Coding Challenge 3

PySpark

# E-commerce Transactions

#Exercises:

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import sum,avg,col


spark = SparkSession.builder.appName("E-commerce Transactions").getOrCreate()

df = spark.read.csv("/content/sample_data/Ecommercedata.csv", header=True,
inferSchema=True)


#1. Calculate the Total Revenue per Category

transactions = df.withColumn("revenue", col("price") * col("quantity") * (1 -
col("discount_percentage") / 100))

total_revenue_per_category =
transactions.groupBy("category").agg(sum("revenue").alias("total_revenue"))

print("Total Revenue:")

total_revenue_per_category.show()


#2. Filter Transactions with a Discount Greater Than 10%

transactions_with_discount = df.filter(col("discount_percentage") > 10)

print("Transactions with Discount Greater Than 10%:")

transactions_with_discount.show()


#3. Find the Most Expensive Product Sold

most_expensive_product = transactions.orderBy(col("price").desc()).limit(1)

print("Most Expensive Product Sold:")

most_expensive_product.show()
```

```
#4. Calculate the Average Quantity of Products Sold per Category
average_quantity_per_category =
transactions.groupBy("category").agg(avg("quantity").alias("average_quantity"))
print("Average Quantity of Products Sold per Category:")
average_quantity_per_category.show()

#5. Identify Customers Who Purchased More Than One Product
customers_multiple_purchases = transactions.filter(col("quantity") > 1)
print("Customers Who Purchased More Than One Product:")
customers_multiple_purchases.show()

#6. Find the Top 3 Highest Revenue Transactions
top_3_revenue = transactions.withColumn("revenue", col("price") * col("quantity") * (1 -
col("discount_percentage") / 100))\
.orderBy(col("revenue").desc())\
.limit(3)
print("Top 3 Highest Revenue Transactions:")
top_3_revenue.show()

#7. Calculate the Total Number of Transactions per Day
transactions_per_day =
transactions.groupBy("transaction_date").count().orderBy("transaction_date")
print("Total Number of Transactions per Day:")
transactions_per_day.show()

#8. Find the Customer Who Spent the Most Money
customer_total_spent =
transactions.groupBy("customer_id").agg(sum("revenue").alias("total_spent"))
customer_most_spent = customer_total_spent.orderBy(col("total_spent").desc()).limit(1)
print("Customer Who Spent the Most Money:")
customer_most_spent.show()
```

```
#9. Calculate the Average Discount Given per Product Category
average_discount_per_category =
transactions.groupBy("category").agg(avg("discount_percentage").alias("average_discount"))
print("Average Discount Given per Product Category:")
average_discount_per_category.show()

#10. Create a New Column for Final Price After Discount
transactions_with_final_price = transactions.withColumn("final_price", col("price") * (1 -
col("discount_percentage") / 100))
print("New Column for Final Price After Discount:")
transactions_with_final_price.show()
```

# Banking Transactions

#Exercises:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum,avg,col,when

spark = SparkSession.builder.appName("Banking Transactions").getOrCreate()
df = spark.read.csv("/content/sample_data/bankdata.csv", header=True, inferSchema=True)

#1. Calculate the Total Deposit and Withdrawal Amounts
total_amounts = df.groupBy("transaction_type").agg(sum("amount").alias("total_amount"))
print("Total Deposit and Withdrawal Amounts:")
total_amounts.show()

#2. Filter Transactions Greater Than $3,000
transactions_above_3000 = df.filter(col("amount") > 3000)
print("Transactions Greater Than $3,000:")
transactions_above_3000.show()
```

```
#3. Find the Largest Deposit Made

largest_deposit = df.filter(col("transaction_type") ==
"Deposit").orderBy(col("amount").desc()).limit(1)

print("Largest Deposit Made:")

largest_deposit.show()


#4. Calculate the Average Transaction Amount for Each Transaction Type

average_amount_per_type =
df.groupBy("transaction_type").agg(avg("amount").alias("average_amount"))

print("Average Transaction Amount for Each Transaction Type:")

average_amount_per_type.show()


#5. Find Customers Who Made Both Deposits and Withdrawals

customers_deposits_withdrawals =
df.groupBy("customer_id").pivot("transaction_type").agg(sum("amount"))

customers_deposits_withdrawals = customers_deposits_withdrawals.filter((col("Deposit") >
0) & (col("Withdrawal") > 0))

print("Customers Who Made Both Deposits and Withdrawals:")

customers_deposits_withdrawals.show()


#6. Calculate the Total Amount of Transactions per Day

transactions_per_day =
df.groupBy("transaction_date").agg(sum("amount").alias("total_amount"))

print("Total Amount of Transactions per Day:")

transactions_per_day.show()


#7. Find the Customer with the Highest Total Withdrawal

total_withdrawals = df.filter(col("transaction_type") ==
"Withdrawal").groupBy("customer_id").agg(sum("amount").alias("total_withdrawn"))

customer_with_max_withdrawal =
total_withdrawals.orderBy(col("total_withdrawn").desc()).limit(1)

print("Customer with the Highest Total Withdrawal:")

customer_with_max_withdrawal.show()
```

```
#8. Calculate the Number of Transactions for Each Customer

transactions_per_customer = df.groupBy("customer_id").count()

print("Number of Transactions for Each Customer:")

transactions_per_customer.show()


#9. Find All Transactions That Occurred on the Same Day as a Withdrawal Greater than $1,000

withdrawal_dates = df.filter((col("transaction_type") == "Withdrawal") & (col("amount") > 1000))\

.select("transaction_date").distinct()


same_day_transactions = df.join(withdrawal_dates, "transaction_date", "inner")

print("All Transactions That Occurred on the Same Day as a Withdrawal Greater than $1,000:")

same_day_transactions.show()


#10. Create a New Column to Classify Transactions as "High" or "Low" Value

transactions_with_value_class = df.withColumn("transaction_value", when(col("amount") > 5000, "High").otherwise("Low"))

print("New Column to Classify Transactions as High or Low Value:")

transactions_with_value_class.show()
```

# Health & Fitness Tracker Data

#Exercises:

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import sum,avg,col,row_number,collect_set,size,count,when


spark = SparkSession.builder.appName("Fitness Tracker").getOrCreate()


df = spark.read.csv("/content/sample_data/healthdata.csv", header=True, inferSchema=True)
```

```
#1. Find the Total Steps Taken by Each User

total_steps_per_user = df.groupBy("user_id").agg(sum("steps").alias("total_steps"))

print("Total Steps Taken by Each User:")

total_steps_per_user.show()


#2. Filter Days with More Than 10,000 Steps

high_activity_days = df.filter(col("steps") > 10000)

print("Days with More Than 10,000 Steps:")

high_activity_days.show()


#3. Calculate the Average Calories Burned by Workout Type

average_calories_per_workout =
df.groupBy("workout_type").agg(avg("calories_burned").alias("average_calories"))

print("Average Calories Burned by Workout Type:")

average_calories_per_workout.show()


#4. Identify the Day with the Most Steps for Each User

from pyspark.sql.window import Window

window = Window.partitionBy("user_id").orderBy(col("steps").desc())


most_steps_per_user = df.withColumn("row_num",
row_number().over(window)).filter(col("row_num") == 1)

print("Day with the Most Steps for Each User:")

most_steps_per_user.select("user_id", "date", "steps").show()


#5. Find Users Who Burned More Than 600 Calories on Any Day

high_calorie_users = df.filter(col("calories_burned") > 600)

print("Users Who Burned More Than 600 Calories on Any Day:")

high_calorie_users.show()
```

```
#6. Calculate the Average Hours of Sleep per User

average_sleep_per_user =
df.groupBy("user_id").agg(avg("hours_of_sleep").alias("average_sleep"))

print("Average Hours of Sleep per User:")

average_sleep_per_user.show()


#7. Find the Total Calories Burned per Day

total_calories_per_day =
df.groupBy("date").agg(sum("calories_burned").alias("total_calories"))

print("Total Calories Burned per Day:")

total_calories_per_day.show()


#8. Identify Users Who Did Different Types of Workouts

users_multiple_workouts =
df.groupBy("user_id").agg(collect_set("workout_type").alias("workout_types"))


users_multiple_workouts = users_multiple_workouts.filter(size(col("workout_types")) > 1)

print("Users Who Did Different Types of Workouts:")

users_multiple_workouts.show()


#9. Calculate the Total Number of Workouts per User

total_workouts_per_user =
df.groupBy("user_id").agg(count("workout_type").alias("total_workouts"))

print("Total Number of Workouts per User:")

total_workouts_per_user.show()


#10. Create a New Column for "Active" Days

tracker_data_with_active_days = df.withColumn("active_day", when(col("steps") > 10000,
"Active").otherwise("Inactive"))

print("New Column for Active Days:")

tracker_data_with_active_days.show()
```

#Music Streaming

#Exercises:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum,avg,col,row_number,collect_set,size,count,when

spark = SparkSession.builder.appName("Music Streaming").getOrCreate()

df = spark.read.csv("/content/sample_data/musicdata.csv", header=True, inferSchema=True)

#1. Calculate the Total Listening Time for Each User
total_listening_time =
df.groupBy("user_id").agg(sum("duration_seconds").alias("total_listening_time"))
print("Total Listening Time for Each User:")
total_listening_time.show()

#2. Filter Songs Streamed for More Than 200 Seconds
long_songs = df.filter(col("duration_seconds") > 200)
print("Songs Streamed for More Than 200 Seconds:")
long_songs.show()

#3. Find the Most Popular Artist (by Total Streams)
most_popular_artist =
df.groupBy("artist").agg(count("song_title").alias("total_streams")).orderBy(col("total_strea
ms").desc())
print("Most Popular Artist:")
most_popular_artist.show(1)
```

```
#4. Identify the Song with the Longest Duration

longest_song = df.orderBy(col("duration_seconds").desc()).limit(1)

print("Song with the Longest Duration:")

longest_song.show()


#5. Calculate the Average Song Duration by Artist

average_duration_per_artist =
df.groupBy("artist").agg(avg("duration_seconds").alias("average_duration"))

print("Average Song Duration by Artist:")

average_duration_per_artist.show()


#6. Find the Top 3 Most Streamed Songs per User

from pyspark.sql.window import Window


window = Window.partitionBy("user_id").orderBy(col("duration_seconds").desc())


top_songs_per_user = df.withColumn("rank", row_number().over(window)).filter(col("rank")
<= 3)

print("Top 3 Most Streamed Songs per User:")

top_songs_per_user.show()


#7. Calculate the Total Number of Streams per Day

from pyspark.sql.functions import to_date


streams_per_day = df.withColumn("streaming_date",
to_date("streaming_time")).groupBy("streaming_date").count()

print("Total Number of Streams per Day:")

streams_per_day.show()


#8. Identify Users Who Streamed Songs from More Than One Artist

users_multiple_artists = df.groupBy("user_id").agg(collect_set("artist").alias("artists_list"))
users_multiple_artists = users_multiple_artists.filter(size(col("artists_list")) > 1)
print("Users Who Streamed Songs from More Than One Artist:")
users_multiple_artists.show()

#9. Calculate the Total Streams for Each Location
streams_per_location = df.groupBy("location").count().alias("total_streams")
print("Total Streams for Each Location:")
streams_per_location.show()

#10. Create a New Column to Classify Long and Short Songs
df_with_song_length = df.withColumn("song_length", when(col("duration_seconds") > 200,
"Long").otherwise("Short"))
print("New Column to Classify Long and Short Songs:")
df_with_song_length.show()
```

#Retail Store Sales

#Exercises:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum,avg,col,row_number,collect_set,size,count,when

spark = SparkSession.builder.appName("Retail Store Sales").getOrCreate()
df = spark.read.csv("/content/sample_data/retailsalesdata.csv", header=True,
inferSchema=True)

#1. Calculate the Total Revenue per Category
print("Total Revenue per Category:")
df.withColumn('revenue', col('price') * col('quantity')) \
.groupBy('category') \
.agg(sum('revenue').alias('total_revenue')) \ .show()
```

```python
#2. Filter Transactions Where the Total Sales Amount is Greater Than $100
print("Transactions Where the Total Sales Amount is Greater Than $100:")
df.withColumn('total_sales', col('price') * col('quantity')) \
.filter(col('total_sales') > 100) \
.show()

#3. Find the Most Sold Product
print("Most Sold Product:")
df.groupBy('product_name') \
.agg(sum('quantity').alias('total_quantity')) \
.orderBy(col('total_quantity').desc()) \
.limit(1) \
.show()

#4. Calculate the Average Price per Product Category
print("Average Price per Product Category:")
df.groupBy('category') \
.agg(avg('price').alias('average_price')) \
.show()

#5. Find the Top 3 Highest Grossing Products
print("Top 3 Highest Grossing Products:")
df.withColumn('revenue', col('price') * col('quantity')) \
.groupBy('product_name') \
.agg(sum('revenue').alias('total_revenue')) \
.orderBy(col('total_revenue').desc()) \
.show(3)

#6. Calculate the Total Number of Items Sold per Day
print("Total Number of Items Sold per Day:")
```

```
df.groupBy('sales_date') \
.agg(sum('quantity').alias('total_quantity')) \
.show()

#7. Identify the Product with the Lowest Price in Each Category
window = Window.partitionBy('category').orderBy(col('price'))
print("Product with the Lowest Price in Each Category:")
df.withColumn('rank', row_number().over(window)) \
.filter(col('rank') == 1) \
.select('category', 'product_name', 'price') \
.show()

#8. Calculate the Total Revenue for Each Product
print("Total Revenue for Each Product:")
df.withColumn('revenue', col('price') * col('quantity')) \
.groupBy('product_name') \
.agg(sum('revenue').alias('total_revenue')) \
.show()

#9. Find the Total Sales per Day for Each Category
print("Total Sales per Day for Each Category:")
df.withColumn('revenue', col('price') * col('quantity')) \
.groupBy('sales_date', 'category') \
.agg(sum('revenue').alias('total_sales')) \
.show()

#10. Create a New Column for Discounted Price
print("New Column for Discounted Price:")
df.withColumn('discounted_price', col('price') * 0.9) \
.show()
```