Large Practical Assignment

~ *Big Data*

By

~ Julia Rosa Martínez Redondo

Context:

In this first task, we were guided to use a template in which

Task 1 – PySpark Implementation of Recommendation System:

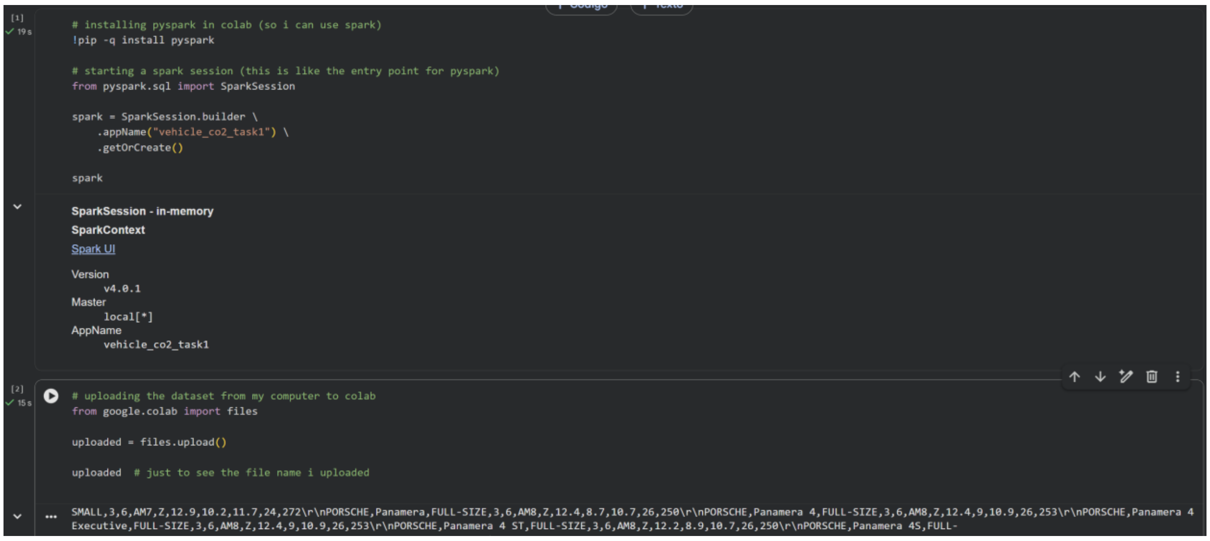
The objective of this task is to design and implement a recommendation system using PySpark based on the Vehicle CO2 Emissions dataset. The dataset contains information about different vehicle models, including technical characteristics such as engine size, fuel type, vehicle class and CO2 emissions.

The aim of the analysis is to organise and analyse the dataset, perform data cleaning and exploratory analysis, and finally build a recommendation system that suggests similar vehicles based on their characteristics. The implementation follows the data analysis workflow covered in the Big Data module, using Apache Spark to handle and process the data.

1º) PYSPARK & DATASET IMPORTATION

The Vehicle CO2 Emissions dataset was downloaded from Kaggle. First seen it contains 7,386 records, where each row represents a specific vehicle configuration. Looking at the attributes, this dataset includes categorical attributes (such as make, model, fuel type and vehicle class) and numerical attributes (such as engine size, fuel consumption and CO2 emissions).

The first step of the task was to initialise a SparkSession, which is the entry point for using PySpark.



```
# installing pyspark in colab (so i can use spark)
!pip -q install pyspark

# starting a spark session (this is like the entry point for pyspark)
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("vehicle_co2_task1") \
    .getOrCreate()

spark
```

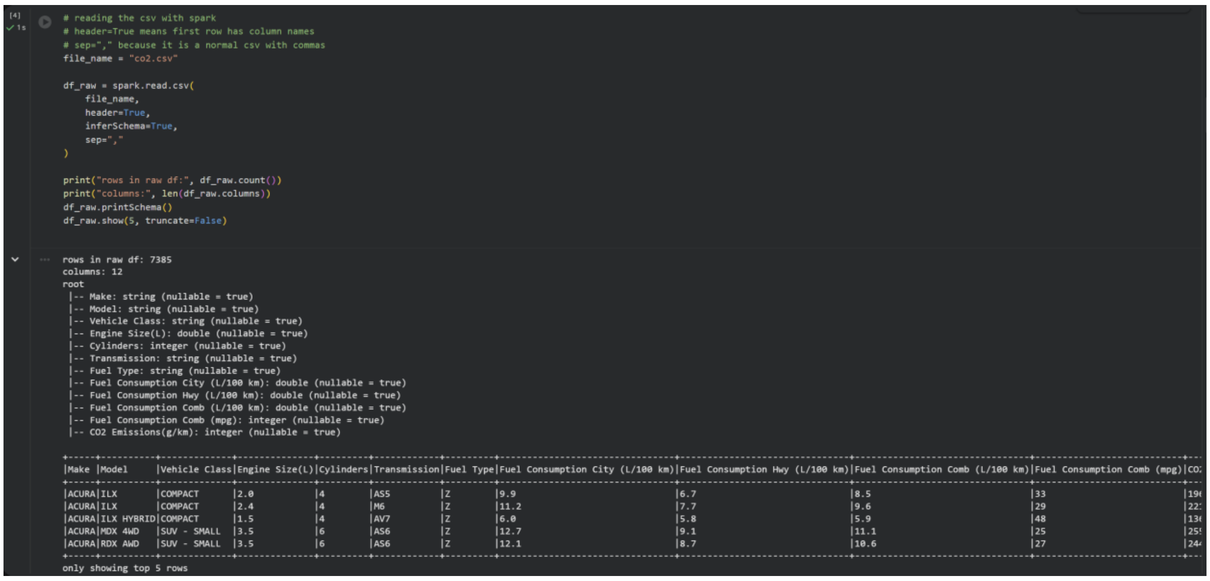
```
SparkSession - in-memory
SparkContext
Spark UI

Version
     v4.0.1
Master
     local[*]
AppName
     vehicle_co2_task1
```

```
# uploading the dataset from my computer to colab
from google.colab import files

uploaded = files.upload()

uploaded  # just to see the file name i uploaded
```

```
SMALL,3,6,AM7,Z,12.9,10.2,11.7,24,272\r\nPORSCHE,Panamera,FULL-SIZE,3,6,AM8,Z,12.4,8.7,10.7,26,250\r\nPORSCHE,Panamera 4,FULL-SIZE,3,6,AM8,Z,12.4,9,10.9,26,253\r\nPORSCHE,Panamera 4
Executive,FULL-SIZE,3,6,AM8,Z,12.4,9,10.9,26,253\r\nPORSCHE,Panamera 4 ST,FULL-SIZE,3,6,AM8,Z,12.2,8.9,10.7,26,250\r\nPORSCHE,Panamera 4S,FULL-
```

After that, the Vehicle CO2 Emissions dataset was loaded from a CSV file into a PySpark DataFrame.

Once the dataset was loaded, the structure of the data was inspected by checking the schema, column names and a small sample of rows. This step was necessary to ensure that the dataset was correctly parsed and that the data types were suitable for further analysis.

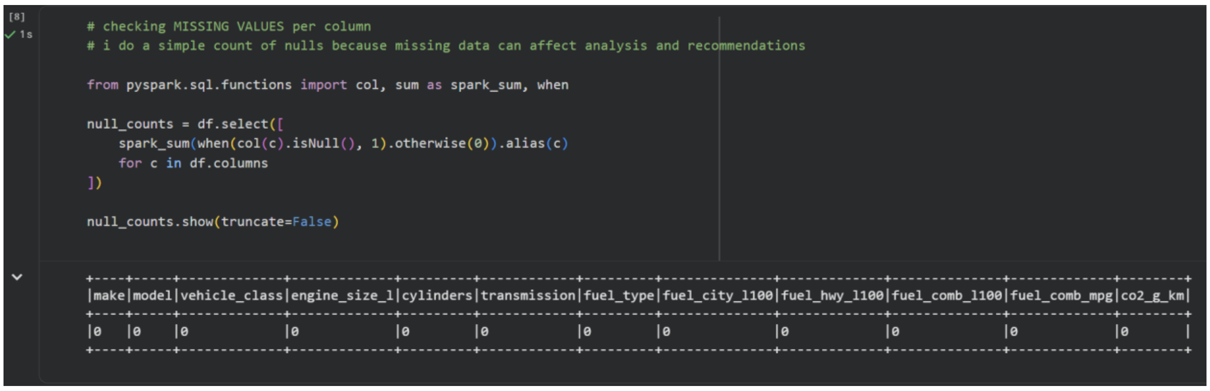


```
# reading the csv with spark
# header=True means first row has column names
# sep="," because it is a normal csv with commas
file_name = "co2.csv"

df_raw = spark.read.csv(
    file_name,
    header=True,
    inferSchema=True,
    sep=","
)

print("rows in raw df:", df_raw.count())
print("columns:", len(df_raw.columns))
df_raw.printSchema()
df_raw.show(5, truncate=False)
```

```
rows in raw df: 7385
columns: 12
root
 |-- Make: string (nullable = true)
 |-- Model: string (nullable = true)
 |-- Vehicle Class: string (nullable = true)
 |-- Engine Size(L): double (nullable = true)
 |-- Cylinders: integer (nullable = true)
 |-- Transmission: string (nullable = true)
 |-- Fuel Type: string (nullable = true)
 |-- Fuel Consumption City (L/100 km): double (nullable = true)
 |-- Fuel Consumption Hwy (L/100 km): double (nullable = true)
 |-- Fuel Consumption Comb (L/100 km): double (nullable = true)
 |-- Fuel Consumption Comb (mpg): integer (nullable = true)
 |-- CO2 Emissions(g/km): integer (nullable = true)

+-----+----------+-------------+--------------+---------+------------+---------+--------------------------------+-------------------------------+--------------------------------+---------------------------+---
|Make |Model     |Vehicle Class|Engine Size(L)|Cylinders|Transmission|Fuel Type|Fuel Consumption City (L/100 km)|Fuel Consumption Hwy (L/100 km)|Fuel Consumption Comb (L/100 km)|Fuel Consumption Comb (mpg)|CO
+-----+----------+-------------+--------------+---------+------------+---------+--------------------------------+-------------------------------+--------------------------------+---------------------------+---
|ACURA|ILX       |COMPACT      |2.0           |4        |AS5         |Z        |9.9                             |6.7                            |8.5                             |33                         |19
|ACURA|ILX       |COMPACT      |2.4           |4        |M6          |Z        |11.2                            |7.7                            |9.6                             |29                         |22
|ACURA|ILX HYBRID|COMPACT      |1.5           |4        |AV7         |Z        |6.0                             |5.8                            |5.9                             |48                         |13
|ACURA|MDX 4WD   |SUV - SMALL  |3.5           |6        |AS6         |Z        |12.7                            |9.1                            |11.1                            |25                         |25
|ACURA|RDX AWD   |SUV - SMALL  |3.5           |6        |AS6         |Z        |12.1                            |8.7                            |10.6                            |27                         |24
+-----+----------+-------------+--------------+---------+------------+---------+--------------------------------+-------------------------------+--------------------------------+---------------------------+---
only showing top 5 rows
```

2º) DATA CLEANNING & PREPROCESSING

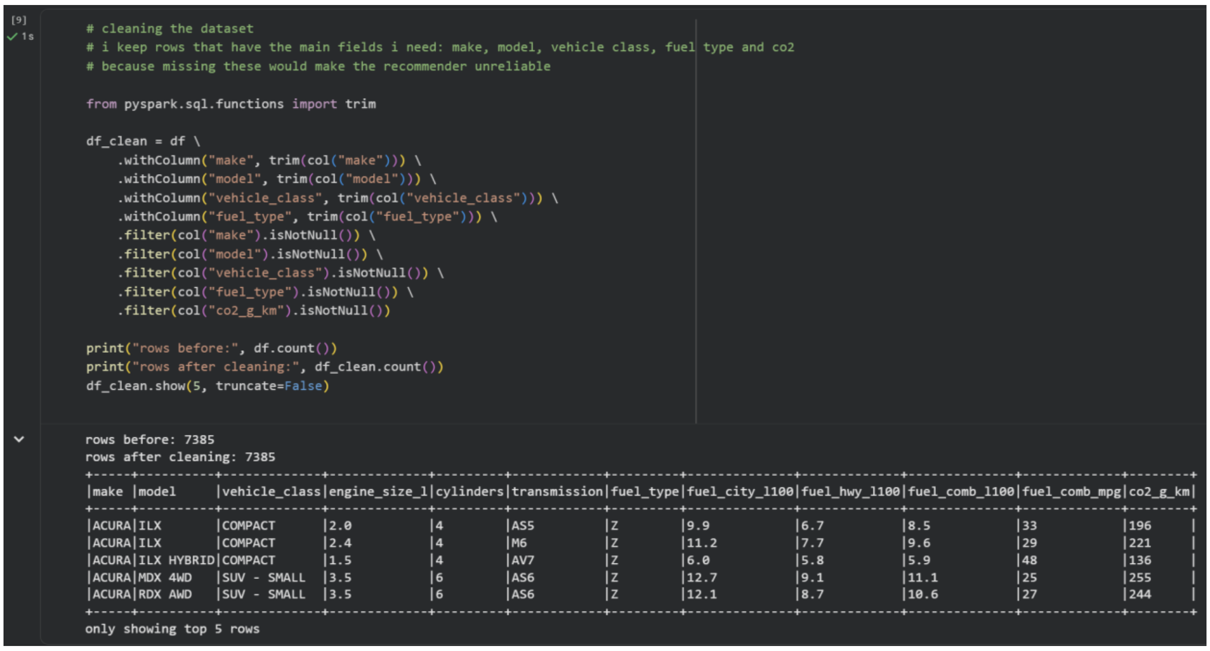
Before performing any analysis or building the recommendation system, data cleaning was carried out.



```
# checking MISSING VALUES per column
# i do a simple count of nulls because missing data can affect analysis and recommendations

from pyspark.sql.functions import col, sum as spark_sum, when

null_counts = df.select([
    spark_sum(when(col(c).isNull(), 1).otherwise(0)).alias(c)
    for c in df.columns
])

null_counts.show(truncate=False)
```

```
+----+-----+-------------+-------------+---------+------------+---------+--------------+-------------+--------------+-------------+--------+
|make|model|vehicle_class|engine_size_l|cylinders|transmission|fuel_type|fuel_city_l100|fuel_hwy_l100|fuel_comb_l100|fuel_comb_mpg|co2_g_km|
+----+-----+-------------+-------------+---------+------------+---------+--------------+-------------+--------------+-------------+--------+
|0   |0    |0            |0            |0        |0           |0        |0             |0            |0             |0            |0       |
+----+-----+-------------+-------------+---------+------------+---------+--------------+-------------+--------------+-------------+--------+
```

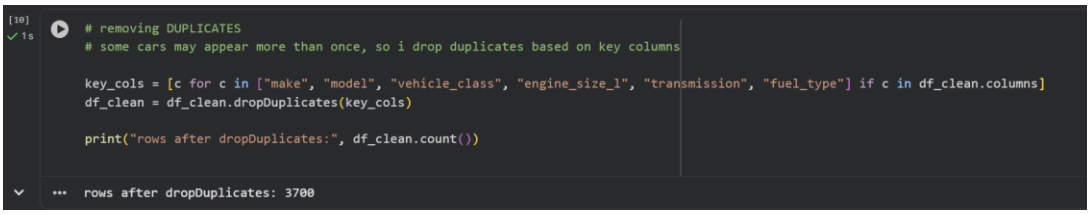
Rows with missing values in key attributes such as make, model, vehicle class, fuel type or CO2 emissions were removed. These attributes are essential for defining similarity between vehicles, and incomplete records could negatively affect the quality of the recommendations.



```
# cleaning the dataset
# i keep rows that have the main fields i need: make, model, vehicle class, fuel type and co2
# because missing these would make the recommender unreliable

from pyspark.sql.functions import trim

df_clean = df \
    .withColumn("make", trim(col("make"))) \
    .withColumn("model", trim(col("model"))) \
    .withColumn("vehicle_class", trim(col("vehicle_class"))) \
    .withColumn("fuel_type", trim(col("fuel_type"))) \
    .filter(col("make").isNotNull()) \
    .filter(col("model").isNotNull()) \
    .filter(col("vehicle_class").isNotNull()) \
    .filter(col("fuel_type").isNotNull()) \
    .filter(col("co2_g_km").isNotNull())

print("rows before:", df.count())
print("rows after cleaning:", df_clean.count())
df_clean.show(5, truncate=False)
```

```
rows before: 7385
rows after cleaning: 7385
+-----+----------+-------------+-------------+---------+------------+---------+--------------+-------------+--------------+-------------+--------+
|make |model     |vehicle_class|engine_size_l|cylinders|transmission|fuel_type|fuel_city_l100|fuel_hwy_l100|fuel_comb_l100|fuel_comb_mpg|co2_g_km|
+-----+----------+-------------+-------------+---------+------------+---------+--------------+-------------+--------------+-------------+--------+
|ACURA|ILX       |COMPACT      |2.0          |4        |AS5         |Z        |9.9           |6.7          |8.5           |33           |196     |
|ACURA|ILX       |COMPACT      |2.4          |4        |M6          |Z        |11.2          |7.7          |9.6           |29           |221     |
|ACURA|ILX HYBRID|COMPACT      |1.5          |4        |AV7         |Z        |6.0           |5.8          |5.9           |48           |136     |
|ACURA|MDX 4WD   |SUV - SMALL  |3.5          |6        |AS6         |Z        |12.7          |9.1          |11.1          |25           |255     |
|ACURA|RDX AWD   |SUV - SMALL  |3.5          |6        |AS6         |Z        |12.1          |8.7          |10.6          |27           |244     |
+-----+----------+-------------+-------------+---------+------------+---------+--------------+-------------+--------------+-------------+--------+
only showing top 5 rows
```

In addition, column names were simplified to improve readability and duplicate vehicle entries were removed based on relevant vehicle attributes. These preprocessing steps helped ensure that the dataset was consistent and suitable for building a recommendation system.

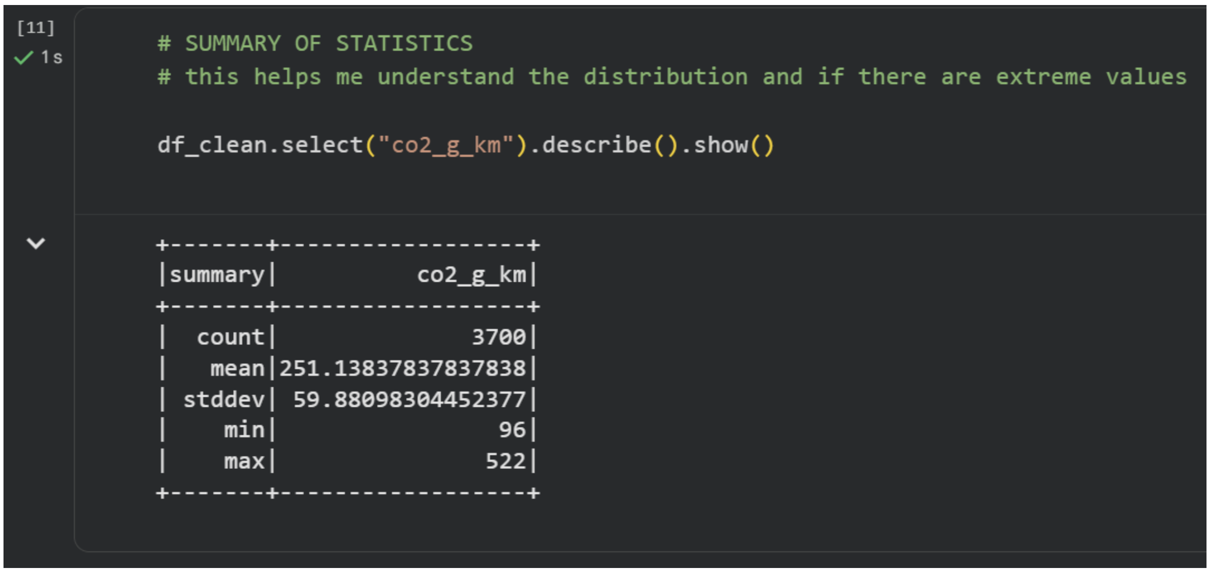


```
# removing DUPLICATES
# some cars may appear more than once, so i drop duplicates based on key columns

key_cols = [c for c in ["make", "model", "vehicle_class", "engine_size_l", "transmission", "fuel_type"] if c in df_clean.columns]
df_clean = df_clean.dropDuplicates(key_cols)

print("rows after dropDuplicates:", df_clean.count())
```

```
rows after dropDuplicates: 3700
```

3º) DATA ANALYSIS EXPLORATORY

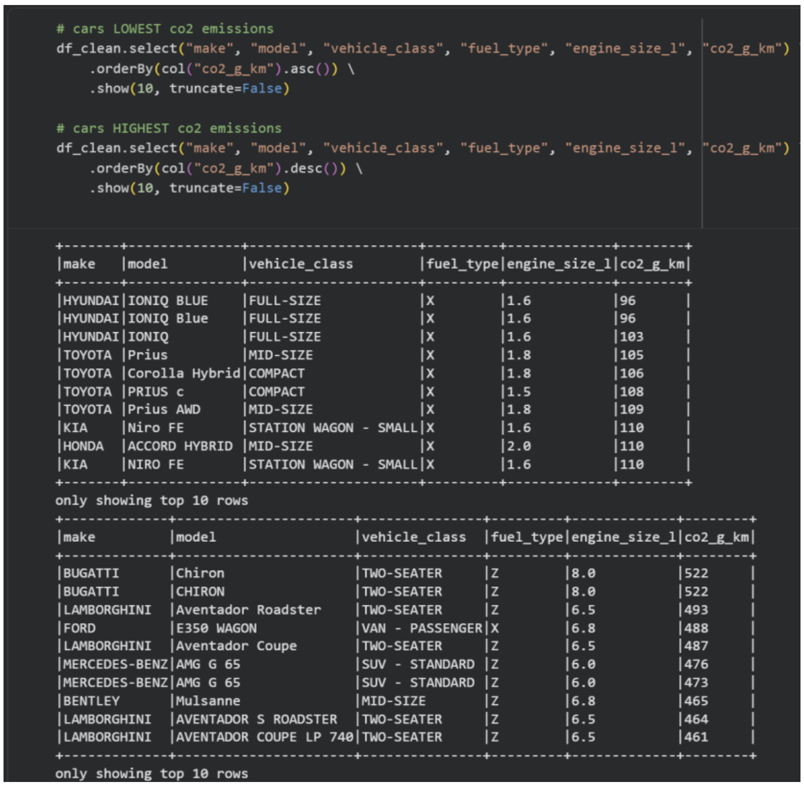
EDA done to gain insights into the dataset and understand patterns related to CO2 emissions. Summary statistics were used to analyse the distribution of CO2 emissions and identify extreme values.



```
[11]
✓ 1s
    # SUMMARY OF STATISTICS
    # this helps me understand the distribution and if there are extreme values

    df_clean.select("co2_g_km").describe().show()
```

```
+-------+------------------+
|summary|          co2_g_km|
+-------+------------------+
|  count|              3700|
|   mean|251.13837837837838|
| stddev| 59.88098304452377|
|    min|                96|
|    max|               522|
+-------+------------------+
```

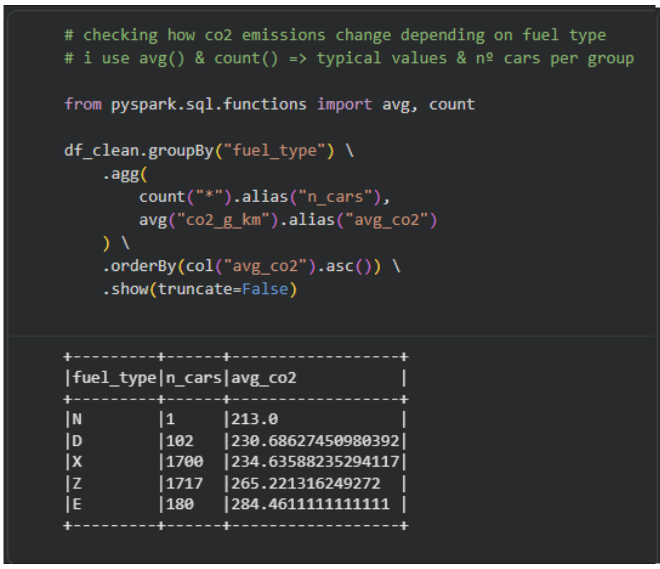
Summary statistics were used to analyse the distribution of CO2 emissions and identify extreme values.



```
# cars LOWEST co2 emissions
df_clean.select("make", "model", "vehicle_class", "fuel_type", "engine_size_l", "co2_g_km")
    .orderBy(col("co2_g_km").asc()) \
    .show(10, truncate=False)

# cars HIGHEST co2 emissions
df_clean.select("make", "model", "vehicle_class", "fuel_type", "engine_size_l", "co2_g_km")
    .orderBy(col("co2_g_km").desc()) \
    .show(10, truncate=False)
```

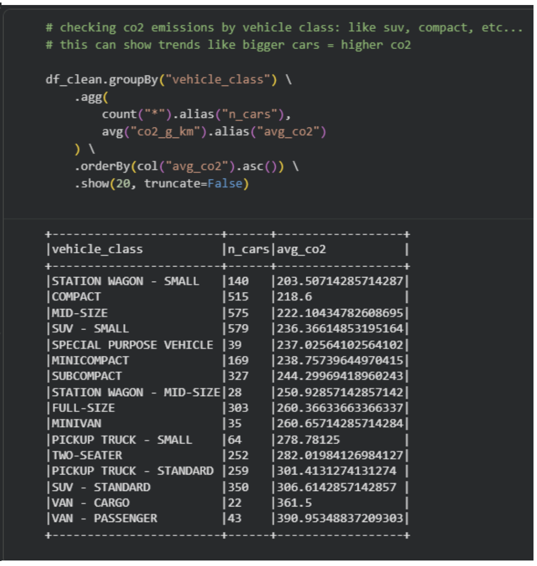
```
+-------+-------------+---------------------+---------+-------------+--------+
|make   |model        |vehicle_class        |fuel_type|engine_size_l|co2_g_km|
+-------+-------------+---------------------+---------+-------------+--------+
|HYUNDAI|IONIQ BLUE   |FULL-SIZE            |X        |1.6          |96      |
|HYUNDAI|IONIQ Blue   |FULL-SIZE            |X        |1.6          |96      |
|HYUNDAI|IONIQ        |FULL-SIZE            |X        |1.6          |103     |
|TOYOTA |Prius        |MID-SIZE             |X        |1.8          |105     |
|TOYOTA |Corolla Hybrid|COMPACT             |X        |1.8          |106     |
|TOYOTA |PRIUS c      |COMPACT              |X        |1.5          |108     |
|TOYOTA |Prius AWD    |MID-SIZE             |X        |1.8          |109     |
|KIA    |Niro FE      |STATION WAGON - SMALL|X        |1.6          |110     |
|HONDA  |ACCORD HYBRID|MID-SIZE             |X        |2.0          |110     |
|KIA    |NIRO FE      |STATION WAGON - SMALL|X        |1.6          |110     |
+-------+-------------+---------------------+---------+-------------+--------+
only showing top 10 rows
+-------------+-----------------------+-----------------+---------+-------------+--------+
|make         |model                  |vehicle_class    |fuel_type|engine_size_l|co2_g_km|
+-------------+-----------------------+-----------------+---------+-------------+--------+
|BUGATTI      |Chiron                 |TWO-SEATER       |Z        |8.0          |522     |
|BUGATTI      |CHIRON                 |TWO-SEATER       |Z        |8.0          |522     |
|LAMBORGHINI  |Aventador Roadster     |TWO-SEATER       |Z        |6.5          |493     |
|FORD         |E350 WAGON             |VAN - PASSENGER  |X        |6.8          |488     |
|LAMBORGHINI  |Aventador Coupe        |TWO-SEATER       |Z        |6.5          |487     |
|MERCEDES-BENZ|AMG G 65               |SUV - STANDARD   |Z        |6.0          |476     |
|MERCEDES-BENZ|AMG G 65               |SUV - STANDARD   |Z        |6.0          |473     |
|BENTLEY      |Mulsanne               |MID-SIZE         |Z        |6.8          |465     |
|LAMBORGHINI  |AVENTADOR S ROADSTER   |TWO-SEATER       |Z        |6.5          |464     |
|LAMBORGHINI  |AVENTADOR COUPE LP 740 |TWO-SEATER       |Z        |6.5          |461     |
+-------------+-----------------------+-----------------+---------+-------------+--------+
only showing top 10 rows
```

Further analysis was carried out by grouping vehicles by fuel type and vehicle class, and by examining the relationship between engine size and CO2 emissions. This analysis helps to understand how different vehicle characteristics influence CO2 emissions.



```
# checking how co2 emissions change depending on fuel type
# i use avg() & count() => typical values & nº cars per group

from pyspark.sql.functions import avg, count

df_clean.groupBy("fuel_type") \
    .agg(
        count("*").alias("n_cars"),
        avg("co2_g_km").alias("avg_co2")
    ) \
    .orderBy(col("avg_co2").asc()) \
    .show(truncate=False)
```

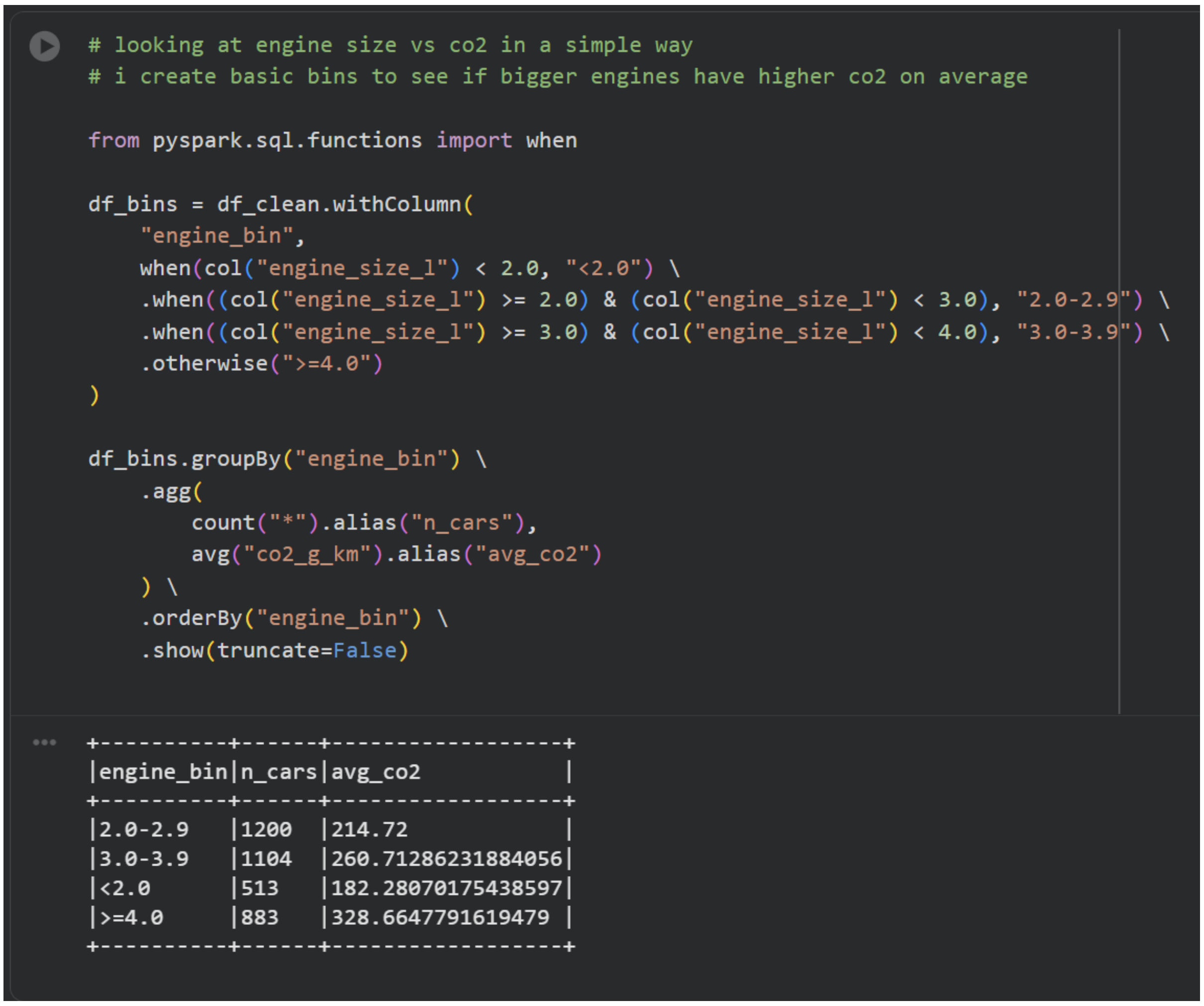
```
+---------+------+------------------+
|fuel_type|n_cars|avg_co2           |
+---------+------+------------------+
|N        |1     |213.0             |
|D        |102   |230.68627450980392|
|X        |1700  |234.63588235294117|
|Z        |1717  |265.221316249272  |
|E        |180   |284.4611111111111 |
+---------+------+------------------+
```



```
# checking co2 emissions by vehicle class: like suv, compact, etc...
# this can show trends like bigger cars = higher co2

df_clean.groupBy("vehicle_class") \
    .agg(
        count("*").alias("n_cars"),
        avg("co2_g_km").alias("avg_co2")
    ) \
    .orderBy(col("avg_co2").asc()) \
    .show(20, truncate=False)
```

```
+------------------------+------+------------------+
|vehicle_class           |n_cars|avg_co2           |
+------------------------+------+------------------+
|STATION WAGON - SMALL   |140   |203.50714285714287|
|COMPACT                 |515   |218.6             |
|MID-SIZE                |575   |222.10434782608695|
|SUV - SMALL             |579   |236.36614853195164|
|SPECIAL PURPOSE VEHICLE |39    |237.02564102564102|
|MINICOMPACT             |169   |238.75739644970415|
|SUBCOMPACT              |327   |244.29969418960243|
|STATION WAGON - MID-SIZE|28    |250.92857142857142|
|FULL-SIZE               |303   |260.36633663366337|
|MINIVAN                 |35    |260.65714285714284|
|PICKUP TRUCK - SMALL    |64    |278.78125         |
|TWO-SEATER              |252   |282.01984126984127|
|PICKUP TRUCK - STANDARD |259   |301.4131274131274 |
|SUV - STANDARD          |350   |306.6142857142857 |
|VAN - CARGO             |22    |361.5             |
|VAN - PASSENGER         |43    |390.95348837209303|
+------------------------+------+------------------+
```

Finally, the relationship between engine size and CO2 emissions was analysed. The results indicate that as engine size increases, the average CO2 emissions also increase. This confirms an expected trend and highlights engine size as an important factor influencing CO2 emissions.



```
# looking at engine size vs co2 in a simple way
# i create basic bins to see if bigger engines have higher co2 on average

from pyspark.sql.functions import when

df_bins = df_clean.withColumn(
    "engine_bin",
    when(col("engine_size_l") < 2.0, "<2.0") \
    .when((col("engine_size_l") >= 2.0) & (col("engine_size_l") < 3.0), "2.0-2.9") \
    .when((col("engine_size_l") >= 3.0) & (col("engine_size_l") < 4.0), "3.0-3.9") \
    .otherwise(">=4.0")
)

df_bins.groupBy("engine_bin") \
    .agg(
        count("*").alias("n_cars"),
        avg("co2_g_km").alias("avg_co2")
    ) \
    .orderBy("engine_bin") \
    .show(truncate=False)
```

```
+----------+------+------------------+
|engine_bin|n_cars|avg_co2           |
+----------+------+------------------+
|2.0-2.9   |1200  |214.72            |
|3.0-3.9   |1104  |260.71286231884056|
|<2.0      |513   |182.28070175438597|
|>=4.0     |883   |328.6647791619479 |
+----------+------+------------------+
```

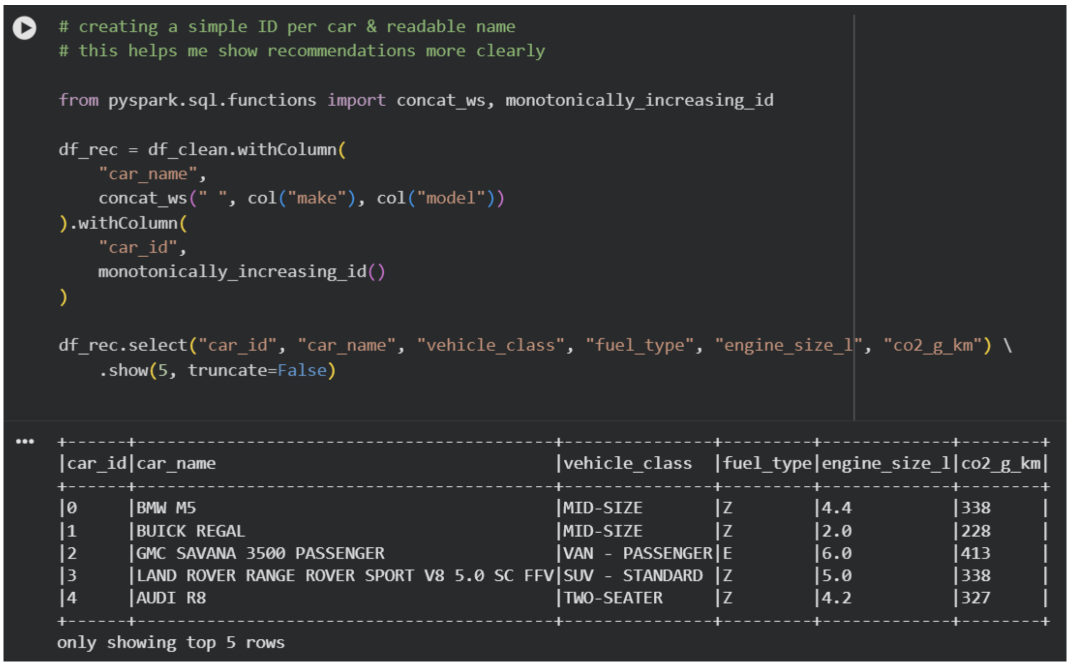
INTERPRETATION -> Overall, the exploratory analysis confirms that vehicle characteristics such as engine size, fuel type and vehicle class have a **strong influence** on CO2 emissions, which supports their *use as relevant features* in the recommendation system coming next

----

4º) RECOMENDATION SYSTEM DESIGNED CHOICE

The dataset does not contain users or ratings, only vehicle characteristics. For this reason, collaborative filtering approaches such as ALS were not suitable for this task.

Instead, a content-based recommendation system was implemented, where vehicles are recommended based on their similarity to other vehicles using their technical attributes.



```python
# creating a simple ID per car & readable name
# this helps me show recommendations more clearly

from pyspark.sql.functions import concat_ws, monotonically_increasing_id

df_rec = df_clean.withColumn(
    "car_name",
    concat_ws(" ", col("make"), col("model"))
).withColumn(
    "car_id",
    monotonically_increasing_id()
)

df_rec.select("car_id", "car_name", "vehicle_class", "fuel_type", "engine_size_l", "co2_g_km") \
    .show(5, truncate=False)
```

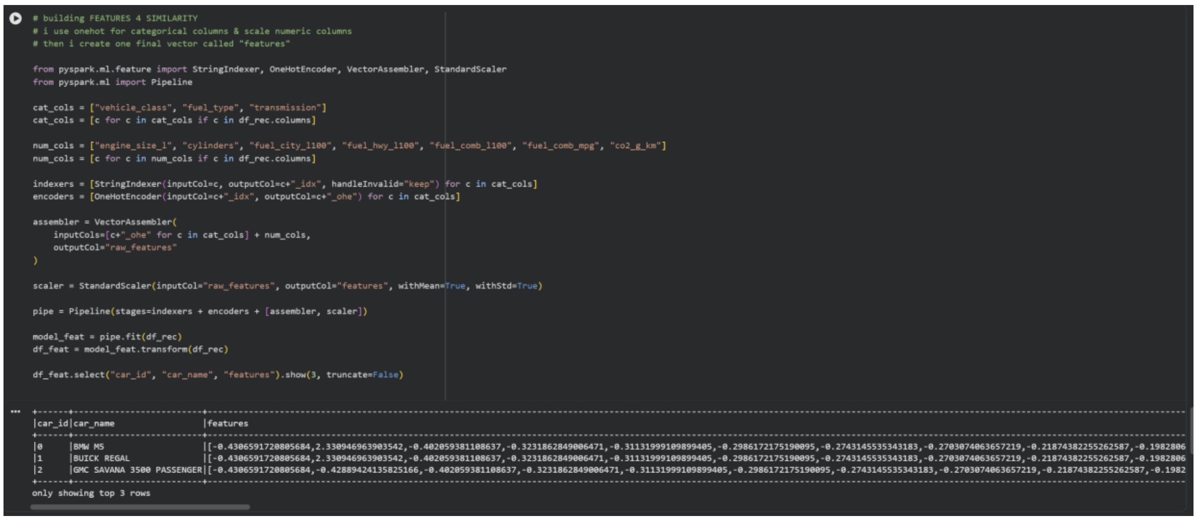
```
+------+------------------------------------------+---------------+---------+-------------+--------+
|car_id|car_name                                  |vehicle_class  |fuel_type|engine_size_l|co2_g_km|
+------+------------------------------------------+---------------+---------+-------------+--------+
|0     |BMW M5                                    |MID-SIZE       |Z        |4.4          |338     |
|1     |BUICK REGAL                               |MID-SIZE       |Z        |2.0          |228     |
|2     |GMC SAVANA 3500 PASSENGER                 |VAN - PASSENGER|E        |6.0          |413     |
|3     |LAND ROVER RANGE ROVER SPORT V8 5.0 SC FFV|SUV - STANDARD |Z        |5.0          |338     |
|4     |AUDI R8                                   |TWO-SEATER     |Z        |4.2          |327     |
+------+------------------------------------------+---------------+---------+-------------+--------+
only showing top 5 rows
```

This design choice is directly related to the structure of the dataset. Since there is no user interaction data, recommendations must rely only on vehicle features. Therefore, a content-based approach is the most appropriate solution for this problem.

5º) FEATURE ENGINEER

To compute similarity between vehicles, a feature vector was created for each car. Categorical attributes such as vehicle class, fuel type and transmission were converted into numerical format using StringIndexer and OneHotEncoder.

Numerical attributes such as engine size, fuel consumption and CO2 emissions were combined with the encoded categorical features. All features were assembled into a single vector and standardised to ensure that no single attribute dominated the similarity calculation.



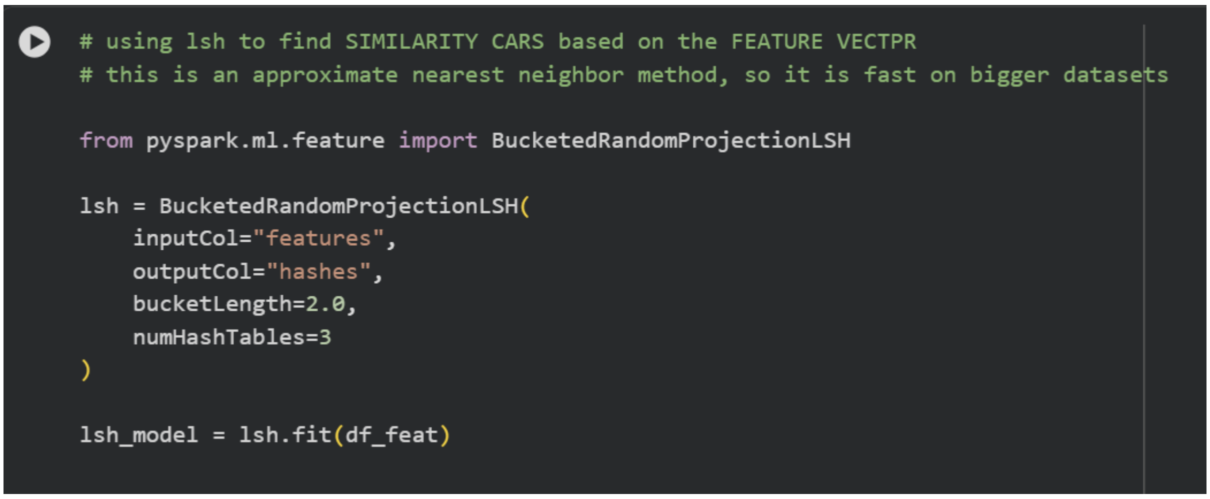
```python
# building FEATURES 4 SIMILARITY
# i use onehot for categorical columns & scale numeric columns
# then i create one final vector called "features"

from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler, StandardScaler
from pyspark.ml import Pipeline

cat_cols = ["vehicle_class", "fuel_type", "transmission"]
cat_cols = [c for c in cat_cols if c in df_rec.columns]

num_cols = ["engine_size_l", "cylinders", "fuel_city_l100", "fuel_hwy_l100", "fuel_comb_l100", "fuel_comb_mpg", "co2_g_km"]
num_cols = [c for c in num_cols if c in df_rec.columns]

indexers = [StringIndexer(inputCol=c, outputCol=c+"_idx", handleInvalid="keep") for c in cat_cols]
encoders = [OneHotEncoder(inputCol=c+"_idx", outputCol=c+"_ohe") for c in cat_cols]

assembler = VectorAssembler(
    inputCols=[c+"_ohe" for c in cat_cols] + num_cols,
    outputCol="raw_features"
)

scaler = StandardScaler(inputCol="raw_features", outputCol="features", withMean=True, withStd=True)

pipe = Pipeline(stages=indexers + encoders + [assembler, scaler])

model_feat = pipe.fit(df_rec)
df_feat = model_feat.transform(df_rec)

df_feat.select("car_id", "car_name", "features").show(3, truncate=False)
```

```
+------+-------------------------+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
|car_id|car_name                 |features
+------+-------------------------+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
|0     |BMW M5                   |[-0.4306591720805684,2.330946963903542,-0.402059381108637,-0.3231862849006471,-0.31131999109899405,-0.2986172175190095,-0.2743145535343183,-0.2703074063657219,-0.21874382255262587,-0.1982806
|1     |BUICK REGAL              |[-0.4306591720805684,2.330946963903542,-0.402059381108637,-0.3231862849006471,-0.31131999109899405,-0.2986172175190095,-0.2743145535343183,-0.2703074063657219,-0.21874382255262587,-0.1982806
|2     |GMC SAVANA 3500 PASSENGER|[-0.4306591720805684,-0.42889424135825166,-0.402059381108637,-0.3231862849006471,-0.31131999109899405,-0.2986172175190095,-0.2743145535343183,-0.2703074063657219,-0.21874382255262587,-0.1982
+------+-------------------------+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
only showing top 3 rows
```

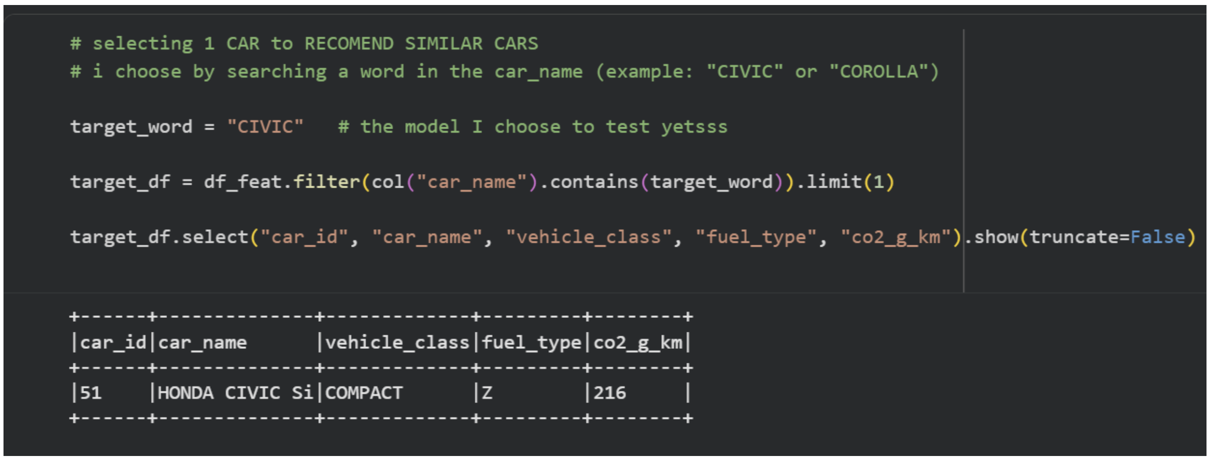
As a result, each vehicle is represented as a numerical vector that captures both its technical characteristics and emission-related information. This representation allows similarity between vehicles to be computed in a consistent and meaningful way and can be efficiently used by approximate nearest neighbour methods such as LSH.

6º) EVALUATING

The evaluation of the recommendation system was performed in a qualitative manner. Since this is a content-based recommendation system and the dataset does not include user feedback or ground truth labels, traditional accuracy metrics are not applicable. Instead, the recommended vehicles were analysed to check whether they had similar technical characteristics and comparable CO2 emission values.



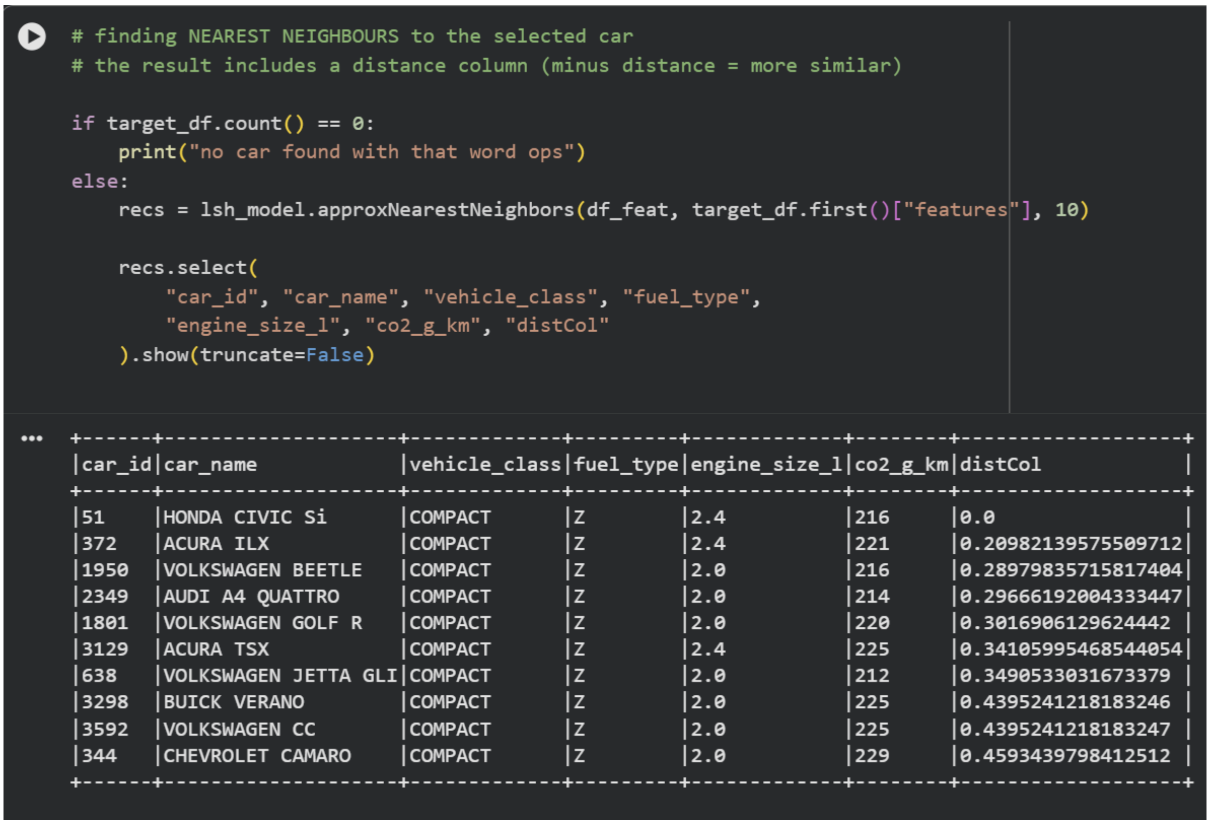
```
# using lsh to find SIMILARITY CARS based on the FEATURE VECTPR
# this is an approximate nearest neighbor method, so it is fast on bigger datasets

from pyspark.ml.feature import BucketedRandomProjectionLSH

lsh = BucketedRandomProjectionLSH(
    inputCol="features",
    outputCol="hashes",
    bucketLength=2.0,
    numHashTables=3
)

lsh_model = lsh.fit(df_feat)
```



```
# selecting 1 CAR to RECOMEND SIMILAR CARS
# i choose by searching a word in the car_name (example: "CIVIC" or "COROLLA")

target_word = "CIVIC"   # the model I choose to test yetsss

target_df = df_feat.filter(col("car_name").contains(target_word)).limit(1)

target_df.select("car_id", "car_name", "vehicle_class", "fuel_type", "co2_g_km").show(truncate=False)
```

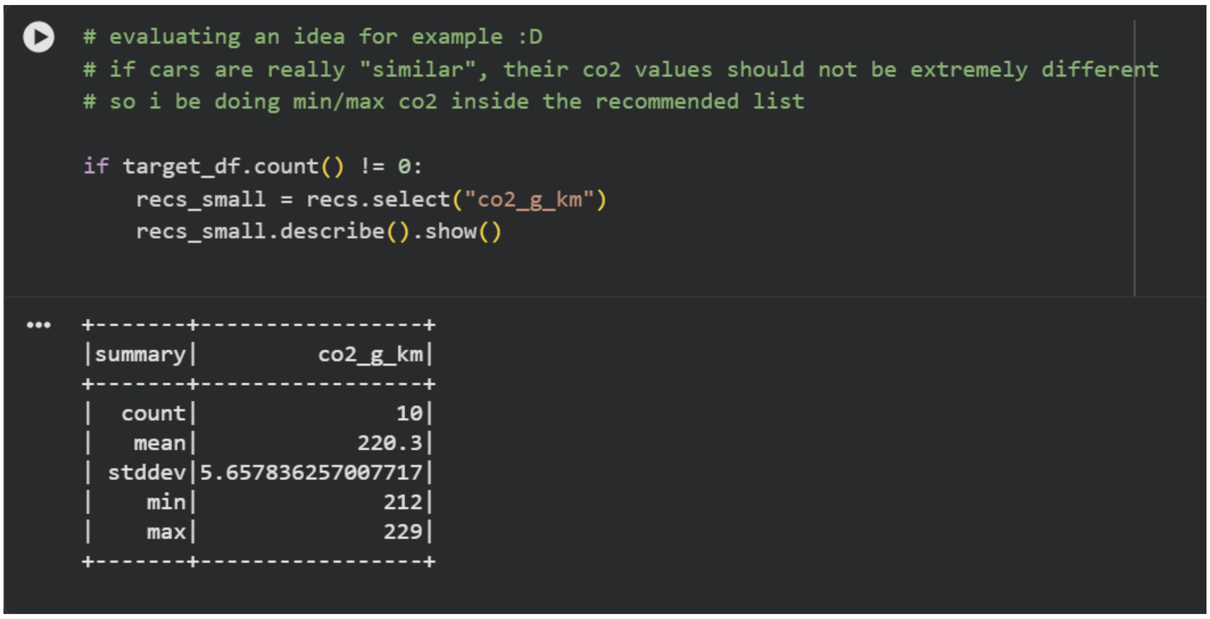
```
+------+--------------+-------------+---------+--------+
|car_id|car_name      |vehicle_class|fuel_type|co2_g_km|
+------+--------------+-------------+---------+--------+
|51    |HONDA CIVIC Si|COMPACT      |Z        |216     |
+------+--------------+-------------+---------+--------+
```

The results indicate that the recommended vehicles are coherent with the selected vehicle, especially in terms of CO2 emissions, engine size and vehicle class. The similarity distances are relatively small, which confirms that the system is grouping vehicles with comparable technical characteristics. This suggests that the recommendation system is able to capture meaningful similarities.



```python
# finding NEAREST NEIGHBOURS to the selected car
# the result includes a distance column (minus distance = more similar)

if target_df.count() == 0:
    print("no car found with that word ops")
else:
    recs = lsh_model.approxNearestNeighbors(df_feat, target_df.first()["features"], 10)

    recs.select(
        "car_id", "car_name", "vehicle_class", "fuel_type",
        "engine_size_l", "co2_g_km", "distCol"
    ).show(truncate=False)
```

```
+------+-------------------+-------------+---------+-------------+--------+-------------------+
|car_id|car_name           |vehicle_class|fuel_type|engine_size_l|co2_g_km|distCol            |
+------+-------------------+-------------+---------+-------------+--------+-------------------+
|51    |HONDA CIVIC Si     |COMPACT      |Z        |2.4          |216     |0.0                |
|372   |ACURA ILX          |COMPACT      |Z        |2.4          |221     |0.20982139575509712|
|1950  |VOLKSWAGEN BEETLE  |COMPACT      |Z        |2.0          |216     |0.28979835715817404|
|2349  |AUDI A4 QUATTRO    |COMPACT      |Z        |2.0          |214     |0.29666192004333447|
|1801  |VOLKSWAGEN GOLF R  |COMPACT      |Z        |2.0          |220     |0.301690612962444 2|
|3129  |ACURA TSX          |COMPACT      |Z        |2.4          |225     |0.34105995468544054|
|638   |VOLKSWAGEN JETTA GLI|COMPACT     |Z        |2.0          |212     |0.3490533031673379 |
|3298  |BUICK VERANO       |COMPACT      |Z        |2.0          |225     |0.4395241218183246 |
|3592  |VOLKSWAGEN CC      |COMPACT      |Z        |2.0          |225     |0.4395241218183247 |
|344   |CHEVROLET CAMARO   |COMPACT      |Z        |2.0          |229     |0.4593439798412512 |
+------+-------------------+-------------+---------+-------------+--------+-------------------+
```



```python
# evaluating an idea for example :D
# if cars are really "similar", their co2 values should not be extremely different
# so i be doing min/max co2 inside the recommended list

if target_df.count() != 0:
    recs_small = recs.select("co2_g_km")
    recs_small.describe().show()
```

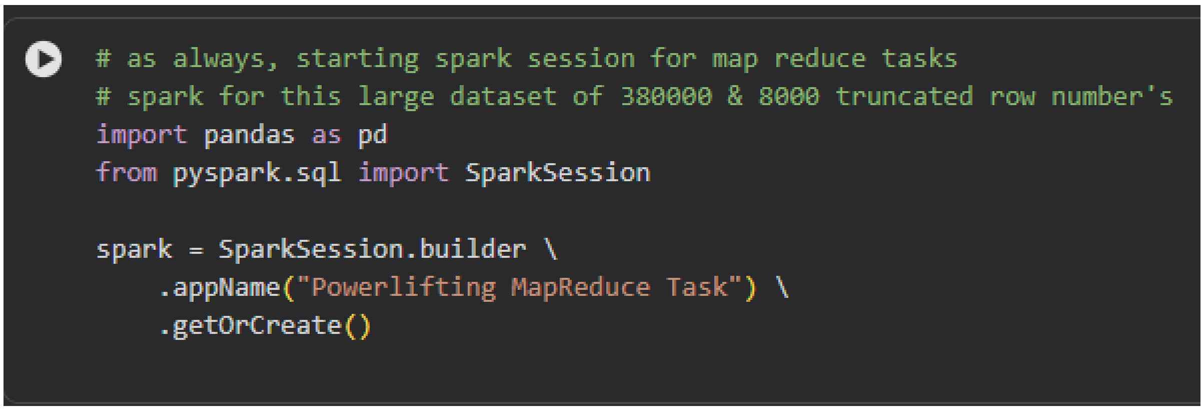
```
+-------+-----------------+
|summary|         co2_g_km|
+-------+-----------------+
|  count|               10|
|   mean|            220.3|
| stddev|5.657836257007717|
|    min|              212|
|    max|              229|
+-------+-----------------+
```

However, the system has limitations. Recommendations are based only on vehicle attributes and do not consider user preferences or driving behaviour. In addition, the quality of the recommendations depends on the selected features and their scaling.

Task 2 – Map reduce for Powerlifting dataset:

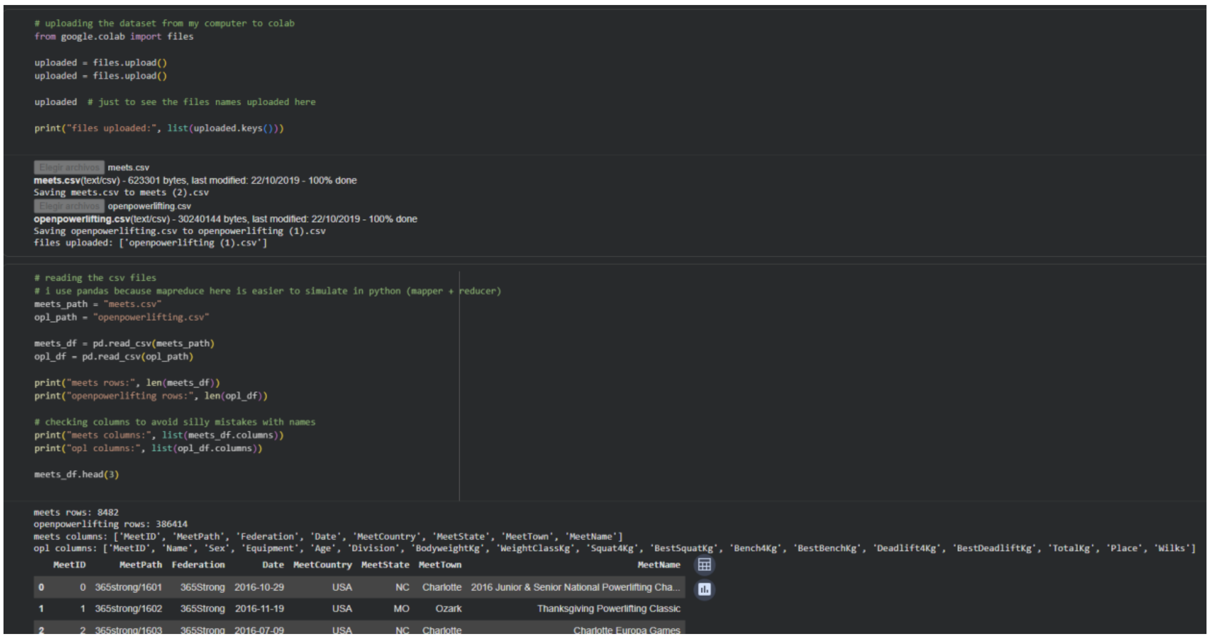
With the powerlifting dataset downloaded from kaggle, the goal is to apply a mapreduce-style approach in python to compute: 2.1 how many participants are in each competition (meet), 2.2 a clean list of competitions with name + country + year + participants, and 3.3 the total weight lifted by each participant across all competitions. because the dataset is big (openpowerlifting ~386k rows) the idea is to avoid doing everything with slow python loops and instead use the “map → shuffle/group → reduce” logic that scales better conceptually.

Spark



```python
# as always, starting spark session for map reduce tasks
# spark for this large dataset of 380000 & 8000 truncated row number's
import pandas as pd
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Powerlifting MapReduce Task") \
    .getOrCreate()
```

Importation & to dataframes



```python
# uploading the dataset from my computer to colab
from google.colab import files

uploaded = files.upload()
uploaded = files.upload()

uploaded  # just to see the files names uploaded here

print("files uploaded:", list(uploaded.keys()))
```

```
Elegir archivos meets.csv
meets.csv(text/csv) - 623301 bytes, last modified: 22/10/2019 - 100% done
Saving meets.csv to meets (2).csv
Elegir archivos openpowerlifting.csv
openpowerlifting.csv(text/csv) - 30240144 bytes, last modified: 22/10/2019 - 100% done
Saving openpowerlifting.csv to openpowerlifting (1).csv
files uploaded: ['openpowerlifting (1).csv']
```

```python
# reading the csv files
# i use pandas because mapreduce here is easier to simulate in python (mapper + reducer)
meets_path = "meets.csv"
opl_path = "openpowerlifting.csv"

meets_df = pd.read_csv(meets_path)
opl_df = pd.read_csv(opl_path)

print("meets rows:", len(meets_df))
print("openpowerlifting rows:", len(opl_df))

# checking columns to avoid silly mistakes with names
print("meets columns:", list(meets_df.columns))
print("opl columns:", list(opl_df.columns))

meets_df.head(3)
```

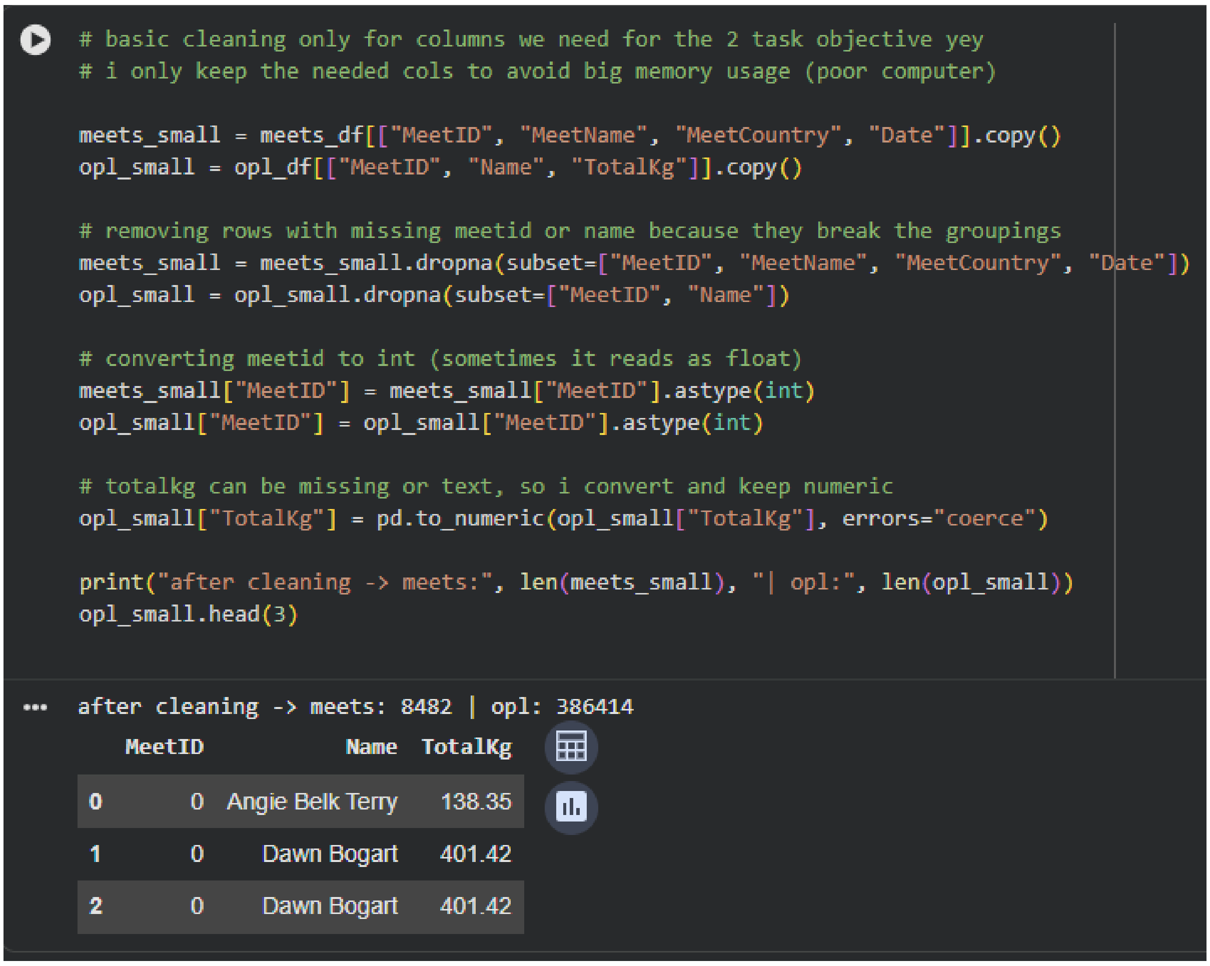
```
meets rows: 8482
openpowerlifting rows: 386414
meets columns: ['MeetID', 'MeetPath', 'Federation', 'Date', 'MeetCountry', 'MeetState', 'MeetTown', 'MeetName']
opl columns: ['MeetID', 'Name', 'Sex', 'Equipment', 'Age', 'Division', 'BodyweightKg', 'WeightClassKg', 'Squat4Kg', 'BestSquatKg', 'Bench4Kg', 'BestBenchKg', 'Deadlift4Kg', 'BestDeadliftKg', 'TotalKg', 'Place', 'Wilks']
   MeetID      MeetPath  Federation        Date  MeetCountry  MeetState  MeetTown                                        MeetName
0       0  365strong/1601    365Strong  2016-10-29        USA         NC  Charlotte  2016 Junior & Senior National Powerlifting Cha...
1       1  365strong/1602    365Strong  2016-11-19        USA         MO      Ozark           Thanksgiving Powerlifting Classic
2       2  365strong/1603    365Strong  2016-07-09        USA         NC  Charlotte                  Charlotte Europa Games
```

0 º CLEANNING

Before applying the MapReduce operations, a basic data preparation step was performed. Only the columns required for each task were kept in order to reduce memory usage and improve processing efficiency. Rows with missing critical values such as MeetID, athlete name, or total lifted weight were removed.

Additionally, data types were standardised. For example, MeetID was converted to an integer and TotalKg was converted to a numeric format. These steps ensure that the aggregation operations in the reduce phase behave correctly and avoid errors caused by missing or malformed values.

This cleaning process follows the general Big Data principle of minimal but necessary preprocessing, where only essential transformations are applied before distributed computation.



```python
# basic cleaning only for columns we need for the 2 task objective yey
# i only keep the needed cols to avoid big memory usage (poor computer)

meets_small = meets_df[["MeetID", "MeetName", "MeetCountry", "Date"]].copy()
opl_small = opl_df[["MeetID", "Name", "TotalKg"]].copy()

# removing rows with missing meetid or name because they break the groupings
meets_small = meets_small.dropna(subset=["MeetID", "MeetName", "MeetCountry", "Date"])
opl_small = opl_small.dropna(subset=["MeetID", "Name"])

# converting meetid to int (sometimes it reads as float)
meets_small["MeetID"] = meets_small["MeetID"].astype(int)
opl_small["MeetID"] = opl_small["MeetID"].astype(int)

# totalkg can be missing or text, so i convert and keep numeric
opl_small["TotalKg"] = pd.to_numeric(opl_small["TotalKg"], errors="coerce")

print("after cleaning -> meets:", len(meets_small), "| opl:", len(opl_small))
opl_small.head(3)
```

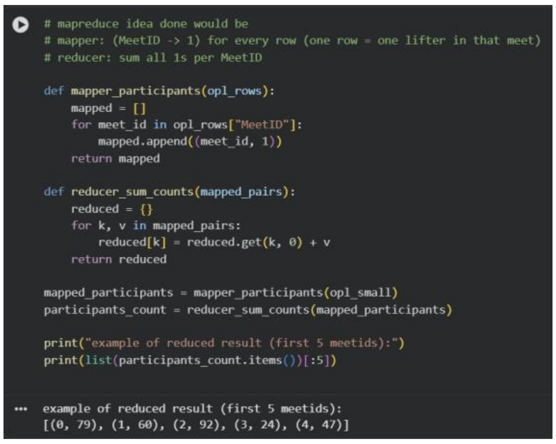
```
after cleaning -> meets: 8482 | opl: 386414
```

	MeetID	Name	TotalKg
0	0	Angie Belk Terry	138.35
1	0	Dawn Bogart	401.42
2	0	Dawn Bogart	401.42

1º) Nº PARTICIPANTS x COMPETITION

To calculate how many participants took part in each competition, a MapReduce approach was implemented. In the map phase, each row of the dataset emits a key-value pair where the key is the competition identifier (MeetID) and the value is 1, representing a single participant.

In the reduce phase, all values associated with the same MeetID are summed. This produces the total number of participants per competition. This approach is a classic example of a counting problem solved using MapReduce.

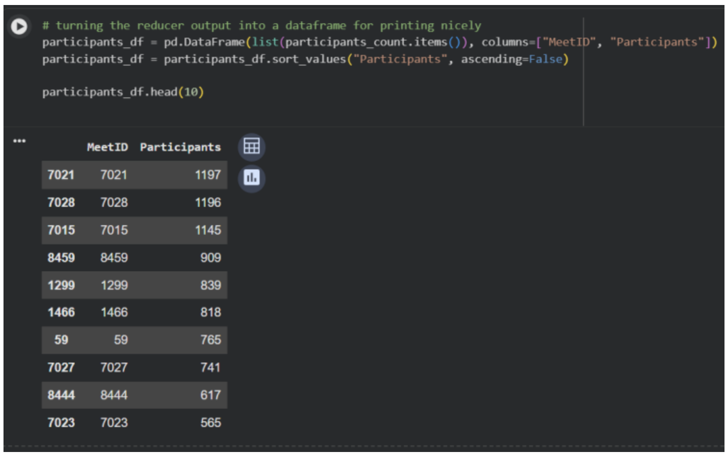


```python
# mapreduce idea done would be
# mapper: (MeetID -> 1) for every row (one row = one lifter in that meet)
# reducer: sum all 1s per MeetID

def mapper_participants(opl_rows):
    mapped = []
    for meet_id in opl_rows["MeetID"]:
        mapped.append((meet_id, 1))
    return mapped

def reducer_sum_counts(mapped_pairs):
    reduced = {}
    for k, v in mapped_pairs:
        reduced[k] = reduced.get(k, 0) + v
    return reduced

mapped_participants = mapper_participants(opl_small)
participants_count = reducer_sum_counts(mapped_participants)

print("example of reduced result (first 5 meetids):")
print(list(participants_count.items())[:5])
```

```
example of reduced result (first 5 meetids):
[(0, 79), (1, 60), (2, 92), (3, 24), (4, 47)]
```

INTERPRETATION: The results show that some competitions have a significantly higher number of participants than others. This indicates that certain events are larger or more popular, possibly due to their national or international relevance.

The distribution of participant counts across competitions highlights the variability in competition size, which is an important characteristic of real-world sports datasets.

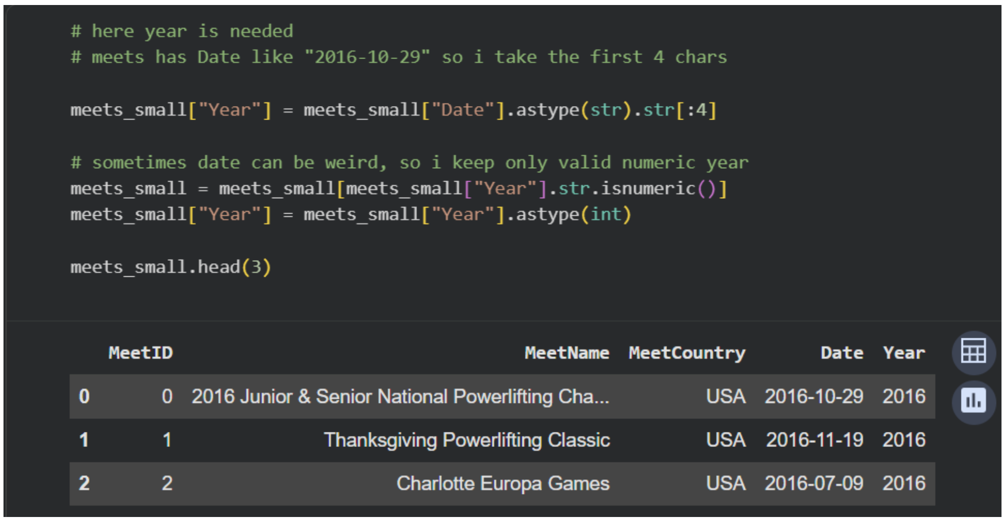


```python
# turning the reducer output into a dataframe for printing nicely
participants_df = pd.DataFrame(list(participants_count.items()), columns=["MeetID", "Participants"])
participants_df = participants_df.sort_values("Participants", ascending=False)

participants_df.head(10)
```

```
      MeetID  Participants
7021    7021          1197
7028    7028          1196
7015    7015          1145
8459    8459           909
1299    1299           839
1466    1466           818
  59      59           765
7027    7027           741
8444    8444           617
7023    7023           565
```

2º) LIST COMPETITIONS (COUNTRY, YEAR & PARTICIPANTS)

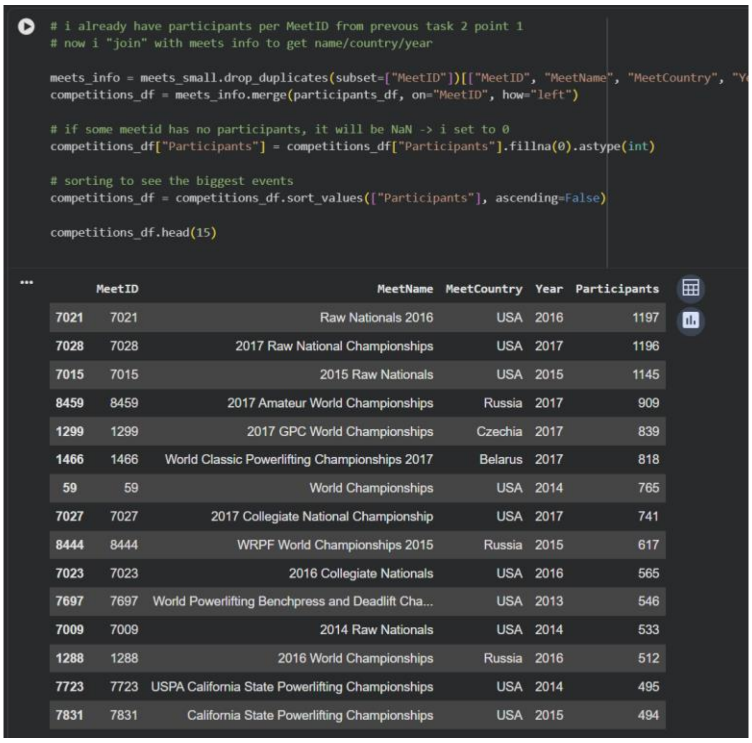
For this task, the participant counts obtained in Task 2.1 were combined with competition metadata such as competition name, country, and year. The year was extracted from the competition date to allow temporal analysis.



```python
# here year is needed
# meets has Date like "2016-10-29" so i take the first 4 chars

meets_small["Year"] = meets_small["Date"].astype(str).str[:4]

# sometimes date can be weird, so i keep only valid numeric year
meets_small = meets_small[meets_small["Year"].str.isnumeric()]
meets_small["Year"] = meets_small["Year"].astype(int)

meets_small.head(3)
```

	MeetID	MeetName	MeetCountry	Date	Year
0	0	2016 Junior & Senior National Powerlifting Cha...	USA	2016-10-29	2016
1	1	Thanksgiving Powerlifting Classic	USA	2016-11-19	2016
2	2	Charlotte Europa Games	USA	2016-07-09	2016

This step resembles a reduce-side join, where aggregated results from one MapReduce process are joined with additional reference data. This approach is common in Big Data systems when combining results from multiple datasets.



```python
# i already have participants per MeetID from prevous task 2 point 1
# now i "join" with meets info to get name/country/year

meets_info = meets_small.drop_duplicates(subset=["MeetID"])[["MeetID", "MeetName", "MeetCountry", "Ye
competitions_df = meets_info.merge(participants_df, on="MeetID", how="left")

# if some meetid has no participants, it will be NaN -> i set to 0
competitions_df["Participants"] = competitions_df["Participants"].fillna(0).astype(int)

# sorting to see the biggest events
competitions_df = competitions_df.sort_values(["Participants"], ascending=False)

competitions_df.head(15)
```

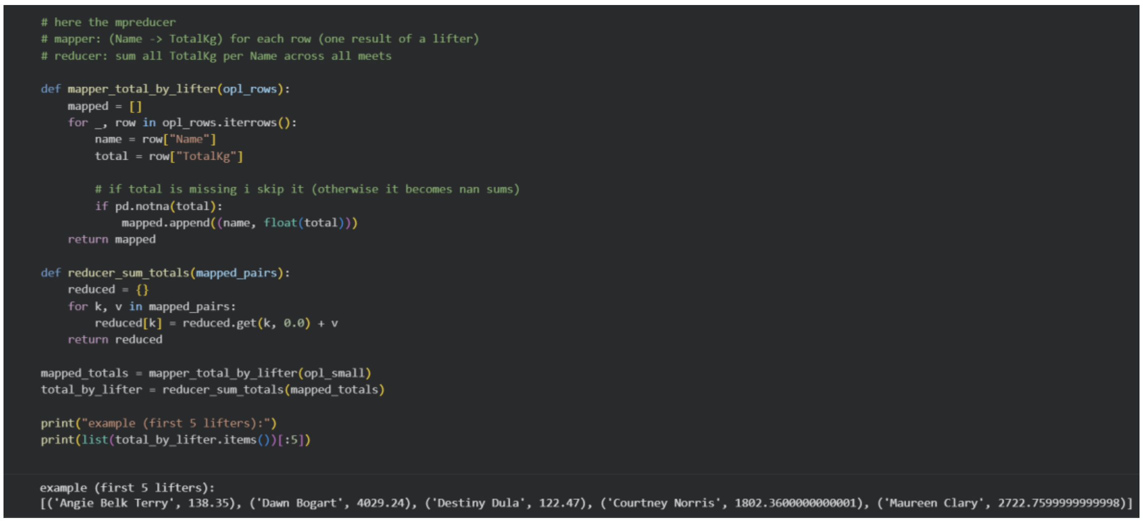
	MeetID	MeetName	MeetCountry	Year	Participants
7021	7021	Raw Nationals 2016	USA	2016	1197
7028	7028	2017 Raw National Championships	USA	2017	1196
7015	7015	2015 Raw Nationals	USA	2015	1145
8459	8459	2017 Amateur World Championships	Russia	2017	909
1299	1299	2017 GPC World Championships	Czechia	2017	839
1466	1466	World Classic Powerlifting Championships 2017	Belarus	2017	818
59	59	World Championships	USA	2014	765
7027	7027	2017 Collegiate National Championship	USA	2017	741
8444	8444	WRPF World Championships 2015	Russia	2015	617
7023	7023	2016 Collegiate Nationals	USA	2016	565
7697	7697	World Powerlifting Benchpress and Deadlift Cha...	USA	2013	546
7009	7009	2014 Raw Nationals	USA	2014	533
1288	1288	2016 World Championships	Russia	2016	512
7723	7723	USPA California State Powerlifting Championships	USA	2014	495
7831	7831	California State Powerlifting Championships	USA	2015	494

INTERPRETATION: The resulting competition list provides a clear overview of powerlifting events, including where and when they took place, as well as how many participants competed in each event.

The results show that competitions are spread across multiple countries and years, with noticeable differences in participation levels. This information could be useful for analysing trends in the growth of powerlifting or identifying major events within the sport.

3º) PER PARTICIPANT -> TOTAL WEIGHT THEY LIFT

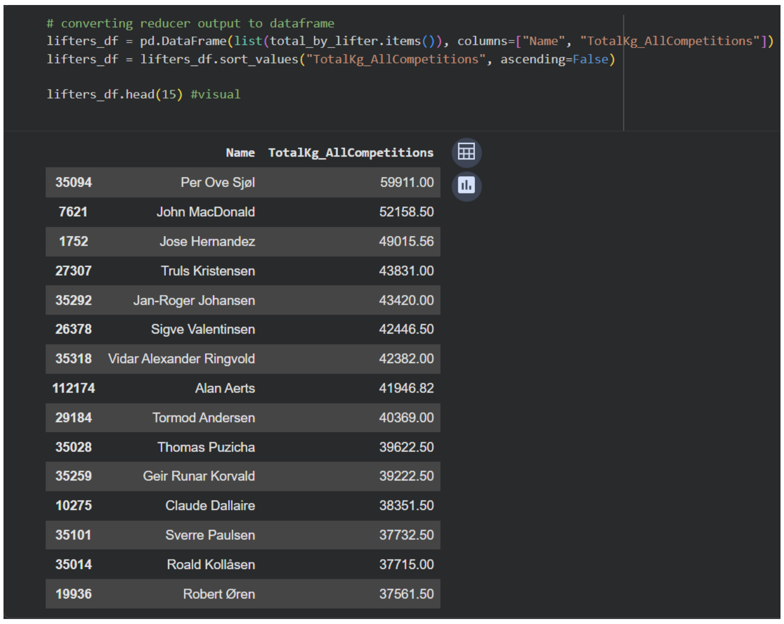
To calculate the total weight lifted by each participant across all competitions, another MapReduce process was applied. In the map phase, each row emits a key-value pair where the key is the athlete's name and the value is the total weight lifted in that competition.



```
# here the mpreducer
# mapper: (Name -> TotalKg) for each row (one result of a lifter)
# reducer: sum all TotalKg per Name across all meets

def mapper_total_by_lifter(opl_rows):
    mapped = []
    for _, row in opl_rows.iterrows():
        name = row["Name"]
        total = row["TotalKg"]

        # if total is missing i skip it (otherwise it becomes nan sums)
        if pd.notna(total):
            mapped.append((name, float(total)))
    return mapped

def reducer_sum_totals(mapped_pairs):
    reduced = {}
    for k, v in mapped_pairs:
        reduced[k] = reduced.get(k, 0.0) + v
    return reduced

mapped_totals = mapper_total_by_lifter(opl_small)
total_by_lifter = reducer_sum_totals(mapped_totals)

print("example (first 5 lifters):")
print(list(total_by_lifter.items())[:5])
```

```
example (first 5 lifters):
[('Angie Belk Terry', 138.35), ('Dawn Bogart', 4029.24), ('Destiny Dula', 122.47), ('Courtney Norris', 1802.3600000000001), ('Maureen Clary', 2722.7599999999998)]
```

In the reduce phase, all values associated with the same athlete are summed, producing the total weight lifted by that athlete across all recorded competitions. This approach demonstrates how MapReduce can be used to aggregate numerical values across very large datasets.

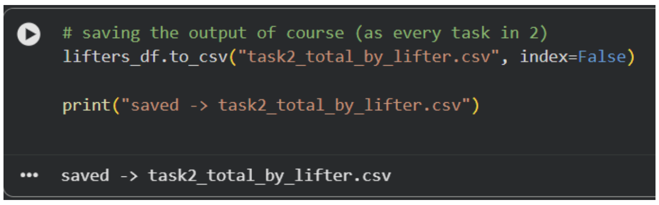


```python
# converting reducer output to dataframe
lifters_df = pd.DataFrame(list(total_by_lifter.items()), columns=["Name", "TotalKg_AllCompetitions"])
lifters_df = lifters_df.sort_values("TotalKg_AllCompetitions", ascending=False)

lifters_df.head(15) #visual
```

	Name	TotalKg_AllCompetitions
35094	Per Ove Sjøl	59911.00
7621	John MacDonald	52158.50
1752	Jose Hernandez	49015.56
27307	Truls Kristensen	43831.00
35292	Jan-Roger Johansen	43420.00
26378	Sigve Valentinsen	42446.50
35318	Vidar Alexander Ringvold	42382.00
112174	Alan Aerts	41946.82
29184	Tormod Andersen	40369.00
35028	Thomas Puzicha	39622.50
35259	Geir Runar Korvald	39222.50
10275	Claude Dallaire	38351.50
35101	Sverre Paulsen	37732.50
35014	Roald Kollåsen	37715.00
19936	Robert Øren	37561.50

INTERPRETATION; Here seen highlight athletes who have accumulated very high total lifted weights across multiple competitions. This typically indicates athletes with long competitive careers or frequent participation.

However, a limitation of this approach is that athletes are identified only by name. This means that different individuals with the same name may be aggregated together. Despite this limitation, the results still provide a meaningful overview of cumulative performance in the dataset. The results are saven into a csv



```python
# saving the output of course (as every task in 2)
lifters_df.to_csv("task2_total_by_lifter.csv", index=False)

print("saved -> task2_total_by_lifter.csv")
```

```
saved -> task2_total_by_lifter.csv
```

Task 3 – Big Data & Technology talk

During this final project task's, I critically evaluate the Big Data tools and techniques used in Task 1 and Task 2 of this assignment. The goal of this section is to reflect on how PySpark an MapReduce helped to solve the proposed problems, based on what I actually implemented in the notebooks and on the theoretical concepts explained during the lectures in estudijas uploaded also

1º task 1 talk) __PySpark__

In Task 1, PySpark was used to build a recommendation system for the Vehicle CO2 Emissions dataset. One of the first things I had to consider was the structure of the dataset. It only contained vehicle characteristics and no users or ratings, which meant that collaborative filtering methods such as ALS were not appropriate for this problem.

Because of this, I implemented a content-based recommendation system, where cars are recommended based on similarity between their technical attributes. PySpark was very useful for this task, especially for data cleaning, exploratory analysis, and feature engineering. Using Spark DataFrames allowed me to work with the dataset in a structured and scalable way, applying filters, groupings, and aggregations efficiently.

The feature engineering process followed the pipeline approach presented in the lectures. Categorical variables were encoded using `StringIndexer` and `OneHotEncoder`, while numerical features were scaled using `StandardScaler`. This step was important to ensure that all features contributed fairly to the similarity calculation. The use of a machine learning pipeline also made the code cleaner and easier to follow.

For the recommendation step, LSH (Locality Sensitive Hashing) was used to find similar vehicles based on their feature vectors. This approach fits well with Big Data concepts, as it avoids computing exact distances between all vehicles and instead provides an efficient approximate solution.

However, PySpark also has some limitations (also for use, that's why colab I used). Debugging can be challenging, especially when working with pipelines, and error messages are not always very clear. In addition, the evaluation of the recommendation system was qualitative, since there is no ground truth or user feedback available. This limits the possibility of using quantitative evaluation metrics.

2º task 2 talk) ***<u>MapReduce</u>***

In Task 2, MapReduce was used to analyse a large powerlifting dataset with more than 380,000 records. The tasks mainly focused on counting participants, summarising competitions, and calculating the total weight lifted by each athlete across all competitions.

MapReduce was a good choice for these tasks because they are classic aggregation problems. The map and reduce phases made it very clear how the data was processed step by step. This aligns well with the theoretical explanation of MapReduce given in the lectures, where the emphasis was on scalability and distributed batch processing.

One of the main strengths of MapReduce is its simplicity and robustness when dealing with very large datasets. Counting and summing values across competitions and participants scaled well and did not require complex logic.

On the other hand, MapReduce also showed some drawbacks. The code is more verbose and less flexible compared to PySpark other friend of ours used, and even simple analyses require multiple steps. It is also not well suited for iterative or interactive analysis, which makes it less practical for exploratory data analysis or machine learning tasks (#team PySpark).

COMPARISON -> *Scalability & more*

Both PySpark and MapReduce are designed to handle large-scale data, but they are better suited for different types of problems. PySpark is more flexible and powerful for analytics and machine learning tasks, especially when multiple transformations and feature engineering steps are needed. Its in-memory processing model makes it more efficient for iterative workloads.

MapReduce, in contrast, is more suitable for large batch processing tasks that involve straightforward aggregations. While it scales well, it is less efficient and more rigid when dealing with complex analytics.

In conclusion… After getting in touch and hands on the task with PySpark (allowed me to design a scalable content-based recommendation system) and MapReduce (effective for processing and aggregating large volumes of competition data), I would say that this assignment helped me understand how important it is to choose the right Big Data tool depending on the dataset structure and the problem to be solved.