**ABSTRACT**

**This project presents an approach to developing a Streaming Platform Recommendation System for tv-shows. Leveraging PySpark, Tableau, and the Ubuntu environment, the study contains data-preprocessing, exploratory analysis, and implementation of a model. Through systematic text processing and feature engineering, the Random-Forest algorithm predicts top streaming platforms. Findings reveal the system's accuracy, highlighting personalization's significance. Also, graphical insights from Tableau shed more light on user preferences and desired content. The project's purpose lies in its ability to reform content discovery, enhancing user experiences and engagement across streaming platforms. The methodology shows the intricate interaction of data analysis procedures, algorithms in machine learning, and visualization tools, contributing to the progression of personalized entertainment recommendations.**

**Keywords— Recommendation Systems, Tv-shows, Data, Machine learning, User Experience, Random forest, Analysis**

**I. INTRODUCTION**

Video streaming platforms like Twitch or Youtube Live, are gradually becoming a major part of people’s daily lives. As of February 2020, Twitch reported 3 million broadcasters monthly and 15 million daily active users [1]. With the ever-growing number of streaming platforms presenting a vast array of TV-shows, viewers often find it overwhelming when trying to decide where to watch their favourite content. To address this challenge, we present SPRST, a Streaming Platform Recommendation System for TV-Shows that uses a Random Forest Classifier. The goal of this project work is to help users efficiently navigate and also find the ideal platform where their Television shows or programs are available on viewing platforms such as Netflix, Hulu, Disney+, and others. Recommender systems [10] have arisen to predict and recommend items that could meet user’s preferences.

The foundation of our project lies in a complete dataset sourced from Kaggle [2], comprising crucial information about TV shows, including features such as ID, Title, Year, Age, IMDb, Rotten-Tomatoes and the accessibility on major streaming platforms like Netflix, Disney+, Prime Video and Hulu. The dataset also includes some features that distinguish between movies and tv-shows.

Our data analysis tasks contain a two-fold approach. Firstly, we are performing exploratory data analysis (EDA) [3] to gain understandings into the distribution of TV shows across different platforms, evaluate rating allocations, and understand the correlations between features like IMDb and Rotten Tomatoes ratings. Secondly, we focus on developing a recommendation system using the Random Forest algorithm for classifications [4]. This includes building a model that can predict the most appropriate streaming platform for a given TV-show, based on its attributes. The model will strengthen streaming platforms to curate selected content suggestions for their users, thereby enhancing user satisfaction and retention.

To achieve our objectives, we are making use of a powerful technological stack. Python, joined with PySpark [5], forms the central of our data processing and analysis pipeline. PySpark enables efficient processing of large amounts of data and enables seamless transformations and calculations. We also use the data visualization capabilities of Tableau, this allows us to create dashboards that display key information from our model output EDA.

The structure of this document is as follows. Section 2 analyzes existing studies in the literature that used sequential and parallel recommendation techniques. Section 3 introduces the data set. Section 4 presents the methodology for conducting the experiments. Section 5 presents the experimental setup of the system. Section 6 shows the results obtained. Section 7 concludes the document with a conclusion, future directions and social implications of this project.

**II. RELATED WORKS**

Many research works have been carried out addressing the use of machine learning algorithms and content-based techniques for video recommendation systems. This section summarizes previous research works to present the existing literature. The summaries are ordered by parallel and sequential implementation.

Reference [6] propose a new content-based recommender system that encompasses a technique to automatically analyze video contents and to extract a set of representative stylistic features (lighting, color, and motion) grounded on existing approaches of Applied Media Theory. The study explores the utilization of automatically extracted visual features from videos within recommender systems, introducing a novel content-based approach. The proposed system employs an automated analysis of video content to extract distinctive stylistic features like lighting, color, and motion, grounded in Applied Media Theory. Evaluated against relevance metrics and compared to existing content-based systems using explicit features like genre, the new technique demonstrates enhanced accuracy in recommendations. Notably, it outperforms even when applied to movie trailers, making it effective for cases without full-length videos or metadata. The recommender can balance traditional techniques or stand alone, addressing the challenge of recommending videos lacking metadata, as often faced on platforms like YouTube with vast user-generated contents.

Reference [8] presents a Recommendation for Live-Streaming Platforms. Because of the challenges faced by Live streaming platforms for recommendation systems due to dynamic contents availability and distinct user behaviors. They propose LiveRec, a self-attentive model that personalizes item ranking based on both historical interactions and current availability. They also show that carefully modelling repeat consumption plays a significant role in model performance. Notably, repeat channel consumption is crucial despite the fleeting nature of content. The released dataset, featuring 475M Twitch interactions over 43 days, validates LiveRec's efficacy. It shows how the system outperforms strong baselines in ranking currently accessible content, displaying its likelihood for personalized live-streaming recommendations.

Content-based RSs create a profile of a user’s preferences, interests and tastes by considering the feedback provided by the user to some items together with the content associated to them. Feedback can be gathered either explicitly from users, by explicitly asking them to rate an item [11], or implicitly by analyzing her activity [12]. The user profile is then matched against the features of all objects to generate recommendations. Keyword-based models can be used to represent content, in which the recommender establishes a Vector Space Model (VSM) representation of item features, in which an item is represented by a vector in a multidimensional space. These dimensions represent the features used to describe the items. By means of this representation, the system measures a relevance score that represents the user’s degree of interest toward any of these items [13]. For instance, in the movie domain, the features that describe an item can be genre, actors, or director. This model may allow content-based recommender systems to naturally tackle the new item problem [14]. Other content-based RS families use semantic analysis (lexicons and ontologies) to construct more accurate item representations. Multiple kinds of content-based recommendation algorithms have been proposed in the literature. A classic example is the "k-nearest neighbor" approach (KNN), which computes a user's preference for an unknown item by comparing it to all the things accessible to the user in the catalog. Every known item contributes to predicting the preference score according to its similarity with the unknown item. The similarity can be measured by typically using Cosine similarity [12].

**III. DATASET**

The dataset used for this research is the Netflix Disney+ Prime Video Hulu Shows Collection Data Set from Kaggle [2], which consists of 11 columns and 5369 rows. The data collected comprises a comprehensive list of shows available on various video streaming platforms. The authors of the dataset shed light on the key attributes that have been proven solid and efficient in checking which streaming platform(s) the user can watch their tv-shows on, the target age group tv-shows vs the streaming application they can be found on, and the year during which a tv show was produced and the streaming platform they can be found on. While proposing some new features, experimentally assign new rules to some well-known features and update some other features. Table 1 presents the attributes and short descriptions.

Table 1. Dataset Attributes and short description.

|  |  |
| --- | --- |
| **Column name (Features)** | **Short description** |
| ID: | Unique identifier for each show. |
| Year | The year of release. |
| Age | The recommended age group for viewers. |
| IMDb | IMDb rating of each show. |
| Rotten Tomatoes | Rotten Tomatoes rating of the show. |
| Netflix | Binary indicator (1 or 0) for availability on Netflix. |
| Disney+ | Binary indicator (1 or 0) for availability on Disney+. |
| Prime Video | Binary indicator (1 or 0) for availability on Prime Video. |
| Hulu | Binary indicator (1 or 0) for availability on Hulu. |
| Type | Denotes whether the entry is a TV show or a movie. |
| Title | The title of the show. |

The data processing involves converting categorical variables, handling missing values, and preparing the dataset for analysis. The provided attributes offer valuable insight about each show's platform availability, ratings, and genre, which will add to the subsequent analysis and recommendation system development.

**IV. METHODOLOGY**

The systematic method to achieving the project objectives involved several steps and techniques. Firstly, a virtual environment was created and Ubuntu was installed on Microsoft Windows 11 using a virtual machine. This environment provided the necessary platform for continuous data analysis. Afterward, with the python3 installed on Ubuntu by default, essential components such as Java-JDK, PySpark and Hadoop was downloaded and installed within the Ubuntu setup. These tools were crucial for efficient data processing and analysis. Figures of the installations are presented in the appendix section

To analyze the dataset, the power of PySpark was leveraged [5], a robust data processing framework. PySpark allows handling of large-scale data operations effective, making sure of optimal performance throughout the analysis. Using the Random Forest algorithm for classification was a pivotal choice. This algorithm is known for its versatility in handling complex datasets and providing accurate results.

Tableau was also used, a powerful data visualization software, to explore and visualize the dataset comprehensively. Tableau enabled the creation of insightful visual representations, helping in the identification of patterns, trends, and correlations inside the data.

**V. EXPERIMENTAL SECTION**

The below presented program outlines the experimental procedure for building and evaluating the recommendation system using PySpark, precisely focusing on the processing of the tv-show dataset. Here is a breakdown of the code and its steps:

**Importing Necessary Libraries:** The required PySpark classes and modules are imported, including those for recommendation, data functions, classification, evaluation, and machine learning operations.

**Initializing Spark Session:** A Spark session named "TVShowRecommendation" is generated, which serves as the entry point for interacting with Spark functionalities.

**Loading the Dataset:** The TV-show dataset is loaded from the indicated CSV file using spark.read.csv(). A subset of the data (500 rows) is selected for demonstration purposes and the graphical representation of the distribution of the four tv\_show sites using matplotlib.

**Text Processing**: The title column is split into individual words using a user-defined function (UDF) and the explode functionality is used to create a new row for each word in the original title column. This step prepares the data for more analysis.

**Indexing Words:** The split words are indexed using the stringIndexer to transform categorical words into numerical indices.

**Feature Vector Assembly:** The indexed words are assembled into a feature vector using VectorAssembler.

**Splitting Dataset:** The dataset is split into testing and training sets (80% training, 20% testing) using randomSplit.

**Random Forest Classifier**: A Random Forest classifier is instantiated, specifying the input and output columns.

**Evaluation:** The evaluation metric is set to "accuracy" and the MulticlassClassificationEvaluator was used.

**Hyperparameter Grid:** A grid of hyperparameters for the Random Forest classifier was defined using ParamGridBuilder.

**Cross-Validation Pipeline:** A pipeline is defined, alongside the Random Forest classifier.

**Cross-Validator:** A CrossValidator is instantiated, specifying the pipeline, hyperparameter grid, evaluator, and the number of folds for cross-validation.

**Running Cross-Validation:** The cross-validation is performed using crossval.fit(training), and the best set of parameters is chosen based on the identified evaluation metric.

**Making Predictions:** Predictions are made on the testing set using the trained model.

**Model Evaluation:** The model's accuracy is evaluated on the testing predictions using the defined evaluator.

**Recommendation System Function:** A function recommend\_platforms(title) is defined to recommend streaming platforms for a user provided TV show title. It returns the top 3 recommended streaming platforms.

**Testing the Recommendation System:** The recommendation function is tested by providing a TV show title ("The Lion King"). The top recommended streaming platforms are printed.

**Stopping Spark Session:** The Spark session is stopped to release the resources.

This code exhibits a systematic approach to processing and analyzing a TV show dataset, including text processing, feature engineering, classifier training, and model evaluation using PySpark. The Random Forest algorithm is utilized in a cross-validation framework to build the recommendation system. The final accuracy score of the model is printed for evaluation.

Tableau was used to explore the dataset. Graphical reports were generated by plotting the following features of the dataset.

1. Year column against the movie sites (Netflix, Prime video, Hulu, Disney+). It depicts the rate at which movies are streamed within each decade.

2. Rotten Tomatoes rating against the movie sites. This shows the kind of rating each movie site got from the users.

**VI. RESULT DISCUSSION**

The project includes data collection, preprocessing, analysis, and modeling using PySpark, Tableau, and the Ubuntu virtual machine. In this section, we evaluate the Streaming Platform Recommendation System. The accuracy of the prediction using the Multi-Classification-Evaluator(MCE) is computed. MCE is a class in PySpark (part of Apache Spark) that is used to evaluate the performance of a multi-class classification model. It provides methods to compute various evaluation metrics for multi-class classification tasks. This class is typically used to assess the quality of predictions made by a machine learning models that performs multi-class classification [9].

The dataset, containing tv-show features like title, ratings, and platform availability, provided the foundation. Through exploratory data analysis, we discovered distribution patterns, contributing insights into viewer preferences. Notably, the Random Forest algorithm was employed for classification, enhancing the recommendation system's accuracy by predicting optimal streaming platforms. Figure 1 presents the distribution pattern of the viewing platforms with histogram (Netflix, Hulu, Disney+ and Prime Video).

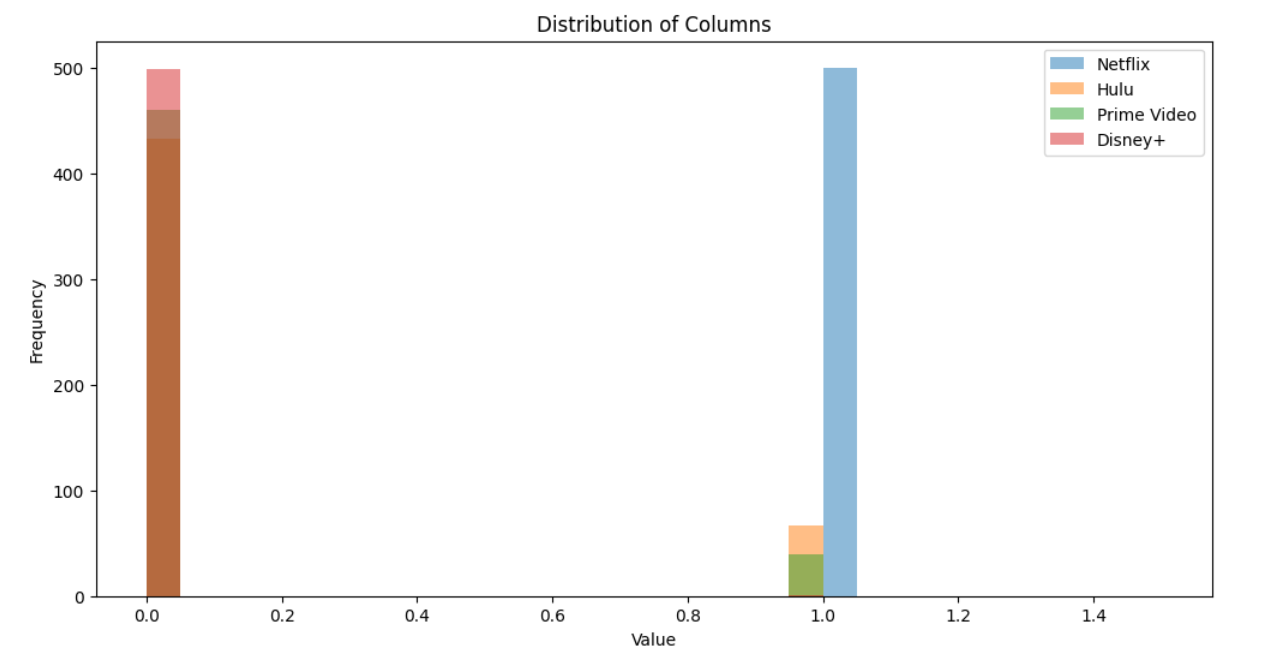


Figure 1. Distribution of the classes of the viewing platforms

From the above figure, the histogram shows of the first 500 rows of the dataset for the viewing platforms. The features are classifies to 1 and 0. 1 representing the availability of the searched tv-show and 0 meaning the show is not present on the platform.

After the evaluation of the system accuracy using the tv-shows dataset, the result shows a 96.41% accuracy using the Multi-Classification-Evaluator.

The experimental section displayed the intricate steps of data preprocessing, text analysis, indexing, and feature engineering. The appendix presents the screenshots of how the programs were installed, the code used for the experiment and snapshots of Tableau visualizing how the dataset was explored.

**VII. CONCLUSION & FUTURE WORKS**

In conclusion, our experimental journey spun around a meticulous protocol, leveraging PySpark, Tableau, and the Ubuntu virtual machine, to construct a robust Streaming Platform Recommendation System. The code snippet was broken down and gives proper clarification of how the system operates, from importing all necessary libraries to initializing the spark session to data processing, training of the model, and evaluation. The Random forest algorithm was used and it has shown that it can improve the recommendation system accuracy.

More light has been shed while analyzing the television shows dataset which reveals the distribution pattern with regards to viewer preferences. Notably, the accuracy evaluation, conducted using the MultiClassification-Evaluator, showcased a 96.41% accuracy, signifying the system's effectiveness.

The step-by-step analysis highlighted the significance of text processing, feature engineering, and cross-validation in optimizing the recommendation model. This approach displayed the intricate interplay between data analysis techniques and machine learning algorithms, concluding in a highly accurate recommendation system.

In the broader context, this project demonstrated the potential of data-driven techniques to revolutionize content recommendations, providing a foundation for enhancing user experiences through streaming platforms. Further exploration could explore into real-world deployment and user feedback to fine-tune the system's performance.

**IIX. SOCIAL IMPACT OF THIS EXPERIMENT**

The social impact of this project is extensive, as it directly addresses the modern challenges of content discovery and viewer engagement on streaming platforms. By developing a Streaming Platform Recommendation System for TV-shows, this project improves user experiences, making entertainment consumption more personalized and efficient.

Furthermore, the implementation of such a recommendation system contributes to the sustainability of streaming platforms by enhancing user satisfaction and retention. Viewers are more likely to continue using platforms that cater to their comforts, leading to long term relationships and reduced churn rates. Ultimately, this project has the possibility to reshape the way users interact with digital entertainment, promoting a more enjoyable and fulfilling viewing experience while driving positive economic results for streaming platforms.

**REFERENCES**

1. Ashton Anderson, Ravi Kumar, Andrew Tomkins, and Sergei Vassilvitskii. 2014. The dynamics of repeat consumption. In Proceedings of the 23rd international conference on World wide web. 419–430.
2. Netflix Disney+ Prime Video Hulu Shows Collection <https://www.kaggle.com/datasets/whenamancodes/netflix-prime-video-disney-hulu>
3. What is exploratory data analysis? <https://www.ibm.com/topics/exploratory-data-analysis>
4. Understand Random Forest Algorithms With Examples (Updated 2023) <https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>
5. Getting started with Spark. <https://www.datacamp.com/tutorial/pyspark-tutorial-getting-started-with-pyspark>
6. Yashar Deldjoo 2021. Content-based Video Recommendation System based on Stylistic Visual Features. <https://core.ac.uk/download/pdf/74313442.pdf>
7. (A consumer perspective on Netflix’s recommender system. A qualitative analysis) Aguirre, E., Mahr, D., Wetzels, M., Grewal, D. and Ruyter, K. (2015). “Unraveling the Personalization Paradox: The Effect of Information Collection and Trust-Building Strategies on Online Advertisements Effectiveness. Journal of Retailing, vol. 91 No. 1, pp. 34-49. <https://doi.org/10.1016/j.jretai.2014.09.005>
8. Recommendation on Live-Streaming Platforms: Dynamic Availability and Repeat Consumption. 27-October 1, 2021, Amsterdam, Netherlands. ACM, New York, NY, USA. <https://doi.org/10.1145/3460231.3474267>
9. Multi-classification-Evaluator <https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.evaluation.MulticlassClassificationEvaluator.html>
10. C. C. Aggarwal, Recommender Systems - The Textbook, Springer, 2016. doi:10.1007/

978-3-319-29659-3.

1. Billsus, D., Pazzani, M.J.: A hybrid user model for news story classification. Springer (1999)
2. Kelly, D., Teevan, J.: Implicit feedback for inferring user preference: a bibliography. In: ACM SIGIR Forum, vol. 37, pp. 18–28. ACM (2003)
3. Lops, P., De Gemmis, M., Semeraro, G.: Content-based recommender systems: State of the art and trends. In: Recommender systems handbook, pp. 73–105. Springer (2011)
4. Elahi, M., Ricci, F., Rubens, N.: Active learning strategies for rating elicitation in collaborative filtering: a system-wide perspective. ACM Transactions on Intelligent Systems and Technology (TIST) 5(1), 13 (2013)
5. Ahn, J.w., Brusilovsky, P., Grady, J., He, D., Syn, S.Y.: Open user profiles for adaptive news systems: help or harm? In: Proceedings of the 16th international conference on World Wide Web, pp. 11–20. ACM (2007)

**APPENDIX**

Below in the installation process of the Pyspark/Hadoop Screenshots.

Figure 2. Complete installation of Ubuntu on the virtual machine environment.

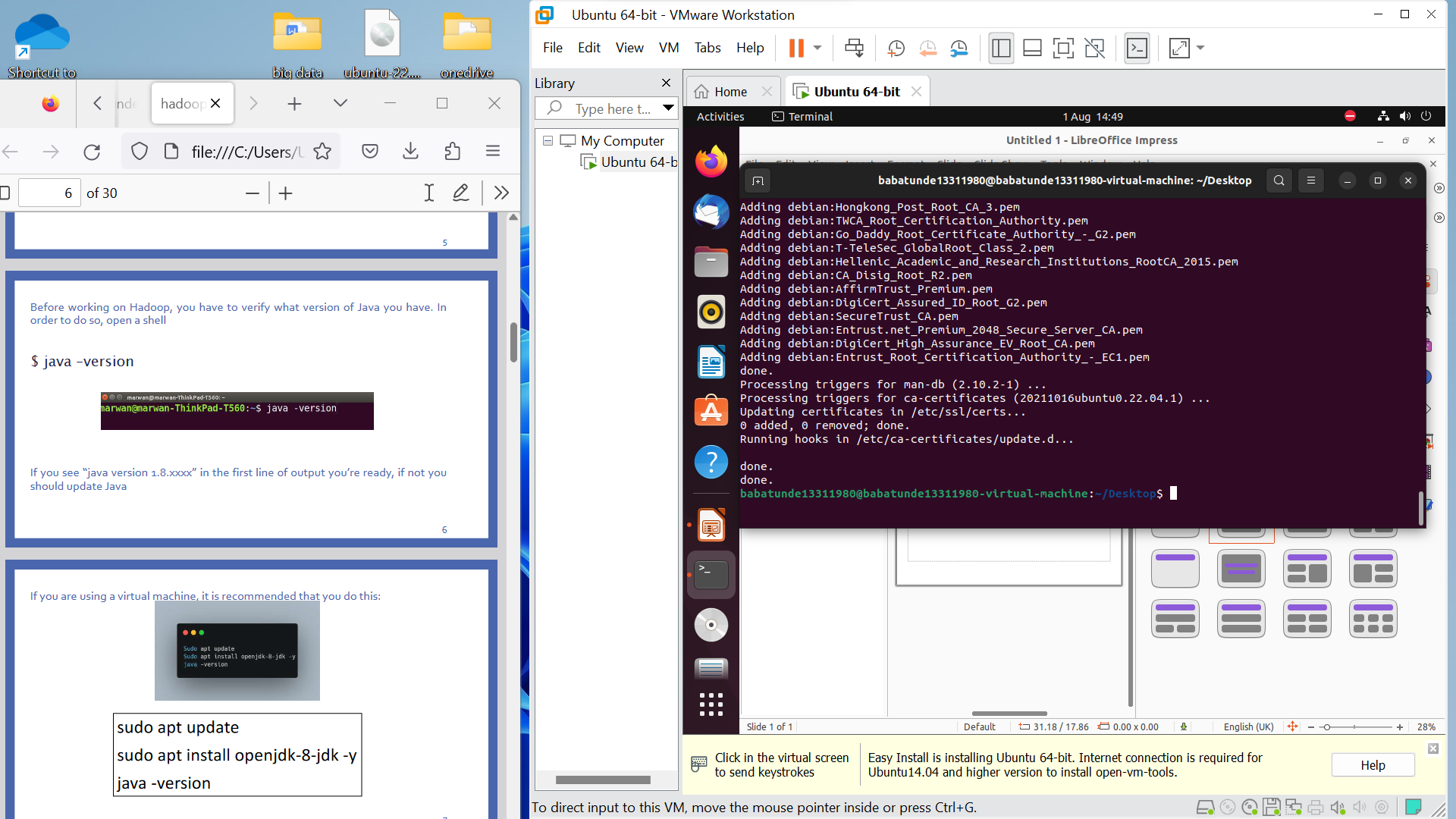


Figure 3. Installing Java jdk for Hadoop and spark support.

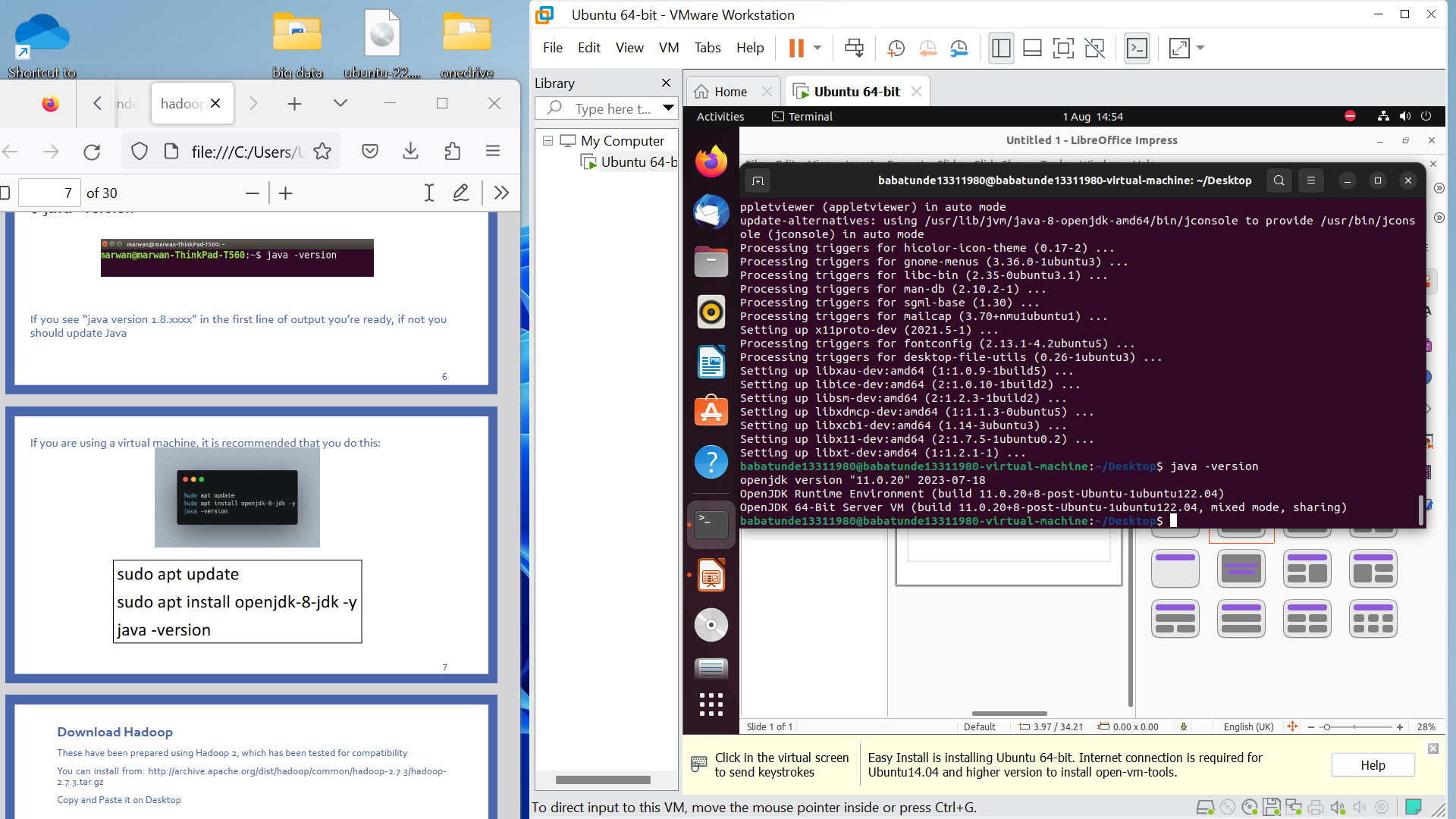


Figure 4. Installing Hadoop

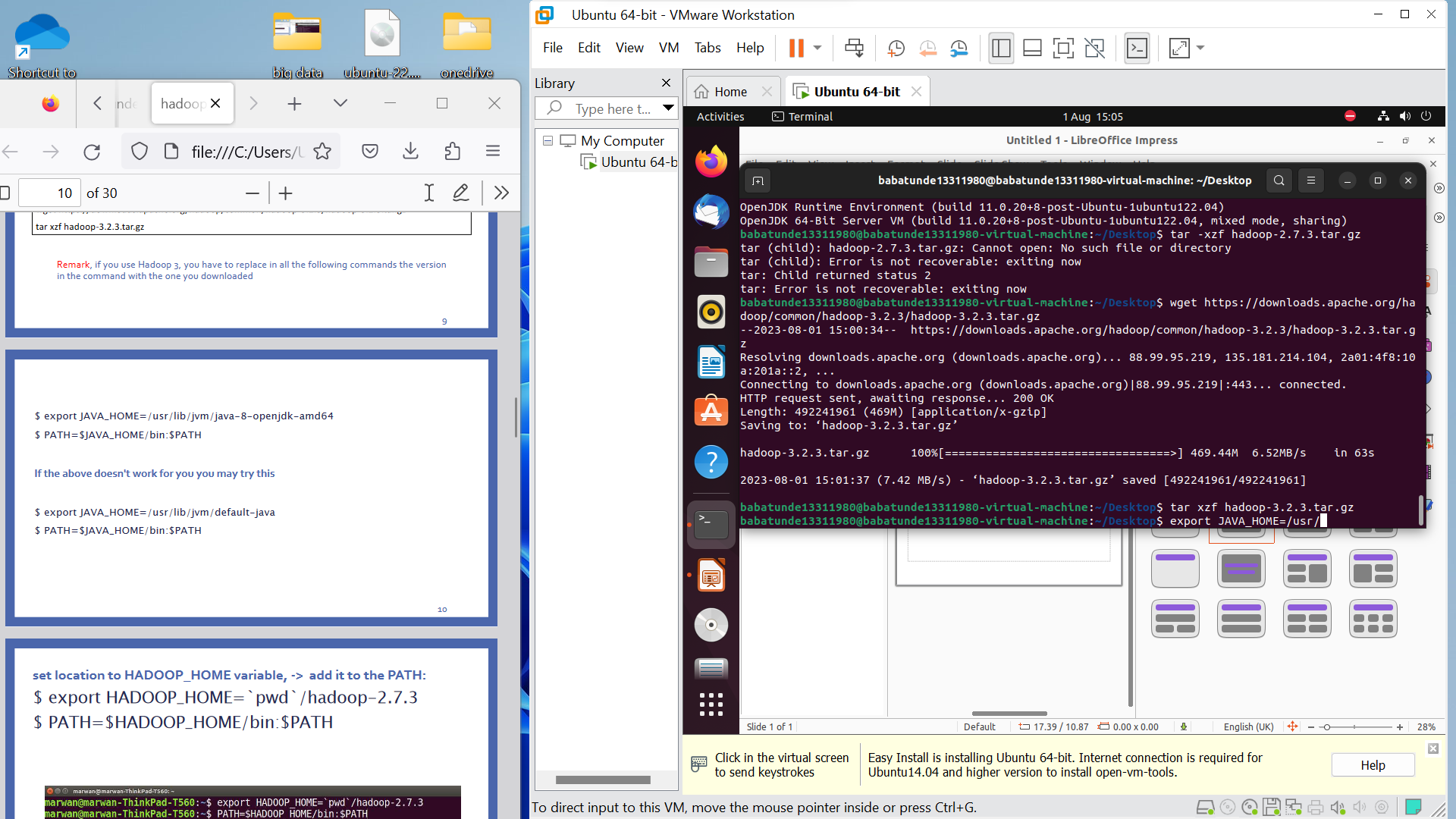


Figure 5. Hadoop and Spark set up.

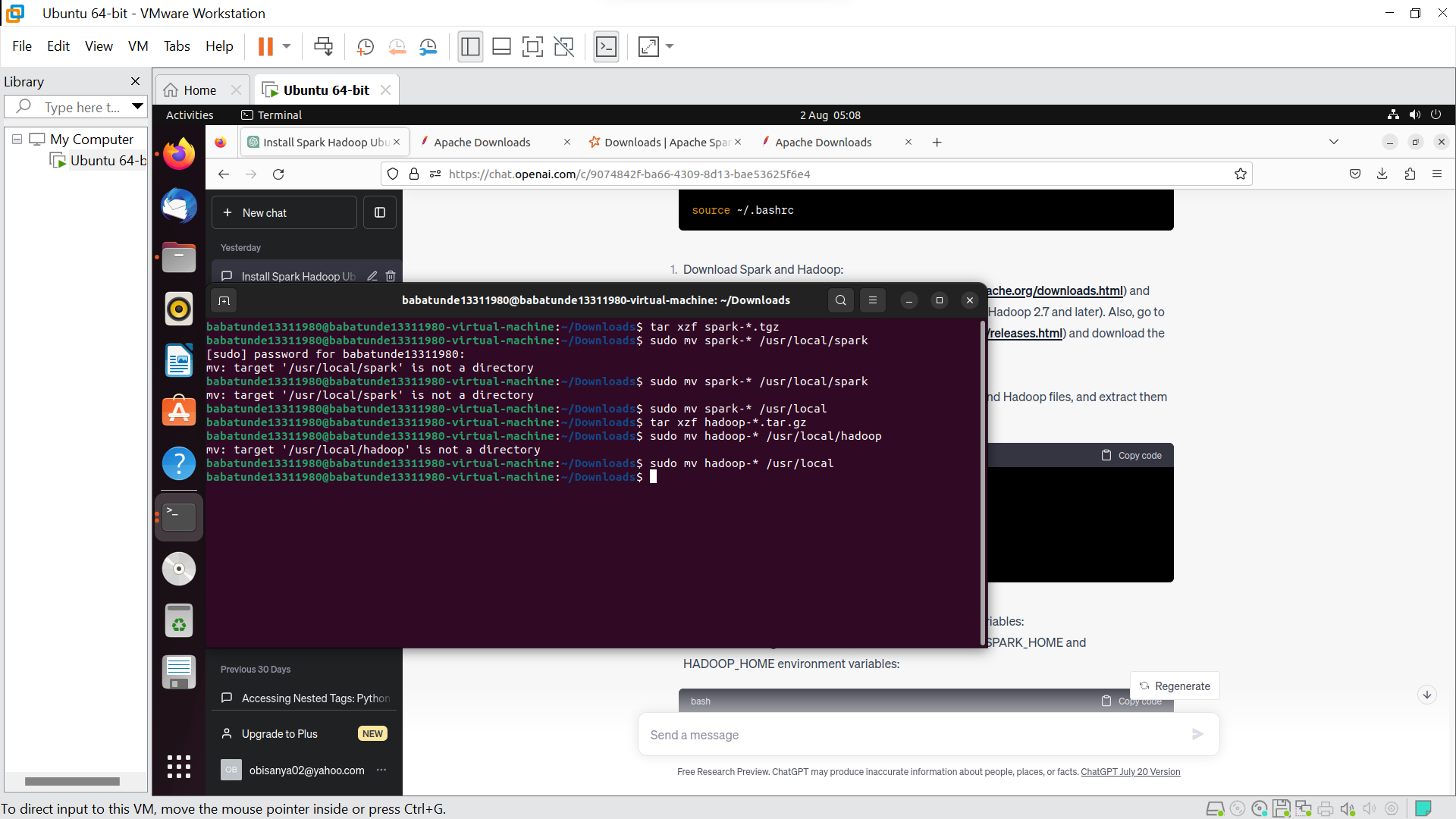


Figure 6. Hadoop and spark configuration.

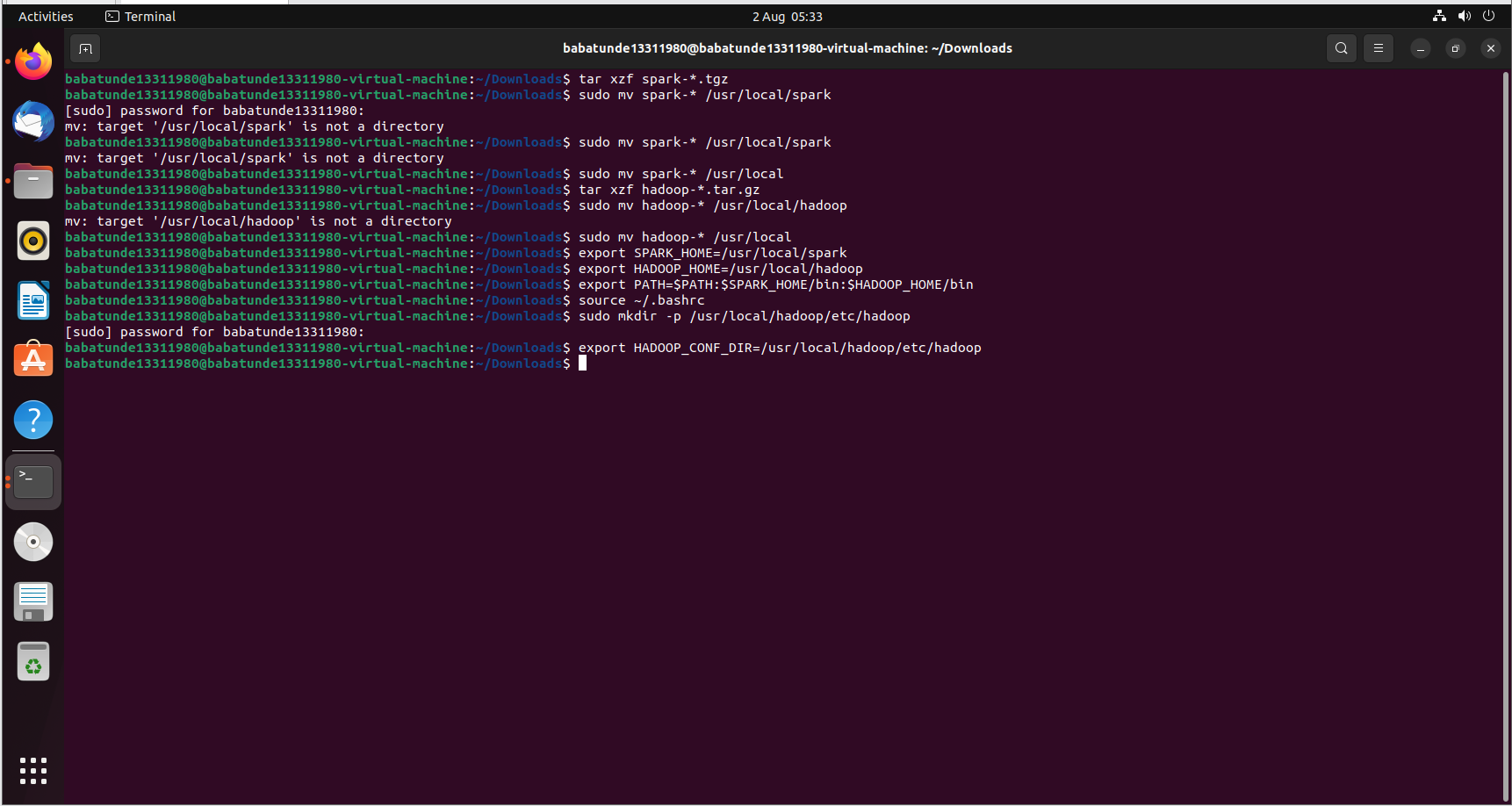


Figure 7. Spark Installation Complete.

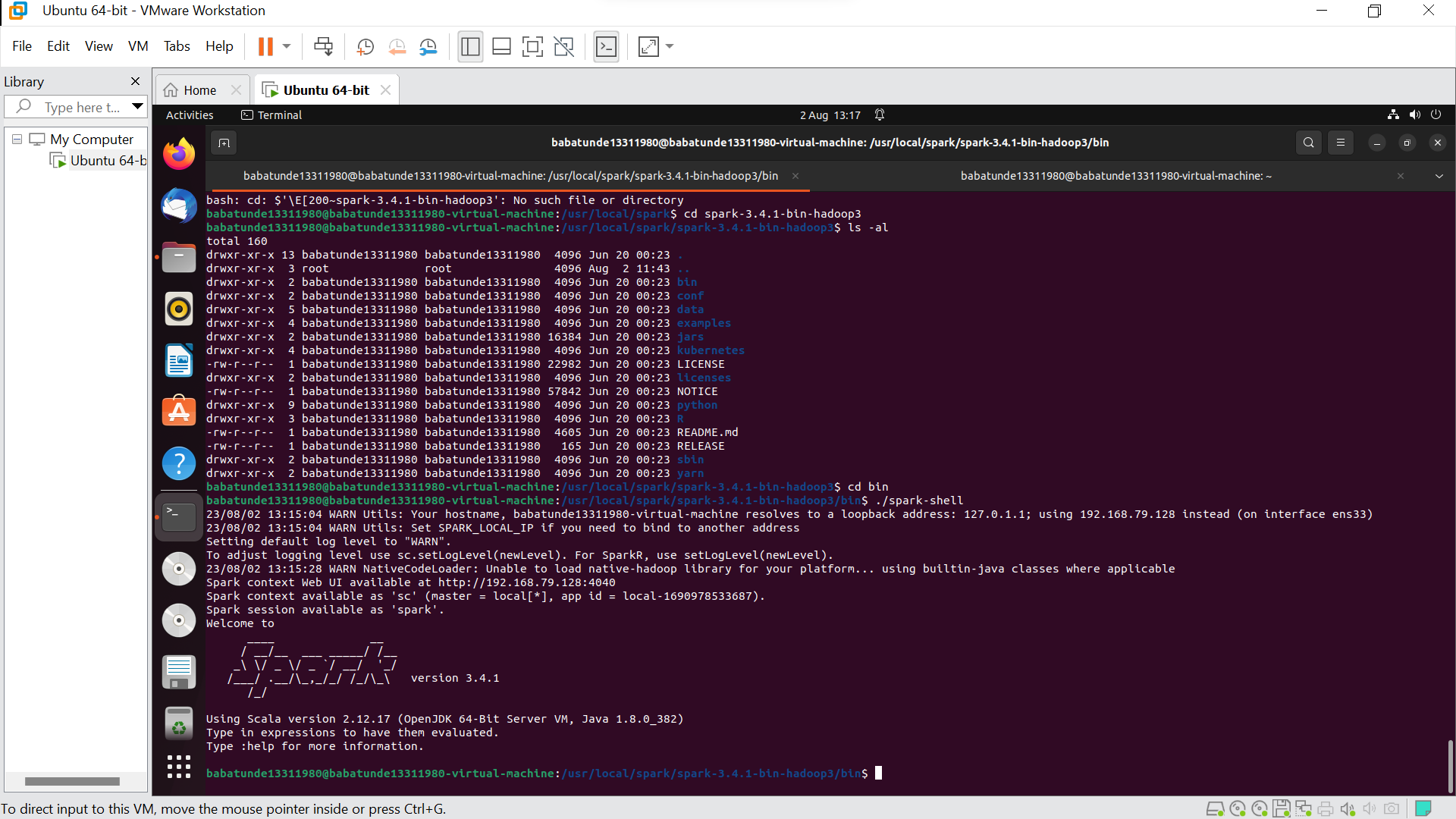
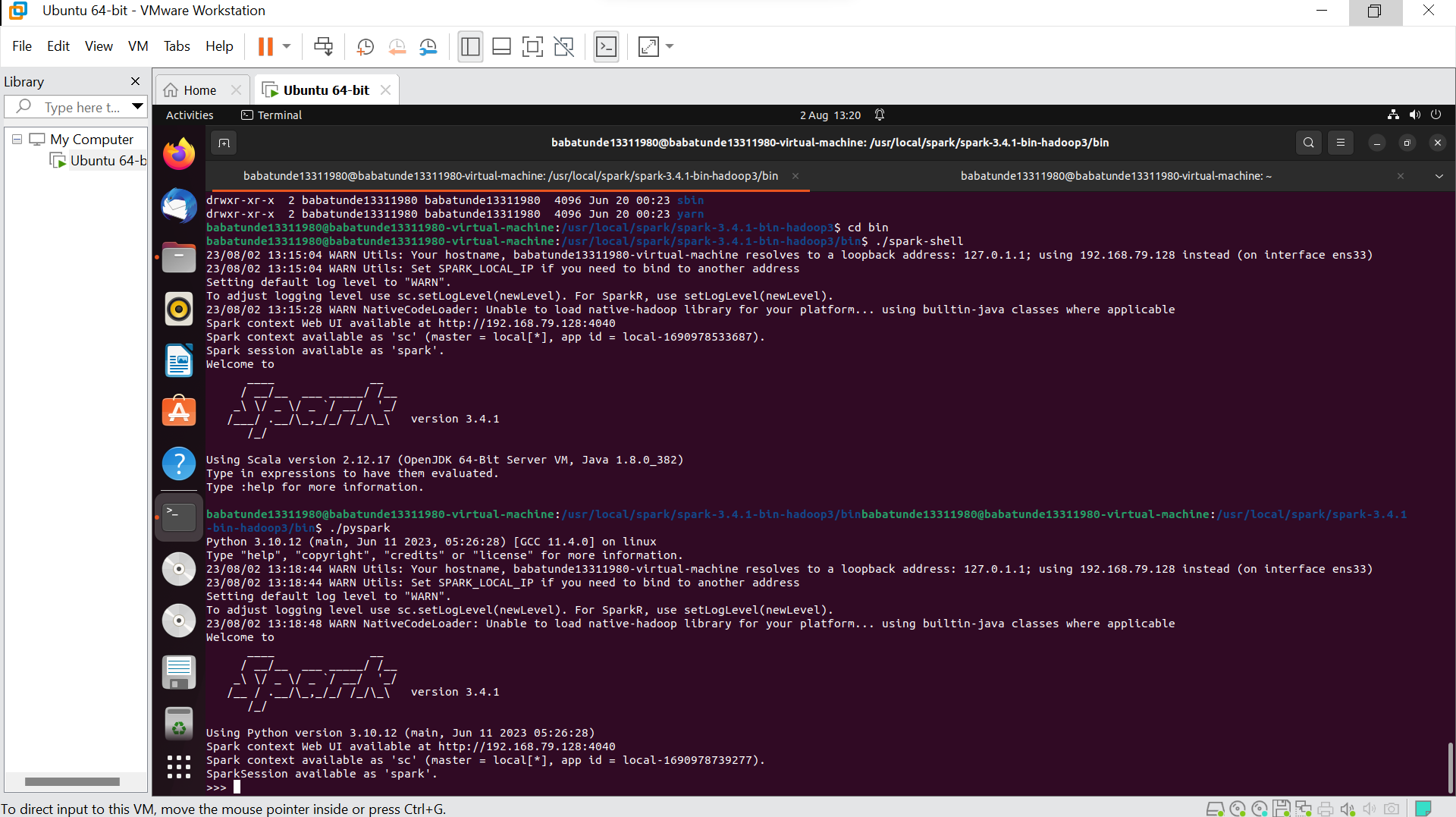


Figure 8.



Code used for the experiment.

**from** **pyspark.sql.functions** **import** explode

**from** **pyspark.sql** **import** SparkSession

**from** **pyspark.ml.recommendation** **import** ALS

**from** **pyspark.ml.classification** **import** RandomForestClassifier

**from** **pyspark.ml.evaluation** **import** MulticlassClassificationEvaluator

**from** **pyspark.sql.functions** **import** split, udf

**from** **pyspark.sql.types** **import** ArrayType, StringType

**from** **pyspark.ml.feature** **import** StringIndexer, VectorAssembler

**from** **pyspark.ml.classification** **import** RandomForestClassifier

**from** **pyspark.ml.evaluation** **import** MulticlassClassificationEvaluator

**from** **pyspark.ml.tuning** **import** CrossValidator, ParamGridBuilder

**from** **pyspark.ml** **import** Pipeline

**import** **matplotlib.pyplot** **as** **plt**

**from** **pyspark.ml.feature** **import** OneHotEncoder

**import** **os**

**import** **sys**

os.environ['PYSPARK\_PYTHON'] = sys.executable

os.environ['PYSPARK\_DRIVER\_PYTHON'] = sys.executable

# Initialize Spark session

spark = SparkSession.builder.appName("TVShowRecommendation").getOrCreate()

# Load the dataset

data = spark.read.csv("/content/tv\_shows.csv", header=True, inferSchema=True)

data = data.limit(**50**)

data.show()

# Select columns for distribution

columns\_to\_plot = ["Netflix", "Hulu", "Prime Video", "Disney+"]

# Collect the data to the driver

data\_collect = data.select(\*columns\_to\_plot).toPandas()

# Plot the distributions using matplotlib

plt.figure(figsize=(**12**, **6**))

**for** col **in** columns\_to\_plot:

plt.hist(data\_collect[col], bins=**20**, alpha=**0.5**, label=col)

plt.title("Distribution of Columns")

plt.xlabel("Value")

plt.ylabel("Frequency")

plt.legend()

plt.show()

# Split the title column into individual words

words = udf(**lambda** x: x.split(), ArrayType(StringType()))

tv\_shows = data.withColumn("words", words(data.Title))

# Explode the array of words to create a row for each word

tv\_shows\_exploded = tv\_shows.select("ID", "words", "Title")

tv\_shows\_exploded = tv\_shows\_exploded.withColumn("word", explode("words"))

# Index the words using StringIndexer

string\_indexer = StringIndexer(inputCol="word", outputCol="indexed\_words")

model = string\_indexer.fit(tv\_shows\_exploded)

indexed\_words = model.transform(tv\_shows\_exploded)

**print**(indexed\_words)

# Apply OneHotEncoder to the indexed words

encoder = OneHotEncoder(inputCol="indexed\_words", outputCol="encoded\_words")

encoder.setDropLast(False)

ohe = encoder.fit(indexed\_words) # indexer is the existing dataframe, see the question

encoded\_words = ohe.transform(indexed\_words)

# Assemble the indexed words into a feature vector

assembler = VectorAssembler(inputCols=["encoded\_words"], outputCol="features")

#indexed\_words = assembler.transform(indexed\_words)

encoded\_words\_assembled = assembler.transform(encoded\_words)

# Split the dataset into training and testing sets

#(training, testing) = indexed\_words.randomSplit([0.8, 0.2])

(training, testing) = encoded\_words\_assembled.randomSplit([**0.8**, **0.2**])

# Define the random forest classifier

rf = RandomForestClassifier(labelCol="ID", featuresCol="features")

# Define the evaluation metric

evaluator = MulticlassClassificationEvaluator(labelCol="ID", predictionCol="prediction", metricName="accuracy")

# Define the hyperparameter grid for cross-validation

param\_grid = ParamGridBuilder().addGrid(rf.numTrees, [**10**, **50**, **100**]).addGrid(rf.maxDepth, [**2**, **5**, **10**]).build()

# Define the cross-validation pipeline

pipeline = Pipeline(stages=[rf])

# Create a CrossValidator

crossval = CrossValidator(

estimator=pipeline,

estimatorParamMaps=param\_grid,

evaluator=evaluator,

numFolds=**3**

)

# Run cross-validation and choose the best set of parameters

cv\_model = crossval.fit(training)

# Make predictions on the testing set

predictions = cv\_model.transform(testing)

# Evaluate the model

accuracy = evaluator.evaluate(predictions)

**print**("Accuracy:", accuracy)

# Define a function to recommend streaming platforms for a given TV show

**def** **recommend\_platforms**(title):

# Split the title into individual words

words = title.split()

# Index the words

indexed\_wrds = model.transform(spark.createDataFrame([(words,)], ["words"])).select("indexed\_words").first()[**0**]

# Assemble the feature vector

features = assembler.transform(spark.createDataFrame([(indexed\_wrds,)], ["indexed\_wrds"])).select("features").first()[**0**]

# Make a prediction using the trained model

prediction = model.transform(spark.createDataFrame([(features,)], ["features"])).select("prediction").first()[**0**]

# Get the top 3 streaming platforms with the highest probabilities

platforms = indexed\_wrds.zip(prediction).filter(**lambda** x: x[**1**] > **0**).sortBy(**lambda** x: x[**1**], ascending=False).take(**3**)

# Return the names of the top 3 streaming platforms

**return** [model.stages[-**1**].labels[int(index)] **for** (index, prob) **in** platforms]

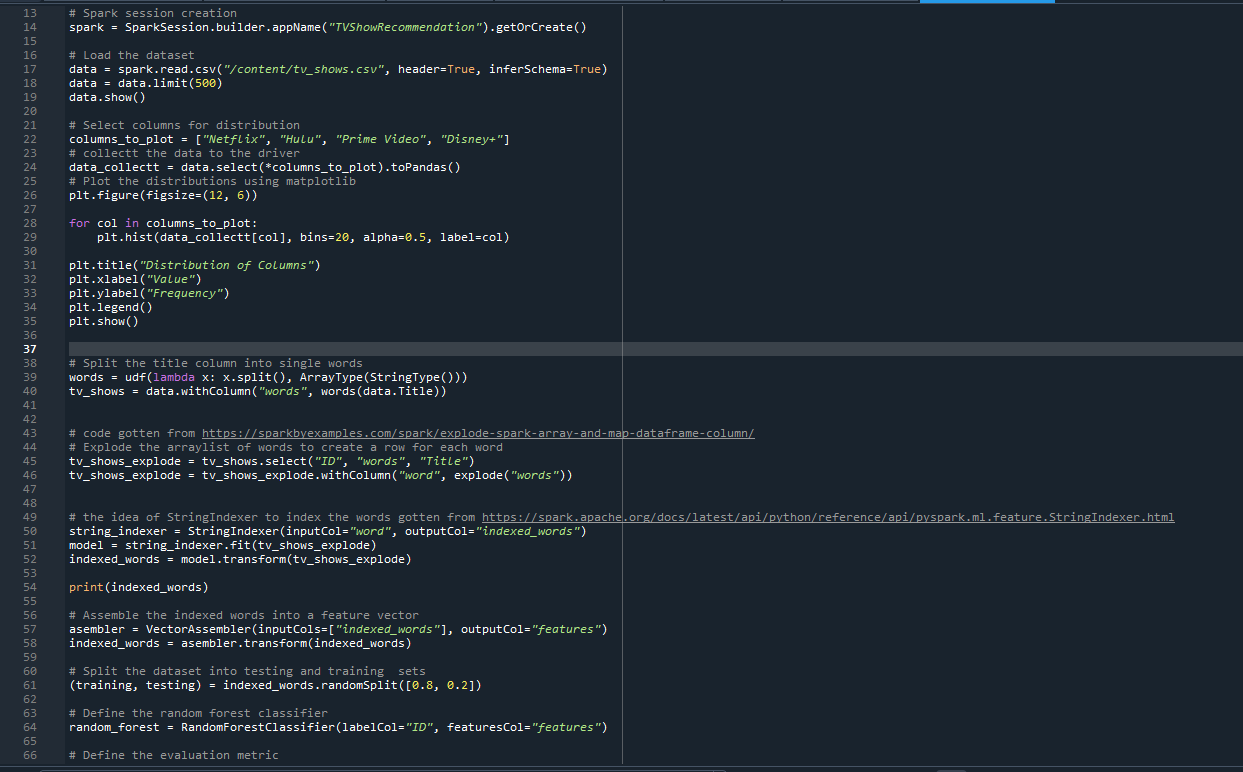
# Test the recommendation system

**print**(recommend\_platforms("The Lion King"))

# Stop the Spark session

spark.stop()

Figure 9.



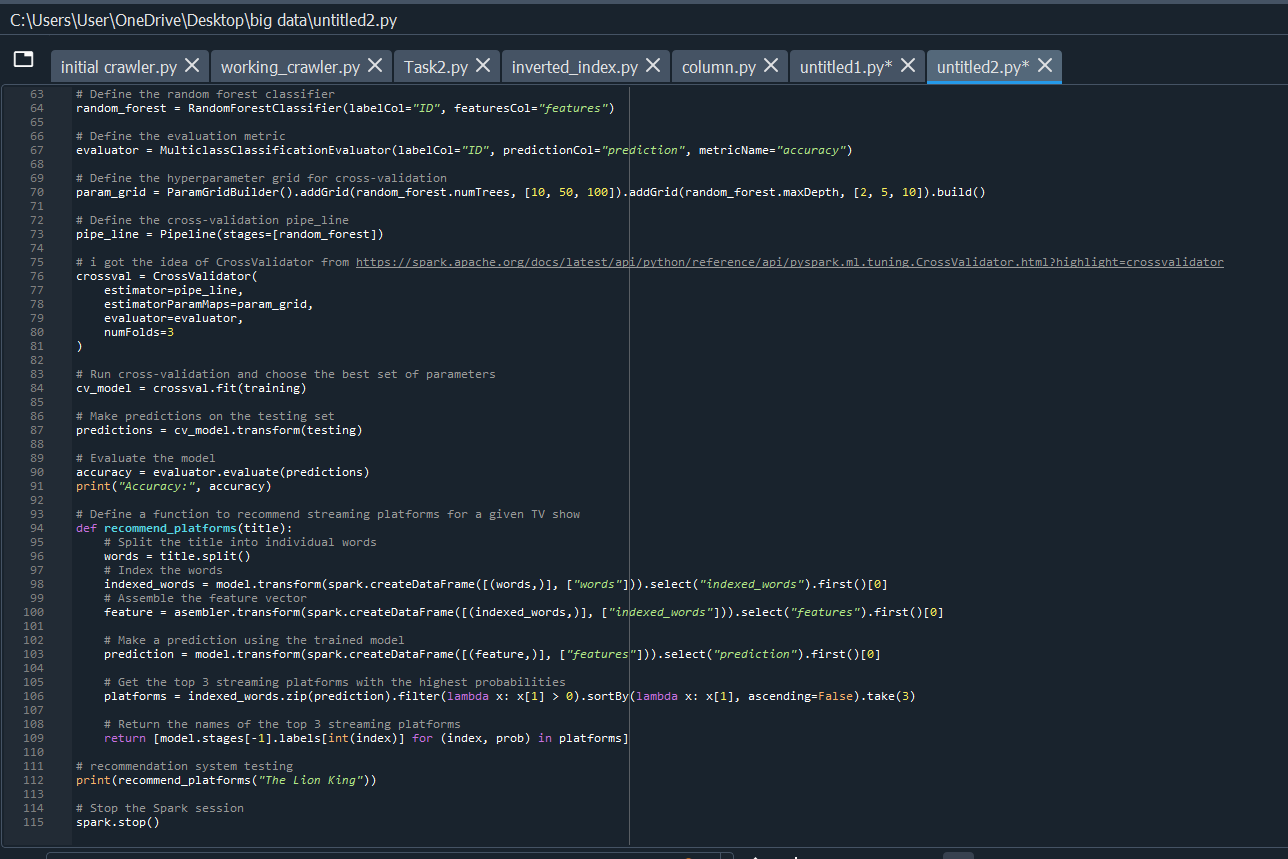


Figure 10. Tableau complete installation

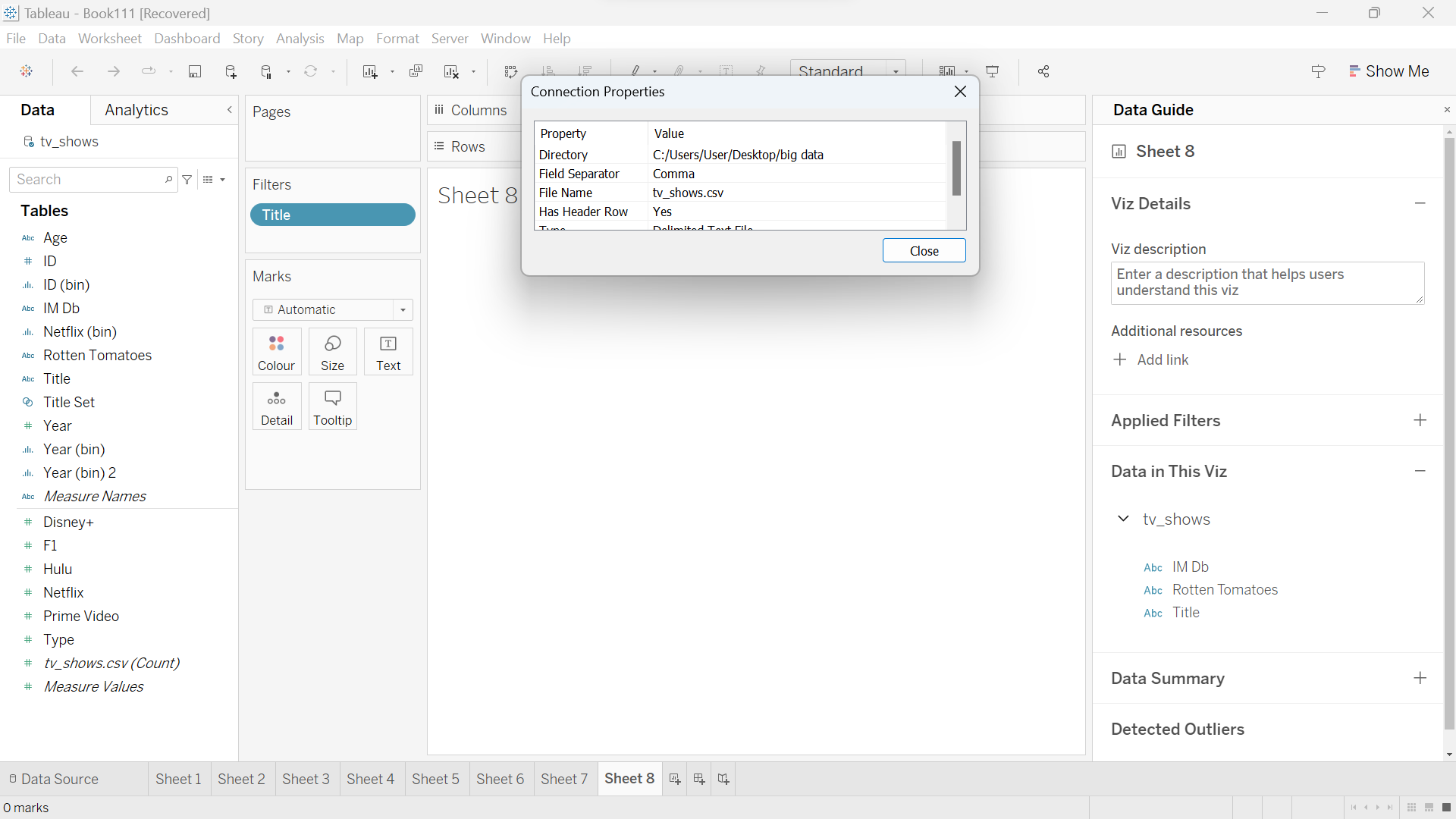


Figure 10 presents the complete installation of the tableau tool showing the tv\_shows dataset ready to be explored.

Figure 11. Yearly report of amount of tv-shows per streaming platform.

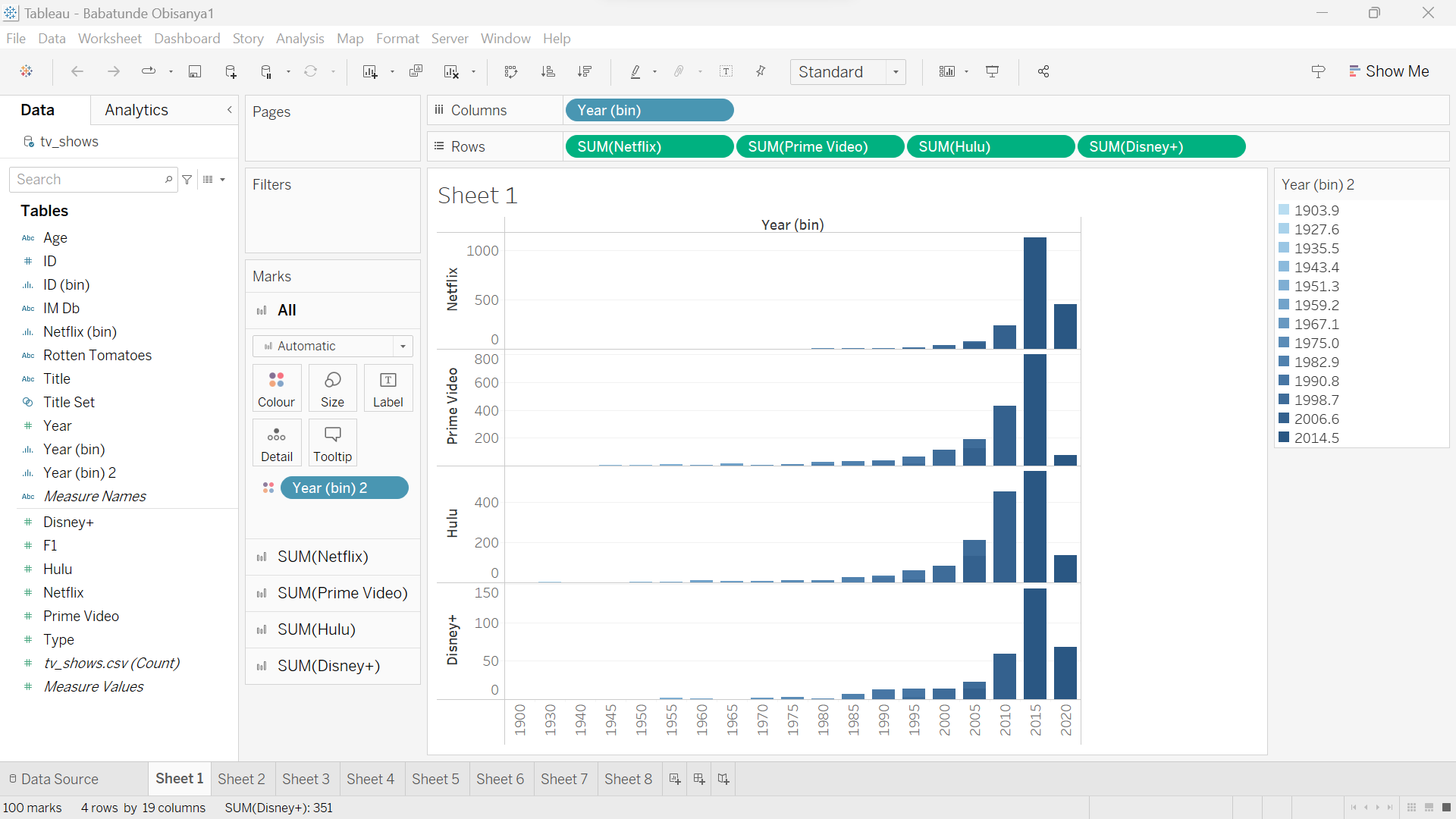


Figure 11 presents the yearly report visualization generated from the dataset. 2015 happens to be the year with the most amount of tv-shows uploads on all four streaming sites using the data gotten from the dataset.

Figure 12. Rotten tomato rating with respect to the streaming platforms

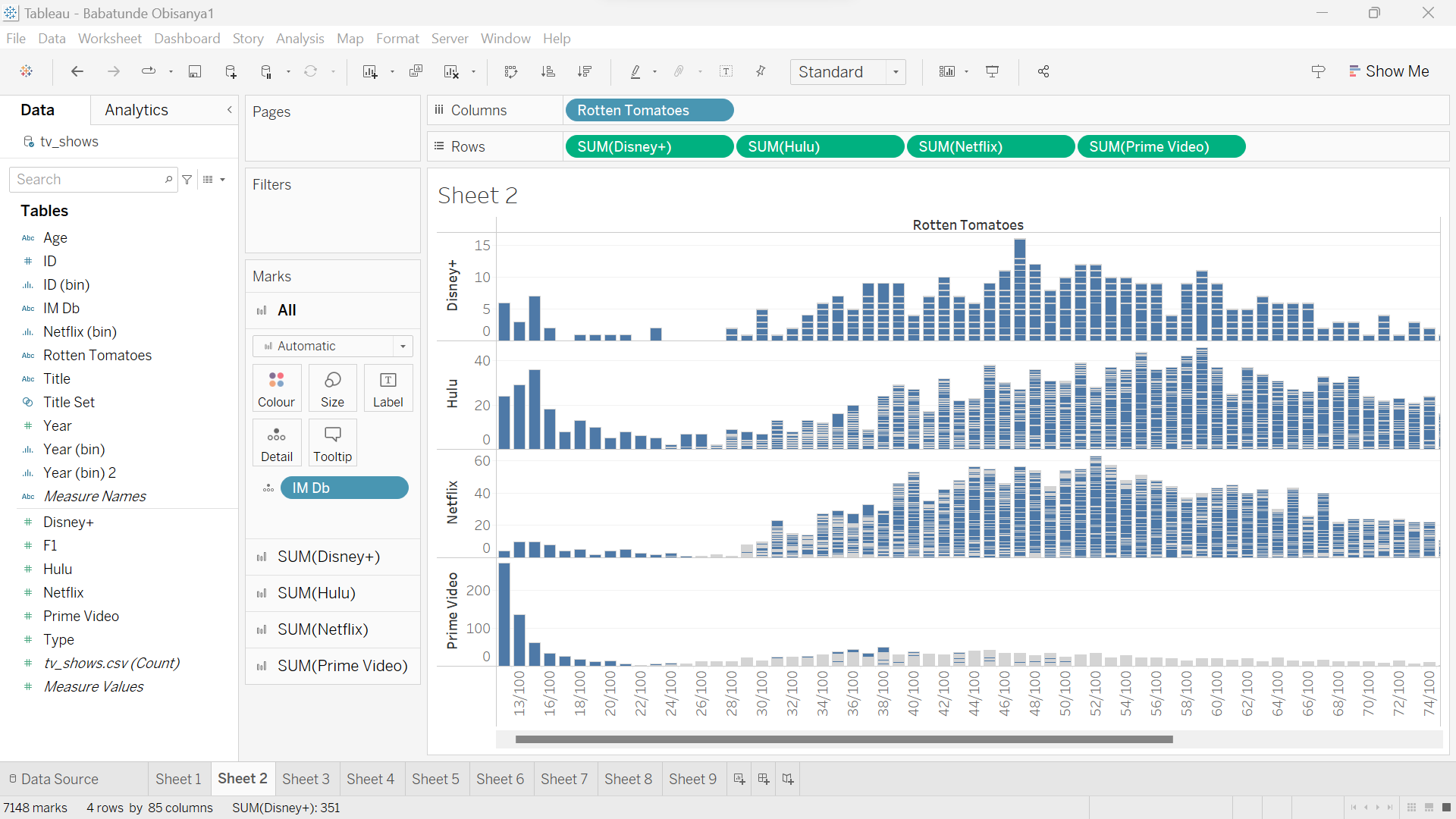


Figure 12 presents Rotten tomato rating with respect to the streaming platforms. The graph above shows the rate of occurrence of a rotten-tomato rating across all movie shows on the streaming platforms. This can help by showing how often the viewers enjoy their time using a particular platform. From the above graph, we can see that the Prime video platform has the lowest amount of rating among all four streaming platforms.