#1 Fitness tracker

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("Fitness Tracker") \

.getOrCreate()

data={

(1,'2023-07-01',12000,500,8.5,9),

(2,'2023-07-01',8000,350,5.6,60),

(3,'2023-07-01',15000,600,10.2,120),

(1,'2023-07-02',11000,480,7.9,85),

(2,'2023-07-02',9000,400,6.2,70),

(3,'2023-07-02',13000,520,9.0,100),

(1,'2023-07-03',10000,450,7.1,80),

(2,'2023-07-03',7000,320,4.9,55),

(3,'2023-07-03',16000,620,11.0,130)

}

columns=['user\_id','date','steps','calories','distance\_km','active\_minutes']

df= spark.createDataFrame(data,columns)

df.show()

# 1. Find the Total Steps Taken by Each User

# Calculate the total number of steps taken by each user across all days.

total\_steps\_by\_user = df.groupBy('user\_id').sum('steps').withColumnRenamed('sum(steps)','total steps')

total\_steps\_by\_user.show()

# 2. Filter Days Where a User Burned More Than 500 Calories

# Identify all days where a user burned more than 500 calories.

filter\_by\_calories = df.filter(col('calories')>500)

filter\_by\_calories.show()

# 3. Calculate the Average Distance Traveled by Each User

# Calculate the average distance traveled ( distance\_km ) by each user

# across all days.

avg\_distance = df.groupBy('user\_id').agg(F.avg(col('distance\_km'))).withColumnRenamed('avg(distance\_km)','distance in km')

avg\_distance.show()

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

# For each user, find the day when they took the maximum number of steps.

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

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

max\_step\_in\_a\_day = ranking.filter(ranking['rank']==1)

max\_step\_in\_a\_day.show()

# 5. Find Users Who Were Active for More Than 100 Minutes on Any Day

# Identify users who had active minutes greater than 100 on any day.

active\_mins = df.filter(col('active\_minutes')>100)

active\_mins.show()

# 6. Calculate the Total Calories Burned per Day

# Group by date and calculate the total number of calories burned by all

# users combined for each day.

total\_cal\_burnt = df.groupBy('date').sum('calories').withColumnRenamed('sum(calories)','total calories')

total\_cal\_burnt.show()

# 7. Calculate the Average Steps per Day

# Find the average number of steps taken across all users for each day.

avg\_steps = df.groupBy('date').agg(F.avg('steps')).withColumnRenamed('avg(steps)','avg steps by date')

avg\_steps.show()

# 8. Rank Users by Total Distance Traveled

# Rank the users by their total distance traveled, from highest to lowest.

distance\_km = df.groupBy('user\_id').sum('distance\_km').withColumnRenamed('sum(distance\_km)','total km')

window\_specs=Window.orderBy(F.desc('total km'))

ranking\_users = distance\_km.withColumn('ranking',F.rank().over(window\_specs))

ranking\_users.show()

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

# Identify the user with the highest total active minutes across all days.

highest\_active\_user = df.orderBy(F.desc('active\_minutes')).limit(1)

highest\_active\_user.show()

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

# Add a new column called calories\_per\_km that calculates how many

# calories were burned per kilometer ( calories / distance\_km ) for each

# row.

new\_col=df.withColumn('calories\_per\_km',col('calories')/col('distance\_km'))

new\_col.show()

#2 Dataset: Book Sales Data

from pyspark.sql import SparkSession

from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DoubleType, DateType

from pyspark.sql.functions import col,avg

from pyspark.sql import functions as F

from pyspark.sql.window import Window

spark = SparkSession.builder \

.appName("Book sales") \

.getOrCreate()

# Sample Data:

data1=[

(1,'The Catcher in the Rye','J.D. Salinger','Fiction',15.99,2,'2023-01-05'),

(2,'To Kill a Mockingbird','Harper Lee','Fiction',18.99,1,'2023-01-10'),

(3,'Becoming','Michelle Obama','Biography',20.00,3,'2023-02-12'),

(4,'Sapiens','Yuval Noah Harari','Non-Fiction',22.50,1,'2023-02-15'),

(5,'Educated','Tara Westover','Biography',17.99,2,'2023-03-10'),

(6,'The Great Gatsby','F. Scott Fitzgerald','Fiction',10.99,5,'2023-03-15'),

(7,'Atomic Habits','James Clear','Self-Help',16.99,3,'2023-04-01'),

(8,'Dune','Frank Herbert','Science Fiction',25.99,1,'2023-04-10'),

(9,'1984','George Orwell','Fiction',14.99,2,'2023-04-12'),

(10,'The Power of Habit','Charles Duhigg','Self-Help',18.00,1,'2023-05-01')

]

schema = StructType([

StructField("sale\_id", IntegerType(), True),

StructField("book\_title", StringType(), True),

StructField("author", StringType(), True),

StructField("genre", StringType(), True),

StructField("sale\_price", DoubleType(), True),

StructField("quantity", IntegerType(), True),

StructField("date", StringType(), True)

])

df1= spark.createDataFrame(data1,schema=schema)

df1.show()

# Exercises:

# 1. Find Total Sales Revenue per Genre

# Group the data by genre and calculate the total sales revenue for each

# genre. (Hint: Multiply sale\_price by quantity to get total sales for

# each book.)

df1 = df1.withColumn('total sales',F.col('sale\_price') \* F.col('quantity'))

group\_by\_genre=df1.groupBy('genre').agg(F.sum('total sales').alias('Total sales'))

group\_by\_genre.show()

# 2. Filter Books Sold in the "Fiction" Genre

# Filter the dataset to include only books sold in the "Fiction" genre.

filter\_book = df1.filter(df1['genre']=='Fiction')

filter\_book.show()

# 3. Find the Book with the Highest Sale Price

# Identify the book with the highest individual sale price.

high\_sale = df1.orderBy(F.desc('sale\_price')).limit(1)

high\_sale.show()

# 4. Calculate Total Quantity of Books Sold by Author

# Group the data by author and calculate the total quantity of books sold

# for each author.

tot\_quantity = df1.groupBy('author').sum('quantity').alias('Quantity')

tot\_quantity.show()

# 5. Identify Sales Transactions Worth More Than $50

# Filter the sales transactions where the total sales amount (sale\_price \*

# quantity) is greater than $50.

sale\_transaction = df1.withColumn('total sales', df1['sale\_price']\* df1['quantity']).filter(F.col('total sales')>50)

sale\_transaction.show()

# 6. Find the Average Sale Price per Genre

# Group the data by genre and calculate the average sale price for books

# in each genre.

group\_genre = df1.groupBy('genre').agg(F.avg('sale\_price'))

group\_genre.show()

# 7. Count the Number of Unique Authors in the Dataset

# Count how many unique authors are present in the dataset.

print(df1.select('author').distinct().count())

# 8. Find the Top 3 Best-Selling Books by Quantity

# Identify the top 3 best-selling books based on the total quantity sold.

top\_3 = df1.orderBy(F.desc('quantity')).limit(3)

top\_3.show()

# 9. Calculate Total Sales for Each Month

# Group the sales data by month and calculate the total sales revenue for

# each month.

group\_by\_month = df1.withColumn('group by month', F.month('date'))

total\_sales\_by\_month = group\_by\_month.withColumn("total\_sales", df1["sale\_price"] \* df1["quantity"]).groupBy("group by month").agg(F.sum("total\_sales").alias("monthly\_sales"))

total\_sales\_by\_month.show()

# 10. Create a New Column for Total Sales Amount

# Add a new column total\_sales that calculates the total sales amount for

# each transaction ( sale\_price \* quantity ).

new\_col = df1.withColumn('total\_sales',F.col('sale\_price')\* F.col('quantity'))

new\_col.show()

# 3. Dataset: Food Delivery Orders

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

data2 = [

(1, 201, 'McDonalds', 'Burger', 2, 5.99, 30, '2023-06-15'),

(2, 202, 'Pizza Hut', 'Pizza', 1, 12.99, 45, '2023-06-16'),

(3, 203, 'KFC', 'Fried Chicken', 3, 8.99, 25, '2023-06-17'),

(4, 201, 'Subway', 'Sandwich', 2, 6.50, 20, '2023-06-17'),

(5, 204, 'Dominos', 'Pizza', 2, 11.99, 40, '2023-06-18'),

(6, 205, 'Starbucks', 'Coffee', 1, 4.50, 15, '2023-06-18'),

(7, 202, 'KFC', 'Fried Chicken', 1, 8.99, 25, '2023-06-19'),

(8, 206, 'McDonalds', 'Fries', 3, 2.99, 15, '2023-06-19'),

(9, 207, 'Burger King', 'Burger', 1, 6.99, 30, '2023-06-20'),

(10, 203, 'Starbucks', 'Coffee', 2, 4.50, 20, '2023-06-20')

]

columns1 = ['order\_id', 'customer\_id', 'restaurant\_name', 'food\_item', 'quantity', 'price', 'delivery\_time\_mins', 'order\_date']

spark = SparkSession.builder.appName("Food delivery").getOrCreate()

df2= spark.createDataFrame(data2,columns1)

df2.show()

# Exercises:

# 1. Calculate Total Revenue per Restaurant

# Group the data by restaurant\_name and calculate the total revenue for

# each restaurant. (Hint: Multiply price by quantity to get total

# revenue per order.)

df2 = df2.withColumn('total revenue',F.col('price') \* F.col('quantity'))

group\_by\_restaurant = df2.groupBy('restaurant\_name').agg(F.sum('total revenue').alias('totat\_revenue'))

group\_by\_restaurant.show()

# 2. Find the Fastest Delivery

# Identify the order with the fastest delivery time.

min\_time = df2.orderBy('delivery\_time\_mins').limit(1)

min\_time.show()

# 3. Calculate Average Delivery Time per Restaurant

# Group the data by restaurant\_name and calculate the average delivery

# time for each restaurant.

avg\_deliv\_time = df2.groupBy("restaurant\_name").agg(avg('delivery\_time\_mins').alias('average delivery time'))

avg\_deliv\_time.show()

# 4. Filter Orders for a Specific Customer

# Filter the dataset to include only orders placed by a specific customer

# (e.g., customer\_id = 201 ).

filter\_orders = df2.filter(col('customer\_id')==201)

filter\_orders.show()

# 5. Find Orders Where Total Amount Spent is Greater Than $20

# Filter orders where the total amount spent (price \* quantity) is greater

# than $20.

find\_orders = df2.filter((col('price')\*col('quantity'))>20)

find\_orders.show()

# 6. Calculate the Total Quantity of Each Food Item Sold

# Group the data by food\_item and calculate the total quantity of each

# food item sold.

calc\_quantity = df2.groupBy('food\_item').sum('quantity').alias('total quantity sold')

calc\_quantity.show()

# 7. Find the Top 3 Most Popular Restaurants by Number of Orders

# Identify the top 3 restaurants with the highest number of orders placed.

top\_three = df2.groupBy('restaurant\_name').agg(F.count('order\_id').alias('count'))

topp\_three = top\_three.orderBy(F.desc('count')).limit(3)

topp\_three.show()

# 8. Calculate Total Revenue per Day

# Group the data by order\_date and calculate the total revenue for each

# day.

tot\_revenue = df2.groupBy('order\_date').agg(F.sum('total revenue').alias('total revenue'))

tot\_revenue.show()

# 9. Find the Longest Delivery Time for Each Restaurant

# For each restaurant, find the longest delivery time.

longest\_time = df2.orderBy(F.desc('delivery\_time\_mins')).limit(1)

longest\_time.show()

# 10. Create a New Column for Total Order Value

# Add a new column total\_order\_value that calculates the total value of

# each order ( price \* quantity ).

new\_column=df2.withColumn('total\_order\_value', F.col('price')\* F.col('quantity'))

new\_column.show()

# 4. Dataset: Weather Data

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

data3 = [

('2023-01-01', 'New York', 5, 60, 20, 'Cloudy'),

('2023-01-01', 'Los Angeles', 15, 40, 10, 'Sunny'),

('2023-01-01', 'Chicago', -2, 75, 25, 'Snow'),

('2023-01-02', 'New York', 3, 65, 15, 'Rain'),

('2023-01-02', 'Los Angeles', 18, 35, 8, 'Sunny'),

('2023-01-02', 'Chicago', -5, 80, 30, 'Snow'),

('2023-01-03', 'New York', 6, 55, 22, 'Sunny'),

('2023-01-03', 'Los Angeles', 20, 38, 12, 'Sunny'),

('2023-01-03', 'Chicago', -1, 70, 18, 'Cloudy')

]

columns2 = ['date', 'city', 'temperature\_c', 'humidity', 'wind\_speed\_kph', 'condition']

spark = SparkSession.builder.appName(" Weather data").getOrCreate()

df3= spark.createDataFrame(data3,columns2)

df3.show()

# Exercises:

# 1. Find the Average Temperature for Each City

# Group the data by city and calculate the average temperature for each

# city.

avg\_temp = df3.groupBy('city').agg(avg('temperature\_c').alias('average temperature'))

avg\_temp.show()

# 2. Filter Days with Temperature Below Freezing

# Filter the data to show only the days where the temperature was below

# freezing (below 0°C).

freezing\_temp = df3.filter(F.col('temperature\_c')<0)

freezing\_temp.show()

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

# Find the city with the highest wind speed on a specific day (e.g., 2023-

# 01-02 ).

high\_winds = df3.filter(F.col('date')=='2023-01-02').orderBy(F.desc('wind\_speed\_kph')).limit(1)

high\_winds.show()

# 4. Calculate the Total Number of Days with Rainy Weather

# Count the number of days where the condition was "Rain."

print(df3.filter(F.col('condition')=='Rain').count())

# 5. Calculate the Average Humidity for Each Weather Condition

# Group the data by condition and calculate the average humidity for each

# weather condition (e.g., Sunny, Rainy, Cloudy).

avg\_humidity = df3.groupBy('condition').agg(F.avg('humidity'))

avg\_humidity.show()

# 6. Find the Hottest Day in Each City

# For each city, find the day with the highest recorded temperature.

window\_spec = Window.partitionBy('city').orderBy(F.desc('temperature\_c'))

hottest\_day = df3.withColumn('rank', F.row\_number().over(window\_spec)).filter(F.col('rank') == 1) .select('city', 'date', 'temperature\_c')

hottest\_day.show()

# 7. Identify Cities That Experienced Snow

# Filter the dataset to show only the cities that experienced "Snow" in

# the condition .

snow\_seas = df3.filter(F.col('condition')=='snow')

snow\_seas.show()

# 8. Calculate the Average Wind Speed for Days When the Condition was Sunny

# Filter the dataset for condition = 'Sunny' and calculate the average

# wind speed on sunny days.

sunny\_seas = df3.filter(F.col('condition')=='sunny').agg(F.avg('wind\_speed\_kph').alias('average wind speed'))

sunny\_seas.show()

# 9. Find the Coldest Day Across All Cities

# Identify the day with the lowest temperature across all cities.

lowest\_temp = df3.orderBy(F.asc('temperature\_c')).limit(1)

lowest\_temp.show()

# 10. Create a New Column for Wind Chill

# Add a new column wind\_chill that estimates the wind chill based on the

# formula: [ \text{Wind Chill} = 13.12 + 0.6215 \times \text{Temperature}

# - 11.37 \times (\text{Wind Speed}^{0.16}) + 0.3965 \times

# \text{Temperature} \times (\text{Wind Speed}^{0.16}) ]

# (Assume wind\_speed\_kph is the wind speed in kilometers per hour.)

# Wind Chill Formula: 13.12 + 0.6215 \* Temperature - 11.37 \* (Wind Speed^0.16) + 0.3965 \* Temperature \* (Wind Speed^0.16)

df3 = df3.withColumn('wind\_chill',

13.12 + 0.6215 \* F.col('temperature\_c') - 11.37 \* (F.col('wind\_speed\_kph') \*\* 0.16) +

0.3965 \* F.col('temperature\_c') \* (F.col('wind\_speed\_kph') \*\* 0.16))

df3.select('date', 'city', 'temperature\_c', 'wind\_speed\_kph', 'wind\_chill').show()

# 5. Dataset: Airline Flight Data

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

data4 = [

(1, 'Delta', 'DL123', 'JFK', 'LAX', '08:00', '11:00', 30, 3970, '2023-07-01'),

(2, 'United', 'UA456', 'SFO', 'ORD', '09:30', '15:00', 45, 2960, '2023-07-01'),

(3, 'Southwest', 'SW789', 'DAL', 'ATL', '06:00', '08:30', 0, 1150, '2023-07-01'),

(4, 'Delta', 'DL124', 'LAX', 'JFK', '12:00', '20:00', 20, 3970, '2023-07-02'),

(5, 'American', 'AA101', 'MIA', 'DEN', '07:00', '10:00', 15, 2770, '2023-07-02'),

(6, 'United', 'UA457', 'ORD', 'SFO', '11:00', '14:30', 0, 2960, '2023-07-02'),

(7, 'JetBlue', 'JB302', 'BOS', 'LAX', '06:30', '09:45', 10, 4180, '2023-07-03'),

(8, 'American', 'AA102', 'DEN', 'MIA', '11:00', '14:00', 25, 2770, '2023-07-03'),

(9, 'Southwest', 'SW790', 'ATL', 'DAL', '09:00', '11:00', 5, 1150, '2023-07-03'),

(10, 'Delta', 'DL125', 'JFK', 'SEA', '13:00', '17:00', 0, 3900, '2023-07-04')

]

columns3 = ['flight\_id','airline','flight\_number','origin','destination','departure\_time','arrival\_time','delay\_min','distance','date']

spark = SparkSession.builder.appName(" Airline flight data").getOrCreate()

df4= spark.createDataFrame(data4,columns3)

df4.show()

# # Exercises:

# 1. Find the Total Distance Traveled by Each Airline

# Group the data by airline and calculate the total distance traveled for

# each airline.

tot\_distance = df4.groupBy('airline').sum('distance').alias('total distance')

tot\_distance.show()

# 2. Filter Flights with Delays Greater than 30 Minutes

# Filter the dataset to show only flights where the delay was greater than

# 30 minutes.

greater\_delay = df4.filter(F.col('delay\_min')>30)

greater\_delay.show()

# 3. Find the Flight with the Longest Distance

# Identify the flight that covered the longest distance.

longest\_dist = df4.orderBy(F.desc('distance')).limit(1)

longest\_dist.show()

# 4. Calculate the Average Delay Time for Each Airline

# Group the data by airline and calculate the average delay time in

# minutes for each airline.

avg\_delay\_time = df4.groupBy('airline').agg(F.avg('delay\_min').alias('average delay time'))

avg\_delay\_time.show()

# 5. Identify Flights That Were Not Delayed

# Filter the dataset to show only flights with delay\_minutes = 0 .

not\_delayed = df4.filter(F.col('delay\_min')==0)

not\_delayed.show()

# 6. Find the Top 3 Most Frequent Routes

# Group the data by origin and destination to find the top 3 most

# frequent flight routes.

double\_grouping = df4.groupBy('origin','destination').count().orderBy(F.desc('count')).limit(3)

double\_grouping.show()

# 7. Calculate the Total Number of Flights per Day

# Group the data by date and calculate the total number of flights on

# each day.

total\_flights = df4.groupBy('date').count()

total\_flights.show()

# 8. Find the Airline with the Most Flights

# Identify the airline that operated the most flights.

most\_flights = df4.groupBy('airline').count().orderBy(F.desc('count')).limit(1)

most\_flights.show()

# 9. Calculate the Average Flight Distance per Day

# Group the data by date and calculate the average flight distance for

# each day.

avg\_flight\_dist = df4.groupBy('date').agg(F.avg('distance').alias('flight distance'))

avg\_flight\_dist.show()

# 10. Create a New Column for On-Time Status

# Add a new column called on\_time that indicates whether a flight was on

# time ( True if delay\_minutes = 0 , otherwise False ).

new\_col\_flight = df4.withColumn('on\_time',F.when(F.col('delay\_min')==0,True).otherwise(False))

new\_col\_flight.show()