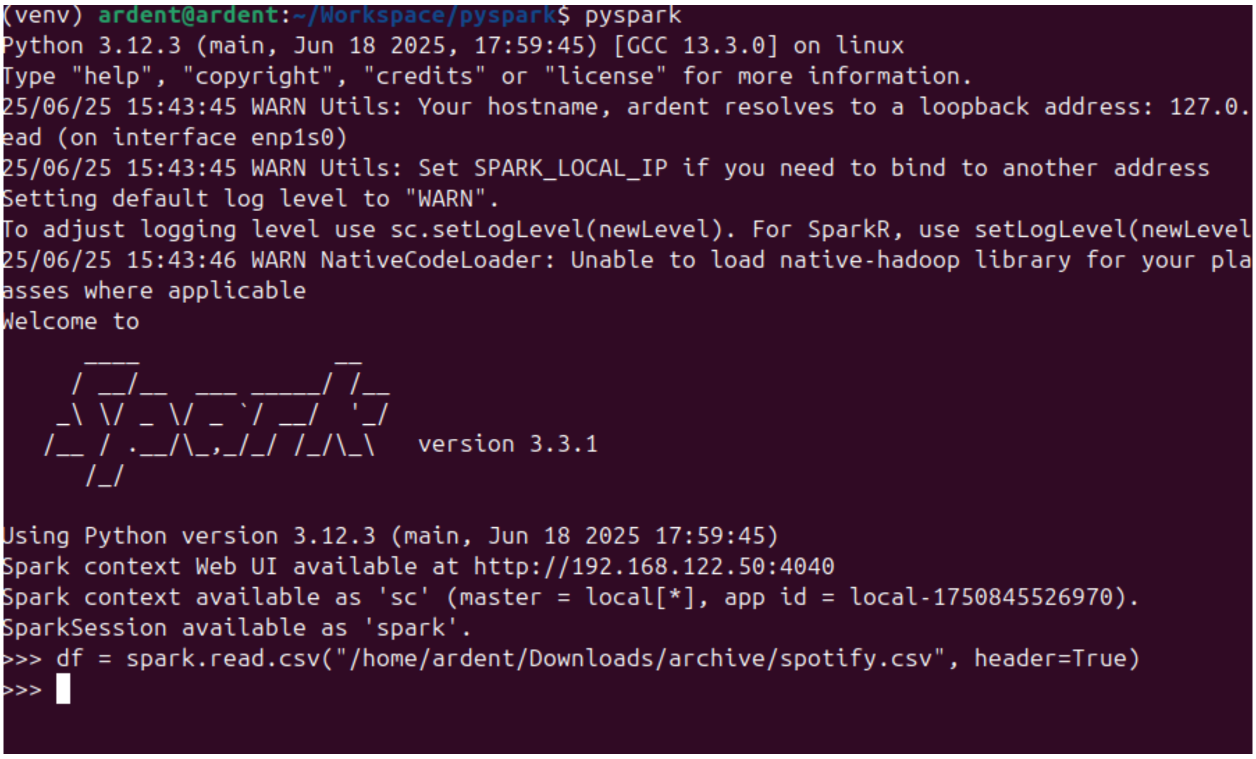
WEEK 2 Manual

1. Enable the virtual environment and type pyspark to open spark-shell

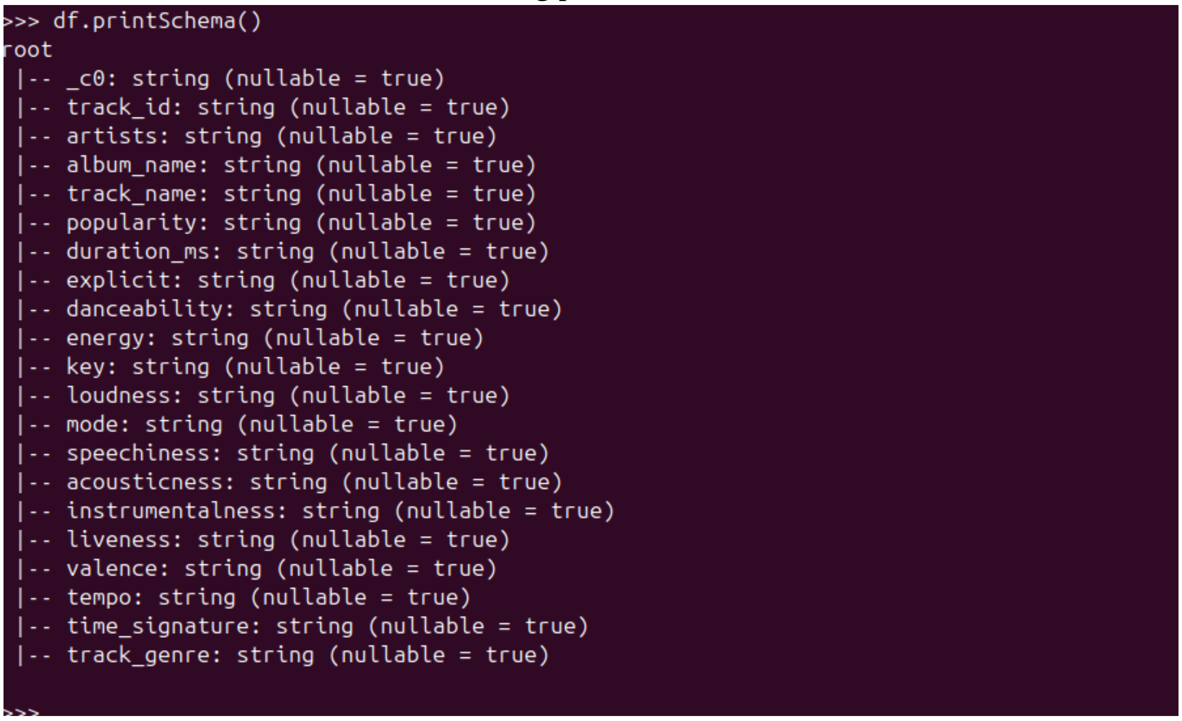
2. Read the csv file using either the relative or absolute path. Notice header=True, it will read the first row as header, test without using that parameter



```
(venv) ardent@ardent:~/Workspace/pyspark$ pyspark
Python 3.12.3 (main, Jun 18 2025, 17:59:45) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
25/06/25 15:43:45 WARN Utils: Your hostname, ardent resolves to a loopback address: 127.0.
ead (on interface enp1s0)
25/06/25 15:43:45 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel
25/06/25 15:43:46 WARN NativeCodeLoader: Unable to load native-hadoop library for your pla
asses where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.1
      /_/

Using Python version 3.12.3 (main, Jun 18 2025 17:59:45)
Spark context Web UI available at http://192.168.122.50:4040
Spark context available as 'sc' (master = local[*], app id = local-1750845526970).
SparkSession available as 'spark'.
>>> df = spark.read.csv("/home/ardent/Downloads/archive/spotify.csv", header=True)
>>> 
```

3. Let us view the columns in the data using printSchema

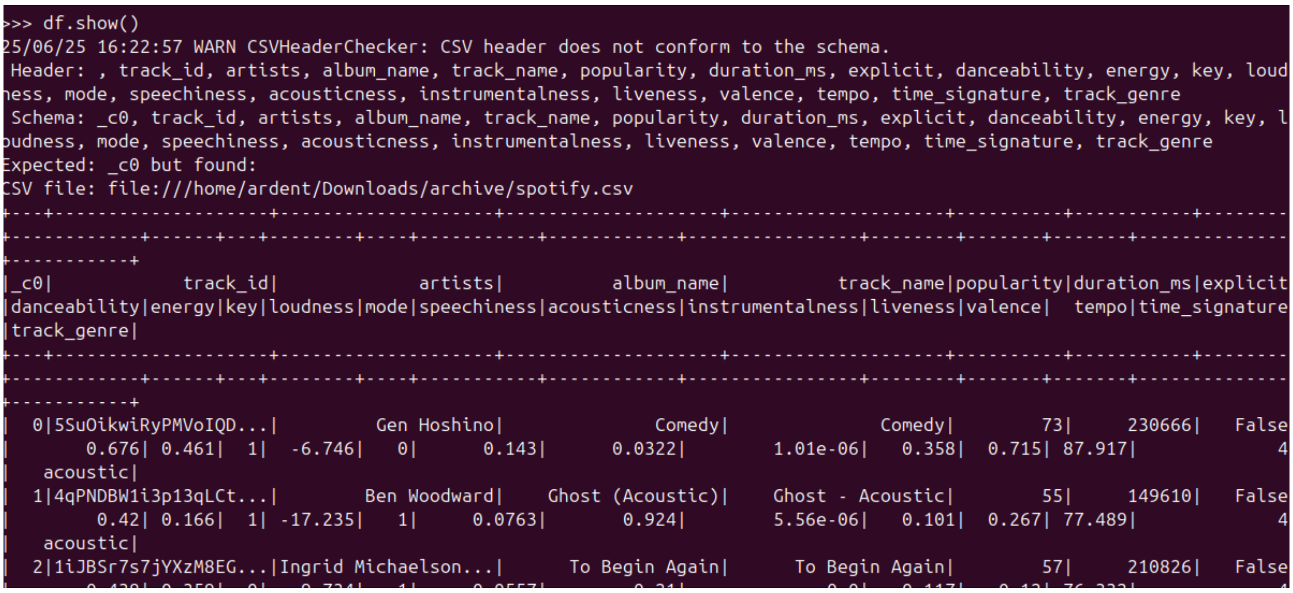


```
>>> df.printSchema()
root
 |-- _c0: string (nullable = true)
 |-- track_id: string (nullable = true)
 |-- artists: string (nullable = true)
 |-- album_name: string (nullable = true)
 |-- track_name: string (nullable = true)
 |-- popularity: string (nullable = true)
 |-- duration_ms: string (nullable = true)
 |-- explicit: string (nullable = true)
 |-- danceability: string (nullable = true)
 |-- energy: string (nullable = true)
 |-- key: string (nullable = true)
 |-- loudness: string (nullable = true)
 |-- mode: string (nullable = true)
 |-- speechiness: string (nullable = true)
 |-- acousticness: string (nullable = true)
 |-- instrumentalness: string (nullable = true)
 |-- liveness: string (nullable = true)
 |-- valence: string (nullable = true)
 |-- tempo: string (nullable = true)
 |-- time_signature: string (nullable = true)
 |-- track_genre: string (nullable = true)

>>>
```

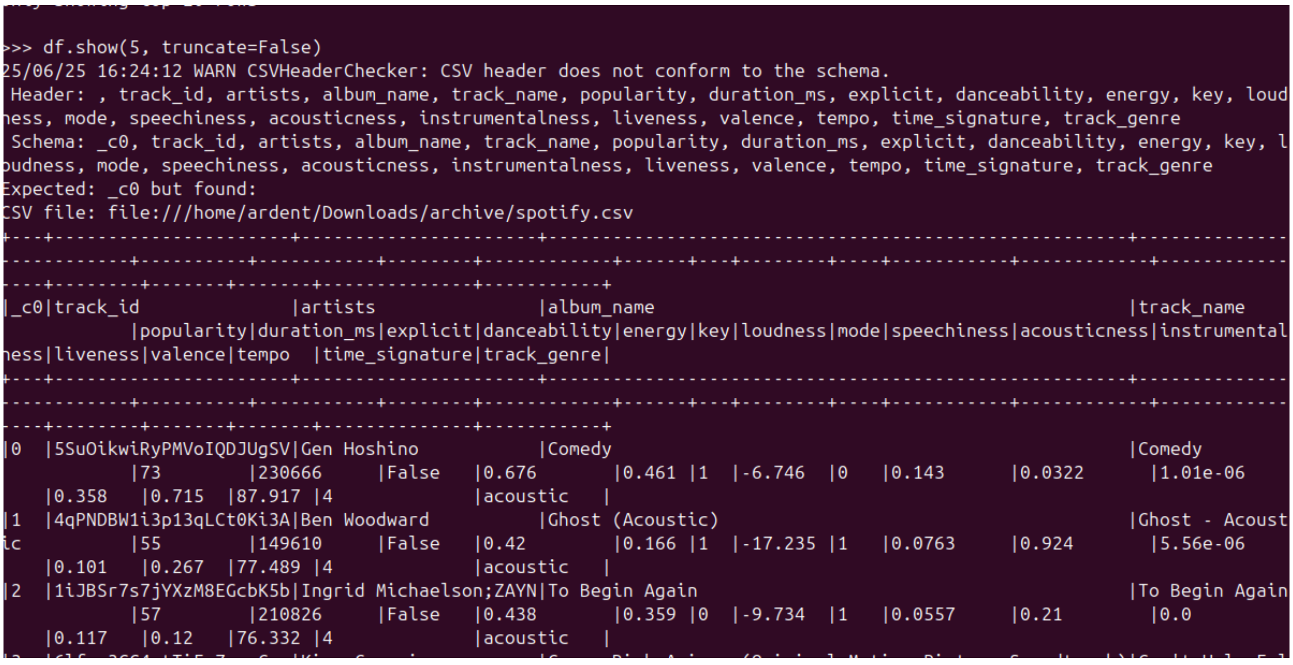
Notice the name and data type of the columns. The first column is shown as _c0, this is the default name given when spark cannot determine the name of the column. Also notice that all the columns are string, when pyspark cannot determine the data type it will set it as string

4. View the content of the file using show



```
>>> df.show()
25/06/25 16:22:57 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: , track_id, artists, album_name, track_name, popularity, duration_ms, explicit, danceability, energy, key, loud
ness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time_signature, track_genre
 Schema: _c0, track_id, artists, album_name, track_name, popularity, duration_ms, explicit, danceability, energy, key, l
oudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time_signature, track_genre
Expected: _c0 but found:
CSV file: file:///home/ardent/Downloads/archive/spotify.csv
+---+--------------------+--------------------+--------------------+--------------------+----------+-----------+--------
------------+------+---+--------+----+-----------+------------+----------------+--------+-------+-------+--------------
+-----------+
|_c0|            track_id|             artists|          album_name|          track_name|popularity|duration_ms|explicit
|danceability|energy|key|loudness|mode|speechiness|acousticness|instrumentalness|liveness|valence|  tempo|time_signature
|track_genre|
+---+--------------------+--------------------+--------------------+--------------------+----------+-----------+--------
------------+------+---+--------+----+-----------+------------+----------------+--------+-------+-------+--------------
+-----------+
|  0|5SuOikwiRyPMVoIQD...|         Gen Hoshino|              Comedy|              Comedy|        73|     230666|   False
|       0.676| 0.461|  1|  -6.746|   0|      0.143|      0.0322|        1.01e-06|   0.358|  0.715| 87.917|             4
|   acoustic|
|  1|4qPNDBW1i3p13qLCt...|        Ben Woodward|    Ghost (Acoustic)|    Ghost - Acoustic|        55|     149610|   False
|        0.42| 0.166|  1| -17.235|   1|     0.0763|       0.924|        5.56e-06|   0.101|  0.267| 77.489|             4
|   acoustic|
|  2|1iJBSr7s7jYXzM8EG...|Ingrid Michaelson...|      To Begin Again|      To Begin Again|        57|     210826|   False
```

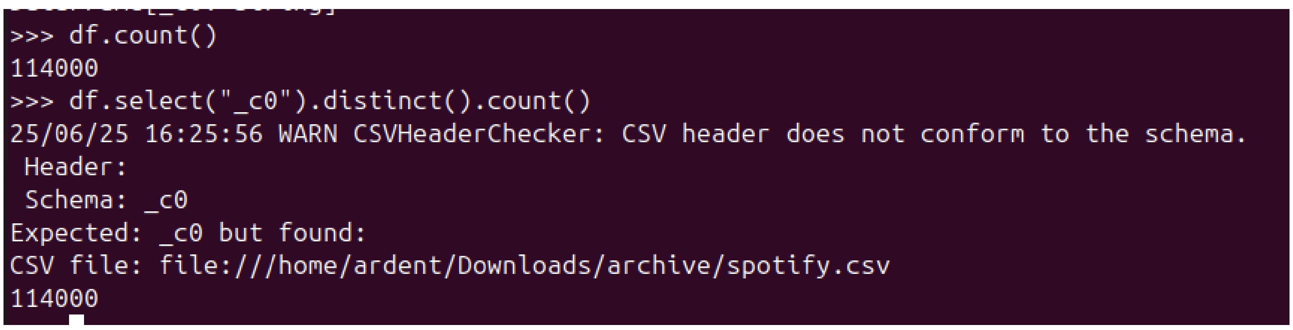
Show only prints the top 20 rows and truncates column that are greater then certain length. You can customize it using following code:
df.show(5, truncate=False)



```
>>> df.show(5, truncate=False)
25/06/25 16:24:12 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header: , track_id, artists, album_name, track_name, popularity, duration_ms, explicit, danceability, energy, key, loud
ness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time_signature, track_genre
 Schema: _c0, track_id, artists, album_name, track_name, popularity, duration_ms, explicit, danceability, energy, key, l
oudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time_signature, track_genre
Expected: _c0 but found:
CSV file: file:///home/ardent/Downloads/archive/spotify.csv
+---+----------------------+----------------------+------------------------------------------------------+------------
-----------+----------+-----------+--------+------------+------+---+--------+----+-----------+------------+-----------
----+--------+-------+-------+--------------+-----------+
|_c0|track_id              |artists               |album_name                                            |track_name
           |popularity|duration_ms|explicit|danceability|energy|key|loudness|mode|speechiness|acousticness|instrumental
ness|liveness|valence|tempo  |time_signature|track_genre|
+---+----------------------+----------------------+------------------------------------------------------+------------
-----------+----------+-----------+--------+------------+------+---+--------+----+-----------+------------+-----------
----+--------+-------+-------+--------------+-----------+
|0  |5SuOikwiRyPMVoIQDJUgSV|Gen Hoshino           |Comedy                                                |Comedy
           |73        |230666     |False   |0.676       |0.461 |1  |-6.746  |0   |0.143      |0.0322      |1.01e-06
    |0.358   |0.715  |87.917 |4             |acoustic   |
|1  |4qPNDBW1i3p13qLCt0Ki3A|Ben Woodward          |Ghost (Acoustic)                                      |Ghost - Acoust
ic         |55        |149610     |False   |0.42        |0.166 |1  |-17.235 |1   |0.0763     |0.924       |5.56e-06
    |0.101   |0.267  |77.489 |4             |acoustic   |
|2  |1iJBSr7s7jYXzM8EGcbK5b|Ingrid Michaelson;ZAYN|To Begin Again                                        |To Begin Again
           |57        |210826     |False   |0.438       |0.359 |0  |-9.734  |1   |0.0557     |0.21        |0.0
    |0.117   |0.12   |76.332 |4             |acoustic   |
```

By observation we can see that first column is integer and it is incremental. This could be assigned as an id but first we need to verify that it is distinct

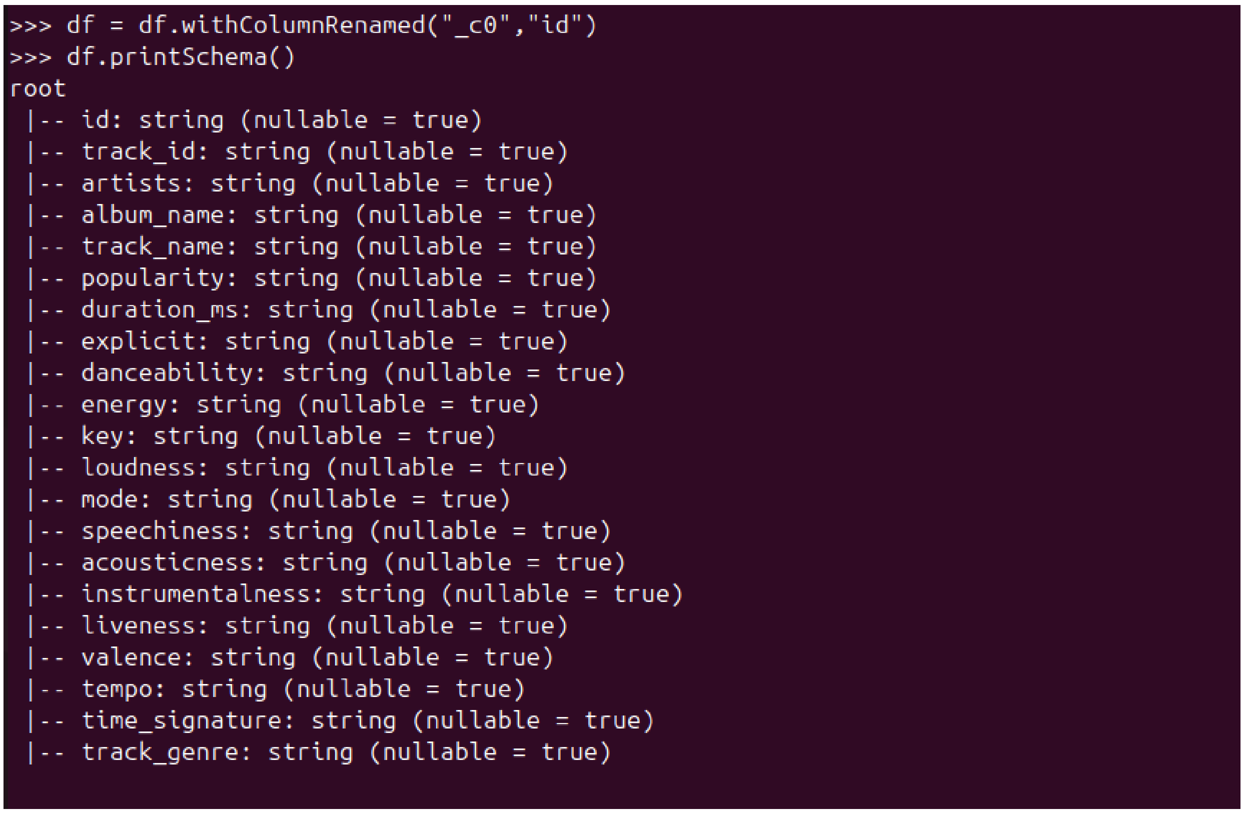
5. Get total rows using count. And the distinct row in first column by first selecting that column and getting the distinct values on it and then counting the values



```
>>> df.count()
114000
>>> df.select("_c0").distinct().count()
25/06/25 16:25:56 WARN CSVHeaderChecker: CSV header does not conform to the schema.
 Header:
 Schema: _c0
Expected: _c0 but found:
CSV file: file:///home/ardent/Downloads/archive/spotify.csv
114000
```

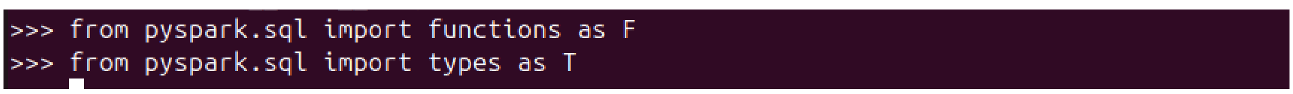
We can now verify that the first column has unique values and it can be set as id

6. Rename column using withColumnRenamed

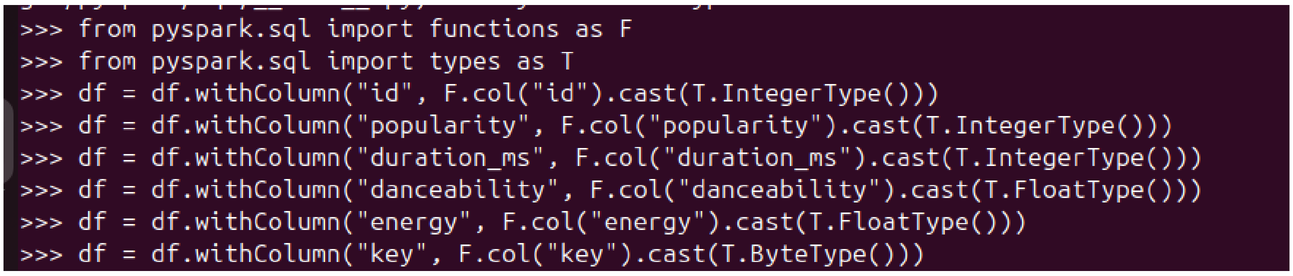


```
>>> df = df.withColumnRenamed("_c0","id")
>>> df.printSchema()
root
 |-- id: string (nullable = true)
 |-- track_id: string (nullable = true)
 |-- artists: string (nullable = true)
 |-- album_name: string (nullable = true)
 |-- track_name: string (nullable = true)
 |-- popularity: string (nullable = true)
 |-- duration_ms: string (nullable = true)
 |-- explicit: string (nullable = true)
 |-- danceability: string (nullable = true)
 |-- energy: string (nullable = true)
 |-- key: string (nullable = true)
 |-- loudness: string (nullable = true)
 |-- mode: string (nullable = true)
 |-- speechiness: string (nullable = true)
 |-- acousticness: string (nullable = true)
 |-- instrumentalness: string (nullable = true)
 |-- liveness: string (nullable = true)
 |-- valence: string (nullable = true)
 |-- tempo: string (nullable = true)
 |-- time_signature: string (nullable = true)
 |-- track_genre: string (nullable = true)
```

7. Cast columns to their proper type using cast we have to import the cast function as well the types we want to cast it to

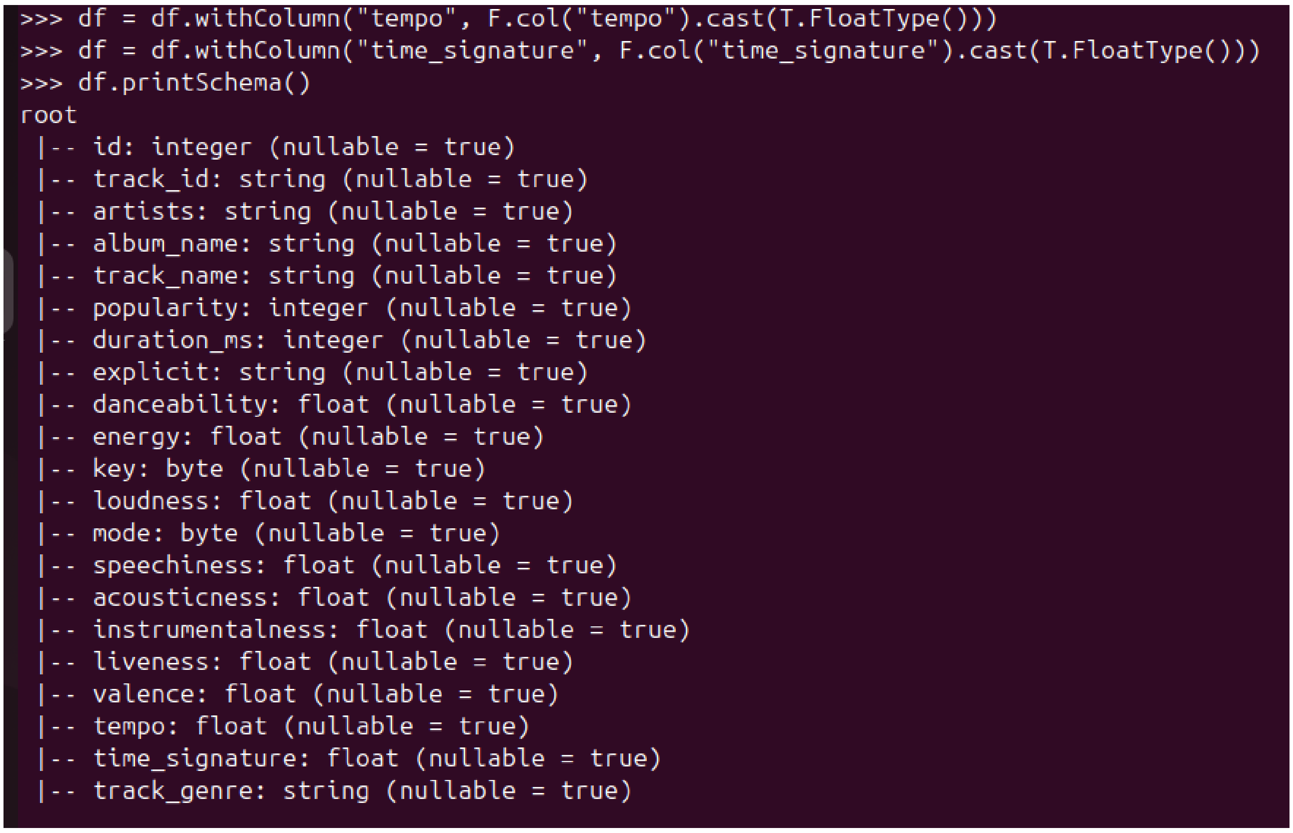


```
>>> from pyspark.sql import functions as F
>>> from pyspark.sql import types as T
```



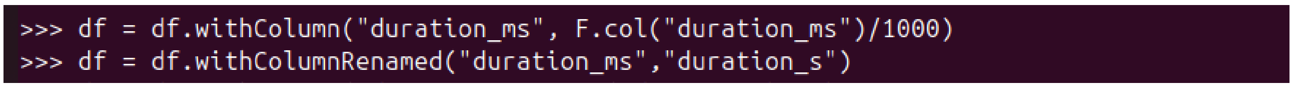
```
>>> from pyspark.sql import functions as F
>>> from pyspark.sql import types as T
>>> df = df.withColumn("id", F.col("id").cast(T.IntegerType()))
>>> df = df.withColumn("popularity", F.col("popularity").cast(T.IntegerType()))
>>> df = df.withColumn("duration_ms", F.col("duration_ms").cast(T.IntegerType()))
>>> df = df.withColumn("danceability", F.col("danceability").cast(T.FloatType()))
>>> df = df.withColumn("energy", F.col("energy").cast(T.FloatType()))
>>> df = df.withColumn("key", F.col("key").cast(T.ByteType()))
```

Do it for all columns that have been mistyped. Some of these might have been mistyped verify yourself that it has been done correctly.

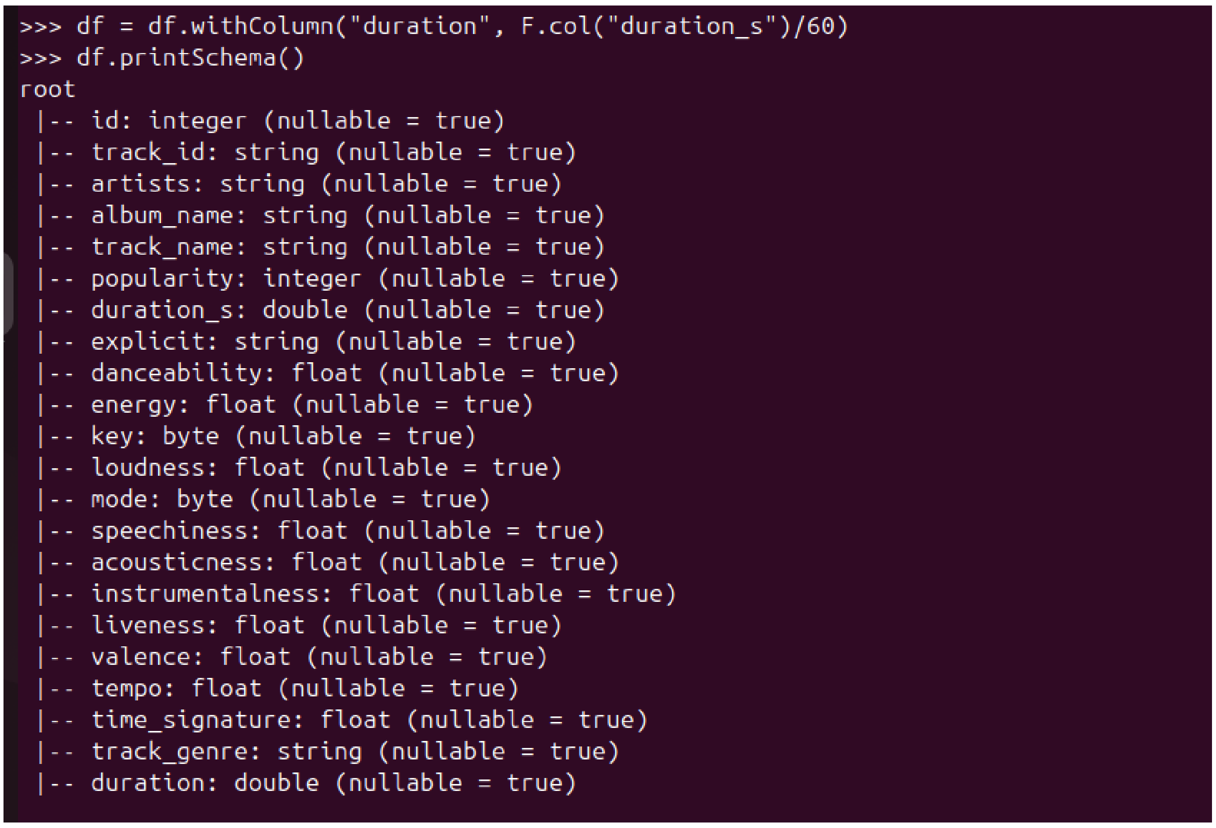


```
>>> df = df.withColumn("tempo", F.col("tempo").cast(T.FloatType()))
>>> df = df.withColumn("time_signature", F.col("time_signature").cast(T.FloatType()))
>>> df.printSchema()
root
 |-- id: integer (nullable = true)
 |-- track_id: string (nullable = true)
 |-- artists: string (nullable = true)
 |-- album_name: string (nullable = true)
 |-- track_name: string (nullable = true)
 |-- popularity: integer (nullable = true)
 |-- duration_ms: integer (nullable = true)
 |-- explicit: string (nullable = true)
 |-- danceability: float (nullable = true)
 |-- energy: float (nullable = true)
 |-- key: byte (nullable = true)
 |-- loudness: float (nullable = true)
 |-- mode: byte (nullable = true)
 |-- speechiness: float (nullable = true)
 |-- acousticness: float (nullable = true)
 |-- instrumentalness: float (nullable = true)
 |-- liveness: float (nullable = true)
 |-- valence: float (nullable = true)
 |-- tempo: float (nullable = true)
 |-- time_signature: float (nullable = true)
 |-- track_genre: string (nullable = true)
```

8. Update the duration_ms column to store the duration in second: rename the column as well, also create a new column duration to store duration in minutes



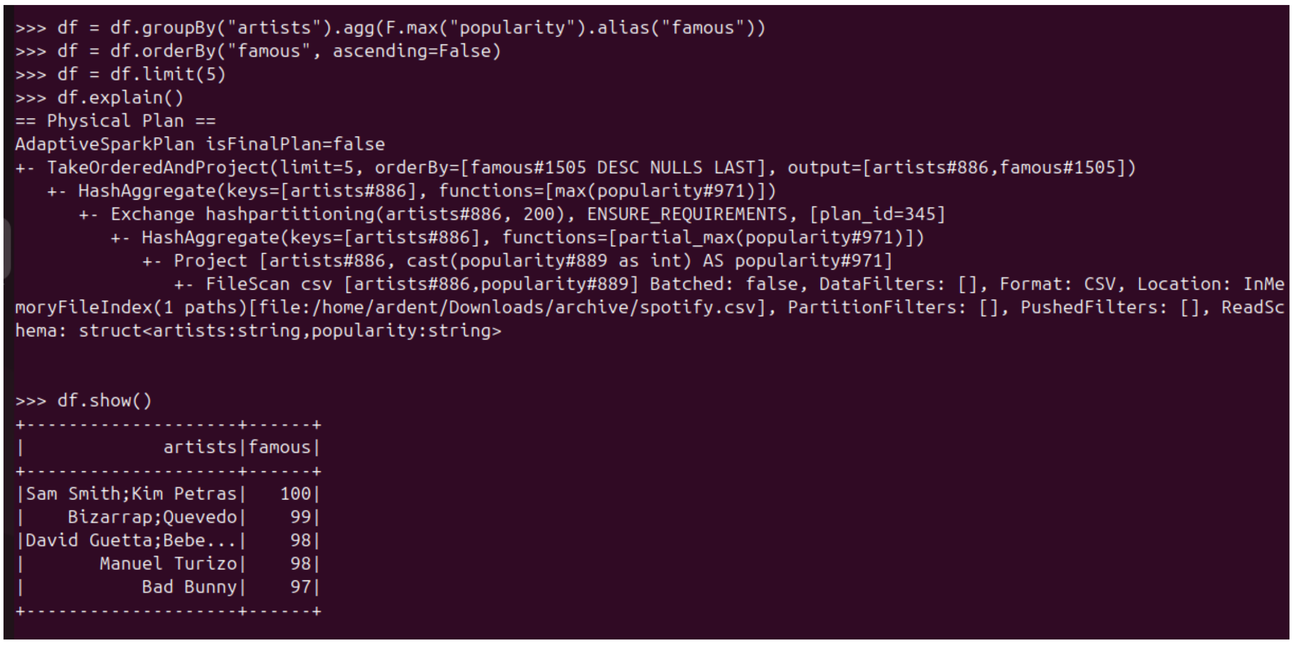
```
>>> df = df.withColumn("duration_ms", F.col("duration_ms")/1000)
>>> df = df.withColumnRenamed("duration_ms","duration_s")
```



```
>>> df = df.withColumn("duration", F.col("duration_s")/60)
>>> df.printSchema()
root
 |-- id: integer (nullable = true)
 |-- track_id: string (nullable = true)
 |-- artists: string (nullable = true)
 |-- album_name: string (nullable = true)
 |-- track_name: string (nullable = true)
 |-- popularity: integer (nullable = true)
 |-- duration_s: double (nullable = true)
 |-- explicit: string (nullable = true)
 |-- danceability: float (nullable = true)
 |-- energy: float (nullable = true)
 |-- key: byte (nullable = true)
 |-- loudness: float (nullable = true)
 |-- mode: byte (nullable = true)
 |-- speechiness: float (nullable = true)
 |-- acousticness: float (nullable = true)
 |-- instrumentalness: float (nullable = true)
 |-- liveness: float (nullable = true)
 |-- valence: float (nullable = true)
 |-- tempo: float (nullable = true)
 |-- time_signature: float (nullable = true)
 |-- track_genre: string (nullable = true)
 |-- duration: double (nullable = true)
```

Notice the creation of new column duration. Also the datatype of these columns have been automatically updated by pyspark

9. Get the top 5 artists based on their popularity



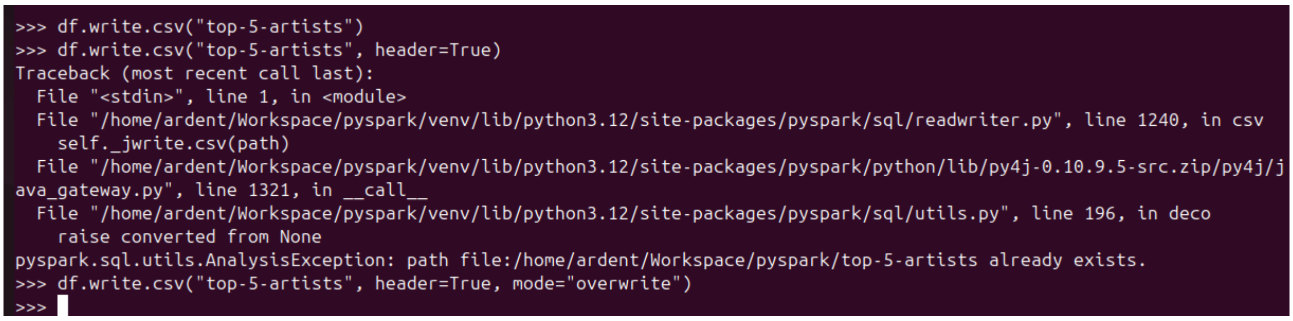
```
>>> df = df.groupBy("artists").agg(F.max("popularity").alias("famous"))
>>> df = df.orderBy("famous", ascending=False)
>>> df = df.limit(5)
>>> df.explain()
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- TakeOrderedAndProject(limit=5, orderBy=[famous#1505 DESC NULLS LAST], output=[artists#886,famous#1505])
   +- HashAggregate(keys=[artists#886], functions=[max(popularity#971)])
      +- Exchange hashpartitioning(artists#886, 200), ENSURE_REQUIREMENTS, [plan_id=345]
         +- HashAggregate(keys=[artists#886], functions=[partial_max(popularity#971)])
            +- Project [artists#886, cast(popularity#889 as int) AS popularity#971]
               +- FileScan csv [artists#886,popularity#889] Batched: false, DataFilters: [], Format: CSV, Location: InMe
moryFileIndex(1 paths)[file:/home/ardent/Downloads/archive/spotify.csv], PartitionFilters: [], PushedFilters: [], ReadSc
hema: struct<artists:string,popularity:string>


>>> df.show()
+--------------------+------+
|             artists|famous|
+--------------------+------+
|Sam Smith;Kim Petras|   100|
|    Bizarrap;Quevedo|    99|
|David Guetta;Bebe...|    98|
|       Manuel Turizo|    98|
|           Bad Bunny|    97|
+--------------------+------+
```

9.1 First we group the dataframe by artists and then calculate the max popularity for each artist and save it as new column famous using alias

9.2 We then order the resulting dataframe in descending order based on the new column famous, this will give us the highest ranked artist in the top
9.3 Now we limit the dataframe to first 5 rows
9.4 explain prints the current query plan designed by the spark engine. We read from the bottom and the most nested first. Notice how the plan includes everything from reading the file to all actions we have performed till now. This is an important concept to understand. In pyspark we have actions and transformations. Transformation only generates plans but action actually perform those. This is essential in optimizing the spark code and understanding how pyspark implements lazy-evaluation

10 Save the content to a csv file



```
>>> df.write.csv("top-5-artists")
>>> df.write.csv("top-5-artists", header=True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/ardent/Workspace/pyspark/venv/lib/python3.12/site-packages/pyspark/sql/readwriter.py", line 1240, in csv
    self._jwrite.csv(path)
  File "/home/ardent/Workspace/pyspark/venv/lib/python3.12/site-packages/pyspark/python/lib/py4j-0.10.9.5-src.zip/py4j/java_gateway.py", line 1321, in __call__
  File "/home/ardent/Workspace/pyspark/venv/lib/python3.12/site-packages/pyspark/sql/utils.py", line 196, in deco
    raise converted from None
pyspark.sql.utils.AnalysisException: path file:/home/ardent/Workspace/pyspark/top-5-artists already exists.
>>> df.write.csv("top-5-artists", header=True, mode="overwrite")
>>>
```

Intially write to csv file using df.write.csv(“top-5-artists”). Open the file and you will notice that headers are missing, this is csv specific logic but header can be added setting header parameter to True. However, notice how when you try to write the same file again, it will throw an error saying the file already exists. Now we use the mode parameter and set it to overwrite which will then overwrite the existing file. Header parameter is specific to csv but mode can be used for all types of file.

11. Play around with the new command you have learned.

11.1 Notice how the artist column sometime has multiple artists separated by ; using the **split** function to convert it to an array and **explode** function to create a new row out of it
11.2 Determine the relation of popularity with other parameters like loudness, energy, liveness and so on using the group by function. Group by as well as agg function can take multiple columns as input. Play around with it and figure it out. GroupBy is one of the most essential tool in data engineering and analysis
11.3 Cast to different types, maybe try with invalid types. People learn by making mistakes so do not hesitate to do that.
11.4 Explore other dataframe functions like **dropDuplicates**,**dropna**,**filter** and so on
11.5 Refer to pyspark documentation for further information:

Read/Write: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/io.html
DataFrame: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/dataframe.html
Types: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/data_types.html
Functions: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/functions.html

Now that we have learned the basics of pyspark with a small dataset. Let us work on something bigger. Download and extract the datasets we will be working on from here: https://www.kaggle.com/datasets/yamaerenay/spotify-dataset-19212020-600k-tracks?resource=download

You will find three different files:
1. artists.csv
This has 5 columns: id, followers, genres, name, popularity

2. dict_artists.json
The ids of artists recommended for fans of a artist. Each recommendation is sorted in descending order (The first one is the most favorite one). The number of recommendations are limited to 20.

3. tracks.csv
It has total of 20 columns: id, name, popularity,duration_ms, explicit, artists, id_artists, release_date, danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time_signature

You can also find the column names and their meta data in the above link.

Perform the following tasks:

1. Read three of these files separately into a separate data frame and cast them into proper types Initially you will have issues reading the dict_artists.json file in pyspark as it does not have default architecture supported by pyspark. Use the code provided for conversion in github to create a new file that works as expected.

2. Flatten the dict_artists.json data to hold single record instead of array of recommended artists. For example, if an artists has 10 other recommended artists linked to it. We should have a total of 10 rows (9 new) added with the mapping being one to one. (Hint: use explode function)

3. Get the name of the artists from artists.csv data (Hint: perform join between two dataframes two times for each column, first to get the artist name, then to get the related artist name)

4. Aggregate the artists and their recommended artists so that data is now in the format similar to before. However, now they will have their name instead of id associated with them

5. List out the recommended artists for nepali band ‘Nepathya’ .

6. Get the recommended artists for your favorite artist. See if the recommendation matches your liking.