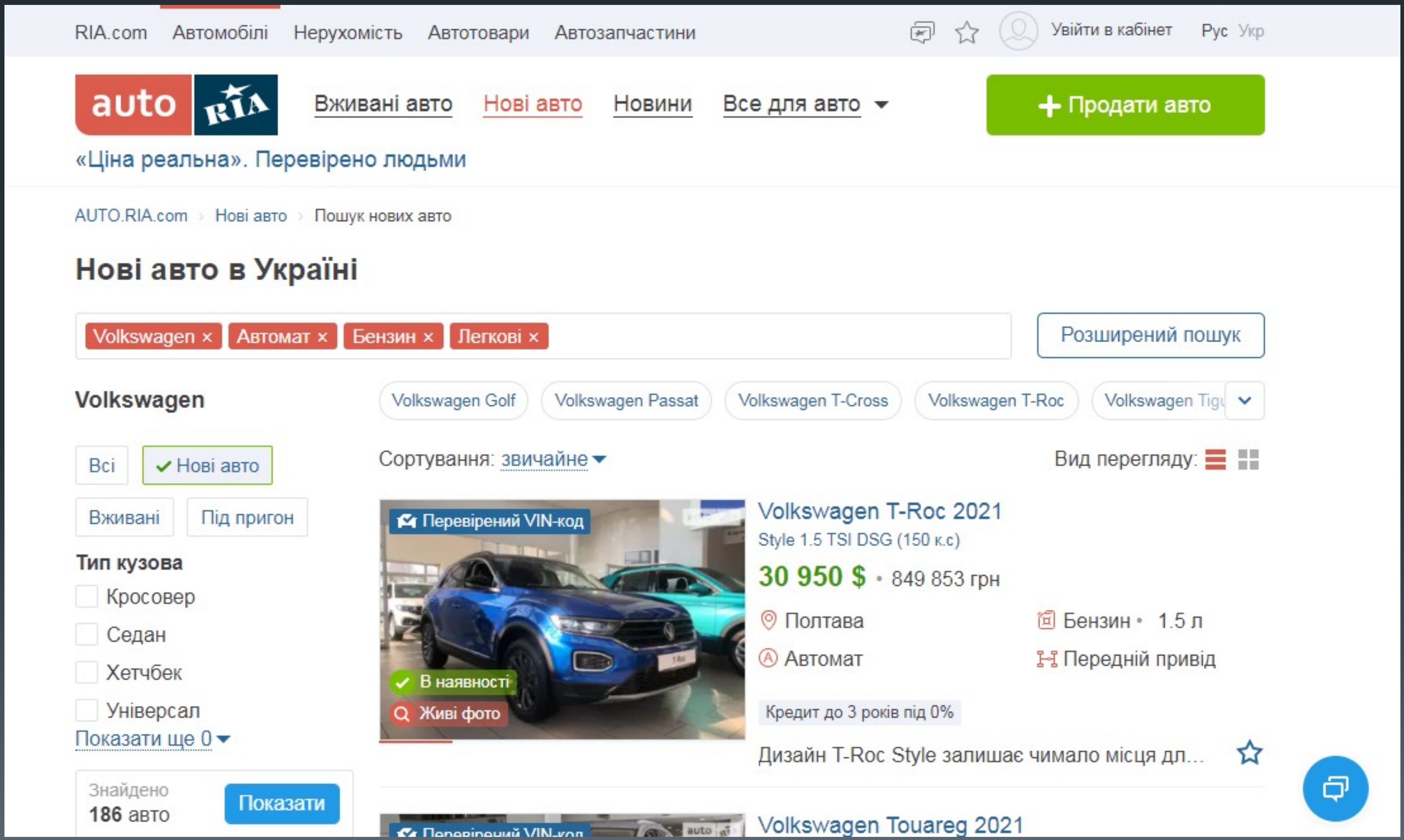
Car price prediction based on model algorithms using the real data from the website auto.ria.com

Input Data:

- NEW cars data from the Ukrainian website Auto.ria.com (parsing of 352 pages with new cars, 6393 objects in total):



Input Data

- Only new cars were selected for analysis in order to equalize the criterion of the condition of cars, which affects the price (for example, in used cars, in addition to mileage, the price is also affected by the general condition of the car, which cannot be described in specific parameters)

Selected parameters for evaluation:

- Brand
- Country of production
- Production year
- Engine_power
- Fuel & tank_volume
- Gear_shift_box
- Drive_train
- Price

Database

- Database with 6393 rows:

Таблица: cars

Filter in any column

	link	brand	country	year	engine_power	fuel	tank_volume	gear_shift_box	drive_train	price
		Фильтр	Фильтр	Фил...	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр
1	a.com/uk/newauto/auto-volkswage...	Volkswagen	Germany	2021.0	231.0	Дизель	3.0	Автомат	Повний	61300.0
2	a.com/uk/newauto/auto-bentley-...	Bentley	Great Britain	2021.0	550.0	Бензин	4.0	Автомат	Повний	355100.0
3	a.com/uk/newauto/auto-bentley-...	Bentley	Great Britain	2021.0	550.0	Бензин	4.0	Автомат	Повний	375002.0
4	a.com/uk/newauto/auto-mercedes...	Mercedes-Benz	Germany	2020.0	435.0	Бензин	3.0	Автомат	Повний	192711.0
5	a.com/uk/newauto/auto-mercedes...	Mercedes-Benz	Germany	2021.0	330.0	Дизель	2.9	Автомат	Повний	116866.0
6	a.com/uk/newauto/auto-mercedes...	Mercedes-Benz	Germany	2020.0	435.0	Бензин	3.0	Автомат	Повний	119529.0
7	a.com/uk/newauto/auto-mercedes...	Mercedes-Benz	Germany	2020.0	194.0	Дизель	1.9	Автомат	Повний	74567.0
8	a.com/uk/newauto/auto-volkswage...	Volkswagen	Germany	2021.0	286.0	Дизель	3.0	Автомат	Повний	79150.0
9	a.com/uk/newauto/auto-mercedes...	Mercedes-Benz	Germany	2021.0	245.0	Дизель	1.9	Автомат	Повний	81038.0
10	a.com/uk/newauto/auto-mercedes...	Mercedes-Benz	Germany	2021.0	163.0	Дизель	1.9	Автомат	Повний	66803.0
11	a.com/uk/newauto/auto-mercedes...	Mercedes-Benz	Germany	2021.0	194.0	Дизель	1.9	Автомат	Повний	75304.0
12	a.com/uk/newauto/auto-mini-...	MINI	Great Britain	2021.0	190.0	Дизель	2.0	Автомат	Повний	50219.0
13	a.com/uk/newauto/auto-mercedes...	Mercedes-Benz	Germany	2020.0	197.0	Бензин	2.0	Автомат	Повний	75304.0
14	a.com/uk/newauto/auto-mercedes...	Mercedes-Benz	Germany	2021.0	330.0	Дизель	2.9	Автомат	Повний	221131.0
15	a.com/uk/newauto/auto-volvo-...	Volvo	Sweden	2021.0	235.0	Гібрид	2.0	Автомат	Повний	70618.0
16	a.com/uk/newauto/auto-mini-...	MINI	Great Britain	2020.0	192.0	Бензин	2.0	Механічна	Передній	37647.0
17	a.com/uk/newauto/auto-...	Lamborghini	Italy	2021.0	650.0	Бензин	4.0	Автомат	Повний	425527.0
18	a.com/uk/newauto/auto-citroen-	Citroen	France	2021.0	130.0	Дизель	1.5	Автомат	Передній	27308.0

1 - 19 из 6393 Перейти к: 1

Working with a DataFrame

- My DataFrame:

```
В [294]: df.head()
```

Out[294]:

	link	brand	country	year	engine_power	fuel	tank_volume	gear_shift_box	drive_train	price
0	https://auto.ria.com/uk/newauto/auto-volkswage...	Volkswagen	Germany	2021.0	231.0	Дизель	3.0	Автомат	Повний	61300.0
1	https://auto.ria.com/uk/newauto/auto-bentley-c...	Bentley	Great Britain	2021.0	550.0	Бензин	4.0	Автомат	Повний	355100.0
2	https://auto.ria.com/uk/newauto/auto-bentley-f...	Bentley	Great Britain	2021.0	550.0	Бензин	4.0	Автомат	Повний	375002.0
3	https://auto.ria.com/uk/newauto/auto-mercedes-...	Mercedes-Benz	Germany	2020.0	435.0	Бензин	3.0	Автомат	Повний	192711.0
4	https://auto.ria.com/uk/newauto/auto-mercedes-...	Mercedes-Benz	Germany	2021.0	330.0	Дизель	2.9	Автомат	Повний	116866.0

```
В [295]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6393 entries, 0 to 6392
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   link            6393 non-null   object 
 1   brand           6393 non-null   object 
 2   country         6230 non-null   object 
 3   year            6393 non-null   float64
 4   engine_power    6393 non-null   float64
 5   fuel            6393 non-null   object 
 6   tank_volume     6393 non-null   float64
 7   gear_shift_box  6393 non-null   object 
 8   drive_train     6393 non-null   object 
 9   price           6393 non-null   float64
dtypes: float64(4), object(6)
memory usage: 499.6+ KB
```

```
В [296]: df.describe()
```

Out[296]:

	year	engine_power	tank_volume	price
count	6393.000000	6393.000000	6393.000000	6.393000e+03
mean	2020.668387	182.301111	1.910605	5.126554e+04
std	0.525772	116.109321	0.816865	7.120338e+04
min	2017.000000	67.000000	0.000000	7.477000e+03
25%	2020.000000	117.000000	1.500000	2.069500e+04
50%	2021.000000	147.000000	1.600000	2.845600e+04
75%	2021.000000	194.000000	2.000000	4.660200e+04
max	2021.000000	800.000000	6.700000	1.398503e+06

Working with a DataFrame

- Correcting zero values, converting object parameters to numeric ones (fuel, drive_train, gear_shift_box):

```
В [80]: df['fuel'].value_counts()

Out[80]: Бензин        3977
         Дизель        1978
         Гібрид         347
         Електро         87
         Бензин/Газ       4
         Name: fuel, dtype: int64
```

```
В [119]: df['fuel'] = df['fuel'].replace(['Гібрид'],'5.0')
          df['fuel'] = df['fuel'].replace(['Електро'],'4.0')
          df['fuel'] = df['fuel'].replace(['Бензин/Газ'],'3.0')
          df['fuel'] = df['fuel'].replace(['Дизель'],'2.0')
          df['fuel'] = df['fuel'].replace(['Бензин'],'1.0')
```

```
df['drive_train'].unique()

array(['Повний', 'Передній', 'Задній'], dtype=object)
```

```
df['drive_train'] = df['drive_train'].replace(['Повний'],'3.0')
df['drive_train'] = df['drive_train'].replace(['Передній'],'2.0')
df['drive_train'] = df['drive_train'].replace(['Задній'],'1.0')
```

- I got a table with 6342 rows;

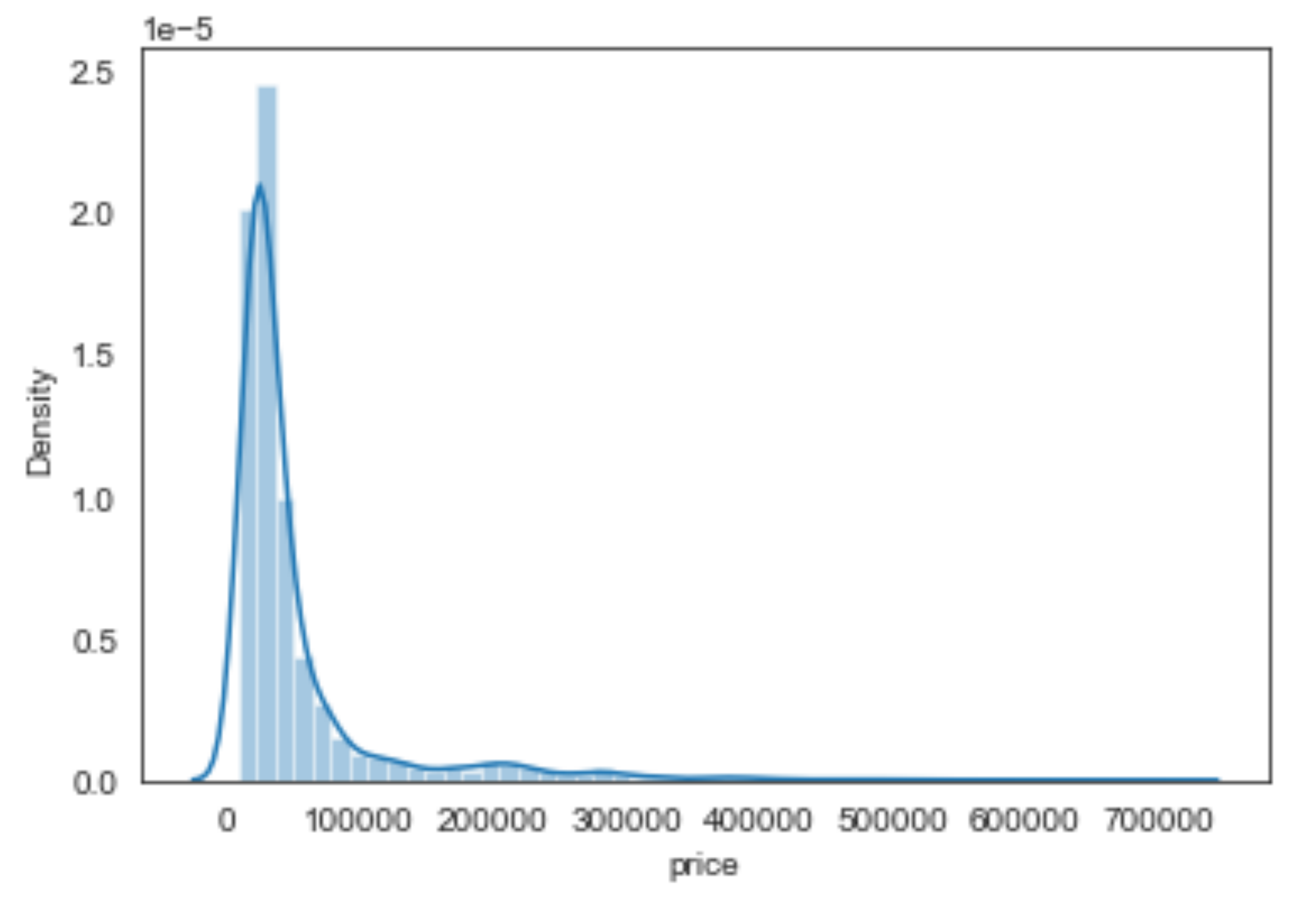
I left the car brand and country of production in the object type for the convenience of graphical data analysis;

set the price up to $800,000

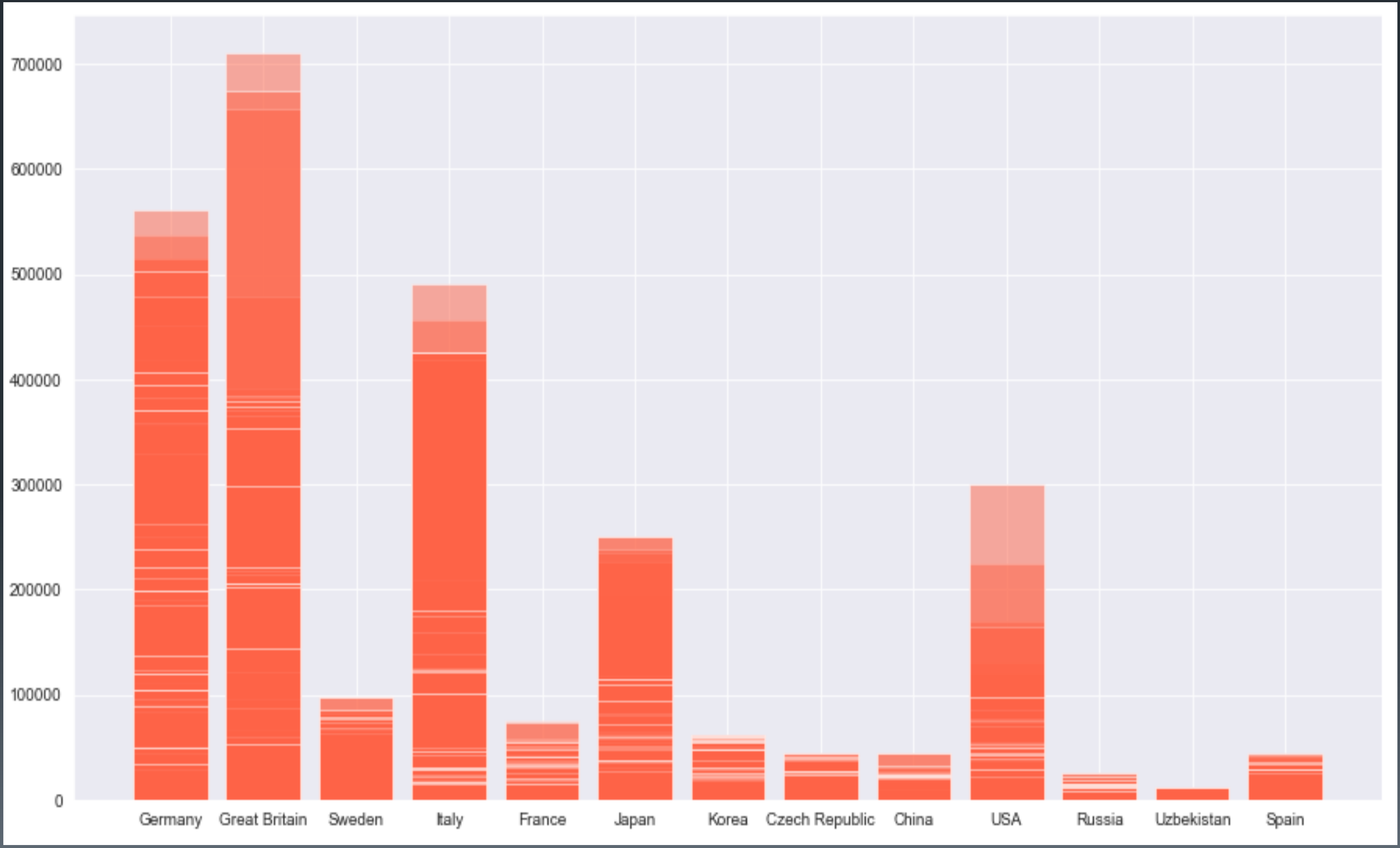
```
df['gear_shift_box'].value_counts()

Автомат        3602
Механічна      1675
Варіатор        920
Роботизована    129
Тіптронік        59
Редуктор          8
Name: gear_shift_box, dtype: int64
```

```
df['gear_shift_box'] = df['gear_shift_box'].replace(['Варіатор'],'6.0')
df['gear_shift_box'] = df['gear_shift_box'].replace(['Автомат'],'5.0')
df['gear_shift_box'] = df['gear_shift_box'].replace(['Тіптронік'],'4.0')
df['gear_shift_box'] = df['gear_shift_box'].replace(['Роботизована'],'3.0')
df['gear_shift_box'] = df['gear_shift_box'].replace(['Механічна'],'2.0')
df['gear_shift_box'] = df['gear_shift_box'].replace(['Редуктор'],'1.0')
```

Data analysis
Price distribution



Car pricing by country of production



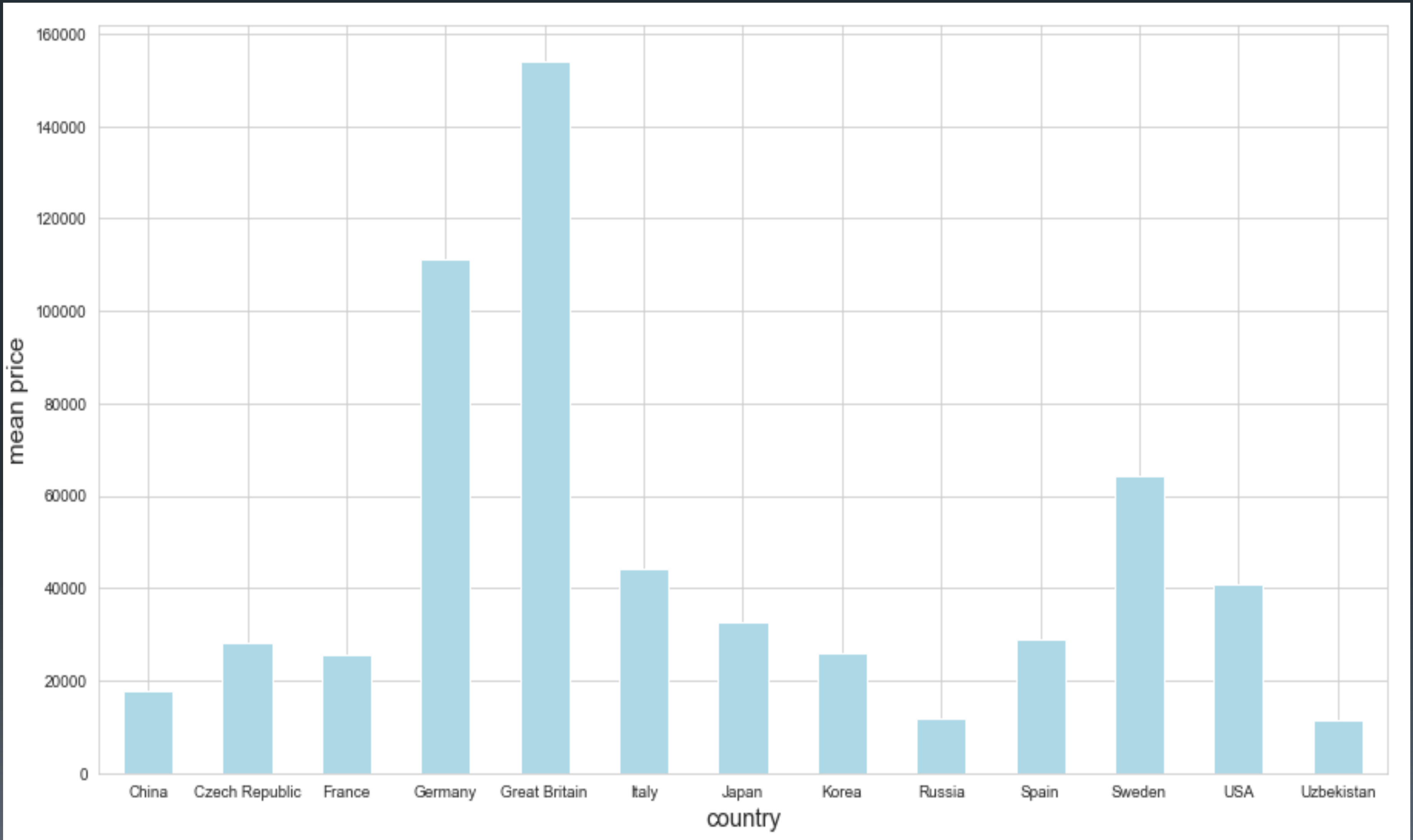
*Groups of countries and car brands included in their group:

country	brand	brand count
China	Chery	290
	FAW	14
	Haval	27
	JAC	28
	Jetour	36
Czech Republic	Skoda	105
France	Citroen	186
	DS	12
	Peugeot	423
	Renault	264
Germany	Audi	88
	BMW	82
	Mercedes-Benz	604
	Opel	138
	Porsche	98
	Volkswagen	306
Great Britain	Bentley	12
	Jaguar	37
	Land Rover	152
	MINI	4
	Rolls-Royce	3
Italy	Alfa Romeo	11
	Ferrari	1
	Fiat	172
	Iveco	2
	Lamborghini	5
	Maserati	21
Japan	Acura	1
	Honda	79
	Infiniti	25
	Isuzu	2
	Lexus	36
	Mazda	265
	Mitsubishi	343
	Nissan	317
	Subaru	81
	Suzuki	209
	Toyota	175
Korea	Hyundai	363
	Kia	518
Russia	Lada	204
	ГАЗ	3
	УАЗ	1
Spain	SEAT	28
Sweden	Volvo	102
USA	Cadillac	2
	Chevrolet	5
	Dodge	1
	Ford	347
	GMC	1
	Jeep	22
	Lincoln	1
Uzbekistan	Ravon	88

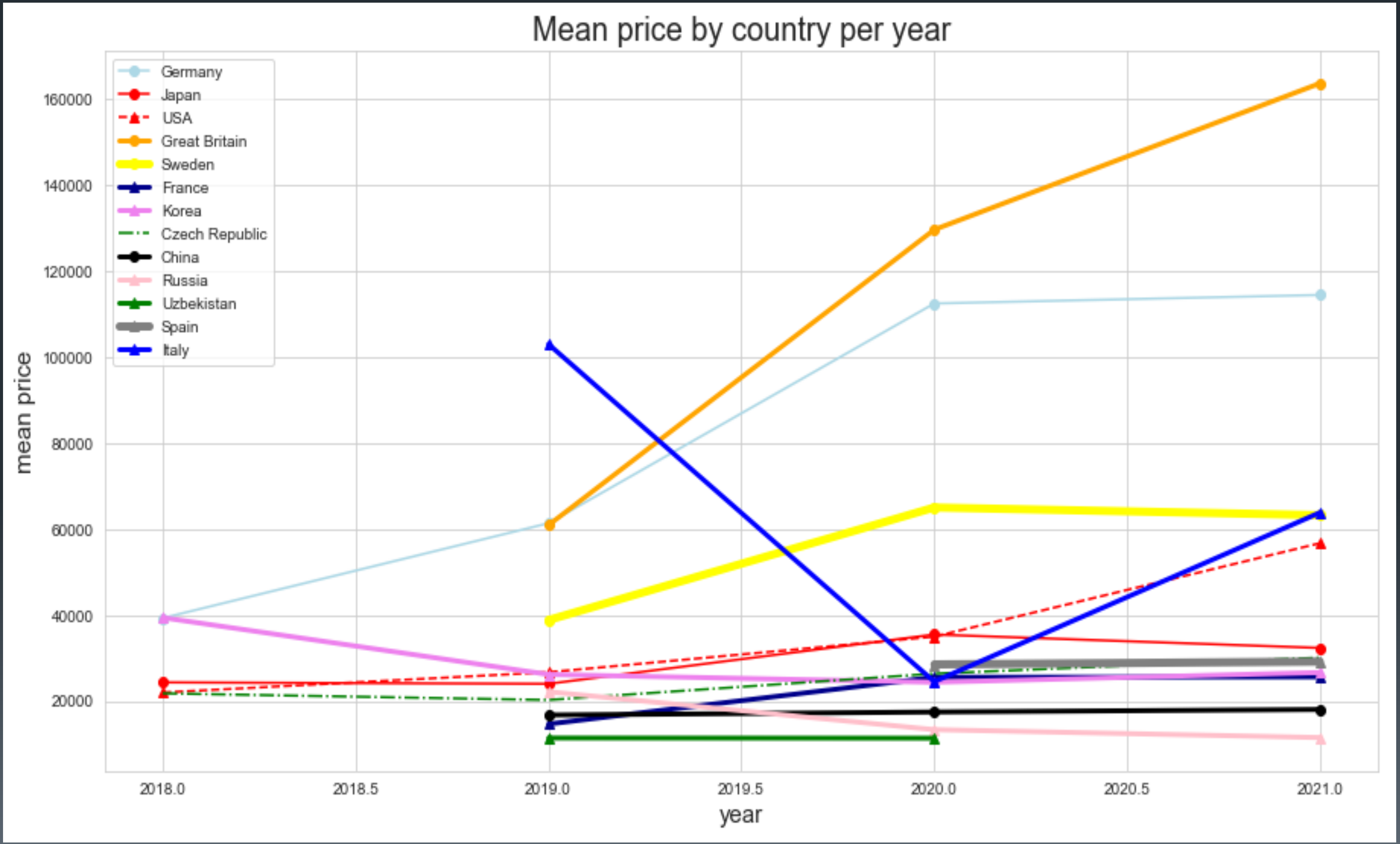
Car pricing by country of production

country	price min	price mean	price max
China	10400	17770.52	44604
Czech Republic	15547	28346.30	44455
France	11689	25597.41	74431
Germany	16499	111065.69	561791
Great Britain	37096	154186.82	711204
Italy	12096	44185.13	490444
Japan	15946	32837.60	251013
Korea	13500	26063.26	62280
Russia	7477	11895.32	26045
Spain	17227	28904.04	43998
Sweden	38847	64208.91	97441
USA	12988	40751.62	299999
Uzbekistan	9782	11376.10	12320

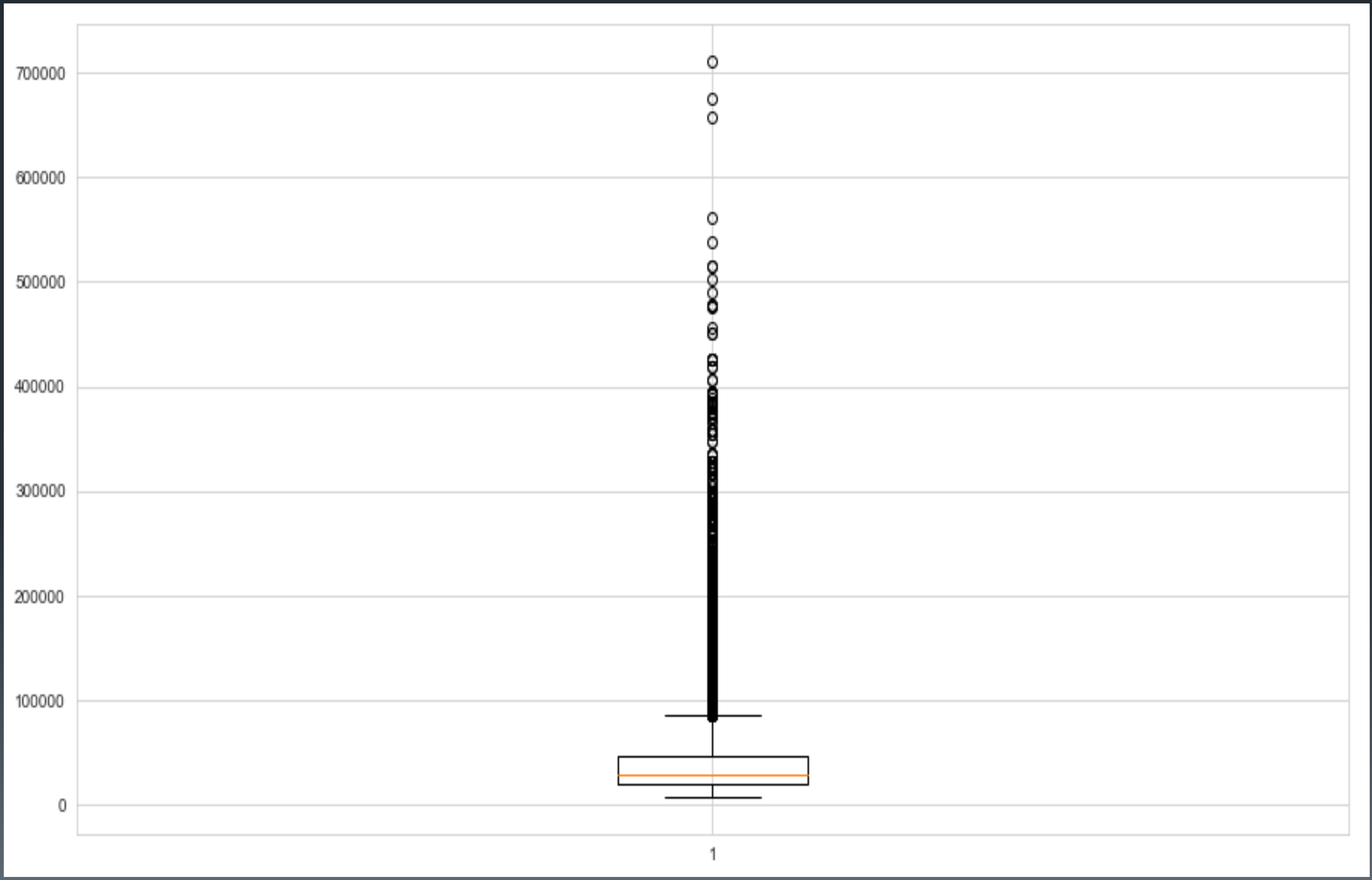
Average car price by country of production



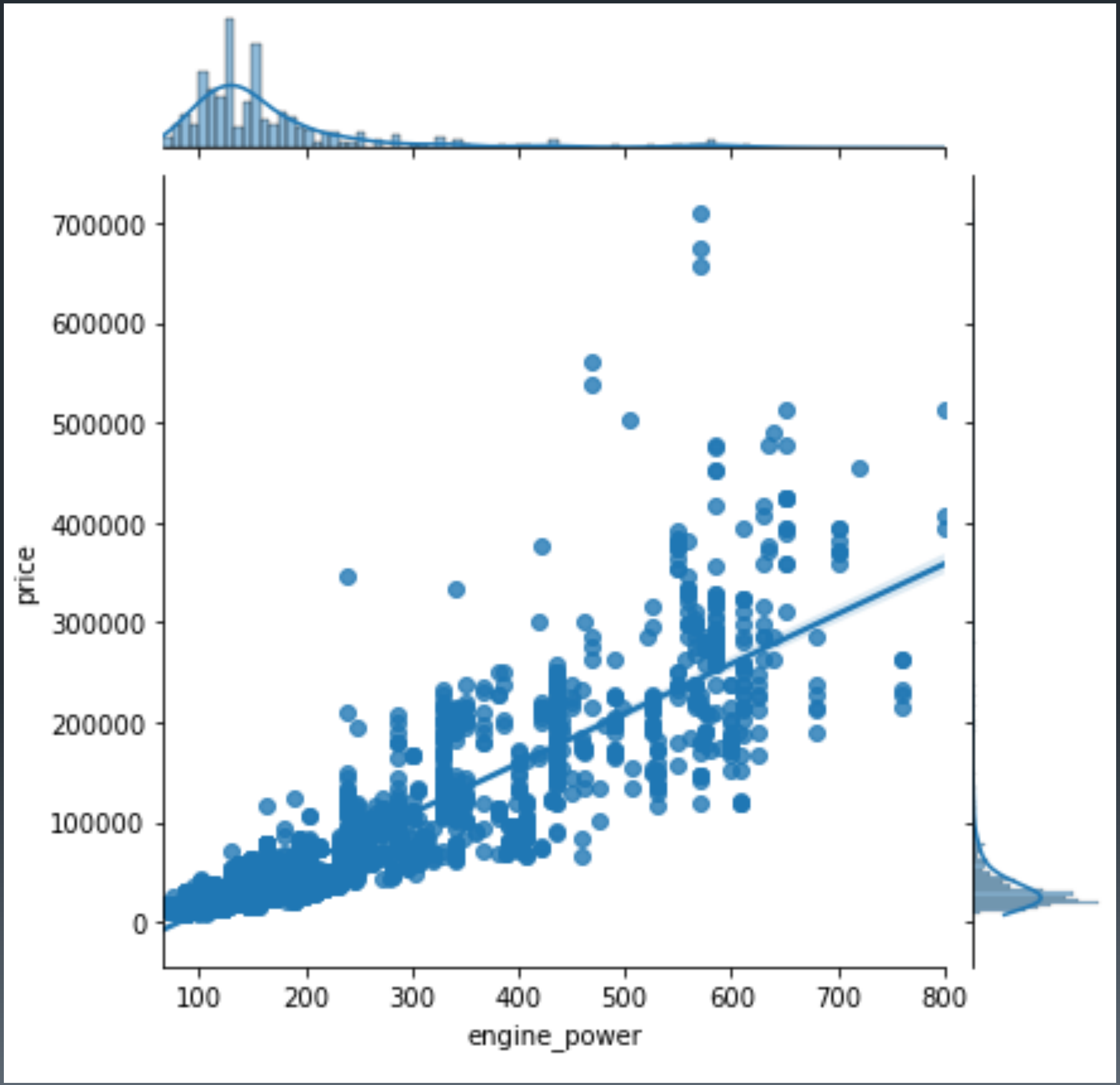
Average car price by country and year of production



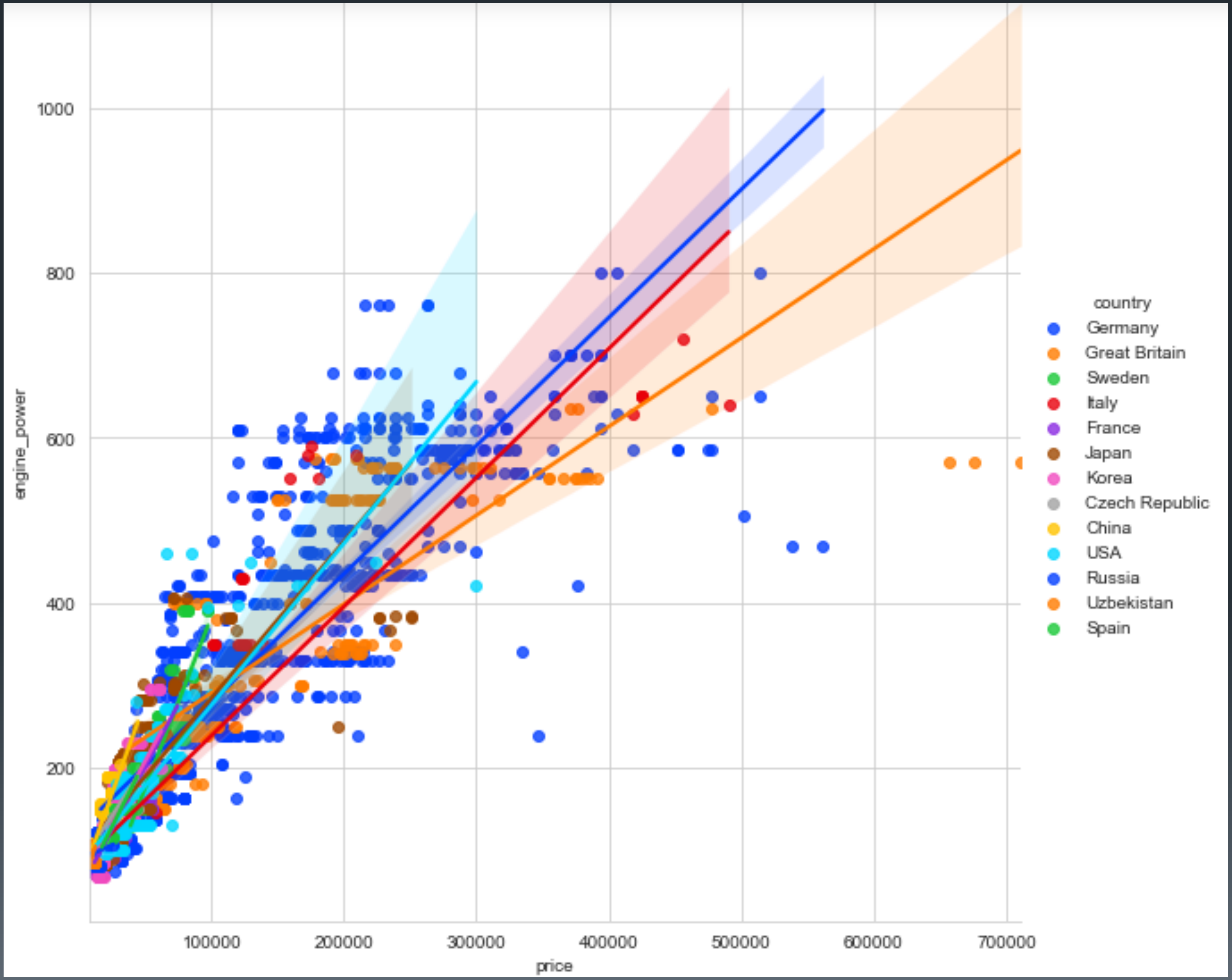
Price analysis with boxplot

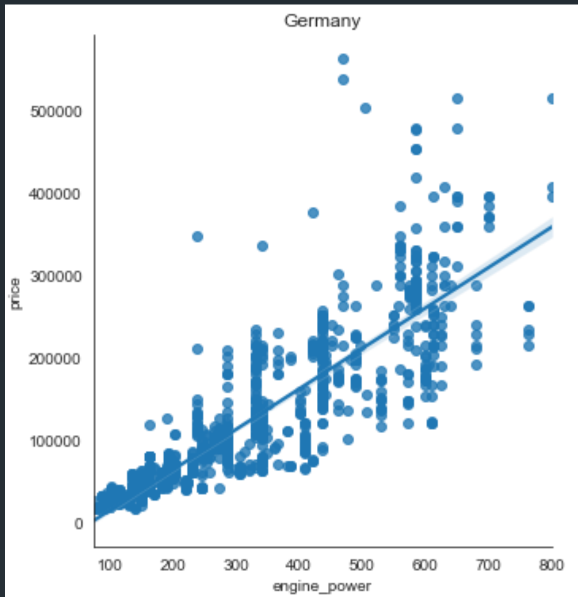


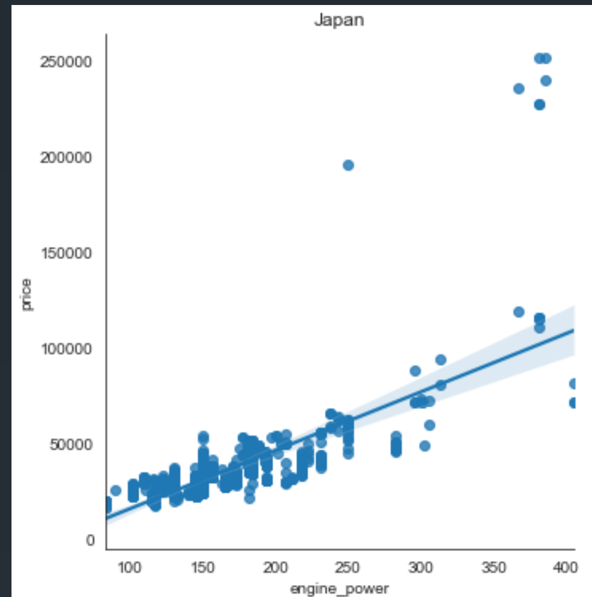
Dependence of price on engine power

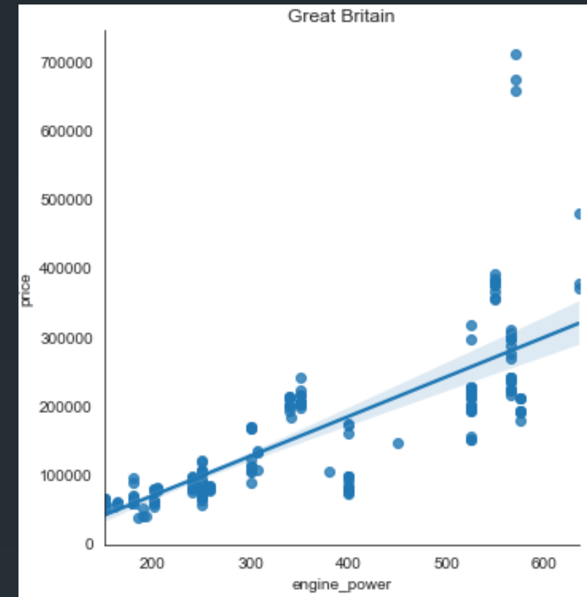


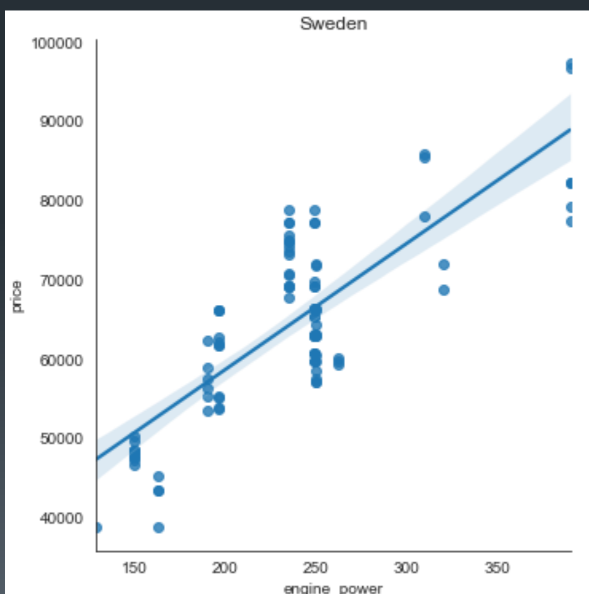
The dependence of the price on the engine power, broken down by country of production

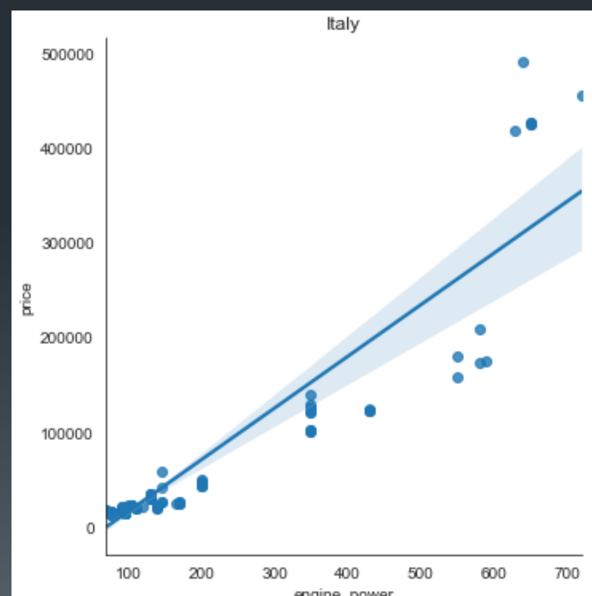


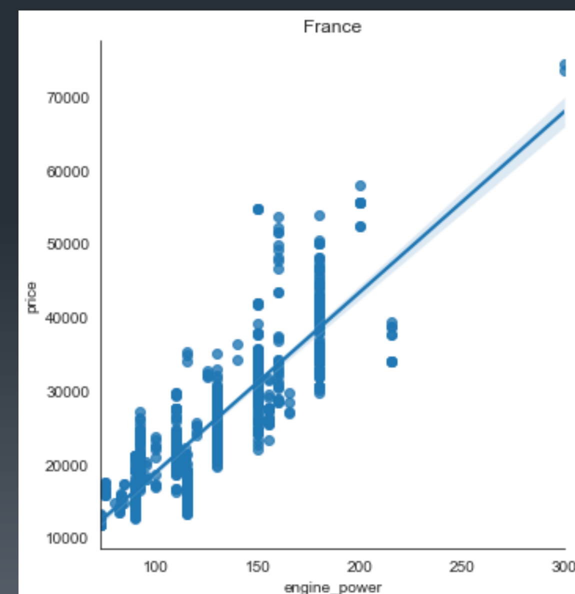

Germany
price
engine_power
0
100000
200000
300000
400000
500000
100
200
300
400
500
600
700
800

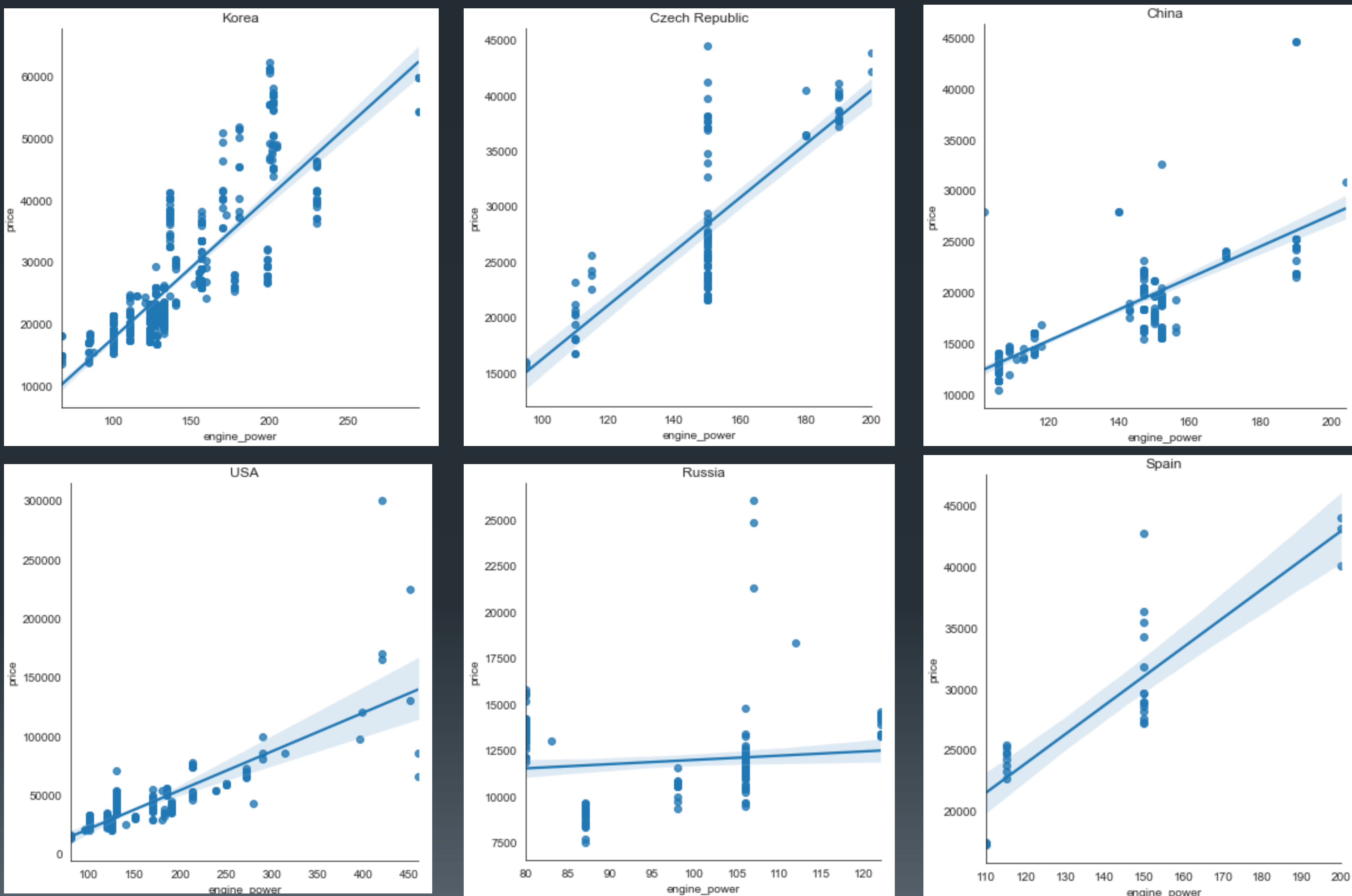

Japan
price
engine_power
0
50000
100000
150000
200000
250000
100
150
200
250
300
350
400


Great Britain
price
engine_power
0
100000
200000
300000
400000
500000
600000
700000
200
300
400
500
600

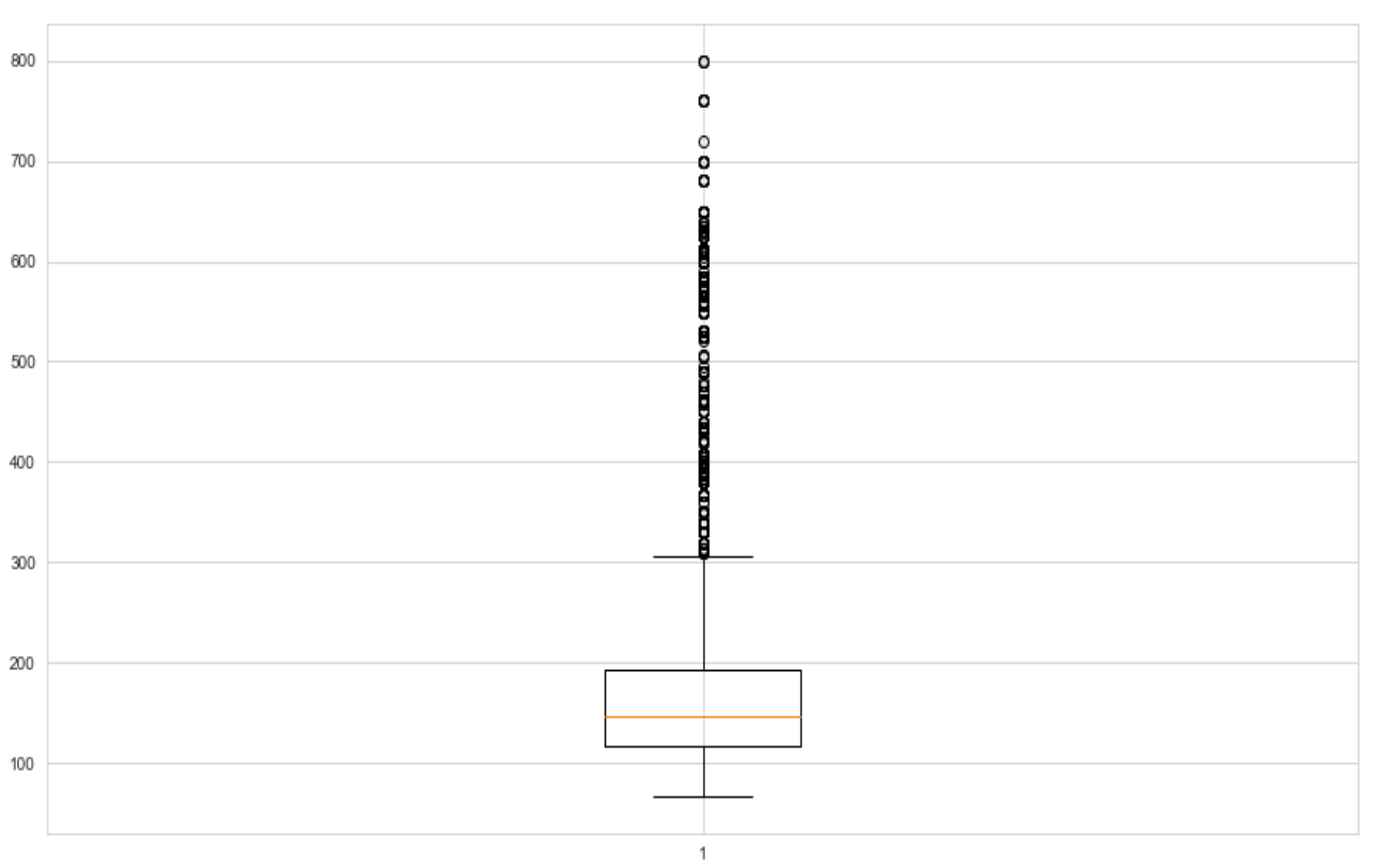

Sweden
price
engine_power
40000
50000
60000
70000
80000
90000
100000
150
200
250
300
350


Italy
price
engine_power
0
100000
200000
300000
400000
500000
100
200
300
400
500
600
700


France
price
engine_power
10000
20000
30000
40000
50000
60000
70000
100
150
200
250
300


Korea
price
60000
50000
40000
30000
20000
10000
100
150
200
250
engine_power
Czech Republic
price
45000
40000
35000
30000
25000
20000
15000
100
120
140
160
180
200
engine_power
China
price
45000
40000
35000
30000
25000
20000
15000
10000
120
140
160
180
200
engine_power
USA
price
300000
250000
200000
150000
100000
50000
0
100
150
200
250
300
350
400
450
engine_power
Russia
price
25000
22500
20000
17500
15000
12500
10000
7500
80
85
90
95
100
105
110
115
120
engine_power
Spain
price
45000
40000
35000
30000
25000
20000
110
120
130
140
150
160
170
180
190
200
engine_power

Engine power analysis with boxplot



Preparing the model

- Next, 2 methods of encoding data by car brand and country of production were performed:

- 1) encoding categorical features by the **LabelEncoder()** method that assign a unique number to each category and replace the feature value with the corresponding number:

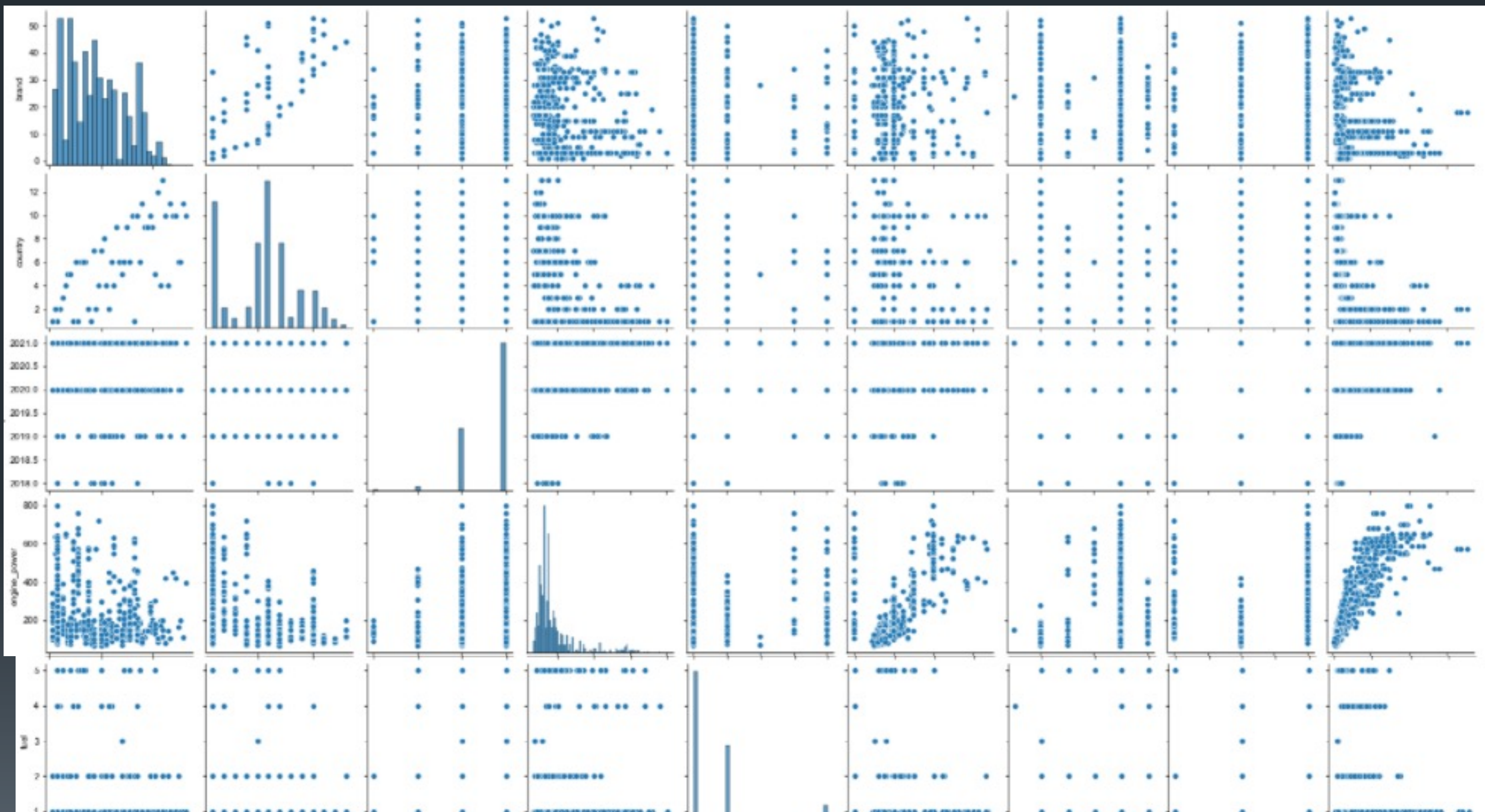
	brand	country	year	engine_power	fuel	tank_volume	gear_shift_box	drive_train	price
0	1.0	1.0	2021	231.0	2.0	3.0	5.0	3.0	61300
1	2.0	2.0	2021	550.0	1.0	4.0	5.0	3.0	355100
2	2.0	2.0	2021	550.0	1.0	4.0	5.0	3.0	375002
3	3.0	1.0	2020	435.0	1.0	3.0	5.0	3.0	192711
4	3.0	1.0	2021	330.0	2.0	2.9	5.0	3.0	116866
...	...	...	...	...	...	...	...	...	...
6388	36.0	11.0	2021	106.0	1.0	1.6	2.0	2.0	11315
6389	36.0	11.0	2021	87.0	1.0	1.6	2.0	2.0	7654
6390	38.0	9.0	2021	147.0	1.0	1.5	5.0	2.0	19900
6391	26.0	9.0	2020	147.0	1.0	1.5	5.0	2.0	22000
6392	31.0	6.0	2021	218.0	5.0	2.5	6.0	2.0	37930

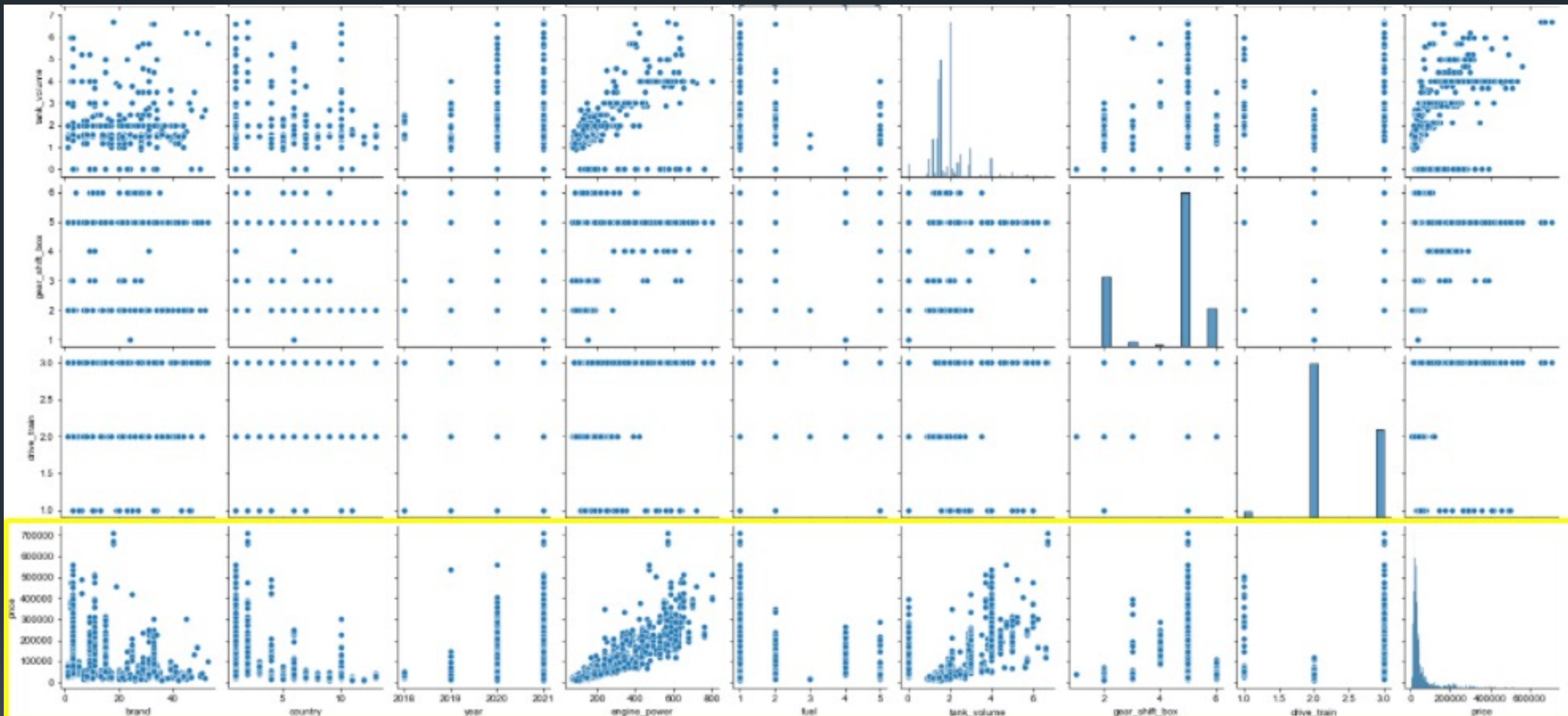
- 2) the **get_dummies** method (turning categorical features into new ones that answer the question whether the car belongs to a certain brand and country or not. With this approach as many new columns appear for each categorical feature as there are possible categories. One of the columns will be filled with 1 and the rest - with 0):

	year	engine_power	fuel	tank_volume	gear_shift_box	drive_train	price	Acura	Alfa Romeo	Audi	...	Germany	Great Britain	Italy	Japan	Korea	Russia	Spain
0	2021	231.0	2.0	3.0	5.0	3.0	61300	0	0	0	...	1	0	0	0	0	0	0
1	2021	550.0	1.0	4.0	5.0	3.0	355100	0	0	0	...	0	1	0	0	0	0	0
2	2021	550.0	1.0	4.0	5.0	3.0	375002	0	0	0	...	0	1	0	0	0	0	0
3	2020	435.0	1.0	3.0	5.0	3.0	192711	0	0	0	...	1	0	0	0	0	0	0
4	2021	330.0	2.0	2.9	5.0	3.0	116866	0	0	0	...	1	0	0	0	0	0	0

- In the new table with 8 columns, it became 73.

A pairplot according to 1 option





- We are only interested in the last line with graphs of price dependence on other parameters

Modeling
Linear Regression

- 1 option (LabelEncoder)

```
model_1.score(X_train,y_train)
```

0.8184907048729514

```
model_1.score(X_test, y_test)
```

0.8355265991027254

```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, predictions_1))
print('MSE:', metrics.mean_squared_error(y_test, predictions_1))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_1)))
```

```
MAE: 14091.60780889322
MSE: 599216655.2981906
RMSE: 24478.902248634244
```

	Coefficient
brand	-99.255871
country	-1223.204132
year	-1562.756368
engine_power	465.020815
fuel	-2531.363785
tank_volume	7579.684452
gear_shift_box	-2350.652990
drive_train	-1749.526459

- 2 option (get_dummies)

```
model_1.score(X_train,y_train)
```

0.8790166448113139

```
model_1.score(X_test, y_test)
```

0.9031410513779253

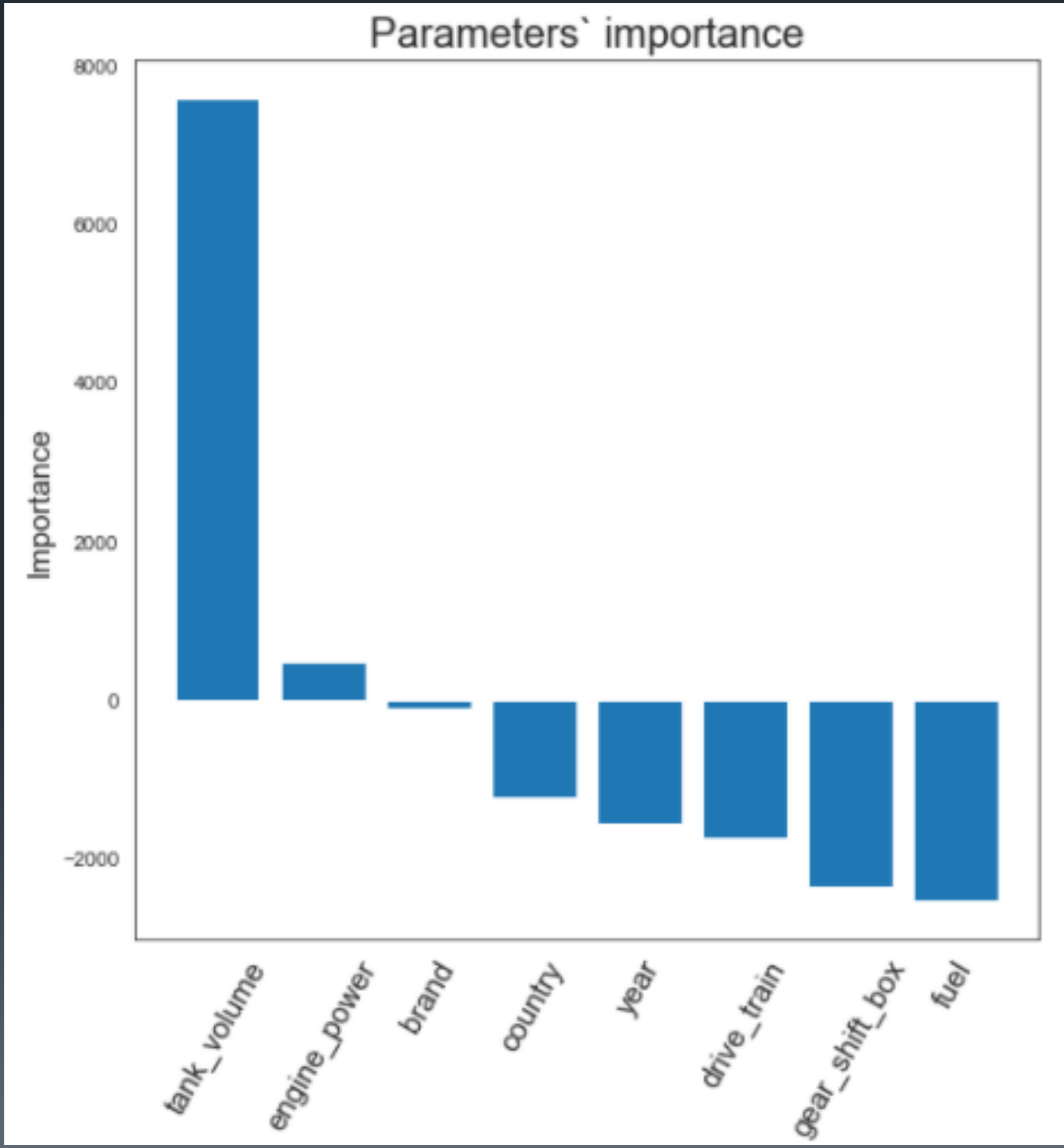
```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, predictions_1))
print('MSE:', metrics.mean_squared_error(y_test, predictions_1))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_1)))
```

```
MAE: 11527.785478838294
MSE: 397621093.9604858
RMSE: 19940.438660182124
```

	Coefficient
year	2305.707284
engine_power	394.896911
fuel	-2137.159067
tank_volume	7607.752652
gear_shift_box	-844.912238
...	...
Russia	-11195.002129
Spain	-6846.217948
Sweden	-6063.095747
USA	-32770.237571
Uzbekistan	-6645.473481

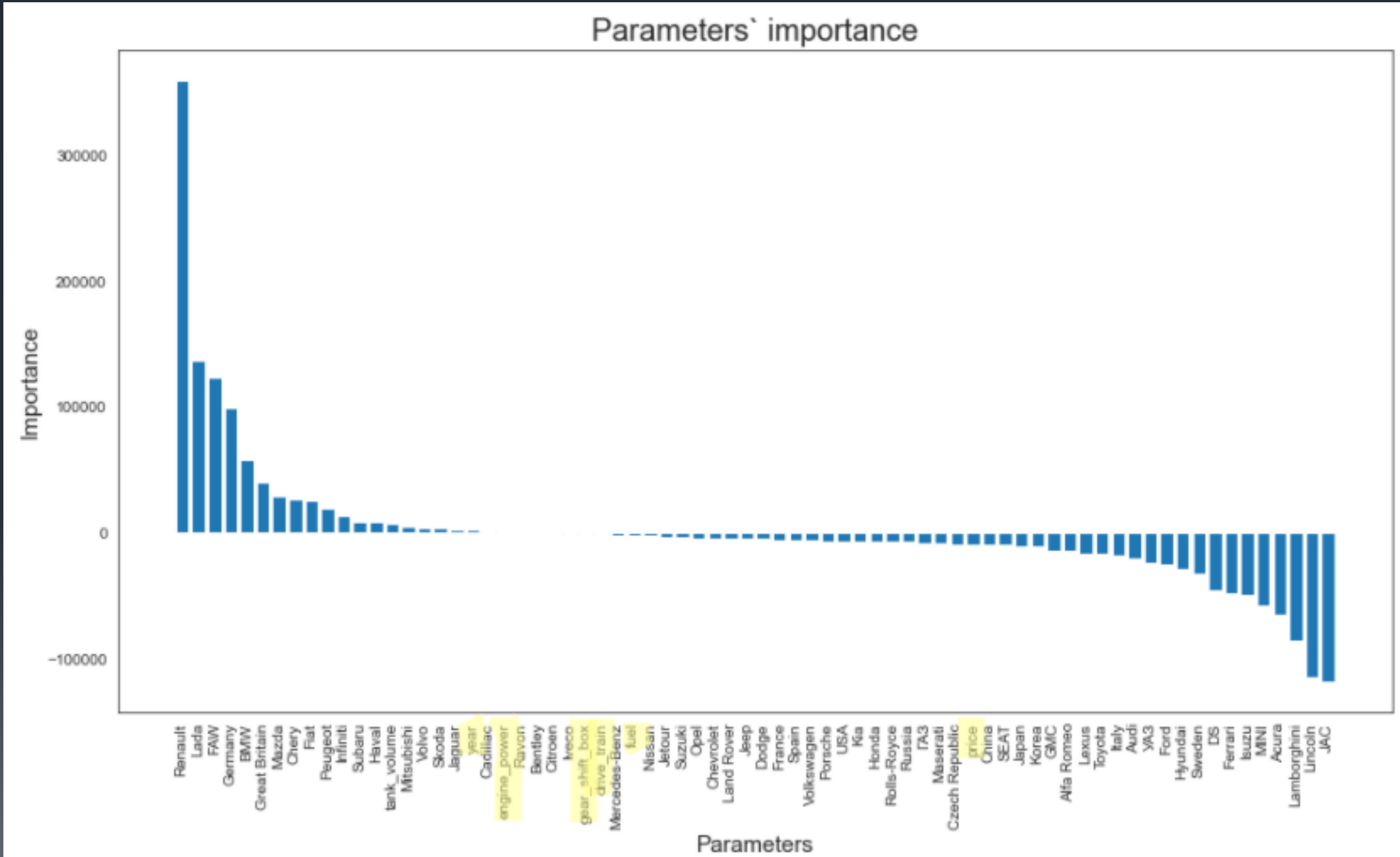
Linear Regression

- 1 option (LabelEncoder)



Linear Regression

- 2 option (get_dummies)



Decision Tree

1 option (LabelEncoder)

```
model_2.score(X_train,y_train)
```

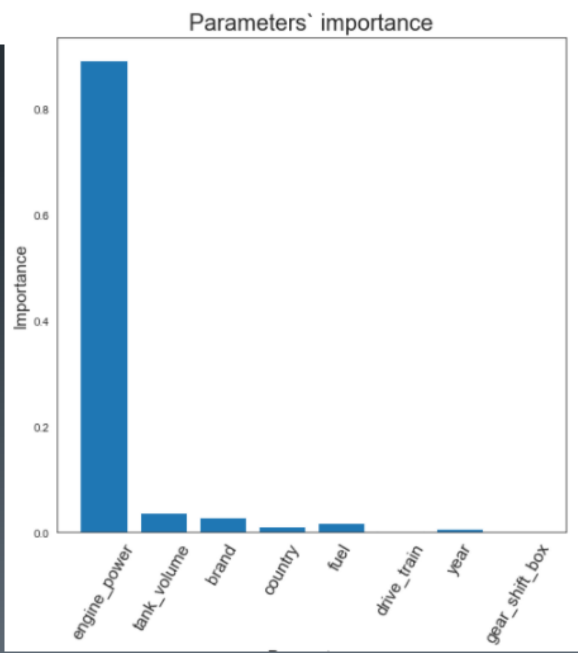
```
0.9730207556536934
```

```
model_2.score(X_test, y_test)
```

```
0.9407030909579164
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions_2))
print('MSE:', metrics.mean_squared_error(y_test, predictions_2))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_2)))
```

```
MAE: 4873.845548611746
MSE: 216033080.80137807
RMSE: 14698.06384532936
```



2 option (get_dummies)

```
model_2.score(X_train,y_train)
```

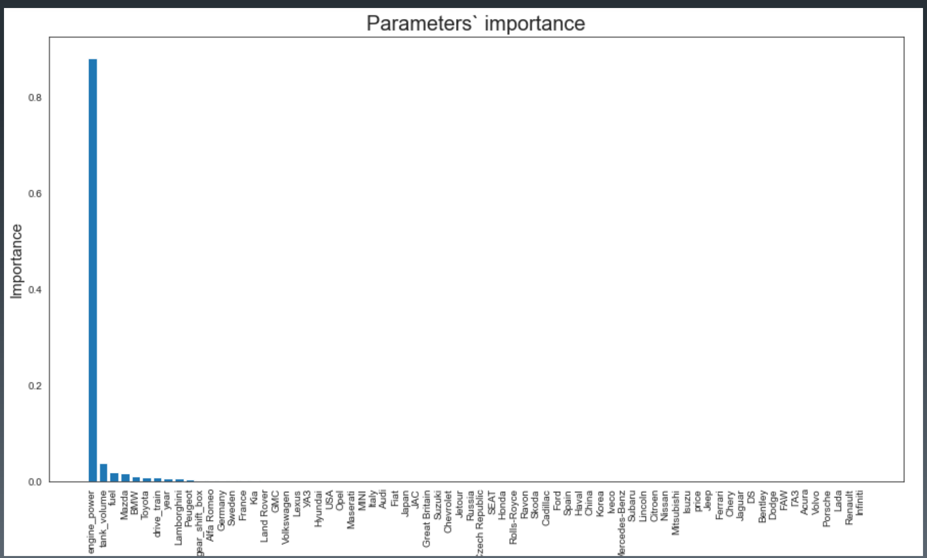
```
0.9725210978626069
```

```
model_2.score(X_test, y_test)
```

```
0.9391317698623047
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions_2))
print('MSE:', metrics.mean_squared_error(y_test, predictions_2))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_2)))
```

```
MAE: 5003.814229460336
MSE: 249873580.07800144
RMSE: 15807.390046367598
```



Random Forest

1 option (LabelEncoder)

```
model_3.score(X_train, y_train)
```

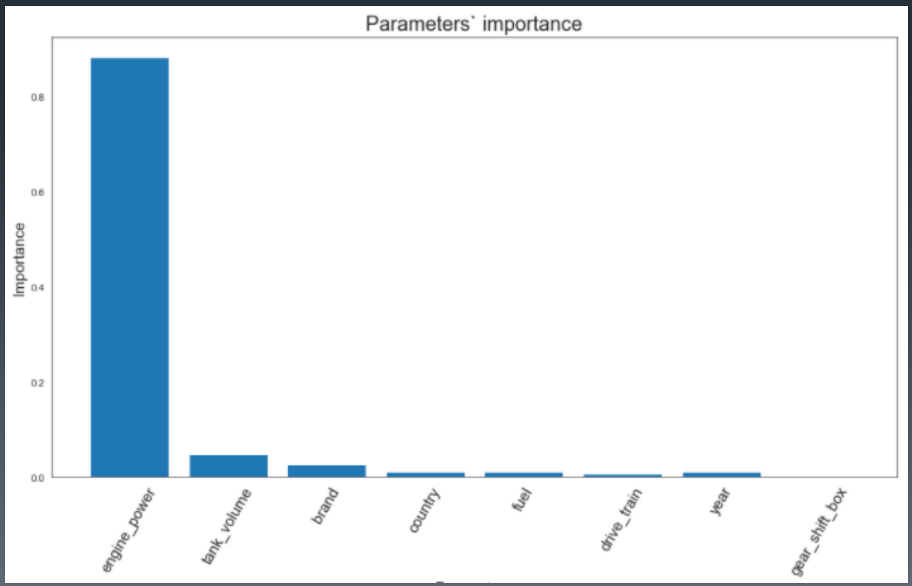
```
0.9687388015492082
```

```
model_3.score(X_test,y_test)
```

```
0.9515702566590641
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions_3))
print('MSE:', metrics.mean_squared_error(y_test, predictions_3))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_3)))
```

```
MAE: 4766.810790403959
MSE: 176441349.5640577
RMSE: 13283.122733907781
```



2 option (get_dummies)

```
model_3.score(X_train, y_train)
```

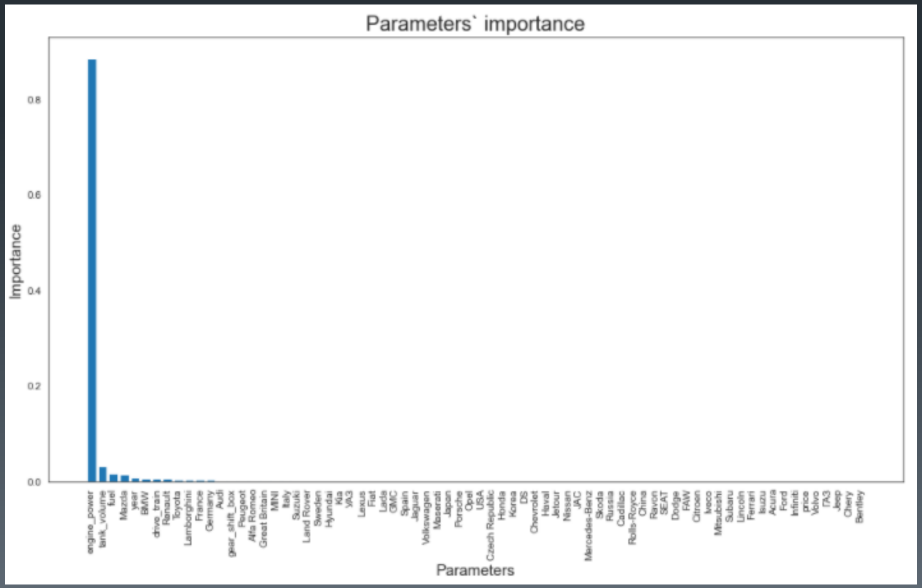
```
0.9663669344627227
```

```
model_3.score(X_test,y_test)
```

```
0.9453378016584275
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions_3))
print('MSE:', metrics.mean_squared_error(y_test, predictions_3))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_3)))
```

```
MAE: 5007.581906675893
MSE: 224396851.42222992
RMSE: 14979.881555680937
```



```python
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LarsCV
from sklearn.linear_model import BayesianRidge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import LinearSVR
from sklearn.svm import SVR
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

- After going through the following algorithms, ***Decision Tree Regressor*** showed the highest accuracy. The other high-accurate algorithms are XGBRegressor, BaggingRegressor, and GradientBoostingRegressor.

XGB Boost

▪ 1 option (LabelEncoder)

```
model_4.score(X_train, y_train)
```

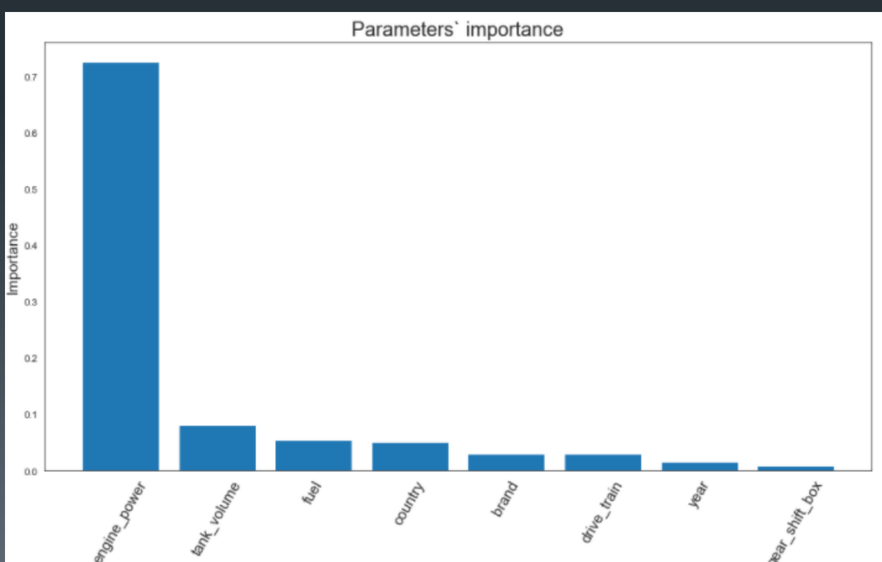
```
0.9729684015610486
```

```
model_4.score(X_test,y_test)
```

```
0.9508025436043144
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions_4))
print('MSE:', metrics.mean_squared_error(y_test, predictions_4))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_4)))
```

```
MAE: 4715.8523917827115
MSE: 179238315.19124255
RMSE: 13387.991454704568
```



▪ 2 option (get_dummies)

```
model_4.score(X_train, y_train)
```

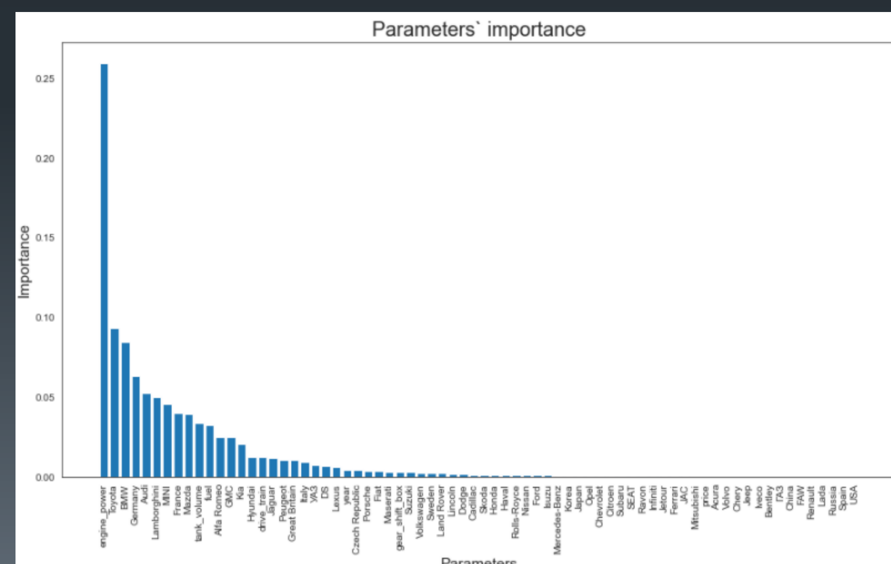
```
0.9723475206340607
```

```
model_4.score(X_test,y_test)
```

```
0.9303614875144459
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions_4))
print('MSE:', metrics.mean_squared_error(y_test, predictions_4))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_4)))
```

```
MAE: 5109.1274691319995
MSE: 285876957.26831686
RMSE: 16907.89629931284
```



Bagging Regressor

- 1 option (LabelEncoder)

```
model_5.score(X_train, y_train)
0.9681546643002782

model_5.score(X_test,y_test)
0.950694507380788

print('MAE:', metrics.mean_absolute_error(y_test, predictions_5))
print('MSE:', metrics.mean_squared_error(y_test, predictions_5))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_5)))
MAE: 4803.856765499738
MSE: 179631917.46467614
RMSE: 13402.68321884376
```

- 2 option (get_dummies)

```
model_5.score(X_train, y_train)
0.9663711388661019

model_5.score(X_test,y_test)
0.9463068908788026

print('MAE:', metrics.mean_absolute_error(y_test, predictions_5))
print('MSE:', metrics.mean_squared_error(y_test, predictions_5))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_5)))
MAE: 4987.589948915292
MSE: 220418588.99596384
RMSE: 14846.500900749774
```

Gradient Boosting Regressor

- 1 option (LabelEncoder)

```
model_6.score(X_train, y_train)
```

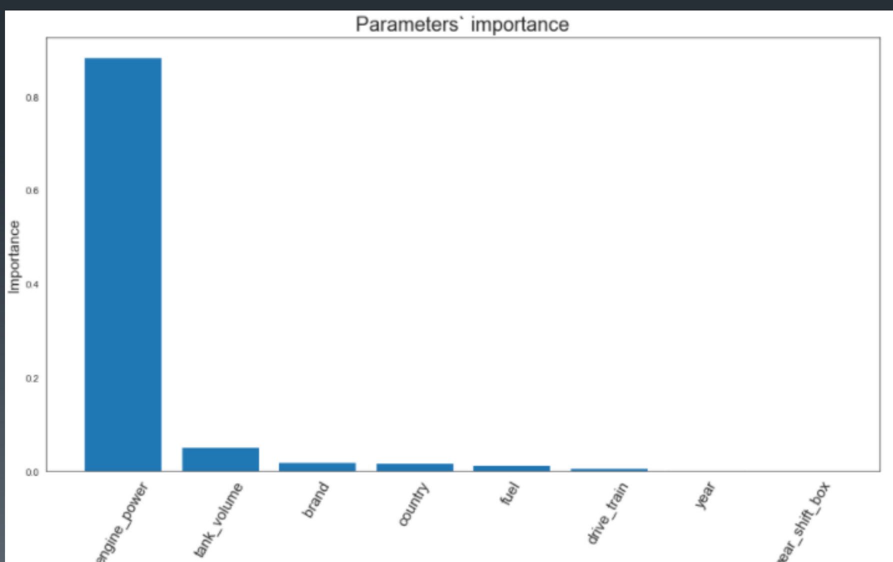
0.9569157670737161

```
model_6.score(X_test,y_test)
```

0.9481500413321353

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions_6))
print('MSE:', metrics.mean_squared_error(y_test, predictions_6))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_6)))
```

```
MAE: 5996.076365627338
MSE: 188902026.95882934
RMSE: 13744.163377915345
```



- 2 option (get_dummies)

```
model_6.score(X_train, y_train)
```

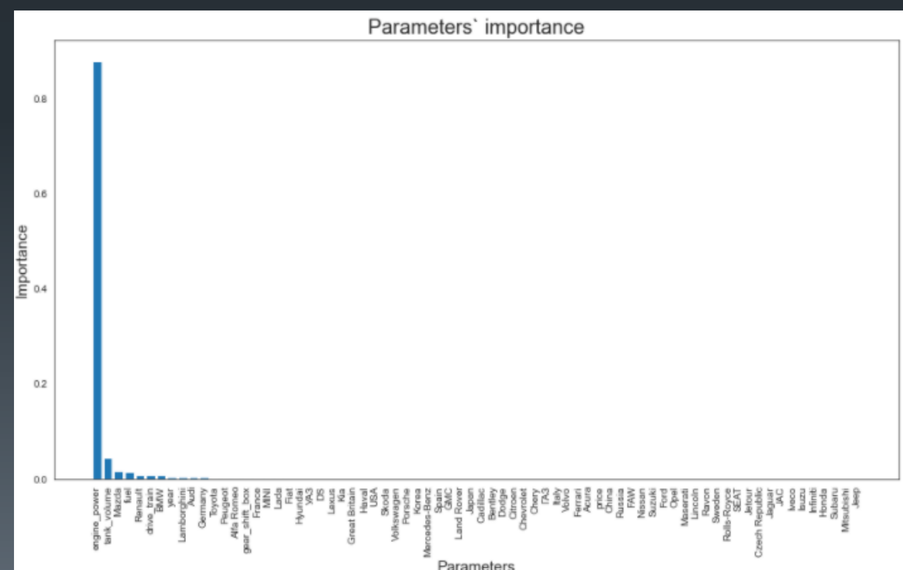
0.9543633800768756

```
model_6.score(X_test,y_test)
```

0.9404646507724556

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions_6))
print('MSE:', metrics.mean_squared_error(y_test, predictions_6))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_6)))
```

```
MAE: 6425.255463728908
MSE: 244401896.01418686
RMSE: 15633.358436823064
```



Ensemble Voting Regressor

- 1 option (LabelEncoder)

```
ensemble.score(X_train, y_train)
```

```
0.9625843066709693
```

```
ensemble.score(X_test,y_test)
```

```
0.9721549877375725
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions_ensemble))
print('MSE:', metrics.mean_squared_error(y_test, predictions_ensemble))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_ensemble)))
```

```
MAE: 4570.963445877583
MSE: 101446161.04247761
RMSE: 10072.048502786194
```

- 2 option (get_dummies)

```
ensemble.score(X_train, y_train)
```

```
0.9638693160068686
```

```
ensemble.score(X_test,y_test)
```

```
0.9738959330481685
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions_ensemble))
print('MSE:', metrics.mean_squared_error(y_test, predictions_ensemble))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions_ensemble)))
```

```
MAE: 4379.340936354549
MSE: 107161267.03691565
RMSE: 10351.872634307072
```

Algorithms comparison

- 1 option (LabelEncoder)

```
resultt
```

```
{'LinearRegression': 'Test R^2 = 0.8355',
 'DecisionTreeRegressor': 'Test R^2 = 0.9730',
 'RandomForestRegressor': 'Test R^2 = 0.9516 ',
 'XGBRegressor': 'Test R^2 = 0.9508',
 'BaggingRegressor': 'Test R^2 = 0.9507',
 'GradientBoostingRegressor': 'Test R^2 = 0.9482',
 'Ensemble VotingRegressor': 'Test R^2 = 0.9722'}
```

- Decision Tree has the highest accuracy – 0.9730. Linear Regression has the lowest accuracy – only 0.8355.

- 2 option (get_dummies)

```
result
```

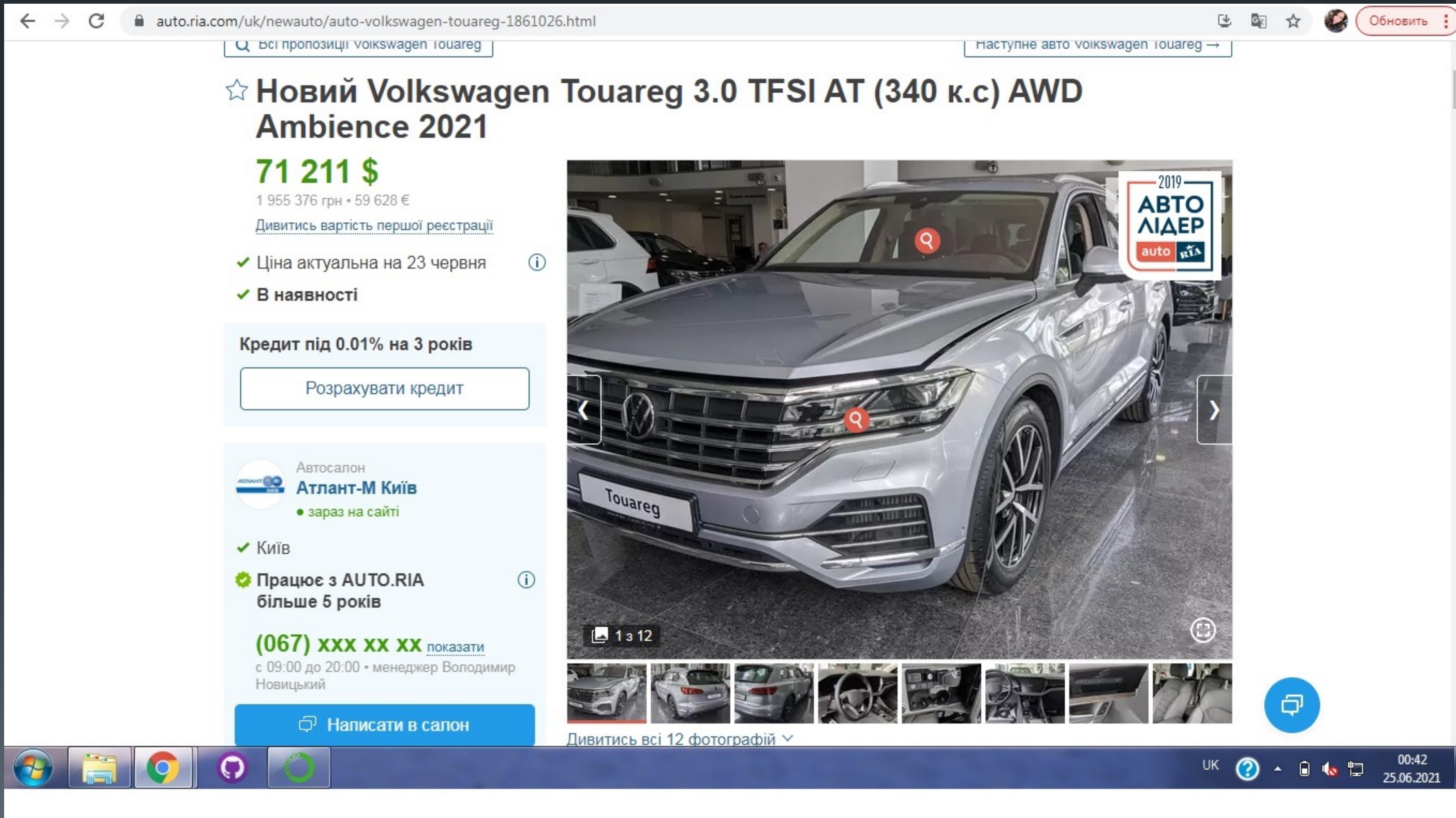
```
{'LinearRegression': 'Test R^2 = 0.9031',
 'DecisionTreeRegressor': 'Test R^2 = 0.9725',
 'RandomForestRegressor': 'Test R^2 = 0.9453 ',
 'XGBRegressor': 'Test R^2 = 0.9304',
 'BaggingRegressor': 'Test R^2 = 0.9463',
 'GradientBoostingRegressor': 'Test R^2 = 0.9405',
 'Ensemble VotingRegressor': 'Test R^2 = 0.9739'}
```

- Ensemble Voting Regressor has the highest accuracy – 0,9739. Linear Regression has the lowest accuracy – 0,9031.

- All in all, the first option with LabelEncoder is more appropriate and shows higher accuracy than the second one.

Price prediction

- On the website Auto.ria.com, I chose a car priced at $**<u>71,211:</u>**



Comparison of the predicted price by different algorithms

- 1 option (LabelEncoder)

```
_predictions

{'LinearRegression': 137805,
 'DecisionTreeRegressor': 72780,
 'RandomForestRegressor': 72800,
 'XGBRegressor': 72800,
 'BaggingRegressor': 73013,
 'GradientBoostingRegressor': 78878,
 'Ensemble VotingRegressor': 83885}
```

- The closest forecast to the real price is by DecisionTreeRegressor algorithm – $72,780

- 2 option (get_dummies)

```
_predictions

{'LinearRegression': 109689,
 'DecisionTreeRegressor': 73737,
 'RandomForestRegressor': 74097,
 'XGBRegressor': 73762,
 'BaggingRegressor': 73722,
 'GradientBoostingRegressor': 106776,
 'Ensemble VotingRegressor': 82623}
```

- The closest forecast to the real price is by BaggingRegressor – $73,722

- In general, the price prediction is better in the first option – the forecast is closer to the real price on the website.

Cross Validation
2 option (get_dummies)

- Evaluation of the effectiveness of each algorithm is performed using cross-validation. The output message contains the following information: the name of the algorithm as an abbreviation, the average score of 10-fold cross-validation on the training data (metric ‘r2’), the standard deviation in parentheses, and the coefficient of determination r2 on the test data.

```
LinearRegression: train = 0.8828 (0.0248) / test = 0.8087
DecisionTreeRegressor: train = 0.9313 (0.0207) / test = 0.7899
RandomForestRegressor: train = 0.9447 (0.0175) / test = 0.8134
XGBRegressor: train = 0.9447 (0.0169) / test = 0.7874
BaggingRegressor: train = 0.9444 (0.0178) / test = 0.8120
GradientBoostingRegressor: train = 0.9383 (0.0194) / test = 0.8025
VotingRegressor: train = 0.9449 (0.0184) / test = 0.8178
```

Boxplot of each algorithm’s span