VASANTDADA PATIL PRATISHTHAN'S COLLEGE OF ENGINEERING AND VISUAL ARTS

ISO 9001:2015 Certified Institute

Department of Information Technology

NBA Accredited Course (Dated 01/07/2024 to 30/06/2027)

EXPERIMENT - 10

Aim: Batch and streamed Data Analysis using Spark.

Theory:

1. Introduction to Big Data Processing

Big data refers to large and complex datasets that require advanced techniques for storage, processing, and analysis. Traditional methods are inefficient in handling large-scale data, leading to the use of distributed computing frameworks like **Apache Spark.**

2. Apache Spark for Data Processing

Apache Spark is a fast, in-memory, distributed computing framework designed for big data processing. It supports various workloads, including **batch processing, real-time streaming, machine learning, and graph processing.**

- **Batch Processing:** Involves processing large volumes of stored data in chunks. It is suitable for historical data analysis and reporting.
- **Stream Processing:** Involves processing real-time data as it arrives, making it suitable for live monitoring and real-time analytics.

3. PySpark for Data Science

PySpark is the Python API for Apache Spark. It enables data scientists to work with Spark's distributed computing power using Python. Key components of PySpark include:

- **SparkSession:** Entry point for PySpark applications.
- **DataFrame API:** A distributed data structure similar to Pandas DataFrame but optimized for parallel computation.
- **Spark Streaming:** A real-time data processing framework in Spark.

4. Dataset Used

In this experiment, a dataset related to **hair health and body metrics** is used. It contains numeric values for factors like **total protein, keratin, vitamins, minerals, body water content, stress levels, liver data, and hair fall.**

5. Batch Processing in PySpark

Batch processing is performed using PySpark's **DataFrame API** by:

1. Loading the dataset into a Spark DataFrame.

2. Performing exploratory data analysis (EDA) using functions like describe(), groupBy(), and aggregations.

3. Finding trends and correlations, such as the relationship between **stress levels and hair fall.**

6. Stream Processing in PySpark

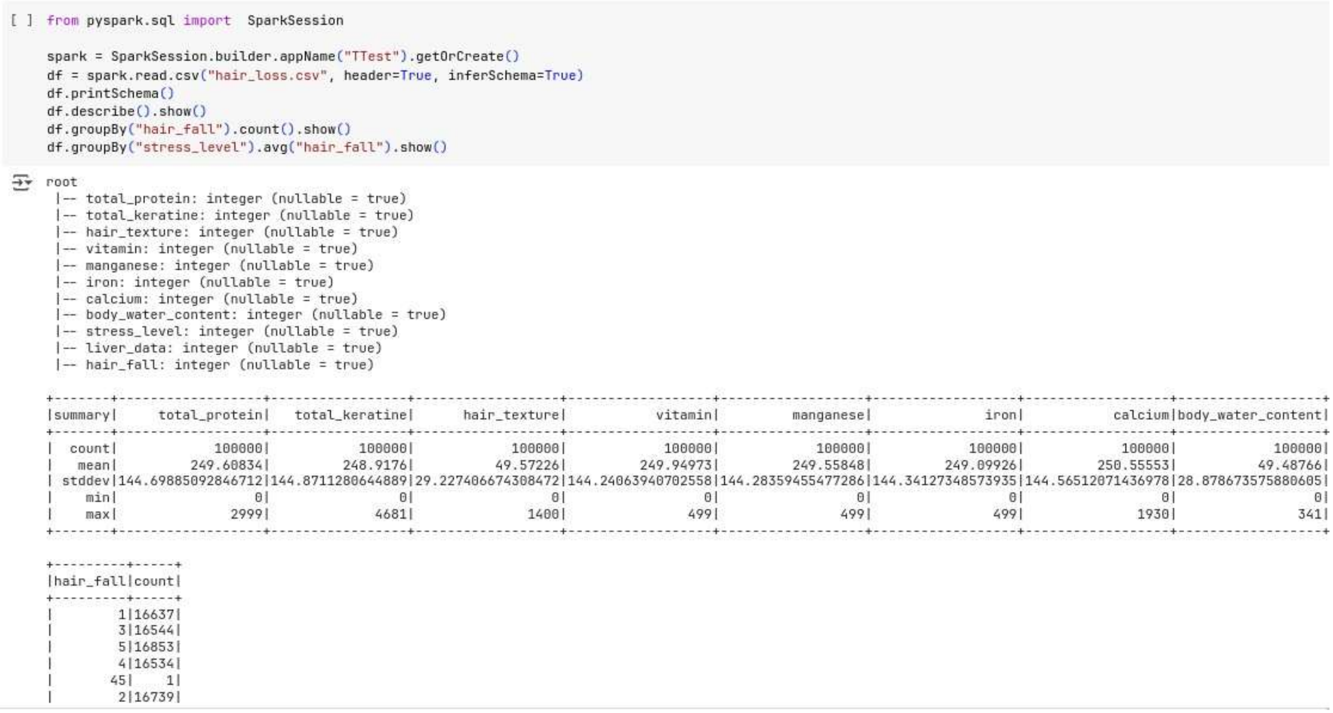
Streaming data processing is done using **Spark Structured Streaming**, where:

1. A streaming DataFrame is created from a folder containing continuously incoming CSV files.

2. Data is processed in near real-time using transformations like groupBy() and agg().

3. The results are displayed continuously using the .writeStream().outputMode("complete") function.

Program:



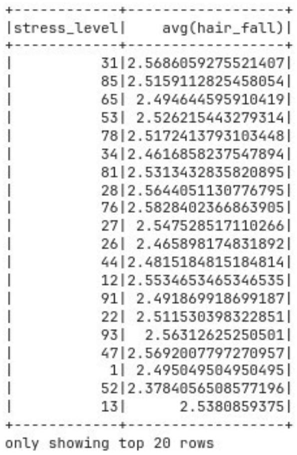
```
[ ] from pyspark.sql import  SparkSession

    spark = SparkSession.builder.appName("TTest").getOrCreate()
    df = spark.read.csv("hair_loss.csv", header=True, inferSchema=True)
    df.printSchema()
    df.describe().show()
    df.groupBy("hair_fall").count().show()
    df.groupBy("stress_level").avg("hair_fall").show()
```

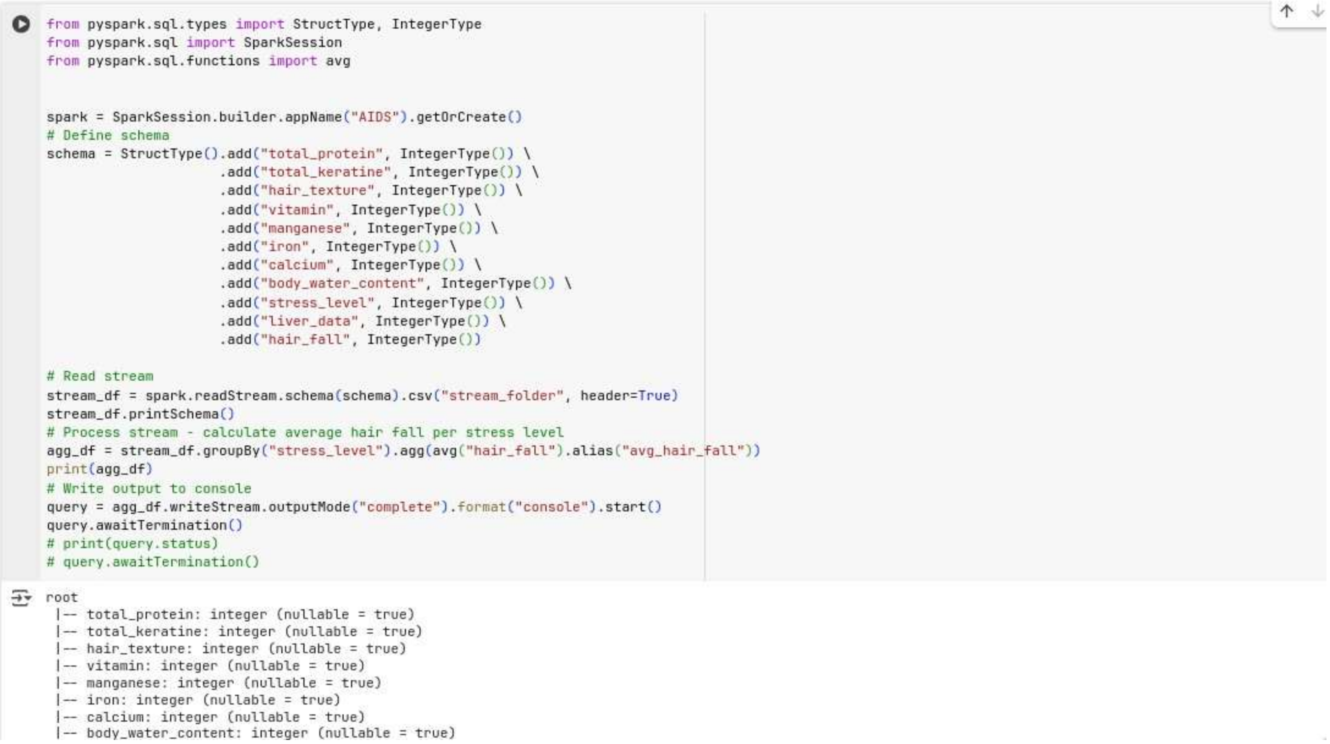
```
root
 |-- total_protein: integer (nullable = true)
 |-- total_keratine: integer (nullable = true)
 |-- hair_texture: integer (nullable = true)
 |-- vitamin: integer (nullable = true)
 |-- manganese: integer (nullable = true)
 |-- iron: integer (nullable = true)
 |-- calcium: integer (nullable = true)
 |-- body_water_content: integer (nullable = true)
 |-- stress_level: integer (nullable = true)
 |-- liver_data: integer (nullable = true)
 |-- hair_fall: integer (nullable = true)

+-------+------------------+-----------------+------------------+------------------+------------------+------------------+------------------+------------------+
|summary|     total_protein|   total_keratine|      hair_texture|           vitamin|         manganese|              iron|           calcium|body_water_content|
+-------+------------------+-----------------+------------------+------------------+------------------+------------------+------------------+------------------+
|  count|            100000|           100000|            100000|            100000|            100000|            100000|            100000|            100000|
|   mean|         249.60834|         248.9176|          49.57226|         249.94973|         249.55848|         249.09926|         250.55553|          49.48766|
| stddev|144.69885092846712|144.8711280644889|29.227406674308472|144.24063940702558|144.28359455477286|144.34127348573935|144.56512071436978|28.878673575880605|
|    min|                 0|                0|                 0|                 0|                 0|                 0|                 0|                 0|
|    max|              2999|             4681|              1400|               499|               499|               499|              1930|               341|
+-------+------------------+-----------------+------------------+------------------+------------------+------------------+------------------+------------------+

+---------+-----+
|hair_fall|count|
+---------+-----+
|        1|16637|
|        3|16544|
|        5|16853|
|        4|16534|
|       45|    1|
|        2|16739|
```



```
+------------+------------------+
|stress_level|    avg(hair_fall)|
+------------+------------------+
|          31|2.5686059275521407|
|          85|2.5159112825458054|
|          65| 2.494644595910419|
|          53| 2.526215443279314|
|          78|2.5172413793103448|
|          34|2.4616858237547894|
|          81|2.5313432835820895|
|          28|2.5644051130776795|
|          76|2.5828402366863905|
|          27| 2.547528517110266|
|          26| 2.465898174831892|
|          44|2.4815184815184814|
|          12|2.5534653465346535|
|          91| 2.491869918699187|
|          22| 2.511530398322851|
|          93|  2.56312625250501|
|          47|2.5692007797270957|
|           1| 2.495049504950495|
|          52|2.3784056508577196|
|          13|      2.5380859375|
+------------+------------------+
only showing top 20 rows
```

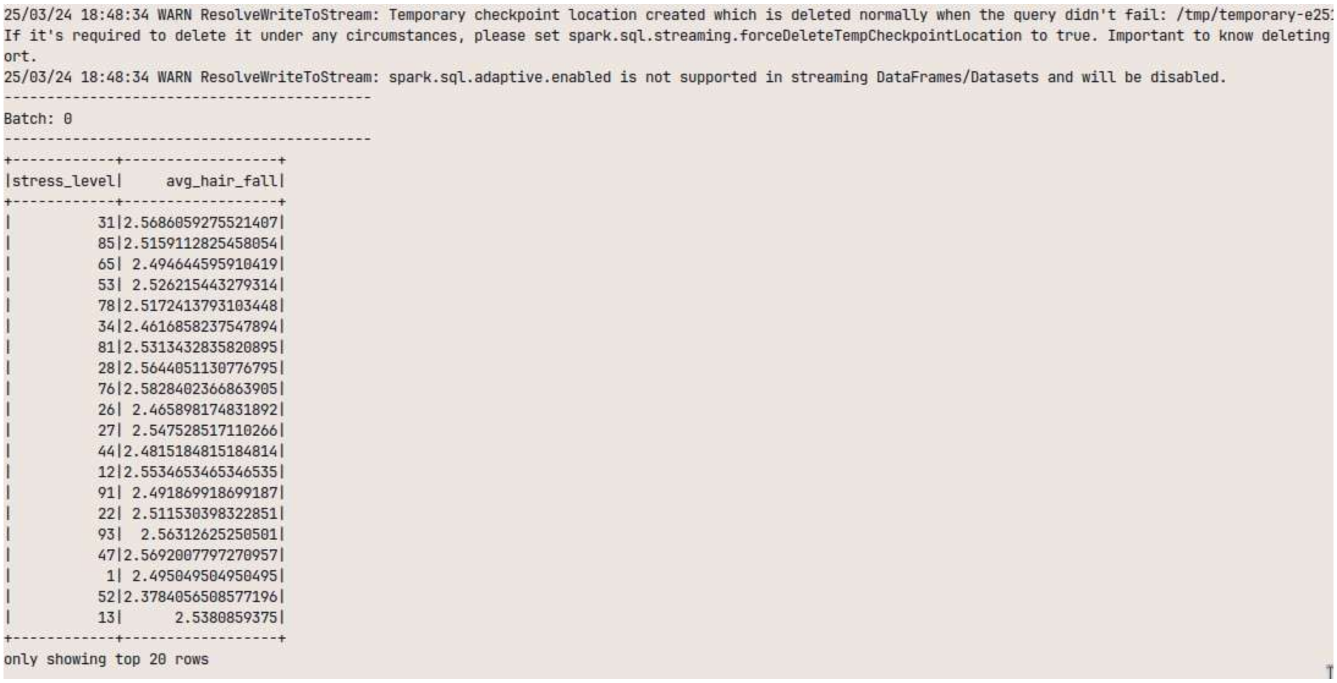


```
from pyspark.sql.types import StructType, IntegerType
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg


spark = SparkSession.builder.appName("AIDS").getOrCreate()
# Define schema
schema = StructType().add("total_protein", IntegerType()) \
                     .add("total_keratine", IntegerType()) \
                     .add("hair_texture", IntegerType()) \
                     .add("vitamin", IntegerType()) \
                     .add("manganese", IntegerType()) \
                     .add("iron", IntegerType()) \
                     .add("calcium", IntegerType()) \
                     .add("body_water_content", IntegerType()) \
                     .add("stress_level", IntegerType()) \
                     .add("liver_data", IntegerType()) \
                     .add("hair_fall", IntegerType())

# Read stream
stream_df = spark.readStream.schema(schema).csv("stream_folder", header=True)
stream_df.printSchema()
# Process stream - calculate average hair fall per stress level
agg_df = stream_df.groupBy("stress_level").agg(avg("hair_fall").alias("avg_hair_fall"))
print(agg_df)
# Write output to console
query = agg_df.writeStream.outputMode("complete").format("console").start()
query.awaitTermination()
# print(query.status)
# query.awaitTermination()
```

```
root
 |-- total_protein: integer (nullable = true)
 |-- total_keratine: integer (nullable = true)
 |-- hair_texture: integer (nullable = true)
 |-- vitamin: integer (nullable = true)
 |-- manganese: integer (nullable = true)
 |-- iron: integer (nullable = true)
 |-- calcium: integer (nullable = true)
 |-- body_water_content: integer (nullable = true)
```



```
25/03/24 18:48:34 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the query didn't fail: /tmp/temporary-e25
If it's required to delete it under any circumstances, please set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting
ort.
25/03/24 18:48:34 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.
-------------------------------------------
Batch: 0
-------------------------------------------
+------------+------------------+
|stress_level|     avg_hair_fall|
+------------+------------------+
|          31|2.5686059275521407|
|          85|2.5159112825458054|
|          65| 2.494644595910419|
|          53| 2.526215443279314|
|          78|2.5172413793103448|
|          34|2.4616858237547894|
|          81|2.5313432835820895|
|          28|2.5644051130776795|
|          76|2.5828402366863905|
|          26| 2.465898174831892|
|          27| 2.547528517110266|
|          44|2.4815184815184814|
|          12|2.5534653465346535|
|          91| 2.491869918699187|
|          22| 2.511530398322851|
|          93|  2.56312625250501|
|          47|2.5692007797270957|
|           1| 2.495049504950495|
|          52|2.3784056508577196|
|          13|      2.5380859375|
+------------+------------------+
only showing top 20 rows
```

Conclusion: Thus, we have successfully performed Batch and streamed Data Analysis using Spark.