## PySpark Assignment:

## Fitness Tracker Data Exercise:

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import col

from pyspark.sql import functions as F

from pyspark.sql.window import Window


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


fitness_df =
spark.read.format("csv").option("header","true").option("inferSchema","true").load("/cont
ent/sample_data/fitness_tracker.csv")
```

**# 1. Find the Total Steps Taken by Each User**

```
df_total_steps_user = fitness_df.groupBy("user_id").agg(F.sum("steps").alias("TotalSteps"))

df_total_steps_user.show()
```

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

```
df_high_calories = fitness_df.filter(col("calories") > 500).select("date")

df_high_calories.show()
```

**#  3. Calculate the Average Distance Traveled by Each User**

```
df_avg_dist =
fitness_df.groupBy("user_id").agg(F.avg("distance_km").alias("AverageDistance"))

df_avg_dist.show()
```

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

```
window_spec = Window.partitionBy("user_id").orderBy(col("steps").desc())

df_max_steps =
fitness_df.withColumn("rank",F.rank().over(window_spec)).filter(col("rank")==1)
```

```
df_max_steps.show()
```

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

```
df_high_active = fitness_df.filter(col("active_minutes ") > 
100).select("user_id","date","active_minutes ")

df_high_active.show()
```

**# 6. Calculate the Total Calories Burned per Day**

```
df_calories_day = fitness_df.groupBy("date").agg(F.sum("calories").alias("TotalCalories"))

df_calories_day.show()
```

**# 7. Calculate the Average Steps per Day**

```
df_avg_steps = fitness_df.groupBy("date").agg(F.avg("steps").alias("AverageSteps"))

df_avg_steps.show()
```

**# 8. Rank Users by Total Distance Traveled**

```
total_distance_df = 
fitness_df.groupBy("user_id").agg(F.sum("distance_km").alias("totalDistance"))

window_spec2 = Window.orderBy(col("totalDistance").desc())

ranked_user_df = total_distance_df.withColumn("rank", F.rank().over(window_spec2))

ranked_user_df.show()
```

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

```
df_total_active_minutes = fitness_df.groupBy("user_id").agg(F.sum("active_minutes 
").alias("totalMinutes"))

most_active_user = df_total_active_minutes.orderBy(col("totalMinutes").desc()).limit(1)

most_active_user.show()
```

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

```
fitness_df = fitness_df.withColumn("calories_per_km", col("calories") / col("distance_km"))
fitness_df.show()
```

## Book Sales Exercise:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.sql import functions as F
from pyspark.sql.window import Window

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

book_df =
spark.read.format("csv").option("header","true").option("inferSchema","true").load("/cont
ent/sample_data/book_data.csv")
```

**# 1. Total Sales Revenue per Genre**

```
total_sales_per_genre = book_df.groupBy("genre").agg(F.sum(F.col("sale_price") *
F.col("quantity")).alias("total_revenue"))
total_sales_per_genre.show()
```

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

```
fiction_books = book_df.filter(F.col("genre") == "Fiction")
fiction_books.show()
```

**# 3. Book with the Highest Sale Price**

```
book_highest_price = book_df.orderBy(F.col("sale_price").desc()).limit(1)
book_highest_price.show()
```

**# 4. Total Quantity of Books Sold by Author**

```
total_quantity_by_author =
book_df.groupBy("author").agg(F.sum("quantity").alias("total_quantity"))

total_quantity_by_author.show()
```

**# 5. Sales Transactions Worth More Than $50**

```
sales_over_50 = book_df.filter((F.col("sale_price") * F.col("quantity")) > 50)

sales_over_50.show()
```

**# 6. Average Sale Price per Genre**

```
average_sale_price_per_genre =
book_df.groupBy("genre").agg(F.avg("sale_price").alias("average_price"))

average_sale_price_per_genre.show()
```

**# 7. Count the Number of Unique Authors**

```
unique_authors_count = book_df.select("author").distinct().count()

print(f"Number of unique authors: {unique_authors_count}")
```

**# 8. Top 3 Best-Selling Books by Quantity**

```
top_3_books_by_quantity =
book_df.groupBy("book_title").agg(F.sum("quantity").alias("total_quantity")).orderBy(F.col(
"total_quantity").desc()).limit(3)

top_3_books_by_quantity.show()
```

**# 9. Total Sales for Each Month**

```
total_sales_by_month = book_df.withColumn("month",
F.month("date")).groupBy("month").agg(F.sum(F.col("sale_price") *
F.col("quantity")).alias("total_revenue"))

total_sales_by_month.show()
```

**# 10. Create a New Column for Total Sales Amount**

```
book_df_with_total_sales = book_df.withColumn("total_sales", F.col("sale_price") * F.col("quantity"))

book_df_with_total_sales.show()
```

## Food Delivery:

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import col

from pyspark.sql import functions as F

from pyspark.sql.window import Window


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


food_df = spark.read.format("csv").option("header","true").option("inferSchema","true").load("/content/sample_data/food_delivery.csv")



# 1. Calculate Total Revenue per Restaurant

total_revenue_per_restaurant = food_df.groupBy("restaurant_name").agg(F.sum(F.col("price") * F.col("quantity")).alias("total_revenue"))

total_revenue_per_restaurant.show()


# 2. Find the Fastest Delivery

fastest_delivery = food_df.orderBy(F.col("delivery_time_mins")).limit(1)

fastest_delivery.show()


# 3. Calculate Average Delivery Time per Restaurant

avg_delivery_time_per_restaurant = food_df.groupBy("restaurant_name").agg(F.avg("delivery_time_mins").alias("avg_delivery_time"))
```

```
avg_delivery_time_per_restaurant.show()
```

**# 4. Filter Orders for a Specific Customer (customer_id = 201)**

```
customer_orders = food_df.filter(F.col("customer_id") == 201)
customer_orders.show()
```

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

```
orders_over_20 = food_df.filter((F.col("price") * F.col("quantity")) > 20)
orders_over_20.show()
```

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

```
total_quantity_per_food_item =
food_df.groupBy("food_item").agg(F.sum("quantity").alias("total_quantity"))
total_quantity_per_food_item.show()
```

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

```
top_3_restaurants =
food_df.groupBy("restaurant_name").agg(F.count("order_id").alias("num_orders")).orderB
y(F.col("num_orders").desc()).limit(3)
top_3_restaurants.show()
```

**# 8. Calculate Total Revenue per Day**

```
total_revenue_per_day = food_df.groupBy("order_date").agg(F.sum(F.col("price") *
F.col("quantity")).alias("total_revenue"))
total_revenue_per_day.show()
```

**# 9. Find the Longest Delivery Time for Each Restaurant**

```
longest_delivery_time =
food_df.groupBy("restaurant_name").agg(F.max("delivery_time_mins").alias("max_deliver
y_time"))
longest_delivery_time.show()
```

**# 10. Create a New Column for Total Order Value**

```
total_order_value = food_df.withColumn("total_order_value", F.col("price") *
F.col("quantity"))

total_order_value.show()
```

## Weather Data:

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import col

from pyspark.sql import functions as F

from pyspark.sql.window import Window


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


weather_df =
spark.read.format("csv").option("header","true").option("inferSchema","true").load("/cont
ent/sample_data/weather_data.csv")
```

**# 1. Find the Average Temperature for Each City**

```
df_avg_temmp_city =
weather_df.groupBy("city").agg(F.avg("temperature_c").alias("average_temperature"))

df_avg_temmp_city.show()
```

**#  2. Filter Days with Temperature Below Freezing**

```
df_freezing = weather_df.filter(col("temperature_c") < 0)

df_freezing.show()
```

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

```
df_highest_wind = weather_df.filter(col("date") == "2023-01-02").orderBy(col("wind_speed_kph").desc()).limit(1)

df_highest_wind.show()
```

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

```
rainy_days_count = weather_df.filter(col("condition") == "Rain").count()

print(f"Total number of rainy days: {rainy_days_count}")
```

**# 5. Calculate the Average Humidity for Each Weather Condition**

```
df_avg_humidity_per_condition = weather_df.groupBy("condition").agg(F.avg("humidity").alias("avg_humidity"))

df_avg_humidity_per_condition.show()
```

**# 6. Find the Hottest Day in Each City**

```
df_hottest_day = weather_df.groupBy("city").agg(F.max("temperature_c").alias("max_temperature"))

df_hottest_day.show()
```

**# 7. Identify Cities That Experienced Snow**

```
df_snow = weather_df.filter(col("condition") == "Snow").select("city")

df_snow.show()
```

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

```
df_avg_wind_sunny_days = weather_df.filter(col("condition") == "Sunny").agg(F.avg("wind_speed_kph").alias("avg_wind_speed"))

df_avg_wind_sunny_days.show()
```

**# 9. Find the Coldest Day Across All Cities**

```
coldest_day = weather_df.orderBy(col("temperature_c").asc()).limit(1)

coldest_day.show()
```

**# 10. Create a New Column for Wind Chill**

```
wind_chill_df = weather_df.withColumn("wind_chill", 13.12+0.6215 * col("temperature_c") - 11.37 * (col("wind_speed_kph")**0.16) + 0.3965 * col("temperature_c") * (col("wind_speed_kph")**0.16))

wind_chill_df.show()
```

## Airline Flight Data:

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import col

from pyspark.sql import functions as F

from pyspark.sql.window import Window


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


flight_df = spark.read.format("csv").option("header","true").option("inferSchema","true").load("/content/sample_data/flight_data.csv")
```

**# 1. Find the Total Distance Traveled by Each Airline**

```
df_airline_distance = flight_df.groupBy("airline").agg(F.sum("distance_travelled").alias("distance_travelled"))

df_airline_distance.show()
```

**# 2. Filter Flights with Delays Greater than 30 Minutes**

```
df_high_delay = flight_df.filter(col("delay_min")>30)
```

```
df_high_delay.show()

# 3. Find the Flight with the Longest Distance
df_longest_dist = flight_df.orderBy(col("distance_travelled").desc()).limit(1)
df_longest_dist.show()

# 4. Calculate the Average Delay Time for Each Airline
df_avg_delay = flight_df.groupBy("airline").agg(F.avg("delay_min").alias("average_delay"))
df_avg_delay.show()

# 5. Identify Flights That Were Not Delayed
df_no_delay = flight_df.filter(col("delay_min") == 0)
df_no_delay.show()

# 6. Find the Top 3 Most Frequent Routes
df_frequent_routes = flight_df.groupBy("origin","destination").agg(F.count("*").alias("route_count")).orderBy(F.col("route_count").desc()).limit(3)
df_frequent_routes.show()

# 7. Calculate the Total Number of Flights per Day
df_total_per_day = flight_df.groupBy("date").agg(F.count("*").alias("flight_per_day"))
df_total_per_day.show()

# 8. Find the Airline with the Most Flights
airline_with_most_flights = flight_df.groupBy("airline").agg(F.count("*").alias("flight_count")).orderBy(F.col("flight_count").desc()).limit(1)
airline_with_most_flights.show()
```

**# 9. Calculate the Average Flight Distance per Day**

```
avg_flight_distance_per_day =
flight_df.groupBy("date").agg(F.avg("distance_travelled").alias("avg_distance"))

avg_flight_distance_per_day.show()
```

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

```
flight_df_with_on_time = flight_df.withColumn("on_time", F.when(F.col("delay_min") == 0,
True).otherwise(False))

flight_df_with_on_time.show()
```