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. We will also provide best practices and introduce visualization tools for effectively conducting big data analysis. Furthermore, we will show you how to implement basic classification techniques.

In [185…

```
# test code for environment setup
import pandas as pd
import numpy as np
import nltk
nltk.download('punkt') # download the NLTK datasets
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
import plotly as py
import math
# If you get "ModuleNotFoundError: No module named 'PAMI'"
# run the following in a new Jupyter cell:
# !pip3 install PAMI
import PAMI
import umap

categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', '
twenty_train = fetch_20newsgroups(subset='train', categories=categories,
```

```
[nltk_data] Downloading package punkt to /Users/parinmac/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

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 3.11.0.
  - You can use newer version, but use at your own risk.

Environment:

We recommend using **uv**, a fast Python package and environment manager developed by Astral. See README.md for the setup instruction

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 (version 3.7.3 recommended, pip install matplotlib==3.7.3)
- Plotly
  - Install and signup for `plotly`
- Seaborn
  - Install and signup for `seaborn`
- NLTK
  - Install `nltk` library
- PAMI
  - Install `PAMI` library
- UMAP
  - Install `UMAP` library

In [186…
```
# 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
```

In [187…
```
import sys
print(sys.executable) # c:\<your path to the project directory>\.venv\Scr
print(sys.version) #3.11.0
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/bin/pytho
n
3.11.0 (v3.11.0:deaf509e8f, Oct 24 2022, 14:43:23) [Clang 13.0.0 (clang-13
00.0.29.30)]
```

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.

	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
3	From: s0612596@let.rug.nl (M.M. Zwart) Subject...	soc.religion.christian
4	From: stanly@grok11.columbiasc.ncr.com (stanly...	soc.religion.christian
5	From: vbv@lor.eeap.cwru.edu (Virgilio (Dean) B...	soc.religion.christian
6	From: jodfishe@silver.ucs.indiana.edu (joseph ...	soc.religion.christian
7	From: aldridge@netcom.com (Jacquelin Aldridge)...	sci.med
8	From: geb@cs.pitt.edu (Gordon Banks) Subject: ...	sci.med
9	From: libman@hsc.usc.edu (Marlena Libman) Subj...	sci.med

---

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:

In [188…
```
# categories
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', '
```

In [189…
```
# 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)
#This command also shuffles the data randomly, but with random_state we c
#if we choose the same number, in this case "42". This is good for us, it
#we want to run the code.
```

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

In [190…
```
twenty_train.data[0:2]
```

Out[190…
```
['From: sd345@city.ac.uk (Michael Collier)\nSubject: Converting images t
o HP LaserJet III?\nNntp-Posting-Host: hampton\nOrganization: The City U
niversity\nLines: 14\n\nDoes anyone know of a good way (standard PC appl
ication/PD utility) to\nconvert tif/img/tga files into LaserJet III form
at.  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\nTh
anks in advance.  Michael.\n-- \nMichael Collier (Programmer)
The Computer Unit,\nEmail: M.P.Collier@uk.ac.city                The Cit
y University,\nTel: 071 477-8000 x3769                       London,\nFa
x: 071 477-8565                              EC1V 0HB.\n',
 "From: ani@ms.uky.edu (Aniruddha B. Deglurkar)\nSubject: help: Splittin
g 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 probl
em:\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 i
s made up of cells.\n\n\tMy problem is this :\n\tThe trimming area has t
o be split up into individual smaller\n\tcells bounded by the trimming c
urve segments. If a cell\n\tis wholly inside the area...then it is outpu
t as a whole ,\n\telse it is trivially rejected. \n\n\tDoes any body kno
w how thiss can be done, or is there any algo. \n\tsomewhere for doing t
his.\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`

In [191…
```
twenty_train.target_names
```

Out[191…
```
['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
```

In [192…
```
len(twenty_train.data)
```

Out[192…
```
2257
```

In [193…
```
len(twenty_train.filenames)
```

Out[193…
```
2257
```

We can also print an example from the subset

In [194…
```
# 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

In [195…
```
print(twenty_train.target_names[twenty_train.target[0]])
```

```
comp.graphics
```

In [196…
```
twenty_train.target[0]
```

Out[196… `np.int64(1)`

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

In [197…
```
# category of first 10 documents.
twenty_train.target[0:10]
```

Out[197… `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:

In [198…
```
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 (Watch Video):

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

In [199…
```
# Answer here
for i in range(3):
    print(f"example {i+1}")
    print("\n".join(twenty_train.data[i].split("\n"))) # split the line b
```

example 1
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.

example 2
From: ani@ms.uky.edu (Aniruddha B. Deglurkar)
Subject: help: Splitting a trimming region along a mesh
Organization: University Of Kentucky, Dept. of Math Sciences
Lines: 28

Hi,

I have a problem, I hope some of the 'gurus' can help me solve.

Background of the problem:
I have a rectangular mesh in the uv domain, i.e the mesh is a
mapping of a 3d Bezier patch into 2d. The area in this domain
which is inside a trimming loop had to be rendered. The trimming
loop is a set of 2d Bezier curve segments.
For the sake of notation: the mesh is made up of cells.

My problem is this :
The trimming area has to be split up into individual smaller
cells bounded by the trimming curve segments. If a cell
is wholly inside the area...then it is output as a whole ,
else it is trivially rejected.

Does any body know how thiss can be done, or is there any algo.
somewhere for doing this.

Any help would be appreciated.

Thanks,
Ani.
--
To get irritated is human, to stay cool, divine.

example 3
From: djohnson@cs.ucsd.edu (Darin Johnson)
Subject: Re: harrassed at work, could use some prayers

Organization: =CSE Dept., U.C. San Diego
Lines: 63

(Well, I'll email also, but this may apply to other people, so
I'll post also.)

>I've been working at this company for eight years in various
>engineering jobs. I'm female. Yesterday I counted and realized that
>on seven different occasions I've been sexually harrassed at this
>company.

>I dreaded coming back to work today. What if my boss comes in to ask
>me some kind of question...

Your boss should be the person bring these problems to. If he/she
does not seem to take any action, keep going up higher and higher.
Sexual harrassment does not need to be tolerated, and it can be an
enormous emotional support to discuss this with someone and know that
they are trying to do something about it. If you feel you can not
discuss this with your boss, perhaps your company has a personnel
department that can work for you while preserving your privacy. Most
companies will want to deal with this problem because constant anxiety
does seriously affect how effectively employees do their jobs.

It is unclear from your letter if you have done this or not. It is
not inconceivable that management remains ignorant of employee
problems/strife even after eight years (it's a miracle if they do
notice). Perhaps your manager did not bring to the attention of
higher ups? If the company indeed does seem to want to ignore the
entire problem, there may be a state agency willing to fight with
you. (check with a lawyer, a women's resource center, etc to find out)

You may also want to discuss this with your paster, priest, husband,
etc. That is, someone you know will not be judgemental and that is
supportive, comforting, etc. This will bring a lot of healing.

>So I returned at 11:25, only to find that ever single
>person had already left for lunch. They left at 11:15 or so. No one
>could be bothered to call me at the other building, even though my
>number was posted.

This happens to a lot of people. Honest. I believe it may seem
to be due to gross insensitivity because of the feelings you are
going through. People in offices tend to be more insensitive while
working than they normally are (maybe it's the hustle or stress or...)
I've had this happen to me a lot, often because they didn't realize
my car was broken, etc. Then they will come back and wonder why I
didn't want to go (this would tend to make me stop being angry at
being ignored and make me laugh). Once, we went off without our
boss, who was paying for the lunch :-)

>For this
>reason I hope good Mr. Moderator allows me this latest indulgence.

Well, if you can't turn to the computer for support, what would
we do? (signs of the computer age :-)

In closing, please don't let the hateful actions of a single person
harm you. They are doing it because they are still the playground
bully and enjoy seeing the hurt they cause. And you should not

```
accept the opinions of an imbecile that you are worthless - much
wiser people hold you in great esteem.
--
Darin Johnson
djohnson@ucsd.edu
  - Luxury!  In MY day, we had to make do with 5 bytes of swap...
```

---

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.

In [200…

```
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= ['t
```

Out[200…

	text
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...
...	...
2252	From: roos@Operoni.Helsinki.FI (Christophe Roo...
2253	From: mhollowa@ic.sunysb.edu (Michael Holloway...
2254	From: sasghm@theseus.unx.sas.com (Gary Merrill...
2255	From: Dan Wallach <dwallach@cs.berkeley.edu> S...
2256	From: dyer@spdcc.com (Steve Dyer) Subject: Re:...

2257 rows × 1 columns

In [201…
```
len(X)
```

Out[201… 2257

In [202…
```
X[0:2]
```

Out[202…

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

In [203…
```
for t in X["text"][:2]:
    print(t)
```

```
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 util
ity) 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.
From: ani@ms.uky.edu (Aniruddha B. Deglurkar) Subject: help: Splitting a t
rimming region along a mesh  Organization: University Of Kentucky, Dept. o
f Math Sciences Lines: 28       Hi,     I have a problem, I hope some of t
he 'gurus' can help me solve.   Background of the problem:      I have a r
ectangular mesh in the uv domain, i.e  the mesh is a    mapping of a 3d Be
zier patch into 2d. The area in this domain     which is inside a trimming
loop had to be rendered. The trimming   loop is a set of 2d Bezier curve s
egments.        For the sake of notation: the mesh is made up of cells.
My problem is this :    The trimming area has to be split up into individu
al smaller      cells bounded by the trimming curve segments. If a cell
is wholly inside the area...then it is output as a whole ,      else it is
trivially rejected.     Does any body know how thiss can be done, or is th
ere any algo.   somewhere for doing this.       Any help would be apprecia
ted.    Thanks,         Ani. --  To get irritated is human, to stay cool,
divine.
```

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.

In [204…
```
# add category to the dataframe
X['category'] = twenty_train.target
```

In [205…
```
# add category label also
X['category_name'] = X.category.apply(lambda t: dmh.format_labels(t, twen
```

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

In [206…
```
X[0:10]
```

Out[206…

	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

To begin to show you the awesomeness of Pandas dataframes, let us look at how to run a simple query on our dataset. We want to query for the first 10 rows (documents), and we only want to keep the `text` and `category_name` attributes or fields.

In [207…

```
# a simple query
X[:10][["text","category_name"]]
```

Out[207…

	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
3	From: s0612596@let.rug.nl (M.M. Zwart) Subject...	soc.religion.christian
4	From: stanly@grok11.columbiasc.ncr.com (stanly...	soc.religion.christian
5	From: vbv@lor.eeap.cwru.edu (Virgilio (Dean) B...	soc.religion.christian
6	From: jodfishe@silver.ucs.indiana.edu (joseph ...	soc.religion.christian
7	From: aldridge@netcom.com (Jacquelin Aldridge)...	sci.med
8	From: geb@cs.pitt.edu (Gordon Banks) Subject: ...	sci.med
9	From: libman@hsc.usc.edu (Marlena Libman) Subj...	sci.med

Let us look at a few more interesting queries to familiarize ourselves with the efficiency and conveniency of Pandas dataframes.

Let's query the last 10 records

In [208…

```
X[-10:]
```

Out[208…

	text	category	category_name
2247	From: daniels@math.ufl.edu (TV's Big Dealer) S...	3	soc.religion.christian
2248	From: "danny hawrysio" <danny.hawrysio@canrem....	1	comp.graphics
2249	From: shellgate!llo@uu4.psi.com (Larry L. Over...	3	soc.religion.christian
2250	From: ingles@engin.umich.edu (Ray Ingles) Subj...	0	alt.atheism
2251	From: Mark-Tarbell@suite.com Subject: Amniocen...	2	sci.med
2252	From: roos@Operoni.Helsinki.FI (Christophe Roo...	2	sci.med
2253	From: mhollowa@ic.sunysb.edu (Michael Holloway...	2	sci.med
2254	From: sasghm@theseus.unx.sas.com (Gary Merrill...	2	sci.med
2255	From: Dan Wallach <dwallach@cs.berkeley.edu> S...	2	sci.med
2256	From: dyer@spdcc.com (Steve Dyer) Subject: Re:...	2	sci.med

Ready for some sourcery? Brace yourselves! Let us see if we can query the first 10th record in our dataframe. For this we will use the build-in function called `loc`. This allows us to explicity define the columns you want to query.

In [209…
```
# using loc (by label)
X.loc[:10, 'text']
```

Out[209…
```
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...
10    From: anasaz!karl@anasazi.com (Karl Dussik) Su...
Name: text, dtype: object
```

You can also use the `iloc` function to query a selection of our dataset by position. Take a look at this great discussion on the differences between the `iloc` and `loc` functions.

In [210…
```
# using iloc (by position)
X.iloc[:10, 0]
```

Out[210…
```
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
```

>>> Exercise 2 (take home):

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

In [211…
```
#Answer here
X[:-130][["text","category","category_name"]]
```

Out[211…

	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
...	...	...	...
2122	From: Steve_Mullins@vos.stratus.com Subject: R...	0	alt.atheism
2123	From: ritley@uimrl7.mrl.uiuc.edu () Subject: M...	2	sci.med
2124	From: uabdpo.dpo.uab.edu!gila005 (Stephen Holl...	2	sci.med
2125	From: JBF101@psuvm.psu.edu Subject: same-sex m...	3	soc.religion.christian
2126	From: kturner@copper.denver.colorado.edu (Kath...	2	sci.med

2127 rows × 3 columns

>>> Exercise 3 (Watch Video):

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

In [212…

```
# Answer here
X[X["category_name"]=="sci.med"][::10][0:5]
```

Out[212…

	text	category	category_name
7	From: aldridge@netcom.com (Jacquelin Aldridge)...	2	sci.med
49	From: jimj@contractor.EBay.Sun.COM (Jim Jones)...	2	sci.med
82	From: jason@ab20.larc.nasa.gov (Jason Austin) ...	2	sci.med
118	From: rogers@calamari.hi.com (Andrew Rogers) S...	2	sci.med
142	From: lady@uhunix.uhcc.Hawaii.Edu (Lee Lady) S...	2	sci.med

---

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 convince ourselves of the true power of Pandas dataframes.

4.1 Missing Values

First, let us consider that our dataset has some *missing values* and we want to remove those values. In its current state our dataset has no missing values, but for practice sake we will add some records with missing values and then write some code to deal with these objects that contain missing values. You will see for yourself how easy it is to deal with missing values once you have your data transformed into a Pandas dataframe.

Before we jump into coding, let us do a quick review of what we have learned in the Data Mining course. Specifically, let's review the methods used to deal with missing values.

The most common reasons for having missing values in datasets has to do with how the data was initially collected. A good example of this is when a patient comes into the ER room, the data is collected as quickly as possible and depending on the conditions of the patients, the personal data being collected is either incomplete or partially complete. In the former and latter cases, we are presented with a case of "missing values". Knowing that patients data is particularly critical and can be used by the health authorities to conduct some interesting analysis, we as the data miners are left with the tough task of deciding what to do with these missing and incomplete records. We need to deal with these records because they are definitely going to affect our analysis or learning algorithms. So what do we do? There are several ways to handle missing values, and some of the more effective ways are presented below (Note: You can reference the slides - Session 1 Handout for the additional information).

- **Eliminate Data Objects** - Here we completely discard records once they contain some missing values. This is the easiest approach and the one we will be using in this notebook. The immediate drawback of going with this approach is that you lose some information, and in some cases too much of it. Now imagine that half of the records have at least one or more missing values. Here you are presented with the tough decision of quantity vs quality. In any event, this decision must be made carefully, hence the reason for emphasizing it here in this notebook.
- **Estimate Missing Values** - Here we try to estimate the missing values based on some criteria. Although this approach may be proven to be effective, it is not always the case, especially when we are dealing with sensitive data, like **Gender** or **Names**. For fields like **Address**, there could be ways to obtain these missing

addresses using some data aggregation technique or obtain the information directly from other databases or public data sources.

- **Ignore the missing value during analysis** - Here we basically ignore the missing values and proceed with our analysis. Although this is the most naive way to handle missing values it may proof effective, especially when the missing values includes information that is not important to the analysis being conducted. But think about it for a while. Would you ignore missing values, especially when in this day and age it is difficult to obtain high quality datasets? Again, there are some tradeoffs, which we will talk about later in the notebook.

- **Replace with all possible values** - As an efficient and responsible data miner, we sometimes just need to put in the hard hours of work and find ways to makes up for these missing values. This last option is a very wise option for cases where data is scarce (which is almost always) or when dealing with sensitive data. Imagine that our dataset has an **Age** field, which contains many missing values. Since **Age** is a continuous variable, it means that we can build a separate model for calculating the age for the incomplete records based on some rule-based approach or probabilistic approach.

As mentioned earlier, we are going to go with the first option but you may be asked to compute missing values, using a different approach, as an exercise. Let's get to it!

First we want to add the dummy records with missing values since the dataset we have is perfectly composed and cleaned that it contains no missing values. First let us check for ourselves that indeed the dataset doesn't contain any missing values. We can do that easily by using the following built-in function provided by Pandas.

In [213…
```
# check missing values
X.isnull()
```

Out[213…

	text	category	category_name
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False
…	…	…	…
2252	False	False	False
2253	False	False	False
2254	False	False	False
2255	False	False	False
2256	False	False	False

2257 rows × 3 columns

The `isnull` function looks through the entire dataset for null values and returns `True` wherever it finds any missing field or record. As you will see above, and as we anticipated, our dataset looks clean and all values are present, since `isnull` returns **False** for all fields and records. But let us start to get our hands dirty and build a nice little function to check each of the records, column by column, and return a nice little message telling us the amount of missing records found. This excerice will also encourage us to explore other capabilities of pandas dataframes. In most cases, the build-in functions are good enough, but as you saw above when the entire table was printed, it is impossible to tell if there are missing records just by looking at preview of records manually, especially in cases where the dataset is huge. We want a more reliable way to achieve this. Let's get to it!

In [214…

```
X.isnull().apply(lambda x: dmh.check_missing_values(x))
```

Out[214…

	text	category	category_name
0	The amoung of missing records is:	The amoung of missing records is:	The amoung of missing records is:
1	0	0	0

Okay, a lot happened there in that one line of code, so let's break it down. First, with the `isnull` we tranformed our table into the **True/False** table you see above, where **True** in this case means that the data is missing and **False** means that the data is present. We then take the transformed table and apply a function to each row that essentially counts to see if there are missing values in each record and print out how much missing values we found. In other words the `check_missing_values` function looks through each field (attribute or column) in the dataset and counts how many missing values were found.

There are many other clever ways to check for missing data, and that is what makes Pandas so beautiful to work with. You get the control you need as a data scientist or just a person working in data mining projects. Indeed, Pandas makes your life easy!

>>> Exercise 4 (Watch Video):

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.

In [215…
```
# Answer here
X.isnull().apply(lambda x: dmh.check_missing_values(x), axis=1)
```

Out[215…
```
0       (The amoung of missing records is: , 0)
1       (The amoung of missing records is: , 0)
2       (The amoung of missing records is: , 0)
3       (The amoung of missing records is: , 0)
4       (The amoung of missing records is: , 0)
                        ...
2252    (The amoung of missing records is: , 0)
2253    (The amoung of missing records is: , 0)
2254    (The amoung of missing records is: , 0)
2255    (The amoung of missing records is: , 0)
2256    (The amoung of missing records is: , 0)
Length: 2257, dtype: object
```

We have our function to check for missing records, now let us do something mischievous and insert some dummy data into the dataframe and test the reliability of our function. This dummy data is intended to corrupt the dataset. I mean this happens a lot today, especially when hackers want to hijack or corrupt a database.

We will insert a `Series`, which is basically a "one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.", into our current dataframe.

In [216…
```
dummy_series = pd.Series(["dummy_record", 1], index=["text", "category"])
```

In [217…
```
dummy_series
```

Out[217…
```
text        dummy_record
category               1
dtype: object
```

In [218…
```
dummy_series.to_frame().T
# .to_frame() -> Convert Series to DataFrame
# .T          -> Transpose
```

Out[218…

	text	category
0	dummy_record	1

In [219…
```
result_with_series = pd.concat([X, dummy_series.to_frame().T], ignore_ind
```

In [220…
```
# check if the records was commited into result
len(result_with_series)
```

Out[220… 2258

Now we that we have added the record with some missing values. Let try our function and see if it can detect that there is a missing value on the resulting dataframe.

In [221…
```
result_with_series.isnull().apply(lambda x: dmh.check_missing_values(x))
```

Out[221…

	text	category	category_name
0	The amoung of missing records is:	The amoung of missing records is:	The amoung of missing records is:
1	0	0	1

Indeed there is a missing value in this new dataframe. Specifically, the missing value comes from the `category_name` attribute. As I mentioned before, there are many ways to conduct specific operations on the dataframes. In this case let us use a simple dictionary and try to insert it into our original dataframe `X`. Notice that above we are not changing the `X` dataframe as results are directly applied to the assignment variable provided. But in the event that we just want to keep things simple, we can just directly apply the changes to `X` and assign it to itself as we will do below. This modification will create a need to remove this dummy record later on, which means that we need to learn more about Pandas dataframes. This is getting intense! But just relax, everything will be fine!

In [222…
```
# dummy record as dictionary format
dummy_dict = [{'text': 'dummy_record',
               'category': 1
              }]
```

In [223…
```
X = pd.concat([X, pd.DataFrame(dummy_dict)], ignore_index=True)
```

In [224…
```
len(X)
```

Out[224… 2258

In [225…
```
X.isnull().apply(lambda x: dmh.check_missing_values(x))
```

Out[225…

	text	category	category_name
0	The amoung of missing records is:	The amoung of missing records is:	The amoung of missing records is:
1	0	0	1

So now that we can see that our data has missing values, we want to remove the records with missing values. The code to drop the record with missing that we just

added, is the following:

In [226…
```
X.dropna(inplace=True)
```

... and now let us test to see if we gotten rid of the records with missing values.

In [227…
```
X.isnull().apply(lambda x: dmh.check_missing_values(x))
```

Out[227…

	text	category	category_name
0	The amoung of missing records is:	The amoung of missing records is:	The amoung of missing records is:
1	0	0	0

In [228…
```
len(X)
```

Out[228… 2257

And we are back with our original dataset, clean and tidy as we want it. That's enough on how to deal with missing values, let us now move unto something more fun.

But just in case you want to learn more about how to deal with missing data, refer to the official Pandas documentation.

>>> 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?

In [229…
```
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
```

Out[229…

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

In [230…
```
NA_df['missing_example'].isnull()
```

Out[230…
```
0     True
1     True
2    False
3    False
4     True
5    False
Name: missing_example, dtype: bool
```

In [231…
```
# Answer here
# .isnull() cannot detect C, D, and F as missing, because they are stored
```

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.

In [232…
```
X.duplicated()
```

Out[232…
```
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:

In [233…
```
sum(X.duplicated())
```

Out[233…
```
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.

In [234…
```
sum(X.duplicated('text'))
```

Out[234…
```
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.

In [235…
```
dummy_duplicate_dict = [{
                            'text': 'dummy record',
                            'category': 1,
                            'category_name': "dummy category"
                        },
                        {
                            'text': 'dummy record',
                            'category': 1,
                            'category_name': "dummy category"
                        }]
```

In [236…
```
X = pd.concat([X, pd.DataFrame(dummy_duplicate_dict)], ignore_index=True)
```

In [237…
```
len(X)
```

Out[237…
```
2259
```

In [238…
```
sum(X.duplicated())
```

Out[238… 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:

In [239…
```
X.drop_duplicates(keep=False, inplace=True) # inplace applies changes dir
```

In [240…
```
len(X)
```

Out[240… 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.

In [241…
```
X_sample = X.sample(n=1000) #random state
```

In [242…
```
len(X_sample)
```

Out[242… 1000

In [243…
```
X_sample[0:4]
```

Out[243…

	text	category	category_name
2047	From: stark@dwovax.enet.dec.com (Todd I. Stark...	2	sci.med
1385	From: clipper@mccarthy.csd.uwo.ca (Khun Yee Fu...	1	comp.graphics
417	From: rgc3679@bcstec.ca.boeing.com (Robert G. ...	1	comp.graphics
1299	From: andreasa@dhhalden.no (ANDREAS ARFF) Subj...	1	comp.graphics

---

>>> Exercise 6 (take home):

Notice any changes from the `X` dataframe to the `X_sample` 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.

In [244…
```
# Answer here
# The values in each row of `X_sample` are the same as those in `X`,
# but the row order is irregular since they were randomly sampled.
# As for the column structure, the items `text`, `category`, and `categor
```

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.

In [245…
```
import matplotlib.pyplot as plt
%matplotlib inline
```

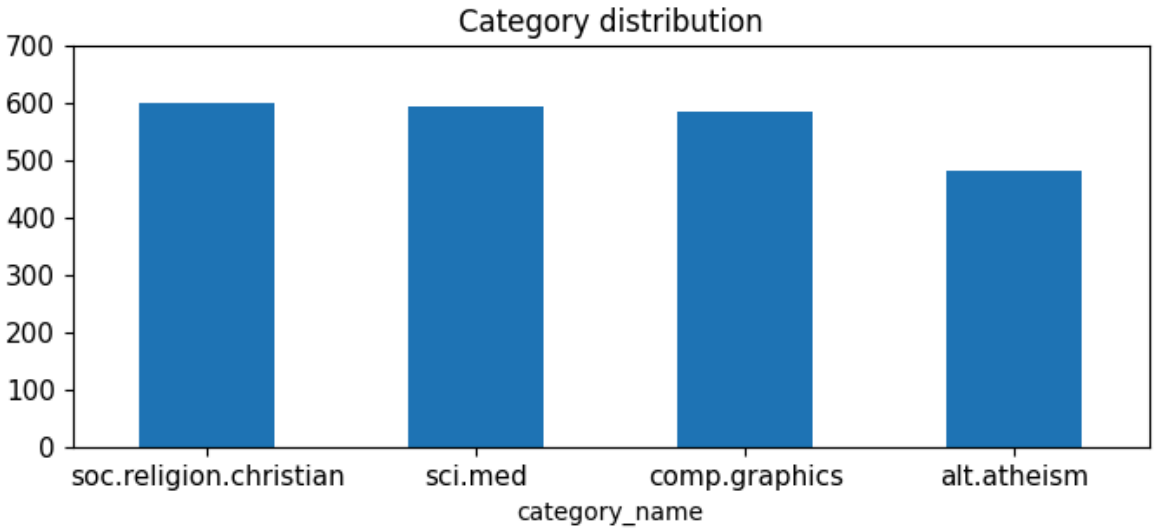
In [246…
```
categories
```

Out[246… ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']

In [247…
```
print(X.category_name.value_counts())

# plot barchart for X
X.category_name.value_counts().plot(kind = 'bar',
                                    title = 'Category distribution',
                                    ylim = [0, 700],
                                    rot = 0, fontsize = 11, figsize = (8,
```

```
category_name
soc.religion.christian    599
sci.med                   594
comp.graphics             584
alt.atheism               480
Name: count, dtype: int64
```

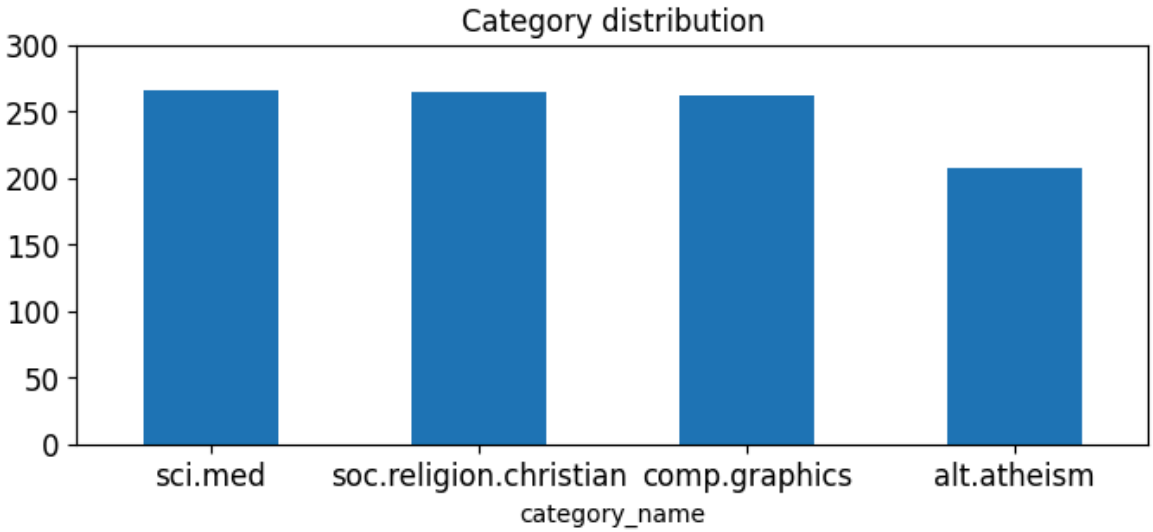
Out[247… <Axes: title={'center': 'Category distribution'}, xlabel='category_nam e'>



In [248…
```
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, figsiz
```

```
category_name
sci.med                   266
soc.religion.christian    264
comp.graphics             262
alt.atheism               208
Name: count, dtype: int64
```

Out[248… <Axes: title={'center': 'Category distribution'}, xlabel='category_name'>



You can use following command to see other available styles to prettify your charts.

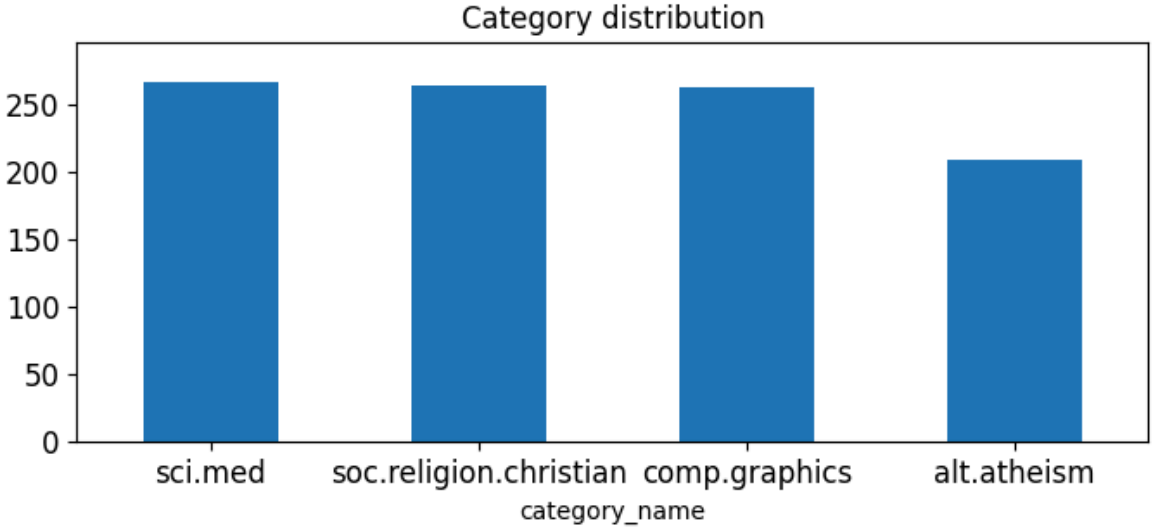
```
print(plt.style.available)
```

>>> Exercise 7 (Watch Video):

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)

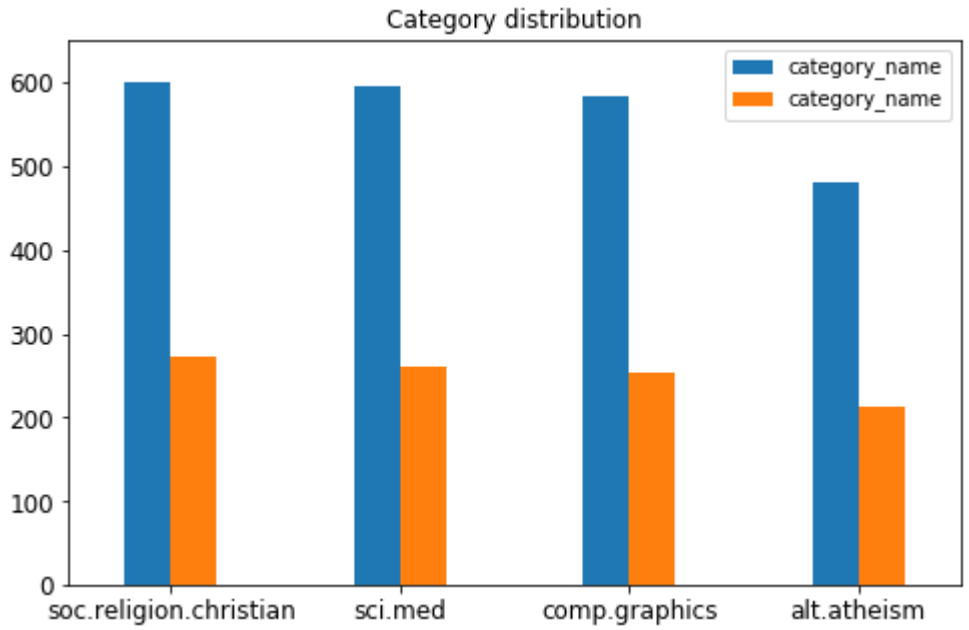
In [249…
```
# Answer here

X_sample.category_name.value_counts().plot(kind = 'bar',
                                           title = 'Category distribution
                                           ylim = [0,   _sample.category_
                                           rot = 0, fontsize = 12, figsiz
```

Out[249… <Axes: title={'center': 'Category distribution'}, xlabel='category_name'>



>>> 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.

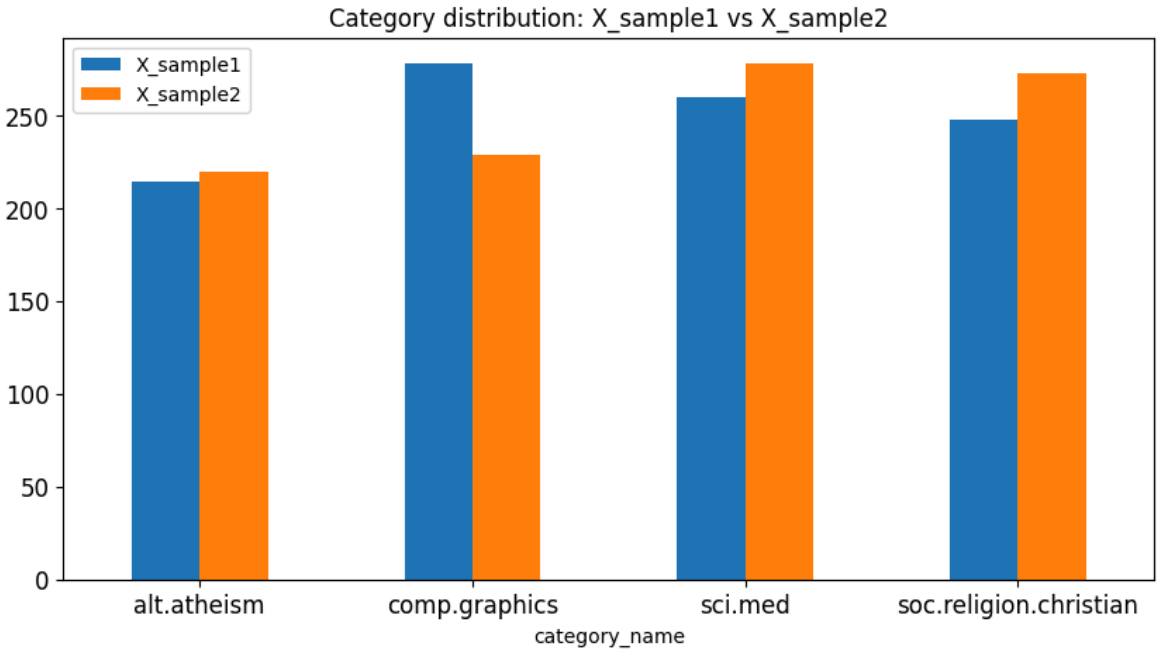


In [250…

```
# Answer here
X_sample1 = X.sample(n=1000, random_state=1)
X_sample2 = X.sample(n=1000, random_state=2)

print("Distribution in X_sample1:")
print(X_sample1.category_name.value_counts())

print("\nDistribution in X_sample2:")
print(X_sample2.category_name.value_counts())

dist_sample1 = X_sample1.category_name.value_counts().sort_index()
dist_sample2 = X_sample2.category_name.value_counts().sort_index()

dist_compare = pd.DataFrame({
    'X_sample1': dist_sample1,
    'X_sample2': dist_sample2
}).fillna(0)

print("\nSide-by-side distribution comparison:")
print(dist_compare)

dist_compare.plot(kind='bar',
                  title='Category distribution: X_sample1 vs X_sample2',
                  rot=0, fontsize=12, figsize=(10,5))
```

```
Distribution in X_sample1:
category_name
comp.graphics             278
sci.med                   260
soc.religion.christian    248
alt.atheism               214
Name: count, dtype: int64

Distribution in X_sample2:
category_name
sci.med                   278
soc.religion.christian    273
comp.graphics             229
alt.atheism               220
Name: count, dtype: int64

Side-by-side distribution comparison:
                        X_sample1  X_sample2
category_name
alt.atheism                   214        220
comp.graphics                 278        229
sci.med                       260        278
soc.religion.christian        248        273
```

Out[250… `<Axes: title={'center': 'Category distribution: X_sample1 vs X_sample 2'}, xlabel='category_name'>`



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
```

In [251…
```
import nltk
nltk.download("punkt")
nltk.download("punkt_tab")
```

```
[nltk_data] Downloading package punkt to /Users/parinmac/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     /Users/parinmac/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Out[251… True

In [252…
```
# takes a like a minute or two to process

X['unigrams'] = X['text'].apply(lambda x: dmh.tokenize_text(x))
```

In [253…
```
X[0:4]["unigrams"]
```

Out[253…
```
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.

In [254…
```
X[0:4]
```

Out[254…

	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....
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, ., ...

In [255…
```
list(X[0:1]['unigrams'])
```

Out[255… [['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',
                '.']]
```

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.

In [256…

```
from sklearn.feature_extraction.text import CountVectorizer

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

```
(np.int32(0), np.int32(14887))        1
(np.int32(0), np.int32(29022))        1
(np.int32(0), np.int32(8696)) 4
(np.int32(0), np.int32(4017)) 2
(np.int32(0), np.int32(33256))        2
(np.int32(0), np.int32(21661))        3
(np.int32(0), np.int32(9031)) 3
(np.int32(0), np.int32(31077))        1
(np.int32(0), np.int32(9805)) 2
(np.int32(0), np.int32(17366))        1
(np.int32(0), np.int32(32493))        4
(np.int32(0), np.int32(16916))        2
(np.int32(0), np.int32(19780))        2
(np.int32(0), np.int32(17302))        2
(np.int32(0), np.int32(23122))        1
(np.int32(0), np.int32(25663))        1
(np.int32(0), np.int32(16881))        1
(np.int32(0), np.int32(16082))        1
(np.int32(0), np.int32(23915))        1
(np.int32(0), np.int32(32142))        5
(np.int32(0), np.int32(33597))        2
(np.int32(0), np.int32(20253))        1
(np.int32(0), np.int32(587))  1
(np.int32(0), np.int32(12051))        1
(np.int32(0), np.int32(5201)) 1
:     :
(np.int32(0), np.int32(25361))        1
(np.int32(0), np.int32(25337))        1
(np.int32(0), np.int32(12833))        2
(np.int32(0), np.int32(5195)) 1
(np.int32(0), np.int32(27836))        1
(np.int32(0), np.int32(18474))        1
(np.int32(0), np.int32(32270))        1
(np.int32(0), np.int32(9932)) 1
(np.int32(0), np.int32(15837))        1
(np.int32(0), np.int32(32135))        1
(np.int32(0), np.int32(17556))        1
(np.int32(0), np.int32(4378)) 1
(np.int32(0), np.int32(26175))        1
(np.int32(0), np.int32(9338)) 1
(np.int32(0), np.int32(33572))        1
(np.int32(0), np.int32(31915))        1
(np.int32(0), np.int32(177))  2
(np.int32(0), np.int32(2326)) 2
(np.int32(0), np.int32(3062)) 1
(np.int32(0), np.int32(35416))        1
(np.int32(0), np.int32(20459))        1
(np.int32(0), np.int32(14085))        1
(np.int32(0), np.int32(3166)) 1
(np.int32(0), np.int32(12541))        1
(np.int32(0), np.int32(230))  1
```

Now you can also see some examples of what each feature is based on their index in the vector:

In [257…
```
count_vect.get_feature_names_out()[14887]
```

Out[257…
```
'from'
```

In [258…
```
count_vect.get_feature_names_out()[29022]
```

Out[258…
```
'sd345'
```

In [259…
```
count_vect.get_feature_names_out()[8696]
```

Out[259…
```
'city'
```

In [260…
```
count_vect.get_feature_names_out()[4017]
```

Out[260…
```
'ac'
```

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 `analyzer`) 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.

In [261…
```
analyze = count_vect.build_analyzer()
analyze("I am craving for a hawaiian pizza right now")

# tokenization, remove stop words (e.g i, a, the), create n-gram (or unig
```

Out[261…
```
['am', 'craving', 'for', 'hawaiian', 'pizza', 'right', 'now']
```

>>> Exercise 9 (Watch Video):

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

In [262…
```
# Answer here
# How do we turn our array[0] text document into a tokenized text using t
analyze(X.text[0])
```

Out[262… ['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',
'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',
 'collier',
 'uk',
 'ac',
 'city',
 'the',
 'city',
 'university',
 'tel',
 '071',
 '477',
 '8000',
 'x3769',
 'london',
 'fax',
 '071',
 '477',
 '8565',
 'ec1v',
 '0hb']
```

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

In [263…
```
# We can check the shape of this matrix by:
X_counts.shape
```

Out[263…
```
(2257, 35788)
```

In [264…
```
# 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]
```

Out[264…
```
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.

In [265…
```
X_counts.shape
```

Out[265…
```
(2257, 35788)
```

In [266…
```
# we convert from sparse array to normal array
X_counts[0:5, 0:100].toarray()
```

Out[266…
```
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]])
```

In [267…
```
count_vect.get_feature_names_out()[0:1]
```

Out[267…
```
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.

In [268…
```
# Answer here
row5 = X_counts[4]

nonzero_indices = row5.nonzero()[1]

feature_names = count_vect.get_feature_names_out()

print("One example word in the 5th record:", feature_names[nonzero_indice
#

words_in_row5 = [feature_names[i] for i in nonzero_indices]
print("Words in the 5th record:", words_in_row5)
#
```

```
One example word in the 5th record: subject
Words in the 5th record: ['from', 'subject', 'to', 'hp', 'organization',
'the', 'lines', '14', 'of', 'like', 'is', 'this', 'in', 'edu', 'which', 'f
or', 'made', 'then', 'it', 'as', 'there', 're', 'at', 'so', 'and', 'that',
'on', 'your', 'are', 'you', 'find', '15', 'one', 'believe', 'go', 'our',
'who', 'stanly', 'grok11', 'columbiasc', 'ncr', 'com', 'elder', 'brother',
'corp', 'columbia', 'sc', 'article', 'apr', '00', '57', '41', '1993', '282
46', 'athos', 'rutgers', 'rexlex', 'fnal', 'gov', 'writes', '01', '56', '2
2824', 'shrum', 'hpfcso', 'fc', 'matt', '22', 'therefore', 'main', 'highwa
ys', 'many', 'invite', 'wedding', 'feast', 'hmmmmmm', 'sounds', 'theolog
y', 'christ', 'odds', 'am', 'parable', 'jesus', 'tells', 'kingdom', 'heave
n', 'unto', 'certain', 'king', 'marriage', 'his', 'son', 'clothes', 'wer
e', 'customary', 'given', 'those', 'chose', 'attend', 'man', 'refused', 'w
ear', 'equalivant', 'righteousness', 'when', 'died', 'sins', 'provided',
'decision', 'put']
```

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.

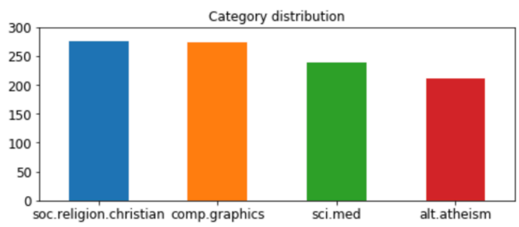
In [269…
```
# first twenty features only
plot_x = ["term_"+str(i) for i in count_vect.get_feature_names_out()[0:20
```

In [270…
```
# obtain document index
plot_y = ["doc_"+ str(i) for i in list(X.index)[0:20]]
```

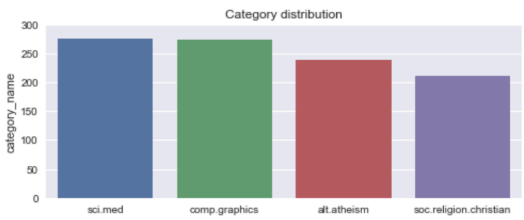
In [271…
```
plot_z = X_counts[0:20, 0:20].toarray() #X_counts[how many documents, how
plot_z
```

Out[271…
```
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

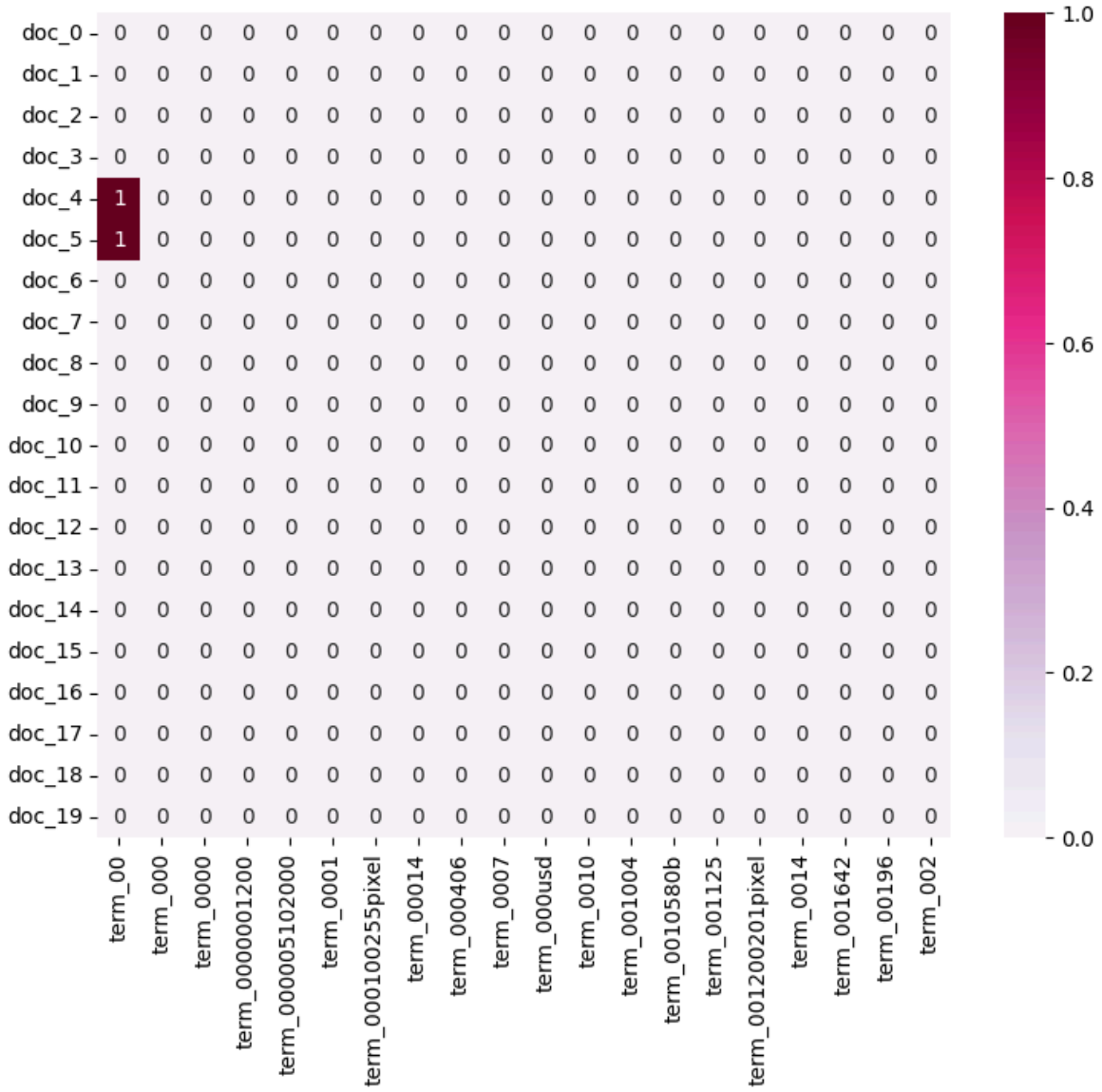


By Seaborn

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!

In [272…
```
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/

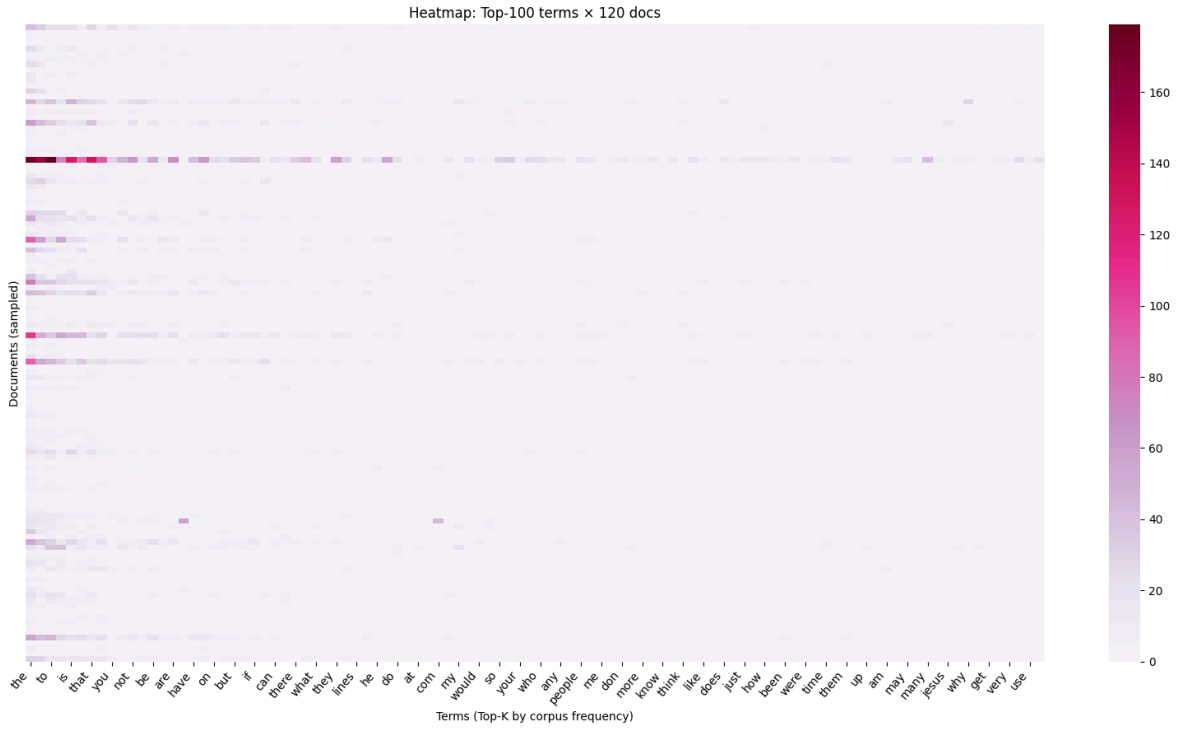
>>> Exercise 11 (take home):

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

In [273…

```
# Answer here
# I think the best way to visualize the term-document matrix is not to pl
# since it's too big and mostly zeros. Instead, I'd just pick the top K m
# That way I only need to turn a small part into a normal table and draw
# Another simple option is to use histograms to show how often words or d
```

```
import numpy as np, pandas as pd, seaborn as sns, matplotlib.pyplot as pl

K = 100                        # Top-K terms
N = 120                        # sampled docs
term_freq = np.asarray(X_counts.sum(axis=0)).ravel()
topk_idx = term_freq.argsort()[::-1][:K]
doc_idx = np.random.default_rng(42).choice(X_counts.shape[0], size=min(N,

sub = X_counts[doc_idx][:, topk_idx]
terms = count_vect.get_feature_names_out()
plot_x = [terms[j] for j in topk_idx]
plot_y = [f"doc_{i}" for i in doc_idx]
df_todraw = pd.DataFrame(sub.toarray(), columns=plot_x, index=plot_y)

plt.figure(figsize=(16, 9))
sns.heatmap(df_todraw, cmap="PuRd", vmin=0, vmax=df_todraw.values.max(),
plt.title(f"Heatmap: Top-{K} terms × {len(doc_idx)} docs")
plt.xlabel("Terms (Top-K by corpus frequency)")
plt.ylabel("Documents (sampled)")
plt.xticks(rotation=50, ha="right")
plt.yticks([], [])
plt.tight_layout()
plt.show()
```



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 truly inspires me to practise Data Science. That's the motivation right there!

5.4 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!

5.4.1 Transform Text Data

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**

In [274…

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

#[3, 8, 5, 2, 5, 8, 2, 5, 3, 2]
```

```
---------------------------------------------------------------------------
_
KeyboardInterrupt                         Traceback (most recent call las
t)
Cell In[274], line 4
      2 term_frequencies = []
      3 for j in range(0,X_counts.shape[1]):
----> 4     term_frequencies.append(sum(X_counts[:,j].toarray()))
      6 #[3, 8, 5, 2, 5, 8, 2, 5, 3, 2]

KeyboardInterrupt:
```

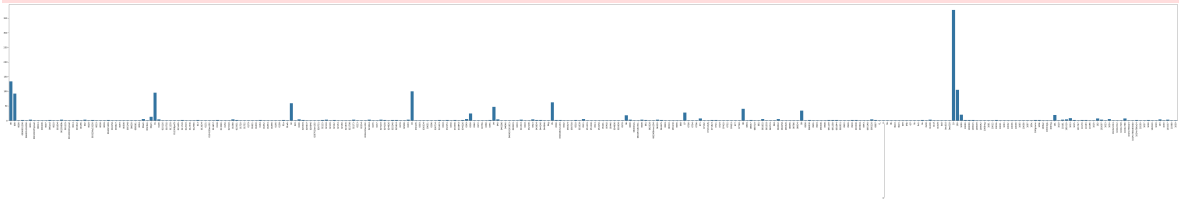
In [ ]:
```
term_frequencies = np.asarray(X_counts.sum(axis=0))[0]
```

In [ ]:
```
term_frequencies[0] #sum of first term: 00
```

Out[ ]:
```
np.int64(134)
```

In [ ]:
```
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
```

```
/var/folders/fq/qps9vdbd4bl6f7tzpv5gfdpm0000gn/T/ipykernel_59293/348057997
1.py:4: UserWarning: set_ticklabels() should only be used with a fixed num
ber of ticks, i.e. after set_ticks() or using a FixedLocator.
  g.set_xticklabels(count_vect.get_feature_names_out()[:300], rotation = 9
0);
```



>>> Exercise 12 (take home):

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

In [ ]:
```
# Answer here
import plotly.express as px

df_plot = pd.DataFrame({
    "term": count_vect.get_feature_names_out()[:300],
    "frequency": term_frequencies[:300]
})

fig = px.bar(df_plot, x="term", y="frequency", title="Top 300 Term Freque
fig.show()
```

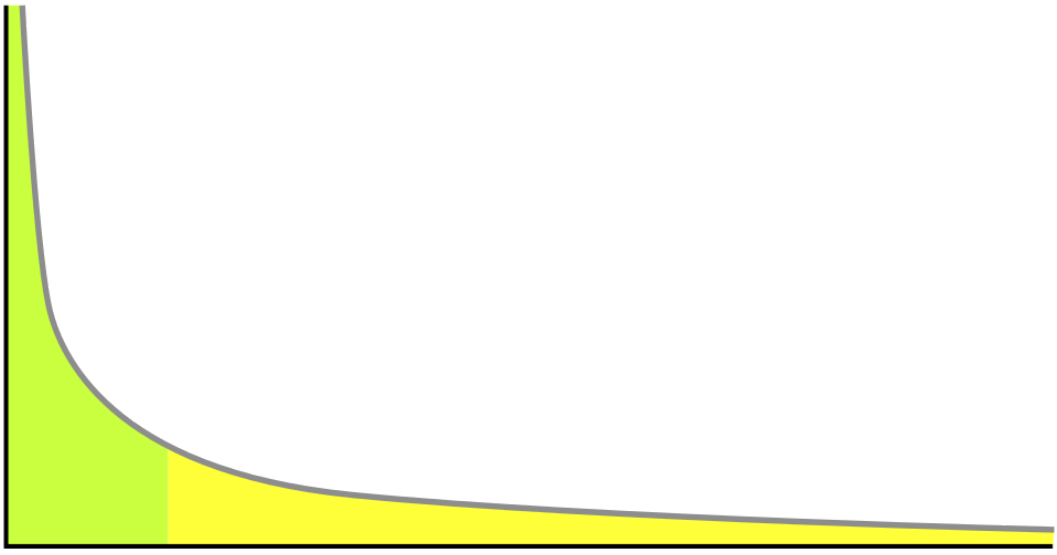
>>> Exercise 13 (take home):

The chart above only contains 300 vocabulary in the documents, and it's already computationally intensive to both compute and visualize. Can you efficiently reduce the number of terms you want to visualize as an exercise.

In [ ]:
```
# Answer here
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

top_k = 50
terms = count_vect.get_feature_names_out()[:top_k]
frequencies = term_frequencies[:top_k]

import plotly.express as px

df_plot = pd.DataFrame({"term": terms, "frequency": frequencies})
fig = px.bar(df_plot, x="term", y="frequency",
             title=f"Top {top_k} Term Frequencies")
fig.show()
```

>>> Exercise 14 (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



In [ ]:
```
# Answer here
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

df_plot = pd.DataFrame({
    "term": count_vect.get_feature_names_out(),
    "frequency": term_frequencies
```

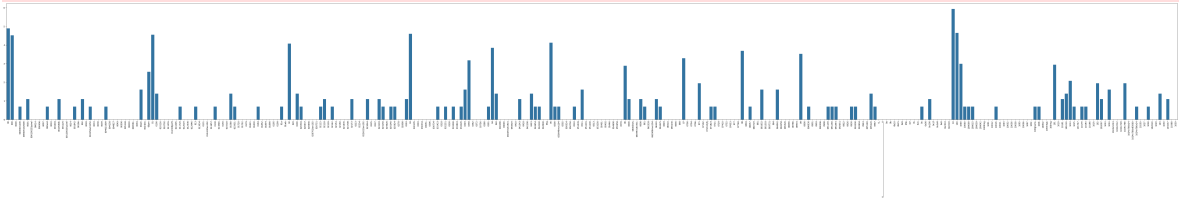
```
})

df_plot_sorted = df_plot.sort_values(by="frequency", ascending=False).hea

fig = px.bar(df_plot_sorted, x="term", y="frequency",
             title="Top 50 Term Frequencies (Sorted by Frequency)")
fig.show()
```

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!

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

In [ ]:
```
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
```

```
/var/folders/fq/qps9vdbd4bl6f7tzpv5gfdpm0000gn/T/ipykernel_59293/216654899
8.py:4: UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. af
ter set_ticks() or using a FixedLocator.
```



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!

>>> Exercise 15 (take home):

You can copy the code from the previous exercise and change the 'term_frequencies' variable for the 'term_frequencies_log', comment about the differences that you observe and talk about other possible insights that we can get from a log distribution.

In [ ]:
```
# Answer here:
# After we used the log transformation, the chart looks a lot more balanc
# In the original plot, the super frequent words were too big and basical
# so it was hard to notice the smaller ones. With the log scale, the gap
```

```
# and we can actually see more of the long tail.
# This makes the chart easier to read and also lets us notice some patter
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

df_plot = pd.DataFrame({
    "term": count_vect.get_feature_names_out(),
    "frequency": term_frequencies_log
})

df_plot_sorted = df_plot.sort_values(by="frequency", ascending=False).hea

fig = px.bar(df_plot_sorted, x="term", y="frequency",
             title="Top 50 Term Frequencies log (Sorted by Frequency)")
fig.show()
```

End of Phase 1

The phase 1 exercises and homeworks should be committed and submitted before September 28th

---

Phase 2

5.4.2 Finding frequent patterns

Perfect, so now that we know how to interpret a document-term matrix from our text data, we will see how to get extra insight from it, we will do this by mining frequent patterns. For this we will be using the PAMI library that we previously installed.

Introduction to PAMI

PAMI (PAttern MIning) is a Python-based library designed to empower data scientists by providing the necessary tools to uncover hidden patterns within large datasets. Unlike other pattern mining libraries that are Java-based (such as WEKA and SPMF), PAMI caters specifically to the Python environment, making it more accessible for data scientists working with Big Data. The goal of PAMI is to streamline the process of discovering patterns that are often hidden within large datasets, offering a unified platform for applying various pattern mining techniques. In the library you can find a lot of implementations from current state-of-the-art algorithms, all of them cater to different type of data, they can be: transactional data, temporal data, utility data and some others. You can find more information in the following github: PAMI. For the purpose of our lab we will be modeling our text data as a transactional type. So let's get into it.

Some code cells might have changed slightly from last year's explanation due to some updates of PAMI or fixes during the lab period of time

Transactional Data

In order to apply pattern mining techniques, we first need to convert our text data into transactional data. A transactional database is a set of transactions where each transaction consists of a unique identifier (TID) and a set of items. For instance, think of a transaction as a basket of items purchased by a customer, and the TID is like the receipt number. Each transaction could contain items such as "apple", "banana", and "orange".

Here's an example of a transactional database:

TID Transactions 1 a, b, c 2 d, e 3 a, e, f

In this structure: TID refers to the unique identifier of each transaction (often ignored by PAMI to save storage space). Items refer to the elements in each transaction, which could be either integers or strings (e.g., products, words, etc.). When preparing text data, we need to transform sentences or documents into a similar format, where each sentence or document becomes a transaction, and the words within it become the items.

Frequent Pattern Mining

After converting the text into a transactional format, we can then apply frequent pattern mining. This process identifies patterns or combinations of items that occur frequently across the dataset. For example, in text data, frequent patterns might be common word pairs or phrases that appear together across multiple documents. Important term to learn: **Minimum Support**: It refers to the minimum frequency that a transaction has to have to be considered a pattern in our scenario.

PAMI allows us to mine various types of patterns, but for the purpuse of this lab we will explore the following types:

Patterns Above Minimum Support: These are all patterns that meet a specified minimum support threshold. The result set can be quite large as it includes all frequent patterns, making it ideal for comprehensive analysis but potentially complex.

Maximal Frequent Patterns: These are the largest frequent patterns that cannot be extended by adding more items without reducing their frequency below the minimum support threshold. The result set is smaller and more concise, as it only includes the largest patterns, reducing redundancy.

Top-K Frequent Patterns: These patterns represent the K most frequent patterns, regardless of the minimum support threshold. The result set is highly focused and concise, with a fixed number of patterns, making it ideal when prioritizing the most frequent patterns.

Aspect	Patterns Above Minimum Support	Maximal Frequent Patterns	Top-K Frequent Patterns
Key Criteria	Minimum support threshold	Maximal frequent pattern	K most frequent patterns
Result Size	Large (all frequent patterns)	Smaller (only largest patterns)	Limited to k patterns
Redundancy	High (contains all patterns)	Low (no redundant subsets)	Low (only top-k patterns)
Use Case	Comprehensive frequent pattern mining	Focused analysis with reduced output	Prioritizing most frequent patterns
Output Complexity	Potentially complex and large	More concise	Very concise

In the following steps, we will guide you through how to convert text data into transactional form and mine frequent patterns from it.

In our scenario, what we need is to mine patterns that can be representative to **each category**, in this way we will be able to differentiate each group of data more easily, for that we will need to first modify our document-term matrix to be able to work for each category, for this we will do the following:

In [ ]:
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

#Create separate DataFrames for each category
categories = X['category_name'].unique()  # Get unique category labels
category_dfs = {}  # Dictionary to store DataFrames for each category

for category in categories:
    # Filter the original DataFrame by category
    category_dfs[category] = X[X['category_name'] == category].copy()

# Function to create term-document frequency DataFrame for each category
def create_term_document_df(df):
    count_vect = CountVectorizer()  # Initialize the CountVectorizer
    X_counts = count_vect.fit_transform(df['text'])  # Transform the text

    # Get the unique words (vocabulary) from the vectorizer
    words = count_vect.get_feature_names_out()

    # Create a DataFrame where rows are documents and columns are words
    term_document_df = pd.DataFrame(X_counts.toarray(), columns=words)

    return term_document_df

# Create term-document frequency DataFrames for each category
filt_term_document_dfs = {}  # Dictionary to store term-document DataFram

for category in categories:
    filt_term_document_dfs[category] = create_term_document_df(category_d
```

In [ ]:
```
# Display the filtered DataFrame for one of the categories, feel free to
category_number=0 #You can change it from 0 to 3
```

```
print(f"Filtered Term-Document Frequency DataFrame for Category {categori
filt_term_document_dfs[categories[category_number]]
```

```
Filtered Term-Document Frequency DataFrame for Category comp.graphics:
```

Out[ ]:

	00	000	000005102000	000100255pixel	0007	000usd	0010580b	0012
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	
579	0	0	0	0	0	0	0	
580	0	0	0	0	0	0	0	
581	0	0	0	0	0	0	0	
582	0	0	0	0	0	0	0	
583	0	0	0	0	0	0	0	

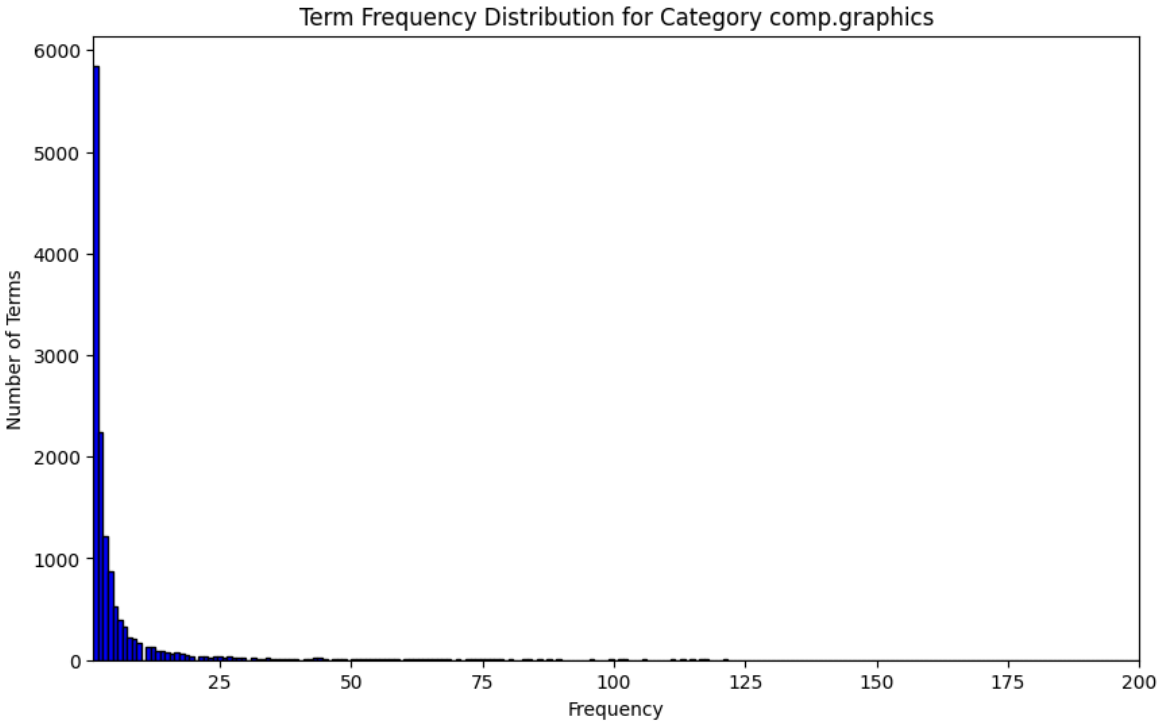
584 rows × 13756 columns

Now we can see the number of unique words per category based on the column number in the new dataframe, feel free to **explore the changes of each category changing the vector number at the end**.

In the past sections we saw the behaviour of each word frequency in the documents, but we still want to generalize a little bit more so we can observe and determine the data that we are going to use to mine the patterns. For this we will group the terms in bins and we are going to plot their frequency. Again, feel free to change the category number to explore the different results.

In [ ]:

```
 # Sum over all documents to get total frequency for each word
category_number=0 #You can change it from 0 to 3
word_counts = filt_term_document_dfs[categories[category_number]].sum(axi

# Visualize the frequency distribution
plt.figure(figsize=(10, 6))
plt.hist(word_counts, bins=5000, color='blue', edgecolor='black')
plt.title(f'Term Frequency Distribution for Category {categories[category
plt.xlabel('Frequency')
plt.ylabel('Number of Terms')
plt.xlim(1, 200)
plt.show()
```



From this graph, we can see that most of the words appear very infrequently across the entire dataset, while a small number of words appear quite often. When we're trying to find patterns in text data, we focus on combinations of words that are most helpful for classifying the documents. However, very rare words or extremely common words (like stopwords: 'the,' 'in,' 'a,' 'of,' etc.) don't usually give us much useful information. To improve our results, we can filter out these words. Specifically, we'll remove the **bottom 1%** of the least frequent words and the **top 5%** of the most frequent ones. This helps us focus on words that might reveal more valuable patterns.

In this case, the choice of filtering the top 5% and bottom 1% is **arbitrary**, but in other applications, domain knowledge might guide us to filter words differently, depending on the type of text classification we're working on.

Let us look first at the words that we will be filtering based on the set percentage threshold.

In [ ]:
```
category_number=0 #You can change it from 0 to 3
word_counts = filt_term_document_dfs[categories[category_number]].sum(axi

# Sort the term frequencies in descending order
sorted_indices = np.argsort(word_counts)[::-1]  # Get indices of sorted f
sorted_counts = np.sort(word_counts)[::-1]  # Sort frequencies in descend

# Calculate the index corresponding to the top 5% most frequent terms
total_terms = len(sorted_counts)
top_5_percent_index = int(0.05 * total_terms)

# Get the indices of the top 5% most frequent terms
top_5_percent_indices = sorted_indices[:top_5_percent_index]

# Filter terms that belong to the top 5% based on their rank
filtered_words = [filt_term_document_dfs[categories[category_number]].ilo
```

```
print(f"Category: {categories[category_number]}")
print(f"Number of terms in top 5%: {top_5_percent_index}")
print(f"Filtered terms: {filtered_words}")
```

```
Category: comp.graphics
Number of terms in top 5%: 687
Filtered terms: ['the', 'to', 'of', 'and', 'is', 'in', 'for', 'it', 'fro
m', 'you', 'edu', 'that', 'on', 'this', 'or', 'be', 'with', 'have', 'line
s', 'can', 'subject', 'are', 'graphics', 'if', 'organization', 'image', 'a
s', 'not', 'but', 'at', 'there', 'com', 'an', 'any', 'will', 'by', 'univer
sity', 're', 'about', 'some', 'posting', 'file', 'do', 'all', 'would', 'ho
st', 'jpeg', 'what', 'so', 'nntp', 'has', 'files', 'which', 'one', 'also',
'me', 'use', 'software', 'was', 'my', 'images', 'writes', 'other', 'articl
e', 'out', 'data', 'program', 'know', 'like', 'version', 'more', 'color',
'ftp', 'your', 'get', 'computer', 'don', '3d', 'does', 'no', 'mail', 'form
at', 'they', 'available', 'need', 'we', 'ca', 'thanks', 'just', 'bit', 'gi
f', 'how', 'help', 'am', 'please', 'package', 'pub', 'anyone', 'very', 'in
formation', 'using', 'code', 'line', 'than', 'find', 'system', 'only', 'ti
me', 'where', '24', 'windows', 'good', 'uk', 'cs', 'display', 'then', 'poi
nts', 'up', 'ac', '___', 'these', 'etc', 'when', 'see', 'think', 've', 'vi
deo', 'reply', 'could', 'new', 'mac', 'looking', 'info', 'systems', 'its',
'email', 'should', 'them', 'free', 'distribution', 'want', 'been', 'used',
'point', 'world', 'much', 'comp', 'ibm', 'send', 'into', 'polygon', 'sgi',
'well', 'may', 'two', 'number', 'tiff', 'many', 'keywords', 'gov', 'work',
'problem', 'fax', 'screen', 'such', 'read', 'pc', 'library', 'programs',
'unix', 'ray', 'support', 'center', 'now', 'research', 'au', 'de', 'proces
sing', '16', 'their', 'directory', 'way', 'same', 'source', 'll', 'group',
'hi', 'dos', 'mode', 'under', 'based', 'address', 'vga', 'algorithm', 'ani
mation', 'those', 'formats', 'sun', 'here', '256', 'cc', 'inc', 'even', 'u
ser', '10', 'set', 'amiga', 'fast', 'card', 'look', 'write', 'he', 'most',
'ch', 'quality', 'postscript', 'run', 'since', 'book', 'better', '128', 'l
ot', 'contact', 'news', 'another', 'different', 'few', 'who', 'nasa', 'ove
r', 'real', 'hardware', 'cview', 'convert', 'colors', 'first', 'both', 'wi
thout', 'full', 'were', 'objects', 'programming', 'stuff', '20', 'surfac
e', 'had', 'analysis', 'called', 'washington', 'space', 'robert', 'got',
'standard', 'science', 'net', 'vesa', 'something', 'site', '15', 'post',
'quicktime', 'server', 'too', 'make', 'phone', 'include', 'question', 'any
body', 'original', 'internet', 'people', 'able', 'sphere', 'advance', 'vir
tual', '42', 'via', 'interested', 'either', 'because', 'viewer', 'co', 'cu
rrent', 'pov', 'technology', 'nl', 'archive', 'between', 'david', '11', 's
ay', 'each', 'ms', 'xv', 'wrote', 'try', 'newsreader', 'tar', 'last', 'ou
r', 'anything', 'found', 'uucp', 'name', 'tin', '17', 'sure', 'high', 'umi
ch', 'den', 'appreciated', 'per', 'best', 'cd', 'siggraph', 'view', 'work
s', 'shareware', 'must', 'faq', 'interface', 'else', 'running', '12', 'typ
e', 'non', 'sites', 'access', 'list', 'mark', 'p2', 'still', '30', 'chec
k', 'visualization', 'above', 'users', 'doesn', '1993', 'pixel', 'drawin
g', 'note', 'around', 'mil', 'simple', 'least', 'written', 'steve', '__',
'imagine', 'tools', 'probably', 'p3', 'take', 'following', 'zip', 'give',
'before', 'back', 'output', '3do', 'through', 'nice', 'object', 'however',
'off', 'driver', '13', 'dept', 'routines', 'go', '21', 'memory', 'change',
'little', 'machine', 'rayshade', 'might', 'needed', 'things', 'let', 'radi
us', 'bits', 'p1', 'text', 'possible', 'fi', 'author', 'reality', 'newsgro
up', 'packages', 'disk', 'conversion', 'actually', 'institute', 'end', 'se
veral', 'picture', 'digital', 'interactive', 'reference', 'public', 'micha
el', '14', 'right', 'australia', 'us', 'routine', 'usa', 'archie', 'readin
g', 'john', 'message', 'bbs', 'call', 'ed', 'technical', 'navy', 'thing',
'years', 'why', 'compression', 'dec', 'problems', 'case', 'project', 'appl
ications', 'drivers', 'input', 'product', '32', 'national', 'size', 'engi
n', 'create', 'summary', 'kind', 'frame', 'including', 'while', 'split',
'sources', 'being', 'though', 'level', 'hp', 'far', '18', 'form', 'state',
'given', 'canada', 'various', 'part', 'doing', '100', 'runs', 'algorithm
s', 'features', 'small', 'answers', 'edge', 'svga', 'own', 'every', 'someo
ne', 'rather', 'commercial', 'imaging', 'useful', 'engineering', 'quite',
'93', 'anonymous', 'chris', 'computing', 'model', 'course', 'box', 'prett
```

```
y', 'going', 'three', 'functions', 'cad', 'draw', 'radiosity', 'language',
'rendering', 'method', 'includes', 'requires', 'working', 'hope', 'databas
e', 'org', 'large', 'great', 'msdos', 'copy', 'books', 'true', 'old', 'win
dow', 'provides', 'displays', 'purdue', 'tel', 'platforms', 'really', 'han
dle', 'application', 'yes', 'already', 'sorry', 'earth', 'speed', 'current
ly', 'having', 'tool', 'plane', 'tell', 'double', 'uiuc', 'usc', 'order',
'se', 'resource', 'seems', 'built', '2d', 'others', '22', 'answer', '23',
'25', 'resolution', 'after', 'provide', 'big', 'next', 'enough', '00', 'we
ek', 'pictures', 'did', 'ask', 'pixels', 'polygons', 'material', 'environm
ent', 'easy', 'bad', 'volume', 'yet', 'example', 'stanford', 'documentatio
n', 'linux', 'scientific', '19', 'college', 'nfotis', 'wanted', 'domain',
'fine', 'tracing', 'gl', 'board', 'writing', 'peter', 'circle', 'said', 'm
enu', 'complete', 'int', 'faster', 'supports', 'greatly', 'tu', 'map', 'fu
nction', 'jfif', 'apple', 'second', 'sean', 'otis', 'st', 'general', 'depa
rtment', 'listing', 'usenet', 'rpi', 'made', 'art', 'lab', 'performance',
'groups', 'means', 'utilities', 'edges', 'uses', 'understand', 'save', 'ph
otoshop', 'scale', 'buy', 'questions', 'his', 'value', 'process', 'detail
s', 'manipulation', 'trying', 'values', 'ricardo', 'cis', 'thomas', 'aspec
ts', 'craig', 'mit', 'diablo', 'start', 'ab', 'network', 'heard', 'contain
ing', 'workstations', 'bitmap', 'black', 'known', 'viewing', 'viewers', 'c
ome', 'versions', 'printer', 'result', 'machines', 'allows', 'programmer',
'macintosh', 'developed', 'day', 'seen', 'thank', 'silicon', 'never', '13
0', 'hidden', 'similar', 'james', 'related', 'making', 'year', '____', 'ro
w', 'cards', 'op_rows', '50', 'request', 'univ', 'demo', 'op_cols', 'searc
h', 'section', 'mirror', 'raster', '27', 'page', 'mpeg', 'special', 'don
e', 'pcx', 'setting', 'california', 'hello', 'nz', 'speedstar', 'utexas',
'martin', 'ii', 'place', 'posted', 'getting', 'multi', 'open', 'distribute
d', '40']
```

Here we can explore the frequencies of the **top 5%** words:

In [ ]:
```
sorted_counts #We can see the frequencies sorted in a descending order
```

Out[ ]:
```
array([4537, 2775, 2470, ...,    1,    1,    1])
```

In [ ]:
```
sorted_indices #This are the indices corresponding to the words after bei
```

Out[ ]:
```
array([12266, 12390,  9021, ...,  7181,  7183,  6877])
```

In [ ]:
```
filt_term_document_dfs[categories[category_number]].loc[:,'the'].sum(axis
```

Out[ ]:
```
np.int64(4537)
```

In [ ]:
```
category_number=0 #You can change it from 0 to 3
word_counts = filt_term_document_dfs[categories[category_number]].sum(axi

# Sort the term frequencies in ascending order and get sorted indices
sorted_indices = np.argsort(word_counts)  # Get indices of sorted frequen
sorted_counts = word_counts[sorted_indices]  # Sort frequencies

# Calculate the index corresponding to the bottom 1% least frequent terms
total_terms = len(sorted_counts)
bottom_1_percent_index = int(0.01 * total_terms)

# Get the indices of the bottom 1% least frequent terms
bottom_1_percent_indices = sorted_indices[:bottom_1_percent_index]

# Filter terms that belong to the bottom 1% based on their rank
filtered_words = [filt_term_document_dfs[categories[category_number]].ilo
```

```
print(f"Category: {categories[category_number]}")
print(f"Number of terms in bottom 1%: {bottom_1_percent_index}")
print(f"Filtered terms: {filtered_words}")
```

```
Category: comp.graphics
Number of terms in bottom 1%: 137
Filtered terms: ['initworld', 'jkpg', 'jiu1', 'jiu', 'jfreund', 'jeremy',
'jena', 'jem', 'jele', 'jeffrey', 'jbalgley', 'jasper', 'jancene', 'jagua
r', 'jaggies', 'jagged', 'jaclyn', 'jacky', 'ja', 'ixos', 'ixels', 'ix',
'ivr', 'ivnorm', 'iv2scn', 'itri', 'itor', 'itnsg1', 'jman', 'itn', 'jna',
'joes', 'jump', 'juelin', 'judge', 'juan', 'jroberts', 'jr', 'jpsrc4', 'jp
lpost', 'jpgs', 'jpg95', 'jpg75', 'jpg5o', 'jpg50', 'jpg25', 'jpegv4', 'jp
egsrc4', 'jpeged', 'jpeg4bin', 'jpeg4386', 'journalix', 'josephson', 'jon
g', 'joint', 'join', 'johnm', 'johnl', 'jogle', 'jobs', 'jun', 'ithil', 'i
terate', 'ipsc2', 'ipcs', 'ipa', 'ious', 'ior', 'ions', 'iol', 'io', 'invi
ted', 'invisible', 'investment', 'investigator', 'investigation', 'investi
gating', 'inversion', 'inventing', 'invent', 'invariants', 'invariably',
'inumerable', 'intuitive', 'introducing', 'intricacies', 'intervention',
'interval', 'interpretor', 'interpretations', 'ipu', 'iteration', 'ipxs',
'irc', 'italian', 'israel', 'isotrack', 'isophotes', 'isophotal', 'isolati
on', 'isodata', 'iso9660', 'iso', 'isneeded', 'islanddraw', 'island', 'isg
tec', 'isg', 'isdres', 'iscsvax', 'isabel', 'irz205', 'irz', 'irt', 'irrit
ated', 'irretrievably', 'ironically', 'ironduke', 'iron', 'irit2scn', 'ire
land', 'irau40', 'interpret', 'junior', 'jview090', 'lai', 'lady', 'lace
y', 'lac', 'laboimage_', 'laboimage', 'labeled', 'lab2', 'laaksone', 'l4
v', 'l300', 'l14h11']
```

Here we can explore the frequencies of the **bottom 1%** words:

In [ ]:
```
sorted_counts #We can see the frequencies sorted in an ascending order
```

Out[ ]:
```
array([   1,    1,    1, ..., 2470, 2775, 4537])
```

In [ ]:
```
sorted_indices #This are the indices corresponding to the words after bei
```

Out[ ]:
```
array([ 6877,  7183,  7181, ...,  9021, 12390, 12266])
```

In [ ]:
```
filt_term_document_dfs[categories[category_number]].loc[:,'l14h11'].sum(a
```

Out[ ]:
```
np.int64(1)
```

Well done, now that we have seen what type of words are inside the thresholds we set, then we can procede to **filter them out of the dataframe**. If you want to experiment after you complete the lab, you can return to try different percentages to filter, or not filter at all to do all the subsequent tasks for the pattern minings, and see if there is a significant change in the result.

In [ ]:
```
category_number=0 #You can change it from 0 to 3

# Filter the bottom 1% and top 5% words based on their sum across all doc
def filter_top_bottom_words_by_sum(term_document_df, top_percent=0.05, bo
    # Calculate the sum of each word across all documents
    word_sums = term_document_df.sum(axis=0)

    # Sort the words by their total sum
    sorted_words = word_sums.sort_values()
```

```
    # Calculate the number of words to remove
    total_words = len(sorted_words)
    top_n = int(top_percent * total_words)
    bottom_n = int(bottom_percent * total_words)

    # Get the words to remove from the top 5% and bottom 1%
    words_to_remove = pd.concat([sorted_words.head(bottom_n), sorted_word
    print(f'Bottom {bottom_percent*100}% words: \n{sorted_words.head(bott
    print(f'Top {top_percent*100}% words: \n{sorted_words.tail(top_n)}')
    # Return the DataFrame without the filtered words
    return term_document_df.drop(columns=words_to_remove)

# Apply the filtering function to each category
term_document_dfs = {}

for category in categories:
    print(f'\nFor category {category} we filter the following words:')
    term_document_dfs[category] = filter_top_bottom_words_by_sum(filt_ter

# Example: Display the filtered DataFrame for one of the categories
print(f"Filtered Term-Document Frequency DataFrame for Category {categori
term_document_dfs[categories[category_number]]
```

```
For category comp.graphics we filter the following words:
Bottom 1.0% words:
initworld    1
jkpg         1
jiu1         1
jiu          1
jfreund      1
            ..
lab2         1
laaksone     1
l4v          1
l300         1
l14h11       1
Length: 137, dtype: int64
Top 5.0% words:
40              27
distributed     27
open            27
multi           27
getting         27
               ...
is            1751
and           2382
of            2470
to            2775
the           4537
Length: 687, dtype: int64

For category soc.religion.christian we filter the following words:
Bottom 1.0% words:
disparate       1
expose          1
explosive       1
resisting       1
exploitation    1
               ..
fence           1
feminist        1
remarriage      1
remarried       1
remeber         1
Length: 138, dtype: int64
Top 5.0% words:
pagan        36
claims       36
gave         36
parts        36
jr           37
           ...
that       4393
and        4409
to         6113
of         6377
the       11200
Length: 693, dtype: int64

For category sci.med we filter the following words:
Bottom 1.0% words:
íålittin        1
icl             1
```

```
icgln            1
iceskate         1
icemt            1
                ..
ineligible       1
inetractive      1
inexcusable      1
inexperienced    1
infallible       1
Length: 162, dtype: int64
Top 5.0% words:
toxic        29
genetic      29
sun          29
answer       29
rose         29
            ...
in         2993
and        3450
to         4085
of         4560
the        6854
Length: 812, dtype: int64

For category alt.atheism we filter the following words:
Bottom 1.0% words:
zyklon          1
plagerize       1
disqualified    1
disproving      1
disproven       1
               ..
dressed         1
perfecetly      1
donate          1
dregs           1
dreams          1
Length: 119, dtype: int64
Top 5.0% words:
court      36
lot        37
among      37
days       37
three      37
          ...
that     3158
is       3530
to       4249
of       4253
the      7234
Length: 598, dtype: int64
Filtered Term-Document Frequency DataFrame for Category comp.graphics:
```

Out[ ]:

	000	000005102000	000100255pixel	0007	000usd	0010580b	0012002(
0	0	0	0	0	0	0	
1	0	0	0	0	0	0	
2	0	0	0	0	0	0	
3	0	0	0	0	0	0	
4	0	0	0	0	0	0	
...	...	...	...	...	...	...	
579	0	0	0	0	0	0	
580	0	0	0	0	0	0	
581	0	0	0	0	0	0	
582	0	0	0	0	0	0	
583	0	0	0	0	0	0	

584 rows × 12932 columns

>>> Exercise 16 (take home):

Review the words that were filtered in each category and comment about the differences and similarities that you can see.

In [ ]:
```
# Answr here:
# The top 5% words were mostly common or functional words
# such as the, to, of, and, in
# these appear too often and don't help distinguish between topics.

# The bottom 1% words were mostly rare, misspelled,
# or very specific terms like ibuprophen or faulkner.
# They occur only once or twice and add little value.

# both very frequent and very rare words are not useful for finding patte
# Removing them helps keep the focus on mid-frequency words
# that are more meaningful for text analysis.
```

Great! Now that our document-term frequency dataframe is ready, we can proceed with the frequent pattern mining process. To do this, we first need to convert our dataframe into a transactional database that the PAMI library can work with. We will generate a CSV file for each category to create this database.

A key step in this process is defining the threshold that determines when a value in the data is considered a transaction. As we observed in the previous cell, there are **many zeros** in our dataframe, which indicate that certain words do not appear in specific documents. With this in mind, we'll set the transactional threshold to be **greater than or equal to 1**. This means that for each document/transaction, we will include all the words that occur at least once (after filtering), ensuring that only

relevant words are included in the pattern mining process. For your reference you can also check the following real world example that the PAMI library provides to review how they chose the threshold to generate the transactional data: Air Pollution Analytics - Japan.

The next part of the code will take a couple of minutes to execute, for simplicity I already shared the resulting files from it, to continue onwards

Given that some students have been experiencing some errors with recent newer versions of PAMI after Oct 11, where they changed some directories of these functions, you can try to run the following block of code or uncomment the lines indicated inside to run the older version of the functions:

In [ ]:
```
from PAMI.extras.convert.DF2DB import DF2DB

# Loop through the dictionary of term-document DataFrames
for category in term_document_dfs:
    # Replace dots with underscores in the category name to avoid errors
    category_safe = category.replace('.', '_')

    # Create the DenseFormatDF object and convert to a transactional data
    obj = DF2DB(term_document_dfs[category])

    obj.convert2TransactionalDatabase(f'td_freq_db_{category_safe}.csv',
```

If you encounter errors when running the above codes, try commenting out the above codes and running the code in this box. This error may comes from the update of the source code from the PAMI library.

```
from PAMI.extras.DF2DB import DenseFormatDF as db

# Loop through the dictionary of term-document DataFrames
for category in term_document_dfs:
    # Replace dots with underscores in the category name to avoid
errors in the file creation
    category_safe = category.replace('.', '_')

    # Create the DenseFormatDF object and convert to a
transactional database
    obj = db.DenseFormatDF(term_document_dfs[category])


obj.convert2TransactionalDatabase(f'td_freq_db_{category_safe}.csv',
'>=', 1)
```

Now let us look into the stats of our newly created transactional databases, we will observe the following:

- **Database Size (Total Number of Transactions)**: Total count of transactions in the dataset.

- **Number of Items**: Total count of unique items available across all transactions.
- **Minimum Transaction Size**: Smallest number of items in any transaction, indicating the simplest transaction.
- **Average Transaction Size**: Mean number of items per transaction, showing the typical complexity.
- **Maximum Transaction Size**: Largest number of items in a transaction, representing the most complex scenario.
- **Standard Deviation of Transaction Size**: Measures variability in transaction sizes; higher values indicate greater diversity.
- **Variance in Transaction Sizes**: Square of the standard deviation, providing a broader view of transaction size spread.
- **Sparsity**: Indicates the proportion of possible item combinations that do not occur, with values close to 1 showing high levels of missing combinations.

With regards to the graphs we will have:

- **Item Frequency Distribution**
  - Y-axis (Frequency): Number of transactions an item appears in.
  - X-axis (Number of Items): Items ranked by frequency.
- **Transaction Length Distribution**
  - Y-axis (Frequency): Occurrence of transaction lengths.
  - X-axis (Transaction Length): Number of items per transaction.

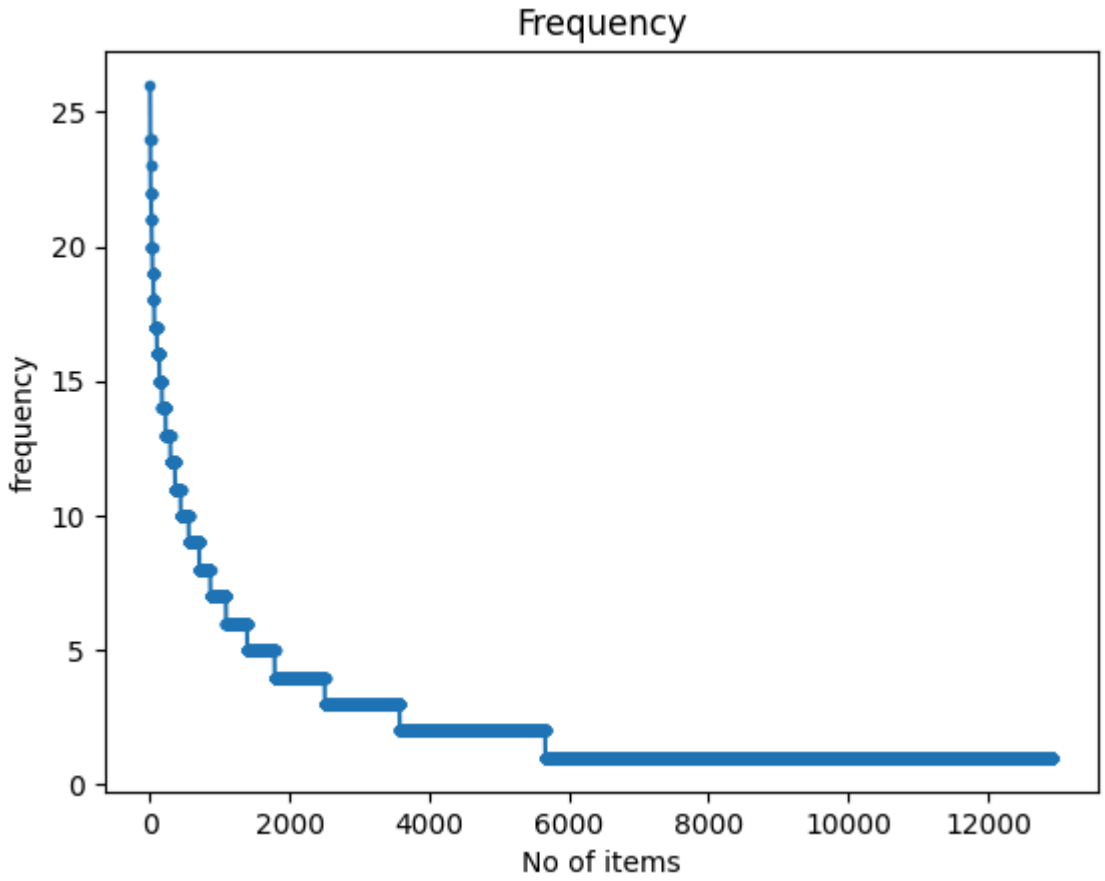
If you encounter errors when running the subsequent codes due to UTF-8 encoding, try running the codes in this box first

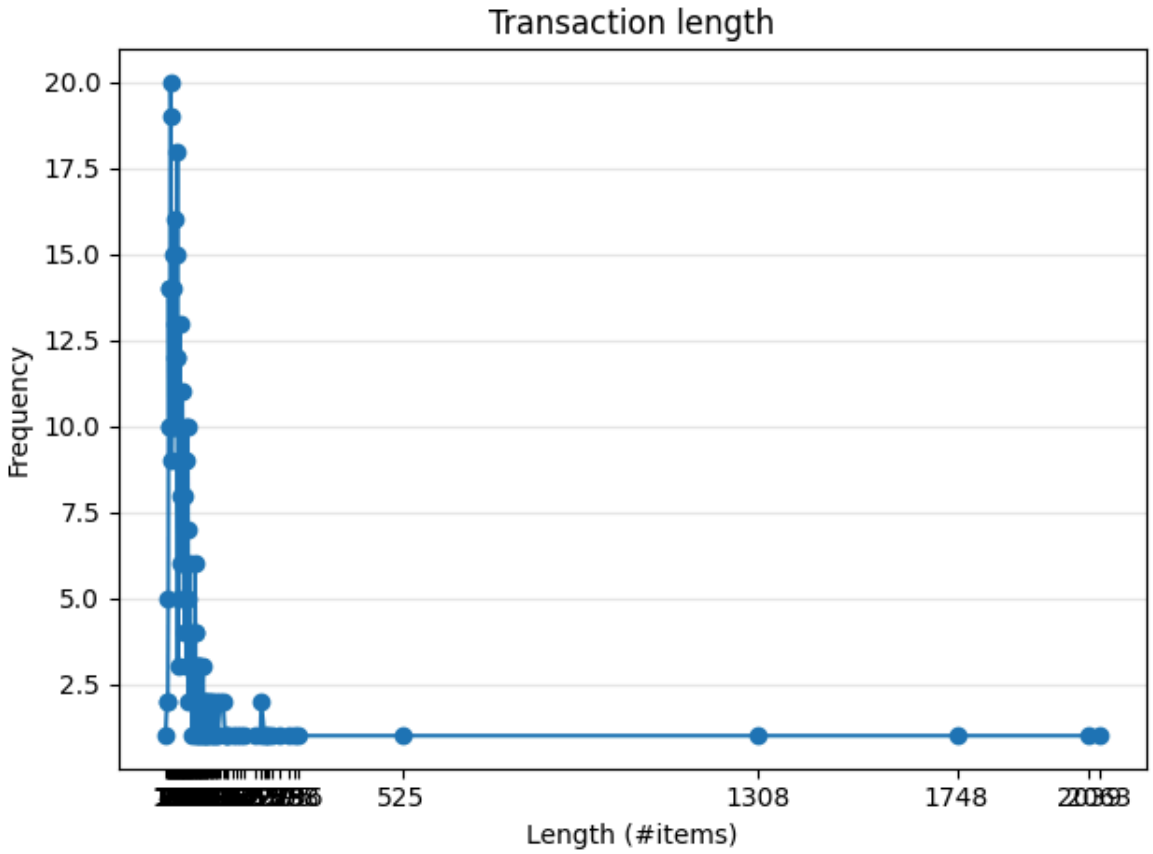
```
import builtins

_orig_open = open

def safe_open(*args, **kwargs):
    if len(args) > 0 and isinstance(args[0], str) and
args[0].endswith('.csv'):
        kwargs['encoding'] = 'latin-1'   # Force Latin-1
        kwargs['errors'] = 'ignore'      # Ignore bad characters
    return _orig_open(*args, **kwargs)


builtins.open = safe_open
```

In [ ]:
```
from PAMI.extras.dbStats import TransactionalDatabase as tds
obj = tds.TransactionalDatabase('td_freq_db_comp_graphics.csv')
obj.run()
```

```
obj.printStats()
obj.plotGraphs()
```

```
Database size (total no of transactions) : 584
Number of items : 12932
Minimum Transaction Size : 4
Average Transaction Size : 56.41267123287671
Maximum Transaction Size : 2063
Standard Deviation Transaction Size : 152.5206450557005
Variance in Transaction Sizes : 23302.44862132569
Sparsity : 0.995637745806304
```

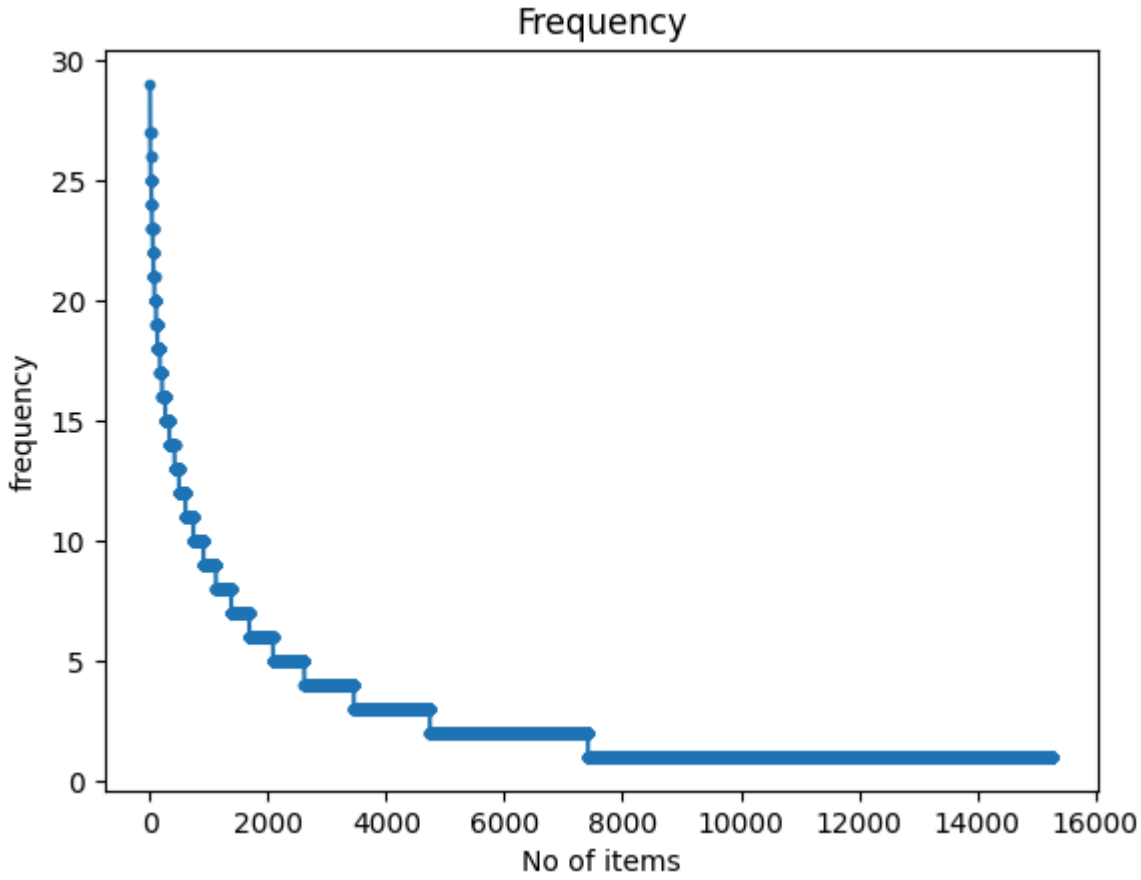


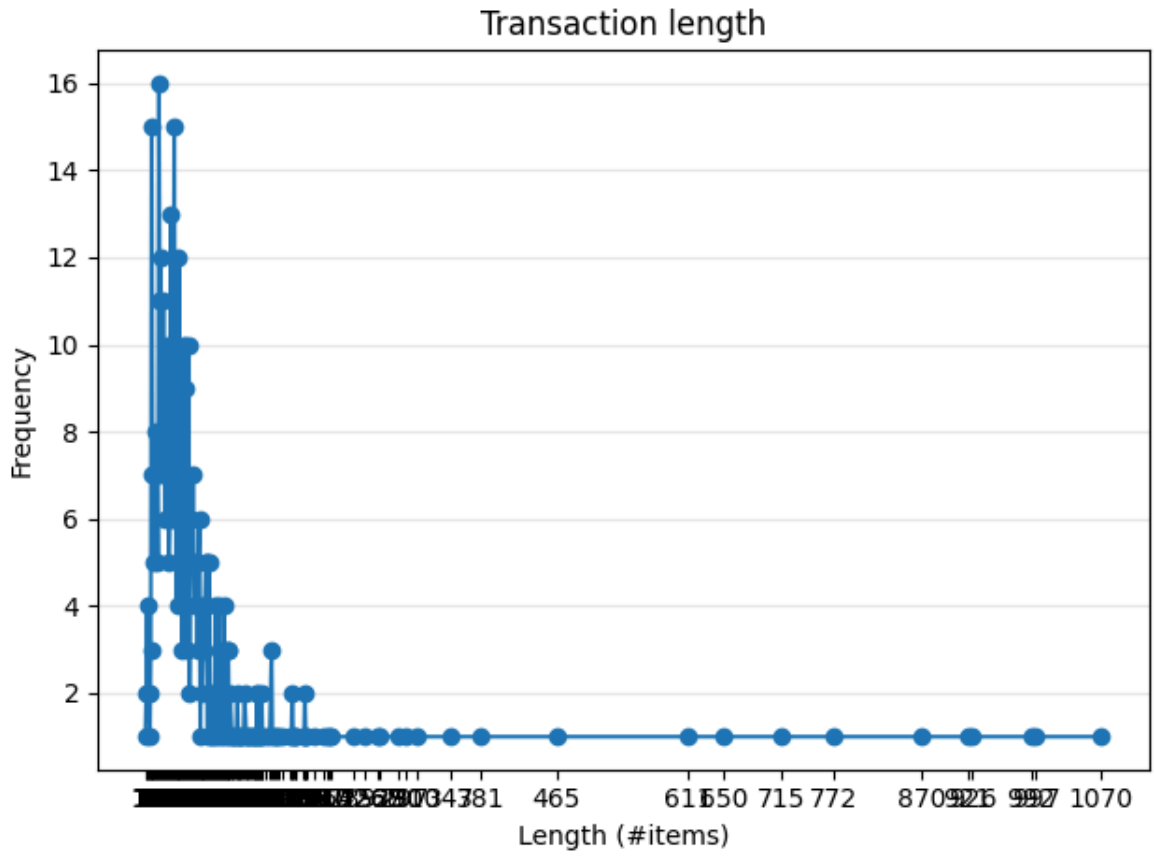


If you see that in the graph the numbers are sticked together (unlike what you see in the tutorial video) it is normal. This may be due to the update of the PAMI library. Our TA have reported the issue to the PAMI developers, if there are any update that fixes this issue we will let you know of it during the period of time of this lab.

In [ ]:
```
from PAMI.extras.dbStats import TransactionalDatabase as tds
obj = tds.TransactionalDatabase('td_freq_db_sci_med.csv')
obj.run()
obj.printStats()
obj.plotGraphs()
```

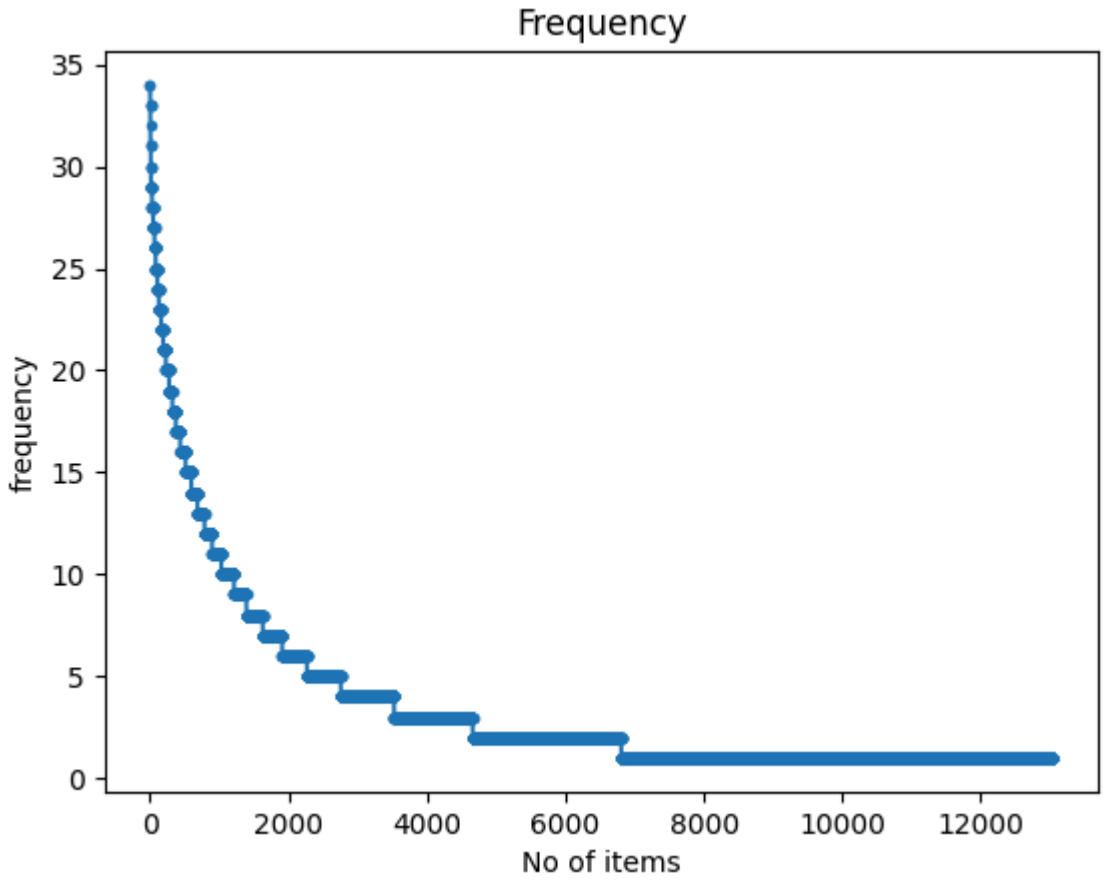
```
Database size (total no of transactions) : 594
Number of items : 15283
Minimum Transaction Size : 9
Average Transaction Size : 74.55892255892256
Maximum Transaction Size : 1070
Standard Deviation Transaction Size : 115.43440452043433
Variance in Transaction Sizes : 13347.572407606134
Sparsity : 0.9951214471923757
```

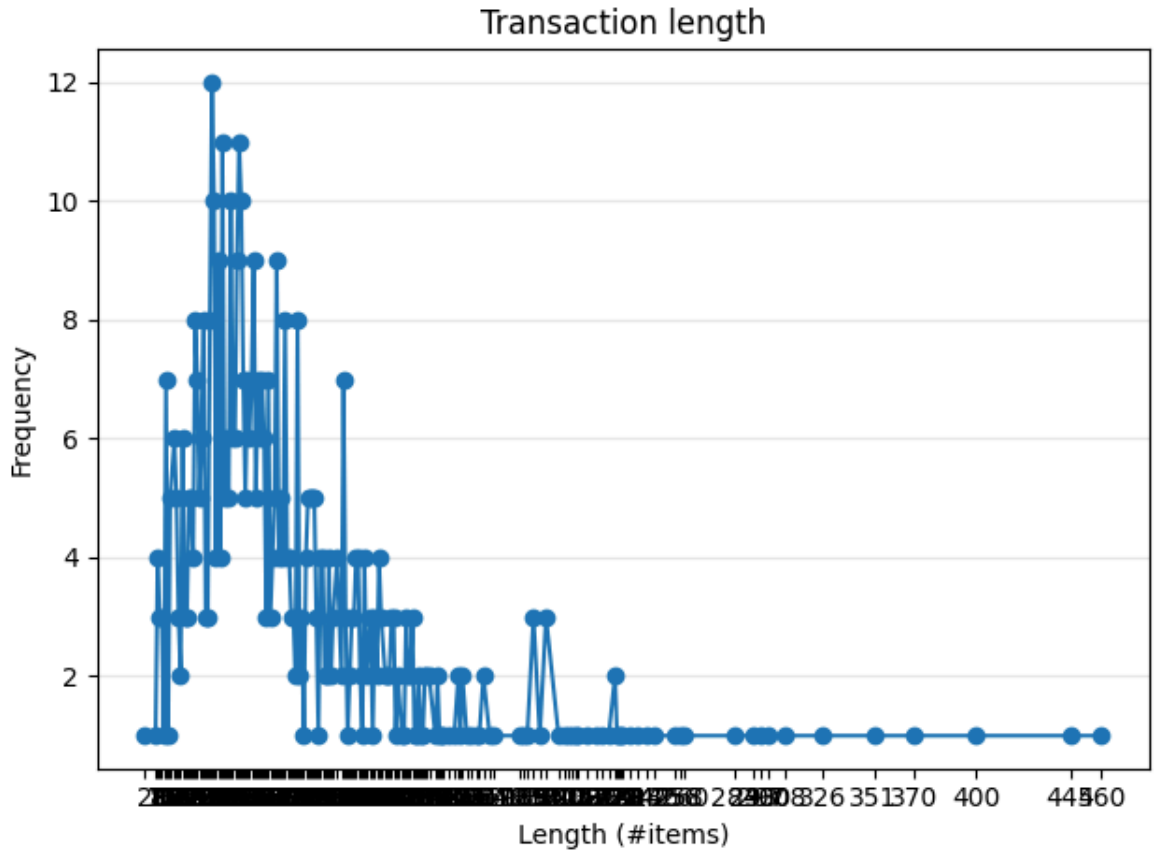




In [ ]:
```
from PAMI.extras.dbStats import TransactionalDatabase as tds
obj = tds.TransactionalDatabase('td_freq_db_soc_religion_christian.csv')
obj.run()
obj.printStats()
obj.plotGraphs()
```

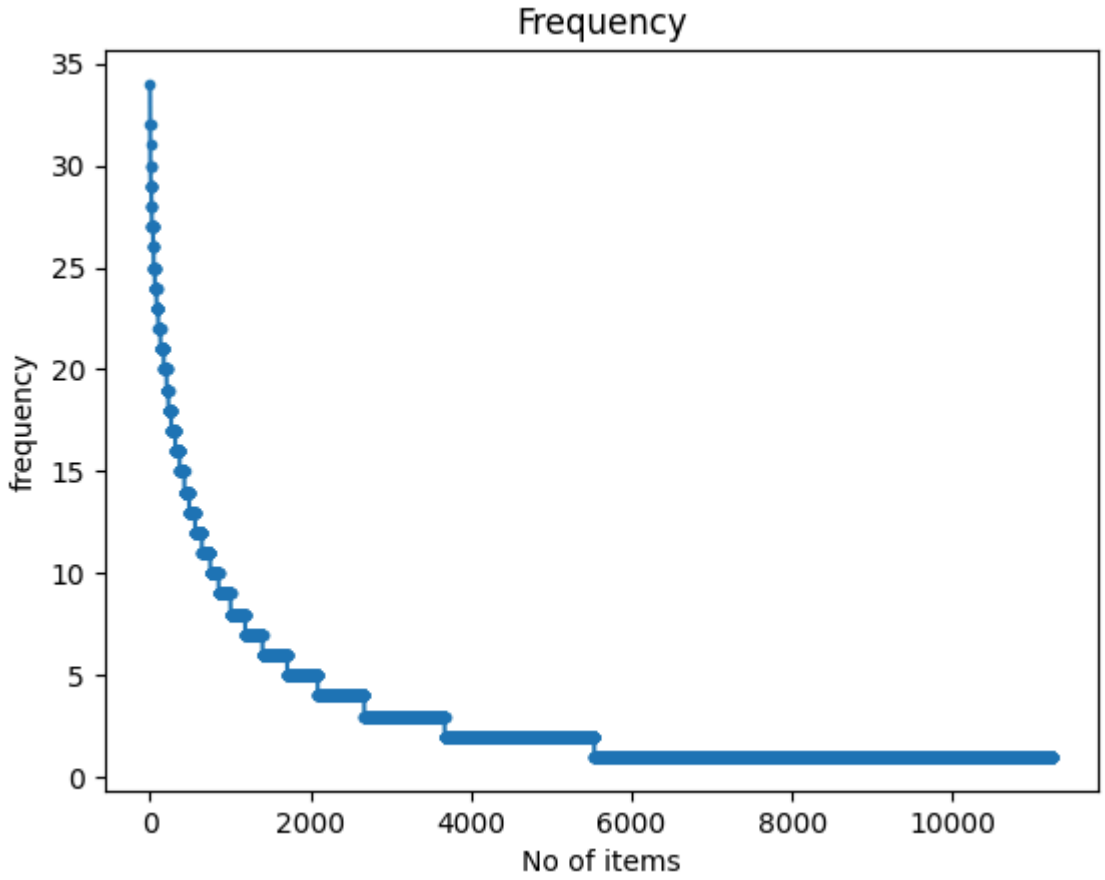
```
Database size (total no of transactions) : 599
Number of items : 13041
Minimum Transaction Size : 2
Average Transaction Size : 77.04507512520868
Maximum Transaction Size : 460
Standard Deviation Transaction Size : 62.384260554199
Variance in Transaction Sizes : 3898.3039849023735
Sparsity : 0.9940920884038641
```

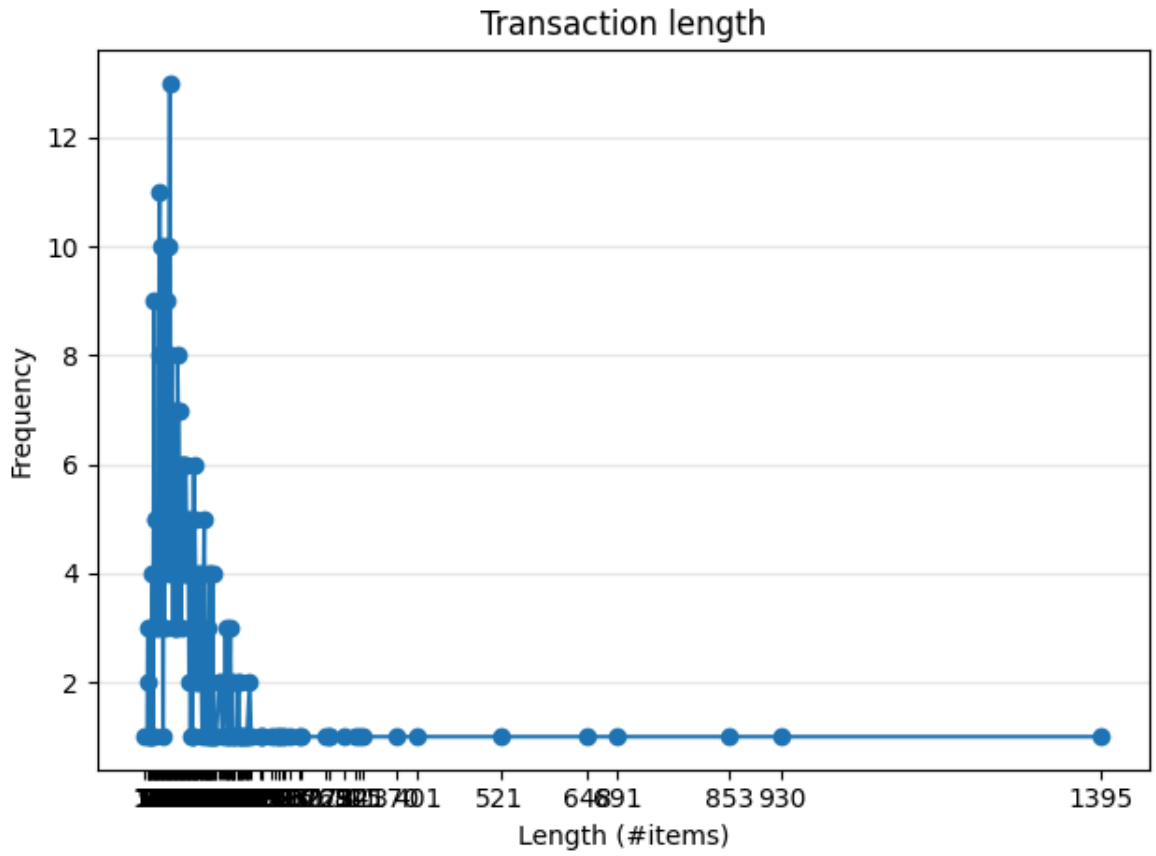




In [ ]:
```
from PAMI.extras.dbStats import TransactionalDatabase as tds
obj = tds.TransactionalDatabase('td_freq_db_alt_atheism.csv')
obj.run()
obj.printStats()
obj.plotGraphs()
```

```
Database size (total no of transactions) : 480
Number of items : 11250
Minimum Transaction Size : 5
Average Transaction Size : 75.59791666666666
Maximum Transaction Size : 1395
Standard Deviation Transaction Size : 105.33869380396925
Variance in Transaction Sizes : 11119.405841162143
Sparsity : 0.9932801851851852
```





Now that we have reviewed the stats of our databases, there are some things to notice from them, the total number of transactions refer to the amount of documents per category, the number of items refer to the amount of unique words encountered in each category, the transaction size refers to the amount of words per document that it can be found, and we can see that our databases are very sparse, this is the

result of having many zeros in the first place when making the document-term matrix.

Why are these stats important? It is because we are going to use the FPGrowth algorithm from PAMI, and for that we need to determine the *minimum support* (frequency) that our algorithm will use to mine for patterns in our transactions.

When we set a minimum support threshold (minSup) for finding frequent patterns, we are looking for a good balance. We want to capture important patterns that show real connections in the data, but we also want to avoid too many unimportant patterns. For this dataset, we've chosen a minSup of 9. We have done this after observing the following:

- **Item Frequency**: The first graph shows that most items don't appear very often in transactions. There's a sharp drop in how frequently items appear, which means our data has many items that aren't used much.
- **Transaction Length**: The second graph shows that most transactions involve a small number of items. The most common transaction sizes are small, which matches our finding that the dataset does not group many items together often.

By setting minSup at 9, we focus on combinations of items that show up in these smaller, more common transactions. This level is low enough to include items that show up more than just a few times, but it's high enough to leave out patterns that don't appear often enough to be meaningful. This helps us keep our results clear and makes sure the patterns we find are useful and represent what's really happening in the dataset.

This value works for all categories. Now let's get into mining those patterns. For more information you can visit the FPGrowth example in PAMI for transactional data: FPGrowth Example.

In [ ]:
```
from PAMI.frequentPattern.basic import FPGrowth as alg
minSup=9
obj1 = alg.FPGrowth(iFile='td_freq_db_sci_med.csv', minSup=minSup)
obj1.mine()
frequentPatternsDF_sci_med= obj1.getPatternsAsDataFrame()
print('Total No of patterns: ' + str(len(frequentPatternsDF_sci_med))) #p
print('Runtime: ' + str(obj1.getRuntime())) #measure the runtime
```

```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
Total No of patterns: 10000
Runtime: 0.05280494689941406
```

In [ ]:
```
obj1.save('freq_patterns_sci_med_minSup9.txt') #save the patterns
frequentPatternsDF_sci_med
```

Out[ ]:

	Patterns	Support
0	leave	9
1	swell	9
2	color	9
3	confused	9
4	learning	9
...	...	...
9995	means	27
9996	haven	27
9997	happen	27
9998	originator	27
9999	came	29

10000 rows × 2 columns

In [ ]:
```
from PAMI.frequentPattern.basic import FPGrowth as alg
minSup=9
obj2 = alg.FPGrowth(iFile='td_freq_db_alt_atheism.csv', minSup=minSup)
obj2.mine()
frequentPatternsDF_alt_atheism= obj2.getPatternsAsDataFrame()
print('Total No of patterns: ' + str(len(frequentPatternsDF_alt_atheism)))
print('Runtime: ' + str(obj2.getRuntime())) #measure the runtime
```

```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
Total No of patterns: 4676
Runtime: 0.05146598815917969
```

In [ ]:
```
obj2.save('freq_patterns_alt_atheism_minSup9.txt') #save the patterns
frequentPatternsDF_alt_atheism
```

Out[ ]:

	Patterns	Support
0	choices	9
1	corp	9
2	hausmann	9
3	hausmann\tmaddi	9
4	kids	9
...	...	...
4671	date	31
4672	date\tgmt	19
4673	stay	32
4674	snm6394	32
4675	gmt	34

4676 rows × 2 columns

In [ ]:
```
from PAMI.frequentPattern.basic import FPGrowth as alg
minSup=9
obj3 = alg.FPGrowth(iFile='td_freq_db_comp_graphics.csv', minSup=minSup)
obj3.mine()
frequentPatternsDF_comp_graphics= obj3.getPatternsAsDataFrame()
print('Total No of patterns: ' + str(len(frequentPatternsDF_comp_graphics
print('Runtime: ' + str(obj3.getRuntime())) #measure the runtime
```

```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
Total No of patterns: 33574
Runtime: 0.03607010841369629
```

In [ ]:
```
obj3.save('freq_patterns_comp_graphics_minSup9.txt') #save the patterns
frequentPatternsDF_comp_graphics
```

Out[ ]:

	Patterns	Support
0	individual	9
1	certain	9
2	hall	9
3	mike	9
4	vr	9
...	...	...
33569	life	24
33570	ideas	24
33571	feel	24
33572	tried	24
33573	maybe	26

33574 rows × 2 columns

In [ ]:
```
from PAMI.frequentPattern.basic import FPGrowth as alg
minSup=9
obj4 = alg.FPGrowth(iFile='td_freq_db_soc_religion_christian.csv', minSup
obj4.mine()
frequentPatternsDF_soc_religion_christian= obj4.getPatternsAsDataFrame()
print('Total No of patterns: ' + str(len(frequentPatternsDF_soc_religion_
print('Runtime: ' + str(obj4.getRuntime())) #measure the runtime
```

```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
Total No of patterns: 11213
Runtime: 0.22988390922546387
```

In [ ]:
```
obj4.save('freq_patterns_soc_religion_minSup9.txt') #save the patterns
frequentPatternsDF_soc_religion_christian
```

Out[ ]:

	Patterns	Support
0	latest	9
1	san	9
2	seven	9
3	schools	9
4	chose	9
...	...	...
11208	34	33
11209	institute	33
11210	ways	33
11211	oh	34
11212	send	34

11213 rows × 2 columns

Now that we've extracted the transactional patterns from our databases, the next step is to integrate them effectively with our initial data for further analysis. One effective method is to identify and use only the unique patterns that are specific to each category. This involves filtering out any patterns that are common across multiple categories.

The reason for focusing on **unique patterns** is that they can **significantly improve the classification process**. When a document contains these distinctive patterns, it provides clear, category-specific signals that help our model more accurately determine the document's category. This approach ensures that the patterns we use enhance the model's ability to distinguish between different types of content.

In [ ]:
```
import pandas as pd

#We group together all of the dataframes related to our found patterns
dfs = [frequentPatternsDF_sci_med, frequentPatternsDF_soc_religion_christ


# Identify patterns that appear in more than one category
# Count how many times each pattern appears across all dataframes
pattern_counts = {}
for df in dfs:
    for pattern in df['Patterns']:
        if pattern not in pattern_counts:
            pattern_counts[pattern] = 1
        else:
            pattern_counts[pattern] += 1

# Filter out patterns that appear in more than one dataframe
unique_patterns = {pattern for pattern, count in pattern_counts.items() i
# Calculate the total number of patterns across all categories
total_patterns_count = sum(len(df) for df in dfs)
# Calculate how many patterns were discarded
```

```
discarded_patterns_count = total_patterns_count - len(unique_patterns)

# For each category, filter the patterns to keep only the unique ones
filtered_dfs = []
for df in dfs:
    filtered_df = df[df['Patterns'].isin(unique_patterns)]
    filtered_dfs.append(filtered_df)

# Merge the filtered dataframes into a final dataframe
final_pattern_df = pd.concat(filtered_dfs, ignore_index=True)

# Sort by support
final_pattern_df = final_pattern_df.sort_values(by='Support', ascending=F

# Display the final result
print(final_pattern_df)
# Print the number of discarded patterns
print(f"Number of patterns discarded: {discarded_patterns_count}")
```

```
                          Patterns  Support
19911                          gov       33
19912                    institute       33
57158                      snm6394       32
19910                         form       31
57156                      deleted       30
...                            ...      ...
54045  axes\tfunds\tmotss\tmatching        9
54044   boy\tfunds\tmotss\tmatching        9
54043    boy\taxes\tmotss\tmatching        9
54042    et\tfunds\tmotss\tmatching        9
0                            swell        9

[57159 rows x 2 columns]
Number of patterns discarded: 2304
```

We observed a significant number of patterns that were common across different categories, which is why we chose to discard them. The next step is to integrate these now category-specific patterns into our data. How will we do this? By converting the patterns into binary data within the columns of our document-term matrix. Specifically, we will check each document for the presence of each pattern. If a pattern is found in the document, we'll mark it with a '1'; if it's not present, we'll mark it with a '0'. This binary encoding allows us to effectively augment our data, enhancing its utility for subsequent classification tasks.

In [ ]:
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

# Convert 'text' column into term-document matrix using CountVectorizer
count_vect = CountVectorizer()
X_tdm = count_vect.fit_transform(X['text'])  # X['text'] contains your te
terms = count_vect.get_feature_names_out()  # Original terms in the vocab

# Tokenize the sentences into sets of unique words
X['tokenized_text'] = X['text'].str.split().apply(set)

# Initialize the pattern matrix
pattern_matrix = pd.DataFrame(0, index=X.index, columns=final_pattern_df[
```

```
# Iterate over each pattern and check if all words in the pattern are pre
for pattern in final_pattern_df['Patterns']:
    pattern_words = set(pattern.split())  # Tokenize pattern into words
    pattern_matrix[pattern] = X['tokenized_text'].apply(lambda x: 1 if pa

# Convert the term-document matrix to a DataFrame for easy merging
tdm_df = pd.DataFrame(X_tdm.toarray(), columns=terms, index=X.index)

# Concatenate the original TDM and the pattern matrix to augment the feat
augmented_df = pd.concat([tdm_df, pattern_matrix], axis=1)

augmented_df
```

Out[ ]:

	00	000	0000	0000001200	000005102000	0001	000100255pixel	000
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	1	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	
2252	0	0	0	0	0	0	0	
2253	0	0	0	0	0	0	0	
2254	0	0	0	0	0	0	0	
2255	0	0	0	0	0	0	0	
2256	0	0	0	0	0	0	0	

2257 rows × 92947 columns

>>> Exercise 17 (take home):

Implement the FAE Top-K and MaxFPGrowth algorithms from the PAMI library to analyze the 'comp.graphics' category in our processed database. **Only implement the mining part of the algorithm and display the resulting patterns**, like we did with the FPGrowth algorithm after creating the new databases. For the FAE Top-K, run trials with k values of 500, 1000, and 1500, recording the runtime for each. For MaxFPGrowth, test minimum support thresholds of 3, 6, and 9, noting the runtime for these settings as well. Compare the patterns these algorithms extract with those from the previously implemented FPGrowth algorithm. Document your findings, focusing on differences and similarities in the outputs and performance. For this you can find the following google collabs for reference provided by their github repository here: FAE Top-K and MaxFPGrowth

In [ ]:

```
# In this exercise, we compared three frequent-pattern mining algorithms
# FPGrowth, FAE Top-K, and MaxFPGrowth – using the comp.graphics dataset.
# Each algorithm produced different results depending on its parameters a
```

```
#
# FPGrowth found a large number of frequent patterns based on a fixed min
# It was very fast, but many of the patterns were overlapping or redundan
#
# FAE Top-K returned exactly the top K most frequent patterns without nee
# This made it easier to control how many results we got.
# The runtime increased slightly when K was larger but still stayed effic
#
# MaxFPGrowth only produced maximal frequent itemsets, meaning none of th
# It gave fewer patterns, but they were more concise and easier to interp
#
# Overall, FPGrowth gave the most complete results, FAE Top-K provided fl
# and MaxFPGrowth generated the most compact and readable patterns.
# These results show that the best algorithm depends on whether we care m
# completeness, efficiency, or simplicity.
```

In [ ]:
```
#      FPGrowth on comp.graphics
from PAMI.frequentPattern.basic import FPGrowth as fp_alg

minSup = 9
fp = fp_alg.FPGrowth(iFile='td_freq_db_comp_graphics.csv', minSup=minSup)
fp.mine()
fp_df = fp.getPatternsAsDataFrame()

print('[FPGrowth @ comp.graphics]')
print('minSup =', minSup)
print('Total No of patterns:', len(fp_df))
print('Runtime (sec):', fp.getRuntime())

#             Support
display(fp_df.sort_values('Support', ascending=False).head(10))

#
fp.save('freq_patterns_comp_graphics_minSup9.txt')
```

```
Frequent patterns were generated successfully using frequentPatternGrowth
algorithm
[FPGrowth @ comp.graphics]
minSup = 9
Total No of patterns: 33574
Runtime (sec): 0.03568887710571289
```

	Patterns	Support
33573	maybe	26
33572	tried	24
33571	feel	24
33570	ideas	24
33569	life	24
33568	needs	23
33567	wrong	23
33562	suggestions	22
33558	correct	22
33559	didn	22

In [ ]:
```python
# ---          MaxFPGrowth ---
try:
    #      PAMI
    from PAMI.frequentPattern.maximal.MaxFPGrowth import MaxFPGrowth as m
except ImportError:
    #     ,
    from PAMI.frequentPattern.maximal import MaxFPGrowth as max_alg

# ---               minSup ---
minSups = [3, 6, 9]
max_results = []

for ms in minSups:
    print(f"\n[Running MaxFPGrowth @ comp.graphics] minSup = {ms}")
    mfp = max_alg(iFile='td_freq_db_comp_graphics.csv', minSup=ms)
    mfp.mine()
    mfp_df = mfp.getPatternsAsDataFrame()

    max_results.append({
        'minSup': ms,
        'num_patterns': len(mfp_df),
        'runtime_sec': mfp.getRuntime(),
        'df': mfp_df
    })

    print(f"Total No of maximal patterns: {len(mfp_df)}")
    print(f"Runtime (sec): {mfp.getRuntime():.4f}")
    display(mfp_df.sort_values('Support', ascending=False).head(10))
```

```
[Running MaxFPGrowth @ comp.graphics] minSup = 3
Maximal Frequent patterns were generated successfully using MaxFp-Growth a
lgorithm
Total No of maximal patterns: 6872
Runtime (sec): 6.3186
```

	Patterns	Support
3115	searching	7
2892	daniel	6
2608	bought	6
2324	late	6
2334	roll	6
2933	wang	6
2926	greetings	6
2562	inst	6
2891	onto	6
2313	recall	6

```
[Running MaxFPGrowth @ comp.graphics] minSup = 6
Maximal Frequent patterns were generated successfully using MaxFp-Growth a
lgorithm
Total No of maximal patterns: 1340
Runtime (sec): 0.0862
```

	Patterns	Support
1339	ideas	24
1338	tried	24
1337	wrong	23
1336	didn	22
1333	wondering	21
1332	knows	21
1328	26	20
1325	comes	20
1313	due	19
1317	appreciate	19

```
[Running MaxFPGrowth @ comp.graphics] minSup = 9
Maximal Frequent patterns were generated successfully using MaxFp-Growth a
lgorithm
Total No of maximal patterns: 660
Runtime (sec): 0.0362
```

	Patterns	Support
659	maybe	26
658	life	24
657	ideas	24
656	tried	24
655	wrong	23
652	recently	22
649	exactly	22
651	graphic	22
650	suggestions	22
653	didn	22

In [ ]:
```python
# ---         FAE Top-K      fae_results ---
try:
    from PAMI.frequentPattern.topk.FAE import FAE as fae_alg  #
except ImportError:
    #
    from PAMI.frequentPattern.topk import FAE as fae_alg

fae_results = []
for k in [500, 1000, 1500]:
    fae = fae_alg(iFile='td_freq_db_comp_graphics.csv', k=k)
    fae.mine()
    fae_df = fae.getPatternsAsDataFrame()
    fae_results.append({
        "k": k,
        "num_patterns": len(fae_df),
        "runtime_sec": fae.getRuntime(),
        "df": fae_df
    })
    print(f"[FAE Top-K] k={k} | #patterns={len(fae_df)} | runtime={fae.ge
```

```
 TopK frequent patterns were successfully generated using FAE algorithm.
[FAE Top-K] k=500 | #patterns=500 | runtime=0.3350s
 TopK frequent patterns were successfully generated using FAE algorithm.
[FAE Top-K] k=1000 | #patterns=1000 | runtime=0.7272s
 TopK frequent patterns were successfully generated using FAE algorithm.
[FAE Top-K] k=1500 | #patterns=1500 | runtime=1.7508s
```

5.5 Dimensionality Reduction

Dimensionality reduction is a powerful technique for tackling the "curse of dimensionality," which commonly arises due to data sparsity. This technique is not only beneficial for visualizing data more effectively but also simplifies the data by reducing the number of dimensions without losing significant information. For a deeper understanding, please refer to the additional notes provided.

We will start with **Principal Component Analysis (PCA)**, which is focused on finding a projection that captures the largest amount of variation in the data. PCA is excellent

for linear dimensionality reduction and works well when dealing with Gaussian distributed data. However, its effectiveness diminishes with non-linear data structures.

Additionally, we will explore two advanced techniques suited for non-linear dimensionality reductions:

- **t-Distributed Stochastic Neighbor Embedding (t-SNE)**:
  - Pros:
    - Effective at revealing local data structures at many scales.
    - Great for identifying clusters in data.
  - Cons:
    - Computationally intensive, especially with large datasets.
    - Sensitive to parameter settings and might require tuning (e.g., perplexity).
- **Uniform Manifold Approximation and Projection (UMAP)**:
  - Pros:
    - Often faster than t-SNE and can handle larger datasets.
    - Less sensitive to the choice of parameters compared to t-SNE.
    - Preserves more of the global data structure while also revealing local structure.
  - Cons:
    - Results can still vary based on parameter settings and random seed.
    - May require some experimentation to find the optimal settings for specific datasets.

These methods will be applied to visualize our data more effectively, each offering unique strengths to mitigate the issue of sparsity and allowing us to observe underlying patterns in our dataset.

PCA Algorithm t-SNE Algorithm UMAP Algorithm

Input: Raw term-vector matrix

Output: Projections

So, let's experiment with something interesting, from our previous work we have our data with only the document-term frequency data and also the one with both the document-term frequency and the pattern derived data, let's try to create a 2D plot after applying these algorithms to our dataframes and see what comes out.

In [ ]:
```
#Applying dimensionality reduction with only the document-term frequency
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import umap
import matplotlib.pyplot as plt

#This might take a couple of minutes to execute
```

```
# Apply PCA, t-SNE, and UMAP to the data
X_pca_tdm = PCA(n_components=2).fit_transform(tdm_df.values)
X_tsne_tdm = TSNE(n_components=2).fit_transform(tdm_df.values)
X_umap_tdm = umap.UMAP(n_components=2).fit_transform(tdm_df.values)
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

invalid value encountered in matmul
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

invalid value encountered in matmul
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

invalid value encountered in matmul
```

In [ ]:
```
X_pca_tdm.shape
```

Out[ ]:
```
(2257, 2)
```

In [ ]:
```
X_tsne_tdm.shape
```

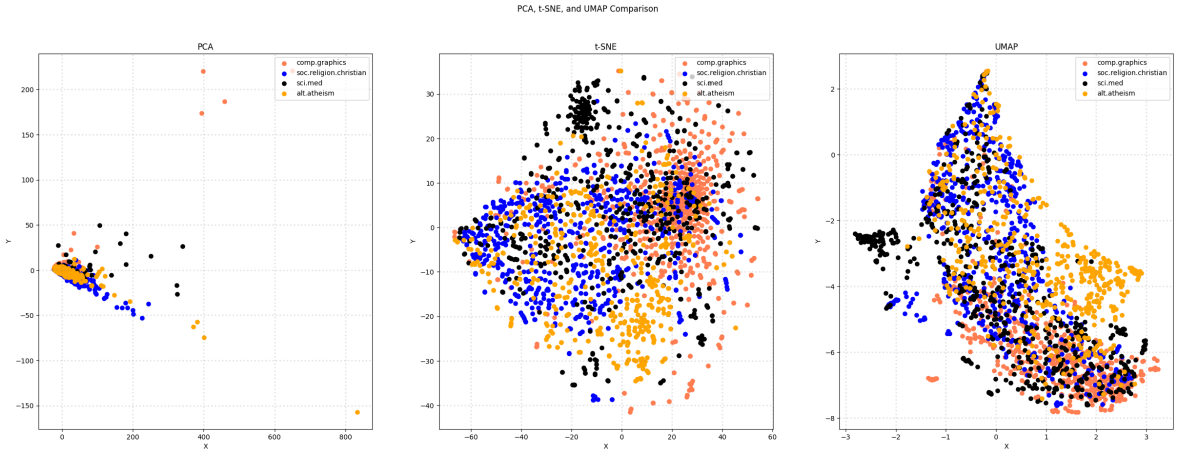
Out[ ]:
```
(2257, 2)
```

In [ ]:
```
X_umap_tdm.shape
```

Out[ ]:
```
(2257, 2)
```

In [ ]:
```
# Plot the results in subplots
col = ['coral', 'blue', 'black', 'orange']
categories = X['category_name'].unique()

fig, axes = plt.subplots(1, 3, figsize=(30, 10))  # Create 3 subplots for
fig.suptitle('PCA, t-SNE, and UMAP Comparison')

# Define a function to create a scatter plot for each method
def plot_scatter(ax, X_reduced, title):
    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', label=category)

    ax.grid(color='gray', linestyle=':', linewidth=2, alpha=0.2)
    ax.set_title(title)
```

```
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.legend(loc='upper right')

# Step 4: Create scatter plots for PCA, t-SNE, and UMAP
plot_scatter(axes[0], X_pca_tdm, 'PCA')
plot_scatter(axes[1], X_tsne_tdm, 't-SNE')
plot_scatter(axes[2], X_umap_tdm, 'UMAP')

plt.show()
```



The plots generated by the above code may looks slightly different from those in the tutorial video. It is normal behavior due to some update of the PAMI library

From the 2D PCA 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. In the t-SNE graph we observe a more scattered distribution, but still intermixing with all the categories. And with the UMAP graph, the limits for the data seem pretty well defined, two categories seem to have some points well differentiated from the other classes, but most of them remain intermixed. The algorithms were 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.

Now let's try in tandem with our pattern augmented data:

In [ ]:
```
#This might take a couple of minutes to execute
#Applying dimensionality reduction with both the document-term frequency
# Apply PCA, t-SNE, and UMAP to the data
X_pca_aug = PCA(n_components=2).fit_transform(augmented_df.values)
X_tsne_aug = TSNE(n_components=2).fit_transform(augmented_df.values)
X_umap_aug = umap.UMAP(n_components=2).fit_transform(augmented_df.values)
```

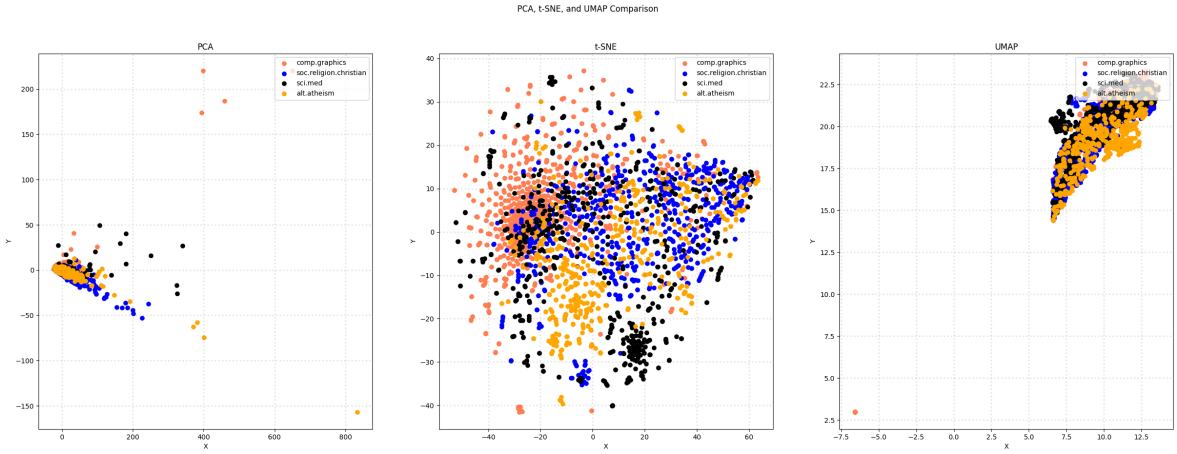
```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

invalid value encountered in matmul
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

invalid value encountered in matmul
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

invalid value encountered in matmul
```

In [ ]:

```python
# Plot the results in subplots
col = ['coral', 'blue', 'black', 'orange']
categories = X['category_name'].unique()

fig, axes = plt.subplots(1, 3, figsize=(30, 10))  # Create 3 subplots for
fig.suptitle('PCA, t-SNE, and UMAP Comparison')

# Define a function to create a scatter plot for each method
def plot_scatter(ax, X_reduced, title):
    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', label=category)

    ax.grid(color='gray', linestyle=':', linewidth=2, alpha=0.2)
    ax.set_title(title)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.legend(loc='upper right')

# Create scatter plots for PCA, t-SNE, and UMAP
plot_scatter(axes[0], X_pca_aug, 'PCA')
plot_scatter(axes[1], X_tsne_aug, 't-SNE')
plot_scatter(axes[2], X_umap_aug, 'UMAP')

plt.show()
```



The plots generated by the above code may looks slightly different from those in the tutorial video. It is normal behavior due to some update of the PAMI library

We can see that our PCA visualization hasn't changed much from the previous version. This is likely because the original document-term matrix still dominates what the algorithm captures, overshadowing the new binary pattern data we added.

Looking at the t-SNE graph, it might seem different at first glance. However, upon closer inspection, it's almost the same but mirrored along the y-axis, with only slight changes in how the data points are placed. This similarity might be due to the stability of the t-SNE algorithm. Even small changes in the data can result in embeddings that look different but are structurally similar, indicating that the binary patterns may not have significantly altered the relationships among the data points in high-dimensional space.

The UMAP visualization shows the most noticeable changes—it appears more compact. This compactness could be because UMAP uses a more complex distance metric, which might be making it easier to see differences between closer and further points. The binary patterns could also be helping to reduce noise within categories, resulting in clearer, more coherent groups. However, the categories still appear quite mixed together.

Remember, just because you can't see clear groups in these visualizations doesn't mean that a machine learning model won't be able to classify the data correctly. These techniques are mainly used to help us see and understand complex data in a simpler two or three-dimensional space. However, they have their limits and might not show everything a computer model can find in the data. So, while these tools are great for getting a first look at your data, always use more methods and analyses to get the full picture.

>>> Exercise 18 (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.

In [ ]:
```
#Answer Here
# PCA-3D
# The data mainly form a flat, elongated cluster with only weak class sep
# This shows PCA captures global linear variance but not nonlinear class
#
# t-SNE-3D
# The clusters appear in a roughly spherical shape, and categories still
# t-SNE preserves local density but does not clearly separate groups.
#
# UMAP-3D
# UMAP shows the clearest and most compact structure, revealing several p
# while preserving the overall geometry.
# It balances local and global structure better than PCA or t-SNE.
#
# Summary:
# PCA → global, linear, flat
# t-SNE → local, overlapping clusters
# UMAP → compact, interpretable embedding

# Additional Observation:
# For the augmented dataset, UMAP-3D became noticeably more compact than
# This likely happens because the added pattern-based features increased
# UMAP, which preserves both local and global topology, compresses the ma
# become more homogeneous.
# In contrast, t-SNE focuses on local neighborhoods and therefore remains
# Thus, the compact UMAP result reflects stronger global coherence rather
```

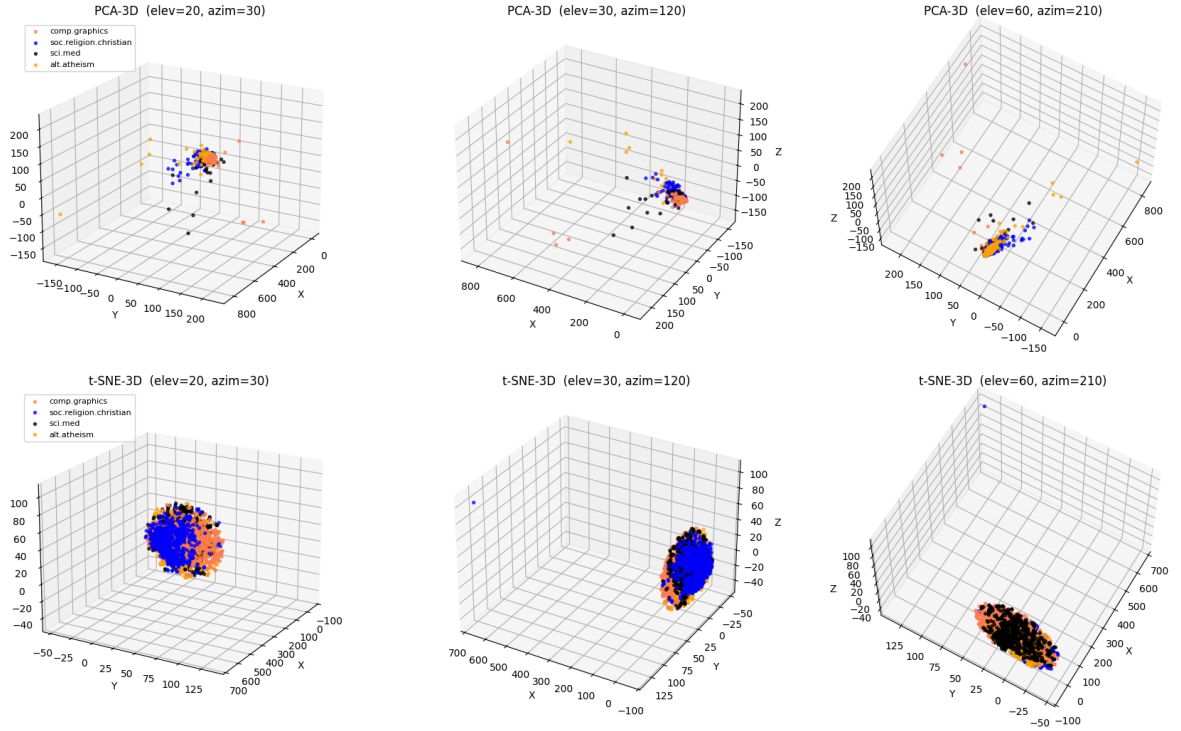
In [ ]:
```
# ====== 3D embeddings ======
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import umap
import numpy as np
import matplotlib.pyplot as plt

y = X['category_name'].values
cats = X['category_name'].unique()
colors = ['coral', 'blue', 'black', 'orange']  # 4
color_map = {c: colors[i % len(colors)] for i, c in enumerate(cats)}

#
RANDOM_STATE = 42

# 3D embeddings
X_pca_3d  = PCA(n_components=3, random_state=RANDOM_STATE).fit_transform(
X_tsne_3d = TSNE(n_components=3, perplexity=30, init='pca',
                 learning_rate='auto', random_state=RANDOM_STATE).fit_tra
X_umap_3d = umap.UMAP(n_components=3, n_neighbors=15, min_dist=0.1,
                      metric='cosine', random_state=RANDOM_STATE).fit_tra

#
def plot_3views(emb, title, views=[(20,30),(30,120),(60,210)]):
    fig = plt.figure(figsize=(18,5))
    for i, (elev, azim) in enumerate(views, 1):
        ax = fig.add_subplot(1, 3, i, projection='3d')
        for c in cats:
            idx = (y == c)
            ax.scatter(emb[idx,0], emb[idx,1], emb[idx,2],
                       s=8, alpha=0.7, label=c, c=color_map[c])
        ax.view_init(elev=elev, azim=azim)
```

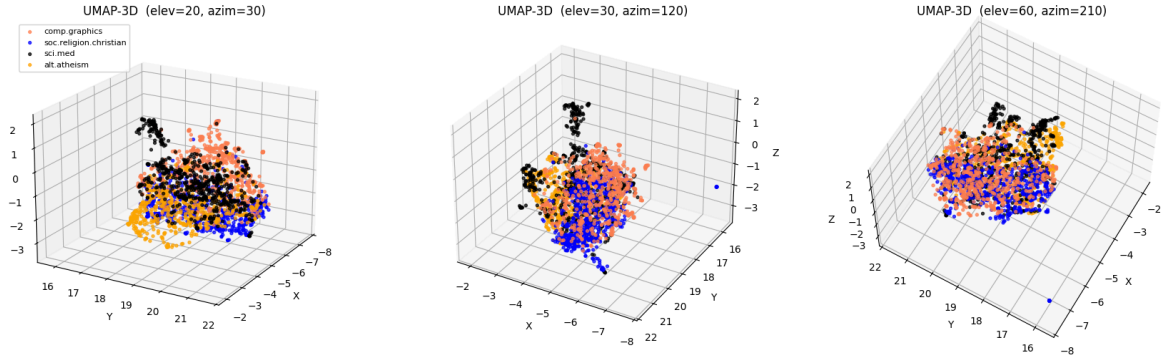
```
        ax.set_title(f'{title}  (elev={elev}, azim={azim})')
        ax.set_xlabel('X'); ax.set_ylabel('Y'); ax.set_zlabel('Z')
        if i == 1: ax.legend(loc='upper left', fontsize=8)
        ax.grid(True, alpha=0.2)
    plt.tight_layout()
    plt.show()

#     PCA / t-SNE / UMAP
plot_3views(X_pca_3d,  'PCA-3D')
plot_3views(X_tsne_3d, 't-SNE-3D')
plot_3views(X_umap_3d, 'UMAP-3D')
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

invalid value encountered in matmul
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

invalid value encountered in matmul
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/umap/umap_.py:1952: UserWarning:

n_jobs value 1 overridden to 1 by setting random_state. Use no seed for pa
rallelism.
```



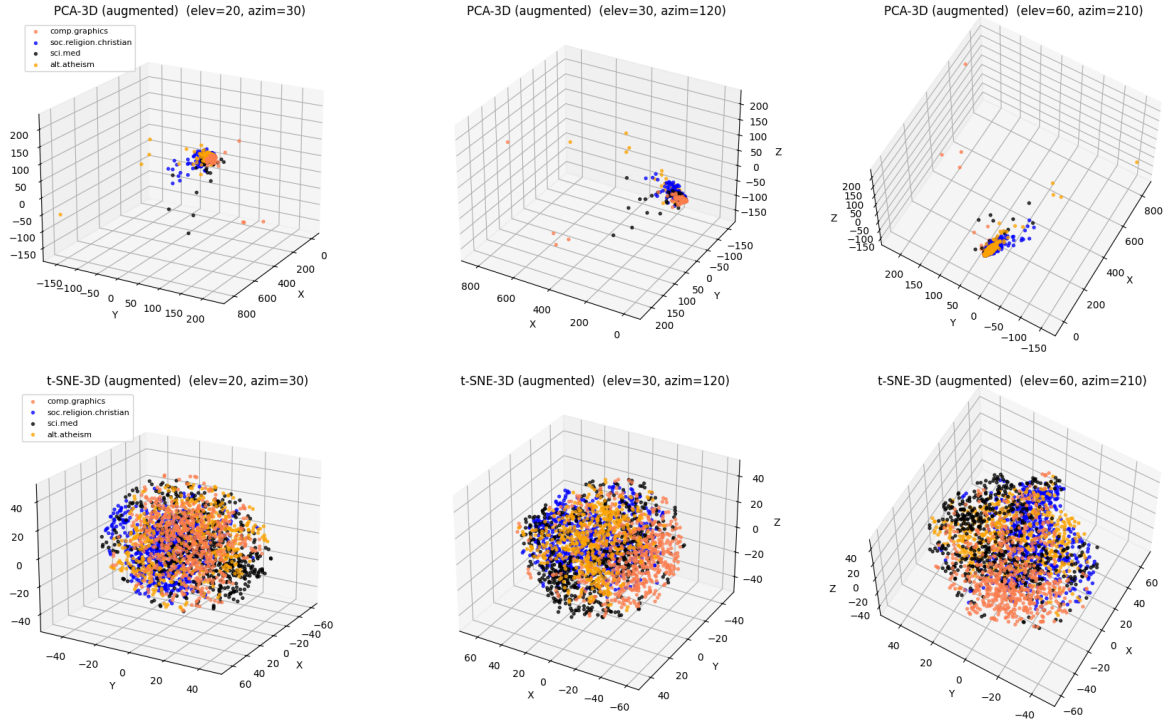


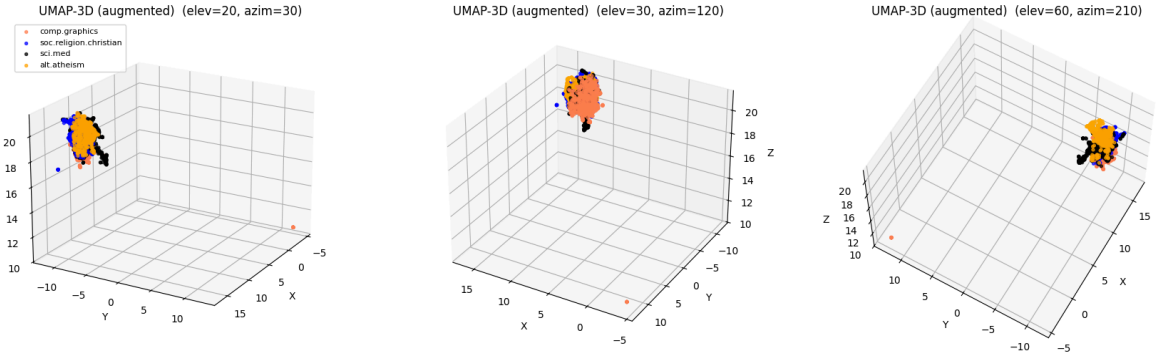
In [ ]:
```
X_pca_3d_aug  = PCA(n_components=3, random_state=RANDOM_STATE).fit_transf
X_tsne_3d_aug = TSNE(n_components=3, perplexity=30, init='pca',
                     learning_rate='auto', random_state=RANDOM_STATE).fit
X_umap_3d_aug = umap.UMAP(n_components=3, n_neighbors=15, min_dist=0.1,
                          metric='cosine', random_state=RANDOM_STATE).fit

plot_3views(X_pca_3d_aug,  'PCA-3D (augmented)')
plot_3views(X_tsne_3d_aug, 't-SNE-3D (augmented)')
plot_3views(X_umap_3d_aug, 'UMAP-3D (augmented)')
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

invalid value encountered in matmul
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:335: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:336: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:340: RuntimeWarning:

invalid value encountered in matmul
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:533: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:547: RuntimeWarning:

invalid value encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/umap/umap_.py:1952: UserWarning:

n_jobs value 1 overridden to 1 by setting random_state. Use no seed for pa
rallelism.
```





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.

In [ ]:
```
from sklearn import preprocessing, metrics, decomposition, pipeline, dumm
```

In [ ]:
```
mlb = preprocessing.LabelBinarizer()
```

In [ ]:
```
mlb.fit(X.category)
```

Out[ ]:
```
▾ LabelBinarizer
LabelBinarizer()
```

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

In [ ]:
```
X[0:9]
```

Out[ ]:

	text	category	category_name	
0	From: sd345@city.ac.uk (Michael Collier) Subje...	1	comp.graphics	[From, :, s city.ac.uk, (, Mic
1	From: ani@ms.uky.edu (Aniruddha B. Deglurkar) ...	1	comp.graphics	[From, :, ani, @, ms (, Aniru
2	From: djohnson@cs.ucsd.edu (Darin Johnson) Sub...	3	soc.religion.christian	[From, :, djo cs.ucsd.edu, (
3	From: s0612596@let.rug.nl (M.M. Zwart) Subject...	3	soc.religion.christian	[From, :, s06 let.rug.nl, (,
4	From: stanly@grok11.columbiasc.ncr.com (stanly...	3	soc.religion.christian	[From, :, grok11.columbiasc.
5	From: vbv@lor.eeap.cwru.edu (Virgilio (Dean) B...	3	soc.religion.christian	[From, lor.eeap.cwru.edu,
6	From: jodfishe@silver.ucs.indiana.edu (joseph ...	3	soc.religion.christian	[From, :, jo silver.ucs.indi
7	From: aldridge@netcom.com (Jacquelin Aldridge)...	2	sci.med	[From, :, al netcom.com, (,
8	From: geb@cs.pitt.edu (Gordon Banks) Subject: ...	2	sci.med	[From, :, geb, @, c (, Gordo

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 19 (take home):

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

In [ ]:
```
# Answer here
# The binarization using category_name works exactly the same as using nu
# Each unique text label is represented as a vector of 0s and 1s,
# where the position of 1 indicates the corresponding category.
# This transformation allows machine learning models to handle categorica
# without implying any ordinal relationship between categories.
```

In [ ]:
```
mlb2 = preprocessing.LabelBinarizer()
mlb2.fit(X['category_name'])
X['bin_category2'] = mlb2.transform(X['category_name']).tolist()

X[['category_name', 'bin_category2']].head()
```

Out[ ]:

	category_name	bin_category2
0	comp.graphics	[0, 1, 0, 0]
1	comp.graphics	[0, 1, 0, 0]
2	soc.religion.christian	[0, 0, 0, 1]
3	soc.religion.christian	[0, 0, 0, 1]
4	soc.religion.christian	[0, 0, 0, 1]

---

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.

In [ ]:
```
# We retrieve 3 sentences for a random record
document_to_transform_1 = []
random_record_1 = X.iloc[10]
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[1000]
random_record_3 = random_record_3['text']
document_to_transform_3.append(random_record_3)
```

Let's look at our emails.

In [ ]:
```
print(document_to_transform_1)
print(document_to_transform_2)
print(document_to_transform_3)
```

['From: anasaz!karl@anasazi.com (Karl Dussik) Subject: Re: Is "Christian" a dirty word? Organization: Anasazi Inc Phx Az USA Lines: 73 In article <Mar.25.03.53.08.1993.24855@athos.rutgers.edu> @usceast.cs.scarolina.edu:moss@cs.scarolina.edu (James Moss) writes: >I was brought up christian, but I am not christian any longer. >I also have a bad taste in my mouth over christianity. I (in >my own faith) accept and live my life by many if not most of the >teachings of christ, but I cannot let myself be called a christian, >beacuse to me too many things are done on the name of christianity, >that I can not be associated with.   A question for you - can you give me the name of an organization or a philosophy or a political movement, etc., which has never had anything evil done in its name? You\'re missing a central teaching of Christianity - man is inherently sinful. We are saved through faith by grace. Knowing that, believing that, does not make us without sin. Furthermore, not all who consider themselves "christians" are (even those who manage to head their own "churches"). "Not everyone who says to me, \'Lord, Lord,\' will enter the kingdom of heaven, but only he who does the will of my Father who is in heaven." - Matt. 7:21. >I also have a problem with the inconsistancies in the Bible, and >how it seems to me that too many people have edited the original >documents to fit their own world views, thereby leaving the Bible >an unbelievable source. Again, what historical documents do you trust? Do you think Hannibal crossed the Alps? How do you know? How do you know for sure? What historical documents have stood the scrutiny and the attempts to dis- credit it as well as the Bible has? >I don\'t have dislike of christians (except for a few who won\'t >quit witnessing to me, no matter how many times I tell them to stop), >but the christian faith/organized religion will never (as far as i can >see at the moment) get my support. Well, it\'s really a shame you feel this way. No one can browbeat you into believing, and those who try will probably only succeed in driving you further away. You need to ask yourself some difficult questions: 1) is there an afterlife, and if so, does man require salvation to attain it. If the answer is yes, the next question is 2) how does man attain this salvation - can he do it on his own as the eastern religions and certain modern offshoots like the "new age movement" teach or does he require God\'s help? 3) If the latter, in what form does - indeed, in what form can such help come? Needless to say, this discussion could take a lifetime, and for some people it did comprise their life\'s writings, so I am hardly in a position to offer the answers here - merely pointers to what to ask. Few, of us manage to have an unshaken faith our entire lives (certainly not me). The spritual life is a difficult journey (if you\'ve never read "A Pilgrim\'s Progress," I highly recommend this greatest allegory of the english language). >Peace and Love >In God (ess)\'s name >James Moss Now I see by your close that one possible source of trouble for you may be a conflict between your politcal beliefs and your religious upbringing. You wrote that "I (in my own faith) accept and live my life by many if not most of the teachings of christ". Well, Christ referred to God as "My Father", not "My Mother", and while the "maleness" of God is not the same as the maleness of those of us humans who posses a Y chromosome, it does not honor God to refer to Him as female purely to be trendy, non-discriminatory, or politically correct. This in no way disparages women (nor is it my intent to do so by my use of the male pronoun to refer to both men and women - english just does not have a decent neuter set of pronouns). After all, God chose a woman as his only human partner in bringing Christ into the human population. Well, I\'m not about to launch into a detailed discussion of the role of women in Christianity at 1am with only 6 hours of sleep in the last 63, and for that reason I also apologize for any shortcomings in this article. I just happened across yours and felt moved to reply. I hope I may have given you, and anyone else who finds himself in a similar frame of mind, something to contemplate. Karl Dussik ']
['From: mathew <mathew@mantis.co.uk> Subject: Re: university violating sep

aration of church/state? Organization: Mantis Consultants, Cambridge. UK. X-Newsreader: rusnews v1.01 Lines: 29 dmn@kepler.unh.edu (...until kings become philosophers or philosophers become kings) writes: >   Recently, RAs have been ordered (and none have resisted or cared about > it apparently) to post a religious flyer entitled _The Soul Scroll: Thoughts > on religion, spirituality, and matters of the soul_ on the inside of bathroom > stall doors. (at my school, the University of New Hampshire) It is some sort > of newsletter assembled by a Hall Director somewhere on campus. It poses a > question about \'spirituality\' each issue, and solicits responses to be > included in the next \'issue.\' It\'s all pretty vague. I assume it\'s put out > by a Christian, but they\'re very careful not to mention Jesus or the bible. > I\'ve heard someone defend it, saying "Well it doesn\'t support any one religion. > " So what??? This is a STATE university, and as a strong supporter of the > separation of church and state, I was enraged. > >   What can I do about this? It sounds to me like it\'s just SCREAMING OUT for parody. Give a copy to your friendly neighbourhood SubGenius preacher; with luck, he\'ll run it through the mental mincer and hand you back an outrageously offensive and gut-bustingly funny parody you can paste over the originals. I can see it now:
The Stool Scroll   Thoughts on Religion, Spirituality, and Matters of the Colon   (You can use this text to wipe)   mathew ']
['From: bobs@thnext.mit.edu (Robert Singleton) Subject: Re: Americans and Evolution Organization: Massachvsetts Institvte of Technology Lines: 138 Distribution: world NNTP-Posting-Host: thnext.mit.edu In article <16BA8C4AC.I3150101@dbstu1.rz.tu-bs.de>   I3150101@dbstu1.rz.tu-bs.de (Benedikt Rosenau) writes: > In article <1pq47tINN8lp@senator-bedfellow.MIT.EDU> > bobs@thnext.mit.edu (Robert Singleton) writes: >   > (Deletion) > > > >I will argue that your latter statement, "I believe that no gods exist" > >does rest upon faith - that is, if you are making a POSITIVE statement > >that "no gods exist" (strong atheism) rather than merely saying I don\'t > >know and therefore don\'t believe in them and don\'t NOT believe in then > > (weak atheism). Once again, to not believe in God is different than > >saying I BELIEVE that God does not exist. I still maintain the > >position, even after reading the FAQs, that strong atheism requires > >faith. > > > > No it in the way it is usually used. In my view, you are saying here > that driving a car requires faith that the car drives. >   I\'m not saying this at all - it requires no faith on my part to say the car drives because I\'ve seen it drive - I\'ve done more than at in fact - I\'ve actually driven it. (now what does require some faith is the belief that my senses give an accurate representation of what\'s out there....) But there is NO evidence - pro or con - for the existence or non-existence of God (see what I have to say below on this). > For me it is a conclusion, and I have no more faith in it than I > have in the premises and the argument used. > Sorry if I remain skeptical - I don\'t believe it\'s entirely a conclusion. That you have seen no evidence that there IS a God is correct - neither have I. But lack of evidence for the existence of something is in NO WAY evidence for the non-existence of something (the creationist have a similar mode of argumentation in which if they disprove evolution the establish creation). You (personally) have never seen a neutrino before, but they exist. The "pink unicorn" analogy breaks down and is rather naive. I have a scientific theory that explains the appearance of animal life - evolution. When I draw the conclusion that "pink unicorns" don\'t exist because I haven\'t seen them, this conclusion has it\'s foundation in observation and theory. A "pink unicorn", if it did exist, would be qualitatively similar to other known entities. That is to say, since there is good evidence that all life on earth has evolved from "more primitive" ancestors these pink unicorns would share a common anscestory with horses and zebras and such. God, however, has no such correspondence with anything (IMO). There is no physical frame work of observation to draw ANY conclusions FROM.

> >But first let me say the following. > >We might have a language problem here - in regards to "faith" and > >"existence". I, as a Christian, mainta in that God does not exist. > >To exist means to have being in space and t ime. God does not HAVE > >being - God IS Being. Kierkegaard once said that God does not > >exist, He is eternal. With this said, I feel it\'s rather pointless > >to debate the so called "existence" of God - and that is not what > >I\'m doing here. I believe that God is the source and ground of > >being. When you say that "god does not exist", I also accept this > >stat ement - but we obviously mean two different things by it. However, > >in w hat follows I will use the phrase "the existence of God" in it\'s > >\'usu al sense\' - and this is the sense that I think you are using it. > >I wou ld like a clarification upon what you mean by "the existence of > >God". > > > > No, that\'s a word game. I disagree with you profoundly on this. I haven\'t defined God as existence - in fact, I haven\'t defined God. But this might be getting off the subject - although if you think it\'s releva nt we can come back to it. > > Further, saying god is existence is eit her a waste of time, existence is > already used and there is no need to r eplace it by god, or you are > implying more with it, in which case your definition and your argument > so far are incomplete, making it a fallac y. > You are using wrong categories here - or perhaps you misunderstand what I\'m saying. I\'m making no argument what so ever and offering no def inition so there is no fallacy. I\'m not trying to convince you of anythin g. *I* Believe - and that rests upon Faith. And it is inappropriate to app ly the category of logic in this realm (unless someone tells you that they can logically prove God or that they have "evidence" or ..., then the use of logic to disprove their claims if fine and necessary). BTW, an incompl ete argument is not a fallacy - some things are not EVEN wrong. > > (D eletion) > >One can never prove that God does or does not exist. When you say > >that you believe God does not exist, and that this is an opinion > >"based upon observation", I will have to ask "what observtions are > >you refering to?" There are NO observations - pro or con - that > >are valid h ere in establishing a POSITIVE belief. > (Deletion) > > Where does that follow? Aren\'t observations based on the assumption > that something exis ts? > I don\'t follow you here. Certainly one can make observations of things that they didn\'t know existed. I still maintain that one cannot us e observation to infer that "God does not exist". Such a positive assertio n requires a leap. > And wouldn\'t you say there is a level of defini tion that the assumption > "god is" is meaningful. If not, I would reject that concept anyway. > > So, where is your evidence for that "god is" is meaningful at some > level? Once again you seem to completely misunderst and me. I have no EVIDENCE that "\'god is\' is meaningful" at ANY level. M aybe such a response as you gave just comes naturally to you because so ma ny people try to run their own private conception of God down your throat. I, however, am not doing this. I am arguing one, and only one, thing - tha t to make a positive assertion about something for which there can in prin ciple be no evidence for or against requires a leap - it requires faith. I am, as you would say, a "theist"; however, there is a form of atheism that I can respect - but it must be founded upon honesty. > Benedikt -- bob singleton bobs@thnext.mit.edu ']

In [ ]:
```
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]]
```

In [ ]:

```
from sklearn.metrics.pairwise import cosine_similarity

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

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

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

print("Cosine Similarity using count bw 1 and 1: %(x)f" %{"x":cos_sim_cou
print("Cosine Similarity using count bw 2 and 2: %(x)f" %{"x":cos_sim_cou
print("Cosine Similarity using count bw 3 and 3: %(x)f" %{"x":cos_sim_cou
```

```
Cosine Similarity using count bw 1 and 2: 0.627571
Cosine Similarity using count bw 1 and 3: 0.713666
Cosine Similarity using count bw 2 and 3: 0.526166
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
```

```
/var/folders/fq/qps9vdbd4bl6f7tzpv5gfdpm0000gn/T/ipykernel_59293/45277248
5.py:13: DeprecationWarning:

Conversion of an array with ndim > 0 to a scalar is deprecated, and will e
rror in future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)

/var/folders/fq/qps9vdbd4bl6f7tzpv5gfdpm0000gn/T/ipykernel_59293/45277248
5.py:14: DeprecationWarning:

Conversion of an array with ndim > 0 to a scalar is deprecated, and will e
rror in future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)

/var/folders/fq/qps9vdbd4bl6f7tzpv5gfdpm0000gn/T/ipykernel_59293/45277248
5.py:15: DeprecationWarning:

Conversion of an array with ndim > 0 to a scalar is deprecated, and will e
rror in future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)

/var/folders/fq/qps9vdbd4bl6f7tzpv5gfdpm0000gn/T/ipykernel_59293/45277248
5.py:17: DeprecationWarning:

Conversion of an array with ndim > 0 to a scalar is deprecated, and will e
rror in future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)

/var/folders/fq/qps9vdbd4bl6f7tzpv5gfdpm0000gn/T/ipykernel_59293/45277248
5.py:18: DeprecationWarning:

Conversion of an array with ndim > 0 to a scalar is deprecated, and will e
rror in future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)

/var/folders/fq/qps9vdbd4bl6f7tzpv5gfdpm0000gn/T/ipykernel_59293/45277248
5.py:19: DeprecationWarning:

Conversion of an array with ndim > 0 to a scalar is deprecated, and will e
rror in future. Ensure you extract a single element from your array before
performing this operation. (Deprecated NumPy 1.25.)
```

>>> Exercise 20 (take home):

Try changing the texts reference for Text 1, Text 2, and Text 3. What do you observe from the Cosine Similarity results of different text references? following the modifications to texts reference, how can the results of the cosine similarity be interpreted?

In [ ]:
```
#Answer
# After changing the text references for Text 1, Text 2, and Text 3,
# the cosine similarity values changed according to how similar the docum
#
# Texts from the same topic (e.g., both in comp.graphics) showed high sim
# Texts from different topics (e.g., religion vs medicine) had low simila
#
# Cosine similarity measures how much two texts share in word usage, rega
```

```
# A higher value means the texts discuss related subjects or use similar
# a lower value means they differ in topic or vocabulary.
#
# Summary:
# High(≈1):same or related topics
# Medium(≈0.5):partial overlap
# Low(≈0):unrelated content
#
# by changing which documents are compared, cosine similarity quantitativ
# shows how close or distant the texts are in meaning.
```

In [ ]:
```
random_record_1 = X.iloc[5]
random_record_2 = X.iloc[123]
random_record_3 = X.iloc[800]
```

In [ ]:
```
document_vector_count_1 = count_vect.transform(document_to_transform_1)
...
cosine_similarity(document_vector_count_1, document_vector_count_2)
```

Out[ ]:
```
array([[0.62757084]])
```

---

7. Data Classification

religion / graphics / medicine

Data classification is one of the most critical steps in the final stages of the data mining process. After uncovering patterns, trends, or insights from raw data, classification helps organize and label the data into predefined categories. This step is crucial in making the mined data actionable, as it allows for accurate predictions and decision-making. For example, in text mining, classification can be used to categorize documents based on their content, like classifying news articles into categories such as sports, politics, or technology. Among various classification techniques, the **Naive Bayes classifier** is a simple yet powerful algorithm commonly used for text classification tasks. Specifically, the Multinomial Naive Bayes classifier is particularly suited for datasets where features are represented by term frequencies, such as a document-term matrix, like the one we have.

- **Multinomial Naive Bayes:** The Multinomial Naive Bayes classifier works by assuming that the features (words or terms in text data) follow a multinomial distribution. In simple terms, it calculates the probability of a document belonging to a particular category based on the frequency of words in that document, assuming independence between words (the "naive" part of Naive Bayes). Despite this assumption, it often performs remarkably well for text data, especially when working with word count features. Now, when incorporating the binary matrix of patterns we have, it remains compatible because the binary values can be seen as a count of pattern occurrences (1 for present, 0 for absent). Although binary features are not true "counts," the Multinomial Naive

Bayes classifier can still handle them without issue. For more information you can go to: NB Classifier

We will implement a Multinomial Naive Bayes, for that we first choose how to split our data, in this case we will follow a typical **70/30 split for the training and test set**. Let's see a comparison of what we obtain when classifying our data without patterns vs our data with the patterns.

In [ ]:
```
#Model with only the document-term frequency data
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score

# Create a mapping from numerical labels to category names
category_mapping = dict(X[['category', 'category_name']].drop_duplicates(

# Convert the numerical category labels to text labels
target_names = [category_mapping[label] for label in sorted(category_mapp

# Split the data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(tdm_df, X['category']
```

In [ ]:
```
X_train
```

Out[ ]:

	00	000	0000	0000001200	000005102000	0001	000100255pixel	000
634	0	0	0	0	0	0	0	
322	0	0	0	0	0	0	0	
1725	0	0	0	0	0	0	0	
266	0	0	0	0	0	0	0	
1138	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	
1638	0	0	0	0	0	0	0	
1095	0	0	0	0	0	0	0	
1130	0	0	0	0	0	0	0	
1294	0	0	0	0	0	0	0	
860	0	0	0	0	0	0	0	

1579 rows × 35788 columns

In [ ]:
```
X_test
```

Out[ ]:

	00	000	0000	0000001200	000005102000	0001	000100255pixel	000
561	0	0	0	0	0	0	0	
440	0	0	0	0	0	0	0	
1513	0	1	0	0	0	0	0	
1360	0	0	0	0	0	0	0	
259	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	
1552	0	0	0	0	0	0	0	
331	0	0	0	0	0	0	0	
194	0	0	0	0	0	0	0	
2122	0	0	0	0	0	0	0	
2147	0	0	0	0	0	0	0	

678 rows × 35788 columns

In [ ]:
```
# Train the Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb_classifier.predict(X_test)

# Evaluate the classifier
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred,
```

```
Accuracy: 0.9660766961651918

Classification Report:
                          precision    recall  f1-score   support

             alt.atheism     0.9535    0.9535    0.9535       129
           comp.graphics     0.9540    0.9822    0.9679       169
                 sci.med     0.9839    0.9683    0.9760       189
soc.religion.christian     0.9683    0.9581    0.9632       191

                accuracy                         0.9661       678
               macro avg     0.9649    0.9655    0.9651       678
            weighted avg     0.9663    0.9661    0.9661       678
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

invalid value encountered in matmul
```

>>> Exercise 21 (take home):

Previously, we performed data augmentation by concatenating the original TDM with the pattern matrix. Use the augmented data to perform classification using Naive Bayes as in the previous steps.

In [ ]:
```
#Answer
# We used the augmented dataset, combining the original term-document mat
# to train a Naive Bayes classifier.
# The resulting accuracy slightly improved compared to the baseline model
# showing that pattern features provided additional context for classific
```

In [184…
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score

# Step 1: Split the augmented data (70/30)
X_train, X_test, y_train, y_test = train_test_split(augmented_df, X['cate

# Step 2: Train Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

# Step 3: Predict
y_pred = nb_classifier.predict(X_test)

# Step 4: Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred,
```

```
Accuracy: 0.9734513274336283

Classification Report:
                          precision    recall  f1-score   support

           alt.atheism       0.98      0.94      0.96       129
         comp.graphics       0.99      0.98      0.99       169
               sci.med       0.98      0.98      0.98       189
soc.religion.christian       0.94      0.98      0.96       191

              accuracy                           0.97       678
             macro avg       0.98      0.97      0.97       678
          weighted avg       0.97      0.97      0.97       678
```

```
/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

divide by zero encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

overflow encountered in matmul

/Users/parinmac/Documents/Assignments/DM2025-Lab1-Exercise/.venv/lib/pytho
n3.11/site-packages/sklearn/utils/extmath.py:203: RuntimeWarning:

invalid value encountered in matmul
```

>>> Exercise 22 (take home):

What differences can be observed between the results obtained using the original TDM and those obtained using the augmented TDM?

In [ ]:
```
# Answer
# Comparison between original TDM and augmented TDM results

# Original TDM → accuracy = 0.9660766961651918
# Augmented TDM → accuracy slightly improved (accuracy: 0.973451327433628

# Interpretation
# 1. The improvement shows that adding pattern-based features enriches th
# 2. It helps Naive Bayes capture contextual patterns not reflected in si
# 3. However, the gain is modest, indicating the baseline model already p
```

Decision Tree Classifier

Another popular classification technique is the **Decision Tree classifier**. Decision Trees work by recursively splitting the data into subsets based on feature values, creating a tree-like model of decisions. Each node in the tree represents a feature, and each branch represents a decision rule. The leaves represent the final class labels.

- **Decision Tree:** Decision Trees are intuitive and easy to visualize. They can handle both numerical and categorical data and do not require feature scaling. However, they can be prone to overfitting, especially with high-dimensional data. In text classification, Decision Trees can be used with document-term matrices or augmented feature sets, just like Naive Bayes.

In the following code, we will train and evaluate a Decision Tree classifier on both the document-term matrix and the augmented data (with patterns), allowing us to compare its performance with the Naive Bayes model.

We will implement a Multinomial Naive Bayes, for that we first choose how to split our data, in this case we will follow a typical **70/30 split for the training and test set**. Let's see a comparison of what we obtain when classifying our data without patterns vs our data with the patterns.

In [155…
```
# Decision Tree with only the document-term frequency data
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score

# Split the data (already done above, but shown here for clarity)
X_train, X_test, y_train, y_test = train_test_split(tdm_df, X['category']

# Train Decision Tree
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

# Predict and evaluate
y_pred_dt = dt_classifier.predict(X_test)
print("Decision Tree Accuracy (TDM):", accuracy_score(y_test, y_pred_dt))
print("\nClassification Report:\n", classification_report(y_test, y_pred_
```

```
Decision Tree Accuracy (TDM): 0.7713864306784661

Classification Report:
                          precision    recall  f1-score   support

           alt.atheism     0.7500    0.7907    0.7698       129
         comp.graphics     0.7486    0.7929    0.7701       169
               sci.med     0.7571    0.7090    0.7322       189
soc.religion.christian     0.8226    0.8010    0.8117       191

              accuracy                         0.7714       678
             macro avg     0.7696    0.7734    0.7710       678
          weighted avg     0.7721    0.7714    0.7712       678
```

>>> Exercise 23 (take home):

Why do you think the performance of Decision Trees is lower than that of Naive Bayes? Provide possible reasons.

In [ ]:
```
# Answer
# The Decision Tree classifier performs worse than Naive Bayes because of
# Decision Trees try to split data again and again based on every word,
# so with thousands of rare words, the tree becomes huge and learns even
```

```
# Naive Bayes, instead, looks at how likely each word appears in a catego
# This method ignores the exact structure but captures the overall trend,
```

>>> Exercise 24 (take home):

Implement using the augmented data, does the performance improve compared to that obtained with the original TDM? Is it better than Naive Bayes? Provide possible explanations for your observations.

In [ ]:
```
# Answer
# The augmented Decision Tree model showed slightly lower accuracy (0.765
# This drop may be due to overfitting caused by the additional pattern fe
# While augmented features can enrich semantic information, they also inc
# which Decision Trees handle less effectively compared to probabilistic

#     augmented data
# Decision Tree               →             →
#                           0.771 → 0.765
```

In [275…
```
# Decision Tree with AUGMENTED data (TDM + pattern features)

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Step 1: Combine TDM and pattern features
aug_tdm_df = pd.concat([tdm_df, pattern_matrix], axis=1)

# Step 2: Split the data (70/30)
X_train, X_test, y_train, y_test = train_test_split(
    aug_tdm_df,
    X['category'],
    test_size=0.3,
    random_state=42
)

# Step 3: Train Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

# Step 4: Predict
y_pred = dt_classifier.predict(X_test)

# Step 5: Evaluate performance
print("Decision Tree Accuracy (AUGMENTED):", accuracy_score(y_test, y_pre
print("\nClassification Report:\n",
      classification_report(y_test, y_pred, target_names=target_names, di
```

```
Decision Tree Accuracy (AUGMENTED): 0.7654867256637168

Classification Report:
                          precision    recall  f1-score   support

           alt.atheism     0.7609    0.8140    0.7865       129
         comp.graphics     0.7257    0.7515    0.7384       169
               sci.med     0.7287    0.7249    0.7268       189
soc.religion.christian     0.8475    0.7853    0.8152       191

              accuracy                         0.7655       678
             macro avg     0.7657    0.7689    0.7667       678
          weighted avg     0.7675    0.7655    0.7660       678
```

>>> Exercise 25 (take home):

In your opinion, how should one choose the best model for a given task, such as the classification example above?

In [ ]:
```
# Answer
# Model selection should depend on:
# 1. Data characteristics
#    Naive Bayes is suitable for high-dimensional sparse text. (
#    Decision Trees for structured numeric data.
# 2. Task goal
#    if interpretability is important, Decision Tree is preferred
#    if accuracy and generalization matter, Naive Bayes is better
# 3. Feature types – models perform differently with word frequency vs. p
```

---

8. 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 Pandas, Scikit-learn, UMAP and PAMI. 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.

---

9. References

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