Q1.) Make up your own set of word features describing 6 different entities; with some obvious overlaps and differences.

The word features are taken from multiple sources that revolve around the definition of Brexit. Following is the snippet -

```
documentA = "Brexit is for British exit referring to the UK decision in a referendum to leave the European Union"
documentB = "Brexit is Britian's exit referring to the UK decision to leave the European Union"
documentC = "Brexit is combination of words Britain and exit which signifies split from the European Union"
documentD = "Brexit is abbreviation for withdrawal of United Kingdom from European Union"
documentE = "Brexit is for British exit referriing to United Kingdom decision to leave the European Union"
documentF = "Brexit is an abbreviation for British exit referring to the UK decision to leave the European Union"
```

a. Modify the Jaccard-Index python program to do Jaccard-Distance and then compute all pairwise distances between the entities. Based on results, show empirically, that the property of triangle inequality holds for measure.

Jaccard Similarity or intersection over union is defined as size of the intersection divided by the size of union of two sets. Formula →

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Jaccard Distance → is 1 - J(A, B)

Following is the process undertaken to prove property of triangle inequality:-

1. 6 documents are preprocessed(tokenization, stop-words removal, lemmatization)
2. Each document is named as A, B, C, D, E, F respectively.
3. Then a pairwise comparison between each document is made to calculate the jaccard distance between them which yields the following result -

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	AB	AC	AD	AE	AF	BC	BD	BE	BF	CD	CE	CF	DE	DF	EF
1	0.16	0.58	0.65	0.3	0.1	0.56	0.62	0.37	0.16	0.62	0.58	0.58	0.41	0.53	0.3

```
### Proving Jaccard Similarity ###
z(AC):0.58
x(BC):0.56
y(AB):0.16
z < x + y: True
```

Here, the dataframe contains pairwise document names and corresponding similarity scores. Then we prove the triangle inequality by comparing documents scores which is represented with the variables x, y and z in the aforementioned snippet.

b. Now implement the difference function for the Dice Coefficient and show that the property of triangle inequality may not hold for this measure

Dice Coefficient is given by the formula -

$$\frac{2 * |X \cap Y|}{|X| + |Y|}$$

Difference function for Dice Coefficient → 1 - dice_coefficient

The same process is continued and each document is given a name from A to F. Here dice coefficient is taken and following is the result that proves triangle inequality doesn't hold true for all the instances.

Case when triangle inequality fails -

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	AB	AC	AD	AE	AF	BC	BD	BE	BF	CD	CE	CF	DE	DF	EF
1	0.16	0.58	0.65	0.3	0.1	0.56	0.62	0.37	0.16	0.62	0.58	0.58	0.41	0.53	0.3

```
### Proving Dice Coefficient ###
z(AD) : 0.65
x(DF) : 0.53
y(AF) : 0.1
z > x + y : True
```

Case when triangle inequality stays true -

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	AB	AC	AD	AE	AF	BC	BD	BE	BF	CD	CE	CF	DE	DF	EF
1	0.16	0.58	0.65	0.3	0.1	0.56	0.62	0.37	0.16	0.62	0.58	0.58	0.41	0.53	0.3

```
### Proving Dice Coefficient Distance ###
z(AC) : 0.58
x(BC) : 0.56
y(AB) : 0.16
z > x + y : False
```

2) Have a look at the Cosine.py program; nb you may need to install the packages its imports.

Cosine similarity → similarity measure that measures cosine of angle between two vectors /documents.

Formula →

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

a. Find 3 short documents about which you might want to know their similarity. Produce 5 variants on one of the documents and see how the cosine similarity changes.

Following is the data taken. (Original documents d1 → d3 and variants v1 → v5)

```
def load_docs():
    print("Loading docs...")
    doc1=('d1', 'Tom and Jerry Potato appears as cartoon as well as there are commercial products like mashed potato c
    doc2=('d2', 'Tommy Jerry Faces appear crispy, mashed, rounded chips and they are tasty')
    doc3=('d3', 'Tom Jerry and their chips are very famously available to the public')
    variant1 = ('v1', 'Tommy Jerry Faces appear crisp, mash, round chips and they are very tasty')
    variant2 = ('v2', 'Tommy Jerry Faces appearing crispy, mashed, rounded chips and they very very tasty')
    variant3 = ('v3', 'Tom and Jerry Faces appear crisp, mashed, rounded chips and they are very tasty')
    variant4 = ('v4', 'Tommy Jerry Faces appear, mashed, rounded chips and they are tasty')
    variant5 = ('v5' ,'Tommy Jerry Faces appear crispy, mashed, round chips tasty')
    return [doc1,doc2,doc3,variant1, variant2, variant3, variant4, variant5]
```

Cosine similarity for all the documents (3 short + 5 variants of 1 short) is calculated with the program and stored in a dict. Following is the result of cosine.py program that calculates cosine similarity between **3 original documents** -

```
Loading docs...
Distance of :d1=>d2
Cosine: 0.005
Distance of :d1=>d3
Cosine: 0.02
Distance of :d2=>d3
Cosine: 0.011
```

The similarity results for comparing these documents with the ‘q’ document is 0 because there are no similar words between document ‘q’ and any of the three documents d1, d2, and d3.

Following is the result of cosine.py program that calculates cosine similarity between all the **8 documents.**

	0	1
0	d1d2	0.005
1	d1d3	0.020
2	d1v1	0.003
3	d1v2	0.000
4	d1v3	0.047
5	d1v4	0.004
6	d1v5	0.000
7	d2d3	0.011
8	d2v1	0.224
9	d2v2	0.464
10	d2v3	0.440
11	d2v4	0.348
12	d2v5	0.613
13	d3v1	0.035
14	d3v2	0.064
15	d3v3	0.131
16	d3v4	0.008
17	d3v5	0.002
18	v1v2	0.173
19	v1v3	0.479
20	v1v4	0.094
21	v1v5	0.448
22	v2v3	0.315
23	v2v4	0.190
24	v2v5	0.272
25	v3v4	0.221
26	v3v5	0.197
27	v4v5	0.131

Now,
Document 'd2' → base document (This document is chosen as the base to compare with other documents)
Following is the result when we calculate distance of d2 with all the other documents.

{'d1': 0.005, 'd2': 1.0, 'd3': 0.011, 'v1': 0.224, 'v2': 0.464, 'v3': 0.44, 'v4': 0.348, 'v5': 0.613}

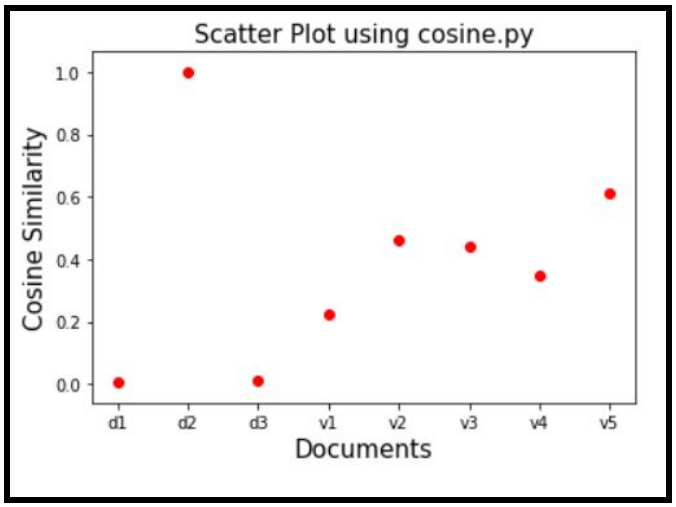
From the results, we can infer that the cosine similarity for the different original documents (d2 → d1, d2→ d3) is very less. Also, it is clearly observed that the cosine similarity for the same document is 1. i.e. (d2→ d2)
The cosine similarity is comparatively higher for variants of d2 i.e. (v1, v2, v3, v4, v5) as compared to the original documents. This clearly proves that more similar the document the less is the angle of difference between the documents and better the similarity metric.

b. Plot the similarity differences on a graph showing their cosine similarity score. Verify that your intuitions about what makes the differing docs less similar does indeed lead to scores that are less similar.

With the help of following code, a scatter plot is produced which shows cosine similarity between document 'd2' and all other documents.

```
import matplotlib.pyplot as plt
lt = cosines_d2.items()
x, y = zip(*lt)
plt.scatter(x,y,color='r')
plt.title('Scatter Plot', fontsize=15)
plt.xlabel('Documents', fontsize=15)
plt.ylabel('Cosine Similarity', fontsize=15)
plt.show()
```

Following is the output snippet -

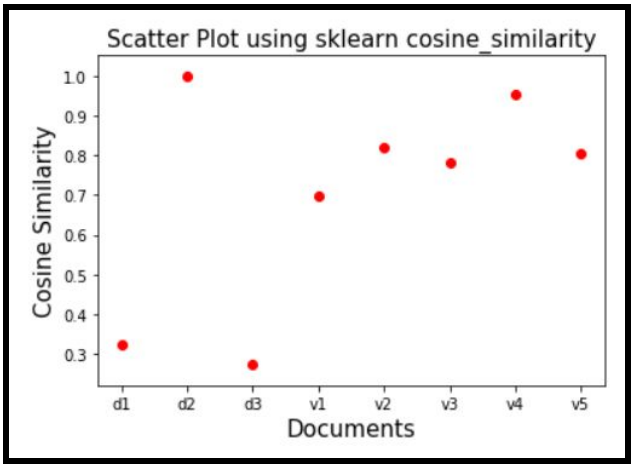


Here,
d1 → d3 := original documents.
v1 → v5 := variations of d2

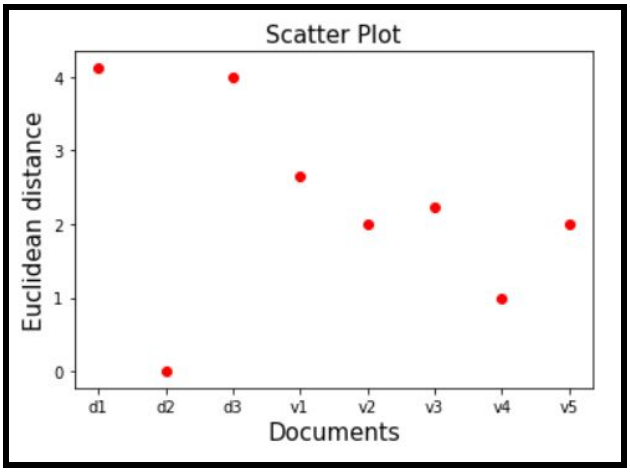
From the plot it is observed that our intuitions are true and more similarity is observed between the variants than the original documents with the base d2. This is because of the fact that variants are generated from d2 and while generation there are many words that are repeated. This in turn increases their similarities.

c. Find a python package that computes cosine similarity and euclidean distance. Use it process the data you have already. Do the answers for Cosine Similarity correspond? What do the Euclidean Distance scores look like relative to the Cosine ones?

The sklearn.metrics.pairwise library provides functions for both the cosine similarity and euclidean distance. Scatter plot for cosine similarity using sklearn library -



Here, we can observe that the plots for cosine similarity using cosine.py program and sklearn package are comparatively similar. However before computing cosine similarity we compute the TFIDF values for vectors which acts as the magnitude. The minor difference between both the plots occurs because of this difference in the TFIDF values.
Scatter plot for euclidean distance using sklearn library -



Euclidean distance is low for similar documents whereas it is high for dissimilar documents. *The plot for euclidean distance appears **inverted** compared to the cosine similarity.*
The results for euclidean distance can be better if the documents were longer and better represented in VSM. Euclidean distance gets equivalent to cosine similarity if we perform normalisation on data. (Euclidean vs. Cosine Distance, 2017). The **possibility behind this inverted result** is because of the fact that cosine similarity looks at the angle between two vectors whereas euclidean similarity at the distance between two points. To better explain this difference following example is referenced. (When to use cosine similarity over Euclidean similarity, 2018). Let's say you are in an e-commerce setting and you want to compare users for product recommendations:

- User 1 bought 1x eggs, 1x flour and 1x sugar.
- User 2 bought 100x eggs, 100x flour and 100x sugar
- User 3 bought 1x eggs, 1x Vodka and 1x Red Bull

By cosine similarity, user 1 and user 2 are more similar. By euclidean similarity, user 3 is more similar to user 1.

Q.3) Create or find 5 “normal” tweets from Twitter. Now take one of these tweets and systematically generate 20 SPAM tweets from it; using the typical techniques of spammers. Now, perform comparisons between these 20 SPAM tweets each of the 5 Normal Tweets. Plot their edit-distance scores in a graph and colour code to show how the SPAM v Normal ones. Are the SPAM tweets obvious, if not why?

The distance metric used here is levenshtein distance which is given by the formula -

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Following are the spam and non-spam tweets -

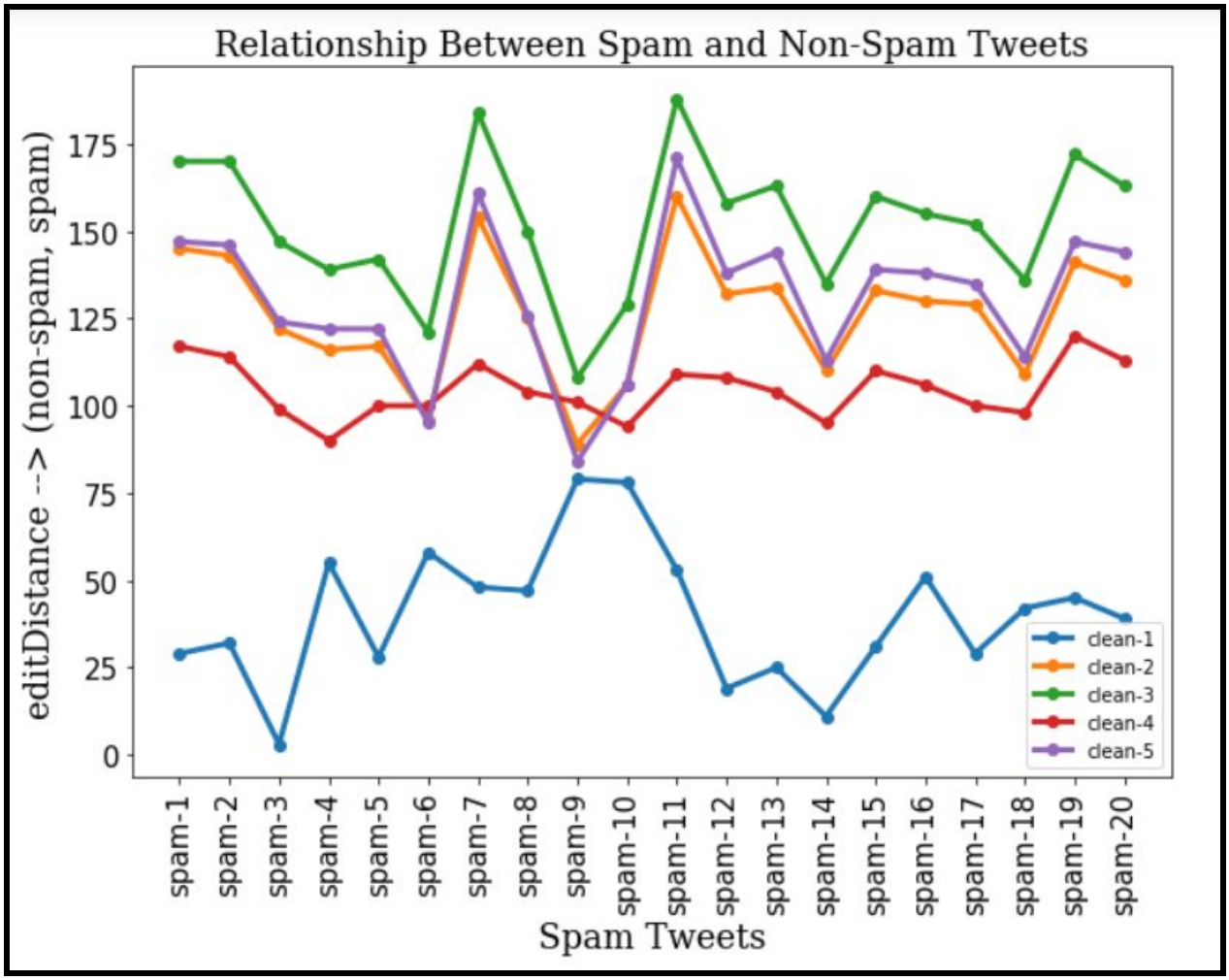
```
spam_set =
[
'Lets rake in dough? RT Retweet for a chance to #WinDominosPizza No shhame in this pizza game there are rules here - h
'Lets rake in dough? Retweet for a chance to #WinDominosPizza No shhame in this pizza game rules here - http://bit.ly/
'Rake in this dough thing? Go retweet for a chance to #WinDominosPizza No shame here the rules here - http://bit.ly/2k
'Go rake in this dough? Retweet for a chance to #WinDominosPizza NO shame in this pizza game the rules http://bit.ly/2
'Lets get doughy? Retweet for a chance to #WinDOminosPizzas Yaaayyyyy!!! Share this. Rules http://bit.ly/2kY0Y4rE',
'Are you ready to rake in the dough? Retweet for a chance to #WinDominosPizza No shame in this pizza game #Pizza  Rule
'Omg! Lets, retweet for a chance to #WinDominosPizza No shaming for this pizza game #Pizza  Rules here - http://bit.ly
'Ready to rake in our dough? #NationalPizzaMonth Lets play this game. Rules here - http://bit.ly/2kY0Y4r',
'Yaaay!!! Lets rake in dough? #NationalPizzaMonth Lets play this yummy pizzy game. Rules here - http://bit.ly/2kY0Y4rE
'Go ahead >>>  Lets start and rake in dough? Retweet for a chance to #WinDominosPizza No shhame in this pizza game rul
'Begin...Omg.. Lets rake in dough? Retweet for a chance to #WinDominosPizza No shame in this pizza game rules here - h
'And lets go!! Lets rake in dough? Retweet for a chance to #WinDominosPizza No shame in this pizza game rules here - h
'Lets rake in dough? Retweet for a chance to #WinDominosPizza No shame in this pizza game rules - http://bit.ly/2kY0Y4
'Yumm!! Lets rake in dough? Retweet for a chance to #WinDominosPizza shame in this pizza game rules here you go - http
'Nom! Are you ready to rake in dough? Retweet for  chance  #WinDominosPizza shame in this pizza game rules here - http
'Lets begin? Why not in dough? Retweet for a chance to #WinDominosPizza shame in this pizza game rules here - http://b
'Why not lets rake in dough? Retweet for  chance  #WinDominosPizza No shame in this pizza game rules http://bit.ly/2kY
'Can we lets rake in dough? Retweet for a chance to #WinDominosPizza There is no shame in this pizza game Yaaay!!!! Ru
'Why not lets rake in dough? Retweet for the chance to #WinDominosPizza shame in this pizza game Woah!! rules here - h
]

normal_set = ['Lets rake in dough? Retweet for a chance to #WinDominosPizza No shame in this pizza game. Rules here -
'Enjoys long walks to the fridge for leftover pizza. #DominosPizza #WinDominosPizza',
'We are giving away FREE PIZZA to celebrate #WinDominosPizza',
'Gear up for game day with pizza on us. Retweet for a chance to #WinDominosPizza in honor of #NationalPizzaMonth.',
'I am officially committed to #pizza nobody else can have any. #NationalPizzaMonth']
```

Following is the result of comparison(levenshtein distance) between 20 spam tweets and 5 normal tweets :-

	clean-1	clean-2	clean-3	clean-4	clean-5
spam-1	29	145	170	117	147
spam-2	32	143	170	114	146
spam-3	3	122	147	99	124
spam-4	55	116	139	90	122
spam-5	28	117	142	100	122
spam-6	58	96	121	100	95
spam-7	48	154	184	112	161
spam-8	47	125	150	104	126
spam-9	79	89	108	101	84
spam-10	78	106	129	94	106
spam-11	53	160	188	109	171
spam-12	19	132	158	108	138
spam-13	25	134	163	104	144
spam-14	11	110	135	95	113
spam-15	31	133	160	110	139
spam-16	51	130	155	106	138
spam-17	29	129	152	100	135
spam-18	42	109	136	98	114
spam-19	45	141	172	120	147
spam-20	39	136	163	113	144

The results for spam tweets are obvious. The clean-1 is taken as the base to produce 20 spam tweets. So it is clearly evident from the above result that the spam tweets are more similar and at less distance from the base(clean-1).

Following is the color coded line plot for the above results -



From the line plot we can infer the same fact that clean-1(base) tweet is more similar(at less distance) to spam tweets.

References :-

Euclidean Distance vs Cosine similarity, 2017, Available at: https://cmry.github.io/notes/euclidean-v-cosine

When to use cosine similarity over Euclidean similarity, 2018, Available at : https://datascience.stackexchange.com/questions/27726/when-to-use-cosine-simlarity-over-euclidean-similarity