# Import the required libraries

from pyspark import SparkContext, SparkConf

# Step **1:** Create a Spark Configuration and Spark Context

conf = SparkConf().setAppName("SparkBasics").setMaster("local[\*]")

SC =

SparkContext(conf=conf)

# Step 2: Create an RDD from a list

data **[1,** 2, 3, 4, 5]

rdd =

sc.parallelize(data)

# Step 3: Perform operations on the RDD

# Example: Multiply each element by 2

mapped\_rdd

=

rdd.map(lambda x: x \* 2)

# Collect the result

result **=** mapped\_rdd.collect()

# Step 4: Print the result

print("Original Data:", data)

print("Transformed Data (multiplied by 2):", result)

# Stop the SparkContext

sc.stop()

Original Data: **[**1, 2, 3, 4, 5]

Transformed Data (multiplied by 2): [2, 4, 6, 8, 10]

from pyspark import SparkContext, SparkConf

# Step **1**: Set up SparkContext

conf =

SC =

SparkConf().setAppName("MinMaxTemperature").setMaster("local [\*]")

SparkContext(conf=conf)

# Step 2: Sample temperature data in the form (Location, Temperature)

data

=

[

("New York", 30),

("Los Angeles", 25),

("New York", 28),

("Los Angeles", 22), ("New York", 35),

("Los Angeles", 20),

("Chicago"**,** 15),

("Chicago", 10),

("Chicago", 12)

# Step 3: Create an RDD from the data

rdd =

sc.parallelize(data)

# Step 4: Group data by location and calculate both min and max temperatures for each location grouped\_rdd rdd.groupByKey()

=

# Find both minimum and maximum temperatures

min\_max\_temp\_rdd

=

grouped\_rdd.mapValues (lambda temps: **(**min(temps), max(temps)))

# Step 5: Collect the results and display

result

=

min\_max\_temp\_rdd.collect()

print("Minimum and Maximum Temperatures by Location:")

for location, (min\_temp, max\_temp) in result:

print(f" {location}: Min

=

{min\_temp}°C, Max = {max\_temp}°C")

# Stop the SparkContext

sc.stop()

**⇒** Minimum and Maximum Temperatures by Location:

New York: Min ==

28°C, Max = 35°C

Los Angeles: Min = 20°C, Max = 25°C

Chicago: Min = 10°C, Max = 15°C

from pyspark.sql import SparkSession

from pyspark.sql.functions import explode, split, col, count

# Step 1: Create a SparkSession

spark

=

SparkSession.builder \

.appName("WordCountDataFrame") \

.master("local[\*]") \

.getOrCreate()

# Step 2: Create a DataFrame with text data

data =

columns

[("Hello world spark spark",), ("Hello again spark",), ("Welcome to the world of spark",)]

["line"]

=

df = spark.createDataFrame(data, columns)

# Step 3: Split lines into words, explode to create one word per row

words\_df

=

df.select(explode (split(col("line"),

"I

")).alias("word"))

# Step 4: Count occurrences of each word

word\_count\_df = words\_df.groupBy("word").agg (count("word")**.**alias("count"))

# Step 5: Show the result

word\_count\_df.show()

# Stop the SparkSession

spark.stop()

+-

word count

│Hello

2|

│spark❘

| world

| again│

|

the

**1|**

of

| Welcome |

**1|**

to

**1|**

**+**-

+-----+

from pyspark.sql import SparkSession

from pyspark.sql.functions import avg, sum**, col**

# Step **1:** Create a SparkSession

spark

=

SparkSession.builder \

.appName("SQLCommands OnDataFrame") \

.master("local[\*]") \

.getOrCreate()

# Step 2: Create a DataFrame with sample data

data

=

[

("Alice", "Electronics", 200),

("Bob", "Electronics", 150),

("Alice", "Clothing", 100),

("Bob", "Clothing", 50),

("Alice", "Electronics", 300)

columns

=

["Name", "Category", "Amount"]

df = spark.createDataFrame(data, columns)

# Step 3: Execute SQL-style functions

# Calculate total and average spending by category

result\_df = df.groupBy("Category") \

.agg(

sum("Amount").alias("TotalSpent"),

avg("Amount").alias ("AverageSpent")

>

# Step 4: Show the result

result\_df.show()

# Stop the SparkSession

spark.stop()

⇒ +-

| Category | TotalSpent|

| Electronics |

Clothing|

+-

AverageSpent❘

650 216.66666666666666 | 150

75.0

**-**+

from pyspark.sql import SparkSession from pyspark.sql.functions import sum

# Step 1: Create a SparkSession

spark

=

SparkSession.builder \ .appName("TotalSpentByCustomer") \

.master("local[\*]") \

.getOrCreate()

# Step 2: Create a DataFrame with sample data

data

]

=

[

("Alice", 200), ("Bob", 150),

("Alice"**,** 100),

("Bob", 50),

("Alice", 300)

columns = ["Customer", "Amount"]

df = spark.createDataFrame(data, columns)

# Step 3: Group by Customer and calculate the total amount spent result\_df

=

df.groupBy("Customer").agg (sum("Amount").alias ("TotalSpent"))

# Step 4: Show the result

result\_df.show()

# Stop the SparkSession spark.stop()

+-

Customer TotalSpent |

Bob Alice

+-

200 |

600

**+**

from pyspark.sql import SparkSession

# Step **1:** Create a SparkSession

= SparkSession.builder \

spark

.appName("BroadcastVariables Example") \

.master("local[\*]") \

.getOrCreate()

# Step 2: Create a DataFrame with movie IDs and ratings

movie\_data [

**(**1**, 4.5**),

(2, 3.0),

**(1,** 5.0),

(3, 4.0)

columns = ["MovieID", "Rating"]

ratings\_df = spark.createDataFrame(movie\_data, columns)

# Step 3: Create a dictionary with movie names

movie\_names

=

{

1: "Inception",

2: "Interstellar",

3: "The Dark Knight"

}

# Step 4: Broadcast the movie names dictionary

movie\_names\_broadcast

=

spark.sparkContext.broadcast(movie\_names)

# Step 5: Map MovieID to Movie Name using the broadcast variable

def map\_movie\_name (movie\_id):

return movie\_names\_broadcast.value.get(movie\_id, "Unknown")

# Register the function as a UDF

from pyspark.sql.functions import udf

from pyspark.sql.types import StringType

map\_movie\_name\_udf = udf (map\_movie\_name, StringType())

**#** Add a new column with movie names

result\_df

=

ratings\_df.withColumn ("MovieName"**,** map\_movie\_name\_udf (ratings\_df.MovieID))

# Step 6: Show the result

result\_df.show()

# Stop the SparkSession spark.stop()

+

MovieID | Rating|

MovieName |

**1**

4.5

Inception |

21

3.0

Interstellar |

**1**

5.0

Inception |

3| 4.0 The Dark Knight|

**+**

from pyspark.sql import SparkSession

from pyspark.ml.recommendation import ALS

from pyspark.sql.types import IntegerType, FloatType

# Step **1:** Create a SparkSession

spark

= SparkSession.builder \

.appName("MovieRecommendations") \

.master("local[\*]") \

.getOrCreate()

# Step 2: Create a DataFrame with sample user**-**movie ratings

[

data =

(0, **1,** 4.0), # User rated Movie 1 with 4.0

# User **1** rated Movie 2 with 5.0

# User 1 rated Movie 3 with 3.0

(0, 2, 2.0), **(1**, 2, 5.0), **(**1, 3, 3.0), (2**, 1,** 1.0), (2, 3, 4.0)

# User rated Movie 2 with 2.0

]

columns =

# User 2 rated Movie **1** with 1.0

**#** User 2 rated Movie 3 with 4.0

["UserID", "MovieID", "Rating"]

ratings\_df = spark.createDataFrame(data, columns)

# Step 3: Create an ALS model

als = ALS(

maxIter=5, # Number of iterations

regParam=0.01,

userCol="UserID",

# Regularization parameter

**#** Column for users

>

itemCol="MovieID", # Column for items

ratingCol="Rating", # Column for ratings coldStartStrategy="drop"

# Step 4: Train the ALS model

model

=

als.fit(ratings\_df)

# Drop rows with null predictions

# Step 5: Generate top 2 movie recommendations for each user

user\_recommendations =

model.recommendForAllUsers (2)

# Step 6: Show the recommendations

user\_recommendations.show(truncate=False)

# Stop the SparkSession

spark.stop()

**H**

UserID recommendations

|0

|[**{1,** 3.994191}, {2, 2.0000634}]| [[2, 4.996521}, {3, 3.0001428}]| |[{3, 3.9961724}, {2, 3.443232}]|

-+

from pyspark.sql import SparkSession

from pyspark.sql. functions import col, count, desc

# Step 1: Create a SparkSession

spark

=

SparkSession.builder \

.appName("Most ViewedURL") \

.master("local[\*]") \

.getOrCreate()

# Step 2: Create a DataFrame with sample log data

data =

[

**(1,** "http://example.com/home"), (2, "http://example.com/about"), (3, "http://example.com/home"), (4, "http://example.com/contact"), (5, "http://example.com/home"), (6, "http://example.com/about"), (7, "http://example.com/contact"), (8, "http://example.com/home")

columns = ["UserID", "URL"]

logs\_df = spark.createDataFrame(data, columns)

# Step 3: Count the number of views for each URL

url\_counts\_df = logs\_df.groupBy("URL").agg (count("URL").alias("ViewCount"))

# Step 4: Sort the URLS by the number of views in descending order sorted\_url\_counts\_df = url\_counts\_df.orderBy(desc("ViewCount"))

# Step 5: Show the most viewed URLs

sorted\_url\_counts\_df.show(**)**

# Stop the SparkSession

spark.stop()

€

+

URL |ViewCount|

21

http://example.co.... http://example.co**...**

http://example.co...|.

2|

+

Start coding or generate with AI.