**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("/content/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("/content/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")).orderBy(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\_delivery\_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("/content/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()