

BIA-678 Big Data Technologies

Building Data Pipeline to Analyze User Behaviour Metrics

Movie Rental Company

Team 12-A

Harshita Mishra | Paras Pandhare | Siddharth Singh

Table of Contents

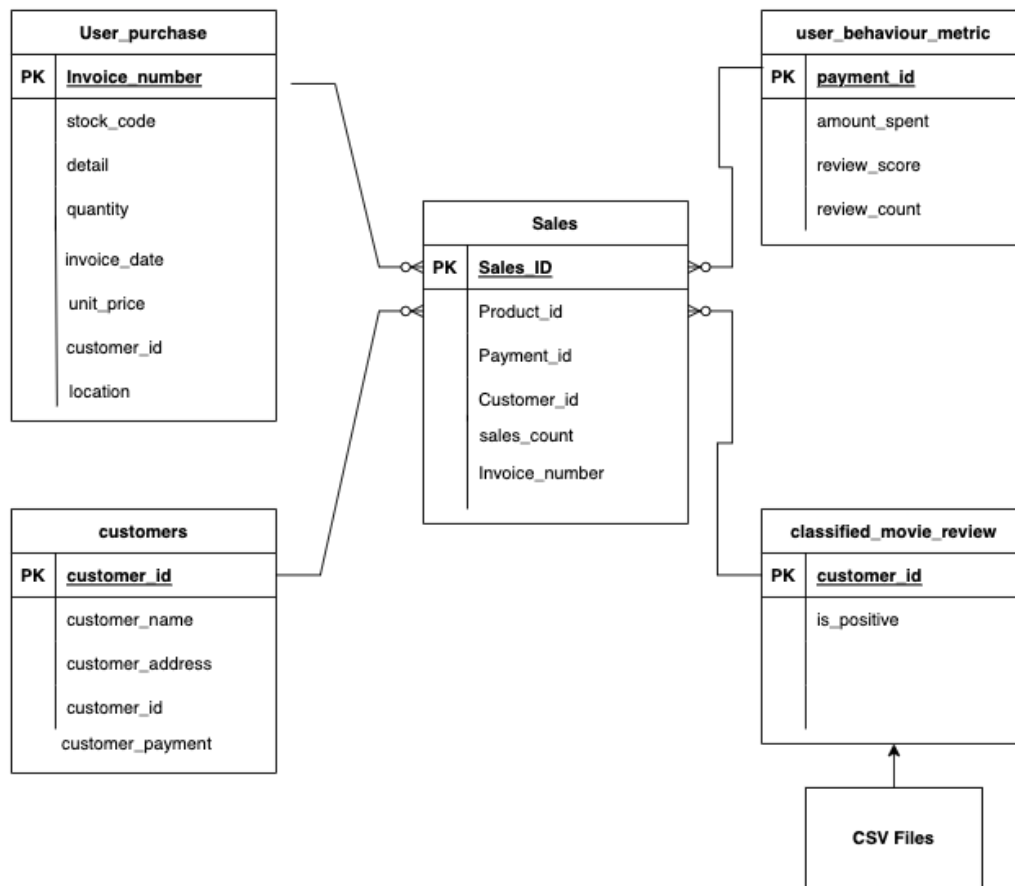
| | |
|--|----|
| Introduction & Objective - - - - - | 3 |
| Design & Architecture - - - - - | 4 |
| Infrastructure Setup | |
| AWS Components - - - - - | 6 |
| PostgreSQL - - - - - | 7 |
| Code Walkthrough - - - - - | 8 |
| Qualitative Performance Computational Analysis - - - - - | 13 |
| Future Scope - - - - - | 14 |
| Conclusion - - - - - | 15 |
| Team Work - - - - - | 16 |
| References - - - - - | 17 |

Introduction & Objective

Our focus is on building an ETL pipeline to data warehouse to analyse user behaviour metrics. Understanding how consumers interact with your product requires an understanding of user behaviour. How much time do they spend with it, specifically? What do they most frequently click on? When do users opt to take a break from their journey? Analysing your consumers' activity can help you answer these questions and improve your product over time. Through big data engineering, we will analyse user behaviour on getting movie rentals.

We are tasked with building a data pipeline to populate the `user_behavior_metric` table. The `user_behavior_metric` table is an OLAP table, meant to be used by analysts, dashboard software, etc. It is built from

1. `user_purchase`: OLTP table with user purchase information.
2. `movie_review`: Data sent every day by an external data vendor.



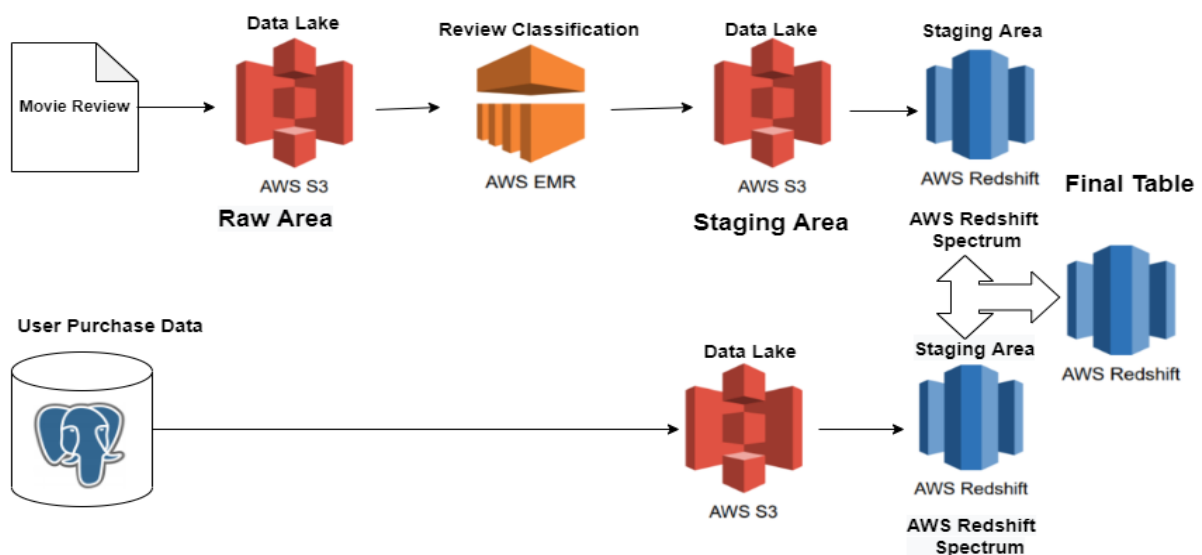
Dimensional Model for OLAP - AWS Redshift

After analysing user metrics, we can understand if user purchases a certain movie rental, there could be high chances the user would rent a similar kind or genre movie next time. We will be able to interpret how an average similarity between two movies a user consecutively watched is high if this user has low activity.

Design & Architecture

We will be using Airflow to orchestrate the following tasks:

- a) Classifying movie reviews with Apache Spark.
- b) Loading the classified movie reviews into the data warehouse.
- c) Extracting user purchase data from an OLTP database and loading it into the data warehouse.
- d) Joining the classified movie review data and user purchase data to get user_behaviour_metric.



Component stack architecture

Infrastructure Setup

Two local components used:

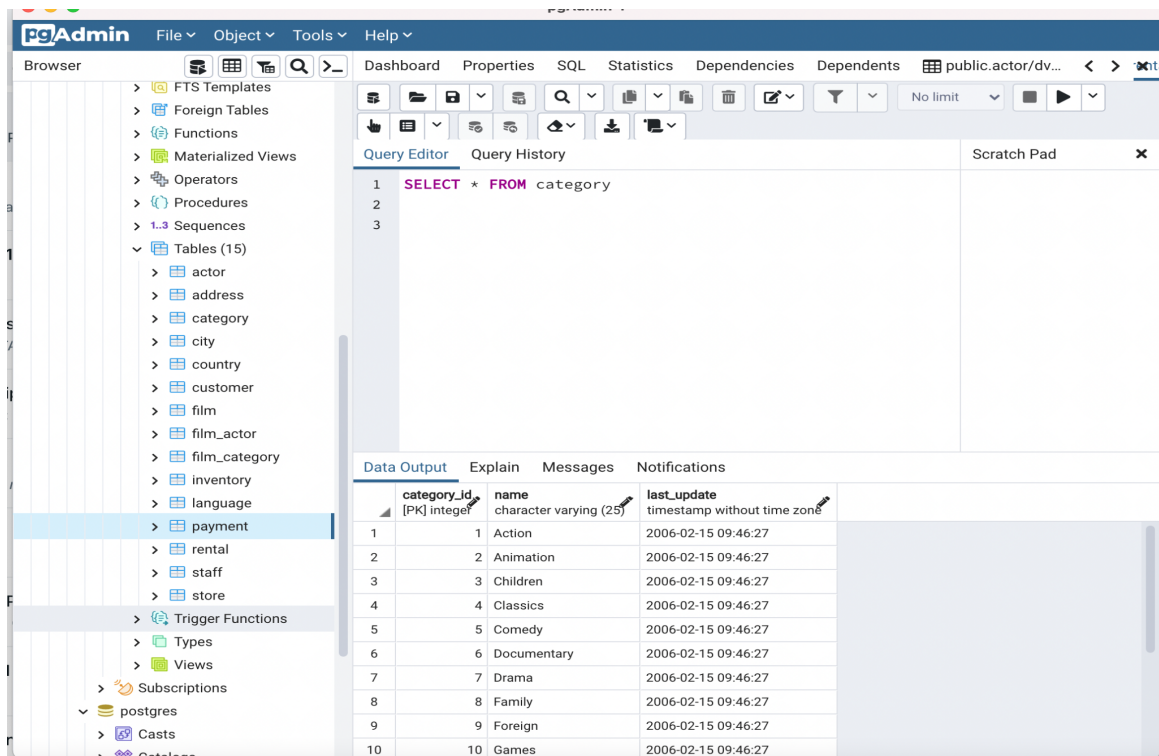
1. **Apache Airflow**: The Apache Airflow manages the orchestration of data pipelines implements the sequencing, coordination, scheduling and managing complex data and its pipelines from the server
2. **Docker**: Apache Airflow's metadata database in docker container. It is used for storing the scheduling details, maintain and orchestrate the containers for dependencies control. For this project, we have used PostgreSQL, since there are multiple containers it will be easy to use Docker Compose in order to deploy all the containers at once

AWS Components used:

1. **AWS S3**: Our Data Lake
2. **AWS Redshift**: Spectrum as our Data Warehouse
3. **AWS IAM**: allows the spectrum to access the data in S3 at given query time
4. **AWS EMR**: is used to run the Apache Spark for text classification

PostgreSQL | Our scope on database tables contains:

- a. Category
- b. Customer
- c. Film
- d. Film_Category
- e. Inventory
- f. Payment



The screenshot shows the pgAdmin interface. On the left, the 'Browser' pane displays a tree of database objects, with 'payment' highlighted under 'Tables (15)'. The main pane shows a 'Query Editor' with the SQL query: `SELECT * FROM category`. Below the query editor, the 'Data Output' tab is active, displaying a table with 10 rows and 3 columns: `category_id` (integer), `name` (character varying), and `last_update` (timestamp without time zone). The data is as follows:

| category_id | name | last_update |
|-------------|-------------|---------------------|
| 1 | Action | 2006-02-15 09:46:27 |
| 2 | Animation | 2006-02-15 09:46:27 |
| 3 | Children | 2006-02-15 09:46:27 |
| 4 | Classics | 2006-02-15 09:46:27 |
| 5 | Comedy | 2006-02-15 09:46:27 |
| 6 | Documentary | 2006-02-15 09:46:27 |
| 7 | Drama | 2006-02-15 09:46:27 |
| 8 | Family | 2006-02-15 09:46:27 |
| 9 | Foreign | 2006-02-15 09:46:27 |
| 10 | Games | 2006-02-15 09:46:27 |

Snapshot of database tables in PostgreSQL

Code Walkthrough

The data for `user_behavior_metric` is generated from 4 main tables: customer, rental, payment and film_category. We will look at how each of them is ingested, transformed, and used to get the data for the final table.

1. `extract_user_purchase` and `user_purchase_to_stage_data_lake`:

The data is synced between our local Postgres and Airflow containers

```
extract_user_purchase_data = PostgresOperator(
    dag=dag,
    task_id="extract_user_purchase_data",
    sql="./scripts/sql/unload_user_purchase.sql",
    postgres_conn_id="postgres_default",
    params={"user_purchase": "/temp/user_purchase.csv"},
    depends_on_past=True,
    wait_for_downstream=True,
)

user_purchase_to_stage_data_lake = PythonOperator(
    dag=dag,
    task_id="user_purchase_to_stage_data_lake",
    python_callable=_local_to_s3,
    op_kwargs={
        "file_name": "/temp/user_purchase.csv",
        "key": "stage/user_purchase/{{ ds }}/user_purchase.csv",
        "bucket_name": BUCKET_MOVIE_REVIEW,
        "remove_local": "true",
    },
)

user_purchase_stage_data_lake_to_stage_tbl = PythonOperator(
    dag=dag,
    task_id="user_purchase_stage_data_lake_to_stage_tbl",
    python_callable=run_redshift_external_query,
    op_kwargs={
        "qry": "alter table spectrum.user_purchase_staging add if not"
```

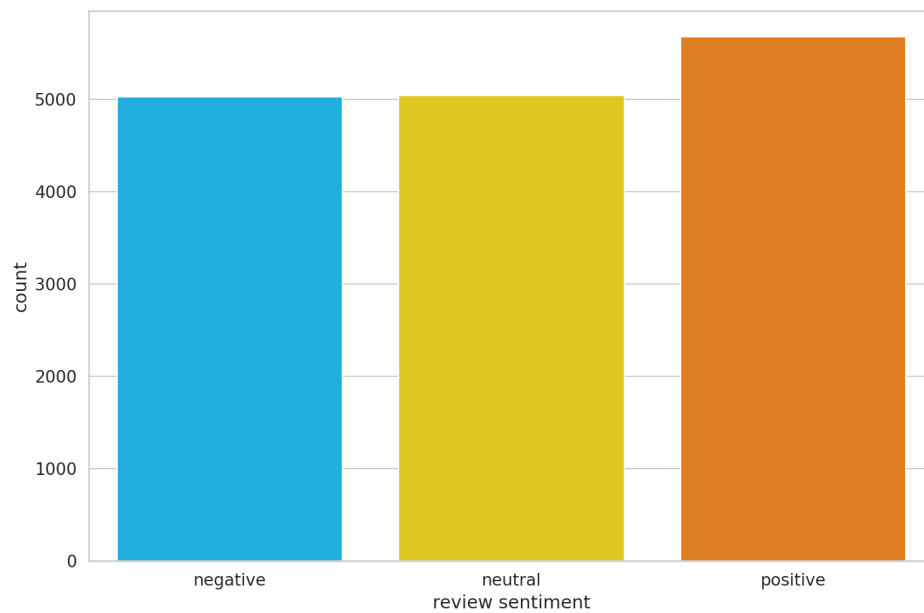


```
exists partition(insert_date='{ ds }') \
    location 's3://'
    + BUCKET_NAME
    + "/stage/user_purchase/{ ds }'",
},
)
```

2. `review_classification_pyspark_script`: We read input data, tokenize text, remove stop words, split the dataset into Positive, Negative and Neutral sentiment from the review scores in customer table. We use this function to use spark to write functions to HDFS output. This allows AWS EMR to reference it

```
def to_sentiment(rating):
    rating = int(rating)
    if rating <= 2:
        return 0
    elif rating == 3:
        return 1
    else:
        return 2

df['sentiment'] = df.score.apply(to_sentiment)
class_names = ['negative', 'neutral', 'positive']
```



Text Classification using Spark function

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--input", type=str, help="HDFS input",
                        default="/movie")
    parser.add_argument("--output", type=str, help="HDFS output",
                        default="/output")
    parser.add_argument("--run-id", type=str, help="run id")
    args = parser.parse_args()
    spark = SparkSession.builder.appName("Random Text
    Classifier").getOrCreate()
    random_text_classifier(
        input_loc=args.input, output_loc=args.output, run_id=args.run_id
    )
```

```
print(classification_report(y_test, y_pred, target_names=class_names))
```

| | precision | recall | f1-score | support |
|----------|-----------|--------|----------|---------|
| negative | 0.89 | 0.87 | 0.88 | 245 |
| neutral | 0.83 | 0.85 | 0.84 | 254 |
| positive | 0.92 | 0.93 | 0.92 | 289 |

| | | | | |
|--------------|------|------|------|-----|
| accuracy | | | 0.88 | 788 |
| macro avg | 0.88 | 0.88 | 0.88 | 788 |
| weighted avg | 0.88 | 0.88 | 0.88 | 788 |

3. `start_emr_movie_classification_script`: Adds the EMR steps defined at `dags/scripts/emr/clean_movie_review.json` to our EMR cluster.

This task adds 3 EMR steps to the cluster, they do the following

- a. Moves raw data from S3 to HDFS: Copies data from data lake's raw area into EMR's HDFS.
- b. Classifies movie reviews: Runs the review classification pyspark script.
- c. Moves classified data from HDFS to S3: Copies data from EMR's HDFS to data lake's staging area.

```
movie_review_to_raw_data_lake = PythonOperator(
    dag=dag,
    task_id="movie_review_to_raw_data_lake",
    python_callable=_local_to_s3,
    op_kwargs={
        "file_name": "/data/movie_review.csv",
        "key": "raw/movie_review/{{ ds }}/movie.csv",
        "bucket_name": BUCKET_MOVIE_REVIEW,
    },
)

spark_script_to_s3 = PythonOperator(
    dag=dag,
    task_id="spark_script_to_s3",
    python_callable=_local_to_s3,
    op_kwargs={
```

```

        "file_name":
        "./dags/scripts/spark/random_text_classification.py",
        "key": "scripts/random_text_classification.py",
        "bucket_name": BUCKET_MOVIE_REVIEW,
    },
)

start_emr_movie_classification_script = EmrAddStepsOperator(
    dag=dag,
    task_id="start_emr_movie_classification_script",
    job_flow_id=EMR_ID,
    aws_conn_id="aws_default",
    steps=EMR_STEPS,
    params={
        "BUCKET_NAME": BUCKET_MOVIE_REVIEW,
        "raw_movie_review": "raw/movie_review",
        "text_classifier_script": "scripts/random_text_classifier.py",
        "stage_movie_review": "stage/movie_review",
    },
    depends_on_past=True,
)

last_step = len(EMR_STEPS) - 1

wait_for_movie_classification_transformation = EmrStepSensor(
    dag=dag,
    task_id="wait_for_movie_classification_transformation",
    job_flow_id=EMR_ID,
    step_id='{{
task_instance.xcom_pull("start_emr_movie_classification_script",
key="return_value")['
    + str(last_step)
    + " ] }}",
    depends_on_past=True,
)

```

4. **Generating user_behaviour_metric:** This is done using the `generate_user_behavior_metric` task. This task runs a redshift SQL script to populate the `public.user_behavior_metric` table

```

generate_user_behavior_metric = PostgresOperator(
    dag=dag,
    task_id="generate_user_behavior_metric",
    sql="scripts/sql/generate_user_behavior_metric.sql",
    postgres_conn_id="redshift",
)

end_of_data_pipeline = DummyOperator(task_id="end_of_data_pipeline",
dag=dag)

```

5. This sql query generates customer level aggregate metrics:

```

-- scripts/sql/generate_user_behavior_metric.sql
DELETE FROM public.user_behavior_metric
WHERE insert_date = '{{ ds }}';
INSERT INTO public.user_behavior_metric (
    customerid,
    amount_spent,
    review_score,
    review_count,
    insert_date
)
SELECT ups.customerid,
    CAST(
        SUM(ups.Quantity * ups.UnitPrice) AS DECIMAL(18, 5)
    ) AS amount_spent,
    SUM(mrcs.positive_review) AS review_score,
    count(mrcs.cid) AS review_count,
    '{{ ds }}'
FROM spectrum.user_purchase_staging ups
JOIN (
    SELECT cid,
        CASE
            WHEN positive_review IS True THEN 1
            ELSE 0
        END AS positive_review
    FROM spectrum.classified_movie_review
    WHERE insert_date = '{{ ds }}'
) mrcs ON ups.customerid = mrcs.cid
WHERE ups.insert_date = '{{ ds }}'
GROUP BY ups.customerid;

```

Qualitative | Performance | Computational Analysis

We have implemented 5 major steps to adjust the infrastructure in our data lake and AWS.

- 1) *The data infrastructure strategy*: This data helps in procurement from the data lake and AWS Redshift to store the information and data
- 2) *Choosing a repository to collect the data*: The data has been collected in AWS S3, which enhances the efficiency of Apache Redshift
- 3) *Cleaning and optimising the data quality*: We have wrangled the data by resolving the null errors and unvalued integers
- 4) *Building an ETL Pipeline*: Take into consideration the constantly increasing requests for analysis of new information and make your pipeline ready not only for basic scripts but for more complicated data challenges
- 5) *Data Governance*: Being a significant element of the model, it helps in key enabling of the data value

Future Scope

1. **Monitoring and Alerting:** We do not have any alerting in case of task failures, data quality issues, hanging tasks, etc. In real projects, there is usually a monitoring and alerting system.
2. **Quality Control:** We can set up basic count, standard deviation, etc checks before we load data into the final table for advanced testing requirements
3. **Changing DAG frequency:** What are the changes necessary to run this DAG every hour? How will the table schema need to change to support this? Do you have to rename the DAG?
4. **Data size:** Our ETL will require improvements across data pipeline to run successfully if our data size increases by 10x, 100x, 1000x

Conclusion:

Our user behaviour metric dimensional model in OLAP warehouse allows us to extract business intelligence - we can validate if a user purchases a certain movie rental based on positive sentiment reviews. We can extract inferences on high confidence prediction scores to validate if the user would rent a similar genre movie next time. We will be able to interpret how

an average similarity between two movies a user consecutively watched is high if this user has low activity. Our project successfully set up best practices across ETL batch processing and addressed designing robust data pipelines from core business requirements.

Team Work

Harshita Mishra:

- Set up the database table and implementing in PostgreSQL, which starts the initial phase of the project
- Developed and simulating the sql queries for exploratory data analysis in database

Paras Pandhare:

- Designed the database in PostgreSQL and preparing dimensional model for OLAP
- Engineered the design and architecture of the ETL Pipeline

Siddharth Singh:

- Modelled and developed the code walkthrough, infrastructure setup, performance analysis using Apache airflow, AWS cloud and Docker

- Integrated the business problem with infrastructure setup to prepare the use case for the report

References

- [Database source](#)
- [Setting up Postgresql with database tables](#)
- [AWS CLI version with Docker image](#)
- [Setting up Docker](#)
- [End to end data engineering project example](#)
- [Dimensional modelling](#)