Spark Task Report

Dataset Used

The dataset used in my task implementation represents student performance in exams, it mainly contains the data of 1000 students and their performance in three exams, math exams, reading exams, and writing exams. Along with the dataset, there are some attributes to the students that represent another 5 columns in the table. These attributes are the students' gender, their race, their parental level of education, whether they ate before the exam, and whether they took a test preparation course. You can see the full dataset through this kaggle link here.

In my analysis, I was mainly interested in determining which attribute that would affect the student's performance in the exam. In other words, which attributes would contribute to the success or failure of the student in the exam. I also chose the scores of the students in the math exam to apply my analysis on. So from now on, we will be only interested in the performance of students in the math exam specifically.

Before we jump into the analysis implementation and results, the two images below show the schema of the data, and its columns representation

```
root
 |-- gender: string (nullable = true)
 |-- race/ethnicity: string (nullable = true)
 |-- parental level of education: string (nullable = true)
 |-- lunch: string (nullable = true)
 |-- test preparation course: string (nullable = true)
 |-- math score: integer (nullable = true)
 |-- reading score: integer (nullable = true)
 |-- writing score: integer (nullable = true)
```

gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
female	group B	bachelor's degree	standard	none	72	72	74
female	group C	some college	standard	completed	69	90	88
female	group B	master's degree	standard	none	90	95	93
male	group A	associate's degree	free/reduced	none	47	57	44
male	group C	some college	standard	none	76	78	75
female	group B	associate's degree	standard	none	71	83	78
female	group B	some college	standard	completed	88	95	92
male	group B	some college	free/reduced	none	40	43	39
male	group D	high school	free/reduced	completed	64	64	67
female	group B	high school	free/reduced	none	38	60	50
male	group C	associate's degree	standard	none	58	54	52
male	group D	associate's degree	standard	none	40	52	43
female	group B	high school	standard	none	65	81	73

I'll attach the full output of the spark application, along with the spark code in the email. However, for the rest of the document, I'll lay down the lines of code used to output the results that will be shown.

I wanted to know how many students passed the exam and the percentage of both success and failure from the total number of students. I figured that I'd compute the percentage a lot

throughout the code, so I implemented a small function that would take a DataFrame, that has a specific form, and it would compute from it the percentage.

```
def computeEachGroupPercentage (DF : DataFrame, totalNumber : Long) : Unit = {
  DF.foreach ((row) => {

    var groupName = ""
    for (i <- 0 to row.length - 2) {
      val name = row.get (i).toString ()

      if (name == "true") {
        groupName += "passed "
      } else if (name == "false") {
        groupName += "failed "
      } else {
        groupName += name + " "
      }
    }

    println ("Percentage of " + groupName + "is: " + (row.getLong (row.length - 1) * 1.0 / totalNumber))
  })
}
```

First, I read the StudentsPerformance.csv file, then I grouped the data by whether a student got a score more than or equal 50. I passed the resulted dataframe to the mentioned function, and it printed the percentage of both the students who succeeded, and the students who failed.

Here's the code, and the output

```
1.      val studentsPerformanceDF = spark.read
2.        .option ("header", "true")
3.        .option ("inferSchema", "true")
4.        .csv ("StudentsPerformance.csv")
5.        .cache ()
6.
7.      studentsPerformanceDF.printSchema ()
8.      studentsPerformanceDF.show ()
9.
10.       val totalNumberOfStudents = studentsPerformanceDF.count ()
11.       println ("Total number of student records is: " +
totalNumberOfStudents)
12.
13.       // I'm going to extract just the students who passed their
Math Exams to make my analysis on them
14.       val passedStudents = studentsPerformanceDF.select
(studentsPerformanceDF ("math score") >= 50).groupBy ("(math score >=
50)").count ()
15.       passedStudents.show ()
16.       computeEachGroupPercentage (passedStudents,
totalNumberOfStudents)
```

Here's the output

```
Total number of student records is: 1000
+------------------+-----+
|(math score >= 50)|count|
+------------------+-----+
|              true|  865|
|             false|  135|
+------------------+-----+

Percentage of passed is: 0.865
Percentage of failed is: 0.135
```

Attributes Contribution

Gender Attribute

I wanted to see how much effect yielded by the gender, but it turned out that the gender won't affect the results that much in the dataset. Both males, and females had approximately the same percentage of success, or failure as shown in the graph below.

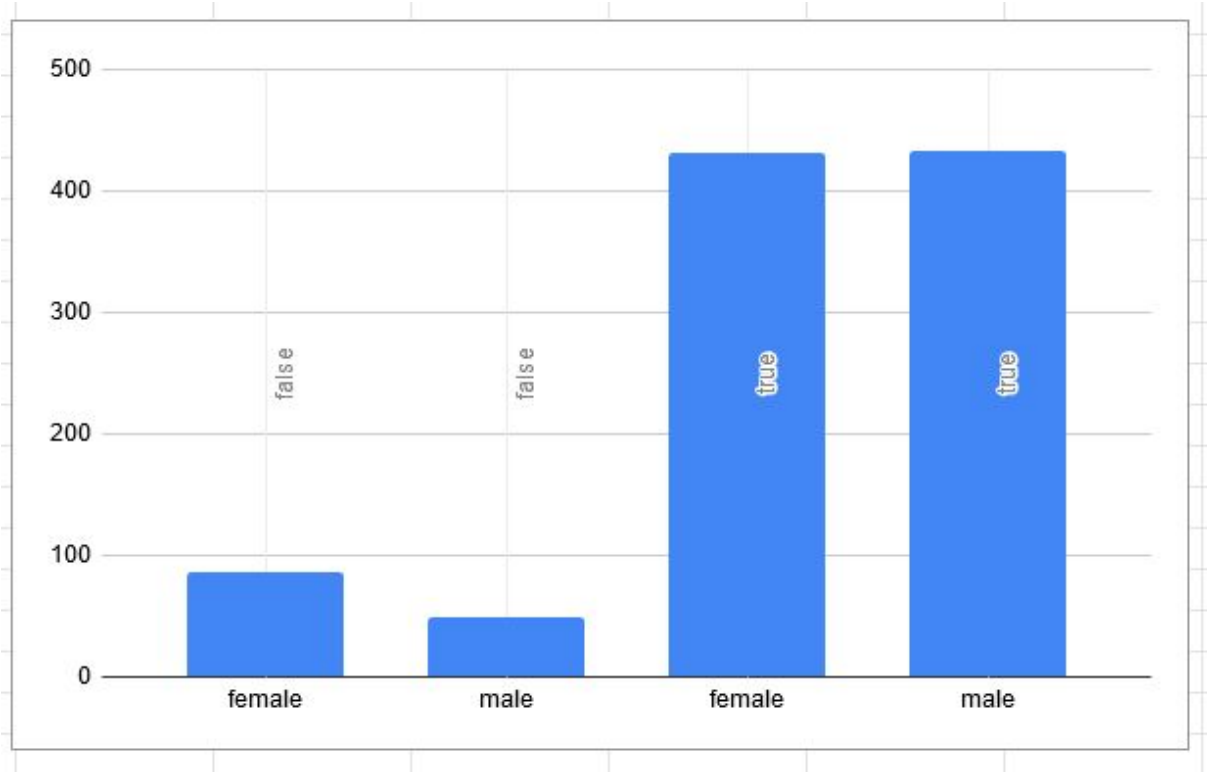
In the code below, I've grouped the students into 4 groups, the first one represents the number of male students who passed the exam, the second one represents the number of female students who passed the exam, the third one represents the number of male students who failed the exam, and the last one represents the number of female students who failed the exam. Here's the implementation

```
    // Relation between Gender and Math Performance
    val studentsGroupedByGender = studentsPerformanceDF
      .select (studentsPerformanceDF ("math score") >= 50, studentsPerformanceDF ("gender"))
      .groupBy ("gender", "(math score >= 50)")
      .count ()

    studentsGroupedByGender.write.csv ("StudentsGroupedByGender.csv")
    studentsGroupedByGender.show ()
    computeEachGroupPercentage (studentsGroupedByGender, totalNumberOfStudents)
```

The above code resulted in the following output

```
+------+-----------------+-----+
|gender|(math score >= 50)|count|
+------+-----------------+-----+
|  male|            false|   49|
|female|             true|  432|
|female|            false|   86|
|  male|             true|  433|
+------+-----------------+-----+

Percentage of male failed is: 0.049
Percentage of female passed is: 0.432
Percentage of female failed is: 0.086
Percentage of male passed is: 0.433
```

Here is a visualization of the outputted csv files as a column graph



Race/Ethnicity Attribute

Below is the code used to observe the relation between the ethnicity and the students' performance

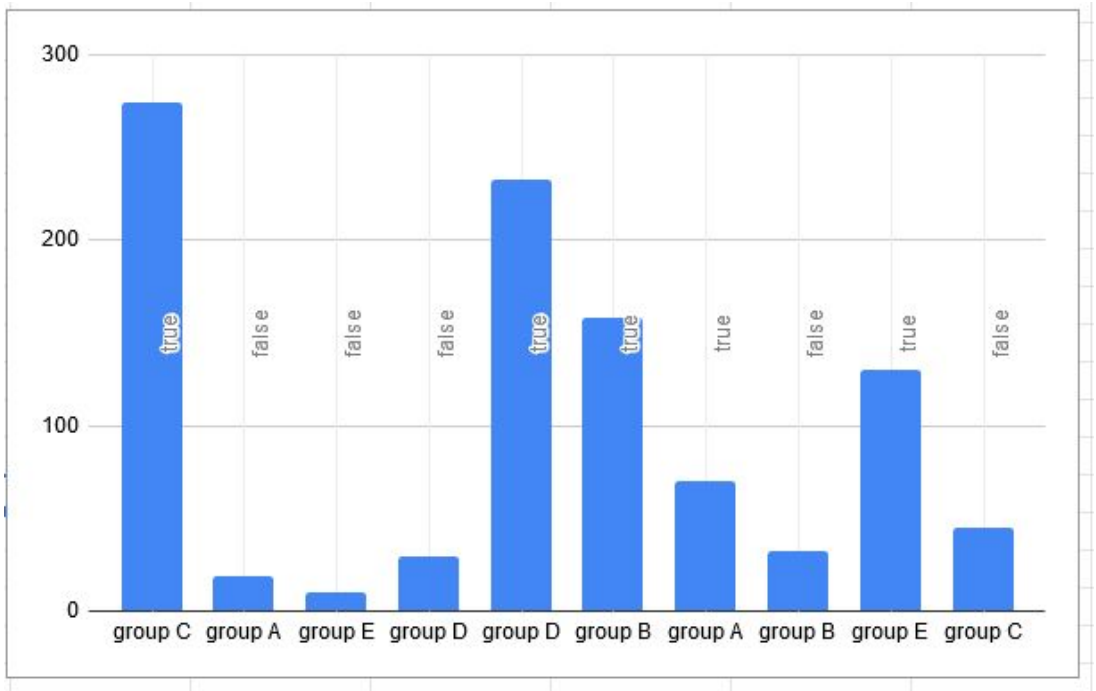
```
1.      // Relation between Race and Math Performance
2.      val studentsGroupedByRace = studentsPerformanceDF
```

```
        .select (studentsPerformanceDF ("math score") >= 50, studentsPerformanceDF ("race/ethnicity"))
        .groupBy ("race/ethnicity", "(math score >= 50)")
        .count ()

    studentsGroupedByRace.write.csv ("StudentsGroupedByRace.csv")
    studentsGroupedByRace.show ()
    computeEachGroupPercentage (studentsGroupedByRace, totalNumberOfStudents)
```

Here's the output

```
+--------------+-----------------+-----+
|race/ethnicity|(math score >= 50)|count|
+--------------+-----------------+-----+
|       group C|             true|  274|
|       group A|            false|   19|
|       group E|            false|   10|
|       group D|            false|   29|
|       group D|             true|  233|
|       group B|             true|  158|
|       group A|             true|   70|
|       group B|            false|   32|
|       group E|             true|  130|
|       group C|            false|   45|
+--------------+-----------------+-----+

Percentage of group C passed is: 0.274
Percentage of group A failed is: 0.019
Percentage of group E failed is: 0.01
Percentage of group D failed is: 0.029
Percentage of group D passed is: 0.233
Percentage of group B passed is: 0.158
Percentage of group A passed is: 0.07
Percentage of group B failed is: 0.032
Percentage of group E passed is: 0.13
Percentage of group C failed is: 0.045
```

Here's the graph of the outputted csv files



I think, from the previous data, we would say that "group C", and "group D" races contribute the most to the efficiency of the students. Most of the students from the groups have passed, and a few of them failed. So we can say that they can play a role in determining the success/failure of the student

Lunch Attribute

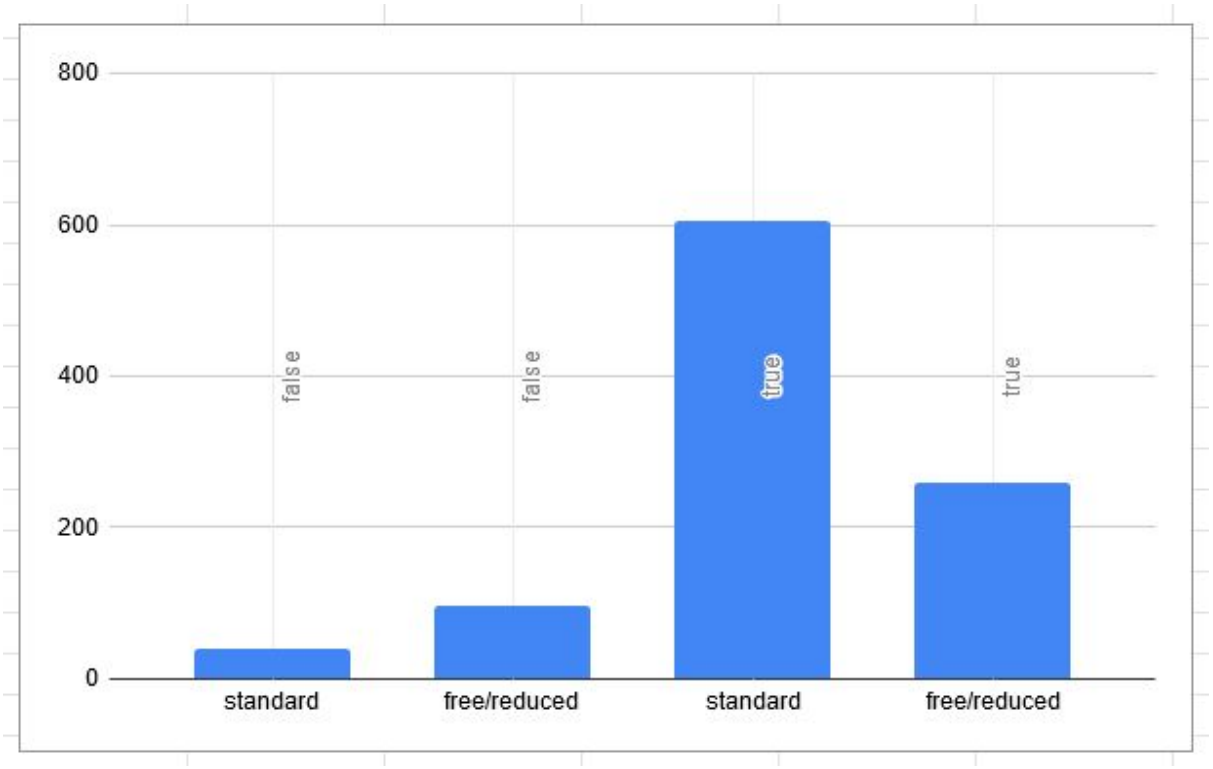
Here's the code to observe the relation between the lunch attribute and the students performance in math

```
    // Relation between Lunch and Math Performance
    val studentsGroupedByLunch = studentsPerformanceDF
      .select (studentsPerformanceDF ("math score") >= 50, studentsPerformanceDF ("lunch"))
      .groupBy ("lunch", "(math score >= 50)")
      .count ()

    studentsGroupedByLunch.write.csv ("StudentsGroupedByLunch.csv")
    studentsGroupedByLunch.show ()
    computeEachGroupPercentage (studentsGroupedByLunch, totalNumberOfStudents)
```

Here's the output of the previous snippet

```
+------------+------------------+-----+
|       lunch|(math score >= 50)|count|
+------------+------------------+-----+
|    standard|             false|   39|
|free/reduced|             false|   96|
|    standard|              true|  606|
|free/reduced|              true|  259|
+------------+------------------+-----+

Percentage of standard failed is: 0.039
Percentage of free/reduced failed is: 0.096
Percentage of standard passed is: 0.606
Percentage of free/reduced passed is: 0.259
```

Here's the graph



Most of the students who ate their standard meal succeeded in the exam, only a small portion of them failed. So we could say that it plays a role too in determining the success/failure of the students.

Parental Level of Education

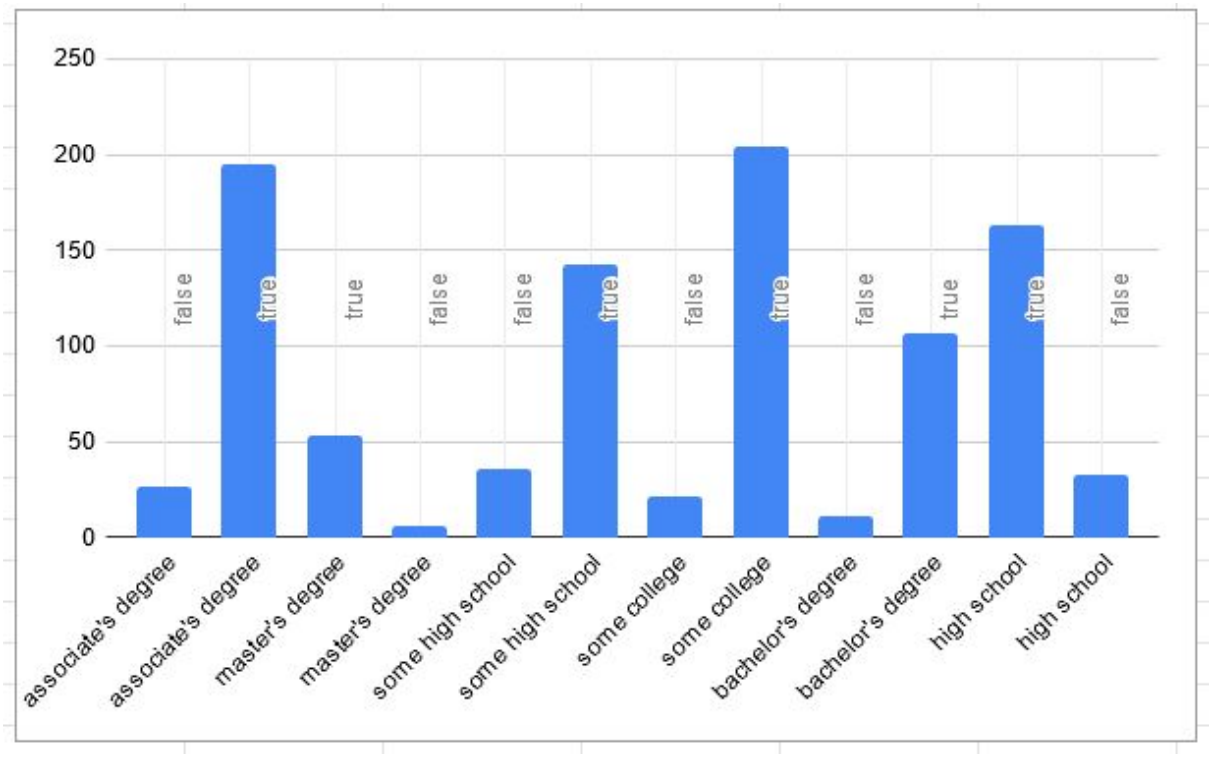
Here's the code to compute observe the relation between the parental level of education and the student performance in math

```
    // Relation between Parental Education Level and Math Performance
    val studentsGroupedByParentalEducation = studentsPerformanceDF
      .select (studentsPerformanceDF ("parental level of education"), studentsPerformanceDF ("math score") >= 50)
      .groupBy ("parental level of education", "(math score >= 50)")
      .count ()

    studentsGroupedByParentalEducation.write.csv ("StudentsGroupedByParentalEducation.csv")
    studentsGroupedByParentalEducation.show ()
    computeEachGroupPercentage (studentsGroupedByParentalEducation, totalNumberOfStudents)
```

Here's the output of the code

```
+---------------------------+-----------------+-----+
|parental level of education|(math score >= 50)|count|
+---------------------------+-----------------+-----+
|         associate's degree|            false|   27|
|            master's degree|             true|   53|
|         associate's degree|             true|  195|
|           some high school|            false|   36|
|               some college|            false|   22|
|          bachelor's degree|            false|   11|
|                high school|             true|  163|
|          bachelor's degree|             true|  107|
|            master's degree|            false|    6|
|                high school|            false|   33|
|               some college|             true|  204|
|           some high school|             true|  143|
+---------------------------+-----------------+-----+

Percentage of associate's degree failed is: 0.027
Percentage of master's degree passed is: 0.053
Percentage of associate's degree passed is: 0.195
Percentage of some high school failed is: 0.036
Percentage of some college failed is: 0.022
Percentage of bachelor's degree failed is: 0.011
Percentage of high school passed is: 0.163
Percentage of bachelor's degree passed is: 0.107
Percentage of master's degree failed is: 0.006
Percentage of high school failed is: 0.033
Percentage of some college passed is: 0.204
Percentage of some high school passed is: 0.143
```

Here's the graph of the output



Most of the success/failure percentages between the different education levels are approximately near to each other. I don't think we can deduce from this graph that any student with a parent of a specific educational degree has a higher chance than other students.

Test Preparation Course

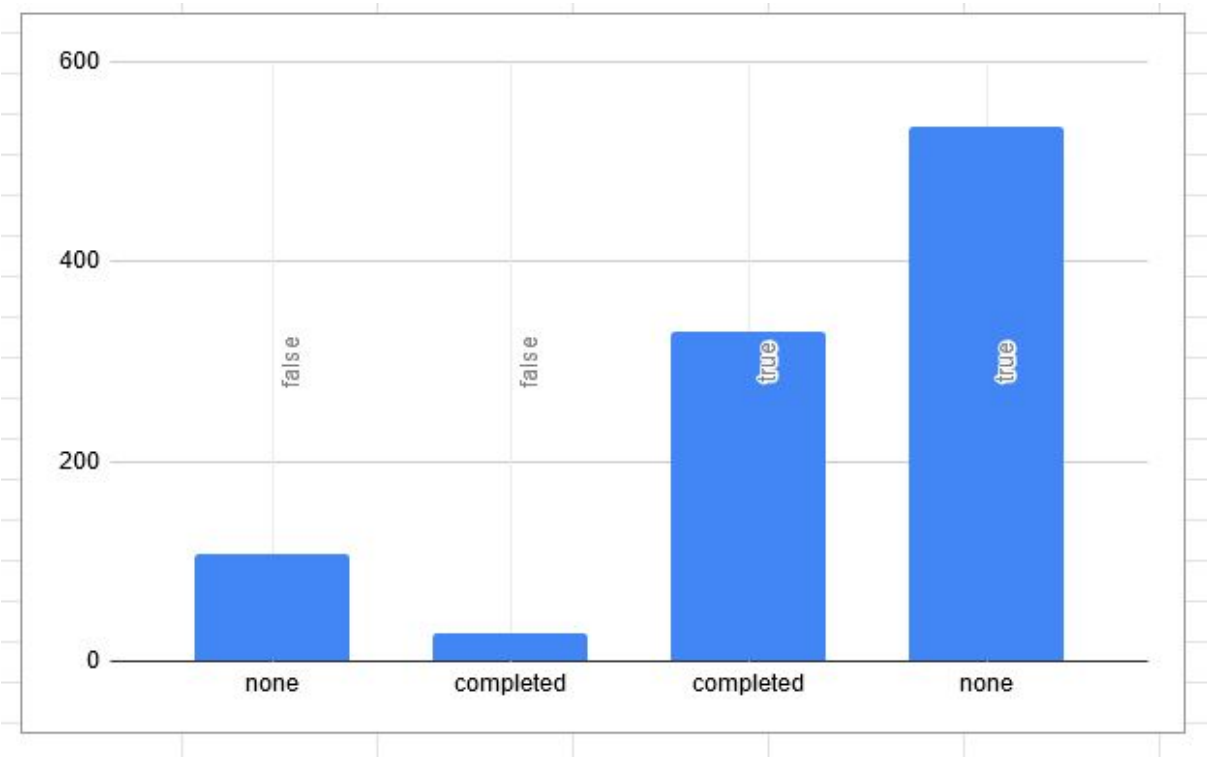
Here's the code

```
    val studentsGroupedByTestPrep = studentsPerformanceDF
      .select (studentsPerformanceDF ("test preparation course"), studentsPerformanceDF ("math score") >= 50)
      .groupBy ("test preparation course", "(math score >= 50)")
      .count ()

    studentsGroupedByTestPrep.write.csv ("StudentsGroupedByTestPrep.csv")
    studentsGroupedByTestPrep.show ()
    computeEachGroupPercentage (studentsGroupedByTestPrep, totalNumberOfStudents)
```

Here's the output

```
+----------------------+-----------------+-----+
|test preparation course|(math score >= 50)|count|
+----------------------+-----------------+-----+
|                  none|            false|  107|
|             completed|            false|   28|
|             completed|             true|  330|
|                  none|             true|  535|
+----------------------+-----------------+-----+

Percentage of none failed is: 0.107
Percentage of completed failed is: 0.028
Percentage of completed passed is: 0.33
Percentage of none passed is: 0.535
```

Here's the graph



Other Analysis Approaches

Retrieving Top 10 Students

Here's the code

```
1.      studentsPerformanceDF
2.        .orderBy (desc ("math score"))
3.        .limit (10)
4.        .show ()
```

Here's the output

gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
male	group E	associate's degree	free/reduced	completed	100	100	93
female	group E	some college	standard	none	100	92	97
female	group E	bachelor's degree	standard	none	100	100	100
male	group A	some college	standard	completed	100	96	86
male	group D	some college	standard	completed	100	97	99
male	group E	bachelor's degree	standard	completed	100	100	100
female	group E	associate's degree	standard	none	100	100	100
female	group E	high school	standard	none	99	93	90
female	group E	bachelor's degree	standard	completed	99	100	100
male	group E	some college	standard	completed	99	87	81

As we observed earlier, most of the students neither the gender, nor the parental level of education make an indication to whether or not the student will perform. However, we can observe that the top 10 students had their standard meal, and they're from "group E" race. So I thought of running another query on the data to have a better idea of what is the "race" and the "lunch" attributes of the students who scored more than 85 (Excellent Students).

Analyzing Top 10 Students

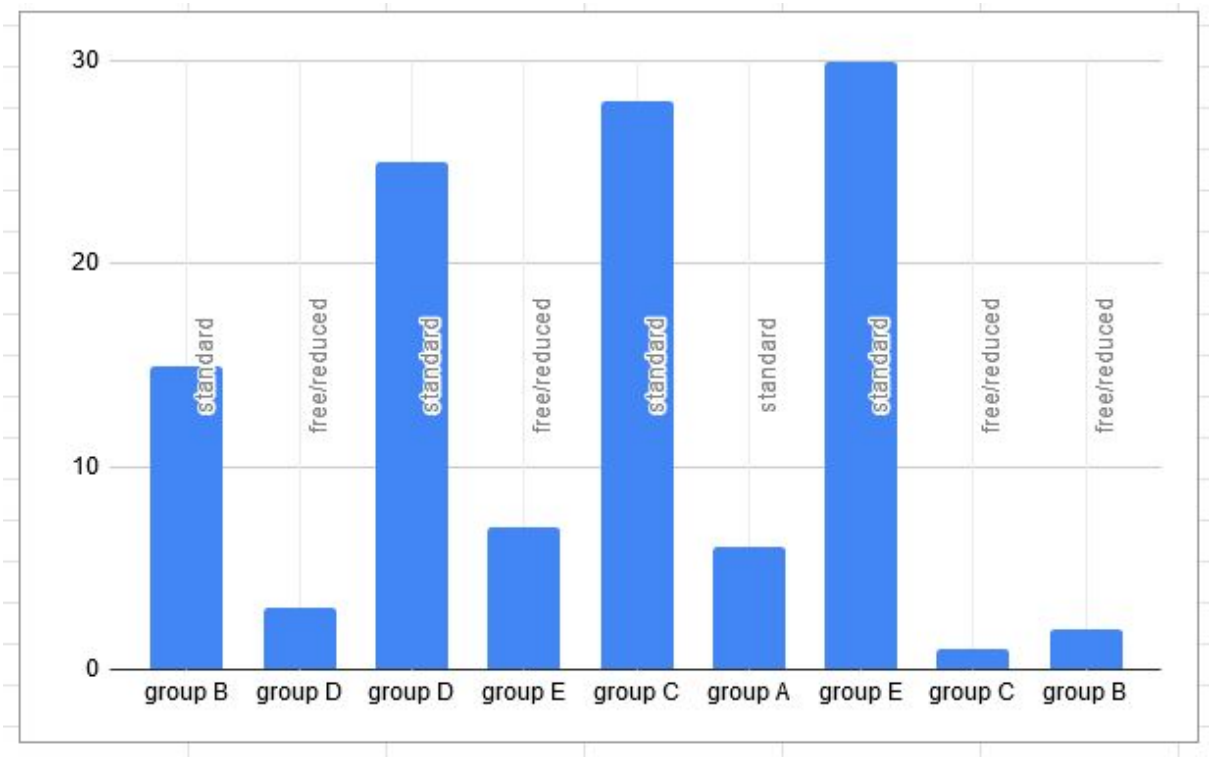
Here's the code

```
// Query to select only students who got excellent
// and analyzing their race, and lunch attributes
val studentsGotExcellentGrouped = studentsPerformanceDF
  .filter (studentsPerformanceDF ("math score") >= 85)
  .select (studentsPerformanceDF ("race/ethnicity"), studentsPerformanceDF ("lunch"))
  .groupBy ("race/ethnicity", "lunch")
  .count ()

// We need to count the sum of the counts outputted from
   // the previous query dataframe
   val studentsGotExcellentCount = studentsGotExcellentGrouped
     .agg (sum ("count"))
     .first ()
     .getLong (0)

   studentsGotExcellentGrouped.show ()
   computeEachGroupPercentage (studentsGotExcellentGrouped, studentsGotExcellentCount)
```

Here's the output

```
+-------------+------------+-----+
|race/ethnicity|       lunch|count|
+-------------+------------+-----+
|       group B|    standard|   15|
|       group D|free/reduced|    3|
|       group D|    standard|   25|
|       group E|free/reduced|    7|
|       group C|    standard|   28|
|       group A|    standard|    6|
|       group E|    standard|   30|
|       group C|free/reduced|    1|
|       group B|free/reduced|    2|
+-------------+------------+-----+

Percentage of group B standard is: 0.1282051282051282
Percentage of group D free/reduced is: 0.02564102564102564
Percentage of group D standard is: 0.21367521367521367
Percentage of group E free/reduced is: 0.05982905982905983
Percentage of group C standard is: 0.23931623931623933
Percentage of group A standard is: 0.05128205128205128
Percentage of group E standard is: 0.2564102564102564
Percentage of group C free/reduced is: 0.008547008547008548
Percentage of group B free/reduced is: 0.017094017094017096
```

Here's the graph



It turned out from the above output, that there is a common attribute between the students who got excellent which is having their standard lunch meal.The majority of the students who got high grades, had also had their standard meal. Another thing we can notice is that race "E" has the highest number of top students.

Retrieving Last 10 Students

Here's the code

```
1.      studentsPerformanceDF
2.        .orderBy ("math score")
3.        .limit (10)
4.        .show ()
```

Here's the output

```
+------+-------------+-------------------------+------------+----------------------+----------+-------------+-------------+
|gender|race/ethnicity|parental level of education|       lunch|test preparation course|math score|reading score|writing score|
+------+-------------+-------------------------+------------+----------------------+----------+-------------+-------------+
|female|      group C|          some high school|free/reduced|                  none|         0|           17|           10|
|female|      group B|               high school|free/reduced|                  none|         8|           24|           23|
|female|      group B|          some high school|free/reduced|                  none|        18|           32|           28|
|female|      group B|              some college|    standard|                  none|        19|           38|           32|
|female|      group C|              some college|free/reduced|                  none|        22|           39|           33|
|female|      group B|               high school|free/reduced|             completed|        23|           44|           36|
|female|      group B|          some high school|free/reduced|                  none|        24|           38|           27|
|female|      group D|         associate's degree|free/reduced|                  none|        26|           31|           38|
|female|      group D|          some high school|free/reduced|                  none|        27|           34|           32|
|  male|      group C|               high school|free/reduced|                  none|        27|           34|           36|
+------+-------------+-------------------------+------------+----------------------+----------+-------------+-------------+
```

Computing Average of Math Scores

Here's the code

```
5.      val studentsMarksSum = studentsPerformanceDF
6.        .agg (sum ("math score"))
7.        .first ()
8.        .getLong (0)
9.  println ("Average of math score between students: " +
   (studentsMarksSum * 1.0 / totalNumberOfStudents))
```

Here's the output

```
Average of math score between students: 66.089
```

Retrieving Student Scored The Highest In All Subjects

Here's the code

```
1.        studentsPerformanceDF
2.          .orderBy (desc ("math score") , desc ("reading score"), desc
   ("writing score"))
3.          .limit (1)
4.          .show ()
```

Here's the result

```
Average of math score between students: 66.089
+------+--------------+--------------------------+--------+----------------------+----------+-------------+-------------+
|gender|race/ethnicity|parental level of education|   lunch|test preparation course|math score|reading score|writing score|
+------+--------------+--------------------------+--------+----------------------+----------+-------------+-------------+
|female|       group E|          bachelor's degree|standard|                  none|       100|          100|          100|
+------+--------------+--------------------------+--------+----------------------+----------+-------------+-------------+
```

As mentioned above, the most common factor/attribute between students who got excellent is having their standard meal, we can see that she also scored 100/100 in other subjects as well. Another important thing to note is that she was from the race of "group E" which confirms the observations mentioned above

SparkSQL

Top 3 female students scored in math for each parental level of education

I tried two approaches to execute the first query.

First Approach

- I made a SQL Query for each group and got the result
- Code

```
        // Top 3 female students from associate's degree
        spark.sql ("SELECT gender, parental_level_of_education, math_score FROM students
WHERE gender=\"female\" AND parental_level_of_education=\"associate's degree\" ORDER BY
math_score DESC LIMIT 3").show ()

        // Top 3 female students from bachelor's degree
        spark.sql ("SELECT gender, parental_level_of_education, math_score FROM students
WHERE gender=\"female\" AND parental_level_of_education=\"bachelor's degree\" ORDER BY
math_score DESC LIMIT 3").show ()

        // Top 3 female students from some college
        spark.sql ("SELECT gender, parental_level_of_education, math_score FROM students
WHERE gender=\"female\" AND parental_level_of_education=\"some college\" ORDER BY
math_score DESC LIMIT 3").show ()

        // Top 3 female students from high school
        spark.sql ("SELECT gender, parental_level_of_education, math_score FROM students
WHERE gender=\"female\" AND parental_level_of_education=\"high school\" ORDER BY
math_score DESC LIMIT 3").show ()

        // Top 3 female students from some high school
        spark.sql ("SELECT gender, parental_level_of_education, math_score FROM students
WHERE gender=\"female\" AND parental_level_of_education=\"some high school\" ORDER BY
math_score DESC LIMIT 3").show ()

        // Top 3 female students from master's degree
        spark.sql ("SELECT gender, parental_level_of_education, math_score FROM students
```

```
WHERE gender=\"female\" AND parental_level_of_education=\"master's degree\" ORDER BY
math_score DESC LIMIT 3").show ()
```

- Output

```
+------+---------------------------+----------+
|gender|parental_level_of_education|math_score|
+------+---------------------------+----------+
|female|         associate's degree|       100|
|female|         associate's degree|        96|
|female|         associate's degree|        95|
+------+---------------------------+----------+

+------+---------------------------+----------+
|gender|parental_level_of_education|math_score|
+------+---------------------------+----------+
|female|          bachelor's degree|       100|
|female|          bachelor's degree|        99|
|female|          bachelor's degree|        97|
+------+---------------------------+----------+

+------+---------------------------+----------+
|gender|parental_level_of_education|math_score|
+------+---------------------------+----------+
|female|               some college|       100|
|female|               some college|        98|
|female|               some college|        88|
+------+---------------------------+----------+
```

```
+------+---------------------------+----------+
|gender|parental_level_of_education|math_score|
+------+---------------------------+----------+
|female|                high school|        99|
|female|                high school|        88|
|female|                high school|        87|
+------+---------------------------+----------+

+------+---------------------------+----------+
|gender|parental_level_of_education|math_score|
+------+---------------------------+----------+
|female|           some high school|        97|
|female|           some high school|        92|
|female|           some high school|        85|
+------+---------------------------+----------+

+------+---------------------------+----------+
|gender|parental_level_of_education|math_score|
+------+---------------------------+----------+
|female|            master's degree|        94|
|female|            master's degree|        92|
|female|            master's degree|        90|
+------+---------------------------+----------+
```

Second Approach

- I tried searching for an easier way to execute the required query, I found the OVER (), and PARTITION BY Clauses
- PARTITION BY partitions our data into some partitions where we can perform our aggregation operation on
- This is what I've understood from reading about the OVER and PARTITION BY, and please correct me if I'm wrong. Basically OVER defines a subset of rows from the table where the aggregation operation we'd like to apply execute on.
- We use partition by to divide our data into several data subsets, each partitioned by the column we're dividing the data upon

```
    // Top 3 female students from each parental level of education using OVER () and
PARTITION BY Clause divided to groups but ordered
    spark.sql ("SELECT gender, parental_level_of_education, math_score FROM ( " +
      "SELECT gender, parental_level_of_education, math_score, row_number () OVER
(PARTITION BY parental_level_of_education ORDER BY math_score DESC) row_num " +
      "FROM students " +
      "WHERE gender=\"female\" ) " +
      "WHERE row_num <= 3 " +
      "ORDER BY math_score DESC").show ()

    // Top 3 female students from each parental level of education using OVER () and
PARTITION BY Clause divided to groups but not ordered
    spark.sql ("SELECT gender, parental_level_of_education, math_score FROM ( " +
      "SELECT gender, parental_level_of_education, math_score, row_number () OVER
(PARTITION BY parental_level_of_education ORDER BY math_score DESC) row_num " +
      "FROM students " +
      "WHERE gender=\"female\" ) " +
      "WHERE row_num <= 3 ").show ()
```

- In the above code, we're using nested queries, where we first partition our data by the parental level of education. Now the returned dataset after the partitioning would be something like this

gender	parental_level_of_education	math_score	row_num
female	High school	100	1
female	High school	99	2
female	High school	99	3
. . .	. . .	. . .	. . .
female	Bachelor's degree	100	1
female	Bachelor's degree	100	2

Female	Bachelor's degree	95	3
. . .	. . .	. . .	. . .

So now we've the data partitioned into groups according to their parental level of education as illustrated above. The outer query would then select only the first 3 row numbers from each group. Here's the output of this query ordered, and not ordered

1. Ordered according to math_score

```
+------+-------------------------+----------+
|gender|parental_level_of_education|math_score|
+------+-------------------------+----------+
|female|         associate's degree|       100|
|female|          bachelor's degree|       100|
|female|               some college|       100|
|female|          bachelor's degree|        99|
|female|                high school|        99|
|female|               some college|        98|
|female|          bachelor's degree|        97|
|female|           some high school|        97|
|female|         associate's degree|        96|
|female|         associate's degree|        95|
|female|            master's degree|        94|
|female|           some high school|        92|
|female|            master's degree|        92|
|female|            master's degree|        90|
|female|                high school|        88|
|female|               some college|        88|
|female|                high school|        87|
|female|           some high school|        85|
+------+-------------------------+----------+
```

2. Not ordered according to math_score, and partitioned as its from the inner partitioning query

```
+------+---------------------------+----------+
|gender|parental_level_of_education|math_score|
+------+---------------------------+----------+
|female|           some high school|        97|
|female|           some high school|        92|
|female|           some high school|        85|
|female|         associate's degree|       100|
|female|         associate's degree|        96|
|female|         associate's degree|        95|
|female|                high school|        99|
|female|                high school|        88|
|female|                high school|        87|
|female|          bachelor's degree|       100|
|female|          bachelor's degree|        99|
|female|          bachelor's degree|        97|
|female|            master's degree|        94|
|female|            master's degree|        92|
|female|            master's degree|        90|
|female|               some college|       100|
|female|               some college|        98|
|female|               some college|        88|
+------+---------------------------+----------+
```

Also here's a link where I read about the "over", and "partition by"

Top 3 students who scored in math more than in reading and less than in writing

- Code

```
        // Top 3 students who scored in math more than in reading and less than in
writing
        spark.sql ("SELECT * FROM students WHERE math_score > reading_score AND
math_score < writing_score ORDER BY math_score DESC LIMIT 3").show ()
```

- Output

```
+------+-------------+-------------------------+--------+----------------------+----------+-------------+-------------+
|gender|race/ethnicity|parental_level_of_education|   lunch|test preparation course|math_score|reading_score|writing_score|
+------+-------------+-------------------------+--------+----------------------+----------+-------------+-------------+
|  male|       group B|         associate's degree|standard|             completed|        91|           89|           92|
|female|       group E|               some college|standard|                  none|        87|           85|           93|
|female|       group E|               some college|standard|             completed|        86|           85|           91|
+------+-------------+-------------------------+--------+----------------------+----------+-------------+-------------+
```

Top 3 students in math, and in reading and in writing

- Code

```
        // Top 3 students in math, reading, and writing
        spark.sql ("SELECT * FROM students ORDER BY math_score DESC, reading_score DESC,
writing_score DESC LIMIT 3").show ()
```

- Output

```
+------+--------------+--------------------------+--------+----------------------+----------+-------------+-------------+
|gender|race/ethnicity|parental_level_of_education|   lunch|test preparation course|math_score|reading_score|writing_score|
+------+--------------+--------------------------+--------+----------------------+----------+-------------+-------------+
|female|       group E|         associate's degree|standard|                  none|       100|          100|          100|
|  male|       group E|          bachelor's degree|standard|             completed|       100|          100|          100|
|female|       group E|          bachelor's degree|standard|                  none|       100|          100|          100|
+------+--------------+--------------------------+--------+----------------------+----------+-------------+-------------+
```

Retrieving top 3 students in math, and in reading, and in writing along with their scores sum

- Code

```
        // Top 3 students in math, reading, and writing
        spark.sql ("SELECT *, (math_score + reading_score + writing_score) as
total_score_sum FROM students ORDER BY total_score_sum DESC LIMIT 3").show ()
```

- Output

```
+------+--------------+--------------------------+--------+----------------------+----------+-------------+-------------+---------------+
|gender|race/ethnicity|parental_level_of_education|   lunch|test preparation course|math_score|reading_score|writing_score|total_score_sum|
+------+--------------+--------------------------+--------+----------------------+----------+-------------+-------------+---------------+
|female|       group E|          bachelor's degree|standard|                  none|       100|          100|          100|            300|
|  male|       group E|          bachelor's degree|standard|             completed|       100|          100|          100|            300|
|female|       group E|         associate's degree|standard|                  none|       100|          100|          100|            300|
+------+--------------+--------------------------+--------+----------------------+----------+-------------+-------------+---------------+
```

Preprocessing Dataframe

I had to preprocess the column names in the dataframe, as Spark had some difficulties in parsing the query. For example, column named “parental level of education” would result in an error if included in the query as is, since it includes spaces. So I had to rename the columns using the .withColumnReanmed () method.

```
    val studentsPerformanceDFRenamed = studentsPerformanceDF
      .withColumnRenamed ("parental level of education", "parental_level_of_education")
      .withColumnRenamed ("math score", "math_score")
      .withColumnRenamed ("reading score", "reading_score")
      .withColumnRenamed ("writing score", "writing_score")

    studentsPerformanceDFRenamed.createOrReplaceTempView ("students")
```

Surrounding Columns With Backticks

- Surrounding column names with backticks in a query indeed results in the query being executed successfully and the correct output
- Here’s the code

```
        studentsPerformanceDF.createOrReplaceTempView ("students")

        println ("Trying out surrounding columns with backticks")
```

```
spark.sql ("SELECT gender, `parental level of education`, `math score` FROM students WHERE `math score` >= 50 ORDER BY `math score` DESC LIMIT 10").show ()
```

- Here's the output

```
Trying out surrounding columns with backticks
+------+--------------------------+----------+
|gender|parental level of education|math score|
+------+--------------------------+----------+
|  male|         associate's degree|       100|
|female|               some college|       100|
|female|          bachelor's degree|       100|
|  male|               some college|       100|
|  male|               some college|       100|
|  male|          bachelor's degree|       100|
|female|         associate's degree|       100|
|female|          bachelor's degree|        99|
|female|                high school|        99|
|  male|               some college|        99|
+------+--------------------------+----------+
```

Data Quality

Custom Schema

- Code

```
// Loading StudentsPerformance.csv with custom schema
val customSchema = new StructType ()
  .add ("gender", StringType, true)
  .add ("race", StringType, true)
  .add ("parental_level_of_education", StringType, true)
  .add ("lunch", StringType, true)
  .add ("test_preparation_course", StringType, true)
  .add ("math_score", IntegerType, true)
  .add ("reading_score", IntegerType, true)
  .add ("writing_score", IntegerType, true)

val studentsPerformanceDFWithCustomSchema = spark.read
  .schema (customSchema)
  .option ("header", "true")
  .option ("inferSchema", "false")
  .option ("nullValue", "null")
  .csv ("StudentsPerformance.csv")

studentsPerformanceDFWithCustomSchema.printSchema ()
studentsPerformanceDFWithCustomSchema.show ()
```

- Output

```
root
 |-- gender: string (nullable = true)
 |-- race: string (nullable = true)
 |-- parental_level_of_education: string (nullable = true)
 |-- lunch: string (nullable = true)
 |-- test_preparation_course: string (nullable = true)
 |-- math_score: integer (nullable = true)
 |-- reading_score: integer (nullable = true)
 |-- writing_score: integer (nullable = true)

+------+-------+-------------------------+------------+----------------------+----------+-------------+-------------+
|gender|   race|parental_level_of_education|       lunch|test_preparation_course|math_score|reading_score|writing_score|
+------+-------+-------------------------+------------+----------------------+----------+-------------+-------------+
|female|group B|          bachelor's degree|    standard|                  none|        72|           72|           74|
|female|group C|              some college|    standard|             completed|        69|           90|           88|
|female|group B|            master's degree|    standard|                  none|        90|           95|           93|
|  male|group A|         associate's degree|free/reduced|                  none|        47|           57|           44|
|  male|group C|              some college|    standard|                  none|        76|           78|           75|
|female|group B|         associate's degree|    standard|                  none|        71|           83|           78|
|female|group B|              some college|    standard|             completed|        88|           95|           92|
|  male|group B|              some college|free/reduced|                  none|        40|           43|           39|
|  male|group D|               high school|free/reduced|             completed|        64|           64|           67|
|female|group B|               high school|free/reduced|                  none|        38|           60|           50|
|  male|group C|         associate's degree|    standard|                  none|        58|           54|           52|
|  male|group D|         associate's degree|    standard|                  none|        40|           52|           43|
|female|group B|               high school|    standard|                  none|        65|           81|           73|
|  male|group A|              some college|    standard|             completed|        78|           72|           70|
|female|group A|            master's degree|    standard|                  none|        50|           53|           58|
|female|group C|           some high school|    standard|                  none|        69|           75|           78|
|  male|group C|               high school|    standard|                  none|        88|           89|           86|
|female|group B|           some high school|free/reduced|                  none|        18|           32|           28|
|  male|group C|            master's degree|free/reduced|             completed|        46|           42|           46|
|female|group C|         associate's degree|free/reduced|                  none|        54|           58|           61|
+------+-------+-------------------------+------------+----------------------+----------+-------------+-------------+
only showing top 20 rows
```

Saving a CSV File Into a Hive Parquet Table

- Code

```
val studentsPerformanceSchema = new StructType ()
        .add ("gender", StringType, true)
        .add ("race", StringType, true)
        .add ("parental_level_of_education", StringType, true)
        .add ("lunch", StringType, true)
        .add ("test_preparation_course", StringType, true)
        .add ("math_score", IntegerType, true)
        .add ("reading_score", IntegerType, true)
        .add ("writing_score", IntegerType, true)

      val studentsPerformanceCSV = spark.read
        .option ("header", "true")
        .option ("inferSchema", "false")
        .option ("nullValue", "null")
        .schema (studentsPerformanceSchema)
        .csv ("StudentsPerformance.csv")

      studentsPerformanceCSV.show ()

      // Saving loaded csv file to a Hive Parquet Table
      studentsPerformanceCSV.write.mode ("overwrite").format
("parquet").saveAsTable ("hive_test_db.students_performance_table")
```

- Here’s the inserted table in the Hue Editor

4.58s hive_test_db text ?

```
SELECT * FROM hive_test_db.students_performance_table;
```

Query History Saved Queries Results (100+)

	gender	race	parental_level_of_education	lunch	test_preparation_course	math_score	reading_score	writing_score
1	female	group B	bachelor's degree	standard	none	72	72	74
2	female	group C	some college	standard	completed	69	90	88
3	female	group B	master's degree	standard	none	90	95	93
4	male	group A	associate's degree	free/reduced	none	47	57	44
5	male	group C	some college	standard	none	76	78	75
6	female	group B	associate's degree	standard	none	71	83	78
7	female	group B	some college	standard	completed	88	95	92
8	male	group B	some college	free/reduced	none	40	43	39
9	male	group D	high school	free/reduced	completed	64	64	67
10	female	group B	high school	free/reduced	none	38	60	50
11	male	group C	associate's degree	standard	none	58	54	52
12	male	group D	associate's degree	standard	none	40	52	43
13	female	group B	high school	standard	none	65	81	73

Loading Hive Parquet Table into a DataFrame

- Code

```
      // Reading Table as a DataFrame
      val studentsPerformanceDF = spark.sql ("SELECT * FROM
hive_test_db.students_performance_table")
      studentsPerformanceDF.printSchema()
      studentsPerformanceDF.show ()
      studentsPerformanceDF.cache ()
```

- printSchema () and show () methods results

```
root
 |-- gender: string (nullable = true)
 |-- race: string (nullable = true)
 |-- parental_level_of_education: string (nullable = true)
 |-- lunch: string (nullable = true)
 |-- test_preparation_course: string (nullable = true)
 |-- math_score: integer (nullable = true)
 |-- reading_score: integer (nullable = true)
 |-- writing_score: integer (nullable = true)

+------+-------+--------------------------+------------+----------------------+----------+-------------+-------------+
|gender|   race|parental_level_of_education|       lunch|test_preparation_course|math_score|reading_score|writing_score|
+------+-------+--------------------------+------------+----------------------+----------+-------------+-------------+
|female|group B|          bachelor's degree|    standard|                  none|        72|           72|           74|
|female|group C|               some college|    standard|             completed|        69|           90|           88|
|female|group B|            master's degree|    standard|                  none|        90|           95|           93|
|  male|group A|         associate's degree|free/reduced|                  none|        47|           57|           44|
|  male|group C|               some college|    standard|                  none|        76|           78|           75|
|female|group B|         associate's degree|    standard|                  none|        71|           83|           78|
|female|group B|               some college|    standard|             completed|        88|           95|           92|
|  male|group B|               some college|free/reduced|                  none|        40|           43|           39|
|  male|group D|                high school|free/reduced|             completed|        64|           64|           67|
|female|group B|                high school|free/reduced|                  none|        38|           60|           50|
|  male|group C|         associate's degree|    standard|                  none|        58|           54|           52|
|  male|group D|         associate's degree|    standard|                  none|        40|           52|           43|
|female|group B|                high school|    standard|                  none|        65|           81|           73|
|  male|group A|               some college|    standard|             completed|        78|           72|           70|
|female|group A|            master's degree|    standard|                  none|        50|           53|           58|
|female|group C|           some high school|    standard|                  none|        69|           75|           78|
|  male|group C|                high school|    standard|                  none|        88|           89|           86|
|female|group B|           some high school|free/reduced|                  none|        18|           32|           28|
|  male|group C|            master's degree|free/reduced|             completed|        46|           42|           46|
|female|group C|         associate's degree|free/reduced|                  none|        54|           58|           61|
+------+-------+--------------------------+------------+----------------------+----------+-------------+-------------+
only showing top 20 rows
```

Difference Between Hive Parquet Table Approach and CSV File Approach

Regarding Code and Results

- There isn't any difference between the two approaches, in the two approaches I used dataframes, and the code was the same to achieve the analysis and spark.sql () queries
- The results outputted were the same as well.
- The difference was only in the way we load our data into a dataframe, which is illustrated above

Regarding Performance

- I'm not sure if I could notice any difference regarding the performance, both of them were approximately the same to me. I tried accessing the Spark UI to notice if there are any time differences between the jobs in the two approaches, unfortunately, I couldn't notice significant improvement.
- I'm guessing this is the case for me, since I'm using relatively small data (1000 records only).
- I've been searching to see what are the advantages of using one approach over the other, I've found that tables stored as parquet files should be much faster for analytical queries (queries that access specific columns only in the table) than the

row-oriented formats like csv. However, CSV files should be more efficient if we were to access all the columns.

- Here are some of these resources
    - https://dataschool.com/data-modeling-101/row-vs-column-oriented-databases/
    - https://stackoverflow.com/questions/39541315/is-querying-against-a-spark-dataframe-based-on-csv-faster-than-one-based-on-parq
    - https://stackoverflow.com/questions/48727052/spark-dataframe-csv-vs-parquet