Project Phase II

TEAMMATE NAMES
Abdul Haque 21I-1769
Ahmad Hassan 21I-1758
Junaid Zafar 21I-2690
Dept and Section: BS-DS_M
Submittedto: SirKifayatAlizai

Project Model Training

IMPORT NECESSARY LIBRARIES

In [3]:
```
from pyspark.sql import SparkSession
from pyspark.sql import Row
from pyspark.sql.functions import *
```

STATE THE INPUT AND OUTPUT URI

In [4]:
```
input_uri = "mongodb://localhost:27017/Amazon.Reviews"
output_uri = "mongodb://localhost:27017/Amazon.Reviews"
```

MAKE A SPARK SESSION AND SET MONGODB CONNECTOR ¶

In [5]:
```
spark = SparkSession.builder \
    .appName("myProject") \
    .config("spark.mongodb.input.uri", input_uri) \
    .config("spark.mongodb.output.uri", output_uri) \
    .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector_2.12:3.0.2") \
    .config("spark.driver.memory", "10g") \
    .config("spark.executor.memory", "10g") \
    .getOrCreate()
```

Import the necessary libraries for the session and start session in spark

READING FROM MONGODB

In [27]:
```
df_mongo = spark.read.format("mongo").load()
```

NOW FOR COLLABORATIVE FILTERING

In [28]:
```
df_collab = df_mongo.select('reviewerID', 'asin', 'overall')
df_collab.createOrReplaceTempView("df_collab")
```

In [ ]:
```
df_distinct_asin = spark.sql("SELECT DISTINCT asin FROM data")
```

In [16]:
```
df_distinct_asin.coalesce(1).write.option("header",True) \
 .csv("asin_csv")
```

In [17]:
```
df_distinct_reviewer = spark.sql("SELECT DISTINCT reviewerID FROM data")
```

In [18]:
```
df_distinct_reviewer.coalesce(1).write.option("header", True) \
    .csv("reviewerID_csv")
```

In [29]:
```
df_asin = spark.read.csv("asin_csv", header=True)
df_asin.createOrReplaceTempView("df_asin")
```

In [ ]:
```
df_reviewer = spark.read.csv("reviewerID_csv", header=True)
df_reviewer.createOrReplaceTempView("df_reviewer")
```

We then read the dataset from mongodb as we have stored everything inside mongodb and load it inside a df_mongo data frame which is a pyspark dataframe. We then did some preprocessing to filter out the important data to get training data out of this whole chunk of data.

Our approach was to get the distinct product reviewerIDs and also asin and then put them inside a csv for later use now csv's were good in this case as we can process them more faster with pyspark dataframe here.

In [ ]:
```
df_joined = spark.sql("SELECT ROW_NUMBER() OVER (ORDER BY df_collab.reviewerID) - 1 AS reviewer_index, \
                    ROW_NUMBER() OVER (ORDER BY df_collab.asin) - 1 AS asin_index, \
                    CAST(df_collab.overall AS INTEGER) AS overall \
                    FROM df_collab \
                    INNER JOIN df_asin ON df_collab.asin = df_asin.asin \
                    INNER JOIN df_reviewer ON df_collab.reviewerID = df_reviewer.reviewerID")
```

In [ ]:
```
df_joined.write \
    .option("header", True) \
    .option("nullValue", "") \
    .option("quote", "") \
    .option("escape", "") \
    .option("mode", "overwrite") \
    .csv("All_joined_csv")
```

In [6]:
```
df_joined_csv = spark.read.csv("All_Joined_csv", header=True)
df_joined_csv.createOrReplaceTempView("df_joined_csv")
```

Here we made a dataframe from the distinct reviewerIDs and asin's. This dataframe now contains the indexes of each of the data. For Als to work we need the indexes of our data here and that is the key feature extraction we performed here.

FILTERING OUT THE DATA

In [7]:
```
query = """
SELECT *
FROM df_joined_csv where overall > 4
"""

filtered_df = spark.sql(query)
```

Get the product IDs with the most reviews

In [8]:
```
popular_products = filtered_df.groupBy("asin_index").count().orderBy(col("count").desc()).limit(300).select("asin_i
```

In [ ]:
```
popular_products.count()
```

Filter for reviews of popular products

In [7]:
```
filtered_df = filtered_df.join(popular_products, on="asin_index")
```

We then filtered the data we needed. In this case we filtered out those products which have overall more than 4 and their asin count is more than 300. Here this is the data we need to train our model. If we choose to have more data then our systems will crash and this model won't be trained.

ALS MODEL PREDICTION

In [7]:
```
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql.functions import col
from pyspark.ml.recommendation import ALSModel
import pyspark.sql.functions as F
```

In [9]:
```
filtered_df = filtered_df.select(col("reviewer_index").cast("int"), col("asin_index").cast("int"), col("overall").ca
filtered_df.describe
```

Out[9]:
```
<bound method DataFrame.describe of DataFrame[reviewer_index: int, asin_index: int, overall: int]>
```

In [ ]:
```
filtered_df.count()
```

In [10]:
```
(training, test) = filtered_df.randomSplit([.7, .3])
```

In [11]:
```
training.count()
```

Out[11]:
```
104056989
```

SAVE TRAINING DATASET TO USE AFTER MODEL TRAIN

In [13]:
```
training.coalesce(1).write.option("header", True).csv("training_csv")
```

SAVE TEST DATASET TO USE AFTER MODEL TRAIN AS WELL

In [14]:
```
test.coalesce(1).write.option("header", True).csv("test_csv")
```

As you can see here we have a lot of rows inside the dataset and training is going to take a lot of time for this type of training. we do have to type cast into integer to train our model, so we did that.

SAVE TRAINING DATASET TO USE AFTER MODEL TRAIN

```
In [13]: training.coalesce(1).write.option("header", True).csv("training_csv")
```

SAVE TEST DATASET TO USE AFTER MODEL TRAIN AS WELL

```
In [14]: test.coalesce(1).write.option("header", True).csv("test_csv")
```

READ THE TRAINING AND TESTING DATASET

```
In [15]: training_df = spark.read.csv("training_csv", header=True)
         training_df = training_df.select(col("reviewer_index").cast("int"), col("asin_index").cast("int"), col("overall").ca
         training_df.count()
Out[15]: 104056989

In [16]: test_df = spark.read.csv("test_csv", header=True)
         test_df = test_df.select(col("reviewer_index").cast("int"), col("asin_index").cast("int"), col("overall").cast("int
         test_df.count()
Out[16]: 44592961
```

TRAINING MODEL

```
In [17]: als = ALS(maxIter=2, rank=6, userCol='reviewer_index', itemCol='asin_index', ratingCol='overall', coldStartStrategy=

In [ ]: model = als.fit(training_df)SS
```

Save the training dataset and test for later use and then train the model here. I used some hyper parameters such as maxIter 2 and rank to be 6. I chose these because if the rank is increased then our model won't work as we want it to.

SAVE THE MODEL

```
In [14]: model.save("ALS")
```

LOAD THE SAVED MODEL

```
In [9]: model = ALSModel.load("ALS")
```

GET RECOMMENDATIONS FROM MODEL

```
In [43]: user_recs=model.recommendForAllUsers(20).show(10)
```

```
+--------------+--------------------+
|reviewer_index|     recommendations|
+--------------+--------------------+
|       3221405|[{100006060, 4.96...|
|       3627754|[{100015935, 4.96...|
|       8511114|[{100011693, 4.96...|
|       9737584|[{100013635, 4.96...|
|      10246362|[{100016101, 4.96...|
|      11582635|[{100002437, 4.96...|
|      11930560|[{100011262, 4.96...|
|      12191461|[{100013570, 4.96...|
|      12296296|[{100016143, 4.96...|
|      12614929|[{100021718, 3.96...|
+--------------+--------------------+
only showing top 10 rows
```

Get the recommendations after saving the model.

EVALUATING MODEL

In [16]:
```
evaluator=RegressionEvaluator(metricName="rmse",labelCol="overall",predictionCol="prediction")
predictions=model.transform(test)
rmse=evaluator.evaluate(predictions)
print("RMSE="+str(rmse))
predictions.show()
```

```
RMSE=0.0021583798928685068
+--------------+----------+-------+----------+
|reviewer_index|asin_index|overall|prediction|
+--------------+----------+-------+----------+
|      10760282|   1000127|      4| 3.9974043|
|      83476127|   1000146|      5|  4.997922|
|     218680506|    100010|      5| 4.9979224|
|     219591444|   1000073|      5|  4.997922|
|       1614443|  10000600|      5| 4.9979224|
|      30351220|  10000021|      5| 4.9979224|
|      52928479|  10000528|      5|  4.997922|
|      75138366|  10000989|      4| 3.9974043|
|      84206832|  10001183|      5| 4.9979224|
|      92575672|  10001380|      5|  4.997922|
|      98123032|  10001599|      5|  4.997922|
|     100427680|  10001677|      5|  4.997922|
|     108372703|  10001910|      5|  4.997922|
|     123574269|  10001272|      5|  4.997922|
|     126250063|  10001331|      4| 3.9974043|
|     184678585|  10002084|      5|  4.997922|
|     185276685|  10002100|      5| 4.9979215|
|     192470325|  10001751|      5|  4.997922|
|     215425950|  10000539|      5|  4.997922|
|     224482398|  10000761|      5|  4.997922|
+--------------+----------+-------+----------+
only showing top 20 rows
```

We evaluated our model by using

RMSE

We trained more than one model but got stuck on this one and then we trained another model with some lesser data to start working on flask . We tested the model first then started.

Testing Model

In [ ]:
```
df = spark.read.format("mongo").load()
df = df.select("asin", "reviewerID", "overall")
```

In [7]:
```
df = spark.read.csv("output.csv", header=True)
df = df.select("asin", "reviewerID", "overall")
```

SAVE INDEXER MODEL

In [20]:
```
from pyspark.ml.feature import StringIndexer
asin_indexer_model = StringIndexer(inputCol="asin", outputCol="asin_indexed")
asin_indexer_model = asin_indexer_model.fit(df)
asin_labels = asin_indexer_model.labels
asin_indexer_model.save("asin_indexer_model")
```

LOAD THE INDEXER MODEL

In [8]:
```
from pyspark.ml.feature import StringIndexerModel

# Load the ASIN Indexer model
asin_indexer_model = StringIndexerModel.load("asin_indexer_model")
```

We used another model which we trained on small dataset to get predictions as on larger datasets we weren't able to do much. It was taking a lot of time and we ended up with a lot of errors and wasted time.

GET USER RECOMMENDATIONS

In [10]:
```
user_id = "A1DS7W9IL8I2QX"
```

In [11]:
```
from pyspark.ml.recommendation import ALSModel
from pyspark.ml.feature import StringIndexer
indexed_user_id = StringIndexer(inputCol="reviewerID", outputCol="reviewer_id_indexed").fit(df).transform(df.filter
if indexed_user_id:
    indexed_user_id = indexed_user_id["reviewer_id_indexed"]
    all_user_recs = model.recommendForAllUsers(10)
    user_recs = all_user_recs.filter(all_user_recs.reviewer_id_indexed == indexed_user_id).first()
    if user_recs:
        user_recs = user_recs["recommendations"]
        print(user_recs)
    else:
        print("No recommendations found for the user.")
else:
    print("User ID not found in the dataset.")
```

```
23/05/14 05:11:19 WARN DAGScheduler: Broadcasting large task binary with size 4.1 MiB
```

```
[Row(asin_indexed=348, rating=4.942039966583252), Row(asin_indexed=1382, rating=3.9631617069244385), Row(asin_inde
xed=2382, rating=3.921555280685425), Row(asin_indexed=3268, rating=3.8937652111053467), Row(asin_indexed=1412, rat
ing=3.853649854660034), Row(asin_indexed=1268, rating=3.7664222717285156), Row(asin_indexed=1192, rating=3.6697537
899017334), Row(asin_indexed=2115, rating=3.62833833694458), Row(asin_indexed=3260, rating=3.599196672439575), Row
(asin_indexed=2387, rating=3.555853843688965)]
```

We then got recommendations by any user id we used here and this is the final output now we didn't want to get in this format so we changed this up a bit here.

In [12]:
```
from pyspark.ml.feature import StringIndexer
from pyspark.sql.functions import col
user_id = "A1DS7W9IL8I2QX"
```

In [13]:
```
from pyspark.sql.functions import col
from pyspark.ml.feature import IndexToString
indexed_user_id = StringIndexer(inputCol="reviewerID", outputCol="reviewer_id_indexed").fit(df).transform(df.filter
if indexed_user_id:
    indexed_user_id = indexed_user_id["reviewer_id_indexed"]
    all_user_recs = model.recommendForAllUsers(10)
    user_recs = all_user_recs.filter(all_user_recs.reviewer_id_indexed == indexed_user_id).first()
    if user_recs:
        recommendations = user_recs["recommendations"]
        recommendations_df = spark.createDataFrame(recommendations)
        recommendations_df = recommendations_df.select("asin_indexed", "rating")
        asin_converter = IndexToString(inputCol="asin_indexed", outputCol="asin", labels=asin_indexer_model.labels)
        recommendations_df = asin_converter.transform(recommendations_df)

        recommendations_df.show(truncate=False)
    else:
        print("No recommendations found for the user.")
else:
    print("User ID not found in the dataset.")
```

```
23/05/14 05:11:30 WARN DAGScheduler: Broadcasting large task binary with size 4.1 MiB
```

```
+------------+------------------+----------+
|asin_indexed|rating            |asin      |
+------------+------------------+----------+
|348         |4.942039966583252 |B0007OC0A2|
|1382        |3.9631617069244385|B000050FDP|
|2382        |3.921555280685425 |B000FELPIC|
|3268        |3.8937652111053467|B01HJCS4HC|
|1412        |3.853649854660034 |B005QFJ6EU|
|1268        |3.7664222717285156|B005MLDPVI|
|1192        |3.6697537899017334|B01AS5F902|
|2115        |3.62833833694458  |B00UL4U0HI|
|3260        |3.599196672439575 |B01HEJL0I0|
|2387        |3.555853843688965 |B000GX1N52|
+------------+------------------+----------+
```

The result here is a data frame which gives us the recommended asin values.

Flask (model)

```python
from flask import Flask, render_template, request
from pyspark.ml.recommendation import ALSModel
from pyspark.ml.feature import StringIndexer
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexerModel
from pyspark.ml.feature import IndexToString
from pymongo import MongoClient

app = Flask(__name__)
spark = SparkSession.builder \
    .appName("Recommendation System") \
    .config("spark.mongodb.input.uri", "mongodb://127.0.0.1/data.reviews") \
    .config("spark.mongodb.output.uri", "mongodb://127.0.0.1/data.reviews") \
    .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector_2.12:3.0.1") \
    .getOrCreate()

sc = spark.sparkContext
df = spark.read.csv("output.csv", header=True)
df = df.select("asin", "reviewerID", "overall")
model = ALSModel.load("als_model.joblib")
asin_indexer_model = StringIndexerModel.load("asin_indexer_model")
mongo_client = MongoClient('mongodb://localhost:27017')
db = mongo_client['recommendations']
collection = db['recommendations']


@app.route('/')
def index():
    return render_template('index.html')


@app.route('/recommendations', methods=['POST'])
def get_recommendations():
    user_id = request.form['user_id']
    indexed_user_id = StringIndexer(inputCol="reviewerID", outputCol="reviewer_id_indexed").fit(df).transform(df.filter(df.reviewerID == user_id)).first()
    if indexed_user_id:
        indexed_user_id = indexed_user_id["reviewer_id_indexed"]
        all_user_recs = model.recommendForAllUsers(10)
        user_recs = all_user_recs.filter(all_user_recs.reviewer_id_indexed == indexed_user_id).first()
        if user_recs:
            recommendations = user_recs["recommendations"]
            recommendations_df = spark.createDataFrame(recommendations)
            recommendations_df = recommendations_df.select("asin_indexed", "rating")
            asin_converter = IndexToString(inputCol="asin_indexed", outputCol="asin", labels=asin_indexer_model.labels)
            recommendations_df = asin_converter.transform(recommendations_df)
            recommendations_list = recommendations_df.select("asin", "rating").collect()
            collection.insert_one({"user_id": user_id, "recommendations": recommendations_list})
            return render_template('recommend.html', user_id=user_id, recommendations=recommendations_list)
        else:
            message = "No recommendations found for the user."
    else:
        message = "User ID not found in the dataset."
    return render_template('error.html', message=message)
```

This is the whole flask part and we integrated it with mongodb and get those values and store back the result for the corresponding values that we get in recommendations.

Flask (Kafka)

```python
from flask import Flask, render_template, request
from pyspark.ml.recommendation import ALSModel
from pyspark.ml.feature import StringIndexer
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, explode
from pyspark.ml.feature import StringIndexerModel
from kafka import KafkaProducer
from pymongo import MongoClient
import json
from pyspark.ml.feature import IndexToString


app = Flask(__name__)
spark = SparkSession.builder \
    .appName("Recommendation System") \
    .config("spark.mongodb.input.uri", "mongodb://127.0.0.1/data.reviews") \
    .config("spark.mongodb.output.uri", "mongodb://127.0.0.1/data.reviews") \
    .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector_2.12:3.0.1") \
    .getOrCreate()


sc = spark.sparkContext
df = spark.read.csv("output.csv", header=True)
df = df.select("asin", "reviewerID", "overall")
model = ALSModel.load("als_model.joblib")
asin_indexer_model = StringIndexerModel.load("asin_indexer_model")

# Kafka producer configuration
kafka_bootstrap_servers = 'localhost:9092'
kafka_topic = 'recommendations1'

# Create Kafka producer
producer = KafkaProducer(
    bootstrap_servers=kafka_bootstrap_servers,
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)
mongo_uri = 'mongodb://localhost:27017'
mongo_db = 'mydatabase'
mongo_collection = 'mycollection'

# Connect to MongoDB
mongo_client = MongoClient(mongo_uri)
db = mongo_client[mongo_db]
collection = db[mongo_collection]

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/recommendations', methods=['POST'])
def get_recommendations():
    user_id = request.form['user_id']
```

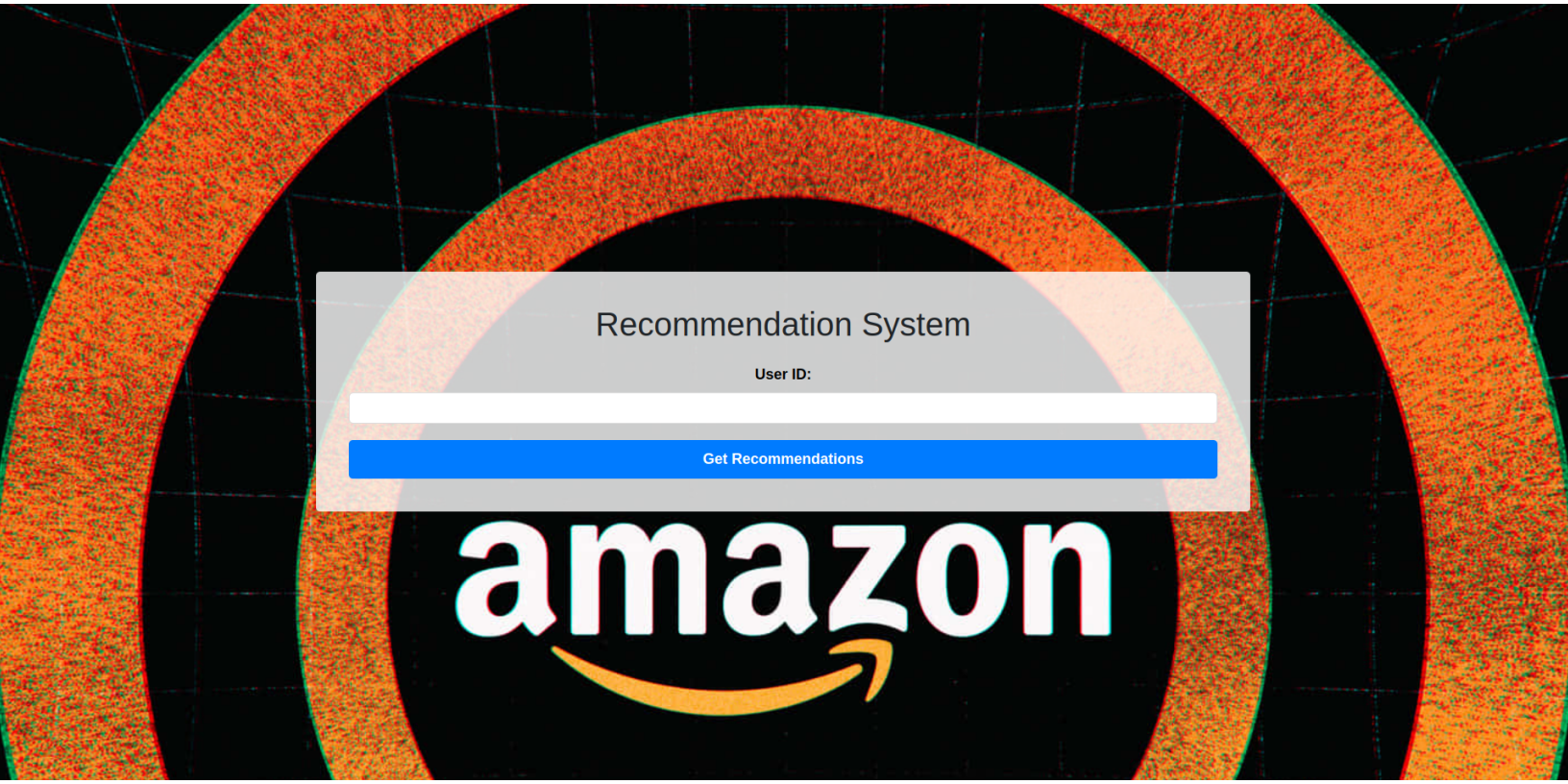
This is the flask kafka part. We made separate files for everything so this is a separate main.py file for the kafka part.

The remaining code is below:

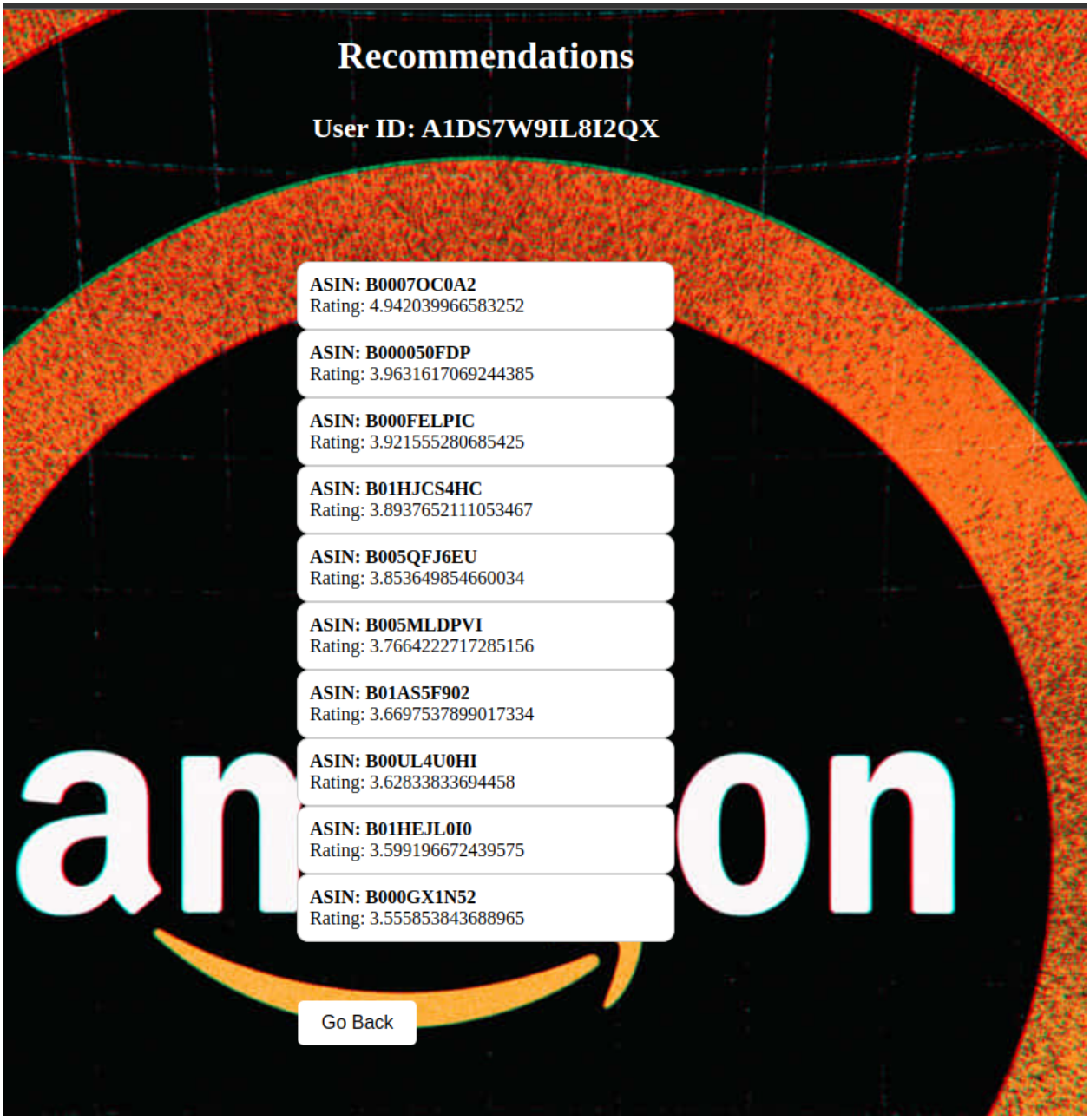
```
    indexed_user_id = StringIndexer(inputCol="reviewerID", outputCol="reviewer_id_indexed").fit(df).transform(df.filter(df.reviewerID == user_id)).first()
    if indexed_user_id:
        indexed_user_id = indexed_user_id["reviewer_id_indexed"]
        all_user_recs = model.recommendForAllUsers(10)
        user_recs = all_user_recs.filter(all_user_recs.reviewer_id_indexed == indexed_user_id).first()
        if user_recs:
            recommendations = user_recs["recommendations"]
            recommendations_df = spark.createDataFrame(recommendations)
            recommendations_df = recommendations_df.select("asin_indexed", "rating")
            asin_converter = IndexToString(inputCol="asin_indexed", outputCol="asin", labels=asin_indexer_model.labels)
            recommendations_df = asin_converter.transform(recommendations_df)
            recommendations_list = recommendations_df.select("asin", "rating").collect()
            for recommendation in recommendations_list:
                producer.send(kafka_topic, recommendation.asDict())
            #return render_template('loading.html')  # Display a loading screen while recommendations are generated
            recommendations = []
            for document in collection.find():
                recommendations.append((document['asin'], document['rating']))
            return render_template('recommendations.html',recommendations=recommendations)
        else:
            message = "No recommendations found for the user."
    else:
        message = "User ID not found in the dataset."
    return render_template('error.html', message=message)

if __name__ == '__main__':
    app.run(debug=True)
```

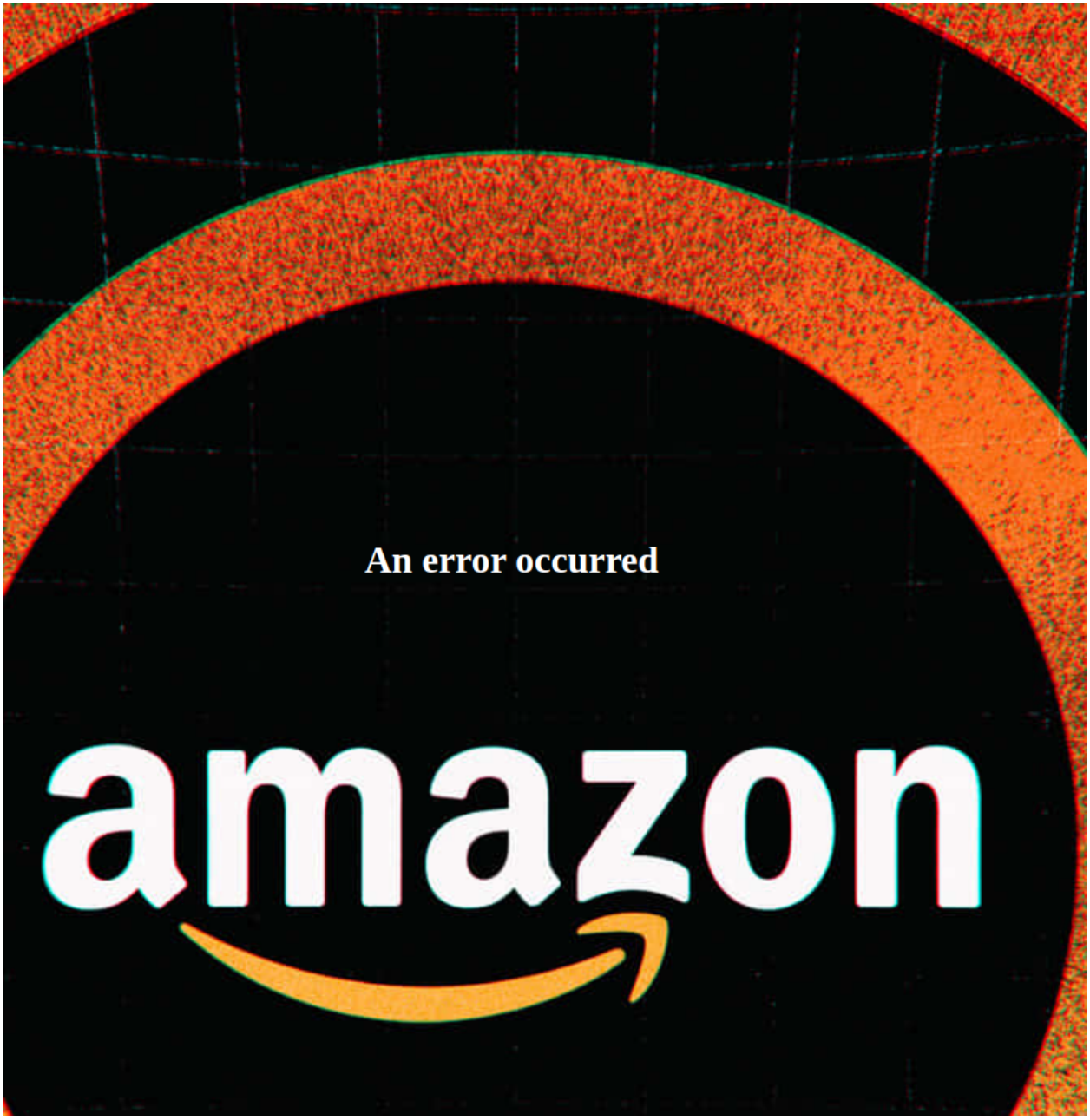
Flask Websites



Home page where you have to enter a User ID.



With the right user ID you will get the results here if not then you'll be shown the error page.

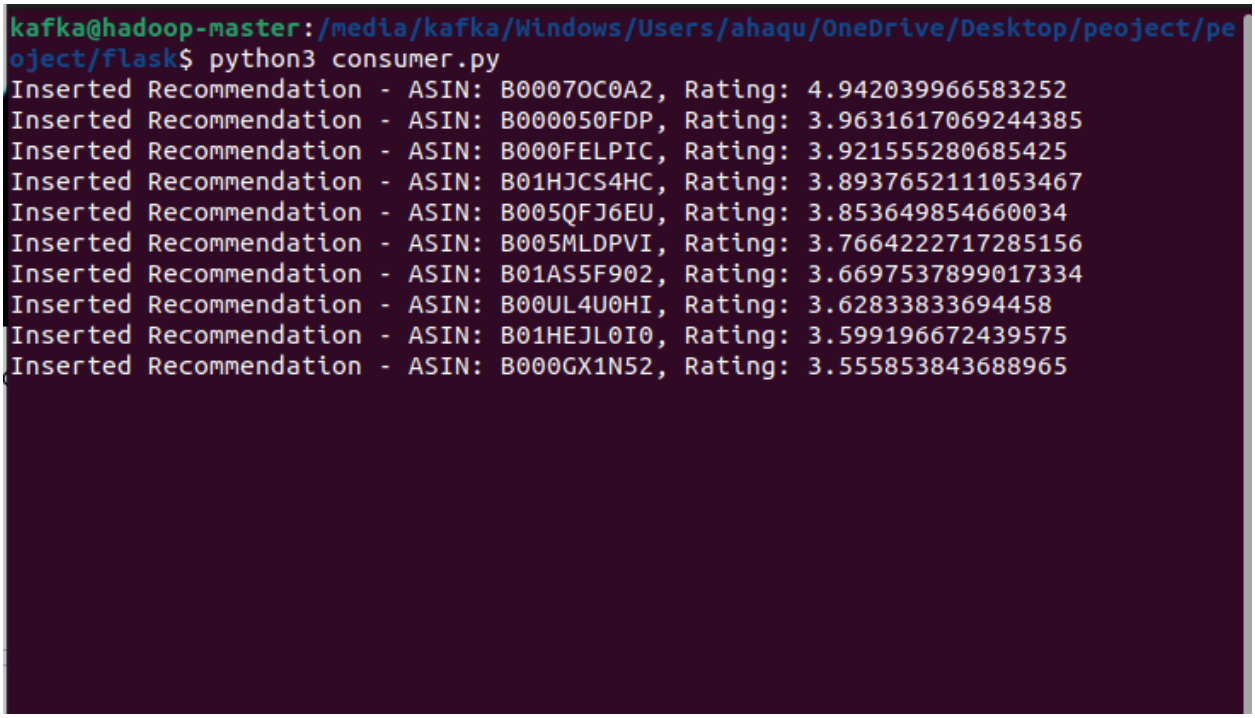


The error page is displayed whenever the user enters the wrong user ID.

Now the kafka website is the same but its outputs are shown in the terminal.

Flask (kafka) output

The recommendations show on the page and here is the consumer.py output.



The consumer is working all fine here.

Conclusion

This overall project was a bit tricky in its own way but we got this through. Our project wasn't done as expected but it was a great experience and we learnt a lot. As we were using a mega dataset then that was a hassle but we stood through and loaded all of the data inside mongodb after that we did analysis on some of the data which we needed.

All in all, we had fun.