Exploring_Ebay_Car_Sales_Data

March 6, 2020

[1]:
```python
import pandas as pd
#Some .csv files can't be accessed without proper encoding method.
autos = pd.read_csv('autos.csv', encoding='Latin-1')
```

[2]:
```python
#To display the dataframe.
autos
```

[2]:
```
                dateCrawled                                              name  \
0       2016-03-26 17:47:46                     Peugeot_807_160_NAVTECH_ON_BOARD
1       2016-04-04 13:38:56           BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik
2       2016-03-26 18:57:24                           Volkswagen_Golf_1.6_United
3       2016-03-12 16:58:10  Smart_smart_fortwo_coupe_softouch/F1/Klima/Pan…
4       2016-04-01 14:38:50  Ford_Focus_1_6_Benzin_TÜV_neu_ist_sehr_gepfleg…
…                       …                                                 …
49995   2016-03-27 14:38:19   Audi_Q5_3.0_TDI_qu._S_tr.__Navi__Panorama__Xenon
49996   2016-03-28 10:50:25  Opel_Astra_F_Cabrio_Bertone_Edition___TÜV_neu+…
49997   2016-04-02 14:44:48                     Fiat_500_C_1.2_Dualogic_Lounge
49998   2016-03-08 19:25:42                 Audi_A3_2.0_TDI_Sportback_Ambition
49999   2016-03-14 00:42:12                                  Opel_Vectra_1.6_16V

       seller offerType    price   abtest vehicleType  yearOfRegistration  \
0      privat   Angebot   $5,000  control         bus                2004
1      privat   Angebot   $8,500  control   limousine                1997
2      privat   Angebot   $8,990     test   limousine                2009
3      privat   Angebot   $4,350  control  kleinwagen                2007
4      privat   Angebot   $1,350     test       kombi                2003
…         …         …        …        …           …                   …
49995  privat   Angebot  $24,900  control   limousine                2011
49996  privat   Angebot   $1,980  control      cabrio                1996
49997  privat   Angebot  $13,200     test      cabrio                2014
49998  privat   Angebot  $22,900  control       kombi                2013
49999  privat   Angebot   $1,250  control   limousine                1996

         gearbox  powerPS   model  odometer  monthOfRegistration fuelType  \
0        manuell      158  andere  150,000km                    3      lpg
1      automatik      286     7er  150,000km                    6   benzin
2        manuell      102    golf   70,000km                    7   benzin
```

```
3      automatik       71  fortwo   70,000km                     6   benzin
4        manuell        0   focus  150,000km                     7   benzin
…           …       …    …       …                     …      …
49995  automatik      239      q5  100,000km                     1   diesel
49996    manuell       75   astra  150,000km                     5   benzin
49997  automatik       69     500    5,000km                    11   benzin
49998    manuell      150      a3   40,000km                    11   diesel
49999    manuell      101  vectra  150,000km                     1   benzin

            brand notRepairedDamage          dateCreated  nrOfPictures  \
0         peugeot              nein  2016-03-26 00:00:00             0
1             bmw              nein  2016-04-04 00:00:00             0
2      volkswagen              nein  2016-03-26 00:00:00             0
3           smart              nein  2016-03-12 00:00:00             0
4            ford              nein  2016-04-01 00:00:00             0
…             …                 …                   …             …
49995        audi              nein  2016-03-27 00:00:00             0
49996        opel              nein  2016-03-28 00:00:00             0
49997        fiat              nein  2016-04-02 00:00:00             0
49998        audi              nein  2016-03-08 00:00:00             0
49999        opel              nein  2016-03-13 00:00:00             0

       postalCode             lastSeen
0           79588  2016-04-06 06:45:54
1           71034  2016-04-06 14:45:08
2           35394  2016-04-06 20:15:37
3           33729  2016-03-15 03:16:28
4           39218  2016-04-01 14:38:50
…             …                   …
49995       82131  2016-04-01 13:47:40
49996       44807  2016-04-02 14:18:02
49997       73430  2016-04-04 11:47:27
49998       35683  2016-04-05 16:45:07
49999       45897  2016-04-06 21:18:48

[50000 rows x 20 columns]
```

[3]:

```python
autos.info() #To display the info of the autos dataframe.
autos.head() #Displaying top 5 rows in the dataframe.
```

```
<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
```

[3]:

```
           dateCrawled                                                name  \
0  2016-03-26 17:47:46                     Peugeot_807_160_NAVTECH_ON_BOARD
1  2016-04-04 13:38:56           BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik
2  2016-03-26 18:57:24                           Volkswagen_Golf_1.6_United
3  2016-03-12 16:58:10  Smart_smart_fortwo_coupe_softouch/F1/Klima/Pan…
4  2016-04-01 14:38:50  Ford_Focus_1_6_Benzin_TÜV_neu_ist_sehr_gepfleg…

   seller offerType   price   abtest vehicleType  yearOfRegistration  \
0  privat   Angebot  $5,000  control         bus                2004
1  privat   Angebot  $8,500  control   limousine                1997
2  privat   Angebot  $8,990     test   limousine                2009
3  privat   Angebot  $4,350  control  kleinwagen                2007
4  privat   Angebot  $1,350     test       kombi                2003

     gearbox  powerPS   model   odometer  monthOfRegistration fuelType  \
0    manuell      158  andere  150,000km                    3      lpg
1  automatik      286     7er  150,000km                    6   benzin
2    manuell      102    golf   70,000km                    7   benzin
3  automatik       71  fortwo   70,000km                    6   benzin
4    manuell        0   focus  150,000km                    7   benzin

        brand notRepairedDamage          dateCreated  nrOfPictures  \
0     peugeot              nein  2016-03-26 00:00:00             0
1         bmw              nein  2016-04-04 00:00:00             0
2  volkswagen              nein  2016-03-26 00:00:00             0
3       smart              nein  2016-03-12 00:00:00             0
4        ford              nein  2016-04-01 00:00:00             0
```

```
   postalCode             lastSeen
0       79588  2016-04-06 06:45:54
1       71034  2016-04-06 14:45:08
2       35394  2016-04-06 20:15:37
3       33729  2016-03-15 03:16:28
4       39218  2016-04-01 14:38:50
```

PART ONE: Analysis on `autos` dataframe

- Upon looking at the information of the entire data frame, there are 5 out of a tota; 20 columns that have `null` or `NaN` data: `vehicle`, `gearbox`, `model`, `fuelType`, and `notRepairedDamage`. But none of the columns have more than 20% of its data as `null` values.
  - `vehicleType`: 5,095 `null`/`NaN` data
  - `gearbox`: 2,680 `null`/`NaN` data
  - `model`: 2,758 `null`/`NaN` data
  - `fuelType`: 4,482 `null`/`NaN` data
  - `notRepairedDamage`: 9,829 `null`/`NaN` data
- There are also 5 columns that have data that are `int` datatype: `yearOfRegistration`,`powerPS`,`monthofRegistration`,`nrOfPictures`,`postalCode`
- Also noticed some data columns have data that's spelled incorrectly, contain a combination of numbers and characters, and datetimes.
- I expected that we will be filling `null` cells with appropriate data, correcting misspelled words, and removing characters from number+char combos, creating datetime objects and extracting either the dates or the times, and converting the values (which would then be numbers) to either the int or float datatypes among other tasks.
- Some cells that have strings are also written in German, so I suspect that we would have to translate all of our non-english cells to english

[4]:
```
autos.columns #To display all unique column titles.
```

[4]:
```
Index(['dateCrawled', 'name', 'seller', 'offerType', 'price', 'abtest',
       'vehicleType', 'yearOfRegistration', 'gearbox', 'powerPS', 'model',
       'odometer', 'monthOfRegistration', 'fuelType', 'brand',
       'notRepairedDamage', 'dateCreated', 'nrOfPictures', 'postalCode',
       'lastSeen'],
      dtype='object')
```

[5]:
```
#Replacing column names to follow Python preferred snakecase format.
autos.rename(columns = {'yearOfRegistration' : 'registration_year',
                     'monthOfRegistration' : 'registration_month',
                     'notRepairedDamage' : 'unrepaired_damage',
                     'dateCreated' : 'ad_created',
                     'offerType' : 'offer_type',
                     'powerPS' : 'power_ps',
                     'nrOfPictures' : 'nr_of_pictures',
                     'postalCode' : 'postal_code',
                     'lastSeen' : 'last_seen',
                        'dateCrawled' : 'date_crawled',
```

```
                            'fuelType' : 'fuel_type',
                            'vehicleType' : 'vehicle_type',
                            'abtest' : 'ab_test'
                        }, inplace=True)
autos.columns
```

[5]:
```
Index(['date_crawled', 'name', 'seller', 'offer_type', 'price', 'ab_test',
       'vehicle_type', 'registration_year', 'gearbox', 'power_ps', 'model',
       'odometer', 'registration_month', 'fuel_type', 'brand',
       'unrepaired_damage', 'ad_created', 'nr_of_pictures', 'postal_code',
       'last_seen'],
      dtype='object')
```

PART TWO: Renaming columns. - We renamed the columns whose names were in the camelcase format to the snakecase format because snakecase is the format most preferred in Python.

[6]:
```
autos.describe() #Describing general information about columns in the dataframe␣
 ↪containing numeric values.
```

[6]:

	registration_year	power_ps	registration_month	nr_of_pictures	\
count	50000.000000	50000.000000	50000.000000	50000.0	
mean	2005.073280	116.355920	5.723360	0.0	
std	105.712813	209.216627	3.711984	0.0	
min	1000.000000	0.000000	0.000000	0.0	
25%	1999.000000	70.000000	3.000000	0.0	
50%	2003.000000	105.000000	6.000000	0.0	
75%	2008.000000	150.000000	9.000000	0.0	
max	9999.000000	17700.000000	12.000000	0.0	

	postal_code
count	50000.000000
mean	50813.627300
std	25779.747957
min	1067.000000
25%	30451.000000
50%	49577.000000
75%	71540.000000
max	99998.000000

PART THREE: Noting descriptive info. about table data - `nr_of-pictures` column has a mean of 0.0 (and everywhere else on the stats sheet) and a count of 50,000 which means that all the data in that column are 0s - `power_ps` which is another name for Horse Power(HP), has a min of 0.00, which means that there are cars registered that can't run at all. - `postal_code` column has a min value of 1067. Postal codes are meant to be 5 digits long which indicates that there are some postal codes that start with a zero which doesn't get shown. In theory, the data type in this column have to be changed to the `string` type instead of `int` or `float` types in order to show that zero in the beginning of each postal code.

[7]:
```
#Removing non-numeric values from columns we want to have as numeric, and then␣
↪converting them to numeric datatypes.
autos['price'] = autos['price'].str.replace('$','').str.replace(',','').
↪astype(float)
autos['odometer'] = autos['odometer'].str.replace('km','').str.replace(',','').
↪astype(float)
autos.rename(columns={'odometer':'odometer_km'}, inplace=True)
autos.head()
```

[7]:
```
          date_crawled                                              name  \
0  2016-03-26 17:47:46                     Peugeot_807_160_NAVTECH_ON_BOARD
1  2016-04-04 13:38:56           BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik
2  2016-03-26 18:57:24                           Volkswagen_Golf_1.6_United
3  2016-03-12 16:58:10  Smart_smart_fortwo_coupe_softouch/F1/Klima/Pan…
4  2016-04-01 14:38:50  Ford_Focus_1_6_Benzin_TÜV_neu_ist_sehr_gepfleg…

   seller offer_type   price   ab_test vehicle_type  registration_year  \
0  privat    Angebot  5000.0  control          bus               2004
1  privat    Angebot  8500.0  control    limousine               1997
2  privat    Angebot  8990.0     test    limousine               2009
3  privat    Angebot  4350.0  control   kleinwagen               2007
4  privat    Angebot  1350.0     test        kombi               2003

     gearbox  power_ps   model  odometer_km  registration_month fuel_type  \
0    manuell       158  andere     150000.0                   3       lpg
1  automatik       286     7er     150000.0                   6    benzin
2    manuell       102    golf      70000.0                   7    benzin
3  automatik        71  fortwo      70000.0                   6    benzin
4    manuell         0   focus     150000.0                   7    benzin

        brand unrepaired_damage           ad_created  nr_of_pictures  \
0     peugeot              nein  2016-03-26 00:00:00               0
1         bmw              nein  2016-04-04 00:00:00               0
2  volkswagen              nein  2016-03-26 00:00:00               0
3       smart              nein  2016-03-12 00:00:00               0
4        ford              nein  2016-04-01 00:00:00               0

   postal_code            last_seen
0        79588  2016-04-06 06:45:54
1        71034  2016-04-06 14:45:08
2        35394  2016-04-06 20:15:37
3        33729  2016-03-15 03:16:28
4        39218  2016-04-01 14:38:50
```

[8]:
```
autos['price'].unique().shape #To get the number of unique prices in the price␣
↪column.
```

[8]: (2357,)

[9]:
```
autos['price'].describe() #General info about price column after numeric data␣
↪conversion.
```

[9]:
```
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
```

PART FOUR: Identifying and removing outliers - Based on the descriptions given above, you can already see a few outliers. For instance: the cheapest car you can get which is the `min` price value is 0. - So essentially you would be getting those cars for free. Thus, you would want to remove free cars from your dataframe because the car very low to no value. - You can also see that the `max` is another outlier, as the most expensive car in the dataframe is listed at 100 million dollars. The most expensive car in the world today costs 13 million dollars. Therefore, you would want to remove any cars priced at more than 13 million. - Ideally, the most accurate listings can be found between 1,100 and 13 million dollars in my opinion.

[10]:
```
autos['price'].value_counts().head() #Displaying top 5 counts of specific␣
↪prices from the price column.
```

[10]:
```
0.0       1421
500.0      781
1500.0     734
2500.0     643
1200.0     639
Name: price, dtype: int64
```

[11]:
```
#To find the cars with the highest prices in autos to remove.
autos['price'].value_counts().sort_index(ascending=False).head()
```

[11]:
```
99999999.0    1
27322222.0    1
12345678.0    3
11111111.0    2
10000000.0    1
Name: price, dtype: int64
```

[39]:
```
#To find the cars with lowest prices that we don't since these cars are up for␣
↪bids.
autos['price'].value_counts().sort_index(ascending=True).head()
```

[39]:
```
1.0    150
2.0      2
3.0      1
5.0      2
8.0      1
Name: price, dtype: int64
```

[13]:
```
#To find unique odometer values.
autos['odometer_km'].unique().shape
```

[13]:
```
(13,)
```

[14]:
```
#Describing general statistics info. on 'odometer_km' column.
autos['odometer_km'].describe()
```

[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
```

[15]:
```
#Showing count of top 5 unique 'odometer_km' values in descending order
autos['odometer_km'].value_counts().head()
```

[15]:
```
150000.0    32424
125000.0     5170
100000.0     2169
90000.0      1757
80000.0      1436
Name: odometer_km, dtype: int64
```

[16]:
```
#Showing count of top 5 unique 'odometer_km' values in descending order.
autos['odometer_km'].value_counts().sort_index(ascending=False).head()
```

[16]:
```
150000.0    32424
125000.0     5170
100000.0     2169
90000.0      1757
80000.0      1436
Name: odometer_km, dtype: int64
```

[17]:
```
#Same data as above, just shown in ascending index order.
autos['odometer_km'].value_counts().sort_index(ascending=True).head()
```

[17]:
```
5000.0      967
10000.0     264
20000.0     784
30000.0     789
40000.0     819
Name: odometer_km, dtype: int64
```

[18]:
```
#General statistics info on 'registration_year' column.
autos["registration_year"].describe()
```

[18]:
```
count    50000.000000
mean      2005.073280
std        105.712813
min       1000.000000
25%       1999.000000
50%       2003.000000
75%       2008.000000
max       9999.000000
Name: registration_year, dtype: float64
```

[19]:
```
#Only looking at cars registered between 1900 and 2016. Anything registered␣
 ↪before or after that is incorrect data.
autos[autos["registration_year"].between(1900.0,2016.0)].describe()
```

[19]:
```
              price  registration_year      power_ps    odometer_km  \
count  4.802800e+04        48028.00000  48028.000000   48028.000000
mean   9.585252e+03         2002.80351    117.070417  125544.161739
std    4.843817e+05            7.31085    195.151278   40106.751417
min    0.000000e+00         1910.00000      0.000000    5000.000000
25%    1.150000e+03         1999.00000     71.000000  100000.000000
50%    2.990000e+03         2003.00000    107.000000  150000.000000
75%    7.400000e+03         2008.00000    150.000000  150000.000000
max    1.000000e+08         2016.00000  17700.000000  150000.000000

       registration_month  nr_of_pictures   postal_code
count        48028.000000         48028.0  48028.000000
mean             5.767760             0.0  50935.867327
std              3.696802             0.0  25792.079828
min              0.000000             0.0   1067.000000
25%              3.000000             0.0  30459.000000
50%              6.000000             0.0  49696.000000
75%              9.000000             0.0  71665.000000
max             12.000000             0.0  99998.000000
```

There are a few discrepencies in the `registration_year` column. The minimum year is listed at year 1000 which is well before the first car was invented, and max year listed at year 9999 which is well into the future. Thus, we'll only looked at cars registered between 1900 - 2016 which will remove any years less than year 1900, and years more than year 2016.

[20]:
```
#Looking for percentages of cars registered based on years.
autos = autos[autos["registration_year"].between(1900.0,2016.0)]
autos["registration_year"].value_counts(normalize=True).head(10).sort_values
```

[20]:
```
<bound method Series.sort_values of 2000    0.069834
2005    0.062776
1999    0.062464
2004    0.056988
2003    0.056779
2006    0.056384
2001    0.056280
2002    0.052740
1998    0.051074
2007    0.047972
Name: registration_year, dtype: float64>
```

We can see above that the majority of cars were sold between 1998 and 2016.

When we look at high price ranges for the column, we can see a significant jump from 350K dollars and up. Therefore it is safe to remove any data with prices 350K dollars or more. You can also that there are some cars less that 100 dollars. It should be safe to keep the prices of any car a dollar and up since Ebay is a site well-known for auctioning off its products.

[21]:
```
#Only looking at cars between $1 and $350,000 dollars in order to narrow down␣
↪common pricing a bit better.
autos = autos[autos['price'].between(1, 350000)]
autos.shape
```

[21]:
```
(46681, 20)
```

[22]:
```
#General statistical info. on 'price' column.
autos['price'].describe()
```

[22]:
```
count     46681.000000
mean       5977.716801
std        9177.909479
min           1.000000
25%        1250.000000
50%        3100.000000
75%        7500.000000
max      350000.000000
Name: price, dtype: float64
```

A few of our columns represent dates in the form of strings: - date_crawled - ad_created - last_seen

Here's a look at a few of rows of these columns down below

[23]:
```
#Looking at date columns.
autos[['date_crawled', 'ad_created', 'last_seen']].head()
```

[23]:

	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
4	2016-04-01 14:38:50	2016-04-01 00:00:00	2016-04-01 14:38:50

We're only interested in the dates, not the times.

[24]:
```
#Not worried about times, the dates so we'll be taking the 1st 10 characters in␣
↪each of the dates column.
(autos['date_crawled'].str[:10].value_counts(normalize=True, dropna=False).
↪sort_index())
```

[24]:
```
2016-03-05    0.025192
2016-03-06    0.014160
2016-03-07    0.036246
2016-03-08    0.033547
2016-03-09    0.033247
2016-03-10    0.032240
2016-03-11    0.032454
2016-03-12    0.036824
2016-03-13    0.015874
2016-03-14    0.036332
2016-03-15    0.034361
2016-03-16    0.029498
2016-03-17    0.031790
2016-03-18    0.012810
2016-03-19    0.034661
2016-03-20    0.038024
2016-03-21    0.037317
2016-03-22    0.032840
2016-03-23    0.032197
2016-03-24    0.029477
2016-03-25    0.031512
2016-03-26    0.032069
2016-03-27    0.030783
2016-03-28    0.034597
2016-03-29    0.034104
2016-03-30    0.033804
2016-03-31    0.031790
2016-04-01    0.033804
2016-04-02    0.035561
2016-04-03    0.038774
```

```
2016-04-04    0.036610
2016-04-05    0.013003
2016-04-06    0.003085
2016-04-07    0.001414
Name: date_crawled, dtype: float64
```

Let's do the same for our `ad_created` column

[25]:
```
#Looking at percentage of cars registered based on unique dates in 'ad_created'␣
↪column.
(autos['ad_created'].str[:10].value_counts(normalize=True, dropna=False).
↪sort_index())
```

[25]:
```
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
                …
2016-04-03    0.039009
2016-04-04    0.036953
2016-04-05    0.011782
2016-04-06    0.003170
2016-04-07    0.001264
Name: ad_created, Length: 74, dtype: float64
```

Now for our `last_seen` column

[26]:
```
#Looking at percentage of cars registered based on unique dates in 'ad_created'␣
↪column.
(autos['last_seen'].str[:10].value_counts(normalize=True, dropna=False).
↪sort_index())
```

[26]:
```
2016-03-05    0.001071
2016-03-06    0.004113
2016-03-07    0.005377
2016-03-08    0.007476
2016-03-09    0.009768
2016-03-10    0.010690
2016-03-11    0.012382
2016-03-12    0.023757
2016-03-13    0.008654
2016-03-14    0.012660
2016-03-15    0.016002
2016-03-16    0.016281
2016-03-17    0.028084
2016-03-18    0.007219
2016-03-19    0.015617
```

```
2016-03-20    0.020629
2016-03-21    0.020587
2016-03-22    0.020844
2016-03-23    0.018359
2016-03-24    0.019687
2016-03-25    0.018937
2016-03-26    0.016795
2016-03-27    0.015638
2016-03-28    0.020694
2016-03-29    0.022086
2016-03-30    0.024614
2016-03-31    0.023628
2016-04-01    0.022943
2016-04-02    0.024657
2016-04-03    0.025149
2016-04-04    0.024121
2016-04-05    0.125404
2016-04-06    0.223324
2016-04-07    0.132752
Name: last_seen, dtype: float64
```

The `last_seen` date column shows a spike in the last 3 days of sales. This is most likely due to the bidding war strategy when bidders typically wait until the last few days or the last day to make their final bids. The days prior can't have any relevant effect since the percentages are pretty evenly distributed.

[27]:
```
#Looking at percentage of cars registered based on unique brands.
autos['brand'].value_counts(normalize=True)
```

[27]:
```
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
```

The top 5 car brands on this list are all German made. The top German brand more than doubles the next car brand from the next country. We'll limit our analysis to brands that accounts for more than 5% of the total sales data

[28]:
```
brands = autos['brand'].value_counts(normalize = True)
most_common_brands = brands[brands > .05].index
most_common_brands
```

[28]:
```
Index(['volkswagen', 'bmw', 'opel', 'mercedes_benz', 'audi', 'ford'],
dtype='object')
```

[29]:
```
common_brand_dict={}
for brand in most_common_brands:
    selected_rows = autos[autos["brand"]==brand]
    mean_price = selected_rows["price"].mean()
    common_brand_dict[brand] = round(mean_price, 2)
import operator
brand_dict_sorted = sorted(common_brand_dict.items(), key=operator.
 ↪itemgetter(1), reverse=True)
print('Average price for each car brand in the top 6 in descending order:',␣
 ↪brand_dict_sorted)
```

```
Average price for each car brand in the top 6 in descending order: [('audi',
9336.69), ('mercedes_benz', 8628.45), ('bmw', 8332.82), ('volkswagen', 5402.41),
```

```
('ford', 3749.47), ('opel', 2975.24)]
```

As we can see, the cheapest commonly sold brands are `ford` and `opel`.

The most expensive commonly sold brands are`audi` and `mercedes_benz`.

The car brands commonly sold that are priced in between are `bmw` and `volkswagen`.

Out of the top 6 cars on the list, volkswagens are the most commonly sold car although it is priced in between which shows that customers not only value saving money, but they also value quality as well. Volkswagen cars are known for their top quality, safety, and engineering.

[30]:
```
#Create a series for common_brand_dict.
bmp_series = pd.Series(common_brand_dict)
print(bmp_series)
```

```
volkswagen       5402.41
bmw              8332.82
opel             2975.24
mercedes_benz    8628.45
audi             9336.69
ford             3749.47
dtype: float64
```

[31]:
```
#Convert common_brand_dict series to a Dataframe.
mean_price_df = pd.DataFrame(bmp_series, columns=['mean_price'])
mean_price_df
```

[31]:

	mean_price
volkswagen	5402.41
bmw	8332.82
opel	2975.24
mercedes_benz	8628.45
audi	9336.69
ford	3749.47

[32]:
```
avg_mileage_dict = {}
for brand in most_common_brands:
    selected_row = autos[autos['brand'] == brand]
    #Converting Kilometers to Miles.
    mileage = (selected_row['odometer_km'] / 1.609)
    mean_mileage = mileage.mean()
    #Round values to 2 decimal places.
    avg_mileage_dict[brand] = round(mean_mileage, 2)
```

[33]:
```
#Create a series for avg_mileage_dict
avg_mileage_series = pd.Series(avg_mileage_dict)
avg_mileage_series
```

[33]:
```
volkswagen      79992.02
bmw             82394.35
opel            80366.71
mercedes_benz   81285.50
audi            80271.84
ford            77231.83
dtype: float64
```

[34]:
```
#Convert avg_mileage_dict series to a Dataframe.
avg_mileage_df = pd.DataFrame(avg_mileage_series, columns=['avg_mileage'])
avg_mileage_df
```

[34]:

	avg_mileage
volkswagen	79992.02
bmw	82394.35
opel	80366.71
mercedes_benz	81285.50
audi	80271.84
ford	77231.83

[35]:
```
#Combine both Dataframes
pd.concat([mean_price_df, avg_mileage_df], axis=1)
```

[35]:

	mean_price	avg_mileage
volkswagen	5402.41	79992.02
bmw	8332.82	82394.35
opel	2975.24	80366.71
mercedes_benz	8628.45	81285.50
audi	9336.69	80271.84
ford	3749.47	77231.83

We observe that `avg_mileage` doesn't vary as much as the `avg_price`. We can see that the more expensive brands (BMV, Audi, and Mercedez Benz) tend to have higher mileages on average than the cheaper brands with Opel being the only exception.

[36]:
```
#Replacing German words with their English equivalents.
autos['seller'] = autos['seller'].replace('privat','private')
autos['offer_type'] = autos['offer_type'].replace('Angebot','Offer')
autos['vehicle_type'] = autos['vehicle_type'].replace('kleinwagen','small_car')
autos['vehicle_type'] = autos['vehicle_type'].replace('kombi','station_wagon')
autos['vehicle_type'] = autos['vehicle_type'].replace('cabrio', 'convertible')
autos['vehicle_type'] = autos['vehicle_type'].replace('andere','other')
autos['gearbox'] = autos['gearbox'].replace('automatik','automatic')
autos['gearbox'] = autos['gearbox'].replace('manuell', 'manual')
autos['unrepaired_damage'] = autos['unrepaired_damage'].replace('nein', 'no')
autos['unrepaired_damage'] = autos['unrepaired_damage'].replace('ja', 'yes')
```

Our dates columns currently contain dates and times. We want those columns to contain dates only and we also want to remove all the -s in those dates which will allow us to convert the dates to integers.

[37]:
```
#Retrieving just the date and not the times and then converting them to type␣
↪integers
autos['last_seen'] = autos['last_seen'].str[:10]
autos['date_crawled'] = autos['date_crawled'].str[:10]
autos['ad_created'] = autos['ad_created'].str[:10]
autos['last_seen'] = autos['last_seen'].str.replace('-','').astype(int)
autos['date_crawled'] = autos['date_crawled'].str.replace('-','').astype(int)
autos['ad_created'] = autos['ad_created'].str.replace('-','').astype(int)
autos
```

[37]:
```
       date_crawled                                       name  \
0          20160326                 Peugeot_807_160_NAVTECH_ON_BOARD
1          20160404       BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik
2          20160326                       Volkswagen_Golf_1.6_United
3          20160312  Smart_smart_fortwo_coupe_softouch/F1/Klima/Pan…
4          20160401  Ford_Focus_1_6_Benzin_TÜV_neu_ist_sehr_gepfleg…
…               …                                              …
49995      20160327   Audi_Q5_3.0_TDI_qu._S_tr.__Navi__Panorama__Xenon
49996      20160328  Opel_Astra_F_Cabrio_Bertone_Edition___TÜV_neu+…
49997      20160402                   Fiat_500_C_1.2_Dualogic_Lounge
49998      20160308               Audi_A3_2.0_TDI_Sportback_Ambition
49999      20160314                             Opel_Vectra_1.6_16V

        seller offer_type    price   ab_test   vehicle_type  registration_year  \
0      private      Offer   5000.0  control            bus               2004
1      private      Offer   8500.0  control      limousine               1997
2      private      Offer   8990.0     test      limousine               2009
3      private      Offer   4350.0  control      small_car               2007
4      private      Offer   1350.0     test  station_wagon               2003
…          …          …        …       …            …                 …
49995  private      Offer  24900.0  control      limousine               2011
49996  private      Offer   1980.0  control    convertible               1996
49997  private      Offer  13200.0     test    convertible               2014
49998  private      Offer  22900.0  control  station_wagon               2013
49999  private      Offer   1250.0  control      limousine               1996

         gearbox  power_ps   model  odometer_km  registration_month fuel_type  \
0         manual       158  andere     150000.0                   3       lpg
1      automatic       286     7er     150000.0                   6    benzin
2         manual       102    golf      70000.0                   7    benzin
3      automatic        71  fortwo      70000.0                   6    benzin
4         manual         0   focus     150000.0                   7    benzin
…            …       …       …          …                   …       …
49995  automatic       239      q5     100000.0                   1    diesel
```

```
49996     manual        75   astra     150000.0                    5    benzin
49997  automatic        69     500       5000.0                   11    benzin
49998     manual       150      a3      40000.0                   11    diesel
49999     manual       101  vectra     150000.0                    1    benzin

            brand unrepaired_damage  ad_created  nr_of_pictures  postal_code  \
0         peugeot                no    20160326               0        79588
1             bmw                no    20160404               0        71034
2      volkswagen                no    20160326               0        35394
3           smart                no    20160312               0        33729
4            ford                no    20160401               0        39218
…             …                 …         …              …           …
49995        audi                no    20160327               0        82131
49996        opel                no    20160328               0        44807
49997        fiat                no    20160402               0        73430
49998        audi                no    20160308               0        35683
49999        opel                no    20160313               0        45897

       last_seen
0       20160406
1       20160406
2       20160406
3       20160315
4       20160401
…            …
49995   20160401
49996   20160402
49997   20160404
49998   20160405
49999   20160406

[46681 rows x 20 columns]
```

Now let's take a look at whether or not there are price discrepencies based on whether or not cars with histories of damages have been repaired. We'll do this using a dictionary.

[38]:
```python
yes_no = ['yes','no']
#Creating a dictionary for damaged prices
damaged_prices = {}
#Finding the average price of each car based on whether or not they are damaged␣
↪and then storing them in dictionary
#before making final analysis.
for answer in yes_no:
    selected_rows = autos[autos['unrepaired_damage'] == answer]
    mean_price = selected_rows['price'].mean()
    damaged_prices[answer] = round(mean_price,2)
damaged_prices
```

[38]: {'yes': 2241.15, 'no': 7164.03}

We can see that cars with unrepaired damage are much cheaper than cars without damage on average.