```
'''
Suggested Research Question:  Your startup is creating an innovative new email
software. How can you help them create an effective spam filter?
'''


'''
For your third capstone, you'll complete an unsupervised learning project. You
can choose your own research question or choose from one below. In
this capstone, you will be graded by your peers.


Here are the steps:

1. Go out and find a dataset of interest. It could be one that helps you work on
one of Thinkful's recommended research questions, or it could be any other dataset
that addresses an unsupervised learning question of your own.

2. Explore the data. Get to know the data. Spend a lot of time going over its
quirks. You should understand how it was gathered, what's in it, and what the
variables look like.

3. Try several different approaches. Really work to tune a variety of models
before choosing what you consider to be the best performer.

Keep the following considerations in mind: How do clustering and modeling compare?
What are the advantages of each? Why would you want to use one over the other?

This will ultimately include the following deliverables:

- A Jupyter Notebook that tells a compelling story about your data. You'll submit
this Notebook at the end of this checkpoint.
- A 15-to-30-minute presentation of your findings. You'll need to produce a deck
and present it to your peers.

Conduct the analysis in Jupyter. Provide a complete research report, using the
research presentation framework introduced in the "Presenting research results"
checkpoint as a starting point. The report should use compelling visualizations
and actionable insights to tell the story to your intended audience. Walk through
the analysis using clean, reproducible code. Include plenty of notes and comments
to guide others through your thinking.
__________
PROJECT
__________
You·are·improving·upon·your·Spam·Detector·(binary·email·classifier·--·Supervised·
Learning·project)·by·building·a·more·complex·model·than·can·identify·and·filter
out·large·amounts·of·unwanted·comments·from·various·sources.·This·model·can·be
used·by·such·public·platforms·like·Youtube·or·TikTok.
'''
```

Identifying Unwanted Comments for Company Security

Overview

With a world that is becoming increasingly digital and social-media focused, the need to manage and maintain the integrity of online platforms during online engagement has become a necessity.

What are spam comments? The more time spent online, they are bound to come across spam comments. They take many forms, such as being automated by spambots, generic messages as a cover for including links. They mainly pose a risk to online platforms because:

- they interfere with the user experience, increasing the difficulty for legitimate visitors to engage
- decrease the overall legitimacy of the site
- contain redirecting and potentially malicious links, for phishing or malware

Cyber-security is the practice of protecting systems and entities from outside forces designed to infiltrate, change, and gleam sensitive information from them. Today, companies have begun to practically invest in software, professionals and in larger agencies, entire departments familiar with addressing, preventing, and managing these crises situations. Such software can include:

- a created list of "blacklisted" keywords
- hyperlink moderation
- anti-spam plugins(active database of spam)

PROPOSAL

A prospective online platform company is in the market for new software that identifies spam. How can you help them create an effective filter?

The Data

The data for this project was obtained from The University of California, Irvine's public dataset archive, and consists of five csv files. Each file contains sets of comments(341_590 bytes, 1956 in all) extracted from the five most popular YouTube music videos(); for the collection period of 07/2013-04/2015. They files originally came separate but were able to be combined into one large csv using GIT GUI. Following are the links to the dataset:

- https://archive-beta.ics.uci.edu/ml/datasets/youtube+spam+collection
- https://archive.ics.uci.edu/ml/machine-learning-databases/00380/

- github: https://github.com/mwarnsle1/Capstone_3_Spam_Detector-Take-2/blob/f518cdf5ae84c0c59e03dbea57f536ef1d20b883/new_combinevid_files.csv

Methods

As mentioned in the previous section, for my research, a public dataset was obtained from UCI's archives, originally consisting of five individual files that were then combined into one csv file of comments. This was for both ease of accessibility and analysis. The combined datasets catalogued the categorized emails received by users. Before the model creation, I initially applied a number of exploratory analyses; such as removing any null values and exploring the interactions between datetime and other variables.

I then applied some preliminary visualizations and descriptive statistics to check the initial distribution of the variables; by themselves and in relation to each other. Afterwards, I proceeded to apply a progressive variation of models to observe their affect and accuracy on the dataset and clustering abilities on unlabeled data. I applied some clustering methods to the chosen features(K-Means, DBSCAN), in order to display the categorization of the target variable and filter Spam content. I also adjusted the parameters, applied hierarchical clustering to the dataset.

After observing the results, I applied another clustering method with TF-IDF and K-Means to the data, with word clustering included. Following after these, I applied a number of natural language processing techniques to the dataset; specifically, word embedding using Word2Vec and word vectorization. Their effectiveness were used to build a better predictive model, filtering Spam. The results are discussed in the next section.

Results

The exploratory analysis showed a number of trends with the variables that were created. May was by far the most active month, with almost 600 comments, and November being the second most-active month(and also having the most popular date - November 7, 2014). Comments tended to be evenly dispersed throughout the week, though activity went slightly up on the weekends.

The cluster modeling with K-Means and DBSCAN continued to show three variables grouped into two large clusters; both in the scatter plots and PCA plots. Also, the Silhouette score, RI, and ARI for the K-Means model were:

```
Silhouette - 0.4521
RI - 0.5179
ARI - 0.5503
```

However, after the features were applied to the TD-IDF + KMeans cluste model, better results were able to be obtained, as well as some word clustering that could be used in conjuntion with Blacklisting.

Following up on this logic was a WordCloud was created alongside a word frequency, which could also be used for Blacklisting of keywords. A predictive model was then able to be created using natural language processing(NLP) techniques. Using word embedding, scatter plotting was used to visualize dots annotated with the words from the text. These visualized the proximity Spam-like comments had to each other compared to non-Spam comments. From the second NLP model, a predictive model was created using word vectorization; where each categorization was used to detect and filter new emails. A confusion matrix was created to easily compare and display, showing that:

```
[[67  4]
 [ 2 99]]
```

- Actually/predicted to be Ham: 99
- Actually/predicted to be Spam: 67
- Predicted Spam/mistaken for Ham: 4
- Predicted Ham/mistaken for Spam: 2

Discussion & Recommendation

A closer look at the data indicates that techniques for applying the most effective Spam Detection and Filters can be obtained by applying TD-IDF/KMeans clustering with word clsutering, WordCloud with word frequency, and NLP word vectorization models. While DBSCAN algorithm works well to find clusters of any shape and works better than hierarchical clustering, they perform better on more robust datasets.

Drawbacks of the dataset was having an higher number of non-spam("Ham") emails; which may have negatively affected the predictive modeling. This imbalance was overcome for the NLP model; however. Future iterations would still implement measures to balance out the dataset by under-sampling the "Ham" variable, and implement more accuracy measures such as K-Folds cross-validation.

Package + Data Installation:

```
from google.colab import files
files.upload()
```

)21 - 100%

```
/Rh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU,Julius NM,2013-11-07T06:20:48,"Huh, anyway check out
```

```
#Importing packages:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline
import seaborn as sns
import scipy

import datetime
import datetime as dt

from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.cluster import MiniBatchKMeans
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.manifold import TSNE
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
from sklearn import metrics
from sklearn import linear_model
from sqlalchemy import create_engine
import warnings
warnings.filterwarnings("ignore")
```

```
#Loading the Dataset:

yt_spam_df = pd.read_csv('new_combinevid_files.csv')
```

```
#Getting a look at the dataset. Showcasing the 1956 comments available:

yt_spam_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1956 entries, 0 to 1955
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   COMMENT_ID  1956 non-null   object
 1   AUTHOR      1956 non-null   object
 2   DATE        1711 non-null   object
 3   CONTENT     1956 non-null   object
 4   CLASS       1956 non-null   int64
dtypes: int64(1), object(4)
memory usage: 76.5+ KB
```

```
#Thus far, the shape of the dataframe is five columns, with 1956 rows:

yt_spam_df.shape
```

```
(1956, 5)
```

```
#An introductory view:

yt_spam_df.head()
```

	COMMENT_ID	AUTHOR	DATE	CONTENT	CLASS
0	LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU	Julius NM	2013-11-07T06:20:48	Huh, anyway check out this you[tube] channel: ...	1
1	LZQPQhLyRh_C2cTtd9MvFRJedxydaVW-2sNg5Diuo4A	adam riyati	2013-11-07T12:37:15	Hey guys check out my new channel and our firs...	1

Exploratory Data Analysis

```
#creating a deep copy of the dataframe, to preserve the original:

yt_spam2_df = yt_spam_df.copy(deep=True)
```

```
#Establishing the amount of null values in the dataset. It appears the only null values are f

yt_spam2_df.isnull().sum()
```

```
COMMENT_ID      0
AUTHOR          0
DATE          245
CONTENT         0
CLASS           0
dtype: int64
```

```
#Counting the percentage of null values in the dataset:

yt_spam2_df.isnull().sum()*100/yt_spam2_df.isnull().count()
```

```
COMMENT_ID     0.000000
AUTHOR         0.000000
DATE          12.525562
CONTENT        0.000000
CLASS          0.000000
dtype: float64
```

```
#Exploring the DATE column. It appears filled with date and time data that isn't separated, a

yt_spam2_df.DATE.value_counts()
```

```
2014-11-07T19:33:46           2
2013-10-05T00:57:25.078000    2
2014-10-29T22:44:41           1
2015-05-26T09:54:10.695000    1
2015-02-17T04:25:01.940000    1
                             ..
2014-11-13T21:48:26           1
2014-01-21T08:22:06           1
2015-05-19T19:30:58.135000    1
2014-09-15T23:53:03           1
2015-05-26T03:34:08.887000    1
Name: DATE, Length: 1709, dtype: int64
```

```
#Also, b/c the time is attached, each value is unique:

yt_spam2_df.DATE.unique()
```

```
array(['2013-11-07T06:20:48', '2013-11-07T12:37:15',
       '2013-11-08T17:34:21', ..., '2013-07-13T12:09:31.188000',
       '2013-07-13T11:17:52.308000', '2013-07-12T22:33:27.916000'],
      dtype=object)
```

```
yt_spam2_df.DATE.nunique()
```

```
1709
```

```
#Before doing more work on date & time, the missing values need to be addressed. This datafra

yt_spam3_df = yt_spam2_df.copy(deep=True)
yt_spam3_df.dropna(axis=0, inplace=True)

yt_spam3_df.isna().sum()
```

```
COMMENT_ID    0
AUTHOR        0
DATE          0
```

```
CONTENT      0
CLASS        0
dtype: int64
```

Exploring Datetime Trends

```
#For better processing, the DATE column will be converted into datetime values, then separate

yt_spam4_df = yt_spam3_df.copy(deep=True)   #df for date/time conversions


yt_spam4_df["new_DATE"] = yt_spam4_df["DATE"]
yt_spam4_df['new_DATE'] = pd.to_datetime(yt_spam4_df['new_DATE'])




#Re-organizing the new_DATE column; splitting date & time, and creating two new columns:

yt_spam4_df['Comment_Date'] = yt_spam4_df['new_DATE'].dt.date
yt_spam4_df['Comment_Time'] = yt_spam4_df['new_DATE'].dt.time


#With the datetime column now separated, the amount of unique dates for videos can be checked
#326 was more than initially assumed, but the descriptive stats and clustering may tell more

yt_spam4_df.Comment_Date.nunique()
```

```
326
```

```
#Which day had the most comments?

yt_spam4_df.Comment_Date.describe()
```

```
count           1711
unique           326
top       2014-11-07
freq              74
Name: Comment_Date, dtype: object
```

```
yt_spam4_df.info()
```

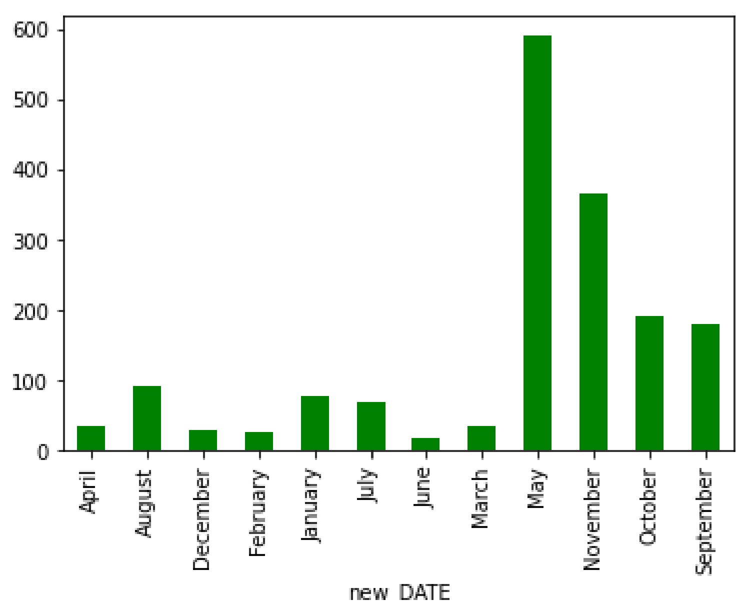
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1711 entries, 0 to 1955
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   COMMENT_ID    1711 non-null   object
 1   AUTHOR        1711 non-null   object
 2   DATE          1711 non-null   object
 3   CONTENT       1711 non-null   object
```

```
 4   CLASS         1711 non-null   int64
 5   new_DATE      1711 non-null   datetime64[ns]
 6   Comment_Date  1711 non-null   object
 7   Comment_Time  1711 non-null   object
dtypes: datetime64[ns](1), int64(1), object(6)
memory usage: 120.3+ KB
```

```
yt_spam4_df.groupby(yt_spam4_df["new_DATE"].dt.strftime('%B'))['CONTENT'].agg('count')
```

```
new_DATE
April         36
August        92
December      28
February      25
January       77
July          68
June          17
March         36
May          591
November     367
October      193
September    181
Name: CONTENT, dtype: int64
```

```
plt.clf()
yt_spam4_df.groupby(yt_spam4_df["new_DATE"].dt.strftime('%B'))["CONTENT"].agg('count').plot(k
plt.show()
```




```
#The above plot appears to follow a monthly pattern. Next, a trend line and monthly indicator
#...or timedeltas?

yt_spam4_df['Years'] = pd.DatetimeIndex(yt_spam4_df['new_DATE']).year
yt_spam4_df['Months'] = pd.DatetimeIndex(yt_spam4_df['new_DATE']).month
```

```
yt_spam4_df['Months'] = pd.DatetimeIndex(yt_spam4_df['new_DATE']).month
yt_spam4_df['Wday'] = pd.DatetimeIndex(yt_spam4_df['new_DATE']).weekday
yt_spam4_df.head()
```

	COMMENT_ID	AUTHOR	DATE	CONTENT	CLASS
0	LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU	Julius NM	2013-11-07T06:20:48	Huh, anyway check out this you[tube] channel: ...	1
1	LZQPQhLyRh_C2cTtd9MvFRJedxydaVW-2sNg5Diuo4A	adam riyati	2013-11-07T12:37:15	Hey guys check out my new channel and our firs...	1
2	LZQPQhLyRh9MSZYnf8djyk0gEF9BHDPYrrK-qCczIY8	Evgeny Murashkin	2013-11-08T17:34:21	just for test I have to say murdev.com	1
				me shaking my	

```
yt_spam4_df.Years.value_counts()
```

```
2014    746
2015    738
2013    227
Name: years, dtype: int64
```

```
yt_spam4_df.Months.value_counts()
```

```
5     591
11    367
10    193
9     181
8      92
1      77
7      68
4      36
3      36
12     28
2      25
6      17
Name: months, dtype: int64
```

```
yt_spam4_df.Wday.value_counts()
```

```
5    280
3    276
2    256
1    252
4    244
6    209
0    194
Name: wday, dtype: int64
```

A brief look at the date values indicate that 2014 and 2015 generated higher comments than the first year comments were collected, and that comments were higher in May and November. Comments tended to be evenly dispersed throughout the week, though slightly higher on Saturdays than Sundays.

Content Analysis

```
#As can be seen, along with comments on the video content are comments redirecting subscriber
#Redirecting comments like these are so common, in fact, that they are written in either the
#We'll addressed how they can be measured for filtering later:

yt_spam3_df.CONTENT.value_counts()
```

```
Check out this video on YouTube:
Check out this playlist on YouTube:
Shakira :-*
Hey Music Fans I really appreciate any of you who will take the time to read this, and (
Like

:) I&#39;ll subscribe to you. You look Nice :)
Song name??
Please Subscribe In My Channel →
He is good boy!!!<br />I am krean I like to eminem~!~
Nicee!!sabrosura viva https://soundcloud.com/yerki-elinmigrante/yerki-myb-move-your-body
Name: CONTENT, Length: 1559, dtype: int64
```

```
#Another kind of spam content are incoherent comment strings. A new column that measures the

yt_spam3_df['Text_Length'] = yt_spam3_df['CONTENT'].str.len()


#Finding the min & max characters amount discoverable.
#These are non-Spam comments.

yt_spam3_df[yt_spam3_df['Text_Length'] <= 2]
```

	COMMENT_ID	AUTHOR	DATE	CONTENT	CI
1821	_2viQ_Qnc6-M2Gjq_TCThUeRGBbSNsclbeFll-ETDD8	ben mashall	2013-10-02T13:42:21.938000	:)	
1825	_2viQ_Qnc6_bcubCrs8YncM7B9016OeduR9RR-	Kenji	2013-10-	:)	

```
#...and max. These are all Spam comments.

yt_spam3_df[yt_spam3_df['Text_Length'] >= 754]
```

	COMMENT_ID	AUTHOR	DATE	
303	z12cehoxozfgg3nok04cjj05xznbgrlpfjo	Elieo Cardiopulmonary	2014-11-08T15:29:52	im sorry f
333	z131idupvn3yhf3mv23dwzhi4pqixvwuw	Александр Федоров	2014-11-13T07:59:33	Look
381	z12jenlhyre0eheyx04ch1aquxfdsvgpd44	Special Pentrutine	2014-08-22T04:50:50	<script&g

```
yt_spam3_df.iloc[260,:] #an outlying comment; 754 characters of non-spam statement:
```

```
COMMENT_ID                      z133tllzkmb0wthup235c5qovo3xzdzqr04
AUTHOR                                             Wilfredo Latorre
DATE                                            2014-11-08T04:02:29
CONTENT        Hey I think I know what where dealing with her...
CLASS                                                             0
Text_Length                                                     753
Name: 260, dtype: object
```

```
#Checking the shape of the DF. Since adding the additional text-based column, the shape of th

yt_spam3_df.shape
```

```
(1711, 6)
```

Preliminary Visualizations

Investigating how your target is distributed will help you understand the relationship between the target and the features. It's also useful for discovering some potential problems with the model.

```
#The below plot shows there's slightly more Ham than Spam emails:

plt.hist(yt_spam3_df.CLASS)
plt.title("Video Content: Spam vs. Not")
plt.xlabel("Content Classification")
plt.ylabel("Content Distribution")
plt.show()
```


```
#We'll redefine the 'CLASS' variable for clarity; renaming it as:  'Spam_Detector':

yt_spam3_df.columns = yt_spam3_df.columns.str.replace('CLASS', 'Spam_Detector')
yt_spam3_df
```

	COMMENT_ID	AUTHOR	DATE	
0	LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU	Julius NM	2013-11-07T06:20:48	H ch
1	LZQPQhLyRh_C2cTtd9MvFRJedxydaVW-2sNg5Diuo4A	adam riyati	2013-11-07T12:37:15	Hey chan
2	LZQPQhLyRh9MSZYnf8djyk0gEF9BHDPYrrK-qCczIY8	Evgeny Murashkin	2013-11-08T17:34:21	ju r
3	z13jhp0bxqncu512g22wvzkasxmvvzjaz04	ElNino Melendez	2013-11-09T08:28:43	me sexy ch
4	z13fwbwp1oujthgqj04chlngpvzmtt3r3dw	GsMega	2013-11-10T16:05:38	v=vtaF Che

```
#showing the encoded column:

yt_spam3_df["Spam_Detector"].tail()
```

```
1951    0
1952    0
1953    0
1954    0
1955    0
Name: Spam_Detector, dtype: int64
```

DESCRIPTIVE STATS & COMPARISONS

```
yt_spam5_df = yt_spam4_df.copy(deep=True)

yt_spam5_df['Text_Length'] = yt_spam5_df['CONTENT'].str.len()
yt_spam5_df.columns = yt_spam5_df.columns.str.replace('CLASS', 'Spam_Detector')
```

```
#Using a combination of the datetime df and spam detector df to check the descriptive stats
yt_spam5_df.info()
```

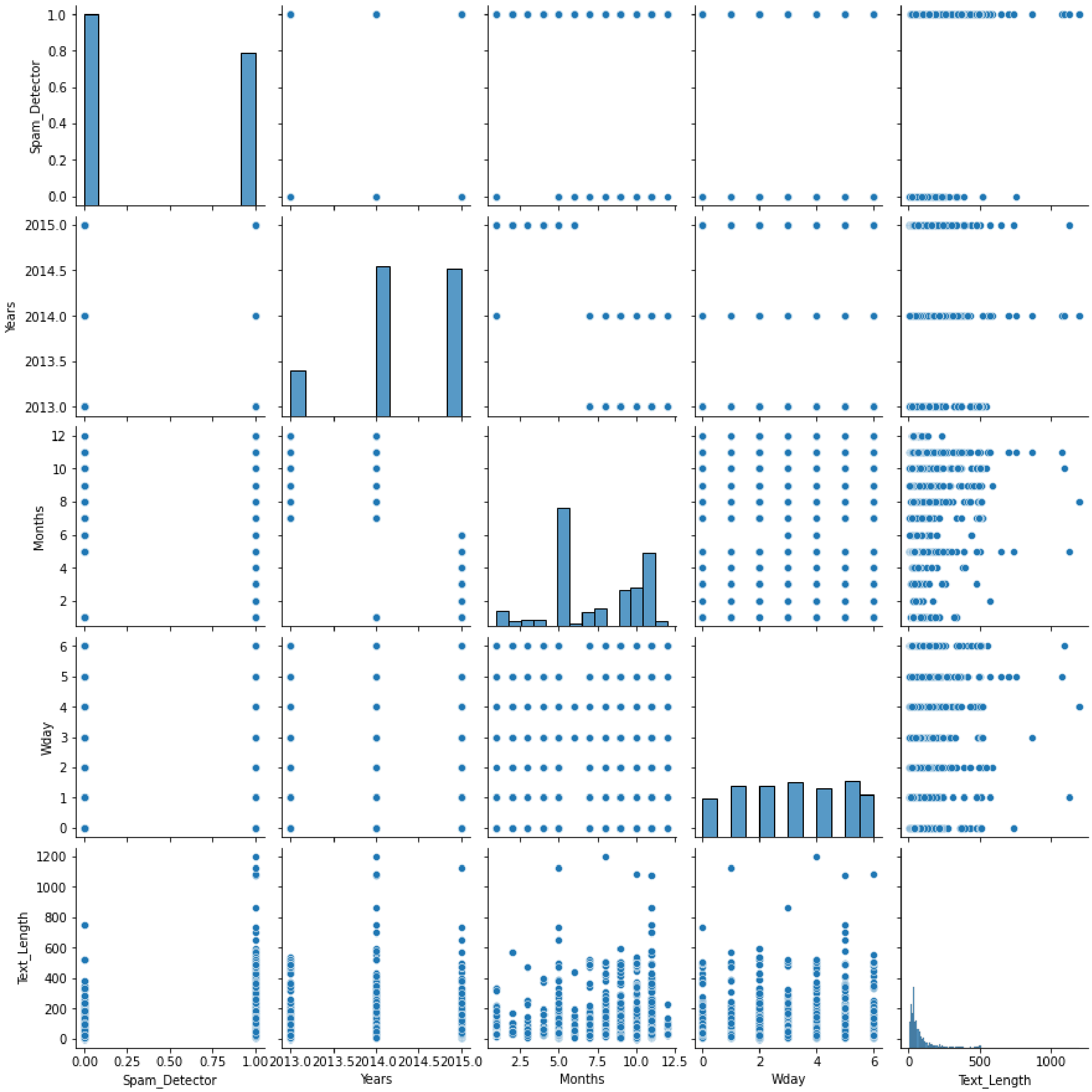
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1711 entries, 0 to 1955
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   COMMENT_ID     1711 non-null   object
 1   AUTHOR         1711 non-null   object
 2   DATE           1711 non-null   object
 3   CONTENT        1711 non-null   object
 4   Spam_Detector  1711 non-null   int64
 5   new_DATE       1711 non-null   datetime64[ns]
 6   Comment_Date   1711 non-null   object
 7   Comment_Time   1711 non-null   object
 8   Years          1711 non-null   int64
 9   Months         1711 non-null   int64
 10  Wday           1711 non-null   int64
 11  Text_Length    1711 non-null   int64
dtypes: datetime64[ns](1), int64(5), object(6)
memory usage: 173.8+ KB
```

```
yt_spam5_df.describe()
```

	Spam_Detector	Years	Months	Wday	Text_Length
count	1711.000000	1711.000000	1711.000000	1711.000000	1711.000000
mean	0.444185	2014.298656	7.352425	3.052016	83.789012
std	0.497020	0.689261	3.019477	1.911854	117.046422
min	0.000000	2013.000000	1.000000	0.000000	2.000000
25%	0.000000	2014.000000	5.000000	1.000000	26.000000
50%	0.000000	2014.000000	8.000000	3.000000	44.000000
75%	1.000000	2015.000000	10.000000	5.000000	86.000000
max	1.000000	2015.000000	12.000000	6.000000	1200.000000

```
sns.pairplot(yt_spam5_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7fc5814dd690>
```



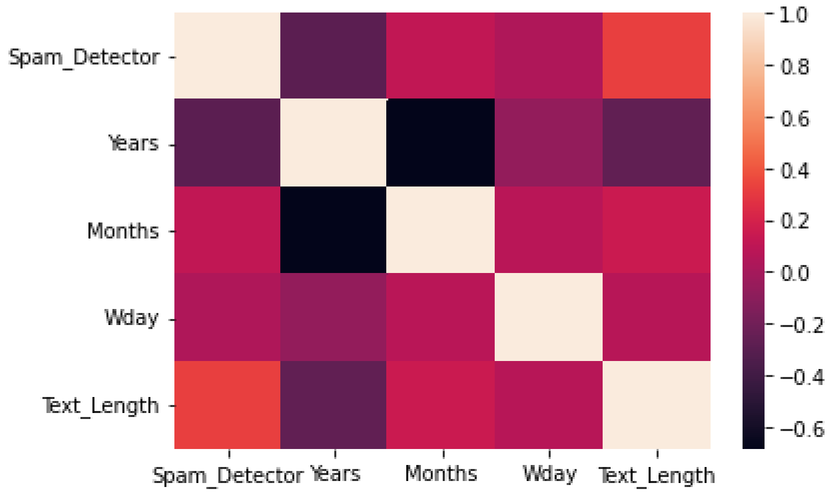

```
#There seems to be a low-moderate positive correlation between the length of video content an

yt_spam5_df.corr()
```

	Spam_Detector	Years	Months	Wday	Text_Length
Spam_Detector	1.000000	-0.290162	0.114624	0.042137	0.326417

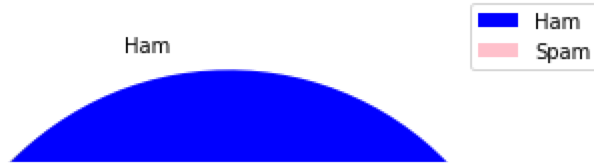
There seems to be a low-moderate positive correlation between 'Text-Length' - the amount commentors write - and whether the comment is Spam.

```
sns.heatmap(yt_spam5_df.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1b61226d90>
```



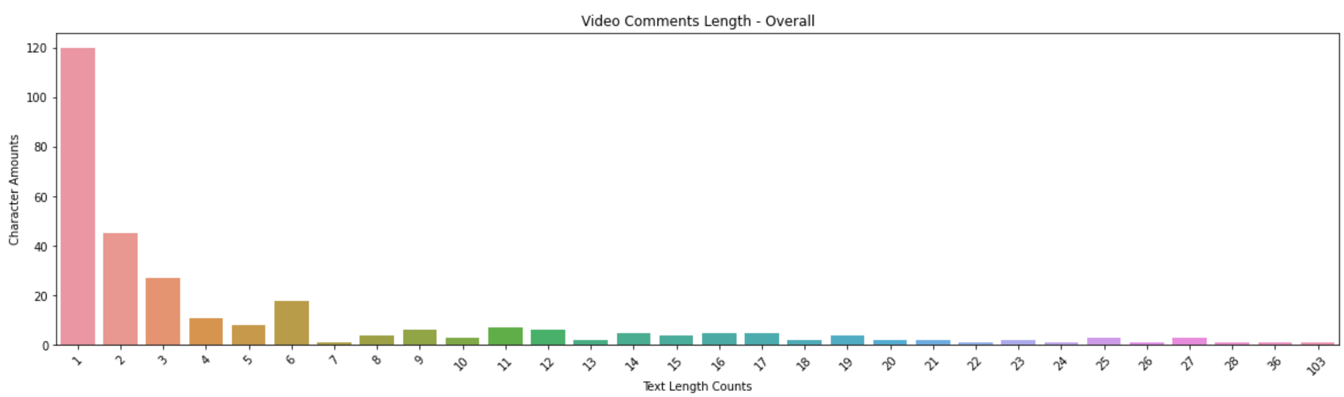

```
plt.figure(figsize=(7,7))
size=yt_spam5_df['Spam_Detector'].value_counts()
label=['Ham','Spam']
color=['Blue','Pink']
explode=[0,0.1]
plt.pie(size,explode=explode,labels=label,colors=color,shadow=True)
plt.legend()
plt.show()
```




```
#the new independent/target variable's column is still 1711 rows long:

yt_spam3_df['Spam_Detector'].shape
```

```
(1711,)
```

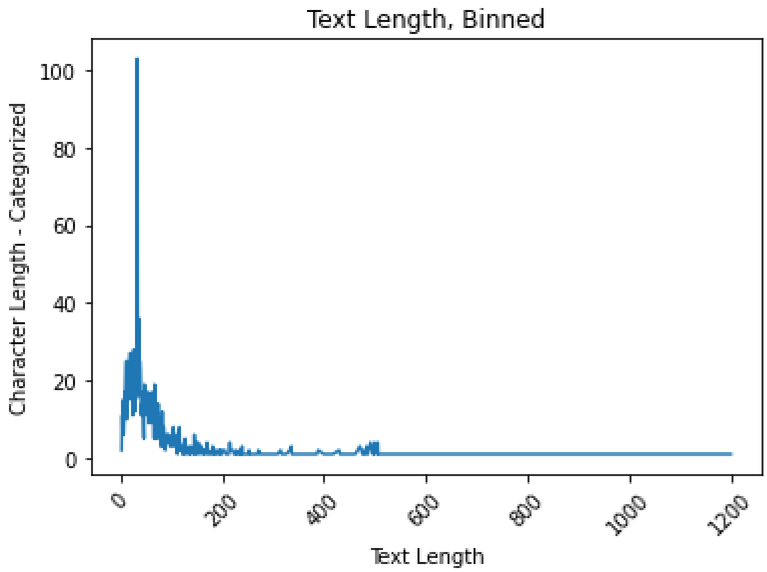
```
plt.figure(figsize=(20,5))

sns.countplot(yt_spam3_df.groupby(['Text_Length']).count()['CONTENT'])

plt.title("Video Comments Length - Overall")
plt.xlabel("Text Length Counts")
plt.ylabel("Character Amounts")
plt.xticks(rotation=45)
plt.show()
```




```
#spam5_df.groupby('text_length').sum()

df = yt_spam3_df.groupby(['Text_Length']).count()['CONTENT']
print(df)

# plot the result
df.plot()
plt.title("Text Length, Binned")
plt.xlabel("Text Length")
plt.ylabel("Character Length - Categorized")
```

```
plt.xticks(rotation=45)
plt.show()
```

```
Text_Length
2        2
3       11
4        6
5       15
6        6
        ..
866      1
1078     1
1089     1
1125     1
1200     1
Name: CONTENT, Length: 301, dtype: int64
```




```
plt.bar(yt_spam5_df['Spam_Detector'],yt_spam5_df['Text_Length'])
plt.title('Text Length over Content Distribution',fontsize=20)
plt.xlabel('Content Classification')
plt.ylabel('Character Amount')
```

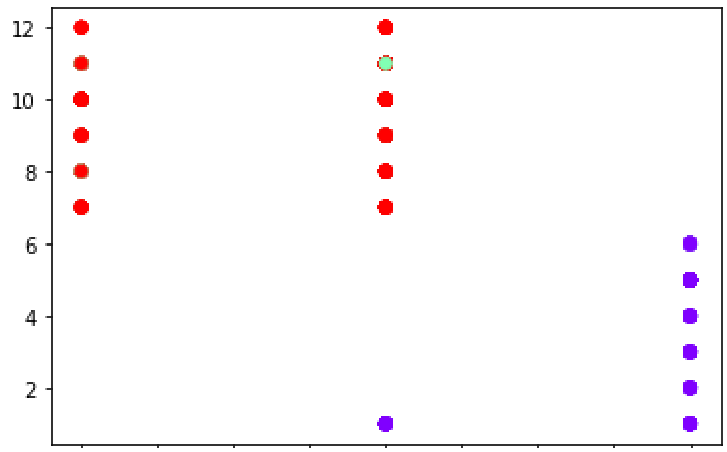
```
Text(0, 0.5, 'Character Amount')
```

Building a Cluster Model with K-Means, DBSCAN, and Hierarchical Clustering


```
# Setting the features:

col_names = ["COMMENT_ID", "AUTHOR", "DATE", "CONTENT", "Spam_Detector", "new_DATE", "Comment


X = yt_spam5_df[col_names].drop(["COMMENT_ID", "AUTHOR", "DATE", "CONTENT", "Spam_Detector",
#yt_spam4_df.drop("CLASS", axis=1).values


#STANDARDIZING THE FEATURES
scaler = StandardScaler()
X_std = scaler.fit_transform(X)



#DEFINING THE K-MEANS:

k_means_cluster = KMeans(n_clusters=3, random_state=123)

#FIT THE MODEL:
%timeit k_means_cluster.fit(X_std)
y_pred = k_means_cluster.predict(X_std)
```

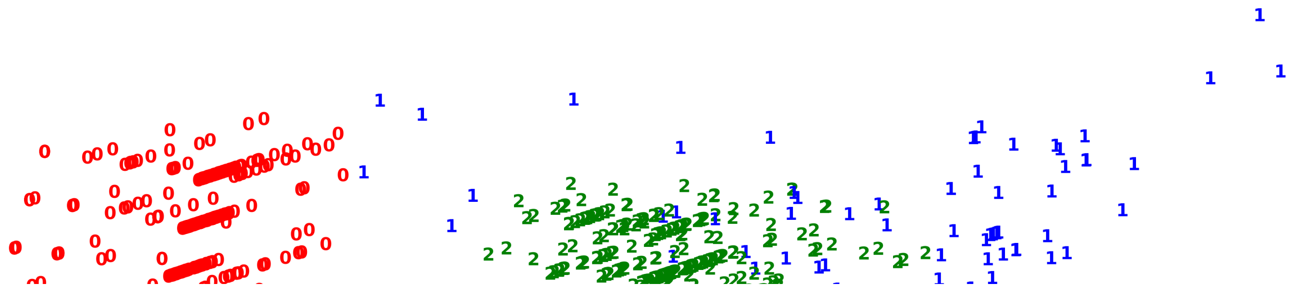
```
10 loops, best of 5: 34.4 ms per loop
```

```
# Finding the final centroids:
centroids = k_means_cluster.cluster_centers_



#NOW, APPLYING K-MEANS FOR THE SUB-SAMPLES TO GET THE PREDICTIONS, AND COMPARING RESULTS OF D

#DF to store features & predicted cluster memberships:
ypred = k_means_cluster.predict(X_std)


# Plotting the clusters:
plt.scatter(X[:, 0], X[:, 1], c=ypred, cmap= 'rainbow' )
plt.show()
```



The colors indicate three clusters with two main variables(violet and red) and an outlier(green)

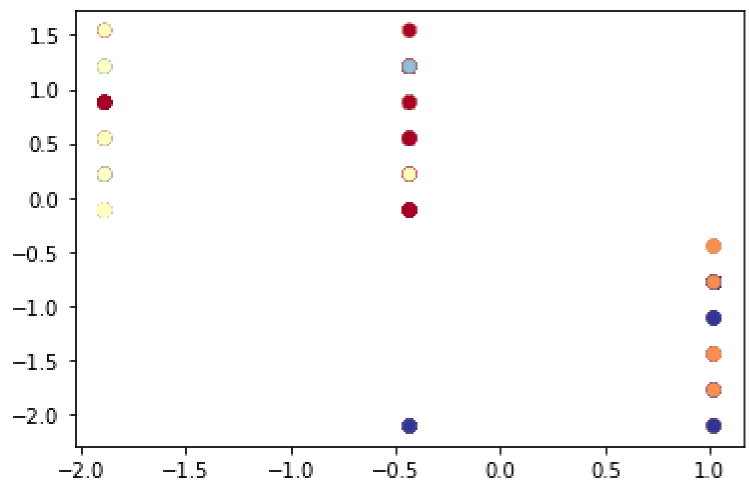
```
pca = PCA(n_components=2).fit_transform(X_std)

plt.figure(figsize=(10,5))
colours = 'rbg'
for i in range(pca.shape[0]):
  plt.text(pca[i, 0], pca[i,1], str(y_pred[i]), color=colours[y_pred[i]], fontdict={'weight':

plt.xticks([])
plt.yticks([])
plt.axis('off')
plt.show()
```



This seems to indicate the 2 major clusters - Spam and Ham content - with the numbers representing the variable separation. Three can be seen here.

```
#Applying MiniBatchMeans for comparison:

mini_batch_k_means = MiniBatchKMeans(n_clusters=5, random_state=123)
%timeit mini_batch_k_means.fit(X_std)
y_pred_mini = mini_batch_k_means.predict(X_std)

#now, plot the solution again:
plt.scatter(X_std[:,0], X_std[:,1], c = y_pred_mini, cmap = 'RdYlBu')
plt.show()
```

```
10 loops, best of 5: 22.3 ms per loop
```



When the amount of clustering was increased to differentiate possibly different kinds of videos(there are 5 artists), 5 were distiguishable, with colors orange, lilac, maroon, violet, and pale yellow.

```
labels = KMeans(n_clusters=3, random_state=123).fit_predict(X_std)
print(metrics.silhouette_score(X_std, labels, metric='euclidean'))
```

```
0.4521432048693259
```

This index score indicates that the 3-cluster solution actually produces clusters of data-points closer to other data-points in the cluster, than are to data-points in the other clusters.

```
#Getting the predicted clusters:
full_pred = KMeans(n_clusters=5, random_state=123).fit_predict(X_std)

pd.crosstab(y_pred, full_pred)
```

col_0	0	1	2	3	4
row_0					
0	479	0	0	0	290
1	1	3	1	94	2
2	0	514	327	0	0

```
def rand_index_score(ground_truths, predictions):
  tp_plus_fp = scipy.special.comb(np.bincount(predictions), 2).sum()
  tp_plus_fn = scipy.special.comb(np.bincount(ground_truths), 2).sum()
  A = np.c_[(ground_truths, predictions)]
  tp = sum(scipy.special.comb(np.bincount(A[A[:, 0] == i, 1]), 2).sum() for i in set(ground_t
  fp = tp_plus_fp - tp
  fn = tp_plus_fn - tp
  tn = scipy.special.comb(len(A), 2) - tp - fp - fn
  return (tp + tn) / (tp + fp + fn + tn)
```

```
rand_index_score(y, full_pred)
```

```
0.5179755349800568
```

```
metrics.adjusted_rand_score(y_pred, full_pred)
```

```
0.5503309560981196
```

Though ARI is pretty moderately low, as it is the corrected score, the fact that it's higher than the RI is positive.

```
#DEFINING:
dbscan_cluster = DBSCAN(eps=2, min_samples=1, metric='euclidean')

#FITTING:
clusters = dbscan_cluster.fit_predict(X_std)
```

```
model=dbscan_cluster.fit(X_std)
```

```
label=model.labels_
```

```
label
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
#applying...

#identifying the points which makes up our core points
sample_cores=np.zeros_like(label,dtype=bool)

sample_cores[dbscan_cluster.core_sample_indices_]=True

#Calculating the number of clusters

n_clusters=len(set(label))- (1 if -1 in label else 0)
print('No of clusters:',n_clusters)
```

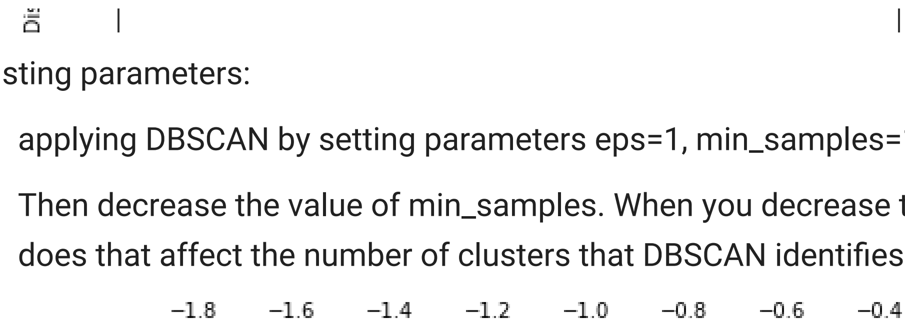
```
No of clusters: 3
```

```
y_means = dbscan_cluster.fit_predict(X_std)
plt.figure(figsize=(7,5))
plt.scatter(X_std[y_means == 0, 0], X_std[y_means == 0, 1], s = 50, c = 'pink')
plt.scatter(X_std[y_means == 1, 0], X_std[y_means == 1, 1], s = 50, c = 'yellow')
plt.scatter(X_std[y_means == 2, 0], X_std[y_means == 2, 1],
s = 50, c = 'cyan')
plt.scatter(X_std[y_means == 3, 0], X_std[y_means == 3, 1], s = 50, c = 'magenta')
plt.scatter(X_std[y_means == 4, 0], X_std[y_means == 4, 1], s = 50, c = 'orange')
plt.scatter(X_std[y_means == 5, 0], X_std[y_means == 5, 1], s = 50, c = 'blue')
plt.scatter(X_std[y_means == 6, 0], X_std[y_means == 6, 1], s = 50, c = 'red')
plt.scatter(X_std[y_means == 7, 0], X_std[y_means == 7, 1], s = 50, c = 'black')
plt.scatter(X_std[y_means == 8, 0], X_std[y_means == 8, 1], s = 50, c = 'violet')
plt.xlabel('Video Class')
plt.ylabel('Content Distribution')
plt.title('Clusters of data')
plt.show()
```

Changing the sample size definitely changed the size of the clusters. While it seemed to increase the amount of cluster variation, it also seemed to decrease their overall inter-relationships.

Adjusting parameters:

- applying DBSCAN by setting parameters eps=1, min_samples=1, metric="euclidean".
- Then decrease the value of min_samples. When you decrease the value of min_samples, how does that affect the number of clusters that DBSCAN identifies?




```
#DEFINING:
dbscan_cluster = DBSCAN(eps=1, min_samples=1, metric='euclidean')

#FITTING:
clusters = dbscan_cluster.fit_predict(X_std)


model=dbscan_cluster.fit(X_std)


label=model.labels_


label
```

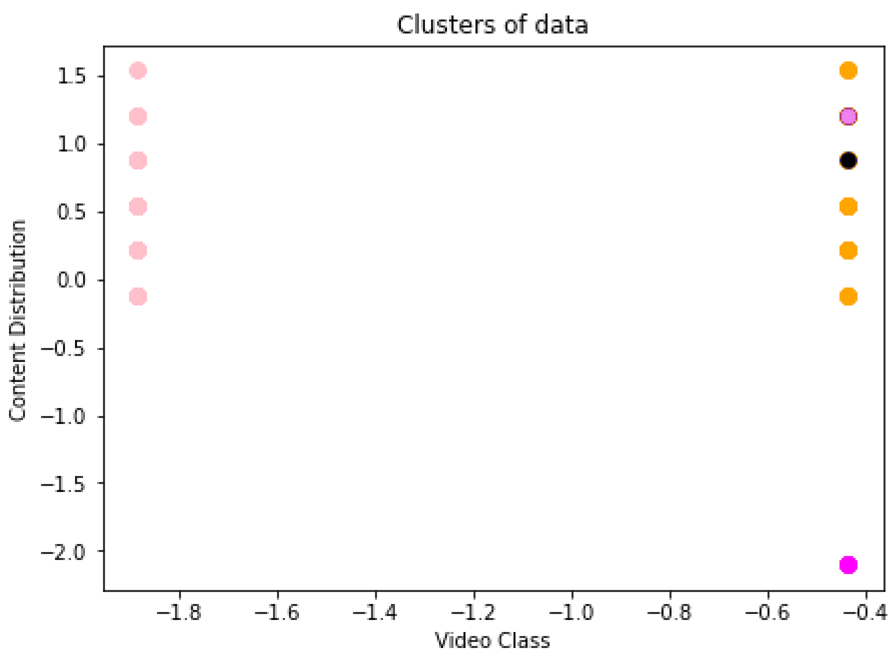
```
array([0, 0, 0, ..., 0, 0, 0])
```

```
#applying...

#identifying the points which makes up our core points
sample_cores=np.zeros_like(label,dtype=bool)

sample_cores[dbscan_cluster.core_sample_indices_]=True

#Calculating the number of clusters

n_clusters=len(set(label))- (1 if -1 in label else 0)
print('No of clusters:',n_clusters)
```

```
No of clusters: 19
```

```
y_means = dbscan_cluster.fit_predict(X_std)
plt.figure(figsize=(7,5))
plt.scatter(X_std[y_means == 0, 0], X_std[y_means == 0, 1], s = 50, c = 'pink')
```

```
plt.scatter(X_std[y_means == 1, 0], X_std[y_means == 1, 1], s = 50, c = 'yellow')
plt.scatter(X_std[y_means == 2, 0], X_std[y_means == 2, 1],
s = 50, c = 'cyan')
plt.scatter(X_std[y_means == 3, 0], X_std[y_means == 3, 1], s = 50, c = 'magenta')
plt.scatter(X_std[y_means == 4, 0], X_std[y_means == 4, 1], s = 50, c = 'orange')
plt.scatter(X_std[y_means == 5, 0], X_std[y_means == 5, 1], s = 50, c = 'blue')
plt.scatter(X_std[y_means == 6, 0], X_std[y_means == 6, 1], s = 50, c = 'red')
plt.scatter(X_std[y_means == 7, 0], X_std[y_means == 7, 1], s = 50, c = 'black')
plt.scatter(X_std[y_means == 8, 0], X_std[y_means == 8, 1], s = 50, c = 'violet')
plt.xlabel('Video Class')
plt.ylabel('Content Distribution')
plt.title('Clusters of data')
plt.show()
```



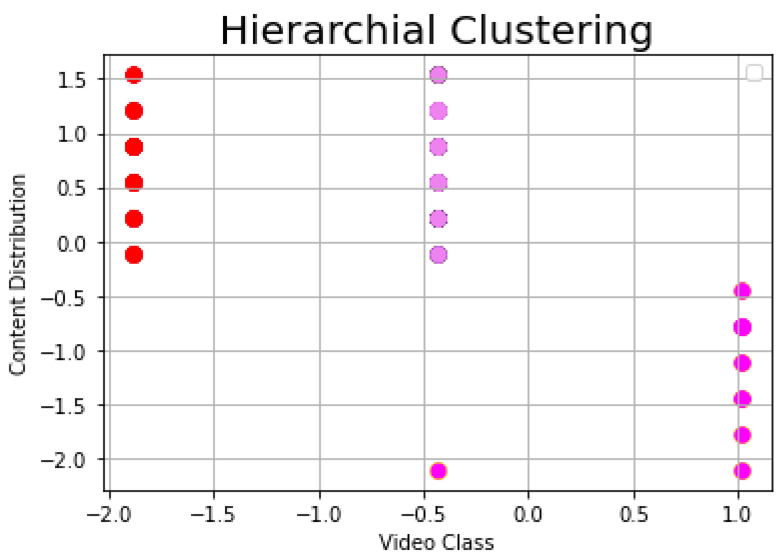
Decreasing the episodes and number of minimum samples decreases the size of the clusters while increasing the variation(the colors increased from 3 to 5). In terms of a filter, this could be used with increased input(i.e. more videos, artists).

Applying Hierachical clustering:

```
from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n_clusters = 9, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X_std)

plt.scatter(X_std[y_hc == 0, 0], X_std[y_hc == 0, 1], s = 50, c = 'pink')
plt.scatter(X_std[y_hc == 1, 0], X_std[y_hc == 1, 1], s = 50, c = 'yellow')
plt.scatter(X_std[y_hc == 2, 0], X_std[y_hc == 2, 1], s = 50, c = 'cyan')
plt.scatter(X_std[y_hc == 3, 0], X_std[y_hc == 3, 1], s = 50, c = 'magenta')
plt.scatter(X_std[y_hc == 4, 0], X_std[y_hc == 4, 1], s = 50, c = 'orange')
```

```
plt.scatter(X_std[y_hc == 5, 0], X_std[y_hc == 5, 1], s = 50, c = 'blue')
plt.scatter(X_std[y_hc == 6, 0], X_std[y_hc == 6, 1], s = 50, c = 'red')
plt.scatter(X_std[y_hc == 7, 0], X_std[y_hc == 7, 1], s = 50, c = 'black')
plt.scatter(X_std[y_hc == 8, 0], X_std[y_hc == 8, 1], s = 50, c = 'violet')


plt.title('Hierarchial Clustering', fontsize = 20)
plt.xlabel('Video Class')
plt.ylabel('Content Distribution')
plt.legend()
plt.grid()
plt.show()
```

```
No handles with labels found to put in legend.
```



Building a Clustering Model with TFIDF and KMeans

```
data = pd.read_csv('new_combinevid_files.csv')
data.head()


#"D:\Datasets\.kaggle\email2.json"
```

	COMMENT_ID	AUTHOR	DATE	CONTENT	CLASS
0	LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU	Julius NM	2013-11-07T06:20:48	Huh, anyway check out this you[tube] channel: ...	1
1	LZQPQhLyRh_C2cTtd9MvFRJedxydaVW-2sNg5Diuo4A	adam riyati	2013-11-07T12:37:15	Hey guys check out my new channel and our firs...	1

```
#Extracting Keywords:

tfidf = TfidfVectorizer(
    min_df = 5,
    max_df = 0.95,
    max_features = 8000,
    stop_words = 'english'
)
tfidf.fit(data.CONTENT)
text = tfidf.transform(data.CONTENT)


#Finding Optimal Clusters:

def find_optimal_clusters(data, max_k):
    iters = range(2, max_k+1, 2)

    sse = []
    for k in iters:
        sse.append(MiniBatchKMeans(n_clusters=k, init_size=1024, batch_size=2048, random_stat
        print('Fit {} clusters'.format(k))

    f, ax = plt.subplots(1, 1)
    ax.plot(iters, sse, marker='o')
    ax.set_xlabel('Cluster Centers')
    ax.set_xticks(iters)
    ax.set_xticklabels(iters)
    ax.set_ylabel('SSE')
    ax.set_title('SSE by Cluster Center Plot')

find_optimal_clusters(text, 20)
```

```
Fit 2 clusters
Fit 4 clusters
Fit 6 clusters
Fit 8 clusters
Fit 10 clusters
Fit 12 clusters
```

```
clusters = MiniBatchKMeans(n_clusters=14, init_size=1024, batch_size=2048, random_state=20).f
```

```
Fit 18 clusters
```

```
#Plotting Clusters:

def plot_tsne_pca(data, labels):
    max_label = max(labels)
    max_items = np.random.choice(range(data.shape[0]), size=500, replace=False)

    pca = PCA(n_components=2).fit_transform(data[max_items,:].todense())
    tsne = TSNE().fit_transform(PCA(n_components=50).fit_transform(data[max_items,:].todense(


    idx = np.random.choice(range(pca.shape[0]), size=300, replace=False)
    label_subset = labels[max_items]
    label_subset = [cm.hsv(i/max_label) for i in label_subset[idx]]

    f, ax = plt.subplots(1, 2, figsize=(14, 6))

    ax[0].scatter(pca[idx, 0], pca[idx, 1], c=label_subset)
    ax[0].set_title('PCA Cluster Plot')

    ax[1].scatter(tsne[idx, 0], tsne[idx, 1], c=label_subset)
    ax[1].set_title('TSNE Cluster Plot')

plot_tsne_pca(text, clusters)
```


```
#Getting Keywords(Spam v. Ham):

def get_top_keywords(data, clusters, labels, n_terms):
    df = pd.DataFrame(data.todense()).groupby(clusters).mean()

    for i,r in df.iterrows():
        print('\nCluster {}'.format(i))
        print(','.join([labels[t] for t in np.argsort(r)[-n_terms:]]))

get_top_keywords(text, clusters, tfidf.get_feature_names(), 10)
```

```
Cluster 0
watching,lyrics,video,videos,watch,reason,song,subscribe,comment,like

Cluster 1
facebook,music,new,fb,popular,br,look,check,video,youtube

Cluster 2
years,love,song,39,subscribe,like,check,video,youtube,br

Cluster 3
php,gofundme,pages,ref,amp,facebook,http,https,www,com

Cluster 4
katy,com,billion,music,just,video,views,39,song,check

Cluster 5
god,gofundme,going,gonna,goal,br,tried,check,youtube,playlist

Cluster 6
subscribe,song,im,comment,listening,39,thumbs,like,watching,2015

Cluster 7
eminem,come,39,fans,like,videos,plz,daily,ll,subscribe

Cluster 8
videos,hey,thanks,youtube,visit,guys,sub,check,subscribe,channel

Cluster 9
wow,check,cool,perry,day,views,39,katy,song,great

Cluster 10
songs,music,br,lmfao,eminem,shakira,cup,world,song,best

Cluster 11
lol,music,way,subscribe,lie,eminem,rihanna,shakira,song,love

Cluster 12
thing,39,girl,old,check,like,times,music,song,good

Cluster 13
like,39,mean,holy,billion,video,funny,share,views,000
```

From this method of clustering, it can be told which clusters are more likely to capture comments that contain Spam content redirecting off the site(e.g. reference to a different channel, or placing an outside link); especially Clusters 0-3.

Building NLP Models using Word Vectorization:

Displaying which words gained the most use:

```
!pip install wordcloud
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.7/dist-packages (1.5
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-packages (from wo
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.7/dist-packages (f
```

```
yt_spam3_df['CONTENT']=yt_spam3_df['CONTENT'].str.lower()
yt_spam3_df['CONTENT'].head()
```

```
0    huh, anyway check out this you[tube] channel: ...
1    hey guys check out my new channel and our firs...
2               just for test i have to say murdev.com
3     me shaking my sexy ass on my channel enjoy ^_^
4              watch?v=vtarggvgtwq   check this out .
Name: CONTENT, dtype: object
```

```
new_spam = yt_spam3_df['CONTENT'].str.split(' ')
new_spam.head()
```

```
0    [huh,, anyway, check, out, this, you[tube], ch...
1    [hey, guys, check, out, my, new, channel, and,...
2      [just, for, test, i, have, to, say, murdev.com]
3    [me, shaking, my, sexy, ass, on, my, channel, ...
4      [watch?v=vtarggvgtwq, , , check, this, out, .]
Name: CONTENT, dtype: object
```

```
import string
import collections
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

```
new_spam_cleaned = []

for content in new_spam:
    content = [x.strip(string.punctuation) for x in content]
    new_spam_cleaned.append(text)
```

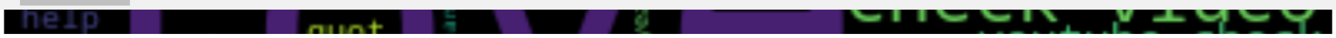
```
new_spam_cleaned[0]

content_spam = [" ".join(text) for content in new_spam_cleaned]
final_text_spam = " ".join(content_spam)
final_text_spam[:500]
```

```
'huh, anyway check out this you[tube] channel: kobyoshi02 hey guys check out my new cha
nnel and our first vid this is us the  monkeys!!! i'm the monkey in the white shirt,ple
ase leave a like comment  and please subscribe!!!! just for test i have to say murdev.c
om me shaking my sexy ass on my channel enjoy ^ ^ \ufeff watch?v=vtarggvgtwq   check th
```

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

stopwords = set(STOPWORDS)
stopwords.update(["ourselves","hers","between","yourself","but","again", "there", "about", "o

content = yt_spam3_df["CONTENT"]

# Generate a word cloud image
wordcloud = WordCloud(max_words=100, background_color="black").generate(" ".join(content))
plt.figure(figsize=(20,10))
# Display the generated image
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")

plt.show()
```


```
#A quick display of the words listed in the wordcloud:

print(wordcloud.words_.keys())
```

```
dict_keys(['song', 'love', 'please', 'video youtube', 'check video', 'br br', 'subscribe
```



```
filtered_words_spam = [word for word in final_text_spam.split() if word not in stopwords]
counted_words_spam = collections.Counter(filtered_words_spam)

word_count_spam = {}

for letter, count in counted_words_spam.most_common(30):
    word_count_spam[letter] = count

for i,j in word_count_spam.items():
        print('Word: {0}, count: {1}'.format(i,j))
```

```
Word: check, count: 533832
Word: video, count: 362732
Word: song, count: 307980
Word: love, count: 277182
Word: , count: 261783
Word: subscribe, count: 241251
Word: please, count: 237829
Word: new, count: 167678
Word: youtube:, count: 164256
Word: channel, count: 152279
Word: music, count: 147146
Word: katy, count: 124903
Word: best, count: 111215
Word: people, count: 104371
Word: money, count: 95816
Word: :), count: 94105
Word: hey, count: 90683
Word: make, count: 87261
Word: views, count: 85550
Word: good, count: 83839
Word: -, count: 82128
Word: comment, count: 80417
Word: 2, count: 80417
Word: guys, count: 78706
Word: billion, count: 76995
Word: ., count: 75284
Word: us, count: 73573
Word: really, count: 73573
Word: know, count: 73573
Word: youtube, count: 71862
```

The generator from the WordCloud is very useful in not only identifying words, but accumulating how often the most common words appeared in comments.

Of the top 10 words(check, video, song, love, subscribe, please, new, youtube, channel, music), five were associated with the first three clusters in the TFIDF/Kmeans Cluster model tied to Spam content.

```
*Creating a Natural Language Processing algori
column:*
```

Creating a Natural Language Processing algorithm to further convert the text column:

```
!pip install top2vec
```

```
Collecting top2vec
  Downloading top2vec-1.0.26-py3-none-any.whl (23 kB)
Requirement already satisfied: wordcloud in /usr/local/lib/python3.7/dist-packages (fron
Collecting hdbscan>=0.8.27
  Downloading hdbscan-0.8.27.tar.gz (6.4 MB)
     |████████████████████████████████| 6.4 MB 6.0 MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
Collecting umap-learn>=0.5.1
  Downloading umap-learn-0.5.1.tar.gz (80 kB)
     |████████████████████████████████| 80 kB 8.9 MB/s
Collecting numpy>=1.20.0
  Using cached numpy-1.21.2-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (1
Requirement already satisfied: gensim<4.0.0 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from to
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (fro
```

```
!pip install --upgrade gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (3.6.0)
Collecting gensim
  Downloading gensim-4.1.2-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (24
     |████████████████████████████████| 24.1 MB 1.6 MB/s
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-packages (
Installing collected packages: gensim
  Attempting uninstall: gensim
    Found existing installation: gensim 3.6.0
    Uninstalling gensim-3.6.0:
      Successfully uninstalled gensim-3.6.0
ERROR: pip's dependency resolver does not currently take into account all the packages t
top2vec 1.0.26 requires gensim<4.0.0, but you have gensim 4.1.2 which is incompatible.
Successfully installed gensim-4.1.2
```

```
  Building wheel for pynndescent (setup.py) ... done
```

```
!pip install python-Levenshtein
```

```
Collecting python-Levenshtein
  Downloading python-Levenshtein-0.12.2.tar.gz (50 kB)
     |████████████████████████████████| 50 kB 3.0 MB/s
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (fro
Building wheels for collected packages: python-Levenshtein
  Building wheel for python-Levenshtein (setup.py) ... done
  Created wheel for python-Levenshtein: filename=python_Levenshtein-0.12.2-cp37-cp37m-li
  Stored in directory: /root/.cache/pip/wheels/05/5f/ca/7c4367734892581bb5ff896f15027a93
Successfully built python-Levenshtein
Installing collected packages: python-Levenshtein
Successfully installed python-Levenshtein-0.12.2
```

```
from gensim.test.utils import common_texts
from gensim.models import Word2Vec
```

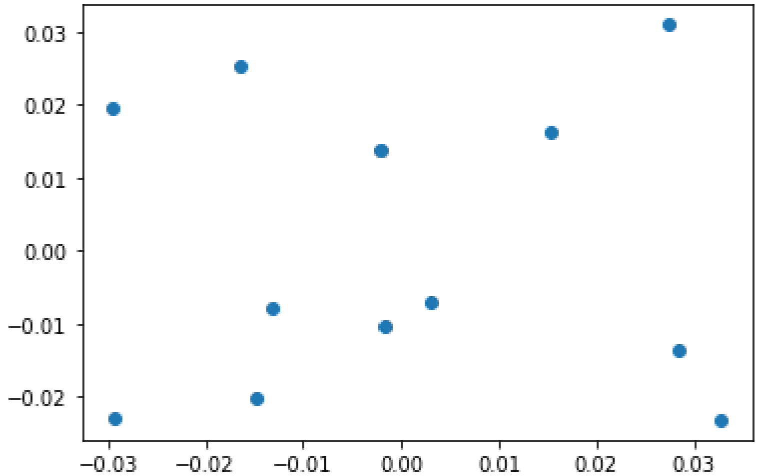
```
model = Word2Vec(sentences=common_texts, vector_size=100, window=5, min_count=1, workers=4)
model.save("word2vec.model")


# define training data
sentences = [['EVERYONE PLEASE GO SUBSCRIBE TO MY CHANNEL OR JUST LOON AT MY VIDEOS', '+44793
             ['That is megan fox', 'no where near one of eminems actual best songs, real fans


# train model
#X = model[model.wv.vocab]
model = Word2Vec(sentences, min_count=1)

# summarize the loaded model
print(model)

# summarize vocabulary
words = list(model.wv.key_to_index)
print(words)

# access vector for one word
#print(model.wv.word)
#vec += model_w2v.wv.word.reshape((1, size))

# save model
model.save('model.bin')

# load model
new_model = Word2Vec.load('model.bin')
print(new_model)
```

```
Word2Vec(vocab=12, vector_size=100, alpha=0.025)
['I love it.', 'In my head this is like 2 years ago.. Time FLIES', 'Who df is Lauren Ber
Word2Vec(vocab=12, vector_size=100, alpha=0.025)
```

```
#retrieving all of the vectors from a trained modeL:

X = model.wv[model.wv.key_to_index]


#creating a 2-dimensional PCA model of the word vectors using the scikit-learn PCA class:
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
result = pca.fit_transform(X)


#plotting the projection using matplotlib, pulling out the two dimensions as x and y coordina
from matplotlib import pyplot
```

```
pyplot.scatter(result[:, 0], result[:, 1])
```

```
<matplotlib.collections.PathCollection at 0x7fc56bc458d0>
```




```
#Creating a scatter plot with the dots annotated with the Sentences:

# define training data

sentences = [['EVERYONE PLEASE GO SUBSCRIBE TO MY CHANNEL OR JUST LOON AT MY VIDEOS', '+44793
             ['That is megan fox', 'no where near one of eminems actual best songs, real fans


# train model
model = Word2Vec(sentences, min_count=1)
# fit a 2d PCA model to the vectors
X = model.wv[model.wv.key_to_index]
pca = PCA(n_components=2)
result = pca.fit_transform(X)
# create a scatter plot of the projection
pyplot.scatter(result[:, 0], result[:, 1])
words = list(model.wv.key_to_index)
for i, word in enumerate(words):
  pyplot.annotate(word, xy=(result[i, 0], result[i, 1]))
pyplot.show()
```


```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (3.2.5)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from nltk)
```

PADDING WITH ZERO(formatting the text_length column w/ zero-padding):

```
# importing pandas
import pandas as pd

# making data frame from csv at url
data = pd.read_csv('new_combinevid_files.csv')

# converting to string dtype
data["CONTENT"] = yt_spam3_df["CONTENT"].astype(str)

# width of output string
width = 755

# calling method and overwriting series
data["CONTENT"] = yt_spam3_df["CONTENT"].str.zfill(width)

# display
data
```

	COMMENT_ID	AUTHOR	DATE	
0	LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU	Julius NM	2013-11-07T06:20:48	00000
1	LZQPQhLyRh_C2cTtd9MvFRJedxydaVW-2sNg5Diuo4A	adam riyati	2013-11-07T12:37:15	00000
2	LZQPQhLyRh9MSZYnf8djyk0gEF9BHDPYrrK-qCczIY8	Evgeny Murashkin	2013-11-08T17:34:21	00000
3	z13jhp0bxqncu512g22wvzkasxmvvzjaz04	ElNino Melendez	2013-11-09T08:28:43	00000
4	z13fwbwp1oujthgqj04chlngpvzmtt3r3dw	GsMega	2013-11-10T16:05:38	00000
...	...	...	...	
1951	_2viQ_Qnc6-bMSjqyL1NKj57ROicCSJV5SwTrw-RFFA	Katie Mettam	2013-07-13T13:27:39.441000	00000
1952	_2viQ_Qnc6-pY-1yR6K2FhmC5i48-WuNx5CumlHLDAI	Sabina Pearson-Smith	2013-07-13T13:14:30.021000	00000

Word Vectorization

```
X_train, X_test, y_train, y_test = train_test_split(yt_spam3_df['CONTENT'], yt_spam3_df['Spam

# training the vectorizer

vectorizer = TfidfVectorizer()

X_train = vectorizer.fit_transform(X_train)
```

```
from sklearn import svm
svm = svm.SVC(C=1000)
svm.fit(X_train, y_train)
```

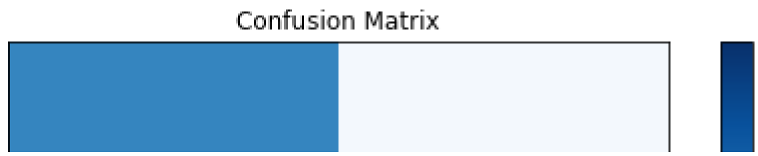
```
SVC(C=1000, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
from sklearn.metrics import confusion_matrix
X_test = vectorizer.transform(X_test)
y_pred = svm.predict(X_test)

confusion = confusion_matrix(y_test, y_pred, labels=[1, 0])
print(confusion)
```

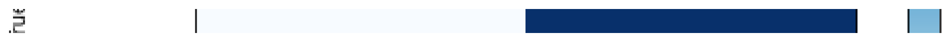
```
[[67  4]
 [ 2 99]]
```

```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):

    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
```

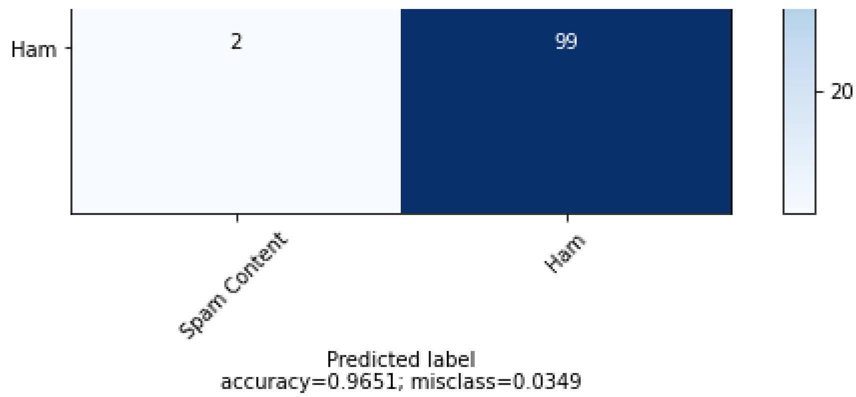
```
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclas
    plt.show()


plot_confusion_matrix(cm=confusion, target_names = ['Spam Content', 'Ham'], title = 'Confusio
```



- Actually/predicted to be Ham: 99
- Actually/predicted to be Spam: 67
- Predicted Spam/mistaken for Ham: 4
- Predicted Ham/mistaken for Spam: 2



This is a rather good outcome for Word Vectorization.



✓ 0s completed at 7:45 AM