**ASSIGNMENT**

**Name:Tejaswini Gokanakonda**

## **Roll no**:**DE142**

## **Date:19-11-2024**

**PYSPARK PROGRAMS**

**1.Creating and Inspecting an RDD in PySpark**

#initialize the program

from pyspark import SparkContext

from pyspark.sql import  SparkSession

sc =SparkContext.getOrCreate()

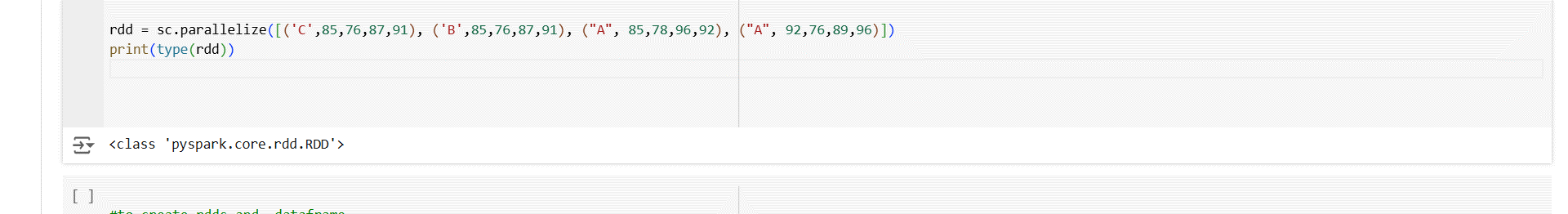
spark = SparkSession.builder.appName('pyspark first program').getOrCreate()

#create the rdd

rdd = sc.parallelize([('C',85,76,87,91), ('B',85,76,87,91), ("A", 85,78,96,92), ("A", 92,76,89,96)])

print(type(rdd))

**Output Screenshot:**



**Summary:**

This code demonstrates how to create an RDD (Resilient Distributed Dataset) in PySpark and inspect its type.

* A SparkContext and SparkSession are initialized using SparkContext.getOrCreate() and SparkSession.builder.appName().
* An RDD (rdd) is created using sc.parallelize(), which contains a list of tuples. Each tuple consists of a string (representing a grade or category) followed by four integer values (representing scores).
* The print(type(rdd)) statement is used to check and display the type of the RDD, which is <class 'pyspark.rdd.RDD'>.

This example shows how to create an RDD in PySpark and inspect its type using the type() function to confirm that it is an RDD.

**2. Creating RDD and DataFrame in PySpark**

#to create rdds and  dataframe

#

from pyspark import SparkContext

from pyspark.sql import  SparkSession

sc =SparkContext.getOrCreate()

spark = SparkSession.builder.appName('pyspark first program').getOrCreate()

#create the rdd

rdd = sc.parallelize([('C',85,76,87,91), ('B',85,76,87,91), ("A", 85,78,96,92), ("A", 92,76,89,96)], 4)

mydata = ['Division','English','Mathematics','Physics','Chemistry']

marks\_df = spark.createDataFrame(rdd, schema=mydata)

print(type(marks\_df))

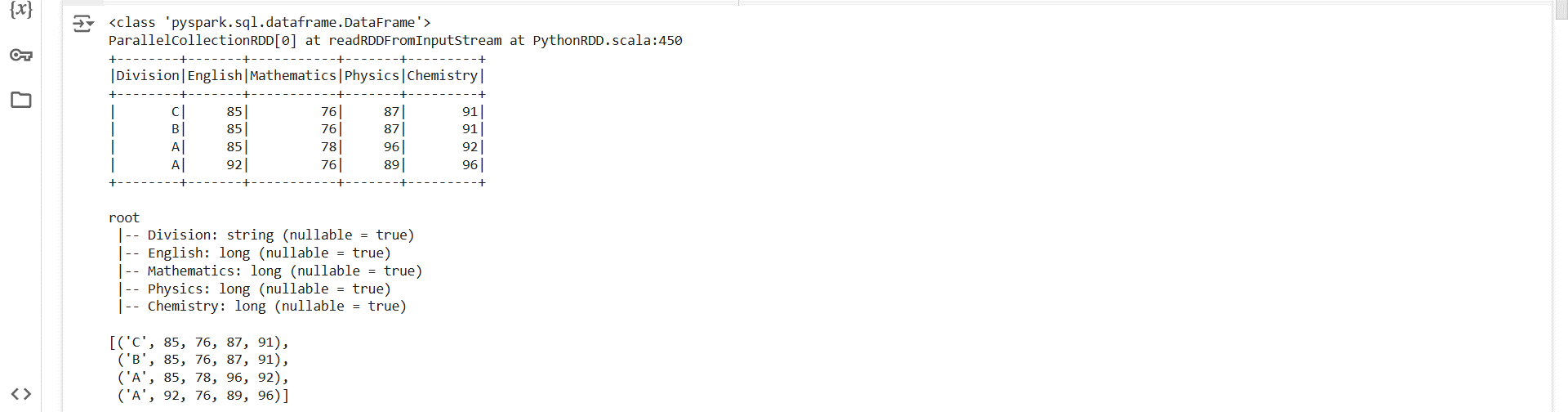
print(rdd)

marks\_df.show()

marks\_df.printSchema()

rdd.collect()

**Output Screenshot:**



**Summary:**

This code demonstrates how to create an RDD and convert it into a DataFrame in PySpark, as well as how to inspect the DataFrame schema and display the data.

* A SparkContext and SparkSession are initialized using SparkContext.getOrCreate() and SparkSession.builder.appName().
* An RDD (rdd) is created using sc.parallelize() with a list of tuples. Each tuple consists of a division/grade and scores for five subjects. The RDD is distributed across 4 partitions.
* A list mydata containing column names for the subjects is created, which is then used as the schema to convert the RDD into a DataFrame (marks\_df) using spark.createDataFrame().
* The type of the DataFrame is displayed using print(type(marks\_df)), confirming that it's a DataFrame.
* The content of the DataFrame is shown using marks\_df.show(), and the schema of the DataFrame is printed using marks\_df.printSchema().
* The rdd.collect() statement retrieves all elements from the RDD and returns them as a list, useful for viewing the RDD data in the driver.

This example illustrates how to create an RDD, convert it into a DataFrame, and display the data and schema of the DataFrame.

**3.** **Creating and Displaying a DataFrame in PySpark**

from pyspark.sql import SparkSession

spark = SparkSession.builder

                    .appName('pyspark\_ex').getOrCreate()

data = [('James','Smith','M',3000),

  ('Anna','Rose','F',4100),

  ('Robert','Williams','M',6200),

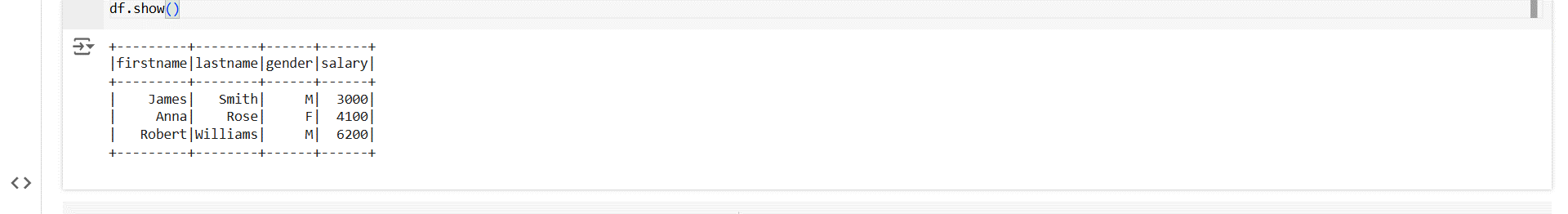
]

columns = ["firstname","lastname","gender","salary"]

df = spark.createDataFrame(data=data, schema = columns)

df.show()

**Output Screenshot:**



**Summary:**

This code demonstrates how to create a DataFrame in PySpark and display its content.

* A SparkSession is initialized using SparkSession.builder.appName() to create a Spark session.
* A list data is created containing tuples with information about employees (first name, last name, gender, and salary).
* The columns list defines the column names for the DataFrame.
* The spark.createDataFrame() method is used to create a DataFrame (df) from the data and columns.
* Finally, df.show() is used to display the content of the DataFrame in a tabular format.

This example shows how to create a simple DataFrame from a list of tuples and display the data using show() in PySpark.

**4.** **Reading CSV Data into DataFrame in PySpark**

#to create rdds and  dataframe

#

from pyspark import SparkContext

from pyspark.sql import  SparkSession

sc =SparkContext.getOrCreate()

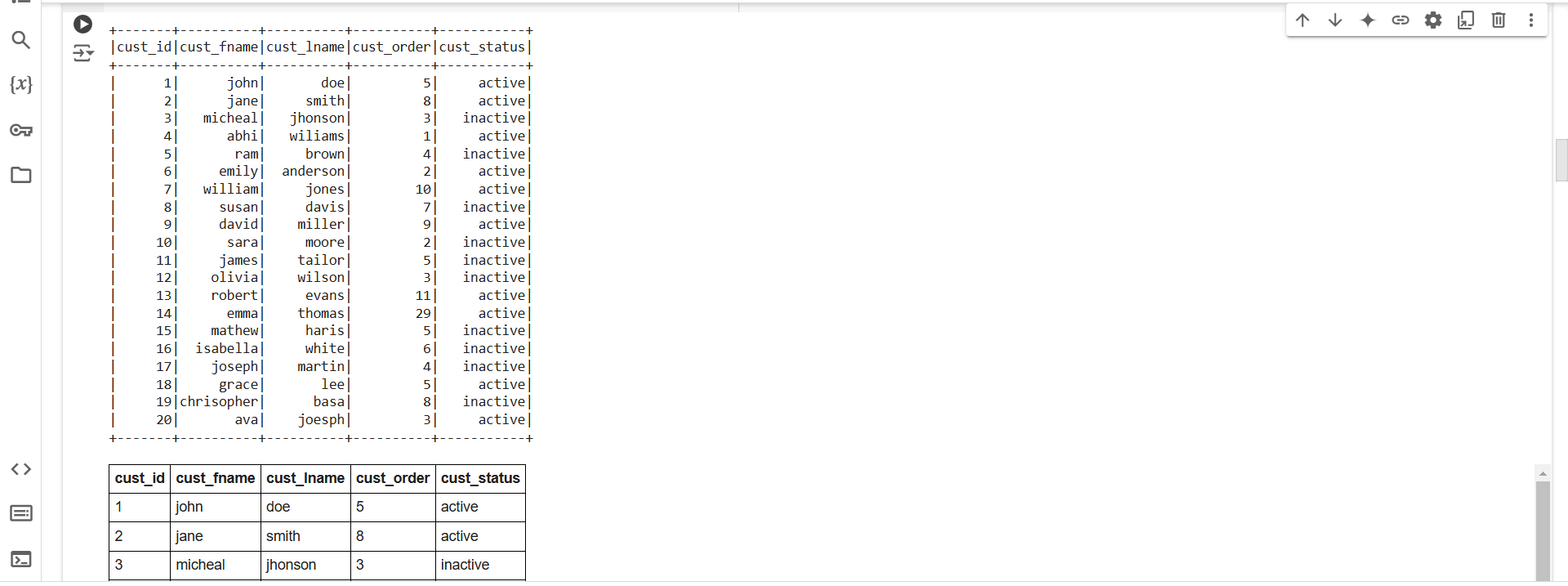
spark = SparkSession.builder.appName('pyspark first program').getOrCreate()

data =spark.read.csv("/FileStore/tables/orders.csv",header = True,inferSchema = True)

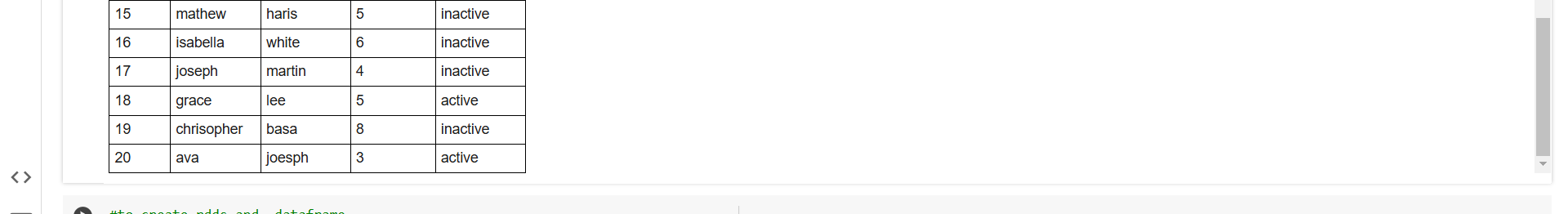
data.show()

display(data)

**Output Screenshot:**







**Summary:**

This code demonstrates how to read a CSV file into a PySpark DataFrame and display its content.

* A SparkContext and SparkSession are initialized using SparkContext.getOrCreate() and SparkSession.builder.appName().
* The spark.read.csv() method is used to read the CSV file (orders.csv) into a DataFrame. The parameters header=True ensure that the first row is treated as column headers, and inferSchema=True automatically infers the data types of the columns based on the data.
* data.show() displays the first few rows of the DataFrame in the console.
* display(data) is used to display the DataFrame in a more readable format (especially in environments like Databricks).

This example shows how to read a CSV file into a DataFrame and view the content using both the show() method and the display() function.

**5. Reading CSV Data into DataFrame Without Header in PySpark**

#to create rdds and  dataframe

#

from pyspark import SparkContext

from pyspark.sql import  SparkSession

sc =SparkContext.getOrCreate()

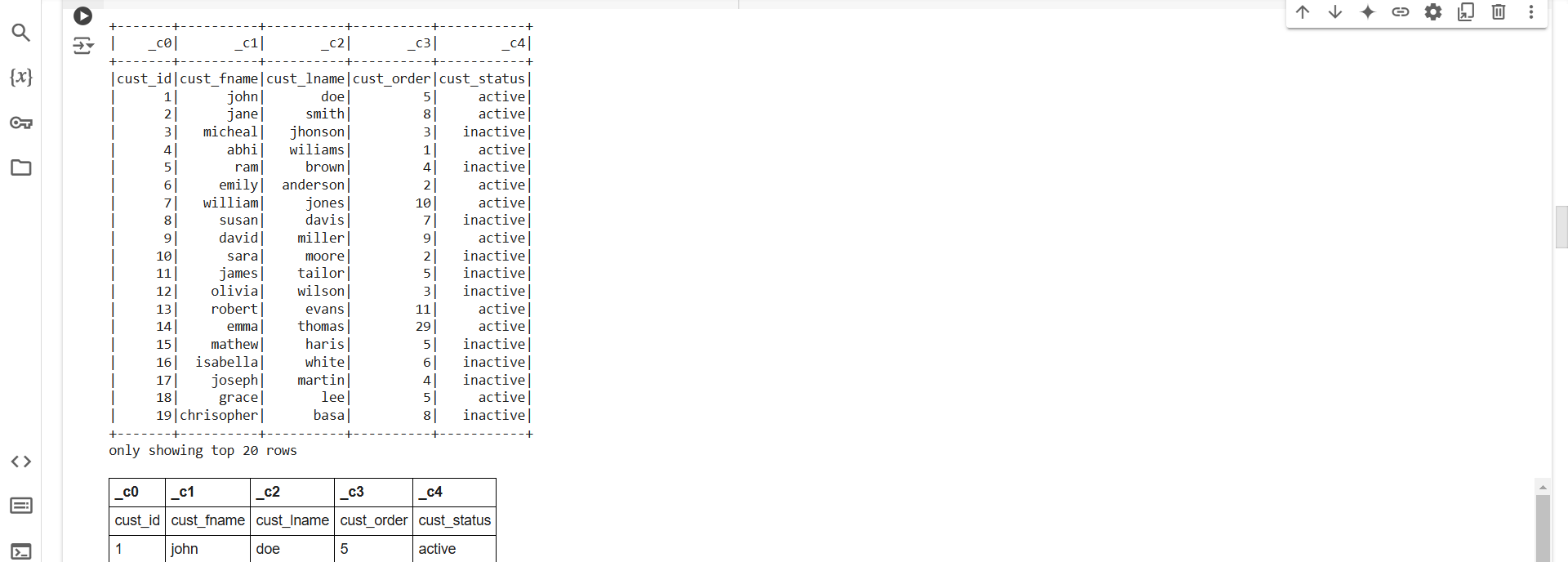
spark = SparkSession.builder.appName('pyspark first program').getOrCreate()

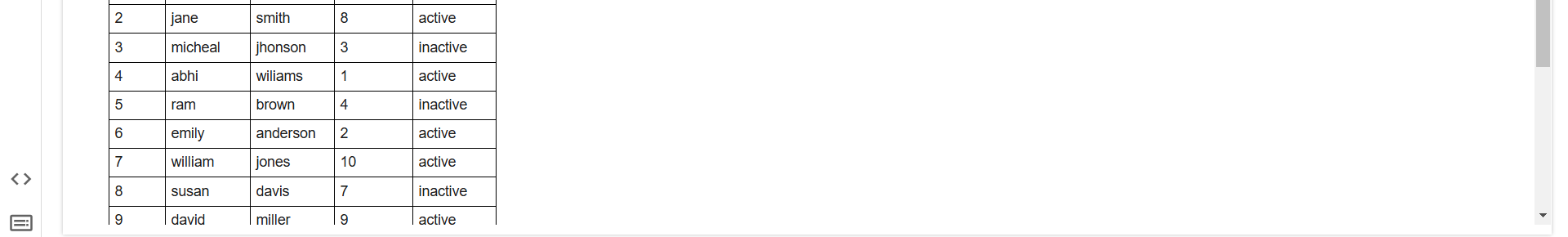
data =spark.read.csv("/FileStore/tables/orders.csv")

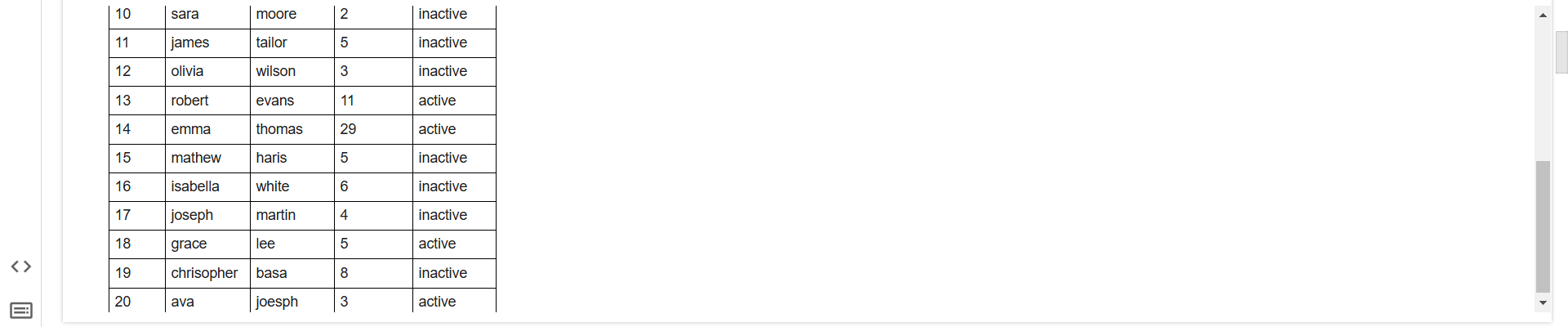
data.show()

display(data)

**Output Screenshot:**







**Summary:**

This code demonstrates how to read a CSV file into a PySpark DataFrame when the header is not explicitly specified.

* A SparkContext and SparkSession are initialized using SparkContext.getOrCreate() and SparkSession.builder.appName().
* The spark.read.csv() method is used to read the CSV file (orders.csv) into a DataFrame. Since the header parameter is not specified, PySpark assumes that the first row contains data rather than column names.
* data.show() is used to display the first few rows of the DataFrame in the console.
* display(data) is used to display the DataFrame in a more interactive format (often used in Databricks environments).

In this case, the first row in the CSV file will be treated as regular data unless further configurations like header=True are added.

**6. Creating DataFrame and Adding New Columns in PySpark**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

                    .appName('pyspark\_ex').getOrCreate()

data = [('James','Smith','M',3000),

  ('Anna','Rose','F',4100),

  ('Robert','Williams','M',6200),

]

columns = ["firstname","lastname","gender","salary"]

df = spark.createDataFrame(data=data, schema = columns)

df.show()

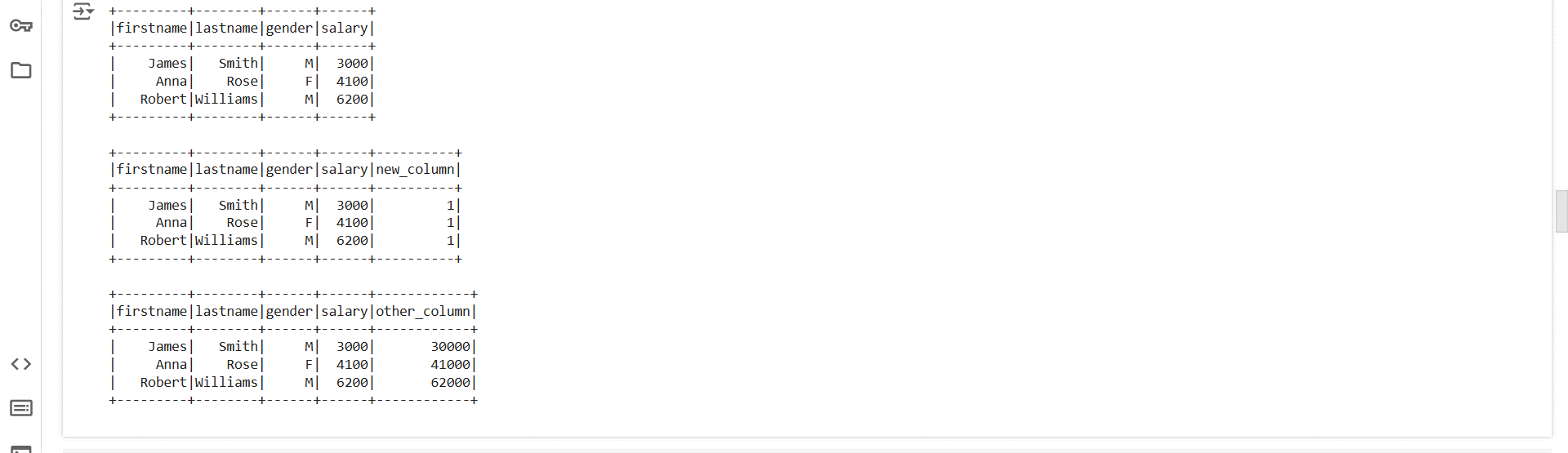
#1.write a program for adding a new column

from pyspark.sql.functions import lit

df.withColumn("new\_column",lit(1)).show()

df.withColumn("other\_column",df.salary\*10).show()

**Output Screenshot:**



**Summary:**

This code creates a DataFrame with employee data and demonstrates how to add new columns:

* A new column new\_column is added with the constant value 1 using lit(1).
* Another column other\_column is created by multiplying the salary column by 10.

The withColumn() method is used to add these columns, and the updated DataFrame is displayed using show().

**7.** **Reading CSV Data and Displaying in PySpark**

#to create rdds and  dataframe

#

from pyspark import SparkContext

from pyspark.sql import  SparkSession

sc =SparkContext.getOrCreate()

spark = SparkSession.builder.appName('pyspark first program').getOrCreate()

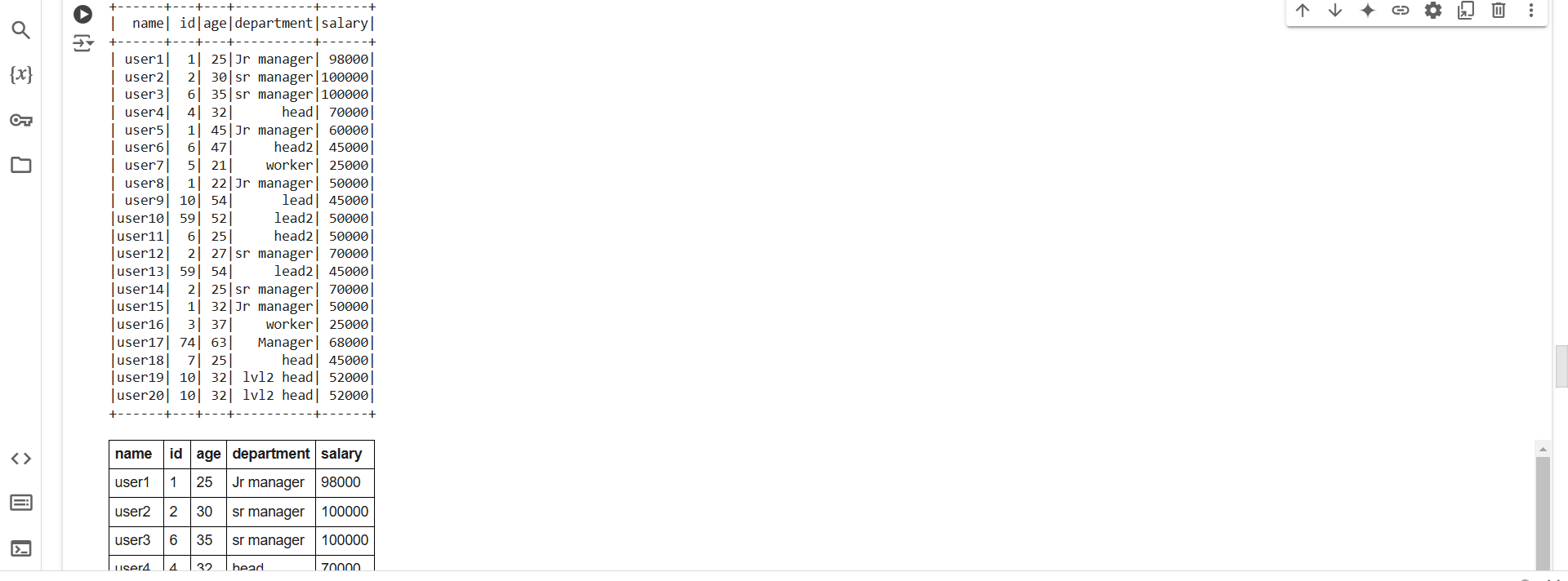
data =spark.read.csv("/FileStore/tables/salary-2.csv",header = True,inferSchema = True)

data.limit(10).toPandas

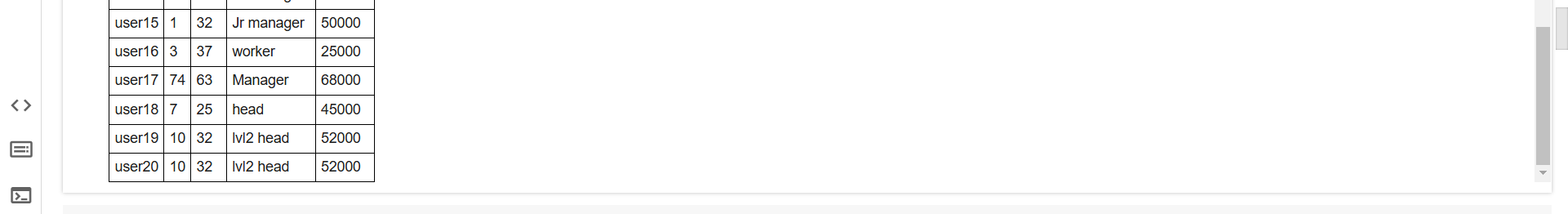
data.show()

display(data)

**Output Screenshot:**







**Summary:**

This code demonstrates how to read a CSV file into a PySpark DataFrame and display the data:

* A SparkContext and SparkSession are initialized using SparkContext.getOrCreate() and SparkSession.builder.appName().
* The spark.read.csv() method is used to read the CSV file (salary-2.csv) into a DataFrame, with header=True to treat the first row as column names, and inferSchema=True to automatically determine the data types of the columns.
* data.limit(10).toPandas limits the DataFrame to the first 10 rows and converts it to a Pandas DataFrame (though it’s missing the () to execute this method).
* data.show() displays the first few rows of the DataFrame in the console.
* display(data) is used to show the DataFrame in a more interactive format, useful in environments like Databricks.

This example shows how to load a CSV file into a DataFrame and preview the data using show() and display().

**8. Converting Pandas DataFrame to Spark DataFrame in PySpark**

#Converting Pandasdf to spark df

from pyspark import SparkContext

from pyspark.sql import  SparkSession

sc =SparkContext.getOrCreate()

spark = SparkSession.builder.appName('pyspark first program').getOrCreate()

import pandas as pd

data = [['Scott', 50], ['Jeff', 45], ['Thomas', 54],['Ann',34]]

# Create the pandas DataFrame

pandasDF = pd.DataFrame(data, columns = ['Name', 'Age'])

# print dataframe.

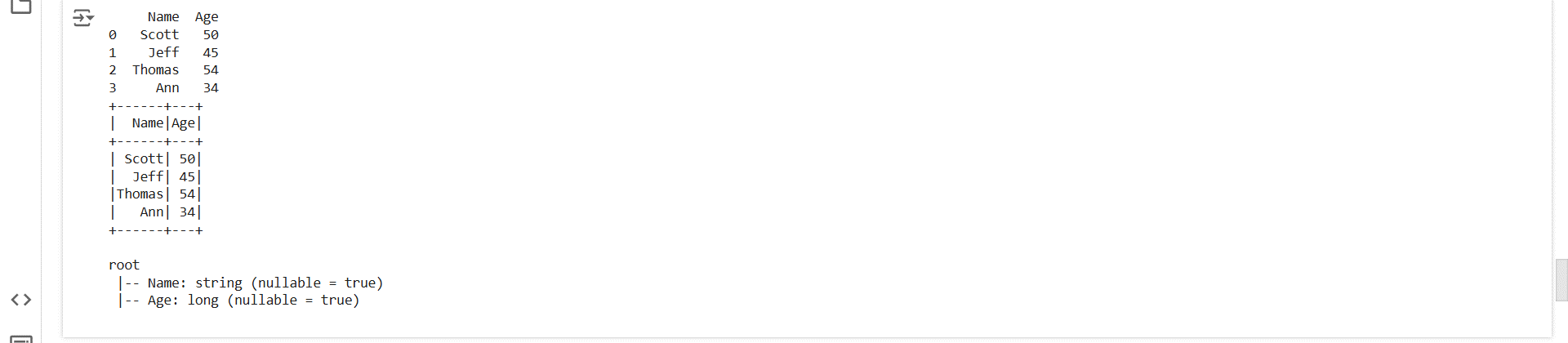
print(pandasDF)

sparkdf =spark.createDataFrame(pandasDF)

sparkdf.show()

sparkdf.printSchema()

**Output Screenshot:**



**Summary:**

This code demonstrates how to convert a Pandas DataFrame into a Spark DataFrame:

* A SparkContext and SparkSession are initialized using SparkContext.getOrCreate() and SparkSession.builder.appName().
* A sample Pandas DataFrame pandasDF is created with data on names and ages using pd.DataFrame().
* spark.createDataFrame(pandasDF) is used to convert the Pandas DataFrame into a Spark DataFrame (sparkdf).
* sparkdf.show() displays the content of the Spark DataFrame.
* sparkdf.printSchema() prints the schema (column names and data types) of the Spark DataFrame.

This example shows how to convert a Pandas DataFrame to a Spark DataFrame using createDataFrame() for distributed data processing in PySpark.

**9. Creating a Spark DataFrame with Custom Schema**

from pyspark.sql.types import StructType,StructField, StringType, IntegerType

mySchema = StructType([ StructField("First Name", StringType(), True)\

                       ,StructField("Age", IntegerType(), True)])

sparkDF2 = spark.createDataFrame(pandasDF,schema=mySchema)

sparkDF2.printSchema()

sparkDF2.show()

**Output Screenshot:**



**Summary:**

This code demonstrates how to create a Spark DataFrame using a custom schema:

* A custom schema mySchema is defined using StructType() and StructField():
  + StructField("First Name", StringType(), True) specifies that the "First Name" column will hold strings and allows null values (True).
  + StructField("Age", IntegerType(), True) defines the "Age" column as integers, also allowing null values.
* The Pandas DataFrame pandasDF is then converted to a Spark DataFrame (sparkDF2) using the createDataFrame() method with the custom schema.
* sparkDF2.printSchema() prints the schema of the newly created DataFrame, showing the column names and types.
* sparkDF2.show() displays the content of the DataFrame.

This example shows how to explicitly define a custom schema for a Spark DataFrame during the conversion from Pandas to Spark.

**10. Enabling Arrow for Efficient Conversion and Converting Spark DataFrame to Pandas DataFrame**

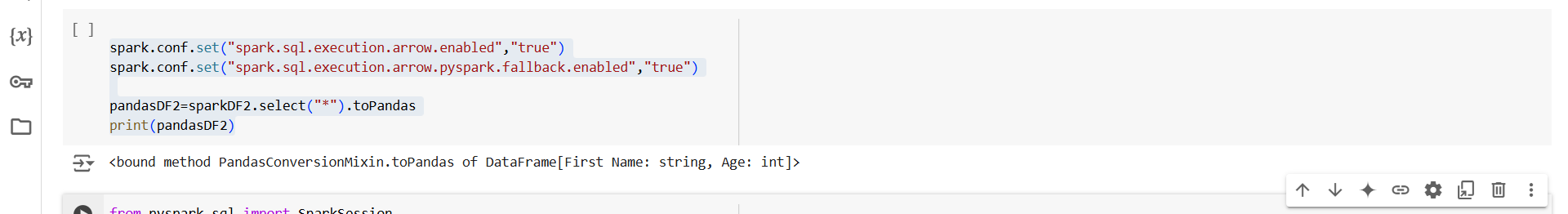
spark.conf.set("spark.sql.execution.arrow.enabled","true")

spark.conf.set("spark.sql.execution.arrow.pyspark.fallback.enabled","true")

pandasDF2=sparkDF2.select("\*").toPandas

print(pandasDF2)

**Output Screenshot:**



**Summary:**

This code demonstrates how to enable Apache Arrow for faster conversion between PySpark DataFrame and Pandas DataFrame, and how to convert a Spark DataFrame to Pandas:

* spark.conf.set("spark.sql.execution.arrow.enabled", "true") enables Apache Arrow, which accelerates the conversion process between PySpark and Pandas DataFrames.
* spark.conf.set("spark.sql.execution.arrow.pyspark.fallback.enabled", "true") ensures that PySpark will fall back to using Arrow even if certain conditions are not met, providing more consistent performance.
* The sparkDF2.select("\*").toPandas() command converts the entire Spark DataFrame (sparkDF2) to a Pandas DataFrame (pandasDF2). However, it is missing the () after toPandas, which would need to be corrected to actually execute the conversion.
* Finally, print(pandasDF2) displays the Pandas DataFrame.

This example highlights the use of Apache Arrow to improve the performance of data conversion between PySpark and Pandas

**11. Using expr and add\_months to Manipulate Dates in PySpark**

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('pyspark\_ex4').getOrCreate()

from pyspark.sql.functions import col,expr

data=[("2019-01-23",1),("2019-06-24",2),("2019-09-20",3)]

spark.createDataFrame(data).toDF("date","increment") \

    .select(col("date"),col("increment"), \

      expr("add\_months(to\_date(date,'yyyy-MM-dd'),cast(increment as int))").alias("inc\_date")) \

    .show()

**Output Screenshot:**



**Summary:**

This code demonstrates how to perform date manipulation in PySpark using expr and add\_months:

* A Spark session is initialized using SparkSession.builder.appName('pyspark\_ex4').getOrCreate().
* Sample data containing dates and increments is created and converted into a DataFrame.
* The select() method is used to select the "date" and "increment" columns from the DataFrame.
* expr("add\_months(to\_date(date,'yyyy-MM-dd'),cast(increment as int))"):
  + to\_date(date, 'yyyy-MM-dd') converts the "date" string to a proper date format.
  + cast(increment as int) casts the "increment" column to an integer.
  + add\_months(date, increment) adds the number of months specified in the "increment" column to the "date" column.
* alias("inc\_date") renames the resulting column as "inc\_date".
* The show() method displays the original date, increment, and the calculated incremented date (inc\_date).

This example demonstrates how to manipulate dates and add months to a date using PySpark SQL functions, improving data transformation and analysis.