PYSPARK ASSIGNMENT

1. Dataset: E-commerce Transactions

```
# 1. Dataset: E-commerce Transactions
from pyspark.sql import SparkSession
import pyspark.sql.functions as F

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","tr
ansaction date"]

# Create a SparkSession
spark = SparkSession.builder.appName("EcommerceAnalysis").getOrCreate()
df= spark.createDataFrame(data,columns)
# Load the data as a DataFrame
# df = spark.read.csv('e commerce transactions.csv', header=True, inferSchema=True)
df.show()

# 1. Calculate the Total Revenue per Category
df = df.withColumn('final price', F.col('price') - (F.col('price') * F.col('discount percentage')
/ 100))
total_revenue_per_category =
df.groupby('category').agg(F.sum('final_price').alias('total_revenue'))
print("Total Revenue per Category:\n", total_revenue_per_category.show())

# 2. Filter Transactions with a Discount Greater Than 10%
discounted_transactions = df.filter(df['discount_percentage'] > 10)
print("Transactions with a Discount Greater Than 10%:\n", discounted_transactions.show())

# 3. Find the Most Expensive Product Sold
most_expensive_product = df.orderBy(F.desc('price')).limit(1)
print("Most Expensive Product Sold:\n", 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'))
print("Average Quantity of Products Sold per Category:\n", average_quantity_per_category.show())

# 5. Identify Customers Who Purchased More Than One Product
customers multiple products=df.filter(df["quantity"]>1)
print("Customers Who Purchased More Than One Product:\n", customers_multiple_products.show())

# 6. Find the Top 3 Highest Revenue Transactions
df = df.withColumn('total_sales', F.col('final_price') * F.col('quantity'))
```

```
top_revenue_transactions = df.orderBy(F.desc('total_sales')).limit(3)
print("Top 3 Highest Revenue Transactions:\n", top_revenue_transactions.show())

# 7. Calculate the Total Number of Transactions per Day
total_transactions_per_day = df.groupby('transaction_date').count()
print("Total Number of Transactions per Day:\n", total_transactions_per_day.show())

# 8. Find the Customer Who Spent the Most Money
customer spending = df.groupby('customer id').agg(F.sum('final price').alias('total spent'))
most_spending_customer = customer_spending.orderBy(F.desc('total_spent')).limit(1)
print("Customer Who Spent the Most Money:\n", most_spending_customer.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'))
print("Average Discount Given per Product Category:\n", average_discount_per_category.show())

# 10. Create a New Column for Final Price After Discount
# Already created in step 1
print("Data with Final Price After Discount column:\n", df.show())
```

2. Dataset: Banking Transactions

```
# 2.Dataset: Banking Transactions

from pyspark.sql import SparkSession
import pyspark.sql.functions as F

transactions = [
    (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"]

# Create a SparkSession
spark = SparkSession.builder.appName("BankingTransactionsAnalysis").getOrCreate()

df = spark.createDataFrame(transactions, columns)
df.show()



# 1. Calculate the Total Deposit and Withdrawal Amounts
total_amounts = df.groupby('transaction_type').agg(F.sum('amount').alias('total_amount'))
print("Total Amounts:\n", total_amounts.show())

# 2. Filter Transactions Greater Than $3,000
filtered transactions = df.filter(df['amount'] > 3000)
print("Transactions Greater Than $3,000:\n", filtered_transactions.show())

# 3. Find the Largest Deposit Made
largest_deposit = df.filter(df['transaction_type'] ==
'Deposit').orderBy(F.desc('amount')).limit(1)
```

```
print("Largest Deposit:\n", largest_deposit.show())

# 4. Calculate the Average Transaction Amount for Each Transaction Type
average amounts = df.groupby('transaction type').agg(F.avg('amount').alias('average amount'))
print("Average Amounts:\n", average_amounts.show())

# 5. Find Customers Who Made Both Deposits and Withdrawals
customers both =
df.groupBy('customer id').agg(F.collect list('transaction type').alias('transaction types')).filt
er(F.array_contains(F.col('transaction_types'), 'Deposit') &
F.array_contains(F.col('transaction_types'), 'Withdrawal'))
print("Customers Who Made Both Deposits and Withdrawals:\n", customers_both.show())

# 6. Calculate the Total Amount of Transactions per Day
total_amounts_per_day = df.groupby('transaction_date').agg(F.sum('amount').alias('total_amount'))
print("Total Amounts per Day:\n", total_amounts_per_day.show())

# 7. Find the Customer with the Highest Total Withdrawal
highest_withdrawal_customer = df.filter(df['transaction_type'] ==
'Withdrawal').groupby('customer_id').agg(F.sum('amount').alias('total_withdrawal')).orderBy(F.des
c('total_withdrawal')).limit(1)
print("Customer with the Highest Total Withdrawal:\n", highest_withdrawal_customer.show())

# 8. Calculate the Number of Transactions for Each Customer
transaction_count_per_customer = df.groupby('customer_id').count()
print("Transaction Count per Customer:\n", transaction_count_per_customer.show())

# 9. Find All Transactions That Occurred on the Same Day as a Withdrawal Greater Than $1,000
withdrawal_threshold = df.filter((df['transaction_type'] == 'Withdrawal') & (df['amount'] >
1000)).select('transaction_date').distinct()
filtered_transactions = df.join(withdrawal_threshold, 'transaction_date', 'inner')
print("Filtered Transactions:\n", filtered_transactions.show())

# 10. Create a New Column to Classify Transactions as "High" or "Low" Value
df = df.withColumn('transaction_value', F.when(df['amount'] > 5000, 'High').otherwise('Low'))
print("Data with Transaction Value Column:\n", df.show())
```

3. Dataset: Health & Fitness Tracker Data

```
# 3.Dataset: Health & Fitness Tracker Data
from pyspark.sql import SparkSession
import pyspark.sql.functions as F

data = [
    (1, '2023-09-01', 12000, 500, 7.0, 'Cardio'),
    (2, '2023-09-01', 8000, 400, 6.5, '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, 7.5, 'Strength'),
    (1, '2023-09-03', 13000, 550, 8.0, 'Yoga'),
    (2, '2023-09-03', 12000, 520, 6.5, 'Yoga'),
    (3, '2023-09-03', 16000, 700, 7.0, 'Cardio')
]

columns = ['user_id', 'date', 'steps', 'calories_burned', 'hours_of_sleep', 'workout_type']

# Create a SparkSession
spark = SparkSession.builder.appName("HealthFitnessAnalysis").getOrCreate()
```

```
df = spark.createDataFrame(data, columns)
df.show()

# 1. Find the Total Steps Taken by Each User
total_steps_per_user = df.groupby('user_id').agg(F.sum('steps').alias('total_steps'))
print("Total Steps Taken by Each User:\n", total_steps_per_user.show())

# 2. Filter Days with More Than 10,000 Steps
filtered data = df.filter(df['steps'] > 10000)
print("Days with More Than 10,000 Steps:\n", filtered_data.show())

# 3. Calculate the Average Calories Burned by Workout Type
average calories per workout =
df.groupby('workout type').agg(F.avg('calories burned').alias('average calories'))
print("Average Calories Burned by Workout Type:\n", average_calories_per_workout.show())

# 4. Identify the Day with the Most Steps for Each User
max steps per user = df.groupby('user id').agg(F.max('steps').alias('max steps'))
print("Day with the Most Steps for Each User:\n", max_steps_per_user.show())

# 5. Find Users Who Burned More Than 600 Calories on Any Day
users_high_calories = df.filter(df['calories_burned'] > 600).select('user_id').distinct()
print("Users Who Burned More Than 600 Calories on Any Day:\n", users high calories.show())

# 6. Calculate the Average Hours of Sleep per User
average_sleep_per_user =
df.groupby('user_id').agg(F.avg('hours_of_sleep').alias('average_sleep'))
print("Average Hours of Sleep per User:\n", average sleep per user.show())

# 7. Find the Total Calories Burned per Day
total_calories_per_day = df.groupby('date').agg(F.sum('calories_burned').alias('total_calories'))
print("Total Calories Burned per Day:\n", total_calories_per_day.show())

# 8. Identify Users Who Did Different Types of Workouts
users_multiple_workouts =
df.groupBy('user_id').agg(F.collect_set('workout_type').alias('workout_types')).filter(F.size(F.c
ol('workout types')) > 1)
print("Users Who Did Different Types of Workouts:\n", users multiple workouts.show())

# 9. Calculate the Total Number of Workouts per User
workout_count_per_user = df.groupby('user_id').count()
print("Total Number of Workouts per User:\n", workout count per user.show())

# 10. Create a New Column for "Active" Days
df = df.withColumn('active_day', F.when(df['steps'] > 10000, 'Active').otherwise('Inactive'))
print("Data with Active Day Column:\n", df.show())
```

4. Dataset: Music Streaming Data

```
# 4.Dataset: Music Streaming Data
from pyspark.sql import SparkSession
import pyspark.sql.functions as F

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 a SparkSession
spark = SparkSession.builder.appName("MusicStreamingAnalysis").getOrCreate()

df = spark.createDataFrame(data, columns)
df.show()


# 1. Calculate the Total Listening Time for Each User
total listening time per user =
df.groupby('user_id').agg(F.sum('duration_seconds').alias('total_listening_time'))
print("Total Listening Time per User:\n", total_listening_time_per_user.show())

# 2. Filter Songs Streamed for More Than 200 Seconds
filtered_songs = df.filter(df['duration_seconds'] > 200)
print("Songs Streamed for More Than 200 Seconds:\n", filtered_songs.show())

# 3. Find the Most Popular Artist (by Total Streams)
most popular artist = df.groupby('artist').count().orderBy(F.desc('count')).first()
print("Most Popular Artist:\n", most_popular_artist)

# 4. Identify the Song with the Longest Duration
longest_song = df.orderBy(F.desc('duration_seconds')).first()
print("Song with the Longest Duration:\n", longest song)

# 5. Calculate the Average Song Duration by Artist
average_duration_by_artist =
df.groupby('artist').agg(F.avg('duration_seconds').alias('average_duration'))
print("Average Song Duration by Artist:\n", average duration by artist.show())

# 6. Find the Top 3 Most Streamed Songs per User
# top_streamed_songs =
df.groupby('user id').agg(F.collect list('song title').alias('streamed songs')).select('user id',
F.array slice(F.col('streamed songs'), 0, 3).alias('top 3 songs'))
# print("Top 3 Most Streamed Songs per User:\n", top_streamed_songs.show())

# 7. Calculate the Total Number of Streams per Day
df = df.withColumn('streaming date', F.to date(df['streaming time']))
total streams per day = df.groupby('streaming date').count()
print("Total Number of Streams per Day:\n", total_streams_per_day.show())

# 8. Identify Users Who Streamed Songs from More Than One Artist
users multiple artists =
df.groupBy('user id').agg(F.collect set('artist').alias('artists')).filter(F.size(F.col('artists'
)) > 1)
print("Users Who Streamed Songs from More Than One Artist:\n", users_multiple_artists.show())

# 9. Calculate the Total Streams for Each Location
total streams per location = df.groupby('location').count()
print("Total Streams for Each Location:\n", total_streams_per_location.show())

# 10. Create a New Column to Classify Long and Short Songs
df = df.withColumn('song length', F.when(df['duration seconds'] > 200,
'Long').otherwise('Short'))
print("Data with Song Length Column:\n", df.show())
```

5. Dataset: Retail Store Sales Data

```
# 5.Dataset: Retail Store Sales Data
from pyspark.sql import SparkSession
import pyspark.sql.functions as F

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')
]
# Create a SparkSession
spark = SparkSession.builder.appName("RetailSalesAnalysis").getOrCreate()

columns = ['product_id', 'product_name', 'category', 'price', 'quantity', 'sales_date']

df = spark.createDataFrame(data, columns)
df.show()

# 1. Calculate the Total Revenue per Category
df = df.withColumn('total sales', F.col('price') * F.col('quantity'))
total_revenue_per_category =
df.groupby('category').agg(F.sum('total_sales').alias('total_revenue'))
print("Total Revenue per Category:\n", total_revenue_per_category.show())

# 2. Filter Transactions Where the Total Sales Amount is Greater Than $100
filtered_transactions = df.filter(df['total_sales'] > 100)
print("Transactions Greater Than $100:\n", filtered_transactions.show())

# 3. Find the Most Sold Product
most sold product =
df.groupby('product_name').agg(F.sum('quantity').alias('total_quantity')).orderBy(F.desc('total_q
uantity')).first()
print("Most Sold Product:\n", most_sold_product)

# 4. Calculate the Average Price per Product Category
average_price_per_category = df.groupby('category').agg(F.avg('price').alias('average_price'))
print("Average Price per Product Category:\n", average_price_per_category.show())

# 5. Find the Top 3 Highest Grossing Products
top grossing products = df.orderBy(F.desc('total sales')).limit(3)
print("Top 3 Highest Grossing Products:\n", top_grossing_products.show())

# 6. Calculate the Total Number of Items Sold per Day
total items sold per day =
df.groupby('sales_date').agg(F.sum('quantity').alias('total_items_sold'))
print("Total Number of Items Sold per Day:\n", total_items_sold_per_day.show())

# 7. Identify the Product with the Lowest Price in Each Category
lowest price product = df.groupby('category').agg(F.min('price').alias('lowest price'))
print("Lowest Price Product in Each Category:\n", lowest_price_product.show())

# 8. Calculate the Total Revenue for Each Product
total_revenue_per_product =
df.groupby('product_name').agg(F.sum('total_sales').alias('total_revenue'))
```

```
print("Total Revenue for Each Product:\n", total_revenue_per_product.show())

# 9. Find the Total Sales per Day for Each Category
total sales per day category = df.groupby(['sales date',
'category']).agg(F.sum('total_sales').alias('total_sales'))
print("Total Sales per Day for Each Category:\n", total_sales_per_day_category.show())

# 10. Create a New Column for Discounted Price
df = df.withColumn('discounted price', F.col('price') * 0.9)
print("Data with Discounted Price Column:\n", df.show())
```