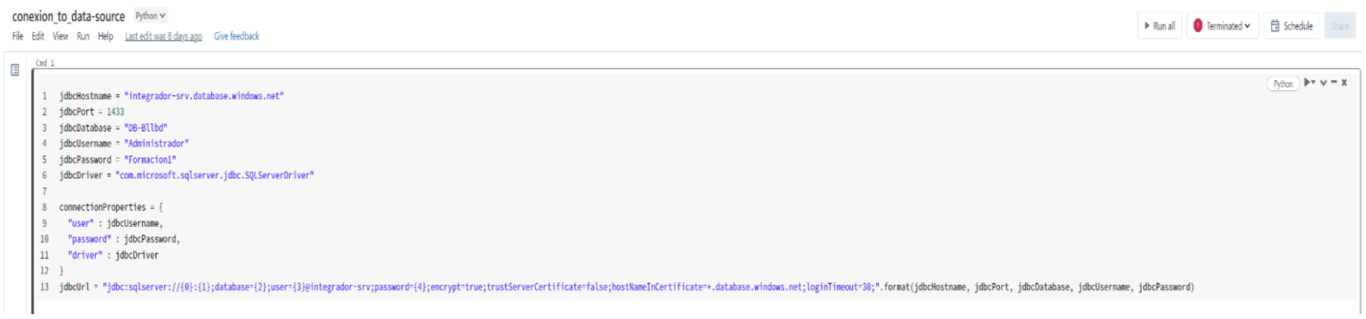
Documentación Billboard.

Rios Rocio Ayelen.

Origen_to_raw

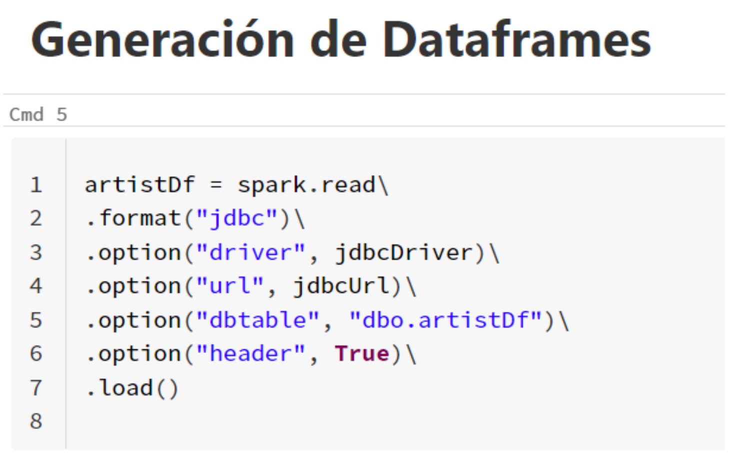
Como primer paso extraemos los archivos de la base de datos y los guardamos en el Datalake. Este paso lo realicé con una notebook de databricks en donde la conexión se hizo con

- **conexion_to_data-source:**



```python
jdbcHostname = "integrador-srv.database.windows.net"
jdbcPort = 1433
jdbcDatabase = "DB-Bllbd"
jdbcUsername = "Administrador"
jdbcPassword = "Formacion1"
jdbcDriver = "com.microsoft.sqlserver.jdbc.SQLServerDriver"

connectionProperties = {
  "user" : jdbcUsername,
  "password" : jdbcPassword,
  "driver" : jdbcDriver
}
jdbcUrl = "jdbc:sqlserver://{0}:{1};database={2};user={3}@integrador-srv;password={4};encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;loginTimeout=30;".format(jdbcHostname, jdbcPort, jdbcDatabase, jdbcUsername, jdbcPassword)
```

Luego de esto se generan los dataframes

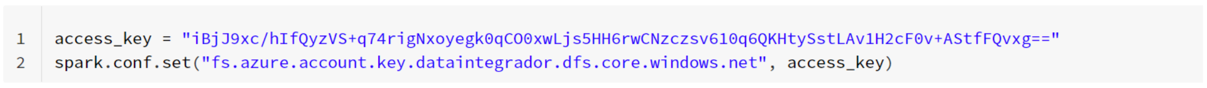


Generación de Dataframes

```python
artistDf = spark.read\
.format("jdbc")\
.option("driver", jdbcDriver)\
.option("url", jdbcUrl)\
.option("dbtable", "dbo.artistDf")\
.option("header", True)\
.load()

```

Se realizó esta acción por cada tabla
[artistDf, billboardHot100, grammyAlbums, grammySongs, riaaAlbumCerts, riaaSingleCerts, songAttribues, spotifyWeeklyTop200Streams]

Luego se realizó la autenticación en el Datalake.



```python
access_key = "iBjJ9xc/hIfQyzVS+q74rigNxoyegk0qCO0xwLjs5HH6rwCNzczsv610q6QKHtySstLAv1H2cF0v+AStfFQvxg=="
spark.conf.set("fs.azure.account.key.dataintegrador.dfs.core.windows.net", access_key)
```

Se generó la conexión con routes.

```
%run "./routes"
```

routes contiene en su notebook:

```
raw = "abfss://integrador-rocio-rios@dataintegrador.dfs.core.windows.net/raw"
trusted = "abfss://integrador-rocio-rios@dataintegrador.dfs.core.windows.net/trusted"
```

Luego de esto se hizo el **guardado de tablas en formato csv**

```
artistDf.write.save(f"{raw}/artistDf", format="csv", mode="overwrite",header='True')
```

Se realizó esta acción con cada dataframe.

Finalmente con la correcta ejecución de todos los pasos se realiza la **salida de Notebook**

```
dbutils.notebook.exit("Todo ok")
```

```
Notebook exited: Todo ok
```

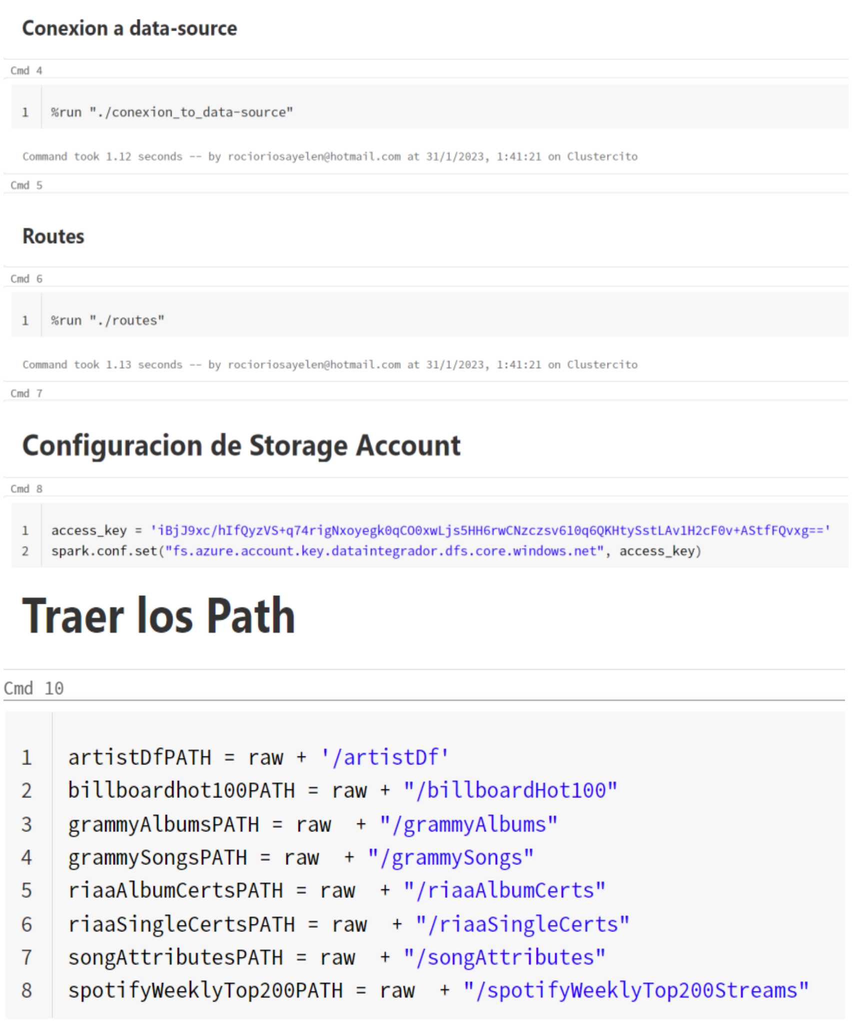
Raw_to_trusted

En esta etapa comenzamos a limpiar y transformar los datos.

Importamos librerías a utilizar:

```python
import pyspark.sql.functions as F
from pyspark.sql import *
from pyspark.sql.types import *
from pyspark import *
from pyspark.sql.functions import initcap
from pyspark.sql.functions import col
from pyspark.sql.functions import from_unixtime
from pyspark.sql.functions import log10, when, abs
from pyspark.sql.functions import to_date
from pyspark.sql.functions import concat, lit
from pyspark.sql.functions import coalesce
```

Generamos las conexiones necesarias:



Conexion a data-source

Cmd 4

```
%run "./conexion_to_data-source"
```

```
Command took 1.12 seconds -- by rocioriosayelen@hotmail.com at 31/1/2023, 1:41:21 on Clustercito
```

Cmd 5

Routes

Cmd 6

```
%run "./routes"
```

```
Command took 1.13 seconds -- by rocioriosayelen@hotmail.com at 31/1/2023, 1:41:21 on Clustercito
```

Cmd 7

Configuracion de Storage Account

Cmd 8

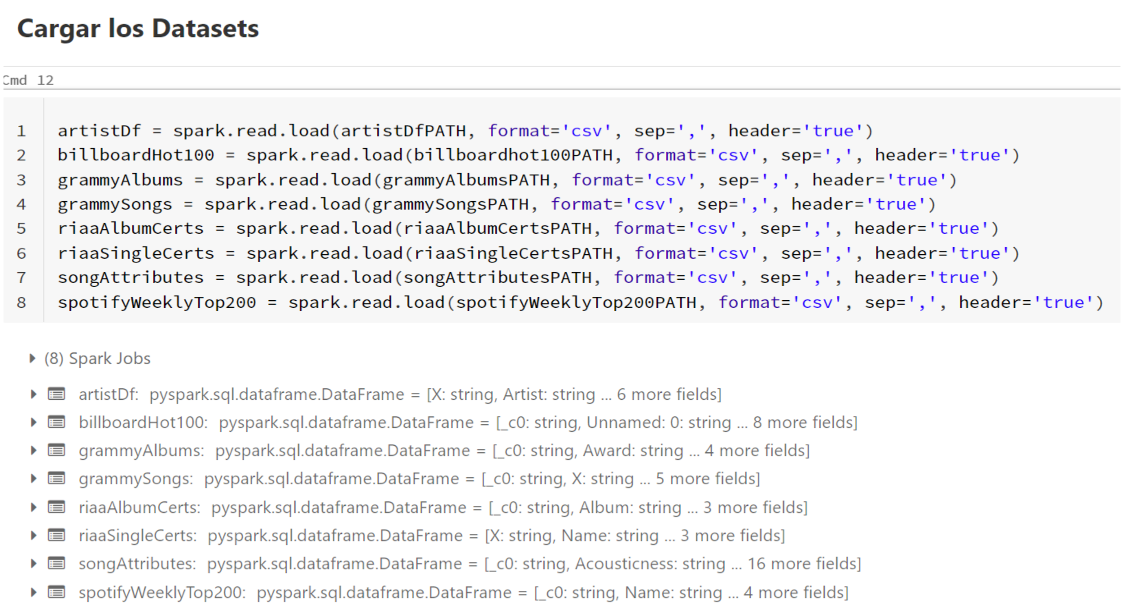
```python
access_key = 'iBjJ9xc/hIfQyzVS+q74rigNxoyegk0qCO0xwLjs5HH6rwCNzczsv610q6QKHtySstLAv1H2cF0v+AStfFQvxg=='
spark.conf.set("fs.azure.account.key.dataintegrador.dfs.core.windows.net", access_key)
```

Traer los Path

Cmd 10

```python
artistDfPATH = raw + '/artistDf'
billboardhot100PATH = raw + "/billboardHot100"
grammyAlbumsPATH = raw  + "/grammyAlbums"
grammySongsPATH = raw  + "/grammySongs"
riaaAlbumCertsPATH = raw  + "/riaaAlbumCerts"
riaaSingleCertsPATH = raw  + "/riaaSingleCerts"
songAttributesPATH = raw  + "/songAttributes"
spotifyWeeklyTop200PATH = raw  + "/spotifyWeeklyTop200Streams"
```

Se cargan los datasets guardados en csv:



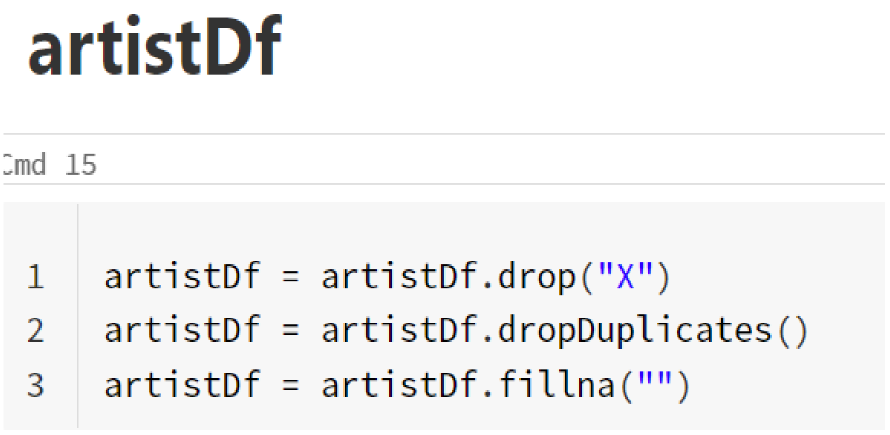
Cargar los Datasets

Cmd 12

```python
artistDf = spark.read.load(artistDfPATH, format='csv', sep=',', header='true')
billboardHot100 = spark.read.load(billboardhot100PATH, format='csv', sep=',', header='true')
grammyAlbums = spark.read.load(grammyAlbumsPATH, format='csv', sep=',', header='true')
grammySongs = spark.read.load(grammySongsPATH, format='csv', sep=',', header='true')
riaaAlbumCerts = spark.read.load(riaaAlbumCertsPATH, format='csv', sep=',', header='true')
riaaSingleCerts = spark.read.load(riaaSingleCertsPATH, format='csv', sep=',', header='true')
songAttributes = spark.read.load(songAttributesPATH, format='csv', sep=',', header='true')
spotifyWeeklyTop200 = spark.read.load(spotifyWeeklyTop200PATH, format='csv', sep=',', header='true')
```

```
(8) Spark Jobs
artistDf:  pyspark.sql.dataframe.DataFrame = [X: string, Artist: string ... 6 more fields]
billboardHot100:  pyspark.sql.dataframe.DataFrame = [_c0: string, Unnamed: 0: string ... 8 more fields]
grammyAlbums:  pyspark.sql.dataframe.DataFrame = [_c0: string, Award: string ... 4 more fields]
grammySongs:  pyspark.sql.dataframe.DataFrame = [_c0: string, X: string ... 5 more fields]
riaaAlbumCerts:  pyspark.sql.dataframe.DataFrame = [_c0: string, Album: string ... 3 more fields]
riaaSingleCerts:  pyspark.sql.dataframe.DataFrame = [X: string, Name: string ... 3 more fields]
songAttributes:  pyspark.sql.dataframe.DataFrame = [_c0: string, Acousticness: string ... 16 more fields]
spotifyWeeklyTop200:  pyspark.sql.dataframe.DataFrame = [_c0: string, Name: string ... 4 more fields]
```

Transformación y limpieza:

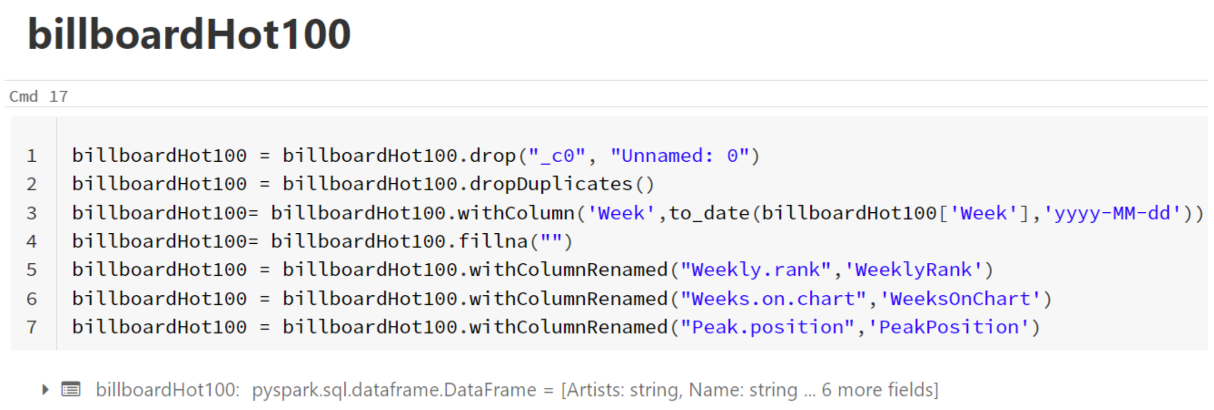


artistDf

Cmd 15

```python
artistDf = artistDf.drop("X")
artistDf = artistDf.dropDuplicates()
artistDf = artistDf.fillna("")
```

- Se eliminó la columna “X”
- Se eliminaron duplicados
- Se rellenaron valores nulls



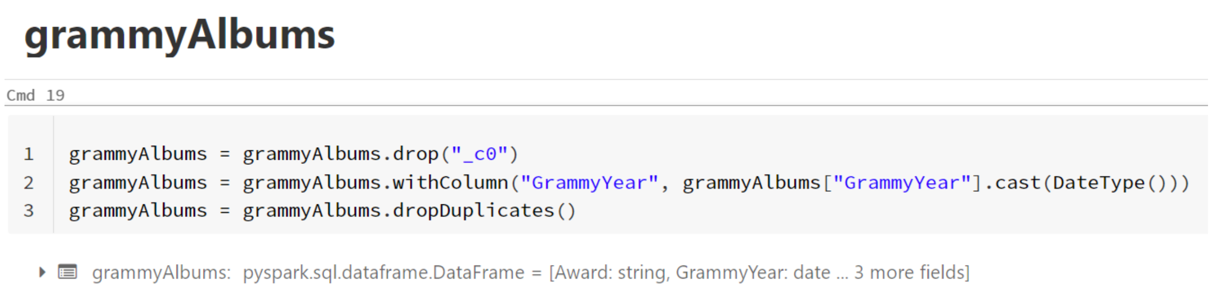
billboardHot100

Cmd 17

```python
billboardHot100 = billboardHot100.drop("_c0", "Unnamed: 0")
billboardHot100 = billboardHot100.dropDuplicates()
billboardHot100= billboardHot100.withColumn('Week',to_date(billboardHot100['Week'],'yyyy-MM-dd'))
billboardHot100= billboardHot100.fillna("")
billboardHot100 = billboardHot100.withColumnRenamed("Weekly.rank",'WeeklyRank')
billboardHot100 = billboardHot100.withColumnRenamed("Weeks.on.chart",'WeeksOnChart')
billboardHot100 = billboardHot100.withColumnRenamed("Peak.position",'PeakPosition')
```

```
billboardHot100: pyspark.sql.dataframe.DataFrame = [Artists: string, Name: string ... 6 more fields]
```

- Se eliminaron columnas “_c0” y “Unnamed: 0”
- Se borraron duplicados
- Se cambió el tipo a fecha
- Se rellenaron valores nulls
- Se renombraron columnas



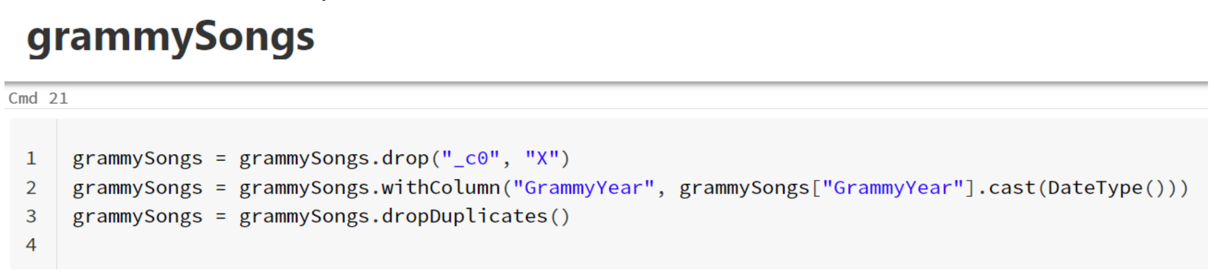
grammyAlbums

Cmd 19

```python
grammyAlbums = grammyAlbums.drop("_c0")
grammyAlbums = grammyAlbums.withColumn("GrammyYear", grammyAlbums["GrammyYear"].cast(DateType()))
grammyAlbums = grammyAlbums.dropDuplicates()
```

```
grammyAlbums: pyspark.sql.dataframe.DataFrame = [Award: string, GrammyYear: date ... 3 more fields]
```

- Se eliminó la columna “_c0”
- Se cambió el tipo de dato a tipo fecha
- Se eliminaron duplicados

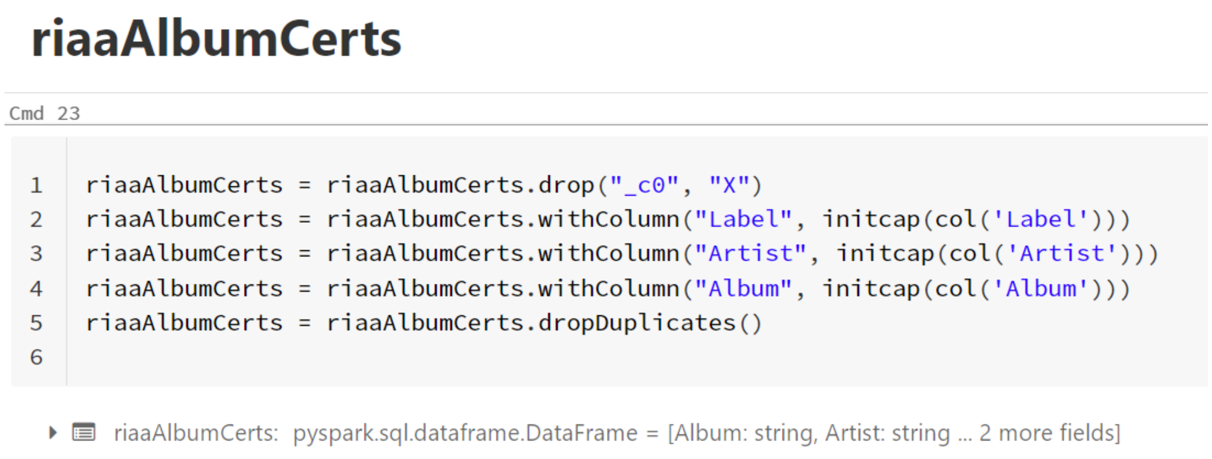


grammySongs

Cmd 21

```python
grammySongs = grammySongs.drop("_c0", "X")
grammySongs = grammySongs.withColumn("GrammyYear", grammySongs["GrammyYear"].cast(DateType()))
grammySongs = grammySongs.dropDuplicates()

```

- Se eliminó la columna “_c0”
- Se cambió el tipo de dato a tipo fecha
- Se eliminaron duplicados



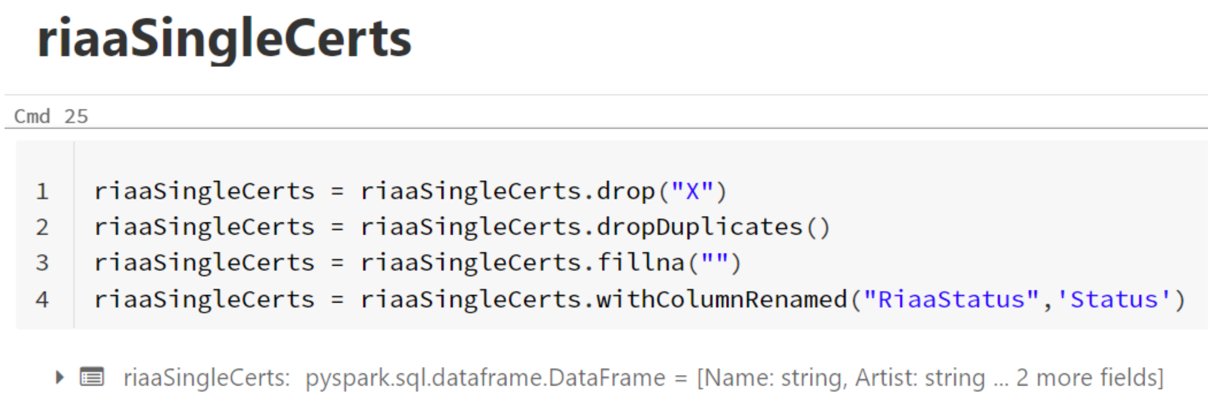
riaaAlbumCerts

Cmd 23

```
riaaAlbumCerts = riaaAlbumCerts.drop("_c0", "X")
riaaAlbumCerts = riaaAlbumCerts.withColumn("Label", initcap(col('Label')))
riaaAlbumCerts = riaaAlbumCerts.withColumn("Artist", initcap(col('Artist')))
riaaAlbumCerts = riaaAlbumCerts.withColumn("Album", initcap(col('Album')))
riaaAlbumCerts = riaaAlbumCerts.dropDuplicates()

```

```
riaaAlbumCerts:  pyspark.sql.dataframe.DataFrame = [Album: string, Artist: string ... 2 more fields]
```

- Se eliminaron columnas "_c0" y "x"
- Se modificaron mayúsculas y minúsculas
- Se eliminaron duplicados



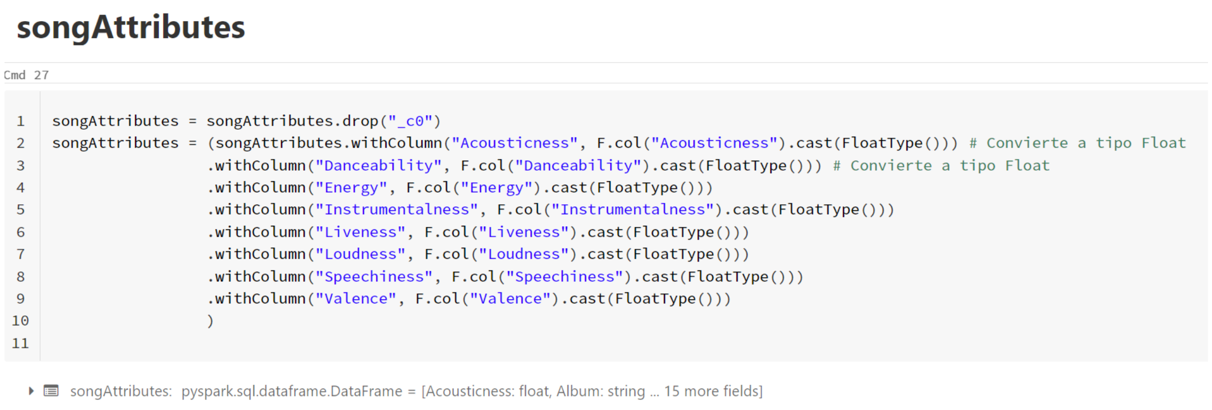
riaaSingleCerts

Cmd 25

```
riaaSingleCerts = riaaSingleCerts.drop("X")
riaaSingleCerts = riaaSingleCerts.dropDuplicates()
riaaSingleCerts = riaaSingleCerts.fillna("")
riaaSingleCerts = riaaSingleCerts.withColumnRenamed("RiaaStatus",'Status')
```

```
riaaSingleCerts:  pyspark.sql.dataframe.DataFrame = [Name: string, Artist: string ... 2 more fields]
```

- Se eliminó columna x
- Se eliminaron duplicados
- Se rellenaron nulls
- Se renombró columna "RiaaStatus" a "Status"



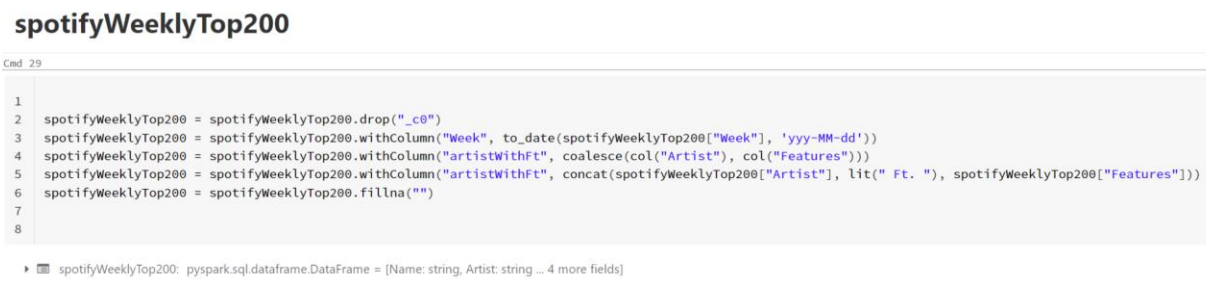
songAttributes

Cmd 27

```
songAttributes = songAttributes.drop("_c0")
songAttributes = (songAttributes.withColumn("Acousticness", F.col("Acousticness").cast(FloatType())) # Convierte a tipo Float
                  .withColumn("Danceability", F.col("Danceability").cast(FloatType())) # Convierte a tipo Float
                  .withColumn("Energy", F.col("Energy").cast(FloatType()))
                  .withColumn("Instrumentalness", F.col("Instrumentalness").cast(FloatType()))
                  .withColumn("Liveness", F.col("Liveness").cast(FloatType()))
                  .withColumn("Loudness", F.col("Loudness").cast(FloatType()))
                  .withColumn("Speechiness", F.col("Speechiness").cast(FloatType()))
                  .withColumn("Valence", F.col("Valence").cast(FloatType()))
                  )

```

```
songAttributes:  pyspark.sql.dataframe.DataFrame = [Acousticness: float, Album: string ... 15 more fields]
```

- Se eliminó columna "_c0"
- Se cambió el tipo de dato según corresponda



spotifyWeeklyTop200

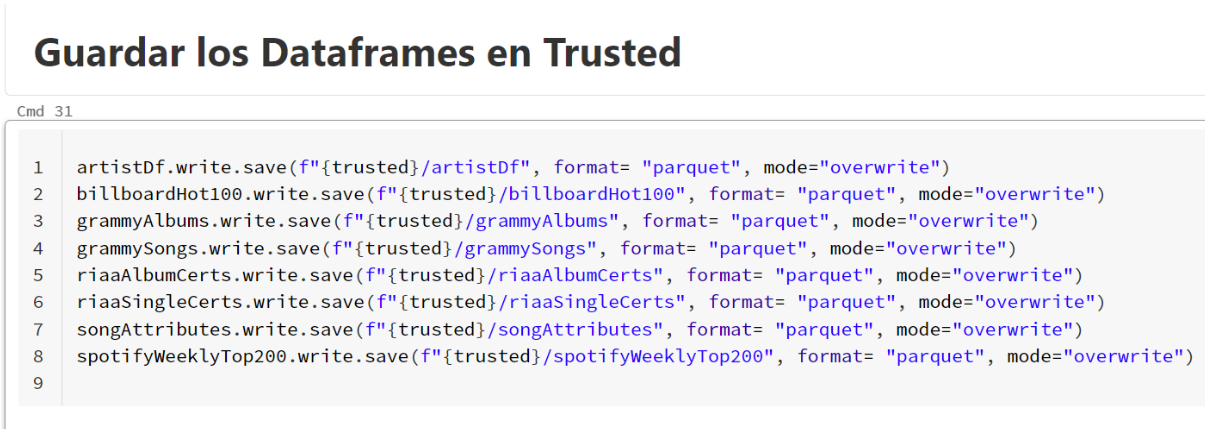
Cmd 29

```python

spotifyWeeklyTop200 = spotifyWeeklyTop200.drop("_c0")
spotifyWeeklyTop200 = spotifyWeeklyTop200.withColumn("Week", to_date(spotifyWeeklyTop200["Week"], 'yyy-MM-dd'))
spotifyWeeklyTop200 = spotifyWeeklyTop200.withColumn("artistWithFt", coalesce(col("Artist"), col("Features")))
spotifyWeeklyTop200 = spotifyWeeklyTop200.withColumn("artistWithFt", concat(spotifyWeeklyTop200["Artist"], lit(" Ft. "), spotifyWeeklyTop200["Features"]))
spotifyWeeklyTop200 = spotifyWeeklyTop200.fillna("")


```

```
spotifyWeeklyTop200:  pyspark.sql.dataframe.DataFrame = [Name: string, Artist: string ... 4 more fields]
```

- Se eliminó columna “_c0”
- Se cambió el tipo de dato a tipo fecha
- Se crearon columnas nuevas
- Se rellenaron nulls

Luego de la transformación y limpieza se guardan los dataframes en formato parquet:

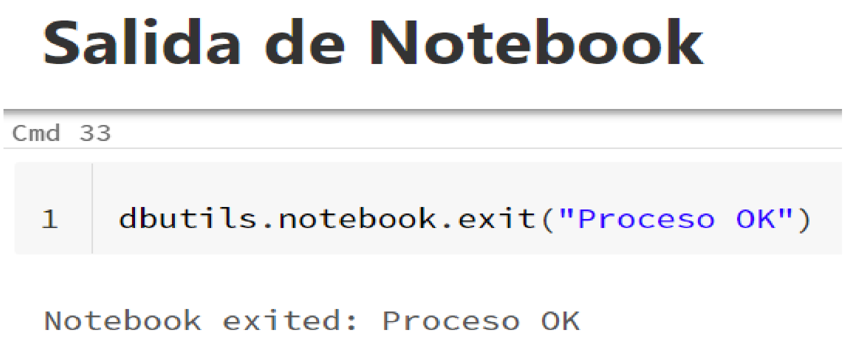


Guardar los Dataframes en Trusted

Cmd 31

```python
artistDf.write.save(f"{trusted}/artistDf", format= "parquet", mode="overwrite")
billboardHot100.write.save(f"{trusted}/billboardHot100", format= "parquet", mode="overwrite")
grammyAlbums.write.save(f"{trusted}/grammyAlbums", format= "parquet", mode="overwrite")
grammySongs.write.save(f"{trusted}/grammySongs", format= "parquet", mode="overwrite")
riaaAlbumCerts.write.save(f"{trusted}/riaaAlbumCerts", format= "parquet", mode="overwrite")
riaaSingleCerts.write.save(f"{trusted}/riaaSingleCerts", format= "parquet", mode="overwrite")
songAttributes.write.save(f"{trusted}/songAttributes", format= "parquet", mode="overwrite")
spotifyWeeklyTop200.write.save(f"{trusted}/spotifyWeeklyTop200", format= "parquet", mode="overwrite")

```

Finalmente con la correcta ejecución de todos los pasos se realiza la **salida de Notebook**



Salida de Notebook

Cmd 33

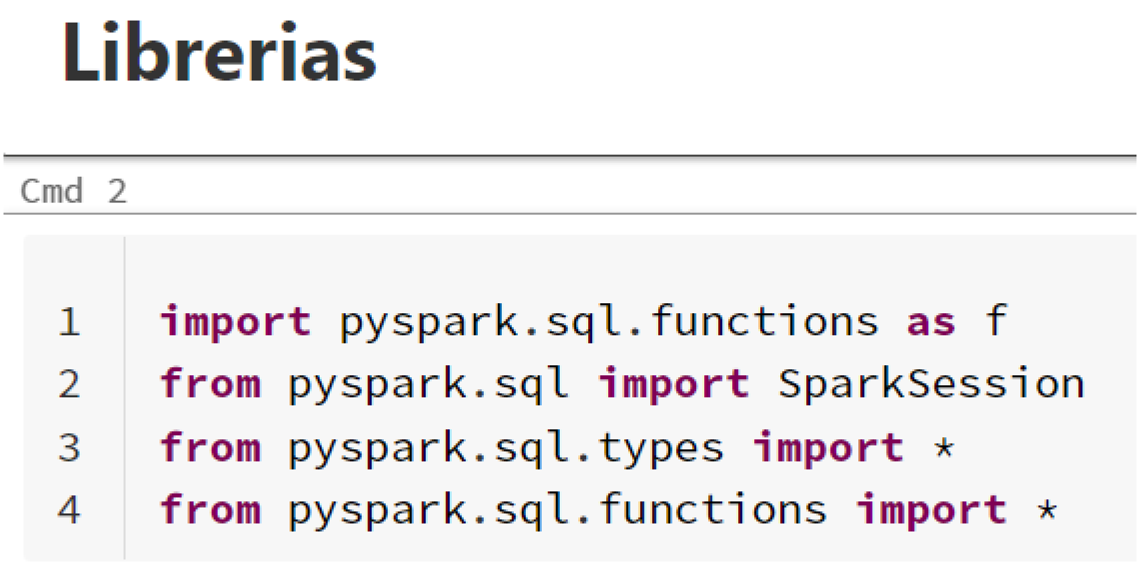
```python
dbutils.notebook.exit("Proceso OK")
```

```
Notebook exited: Proceso OK
```

Trusted_to_refined

En esta etapa generamos nuevas tablas o tablones (según corresponda el caso) para poder hacer consultas y análisis de los datos. Pueden surgir transformaciones, pero deberían estar en la etapa previa. Luego se exportan a PowerBI para generar los gráficos.

Importamos librerías a utilizar:

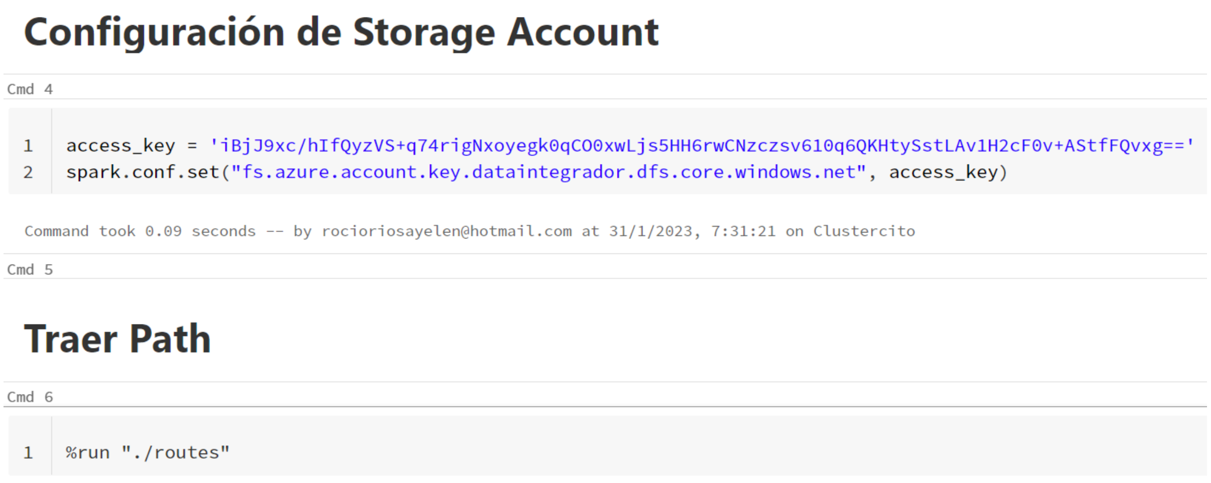


Librerias

Cmd 2

```
import pyspark.sql.functions as f
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *
```

Creamos las conexiones necesarias



Configuración de Storage Account

Cmd 4

```
access_key = 'iBjJ9xc/hIfQyzVS+q74rigNxoyegk0qCO0xwLjs5HH6rwCNzczsv610q6QKHtySstLAv1H2cF0v+AStfFQvxg=='
spark.conf.set("fs.azure.account.key.dataintegrador.dfs.core.windows.net", access_key)

Command took 0.09 seconds -- by rocioriosayelen@hotmail.com at 31/1/2023, 7:31:21 on Clustercito
```

Cmd 5

Traer Path

Cmd 6

```
%run "./routes"
```

Crear los Path

Cmd 9

```python
artistDfPATH = trusted + "/artistDf"
billboardHot100PATH = trusted + "/billboardHot100"
grammyAlbumsPATH = trusted + "/grammyAlbums"
grammySongsPATH = trusted + "/grammySongs"
riaaAlbumCertsPATH = trusted + "/riaaAlbumCerts"
riaaSingleCertsPATH = trusted + "/riaaSingleCerts"
songAttributesPATH = trusted + "/songAttributes"
spotifyWeeklyTop200PATH = trusted + "/spotifyWeeklyTop200"

```

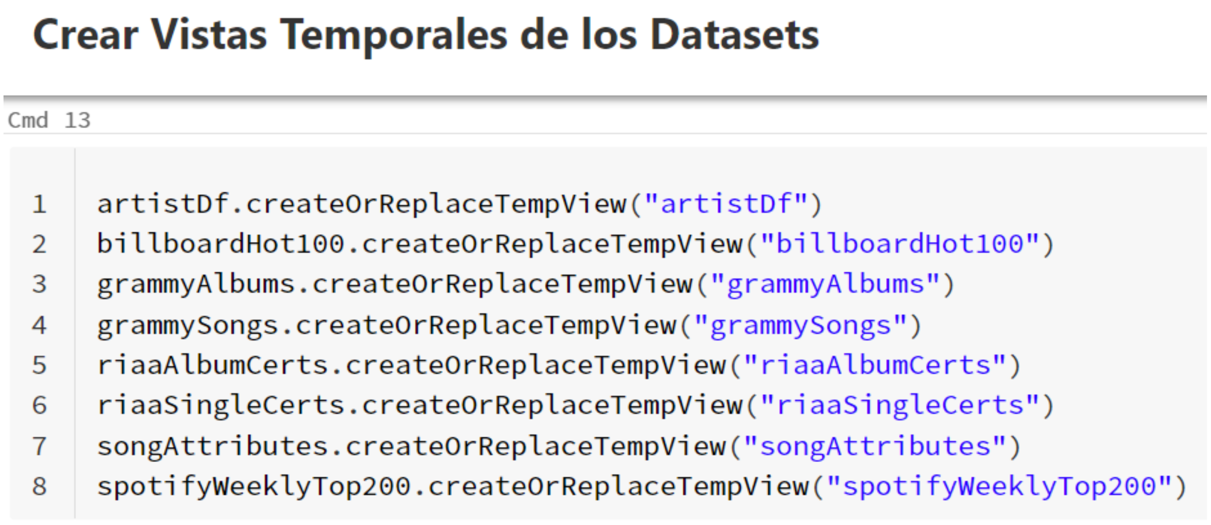
Cargar los Datasets

:md 11

```python
artistDf = spark.read.parquet(artistDfPATH)
billboardHot100 = spark.read.parquet(billboardHot100PATH)
grammyAlbums = spark.read.parquet(grammyAlbumsPATH)
grammySongs  = spark.read.parquet(grammySongsPATH)
riaaAlbumCerts= spark.read.parquet(riaaAlbumCertsPATH)
riaaSingleCerts= spark.read.parquet(riaaSingleCertsPATH)
songAttributes = spark.read.parquet(songAttributesPATH)
spotifyWeeklyTop200 = spark.read.parquet(spotifyWeeklyTop200PATH)
```

```
▸ (8) Spark Jobs
▸ artistDf:  pyspark.sql.dataframe.DataFrame = [Artist: string, Followers: string ... 5 more fields]
▸ billboardHot100:  pyspark.sql.dataframe.DataFrame = [Artists: string, Name: string ... 6 more fields]
▸ grammyAlbums:  pyspark.sql.dataframe.DataFrame = [Award: string, GrammyYear: date ... 3 more fields]
▸ grammySongs:  pyspark.sql.dataframe.DataFrame = [GrammyAward: string, GrammyYear: date ... 3 more fields]
▸ riaaAlbumCerts:  pyspark.sql.dataframe.DataFrame = [Album: string, Artist: string ... 2 more fields]
▸ riaaSingleCerts:  pyspark.sql.dataframe.DataFrame = [Name: string, Artist: string ... 2 more fields]
▸ songAttributes:  pyspark.sql.dataframe.DataFrame = [Acousticness: float, Album: string ... 15 more fields]
▸ spotifyWeeklyTop200:  pyspark.sql.dataframe.DataFrame = [Name: string, Artist: string ... 4 more fields]
```

Se crean las vistas temporales para poder hacer consultas SQL



Crear Vistas Temporales de los Datasets

Cmd 13

```
artistDf.createOrReplaceTempView("artistDf")
billboardHot100.createOrReplaceTempView("billboardHot100")
grammyAlbums.createOrReplaceTempView("grammyAlbums")
grammySongs.createOrReplaceTempView("grammySongs")
riaaAlbumCerts.createOrReplaceTempView("riaaAlbumCerts")
riaaSingleCerts.createOrReplaceTempView("riaaSingleCerts")
songAttributes.createOrReplaceTempView("songAttributes")
spotifyWeeklyTop200.createOrReplaceTempView("spotifyWeeklyTop200")
```

En este caso, creamos 3 “mini tablones” para cada caso de uso:

Tablon1:

```
artistGrammy = spark.sql("""
SELECT
a.Artist,
a.Followers,
a.Gender,
ga.Award as AwardAlbum,
ga.GrammyYear as GrammyYearAlbum,
ga.Genre as GenreAlbum,
gs.GrammyAward as AwardSong,
gs.GrammyYear as GrammyYearSong,
gs.Genre as GenreSong
FROM
ArtistDf a
LEFT JOIN grammyAlbums ga ON a.Artist = ga.Artist
LEFT JOIN grammySongs gs ON a.Artist = gs.Artist
""")
```

```
artistGrammy: pyspark.sql.dataframe.DataFrame = [Artist: string, Followers: string ... 7 more fields]
```

Tablon2:

```
billbSpotify = spark.sql("""
SELECT
b.Artists,
b.Name,
b.WeeklyRank,
b.WeeksOnChart,
b.Week as WeekBB,
b.Date,
b.Genre,
s.Name as NameSpoti,
s.Features,
s.Week as WeekSpoti
FROM
billboardHot100 b
LEFT JOIN spotifyWeeklyTop200 s
ON b.Artists = s.Artist
""")


```

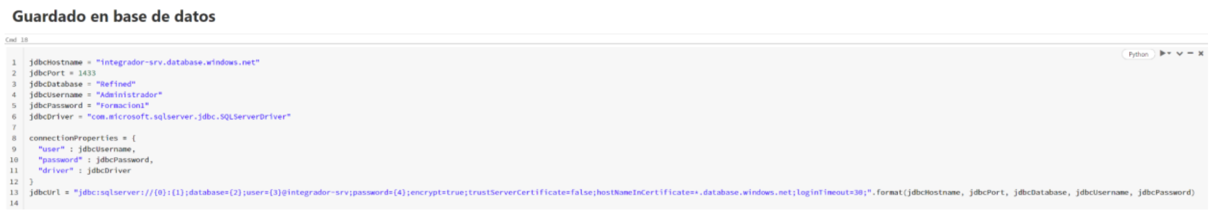
```
▸ billbSpotify:  pyspark.sql.dataframe.DataFrame = [Artists: string, Name: string ... 8 more fields]
```

Tablon3:

```
songs= spark.sql("""
SELECT
gs.Artist,
gs.Name as NameGrammy,
gs.GrammyAward,
gs.GrammyYear,
gs.Genre,
sa. Album,
sa.Danceability,
sa.Energy,
sa.Explicit,
sa.Name,
sa.Popularity
FROM
grammySongs gs
LEFT JOIN songAttributes sa
ON gs.Artist = sa.Artist
""")
```

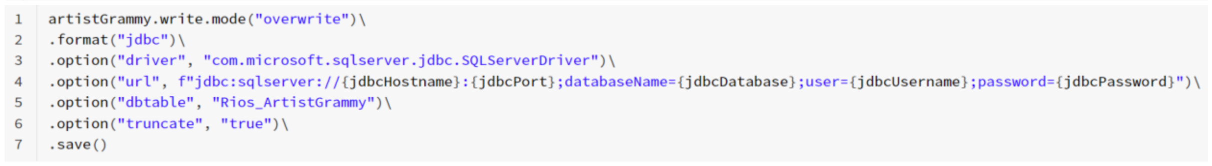
```
▸ songs:  pyspark.sql.dataframe.DataFrame = [Artist: string, NameGrammy: string ... 9 more fields]
```

Luego de la creación de los mini tablones se guardan en la base de datos esta vez “refinados”

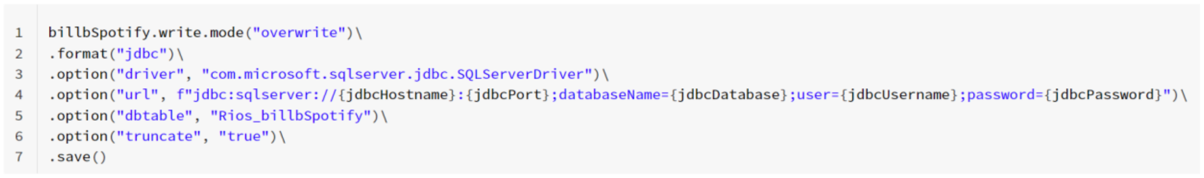


Guardado en base de datos

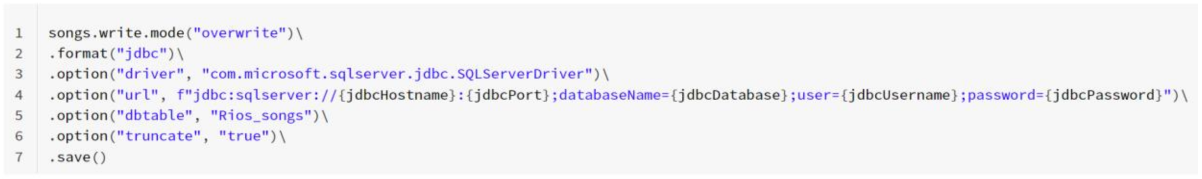
```python
jdbcHostname = "integrador-srv.database.windows.net"
jdbcPort = 1433
jdbcDatabase = "Refined"
jdbcUsername = "Administrador"
jdbcPassword = "Formacion1"
jdbcDriver = "com.microsoft.sqlserver.jdbc.SQLServerDriver"

connectionProperties = {
  "user" : jdbcUsername,
  "password" : jdbcPassword,
  "driver" : jdbcDriver
}
jdbcUrl = "jdbc:sqlserver://{0}:{1};database={2};user={3}@integrador-srv;password={4};encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;loginTimeout=30;".format(jdbcHostname, jdbcPort, jdbcDatabase, jdbcUsername, jdbcPassword)
```



```python
artistGrammy.write.mode("overwrite")\
.format("jdbc")\
.option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver")\
.option("url", f"jdbc:sqlserver://{jdbcHostname}:{jdbcPort};databaseName={jdbcDatabase};user={jdbcUsername};password={jdbcPassword}")\
.option("dbtable", "Rios_ArtistGrammy")\
.option("truncate", "true")\
.save()
```



```python
billbSpotify.write.mode("overwrite")\
.format("jdbc")\
.option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver")\
.option("url", f"jdbc:sqlserver://{jdbcHostname}:{jdbcPort};databaseName={jdbcDatabase};user={jdbcUsername};password={jdbcPassword}")\
.option("dbtable", "Rios_billbSpotify")\
.option("truncate", "true")\
.save()
```



```python
songs.write.mode("overwrite")\
.format("jdbc")\
.option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver")\
.option("url", f"jdbc:sqlserver://{jdbcHostname}:{jdbcPort};databaseName={jdbcDatabase};user={jdbcUsername};password={jdbcPassword}")\
.option("dbtable", "Rios_songs")\
.option("truncate", "true")\
.save()
```