Objective 1 - Conceptual Foundations (8 points)

This week's lesson introduces Spark SQL, a key component of Apache Spark designed to simplify big data processing for users familiar with SQL. While Spark Core and RDDs offer flexibility, they can be complex to work with. Spark SQL bridges that gap by allowing users to write standard SQL queries that Spark executes in parallel across a distributed cluster. This makes Spark more accessible to data analysts and engineers alike, enabling them to leverage powerful distributed computing without having to write complex low-level code.

Under the hood, Spark SQL follows a structured process. When a user submits a SQL query, Spark first parses the query to validate syntax, then analyzes it to identify the tables and columns being referenced. The Catalyst Optimizer, Spark's optimization engine, then restructures the query for maximum efficiency by reordering joins, pruning unnecessary columns, or pushing filters closer to the data. Finally, Spark generates an execution plan—a series of parallel tasks distributed across the cluster—which allows it to process massive datasets quickly. Spark SQL can connect to various data sources, including Hive tables, Parquet files, CSVs, relational databases, and cloud storage platforms like AWS S3 or Azure ADLS.

Ultimately, Spark SQL combines the ease of SQL with the scalability of Spark's distributed framework. It enables analysts to run queries on huge datasets without worrying about cluster management, while developers retain the flexibility to integrate SQL queries with Spark's APIs in Python or Scala. This architecture—originally pioneered by Spark SQL—now underpins modern cloud systems such as Databricks SQL, BigQuery, and Snowflake, all of which build on Spark's model of bringing database-like simplicity to big data processing.

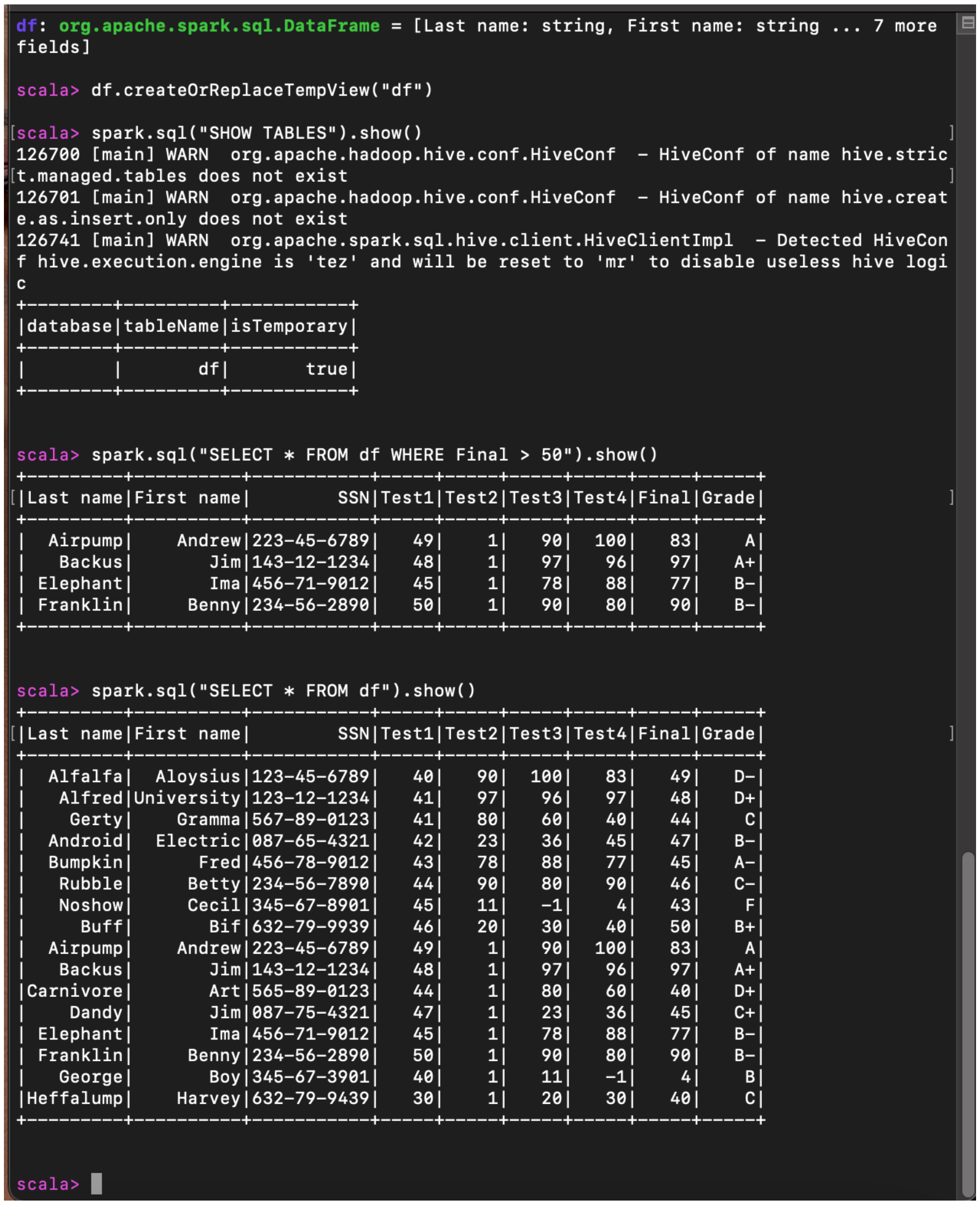
Objective 2 - SparkSQL with Scala (20 points):

2. SparkSQL with Scala

- Run the following SparkSQL commands in Scala:

```
val df = spark.read.format("csv").option("header",
"true").load("/data/grades.csv")
df.createOrReplaceTempView("df")

spark.sql("SHOW TABLES").show()
spark.sql("SELECT * FROM df WHERE Final > 50").show()
spark.sql("SELECT * FROM df").show()
```

Deliverable 1: Screenshot of the results from the provided SparkSQL commands in Scala, plus 1–2 sentences explaining what the commands did and what the output shows.

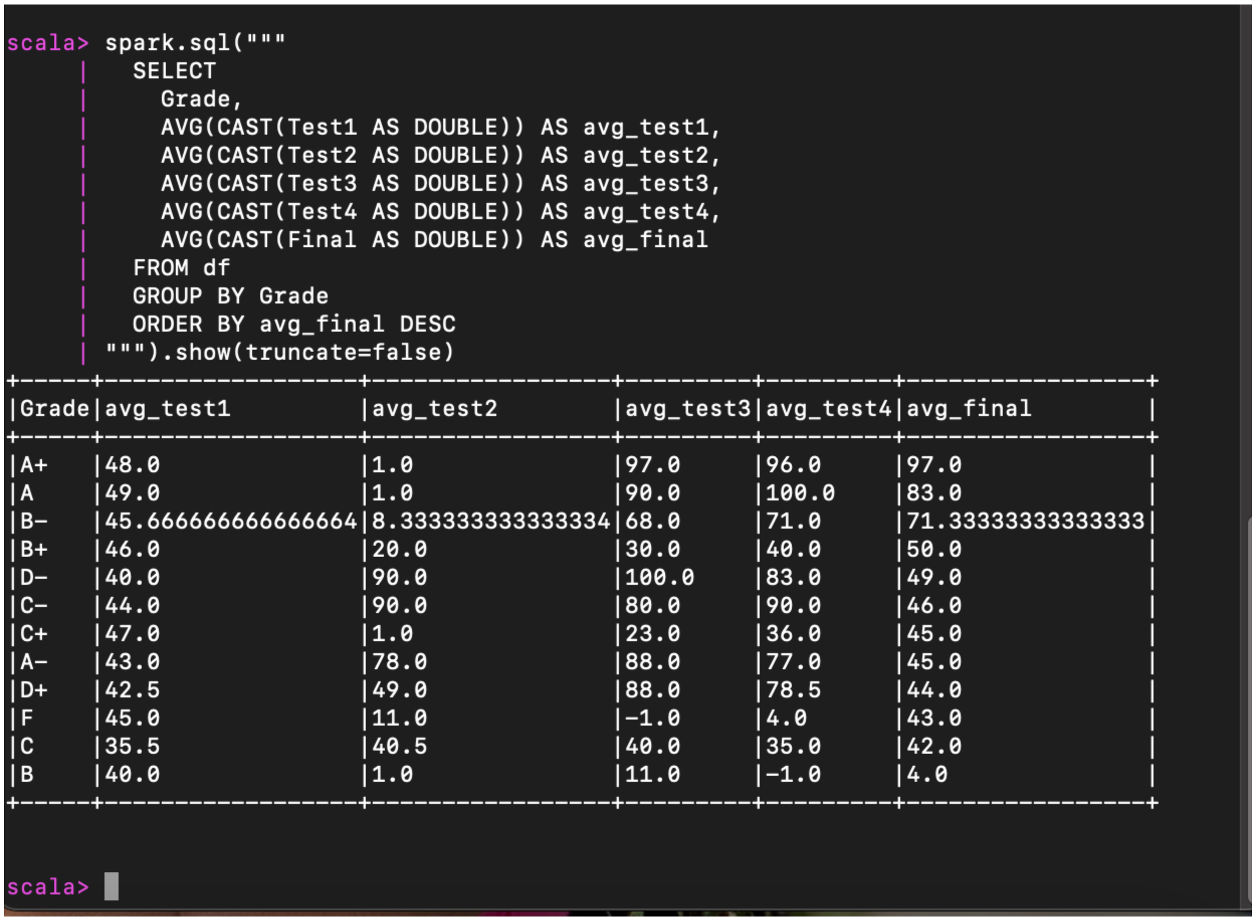


```
df: org.apache.spark.sql.DataFrame = [Last name: string, First name: string ... 7 more
fields]

scala> df.createOrReplaceTempView("df")

scala> spark.sql("SHOW TABLES").show()
126700 [main] WARN  org.apache.hadoop.hive.conf.HiveConf  - HiveConf of name hive.stric
t.managed.tables does not exist
126701 [main] WARN  org.apache.hadoop.hive.conf.HiveConf  - HiveConf of name hive.creat
e.as.insert.only does not exist
126741 [main] WARN  org.apache.spark.sql.hive.client.HiveClientImpl  - Detected HiveCon
f hive.execution.engine is 'tez' and will be reset to 'mr' to disable useless hive logi
c
+--------+---------+-----------+
|database|tableName|isTemporary|
+--------+---------+-----------+
|        |       df|       true|
+--------+---------+-----------+


scala> spark.sql("SELECT * FROM df WHERE Final > 50").show()
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+
|Last name|First name|        SSN|Test1|Test2|Test3|Test4|Final|Grade|
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+
|  Airpump|    Andrew|223-45-6789|   49|    1|   90|  100|   83|    A|
|   Backus|       Jim|143-12-1234|   48|    1|   97|   96|   97|   A+|
| Elephant|       Ima|456-71-9012|   45|    1|   78|   88|   77|   B-|
| Franklin|     Benny|234-56-2890|   50|    1|   90|   80|   90|   B-|
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+


scala> spark.sql("SELECT * FROM df").show()
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+
|Last name|First name|        SSN|Test1|Test2|Test3|Test4|Final|Grade|
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+
|  Alfalfa|  Aloysius|123-45-6789|   40|   90|  100|   83|   49|   D-|
|   Alfred|University|123-12-1234|   41|   97|   96|   97|   48|   D+|
|    Gerty|    Gramma|567-89-0123|   41|   80|   60|   40|   44|    C|
|  Android|  Electric|087-65-4321|   42|   23|   36|   45|   47|   B-|
|  Bumpkin|      Fred|456-78-9012|   43|   78|   88|   77|   45|   A-|
|   Rubble|     Betty|234-56-7890|   44|   90|   80|   90|   46|   C-|
|   Noshow|     Cecil|345-67-8901|   45|   11|   -1|    4|   43|    F|
|     Buff|       Bif|632-79-9939|   46|   20|   30|   40|   50|   B+|
|  Airpump|    Andrew|223-45-6789|   49|    1|   90|  100|   83|    A|
|   Backus|       Jim|143-12-1234|   48|    1|   97|   96|   97|   A+|
|Carnivore|       Art|565-89-0123|   44|    1|   80|   60|   40|   D+|
|    Dandy|       Jim|087-75-4321|   47|    1|   23|   36|   45|   C+|
| Elephant|       Ima|456-71-9012|   45|    1|   78|   88|   77|   B-|
| Franklin|     Benny|234-56-2890|   50|    1|   90|   80|   90|   B-|
|   George|       Boy|345-67-3901|   40|    1|   11|   -1|    4|    B|
|Heffalump|    Harvey|632-79-9439|   30|    1|   20|   30|   40|    C|
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+


scala> 
```

Explanation: The provided SparkSQL commands demonstrate how to load, register, and query a CSV file using **Spark SQL** within the Scala shell. The screenshot confirms that the dataset has been loaded and queried successfully. The filtered query (Final > 50) returns only students with passing final scores, demonstrating how SparkSQL can efficiently filter and display data using SQL commands on distributed datasets.

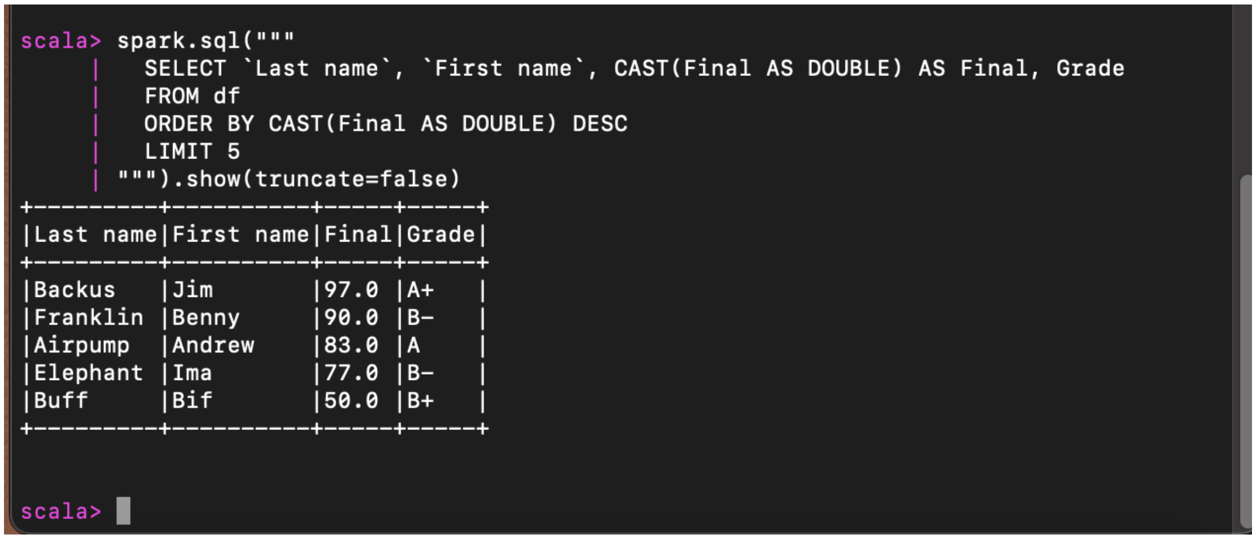
- Run 3 other SQL queries in the Spark Shell:

Deliverable 2: Screenshots of three additional SQL queries you wrote in Scala, each with its results and a short explanation of what the query does and what the output means.



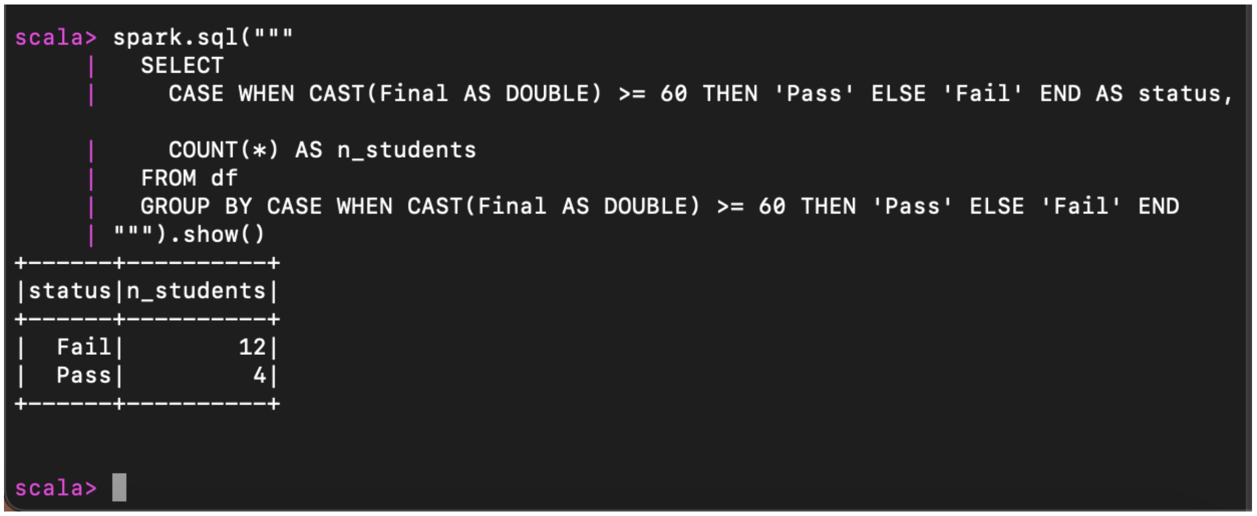
```
scala> spark.sql("""
     |   SELECT
     |     Grade,
     |     AVG(CAST(Test1 AS DOUBLE)) AS avg_test1,
     |     AVG(CAST(Test2 AS DOUBLE)) AS avg_test2,
     |     AVG(CAST(Test3 AS DOUBLE)) AS avg_test3,
     |     AVG(CAST(Test4 AS DOUBLE)) AS avg_test4,
     |     AVG(CAST(Final AS DOUBLE)) AS avg_final
     |   FROM df
     |   GROUP BY Grade
     |   ORDER BY avg_final DESC
     | """).show(truncate=false)
+-----+-----------------+-----------------+---------+---------+-----------------+
|Grade|avg_test1        |avg_test2        |avg_test3|avg_test4|avg_final        |
+-----+-----------------+-----------------+---------+---------+-----------------+
|A+   |48.0             |1.0              |97.0     |96.0     |97.0             |
|A    |49.0             |1.0              |90.0     |100.0    |83.0             |
|B-   |45.666666666666664|8.333333333333334|68.0     |71.0     |71.33333333333333|
|B+   |46.0             |20.0             |30.0     |40.0     |50.0             |
|D-   |40.0             |90.0             |100.0    |83.0     |49.0             |
|C-   |44.0             |90.0             |80.0     |90.0     |46.0             |
|C+   |47.0             |1.0              |23.0     |36.0     |45.0             |
|A-   |43.0             |78.0             |88.0     |77.0     |45.0             |
|D+   |42.5             |49.0             |88.0     |78.5     |44.0             |
|F    |45.0             |11.0             |-1.0     |4.0      |43.0             |
|C    |35.5             |40.5             |40.0     |35.0     |42.0             |
|B    |40.0             |1.0              |11.0     |-1.0     |4.0              |
+-----+-----------------+-----------------+---------+---------+-----------------+


scala> 
```

Explanation: Groups students by **Grade** and computes average Test1–Test4 and **Final**. You can see which letter grades have the highest/lowest average finals and how the test averages compare across grades.



```
scala> spark.sql("""
     |   SELECT `Last name`, `First name`, CAST(Final AS DOUBLE) AS Final, Grade
     |   FROM df
     |   ORDER BY CAST(Final AS DOUBLE) DESC
     |   LIMIT 5
     | """).show(truncate=false)
+---------+----------+-----+-----+
|Last name|First name|Final|Grade|
+---------+----------+-----+-----+
|Backus   |Jim       |97.0 |A+   |
|Franklin |Benny     |90.0 |B-   |
|Airpump  |Andrew    |83.0 |A    |
|Elephant |Ima       |77.0 |B-   |
|Buff     |Bif       |50.0 |B+   |
+---------+----------+-----+-----+


scala> 
```

Explanation: Sorts by numeric **Final** score and returns the top 5 rows. Identifies the strongest overall final scores and their letter grades.



```
scala> spark.sql("""
     |   SELECT
     |     CASE WHEN CAST(Final AS DOUBLE) >= 60 THEN 'Pass' ELSE 'Fail' END AS status,

     |     COUNT(*) AS n_students
     |   FROM df
     |   GROUP BY CASE WHEN CAST(Final AS DOUBLE) >= 60 THEN 'Pass' ELSE 'Fail' END
     | """).show()
+------+----------+
|status|n_students|
+------+----------+
|  Fail|        12|
|  Pass|         4|
+------+----------+


scala> 
```

Explanation: Buckets students into **Pass** (Final ≥ 60) and **Fail**. Quick class-level summary of how many passed vs. failed by this threshold.

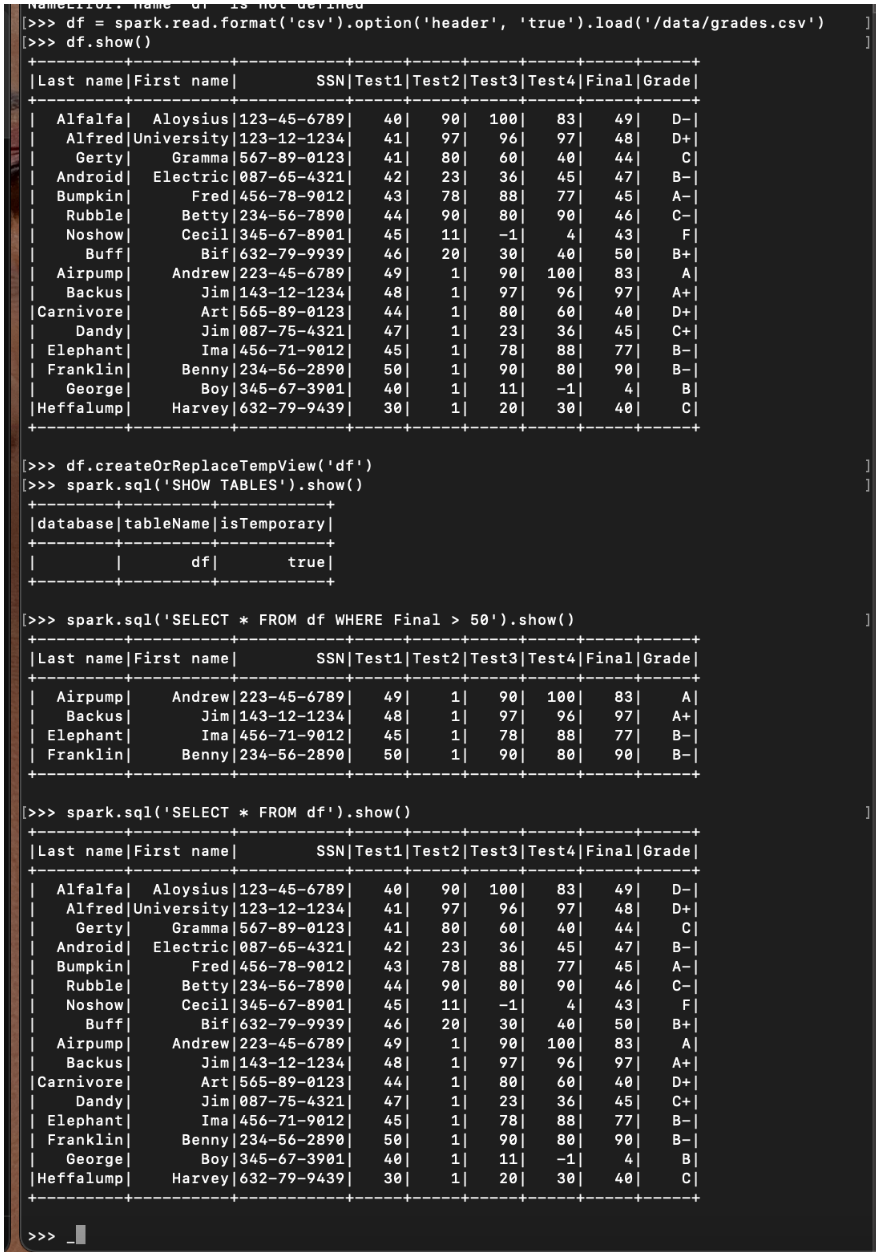
Objective 3 - SparkSQL with Python (20 points):

- Run the following SparkSQL commands in Python:

```
df = spark.read.format('csv').option('header',
'true').load('/data/grades.csv')
df.show()

df.createOrReplaceTempView('df')
```

```
spark.sql('SHOW TABLES').show()
spark.sql('SELECT * FROM df WHERE Final > 50').show()
spark.sql('SELECT * FROM df').show()
```

Deliverable 1: Screenshot of the results from the provided SparkSQL commands in Python, plus 1–2 sentences explaining what the commands did and what the output shows.



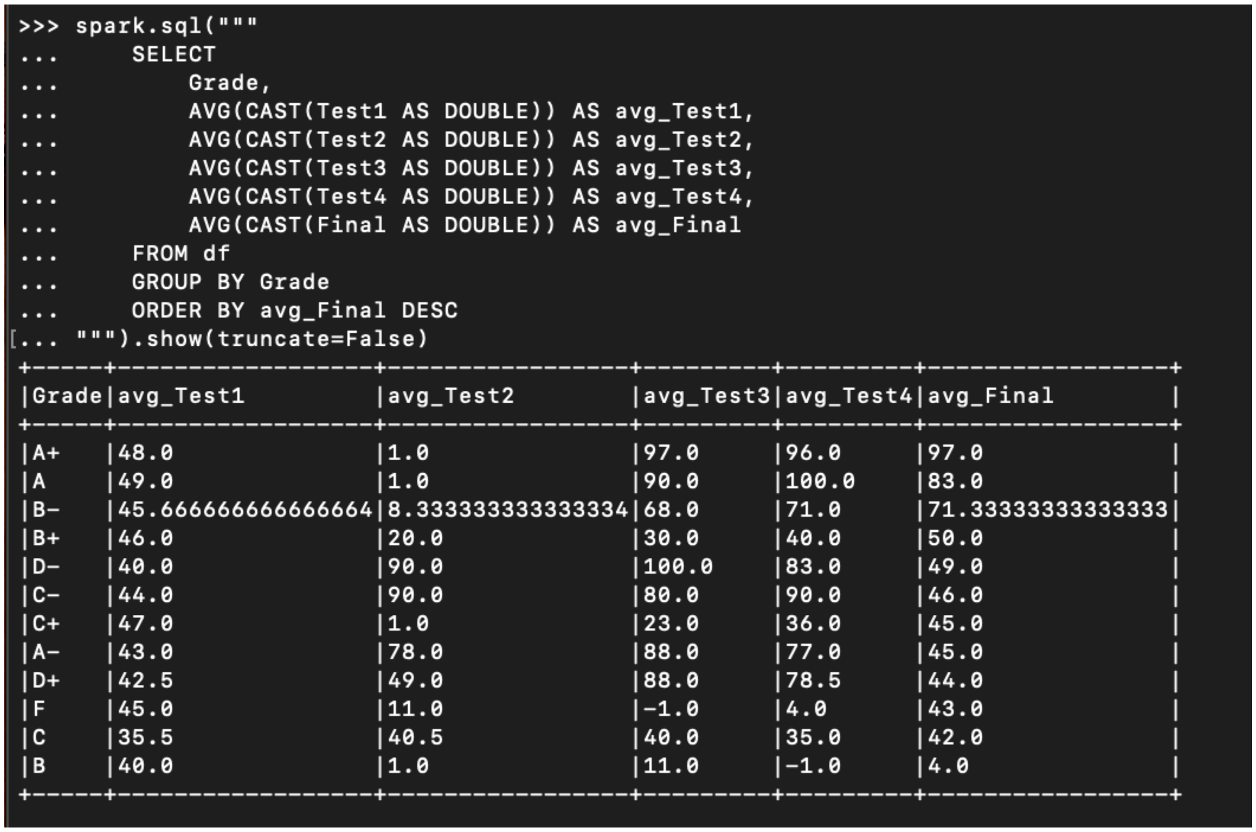
```
NameError: name 'df' is not defined
[>>> df = spark.read.format('csv').option('header', 'true').load('/data/grades.csv')    ]
[>>> df.show()                                                                          ]
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+
|Last name|First name|        SSN|Test1|Test2|Test3|Test4|Final|Grade|
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+
|  Alfalfa|  Aloysius|123-45-6789|   40|   90|  100|   83|   49|   D-|
|   Alfred|University|123-12-1234|   41|   97|   96|   97|   48|   D+|
|    Gerty|    Gramma|567-89-0123|   41|   80|   60|   40|   44|    C|
|  Android|  Electric|087-65-4321|   42|   23|   36|   45|   47|   B-|
|  Bumpkin|      Fred|456-78-9012|   43|   78|   88|   77|   45|   A-|
|   Rubble|     Betty|234-56-7890|   44|   90|   80|   90|   46|   C-|
|   Noshow|     Cecil|345-67-8901|   45|   11|   -1|    4|   43|    F|
|     Buff|       Bif|632-79-9939|   46|   20|   30|   40|   50|   B+|
|  Airpump|    Andrew|223-45-6789|   49|    1|   90|  100|   83|    A|
|   Backus|       Jim|143-12-1234|   48|    1|   97|   96|   97|   A+|
|Carnivore|       Art|565-89-0123|   44|    1|   80|   60|   40|   D+|
|    Dandy|       Jim|087-75-4321|   47|    1|   23|   36|   45|   C+|
| Elephant|       Ima|456-71-9012|   45|    1|   78|   88|   77|   B-|
| Franklin|     Benny|234-56-2890|   50|    1|   90|   80|   90|   B-|
|   George|       Boy|345-67-3901|   40|    1|   11|   -1|    4|    B|
|Heffalump|    Harvey|632-79-9439|   30|    1|   20|   30|   40|    C|
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+

[>>> df.createOrReplaceTempView('df')                                                   ]
[>>> spark.sql('SHOW TABLES').show()                                                    ]
+--------+---------+-----------+
|database|tableName|isTemporary|
+--------+---------+-----------+
|        |       df|       true|
+--------+---------+-----------+

[>>> spark.sql('SELECT * FROM df WHERE Final > 50').show()                              ]
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+
|Last name|First name|        SSN|Test1|Test2|Test3|Test4|Final|Grade|
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+
|  Airpump|    Andrew|223-45-6789|   49|    1|   90|  100|   83|    A|
|   Backus|       Jim|143-12-1234|   48|    1|   97|   96|   97|   A+|
| Elephant|       Ima|456-71-9012|   45|    1|   78|   88|   77|   B-|
| Franklin|     Benny|234-56-2890|   50|    1|   90|   80|   90|   B-|
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+

[>>> spark.sql('SELECT * FROM df').show()                                               ]
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+
|Last name|First name|        SSN|Test1|Test2|Test3|Test4|Final|Grade|
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+
|  Alfalfa|  Aloysius|123-45-6789|   40|   90|  100|   83|   49|   D-|
|   Alfred|University|123-12-1234|   41|   97|   96|   97|   48|   D+|
|    Gerty|    Gramma|567-89-0123|   41|   80|   60|   40|   44|    C|
|  Android|  Electric|087-65-4321|   42|   23|   36|   45|   47|   B-|
|  Bumpkin|      Fred|456-78-9012|   43|   78|   88|   77|   45|   A-|
|   Rubble|     Betty|234-56-7890|   44|   90|   80|   90|   46|   C-|
|   Noshow|     Cecil|345-67-8901|   45|   11|   -1|    4|   43|    F|
|     Buff|       Bif|632-79-9939|   46|   20|   30|   40|   50|   B+|
|  Airpump|    Andrew|223-45-6789|   49|    1|   90|  100|   83|    A|
|   Backus|       Jim|143-12-1234|   48|    1|   97|   96|   97|   A+|
|Carnivore|       Art|565-89-0123|   44|    1|   80|   60|   40|   D+|
|    Dandy|       Jim|087-75-4321|   47|    1|   23|   36|   45|   C+|
| Elephant|       Ima|456-71-9012|   45|    1|   78|   88|   77|   B-|
| Franklin|     Benny|234-56-2890|   50|    1|   90|   80|   90|   B-|
|   George|       Boy|345-67-3901|   40|    1|   11|   -1|    4|    B|
|Heffalump|    Harvey|632-79-9439|   30|    1|   20|   30|   40|    C|
+---------+----------+-----------+-----+-----+-----+-----+-----+-----+

>>> _
```

Explanation: The commands shown in the screenshot demonstrate how to load and query a CSV file using **Spark SQL** within **PySpark**. The output verifies that Spark successfully loaded and queried the dataset. The filtered results highlight only the students who passed

(Final > 50), while the full table output confirms all student data was correctly read from the CSV file and made accessible through Spark SQL.

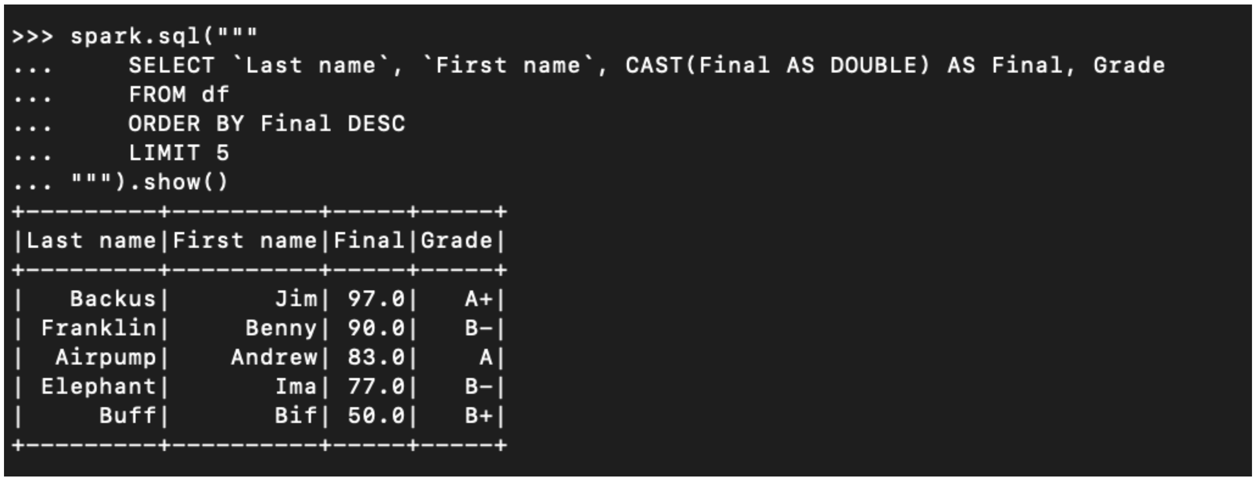
- Run 3 other SQL queries in the PySpark Shell

  Deliverable 2: Screenshots of three additional SQL queries you wrote in PySpark, each with its results and a short explanation of what the query does and what the output means.



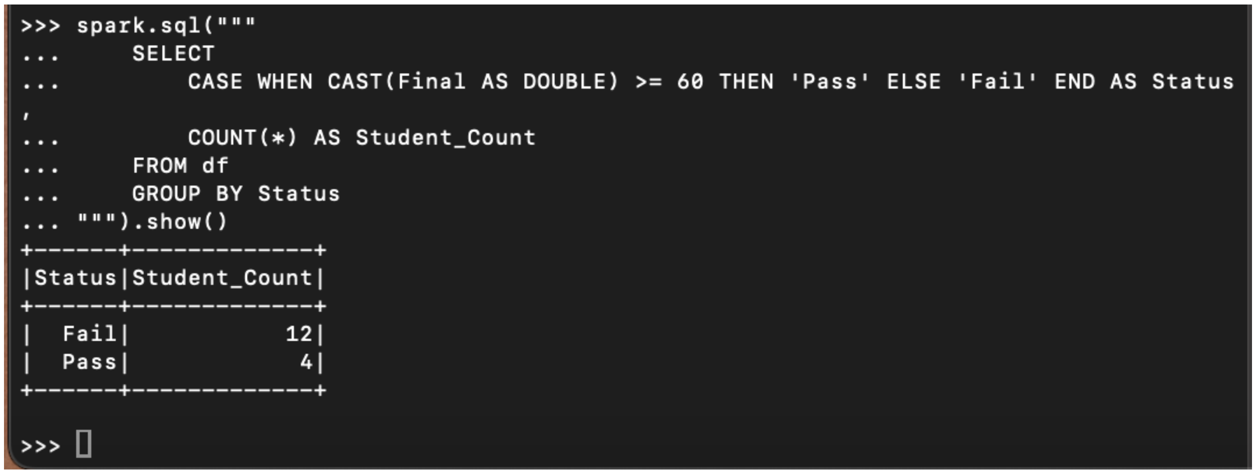
```
>>> spark.sql("""
...     SELECT
...         Grade,
...         AVG(CAST(Test1 AS DOUBLE)) AS avg_Test1,
...         AVG(CAST(Test2 AS DOUBLE)) AS avg_Test2,
...         AVG(CAST(Test3 AS DOUBLE)) AS avg_Test3,
...         AVG(CAST(Test4 AS DOUBLE)) AS avg_Test4,
...         AVG(CAST(Final AS DOUBLE)) AS avg_Final
...     FROM df
...     GROUP BY Grade
...     ORDER BY avg_Final DESC
[... """).show(truncate=False)
+-----+------------------+-----------------+---------+---------+-----------------+
|Grade|avg_Test1         |avg_Test2        |avg_Test3|avg_Test4|avg_Final        |
+-----+------------------+-----------------+---------+---------+-----------------+
|A+   |48.0              |1.0              |97.0     |96.0     |97.0             |
|A    |49.0              |1.0              |90.0     |100.0    |83.0             |
|B-   |45.666666666666664|8.333333333333334|68.0     |71.0     |71.33333333333333|
|B+   |46.0              |20.0             |30.0     |40.0     |50.0             |
|D-   |40.0              |90.0             |100.0    |83.0     |49.0             |
|C-   |44.0              |90.0             |80.0     |90.0     |46.0             |
|C+   |47.0              |1.0              |23.0     |36.0     |45.0             |
|A-   |43.0              |78.0             |88.0     |77.0     |45.0             |
|D+   |42.5              |49.0             |88.0     |78.5     |44.0             |
|F    |45.0              |11.0             |-1.0     |4.0      |43.0             |
|C    |35.5              |40.5             |40.0     |35.0     |42.0             |
|B    |40.0              |1.0              |11.0     |-1.0     |4.0              |
+-----+------------------+-----------------+---------+---------+-----------------+
```

Explanation: This query groups students by their **Grade** and calculates the average scores across all tests. It shows how each letter grade corresponds to average performance—helpful for seeing which grades have the strongest or weakest test averages.



```
>>> spark.sql("""
...     SELECT `Last name`, `First name`, CAST(Final AS DOUBLE) AS Final, Grade
...     FROM df
...     ORDER BY Final DESC
...     LIMIT 5
... """).show()
+---------+----------+-----+-----+
|Last name|First name|Final|Grade|
+---------+----------+-----+-----+
|   Backus|       Jim| 97.0|   A+|
| Franklin|     Benny| 90.0|   B-|
|  Airpump|    Andrew| 83.0|    A|
| Elephant|       Ima| 77.0|   B-|
|     Buff|       Bif| 50.0|   B+|
+---------+----------+-----+-----+
```

Explanation: This query sorts students by their **Final** score (converted to numeric) in descending order and returns only the top 5. It identifies the **highest-performing students** in the dataset based on their final exam scores.

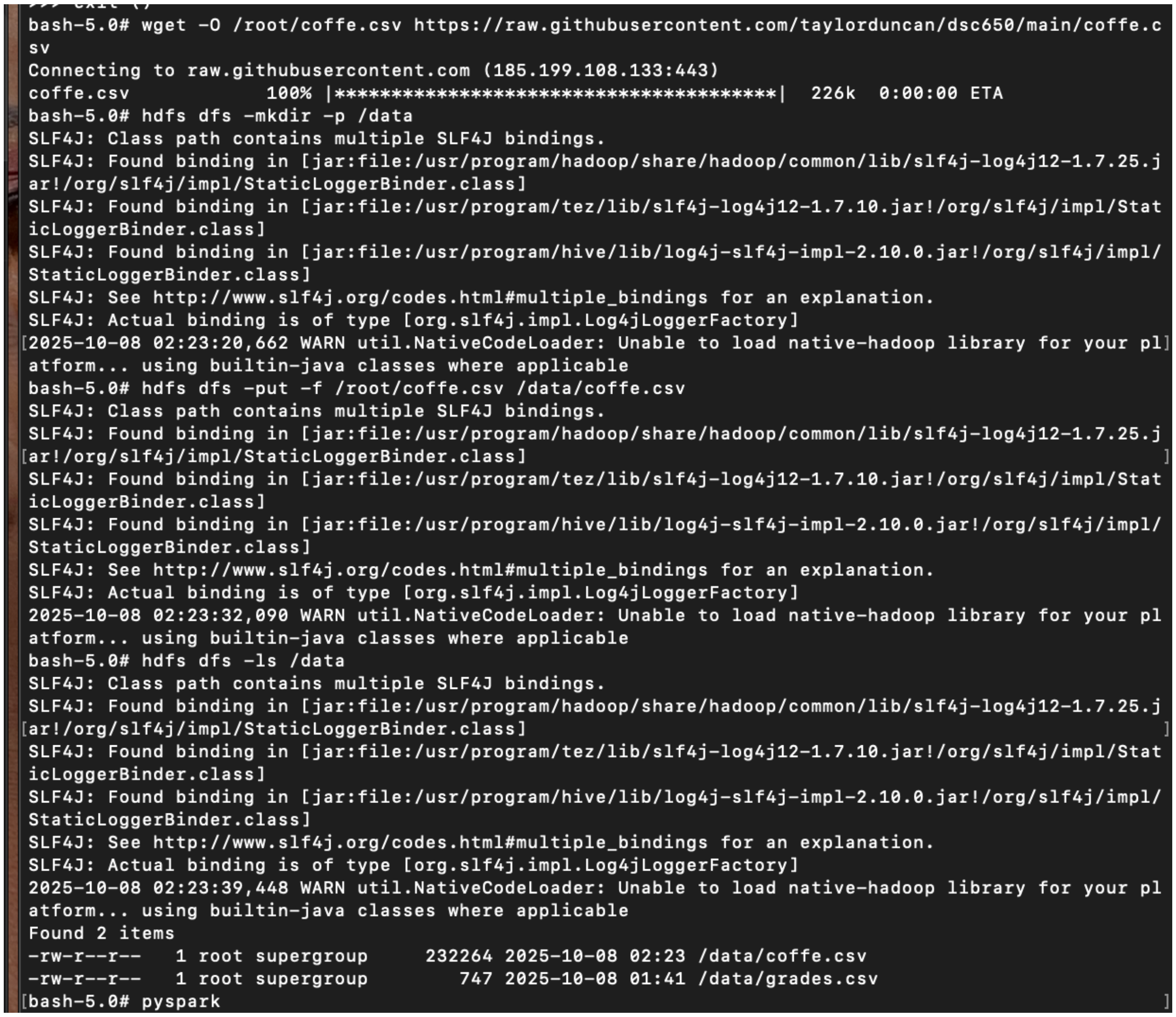


```
>>> spark.sql("""
...     SELECT
...         CASE WHEN CAST(Final AS DOUBLE) >= 60 THEN 'Pass' ELSE 'Fail' END AS Status
,
...         COUNT(*) AS Student_Count
...     FROM df
...     GROUP BY Status
... """).show()
+------+-------------+
|Status|Student_Count|
+------+-------------+
|  Fail|           12|
|  Pass|            4|
+------+-------------+

>>> ▯
```

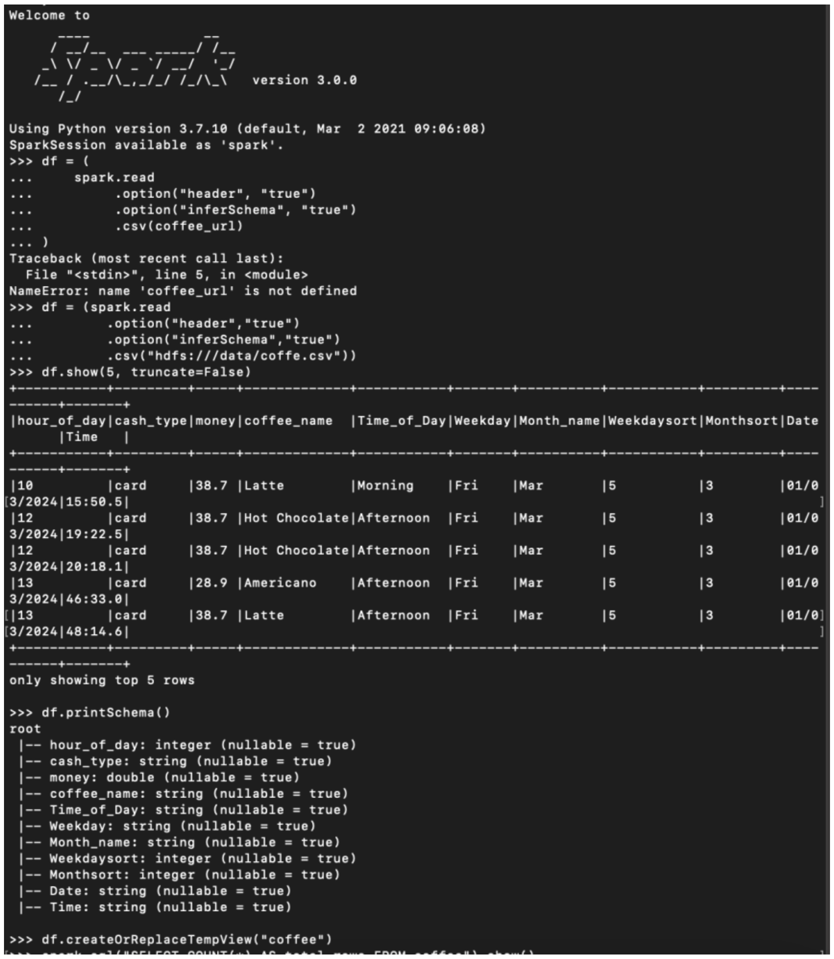
Explanation: This query classifies each student as **Pass** or **Fail** depending on whether their Final ≥ 60, then counts how many fall in each category. It provides a simple **class performance breakdown**, showing how many students passed versus failed.

Objective 4 - SparkSQL with Custom Dataset (40 points):

Deliverable 1: Your Scala or PySpark code used to load and query the custom dataset, plus 2–3 sentences explaining what the code does and how you confirmed it worked.



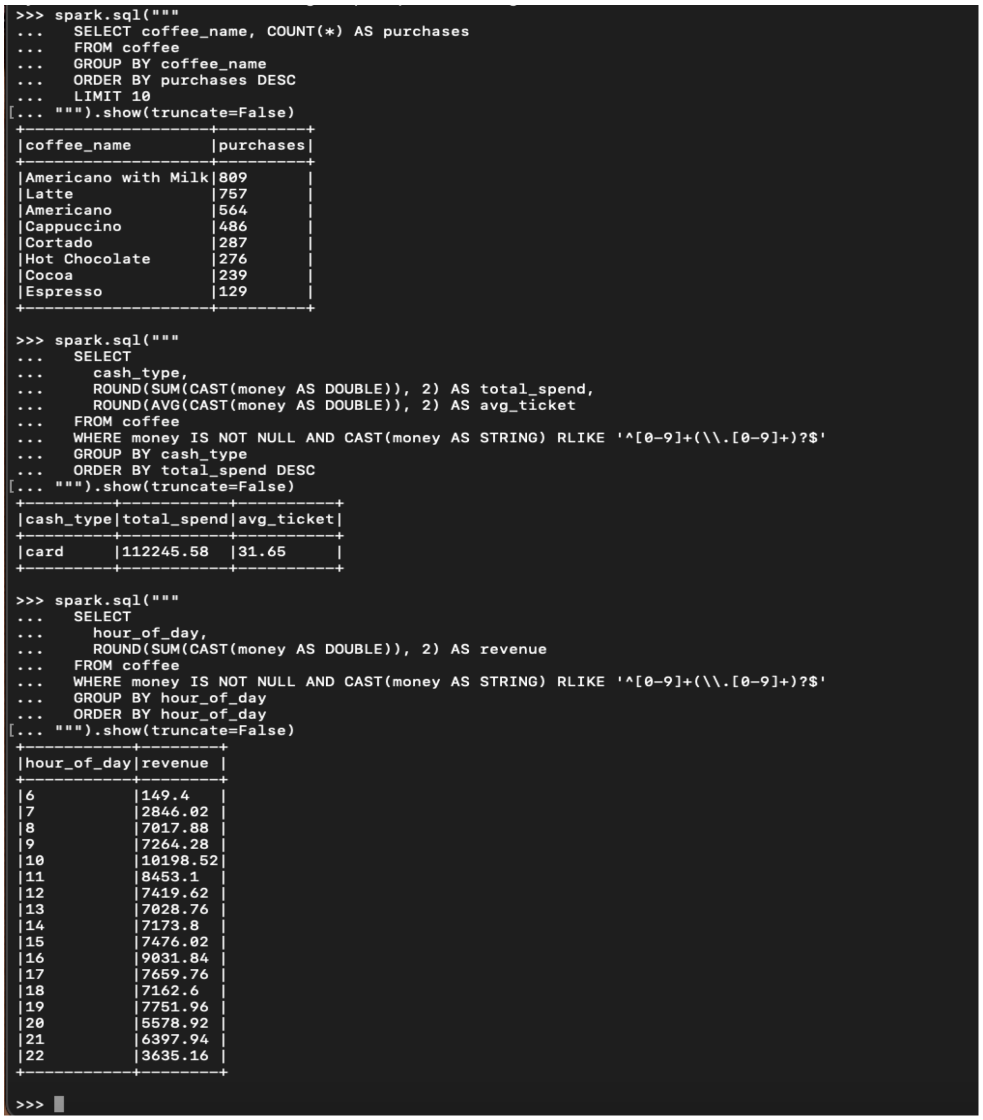
```
>>> exit()
bash-5.0# wget -O /root/coffe.csv https://raw.githubusercontent.com/taylorduncan/dsc650/main/coffe.c
sv
Connecting to raw.githubusercontent.com (185.199.108.133:443)
coffe.csv            100% |***************************************|  226k  0:00:00 ETA
bash-5.0# hdfs dfs -mkdir -p /data
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.j
ar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/Stat
icLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/
StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2025-10-08 02:23:20,662 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your pl
atform... using builtin-java classes where applicable
bash-5.0# hdfs dfs -put -f /root/coffe.csv /data/coffe.csv
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.j
ar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/Stat
icLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/
StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2025-10-08 02:23:32,090 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your pl
atform... using builtin-java classes where applicable
bash-5.0# hdfs dfs -ls /data
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/program/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.j
ar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/Stat
icLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/program/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/
StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2025-10-08 02:23:39,448 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your pl
atform... using builtin-java classes where applicable
Found 2 items
-rw-r--r--   1 root supergroup     232264 2025-10-08 02:23 /data/coffe.csv
-rw-r--r--   1 root supergroup        747 2025-10-08 01:41 /data/grades.csv
bash-5.0# pyspark
```



```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.0
      /_/

Using Python version 3.7.10 (default, Mar  2 2021 09:06:08)
SparkSession available as 'spark'.
>>> df = (
...     spark.read
...          .option("header", "true")
...          .option("inferSchema", "true")
...          .csv(coffee_url)
... )
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
NameError: name 'coffee_url' is not defined
>>> df = (spark.read
...         .option("header","true")
...         .option("inferSchema","true")
...         .csv("hdfs:///data/coffe.csv"))
>>> df.show(5, truncate=False)
+-----------+---------+-----+-------------+-----------+-------+----------+-----------+---------+----
------+-------+
|hour_of_day|cash_type|money|coffee_name  |Time_of_Day|Weekday|Month_name|Weekdaysort|Monthsort|Date
      |Time   |
+-----------+---------+-----+-------------+-----------+-------+----------+-----------+---------+----
------+-------+
|10         |card     |38.7 |Latte        |Morning    |Fri    |Mar       |5          |3        |01/0
3/2024|15:50.5|
|12         |card     |38.7 |Hot Chocolate|Afternoon  |Fri    |Mar       |5          |3        |01/0
3/2024|19:22.5|
|12         |card     |38.7 |Hot Chocolate|Afternoon  |Fri    |Mar       |5          |3        |01/0
3/2024|20:18.1|
|13         |card     |28.9 |Americano    |Afternoon  |Fri    |Mar       |5          |3        |01/0
3/2024|46:33.0|
|13         |card     |38.7 |Latte        |Afternoon  |Fri    |Mar       |5          |3        |01/0
3/2024|48:14.6|
+-----------+---------+-----+-------------+-----------+-------+----------+-----------+---------+----
------+-------+
only showing top 5 rows

>>> df.printSchema()
root
 |-- hour_of_day: integer (nullable = true)
 |-- cash_type: string (nullable = true)
 |-- money: double (nullable = true)
 |-- coffee_name: string (nullable = true)
 |-- Time_of_Day: string (nullable = true)
 |-- Weekday: string (nullable = true)
 |-- Month_name: string (nullable = true)
 |-- Weekdaysort: integer (nullable = true)
 |-- Monthsort: integer (nullable = true)
 |-- Date: string (nullable = true)
 |-- Time: string (nullable = true)

>>> df.createOrReplaceTempView("coffee")
```

- The code loads the **Coffee dataset** directly from GitHub into Spark using spark.read.csv().
- It registers the DataFrame as a **temporary SQL view (coffee)** so SQL queries can be executed directly on the data.
- The .show() results confirm the queries executed properly and the dataset is active in Spark.

Deliverable 2: Screenshots of three SQL query results on your dataset, each with a short explanation of what the query is doing and what the output shows.



```
>>> spark.sql("""
...   SELECT coffee_name, COUNT(*) AS purchases
...   FROM coffee
...   GROUP BY coffee_name
...   ORDER BY purchases DESC
...   LIMIT 10
[... """).show(truncate=False)
+-------------------+---------+
|coffee_name        |purchases|
+-------------------+---------+
|Americano with Milk|809      |
|Latte              |757      |
|Americano          |564      |
|Cappuccino         |486      |
|Cortado            |287      |
|Hot Chocolate      |276      |
|Cocoa              |239      |
|Espresso           |129      |
+-------------------+---------+

>>> spark.sql("""
...   SELECT
...     cash_type,
...     ROUND(SUM(CAST(money AS DOUBLE)), 2) AS total_spend,
...     ROUND(AVG(CAST(money AS DOUBLE)), 2) AS avg_ticket
...   FROM coffee
...   WHERE money IS NOT NULL AND CAST(money AS STRING) RLIKE '^[0-9]+(\\.[0-9]+)?$'
...   GROUP BY cash_type
...   ORDER BY total_spend DESC
[... """).show(truncate=False)
+---------+-----------+----------+
|cash_type|total_spend|avg_ticket|
+---------+-----------+----------+
|card     |112245.58  |31.65     |
+---------+-----------+----------+

>>> spark.sql("""
...   SELECT
...     hour_of_day,
...     ROUND(SUM(CAST(money AS DOUBLE)), 2) AS revenue
...   FROM coffee
...   WHERE money IS NOT NULL AND CAST(money AS STRING) RLIKE '^[0-9]+(\\.[0-9]+)?$'
...   GROUP BY hour_of_day
...   ORDER BY hour_of_day
[... """).show(truncate=False)
+-----------+--------+
|hour_of_day|revenue |
+-----------+--------+
|6          |149.4   |
|7          |2846.02 |
|8          |7017.88 |
|9          |7264.28 |
|10         |10198.52|
|11         |8453.1  |
|12         |7419.62 |
|13         |7028.76 |
|14         |7173.8  |
|15         |7476.02 |
|16         |9031.84 |
|17         |7659.76 |
|18         |7162.6  |
|19         |7751.96 |
|20         |5578.92 |
|21         |6397.94 |
|22         |3635.16 |
+-----------+--------+

>>>
```

Top 10 coffee items by number of purchases

Explanation: Counts rows per coffee_name to find the most-ordered items.
Output meaning: Your top-selling drinks by frequency.

Total and average spend by payment type

Explanation: Aggregates money by cash_type (e.g., cash vs. card).
Output meaning: Which payment types drive the most revenue and their average ticket size.

Hourly revenue profile (what hours earn the most)

Explanation: Sums spend per hour_of_day.
Output meaning: Your sales curve over the day—useful to identify peak hours.