

Coding Assesment:

1. Implement Processing JSON and CSV data with PySpark

Syntax

Importing the csv files by using below commands

```
import pyspark
```

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.appName('Reading csv file').getOrCreate()
```

```
file=spark.read.load("path.csv", format="csv") (using the absolute file path to avoid error)
```

```
file.show() - to show the top 20 rows of csv files.
```

```
Administrator: Command Prompt - pyspark
Welcome to
PySpark version 3.5.0

Using Python version 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021 19:10:37)
Spark context Web UI available at http://localhost:4041
Spark context available as 'sc' (master = local[*], app id = local-1703673965462).
SparkSession available as 'spark'.
>>> import pyspark
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder.appName('Reading csv file').getOrCreate()
23/12/27 10:42:11 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
>>> file = spark.read.load("C:/Spark/annual-enterprise-survey-2021-financial-year-provisional-size-bands.csv", format="csv", sep=";", inferSchema="true", header="true")
>>> file.show()

(year, industry_code, ANZSIC, industry_name, ANZSIC, rme_size_grp, variable, value, unit)
-----
[2011] A[Agriculture, Fore... a_0 Activity unit 46134 COUNT
[2011] A[Agriculture, Fore... a_0 Rolling mean empl... 0 COUNT
[2011] A[Agriculture, Fore... a_0 Salaries and wage... 279 DOLLARS(millions)
[2011] A[Agriculture, Fore... a_0 Sales, government... 8187 DOLLARS(millions)
[2011] A[Agriculture, Fore... a_0 Total income 8866 DOLLARS(millions)
[2011] A[Agriculture, Fore... a_0 Total expenditure 7618 DOLLARS(millions)
[2011] A[Agriculture, Fore... a_0 Operating profit ... 770 DOLLARS(millions)
[2011] A[Agriculture, Fore... a_0 Total assets 15700 DOLLARS(millions)
[2011] A[Agriculture, Fore... a_0 Fixed tangible as... 12155 DOLLARS(millions)
[2011] A[Agriculture, Fore... b_1-5 Activity unit 21777 COUNT
[2011] A[Agriculture, Fore... b_1-5 Rolling mean empl... 138136 COUNT
[2011] A[Agriculture, Fore... b_1-5 Salaries and wage... 1435 DOLLARS(millions)
[2011] A[Agriculture, Fore... b_1-5 Sales, government... 13359 DOLLARS(millions)
[2011] A[Agriculture, Fore... b_1-5 Total income 13771 DOLLARS(millions)
[2011] A[Agriculture, Fore... b_1-5 Total expenditure 12316 DOLLARS(millions)
[2011] A[Agriculture, Fore... b_1-5 Operating profit ... 1247 DOLLARS(millions)
[2011] A[Agriculture, Fore... b_1-5 Total assets 15266 DOLLARS(millions)
[2011] A[Agriculture, Fore... b_1-5 Fixed tangible as... 12135 DOLLARS(millions)
[2011] A[Agriculture, Fore... c_6-9 Activity unit 1965 COUNT
[2011] A[Agriculture, Fore... c_6-9 Rolling mean empl... 13848 COUNT

only showing top 20 rows

>>>
```

Using **withColumnRenamed()**:

Used to change the column name

Syntax:

```
file = file.withColumnRenamed("rme_size_grp", "Grp_size")
```

```
file.show()
```

```
Administrator: Command Prompt - pyspark

[2011] | A|Agriculture, Fore... | c_6-9| Activity unit| 1965| COUNT|
[2011] | A|Agriculture, Fore... | c_6-9|Rolling mean empl...|13848| COUNT|
-----+-----+-----+-----+-----+-----+
only showing top 20 rows

>>> file = file.withColumnRename("rme_size_grp","Grp_size")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Spark\spark-3.5.0-bin-hadoop3\python\pyspark\sql\dataframe.py", line 3123, in __getattr__
    raise AttributeError(
AttributeError: "DataFrame" object has no attribute 'withColumnRename'. Did you mean: 'withColumnRenamed'?
>>> file = file.withColumnRenamed("rme_size_grp","Grp_size")
>>> file.show()

+-----+-----+-----+-----+-----+-----+
|year|industry_code_ANZSIC|industry_name_ANZSIC|Grp_size|variable|value|unit|
+-----+-----+-----+-----+-----+-----+
[2011] | A|Agriculture, Fore... | a_0| Activity unit|46134| COUNT|
[2011] | A|Agriculture, Fore... | a_0|Rolling mean empl...| 0| COUNT|
[2011] | A|Agriculture, Fore... | a_0|Salaries and wage...| 279|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | a_0|Sales, government...| 8187|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | a_0| Total income| 8866|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | a_0| Total expenditure| 7618|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | a_0|Operating profit ...| 770|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | a_0| Total assets|55700|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | a_0|Fixed tangible as...|32155|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | b_1-5| Activity unit|21777| COUNT|
[2011] | A|Agriculture, Fore... | b_1-5|Rolling mean empl...|38136| COUNT|
[2011] | A|Agriculture, Fore... | b_1-5|Salaries and wage...|1435|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | b_1-5|Sales, government...|13359|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | b_1-5| Total income|13771|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | b_1-5| Total expenditure|12316|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | b_1-5|Operating profit ...|1247|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | b_1-5| Total assets|52666|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | b_1-5|Fixed tangible as...|31235|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | b_1-5| Activity unit|1965| COUNT|
[2011] | A|Agriculture, Fore... | c_6-9|Rolling mean empl...|13848| COUNT|
-----+-----+-----+-----+-----+-----+
only showing top 20 rows

>>>
```

Using **select** -Selecting the particular column

Syntax:

```
file = file.select('year','Grp_size','unit')
```

```
file.show()
```

```
Administrator: Command Prompt - pyspark

[2011] | A|Agriculture, Fore... | b_1-5|Operating profit ...| 1247|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | b_1-5| Total assets|52666|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | b_1-5|Fixed tangible as...|31235|DOLLARS(millions)|
[2011] | A|Agriculture, Fore... | c_6-9| Activity unit|1965| COUNT|
[2011] | A|Agriculture, Fore... | c_6-9|Rolling mean empl...|13848| COUNT|
-----+-----+-----+-----+-----+
only showing top 20 rows

>>> file = file.select('year','Grp_size','unit')
>>> file.show()

+-----+-----+-----+
|year|Grp_size|unit|
+-----+-----+-----+
[2011] | a_0| COUNT|
[2011] | a_0| COUNT|
[2011] | a_0|DOLLARS(millions)|
[2011] | a_0|DOLLARS(millions)|
[2011] | a_0|DOLLARS(millions)|
[2011] | a_0|DOLLARS(millions)|
[2011] | a_0|DOLLARS(millions)|
[2011] | a_0|DOLLARS(millions)|
[2011] | a_0|DOLLARS(millions)|
[2011] | b_1-5| COUNT|
[2011] | b_1-5| COUNT|
[2011] | b_1-5|DOLLARS(millions)|
[2011] | b_1-5|DOLLARS(millions)|
[2011] | b_1-5|DOLLARS(millions)|
[2011] | b_1-5|DOLLARS(millions)|
[2011] | b_1-5|DOLLARS(millions)|
[2011] | b_1-5|DOLLARS(millions)|
[2011] | b_1-5|DOLLARS(millions)|
[2011] | b_1-5|DOLLARS(millions)|
[2011] | c_6-9| COUNT|
[2011] | c_6-9| COUNT|
-----+-----+-----+
only showing top 20 rows

>>>
```

2.Explain ETL (Extract, Transform, Load) with PySpark

PySpark for ETL:

ETL (Extract, Transform, Load) is a common data processing pattern used in the field of data engineering and analytics. PySpark, being a powerful distributed data processing library, is well-suited for ETL tasks

- Performance
- Ease of Use
- Scalability
- Rich Ecosystem

The PySpark ETL **Workflow**:

Extract: Retrieve data from various sources like databases, files, or APIs.

- Initialize a Spark session.
- Define the external source and target paths.
- Extract: Read data from an CSV file
`df = spark.read.csv(source_path, header=True,schema = 'column names').`

Transform: Clean, aggregate, and manipulate data to fit your analysis needs.

- Using `concat()`,`filter()`,adding a column,`group by` ,`avg()`,`orderby()`,`floor()`
- `map()`,`sortby()`,`union()`,`intersection()`
- `coalesce()`,`zip()`,`subtract()`,`partitionby()`

Some examples are, -using `groupby()`,`filter()`,`max()`,`min()`

```
Administrator Command Prompt - pyspark
[DOLLARS(millions)] d_10-19 NULL [DOLLARS(millions)]
[DOLLARS(millions)] e_20-49 NULL [DOLLARS(millions)]
[DOLLARS(millions)] f_50-99 NULL [DOLLARS(millions)]
[DOLLARS(millions)] g_100-199 NULL [DOLLARS(millions)]
[DOLLARS(millions)] h_200+ NULL [DOLLARS(millions)]
[DOLLARS(millions)] i_Industry_Total NULL [DOLLARS(millions)]
[DOLLARS(millions)] j_Grand_Total NULL [DOLLARS(millions)]
-----
>>> file.sort("unit").show()
+-----+-----+-----+
|year|Grp_size|unit|
+-----+-----+-----+
|2011|d_10-19|COUNT|
|2011|e_20-49|COUNT|
|2011|d_10-19|COUNT|
|2011|a_0|COUNT|
|2011|e_20-49|COUNT|
|2011|h_200+|COUNT|
|2011|d_10-19|COUNT|
|2011|i_Industry_Total|COUNT|
|2011|b_1-5|COUNT|
|2011|i_Industry_Total|COUNT|
|2011|c_6-9|COUNT|
|2011|a_0|COUNT|
|2011|e_20-49|COUNT|
|2011|a_0|COUNT|
|2011|f_50-99|COUNT|
|2011|b_1-5|COUNT|
|2011|g_100-199|COUNT|
|2011|b_1-5|COUNT|
|2011|b_1-5|COUNT|
|2011|c_6-9|COUNT|
+-----+-----+-----+
only showing top 20 rows

>>> file.filter((file.unit=='COUNT') & (file.year>2000)).show()
+-----+-----+-----+
|year|Grp_size|unit|
+-----+-----+-----+
|2011|a_0|COUNT|
|2011|a_0|COUNT|
|2011|b_1-5|COUNT|
|2011|b_1-5|COUNT|
|2011|c_6-9|COUNT|
|2011|c_6-9|COUNT|
|2011|d_10-19|COUNT|
|2011|d_10-19|COUNT|
|2011|e_20-49|COUNT|
|2011|e_20-49|COUNT|
+-----+-----+-----+
```

Load: Store the transformed data into a database or data warehouse for analysis.

- Save the transformed data to an CSV file
`df.write.csv(target_path, mode="overwrite", header=True)`

4.Using Spark SQL - Transformations such as Filter, Join, Simple Aggregations, GroupBy.

Group by()

To group data in a DataFrame based on one or more columns, and then perform aggregate functions on each group.

Syntax :

from pyspark.sql import functions as F

file.groupBy(["unit", "Grp_size"]).agg(F.sum("unit"), F.max("unit")).show()

```
Administrator: Command Prompt - pyspark
File "<stdin>", line 1
  file.groupBy(["unit", "Grp_size"]).agg(F.sum("unit"), F.max("unit")).show()
IndentationError: unexpected indent
>>> file.groupBy(["unit", "Grp_size"]).agg(F.sum("unit"), F.max("unit")).show()
+-----+
|unit|    Grp_size|sum(unit)|max(unit)|
+-----+
|COUNT|    a_0|NULL|COUNT|
|COUNT|  b_1-5|NULL|COUNT|
|COUNT|  c_6-9|NULL|COUNT|
|COUNT| d_10-19|NULL|COUNT|
|COUNT| e_20-49|NULL|COUNT|
|COUNT| f_50-99|NULL|COUNT|
|COUNT| g_100-199|NULL|COUNT|
|COUNT| h_200+|NULL|COUNT|
|COUNT|i_Industry_Total|NULL|COUNT|
|COUNT|j_Grand_Total|NULL|COUNT|
|DOLLARS(millions)|a_0|NULL|DOLLARS(millions)|
|DOLLARS(millions)|b_1-5|NULL|DOLLARS(millions)|
|DOLLARS(millions)|c_6-9|NULL|DOLLARS(millions)|
|DOLLARS(millions)|d_10-19|NULL|DOLLARS(millions)|
|DOLLARS(millions)|e_20-49|NULL|DOLLARS(millions)|
|DOLLARS(millions)|f_50-99|NULL|DOLLARS(millions)|
|DOLLARS(millions)|g_100-199|NULL|DOLLARS(millions)|
|DOLLARS(millions)|h_200+|NULL|DOLLARS(millions)|
|DOLLARS(millions)|i_Industry_Total|NULL|DOLLARS(millions)|
|DOLLARS(millions)|j_Grand_Total|NULL|DOLLARS(millions)|
+-----+
>>>
```

Sorting()

To sort the rows in a DataFrame based on one or more columns.

syntax:

file.sort("unit").show()

```
Administrator: Command Prompt - pyspark
|DOLLARS(millions)|c_6-9|NULL|DOLLARS(millions)|
|DOLLARS(millions)|d_10-19|NULL|DOLLARS(millions)|
|DOLLARS(millions)|e_20-49|NULL|DOLLARS(millions)|
|DOLLARS(millions)|f_50-99|NULL|DOLLARS(millions)|
|DOLLARS(millions)|g_100-199|NULL|DOLLARS(millions)|
|DOLLARS(millions)|h_200+|NULL|DOLLARS(millions)|
|DOLLARS(millions)|i_Industry_Total|NULL|DOLLARS(millions)|
|DOLLARS(millions)|j_Grand_Total|NULL|DOLLARS(millions)|
+-----+
>>> file.sort("unit").show()
+year|    Grp_size|unit|
+----+-----+----+
|2011|d_10-19|COUNT|
|2011|e_20-49|COUNT|
|2011|d_10-19|COUNT|
|2011|a_0|COUNT|
|2011|e_20-49|COUNT|
|2011|h_200+|COUNT|
|2011|d_10-19|COUNT|
|2011|i_Industry_Total|COUNT|
|2011|b_1-5|COUNT|
|2011|i_Industry_Total|COUNT|
|2011|c_6-9|COUNT|
|2011|a_0|COUNT|
|2011|e_20-49|COUNT|
|2011|a_0|COUNT|
|2011|f_50-99|COUNT|
|2011|b_1-5|COUNT|
|2011|g_100-199|COUNT|
|2011|b_1-5|COUNT|
|2011|b_1-5|COUNT|
|2011|c_6-9|COUNT|
+-----+
only showing top 20 rows
>>>
```

Filter-To filter rows in a DataFrame based on a specified condition.

syntax-file.filter((file.unit=='COUNT') & (file.year>2000)).show()

```
Administrator Command Prompt - pyspark
[2011]      a_0|COUNT|
[2011]      f_50-99|COUNT|
[2011]      b_1-5|COUNT|
[2011]      g_100-199|COUNT|
[2011]      b_1-5|COUNT|
[2011]      b_1-5|COUNT|
[2011]      c_6-9|COUNT|
-----
only showing top 20 rows

>>> file.filter((file.unit=='COUNT') & (file.year>2000)).show()
-----
[year]      Grp_size|unit|
[2011]      a_0|COUNT|
[2011]      a_0|COUNT|
[2011]      b_1-5|COUNT|
[2011]      b_1-5|COUNT|
[2011]      c_6-9|COUNT|
[2011]      c_6-9|COUNT|
[2011]      d_10-19|COUNT|
[2011]      d_10-19|COUNT|
[2011]      e_20-49|COUNT|
[2011]      e_20-49|COUNT|
[2011]      f_50-99|COUNT|
[2011]      f_50-99|COUNT|
[2011]      g_100-199|COUNT|
[2011]      g_100-199|COUNT|
[2011]      h_200+|COUNT|
[2011]      h_200+|COUNT|
[2011] i_Industry_Total|COUNT|
[2011] i_Industry_Total|COUNT|
[2011]      a_0|COUNT|
[2011]      a_0|COUNT|
-----
only showing top 20 rows

>>>
```

Aggregation

aggregation operations on a DataFrame to summarize or compute statistics on the data.

Min,max,count,avg

syntax

file.groupBy(["unit","Grp_size"]).agg(F.sum("unit"),F.min("unit")).show()

```
Administrator Command Prompt - pyspark
[2011]      e_20-49|COUNT|
[2011]      e_20-49|COUNT|
[2011]      f_50-99|COUNT|
[2011]      f_50-99|COUNT|
[2011]      g_100-199|COUNT|
[2011]      g_100-199|COUNT|
[2011]      h_200+|COUNT|
[2011]      h_200+|COUNT|
[2011] i_Industry_Total|COUNT|
[2011] i_Industry_Total|COUNT|
[2011]      a_0|COUNT|
[2011]      a_0|COUNT|
-----
only showing top 20 rows

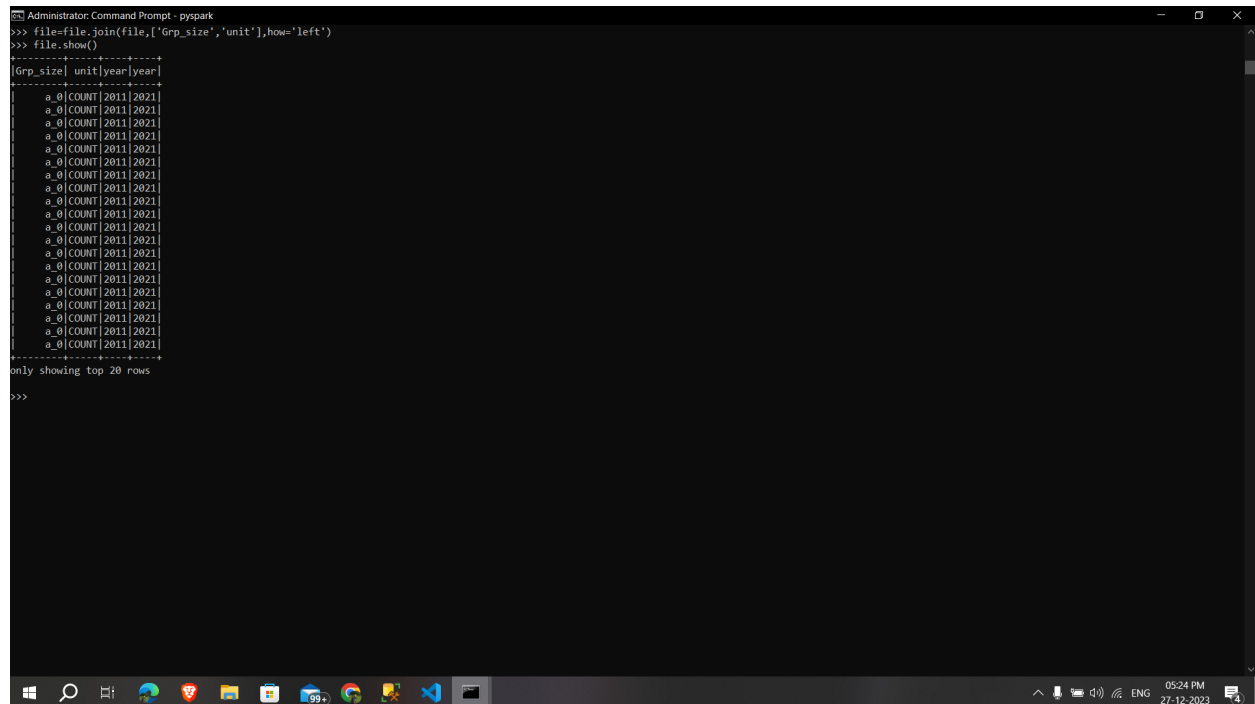
>>> file.groupBy(["unit","Grp_size"]).agg(F.sum("unit"),F.min("unit")).show()
File "stdin", line 1
file.groupBy(["unit","Grp_size"]).agg(F.sum("unit"),F.min("unit")).show()
IndentationError: unexpected indent
>>> file.groupBy(["unit","Grp_size"]).agg(F.sum("unit"),F.min("unit")).show()
-----
[unit]      Grp_size|sum(unit)|      min(unit)|
-----
COUNT|      a_0|      NULL|      COUNT|
COUNT|      b_1-5|      NULL|      COUNT|
COUNT|      c_6-9|      NULL|      COUNT|
COUNT|      d_10-19|      NULL|      COUNT|
COUNT|      e_20-49|      NULL|      COUNT|
COUNT|      f_50-99|      NULL|      COUNT|
COUNT|      g_100-199|      NULL|      COUNT|
COUNT|      h_200+|      NULL|      COUNT|
COUNT| i_Industry_Total|      NULL|      COUNT|
COUNT| j_Grand_Total|      NULL|      COUNT|
(DOLLARS(millions))|      a_0|      NULL|(DOLLARS(millions))|
(DOLLARS(millions))|      b_1-5|      NULL|(DOLLARS(millions))|
(DOLLARS(millions))|      c_6-9|      NULL|(DOLLARS(millions))|
(DOLLARS(millions))|      d_10-19|      NULL|(DOLLARS(millions))|
(DOLLARS(millions))|      e_20-49|      NULL|(DOLLARS(millions))|
(DOLLARS(millions))|      f_50-99|      NULL|(DOLLARS(millions))|
(DOLLARS(millions))|      g_100-199|      NULL|(DOLLARS(millions))|
(DOLLARS(millions))|      h_200+|      NULL|(DOLLARS(millions))|
(DOLLARS(millions))| i_Industry_Total|      NULL|(DOLLARS(millions))|
(DOLLARS(millions))| j_Grand_Total|      NULL|(DOLLARS(millions))|
-----

>>> files.limit(10).toPandas()
Traceback (most recent call last):
  File "stdin", line 1, in <module>
NameError: name 'files' is not defined. Did you mean: 'file'?
>>> file.limit(10).toPandas()
```

Joins - DataFrames to combine data from different sources based on common columns.

Syntax

```
file=file.join(file,['Grp_size','unit'],how='left')  
file.show()
```



```
Administrator: Command Prompt - pyspark  
>>> file=file.join(file,['Grp_size','unit'],how='left')  
>>> file.show()  
+-----+-----+-----+-----+  
|Grp_size|unit|year|year|  
+-----+-----+-----+-----+  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
|a_0|COUNT|2011|2021|  
+-----+-----+-----+-----+  
only showing top 20 rows  
>>>
```

3.Using Spark SQL - Creating databases, tables

Database:

A database in Spark is a namespace for tables. It helps organize and categorize tables based on their purpose or data domain.

Table:

- In Spark, tables are representations of structured data. They are organized into databases, providing a logical structure for querying and analyzing data.
- Tables can be created from DataFrames, existing RDDs, or external data sources.

Using RDD:

- By using **parallelize()** function
- By using **createDataFrame()** function:
- By using **read and load** functions:
 1. Read dataset from **.csv file**
 2. Read dataset from **DataBase**
 3. Read dataset from **HDFS**

The PySpark library to create databases and tables. PySpark provides a SQL API that allows you to interact with Spark using SQL-like syntax.

```
from pyspark.sql import SparkSession
# Create a Spark session
spark = SparkSession.builder.appName("SparkPythonExample") .getOrCreate()
# Sample data
data = [("Manoj", 25), ("karan", 30), ("hari", 22)]
# Define schema for the DataFrame
columns = ["name", "age"]
# Create a DataFrame
df = spark.createDataFrame(data, columns)
# Show the DataFrame
df.show()
# Save the DataFrame as a table
table_name = "people"
df.write.saveAsTable(table_name)
# Query the table
result = spark.sql(f"SELECT * FROM {table_name}")
# Show the result
result.show()
```

Explanation:

- We create a Spark session.
- We define some sample data in the form of a list of tuples.
- We create a PySpark DataFrame (df) from the sample data with specified column names.
- We display the DataFrame using df.show().
- We save the DataFrame as a table using df.write.saveAsTable(table_name).
- We execute a SQL query using spark.sql to select all data from the table.
- Finally, we display the result of the SQL query using result.show().