UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
DIRECCIÓN DE POSGRADO

DIPLOMADO ESTADÍSTICA APLICADA A LA TOMA DE DECISIONES

SEGUNDA VERSIÓN

Laboratorio Dataframes

NOMBRE : HEBERT JUAN DE DIOS DELGADILLO FERNANDEZ

CARLOS ALFREDO ORIHUELA BERRIOS

DOCENTE : DANNY LUIS HUANCA SEVILLA

Cochabamba – Bolivia

2023

Instalando pyspark en COLAB, usando el comando **`!pip install pyspark`**

```
[2] !pip install pyspark
    #import findspark
    #findspark.init()
```

```
Collecting pyspark
  Downloading pyspark-3.4.1.tar.gz (310.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 310.8/310.8 MB 2.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.4.1-py2.py3-none-any.whl size=311285387 sha256=c07154b711aa372423b25
  Stored in directory: /root/.cache/pip/wheels/0d/77/a3/ff2f74cc9ab41f8f594dabf0579c2a7c6de920d584206e0834
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.4.1
```

Importando la librería Spark, para poder interactuar con un clúster de Spark y realizar operaciones de procesamiento de datos distribuidas.

```
[3] from pyspark.sql import SparkSession
```

```
[4] spark = SparkSession.builder.appName("SesionDataframes").getOrCreate()
```

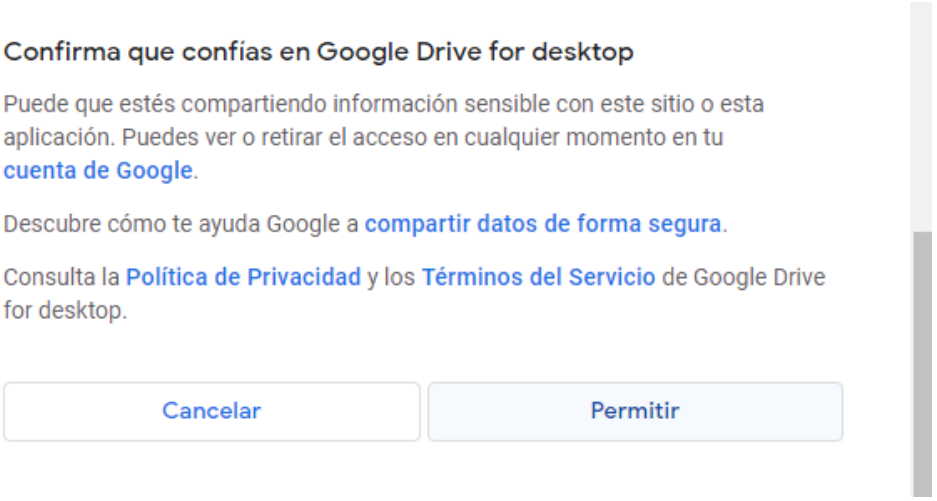
Montando drive en COLAB, para utilizar datos almacenados. usando los siguientes comandos..

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

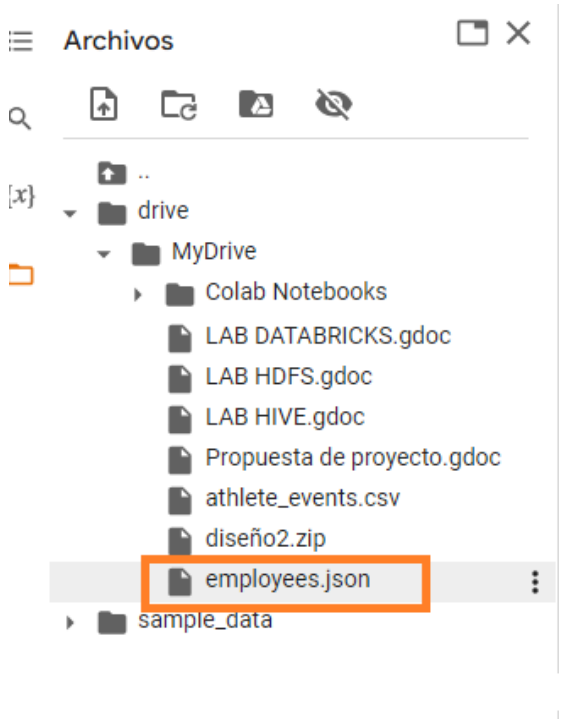
```
[5] from google.colab import drive
    drive.mount('/content/drive')
```

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

Seleccionamos la cuenta de google Drive y presionamos **permitir** para finalizar con exito montar la unidad de drive en nuestro notebook.



Verificamos que esté el archivo con los datos de `employees.json` en la unidad virtual de COLAB



cargamos los datos de **employees.json,** desde la unidad montada en COLAB, luego con el comando **`employee_df.show()`** mostramos los datos de la tabla.

```
[6]  employee_df = spark.read.json("/content/drive/MyDrive/employees.json")
```

```
[7]  employee_df.show()
```

```
+------+-----------+-----+------+-------+----+
|deptno|designation|empno| ename|manager| sal|
+------+-----------+-----+------+-------+----+
|    20|      CLERK| 7369| SMITH|   7902| 800|
|    30|   SALESMAN| 7499| ALLEN|   7698|1600|
|    30|   SALESMAN| 7521|  WARD|   7698|1250|
|    20|    MANAGER| 7566|TURNER|   7839|2975|
|    30|   SALESMAN| 7654|MARTIN|   7698|1250|
|    30|    MANAGER| 7698|MILLER|   7839|2850|
|    10|    MANAGER| 7782| CLARK|   7839|2450|
|    20|    ANALYST| 7788| SCOTT|   7566|3000|
|    10|  PRESIDENT| 7839|  KING|   NULL|5000|
+------+-----------+-----+------+-------+----+
```

Mostramos el esquema del DataFrame, la estructura de las columnas como sus nombres y tipos de datos.

```
[8]  employee_df.printSchema()
```

```
root
 |-- deptno: long (nullable = true)
 |-- designation: string (nullable = true)
 |-- empno: long (nullable = true)
 |-- ename: string (nullable = true)
 |-- manager: string (nullable = true)
 |-- sal: long (nullable = true)
```

Realizamos algunas operación **SQL** con el dataframe tales como:

La lista de todos los cargos de los empleados en la empresa

Operaciones con Dataframes

```
[9]  employee_df.select("designation").show()
```

```
+-----------+
|designation|
+-----------+
|      CLERK|
|   SALESMAN|
|   SALESMAN|
|    MANAGER|
|   SALESMAN|
|    MANAGER|
|    MANAGER|
|    ANALYST|
|  PRESIDENT|
+-----------+
```

Mostrar los datos de la columna salario

```
employee_df.select("sal").show()
```

```
+----+
| sal|
+----+
| 800|
|1600|
|1250|
|2975|
|1250|
|2850|
|2450|
|3000|
|5000|
+----+
```

Mostrar todos los datos del dataframe empleado

```
[11] employee_df.select("*").show()
```

```
+------+-----------+-----+------+-------+----+
|deptno|designation|empno| ename|manager| sal|
+------+-----------+-----+------+-------+----+
|    20|      CLERK| 7369| SMITH|   7902| 800|
|    30|   SALESMAN| 7499| ALLEN|   7698|1600|
|    30|   SALESMAN| 7521|  WARD|   7698|1250|
|    20|    MANAGER| 7566|TURNER|   7839|2975|
|    30|   SALESMAN| 7654|MARTIN|   7698|1250|
|    30|    MANAGER| 7698|MILLER|   7839|2850|
|    10|    MANAGER| 7782| CLARK|   7839|2450|
|    20|    ANALYST| 7788| SCOTT|   7566|3000|
|    10|  PRESIDENT| 7839|  KING|   NULL|5000|
+------+-----------+-----+------+-------+----+
```

Mostrar datos de las columnas departamento, cargo y nombre

```
[12] employee_df.select("deptno", "designation", "ename").show()
```

```
+------+-----------+------+
|deptno|designation| ename|
+------+-----------+------+
|    20|      CLERK| SMITH|
|    30|   SALESMAN| ALLEN|
|    30|   SALESMAN|  WARD|
|    20|    MANAGER|TURNER|
|    30|   SALESMAN|MARTIN|
|    30|    MANAGER|MILLER|
|    10|    MANAGER| CLARK|
|    20|    ANALYST| SCOTT|
|    10|  PRESIDENT|  KING|
+------+-----------+------+
```

Incrementar en 10 a todos los datos de la columna salario

```
[13] employee_df.select(employee_df['sal']+10).show()
```

```
+----------+
|(sal + 10)|
+----------+
|       810|
|      1610|
|      1260|
|      2985|
|      1260|
|      2860|
|      2460|
|      3010|
|      5010|
+----------+
```

Mostrar todos los empleados que tengan un salario mayor a 2000

```
[14] employee_df.filter(employee_df['sal'] > 2000).show()
```

```
+------+-----------+-----+------+-------+----+
|deptno|designation|empno| ename|manager| sal|
+------+-----------+-----+------+-------+----+
|    20|    MANAGER| 7566|TURNER|   7839|2975|
|    30|    MANAGER| 7698|MILLER|   7839|2850|
|    10|    MANAGER| 7782| CLARK|   7839|2450|
|    20|    ANALYST| 7788| SCOTT|   7566|3000|
|    10|  PRESIDENT| 7839|  KING|   NULL|5000|
+------+-----------+-----+------+-------+----+
```

agrupar a los empleados según su cargo en la empresa y mostrar la cantidad

```
[15] employee_df.groupBy(employee_df['designation']).count().show()
```

```
+-----------+-----+
|designation|count|
+-----------+-----+
|    ANALYST|    1|
|   SALESMAN|    3|
|      CLERK|    1|
|    MANAGER|    3|
|  PRESIDENT|    1|
+-----------+-----+
```

Creamos una vista del Dataframe con el comando `employee_df.createOrReplaceTempView("empleados"),`

Luego realizamos una consulta de la vista para almacenar en otro objeto llamado **sqlDF** para luego mostrarlos

```
[16] employee_df.createOrReplaceTempView("empleados")

[17] sqlDF = spark.sql("select * from empleados")
     sqlDF.show()
```

```
+------+-----------+-----+------+-------+----+
|deptno|designation|empno| ename|manager| sal|
+------+-----------+-----+------+-------+----+
|    20|      CLERK| 7369| SMITH|   7902| 800|
|    30|   SALESMAN| 7499| ALLEN|   7698|1600|
|    30|   SALESMAN| 7521|  WARD|   7698|1250|
|    20|    MANAGER| 7566|TURNER|   7839|2975|
|    30|   SALESMAN| 7654|MARTIN|   7698|1250|
|    30|    MANAGER| 7698|MILLER|   7839|2850|
|    10|    MANAGER| 7782| CLARK|   7839|2450|
|    20|    ANALYST| 7788| SCOTT|   7566|3000|
|    10|  PRESIDENT| 7839|  KING|   NULL|5000|
+------+-----------+-----+------+-------+----+
```

También creamos una vista Global llamada **empleadosGlobal2** con el comando

`employee_df.createGlobalTempView("empleadosGlobal2")`

esto para que otros usuario puedan usar los datos,

```
[20] employee_df.createGlobalTempView("empleadosGlobal2")

[21] spark.sql("select * from global_temp.empleadosGlobal2").show()
```

```
+------+-----------+-----+------+-------+----+
|deptno|designation|empno| ename|manager| sal|
+------+-----------+-----+------+-------+----+
|    20|      CLERK| 7369| SMITH|   7902| 800|
|    30|   SALESMAN| 7499| ALLEN|   7698|1600|
|    30|   SALESMAN| 7521|  WARD|   7698|1250|
|    20|    MANAGER| 7566|TURNER|   7839|2975|
|    30|   SALESMAN| 7654|MARTIN|   7698|1250|
|    30|    MANAGER| 7698|MILLER|   7839|2850|
|    10|    MANAGER| 7782| CLARK|   7839|2450|
|    20|    ANALYST| 7788| SCOTT|   7566|3000|
|    10|  PRESIDENT| 7839|  KING|   NULL|5000|
+------+-----------+-----+------+-------+----+
```

Con spark tambien se puede realizar subconsultas sql, como en el ejemplo donde mostramos la cantidad de empleados que su nombre sea igual a SMITH.

```
[22] spark.sql("select count(*) from empleados where ename = 'SMITH'").show()
```

```
+--------+
|count(1)|
+--------+
|       1|
+--------+
```

Creamos una segunda Vista del dataframe llamado **empleados2,**

Luego realizamos un JOIN de las 2 vistas creadas anteriormente empleados y empleados2.

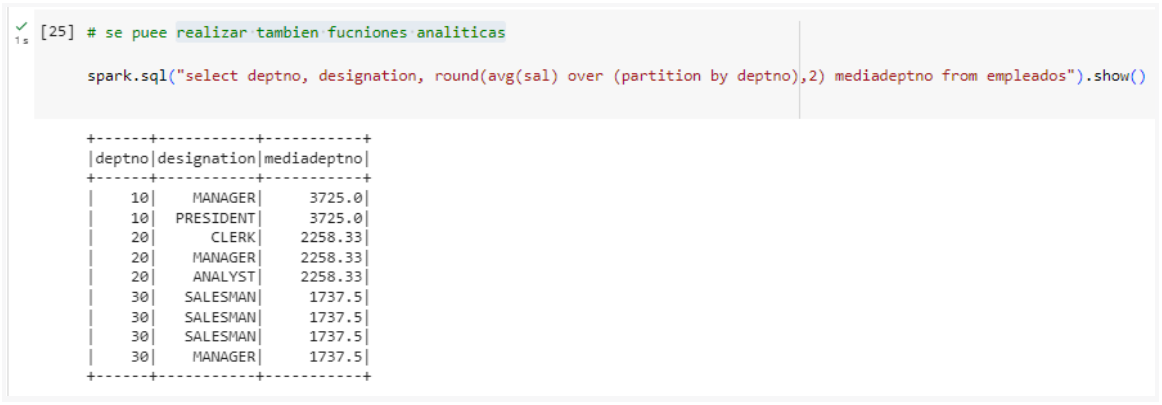
```
[23] employee_df.createOrReplaceTempView("empleados2")
```

```
[24] spark.sql("select * from empleados a \
                left join empleados2 b on a.deptno = b.deptno").show()
```

```
+------+-----------+-----+------+-------+----+------+-----------+-----+------+-------+----+
|deptno|designation|empno| ename|manager| sal|deptno|designation|empno| ename|manager| sal|
+------+-----------+-----+------+-------+----+------+-----------+-----+------+-------+----+
|    20|      CLERK| 7369| SMITH|   7902| 800|    20|    ANALYST| 7788| SCOTT|   7566|3000|
|    20|      CLERK| 7369| SMITH|   7902| 800|    20|    MANAGER| 7566|TURNER|   7839|2975|
|    20|      CLERK| 7369| SMITH|   7902| 800|    20|      CLERK| 7369| SMITH|   7902| 800|
|    30|   SALESMAN| 7499| ALLEN|   7698|1600|    30|    MANAGER| 7698|MILLER|   7839|2850|
|    30|   SALESMAN| 7499| ALLEN|   7698|1600|    30|   SALESMAN| 7654|MARTIN|   7698|1250|
|    30|   SALESMAN| 7499| ALLEN|   7698|1600|    30|   SALESMAN| 7521|  WARD|   7698|1250|
|    30|   SALESMAN| 7499| ALLEN|   7698|1600|    30|   SALESMAN| 7499| ALLEN|   7698|1600|
|    30|   SALESMAN| 7521|  WARD|   7698|1250|    30|    MANAGER| 7698|MILLER|   7839|2850|
|    30|   SALESMAN| 7521|  WARD|   7698|1250|    30|   SALESMAN| 7654|MARTIN|   7698|1250|
|    30|   SALESMAN| 7521|  WARD|   7698|1250|    30|   SALESMAN| 7521|  WARD|   7698|1250|
|    30|   SALESMAN| 7521|  WARD|   7698|1250|    30|   SALESMAN| 7499| ALLEN|   7698|1600|
|    20|    MANAGER| 7566|TURNER|   7839|2975|    20|    ANALYST| 7788| SCOTT|   7566|3000|
|    20|    MANAGER| 7566|TURNER|   7839|2975|    20|    MANAGER| 7566|TURNER|   7839|2975|
|    20|    MANAGER| 7566|TURNER|   7839|2975|    20|      CLERK| 7369| SMITH|   7902| 800|
|    30|   SALESMAN| 7654|MARTIN|   7698|1250|    30|    MANAGER| 7698|MILLER|   7839|2850|
|    30|   SALESMAN| 7654|MARTIN|   7698|1250|    30|   SALESMAN| 7654|MARTIN|   7698|1250|
|    30|   SALESMAN| 7654|MARTIN|   7698|1250|    30|   SALESMAN| 7521|  WARD|   7698|1250|
|    30|   SALESMAN| 7654|MARTIN|   7698|1250|    30|   SALESMAN| 7499| ALLEN|   7698|1600|
|    30|    MANAGER| 7698|MILLER|   7839|2850|    30|    MANAGER| 7698|MILLER|   7839|2850|
|    30|    MANAGER| 7698|MILLER|   7839|2850|    30|   SALESMAN| 7654|MARTIN|   7698|1250|
+------+-----------+-----+------+-------+----+------+-----------+-----+------+-------+----+
only showing top 20 rows
```

Se realiza también funciones analíticas con spark, como por ejemplo el de realizar un promedio de los salario de empleado que estén en el mismo departamento.con el comando

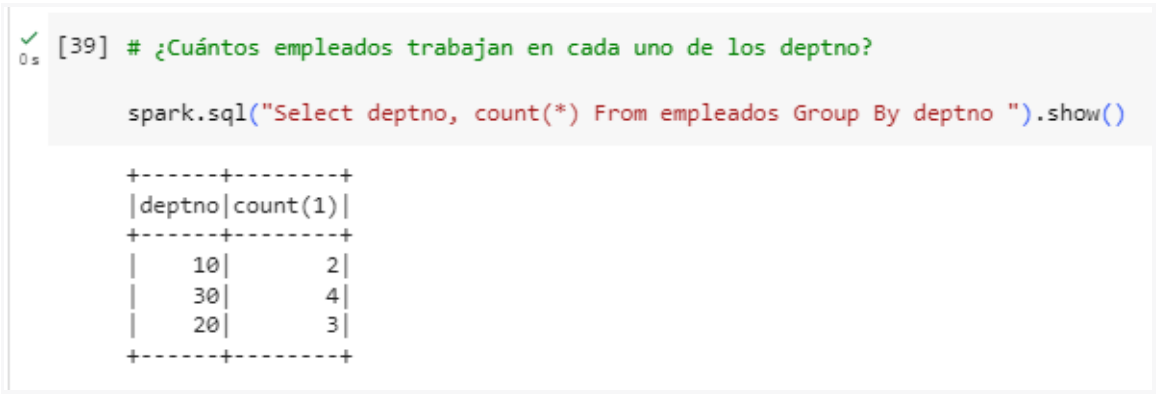
```
spark.sql("select deptno, designation, round(avg(sal) over (partition by deptno),2) mediadeptno from empleados").show()
```



Ejecute las siguientes sentencias

Realizando las preguntas vemos que hay 3 departamentos, en los cuales hay:

2 empleados en el departamento 10
4 empleados en el departamento 30
3 empleados en el departamento 20



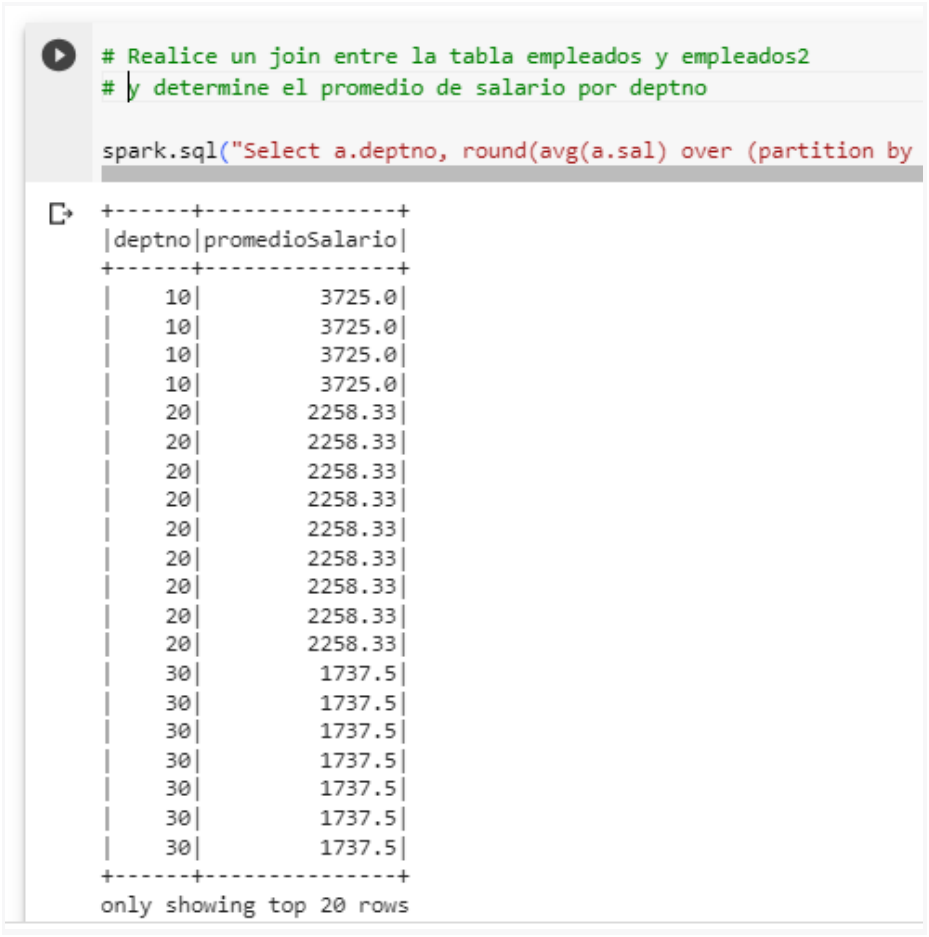
El promedio del salario de los empleados es: 2352.78

```
[42] # Determine cual es el promedio de salario de todos los empleados
     spark.sql("Select ROUND(AVG(sal), 2) AS promedio_salario From empleados").show()
```

```
+----------------+
|promedio_salario|
+----------------+
|         2352.78|
+----------------+
```

Realizamos un JOIN con las tablas empleados y empleados 2, y sacamos el promedio de los salarios según el departamento donde trabajan.

```
spark.sql("Select a.deptno, round(avg(a.sal) over (partition by a.deptno),2) as promedioSalario From empleados a join empleados2 b on a.deptno = b.deptno").show()
```



Creamos una funcion analitica para ordenar por salario de menor a mayor para cada departamento

spark.sql("Select deptno , sal, RANK() over (partition by deptno ORDER BY sal ASC) as orden From empleados").show()

```
[55] # Cree una funcion analitica para ordenar por salario de menor a mayor para cada departamento
     spark.sql("Select deptno , sal, RANK() over (partition by deptno ORDER BY sal ASC) as orden From empleados").show()
```

```
+------+----+-----+
|deptno| sal|orden|
+------+----+-----+
|    10|2450|    1|
|    10|5000|    2|
|    20| 800|    1|
|    20|2975|    2|
|    20|3000|    3|
|    30|1250|    1|
|    30|1250|    1|
|    30|1600|    3|
|    30|2850|    4|
+------+----+-----+
```

Podremos convertir también un DataFrame de PySpark en un DataFrame de Pandas, lo que nos permite trabajar con los datos usando pandas para su análisis de datos en Python correspondiente.

```
[56] pandas_df = employee_df.toPandas()
```

```
pandas_df.head()
```

	deptno	designation	empno	ename	manager	sal
0	20	CLERK	7369	SMITH	7902	800
1	30	SALESMAN	7499	ALLEN	7698	1600
2	30	SALESMAN	7521	WARD	7698	1250
3	20	MANAGER	7566	TURNER	7839	2975
4	30	SALESMAN	7654	MARTIN	7698	1250

Esto es muy útil cuando deseamos realizar análisis de datos o visualizaciones que son más fáciles de realizar con Pandas en lugar de PySpark.