

CIS- 8391 Big Data Analytics Experience

Project Proposal

April 11, 2020



TELL ME A BOOK!

Team: Data Guys

Team Members:

1. **Aditi Deokar**
2. **Clinton Roy**
3. **Mrunal Limaye**
4. **Nikita Kirane**

Table of Contents:

- I. Business Problem
- II. Value Creation
- III. Road Map
- IV. Architecture Diagram
- V. AWS Configuration
- VI. Lex – Messenger Integration
- VII. Workflow of the project
- VIII. Recommendation System
- IX. Challenges and Solutions
- X. Learning Outcome
- XI. Business Prospects

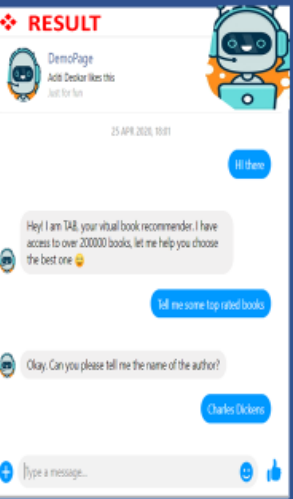

Summary Slide:



Tell A Book

By: Aditi Deokar, Clinton Roy, Mrunal Limaye, Nikita Kirane
MS in Information Systems Program, Robinson College of Business
Advisor: Professor Vijay Gandapodi



❖ INTRODUCTION <ul style="list-style-type: none">Millions of books available making it too difficult to choose one. TAB can easily give suggestion from its exhaustive database.TAB provides recommendations based on as per genre, page count, author and title.	❖ METHODOLOGY <ul style="list-style-type: none">Amazon Lex Chatbot is the starting point in our project. We have built a chatbot using Amazon Lex for our book recommendation system.The chatbot has been integrated with messenger. The user types any question in the chatbot, which will be passed further to an intent in AWS Lex.As and when any question is asked to the bot, the corresponding functions will be called with the help of AWS lambda.The lambda instance will contain python code that would interact with the lambda functions and the S3 service. The book recommender data will be stored as a CSV file in the S3.The functions will read the data from the S3 and pass the correct output again to the bot through lambda, which will be shown to the user.	❖ RESULT  <p>The screenshot shows a chatbot interface with a blue header and a white background. The chatbot's name is 'Tell A Book' and it is described as 'Your virtual book recommender'. The chat history shows a user asking for top-rated books, and the chatbot responding with 'Charles Dickens'. The chatbot is currently asking the user for the name of the author.</p>
❖ DATASET <ul style="list-style-type: none">Search based on: Author, Genre, Number of pages, Book Title, Top 10 by popularityRecommendation based on user similarity(ratings)Total no. of data sources : 3 (.csv)Programming Language: PythonVisualization tool: Python/Tableau	❖ Architecture Components:  <p>The diagram shows the architecture components: Amazon Lex (a blue speech bubble icon), AWS Lambda (a blue icon with a white lambda symbol), and Amazon S3 (a green icon with a white S3 symbol).</p>	

I] Business Problem:

"A book is a gift you can open again and again."

There are so many types of books like Science Fiction, Drama, Action and Adventure, Romance, Mystery, Horror, self-help, Travel, and this list can go on and on. It can be a difficult choice for the reader when such diverse options are available.

A chatbot is like a virtual assistant who creates a better customer experience!

In technical terms, it is a software application designed to simulate human conversation over the internet. Chatbot technology uses natural language processing to understand what a human asks and adapts its response to help users.

A chatbot saves time, which gets used up in searching as it already has answers to specific questions that the users generally ask.

The idea behind this project is to build a chatbot that would work as a book search assistant and would recommend books based on the interaction with the user.

The name of our chatbot is TAB – Tell A Book!

The user can ask TAB for the book title, publication, author, genre, ratings, etc.

Moreover, TAB will also tell the user top-rated books, top-rated authors, genre with maximum books, and other insights like these.

Recommender systems are among the most popular applications of data science.

They are used to predict the "rating" or "preference" that a user would give to an item.

Amazon -suggests products to customers, YouTube suggests which video to play next on autoplay, and Facebook uses it to recommend pages to like and people to follow.

Netflix and Spotify- the business model and its success revolve around the potency of their recommendations. Netflix even offered a million dollars in 2009 to anyone who could improve its system by 10%.

Examples of questions TAB will answer:

1. Tell me the top-rated authors.
2. Tell me highly-rated books.
3. Which are the top books in a genre?
4. Books published in a particular year, etc.

III] Value Creation:

1. People will find it easy to choose books.
2. No need to memorize author names, titles, etc.
3. Users can get suggestions for new books worth reading.
4. Saves a lot of time as there is no need to Google search,
5. Integration on popular platforms like Kindle, Amazon Bookstore or even Georgia State University Library.

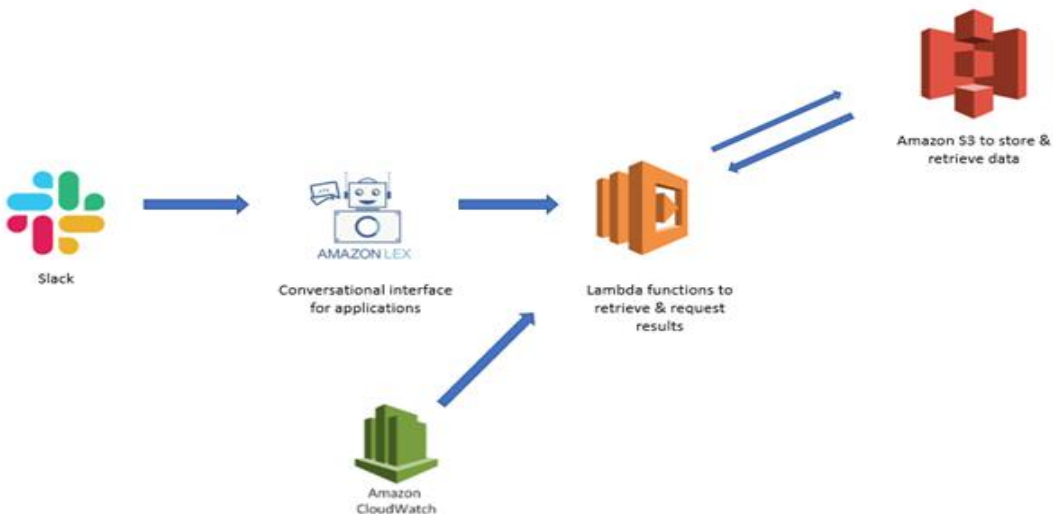
III] Road Map:

We followed the following steps to achieve what we had proposed in the initial part of the project:

1. Data Sourcing
2. Data Cleaning
3. Data Pre-Processing
4. Data Stitching
5. Data Visualization
6. Data Storage
7. Integration
8. Business Solution

Data Sourcing to Data stitching has been covered in detail in the proposal paper so here we will explain the actual analysis, integration and working of the recommendation system.

IV] Architecture Diagram:



Amazon Lex Chabot is the starting point in our project. We have built a chabot using Amazon Lex for our book recommendation system. The chatbot has been integrated with messenger.

The user will type any question in the chatbot, which will be passed further to an intent in AWS Lex.

As and when any question is asked to the bot, the corresponding functions will be called with the help of AWS lambda.

The lambda instance will contain python code that would interact with the lambda functions and the S3 service. The book recommender data will be stored as a CSV file in the S3.

The functions will read the data from the S3 and pass the correct output again to the bot through lambda, which will be shown to the user

V] AWS Configuration:

AWS consists of so many different cloud computing products and services. The highly profitable Amazon division provides servers, storage, networking, remote computing, email, mobile development, and security.

Amazon Web Services provides services from dozens of data centers spread across various availability zones in regions across the world.

A business will choose one or multiple availability zones for a variety of reasons, such as compliance and proximity to end customers.

In this project, we are using the following AWS services.

1. Amazon Simple Storage Service (S3)

Amazon Simple Storage Service (S3) provides scalable object storage for data backup, collection, and analytics. An IT professional stores data and files as S3 objects, which can range up to 5 gigabytes (GB) inside S3 buckets to keep them organized. A business can save money with S3 through its Infrequent Access storage tier or by using Amazon Glacier for long-term cold storage.

[Amazon S3](#) > [bigdata-project1904201](#)

bigdata-project1904201

Overview

Properties

Permissions

Management

Access points

Q

Type a prefix and press Enter to search. Press ESC to clear.

Upload





Create folder

Download

Actions

US East (N. Virginia)

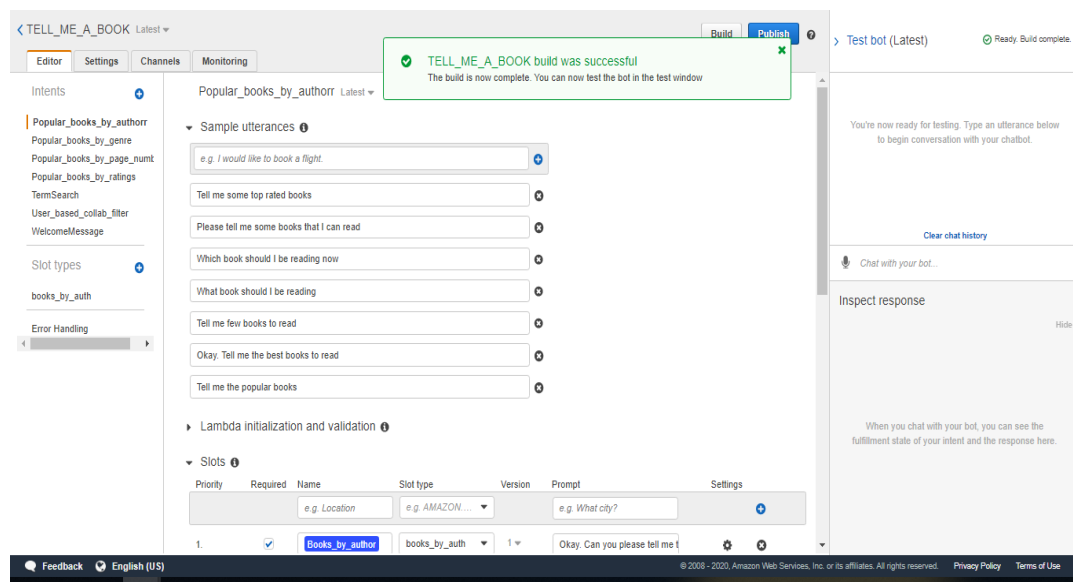
Viewing 1 to 8

<input type="checkbox"/>	Name	Last modified	Size	Storage class
<input type="checkbox"/>	 Books_Ratings (2).csv	Apr 25, 2020 12:04:49 PM GMT-0400	29.3 MB	Standard
<input type="checkbox"/>	 Books_Ratings.csv	Apr 25, 2020 11:04:19 AM GMT-0400	29.3 MB	Standard
<input type="checkbox"/>	 Books_cleaned.csv	Apr 20, 2020 5:12:55 PM GMT-0400	51.6 MB	Standard
<input type="checkbox"/>	 Books_cleaned_nodups.csv	Apr 25, 2020 11:05:29 AM GMT-0400	46.4 MB	Standard

2. Amazon Lex

Amazon Lex is an AWS service for building conversational interfaces for applications using voice and text. With Amazon Lex, the same conversational engine that powers Amazon Alexa is now available to any developer, enabling you to build sophisticated, natural language Chabot into your new and existing applications.

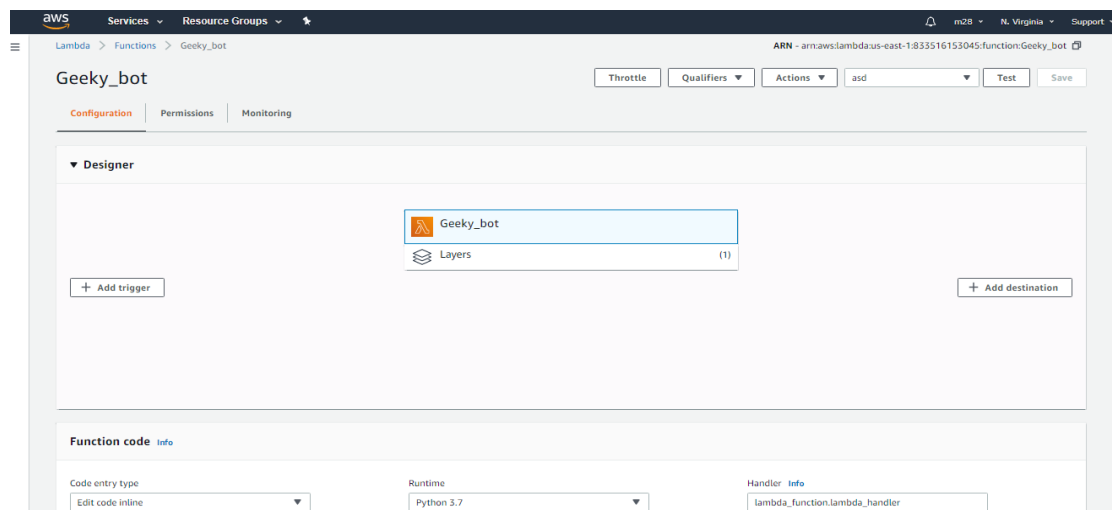
Amazon Lex provides the deep functionality and flexibility of natural language understanding and automatic speech recognition so that you can build highly engaging user experiences with lifelike, conversational interactions, and create new categories of products.



3. Amazon Lambda

AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume.

With Lambda, you can run code for virtually any type of application or backend service, all with zero administration. Just upload your system, and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services or call it directly from any web or mobile app.



4. Amazon CloudWatch

Amazon CloudWatch is a monitoring and observability service built for DevOps engineers, developers, site reliability engineers, and IT managers. CloudWatch provides you with data and actionable insights to monitor your applications, respond to system-wide performance changes, optimize resource utilization, and get a unified view of operational health.

CloudWatch
Dashboards
Alarms
ALARM
INSUFFICIENT
OK
Billing
Logs
Log groups
Insights
Metrics
Events
Rules
Event Buses
ServiceLens
Service Map
Traces
Container Insights
Resources
Performance Monitoring
Synthetics
Canaries
Contributor Insights
Settings

Update
Respond quickly with automated actions if your metrics hit thresholds or behave anomalously. [Learn more](#)

CloudWatch: Overview
All resources

1h 3h 12h 1d 3d 1w custom Actions

Alarms by AWS service

Services	Alarm	Insufficient	OK
CloudWatch Logs	-	-	-
Lambda	-	-	-
Lex Metrics	-	-	-
S3	-	-	-
Usage	-	-	-

Recent alarms View recent alarms dashboard

Recent alarms will appear here.
[Learn more about CloudWatch Alarms.](#)

Default dashboard
Name any CloudWatch dashboard CloudWatch-Default to display it here. [Create a new CloudWatch-Default dashboard](#)

VI] Lex – Messenger Integration:

Step 1

In this step, after Lex chatbot has been configured, there are few details like Name, following changes should be done in Channel setting such as name of channel, verification token, page access key, app secret key, generate callback URL.

The screenshot shows the AWS Lex console interface for configuring a chatbot named "BookTrip". The "Channels" tab is active, and the "Facebook" channel is selected. The form contains the following fields and values:

- Name:** BotFacebookAssociation
- Description:** Channel for associating Facebook
- IAM Role:** AWSServiceRoleForLexChannels (Automatically created on your behalf)
- KMS key:** aws/lex
- Alias:** Beta
- Verify token:** ExampleToken
- Page access token:** Page access token
- App secret key:** App secret key

An "Activate" button is located at the bottom of the form. A "Test Bot" button is visible in the bottom right corner.

Step 2

Next step is to create a developer account on facebook for developers website and creating an app. This app will act as channel for both facebook messenger and the lex application. The secret key that was generated from lex has to be used in this so that a connection is established.

facebook for developers
Docs
Tools
Support
My Apps
Search developer documentation

ClintonAWSApp
APP ID: 564364320858988
In development
View Analytics
Help

Dashboard
Settings
Basic
Advanced
Roles
Alerts
App Review
PRODUCTS
Webhooks
Messenger
Activity Log

App ID
564364320858988
App Secret
..... Show

Display Name
TellMeABook
Namespace

App Domains
Contact Email
royjck1993@gmail.com

Privacy Policy URL
Privacy policy for Login dialog and App Details
Terms of Service URL
Terms of Service for Login dialog and App Details

App Icon (1024 x 1024)
Category
Choose a Category
Find out more information about app categories here

Business Use
This app uses Facebook tools or data to
Discard
Save Changes

Step 3

A sample page should be created in the facebook app and that page has to be integrated in the developer account. Upon doing this, a token will be generated which should be used in the lex application to establish another connection.

facebook for developers
Docs
Tools
Support
My Apps
Search developer documentation

ClintonAWSApp
APP ID: 564364320858988
In development
View Analytics
Help

Dashboard
Settings
Roles
Alerts
App Review
PRODUCTS
Webhooks
Messenger
Settings
Activity Log

Pages
Tokens

SamplePage
107160777643847
Token Generated
Generate Token

Add or Remove Pages

Webhooks

To receive messages and other events sent by Messenger users, the app should enable webhooks integration.

Callback URL
https://channels.lex.us-east-1.amazonaws.com/facebook/webhook...
Verify Token
.....

Validation requests and Webhook notifications for this object will be sent to this URL.
Token that Facebook will echo back to you as part of callback URL verification.

Edit Callback URL
Show Recent Errors

12 | Page

Step 4

Using the messenger feature in the sample page that was created, the user can pass his queries and the corresponding responses can be obtained. This functionality has been made available only to the developer who created this page and not to the other users.



VIII] Recommendation Systems:

A recommendation system is a system which broadly recommends products to customers best suited to their tastes and traits. Today, the Internet is flooded with humongous amount of information. We humans cannot process this information at a time, and to avoid being overwhelmed we need some strategies to reduce the complexity of what we are perceiving.

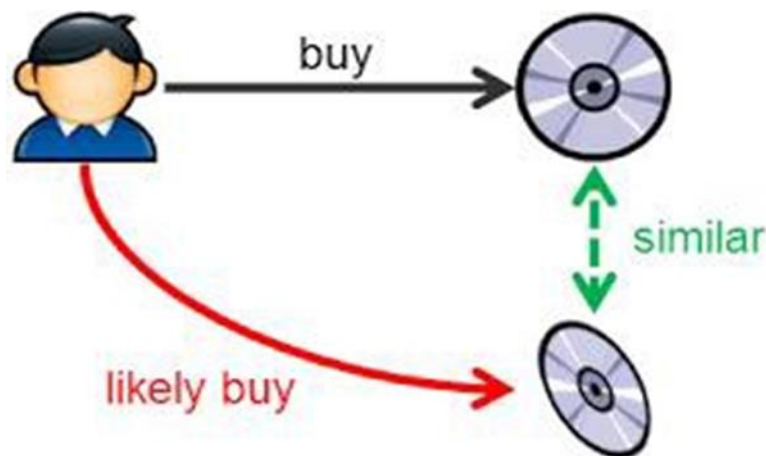
This is where recommendation systems come to the rescue. They discover data patterns in the data set by learning consumers choices and produce the outcomes that co-relates to their needs and interests.

Recommendation systems are helping us to explore more and more options which will pique our interest. Almost every major tech company has applied them in some form or the other: Amazon uses it to suggest products to customers, YouTube uses it to decide which video to play next on autoplay, and Facebook uses it to recommend pages to like and people to follow.

Types of Recommendation Systems

1) Content-based recommendation systems-

The idea behind content-based recommendation systems is if a person liked a particular book, he/she will also like similar book. This system makes use of the book metadata such as genre, book title, authors, book description, etc. New content can be easily recommended to the users using this system.



This system uses TF-IDF (Term Frequency- Inverse Document Frequency) and Cosine Similarity between space vector model techniques.

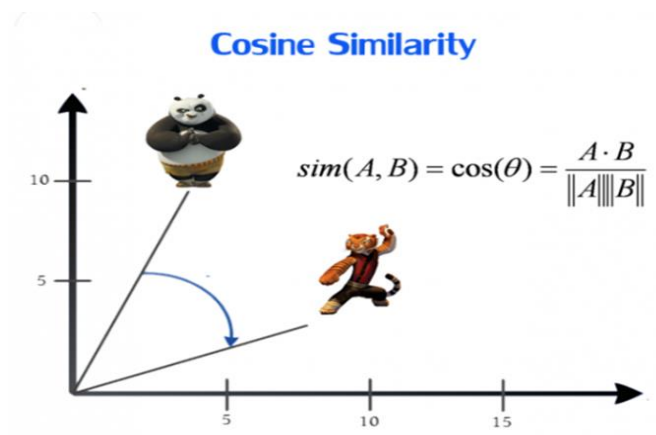
TF-IDF:

TF measures the frequency of a term in the document and IDF measures the overall importance of a given term.

Cosine Similarity:

It calculates the cosine of the angle between 2 vectors of books, vector A and vector B

It ranges from 0-1 where '1' means the 2 vectors are highly similar.



We have implemented 2 types of content-based recommendation systems.

1. Search term based system-

This system makes use of any keyword that a user provides and search for the books whose titles contain that particular keyword. If a user wants to read books related to a topic like “Yoga”, the books with the term “Yoga” in their titles will be suggested to that particular user.

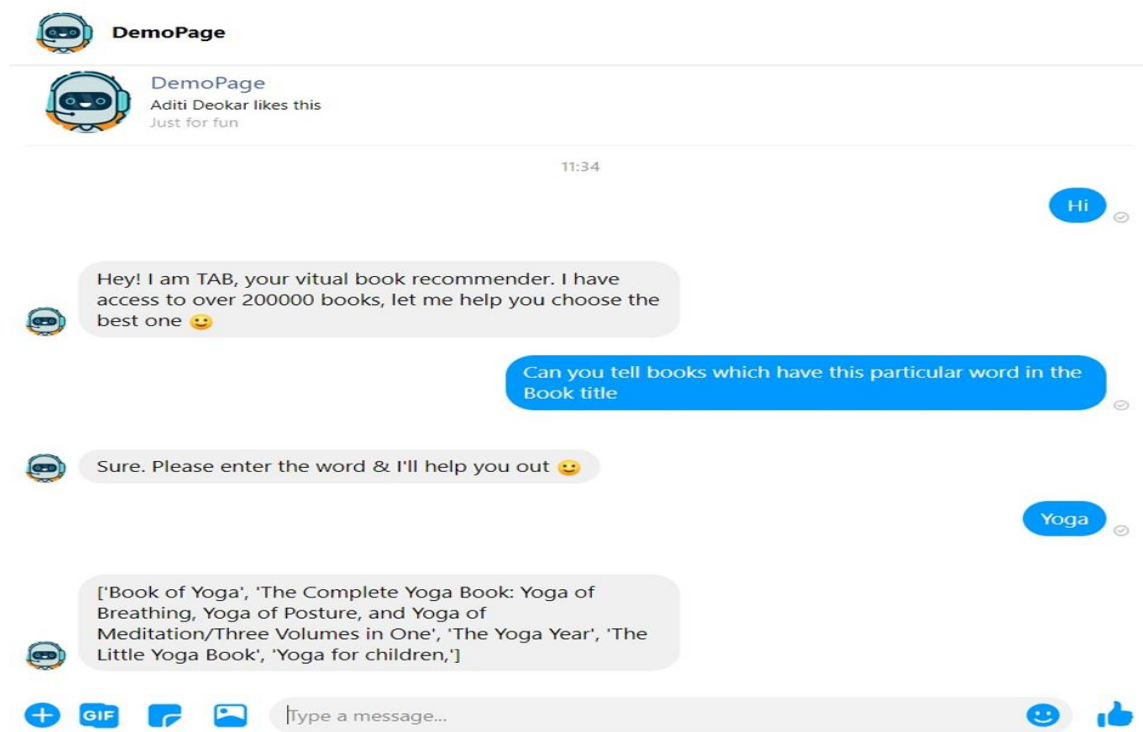
TF-IDF Vectorization-

```
1  ## TF-IDF vectorization
2  def tfidf_fit(docs):|
3      docs = [text_prepare(text) for text in docs]
4      tfidf_vectorizer = TfidfVectorizer(stop_words='english')
5      tfidf_fit = tfidf_vectorizer.fit(docs)
6      tfidf_matrix = tfidf_fit.transform(docs)
7      return tfidf_fit, tfidf_matrix
```

Calculating Cosine Similarity score and sorting the score in order to recommend the books with highest scores-

```
1 ## Function to search the books based on the search term
2 def keyword_based_search(keyword,list_index,tfidf_fit1,tfidf_matrix1):
3     cos_sim = cosine_similarity(tfidf_fit1.transform([text_prepare(keyword)]),
4                               tfidf_matrix1.tocsr()[list_index, :])
5     a = list(cos_sim[0])
6     book_idx = sorted(range(len(a)), key=lambda i: a[i], reverse=True)[:5]
7     return book_idx
```

Results-



2. Book title based system

In this system, the metadata of books is considered like genre, author name, book description and book title in order to recommend a book to a user. If the user provides the book title as “The Chronicles of Narnia”, the cosine similarity score between this book and other books is calculated and based on this score, similar books are recommended to the user.

Feature creation

```
1 # Create a new feature
2 def create_feature(x):
3     return ' '.join(x['book_genres_clean']) + ' ' + ' '.join(x['book_authors_clean']) + ' ' +
4             x['book_desc_clean'] + ' ' + ' '.join(x['book_title_clean'])
```

Vectorization using CountVectorizer

```
1 # Import CountVectorizer and create the count matrix
2 from sklearn.feature_extraction.text import CountVectorizer
3
4 count = CountVectorizer(stop_words='english')
5 count_matrix = count.fit_transform(books_limit['soup'])
```

Here, we are using CountVectorizer instead of TfidfVectorizer to convert the text data into vectors because we are interested in only the count of number of times a word appears in the document as we are considering metadata of the books. TfidfVectorizer considers the overall document weightage of a term which we do not need in this particular scenario.

Cosine Similarity Matrix

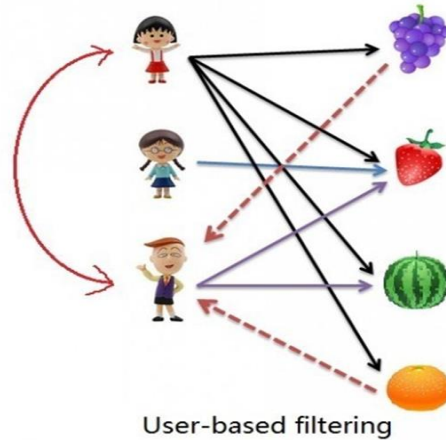
```
1 # Compute the Cosine Similarity matrix based on the count_matrix
2 from sklearn.metrics.pairwise import cosine_similarity
3
4 cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

Results-

```
1 get_recommendations('The Chronicles of Narnia', cosine_sim2)
481                                     The Last Battle
180                                     Alice in Wonderland
468                                     The Voyage of the Dawn Treader
261    The Guernsey Literary and Potato Peel Pie Society
648                                     A Brief History of Time
791                                     84, Charing Cross Road
146    The Lion, the Witch and the Wardrobe
96                                     Winnie-the-Pooh
119                                     The Shadow of the Wind
116    The Very Hungry Caterpillar
Name: book_title, dtype: object
```

2) User-Based Collaborative Filtering

These systems recommend products to a user that similar users have liked. For example, let's say A and B have similar tastes in books, and they have liked similar books before, so if a new book is launched and A has read and liked the book, it is highly likely that B will also like the book and hence is recommended to him.



Nearest Neighborhood approach

The standard method of Collaborative Filtering is known as Nearest Neighborhood algorithm. The process is to calculate the similarities between target user and all other users, select the top X similar users, and take the weighted average of ratings from these X users with similarities as weights. This gives the predicted rating of target user for all those items that have been rated by k similar neighbors, but target user has not.

Function to find K nearest neighbors

```
1 # K Similar users are found out given the user_id and ratings matrix
2 def findsimilarusers(user_id, ratings, metric='cosine', k=10):
3     similarities = []
4     indices = []
5     model_knn = NearestNeighbors(metric=metric, algorithm='brute')
6     model_knn.fit(ratings)
7     loc = ratings.index.get_loc(user_id)
8     distances, indices = model_knn.kneighbors(ratings.iloc[loc, :].values.reshape(1, -1),
9                                             n_neighbors=k + 1)
10    similarities = 1 - distances.flatten()
11
12    return similarities, indices
```

We took the fact into consideration that not all users have same baselines while giving the ratings. Some users are too generous and tend to give high ratings generally while some are pretty strict even though they are satisfied with the items. To avoid this bias, we have subtracted each user's average rating of all items when computing weighted average, and add it back for target user, shown in the below code.

Function to predict target user rating and avoid the bias in ratings given by other users

```

1 # This function predicts rating for specified user-item combination based on user-based approach
2 def predict_userbased(user_id, item_id, ratings, metric='cosine', k=10):
3     prediction = 0
4     user_loc = ratings.index.get_loc(user_id)
5     item_loc = ratings.columns.get_loc(item_id)
6     similarities, indices = findsimilarusers(user_id, ratings, metric, k) # similar users based on cosine similarity
7     mean_rating = ratings.iloc[user_loc, :].mean()
8     sum_wt = np.sum(similarities) - 1
9     product = 1
10    wtd_sum = 0
11
12    for i in range(0, len(indices.flatten())):
13        if indices.flatten()[i] == user_loc:
14            continue
15        else:
16            # the probable bias in the ratings is taken care of here
17            ratings_diff = ratings.iloc[indices.flatten()[i], item_loc] - np.mean(ratings.iloc[indices.flatten()[i], :])
18            product = ratings_diff * (similarities[i])
19            wtd_sum = wtd_sum + product
20
21    prediction = int(round(mean_rating + (wtd_sum / sum_wt)))
22    return prediction

```

Function to recommend the items to target user

```

1 def recommendItem(user_id, ratings, metric='cosine'):
2     if user_id not in ratings.index.values:
3         print ("User id should be a valid integer from this list :\n\n {} ".format(re.sub('[\[\]]',
4         |', np.array_str(ratings.index.values))))
5     else:
6         prediction=[]
7         for i in range(ratings.shape[1]):
8             if (ratings[str(ratings.columns[i])][user_id] !=0): #not rated already
9                 prediction.append(predict_userbased(user_id, str(ratings.columns[i]), ratings, metric))
10            else:
11                prediction.append(-1) #for already rated items
12        prediction = pd.Series(prediction)
13        prediction = prediction.sort_values(ascending=False)
14        recommended = prediction[:10]
15        book_indices=[recommended.index[i] for i in range(len(recommended))]
16        return list(books['bookTitle'].iloc[book_indices])

```

Recommendations-

```

1 recommendItem(219566, ratings_matrix_limit)

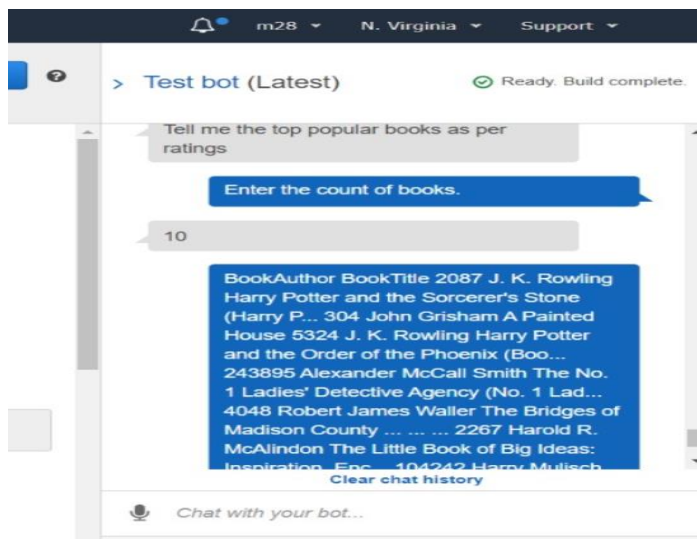
Predicted rating for user 219566 -> item 031205436X: 8
1. b'Riding the Bus with My Sister: A True Life Journey'
2. b'The Christmas Shoes'
3. b'FIRE IN BLOOD (Kangaroo Book)'
4. b'The Charm School'
5. b'The Glitter Dome'
6. b'Sunstrike'
7. b'Legacy Of Passion'
8. b'Masquerade'
9. b'The Other Side of Midnight'
10. b'Lords of Misrule'

```

3) Popularity Based Recommendation:

We have considered ratings as the criteria for popularity. Higher the ratings for a book. More popular it is. In our data source the books have been rated on a scale of 1 to 10. When the user asks the chatbot to recommend popular books it shows the top 10 books in descending order of their respective ratings.

Below is the screenshot of actual dialogue from messenger:



Below is the screenshot from Jupyter Notebook:

```
In [20]: number=input("Enter the number of top books you want: ")
topbooks(number)

Enter the number of top books you want: 10
The TOP 10 books as per ratings are:
```

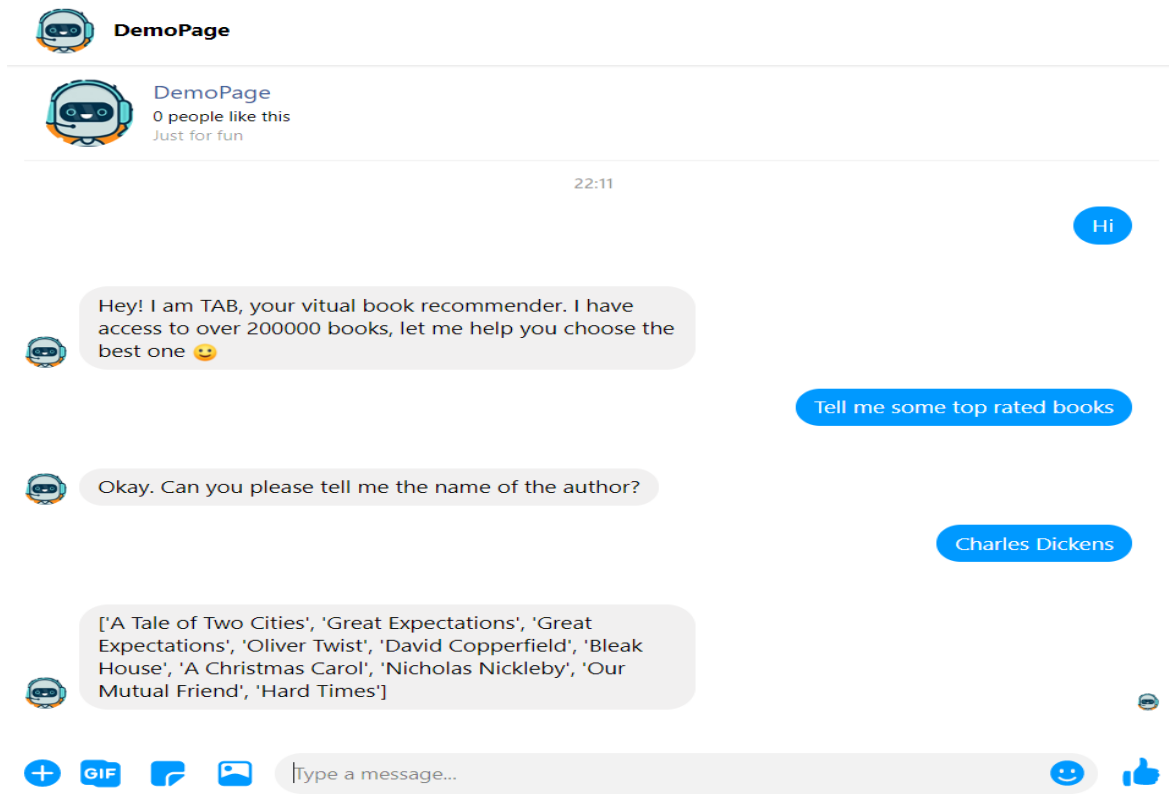
out[20]:

	BookAuthor	BookTitle
2142	J. K. Rowling	Harry Potter and the Sorcerer's Stone (Harry P...
305	John Grisham	A Painted House
5504	J. K. Rowling	Harry Potter and the Order of the Phoenix (Boo...
274418	Alexander McCall Smith	The No. 1 Ladies' Detective Agency (No. 1 Lad...
4178	Robert James Waller	The Bridges of Madison County
...
2328	Harold R. McAlindon	The Little Book of Big Ideas: Inspiration, Enc...
114855	Harry Mulisch	De compositie van de wereld
86521	Tian Dayton	It's My Life! A Workout for Your Mind
67817	Hanneke van Veen	Wie werde ich ein echter Geizhals? So knausern...
173554	Johann Nestroy	Der Unbedeutende

4) Recommendation Based on Authors:

As this chatbot is a search assistant along with a recommender, we added a functionality wherein the chatbot would show books as per the user's author of interest.

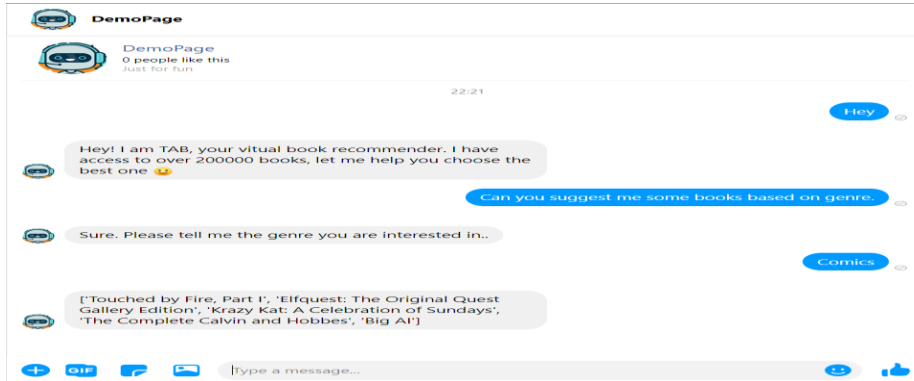
Below is the screenshot for this scenario where the user wants books written by Charles Dickens:



5) Recommendation Based on Genres:

There is another option to get book recommendations based on the genre mentioned in the input CSV.

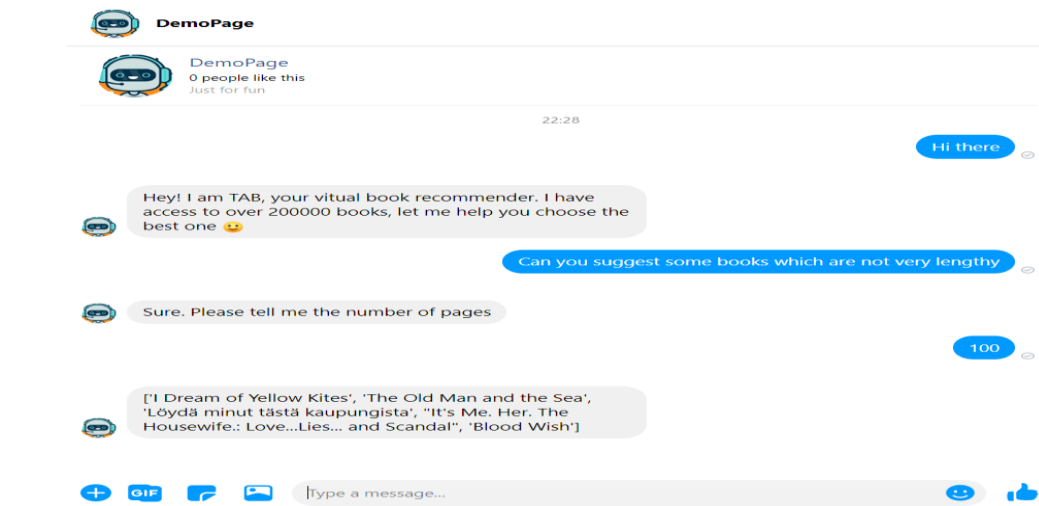
In this example, the user wants some book suggestions for the comic genre. We have restricted the number of books suggested to 5.



6) Recommendation Based on number of pages:

Some people prefer to read books which are not very long or some love to read big fat books. We have an option for both. Users can get suggestions as per the number of pages a book has.

Here, the user wants to read a short book i.e., 100 pages and gets recommended accordingly.



IX] Challenges and Solutions:

A. Data Cleaning and Stitching issues

1. Data cleaning was one of the significant problems which we solved using various techniques such as:-
2. NA values: The .csv files which contained the details such as book titles, author details, ratings, etc. were all having missing data. There were “NA” values in various fields that had to be removed as they would eventually create a problem while querying this data.
3. The missing values for ratings were replacing missing values with mean values.
4. Regular expression technique was used to remove special characters from book titles and replace them with no cost.
5. Duplicate values: The files also had duplicated values for various fields like book titles. These had to be removed as the recommendation system would behave inefficiently.
6. Data errors: The fields with book titles of different languages had the title fields which could not be deciphered. These values made no sense as they looked like encrypted text. Latin encoding was used while reading the CSV files to consider languages other than English.
7. Unnecessary data: Columns that were extravagant such as the image details column of Publication Company, were removed as they were not to be considered for recommendation.
8. The data stitching problem was faced as we had to find data from three different sources.
9. Integrating messenger and lex was a difficult task as Lex is an AWS service, and the messenger was not on AWS. Establishing a communication link between those components took a lot of trial and error.
10. Our input was massive data ~ 2, 80,000 rows; hence it took a lot of execution time during matrix calculation, which is a part of collaborative filtering.

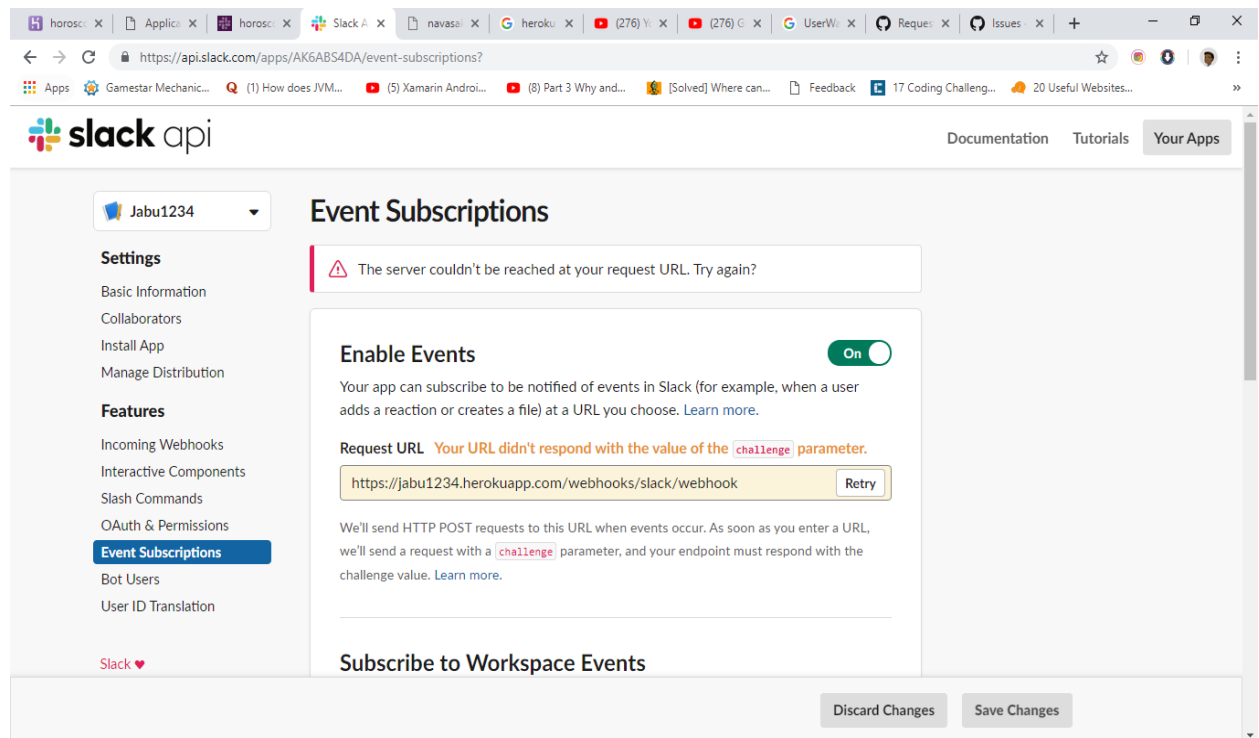
B. AWS Challenges

1. There were some challenges in the construction of lambda layers for our lambda function.
2. We had to package huge python libraries of sklearn, numpy, pandas, nltk. AWS has limitation that lambda layer can be at max of 250MB.
3. So we had to do some optimizations in our library packaging.
4. Also AWS has another limitation that creates a timeout of 30 secs for interaction between Lex and Lambda.
5. We had issues while running huge recommendation python codes on lambda that took more than 30 seconds for execution.
6. This 30 seconds timeout in AWS (limitation), which failed in query execution.
7. So as to solve this issue we increased the memory and the computing capacity of our lambda.
8. Term based book search was not giving desired results

C. Lex-Slack Integration Issues

1. We needed a medium through which the user can interact with the chatbot. AWS has the capability of integrating with a variety of services so that input from the user can be sent to the chatbot, and the corresponding responses can be obtained. AWS has channels like Slack, Facebook Messenger, Twilio, Kik services that can be used for this purpose. Our initial proposal was to use Slack as our platform for communication between the user and Chatbot service.
2. Slack is essentially a chat room for a company designed to replace email as your primary method of communication and sharing. Its workspaces allow in organizing contacts by channels for group discussions and allows for private messages to share information, files, and more all in one place.
3. With these advantages in mind, we integrated our chatbot application with Slack. Few steps that we did were building a slack application on the Slack

- API console, configuring the application to add interactive messaging to the bot. To associate the Slack application with the bot, we added a bot channel association in Amazon Lex. When the bot channel association is activated, Amazon Lex returns two URLs (Postback URL and OAuth URL).
4. After generating the postback URL and the OAuth URL, there were few more steps in the AWS console that should be done.
 5. After this, when we tried to test our slack channel, the messages weren't getting passed to the Lex console.
 6. A representation of the error we faced is in the screenshot below:



7. The reason for this issue was we needed a few extra services so that the user is able to interact with the chatbot application, which is available only for a premium account. Whenever a user has a premium account on Slack, on creating an application on Slack, challenge parameters will be provided.
8. These parameters should be used in the corresponding Lex application so that the application is able to pass the messages along with the parameters back to the Lex application.
9. Solution to this was as below:
10. Since there was the issue of premium membership with Slack, we decided to go for other options. Lex can work with other services like Messenger, Twilio, Kik, we decided to go with Facebook messenger. On researching we found that the Lex works seamlessly with Facebook messenger compared to other services. Messenger has handled uses with respect to callback URL and does not require challenge parameters which Slack required. Also, Messenger gave functionalities like adding multiple admins to the chatbot application, publishing the chatbot to known individuals for free that was available on Slack only for premium me.

X] Learning Outcome:

1. Understanding of recommendation systems concepts, working and types.
2. Integration of messenger and Lex
3. Creation and understanding of intents, slots, responses of Lex.
4. Writing Lambda Functions
5. Configuring data on S3
6. Retrieving results from S3
7. Overall experience of using and building application on important services of AWS like Lambda and S3.

XI] Business Prospects:

1. Currently, there is not a chatbot system in the Amazon bookstore or Kindle.
2. Users need to search for books/ authors by typing in the search bar. It is not an interactive experience.
3. Our chatbot can be integrated with the current system, and it will be beneficial for any user who wants to read.
4. He/She can also get book suggestions as per their search history or according to the books read by similar users.