Karishma Shah

Analysis on used car dataset
Apache Spark

Problem Background:

A large investment firm that is contemplating to invest in a used car business and task is to provide data driven advice to the stakeholders, that will enable them to make a sound investment decision.

Abstract

The classified ads for cars dataset extracted from Kaggle includes information on a variety of used automobile sales from across Germany and the Czech Republic since 2015. It includes large amount of data on various brands, models, mileage counts of those cars, duration of the advertisement posts and pricing. Along with that, data also features statistics on fuel type as well as transmission type. In addition, it also provides insight in number of doors and numbers of seat for each advertisement posted. This project aims to clean this data for better analysis and provide data-driven recommendation for potential investment in used car business.

Objectives

The aim of this analysis is to provide useful insights on the market situation for used car business in Germany and Czech Republic. It aims to examine any potential relationships in the following:

- Overall counts for Car Maker/Model
- Maker/Model Vs Mileage
- Maker/Model vs Price
- Maker/Model vs Date Posted vs Last seen

Data

The data was scraped from several websites in Czech Republic and Germany over a period of more than a year.

There are roughly 3,5 Million rows and the following columns:

- maker - normalized all lowercase
- model - normalized all lowercase
- mileage - in KM
- manufacture_year
- engine_displacement - in ccm
- engine_power - in kW
- body_type - almost never present, but I scraped only personal cars, no motorcycles or utility vehicles
- color_slug - also almost never present

- stk_year - year of the last emission control
- transmission - automatic or manual
- door_count
- seat_count
- fuel_type - gasoline, diesel, cng, lpg, electric
- date_created - when the ad was scraped
- datelastseen - when the ad was last seen. Our policy was to remove all ads older than 60 days
- price_eur - list price converted to EUR

Preliminary Analysis Result

1. **Count total no. of cars for selected manufacture_year.**

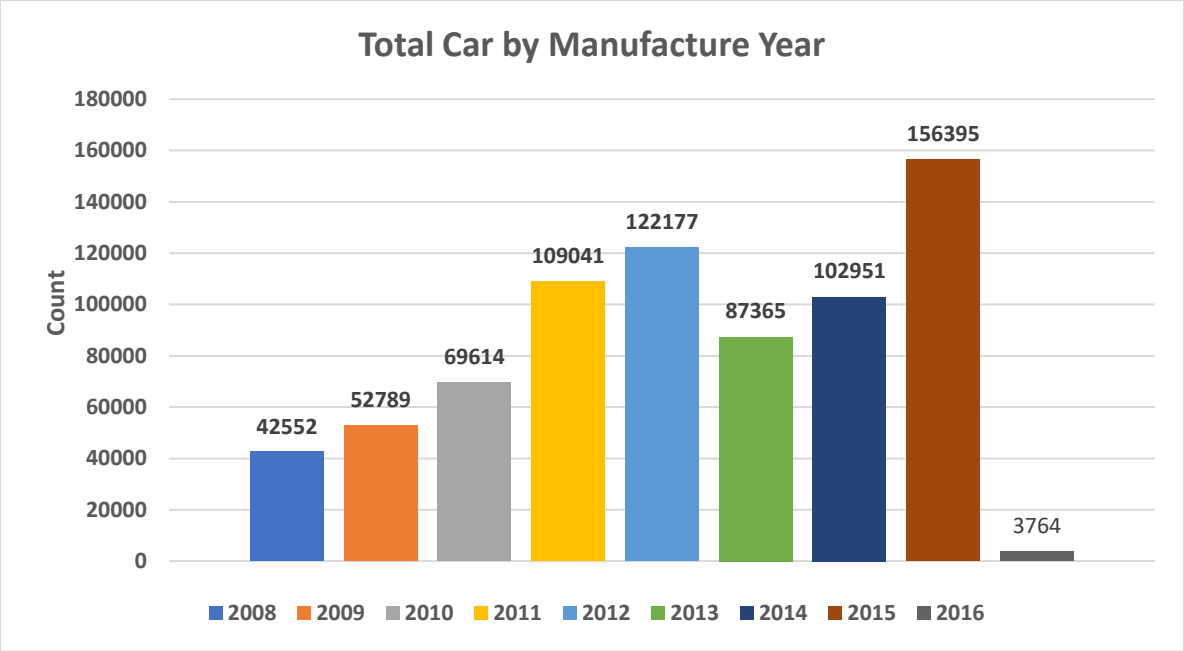

Figure:1 Total no. of cars available for selected manufacture_year

2. Count total cars for selected seat_count.

seat_count	count
5	450478
4	76640
7	31326
2	17471
3	7339
9	3215
6	2189
8	1141

Table:1 total car by seat_count

3. Count total cars for fuel_type columns.

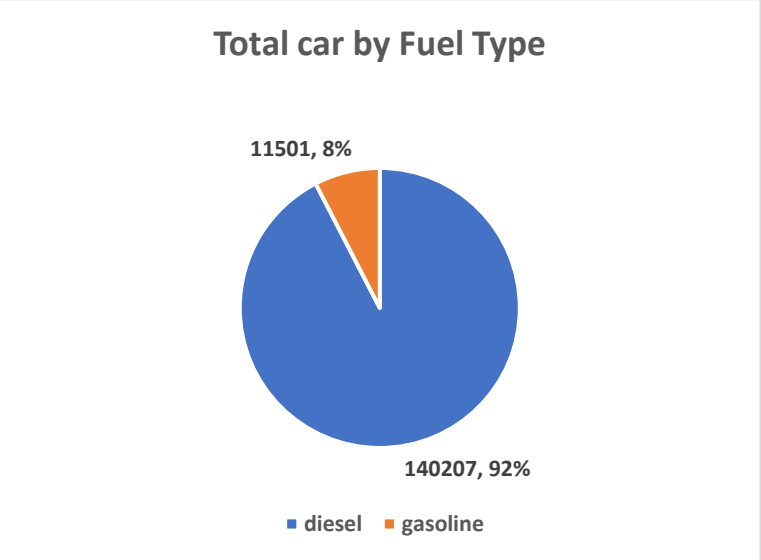

Figure:2 Total car by Fuel Type

4. Count total no. of entries for each transmission type.

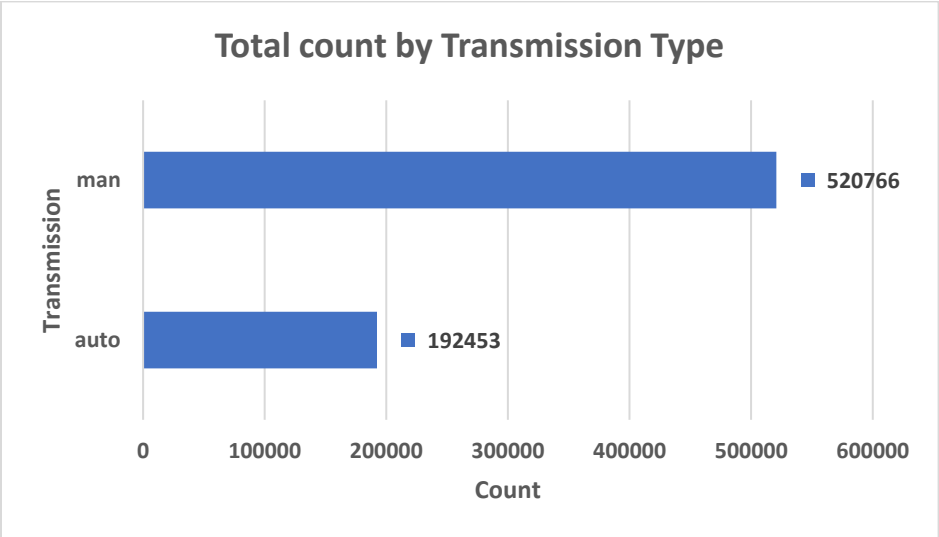

Figure 3: Total car by Transmission type

The above charts and tables show the most popular results for fuel type, transmission type and seat count.

- More car for Manufacture year 2011 to 2015.
- More cars use diesel than gasoline.
- Most vehicles sold have manual transmission.
- Cars with 5-seats are most common.

Analysis Result

1. Calculate total entries by each maker.

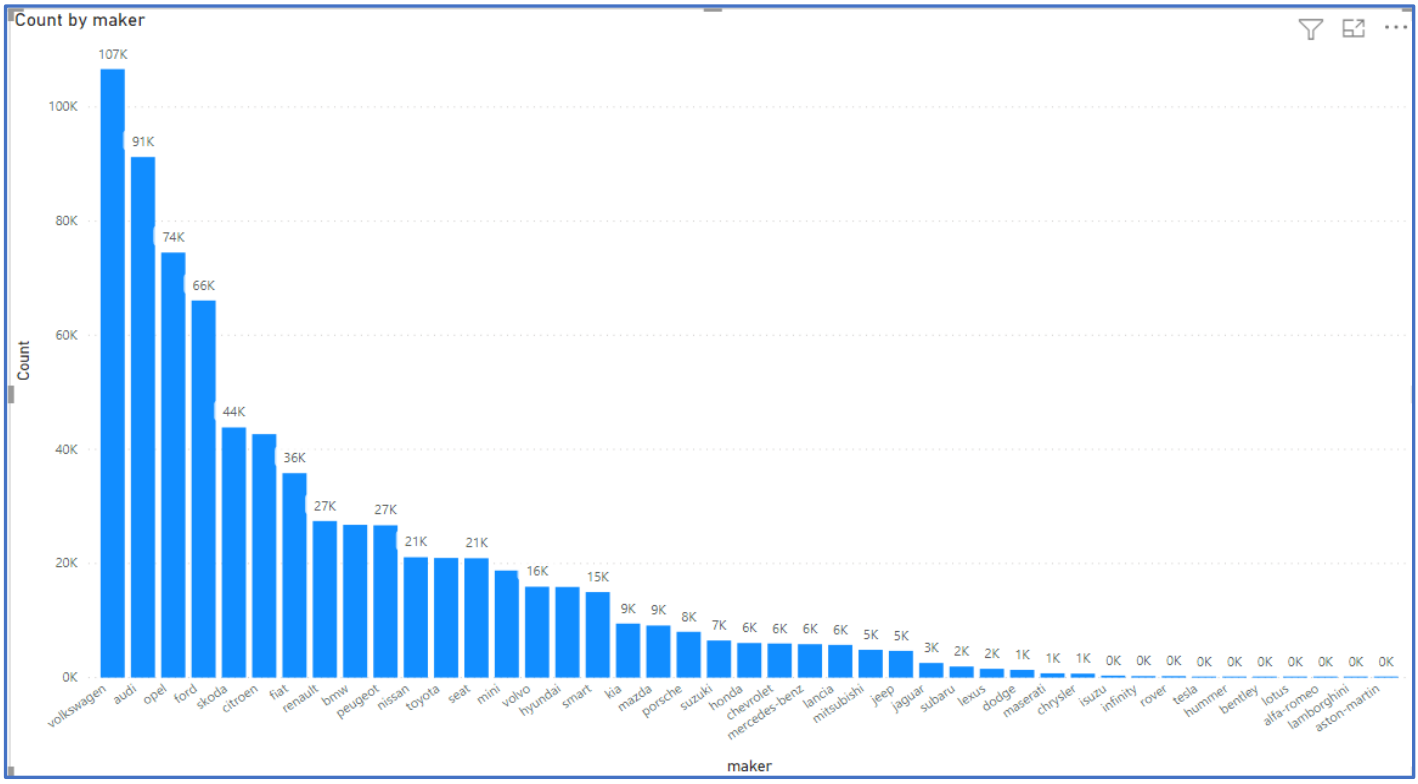

Figure 4: Total count for Top 10 maker

Following Top 10 maker were considered for further analysis.

- volkswagen
- audi
- opel
- ford
- skoda
- citroen
- fiat
- renault
- bmw
- peugeot

2. **Find out top model for top 5 makers extracted from above query.**

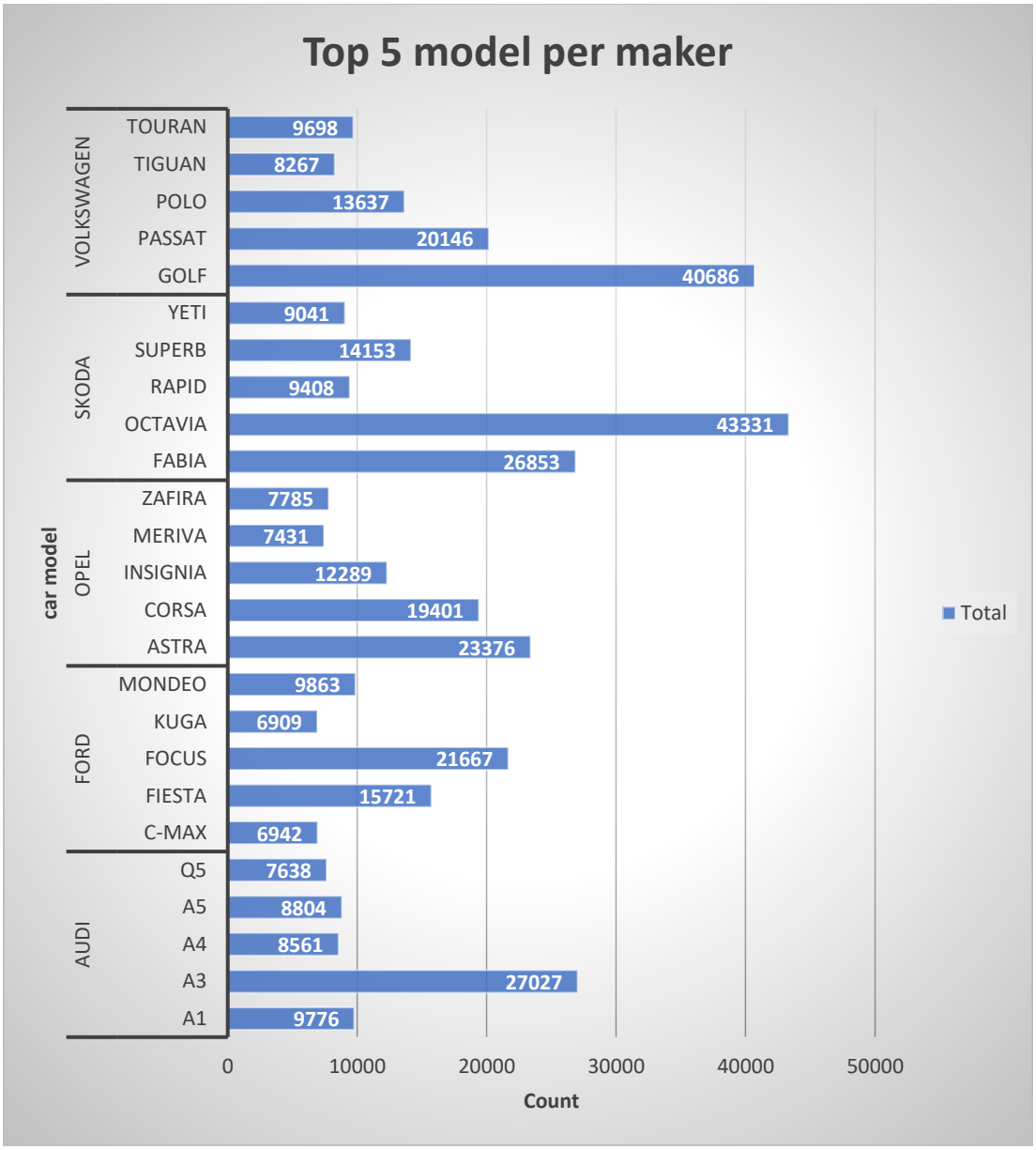

Figure 5: Top 5 model for selected manufacturer

3. Checking the relationship between price and mileage feature.

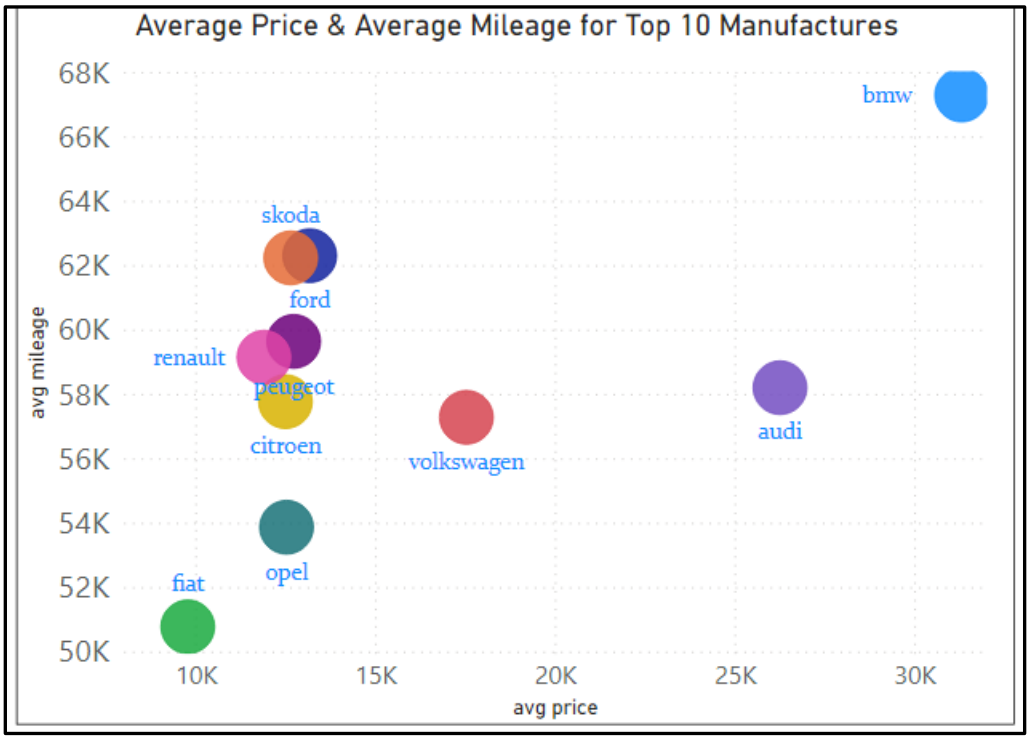

Figure 6: Average price and Average mileage by top 10 maker

Generally dividing average car price into 3 range: high, mid and low, it can be seen that Audi and BMW are the highest selling car in high range Category. Ford, Opel, Skoda, Renault, citroen, Peugeot and fiat have similar no of units, which falls under low range category. Volkswagen has highest no of units in all categories, which falls under mid-range category.

- Ford and AUDI are selected for further consideration because they have number advantage over there counterparts.

4.What is the popular model for selected maker that is AUDI and FORD that sells quickly?

- Maker – Audi
- Model –
  A1
  A3
  A4
  A5
  Q5
- Manufacture_year – 2011 to 2015
- Sold with in 2 weeks of posting.

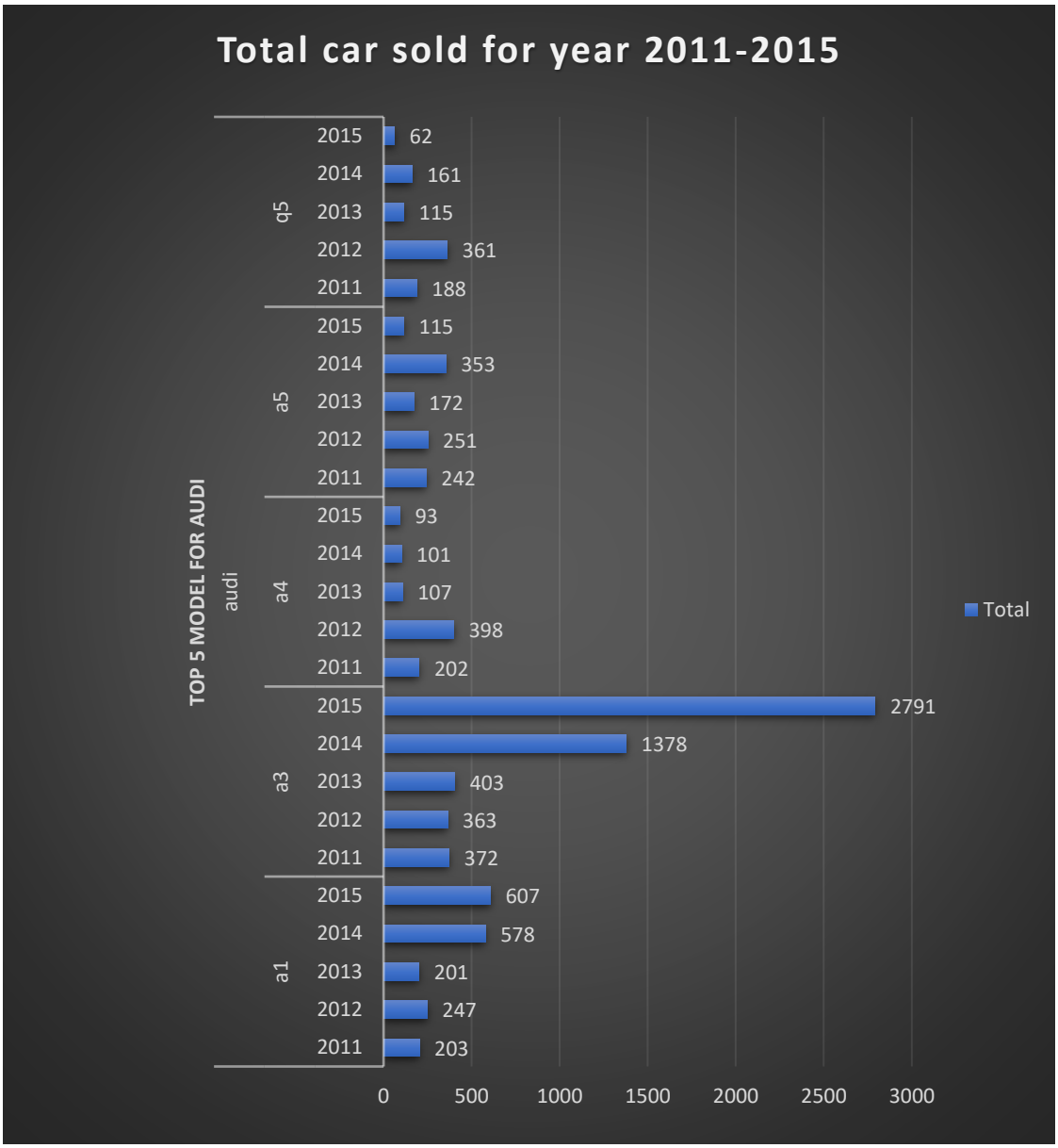

Figure 7: Top 5 model for manufacturer AUDI with sells count for year 2011 to 201

- **Maker – Ford**
- **Model –**
  c-max
  fiesta
  focus
  kuga
  mondue
- **Manufacture_year – 2011 to 2015**
- **Sold with in 2 weeks of posting.**

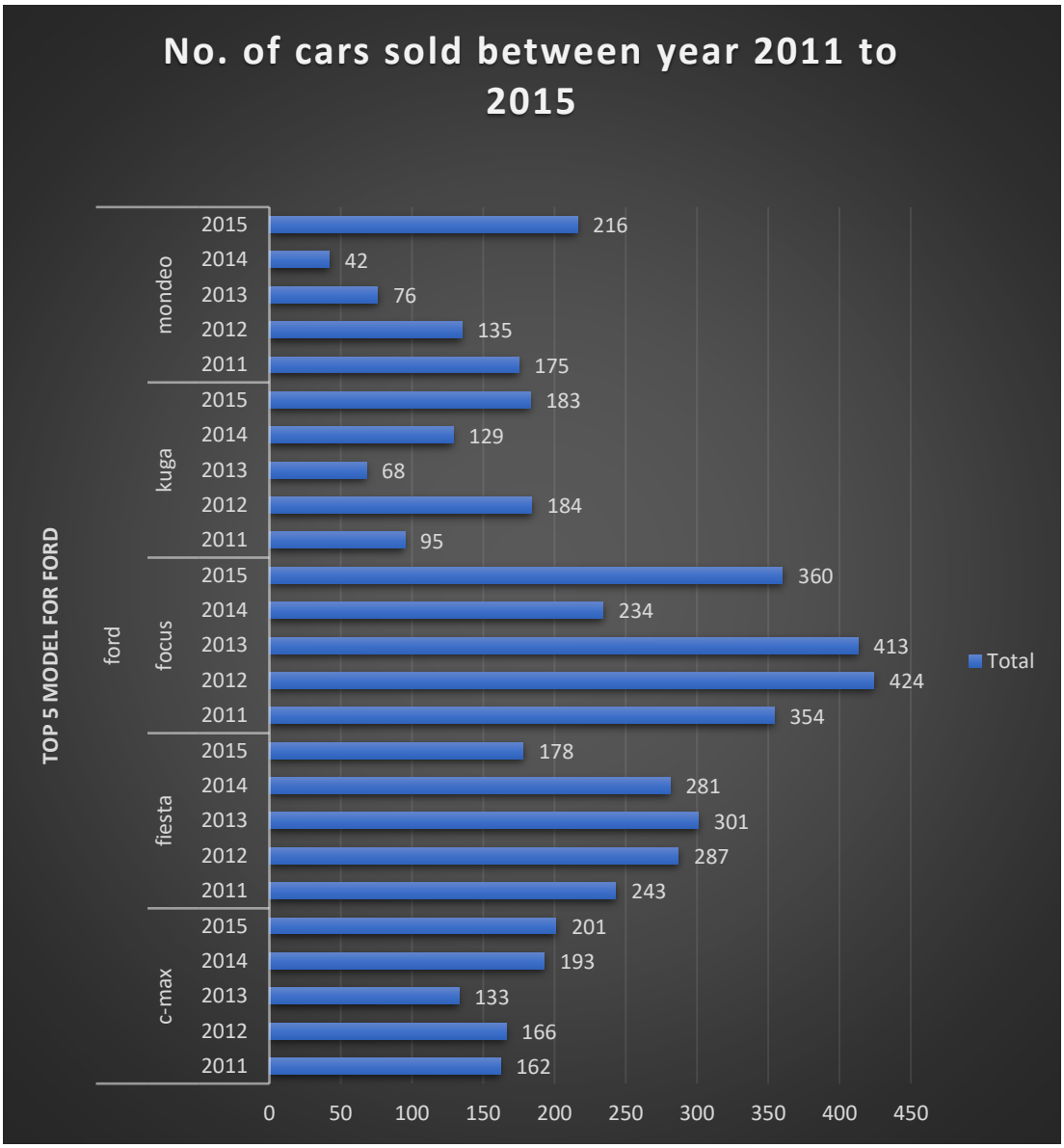

Figure 8: Top 5 model for manufacturer FORD with sells count for year 2011 to 2015

Above analysis can be performed for all remaining makers to find the sell count for car mode

5.What is the most common engine displacement value found in modern car?

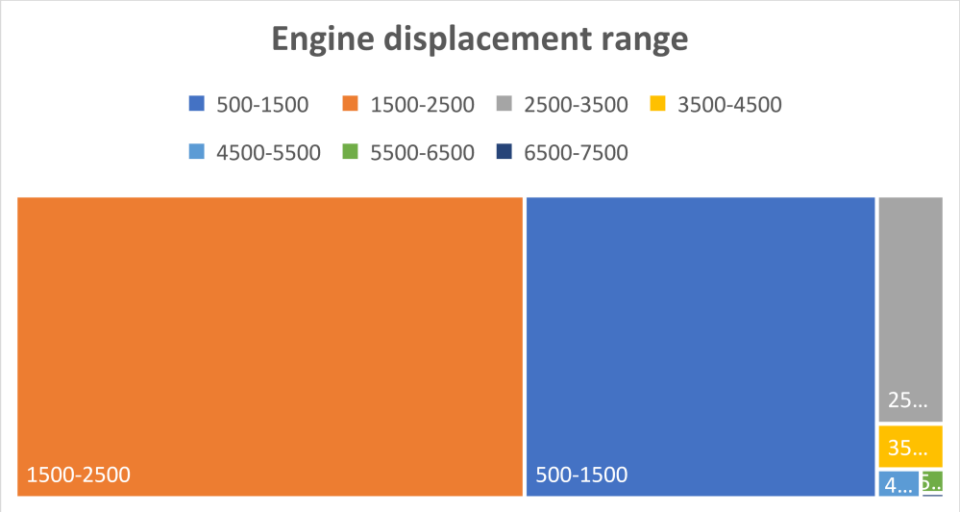

Figure 6: tree map showing maximum car available is between 1500-2500 engine displacement range

Dividing engine displacement value into various class range and counting total car available in each class shows that, maximum cars have engine_displacement value between 1500-2500. Most modern cars run on 2000 cc engine displacement.[1]

Therefore, investing on cars with engine_displacement range between 1500-2500 would be advisable.

Recommendations

According to the overall analysis of the dataset, the recommendation is to move ahead with the used car business investment because of the following outcome from the analysis.

- The analysis indicates that certain car makers have very high count of cars in terms of adds posted, implying that investing on these car makers would be beneficial.
- Considering the fact that counts for low, mid, and high price range are not widely disproportionate, focusing on each category equally would attract customer throughout all price range.
- Based on the analysis, people tend to buy car having fuel_type diesel, transmission_type manual and seat_count between 2 to 5. Considering car model with these factors will be beneficial to the business.
- Considering factors such as warranty, extended warranty and individual maintenance, customers tend to buy cars no older than 10 years.

- Mileage count should be less as much as possible (i.e. <160000). This will attract more customer who tend to invest on used car with profitable price.

Conclusion:

Based on the analyses, it can be concluded that investing on car business for selected dominant car makers would be beneficial for a business. Furthermore, price, mileage and engine_displacement, sell_count, fuel_type, seat_count, transmission_type features play an important role while selecting any car model to be considered for used car business.

Appendix

This section focuses on necessary steps taken to perform Data Cleaning, Data Analysis and Recommendation for the investors.

Data Collection

The used car dataset was downloaded from https://www.kaggle.com/mirosval/personal-cars-classifieds.

Data Loading

Steps to load data into HDFS:

1. Downloading dataset from the following link: https://www.kaggle.com/mirosval/personal-cars-classifieds using.

2. Copy downloaded dataset to /BigData folder in HDFS

```
[cloudera@quickstart ~]$ hadoop fs -ls /BigData/
Found 3 items
-rw-r--r--   1 cloudera supergroup  125979649 2021-03-02 11:56 /BigData/all_anonymized_2015_11_2017_03.csv
drwxr-xr-x   - cloudera supergroup          0 2021-02-20 16:00 /BigData/happiness
-rw-r--r--   1 cloudera supergroup      38740 2021-02-18 22:05 /BigData/happiness.csv
[cloudera@quickstart ~]$
```

3. Connect to csv file loaded in /BigData folder in HDFS.

```
scala> import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.SQLContext

scala> val sqlContext = new SQLContext(sc)
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@523eae6

scala> :paste
// Entering paste mode (ctrl-D to finish)

val cars = sqlContext.read.format("com.databricks.spark.csv")
.option("header", "true")
.option("inferSchema", "true")
.load("hdfs://10.0.2.15:8020/BigData/all_anonymized_2015_11_2017_03.csv")

// Exiting paste mode, now interpreting.
```

Before performing data cleaning, some explorations were applied on data set. This gave some idea about features of dataset, total records, schema etc.

1. Display records of cars Data Frame.
   >>cars.show()

```
scala> cars.show(10)
+-----+-------+-------+--------------+-----------------+-----------+---------+----------+-------+------------+----------+----------+---------+-------------------+-------------------+
|maker|  model|mileage|manufacture_year|engine_displacement|engine_power|body_type|color_slug|stk_year|transmission|door_count|seat_count|fuel_type|       date_created|      date_last_seen|price_eur|
+-----+-------+-------+--------------+-----------------+-----------+---------+----------+-------+------------+----------+----------+---------+-------------------+-------------------+---------+
| ford| galaxy| 151000|          2011|             2000|        103|         |          |   None|         man|         5|         7|   diesel|2015-11-14 18:10:...|2016-01-27 20:40:...| 10584.75|
|skoda|octavia| 143476|          2012|             2000|         81|         |          |   None|         man|         5|         5|   diesel|2015-11-14 18:10:...|2016-01-27 20:40:...|  8882.31|
|  bmw|       |  97676|          2010|             1995|         85|         |          |   None|         man|         5|         5|   diesel|2015-11-14 18:10:...|2016-01-27 20:40:...| 12065.06|
|skoda|  fabia| 111970|          2004|             1200|         47|         |          |   None|         man|         5|         5| gasoline|2015-11-14 18:10:...|2016-01-27 20:40:...|  2960.77|
|skoda|  fabia| 128886|          2004|             1200|         47|         |          |   None|         man|         5|         5| gasoline|2015-11-14 18:10:...|2016-01-27 20:40:...|  2738.71|
|skoda|  fabia| 140932|          2003|             1200|         40|         |          |   None|         man|         5|         5| gasoline|2015-11-14 18:10:...|2016-01-27 20:40:...|  1628.42|
|skoda|  fabia| 167220|          2001|             1400|         74|         |          |   None|         man|         5|         5| gasoline|2015-11-14 18:10:...|2016-01-27 20:40:...|  2072.54|
|  bmw|       | 148500|          2009|             2000|        130|         |          |   None|        auto|         5|         5|   diesel|2015-11-14 18:10:...|2016-01-27 20:40:...| 10547.74|
|skoda|octavia| 105389|          2003|             1900|         81|         |          |   None|         man|         5|         5|   diesel|2015-11-14 18:10:...|2016-01-27 20:40:...|  4293.12|
|     |       | 301381|          2002|             1900|         88|         |          |   None|         man|         5|         5|   diesel|2015-11-14 18:10:...|2016-01-27 20:40:...|  1332.35|
+-----+-------+-------+--------------+-----------------+-----------+---------+----------+-------+------------+----------+----------+---------+-------------------+-------------------+---------+
only showing top 10 rows
```

2. Display columns names.

>>cars.column

```
scala> cars.columns
res2: Array[String] = Array(maker, model, mileage, manufacture_year, engine_displ
lacement, engine_power, body_type, color_slug, stk_year, transmission, door_coun
t, seat_count, fuel_type, date_created, date_last_seen, price_eur)
```

3. Display schema of cars DataFrame.
   >>cars.schema

```
scala> cars.schema
res3: org.apache.spark.sql.types.StructType = StructType(StructField(maker,StringTy
pe,true), StructField(model,StringType,true), StructField(mileage,IntegerType,true)
, StructField(manufacture_year,IntegerType,true), StructField(engine_displacement,I
ntegerType,true), StructField(engine_power,IntegerType,true), StructField(body_type
,StringType,true), StructField(color_slug,StringType,true), StructField(stk_year,St
ringType,true), StructField(transmission,StringType,true), StructField(door_count,S
tringType,true), StructField(seat_count,StringType,true), StructField(fuel_type,Str
ingType,true), StructField(date_created,StringType,true), StructField(date_last_see
n,StringType,true), StructField(price_eur,DoubleType,true))
```

4. Display total records in DataFrame.

>>cars.count()

```
scala> cars.count()
res0: Long = 3552912
```

There are total 3551912 rows and 16 columns in dataset.

Data Cleaning

The original dataset has 16 columns with approximate 3.5 million of record. According to the owner, The data was scraped from several websites in Czech Republic and Germany and some of the sources were completely unstructured, so as a result the data is dirty, there are missing values and some values are very obviously wrong (e.g. phone numbers scraped as mileage etc.).

To understand the data following steps were taken:

1. Identify total no. of null/missing record for the columns.
2. Perform aggregate functions such as count, min, max for specific fields to understand the trend of data for that columns.

3. Review fields that have unrealistic values.

Following queries were performed for data cleaning:

1. **Count of NULL/NOTNULL values in makers column.**

```
>>val cars_maker_notnull = cars.filter(!($”maker”===” “))
    cars_maker_notnull.count()

>> val cars_maker_null = cars.filter(($”maker”===” “))
    cars_maker_null.count()
```

```
scala> val cars_maker_notnull = cars.filter(!($"maker"===""))
cars_maker_notnull: org.apache.spark.sql.DataFrame = [maker: s
r_slug: string, stk_year: string, transmission: string, door_c

scala> cars_maker_notnull.count()
res0: Long = 3033997

scala> val cars_maker_null = cars.filter(($"maker"===""))
cars_maker_null: org.apache.spark.sql.DataFrame = [maker: stri
lug: string, stk_year: string, transmission: string, door_coun

scala> cars_maker_null.count()
res1: Long = 518915
```

2. **Check whether null in maker is also null in model?**

```
>> val cars_model = cars.filter(($”maker”===””) && ($”model”===” “))
    cars_model.count()
```

```
scala> val cars_model =cars.filter(($"maker"===""") && ($"model"===""))
cars_model: org.apache.spark.sql.DataFrame = [maker: string, model: string, m
string, stk_year: string, transmission: string, door_count: string, seat_coun

scala> cars_model.count()
res2: Long = 518915
```

3. **Find Min and max value for mileage column.**

>>val mileage_min_max = cars.select(min(col("mileage")),max(col("mileage"))).show()

```
scala> val mileage_min_max = cars.select(min(col("mileage")), max(col("mileage"))).show()
+------------+------------+
|min(mileage)|max(mileage)|
+------------+------------+
|           0|     9999999|
+------------+------------+
```

4. **Perform group By on Manufacture year and find total car for each year.**

>> val car_by_year = cars.groupBy(col("manufacture_year")).agg(count(col("manufacture_year")).alias("total_car")).orderBy(col("total_car").decs)

```
scala> val car_by_year = cars.groupBy(col("manufacture_year")).agg(count(col("manufacture_year")).alias("total_car")).orderBy(col("total_car").desc)
car_by_year: org.apache.spark.sql.DataFrame = [manufacture_year: int, total_car: bigint]

scala> car_by_year.show()
+---------------+---------+
|manufacture_year|total_car|
+---------------+---------+
|           2015|   441383|
|           2012|   246152|
|           2011|   219843|
|           2014|   201342|
|           2013|   165305|
|           2007|   158319|
|           2010|   157244|
|           2008|   155255|
|           2006|   154670|
|           2009|   145305|
|           2005|   143435|
|           2004|   128594|
|           2016|   123695|
|           2003|   116947|
|           2002|   105510|
|           2001|    98724|
|           2000|    91530|
|           1999|    75095|
|           1998|    55658|
|           1997|    37943|
+---------------+---------+
only showing top 20 rows
```

5. Review Sticker_year column.

>> val cars_by_stkyear = cars.groupBy(col("stk_year")).count().orderBy(col("stk_year").cast("int")).show()

```
scala> val cars_by_stkyear =
     | cars.groupBy(col("stk_year")).count().orderBy(col("stk_year").cast("int")).show()
+--------+-------+
|stk_year|  count|
+--------+-------+
|        |1708156|
|    None|1308651|
|    2015|    869|
|    2016| 124781|
|    2017| 180675|
|    2018| 183761|
|    2019|  44209|
|    2020|    859|
|    2021|     79|
|    2023|      1|
|    2040|      1|
|    2041|      3|
|    2048|      1|
|    2050|      4|
|    2060|      1|
|    2070|     10|
|    2071|      2|
|    2075|      2|
|    2080|      1|
|    2090|      1|
+--------+-------+
only showing top 20 rows
```

6. **Review door_count column**

>> val cars_by_doorcount = cars.groupBy(col("door_count")).count().orderBy(col("count").desc).show()

```
scala> val cars_by_doorcount =
     | cars.groupBy(col("door_count")).count().orderBy(col("count").desc).show()
+----------+-------+
|door_count|  count|
+----------+-------+
|         4|1130741|
|         5| 894084|
|          | 614373|
|      None| 475693|
|         2| 307824|
|         3| 120593|
|         0|   8010|
|         6|   1253|
|         1|    273|
|         7|     43|
|        55|      9|
|         9|      4|
|         8|      3|
|        58|      3|
|        54|      1|
|        22|      1|
|        77|      1|
|        45|      1|
|        49|      1|
|        17|      1|
+----------+-------+
```

7. **Review Seat_count column.**

>> val cars_by_seatcount = cars.groupBy(col("seat_count")).count().orderBy(col("count").desc).show()

```
scala> val cars_by_seatcount =
     | cars.groupBy(col("seat_count")).count().orderBy(col("count").desc).show()
+----------+-------+
|seat_count|  count|
+----------+-------+
|         5|1767868|
|          | 749489|
|      None| 537610|
|         4| 244797|
|         7| 100744|
|         2|  72685|
|         3|  33607|
|         6|  14174|
|         9|  12575|
|         0|  11695|
|         8|   6754|
|         1|    567|
|        17|     39|
|        10|     35|
|        12|     31|
|        15|     19|
|        14|     19|
|        18|     16|
|        19|     14|
|        21|     13|
+----------+-------+
only showing top 20 rows
```

8. Review date_created and date_last_seen column.

>> val cars_cols = cars.select(col("date_created"),col("date_last_seen")).show()

```
scala> val cars_cols = cars.select(col("date_created"), col("date_last_seen")).show()
+-------------------+-------------------+
|       date_created|     date_last_seen|
+-------------------+-------------------+
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
|2015-11-14 18:10:...|2016-01-27 20:40:...|
+-------------------+-------------------+
only showing top 20 rows
```

9. Calculate min and max value of price column.

>> val price_min_max = cars.select(min(col("price_eur")),max(col("price_eur"))).show()

```
scala> val price_min_max = cars.select(min(col("price_eur")), max(col("price_eur"))).show()
+--------------+-----------------+
|min(price_eur)|   max(price_eur)|
+--------------+-----------------+
|          0.04|2.7061490530644E12|
+--------------+-----------------+
```

10. Identify NULL/missing value in color_slug column.

>>val = color_clug_null = cars.filter(($"color_clug"===" "))
color_slug_null.count()

```
scala> val color_slug_null = cars.filter(($"color_slug"===""))
color_slug_null: org.apache.spark.sql.DataFrame = [maker: string,
lug: string, stk_year: string, transmission: string, door_count:

scala> color_slug_null.count()
res9: Long = 3343411
```

11. Review columns engine_displacement, transmission, power, fuel_type

>>val cars_cols = cars.select(col("engine_displacement"),col("engine_power"),col("transmission"),col("fuel_type")).show()

```
scala> val cars_cols = cars.select(col("engine_displacement"), col("engine_power"),col("transmission"),col("fuel_type")).show()
+------------------+------------+------------+---------+
|engine_displacement|engine_power|transmission|fuel_type|
+------------------+------------+------------+---------+
|              2000|         103|         man|   diesel|
|              2000|          81|         man|   diesel|
|              1995|          85|         man|   diesel|
|              1200|          47|         man| gasoline|
|              1200|          47|         man| gasoline|
|              1200|          40|         man| gasoline|
|              1400|          74|         man| gasoline|
|              2000|         130|        auto|   diesel|
|              1900|          81|         man|   diesel|
|              1900|          88|         man|   diesel|
|              1400|          55|         man| gasoline|
|              1900|          81|         man|   diesel|
|              1360|          55|         man| gasoline|
|              1300|          44|         man| gasoline|
|              1000|          39|         man| gasoline|
|              2500|         121|        auto| gasoline|
|              1900|          88|         man|   diesel|
|              1700|          74|         man|   diesel|
|              1900|          96|         man|   diesel|
|              1200|          44|         man| gasoline|
+------------------+------------+------------+---------+
only showing top 20 rows
```

12. Identify Null/missing values in body_type columns.

>> val body_type_null = cars.filter(($"body_type"===""))
Body_type_null.count()

```
scala> val body_type_null = cars.filter(($"body_type"===""))
body_type_null: org.apache.spark.sql.DataFrame = [maker: string, mo
ug: string, stk_year: string, transmission: string, door_count: str

scala> body_type_null.count()
res8: Long = 1122914
```

Observations from the above analysis:

1. There are total 518519 missing value in makers column.
2. When a maker field is blank, the model field is also automatically blank.
3. The mileage column may contain unrealistic data. According to the Kaggle description, extreme values are likely phone numbers. Some fields also have numbers that are too low (i.e. under 10 km).
4. The manufacture year column includes data from unlikely time periods.
5. The body type column has 1122914 missing entries which is 31.6% of total record.
6. The color_clug has 3343411 missing entries which is 94.1% of total record.
7. The sticker_year column has some unrealistic values. (i.e year 2023)
8. The door_count and seat_count also has some unrealistic values.
9. The date fields included a timestamp that is down to the millisecond.
10. There are inaccuracies in some car prices - for instance, multiple vehicles including luxury cars were priced at under 1€ while sedans were valued at over 2€ trillion.

Data Preparation

After performing data cleaning and observing the result the following decision were made for each column for further analysis.

1. maker: all the maker records excluding null values.
2. model: all the model records excluding null values.
3. mileage: records with value between 5000 and 160000
    - dataset contains lots of values that were not feasible i.e maximum mileage value was 9999999.0. in addition, customers generally would not prefer cars that has been driven more than certain amount.
4. manufacture_year: considering manufacture_year between 2008 and 2017
    - Considering factors such as warranty, extended warranty and individual maintenance, customers tend to buy cars no older than 10 years. In reference to that, cars manufactured later than year 2008 are considered for analysis.
5. engine_displacement: all the records engine_displacement.
6. engine_power: all the records of engine_power.
7. transmission: all the records of transmission
8. seat_count: all the records of seat_count
    - Although, seat_count has some unrealistic data but, combining them with maker, model, manufacture_year and other column can lead to appropriate analysis.
9. fuel_type: all the records of fuel_type

10. price_eur: price in euro between 5000 and 100000.
    - Dataset contains large no. of records having car price in millions which is not appropriate. One would seldom invest such large amount in used cars.
11. body_type, color_slug and stk_year is not considered for further analysis because of lack of data.
12. All records of date_created and date_last_seen.
13. Considering door_count and seat_count, seat_count gives better idea about the type of car. Therefore, door_count is not considered for further analysis.

Data Dictionary

Attributes	Datatype
maker	STRING
model	STRING
mileage	FLOAT
manufacture_year	INTEGER
engine_displacement	FLOAT
engine_power	STRING
Transmission	STRING
seat_count	INTEGER
fuel_type	STRING
date_created	DATE
date_last_seen	DATE
price_eur	FLOAT

The new DataFrame was created based on the values mentioned above.

1. Remove NULL values from maker column.
   >>val cars_clean = cars.filter(!($"maker"===""))

```
scala> val cars_clean = cars.filter(!($"maker"===""))
cars_clean: org.apache.spark.sql.DataFrame = [maker: string,
string, stk_year: string, transmission: string, door_count:

scala> cars_clean.count()
res0: Long = 3033997
```

2. Remove NULL values from model column.
   >>val cars_clean1 = cars_clean.filter(!($"model"===""))

```
scala> val cars_clean1 = cars_clean.filter(!($"model"===""))
cars_clean1: org.apache.spark.sql.DataFrame = [maker: string, mode
 string, stk_year: string, transmission: string, door_count: strin

scala> cars_clean1.count()
res1: Long = 2419551
```

3. Final DataFrame with all the above-mentioned conditions.

>> val cars_clean2 = cars_clean1.select(col("maker"), col ("model"), col("mileage").cast("int"),col("manufacture_year").cast("int"), col("engine_displacement").cast("int"),col("engine_power").cast("int"), col("transmission"),col("seat_count").cast("int"), col("fuel_type"), col("date_created").cast("date"),col("date_last_seen").cast("date"), col("price_eur").cast("int")).filter(col("mileage").between (5000, 160000) && col("manufacture_year").between (2008, 2017) && col("price_eur").between(5000, 100000))

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

val cars_clean2 = cars_clean1.select(col("maker"), col ("model"), col("mileage").cast("int"),
col("manufacture_year").cast("int"),col("engine_displacement").cast("int"),col("engine_power").cast("int"),
col("transmission"),col("seat_count").cast("int"), col("fuel_type"), col("date_created").cast("date"),
col("date_last_seen").cast("date"),col("price_eur").cast("int")).filter(col("mileage").between (5000, 160000) &&
col("manufacture_year").between (2008, 2017) && col("price_eur").between(5000, 100000))
```

- Display records of cars_clean2 DataFrame
  >>cars_clean2.show()

```
scala> cars_clean2.show()
+-------+--------+-------+---------------+------------------+-----------+------------+----------+---------+------------+-------------+---------+
|  maker|   model|mileage|manufacture_year|engine_displacement|engine_power|transmission|seat_count|fuel_type|date_created|date_last_seen|price_eur|
+-------+--------+-------+---------------+------------------+-----------+------------+----------+---------+------------+-------------+---------+
|   ford|  galaxy| 151000|           2011|              2000|        103|         man|         7|   diesel|  2015-11-14|   2016-01-27|    10584|
|  skoda| octavia| 143476|           2012|              2000|         81|         man|         5|   diesel|  2015-11-14|   2016-01-27|     8882|
| suzuki|   swift| 113175|           2013|              1600|        100|         man|         4| gasoline|  2015-11-14|   2016-01-27|     7401|
|    kia|    soul|  40184|           2010|              1600|         93|         man|         5| gasoline|  2015-11-14|   2016-01-27|     6883|
|   ford|   focus| 159427|           2012|              1600|         85|         man|         5|   diesel|  2015-11-14|   2016-01-27|     8771|
|  skoda| octavia|  38513|           2009|              1600|         75|         man|         5| gasoline|  2015-11-14|   2016-01-27|     7957|
|citroen|      c5| 143130|           2011|              1600|         82|         man|         5|   diesel|  2015-11-14|   2016-01-27|     7512|
|   ford|   c-max|  99713|           2008|              1600|         74|         man|         5| gasoline|  2015-11-14|   2016-01-27|     6476|
|  skoda| octavia| 101761|           2009|              1900|         77|         man|         5|   diesel|  2015-11-14|   2016-01-27|     9363|
|  skoda| octavia|  55569|           2011|              1600|         77|         man|         5|   diesel|  2015-11-14|   2016-01-27|     9548|
|   ford|   focus| 134000|           2008|              1600|         80|         man|         5|   diesel|  2015-11-14|   2016-01-27|     5107|
|  skoda| octavia| 132140|           2010|              1896|         77|         man|         5|   diesel|  2015-11-14|   2016-01-27|     7031|
|  skoda| octavia|  70295|           2009|              1900|         74|         man|         5|   diesel|  2015-11-14|   2016-01-27|     6994|
|    bmw|      x6| 107763|           2008|              3000|        225|        auto|         4| gasoline|  2015-11-14|   2016-01-27|    22205|
|  skoda|    yeti| 148100|           2010|              2000|        103|         man|         5|   diesel|  2015-11-14|   2016-01-27|    12213|
|hyundai|     i20|  29934|           2010|              1200|         57|         man|         5| gasoline|  2015-11-14|   2016-01-27|     5773|
|  skoda|   fabia|   7993|           2015|              1200|         81|         man|         5| gasoline|  2015-11-14|   2016-01-27|    11102|
|  skoda|roomster|  56145|           2008|              1200|         51|         man|         5| gasoline|  2015-11-14|   2016-01-27|     5551|
| nissan| qashqai|  44450|           2010|              2000|        110|         man|         5|   diesel|  2015-11-14|   2016-01-27|    13693|
|hyundai|    ix35|  67708|           2014|              2000|        100|        auto|         5|   diesel|  2015-11-14|   2016-01-27|    17764|
+-------+--------+-------+---------------+------------------+-----------+------------+----------+---------+------------+-------------+---------+
only showing top 20 rows
```

- Further analysis was conducted on 746648 rows x 12 columns.

Data Analysis

- The following queries were performed for further analyzing the data.

1. Count total no. of cars for selected manufacture_year.
   >> val car_count_by_year = cars_clean2.groupBy(col("manufacture_year")).count().orderBy(col("manufacture_year")).show()

```
val car_count_by_year =
cars_clean2.groupBy(col("manufacture_year")).count().orderBy(col("manufacture_year")).show()

// Exiting paste mode, now interpreting.

+-----------------+------+
|manufacture_year| count|
+-----------------+------+
|             2008| 42552|
|             2009| 52789|
|             2010| 69614|
|             2011|109041|
|             2012|122177|
|             2013| 87365|
|             2014|102951|
|             2015|156395|
|             2016|  3764|
+-----------------+------+
```

The above result shows maximum entries for year 2011 to 2015.

2. Count total cars for selected seat_count.

```
>> val cars_by_seatcount =
cars_clean2.groupBy(col("seat_count")).count().orderBy(col("count").desc)
.filter(col("seat_count").between(2,9)).show
```

```
val cars_by_seatcount =
cars_clean2.groupBy(col("seat_count")).count().orderBy(col("count").desc).filter(col("seat_count")
.between(2,9)).show

// Exiting paste mode, now interpreting.

+----------+------+
|seat_count| count|
+----------+------+
|         5|450478|
|         4| 76640|
|         7| 31326|
|         2| 17471|
|         3|  7339|
|         9|  3215|
|         6|  2189|
|         8|  1141|
+----------+------+
```

3. Count total cars for fuel_type columns.

```
>> val cars_by_fueltype =
cars_clean2.groupBy(col("fuel_type")).count().orderBy(col("fuel_type")).show()
```

```
val cars_by_fueltype =
cars_clean2.groupBy(col("fuel_type")).count().orderBy(col("fuel_type")).show()

// Exiting paste mode, now interpreting.

+---------+------+
|fuel_type| count|
+---------+------+
|         |490940|
|   diesel|140207|
| gasoline|115501|
+---------+------+
```

4. Count total no. of entries for each transmission type.

```
>> val cars_by_transmissiontype =
cars_clean2.groupBy(col("transmission")).count().orderBy(col("transmission")).show()
```

```
val cars_by_transmissiontype =
cars_clean2.groupBy(col("transmission")).count().orderBy(col("transmission")).show()

// Exiting paste mode, now interpreting.

+------------+------+
|transmission| count|
+------------+------+
|             | 33429|
|        auto|192453|
|         man|520766|
+------------+------+
```

To narrow down the analysis process, following queries were performed to understand the popular trends.

1. Calculate total entries by each maker.

```
>> val cars_by_maker =
cars_clean2.groupBy(col("maker")).count().orderBy(col("count").desc).show()
```

result from above query was saved in hdfs CSV format.

maker	Count
volkswagen	106640
audi	91229
opel	74489
ford	66089
skoda	43838
citroen	42664
fiat	35823
renault	27440
bmw	26770
peugeot	26686
nissan	21091
toyota	20963
seat	20916
mini	18763
volvo	15926
hyundai	15884
smart	14986
kia	9463
mazda	9123
porsche	8026
suzuki	6501
honda	6058
chevrolet	5972
mercedes-benz	5852
lancia	5735
mitsubishi	4864
jeep	4684
jaguar	2560
subaru	1940
lexus	1537
dodge	1347
maserati	750
chrysler	696
isuzu	348
infinity	263
rover	261
tesla	127
hummer	111
bentley	107
lotus	47
alfa-romeo	44
lamborghini	34
aston-martin	1

2. Find out top model for top 10 makers extracted from above query.

```
>> val cars_year =
cars_top10maker.groupBy(col("maker"),col("model"),col("manufacture_year")).count()
.orderBy(col("count").desc).show()

>>cars_top10maker.count()
    541668
```

result from above query was saved in hdfs CSV format.

```
val cars_year =
cars_top10maker.groupBy(col("maker"),col("model"),col("manufacture_year")).count().orderBy(col("count").desc).show()

// Exiting paste mode, now interpreting.

+----------+-------+---------------+-----+
|     maker|  model|manufacture_year|count|
+----------+-------+---------------+-----+
|volkswagen|   golf|           2015|11967|
|      audi|     a3|           2015| 9966|
|      opel|  corsa|           2015| 4684|
|      audi|     a3|           2014| 4349|
|volkswagen|   golf|           2012| 4323|
|volkswagen|   golf|           2014| 4288|
|volkswagen|   golf|           2011| 4260|
|      opel|  astra|           2015| 3836|
|      ford|  focus|           2015| 3492|
|      opel|  astra|           2012| 3413|
|volkswagen| passat|           2015| 3393|
|      opel|  astra|           2011| 3383|
|volkswagen|   golf|           2013| 3264|
|     skoda|octavia|           2015| 3043|
|     skoda|octavia|           2012| 2979|
|volkswagen| passat|           2012| 2964|
|      ford|  focus|           2013| 2912|
|volkswagen|   polo|           2015| 2909|
|      audi|     a1|           2015| 2895|
|      opel|  mokka|           2015| 2851|
+----------+-------+---------------+-----+
only showing top 20 rows
```

3. Checking the relationship between price and mileage feature.

```
>> val cars_avg =
cars_top10maker.groupBy(col("maker")).agg(avg("price_eur"),avg("mileage")).show()
```

```
scala> val cars_avg = cars_top10maker.groupBy(col("maker")).agg(avg("price_eur"),avg("mileage")).show()
+----------+------------------+-----------------+
|     maker|    avg(price_eur)|     avg(mileage)|
+----------+------------------+-----------------+
|      fiat| 9767.845992797924|50791.95416352623|
|       bmw|31296.612850205453|67308.59850579007|
|volkswagen|17518.743914103525|57298.76009002251|
|      audi|26243.749750627543|58219.72514222451|
|      ford|13160.788754558247|62320.14577312412|
|   renault|11884.807653061225|59164.46599854228|
|     skoda|12632.213353711391|  62251.805921803|
|      opel|12514.524225053363|53884.79986306703|
|   peugeot|12714.577868545304|59657.43768268006|
|   citroen|12484.301354772173|57785.91944027752|
+----------+------------------+-----------------+
```

4. Review the typical number of days that the ads were posted for a particular maker/model with a manufacture year between 2008 and 2017.

```
>> val cars_days = cars_top10maker.groupBy(col("maker"), col("model"),
col("manufacture_year")).agg(min(datediff(col("date_last_seen"),col("date_created"))).alias(
"min_datediff"),max(datediff(col("date_last_seen"),col("date_created"))).alias("max_datediff
")).filter(col("manufacture_year").between(2008,2017)).orderBy(col("maker"),col("model"),
col("manufacture_year")).show()
```

```
val cars_days = cars_top10maker.groupBy(col("maker"), col("model"),
col("manufacture_year")).agg(min(datediff(col("date_last_seen"),col("date_created"))).alias("min_datediff"),
max(datediff(col("date_last_seen"),col("date_created"))).alias("max_datediff")).filter(col("manufacture_year")
.between(2008, 2017)).orderBy(col("maker"),col("model"),col("manufacture_year")).show()

// Exiting paste mode, now interpreting.

+-----+-----+---------------+------------+------------+
|maker|model|manufacture_year|min_datediff|max_datediff|
+-----+-----+---------------+------------+------------+
| audi|  100|           2008|           4|         139|
| audi|  100|           2009|         121|         174|
| audi|  100|           2010|          10|         137|
| audi|  100|           2011|           3|         175|
| audi|  100|           2012|          11|         142|
| audi|  100|           2013|           9|         174|
| audi|  100|           2014|           7|         143|
| audi|  100|           2015|          22|         176|
| audi|  200|           2008|           3|         175|
| audi|  200|           2009|           2|         172|
| audi|  200|           2010|           2|         175|
| audi|  200|           2011|           3|         141|
| audi|  200|           2012|           3|         142|
| audi|  200|           2013|          10|         135|
| audi|  200|           2014|          10|         173|
| audi|  200|           2015|           1|         167|
| audi|  200|           2016|         118|         143|
| audi|   80|           2008|          16|         164|
| audi|   80|           2009|           2|         170|
| audi|   80|           2010|           3|         163|
+-----+-----+---------------+------------+------------+
only showing top 20 rows
```

All the results were saved in CSV and further used to generate visualization using excel and PowerBI.

Reference:

[1] https://www.carpart.com.au/blog/educational/what-is-engine-displacement-and-why-does-it-matter#:~:text=Engine%20displacement%20is%20expressed%20in,one%2Dhalf%20litre%20or%20500cc.