Project: Used_Cars - Prices - Predictions

1. <u>Preliminary considerations:</u>
    - When selecting the data set, I chose the automotive sector in the marketing area. The aim here is to analyze, visualize and predict data.
    - The used car market has developed in different directions in recent years. The reasons for this are diverse and cannot be reduced solely to so-called epidemic times (Corona). It will be all the more important in the future to respond to this in a timely manner and with the right strategies.
    - This project is about analyzing and visualizing an existing data set. The cleaned, prepared data is then trained with algorithms in the form of a machine learning model so that precise price predictions can then be made.
    - Since the data set already contains a target label (“price”), you can proceed with a supervised learning algorithm. The values here are numerical, so I will need regressors.
2. <u>Aim:</u>
    - ✔ From a selection of several algorithms from this machine learning area, after comparing calculations, I choose an ML model with which I can make the best possible precise prediction of how expensive a product (car) will be if it has certain parameters (characteristic values) should fulfill.
    - ✔ A precise price level should be able to be determined with changed key data.

Table of contents

Project: Used_Cars - Prices - Predictions 1
.Dataset analysis - Used_Cars....3
1.Import a free dataset from Kaggle....3
2.Analysis of the Dataset....3
3.Prepare/clean data....8
4.Show correlations of features:....12
5.Visualizations in Tableau:....17
.The 5 best-selling brands should be used for processing in machine learning....22
.Create a Machine Learning Model....23
1Preparation of the data set: 23
2train_test_split --> ML, Supervised Learning....25
3Evaluation and conclusion → best algorithm....28
.Overall conclusion:....33
.Outlook:....34

. Dataset analysis - Used_Cars

1. Import a free dataset from Kaggle

- Data set from an online marketplace for vehicles
- Download and import the data set in .csv format

2. Analysis of the Dataset

A) Data set with its specific requirements:

- 9 different Features (columns) with 46405 Samples (rows)
- 9 Features ('mileage', 'make', 'model', 'fuel', 'gear', 'offerType', 'price', 'hp','year') are present in total, one, 'price', offers itself as a target (“label”).
- Features content:
  - ✗ mileage
  - ✗ make (brand)
  - ✗ model
  - ✗ fuel
  - ✗ gear (vehicle transmission)
  - ✗ offerType
  - ✗ price
  - ✗ hp (engine power)
  - ✗ year (Construction year)

B) Examine the data set in detail:

- Import necessary libraries to process the data set (Numpy, Pandas, MatPlotLib, Seaborn...)

In [1]:
```python

# import libraries:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_theme()
# execute if warnings should be ignored:
import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_columns', 35)
pd.set_option('display.max_rows', 2500)

```

- Import data into Jupyter Notebook and create DataFrame (Pandas) with pd.read_csv("...")
- Data set excerpt: → an overview of where there are numerical values and where there are categorical values

In [4]:
```python
df01.head()
```

Out[4]:

	mileage	make	model	fuel	gear	offerType	price	hp	year
0	235000	BMW	316	Diesel	Manual	Used	6800	116.0	2011
1	92800	Volkswagen	Golf	Gasoline	Manual	Used	6877	122.0	2011
2	149300	SEAT	Exeo	Gasoline	Manual	Used	6900	160.0	2011
3	96200	Renault	Megane	Gasoline	Manual	Used	6950	110.0	2011
4	156000	Peugeot	308	Gasoline	Manual	Used	6950	156.0	2011

- in which ranges do the numerical values range, where are there outliers:

In [5]:

```
# 1. --> sold cars, carAge between 0 and 10 years
# outliers: mileage,hp,price

df01.describe()

```

Out[5]:

	mileage	price	hp	year
count	4.640500e+04	4.640500e+04	46376.000000	46405.000000
mean	7.117786e+04	1.657234e+04	132.990987	2016.012951
std	6.262531e+04	1.930470e+04	75.449284	3.155214
min	0.000000e+00	1.100000e+03	1.000000	2011.000000
25%	1.980000e+04	7.490000e+03	86.000000	2013.000000
50%	6.000000e+04	1.099900e+04	116.000000	2016.000000
75%	1.050000e+05	1.949000e+04	150.000000	2019.000000
max	1.111111e+06	1.199900e+06	850.000000	2021.000000

- Where are there NaN-values, unique values, etc::

```
In [7]:

df01.isnull().sum()

Out[7]: mileage          0
        make             0
        model          143
        fuel             0
        gear           182
        offerType        0
        price            0
        hp              29
        year             0
        dtype: int64
```

```
In [8]:

df01.nunique()

Out[8]: mileage      20117
        make            77
        model          841
        fuel            11
        gear             3
        offerType        5
        price         6668
        hp             328
        year            11
        dtype: int64
```

```

# --> find/delete the outliers of the feature "mileage" (delete 2 unrealistic values!):

df01[df01["mileage"] >= 999999]

# delete sample at specific index:
df01.drop(df01[df01["mileage"] > 999990].index, inplace=True)

```

```

# --> no outliers at feature "hp", but NaN-values
# (NaN-values would have too many different values,
# so filling them with mean values makes no sense!):

# delete sample at specific index:
df01.drop(df01[df01["hp"].isnull()].index, inplace=True)

```

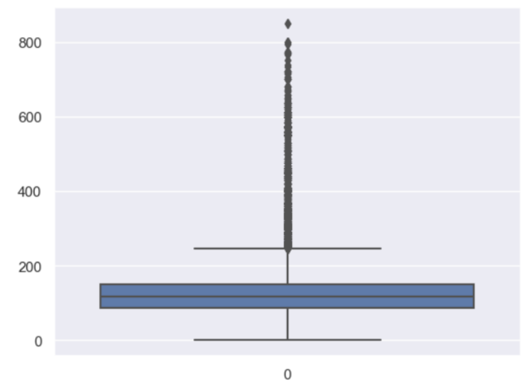
NaN-values in Feature „model“:

```

# NaN-values at feature "model" are filled with "different"
# so that the data remains usable:
df01["model"] = df01["model"].fillna("different")

```

Distribution of hp values:

```

sns.boxplot(df01["hp"])

```

<Axes: >



- NaN-values for “gear” are filled with the majority value:

```python

# NaN-values at feature "gear" are filled with "Manual"
# because usually they have a gear shift:
df01.gear.fillna("Manual",inplace=True)

```

- In the “make” feature there are poorly fitting values, which I remove:

```python

# "Trailer-Anhänger" deleted:
df01.drop(df01[df01["make"] == "Trailer-Anhänger"].index,inplace=True)

```

- Now there are 46371 Samples and 9 Columns
- Before I convert the remaining categorical values to numeric, I export the data set so that I can optimally use it in Tableau for visualization

```python

# export file for Tableau,
# before converting the final categorical values to numeric:
df01.to_csv("autoscout24_vis.csv",index=False)

```

- The data set currently looks like this: numerical/categorical values

```
df01.sample(10)
```

	mileage	make	model	fuel	gear	offerType	price	hp	year
5289	112600	Ford	Mondeo	Diesel	Automatic	Used	9500	163.0	2011
23432	3900	Mitsubishi	Outlander	Electric/Gasoline	Automatic	Demonstration	33300	135.0	2020
22403	15	Nissan	Qashqai	Gasoline	Automatic	Pre-registered	24990	158.0	2021
2688	10	Skoda	Fabia	Gasoline	Automatic	Pre-registered	16688	95.0	2021
8649	155000	Volkswagen	Tiguan	Diesel	Automatic	Used	11490	140.0	2011
23227	500	Nissan	Qashqai	Gasoline	Automatic	Pre-registered	25490	159.0	2021
27842	75421	Volkswagen	Tiguan	Gasoline	Automatic	Used	28477	179.0	2018
8679	130359	MINI	Cooper S Countryman	Gasoline	Manual	Used	10499	184.0	2012
12922	109900	Volkswagen	Golf	Gasoline	Manual	Used	7790	86.0	2013
9523	102000	Hyundai	i30	Gasoline	Manual	Used	7400	99.0	2013

3. Prepare/clean data

A) In order to encode the categorical values (make them numeric), I have to transform the one-dimensional arrays of the features "make", "model", "fuel", "gear", "offerType" and "year" into lists, which must have the same length as the rest of the data set:

```
liste_make = []

arr01 = np.array(df01["make"])

for i in arr01:
    liste_make.append(i)

print(len(liste_make))

```

```
46371
```

B) I use LabelEncoder from sklearn: → import, initialize and fit/transform in one step

```
# converting features: "make", "model","fuel","gear","offerType",
#                      "year"

from sklearn.preprocessing import LabelEncoder

# individual string-features to lists: liste_make, liste_model,
# liste_fuel, liste_gear, liste_offerType, liste_year

# initialize the encoder:
le = LabelEncoder()

# encode the features to lists-form:
# fit --> create unique values:
# transform --> convert to numeric values:
encoded_make = le.fit_transform(liste_make)
encoded_model = le.fit_transform(liste_model)
encoded_fuel = le.fit_transform(liste_fuel)
encoded_gear = le.fit_transform(liste_gear)
encoded_offerType = le.fit_transform(liste_offerType)
encoded_year = le.fit_transform(liste_year)

```

C) Create new features and fill them with the new values:

```

# filling the new features with numerical values
# from the respective lists:
df01["encoded_make"] = encoded_make
df01["encoded_model"] = encoded_model
df01["encoded_fuel"] = encoded_fuel
df01["encoded_gear"] = encoded_gear
df01["encoded_offerType"] = encoded_offerType
df01["encoded_year"] = encoded_year

```

D) Without having to eliminate the categorical values, I can check whether the values are correct: → with „..describe()“ (descriptive data analysis)

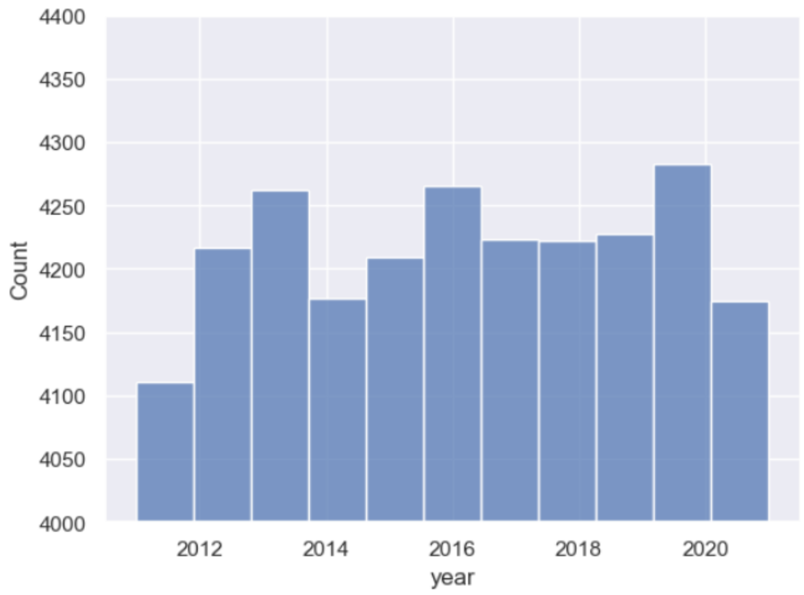
```
df01.describe()
```

	mileage	price	hp	year	encoded_make	encoded_model	encoded_fuel	encoded_gear	encoded_offerType	encoded_year
count	46371.000000	4.637100e+04	46371.000000	46371.000000	46371.000000	46371.000000	46371.000000	46371.000000	46371.000000	46371.000000
mean	71150.808027	1.657495e+04	133.003494	2016.012529	47.120399	421.950918	5.228677	0.660973	3.663475	5.012529
std	62268.606055	1.930898e+04	75.443174	3.154958	21.620282	256.843797	2.363117	0.475928	0.989698	3.154958
min	0.000000	1.100000e+03	1.000000	2011.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	19833.500000	7.490000e+03	86.000000	2013.000000	29.000000	184.000000	2.000000	0.000000	4.000000	2.000000
50%	60000.000000	1.099900e+04	116.000000	2016.000000	54.000000	402.000000	7.000000	1.000000	4.000000	5.000000
75%	105000.000000	1.949000e+04	150.000000	2019.000000	64.000000	647.000000	7.000000	1.000000	4.000000	8.000000
max	699000.000000	1.199900e+06	850.000000	2021.000000	75.000000	840.000000	10.000000	2.000000	4.000000	10.000000

E) A few visualizations in advance about sales depending on other features:

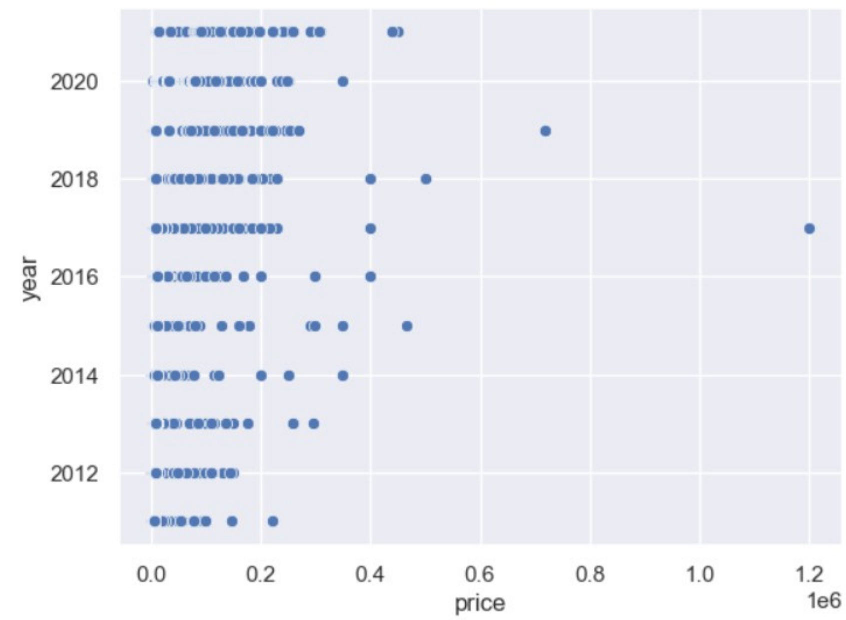
```python
# --> distribution of sales/year of manufacture (carAge):

plt.ylim(4000,4400)
sns.histplot(data=df01,x="year",bins=11)
```

```
<Axes: xlabel='year', ylabel='Count'>
```




```python
sns.scatterplot(data=df01,y="year",x="price")
```

```
<Axes: xlabel='price', ylabel='year'>
```



4. Show correlations of features:

A) To show the correlations, I can already use the current data set:

```

df01.corr(numeric_only=True)

```

	mileage	price	hp	year	encoded_make	encoded_model	encoded_fuel	encoded_gear	encoded_offerType	encoded_year
mileage	1.000000	-0.304146	-0.015018	-0.679989	-0.018031	-0.063810	-0.385340	0.088982	0.354142	-0.679989
price	-0.304146	1.000000	0.747624	0.405567	-0.121860	0.033451	-0.079765	-0.432103	-0.264439	0.405567
hp	-0.015018	0.747624	1.000000	0.167302	-0.230508	-0.019706	-0.193633	-0.527931	-0.107402	0.167302
year	-0.679989	0.405567	0.167302	1.000000	-0.016238	0.037764	0.067137	-0.235300	-0.465642	1.000000
encoded_make	-0.018031	-0.121860	-0.230508	-0.016238	1.000000	0.299285	0.062220	0.071154	0.007274	-0.016238
encoded_model	-0.063810	0.033451	-0.019706	0.037764	0.299285	1.000000	-0.002275	-0.056982	-0.028380	0.037764
encoded_fuel	-0.385340	-0.079765	-0.193633	0.067137	0.062220	-0.002275	1.000000	0.248413	-0.055395	0.067137
encoded_gear	0.088982	-0.432103	-0.527931	-0.235300	0.071154	-0.056982	0.248413	1.000000	0.124602	-0.235300
encoded_offerType	0.354142	-0.264439	-0.107402	-0.465642	0.007274	-0.028380	-0.055395	0.124602	1.000000	-0.465642
encoded_year	-0.679989	0.405567	0.167302	1.000000	-0.016238	0.037764	0.067137	-0.235300	-0.465642	1.000000

B) Correlations between the initially existing numerical values: → maximum values are 0.7 and can be found at year/mileage and hp/price

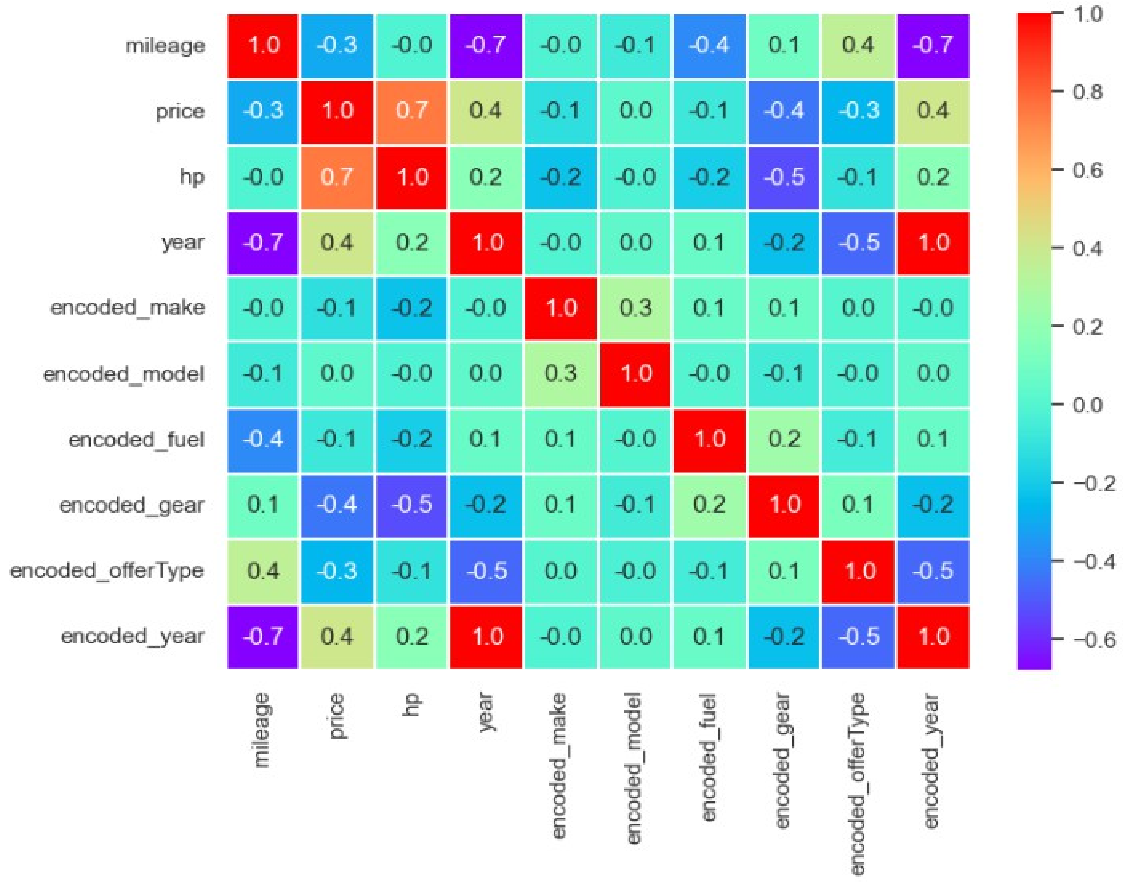
```python
# correlations among the numerical values:
# --> highest correlations 0.7:   year/mileage and hp/price

sns.heatmap(df01.drop(["make","model","fuel","gear","offerType",
                       "encoded_make","encoded_model","encoded_fuel","encoded_gear","encoded_offerType"],
                      axis=1).corr(),cmap="viridis",linewidths=0.1,fmt=".2f",annot=True)

```

```
<Axes: >
```

C) Heatmap with all features

```python

# heaatmap with all correlations:
plt.figure(figsize=(8,6))
sns.heatmap(df01.corr(numeric_only=True),cmap="rainbow",linewidths=0.1,fmt=".1f",annot=True)

```

```
<Axes: >
```



D) Most important correlations in detail:

--> <u>positive:</u>

CORRELATION 0.8 bzw 0.7:
price - hp

CORRELATION 0.4:
price - year

CORRELATION 0.4:
offerType - mileage

CORRELATION 0.3:
mark - model

--> <u>negative:</u>

CORRELATION -0.7:
mileage - year

CORRELATION -0.5:
encoded_gear - hp
offerType - year

CORRELATION -0.4:
encoded_fuel - mileage
encoded_gear - price

CORRELATION -0.3:
offerType - price
mileage - price

E) Complete data set now has 46371 samples and 15 features:

```
df01
```

	mileage	make	model	fuel	gear	offerType	price	hp	year	encoded_make	encoded_model	encoded_fuel	encoded_gear	en
0	235000	BMW	316	Diesel	Manual	Used	6800	116.0	2011	8	33	2	1	
1	92800	Volkswagen	Golf	Gasoline	Manual	Used	6877	122.0	2011	72	396	7	1	
2	149300	SEAT	Exeo	Gasoline	Manual	Used	6900	160.0	2011	63	323	7	1	
3	96200	Renault	Megane	Gasoline	Manual	Used	6950	110.0	2011	61	508	7	1	
4	156000	Peugeot	308	Gasoline	Manual	Used	6950	156.0	2011	56	32	7	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
46400	99	Fiat	500	Electric/Gasoline	Manual	Pre-registered	12990	71.0	2021	28	54	5	1	
46401	99	Fiat	500	Electric/Gasoline	Manual	Pre-registered	12990	71.0	2021	28	54	5	1	
46402	99	Fiat	500	Electric/Gasoline	Manual	Pre-registered	12990	71.0	2021	28	54	5	1	
46403	99	Fiat	500	Electric/Gasoline	Manual	Pre-registered	12990	71.0	2021	28	54	5	1	
46404	99	Fiat	500	Electric/Gasoline	Manual	Pre-registered	12990	71.0	2021	28	54	5	1	

46371 rows × 15 columns

```

# export the numeric/cleaned data set:
df01.to_csv("autoscout24_numerisch.csv",index=False)

```

F) Checking the numerical values with .describe:

```
1 df01.describe()
```

	mileage	price	hp	year	encoded_make	encoded_model	encoded_fuel	encoded_gear	encoded_offerType	encoded_year
count	46371.000000	4.637100e+04	46371.000000	46371.000000	46371.000000	46371.000000	46371.000000	46371.000000	46371.000000	46371.000000
mean	71150.808027	1.657495e+04	133.003494	2016.012529	47.120399	421.950918	5.228677	0.660973	3.663475	5.012529
std	62268.606055	1.930898e+04	75.443174	3.154958	21.620282	256.843797	2.363117	0.475928	0.989698	3.154958
min	0.000000	1.100000e+03	1.000000	2011.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	19833.500000	7.490000e+03	86.000000	2013.000000	29.000000	184.000000	2.000000	0.000000	4.000000	2.000000
50%	60000.000000	1.099900e+04	116.000000	2016.000000	54.000000	402.000000	7.000000	1.000000	4.000000	5.000000
75%	105000.000000	1.949000e+04	150.000000	2019.000000	64.000000	647.000000	7.000000	1.000000	4.000000	8.000000
max	699000.000000	1.199900e+06	850.000000	2021.000000	75.000000	840.000000	10.000000	2.000000	4.000000	10.000000

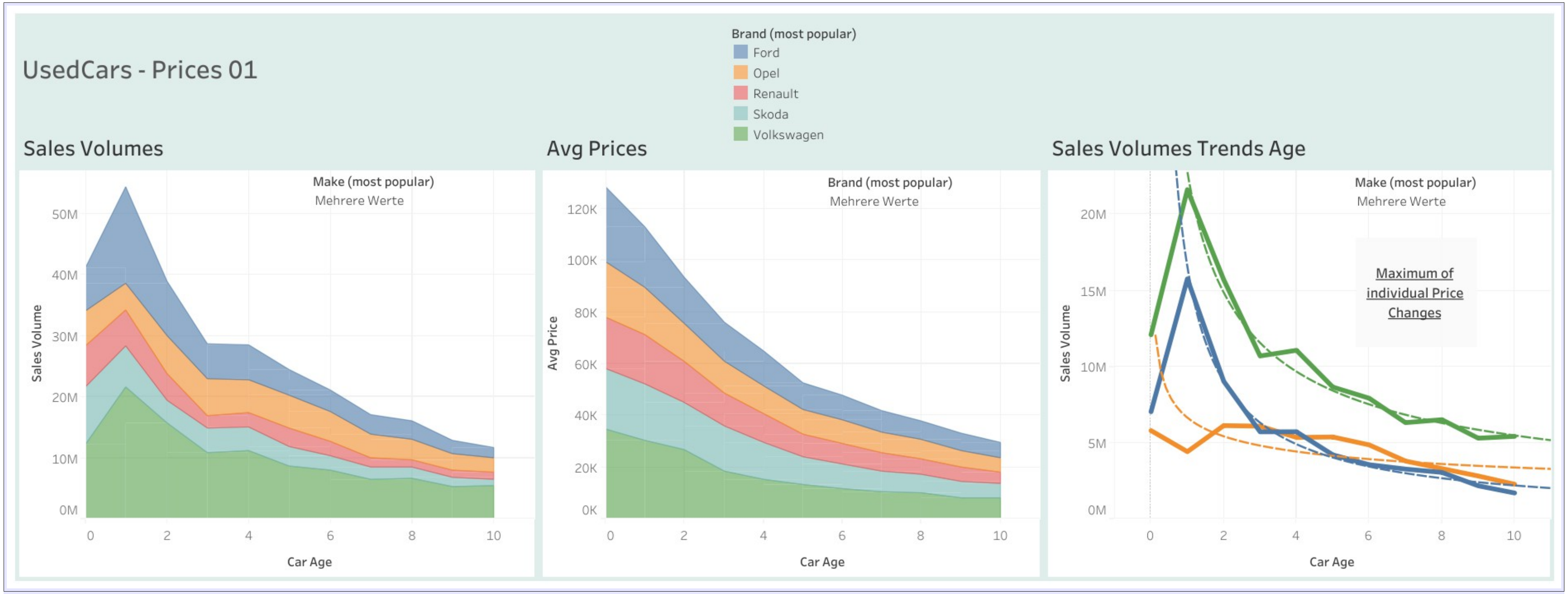
5. Visualizations in Tableau:

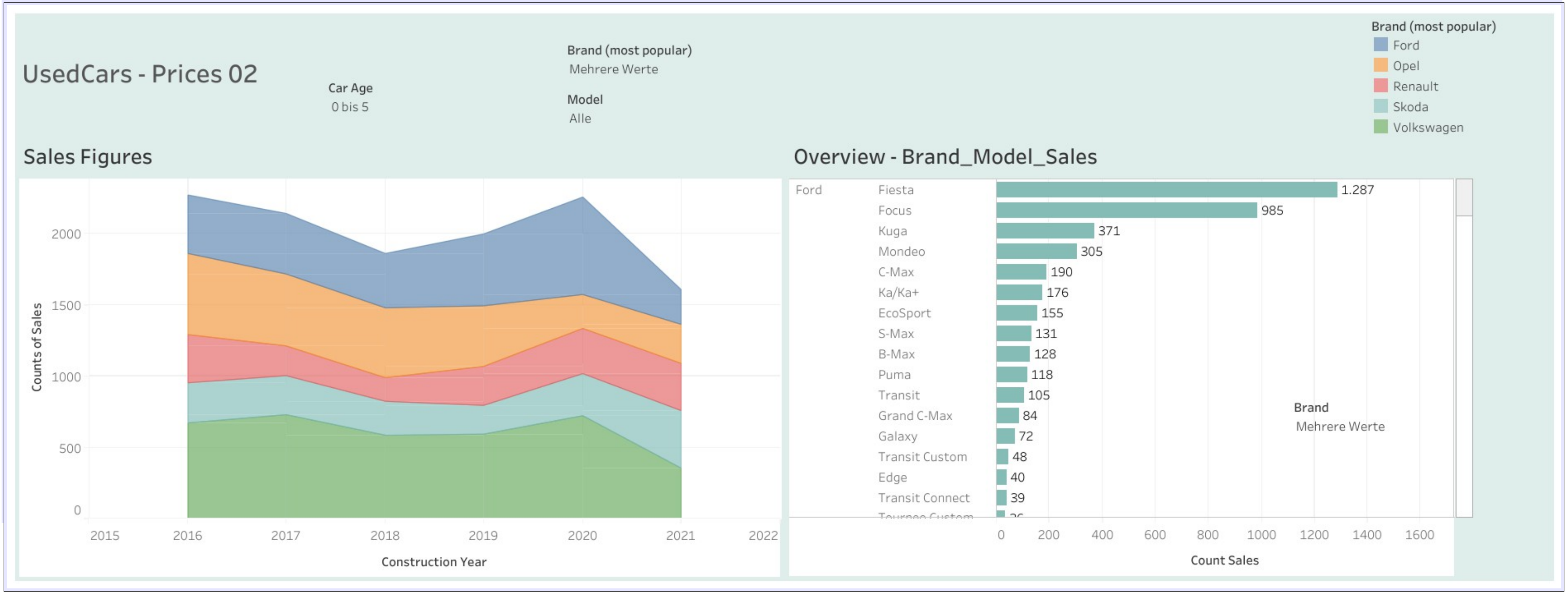
A) Using the already exported data set: → A) Creating dashboards

B) Presentation of the analysis of the data set available at the following link::

[UsedCars - Prices]

C) The link is necessary because otherwise you cannot use the filters built into the dashboards for a specific view of the diagrams.


UsedCars - Prices 01
Brand (most popular)
Ford
Opel
Renault
Skoda
Volkswagen
Sales Volumes
Make (most popular)
Mehrere Werte
Sales Volume
50M
40M
30M
20M
10M
0M
0
2
4
6
8
10
Car Age
Avg Prices
Brand (most popular)
Mehrere Werte
Avg Price
120K
100K
80K
60K
40K
20K
0K
Car Age
Sales Volumes Trends Age
Make (most popular)
Mehrere Werte
Maximum of individual Price Changes
Sales Volume
20M
15M
10M
5M
0M
Car Age

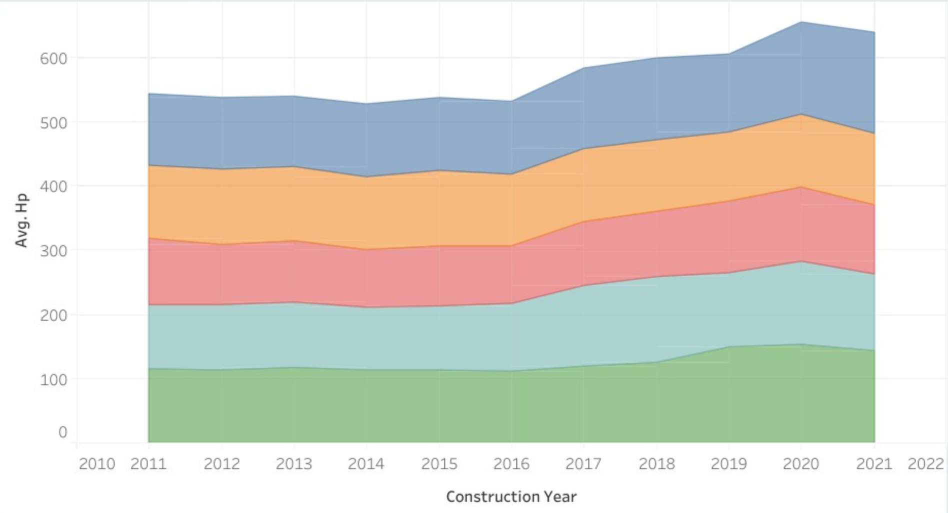

UsedCars - Prices 02
Car Age
0 bis 5
Brand (most popular)
Mehrere Werte
Model
Alle
Brand (most popular)
Ford
Opel
Renault
Skoda
Volkswagen
Sales Figures
Counts of Sales
2000
1500
1000
500
0
2015
2016
2017
2018
2019
2020
2021
2022
Construction Year
Overview - Brand_Model_Sales
Ford
Fiesta 1.287
Focus 985
Kuga 371
Mondeo 305
C-Max 190
Ka/Ka+ 176
EcoSport 155
S-Max 131
B-Max 128
Puma 118
Transit 105
Grand C-Max 84
Galaxy 72
Transit Custom 48
Edge 40
Transit Connect 39
Brand
Mehrere Werte
0
200
400
600
800
1000
1200
1400
1600
Count Sales

UsedCars - Prices 03

Brand (most popular)
Ford
Opel
Renault
Skoda
Volkswagen

Engine - HP (Avg)

Brand (most popular)
Mehrere Werte

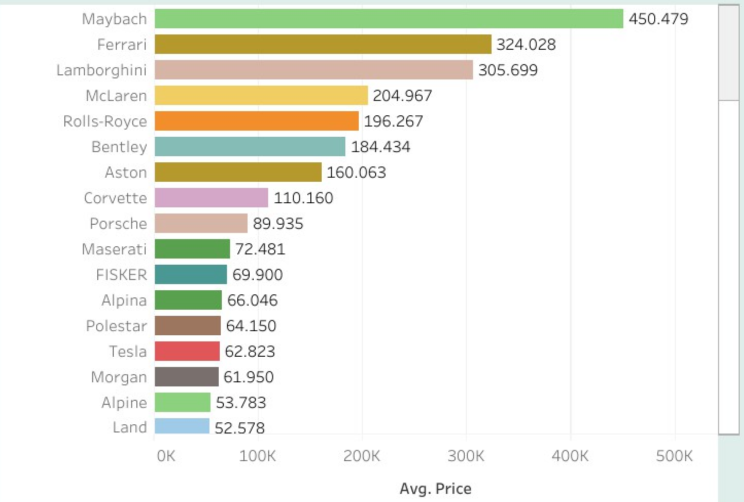


Engine - HP - Price

UsedCars - Prices 04

Engine - Brand - Price

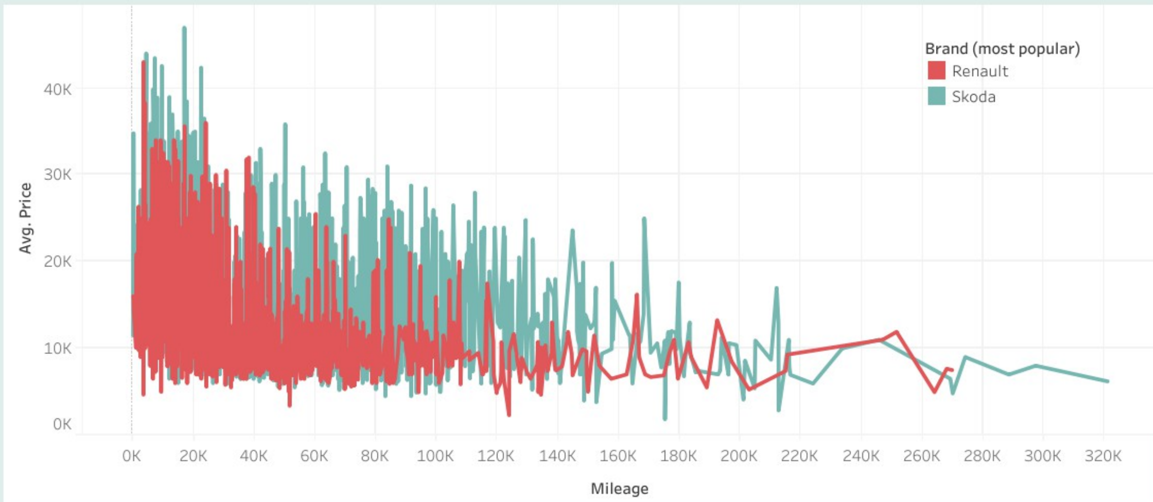
Brand
Alle



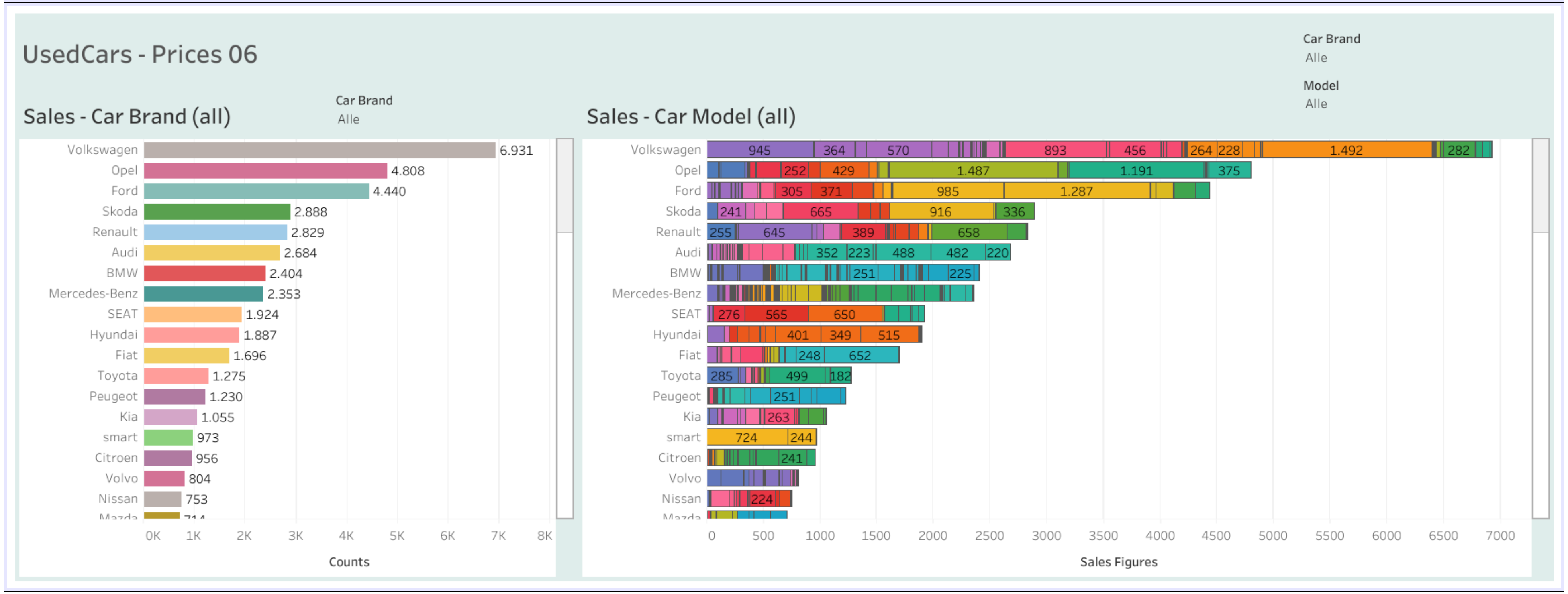
AvgPrice - Mileage

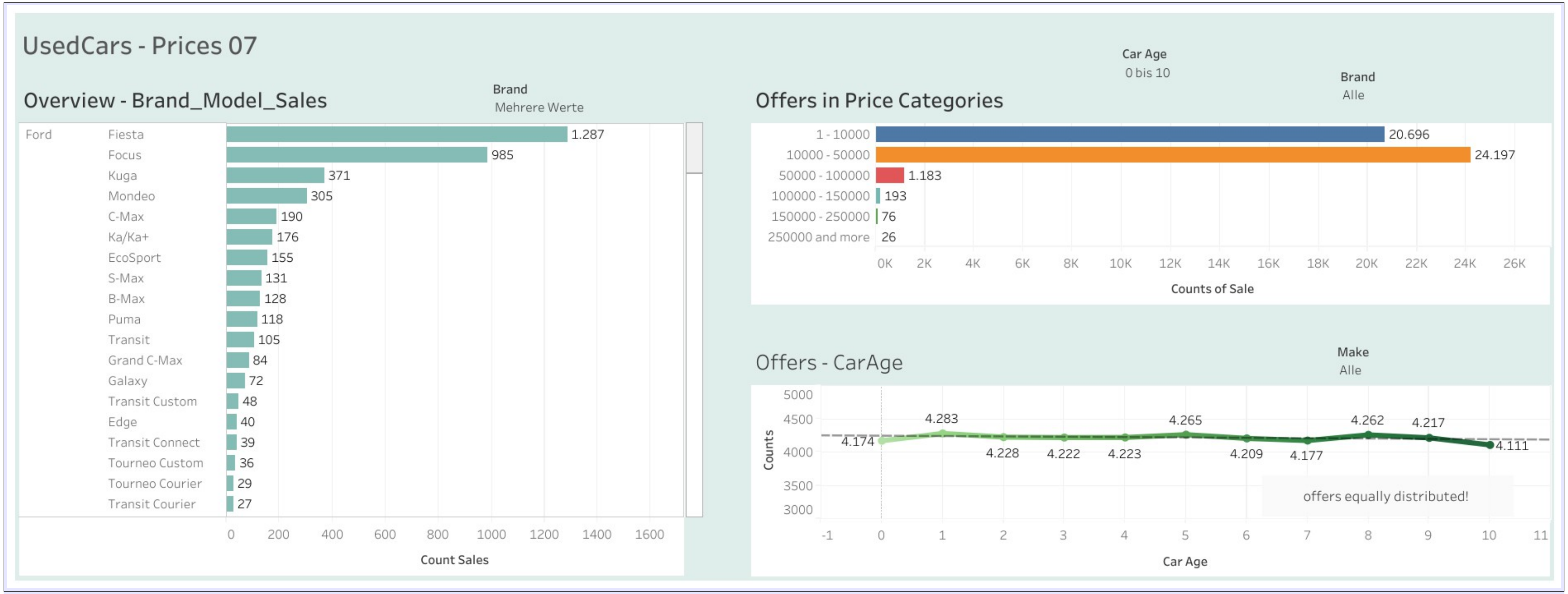
Car Age
1 bis 6

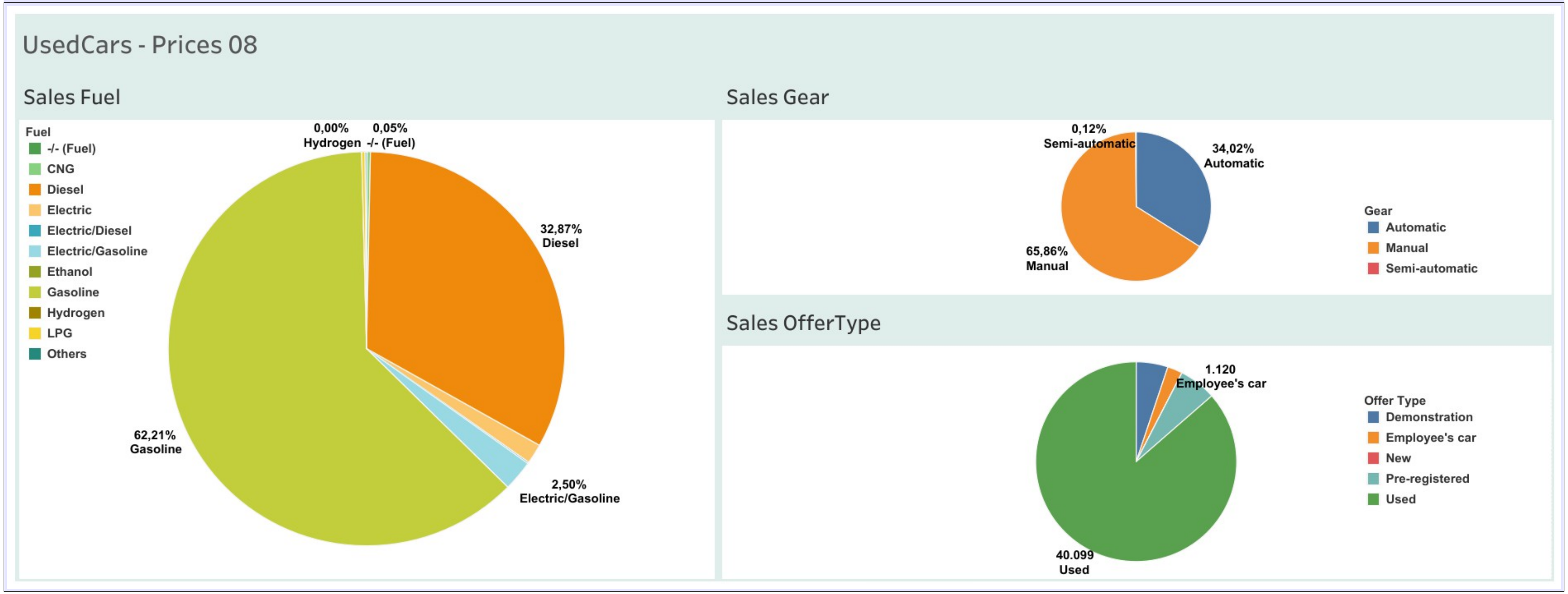
Brand (most popular)
Mehrere Werte




UsedCars - Prices 05
Brand
Mehrere Werte
CarAge - Brand - Price (Avg)
Car Age
Price Level of individual Brands
Avg. Price
200K
150K
100K
50K
0K
Audi 75.262
BMW 65.320
Ford 28.846
Skoda 23.452
Audi 46.527
BMW 45.780
Ford 23.185
Skoda 22.164
Audi 42.106
BMW 38.319
Audi 33.529
BMW 31.837
Audi 30.085
BMW 29.774
Audi 24.078
0
1
2
3
4
5
6
7
8
9
10


UsedCars - Prices 06
Car Brand
Alle
Model
Alle
Sales - Car Brand (all)
Car Brand
Alle
Sales - Car Model (all)
Volkswagen 6.931
Opel 4.808
Ford 4.440
Skoda 2.888
Renault 2.829
Audi 2.684
BMW 2.404
Mercedes-Benz 2.353
SEAT 1.924
Hyundai 1.887
Fiat 1.696
Toyota 1.275
Peugeot 1.230
Kia 1.055
smart 973
Citroen 956
Volvo 804
Nissan 753
0K 1K 2K 3K 4K 5K 6K 7K 8K
Counts
Volkswagen 945 364 570 893 456 264 228 1.492 282
Opel 252 429 1.487 1.191 375
Ford 305 371 985 1.287
Skoda 241 665 916 336
Renault 255 645 389 658
Audi 352 223 488 482 220
BMW 251 225
Mercedes-Benz
SEAT 276 565 650
Hyundai 401 349 515
Fiat 248 652
Toyota 285 499 182
Peugeot 251
Kia 263
smart 724 244
Citroen 241
Volvo
Nissan 224
0 500 1000 1500 2000 2500 3000 3500 4000 4500 5000 5500 6000 6500 7000
Sales Figures


UsedCars - Prices 07
Overview - Brand_Model_Sales
Brand
Mehrere Werte
Ford
Fiesta 1.287
Focus 985
Kuga 371
Mondeo 305
C-Max 190
Ka/Ka+ 176
EcoSport 155
S-Max 131
B-Max 128
Puma 118
Transit 105
Grand C-Max 84
Galaxy 72
Transit Custom 48
Edge 40
Transit Connect 39
Tourneo Custom 36
Tourneo Courier 29
Transit Courier 27
0 200 400 600 800 1000 1200 1400 1600
Count Sales
Offers in Price Categories
Car Age
0 bis 10
Brand
Alle
1 - 10000 20.696
10000 - 50000 24.197
50000 - 100000 1.183
100000 - 150000 193
150000 - 250000 76
250000 and more 26
0K 2K 4K 6K 8K 10K 12K 14K 16K 18K 20K 22K 24K 26K
Counts of Sale
Offers - CarAge
Make
Alle
5000 4500 4000 3500 3000
Counts
4.174 4.283 4.228 4.222 4.223 4.265 4.209 4.177 4.262 4.217 4.111
offers equally distributed!
-1 0 1 2 3 4 5 6 7 8 9 10 11
Car Age


UsedCars - Prices 08
Sales Fuel
Fuel
-/- (Fuel)
CNG
Diesel
Electric
Electric/Diesel
Electric/Gasoline
Ethanol
Gasoline
Hydrogen
LPG
Others
0,00% Hydrogen
0,05% -/- (Fuel)
32,87% Diesel
62,21% Gasoline
2,50% Electric/Gasoline
Sales Gear
0,12% Semi-automatic
34,02% Automatic
65,86% Manual
Gear
Automatic
Manual
Semi-automatic
Sales OfferType
1.120 Employee's car
40.099 Used
Offer Type
Demonstration
Employee's car
New
Pre-registered
Used

The 5 best-selling brands should be used for processing in machine learning

- ✔ I have a look on these brands before deleting the rest and then determining the average prices
- ✔ I also now have a new data set df02 ready to be processed in machine learning

```
# data set with the 5 best-selling car brands:
df01["make"].value_counts().head(5)
```

```
make
Volkswagen     6931
Opel           4808
Ford           4440
Skoda          2888
Renault        2829
Name: count, dtype: int64
```

```
# average price by brand:
vw = df01[(df01["make"] == "Volkswagen")]
opel = df01[(df01["make"] == "Opel")]
ford = df01[(df01["make"] == "Ford")]
skoda = df01[(df01["make"] == "Skoda")]
renault = df01[(df01["make"] == "Renault")]

print(f"Price(Avg) Volkswagen: {vw['price'].mean().round(2)} €")
print(f"Price(Avg) Opel:       {opel['price'].mean().round(2)} €")
print(f"Price(Avg) Ford:       {ford['price'].mean().round(2)} €")
print(f"Price(Avg) Skoda:      {skoda['price'].mean().round(2)} €")
print(f"Price(Avg) Renault:    {renault['price'].mean().round(2)} €")
```

```
Price(Avg) Volkswagen: 16065.93 €
Price(Avg) Opel:       10443.97 €
Price(Avg) Ford:       13794.86 €
Price(Avg) Skoda:      13726.29 €
Price(Avg) Renault:    11286.84 €
```

```
# remove all other car brands from the data set:
df02 = df01[(df01["make"] == "Volkswagen") | (df01["make"] == "Opel") |
            (df01["make"] == "Ford") | (df01["make"] == "Skoda") | (df01["make"] == "Renault")]
```

. Create a Machine Learning Model

1 Preparation of the data set:

- ✗ --> Supervised Learning (numeric): Data set has the target with label “price”.
- ✗ for numeric predictions and speed: Choice of algorithm --> LinearRegression, DecisionTree
- ✗ for numeric predictions and accuracy: Choice of algorithm --> RandomForest
- ✗ the data set now has 21896 rows
- ✗ now categorical features are removed and the data set is split into X and y:

```

# with all features without the target "price":
X01 = df02.drop(df02.iloc[::,1:7],axis=1)

```

```
X02 = X01.drop("year",axis=1)
```

```
X02
```

	mileage	hp	encoded_make	encoded_model	encoded_fuel	encoded_gear	encoded_offerType	encoded_year
1	92800	122.0	72	396	7	1	4	0
3	96200	110.0	61	508	7	1	4	0
6	91894	131.0	61	678	2	1	4	0
7	127500	116.0	54	818	7	1	4	0
9	104	86.0	29	740	2	1	4	0
...	...	...	...	...	...	...	...	...
46342	5000	158.0	61	413	7	1	0	10
46343	100	150.0	72	396	2	0	3	10
46345	6000	158.0	61	449	7	1	0	10
46347	4800	150.0	72	700	7	0	0	10
46365	1500	60.0	64	333	7	1	0	10

21896 rows × 8 columns

```

# the label "price":
y01 = df02["price"]

```

```
y01
```

```
1         6877
3         6950
6         6970
7         6972
9         6990
         ...
46342    32480
46343    32490
46345    32680
46347    32880
46365    12980
Name: price, Length: 21896, dtype: int64
```

2 train_test_split --> ML, Supervised Learning

A) Supervised Learning with all features:

✗ Create train/test data → Import library: from sklearn.model_selection import train_test_split

```

# create test/training data set:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X02, y01, test_size=0.20, random_state=101)

```

✗ standardScaler by using LinearRegression,DecisionTree and RandomForest not recommended

--> so I forego the standardization!

✗ Import and initialize algorithms:

```

# import algorithms:
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

#initialize:
lin = LinearRegression()
dec = DecisionTreeRegressor()
rfc = RandomForestRegressor()

```

- train:

```python

# train:
lin01 = lin.fit(X_train,y_train)
dec01 = DecisionTreeRegressor(random_state=101).fit(X_train,y_train)
rfc01 = RandomForestRegressor(n_estimators=1000,random_state=101).fit(X_train,y_train)

```

- predict:

```python

# predictions:
pred_lin01 = lin01.predict(X_test)
pred_dec01 = dec01.predict(X_test)
pred_rfc01 = rfc01.predict(X_test)

```

✗ Verification of predictions and target with scatterplots: → RandomForest and DecisionTree work much better than LinearRegressor

```python
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15,5))
axes[0].scatter(y_test,pred_lin01,c="r",s=4)
axes[0].set_title("LinearRegression")
axes[1].scatter(y_test,pred_dec01,c="b",s=4)
axes[1].set_title('DecisionTreeRegressor')
axes[2].scatter(y_test,pred_rfc01,c="g",s=4)
axes[2].set_title('RandomForestRegressor')
fig.tight_layout()
plt.show()
```

✗ To evaluate the model I need metrics from sklearn to calculate the errors

```

from sklearn import metrics

```

3 Evaluation and conclusion → best algorithm

✔ From the values you can clearly see that the RandomForest algorithm works best:

```

# evaluate model:
#         --> best results with randomForest,
#             closely followed by decisionTree
print("LinearRegression:")
print("Mean absolute error:", metrics.mean_absolute_error(y_test, pred_lin01))
print("Mean squared error :", metrics.mean_squared_error(y_test, pred_lin01))
print("Root squared error :", np.sqrt(metrics.mean_squared_error(y_test, pred_lin01)))
print()
print("DecisionTreeRegressor:")
print("Mean absolute error:", metrics.mean_absolute_error(y_test, pred_dec01))
print("Mean squared error :", metrics.mean_squared_error(y_test, pred_dec01))
print("Root squared error :", np.sqrt(metrics.mean_squared_error(y_test, pred_dec01)))
print()
print("RandomForestRegressor:")
print("Mean absolute error:", metrics.mean_absolute_error(y_test, pred_rfc01))
print("Mean squared error :", metrics.mean_squared_error(y_test, pred_rfc01))
print("Root squared error :", np.sqrt(metrics.mean_squared_error(y_test, pred_rfc01)))

```

```
LinearRegression:
Mean absolute error: 2717.379954016268
Mean squared error : 18210092.674564373
Root squared error : 4267.3285173003

DecisionTreeRegressor:
Mean absolute error: 1641.3577854644222
Mean squared error : 9231177.698564775
Root squared error : 3038.285322112585

RandomForestRegressor:
Mean absolute error: 1325.4164684246298
Mean squared error : 6076575.911530212
Root squared error : 2465.0711777817314
```

✔ a sample shows this quite clearly:

```python

# test run - random sample:
df02.sample(5,random_state=80)

```

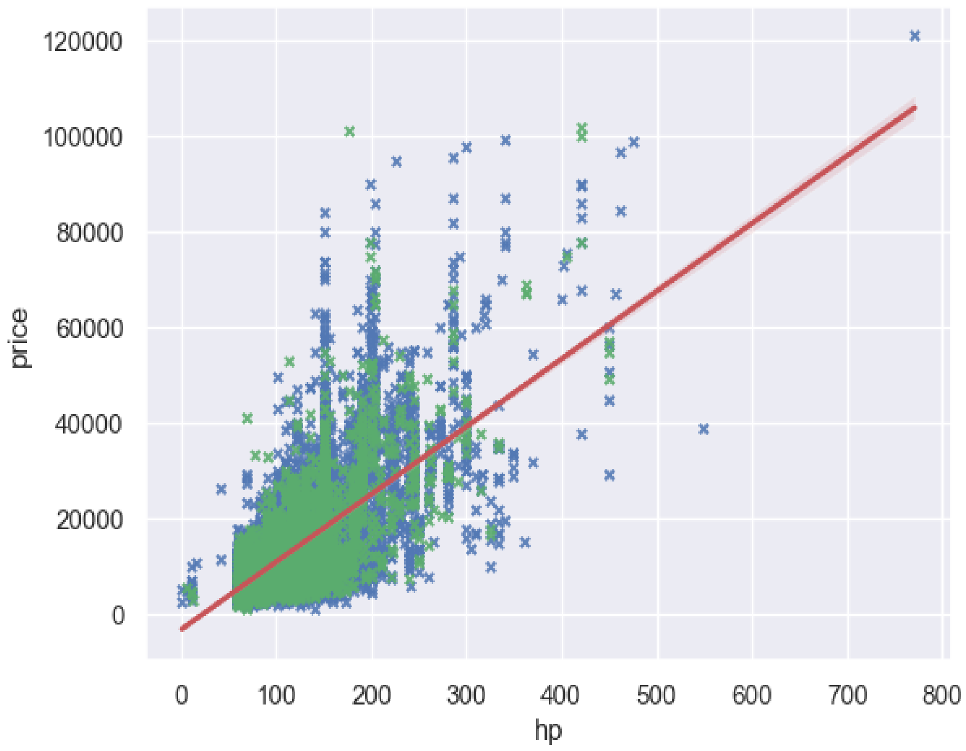
	mileage	make	model	fuel	gear	offerType	price	hp	year	encoded_make	encoded_model	encoded_fuel	encoded_gear	encc
36276	70000	Skoda	Roomster	Gasoline	Automatic	Used	9890	105.0	2013	64	629	7	0	
45196	10000	Volkswagen	Golf	Gasoline	Manual	Demonstration	28890	150.0	2020	72	396	7	1	
12829	20	Renault	Kangoo	Gasoline	Automatic	Used	15070	114.0	2019	61	452	7	0	
21289	15750	Volkswagen	Caddy	Diesel	Automatic	Demonstration	28750	150.0	2020	72	219	2	0	
9204	16551	Volkswagen	Polo	Gasoline	Manual	Used	11944	65.0	2018	72	566	7	1	

```python

# e.g. "renault kangoo":
pred_lin01_probe = lin01.predict([[20,114.0,61,452,7,0,4,8]])
pred_dec01_probe = dec01.predict([[20,114.0,61,452,7,0,4,8]])
pred_rfc01_probe = rfc01.predict([[20,114.0,61,452,7,0,4,8]])
print("LinearRegression: ",pred_lin01_probe,"; tatsächlicher Preis: 15070 €")
print()
print("DecisionTreeRegressor: ",pred_dec01_probe,"; tatsächlicher Preis: 15070 €")
print()
print("RandomForestRegressor: ",pred_rfc01_probe,"; tatsächlicher Preis: 15070 €")

```

```
LinearRegression:  [19613.41088692] ; tatsächlicher Preis: 15070 €

DecisionTreeRegressor:  [14916.66666667] ; tatsächlicher Preis: 15070 €

RandomForestRegressor:  [15083.14115296] ; tatsächlicher Preis: 15070 €
```

✔ The regression line using the example with the features with the highest correlation confirms this:

```
# visualization of the model

# comparison: feature with the strongest correlation is “hp” with the label “price”
# regression line in red
plt.scatter(X_train["hp"],y_train,color="b",alpha=0.9,s=15,marker="x")
plt.scatter(X_test["hp"],y_test,color="g",alpha=0.9,s=15,marker="x")
sns.regplot(x=df02["hp"], y=df02["price"], scatter=False, logx=False,color="r")
plt.show()
```



✔ Conclusion: The RandomForest regressor provides the most precise predictions and is the best in this comparison. But the results for DecisionTree are only slightly worse. The LinearRegressor is the clear loser because its errors are almost twice as large as those of the RandomForest.

. Overall conclusion:

- A manageable number of features allowed the data set to be processed and analyzed well
  - NaN values could be easily equalized; only a few samples had to be deleted
  - The “price” label was already there and therefore provided the direct target
- Very good results could be achieved in supervised learning with the features because the categorical values could be cleanly converted into numerical ones in order to work with the regression algorithms
- Evaluation of the 3 algorithms used:

Best Results:		Minimally worse results:		Worst results:	
RandomForestRegressor:		DecisionTreeRegressor:		LinearRegression:	
Mean absolute error:	1325.416	Mean absolute error:	1641.358	Mean absolute error:	2717.380
Mean squared error :	6076575.912	Mean squared error :	9231177.699	Mean squared error :	18210092.675
Root squared error :	2465.071	Root squared error :	3038.285	Root squared error :	4267.329

. Outlook:

1. Predictions: The more precise they are, the more credible they are and are therefore used more frequently. In addition to online activity, this also has a positive effect on the willingness to buy.
2. Marketing advantages:
    - Short-term as well as long-term developments can be identified quickly and efficiently. This means you can react in a timely manner and drive sales figures.