Task 1: Movie Ratings Data Ingestion

CSV data representing Movie ratings

UserID	MovieID	Rating	Timestamp	UserID
U001	M001	4	2024-05-01 14:30:00	U001
U002	M002	5	2024-05-01 16:00:00	U002
U003	M001	3	2024-05-02 10:15:00	U003
U001	M003	2	2024-05-02 13:45:00	U001
U004	M002	4	2024-05-03 18:30:00	U004

```
dbutils.fs.cp(“file:/Workspace/Shared/movie_ratings.csv”,”dbfs:/FileStore/movie_ratings.csv”)
```

- - Ingest this CSV data into a Delta table in Databricks.

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import col, to_timestamp

from pyspark.sql.types import StructType, StructField, StringType, IntegerType, TimestampType

import logging


spark = SparkSession.builder \
    .appName("Movie Ratings Ingestion") \
    .getOrCreate()


schema = StructType([
    StructField("UserID", StringType(), True),
    StructField("MovieID", StringType(), True),
    StructField("Rating", IntegerType(), True),
    StructField("Timestamp", StringType(), True)
])


logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger()
```

```
try:
    movie_ratings_df = spark.read.csv("dbfs:/FileStore/movie_ratings.csv", schema=schema,
header=True)
    movie_ratings_df = movie_ratings_df.withColumn("Timestamp", to_timestamp(col("Timestamp"),
                                                                                    "yyyy-MM-dd HH:mm:ss"))

    cleaned_df = movie_ratings_df.dropna()
    cleaned_df.write.format("delta").mode("overwrite").save("/delta/movie_ratings")
```

- - Ensure proper error handling for missing or inconsistent data, and log errors accordingly.

```
    logger.info("Movie ratings data ingested successfully.")
except Exception as e:
    logger.error(f"Error during data ingestion: {e}")
```

Task 2: Data Cleaning

- - Ensure that the Rating column contains values between 1 and 5.

- - Remove any duplicate entries (same UserID and MovieID).

```
cleaned_ratings_df = movie_ratings_df \
    .filter((col("Rating") >= 1) & (col("Rating") <= 5)) \
    .dropDuplicates(["UserID", "MovieID"])
```

- - Save the cleaned data to a new Delta table.

```
cleaned_ratings_df.write.format("delta").mode("overwrite").save("/delta/cleaned_movie_ratings")
```

Task 3: Movie Rating Analysis

- - Calculate the average rating for each movie.

```
avg_ratings_df = cleaned_ratings_df.groupBy("MovieID")\
                    .agg({"Rating": "avg"})\
                    .withColumnRenamed("avg(Rating)", "AverageRating")
```

- - Identify the movies with the highest and lowest average ratings.

```
highest_rated = avg_ratings_df.orderBy(col("AverageRating").desc()).limit(1)
lowest_rated = avg_ratings_df.orderBy(col("AverageRating").asc()).limit(1)
```

- - Save the analysis results to a Delta table.

```
avg_ratings_df.write.format("delta").mode("overwrite")\
                    .save("/delta/movie_rating_analysis")
```

Task 4: Time Travel and Delta Lake History

- - Perform an update to the movie ratings data

```
cleaned_ratings_df = cleaned_ratings_df\
            .withColumn("Rating", when(col("MovieID") == "M001", 5)\
            .otherwise(col("Rating")))
cleaned_ratings_df.write.format("delta").mode("overwrite")\
            .save("/delta/cleaned_movie_ratings")
```

- - Roll back to a previous version of the Delta table

```
rolled_back_df = spark.read.format("delta").option("versionAsOf", 1)\
            .load("/delta/cleaned_movie_ratings")
```

- - Use DESCRIBE HISTORY to view the history of changes to the Delta table.

```
spark.sql("DESCRIBE HISTORY delta.` /delta/cleaned_movie_ratings`").show()
```

Task 5: Optimize Delta Table

- - Implement Z-ordering on the MovieID column to improve query performance.

- - Use the OPTIMIZE command to compact the data and improve performance.

```
spark.sql("OPTIMIZE delta.`/delta/cleaned_movie_ratings` ZORDER BY (MovieID)")
```

- - Use VACUUM to clean up older versions of the table.

```
spark.sql("VACUUM delta.`/path/to/delta/cleaned_movie_ratings` RETAIN 0 HOURS")
```