Self-marketing Approaches

An Analysis of Online Dating

Li Liu, Abhishek Pandit, Adam Shelton

12/11/2019

Contents

Contributions

Liu	**Pandit**	**Shelton**
KMeans	Overall Strategy	EDA
Structural Topic Modeling	Data Cleaning	AGNES
Volunteer for the App Demo	Topic Modeling Selection & Doc2Vec Model	DBSCAN

Introduction

"So tell me about yourself!" This seemingly straightforward question in day-to-day interactions is usually met with silence and hesitation. That can no longer be the case for the 1.67 trillion online dating industry, which has grown exponentially in popularity over the last decade. The dating apps, such as OkCupid and Coffee Meets Bagel, are designed to help the singles 'get to know' other people for short or long-term romantic relationships. In order to be popular and memorable, users usually have to write a short introduction to advertise themselves. Such activity could be regarded as self-marketing. As the users of dating apps come from diverse backgrounds, we are interested in how users from distinct backgrounds take different approaches to make themselves more memorable. Moreover, we design the framework of scoring users' self-introduction and algorithm for providing writing tips (such as words for being memorable). Although our project is still preliminary, it has gained a lot of interest among our friends who struggle to find a date online. Also, our methods and analysis have the potential to be adopted by the dating website to improve the users' experience and better achieve their mission as matchmakers.

Literature Review

Self-concept and self-representation have long served as grounds of debate in cognitive and positive psychology (Bruning, Schraw, and Ronning 1999) as well as social anthropology (Goffman 1975). The recent spread of social networking and its specific affordances have allowed individuals to build different online 'selves' (Papacharissi 2010). One such critical scenario may be that of mate selection, which several economists and sociologists have likened this to 'marriage marketplace' (Hitsch, Hortaçsu, and Ariely 2010). Several online dating service providers in developed countries may facilitate the expansion of potential mates beyond the limits of even extended offline social networks Cacioppo et al. (2013) assert that as many as one in three marriages in the United States is facilitated through these portals. Heino, Ellison, and Gibbs (2010) argue that these avenues further entrench the economic dimension through an acute, implicit awareness of 'relationshopping'. Herein, potential partners are reduced to entries in a catalog to be scrolled through. In this sense, they suggest an emerging conscientiousness of 'marketing', with the product being themselves, and the potential mate assuming the role of a buyer (ibid). This perception thus links the private worlds of romantic intimacy with those of mass consumption and broader perceived appeal to the opposite sex.

Potentially, we will also use some marketing theories to understand our findings. Selling themselves and finding a mate on OkCupid is not very different from selling a product on eBay. Economists have been

interested in the matching problem of demand and supply, such as Hitsch, Hortaçsu, and Ariely (2010). Since we do not have data on users' interactions, we will focus primarily on understanding how people brand themselves to stand out in a crowd. For example, brand awareness is a key metric in marketing to quantify the degree to which people recall or recognize a brand. A high level of brand awareness helps a product stand out and get chosen when consumers face many alternatives.

This could be applied to understand online dating. Let us imagine your future mate uses the filter to narrow down the consideration sets. He/She might still face many similar choices with high matching scores to choose from. If you want to stand out from the pool, you must make yourself memorable by highlighting the uniqueness. Thus, one possible idea in this project is to explore and understand how users could increase their brand awareness and differentiate themselves in their segments

Empirical Strategy

This study will leverage publicly open and anonymized user profile data for 59,946 users of OkCupid within a 25 mile radius of San Francisco that were extracted with permission (Kim & Escobedo-Land, 2012). These will then be harnessed to address questions of self-representation in the essay section specifically for male users. To ensure that these were active users, profiles were accepted into dataset only if they had been members as of 06/26/2012, had been active in the previous year, and had at least one photo in their profile. The data set includes "typical user information, lifestyle variables, and text responses to 10 essay questions. After their submission, the data were then scraped but with anonymization and removal of personally identifiable information for legal and privacy concerns. We dropped all female users, and male users with missing entries for any of the 4 key variables or essays. This reduced the total sample size to 18330.

Like with several products and services, the user (who is the product in this case) must gain more clarity on three key dimensions of this dating marketplace: 1) Who his closest competitors are: We will aim to develop 'clusters' of similar users. The clustering could occur either on: i) Demographic Variables- Check on categorical variables using Gower's distance ii) Text Variables- First by building a Doc2Vec model In either case, a Principal Components Analysis will most likely be necessary for dimensionality reduction and being able to visually verify the presence of clusters.

2) What strategies are the competitors applying- In the absence of photos, the core 'strategies' available are twofold-

a) What other men talk about- the content of their essay: In terms of our machine learning toolbox, the

best approach appears to be that of Topic Modelling. Such an exploration would look for the topics themselves, and what proportions of user profiles are devoted to different topics

The profile may also benefit in terms of memorability from low frequency words, which a Topic Model would typically ignore b) How they take about their content: This includes a number of aspects, including, but not limited: i) Complexity of the Language (measurable by Flesch Index) ii) Vocabulary Level (measurable by proportion of long words- with at least 3 syllables) iii) Humour- we will need to develop a proxy for this variable. iv) Length of the profile- this is directly measurable by the number of words

3) How to stand out from the competition This would require a clear understanding of the previous steps, as well as a similarity score. Such a score could be built using:

a) cosine Similarity in the DOc2Vec Model(purely text), which could then be subsetted using demographic information represented by the cluster
b) Check for topic distributions in their demographic and recommend deviations. This would be enabled (at least in part) by Structural Topic Modelling

As would evident from above, there are 4 major methods we will be harnessing: 1) Doc2Vec, Clustering and Topic Modeling on Text Variables 2) Clustering on Demographic Variables 3) Combining Demographic and Text Variables through Structural Topic Models

In particular, we will use several key demographic variables and the self-introduction text. Also, we currently focus on only male users.This is for two reasons: 1) Women in the US typically accord 50% higher importance to their suitors' descriptions relative to men (who place greater weight on the photos) (Emory, 2017) 2) Most American dating sites (with a few exceptions) feature a higher proportion of male users, thus resulting in higher competition (Thottam, 2019)

Our preparations proceed in three steps:

First, we fine tune our demographic variables. Besides the choice of 'male' for gender (and 'straight' for orientation, since our focus is on heterosexual users), 12 other variables with multiple levels are available. Choice will need to be guided by theory. Our key statistical assumption is that difference in the text are CAUSED by differences in demographics. In this sense, we would seek to consider those demographics whose values may lead men to alter their choice of language. Through research, we found these to be:

1) Education: Through perceived improvements in earning power as well as use of language (Torkey, 2018), (Fiore et al, 2015), (Stevens & Schaefer, 1990)

2) Height: Shepperd & Strattman (1989) OkCupid (2010)- More biological preference for taller men. Men also overstate their height on average. Using the 1st quartile as a cutoff, we split this group into short and not short.
3) Race: Lindqvist and Lin, 2010- Online preferences veer towards homophily within races
4) Fitness Level: Burke, 2019- Based on their choice of photos, it seems that men (if not the women themselves) consider physique to be an importnant determinant of attractiveness

Secondly, we use topic modeling (LDA) to visualize the latent topics behind the text. We also compare the differences in topic proportions for different groups and clusters of users. We attempt this with both Non-Negative Matrix Factorization (NMF) and Latent Dirichlet Allocation, given its connections with other tools such as Structural Topic Models

Thirdly, we use the structural topic modeling to estimate the impact of the demographic variables and the humor measure (DBScan on Doc2Vec vectors) on the topic proportions.

Lastly, we discuss the potential use of our analysis and also acknowledge the challenges and limitations we face at this stage.

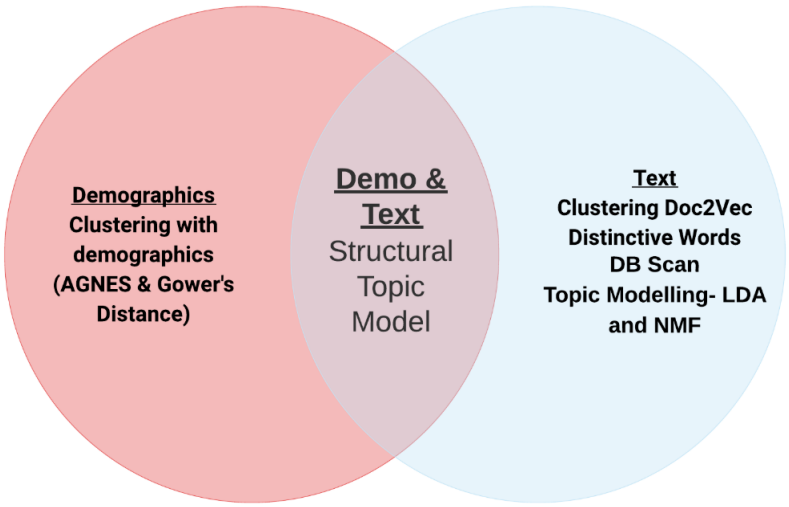


Figure 1: Venn Diagram of Methods

Firstly, we use different cluster analysis to group the users. We would expect users with similar backgrounds to introduce themselves in a similar manner. Since the demographic variables are mostly categorical with many levels and we use the Gowler distance, including them all would be computationally impossible. Instead, we rely on the literature on selecting the four most interesting variables and also apply other techniques to generate new features from the self-introductions.

Table 1: Continuous Variables

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
age	0	1	32.02	9.09	18.0	26.00	30.00	36.00	69
height	0	1	70.51	3.03	3.0	69.00	70.00	72.00	95
long_words	0	1	11.33	13.28	0.0	3.00	8.00	15.00	446
flesch	0	1	7.30	4.75	-3.6	4.86	6.73	8.96	268
profile_length	0	1	117.84	122.21	1.0	43.00	85.00	153.00	2973
prop_longwords	0	1	0.10	0.08	0.0	0.06	0.09	0.12	3

Table 2: Other Variables

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
clean_text	0	1	1	16952	0	18790	0
essay9	0	1	1	10849	0	18125	0
edu	0	1	7	21	0	3	0
fit	0	1	3	7	0	3	0
race_ethnicity	0	1	5	8	0	6	0
height_group	0	1	5	9	0	2	0

Secondly, we use topic modeling (LDA) to visualize the latent topics behind the text. We also compare the differences in topic propotions for different groups and clusters of users.

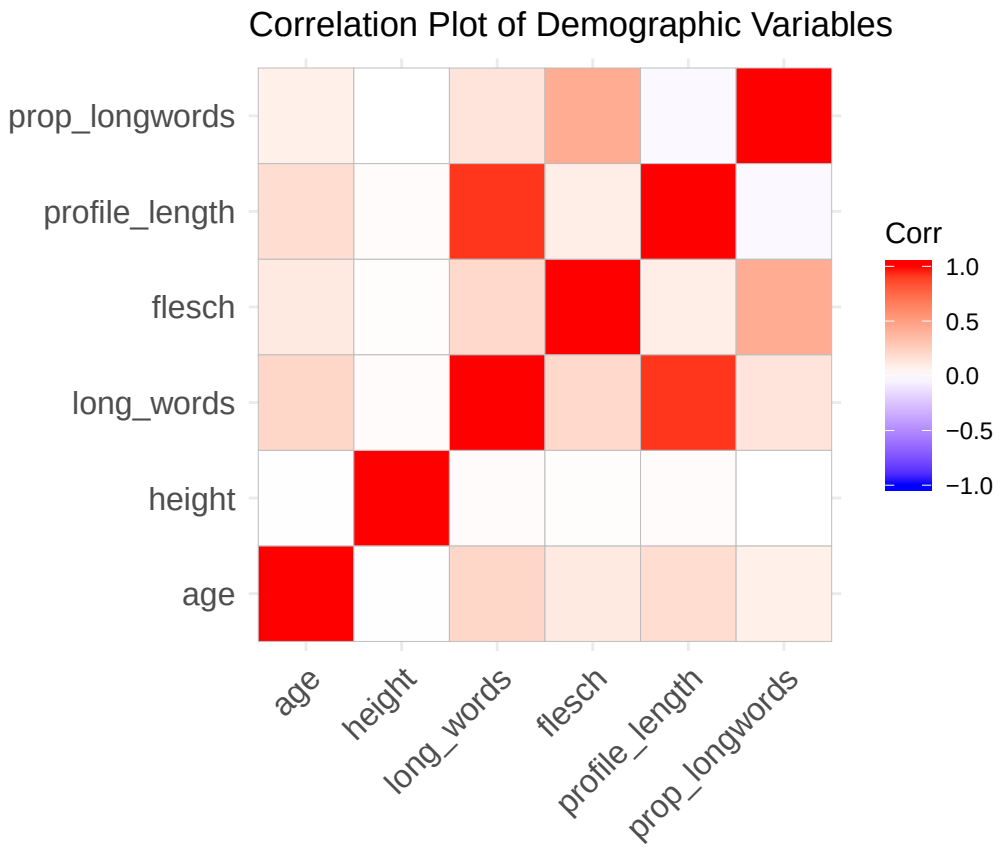
Thirdly, we use the structural topic modeling to estimate the impact of the demographic variables and humor measure (DBScan on Doc2Vec vectors) on the topic proportions.

Lastly, we discuss the potential use of our analysis and also acknowledge the challenges and limitations we face at this stage.

Analysis & Results

Exploratory Data Analysis

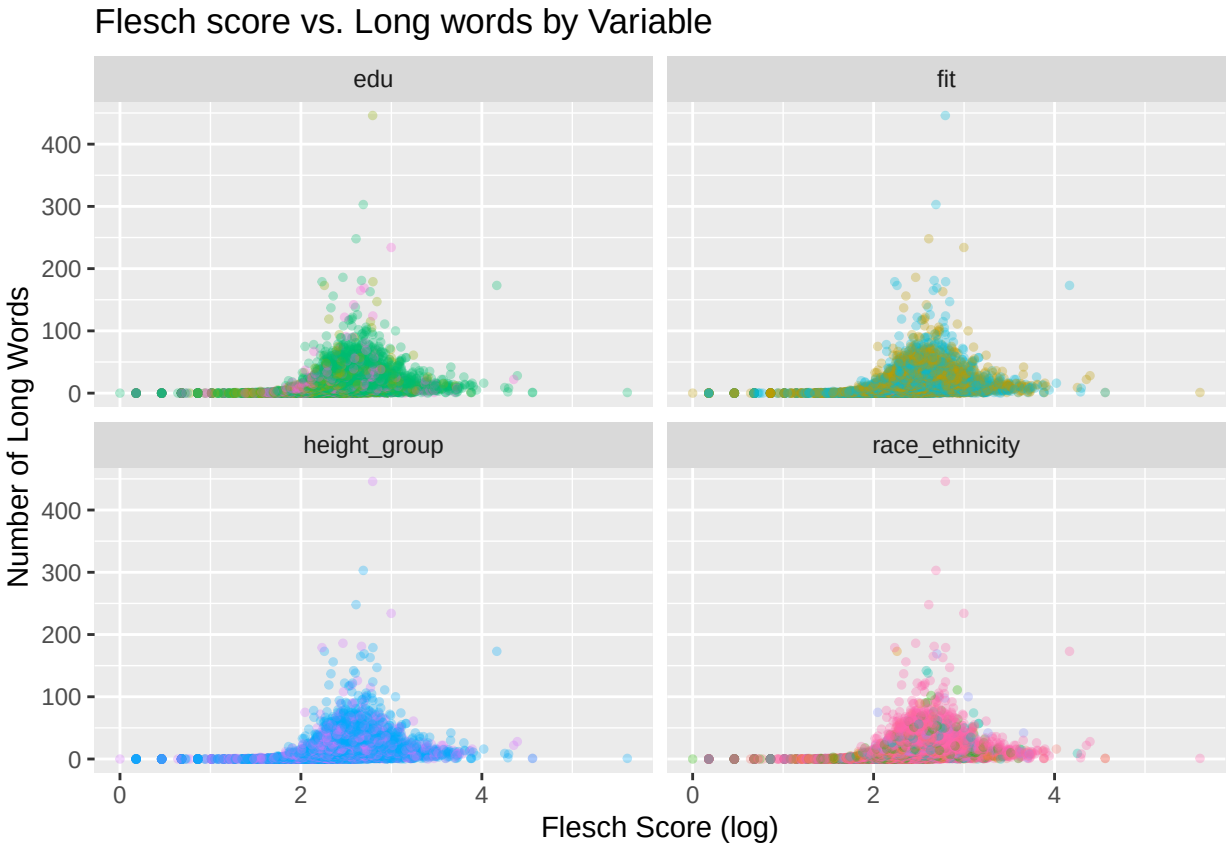
Descriptive Statistics



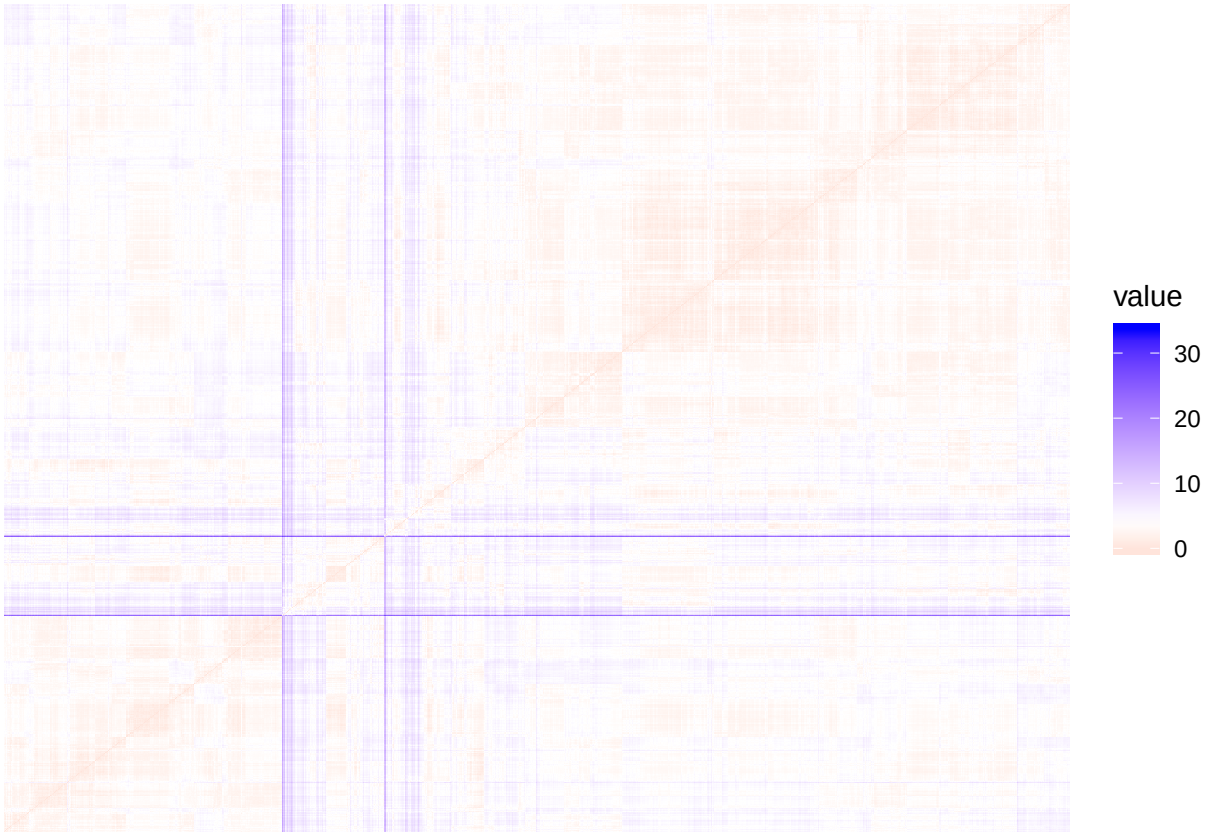
The data we used was approximately 60,000 anonymous OkCupid profiles from 2012 that were gathered with consent from users in the San Francisco area (Kim and Escobedo-Land 2015). This data was downloaded from the GitHub page for Kim and Escobedo-Land (2015), https://github.com/rudeboybert/JSE_OkCupid. The data contains demographic attributes of users that were submitted to their profile, including variables like age, height, race, and education, in addition to a selection of ten short essays that users have written in response to different prompts to display on their profiles. We subsetted this data to 18817 profiles of men, and generated additional features for the numbers and proportions of long words and Flesch–Kincaid readability scores of the main profile essay. The majority of male users in our sample or white, fit, not-short, and have more than a high school education. The mean reported age is 32 and the mean reported height is 70.5 inches (approximately 5 foot 9 inches).

The majority of the variables included in the demographic data are independent, but some weaker correlations

do exist. As expected there are positive correlations between all the features generated from the essay text. Age is also positively correlated with our text-generated features, perhaps suggesting that older people are more educated and write with more complexity. While Flesch scores and the amount of long words are correlated, there do not appear to be any demographic interactions with that relationship.


Flesch score vs. Long words by Variable
edu
fit
height_group
race_ethnicity
Number of Long Words
0
100
200
300
400
0
2
4
Flesch Score (log)

Clusterability



The demographic data is highly clusterable with a Hopkins Statistic of 0.929 on a random subset of 2,000 observations. The ordered dissimilarity image not only provides visual evidence of this clusterability, but also gives some indication of how many clusters may be in this data. Unfortuantely, clusterability could not be determined for the whole dataset due to performance limitations. However, 2,000 observations should be sufficient for determining clusterability.

Clustering of Demographic Data

K-means

Kmeans is a type of partitioning clustering algorithm to maximize intra-cluster homogeneity and maximize inter-cluster heterogeneity. We use four demographic variables: 'fit', 'education', 'height_group', 'race_ethnicity' as previous research shows evidence these four variables are associated with users' different behavior in online self-introductions. We use Gower's distance as these variables are categorical. We start with 3 clusters and plot with the `fviz_cluster` function.

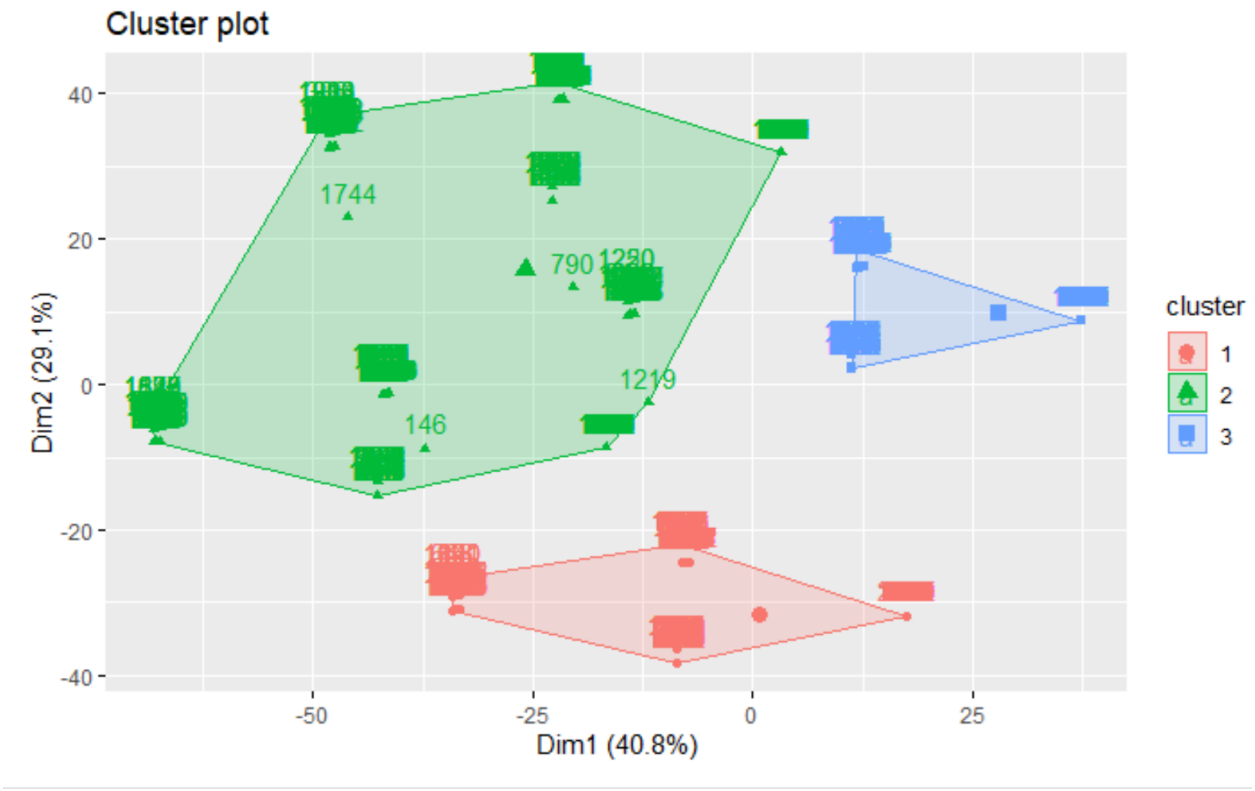


Figure 2: Kmeans with 3 clusters

Then we find the optimal number of clusters by comparing the metric of average silhouette width. However, the metric does not converge and keeps growing with more clusters. As a result, kmeans is not the best for finding the optimal demographic clusters.

AGNES

Agglomerative Nesting was used to cluster the demographic data with a bottom-up approach to contrast the K-means clustering. As an AGNES model would not complete on the full data-set, a random subset of 5,000 observations was used instead. This AGNES model used a Gower's dissimilarity matrix of the data to aid in the clustering of categorical variables, since most of the demographic variables are categorical. Using the `NbClust` package in R, we determined that the optimal number of clusters was two, when using the `ward.D2` method to match our AGNES model. The model gives us two well-defined clusters along the first two principal components. The defining factor between these two clusters, majorly appears to be height, which, per the Wilcox Test, has a statistically significant difference between the means of the two clusters by about five inches. These AGNES clusters can be used to reduce demensionality among a user's demographics so it can be more effectively used in further analyses. The dendrogram and cluster plots were generated using the `factoextra` package in R.

Text Analysis

Word2Vec

Topic Modeling

A topic is defined as a mixture of words where each word has a probability of generating from a topic. The example would be words such as 'books', 'college', 'MOOC' could come from the topic of study. A document is a mixture of topics, where a single document can have covered multiple topics. The example would be that a user talks about topics related to career, study, hobbies, and religion in the self-introduction.

We start the topic modeling by using the nonnegative matrix factorization (NMF). It is sometimes preferred to Latent Dirichlet allocation (LDA) as it is a deterministic algorithm and assumes that topic probabilities have to be fixed per self-introduction. Also, it is easy to implement with the `gensim` package in Python as we have prior experience with it. We train the NMF topic model with 25 latent topics and then further visualize the proportions of users who use each of the 25 topics most. We also compare the proportions across different levels in the 4 chosen demographic variables.

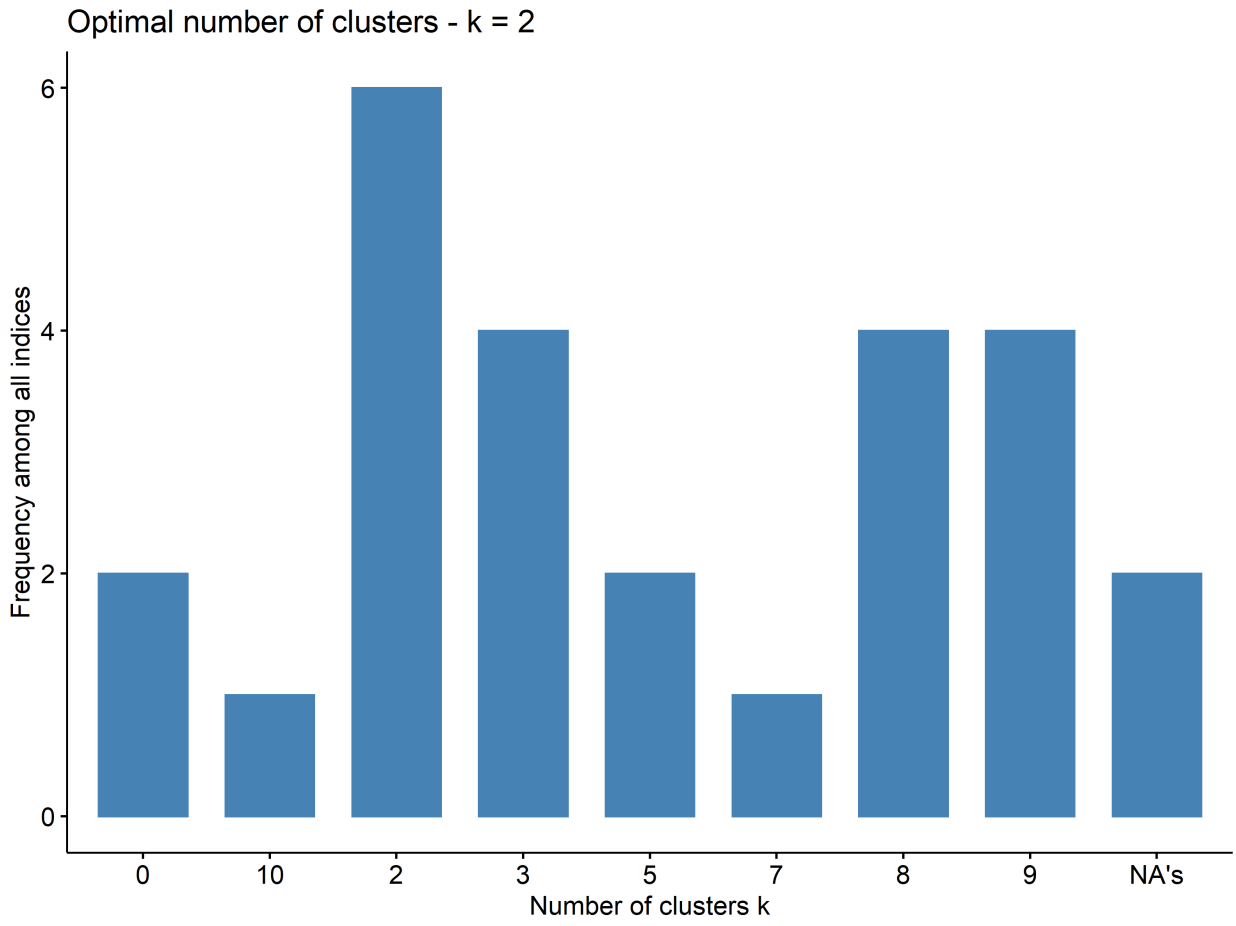


Figure 3: NbClust Results

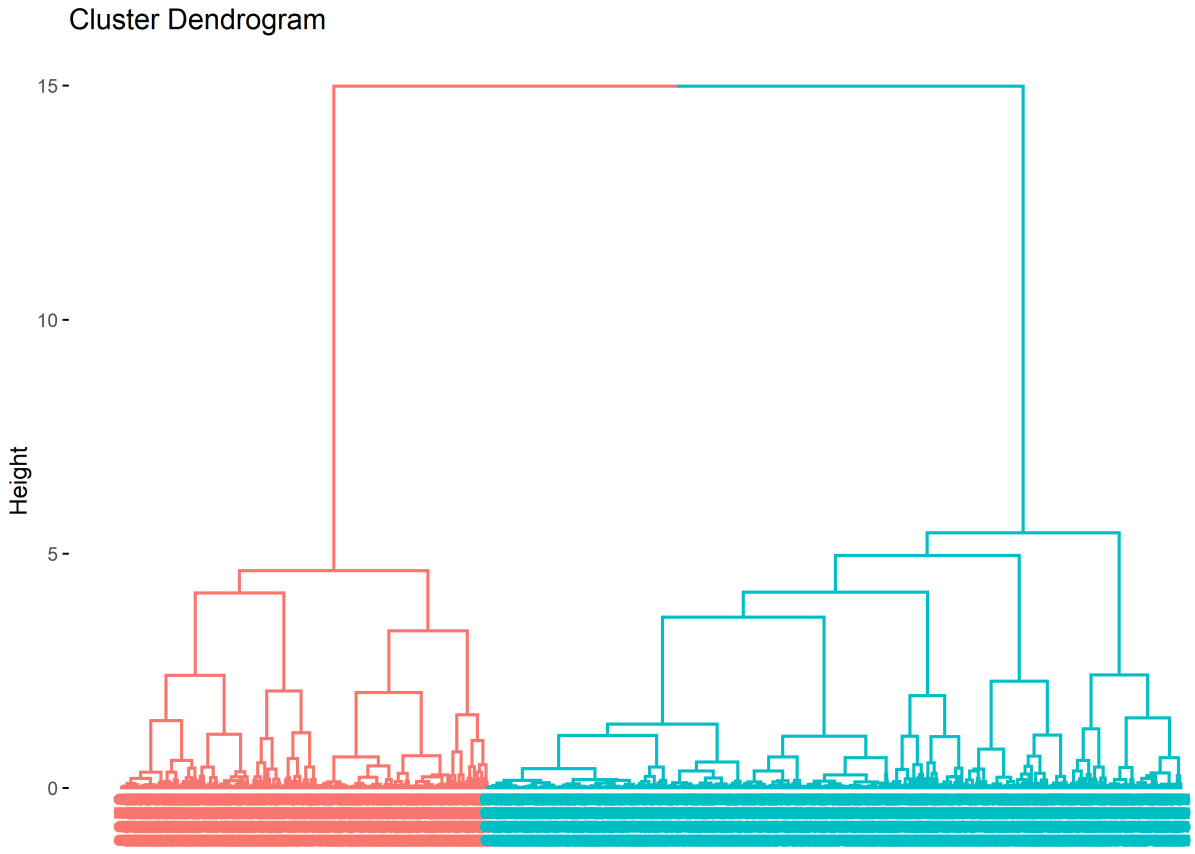


Figure 4: AGNES Dendrogram

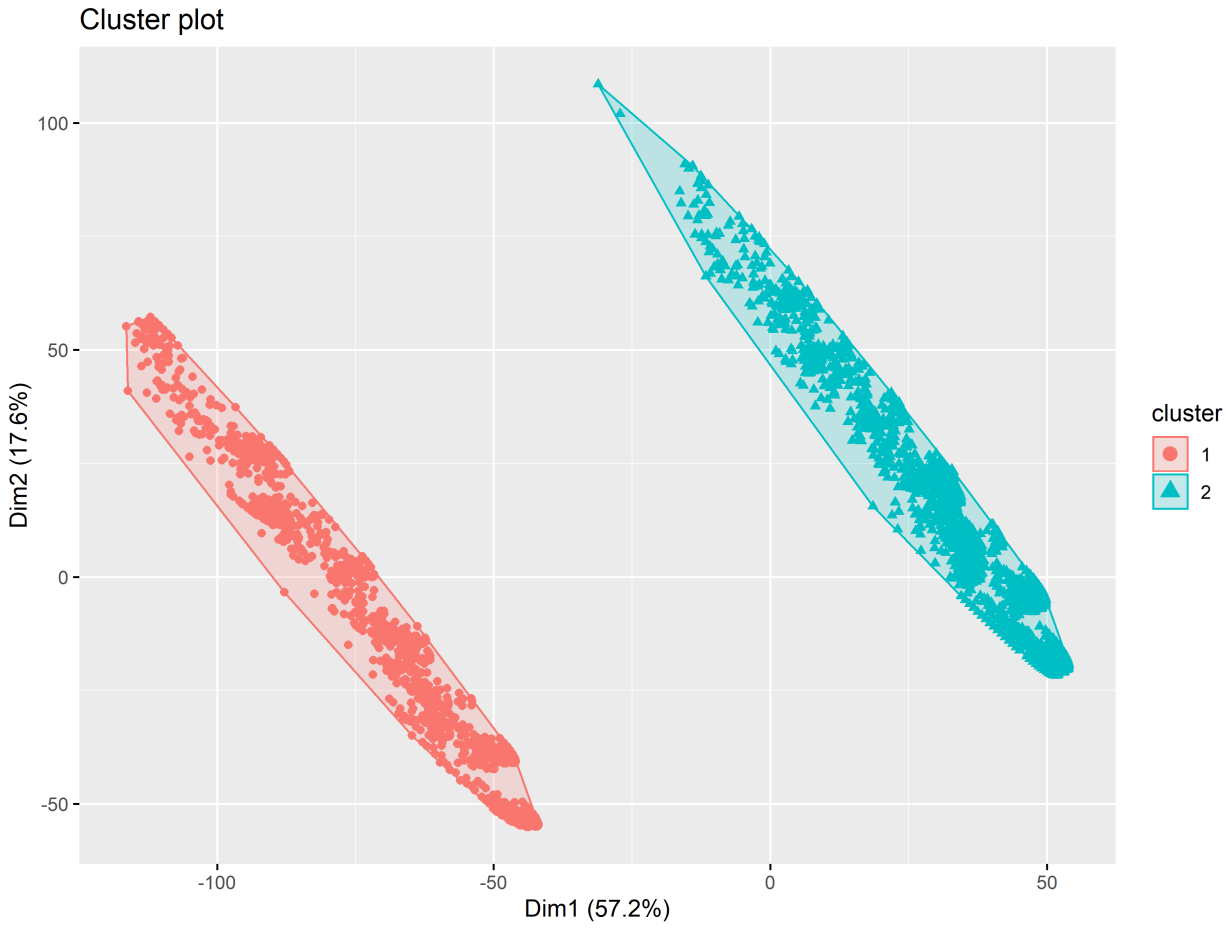
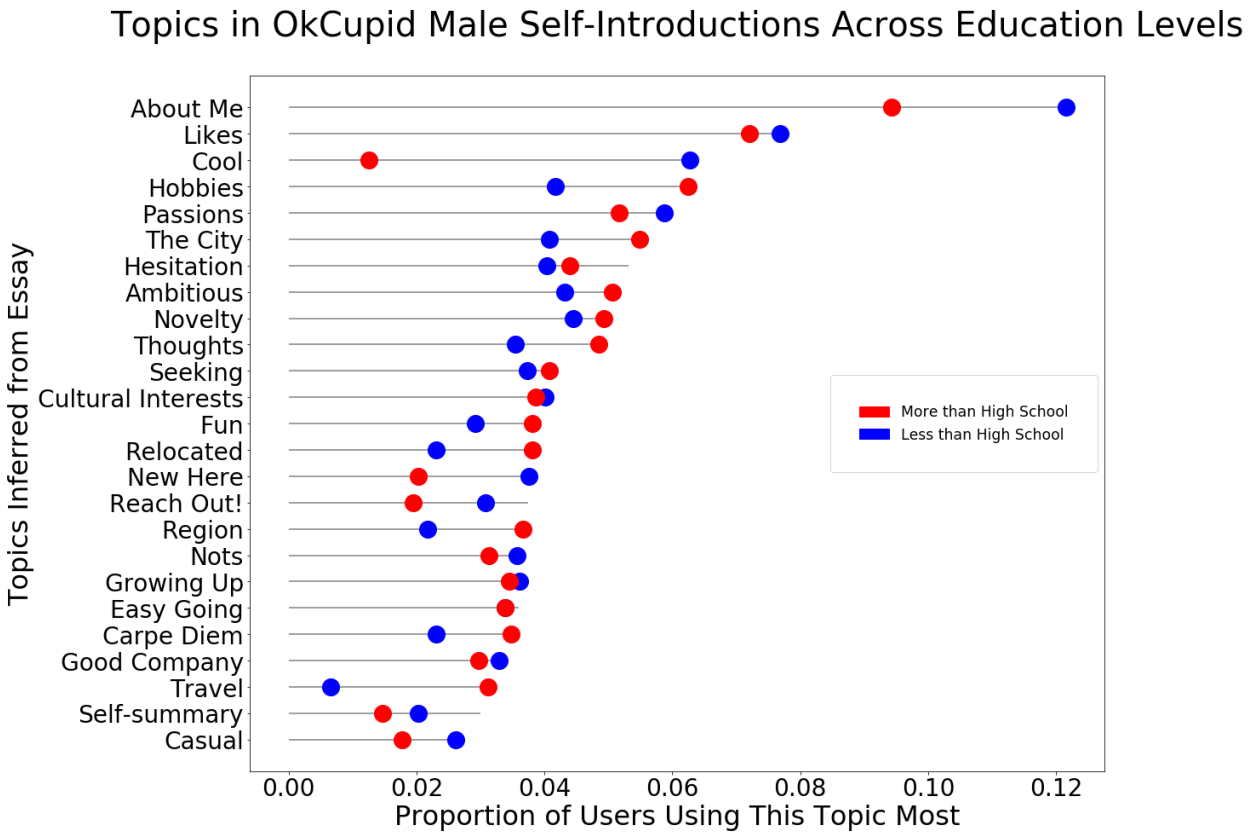
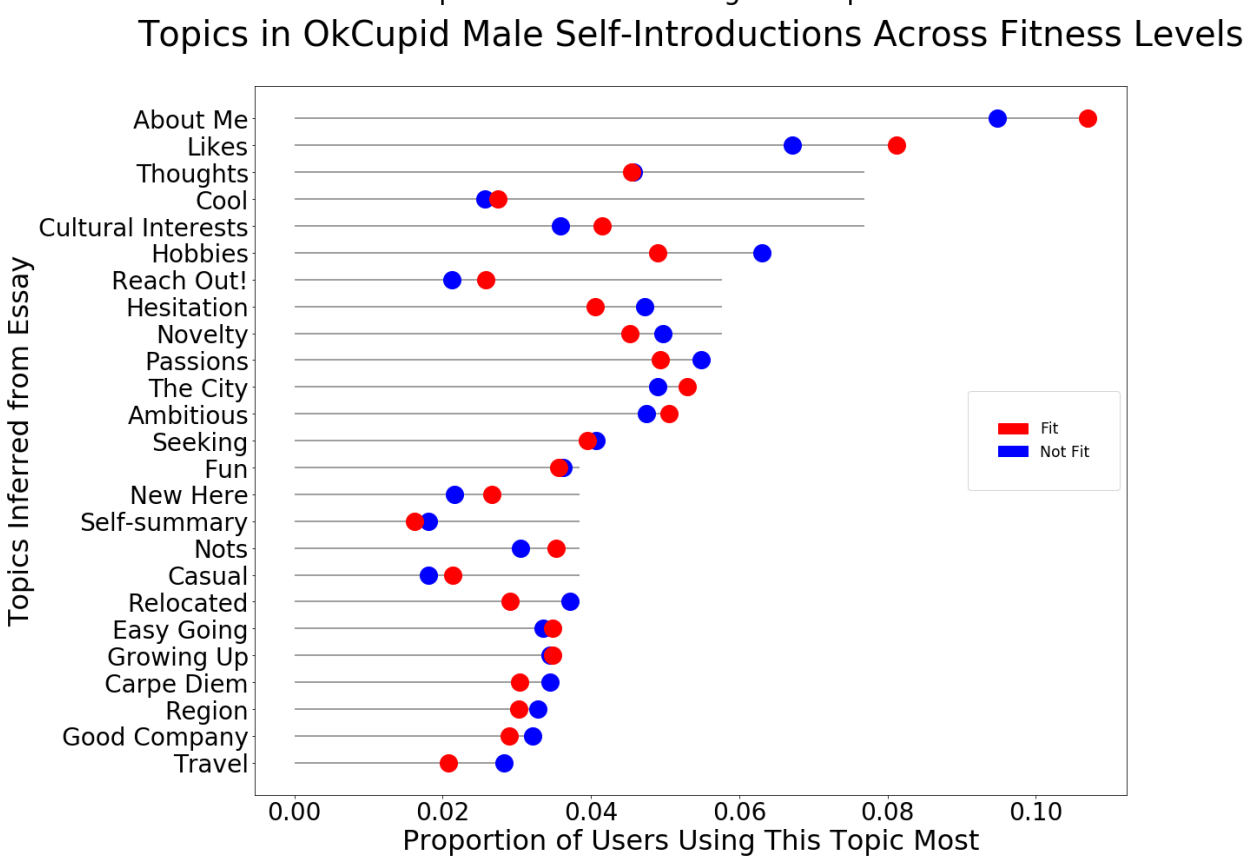
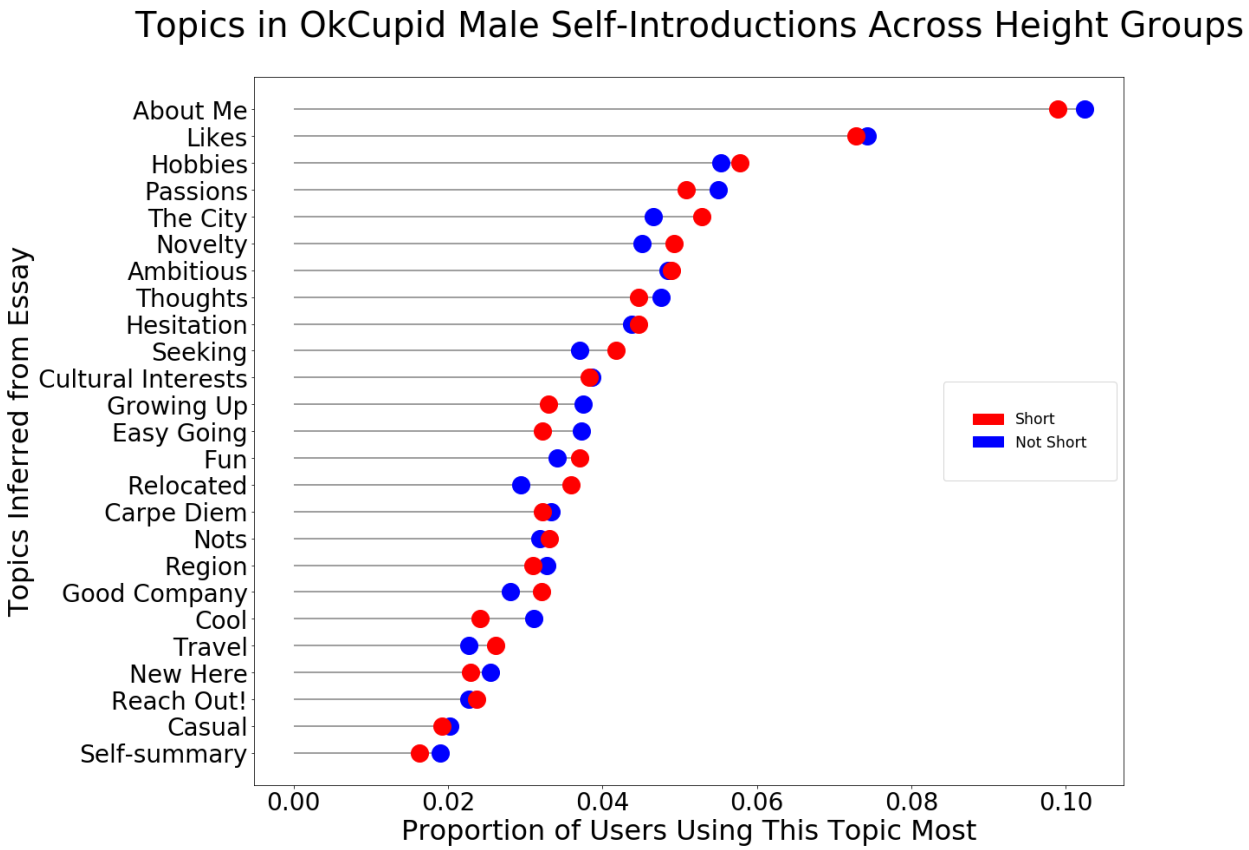
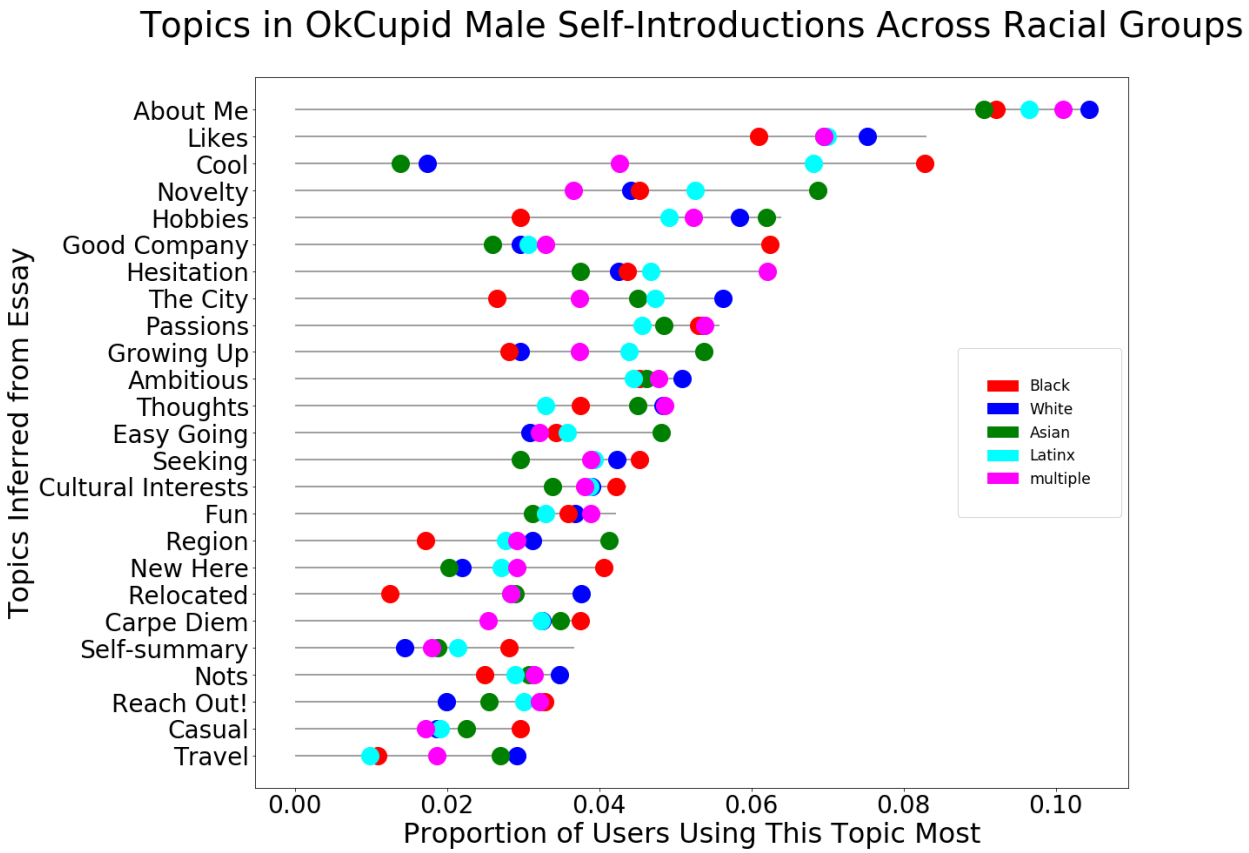


Figure 5: AGNES Cluster Results


Topics in OkCupid Male Self-Introductions Across Education Levels
About Me
Likes
Cool
Hobbies
Passions
The City
Hesitation
Ambitious
Novelty
Thoughts
Seeking
Cultural Interests
Fun
Relocated
New Here
Reach Out!
Region
Nots
Growing Up
Easy Going
Carpe Diem
Good Company
Travel
Self-summary
Casual
Topics Inferred from Essay
More than High School
Less than High School
0.00
0.02
0.04
0.06
0.08
0.10
0.12
Proportion of Users Using This Topic Most

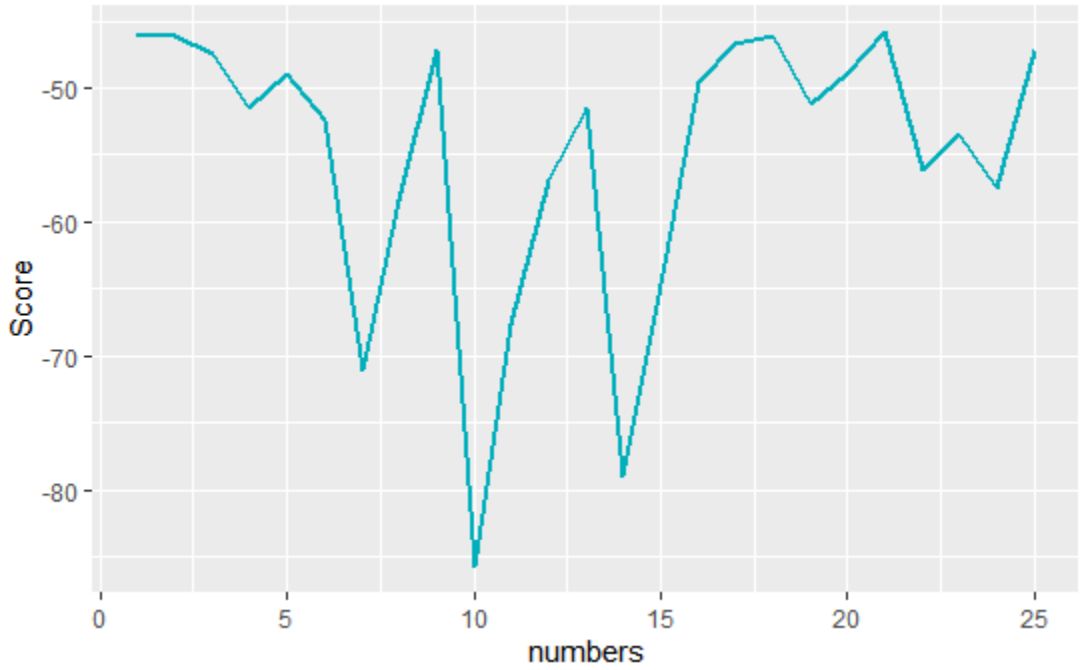

Topics in OkCupid Male Self-Introductions Across Fitness Levels
About Me
Likes
Thoughts
Cool
Cultural Interests
Hobbies
Reach Out!
Hesitation
Novelty
Passions
The City
Ambitious
Seeking
Fun
New Here
Self-summary
Nots
Casual
Relocated
Easy Going
Growing Up
Carpe Diem
Region
Good Company
Travel
Topics Inferred from Essay
Fit
Not Fit
0.00
0.02
0.04
0.06
0.08
0.10
Proportion of Users Using This Topic Most


Topics in OkCupid Male Self-Introductions Across Height Groups
About Me
Likes
Hobbies
Passions
The City
Novelty
Ambitious
Thoughts
Hesitation
Seeking
Cultural Interests
Growing Up
Easy Going
Fun
Relocated
Carpe Diem
Nots
Region
Good Company
Cool
Travel
New Here
Reach Out!
Casual
Self-summary
Short
Not Short
0.00
0.02
0.04
0.06
0.08
0.10
Topics Inferred from Essay
Proportion of Users Using This Topic Most


Topics in OkCupid Male Self-Introductions Across Racial Groups
About Me
Likes
Cool
Novelty
Hobbies
Good Company
Hesitation
The City
Passions
Growing Up
Ambitious
Thoughts
Easy Going
Seeking
Cultural Interests
Fun
Region
New Here
Relocated
Carpe Diem
Self-summary
Nots
Reach Out!
Casual
Travel
Black
White
Asian
Latinx
multiple
0.00
0.02
0.04
0.06
0.08
0.10
Topics Inferred from Essay
Proportion of Users Using This Topic Most

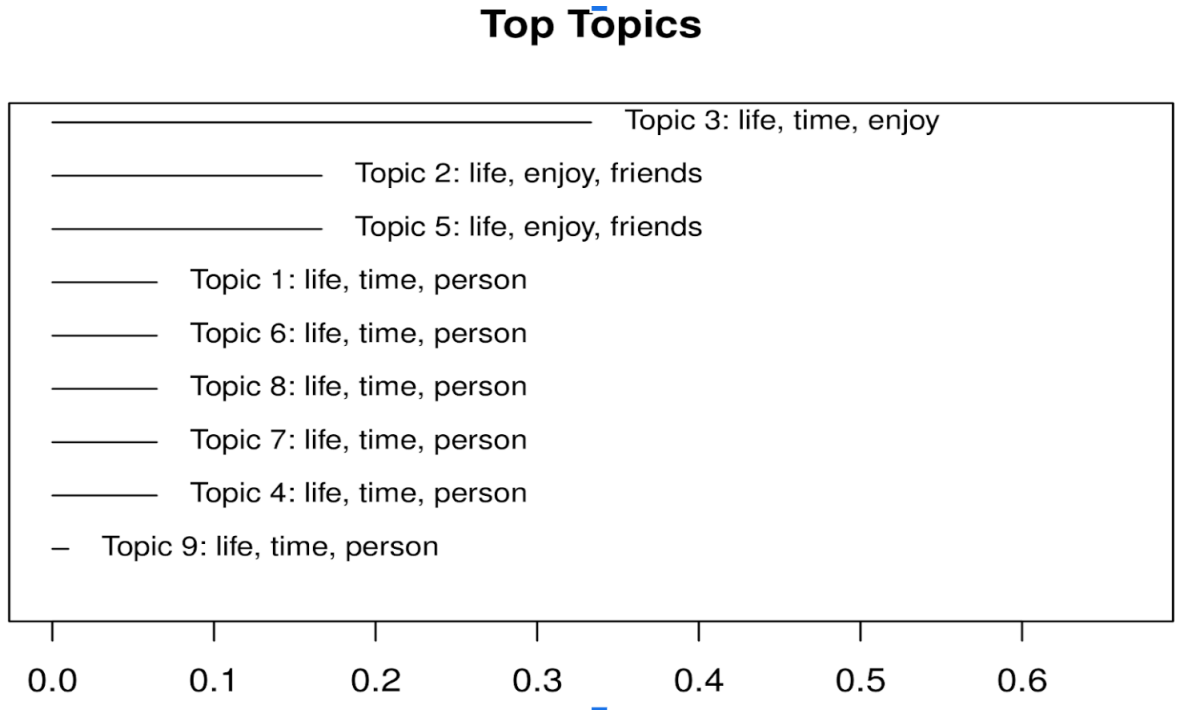
However, although the plots suggest some differences, we don't know if they are statistically significant and also cannot quantify its association with the topic proportions.

As a result, we return to the LDA approach. We choose the topics based on semantic coherence scores.



The plot above suggests 25 topics yields the highest score, but 9 topics yields a good score as well. So we further calculate perplexity , which is regarded as the model's "surprise" at the data. The perplexities are 3261and 3265 for 9 and 25 topics, respectively. Thus, we fix 9 topics for the following analysis.

Next, we fit a LDA topic model with 9 topics where each document is labeled with cluster number (1, 2, 3).



This plot shows that for Cluster 1, almost all topics are associated with the common words such as "life, time, friends". This result is somewhat discouraging as we cannot draw any insights from it. The topic proportion plots are almost identical for Cluster 2 and Cluster 3.

DBSCAN

As we wanted to determine factors behind why a profile might stick out, we decided to use a DBSCAN model was used to detect outliers within a data-set of 50 vectors calculated by Doc2Vec. As Doc2Vec vectors capture different characteristics about the text, any outliers in these vectors should be profiles that deviate from the norm to some degree. As shown below, these Doc2Vec vectors are highly independent with very few correlations, but also highly clusterable, with a Hopkins Statistic of 0.808.

Using a K-nearest-neighbors distribution plot, we determined that the optimal value for the epsilon neighborhood size of the DBSCAN model was 9. This was determined using 5 nearest neighbors, to match the default minimum number of points in the epsilon region, which we used for the model.

The DBSCAN model identified one cluster, and 108 outliers, accounting for 0.57% of the observations. Using a Wilcox Test, we can determine that the difference in the means between the "typical" profiles and outlier profiles are insignificant for the continuous variables for height and number of long words, but are highly significant for Flesch score and age. Mean age is higher among the outliers by about 4.5 years, while the mean Flesch score of outliers is about 8 points higher. This suggests that the outliers, being older and writing at

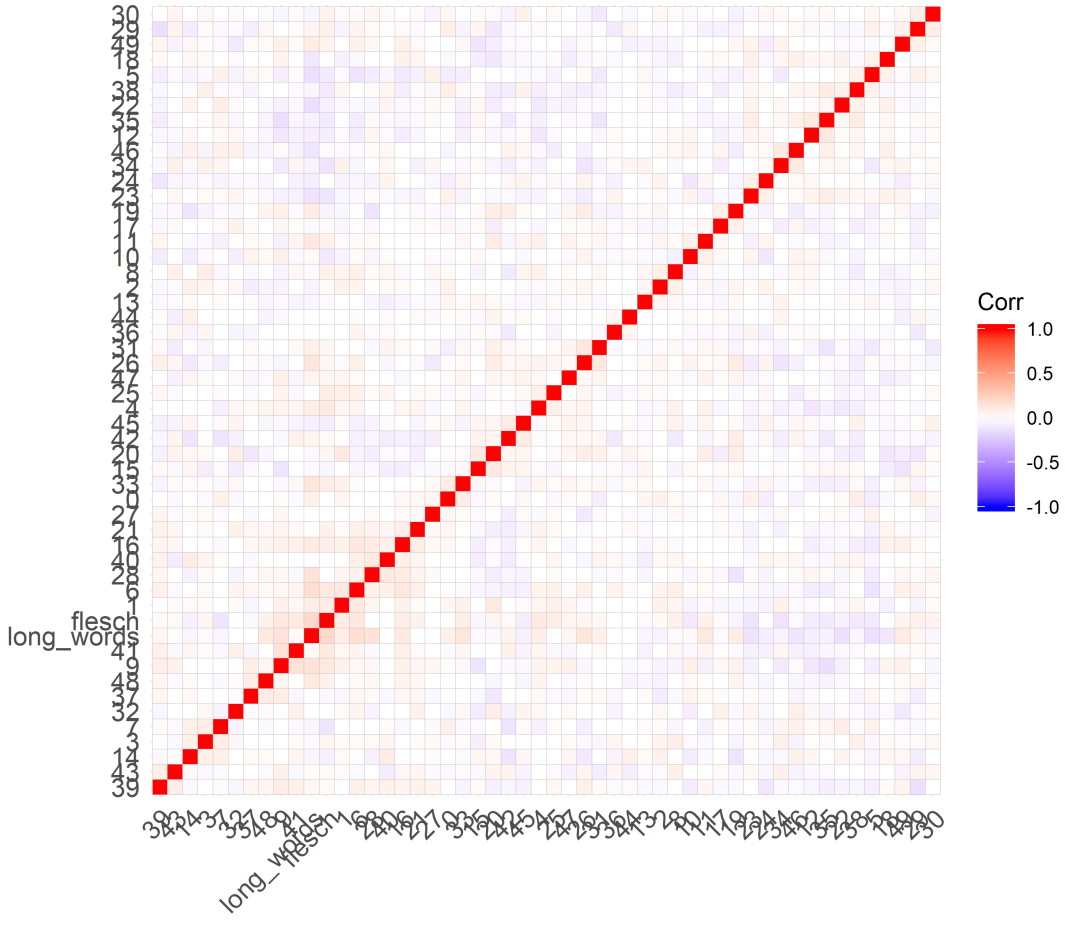


Figure 6: Doc2Vec Correlation Plot

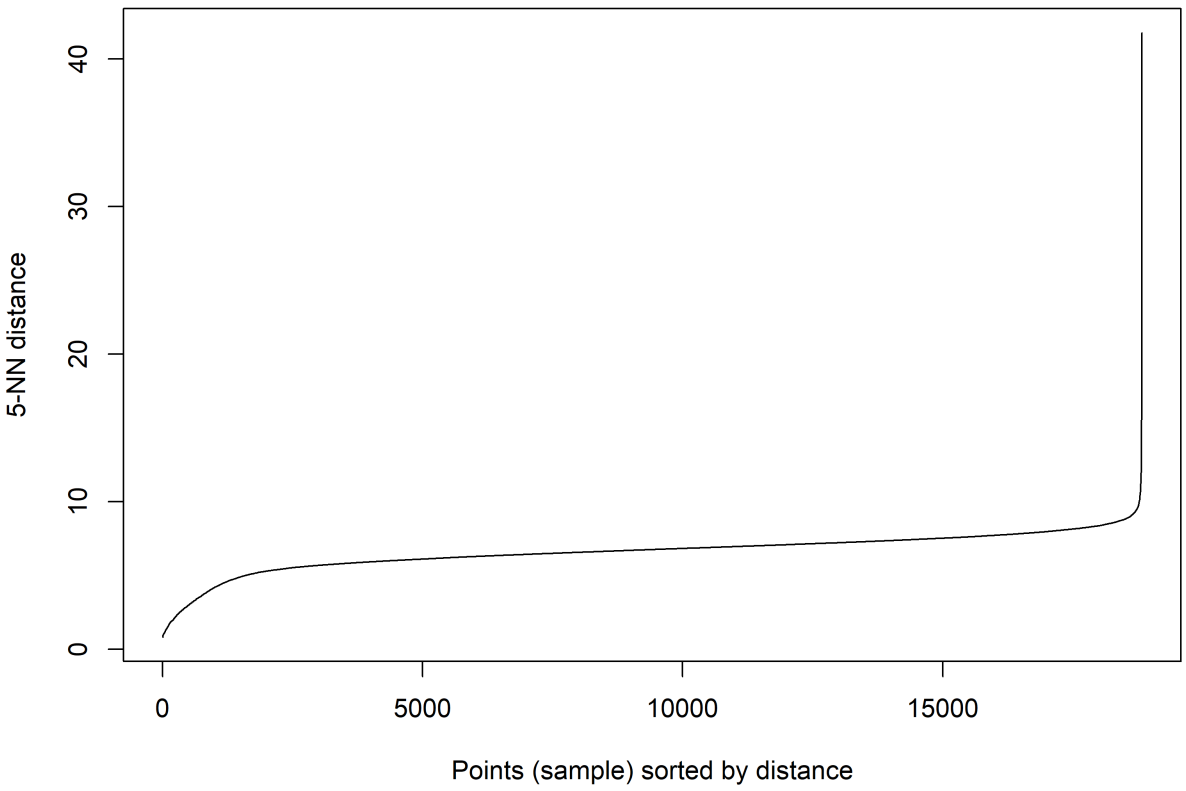


Figure 7: DBSCAN KNN Distribution Plot

a more advanced level, might be more educated (or at least trying to appear so) than their "typical" peers.

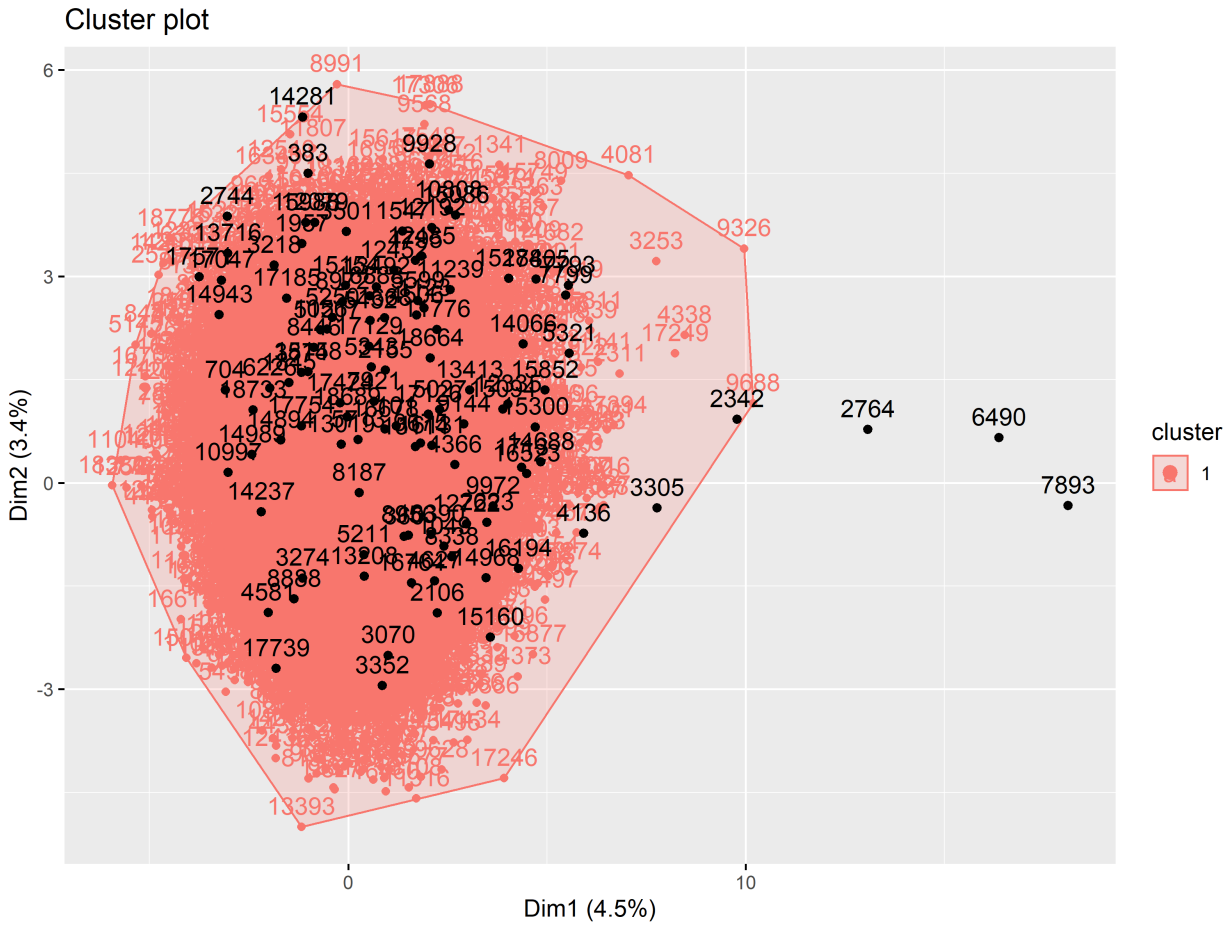


Figure 8: DBSCAN Cluster Plot

Structural Topic Modeling

Motivation

So far, we have explored the patterns within the demographic variables using clustering and self-introduction text using LDA. However, in order to quantify whether people of different backgrounds write different topics, we need to model the topic distributions as a function of the metadata (demographic variables and DBScan score). The structural topic modeling (STM) solves the problem as it allows us to estimate the relationships between topic proportions and document metadata. Similar to the Latent Dirichlet Allocation model, STM also assumes there are some latent document-topic and topic-word distributions generating documents. However, STM differs from LDA as it handles the document-associated metadata. As a result, STM allows

us to predict how topical prevalence (proportion of a topic across multiple documents) or topic content (the topic composition of terms) would shift when the metadata changes.

Estimation

We use the `stm` R package to estimate the model, summarize results, and visualize with the word cloud and topic network. In our project, we are interested in the topical prevalence. So the response (dependent variable) is the proportion of a topic across multiple self-introductions. The metadata (independent variables) is 'fit', 'education', 'height_group', 'race_ethnicity', and 'dbscan_cluster'.

In the preparation step, we use `textProcessor` function to stem the words and remove stopwords. Then we use `prepDocuments` function to structure and index the data. As the low-frequency words are probably the more memorable ones, we set the 'lower.thresh' option to 0 to keep all words.

Then we estimate the model for the topic prevalence using 9 topic models. During the estimation, the proportions of 9 topics are regressed on the metadata. There are many ways to visualize the results. The following plot shows the expected topic proportions

The proportions across 9 topics do not have a large gap, as the most common topic is 0.14 and the least common one is 0.10. We observe that the top three topics seem to consist of common nouns and verbs that are not very memorable. So we further explore the most frequent words in the model for both the most common topics (#5, #4, and #7) and the least common topics (#1, #2, and #9).

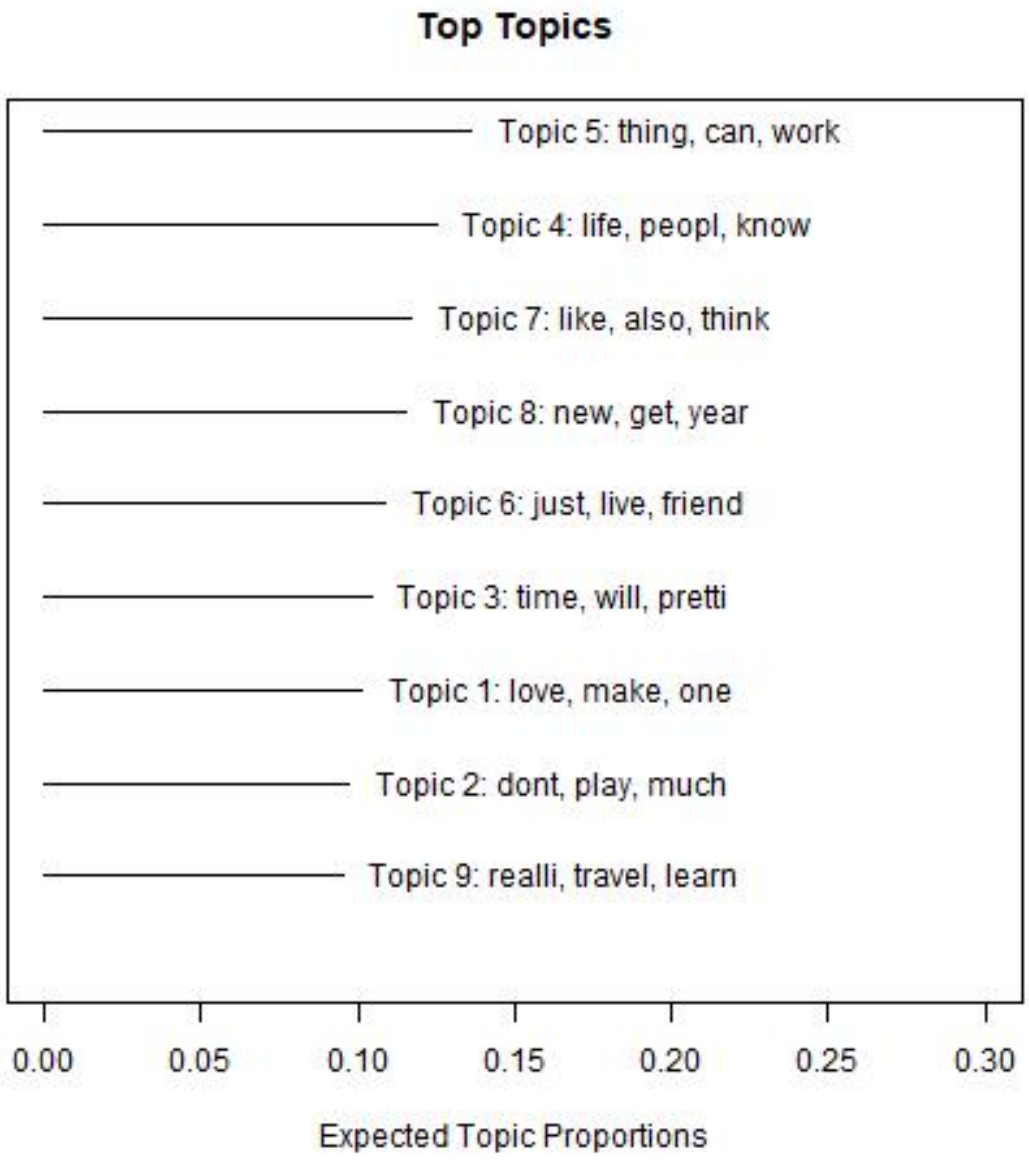


Figure 9: Top Topic

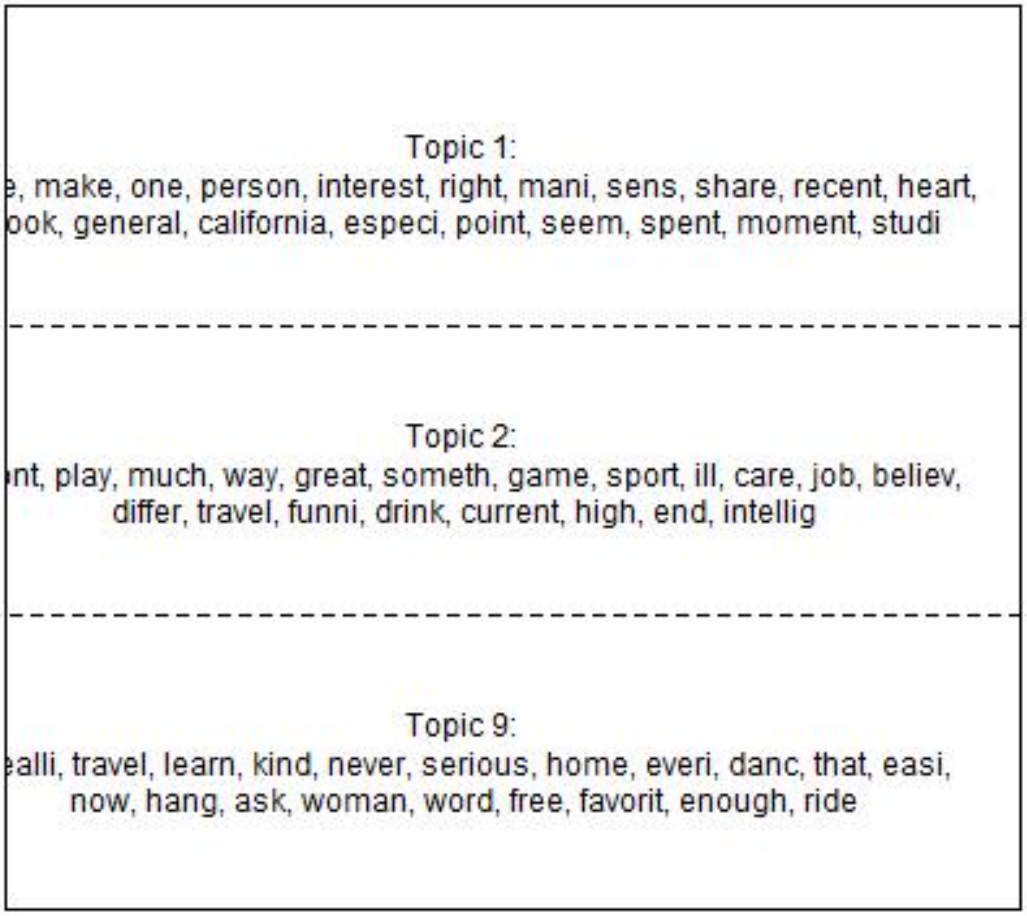
Topic 5:
g, can, work, enjoy, good, music, move, take, day, well, area, feel, now, laugh, adventur, movi, lot, art, better, nice

---

Topic 4:
life, peopl, know, want, alway, meet, citi, place, littl, francisco, still, famili, come, spend, sometim, bit, fun, mean, happi, hope

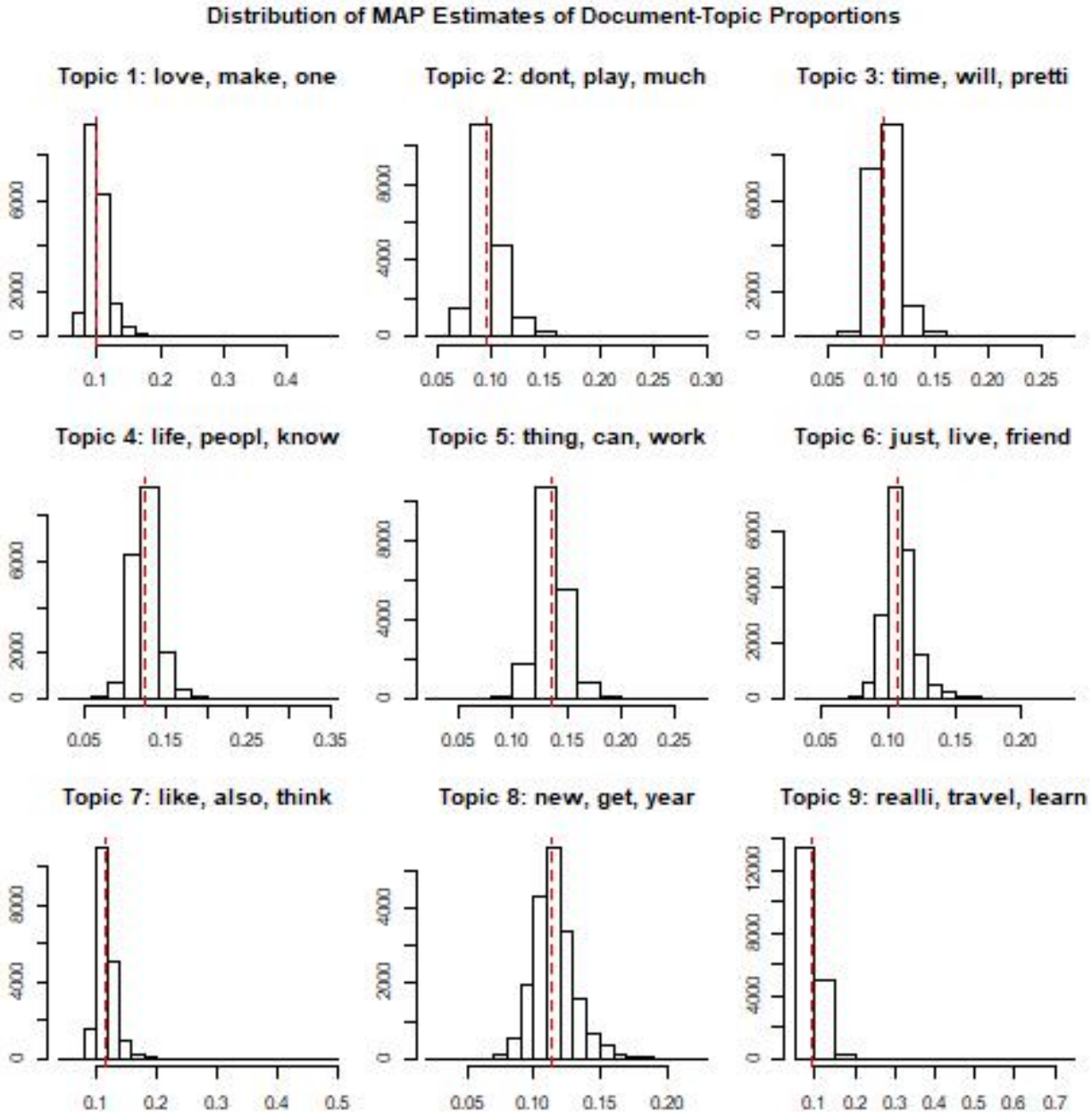
---

Topic 7:
, also, think, someon, ive, world, san, see, fun, even, hike, cook, best, passion, use, self, date, need, often, night

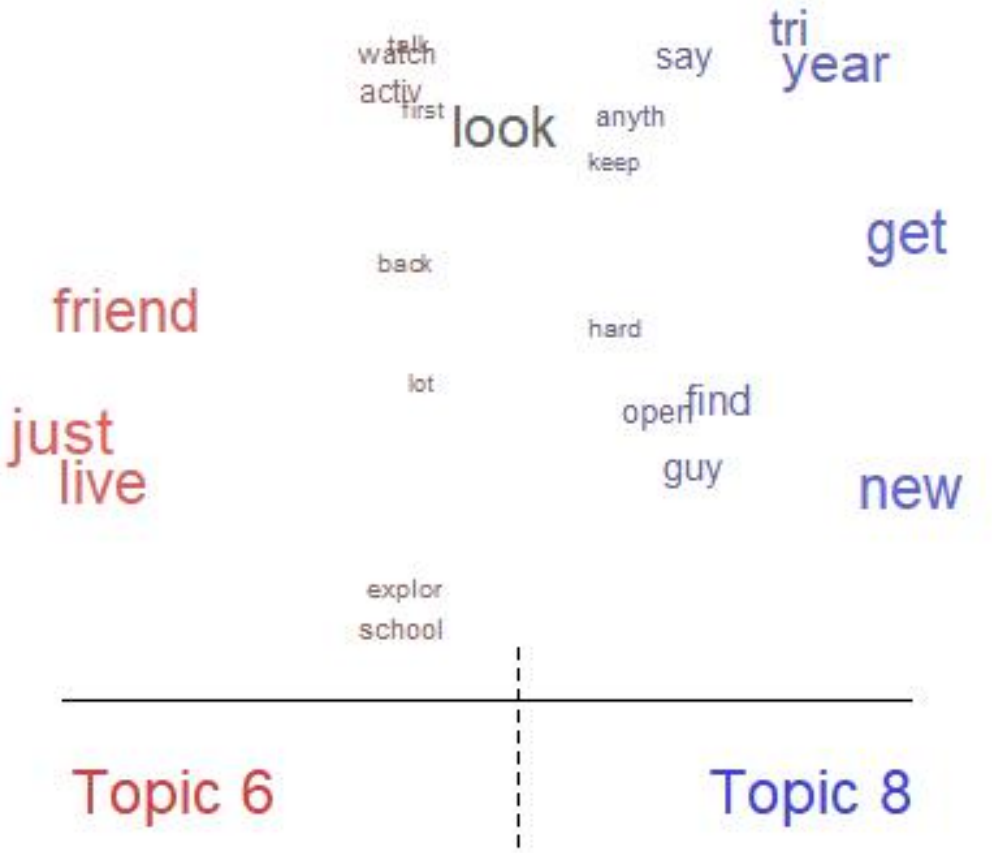


Although it's hard to tell the quantified difference, we find the words in the least common topics are a little more memorable, such as 'intelligent' and 'dance'.

When we plot the histograms of topics, we find that most of the topics consist of around 10% of the topics of the documents. Interestingly, the histograms of topics #6 and #8 seem to have a bell-curve shape, which suggests there are many documents have higher or lower proportions of 10%.



In particular, we plot these two plots on the same line. Topic #6 is about 'friend' and 'live' and topic #8 is about 'year' and 'get'. So our interpretation is that topic #6 s about casually looking for friends while topic #8 is about setting a goal of getting something new in the year. As a result, although most users talk about them explicitly, some talk them more and some use other expressions.



Also, `topicCorr()` calculates correlations between topics, where positive correlations mean that both topics are likely to be discussed within a document.

For the OkCupid beginners, it will be helpful if they could learn what topics are they expected to write together. However, this would only make an ordinary self-introduction. To make it really memorable and stand out, they should consider writing a few topics that are unique and quirky.

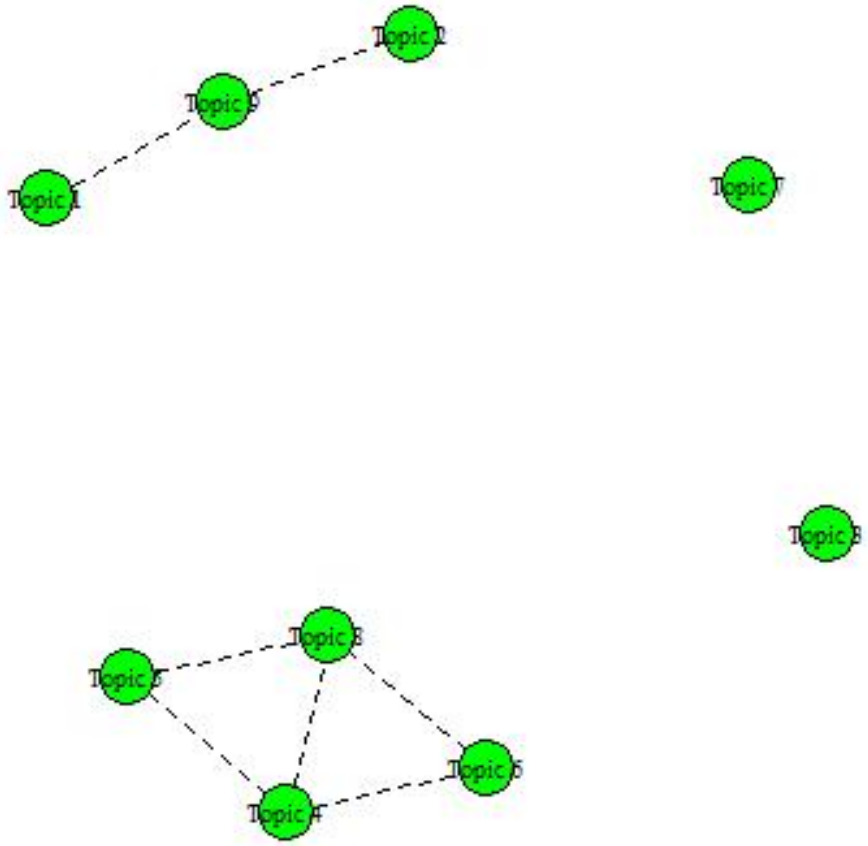


Figure 10: Topic Networks

Evaluation

To further understand whether the topics make sense, we use the topicQuality() function to plot the semantic coherence and exclusivity values associated with each topic.

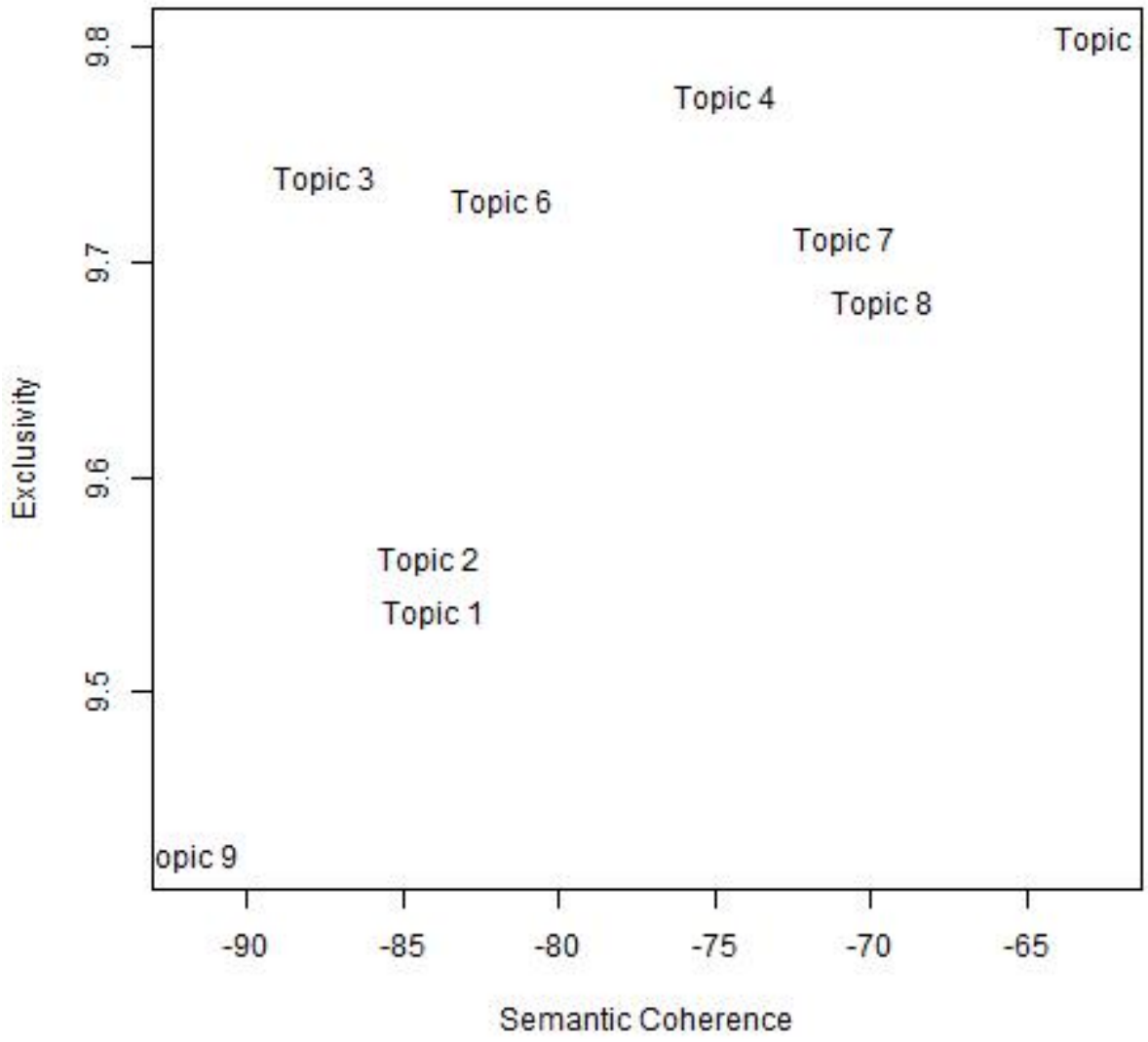


Figure 11: Topic Quality

The results seem to highly correlated with the topic proportions, as topics with higher proportions also have higher semantic coherence and exclusivity values.

Effect

Finally, we use `estimateEffect()` function to estimating relationships between metadata and topic prevalence. There are 9 regressions under the hood and 12 coefficients associated with each. To effectively analyze the results with 108 coefficients, we compile them into a table by indicating the coefficients which are statistically significant at the level of 5%. These significant coefficients suggest that we can reject the null hypothesis and accept the alternative hypothesis that a relationship exists between the topic prevalence and metadata.

Significant Covariates of STM (0: '***' 0.001: '**' 0.01: '*' 0.05 : '.')

Topic / X	fit_not_fit	fit_unknown	edu_More than High School	edu_unknown	height_groupshort	race_ethnicityBlack	race_ethnicityLatinx	race_ethnicitymultiple	race_ethnicityother	race_ethnicityWhite	dbscan_cluster
1			*					***	**	**	***
2	***		***	**			**	*		**	
3	***	*	***	***				**			
4	.		***	*				*		***	**
5	.		*			**					
6			***								
7		.	.		.	.			*		**
8	***		**	***		***	***	***		***	*
9	***		***			***					**

Figure 12: Significant Coefficients

We observe from the table that the factor “edu_More than HIgh Schoo” is significant at all of the 9 regressions. This means that being more educated is associated with some variations in the topic proportions. Intuitively, this suggests more educated people might want to write certain topics that make them memorized as intelligent and knowledgable.

We notice ‘height_groupshort’ is only significant for regression #7. Topic # 7 has high probability words such as “like, also, think, someone”, which seems quite ordinary. It’s likely the use of ‘also’ indicates they are disconfident when introducing themselves, which might because they feel they are too short.

When looking at the coefficients at each regression, we are surprised that in the regression #6, only edu_More than HIgh Schoo" matters for Topic #6’s proportions. In comparison, 8 out of the 11 coefficients are

significant in the regression #8. Our current understanding is that for topic#8, people with different backgrounds have various ideas of how much to write about it.

In summary, we get some interesting results during our first time using structural topic modeling. Although we are still learning it, we believe it has great potential to undercover the underlying relationships between metadata and topic prevalence. Such relationships would be useful for OkCupid to provide users with writing tips to make their self-introduction more memorable.

Output

The above three pictures show the protocol of the final product that we have in mind. The protocol uses the example of Li (one of the project members), although the information is hypothetical. The first picture displays his demographic information and a short self-introduction. The second picture shows that the new algorithm highlights the common and memorable words by comparing them with the frequencies among his peers. He also gets a score of 63 and some tips on improving the writing. The third picture is the revised version of previous writing, which now Li has a memorable self-introduction with an increased score of 87. He should be confident to reach out to other users on OkCupid now.

Discussion

The data of this project has several limitations at this stage. Firstly, without data on the outcome (such as the number of messages), we do not know have a relative measure for the self-introductions' memorabilities. As a result, we don't have the outcomes to train the scoring function with machine learning. Secondly, without follow-up interviews, we cannot measure whether the specific choice of words was aimed at authenticity or matches with an awareness of 'relationshopping' (experienced users use certain phrases fraudulently to hook others). Thirdly, without the users' other profile images, we cannot estimate the effect of a memorable self-introduction on outcomes as the quality of images is a potential moderating variable.

One important issue of applying the structural topic modeling is overfitting, especially when we have a large number of covariates. We tried to measure the model's accuracy by residual sum of squares on both training set and testing test. However, at this point, we haven't figured out how to predict the new topic prevalences for the testing set. Also there are many remaining questions for this to work. For example, should we measure the accuracy by the average of 9 RSSs? If we use Lasso regression to select a subset of covariates, how should we make the interpretation as there could be different covariates in 9 regressions?

Conclusion

In this project, we apply different unsupervised machine learning methods learned from class, especially clustering and text mining. Although we have got some interesting results, we realize that there are many other alternative methods that are potentially helpful, such as factor model and association rule mining. A factor analysis may be useful for future analysis as the variables driving choice of words- such as cognitive skills, cultural attitudes, personality traits are likely to be latent and unobservable factors. Association rule mining would further discover the interesting associations within people's demographics or find certain phrases in self-introductions that often occur together.

Lastly, we want to express our excitement for learning these useful tools from the UML class. Also we would like to thank you Dr. Waggoner and our classmates for providing valuable feedback during the two rounds of presentation. This experience greatly arises our interest to study more UML methods in the future.

Appendices

Below is all the source code used for this project, which is available on the GitHub repo for this project: https://github.com/tonofshell/unsupervised-dating

Here is the contents for you to navigate:

Kmeans Code

```
## ----prep-----------------------------------------------------------------
library(tidyverse)
library(here)
library(cluster)
library(FactoMineR)
library(factoextra)
library(NbClust)
library(dbscan)
library(cowplot)
library(skimr)
library(ggcorrplot)
library(missMDA)
library(cluster)
library(data.table)


setwd("C:/Users/lliu9/Desktop/UML_Project/unsupervised-dating/Data")
```

```r
data <- read_csv("final_okcupid.csv")
data_sample <- sample_n(data, 2000)

demo <- data_sample[c("fit", "edu", "height_group",
    "race_ethnicity")]

gower_demo <- demo %>% mutate_all(factor) %>% daisy(metric = "gower")



## ----Kmeans-----------------------------------------------------------------
set.seed(123)

kmeans <- kmeans(gower_demo, centers = 3, nstart = 15)




## ------------------------------------------------------------------------
text <- data_sample[c("clean_text")]
text$kmeanscluster = as.factor(kmeans$cluster)
write_csv(text, "essay0.csv")



## ----Examination------------------------------------------------------------
data_sample$kmeanscluster = as.factor(kmeans$cluster)

# Inspect the kmeans object
str(kmeans)



## ----Visualization----------------------------------------------------------
jpeg("kmeans3.jpeg")
fviz_cluster(kmeans, gower_demo)
```

```
dev.off()



## ----validation-----------------------------------------------------------
library(fpc)
# Calinski Harabasz index
clustering.ch <- kmeansruns(gower_demo, krange = 1:5,
    criterion = "ch")
# average silhouette width
clustering.asw <- kmeansruns(gower_demo, krange = 1:5,
    criterion = "asw")

critframe <- data.frame(k = 1:10, ch = scale(clustering.ch$crit),
    asw = scale(clustering.asw$crit))

critframe <- melt(critframe, id.vars = c("k"), variable.name = "measure",
    value.name = "score")

ggplot(critframe, aes(x = k, y = score, color = measure)) +
    geom_point(aes(shape = measure)) + geom_line(aes(linetype = measure)) +
    scale_x_continuous(breaks = 1:10, labels = 1:10)
summary(clustering.ch)
```

AGNES Code

```
## ----setup,
## include=FALSE------------------------------------------------------
library(tidyverse)
library(knitr)
library(here)
library(cluster)
library(FactoMineR)
```

```
library(factoextra)
library(NbClust)
library(dbscan)
library(cowplot)
library(skimr)
library(ggcorrplot)
library(missMDA)
library(cluster)
library(missForest)
library(tictoc)
library(doParallel)

knitr::opts_chunk$set(echo = TRUE, fig.height = 6,
    fig.width = 8, dpi = 400)

set.seed(60615)



## ----data-----------------------------------------------------------------
original_data = read_csv(here("Data", "final_okcupid.csv")) %>%
    select(-c("new_index", "orig_index", "clean_text",
        "essay9", "long_words", "flesch", "dbscan_cluster",
        "profile_length", "prop_longwords"))

names(original_data)



## ----descr-stats, cache=TRUE, fig.height = 8,
## fig.width = 11-----------------
skim_list = original_data %>% skim() %>% partition()

skim_list$numeric %>% kable()
```

```r
skim_list$character %>% kable()


original_data %>% mutate_if(is.character, factor) %>%
    mutate_all(as.numeric) %>% cor(use = "pairwise.complete.obs") %>%
    ggcorrplot()

clusterability = original_data %>% mutate_if(is.character,
    factor) %>% mutate_all(as.numeric) %>% sample_n(5000) %>%
    get_clust_tendency(n = 50)
clusterability$hopkins_stat
clusterability$plot



## ----pca,
## cache=TRUE-------------------------------------------------------------
original_data %>% mutate_if(is.character, factor) %>%
    mutate_all(as.numeric) %>% mutate_all(scale) %>%
    PCA(graph = FALSE) %>% fviz_pca_biplot(label = "var",
    col.var = "red", col.ind = "grey")
ggsave2(here("Clustering", "pca_v2.png"), height = 7,
    width = 11)



## ----agg-nest, error=TRUE,
## cache=TRUE-----------------------------------------
sampled_data = original_data %>% sample_n(5000)
agnes_data = sampled_data %>% mutate_if(is.character,
    factor) %>% mutate_all(as.numeric) %>% mutate_all(scale)
agnes_diss = agnes_data %>% as.matrix() %>% daisy(metric = "gower")
nb_results = NbClust(data = agnes_data, diss = agnes_diss,
    distance = NULL, min.nc = 2, max.nc = 10, method = "ward.D2")
```

```r
fviz_nbclust(nb_results)


agnes_mod = agnes_diss %>% hcut(isdiss = TRUE, k = 2,
    hc_func = "agnes", hc_method = "ward.D2")
fviz_dend(agnes_mod)
sampled_data$cluster = agnes_mod$cluster
fviz_cluster(agnes_mod, data = agnes_diss, labelsize = 0)


saveRDS(sampled_data, here("Data", "Results", "agnes_results.rds"))
write_csv(sampled_data, here("Data", "Results", "agnes_results.csv"))



## ----clust-inter-----------------------------------------------------------
modal = function(vect, percent = FALSE, only_one = FALSE) {
    library(tidyverse)
    modal_val = vect %>% unlist() %>% table() %>% .[. ==
        max(.)] %>% names()
    if (only_one) {
        modal_val = modal_val[1]
    }
    if (percent) {
        return(vect %>% unlist() %>% .[. == modal_val] %>%
            (function(x) length(x)/length(vect)))
    }
    modal_val
}


cluster_significance = function(var, data, clus_var = "cluster") {
    wilcox.test(as.formula(paste(var, "~", clus_var)),
        data)$p.value
}
```

```r
sampled_data %>% select_if(is.numeric) %>% group_by(cluster) %>%
    summarise_all(mean) %>% kable(caption = "Mean by Cluster")

sampled_data %>% select_if(is.numeric) %>% {
    sapply(names(select(., -cluster)), cluster_significance,
        data = .)
} %>% round(3) %>% enframe() %>% kable(caption = "Wilcox Test P-values")

sampled_data %>% select_if(is.numeric) %>% group_by(cluster) %>%
    group_by(cluster) %>% summarise_all(sd) %>% kable(caption = "Standard Deviation by Cluster")

sampled_data %>% select_if(is.numeric) %>% group_by(cluster) %>%
    group_by(cluster) %>% summarise_all(median) %>%
    kable(caption = "Median by Cluster")

sampled_data %>% mutate(cluster = factor(cluster)) %>%
    select_if((function(x) !is.numeric(x))) %>% group_by(cluster) %>%
    summarise_all(modal, only_one = TRUE) %>% kable(caption = "Mode by Cluster")
sampled_data %>% mutate(cluster = factor(cluster)) %>%
    select_if((function(x) !is.numeric(x))) %>% group_by(cluster) %>%
    summarise_all(modal, percent = TRUE, only_one = TRUE) %>%
    mutate_if(is.numeric, round, 3) %>% kable(caption = "Mode by Cluster")
```

DBSCAN Code

```r
## ----setup,
## include=FALSE-------------------------------------------------------
library(tidyverse)
library(knitr)
library(here)
library(cluster)
library(FactoMineR)
```

```
library(factoextra)
library(NbClust)
library(dbscan)
library(cowplot)
library(skimr)
library(ggcorrplot)

knitr::opts_chunk$set(echo = TRUE, fig.height = 6,
    fig.width = 8, dpi = 400)

set.seed(60615)



## ----data-----------------------------------------------------------------
demo_data = read_csv(here("Data", "compressed_okcupid.csv"))
doc2vec_data = read_csv(here("Data", "doc2vec_results.csv")) %>%
    bind_cols(select(demo_data, long_words, flesch)) %>%
    scale() %>% as_tibble()
doc2vec_data %>% skim() %>% partition() %>% .$numeric %>%
    kable()
doc2vec_data %>% select(-X1) %>% {
    ggcorrplot(cor(.), p.mat = cor_pmat(.), hc.order = TRUE,
        insig = "blank")
}



## ----clusterability-------------------------------------------------------
clusterability = doc2vec_data %>% select(-X1) %>% get_clust_tendency(n = 15)
# clusterability$plot



## ----dbscan---------------------------------------------------------------
```

```
doc2vec_data %>% select(-X1) %>% kNNdistplot(k = 5)
dbscan_mod = doc2vec_data %>% select(-X1) %>% dbscan(9,
    5)
doc2vec_data %>% select(-X1) %>% {
    fviz_cluster(dbscan_mod, data = .)
}

dbscan_results = doc2vec_data %>% bind_cols(enframe(dbscan_mod$cluster,
    name = NULL, value = "cluster"))
read_csv(here("Data", "compressed_with_results.csv")) %>%
    mutate(dbscan_cluster = dbscan_mod$cluster) %>%
    write_csv(here("Data", "compressed_with_results.csv"))



## ----merge-----------------------------------------------------------------
merged_demo_data = demo_data %>% select(-essay0, -essay9) %>%
    bind_cols(select(dbscan_results, cluster))
merged_doc2vec_data = demo_data %>% select(X1, essay0,
    essay9) %>% bind_cols(select(dbscan_results, -X1))



## ----demo-data-------------------------------------------------------------
modal = function(vect, percent = FALSE, only_one = FALSE) {
    library(tidyverse)
    modal_val = vect %>% unlist() %>% table() %>% .[. ==
        max(.)] %>% names()
    if (only_one) {
        modal_val = modal_val[1]
    }
    if (percent) {
        return(vect %>% unlist() %>% .[. == modal_val] %>%
            (function(x) length(x)/length(vect)))
```

```
    }
    modal_val
}

cluster_significance = function(var, data, clus_var = "cluster") {
    wilcox.test(as.formula(paste(var, "~", clus_var)),
        data)$p.value
}

merged_demo_data %>% select(-c(X1, education)) %>%
    select_if(is.numeric) %>% group_by(cluster) %>%
    summarise_all(mean) %>% kable(caption = "Mean by Cluster")

merged_demo_data %>% select(-X1) %>% select_if(is.numeric) %>%
    {
        sapply(names(select(., -cluster)), cluster_significance,
            data = .)
    } %>% round(3) %>% enframe() %>% kable(caption = "Wilcox Test P-values")

merged_demo_data %>% select(-c(X1, education)) %>%
    select_if(is.numeric) %>% group_by(cluster) %>%
    group_by(cluster) %>% summarise_all(sd) %>% kable(caption = "Standard Deviation by Cluster")
merged_demo_data %>% select(-c(X1, education)) %>%
    select_if(is.numeric) %>% group_by(cluster) %>%
    group_by(cluster) %>% summarise_all(median) %>%
    kable(caption = "Median by Cluster")

merged_demo_data %>% select(-c(X1, education)) %>%
    mutate(cluster = factor(cluster)) %>% select_if((function(x) !is.numeric(x))) %>%
    group_by(cluster) %>% summarise_all(modal, only_one = TRUE) %>%
    kable(caption = "Mode by Cluster")
merged_demo_data %>% select(-c(X1, education)) %>%
```

```
    mutate(cluster = factor(cluster)) %>% select_if((function(x) !is.numeric(x))) %>%
    group_by(cluster) %>% summarise_all(modal, percent = TRUE,
    only_one = TRUE) %>% mutate_if(is.numeric, round,
    3) %>% kable(caption = "Mode by Cluster")


## ----interpret-outliers--------------------------------------------------------
dbscan_results %>% select(-X1) %>% group_by(cluster) %>%
    summarise_all(mean) %>% kable(caption = "Mean by Cluster")
dbscan_results %>% select(-X1) %>% group_by(cluster) %>%
    summarise_all(sd) %>% kable(caption = "Standard Deviation by Cluster")
dbscan_results %>% select(-X1) %>% group_by(cluster) %>%
    summarise_all(median) %>% kable(caption = "Median by Cluster")


## ----profile-diff--------------------------------------------------------------
merged_doc2vec_data %>% select(cluster, essay0) %>%
    group_by(cluster) %>% sample_n(10) %>% kable()
```

Doc2Vec Code

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
import numpy as np
```

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns


# In[2]:


import matplotlib.pyplot as plt


# In[3]:


df = pd.read_csv('../Data/compressed_okcupid.csv').dropna(subset=['essay0'])
df.info()


# In[4]:


data = df['essay0']


# In[5]:


tagged_data = [TaggedDocument(words=word_tokenize(_d.lower()),
                              tags=[str(i)]) for i, _d in enumerate(df['essay0'])]
```

```
# In[6]:



from gensim.models.doc2vec import Doc2Vec

max_epochs = 50
vec_size = 50
alpha = 0.025

model = Doc2Vec(size=vec_size,
                alpha=alpha,
                min_alpha=0.00025,
                min_count=1,
                dm =1)

model.build_vocab(tagged_data)

for epoch in range(max_epochs):
    print('iteration {0}'.format(epoch))
    model.train(tagged_data,
                total_examples=model.corpus_count,
                epochs=model.iter)
    # decrease the learning rate
    model.alpha -= 0.0002
    # fix the learning rate, no decay
    model.min_alpha = model.alpha

model.save("dv_50.model")
print("Model Saved")



# In[7]:
```

```
#from gensim.models.doc2vec import Doc2Vec

model= Doc2Vec.load("dv_50.model")
#to find the vector of a document which is not in training data
#test_data = word_tokenize("I love chatbots".lower())
#v1 = model.infer_vector(test_data)
#print("V1_infer", v1)

# to find most similar doc using tags
similar_doc = model.docvecs.most_similar('0')
print(similar_doc)


# In[8]:


model.docvecs.vectors_docs.shape
type(model.docvecs.vectors_docs)


# In[9]:


#Visualize TSNE with doc2vec
from sklearn.manifold import TSNE
def doc2vec_tsne_plot(doc_model, labels):

    tokens = []
    for i in range(len(doc_model.docvecs.vectors_docs)):
        tokens.append(doc_model.docvecs.vectors_docs[i])
```

```
    # Reduce 100 dimensional vectors down into 2-dimensional space so that we can see them
    tsne_model = TSNE(perplexity=40, n_components=2, init='pca', n_iter=2500, random_state=23)
    new_values = tsne_model.fit_transform(tokens)

    X = [doc[0] for doc in new_values]
    y = [doc[1] for doc in new_values]

    # Combine data into DataFrame, so that we plot it easily using Seaborn
    df = pd.DataFrame({'X':X, 'y':y, 'Cuisine':labels})
    plt.figure(figsize=(16, 16))
    sns.scatterplot(x="X", y="y", hue="Cuisine", data=df)
    return

doc2vec_tsne_plot(model, df['height'])



# In[10]:



from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

X = model.docvecs.vectors_docs
Sum_of_squared_distances = []
K = range(2,15)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(X)
    Sum_of_squared_distances.append(km.inertia_)
```

```
# In[11]:



plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Distances from Cluster Centres')
plt.title('Elbow Method For Optimal Number of Clusters', fontsize=15)
plt.savefig('Elbow_Plot.png', bbox_inches='tight')



# In[12]:



num_clusters = 4
num_seeds = 4
max_iterations = 300
labels_color_map = {
    0: '#20b2aa', 1: '#ff7373', 2: '#ffe4e1', 3: '#005073', 4: '#4d0404'
}
pca_num_components = 2
#texts_list = df['essay0']
# calculate tf-idf of texts
#tf_idf_vectorizer = TfidfVectorizer(analyzer="word", use_idf=True,
                                    #smooth_idf=True, ngram_range=(2, 3))
#tf_idf_matrix = tf_idf_vectorizer.fit_transform(texts_list)

# create k-means model with custom config
clustering_model = KMeans(
    n_clusters=num_clusters,
    max_iter=max_iterations,
    precompute_distances="auto",
    n_jobs=-1
```

```
)


X = model.docvecs.vectors_docs
reduced_data = PCA(n_components=pca_num_components).fit_transform(X)
labels = clustering_model.fit_predict(reduced_data)



# In[13]:



pca_num_components = 2
X = model.docvecs.vectors_docs
reduced_data = PCA(n_components=pca_num_components).fit_transform(X)



# In[16]:



# there appears to be no column named 'clust_label' in df
#df[df['clust_label']==1]['essay0']



# In[17]:



from sklearn.manifold import TSNE
# Creating and fitting the tsne model to the document embeddings
tsne_model = TSNE(early_exaggeration=4,
                  n_components=2,
                  verbose=1,
                  random_state=2018,
                  n_iter=300)
```

```
tsne_d2v = tsne_model.fit_transform(model.docvecs.vectors_docs)



# In[18]:



plt.scatter(tsne_d2v [:, 0], tsne_d2v [:, 1])
plt.show()
plt.savefig('TSNE_blob.png', bbox_inches='tight')



# In[30]:



output = pd.DataFrame(model.docvecs.vectors_docs)
output.to_csv("../Data/doc2vec_results.csv")
output
```

Data Filtering Code

```
#!/usr/bin/env python
# coding: utf-8

# # Data Filter
# ### The purpose of this notebook is four-fold:
# 1) Filter data to only the relevant rows
#
# 2) Delete the unnecessary columns
#
# 3) Suitably edit the text to allow for topic modeling
#
# 4) Create new variables to assist with demographic comparisons of topics
```

```
#

# ### Note:
# URL: https://github.com/rudeboybert/JSE_OkCupid
#
# At 141 MB (unzipped) and 53 MB (zipped), this is too large to be uploaded onto Github.
# This notebook is only required to convert the original data to a manageable size.
# If you already have access to the filtered data, 'compressed_okcupid.csv', then you may proceed to th

# In[1]:


get_ipython().system('pip install pyenchant==1.6.6')
get_ipython().system('pip install compound-word-splitter')
get_ipython().system('pip install pyspellchecker')
get_ipython().system('pip install wordninja')


# In[2]:


get_ipython().system(' pip install spellchecker')


# In[4]:


#General Purpose Imports
import numpy as np
import pandas as pd
import warnings
#import pyenchant
```

```
#import splitter
from spellchecker import SpellChecker
warnings.filterwarnings('ignore')
from tqdm import tqdm
tqdm.pandas()
import wordninja



# In[5]:



spell = SpellChecker(distance=2)
lolly = ['I', 'love', 'cantilever', 'pecocks']
spell.correction('greatbutt')



# In[18]:



import re

def split_word(compound_word):
    '''
    Takes in compound word
    Splits it into individual words
    Returns string with spaced words
    '''
    sep_words = wordninja.split(compound_word)
    #print('The separated words are {}'.format(sep_words))
    cleaned = ' '.join(sep_words)
    #print(cleaned)
    return cleaned
```

```
def decide_split(word):
    spellcheck = SpellChecker()
    if not spellcheck[word]:
        nearest = spellcheck.correction(word)
        #When there is no valid word, the nearest word
        #is the same as the original
        if word == nearest:
            #print('The compound word is {}'.format(word))
            return split_word(word)
        else:
            #print('The accepted word is {}'.format(word))
            #print(nearest)
            return nearest
    else:
        return word

def split_incorrect(text, punctuation=True):
    '''
    Takes in a long string
    The punctuation parameter checks if punctuation marks are to
    be preserved
    Splits into component words, checks for incorrect spellings
    For incorrect spellings, checks if is possibly compound
    If not, then looks for the closest one word in the dictionary
    Returns the entire text with all words corrected
    '''
    #if punctuation:

    cleaned_words = []
    words= re.split('\s+|\.|\?|,', text)
    #print(words)
    for word in words:
```

```
        checked_word = decide_split(word)
        #print('The checked word is {}'.format(checked_word))
        cleaned_words.append(checked_word)
    final = " ".join(cleaned_words)
    #print(final)
    return final

test_text = 'John had a big cock, it crowedatsunrise every morning'
real_text = 'i really like meeting new people. small-world networks fascinateme.beer is important. with
real_text_2 = "i'm an adventurer first, i take calculatedrisks whenever it seems fun, and i always take
#print(split_incorrect(test_text))
print(split_incorrect(real_text_2))


# In[17]:


import wordninja
#wordninja.split('hammerandtongs')

def split_word(compound_word):
    '''
    Takes in compound word
    Splits it into individual words
    Returns string with spaced words
    '''
    sep_words = wordninja.split(compound_word)
    cleaned = ' '.join(sep_words)
    return cleaned

split_word('missit')
```

```
# In[14]:


#Imports for Text Analytics
from bs4 import BeautifulSoup
import spacy
import textacy
nlp = spacy.load('en_core_web_sm')
en = textacy.load_spacy_lang("en_core_web_sm")
from textacy import TextStats


# In[13]:


df = pd.read_csv('profiles.csv')
must_haves = ['body_type', 'height', 'education', 'ethnicity', 'sex', 'essay0', 'essay9']
df = df.dropna(subset= must_haves)
df = df[(df['sex']=="m")&(df['orientation']=="straight")]
df = df.drop(columns=['essay1', 'essay2', 'essay3','essay4','essay5','essay6','essay7',
                      'essay8', 'essay9', 'income','job','last_online','location','offspring',
                      'orientation','pets','religion','sex','sign','smokes','speaks','status',
                      'diet', 'drinks', 'drugs'])


# ### CREATING NEW COLUMNS
#

# Many of the sections here are taken directly from the following link, with specific modifications
# Taken directly from:
```

```
# https://github.com/UM-CSS/CSSLabs-NLP/blob/master/1_Data_munging.ipynb

# In[18]:



def recode(text, dictionary, default=np.nan):
    '''Function for recoding categories in a column based on exact matches'''
    out = default
    text = str(text)

    for x in dictionary.keys():
        for y in dictionary[x]:
            if y == text: #exact match
                out = x
                return out
    return out

def recode_fuzzy(text, dictionary, default=np.nan):
    '''Function for recoding categories in a column based on partial matches'''
    out = default
    text = str(text)

    for x in dictionary.keys():
        for y in dictionary[x]:
            if y in text: #partial match
                out = x
                return out
    return out



# In[19]:
```

```
ed_levels = {'High School or less': ['dropped out of high school', 'working on high school','graduated
                    'two-year college', 'dropped out of college/university',
                    'high school'],
             'More than High School': ['graduated from college/university',
                    'working on masters program', 'working on ph.d program',
                    'college/university', 'working on law school',
                    'dropped out of masters program',
                    'dropped out of ph.d program', 'dropped out of law school',
                    'dropped out of med school',
                    'graduated from masters program',
                    'graduated from ph.d program',
                    'graduated from law school',
                    'graduated from med school', 'masters program',
                    'ph.d program', 'law school', 'med school']}

#body type
bodies = {'fit': ['fit', 'athletic', 'jacked'],
          'not_fit': ['average', 'thin', 'skinny','curvey', 'a little extra',
                      'full figured', 'overweight', 'rather not say', 'used up']
         }


# In[20]:


df['edu'] = df.education.apply(recode, dictionary=ed_levels,
                                           default='unknown')
df['fit'] = df.body_type.apply(recode, dictionary=bodies,
                                           default='unknown')


# In[21]:
```

```
# race/ethnicity for exact matching
ethn = {'White': ['white', 'middle eastern', 'middle eastern, white'],
        'Asian': ['asian', 'indian', 'asian, pacific islander'],
        'Black': ['black']       }

# race/ethnicityfor fuzzy matching
ethn2 = {'Latinx': ['latin'], 'multiple': [','], np.nan: ['nan']}


# In[22]:


def census_2010_ethnicity(t):
    text = str(t)
    e = recode(text, ethn, default='other')
    if 'other' == e:
        e = recode_fuzzy(text, ethn2, default='other')
    return e

df['race_ethnicity'] = df.ethnicity.apply(census_2010_ethnicity)


# In[23]:


def height_check(inches):
    h = 'not_short'
    if inches <= 69:
        h = 'short'
    return h
df['height'] = pd.to_numeric(df['height'])
```

```
df['height_group'] = df.height.apply(height_check)



# ## TEXT EDITING

# In[24]:



# Some of the essays have just a link in the text. BeautifulSoup sees that and gets
# the wrong idea. This line hides those warnings.
warnings.filterwarnings("ignore", category=UserWarning, module='bs4')
def clean(text):
    if pd.isnull(text):
        t = np.nan
    else:
        t = BeautifulSoup(text, 'lxml').get_text()
        t = t.lower()
        t = t.strip().replace('\n','').replace("\r", " ").replace('\t', '')
        bad_words = ['http', 'www', '\nnan']

        for b in bad_words:
            t = t.replace(b, '')
    if t == '':
        t = np.nan

    return t



# In[25]:



#Clearing out all HTML and unnecessary characters
```

```
df['essay0'] =df['essay0'].progress_apply(clean)
df['essay9'] =df['essay9'].progress_apply(clean)



# In[33]:



#These functions help assess the level of complexity
def get_flesch(text):
    doc = textacy.make_spacy_doc(text, lang=en)
    ts = TextStats(doc)
    try:
        return ts.flesch_kincaid_grade_level
    except ZeroDivisionError:
        return (11.8 * ts.n_syllables) + (0.39 * ts.n_words) - 15.59

def get_npoly(text):
    doc = textacy.make_spacy_doc(text, lang=en)
    ts = TextStats(doc)
    return ts.n_polysyllable_words



# In[28]:



df['long_words'] = df['essay0'].progress_apply(get_npoly)



# In[30]:



df['flesch'] = df['essay0'].progress_apply(get_flesch)
```

```
# In[38]:


df['long_words'].describe()


# In[39]:


df['flesch'].describe()


# In[35]:


df.to_csv('compressed_okcupid.csv')


# In[ ]:


pip3 install pyenchant==1.6.6
```

Non-negative Matrix Factorization Code

```
#!/usr/bin/env python
# coding: utf-8

# ## Topic Modeling and Visualization
```

```
# In[1]:



#General Imports
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from collections import defaultdict
import re
from string import punctuation
from scipy.stats import ttest_ind
from sklearn.metrics import accuracy_score



import nltk
import pandas as pd
#from spacy.en import English

from utils.permutation import print_pvalues
from utils.text_representation import _levels, _multinomial

import spacy
nlp = spacy.load('en_core_web_sm')
#nlp = English(tagger=True, entity=False)

get_ipython().run_line_magic('matplotlib', 'inline')

from sklearn.decomposition import NMF
```

```
# In[ ]:



#Support Functions
# Text Representation



def _levels(demographics, d_levels=None, print_levels=False):
    """The demographic levels to iterate over

    Parameters
    ----------
    demographics : pd.Series
        Demographic labels
    d_levels : list, default None
        The specific demographic levels desired
    print_levels : bool, default False
        Whether to print the demographic levels

    Returns
    -------
    levels : iterable
        The unique (sorted) levels in `demographics`
    """
    levels = demographics.unique()
    if d_levels:
        assert set(d_levels).issubset(levels)
        levels = d_levels
    levels.sort()
    if print_levels:
        print('Levels (in order):', levels, end='\n\n')
    return levels
```

```
def _multinomial(corpus, kwargs):
    """Tokens counts by document using the spaCy tokenizer

    Parameters
    ----------
    corpus : array-like
        A collection of documents
    kwargs : dict or None
        Keyword arguments of variable length

    Returns
    -------
    X : scipy.sparse.csr.csr_matrix
        The multinomial representation shape (n_samples, n_features)
    v : list
        Vocabulary
    """
    if kwargs:
        cv = CountVectorizer(tokenizer=spacy_tokenize, **kwargs)
    else:
        cv = CountVectorizer(tokenizer=spacy_tokenize)
    X = cv.fit_transform(corpus)
    v = cv.get_feature_names()
    return X, v

def print_pvalues(a, b):
    """Wrapper function for printing meand and p-values
    both permutation-based and classical

    Parameters
    ----------
    a : np.ndarray
```

```
        Data for one class or
        ground truth (correct) labels
    b : np.ndarray
        Data for another class or
        predicted labels, as returned by a classifier

    Returns
    -------
    None
    """
    assert isinstance(a, np.ndarray) and isinstance(b, np.ndarray)
    rnd = lambda x: np.round(x, 8)
    permutation = _permute(a, b, 'means')
    classical = ttest_ind(a, b, equal_var=False)[1]
    print("[means] 'a':", rnd(a.mean()), "'b':", rnd(b.mean()))
    print("p-values:")
    print("  [permutation]:", rnd(permutation))
    print("  [classical]:  ", rnd(classical))









##
def nmf_labels(tfidfmatrix, k):
    """For getting the labels (group assignment) associated with
    each sample (user, in this case)

    Parameters
    ----------
```

```
    tfidfmatrix : scipy.sparse.csr.csr_matrix
        The output from calling `TfidfVectorizer` on the users/features data

    k : int
        The number of groupings to create

    Returns
    -------
    labels : np.ndarray
        An array of group assignments of length tfidfmatrix.shape[0] (users)
    """
    H = NMF(n_components=k, random_state=42).fit_transform(tfidfmatrix)
    labels = np.argmax(H, axis=1)
    return labels


def nmf_inspect(tfidfmatrix, feature_names, k_vals=[3, 5, 7, 9], n_words=10):
    """For looping over various values of `k` and printing the
    top `n_words`

    Parameters
    ----------
    tfidfmatrix : scipy.sparse.csr.csr_matrix
        The output from calling `TfidfVectorizer` on the users/features data

    feature_names : list
        The output from calling the `.get_feature_names()` on
        the TfidfVectorizer object

    k_vals : list
        A list of values for `k`, the number of groupings
```

```
    n_words : int
        The top n words to print for each grouping

    Returns
    -------
    None
    """
    for k in k_vals:
        nmf = NMF(n_components=k, random_state=42).fit(tfidfmatrix)
        print(k, end='\n')
        _print_words(nmf, feature_names, n_words)


def subset_df(df, col, vals):
    """Return a subset of `df` based on particular `vals` for `col`

    Parameters
    ----------
    df : pd.DataFrame
        Input DataFrame
    col : str
        Valid column name
    vals : list
        Values to subset on

    Returns
    -------
    subset : pd.DataFrame
        The rows in `df` with values in `val` for `col`
    """
    df = df.copy()
    subset = df[df[col].isin(vals)]
    return subset
```

```
def group_pct(df, demographic):
    """Calculate the percentage of users in each `demographic` level

    Parameters
    ----------
    df : pd.DataFrame
        Where applicable, this should be a subset of the original DataFrame and
        should include a `group` column corresponding to the NMF groupings
    demographic : str
        Valid column name

    Returns
    -------
    by_dg : pd.DataFrame
        Including `demographic` levels and `group` percentages
    """
    df = df.copy()
    by_dg = pd.DataFrame({'count' :
                          df.groupby([demographic, 'group'])['group'].count()}).reset_index()
    by_d = by_dg.groupby(demographic, as_index=False)['count'].sum()
    by_dg = pd.merge(by_dg, by_d, on=demographic)
    by_dg['pct'] = by_dg.count_x / by_dg.count_y
    return by_dg

def feature_vectors(corpus, kwargs=None):
    """Multinomial and TF-IDF representations

    Paramaters
    ----------
    corpus : array-like
        A collection of documents
    kwargs : dict, default None
```

```
        Keyword arguments of variable length
        See sklearn.feature_extraction.text.CountVectorizer
        for accepted keyword arguments

    Returns
    -------
    count : scipy.sparse.csr.csr_matrix
        The multinomial representation shape (n_samples, n_features)
    tfidf : scipy.sparse.csr.csr_matrix
        The tf-idf representation
    vocab : list
        Vocabulary
    """
    assert isinstance(corpus, (list, pd.Series))
    count, vocab = _multinomial(corpus, kwargs)
    tfidf = _tfidf(count)
    return count, tfidf, vocab


def tagger(doc):
    """For tagging a document
    Yields a (token, part-of-speech) tag tuple

    Parameters
    ----------
    doc : str
        A document with tokens to tag

    Yields
    ------
    tuple
        (token, tag)
```

```
    """
    text = nlp(doc)
    for sent in text.sents:
        for token in sent:
            yield (str(token), str(token.pos_))

def tag_corpus(corpus):
    """For tagging corpus document tokens

    Parameters
    ----------
    corpus : array-like
        A collection of documents

    Returns
    -------
    tagged : list
        (token, tag) tuples
    """
    assert isinstance(corpus, (list, pd.Series))
    tagged = []
    for doc in corpus:
        tagged.extend(tagger(doc))
    return tagged

def pos_tokens(tagged, pos):
    """Extract particular part-of-speech tokens

    Parameters
    ----------
    tagged : list
        (token, tag) tuples
```

```
    pos : str
        A valid part-of-speech tag

    Returns
    -------
    list

    Notes
    -----
    The available tags are:
        ADJ, ADP, ADV, AUX, CONJ, DET, INTJ, NOUN, NUM, PART,
        PRON, PROPN, PUNCT, SCONJ, SYM, VERB, X, EOL, SPACE
    Source: https://spacy.io/docs#token-postags
    """
    return [t for t, p in tagged if p == pos]

def _pos_freq(doc):
    """Part of speech frequencies for individual documents

    Parameters
    ----------
    doc : str
        A document with tokens to tag

    Returns
    -------
    pos : dict
        With counts by tag
    """
    pos = defaultdict(float)
    for _, p in tagger(doc):
        pos[p] += 1
```

```
    return pos


def pos_df(corpus):
    """Create a DataFrame of part of speech
    frequencies for a corpus of documents

    Parameters
    ----------
    corpus : array-like
        A collection of documents

    Returns
    -------
    df : pd.DataFrame
    """
    assert isinstance(corpus, (list, pd.Series))
    pos_dfs = []
    for doc in corpus:
        frequencies = pd.DataFrame(_pos_freq(doc), index=[0])
        pos_dfs.append(frequencies)
    df = pd.concat(pos_dfs, ignore_index=True)
    df.fillna(0.0, inplace=True)
    return df


def pos_normalize(df):
    """Normalize (row-wise) part-of-speech frequencies

    Parameters
    ----------
    df : pd.DataFrame
        `pos_df()` DataFrame
```

```
    Returns
    -------
    pd.DataFrame
    """
    assert isinstance(df, pd.DataFrame)
    return (df.T / df.sum(axis=1)).T


def _arrs_pos(df_orig, df_pos, demographic, pos,
              d_levels=None, print_levels=False):
    """Individual part-of-speech
    arrays for a particular demographic

    Parameters
    ----------
    df_orig : pd.DataFrame
        The DataFrame from which `df_pos` was created
    df_pos : pd.DataFrame
        The part-of-speech DataFrame
    demographic : str
        A valid demographic-data column in `df_orig`
    pos : str
        A column in `df_pos` corresponding
        to a part of speech
    d_levels : list, default None
        The specific demographic levels desired
    print_levels : bool, default False
        Whether to print the demographic levels

    Returns
    -------
    arrs : tuple of np.arrays
        The corresponding `pos` values for each `demographic`
```

```
    """
    df_pos = df_pos.copy() # so we don't modify it
    df_pos[demographic] = df_orig[demographic].values
    levels = _levels(df_orig[demographic], d_levels, print_levels)
    arrs = []
    for d in levels:
        arr = df_pos[df_pos[demographic] == d][pos].values
        n = arr.shape[0]
        if n < 0.1 * df_pos.shape[0]:
            print("Warning: '" + d +
                  "' category has less than 10% of observations (" +
                  str(n) + ")")
        arrs.append(arr)
    return tuple(arrs)


def pos_by_split(df_orig, df_pos, demographic, pos=None,
                 d_levels=None, print_levels=False):
    """Wrapper for handling multiple parts-of-speech with `_arrs_pos()`

    Parameters
    ----------
    df_orig : pd.DataFrame
        The DataFrame from which `df_pos` was created
    df_pos : pd.DataFrame
        The part-of-speech DataFrame
    demographic : str
        A valid demographic-data column in `df_orig`
    pos : list, default None
        Parts-of-speech to compare
    d_levels : list, default None
        The specific demographic levels desired
    print_levels : bool, default False
```

```
        Whether to print the demographic levels

    Returns
    -------
    None

    Notes
    -----
    The number of unique values in `demographic` must be two
    """
    assert (isinstance(df_orig, pd.DataFrame) and
            isinstance(df_pos, pd.DataFrame))
    assert df_orig.shape[0] == df_pos.shape[0]
    assert demographic in df_orig.columns
    assert set(pos).issubset(df_pos.columns)
    for p in pos:
        a, b = _arrs_pos(df_orig, df_pos, demographic, p, d_levels, print_levels)
        print(p)
        print_pvalues(a, b)
        print()

def load_words(path):
    """To load profane and slang words

    Parameters
    ----------
    path : str
        Relative or absolute filepath

    Returns
    -------
    list
```

```
    """
    assert isinstance(path, str)
    with open(path, 'r') as f:
        return list(set([w.rstrip() for w in f.readlines()]))


def _contains_n(words, corpus):
    """Count the number of times a document contains particular words

    Parameters
    ----------
    words : list
        Words to check for
    corpus : array-like
        A collection of documents

    Returns
    -------
    np.ndarray
        Number of tokens by document
    """
    X, _ = _multinomial(corpus, {'vocabulary' : words})
    return X.toarray().sum(axis=1)


def contains(words, corpus):
    """Determine whether a document contains particular words

    Parameters
    ----------
    words : list
        Words to check for
    corpus : array-like
        A collection of documents
```

```
    Returns
    -------
    n_words : np.ndarray
        Binary representation
    """
    assert isinstance(words, list)
    assert isinstance(corpus, (list, pd.Series))
    n_words = _contains_n(words, corpus)
    n_words[n_words > 0] = 1
    return n_words


def _token_counts(a, b, pos):
    """Create a DataFrame of `pos` token frequencies for particular
    demographic splits. `a` and `b` are lists of token, part-of-speech
    tuples (output from `tag_corpus()`).

    Parameters
    ----------
    a : list
        token, pos tuples
    b : list
        token, pos tuples
    pos : str
        A valid part-of-speech tag

    Returns
    -------
    df : pd.DataFrame
        With row 0 corresponding to `a` and row 1 to `b`
    """
    pos_a = nltk.FreqDist(pos_tokens(a, pos))
    pos_b = nltk.FreqDist(pos_tokens(b, pos))
```

```
    df_a = pd.DataFrame(pos_a, index=[0])
    df_b = pd.DataFrame(pos_b, index=[0])
    df = pd.concat([df_a, df_b], ignore_index=True)
    df.fillna(0, inplace=True)
    return df

def print_terms(df, n):
    measure = df.columns[0]
    print(" | ".join(df.sort_values(measure, ascending=False)[:n].index))
    print()
    print(" | ".join(df.sort_values(measure)[:n].index))

def top_terms(a, b, pos, fn, n):
    """Print the top `n` tokens (resulting from `fn`) for
    demographic splits associated with `a` and `b`

    Parameters
    ----------
    a : list
        token, pos tuples
    b : list
        token, pos tuples
    pos : str
        A valid part-of-speech tag
    fn : callable
        Either `diff_prop` of `log_odds_ratio`
    n : int
        Number of terms to print for each demographic split

    Returns
    -------
    None
```

```
    """
    df = _token_counts(a, b, pos)
    df = fn(df.values, df.columns.tolist())
    print_terms(df, n)


# #### First, we generate the topics and assign some meaning to them

# In[ ]:


df = pd.read_csv('compressed_okcupid.csv')


# In[ ]:


#The major part of the algorithm- can take some time
specs = {'stop_words' : 'english', 'ngram_range' : (1, 3), 'min_df' : 0.005}
counts, tfidf, vocab = feature_vectors(df.essay0, specs)


# In[ ]:


K = 25
nmf_inspect(tfidf, vocab, k_vals=[K], n_words=50)


# In[ ]:
```

```
#These labels are based on the categories as assessed by Juan Shishido, then modified by me
labels=['Reach Out!','Relocated', 'About Me', 'Hesitation', 'Casual', 'The City',
       'Novelty', 'Cool', 'Likes', 'Passions', 'Easy Going', 'Region', 'Seeking', 'Thoughts', 'Fun', 'N
        'Travel','Self-summary', 'Nots', 'Growing Up','Carpe Diem', 'Good Company','Hobbies',
        'Cultural Interests', 'Ambitious']

label_dict = {}
for c, value in enumerate(labels):
    label_dict[c] = value
print(label_dict)



# #### Next, we find a way of calculating and visualizing these topic distributions across our 4 chosen

# In[ ]:



def split_by_demog(model, feature_names, n_top_words):
    """For printing the `n_top_words` for each grouping

    Parameters
    ----------
    model : sklearn.decomposition.nmf.NMF
        The NMF object

    feature_names : list
        The output from calling `TfidfVectorizer` on the users/features data

    n_top_words : int
        The top n words to print for a particular grouping

    Returns
```

```
    -------
    None
    """
    for topic_idx, topic in enumerate(model.components_):
        print("Group %d:" % topic_idx)
        print(" | ".join([feature_names[i]
            for i in topic.argsort()[ : -n_top_words-1 : -1]]))
        print()
    print()


# In[ ]:


def get_label(group_num):
    return label_dict[group_num]

def format_df(df, demog, tfidf):
    df['group'] = nmf_labels(tfidf, k=K)
    subset = subset_df(df, demog, df[demog].unique())
    grouped = group_pct(subset, demog)
    percent_only = grouped.drop(['count_x', 'count_y'], axis=1)
    #percent_only
    pivoted = percent_only.pivot(index='group', columns=demog)
    pivoted['max_value'] = pivoted.max(axis=1)
    ordered_df = pivoted.sort_values(by='max_value', ascending=True)
    #Getting rid of the multi-line index
    ordered_df.columns = ordered_df.columns.droplevel(0)
    ordered_df = ordered_df.reset_index().rename_axis(None, axis=1)
    #Renaming the max
    ordered_df = ordered_df.rename(columns={'':'max'})
    #Linking to label
```

```
    ordered_df['label'] = ordered_df['group'].apply(get_label)
    return ordered_df


# In[ ]:


height_df, race_df, edu_df, fit_df= format_df(df, 'height_group', tfidf),
                                    format_df(df, 'race_ethnicity', tfidf),
                                    format_df(df, 'edu', tfidf),
                                    format_df(df, 'fit', tfidf)


# In[ ]:


#Plot for Education Levels
ordered_df = edu_df
import matplotlib.patches as mpatches

my_range=range(1,len(ordered_df.index)+1)
fig, ax = plt.subplots(figsize=(18, 15))
ttl = ax.title
ttl.set_position([.5, 1.05])

# The vertival plot is made using the hline function
# I load the seaborn library only to benefit the nice looking feature
import seaborn as sns
plt.hlines(y=my_range, xmin=0, xmax=ordered_df['max'], color='Gray')
plt.plot(ordered_df['High School or less'], my_range, "o", markersize=20, color='blue')
plt.plot(ordered_df['More than High School'], my_range, "o", markersize=20, color='red')
plt.rc('ytick',labelsize=28)
```

```
plt.rc('xtick',labelsize=28)
# Add titles and axis names
plt.yticks(my_range, ordered_df['label'])
plt.title("Topics in OkCupid Male Self-Introductions Across Education Levels", loc='center', fontsize=4
plt.xlabel('Proportion of Users Using This Topic', fontsize=32)
plt.ylabel('Topics Inferred from Essay',fontsize=32)
maroon_patch = mpatches.Patch(color='red', label='More than High School')
blue_patch = mpatches.Patch(color='blue', label='Less than High School')
plt.legend(handles=[maroon_patch, blue_patch], loc='center right', fontsize='xx-large', borderpad=2)
plt.savefig('opinions.png', bbox_inches='tight')



# In[ ]:



#Plot for Fitness Levels
ordered_df = fit_df
import matplotlib.patches as mpatches

my_range=range(1,len(fit_df.index)+1)
fig, ax = plt.subplots(figsize=(18, 15))
ttl = ax.title
ttl.set_position([.5, 1.05])

# The vertival plot is made using the hline function
# I load the seaborn library only to benefit the nice looking feature
import seaborn as sns
plt.hlines(y=my_range, xmin=0, xmax=ordered_df['max'], color='Gray')
plt.plot(ordered_df['fit'], my_range, "o", markersize=20, color='blue')
plt.plot(ordered_df['not_fit'], my_range, "o", markersize=20, color='red')
plt.rc('ytick',labelsize=28)
plt.rc('xtick',labelsize=28)
```

```
# Add titles and axis names
plt.yticks(my_range, ordered_df['label'])
plt.title("Topics in OkCupid Male Self-Introductions Across Fitness Levels", loc='center', fontsize=40)
plt.xlabel('Proportion of Users Using This Topic', fontsize=32)
plt.ylabel('Topics Inferred from Essay',fontsize=32)
maroon_patch = mpatches.Patch(color='red', label='Fit')
blue_patch = mpatches.Patch(color='blue', label='Not Fit')
plt.legend(handles=[maroon_patch, blue_patch], loc='center right', fontsize='xx-large', borderpad=2)
plt.savefig('fit.png', bbox_inches='tight')



# In[ ]:



#The Plot for Height
ordered_df = height_df
import matplotlib.patches as mpatches

my_range=range(1,len(ordered_df.index)+1)
fig, ax = plt.subplots(figsize=(18, 15))
ttl = ax.title
ttl.set_position([.5, 1.05])

# The vertival plot is made using the hline function
# I load the seaborn library only to benefit the nice looking feature
import seaborn as sns
plt.hlines(y=my_range, xmin=0, xmax=ordered_df['max'], color='Gray')
plt.plot(ordered_df['short'], my_range, "o", markersize=20, color='blue')
plt.plot(ordered_df['not_short'], my_range, "o", markersize=20, color='red')
plt.rc('ytick',labelsize=28)
plt.rc('xtick',labelsize=28)
# Add titles and axis names
```

```
plt.yticks(my_range, ordered_df['label'])
plt.title("Topics in OkCupid Male Self-Introductions Across Height Groups", loc='center', fontsize=40)
plt.xlabel('Proportion of Users Using This Topic', fontsize=32)
plt.ylabel('Topics Inferred from Essay',fontsize=32)
maroon_patch = mpatches.Patch(color='red', label='Short')
blue_patch = mpatches.Patch(color='blue', label='Not Short')
plt.legend(handles=[maroon_patch, blue_patch], loc='center right', fontsize='xx-large', borderpad=2)
plt.savefig('height.png', bbox_inches='tight')



# In[ ]:



# The Plot for Races
ordered_df = race_df
my_range=range(1,len(ordered_df.index)+1)
fig, ax = plt.subplots(figsize=(18, 15))
ttl = ax.title
ttl.set_position([.5, 1.05])

# The vertival plot is made using the hline function
# I load the seaborn library only to benefit the nice looking feature
import seaborn as sns
plt.hlines(y=my_range, xmin=0, xmax=ordered_df['max'], color='Gray')
plt.plot(ordered_df['White'], my_range, "o", markersize=20, color='blue')
plt.plot(ordered_df['Black'], my_range, "o", markersize=20, color='red')
plt.plot(ordered_df['Asian'], my_range, "o", markersize=20, color='green')
plt.plot(ordered_df['Latinx'], my_range, "o", markersize=20, color='cyan')
plt.plot(ordered_df['multiple'], my_range, "o", markersize=20, color='magenta')
plt.rc('ytick',labelsize=28)
plt.rc('xtick',labelsize=28)
# Add titles and axis names
```

```
plt.yticks(my_range, ordered_df['label'])
plt.title("Topics in OkCupid Male Self-Introductions Across Racial Groups", loc='center', fontsize=40)
plt.xlabel('Proportion of Users Using This Topic', fontsize=32)
plt.ylabel('Topics Inferred from Essay',fontsize=32)
blue_patch = mpatches.Patch(color='blue', label='White')
maroon_patch = mpatches.Patch(color='red', label='Black')
green_patch = mpatches.Patch(color='green', label='Asian')
cyan_patch = mpatches.Patch(color='cyan', label='Latinx')
magenta_patch = mpatches.Patch(color='magenta', label='multiple')
plt.legend(handles=[maroon_patch, blue_patch, green_patch, cyan_patch, magenta_patch], loc='center righ
plt.savefig('race.png', bbox_inches='tight')
```

LDA Topic modeling Code

```
## ----setup-one,
## include=FALSE-------------------------------------------------
knitr::opts_chunk$set(echo = TRUE)



## ----setup-----------------------------------------------------------------
library(stm)
library(quanteda)
library(topicmodels)
library(tidytext)
library(ggplot2)
library(dplyr)
library(tidyr)
library(scales)
library(tm)
library(grid)
library(wordcloud)
library(wordcloud2)
```

```
library(tidyverse)


# sample data with essay 0 and demo clusters
essay <- read.csv("compressed_okcupid.csv")



## ----wrangling-----------------------------------------------------------------
## convert factor type to character
essay$essay0 <- as.character(essay$essay0)

list_of_values <- c("love", "people", "life", "time",
    "enjoy", "friends", "fun", "people", "music")

"%ni%" <- Negate("%in%")

tidy_essay <- essay %>% mutate(race_ethnicity = factor(race_ethnicity,
    levels = unique(race_ethnicity))) %>% mutate(line = row_number()) %>%
    unnest_tokens(word, essay0) %>% anti_join(stop_words) %>%
    filter(word %ni% list_of_values)




## ----exploring-----------------------------------------------------------------
essay_tf_idf <- tidy_essay %>% count(race_ethnicity,
    word, sort = TRUE) %>% bind_tf_idf(word, race_ethnicity,
    n) %>% arrange(-tf_idf) %>% group_by(race_ethnicity) %>%
    top_n(15) %>% ungroup



essay_tf_idf %>% mutate(word = reorder_within(word,
    tf_idf, race_ethnicity)) %>% ggplot(aes(word, tf_idf,
    fill = race_ethnicity)) + geom_col(alpha = 0.8,
```

```
    show.legend = FALSE) + facet_wrap(~race_ethnicity,
    scales = "free", ncol = 3) + scale_x_reordered() +
    coord_flip() + theme(strip.text = element_text(size = 11)) +
    labs(x = NULL, y = "tf-idf", title = "Highest tf-idf words in Each Demographic Clusters",
        subtitle = "Individual cluster have different words to represent themselves")



## ----dt-mat-------------------------------------------------------------------
essay_dfm <- tidy_essay %>% count(race_ethnicity, word,
    sort = TRUE) %>% cast_dfm(race_ethnicity, word,
    n)

essay_sparse <- tidy_essay %>% count(race_ethnicity,
    word, sort = TRUE) %>% cast_sparse(race_ethnicity,
    word, n)



## ----struc-topic-mod----------------------------------------------------------
topic_num <- 15

essay_topic_model <- stm(essay_dfm, K = topic_num,
    verbose = FALSE, init.type = "Spectral", prevalence = ~essay$race_ethnicity)

summary(essay_topic_model)



## ----plot-stm-----------------------------------------------------------------
plot.STM(essay_topic_model, type = "labels")
```

```
## ----plot-tts-----------------------------------------------------------------
plot(essay_topic_model, type = "summary", text.cex = 0.8)



## ----topic-contrast-----------------------------------------------------------
plot(essay_topic_model, type = "perspectives", topics = c(1,
    3))



## ----topic-prop---------------------------------------------------------------
plot(essay_topic_model, type = "hist")



## ----topic-net----------------------------------------------------------------
mod.out.corr <- topicCorr(essay_topic_model)
plot(mod.out.corr)



## ----wc-topic-1---------------------------------------------------------------
cloud(essay_topic_model, topic = 2)



## ----wc-topic-2---------------------------------------------------------------
cloud(essay_topic_model, topic = 4)



## ----word-prob-dist-----------------------------------------------------------
td_beta <- tidy(essay_topic_model)

td_beta %>% group_by(topic) %>% top_n(10, beta) %>%
    ungroup() %>% mutate(topic = paste0("Topic ", topic),
    term = reorder_within(term, beta, topic)) %>% ggplot(aes(term,
```

```
    beta, fill = as.factor(topic))) + geom_col(alpha = 0.8,
    show.legend = FALSE) + facet_wrap(~topic, scales = "free_y") +
    coord_flip() + scale_x_reordered() + labs(x = NULL,
    y = expression(beta), title = "Highest word probabilities for each topic",
    subtitle = "Different words are associated with different topics")



## ----gamma-prob-dist-----------------------------------------------------------
td_gamma <- tidy(essay_topic_model, matrix = "gamma",
    document_names = rownames(essay_dfm))

ggplot(td_gamma, aes(gamma, fill = as.factor(topic))) +
    geom_histogram(alpha = 0.8, show.legend = FALSE) +
    facet_wrap(~topic, ncol = 3) + labs(title = "Distribution of document probabilities for each topic"
    subtitle = "Each topic is associated with 1-2 clusters",
    y = "Number of stories", x = expression(gamma))
```

Structural topic modeling Code

```
## ----setup,
## include=FALSE-----------------------------------------------------
knitr::opts_chunk$set(echo = FALSE)
library(stm)
library(quanteda)
library(topicmodels)
library(tidytext)
library(ggplot2)
library(dplyr)
library(tidyr)
library(scales)
library(tm)
library(grid)
```

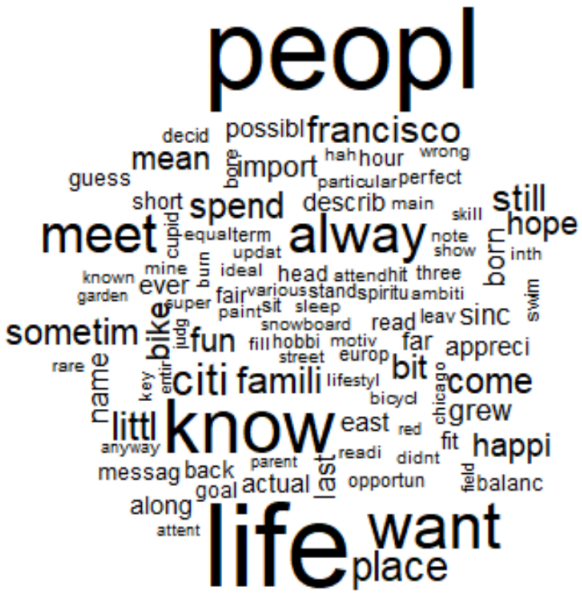
```r
library(wordcloud)
library(wordcloud2)
library(tidyverse)
library(igraph)
library(stmCorrViz)



## ----data,
## echo=FALSE-------------------------------------------------------------
setwd("C:/Users/lliu9/Desktop/UML_Project/unsupervised-dating/Data")
# cleaned okcupid data
data <- read_csv("final_okcupid.csv")

# data <- sample_n(essay, 100)
data <- data[c("clean_text", "fit", "edu", "height_group",
    "race_ethnicity", "dbscan_cluster")]



## ----clean--------------------------------------------------------------------
processed <- textProcessor(data$clean_text, metadata = data)
out <- prepDocuments(processed$documents, processed$vocab,
    processed$meta, lower.thresh = 0)
docs <- out$documents
vocab <- out$vocab
meta <- out$meta



## ----stm----------------------------------------------------------------------
Fit <- stm(documents = out$documents, vocab = out$vocab,
    K = 9, prevalence = ~fit + edu + height_group +
        race_ethnicity + dbscan_cluster, max.em.its = 50,
    data = out$meta, init.type = "Spectral", verbose = FALSE)
```

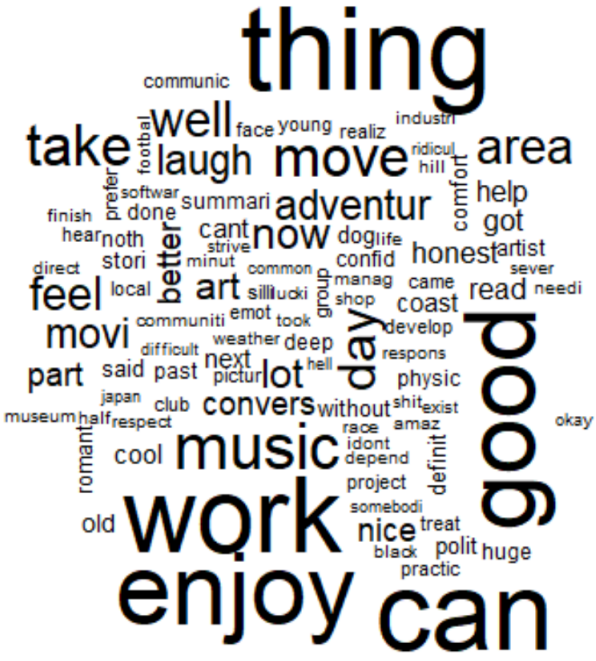
```r
## ----label-------------------------------------------------------------------
labelTopics(Fit, c(1:9))



## ----estimate----------------------------------------------------------------
prep <- estimateEffect(~fit + edu + height_group +
    race_ethnicity + dbscan_cluster, Fit, meta = out$meta,
    uncertainty = "Global")



## ----print-------------------------------------------------------------------
for (i in seq(1:9)) {
    print(summary(prep, topic = i))
}



## ----summary-----------------------------------------------------------------
## jpeg('toptopics.jpeg')
plot(Fit, type = "summary", xlim = c(0, 0.3))
# dev.off



## ----labels------------------------------------------------------------------
jpeg("top3topics.jpeg")
plot(Fit, type = "labels", topics = c(5, 4, 7))
dev.off()



## ----labels2-----------------------------------------------------------------
jpeg("bottom3topics.jpeg")
plot(Fit, type = "labels", topics = c(1, 2, 9))
```

```r
dev.off()



## ----hist-----------------------------------------------------------------
jpeg("hist.jpeg")
plot(Fit, type = "hist")
dev.off()



## ----perspectives---------------------------------------------------------
jpeg("comparison.jpeg")
plot(Fit, type = "perspectives", topics = c(6, 8))
dev.off()




## ----quality--------------------------------------------------------------
jpeg("topicQuality.jpeg")
topicQuality(model = Fit, documents = docs)
dev.off()



## ----word-cloud-----------------------------------------------------------
cloud(Fit, topic = 1)



## ----cloud----------------------------------------------------------------
for (i in seq(1:9)) {
    name <- paste("topic", str(i), ".jpg", sep = "")
    jpeg(name)
    cloud(Fit, topic = i, scale = c(4, 0.2))
    dev.off
```

```
}



## ----network-----------------------------------------------------------------
## Positive correlations between topics indicate
## that both topics are likely to be discussed
## within a document.
mod.out.corr <- topicCorr(Fit)
# Graphical display of topic correlations.
jpeg("network.jpeg")
plot(mod.out.corr)
dev.off()
```

Word Clouds Visualization (Code inside the previous chunk)

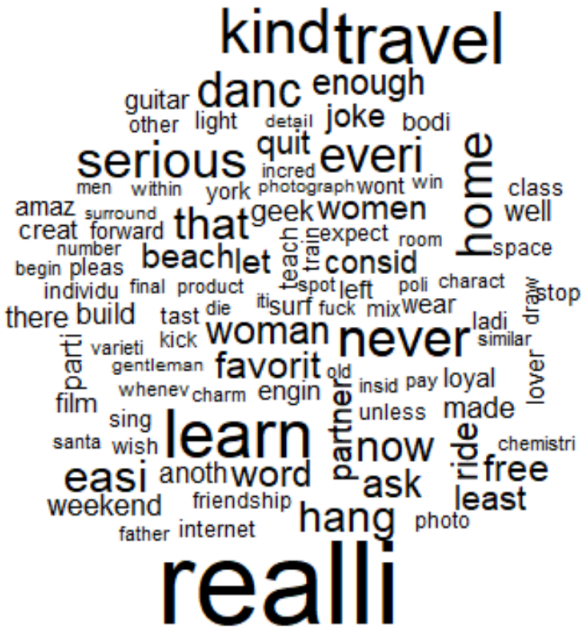



thing
communic
well
face
young
realiz
industri
take
laugh
move
area
ridicul
hill
comfort
help
got
prefer
softwar
summari
adventur
finish
done
hear
noth
cant
now
dog
life
strive
better
stori
minut
confid
honest
artist
direct
common
sever
feel
local
art
manag
came
read
needi
group
shop
coast
movi
communiti
emot
took
develop
weather
deep
difficult
respons
part
said
past
next
pictur
lot
hell
day
physic
good
japan
club
convers
without
shit
exist
museum
half
respect
race
amaz
okay
romant
cool
music
idont
depend
definit
project
somebodi
old
work
nice
treat
black
polit
huge
practic
enjoy
can


chanc
yet
degre
volunt
relat
improv
american
healthi
back
long
talk
south
almost
hair
dark
sort
live
explor
join
sound
marri
somewhat
giant
other
mayb
lie
west
human
chat
reason
deal
listen
cultur
drive
school
lot
girl
may
outgo
abl
futur
admit
settl
typic
coffe
student
caus
fine
month
truth
star
complet
involv
togeth
taken
eat
case
sensit
sarcast
hate
exercis
visit
seen
quick
trail
tell
tech
wit
sure
weird
easygo
restaur
side
first
busi
five
thank
chill
later
god
coupl
respect
land
soon
call
tuesl
dri
cours
intern
speak
watch
nerd
strong
give
fashion
just
town
look
activ
friend


best
hey
also
classic
isnt
despit
havent
like
factels
offer
inspir
blah
world
true
close
cook
even
other
hous
let
check
honesti
ball
work
often
style
everyon
oakland
adult
yoga
awesom
meal
warm
occas
most
easili
hike
constant
think
loud
ocean
fish
tri
late
sail
yes
good
crazi
night
focus
boy
french
self
simpl
air
singl
pleasur
old
san
limit
ofth
use
america
fli
north
curios
water
concert
sarcasm
photographi
goe
fun
master
theyr
public
smart
wake
tree
graduat
across
ive
need
daili
comput
independ
cal
stay
passion
climb
date
state
someon
profil
problem
see
search
form
orient
intens


new
perspect
get
keep
two
profession
soccer
might
truli
basic
bring
believ
introvert
express
rather
support
hard
becom
fire
view
say
witti
figur
spontan
white
camp
middl
fascin
jump
grand
open
idea
guy
humbl
wait
big
man
profil
challeng
pursu
fan
situat
hello
dude
road
kinda
ski
mei
went
smile
excit
experi
engag
sun
straight
lost
pass
touch
doesnt
except
walk
shi
trip
present
qualiti
anyon
base
box
golf
lol
answer
find
whole
record
rock
basebal
success
whos
tri
plan
gym
everyday
anyth
random
run
although
type
perhap
afraid
philosophi
winter
comedi
look
outdoor
bay
origin
year
berkeley
backpack


kind
travel
danc
enough
guitar
other
light
detail
joke
bodi
quit
serious
incred
everi
home
men
within
york
photograph
wont
win
class
amaz
surround
geek
women
well
creat
forward
that
expect
room
space
number
beach
let
teach
train
consid
begin
pleas
individu
final
product
spot
left
poli
charact
stop
draw
there
build
tast
die
iti
surf
fuck
mix
wear
ladi
similar
woman
never
parti
varieti
kick
gentleman
favorit
old
insid
pay
loyal
lover
film
whenev
charm
engin
partner
unless
made
sing
learn
now
ride
chemistri
santa
wish
free
easi
anoth
word
ask
least
weekend
friendship
hang
photo
father
internet
realli

References

Bruch, Elizabeth E, and MEJ Newman. 2018. "Aspirational Pursuit of Mates in Online Dating Markets." *Science Advances* 4 (8): eaap9815.

———. 2019. "Structure of Online Dating Markets in Us Cities." *Sociological Science* 6: 219–34.

Bruning, Roger H, Gregory J Schraw, and Royce R Ronning. 1999. *Cognitive Psychology and Instruction*. ERIC.

Burke, Rose. 2019. "Snapchat's Gender Switch Filter Shows Men What Online Dating Is Like for Women." https://socialnewsdaily.com/86938/snapchats-gender-switch-filter-shows-men-what-online-dating-is-like-for-women/.

Cacioppo, John T, Stephanie Cacioppo, Gian C Gonzaga, Elizabeth L Ogburn, and Tyler J VanderWeele. 2013. "Marital Satisfaction and Break-Ups Differ Across on-Line and Off-Line Meeting Venues." *Proceedings of the National Academy of Sciences* 110 (25): 10135–40.

Dion, Karen, Ellen Berscheid, and Elaine Walster. 1972. "What Is Beautiful Is Good." *Journal of Personality and Social Psychology* 24 (3): 285.

Duck, Steve. 2000. *The Social Psychology of Personal Relationships*. Vol. 3. John Wiley & Sons Inc.

Fiore, Andrew T, and Judith S Donath. 2005. "Homophily in Online Dating: When Do You Like Someone Like Yourself?" In *CHI'05 Extended Abstracts on Human Factors in Computing Systems*, 1371–4. ACM.

Fiore, Andrew T, Lindsay Shaw Taylor, Gerald A Mendelsohn, and Marti Hearst. 2008. "Assessing Attractiveness in Online Dating Profiles." In *Proceedings of the Sigchi Conference on Human Factors in Computing Systems*, 797–806. ACM.

Fiore, Andrew T, Lindsay Shaw Taylor, Xiaomeng Zhong, Gerald A Mendelsohn, and Coye Cheshire. 2010. "Who's Right and Who Writes: People, Profiles, Contacts, and Replies in Online Dating." In *2010 43rd Hawaii International Conference on System Sciences*, 1–10. IEEE.

Gibbs, Jennifer L, Nicole B Ellison, and Rebecca D Heino. 2006. "Self-Presentation in Online Personals: The Role of Anticipated Future Interaction, Self-Disclosure, and Perceived Success in Internet Dating." *Communication Research* 33 (2): 152–77.

Goffman, E. 1975. *The Presentation of Self in Everyday Life*. Pelican Book. Penguin Books. https://books.google.com/books?id=tYvNnQEACAAJ.

Heino, Rebecca D, Nicole B Ellison, and Jennifer L Gibbs. 2010. "Relationshopping: Investigating the Market Metaphor in Online Dating." *Journal of Social and Personal Relationships* 27 (4): 427–47.

Hitsch, Gunter J, Ali Hortaçsu, and Dan Ariely. 2010. "Matching and Sorting in Online Dating." *American Economic Review* 100 (1): 130–63.

Kim, Albert Y, and Adriana Escobedo-Land. 2015. "OkCupid Data for Introductory Statistics and Data Science Courses." *Journal of Statistics Education* 23 (2).

Lin, Ken-Hou, and Jennifer Lundquist. 2013. "Mate Selection in Cyberspace: The Intersection of Race, Gender, and Education." *American Journal of Sociology* 119 (1): 183–215.

Nagarajan, Meenakshi, and Marti A Hearst. 2009. "An Examination of Language Use in Online Dating Profiles." In *Third International Aaai Conference on Weblogs and Social Media*.

Papacharissi, Zizi. 2010. *A Networked Self: Identity, Community, and Culture on Social Network Sites*. Routledge.

———. 2018. *A Networked Self and Love*. Routledge.

Schwartz, Pepper, and Nicholas Velotta. 2018. "Online Dating: Changing Intimacy One Swipe at a Time?" In *Families and Technology*, 57–88. Springer.

Shishido, Juan, Jaya Narasimhan, and Matar Haller. 2016. "Tell Me Something I Don't Know: Analyzing Okcupid Profiles."

Stevens, Gillian, Dawn Owens, and Eric C Schaefer. 1990. "Education and Attractiveness in Marriage Choices." *Social Psychology Quarterly*, 62–70.