**CAPSTONE PROJECT**

**Name:** Amit Verma

**ID:** 1394025

**------------------------------------------------------------------------------------------------------------------**

**Data Cleaning and Formatting Tasks**

**1. Using Spark Core**

**1a)**

from pyspark.sql import SparkSession

# creating sparksession and sparkcontext

spark = SparkSession.builder.getOrCreate()

sc = spark.sparkContext

# loading datasets from the hdfs into respective rdds

UserRDD = sc.textFile("./datasets/User\_Data.csv")

ContentRDD = sc.textFile("./datasets/Content\_Data.csv")

EngagementRDD = sc.textFile("./datasets/Engagement\_Data.csv")

# fetching five records excluding header from the rdds

user\_five\_records = UserRDD.take(6)

content\_five\_records = ContentRDD.take(6)

engagement\_five\_records = EngagementRDD.take(6)

# displaying the five fetched records

print("-----UserData-----")

for data in user\_five\_records:

    print(data)

print("-----ContentData-----")

for data in content\_five\_records:

    print(data)

print("-----EngagementData-----")

for data in engagement\_five\_records:

    print(data)

# calculating and printing total records in each rdd excluding header

user\_total = UserRDD.count() - 1

content\_total = ContentRDD.count() - 1

engagement\_total = EngagementRDD.count() - 1

print("Total Records in UserRDD: ", user\_total)

print("Total Records in ContentRDD: ", content\_total)

print("Total Records in EngagementRDD: ", engagement\_total)

**Output:**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**1b)**

def isValid(row, idxs):

    data = row.split(",")

    for idx in idxs:

        if (data[idx] == "?"): return False

    return True

def filterRDD(rdd, header, idxs=[0]):

    data = rdd.filter(lambda row : row != header) # remove header

    filtered\_data = data.filter(lambda row : isValid(row, idxs))

    distinct\_data = filtered\_data.distinct()

    return distinct\_data

# headers

user\_header = UserRDD.first()

content\_header = ContentRDD.first()

engagement\_header = EngagementRDD.first()

filtered\_user = filterRDD(UserRDD, user\_header)

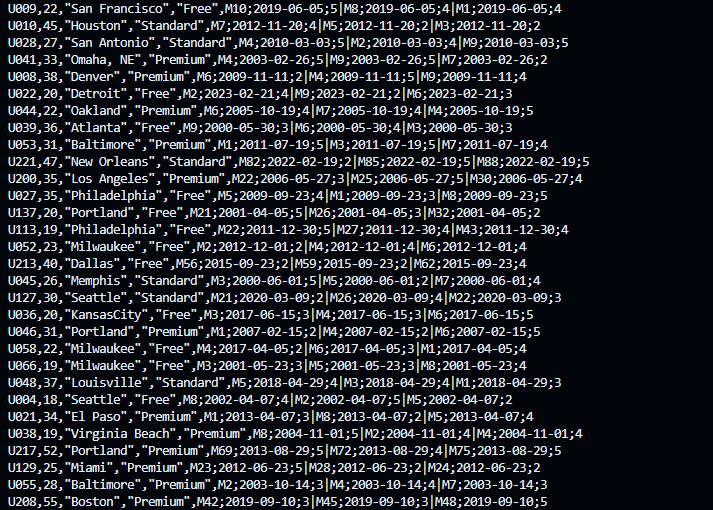
filtered\_content = filterRDD(ContentRDD, content\_header)

filtered\_engagement = filterRDD(EngagementRDD, engagement\_header, [0, 1])

# fixing showids for userrdd after removing unwanted rows

filtered\_user = filtered\_user.map(lambda line : \_\_import\_\_('re').sub(r'\dM','M',line)).distinct()

**Output:**

****

**1c)**

import shutil

import os

# output directories

user\_dir = './datasets/UserRDD.csv'

content\_dir = './datasets/ContentRDD.csv'

engagement\_dir = './datasets/EngagementRDD.csv'

# delete output directories if they exits

paths = [user\_dir, content\_dir, engagement\_dir]

for path in paths:

    if os.path.exists(path):

        shutil.rmtree(path)

# saving the cleaned

filtered\_user.saveAsTextFile(user\_dir)

filtered\_content.saveAsTextFile(content\_dir)

filtered\_engagement.saveAsTextFile(engagement\_dir)

**Output:**

**A black background with blue and white text

AI-generated content may be incorrect.**

**2. Using Spark SQL**

user\_df = spark.read.csv('./datasets/UserRDD.csv',inferSchema=True, header=False)

user\_df.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**2b)**

# create dataframe

user\_df = spark.read.csv('./datasets/UserRDD.csv', schema="UserID string,Age int,Location string,Subscription string,WatchHistory string",inferSchema=True, header=False)

# split

user\_df\_split = user\_df.withColumn("WatchHistoryArray", split(col("WatchHistory"), '\|')).withColumn("Exploded", explode(col("WatchHistoryArray")) )

user\_watch\_df = user\_df\_split.withColumn("ShowID", split(col("Exploded"), ";").getItem(0)) \

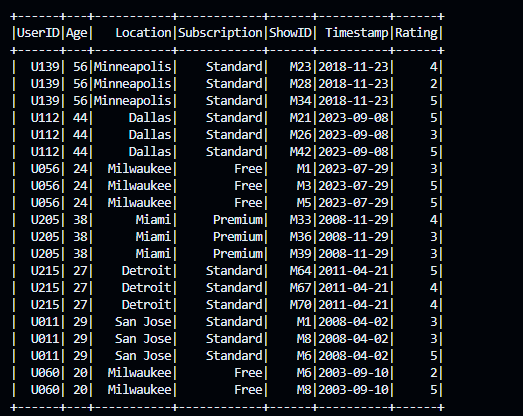
                        .withColumn("Timestamp", split(col("Exploded"), ";").getItem(1)) \

                        .withColumn("Rating", split(col("Exploded"), ";").getItem(2)) \

                        .select("UserID", "Age", "Location", "Subscription", "ShowID", "Timestamp", "Rating")

user\_watch\_df.show()

**Output:**

****

**2c)**

user\_watch\_df.write.mode("overwrite").parquet('./datasets/UserWatchData.parquet')

# create content dataframe

content\_df = spark.read.csv('./datasets/ContentRDD.csv', schema="ShowID string,Genre string,Actors string,Director string,Release\_Year int,Synopsis string",inferSchema=True, header=False)

content\_df.write.mode("overwrite").parquet('./datasets/ContentData.parquet')

# create engagement dataframe

engagement\_df = spark.read.csv('./datasets/EngagementRDD.csv', schema="UserID string,ShowID string,PlaybackStarted string,PlaybackStopped string,CompletionPercent int",inferSchema=True, header=False)

engagement\_df.write.mode("overwrite").parquet('./datasets/EngagementData.parquet')

**Output:**

**A black background with blue and white text

AI-generated content may be incorrect.**

**------------------------------------------------------------------------------------------------------------------**

**Basic Analytical Requirements**

**1.**

# selecting only unique userid and location subscribed

filteredDF = userDF.select('UserID', 'Location').distinct()

locDF = filteredDF.groupBy('Location').agg(count('UserID').alias('SubscriberCount'))

# location with maximum number of users subscribed

maxUsers = locDF.agg(max('SubscriberCount').alias('MaxCount')).first()[0]

locationsDF = locDF.filter(col('SubscriberCount') == maxUsers)

locationsDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**2.**

uniqueDF = userDF.select('UserID', 'Subscription').distinct()

aggDF = userDF.groupBy('Subscription').agg(count('UserID').alias('SubscriberCount'))

# max users and min users

maxUsers = aggDF.agg(max('SubscriberCount').alias('MaxCount')).first()[0]

minUsers = aggDF.agg(min('SubscriberCount').alias('MinCount')).first()[0]

maxUserDF = aggDF.filter(col('SubscriberCount') == maxUsers)

minUserDF = aggDF.filter(col('SubscriberCount') == minUsers)

maxUserDF.show()

minUserDF.show()

**Output:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**3.**

# paired df of showid and average rating

aggDF = userDF.groupBy('ShowID').agg(avg('Rating').alias('AverageRating'))

aggDF.show()

# shows with highest avg rating

maxRating = aggDF.agg(max('AverageRating').alias('MaxRating')).first()[0]

ratedShowDF = aggDF.filter(col('AverageRating') == maxRating)

ratedShowDF.show()

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**4.**

# unique and filtered column

uniqueDF = userDF.select('UserID', 'Location', 'Rating')

# paired df of location and average rating

aggDF = uniqueDF.groupBy('Location').agg(avg('Rating').alias('AverageRating'))

# location with highest avg rating

ratedLocationDF = aggDF.orderBy(desc('AverageRating')).limit(1)

ratedLocationDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**5.**

# paired df of showid and count of users

aggDF = userDF.groupBy('ShowID').agg(count('UserID').alias('WatchCount'))

# showid with max number of count

maxCount = aggDF.agg(max('WatchCount').alias('MaxWatchCount')).first()[0]

showDF = aggDF.filter(col('WatchCount') == maxCount)

showDF.show()

**Output:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**6.**

# earliest timestamp

earliestTimestampDF = userDF.agg(min('Timestamp').alias('EarliestTimestamp')).limit(1)

earliestTimestampDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

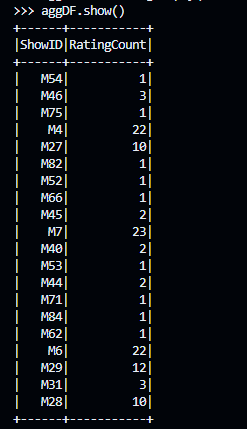
**7.**

# paired df of showid and count of ratings received

aggDF = userDF.groupBy('ShowID').agg(count('Rating').alias('RatingCount'))

aggDF.show()

**Output:**

****

**8.**

# users within age range of 18 to 30 inclusive

filteredUserDF = userDF.select('UserID', 'Age', 'Subscription').distinct().where(col('Age').between(18, 30))

# paired df of subscription and usercount

aggDF = filteredUserDF.groupBy('Subscription').agg(count('UserID').alias('UserCount'))

aggDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**9.**

# counting unique genre

genreCountDF = contentDF.select('Genre').distinct().agg(count('\*').alias('GenreCount'))

# unique genre count

genreCountDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**10.**

# creating dataframe

contentDF = spark.read.parquet('./datasets/ContentData.parquet')

# director with most shows

aggDF = contentDF.groupBy('Director').agg(count('\*').alias('ShowCount'))

# popular directors

maxShowCount = aggDF.agg(max('ShowCount').alias('MaxShowCount')).first()[0]

popularDirectors = aggDF.filter(col('ShowCount') == maxShowCount)

popularDirectors.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**11.**

# creating dataframe

contentDF = spark.read.parquet('./datasets/ContentData.parquet')

# number of shows per year

aggDF = contentDF.groupBy('Release\_Year').agg(count('\*').alias('ShowCount'))

aggDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**12.**

# splitted df by actor

splitDF = contentDF.withColumn('ActorsArray', split(col('Actors'),'\|')) \

    .withColumn('Actor', explode(col('ActorsArray')))

# filtered col

filteredDF = splitDF.select('Actor')

# count of show actor wise

aggDF = filteredDF.groupBy('Actor').agg(count('\*').alias('ShowCount'))

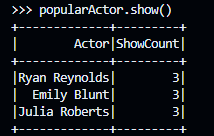
# max show count

maxShowCount = aggDF.agg(max('ShowCount').alias('MaxShowCount')).first()[0]

popularActor = aggDF.filter(col('ShowCount') == maxShowCount)

popularActor.show()

**Output:**

****

**13.**

# filtered dataframes

filtered\_user = userDF.select('UserID', 'ShowID', 'Rating')

filtered\_content = contentDF.select('ShowID','Genre')

# joined dataframe

joinedDF = filtered\_user.join(filtered\_content, on="ShowID", how="inner")

# maximum rating

aggMaxDF = joinedDF.groupBy('Genre').agg(max('Rating').alias('MaxRating'))

maxRating = aggMaxDF.agg(max('MaxRating').alias('MaxRating')).first()[0]

maxDF = aggMaxDF.filter(col('MaxRating') == maxRating)

maxDF.show()

# highest average rating

aggHighestAvgDF = joinedDF.groupBy('Genre').agg(avg('Rating').alias('HighestAvgRating'))

highestAvgRating = aggHighestAvgDF.agg(max('HighestAvgRating').alias('HighestAvgRating')).first()[0]

highestAvgDF = aggHighestAvgDF.filter(col('HighestAvgRating') == highestAvgRating)

highestAvgDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**14.**

# paired with genre and show count

aggDF = contentDF.groupBy('Genre').agg(count('Genre').alias('ShowCount'))

# show count with comedy genre

comedyCount = aggDF.where(col('Genre') == 'Comedy')

comedyCount.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**15.**

# paired with Director and showcount

aggDF = contentDF.groupBy('Director').agg(count('\*').alias('ShowCount'))

# max show and min show

maxShowCount = aggDF.agg(max('ShowCount').alias('MaxShowCount')).first()[0]

minShowCount = aggDF.agg(min('ShowCount').alias('MinShowCount')).first()[0]

maxDirectorDF = aggDF.filter(col('ShowCount') == maxShowCount)

minDirectorDF = aggDF.filter(col('ShowCount') == minShowCount)

maxDirectorDF.show()

minDirectorDF.show()

**Output:**

****

**16.**

# paired with showid and avg of completion percent

aggDF = engagementDF.groupBy('ShowID').agg(avg('CompletionPercent').alias('AverageCompletion'))

# show with maximum completion rate

maxCompletionRate = aggDF.agg(max('AverageCompletion').alias('MaxAvgCompletionRate')).first()[0]

showDF = aggDF.filter(col('AverageCompletion') == maxCompletionRate)

showDF.show()

**Output:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**17.**

# creating dataframe

engagementDF = spark.read.parquet('./datasets/EngagementData.parquet')

# column with diff

diffDF = engagementDF \

.withColumn('PlaybackStarted', to\_date('PlaybackStarted', 'yyyy-MM-dd')) \

.withColumn('PlaybackStopped', to\_date('PlaybackStopped', 'yyyy-MM-dd')) \

.withColumn('Duration', datediff('PlaybackStopped', 'PlaybackStarted'))

# grouped with userid

userDF = diffDF.groupBy('UserID').agg(sum('Duration').alias('DurationInDays'))

userDF.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**18**

# column with diff

diffDF = engagementDF \

.withColumn('PlaybackStarted', to\_date('PlaybackStarted', 'yyyy-MM-dd')) \

.withColumn('PlaybackStopped', to\_date('PlaybackStopped', 'yyyy-MM-dd')) \

.withColumn('Duration', datediff('PlaybackStopped', 'PlaybackStarted'))

# user with longest playtime

aggDF = diffDF.groupBy('UserID').agg(avg('Duration').alias('DurationInDays'))

longestPlaytime = aggDF.agg(max('DurationInDays').alias('LongestPlaytime')).first()[0]

userDF = aggDF.filter(col('DurationInDays') == longestPlaytime)

userDF.show()

**Output:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**19.**

# grouped df with show and average completion percent

aggDF = engagementDF.groupBy('ShowID').agg(avg('CompletionPercent').alias('AverageCompletionPercent'))

# filter records with less than 60%

filteredDF = aggDF.where(col('AverageCompletionPercent') < 60)

filteredDF.show()

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**20.**

# grouped df with show and average completion percent

aggDF = engagementDF.groupBy('ShowID').agg(avg('CompletionPercent').alias('AverageCompletionPercent'))

# show with lowest average completion rate

lowestCompletion = aggDF.agg(min('AverageCompletionPercent').alias('LowestCompletion')).first()[0]

showDF = aggDF.filter(col('AverageCompletionPercent') == lowestCompletion)

showDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Advanced Analytical Requirements**

**A. Content Genre Analysis**

**A1)**

# creating dataframe

query = "SELECT \* FROM users"

filteredUserDF = spark.sql(query)

filteredUserDF.show()

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A2)**

# grouped userid and genre

query = """

    SELECT u.UserID, c.Genre, u.ShowID, u.Timestamp, u.Rating

    FROM users u

    JOIN content c ON u.ShowID = c.ShowID

"""

joinedDF = spark.sql(query)

joinedDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A3)**

# average rating and total watch time grouped by userid and genre

query = """

    SELECT u.UserID, c.Genre, AVG(u.Rating) AS AverageRating, SUM(DATEDIFF(TO\_DATE(e.PlaybackStopped,'yyyy-MM-dd'), TO\_DATE(e.PlaybackStarted,'yyyy-MM-dd'))) AS TotalWatchTime

    FROM engagements e

    JOIN users u ON e.UserID = u.UserID

    JOIN content c ON e.ShowID = c.ShowID

    GROUP BY u.UserID, c.Genre

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**A4)**

# average rating for movie with specific director

query = """

        SELECT TRIM(actor) AS Actor, AVG(Rating) AS AvgRating

        FROM (

                SELECT u.Rating, EXPLODE(SPLIT(c.actors, '\\\|')) AS Actor

                FROM content c

                JOIN users u ON u.ShowID = c.ShowID

            )

        GROUP BY TRIM(Actor)

"""

actorDF = spark.sql(query)

actorDF.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**A5)**

# join using broadcast

optimizedDF = contentDF.join(broadcast(userDF), 'ShowID')

# create temp view for joineddf

optimizedDF.createOrReplaceTempView("contentWatch")

# query

query = """

    SELECT \* FROM contentWatch

"""

joinedDF = spark.sql(query)

joinedDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**B. Demographic and Subscription Tier**

**B1)**

# genre popular in age group 18 to 30

query = """

    SELECT c.Genre, count(Genre) AS Count

    FROM users u

    JOIN content c ON u.ShowID = c.ShowID

    WHERE age BETWEEN 18 AND 30

    GROUP BY c.Genre

    ORDER BY Count DESC

    LIMIT 1

"""

genreDF = spark.sql(query)

genreDF.show()

**Output:**

**A black screen with white text

AI-generated content may be incorrect.**

**B2)**

# popular movie type in New York

query = """

    WITH movieCount AS (

        SELECT c.genre, COUNT(c.genre) As movieCount

        FROM content c

        JOIN users u ON c.ShowID = u.ShowID

        WHERE u.Location = 'New York'

        GROUP BY c.Genre

        ORDER BY MovieCount DESC

    ),

    maxCount AS (

        SELECT MAX(movieCount) AS maxMovieCount

        FROM movieCount

    )

    SELECT s.genre, s.movieCount

    FROM movieCount s

    JOIN maxCount h ON s.movieCount = h.maxMovieCount

"""

genreDF = spark.sql(query)

genreDF.show()

**Output:**

**A screen shot of a computer screen

AI-generated content may be incorrect.**

**B3)**

# average rating by age group

query = """

    SELECT u.Age, AVG(u.Rating) AS AvgRating

    FROM users u

    WHERE age > 45

    GROUP BY u.Age

"""

ratingDF = spark.sql(query)

ratingDF.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**B4)**

# shows with premium subscription

query = """

    SELECT u.ShowID, u.Subscription

    FROM users u

    WHERE u.Subscription = 'Premium'

"""

subscriptionDF = spark.sql(query)

subscriptionDF.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**B5)**

# shows popular among standard users

query = """

    WITH standardUsers AS (

        SELECT \*

        FROM users

        WHERE subscription = 'Standard'

    ),

    userCounts AS (

        SELECT showid, COUNT(showid) AS usersCount

        FROM standardUsers

        GROUP BY showid

    ),

    maxCount AS (

        SELECT MAX(usersCount) AS maxCount

        FROM userCounts

    )

    SELECT s.showid, s.usersCount

    FROM userCounts s

    JOIN maxCount m

    ON s.usersCount = m.maxCount

"""

showsDF = spark.sql(query)

showsDF.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**B6)**

# comedymovie names watched by users

query = """

    SELECT DISTINCT(u.showid), c.\*

    FROM users u

    JOIN content c ON u.showid = c.showid

    WHERE c.genre = 'Comedy'

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**C. Demographic and Subscription Tier**

**C1)**

# shows with id Mxx and year 20xx

query = """

    WITH filteredShows AS (

        SELECT u.showid, COUNT(u.showid) AS watchcount

        FROM users u

        WHERE u.showid LIKE 'M%' AND u.timestamp LIKE '20%'

        GROUP BY u.showid

    )

    SELECT c.showid, c.genre, c.director, f.watchCount

    FROM content c

    JOIN filteredShows f ON f.showid = c.showid

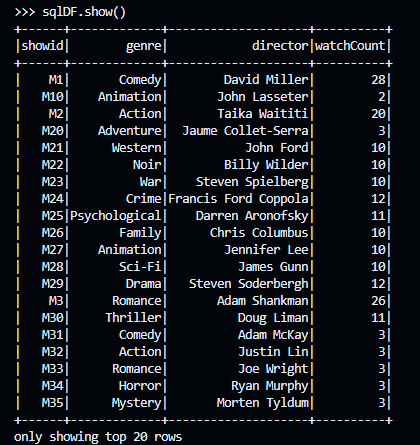
    ORDER BY c.showid ASC

"""

showsDF = spark.sql(query)

showsDF.show()

**Output:**

****

**C2)**

# movies with highest average watchtime

query = """

    WITH averageWatchtime AS (

        SELECT showid, AVG(DATEDIFF(playbackStopped, playbackStarted)) AS avgWatchTime

        FROM engagements

        GROUP BY showid

    ),

    maxAvgWatchTime AS (

        SELECT MAX(avgWatchTime) AS maxAvgWatchTime

        FROM averageWatchTime

    )

    SELECT showid, avgWatchTime

    FROM averageWatchTime a

    JOIN maxAvgWatchTime m

    ON a.avgWatchTime = m.maxAvgWatchTime

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**C3)**

# byuserid: users with longer watchtime

query1 = """

    WITH watchTime AS (

        SELECT userid, SUM(DATEDIFF(playbackStopped, playbackStarted)) AS watchTime

        FROM engagements

        GROUP BY userid

    ),

    maxWatchTime AS (

        SELECT MAX(watchTime) AS maxWatchTime

        FROM watchTime

    )

    SELECT userid, watchTime

    FROM watchTime w

    JOIN maxWatchTime m

    ON w.watchTime = m.maxWatchTime

"""

sql1DF = spark.sql(query1)

sql1DF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

# byshowid: shows with higher or lower average watchtime

query2 = """

    WITH watchTime AS (

        SELECT showid, SUM(DATEDIFF(playbackStopped, playbackStarted)) AS watchTime

        FROM engagements

        GROUP BY showid

    ),

    maxWatchTime AS (

        SELECT MAX(watchTime) AS maxWatchTime

        FROM watchTime

    ),

    minWatchTime AS (

        SELECT MIN(watchTime) AS minWatchTime

        FROM watchTime

    )

    SELECT w.showid, w.watchTime

    FROM watchTime w

    JOIN maxWatchTime h ON w.watchTime = h.maxWatchTime

    UNION

    SELECT w.showid, w.watchTime

    FROM watchTime w

    JOIN minWatchTime l ON w.watchTime = l.minWatchTime

"""

sql2DF = spark.sql(query2)

sql2DF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**D. Completion Rate Analysis**

**D1)**

# user with most completed movies

query = """

    WITH completedUsers AS (

        SELECT userid, completionpercent

        FROM engagements

        WHERE completionpercent = 100

    ),

    countUsers AS (

        SELECT userid, COUNT(\*) AS userCount

        FROM completedUsers

        GROUP BY userid

    ),

    maxCountUsers AS (

        SELECT MAX(userCount) AS maxUserCount

        FROM countUsers

    )

    SELECT c.userid, c.userCount

    FROM countUsers c

    JOIN maxCountUsers m ON c.userCount = m.maxUserCount

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**D2)**

# shows with average completion rate

query = """

    SELECT showid, AVG(completionPercent) AS AverageCompletionPercent

    FROM engagements

    GROUP BY showid

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

**A computer screen shot of numbers

AI-generated content may be incorrect.**

**D3)**

query = """

    SELECT userID, AVG(DATEDIFF(TO\_DATE(e.playbackStopped), TO\_DATE(e.playbackStarted))) AS averageWatchTime, AVG(completionPercent) AS averageCompletionPercent

    FROM engagements e

    GROUP BY e.userID

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**E. Content Type Analysis**

**E1)**

# shows that have not been watched

query = """

    SELECT c.\*

    FROM content c

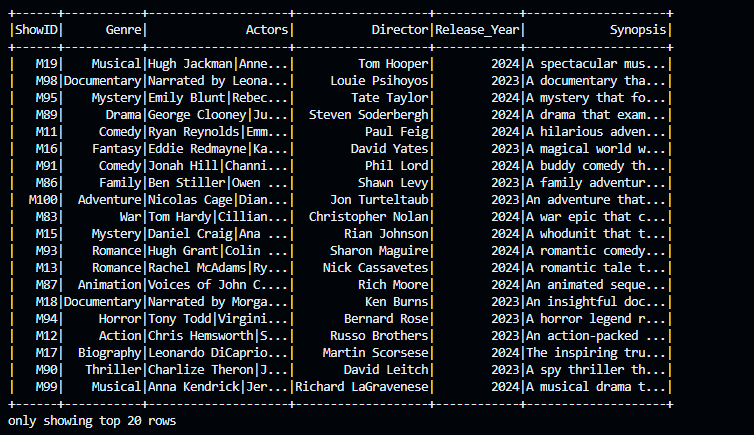
    WHERE c.showid NOT IN (SELECT showid FROM users)

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

****

**E2)**

# higher engagement based on genre

query1 = """

    SELECT genre, AVG(completionPercent) AS averageCompletionPercent, AVG(DATEDIFF(TO\_DATE(playbackStopped), TO\_DATE(playbackStarted))) AS averageWatchTime

    FROM content c

    JOIN engagement e ON c.showid = e.showid

    GROUP BY genre

"""

sql1DF = spark.sql(query1)

sql1DF.show()

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

# higher engagement based on actor

query2 = """

    SELECT actor, AVG(completionPercent) AS averageCompletionPercent, AVG(DATEDIFF(TO\_DATE(playbackStopped,'yyyy-MM-dd'), TO\_DATE(playbackStarted,'yyyy-MM-dd'))) AS averageWatchTime

    FROM (

        SELECT playbackStarted, playbackStopped, completionPercent, EXPLODE(SPLIT(actors, '\\\\|')) AS actor

        FROM engagement e

        JOIN content c ON e.showid = c.showid

    )

    GROUP BY actor

"""

sql2DF = spark.sql(query2)

sql2DF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**F. Content Analysis**

**F1)**

# load dataset to dataframe

contentDF = spark.read.parquet('./datasets/ContentData.parquet')

userDF = spark.read.parquet('./datasets/UsersWatchData.parquet')

**F2)**

# joined table

query = """

    SELECT \*

    FROM users u

    JOIN content ON u.showid = c.showid

"""

joinedDF = spark.sql(query)

joinedDF.show()

**Output:**

**A screen shot of a computer screen

AI-generated content may be incorrect.**

**F3)**

# shows with high ratings

query = """

    SELECT u.userid, u.rating, c.\*

    FROM users u

    JOIN content c ON u.showid = c.showid

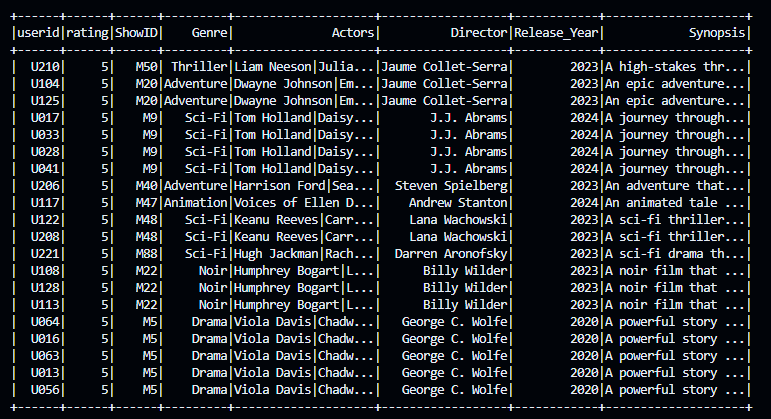
    WHERE rating = 5

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

****

**F4)**

query = """

    WITH genreRank AS (

        SELECT u.userid, c.genre, CAST(u.rating AS INT) AS rating, RANK() OVER (PARTITION BY u.userid ORDER BY u.rating DESC) AS genreRanking

        FROM users u

        JOIN content c ON u.showid = c.showid

    )

    SELECT userid, genre, rating

    FROM genreRank

    WHERE genreRanking = '1'

    ORDER BY userid

"""

favGenreDF = spark.sql(query)

favGenreDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**F5)**

# movies from fav genre that users have not watched

query = """

    WITH genreRank AS (

        SELECT u.userid, c.genre, u.showid, CAST(u.rating AS INT) AS rating, RANK() OVER (PARTITION BY u.userid ORDER BY u.rating DESC) AS genreRanking

        FROM users u

        JOIN content c ON u.showid = c.showid

    )

    SELECT ug.userid, ug.genre AS favGenre, c.showid AS suggestedShow

    FROM content c

    JOIN genreRank ug ON c.genre = ug.genre AND genreRanking = 1 AND c.showid != ug.showid

    ORDER BY ug.userid, favGenre, suggestedShow

"""

moviesDF = spark.sql(query)

moviesDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**G. Recent Activity Analysis and Recommendations**

**G1)**

# group by watchhistory by userid

query = """

    SELECT UserId, COLLECT\_LIST(CONCAT\_WS( '|', ShowID , Timestamp, Rating)) AS WatchHistory

    FROM users group by UserID

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

**A computer screen shot of a number

AI-generated content may be incorrect.**

**G2)**

# group userid with watchhistoty by timstamp in desc order

query = """

    SELECT UserID, TO\_DATE(Timestamp,'yyyy-MM-dd') AS LastDateWatched

    FROM users

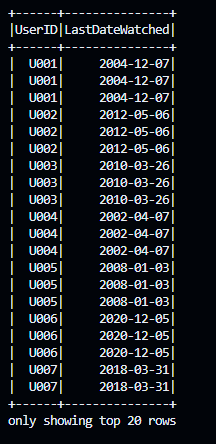
    ORDER BY UserID, LastDateWatched DESC

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**



**G3)**

# recently watched content withing last week or month

query = """

    SELECT c.\*

    FROM engagement e

    JOIN content c ON e.showid = c.showid

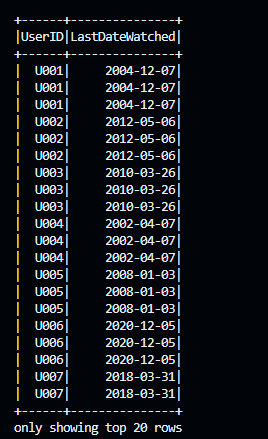
    ORDER BY TO\_DATE(e.playbackStarted) DESC

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

****

**G4)**

# recently watched content with highratings

query = """

    WITH recentlyWatched AS (

        SELECT u.UserID, e.ShowID, TO\_DATE(playbackStopped,'yyyy-MM-dd') AS LastDateWatched, Rating

        FROM engagement e

        JOIN users u ON e.showid = u.showid

        ORDER BY LastDateWatched DESC LIMIT 10

    )

    SELECT \*

    FROM recentlyWatched

    ORDER BY Rating DESC

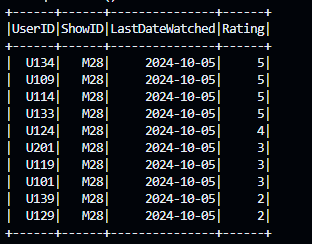
    LIMIT 10

"""

sqlDF = spark.sql(query)

sqlDF.show()

**Output:**

****

**G5)**

# recent activity

recentContentDF = spark.sql("SELECT \* FROM users ORDER BY timestamp DESC")

recentContentDF.show()

# creating tempview for recentData

recentContentDF.createOrReplaceTempView('recentWatchedContent')

# content details from recent watchhistory

query1 = """

    SELECT r.userid, c.genre, c.actors, c.director

    FROM recentWatchedContent r

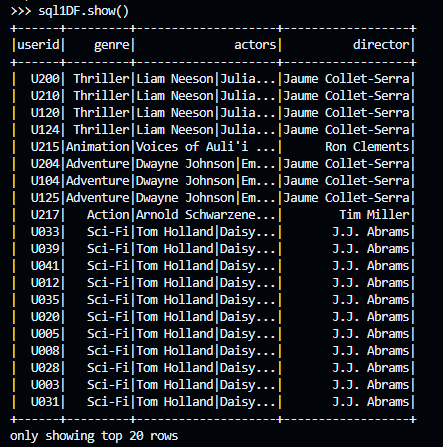
    JOIN content c ON r.showid = c.showid

"""

sql1DF = spark.sql(query1)

sql1DF.show()

**Output:**

****

# number of unique users watching specific genre or actor

query2="""

    SELECT actor, COUNT(DISTINCT userid) AS uniqueUsers

    FROM (

        SELECT r.userid, c.showid, EXPLODE(SPLIT(c.actors, '\\\|')) AS actor

        FROM content c

        JOIN recentWatchedContent r ON r.showid = c.showid

    ) t

    GROUP BY actor

    ORDER BY uniqueUsers DESC

"""

sql2DF = spark.sql(query2)

sql2DF.show()

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

# recommend content on recenlty watched with high ratings

query3 = """

    SELECT c.showid, c.genre

    FROM recentWatchedContent rs

    JOIN content c ON r.showid = c.showid

    WHERE r.rating >= 4

    GROUP BY c.showid, c.genre

"""

recommendDF = spark.sql(query3)

recommendDF.show()

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**