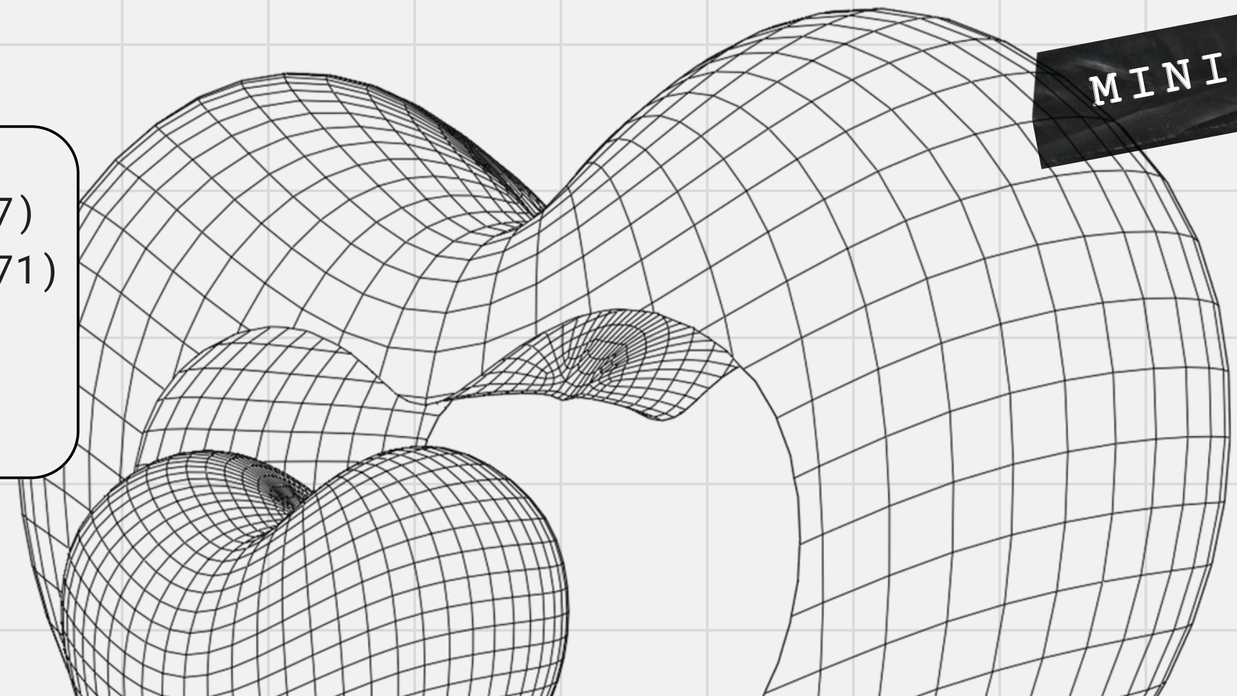
WINTER 2030

BORCELLE

RECOMMENDATION ALGORITHM AND CONTENT-BASED FILTERING

MINI PROJECT 1

- Lu Phone Maw (6511157)
- Wai Yan Paing (6511171)
- Noel Paing Oak Soe (6530183)



BORCELLE

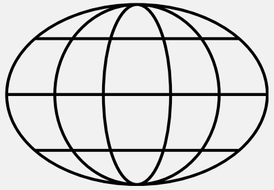
0012 7482901 2744103 0592346 8774510 7255

INTRODUCTION

This project builds a recommendation system that profiles users based on their reading history to suggest books using similarity measures.

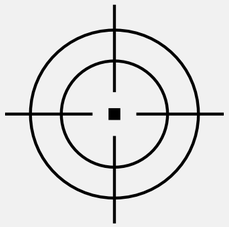
DATASETS

- bookData.csv
- UserData.csv
- UserHistoricalView.csv
- TestUserAnswers.csv



OBJECTIVE

Create user profiles and recommend books using a similarity measure.



METHODS USED

- Jaccard Similarity (Part 1)
- Cosine Similarity (Part 2)

1. Data Preprocessing

2. Building User Profiles

3. Finding Similarity

4. Display the first 5 books to each user

PART 1 OVERVIEW

DATA PREPROCESSING (PART 1)

```
import pandas as pd
import re
import numpy as np
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
```

1. Load bookData.csv into a DataFrame.
2. Fill in missing values and combine text fields.
3. Clean and lemmatize text.
4. Create a list of unique words.
5. Generate a binary term matrix.
6. Convert matrix to DataFrame and add 'isbn'.

ISBN	10	103	12	14th	150	16	1896	1920s	1925	...	york	yorker	you	young	younger	your	zellweger	zilpah	zoo	zurer
440234743	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
971880107	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
345417623	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
446310786	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
671027360	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1

```
(114, 2315)
```

BUILDING USER PROFILES (PART 1)

1. Load UserHistoricalView.csv into a DataFrame.
2. Aggregate ISBNs for each user and retrieve corresponding rows from term_matrix_df.
3. Sum the rows for each user and convert values to binary (1/0).
4. Create and reorder a DataFrame with user profiles.
5. Save the user profile DataFrame to a CSV file.

userid	10	103	12	14th	150	16	1896	1920s	1925	...	york	yorker	you	young	younger	your	zellweger	zilpah	zoo	zurer
11676	0	0	0	1	1	0	0	0	0	...	1	0	0	0	0	0	0	1	0	0
16795	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
22625	0	1	1	0	0	0	0	0	0	...	1	0	0	1	0	0	0	0	0	0
35859	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	1	0	0
95359	0	1	1	0	0	0	0	1	1	...	0	1	0	1	0	0	0	0	0	0
104636	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
110912	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	1	0	0	0
204864	0	0	0	0	0	0	0	0	0	...	1	0	0	1	0	0	1	1	1	0
271448	0	0	1	0	0	0	0	0	0	...	1	0	0	1	0	1	0	0	0	0

FINDING SIMILARITY (PART 1)

```python
def jaccard_similarity(v1, v2):
    # Initialize intersection and union counts
    intersection_count = 0
    union_count = 0

    # Iterate over both vectors simultaneously
    for i in range(len(v1)):
        # Convert to boolean to ensure exact comparison as done by np.logical_and and np.logical_or
        v1_bool = bool(v1[i])
        v2_bool = bool(v2[i])

        # Check for intersection (both are True/1)
        if v1_bool and v2_bool:
            intersection_count += 1

        # Check for union (at least one is True/1)
        if v1_bool or v2_bool:
            union_count += 1

    # Calculate Jaccard similarity
    if union_count == 0:
        return 0.0  # Handle edge case where both vectors are all zeros
    similarity = intersection_count / union_count
    return similarity
```

1. Define a function to calculate Jaccard similarity between two vectors.
2. Extract user profiles and item vectors from DataFrames.
3. Initialize a similarity matrix to compare users and items.
4. For each user, compute similarity scores for unread books and update the matrix.
5. Create a DataFrame for the similarity matrix, including user and item IDs.
6. Save the similarity matrix DataFrame to a CSV file.

userid	440234743	971880107	345417623	446310786	671027360	60976845	044021145X	60938455	446672211	...	375727345	312924585	684872153	316601950	439139597	439064864
11676	0.038664	0.024648	0.000000	0.038062	0.030822	0.021201	0.046552	0.000000	0.000000	...	0.021314	0.046312	0.035524	0.028021	0.026502	0.000000
16795	0.053398	0.050505	0.044444	0.097087	0.045872	0.051546	0.093897	0.030151	0.053097	...	0.063492	0.048889	0.077320	0.070352	0.040201	0.063725
22625	0.043360	0.041551	0.044041	0.045093	0.036649	0.030471	0.063660	0.021918	0.041026	...	0.033613	0.057592	0.047222	0.000000	0.030220	0.000000
35859	0.062731	0.048872	0.051546	0.064516	0.038194	0.045627	0.000000	0.029851	0.047458	...	0.042146	0.069686	0.072797	0.044118	0.037313	0.066667
95359	0.050595	0.036145	0.050992	0.000000	0.028249	0.036585	0.063401	0.020958	0.047619	...	0.040123	0.059829	0.055046	0.000000	0.033133	0.044379
104636	0.048000	0.037838	0.045802	0.041451	0.035897	0.027027	0.000000	0.021448	0.000000	...	0.027248	0.056410	0.040541	0.037333	0.040761	0.065041
110912	0.050992	0.037249	0.045699	0.058496	0.046575	0.040698	0.054496	0.031609	0.045333	...	0.028986	0.048387	0.052174	0.036723	0.000000	0.000000
204864	0.052632	0.037135	0.034653	0.048718	0.035264	0.029255	0.061224	0.021053	0.034398	...	0.035040	0.071611	0.065217	0.028571	0.034483	0.049869
271448	0.047486	0.045714	0.036939	0.032258	0.032086	0.028409	0.077135	0.019663	0.000000	...	0.040580	0.045093	0.048571	0.039216	0.042857	0.062323

DISPLAY THE FIRST 5 BOOKS TO EACH USER (PART 1)

1. Retrieve top 5 book recommendations for each user from the similarity DataFrame.
2. Fetch book titles from the original DataFrame based on ISBN.
3. Collect user, book, title, and similarity value into a list.
4. Create a DataFrame from the recommendations list.
5. Save the recommendations DataFrame to a CSV file.

User ID	Book's ISBN	Book's Title	Similarity Value
11676	590353403	Harry Potter and the Sorcerer s Stone (Book 1)	0.114545
11676	345337662	Interview with the Vampire	0.064784
11676	439136369	Harry Potter and the Prisoner of Azkaban (Book 3)	0.061750
11676	440211727	A Time to Kill	0.056604
11676	439064872	Harry Potter and the Chamber of Secrets (Book 2)	0.054386
16795	446310786	To Kill a Mockingbird	0.097087
16795	014028009X	Bridget Jones s Diary	0.094787
16795	345337662	Interview with the Vampire	0.094262
16795	044021145X	The Firm	0.093897
16795	068484477X	STONES FROM THE RIVER	0.090452
22625	590353403	Harry Potter and the Sorcerer s Stone (Book 1)	0.183140
22625	439136369	Harry Potter and the Prisoner of Azkaban (Book 3)	0.086842
22625	439064872	Harry Potter and the Chamber of Secrets (Book 2)	0.085165
22625	345337662	Interview with the Vampire	0.084788
22625	439139600	Harry Potter and the Goblet of Fire (Book 4)	0.079787
35859	590353403	Harry Potter and the Sorcerer s Stone (Book 1)	0.255061
35859	043935806X	Harry Potter and the Order of the Phoenix (Boo...	0.101695
35859	345337662	Interview with the Vampire	0.100977
35859	439136369	Harry Potter and the Prisoner of Azkaban (Book 3)	0.093426
35859	440224764	The Partner	0.090226

STEP 1

Data Preprocessing

STEP 2

Building User Profiles

STEP 3

Finding Similarity

STEP 4

Display the first 10 books to each user

STEP 5

Evaluation metrics

PART 2 OVERVIEW

GROUP 7

27/8

DATA PREPROCESSING (PART 2)

```
import pandas as pd
import re
import numpy as np
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
```

1. Load bookData.csv into a DataFrame.
2. Combine book synopses with publisher information.
3. Clean and lemmatize text for standardization.
4. Build a vocabulary of unique words.
5. Create a word index mapping each word to a unique ID.
6. Generate a term frequency matrix for the documents.

ISBN	10	103	12	14th	150	16	1896	1920s	1925	...	york	yorker	you	young	younger	your	zellweger	zilpah	zoo	zurer
440234743	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
971880107	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
345417623	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
446310786	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
671027360	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
439064864	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
43935806X	0	0	1	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
440220602	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0

```
(114, 2243)
```

DATA PREPROCESSING (PART 2)

1. Initialize a term frequency (TF) matrix with zeros.
2. Calculate normalized term frequency for each document.
3. Compute document frequency for each word.
4. Calculate inverse document frequency (IDF) for all words.
5. Generate the TF-IDF matrix by multiplying TF with IDF.

ISBN	10	103	12	14th	150	16	1896	1920s	1925	...	york	yorker	you	young	younger	your	zellweger	zilpah	zoo	zurer
440234743	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.00000
971880107	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.080334	0.0	0.0	0.0	0.0	0.0	0.00000
345417623	0.0	0.0	0.000000	0.064922	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.00000
446310786	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.00000
671027360	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.07256
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
439064864	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.00000
043935806X	0.0	0.0	0.045267	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.027269	0.0	0.0	0.0	0.0	0.0	0.00000
440220602	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.00000
671001795	0.0	0.0	0.000000	0.000000	0.112139	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.00000
440222656	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.00000

TF-IDF CALCULATION

```python
# Calculate term frequencies (TF)
tf_matrix = []
for i in range(num_docs):
    row_sum = sum(matrix[i])
    if row_sum == 0:
        tf_matrix.append([0] * num_words)
    else:
        tf_matrix.append([count / row_sum for count in matrix[i]])
```

TF Calculation

```python
# Calculate document frequencies (DF)
df_count = [0] * num_words
for j in range(num_words):
    for i in range(num_docs):
        if matrix[i][j] > 0:
            df_count[j] += 1

# Calculate inverse document frequency (IDF)
idf = [0] * num_words
for j in range(num_words):
    idf_value = (1 + num_docs) / (1 + df_count[j])

    # Custom logarithm function (natural log)
    log_value = 0
    x = (idf_value - 1) / (idf_value + 1)
    iteration_count = 20  # Number of terms to sum in the series (higher is more accurate)

    for n in range(1, iteration_count * 2, 2):
        log_value += (x**n) / n

    idf[j] = 1 + 2 * log_value  # 2 * log(x) approximates the natural log of (1 + x)
```

IDF Calculation

TF * IDF

```python
# Calculate TF-IDF manually
tfidf_matrix = []
for i in range(num_docs):
    tfidf_row = []
    for j in range(num_words):
        tfidf_value = tf_matrix[i][j] * idf[j]
        tfidf_row.append(tfidf_value)
    tfidf_matrix.append(tfidf_row)
```

BUILDING USER PROFILES (PART 2)

1. Load user history data from a CSV file.
2. Extract unique user IDs and their corresponding ISBNs.
3. Sum TF-IDF vectors of books read by each user.
4. Create a DataFrame from the combined user profiles.
5. Save the user profiles DataFrame to a CSV file.

userid	10	103	12	14th	150	16	1896	1920s	1925	1940s	1950s	1951	1954	1960
11676	0	0	0	0.064922	0.1121388	0	0	0	0	0	0.097936	0	0.085071	0
16795	0	0	0	0	0	0	0	0	0	0.0580483	0	0	0	0
22625	0	0.096747	0.0452670	0	0	0	0	0	0	0	0	0	0.085071	0
35859	0	0	0	0	0	0	0	0	0	0	0	0	0	0
95359	0	0.096747	0.0452670	0	0	0	0	0.07256	0.07256	0	0	0.07256	0	0.068529
104636	0	0	0	0	0	0	0	0	0	0	0	0	0	0
110912	0	0	0	0	0	0	0	0	0	0	0	0	0	0
204864	0	0	0	0	0	0	0	0	0	0	0	0	0.085071	0
271448	0	0	0.0452670	0	0	0	0	0	0	0	0	0	0	0

FINDING SIMILARITY (PART 2)

```python
def dot_product(v1, v2):
    return sum(x * y for x, y in zip(v1, v2))

def vector_norm(v):
    return sum(x ** 2 for x in v) ** 0.5

def cosine_similarity(v1, v2):
    dot_prod = dot_product(v1, v2)
    norm_v1 = vector_norm(v1)
    norm_v2 = vector_norm(v2)
    if norm_v1 == 0 or norm_v2 == 0:
        return 0  # Handle cases where a vector is all zeros
    return dot_prod / (norm_v1 * norm_v2)
```

1. Define a cosine similarity function.
2. Extract user profiles and item vectors from DataFrames.
3. Initialize a similarity matrix to store results.
4. For each user, check unread books and compute cosine similarity.
5. Store the similarity results in a DataFrame.
6. Save the similarity matrix DataFrame to a CSV file.

ISBN	440234743	971880107	345417623	446310786	671027360	60976845	044021145X	60938455	446672211	142000205	...	375727345	312924585	684872153	3166
userid															
11676	0.191029	0.147187	0.000000	0.231365	0.182759	0.213336	0.229177	0.000000	0.000000	0.200096	...	0.125387	0.177176	0.188097	0.23
16795	0.036566	0.029928	0.029657	0.080619	0.039451	0.052325	0.067684	0.025753	0.048731	0.065836	...	0.044937	0.028451	0.050374	0.0
22625	0.164726	0.132584	0.140318	0.163478	0.169480	0.199054	0.221548	0.132304	0.135666	0.198274	...	0.150458	0.135189	0.159141	0.0
35859	0.209272	0.145236	0.115314	0.179073	0.138981	0.198614	0.000000	0.103294	0.121453	0.153736	...	0.099983	0.149533	0.203659	0.1
95359	0.162871	0.128480	0.155836	0.000000	0.123489	0.187253	0.189103	0.138695	0.143043	0.169302	...	0.106110	0.142143	0.154014	0.0
104636	0.190746	0.127961	0.132450	0.147872	0.136943	0.158015	0.000000	0.087683	0.000000	0.113197	...	0.087494	0.143553	0.152878	0.1
110912	0.172254	0.125092	0.150686	0.228842	0.175565	0.199868	0.209219	0.177325	0.127353	0.189644	...	0.108448	0.134832	0.141726	0.1
204864	0.214402	0.152313	0.134376	0.237336	0.173566	0.202612	0.239934	0.152741	0.125101	0.202731	...	0.139971	0.177573	0.272118	0.1
271448	0.156100	0.133632	0.093455	0.135950	0.106772	0.208407	0.224568	0.080481	0.000000	0.176722	...	0.159947	0.147887	0.179116	0.1

DISPLAY THE FIRST 10 BOOKS TO EACH USER (PART 2)

1. Retrieve top 10 book recommendations for each user from the similarity DataFrame.
2. Fetch book titles from the original DataFrame based on ISBN.
3. Collect user, book, title, and similarity value into a list.
4. Create a DataFrame from the recommendations list.
5. Save the recommendations DataFrame to a CSV file.

User ID	Book's ISBN	Book's Title	Model's Calculated Value
11676	590353403	Harry Potter and the Sorcerer s Stone (Book 1)	0.346404
11676	385722206	Balzac and the Little Chinese Seamstress	0.314497
11676	345337662	Interview with the Vampire	0.301223
11676	1400034779	The No. 1 Ladies Detective Agency (Today Show...	0.291726
11676	014028009X	Bridget Jones s Diary	0.286780
...	...	...	...
271448	044021145X	The Firm	0.224568
271448	014028009X	Bridget Jones s Diary	0.221735
271448	059035342X	Harry Potter and the Sorcerer s Stone (Harry P...	0.220563
271448	590353403	Harry Potter and the Sorcerer s Stone (Book 1)	0.219570
271448	60959037	Prodigal Summer	0.215585

EVALUATING THE RESULTS (PART 2)

```
# Calculate correct predictions
correct_predictions = correctly_predicted.get(user, [])
incorrect_predictions = incorrectly_predicted.get(user, [])

# Calculate precision
if len(recommendations2[user]) > 0:
    precision = len(correct_predictions) / 90
else:
    precision = 0

# Calculate recall
if len(grouped_isbns[user]) > 0:
    recall = len(correct_predictions) / len(grouped_isbns[user])
else:
    recall = 0

# Calculate F1-score
if precision + recall > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
else:
    f1_score = 0
```

RESULTS (PART 2)

```
Overall Metrics:
Overall Precision: 0.2667
Overall Recall: 0.1491
Overall F1-Score: 0.1913
```

```
Correctly Predicted Books:
User 11676: ['590353403', '385722206', '345337662', '014028009X', '375726403', '439064872', '60987103']
User 16795: ['316569321', '440214041', '316769487', '61009059']
User 22625: ['385722206', '142001740']
User 35859: ['385722206']
User 95359: ['60915544', '312195516']
User 104636: ['440213525', '440224764', '014028009X']
User 110912: ['60959037']
User 204864: ['385722206', '375726403']
User 271448: ['316569321', '312195516']

Incorrectly Predicted Books:
User 11676: ['1400034779', '316769487', '60959037']
User 16795: ['043935806X', '014028009X', '385335482', '316096199', '446310786', '345337662']
User 22625: ['590353403', '439064872', '014028009X', '385335482', '439139600', '440213525', '316569321', '61009059']
User 35859: ['590353403', '440213525', '440224764', '316769487', '345337662', '439064864', '439136350', '1400034779', '014028009X']
User 95359: ['60987103', '380789035', '385722206', '345337662', '1400034779', '014028009X', '059035342X', '449005615']
User 104636: ['590353403', '439136350', '043935806X', '439064864', '446608955', '385722206', '316769487']
User 110912: ['60987103', '380789035', '345337662', '316769487', '375726403', '439139600', '059035342X', '590353403', '679746048']
User 204864: ['345337662', '1400034779', '684872153', '60987103', '440224764', '059035342X', '044021145X', '375725784']
User 271448: ['312278586', '440214041', '142001740', '044021145X', '014028009X', '059035342X', '590353403', '60959037']
```

REFERENCES

- https://www.nltk.org/
- https://www.kaggle.com/code/uthamkanth/beginner-tf-idf-and-cosine-similarity-from-scratch
- Source code for Mini Project 1, Part 1 (Jaccard Similarity)
- Source code for Mini Project 1 , Part 2 (Cosine Similarity)

THANK YOU