Assignment-3

Pyspark

#Fitness Tracker

#Exercises:

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

spark = SparkSession.builder.appName("FitnessTracker").getOrCreate()

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

#1. Find the Total Steps Taken by Each User
total_steps_df = df.groupBy("user_id").agg(sum("steps").alias("total_steps"))
print("Total Steps by Each User:")
total_steps_df.show()

#2. Filter Days Where a User Burned More Than 500 Calories
filtered_days_df = df.filter(df.calories > 500)
print("Days with More than 500 Calories Burned:")
filtered_days_df.show()

#3. Calculate the Average Distance Traveled by Each User
average_distance_df =
df.groupBy("user_id").agg(avg("distance_km").alias("average_distance"))
print("Average Distance Traveled by Each User:")
average_distance_df.show()
```

```
#4. Identify the Day with the Maximum Steps for Each User
max_steps_per_user = df.groupBy("user_id", "date").agg(max("steps").alias("max_steps"))
print("Day with maximum steps: ")
max_steps_per_user.show()

#5. Find Users Who Were Active for More Than 100 Minutes on Any Day
active_users_df = df.filter(df.active_minutes > 100).select("user_id").distinct()
print("Users active for more than 100 minutes on any day:")
active_users_df.show()

#6. Calculate the Total Calories Burned per Day
# Total calories burned per day
total_calories_df = df.groupBy("date").agg(sum("calories").alias("total_calories"))
print("Total calories burned per day:")
total_calories_df.show()

#7. Calculate the Average Steps per Day
average_steps_df = df.groupBy("date").agg(avg("steps").alias("average_steps"))
print("Average steps per day:")
average_steps_df.show()

#8. Rank Users by Total Distance Traveled
from pyspark.sql.functions import rank
from pyspark.sql.window import Window

total_distance_df = df.groupBy("user_id").agg(sum("distance_km").alias("total_distance"))
window = Window.orderBy(total_distance_df["total_distance"].desc())
ranked_users_df = total_distance_df.withColumn("rank", rank().over(window))
print("Users ranked by total distance traveled:")
ranked_users_df.show()
```

#9. Find the Most Active User by Total Active Minutes

```
most_active_user_df = df.groupBy("user_id").agg(sum("active_minutes").alias("total_active_minutes")).orderBy("total_active_minutes", ascending=False).limit(1)

print("Most active user by total active minutes:")

most_active_user_df.show()
```

#10. Create a New Column for Calories Burned per Kilometer

```
calories_per_km_df = df.withColumn("calories_per_km", (col("calories") / col("distance_km")))

print("New column for calories burned per kilometer:")

calories_per_km_df.show()
```

#Book Sales

#Exercises:

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

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

#1. Find Total Sales Revenue per Genre
total_revenue_df = df.withColumn("total_sales", col("sale_price") * col("quantity")) \
.groupBy("genre").agg(sum("total_sales").alias("total_revenue"))
print("Total sales revenue per genre:")
total_revenue_df.show()

#2. Filter Books Sold in the "Fiction" Genre
fiction_books_df = df.filter(col("genre") == "Fiction")
print("Books sold in the 'Fiction' genre:")
fiction_books_df.show()

#3. Find the Book with the Highest Sale Price
highest_price_book_df = df.orderBy(col("sale_price").desc()).limit(1)
print("Book with the highest sale price:")
highest_price_book_df.show()

#4. Calculate Total Quantity of Books Sold by Author
total_quantity_by_author_df =
df.groupBy("author").agg(sum("quantity").alias("total_quantity"))
print("Total quantity of books sold by author:")
total_quantity_by_author_df.show()
```

```
#5. Identify Sales Transactions Worth More Than $50
expensive_transactions_df = df.filter(col("sale_price") > 50)
print("Sales transactions worth more than $50:")
expensive_transactions_df.show()

#6. Find the Average Sale Price per Genre
avg_sale_price_df = df.groupBy("genre").agg(avg("sale_price").alias("avg_sale_price"))
print("Average sale price per genre:")
avg_sale_price_df.show()

#7. Count the Number of Unique Authors in the Dataset
unique_authors_count = df.select("author").distinct().count()
print(f"Number of unique authors: {unique_authors_count}")

#8. Find the Top 3 Best-Selling Books by Quantity
top_3_best_selling_df = df.orderBy(col("quantity").desc()).limit(3)
print("Top 3 best-selling books by quantity:")
top_3_best_selling_df.show()

#9. Calculate Total Sales for Each Month
df = df.withColumn("month", month(col("date")))
total_sales_per_month_df = df.groupBy("month").agg(sum("sale_price").alias("total_sales"))
print("Total sales for each month:")
total_sales_per_month_df.show()

#10. Create a New Column for Total Sales Amount
total_sales_amount_df = df.withColumn("total_sales_amount", col("sale_price") *
col("quantity"))
print("New column for total sales amount:")
total_sales_amount_df.show()
```

#Food Delivery Orders

#Exercises:

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

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

#1. Calculate Total Revenue per Restaurant
total_revenue_df = df.withColumn("total_revenue", col("price") * col("quantity")) \
.groupBy("restaurant_name").agg(sum("total_revenue").alias("total_revenue"))
print("Total Revenue:")
total_revenue_df.show()

#2. Find the Fastest Delivery
fastest_delivery_df = df.orderBy(col("delivery_time_mins").asc()).limit(1)
print("Fastest Delivery:")
fastest_delivery_df.show()

#3. Calculate Average Delivery Time per Restaurant
avg_delivery_time_df =
df.groupBy("restaurant_name").agg(avg("delivery_time_mins").alias("avg_delivery_time"))
print("Average Delivery Time:")
avg_delivery_time_df.show()

#4. Filter Orders for a Specific Customer
customer_orders_df = df.filter(col("customer_id") == 201)
print("Orders for a specific customer:")
customer_orders_df.show()
```

```
#5. Find Orders Where Total Amount Spent is Greater Than $20
high_spending_orders_df = df.withColumn("total_amount", col("price") * col("quantity")) \
.filter(col("total_amount") > 20)
print("Orders with total amount greater than $20:")
high_spending_orders_df.show()

#6. Calculate the Total Quantity of Each Food Item Sold
total_quantity_df = df.groupBy("food_item").agg(sum("quantity").alias("total_quantity"))
print("Total quantity of each food item sold:")
total_quantity_df.show()

#7. Find the Top 3 Most Popular Restaurants by Number of Orders
top_3_popular_restaurants_df =
df.groupBy("restaurant_name").count().orderBy(col("count").desc()).limit(3)
print("Top 3 most popular restaurants by number of orders:")
top_3_popular_restaurants_df.show()

#8. Calculate Total Revenue per Day
total_revenue_per_day_df = df.withColumn("total_revenue", col("price") * col("quantity")) \
.groupBy("order_d").agg(sum("total_revenue").alias("total_revenue"))
print("Total Revenue per Day:")
total_revenue_per_day_df.show()

#9. Find the Longest Delivery Time for Each Restaurant
longest_delivery_time_df =
df.groupBy("restaurant_name").agg(max("delivery_time_mins").alias("longest_delivery_tim
e"))
print("Longest Delivery Time for Each Restaurant:")
longest_delivery_time_df.show()
```

#10. Create a New Column for Total Order Value

total_order_value_df = df.withColumn("total_order_value", col("price") * col("quantity"))

print("New column for total order value:")

total_order_value_df.show()

 # Weather Data

#Exercises:

from pyspark.sql.functions import sum, col, avg ,month,pow,max

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

#1. Find the Average Temperature for Each City

avg_temp_df = df.groupBy("city").agg(avg("temperature_c").alias("avg_temperature"))

print("Average Temperature:")

avg_temp_df.show()

#2. Filter Days with Temperature Below Freezing

freezing_days_df = df.filter(col("temperature_c") < 0)

print("Days with temperature below freezing:")

freezing_days_df.show()

#3. Find the City with the Highest Wind Speed on a Specific Day

specific_day_df = df.filter(col("date") == "2023-01-02")

city_with_highest_wind_df = specific_day_df.orderBy(col("wind_speed_kph").desc()).limit(1)

print("City with the highest wind speed on a specific day:")

city_with_highest_wind_df.show()

```
#4. Calculate the Total Number of Days with Rainy Weather
rainy_days_count = df.filter(col("condition") == "Rain").count()
print(f"Total number of days with rainy weather: {rainy_days_count}")

#5. Calculate the Average Humidity for Each Weather Condition
avg_humidity_df = df.groupBy("condition").agg(avg("humidity").alias("avg_humidity"))
print("Average humidity for each weather condition:")
avg_humidity_df.show()

#6. Find the Hottest Day in Each City
hottest_days_df = df.groupBy("city").agg(max("temperature_c").alias("hottest_day"))
print("Hottest day in each city:")
hottest_days_df.show()

#7. Identify Cities That Experienced Snow
snowy_cities_df = df.filter(col("condition") == "Snow").select("city").distinct()
print("Cities that experienced snow:")
snowy_cities_df.show()

#8. Calculate the Average Wind Speed for Days When the Condition was Sunny
sunny_days_avg_wind_speed_df = df.filter(col("condition") ==
"Sunny").agg(avg("wind_speed_kph").alias("avg_wind_speed"))
print("Average wind speed for days when the condition was sunny:")
sunny_days_avg_wind_speed_df.show()

#9. Find the Coldest Day Across All Cities
coldest_day_df = df.orderBy(col("temperature_c").asc()).limit(1)
print("Coldest day across all cities:")
coldest_day_df.show()
```

```
#10. Create a New Column for Wind Chill
df_with_wind_chill = df.withColumn("wind_chill",
13.12 + 0.6215 * col("temperature_c") - 11.37 * pow(col("wind_speed_kph"), 0.16) +
0.3965 * col("temperature_c") * pow(col("wind_speed_kph"), 0.16))
print("New column for wind chill:")
df_with_wind_chill.show()
```

#Airline Flight

#Exercises:

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

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

#1. Find the Total Distance Traveled by Each Airline
total_distance_df = df.groupBy("airline").agg(sum("distance").alias("total_distance"))
print("Total distance traveled by each airline:")
total_distance_df.show()

#2. Filter Flights with Delays Greater than 30 Minutes
delayed_flights_df = df.filter(col("delay_min") > 30)
print("Flights with delays greater than 30 minutes:")
delayed_flights_df.show()

#3. Find the Flight with the Longest Distance
longest_flight_df = df.orderBy(col("distance").desc()).limit(1)
print("Flight with the longest distance:")
longest_flight_df.show()
```

```
#4. Calculate the Average Delay Time for Each Airline
avg_delay_time_df = df.groupBy("airline").agg(avg("delay_min").alias("avg_delay_time"))
print("Average delay time for each airline:")
avg_delay_time_df.show()

#5. Identify Flights That Were Not Delayed
not_delayed_flights_df = df.filter(col("delay_min") == 0)
print("Flights that were not delayed:")
not_delayed_flights_df.show()

#6. Find the Top 3 Most Frequent Routes
top_3_frequent_routes_df = df.groupBy("origin",
"destination").count().orderBy(col("count").desc()).limit(3)
print("Top 3 most frequent routes:")
top_3_frequent_routes_df.show()

#7. Calculate the Total Number of Flights per Day
flights_per_day_df = df.groupBy("date").count()
print("Total number of flights per day:")
flights_per_day_df.show()

#8. Find the Airline with the Most Flights
most_flights_airline_df = df.groupBy("airline").count().orderBy(col("count").desc()).limit(1)
print("Airline with the most flights:")
most_flights_airline_df.show()

#9. Calculate the Average Flight Distance per Day
avg_distance_per_day_df = df.groupBy("date").agg(avg("distance").alias("avg_distance"))
print("Average flight distance per day:")
avg_distance_per_day_df.show()
```

```
#10. Create a New Column for On-Time Status
df_with_on_time_status = df.withColumn("on_time_status",
when(col("delay_min") <= 0, "On Time").otherwise("Delayed"))
print("New column for on-time status:")
df_with_on_time_status.show()
```