UNIVERSITÀ
DI PAVIA

From Raw Data to Informed Decisions: Analyzing Amazon Book Reviews

Alberti A. Ligari D. Andreoli A.[1]

[1] *Data Science and Big data Analytics course, University of Pavia, Department of Computer Engineering (Data Science), Pavia, Italy*
Github page: https://github.com/DavideLigari01/data-science-project

Date: September 18, 2023

Abstract —prova

Keywords —Big Data • Hadoop • Spark • ML • MongoDB • Data Analysis • Data Visualization • Python

Contents

1. Introduction

In In the age of digital commerce, customer reviews profoundly impact product perception and purchase decisions. Amazon, with its extensive repository of book reviews spanning nearly two decades, holds a wealth of valuable insights, sentiments, and trends. This project aims to create a scalable solution for uncovering patterns, sentiment trends, and correlations within the realm of book reviews, utilizing advanced tools and technologies like Hadoop, Spark, MongoDB, and Python libraries such as Pandas and Scikit-learn. In this report, we provide a detailed exploration of our project, covering stages from initial data discovery and preparation to feature extraction, model building, and evaluation.

2. Discovery

To begin our data science campaign was crucial to compose the team, clearly fix our goal and exploring about the available resources.

a. Team

The team is composed by three members:

Andrea Alberti: github.com/AndreaAlberti07
Davide Ligari: github.com/DavideLigari01
Cristian Andreoli: github.com/CristianAndreoli94

b. Framing

The **goal** of the project is to develop a **scalable** solution to analyze a dataset of reviews of books from Amazon and eventually build a model able to predict the helpfulness of a review based on its content.

c. Tools

The tools were chosen with the aim of developing a scalable solution, able to work in a **Big Data** environment.

- **Virtual Machine**: to work in a controlled environment.
- **Hadoop**: to store data in a distributed file system and to perform MapReduce operation on it.
- **Spark**: used as an improved alternative to MapReduce, to perform operations on the distributed dataset.
- **Python**: as the reference programming language thanks to its libraries.
- **MongoDB**: used as NoSQL database sandbox to safely work with local data.
- **GitHub**: to share and collaborate on the project.
- **LaTeX**: to write the report.

3. Data Preparation

To start with the project, we needed to retrieve and prepare the data.

Data Retrieval and Prior Analysis

The chosen dataset is composed by 2 tables and about 3 millon reviews, accessible at the following link: Amazon Books Reviews.
Once the dataset has been acquired we performed the following steps:
1. **HDFS loading** using the following commands:

```
# Create HDFS directories
hdfs dfs -mkdir -p "$HDFS_PATH/ratings"
hdfs dfs -mkdir -p "$HDFS_PATH/
    books_info"

# Copy local files to HDFS
hdfs dfs -copyFromLocal "$LOCAL_PATH/
    ratings.csv" "$HDFS_PATH/ratings/"
hdfs dfs -copyFromLocal "$LOCAL_PATH/
    books_info.csv" "$HDFS_PATH/
    books_info/"
```

2. **Prior analysis** using *pyspark* to get a better understanding of the data. In this phase we defined a schema for our data and we computed some statistics together with the percentage of missing values and unique values for each field or our dataset.

Hypothesis Generation

After the prior analysis, we started to think about some hypotheses that we could test on our data. We came up with the following ones:

- **H1**: Reviews with longer text have higher helpfulness ratings.
- **H2**: Reviews with more positive sentiment words receive higher helpfulness ratings.
- **H3**: Reviews with higher book ratings have higher helpfulness ratings.
- **H4**: The rating score is influenced by individual users, which may either overestimate or underestimate a book's quality. Anonymous tends to overrate the books.
- **H5**: The review score is influenced by the category of the book.
- **H6**: The larger the number of books published for a category or publisher, the higher the review score.

Data Cleaning

In this step we cleaned the data, removing duplicates, useless columns for our analysis and any symbol that could have interfered with the reading of the csv files. All the cleaning operations were performed using *pyspark*.

Data Aggregation

The MapReduce job performs an inner join operation between the "Data table" and the "Rating table" based on the book title, resulting in a single file containing the joined records from both tables.

Mapper

The Mapper script processes input data line by line, converting each line into a key-value structure. The key represents the book title, and the value contains the remaining line content. To distinguish between records from the 'Data table' and 'Rating table' and ensure the correct processing order in the Reducer phase, the Mapper appends a special character ('-' for 'Data table' and 'www' for 'Rating table') as the second key element. This ensures that 'Data table' records are processed before 'Rating table' records during subsequent MapReduce phases.

Reducer

The Reducer script processes intermediate output records generated by the Mapper, aiming to join 'Data' and 'Rating' records for the same title. The Reducer reads records sequentially, storing 'Data' and 'Rating' information separately. When both 'Data' and 'Rating' records for the same title are available, the Reducer performs the join operation by combining the data from these records.

MongoDB loading

Once all the previous operations were completed, we loaded the data into MongoDB. To do so the steps executed are:

- Connect to MongoDB using *'pymongo'*
- Connect to HDFS and read the data using *'spark.read.csv'*
- Select a subset of the Spark DataFrame to import using *'sample'* method
- Transform the data into a dictionary using *'to_dict'* method
- Insert the data into MongoDB using *'insert_many'* method

We loaded in MongoDB the two tables *ratings* and *books_info*, together with the joined table produced by MapReduce. These data have been used to perform the local hypotheses testing described below.

4. Local Hypotheses Testing

Hypothesis 1

Reviews with longer text have higher helpfulness ratings.

Description and Results The data cleaning and 'review/helpfulness' transformation process ($helpfulness = \frac{x}{y}\sqrt(y)$) was performed using the *'pymongo'* library to exploit the MongoDB efficiency. Specifically, we defined a pipeline to perform the needed operations. As regards the 'review/text' transformation, we used the *'nltk'* library to tokenize the text, remove punctuation, stopwords and eventually count the number of words.
The correlation coefficient between the two variables is 0.3313 with a p-value < 0.05, indicating a statistically significant correlation.
A graphical confirmation is provided by Figure 1. Indeed there is a positive correlation until around 400 words, after which the boxplot stabilizes. Thereby, we decided to

analyze the correlation within the review length groups. As a results (Table 4) we got that the correlation is positive and statistically significant for the reviews with length between 0 and 400 words, while it becomes negative and statistically significant for the reviews with length greater than 750 words. As regards the reviews in the middle (i.e. between 400 and 750 words), the correlation is negligible.
Conclusion: The hypothesis is confirmed, but the correlation is not very strong and changes depending on the review length.

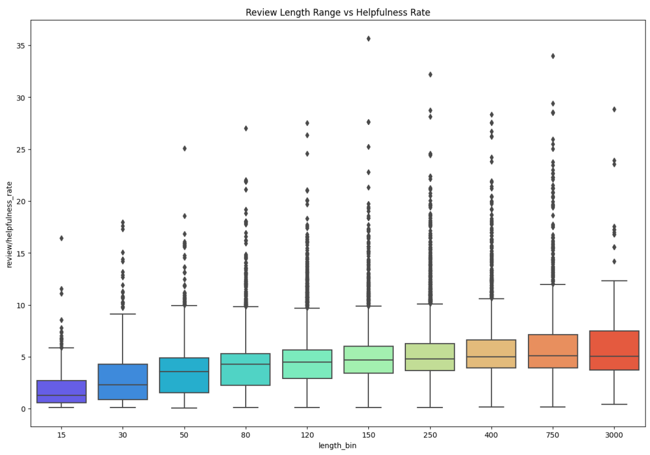

Fig. 1: Correlation between review length and helpfulness score for different review length groups

Table 1: Correlation Coefficients and P-values for Different Groups

Group Number	Correlation Coefficient	P-value
400	0.2216	0.0000
750	-0.0188	0.2585
3000	-0.1418	0.0065

Hypothesis 2

Hypothesis 3

Reviews with higher book rating have higher helpfulness ratings.

Description and Results As in the previous hypothesis, we decided to deal with missing values and data transformation directly with a MongoDB query. With the data ready to be processed, we decided to perform a prior investigation on the distribution of votes on the 4 rating categories. As shown by Figure 2, there is a **positive bias** as people are more prone to vote a positive reviews rather than a negative one. Specifically, the large majority of votes for rating 5, were composed by a number of total votes just equal to 1. This introduces a bias in our results, since according to the formula used to compute the helpfulness score, such a small number of total votes would lead to a small helpfulness score. To avoid this we decided to retain only the reviews with a number of total votes greater than 20.

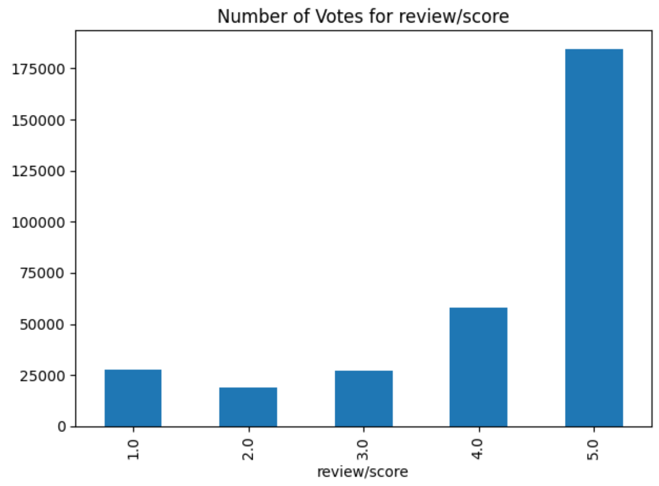

Fig. 2: Distribution of votes on the 4 rating categories

The Spearman correlation coefficient between the two variables is equal to 0.5247, with a p-value of 0.0.
Conclusion: The hypothesis is confirmed as there is a positive and statistically significant correlation between the two variables. This result is confirmed by the boxplot in Figure 3.

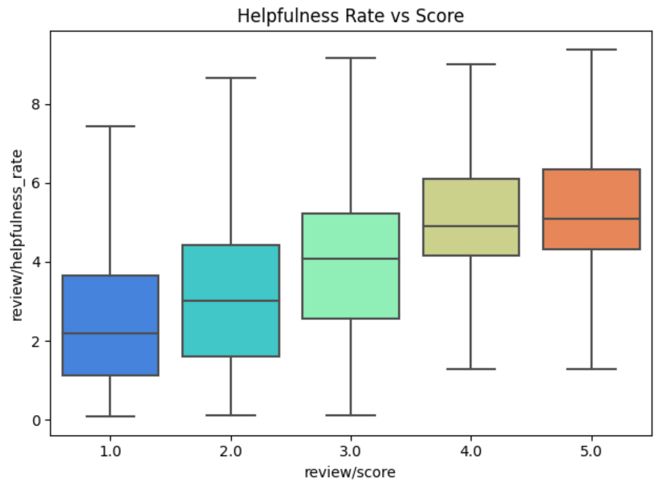

Fig. 3: Boxplot of the correlation between review rating and helpfulness score

Hypothesis 4

Hypothesis 4 explores the impact of individual users' unique personalities, personal preferences, and the potential for anonymous users to overrate books on rating scores. We tested this hypothesis by considering the rating score as the primary metric, The research team and any records with missing values were excluded from the analysis. The hypotheses under examination were as follows:
H0 (Null Hypothesis): The rating score is not influenced by the user's profileName. All rating scores are drawn from the same distribution, implying equal means and variances for each user's rating scores.
H1 (Alternative Hypothesis): The rating score is affected by the user, suggesting that each user's rating scores follow a distinct distribution.
For the sake of consistency, users with fewer than 20 reviews were excluded from the analysis, as a limited number of reviews cannot reliably estimate statistical measures.

The statistical test employed was ANOVA, which assesses differences in means between user groups. The results yielded an F-statistic of 1.5374 and a corresponding P-value of 0.0670. These results indicate that although there may be some variance in rating scores among different users, the evidence to reject the null hypothesis (H0) and conclude that user personalities significantly impact rating scores is not robust.
This conclusion is further supported by the accompanying boxplot (Figure 4), which illustrates variations in the distribution of rating scores across users.

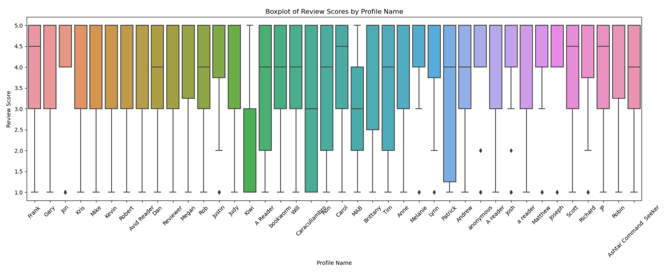

Fig. 4: Distribution of rating scores across users

Hypothesis 5

Hypothesis 5 examines the influence of book categories on review scores. To test this hypothesis,we considered the rating score as the metric and removed missing values. two competing hypotheses were established:
H0 (Null Hypothesis): Rating scores are not related to the book categories, as all rating scores are drawn from the same distribution.
H1 (Alternative Hypothesis): Rating scores are affected by the book category, indicating that the rating scores of each category follow different distributions.
As in the previous hypothesis, categories with fewer than 20 reviews were omitted for consistency. An ANOVA (Analysis of Variance) test was conducted to assess the validity of these hypotheses. The results of the test revealed an F-statistic of 0.177 and a P-value of 0.999. A low F-statistic value and a P-value close to 1 suggest that there is not much variation between the means of different categories. Therefore, we could not reject the null hypothesis (H0) and concluded that book categories do not significantly impact rating scores. This result was further supported by the accompanying boxplot (Figure 5), which showed that the distribution of rating scores was similar across categories.

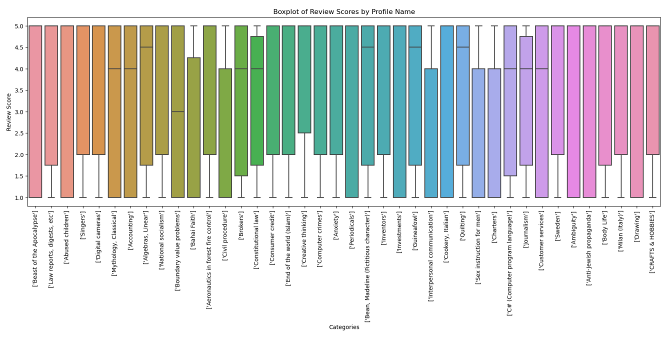

Fig. 5: Distribution of rating scores across categories

Hypothesis 6

- The larger the number of books published for a category, the higher the review score.
- The larger the number of books published by publishers, the higher the review score.

Description and Results All the data cleaning and transformation steps were performed using MongoDB with the aggregation pipeline, to have a more efficient and faster computation.
Specifically the data cleaning steps were performed using the *$match* operator, while the data transformation steps were performed using the *$group* operator with *$avg* operation. Finally a *$project* operator was used to select the fields of interest. To reduce bias, we removed the categories having less than 50 books and the publishers having less than 20 books. The results are shown in Table 2.
Conclusion: The hypotheses are falsified since the metrics shows no correlation between the two variables in both cases.

Table 2: Correlation Values and P-values for Categories and Publishers

Variable	Correlation Value	P-value
Category	-0.0806	0.558
Publisher	-0.0673	0.151

Curiosity We performed two complex MongoDB queries (reported as example in section 'Code') to answer to two questions:

- Which are the best publishers? (i.e. capable of getting avg scores above 4.5 in lots of categories)
- In which categories are the best publishers focused?

The results are reported in Figure 6 and Figure 7.

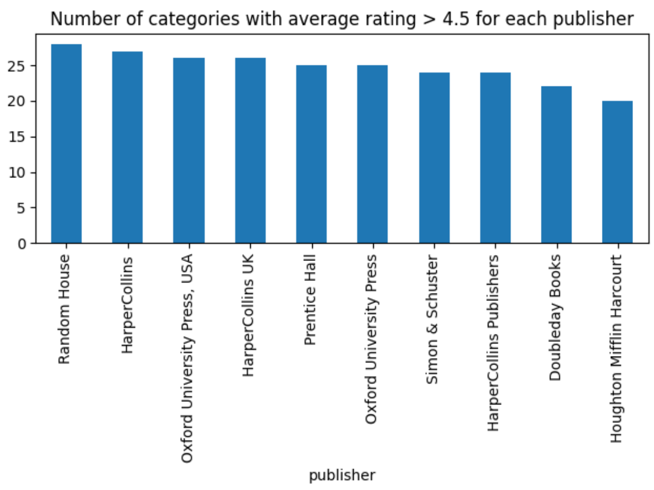

Fig. 6: Which are the best publishers?

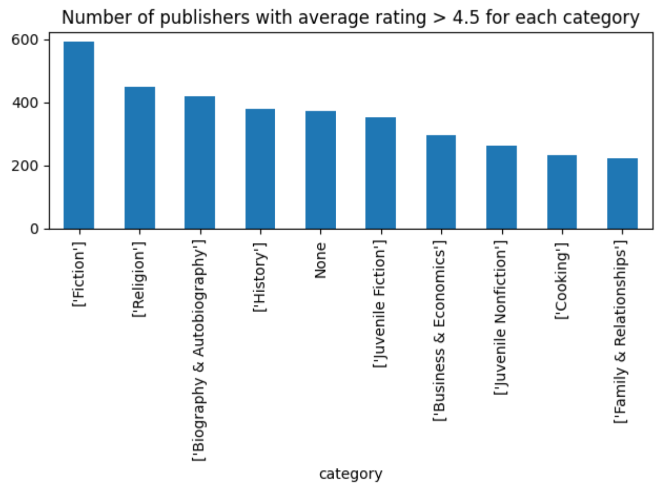

Fig. 7: In which categories are the best publishers focused?

5. Spark Hypotheses Testing

To show the possibility of implementing the data analysis in a Big Data context, we decided to replicate some hypotheses testing using Spark. In particular, we focused on the hypotheses 1 and 3.

Hypothesis 1

This hypothesis required three things:

- **Compute the helpfulness score:** That was simply achieved exploiting the *WithColumn* method of the Spark DataFrame, creating a new column with the new values.
- **Compute the text length:** This was achieved using *regexp_replace* to remove punctuations, *Tokenizer and StopWordsRemover* to tokenize the text and remove the stop words. Finally a new column with the length of the text was created.
- **Bucketize the text length:** This issue was solved exploiting the *Bucketizer* class of Spark MLlib together with a UDF to assign a proper label to the classes.
- **Compute the correlation coefficient:** Finally the correlation coefficient was computed using the *Correlation.corr* method of the Spark MLlib, since we needed the Spearman correlation coefficient. The data were reshaped to match the required format using *VectorAssembler*.

Hypothesis 3

This hypothesis required just to compute the helpfulness score and the correlation coefficient. Both of them were computed using the same methods described in the previous hypothesis.

Results

For both the tested cases, the results were almost the same as the ones obtained in the local environment.

6. Helpfulness Prediction

Our ambitious goal was that of building a model able to predict the helpfulness of a review, looking only at the text of the review itself. To create such a model, we needed firstly to convert the text in a machine-readable format, performing a process called *feature extraction*. Then we tried different models, in order to find the one that best fits our needs eventually selecting the best one.

a. Feature Extraction

Seen the complexity of the problem we decided to use a *Word Embedding* technique, called *Word2Vec*, that is able to convert a word in a vector of real numbers while preserving the semantic meaning of the word itself. We used the *Gensim* library to perform this task using the following parameters:

- **Size**: 30
- **Window**: 5
- **Min Count**: 2
- **Workers**: -1

As a consequence, each word is represented by a vector of 30 real numbers. The average of the vectors of all the words in a review is the vector representation of the review.

b. Models

We tried three different models: *Random Forest*, *Support Vector Regressor (RBF kernel)* and *MLP*. We used the *Scikit-Learn* library to perform the training and the testing of the models. Specifically for this last step we used the *GridSearchCV* class, that performs a cross-validation on the training set, in order to find the best parameters for the model. The results are show in Table 3 and supported by Figure 8.

Table 3: Model Results

Model	MSE	RMSE	R^2
RF	0.0259	0.1609	0.2532
SVR	0.0279	0.1670	0.1955
MLP	0.0282	0.1680	0.1858

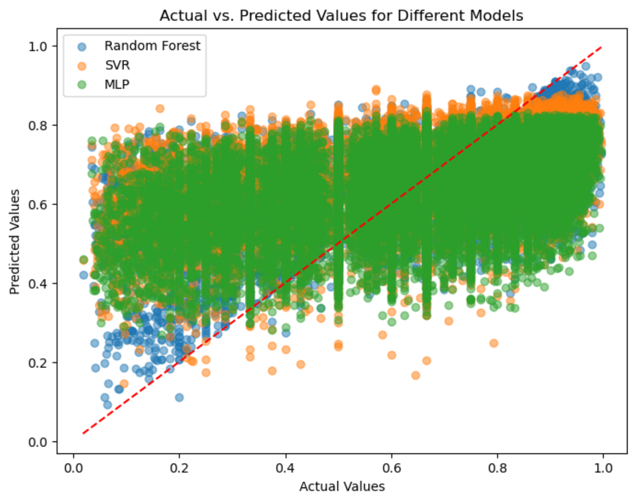

Fig. 8: Model Results

As we can see from the graph and the metrics used, the Random Forest model outperforms the other models in terms of Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). The Random Forest model achieved the lowest MSE of approximately 0.026 and RMSE of approximately 0.161, indicating that its predictions are, on average, the closest to the actual values. This suggests that Random Forest has the best overall predictive performance among the three models.

c. Results Interpretation

Figure 9 and Figure 10 help us in the results Interpretation.

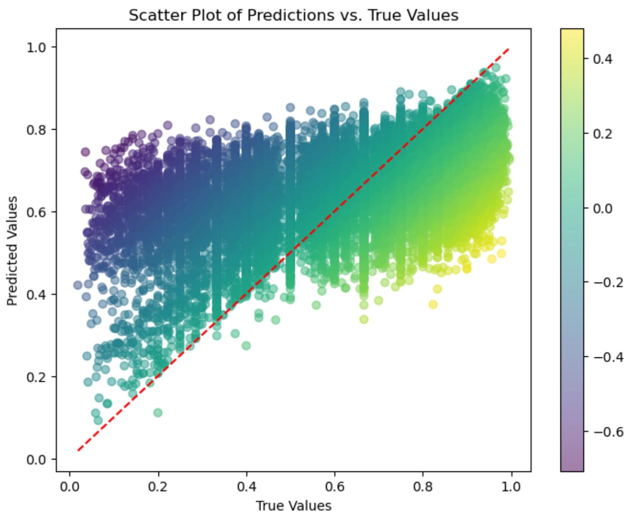

Fig. 9: Best Model Errors

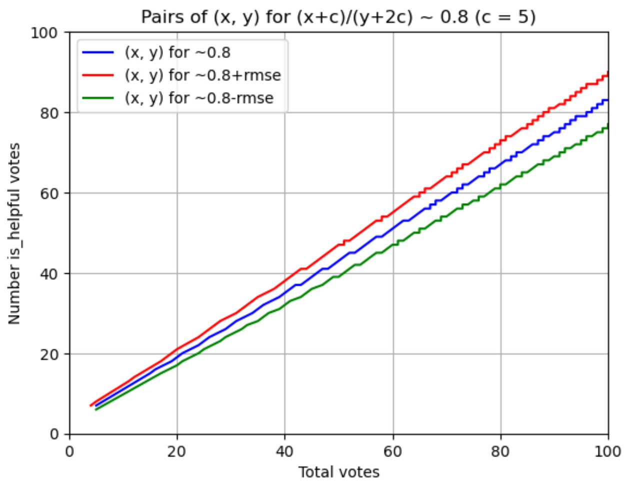

Fig. 10: Best Model Errors Translation

7. Conclusions

8. Python Script

This is the script to answer the question: In which categories are the best publishers focused?

```
# Deal with missing values
pipeline_missing = {'$match': {
    'review/score': {'$exists': True, '$ne': 0},
    'publisher': {'$exists': True, '$ne': None},
    'categories': {'$exists': True},
}
}

# Compute average rating for each tuple category,
    publisher
pipeline_average_rating = {'$group': {
    '_id': {
        'category': '$categories',
        'publisher': '$publisher',
    },
    'avg_score': {'$avg': '$review/score'},
    'count': {'$sum': 1}
}
}

# Show average rating for category for each
    publisher
pipeline_publisher = {'$group': {
    '_id': '$_id.category',
    'avg_score/publisher': {
        '$push': {
            'publisher': '$_id.publisher',
            'avg_score': '$avg_score',
            'count': '$count'
        }
    }
}
}

# Unwind the list of categories
pipeline_unwind = {'$unwind': '$avg_score/
    publisher'}

# Remove categories or publisher with less than '
    threshold' reviews
threshold = 0
pipeline_remove = {'$match': {
    'avg_score/publisher.count': {'$gte':
        threshold}
}
}

# Count the number of categories with average
    rating > 4.5
pipeline_counts = {'$project': {
    'category': '$_id',
    '_id': 0,
    'publisher': '$avg_score/publisher.publisher',
    'count': {
        '$sum': {
            '$cond': {

                'if': {'$gt': ['$avg_score/
                    publisher.avg_score', 4.5]},
                'then': 1,
                'else': 0
            }
        }
    }
}
}

# Sum the results for each publisher. If Total >
    10, then the hypothesis is False
pipeline_sum = {'$group': {
    '_id': '$category',
    'total': {'$sum': '$count'}
}
}

pipeline_sort = {'$sort': {
    'total': -1
}
}

results = books.aggregate([pipeline_missing,
    pipeline_average_rating, pipeline_publisher,
                          pipeline_unwind,
                              pipeline_remove,
                              pipeline_counts,
                          pipeline_sum,
                              pipeline_sort])
```