BDA PROJECT

Phase 2

Developed by
21i_1679 21i_1700 21i_1705

Ibrahim’s Section:

Order of errors:

- Py4JJavaError
- NameError
- ModuleNotFoundError
- AnalysisException
- AssertionError
- TypeError
- AttributeError
- ValueError
- AnalysisException (again)
- AssertionError (again)
- Py4JJavaError (again)
- AttributeError (again)
- Py4JJavaError (again)
- TypeError (again)
- AttributeError (again)
- Py4JJavaError (again)
- AttributeError (again)
- NameError (again)

AND: a lot of java.heap out of memory errors.

A little breakdown:

1. **Py4JJavaError**: This error occurs when there is an issue with the Java Virtual Machine (JVM) that Spark runs on. For example, if you try to use a version of Spark that is not compatible with your JVM, you may see this error. Here is an example of when you encountered this error:

pythonCopy code

```
Py4JJavaError: An error occurred while calling o255.fit. : java.lang.IllegalArgumentException:
requirement failed: ALS: Column id must be of type integer but was actually type string.
```

2. **AnalysisException**: This error occurs when there is an issue with the structure or content of your data. For example, if you try to use a column that does not exist in your DataFrame, you may see this error. Here is an example of when you encountered this error:

pythonCopy code

```
AnalysisException: "cannot resolve '`userId`' given input columns: [asin_index, id, overall,
processed_summary, unixReviewTime, verified];;\n'Project [userId#478L]\n+-
```

Relation[overall#414,unixReviewTime#415L,verified#416,processed_summary#417,id#474L,asin_index#477.0] csv\n"

3. **AttributeError**: This error occurs when you try to access an attribute or method that does not exist on an object. For example, if you try to access a method that is not defined on a DataFrame, you may see this error. Here is an example of when you encountered this error:

pythonCopy code

AttributeError: 'DataFrame' object has no attribute 'filter'

4. **NameError**: This error occurs when you try to use a variable or function that is not defined in your code. For example, if you try to use a variable that was never declared, you may see this error. Here is an example of when you encountered this error:

pythonCopy code

NameError: name 'spark' is not defined

5. **ValueError**: This error occurs when there is an issue with the value of a variable or argument in your code. For example, if you try to pass a non-numeric value to a function that expects a number, you may see this error. Here is an example of when you encountered this error:

pythonCopy code

ValueError: Attempted to convert a value of type string to a value of type double, but no valid double value could be found.``` I hope this helps!

This project was absolute hell to complete and im writing this documentation on day 3 of no sleep.

Please, provide future batches with a project on topics that they are more practiced in because Spark was just taught and this was too big of a task that I was able to complete.
The main reason for this is that I don't own a good machine and processing takes a very long time for me.

<u>A SAMPLE WAS USED IN THE LATER STAGES TO OVERCOME THE 8 HOUR LONG ALS TRAINING THAT EVENTUALLY MADE MY MACHINE OVERHEAT AND CRASH MULTIPLE TIMES.</u>

The code begins by importing the necessary modules **SparkSession** and **SparkConf** from the **pyspark.sql** and **pyspark** libraries respectively. It also sets up a **SparkConf** object that is used to configure the Spark session. The configuration includes setting the input and output URIs for MongoDB, adding the MongoDB Spark connector package, and setting the serialization buffer size. The **SparkSession** object is then created with the specified configuration parameters.

Next, the code reads data from MongoDB using the **load()** method of the **DefaultSource** class in the **com.mongodb.spark.sql** package. The data is loaded into a DataFrame object named **df**. The first 10 rows of the DataFrame are then printed using the **show()** method.

The code then imports the Natural Language Toolkit (NLTK) module, downloads the stopwords and punkt data, and defines two lists of positive and negative words. It also defines a UDF named **preprocess_summary()** that tokenizes, lowercases, and removes stop words from the summary column of the DataFrame while also filtering based on the positive and negative words lists. Another column named **processed_summary** is then added to the DataFrame by applying the UDF to the summary column.

The **verified** column of the DataFrame is then transformed to be a binary column with 1 for "true" and 0 for "false" using the **withColumn()** method.

A new column named **id** is then added to the DataFrame using the **monotonically_increasing_id()** method. This method generates a unique ID for each row in the DataFrame. The **asin** column is then indexed using the **StringIndexer** transformer from the **pyspark.ml.feature** library. The resulting indexed DataFrame is named **indexed_df**.

The original DataFrame is then modified to remove the **summary**, **_id**, and **asin** columns using the **drop()** method.

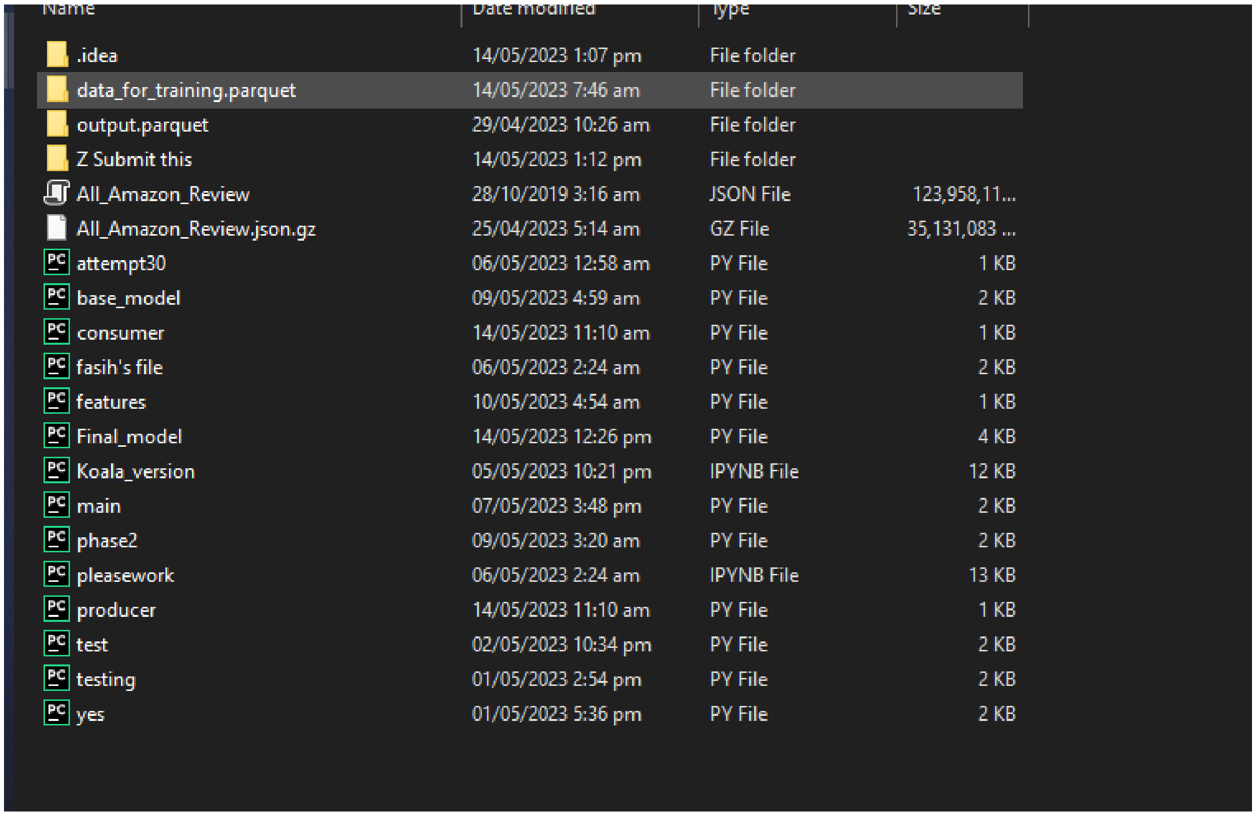
A subset of the DataFrame named **sampled_df** is then created by randomly sampling 10% of the data without replacement. The **sampled_df** DataFrame is then split into training and test sets with an 80:20 ratio.

An ALS model is then created using the **ALS()** class from the **pyspark.ml.recommendation** module. The user ID column is set to **id**, the item ID column is set to **asin_index**, the rating column is set to **overall**, and the cold start strategy is set to "drop". The model is then fitted to the training data using the **fit()** method.

The model is then used to make predictions on the test data using the **transform()** method. The **RegressionEvaluator** class from the **pyspark.ml.evaluation** module is used to evaluate the model's performance by computing the root-mean-square error (RMSE) between the predicted and actual ratings. The **evaluate()** method of the evaluator object is used to compute the RMSE.

The model is then used to generate recommendations for all users using the **recommendForAllUsers()** method. The resulting recommendations are saved to a Parquet file named **user_recommendations.parquet**.

A sample of what one of the 15 folders I made for this project looked like:



Name	Date modified	Type	Size
.idea	14/05/2023 1:07 pm	File folder	
data_for_training.parquet	14/05/2023 7:46 am	File folder	
output.parquet	29/04/2023 10:26 am	File folder	
Z Submit this	14/05/2023 1:12 pm	File folder	
All_Amazon_Review	28/10/2019 3:16 am	JSON File	123,958,11...
All_Amazon_Review.json.gz	25/04/2023 5:14 am	GZ File	35,131,083 ...
attempt30	06/05/2023 12:58 am	PY File	1 KB
base_model	09/05/2023 4:59 am	PY File	2 KB
consumer	14/05/2023 11:10 am	PY File	1 KB
fasih's file	06/05/2023 2:24 am	PY File	2 KB
features	10/05/2023 4:54 am	PY File	1 KB
Final_model	14/05/2023 12:26 pm	PY File	4 KB
Koala_version	05/05/2023 10:21 pm	IPYNB File	12 KB
main	07/05/2023 3:48 pm	PY File	2 KB
phase2	09/05/2023 3:20 am	PY File	2 KB
pleasework	06/05/2023 2:24 am	IPYNB File	13 KB
producer	14/05/2023 11:10 am	PY File	1 KB
test	02/05/2023 10:34 pm	PY File	2 KB
testing	01/05/2023 2:54 pm	PY File	2 KB
yes	01/05/2023 5:36 pm	PY File	2 KB

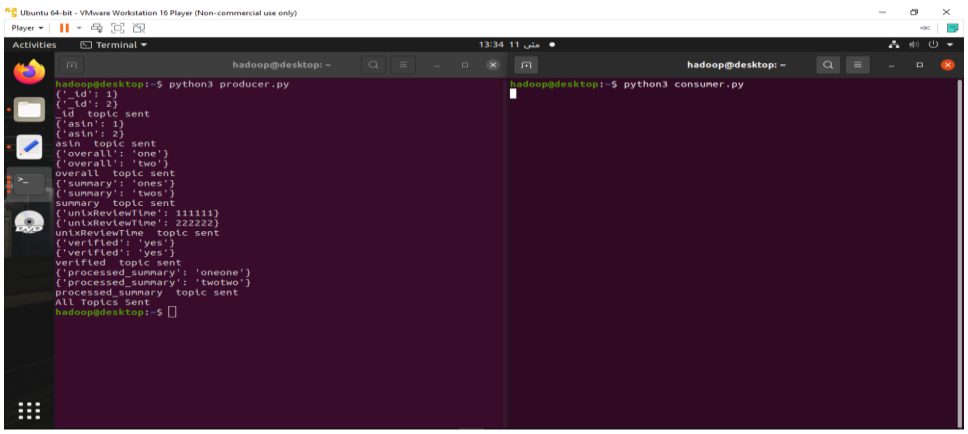
Afaq’s Section:

I was tasked with loading the database, and applying the ML model on it with proper feature engineering and cleaning.

I also had to link the flask app obtained from Fasih and the Kafka producer and consumer files obtained from Afaq.

Error:

When there was an error is the consumer file, it would not run and get stuck. Since, I was using notepad to write my code and then run it directly through terminal so at time it would not properly execute the consumer file.

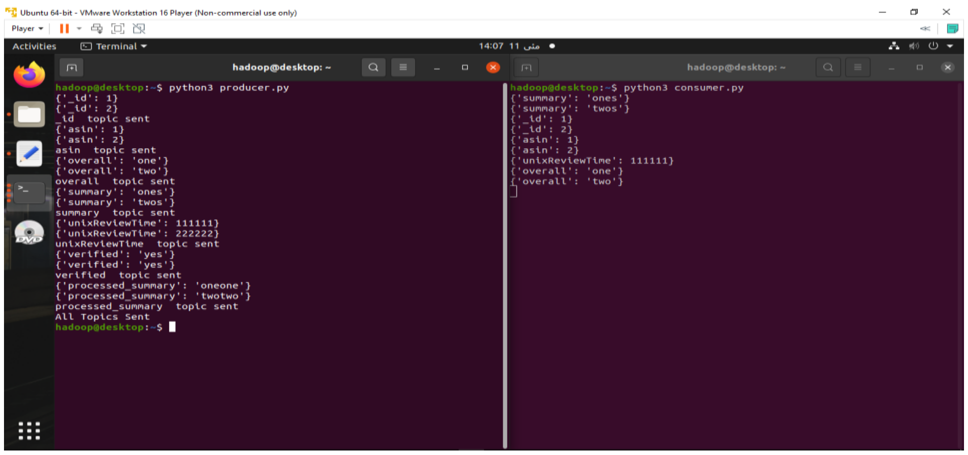

Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
13:34 11 مئی
hadoop@desktop: ~
hadoop@desktop: ~

```
hadoop@desktop:~$ python3 producer.py
{'_id': 1}
{'_id': 2}
_id  topic sent
{'asin': 1}
{'asin': 2}
asin  topic sent
{'overall': 'one'}
{'overall': 'two'}
overall  topic sent
{'summary': 'ones'}
{'summary': 'twos'}
summary  topic sent
{'unixReviewTime': 111111}
{'unixReviewTime': 222222}
unixReviewTime  topic sent
{'verified': 'yes'}
{'verified': 'yes'}
verified  topic sent
{'processed_summary': 'oneone'}
{'processed_summary': 'twotwo'}
processed_summary  topic sent
All Topics Sent
hadoop@desktop:~$
```

```
hadoop@desktop:~$ python3 consumer.py
```

Not synced:

Sometime the consumer and the producer were not in sync which caused the topics to receive them earlier than the other and resulted in termination of producer. To resolve this issue, I used "producer.flush" in the producer file, this ensured that both files are synchronized.


Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
14:07 11 مئی
hadoop@desktop: ~
hadoop@desktop: ~

```
hadoop@desktop:~$ python3 producer.py
{'_id': 1}
{'_id': 2}
_id  topic sent
{'asin': 1}
{'asin': 2}
asin  topic sent
{'overall': 'one'}
{'overall': 'two'}
overall  topic sent
{'summary': 'ones'}
{'summary': 'twos'}
summary  topic sent
{'unixReviewTime': 111111}
{'unixReviewTime': 222222}
unixReviewTime  topic sent
{'verified': 'yes'}
{'verified': 'yes'}
verified  topic sent
{'processed_summary': 'oneone'}
{'processed_summary': 'twotwo'}
processed_summary  topic sent
All Topics Sent
hadoop@desktop:~$
```

```
hadoop@desktop:~$ python3 consumer.py
{'summary': 'ones'}
{'summary': 'twos'}
{'_id': 1}
{'_id': 2}
{'asin': 1}
{'asin': 2}
{'unixReviewTime': 111111}
{'overall': 'one'}
{'overall': 'two'}
```

Contribution:

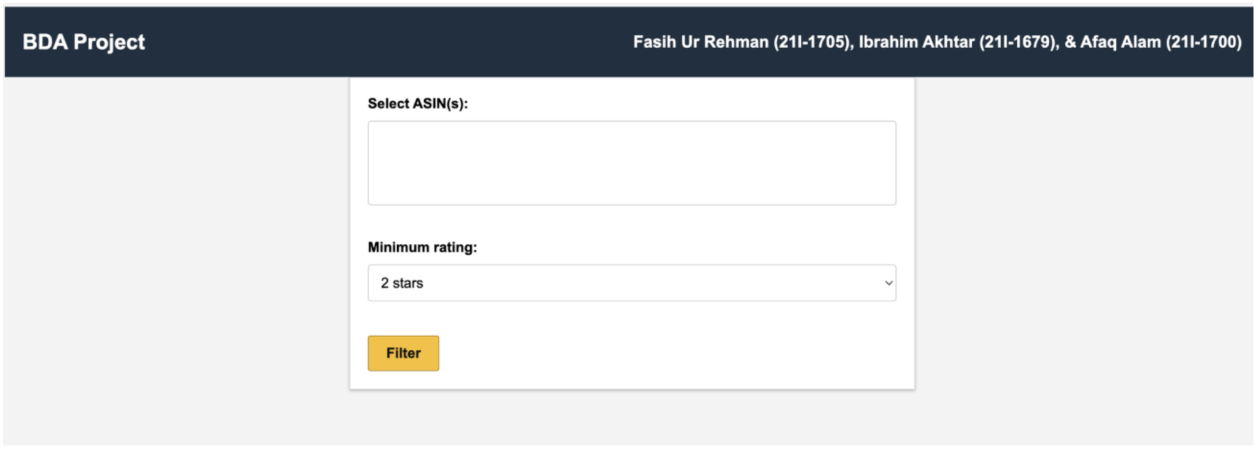
My part in this was to provide code for Kafka producer and consumer file to transfer the inputs and outputs generated for/by the ML model.

Code execution:

The producer reads from a csv file and then extracts each column, which are then sent to consumer where it consumes the topics from the columns and writes them back to another csv file.

Fasih's Section:

I was tasked with the front end side of this phase of the project, so I had to create the main flask application that would take in the necessary inputs required for the ML Model Ibrahim was implementing. At the time I was working, we were unsure as to what the ML Model will be working on, so I just created a random input of ratings and ASINs, which is why you will see a random list of ASINs in the flask application.


BDA Project
Fasih Ur Rehman (21I-1705), Ibrahim Akhtar (21I-1679), & Afaq Alam (21I-1700)
Select ASIN(s):
Minimum rating:
2 stars
Filter

The next task was to display the recommendations on our page in the form of a list, this was done by creating a list of the recommended products called *filtered_products* and sending that to the flask application. This was then all displayed on the flask application.

Errors:

Basically none, since this was the easiest task of the lot. A couple errors include:

1. **Bad Request**
2. **Internal Server error**
3. **Invalid Request error**