Honours Degree in Computing

Data Analytics Assessment: Analyse a dataset

Submitted by:

Dylan Hallissey

B00089794

Submission date

21/08/2022

Declaration

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, except where otherwise stated. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I/We understand that plagiarism, collusion, and copying are grave and serious offences and accept the penalties that would be imposed should I/we engage in plagiarism, collusion or copying. I acknowledge that copying someone else's assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism. I/We have read and understood the colleges plagiarism policy 3AS08 (available here).

This material, or any part of it, has not been previously submitted for assessment for an academic purpose at this or any other academic institution.

I have not allowed anyone to copy my work with the intention of passing it off as their own work.

Name: Dylan Hallissey   Dated: 21/08/2022

Contents

Overview

The three categories of the topic that will be related to the project is Xbox, PlayStation and the Premier League football. The two topic that are very similar they are two gaming consoles and the other topic is based on football the premier league.

Business objective

The business objective of this project is to find the dataset of the three topics I choose . The three datasets will contain attributes and an class label. There are general attributes such as articles which will come up all the articles related to the topics that I've chosen. Which are PlayStation, Xbox and the premiere league. The plan for this project is to get the main article for the three main topics and have the articles print out for the classes.

Mining objective

This stage should generate a model to see how many words are in a class. The mining objective will be process collections of text with document which PlayStation, xbox and premiere League. Text Mining objective is a multi-stage process. With some informational of the content included in any of the items like newspaper, articles, books, reports, blogs and articles. The goal of the mining objective is to identify the useful information without duplication from various documents with synonymous understanding and this is what we be doing related to the mining objective

The process employed

There are 30 articles/documents with each class which be for premiere league and two classes that should be similar. Which is xbox and playstation

The goal is to focus on related textual data which shows text that can be stored in very large datasets. We will use the Beautiful Soup which is a library which makes it easy for us to scrape any information from web pages. To do this we will have to use a web structure mining tool, it would make us find a lot of useful information and knowledge of an analysis hyperlinks. With the 90 source pages that we have some of the links would have to be removed for example maybe they didn't have a image/picture and wasn't very structured to be able to add the link to the 90 sources. The scraping process has two main libraries which be requests and bs4. Request library will used to send are HTTP link request to a website and then it will store our response object within a variable, how we fetch the request is by using a beautiful soup which will catch the data using Html tag, class, etc. The second main library is the BS4 which will be used for pulling the data out of the HTML and XML files.

Select the relevant html elements data,

The text content of the relevant html elements, The category's for premier league , xbox and playstation, all each have a URLs to retrieve data from the h1 and p-elements, we can use the len for printing the length for documentation. Each class will contain the length of 30 documents. When all classes are together we use all_docs. This is used by displaying a result for all documents which be 90. We get text by using h1 and p. H1 and P are important elements to the articles. Why we use the developer tools is for the structure of the websites and clarify every elements that are important. The main developer tools that be scraped is h1 and p.

Building the corpus

Before the corpus be built we research more related to corpus to see how it works. The corpus is collection of documents, books or articles. It can be split into individual documents and also categories. It then be broken down into sentences, words or characters. We have to import the NLTK, this can offer some functionalities for corpus reading. We can use an Domain Specific Corpora tool for building application related to a specific language models. After we build a Domain Specific Corpora, we then map each document to a corresponding word from using a dictionary structure. After that's all done we use dataframe structures to build a labelled corpus, by using build corpus an "def build_corpus(docs,labels)" for literally building the corpus, then after use "corpus = build_corpus(all_docs, all_labels)" to relate to all of the 90 source documents. Once the corpus is built, it be written as a csv file, then we use the csv function from importing pandas.

Thw 3 classes and the labels are then finalled contained into one. The corpus DataFrame contains the article and class with there documents, therefore its why we use len for the corpus. The corpus display random rows and columns example 90 rows and columns. With the corpus file called corpus.cvs already we use this by reduce the size of each of the document.

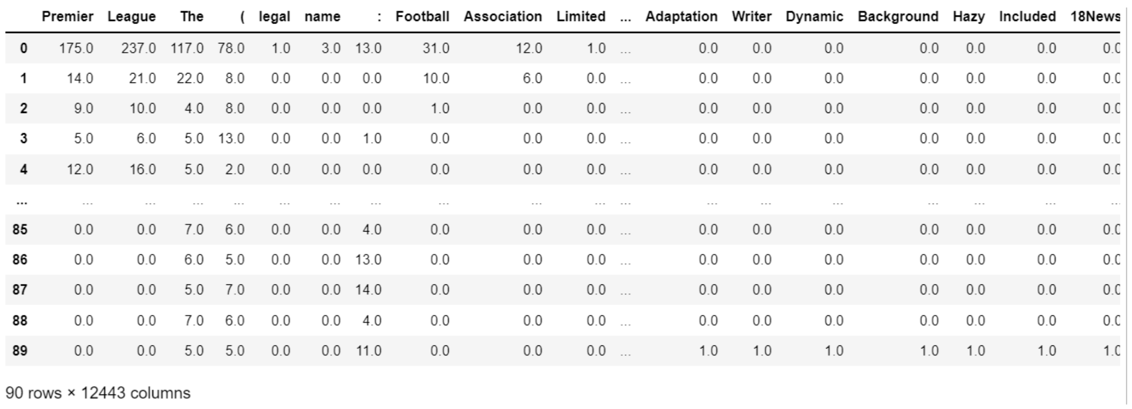
Baseline Models

Derive 3 matrices using 3 vectorisation techniques

The three matrices which be able to use for the baseline model is counts, normalised counts and tfidf. The matrices is important mainly because we can extract set of unique words/tokens from the documents and we can use them to create an matrix with entries that will let us know what occurred in the document. The matrices generate every document that's in our dataset which are dataset be for our project is corpus dataset. Tackling this task we will use three matrices. Start of using first of use def build_count_matrix to tokenise the corpus, then we will create group list of dictionaries they be are frequency distributions. while documenting tokenised in the def build count we generate a dictionary in the documents.

We start of now using a def build_tf_matrix method. We use this function to generate matrix with normalised frequencies. The build_tf_matrix uses the count_matrix and the doc_lengths for normalising the frequency. The third matrices uses the def build_tfidf_matrix(docs). Therefore the matrices function will generate a matrix with tfidfs scores.

Conclusion of this this be the tf is will have the number of times that a word occurs in the documents but will divide by total number with words in the document however a cell entry represents normalized frequencies count of total number of words in a document. for the Derive 3 matrices using 3 vectorisation techniques is the difference between the three matrices and the count matrices will create list of dictionaries and generate an count for dictionary in each document. The Idf is number documents which be collection that is divide by number documents where words occurs and the normalised matrices will generate the matrices for the normalised frequencies and the tfidf matrices will generate a score. The resulting matrix contains features of 12443 columns attributes with 90 documents.



	Premier	League	The	(	legal	name	:	Football	Association	Limited	...	Adaptation	Writer	Dynamic	Background	Hazy	Included	18News
0	175.0	237.0	117.0	78.0	1.0	3.0	13.0	31.0	12.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.C
1	14.0	21.0	22.0	8.0	0.0	0.0	0.0	10.0	6.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.C
2	9.0	10.0	4.0	8.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.C
3	5.0	6.0	5.0	13.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.C
4	12.0	16.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.C
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	0.0	0.0	7.0	6.0	0.0	0.0	4.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.C
86	0.0	0.0	6.0	5.0	0.0	0.0	13.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.C
87	0.0	0.0	5.0	7.0	0.0	0.0	14.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.C
88	0.0	0.0	7.0	6.0	0.0	0.0	4.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.C
89	0.0	0.0	5.0	5.0	0.0	0.0	11.0	0.0	0.0	0.0	...	1.0	1.0	1.0	1.0	1.0	1.0	1.C

90 rows × 12443 columns

Choose at least 1 classification algorithm for baseline modelling

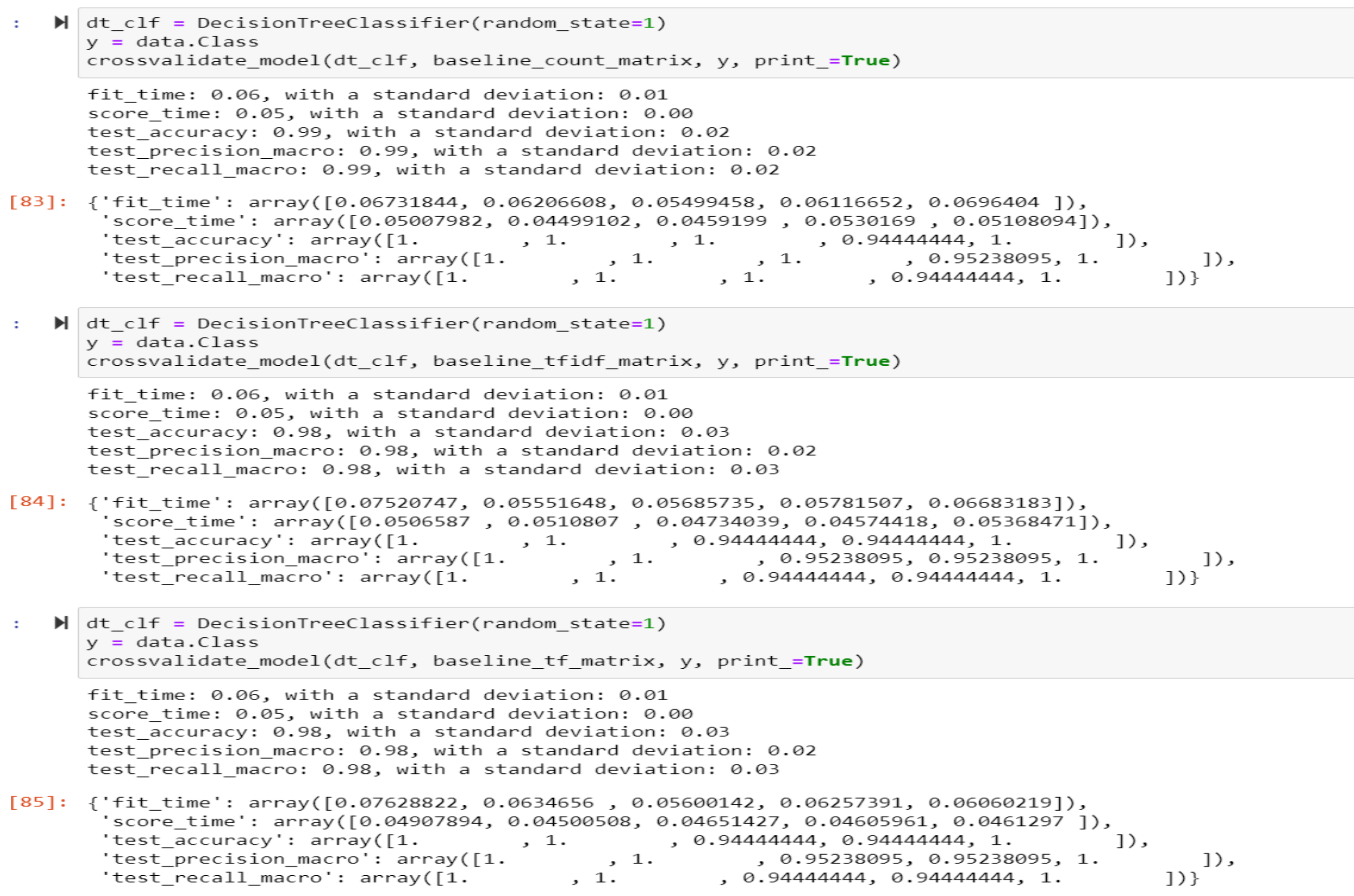
The 1 classification algorithm to use with the baseline model will be the Decision tree. We used the decision tree analysis for outlining outcomes, costs, and frequencies of decision that's complex. Why we use this classification is because its easy analysing quantitative data and having a decision that's based with numbers and is very helpful finding result. This will make it all easy to use and apply it to the 3 matrices and td our result. Another option was KNeighborsClassifier but thought the decision tree be better to use.

Apply the algorithm to the 3 matrices

We used the decision tree algorithm to the 3 matrices by using the crossvalidate_model for their performances. By apply with the decision tree matrices we use "DecisionTreeClassifier(random_state=1)" to randomly state the scores, then finding the data class with using "y = data.Class".

We will then use a cross validation method for finding performance with using "crossvalidate_model(dt_clf, baseline_count_matrix, y, print_=True)".

In the crossvalidate_model we put it in our matrices in the brackets of that method and print out the results. For each matrices we use with using the algorithm it be document the results for the image below for example of the result for each algorithm to the 3 matrices. We used to the count. tf and tfidf and find the results for each score time, percission and recall.



```
dt_clf = DecisionTreeClassifier(random_state=1)
y = data.Class
crossvalidate_model(dt_clf, baseline_count_matrix, y, print_=True)

fit_time: 0.06, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.99, with a standard deviation: 0.02
test_precision_macro: 0.99, with a standard deviation: 0.02
test_recall_macro: 0.99, with a standard deviation: 0.02

[83]: {'fit_time': array([0.06731844, 0.06206608, 0.05499458, 0.06116652, 0.0696404 ]),
       'score_time': array([0.05007982, 0.04499102, 0.0459199 , 0.0530169 , 0.05108094]),
       'test_accuracy': array([1.        , 1.        , 1.        , 0.94444444, 1.        ]),
       'test_precision_macro': array([1.        , 1.        , 1.        , 0.95238095, 1.        ]),
       'test_recall_macro': array([1.        , 1.        , 1.        , 0.94444444, 1.        ])}

dt_clf = DecisionTreeClassifier(random_state=1)
y = data.Class
crossvalidate_model(dt_clf, baseline_tfidf_matrix, y, print_=True)

fit_time: 0.06, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.98, with a standard deviation: 0.03
test_precision_macro: 0.98, with a standard deviation: 0.02
test_recall_macro: 0.98, with a standard deviation: 0.03

[84]: {'fit_time': array([0.07520747, 0.05551648, 0.05685735, 0.05781507, 0.06683183]),
       'score_time': array([0.0506587 , 0.0510807 , 0.04734039, 0.04574418, 0.05368471]),
       'test_accuracy': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ]),
       'test_precision_macro': array([1.        , 1.        , 0.95238095, 0.95238095, 1.        ]),
       'test_recall_macro': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ])}

dt_clf = DecisionTreeClassifier(random_state=1)
y = data.Class
crossvalidate_model(dt_clf, baseline_tf_matrix, y, print_=True)

fit_time: 0.06, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.98, with a standard deviation: 0.03
test_precision_macro: 0.98, with a standard deviation: 0.02
test_recall_macro: 0.98, with a standard deviation: 0.03

[85]: {'fit_time': array([0.07628822, 0.0634656 , 0.05600142, 0.06257391, 0.06060219]),
       'score_time': array([0.04907894, 0.04500508, 0.04651427, 0.04605961, 0.0461297 ]),
       'test_accuracy': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ]),
       'test_precision_macro': array([1.        , 1.        , 0.95238095, 0.95238095, 1.        ]),
       'test_recall_macro': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ])}
```

Data Understanding

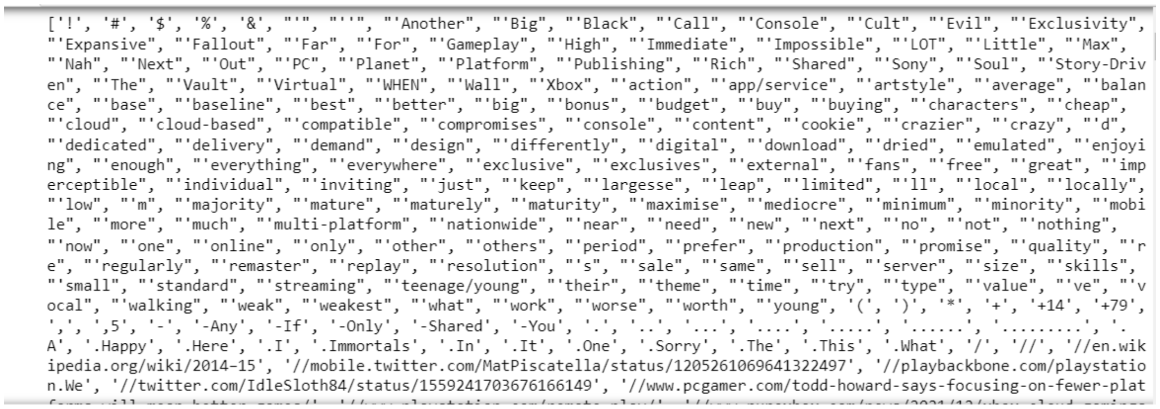
Derive word/token statistics

A token for our project would be a unit that represents any word or character sequence as it appears in text. For the project we have 30 token types appears twice. In the project it represents as one type and It’s how Token works meaning we would have to derive a token in a category.

while using Word Tokenization it divides a text into words that were delimited by characters that act as word boundaries. Word Tokenization is used cause its known for generating word vectors for mining. While finding the number of token statics of 30 of each category, we would use the most Frequequent method to find the text of each token: Premier League : “tm.print_n_mostFrequent("premierleague", premierleague_text, 30)” playstation: tm.print_n_mostFrequent("playstation", playstation_text, 30) xbox: tm.print_n_mostFrequent("xbox", xbox_text, 30). Therefore now we will use the

attributes =sorted(set(list(baseline_count_matrix.columns))) print(attributes)

to print out the word and token.



```
['!', '#', '$', '%', '&', "'", "''", "'Another", "'Big", "'Black", "'Call", "'Console", "'Cult", "'Evil", "'Exclusivity",
"'Expansive", "'Fallout", "'Far", "'For", "'Gameplay", "'High", "'Immediate", "'Impossible", "'LOT", "'Little", "'Max",
"'Nah", "'Next", "'Out", "'PC", "'Planet", "'Platform", "'Publishing", "'Rich", "'Shared", "'Sony", "'Soul", "'Story-Driv
en", "'The", "'Vault", "'Virtual", "'WHEN", "'Wall", "'Xbox", "'action", "'app/service", "'artstyle", "'average", "'balan
ce", "'base", "'baseline", "'best", "'better", "'big", "'bonus", "'budget", "'buy", "'buying", "'characters", "'cheap",
"'cloud", "'cloud-based", "'compatible", "'compromises", "'console", "'content", "'cookie", "'crazier", "'crazy", "'d",
"'dedicated", "'delivery", "'demand", "'design", "'differently", "'digital", "'download", "'dried", "'emulated", "'enjoyi
ng", "'enough", "'everything", "'everywhere", "'exclusive", "'exclusives", "'external", "'fans", "'free", "'great", "'imp
erceptible", "'individual", "'inviting", "'just", "'keep", "'largesse", "'leap", "'limited", "'ll", "'local", "'locally",
"'low", "'m", "'majority", "'mature", "'maturely", "'maturity", "'maximise", "'mediocre", "'minimum", "'minority", "'mobi
le", "'more", "'much", "'multi-platform", "'nationwide", "'near", "'need", "'new", "'next", "'no", "'not", "'nothing",
"'now", "'one", "'online", "'only", "'other", "'others", "'period", "'prefer", "'production", "'promise", "'quality", "'r
e", "'regularly", "'remaster", "'replay", "'resolution", "'s", "'sale", "'same", "'sell", "'server", "'size", "'skills",
"'small", "'standard", "'streaming", "'teenage/young", "'their", "'theme", "'time", "'try", "'type", "'value", "'ve", "'v
ocal", "'walking", "'weak", "'weakest", "'what", "'work", "'worse", "'worth", "'young", '(', ')', '*', '+', '+14', '+79',
',', ',5', '-', '-Any', '-If', '-Only', '-Shared', '-You', '.', '..', '...', '....', '.....', '......', '.........', '.
A', '.Happy', '.Here', '.I', '.Immortals', '.In', '.It', '.One', '.Sorry', '.The', '.This', '.What', '/', '//', '//en.wik
ipedia.org/wiki/2014-15', '//mobile.twitter.com/MatPiscatella/status/1205261069641322497', '//playbackbone.com/playstatio
n.We', '//twitter.com/IdleSloth84/status/1559241703676166149', '//www.pcgamer.com/todd-howard-says-focusing-on-fewer-plat
```

visualisations techniques

Bar charts

The normalise count for pos with each document takes a list of tagged docs. When we plot the frequency for pos text w ehave it in a list meaning the function would plot a bar charts with Frequency distribution of CC , label Frequency.

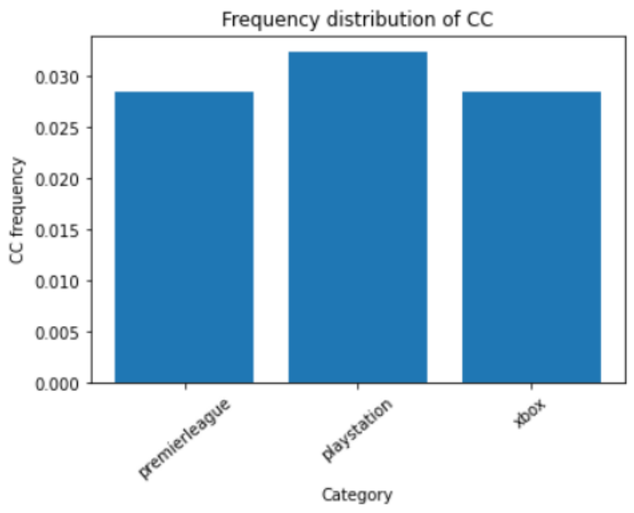
To visualise a frequency for CC part of speech we have to start of with documents to store a text = [premierleague_text, playstation_text, xbox_text], then visualise the frequency of CC part of speech which be conjunctions. Then across the 3 categories we use plot_POS_freq(texts, 'CC', ['premierleague', 'playstation', 'xbox']).

For the CC frequency the PlayStation is the most occurring at 0.030, but the premier league and xbox are on the same frequency with 0.028.

We then change CC to CD (Cardinal number), the one that oocurs with the most articles or classes is premier league with 0.06, the xbox with 0.03 and the PlayStation this time is the lowest with 0.01.

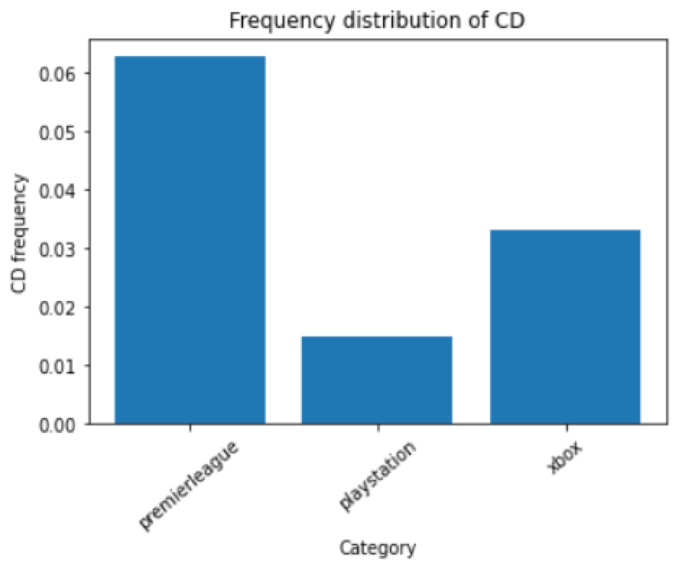
Frequency distribution of CC

The part of speech for CC, the highest occurring from the documents is Playstation and then xbox and premier league is the same.



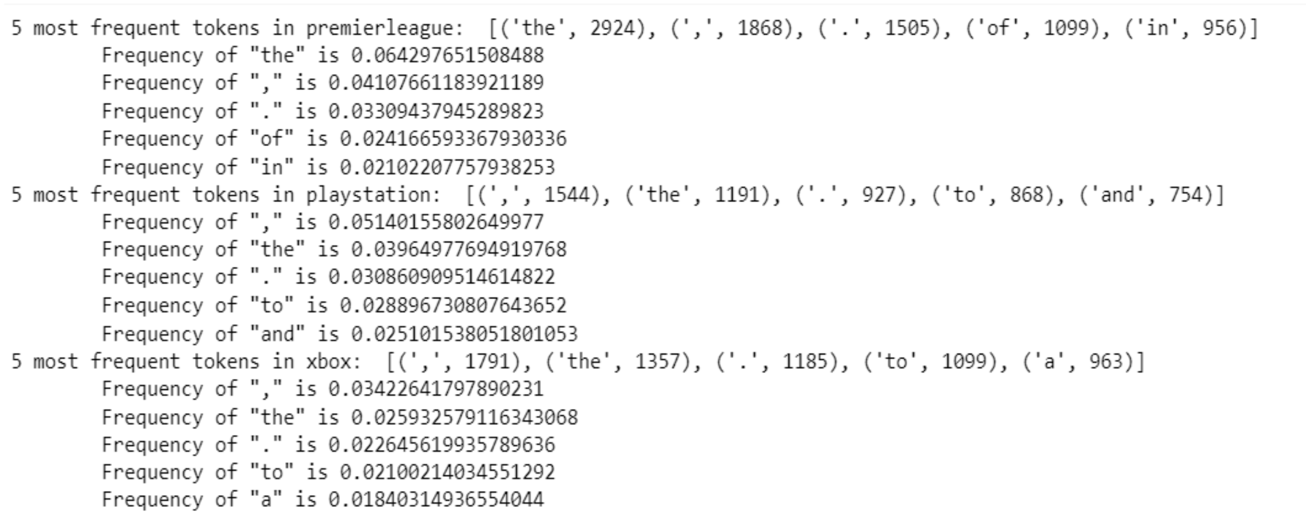
Frequency distribution of CD

The part of speech for CD, the highest occurring from the documents is premier league, then its xbox and last playstation.



Identify frequently occurring terms, potential stop words, synonyms, concepts, and word variations

We will see what is the most 5 frequently occurring terms, potential stop words, synonyms, concepts, and word variations for each class. We check for each premier league, PlayStation and xbox for the most frequently that occurs.



```
5 most frequent tokens in premierleague:  [('the', 2924), (',', 1868), ('.', 1505), ('of', 1099), ('in', 956)]
        Frequency of "the" is 0.064297651508488
        Frequency of "," is 0.04107661183921189
        Frequency of "." is 0.03309437945289823
        Frequency of "of" is 0.024166593367930336
        Frequency of "in" is 0.02102207757938253
5 most frequent tokens in playstation:  [(',', 1544), ('the', 1191), ('.', 927), ('to', 868), ('and', 754)]
        Frequency of "," is 0.05140155802649977
        Frequency of "the" is 0.03964977694919768
        Frequency of "." is 0.030860909514614822
        Frequency of "to" is 0.028896730807643652
        Frequency of "and" is 0.025101538051801053
5 most frequent tokens in xbox:  [(',', 1791), ('the', 1357), ('.', 1185), ('to', 1099), ('a', 963)]
        Frequency of "," is 0.03422641797890231
        Frequency of "the" is 0.025932579116343068
        Frequency of "." is 0.022645619935789636
        Frequency of "to" is 0.02100214034551292
        Frequency of "a" is 0.01840314936554044
```

For the premier league for the most 5 frequency tokens:
[('the', 2924), (',', 1868), ('.', 1505), ('of', 1099), ('in', 956).
'The' occurs the most 2924 times and frequency of 0.06, then ',' stop word occurs 1868 times frequency of 0.04

For the xbox frequency for the most 5 frequency tokens :
[(',', 1544), ('the', 1191), ('.', 927), ('to', 868), ('and', 754)]
'The' is the most word appeared 1191 times, frequency of 0.03 and the most stop wor ds occur was ',' 1544 times with frequency of 0.05.

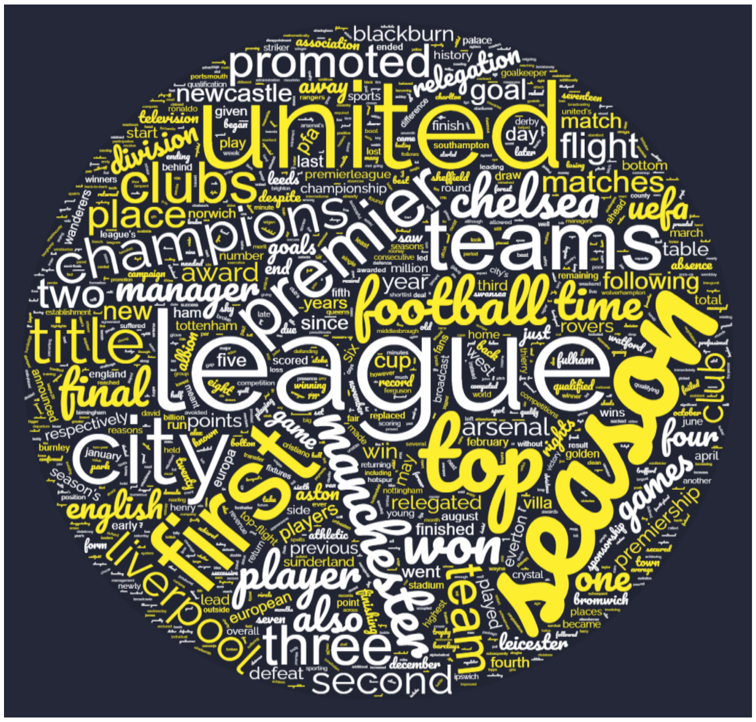
For the xbox frequency for the most 5 frequency tokens :
[(',', 1791), ('the', 1357), ('.', 1185), ('to', 1099), ('a', 963)]
'The' is the most word appeared 1357 times, frequency of 0.02 and the most stop wor ds occur was ',' 1791 times with frequency of 0.03.

Word cloud

The word cloud is to visual show frequency for terms with each documents for premier league, xbox and playstation. The word cloud great tool for mainly identifying its frequent/predictive terms. its great for identify synonyms and words variations and it also identify which terms occur for each topics. I had to use a website to get the word cloud, by trying to use it on jupyter lab it wouldn't work therefore I used a website that a lecturer provided the class to use. The Website is used to the word cloud for me, the website is wordcloud.com http://www.wordclouds.com. To get the word cloud working we use the corpus.csv to print out each class for the premier league, xbox and playstation. Therefore I was able to get the results for the word cloud to show what words occur the most in the document for the articles.

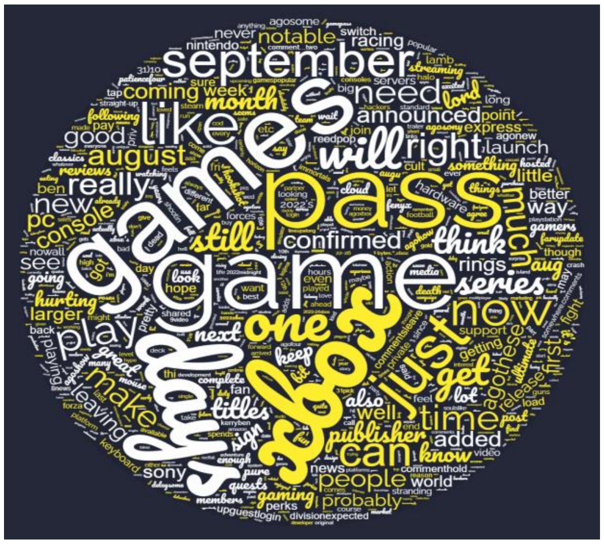
Premier League

This is the word cloud is for the premier league. generate the word cloud all the text with the links that's connected to the premier league. The visual representation with all frequency terms you can see appear the most be club names , season, title, top, season, promoted all these words you see are words that occur the most in our documents for this topic.



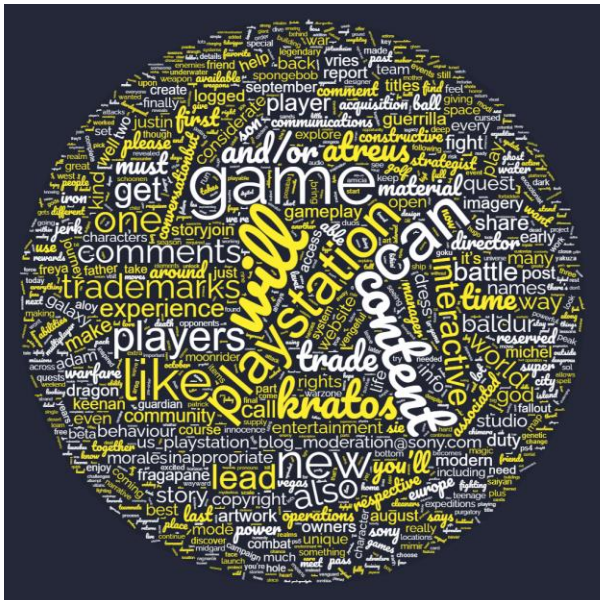
Xbox

This is the word cloud is for the xbox, it generate the word cloud all the text with the links that's connected to the xbox. The visual representation with all frequency terms you can see appear the most be games, game pass, days, gaming, console and xbox one all these words you see are words that occur the most in our documents for this topic.



PlayStation

This is the word cloud is for the playstation, it generate the word cloud all the text with the links that's connected to the playstation. The visual representation with all frequency terms you can see appear the most be game, players, lead, new, comments and content all these words you see are words that occur the most in our documents for this topic.



Use 2 clustering algorithms with 2 different linkage schemes

The two linkage we will be using is single and complete. We start of explaining single linkage and then the complete linkage for each clustering and linkage schemes. The Agglomerative and K mean clustering. the cosine is a Asymmetric, it cosine similarity is measured cosine of the angle between vectors.

Cosine is main cluster for the project mainly cause we now have a means to calculate distance. The first clustering we will use is the K means, we using linkage of single and complete so we will use that first for the kmeans method then we try the agglomerative.

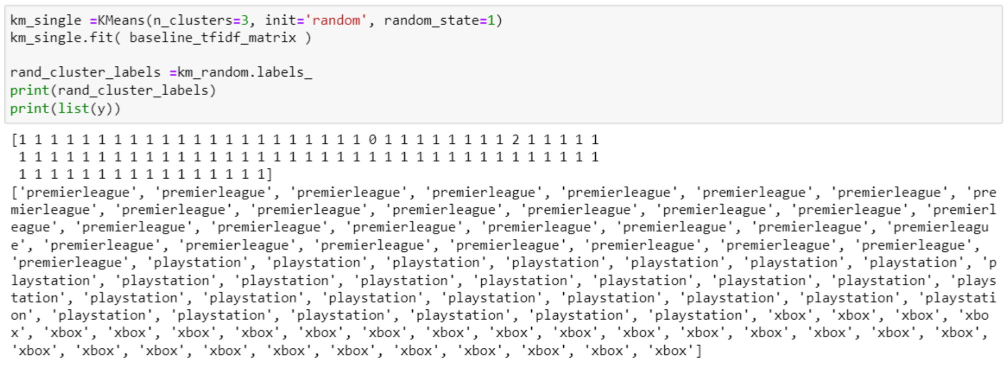
KMeans

The algorithm organises objects into the specified number of clusters. The K means cluster is the objects from the same cluster being to close eachother, and then objects from different clusters are far away from eachother.

We have rows of the data to be selected at random, they represent a cluster mean and centre. a new mean is calculated for each cluster when the objects are all allocated. Then tuples in the database will be compared to an new mean. That's how we use the K Mean cluster for the project.

Single:

We tried the KM_random. Labels for the single to get the result. The is loads of clustering in this.

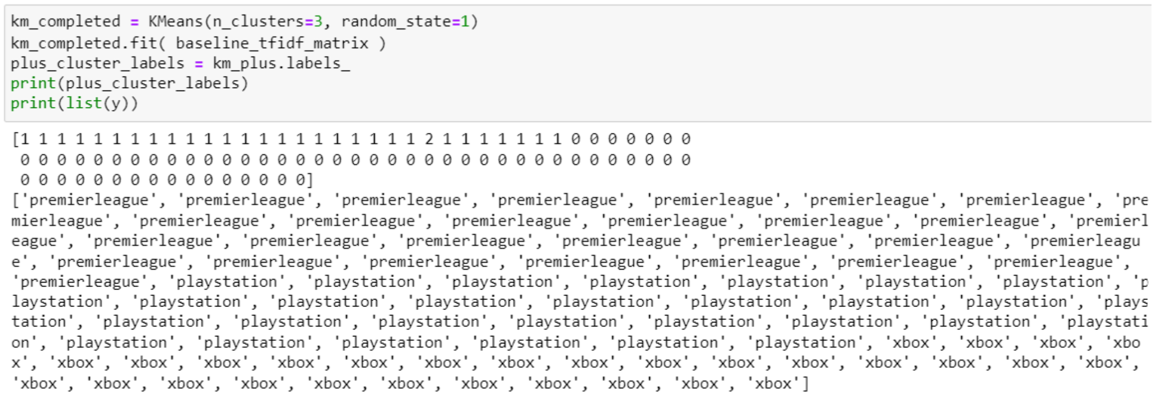


```
km_single =KMeans(n_clusters=3, init='random', random_state=1)
km_single.fit( baseline_tfidf_matrix )

rand_cluster_labels =km_random.labels_
print(rand_cluster_labels)
print(list(y))
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 2 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
['premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'pre
mierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierl
eague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleagu
e', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague',
'premierleague', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'p
laystation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'plays
tation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstati
on', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'xbox', 'xbox', 'xbox', 'xbo
x', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox',
'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox']
```

Completed:

It has less clusterin 1s than it did before there is now more 0s thans 1st while its completed.



```
km_completed = KMeans(n_clusters=3, random_state=1)
km_completed.fit( baseline_tfidf_matrix )
plus_cluster_labels = km_plus.labels_
print(plus_cluster_labels)
print(list(y))
```

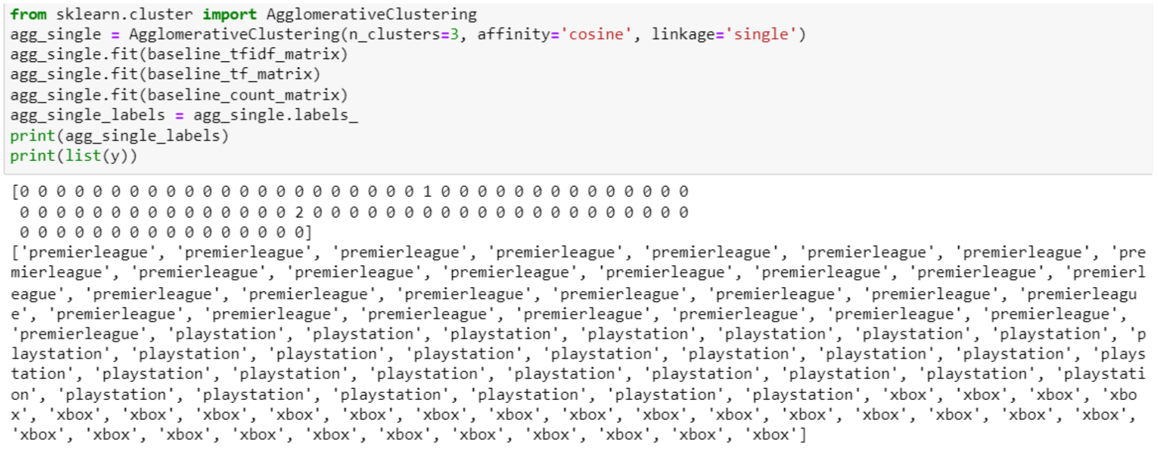
```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
['premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'pre
mierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierl
eague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleagu
e', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague',
'premierleague', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'p
laystation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'plays
tation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstati
on', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'xbox', 'xbox', 'xbox', 'xbo
x', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox',
'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox']
```

Agglomerative Clustering

The Agglomerative Clustering we got to use to connect with the linkage are also single and complete. By getting the Agglomerative to work we got to use the homogeneity and the completeness score to work, they score for an single approach. The only issue was the dendrogram it was to close to each other and couldn't see the bottom part of the numbers. But overall the algorithms, schemes and measures worked. The affinity performed for this was cosine.

Single:

You can see we have 1 in one document and 2s which means cluster 1 document

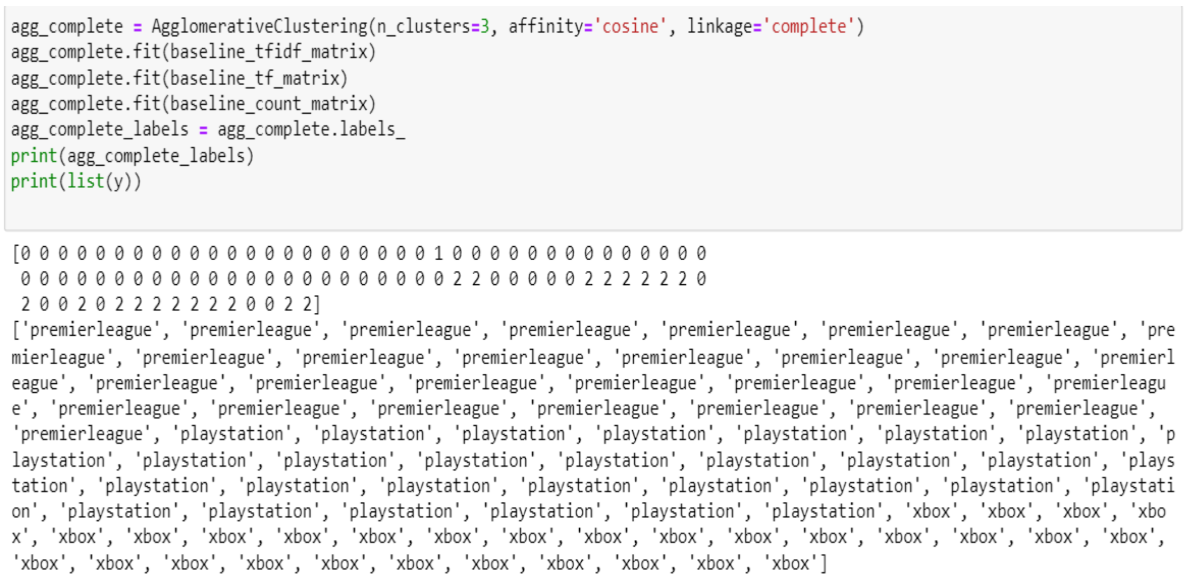


```
from sklearn.cluster import AgglomerativeClustering
agg_single = AgglomerativeClustering(n_clusters=3, affinity='cosine', linkage='single')
agg_single.fit(baseline_tfidf_matrix)
agg_single.fit(baseline_tf_matrix)
agg_single.fit(baseline_count_matrix)
agg_single_labels = agg_single.labels_
print(agg_single_labels)
print(list(y))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
['premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'pre
mierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierl
eague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleagu
e', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague',
'premierleague', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'p
laystation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'plays
tation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstati
on', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'xbox', 'xbox', 'xbox', 'xbo
x', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox',
'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox']
```

Completed:

You can see its been completed and there are now more 2s. so this performed bad.



```
agg_complete = AgglomerativeClustering(n_clusters=3, affinity='cosine', linkage='complete')
agg_complete.fit(baseline_tfidf_matrix)
agg_complete.fit(baseline_tf_matrix)
agg_complete.fit(baseline_count_matrix)
agg_complete_labels = agg_complete.labels_
print(agg_complete_labels)
print(list(y))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 2 2 2 2 2 2 0
 2 0 0 2 0 2 2 2 2 2 2 2 0 0 2 2]
['premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'pre
mierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierl
eague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleagu
e', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague',
'premierleague', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'p
laystation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'plays
tation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstati
on', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'xbox', 'xbox', 'xbox', 'xbo
x', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox',
'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox']
```

Main Data Preparation

Text preprocessing tasks

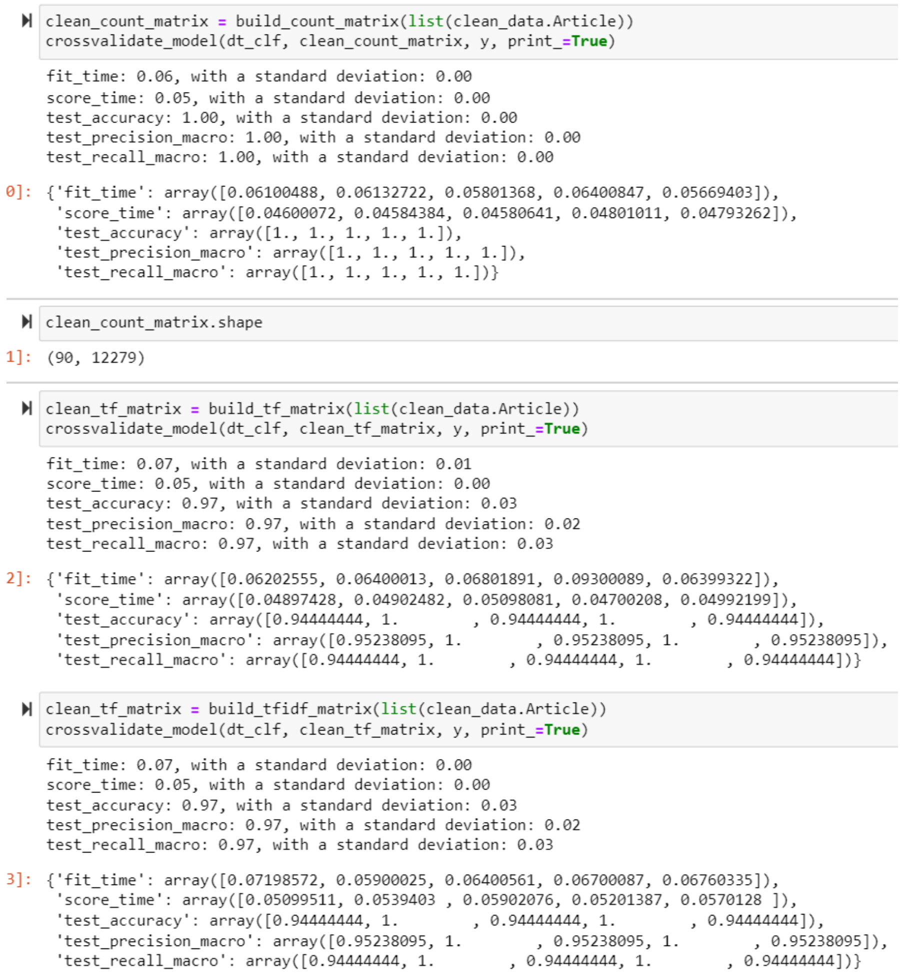
The pre-processing tasks main processing for data is initial clean data, improving bag of words and stop words. It can clean data which would reduce number of things that aren't useful. such. Improving bag of words will reduced number things like the terms that replaced by synonyms. Like replacing word variation game_play to gameplay which is the same. Its changing hyponyms to corresponding hypernyms. Then Stop words removal removes attributes because mainly lot of filters contain barely any information, with removing them would increase performance big time.

Initial Cleaning

The first task I am picking for the ask is Initial Cleaning. Initial cleaning it will generate new modules and it counts my projects, but I do this mainly cause it improves scores with fewer attributes. I use this with a count, tfidf and count.

The performance of initial clean data for the count has test accuracy 1.00, the test precision macro is 1 and same as test recall and then the standard deviation is 0 for the tf matrix and tfidf both have same with test accuracy 0.97, the test precision and test recall macro is 0.97 but different standard deviation.

Therefore the matrix count has a higher test accuracy, test precision and test recall.



```
clean_count_matrix = build_count_matrix(list(clean_data.Article))
crossvalidate_model(dt_clf, clean_count_matrix, y, print_=True)

fit_time: 0.06, with a standard deviation: 0.00
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 1.00, with a standard deviation: 0.00
test_precision_macro: 1.00, with a standard deviation: 0.00
test_recall_macro: 1.00, with a standard deviation: 0.00

0]: {'fit_time': array([0.06100488, 0.06132722, 0.05801368, 0.06400847, 0.05669403]),
     'score_time': array([0.04600072, 0.04584384, 0.04580641, 0.04801011, 0.04793262]),
     'test_accuracy': array([1., 1., 1., 1., 1.]),
     'test_precision_macro': array([1., 1., 1., 1., 1.]),
     'test_recall_macro': array([1., 1., 1., 1., 1.])}
```

```
clean_count_matrix.shape

1]: (90, 12279)
```

```
clean_tf_matrix = build_tf_matrix(list(clean_data.Article))
crossvalidate_model(dt_clf, clean_tf_matrix, y, print_=True)

fit_time: 0.07, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.97, with a standard deviation: 0.03
test_precision_macro: 0.97, with a standard deviation: 0.02
test_recall_macro: 0.97, with a standard deviation: 0.03

2]: {'fit_time': array([0.06202555, 0.06400013, 0.06801891, 0.09300089, 0.06399322]),
     'score_time': array([0.04897428, 0.04902482, 0.05098081, 0.04700208, 0.04992199]),
     'test_accuracy': array([0.94444444, 1.        , 0.94444444, 1.        , 0.94444444]),
     'test_precision_macro': array([0.95238095, 1.        , 0.95238095, 1.        , 0.95238095]),
     'test_recall_macro': array([0.94444444, 1.        , 0.94444444, 1.        , 0.94444444])}
```

```
clean_tf_matrix = build_tfidf_matrix(list(clean_data.Article))
crossvalidate_model(dt_clf, clean_tf_matrix, y, print_=True)

fit_time: 0.07, with a standard deviation: 0.00
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.97, with a standard deviation: 0.03
test_precision_macro: 0.97, with a standard deviation: 0.02
test_recall_macro: 0.97, with a standard deviation: 0.03

3]: {'fit_time': array([0.07198572, 0.05900025, 0.06400561, 0.06700087, 0.06760335]),
     'score_time': array([0.05099511, 0.0539403 , 0.05902076, 0.05201387, 0.0570128 ]),
     'test_accuracy': array([0.94444444, 1.        , 0.94444444, 1.        , 0.94444444]),
     'test_precision_macro': array([0.95238095, 1.        , 0.95238095, 1.        , 0.95238095]),
     'test_recall_macro': array([0.94444444, 1.        , 0.94444444, 1.        , 0.94444444])}
```

Improving Bag of Words

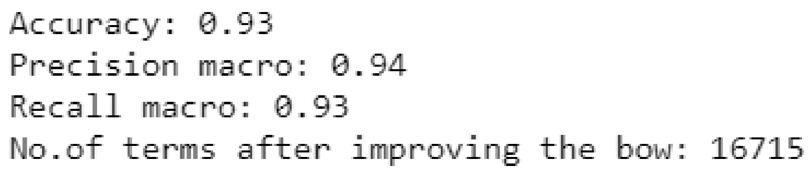
The second task is Improving Bag of Words which can improve the project including recognising word relations, Recognise informal word and Recognise Collocations of phrases or expressions and Linking canonical forms.

We use these to improve the bow for our project. But we replace synonyms we use 'premierleague': ['premier[_]?league'],

'xbox': ['gaming'],

'playstation': ['\bgame(s)?\b', 'player(s)?', 'platform(s)?']

We do concepts to replace general selection to identify. The results we with the number of terms after improving the bow was very high it was 16715.

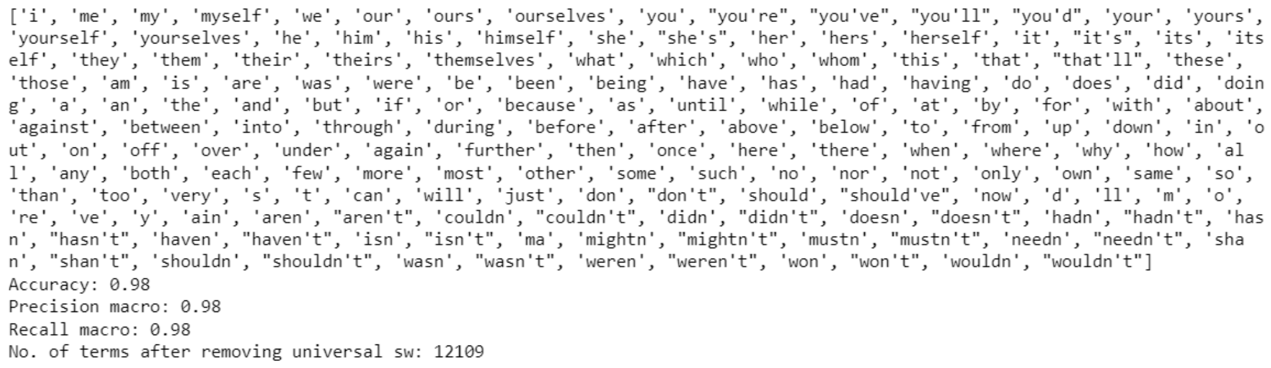


```
Accuracy: 0.93
Precision macro: 0.94
Recall macro: 0.93
No.of terms after improving the bow: 16715
```

Stop words Removal

The third task was Stop words Removal. I used this to reduce the number features, it be based on the text exploration. We use the custom list for stop removal words, meaning it decreases all scores and we will stick with just using the universal one svm for the project, mainly cause svm one that works perfect for my project.

The terms after removing universal sw is 12109.



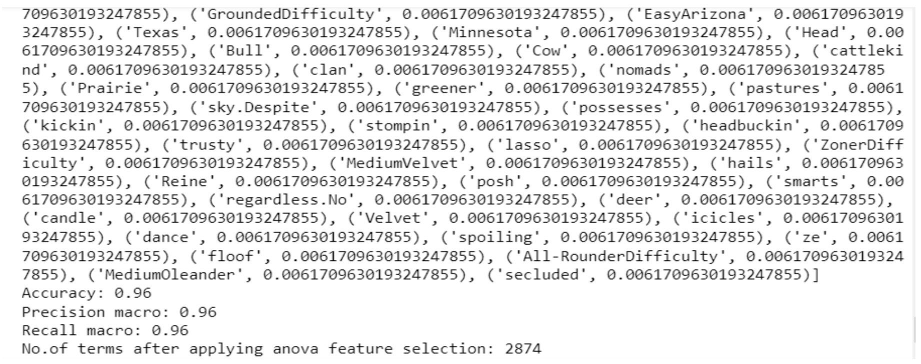
```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'its
elf', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doin
g', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'o
ut', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'al
l', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o',
're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'has
n', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'sha
n', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
Accuracy: 0.98
Precision macro: 0.98
Recall macro: 0.98
No. of terms after removing universal sw: 12109
```

Algorithms-based Feature selection/reduction tasks

I will now have to choose least 2 techniques for the algoritm based feature selection to reduce the tasks. I will Document and discuss the performance after each applied technique and decide which one to include in the final pipeline. The two technique I will test out is meta section and RFE selection.

Meta Section

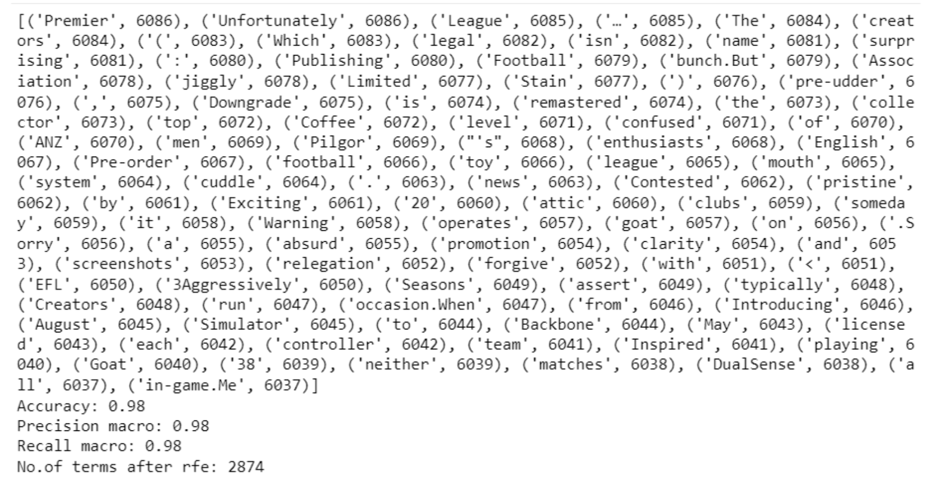
The meta section shows amount times word occurs in documents. for example. The meta section performances is high, the accuracy, precision and recall is both 96% and terms applying anova is high aswell with 2874.



```
709630193247855), ('GroundedDifficulty', 0.006170963019324785
5), ('EasyArizona', 0.006170963019
3247855), ('Texas', 0.006170963019324785
5), ('Minnesota', 0.006170963019324785
5), ('Head', 0.00
6170963019324785
5), ('Bull', 0.006170963019324785
5), ('Cow', 0.006170963019324785
5), ('cattleki
nd', 0.006170963019324785
5), ('clan', 0.006170963019324785
5), ('nomads', 0.00617096301932478
5
5), ('Prairie', 0.006170963019324785
5), ('greener', 0.006170963019324785
5), ('pastures', 0.0061
709630193247855), ('sky.Despite', 0.006170963019324785
5), ('possesses', 0.006170963019324785
5),
('kickin', 0.006170963019324785
5), ('stompin', 0.006170963019324785
5), ('headbuckin', 0.0061709
630193247855), ('trusty', 0.006170963019324785
5), ('lasso', 0.006170963019324785
5), ('ZonerDiff
iculty', 0.006170963019324785
5), ('MediumVelvet', 0.006170963019324785
5), ('hails', 0.006170963
0193247855), ('Reine', 0.006170963019324785
5), ('posh', 0.006170963019324785
5), ('smarts', 0.00
6170963019324785
5), ('regardless.No', 0.006170963019324785
5), ('deer', 0.006170963019324785
5),
('candle', 0.006170963019324785
5), ('Velvet', 0.006170963019324785
5), ('icicles', 0.00617096301
93247855), ('dance', 0.006170963019324785
5), ('spoiling', 0.006170963019324785
5), ('ze', 0.0061
709630193247855), ('floof', 0.006170963019324785
5), ('All-RounderDifficulty', 0.00617096301932 4
7855), ('MediumOleander', 0.006170963019324785
5), ('secluded', 0.006170963019324785
5)]
Accuracy: 0.96
Precision macro: 0.96
Recall macro: 0.96
No.of terms after applying anova feature selection: 2874
```

RFE selections

The RFE selections is to get rid of features by selecting number of features to eliminate. RFE seem to be higher than the meta section. Its accuracy, precision and recall is 98% which is very high even higher than 96% meta. Therefore we are going to use the RFE selection for the final pipeline.



```
[('Premier', 6086), ('Unfortunately', 6086), ('League', 6085), ('…', 6085), ('The', 6084), ('creat
ors', 6084), ('(', 6083), ('Which', 6083), ('legal', 6082), ('isn', 6082), ('name', 6081), ('surpr
ising', 6081), (':', 6080), ('Publishing', 6080), ('Football', 6079), ('bunch.But', 6079), ('Assoc
iation', 6078), ('jiggly', 6078), ('Limited', 6077), ('Stain', 6077), (')', 6076), ('pre-udder', 6
076), (',', 6075), ('Downgrade', 6075), ('is', 6074), ('remastered', 6074), ('the', 6073), ('colle
ctor', 6073), ('top', 6072), ('Coffee', 6072), ('level', 6071), ('confused', 6071), ('of', 6070),
('ANZ', 6070), ('men', 6069), ('Pilgor', 6069), ("'s", 6068), ('enthusiasts', 6068), ('English', 6
067), ('Pre-order', 6067), ('football', 6066), ('toy', 6066), ('league', 6065), ('mouth', 6065),
('system', 6064), ('cuddle', 6064), ('.', 6063), ('news', 6063), ('Contested', 6062), ('pristine',
6062), ('by', 6061), ('Exciting', 6061), ('20', 6060), ('attic', 6060), ('clubs', 6059), ('someda
y', 6059), ('it', 6058), ('Warning', 6058), ('operates', 6057), ('goat', 6057), ('on', 6056), ('.S
orry', 6056), ('a', 6055), ('absurd', 6055), ('promotion', 6054), ('clarity', 6054), ('and', 605
3), ('screenshots', 6053), ('relegation', 6052), ('forgive', 6052), ('with', 6051), ('<', 6051),
('EFL', 6050), ('3Aggressively', 6050), ('Seasons', 6049), ('assert', 6049), ('typically', 6048),
('Creators', 6048), ('run', 6047), ('occasion.When', 6047), ('from', 6046), ('Introducing', 6046),
('August', 6045), ('Simulator', 6045), ('to', 6044), ('Backbone', 6044), ('May', 6043), ('license
d', 6043), ('each', 6042), ('controller', 6042), ('team', 6041), ('Inspired', 6041), ('playing', 6
040), ('Goat', 6040), ('38', 6039), ('neither', 6039), ('matches', 6038), ('DualSense', 6038), ('a
ll', 6037), ('in-game.Me', 6037)]
Accuracy: 0.98
Precision macro: 0.98
Recall macro: 0.98
No.of terms after rfe: 2874
```

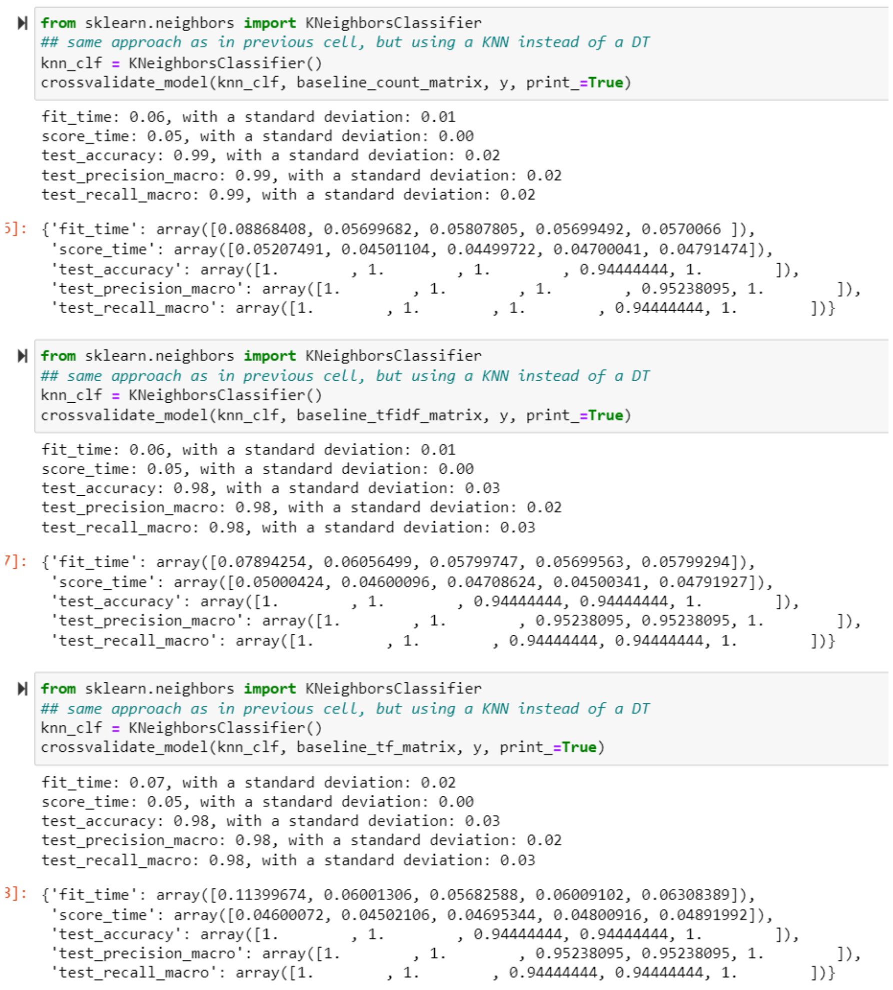
Build Classification Models

Choose another suitable algorithm

The other suitable algorithm to use for baseline matrices is KNeighborsClassifier we use this algorithm to output 3 matrices. I choose KNeighbors algorithm for the baseline modelling mainly cause it could also give good results.

The algorithm I choose at the start was the decision tree. I will compare the both of the algorithms.

The count matrices accuracy for KNeighbours, precision and recall has percentage of 99% and same goes with the decision tree. With the Kneighbours having 98% with the tfidf and the tf that is exactly same percentage as the decision tree. By looking at the score time and the standard deviation seems like the KNeighbours and the Decision tree algorithms are exactly the same with the three matrices.



```
from sklearn.neighbors import KNeighborsClassifier
## same approach as in previous cell, but using a KNN instead of a DT
knn_clf = KNeighborsClassifier()
crossvalidate_model(knn_clf, baseline_count_matrix, y, print_=True)
```

```
fit_time: 0.06, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.99, with a standard deviation: 0.02
test_precision_macro: 0.99, with a standard deviation: 0.02
test_recall_macro: 0.99, with a standard deviation: 0.02

5]: {'fit_time': array([0.08868408, 0.05699682, 0.05807805, 0.05699492, 0.0570066 ]),
     'score_time': array([0.05207491, 0.04501104, 0.04499722, 0.04700041, 0.04791474]),
     'test_accuracy': array([1.        , 1.        , 1.        , 0.94444444, 1.        ]),
     'test_precision_macro': array([1.        , 1.        , 1.        , 0.95238095, 1.        ]),
     'test_recall_macro': array([1.        , 1.        , 1.        , 0.94444444, 1.        ])}
```

```
from sklearn.neighbors import KNeighborsClassifier
## same approach as in previous cell, but using a KNN instead of a DT
knn_clf = KNeighborsClassifier()
crossvalidate_model(knn_clf, baseline_tfidf_matrix, y, print_=True)
```

```
fit_time: 0.06, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.98, with a standard deviation: 0.03
test_precision_macro: 0.98, with a standard deviation: 0.02
test_recall_macro: 0.98, with a standard deviation: 0.03

7]: {'fit_time': array([0.07894254, 0.06056499, 0.05799747, 0.05699563, 0.05799294]),
     'score_time': array([0.05000424, 0.04600096, 0.04708624, 0.04500341, 0.04791927]),
     'test_accuracy': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ]),
     'test_precision_macro': array([1.        , 1.        , 0.95238095, 0.95238095, 1.        ]),
     'test_recall_macro': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ])}
```

```
from sklearn.neighbors import KNeighborsClassifier
## same approach as in previous cell, but using a KNN instead of a DT
knn_clf = KNeighborsClassifier()
crossvalidate_model(knn_clf, baseline_tf_matrix, y, print_=True)
```

```
fit_time: 0.07, with a standard deviation: 0.02
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.98, with a standard deviation: 0.03
test_precision_macro: 0.98, with a standard deviation: 0.02
test_recall_macro: 0.98, with a standard deviation: 0.03

3]: {'fit_time': array([0.11399674, 0.06001306, 0.05682588, 0.06009102, 0.06308389]),
     'score_time': array([0.04600072, 0.04502106, 0.04695344, 0.04800916, 0.04891992]),
     'test_accuracy': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ]),
     'test_precision_macro': array([1.        , 1.        , 0.95238095, 0.95238095, 1.        ]),
     'test_recall_macro': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ])}
```

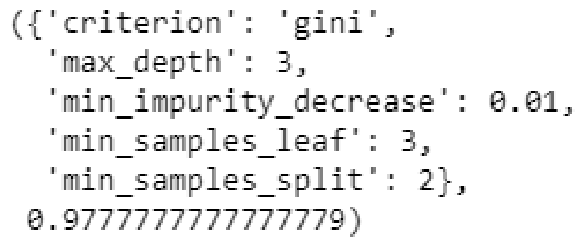
Parameter tuning

The algorithm that I picked for the parameter turning is DT , tunes parameters with criterion max_depth , min_samples_split, example:

- min_impurity_decrease is value of threshold with split internal nod.
- max_depth is the maximum number of nodes which grow decision till its cut off.
- min_samples_split is minimum number of rows split internal node.
- min_samples_leaf the minimum numbers, requires to be present in the leaf node.

The hyperparameter tuning will have criterion shown that is choose an gini option. The max_depth is 3. The min_samples_leaf is 3 meaning minimum samples need to present in the leaf, has 3% in the data min. the min_samples_split is 2 mean number rows to split in the internal node, The min_impurity_decrease is 0.01 in the threshold.

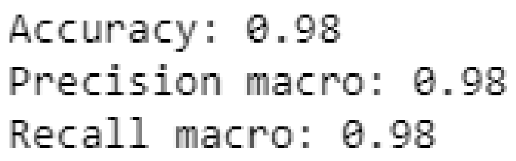
Hyperparameter:



```
({'criterion': 'gini',
  'max_depth': 3,
  'min_impurity_decrease': 0.01,
  'min_samples_leaf': 3,
  'min_samples_split': 2},
 0.9777777777777779)
```

The Optimal parameters stat is random for all the parameters from hyperparameter, then we can finally find the performance. The Accuracy wend up being very high with 0.98. the precision is 0.98 and recall also 0.98. therefore the are all 0.98

Opt:



```
Accuracy: 0.98
Precision macro: 0.98
Recall macro: 0.98
```

Overall evaluation

a) **Which vectorization technique produced best results?**
The count technique has performed gave the best result with 98%, compared to tf matrix and tfidf matrix. The Tf matrix accuracy is 0.97%, precision and tfidf is 97% aswell.
Best result = count technique.

b) **Which preparation techniques produced best results?**
The initial cleaning had a 97% performed well but the stop removal performed the best with 98% and the lowest was the bags of words with 94%.
Best result = Stop Removal.

c) **Overall, which classifier produced the best results**
The best result for the classifier is the count matrix performed 99% and the tfidf and tf had the same again which was 98%. The count test_accuracy: 0.99, test_precision_macro: 0.99 and test_recall_macro: 0.99 all 0.99, with tfidf and tf have around 0.98 or under.
Best Results=count matrix

d) **Overall, which model (pipeline) produced best result (include vectorisation, preparation, and the classifier and its optimal parameters in your answer)**

The model with best results vectorisation is the count with 97%.

The best result of preparation for the pipeline is the stop removal 98%.

The optimal parameters best results was the tfidf with 98%.

e) **What features/terms were the most predictive?**
The most predictive features pred terms are league, play and game.

f) **Do the classification results concur the results of the clustering and to what extent?**

The performance of the classification with test accuracy of 0.89%, precision of 0.89% and recall of 0.89%

g) **Discuss the limitations of your project and provide recommendations for future improvements**

The limitation of the project was that there was 30 urls links for each of the three class documents, articles, websites etc but overall was 90 urls. But overall can have better links as improvements, urls were sometimes not working therefore had to get a new url link related to premier league, xbox or playstation.

Reference

[1]. *MOODLE - TU Dublin Blanchardstown Campus: Log in to the site*.

[online]: <https://vle-bn.tudublin.ie/pluginfile.php/258769/mod_resource/content/7/Lecture3.pdf>

[2]. *MOODLE - TU Dublin Blanchardstown Campus: Log in to the site*.

[online]: <https://vle-bn.tudublin.ie/pluginfile.php/402684/mod_resource/content/1/lab4%20Indicative%20Solutions.pdf>

[3]. *MOODLE - TU Dublin Blanchardstown Campus: Log in to the site*.

[online]: <https://vle-bn.tudublin.ie/pluginfile.php/258769/mod_resource/content/7/Lecture4.pdf>

[4]. *MOODLE - TU Dublin Blanchardstown Campus: Log in to the site*.

[online]: <https://vle-bn.tudublin.ie/pluginfile.php/402684/mod_resource/content/1/lab5%20Indicative%20Solutions.pdf>

[5]. *MOODLE - TU Dublin Blanchardstown Campus: Log in to the site*.

[online]: <https://vle-bn.tudublin.ie/pluginfile.php/258769/mod_resource/content/7/Lecture6.pdf>

[6]. *MOODLE - TU Dublin Blanchardstown Campus: Log in to the site*.

[online]: <https://vle-bn.tudublin.ie/pluginfile.php/402684/mod_resource/content/1/lab6%20Indicative%20Solutions.pdf>

[7]. *MOODLE - TU Dublin Blanchardstown Campus: Log in to the site*.

[online] :<https://vle-bn.tudublin.ie/pluginfile.php/258769/mod_resource/content/7/Lecture8.pdf>

[8]. *MOODLE - TU Dublin Blanchardstown Campus: Log in to the site*.

[online]: <https://vle-bn.tudublin.ie/pluginfile.php/402684/mod_resource/content/1/lab8%20Indicative%20Solutions.pdf>

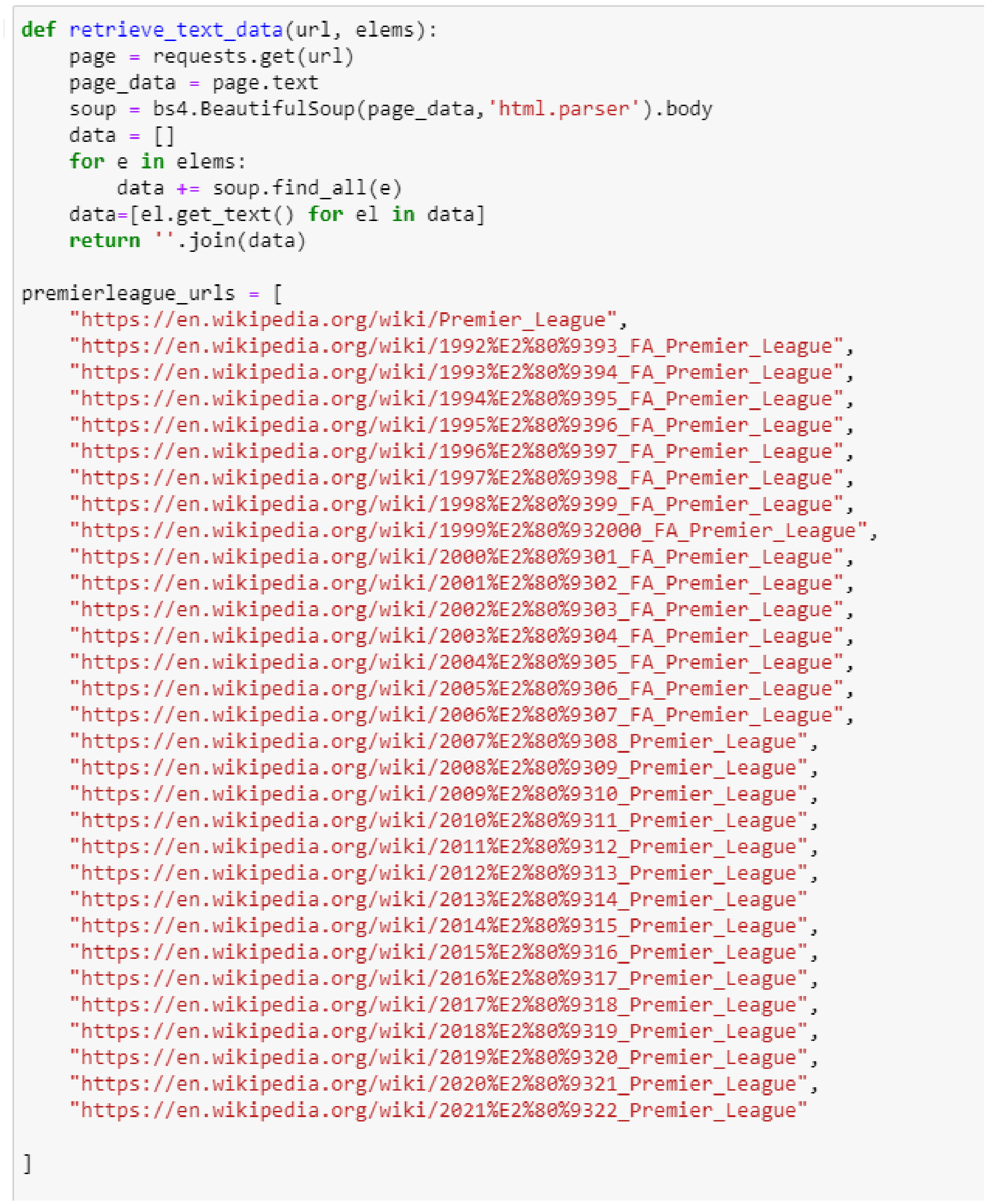
[9]. *MOODLE - TU Dublin Blanchardstown Campus: Log in to the site*.

[online]: <https://vle-bn.tudublin.ie/pluginfile.php/402684/mod_resource/content/1/lab9%20Indicative%20Solutions.pdf>

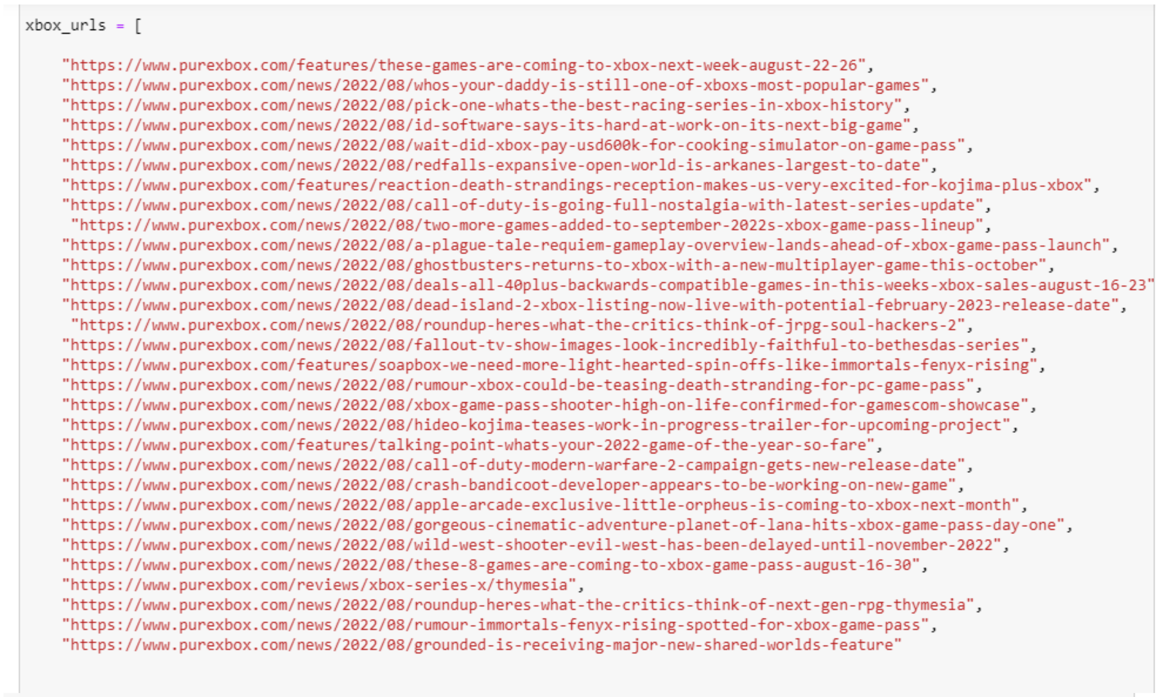
[10]. wordclouds.com.

[online]: <https://www.wordclouds.com/>

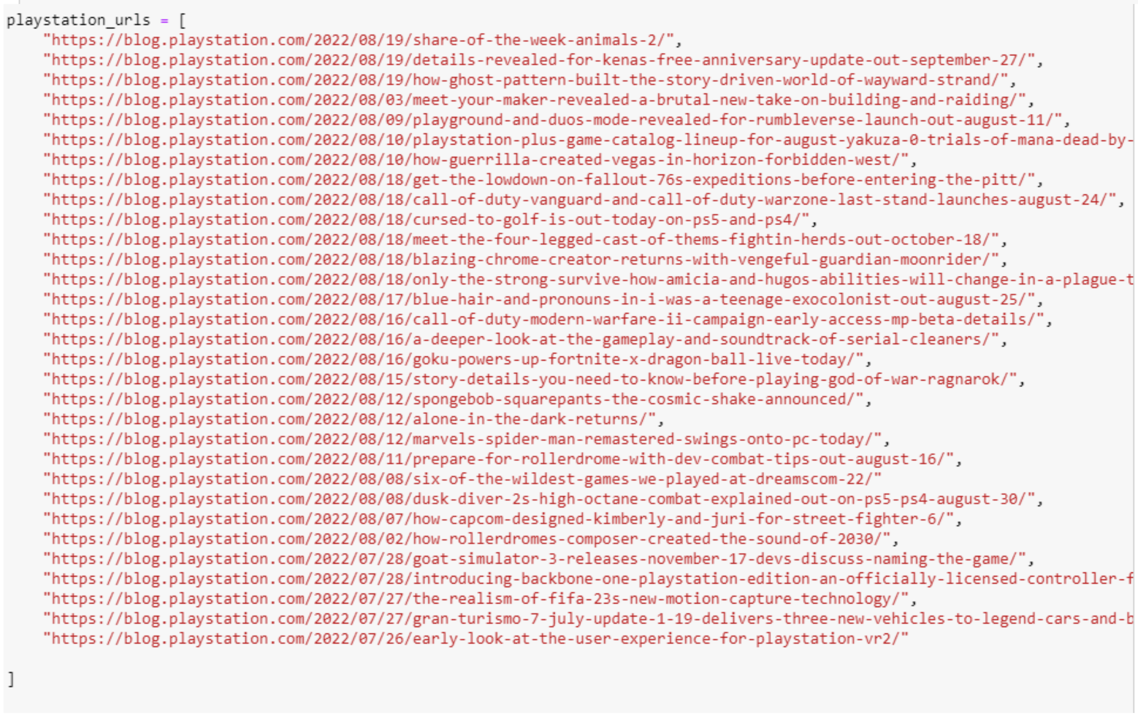
Appendix



```
def retrieve_text_data(url, elems):
    page = requests.get(url)
    page_data = page.text
    soup = bs4.BeautifulSoup(page_data,'html.parser').body
    data = []
    for e in elems:
        data += soup.find_all(e)
    data=[el.get_text() for el in data]
    return ''.join(data)

premierleague_urls = [
    "https://en.wikipedia.org/wiki/Premier_League",
    "https://en.wikipedia.org/wiki/1992%E2%80%9393_FA_Premier_League",
    "https://en.wikipedia.org/wiki/1993%E2%80%9394_FA_Premier_League",
    "https://en.wikipedia.org/wiki/1994%E2%80%9395_FA_Premier_League",
    "https://en.wikipedia.org/wiki/1995%E2%80%9396_FA_Premier_League",
    "https://en.wikipedia.org/wiki/1996%E2%80%9397_FA_Premier_League",
    "https://en.wikipedia.org/wiki/1997%E2%80%9398_FA_Premier_League",
    "https://en.wikipedia.org/wiki/1998%E2%80%9399_FA_Premier_League",
    "https://en.wikipedia.org/wiki/1999%E2%80%932000_FA_Premier_League",
    "https://en.wikipedia.org/wiki/2000%E2%80%9301_FA_Premier_League",
    "https://en.wikipedia.org/wiki/2001%E2%80%9302_FA_Premier_League",
    "https://en.wikipedia.org/wiki/2002%E2%80%9303_FA_Premier_League",
    "https://en.wikipedia.org/wiki/2003%E2%80%9304_FA_Premier_League",
    "https://en.wikipedia.org/wiki/2004%E2%80%9305_FA_Premier_League",
    "https://en.wikipedia.org/wiki/2005%E2%80%9306_FA_Premier_League",
    "https://en.wikipedia.org/wiki/2006%E2%80%9307_FA_Premier_League",
    "https://en.wikipedia.org/wiki/2007%E2%80%9308_Premier_League",
    "https://en.wikipedia.org/wiki/2008%E2%80%9309_Premier_League",
    "https://en.wikipedia.org/wiki/2009%E2%80%9310_Premier_League",
    "https://en.wikipedia.org/wiki/2010%E2%80%9311_Premier_League",
    "https://en.wikipedia.org/wiki/2011%E2%80%9312_Premier_League",
    "https://en.wikipedia.org/wiki/2012%E2%80%9313_Premier_League",
    "https://en.wikipedia.org/wiki/2013%E2%80%9314_Premier_League"
    "https://en.wikipedia.org/wiki/2014%E2%80%9315_Premier_League",
    "https://en.wikipedia.org/wiki/2015%E2%80%9316_Premier_League",
    "https://en.wikipedia.org/wiki/2016%E2%80%9317_Premier_League",
    "https://en.wikipedia.org/wiki/2017%E2%80%9318_Premier_League",
    "https://en.wikipedia.org/wiki/2018%E2%80%9319_Premier_League",
    "https://en.wikipedia.org/wiki/2019%E2%80%9320_Premier_League",
    "https://en.wikipedia.org/wiki/2020%E2%80%9321_Premier_League",
    "https://en.wikipedia.org/wiki/2021%E2%80%9322_Premier_League"

]
```



```
xbox_urls = [

    "https://www.purexbox.com/features/these-games-are-coming-to-xbox-next-week-august-22-26",
    "https://www.purexbox.com/news/2022/08/whos-your-daddy-is-still-one-of-xboxs-most-popular-games",
    "https://www.purexbox.com/news/2022/08/pick-one-whats-the-best-racing-series-in-xbox-history",
    "https://www.purexbox.com/news/2022/08/id-software-says-its-hard-at-work-on-its-next-big-game",
    "https://www.purexbox.com/news/2022/08/wait-did-xbox-pay-usd600k-for-cooking-simulator-on-game-pass",
    "https://www.purexbox.com/news/2022/08/redfalls-expansive-open-world-is-arkanes-largest-to-date",
    "https://www.purexbox.com/features/reaction-death-strandings-reception-makes-us-very-excited-for-kojima-plus-xbox",
    "https://www.purexbox.com/news/2022/08/call-of-duty-is-going-full-nostalgia-with-latest-series-update",
     "https://www.purexbox.com/news/2022/08/two-more-games-added-to-september-2022s-xbox-game-pass-lineup",
    "https://www.purexbox.com/news/2022/08/a-plague-tale-requiem-gameplay-overview-lands-ahead-of-xbox-game-pass-launch",
    "https://www.purexbox.com/news/2022/08/ghostbusters-returns-to-xbox-with-a-new-multiplayer-game-this-october",
    "https://www.purexbox.com/news/2022/08/deals-all-40plus-backwards-compatible-games-in-this-weeks-xbox-sales-august-16-23"
    "https://www.purexbox.com/news/2022/08/dead-island-2-xbox-listing-now-live-with-potential-february-2023-release-date",
     "https://www.purexbox.com/news/2022/08/roundup-heres-what-the-critics-think-of-jrpg-soul-hackers-2",
    "https://www.purexbox.com/news/2022/08/fallout-tv-show-images-look-incredibly-faithful-to-bethesdas-series",
    "https://www.purexbox.com/features/soapbox-we-need-more-light-hearted-spin-offs-like-immortals-fenyx-rising",
    "https://www.purexbox.com/news/2022/08/rumour-xbox-could-be-teasing-death-stranding-for-pc-game-pass",
    "https://www.purexbox.com/news/2022/08/xbox-game-pass-shooter-high-on-life-confirmed-for-gamescom-showcase",
    "https://www.purexbox.com/news/2022/08/hideo-kojima-teases-work-in-progress-trailer-for-upcoming-project",
    "https://www.purexbox.com/features/talking-point-whats-your-2022-game-of-the-year-so-fare",
    "https://www.purexbox.com/news/2022/08/call-of-duty-modern-warfare-2-campaign-gets-new-release-date",
    "https://www.purexbox.com/news/2022/08/crash-bandicoot-developer-appears-to-be-working-on-new-game",
    "https://www.purexbox.com/news/2022/08/apple-arcade-exclusive-little-orpheus-is-coming-to-xbox-next-month",
    "https://www.purexbox.com/news/2022/08/gorgeous-cinematic-adventure-planet-of-lana-hits-xbox-game-pass-day-one",
    "https://www.purexbox.com/news/2022/08/wild-west-shooter-evil-west-has-been-delayed-until-november-2022",
    "https://www.purexbox.com/news/2022/08/these-8-games-are-coming-to-xbox-game-pass-august-16-30",
    "https://www.purexbox.com/reviews/xbox-series-x/thymesia",
    "https://www.purexbox.com/news/2022/08/roundup-heres-what-the-critics-think-of-next-gen-rpg-thymesia",
    "https://www.purexbox.com/news/2022/08/rumour-immortals-fenyx-rising-spotted-for-xbox-game-pass",
    "https://www.purexbox.com/news/2022/08/grounded-is-receiving-major-new-shared-worlds-feature"
```



```
playstation_urls = [
    "https://blog.playstation.com/2022/08/19/share-of-the-week-animals-2/",
    "https://blog.playstation.com/2022/08/19/details-revealed-for-kenas-free-anniversary-update-out-september-27/",
    "https://blog.playstation.com/2022/08/19/how-ghost-pattern-built-the-story-driven-world-of-wayward-strand/",
    "https://blog.playstation.com/2022/08/03/meet-your-maker-revealed-a-brutal-new-take-on-building-and-raiding/",
    "https://blog.playstation.com/2022/08/09/playground-and-duos-mode-revealed-for-rumbleverse-launch-out-august-11/",
    "https://blog.playstation.com/2022/08/10/playstation-plus-game-catalog-lineup-for-august-yakuza-0-trials-of-mana-dead-by-
    "https://blog.playstation.com/2022/08/10/how-guerrilla-created-vegas-in-horizon-forbidden-west/",
    "https://blog.playstation.com/2022/08/18/get-the-lowdown-on-fallout-76s-expeditions-before-entering-the-pitt/",
    "https://blog.playstation.com/2022/08/18/call-of-duty-vanguard-and-call-of-duty-warzone-last-stand-launches-august-24/",
    "https://blog.playstation.com/2022/08/18/cursed-to-golf-is-out-today-on-ps5-and-ps4/",
    "https://blog.playstation.com/2022/08/18/meet-the-four-legged-cast-of-thems-fightin-herds-out-october-18/",
    "https://blog.playstation.com/2022/08/18/blazing-chrome-creator-returns-with-vengeful-guardian-moonrider/",
    "https://blog.playstation.com/2022/08/18/only-the-strong-survive-how-amicia-and-hugos-abilities-will-change-in-a-plague-t
    "https://blog.playstation.com/2022/08/17/blue-hair-and-pronouns-in-i-was-a-teenage-exocolonist-out-august-25/",
    "https://blog.playstation.com/2022/08/16/call-of-duty-modern-warfare-ii-campaign-early-access-mp-beta-details/",
    "https://blog.playstation.com/2022/08/16/a-deeper-look-at-the-gameplay-and-soundtrack-of-serial-cleaners/",
    "https://blog.playstation.com/2022/08/16/goku-powers-up-fortnite-x-dragon-ball-live-today/",
    "https://blog.playstation.com/2022/08/15/story-details-you-need-to-know-before-playing-god-of-war-ragnarok/",
    "https://blog.playstation.com/2022/08/12/spongebob-squarepants-the-cosmic-shake-announced/",
    "https://blog.playstation.com/2022/08/12/alone-in-the-dark-returns/",
    "https://blog.playstation.com/2022/08/12/marvels-spider-man-remastered-swings-onto-pc-today/",
    "https://blog.playstation.com/2022/08/11/prepare-for-rollerdrome-with-dev-combat-tips-out-august-16/",
    "https://blog.playstation.com/2022/08/08/six-of-the-wildest-games-we-played-at-dreamscom-22/"
    "https://blog.playstation.com/2022/08/08/dusk-diver-2s-high-octane-combat-explained-out-on-ps5-ps4-august-30/",
    "https://blog.playstation.com/2022/08/07/how-capcom-designed-kimberly-and-juri-for-street-fighter-6/",
    "https://blog.playstation.com/2022/08/02/how-rollerdromes-composer-created-the-sound-of-2030/",
    "https://blog.playstation.com/2022/07/28/goat-simulator-3-releases-november-17-devs-discuss-naming-the-game/",
    "https://blog.playstation.com/2022/07/28/introducing-backbone-one-playstation-edition-an-officially-licensed-controller-f
    "https://blog.playstation.com/2022/07/27/the-realism-of-fifa-23s-new-motion-capture-technology/",
    "https://blog.playstation.com/2022/07/27/gran-turismo-7-july-update-1-19-delivers-three-new-vehicles-to-legend-cars-and-b
    "https://blog.playstation.com/2022/07/26/early-look-at-the-user-experience-for-playstation-vr2/"

]
```

```
playstation_docs = [retrieve_text_data(url, ['h1','p']) for url
            in playstation_urls]
playstation_docs
print(len(playstation_docs))
```

```
30
```

```
premierleague_docs = [retrieve_text_data(url, ['h1','p']) for url
            in premierleague_urls]
premierleague_docs
print(len(premierleague_docs))
```

```
30
```

```
xbox_docs = [retrieve_text_data(url, ['h1','p']) for url
            in xbox_urls]
xbox_docs
print(len(xbox_docs))
```

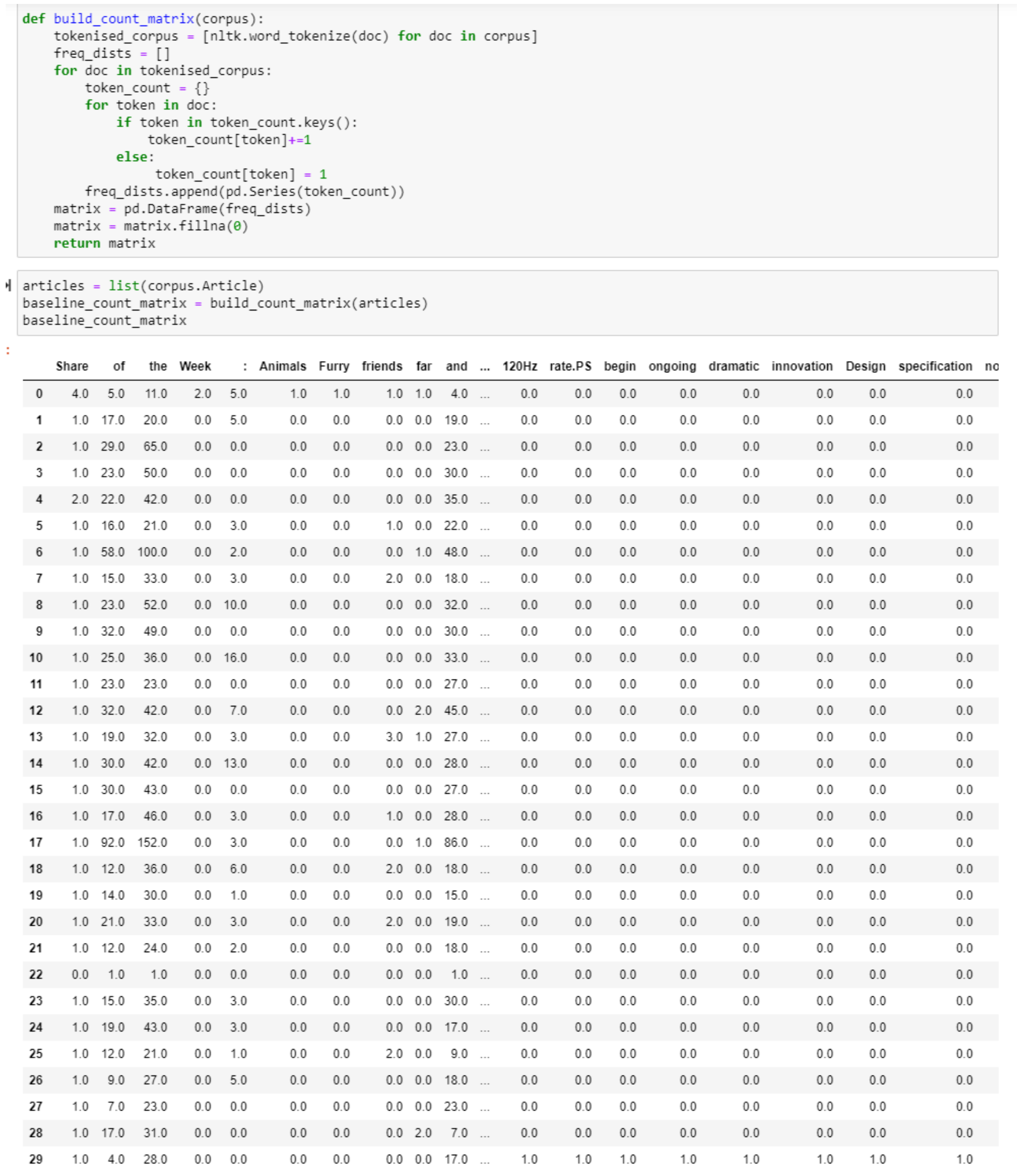
```
30
```

```
all_docs = premierleague_docs + playstation_docs + xbox_docs
all_labels = (['premierleague'] * len(premierleague_docs) +
              ['playstation'] * len(playstation_docs) +
              ['xbox'] * len(xbox_docs))
all_docs
```

```
£1,941,609 = £9,708,045 merit payment).[7]\nSince its split with the Football League, established clubs in the Premier L
eague have a funding disparity from counterparts in lower leagues. Revenue from television rights between the leagues ha
s played a part in this.[134]\nPromoted teams have found it difficult to avoid relegation in their first Premier League
season. One Premier League newcomer has been relegated back to the Football League every season, save the 2001-02, 2011-
12 and 2017-18 seasons. In the 1997-98 season, all three promoted clubs were relegated by the season\'s end.[135]\nThe P
remier League distributes a portion of its television revenue as "parachute payments" to relegated clubs for adjustment
to television revenue loss. The average Premier League team receives £41\xa0million[136] while the average Championship
club receives £2\xa0million.[137] Starting with the 2013-14 season, these payments are in excess of £60\xa0million over
four seasons.[138] Critics maintain that the payments widen the gap between teams that have reached the Premier League a
nd those that have not,[139] leading to the common occurrence of teams "bouncing back" soon after their relegation.\nClu
bs which have failed to win immediate promotion back to the Premier League have seen financial problems, in some cases a
dministration or liquidation. Further relegations down the footballing ladder have occurred for multiple clubs unable to
cope with the gap.[140][141]\nTelevision has played a major role in the history of the Premier League. The League\'s dec
ision to assign broadcasting rights to BSkyB in 1992 was at the time a radical decision, but one that has paid off. At t
he time pay television was an almost untested proposition in the UK market, as was charging fans to watch live televised
football. However, a combination of Sky\'s strategy, the quality of Premier League football and the public\'s appetite f
or the game has seen the value of the Premier League\'s TV rights soar.[23]\nThe Premier League sells its television rig
hts on a collective basis. This is in contrast to some other European leagues, including La Liga, in which each club sel
ls its rights individually, leading to a much higher share of the total income going to the top few clubs.[142] The mone
y is divided into three parts:[143] half is divided equally between the clubs; one quarter is awarded on a merit basis b
```

```
def build_corpus(docs,labels):
    corpus = np.array(docs)
    corpus = pd.DataFrame({'Article': corpus, 'Class': labels})
    corpus = corpus.sample(len(corpus))
    return corpus

corpus = build_corpus(all_docs, all_labels)
corpus
```

]:

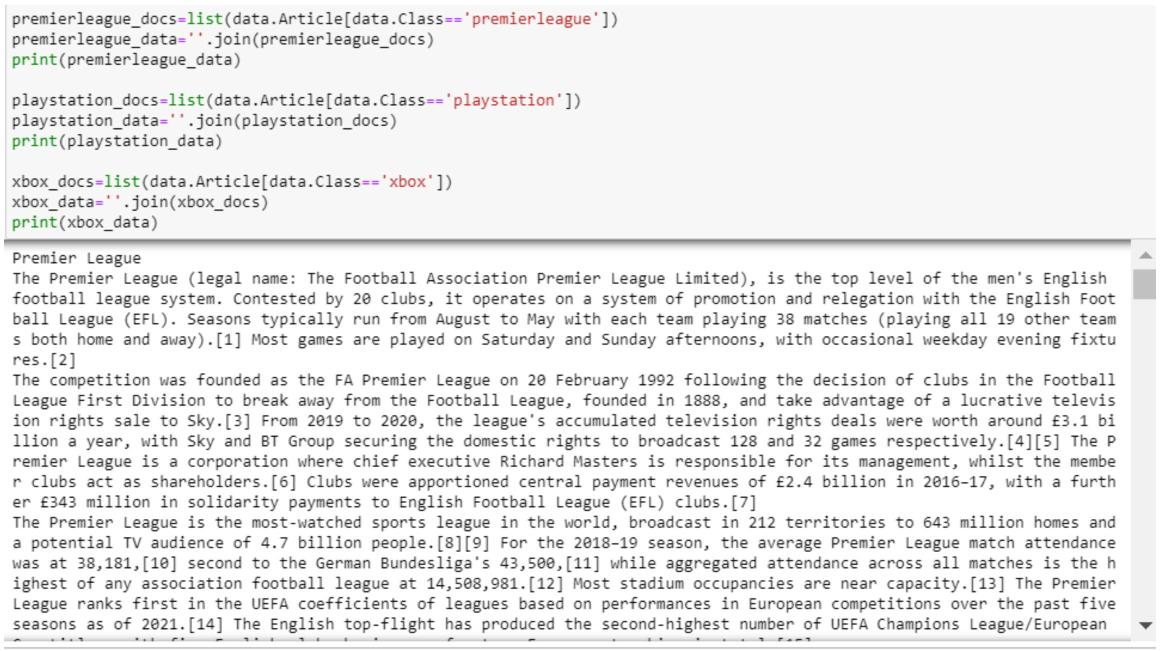


```python
def build_count_matrix(corpus):
    tokenised_corpus = [nltk.word_tokenize(doc) for doc in corpus]
    freq_dists = []
    for doc in tokenised_corpus:
        token_count = {}
        for token in doc:
            if token in token_count.keys():
                token_count[token]+=1
            else:
                 token_count[token] = 1
        freq_dists.append(pd.Series(token_count))
    matrix = pd.DataFrame(freq_dists)
    matrix = matrix.fillna(0)
    return matrix
```

```python
articles = list(corpus.Article)
baseline_count_matrix = build_count_matrix(articles)
baseline_count_matrix
```

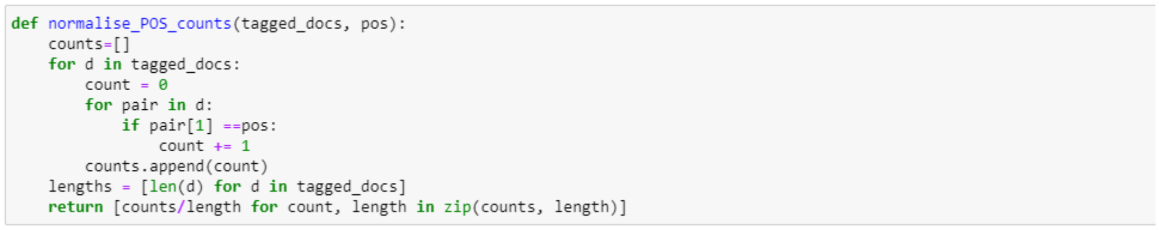
:

	Share	of	the	Week	:	Animals	Furry	friends	far	and	...	120Hz	rate.PS	begin	ongoing	dramatic	innovation	Design	specification	no
0	4.0	5.0	11.0	2.0	5.0	1.0	1.0	1.0	1.0	4.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	17.0	20.0	0.0	5.0	0.0	0.0	0.0	0.0	19.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	1.0	29.0	65.0	0.0	0.0	0.0	0.0	0.0	0.0	23.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	1.0	23.0	50.0	0.0	0.0	0.0	0.0	0.0	0.0	30.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	2.0	22.0	42.0	0.0	0.0	0.0	0.0	0.0	0.0	35.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
5	1.0	16.0	21.0	0.0	3.0	0.0	0.0	1.0	0.0	22.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
6	1.0	58.0	100.0	0.0	2.0	0.0	0.0	0.0	1.0	48.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7	1.0	15.0	33.0	0.0	3.0	0.0	0.0	2.0	0.0	18.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
8	1.0	23.0	52.0	0.0	10.0	0.0	0.0	0.0	0.0	32.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
9	1.0	32.0	49.0	0.0	0.0	0.0	0.0	0.0	0.0	30.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
10	1.0	25.0	36.0	0.0	16.0	0.0	0.0	0.0	0.0	33.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
11	1.0	23.0	23.0	0.0	0.0	0.0	0.0	0.0	0.0	27.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
12	1.0	32.0	42.0	0.0	7.0	0.0	0.0	0.0	2.0	45.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
13	1.0	19.0	32.0	0.0	3.0	0.0	0.0	3.0	1.0	27.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
14	1.0	30.0	42.0	0.0	13.0	0.0	0.0	0.0	0.0	28.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
15	1.0	30.0	43.0	0.0	0.0	0.0	0.0	0.0	0.0	27.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16	1.0	17.0	46.0	0.0	3.0	0.0	0.0	1.0	0.0	28.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
17	1.0	92.0	152.0	0.0	3.0	0.0	0.0	0.0	1.0	86.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
18	1.0	12.0	36.0	0.0	6.0	0.0	0.0	2.0	0.0	18.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
19	1.0	14.0	30.0	0.0	1.0	0.0	0.0	0.0	0.0	15.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
20	1.0	21.0	33.0	0.0	3.0	0.0	0.0	2.0	0.0	19.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
21	1.0	12.0	24.0	0.0	2.0	0.0	0.0	0.0	0.0	18.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
22	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
23	1.0	15.0	35.0	0.0	3.0	0.0	0.0	0.0	0.0	30.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
24	1.0	19.0	43.0	0.0	3.0	0.0	0.0	0.0	0.0	17.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
25	1.0	12.0	21.0	0.0	1.0	0.0	0.0	2.0	0.0	9.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
26	1.0	9.0	27.0	0.0	5.0	0.0	0.0	0.0	0.0	18.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
27	1.0	7.0	23.0	0.0	0.0	0.0	0.0	0.0	0.0	23.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
28	1.0	17.0	31.0	0.0	0.0	0.0	0.0	0.0	2.0	7.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
29	1.0	4.0	28.0	0.0	0.0	0.0	0.0	0.0	0.0	17.0	...	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	

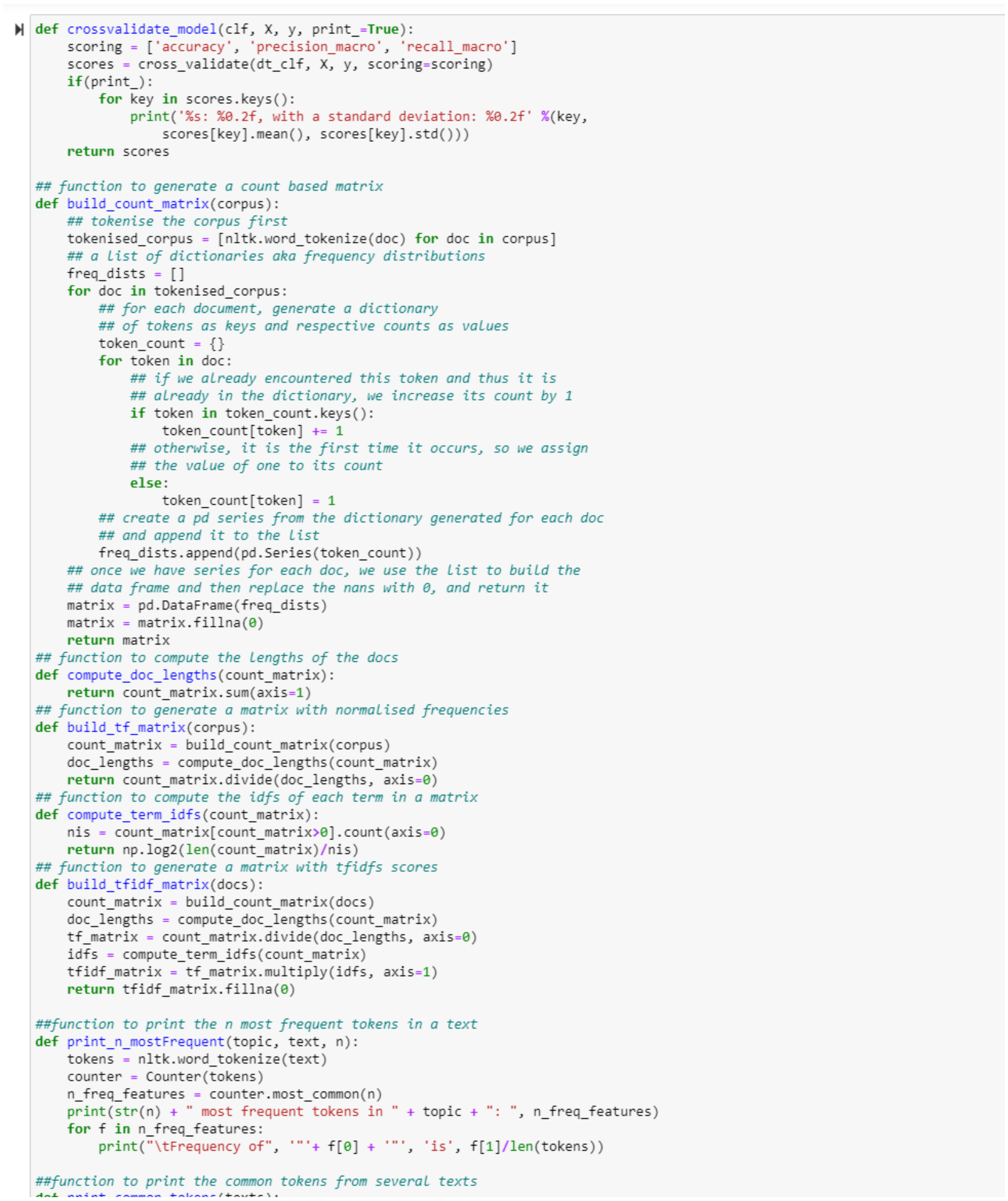


```python
premierleague_docs=list(data.Article[data.Class=='premierleague'])
premierleague_data=''.join(premierleague_docs)
print(premierleague_data)

playstation_docs=list(data.Article[data.Class=='playstation'])
playstation_data=''.join(playstation_docs)
print(playstation_data)

xbox_docs=list(data.Article[data.Class=='xbox'])
xbox_data=''.join(xbox_docs)
print(xbox_data)
```

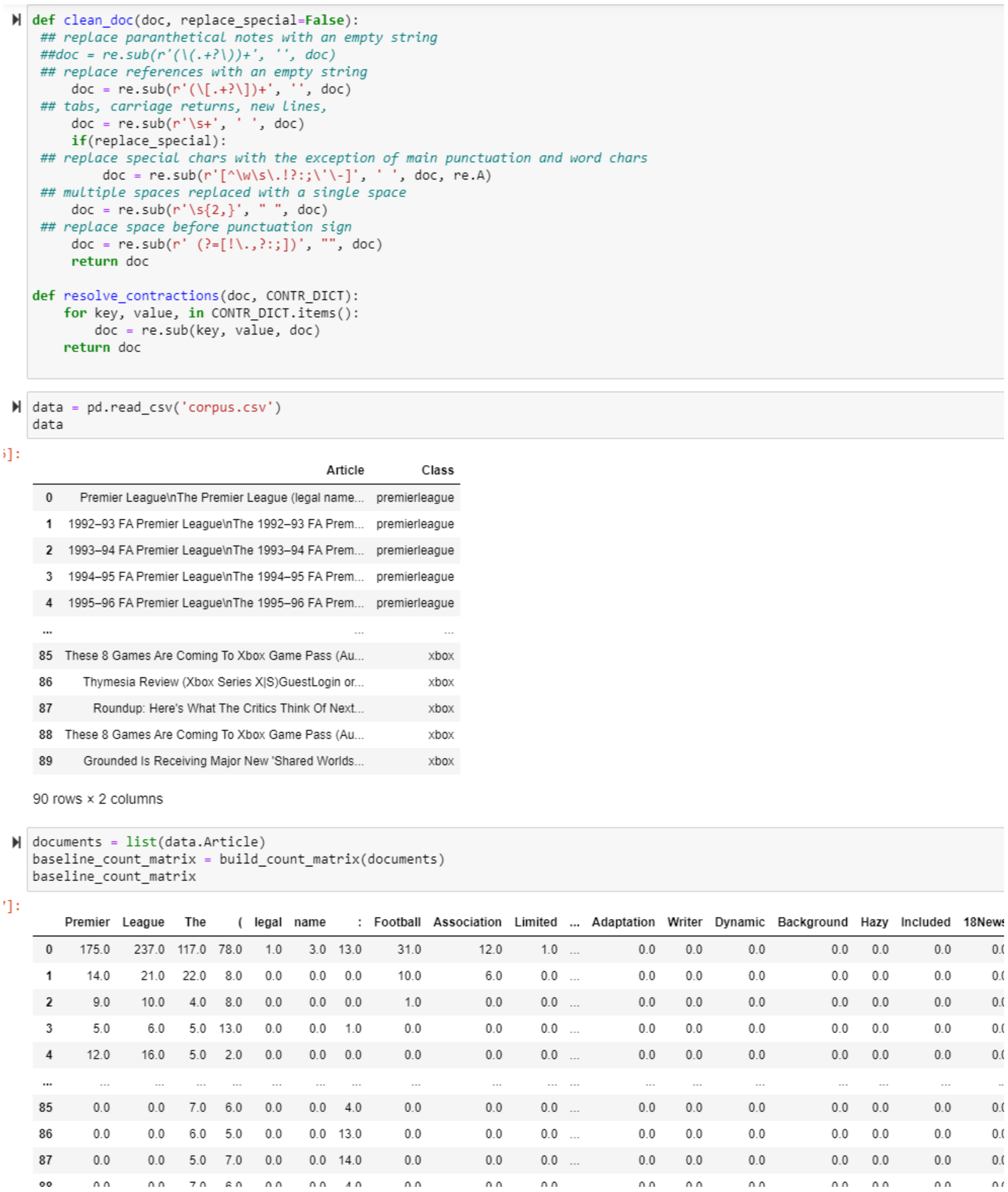
```
Premier League
The Premier League (legal name: The Football Association Premier League Limited), is the top level of the men's English
football league system. Contested by 20 clubs, it operates on a system of promotion and relegation with the English Foot
ball League (EFL). Seasons typically run from August to May with each team playing 38 matches (playing all 19 other team
s both home and away).[1] Most games are played on Saturday and Sunday afternoons, with occasional weekday evening fixtu
res.[2]
The competition was founded as the FA Premier League on 20 February 1992 following the decision of clubs in the Football
League First Division to break away from the Football League, founded in 1888, and take advantage of a lucrative televis
ion rights sale to Sky.[3] From 2019 to 2020, the league's accumulated television rights deals were worth around £3.1 bi
llion a year, with Sky and BT Group securing the domestic rights to broadcast 128 and 32 games respectively.[4][5] The P
remier League is a corporation where chief executive Richard Masters is responsible for its management, whilst the membe
r clubs act as shareholders.[6] Clubs were apportioned central payment revenues of £2.4 billion in 2016-17, with a furth
er £343 million in solidarity payments to English Football League (EFL) clubs.[7]
The Premier League is the most-watched sports league in the world, broadcast in 212 territories to 643 million homes and
a potential TV audience of 4.7 billion people.[8][9] For the 2018-19 season, the average Premier League match attendance
was at 38,181,[10] second to the German Bundesliga's 43,500,[11] while aggregated attendance across all matches is the h
ighest of any association football league at 14,508,981.[12] Most stadium occupancies are near capacity.[13] The Premier
League ranks first in the UEFA coefficients of leagues based on performances in European competitions over the past five
seasons as of 2021.[14] The English top-flight has produced the second-highest number of UEFA Champions League/European
```



```python
def normalise_POS_counts(tagged_docs, pos):
    counts=[]
    for d in tagged_docs:
        count = 0
        for pair in d:
            if pair[1] ==pos:
                count += 1
        counts.append(count)
    lengths = [len(d) for d in tagged_docs]
    return [counts/length for count, length in zip(counts, length)]
```



```python
def crossvalidate_model(clf, X, y, print_=True):
    scoring = ['accuracy', 'precision_macro', 'recall_macro']
    scores = cross_validate(dt_clf, X, y, scoring=scoring)
    if(print_):
        for key in scores.keys():
            print('%s: %0.2f, with a standard deviation: %0.2f' %(key,
                scores[key].mean(), scores[key].std()))
    return scores

## function to generate a count based matrix
def build_count_matrix(corpus):
    ## tokenise the corpus first
    tokenised_corpus = [nltk.word_tokenize(doc) for doc in corpus]
    ## a list of dictionaries aka frequency distributions
    freq_dists = []
    for doc in tokenised_corpus:
        ## for each document, generate a dictionary
        ## of tokens as keys and respective counts as values
        token_count = {}
        for token in doc:
            ## if we already encountered this token and thus it is
            ## already in the dictionary, we increase its count by 1
            if token in token_count.keys():
                token_count[token] += 1
            ## otherwise, it is the first time it occurs, so we assign
            ## the value of one to its count
            else:
                token_count[token] = 1
        ## create a pd series from the dictionary generated for each doc
        ## and append it to the list
        freq_dists.append(pd.Series(token_count))
    ## once we have series for each doc, we use the list to build the
    ## data frame and then replace the nans with 0, and return it
    matrix = pd.DataFrame(freq_dists)
    matrix = matrix.fillna(0)
    return matrix
## function to compute the lengths of the docs
def compute_doc_lengths(count_matrix):
    return count_matrix.sum(axis=1)
## function to generate a matrix with normalised frequencies
def build_tf_matrix(corpus):
    count_matrix = build_count_matrix(corpus)
    doc_lengths = compute_doc_lengths(count_matrix)
    return count_matrix.divide(doc_lengths, axis=0)
## function to compute the idfs of each term in a matrix
def compute_term_idfs(count_matrix):
    nis = count_matrix[count_matrix>0].count(axis=0)
    return np.log2(len(count_matrix)/nis)
## function to generate a matrix with tfidfs scores
def build_tfidf_matrix(docs):
    count_matrix = build_count_matrix(docs)
    doc_lengths = compute_doc_lengths(count_matrix)
    tf_matrix = count_matrix.divide(doc_lengths, axis=0)
    idfs = compute_term_idfs(count_matrix)
    tfidf_matrix = tf_matrix.multiply(idfs, axis=1)
    return tfidf_matrix.fillna(0)

##function to print the n most frequent tokens in a text
def print_n_mostFrequent(topic, text, n):
    tokens = nltk.word_tokenize(text)
    counter = Counter(tokens)
    n_freq_features = counter.most_common(n)
    print(str(n) + " most frequent tokens in " + topic + ": ", n_freq_features)
    for f in n_freq_features:
        print("\tFrequency of", '"'+ f[0] + '"', 'is', f[1]/len(tokens))

##function to print the common tokens from several texts
```



```
def clean_doc(doc, replace_special=False):
 ## replace paranthetical notes with an empty string
 ##doc = re.sub(r'(\(.+?\))+', '', doc)
 ## replace references with an empty string
     doc = re.sub(r'(\[.+?\])+', '', doc)
 ## tabs, carriage returns, new lines,
     doc = re.sub(r'\s+', ' ', doc)
     if(replace_special):
 ## replace special chars with the exception of main punctuation and word chars
         doc = re.sub(r'[^\w\s\.!?:;\'\-]', ' ', doc, re.A)
 ## multiple spaces replaced with a single space
     doc = re.sub(r'\s{2,}', " ", doc)
 ## replace space before punctuation sign
     doc = re.sub(r' (?=[!\.,?:;])', "", doc)
     return doc

def resolve_contractions(doc, CONTR_DICT):
    for key, value, in CONTR_DICT.items():
        doc = re.sub(key, value, doc)
    return doc
```

```
data = pd.read_csv('corpus.csv')
data
```

]:

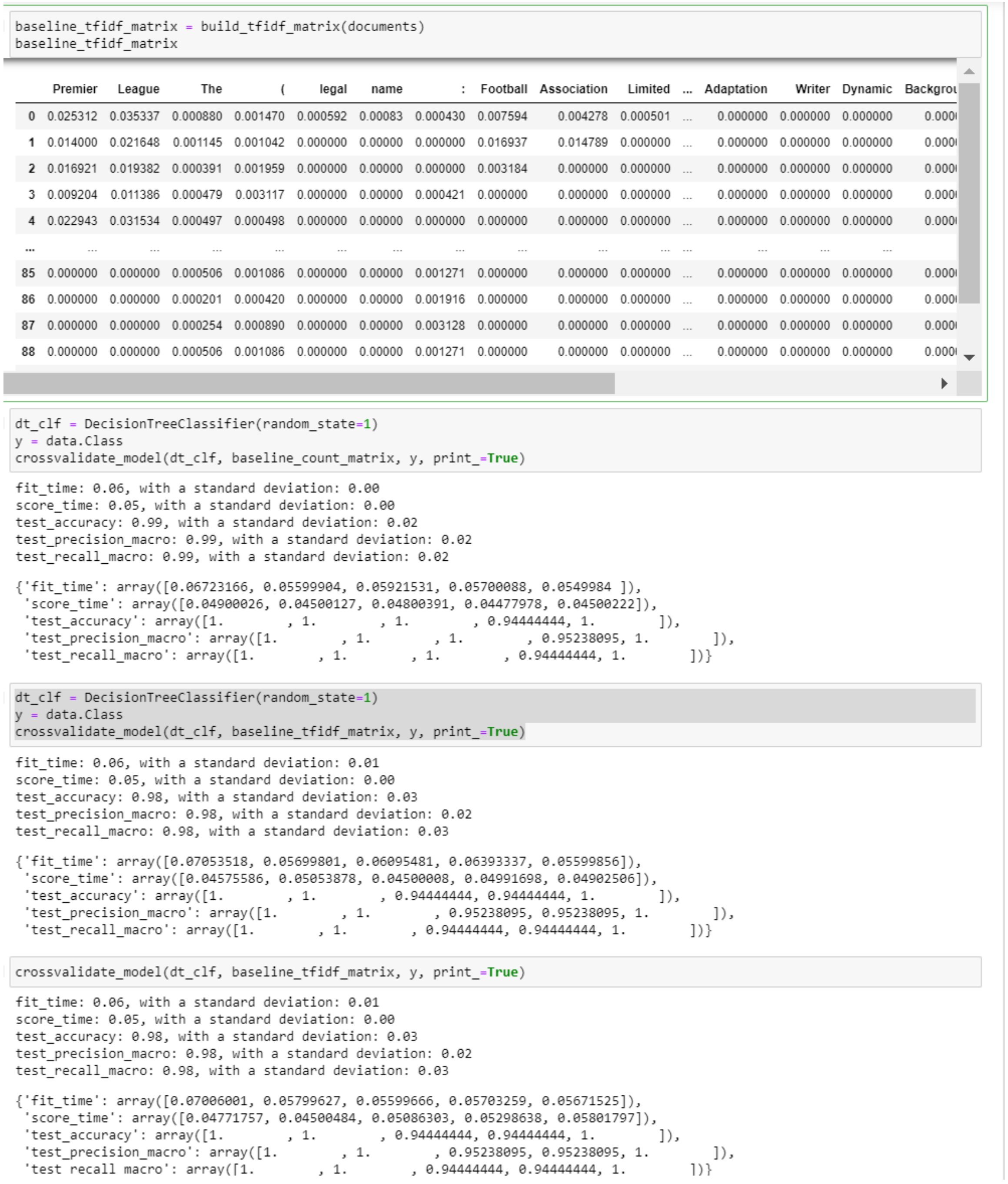
	Article	Class
0	Premier League\nThe Premier League (legal name...	premierleague
1	1992–93 FA Premier League\nThe 1992–93 FA Prem...	premierleague
2	1993–94 FA Premier League\nThe 1993–94 FA Prem...	premierleague
3	1994–95 FA Premier League\nThe 1994–95 FA Prem...	premierleague
4	1995–96 FA Premier League\nThe 1995–96 FA Prem...	premierleague
...	...	...
85	These 8 Games Are Coming To Xbox Game Pass (Au...	xbox
86	Thymesia Review (Xbox Series X|S)GuestLogin or...	xbox
87	Roundup: Here's What The Critics Think Of Next...	xbox
88	These 8 Games Are Coming To Xbox Game Pass (Au...	xbox
89	Grounded Is Receiving Major New 'Shared Worlds...	xbox

90 rows × 2 columns

```
documents = list(data.Article)
baseline_count_matrix = build_count_matrix(documents)
baseline_count_matrix
```

]:

	Premier	League	The	(	legal	name	:	Football	Association	Limited	...	Adaptation	Writer	Dynamic	Background	Hazy	Included	18New
0	175.0	237.0	117.0	78.0	1.0	3.0	13.0	31.0	12.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
1	14.0	21.0	22.0	8.0	0.0	0.0	0.0	10.0	6.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	9.0	10.0	4.0	8.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	5.0	6.0	5.0	13.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	12.0	16.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
85	0.0	0.0	7.0	6.0	0.0	0.0	4.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
86	0.0	0.0	6.0	5.0	0.0	0.0	13.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
87	0.0	0.0	5.0	7.0	0.0	0.0	14.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
88	0.0	0.0	7.0	6.0	0.0	0.0	4.0	0.0	0.0	0.0		0.0	0.0	0.0	0.0	0.0	0.0	0.



```
baseline_tfidf_matrix = build_tfidf_matrix(documents)
baseline_tfidf_matrix
```

	Premier	League	The	(	legal	name	:	Football	Association	Limited	...	Adaptation	Writer	Dynamic	Backgrou
0	0.025312	0.035337	0.000880	0.001470	0.000592	0.00083	0.000430	0.007594	0.004278	0.000501	...	0.000000	0.000000	0.000000	0.000
1	0.014000	0.021648	0.001145	0.001042	0.000000	0.00000	0.000000	0.016937	0.014789	0.000000	...	0.000000	0.000000	0.000000	0.000
2	0.016921	0.019382	0.000391	0.001959	0.000000	0.00000	0.000000	0.003184	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000
3	0.009204	0.011386	0.000479	0.003117	0.000000	0.00000	0.000421	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000
4	0.022943	0.031534	0.000497	0.000498	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
85	0.000000	0.000000	0.000506	0.001086	0.000000	0.00000	0.001271	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000
86	0.000000	0.000000	0.000201	0.000420	0.000000	0.00000	0.001916	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000
87	0.000000	0.000000	0.000254	0.000890	0.000000	0.00000	0.003128	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000
88	0.000000	0.000000	0.000506	0.001086	0.000000	0.00000	0.001271	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000

```
dt_clf = DecisionTreeClassifier(random_state=1)
y = data.Class
crossvalidate_model(dt_clf, baseline_count_matrix, y, print_=True)
```

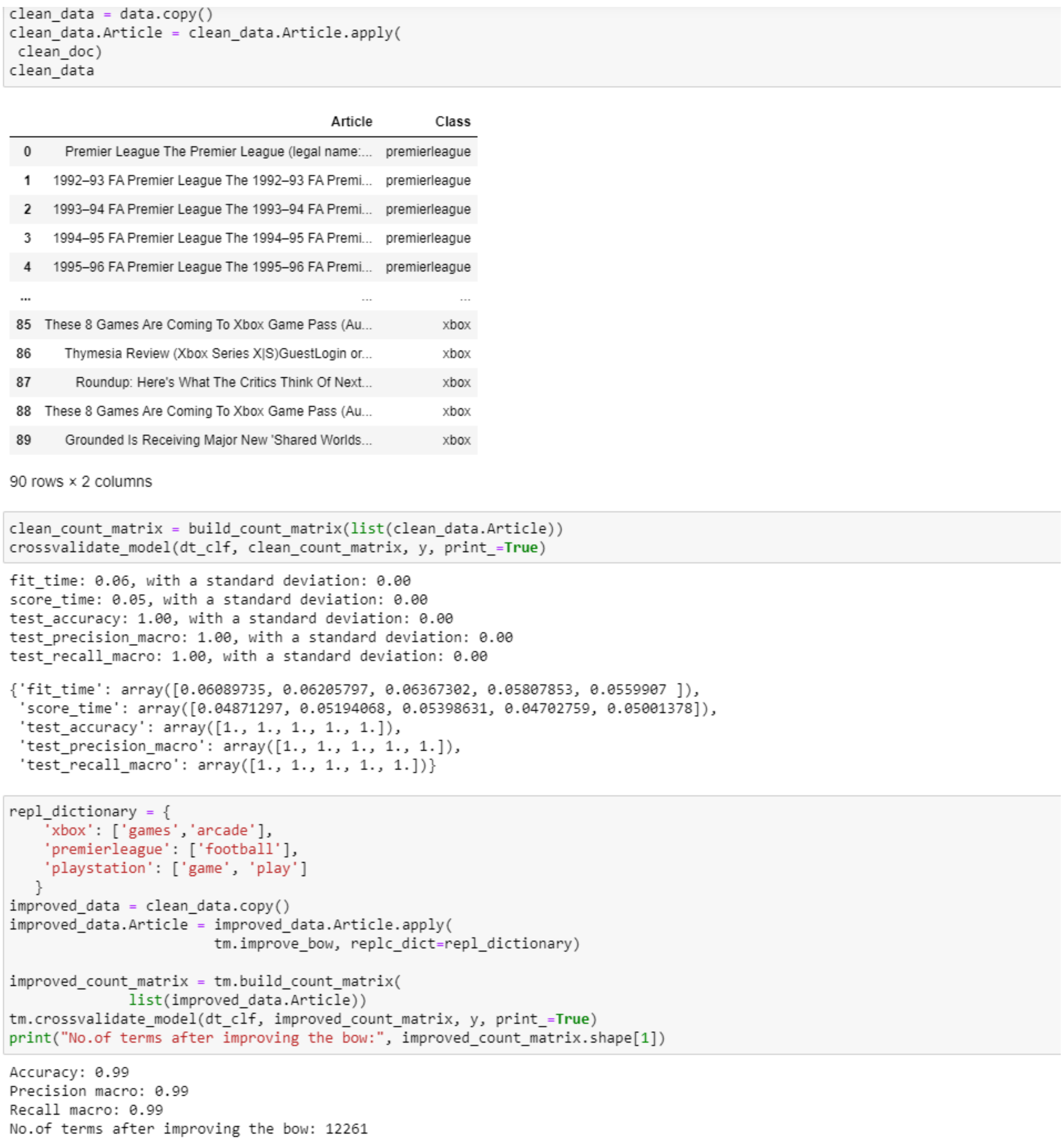
```
fit_time: 0.06, with a standard deviation: 0.00
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.99, with a standard deviation: 0.02
test_precision_macro: 0.99, with a standard deviation: 0.02
test_recall_macro: 0.99, with a standard deviation: 0.02

{'fit_time': array([0.06723166, 0.05599904, 0.05921531, 0.05700088, 0.0549984 ]),
 'score_time': array([0.04900026, 0.04500127, 0.04800391, 0.04477978, 0.04500222]),
 'test_accuracy': array([1.        , 1.        , 1.        , 0.94444444, 1.        ]),
 'test_precision_macro': array([1.        , 1.        , 1.        , 0.95238095, 1.        ]),
 'test_recall_macro': array([1.        , 1.        , 1.        , 0.94444444, 1.        ])}
```

```
dt_clf = DecisionTreeClassifier(random_state=1)
y = data.Class
crossvalidate_model(dt_clf, baseline_tfidf_matrix, y, print_=True)
```

```
fit_time: 0.06, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.98, with a standard deviation: 0.03
test_precision_macro: 0.98, with a standard deviation: 0.02
test_recall_macro: 0.98, with a standard deviation: 0.03

{'fit_time': array([0.07053518, 0.05699801, 0.06095481, 0.06393337, 0.05599856]),
 'score_time': array([0.04575586, 0.05053878, 0.04500008, 0.04991698, 0.04902506]),
 'test_accuracy': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ]),
 'test_precision_macro': array([1.        , 1.        , 0.95238095, 0.95238095, 1.        ]),
 'test_recall_macro': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ])}
```

```
crossvalidate_model(dt_clf, baseline_tfidf_matrix, y, print_=True)
```

```
fit_time: 0.06, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.98, with a standard deviation: 0.03
test_precision_macro: 0.98, with a standard deviation: 0.02
test_recall_macro: 0.98, with a standard deviation: 0.03

{'fit_time': array([0.07006001, 0.05799627, 0.05599666, 0.05703259, 0.05671525]),
 'score_time': array([0.04771757, 0.04500484, 0.05086303, 0.05298638, 0.05801797]),
 'test_accuracy': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ]),
 'test_precision_macro': array([1.        , 1.        , 0.95238095, 0.95238095, 1.        ]),
 'test_recall_macro': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ])}
```



```
clean_data = data.copy()
clean_data.Article = clean_data.Article.apply(
 clean_doc)
clean_data
```

	Article	Class
0	Premier League The Premier League (legal name:...	premierleague
1	1992–93 FA Premier League The 1992–93 FA Premi...	premierleague
2	1993–94 FA Premier League The 1993–94 FA Premi...	premierleague
3	1994–95 FA Premier League The 1994–95 FA Premi...	premierleague
4	1995–96 FA Premier League The 1995–96 FA Premi...	premierleague
...	...	...
85	These 8 Games Are Coming To Xbox Game Pass (Au...	xbox
86	Thymesia Review (Xbox Series X|S)GuestLogin or...	xbox
87	Roundup: Here's What The Critics Think Of Next...	xbox
88	These 8 Games Are Coming To Xbox Game Pass (Au...	xbox
89	Grounded Is Receiving Major New 'Shared Worlds...	xbox

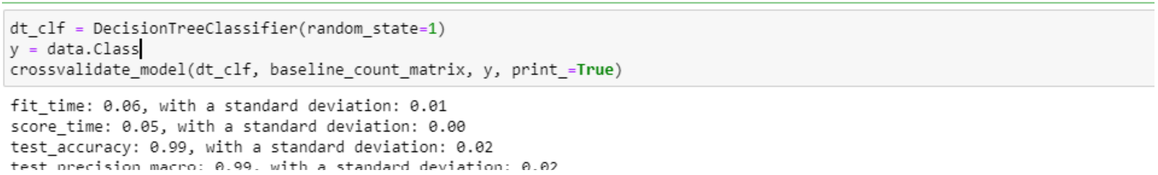
90 rows × 2 columns

```
clean_count_matrix = build_count_matrix(list(clean_data.Article))
crossvalidate_model(dt_clf, clean_count_matrix, y, print_=True)
```

```
fit_time: 0.06, with a standard deviation: 0.00
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 1.00, with a standard deviation: 0.00
test_precision_macro: 1.00, with a standard deviation: 0.00
test_recall_macro: 1.00, with a standard deviation: 0.00

{'fit_time': array([0.06089735, 0.06205797, 0.06367302, 0.05807853, 0.0559907 ]),
 'score_time': array([0.04871297, 0.05194068, 0.05398631, 0.04702759, 0.05001378]),
 'test_accuracy': array([1., 1., 1., 1., 1.]),
 'test_precision_macro': array([1., 1., 1., 1., 1.]),
 'test_recall_macro': array([1., 1., 1., 1., 1.])}
```

```
repl_dictionary = {
    'xbox': ['games','arcade'],
    'premierleague': ['football'],
    'playstation': ['game', 'play']
   }
improved_data = clean_data.copy()
improved_data.Article = improved_data.Article.apply(
                        tm.improve_bow, replc_dict=repl_dictionary)

improved_count_matrix = tm.build_count_matrix(
              list(improved_data.Article))
tm.crossvalidate_model(dt_clf, improved_count_matrix, y, print_=True)
print("No.of terms after improving the bow:", improved_count_matrix.shape[1])
```

```
Accuracy: 0.99
Precision macro: 0.99
Recall macro: 0.99
No.of terms after improving the bow: 12261
```

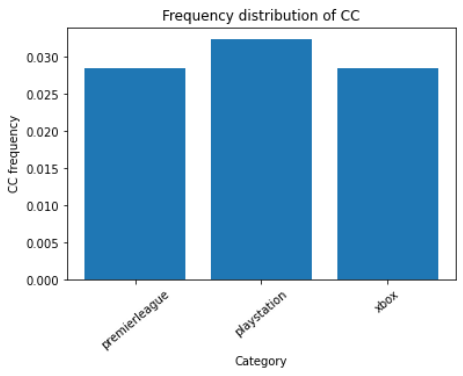


```
dt_clf = DecisionTreeClassifier(random_state=1)
y = data.Class
crossvalidate_model(dt_clf, baseline_count_matrix, y, print_=True)
```

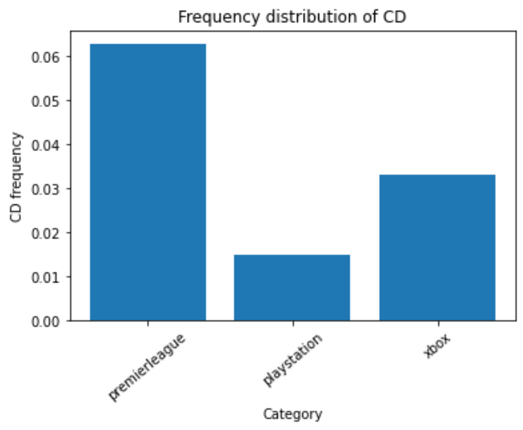
```
fit_time: 0.06, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.99, with a standard deviation: 0.02
test_precision_macro: 0.99, with a standard deviation: 0.02
```

```
def normalise_POS_counts(tagged_docs, pos):
    counts=[]
 ## go through each tagged document (which is a list of tuples)
    for d in tagged_docs:
        count = 0
 ## go through each tuple
        for pair in d:
            if pair[1] == pos:
                count += 1
        counts.append(count)
    lengths = [len(d) for d in tagged_docs]
 ## return the list of each document' POS count dived by the total number of tuples
    return [count/length for count, length in zip(counts, lengths)]
## plots the frequency of a POS across texts/docs in a list
def plot_POS_freq(docs, pos, categories):
 ## tokenise and tag each document from the list docs
    tagged_docs = [nltk.pos_tag(nltk.word_tokenize(doc)) for doc in docs]
 ## get the normalised counts of the POS from those tagged docs
    normalised_counts = normalise_POS_counts(tagged_docs, pos)
 ## create the bar chart with those normalised counts
    plt.bar(np.arange(len(docs)), normalised_counts, align='center')
 ## label the ticks with the name of each category and place at 40 degrees
    plt.xticks(np.arange(len(docs)), categories, rotation=40)
    plt.xlabel('Category')
    plt.ylabel(pos + " frequency")
    plt.title('Frequency distribution of ' + pos)
```

```
texts = [premierleague_text, playstation_text, xbox_text]
## visualise the frequency of CC part of speech (conjunctions)
## across the 3 categories
plot_POS_freq(texts, 'CC', ['premierleague', 'playstation', 'xbox'])
```



```
texts = [premierleague_text, playstation_text, xbox_text]
## visualise the frequency of CC part of speech (conjunctions)
## across the 3 categories
plot_POS_freq(texts, 'CD', ['premierleague', 'playstation', 'xbox'])
```



```
#clustering
y = data.Class
print(y)
```

```
0     premierleague
1     premierleague
2     premierleague
3     premierleague
4     premierleague
          ...
85             xbox
86             xbox
87             xbox
88             xbox
89             xbox
Name: Class, Length: 90, dtype: object
```

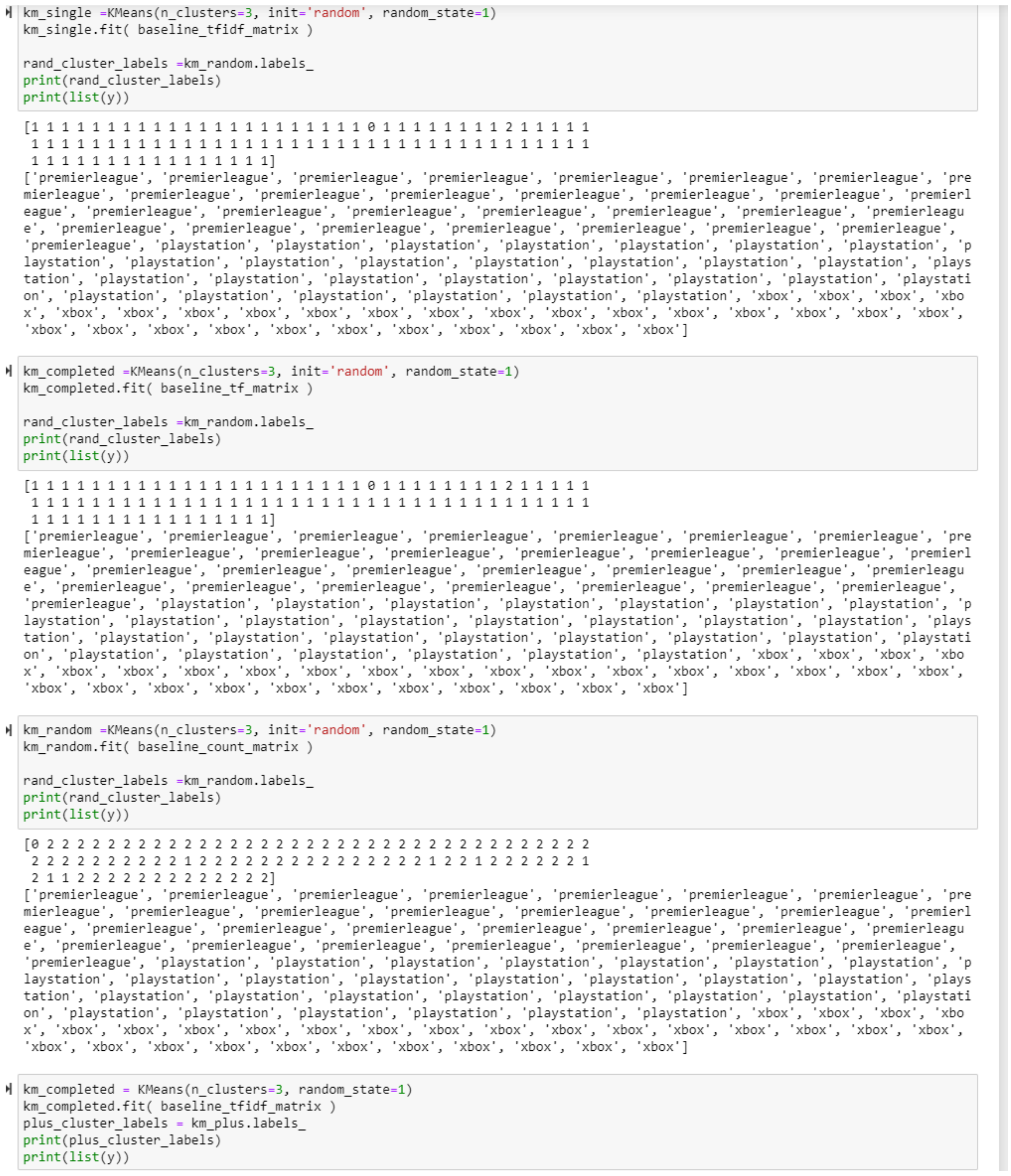
```
cosine_similarity([
    baseline_tfidf_matrix.iloc[0],
    baseline_tfidf_matrix.iloc[30],
    baseline_tfidf_matrix.iloc[60]
])
```

```
array([[1.        , 0.0083948 , 0.0202227 ],
       [0.0083948 , 1.        , 0.02358423],
       [0.0202227 , 0.02358423, 1.        ]])
```

```
cosine_distances([
    baseline_tfidf_matrix.iloc[0],
    baseline_tfidf_matrix.iloc[1],
    baseline_tfidf_matrix.iloc[2]
])
```

```
array([[0.        , 0.51762938, 0.74059582],
       [0.51762938, 0.        , 0.62998056],
       [0.74059582, 0.62998056, 0.        ]])
```

```
km_single =KMeans(n_clusters=3, init='random', random_state=1)
km_single.fit( baseline_tfidf_matrix )

rand_cluster_labels =km_random.labels_
print(rand_cluster_labels)
```



```
km_single =KMeans(n_clusters=3, init='random', random_state=1)
km_single.fit( baseline_tfidf_matrix )

rand_cluster_labels =km_random.labels_
print(rand_cluster_labels)
print(list(y))
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 2 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
['premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'pre
mierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierl
eague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleagu
e', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague',
'premierleague', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'p
laystation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'plays
tation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstati
on', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'xbox', 'xbox', 'xbox', 'xbo
x', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox',
'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox']
```

```
km_completed =KMeans(n_clusters=3, init='random', random_state=1)
km_completed.fit( baseline_tf_matrix )

rand_cluster_labels =km_random.labels_
print(rand_cluster_labels)
print(list(y))
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 2 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
['premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'pre
mierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierl
eague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleagu
e', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague',
'premierleague', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'p
laystation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'plays
tation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstati
on', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'xbox', 'xbox', 'xbox', 'xbo
x', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox',
'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox']
```

```
km_random =KMeans(n_clusters=3, init='random', random_state=1)
km_random.fit( baseline_count_matrix )

rand_cluster_labels =km_random.labels_
print(rand_cluster_labels)
print(list(y))
```

```
[0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2 2 2 1
 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2]
['premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'pre
mierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierl
eague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleagu
e', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague',
'premierleague', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'p
laystation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'plays
tation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstati
on', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'xbox', 'xbox', 'xbox', 'xbo
x', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox',
'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox']
```

```
km_completed = KMeans(n_clusters=3, random_state=1)
km_completed.fit( baseline_tfidf_matrix )
plus_cluster_labels = km_plus.labels_
print(plus_cluster_labels)
print(list(y))
```

```
km_plus = KMeans(n_clusters=3, random_state=1)
km_plus.fit( baseline_count_matrix )
plus_cluster_labels = km_plus.labels_
print(plus_cluster_labels)
print(list(y))
```

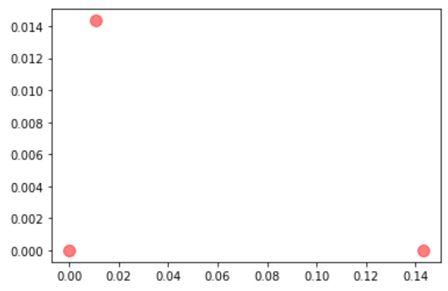
```
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0]
['premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'pre
mierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierl
eague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleagu
e', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague',
'premierleague', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'p
laystation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'plays
tation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstati
on', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'xbox', 'xbox', 'xbox', 'xbo
x', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox',
'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox']
```

```
km_plus = KMeans(n_clusters=3, random_state=1)
km_plus.fit( baseline_tf_matrix )
plus_cluster_labels = km_plus.labels_
print(plus_cluster_labels)
print(list(y))
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
['premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'pre
mierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierl
eague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleagu
e', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague', 'premierleague',
'premierleague', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'p
laystation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'plays
tation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstati
on', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'playstation', 'xbox', 'xbox', 'xbox', 'xbo
x', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox',
'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox', 'xbox']
```

```
km_plus_centers = km_plus.cluster_centers_
plt.scatter(km_plus_centers[:,0], km_plus_centers[:, 1],
            c='red', s=100, alpha=0.5)
```

```
<matplotlib.collections.PathCollection at 0x1f7940a9910>
```



```
print(homogeneity_score(plus_cluster_labels, list(y)))
print(completeness_score(plus_cluster_labels, list(y)))
```

```
0.9289071107752024
0.5793801642856948
```

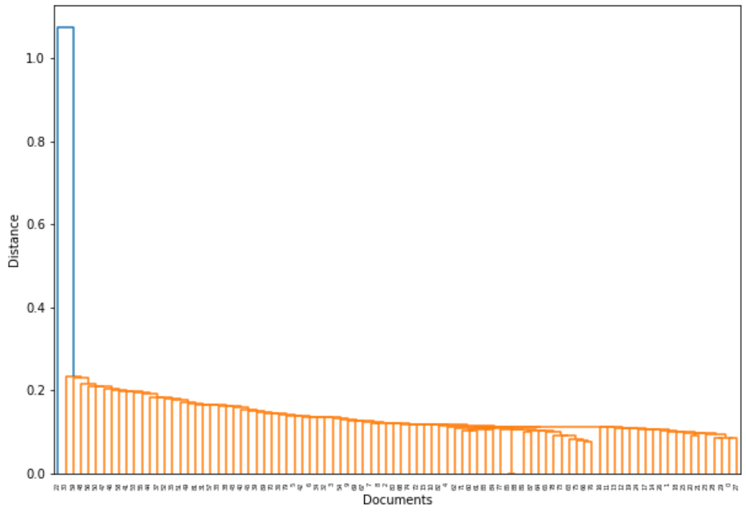
```python
print(homogeneity_score(agg_single_labels, list(y)))
print(completeness_score(agg_single_labels, list(y)))
```

```
0.20119335781587197
0.022336723417234094
```

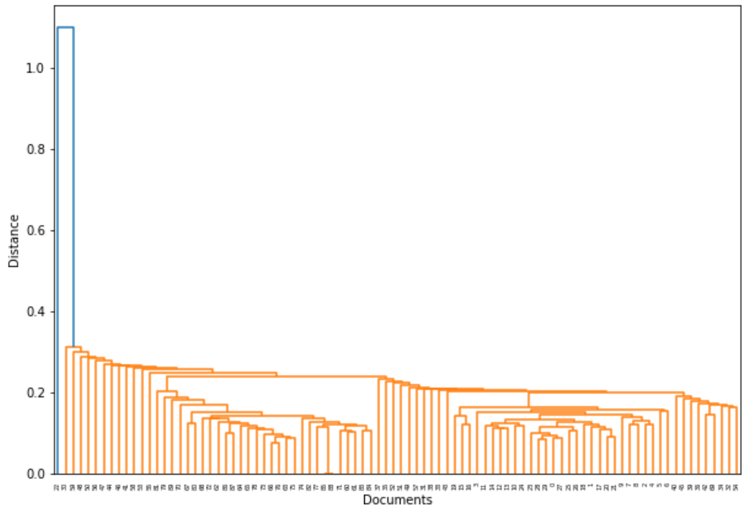
```python
print(homogeneity_score(agg_complete_labels, list(y)))
print(completeness_score(agg_complete_labels, list(y)))
```

```
0.533360327308301
0.2785816235935532
```

```python
tm.display_dendrogram(baseline_tfidf_matrix, method='single')
```

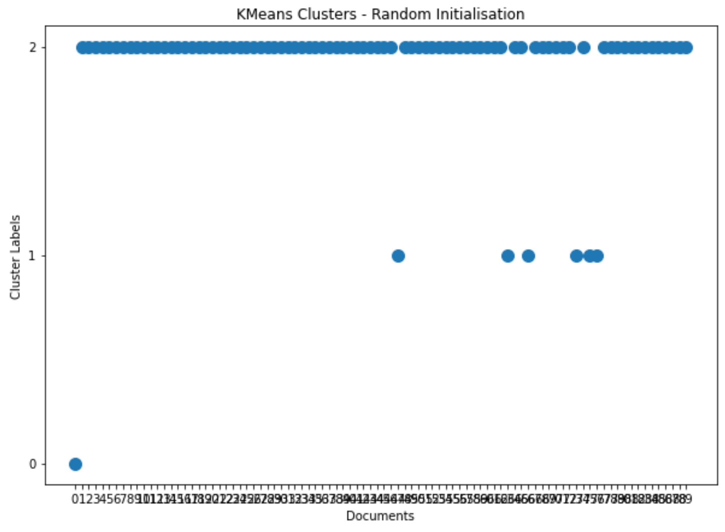


```python
tm.display_dendrogram(baseline_tfidf_matrix, method='complete')
```

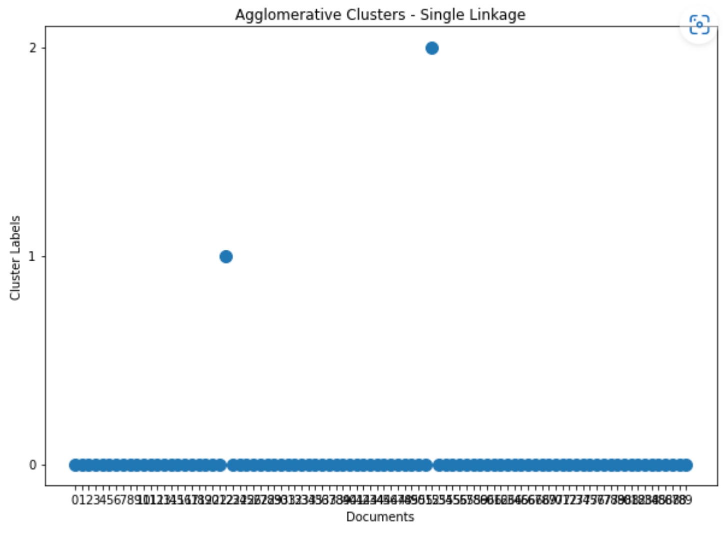


Documents

```
tm.plot_clusters(baseline_tfidf_matrix, rand_cluster_labels, 'KMeans Clusters - Random Initialisation')
```

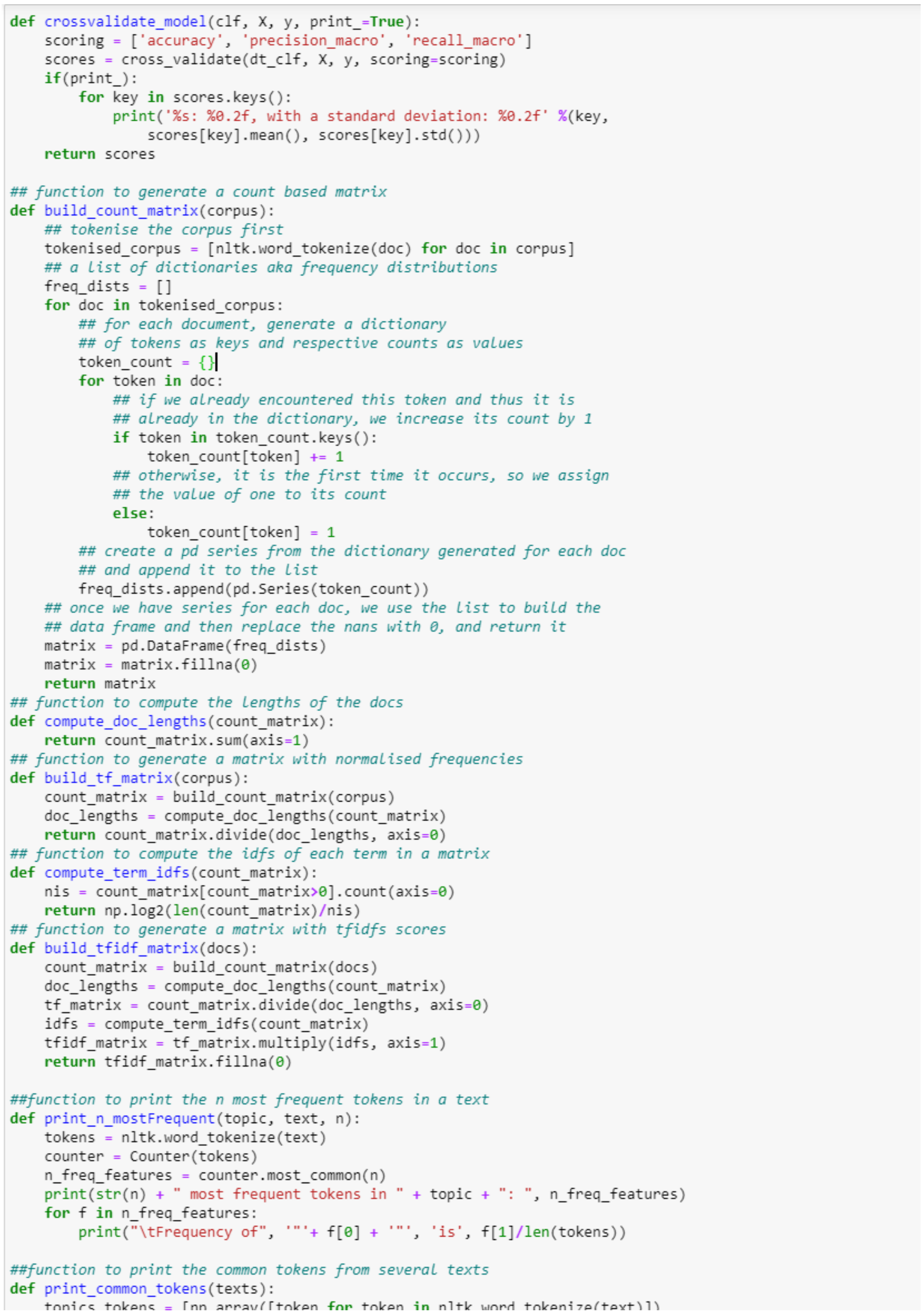


```
tm.plot_clusters(baseline_tfidf_matrix, agg_single_labels, 'Agglomerative Clusters - Single Linkage')
```

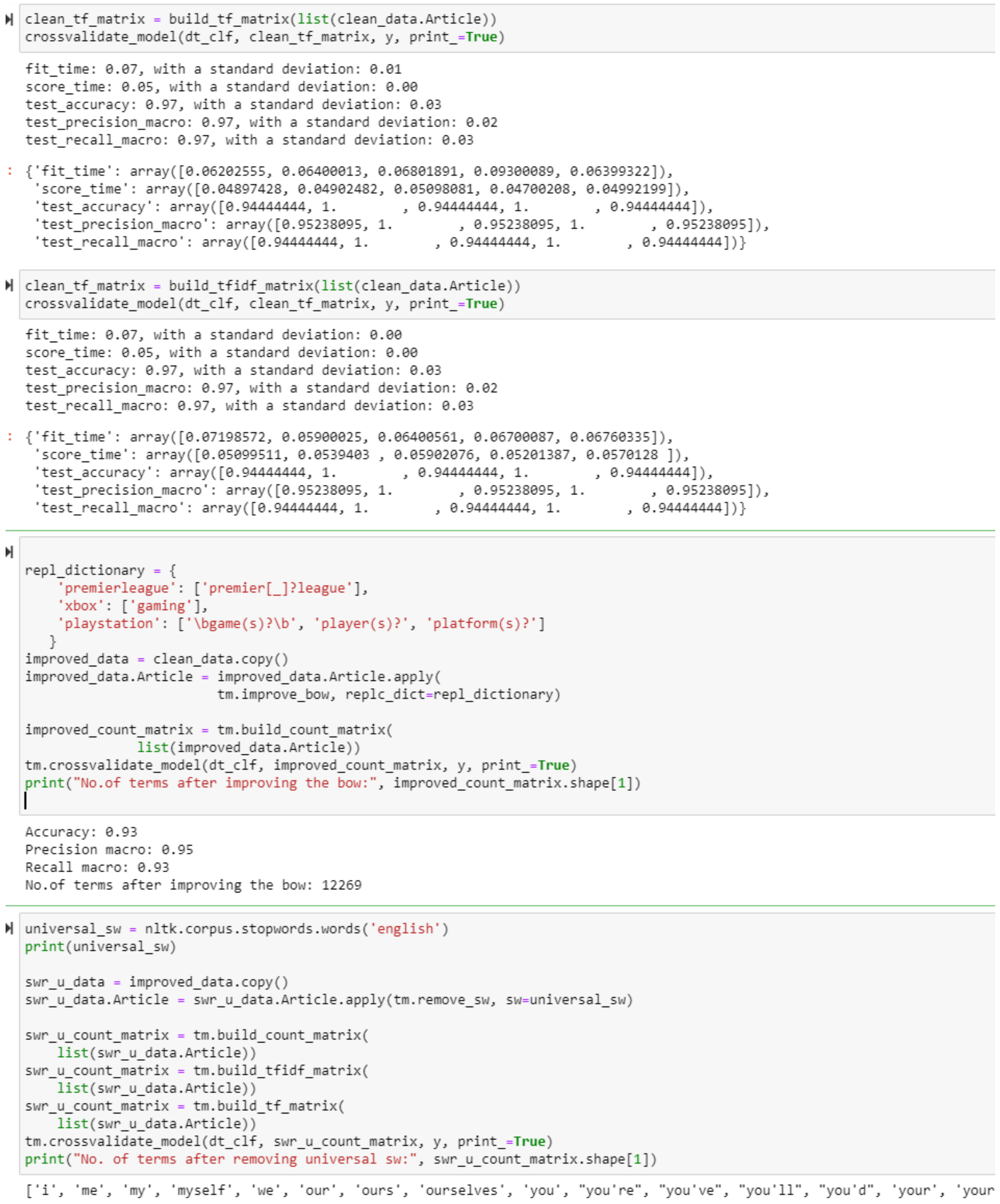


```
tm.print_n_mostFrequent("premierleague", premierleague_text, 5)
tm.print_n_mostFrequent("playstation", playstation_text, 5)
tm.print_n_mostFrequent("xbox", xbox_text, 5)
```

```
5 most frequent tokens in premierleague:  [('the', 2924), (',', 1868), ('.', 1505), ('of', 1099), ('in', 956)]
        Frequency of "the" is 0.064297651508488
        Frequency of "," is 0.04107661183921189
        Frequency of "." is 0.03309437945289823
        Frequency of "of" is 0.024166593367930336
        Frequency of "in" is 0.02102207757938253
5 most frequent tokens in playstation:  [(',', 1544), ('the', 1191), ('.', 927), ('to', 868), ('and', 754)]
        Frequency of "," is 0.05140155802649977
        Frequency of "the" is 0.03964977694919768
        Frequency of "." is 0.030860909514614822
        Frequency of "to" is 0.028896730807643652
        Frequency of "and" is 0.025101538051801053
5 most frequent tokens in xbox:  [(',', 1791), ('the', 1357), ('.', 1185), ('to', 1099), ('a', 963)]
        Frequency of "," is 0.03422641797890231
        Frequency of "the" is 0.025932579116343068
        Frequency of "." is 0.022645619935789636
        Frequency of "to" is 0.02100214034551292
        Frequency of "a" is 0.01840314936554044
```



```python
def crossvalidate_model(clf, X, y, print_=True):
    scoring = ['accuracy', 'precision_macro', 'recall_macro']
    scores = cross_validate(dt_clf, X, y, scoring=scoring)
    if(print_):
        for key in scores.keys():
            print('%s: %0.2f, with a standard deviation: %0.2f' %(key,
                scores[key].mean(), scores[key].std()))
    return scores

## function to generate a count based matrix
def build_count_matrix(corpus):
    ## tokenise the corpus first
    tokenised_corpus = [nltk.word_tokenize(doc) for doc in corpus]
    ## a list of dictionaries aka frequency distributions
    freq_dists = []
    for doc in tokenised_corpus:
        ## for each document, generate a dictionary
        ## of tokens as keys and respective counts as values
        token_count = {}
        for token in doc:
            ## if we already encountered this token and thus it is
            ## already in the dictionary, we increase its count by 1
            if token in token_count.keys():
                token_count[token] += 1
            ## otherwise, it is the first time it occurs, so we assign
            ## the value of one to its count
            else:
                token_count[token] = 1
        ## create a pd series from the dictionary generated for each doc
        ## and append it to the list
        freq_dists.append(pd.Series(token_count))
    ## once we have series for each doc, we use the list to build the
    ## data frame and then replace the nans with 0, and return it
    matrix = pd.DataFrame(freq_dists)
    matrix = matrix.fillna(0)
    return matrix
## function to compute the lengths of the docs
def compute_doc_lengths(count_matrix):
    return count_matrix.sum(axis=1)
## function to generate a matrix with normalised frequencies
def build_tf_matrix(corpus):
    count_matrix = build_count_matrix(corpus)
    doc_lengths = compute_doc_lengths(count_matrix)
    return count_matrix.divide(doc_lengths, axis=0)
## function to compute the idfs of each term in a matrix
def compute_term_idfs(count_matrix):
    nis = count_matrix[count_matrix>0].count(axis=0)
    return np.log2(len(count_matrix)/nis)
## function to generate a matrix with tfidfs scores
def build_tfidf_matrix(docs):
    count_matrix = build_count_matrix(docs)
    doc_lengths = compute_doc_lengths(count_matrix)
    tf_matrix = count_matrix.divide(doc_lengths, axis=0)
    idfs = compute_term_idfs(count_matrix)
    tfidf_matrix = tf_matrix.multiply(idfs, axis=1)
    return tfidf_matrix.fillna(0)

##function to print the n most frequent tokens in a text
def print_n_mostFrequent(topic, text, n):
    tokens = nltk.word_tokenize(text)
    counter = Counter(tokens)
    n_freq_features = counter.most_common(n)
    print(str(n) + " most frequent tokens in " + topic + ": ", n_freq_features)
    for f in n_freq_features:
        print("\tFrequency of", '"'+ f[0] + '"', 'is', f[1]/len(tokens))

##function to print the common tokens from several texts
def print_common_tokens(texts):
    topics_tokens = [np.array([token for token in nltk.word_tokenize(text)])
```



```
clean_tf_matrix = build_tf_matrix(list(clean_data.Article))
crossvalidate_model(dt_clf, clean_tf_matrix, y, print_=True)
```

```
fit_time: 0.07, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.97, with a standard deviation: 0.03
test_precision_macro: 0.97, with a standard deviation: 0.02
test_recall_macro: 0.97, with a standard deviation: 0.03
```

```
: {'fit_time': array([0.06202555, 0.06400013, 0.06801891, 0.09300089, 0.06399322]),
   'score_time': array([0.04897428, 0.04902482, 0.05098081, 0.04700208, 0.04992199]),
   'test_accuracy': array([0.94444444, 1.        , 0.94444444, 1.        , 0.94444444]),
   'test_precision_macro': array([0.95238095, 1.        , 0.95238095, 1.        , 0.95238095]),
   'test_recall_macro': array([0.94444444, 1.        , 0.94444444, 1.        , 0.94444444])}
```

```
clean_tf_matrix = build_tfidf_matrix(list(clean_data.Article))
crossvalidate_model(dt_clf, clean_tf_matrix, y, print_=True)
```

```
fit_time: 0.07, with a standard deviation: 0.00
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.97, with a standard deviation: 0.03
test_precision_macro: 0.97, with a standard deviation: 0.02
test_recall_macro: 0.97, with a standard deviation: 0.03
```

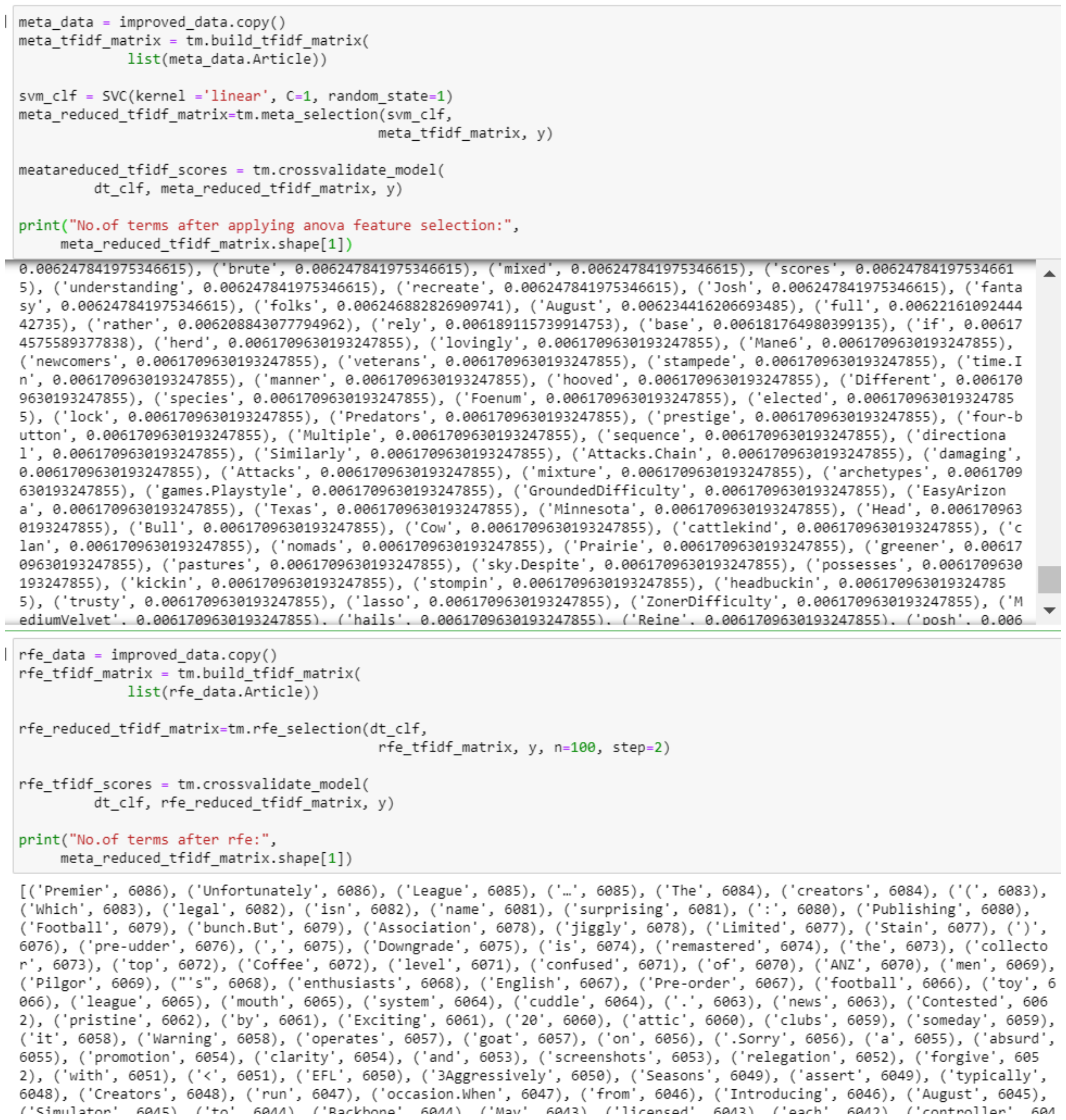
```
: {'fit_time': array([0.07198572, 0.05900025, 0.06400561, 0.06700087, 0.06760335]),
   'score_time': array([0.05099511, 0.0539403 , 0.05902076, 0.05201387, 0.0570128 ]),
   'test_accuracy': array([0.94444444, 1.        , 0.94444444, 1.        , 0.94444444]),
   'test_precision_macro': array([0.95238095, 1.        , 0.95238095, 1.        , 0.95238095]),
   'test_recall_macro': array([0.94444444, 1.        , 0.94444444, 1.        , 0.94444444])}
```

```

repl_dictionary = {
    'premierleague': ['premier[_]?league'],
    'xbox': ['gaming'],
    'playstation': ['\bgame(s)?\b', 'player(s)?', 'platform(s)?']
   }
improved_data = clean_data.copy()
improved_data.Article = improved_data.Article.apply(
                        tm.improve_bow, replc_dict=repl_dictionary)

improved_count_matrix = tm.build_count_matrix(
              list(improved_data.Article))
tm.crossvalidate_model(dt_clf, improved_count_matrix, y, print_=True)
print("No.of terms after improving the bow:", improved_count_matrix.shape[1])
```

```
Accuracy: 0.93
Precision macro: 0.95
Recall macro: 0.93
No.of terms after improving the bow: 12269
```

```
universal_sw = nltk.corpus.stopwords.words('english')
print(universal_sw)

swr_u_data = improved_data.copy()
swr_u_data.Article = swr_u_data.Article.apply(tm.remove_sw, sw=universal_sw)

swr_u_count_matrix = tm.build_count_matrix(
    list(swr_u_data.Article))
swr_u_count_matrix = tm.build_tfidf_matrix(
    list(swr_u_data.Article))
swr_u_count_matrix = tm.build_tf_matrix(
    list(swr_u_data.Article))
tm.crossvalidate_model(dt_clf, swr_u_count_matrix, y, print_=True)
print("No. of terms after removing universal sw:", swr_u_count_matrix.shape[1])
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'your
```



```
meta_data = improved_data.copy()
meta_tfidf_matrix = tm.build_tfidf_matrix(
             list(meta_data.Article))

svm_clf = SVC(kernel ='linear', C=1, random_state=1)
meta_reduced_tfidf_matrix=tm.meta_selection(svm_clf,
                                          meta_tfidf_matrix, y)

meatareduced_tfidf_scores = tm.crossvalidate_model(
         dt_clf, meta_reduced_tfidf_matrix, y)

print("No.of terms after applying anova feature selection:",
     meta_reduced_tfidf_matrix.shape[1])
```

```
0.006247841975346615), ('brute', 0.006247841975346615), ('mixed', 0.006247841975346615), ('scores', 0.00624784197534661
5), ('understanding', 0.006247841975346615), ('recreate', 0.006247841975346615), ('Josh', 0.006247841975346615), ('fanta
sy', 0.006247841975346615), ('folks', 0.006246882826909741), ('August', 0.006234416206693485), ('full', 0.00622161092444
42735), ('rather', 0.006208843077794962), ('rely', 0.006189115739914753), ('base', 0.006181764980399135), ('if', 0.00617
4575589377838), ('herd', 0.006170963019324785), ('lovingly', 0.006170963019324785), ('Mane6', 0.006170963019324785),
('newcomers', 0.006170963019324785), ('veterans', 0.006170963019324785), ('stampede', 0.006170963019324785), ('time.I
n', 0.006170963019324785), ('manner', 0.006170963019324785), ('hooved', 0.006170963019324785), ('Different', 0.006170
963019324785), ('species', 0.006170963019324785), ('Foenum', 0.006170963019324785), ('elected', 0.00617096301932478
5), ('lock', 0.006170963019324785), ('Predators', 0.006170963019324785), ('prestige', 0.006170963019324785), ('four-b
utton', 0.006170963019324785), ('Multiple', 0.006170963019324785), ('sequence', 0.006170963019324785), ('directiona
l', 0.006170963019324785), ('Similarly', 0.006170963019324785), ('Attacks.Chain', 0.006170963019324785), ('damaging',
0.006170963019324785), ('Attacks', 0.006170963019324785), ('mixture', 0.006170963019324785), ('archetypes', 0.0061709
63019324785), ('games.Playstyle', 0.006170963019324785), ('GroundedDifficulty', 0.006170963019324785), ('EasyArizon
a', 0.006170963019324785), ('Texas', 0.006170963019324785), ('Minnesota', 0.006170963019324785), ('Head', 0.006170963
019324785), ('Bull', 0.006170963019324785), ('Cow', 0.006170963019324785), ('cattlekind', 0.006170963019324785), ('c
lan', 0.006170963019324785), ('nomads', 0.006170963019324785), ('Prairie', 0.006170963019324785), ('greener', 0.00617
0963019324785), ('pastures', 0.006170963019324785), ('sky.Despite', 0.006170963019324785), ('possesses', 0.0061709630
19324785), ('kickin', 0.006170963019324785), ('stompin', 0.006170963019324785), ('headbuckin', 0.00617096301932478
5), ('trusty', 0.006170963019324785), ('lasso', 0.006170963019324785), ('ZonerDifficulty', 0.006170963019324785), ('M
ediumVelvet', 0.006170963019324785), ('hails', 0.006170963019324785), ('Reine', 0.006170963019324785), ('posh', 0.006
```

```
rfe_data = improved_data.copy()
rfe_tfidf_matrix = tm.build_tfidf_matrix(
             list(rfe_data.Article))

rfe_reduced_tfidf_matrix=tm.rfe_selection(dt_clf,
                                          rfe_tfidf_matrix, y, n=100, step=2)

rfe_tfidf_scores = tm.crossvalidate_model(
         dt_clf, rfe_reduced_tfidf_matrix, y)

print("No.of terms after rfe:",
     meta_reduced_tfidf_matrix.shape[1])
```

```
[('Premier', 6086), ('Unfortunately', 6086), ('League', 6085), ('…', 6085), ('The', 6084), ('creators', 6084), ('(', 6083),
('Which', 6083), ('legal', 6082), ('isn', 6082), ('name', 6081), ('surprising', 6081), (':', 6080), ('Publishing', 6080),
('Football', 6079), ('bunch.But', 6079), ('Association', 6078), ('jiggly', 6078), ('Limited', 6077), ('Stain', 6077), (')',
6076), ('pre-udder', 6076), (',', 6075), ('Downgrade', 6075), ('is', 6074), ('remastered', 6074), ('the', 6073), ('collecto
r', 6073), ('top', 6072), ('Coffee', 6072), ('level', 6071), ('confused', 6071), ('of', 6070), ('ANZ', 6070), ('men', 6069),
('Pilgor', 6069), ("'s", 6068), ('enthusiasts', 6068), ('English', 6067), ('Pre-order', 6067), ('football', 6066), ('toy', 6
066), ('league', 6065), ('mouth', 6065), ('system', 6064), ('cuddle', 6064), ('.', 6063), ('news', 6063), ('Contested', 606
2), ('pristine', 6062), ('by', 6061), ('Exciting', 6061), ('20', 6060), ('attic', 6060), ('clubs', 6059), ('someday', 6059),
('it', 6058), ('Warning', 6058), ('operates', 6057), ('goat', 6057), ('on', 6056), ('.Sorry', 6056), ('a', 6055), ('absurd',
6055), ('promotion', 6054), ('clarity', 6054), ('and', 6053), ('screenshots', 6053), ('relegation', 6052), ('forgive', 605
2), ('with', 6051), ('<', 6051), ('EFL', 6050), ('3Aggressively', 6050), ('Seasons', 6049), ('assert', 6049), ('typically',
6048), ('Creators', 6048), ('run', 6047), ('occasion.When', 6047), ('from', 6046), ('Introducing', 6046), ('August', 6045),
('Simulator', 6045), ('to', 6044), ('Backbone', 6044), ('May', 6043), ('licensed', 6043), ('each', 6042), ('controller', 604
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier()
crossvalidate_model(knn_clf, baseline_count_matix, y, print_=true)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9368/3397601467.py in <module>
      1 from sklearn.neighbors import KNeighborsClassifier
      2 knn_clf = KNeighborsClassifier()
----> 3 crossvalidate_model(knn_clf, baseline_count_matix, y, print_=true)

NameError: name 'baseline_count_matix' is not defined
```

```
from sklearn.neighbors import KNeighborsClassifier
## same approach as in previous cell, but using a KNN instead of a DT
knn_clf = KNeighborsClassifier()
crossvalidate_model(knn_clf, baseline_count_matrix, y, print_=True)
```

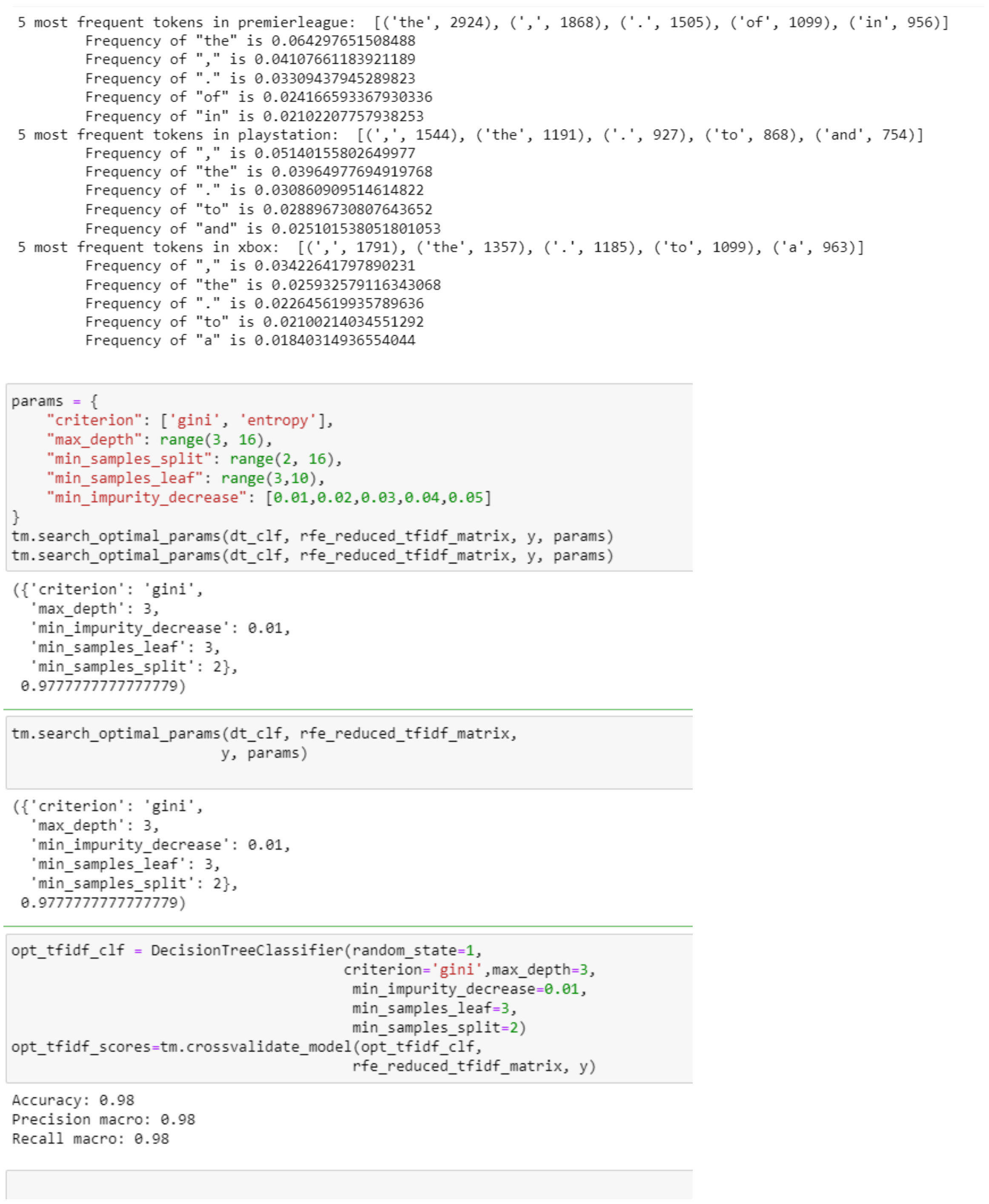
```
fit_time: 0.06, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.99, with a standard deviation: 0.02
test_precision_macro: 0.99, with a standard deviation: 0.02
test_recall_macro: 0.99, with a standard deviation: 0.02

{'fit_time': array([0.08868408, 0.05699682, 0.05807805, 0.05699492, 0.0570066 ]),
 'score_time': array([0.05207491, 0.04501104, 0.04499722, 0.04700041, 0.04791474]),
 'test_accuracy': array([1.        , 1.        , 1.        , 0.94444444, 1.        ]),
 'test_precision_macro': array([1.        , 1.        , 1.        , 0.95238095, 1.        ]),
 'test_recall_macro': array([1.        , 1.        , 1.        , 0.94444444, 1.        ])}
```

```
from sklearn.neighbors import KNeighborsClassifier
## same approach as in previous cell, but using a KNN instead of a DT
knn_clf = KNeighborsClassifier()
crossvalidate_model(knn_clf, baseline_tfidf_matrix, y, print_=True)
```

```
fit_time: 0.06, with a standard deviation: 0.01
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.98, with a standard deviation: 0.03
test_precision_macro: 0.98, with a standard deviation: 0.02
test_recall_macro: 0.98, with a standard deviation: 0.03

{'fit_time': array([0.07894254, 0.06056499, 0.05799747, 0.05699563, 0.05799294]),
 'score_time': array([0.05000424, 0.04600096, 0.04708624, 0.04500341, 0.04791927]),
 'test_accuracy': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ]),
 'test_precision_macro': array([1.        , 1.        , 0.95238095, 0.95238095, 1.        ]),
 'test_recall_macro': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ])}
```

```
from sklearn.neighbors import KNeighborsClassifier
## same approach as in previous cell, but using a KNN instead of a DT
knn_clf = KNeighborsClassifier()
crossvalidate_model(knn_clf, baseline_tf_matrix, y, print_=True)
```

```
fit_time: 0.07, with a standard deviation: 0.02
score_time: 0.05, with a standard deviation: 0.00
test_accuracy: 0.98, with a standard deviation: 0.03
test_precision_macro: 0.98, with a standard deviation: 0.02
test_recall_macro: 0.98, with a standard deviation: 0.03

{'fit_time': array([0.11399674, 0.06001306, 0.05682588, 0.06009102, 0.06308389]),
 'score_time': array([0.04600072, 0.04502106, 0.04695344, 0.04800916, 0.04891992]),
 'test_accuracy': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ]),
 'test_precision_macro': array([1.        , 1.        , 0.95238095, 0.95238095, 1.        ]),
 'test_recall_macro': array([1.        , 1.        , 0.94444444, 0.94444444, 1.        ])}
```



```
5 most frequent tokens in premierleague:  [('the', 2924), (',', 1868), ('.', 1505), ('of', 1099), ('in', 956)]
        Frequency of "the" is 0.064297651508488
        Frequency of "," is 0.04107661183921189
        Frequency of "." is 0.03309437945289823
        Frequency of "of" is 0.024166593367930336
        Frequency of "in" is 0.02102207757938253
5 most frequent tokens in playstation:  [(',', 1544), ('the', 1191), ('.', 927), ('to', 868), ('and', 754)]
        Frequency of "," is 0.05140155802649977
        Frequency of "the" is 0.03964977694919768
        Frequency of "." is 0.030860909514614822
        Frequency of "to" is 0.028896730807643652
        Frequency of "and" is 0.025101538051801053
5 most frequent tokens in xbox:  [(',', 1791), ('the', 1357), ('.', 1185), ('to', 1099), ('a', 963)]
        Frequency of "," is 0.03422641797890231
        Frequency of "the" is 0.025932579116343068
        Frequency of "." is 0.022645619935789636
        Frequency of "to" is 0.02100214034551292
        Frequency of "a" is 0.01840314936554044
```

```
params = {
    "criterion": ['gini', 'entropy'],
    "max_depth": range(3, 16),
    "min_samples_split": range(2, 16),
    "min_samples_leaf": range(3,10),
    "min_impurity_decrease": [0.01,0.02,0.03,0.04,0.05]
}
tm.search_optimal_params(dt_clf, rfe_reduced_tfidf_matrix, y, params)
tm.search_optimal_params(dt_clf, rfe_reduced_tfidf_matrix, y, params)
```

```
({'criterion': 'gini',
  'max_depth': 3,
  'min_impurity_decrease': 0.01,
  'min_samples_leaf': 3,
  'min_samples_split': 2},
 0.9777777777777779)
```

```
tm.search_optimal_params(dt_clf, rfe_reduced_tfidf_matrix,
                         y, params)
```

```
({'criterion': 'gini',
  'max_depth': 3,
  'min_impurity_decrease': 0.01,
  'min_samples_leaf': 3,
  'min_samples_split': 2},
 0.9777777777777779)
```

```
opt_tfidf_clf = DecisionTreeClassifier(random_state=1,
                                       criterion='gini',max_depth=3,
                                        min_impurity_decrease=0.01,
                                        min_samples_leaf=3,
                                        min_samples_split=2)
opt_tfidf_scores=tm.crossvalidate_model(opt_tfidf_clf,
                                        rfe_reduced_tfidf_matrix, y)
```

```
Accuracy: 0.98
Precision macro: 0.98
Recall macro: 0.98
```