**This Project aims to explore the u.user file from the Movielnes 100k Dataset.**

**Download the file to a local temp folder**

%sh

wget http://media.sundog-soft.com/hadoop/ml-100k/u.user -O /tmp/u.user

wget http://media.sundog-soft.com/hadoop/ml-100k/u.data -O /tmp/u.data

wget http://media.sundog-soft.com/hadoop/ml-100k/u.item -O /tmp/u.item

echo "Downloaded ml-100k dataset!"

**Remove the directory if already available**

%sh

hadoop fs -rm -r -f /tmp/ml-100k

**Create a new directory**

%sh

hadoop fs -mkdir /tmp/ml-100k/

**Put the file from local to the HDFS**

%sh

hadoop fs -put /tmp/u.user /tmp/ml-100k/

hadoop fs -put /tmp/u.data /tmp/ml-100k/

hadoop fs -put /tmp/u.item /tmp/ml-100k/

**Ensure tables have been created in Cassandra**

CREATE TABLE ratings (

user\_id int,

movieid int,

rating int,

PRIMARY KEY (user\_id, movieid)

);

CREATE TABLE titles (

movieid int PRIMARY KEY,

title text,

unknown int,

action int,

adventure int,

animation int,

children int,

comedy int,

crime int,

documentary int,

drama int,

fantasy int,

filmnoir int,

horror int,

musical int,

mystery int,

romance int,

scifi int,

thriller int,

war int,

western int

);

CREATE TABLE users (

user\_id int,

age int,

gender text,

occupation text,

zip text,

PRIMARY KEY (user\_id)

);

**Create RDD for Spark analysis**

**Create Users RDD and DataFrame**

print(spark.version)

%pyspark

from pyspark.sql import Row

from pyspark.sql import functions as F

# ParseInput function

def parseInput(line):

fields = line.split('|')

return Row(user\_id=int(fields[0]), age=int(fields[1]), gender=fields[2], occupation=fields[3], zip=fields[4])

# Load raw user data from HDFS

lines = spark.sparkContext.textFile("hdfs:///tmp/ml-100k/u.user")

# Apply transformation

users = lines.map(parseInput)

# Create DataFrame

usersDataset = spark.createDataFrame(users)

# Write to Cassandra

usersDataset.write \

.format("org.apache.spark.sql.cassandra") \

.mode("append") \

.options(table="users", keyspace="movielens") \

.save()

# Read back from Cassandra

readUsers = spark.read \

.format("org.apache.spark.sql.cassandra") \

.options(table="users", keyspace="movielens") \

.load()

readUsers.show()

**Create Ratings RDD and DataFrame**

%pyspark

from pyspark.sql import Row

from pyspark.sql import functions as F

# Parse each line into (user\_id, movieid, rating)

def parseInput2(line):

fields = line.split('\t')

return Row(user\_id=int(fields[0]), movie\_id=int(fields[1]), rating=int(fields[2]))

# Load ratings data from HDFS

lines = spark.sparkContext.textFile("hdfs:///tmp/ml-100k/u.data")

# Map lines to Rows

ratings = lines.map(parseInput2)

# Create DataFrame from ratings

ratingsDataset = spark.createDataFrame(ratings)

# Write to Cassandra

ratingsDataset.write \

.format("org.apache.spark.sql.cassandra") \

.mode("append") \

.options(table="ratings", keyspace="movielens") \

.save()

# Read back from Cassandra

readRatings = spark.read \

.format("org.apache.spark.sql.cassandra") \

.options(table="ratings", keyspace="movielens") \

.load()

# Show the result

readRatings.show()

**Create Movie RDD and DataFrame**

%pyspark

from pyspark.sql import Row

from pyspark.sql import functions as F

# Parse movie lines

def parse\_movie(line):

fields = line.split('|')

return Row(

movie\_id=int(fields[0]),

title=fields[1],

unknown=int(fields[5]),

action=int(fields[6]),

adventure=int(fields[7]),

animation=int(fields[8]),

children=int(fields[9]),

comedy=int(fields[10]),

crime=int(fields[11]),

documentary=int(fields[12]),

drama=int(fields[13]),

fantasy=int(fields[14]),

filmnoir=int(fields[15]),

horror=int(fields[16]),

musical=int(fields[17]),

mystery=int(fields[18]),

romance=int(fields[19]),

scifi=int(fields[20]),

thriller=int(fields[21]),

war=int(fields[22]),

western=int(fields[23])

)

# Load raw movie data from HDFS

movie\_lines = spark.sparkContext.textFile("hdfs:///tmp/ml-100k/u.item")

# Apply parsing

movies = movie\_lines.map(parse\_movie)

# Convert to DataFrame

movies\_df = spark.createDataFrame(movies)

# Write to Cassandra

movies\_df.write \

.format("org.apache.spark.sql.cassandra") \

.mode("append") \

.options(table="titles", keyspace="movielens") \

.save()

# Read back from Cassandra

readMovies = spark.read \

.format("org.apache.spark.sql.cassandra") \

.options(table="titles", keyspace="movielens") \

.load()

# Show result

readMovies.show()

**1) Calculate the average rating for each movie.**

%pyspark

from pyspark.sql.functions import avg

# Join ratings and movie metadata on movieid

joined\_df = readRatings.join(readMovies, on="movie\_id")

# Group by title and compute average rating

avg\_ratings\_df = joined\_df.groupBy("title").agg(avg("rating").alias("avg\_rating"))

# Order by average rating descending and get top 20

top\_20\_movies = avg\_ratings\_df.orderBy("avg\_rating", ascending=False).limit(20)

# Show the result

top\_20\_movies.show()

**2) Identify the top ten movies with the highest average ratings.**

%pyspark

top\_20\_movies.limit(10).show()

**3) Find the users who have rated at least 50 movies and identify their favourite movie genres.**

%pyspark

from pyspark.sql.functions import col, count, row\_number

from pyspark.sql.window import Window

# Filter users who rated at least 50 movies

filtered\_users = readRatings.groupBy("user\_id") \

.agg(count("\*").alias("num\_ratings")) \

.filter(col("num\_ratings") >= 50) \

.select("user\_id")

# Join ratings with filtered users and titles

filtered\_ratings = readRatings \

.join(filtered\_users, on="user\_id") \

.join(readMovies, on="movie\_id")

# Unpivot genre columns

genre\_columns = [

"unknown", "action", "adventure", "animation", "children", "comedy", "crime",

"documentary", "drama", "fantasy", "filmNoir", "horror", "musical", "mystery",

"romance", "sciFi", "thriller", "war", "western"

]

# Flatten genres (manual unpivot via union)

user\_genres = None

for genre in genre\_columns:

temp = filtered\_ratings.filter(col(genre) == 1) \

.select("user\_id").withColumn("genre", F.lit(genre))

user\_genres = temp if user\_genres is None else user\_genres.union(temp)

# Count genre frequencies per user

user\_genre\_counts = user\_genres.groupBy("user\_id", "genre") \

.agg(count("\*").alias("count"))

# Rank genres per user

windowSpec = Window.partitionBy("user\_id").orderBy(col("count").desc())

ranked\_genres = user\_genre\_counts.withColumn("rank", row\_number().over(windowSpec))

# Select top genre per user

top\_genres\_per\_user = ranked\_genres.filter(col("rank") == 1) \

.orderBy(col("count").desc())

top\_genres\_per\_user.select("user\_id", "genre", "count").show()

**4) Find all the users who are less than 20 years old.**

%pyspark

young = readUsers.filter("age < 20")

young.show()

**5) Find all the users whose occupation is “scientist” and whose age is between 30 and 40 years old.**

%pyspark

scientist3040 = readUsers.filter(

(col("age") > 29) & (col("age") < 41) & (col("occupation") == "scientist"))

scientist3040.show()