashId, beer in beersList.iteritems():
344             url = beer.label
345             if url and (url != 'https://d1e8v1qci5en44.cloudfront.net/site/assets/images/temp/badge-beer-default.png'):
346                 fileType = url.split("/")[-1].split(".")[1]
347                 filePath = imgPath + str(beer.bid) + '.' + fileType
348                 fileName = str(beer.bid) + '.' + fileType
349                 if fileName not in fileList:
350                     r = requests.get(url, stream=True)
351                     if r.status_code == 200:
352                         with open(filePath, 'wb') as f:
353                             for chunk in r.iter_content(1024):
354                                 f.write(chunk)
355                 progress.tick()
356
357
358     def getBeer(bid, beersList):
359         """Search for beer based on bid."""
360         for beer in beersList.values():
361             if beer.bid == bid:
362                 return beer
363         return None
364
365
366     def RGBtoYUV(RGB):
367         """Convert array of RGB values to YUV colorspace."""
368         T = np.matrix('0.299 0.587 0.114; '
369                       '-0.14713 -0.28886 0.436; '
370                       '0.615 -0.51499 -0.10001')
371         try:
372             return T*np.transpose(np.matrix(RGB))
373         except:
374             print "Unable to convert RGB->YUV."
375             return 0
376
377     def rgbToHex(rgb):
378         """Convert RGB values (in list) to its HEX equivalent."""
379         return '%02x%02x%02x' % rgb
380
381     # For database export
382     # pal = ColorPalette()
383     # pal.readFromFile('../data/colorPalette.json')
384     # pal.genCSV()

```

```

1     """
2     Functions to load/downloaded json data.
3
4     To do: Make this general.
5     """
6
7     import jsonpickle as jpickle
8     import csv
9     import labels
10
11
12     def readUsers():
13         """Load already processed users UntappdUser."""
14         try:
15             usersFile = open('../data/users.json', 'rb')
16         except IOError:
17             usersFile = open('../data/users.json', 'wb')
18
19         try:
20             f = usersFile.read()
21             usersList = jpickle.decode(f)
22         except:
23             usersList = {}
24         usersFile.close()
25         return usersList
26
27
28     def readBeers():
29         """Load already processed beers UntappdBeer."""
30         try:
31             beersFile = open('../data/beers.json', 'rb')
32         except IOError:
33             beersFile = open('../data/beers.json', 'wb')
34
35         try:
36             f = beersFile.read()

```

```

37     beersList = jpickle.decode(f)
38 except:
39     beersList = {}
40 beersFile.close()
41 return beersList
42
43
44 def readBreweries():
45     """Load_already_processed_breweries_UntappdBrewery."""
46     try:
47         breweriesFile = open('../data/breweries.json', 'rb')
48     except IOError:
49         breweriesFile = open('../data/breweries.json', 'wb')
50
51     try:
52         f = breweriesFile.read()
53         breweryList = jpickle.decode(f)
54     except:
55         breweryList = {}
56     breweriesFile.close()
57     return breweryList
58
59
60 def readBreweryToBeers():
61     """Load_already_processed_breweries_dictionary."""
62     try:
63         breweryToBeersFile = open('../data/breweryToBeers.json', 'rb')
64     except IOError:
65         breweryToBeersFile = open('../data/breweryToBeers.json', 'wb')
66
67     try:
68         f = breweryToBeersFile.read()
69         breweryToBeers = jpickle.decode(f)
70     except:
71         breweryToBeers = {}
72     breweryToBeersFile.close()
73     return breweryToBeers
74
75
76 def readDataPoints():
77     """Load_dataPoints."""
78     try:
79         dataPointsFile = open('../data/dataPoints.json', 'rb')
80     except IOError:
81         dataPointsFile = open('../data/dataPoints.json', 'wb')
82
83     try:
84         f = dataPointsFile.read()
85         dataPoints = jpickle.decode(f).points
86     except:
87         dataPoints = []
88     dataPointsFile.close()
89
90     return dataPoints
91
92
93 def readBeerStyles():
94     """Load_most_rated_beer_styles."""
95     styles = []
96     with open('../data/styles.csv') as stylesFile:
97         reader = csv.DictReader(stylesFile)
98         for row in reader:
99             if row['style'] not in styles:
100                 styles.append(row['style'])
101     return styles
102
103
104 def readBeerColors():
105     """Load_the_dominant_label_colors."""
106     try:
107         beerColorsFile = open('../data/beerColors.json', 'rb')
108     except IOError:
109         beerColorsFile = open('../data/beerColors.json', 'wb')
110
111     try:
112         f = beerColorsFile.read()
113         beerColorsDict = jpickle.decode(f)
114     except:
115         beerColorsDict = labels.BeerColorsDict()
116     beerColorsFile.close()
117     return beerColorsDict

```

```

1 """
2 Single-purpose_script_for_easy_monitoring_of_data_quantity.
3
4 Load_each_json_data_file, find_its_size_and_generate

```

```

5  a_plot_for_presentation.
6  """
7
8  import fileReader as files
9  import matplotlib.pyplot as plt
10 import os
11 import numpy as np
12
13 # Load files
14 print "Loading_beers..."
15 beersList = files.readBeers()
16 print "Loading_users..."
17 usersList = files.readUsers()
18 print "Loading_breweries..."
19 breweriesList = files.readBreweries()
20
21 # Path for saving the images
22 path = "../data/labels/"
23 fileList = os.listdir(path)
24
25 # Data gathering
26 labels = ('Beers', 'Reviews', 'Users', 'Breweries', 'Labels')
27 index = np.arange(len(labels))
28 quantities = (len(beersList), sum([len(x.ratings) for x in usersList.values()]),
29               len(usersList), len(breweriesList), len(fileList))
30
31 # Plot the quantities
32 plt.figure(1)
33 bar_width = 0.35
34 opacity = 0.7
35
36 bars = plt.bar(index, quantities,
37                bar_width,
38                alpha=opacity,
39                color='g')
40
41 plt.xticks(index + bar_width/2, labels)
42 plt.title('Amount_of_mined_data')
43 plt.ylabel('log(N)')
44 plt.yscale('log')
45 plt.grid()
46
47 def autoLabel(bars):
48     # attach some text labels
49     for rect in bars:
50         height = rect.get_height()
51         plt.text(rect.get_x()+rect.get_width()/2., 1.05*height, '%d'%int(height),
52                 ha='center', va='bottom')
53
54 autoLabel(bars)
55 plt.show()

```

APPENDIX B

AUTOMATIC GENERATION OF DOCUMENTATION

Demonstration using epydoc:

```
epydoc --pdf -o /home/fnielsen/tmp/epydoc/ --name RBBase wikipedia/api.py
```

This example does not use `brede_str_nmf` but another more well-documented module called `api.py` that are used to download material from Wikipedia.