Data Mining Lab 1

In this lab session we will focus on the use of scientific computing libraries to efficiently process, transform, and manage data. Furthermore, we will provide best practices and introduce visualization tools for effectively conducting big data analysis and visualization.

Table of Contents

Introduction

In this notebook I will explore a text-based, document-based dataset using scientific computing tools such as Pandas and Numpy. In addition, several fundamental Data Mining concepts will be explored and explained in details, ranging from calculating distance measures to computing term frequency vectors. Coding examples, visualizations and demonstrations will be provided where necessary. Furthermore, additional exercises are provided after special topics. These exercises are geared towards testing the proficiency of students and motivate students to explore beyond the techniques covered in the notebook.

Requirements

Here are the computing and software requirements

Computing Resources

- Operating system: Preferably Linux or MacOS
- RAM: 8 GB
- Disk space: Mininium 8 GB

Software Requirements

Here is a list of the required programs and libraries necessary for this lab session:

Language:

- Python 3+ (Note: coding will be done strictly on Python 3)
  - We are using Python 3.9.6.
  - You can use newer version, but use at your own risk.

Environment:

Using an environment is to avoid some library conflict problems. You can refer this Setup Instructions to install and setup.

- Anaconda (recommended but not required)
  - Install anaconda environment
- Python virtualenv (recommended to Linux/MacOS user)
  - Install virtual environment
- Kaggle Kernel
  - Run on the cloud (with some limitations)
  - Reference: Kaggle Kernels Instructions

Necessary Libraries:

- Jupyter (Strongly recommended but not required)
    - Install `jupyter` and Use `$jupyter notebook` in terminal to run
- Scikit Learn
    - Install `sklearn` latest python library
- Pandas
    - Install `pandas` python library
- Numpy
    - Install `numpy` python library
- Matplotlib
    - Install `maplotlib` for python
- Plotly
    - Install and signup for `plotly`
- Seaborn
    - Install and signup for `seaborn`
- NLTK
    - Install `nltk` library

```
# TEST necessary for when working with external scripts
%load_ext autoreload
%autoreload 2
```

```
    The autoreload extension is already loaded. To reload it, use:
      %reload_ext autoreload
```

1. The Data

In this notebook we will explore the popular 20 newsgroup dataset, originally provided here. The dataset is called "Twenty Newsgroups", which means there are 20 categories of news articles available in the entire dataset. A short description of the dataset, provided by the authors, is provided below:

- *The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. To the best of our knowledge, it was originally collected by Ken Lang, probably for his paper "Newsweeder: Learning to filter netnews," though he does not explicitly mention this collection. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.*

If you need more information about the dataset please refer to the reference provided above. Below is a snapshot of the dataset already converted into a table. Keep in mind that the original dataset is not in this nice pretty format. That work is left to us. That is one of the tasks that will be covered in this notebook: how to convert raw data into convenient tabular formats using Pandas.

pic1.png

---

2. Data Preparation

In the following we will use the built-in dataset loader for 20 newsgroups from scikit-learn. Alternatively, it is possible to download the dataset manually from the website and use the sklearn.datasets.load_files function by pointing it to the 20news-bydate-train sub-folder of the uncompressed archive folder.

In order to get faster execution times for this first example we will work on a partial dataset with only 4 categories out of the 20 available in the dataset:

```
# categories
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med'


# obtain the documents containing the categories provided
from sklearn.datasets import fetch_20newsgroups

twenty_train = fetch_20newsgroups(subset='train', categories=categories,
                                  shuffle=True, random_state=42)
```

Let's take at look some of the records that are contained in our subset of the data

```
twenty_train.data[0:2]
```

```
    ['From: sd345@city.ac.uk (Michael Collier)\nSubject: Converting images to
    HP LaserJet III?\nNntp-Posting-Host: hampton\nOrganization: The City
    University\nLines: 14\n\nDoes anyone know of a good way (standard PC
    application/PD utility) to\nconvert tif/img/tga files into LaserJet III
    format.  We would also like to\ndo the same, converting to HPGL (HP
    plotter) files.\n\nPlease email any response.\n\nIs this the correct group?
    \n\nThanks in advance.  Michael.\n-- \nMichael Collier (Programmer)
    The Computer Unit,\nEmail: M.P.Collier@uk.ac.city                 The City
    University,\nTel: 071 477-8000 x3769                        London,\nFax: 071
    477-8565                                 EC1V 0HB.\n',
     "From: ani@ms.uky.edu (Aniruddha B. Deglurkar)\nSubject: help: Splitting a
    trimming region along a mesh \nOrganization: University Of Kentucky, Dept.
    of Math Sciences\nLines: 28\n\n\n\n\tHi,\n\n\tI have a problem, I hope some
    of the 'gurus' can help me solve.\n\n\tBackground of the problem:\n\tI have
    a rectangular mesh in the uv domain, i.e  the mesh is a \n\tmapping of a 3d
    Bezier patch into 2d. The area in this domain\n\twhich is inside a trimming
    loop had to be rendered. The trimming\n\tloop is a set of 2d Bezier curve
    segments.\n\tFor the sake of notation: the mesh is made up of
    cells.\n\n\tMy problem is this :\n\tThe trimming area has to be split up
    into individual smaller\n\tcells bounded by the trimming curve segments. If
    a cell\n\tis wholly inside the area...then it is output as a whole
    ,\n\telse it is trivially rejected. \n\n\tDoes any body know how thiss can
    be done, or is there any algo. \n\tsomewhere for doing this.\n\n\tAny help
    would be appreciated.\n\n\tThanks, \n\tAni.\n-- \nTo get irritated is
    human, to stay cool, divine.\n"]
```

Note the `twenty_train` is just a bunch of objects that can be accessed as python dictionaries; so, you can do the following operations on `twenty_train`

```
twenty_train.target_names
```

```
    ['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
```

```
len(twenty_train.data)
```

```
    2257
```

```
len(twenty_train.filenames)
```

```
    2257
```

We can also print an example from the subset

```
# An example of what the subset contains
print("\n".join(twenty_train.data[0].split("\n")))
```

```
    From: sd345@city.ac.uk (Michael Collier)
    Subject: Converting images to HP LaserJet III?
    Nntp-Posting-Host: hampton
    Organization: The City University
    Lines: 14

    Does anyone know of a good way (standard PC application/PD utility) to
    convert tif/img/tga files into LaserJet III format.  We would also like to
    do the same, converting to HPGL (HP plotter) files.

    Please email any response.

    Is this the correct group?

    Thanks in advance.  Michael.
    --
    Michael Collier (Programmer)                 The Computer Unit,
    Email: M.P.Collier@uk.ac.city                The City University,
    Tel: 071 477-8000 x3769                      London,
    Fax: 071 477-8565                                EC1V 0HB.
```

... and determine the label of the example via `target_names` key value

```
print(twenty_train.target_names[twenty_train.target[0]])
```

```
    comp.graphics
```

```
twenty_train.target[0]
```

```
    1
```

... we can also get the category of 10 documents via `target` key value

```
# category of first 10 documents.
twenty_train.target[:10]
```

```
    array([1, 1, 3, 3, 3, 3, 3, 2, 2, 2])
```

Note: As you can observe, both approaches above provide two different ways of obtaining the `category` value for the dataset. Ideally, we want to have access to both types -- numerical and nominal -- in the event some particular library favors a particular type.

As you may have already noticed as well, there is no **tabular format** for the current version of the data. As data miners, we are interested in having our dataset in the most convenient format as possible; something we can manipulate easily and is compatible with our algorithms, and so forth.

Here is one way to get access to the *text* version of the label of a subset of our training data:

```
for t in twenty_train.target[:10]:
    print(twenty_train.target_names[t])
```

```
    comp.graphics
    comp.graphics
    soc.religion.christian
    soc.religion.christian
    soc.religion.christian
    soc.religion.christian
    soc.religion.christian
    sci.med
    sci.med
    sci.med
```

** >>> Exercise 1 (5 min): **

In this exercise, please print out the *text* data for the first three samples in the dataset. (See the above code for help)

[ ] ↳ *2 個隱藏的儲藏格*

3. Data Transformation

So we want to explore and understand our data a little bit better. Before we do that we definitely need to apply some transformations just so we can have our dataset in a nice format to be able to explore it freely and more efficient. Lucky for us, there are powerful scientific tools to transform our data into that tabular format we are so farmiliar with. So that is what we will do in the next section--transform our data into a nice table format.

3.1 Converting Dictionary into Pandas Dataframe

Here we will show you how to convert dictionary objects into a pandas dataframe. And by the way, a pandas dataframe is nothing more than a table magically stored for efficient information retrieval.

```
import pandas as pd

# my functions
import helpers.data_mining_helpers as dmh

# construct dataframe from a list
X = pd.DataFrame.from_records(dmh.format_rows(twenty_train), columns= ['text'])
```

```
len(X)
```

```
    2257
```

```
X[0:2]
```

	text
0	From: sd345@city.ac.uk (Michael Collier) Subje...
1	From: ani@ms.uky.edu (Aniruddha B. Deglurkar) ...

```
for t in X["text"][:3]:
    print(t)
```

```
From: sd345@city.ac.uk (Michael Collier) Subject: Converting images to HP La
From: ani@ms.uky.edu (Aniruddha B. Deglurkar) Subject: help: Splitting a tr
From: djohnson@cs.ucsd.edu (Darin Johnson) Subject: Re: harrassed at work, 
```

Adding Columns

One of the great advantages of a pandas dataframe is its flexibility. We can add columns to the current dataset programmatically with very little effort.

```
# add category to the dataframe
X['category'] = twenty_train.target
```

```
# add category label also
X['category_name'] = X.category.apply(lambda t: dmh.format_labels(t, twenty_trair
```

Now we can print and see what our table looks like.

```
X[0:10]
```

	text	category	category_name
0	From: sd345@city.ac.uk (Michael Collier) Subje...	1	comp.graphics
1	From: ani@ms.uky.edu (Aniruddha B. Deglurkar) ...	1	comp.graphics
2	From: djohnson@cs.ucsd.edu (Darin Johnson) Sub...	3	soc.religion.christian
3	From: s0612596@let.rug.nl (M.M. Zwart) Subject...	3	soc.religion.christian
4	From: stanly@grok11.columbiasc.ncr.com (stanly...	3	soc.religion.christian
5	From: vbv@lor.eeap.cwru.edu (Virgilio (Dean) B...	3	soc.religion.christian
6	From: jodfishe@silver.ucs.indiana.edu (joseph ...	3	soc.religion.christian
7	From: aldridge@netcom.com (Jacquelin Aldridge)...	2	sci.med
8	From: geb@cs.pitt.edu (Gordon Banks) Subject: ...	2	sci.med
9	From: libman@hsc.usc.edu (Marlena Libman) Subj...	2	sci.med

Nice! Isn't it? With this format we can conduct many operations easily and efficiently since Pandas dataframes provide us with a wide range of built-in features/functionalities. These features are operations which can directly and quickly be applied to the dataset. These operations may include standard operations like **removing records with missing values** and **aggregating new fields** to the current table (hereinafter referred to as a dataframe), which is desirable in almost every data mining project. Go Pandas!

3.2 Familiarizing yourself with the Data

[ ] ↳ 9 個隱藏的儲藏格

** >>> Exercise 2 (take home):**

Experiment with other querying techniques using pandas dataframes. Refer to their documentation for more information.

```
#Answer here

# using iloc (by position)
print("X.iloc[:10, 0]: \n" , X.iloc[:10, 0], "\n")

print("X[:5]\n")
display(X[:5])
print("\n")

#Backwards Query
print("X[:2250:-1]\n")
display(X[:2250:-1])
print("\n")
```

```
X.iloc[:10, 0]:
 0    From: sd345@city.ac.uk (Michael Collier) Subje...
1    From: ani@ms.uky.edu (Aniruddha B. Deglurkar) ...
2    From: djohnson@cs.ucsd.edu (Darin Johnson) Sub...
3    From: s0612596@let.rug.nl (M.M. Zwart) Subject...
4    From: stanly@grok11.columbiasc.ncr.com (stanly...
5    From: vbv@lor.eeap.cwru.edu (Virgilio (Dean) B...
6    From: jodfishe@silver.ucs.indiana.edu (joseph ...
7    From: aldridge@netcom.com (Jacquelin Aldridge)...
8    From: geb@cs.pitt.edu (Gordon Banks) Subject: ...
9    From: libman@hsc.usc.edu (Marlena Libman) Subj...
Name: text, dtype: object

X[:5]
```

	text	category	category_name

```
# Pass a list of columns to [] to select columns in that order.
X[['text', 'category']] = X[['category', 'text']]
print("After Switching Value: \n")
display(X)

X[['text', 'category']] = X[['category', 'text']]
print("Switch Back: \n")
display(X)
```

```
After Switching Value:
```

	text	category	category_name
0	1	From: sd345@city.ac.uk (Michael Collier) Subje...	comp.graphics
1	1	From: ani@ms.uky.edu (Aniruddha B. Deglurkar) ...	comp.graphics
2	3	From: djohnson@cs.ucsd.edu (Darin Johnson) Sub...	soc.religion.christian
3	3	From: s0612596@let.rug.nl (M.M. Zwart) Subject...	soc.religion.christian
4	3	From: stanly@grok11.columbiasc.ncr.com (stanly...	soc.religion.christian
...	...	...	...
2252	2	From: roos@Operoni.Helsinki.FI (Christophe Roo...	sci.med
2253	2	From: mhollowa@ic.sunysb.edu (Michael Holloway...	sci.med
2254	2	From: sasghm@theseus.unx.sas.com (Gary Merrill...	sci.med
2255	2	From: Dan Wallach <dwallach@cs.berkeley.edu> S...	sci.med

```
#Select by label

Y = pd.DataFrame.from_records(dmh.format_rows(twenty_train),
                   columns=['text'], index = pd.date_range('1/1/2000', periods=22

#select the row from '2000-02-25' to the end
print("1. Select the row from '2000-02-25' to the end: \n")
display(Y.loc['2000-02-25':])
print("\n")

#select the row '2000-02-25' and '2000-03-27' (Add one more bracket inside)
Y.loc[['2000-02-25', '2000-03-27']]

print("2. Select the row '2000-02-25' and '2000-03-27': \n")
display(Y.loc[['2000-02-25', '2000-03-27']])
print("\n")
```

```
1. Select the row from '2000-02-25' to the end:
```

	text
2000-02-25	From: ken@cs.UAlberta.CA (Huisman Kenneth M) S...
2000-02-26	From: kaminski@netcom.com (Peter Kaminski) Sub...
2000-02-27	From: mauaf@csv.warwick.ac.uk (Mr P D Simmons)...
2000-02-28	From: timmbake@mcl.ucsb.edu (Bake Timmons) Sub...
2000-02-29	From: joachim@kih.no (joachim lous) Subject: R...
...	...
2006-03-02	From: roos@Operoni.Helsinki.FI (Christophe Roo...
2006-03-03	From: mhollowa@ic.sunysb.edu (Michael Holloway...
2006-03-04	From: sasghm@theseus.unx.sas.com (Gary Merrill

按兩下 (或按 Enter 鍵) 即可編輯

```
#Selection by Callable
Z = pd.DataFrame.from_records(dmh.format_rows(twenty_train),
                   columns=['text'])
# add category to the dataframe
Z['category'] = twenty_train.target
# add category label also
Z['category_name'] = Z.category.apply(lambda t: dmh.format_labels(t, twenty_trai

#index can be a callable

print("Index can be a callable: \n")
display(Z.loc[lambda date: date['category'] > 2 , :])
print("\n")
```

```
Index can be a callable:
```

	text	category	category_name
2	From: djohnson@cs.ucsd.edu (Darin Johnson) Sub...	3	soc.religion.christian
3	From: s0612596@let.rug.nl (M.M. Zwart) Subject...	3	soc.religion.christian

```
#Combining positional and label-based indexing


#The row from 0 to 2
print("The row from 0 to 2: \n")
display(Z.loc[0:2, ['text', 'category_name']])
print("\n")

##The row 0 and 2

print("The row 0 and 2: \n")
display(Z.loc[[0,2], ['text', 'category_name']])
print("\n")
```

```
The row from 0 to 2:
```

	text	category_name
0	From: sd345@city.ac.uk (Michael Collier) Subje...	comp.graphics
1	From: ani@ms.uky.edu (Aniruddha B. Deglurkar) ...	comp.graphics
2	From: djohnson@cs.ucsd.edu (Darin Johnson) Sub...	soc.religion.christian

```
The row 0 and 2:
```

	text	category_name
0	From: sd345@city.ac.uk (Michael Collier) Subje...	comp.graphics
2	From: djohnson@cs.ucsd.edu (Darin Johnson) Sub...	soc.religion.christian

---

** >>> Exercise 3 (5 min): **

Try to fetch records belonging to the `sci.med` category, and query every 10th record. Only show the first 5 records.

```
# Answer here
```

4. Data Mining using Pandas

Let's do some serious work now. Let's learn to program some of the ideas and concepts learned so far in the data mining course. This is the only way we can be convince ourselves of the true power of Pandas dataframes.

4.1 Missing Values

[ ] ↳ *7 個隱藏的儲藏格*

>>> **Exercise 4 (5 min):**

Let's try something different. Instead of calculating missing values by column let's try to calculate the missing values in every record instead of every column.
$Hint$: `axis` parameter. Check the documentation for more information.

[ ] ↳ *23 個隱藏的儲藏格*

>>> **Exercise 5 (take home)**

There is an old saying that goes, "The devil is in the details." When we are working with extremely large data, it's difficult to check records one by one (as we have been doing so far). And also, we don't even know what kind of missing values we are facing. Thus, "debugging" skills get sharper as we spend more time solving bugs. Let's focus on a different method to check for missing values and the kinds of missing values you may encounter. It's not easy to check for missing values as you will find out in a minute.

Please check the data and the process below, describe what you observe and why it happened.
$Hint$: why `.isnull()` didn't work?

```
import numpy as np

NA_dict = [{ 'id': 'A', 'missing_example': np.nan },
           { 'id': 'B'                          },
           { 'id': 'C', 'missing_example': 'NaN'  },
           { 'id': 'D', 'missing_example': 'None' },
           { 'id': 'E', 'missing_example':  None  },
           { 'id': 'F', 'missing_example': ''     }]

NA_df = pd.DataFrame(NA_dict, columns = ['id','missing_example'])
NA_df
```

	id	missing_example
0	A	NaN
1	B	NaN
2	C	NaN
3	D	None
4	E	None
5	F	

```
NA_df['missing_example'].isnull()
```

```
    0     True
    1     True
    2    False
    3    False
    4     True
    5    False
    Name: missing_example, dtype: bool
```

Answer:

From my observation, the rows which can be identified by the function `.isnull()` has the regular format of expressing NaN. For example: `np.nan`, `None` or not even input any information. However, in reality we cannot expect how the dataset creator to inputing their data. To efficiently find those missing values, such as inputting the NaN or None as a string type, or even typing a space. We can first know the type of the column, and filter the NaN data by checking the data type in each input. If the column type is string, we can first create a list of "possible NaN inputs" to efficiently find out the hidden missing data.

```
# Answer here
# Import Regular Expressions lib
import re
# Using regex to find the NA values
# first separate the NaN data and other datas
NA_df.missing_example.apply(lambda x: len(re.findall('NaN|[x|?|!|#|-|]|None|', x
```

```
    0    True
    1    True
    2    True
    3    True
    4    True
    5    True
    Name: missing_example, dtype: bool
```

4.2 Dealing with Duplicate Data

Dealing with duplicate data is just as painful as dealing with missing data. The worst case is that you have duplicate data that has missing values. But let us not get carried away. Let us stick with the basics. As we have learned in our Data Mining course, duplicate data can occur because of many reasons. The majority of the times it has to do with how we store data or how we collect and merge data. For instance, we may have collected and stored a tweet, and a retweet of that same tweet as two different records; this results in a case of data duplication; the only difference being that one is the original tweet and the other the retweeted one. Here you will learn that dealing with duplicate data is not as challenging as missing values. But this also all depends on what you consider as duplicate data, i.e., this all depends on your criteria for what is considered as a duplicate record and also what type of data you are dealing with. For textual data, it may not be so trivial as it is for numerical values or images. Anyhow, let us look at some code on how to deal with duplicate records in our `X` dataframe.

First, let us check how many duplicates we have in our current dataset. Here is the line of code that checks for duplicates; it is very similar to the `isnull` function that we used to check for missing values.

```
X.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
2252    False
2253    False
2254    False
2255    False
2256    False
Length: 2257, dtype: bool
```

We can also check the sum of duplicate records by simply doing:

```
sum(X.duplicated())
```

```
0
```

Based on that output, you may be asking why did the `duplicated` operation only returned one single column that indicates whether there is a duplicate record or not. So yes, all the `duplicated()` operation does is to check per records instead of per column. That is why the operation only returns one value instead of three values for each column. It appears that we don't have any duplicates since none of our records resulted in `True`. If we want to check for duplicates as we did above for some particular column, instead of all columns, we do something as shown below. As you may have noticed, in the case where we select some columns instead of checking by all columns, we are kind of lowering the criteria of what is considered as a duplicate record. So let us only check for duplicates by only checking the `text` attribute.

```
sum(X.duplicated('text'))
```

```
    0
```

Now let us create some duplicated dummy records and append it to the main dataframe `X`. Subsequenlty, let us try to get rid of the duplicates.

```
dummy_duplicate_dict = [{
                             'text': 'dummy record',
                             'category': 1,
                             'category_name': "dummy category"
                        },
                        {
                             'text': 'dummy record',
                             'category': 1,
                             'category_name': "dummy category"
                        }]
```

```
X = pd.concat([X, pd.DataFrame(dummy_duplicate_dict)], ignore_index=True)
```

```
len(X)
```

```
    2259
```

```
sum(X.duplicated('text'))
```

```
    1
```

We have added the dummy duplicates to `X`. Now we are faced with the decision as to what to do with the duplicated records after we have found it. In our case, we want to get rid of all the duplicated records without preserving a copy. We can simply do that with the following line of code:

```
X.drop_duplicates(keep=False, inplace=True) # inplace applies changes directly or
```

```
len(X)
```

```
    2257
```

Check out the Pandas documentation for more information on dealing with duplicate data.

---

5. Data Preprocessing

In the Data Mining course we learned about the many ways of performing data preprocessing. In reality, the list is quiet general as the specifics of what data preprocessing involves is too much to cover in one course. This is especially true when you are dealing with unstructured data, as we are dealing with in this particular notebook. But let us look at some examples for each data preprocessing technique that we learned in the class. We will cover each item one by one, and provide example code for each category. You will learn how to perform each of the operations, using Pandas, that cover the essentials to Preprocessing in Data Mining. We are not going to follow any strict order, but the items we will cover in the preprocessing section of this notebook are as follows:

- Aggregation
- Sampling
- Dimensionality Reduction
- Feature Subset Selection
- Feature Creation
- Discretization and Binarization
- Attribute Transformation

5.1 Sampling

The first concept that we are going to cover from the above list is sampling. Sampling refers to the technique used for selecting data. The functionalities that we use to selected data through queries provided by Pandas are actually basic methods for sampling. The reasons for sampling are sometimes due to the size of data -- we want a smaller subset of the data that is still representatitive enough as compared to the original dataset.

We don't have a problem of size in our current dataset since it is just a couple thousand records long. But if we pay attention to how much content is included in the `text` field of each of those records, you will realize that sampling may not be a bad idea after all. In fact, we have already done some sampling by just reducing the records we are using here in this notebook; remember that we are only using four categories from the all the 20 categories available. Let us get an idea on how to sample using pandas operations.

```
len(X)
```

```
    2257
```

```
X_sample = X.sample(n=1000) #random state
```

```
len(X_sample)
```

```
    1000
```

```
X_sample[0:4]
```

	text	category	category_name
614	From: pharvey@quack.kfu.com (Paul Harvey) Subj...	3	soc.religion.christian
1943	From: ccgwt@trentu.ca (Grant Totten) Subject: ...	1	comp.graphics
1018	From: mangoe@cs.umd.edu (Charley Wingate) Subj...	3	soc.religion.christian
842	From: thester@nyx.cs.du.edu (Uncle Fester) Sub...	1	comp.graphics

>>> Exercise 6 (take home):

Notice any changes to the `X` dataframe? What are they? Report every change you noticed as compared to the previous state of `X`. Feel free to query and look more closely at the dataframe for these changes.

```
# Answer here
print("X length : " , len(X),"\n")
display(X[:15])
print("\n")
print("X_sample length : ", len(X_sample), "\n")
display(X_sample[:15])
```

```
X length :  2257
```

	text	category	category_name
0	From: sd345@city.ac.uk (Michael Collier) Subje...	1	comp.graphics
1	From: ani@ms.uky.edu (Aniruddha B. Deglurkar) ...	1	comp.graphics
2	From: djohnson@cs.ucsd.edu (Darin Johnson) Sub...	3	soc.religion.christian
3	From: s0612596@let.rug.nl (M.M. Zwart) Subject...	3	soc.religion.christian
4	From: stanly@grok11.columbiasc.ncr.com (stanly...	3	soc.religion.christian
5	From: vbv@lor.eeap.cwru.edu (Virgilio (Dean) B...	3	soc.religion.christian
6	From: jodfishe@silver.ucs.indiana.edu (joseph ...	3	soc.religion.christian
7	From: aldridge@netcom.com (Jacquelin Aldridge)...	2	sci.med
8	From: geb@cs.pitt.edu (Gordon Banks) Subject: ...	2	sci.med
9	From: libman@hsc.usc.edu (Marlena Libman) Subj...	2	sci.med
10	From: anasaz!karl@anasazi.com (Karl Dussik) Su...	3	soc.religion.christian
11	From: amjad@eng.umd.edu (Amjad A Soomro) Subje...	1	comp.graphics
12	From: I3150101@dbstu1.rz.tu-bs.de (Benedikt Ro...	0	alt.atheism
13	Subject: So what is Maddi? From: madhaus@netco...	0	alt.atheism
14	From: sloan@cis.uab.edu (Kenneth Sloan) Subjec...	1	comp.graphics

```
X_sample length :  1000
```

	text	category	category_name
614	From: pharvey@quack.kfu.com (Paul Harvey) Subj...	3	soc.religion.christian
1943	From: ccgwt@trentu.ca (Grant Totten) Subject: ...	1	comp.graphics
1018	From: mangoe@cs.umd.edu (Charley Wingate) Subj...	3	soc.religion.christian
842	From: thester@nyx.cs.du.edu (Uncle Fester) Sub...	1	comp.graphics
1538	From: morgan@socs.uts.edu.au Subject: re: tech...	3	soc.religion.christian

Comparison:

Compared to the X dataframe and X_sample dataframe, we can know the size is different(2257/ 1000). Furthermore, the X_sample is sampled in random order instead of taking the first 1000th data from X dataset.

509	From: kshin@stein.u.washington.edu (Kevin Shin...	1	comp.graphics

Let's do something cool here while we are working with sampling! Let us look at the distribution of categories in both the sample and original dataset. Let us visualize and analyze the disparity between the two datasets. To generate some visualizations, we are going to use `matplotlib` python library. With matplotlib, things are faster and compatability-wise it may just be the best visualization library for visualizing content extracted from dataframes and when using Jupyter notebooks. Let's take a loot at the magic of `matplotlib` below.

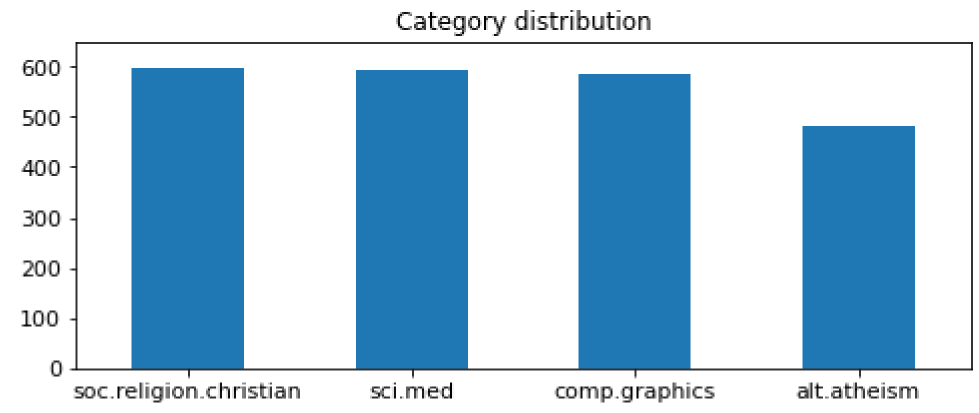
```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
categories
```

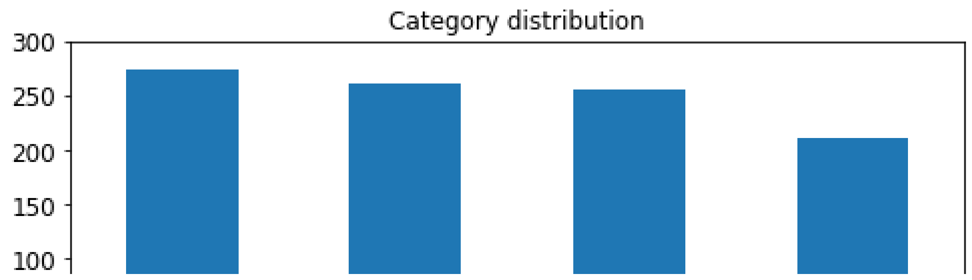
```
['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
```

```
print(X.category_name.value_counts())

# plot barchart for X
X.category_name.value_counts().plot(kind = 'bar',
                                    title = 'Category distribution',
                                    ylim = [0, 650],
                                    rot = 0, fontsize = 11, figsize = (8,3))
```

```
soc.religion.christian    599
sci.med                   594
comp.graphics             584
alt.atheism               480
Name: category_name, dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7efd7ce61210>
```



```
print(X_sample.category_name.value_counts())

# plot barchart for X_sample
X_sample.category_name.value_counts().plot(kind = 'bar',
                                           title = 'Category distribution',
                                           ylim = [0, 300],
                                           rot = 0, fontsize = 12, figsize = (8,
soc.religion.christian    274
sci.med                   260
comp.graphics             255
alt.atheism               211
Name: category_name, dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7efd7cc87210>
```



You can use following command to see other available styles to prettify your charts. `python print(plt.style.available)`

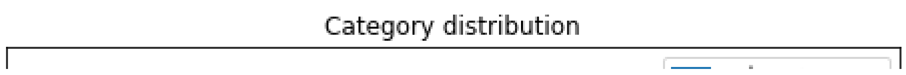
>>> Exercise 7 (5 min):

Notice that for the `ylim` parameters we hardcoded the maximum value for y. Is it possible to automate this instead of hard-coding it? How would you go about doing that? (Hint: look at code above for clues)

[ ] ↳ *2 個隱藏的儲藏格*

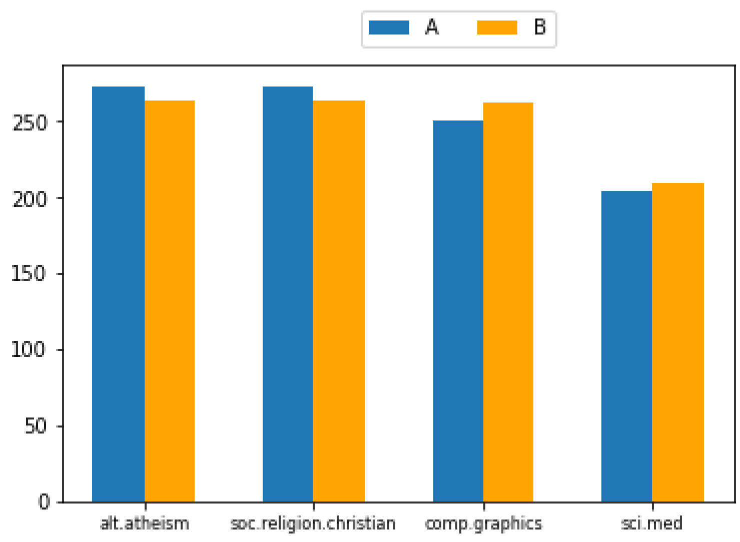
>>> Exercise 8 (take home):

We can also do a side-by-side comparison of the distribution between the two datasets, but maybe you can try that as an excerise. Below we show you an snapshot of the type of chart we are looking for.



```
# Answer here
import matplotlib.pyplot as plt
%matplotlib inline

# Create two dataset from X
A = X[:1000]
B = X[1000:2000]

print("A:")
print(A.category_name.value_counts())
print("\n")
print("B:")
print(B.category_name.value_counts())
print("\n")
bar = np.arange(len(categories))
width = 0.3
plt.bar(bar, A.category_name.value_counts(), width,  label='A')
plt.bar(bar+width, B.category_name.value_counts(), width, color='orange', label='
plt.legend(bbox_to_anchor =(0.75, 1.15), ncol = 2)
plt.xticks(bar + width / 2, categories)
plt.xticks(fontsize = 8)
plt.show()
```

```
A:
soc.religion.christian    273
sci.med                   273
comp.graphics             250
alt.atheism               204
Name: category_name, dtype: int64


B:
soc.religion.christian    264
comp.graphics             264
sci.med                   262
alt.atheism               210
Name: category_name, dtype: int64
```



按兩下 (或按 Enter 鍵) 即可編輯

One thing that stood out from the both datasets, is that the distribution of the categories remain relatively the same, which is a good sign for us data scientist. There are many ways to conduct sampling on the dataset and still obtain a representative enough dataset. That is not the main focus in this notebook, but if you would like to know more about sampling and how the `sample` feature works, just reference the Pandas documentation and you will find interesting ways to conduct more advanced sampling.

5.2 Feature Creation

The other operation from the list above that we are going to practise on is the so-called feature creation. As the name suggests, in feature creation we are looking at creating new interesting and useful features from the original dataset; a feature which captures the most important information from the raw information we already have access to. In our `X` table, we would like to create some features from the `text` field, but we are still not sure what kind of features we want to create. We can think of an interesting problem we want to solve, or something we want to analyze from the data, or some questions we want to answer. This is one process to come up with features -- this process is usually called `feature engineering` in the data science community.

We know what feature creation is so let us get real involved with our dataset and make it more interesting by adding some special features or attributes if you will. First, we are going to obtain the **unigrams** for each text. (Unigram is just a fancy word we use in Text Mining which stands for 'tokens' or 'individual words'.) Yes, we want to extract all the words found in each text and append it as a new feature to the pandas dataframe. The reason for extracting unigrams is not so clear yet, but we can start to think of obtaining some statistics about the articles we have: something like **word distribution** or **word frequency**.

Before going into any further coding, we will also introduce a useful text mining library called NLTK. The NLTK library is a natural language processing tool used for text mining tasks, so might as well we start to familiarize ourselves with it from now (It may come in handy for the final project!). In partcular, we are going to use the NLTK library to conduct tokenization because we are interested in splitting a sentence into its individual components, which we refer to as words, emojis, emails, etc. So let us go for it! We can call the `nltk` library as follows:

```
import nltk
import nltk
nltk.download('punkt')
```

```
    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Unzipping tokenizers/punkt.zip.
    True
```

```
# takes a like a minute or two to process
X['unigrams'] = X['text'].apply(lambda x: dmh.tokenize_text(x))
```

```
X[0:4]["unigrams"]
```

```
    0    [From, :, sd345, @, city.ac.uk, (, Michael, Co...
    1    [From, :, ani, @, ms.uky.edu, (, Aniruddha, B....
    2    [From, :, djohnson, @, cs.ucsd.edu, (, Darin, ...
    3    [From, :, s0612596, @, let.rug.nl, (, M.M, ., ...
    Name: unigrams, dtype: object
```

If you take a closer look at the `X` table now, you will see the new columns `unigrams` that we have added. You will notice that it contains an array of tokens, which were extracted from the original `text` field. At first glance, you will notice that the tokenizer is not doing a great job, let us take a closer at a single record and see what was the exact result of the tokenization using the `nltk` library.

```
X[0:4]
```

	text	category	category_name	unigrams
0	From: sd345@city.ac.uk (Michael Collier) Subje...	1	comp.graphics	[From, :, sd345, @, city.ac.uk, (, Michael, Co...
1	From: ani@ms.uky.edu (Aniruddha B. Deglurkar) ...	1	comp.graphics	[From, :, ani, @, ms.uky.edu, (, Aniruddha, B....

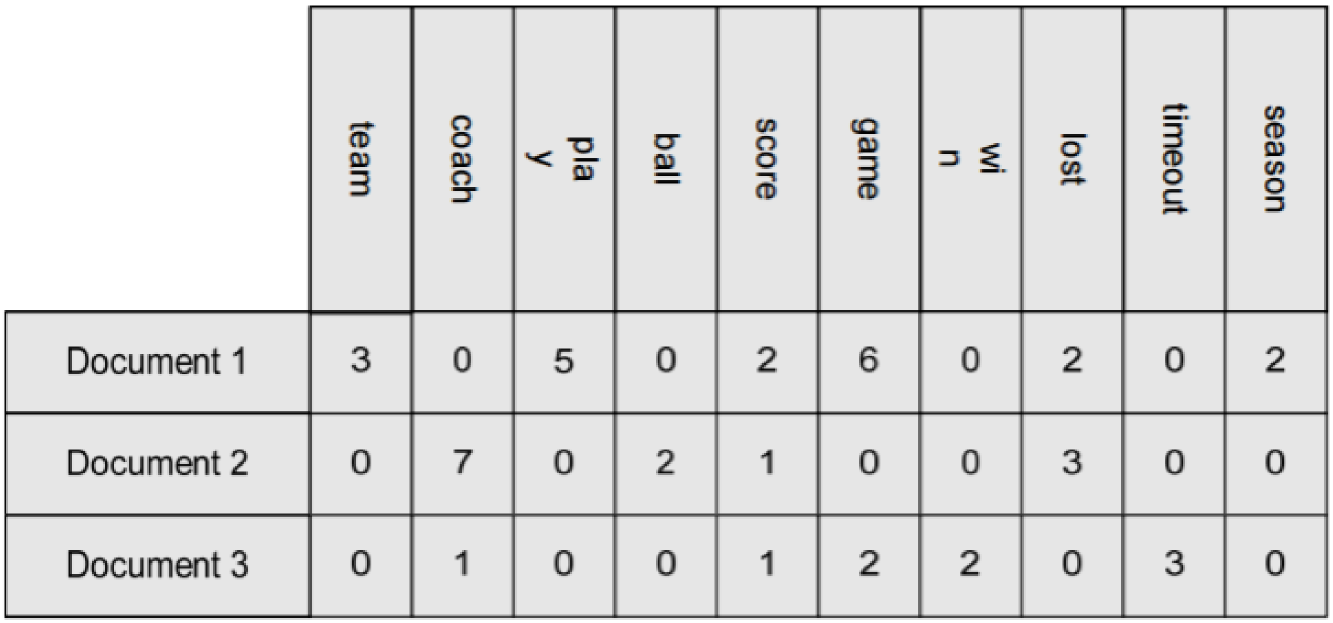
```
list(X[0:1]['unigrams'])
```

```
    [['From',
      ':',
      'sd345',
      '@',
      'city.ac.uk',
      '(',
      'Michael',
      'Collier',
      ')',
      'Subject',
      ':',
      'Converting',
      'images',
      'to',
      'HP',
      'LaserJet',
      'III',
      '?',
      'Nntp-Posting-Host',
      ':',
      'hampton',
      'Organization',
      ':',
      'The',
      'City',
      'University',
      'Lines',
      ':',
      '14',
      'Does',
      'anyone',
      'know',
      'of',
      'a',
      'good',
      'way',
      '(',
      'standard',
      'PC',
      'application/PD',
      'utility',
      ')',
      'to',
      'convert',
      'tif/img/tga',
      'files',
      'into',
      'LaserJet',
      'III',
      'format',
      '.',
      'We',
      'would',
      'also',
      'like',
      'to',
      'do',
      'the'
```

The `nltk` library does a pretty decent job of tokenizing our text. There are many other tokenizers online, such as spaCy, and the built in libraries provided by scikit-learn. We are making use of the NLTK library because it is open source and because it does a good job of segmentating text-based data.

5.3 Feature subset selection

Okay, so we are making some headway here. Let us now make things a bit more interesting. We are going to do something different from what we have been doing thus far. We are going use a bit of everything that we have learned so far. Briefly speaking, we are going to move away from our main dataset (one form of feature subset selection), and we are going to generate a document-term matrix from the original dataset. In other words we are going to be creating something like this.



	team	coach	play	ball	score	game	win	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0

Initially, it won't have the same shape as the table above, but we will get into that later. For now, let us use scikit learn built in functionalities to generate this document. You will see for yourself how easy it is to generate this table without much coding.

```
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()
X_counts = count_vect.fit_transform(X.text) #learn the vocabulary and return docu
print(X_counts[0])

      (0, 14887)    1
      (0, 29022)    1
      (0, 8696)     4
      (0, 4017)     2
      (0, 33256)    2
      (0, 21661)    3
      (0, 9031)     3
      (0, 31077)    1
      (0, 9805)     2
      (0, 17366)    1
      (0, 32493)    4
      (0, 16916)    2
      (0, 19780)    2
```

```
  (0, 17302)	2
  (0, 23122)	1
  (0, 25663)	1
  (0, 16881)	1
  (0, 16082)	1
  (0, 23915)	1
  (0, 32142)	5
  (0, 33597)	2
  (0, 20253)	1
  (0, 587)	1
  (0, 12051)	1
  (0, 5201)	1
  :	:
  (0, 25361)	1
  (0, 25337)	1
  (0, 12833)	2
  (0, 5195)	1
  (0, 27836)	1
  (0, 18474)	1
  (0, 32270)	1
  (0, 9932)	1
  (0, 15837)	1
  (0, 32135)	1
  (0, 17556)	1
  (0, 4378)	1
  (0, 26175)	1
  (0, 9338)	1
  (0, 33572)	1
  (0, 31915)	1
  (0, 177)	2
  (0, 2326)	2
  (0, 3062)	1
  (0, 35416)	1
  (0, 20459)	1
  (0, 14085)	1
  (0, 3166)	1
  (0, 12541)	1
  (0, 230)	1
```

What we did with those two lines of code is that we transformed the articles into a **term-document matrix**. Those lines of code tokenize each article using a built-in, default tokenizer (often referred to as an `analzyer`) and then produces the word frequency vector for each document. We can create our own analyzers or even use the nltk analyzer that we previously built. To keep things tidy and minimal we are going to use the default analyzer provided by `CountVectorizer`. Let us look closely at this analyzer.

```
analyze = count_vect.build_analyzer()
analyze("I am craving for a hawaiian pizza right now")
```

```
    ['am', 'craving', 'for', 'hawaiian', 'pizza', 'right', 'now']
```

>>> Exercise 9 (5 min):

Let's analyze the first record of our X dataframe with the new analyzer we have just built. Go ahead try it!

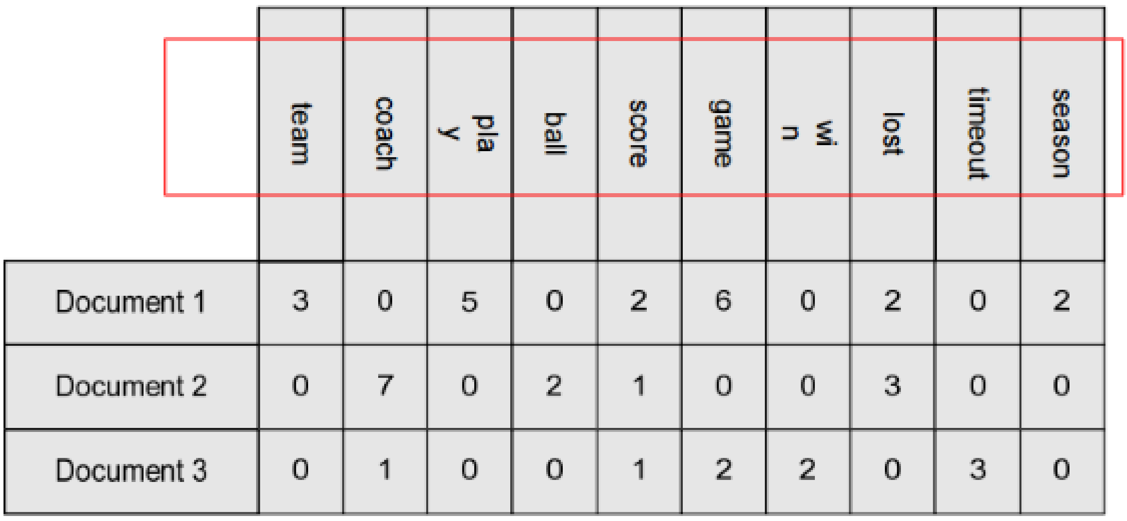
Now let us look at the term-document matrix we built above.

```
# We can check the shape of this matrix by:
X_counts.shape
```

```
    (2257, 35788)
```

```
# We can obtain the feature names of the vectorizer, i.e., the terms
# usually on the horizontal axis
count_vect.get_feature_names_out()[0:10]
```

```
    array(['00', '000', '0000', '0000001200', '000005102000', '0001',
           '000100255pixel', '00014', '000406', '0007'], dtype=object)
```



	team	coach	play	ball	score	game	win	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0

Above we can see the features found in the all the documents `X`, which are basically all the terms found in all the documents. As I said earlier, the transformation is not in the pretty format (table) we saw above -- the term-document matrix. We can do many things with the `count_vect` vectorizer and its transformation `X_counts`. You can find more information on other cool stuff you can do with the CountVectorizer.

Now let us try to obtain something that is as close to the pretty table I provided above. Before jumping into the code for doing just that, it is important to mention that the reason for choosing the `fit_transform` for the `CountVectorizer` is that it efficiently learns the vocabulary dictionary and returns a term-document matrix.

In the next bit of code, we want to extract the first five articles and transform them into document-term matrix, or in this case a 2-dimensional array. Here it goes.

```
print(X_counts.shape)
```

```
    (2257, 35788)
```

```
# we convert from sparse array to normal array
X_counts[0:5, 0:100].toarray()
```

```
    array([[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, 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, 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, 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],
           [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, 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],
           [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, 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]])
```

```
count_vect.get_feature_names_out()[0:1]
```

```
    array(['00'], dtype=object)
```

As you can see the result is just this huge sparse matrix, which is computationally intensive to generate and difficult to visualize. But we can see that the fifth record, specifically, contains a `1` in the beginning, which from our feature names we can deduce that this article contains exactly one `00` term.

>>> Exercise 10 (take home):

We said that the `1` at the beginning of the fifth record represents the `00` term. Notice that there is another 1 in the same record. Can you provide code that can verify what word this 1 represents from the vocabulary. Try to do this as efficient as possible.

Answer:

```
# Answer here
print(X_counts[0:5, 0:100])
```

```
      (4, 0)        1
      (4, 37)       1
```

We print out the X_counts to know that the location (4, 37) contains a one as well.

The `inverse_transform(X)` function return terms per document with nonzero entries in X. (X is an array)

```
count_vect.inverse_transform(X_counts[0:5, 0:100].toarray())
```

```
    [array([], dtype='<U80'),
     array([], dtype='<U80'),
     array([], dtype='<U80'),
     array([], dtype='<U80'),
     array(['00', '01'], dtype='<U80')]
```

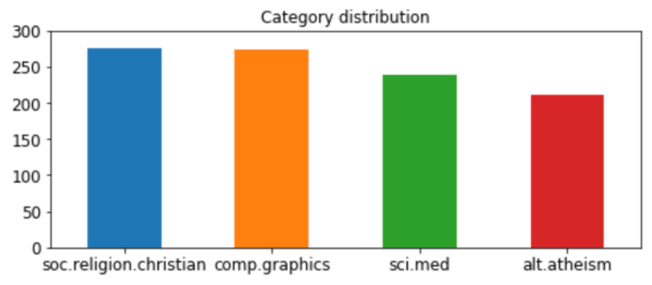
Therefore, we can know that beside ’00’ , ’01’ also contains a 1.

To get you started in thinking about how to better analyze your data or transformation, let us look at this nice little heat map of our term-document matrix. It may come as a surpise to see the gems you can mine when you start to look at the data from a different perspective. Visualization are good for this reason.

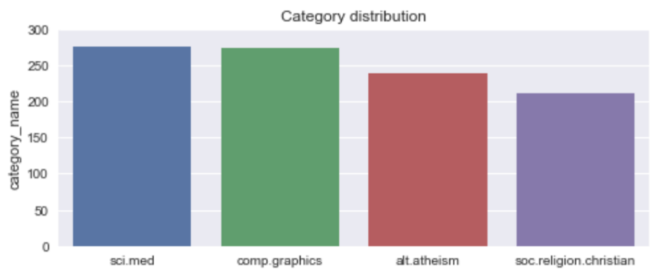
```
# first twenty features only
plot_x = ["term_"+str(i) for i in count_vect.get_feature_names_out()[0:20]]

# obtain document index
plot_y = ["doc_"+ str(i) for i in list(X.index)[0:20]]

plot_z = X_counts[0:20, 0:20].toarray()
plot_z
```

```
array([[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, 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],
       [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, 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, 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, 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]])
```

For the heat map, we are going to use another visualization library called `seaborn`. It's built on top of matplotlib and closely integrated with pandas data structures. One of the biggest advantages of seaborn is that its default aesthetics are much more visually appealing than matplotlib. See comparison below.



By Matplotlib

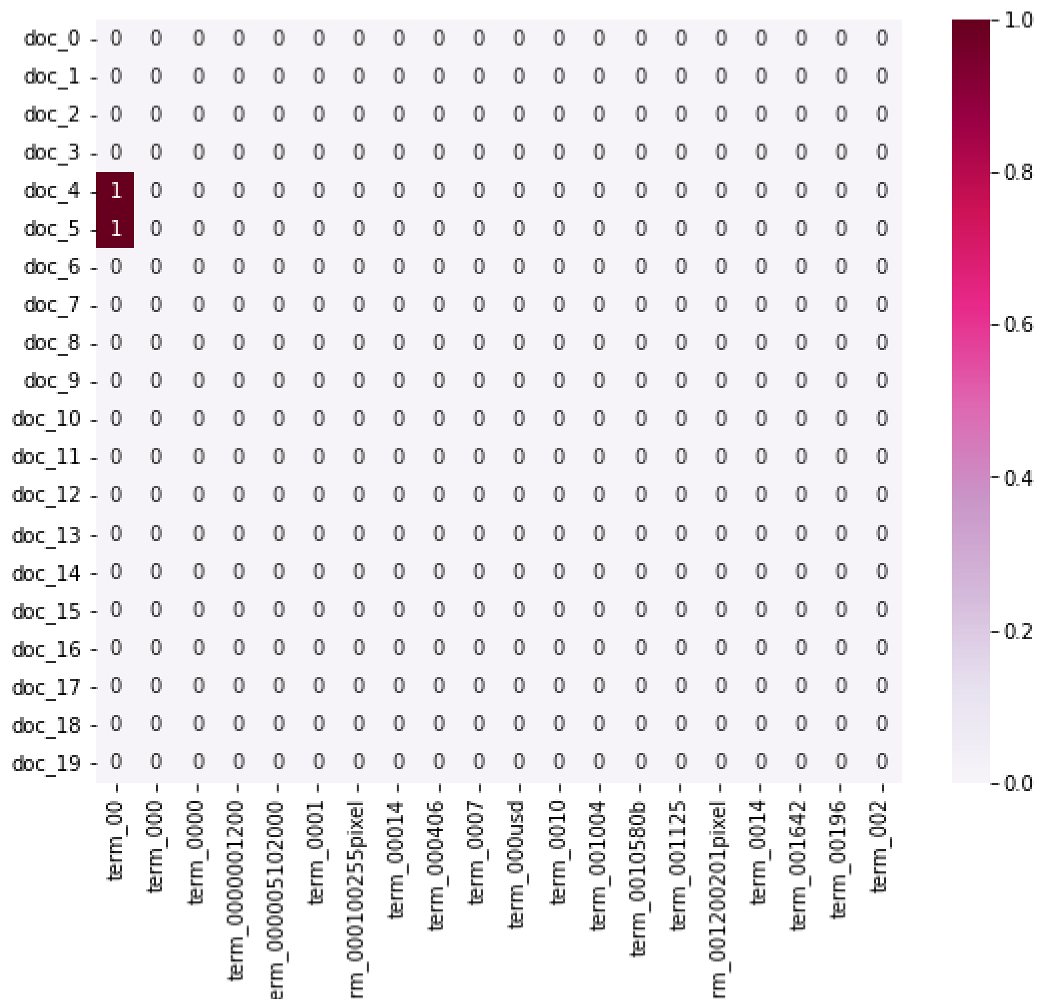


By Seaborn

The other big advantage of seaborn is that seaborn has some built-in plots that matplotlib does not support. Most of these can eventually be replicated by hacking away at matplotlib, but they're not built in and require much more effort to build.

So without further ado, let us try it now!

```
import seaborn as sns

df_todraw = pd.DataFrame(plot_z, columns = plot_x, index = plot_y)
plt.subplots(figsize=(9, 7))
ax = sns.heatmap(df_todraw,
                 cmap="PuRd",
                 vmin=0, vmax=1, annot=True)
```



Check out more beautiful color palettes here: https://python-graph-gallery.com/197-available-color-palettes-with-matplotlib/

>>> Exercise 11 (take home):

From the chart above, we can see how sparse the term-document matrix is; i.e., there is only one terms with frequency of 1 in the subselection of the matrix. By the way, you may have noticed that we only selected 20 articles and 20 terms to plot the histrogram. As an excersise you can try to modify the code above to plot the entire term-document matrix or just a sample of it. How would you do this efficiently? Remember there is a lot of words in the vocab. Report below what methods you would use to get a nice and useful visualization

-- **Answer** --

Problem: Since the frequency of the term exist is sparse, it is difficult to draw the whole map out when the matrix is huge, and also if we split it into "subheatmap", it makes no sense to analysis.

Therefore, if we aggregate the frequency of the term in the whole document and make a plot table, we are able to know how frequently those certain terms exist in the whole dataset.

Since we want to know about the frequency of the terms, we may delete some of the less-frequency terms, it might be more efficient to see more imformation from the plot table.

Although it might not be a good visualization for prediction or other machine learning processes, it might help people initially know clearly about the dataset.

Since the dataset is too big to perform the table clearly, I will query the terms from index 1650~1850 to implement the plot table

```
#Filter the less-frequency word out
count_vector_more = CountVectorizer(min_df=10, max_df = 1000)
X_counts_more = count_vector_more.fit_transform(X.text)
#Aggregation
term_frequencies = []
for j in range(1650, 1850):
    term_frequencies.append(sum(X_counts_more[:,j].toarray()))

term_frequencies = np.asarray(X_counts_more.sum(axis=0))[0]
plt.subplots(figsize=(100, 40))
g = sns.barplot(x=count_vector_more.get_feature_names_out()[1650:1850],
            y=term_frequencies[1650:1850])
g.set(xlabel='Term Name', ylabel='Frequency')
g.set_xticklabels(count_vector_more.get_feature_names_out()[1650:1850], rotation
```

The great thing about what we have done so far is that we now open doors to new problems. Let us be optimistic. Even though we have the problem of sparsity and a very high dimensional data, we are now closer to uncovering wonders from the data. You see, the price you pay for the hard work is worth it because now you are gaining a lot of knowledge from what was just a list of what appeared to be irrelevant articles. Just the fact that you can blow up the data and find out interesting characteristics about the dataset in just a couple lines of code, is something that

5.4 Dimensionality Reduction

Since we have just touched on the concept of sparsity most naturally the problem of "curse of dimentionality" comes up. I am not going to get into the full details of what dimensionality reduction is and what it is good for just the fact that is an excellent technique for visualizing data efficiently (please refer to notes for more information). All I can say is that we are going to deal with the issue of sparsity with a few lines of code. And we are going to try to visualize our data more efficiently with the results.

We are going to make use of Principal Component Analysis to efficiently reduce the dimensions of our data, with the main goal of "finding a projection that captures the largest amount of variation in the data." This concept is important as it is very useful for visualizing and observing the characteristics of our dataset.

PCA Algorithm

Input: Raw term-vector matrix

Output: Projections

```
from sklearn.decomposition import PCA
```

```
X_reduced = PCA(n_components = 2).fit_transform(X_counts.toarray())
```

```
X_reduced.shape
```

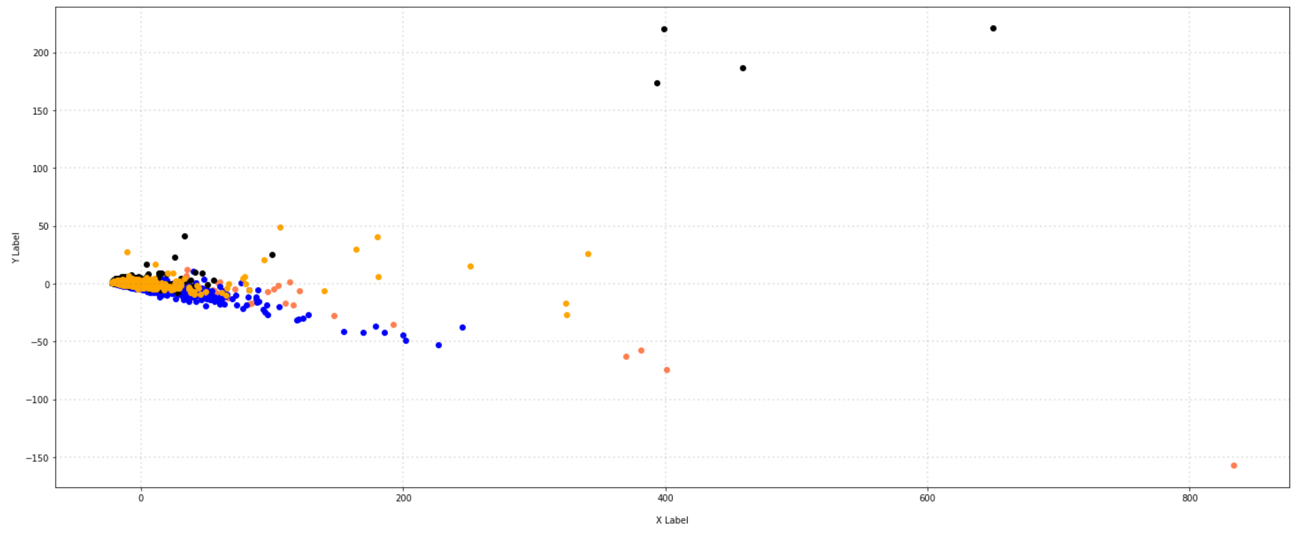
```
    (2257, 2)
```

```
categories
```

```
    ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
```

```
col = ['coral', 'blue', 'black', 'orange']

# plot
fig = plt.figure(figsize = (25,10))
```

```
ax = fig.subplots()

for c, category in zip(col, categories):
    xs = X_reduced[X['category_name'] == category].T[0]
    ys = X_reduced[X['category_name'] == category].T[1]

    ax.scatter(xs, ys, c = c, marker='o')

ax.grid(color='gray', linestyle=':', linewidth=2, alpha=0.2)
ax.set_xlabel('\nX Label')
ax.set_ylabel('\nY Label')

plt.show()
```



From the 2D visualization above, we can see a slight "hint of separation in the data"; i.e., they might have some special grouping by category, but it is not immediately clear. The PCA was applied to the raw frequencies and this is considered a very naive approach as some words are not really unique to a document. Only categorizing by word frequency is considered a "bag of words" approach. Later on in the course you will learn about different approaches on how to create better features from the term-vector matrix, such as term-frequency inverse document frequency so-called TF-IDF.

>>> Exercise 12 (take home):

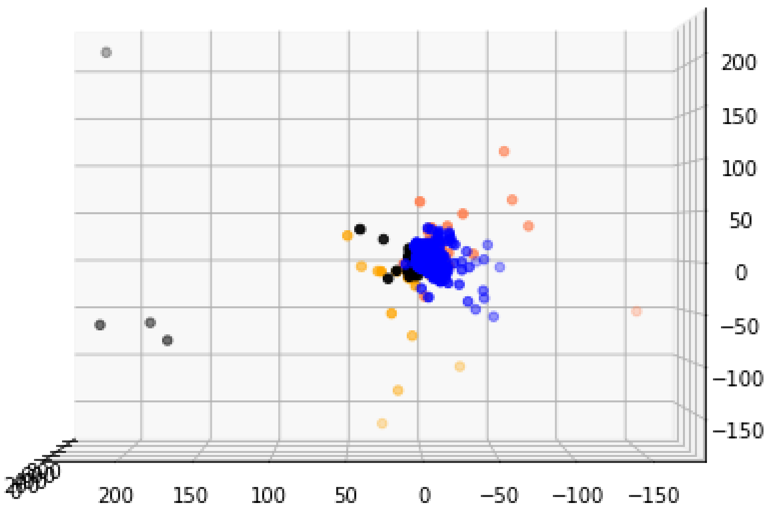
Please try to reduce the dimension to 3, and plot the result use 3-D plot. Use at least 3 different angle (camera position) to check your result and describe what you found.

Hint: you can refer to Axes3D in the documentation.

```
# Answer here
X_reduced_3 = PCA(n_components = 3).fit_transform(X_counts.toarray())
```

1.

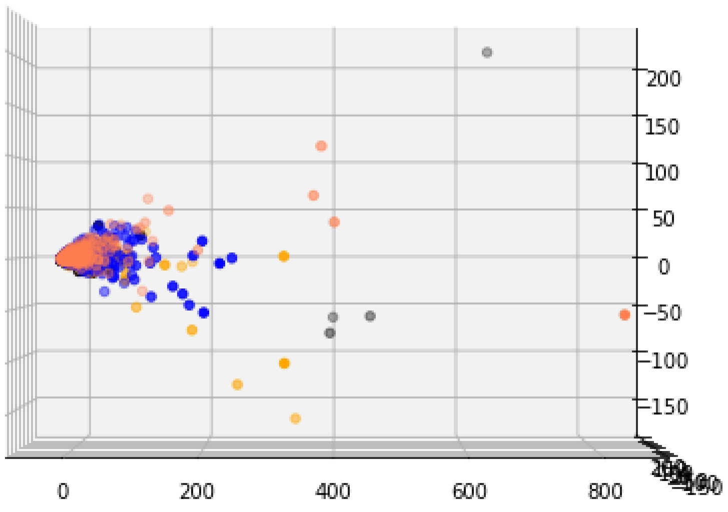
```
# Creating figure
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
ax = Axes3D(fig)
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")


for c, category in zip(col, categories):
    xs = X_reduced_3[X['category_name'] == category].T[0]
    ys = X_reduced_3[X['category_name'] == category].T[1]
    zs = X_reduced_3[X['category_name'] == category].T[2]
    ax.scatter3D(xs, ys, zs, color = c)

# Creating plot

ax.view_init(elev=0, azim=180)
plt.show()
```



2.

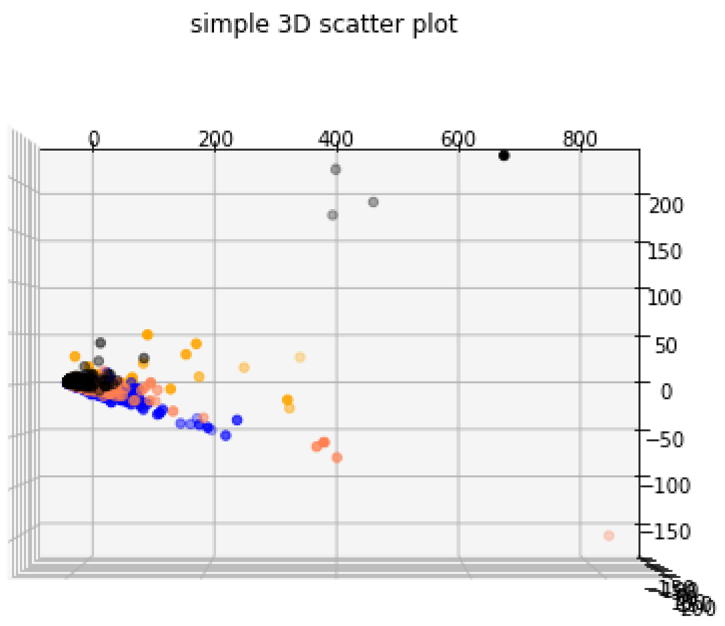
```
# Creating figure
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
ax = Axes3D(fig)
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")


for c, category in zip(col, categories):
    xs = X_reduced_3[X['category_name'] == category].T[0]
    ys = X_reduced_3[X['category_name'] == category].T[1]
    zs = X_reduced_3[X['category_name'] == category].T[2]
    ax.scatter3D(xs, ys, zs, color = c)

# Creating plot
ax.view_init(elev=0, azim=-90)
ax = plt.show()
```



3.

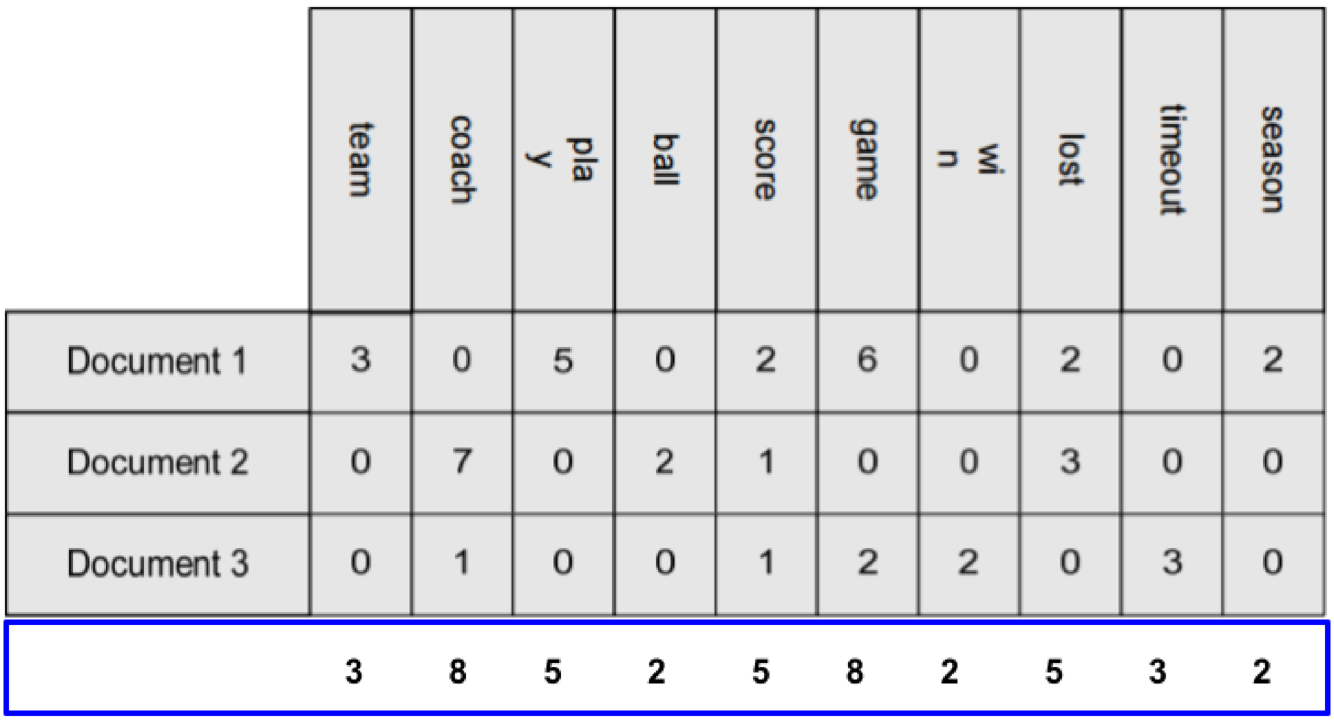
```
# Creating figure
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
ax = Axes3D(fig)
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")


for c, category in zip(col, categories):
    xs = X_reduced_3[X['category_name'] == category].T[0]
    ys = X_reduced_3[X['category_name'] == category].T[1]
    zs = X_reduced_3[X['category_name'] == category].T[2]
    ax.scatter3D(xs, ys, zs, color = c)

# Creating plot
#YZ
plt.title("simple 3D scatter plot")
ax.view_init(elev=90, azim=-90)
ax = plt.show()
```



5.5 Attribute Transformation / Aggregation

We can do other things with the term-vector matrix besides applying dimensionality reduction technique to deal with sparsity problem. Here we are going to generate a simple distribution of the words found in all the entire set of articles. Intuitively, this may not make any sense, but in data science sometimes we take some things for granted, and we just have to explore the data first before making any premature conclusions. On the topic of attribute transformation, we will take the word distribution and put the distribution in a scale that makes it easy to analyze patterns in the distrubution of words. Let us get into it!

First, we need to compute these frequencies for each term in all documents. Visually speaking, we are seeking to add values of the 2D matrix, vertically; i.e., sum of each column. You can also refer to this process as aggregation, which we won't explore further in this notebook because of the type of data we are dealing with. But I believe you get the idea of what that includes.



	team	coach	play	ball	score	game	win	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0
	3	8	5	2	5	8	2	5	3	2

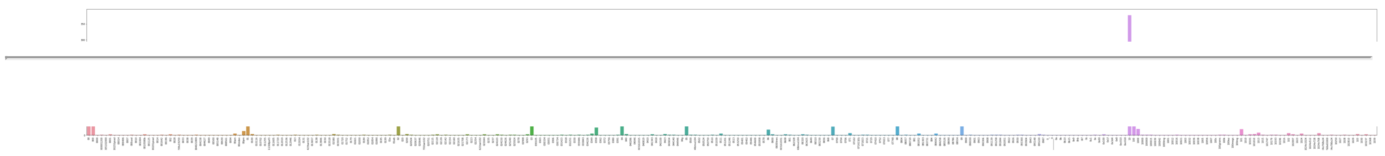
```
# note this takes time to compute. You may want to reduce the amount of terms yo
term_frequencies = []
for j in range(0,X_counts.shape[1]):
    term_frequencies.append(sum(X_counts[:,j].toarray()))
```

```
term_frequencies = np.asarray(X_counts.sum(axis=0))[0]
```

```
term_frequencies[0] #sum of first term
```

```
     134
```

```
plt.subplots(figsize=(100, 10))
g = sns.barplot(x=count_vect.get_feature_names_out()[:300],
            y=term_frequencies[:300])
g.set_xticklabels(count_vect.get_feature_names_out()[:300], rotation = 90);
```

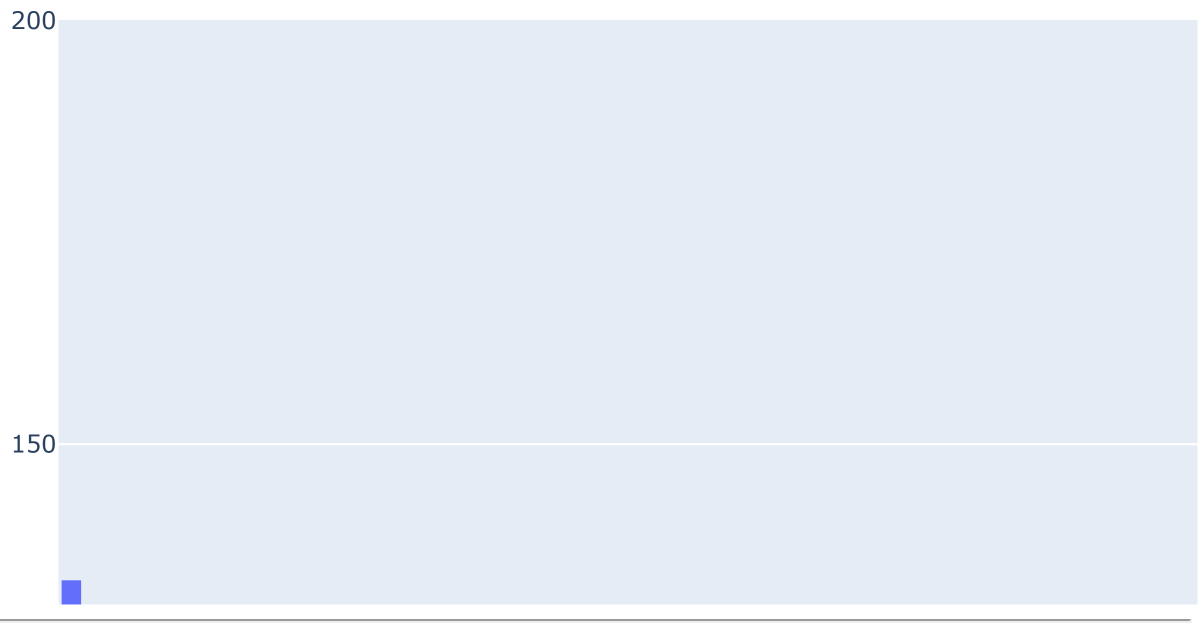


按兩下 (或按 Enter 鍵) 即可編輯

>>> Exercise 13 (take home):

If you want a nicer interactive visualization here, I would encourage you try to install and use plotly to achieve this.

```
import plotly.express as px
import plotly.graph_objs as go

fig = px.bar(x = count_vect.get_feature_names_out()[:300], y = term_frequencies[
fig.update_layout(yaxis_range=[0, 200])
fig.show()
```



>>> Exercise 14 (take home):

The chart above contains all the vocabulary, and it's computationally intensive to both compute and visualize. Can you efficiently reduce the number of terms you want to visualize as an exercise.

Answer:

From my observation, most of the terms only exist once, and some of the conjuction or preposition which often exist in the sentence but are not useful for analysis.

Therefore, I tend to reduce the term which only exist once, and terms exist over 1000 hundred times.

```
# Answer here
#Filter the less-frequency word out
count_vector_more = CountVectorizer(min_df=10, max_df = 1000)
X_counts_more = count_vector_more.fit_transform(X.text)

print("Before reduciing terms data shape: ", X_counts.shape)
print("After reducing terms data shape: ", X_counts_more.shape)
```

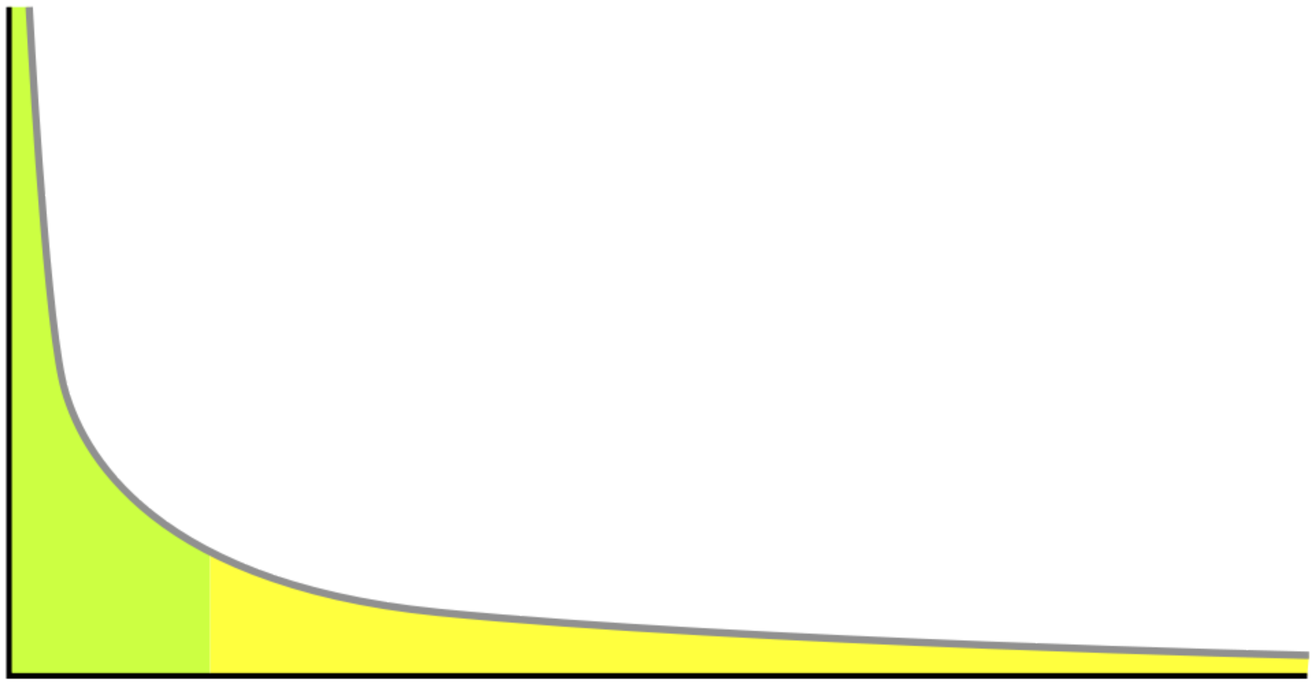
```
    Before reduciing terms data shape:  (2257, 35788)
    After reducing terms data shape:  (2257, 4781)
```

We can see from the shape the term reduce rapidly.

按兩下 (或按 Enter 鍵) 即可編輯

>>> Exercise 15 (take home):

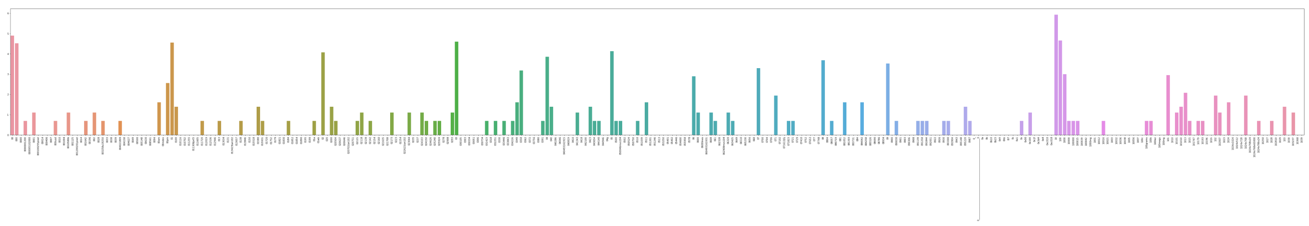
Additionally, you can attempt to sort the terms on the `x-axis` by frequency instead of in alphabetical order. This way the visualization is more meaninfgul and you will be able to observe the so called long tail (get familiar with this term since it will appear a lot in data mining and other statistics courses). see picture below



```
# Answer here
word_frequent_sort = [(word, term_frequencies[idx]) for word, idx in count_vect.\
word_frequent_sort = sorted(word_frequent_sort, key = lambda x: x[1], reverse=Tru

word_list = []
freq_list = []
for x, y in word_frequent_sort:
  word_list.append(x)
  freq_list.append(y)

plt.subplots(figsize=(100, 10))
g = sns.barplot(x=word_list[:300],
            y=freq_list[:300])
g.set_xticklabels(word_list[:300], rotation = 90);
```

Since we already have those term frequencies, we can also transform the values in that vector into the log distribution. All we need is to import the `math` library provided by python and apply it to the array of values of the term frequency vector. This is a typical example of attribute transformation. Let's go for it. The log distribution is a technique to visualize the term frequency into a scale that makes you easily visualize the distribution in a more readable format. In other words, the variations between the term frequencies are now easy to observe. Let us try it out!

```
import math
term_frequencies_log = [math.log(i) for i in term_frequencies]


plt.subplots(figsize=(100, 10))
g = sns.barplot(x=count_vect.get_feature_names_out()[:300],
                y=term_frequencies_log[:300])
g.set_xticklabels(count_vect.get_feature_names_out()[:300], rotation = 90);
```



Besides observing a complete transformation on the disrtibution, notice the scale on the y-axis. The log distribution in our unsorted example has no meaning, but try to properly sort the terms by their frequency, and you will see an interesting effect. Go for it!

5.6 Discretization and Binarization

In this section we are going to discuss a very important pre-preprocessing technique used to transform the data, specifically categorical values, into a format that satisfies certain criteria required by particular algorithms. Given our current original dataset, we would like to transform one of the attributes, `category_name`, into four binary attributes. In other words, we are taking the category name and replacing it with a `n` asymmetric binary attributes. The logic behind this transformation is discussed in detail in the recommended Data Mining text book (please refer to it on page 58). People from the machine learning community also refer to this transformation as one-hot encoding, but as you may become aware later in the course, these concepts are all the same, we just have different prefrence on how we refer to the concepts. Let us take a look at what we want to achieve in code.

```
from sklearn import preprocessing, metrics, decomposition, pipeline, dummy

mlb = preprocessing.LabelBinarizer()

mlb.fit(X.category)
```

```
    LabelBinarizer()
```

```
X['bin_category'] = mlb.transform(X['category']).tolist()

X[0:9]
```

	text	category	category_name	unigra
0	From: sd345@city.ac.uk (Michael Collier) Subje...	1	comp.graphics	[From, :, sd345, @ city.ac.uk, (, Michael, Co
1	From: ani@ms.uky.edu (Aniruddha B. Deglurkar) ...	1	comp.graphics	[From, :, ani, @, ms.uky.ed (, Aniruddha, B.
2	From: djohnson@cs.ucsd.edu (Darin Johnson) Sub...	3	soc.religion.christian	[From, :, djohnson, @ cs.ucsd.edu, (, Darin,
3	From: s0612596@let.rug.nl (M.M. Zwart) Subject...	3	soc.religion.christian	[From, :, s0612596, @ let.rug.nl, (, M.M, .,
4	From: stanly@grok11.columbiasc.ncr.com (stanly...	3	soc.religion.christian	[From, :, stanly, @ grok11.columbiasc.ncr.com

Take a look at the new attribute we have added to the `X` table. You can see that the new attribute, which is called `bin_category`, contains an array of 0's and 1's. The `1` is basically to indicate the position of the label or category we binarized. If you look at the first two records, the one is places in slot 2 in the array; this helps to indicate to any of the algorithms which we are feeding this data to, that the record belong to that specific category.

Attributes with **continuous values** also have strategies to tranform the data; this is usually called **Discretization** (please refer to the text book for more inforamation).

>>> Exercise 16 (take home):

Try to generate the binarization using the `category_name` column instead. Does it work?

```
# Answer here
mlb.fit(X.category_name)
X['bin_category_name'] = mlb.transform(X['category_name']).tolist()
X[0:9]
```

	text	category	category_name	unigra
0	From: sd345@city.ac.uk (Michael Collier) Subje...	1	comp.graphics	[From, :, sd345, @, city.ac.uk, (, Michael, Co
1	From: ani@ms.uky.edu (Aniruddha B. Deglurkar) ...	1	comp.graphics	[From, :, ani, @, ms.uky.ed (, Aniruddha, B.
2	From: djohnson@cs.ucsd.edu (Darin Johnson) Sub...	3	soc.religion.christian	[From, :, djohnson, @, cs.ucsd.edu, (, Darin,
3	From: s0612596@let.rug.nl (M.M. Zwart) Subject...	3	soc.religion.christian	[From, :, s0612596, @, let.rug.nl, (, M.M, .,
4	From: stanly@grok11.columbiasc.ncr.com (stanly...	3	soc.religion.christian	[From, :, stanly, @, grok11.columbiasc.ncr.com

Since the category and the category name has the same information, they should have the same result in binarization. Obviously, it works.

---

6. Data Exploration

Sometimes you need to take a peek at your data to understand the relationships in your dataset. Here, we will focus in a similarity example. Let's take 3 documents and compare them.

```
# We retrieve 3 sentences for a random record
document_to_transform_1 = []
random_record_1 = X.iloc[50]
random_record_1 = random_record_1['text']
document_to_transform_1.append(random_record_1)

document_to_transform_2 = []
random_record_2 = X.iloc[100]
random_record_2 = random_record_2['text']
document_to_transform_2.append(random_record_2)

document_to_transform_3 = []
random_record_3 = X.iloc[150]
random_record_3 = random_record_3['text']
document_to_transform_3.append(random_record_3)
```

Let's look at our emails.

```
print(document_to_transform_1)
print(document_to_transform_2)
print(document_to_transform_3)
```

```
['From: ab@nova.cc.purdue.edu (Allen B) Subject: Re: TIFF: philosophical si
['From: mathew <mathew@mantis.co.uk> Subject: Re: university violating sepa
['From: lfoard@hopper.virginia.edu (Lawrence C. Foard) Subject: Re: Assuran
```

```
from sklearn.preprocessing import binarize

# Transform sentence with Vectorizers
document_vector_count_1 = count_vect.transform(document_to_transform_1)
document_vector_count_2 = count_vect.transform(document_to_transform_2)
document_vector_count_3 = count_vect.transform(document_to_transform_3)

# Binarize vectors to simplify: 0 for abscence, 1 for prescence
document_vector_count_1_bin = binarize(document_vector_count_1)
document_vector_count_2_bin = binarize(document_vector_count_2)
document_vector_count_3_bin = binarize(document_vector_count_3)

# print vectors
print("Let's take a look at the count vectors:")
print(document_vector_count_1.todense())
print(document_vector_count_2.todense())
print(document_vector_count_3.todense())
```

```
    Let's take a look at the count vectors:
    [[0 0 0 ... 0 0 0]]
    [[0 0 0 ... 0 0 0]]
    [[0 0 0 ... 0 0 0]]
```

```
from sklearn.metrics.pairwise import cosine_similarity

# Calculate Cosine Similarity
cos_sim_count_1_2 = cosine_similarity(document_vector_count_1, document_vector_co
cos_sim_count_1_3 = cosine_similarity(document_vector_count_1, document_vector_co
cos_sim_count_2_3 = cosine_similarity(document_vector_count_2, document_vector_co

cos_sim_count_1_1 = cosine_similarity(document_vector_count_1, document_vector_co
cos_sim_count_2_2 = cosine_similarity(document_vector_count_2, document_vector_co
cos_sim_count_3_3 = cosine_similarity(document_vector_count_3, document_vector_co

# Print
print("Cosine Similarity using count bw 1 and 2: %(x)f" %{"x":cos_sim_count_1_2}
print("Cosine Similarity using count bw 1 and 3: %(x)f" %{"x":cos_sim_count_1_3}
print("Cosine Similarity using count bw 2 and 3: %(x)f" %{"x":cos_sim_count_2_3}

print("Cosine Similarity using count bw 1 and 1: %(x)f" %{"x":cos_sim_count_1_1}
print("Cosine Similarity using count bw 2 and 2: %(x)f" %{"x":cos_sim_count_2_2}
print("Cosine Similarity using count bw 3 and 3: %(x)f" %{"x":cos_sim_count_3_3}
```

```
    Cosine Similarity using count bw 1 and 2: 0.608862
    Cosine Similarity using count bw 1 and 3: 0.622050
    Cosine Similarity using count bw 2 and 3: 0.565566
    Cosine Similarity using count bw 1 and 1: 1.000000
    Cosine Similarity using count bw 2 and 2: 1.000000
    Cosine Similarity using count bw 3 and 3: 1.000000
```

As expected, cosine similarity between a sentence and itself is 1. Between 2 entirely different sentences, it will be 0.

We can assume that we have the more common features in the documents 1 and 3 than in documents 1 and 2. This reflects indeed in a higher similarity than that of sentences 1 and 3.

---

7. Concluding Remarks

Wow! We have come a long way! We can now call ourselves experts of Data Preprocessing. You should feel excited and proud because the process of Data Mining usually involves 70% preprocessing and 30% training learning models. You will learn this as you progress in the Data

Mining course. I really feel that if you go through the exercises and challenge yourself, you are on your way to becoming a super Data Scientist.

From here the possibilities for you are endless. You now know how to use almost every common technique for preprocessing with state-of-the-art tools, such as as Pandas and Scikit-learn. You are now with the trend!

After completing this notebook you can do a lot with the results we have generated. You can train algorithms and models that are able to classify articles into certain categories and much more. You can also try to experiment with different datasets, or venture further into text analytics by using new deep learning techniques such as word2vec. All of this will be presented in the next lab session. Until then, go teach machines how to be intelligent to make the world a better place.

. References

- Pandas cook book (Recommended for starters)
- Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Introduction to Data Mining, Addison Wesley