# BOOK RECOMMENDATION SYSTEM

Prepared for

**Prof: Arya Rahgozar**

Faculty of Engineering, University of uOttawa
Data Science Applications

Prepared by


**Ahmed Ibraheem Ali Badwy**
**Anas Ibrahim Ali Elbatra**
**Esmael Alahmady Ebrahim Ezz**
**Yousef Abd Al Haleem Ahmed Shindy**


Aug 5,2023

# Contents

# 1 Problem Formulation:

This project presents a content-based book recommender system that uses natural language processing and machine learning to offer personalized book suggestions based on book summaries. By analyzing book summaries, the system identifies similarities and associations between books, ensuring accurate and relevant recommendations. The goal is to connect readers with books that match their interests, enhancing their reading experience and promoting literature exploration in the digital age.

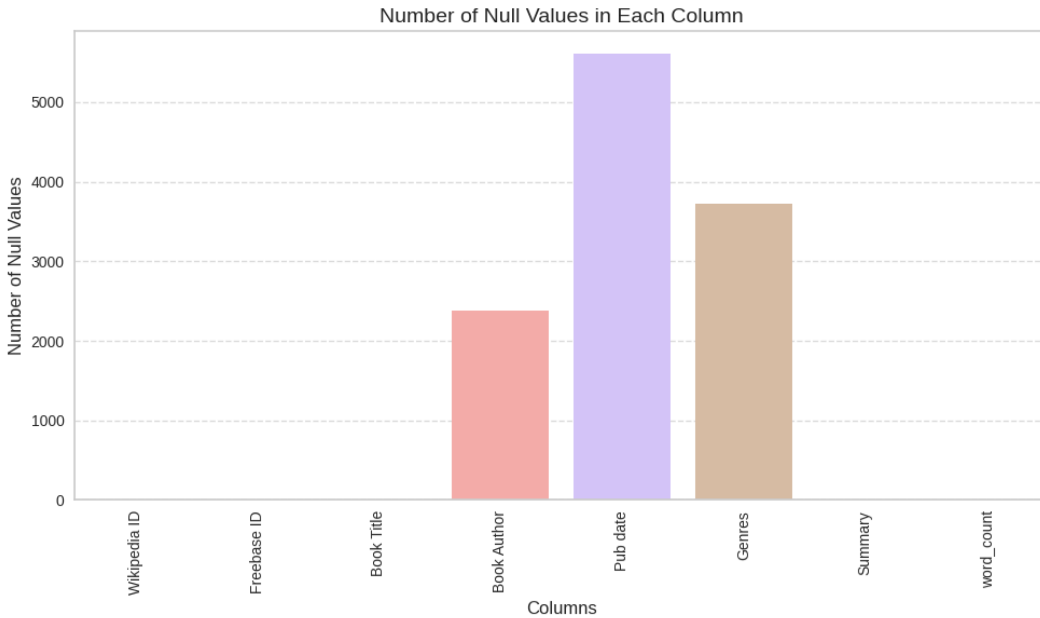
# 2 Data Preparation:

The dataset contains 16559 rows with 7 columns:

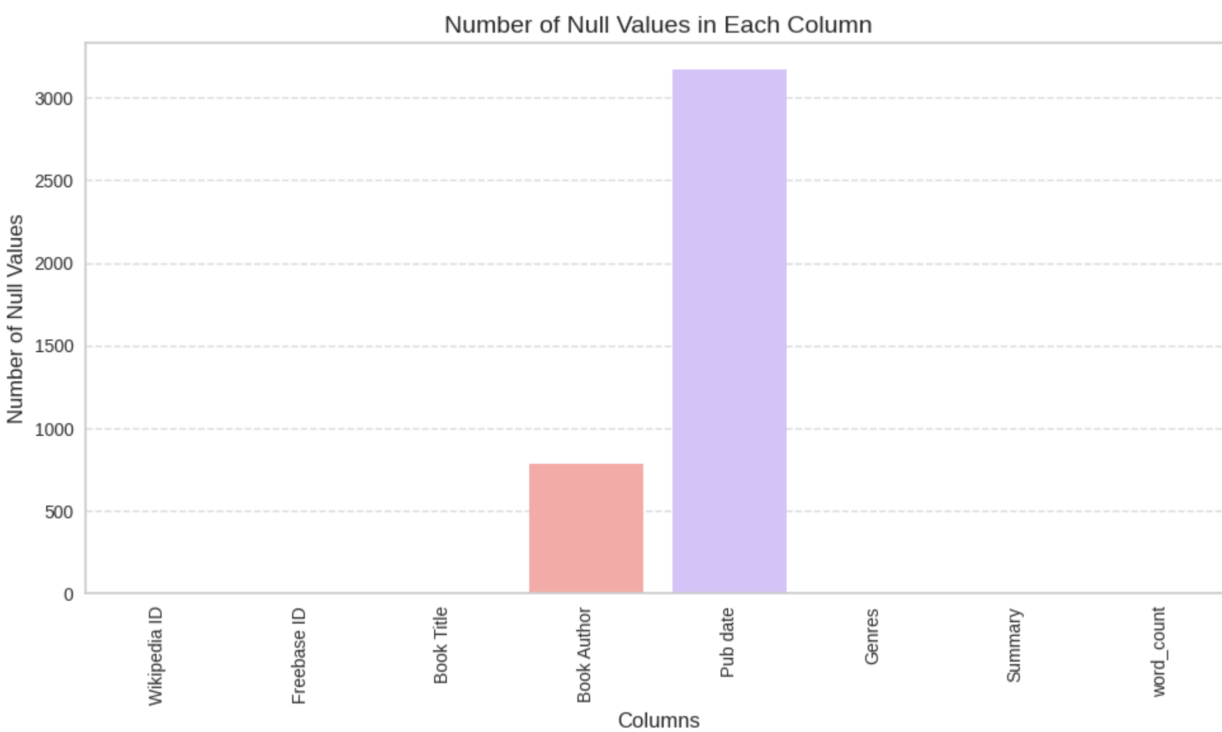
- **Wikipedia ID:** The unique identifier for each book from Wikipedia
- **Freebase ID:** The unique identifier for each book in the Freebase database
- **Book Title:** The title of the book
- **Book Author:** The name of the author who wrote the book
- **Pub Date:** The publication date of the book
- **Genres:** The genres associated with the book
- **Summary:** A short summary of the book

| | Wikipedia ID | Freebase ID | Book Title | Book Author | Pub date | Genres | Summary |
|---|---|---|---|---|---|---|---|
| **0** | 620 | /m/0hhy | Animal Farm | George Orwell | 1945-08-17 | {"/m/016lj8": "Roman \u00e0 clef", "/m/06nbt":... | Old Major, the old boar on the Manor Farm, ca... |
| **1** | 843 | /m/0k36 | A Clockwork Orange | Anthony Burgess | 1962 | {"/m/06n90": "Science Fiction", "/m/0l67h": "N... | Alex, a teenager living in near-future Englan... |
| **2** | 986 | /m/0ldx | The Plague | Albert Camus | 1947 | {"/m/02m4t": "Existentialism", "/m/02xlf": "Fi... | The text of The Plague is divided into five p... |
| **3** | 1756 | /m/0sww | An Enquiry Concerning Human Understanding | David Hume | NaN | NaN | The argument of the Enquiry proceeds by a ser... |
| **4** | 2080 | /m/0wkt | A Fire Upon the Deep | Vernor Vinge | NaN | {"/m/03lrw": "Hard science fiction", "/m/06n90... | The novel posits that space around the Milky ... |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **16554** | 36934824 | /m/0m0p0hr | Under Wildwood | Colin Meloy | 2012-09-25 | NaN | Prue McKeel, having rescued her brother from ... |
| **16555** | 37054020 | /m/04f1nbs | Transfer of Power | Vince Flynn | 2000-06-01 | {"/m/01jfsb": "Thriller", "/m/02xlf": "Fiction"} | The reader first meets Rapp while he is doing... |
| **16556** | 37122323 | /m/0n5236t | Decoded | Jay-Z | 2010-11-16 | {"/m/0xdf": "Autobiography"} | The book follows very rough chronological ord... |
| **16557** | 37132319 | /m/0n4bqb1 | America Again: Re-becoming The Greatness We Ne... | Stephen Colbert | 2012-10-02 | NaN | Colbert addresses topics including Wall Stree... |
| **16558** | 37159503 | /m/073nkd | Poor Folk | Fyodor Dostoyevsky | 1846 | {"/m/02ql9": "Epistolary novel", "/m/014dfn": ... | Makar Devushkin and Varvara Dobroselova are s... |

First, we handle the missing values:




We discover that we have null values in the **Genres** column so we drop these rows.



Then we define a function to get the word count of the summary column

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

df["word_count"] = df["Summary"].apply(count_words)
```

| | Wikipedia ID | Freebase ID | Book Title | Book Author | Pub date | Genres | Summary | word_count |
|---|---|---|---|---|---|---|---|---|
| **0** | 620 | /m/0hhy | Animal Farm | George Orwell | 1945-08-17 | {"/m/016lj8": "Roman \u00e0 clef", "/m/06nbt":... | Old Major, the old boar on the Manor Farm, ca... | 957 |
| **1** | 843 | /m/0k36 | A Clockwork Orange | Anthony Burgess | 1962 | {"/m/06n90": "Science Fiction", "/m/0l67h": "N... | Alex, a teenager living in near-future Englan... | 998 |
| **2** | 986 | /m/0ldx | The Plague | Albert Camus | 1947 | {"/m/02m4t": "Existentialism", "/m/02xlf": "Fi... | The text of The Plague is divided into five p... | 1119 |
| **3** | 1756 | /m/0sww | An Enquiry Concerning Human Understanding | David Hume | NaN | NaN | The argument of the Enquiry proceeds by a ser... | 2825 |
| **4** | 2080 | /m/0wkt | A Fire Upon the Deep | Vernor Vinge | NaN | {"/m/03lrw": "Hard science fiction", "/m/06n90... | The novel posits that space around the Milky ... | 722 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **16554** | 36934824 | /m/0m0p0hr | Under Wildwood | Colin Meloy | 2012-09-25 | NaN | Prue McKeel, having rescued her brother from ... | 151 |
| **16555** | 37054020 | /m/04f1nbs | Transfer of Power | Vince Flynn | 2000-06-01 | {"/m/01jfsb": "Thriller", "/m/02xlf": "Fiction"} | The reader first meets Rapp while he is doing... | 211 |
| **16556** | 37122323 | /m/0n5236t | Decoded | Jay-Z | 2010-11-16 | {"/m/0xdf": "Autobiography"} | The book follows very rough chronological ord... | 307 |
| **16557** | 37132319 | /m/0n4bqb1 | America Again: Re-becoming The Greatness We Ne... | Stephen Colbert | 2012-10-02 | NaN | Colbert addresses topics including Wall Stree... | 20 |
| **16558** | 37159503 | /m/073nkd | Poor Folk | Fyodor Dostoyevsky | 1846 | {"/m/02ql9": "Epistolary novel", "/m/014dfn": ... | Makar Devushkin and Varvara Dobroselova are s... | 636 |

Then we convert the **Genres** column from JSON format to a list of genres.

```
[16] try:
         df["Genres"] = df["Genres"].map(
             lambda genre: list(json.loads(genre).values()))
     except TypeError:
         print("error")
```

| Genres |
|---|
| [Roman à clef, Satire, Children's literature, ... |
| [Science Fiction, Novella, Speculative fiction... |
| [Existentialism, Fiction, Absurdist fiction, N... |
| [Hard science fiction, Science Fiction, Specul... |
| [War novel, Roman à clef] |
| ... |
| [Science Fiction] |
| [Thriller, Fiction, Suspense] |
| [Thriller, Fiction] |
| [Autobiography] |
| [Epistolary novel, Speculative fiction] |

Due to the large amount of data and text in the summary we couldn't work on the whole dataset due to memory limitations so we took a random sample of 1000 rows from the dataset.

```
random_sample = df.sample(n=1000, random_state=42)
```

```
random_sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 10289 to 14322
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Wikipedia ID  1000 non-null   int64
 1   Freebase ID   1000 non-null   object
 2   Book Title    1000 non-null   object
 3   Book Author   950 non-null    object
 4   Pub date      754 non-null    object
 5   Genres        1000 non-null   object
 6   Summary       1000 non-null   object
 7   word_count    1000 non-null   int64
dtypes: int64(2), object(6)
memory usage: 70.3+ KB
```

## 3 Text Feature Engineering

We start by defining a function to get the word net to use in lemmatization.

```
def get_word_net_pos(tag):
    if tag.startswith("J"):
        return wordnet.ADJ
    elif tag.startswith("V"):
        return wordnet.VERB
    elif tag.startswith("N"):
        return wordnet.NOUN
    elif tag.startswith("R"):
        return wordnet.ADV
    else:
        return wordnet.NOUN
```

Then we define a function to tokenize the words, convert them into lowercase, remove stop words and words less than 3 letters and lemmatize the words based on their parts of speech then returns the processed summary.

```
lemmatizer = WordNetLemmatizer()
def preprocess_sentence(sentence):
    # Tokenize and convert to lowercase
    words = nltk.word_tokenize(sentence.lower())
    filtered_words = [
        word for word in words if word not in stop_words and len(word) >= 3
    ]
    sent = ""
    x = nltk.pos_tag(filtered_words)
    for word, tag in x:
        lemma = lemmatizer.lemmatize(word, pos=get_word_net_pos(tag))
        sent += lemma + " "
    sentence = regexp_tokenize(sent, r"([a-zA-Z]{3,})[\s]")
    return " ".join(sentence)
```

We apply this function to our random sample then we have summary text data ready for transformation.

```
random_sample["Processed_Sentences"] = random_sample["Summary"].apply(
    preprocess_sentence
)
```

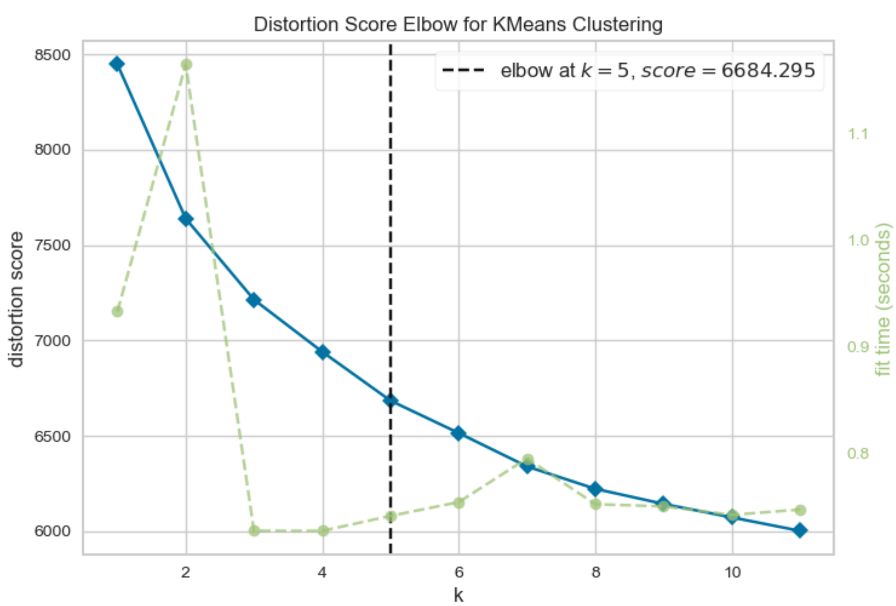
| Processed_Sentences |
|---|
| book alternately comic serious chart durrell e... |
| cassandra palmer clairvoyant see vision future... |
| protagonist arthur art mumby old sister myrtle... |
| book begin rachel adopt sister hilary living r... |
| story begin take laura mile home dead winter f... |
| ... |

Then we define a function to transform the text in the summary columns into numerical vectors using Word2Vec and apply this function to the **Summary** column.

```
def word2vec_transform():
    # Train the Word2Vec model
    sentences = [
        paragraph.split() for paragraph in random_sample["Processed_Sentences"]
    ]
    model = Word2Vec(sentences, vector_size=50,
                     window=5, min_count=1, epochs=100)

    # Transform each paragraph using Word2Vec
    transformed_data = []
    for paragraph in sentences:
        paragraph_vector = np.mean([model.wv[word]
                               for word in paragraph], axis=0)
        transformed_data.append(paragraph_vector)

    # Create a DataFrame from the transformed data
    vocab = [f"feature_{i+1}" for i in range(model.vector_size)]
    data = pd.DataFrame(transformed_data, columns=vocab)

    return data
```

# 4 Clustering

We used the K-Means algorithm for clustering. First, we run the algorithm on a range of numbers to decide on the best number of clusters using the elbow method.



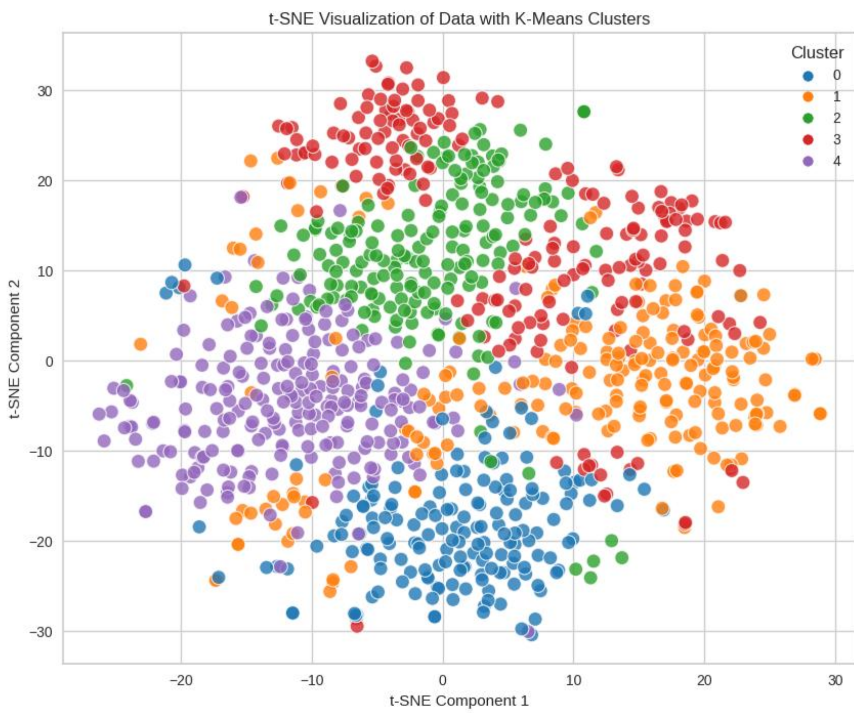
From the graph we see that the best number of clusters is 5 so we use the K-Means with K=5 and plot the TSNE visualization of the clusters.

```
def KMEANS(data, num_cluster):
    km_cls = KMeans(
        n_clusters=num_cluster,
        init="k-means++",
        max_iter=300,
        n_init=10,
        random_state=42,
    )
    km_cls.fit(data)
    km_labels = km_cls.predict(data)
    score = silhouettescore(data, km_labels)
    return km_labels, score
```

```
labels, score = KMEANS(train_data, 5)
```

```
tsne = TSNE(n_components=2, random_state=42)
data_tsne = tsne.fit_transform(train_data)
plt.figure(figsize=(10, 8))
sns.scatterplot(x=data_tsne[:, 0], y=data_tsne[:, 1], hue=labels, palette='tab10', s=100, alpha=0.8)
plt.title("t-SNE Visualization of Data with K-Means Clusters")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.legend(title="Cluster")
plt.show()
```

We can see from the visualization that there is some separation between different clusters.



## 5 Classification

In order to classify our data, we will have to prepare a target column for classification. Our target column is the **Genres** column. And since each book has multiple values in the **Genre** column we will use the new-found cluster labels as our new genre. We start by assigning the cluster labels to the data and create smaller subsets of the data, each with a unique label.

```
random_sample["labels"] = labels
random_sample_0 = random_sample[random_sample["labels"] == 0]
random_sample_1 = random_sample[random_sample["labels"] == 1]
random_sample_2 = random_sample[random_sample["labels"] == 2]
random_sample_3 = random_sample[random_sample["labels"] == 3]
random_sample_4 = random_sample[random_sample["labels"] == 4]
```

Then we iterate over the genres values of each subset to get the most frequent genre of each cluster

```
genres = list()
for genre in random_sample_0["Genres"]:
    for i in range(len(genre)):
        genres.append(genre[i])
print(Counter(genres))
```

```
Counter({'Science Fiction': 135, 'Speculative fiction': 123, 'Fiction': 63,
```

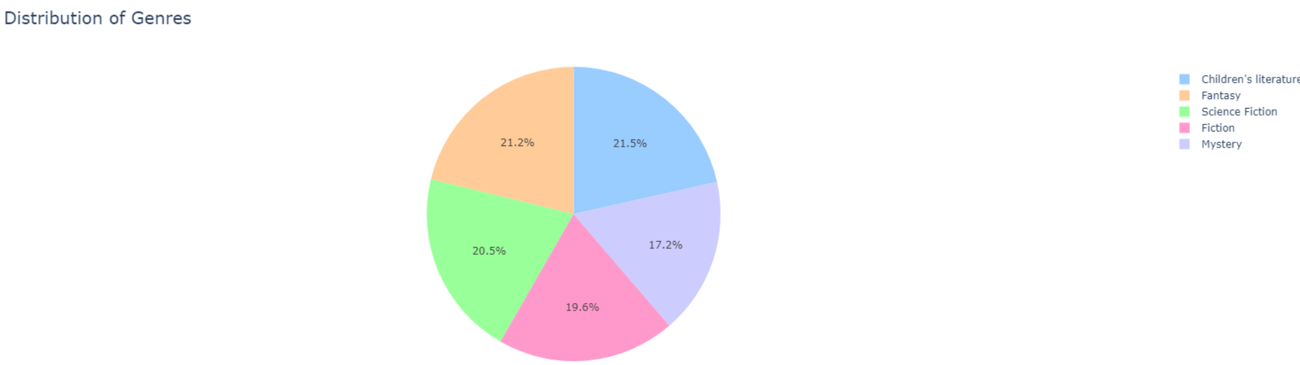
Then we assign the most frequent genre of each cluster as the new genre for this cluster

```
random_sample.loc[random_sample["labels"] == 0, "Genres"] = "Science Fiction"
random_sample.loc[random_sample["labels"] == 1, "Genres"] = "Fiction"
random_sample.loc[random_sample["labels"] == 2, "Genres"] = "Fantasy"
random_sample.loc[random_sample["labels"] == 3, "Genres"] = "Children's literature"
random_sample.loc[random_sample["labels"] == 4, "Genres"] = "Mystery"
```

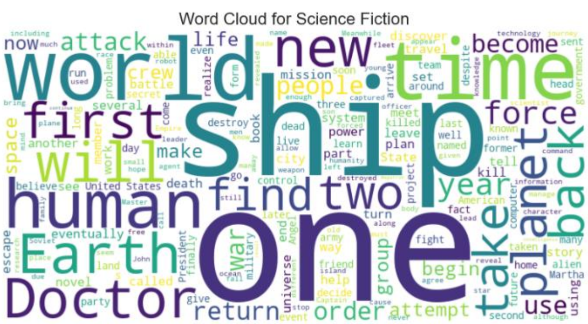
```
random_sample
```

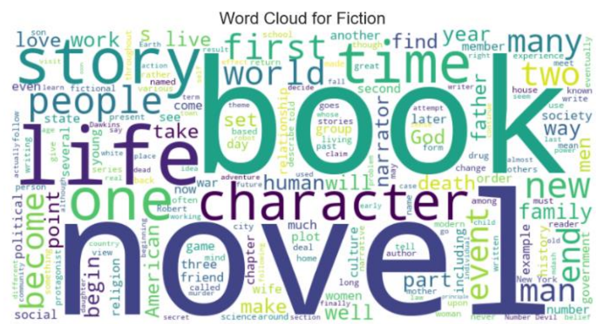
| | Wikipedia ID | Freebase ID | Book Title | Book Author | Pub date | Genres | Summary | word_count | Processed_Sentences | labels |
|---|---|---|---|---|---|---|---|---|---|---|
| **10289** | 12200712 | /m/02vvm7d | Bitter Lemons | Lawrence Durrell | NaN | Fiction | The book is alternately comic and serious, ch... | 324 | book alternately comic serious chart durrell e... | 1 |
| **14582** | 24694031 | /m/080nbjn | Touch the Dark | NaN | 2006-06 | Fantasy | Cassandra Palmer is a clairvoyant. She can se... | 495 | cassandra palmer clairvoyant see vision future... | 2 |
| **9975** | 11494990 | /m/02rftn2 | Starcross | Philip Reeve | 2007-10 | Fantasy | Protagonist Arthur ("Art") Mumby and his olde... | 321 | protagonist arthur art mumby old sister myrtle... | 2 |
| **10735** | 13340278 | /m/03c273l | Wintle's Wonders | Noel Streatfeild | 1957 | Children's literature | As the book begins, Rachel and her adopted si... | 333 | book begin rachel adopt sister hilary living r... | 3 |
| **10532** | 12861063 | /m/02x87t_ | These Happy Golden Years | Laura Ingalls Wilder | 1943 | Children's literature | As the story begins, Pa is taking Laura 12 mi... | 851 | story begin take laura mile home dead winter f... | 3 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

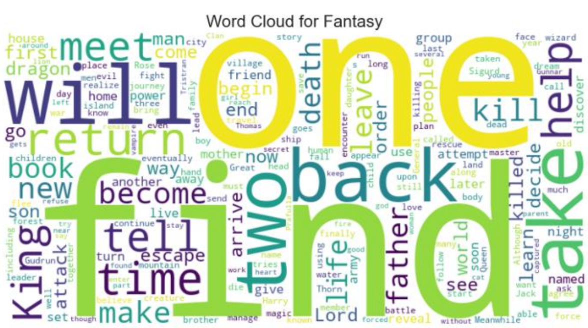
And plot a pie chart to check the genres column for imbalance. We can see that the data is balanced.

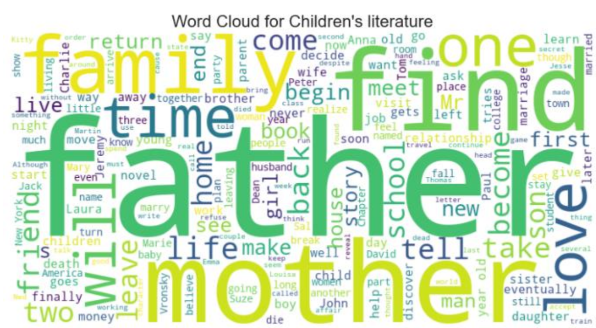


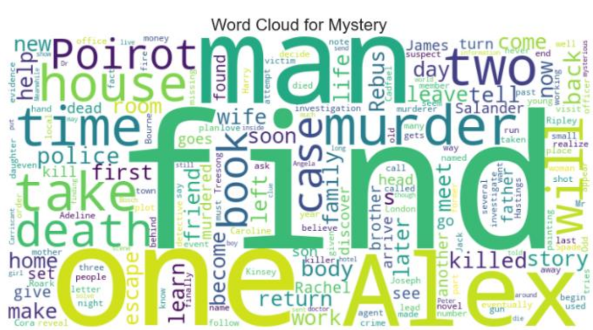

We explore the summaries of different genres using word clouds.



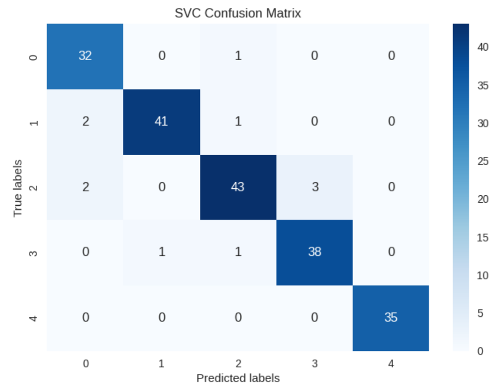



Now the data is ready for classification. We used five different classification models.

## SVC:

Score SVC
94.5

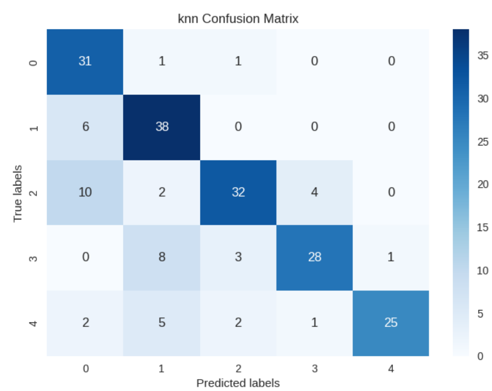
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.89 | 0.97 | 0.93 | 33 |
| 1 | 0.98 | 0.93 | 0.95 | 44 |
| 2 | 0.93 | 0.90 | 0.91 | 48 |
| 3 | 0.93 | 0.95 | 0.94 | 40 |
| 4 | 1.00 | 1.00 | 1.00 | 35 |
| accuracy | | | 0.94 | 200 |
| macro avg | 0.95 | 0.95 | 0.95 | 200 |
| weighted avg | 0.95 | 0.94 | 0.95 | 200 |




## KNN:

Score knn
77.0

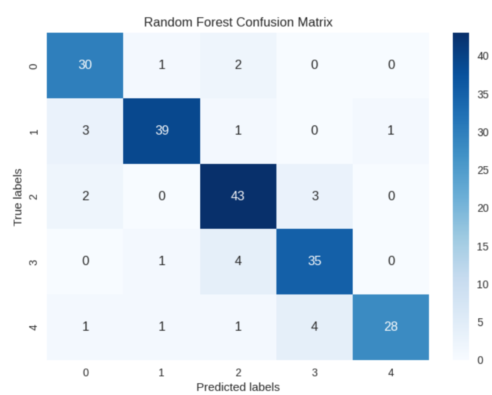
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.63 | 0.94 | 0.76 | 33 |
| 1 | 0.70 | 0.86 | 0.78 | 44 |
| 2 | 0.84 | 0.67 | 0.74 | 48 |
| 3 | 0.85 | 0.70 | 0.77 | 40 |
| 4 | 0.96 | 0.71 | 0.82 | 35 |
| accuracy | | | 0.77 | 200 |
| macro avg | 0.80 | 0.78 | 0.77 | 200 |
| weighted avg | 0.80 | 0.77 | 0.77 | 200 |




## Random Forrest:

Score Random Forest
87.5

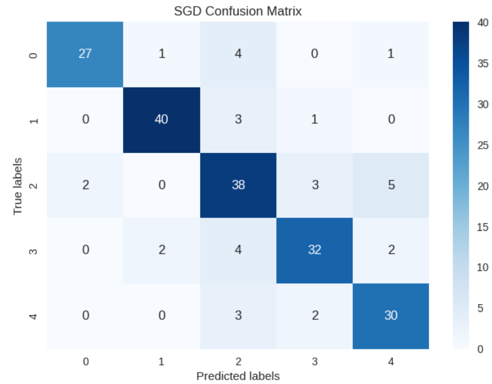
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.83 | 0.91 | 0.87 | 33 |
| 1 | 0.93 | 0.89 | 0.91 | 44 |
| 2 | 0.84 | 0.90 | 0.87 | 48 |
| 3 | 0.83 | 0.88 | 0.85 | 40 |
| 4 | 0.97 | 0.80 | 0.88 | 35 |
| accuracy | | | 0.88 | 200 |
| macro avg | 0.88 | 0.87 | 0.87 | 200 |
| weighted avg | 0.88 | 0.88 | 0.88 | 200 |



## SGD:

Score SGD
83.5

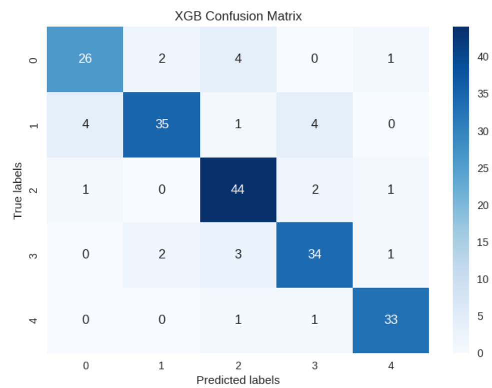
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.93 | 0.82 | 0.87 | 33 |
| 1 | 0.93 | 0.91 | 0.92 | 44 |
| 2 | 0.73 | 0.79 | 0.76 | 48 |
| 3 | 0.84 | 0.80 | 0.82 | 40 |
| 4 | 0.79 | 0.86 | 0.82 | 35 |
| accuracy | | | 0.83 | 200 |
| macro avg | 0.84 | 0.84 | 0.84 | 200 |
| weighted avg | 0.84 | 0.83 | 0.84 | 200 |



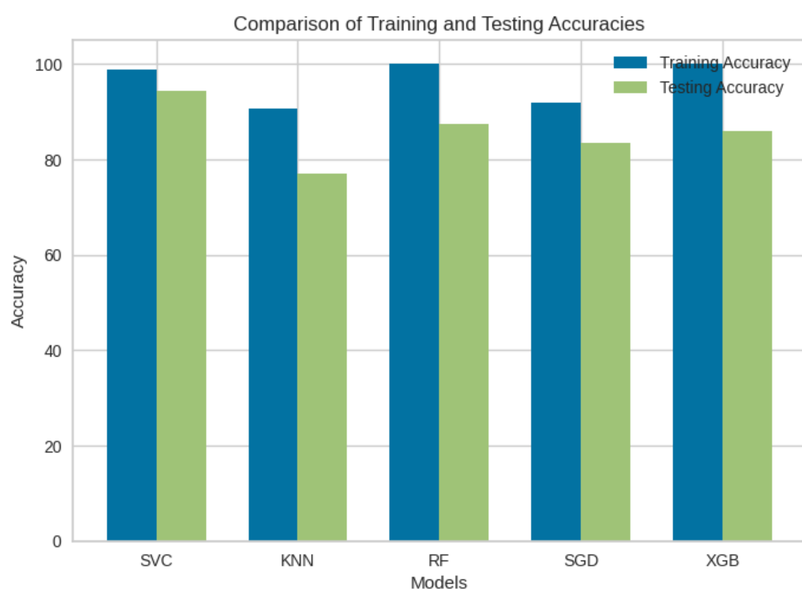

## XGB:

Score XGB
86.0

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.84 | 0.79 | 0.81 | 33 |
| 1 | 0.90 | 0.80 | 0.84 | 44 |
| 2 | 0.83 | 0.92 | 0.87 | 48 |
| 3 | 0.83 | 0.85 | 0.84 | 40 |
| 4 | 0.92 | 0.94 | 0.93 | 35 |
| accuracy | | | 0.86 | 200 |
| macro avg | 0.86 | 0.86 | 0.86 | 200 |
| weighted avg | 0.86 | 0.86 | 0.86 | 200 |




And compare the Train and test accuracies of the models:



# 6 Chatbot

We created an interactive chatbot that the user can use to get recommendation from. Our chatbot has two options:

- **Recommendation by book name**: This option allows users to input a specific book name, and the chatbot extracts this name and retrieves corresponding recommendations. This option is useful for users who have a specific book in mind.
- **Recommendation by genre**: This option allows users to receive recommendation based on their favorite genre. This option is useful for users who have a specific preference in genres.

We Integrated the chatbot with Dialogflow to build a conversational interface for our chatbot. However, because the webhook doesn’t accept HTTP, we used ngrok to establish a secure connection and enable HTTPS.
We start by defining a function to get recommendations based on a certain book title.

```python
def get_similar_books_in_same_genre(dataframe,input_book_title, book_embeddings, genre_mapping, num_similar_books=5):
    input_genre = genre_mapping.get(input_book_title)
    genres = dataframe["Genres"].unique()
    if input_book_title in book_embeddings:
        input_embedding = book_embeddings[input_book_title]

        similarities = {}
        for book_title, embedding in book_embeddings.items():
            if book_title != input_book_title and genre_mapping.get(book_title) == input_genre:
                similarity = cosine_similarity( X: [input_embedding],  Y: [embedding])[0][0]
                similarities[book_title] = similarity

        similar_books = sorted(similarities, key=similarities.get, reverse=True)
        similar_books = [f'"{book}"' for book in similar_books if book != input_book_title][:num_similar_books]
        return  f'Here are the most similar books for you :  {" - ".join(similar_books)} ........ Enter a different genre or book for another recommendation!.'
    else:
        random_books = []
        for genre in genres:
            genre_books = dataframe[dataframe["Genres"] == genre]
            random_book_title = random.choice(genre_books["Book Title"].tolist())
            random_books.append(
                f"'{random_book_title}' from the '{genre}' genre."
            )

        return f"I'm sorry, I don't have any recommendation for this but you can Try {random_books}"
```

We take the book title from the user and search for it in our dataset to get its corresponding genre if this title exists in the dataset then it applies cosine similarity on all books of the same genre and returns a dictionary with the five most similar books to the requested book or return five random samples from the dataset if it doesn’t exist.

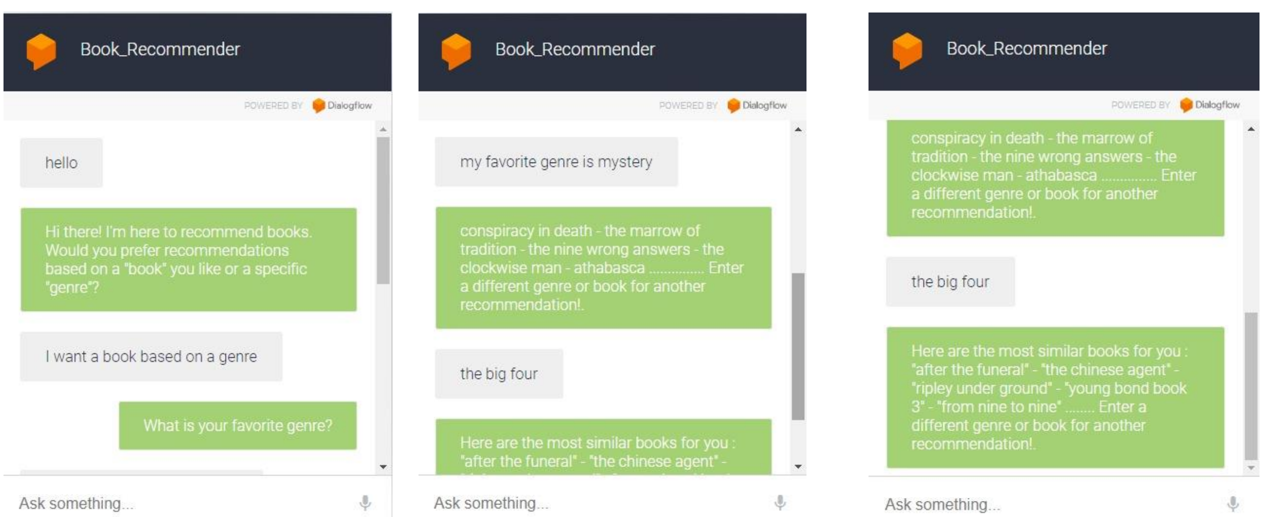
We also define a function to provide recommendations based on a specific genre.

```python
def recommend_books_by_genre(dataframe, input_genre):
    genres = dataframe["Genres"].unique()
    if input_genre in ['crime', 'suspense', 'detective', 'thriller', 'spy fiction',  'hardboiled', 'whodunnit', 'music', 'cabal', 'conspiracy', 'cyberpunk' ]:
        input_genre = 'mystery'
    elif input_genre in ['war', 'novel', 'ya', 'non-fiction', 'comic', 'novella', 'utopian', 'dystopian', 'action']:
        input_genre = 'fiction'
    elif input_genre in ['adventure', 'horror', 'comedy', 'historical', 'parallel','humour']:
        input_genre = 'fantasy'
    elif input_genre in ['satire', 'short story', 'romance', 'autobiography', 'biography', 'gothic', 'goth',"children's literature"]:
        input_genre = "children's literature"
    elif input_genre in ['speculative fiction']:
        input_genre = 'science fiction'
    if input_genre in genres:
        genre_books = dataframe[dataframe["Genres"] == input_genre]
        recommended_books = genre_books.sample(n=5)
        return f'{" - ".join(recommended_books["Book Title"].tolist())} ............... Enter a different genre or book for another recommendation!.'
    else:
        random_books = []
        for genre in genres:
            genre_books = dataframe[dataframe["Genres"] == genre]
            random_book_title = random.choice(genre_books["Book Title"].tolist())
            random_books.append(
                f"'{random_book_title}' from the '{genre}' genre."
            )

        return f"I'm sorry, I don't have any recommendation for this type of books but you can Try {random_books}"
```

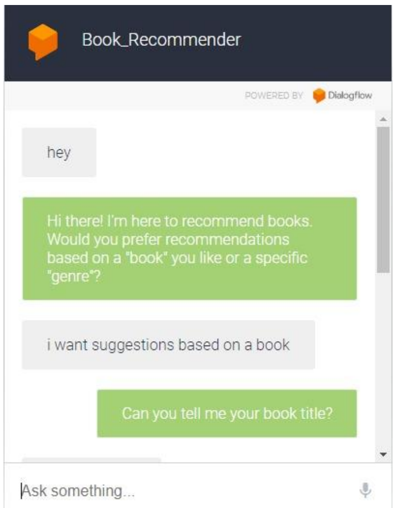
We defined some conditions to map various similar genres together under one genre that's present in the dataset. Then it takes the resulting genre and checks for its availability in our dataset. Then returns five random books from the same genre as a recommendation or five random samples from the dataset if the requested genre doesn't exist in the dataset.

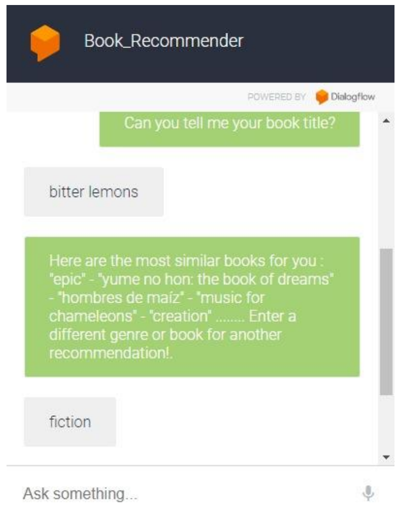
## 7 Chatbot Performance

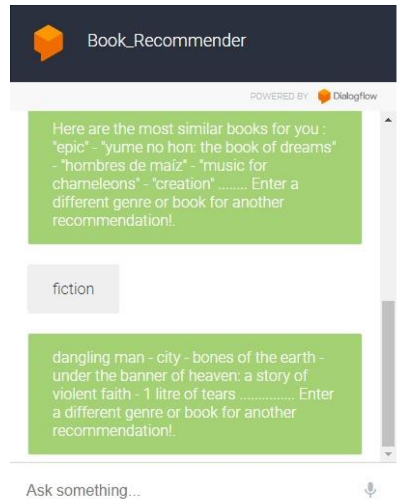
We tried the chatbot with different scenarios and it had a good performance with the pretrained scenarios. We prompted the book for a suggestion based on a genre and specified mystery as the requested genre then we asked for a recommendation based on a book which is "The big four" to which it responded with five books, the first of which is "After the Funeral". Both books are crime-mystery books and both are written by Agatha Christie showing how accurate the recommendation model is.



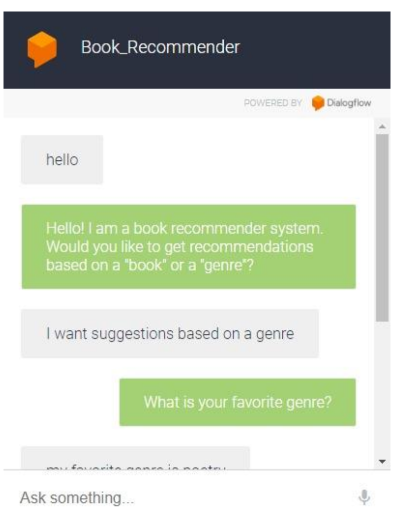
We tried another case where we prompted the chatbot for a recommendation based on a book title then for a recommendation based on a genre to see how well the chatbot handled switching between requesting based on books and genres

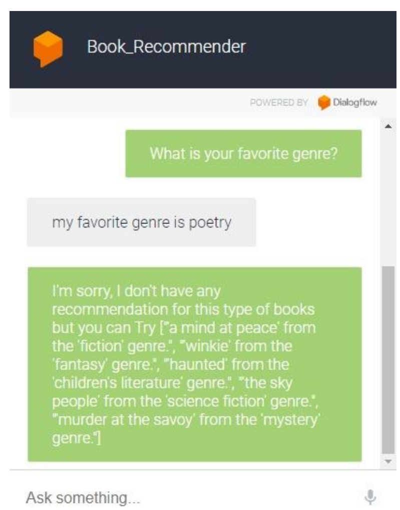



It showed a weaker performance when prompted with a book title or a genre that was not in the dataset due to the small amount of data it was trained on. However, we made the chatbot respond with five random books from the dataset if it couldn't comply to the user's request.



## 8 Innovativeness

The values of the **Genres** columns in our data was in JSON format and had multiple values per entry so it was difficult to use this column for classification. So, we converted the JSON data to a list of genres, then we applied our text feature engineering on the **Summary** column and transformed the result to numerical vectors ready for clustering using Word2Vec, applied K-Means on the data and labeled the data with the resulting cluster labels. Then explored each cluster and assigned the most frequent genre in the cluster as the only genre for all the cluster members, resulting in a data set with 5 unique labels across the data instead of the initial 217 genres.

So, our data preprocessing, feature engineering and clustering allowed us to classify the books based on their summary.

## 9 References

[1] https://www.kaggle.com/datasets/ymaricar/cmu-book-summary-dataset