Analyzing Used Car Listings on eBay Kleinanzeigen

We will be working on a dataset of used cars from eBay Kleinanzeigen, a classifieds section of the German eBay website.

The dataset was originally scraped and uploaded to Kaggle. The version of the dataset we are working with is a sample of 50,000 data points that was prepared by Dataquest including simulating a less-cleaned version of the data.

The data dictionary provided with data is as follows:

```
dateCrawled - When this ad was first crawled. All field-values a
re taken from this date.
name - Name of the car.
seller - Whether the seller is private or a dealer.
offerType - The type of listing
price - The price on the ad to sell the car.
abtest - Whether the listing is included in an A/B test.
vehicleType - The vehicle Type.
yearOfRegistration - The year in which which year the car was fi
rst registered.
gearbox - The transmission type.
powerPS - The power of the car in PS.
model - The car model name.
kilometer - How many kilometers the car has driven.
monthOfRegistration - The month in which which year the car was
 first registered.
fuelType - What type of fuel the car uses.
brand - The brand of the car.
notRepairedDamage - If the car has a damage which is not yet rep
aired.
dateCreated - The date on which the eBay listing was created.
```

```
nrOfPictures - The number of pictures in the ad.
postalCode - The postal code for the location of the vehicle.
lastSeenOnline - When the crawler saw this ad last online.
```

The aim of this project is to clean the data and analyze the included used car listings.

import pandas as pd import numpy as np

In [4]:
```
import pandas as pd
import numpy as np

autos = pd.read_csv('autos.csv', encoding='Latin-1')
autos.info()
autos.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 20 columns):
dateCrawled            50000 non-null object
name                   50000 non-null object
seller                 50000 non-null object
offerType              50000 non-null object
price                  50000 non-null object
abtest                 50000 non-null object
vehicleType            44905 non-null object
yearOfRegistration     50000 non-null int64
gearbox                47320 non-null object
powerPS                50000 non-null int64
model                  47242 non-null object
odometer               50000 non-null object
monthOfRegistration    50000 non-null int64
fuelType               45518 non-null object
brand                  50000 non-null object
notRepairedDamage      40171 non-null object
dateCreated            50000 non-null object
nrOfPictures           50000 non-null int64
postalCode             50000 non-null int64
```

```
lastSeen                  50000 non-null object
dtypes: int64(5), object(15)
memory usage: 7.6+ MB
```

Out[4]:

	dateCrawled	name	seller	offerType	price	abtes
0	2016-03-26 17:47:46	Peugeot_807_160_NAVTECH_ON_BOARD	privat	Angebot	$5,000	contrc
1	2016-04-04 13:38:56	BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik	privat	Angebot	$8,500	contrc
2	2016-03-26 18:57:24	Volkswagen_Golf_1.6_United	privat	Angebot	$8,990	tes
3	2016-03-12 16:58:10	Smart_smart_fortwo_coupe_softouch/F1/Klima/Pan...	privat	Angebot	$4,350	contrc
4	2016-04-01 14:38:50	Ford_Focus_1_6_Benzin_TÜV_neu_ist_sehr_gepfleg...	privat	Angebot	$1,350	tes

autos' dataset contains 20 columns, most of columns stored as strings. There are a few columns with null values, but no columns have more than ~20% null values. There are some columns that contain dates stored as strings.

The columns names has been written with Camelcase which required to converted to Python' sneakcase.

However we will convert the column names from camelcase to snakecase and reword some of the column names based on the data dictionary to be more descriptive.

In [5]:
```
autos.columns
```

Out[5]:
```
Index(['dateCrawled', 'name', 'seller', 'offerType', 'price', 'abtest',
       'vehicleType', 'yearOfRegistration', 'gearbox', 'powerPS', 'mode
```

```
l',
       'odometer', 'monthOfRegistration', 'fuelType', 'brand',
       'notRepairedDamage', 'dateCreated', 'nrOfPictures', 'postalCod
e',
       'lastSeen'],
      dtype='object')
```

We'll make a few changes below: # Change the columns from camelcase to snakecase. # Change a few wordings to more accurately describe the columns.

In [6]:
```
autos.columns = ['date_crawled', 'name', 'seller', 'offer_type', 'pric
e', 'ab_test',
       'vehicle_type', 'registration_year', 'gearbox', 'power_ps', 'mod
el',
       'odometer', 'registration_month', 'fuel_type', 'brand',
       'unrepaired_damage', 'ad_created', 'num_photos', 'postal_code',
       'last_seen']
autos.head()
```

Out[6]:

	date_crawled	name	seller	offer_type	price	ab_
0	2016-03-26 17:47:46	Peugeot_807_160_NAVTECH_ON_BOARD	privat	Angebot	$5,000	cor
1	2016-04-04 13:38:56	BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik	privat	Angebot	$8,500	cor
2	2016-03-26 18:57:24	Volkswagen_Golf_1.6_United	privat	Angebot	$8,990	
3	2016-03-12 16:58:10	Smart_smart_fortwo_coupe_softouch/F1/Klima/Pan...	privat	Angebot	$4,350	cor
4	2016-04-01 14:38:50	Ford_Focus_1_6_Benzin_TÜV_neu_ist_sehr_gepfleg...	privat	Angebot	$1,350	

Exploring and cleaning for the numeric data which is unuseful for analysing as it is stored in text.

In [7]:
```
autos.describe(include='all') #  to get both categorical and numeric co
lumns.
```

Out[7]:

	date_crawled	name	seller	offer_type	price	ab_test	vehicle_type	registration_y
count	50000	50000	50000	50000	50000	50000	44905	50000.0000
unique	48213	38754	2	2	2357	2	8	N
top	2016-03-30 19:48:02	Ford_Fiesta	privat	Angebot	$0	test	limousine	N
freq	3	78	49999	49999	1421	25756	12859	N
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2005.0732
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	105.7128
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1000.0000
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1999.0000
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2003.0000
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2008.0000
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	9999.0000

Our initial observations above: There are a number of text columns where it's values are the same to each other as: (seller) with(offer_type) The (num_photos) column seems to be odd, we'll need to investigate this furthe. in addition, found that the (price) and (odometer) columns are numeric values stored as text.

In [8]:
```
print(autos["num_photos"].value_counts())
```

```
0    50000
Name: num_photos, dtype: int64
```

In [9]:
```
print(autos["seller"].value_counts())
```

```
privat        49999
gewerblich        1
Name: seller, dtype: int64
```

It looks like the num_photos column has 0 for every column. We'll drop this column, plus the other two we noted as mostly one value.

In [10]:
```
autos= autos.drop(["seller","offer_type","num_photos"], axis=1)
```

There are two columns, price and auto, which are numeric values with extra characters being stored as text. We'll clean and convert these.

In [11]:
```
autos["price"] = (autos["price"]
                          .str.replace("$","")
                          .str.replace(",","")
                          .astype(int)
                          )
autos["price"].head()
```

Out[11]:
```
0    5000
1    8500
2    8990
3    4350
4    1350
Name: price, dtype: int64
```

In [12]:
```
autos["odometer"] = (autos["odometer"]
                             .str.replace("km","")
                             .str.replace(",","")
                             .astype(int)
                             )
autos.rename({"odometer": "odometer_km"}, axis=1, inplace=True)
autos["odometer_km"].head()
```

Out[12]:
```
0    150000
1    150000
2     70000
3     70000
4    150000
Name: odometer_km, dtype: int64
```

Exploring odometer_km and price values.

```
In [13]: autos["odometer_km"].sort_index(ascending=False).head()
```

```
Out[13]: 49999    150000
         49998     40000
         49997      5000
         49996    150000
         49995    100000
         Name: odometer_km, dtype: int64
```

```
In [14]: print(autos["odometer_km"].unique().shape)
         autos["odometer_km"].describe()
```

```
(13,)
```

```
Out[14]: count     50000.000000
         mean     125732.700000
         std       40042.211706
         min        5000.000000
         25%      125000.000000
         50%      150000.000000
         75%      150000.000000
         max      150000.000000
         Name: odometer_km, dtype: float64
```

We can see that the values in this field are rounded, which might indicate that sellers had to choose from pre-set options for this column. Additionally, there are more high mileage than low mileage vehicles.

```
In [73]: autos["price"].unique()
```

```
Out[73]: array([ 5000,  8500,  8990, ...,   385, 22200, 16995])
```

```
In [74]: autos["price"].value_counts(normalize=False).sum()/autos.shape
```

```
Out[74]: array([1.00000000e+00, 2.94117647e+03])
```

```
In [75]: autos["price"].describe()
```

```
Out[75]: count    5.000000e+04
         mean     9.840044e+03
         std      4.811044e+05
         min      0.000000e+00
         25%      1.100000e+03
         50%      2.950000e+03
         75%      7.200000e+03
         max      1.000000e+08
         Name: price, dtype: float64
```

```
In [76]: autos["price"].value_counts().head(15)
```

```
Out[76]: 0        1421
         500       781
         1500      734
         2500      643
         1000      639
         1200      639
         600       531
         800       498
         3500      498
         2000      460
         999       434
         750       433
         900       420
         650       419
         850       410
         Name: price, dtype: int64
```

From the prices in this column seem rounded. However, there are 2357 unique values in the column, that may just be people's tend to round prices on the site. here are 1,421 cars listed with $0 price - with a percentage of only 2 %of the total number of cars, we might consider removing these rows. The maximum price is one hundred million dollars, which seems expensive.

```
In [77]: autos["price"].value_counts().sort_index(ascending=False).head(20)
```

```
Out[77]: 99999999    1
         27322222    1
         12345678    3
```

```
11111111    2
10000000    1
3890000     1
1300000     1
1234566     1
999999      2
999990      1
350000      1
345000      1
299000      1
295000      1
265000      1
259000      1
250000      1
220000      1
198000      1
197000      1
Name: price, dtype: int64
```

In [78]:
```
autos["price"].value_counts().sort_index(ascending=True).head(20)
```

Out[78]:
```
0     1421
1      156
2        3
3        1
5        2
8        1
9        1
10       7
11       2
12       3
13       2
14       1
15       2
17       3
18       1
20       4
25       5
29       1
```

```
30       7
35       1
Name: price, dtype: int64
```

There are a number of listings with prices below $30, including about 1,450 at $0. There are also approximately 14 listings number with very high values, which is over 1$. Given that eBay is an auction site, there could legitimately be items where the opening bid is $1. We will keep the $1 items, but remove anything above $350,000, as it looks that prices increase steadily to that number and then jump up to less realistic numbers.

In [79]:
```
autos = autos.loc[autos["price"].between(1,350001)]
autos["price"].describe()
```

Out[79]:
```
count     48565.000000
mean       5888.935591
std        9059.854754
min           1.000000
25%        1200.000000
50%        3000.000000
75%        7490.000000
max      350000.000000
Name: price, dtype: float64
```

There are a number of columns with date information that require to explore: .date_crawled .registration_month .registration_year .ad_created .last_seen These are a combination of dates that were crawled, and dates with meta-information from the crawler. The non-registration dates are stored as strings. We'll explore each of these columns to learn more about the listings.

In [80]:
```
autos[['date_crawled','ad_created','last_seen']][:5]
```

Out[80]:

	date_crawled	ad_created	last_seen
0	2016-03-26 17:47:46	2016-03-26 00:00:00	2016-04-06 06:45:54
1	2016-04-04 13:38:56	2016-04-04 00:00:00	2016-04-06 14:45:08
2	2016-03-26 18:57:24	2016-03-26 00:00:00	2016-04-06 20:15:37
3	2016-03-12 16:58:10	2016-03-12 00:00:00	2016-03-15 03:16:28

	date_crawled	ad_created	last_seen
4	2016-04-01 14:38:50	2016-04-01 00:00:00	2016-04-01 14:38:50

In [81]:
```
autos['date_crawled'].str[:10].value_counts(normalize=True,dropna=False
).sort_index()
```

Out[81]:
```
2016-03-05    0.025327
2016-03-06    0.014043
2016-03-07    0.036014
2016-03-08    0.033296
2016-03-09    0.033090
2016-03-10    0.032184
2016-03-11    0.032575
2016-03-12    0.036920
2016-03-13    0.015670
2016-03-14    0.036549
2016-03-15    0.034284
2016-03-16    0.029610
2016-03-17    0.031628
2016-03-18    0.012911
2016-03-19    0.034778
2016-03-20    0.037887
2016-03-21    0.037373
2016-03-22    0.032987
2016-03-23    0.032225
2016-03-24    0.029342
2016-03-25    0.031607
2016-03-26    0.032204
2016-03-27    0.031092
2016-03-28    0.034860
2016-03-29    0.034099
2016-03-30    0.033687
2016-03-31    0.031834
2016-04-01    0.033687
2016-04-02    0.035478
2016-04-03    0.038608
2016-04-04    0.036487
2016-04-05    0.013096
2016-04-06    0.003171
```

```
2016-04-07    0.001400
Name: date_crawled, dtype: float64
```

In [82]:
```python
autos["ad_created"].str[:10].value_counts(normalize=True,dropna=False).
sort_index()
```

Out[82]:
```
2015-06-11    0.000021
2015-08-10    0.000021
2015-09-09    0.000021
2015-11-10    0.000021
2015-12-05    0.000021
2015-12-30    0.000021
2016-01-03    0.000021
2016-01-07    0.000021
2016-01-10    0.000041
2016-01-13    0.000021
2016-01-14    0.000021
2016-01-16    0.000021
2016-01-22    0.000021
2016-01-27    0.000062
2016-01-29    0.000021
2016-02-01    0.000021
2016-02-02    0.000041
2016-02-05    0.000041
2016-02-07    0.000021
2016-02-08    0.000021
2016-02-09    0.000021
2016-02-11    0.000021
2016-02-12    0.000041
2016-02-14    0.000041
2016-02-16    0.000021
2016-02-17    0.000021
2016-02-18    0.000041
2016-02-19    0.000062
2016-02-20    0.000041
2016-02-21    0.000062
                ...
2016-03-09    0.033151
2016-03-10    0.031895
```

```
2016-03-11    0.032904
2016-03-12    0.036755
2016-03-13    0.017008
2016-03-14    0.035190
2016-03-15    0.034016
2016-03-16    0.030125
2016-03-17    0.031278
2016-03-18    0.013590
2016-03-19    0.033687
2016-03-20    0.037949
2016-03-21    0.037579
2016-03-22    0.032801
2016-03-23    0.032060
2016-03-24    0.029280
2016-03-25    0.031751
2016-03-26    0.032266
2016-03-27    0.030989
2016-03-28    0.034984
2016-03-29    0.034037
2016-03-30    0.033501
2016-03-31    0.031875
2016-04-01    0.033687
2016-04-02    0.035149
2016-04-03    0.038855
2016-04-04    0.036858
2016-04-05    0.011819
2016-04-06    0.003253
2016-04-07    0.001256
Name: ad_created, Length: 76, dtype: float64
```

In [83]:

```
autos["ad_created"].str[:10].value_counts(normalize=True, dropna=False)
.sort_values()
```

Out[83]:

```
2016-01-13    0.000021
2015-06-11    0.000021
2016-01-29    0.000021
2016-01-07    0.000021
2016-02-07    0.000021
2016-01-16    0.000021
```

```
2016-02-11    0.000021
2016-02-16    0.000021
2016-01-03    0.000021
2016-01-14    0.000021
2016-02-09    0.000021
2016-02-17    0.000021
2015-12-30    0.000021
2015-11-10    0.000021
2016-02-22    0.000021
2015-12-05    0.000021
2015-09-09    0.000021
2016-02-08    0.000021
2016-02-01    0.000021
2016-01-22    0.000021
2015-08-10    0.000021
2016-02-02    0.000041
2016-02-18    0.000041
2016-02-14    0.000041
2016-02-26    0.000041
2016-02-12    0.000041
2016-02-20    0.000041
2016-02-24    0.000041
2016-02-05    0.000041
2016-01-10    0.000041
                ...
2016-03-06    0.015320
2016-03-13    0.017008
2016-03-05    0.022897
2016-03-24    0.029280
2016-03-16    0.030125
2016-03-27    0.030989
2016-03-17    0.031278
2016-03-25    0.031751
2016-03-31    0.031875
2016-03-10    0.031895
2016-03-23    0.032060
2016-03-26    0.032266
2016-03-22    0.032801
2016-03-11    0.032904
```

```
2016-03-09    0.033151
2016-03-08    0.033316
2016-03-30    0.033501
2016-03-19    0.033687
2016-04-01    0.033687
2016-03-15    0.034016
2016-03-29    0.034037
2016-03-07    0.034737
2016-03-28    0.034984
2016-04-02    0.035149
2016-03-14    0.035190
2016-03-12    0.036755
2016-04-04    0.036858
2016-03-21    0.037579
2016-03-20    0.037949
2016-04-03    0.038855
Name: ad_created, Length: 76, dtype: float64
```

In [84]:
```
autos["last_seen"].str[:10].value_counts(normalize=True, dropna=False).
sort_index()
```

Out[84]:
```
2016-03-05    0.001071
2016-03-06    0.004324
2016-03-07    0.005395
2016-03-08    0.007413
2016-03-09    0.009595
2016-03-10    0.010666
2016-03-11    0.012375
2016-03-12    0.023783
2016-03-13    0.008895
2016-03-14    0.012602
2016-03-15    0.015876
2016-03-16    0.016452
2016-03-17    0.028086
2016-03-18    0.007351
2016-03-19    0.015834
2016-03-20    0.020653
2016-03-21    0.020632
2016-03-22    0.021373
```

```
2016-03-23    0.018532
2016-03-24    0.019767
2016-03-25    0.019211
2016-03-26    0.016802
2016-03-27    0.015649
2016-03-28    0.020859
2016-03-29    0.022341
2016-03-30    0.024771
2016-03-31    0.023783
2016-04-01    0.022794
2016-04-02    0.024915
2016-04-03    0.025203
2016-04-04    0.024483
2016-04-05    0.124761
2016-04-06    0.221806
2016-04-07    0.131947
Name: last_seen, dtype: float64
```

The crawler recorded the date it last saw any listing, which allows us to determine on what day a listing was removed, presumably because the car was sold. The last three days contain a disproportionate amount of 'last seen' values. Given that these are 6-10x the values from the previous days, it's unlikely that there was a massive spike in sales, and more likely that these values are to do with the crawling period ending and don't indicate car sales.

In [85]:
```
autos["ad_created"].str[:10].value_counts(normalize=True,dropna=False).
sort_index()
```

Out[85]:
```
2015-06-11    0.000021
2015-08-10    0.000021
2015-09-09    0.000021
2015-11-10    0.000021
2015-12-05    0.000021
2015-12-30    0.000021
2016-01-03    0.000021
2016-01-07    0.000021
2016-01-10    0.000041
2016-01-13    0.000021
2016-01-14    0.000021
2016-01-16    0.000021
2016-01-22    0.000021
```

```
2016-01-27    0.000062
2016-01-29    0.000021
2016-02-01    0.000021
2016-02-02    0.000041
2016-02-05    0.000041
2016-02-07    0.000021
2016-02-08    0.000021
2016-02-09    0.000021
2016-02-11    0.000021
2016-02-12    0.000041
2016-02-14    0.000041
2016-02-16    0.000021
2016-02-17    0.000021
2016-02-18    0.000041
2016-02-19    0.000062
2016-02-20    0.000041
2016-02-21    0.000062
                ...
2016-03-09    0.033151
2016-03-10    0.031895
2016-03-11    0.032904
2016-03-12    0.036755
2016-03-13    0.017008
2016-03-14    0.035190
2016-03-15    0.034016
2016-03-16    0.030125
2016-03-17    0.031278
2016-03-18    0.013590
2016-03-19    0.033687
2016-03-20    0.037949
2016-03-21    0.037579
2016-03-22    0.032801
2016-03-23    0.032060
2016-03-24    0.029280
2016-03-25    0.031751
2016-03-26    0.032266
2016-03-27    0.030989
2016-03-28    0.034984
2016-03-29    0.034037
```

```
2016-03-30    0.033501
2016-03-31    0.031875
2016-04-01    0.033687
2016-04-02    0.035149
2016-04-03    0.038855
2016-04-04    0.036858
2016-04-05    0.011819
2016-04-06    0.003253
2016-04-07    0.001256
Name: ad_created, Length: 76, dtype: float64
```

There is a large variety of ad created dates. Most fall within 1-2 months of the listing date, but a few are quite old, with the oldest at around 9 months.

In [86]:
```
autos["registration_year"].describe()
```

Out[86]:
```
count    48565.000000
mean      2004.755421
std         88.643887
min       1000.000000
25%       1999.000000
50%       2004.000000
75%       2008.000000
max       9999.000000
Name: registration_year, dtype: float64
```

The registration_year that the car was first registered means the age of cars. Meanwhile, the values shown of: The minimum value is 1000, before cars were invented. The maximum value is 9999, many years into the future. However,the data has been collected on 2016 and the count the number of listings with cars that fall outside the 1900 - 2016 interval and see if it's safe to remove those rows entirely. Dealing with Incorrect Registration Year Data Because a car can't be first registered before the listing was seen, any vehicle with a registration year above 2016 is definitely inaccurate. Determining the earliest valid year is more difficult. Realistically, it could be somewhere in the first few decades of the 1900s. One option is to remove the listings with these values. Let's determine what percentage of our data has invalid values in this column:

In [87]:
```
(~autos['registration_year'].between(1900, 2016)).sum() / (autos.shape[
0])
```

Out[87]:
```
0.038793369710697
```

Given that this is less than 4% of our data, we will remove these rows.

In [88]:
```
autos= autos.loc[autos["registration_year"].between(1900,2016)]
```

In [89]:
```
autos["registration_year"].value_counts(normalize=True).head(10)
```

Out[89]:
```
2000    0.067608
2005    0.062895
1999    0.062060
2004    0.057904
2003    0.057818
2006    0.057197
2001    0.056468
2002    0.053255
1998    0.050620
2007    0.048778
Name: registration_year, dtype: float64
```

It obviouss that most of the vehicles were first registered in the past 20 years.

Exploring Price against Brand.

In [90]:
```
autos["brand"].value_counts(normalize=True)
```

Out[90]:
```
volkswagen        0.211264
bmw               0.110045
opel              0.107581
mercedes_benz     0.096463
audi              0.086566
ford              0.069900
renault           0.047150
peugeot           0.029841
fiat              0.025642
seat              0.018273
skoda             0.016409
nissan            0.015274
mazda             0.015188
smart             0.014160
```

```
citroen          0.014010
toyota           0.012703
hyundai          0.010025
sonstige_autos   0.009811
volvo            0.009147
mini             0.008762
mitsubishi       0.008226
honda            0.007840
kia              0.007069
alfa_romeo       0.006641
porsche          0.006127
suzuki           0.005934
chevrolet        0.005698
chrysler         0.003513
dacia            0.002635
daihatsu         0.002506
jeep             0.002271
subaru           0.002142
land_rover       0.002099
saab             0.001649
jaguar           0.001564
daewoo           0.001500
trabant          0.001392
rover            0.001328
lancia           0.001071
lada             0.000578
Name: brand, dtype: float64
```

German manufacturers represent four out of the top five brands, almost 50% of the overall listings. Volkswagen is by far the most popular brand, with approximately double the cars for sale of the next two brands combined. There are lots of brands that don't have a significant percentage of listings, so we will limit our analysis to brands representing more than 5% of total listings.

In [91]:
```
brand_counts = autos["brand"].value_counts(normalize=True)
common_brands = brand_counts[brand_counts > .05].index
print(common_brands)
```

```
Index(['volkswagen', 'bmw', 'opel', 'mercedes_benz', 'audi', 'ford'], d
type='object')
```

In [92]:
```
brand_counts = autos["brand"].value_counts(normalize=True)
tops_brand = brand_counts[brand_counts > 0.05]
print(tops_brand)
```

```
volkswagen       0.211264
bmw              0.110045
opel             0.107581
mercedes_benz    0.096463
audi             0.086566
ford             0.069900
Name: brand, dtype: float64
```

In [93]:
```
brand_mean_prices = {}

for brand in common_brands:
    brand_only = autos[autos["brand"] == brand]
    mean_price = brand_only["price"].mean()
    brand_mean_prices[brand] = int(mean_price)

brand_mean_prices
```

Out[93]:
```
{'audi': 9336,
 'bmw': 8332,
 'ford': 3749,
 'mercedes_benz': 8628,
 'opel': 2975,
 'volkswagen': 5402}
```

The brand_mean_prices indicates that in the top 6 brands, there's a distinct price gap. Audi, BMW and Mercedes Benz are more expensive cars.in comparision with, Ford and Opel are less expensive. Volkswagen is in between - this may explain its popularity, it may be a 'best of 'both worlds' option.

In [94]:
```
bmp_series= pd.Series(brand_mean_prices)# transfer dictionary to serie
s.
print(bmp_series)
```

```
audi             9336
bmw              8332
ford             3749
mercedes_benz    8628
```

```
mercedes_benz    8628
opel             2975
volkswagen       5402
dtype: int64
```

In [95]:
```
bmp_dataframe= pd.DataFrame(bmp_series,columns=['mean_price'])
```

Explore Mileage by brand.

In [96]:
```
brand_mean_mileage = {}

for brand in common_brands:
    brand_only = autos.loc[autos["brand"] == brand]
    mean_mileage = brand_only["odometer_km"].mean()
#or mean_mileage = brand_only["odometer_km"].sum()/autos.shape
    brand_mean_mileage[brand] = int(mean_mileage)
brand_mean_mileage
```

Out[96]:
```
{'audi': 129157,
 'bmw': 132572,
 'ford': 124266,
 'mercedes_benz': 130788,
 'opel': 129310,
 'volkswagen': 128707}
```

In [97]:
```
mean_mileage = pd.Series(brand_mean_mileage).sort_values(ascending=Fals
e)
mean_prices = pd.Series(brand_mean_prices).sort_values(ascending=False)
```

In [98]:
```
brand_info = pd.DataFrame(mean_mileage, columns=["mean_mileage"])
brand_info
```

Out[98]:

	mean_mileage
bmw	132572
mercedes_benz	130788

	mean_mileage
opel	129310
audi	129157
volkswagen	128707
ford	124266

In [99]:
```
brand_info["mean_prices"]=mean_prices
brand_info
```

Out[99]:

	mean_mileage	mean_prices
bmw	132572	8332
mercedes_benz	130788	8628
opel	129310	2975
audi	129157	9336
volkswagen	128707	5402
ford	124266	3749

The range of car mileages does not vary as much as the prices do by brand, instead all falling within 10% for the top brands. There is a slight trend to the more expensive vehicles having higher mileage, with the less expensive vehicles having lower mileage.

In [100]:
```
autos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 46681 entries, 0 to 49999
Data columns (total 17 columns):
date_crawled          46681 non-null object
name                  46681 non-null object
price                 46681 non-null int64
ab_test               46681 non-null object
vehicle_type          43977 non-null object
registration_year     46681 non-null int64
```

```
gearbox                44571 non-null object
power_ps               46681 non-null int64
model                  44488 non-null object
odometer_km            46681 non-null int64
registration_month     46681 non-null int64
fuel_type              43363 non-null object
brand                  46681 non-null object
unrepaired_damage      38374 non-null object
ad_created             46681 non-null object
postal_code            46681 non-null int64
last_seen              46681 non-null object
dtypes: int64(6), object(11)
memory usage: 6.4+ MB
```

Identify categorical data that uses German words and translate them to English ones.

In [101]:
```
autos["unrepaired_damage"].unique()
```

Out[101]:
```
array(['nein', nan, 'ja'], dtype=object)
```

In [102]:
```
# translate values.
autos["unrepaired_damage"]= autos['unrepaired_damage'].str.replace("nei
n","no").str.replace("ja","yes")
```

In [103]:
```
autos["fuel_type"].unique()
```

Out[103]:
```
array(['lpg', 'benzin', 'diesel', nan, 'cng', 'hybrid', 'elektro',
       'andere'], dtype=object)
```

In [104]:
```
# translate values.
autos["fuel_type"]=autos['fuel_type'].str.replace("andere","other").str
.replace("elektro","electric")
```

In [105]:
```
autos["vehicle_type"].unique()
```

Out[105]:
```
array(['bus', 'limousine', 'kleinwagen', 'kombi', nan, 'coupe', 'suv',
       'cabrio', 'andere'], dtype=object)
```

In [106]:
```
# translate values.
vehicles_type = {
    'bus': 'bus',
    'limousine': 'limousine',
    'kleinwagen': 'small car',
    'kombi': 'station wagon',
    'nan': 'nan',
    'coupe': 'coupe',
    'suv': 'suv',
    'cabrio': 'convertible',
    'andere': 'other'
}
autos["vehicle_type"]= autos["vehicle_type"].map(vehicles_type)
```

In [107]:
```
autos["gearbox"].unique()
```

Out[107]:
```
array(['manuell', 'automatik', nan], dtype=object)
```

In [108]:
```
# translate values.
gearboxs={'manuell':"Manual",
  'automatik': "Automatic",
        "nan":"NaN"}
autos["gearbox"]=autos["gearbox"].map(gearboxs)
```

Let'Convert the dates to be uniform numeric data.

In [109]:
```
autos["ad_created"]=autos["ad_created"].str[:10].str.replace("-","").as
type(int)
```

In [110]:
```
autos["ad_created"][0:5]
```

Out[110]:
```
0    20160326
1    20160404
2    20160326
3    20160312
```

```
4    20160401
Name: ad_created, dtype: int64
```

In [111]:
```
autos["date_crawled"]=autos["date_crawled"].str[:10].str.replace("-",""
).astype(int)
```

In [112]:
```
autos["date_crawled"].head(5)
```

Out[112]:
```
0    20160326
1    20160404
2    20160326
3    20160312
4    20160401
Name: date_crawled, dtype: int64
```

In [113]:
```
autos.describe(include="all")
```

Out[113]:

	date_crawled	name	price	ab_test	vehicle_type	registration_year	gearbo
count	4.668100e+04	46681	46681.000000	46681	43977	46681.000000	4457
unique	NaN	35812	NaN	2	8	NaN	2
top	NaN	BMW_316i	NaN	test	limousine	NaN	Manua
freq	NaN	75	NaN	24062	12598	NaN	3471
mean	2.016033e+07	NaN	5977.716801	NaN	NaN	2002.910756	Na
std	3.192964e+01	NaN	9177.909479	NaN	NaN	7.185103	Na
min	2.016030e+07	NaN	1.000000	NaN	NaN	1910.000000	Na
25%	2.016031e+07	NaN	1250.000000	NaN	NaN	1999.000000	Na
50%	2.016032e+07	NaN	3100.000000	NaN	NaN	2003.000000	Na
75%	2.016033e+07	NaN	7500.000000	NaN	NaN	2008.000000	Na
max	2.016041e+07	NaN	350000.000000	NaN	NaN	2016.000000	Na

Extraction of keywords in the name column

Next we will try to extract key words form the name of the each listing .

In [114]:
```
autos["name"].unique()
```

Out[114]:
```
array(['Peugeot_807_160_NAVTECH_ON_BOARD',
       'BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik',
       'Volkswagen_Golf_1.6_United', ...,
       'Audi_Q5_3.0_TDI_qu._S_tr.__Navi__Panorama__Xenon',
       'Opel_Astra_F_Cabrio_Bertone_Edition___TÜV_neu+Reifen_neu_!!',
       'Fiat_500_C_1.2_Dualogic_Lounge'], dtype=object)
```

we can see that name string using "_" to seperate the words, so we will split the string by the underscore.

In [115]:
```
autos["name1"] = autos["name"].str.split('_').str.get(0)
autos["name2"] = autos["name"].str.split('_').str.get(1)
autos["name3"] = autos["name"].str.split("_").str.get(2)
```

In [116]:
```
autos["name1"].value_counts(normalize=True,dropna=False)[:5]
```

Out[116]:
```
Volkswagen    0.106103
Opel          0.088601
BMW           0.087659
Mercedes      0.082196
Audi          0.075363
Name: name1, dtype: float64
```

In [117]:
```
autos["name2"].value_counts(normalize=True,dropna=False).head(5)
```

Out[117]:
```
Benz     0.067951
Golf     0.052827
Corsa    0.026156
Polo     0.024271
Astra    0.023307
Name: name2, dtype: float64
```

In [118]:
```
autos["name3"].value_counts(normalize=True)[:5]
```

Out[118]:
```
2.0    0.043667
1.6    0.039472
1.4    0.036783
1.2    0.031661
       0.029830
Name: name3, dtype: float64
```

In [119]:
```
autos.loc[:5,["name1","name2","name3","price"]]
```

Out[119]:

	name1	name2	name3	price
0	Peugeot	807	160	5000
1	BMW	740i	4	8500
2	Volkswagen	Golf	1.6	8990
3	Smart	smart	fortwo	4350
4	Ford	Focus	1	1350
5	Chrysler	Grand	Voyager	7900

The most common brand/model combinations.

In [120]:
```
brand_model_combo = autos.groupby(["brand","model"]).count()
brand_model_combo
```

Out[120]:

		date_crawled	name	price	ab_test	vehicle_type	registration_year	gea
brand	**model**							
alfa_romeo	**145**	4	4	4	4	2	4	
	147	80	80	80	80	73	80	
	156	88	88	88	88	84	88	
	159	32	32	32	32	31	32	

brand	model	date_crawled	name	price	ab_test	vehicle_type	registration_year	gea
	andere	60	60	60	60	56	60	
	spider	32	32	32	32	32	32	
audi	100	57	57	57	57	55	57	
	200	1	1	1	1	1	1	
	80	198	198	198	198	179	198	
	90	8	8	8	8	7	8	
	a1	82	82	82	82	82	82	
	a2	42	42	42	42	40	42	
	a3	825	825	825	825	779	825	
	a4	1231	1231	1231	1231	1206	1231	
	a5	126	126	126	126	126	126	
	a6	797	797	797	797	778	797	
	a8	69	69	69	69	66	69	
	andere	216	216	216	216	214	216	
	q3	28	28	28	28	28	28	
	q5	62	62	62	62	62	62	
	q7	40	40	40	40	40	40	
	tt	144	144	144	144	140	144	
bmw	1er	521	521	521	521	502	521	
	3er	2615	2615	2615	2615	2526	2615	
	5er	1132	1132	1132	1132	1103	1132	
	6er	30	30	30	30	29	30	
	7er	126	126	126	126	123	126	
	andere	38	38	38	38	34	38	

		date_crawled	name	price	ab_test	vehicle_type	registration_year	gea
brand	**model**							
	i3	1	1	1	1	0	1	
	m_reihe	43	43	43	43	43	43	
...	**...**	...	...	...	...	...	...	
volkswagen	**andere**	96	96	96	96	90	96	
	beetle	123	123	123	123	119	123	
	bora	100	100	100	100	97	100	
	caddy	204	204	204	204	184	204	
	cc	18	18	18	18	17	18	
	eos	66	66	66	66	65	66	
	fox	82	82	82	82	79	82	
	golf	3707	3707	3707	3707	3414	3707	
	jetta	38	38	38	38	33	38	
	kaefer	57	57	57	57	49	57	
	lupo	322	322	322	322	298	322	
	passat	1349	1349	1349	1349	1306	1349	
	phaeton	31	31	31	31	31	31	
	polo	1609	1609	1609	1609	1484	1609	
	scirocco	85	85	85	85	81	85	
	sharan	222	222	222	222	210	222	
	tiguan	118	118	118	118	114	118	
	touareg	94	94	94	94	92	94	
	touran	433	433	433	433	415	433	
	transporter	674	674	674	674	650	674	
	up	51	51	51	51	49	51	

		date_crawled	name	price	ab_test	vehicle_type	registration_year	gea
brand	**model**							
volvo	**850**	28	28	28	28	27	28	
	andere	82	82	82	82	75	82	
	c_reihe	28	28	28	28	27	28	
	s60	17	17	17	17	17	17	
	v40	87	87	87	87	84	87	
	v50	29	29	29	29	28	29	
	v60	3	3	3	3	3	3	
	v70	91	91	91	91	87	91	
	xc_reihe	48	48	48	48	48	48	

290 rows × 18 columns

In [121]:

```python
common_combo = brand_model_combo.sort_values("price",ascending=False)
common_combo
```

Out[121]:

		date_crawled	name	price	ab_test	vehicle_type	registratio
brand	**model**						
volkswagen	**golf**	3707	3707	3707	3707	3414	
bmw	**3er**	2615	2615	2615	2615	2526	
volkswagen	**polo**	1609	1609	1609	1609	1484	
opel	**corsa**	1592	1592	1592	1592	1467	
volkswagen	**passat**	1349	1349	1349	1349	1306	
opel	**astra**	1348	1348	1348	1348	1274	
audi	**a4**	1231	1231	1231	1231	1206	

		date_crawled	name	price	ab_test	vehicle_type	registratio
brand	model						
mercedes_benz	c_klasse	1136	1136	1136	1136	1114	
bmw	5er	1132	1132	1132	1132	1103	
mercedes_benz	e_klasse	958	958	958	958	932	
audi	a3	825	825	825	825	779	
	a6	797	797	797	797	778	
ford	focus	762	762	762	762	728	
	fiesta	722	722	722	722	673	
volkswagen	transporter	674	674	674	674	650	
renault	twingo	615	615	615	615	568	
peugeot	2_reihe	600	600	600	600	579	
smart	fortwo	550	550	550	550	522	
opel	vectra	544	544	544	544	500	
mercedes_benz	a_klasse	539	539	539	539	498	
bmw	1er	521	521	521	521	502	
ford	mondeo	479	479	479	479	461	
renault	clio	473	473	473	473	451	
mercedes_benz	andere	439	439	439	439	430	
volkswagen	touran	433	433	433	433	415	
fiat	punto	415	415	415	415	370	
opel	zafira	394	394	394	394	372	
ford	ka	349	349	349	349	315	
renault	megane	335	335	335	335	319	
seat	ibiza	328	328	328	328	309	
...	...	...	...	...	...	...	

		date_crawled	name	price	ab_test	vehicle_type	registratic
brand	model						
daihatsu	move	9	9	9	9	9	
renault	r19	9	9	9	9	9	
land_rover	range_rover	9	9	9	9	9	
trabant	andere	8	8	8	8	6	
audi	90	8	8	8	8	7	
daewoo	nubira	8	8	8	8	8	
lada	andere	7	7	7	7	6	
smart	andere	7	7	7	7	7	
chrysler	crossfire	6	6	6	6	6	
seat	exeo	6	6	6	6	6	
volkswagen	amarok	6	6	6	6	6	
dacia	lodgy	5	5	5	5	5	
land_rover	range_rover_evoque	5	5	5	5	5	
saab	9000	5	5	5	5	5	
lancia	delta	5	5	5	5	3	
alfa_romeo	145	4	4	4	4	2	
daihatsu	materia	4	4	4	4	4	
fiat	croma	4	4	4	4	4	
land_rover	andere	4	4	4	4	4	
daihatsu	charade	3	3	3	3	2	
lada	samara	3	3	3	3	3	
volvo	v60	3	3	3	3	3	
dacia	andere	2	2	2	2	2	
lancia	kappa	2	2	2	2	2	

brand	model	date_crawled	name	price	ab_test	vehicle_type	registratio
rover	freelander	2	2	2	2	2	
ford	b_max	1	1	1	1	1	
rover	rangerover	1	1	1	1	1	
bmw	i3	1	1	1	1	0	
rover	discovery	1	1	1	1	1	
audi	200	1	1	1	1	1	

290 rows × 18 columns

In [122]:

```
common_combo = brand_model_combo["name"].sort_values(ascending=False)
common_combo
```

Out[122]:

```
brand          model
volkswagen     golf                3707
bmw            3er                 2615
volkswagen     polo                1609
opel           corsa               1592
volkswagen     passat              1349
opel           astra               1348
audi           a4                  1231
mercedes_benz  c_klasse            1136
bmw            5er                 1132
mercedes_benz  e_klasse             958
audi           a3                   825
               a6                   797
ford           focus                762
               fiesta               722
volkswagen     transporter          674
renault        twingo               615
peugeot        2_reihe              600
smart          fortwo               550
opel           vectra               544
```

```
mercedes_benz  a_klasse                539
bmw            1er                     521
ford           mondeo                  479
renault        clio                    473
mercedes_benz  andere                  439
volkswagen     touran                  433
fiat           punto                   415
opel           zafira                  394
ford           ka                      349
renault        megane                  335
seat           ibiza                   328
                                      ...
land_rover     range_rover               9
daewoo         andere                    9
renault        r19                       9
audi           90                        8
daewoo         nubira                    8
trabant        andere                    8
smart          andere                    7
lada           andere                    7
chrysler       crossfire                 6
seat           exeo                      6
volkswagen     amarok                    6
dacia          lodgy                     5
saab           9000                      5
lancia         delta                     5
land_rover     range_rover_evoque        5
fiat           croma                     4
daihatsu       materia                   4
land_rover     andere                    4
alfa_romeo     145                       4
daihatsu       charade                   3
lada           samara                    3
volvo          v60                       3
dacia          andere                    2
rover          freelander                2
lancia         kappa                     2
bmw            i3                        1
rover          rangerover                1
```

```
ford           b_max                      1
audi           200                        1
rover          discovery                  1
Name: name, Length: 290, dtype: int64
```

In [123]:
```python
spilt_odometer_km = autos.groupby("odometer_km").count()
spilt_odometer_km
```

Out[123]:

odometer_km	date_crawled	name	price	ab_test	vehicle_type	registration_year	gearbox	pow
5000	785	785	785	785	586	785	593	
10000	241	241	241	241	229	241	221	
20000	742	742	742	742	714	742	712	
30000	760	760	760	760	738	760	733	
40000	797	797	797	797	780	797	768	
50000	993	993	993	993	964	993	967	
60000	1128	1128	1128	1128	1096	1128	1100	
70000	1187	1187	1187	1187	1148	1187	1157	
80000	1375	1375	1375	1375	1331	1375	1338	
90000	1673	1673	1673	1673	1608	1673	1618	
100000	2058	2058	2058	2058	1964	2058	1957	
125000	4857	4857	4857	4857	4674	4857	4695	
150000	30085	30085	30085	30085	28145	30085	28712	

In [124]:
```python
odometer_price = autos.groupby("odometer_km")
odometer_price["price"].mean().sort_values(ascending=False)
```

Out[124]:
```
odometer_km
10000      20550.867220
```

```
20000     18448.477089
30000     16608.836842
40000     15499.568381
50000     13812.173212
60000     12385.004433
70000     10927.182814
80000      9721.947636
5000       8873.515924
90000      8465.025105
100000     8132.697279
125000     6214.022030
150000     3767.927107
Name: price, dtype: float64
```

It seems for listing values that cars with hightest mileage were cheap.

Estimate the damage cost.

It is certainely that Damaged cars are cheaper than non-damaged cars.But, lets evalute the gap price between them?

In [78]:
```
combm_unrepaired_cheap  = autos.groupby("unrepaired_damage").count()
combm_unrepaired_cheap
```

Out[78]:

	date_crawled	name	price	ab_test	vehicle_type	registration_year	gearbox
unrepaired_damage							
no	33834	33834	33834	33834	33074	33834	33175
yes	4540	4540	4540	4540	4244	4540	4381

In [82]:
```
damage_price = autos.groupby("unrepaired_damage")
damage_price_mean = damage_price["price"].mean()
damage_price_mean
```

Out[82]:
```
unrepaired_damage
no      7164.033103
yes     2241.146035
Name: price, dtype: float64
```

In [84]:
```
Gap_diference_price= damage_price_mean["no"]- damage_price_mean["yes"]
print("The average gap price is {:,.3f}".format(Gap_diference_price))
```

```
The average gap price is 4,922.887
```

In [ ]: