Name of the Student : Rebecca Hilary Dias
Rollno :19
PID:182027

as per the ethnicity compute the avg of math score

What is Pyspark?
PySpark is a Python API for Spark released by the Apache Spark community to support Python with Spark. Using PySpark, one can easily integrate and work with RDDs in Python programming language too. There are numerous features that make PySpark such an amazing framework when it comes to working with huge datasets.

Why is PYspark used?
PySpark is a great language for data scientists to learn because it enables scalable analysis and ML pipelines. If you're already familiar with Python and Pandas, then much of your knowledge can be applied to Spark. I've shown how to perform some common operations with PySpark to bootstrap the learning process

- PySpark ML and MLlib is used for machine learning.
- PySpark GraphX and GraphFrames are used for Graph processing
- PySpark RDD and DataFrame’s are used to process batch pipelines where you would need high throughput.
- PySpark Streaming is used to for real time processing.

Where is Pyspark used?
PySpark brings robust and cost-effective ways to run machine learning applications on billions and trillions of data on distributed clusters 100 times faster than the traditional python applications. PySpark has been used by many organizations like Amazon, Walmart, Trivago, Sanofi, Runtastic, and many more.
PySpark has been used by many organizations like Amazon, Walmart, Trivago, Sanofi, Runtastic, and many more. PySpark also used in different sectors.

- Health
- E-commerce and many more
- Financials
- Education
- Entertainment

- Utilities

Connecting Drive to Colab: Try connecting to your own google drive. You would be prompted to copy the authorization code. Please do that to proceed further

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

1. **Loading your data** The data to be downloaded in pyspark can be in zip file. Just upload the zip file in the drive and use your unzip extention to pull it out.

For loading the data, kindly search in UCL repository, Kaggle and other other valid data sources and try downloading the dataset which you can analyze. I shall shown one example here. But I request you'll to kindly download you own datasets and do the analysis. You may download the dataset in the drive directly and you may unzip it. Follow step-by-step the instructions given below for working with your data sets using Pyspark

This *AsianReligiousData.zip* has two csv files in it. AllBooks_baseline_DTM_labelled.csv and AllBooks_baseline_DTM_Unlabelled.csv. and a text file about the dataset

Just investigate what happens when you try to unzip the code for the second time. Add atext box and write your investigations here

Observation:

1.Setting up Pyspark in Colab a. The first step is to download java.

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

Observation:- Why is the above command used?

The above command is used to install Java.

```
!wget -q https://mirrors.estointernet.in/apache/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.t
```

Observation: What is the use of wget function?

Wget is a free GNU command-line utility tool used to download files from the internet. It retrieves files using HTTP, HTTPS, and FTP protocols.

It serves as a tool to sustain unstable and slow network connections. If a network problem occurs during a download, this helpful software can resume retrieving the files without starting from scratch.

```
mv spark-3.1.2-bin-hadoop3.2.tgz spark3.tgz
```

Observation: What is a mv function and why is it used here?

Linux mv command is used to move existing file or directory from one location to another. It is also used to rename a file or directory. If you want to rename a single directory or file then 'mv' option will be better to use.

```
!tar xf spark3.tgz
!pip install -q findspark
```

Observation: What is the use of the tar function?

The Linux ‘tar’ stands for tape archive, is used to create Archive and extract the Archive files. tar command in Linux is one of the important command which provides archiving functionality in Linux. We can use Linux tar command to create compressed or uncompressed Archive files and also maintain and modify them.

```
 tar [options] [archive-file] [file or directory to be archived]
```

The findspark library. It will locate Spark on the system and import it as a regular library.

```
!pip install -q findspark
```

Now that we have installed all the necessary dependencies in Colab, it is time to set the environment path. This will enable us to run Pyspark in the Colab environment.

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
```

```
os.environ["SPARK_HOME"] = "/content/spark-3.1.2-bin-hadoop3.2"
```

Pyspark is located and initiated in your system by using these two commands

```
import findspark
findspark.init()
```

```
findspark.find()
```

```
'/content/spark-3.1.2-bin-hadoop3.2'
```

we can import SparkSession from pyspark.sql and create a SparkSession, which is the entry point to Spark.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder\
        .master("local")\
        .appName("Colab")\
        .config('spark.ui.port', '4050')\
        .getOrCreate()
```

Finally, print the SparkSession variable.

```
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
v3.1.2
Master
local
AppName
Colab

If you have completed so far, you are ready to move forward with Pyspark

Loading Data into Pyspark

```
df = spark.read.csv("StudentsPerformance.csv", header=True, inferSchema=True)
```

Data Exploration with Pyspark DF

First job is to check out with the schema

```
df.printSchema()
```

```
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)
```

Displaying the rows in the dataset

```
df.show(10)
```

```
+------+--------------+---------------------------+------------+----------------------
|gender|race/ethnicity|parental level of education|       lunch|test preparation course
+------+--------------+---------------------------+------------+----------------------
|female|       group B|          bachelor's degree|    standard|                  none
|female|       group C|               some college|    standard|             completed
|female|       group B|           master's degree|    standard|                  none
|  male|       group A|         associate's degree|free/reduced|                  none
|  male|       group C|               some college|    standard|                  none
|female|       group B|         associate's degree|    standard|                  none
|female|       group B|               some college|    standard|             completed
|  male|       group B|               some college|free/reduced|                  none
|  male|       group D|                high school|free/reduced|             completed
|female|       group B|                high school|free/reduced|                  none
+------+--------------+---------------------------+------------+----------------------
only showing top 10 rows
```

Displaying the first 10 rows of the dataset

```
df.head()
```

```
Row(gender='female', race/ethnicity='group B', parental level of education="bachelor's
```

```
df.count()
```

```
1000
```

The count command is used for displaying the number of rows in the dataset

```
df.select()
```

```
DataFrame[]
```

The above command is used to display the columns specified

Observation: Try displaying specific columns and display the output

Describing columns:- The columns are described using this command

```
df.describe().show()
```

```
+-------+------+--------------+--------------------------+------------+---------------
|summary|gender|race/ethnicity|parental level of education|       lunch|test preparatio
+-------+------+--------------+--------------------------+------------+---------------
|  count|  1000|          1000|                      1000|        1000|
|   mean|  null|          null|                      null|        null|
| stddev|  null|          null|                      null|        null|
|    min|female|       group A|         associate's degree|free/reduced|              c
|    max|  male|       group E|           some high school|    standard|
+-------+------+--------------+--------------------------+------------+---------------
```

Another way of uploading files: Alternatively to downloading the zip file to the drive and uploading. You may also download the file of your choice in the local disk and then get it into the dataframe with the method mentioned below

```
from google.colab import files
```

```
uploaded = files.upload()
```

Choose Files StudentsPerformance.csv

- **StudentsPerformance.csv**(application/vnd.ms-excel) - 57021 bytes, last modified: 8/30/2021 - 100% done

```
Saving StudentsPerformance.csv to StudentsPerformance (1).csv
```

Observation:(To be done by the student: Try uploading any dataset through with analytical

```
df1 = spark.read.csv('StudentsPerformance.csv', header=True)
```

Schema of Mall_Customers

```
df1.printSchema()
```

```
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: string (nullable = true)
 |-- reading score: string (nullable = true)
 |-- writing score: string (nullable = true)
```

Implement: Write the command to display the first 5 entries of the dataset you have downloaded using *files.upload()* function

```
df1.head(5)
```

```
[Row(gender='female', race/ethnicity='group B', parental level of education="bachelor's
 Row(gender='female', race/ethnicity='group C', parental level of education='some colle
 Row(gender='female', race/ethnicity='group B', parental level of education="master's d
 Row(gender='male', race/ethnicity='group A', parental level of education="associate's
 Row(gender='male', race/ethnicity='group C', parental level of education='some college
```

Implementation: What is the total number of records in your dataset?

```
df1.count()
```

```
1000
```

Displaying specific columns: You can display specific coloums using df.select() command
Implementation: Display any 2 rows in the dataset you have uploaded into the dataframe

```
df1.select("gender","math score").show()
```

```
+------+----------+
|gender|math score|
+------+----------+
|female|        72|
```

```
|female|        69|
|female|        90|
|  male|        47|
|  male|        76|
|female|        71|
|female|        88|
|  male|        40|
|  male|        64|
|female|        38|
|  male|        58|
|  male|        40|
|female|        65|
|  male|        78|
|female|        50|
|female|        69|
|  male|        88|
|female|        18|
|  male|        46|
|female|        54|
+------+----------+
only showing top 20 rows
```

Implementation: Try finding the count max and min of all the columns in your dataframe.

```
df1.describe().show()
```

```
+-------+------+--------------+--------------------------+------------+---------------
|summary|gender|race/ethnicity|parental level of education|       lunch|test preparatio
+-------+------+--------------+--------------------------+------------+---------------
|  count|  1000|          1000|                      1000|        1000|
|   mean|  null|          null|                      null|        null|
| stddev|  null|          null|                      null|        null|
|    min|female|       group A|         associate's degree|free/reduced|              c
|    max|  male|       group E|           some high school|    standard|
+-------+------+--------------+--------------------------+------------+---------------
```

Computing distinct values of categorical columns

```
df1.select("parental level of education").distinct().show()
```

```
+--------------------------+
|parental level of education|
+--------------------------+
|           some high school|
|         associate's degree|
|                high school|
|          bachelor's degree|
|            master's degree|
|               some college|
```

```
+--------------------------+
```

Aggregate with Groupby

```
from pyspark.sql import functions as F
df1.groupBy("parental level of education").agg(F.avg("reading score")).show()
```

```
+--------------------------+------------------+
|parental level of education|avg(reading score)|
+--------------------------+------------------+
|           some high school| 66.93854748603351|
|         associate's degree| 70.92792792792793|
|                high school| 64.70408163265306|
|          bachelor's degree|              73.0|
|            master's degree| 75.37288135593221|
|               some college| 69.46017699115045|
+--------------------------+------------------+
```

```
from pyspark.sql import functions as F
df1.groupBy("parental level of education").agg(F.max("reading score")).show()
```

```
+--------------------------+------------------+
|parental level of education|max(reading score)|
+--------------------------+------------------+
|           some high school|                97|
|         associate's degree|                96|
|                high school|                99|
|          bachelor's degree|                97|
|            master's degree|                99|
|               some college|                97|
+--------------------------+------------------+
```

```
from pyspark.sql import functions as F
df1.groupBy("parental level of education").agg(F.min("writing score")).show()
```

```
+--------------------------+------------------+
|parental level of education|min(writing score)|
+--------------------------+------------------+
|           some high school|                10|
|         associate's degree|               100|
|                high school|               100|
|          bachelor's degree|               100|
|            master's degree|               100|
|               some college|                19|
+--------------------------+------------------+
```

```
from pyspark.sql import functions as F
```

```
df1.groupBy("gender").agg(F.min("test preparation course")).show()
```

```
+------+---------------------------+
|gender|min(test preparation course)|
+------+---------------------------+
|female|                  completed|
|  male|                  completed|
+------+---------------------------+
```

Implementation: If you have encountered any errors because of the columns name try fixing it before proceeding. Also write the code for the same

Counting and Removing Null values

The columns having null values can be displayed using the command below. For removing it using fillna function. But try it on your own for your own dataset

```
df1.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in df1.columns]).show()
```

```
+------+--------------+--------------------------+-----+----------------------+------
|gender|race/ethnicity|parental level of education|lunch|test preparation course|math s
+------+--------------+--------------------------+-----+----------------------+------
|     0|             0|                         0|    0|                     0|
+------+--------------+--------------------------+-----+----------------------+------
```

Save to file: After analysis save the output. Finally, after doing all the analysis if you want to save your results into a new CSV file, you can do that using the write.csv() function.

```
df.write.csv("final_score.csv") #the path you want the file to be saved in the drive has to b
```

Extra Queries

As per the ethnicity compute the avg of math score

```
df1.groupBy("race/ethnicity").agg(F.avg("math score")).show()
```

```
+--------------+-----------------+
|race/ethnicity|  avg(math score)|
+--------------+-----------------+
|       group B|63.45263157894737|
```

```
|       group C|64.46394984326018|
|       group D|67.36259541984732|
|       group A|61.62921348314607|
|       group E|73.82142857142857|
+--------------+-----------------+
```

Lowest Math Score

```
from pyspark.sql.functions import desc, asc


highest_mathScore = df1.orderBy(['math score'],ascending =[True]).take(1)
print(f"The Level of Education is {highest_mathScore[0][2]} with {highest_mathScore[0][5]} as
```

```
The Level of Education is some high school with 0 as the Lowest Math Score
```

Highest Reading Score

```
highest_readScore = df1.orderBy(['reading score'],ascending =[False]).take(1)
print(f"The Level of Education is {highest_readScore[0][2]} with {highest_readScore[0][6]} as
```

```
The Level of Education is high school with 99 as the Highest Reading Score
```

Highest Writing Score

```
highest_writeScore = df1.orderBy(['writing score'],ascending =[False]).take(1)
print(f"The Level of Education is {highest_writeScore[0][2]} with {highest_writeScore[0][7]}
```

```
The Level of Education is bachelor's degree with 99 as the Highest Writing Score
```

✓ 0s completed at 16:17