



# INFO7250 ENGINEERING BIG DATA SYSTEMS: TEAM 4

Final Project Report

## **Authors**

Fredy Daruwala, Gautam Pawar, Nikita Anand, Tapadyuti Maiti

Date: 24<sup>th</sup> April 2017

Version 1.0

## **Table of Contents**

- 1. Team Members**
- 2. Problem Statement/Objective**
- 3. Problem Approach**
- 4. Application Used**
- 5. Contributions**
- 6. Research & Findings**
- 7. Implementation Details**
- 8. Issues Faced & Their Resolutions**
- 9. Conclusion/Results**
- 10. Future Scope**

**Our Team: -**

Fredy Daruwala  
Gautam Pawar  
Nikita Anand  
Tapadyuti Maiti

**Problem Statement/Objectives: -**

Online reviews play a very important role in information dissemination and play a major role in influencing a user's decision while buying a product. However, a user may only read a limited number of reviews before deciding to purchase an item. An important aspect to the success of a rating and reviews portal for amazon is to identify which reviews to promote as being useful. The main goal of our project is to analyze the Amazon Books Dataset based on the user's ratings and reviews and **host the results on our web application for a publisher** to analyze and determine the usefulness of a review. We are also performing an NLP sentiment analysis on the reviews to find out if a review has been incentivized or not. Sellers incentives for reviews aren't necessarily asking for positive reviews, but in general, they aren't handing out products that they suspect you'll hate. By law, these incentivized reviews must also be disclosed to the public to avoid dishonest sales practices.

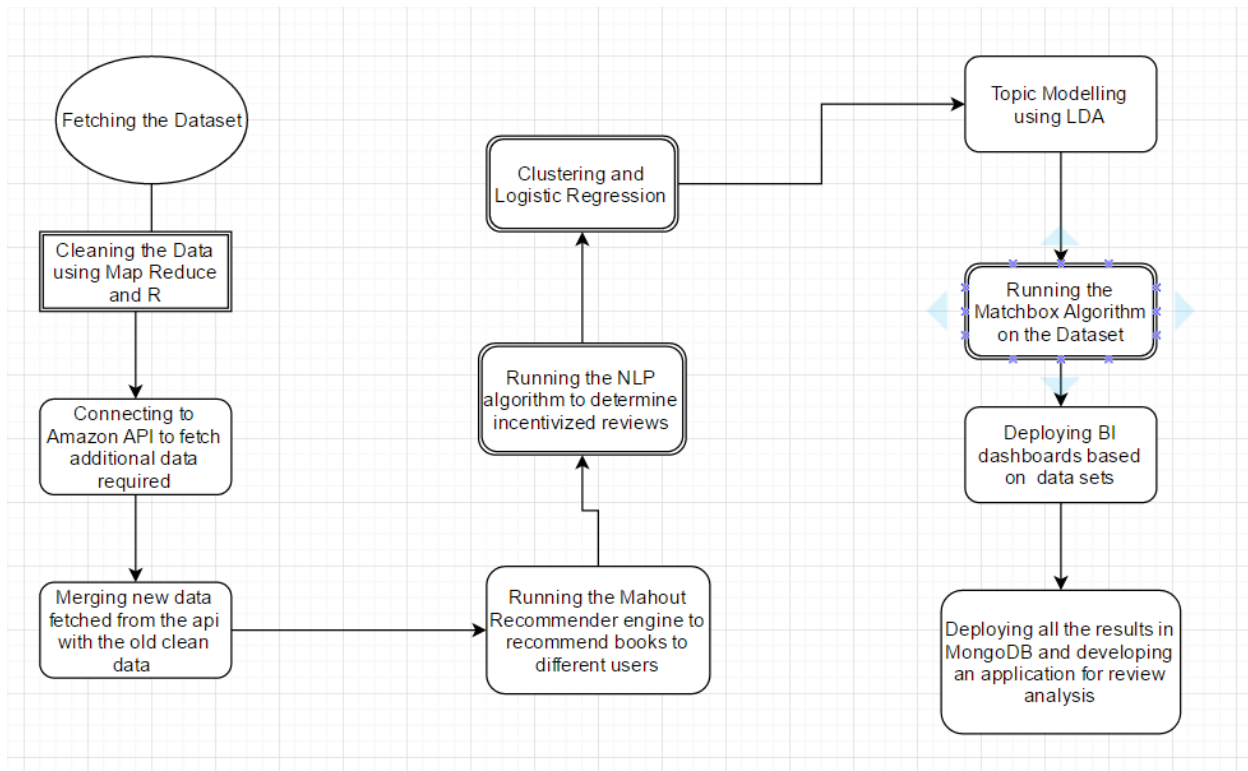
For publishers to improve their book sales they need tools to carefully analyze the review data and study market behavior based on the reviews and the ratings users provide. This has made it essential to have enough tools so that the users and the publishers can easily analyze the data and derive meaningful results.

**Problem Approach: -**

Our main dataset file was an Amazon Books review dataset file which was a total size of 9 GB and is in the json format. The main challenge with this dataset was that since it was in the **Json format** we had to clean the data set and parse it to extract required information for analysis using Map and Reduce framework. It is not easy to parse such a huge data set file using normal Java, Python or R programs as it would take a large amount of time and Python would not be able to parse such a huge data set. We used 3 **Map and Reduce programs** to parse the dataset. Later we used the **Amazon Product Advertisement API** to fetch missing data in the Json file such as title, book author and genre of the books. We did this using the custom uri generated from Amazon to hit the api using a map-reduce program and fetch the additional data. After obtaining the additional data, we merged this data with the cleaned Json data **using R scripts**. Once we had cleaned and

merged all the data, we hosted the data on mongodb and performed different recommendations and analysis on the same.

## RoadMap: -



## Application Use Cases: -

- Analyze the data from the Amazon Books Review dataset and the Amazon Product advertisement API.
- Displaying the top-rated books in the data set using Mongodb Queries.
- Analyze the dataset for recommendations based on user, books and their ratings.
- Show data visualizations for various categories of books using the Tableau server.
- Using NLP algorithms to predict whether a user review is incentivized or non-incentivized.

The application we built is a Spring MVC application with jsp pages as the front end and MongoDB database server as the backend. We staged all the clean data in a Mongodb table and interfaced it with our application. We also interfaced our application with the Amazon Product Advertisement API to fetch live data about the books from the Amazon Server.

## Dataset Characteristics: -

Our dataset was the Amazon Books Review dataset. We obtained the dataset from the below URL: -

<https://snap.stanford.edu/data/web-Amazon.html>

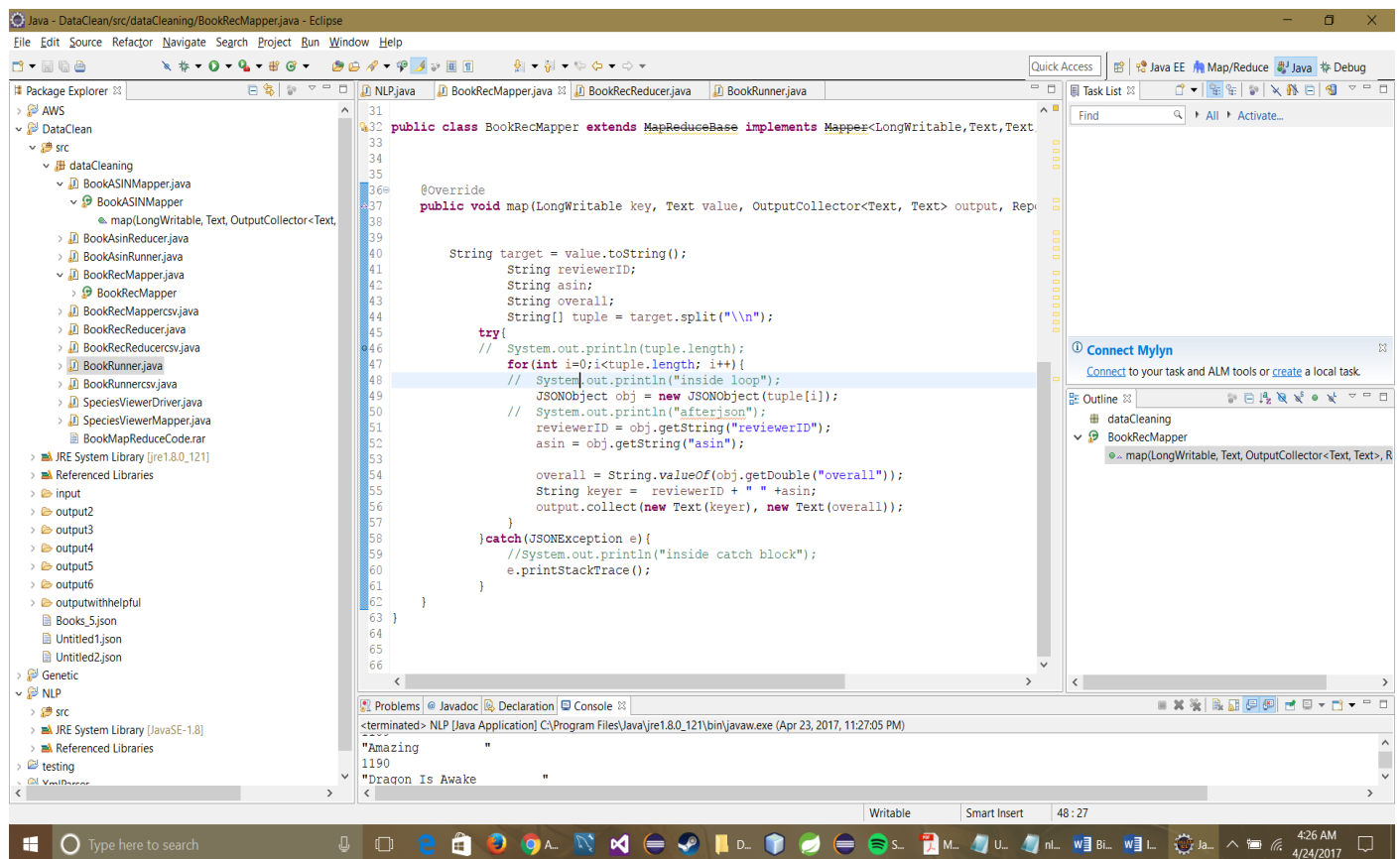
This dataset is a json file containing 9 million reviews and has the following characteristics: -

```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the
piano. He is having a wonderful time playing these old hymns.
The music is at times hard to read because we think the book
was published for singing from more than playing from. Great
purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```

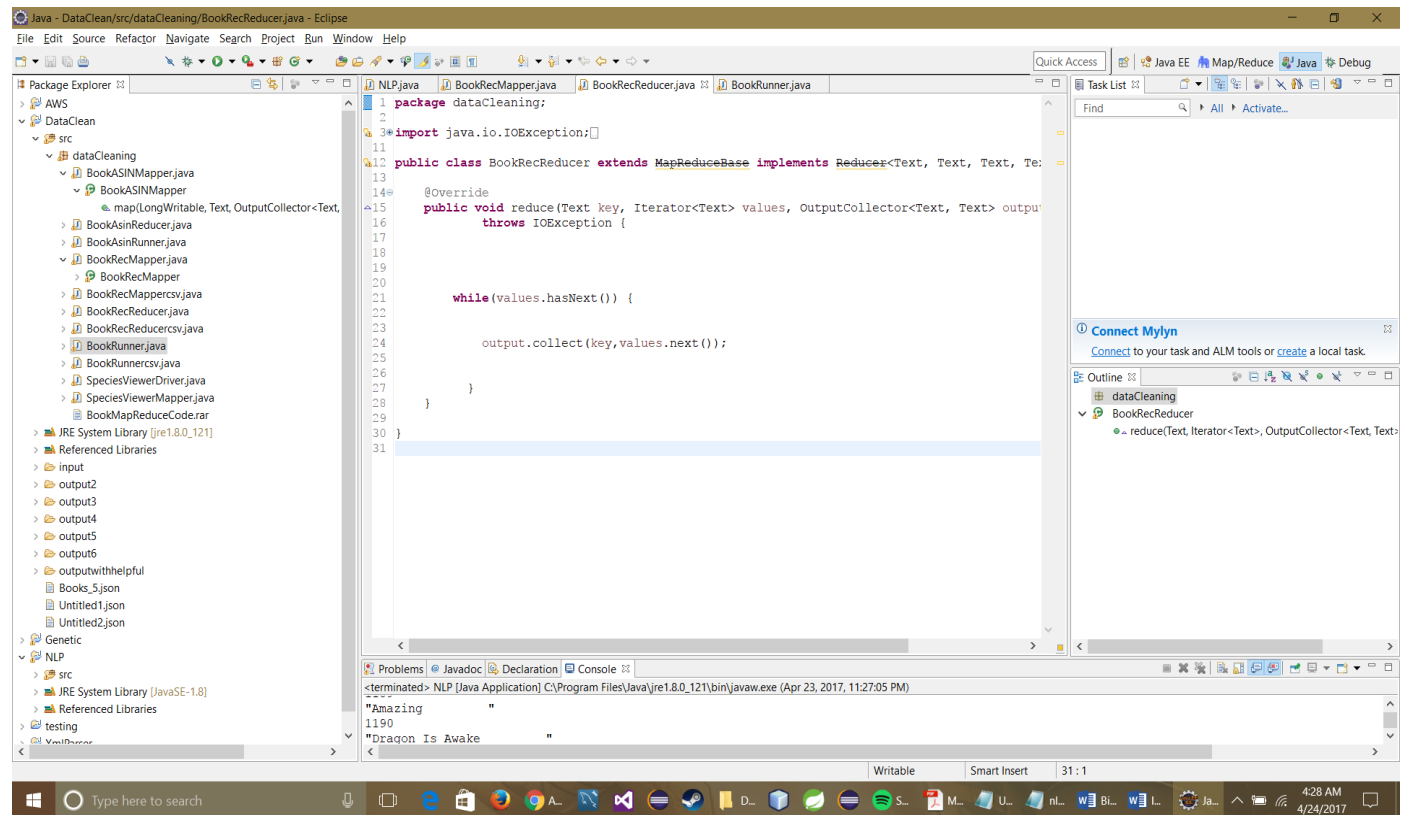
We have used all the above characteristics of the data set for our analysis. There 8898041

### **Data Cleaning:** -

Using the power of Map Reduce we could clean the data. We ran the Map and Reduce algorithms to first convert all the JSON data into rows and columns format. We used the org.json.\* library in java to parse the JSON file using Map-Reduce algorithm. We are fetching all the values from the JSON object using the obj.getString().



## **BookRecReducer:** -

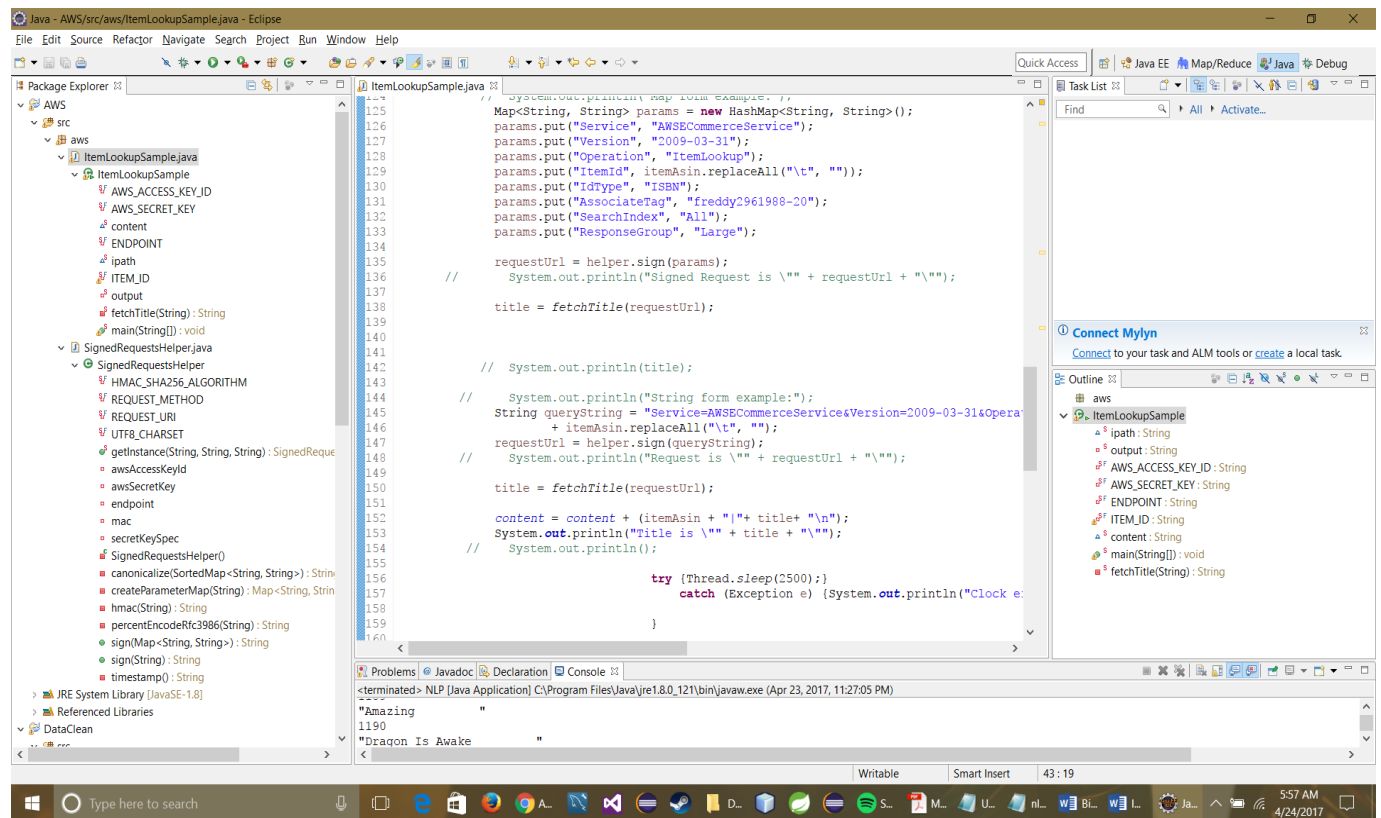


We used a series of mappers and reducers to obtain the final cleaned data. This resulted in the generation of a csv file which contained all the data from the parsed json file.

## **Fetching Data from the Amazon Product API:** -

This was one of our most challenging tasks in the project. This task took a long time as it was not easy to fetch data from the API. The data provided in the Amazon books dataset was not complete as it lacked the book title, book author and the genre of the book. We wrote a program in java which used the unique ASIN ID of the books in our data set and fetched the values from the amazon product API service. We iterated through the entire ASIN data set and fetched all the values.

The main reason as to why the task was challenging is because we could not fetch an API call more than once in a second and running a for loop was doing exactly that. After researching we realized that we had to use the thread function to accomplish the same. The normal for loop was hitting the api more than 13 times a second and hence we were getting error 503.



## Mahout Recommendation: -

We also derived the input file in the form of Reviewer ID, ASIN ID and Rating to run on Mahout. We derived this by running a Map Reduce implementation on the entire data set. Once we got all the data we ran a Mahout recommendation command: -

**bin/Hadoopjarmahout-core-0.7-job.jar**

**org.apache.mahout.cf.taste.hadoop.item.RecommenderJob -s SIMILARITY\_COOCCURRENCE --input ./genrerecommender --output ./outputgenrerecommender**

This provided us with a recommendation for all the user IDs. We obtained 10 recommendations for each user. The main challenge we faced while running the Mahout Recommendation engine was that the Reviewer ID and the ASIN ID were in the Alphanumeric format and Mahout does not accept alphanumeric values. To do that we had to generate numeric IDs for each alphanumeric ID in the dataset. Below is the output file screenshot for the Mahout job: -



```

167 1228 [203625:5.0, 216598:5.0, 208770:5.0, 210157:5.0, 208216:5.0, 210504:5.0, 201616:5.0, 204399:5.0, 201055:5.0, 208295:5.0]
168 1229 [201833:5.0, 209867:5.0, 209865:5.0, 208822:5.0, 209857:5.0, 202582:5.0, 202581:5.0, 202580:5.0, 208813:5.0, 209850:5.0]
169 1231 [200196:4.9333334, 208199:4.857143, 201831:4.857143, 215643:4.5897436, 215660:4.5660377, 204866:4.5510206, 200254:4.5471697, 211871:4.537037, 216594:4.512397, 208468:4.5079365]
170 1232 [212458:4.0, 214004:4.0, 208712:4.0, 208398:4.0, 208422:4.0, 203279:4.0, 208490:4.0, 201370:4.0, 216837:4.0, 203280:4.0]
171 1233 [201632:5.0, 201629:5.0, 202737:5.0, 216660:5.0, 201619:5.0, 201618:5.0, 201617:5.0, 201616:5.0, 203286:5.0, 207741:5.0]
172 1234 [202895:5.0, 216660:5.0, 208492:5.0, 211240:5.0, 202029:5.0, 205469:5.0, 202566:5.0, 208483:5.0, 201086:5.0, 211220:5.0]
173 1235 [202040:5.0, 208712:5.0, 201243:5.0, 215660:5.0, 208444:5.0, 210421:5.0, 211288:5.0, 200523:5.0, 208535:5.0, 201397:5.0]
174 1236 [210819:5.0, 213141:5.0, 208648:5.0, 213171:5.0, 212059:5.0, 213850:5.0, 209601:5.0, 214531:5.0, 208649:5.0, 202197:5.0]
175 1238 [201831:5.0, 208966:5.0, 201735:5.0, 209386:5.0, 204484:5.0, 200798:5.0, 208482:5.0, 201791:5.0, 200523:5.0, 209031:5.0]
176 1239 [212939:5.0, 203476:5.0, 200237:5.0, 208126:5.0, 202706:5.0, 201392:5.0, 200859:5.0, 216660:5.0, 208813:5.0, 201353:5.0]
177 1240 [214020:5.0, 214030:5.0, 214032:5.0, 214028:5.0, 214027:5.0, 214029:5.0, 200517:5.0, 214023:5.0, 214022:5.0, 214021:5.0]
178 1242 [209764:5.0, 201434:5.0, 201435:5.0, 213150:5.0, 200358:5.0, 201511:5.0, 202566:5.0, 207188:5.0, 208201:5.0, 200483:5.0]
179 1243 [212336:5.0, 208201:5.0, 203323:5.0, 202632:5.0]
180 1244 [201632:5.0, 211651:5.0, 202737:5.0, 216660:5.0, 201618:5.0, 201617:5.0, 216655:5.0, 203286:5.0, 201055:5.0, 203280:5.0]
181 1245 [209638:4.0, 201804:4.0, 208199:4.0, 212336:4.0, 203322:3.985444, 201509:3.9597316, 201757:3.9299364, 202515:3.9298246, 216804:3.9268293, 212059:3.9242425]
182 1247 [204537:5.0, 201509:5.0, 202038:5.0, 202013:5.0, 210226:5.0, 211995:5.0, 209386:5.0, 202515:5.0, 203452:5.0]
183 1248 [204537:4.0, 201243:4.0, 200283:4.0, 202040:4.0, 202038:4.0, 200798:4.0, 210226:4.0, 209386:4.0, 200067:4.0, 211973:4.0]
184 1249 [211520:5.0, 203183:5.0, 201987:5.0, 209367:5.0, 202643:5.0, 207188:5.0, 208792:5.0, 203278:5.0, 205477:5.0, 203242:5.0]
185 1250 [208199:4.7647057, 208159:4.724138, 208813:4.6666665, 213923:4.637931, 213850:4.625, 209627:4.617021, 201509:4.605263, 216832:4.5833335, 216804:4.5238094, 208649:4.464286]
186 1251 [202632:5.0, 207354:5.0, 204484:5.0, 203048:4.9, 212830:4.8, 209709:4.8, 203483:4.8, 200872:4.8, 203065:4.7777777, 209341:4.75]
187 1252 [204484:5.0, 201726:5.0, 202029:5.0, 200294:5.0, 202038:5.0, 202081:5.0, 212652:5.0, 202040:5.0, 209935:5.0, 203210:5.0]
188 1253 [213118:5.0, 212044:5.0, 201805:5.0, 204915:5.0, 202965:5.0, 212622:5.0, 200364:5.0, 203194:5.0, 211043:5.0, 209336:5.0]
189 1255 [203323:5.0, 201802:5.0]
190 1256 [204537:5.0, 201791:5.0, 208490:5.0, 208201:5.0, 200798:5.0, 200523:5.0, 201745:5.0, 204484:5.0, 210226:5.0, 208979:5.0]
191 1257 [204537:4.0, 211520:4.0, 212336:4.0, 201509:4.0, 202040:4.0, 202038:4.0, 210226:4.0, 211995:4.0, 200067:4.0, 202515:4.0]
192 1258 [202038:4.5, 204842:4.5]
193 1259 [203163:5.0, 213556:5.0, 206156:5.0, 201760:5.0, 201619:5.0, 205031:5.0, 205006:5.0, 200879:5.0, 201424:5.0, 201826:5.0]
194 1261 [205204:5.0, 206156:5.0, 210504:5.0]
195 1263 [202038:5.0, 201760:5.0, 210504:5.0, 208535:5.0, 209614:5.0]

```

In the above screenshot 1228 is the user ID and 203625 is the book ID recommendation for the user.

### Calculation of Incentivized and Non-Incentivized reviews using Stanford NLP algorithm: -

The goal of this activity is to find out if a review is incentivized or not. A lot of suppliers these days pay their reviewers incentives to give them a good rating and a good review. The way we are doing this is by considering the review column and the summary column in the dataset and assigning a combined score to them. Then we calculate the difference between the actual rating and the calculated rating. If the difference is greater than 2 then the rating is an incentivized rating else the rating is non-incentivized.

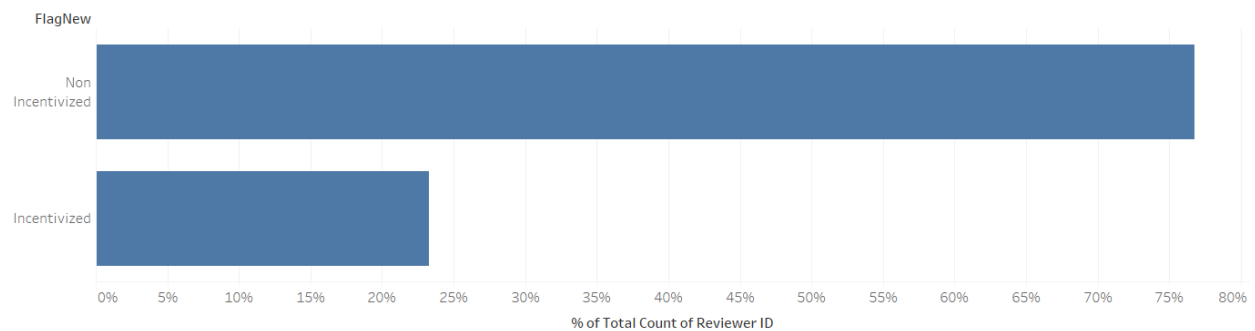
### **Incentivized Reviews based on Text Analytics:**

After performing calculation of Incentivized and Non Incentivized reviews on Stanford NLP algorithm, we have got the data of both kind of reviews. We will perform some interesting visualization on it. Incentivized reviews explains the bias towards the review user shows by

writing words like “**Honest**” and “**Unbiased**” and rate those books highly to avail discounts on them.

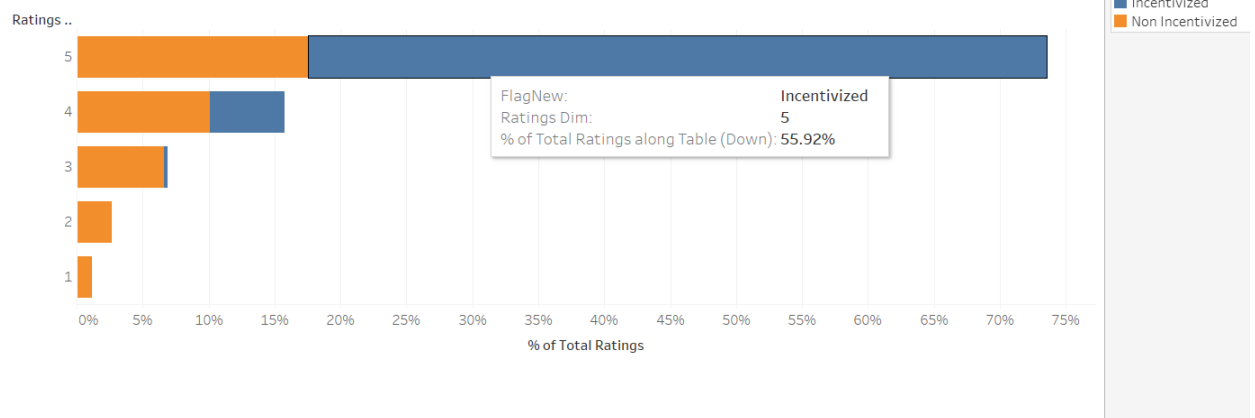
As depicted below, over 25% of the reviews were **incentivized** while rest 75 were non incentivized.

Count of Incentivized and Non Incentivized Ratings

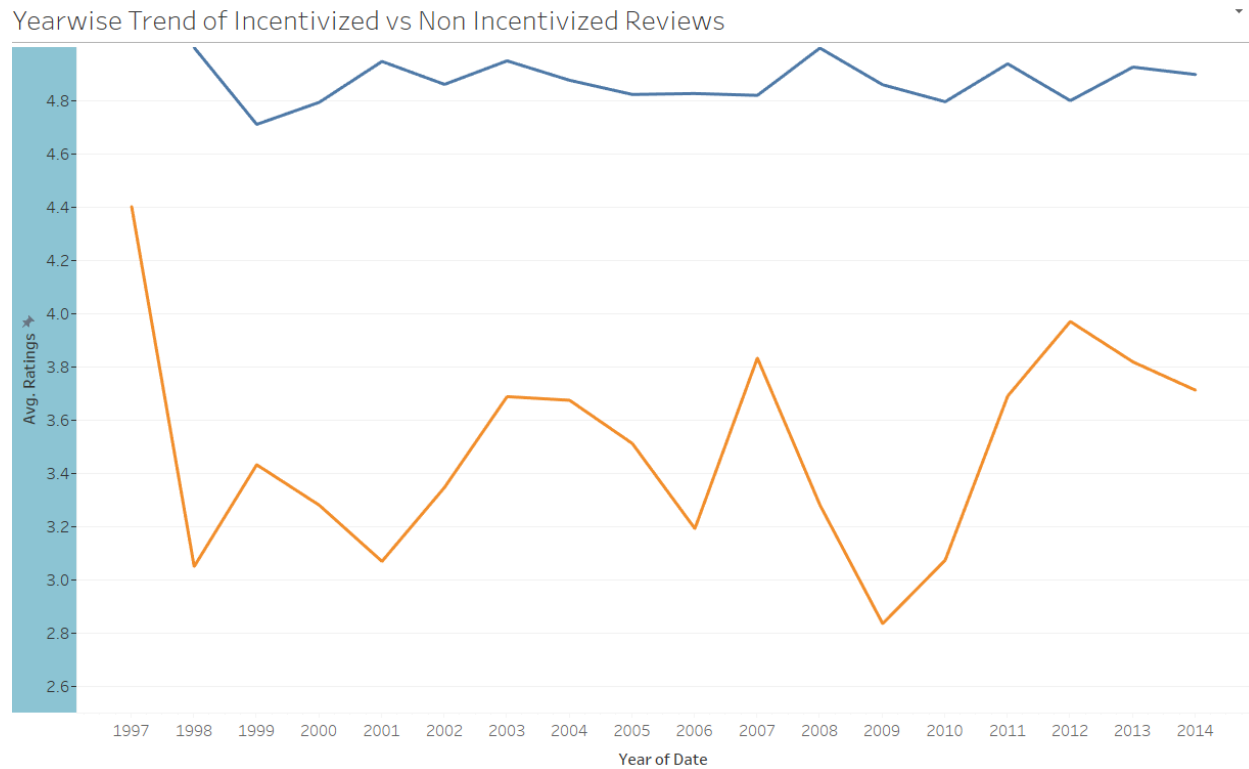


It is clearly observed that most of the non- incentivized reviews pattern cover 55% of the 5-start reviews. i.e. users who tends to give incentivized reviews have given the 5 rating the most as compared to other ratings where percentage of non-incentivized reviews are very less.

Incentivized vs Non Incentivized Reviews

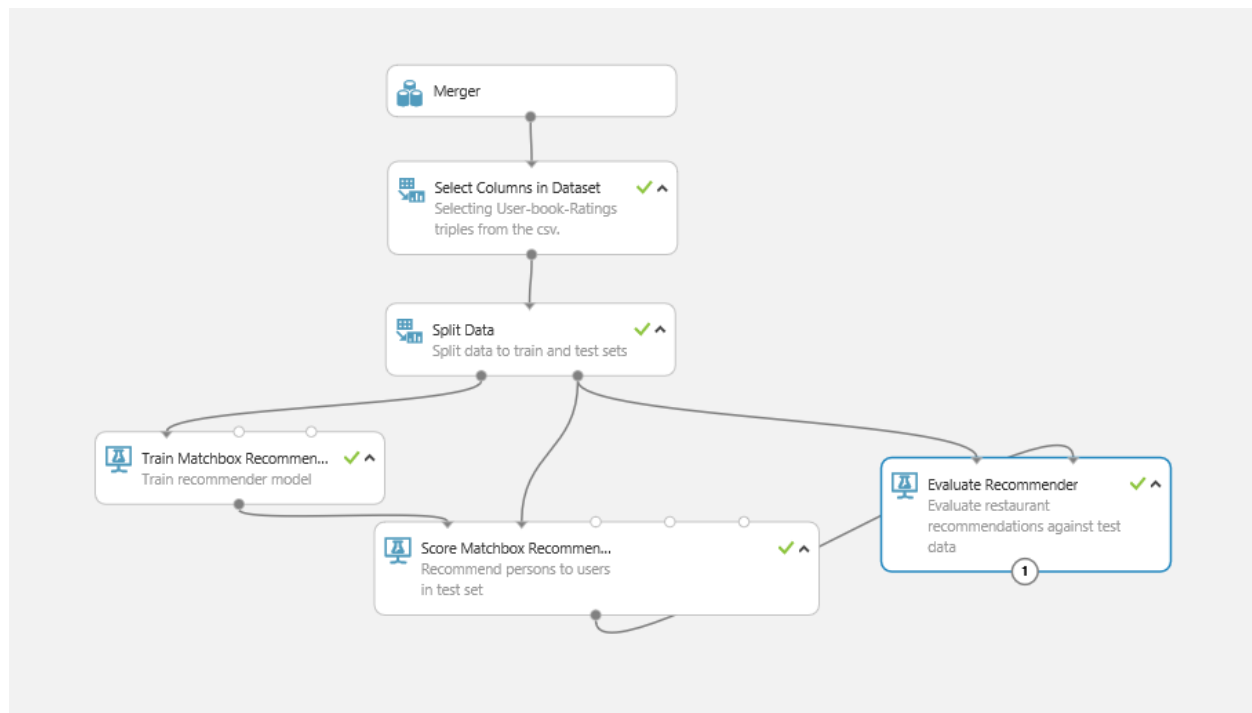


Year wise trend depicts how **average incentivized** reviews range between **4.5 to 5** whereas the average non-incentivized reviews range between **4.5 and 2.8**.



### Person Recommendation Using MatchBox Recommendation:

Whenever user visits the homepage, we will be recommending him the related person based on the user's preferences. Those related users will be suggesting the books based on their match with the user. We are using Matchbox Recommendation algorithm that combines collaborative filtering with a content-based approach. It is therefore considered a **hybrid recommender**. It expects user-book-rating triples as input to train the model and outputs the related users based on the ratings provided by them to different books.

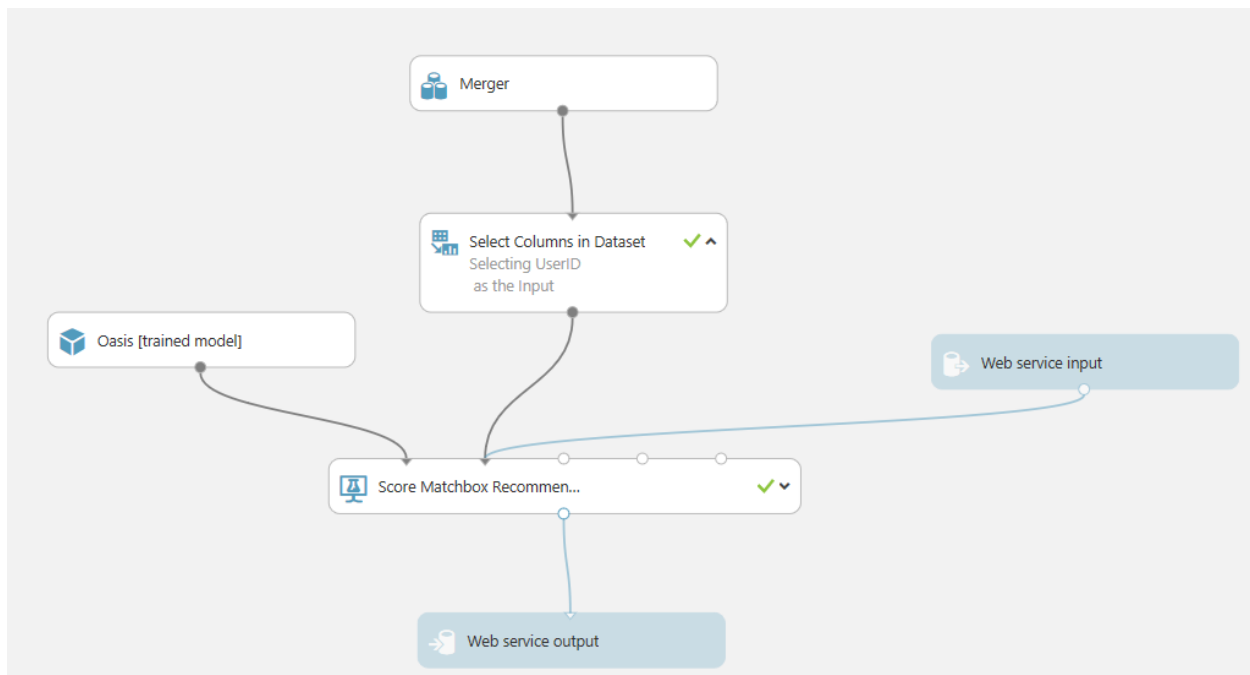


L1 Sim NDCG	L2 Sim NDCG
0.817591	0.803753

**Evaluate Recommender** computes the average normalized discounted cumulative gain (**NDCG**), based on Manhattan (**L1 Sim NDCG**) and Euclidean (**L2 Sim NDCG**) distances, and returns both values in the output dataset. In our case, both the values are 0.81 and 0.80 that shows a good performance by the model.

### Deploying the Web Service using the Trained Recommended Model:

Once we have trained the model, we will deploy it using the web service.



**Output (Related Users up to 5):** Below is the output of the user ID with the recommended users based on the book rating.

User	Related User 1	Related User 2	Related User 3	Related User 4	Related User 5
A11TYILTAFKPR3	A2INDDW3XYFFV1	A1JAG4YE0MXCKE	A370Z6I5GBWU44	AMKZHBOK7VMQR	A1K1JW1C5CUSUZ
A12W8NRSYR593I	A15ACUAJEJXCS3	AMRZ5G7HF7I03			
A133S8CUIVRFKN	A59LBV682DWGM	A188JKV8QVVOT7	AMALQ15DP0YDJ	A3ASXNNWLR4J4I	AI9AFMD6OVGUG
A1340OFLZBW5NG	AHD101501WCN1	ABFOAYZA2UHD3	A308E1C7MU3462		
A157UENZPTI1TD	A1DYXCF4148PJT	A2GPEV42IO41CI			
A15ACUAJEJXCS3	AJQ1S39GZBKUG	A38AAPXSJN4C5G	A2SHQJP6PNQTLTD	A2W6WXEUAVM3E	A12W8NRSYR593I
A1BC4GPH9LBSNR	A2QZQBINBG6B5N	A3R8PXSFGY9MC2	AS5ERWDSXRDNX	AZXGPM8EKSHE9	A34UTL4AVX80MK
A1BM81XB4QHOA3	A1K1JW1C5CUSUZ	A3W43PSHRIG8KV	A1NPNGBWBD9AK3	AHD101501WCN1	A2NHD7LUXVGTD3
A1C9NH907F3RHE	AK1C761G3YD0J	A545U4UQEPT1J	A3H9YD6K9TVKDP	A281NPSIMI1C2R	A1K1JW1C5CUSUZ
A1DW9QEARBGRFO	A281NPSIMI1C2R	A2PH70X2FVDDGH	A3MFKQXPA7PERW	A3UJJGY799F76I	AZTK9T1ECFSX1
A1DYXCF4148PJT	A34K6MOV1NC10E	A1IU7S4HCK1XK0	A151O3YT8NS68A	A222LQEP7O7BV	A2NNPISXOK1K4

Below are the list of suggested users if we enter a user id.

Default Endpoint

API HELP PAGE TEST APPS

REQUEST/RESPONSE Test Test preview

BATCH EXECUTION Test preview

Excel 2013 or later Excel 2010 or earlier workbook

Excel 2013 or later workbook

✓ 'Recommender: Book Recommendations [Predictive Exp.]' test returned ["A1BC4GPH9LBSNR","A1QK7T40HIQF4S","ARDQ9KNB8K22N","A372UKGN0YXF6L","A1V0XL4QZDKM9G","AI9CMLQWH9KRP"]...

## Machine Learning Algorithms: -

**Using K-Means clustering** to predict the helpfulness of a review:

We have analyzed the structure of text the book reviews provided by the amazon users and predicted the how the reviews posted by the users effect the helpfulness of the product.

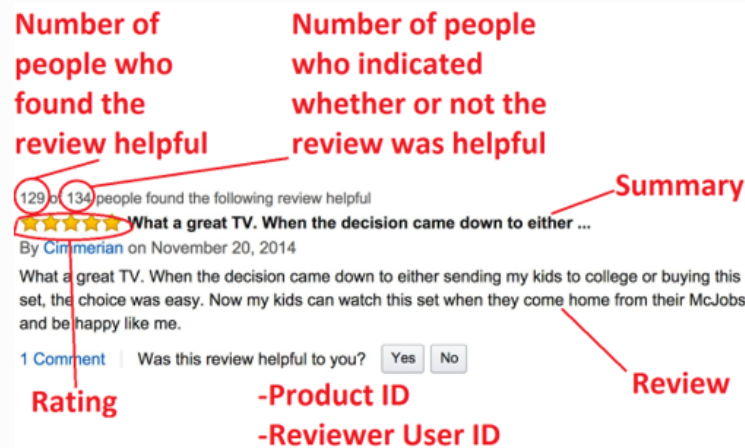
Language and Services used: **Python and Apache Spark 1.6**

Terms Used:

**Helpfulness Numerator:** Number of people who found the review helpful.

**Helpfulness Denominator:** Number of people who indicated whether review was helpful or not.

**Helpfulness** = 1 if Helpfulness Numerator/ Helpfulness Denominator >0.5 else 0



### Dataset:

### Exploratory Analysis:

We have done feature selection and will focus on review related data and the related columns as required.

	HelpfulNumerator	HelpfulDenominator	Review_Text	Ratings
0	0	0	Spiritually and mentally inspiring! A book tha...	5.0
1	4	4	When Gibran was first introduced to me I had ...	5.0
2	0	0	As you read Gibran's poetry brings spiritual ...	5.0
3	0	0	Jubran Kahlil Jubran had a teacher make a mist...	5.0
4	0	0	When I first started writing poetry at age 12 ...	5.0

```
#include reviews that have more than 15 helpfulness data point only
df1 = df1[(df1.HelpfulDenominator > 15)]
```

```
#converting the book reviews to lower case
df1.loc[:, 'Review_Text'] = df1['Review_Text'].str.lower()
df1["Review_Text"].head(10)
```

```
9      this man was a son of a pastor  but worshipped...
35     this is one of the first (literary) books i re...
40     certainly the words are of kahlil gibran  but ...
42     this book was given to me as a gift before i j...
49     gibran gets right down to the bedrock of what ...
59     the prophet is almustafa  called ""the chosen ...
```

**Creating the Helpfulness Column(Derived):**

```
#transform Helpfulness into a binary variable with 0.50 ratio
df1.loc[:, 'Helpfulness'] = np.where(df1.loc[:, 'HelpfulNumerator'] / df1.loc[:, 'HelpfulDenominator'] > 0.50, 1, 0)
df1.head(3)
```

	HelpfulNumerator	HelpfulDenominator	Review_Text	Ratings	Helpfulness
9	0	27	This man was a son of a pastor but worshipped...	1.0	0
35	81	92	This is one of the first (literary) books I re...	5.0	1
42	19	25	This book was given to me as a gift before I j...	5.0	1

**Summary Measures and Correlation Matrix:**

```
df1.groupby('Helpfulness').count()
```

	HelpfulNumerator	HelpfulDenominator	Review_Text	Ratings
Helpfulness				
0	10926	10926	10926	10926
1	35981	35981	35981	35981

```
df1.corr()
```

	HelpfulNumerator	HelpfulDenominator	Ratings	Helpfulness
HelpfulNumerator	1.000000	0.968204	0.085496	0.155268
HelpfulDenominator	0.968204	1.000000	-0.009486	0.031488
Ratings	0.085496	-0.009486	1.000000	0.449498
Helpfulness	0.155268	0.031488	0.449498	1.000000

**Bag of Words Model:** Using Bag of Words model, we got less accuracy, so we went for Logistic Regression classification modeling.

```
# ROC/AUC score
#Accuracy of bag of Words Model is 75 Percent
y_score = probas
test2 = np.array(list(df2.Helpfulness))
y_true = test2
roc_auc_score(y_true, y_score[:,1].T)

0.75004537957503881
```

**Used K-Means Clustering** and created 5 clusters for the review words. Selecting top 15 words.



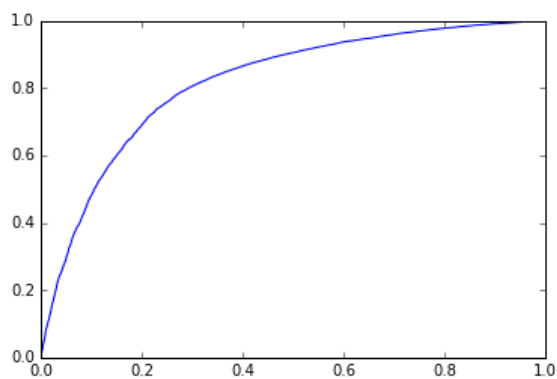
```
[ 'quot',
  'new',
  'characters',
  'reading',
  'like',
  'good',
  'books',
  'character',
  'love',
  'life',
  'history',
  'time',
  'work',
  'really',
  'don',
  'american',
  'read',
```

Cluster wise Summary:

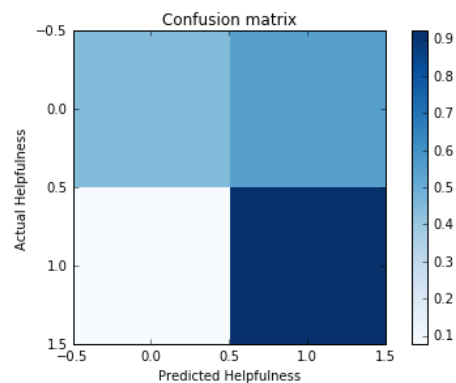
ROC Curve, Confusion Matrix and accuracy using Logistic Regression:

```
roc_auc_score(y_true, y_score[:,1].T)

0.81728574200257609
```



```
[ [ 4898  6028]
  [ 2808 33173]]
```



**Result:** Below are the top words that effects helpfulness of a review. Plot, Story , Novel, Character etc words with high coefficients shows that they are contributors to a helpful review posted by a user. Read, Read book and american are negatively correlated but we need to remove more common words to remove. We can use these words to suggest user what all words can led to a positive and a negative review This can be a suggestion or a tip for a healthy and good review. Negative reviews will be rejected.

```

: #Displayin the top parameters
words.extend(['score'])
sorted(zip(words,gs.best_estimator_.coef_[0]),key=lambda x:x[1])

: [('read book', -1.4517062337490938),
  ('read', -0.65564660289473486),
  ('american', -0.33220442695913532),
  ('good', 0.097733434820765394),
  ('don', 0.099658412191117013),
  ('people', 0.1688742692035092),
  ('war', 0.24795276487231696),
  ('reading', 0.29325034925531579),
  ('world', 0.40865494487118792),
  ('history', 0.56390817369734958),
  ('score', 0.73301317655666598),
  ('love', 0.74225660023473528),
  ('like', 0.75846148131849778),
  ('just', 0.76507499906274401),
  ('books', 0.92842676539356794),
  ('time', 1.1204680692599365),
  ('really', 1.1350328179437257),
  ('quot', 1.1627607462714684),
  ('ve', 1.299041736152452),
  ('author', 1.4240716916070468),
  ('new', 1.5545441216921103),
  ('characters', 1.5691960894924462),
  ('plot', 1.7299017179279561),
  ('story', 1.7573310939537168),
  ('book', 1.8228550220322288),
  ('novel', 1.8706610916276731),
  ('character', 1.9134522122828601),
  ('work', 2.0474878313886276),
  ('life', 2.3307098581062005)]

```

## Topic Modeling using LDA (Latent Dirichlet Allocation) in R:

Latent Dirichlet allocation (LDA) is a topic model that generates topics based on word frequency from a set of documents. LDA is particularly useful for finding reasonably accurate mixtures of topics within a given document set

**Packages Used:** nltk, numpy, genism, csv.

We have divided the book reviews text posted by users into different Topics that will consist of number of Documents. These topics will describe the structure of the document and contains the most probable documents in it.

### Implementation:

#### 1) Cleaning Data:

We will remove the stop words, punctuations and convert the sentences into lower case and append it into a list “texts”

We will use the **tokenizer** to convert texts into tokens and **stemming** to group(stem) similar words.

```
# clean and tokenize document string
tokens = tokenizer.tokenize(raw_lower)

# remove stop words from tokens
stopped_tokens = [i for i in tokens if not i in en_stop]

# stem tokens
stemmed_tokens = [p_stemmer.stem(i) for i in stopped_tokens]

# add tokens to list
texts.append(stemmed_tokens)
```

## 2) Implementing the Model:

We will convert the text into a dictionary using `doc2bow()`. We will build the LDA model using the `gensim` default `LdaModel` method. We will initially give the **number of topics** as 5. We can change it according to the number of words and depending upon the result.

```
# # convert tokenized documents into a document-term using doc2bow that will produce documents.
corpus = [dictionary.doc2bow(text) for text in texts]

#generate LDA model
lda = gensim.models.ldamodel.LdaModel(corpus, num_topics=5, id2word = dictionary, passes=20)
```

## 3) Analyze the Result:

As we can analyze below is a topic wise division of a specific book. We can see in each topic there are different documents with their probable score. More the probability, more likely is the correlation of the word in the topic.

In topic 1, all the words **“wife”, “husband”, “girl”, “marriage”** words depict the script of the novel *The Gone girl* with a high probable score. If a user is seeing the novel for the first time, he can get an idea of the kind of reviews and an overview of the book.

```
#Topic Wise words with probability..
for i in lda.show_topics():
    print(i[0], i[1])

0 0.047*“ami” + 0.041*“nick” + 0.034*“s” + 0.010*“wife” + 0.008*“stori” + 0.008*“husband” + 0.007*“gone” + 0.007*“dis
appear” + 0.007*“girl” + 0.007*“marriag”
1 0.053*“book” + 0.041*“read” + 0.022*“twist” + 0.018*“stori” + 0.017*“end” + 0.016*“charact” + 0.015*“turn” + 0.013
*“love” + 0.012*“great” + 0.012*“plot”
2 0.053*“book” + 0.041*“t” + 0.033*“end” + 0.029*“read” + 0.018*“like” + 0.015*“just” + 0.012*“realli” + 0.010*“didn”
+ 0.010*“charact” + 0.010*“stori”
3 0.032*“charact” + 0.011*“person” + 0.010*“stori” + 0.010*“plot” + 0.010*“peopl” + 0.010*“main” + 0.008*“two” + 0.00
8*“narrat” + 0.006*“like” + 0.006*“well”
4 0.025*“s” + 0.017*“flynn” + 0.016*“girl” + 0.015*“gone” + 0.011*“novel” + 0.009*“charact” + 0.008*“one” + 0.008*“ca
n” + 0.006*“marriag” + 0.006*“gillian”
```

### Trying with different parameters

```
#Trying with different topic value..
print(lda.print_topics(num_topics=10, num_words=10))

[(0, '0.073*"book" + 0.054*"read" + 0.045*"t" + 0.024*"twist" + 0.024*"put" + 0.016*"turn" + 0.016*"love" + 0.015*"gr
eat" + 0.015*"end" + 0.015*"couldn"', (1, '0.018*"movi" + 0.015*"war" + 0.012*"rose" + 0.011*"ben" + 0.010*"affleck"
+ 0.005*"cover" + 0.005*"scott" + 0.005*"1st" + 0.004*"recogn" + 0.004*"21st"', (2, '0.063*"t" + 0.034*"end" + 0.028
*"book" + 0.025*"like" + 0.020*"character" + 0.019*"didn" + 0.017*"don" + 0.017*"just" + 0.013*"star" + 0.012*"s"',
(3, '0.066*"book" + 0.038*"read" + 0.032*"end" + 0.017*"like" + 0.013*"just" + 0.012*"good" + 0.010*"get" + 0.010*"ti
me" + 0.010*"author" + 0.009*"realli"', (4, '0.019*"s" + 0.018*"character" + 0.016*"stori" + 0.011*"one" + 0.010*"nove
l" + 0.010*"book" + 0.009*"plot" + 0.009*"two" + 0.009*"gone" + 0.009*"wife"', (5, '0.010*"sister" + 0.008*"twin" +
0.007*"margo" + 0.007*"bar" + 0.007*"call" + 0.007*"front" + 0.006*"cat" + 0.005*"town" + 0.005*"york" + 0.005*"la
w"', (6, '0.063*"ami" + 0.055*"nick" + 0.038*"s" + 0.008*"wife" + 0.007*"disappear" + 0.007*"stori" + 0.006*"anniver
sari" + 0.006*"marriag" + 0.006*"dunn" + 0.006*"husband"', (7, '0.012*"don8217t" + 0.011*"it8217" + 0.009*"8211" +
0.007*"can8217t" + 0.007*"i8217m" + 0.006*"amy8217" + 0.006*"extent" + 0.006*"didn8217t" + 0.005*"nick8217" + 0.004
*"8221"', (8, '0.031*"s" + 0.027*"girl" + 0.024*"gone" + 0.020*"flynn" + 0.011*"can" + 0.011*"gillian" + 0.011*"t" +
0.009*"know" + 0.008*"one" + 0.008*"read"', (9, '0.030*"book" + 0.026*"character" + 0.025*"read" + 0.023*"end" + 0.023
*"flynn" + 0.021*"twist" + 0.018*"stori" + 0.018*"well" + 0.015*"plot" + 0.015*"turn"')]
```

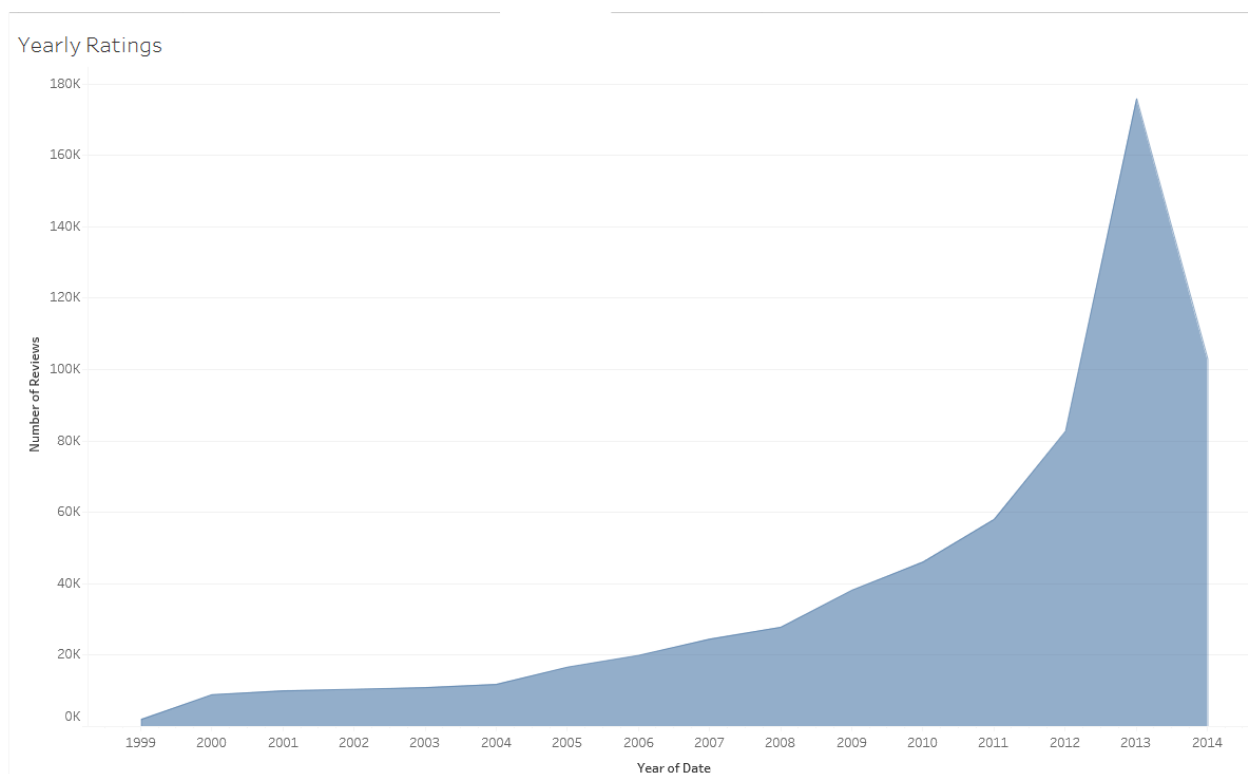
### Application and Future Scope:

Whenever user selects a book, we can display the **topic wise top documents** so that he would not have to go through all the reviews and check to get the idea of the type of book. He can get an idea in nutshell of the book and it will save time and customer experience.

### **Exploratory Analysis (Using Tableau):**

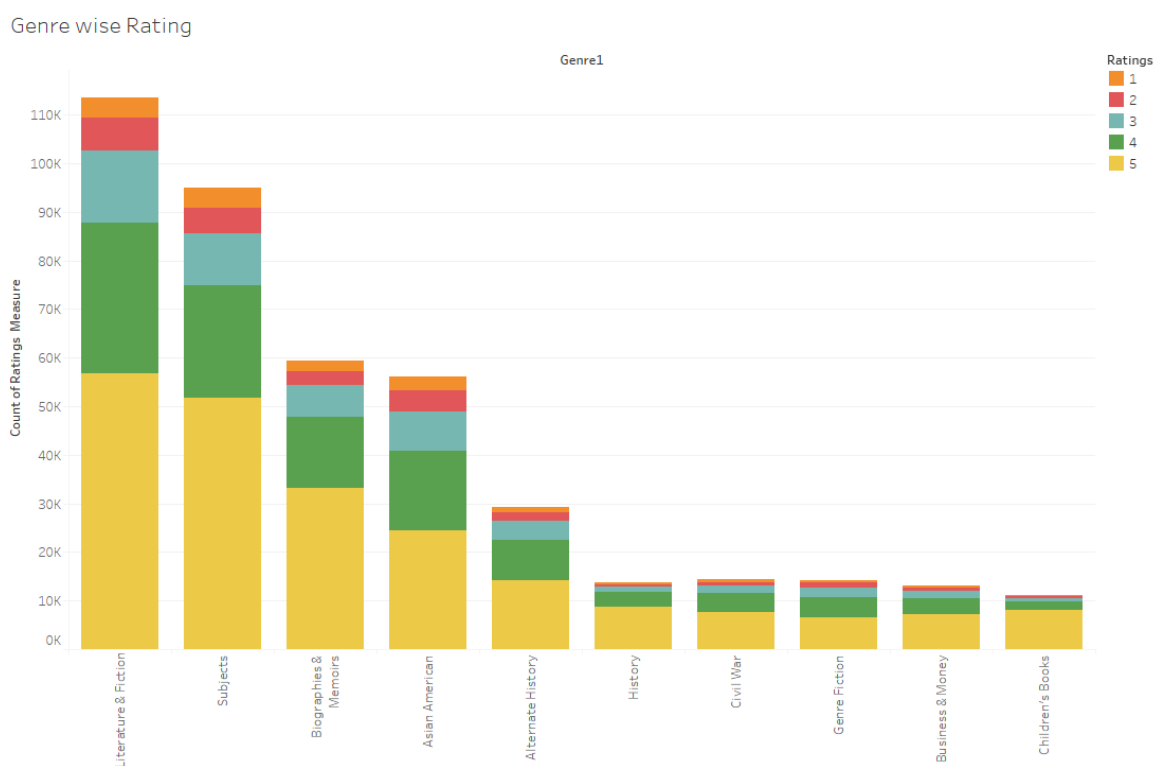
Reviews from users who have used the product in question can give us more context to the product. Each reviewer rates the product from 1 to 5 stars, and provides a text summary of their experiences and opinions about the product. The ratings for each product are averaged together to get an overall product rating.

As depicted below, number of reviewers keeps increasing year by year using the amazon book store.



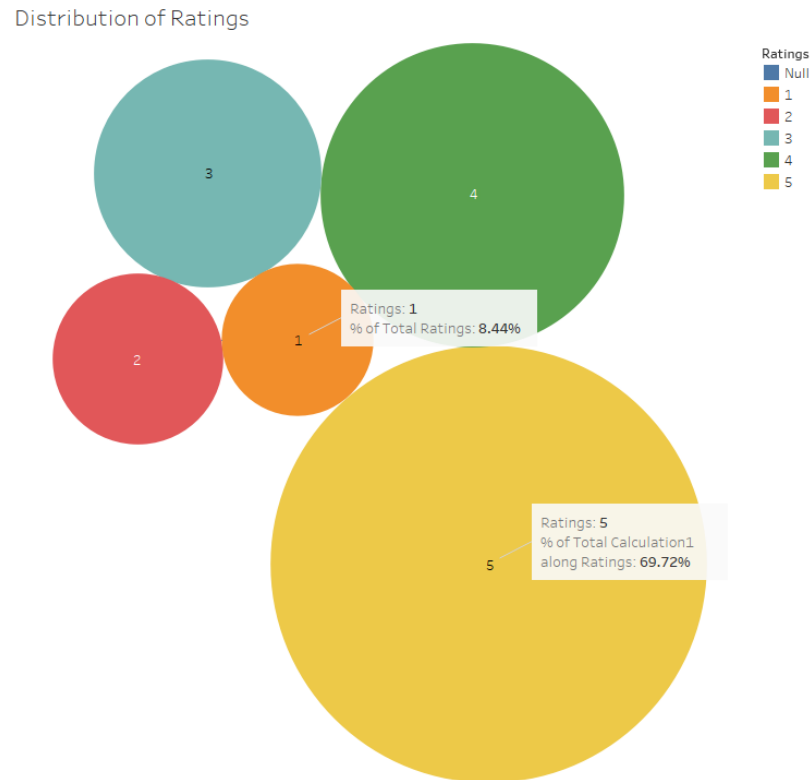
## 1) Genre based Ratings Distribution

Given stacked bar chart shows the distribution of ratings over the genre. Literature leads the race with around 55k 5-star ratings followed by Biographies and memories. This visualization proves that the top categories have a significantly higher percentage of 4/5-star ratings than the bottom categories, and a much a lower proportion of 1/2/3-star ratings. The inverse holds true for the bottom categories.



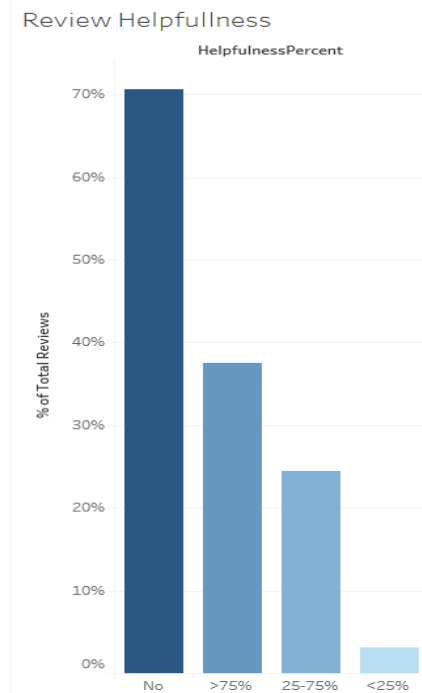
## 2) Distribution of Ratings:

We can see the distribution of the ratings over the bookstore reviews writer. 5-star rating composes the major portion i.e. 69.72% and 1-star lowest with 8.44 %. Rest of the ratings follow with average percentages.



### 3) Distribution of Ratings:

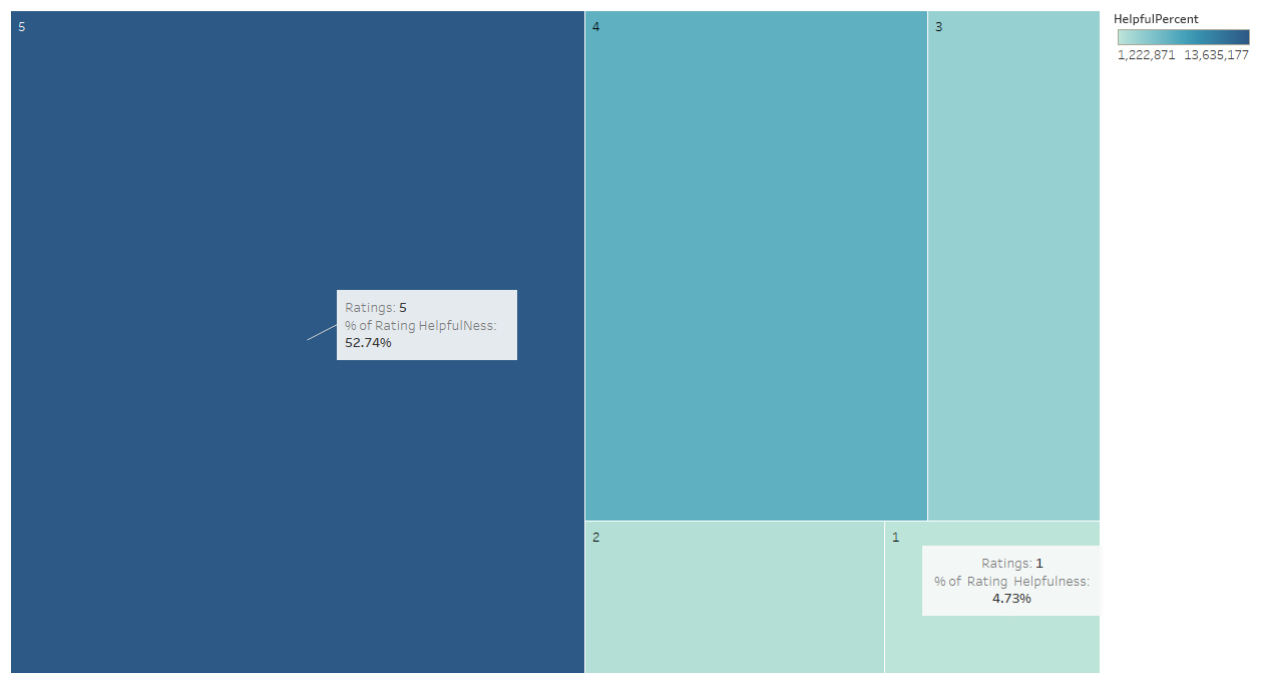
Around 70% reviews are not voted. We have created a derived column Helpfulness by dividing Helpful Numerator by total reviewed ratings. As we can see, around 40 percent reviews are >75 % helpful and the trend goes on as the helpfulness decreases.



#### 4) Helpfulness by Ratings :

As expected, 5 rating reviews were most helpful with 52 percent and the 1 rating reviews were least 4.73 % helpful

Helpfulness By Ratings

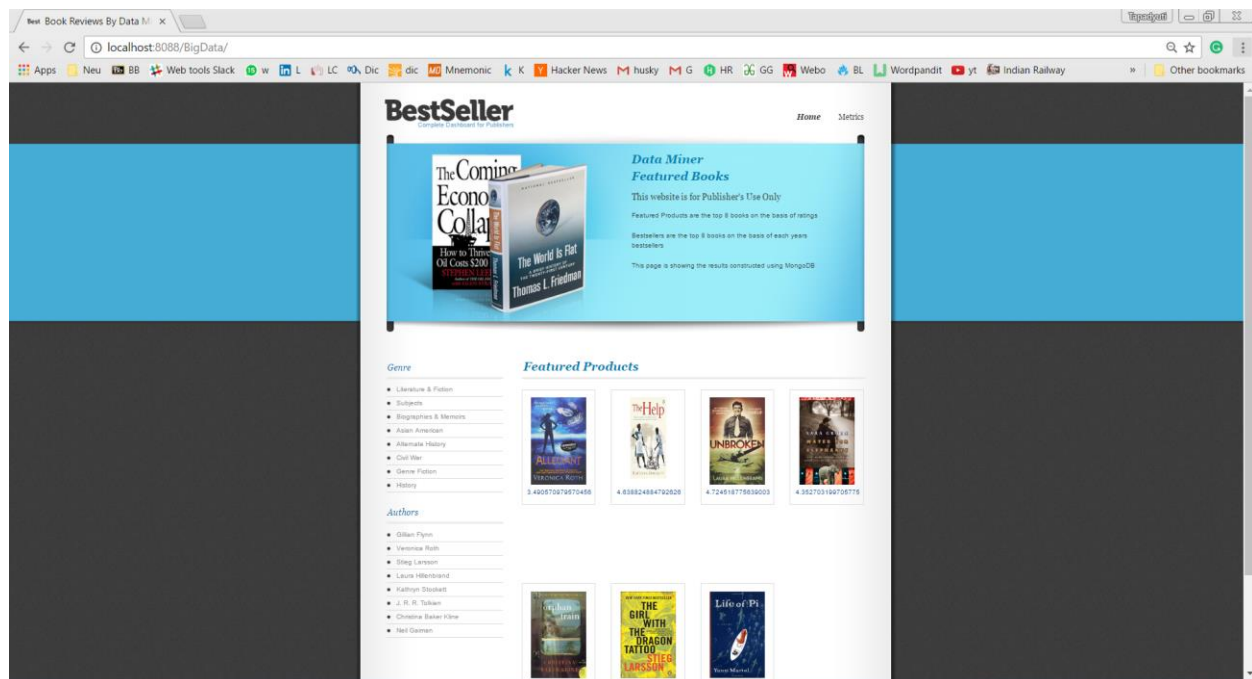






Word Cloud for the book Gone Girl.

## Web Application/MongoDB connectivity :



### Priliminary Step :

Importing Data from json format to MongoDB 3.4

Following Command has been used:

Test- default Database Schema present in MongoDB

```
./mongoimport -d test -c DataFinal -f
```

```
Asin,ReviewerID,Genre1,HelpfulNumerator,HelpfulDenominator,Review_Text,Review_Summary,ReviewerName,Date,Ratings,Title,Genre,Author,GenreID,UniqueReviewerID,UniqueASINID,YEAR --file F:\MyFinalData.json
```

MongoDb Basic Queries:

### Steps taken :

```

38 MongoClient mongo = new MongoClient("localhost", 27017);
39
40 DB db = mongo.getDB("test");
41
42 DBCollection table = db.getCollection("DataFinal");
43
44 List<String> unid = table.distinct("Asin");
45
46
47
48
49 //BasicDBObject searchQuery = new BasicDBObject();
50 //searchQuery.put("Asin", "B001B6130Y");
51 DBObject groupFields = new BasicDBObject( "_id", "$Asin");
52
53 groupFields.put("count", new BasicDBObject( "$sum", 1));
54 //DBObject group = new BasicDBObject("$group", groupFields );
55 DBObject sortFields = new BasicDBObject("count", -1);
56 //AggregationOutput output = table.aggregate(group, new BasicDBObject("$sort", sortFields ));
57 //System.out.println(output.results()); //Top reviews
58
59 groupFields.put("average", new BasicDBObject( "$avg", "$Ratings"));
60 DBObject group2 = new BasicDBObject("$group", groupFields);
61
62 AggregationOutput output2 = table.aggregate(group2, new BasicDBObject("$sort", sortFields ));
63
64
65 Iterable<DBObject> iterable = output2.results();
66
67 //LinkedList<String>stringResult = new LinkedList<String>();
68 //LinkedList<String>stringRating = new LinkedList<String>();
69
70 //if(awsResult.size()!=0){awsResult.remove();}
71 //if(stringResult.size()!=0){stringResult.remove();}
72
73
74 //System.out.println("*****"+stringResult.size());

```

Console: No consoles to display at this time.

Updates Available: Updates are available for your software. Click to review and install updates. You will be reminded in 30 minutes. Set reminder preferences.

Validating BigData: (7%)

1.  
db.DataFinal.aggregate( {\$sortByCount:"\$ReviewerID"});

Finding AsinId , sorted by numberof Reviews.

Key	Value	Type
(1) AFVQZ8PWOL _id count	{ 2 fields } AFVQZ8PWOL 2980	Object String Int32
(2) A2F6N60Z96CAII _id count	{ 2 fields } A2F6N60Z96CAII 1790	Object String Int32
(3) A140IS0VWMO5WIO _id count	{ 2 fields } A140IS0VWMO5WIO 780	Object String Int32
(4) A1K1JW1CSCUSUZ _id count	{ 2 fields } A1K1JW1CSCUSUZ 634	Object String Int32
(5) AHUT55E9B0RDR _id count	{ 2 fields } AHUT55E9B0RDR 552	Object String Int32
(6) A22R8N8CND3A _id count	{ 2 fields } A22R8N8CND3A 442	Object String Int32
(7) A21NVBFIEQWDSG _id count	{ 2 fields } A21NVBFIEQWDSG 442	Object String Int32
(8) A2MF2QVSCUI2TG _id count	{ 2 fields } A2MF2QVSCUI2TG 442	Object String Int32
(9) A2TX179KATSGRP _id count	{ 2 fields } A2TX179KATSGRP 442	Object String Int32
(10) AKPKPMW6GIUS _id count	{ 2 fields } AKPKPMW6GIUS 442	Object String Int32
(11) AHD101501WCN1 _id count	{ 2 fields } AHD101501WCN1 442	Object String Int32
(12) A1G37DF08MQW0M _id count	{ 2 fields } A1G37DF08MQW0M 442	Object String Int32
(13) A102COWDCSHUWZ _id count	{ 2 fields } A102COWDCSHUWZ 442	Object String Int32
(14) AC1K4QOZ96RS _id count	{ 2 fields } AC1K4QOZ96RS 442	Object String Int32
(15) A1NATT3PN24QWY _id count	{ 2 fields } A1NATT3PN24QWY 442	Object String Int32
(16) A3M174K0VXD5X _id count	{ 2 fields } A3M174K0VXD5X 442	Object String Int32
(17) A1D5RCOLPC9LX _id count	{ 2 fields } A1D5RCOLPC9LX 442	Object String Int32
(18) A320TMDV6KCFU _id count	{ 2 fields } A320TMDV6KCFU 442	Object String Int32
(19) A3U7ELIED4WP4R _id count	{ 2 fields } A3U7ELIED4WP4R 442	Object String Int32
(20) A39650P2CZUUC9 _id count	{ 2 fields } A39650P2CZUUC9 442	Object String Int32
(21) A25KXUQ30CLWB _id count	{ 2 fields } A25KXUQ30CLWB 442	Object String Int32
(22) A8IPQ1Q107YXS _id count	{ 2 fields } A8IPQ1Q107YXS 442	Object String Int32

## 2. Finding all distinct Asin Id in the document.

The screenshot shows the Robomongo 1.0 interface. The left sidebar displays a database structure with collections like BookReco, BookRecommendor, DataFinal, DataFinalTest, Tester, homye, page, testmodels, traffic, user, website, widget, Functions, and Users. The main window shows a query executed on the 'DataFinal' collection: `db.DataFinal.distinct("Asin")`. The query took 1.905 seconds to execute. The results are displayed in a table with three columns: Key, Value, and Type. The 'Value' column contains a list of 32 distinct ASIN strings, starting from '000100039X' and ending with '000719739X'. The 'Type' column indicates that the values are of type 'String'.

Key	Value	Type
[0]	ASIN	Array
[1]	000100039X	String
[2]	000171287X	String
[3]	000215725X	String
[4]	000215949X	String
[5]	000221665X	String
[6]	000221735X	String
[7]	00022383X	String
[8]	000224053X	String
[9]	000255383X	String
[10]	000470763X	String
[11]	000612609X	String
[12]	000615591X	String
[13]	000616174X	String
[14]	000616823X	String
[15]	000617471X	String
[16]	000617681X	String
[17]	000639325X	String
[18]	000647019X	String
[19]	000648204X	String
[20]	000649319X	String
[21]	000649692X	String
[22]	000649885X	String
[23]	000651412X	String
[24]	000712113X	String
[25]	000712614X	String
[26]	000712693X	String
[27]	000712774X	String
[28]	000713326X	String
[29]	000716212X	String
[30]	000716307X	String
[31]	000717604X	String
[32]	000719739X	String

## 3. Finding Count of the Document:

The screenshot shows the Robomongo 1.0 interface. The left sidebar displays the same database structure as the previous screenshot. The main window shows a query executed on the 'DataFinal' collection: `db.DataFinal.count()`. The query took 0.12 seconds to execute. The result is displayed as a single value: 1295832.

Count
1295832

#### 4. Getting all the Distinct Genres

The screenshot shows the Robomongo 1.0 interface. On the left, the 'test' database is expanded, showing the 'DataFinal' collection. The main window displays the query `db.DataFinal.distinct("Genre1")` and its results. The query took 1.445 seconds to execute. The results are shown in a table with three columns: Key, Value, and Type.

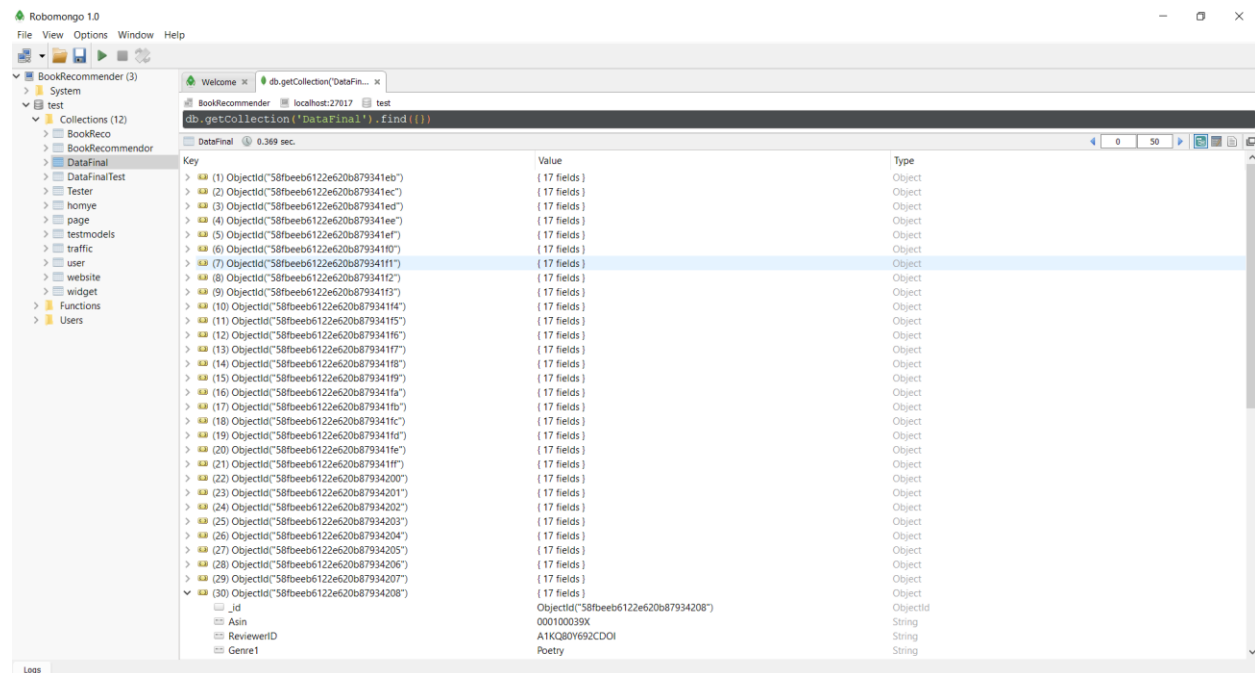
Key	Value	Type
(1)	[ 384 elements ]	Array
[0]	Genre1	String
[1]	Poetry	String
[2]	Literature & Fiction	String
[3]	History	String
[4]	Cookbooks Food & Wine	String
[5]	Ancient World	String
[6]	Alternate History	String
[7]	Subjects	String
[8]	Science Fiction & Fantasy	String
[9]	Reference	String
[10]	Mystery Thriller & Suspense	String
[11]	Biographies & Memoirs	String
[12]	Action & Adventure	String
[13]	Genre Fiction	String
[14]	African American	String
[15]	Anthologies	String
[16]	Oil Painting	String
[17]	Americas	String
[18]	Biographies	String
[19]	British & Irish	String
[20]	Individual Directors	String
[21]	Coming of Age	String
[22]	Anger Management	String
[23]	Diplomacy	String
[24]	Crafts Hobbies & Home	String
[25]	Asian American	String
[26]	Civil War	String
[27]	Engineering & Transportation	String
[28]	Arts & Photography	String
[29]	Law	String
[30]	Humor & Entertainment	String
[31]	Christian Books & Bibles	String
[32]	Children's Books	String

#### 5. Number of Review for each books in Ascending order.

The screenshot shows the Robomongo 1.0 interface. On the left, the 'test' database is expanded, showing the 'DataFinal' collection. The main window displays the aggregation query `db.DataFinal.aggregate([{$group:{$_id:"$asin",total:{$sum:1}}},{ $sort:{$total:1}}])` and its results. The query took 1.237 seconds to execute. The results are shown in a table with three columns: Key, Value, and Type.

Key	Value	Type
(1) 030758836X	{ 2 fields }	Object
_id	030758836X	String
total	14868.0	Double
(2) 7444117	{ 2 fields }	Object
_id	7444117	Int32
total	7636.0	Double
(3) 141039280	{ 2 fields }	Object
(4) 7386648	{ 2 fields }	Object
(5) 2007770	{ 2 fields }	Object
(6) 61950726	{ 2 fields }	Object
(7) 143170090	{ 2 fields }	Object
(8) 151008116	{ 2 fields }	Object
_id	151008116	Int32
total	4480.0	Double
(9) 2247399	{ 2 fields }	Object
(10) 030728090X	{ 2 fields }	Object
(11) 143124544	{ 2 fields }	Object
(12) 307277674	{ 2 fields }	Object
(13) 261103288	{ 2 fields }	Object
_id	261103288	Int32
total	3594.0	Double
(14) 307265439	{ 2 fields }	Object
(15) 307346609	{ 2 fields }	Object
(16) 61537934	{ 2 fields }	Object
(17) 60098902	{ 2 fields }	Object
_id	60098902	Int32
total	3400.0	Double
(18) 307476073	{ 2 fields }	Object
(19) 141326085	{ 2 fields }	Object
(20) 622525657	{ 2 fields }	Object
(21) 143170104	{ 2 fields }	Object
_id	143170104	Int32
total	2904.0	Double
(22) 7124015	{ 2 fields }	Object

## 6. Showing all Documents:



## 7. Changing String format to Int for Rating field.

```
db.DataFinal.find({Ratings: {$exists: true}}).forEach(function(obj) {
    obj.Ratings = new NumberInt(obj.Ratings);
    db.DataFinal.save(obj);
});
```

## **Contribution by Each Individual: -**

As a team, we have worked effectively on all areas of the project in a collaborative manner and contributed equally to each part. As some of the functionalities are dependent to each other we must schedule and allot the tasks sequentially.

**Fredy Daruwala:** - He is responsible for the data cleaning part of the application using the Map and Reduce operation. He is also responsible for most of the storage and management of the data. He is also responsible for handling the Mahout Recommendation engine and to recommend books to the users.

**Gautam Pawar:** - He is responsible for running all the machine learning algorithms on the data set. He also is responsible for cleaning the data and running the machine learning G algorithms on the data.

**Nikita Anand:** - She is responsible for all the Tableau visualizations, data cleaning activity using R and Python. She is also responsible for building the Web application for performing different analysis on and predict future and study past trends.

**Tapadyuti Maiti:** - He is responsible for building the web application and running the NLP analysis on reviews data so that we can obtain incentivized and non-incentivized reviews analysis.

## **Issues and Their Resolution: -**

The main issues we faced in the project were the following: -

- Data Cleaning activity which took a long time as there were a lot of missing values on the json file and data would not be loaded because of that. The solution was that we had to use a series of mappers and reducers to obtain the clean data that we could use.
- Another main challenge we faced was connecting to the Amazon Product API to fetch the additional data values as it would not allow many calls to the api in a second. The solution to that was to use multi-threading or use thread.sleep method to hold on for the next iteration.
- Another challenge was to deploy the data into MongoDB and access the same. It was not easy connecting the application to the MongoDB server. There were also issues loading the data into the same.

**Conclusion/Result:**

- We can recommend users with a variety of books from the Mahout Recommendation engine.
- We can analyze multiple reviews to determine if they are incentivized or not.
- We were successfully able to predict the relationship between helpfulness and the review text.
- We could describe the structure and nature of the review into various topics.
- Also, we could recommend friends of similar user and in turn predict the books they would like to read.

**Future Scope : -**

We can do real time data analytics of the Amazon Product API as we have used the dataset of the amazon movie review datasets. We can create a module for suggesting modules for helpful reviews to the users and publishers.