PYSPARK EXERCISES

1.Dataset: Fitness Tracker Data

```
#1.Dataset: Fitness Tracker Data

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

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']

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

# Create a DataFrame from the data
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 Where a User Burned More Than 500 Calories
high_calorie_days = df.filter(df['calories'] > 500)
print("Days with more than 500 calories burned:\n", high_calorie_days.show())

# 3. Calculate the Average Distance Traveled by Each User
average_distance_per_user = df.groupby('user_id').agg(F.avg('distance_km').alias('average_distance'))
print("Average Distance Traveled by Each User:\n", average_distance_per_user.show())

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

# 5. Find Users Who Were Active for More Than 100 Minutes on Any Day
active_users = df.filter(df['active_minutes'] > 100).select('user_id')
print("Users who were active for more than 100 minutes:\n", active_users.show())

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

# 7. Calculate the Average Steps per Day
average_steps_per_day = df.groupby('date').agg(F.avg('steps').alias('average_steps'))
print("Average Steps per Day:\n", average_steps_per_day.show())
```

```
# 8. Rank Users by Total Distance Traveled
user_rank =
df.groupby('user_id').agg(F.sum('distance_km').alias('total_distance')).sort(F.col('total_distance').desc())
print("Users ranked by total distance traveled:\n", user_rank.show())

# 9. Find the Most Active User by Total Active Minutes
most_active_user =
df.groupby('user_id').agg(F.sum('active_minutes').alias('total_active_minutes')).sort(F.col('total_active_minut
es').desc()).first()
print("Most Active User:\n", most_active_user)

# 10. Create a New Column for Calories Burned per Kilometer
df = df.withColumn('calories_per_km', F.col('calories') / F.col('distance_km'))
print("Data with calories_per_km column:\n", df.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

```
# 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

```
# 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').alias("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

```
# 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','dist
ance','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()
```

```python
# 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.

tot_flights = df4.groupBy('date').count()
tot_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()
```