

# DS5110 Final Project

Team D.A.D.S. - Tyler Beaulieu, Tim Paylor

December 9, 2025

## Abstract

We will be working with the Maine Port Authority to analyse their current data storage and share insights on the types of files they have and how they are organised. Our goal is to use our analysis to make their data more accessible and searchable.

Drive and extract metadata. We retrieve the file ID, filename, file type, file size, file path, and parent folder name. This was then stored in a Pandas DataFrame.

We determined that we were working with 8672 files inside 653 folders. Looking at file types, we found that the files were 78% documents and 22% images (as seen in Figure 1).

## 1 Introduction

The [Maine Port Authority](#) works in coordination with the Maine Department of Transportation in acquiring, financing, constructing, and operating any marine port terminal facility within the state of Maine. [Legislature \[2025\]](#) They operate three major ports in Portland, Searsport, and Eastport. [Authority \[2025\]](#)

Important documentation for the Port Authority is stored on a SharePoint drive. This drive has become increasingly disorganised over the years. The client is looking for assistance in making their data more organised and accessible.

Our first goal is to analyse the data as it exists today to help formulate a path forward. We'll use text mining and clustering techniques to find similar types of documents. Our secondary goal was to build a standard operating procedure to better name files moving forward.

## 2 Data Processing

### 2.1 The Dataset

Instead of working with the contents of their full SharePoint drive, we were given a smaller sample via a Google Drive folder. We will start by analyzing the metadata from these files.

We used the Google Drive API for Python to pull data about the files from Google Drive. [Google \[2021\]](#) We used it to recursively go through folders in the

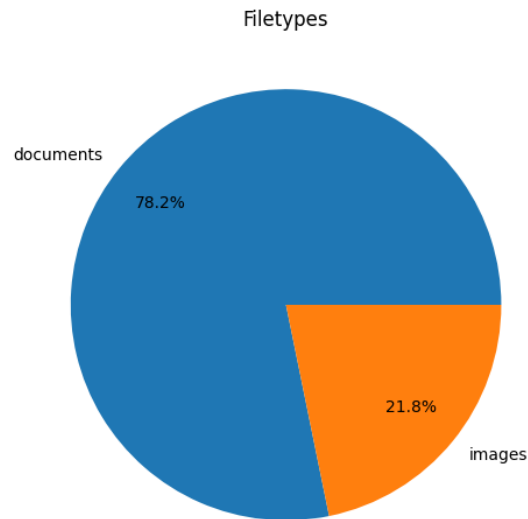


Figure 1: File Types  
Proportion of document and image file types.

The most popular document types were Word documents, PDF files, and Excel worksheets (as seen in Figure 2). Word documents made up over half of the non-image files in the drive.

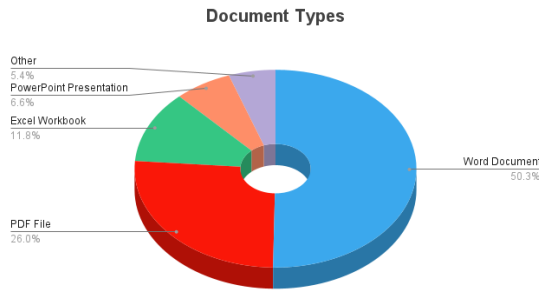


Figure 2: Document Types  
Proportion of document types.

## 2.2 Duplicates

Now that we have our metadata, we can check for duplicate files in the dataset. We made a copy of our DataFrame with just the filename, file size, and file type columns. Then we eliminated duplicate rows. From this, we found that there were 3202 unique files of 8672 total files. When we looked at folders, we found 257 unique folder names of 653 total folders.

Next, we checked for duplicate folders. We created a DataFrame of folder names, summed up the size of all files inside, and counted the number of documents and images present. Then we eliminated duplicate rows. From this, we discover 89 folders that have a duplicate in the system. Most of them appear to be from the CruiseMaine directories.

**Deliverable:** From this analysis, we will be able to provide the client with a list of duplicate files and folders.

## 2.3 Data Preparation

Using our new DataFrame of unique files, we have a directory upon which we can begin our analysis using machine learning. Metadata analysis can be found in the GitHub repository under the filename `metadata_workbook.ipynb`. [Beaulieu and Paylor \[2025\]](#)

# 3 Clustering

## 3.1 Introduction to Clustering

Our goal is to group documents by finding common words between them. This starts with text mining. Files are opened, and the text of each document is simplified to a recipe of words. In this form, the relative proportions of words are like the proportions of

ingredients in baked goods (as seen in Figure 3). More of one word and less of another word could help point us towards the type of document being analysed.



Figure 3: Baked Goods Example  
The proportion of ingredients leads to a different category. [Rebello \[2023\]](#)

These similarities allow us to categorise and sort the documents without having to read them. We can see this graphically in Figure 1 where the three ingredients, flour, sugar, and butter, are used to find clusters of baked goods that fall under a similar category. For the Maine Port Authority data, the same concept is used on hundreds of words in documents instead of just 3 ingredients in a recipe.

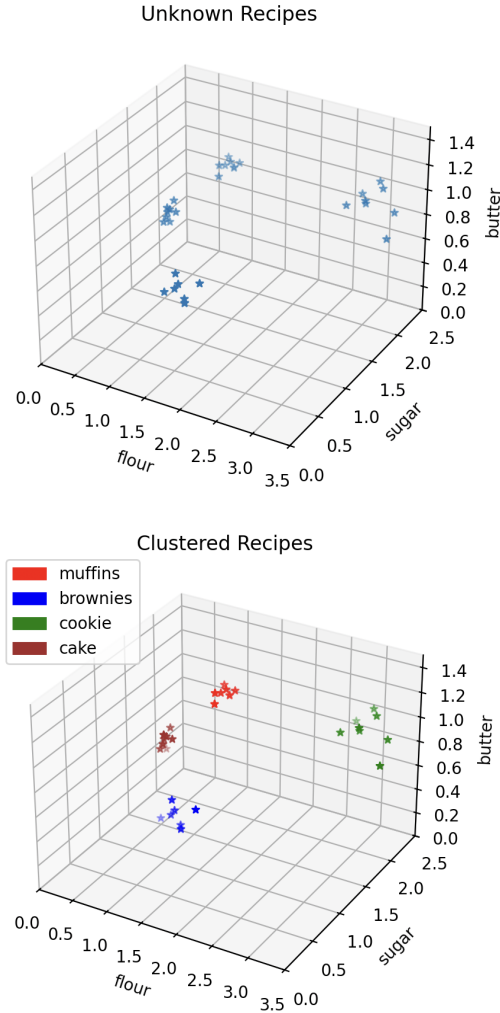


Figure 4: Clustering Example  
The algorithm finds the clusters, then the human identifies the category.

### 3.2 Textual Data Cleaning Methods

Accessing, cleaning, and analysing the data from the Google Drive required several steps. Full descriptions are in the appendix, but the essential steps are summarised here. As mentioned, we created a Pandas DataFrame of all unique files and our textual analysis focused on Word files.

We used an input/output data structure in Python that allowed us to open the documents as virtual files and pull down the entire text from each of them. Because we were accessing hundreds of files on a Google Drive in rapid succession, Google identified the activity as a non-human user and blocked our attempts, so we had to insert 5-10 second lags in

our code between each file query, and we saved our progress in 50-file blocks.

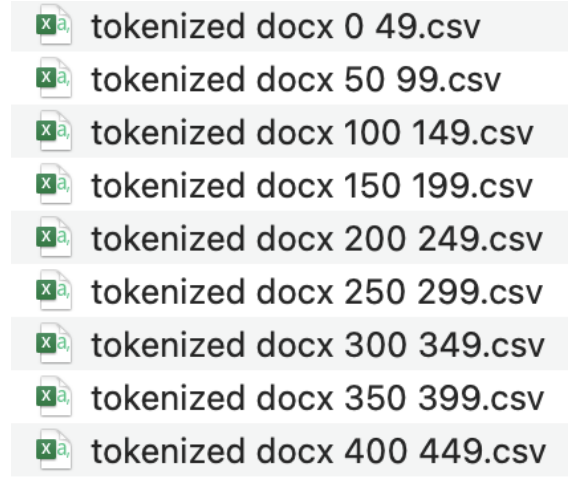


Figure 5: Sampling of Documents  
The CSVs made to put together our sample corpus.

The raw text of each file needed to be cleaned and processed in several ways. We removed individual words that are not specific to the meaning of the text, such as prepositions and helping verbs (“this”, “that”, “these”, “am”, “is”, “are”, “were”). We then reduced words to their meaningful base using a Python language library called SpaCy. [ExplosionAI \[2025\]](#) For example, in a document containing the words “industrial”, “industry”, and “industrious”, all three would be reduced to the common base “industry” and be counted as three occurrences of that word. In that way, the prevalence of that word as an ingredient would more accurately characterise the document in our quantitative analysis.

We combined all the documents into a single Pandas DataFrame, called a “corpus” of files, and produced several statistics based on the individual words, the files, and the corpus. A measurement of the frequency of a word within the entire corpus is called an Inverse Document Frequency (IDF). It is called “inverse” because it is expressed as the rarity, not the prevalence, of the word. All occurrences of a word in the corpus are assigned the same IDF. A measurement of the frequency of a word within an individual file is called the Normalized Frequency (NF). Each word in a document as an NF specific to the document.

Conceptually, IDF is the rarity of an ingredient across all recipes in the cookbook, while the NF is the prevalence of an ingredient in a single recipe. Each document is then expressed as a list of words, where

What we discovered from clustering the vectors was that there were, unfortunately, even more duplicate files than we originally counted. Our list of 400+ unique Word documents in .docx format turned out to be only 200 unique documents. In many cases, there were copies in 4 locations with small differences in size of 5 or 10 KB. The clustering algorithm was therefore producing many 4-document clusters (made up of identical files), and one or two clusters of 50 or 100 documents (made up of very different files). This did not meet the goal of labelling and organising files.

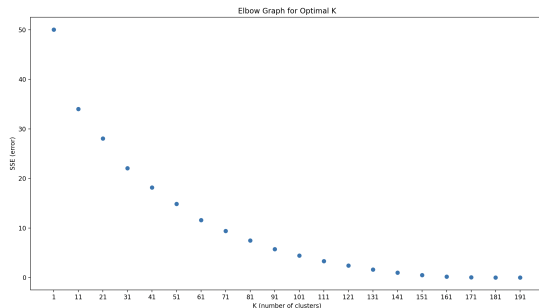


Figure 6: Initial Elbow Graph

The elbow graph indicated 160 clusters for the full docx corpus, but with less than 500 documents, that doesn't achieve the goal of organising the files.

[illegible]

Figure 7: Duplicate Files

Our analysis found more duplicates with small differences in size of 5 or 10 KB.

We eliminated these duplicates and subjected the data to the clustering algorithm again. On the new corpus of 200 documents, we had another disappointing result. The clustering approach did not find enough similarities between most documents to cluster them, and we had many clusters of 1 or 2 documents, along with a single cluster of 127 documents. The best conclusion we can draw from this is that the files are not suited to be studied as data points in a textual analysis, most likely due to the small size of the corpus, relative to real-world use cases with millions of documents.

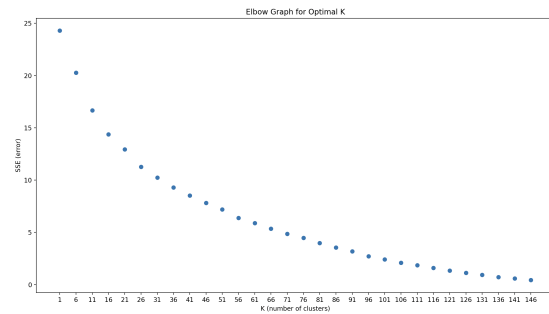


Figure 8: Second Elbow Graph

The elbow graph again indicated an unacceptably high number of clusters, over 100 clusters on 200 data points, was required to reduce error.

## 4 Conclusions

## 4.1 Deliverables and Future Work

For the client we are able to give them Excel files that contain lists of duplicate files and folders.

We would continue our clustering work on other document types (PDFs, Excel documents, PowerPoint). This would provide us with the necessary information to develop a Standard Operating Procedure for future file naming and storage.

We could also build a script to allow us to reorganise and rename their files. We want to build it with human validation to avoid major errors in categorisation.

## References

- Maine Port Authority. Maine port authority, 2025. URL <https://www.maineports.com>.
- Tyler Beaulieu and Tim Paylor. Github - baconarray/ds5110\_final: Final project for ds5110, fall 2025, roux institute., 11 2025. URL [https://github.com/baconarray/DS5110\\_final.git](https://github.com/baconarray/DS5110_final.git).
- ExplosionAI. Doc · spacy api documentation, 2025. URL <https://spacy.io/api/doc>.
- Google. Google api client, 10 2021. URL <https://github.com/googleapis/google-api-python-client>.
- Maine Legislature. Title 23: Transportation, 2025. URL <https://legislature.maine.gov/statutes/23/title23ch0sec0.html>.
- Sanchia Rebello. Baking ratios, 12 2023. URL <https://thelittleshine.com/baking-ratios/>.

## 5 Appendix - Textual Analysis Functions

### 5.1 extension\_sample.py

**Input:**

‘unique files.csv’ (all Google IDs and metadata, all file extensions)

**Output:**

‘docx [i] [i+50].csv’ (50-file long lists of Google ID and metadata for docx extensions)

### 5.2 sample\_corpus.py

**Input:**

‘docx [i] [i+50].csv’ (list of CSVs containing 50-row file ids and metadata)

**Output:**

‘tokenised docx [i] [i+50].csv’ (tokenised dfs of contents of 50-files each)

**Methods:**

docx\_reader(): reads file from API into a memory object

word\_cleaner(): cleans and tokenises strings as lemmas

tokenizer(): builds dictionary as {google id: tokenised document as df} pairs

token\_saver(): concatenates the dictionary to a DataFrame and exports to CSV

### 5.3 corpus\_builder.py

**Input:**

‘tokenised docx [i] [i+50].csv’ (list of tokenised dfs)

**Output:**

‘tokenised docx corpus.csv’ (concatenates files into a single corpus, ~160,000 rows long)

### 5.4 corpus\_vectorizer.py

**Input:**

‘tokenised docx corpus.csv’

**Output:**

‘vectorised docx corpus.csv’ (400+ docx files in Feature Vector Form with 9,000+ token parameters)

### 5.5 clustering.py

**Input:**

‘vectorised docx corpus.csv’ (vectorised documents)

**Output:**

‘clustered docx vectors.csv’ (vectorised documents with a cluster index assigned to each Google ID)

**Methods:**

elbow\_graph(): find optimal number of clusters for scikitlearn K-Means algorithm

cluster\_indexer(): assign cluster numbers to documents