# DATA MANAGEMENT SYSTEMS

# PROJECT 2
# ANALYSIS OF A REAL WORLD PROBLEM

# USED CAR SALES IN THE USA AND RUSSIA

Amelia Waszkowska

## 1. Objective

The goal of this project is to develop a machine learning model using big data analysis techniques to predict the prices of used cars based on various features and attributes. By leveraging open datasets related to used car offers, the project aims to explore patterns, correlations, and factors influencing car prices, ultimately enabling accurate price predictions.

## 2. Datasets

I used following big datasets available on kaggle:

https://www.kaggle.com/datasets/ekibee/car-sales-information
https://www.kaggle.com/datasets/rupeshraundal/marketcheck-automotive-data-us-canada?select=us-dealers-used.csv

## 3. Storage

Massive data require using specific data processing framework, like Apache Hadoop and or Apache Spark. These two frameworks are often used together:

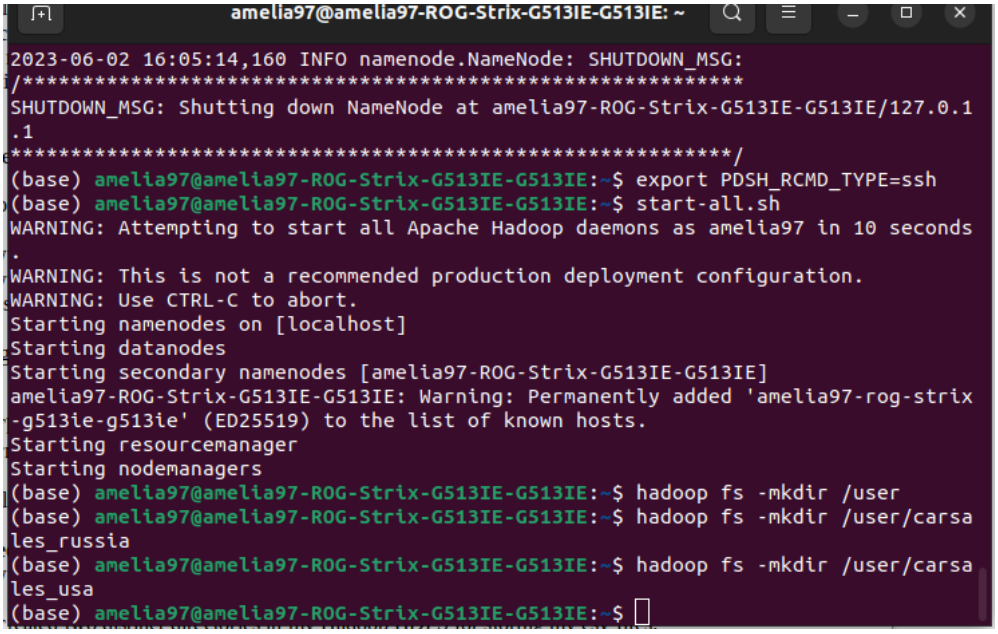
Hadoop HDFS (Hadoop Distributed File System) is a distributed file system designed to store and manage large volumes of data across multiple computers in a Hadoop cluster. It is a key component of the Apache Hadoop framework, which is widely used for big data processing and analytics.

Apache Spark is an open-source distributed computing framework that is designed for big data processing and analytics. It provides an interface for programming and executing data processing tasks across a cluster of machines, making it highly scalable and efficient.

## 4. Installing Hadoop

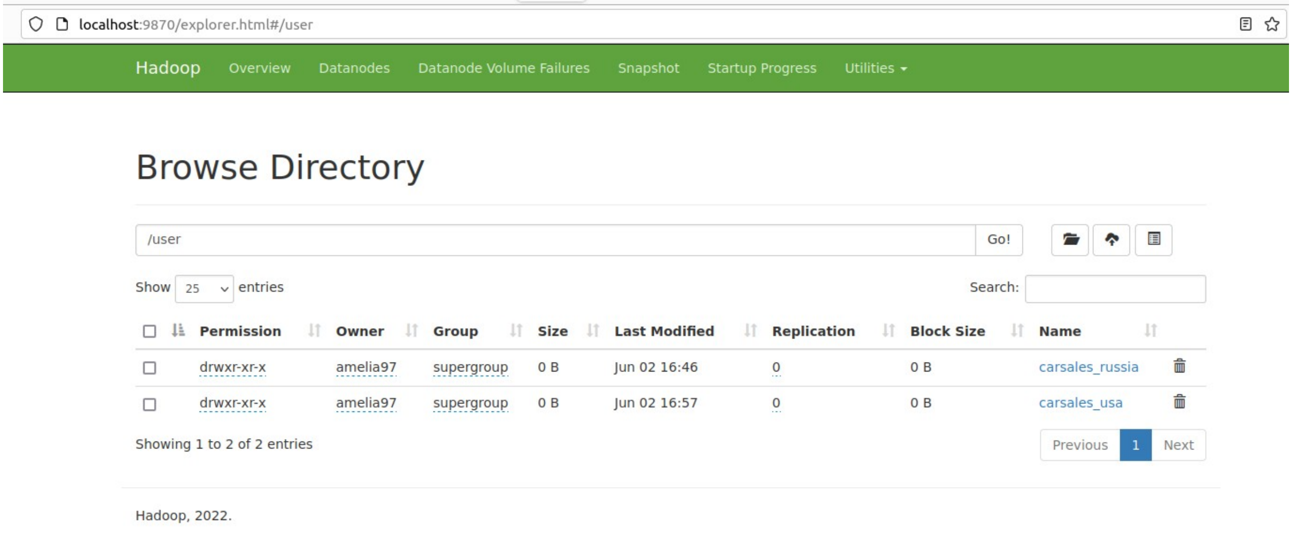
I installed Hadoop 3.2.4 following this tutorial:
https://www.youtube.com/watch?v=Slbi-uzPtnw

Next, I created two distinct directories in my Hadoop HDFS for storing my csv files:



```
2023-06-02 16:05:14,160 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at amelia97-ROG-Strix-G513IE-G513IE/127.0.1
.1
************************************************************/
(base) amelia97@amelia97-ROG-Strix-G513IE-G513IE:~$ export PDSH_RCMD_TYPE=ssh
(base) amelia97@amelia97-ROG-Strix-G513IE-G513IE:~$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as amelia97 in 10 seconds
.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [amelia97-ROG-Strix-G513IE-G513IE]
amelia97-ROG-Strix-G513IE-G513IE: Warning: Permanently added 'amelia97-rog-strix
-g513ie-g513ie' (ED25519) to the list of known hosts.
Starting resourcemanager
Starting nodemanagers
(base) amelia97@amelia97-ROG-Strix-G513IE-G513IE:~$ hadoop fs -mkdir /user
(base) amelia97@amelia97-ROG-Strix-G513IE-G513IE:~$ hadoop fs -mkdir /user/carsa
les_russia
(base) amelia97@amelia97-ROG-Strix-G513IE-G513IE:~$ hadoop fs -mkdir /user/carsa
les_usa
(base) amelia97@amelia97-ROG-Strix-G513IE-G513IE:~$ 
```

After opening the localhost 9870 in the browser, it looks like this:



Browse Directory

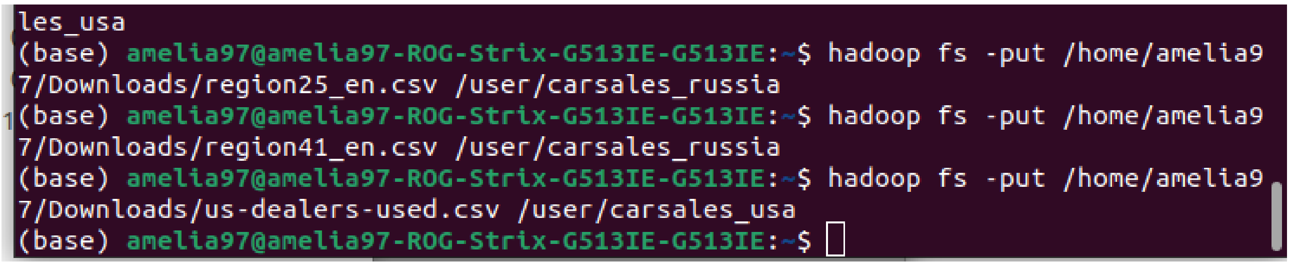
/user

| □ | Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|---|---|---|---|---|---|---|---|---|
| □ | drwxr-xr-x | amelia97 | supergroup | 0 B | Jun 02 16:46 | 0 | 0 B | carsales_russia |
| □ | drwxr-xr-x | amelia97 | supergroup | 0 B | Jun 02 16:57 | 0 | 0 B | carsales_usa |

Showing 1 to 2 of 2 entries

Hadoop, 2022.

Next, I uploaded the files:



```
les_usa
(base) amelia97@amelia97-ROG-Strix-G513IE-G513IE:~$ hadoop fs -put /home/amelia9
7/Downloads/region25_en.csv /user/carsales_russia
(base) amelia97@amelia97-ROG-Strix-G513IE-G513IE:~$ hadoop fs -put /home/amelia9
7/Downloads/region41_en.csv /user/carsales_russia
(base) amelia97@amelia97-ROG-Strix-G513IE-G513IE:~$ hadoop fs -put /home/amelia9
7/Downloads/us-dealers-used.csv /user/carsales_usa
(base) amelia97@amelia97-ROG-Strix-G513IE-G513IE:~$ 
```

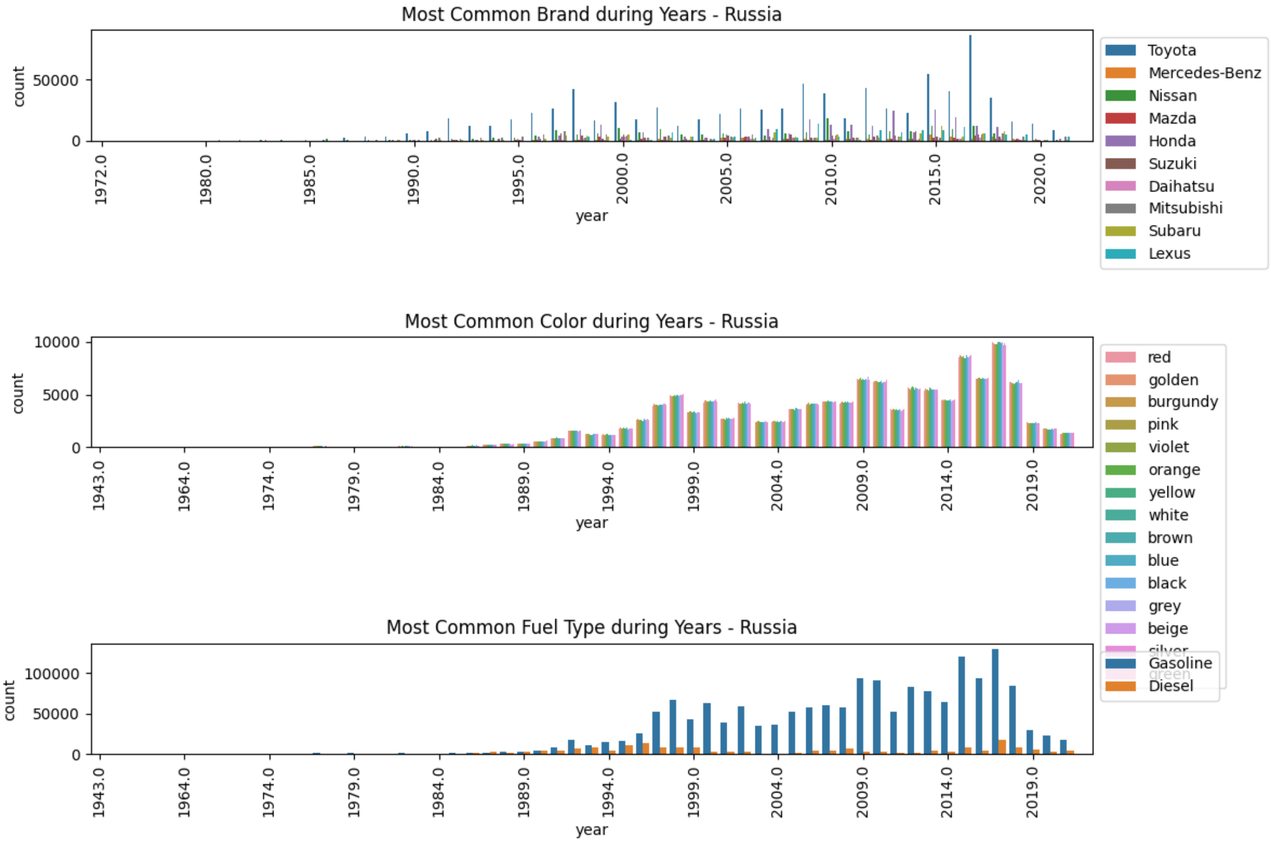
5. Setting up Spark and Jupyter

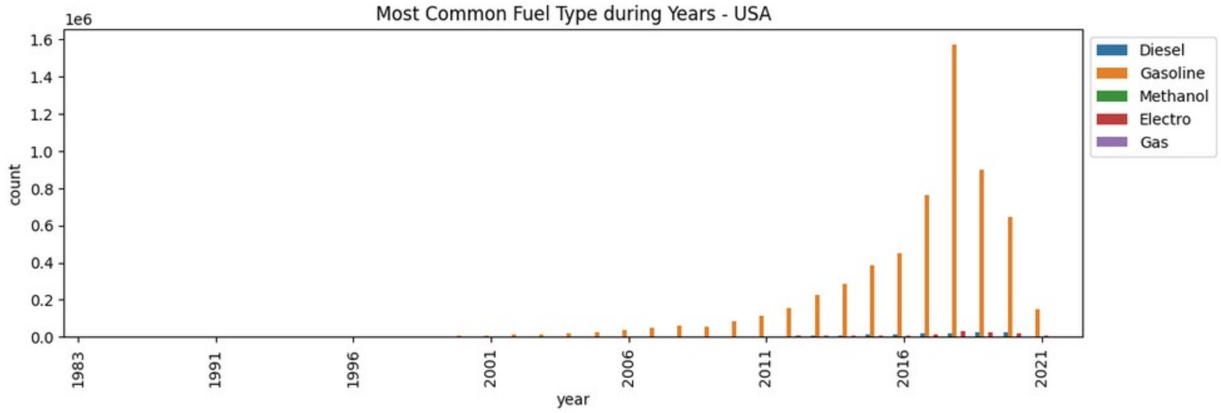
Since Spark may be problematic to install, it is advisable to create a new environment for Jupyter notebook, in which we want to use Spark. So I did. To use Spark in Jupyter notebook, we can just run "!pip install" in the notebook. Of course, this installation approach has a limitation: it only installs pyspark within the notebook environment, and other Jupyter Notebook sessions or terminals won't have access to it. But in this project it's not a problem.

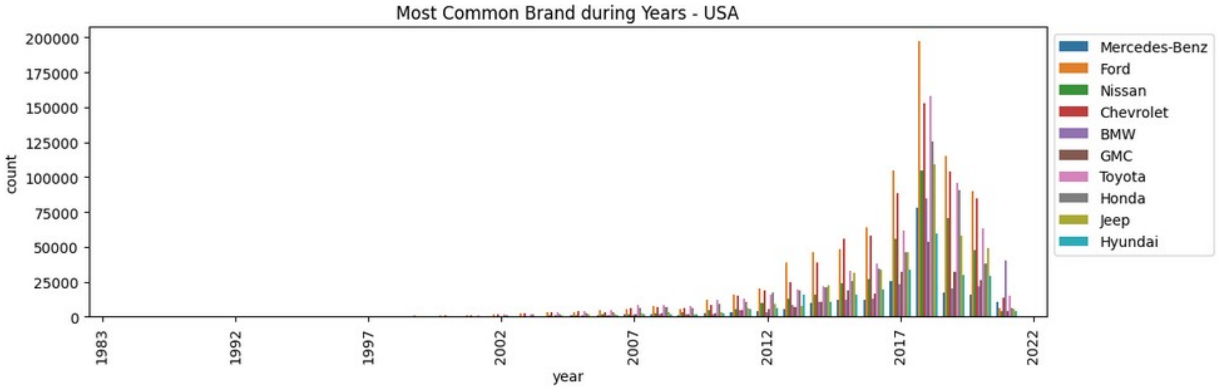
From now and on, I will write majority of descriptions in the Jupyter notebook.

6. Pipeline decription

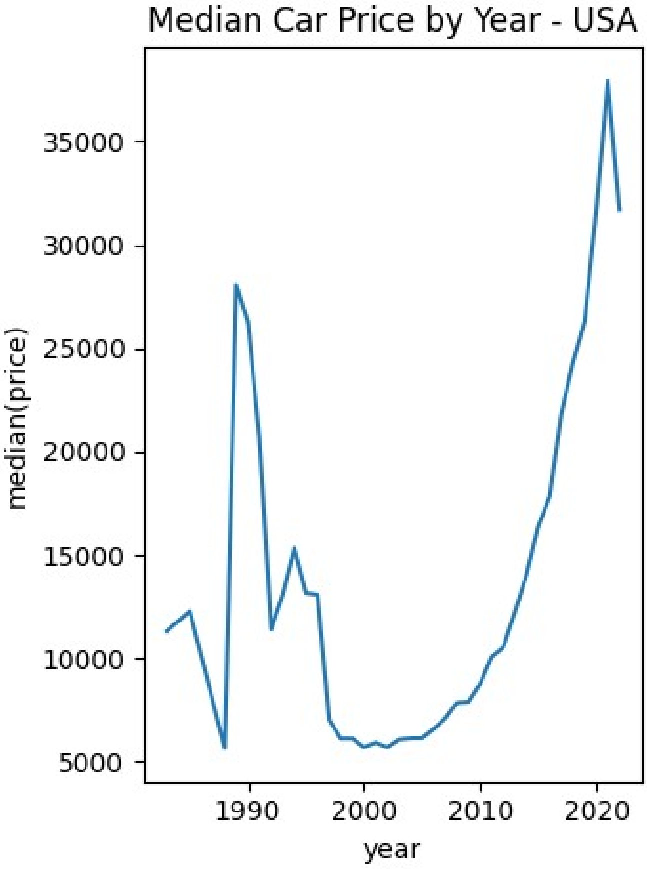
7. Most common brand, color and fuel type throughout the years.

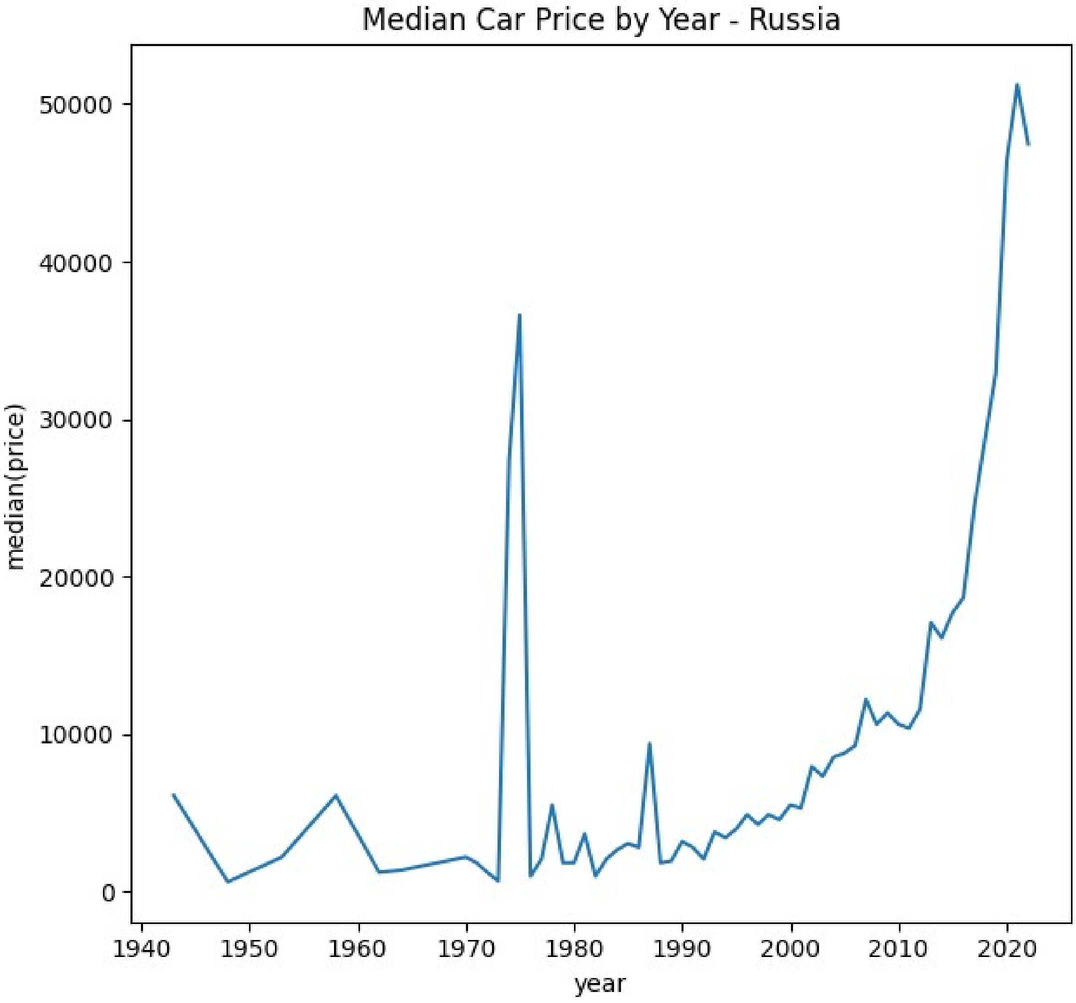




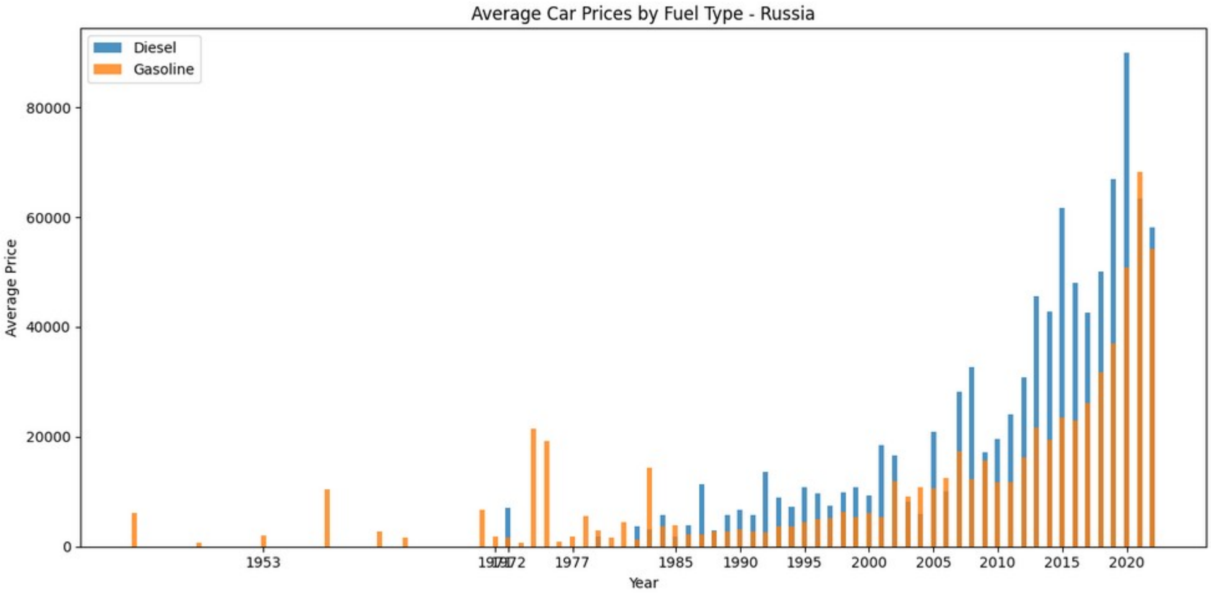


8. Median car price

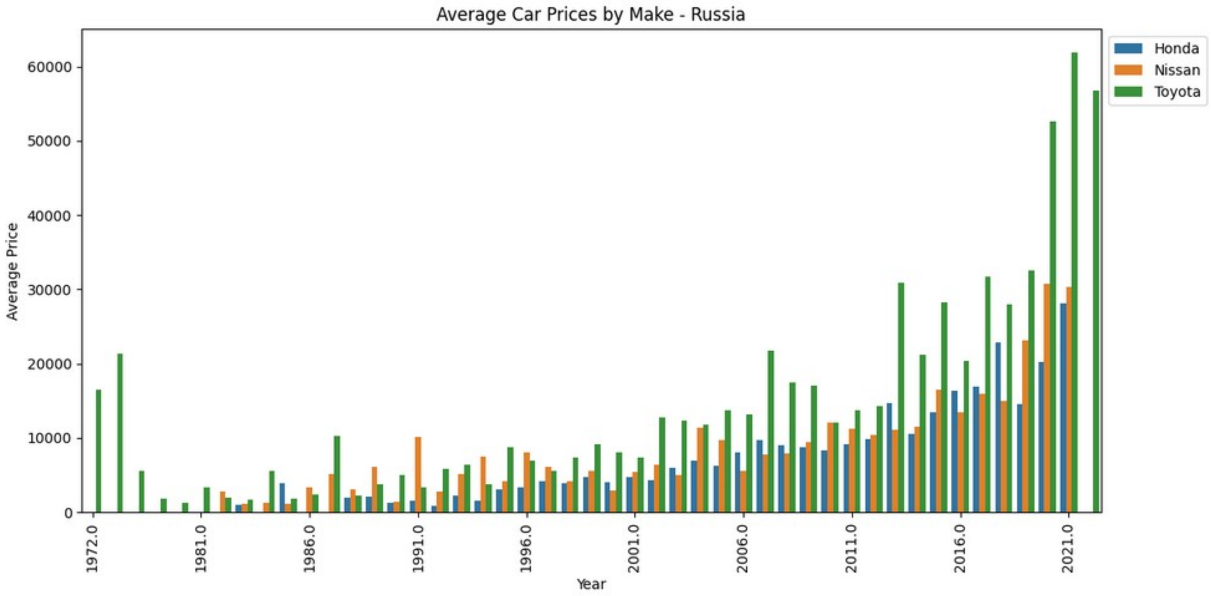



Median Car Price by Year - Russia
median(price)
0
10000
20000
30000
40000
50000
year
1940
1950
1960
1970
1980
1990
2000
2010
2020

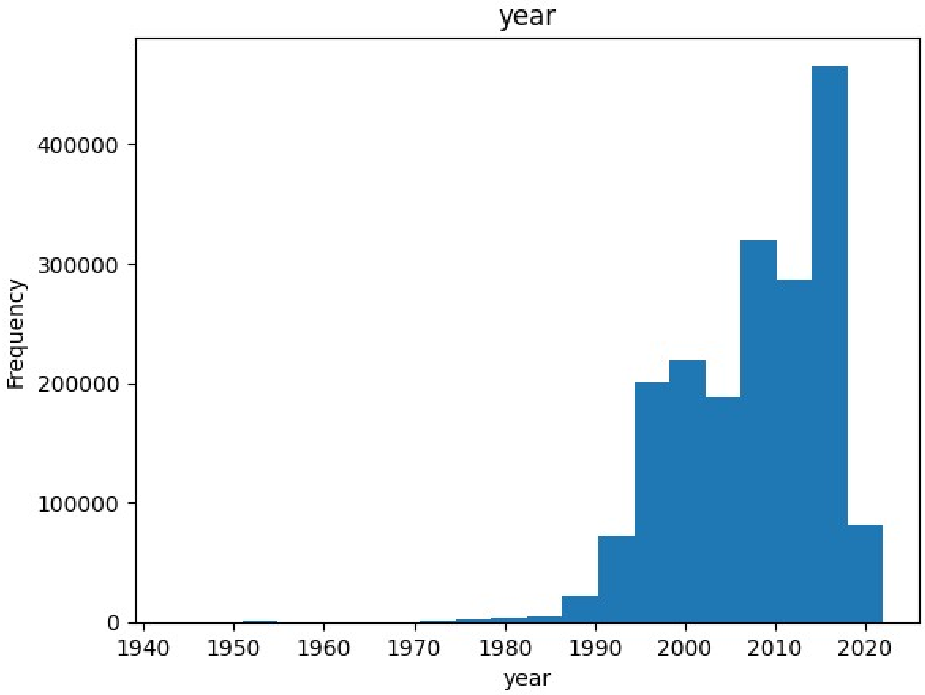
9. Average price for fuel type

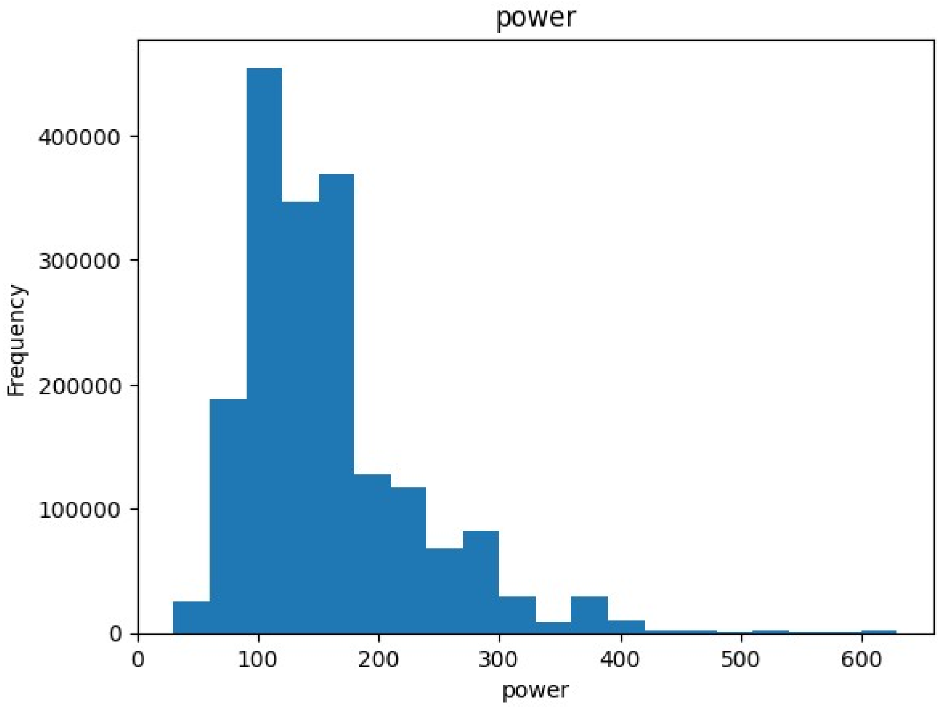


10. Average price for brand



11. Check if data follow a normal distribution (examples)

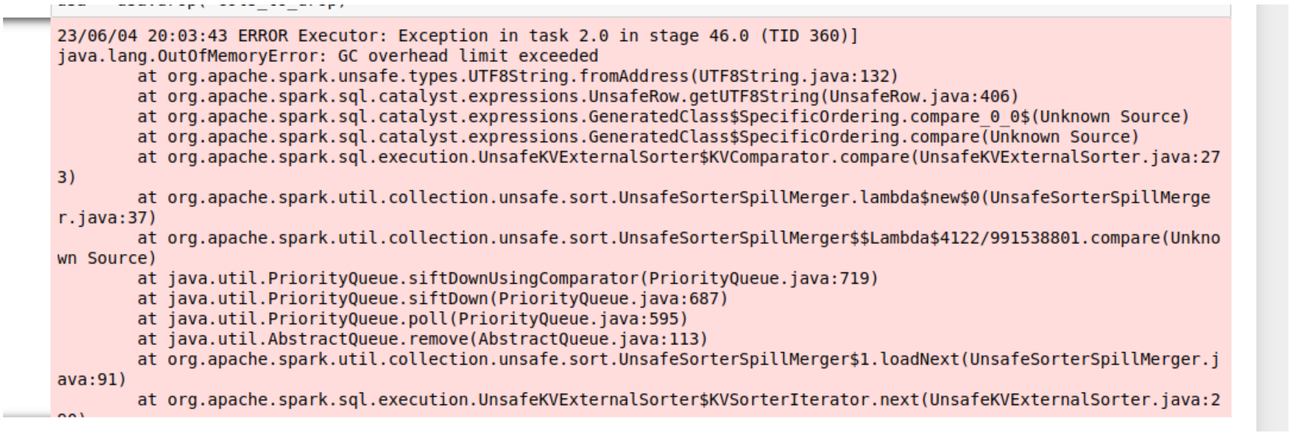




Generally, the dataset is skewed and doesn't follow the normal distribution.

12. The lack of some USA plots and problem with running out of the memory.

During the project I encountered the following error:



```
23/06/04 20:03:43 ERROR Executor: Exception in task 2.0 in stage 46.0 (TID 360)]
java.lang.OutOfMemoryError: GC overhead limit exceeded
        at org.apache.spark.unsafe.types.UTF8String.fromAddress(UTF8String.java:132)
        at org.apache.spark.sql.catalyst.expressions.UnsafeRow.getUTF8String(UnsafeRow.java:406)
        at org.apache.spark.sql.catalyst.expressions.GeneratedClass$SpecificOrdering.compare_0_0$(Unknown Source)
        at org.apache.spark.sql.catalyst.expressions.GeneratedClass$SpecificOrdering.compare(Unknown Source)
        at org.apache.spark.sql.execution.UnsafeKVExternalSorter$KVComparator.compare(UnsafeKVExternalSorter.java:27
3)
        at org.apache.spark.util.collection.unsafe.sort.UnsafeSorterSpillMerger.lambda$new$0(UnsafeSorterSpillMerge
r.java:37)
        at org.apache.spark.util.collection.unsafe.sort.UnsafeSorterSpillMerger$$Lambda$4122/991538801.compare(Unkno
wn Source)
        at java.util.PriorityQueue.siftDownUsingComparator(PriorityQueue.java:719)
        at java.util.PriorityQueue.siftDown(PriorityQueue.java:687)
        at java.util.PriorityQueue.poll(PriorityQueue.java:595)
        at java.util.AbstractQueue.remove(AbstractQueue.java:113)
        at org.apache.spark.util.collection.unsafe.sort.UnsafeSorterSpillMerger$1.loadNext(UnsafeSorterSpillMerger.j
ava:91)
        at org.apache.spark.sql.execution.UnsafeKVExternalSorter$KVSorterIterator.next(UnsafeKVExternalSorter.java:2
```

This means that there is not enough memory to prcess the data. It happened several times during processing of the USA dataset, which is significantly bigger.

It's known that Spark is memory-centric. That out of memory exception can occur at the Driver or Executor level. The Driver is a Java process, where the main() runs. It submits jobs, converts jobs to stages and coordinates tasks between executors, creates a SparkSession etc. Executors are launched at the start of a Spark Application with the help of luster Manager. It runs an individual task and returns the result to the Driver. It can also persist data in the worker nodes for re-usability.

This kind of error can occur due to incorrect usage of Spark. The driver is only an orchestrator/coordinator, the executors are provided with more memory to operate. We can even set a proper limit using spark.driver.maxResultSize.

Out of memory at the Executor level on the other hand, occurs usually because of either inefficient queries, high concurrency or incorrect configuration.

In case of high concurrency, there will be inappropriate number of spark cores for our executors, and we will have to process too many partitions. All these will be running in parallel and will have their memory overhead; therefore, they need the executor memory and can probably cause OutOfMemory errors. To fix this, we can configure spark.default.parallelism and spark.executor.cores and based on your requirement you can decide the numbers.

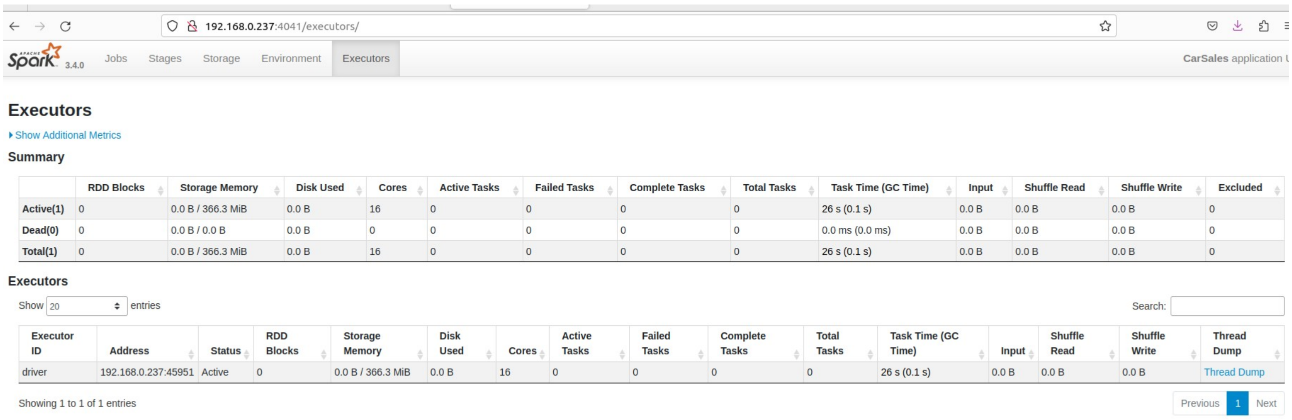
In case of incorrect configuration, there is a possibility that the application fails due to YARN memory overhead issue(if Spark is running on YARN). Therefore, based on each requirement, the configuration has to be done properly so that output does not spill on the disk. Configuring memory using spark.yarn.executor.memoryOverhead will help to resolve this.

Since I'm configuring Spark application like this:

```
import os
memory = '20g'
pyspark_submit_args = ' --driver-memory ' + memory + ' pyspark-shell'
pyspark_submit_args = ' --executor-memory ' + memory + ' pyspark-shell'
os.environ["PYSPARK_SUBMIT_ARGS"] = pyspark_submit_args

spark = SparkSession.builder \
   .config("spark.executor.instances", "4") \
   .appName("CarSales") \
    .getOrCreate()
```

Iin my case the reason of the error is that Spark in Jupyter deos not read those configurations properly, as we can see going to the Spark UI after configuration of the app:



192.168.0.237:4041/executors/

Executors

Show Additional Metrics

Summary

| | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Excluded |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Active(1) | 0 | 0.0 B / 366.3 MiB | 0.0 B | 16 | 0 | 0 | 0 | 0 | 26 s (0.1 s) | 0.0 B | 0.0 B | 0.0 B | 0 |
| Dead(0) | 0 | 0.0 B / 0.0 B | 0.0 B | 0 | 0 | 0 | 0 | 0 | 0.0 ms (0.0 ms) | 0.0 B | 0.0 B | 0.0 B | 0 |
| Total(1) | 0 | 0.0 B / 366.3 MiB | 0.0 B | 16 | 0 | 0 | 0 | 0 | 26 s (0.1 s) | 0.0 B | 0.0 B | 0.0 B | 0 |

Executors

Show 20 entries

Search:

| Executor ID | Address | Status | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Thread Dump |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| driver | 192.168.0.237:45951 | Active | 0 | 0.0 B / 366.3 MiB | 0.0 B | 16 | 0 | 0 | 0 | 0 | 26 s (0.1 s) | 0.0 B | 0.0 B | 0.0 B | Thread Dump |

Showing 1 to 1 of 1 entries

It is clear that I have only the driver, no executors. The steps I would take to solve this issue would start with installing Spark from the command line, and configuring it carefully, not like in this case I just !pip installed it in the Jupyter Notebook.

## 13. Results of the predictions of the Linear Regression model

```
MAE for rus model: 4663.33656174515
RMSE for rus model: 9984.815625262503
```

The predictions given by the model are far from satisfactory. Steps that could be taken in order to obtain better results would consist of: normalizing numerical data, reemoving poorly represented areas of features (just as I have done in case of the year column- removing representation under 2010). From this approach however, we can deduce 2 conclusions. First, it is hard to predict prices on the market so diverse as it is in Russia- world's biggest country. Second, it is rpobably impossible to obtain a good model, from such a diversed dataset, containing underrepresented areas. It would most likely possible however, to narrow this dataset to e.g. years 2010-2015, brands to 5 top selling etc., and then obtain a good-performing model.