```python
import pandas as pd
import numpy as np
from tqdm import tqdm
import nltk
import importlib
import utils.preprocessing as preprocessing
import clustering.wiki_graph as wiki_graph
```

Data Preprocessing

In this first part we will be Preprocessing text data to prepare them for clustering and classification. This will include the following steps:

- Noise Removal
- Normalization
- Tekenization & Segmentation

Data Loading

```python
df = pd.read_pickle("data/dataset_business_technology_cybersecurity.pickle")
df = pd.DataFrame(df)
df.sample(5)
```

	title	content	topic
84	Partnership	<p>A partnership is an arrangement wher...	business
219	Cable car (railway)	<p>A cable car (usually known as a c...	technology
90	Benefit shortfall	<p>When the actual benefits of a venture are l...	business
261	Computer virus	<p class="mw-empty-elt">\n\n \n\n A co...	cybersecurity
101	Trade name	<p>A trade name, trading name, o...	business

```python
# explore the data format in a txt file
df.to_csv("data/backup_preprocess/content.txt")
```

Noise Removal

Noise removal can be defined as text-specific normalization. As we are dealing with html row data, our data preprocessing pipeline will include striping away all HTML markup with the help of the BeautifulSoup library. We will also be replacing contractions with their expansions.

```python
importlib.reload(preprocessing)
df["content"] = preprocessing.remove_noise_from_df(df["content"])
# backup saving
df.to_csv("data/backup_preprocess/content_without_noise.txt")
```

Normalization

Normalization refers to a series of tasks that put all text on a level of playing field: converting all text to the same case(upper or lower), removing special characters(punctuation) and numbers, stemming, lemmatization, ... Normalization puts all words on equal footing and alows processing to proceed uniformly.

```python
importlib.reload(preprocessing)
df["content"] = preprocessing.normalize_df(df["content"])
# backup save
df.to_csv("data/backup_preprocess/content_normalized.txt")
```

Tockenization

```python
importlib.reload(preprocessing)
df["content"] = df["content"].progress_apply(nltk.word_tokenize)
df.to_csv("data/backup_preprocess/content_tokenized.txt")
df.head(5)
```

Part1: Clustering

```python
df = pd.read_csv('data/backup_preprocess/content_tokenized.txt')
df.head(5)
```

	Unnamed: 0	title	content	topic
0	0	Accounting	['account', 'account', 'measur', 'process', 'c...	business
1	1	Commerce	['commerc', 'exchang', 'good', 'servic', 'espe...	business
2	2	Finance	['financ', 'term', 'matter', 'regard', 'manag'...	business
3	3	Industrial relations	['industri', 'relat', 'employ', 'relat', 'mult...	business
4	4	Management	['manag', 'manag', 'administr', 'organ', 'whet...	business

A. Graph Clustering

1. Building a graph

```python
importlib.reload(wiki_graph)
wiki_pages = df.to_dict(orient="records")
graph = wiki_graph.WikiGraph()
graph.build_graph(wiki_pages, constraint=20)
```

2. Time complexity of building the graph

Let N be the number of wikipedia pages in the dataset.

If we don't consider preprocessing data as part of building the graph, we need to evaluate the time complexity of the following tasks:

- Create all the nodes of the graph: $O(N)$
- Connect all pairs of nodes that share at least n tokens: $O(kN^2)$

Given two nodes $(n1, n2)$ and their respective contents $(c1, c2)$ of length $(l1, l2)$, $n1$ and $n2$ share at least n tokens if the length of the intersection of the sets (unique words) of their contents $(s1, s2)$ is greater or equal to n, that is $len(s1 \cap s2) >= n$.

The time complexity of finding the intersection of two sets $(s1, s2)$ of respective length $(l1, l2)$ is $O(l1 + l2)$. Let $k = max(l1, l2)$, the time complexity of creating the edges of the graph is therefore $O(kN^2)$.

That is said, the overall time complexity of building a graph is $O(kN^2 + N) = O(kN^2)$.

3. Find the connected components of the graph

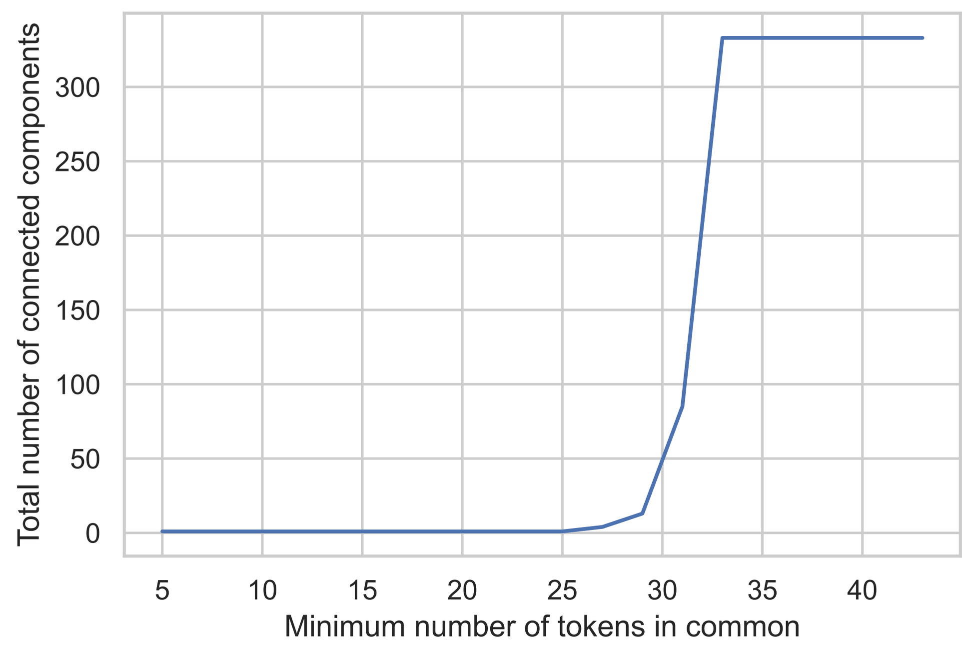
In graph theory, a connected component is a maximal connected subgrapgh of an undirected graph. each vertex belongs to exactly one connected cmponent, as does each edge. A graph is commected if and only if it has exactly one connected component.

In the following, we will be experimenting with different values of n: the min number of tokens in common.

```python
importlib.reload(wiki_graph)
n_tokens = list(range(5, 45, 2))
nb_clusters = {}
clusters = {}
for n in tqdm(n_tokens):
    graph = wiki_graph.WikiGraph()
    graph.build_graph(wiki_pages, constraint=n)
    clusters[n] = graph.get_wiki_clusters()
    nb_clusters[n] = len(clusters[n])
```

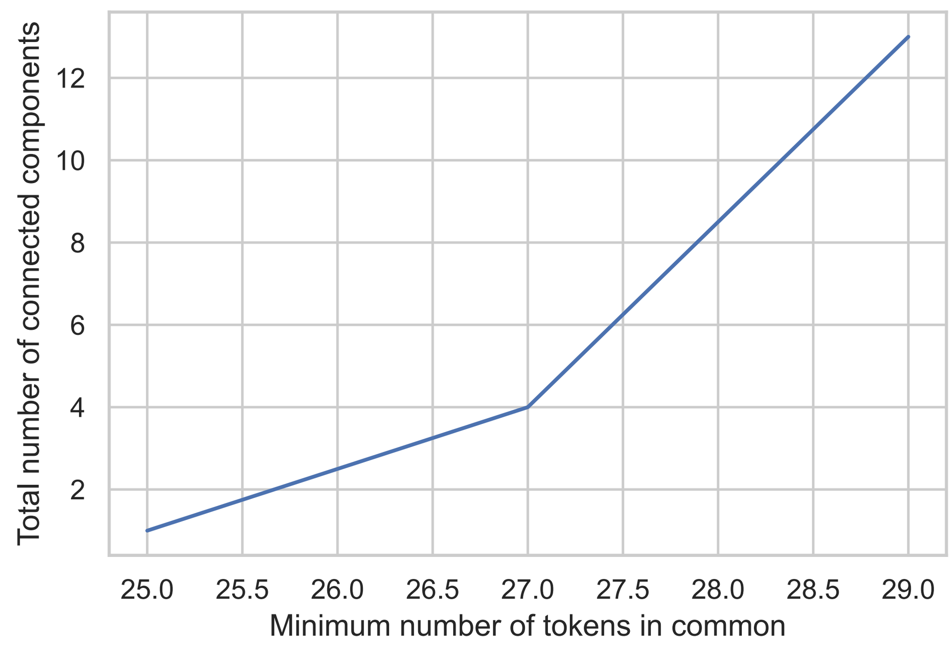
```python
np.save("data/backup_preprocess/nb_clusters.npy", nb_clusters)
nb_clusters = np.load('data/backup_preprocess/nb_clusters.npy',allow_pickle='TRUE').item()
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="whitegrid")
plt.plot(nb_clusters.keys(), nb_clusters.values())
plt.ylabel('Total number of connected components')
plt.xlabel('Minimum number of tokens in common')
plt.savefig('data/images/n_cl_vs_n_tokens.png')
```



As we can see the optimal nb of clusters is between 25 and 30, let's check these values.

```python
interesting_nb_clusters = {k:v for k, v in nb_clusters.items() if 25<=k<=30}
plt.plot(interesting_nb_clusters.keys(), interesting_nb_clusters.values())
plt.ylabel('Total number of connected components')
plt.xlabel('Minimum number of tokens in common')
plt.savefig('data/images/n_cl_vs_n_tokens_int.png')
```



Therefore, for all the following we will be using n=27 for the min number of tokens in common as it gives us 4 clusters which is close to our goal of 3 clusters.

```python
graph = wiki_graph.WikiGraph()
graph.build_graph(wiki_pages, constraint=27)
clusters = graph.get_wiki_clusters()
```

4. Complexity of finding all components:

- ***Algorithm***: We will be using an iterative version (with a stack) of dfs (depth-first search) to find the connected components.
- ***Time Complexity***: we will go through all the nodes of the graph with dfs, which gives $O(N)$ time Complexity.
- ***Space Complexity***: we need $O(N)$ extra space for dfs. ### 5. Title definition We can determine a title as the most common topic of the cluster. ### 6. Let's explore an example In the following we will define title for the 4th cluster (connected component) in the graph. this cluster contains only one wiki page and tehrefore it's topic is cybersecurity.

```python
print("Cluster nb 4 has title:", clusters[3].get_title())
print("This cluster contains", len(clusters[3].wiki_nodes), "wiki page")
print("With the following frequencies", clusters[3].get_topics_count())
```

```
Cluster nb 4 has title: cybersecurity
This cluster contains 1 wiki pages
With the following frequencies {'business': 0, 'cybersecurity': 1, 'technology': 0}
```

B. End to End Pipeline

1. Efficient main

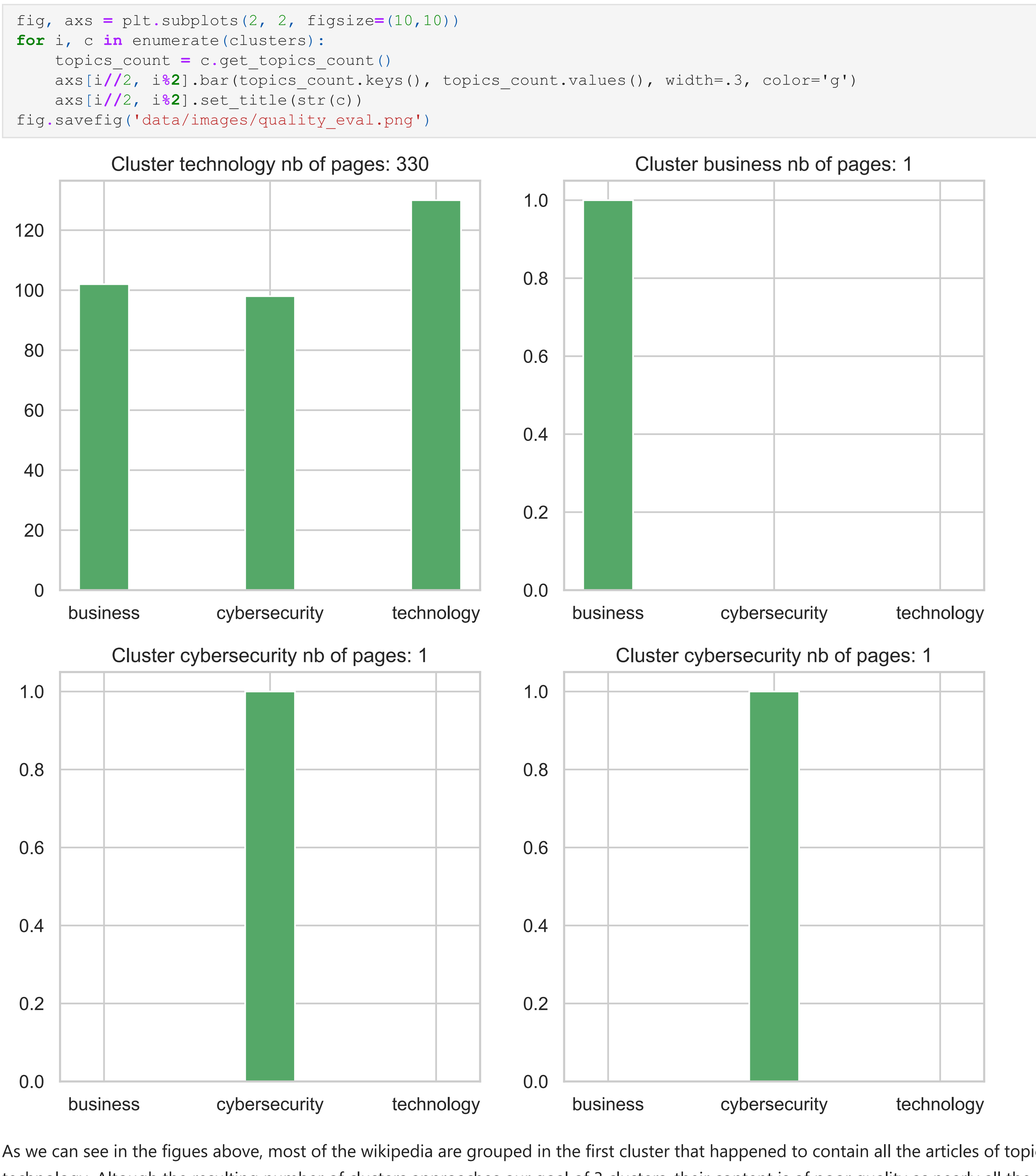
An efficient main to build the Pipeline can be find in the file `main.py`

2. Pipeline Running

Please check the `README.md` file for running the Pipeline in experiment mode or backup mode.

3. Quality evaluation

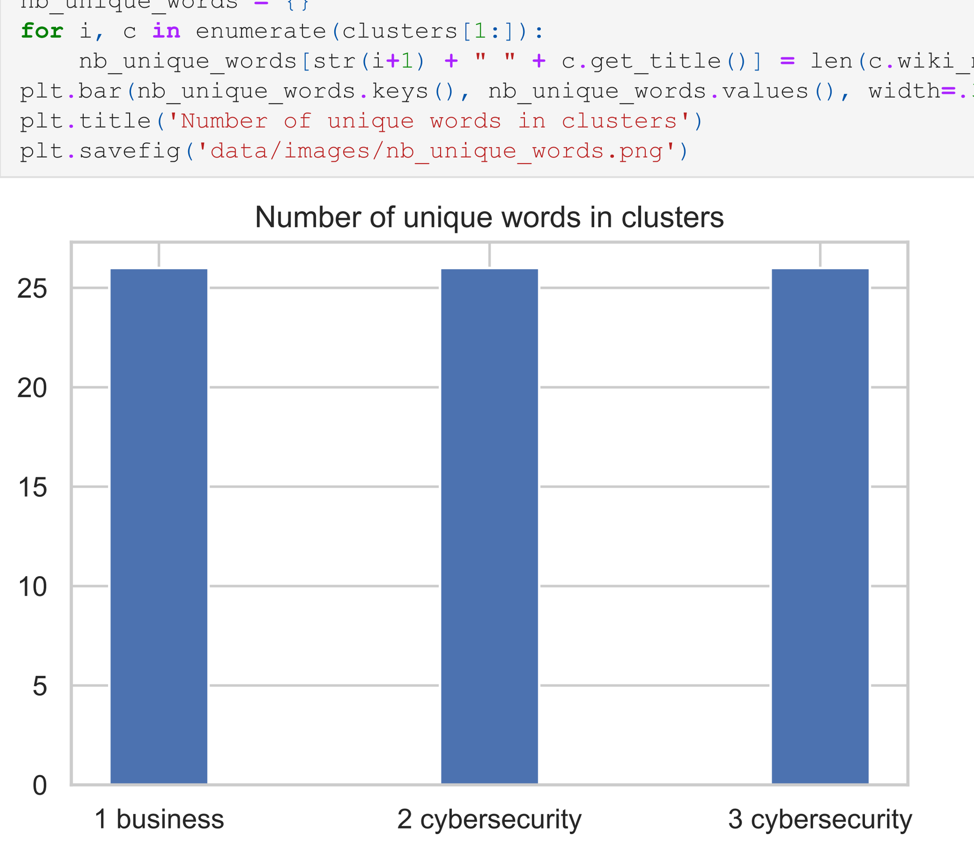
```python
fig, axs = plt.subplots(2, 2, figsize=(10,10))
for i, c in enumerate(clusters):
    topics_count = c.get_topics_count()
    axs[i//2, i%2].bar(topics_count.keys(), topics_count.values(), width=.3, color='g')
    axs[i//2, i%2].set_title(str(c))
fig.savefig('data/images/quality_eval.png')
```



As we can see in the figues above, most of the wikipedia are grouped in the first cluster that happened to contain all the articles of topic technology. Altough the resulting number of clusters approaches our goal of 3 clusters, their content is of poor quality as nearly all the wikipedia pages are grouped in the first cluster. The remaining clusters contain only one article which may contain a number of unique words smaller then the min number of tokens in common.

Let's display the number of unique words in theses clusters.

```python
nb_unique_words = {}
for i, c in enumerate(clusters[1:]):
    nb_unique_words[str(i+1) + " " + c.get_title()] = len(c.wiki_nodes[0].wiki_page.content)
plt.bar(nb_unique_words.keys(), nb_unique_words.values(), width=.3, color='b')
plt.title('Number of unique words in clusters')
plt.savefig('data/images/nb_unique_words.png')
```



Indeed the number of unique words is $26 < 27$. If we choose $n-tokens < 26$ we get one connected component as all the nodes in the graph are connected. The constraint on the number of token sin common is therefore biased, we can improve it bu using a ratio $\frac{n-tokens-in-common}{n-total-tokens}$ for example and setting the constraint to be a percentage. (i.e two nodes are connected if 40% of their content is similar)

5. Repeatability

In order to ensure the quality of the code, a set of unit test has been defined.

Please check `README.md` file to be able to reproduce the results with a python script.