Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Advanced Data Engineering
Course Number:	EE8222
Semester/Year (e.g.F2016)	W2024

Instructor:	Ghassem Tofighi

Assignment/Lab Number:	1
Assignment/Lab Title:	Hadoop, Hive, and SparkSQL

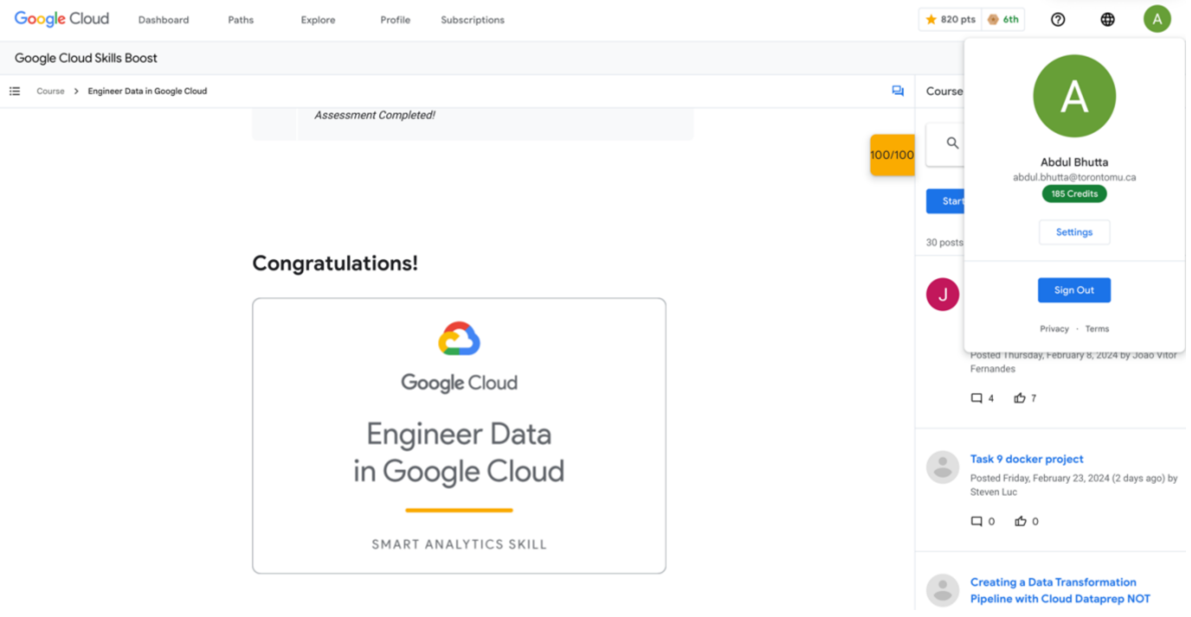
Submission Date:	Monday, February 26, 2024
Due Date:	Monday, February 26, 2024

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Bhutta	Abdul	500447025	1	AB

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Deliverable Part 1 – Engineer Data in Google Cloud Quest

Engineer Data in Google Cloud Badge



Assessment Completed!

Congratulations!

Google Cloud

Engineer Data in Google Cloud

SMART ANALYTICS SKILL

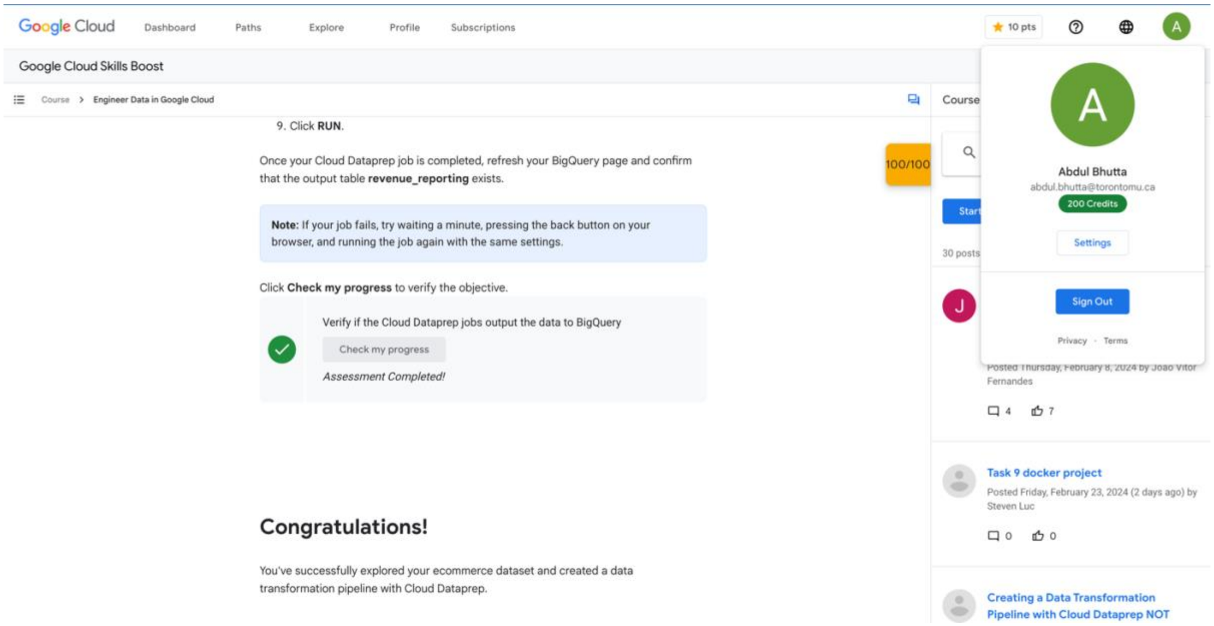
Abdul Bhutta
abdul.bhutta@torontomu.ca
185 Credits

Posted Thursday, February 8, 2024 by Joao Vitor Fernandes

Task 9 docker project
Posted Friday, February 23, 2024 (2 days ago) by Steven Luc

Creating a Data Transformation Pipeline with Cloud Dataprep NOT

Lab 1 completion



9. Click **RUN**.

Once your Cloud Dataprep job is completed, refresh your BigQuery page and confirm that the output table **revenue_reporting** exists.

Note: If your job fails, try waiting a minute, pressing the back button on your browser, and running the job again with the same settings.

Click **Check my progress** to verify the objective.

✓ Verify if the Cloud Dataprep jobs output the data to BigQuery

Assessment Completed!

Congratulations!

You've successfully explored your ecommerce dataset and created a data transformation pipeline with Cloud Dataprep.

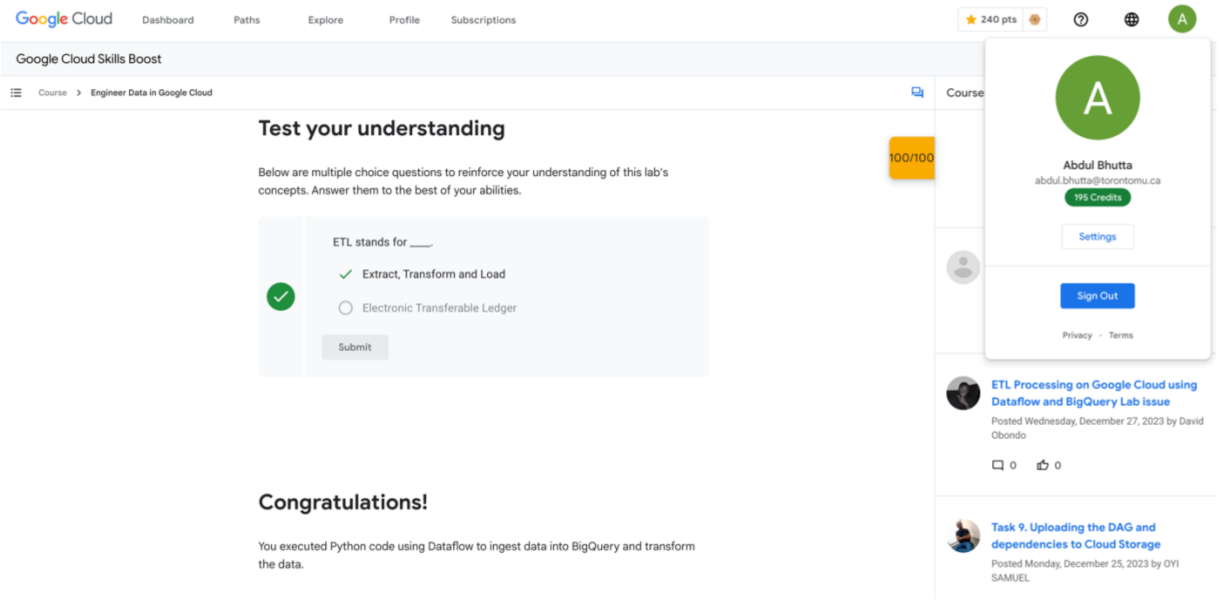
Abdul Bhutta
abdul.bhutta@torontomu.ca
200 Credits

Posted Thursday, February 8, 2024 by Joao Vitor Fernandes

Task 9 docker project
Posted Friday, February 23, 2024 (2 days ago) by Steven Luc

Creating a Data Transformation Pipeline with Cloud Dataprep NOT

Lab 2 completion



Test your understanding

Below are multiple choice questions to reinforce your understanding of this lab's concepts. Answer them to the best of your abilities.

ETL stands for ____.

✓ Extract, Transform and Load

○ Electronic Transferable Ledger

Submit

Congratulations!

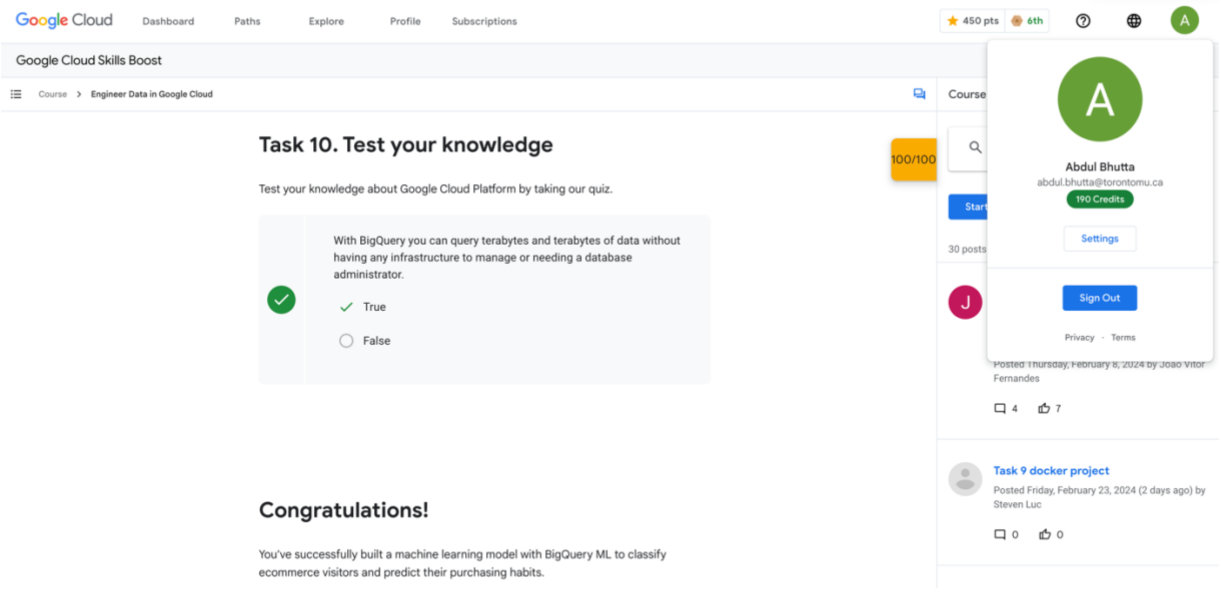
You executed Python code using Dataflow to ingest data into BigQuery and transform the data.

Abdul Bhutta
abdul.bhutta@torontomu.ca
195 Credits

ETL Processing on Google Cloud using Dataflow and BigQuery Lab issue
Posted Wednesday, December 27, 2023 by David Obondo

Task 9. Uploading the DAG and dependencies to Cloud Storage
Posted Monday, December 25, 2023 by OYI SAMUEL

Lab 3 completion



Task 10. Test your knowledge

Test your knowledge about Google Cloud Platform by taking our quiz.

With BigQuery you can query terabytes and terabytes of data without having any infrastructure to manage or needing a database administrator.

✓ True

○ False

Congratulations!

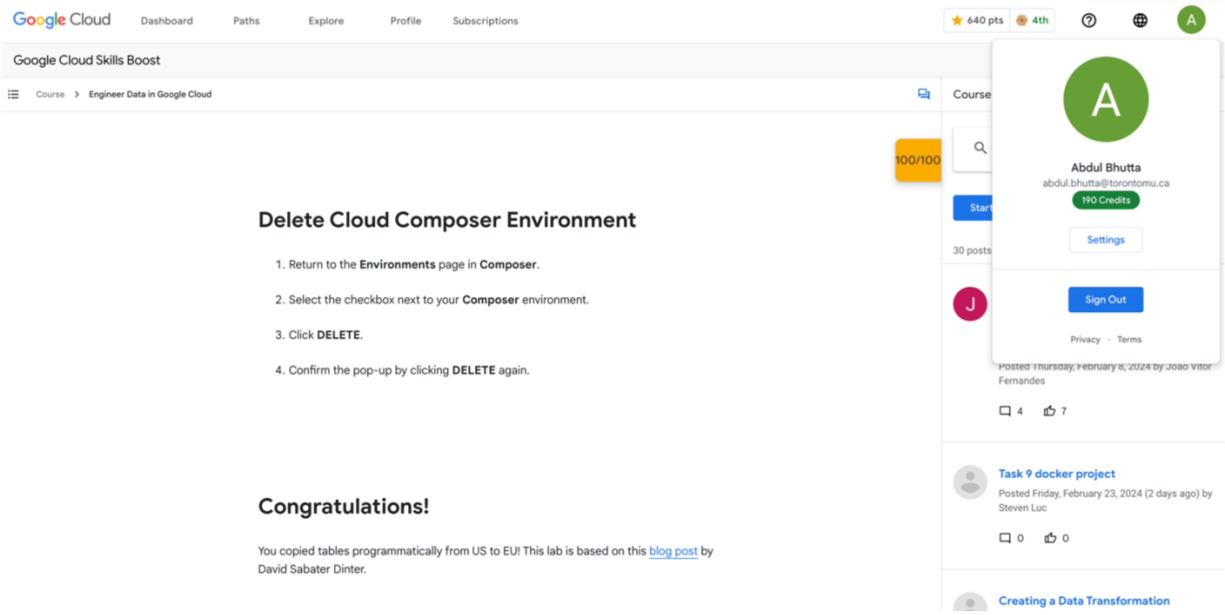
You've successfully built a machine learning model with BigQuery ML to classify ecommerce visitors and predict their purchasing habits.

Abdul Bhutta
abdul.bhutta@torontomu.ca
190 Credits

Posted Thursday, February 8, 2024 by Joao Vitor Fernandes

Task 9 docker project
Posted Friday, February 23, 2024 (2 days ago) by Steven Luc

Lab 4 Completion



Abdul Bhutta
abdul.bhutta@torontomu.ca
190 Credits

Delete Cloud Composer Environment

1. Return to the **Environments** page in **Composer**.
2. Select the checkbox next to your **Composer** environment.
3. Click **DELETE**.
4. Confirm the pop-up by clicking **DELETE** again.

Congratulations!

You copied tables programmatically from US to EU! This lab is based on this blog post by David Sabater Dinter.

Posted Thursday, February 8, 2024 by Joao Vitor Fernandes

□ 4 👍 7

Task 9 docker project

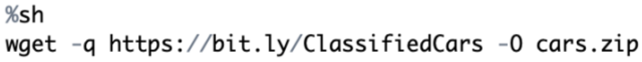
Posted Friday, February 23, 2024 (2 days ago) by Steven Luc

□ 0 👍 0

Creating a Data Transformation

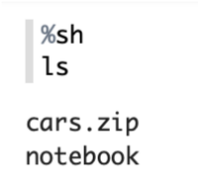
Deliverable Part 2 - Hive Tasks

Download Dataset



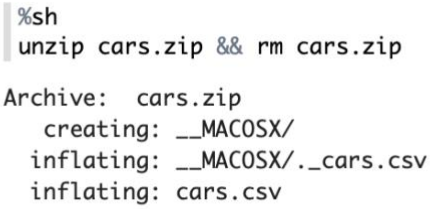
```
%sh
wget -q https://bit.ly/ClassifiedCars -O cars.zip
```

Verify the dataset has been downloaded



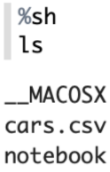
```
%sh
ls

cars.zip
notebook
```

Unzip the dataset and delete the zip file



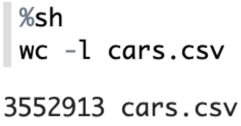
```
%sh
unzip cars.zip && rm cars.zip

Archive:  cars.zip
   creating: __MACOSX/
  inflating: __MACOSX/._cars.csv
  inflating: cars.csv
```

Verify the csv file has been extracted



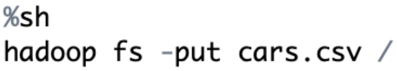
```
%sh
ls

__MACOSX
cars.csv
notebook
```

Verify total rows in the dataset



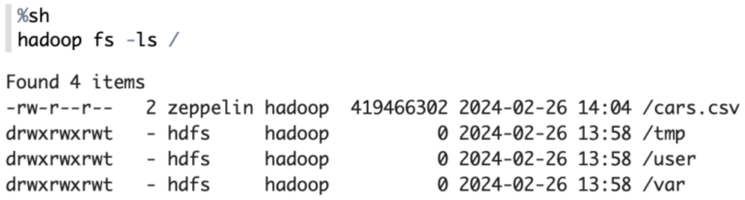
```
%sh
wc -l cars.csv

3552913 cars.csv
```

Transfer to Hadoop root folder



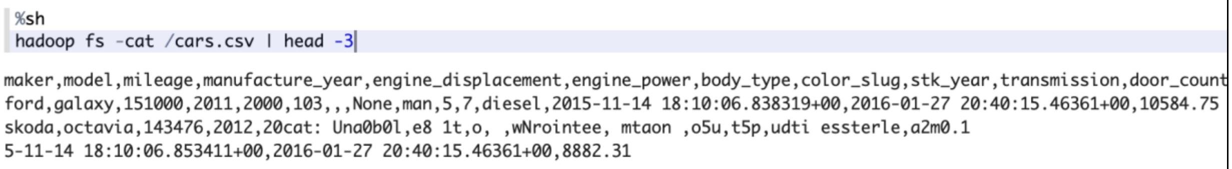
```
%sh
hadoop fs -put cars.csv /
```

Verify the csv file is in the root folder



```
%sh
hadoop fs -ls /

Found 4 items
-rw-r--r--   2 zeppelin hadoop  419466302 2024-02-26 14:04 /cars.csv
drwxrwxrwt   - hdfs     hadoop          0 2024-02-26 13:58 /tmp
drwxrwxrwt   - hdfs     hadoop          0 2024-02-26 13:58 /user
drwxrwxrwt   - hdfs     hadoop          0 2024-02-26 13:58 /var
```

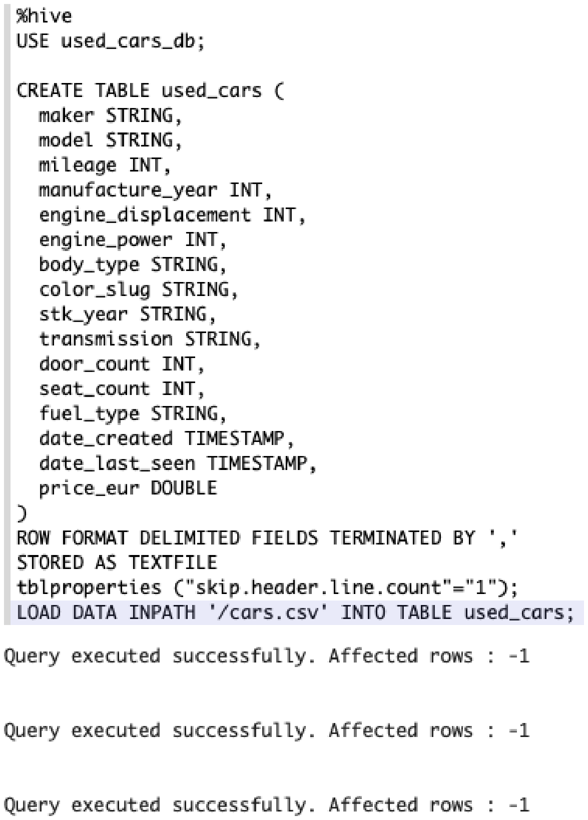
Verify the data in the csv file



```
%sh
hadoop fs -cat /cars.csv | head -3

maker,model,mileage,manufacture_year,engine_displacement,engine_power,body_type,color_slug,stk_year,transmission,door_count
ford,galaxy,151000,2011,2000,103,,,None,man,5,7,diesel,2015-11-14 18:10:06.838319+00,2016-01-27 20:40:15.46361+00,10584.75
skoda,octavia,143476,2012,20cat: Una0b0l,e8 1t,o, ,wNrointee, mtaon ,o5u,t5p,udti esstere,a2m0.1
5-11-14 18:10:06.853411+00,2016-01-27 20:40:15.46361+00,8882.31
```

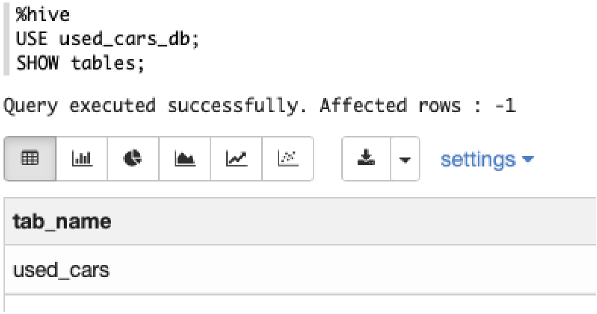
1. **Write a Hive query to create a table called used_cars from the data.**

Hive query to create a table for used_cars



```
%hive
USE used_cars_db;

CREATE TABLE used_cars (
  maker STRING,
  model STRING,
  mileage INT,
  manufacture_year INT,
  engine_displacement INT,
  engine_power INT,
  body_type STRING,
  color_slug STRING,
  stk_year STRING,
  transmission STRING,
  door_count INT,
  seat_count INT,
  fuel_type STRING,
  date_created TIMESTAMP,
  date_last_seen TIMESTAMP,
  price_eur DOUBLE
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
tblproperties ("skip.header.line.count"="1");
LOAD DATA INPATH '/cars.csv' INTO TABLE used_cars;

Query executed successfully. Affected rows : -1


Query executed successfully. Affected rows : -1


Query executed successfully. Affected rows : -1
```

Verify table has been created

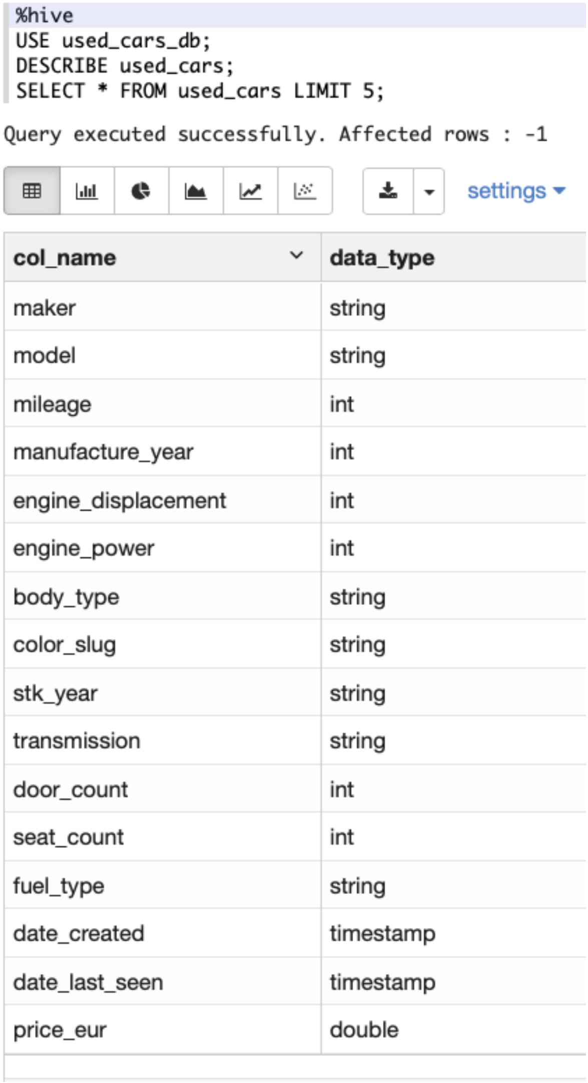


```
%hive
USE used_cars_db;
SHOW tables;
```

```
Query executed successfully. Affected rows : -1
```

tab_name
used_cars

Verify all the columns are correct



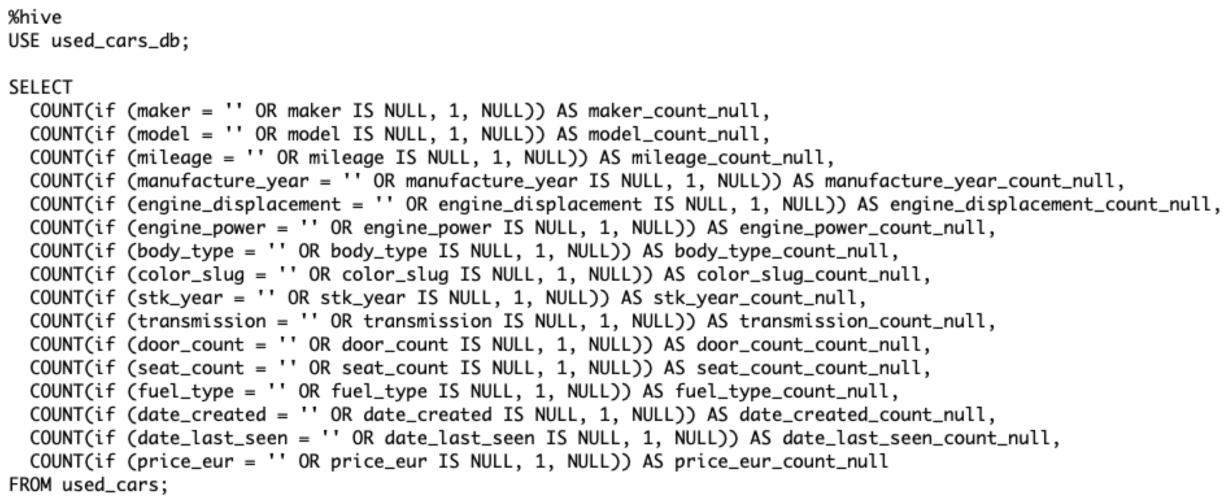
```
%hive
USE used_cars_db;
DESCRIBE used_cars;
SELECT * FROM used_cars LIMIT 5;
```

```
Query executed successfully. Affected rows : -1
```

col_name	data_type
maker	string
model	string
mileage	int
manufacture_year	int
engine_displacement	int
engine_power	int
body_type	string
color_slug	string
stk_year	string
transmission	string
door_count	int
seat_count	int
fuel_type	string
date_created	timestamp
date_last_seen	timestamp
price_eur	double

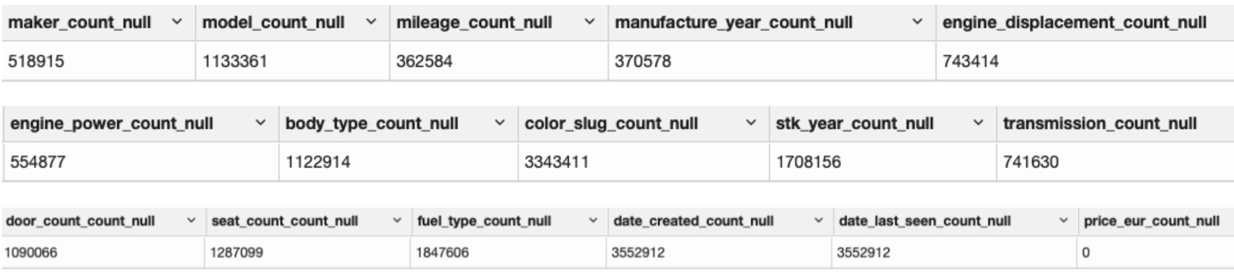
2. **Write several Hive queries to find how many missing values (NULL in an attribute of a record) you have in each column (attribute).**

Query to check if it contains NULL or an empty string (‘’)



```
%hive
USE used_cars_db;

SELECT
  COUNT(if (maker = '' OR maker IS NULL, 1, NULL)) AS maker_count_null,
  COUNT(if (model = '' OR model IS NULL, 1, NULL)) AS model_count_null,
  COUNT(if (mileage = '' OR mileage IS NULL, 1, NULL)) AS mileage_count_null,
  COUNT(if (manufacture_year = '' OR manufacture_year IS NULL, 1, NULL)) AS manufacture_year_count_null,
  COUNT(if (engine_displacement = '' OR engine_displacement IS NULL, 1, NULL)) AS engine_displacement_count_null,
  COUNT(if (engine_power = '' OR engine_power IS NULL, 1, NULL)) AS engine_power_count_null,
  COUNT(if (body_type = '' OR body_type IS NULL, 1, NULL)) AS body_type_count_null,
  COUNT(if (color_slug = '' OR color_slug IS NULL, 1, NULL)) AS color_slug_count_null,
  COUNT(if (stk_year = '' OR stk_year IS NULL, 1, NULL)) AS stk_year_count_null,
  COUNT(if (transmission = '' OR transmission IS NULL, 1, NULL)) AS transmission_count_null,
  COUNT(if (door_count = '' OR door_count IS NULL, 1, NULL)) AS door_count_count_null,
  COUNT(if (seat_count = '' OR seat_count IS NULL, 1, NULL)) AS seat_count_count_null,
  COUNT(if (fuel_type = '' OR fuel_type IS NULL, 1, NULL)) AS fuel_type_count_null,
  COUNT(if (date_created = '' OR date_created IS NULL, 1, NULL)) AS date_created_count_null,
  COUNT(if (date_last_seen = '' OR date_last_seen IS NULL, 1, NULL)) AS date_last_seen_count_null,
  COUNT(if (price_eur = '' OR price_eur IS NULL, 1, NULL)) AS price_eur_count_null
FROM used_cars;
```

Output of the query



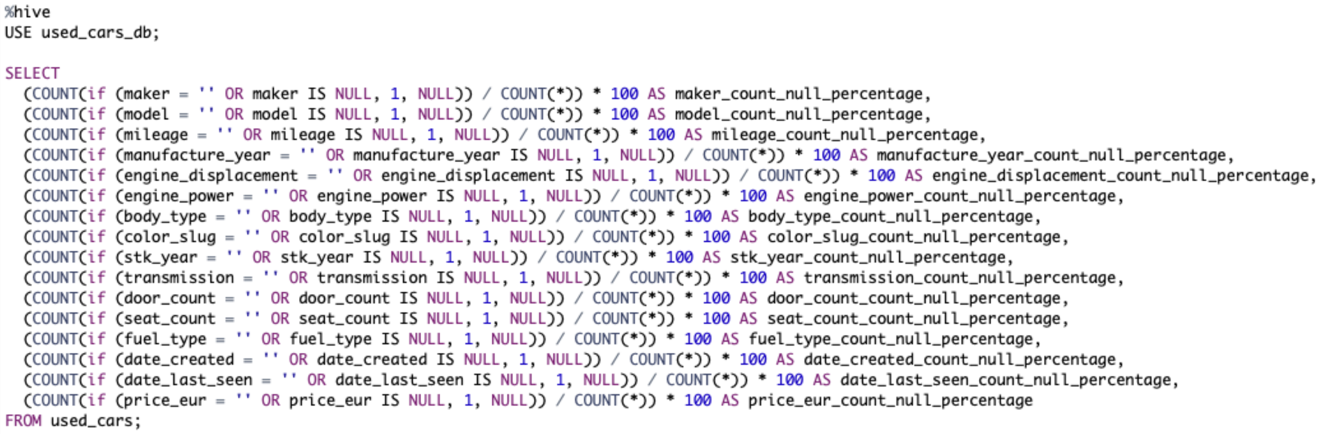
maker_count_null	model_count_null	mileage_count_null	manufacture_year_count_null	engine_displacement_count_null
518915	1133361	362584	370578	743414

engine_power_count_null	body_type_count_null	color_slug_count_null	stk_year_count_null	transmission_count_null
554877	1122914	3343411	1708156	741630

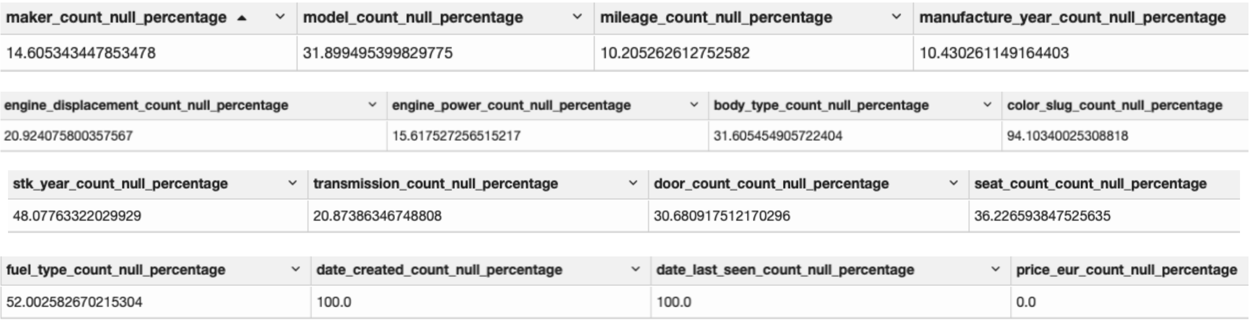
door_count_count_null	seat_count_count_null	fuel_type_count_null	date_created_count_null	date_last_seen_count_null	price_eur_count_null
1090066	1287099	1847606	3552912	3552912	0

3. **Drop the columns (attribute) with more than 50% missing values. (For example if 50% or more of body_type is missing, drop body_type column using REPLACE COLUMNS using HQL)**

Calculate the columns with 50% or more missing values



```
%hive
USE used_cars_db;

SELECT
  (COUNT(if (maker = '' OR maker IS NULL, 1, NULL)) / COUNT(*)) * 100 AS maker_count_null_percentage,
  (COUNT(if (model = '' OR model IS NULL, 1, NULL)) / COUNT(*)) * 100 AS model_count_null_percentage,
  (COUNT(if (mileage = '' OR mileage IS NULL, 1, NULL)) / COUNT(*)) * 100 AS mileage_count_null_percentage,
  (COUNT(if (manufacture_year = '' OR manufacture_year IS NULL, 1, NULL)) / COUNT(*)) * 100 AS manufacture_year_count_null_percentage,
  (COUNT(if (engine_displacement = '' OR engine_displacement IS NULL, 1, NULL)) / COUNT(*)) * 100 AS engine_displacement_count_null_percentage,
  (COUNT(if (engine_power = '' OR engine_power IS NULL, 1, NULL)) / COUNT(*)) * 100 AS engine_power_count_null_percentage,
  (COUNT(if (body_type = '' OR body_type IS NULL, 1, NULL)) / COUNT(*)) * 100 AS body_type_count_null_percentage,
  (COUNT(if (color_slug = '' OR color_slug IS NULL, 1, NULL)) / COUNT(*)) * 100 AS color_slug_count_null_percentage,
  (COUNT(if (stk_year = '' OR stk_year IS NULL, 1, NULL)) / COUNT(*)) * 100 AS stk_year_count_null_percentage,
  (COUNT(if (transmission = '' OR transmission IS NULL, 1, NULL)) / COUNT(*)) * 100 AS transmission_count_null_percentage,
  (COUNT(if (door_count = '' OR door_count IS NULL, 1, NULL)) / COUNT(*)) * 100 AS door_count_count_null_percentage,
  (COUNT(if (seat_count = '' OR seat_count IS NULL, 1, NULL)) / COUNT(*)) * 100 AS seat_count_count_null_percentage,
  (COUNT(if (fuel_type = '' OR fuel_type IS NULL, 1, NULL)) / COUNT(*)) * 100 AS fuel_type_count_null_percentage,
  (COUNT(if (date_created = '' OR date_created IS NULL, 1, NULL)) / COUNT(*)) * 100 AS date_created_count_null_percentage,
  (COUNT(if (date_last_seen = '' OR date_last_seen IS NULL, 1, NULL)) / COUNT(*)) * 100 AS date_last_seen_count_null_percentage,
  (COUNT(if (price_eur = '' OR price_eur IS NULL, 1, NULL)) / COUNT(*)) * 100 AS price_eur_count_null_percentage
FROM used_cars;
```



maker_count_null_percentage	model_count_null_percentage	mileage_count_null_percentage	manufacture_year_count_null_percentage
14.605343447853478	31.899495399829775	10.205262612752582	10.430261149164403

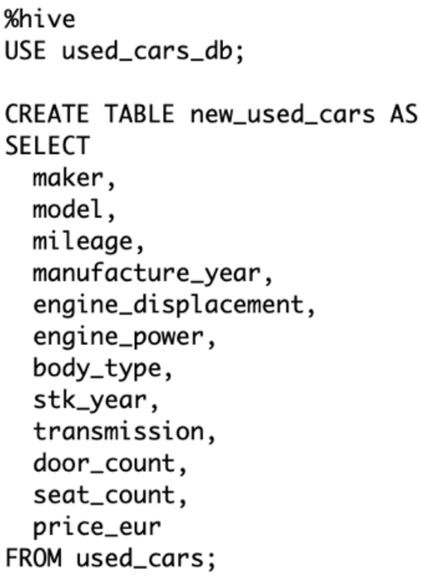
engine_displacement_count_null_percentage	engine_power_count_null_percentage	body_type_count_null_percentage	color_slug_count_null_percentage
20.924075800357567	15.617527256515217	31.605454905722404	94.10340025308818

stk_year_count_null_percentage	transmission_count_null_percentage	door_count_count_null_percentage	seat_count_count_null_percentage
48.07763322029929	20.87386346748808	30.680917512170296	36.226593847525635

fuel_type_count_null_percentage	date_created_count_null_percentage	date_last_seen_count_null_percentage	price_eur_count_null_percentage
52.002582670215304	100.0	100.0	0.0

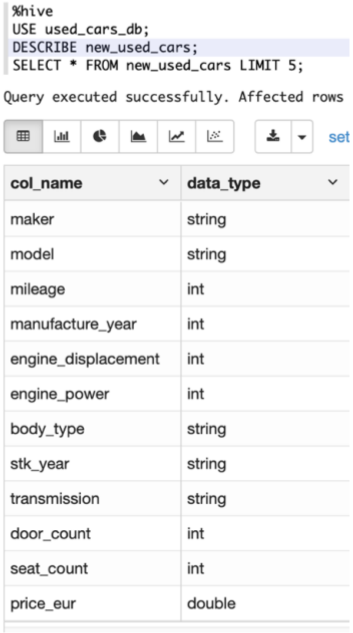
Drop the following columns: color_slug(94%), fuel_type(52%), date_created(100%), date_last_seen(100%)

Create a new schema for the updated table



```
%hive
USE used_cars_db;

CREATE TABLE new_used_cars AS
SELECT
  maker,
  model,
  mileage,
  manufacture_year,
  engine_displacement,
  engine_power,
  body_type,
  stk_year,
  transmission,
  door_count,
  seat_count,
  price_eur
FROM used_cars;
```

Verify the columns

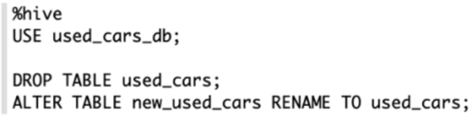


```
%hive
USE used_cars_db;
DESCRIBE new_used_cars;
SELECT * FROM new_used_cars LIMIT 5;
```

```
Query executed successfully. Affected rows
```

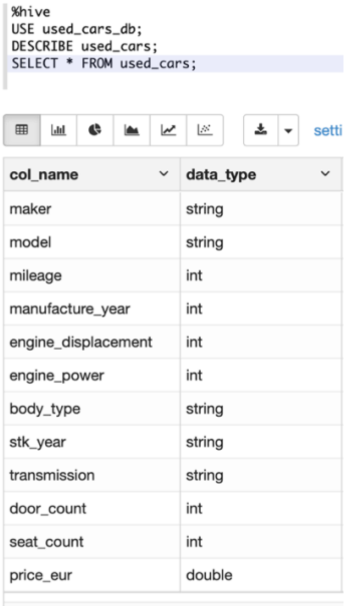
col_name	data_type
maker	string
model	string
mileage	int
manufacture_year	int
engine_displacement	int
engine_power	int
body_type	string
stk_year	string
transmission	string
door_count	int
seat_count	int
price_eur	double

Drop the old table and rename the new one to used_cars



```
%hive
USE used_cars_db;

DROP TABLE used_cars;
ALTER TABLE new_used_cars RENAME TO used_cars;
```

Verify its been updated



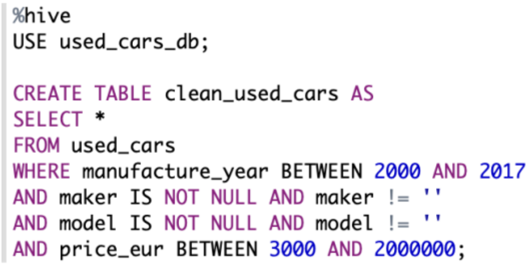
```
%hive
USE used_cars_db;
DESCRIBE used_cars;
SELECT * FROM used_cars;
```

col_name	data_type
maker	string
model	string
mileage	int
manufacture_year	int
engine_displacement	int
engine_power	int
body_type	string
stk_year	string
transmission	string
door_count	int
seat_count	int
price_eur	double

4. Write several Hive queries to create a new table called clean_used_cars from used_cars with the following conditions:

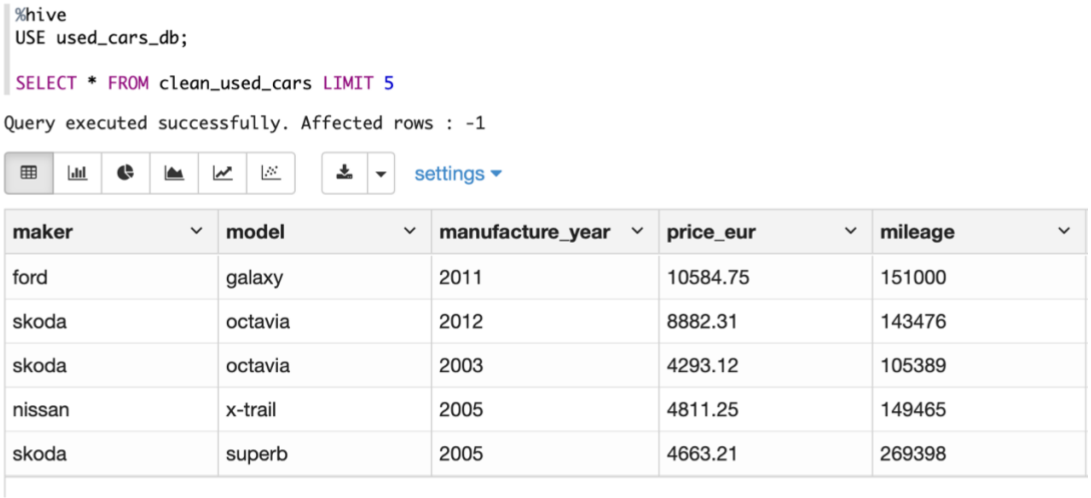
- *The manufacturing year between 2000 and 2017 including 2000 and 2017*
- *Both maker and model exist (NOT NULL) in the row*
- *The price range is from 3000 to 2,000,000 (3000 ≤ price ≤ 2,000,000)*

Hive query to create a new table with the given conditions



```
%hive
USE used_cars_db;

CREATE TABLE clean_used_cars AS
SELECT *
FROM used_cars
WHERE manufacture_year BETWEEN 2000 AND 2017
AND maker IS NOT NULL AND maker != ''
AND model IS NOT NULL AND model != ''
AND price_eur BETWEEN 3000 AND 2000000;
```

Verify output

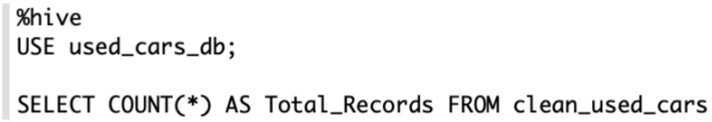


```
%hive
USE used_cars_db;

SELECT * FROM clean_used_cars LIMIT 5

Query executed successfully. Affected rows : -1
```

maker	model	manufacture_year	price_eur	mileage
ford	galaxy	2011	10584.75	151000
skoda	octavia	2012	8882.31	143476
skoda	octavia	2003	4293.12	105389
nissan	x-trail	2005	4811.25	149465
skoda	superb	2005	4663.21	269398

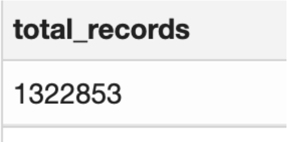
5. Write Hive to find how many records remained clean_used_cars.

Hive query to determin the total records in the clean_used_cars



```
%hive
USE used_cars_db;

SELECT COUNT(*) AS Total_Records FROM clean_used_cars
```

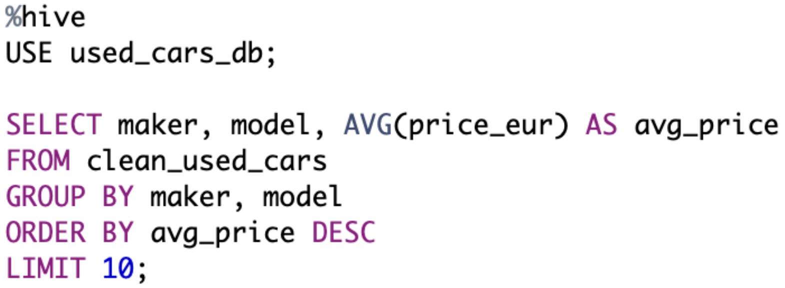
Query output



total_records
1322853

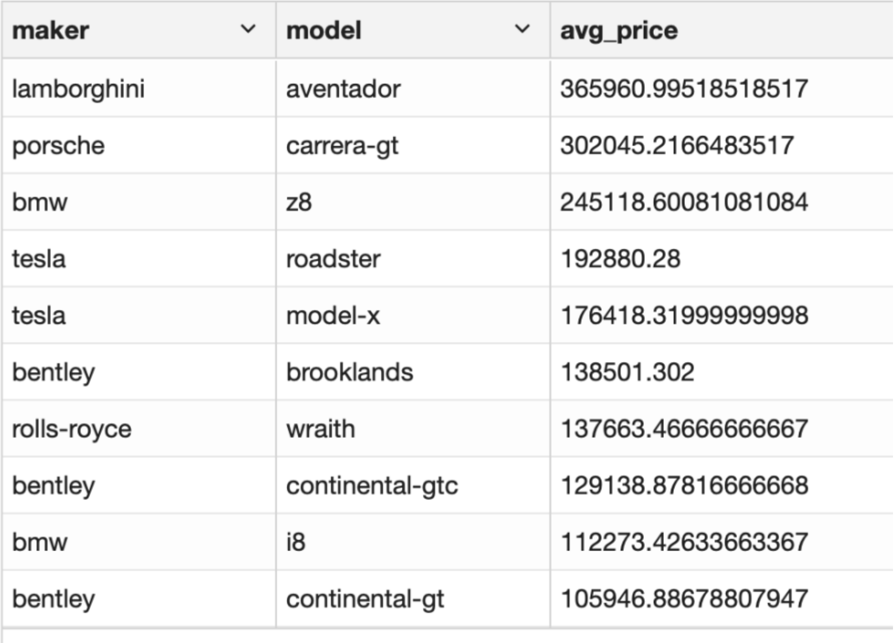
6. **Write a Hive query to find the make and model for the cars with the top 10 highest average prices.**

Hive query to determine the top make and model with the 10 highest average prices



```
%hive
USE used_cars_db;

SELECT maker, model, AVG(price_eur) AS avg_price
FROM clean_used_cars
GROUP BY maker, model
ORDER BY avg_price DESC
LIMIT 10;
```

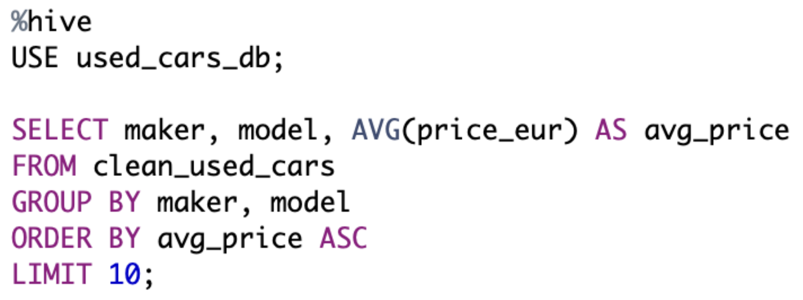
Query output



maker	model	avg_price
lamborghini	aventador	365960.99518518517
porsche	carrera-gt	302045.2166483517
bmw	z8	245118.60081081084
tesla	roadster	192880.28
tesla	model-x	176418.31999999998
bentley	brooklands	138501.302
rolls-royce	wraith	137663.46666666667
bentley	continental-gtc	129138.87816666668
bmw	i8	112273.42633663367
bentley	continental-gt	105946.88678807947

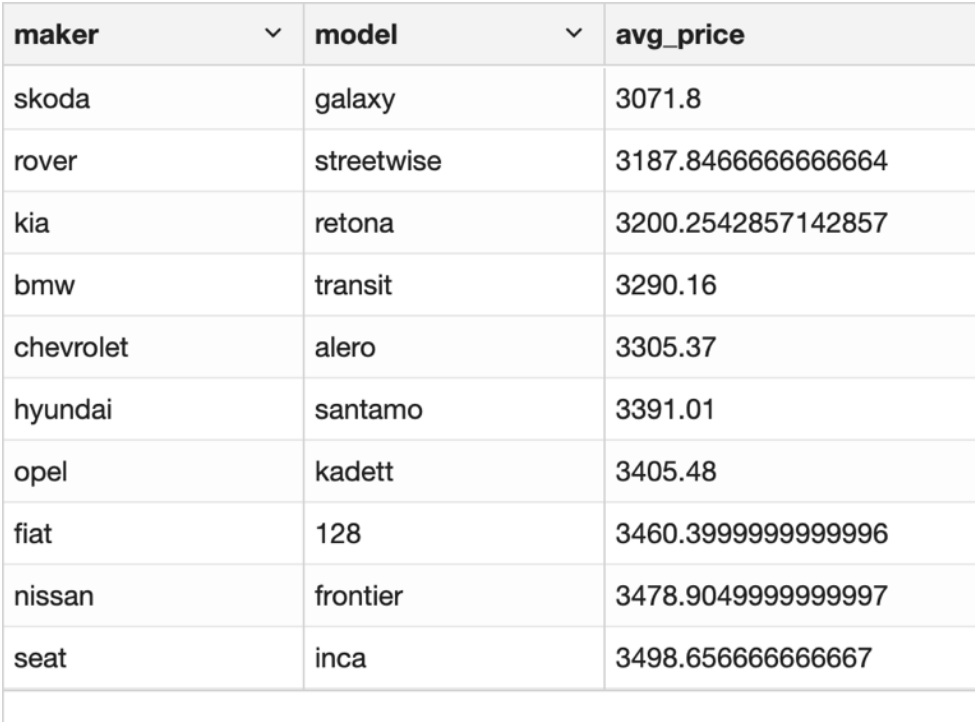
7. **Write a Hive query to find the make and model for the cars with the top 10 lowest average prices.**

Hive query



```
%hive
USE used_cars_db;

SELECT maker, model, AVG(price_eur) AS avg_price
FROM clean_used_cars
GROUP BY maker, model
ORDER BY avg_price ASC
LIMIT 10;
```

Query output

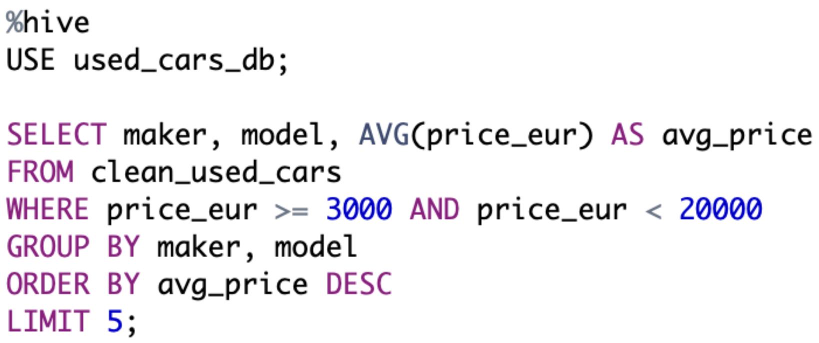


maker	model	avg_price
skoda	galaxy	3071.8
rover	streetwise	3187.846666666664
kia	retona	3200.2542857142857
bmw	transit	3290.16
chevrolet	alero	3305.37
hyundai	santamo	3391.01
opel	kadett	3405.48
fiat	128	3460.3999999999996
nissan	frontier	3478.9049999999997
seat	inca	3498.656666666667

8. **Write a Hive query to recommend the top five make and models for Economic Segment customers.**

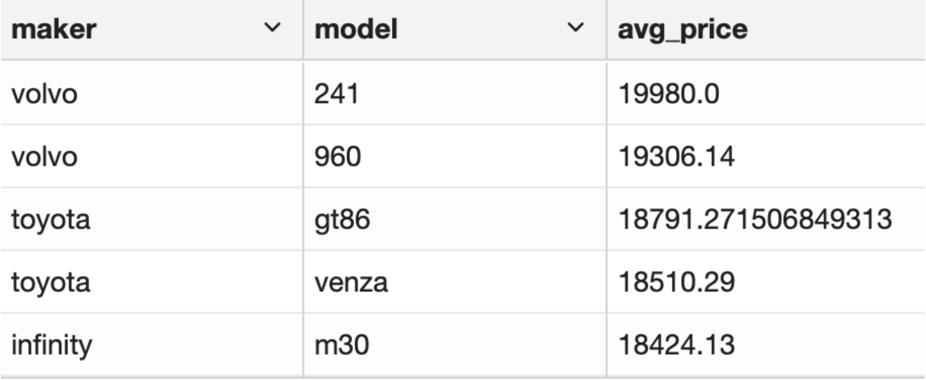
Economic Segment: Top five manufacturers in the 3000 to 20,000 price range; 3000 ≤ price < 20,000 based on the top average price.

Hive query to determine recommendation for the economic segment customers



```
%hive
USE used_cars_db;

SELECT maker, model, AVG(price_eur) AS avg_price
FROM clean_used_cars
WHERE price_eur >= 3000 AND price_eur < 20000
GROUP BY maker, model
ORDER BY avg_price DESC
LIMIT 5;
```

Query output

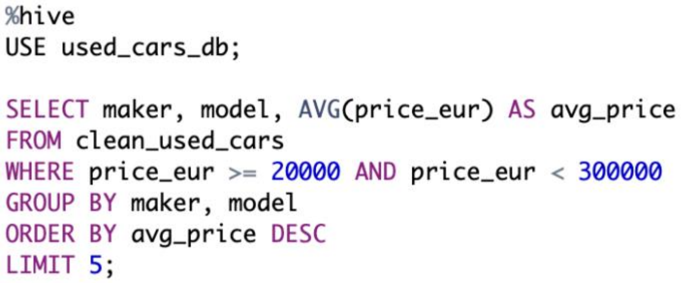


maker	model	avg_price
volvo	241	19980.0
volvo	960	19306.14
toyota	gt86	18791.271506849313
toyota	venza	18510.29
infinity	m30	18424.13

9. **Write a Hive query to recommend the top five make and models for Intermediate Segment customers**

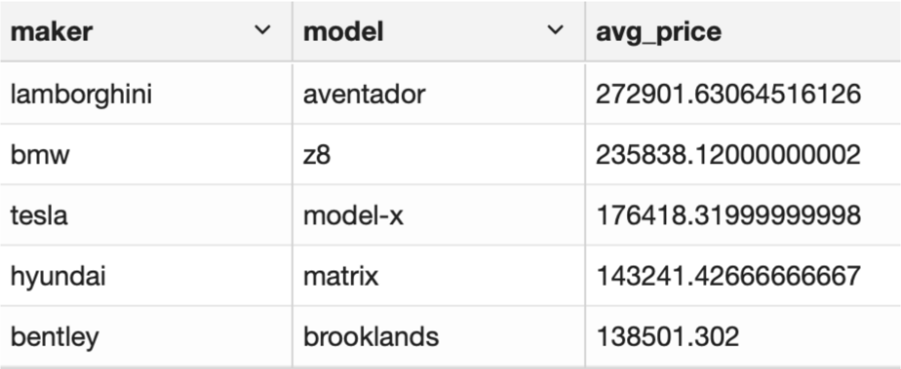
Intermediate Segment: Top five manufacturers in the 20,000 to 300,000 price range; 20,000 ≤ price < 300,000 based on the top average price.

Hive query to determine recommendation for the intermediate segment customers



```
%hive
USE used_cars_db;

SELECT maker, model, AVG(price_eur) AS avg_price
FROM clean_used_cars
WHERE price_eur >= 20000 AND price_eur < 300000
GROUP BY maker, model
ORDER BY avg_price DESC
LIMIT 5;
```

Query output

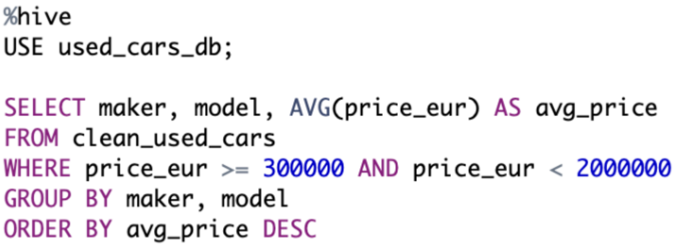


maker	model	avg_price
lamborghini	aventador	272901.63064516126
bmw	z8	235838.12000000002
tesla	model-x	176418.31999999998
hyundai	matrix	143241.42666666667
bentley	brooklands	138501.302

10. Write a Hive query to recommend the top five make and models for the Luxury Segment customers

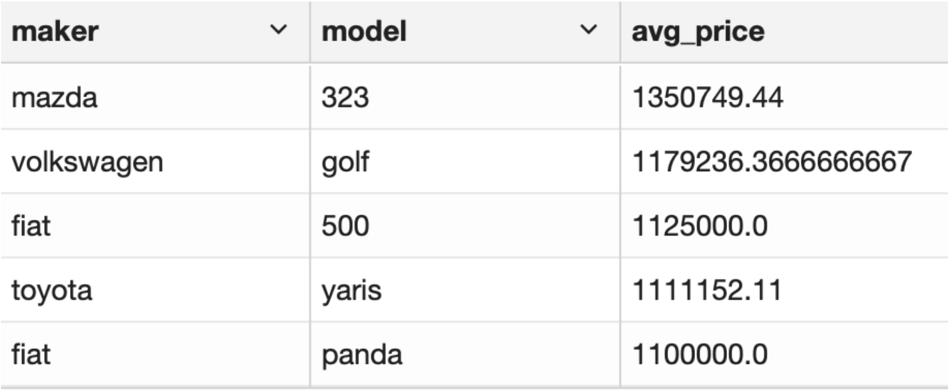
Luxury Segment: Top five manufacturers in the 300,000 to 2,000,000 price range; 300,000 ≤ price < 2,000,000 based on the top average price.

Hive query to determine recommendation for the luxury segment customers



```
%hive
USE used_cars_db;

SELECT maker, model, AVG(price_eur) AS avg_price
FROM clean_used_cars
WHERE price_eur >= 300000 AND price_eur < 2000000
GROUP BY maker, model
ORDER BY avg_price DESC
```

Query output

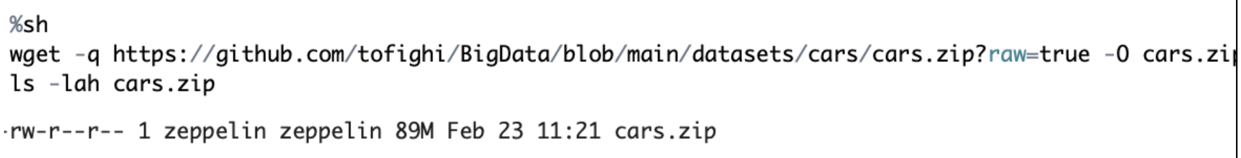


maker	model	avg_price
mazda	323	1350749.44
volkswagen	golf	1179236.3666666667
fiat	500	1125000.0
toyota	yaris	1111152.11
fiat	panda	1100000.0

Deliverable Part 2 - SparkSQL Tasks

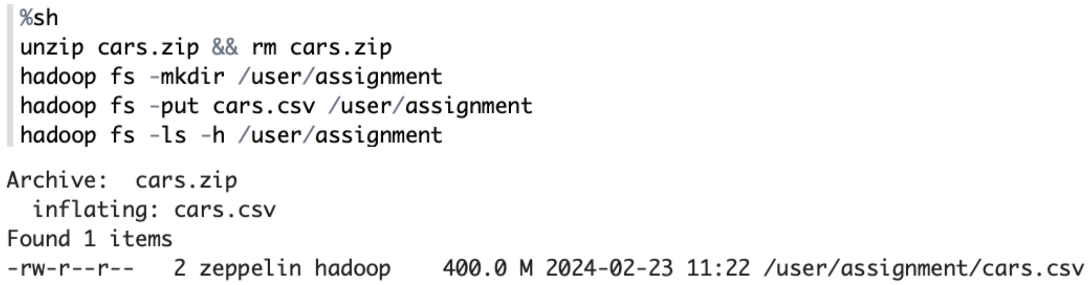
Pre-Setup

Download the dataset from github



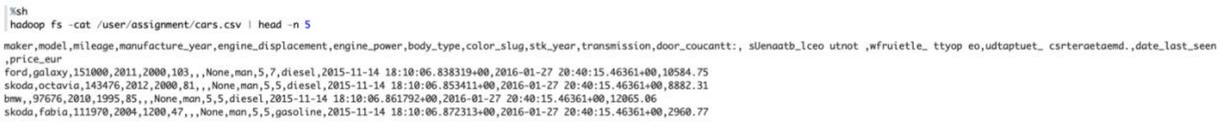
```
%sh
wget -q https://github.com/tofighi/BigData/blob/main/datasets/cars/cars.zip?raw=true -O cars.zi
ls -lah cars.zip

-rw-r--r-- 1 zeppelin zeppelin 89M Feb 23 11:21 cars.zip
```

Unzip and move to hadoop



```
%sh
unzip cars.zip && rm cars.zip
hadoop fs -mkdir /user/assignment
hadoop fs -put cars.csv /user/assignment
hadoop fs -ls -h /user/assignment

Archive:  cars.zip
  inflating: cars.csv
Found 1 items
-rw-r--r--   2 zeppelin hadoop    400.0 M 2024-02-23 11:22 /user/assignment/cars.csv
```

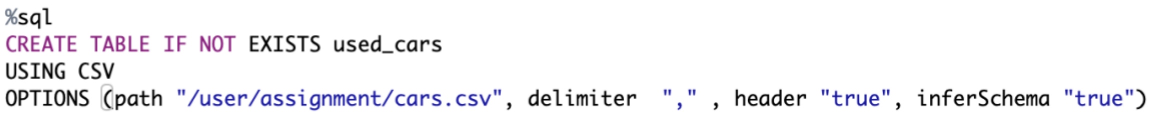
Verify the file



```
%sh
hadoop fs -cat /user/assignment/cars.csv | head -n 5

maker,model,mileage,manufacture_year,engine_displacement,engine_power,body_type,color_slug,stk_year,transmission,door_coucantt:, sUenaatb_lceo utnot ,wfruietle_ ttyop eo,udtaptuet_ csrteraetaemd.,date_last_seen
,price_eur
ford,galaxy,151000,2011,2000,103,,,None,man,5,7,diesel,2015-11-14 18:10:06.838319+00,2016-01-27 20:40:15.46361+00,10584.75
skoda,octavia,143476,2012,2000,81,,,None,man,5,5,diesel,2015-11-14 18:10:06.853411+00,2016-01-27 20:40:15.46361+00,8882.31
bmw,,97676,2010,1995,85,,,None,man,5,5,diesel,2015-11-14 18:10:06.861792+00,2016-01-27 20:40:15.46361+00,12065.06
skoda,fabia,111970,2004,1200,47,,,None,man,5,5,gasoline,2015-11-14 18:10:06.872313+00,2016-01-27 20:40:15.46361+00,2960.77
```

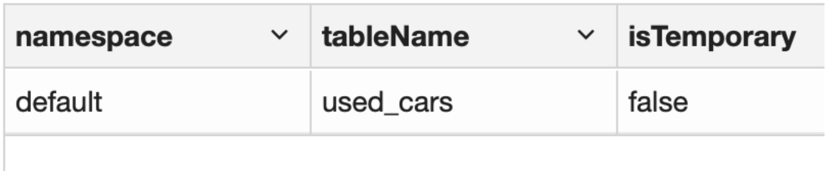
1. Write a SparkSQL query to create a table called used_cars from the data.

SparkSQL query to create a table for used_cars



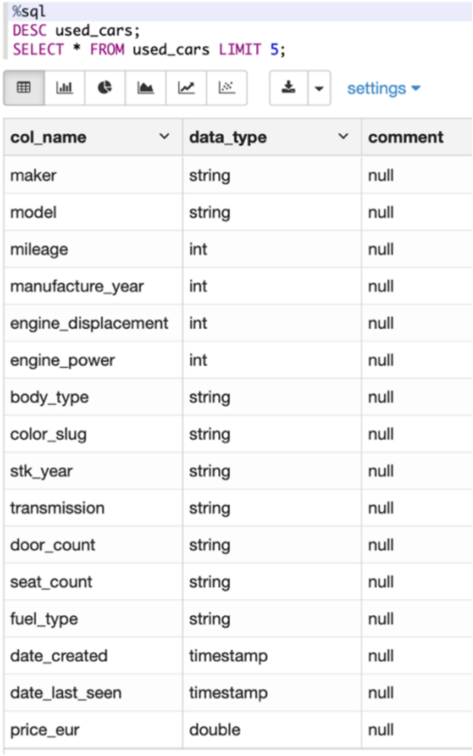
```
%sql
CREATE TABLE IF NOT EXISTS used_cars
USING CSV
OPTIONS (path "/user/assignment/cars.csv", delimiter  "," , header "true", inferSchema "true")
```

Verify table has been created



namespace	tableName	isTemporary
default	used_cars	false

Verify all the columns are correct

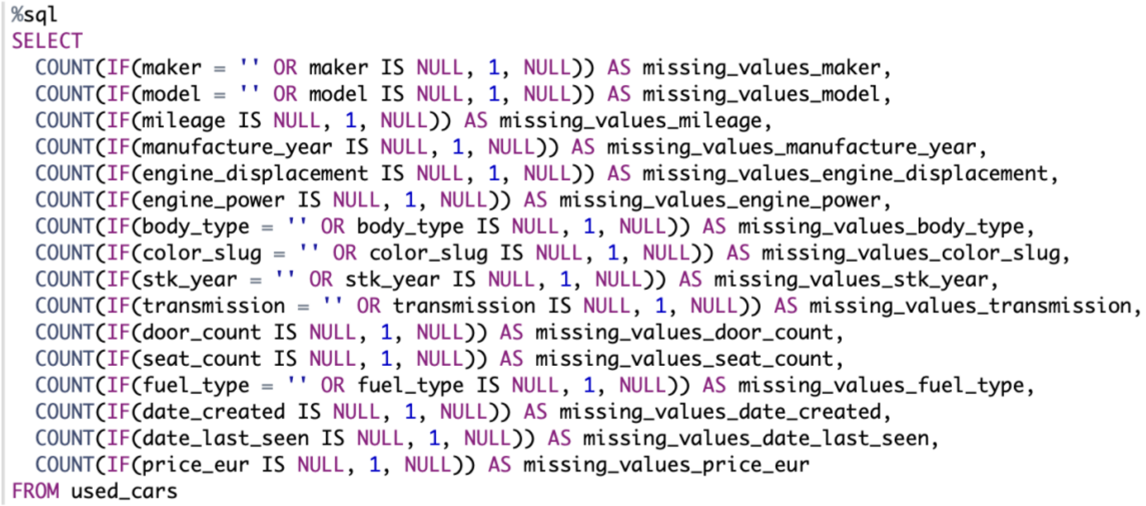


```
%sql
DESC used_cars;
SELECT * FROM used_cars LIMIT 5;
```

col_name	data_type	comment
maker	string	null
model	string	null
mileage	int	null
manufacture_year	int	null
engine_displacement	int	null
engine_power	int	null
body_type	string	null
color_slug	string	null
stk_year	string	null
transmission	string	null
door_count	string	null
seat_count	string	null
fuel_type	string	null
date_created	timestamp	null
date_last_seen	timestamp	null
price_eur	double	null

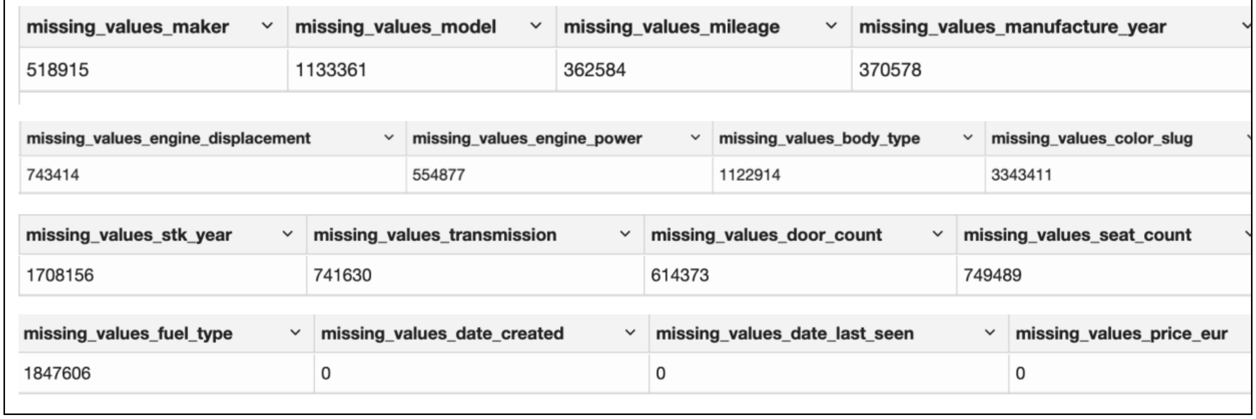
2. **Write several SparkSQL queries to find how many missing values (NULL in an attribute of a record) you have in each column (attribute).**

SparkSQL query to check if it contains NULL or an empty string (‘’)



```
%sql
SELECT
  COUNT(IF(maker = '' OR maker IS NULL, 1, NULL)) AS missing_values_maker,
  COUNT(IF(model = '' OR model IS NULL, 1, NULL)) AS missing_values_model,
  COUNT(IF(mileage IS NULL, 1, NULL)) AS missing_values_mileage,
  COUNT(IF(manufacture_year IS NULL, 1, NULL)) AS missing_values_manufacture_year,
  COUNT(IF(engine_displacement IS NULL, 1, NULL)) AS missing_values_engine_displacement,
  COUNT(IF(engine_power IS NULL, 1, NULL)) AS missing_values_engine_power,
  COUNT(IF(body_type = '' OR body_type IS NULL, 1, NULL)) AS missing_values_body_type,
  COUNT(IF(color_slug = '' OR color_slug IS NULL, 1, NULL)) AS missing_values_color_slug,
  COUNT(IF(stk_year = '' OR stk_year IS NULL, 1, NULL)) AS missing_values_stk_year,
  COUNT(IF(transmission = '' OR transmission IS NULL, 1, NULL)) AS missing_values_transmission,
  COUNT(IF(door_count IS NULL, 1, NULL)) AS missing_values_door_count,
  COUNT(IF(seat_count IS NULL, 1, NULL)) AS missing_values_seat_count,
  COUNT(IF(fuel_type = '' OR fuel_type IS NULL, 1, NULL)) AS missing_values_fuel_type,
  COUNT(IF(date_created IS NULL, 1, NULL)) AS missing_values_date_created,
  COUNT(IF(date_last_seen IS NULL, 1, NULL)) AS missing_values_date_last_seen,
  COUNT(IF(price_eur IS NULL, 1, NULL)) AS missing_values_price_eur
FROM used_cars
```

Output of the query



missing_values_maker	missing_values_model	missing_values_mileage	missing_values_manufacture_year
518915	1133361	362584	370578

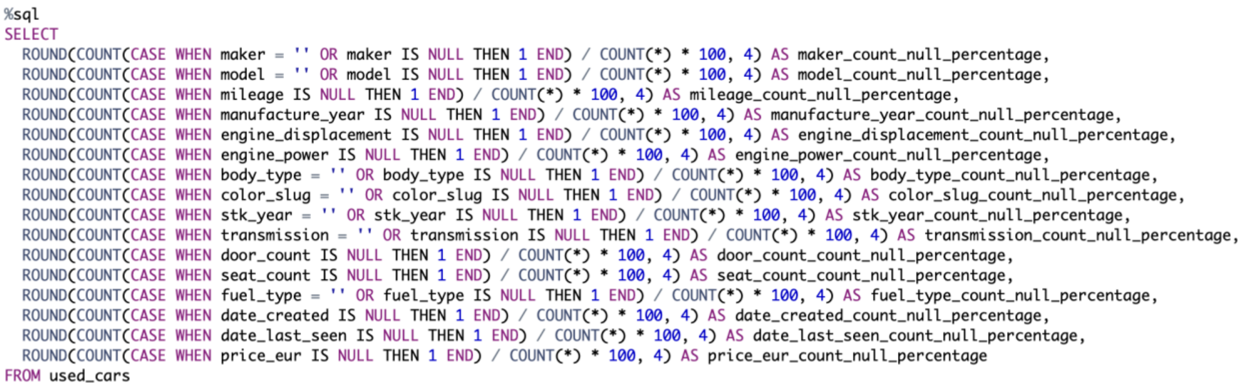
missing_values_engine_displacement	missing_values_engine_power	missing_values_body_type	missing_values_color_slug
743414	554877	1122914	3343411

missing_values_stk_year	missing_values_transmission	missing_values_door_count	missing_values_seat_count
1708156	741630	614373	749489

missing_values_fuel_type	missing_values_date_created	missing_values_date_last_seen	missing_values_price_eur
1847606	0	0	0

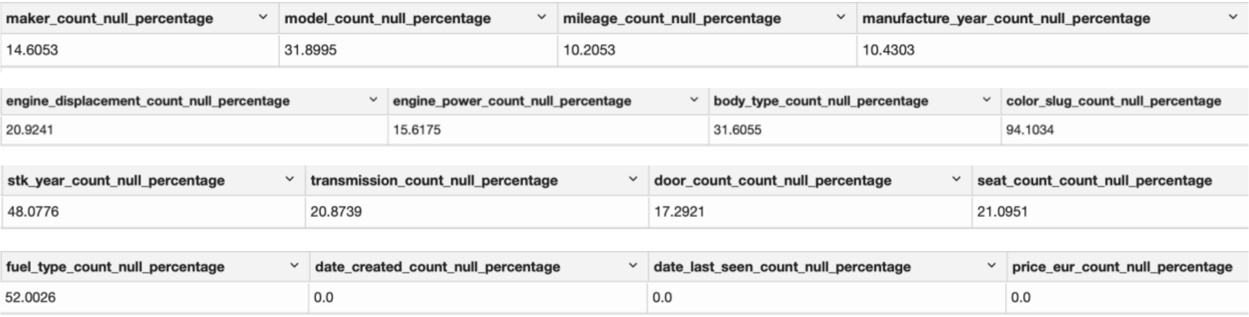
3. Drop the columns (attribute) with more than 50% missing values.

Query to calculate the columns with 50% or more missing values



```
%sql
SELECT
  ROUND(COUNT(CASE WHEN maker = '' OR maker IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS maker_count_null_percentage,
  ROUND(COUNT(CASE WHEN model = '' OR model IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS model_count_null_percentage,
  ROUND(COUNT(CASE WHEN mileage IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS mileage_count_null_percentage,
  ROUND(COUNT(CASE WHEN manufacture_year IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS manufacture_year_count_null_percentage,
  ROUND(COUNT(CASE WHEN engine_displacement IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS engine_displacement_count_null_percentage,
  ROUND(COUNT(CASE WHEN engine_power IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS engine_power_count_null_percentage,
  ROUND(COUNT(CASE WHEN body_type = '' OR body_type IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS body_type_count_null_percentage,
  ROUND(COUNT(CASE WHEN color_slug = '' OR color_slug IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS color_slug_count_null_percentage,
  ROUND(COUNT(CASE WHEN stk_year = '' OR stk_year IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS stk_year_count_null_percentage,
  ROUND(COUNT(CASE WHEN transmission = '' OR transmission IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS transmission_count_null_percentage,
  ROUND(COUNT(CASE WHEN door_count IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS door_count_count_null_percentage,
  ROUND(COUNT(CASE WHEN seat_count IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS seat_count_count_null_percentage,
  ROUND(COUNT(CASE WHEN fuel_type = '' OR fuel_type IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS fuel_type_count_null_percentage,
  ROUND(COUNT(CASE WHEN date_created IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS date_created_count_null_percentage,
  ROUND(COUNT(CASE WHEN date_last_seen IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS date_last_seen_count_null_percentage,
  ROUND(COUNT(CASE WHEN price_eur IS NULL THEN 1 END) / COUNT(*) * 100, 4) AS price_eur_count_null_percentage
FROM used_cars
```

Output of the query



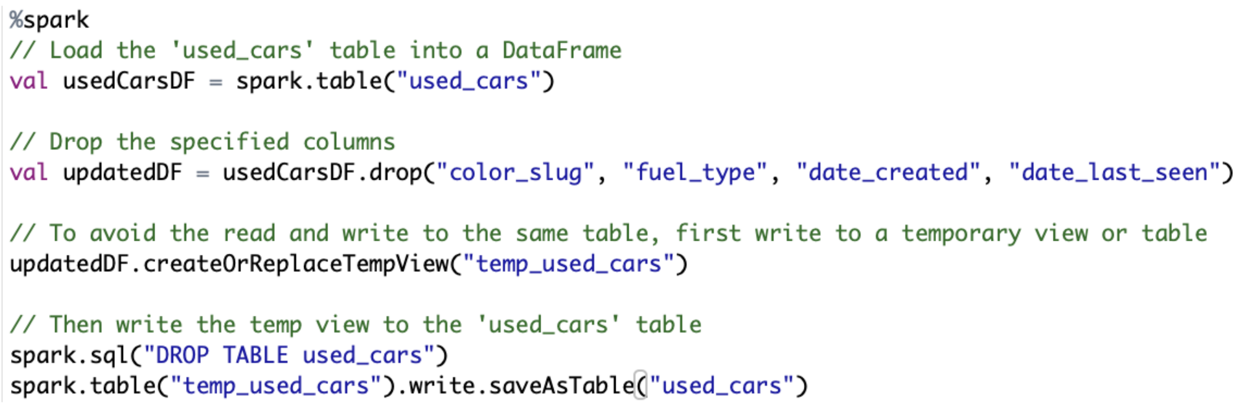
maker_count_null_percentage	model_count_null_percentage	mileage_count_null_percentage	manufacture_year_count_null_percentage
14.6053	31.8995	10.2053	10.4303

engine_displacement_count_null_percentage	engine_power_count_null_percentage	body_type_count_null_percentage	color_slug_count_null_percentage
20.9241	15.6175	31.6055	94.1034

stk_year_count_null_percentage	transmission_count_null_percentage	door_count_count_null_percentage	seat_count_count_null_percentage
48.0776	20.8739	17.2921	21.0951

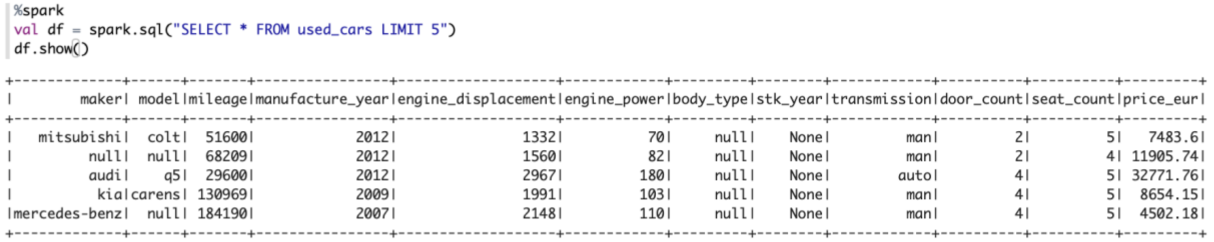
fuel_type_count_null_percentage	date_created_count_null_percentage	date_last_seen_count_null_percentage	price_eur_count_null_percentage
52.0026	0.0	0.0	0.0

Drop the following columns: color_slug(94%), fuel_type(52%), date_created(100%), date_last_seen(100%).
Note: Used a dateframe approach



```
%spark
// Load the 'used_cars' table into a DataFrame
val usedCarsDF = spark.table("used_cars")

// Drop the specified columns
val updatedDF = usedCarsDF.drop("color_slug", "fuel_type", "date_created", "date_last_seen")

// To avoid the read and write to the same table, first write to a temporary view or table
updatedDF.createOrReplaceTempView("temp_used_cars")

// Then write the temp view to the 'used_cars' table
spark.sql("DROP TABLE used_cars")
spark.table("temp_used_cars").write.saveAsTable("used_cars")
```

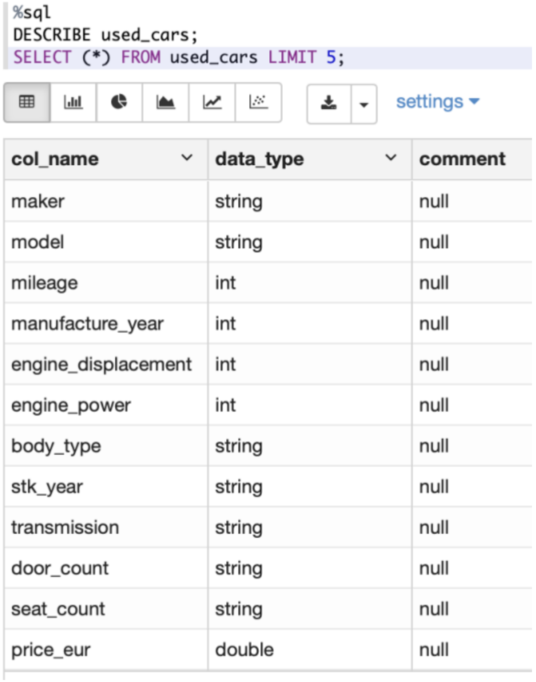
Verify the dataframe



```
%spark
val df = spark.sql("SELECT * FROM used_cars LIMIT 5")
df.show()
```

```
+-------------+------+-------+----------------+-------------------+------------+---------+--------+------------+----------+----------+---------+
|        maker| model|mileage|manufacture_year|engine_displacement|engine_power|body_type|stk_year|transmission|door_count|seat_count|price_eur|
+-------------+------+-------+----------------+-------------------+------------+---------+--------+------------+----------+----------+---------+
|   mitsubishi|  colt|  51600|            2012|               1332|          70|     null|    None|         man|         2|         5|   7483.6|
|         null|  null|  68209|            2012|               1560|          82|     null|    None|         man|         2|         4| 11905.74|
|         audi|    q5|  29600|            2012|               2967|         180|     null|    None|        auto|         4|         5| 32771.76|
|          kia|carens| 130969|            2009|               1991|         103|     null|    None|         man|         4|         5|  8654.15|
|mercedes-benz|  null| 184190|            2007|               2148|         110|     null|    None|         man|         4|         5|  4502.18|
+-------------+------+-------+----------------+-------------------+------------+---------+--------+------------+----------+----------+---------+
```

Verify the table has been updated



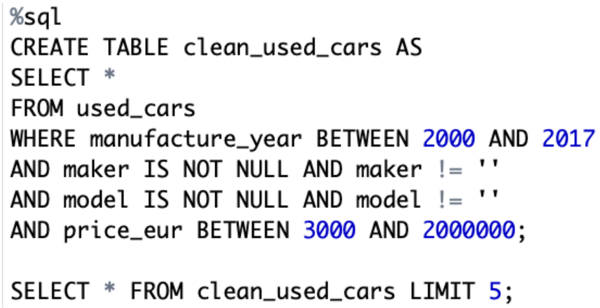
```
%sql
DESCRIBE used_cars;
SELECT (*) FROM used_cars LIMIT 5;
```

col_name	data_type	comment
maker	string	null
model	string	null
mileage	int	null
manufacture_year	int	null
engine_displacement	int	null
engine_power	int	null
body_type	string	null
stk_year	string	null
transmission	string	null
door_count	string	null
seat_count	string	null
price_eur	double	null

4. Write several SparkSQL queries to create a new table called clean_used_cars from used_cars with the following conditions:

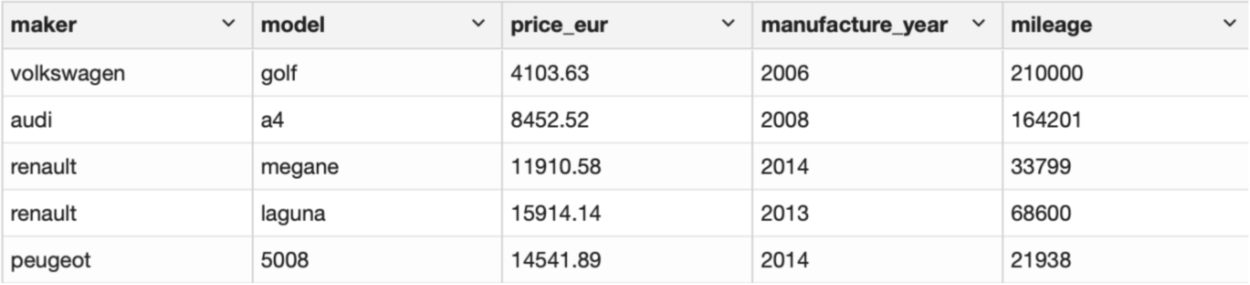
- *The manufacturing year between 2000 and 2017 including 2000 and 2017*
- *Both maker and model exist (NOT NULL) in the row*
- *The price range is from 3000 to 2,000,000 (3000 ≤ price ≤ 2,000,000)*

SparkSQL query to create a new table with the given conditions



```
%sql
CREATE TABLE clean_used_cars AS
SELECT *
FROM used_cars
WHERE manufacture_year BETWEEN 2000 AND 2017
AND maker IS NOT NULL AND maker != ''
AND model IS NOT NULL AND model != ''
AND price_eur BETWEEN 3000 AND 2000000;

SELECT * FROM clean_used_cars LIMIT 5;
```

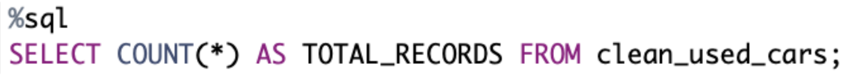
Verify output



maker	model	price_eur	manufacture_year	mileage
volkswagen	golf	4103.63	2006	210000
audi	a4	8452.52	2008	164201
renault	megane	11910.58	2014	33799
renault	laguna	15914.14	2013	68600
peugeot	5008	14541.89	2014	21938

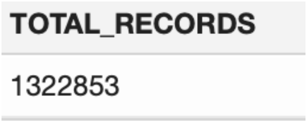
5. Write SparkSQL to find how many records remained clean_used_cars.

SparkSQL query to determine the total records in the clean_used_cars



```
%sql
SELECT COUNT(*) AS TOTAL_RECORDS FROM clean_used_cars;
```

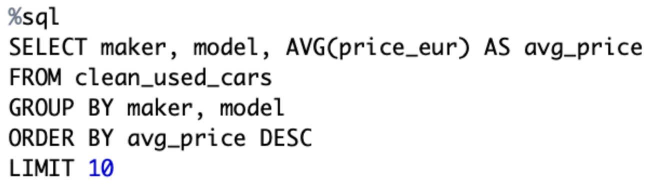
Query output



TOTAL_RECORDS
1322853

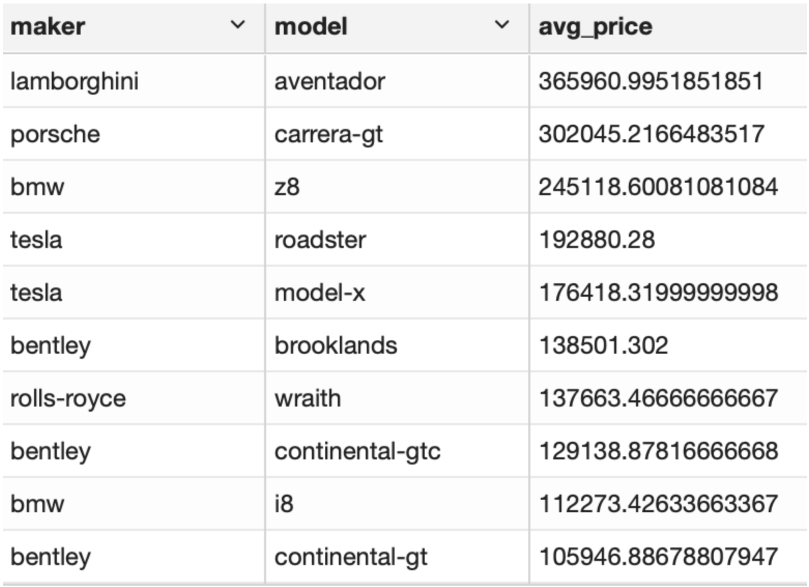
6. **Write a SparkSQL query to find the make and model for the cars with the top 10 highest average prices.**

SparkSQL query to determine the top make and model with the 10 highest average prices



```
%sql
SELECT maker, model, AVG(price_eur) AS avg_price
FROM clean_used_cars
GROUP BY maker, model
ORDER BY avg_price DESC
LIMIT 10
```

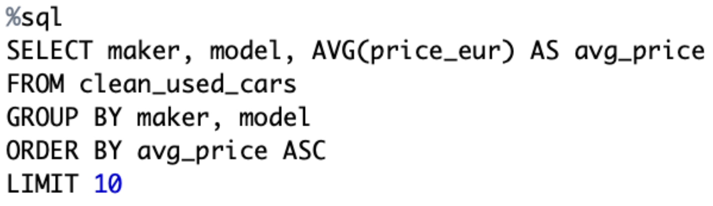
Query output



maker	model	avg_price
lamborghini	aventador	365960.9951851851
porsche	carrera-gt	302045.2166483517
bmw	z8	245118.60081081084
tesla	roadster	192880.28
tesla	model-x	176418.31999999998
bentley	brooklands	138501.302
rolls-royce	wraith	137663.46666666667
bentley	continental-gtc	129138.87816666668
bmw	i8	112273.42633663367
bentley	continental-gt	105946.88678807947

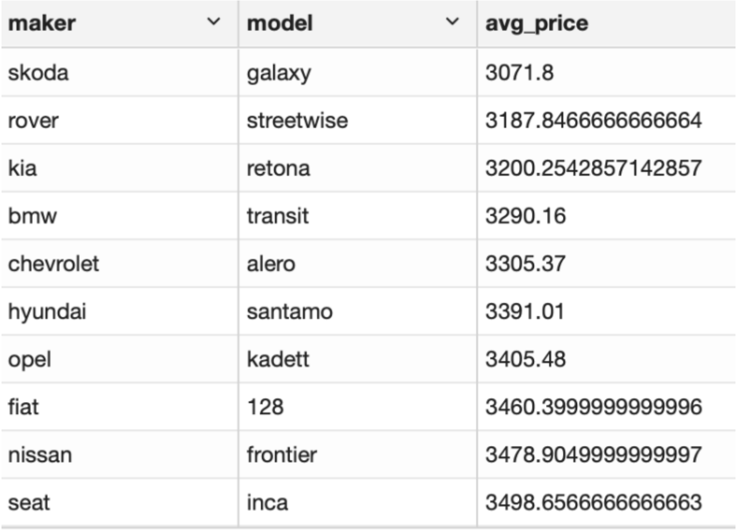
7. **Write a SparkSQL query to find the make and model for the cars with the top 10 lowest average prices.**

SparkSQL query



```
%sql
SELECT maker, model, AVG(price_eur) AS avg_price
FROM clean_used_cars
GROUP BY maker, model
ORDER BY avg_price ASC
LIMIT 10
```

Query output

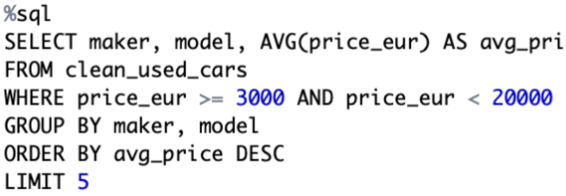


maker	model	avg_price
skoda	galaxy	3071.8
rover	streetwise	3187.8466666666664
kia	retona	3200.2542857142857
bmw	transit	3290.16
chevrolet	alero	3305.37
hyundai	santamo	3391.01
opel	kadett	3405.48
fiat	128	3460.3999999999996
nissan	frontier	3478.9049999999997
seat	inca	3498.6566666666663

8. Write a SparkSQL query to recommend the top five make and models for Economic Segment customers.

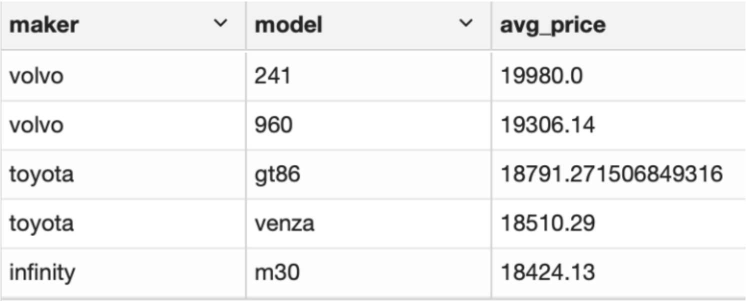
Economic Segment: Top five manufacturers in the 3000 to 20,000 price range; 3000 ≤ price < 20,000 based on the top average price.

SparkSQL query to determine recommendation for the economic segment customers



```
%sql
SELECT maker, model, AVG(price_eur) AS avg_price
FROM clean_used_cars
WHERE price_eur >= 3000 AND price_eur < 20000
GROUP BY maker, model
ORDER BY avg_price DESC
LIMIT 5
```

Query output

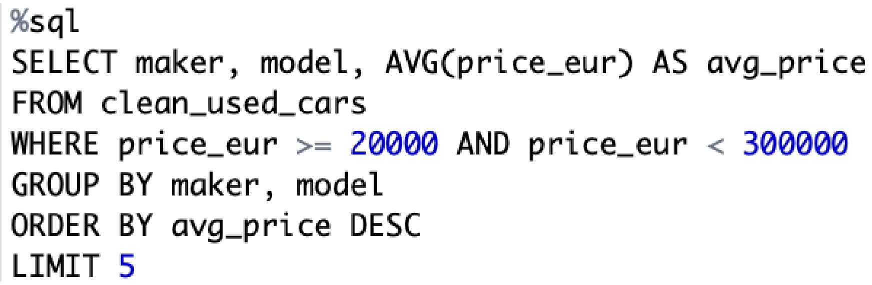


maker	model	avg_price
volvo	241	19980.0
volvo	960	19306.14
toyota	gt86	18791.271506849316
toyota	venza	18510.29
infinity	m30	18424.13

9. Write a SparkSQL query to recommend the top five make and models for Intermediate Segment customers

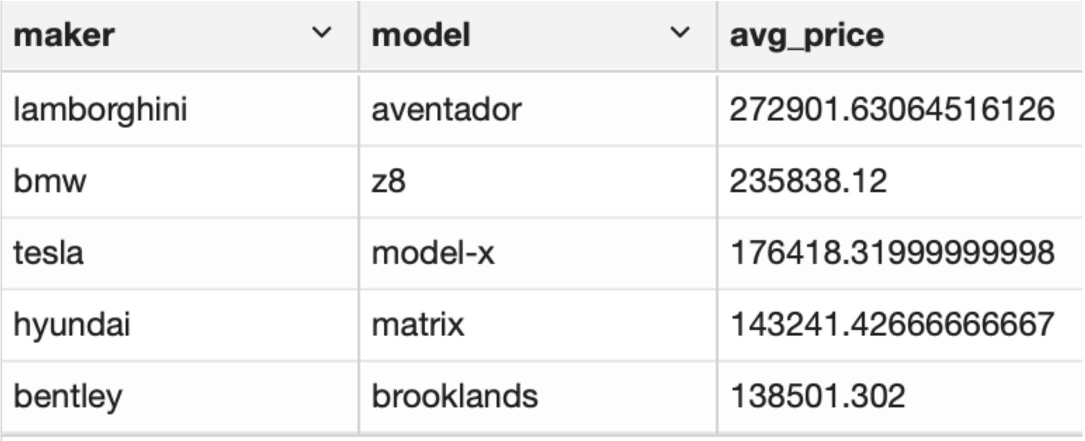
Intermediate Segment: Top five manufacturers in the 20,000 to 300,000 price range; 20,000 ≤ price < 300,000 based on the top average price.

SparkSQL query to determine recommendation for the intermediate segment customers



```
%sql
SELECT maker, model, AVG(price_eur) AS avg_price
FROM clean_used_cars
WHERE price_eur >= 20000 AND price_eur < 300000
GROUP BY maker, model
ORDER BY avg_price DESC
LIMIT 5
```

Query output

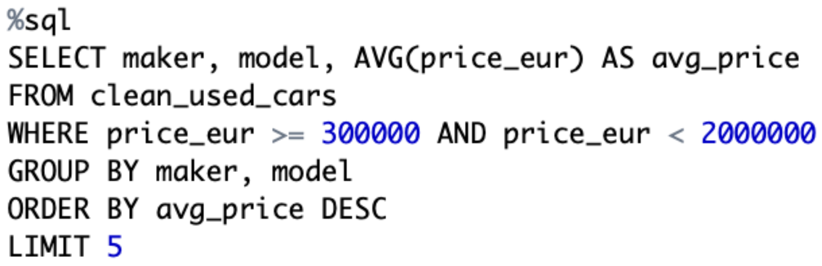


maker	model	avg_price
lamborghini	aventador	272901.63064516126
bmw	z8	235838.12
tesla	model-x	176418.31999999998
hyundai	matrix	143241.42666666667
bentley	brooklands	138501.302

10. Write a SparkSQL query to recommend the top five make and models for the Luxury Segment customers

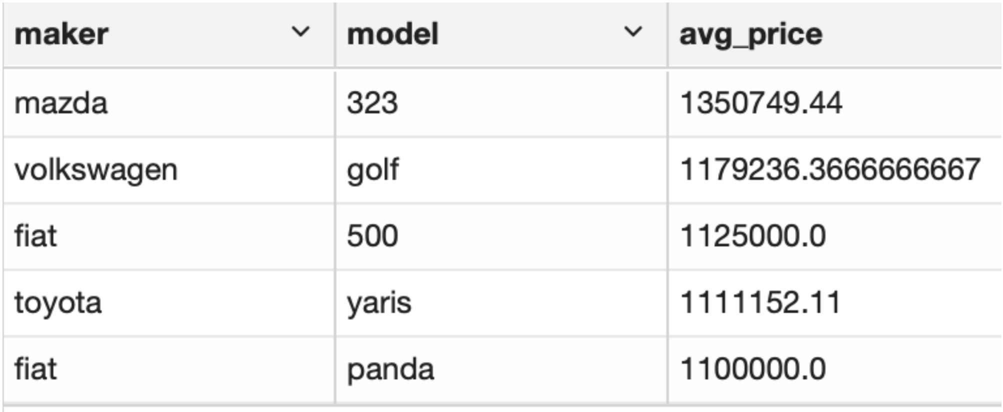
Luxury Segment: Top five manufacturers in the 300,000 to 2,000,000 price range; 300,000 ≤ price < 2,000,000 based on the top average price.

SparkSQL query to determine recommendation for the luxury segment customers



```
%sql
SELECT maker, model, AVG(price_eur) AS avg_price
FROM clean_used_cars
WHERE price_eur >= 300000 AND price_eur < 2000000
GROUP BY maker, model
ORDER BY avg_price DESC
LIMIT 5
```

Query output



maker	model	avg_price
mazda	323	1350749.44
volkswagen	golf	1179236.3666666667
fiat	500	1125000.0
toyota	yaris	1111152.11
fiat	panda	1100000.0

Discussion

SparkSQL queries compared to the HQL had the same results, but the only difference was computation time. The SparkSQL was computed in much less time even though various techniques were applied and were not similar in both methods.

Summary Report

1. **Number of records in used_cars table**

3,552,913

2. **Number of records in clean_used_cars table**

1,322,853

3. **Make and model for the cars with the top 10 highest average prices and their average price**

Maker	Model	Average Price
lamborghini	aventador	365960.9951851850
porsche	carrera-gt	302045.2166483520
bmw	z8	245118.60081081100
tesla	roadster	192880.28
tesla	model-x	176418.32000000000
bentley	brooklands	138501.302
rolls-royce	wraith	137663.46666666700
bentley	continental-gtc	129138.87816666700
bmw	i8	112273.42633663400
bentley	continental-gt	105946.88678807900

4. **Make and model for the cars with the top 10 lowest average prices and their average price**

Maker	Model	Average Price
skoda	galaxy	3071.8
rover	streetwise	3187.8466666666700
kia	retona	3200.2542857142900
bmw	transit	3290.16
chevrolet	alero	3305.37
hyundai	santamo	3391.01
opel	kadett	3405.48
fiat	128	3460.4000000000000
nissan	frontier	3478.9050000000000
seat	inca	3498.6566666666700

5. **Write the name of the top five makes and models for Economic segment customers**

Maker	Model	Average Price
volvo	241	19980.0
volvo	960	19306.14
toyota	gt86	18791.271506849300
toyota	venza	18510.29
infinity	m30	18424.13

6. **Write the name of the top five makes and models for Intermediate segment customers**

Maker	Model	Average Price
lamborghini	aventador	272901.63064516100
bmw	z8	235838.12
tesla	model-x	176418.32000000000
hyundai	matrix	143241.42666666700
bentley	brooklands	138501.302

7. **Write the name of the top five makes and models for Luxury segment customers**

Maker	Model	Average Price
mazda	323	1350749.44
volkswagen	golf	1179236.3666666700
fiat	500	1125000.0
toyota	yaris	1111152.11
fiat	panda	1100000.0