PYSPARK CODING ASSESSMENT

EXERCISE 1:

from pyspark.sql import SparkSession

from pyspark.sql.functions import col,avg

from pyspark.sql import functions as F

from pyspark.sql.window import Window

spark = SparkSession.builder \

.appName("E-Commerce Transactions") \

.getOrCreate()

data=[

(1,101,'Laptop','Electronics',1000,1,10,'2023-08-01'),

(2,102,'Smartphone','Electronics',700,2,5,'2023-08-01'),

(3,103,'Shirt','Fashion',40,3,0,'2023-08-02'),

(4,104,'Blender','Home Appliance',150,1,15,'2023-08-03'),

(5,101,'Headphones','Electronics',100,2,10,'2023-08-03'),

(6,105,'Shoes','Fashion',60,1,20,'2023-08-04'),

(7,106,'Refrigerator','Home Appliance',800,1,25,'2023-08-05'),

(8,107,'Book','Books',20,4,0,'2023-08-05'),

(9,108,'Toaster','Home Appliance',30,1,5,'2023-08-06'),

(10,102,'Tablet','Electronics',300,2,10,'2023-08-06')

]

columns=['transaction\_id','customer\_id','product','category','price','quantity','discount\_percentage','transaction date']

df= spark.createDataFrame(data,columns)

df.show()

#Exercises:

#1. Calculate the Total Revenue per Category

total\_revenue\_per\_category=df.groupBy('category').agg(F.sum('price').alias('total\_revenue'))

total\_revenue\_per\_category.show()

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

filter\_by\_discount = df.filter(col('discount\_percentage')>10)

filter\_by\_discount.show()

#3. Find the Most Expensive Product Sold

most\_expensive\_product = df.orderBy(col('price').desc()).limit(1)

most\_expensive\_product.show()

#4. Calculate the Average Quantity of Products Sold per Category

average\_quantity\_per\_category = df.groupBy('category').agg(F.avg('quantity').alias('average\_quantity'))

average\_quantity\_per\_category.show()

#5. Identify Customers Who Purchased More Than One Product

customers\_purchased\_more\_than\_one = df.groupBy('customer\_id').agg(F.countDistinct('product').alias('product\_count'))

customers\_purchased\_more\_than\_one = customers\_purchased\_more\_than\_one.filter(col('product\_count')>1)

customers\_purchased\_more\_than\_one.show()

#6. Find the Top 3 Highest Revenue Transactions

top\_3\_highest\_revenue\_transactions = df.orderBy(col('price').desc()).limit(3)

top\_3\_highest\_revenue\_transactions.show()

#7. Calculate the Total Number of Transactions per Day

total\_transactions\_per\_day = df.groupBy('transaction date').count()

total\_transactions\_per\_day.show()

#8. Find the Customer Who Spent the Most Money

customer\_spent\_most\_money = df.groupBy('customer\_id').agg(F.sum('price').alias('total\_spent'))

customer\_spent\_most\_money = customer\_spent\_most\_money.orderBy(col('total\_spent').desc()).limit(1)

customer\_spent\_most\_money.show()

#9. Calculate the Average Discount Given per Product Category

average\_discount\_per\_category = df.groupBy('category').agg(F.avg('discount\_percentage').alias('average\_discount'))

average\_discount\_per\_category.show()

#10. Create a New Column for Final Price After Discount

df = df.withColumn('final\_price',col('price')\*(1-col('discount\_percentage')/100))

df.show()

EXERCISE 2:

from pyspark.sql import SparkSession

from pyspark.sql.functions import col,avg

from pyspark.sql import functions as F

from pyspark.sql.window import Window

spark = SparkSession.builder \

.appName("Banking Transactions") \

.getOrCreate()

data=[

(1,201,'Deposit',5000,'2023-09-01'),

(2,202,'Withdrawal',2000,'2023-09-01'),

(3,203,'Deposit',3000,'2023-09-02'),

(4,201,'Withdrawal',1500,'2023-09-02'),

(5,204,'Deposit',10000,'2023-09-03'),

(6,205,'Withdrawal',500,'2023-09-03'),

(7,202,'Deposit',2500,'2023-09-04'),

(8,206,'Withdrawal',700,'2023-09-04'),

(9,203,'Deposit',4000,'2023-09-05'),

(10,204,'Withdrawal',3000,'2023-09-05')

]

columns=['transaction\_id','customer\_id','transaction\_type','amount','transaction date']

df= spark.createDataFrame(data,columns)

df.show()

#Exercises:

#1. Calculate the Total Deposit and Withdrawal Amounts

total\_deposit\_withdrawal = df.groupBy('transaction\_type').agg(F.sum('amount').alias('total\_amount'))

total\_deposit\_withdrawal.show()

#2. Filter Transactions Greater Than $3,000

filter\_by\_amount = df.filter(col('amount')>3000)

filter\_by\_amount.show()

#3. Find the Largest Deposit Made

largest\_deposit = df.filter(col('transaction\_type')=='Deposit').orderBy(col('amount').desc()).limit(1)

largest\_deposit.show()

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

average\_transaction\_amount = df.groupBy('transaction\_type').agg(F.avg('amount').alias('average\_amount'))

average\_transaction\_amount.show()

#5. Find Customers Who Made Both Deposits and Withdrawals

customers\_deposits\_withdrawals = df.groupBy('customer\_id').agg(F.countDistinct('transaction\_type').alias('transaction\_count'))

customers\_deposits\_withdrawals = customers\_deposits\_withdrawals.filter(col('transaction\_count')==2)

#6. Calculate the Total Amount of Transactions per Day

total\_transactions\_per\_day = df.groupBy('transaction date').agg(F.sum('amount').alias('total\_amount'))

total\_transactions\_per\_day.show()

#7. Find the Customer with the Highest Total Withdrawal

customer\_highest\_withdrawal = df.filter(col('transaction\_type')=='Withdrawal').groupBy('customer\_id').agg(F.sum('amount').alias('total\_amount'))

customer\_highest\_withdrawal.show()

#8. Calculate the Number of Transactions for Each Customer

transactions\_per\_customer = df.groupBy('customer\_id').count()

transactions\_per\_customer.show()

#9. Find All Transactions That Occurred on the Same Day as a Withdrawal Greater

#Than $1,000

transactions\_same\_day\_withdrawal = df.filter((col('transaction\_type')=='Withdrawal') & (col('amount')>1000))

transactions\_same\_day\_withdrawal.show()

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

df = df.withColumn('transaction\_value',

F.when(col('amount')>5000,'High').otherwise('Low'))

df.show()

EXERCISE 3:

from pyspark.sql import SparkSession

from pyspark.sql.functions import col,avg

from pyspark.sql import functions as F

from pyspark.sql.window import Window

spark = SparkSession.builder \

.appName("Health & Fitness Tracker Data") \

.getOrCreate()

data=[

(1,'2023-09-01',12000,500,7.0,'Cardio'),

(2,'2023-09-01',8000,400,6.50,'Strength'),

(3,'2023-09-01',15000,650,8.0,'Yoga'),

(1,'2023-09-02',10000,450,6.0,'Cardio'),

(2,'2023-09-02',9500,500,7.0,'Cardio'),

(3,'2023-09-02',14000,600,7.50,'Strength'),

(1,'2023-09-03',13000,550,8.0,'Yoga'),

(2,'2023-09-03',12000,520,6.50,'Yoga'),

(3,'2023-09-03',16000,700,7.0,'Cardio')

]

columns=['user\_id','date','steps','calories\_burned','hours\_of\_sleep','workout\_type']

df= spark.createDataFrame(data,columns)

df.show()

#Exercises:

#1. Find the Total Steps Taken by Each User

total\_steps\_taken=df.groupBy('user\_id').agg(F.sum('steps').alias('total\_steps'))

total\_steps\_taken.show()

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

filtering\_days=df.filter(col('steps')>10000)

filtering\_days.show()

#3. Calculate the Average Calories Burned by Workout Type

average\_calories\_burned=df.groupBy('workout\_type').agg(F.avg('calories\_burned').alias('average\_calories'))

average\_calories\_burned.show()

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

window\_spec=Window.partitionBy('user\_id').orderBy(col('steps').desc())

most\_steps\_day=df.withColumn('rank',F.row\_number().over(window\_spec))

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

users\_burned\_calories=df.filter(col('calories\_burned')>600)

users\_burned\_calories.show()

#6. Calculate the Average Hours of Sleep per User

average\_hours\_sleep=df.groupBy('user\_id').agg(F.avg('hours\_of\_sleep').alias('average\_hours\_sleep'))

average\_hours\_sleep.show()

#7. Find the Total Calories Burned per Day

total\_calories\_burned=df.groupBy('date').agg(F.sum('calories\_burned').alias('total\_calories'))

total\_calories\_burned.show()

#8. Identify Users Who Did Different Types of Workouts

users\_different\_workouts=df.groupBy('user\_id').agg(F.collect\_set('workout\_type').alias('different\_workouts'))

users\_different\_workouts.show()

#9. Calculate the Total Number of Workouts per User

total\_workouts\_per\_user=df.groupBy('user\_id').count()

total\_workouts\_per\_user.show()

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

df=df.withColumn('active\_day',F.when(col('hours\_of\_sleep')>7,1).otherwise(0))

df.show()

EXERCISE 4:

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, sum as spark\_sum, avg, count, dense\_rank, to\_date, when

from pyspark.sql.window import Window

# Initialize Spark session

spark = SparkSession.builder \

.appName("Health & Fitness Tracker Data") \

.getOrCreate()

# Sample data

data = [

(1, "Blinding Lights", "The Weeknd", 200, "2023-09-01 08:15:00", "New York"),

(2, "Shape of You", "Ed Sheeran", 240, "2023-09-01 09:20:00", "Los Angeles"),

(3, "Levitating", "Dua Lipa", 180, "2023-09-01 10:30:00", "London"),

(1, "Starboy", "The Weeknd", 220, "2023-09-01 11:00:00", "New York"),

(2, "Perfect", "Ed Sheeran", 250, "2023-09-01 12:15:00", "Los Angeles"),

(3, "Don't Start Now", "Dua Lipa", 200, "2023-09-02 08:10:00", "London"),

(1, "Save Your Tears", "The Weeknd", 210, "2023-09-02 09:00:00", "New York"),

(2, "Galway Girl", "Ed Sheeran", 190, "2023-09-02 10:00:00", "Los Angeles"),

(3, "New Rules", "Dua Lipa", 230, "2023-09-02 11:00:00", "London")

]

columns = ["user\_id", "song\_title", "artist", "duration\_seconds", "streaming\_time", "location"]

# Create DataFrame

df = spark.createDataFrame(data, columns)

# 1. Total Listening Time for Each User

total\_listening\_time = df.groupBy("user\_id").agg(spark\_sum("duration\_seconds").alias("total\_listening\_time"))

total\_listening\_time.show()

# 2. Filter Songs Streamed for More Than 200 Seconds

songs\_over\_200\_seconds = df.filter(col("duration\_seconds") > 200)

songs\_over\_200\_seconds.show()

# 3. Most Popular Artist (by Total Streams)

most\_popular\_artist = df.groupBy("artist").agg(count("song\_title").alias("total\_streams")).orderBy(col("total\_streams").desc())

most\_popular\_artist.show(1)

# 4. Song with the Longest Duration

longest\_song = df.orderBy(col("duration\_seconds").desc()).select("song\_title", "duration\_seconds").limit(1)

longest\_song.show()

# 5. Average Song Duration by Artist

average\_duration\_by\_artist = df.groupBy("artist").agg(avg("duration\_seconds").alias("avg\_duration"))

average\_duration\_by\_artist.show()

# 6. Top 3 Most Streamed Songs per User

window\_spec = Window.partitionBy("user\_id").orderBy(col("song\_title"))

df\_with\_rank = df.withColumn("rank", dense\_rank().over(window\_spec))

top\_3\_songs\_per\_user = df\_with\_rank.filter(col("rank") <= 3)

top\_3\_songs\_per\_user.show()

# 7. Total Number of Streams per Day

total\_streams\_per\_day = df.withColumn("streaming\_date", to\_date(col("streaming\_time"))) \

.groupBy("streaming\_date").agg(count("song\_title").alias("total\_streams"))

total\_streams\_per\_day.show()

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

multiple\_artist\_users = df.groupBy("user\_id").agg(count("artist").alias("artist\_count")).filter(col("artist\_count") > 1)

multiple\_artist\_users.show()

# 9. Total Streams for Each Location

total\_streams\_by\_location = df.groupBy("location").agg(count("song\_title").alias("total\_streams"))

total\_streams\_by\_location.show()

# 10. Create a New Column to Classify Long and Short Songs

df\_with\_length\_classification = df.withColumn("song\_length", when(col("duration\_seconds") > 200, "Long").otherwise("Short"))

df\_with\_length\_classification.show()

EXERCISE 5:

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, sum as spark\_sum, avg, count, dense\_rank, to\_date, when

from pyspark.sql.window import Window

# Initialize Spark session

spark = SparkSession.builder \

.appName("Retail Store Sales Data") \

.getOrCreate()

data=[

(1,'Apple','Groceries',0.50,10,'2023-09-01'),

(2,'T-shirt','Clothing',15.00,2,'2023-09-01'),

(3,'Notebook','Stationery',2.00,5,'2023-09-02'),

(4,'Banana','Groceries',0.30,12,'2023-09-02'),

(5,'Laptop','Electronics',800.00,1,'2023-09-03'),

(6,'Pants','Clothing',25.00,3,'2023-09-03'),

(7,'Headphones','Electronics',100.00,2,'2023-09-04'),

(8,'Pen','Stationery',1.00,10,'2023-09-04'),

(9,'Orange','Groceries',0.60,8,'2023-09-05'),

(10,'Sneakers','Clothing',50.00,1,'2023-09-05'),

]

columns=['product\_id','product\_name','category','price','quantity','sale\_date']

df= spark.createDataFrame(data,columns)

df.show()

#Exercises:

#1. Calculate the Total Revenue per Category

total\_revenue\_per\_category=df.groupBy('category').agg(spark\_sum('price').alias('total\_revenue'))

total\_revenue\_per\_category.show()

#2. Filter Transactions Where the Total Sales Amount is Greater Than $100

filter\_by\_total\_sales=df.groupBy('product\_id').agg(spark\_sum('price').alias('total\_sales'))

filter\_by\_total\_sales=filter\_by\_total\_sales.filter(col('total\_sales')>100)

filter\_by\_total\_sales.show()

#3. Find the Most Sold Product

most\_sold\_product=df.groupBy('product\_name').agg(spark\_sum('quantity').alias('total\_quantity'))

most\_sold\_product=most\_sold\_product.orderBy(col('total\_quantity').desc()).limit(1)

most\_sold\_product.show()

#4. Calculate the Average Price per Product Category

average\_price\_per\_category=df.groupBy('category').agg(avg('price').alias('average\_price'))

average\_price\_per\_category.show()

#5. Find the Top 3 Highest Grossing Products

top\_3\_highest\_grossing=df.groupBy('product\_name').agg(spark\_sum('price').alias('total\_sales'))

top\_3\_highest\_grossing=top\_3\_highest\_grossing.orderBy(col('total\_sales').desc()).limit(3)

top\_3\_highest\_grossing.show()

#6. Calculate the Total Number of Items Sold per Day

total\_items\_sold\_per\_day=df.groupBy('sale\_date').agg(spark\_sum('quantity').alias('total\_items\_sold'))

total\_items\_sold\_per\_day.show()

#7. Identify the Product with the Lowest Price in Each Category

window\_spec=Window.partitionBy('category').orderBy(col('price'))

product\_lowest\_price=df.withColumn('rank',dense\_rank().over(window\_spec))

product\_lowest\_price=product\_lowest\_price.filter(col('rank')==1)

product\_lowest\_price.show()

#8. Calculate the Total Revenue for Each Product

total\_revenue\_per\_product=df.groupBy('product\_name').agg(spark\_sum('price').alias('total\_revenue'))

total\_revenue\_per\_product.show()

#9. Find the Total Sales per Day for Each Category

total\_sales\_per\_day\_category=df.groupBy('sale\_date','category').agg(spark\_sum('price').alias('total\_sales'))

total\_sales\_per\_day\_category.show()

#10. Create a New Column for Discounted Price

df=df.withColumn('discounted\_price',when(col('quantity')>5,col('price')\*0.9).otherwise(col('price')))

df.show()