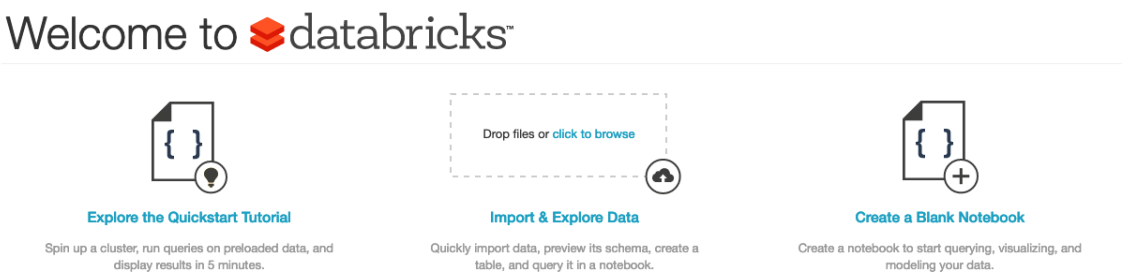
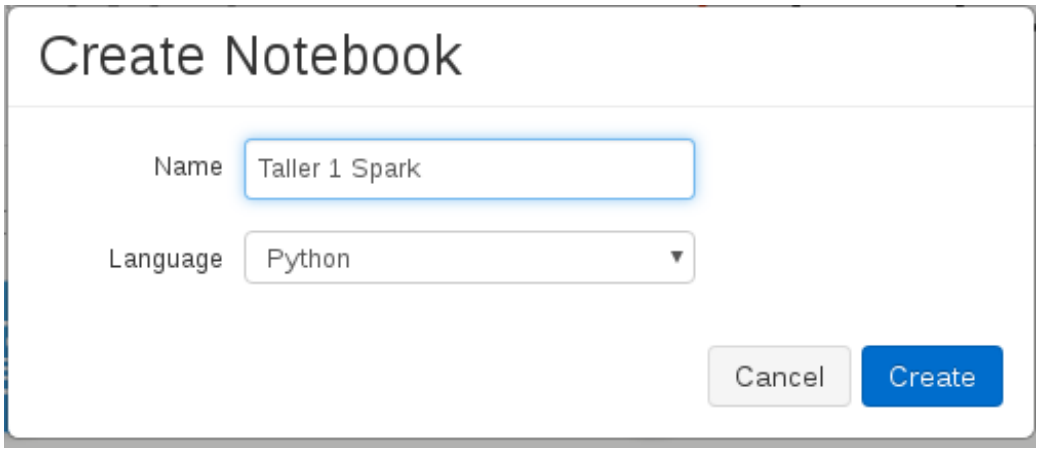
Taller spark

Interfaz de databricks

1. Ingrese a databricks version “community edition“.



2. Cree un nuevo Notebook. Establezca como nombre “Taller 1 Spark” y lenguaje Python.



3. Identifique los componentes de una celda de notebook

    1. Celda para la escritura de código en Python.

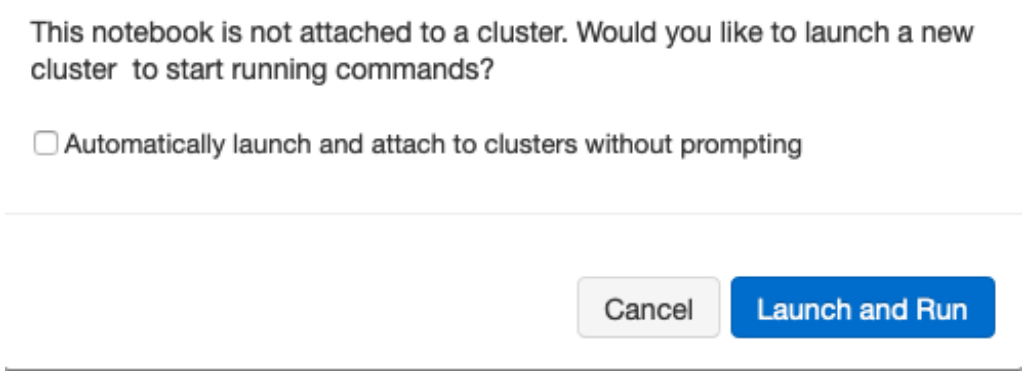
2. Comando para ejecutar el código de la celda actual.
3. Crear una nueva celda.
4. Ejecutar todas las celdas. La ejecución es ordenada, empezando por la primera celda.
5. Reinicia el resultado de ejecución de las celdas.

SparkSQL: Manipulación de datos con `DataFrame`

1. Leer CSV de ejemplo como `DataFrame`

```
diamonds =
sqlContext.read.format('com.databricks.spark.csv').options(
header='true', inferSchema='true').load('/databricks-
datasets/Rdatasets/data-001/csv/ggplot2/diamonds.csv')
```

Si el notebook no está asociado a un cluster, se mostrará la opción de iniciar uno y vincularlo:



2. Agregue una nueva celda y visualice el contenido del csv:

```
display(diamonds)
```

_c0	carat	cut
1	0.23	Idea
2	0.21	Prer
3	0.23	Goo

3. Mostrar el esquema del `DataFrame`:

```
diamonds.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- carat: double (nullable = true)
 |-- cut: string (nullable = true)
 |-- color: string (nullable = true)
 |-- clarity: string (nullable = true)
 |-- depth: double (nullable = true)
 |-- table: double (nullable = true)
 |-- price: integer (nullable = true)
 |-- x: double (nullable = true)
 |-- y: double (nullable = true)
 |-- z: double (nullable = true)
```

4. Contar el número de filas en el conjunto de datos:

```
print(diamonds.count())
```

5. Ver los distintos colores de diamantes en el conjunto de datos:

```
display(diamonds.select('color').distinct().collect())
```

6. Crear un nuevo `DataFrame` con la columna `price` de tipo `Double`:

```
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import *
diamondsCast = diamonds.withColumn("price",
diamonds["price"].cast(DoubleType()))
```

7. Calcular el precio promedio por valor `carat`:

```
carat_avgPrice = (diamondsCast
            .groupBy("carat")
            .avg("price")
            .withColumnRenamed("avg(price)", "avgPrice")
            .orderBy(desc("avgPrice")))
```

8. Ver el top-10 de los precios promedio más altos para el valor carat:

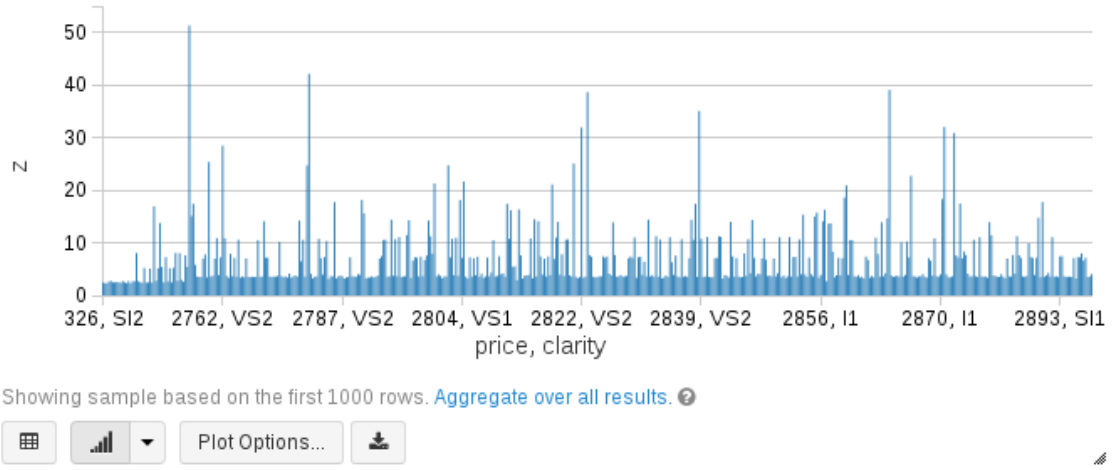
```
carat_avgPrice.show(10)
```

▸ (1) Spark Jobs

```
+-----+------------------+
|carat|          avgPrice|
+-----+------------------+
| 3.51|           18701.0|
| 2.67|           18686.0|
|  4.5|           18531.0|
| 5.01|           18018.0|
| 2.57|17841.666666666668|
|  2.6|           17535.0|
| 2.64|           17407.0|
| 4.13|           17329.0|
| 2.39|17182.428571428572|
| 2.71|           17146.0|
+-----+------------------+
only showing top 10 rows
```

9. Para realizar visualizaciones rápidas sobre los datos, puede utilizar las opciones proporcionadas por el notebook. Utilice el botón que está en la parte inferior en forma de gráfico de barras:

```
display(diamonds)
```



Manipulación de datos con `RDD`

1. Se puede obtener un `RDD` directamente del `DataFrame`.

```
diamonds_rdd = diamonds.rdd
```

2. Ver los primeros tres elementos del RDD:

```
diamonds_rdd.take(3)
```

3. Contar diamantes por `cut`:

```
countByGroup = diamonds_rdd.map(lambda x: (x.cut,
1)).reduceByKey(lambda x,y: x+y)
display(countByGroup.collect())
```

4. Distintos tipos de `clarity` para los diamantes en el conjunto de datos:

```
distinctClarity = diamonds_rdd.map(lambda x: x.clarity).distinct()
distinctClarity.collect()
```

5. Precio promedio de los diamantes por `cut`:

```
avgPrice = diamonds_rdd.map(lambda x: (x.cut,
float(x.price))).reduceByKey(lambda x,y: (x+y)/2)
display(avgPrice.collect())
```

Exportar los resultados a CSV

1. Eliminar el archivo si ya existe:

```
%fs rm -r /FileStore/carat_avgPrice
```

2. Guardar la consulta al archivo CSV en la ruta FileStore. Cada partición

será guardada en un archivo individual. Colocando `repartition(1)` se creará un único archivo de salida:

```
(carat_avgPrice.repartition(1)
        .write
        .format('com.databricks.spark.csv')
        .options(header='true'
).save('/FileStore/carat_avgPrice'))
```

3. Eliminar el archivo si ya existe

```
%fs rm -r /FileStore/cutPrice
```

4. Seleccionar las columnas `cut` y `price`

```
(diamonds.repartition(1)
  .select('cut', 'price')
  .write
  .format('com.databricks.spark.csv')
  .options(header='true'
).save('/FileStore/cutPrice'))
```

5. Podrá acceder a los archivos en las rutas:
   - `/FileStore/carat_avgPrice`
   - `/FileStore/cutPrice`

6. Leer y mostrar el CSV guardado anteriormente. Los puede descargar dando click en  :

```
cutPrice =
sqlContext.read.format('com.databricks.spark.csv').options(header='t
rue', inferSchema='true').load('/FileStore/cutPrice')
display(cutPrice)
```

7. Puede también usar sintaxis SQL para consultar las tablas creadas:

```
diamondsCast.registerTempTable("diamonds")
color_agg = sqlContext.sql("""
                SELECT color, avg(price) AS price
                FROM diamonds
                GROUP BY color
                ORDER BY color
                """)
display(color_agg)
```

8. O consultando directamente la tabla creada:

```
%sql
SELECT color, avg(price) AS price FROM diamonds GROUP BY color
ORDER BY color
```

Práctica: WordCount con RDDs

Para este ejercicio copie cada línea de código por separado:

```
def removePunctuation(text):
  import re
  return re.sub(r'[^A-Za-z0-9 ]', '', text).lower().strip()

fileName = '/databricks-datasets/cs100/lab1/data-001/shakespeare.txt'

# 1. Cargue el archivo localizado en fileName como un RDD llamado shakespeareRDD.
shakespeareRDD = sc.textFile(fileName)

# 2. Aquí se aplica la función removePunctuation al contenido de shakespeareRDD.
shakespeareRDD = shakespeareRDD.map(removePunctuation)
shakespeareRDD.collect()

# 3. shakespeareWordsRDD contiene las palabras separadas por espacios.
shakespeareWordsRDD = shakespeareRDD.flatMap(lambda line: line.split(' '))
shakespeareWordsRDD.take(10)

# 4. Eliminamos las palabras de longitud 0 de shakespeareWordsRDD.
wordsRDD = shakespeareWordsRDD.filter(lambda line: len(line) > 0)


# 5. Imprimimos el conteo de palabras en wordsRDD.
wordsRDD.count()


# 6. Calculamos la frecuencia de cada palabra en wordsRDD y guardamos el resultado en freqRDD.
freqRDD = wordsRDD.map(lambda word: (word, 1)).reduceByKey(lambda a, b : a + b)


# 7. Imprimimos el contenido de freqRDD.
display(freqRDD.collect())
```