In [ ]:

Final Project: Data Analysis using Spark

Estimated time needed: **60** minutes

This final project is similar to the Practice Project you did. In this project, you will not be provided with hints or solutions. You will create a DataFrame by loading data from a CSV file and apply transformations and actions using Spark SQL. This needs to be achieved by performing the following tasks:

- Task 1: Generate DataFrame from CSV data.
- Task 2: Define a schema for the data.
- Task 3: Display schema of DataFrame.
- Task 4: Create a temporary view.
- Task 5: Execute an SQL query.
- Task 6: Calculate Average Salary by Department.
- Task 7: Filter and Display IT Department Employees.
- Task 8: Add 10% Bonus to Salaries.
- Task 9: Find Maximum Salary by Age.
- Task 10: Self-Join on Employee Data.
- Task 11: Calculate Average Employee Age.
- Task 12: Calculate Total Salary by Department.
- Task 13: Sort Data by Age and Salary.
- Task 14: Count Employees in Each Department.
- Task 15: Filter Employees with the letter o in the Name.

Prerequisites

1. For this lab assignment, you will be using Python and Spark (PySpark). Therefore, it's essential to make sure that the following libraries are installed in your lab environment or within Skills Network (SN) Labs

In [1]:
```
# Installing required packages

!pip install pyspark  findspark wget
```

```
Collecting pyspark
  Downloading pyspark-3.4.3.tar.gz (311.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 311.4/311.4 MB 929.5 kB/s eta 0:00:0000:0100:01
  Preparing metadata (setup.py) ... done
Collecting findspark
  Downloading findspark-2.0.1-py2.py3-none-any.whl (4.4 kB)
Requirement already satisfied: wget in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (3.2)
Collecting py4j==0.10.9.7 (from pyspark)
  Downloading py4j-0.10.9.7-py2.py3-none-any.whl (200 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.5/200.5 kB 7.8 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.4.3-py2.py3-none-any.whl size=311885504 sha256=9178c863fce6
34ea52832e715a394bad9825869a05d4616c76d28f44ab7724b5
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/37/bc/bb/77785f6fcd2c83e663647f73225b76f3a3d5fd00
762d7daf6f
Successfully built pyspark
Installing collected packages: py4j, findspark, pyspark
Successfully installed findspark-2.0.1 py4j-0.10.9.7 pyspark-3.4.3
```

In [2]:
```
import findspark

findspark.init()
```

In [3]:
```
# PySpark is the Spark API for Python. In this lab, we use PySpark to initialize the SparkContext.

from pyspark import SparkContext, SparkConf

from pyspark.sql import SparkSession
```

In [4]:
```
# Creating a SparkContext object

sc = SparkContext.getOrCreate()

# Creating a SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark DataFrames basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

```
24/05/02 15:30:23 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... usi
ng builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
```

2. Download the CSV data.

In [5]:
```
# Download the CSV data first into a local `employees.csv` file
import wget
wget.download("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0225EN-SkillsNetwo
```

Out[5]:
```
'employees (2).csv'
```

Tasks

Task 1: Generate a Spark DataFrame from the CSV data

Read data from the provided CSV file, `employees.csv` and import it into a Spark DataFrame variable named `employees_df`.

In [6]:
```
# Read data from the "emp" CSV file and import it into a DataFrame variable named "employees_df"
employees_df = spark.read.csv("employees.csv", header=True, inferSchema=True)
```

Task 2: Define a schema for the data

Construct a schema for the input data and then utilize the defined schema to read the CSV file to create a DataFrame named `employees_df`.

In [7]:
```
# Define a Schema for the input data and read the file using the user-defined Schema
employees_df = spark.read.format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load("employees.csv")
```

Task 3: Display schema of DataFrame

Display the schema of the `employees_df` DataFrame, showing all columns and their respective data types.

In [8]:
```
# Display all columns of the DataFrame, along with their respective data types
employees_df.printSchema()
```

```
root
 |-- Emp_No: integer (nullable = true)
 |-- Emp_Name: string (nullable = true)
 |-- Salary: integer (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Department: string (nullable = true)
```

Task 4: Create a temporary view

Create a temporary view named `employees` for the `employees_df` DataFrame, enabling Spark SQL queries on the data.

In [10]:
```
# Create a temporary view named "employees" for the DataFrame
employees_df.createTempView("employees")
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
~/spark-2.4.3/python/pyspark/sql/utils.py in deco(*a, **kw)
     62         try:
---> 63             return f(*a, **kw)
     64         except py4j.protocol.Py4JJavaError as e:

~/spark-2.4.3/python/lib/py4j-0.10.7-src.zip/py4j/protocol.py in get_return_value(answer, gateway_client, t
arget_id, name)
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:

Py4JJavaError: An error occurred while calling o43.createTempView.
: org.apache.spark.sql.catalyst.analysis.TempTableAlreadyExistsException: Temporary view 'employees' alread
y exists;
        at org.apache.spark.sql.catalyst.catalog.SessionCatalog.createTempView(SessionCatalog.scala:495)
        at org.apache.spark.sql.execution.command.CreateViewCommand.run(views.scala:146)
        at org.apache.spark.sql.execution.command.ExecutedCommandExec.sideEffectResult$lzycompute(commands.
scala:70)
        at org.apache.spark.sql.execution.command.ExecutedCommandExec.sideEffectResult(commands.scala:68)
        at org.apache.spark.sql.execution.command.ExecutedCommandExec.executeCollect(commands.scala:79)
        at org.apache.spark.sql.Dataset$$anonfun$6.apply(Dataset.scala:194)
        at org.apache.spark.sql.Dataset$$anonfun$6.apply(Dataset.scala:194)
        at org.apache.spark.sql.Dataset$$anonfun$53.apply(Dataset.scala:3364)
        at org.apache.spark.sql.execution.SQLExecution$$anonfun$withNewExecutionId$1.apply(SQLExecution.sca
la:78)
        at org.apache.spark.sql.execution.SQLExecution$.withSQLConfPropagated(SQLExecution.scala:125)
        at org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExecution.scala:73)
        at org.apache.spark.sql.Dataset.withAction(Dataset.scala:3363)
        at org.apache.spark.sql.Dataset.<init>(Dataset.scala:194)
        at org.apache.spark.sql.Dataset$.ofRows(Dataset.scala:79)
        at org.apache.spark.sql.Dataset.org$apache$spark$sql$Dataset$$withPlan(Dataset.scala:3406)
        at org.apache.spark.sql.Dataset.createTempView(Dataset.scala:3082)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
        at py4j.Gateway.invoke(Gateway.java:282)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
```

```
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:238)
        at java.lang.Thread.run(Thread.java:745)


During handling of the above exception, another exception occurred:

AnalysisException                         Traceback (most recent call last)
/tmp/ipykernel_68/1826009494.py in <module>
      1 # Create a temporary view named "employees" for the DataFrame
----> 2 employees_df.createTempView("employees")

~/spark-2.4.3/python/pyspark/sql/dataframe.py in createTempView(self, name)
    159
    160         """
--> 161         self._jdf.createTempView(name)
    162
    163     @since(2.0)

~/spark-2.4.3/python/lib/py4j-0.10.7-src.zip/py4j/java_gateway.py in __call__(self, *args)
   1255         answer = self.gateway_client.send_command(command)
   1256         return_value = get_return_value(
-> 1257             answer, self.gateway_client, self.target_id, self.name)
   1258
   1259         for temp_arg in temp_args:

~/spark-2.4.3/python/pyspark/sql/utils.py in deco(*a, **kw)
     69                 raise AnalysisException(s.split(': ', 1)[1], stackTrace)
     70             if s.startswith('org.apache.spark.sql.catalyst.analysis'):
---> 71                 raise AnalysisException(s.split(': ', 1)[1], stackTrace)
     72             if s.startswith('org.apache.spark.sql.catalyst.parser.ParseException: '):
     73                 raise ParseException(s.split(': ', 1)[1], stackTrace)

AnalysisException: "Temporary view 'employees' already exists;"
```

Task 5: Execute an SQL query

Compose and execute an SQL query to fetch the records from the `employees` view where the age of employees exceeds 30. Then, display the result of the SQL query, showcasing the filtered records.

In [11]:
```
# SQL query to fetch solely the records from the View where the age exceeds 30
spark.sql("SELECT * FROM employees WHERE age > 30").show()
```

```
+------+----------+------+---+----------+
|Emp_No|  Emp_Name|Salary|Age|Department|
+------+----------+------+---+----------+
|   199|   Douglas|  2600| 34|     Sales|
|   200|  Jennifer|  4400| 36| Marketing|
|   201|   Michael| 13000| 32|        IT|
|   202|       Pat|  6000| 39|        HR|
|   203|     Susan|  6500| 36| Marketing|
|   205|   Shelley| 12008| 33|   Finance|
|   206|   William|  8300| 37|        IT|
|   100|    Steven| 24000| 39|        IT|
|   102|       Lex| 17000| 37| Marketing|
|   103| Alexander|  9000| 39| Marketing|
|   104|     Bruce|  6000| 38|        IT|
|   105|     David|  4800| 39|        IT|
|   106|     Valli|  4800| 38|     Sales|
|   107|     Diana|  4200| 35|     Sales|
|   109|    Daniel|  9000| 35|        HR|
|   110|      John|  8200| 31| Marketing|
|   111|    Ismael|  7700| 32|        IT|
|   112|Jose Manuel|  7800| 34|        HR|
|   113|      Luis|  6900| 34|     Sales|
|   116|    Shelli|  2900| 37|   Finance|
+------+----------+------+---+----------+
only showing top 20 rows
```

Task 6: Calculate Average Salary by Department

Compose an SQL query to retrieve the average salary of employees grouped by department. Display the result.

In [12]:
```
# SQL query to calculate the average salary of employees grouped by department
spark.sql("SELECT department, AVG(salary) as avg_salary FROM employees GROUP BY department").show()
```

```
[Stage 14:=========================================>              (56 + 8) / 75]
```

```
+----------+-----------------+
|department|       avg_salary|
+----------+-----------------+
|     Sales|5492.923076923077|
|        HR|           5837.5|
|   Finance|           5730.8|
| Marketing|6633.333333333333|
|        IT|           7400.0|
+----------+-----------------+
```

Task 7: Filter and Display IT Department Employees

Apply a filter on the `employees_df` DataFrame to select records where the department is `'IT'`. Display the filtered DataFrame.

In [13]:
```
# Apply a filter to select records where the department is 'IT'
employees_df.filter(employees_df['department'] == 'IT').show()
```

```
+------+--------+------+---+----------+
|Emp_No|Emp_Name|Salary|Age|Department|
+------+--------+------+---+----------+
|   198|  Donald|  2600| 29|        IT|
|   201| Michael| 13000| 32|        IT|
|   206| William|  8300| 37|        IT|
|   100|  Steven| 24000| 39|        IT|
|   104|   Bruce|  6000| 38|        IT|
|   105|   David|  4800| 39|        IT|
|   111|  Ismael|  7700| 32|        IT|
|   129|   Laura|  3300| 38|        IT|
|   132|      TJ|  2100| 34|        IT|
|   136|   Hazel|  2200| 29|        IT|
+------+--------+------+---+----------+
```

Task 8: Add 10% Bonus to Salaries

Perform a transformation to add a new column named "SalaryAfterBonus" to the DataFrame. Calculate the new salary by adding a 10% bonus to each employee's salary.

In [14]:
```
from pyspark.sql.functions import col
employees_df = employees_df.withColumn("SalaryAfterBonus", col("salary") * 1.1)
employees_df.show()

# Add a new column "SalaryAfterBonus" with 10% bonus added to the original salary
```

```
[Stage 16:>                                                         (0 + 1) / 1]
+------+---------+------+---+----------+------------------+
|Emp_No| Emp_Name|Salary|Age|Department|  SalaryAfterBonus|
+------+---------+------+---+----------+------------------+
|   198|   Donald|  2600| 29|        IT|2860.0000000000005|
|   199|  Douglas|  2600| 34|     Sales|2860.0000000000005|
|   200| Jennifer|  4400| 36| Marketing|            4840.0|
|   201|  Michael| 13000| 32|        IT|14300.000000000002|
|   202|      Pat|  6000| 39|        HR| 6600.000000000001|
|   203|    Susan|  6500| 36| Marketing| 7150.000000000001|
|   204|  Hermann| 10000| 29|   Finance|           11000.0|
|   205|  Shelley| 12008| 33|   Finance|13208.800000000001|
|   206|  William|  8300| 37|        IT|            9130.0|
|   100|   Steven| 24000| 39|        IT|26400.000000000004|
|   101|    Neena| 17000| 27|     Sales|           18700.0|
|   102|      Lex| 17000| 37| Marketing|           18700.0|
|   103|Alexander|  9000| 39| Marketing|            9900.0|
|   104|    Bruce|  6000| 38|        IT| 6600.000000000001|
|   105|    David|  4800| 39|        IT|            5280.0|
|   106|    Valli|  4800| 38|     Sales|            5280.0|
|   107|    Diana|  4200| 35|     Sales|            4620.0|
|   108|    Nancy| 12008| 28|     Sales|13208.800000000001|
|   109|   Daniel|  9000| 35|        HR|            9900.0|
|   110|     John|  8200| 31| Marketing|            9020.0|
+------+---------+------+---+----------+------------------+
only showing top 20 rows
```

Task 9: Find Maximum Salary by Age

Group the data by age and calculate the maximum salary for each age group. Display the result.

In [15]:
```
from pyspark.sql.functions import max

# Group data by age and calculate the maximum salary for each age group
employees_df.groupBy('age').agg(max('salary').alias('max_salary')).show()
```

```
[Stage 26:=====================================>                   (47 + 9) / 75]
+---+----------+
|age|max_salary|
+---+----------+
| 31|      8200|
| 34|      7800|
| 28|     12008|
| 27|     17000|
| 26|      3600|
| 37|     17000|
| 35|      9000|
| 39|     24000|
| 38|      6000|
| 29|     10000|
| 32|     13000|
| 33|     12008|
| 30|      8000|
| 36|      7900|
+---+----------+
```

Task 10: Self-Join on Employee Data

Join the "employees_df" DataFrame with itself based on the "Emp_No" column. Display the result.

In [16]:
```
# Join the DataFrame with itself based on the "Emp_No" column
employees_df.join(employees_df, 'Emp_No', how='inner').show()
```

```
+------+---------+------+---+----------+-----------------+---------+------+---+----------+-----------------+
|Emp_No| Emp_Name|Salary|Age|Department|  SalaryAfterBonus| Emp_Name|Salary|Age|Department|  SalaryAfterBonus|
+------+---------+------+---+----------+-----------------+---------+------+---+----------+-----------------+
|   198|   Donald|  2600| 29|        IT|2860.0000000000005|   Donald|  2600| 29|        IT|2860.0000000000005|
|   199|  Douglas|  2600| 34|     Sales|2860.0000000000005|  Douglas|  2600| 34|     Sales|2860.0000000000005|
|   200| Jennifer|  4400| 36| Marketing|           4840.0| Jennifer|  4400| 36| Marketing|            4840.0|
|   201|  Michael| 13000| 32|        IT|14300.000000000002|  Michael| 13000| 32|        IT|14300.000000000002|
|   202|      Pat|  6000| 39|        HR| 6600.000000000001|      Pat|  6000| 39|        HR| 6600.000000000001|
|   203|    Susan|  6500| 36| Marketing| 7150.000000000001|    Susan|  6500| 36| Marketing| 7150.000000000001|
|   204|  Hermann| 10000| 29|   Finance|          11000.0|  Hermann| 10000| 29|   Finance|           11000.0|
|   205|  Shelley| 12008| 33|   Finance|13208.800000000001|  Shelley| 12008| 33|   Finance|13208.800000000001|
|   206|  William|  8300| 37|        IT|           9130.0|  William|  8300| 37|        IT|            9130.0|
|   100|   Steven| 24000| 39|        IT|26400.000000000004|   Steven| 24000| 39|        IT|26400.000000000004|
|   101|    Neena| 17000| 27|     Sales|          18700.0|    Neena| 17000| 27|     Sales|           18700.0|
|   102|      Lex| 17000| 37| Marketing|          18700.0|      Lex| 17000| 37| Marketing|           18700.0|
|   103|Alexander|  9000| 39| Marketing|           9900.0|Alexander|  9000| 39| Marketing|            9900.0|
|   104|    Bruce|  6000| 38|        IT| 6600.000000000001|    Bruce|  6000| 38|        IT| 6600.000000000001|
|   105|    David|  4800| 39|        IT|           5280.0|    David|  4800| 39|        IT|            5280.0|
|   106|    Valli|  4800| 38|     Sales|           5280.0|    Valli|  4800| 38|     Sales|            5280.0|
|   107|    Diana|  4200| 35|     Sales|           4620.0|    Diana|  4200| 35|     Sales|            4620.0|
|   108|    Nancy| 12008| 28|     Sales|13208.800000000001|    Nancy| 12008| 28|     Sales|13208.800000000001|
```

```
|   109|   Daniel|  9000| 35|        HR|            9900.0|   Daniel|  9000| 35|        HR|            990
0.0|
|   110|     John|  8200| 31| Marketing|            9020.0|     John|  8200| 31| Marketing|            902
0.0|
+------+---------+------+---+----------+------------------+---------+------+---+----------+----------------
--+
only showing top 20 rows
```

Task 11: Calculate Average Employee Age

Calculate the average age of employees using the built-in aggregation function. Display the result.

In [18]:
```
# Calculate the average age of employees
from pyspark.sql.functions import avg
employees_df.agg(avg('age').alias('average_age')).show()
```

```
+-----------+
|average_age|
+-----------+
|      33.56|
+-----------+
```

Task 12: Calculate Total Salary by Department

Calculate the total salary for each department using the built-in aggregation function. Display the result.

In [19]:
```
# Calculate the total salary for each department. Hint - User GroupBy and Aggregate functions
from pyspark.sql.functions import sum
employees_df.groupBy('department').agg(sum('salary').alias('total_salary')).show()
```

```
+----------+------------+
|department|total_salary|
+----------+------------+
|     Sales|       71408|
|        HR|       46700|
|   Finance|       57308|
| Marketing|       59700|
|        IT|       74000|
+----------+------------+
```

Task 13: Sort Data by Age and Salary

Apply a transformation to sort the DataFrame by age in ascending order and then by salary in descending order. Display the sorted DataFrame.

In [20]:
```
# Sort the DataFrame by age in ascending order and then by salary in descending order
employees_df.orderBy(employees_df["age"].asc(), employees_df["salary"].desc()).show()
```

```
+------+---------+------+---+----------+------------------+
|Emp_No| Emp_Name|Salary|Age|Department|  SalaryAfterBonus|
+------+---------+------+---+----------+------------------+
|   137|   Renske|  3600| 26| Marketing|3960.0000000000005|
|   101|    Neena| 17000| 27|     Sales|           18700.0|
|   114|      Den| 11000| 27|   Finance|12100.000000000002|
|   108|    Nancy| 12008| 28|     Sales|13208.800000000001|
|   130|    Mozhe|  2800| 28| Marketing|3080.0000000000005|
|   126|    Irene|  2700| 28|        HR|2970.0000000000005|
|   204|  Hermann| 10000| 29|   Finance|           11000.0|
|   115|Alexander|  3100| 29|   Finance|3410.0000000000005|
|   134|  Michael|  2900| 29|     Sales|3190.0000000000005|
|   198|   Donald|  2600| 29|        IT|2860.0000000000005|
|   140|   Joshua|  2500| 29|   Finance|            2750.0|
|   136|    Hazel|  2200| 29|        IT|            2420.0|
|   120|  Matthew|  8000| 30|        HR|            8800.0|
|   110|     John|  8200| 31| Marketing|            9020.0|
|   127|    James|  2400| 31|        HR|            2640.0|
|   201|  Michael| 13000| 32|        IT|14300.000000000002|
|   111|   Ismael|  7700| 32|        IT|            8470.0|
|   119|    Karen|  2500| 32|   Finance|            2750.0|
|   205|  Shelley| 12008| 33|   Finance|13208.800000000001|
|   124|    Kevin|  5800| 33| Marketing| 6380.000000000001|
+------+---------+------+---+----------+------------------+
only showing top 20 rows
```

Task 14: Count Employees in Each Department

Calculate the number of employees in each department. Display the result.

In [25]:
```python
from pyspark.sql.functions import count

# Calculate the number of employees in each department
employees_df.groupBy('department').agg(count('Emp_Name').alias('numbers_employees')).show()
```

```
+----------+-----------------+
|department|numbers_employees|
+----------+-----------------+
|     Sales|               13|
|        HR|                8|
|   Finance|               10|
| Marketing|                9|
|        IT|               10|
+----------+-----------------+
```

Task 15: Filter Employees with the letter o in the Name

Apply a filter to select records where the employee's name contains the letter `'o'`. Display the filtered DataFrame.

In [26]:
```
# Apply a filter to select records where the employee's name contains the letter 'o'
employees_df.filter('Emp_Name like "%o%"').show()
```

```
+------+-----------+------+---+----------+------------------+
|Emp_No|   Emp_Name|Salary|Age|Department|  SalaryAfterBonus|
+------+-----------+------+---+----------+------------------+
|   198|     Donald|  2600| 29|        IT|2860.0000000000005|
|   199|    Douglas|  2600| 34|     Sales|2860.0000000000005|
|   110|       John|  8200| 31| Marketing|            9020.0|
|   112|Jose Manuel|  7800| 34|        HR|            8580.0|
|   130|      Mozhe|  2800| 28| Marketing|3080.0000000000005|
|   133|      Jason|  3300| 38|     Sales|3630.0000000000005|
|   139|       John|  2700| 36|     Sales|2970.0000000000005|
|   140|     Joshua|  2500| 29|   Finance|            2750.0|
+------+-----------+------+---+----------+------------------+
```

Congratulations! You have completed the project.

Now you know how to create a DataFrame from a CSV data file and perform a variety of DataFrame transformations and actions using Spark SQL.

Authors

Raghul Ramesh

Lavanya T S

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-09-01	0.1	Lavanya T S	Initial version
2023-09-11	0.2	Pornima More	QA pass with edits