National College of Ireland

Configuration Manual

MSc Research Project
MSc AI Top-up

Gabriel Amariei
Student ID: 13130510

School of Computing
National College of Ireland

Supervisor: Faithful Onwuegbuche

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Gabriel Amariei		
Student ID:	13130510		
Programme:	MSc AI Top-up	**Year:**	2024
Module:	MSc Research Project		
Lecturer:	Faithful Onwuegbuche		
Submission Due Date:	12/08/2024		
Project Title:	Configuration Manual		
Word Count:	**xyz**	**Page Count:**	26

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Gabriel Amariei
Date: 12/08/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Gabriel Amariei
Student ID: 13130510

Table of Contents

1 Introduction

This Configuration manual represents all the necessary requirements and details for the Project "*Document Clustering of Irish Government Circulars using Machine Learning Techniques*". It is structured as follows: the next section shows the hardware requirements followed by the software and library package requirements needed to run the project. The next section provides a description of the data and its location. The rest of the sections are divided into parts of workflow of the code, their execution, result and their evaluation.

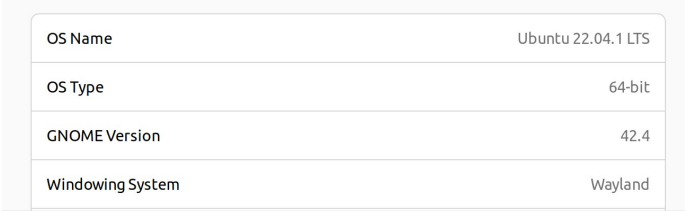
2 Hardware Requirements

The configuration of the machine used for this project can be found below.

.

3 Software Requirement

The operating system was Ubuntu.



Python 3.9.19 with the Jupyter lab development environment .

```
1 print (sys.version)
```

```
3.9.19 (main, May  6 2024, 19:43:03)
[GCC 11.2.0]
```

4 Package Requirements

The following python packages were imported:

```
import requests
from bs4 import BeautifulSoup
import os
```

```
import re
import pandas as pd
from spire.doc import *
from spire.doc import Document
from spire.doc.common import *
import os
import numpy as np
import pdfplumber
import matplotlib
import matplotlib.pyplot as plt
from pylab import rcParams
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk import word_tokenize
import nltk
import re
import string
import warnings
import random
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
from sklearn.metrics import davies_bouldin_score
from sklearn.metrics import silhouette_samples
import skfuzzy as fuzz

from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import LSTM, Dense, Input, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
from tensorflow.keras.layers import LayerNormalization

from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
from sklearn.metrics.cluster import normalized_mutual_info_score
from sklearn.preprocessing import LabelEncoder

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
import plotly.express as px
import plotly.graph_objects as go

import cufflinks as cf
from plotly.offline import iplot, init_notebook_mode
import gensim
from gensim.models import Word2Vec
```

from transformers import BertTokenizer, BertModel
import torch
import pickle

5 Data Download code

Step one: Download the individual links to the circulars
File: ***1.Download documents with dat info df.ipynb***

1. Download all individual page links

[n [2]:
```
base_url='https://www.gov.ie/en/circulars/?referrer=http%3A%2F%2Fwww.gov.ie%2Fcirculars%2F&page=1'
total_pages=350
circulars_links=[]
```

[n [3]:
```
for page_num in range(1, total_pages + 1):
    page_url = f"{base_url}&page={page_num}"
    response1 = requests.get(page_url)
    soup1 = BeautifulSoup(response1.text, 'html.parser')
    for link in soup1.find_all('a', href=True):
            href = link['href']
            if href.startswith('/en/circular/'):
                circulars_links.append(href)
```

Step 2: Download pdf, doc and docx type documents

```
#create function to scrape data info

def find_data(soup_tag, text):
    selected_tags = soup2.find_all(soup_tag)
    abc_span = None
    for tag in selected_tags:
        if text in tag.get_text(strip=True):
            # Remove the text from the tag's content
            abc_span = tag.get_text(strip=True).replace(text, '').strip()
            break
    return abc_span
```

```
circulars_info=pd.DataFrame(columns=('Link_n','Circular_n', 'page_title', 'From', 'Published_on',
                                     'Effective_from',
                                     'Effective_to', 'File_name', 'Sector', 'Author', 'Followed_by',
                                     'Succeeds', 'Succeeded_by', 'Succeeds_L','Succeeded_by_L', 'url'))
```

Iterate trough each link and download the documents and scrape further info for each circular from the website.

```
base_link='https://www.gov.ie'
 # Maximum allowed length for file name as some titles are too long to use as document name
MAX_FILE_NAME_LENGTH = 70

link_n=0

for item in circulars_links:
    url=base_link+item
    print(url)
    # Requests URL and get response object
    response2 = requests.get(url)
    # Parse text obtained
    soup2 = BeautifulSoup(response2.text, 'html.parser')
    # Find all hyperlinks present on webpage
    links = soup2.find_all('a')
    i = 0
    #page_title = soup2.find('title')
    page_title = soup2.find('h1')

    #n of link included to avoid replacing documents with same truncated name
    link_n=link_n+1

    Circular_n=find_data('span', 'Circular')
    From=find_data('p', 'From')
    Published_on=find_data('p', 'Published on')
    Effective_from=find_data('p', 'Effective from')
    Effective_to=find_data('p', 'Effective to')
    Sector=find_data('p', 'Sector')
    Author=find_data('p', 'Author')
    Followed_by=find_data('p', 'Followed by')
    Succeeds=find_data('p', 'Succeeds')
    Succeeded_by=find_data('p', 'Succeeded by')
    Succeeds_L=find_data('p', 'succeeds')
    Succeeded_by_L=find_data('p', 'succeeded by')

```

```
    for link in links:
        href=link.get('href')

        if href and href.endswith(('.pdf','.doc', '.docx')):  # Check if href is not None before calling endswith
            i += 1
            print("Downloading file: ", i)

            # Get response object for link
            response = requests.get(link.get('href'))

            cleaned_title= re.sub(r'[,\\/*?:"<>|]', '', page_title.string)

            # Truncate the title to fit within the maximum file name length
            truncated_title = cleaned_title[:MAX_FILE_NAME_LENGTH - len(str(i)) - 5]  # Subtracting length of "_{
            file_extension = os.path.splitext(href)[1]

            pdf_name = f"{link_n}_{truncated_title}_{i}{file_extension}"


            ### Write content in files

            with open(pdf_name, 'wb') as pdf:
                pdf.write(response.content)

            pdf.close()

            ## populate info circular info data frame

            df_new_row=pd.DataFrame({'Link_n':[link_n],'Circular_n':[Circular_n],
                                     'page_title':[cleaned_title], 'From':[From],
                                     'Published_on':[Published_on], 'Effective_from':[Effective_from],
                                     'Effective_to':[Effective_to], 'File_name':[pdf_name],
                                     'Sector':[Sector], 'Author':[Author], 'Followed_by':[Followed_by],
                                     'Succeeds':[Succeeds], 'Succeeded_by':[Succeeded_by],
                                     'Succeeds_L':[Succeeds_L], 'Succeeded_by_L':[Succeeded_by_L],'url':[url]
                                    })

            circulars_info=pd.concat([circulars_info, df_new_row], ignore_index=True)

            print("File ", pdf_name, " downloaded")


## write info into files (csv and xlsx)

circulars_info.to_csv('/home/ga/Desktop/project MSc AI/data/circulars_info.csv')
circulars_info.to_excel('/home/ga/Desktop/project MSc AI/data/circulars_info.xlsx',
                        sheet_name='circulars_links', index=False)

```

6 Initial exploratory data analysis

File: ***2. circulars_info_clean Exploratory data analysis.ipynb***

```
circulars_info=pd.read_csv('/home/ga/Desktop/project MSc AI/data/circulars_info.csv', index_col = [0])
```

```
#set default parameters for chart

import matplotlib
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
from pylab import rcParams
matplotlib.rcParams['axes.labelsize'] = 8
matplotlib.rcParams['xtick.labelsize'] = 7
matplotlib.rcParams['ytick.labelsize'] = 7
matplotlib.rcParams['text.color'] = 'k'


rcParams['figure.figsize'] = 10, 7
```

```
circulars_info.head()
```

	Link_n	Circular_n	page_title	From	Published_on	Effective_from	Effective_to	File_name	Sector	Author	Followed_by	Succeed
0	1	12/2024	Circular 122024 Arrangements for Occupational ...	Department of Public Expenditure, NDP Delivery...	24 July 2024	NaN	NaN	1_Circular 122024 Arrangements for Occupationa...	NaN	NaN	NaN	Na
1	2	LG	LG 08-2024 Guidelines for	Department of Housing, Local	17 July 2024	NaN	NaN	2_LG 08-2024 Guidelines for	NaN	LG 08-2024 Guidelines	NaN	Na

```
circulars_info.shape
```

```
(4017, 16)
```

```
circulars_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4017 entries, 0 to 4016
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
```

plot NAs

```
def plot_nas(df: pd.DataFrame):
    if df.isnull().sum().sum() != 0:
        na_df = (df.isnull().sum() / len(df)) * 100
        na_df = na_df.drop(na_df[na_df == 0].index).sort_values(ascending=False)
        missing_data = pd.DataFrame({'Missing Ratio %' :na_df})
        missing_data.plot(kind = "barh")
        plt.show()
    else:
        print('No NAs found')
plot_nas(circulars_info)
```

```
#drom empty columns
drop_col=['Succeeds', 'Succeeds_L' , 'Succeeded_by', 'Succeeded_by_L']
circulars_info=circulars_info.drop(drop_col, axis=1)
```

```
circulars_info1=circulars_info
```

```
# Added column 'Year_published' for further analysis
circulars_info1['Year_published']= circulars_info1['Published_on'].str[-4:]
```

```python
#transform the Published_on column to date time
circulars_info1['Published_on'] = pd.to_datetime(circulars_info1['Published_on'])
```

```python
# Pivoted circulars_info1 to get the number of circulars per department

circular_counts = circulars_info1.groupby(['From'])['Circular_n'].nunique().reset_index(name='Count')
# Print the result
print(circular_counts)
```

```python
circular_counts['From'] = circular_counts['From'].str.replace('Department', 'Dep')
#circular_counts.plot(kind='bar', x='From', y='Count')

# Plot the results
plt.figure(figsize=(10, 6))
ax = circular_counts.plot(kind='bar',  x='From', y='Count')
plt.xticks(rotation=45)

# Add the number on top of each column
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='center',

plt.tight_layout()
plt.show()

```

```python
#print the number of circulars per year

pd.set_option('display.max_rows', None)

df_c3=circulars_info1.pivot_table(values=['Circular_n'],
                            index=[ 'Published_on'],
                          aggfunc=pd.Series.nunique
                        )
df_c3

df_c3.resample('YE',level=0).sum().plot()
#print(df_c3.resample('Y',level=0).sum())
```

```python
# Convert entire data frame as markdown and print
print(df_c3.resample('YE',level=0).sum().to_markdown())
```

```python
# Add column 'Document_type'

# Use np.where to conditionally assign values based on a condition
circulars_info1['Document_type'] = np.where(circulars_info1['File_name'].str[-4:] == 'docx', 'docx',
                                            circulars_info1['File_name'].str[-3:])
# Group by 'File_type' and count occurrences
file_type_counts = circulars_info1['Document_type'].value_counts()

# Plot the results
plt.figure(figsize=(10, 6))
ax = file_type_counts.plot(kind='bar')
plt.xlabel('File Type')
plt.ylabel('Number of Documents')
plt.title('Number of Documents by File Type')
plt.xticks(rotation=45)

# Add the number on top of each column
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='center',

plt.tight_layout()
plt.show()
```

```python
# Replace illegal characters in the entire DataFrame
circulars_info1_cleaned = circulars_info1.replace(r'\x02', '', regex=True)

#circulars_info1.to_excel('/home/ga/Desktop/project MSc AI/data/circulars_info.xlsx', sheet_name='circulars_info'


# Replace illegal characters in the entire DataFrame
#circulars_info1_cleaned = circulars_info1.replace(r'\x02', '', regex=True)

# Save the cleaned DataFrame to Excel
circulars_info1_cleaned.to_excel('/home/ga/Desktop/project MSc AI/data/circulars_info1.xlsx', sheet_name='circula

```

```
/tmp/ipykernel_1959020/2346982939.py:11: UserWarning: Pandas requires version '3.0.5' or newer of 'xlsxwriter' (vers
ion '3.0.3' currently installed).
  circulars_info1_cleaned.to_excel('/home/ga/Desktop/project MSc AI/data/circulars_info1.xlsx', sheet_name='circular
s_info', index=False)
```

```python
circulars_info1.to_csv('/home/ga/Desktop/project MSc AI/data/circulars_info1.csv')
```

7 Convert doc and docx files to pdf

File: *3. Convert doc and docx files to pdf.ipynb*

```
# Directory containing DOC and DOCX files
directory = '/home/ga/Desktop/project MSc AI/data/downloaded files'
```

```
#function to convert file to pdf
def convert_to_pdf(file_path, pdf_file):
    try:
        # Load the document
        document = Document()
        document.LoadFromFile(file_path)

        # Save the document as PDF
        document.SaveToFile(pdf_file, FileFormat.PDF)
        return True
    except Exception as e:
        print(f"Error occurred while converting {file_path} to PDF: {e}")
        return False

    #function to convert batch to pdf
def batch_convert_to_pdf(directory):
    success_count = 0
    error_count = 0

    for filename in os.listdir(directory):
        if filename.endswith(".docx") or filename.endswith(".doc"):
            file_path = os.path.join(directory, filename)
            pdf_file = os.path.splitext(file_path)[0] + '.pdf'
            if convert_to_pdf(file_path, pdf_file):
                success_count += 1
            else:
                error_count += 1

    print(f"Successfully converted {success_count} files to PDF.")
    print(f"Failed to convert {error_count} files.")

```

```
# run the code
batch_convert_to_pdf(directory)
```

8 Extract text and metadata from pdf

File: *4. extract data from pdf.ipynb*

```
# create Function to count words in a text
#https://www.geeksforgeeks.org/count-words-in-a-given-string/

def countWords(s):

    # Check if the string is null
    # or empty then return zero
    if s.strip() == "":
        return 0

    # Splitting the string

    words = s.split()

    return len(words)
#def count_words(text):
    #return len(text.split())
```

```
# create Function to count words in a text
#https://www.geeksforgeeks.org/count-words-in-a-given-string/

def countWords(s):

    # Check if the string is null
    # or empty then return zero
    if s.strip() == "":
        return 0
    # Splitting the string
    words = s.split()
    return len(words)
```

```python
#create function to extract pdf info including text

def extract_pdf_info(pdf_file):
    try:
        # Ensure pdf_file is a string representing the file path
        if not isinstance(pdf_file, str):
            raise ValueError("pdf_file must be a string representing the file path.")

        # Open the PDF file
        with pdfplumber.open(pdf_file) as pdf:
            # Extract document information
            doc_info = {
                'Document Name': os.path.basename(pdf_file),
                'Number of Pages': len(pdf.pages),
                'File Size (bytes)': os.path.getsize(pdf_file),
                'Author_pdf': pdf.metadata.get('Author', None),
                'Creator_pdf': pdf.metadata.get('Creator', None),
                'Producer_pdf': pdf.metadata.get('Producer', None),
                'Subject_pdf': pdf.metadata.get('Subject', None),
                'Title_pdf': pdf.metadata.get('Title', None)
            }

            # Extract text from each page
            results = []
            for page in pdf.pages:
                text = page.extract_text()
                if text:
                    results.append(text)
            end_text = ''.join(results)

            doc_info['word_count'] = countWords(end_text)
            doc_info['Text'] = end_text

    except Exception as e:
        print(f"Error occurred while reading {pdf_file}: {e}")
        doc_info = None

    return doc_info
```

```python
# Directory containing PDF files
pdf_directory = '/home/ga/Desktop/project MSc AI/data/downloaded files'

# List to store extracted information
pdf_info_list = []
failed_count = 0

```

```python
# Iterate through all PDF files in the directory and extract data

for filename in os.listdir(pdf_directory):
    if filename.endswith('.pdf'):
        pdf_file = os.path.join(pdf_directory, filename)
        pdf_info = extract_pdf_info(pdf_file)
        if pdf_info is None:
            failed_count += 1
        else:
            pdf_info_list.append(pdf_info)

print("Number of files failed to read:", failed_count)
```

```python
# Filter out None values from the list if needed
#pdf_info_list1 = [info for info in pdf_info_list if info is not None]

# Create DataFrame from the list of extracted information
pdf_df = pd.DataFrame(pdf_info_list)


```

```python
# Save DataFrame to CSV file
csv_file = '/home/ga/Desktop/project MSc AI/data/pdf_info_and_text.csv'
xlsx_file = '/home/ga/Desktop/project MSc AI/data/pdf_info_and_text.xlsx'
```

```python
circulars_info.to_csv(csv_file)
circulars_info.to_excel(xlsx_file, sheet_name='data', index=False)
```

```python
# Check for duplicate rows
duplicate_rows = pdf_df[pdf_df.duplicated()]
print("Number of duplicate rows:", duplicate_rows.shape[0])

# Display duplicate rows if any
if not duplicate_rows.empty:
  print(duplicate_rows)
```

```
Number of duplicate rows: 0
```

```python
print(len(pdf_df))
```

```
3841
```

```python
# Replace Evaluation Warning: The document was created with Spire.Doc for Python. with  in pdf_df
pdf_df2 = pdf_df.astype(str).replace(
    "(?i)Evaluation Warning: The document was created with Spire.Doc for Python.", "",
                                     regex=True).astype(pdf_df.dtypes.to_dict())

```

```python
# Apply the count_words function to each row of the Text column and create a new column with word counts
pdf_df2['word_count_2'] = pdf_df2['Text'].apply(lambda x: countWords(x))


```

```python
# Group the DataFrame by word counts and summarize the data
summary_df = pdf_df2.groupby('word_count_2').agg({'Text': 'count'}).reset_index()
summary_df.columns = ['Word Count', 'Number of Texts']

#check to see the number of words
print(summary_df.to_string())
```

```python
#Histogram for word count
# Modify the 'word_count' column to group documents with over 10000 words
pdf_df2['word_count_grouped'] = np.where(pdf_df2['word_count'] > 10000, 10000, pdf_df2['word_count'])

# Plot the histogram with the modified data, removing the string conversion
pdf_df2['word_count_grouped'].plot(kind='hist',bins=100, title='word_count') # Removed .astype(str)
plt.gca().spines[['top', 'right']].set_visible(False)
plt.show()
```

```python
less_than_100 = pdf_df2[pdf_df2['word_count'] < 100]['word_count'].count()
print("Number of documents with less than 100 words:", less_than_100)
```

```
Number of documents with less than 100 words: 69
```

```python
over_10000 = pdf_df2[pdf_df2['word_count'] > 10000]['word_count'].count()
print("Number of documents more than than 10000 words:", over_10000)
```

```
Number of documents more than than 10000 words: 79
```

```python
#as thee are 69 documents with less than 100 words and 79 with over 10000 words
# we select only documents with word count of more than 100 words and which contain the first document in the
pdf_df3 = pdf_df2[(pdf_df2['word_count_2'] > 100) & (pdf_df2['Document Name'].str.contains('_1.pdf'))]
```

```python
len(pdf_df3)
```

```
3304
```

```python
# Modify the 'word_count' column to group documents with over 10000 words
pdf_df3['word_count_grouped'] = np.where(pdf_df3['word_count_2'] > 10000, 10000, pdf_df3['word_count_2'])

# Plot the histogram with the modified data, removing the string conversion
pdf_df3['word_count_grouped'].plot(kind='hist',bins=100, title='word_count') #
plt.gca().spines[['top', 'right']].set_visible(False)
plt.show()
```

```python
#most pdf documents did not have an author with the highest stated being the Dep of Finance
# Count the occurrences of each Author_pdf
author_counts = pdf_df3['Author_pdf'].value_counts()

# Select the top 10 authors
top_10_authors = author_counts.head(10)

# Plot the bar chart
plt.figure(figsize=(10, 6))
ax=top_10_authors.plot(kind='bar')
plt.title('Top 10 Authors')
plt.xlabel('Author')
plt.ylabel('Number of Documents')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability


# Add the number on top of each column
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='center',

plt.tight_layout()# Adjust layout to prevent labels from overlapping
plt.show()
```

merge the circulars info and the pdf_df3

```
file_to_merge = '/home/ga/Desktop/project MSc AI/data/circulars_info1.csv'
#xlsx_file = '/home/ga/Desktop/readings master prep/project/data/circulars_info.xlsx'
```

```


circulars_info=pd.read_csv(file_to_merge)
circulars_info = circulars_info.drop(columns=['Unnamed: 0'])
```

```
# the file name works as unique identifier of the documents
#in order to merge the circulars_info and the data from the Pdf_df3 we will replace the extension of the doc and

def change_extension(filename):
  if filename.endswith('.doc'):
    return filename[:-4] + '.pdf'  # Replace .doc with .pdf
  elif filename.endswith('.docx'):
    return filename[:-5] + '.pdf'  # Replace .docx with .pdf
  else:
    return filename

circulars_info['Document Name'] = circulars_info['File_name'].apply(change_extension)
```

```
# Perform the merge
merged_df = pdf_df3.merge(circulars_info, on='Document Name', how='left')
```

```
merged_df[['Sector','Author']].head(5)
```

```
# Remove ':' and leading spaces from 'Sector' and 'Author' columns
merged_df['Sector'] = merged_df['Sector'].str.lstrip(': ').str.strip()
merged_df['Author'] = merged_df['Author'].str.lstrip(': ').str.strip()
```

```
# Replace illegal characters with a suitable replacement (e.g., a space)
merged_df = merged_df.replace('\x02', ' ', regex=True)
```

```
#transform the Published_on column to date time
merged_df['Published_on'] = pd.to_datetime(merged_df['Published_on'])
```

```
# Save DataFrame to CSV file
csv_file =  '/home/ga/Desktop/project MSc AI/data/pdf_df3_1.csv'
xlsx_file = '/home/ga/Desktop/project MSc AI/data/pdf_df3_1.xlsx'
```

```
merged_df.to_excel(xlsx_file, sheet_name='files_detail', index=False)
merged_df.to_csv(csv_file, index=False)

print("PDF information saved to:", csv_file)
```

```
#Create chart for the departments that published the Circulars as per cleaned data
merged_df['From']=merged_df['From'].str.replace('Department', 'Dep')

# Count the occurrences of each Department
department_counts = merged_df['From'].value_counts()

# Select the top 10 departments
top_10_departments = department_counts.head(10)

# Plot the bar chart
plt.figure(figsize=(10, 6))
bars = top_10_departments.plot(kind='bar')
plt.title('Top 10 Departments')
plt.xlabel('Department')
plt.ylabel('Number of Documents')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability

# Add count labels on top of each bar
for bar in bars.patches:
    bars.annotate(format(bar.get_height(), '.0f'),
                  (bar.get_x() + bar.get_width() / 2, bar.get_height()),
                  ha='center', va='center',
                  size=10, xytext=(0, 8),
                  textcoords='offset points')

plt.tight_layout()  # Adjust layout to prevent labels from overlapping
plt.show()
```

```
#plot the number of documents in the corpus per year published

pd.set_option('display.max_rows', None)

df=merged_df.pivot_table(values=['Circular_n'],
                         index=[ 'Published_on'],
                      aggfunc=pd.Series.nunique
                    )


df.resample('YE',level=0).sum().plot()
```

```
# Convert entire data frame as markdown and print year
print(df.resample('YE',level=0).sum().to_markdown())
```

9 Clean and vectorize the text

File: *5. Clean and vectorize the text.ipynb*

```
#import the data


csv_file3 = '/home/ga/Desktop/project MSc AI/data/pdf_df3_1.csv'

pdf_df4=pd.read_csv(csv_file3, index_col='Document Name')
```

2. Text pre-processing

```

## Download NLTK resources (stopwords and WordNet corpus) if they were not downloaded
#nltk.download('stopwords')
#nltk.download('wordnet')
#nltk.download('punkt')
```

```
#https://stackoverflow.com/questions/54396405/how-can-i-preprocess-nlp-text-lowercase-remove-sp




lemmatizer = WordNetLemmatizer()

# set the stopwords dictionary
custom_stopwords = set(stopwords.words('english') + ['circular', 'education'])

#create function to clean the text with the arguments being the text to tokenize,
#the tokenizer and the stopwords dictionary
#function returns the cleaned ext

def clean_text(text, tokenizer, stopwords):

    text = str(text).lower()  # Lowercase words
    text = re.sub(r"\[(.*?)\]", "", text)  # Remove [+XYZ chars] in content
    text=re.sub(r'http\S+', '',text) #remove web links
    text = re.sub(r"\s+", " ", text)  # Remove multiple spaces tabs, and line breaks in content
    text = re.sub(r"\w+…|…", "", text)  # Remove ellipsis (and last word)
    text = re.sub(r"(?<=\w)-(?=\w)", " ", text)  # Replace dash between words
    text = re.sub('[0-9]+', '', text) #Remove numbers
    text = re.sub(f"[{re.escape(string.punctuation)}]", "", text)  # Remove punctuation

    tokens = tokenizer(text)  # Get tokens from text
    tokens = [t for t in tokens if not t in stopwords]  # Remove stopwords
    tokens = ["" if t.isdigit() else t for t in tokens]  # Remove digits
    #Lemmatization - to return the base or dictionary form of a word, which is known as the lem
    tokens=[lemmatizer.lemmatize(w) for w in tokens]

    tokens = [t for t in tokens if len(t) > 1]  # Remove short tokens
    return " ".join(tokens)
```

```
#run the function
pdf_df4['cleanText']= pdf_df4['Text'].map(lambda x: clean_text(x, word_tokenize, custom_stopwords))
```

```
# Tokenize the text data
pdf_df4['tokenizedText'] = pdf_df4['cleanText'].apply(word_tokenize)

```

```
#function to display HTML

from IPython.core.display import HTML, display
def visualise1(title, content):
    text = ' '
    display(HTML(f'<h1>{title}</h1>'))
    text += ' ' + content.replace('\n', '<br>')
    display(HTML(f'''<p>{text}</p>'''))
```

```
pdf_df4['Text'][2]
```

```
#see the text
visualise1(pdf_df4['File_name'][5], pdf_df4['Text'][5])
```

```
# see the cleaned text
visualise1(pdf_df4['File_name'][5], pdf_df4['cleanText'][5])

#plot the document length destribution

pdf_df4['word_count_clean'].iplot(
    kind='hist',
    bins=100,
    xTitle='Clean text word count',
    yTitle='Count',
    title='Cleaned Text Word Count Distribution',
    colors='green'
)
```

```
#print the top 20 words in the original text

common_words = get_top_n_words(pdf_df4['Text'], 20)

#for word, freq in common_words:
    #print(word, freq)

df1 = pd.DataFrame(common_words, columns = ['Words' , 'count'])

custom_iplot(df1, 'Words', title='Top 20 words in review before cleaning the text')
```

```
#plot top 20 words after leaning the text

common_words = get_top_n_words(pdf_df4['cleanText'], 20)

#for word, freq in common_words:
    #print(word, freq)

df2 = pd.DataFrame(common_words, columns = ['Words' , 'count'])

custom_iplot(df2, 'Words', title='Top 20 words in review after cleaning the text')
```

```
#print most frequent bigrams in te original text
common_words = get_top_n_bigram(pdf_df4['Text'], 20,2)

#for word, freq in common_words:
    #print(word, freq)
df3 = pd.DataFrame(common_words, columns = ['Words' , 'count'])
custom_iplot(df3, 'Words', title='Top 20 bigrams before cleaning the text')
```

```
#print most frequent bigrams in the cleaned text

common_words = get_top_n_bigram(pdf_df4['cleanText'], 20,2)

#for word, freq in common_words:
    #print(word, freq)
df4 = pd.DataFrame(common_words, columns = ['Words' , 'count'])
custom_iplot(df4, 'Words', title='Top 20 bigrams after cleaning the text')
```

```
##get ngrams before and after removing stop words
# plot Top 20 trigrams before cleaning the text

common_words = get_top_n_bigram(pdf_df4['Text'], 20,3)

#for word, freq in common_words:
    #print(word, freq)
df5 = pd.DataFrame(common_words, columns = ['Words' , 'count'])
custom_iplot(df5, 'Words', title='Top 20 trigrams before cleaning the text')
```

```
#print most frequent trigrams in the cleaned text
common_words = get_top_n_bigram(pdf_df4['cleanText'], 20,3)

#for word, freq in common_words:
    #print(word, freq)
df6 = pd.DataFrame(common_words, columns = ['Words' , 'count'])
custom_iplot(df6, 'Words', title='Top 20 trigrams in the cleaned the text')
```

```
#save the file for further reference
csv_file_pdf_df4 = '/home/ga/Desktop/project MSc AI/data/pdf_df4.csv'
xlsx_file_pdf_df4 = '/home/ga/Desktop/project MSc AI/data/pdf_df4.xlsx'
```

3. Text vectorization

3.1 Text vectorization using TF-IDF

```
# TF-IDF document frequency function
vectorizer = TfidfVectorizer(
    min_df = 0.01, #to eliminate words that are not too common - ignore terms that ap
    max_df = 0.97, #to eliminate words that are too common - ignore terms that appear
    #max_features = 5000,
    #stop_words = 'english', #remove the english stopwords - done already
    #token_pattern=u'(?ui)\\b\\w*[a-z]+\\w*\\b' # eliminate words that contain number
)
```

```
# Fit and transform the text data using TF-IDF vectorizer
tfidf_matrix = vectorizer.fit_transform(pdf_df4['cleanText'])
```

```
tfidf_matrix
```

```
<3304x4125 sparse matrix of type '<class 'numpy.float64'>'
        with 966753 stored elements in Compressed Sparse Row format>
```

```
# Print how many rows and columns of the TF-IDF matrix consists
print("n_documents: %d, n_features: %d" % tfidf_matrix.shape)
```

```
n_documents: 3304, n_features: 4125
```

```
# Select the first ten documents from the data set
#see https://okan.cloud/posts/2022-01-16-text-vectorization-using-python-tf-idf/

tf_idf_df = pd.DataFrame(tfidf_matrix.todense())
tf_idf_df.columns = vectorizer.get_feature_names_out()
#tf_idf_df.index=pdf_df4['Document name']
```

```
tf_idf_df.tail(3)
```

```
#save only the matrix
tfidf_matrix_save = pd.DataFrame(tfidf_matrix.toarray(), columns = vectorizer.get_feature_names_out())
```

```
tfidf_matrix_save.to_csv('/home/ga/Desktop/project MSc AI/data/tf_idf_df.csv', index=True)
```

```
# save the tf-idf model
import pickle
```

```
# save the model to disk
#https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/
filename = '/home/ga/Desktop/project MSc AI/data/tfidf_matrix_model.sav'
pickle.dump(tfidf_matrix, open(filename, 'wb'))
```

```
# load the model from disk
tfidf_matrix_from_file = pickle.load(open(filename, 'rb'))
```

3.2 Text vectorization word2vec

https://www.tensorflow.org/text/tutorials/word2vec https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/ https://dylancastillo.co/nlp-snippets-cluster-documents-using-word2vec/

9]:
```
import gensim
from gensim.models import Word2Vec
```

```
# Train Word2Vec model
word2vec_model = Word2Vec(pdf_df4['tokenizedText'], min_count=50,
                          vector_size=100,epochs=100, window=5, sg=1, workers=1, seed=SEED)


#https://www.askpython.com/python-modules/gensim-word2vec
#https://radimrehurek.com/gensim/models/word2vec.html
#https://phdstatsphys.wordpress.com/2018/12/27/word2vec-how-to-train-and-update-it/
#min_count: It represents the minimum frequency value of words to be present in the vocabulary. Its default value
#size: It represents how long you want the dimensionality of your vector to be for each word in the vocabulary. I
#window: The maximum distance between the current word and its neighboring words. If your neighboring word is gre
#epochs It represents the number of iterations/epochs over the dataset. Its default value is 5.
#workers defines how many cores you use for training. I set it to 1 to make sure the code is deterministically re
#seed sets the seed for random number generation. It's set to the constant SEED you defined in the first cell.
```

by using min_count=50 we reduced the Vocabulary Size: 63817 to 4846

:
```
#save the model for further reference

word2vec_model.save('/home/ga/Desktop/project MSc AI/data/word2vec.model')
```

:
```
# Load back
#word2vec_model = Word2Vec.load('/content/drive/My Drive/word2vec.model')
```

:
```
# Get the list of words in the vocabulary
vocab = list(word2vec_model.wv.key_to_index.keys())

# Print the vocabulary size and some example words
print(f"Vocabulary Size: {len(vocab)}")
print("Sample words in the vocabulary:", vocab[:100])
```

There are several ways to vectorize a Word2Vec model for document representation which include Averaging Word Vectors, TF-IDF Weighted Averaging where each word vector is weighted by its TF-IDF score, Concatenation of Min, Max, and Average Vectors for the words in a document which captures different aspects of the word distributions and Using Pre-trained Sentence Embeddings.

:
```
# vectorize using the min max and average
def vectorize_word2vec(line):
    words = []
    for word in line:  # line - iterable list of tokens
        try:
            w2v_idx = word2vec_indices[word]
        except KeyError:  # if we do not have a vector for this word in your w2v model, continue
            continue
        words.append(word2vec_vectors[w2v_idx])
    if words:
        words = np.asarray(words)
        min_vec = words.min(axis=0)
        max_vec = words.max(axis=0)
        avg_vec = words.mean(axis=0)
        return np.concatenate((min_vec, max_vec, avg_vec))

    else:
        return None
```

```
4]:  1 word2vec_matrix_vectors = pd.DataFrame(vectorized_docs)
     2

:    1 vectorized_docs = [vectorize_word2vec(doc) for doc in tokenized_text]
```

```
1 # save matix for further reference
2 word2vec_matrix_vectors.to_csv('/home/ga/Desktop/project MSc AI/data/word2vec_matrix_vectors.csv', index=False)
```

3.3 Text vectorization BERT

```
4 # Load pre-trained BERT model and tokenizer
5 tokenizer_BERT = BertTokenizer.from_pretrained('bert-base-uncased')
6 model_BERT = BertModel.from_pretrained('bert-base-uncased')
```

```
1  #basic Pre-cleaning for BERT
2
3  def text_clean_BERT(text):
4
5      #text = text.lower() # lowercase everything
6      x = text.encode('ascii', 'ignore').decode()  # remove unicode characters
7      x = re.sub(r'https*\S+', ' ', x) # remove links
8      x = re.sub(r'http*\S+', ' ', x)
9      x = re.sub(r'\s{2,}', ' ', x) #replaces multiple consecutive spaces with a single space
10
11     # cleaning up text
12     #x = re.sub(r'\'\w+', '', x) # removes words that start with an apostrophe (contractions or possessives).
13     #x = re.sub(r'\w*\d+\w*', '', x) #removes words containing digits
14     #x = re.sub(r'\s{2,}', ' ', x) #replaces multiple consecutive spaces with a single space
15     #x = re.sub(r'_', '', x) #remove the underscore character (.)
16     #x = re.sub(r'(.)', '', x) #remove the underscore character
17
18     # Consider keeping punctuation that might be relevant to context
19     # x = re.sub(r'\s[^\w\s]\s', '', x)  # Commenting out to keep punctuation surrounded by spaces
20
21     # Handle specific cases where spacing might be important
22     x = re.sub(r'\.([a-zA-Z])', r'. \1', x)
23     x = re.sub(r'\,([a-zA-Z])', r', \1', x)
24
25     return x.strip()
```

```
1 circulars_text_BERT = pdf_df4['Text'].apply(text_clean_BERT).tolist()
```

```
1  # BERT can work with text of maximum length of 512 tokens. This is why if the document is too long
2  #we have to split and encode chunks, then average
3  #Function to encode documents using BERT with handling long documents
4  def encode_documents(docs, max_length=512):
5      embeddings = []
6      for doc in docs:
7          inputs = tokenizer_BERT(doc, return_tensors='pt', max_length=max_length, truncation=True, padding=True)
8          with torch.no_grad():
9              outputs = model_BERT(**inputs)
10         doc_embedding = outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
11
12         # If document is too long, split and encode chunks, then average
13         #it does that by converting the doc in a tnsor format, truncating the document if it exceeds the maximum
14         #or padding the document if it is shorter than the maximm length
15         if len(inputs['input_ids'][0]) > max_length:
16             chunk_embeddings = []
17             for i in range(0, len(doc), max_length):
18                 chunk = doc[i:i+max_length]
19                 inputs = tokenizer_BERT(chunk, return_tensors='pt', max_length=max_length,
20                                         truncation=True, padding=True)
21                 # Passes the tokenized inputs through the BERT model without computing gradients (saves memory an
22                 #Computes the average embedding of the tokens in the last hidden state for the entire document an
23                 with torch.no_grad():
24                     outputs = model(**inputs)
25                 chunk_embeddings.append(outputs.last_hidden_state.mean(dim=1).squeeze().numpy())
26             doc_embedding = np.mean(chunk_embeddings, axis=0)
27
28         embeddings.append(doc_embedding)
29     return np.array(embeddings)
```

```
# Encode the documents using BERT
document_vectors_BERT = encode_documents(circulars_text_BERT)
```

```
document_vectors_BERT.shape
```

```
(3304, 768)
```

```
#save for further reference
BERT_matrix_vectors = pd.DataFrame(document_vectors_BERT)
BERT_matrix_vectors.to_csv('/home/ga/Desktop/project MSc AI/data/BERT_matrix_vectors.csv', index=False)
```

10 Check the right number of components and find the optimum n of clusters

File: *6 Check the right number of components and find the optimum n of clusters.ipynb*

```
#import the data
csv_file_pdf_df4 = '/home/ga/Desktop/project MSc AI/data/pdf_df4.csv'
tf_idf_df=pd.read_csv('/home/ga/Desktop/project MSc AI/data/tf_idf_df.csv', index_col='Unnamed: 0')
```

2. TF-IDF performing dimensionality reduction using TruncatedSVD

https://scikit-learn.org/stable/auto_examples/text/plot_document_clustering.html

```
# tf_idf_df is the DataFrame with TF-IDF values
X = tf_idf_df.values
```

```
 #Try different number of components
n_components_range = range(1, min(X.shape[0], X.shape[1]), 50)  # Adjust step size as needed
explained_variance_ratios = []
# Finding optimal number of features for clustering
for n_components in n_components_range:
    svd = TruncatedSVD(n_components=n_components)
    svd.fit(X)
    explained_variance = svd.explained_variance_ratio_.sum()
    explained_variance_ratios.append(explained_variance)
    print(f"Number of components = {n_components}, Explained Variance = {explained_variance}")
```

```
# Plot explained variance vs. number of components
plt.plot(n_components_range, explained_variance_ratios)
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance vs. Number of Components (Truncated SVD)')
plt.show()
```

It looks like the first few 1000 components explain 94% of the variance.After this adding more components does not signiffically improve the representation of the data.

```
svd = TruncatedSVD(n_components=1000)
X_SVD=svd.fit_transform(X)
```

3. Find the optimum n of clusters

```
# Use Elbow Method to find optimal number of clusters
# Try different ranges of clusters
visualizer = KElbowVisualizer(KMeans(max_iter=500, n_init=20, random_state=SEED), k=(2,40))
visualizer.fit(X_SVD)
visualizer.show()
```

```
#optimum n of clusters using the silhouette metric
visualizer = KElbowVisualizer(KMeans(max_iter=500, n_init=20, random_state=SEED), k=(2,40), metric='silhouette')
visualizer.fit(X_SVD)
visualizer.show()
```

```
#optimum n of clusters using the calinsky harabasz
visualizer = KElbowVisualizer(KMeans(max_iter=500, n_init=20, random_state=SEED), k=(2,40), metric='calinski_hara
visualizer.fit(X_SVD)
visualizer.show()
```

```
#Calculate Davies-Bouldin scores for different numbers of clusters
from sklearn.metrics import davies_bouldin_score
davies_bouldin_scores = []
for n in range(2, 41):
    kmeans = KMeans(n_clusters=n, max_iter=500, n_init=20, random_state=SEED)
    labels = kmeans.fit_predict(X_SVD)
    score = davies_bouldin_score(X_SVD, labels)
    davies_bouldin_scores.append(score)
    print(f"Number of clusters = {n}, Davies-Bouldin score = {score}")
```

```
#plot Davies-Bouldin scores vs. number of clusters
plt.plot(range(2, 41), davies_bouldin_scores)
plt.xlabel('Number of Clusters')
plt.ylabel('Davies-Bouldin Score')
plt.title('Davies-Bouldin Score vs. Number of Clusters')
plt.show()
```

```
#plot the silhouette plot for the selected 21 n of clusters
#https://stackabuse.com/k-means-elbow-method-and-silhouette-analysis-with-yellowbrick-and-scikit-learn/
from sklearn.metrics import silhouette_samples

model_3clust = KMeans(n_clusters = 21, max_iter=500, n_init=20, random_state=SEED)
model_3clust.fit(X_SVD)
labels = model_3clust.labels_
silhouette_vals = silhouette_samples(X_SVD, labels)

visualizer = SilhouetteVisualizer(model_3clust)
visualizer.fit(X_SVD)        # Fit the data to the visualizer
visualizer.show()        # Finalize and render the figure
```

4. TruncatedSVD for word2vec

```
#load data
word2vec_df=pd.read_csv('/home/ga/Desktop/project MSc AI/data/word2vec_matrix_vectors.csv')
```

```
#Try different number of components
n_components_range = range(1, min(X.shape[0], X.shape[1]), 50)  # Adjust step size as needed
explained_variance_ratios = []
# Finding optimal number of features for clustering
for n_components in n_components_range:
    svd = TruncatedSVD(n_components=n_components)
    svd.fit(X)
    explained_variance = svd.explained_variance_ratio_.sum()
    explained_variance_ratios.append(explained_variance)
    print(f"Number of components = {n_components}, Explained Variance = {explained_variance}")
```

```
# Plot explained variance vs. number of components
plt.plot(n_components_range, explained_variance_ratios)
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance vs. Number of Components (Truncated SVD)')
plt.show()
```

5. TruncatedSVD for BERT

```
#load data
BERT_df1=pd.read_csv('/home/ga/Desktop/project MSc AI/data/BERT_matrix_vectors.csv')
```

```
X = BERT_df1.values
```

```
#Try different number of components
n_components_range = range(1, min(X.shape[0], X.shape[1]), 50)  # Adjust step size as needed
explained_variance_ratios = []
# Finding optimal number of features for clustering
for n_components in n_components_range:
    svd = TruncatedSVD(n_components=n_components)
    svd.fit(X)
    explained_variance = svd.explained_variance_ratio_.sum()
    explained_variance_ratios.append(explained_variance)
    print(f"Number of components = {n_components}, Explained Variance = {explained_variance}")
```

```
# Plot explained variance vs. number of components
plt.plot(n_components_range, explained_variance_ratios)
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance vs. Number of Components (Truncated SVD)')
plt.show()
```

11 Clustering implementation, evaluation and visualization

file: *7. Implementing the clustering.ipynb*

2. Load the data and reduce dimensionality

```
# Load the data

base_address_data='/home/ga/Desktop/project MSc AI/data/'
tfidf_df = pd.read_csv(base_address_data+'tf_idf_df.csv', index_col='Unnamed: 0')
word2vec_df = pd.read_csv(base_address_data+'word2vec_matrix_vectors.csv')
bert_df = pd.read_csv(base_address_data+'BERT_matrix_vectors.csv')
```

```
# Reduce dimensionality to xyz components
svd = TruncatedSVD(n_components=750)
tfidf_df_r= svd.fit_transform(tfidf_df)
# Convert the NumPy array back to a Pandas DataFrame
tfidf_df_r = pd.DataFrame(tfidf_df_r, index=tfidf_df.index)
# We are adding back the index from the original word2vec_df DataFrame
```

```

# Reduce dimensionality to xyz components
svd = TruncatedSVD(n_components=135)
word2vec_df_r= svd.fit_transform(word2vec_df)
# Convert the NumPy array back to a Pandas DataFrame
word2vec_df_r = pd.DataFrame(word2vec_df_r, index=word2vec_df.index)
# We are adding back the index from the original word2vec_df DataFrame
```

```
# Reduce dimensionality to xyz components
svd = TruncatedSVD(n_components=200)
bert_df_r= svd.fit_transform(bert_df)
# Convert the NumPy array back to a Pandas DataFrame
```

3. Set clustering functions

```
#set k
k=21
```

```
def kmeans_clustering(df, n_clusters=k):
  """
  Performs k-means clustering on the given dataframe.

  Args:
    df: The dataframe containing the vectorized data.
    n_clusters: The number of clusters to form.

  Returns:
    The KMeans model fitted to the data.
  """
    kmeans = KMeans(n_clusters=n_clusters, max_iter=500, n_init=20, init='k-means++',
                    random_state=SEED)#n_init=20,
    kmeans.fit(df)
    return kmeans
```

```python
def efcm_clustering(df, n_clusters=k, m=1.1):
    """
    Performs Enhanced Fuzzy C-Means clustering on the given dataframe.

    Args:
        df: The dataframe containing the vectorized data.
        n_clusters: The number of clusters to form.
        m: fuziness factor

    Returns:
        The FCM model fitted to the data.
    """
    scaler = StandardScaler()
    scaled_data = df.values


    # Use cmeans from skfuzzy

    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        scaled_data.T, n_clusters, m, error=0.005, maxiter=1000, init=None, seed=SEED
    )
    cluster_membership = np.argmax(u, axis=0)
    return cluster_membership
```

```python
#lstm_clustering

def build_autoencoder(input_shape, dense_units, dropout_rate=0.2):
    """
    Builds a simple autoencoder for tabular data using Dense layers.

    Args:
        input_shape: The shape of the input data.
        dense_units: A list containing the number of units in each Dense layer.
        dropout_rate: Dropout rate for regularization.

    Returns:
        The autoencoder model, input layer, and the encoded (bottleneck) layer output.
    """
    input_layer = Input(shape=input_shape)
```

```python
def lstm_clustering(df, n_clusters=3, learning_rate=0.001, epochs=50, batch_size=64,
                    dense_units=[128, 64, 32], pretrain_epochs=50, dropout_rate=0.2):
    # Scale the data
    scaler = StandardScaler()
    df_scaled = scaler.fit_transform(df)

    # 1. Pretrain Autoencoder:
    input_shape = (df_scaled.shape[1],)
    autoencoder, input_layer, encoded_output = build_autoencoder(input_shape, dense_units, dropout_rate)
    autoencoder.compile(optimizer=Adam(learning_rate=learning_rate), loss='mse')
    autoencoder.fit(df_scaled, df_scaled, epochs=pretrain_epochs, batch_size=batch_size, verbose=0)

    # 2. Extract Encoder:
    encoder = Model(inputs=input_layer, outputs=encoded_output)
    encoded_X = encoder.predict(df_scaled)

    # 3. Initialize Cluster Centers (using K-Means++ for better initialization):
    kmeans = KMeans(n_clusters=n_clusters, n_init=20, init='k-means++')
    y_pred = kmeans.fit_predict(encoded_X)

    # 4. Build Clustering Layer:
    clustering_layer = Dense(n_clusters, activation='softmax', name='clustering')(encoded_output)
    model = Model(inputs=input_layer, outputs=clustering_layer)

    def target_distribution(q):
        weight = q ** 2 / q.sum(0)
        return (weight.T / weight.sum(1)).T

    # 5. Train Clustering Model:
    model.compile(optimizer=Adam(learning_rate=learning_rate), loss='kld')
    y_pred_last = np.copy(y_pred)
    y_pred = tf.keras.utils.to_categorical(y_pred, n_clusters)

    for epoch in range(epochs):
        q = model.predict(df_scaled, verbose=0)  # Reduced verbosity
        p = target_distribution(q)
        model.fit(df_scaled, p, epochs=1, batch_size=batch_size, verbose=0)  # Reduced verbosity

        y_pred = q.argmax(1)
        delta_label = np.sum(y_pred != y_pred_last).astype(np.float32) / y_pred.shape[0]
        y_pred_last = np.copy(y_pred)
        if epoch > 0 and delta_label < 0.001:
            print(f'Converged at epoch {epoch}')
            break

    return y_pred
```

4. Run Clustering

```python
def run_clustering(df):
  """
  Runs all clustering methods on the given dataframe and evaluates their performance.

  Args:
    df: The dataframe containing the vectorized data.

  Returns:
    A dictionary containing the evaluation metrics for each clustering method.
  """
  results = {}

  # K-means clustering
  kmeans_model = kmeans_clustering(df)
  kmeans_predictions = kmeans_model.labels_
  results['K-means'] = kmeans_predictions #

  # Enhanced Fuzzy C-Means clustering
  efcm_predictions = efcm_clustering(df)
  #efcm_model = efcm_clustering(df)
 #efcm_predictions = efcm_model.predict(df.values)
  results['EFCM'] = efcm_predictions

  # LSTM clustering
  lstm_model = lstm_clustering(df)
  #lstm_predictions = np.argmax(lstm_model.predict(df.values.reshape(df.shape[0], 1, df.shape[1])), axis=1)
  results['LSTM'] = lstm_model #lstm_predictions

  return results
```

```python
# Run clustering and evaluate for each dataset
tfidf_results = run_clustering(tfidf_df_r)
word2vec_results = run_clustering(word2vec_df_r)
bert_results = run_clustering(bert_df_r)
```

```python
# put the results together
tfidf_results_df = pd.DataFrame(tfidf_results)
word2vec_results_df = pd.DataFrame(word2vec_results)
bert_results_df = pd.DataFrame(bert_results)

# Concatenate the results horizontally to avoid doubling the index length
combined_results = pd.concat([
    tfidf_results_df.add_suffix('_tfidf'),
    word2vec_results_df.add_suffix('_word2vec'),
    bert_results_df.add_suffix('_bert')
], axis=1) # Change axis to 1 for horizontal concatenation

# Now you should be able to add the clustering results without errors
combined_results['K-means_tfidf'] = tfidf_results['K-means']
combined_results['EFCM_tfidf'] = tfidf_results['EFCM']
combined_results['LSTM_tfidf'] = tfidf_results['LSTM']

# Display the combined dataframe
print(combined_results.head())
```

5. Calculate metrics

```python
# function to measure the quality of the clustering results

def calculate_metrics(df, predicted_labels):
  """
  Calculates clustering metrics.

  Args:
    df: The dataframe containing the vectorized data.
    predicted_labels: The predicted cluster labels.

  Returns:
    A dictionary containing the Calinski-Harabasz Index, Silhouette Score, and Entropy.
  """
  # Check if number of unique labels is greater than 1
  if len(np.unique(predicted_labels)) > 1:
    chi = calinski_harabasz_score(df, predicted_labels)
    silhouette = silhouette_score(df, predicted_labels)
    dbs=davies_bouldin_score(df, predicted_labels)

    # Calculate entropy - need laels for this
    #unique_labels, counts = np.unique(predicted_labels, return_counts=True)
    #probabilities = counts / counts.sum()
    #entropy = -np.sum(probabilities * np.log2(probabilities))

    return {'Calinski-Harabasz Index': chi, 'Silhouette Score': silhouette, 'Davies Bouldin Index': dbs} #, 'E
  else:
    return {'Calinski-Harabasz Index': None, 'Silhouette Score': None, 'Davies Bouldin Index': None} # Return
```

```python
# Calculate metrics for each clustering method and dataset
metrics = {}
for dataset_name, results in [('TF-IDF', tfidf_results), ('Word2Vec', word2vec_results), ('BERT', bert_results
  metrics[dataset_name] = {}
  for method_name, predictions in results.items():
    metrics[dataset_name][method_name] = calculate_metrics(
        tfidf_df if dataset_name == 'TF-IDF' else (word2vec_df if dataset_name == 'Word2Vec' else bert_df),
        predictions
    )
```

```python
#save the metrics as a data frame

metrics_df = pd.DataFrame.from_dict({(i,j): metrics[i][j]
                           for i in metrics.keys()
                           for j in metrics[i].keys()},
                       orient='index')
metrics_df
```

6 Plot the clusters using PCA

```python
# plot the clusters for each of the dataset and for each of the clustering method used using PCA

#import matplotlib.pyplot as plt
#from sklearn.decomposition import PCA

"""
  Plots the clusters using PCA for dimensionality reduction.

  Args:
    df: The dataframe containing the vectorized data.
    labels: The predicted cluster labels.
    title: The title of the plot.
"""

def plot_clusters(df, labels, title):
    # Reduce dimensionality to 2D using PCA
    pca = PCA(n_components=2)
    reduced_data = pca.fit_transform(df)

  # Plot the clusters
    fig, ax = plt.subplots(figsize=(8, 6))
    scatter = ax.scatter(reduced_data[:, 0], reduced_data[:, 1], c=labels, cmap='viridis', alpha=0.7)
    ax.set_title(title)
    ax.set_xlabel("PCA Component 1")
    ax.set_ylabel("PCA Component 2")

  # Create a legend with all clusters
    unique_labels = np.unique(labels)
    handles = [plt.Line2D([0], [0], marker='o', color='w', label=f"Cluster {label}",
                          markerfacecolor=plt.cm.viridis(label / unique_labels.max()), markersize=10)
               for label in unique_labels]

  # Position the legend outside to the right
    ax.legend(handles=handles, loc='center left', bbox_to_anchor=(1, 0.5))

 # Plot clusters for each dataset and clustering method
for dataset_name, results in [('TF-IDF', tfidf_results), ('Word2Vec', word2vec_results),
                              ('BERT', bert_results)]:
    for method_name, predictions in results.items():
      plot_clusters(
          tfidf_df if dataset_name == 'TF-IDF' else (word2vec_df_r if dataset_name == 'Word2Vec' else bert_df_r),
          predictions,
          f"{dataset_name} - {method_name} Clustering"
    )
```

7. Plot the clusters using t-SNE

```python
#plot the clusters for each of the dataset and for each of the clustering method used using t-SNE
#import matplotlib.pyplot as plt
#from sklearn.manifold import TSNE

def plot_clusters(df, labels, title):
  # Reduce dimensionality to 2D using TSNE
    tsne = TSNE(n_components=2,  random_state=SEED)
    reduced_data = tsne.fit_transform(df)

  # Plot the clusters
    fig, ax = plt.subplots(figsize=(8, 6))
    scatter = ax.scatter(reduced_data[:, 0], reduced_data[:, 1], c=labels, cmap='viridis', alpha=0.7)
    ax.set_title(title)
    ax.set_xlabel("t-SNE Component 1")
    ax.set_ylabel("t-SNE Component 2")

  # Create a legend with all clusters
    unique_labels = np.unique(labels)
    handles = [plt.Line2D([0], [0], marker='o', color='w', label=f"Cluster {label}",
                          markerfacecolor=plt.cm.viridis(label / unique_labels.max()), markersize=10)
               for label in unique_labels]

  # Position the legend outside to the right
    ax.legend(handles=handles, loc='center left', bbox_to_anchor=(1, 0.5))

# Plot clusters for each dataset and clustering method
for dataset_name, results in [('TF-IDF', tfidf_results), ('Word2Vec', word2vec_results), ('BERT', bert_results)]:
  for method_name, predictions in results.items():
    plot_clusters(
        tfidf_df if dataset_name == 'TF-IDF' else (word2vec_df_r if dataset_name == 'Word2Vec' else bert_df_r),
        predictions,
        f"{dataset_name} - {method_name} Clustering"
    )
```

8. Plot interactive clusters for Word2Vec with k-means with both the PCA and the t-SNA

```
# For interactive plots
import plotly.express as px
import plotly.graph_objects as go

pdf_df4 = pd.read_csv(base_address_data+'pdf_df4.csv')
word2vec_results_df=pd.DataFrame(word2vec_results)

tsne = TSNE(n_components=2,  random_state=SEED)
word2vec_tsne = tsne.fit_transform(word2vec_df)


# Create a DataFrame for plotting
plot_df = pd.DataFrame({
    'x': word2vec_tsne[:, 0],
    'y': word2vec_tsne[:, 1],
    'Cluster': word2vec_results_df['K-means'],
    'Document Name': pdf_df4['Document Name']
})

# Interactive Plot using Plotly for t-SNE results with hover data
fig_tsne = px.scatter(plot_df, x='x', y='y', color='Cluster', hover_data=['Document Name'],
                      labels={'x': 't-SNE Component 1', 'y': 't-SNE Component 2', 'color': 'Cluster'},
                      title='Cluster Assignments (t-SNE)')
fig_tsne.show()
```

```
#Plot interactive for PCA
pca = PCA(n_components=2,  random_state=SEED)
word2vec_pca = pca.fit_transform(word2vec_df)


# Create a DataFrame for plotting
plot_df = pd.DataFrame({
    'x': word2vec_pca[:, 0],
    'y': word2vec_pca[:, 1],
    'Cluster': word2vec_results_df['K-means'],
    'Document Name': pdf_df4['Document Name']
})

# Interactive Plot using Plotly for t-SNE results with hover data
fig_tsne = px.scatter(plot_df, x='x', y='y', color='Cluster', hover_data=['Document Name'],
                      labels={'x': 'PCA Component 1', 'y': 'PCA Component 2', 'color': 'Cluster'},
                      title='Cluster Assignments (PCA)')
fig_tsne.show()
```