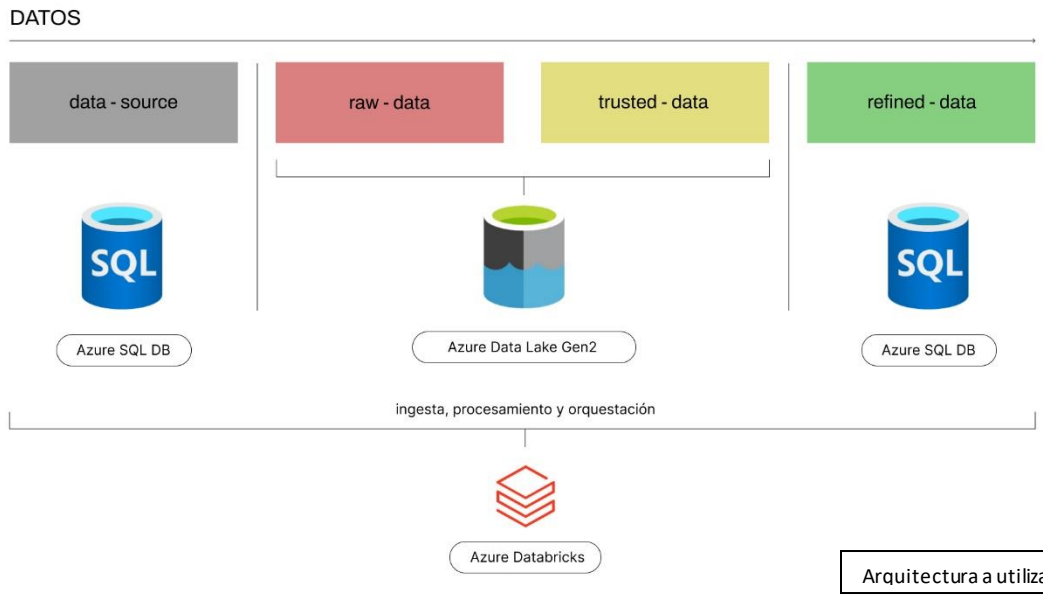
Documentación del proceso End-to-End para el cliente Escuelitas S.A.

En el escrito presente se dejará constancia del proceso end to end que se llevó a cabo para demostrar los conocimientos aprendidos durante la formación del mes de Enero, 2023 mediante la obtención de resultados analíticos sobre los datos que se dispone de la revista musical Billboard que brindan información sobre los últimos veinte años de la industria mediante los siguientes dataframes: artistas, Billboard Hot 100, Grammy Albums, Grammy Songs, RIAA Album Certs, RIAA Single Certs, Song Attributes y por último Spotify Weekly Top 200 Streams.

Esta documentación está destinada a 'Escuelitas' S.A. una compañía discográfica que está interesada en conocer las tendencias de la industria musical para mejorar su estrategia de marketing y financiamiento de nuevos artistas. Con el foco en buscar futuras promesas solistas. El análisis de datos de las estadísticas de la revista Billboard les permitiría a los ejecutivos de la compañía ver qué géneros y artistas están teniendo éxito, así como identificar patrones y tendencias en la popularidad de la música. Esto les ayudará a tomar decisiones informadas sobre qué artistas apoyar y en qué géneros enfocarse para maximizar su inversión.

Los pasos que se detallarán a lo largo del documento tienen como foco cumplir el objetivo del cliente respetando la arquitectura de la imagen a continuación.



Arquitectura a utilizar

En la primera etapa, denominada **Origin to Raw**, se creó una notebook en Databricks llamada 'origin_to_raw' y, luego de importar las librerías pySpark necesarias, se levantaron los datos que se encontraban en una Base de datos origen SQL, llamada "DB-Bllbd" utilizando una conexión a SQL desde Databricks para llevar los datos al Data Lake.

Conexión a la BD de Billboard

```
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)
```


```
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)
```


Conexión al Datalake y Key de la formación

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

Una vez establecida la conexión, se generó cada uno de los Dataframes mencionados mediante un método visto durante la formación que utiliza el esquema con información de cada uno de los df para iterar sobre ellos e irlos leyendo y escribiendo en formato csv en el directorio del datalake integrador-longas-elias/rawdata/azuresqldb respectivamente sin necesidad de hacerlo manualmente uno por uno.

Generación del information schema

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

Filtrando lo necesario del information schema

```
tables = information_schema.filter(
    (F.col("TABLE_SCHEMA") == "dbo") & (F.col("TABLE_TYPE") == "BASE TABLE")
).select(F.col("TABLE_SCHEMA").alias("SCHEMA"), "TABLE_NAME")
```

```
# funcion que itera las tablas generando y escribiendo automáticamente cada df
en rawdata
for table in tables.collect():
    df = spark.read.format("jdbc")\
    .option("driver", jdbcDriver)\
    .option("url", jdbcUrl)\
    .option("dbtable", f"{table[0]}.{table[1]}")\
    .option("header", True)\
    .load()

    df.write.save(f"{raw_data}/azuresqldb/{table[0]}.{table[1]}", format="csv",
mode="overwrite", header = True)
```

Bucle que itera leyendo y escribiendo cada uno de los df del origen a raw data

La segunda etapa, **Raw to Trusted**, se trabajó desde la notebook de Databricks llamada 'raw_to_trusted', posteriormente de haber importado las librerías/módulos necesarios y de configurar nuevamente el *acces key*, se cargaron todos los datasets.

```
artist = spark.read.option("header",True).csv(artistPATH)
bbHot100 = spark.read.option("header",True).csv(billboardHot100PATH)
grammyAlbums = spark.read.option("header",True).csv(grammyAlbumsPATH)
grammySongs = spark.read.option("header",True).csv(grammySongsPATH)
riaaAlbumCerts = spark.read.option("header",True).csv(riaaAlbumCertsPATH)
riaaSingleCerts = spark.read.option("header",True).csv(riaaSingleCertsPATH)
songAttributes = spark.read.option("header",True).csv(songAttributesPATH)
spotifyWeeklyTop200 =
spark.read.option("header",True).csv(spotifyWeeklyTop200PATH)
```

Carga de datasets

Posteriormente, se realizaron las transformaciones consideradas pertinentes para el posterior análisis de cada uno de los Dataframes. Las más recurrentes como se mostrará a continuación utilizando el ejemplo del df de artistas, son la eliminación de columnas que no aportan para el análisis, borrado de duplicados, quitar valores nulos reemplazandolos por '0' o '' según corresponda y casteos de datatypes de columnas de string a int, float, date, etc.

```
#Eliminando la columna "X"
artist = artist.drop("X")
```

```
# Borrando registros duplicados si los hay
artist = artist.dropDuplicates()
```

```
# Reemplazando todos los nulos por "0" o "" respectivamente
artist = artist.fillna(value = 0).fillna("")
```

```
#Casteando columnas de str a int
artist = artist.withColumn("Followers", col("Followers").cast("int"))
artist = artist.withColumn("NumAlbums", col("NumAlbums").cast("int"))
```

Por otra parte, en el Dataframe de Billboard Hot 100 se tuvo que hacer un tratamiento especial a la columna 'Date' que poseía datos 'string' como por ejemplo 'April 5, 2019' con el fin de castear dicha columna a tipo de dato 'date'. En principio se

le cambió su nombre a 'Release_date' para evitar confusiones de interpretación. Luego se aplicaron transformaciones a la columna para quede dividida en tres columnas distintas cada una acorde al año, mes y día.

```
# Separando el mes y dia con " "
split_test2 = F.split(bbHot100['Rel_mon_day'], ' ')
bbHot100 = bbHot100.withColumn('Rel_month', split_test2.getItem(0))
bbHot100 = bbHot100.withColumn('Rel_day', split_test2.getItem(1))
```

```
# Separando el año con ","
split_test = F.split(bbHot100['Release_date'], ',')
bbHot100 = bbHot100.withColumn('Rel_year', split_test.getItem(1))
bbHot100 = bbHot100.withColumn('Rel_mon_day', split_test.getItem(0))
```

Sin embargo, como aún había problema con los meses ya que estaban escritos por sus nombres y no por números, se aplicó una función que recibiendo un dataframe, una columna y un diccionario, intercambia los valores de la columna acorde al diccionario indicado. Permitiendo así que la columna quede en un formato casteable a 'Date', dropeando luego las columnas innecesarias o redundantes.

```
# función que recibe un dataframe, una columna a modificar y un diccionario con los nuevos
valores los cuales va reemplazando acorde el mismo
def reemplazar_valores(df, nombre_columna, nuevos_valores):
    reemplazar_val = udf(lambda x: nuevos_valores.get(x, x))
    df = df.withColumn(nombre_columna, reemplazar_val(nombre_columna))
    return df

months_dict = {'January': '1', 'February': '2', 'March': '3', 'April': '4', 'May': '5',
'June': '6',
               'July': '7', 'August': '8', 'September': '9', 'October': '10', 'November':
'11', 'December': '12'}
```

```
bbHot100 = reemplazar_valores(bbHot100, 'Rel_month', months_dict)
```

Finalmente, luego de aplicar las transformaciones comunes ya mencionadas en el resto de los dataframes, se procedió al guardado de los mismos en formato parquet en el directorio del Datalake integrador-longas-elias/trusted-data/azuresqldb.

```
artist.write.save(f"{trusted_data}/azuresqldb/dbo.artistDf", format="parquet", mode="overwrite")
bbHot100.write.save(f"{trusted_data}/azuresqldb/dbo.billboardHot100_1999_2019", format="parquet", mode="overwrite")
grammyAlbums.write.save(f"{trusted_data}/azuresqldb/dbo.grammyAlbums_1999_2019", format="parquet", mode="overwrite")
grammySongs.write.save(f"{trusted_data}/azuresqldb/dbo.grammySongs_1999_2019", format="parquet", mode="overwrite")
riaaAlbumCerts.write.save(f"{trusted_data}/azuresqldb/dbo.riaaAlbumCerts_1999_2019", format="parquet", mode="overwrite")
riaaSingleCerts.write.save(f"{trusted_data}/azuresqldb/dbo.riaaSingleCerts_1999_2019", format="parquet", mode="overwrite")
songAttributes.write.save(f"{trusted_data}/azuresqldb/dbo.songAttributes_1999_2019", format="parquet", mode="overwrite")
spotifyWeeklyTop200.write.save(f"{trusted_data}/azuresqldb/dbo.spotifyWeeklyTop200Streams", format="parquet", mode="overwrite")
```

En la tercera etapa, **Trusted to Refined**, se trabajó nuevamente desde Databricks en la notebook llamada 'trusted_to_refined', volviendo a importar las librerías necesarias y a configurar el access key. Seguidamente, se cargaron los Datasets almacenados en la etapa anterior en formato parquet.

```
artistPATH = trusted_data + '/azuresqldb/dbo.artistDf'
billboardHot100PATH = trusted_data + '/azuresqldb/dbo.billboardHot100_1999_2019'
grammyAlbumsPATH = trusted_data + '/azuresqldb/dbo.grammyAlbums_1999_2019'
grammySongsPATH = trusted_data + '/azuresqldb/dbo.grammySongs_1999_2019'
riaaAlbumCertsPATH = trusted_data + '/azuresqldb/dbo.riaaAlbumCerts_1999_2019'
riaaSingleCertsPATH = trusted_data + '/azuresqldb/dbo.riaaSingleCerts_1999_2019'
songAttributesPATH = trusted_data + '/azuresqldb/dbo.songAttributes_1999_2019'
spotifyWeeklyTop200PATH = trusted_data + '/azuresqldb/dbo.spotifyWeeklyTop200Streams'
```

```
artist = spark.read.parquet(artistPATH)
bbHot100 = 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)
```

Se les creó una vista temporal mediante el método .createOrReplaceTempView para poder ejecutar consultas SQL en los Dataframes

```
artist.createOrReplaceTempView("artist")
bbHot100.createOrReplaceTempView("bbHot100")
grammyAlbums.createOrReplaceTempView("grammyAlbums")
grammySongs.createOrReplaceTempView("grammySongs")
riaaAlbumCerts.createOrReplaceTempView("riaaAlbumCerts")
riaaSingleCerts.createOrReplaceTempView("riaaSingleCerts")
songAttributes.createOrReplaceTempView("songAttributes")
spotifyWeeklyTop200.createOrReplaceTempView("spotifyWeeklyTop200")
```

A continuación, mediante Spark SQL se armaron diez tablones enfocados a los futuros análisis que se le realizarán de los datos. Seis tablones con datos sobre artistas solitas con reconocimientos en diferentes industrias(BillBoard, Grammy, RIAA, Spotify) y otros cuatro tablones que contienen información de canciones con sus atributos que destacaron en las industrias ya mencionadas. A modo ejemplo, se muestra cómo se construyó el tablón de solitas con reconocimientos en el Hot 100 de Billboard:

```
tablon_solistas_bbHot100 = spark.sql("""
SELECT
/*art.Artist,*/
bb.Artists,
art.Followers,
/*art.Genres,*/
art.NumAlbums,
art.YearFirstAlbum,
art.Gender,
art.Group_Solo,
bb.Name AS songName,
bb.Weeky_rank AS songWeekly_rank,
bb.Peak_position AS songWeekly_position,
bb.Weeks_on_chart AS songWeeks_on_chart,
bb.Week AS songWeek,
bb.Release_date AS songRelease_date,
bb.Genre AS songGenre
FROM artist art
LEFT JOIN bbhot100 bb ON (UPPER(bb.Artists)=UPPER(art.Artist))
WHERE bb.Artists IS NOT null AND art.Group_Solo="Solo"
""")
```

LEFT JOIN entre la tabla artista y la tabla bbhot100

Finalmente, al concluir la creación de los diez tablones, se estableció conexión a la Base de Datos destino llamada “Refined” para enviar los dataframes creados al Datawarehouse.

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

Conexión a DB

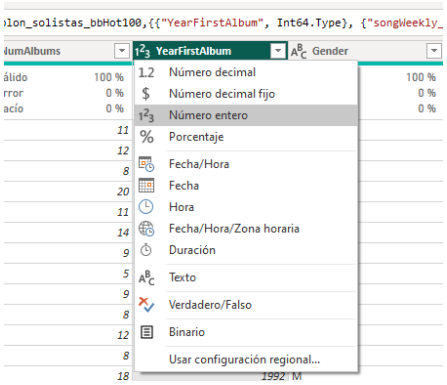
```
connectionProperties = {
  "user" : jdbcUsername,
  "password" : jdbcPassword,
  "driver" : jdbcDriver
}
jdbcUrl = "jdbc:sqlserver://{0}:{1};database={2};user={3}@integrador-srv;password=
{4};encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.net
;loginTimeout=30;".format(jdbcHostname,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

```
tablon_solistas_bbHot100.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", "Longas.tablon_solistas_bbHot100")\
.option("truncate", "true")\
.save()
```

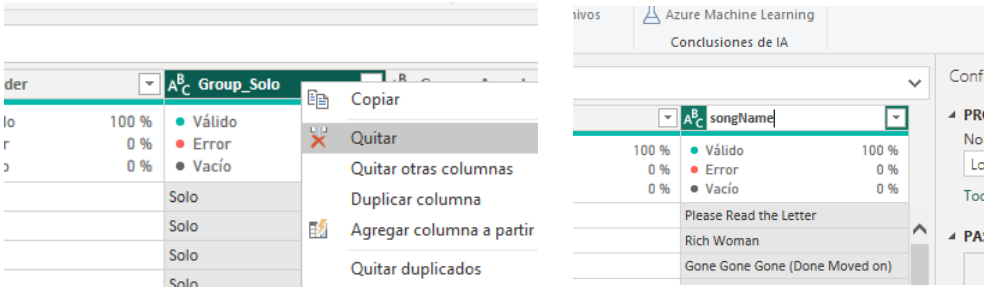
Ejemplo de escritura del tablon_solistas_bbHot100 en el Datawarehouse

Entrando en la cuarta y última etapa, se analizaron los datos mediante el servicio **Power BI** que proporciona visualizaciones con una interfaz simplificada para grandes conjuntos de datos.

Para empezar se continuó aplicando algunas transformaciones para terminar de refinar los tablones y que las visualizaciones sean más agradables. Por ejemplo, en los tablones que se crearon para los solistas reconocidos por la industria, se castearon algunas columnas que quedaron en string a int, como por ejemplo las columnas YearFirstAlbum, songWeekly_position(para la cual se tuvo que ajustar la configuración regional a Estados Unidos desde Opciones debido a que el separador decimal causaba problemas), songWeekly_rank, etc.

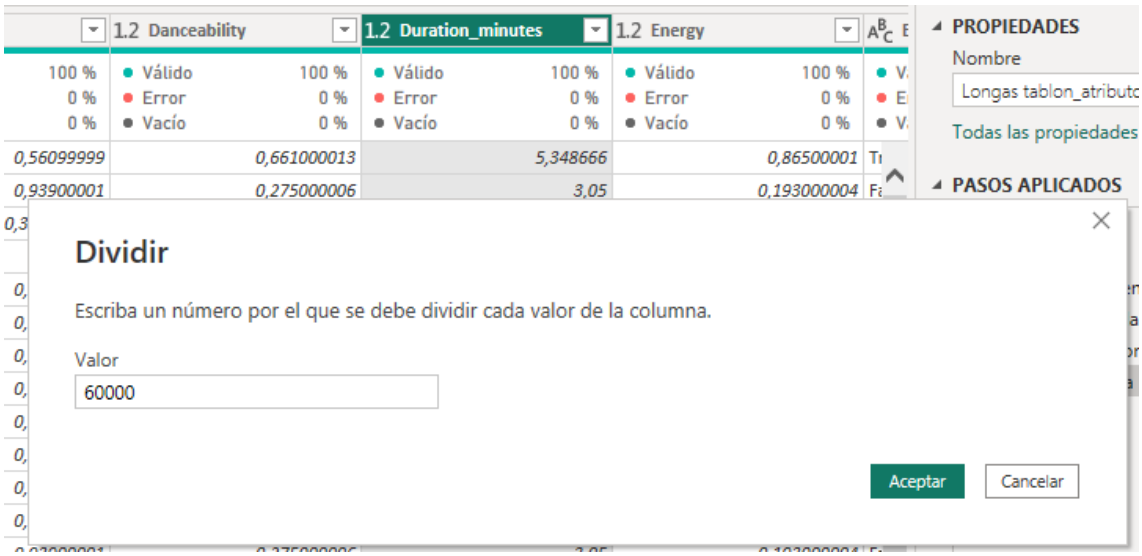


Además, se eliminó la columna Group_solo ya que, por el nombre de los tablones resulta obvio que se trata de artistas solistas. Se cambiaron nombres de columnas de name a songName. De Duration a Duration_seconds.

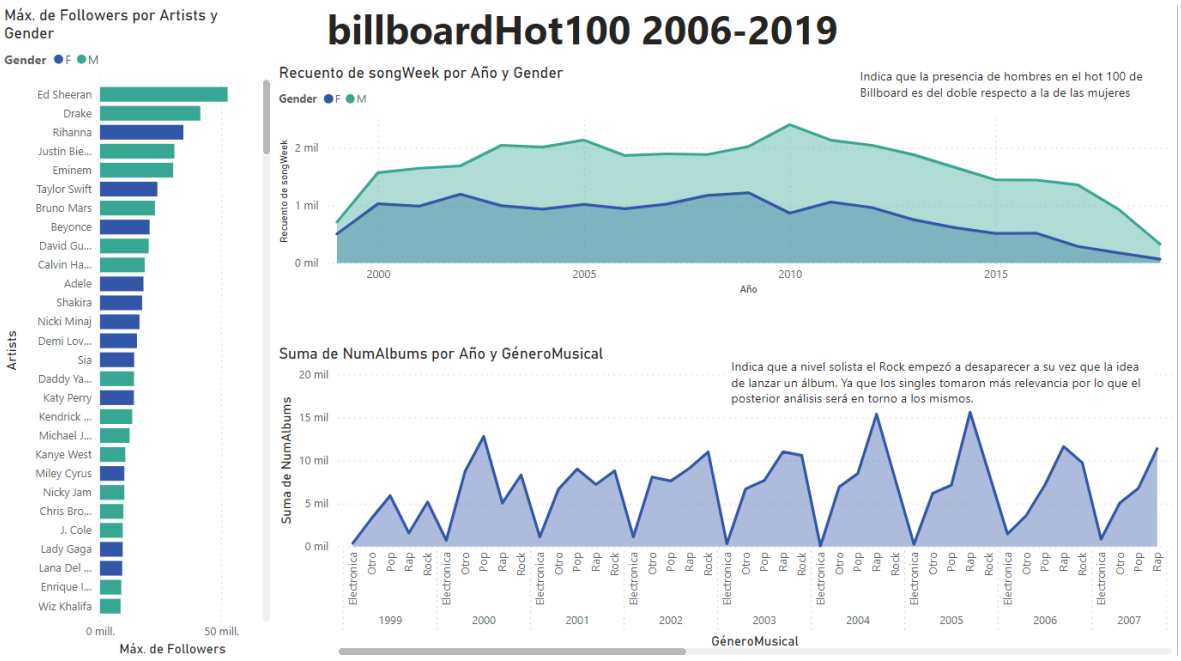


También, se crearon columnas condicionales para filtrar cada uno de los cuatro géneros considerados principales para el análisis(Rock, Electrónica, Rap, Pop), colocando valor '1' si el género aparece contenido dentro de la columna 'Genres' y '0' en caso de que no.

Por otra parte, para los tablones destinados a los atributos de las canciones reconocidas por la industria, a parte de realizar algunos pocos casteos de tipo de dato, o cambios de nombre de columna, se realizaron conversiones en la columna 'Duration' ya que sus valores estaban en milisegundos. Se los dividió por 60000 para que sus valores queden expresados en minutos y sean más fáciles de analizar gráficamente.



Una vez finalizadas las transformaciones en Power BI, se construyeron los gráficos considerados pertinentes para cada análisis. Principalmente gráficos de barra, torta y gráficos de líneas para marcar tendencias en el tiempo. Los mismos están expuestos en mayor profundidad en una presentación de Power Point. A continuación, un ejemplo de los gráficos desarrollados para el análisis de solistas destacados en el Billboard Hot 100.



De esta forma, se concluyó con el análisis de datos para el cliente 'Escuelitas' S.A. que consistió en respetar la arquitectura presentada al principio de la documentación en la que se extrayó la raw-data de Billboard desde una Base de Datos SQL para ingestarla y procesarla en el DataLake, convirtiéndola en trusted-data y enviarla al DataWarehouse(otra Base de Datos SQL) en forma de refined-data. Todo lo mencionado fue orquestado por Azure Databricks que proporciona un conjunto de herramientas para crear, implementar, compartir y mantener conjuntos de datos, a la vez que integra con el almacenamiento en la nube y seguridad en las cuentas de la nube.