THREE GENERATIONS SEEKING ROMANCE: A DATA SCIENCE APPROACH TO PROFILING ONLINE DATERS ACCORDING TO AGE & GENERATION

MACHINE LEARNING FUNDAMENTALS CAPSTONE PROJECT

BY MARIO LUIS “LOUIE” BALDERRAMA

THE TWO BIG QUESTIONS

- *Can your dating profile predict your age?*
- *Can your dating profile predict the generation you belong to?*

Using a dataset of almost 60,000 anonymized entries from the dating platform *OKCupid*, we'll explore the above questions using supervised machine learning techniques. We'll examine different aspects or features of user profiles and see which ones we'll integrate into our models to achieve the most satisfactory scores.

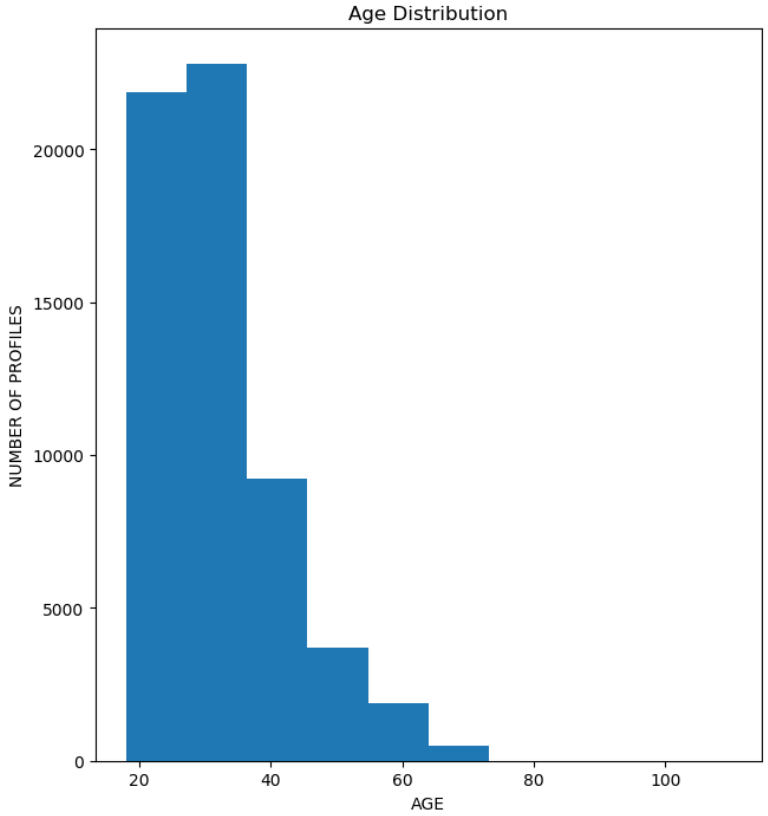
We'll use regression techniques to tackle the first question and classification models to address the second.

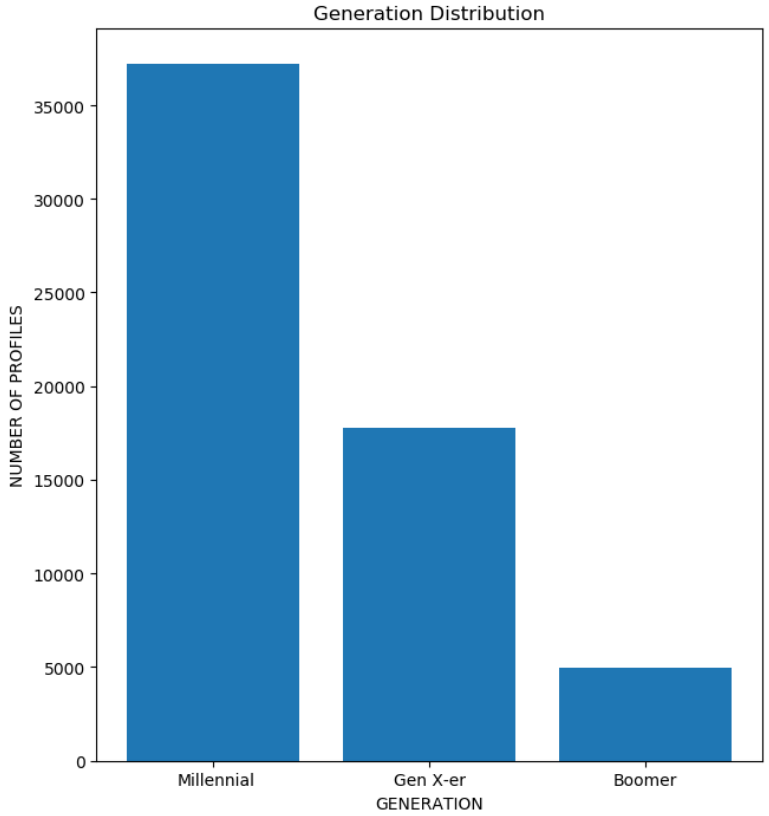
FEATURE ENGINEERING

WE'LL EXPLORE SOME OF THE FEATURES OF THE PROFILES AND SEE WHAT MAKES SENSE TO UTILIZE IN OUR MODEL.

FEATURE – GENERATION

Central to our exploration is the ages of the user profiles. The distributions below confirm what recent research has already told us: that younger people are more inclined to finding love online. But that doesn't mean the older folks aren't getting in on it!





CODE: `plt.hist(df.age, range=(df.age.min(), df.age.max()))`

FEATURE – GENERATION

By exploring the “last_online” feature’s min and max, the OKCupid dataset could be inferred to be dated around 2011 or 2012. At the time, a “Millennial” would be aged from 18 to 32, a “Gen X-er” 33 to 47, and “Boomers” from 48 to 70 years old.

```
generation = []
for i in range(len(df)):
    if 18 <= df["age"].iloc[i] <= 32:
        generation.append(0)
    elif 32 < df["age"].iloc[i] <= 47:
        generation.append(1)
    else:
        generation.append(2)
df["generation"] = generation
```

In Pandas, I usually use the *df.apply* function and execute *lambdas* in order to perform modifications on the entries. However, we have 3 discrete values to this scale and the *df.apply* function is limited to just one “if-else” condition (thus a max of two conditions) – it does not allow *elifs*. To get around this, I created a list and based my “generation” series on this.

FEATURE – DRINKS, DRUGS, SMOKES

These readily available features were convenient to use yet still speaks a lot about the profiles. Statistics would say that younger people are more likely to take risks and perhaps even engage in risky behaviors like smoking, drinking, or getting high.

The strings were translated into a scale that represents the frequency of use:

```
df["drinks"] = df["drinks"].map({"not at all":0, "rarely":1, "socially":2, "often":3, "very often":4, "desperately":5})
df["drugs"] = df["drugs"].map({"never":0, "sometimes":1, "often":2})
df["smokes"] = df["smokes"].map({"no":0, "trying to quit":1, "sometimes":2, "when drinking":3, "yes":4})
```

After updating these features, if we take out the NaN entries, we'd have removed about ~17.5K entries. We'll have about 70% of the data remaining.

```
len(df) - len((df.drinks + df.drugs + df.smokes).dropna())
```

I made a choice of sticking with the 30% data loss than making a big assumption of converting these NaNs into, say, a "moderate" answer of *2* or *3* (or any other value for that matter).

FEATURE – TOTAL TEXT LENGTH

If pop psychology articles are to be trusted, then expressiveness is directly proportional with age given what is called the *reading gap* in the three generations we'll explore. I interpreted "length of composition" as expressiveness in the dataset.

```
import math
df["essay0_length"] = df["essay0"].apply(lambda row: len(row) if type(row) == str else 0)
df["essay1_length"] = df["essay1"].apply(lambda row: len(row) if type(row) == str else 0)
df["essay2_length"] = df["essay2"].apply(lambda row: len(row) if type(row) == str else 0)
df["essay3_length"] = df["essay3"].apply(lambda row: len(row) if type(row) == str else 0)
df["essay4_length"] = df["essay4"].apply(lambda row: len(row) if type(row) == str else 0)
df["essay5_length"] = df["essay5"].apply(lambda row: len(row) if type(row) == str else 0)
df["essay6_length"] = df["essay6"].apply(lambda row: len(row) if type(row) == str else 0)
df["essay7_length"] = df["essay7"].apply(lambda row: len(row) if type(row) == str else 0)
df["essay8_length"] = df["essay8"].apply(lambda row: len(row) if type(row) == str else 0)
df["essay9_length"] = df["essay9"].apply(lambda row: len(row) if type(row) == str else 0)
df["total_text_length"] = df.apply(lambda row: row.essay0_length + row.essay1_length +
    row.essay2_length + row.essay3_length + row.essay4_length + row.essay5_length +
    row.essay6_length + row.essay7_length + row.essay8_length + row.essay9_length, axis=1)
```

I created a "_length" series for each essay portion using the *df.apply* function. Then for each profile, the total "_length" is summed to get our "total_text_length". If an essay is made up of NaN, then its length is 0. We therefore do not have any NaNs in our "total_text_length" feature, only integers.

FEATURE – RELIGIOUS SERIOUSNESS

The “religion” column had entries that ran the gamut. What I’ve noticed though is that users would express how seriously they take their religion for – from “laughing about it” to “very serious”. So this looked to me like a scale.

```
religious_seriousness = []
for i in range(len(df)):
    if type(df["religion"][i]) == str:
        if "laughing" in df["religion"][i]:
            religious_seriousness.append(0)
        elif "not too serious" in df["religion"][i]:
            religious_seriousness.append(1)
        elif "somewhat serious" in df["religion"][i]:
            religious_seriousness.append(2)
        elif "very serious" in df["religion"][i]:
            religious_seriousness.append(4)
        else:
            religious_seriousness.append(3) #"serious"
    else:
        religious_seriousness.append(1)
df["religious_seriousness"] = religious_seriousness
```

As with before, we have 5 discrete values to this scale and therefore I opted to create a new series called “religious_seriousness” instead of incorporating *lambda* with the *df.apply* function.

FEATURE – RELIGIOUS SERIOUSNESS

In the scale, *0* represents not serious at all (entries that include the word "laughing") and *4* represents most serious (entries that include the phrase "very serious"). The value *3* would mean "serious" and this is the classification for entries that do not entail any modifying word, i.e. just "Judaism" or just "Islam".

```
religious_seriousness = []
for i in range(len(df)):
    if type(df["religion"][i]) == str:
        if "laughing" in df["religion"][i]:
            religious_seriousness.append(0)
        elif "not too serious" in df["religion"][i]:
            religious_seriousness.append(1)
        elif "somewhat serious" in df["religion"][i]:
            religious_seriousness.append(2)
        elif "very serious" in df["religion"][i]:
            religious_seriousness.append(4)
        else:
            religious_seriousness.append(3) #"serious"
    else:
        religious_seriousness.append(1)
df["religious_seriousness"] = religious_seriousness
```

This time, as opposed to the earlier case, NaNs are classified as value *1* ("not too serious") – an assumption made here is that a user who did not answer this entry can be said to be *not too serious* about his or her religion.

FEATURE – LANGUAGES

The “speaks” column lists all the languages each user knows. A user may even opt to share the proficiency of each reported language.

```
df["languages"] = df["speaks"].apply(lambda row: 0 if pd.isnull(row) else len(row.split(", ")))
```

Since each language is separated by a comma, the string is first split by a comma and a space then, once transformed into a list, the length of the list is recorded under the new feature, “languages”.

Since there are only 50 NaN values (or 0.08% of the total number of entries) under “speaks”, we consider these as outliers and have given them the value “0”.

FEATURE – LOVE

Lastly, it was interesting to find out how seriously these users make use of the dating site in their quest for romance. After all, this dataset just happens to coincide with the dawn of dating platforms more inclined for finding hookups than finding love: swipe-right online dating apps (Tinder was founded circa 2012).

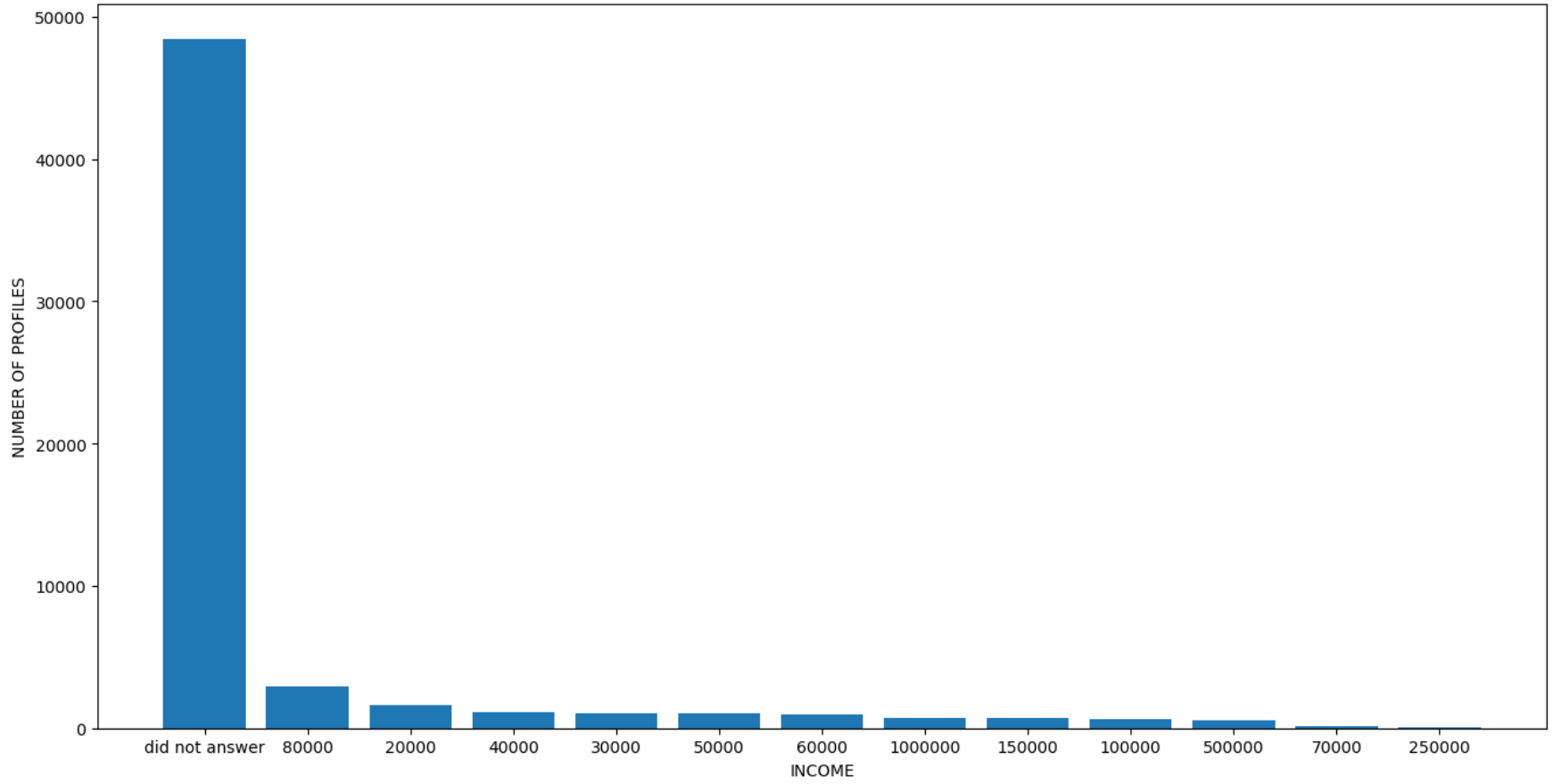
```
df["love"] = df["essay9"].apply(lambda row: 0 if pd.isnull(row) else row.casefold().count("love"))
```

This custom feature isn't complicated. It merely totals the instances of the word "love" in the user's *essay9*, which asks, "You should message me if…" – the assumption here is that a user who is straightforward with the use of the term, the more likely that person is serious in getting something more long-term from the use of the site.

It's quite an assumption, but it's worth playing with the idea.

UNUSED FEATURE: INCOME

Those who chose not to report income – as denoted in the .csv as *-1* – greatly outnumbered those who did. This would've been interesting as a feature…



CODE: `plt.bar(df.income_.unique(), df.income_.value_counts())`

UNUSED FEATURE: LOCATION

From the “location” feature, it was determined that there were 199 total unique entries, all of which had a format of “<specific location>, <general area>”.

I decided to take just the general area and called this new series as “state”.

I found out that the dating site had disproportionately more Californian users – a good ~99%. Classifying people as “from CA” vs. “not from CA” would be too uneven.

```
>>> df["state"] = df.apply(lambda row:
                           row["location"].split(", ")[-1], axis=1)
>>> df.state.value_counts()
california                59855
new york                     17
illinois                      8
massachusetts                 5
texas                         4
michigan                      4
oregon                        4
florida                       3
arizona                       3
ohio                          2
virginia                      2
spain                         2
georgia                       2
minnesota                     2
washington                    2
district of columbia          2
utah                          2
hawaii                        2
united kingdom                2
colorado                      2
west virginia                 1
germany                       1
switzerland                   1
netherlands                   1
tennessee                     1
canada                        1
vietnam                       1
idaho                         1
rhode island                  1
wisconsin                     1
connecticut                   1
```

UNUSED FEATURE: LOCATION

Out of curiosity I wanted to find out where exactly in California most profiles come from. As it turns out ~52% of the total population are from San Francisco. The rest that had reported as being from California are from the Northern region of CA (as determined by inspecting the cities by name).

It seems clearer that perhaps the dating site is primarily catered to Bay Areans.

```
>>> df[df["state"] == "california"]["location"].value_counts()
san francisco, california          31064
oakland, california                 7214
berkeley, california                4212
san mateo, california               1331
palo alto, california               1064
alameda, california                  910
san rafael, california               755
hayward, california                  747
emeryville, california               738
redwood city, california             693
daly city, california                681
san leandro, california              651
walnut creek, california             644
vallejo, california                  558
menlo park, california               479
richmond, california                 424
south san francisco, california      416
mountain view, california            384
novato, california                   369
burlingame, california               361
pleasant hill, california            347
castro valley, california            345
stanford, california                 341
el cerrito, california               325
pacifica, california                 323
martinez, california                 316
mill valley, california              315
san bruno, california                290
san pablo, california                245
belmont, california                  243
                                    ...
```

NORMALIZATION

WHY MIN-MAX SCALING WAS CHOSEN TO NORMALIZE THE FEATURES OF OUR MODEL

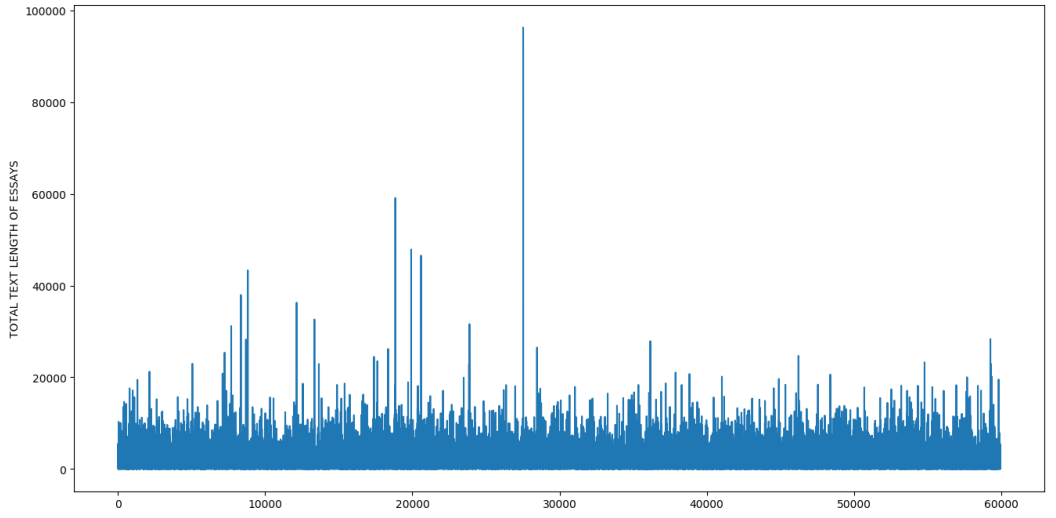
NORMALIZATION

Most of the features are made up of discrete values that represent a scale – a “0” means “*least*” and the max value (say, “5”) corresponds to “*most*”. Even the “languages” column has, coincidentally, a narrow integer range from 0 to just 5.

Given these, it would appear that using either Min-Max or Z-score normalization should work. However, one particular feature has an outlier problem. And its only outlier is massive enough to shrink the rest of the entries due to its magnitude.

The “total_text_length” column features integers that represent the character count of all the essays written by each user. So it’s no surprise that a typical entry is in the range of thousands.

Its outlier, however, stands at almost one hundred thousand characters!



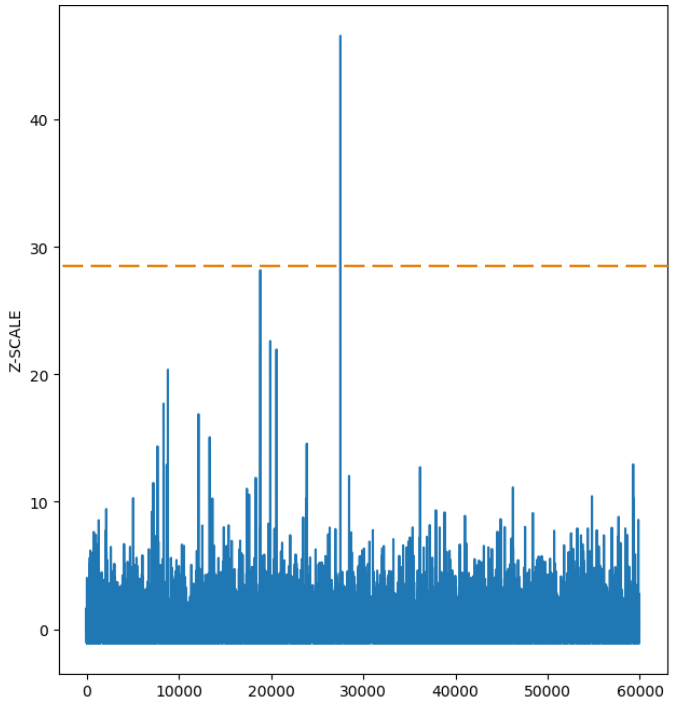
NORMALIZATION

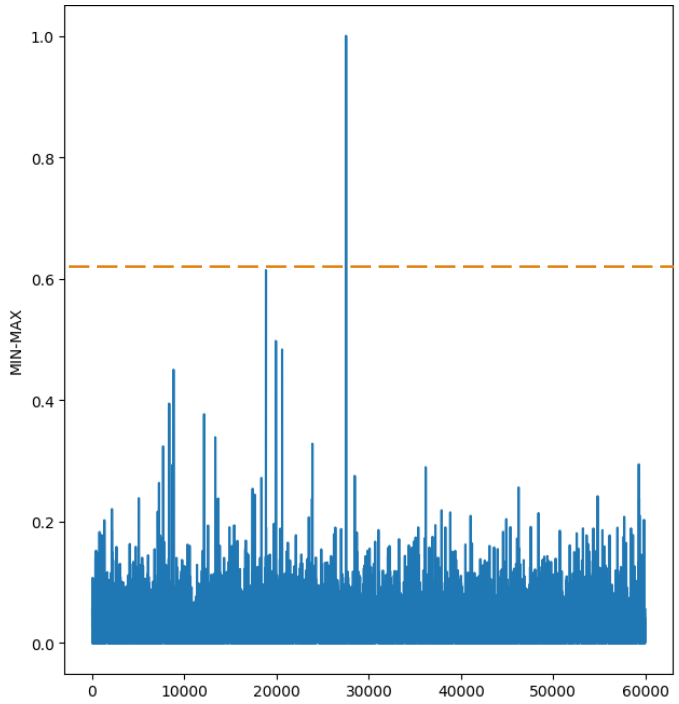
Exploring the outlier, one could argue that this user is indeed *truly* expressive.

```
>>> df.total_text_length.max()
96268
>>> df[df["total_text_length"] == 96268].index
Int64Index([27528], dtype='int64')
>>> df.essay0.iloc[27528]
'before i say anything, can i offer, please check out<br />\n<br />\n<a class="ilink" href=\n"/interests?i=the+book%2c+anastasia%2c+by+megre%2c+the+ringing+cedar+series%2c%3cbr+%2f%3e%0aand+t
he+films%2c+%22zeitgeist%22%2c+%22kymatica%22%2c+the+esoteric+agenda+and%0a%22thrive%22+able+to+be+seen+on+youtube">\nthe book, anastasia, by megre, the ringing cedar series,<br />\nand the f
ilms, "zeitgeist", "kymatica", the esoteric agenda and\n"thrive" able to be seen on youtube</a><br />\n<br />\nam reikiteacher redwood on facebook (where you may find more\ninformation on my
wall).<br />\n<br />\nringing a few bells<br />\n<a class="ilink" href=\n"/interests?i=awakening...krysthl-a%3cbr+%2f%3e%0a...melchizedek...heinlein...nacaals...%22serpent+of+life%22...+egypt
...%0athe+power+of+love+is+stronger+than+you+know+%28%22not+romantic+love%22%0abut+real+love%29">\nawakening...krysthl-a<br />\n...melchizedek...heinlein...nacaals..."serpent of life"... egyp
t...\nthe power of love is stronger than you know ("not romantic love"\nbut real love)</a><br />\n<br />\nunlike many here who select, "seeking friends", as one of their\noptions, and who spu
rn those that don\'t meet their fantasy ideal\nromantic partner, i am open to any and all manner of associations\nthat are meant to be (that feel "right" or "true").<br />\n<br />\nheya,<br /
>\nlike to read? here\'s a little something. use in moderation\nnow.<br />\nthis profile is like a refrigerator. there\'s something in it you\nmight consume at different times, but consuming
the whole thing at\none sitting would be a lot to digest.<br />\nhere goes...<br />\n<br />\na bit of a thumbnail summary: adventurous, kind, intelligent,\nconscious, super creative and playf
ul, expressive, communicator,\nmeditator, fitness freak, kinesthetic, tactile, perceptive, nature\nloving, high sensitivity-awareness, comfortable and adept with\nverbal/emotional and physica
l intimacy, healer/culture healer,\nmystic with very complex, dynamic, and also very simple, nature.\nwilling to be different, individual or unique, without pretension.\nvisionary.. committed
. community oriented. <a class="ilink" href=\n"/interests?i=love+children+and%0aanimals+and+adults+who+are+big+kids+with+open+hearts.">\nlove children and animals and adults who are big kids
with open\nhearts.</a> easy transmitter and receiver of energy. tantra,\ntaoism, yoga, qi gong, reiki, quantum mechanics, abraham hicks,\ndrunvalo melchizedek, bruce lipton, etc. a true heart
. said\nelsewhere, "you are strong minded, the rare mix of intense and\nmellow, silly and serious, actively compassionate and optimistic.\n"<br />\n<br />\na good bet if you are interested in
<a class="ilink" href=\n"/interests?i=friendship">friendship</a>, <a class="ilink" href=\n"/interests?i=growth">growth</a>, <a class="ilink" href=\n"/interests?i=creative+playfulness">creativ
e playfulness</a>,\n<a class="ilink" href="/interests?i=loyalty">loyalty</a>, <a class=\n"ilink" href=\n"/interests?i=being+seen%2c+heard+and%0aunderstood">being seen,\nheard and understood</
a> in ways you may not have been before,\n<a class="ilink" href="/interests?i=intimacy">intimacy</a>. if you\nare right to meet a healer or friend, or acquaintance, or ally, or\nlover, or par
tner who is a believer and practitioner in <a class=\n"ilink" href="/interests?i=unconditional+love">unconditional\nlove</a>. i am drawn to that which is, and to act in ways that are,\n<a cla
ss="ilink" href="/interests?i=life-giving">life-giving</a>.\nspecifically in regard to relationships, i am open to and capable\nof <a class="ilink" href=\n"/interests?i=alternative+types+of%0
arelationship">alternative\ntypes of relationship</a>. when addressing relationship, i prefer\nto let the substance (the truth revealing itself) determine the\nreality rather than imposing a
held fantasy onto a person or\ncultural "script". lest you think the above means the opposite, i\nam as solid (earthed/grounded) or committed a person that you may\nmeet.<br />\nthe one thing
that is different about me, particularly in u.s.\nculture is the ability to love. we grow up with love defined as\nneed, desire, something we do to avoid the fear of being alone or\n"dying al
one", liking, possession, or even attraction. none of\nthese are what i call love. love is much different and without\nconditions. i will leave that here for now, as it is a much deeper\nsubj
ect i will not delve into further at this moment.<br />\n<br />\nalso, not hip, and not unhip. i know what is culturally savvy, but\nmy values are generally geared more toward meaning than\ns
uperficiality. i\'ve got my own style, which gets expressed in\ndifferent ways all the time, but it doesn\'t get in the way of\nwhat\'s important to me, like people, animals, plants, life,\nc
onsciousness, expression from the heart and such. <a class="ilink"\nhref=\n"/interests?i=%28raspberry%0asilly+face+%3a%7ed%29+nyaaaaa-nyaaaa%29">\n(raspberry silly face :~d) nyaaaaa-nyaaaa)</
a><br />\n<br />\nwalking down a wooded path you come to a fork.<br />\ndown one direction is <a class="ilink" href=\n"/interests?i=fear">fear</a>.<br />\ndown the other is <a class="ilink" h
ref=\n"/interests?i=love">love</a>. which road will you take today?<br />\n<br />\nif you have read or are reading <a class="ilink" href=\n"/interests?i=%22anastasia%22+by+megre">"anastasia"
by megre</a>\nand you have instant understanding or some of what you know is\nre-affirmed, then you are in good company.<br />\n<br />\ngot a body needing some release, some transitioning or
changes to\nmake, some healing or growing to focus on, you can find\nprofessional work here at my web address <a class="ilink" href=\n"/interests?i=wholebeinghealingcom">wholebeinghealingcom<
/a><br />\n<a class="ilink" href=\n"/interests?i=+if+you+are+a+professional+bodyworker+or+energy+healer+too%2c+a%0atrade+may+be+available">\nif you are a professional bodyworker or energy hea
ler too, a trade\nmay be available</a>. community events at <a class="ilink" href=\n"/interests?i=reiki+meetup+com%0a484">reiki meetup com\n484</a><br />\n<br />\ni am a <a class="ilink" href
="/interests?i=reiki">reiki</a>\nteacher, <a class="ilink" href=\n"/interests?i=shamanic+energy+practitioner">shamanic energy\npractitioner</a>, life<br />\ntransformation <a class="ilink" hr
ef=\n"/interests?i=counselor">counselor</a> and coach, <a class="ilink"\nhref="/interests?i=teacher+of%0ahealers">teacher of\nhealers</a>,<br />\n<a class="ilink" href="/interests?i=intuitive
s">intuitives</a>,\n<a class="ilink" href="/interests?i=sensitives">sensitives</a> -\nexperienced or not,<br />\n<a class="ilink" href="/interests?i=workshop+leader">workshop\nleader</a>, cul
ture healer, <a class="ilink" href=\n"/interests?i=massage+therapist+and%0asomatic+bodyworker">massage\ntherapist and somatic bodyworker</a><br />\nat <a class="ilink" href=\n"/interests?i=wh
olebeinghealingdotcom">wholebeinghealingdotcom</a>\nseeing clients from a distance via skype and locally.<br />\n<br />\n----------------------------------------------------------------------------
------<br />\n\n<br />\n<a class="ilink" href=\n"/interests?i=have+you+been+searchin+for+a+heart+of+gold%3f">have\nyou been searchin for a heart of gold?</a><br />\n<br />\n<a class="ilink" h
ref="/interests?i=me+too.">me too.</a><br />\n<br />\n<a class="ilink" href="/interests?i=i+got+a+hint.">i got a\nhint.</a><br />\n<br />\n<a class="ilink" href="/interests?i=it%27s+your+own.
">it\'s your\nown.</a><br />\n<br />\n<a class="ilink" href=\n"/interests?i=before+you+doubt%2c+consider+it...+">before you\ndoubt, consider it...</a><br />\n<br />\nhttp://www.youtube.com/wa
tch?v=eh44qpt1mpe&amp;feature=related<br />\n\n----------------------------------------------------------------------------<br />\n\n<br />\nfor your consideration: how much does appearance come i
```

NORMALIZATION

Instead of deleting this entry, I decided to trim the outlier to the second maximum value. We'll see that the scale by Z-scoring will then drop to an upper limit of ~29. Still, I opted to make use of the Min-Max scaler as I prefer to have all features on an aligned scale from 0 to 1. This avoids any feature overriding the importance of another just because one has a larger scale.





SPLITTING THE DATA

FOR TRAINING AND FOR VALIDATION

SPLITTING THE DATA

A simple 80-20 split was performed. Eighty percent goes to training the model and the rest were used for validation (i.e. calculating the accuracy of the supervised model).

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,train_size=0.8,test_size=0.2,random_state = 42)
```

Random state of 42 appears to be most commonly used. It is, after all, the answer to the universe.

REGRESSION MODEL

USING MULTIPLE LINEAR REGRESSION AND K-NEAREST NEIGHBORS REGRESSION TO DETERMINE USER'S AGE

MULTIPLE LINEAR REGRESSION MODEL

Again, the goal is to predict the age of a user based on his/her online dating profile using machine learning. We'll use the features we've explored as our independent variables to feed into our model.

```
from sklearn.linear_model import LinearRegression

def linear_regression(x, y):
    x_train, x_test, y_train, y_test = train_test_split(
        x,y,train_size=0.8,test_size=0.2,random_state = 42)

    line_fitter = LinearRegression()
    line_fitter.fit(x_train, y_train)

    predicted_y = line_fitter.predict(x_train)

    score = line_fitter.score(x_test, y_test)

    return score
```

Then we'll make use of accuracy as our score to determine how well our model will do. This score is the calculated R^2 coefficient, or the percentage variation in our "y" as explained by all our "x" variables together.

MULTIPLE LINEAR REGRESSION

The column names we've elected to feed into our model are inserted in a list called "features". Any one of these features and any combination of each – from pairs to the combination of all eight – will work and each feature or combination will have a corresponding "score". Ultimately, what we want is the optimal combination that will give us the best accuracy possible in our model.

```
from itertools import combinations
features = ["generation", "total_text_length", "languages", "drinks", "drugs", "smokes", "religious_seriousness", "love"]
combo = []
for i in range(1,len(features)):
    combo.append(list(combinations(features, i)))
```

To ensure we get the optimum, I utilized the *combinations* function to gather all the possible combinations of our features. Each group is then appended to the "combo" list shown above.

MULTIPLE LINEAR REGRESSION

The “combo” list has a total of seven elements, with each element representing all the unique *n*-group of the features.

So the 1st element (*n*=1) is each single feature:

```
[('generation',), ('total_text_length',), ('languages',), ('drinks',), ('drugs',), ('smokes',), ('religious_seriousness',), ('love',)]
```

And the 2nd element (*n*=2) is all the pair combinations of the features:

```
[('generation', 'total_text_length'), ('generation', 'languages'), ('generation', 'drinks'), ('generation', 'drugs'), ('generation', '
smokes'), ('generation', 'religious_seriousness'), ('generation', 'love'), ('total_text_length', 'languages'), ('total_text_length', '
drinks'), ('total_text_length', 'drugs'), ('total_text_length', 'smokes'), ('total_text_length', 'religious_seriousness'), ('total_tex
t_length', 'love'), ('languages', 'drinks'), ('languages', 'drugs'), ('languages', 'smokes'), ('languages', 'religious_seriousness'),
('languages', 'love'), ('drinks', 'drugs'), ('drinks', 'smokes'), ('drinks', 'religious_seriousness'), ('drinks', 'love'), ('drugs', '
smokes'), ('drugs', 'religious_seriousness'), ('drugs', 'love'), ('smokes', 'religious_seriousness'), ('smokes', 'love'), ('religious_
seriousness', 'love')]
```

And so on and so forth…

Note that though there are eight features, there is no such thing as a *zero*-group (or *n*=0) combination hence only seven elements.

MULTIPLE LINEAR REGRESSION

The idea is to input each combination – essentially, each member of each element of our “combo” list – into our *linear_regression* function and to append all the accuracy scores of each into the “scores” list. From there, we’ll determine what optimal combination of features gives us the best score for our model.

```
scores = []
for i in range(len(combo)):
    for j in range(len(combo[i])):
        x = df[[*combo[i][j]]]
        y = df[["age"]]
        scores.append([linear_regression(x, y), combo[i][j]])
```

The dataframe “x” is based on the normalized dataframe “df” and only contains the independent variables (our combination of features). The “dataframe” y is the dataframe that holds our dependent variable, “age”.

MULTIPLE LINEAR REGRESSION

Our “scores” is a list of lists where every element of the root list contains a tuple enumerating the combination of features, and its corresponding score.

```
>>> max(scores, key=lambda x: x[0])

[0.8361902472127223, ('generation', 'total_text_length', 'languages', 'drugs', 'smokes', 'religious_seriousness', 'love')]
```

To find our optimal score, we print the element of “scores” that has the highest accuracy using the *max* function. And as it turns out – with an accuracy of 83.62% when tested with our validation set – our model works best when we utilize seven of the eight features we have explored.

It can then be said that for our dataset, the “drinks” column as a feature does not really establish a linear relationship with age.

Perhaps the younger folks do not necessarily drink any more than the older folks do (or vise versa), or at least not enough to establish a distinction.

K-NEAREST NEIGHBORS REGRESSION

Now let's see if we can arrive to the same conclusion when we use a different approach. Employing the same set of features we've used, this time we'll make our prediction using K-Nearest Neighbors Weighted Regression to guess the age of a user based on his/her online dating profile.

```
from sklearn.neighbors import KNeighborsRegressor

def KNN_regression(x, y, neighbors):
    x_train, x_test, y_train, y_test = train_test_split(
        x,y,train_size=0.8,test_size=0.2,random_state = 42)

    regressor = KNeighborsRegressor(n_neighbors = neighbors, weights = "distance")
    regressor.fit(x_train, y_train)

    score = regressor.score(x_train, y_train)

    return score
```

We indicate *distance* in our *weights* in order for the "regressor" to take account the distances of the neighbors. This way, we don't treat each neighbor as though it has an equal distance to our unknown.

K-NEAREST NEIGHBORS REGRESSION

We still make use of the same “combo” list so we can determine what the ideal combination of features is to get the best results for our model.

```
combo = []
for i in range(1,len(features)):
    combo.append(list(combinations(features, i)))
```

Again, we store our accuracy values per combination in our “scores” list.

```
scores = []
for i in range(len(combo)):
    for j in range(len(combo[i])):
        x = df[[*combo[i][j]]]
        y = df[["age"]]
        scores.append([KNN_regression(x, y, 5), combo[i][j]])
```

For now we’ll set a constant of five neighbors on our regressor for all the different pairs of “x” and “y” dataframes.

K-NEAREST NEIGHBORS REGRESSION

Again, we get the best results for our model if we apply only seven of the eight features we came up with. However, upon closer inspection, this isn't exactly the same combination of features we determined was best for our Multiple Linear Regression approach. This time, the feature "drinks" is considered helpful – it's the "love" column that's best not to include in our model.

```
>>> max(scores, key=lambda x: x[0])
[0.9888113141570314, ('generation', 'total_text_length', 'languages', 'drinks', 'drugs', 'smokes', 'religious_seriousness')]
```

Another surprising aspect is the level of accuracy these features achieved in this technique – an astounding 98.88%!

To be fair, we have established the "generation" column directly from "age", which is what we intend to predict. So in a way, there is *circular reference* in our method. Still, as we've seen, "generation" alone does not fare better than when we employ the seven custom features altogether.

K-NEAREST NEIGHBORS REGRESSION

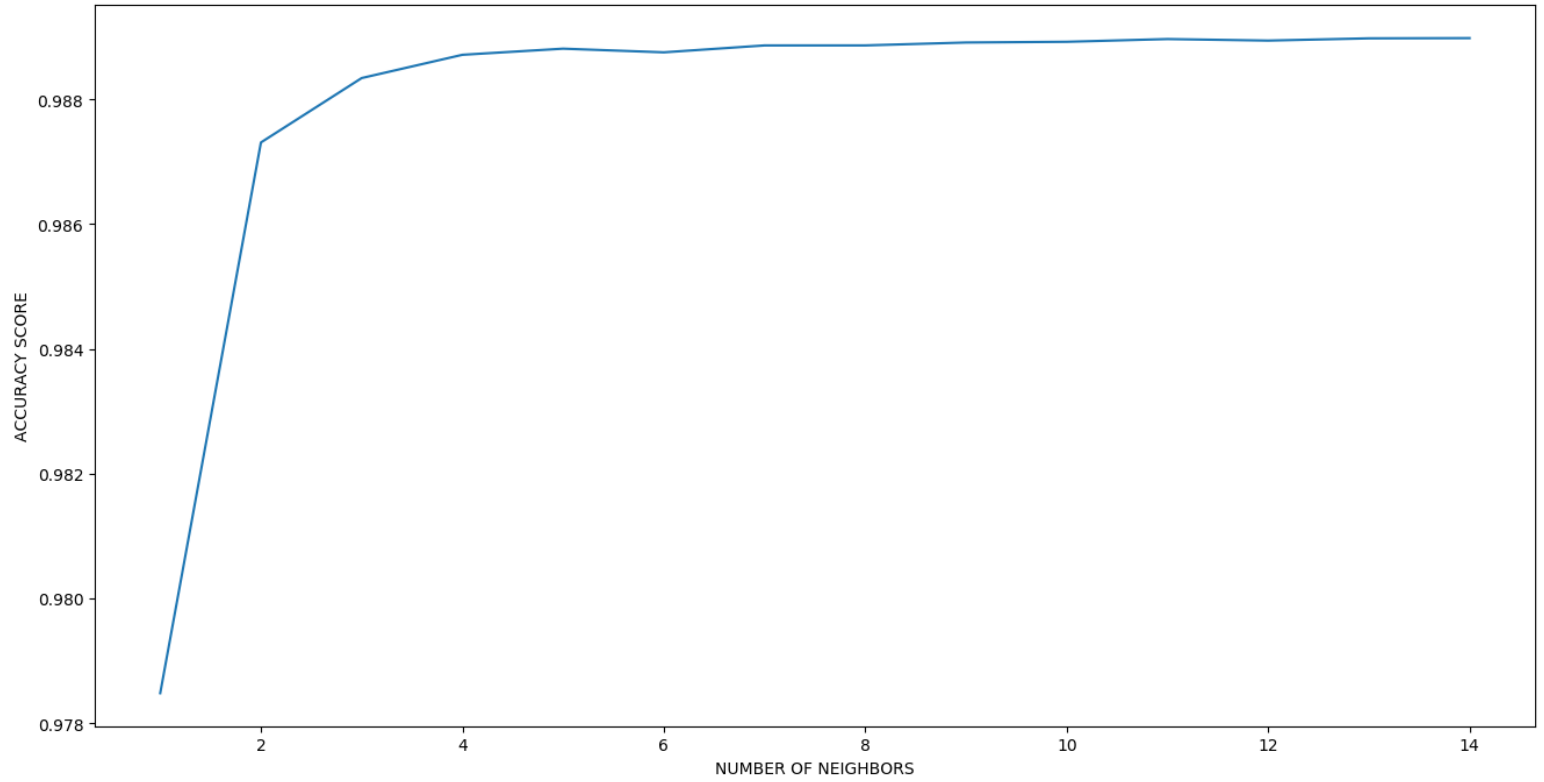
Now that we know the best combination of features is for this model, let's dig a bit deeper and find out if we get better or worse results if we make use of a different number of neighbors.

```
N_score = []
for neighbors in range(1,15):
        x = df[["generation", "total_text_length", "languages", "drinks", "drugs", "smokes", "religious_seriousness"]]
        y = df[["age"]]
        N_score.append([KNN_regression(x, y, neighbors), neighbors])
```

The "N_score" will store the values of the accuracy scores for each of the K-Nearest Neighbors regression performed. Since the K-Nearest Neighbors is substantially slower to run, we'll only use up to 14 neighbors.

K-NEAREST NEIGHBORS REGRESSION

As it turns out, using fourteen neighbors does better with an accuracy of 98.898% as compared with our original five neighbors with 98.881%. Still, this is not much of a difference, and based on the chart, the plateau forms at around the *K* of 5.



CODE: `plt.plot([i for i in range(1,15)], [i[0] for i in N_score])`

CLASSIFIER MODEL

USING K-NEAREST NEIGHBORS CLASSIFIER AND SUPPORT VECTORS MACHINE TO DETERMINE THE USER'S GENERATION

K-NEAREST NEIGHBORS CLASSIFIER

Instead of age, this time we'll try to find out what generation the user belongs to based on his/her dating profile. With classifier models, we'll be dealing with discrete values instead of continuous ones. Our OKCupid dataset encompasses three generations of users: Millennials, Gen X-ers, and Boomers. The output of our model will then be just either of the three.

```
from sklearn.neighbors import KNeighborsClassifier

def KNNClassifier(x, y, neighbors):
    x_train, x_test, y_train, y_test = train_test_split(
        x,y,train_size=0.8,test_size=0.2,random_state = 42)

    classifier = KNeighborsClassifier(n_neighbors = neighbors)
    classifier.fit(x_train, y_train)

    score = classifier.score(x_test, y_test)

    return score
```

Just like our regression functions, we'll use accuracy as our basis for scoring. We will not encounter situations where we'll have "false positives" or "false negatives" since we have three distinct classes. So if the model mislabels an input, it simply "misclassifies" it. Therefore, accuracy will be sufficient (i.e., no need for F1 scores).

K-NEAREST NEIGHBORS CLASSIFIER

We'll use the same technique we previously employed in order to come up with all combinations of our "features" list. One difference we have now is that we'll have to drop the "generation" column from our independent variable since we'll be making use of this feature for our "y" value.

```
from itertools import combinations
features = ["total_text_length", "languages", "drinks", "drugs", "smokes", "religious_seriousness", "love"]
combo = []
for i in range(1,len(features)):
    combo.append(list(combinations(features, i)))
```

We'll feed the "combo" list to our *KNNClassifier* function. When we performed our regression study, we scaled our "generation" for normalization. We undo this using the *inverse_transform* function for it to return to its original, discrete form. The rest of the features will remain scaled.

```
scores = []
y = scaler.inverse_transform(np.reshape(df[["generation"]], (-1,1)))
y = np.ravel(y)
for i in range(len(combo)):
    for j in range(len(combo[i])):
        x = df[[*combo[i][j]]]
        scores.append([KNNClassifier(x, y, neighbors=5), combo[i][j]])
```

K-NEAREST NEIGHBORS CLASSIFIER

Now that we have our scores, we use the *max* function to determine which group of features gives our model the optimal accuracy.

```
>>> max(scores, key=lambda x: x[0])

[0.6103070949523474, ('religious_seriousness', 'love')]
```

Unfortunately the best we could do isn't good enough, with an accuracy of 61.03%. This means that our model would do only slightly better as compared to randomly guessing the outcome of a coin flip.

Still, we can infer from our model's results that we can *attempt* to determine what generation a person belongs to if we know how seriously they take religion and how seriously they take love. Indeed, recent polls have shown that younger generations' affiliation with any religion have waned as compared to their precedents. And it almost seems intuitive that an older, single person with an online dating profile is more likely to look for love than to look for casual encounters. So there is *some* relationship there, though not as clear cut as we had hoped.

K-NEAREST NEIGHBORS CLASSIFIER

We've arrived to our accuracy score by using five neighbors. Now let's try different values for our neighbors and see if we can get a better score than ~61%.

We'll plug in each of the different *K* values and run the *KNNClassifier* function. The accuracy score for each neighbor (from 1 to 14) will be stored in our "N_score" list.

```
N_score = []
x = df[["religious_seriousness", "love"]]
for neighbors in range(1,15):
        N_score.append([KNNClassifier(x, y, neighbors), neighbors])
```

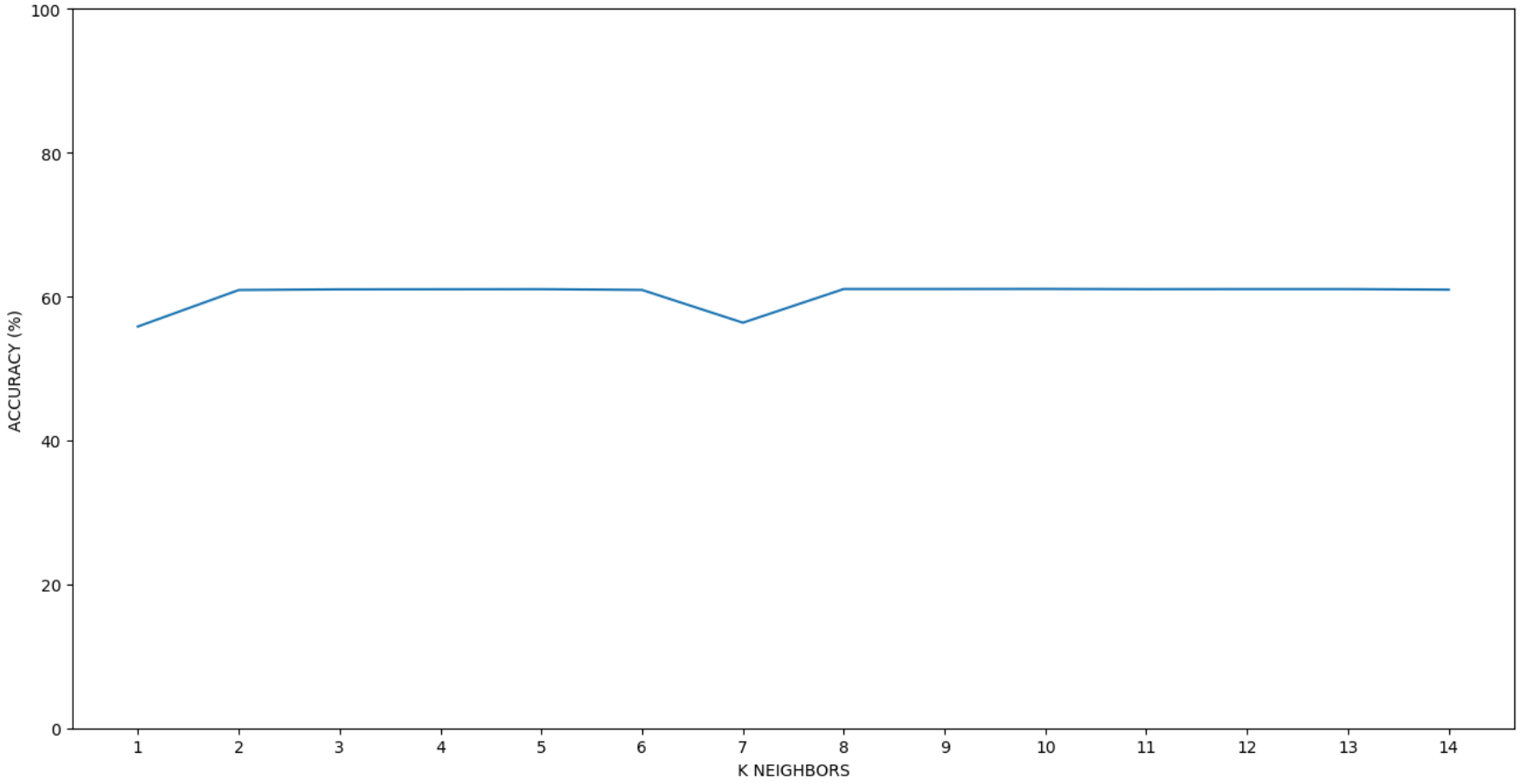
Using the *max* function, we can determine what number of neighbors would give us the best accuracy.

```
>>> max(N_score, key=lambda x: x[0])

[0.6106600776561949, 10]
```

K-NEAREST NEIGHBORS CLASSIFIER

We have nudged our accuracy just a bit higher when we used 10 neighbors instead of 5 – but with a 0.04 percentage points increase, it isn't much of a difference.



CODE: `plt.plot([i[1] for i in N_score], [i[0]*100 for i in N_score])`

SUPPORT VECTORS MACHINE MODEL

Now let's turn to using a Support Vectors Machine classifier and see if we'll come close to reproducing our KNN's accuracy or if we'll get something different.

```
from sklearn.svm import SVC

def SVCClassifier(x, y, C, gamma):
    x_train, x_test, y_train, y_test = train_test_split(
        x,y,train_size=0.8,test_size=0.2,random_state = 42)

    classifier = SVC(kernel="rbf", C=C, gamma=gamma)
    classifier.fit(x_train, y_train)

    score = classifier.score(x_test, y_test)

    return score
```

Our SVM (or SVC) classifier uses RBF or Radial Bias Function as its kernel. Since we may have multi-dimensional features as our independent variable, we will likely encounter data that is not linearly separable. Using RBF works best in these cases.

SUPPORT VECTORS MACHINE MODEL

Since Support Vectors Machine models take much longer to run, we'll opt not to use the "combo" list that had all the combinations of the features – instead, we'll input in our SVM what was already determined by our K-Nearest Neighbors classifier as the best combination of features. Again it's the "religious_seriousness" and "love" pair.

```
x = df[["religious_seriousness", "love"]]
y = scaler.inverse_transform(np.reshape(df[["generation"]], (-1,1)))
y = np.ravel(y)
```

What we'll do, however, is to make use of *for* loops to iterate through a range of different *C* and *gamma values* and see which pair will give us the highest accuracy score for our model.

```
c = [0.1, 1.0, 10, 100, 1000]
gamma = [1, 10, 100]
scores = []
for C_ in c:
    for gamma_ in gamma:
        scores.append([SVCClassifier(x, y, C_, gamma_), C_, gamma_])
```

SUPPORT VECTORS MACHINE MODEL

Now that we have our scores, we use the *max* function to determine which *C* and *gamma* pair gives our model the optimal accuracy. With 60.96%, we have obtained a slightly lower score than what our KNN classifier yielded.

```
>>> max(scores, key=lambda x: x[0])
[0.6096011295446523, 1.0, 10]
```

We'll use a heat map to show us that different values for *C* and *gamma* produce a variety of results, though the delta between the worst and best in our case is quite minute.

```
import seaborn as sns

scores_accuracy = [i[0] for i in scores]

scores_ = []
for i in range(int(len(scores_accuracy)/3)):
    scores_.append(scores_accuracy[i*3:(i*3)+3])

heat_df = pd.DataFrame(scores_, index=c, columns=gamma)
sns.heatmap(heat_df, annot=True, fmt="f", linewidths=0.5, cbar_kws={"label": "ACCURACY"})

plt.xlabel("GAMMA")
plt.ylabel("C VALUE")
plt.show()
```

SUPPORT VECTORS MACHINE MODEL

We can see that a *C* of 1.0 and a *gamma* of 10 worked best in our SVC classifier.

CONCLUSION

FINAL THOUGHTS

CONCLUSION

In order to come up with a good supervised machine learning model, we'll need to have solid and reliable data. This *OKCupid* dataset had a fair amount of invalid entries (NaNs) especially in columns we would have found useful to use, like the *Income* feature.

Feature Engineering is also a significant aspect of modeling that a data scientist should meticulously carry out. Datasets can be massive and all that resource would mean nothing if they weren't put into the right context. It also takes a lot of probing and exploration in order to get on the right track to suitably tackle the problem statements.

When we performed Regression, we had determined that the features that worked best with the Linear Regression weren't necessarily the same exact ones that got the best accuracy for the KNN Regression model. A good follow-up activity would be to apply a different train-test split technique – perhaps *N*-Fold Cross Validation – to see which features provide us the highest overall accuracy in different facets of the same data. And moreover, if a certain combination of features performs optimally for *both* regression approaches.

CONCLUSION

For Classification, the accuracies of the same set of independent variables aligned well when applied to our KNN and SVM classifiers. But then again the scores we obtained were only around ~60%. We've even tried playing around with classifiers' inherent parameters to see if we can bump up our accuracy scores, but there's only so much this can do. Ultimately, at this point, the best way forward is to reconsider the features or even swap or add new features entirely.

Here are some ideas that may be worth looking into:

1. Applying sentiment analysis on *essay0*, or "My Self Summary". I would think edgier Millennials would have higher negative sentiment scores across the board.
2. On the same token, if it is permitted on the dating site, an expletive counter would easily reveal if a user is likely a Gen X-er.
3. On the "*favorite*" essay, perhaps we can correlate the movies listed by users against an IMDB database and make use of its "year" feature. The assumption is that on average, users are more inclined to like movies that are released in their prime.