Setting up PySpark envioremnent in Colab

In [ ]:

```
!pip install pyspark py4j
```

Loading Fake Data CSV into Spark DataFrame

The loading Phase came from PySpark_HW1 because when I tried to use the `spark.read.csv()` from PySpark_HW3 it wasn't loading correctly. I had columns like: C_0 (_c0), C_1 (_c0) etc. Due to this, I decided to use the `spark.read.option('header', 'true')` to get the first rows as the DataFrame columns.

In [2]:

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

In [3]:

```
spark = SparkSession.builder.appName('DataFrame').getOrCreate()
```

In [4]:

```
#df = spark.read.csv('Fake_data.csv') # HW3 csv reading option (not working)
#df.show(truncate = False)

df = spark.read.option('header', 'true').csv('Fake_data.csv')
df.show(n=5, truncate=False)
```

```
+---+----------------------+------------------------------+----------+------+------------------------------------+---------+-------------+-----------+
|_c0|Birth_Country         |Email                         |First_Name|Income|Job                                 |Last_name|Loan_Approved|SSN        |
+---+----------------------+------------------------------+----------+------+------------------------------------+---------+-------------+-----------+
|0  |Bosnia and Herzegovina|emily15@whitehead.com         |Melissa   |109957|Telecommunications researcher       |Miranda  |False        |129-41-7773|
|1  |Belgium               |ronald87@yahoo.com            |Curtis    |301884|Animal nutritionist                 |Garrett  |True         |212-74-3976|
|2  |United Kingdom        |hannah29@gmail.com            |Connor    |341584|English as a foreign language teacher|Steele   |False        |024-35-3834|
|3  |Kiribati              |derrick59@hotmail.com         |Adam      |448293|Surveyor, commercial/residential    |Newman   |False        |157-82-4486|
|4  |Malaysia              |wendycarpenter@walker-knox.com|Jared     |53621 |Drilling engineer                   |Mann     |True         |199-56-2824|
+---+----------------------+------------------------------+----------+------+------------------------------------+---------+-------------+-----------+
only showing top 5 rows
```

In [5]:

```
df.printSchema() # This helps understanding the DF structure
```

```
root
 |-- _c0: string (nullable = true)
 |-- Birth_Country: string (nullable = true)
 |-- Email: string (nullable = true)
 |-- First_Name: string (nullable = true)
 |-- Income: string (nullable = true)
 |-- Job: string (nullable = true)
 |-- Last_name: string (nullable = true)
 |-- Loan_Approved: string (nullable = true)
 |-- SSN: string (nullable = true)
```

Transformations & Actions:

`Transformation` : Transformation refers to the operation applied on a RDD to create new RDD. Filter, groupBy and map are the examples of transformations.

`Actions` : Actions refer to an operation which also applies on RDD, that instructs Spark to perform computation and send the result back to driver.

I am going to be creating a temporary table named "Fake_data_temp" so I can practice with the Spark SQL API and use SQL Syntax. The idea behind this was to use a couple of things from last homework and to also practice some SQL syntax. It is important to mentioned that many a few syntax/notes (SQLite Syntax) were taken from "Applied Database Technologies" class with Professor Olga Scrivner. PySpark with SQL API HW4 uses common SQL Syntax to achieve results.

Question 1

Find birth country which has highest amount of people.

In [6]:

```
def_temp = df.createOrReplaceTempView('Fake_data_temp')
```

In [7]:

```
# Checking if temporary table exists or not
table_exists = spark.catalog.tableExists('Fake_data_temp')

if table_exists:
    print('Fake_data_temp exists')
else:
    print('Fake_data_temp does not exist')
```

```
Fake_data_temp exists
```

SQLite Syntax example from Applied Database Technologies: It doesn't matter if you decide to the write the syntax in caps lock or not.

```
# * Just for me to practice (I like the aesthetic of DataFrames in Jupyter No
# We can also print column names with values as a pd df
c.execute('''
select All_weekly,M_weekly,F_weekly
    from income;
''')
```

`query_1` shows the implementation of spark.sql and how do we select and count all records in Birth_Country, after that we assign the alias Total_Country to it. Then we proceed to group be Birth_Country and order it by Total_Country in descending order. Limit 1 is used to get the first row of the sorted Total_Country DESC. It's also worth mentioning that the first() function will return the first value of the selected column.

In [23]:

```
query_1 = spark.sql('''
      SELECT Birth_Country, COUNT(*) AS Total_Country
      FROM Fake_data_temp
      GROUP BY Birth_Country
      ORDER By Total_Country DESC
      LIMIT 1;
''')
```

In [21]:

```
query_1.show(truncate=False)
```

```
+-------------+-------------+
|Birth_Country|Total_Country|
+-------------+-------------+
|Korea        |91           |
+-------------+-------------+
```

In [24]:

```
query1 = query_1.first()['Birth_Country'] # get the first value of the Birth_Country column from first row of query_1

print('Country with the highest amount of people is:', query1)
```

```
Country with the highest amount of people is: Korea
```

In [19]:

```
query1_sql = query_1.first()[0]
print('Country with the highest amount of people is:', query1_sql)
```

```
Country with the highest amount of people is: Korea
```

Question 2

Find Maximum income in each country and display the countries from highest to lowest Income.

The code `query_2` is straight forward, we select Birth_Country and Income from the DataFrame and assigned an alias to the MAX(Income) names Maximum_Income. We group by Birth_Country and order by Maximum_Income in descending order. We store the new DataFrame in a variable called maximum_income_df and show results from highest to lowest.

In [11]:

```
query_2 = spark.sql('''
      SELECT Birth_Country, MAX(Income) AS Maximum_Income
      FROM Fake_data_temp
      GROUP BY Birth_Country
      ORDER BY Maximum_Income DESC;
''')

maximum_income_df = query_2 # storing query_2 dataframe into new variable

maximum_income_df.show(truncate=False)
```

```
+--------------------------------------------+--------------+
|Birth_Country                               |Maximum_Income|
+--------------------------------------------+--------------+
|Denmark                                     |99900         |
|Madagascar                                  |99895         |
|Cayman Islands                              |99731         |
|Chad                                        |99721         |
|Angola                                      |9961          |
|Falkland Islands (Malvinas)                 |9939          |
|Korea                                       |99374         |
|United States Minor Outlying Islands        |99334         |
|Uganda                                      |99315         |
|Brazil                                      |99311         |
|Antarctica (the territory South of 60 deg S)|99309         |
|Nepal                                       |99284         |
|Oman                                        |99283         |
|Romania                                     |99260         |
|Holy See (Vatican City State)               |99259         |
|Guernsey                                    |99212         |
|Haiti                                       |99209         |
|Bermuda                                     |99183         |
|United Kingdom                              |98970         |
|Seychelles                                  |98950         |
+--------------------------------------------+--------------+
only showing top 20 rows
```

Question 3

How many people has income over 100,000 but their loan is approved.

`query_3` follows kind of the same approach as query_1, but in much simpler way. We select the count of the records satisfied by the "where" clause and assign it to alias People. The "and" clause is the expand the task and find the Income greater than 100000 and Loan_Approved equals to True to get the result. Just like query_1, first() function is implemented to get the first value of the column People.

In [25]:

```
query_3 = spark.sql('''
      SELECT COUNT(*) AS People
      FROM Fake_data_temp
      WHERE Income > 100000
      AND Loan_Approved = 'True';
''')
```

In [26]:

```
query_3.show(truncate=False)
```

```
+------+
|People|
+------+
|3958  |
+------+
```

In [27]:

```
query3 = query_3.first()['People'] # get the first value of the People column from first row of query_3

print('How many people has income over 100,000 but their loan is approved?', query3, 'people.')
```

```
How many people has income over 100,000 but their loan is approved? 3958 people.
```

In [28]:

```
query3_sql = query_3.first()[0]

print('How many people has income over 100,000 but their loan is approved?', query3_sql, 'people.')
```

```
How many people has income over 100,000 but their loan is approved? 3958 people.
```

Question 4

Find top 10 people with highest income in United States of America. (Print their names, income and jobs from the highest income to lowest).

`query4` takes on the name (incluing Last_Name), Income, and Job from the temporary table Fake_data_temp to get the top 10 people with highest income in the USA. To achieve this we need to use the "where" clause to set the Birth_Country to 'United States of America' and order by income in descending order. Limit to 10 because we just want the top 10 people.

In [ ]:

```
query_4 = spark.sql('''
      SELECT First_Name, Last_Name, Income, Job
      FROM Fake_data_temp
      WHERE Birth_Country = 'United States of America'
      ORDER BY Income DESC
      LIMIT 10;
''')

query4 = query_4 # storing query_4 dataframe into new variable

query4.show(truncate=False)
```

```
+-----------+---------+------+-----------------------------------------+
|First_Name |Last_Name|Income|Job                                      |
+-----------+---------+------+-----------------------------------------+
|Bobby      |Lopez    |7477  |Dietitian                                |
|John       |Patterson|74237 |Facilities manager                       |
|Arthur     |Thompson |74114 |Electronics engineer                     |
|Christopher|Gonzales |71967 |Midwife                                  |
|Dorothy    |Hart     |68837 |Magazine features editor                 |
|Alejandro  |Hernandez|66730 |Warden/ranger                            |
|Martha     |Vargas   |62632 |Illustrator                              |
|Alyssa     |Miller   |482588|Amenity horticulturist                   |
|Tiffany    |Meyer    |47759 |Solicitor, Scotland                      |
|Hunter     |Walls    |468946|Psychologist, prison and probation services|
+-----------+---------+------+-----------------------------------------+
```

Question 5

How many numbers of distinct jobs are there? Again, `query5` follows the basic SQL Syntax to calculate the number of distinct jobs out there. We select and count the distinct Job and assing an alias to it named "Distnict_Job". Then we call the first value of this selected column to get the result.

In [16]:

```
query_5 = spark.sql('''
      SELECT COUNT(DISTINCT Job) AS Distinct_Job
      FROM Fake_data_temp;
''')
```

```
How many number of distinct jobs are there? 639 jobs.
```

In [30]:

```
query_5.show(truncate=False)
```

```
+------------+
|Distinct_Job|
+------------+
|639         |
+------------+
```

In [29]:

```
query5 = query_5.first()['Distinct_Job'] # get the first value of the Distinct_Job column from first row of query_5

print('How many number of distinct jobs are there?', query5,'jobs.')
```

```
How many number of distinct jobs are there? 639 jobs.
```

In [31]:

```
query5_sql = query_5.first()[0]

print('How many number of distinct jobs are there?', query5_sql,'jobs.')
```

```
How many number of distinct jobs are there? 639 jobs.
```

Question 6

How many writers earn less than 100,000? `query6` calculates the total count of the rows where Job == 'Writer' and Income is less than $100,000. Again, we use the first() function to obtain the first value fo the selected column -> Writer.

In [32]:

```
query_6 = spark.sql('''
      SELECT COUNT(*) AS Writers
      FROM Fake_data_temp
      WHERE Job = 'Writer'
      AND Income < 100000;
''')
```

In [33]:

```
query_6.show(truncate=False)
```

```
+-------+
|Writers|
+-------+
|5      |
+-------+
```

In [34]:

```
query6 = query_6.first()['Writers'] # get the first value (count of writers) of the Writers column from first row of query_6
print('How many writers earn less than 100,000?', query6)
```

```
How many writers earn less than 100,000? 5
```

In [35]:

```
query6_sql = query_6.first()[0]
print('How many writers earn less than 100,000?', query6_sql)
```

```
How many writers earn less than 100,000? 5
```

Question 7

List the Top 5 countries with the count which have the highest persons with their first name "Stephen" or "Alexandra" and whose salary is greater then 200000.

This code it's a more elaborated than the others, however it still uses the basics of SQL Syntax to achieve results. `query_7` code calculates the total count of persons in each country who satisfied the filtering results. We assign these results to the column Highest_Person_Count using alias. We also use the WHERE, OR, AND to filter results. Finally, we group by Birth_Country and we order in descending order the Highest_Person_Count. We limit to 5 since we only want the top 5 countries results.

In [ ]:

```
query_7 = spark.sql('''
      SELECT Birth_Country, COUNT(*) AS Highest_Person_Count
      FROM Fake_data_temp
      WHERE First_Name = 'Stephen'
      OR First_Name = 'Alezander'
      AND Income > 200000
      GROUP BY Birth_Country
      ORDER BY Highest_Person_Count DESC
      LIMIT 5;
''')

query_7.show()
```

```
+--------------+--------------------+
| Birth_Country|Highest_Person_Count|
+--------------+--------------------+
|       Algeria|                   2|
|Cayman Islands|                   2|
|       Tokelau|                   1|
|        Guyana|                   1|
|      Djibouti|                   1|
+--------------+--------------------+
```

Question 8

Check if we have people with same SSN number and if they have , print the SSN Number.

`query_8` continues with the basic SQL Syntax usage, we select SSN and calculate the total count of persons who contain the same SSN. We group by SSN and use HAVING COUNT(*) > 1) to filter the results than contain SSN duplicates.

In [ ]:

```
query_8 = spark.sql('''
      SELECT SSN, COUNT(*) AS SSN_Check
      FROM Fake_data_temp
      GROUP BY SSN
      HAVING COUNT(*) > 1;
''')


query_8.show()
```

```
+-----------+---------+
|        SSN|SSN_Check|
+-----------+---------+
|879-74-5521|        2|
+-----------+---------+
```

References:

- Checking for temporary table: https://stackoverflow.com/questions/58067388/how-to-check-if-a-hive-table-exists-using-pyspark
- SQL Queries in PySpark: https://medium.datadriveninvestor.com/pyspark-sql-and-dataframes-4c821615eafe
- PySpark Transformations and Actions Guide: https://www.analyticsvidhya.com/blog/2016/10/using-pyspark-to-perform-transformations-and-actions-on-rdd/
- PySpark Transformations: https://sparkbyexamples.com/pyspark/pyspark-rdd-transformations/
- SQL AGG and Scalar functions: https://www.geeksforgeeks.org/sql-functions-aggregate-scalar-functions/
- SQL Tutorial/Functions: http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/sql/sql_alias.asp.html#gsc.tab=0
- HAVING COUNT (*) > 1: https://stackoverflow.com/questions/30800889/retrieving-the-duplicate-ssn-from-the-huge-number-of-employess-all-of-them
- Useful SQL SYNTAX: https://stackoverflow.com/questions/5391564/how-to-use-distinct-and-order-by-in-same-select-statement