Data Exploration of eBay Car Sales

In this Project, I will be cleaning, exploring and analysing a dataset of used cars from eBay Kleinanzeigen, a classified section of the German eBay website.

I will commence by importing Pandas and Munpy libraries I will be using for the data exploration

In [1]:
```
import pandas as pd
import numpy as np
```

Reading the autos.csv file

In [2]:
```
autos = pd.read_csv("Desktop/autos.csv" , encoding = "Latin-1")
```

To print information about the autos dataframe as well as take a look at the first few rows:

In [3]:
```
autos.info()
autos.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   dateCrawled          50000 non-null  object
 1   name                 50000 non-null  object
 2   seller               50000 non-null  object
 3   offerType            50000 non-null  object
 4   price                50000 non-null  object
 5   abtest               50000 non-null  object
 6   vehicleType          44905 non-null  object
 7   yearOfRegistration   50000 non-null  int64 
 8   gearbox              47320 non-null  object
 9   powerPS              50000 non-null  int64 
 10  model                47242 non-null  object
 11  odometer             50000 non-null  object
 12  monthOfRegistration  50000 non-null  int64 
 13  fuelType             45518 non-null  object
 14  brand                50000 non-null  object
 15  notRepairedDamage    40171 non-null  object
 16  dateCreated          50000 non-null  object
 17  nrOfPictures         50000 non-null  int64 
 18  postalCode           50000 non-null  int64 
 19  lastSeen             50000 non-null  object
dtypes: int64(5), object(15)
memory usage: 7.6+ MB
```

Out[3]:

	dateCrawled	name	seller	offerType	price	abte
0	3/26/2016 17:47	Peugeot_807_160_NAVTECH_ON_BOARD	privat	Angebot	$5,000	contr
1	4/4/2016 13:38	BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik	privat	Angebot	$8,500	contr
2	3/26/2016 18:57	Volkswagen_Golf_1.6_United	privat	Angebot	$8,990	te
3	3/12/2016 16:58	Smart_smart_fortwo_coupe_softouch/F1/Klima/Pan...	privat	Angebot	$4,350	contr
4	4/1/2016 14:38	Ford_Focus_1_6_Benzin_TÜV_neu_ist_sehr_gepfleg...	privat	Angebot	$1,350	te

From above dispalyed information, I can make the following observations:

1. The dataset contains 20 columns, with most of it being strings.
2. Some coolumns have null values
3. The columns names use camelCase instead of the Python's preferred snake_case, which means we cannot just replace spaces with underscores.

Data Cleaning

To make the daaset to be easier to work with, I will be cleaning the data.

Rewording the column names

To make the columns more descriptive based on the data dictionary, I will have to reword most of the column names and also change from the camelCase to the Python's snake_case.

In [4]:
```
autos.columns
```

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

Above shows the existing columns, and I will be editting them as below:

In [5]:
```
autos.rename(columns = {'dateCrawled' : 'date_crawled', 'name' : 'name', 'sell
er' : 'seller', 'offerType' : 'offer_type', 'price' : 'price', 'abtest' : 'abt
est',
       'vehicleType' : 'vehicle_type', 'yearOfRegistration' : 'registration_ye
ar', 'gearbox' : 'gearbox', 'powerPS' : 'power_ps', 'model' : 'model',
       'odometer' : 'odometer', 'monthOfRegistration' : 'registration_month',
'fuelType' : 'fuel_type', 'brand' : 'brand',
       'notRepairedDamage' : 'unrepaired_damage', 'dateCreated' : 'ad_created'
, 'nrOfPictures' : 'number_of_pictures', 'postalCode' : 'postal_code',
       'lastSeen' : 'last_seen'}, inplace = True)
```

In [6]:
```
autos.head()
```

Out[6]:

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

Data Exploration

To determine what other cleaning tasks need to be done, I use the autos.describe() with (include="all") to look at descriptive (numeric and categorical columns) statistics for all columns.

In [7]:
```
autos.describe(include = "all")
```

Out[7]:

	date_crawled	name	seller	offer_type	price	abtest	vehicle_type	registration_ye
count	50000	50000	50000	50000	50000	50000	44905	50000.0000
unique	12073	38754	2	2	2357	2	8	Na
top	3/28/2016 14:49	Ford_Fiesta	privat	Angebot	$0	test	limousine	Na
freq	15	78	49999	49999	1421	25756	12859	Na
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

The following are observed from above:

1. Seller and offer_type column have the same values.
2. number_of_pictures entries is zero.
3. The price and the odometer column are strings and have to be converted to numeric dtype to be able to work with.
4. registration_month has a minimum values of 0 and max of 12. This is false for minimum value of month cannot be has to be 1 for any given year.

I will be dropping the the seller, offer_type and the number_of_pictures columns as they all have either zero entries or same values.

In [8]:
```
autos = autos.drop(["seller","offer_type","number_of_pictures"], axis = 1)
```

In [9]:
```
autos.shape
```

Out[9]:
```
(50000, 17)
```

Now we have dropped the 3 columns, the dataset has 50000 rows and 17 columns.

Removing the strings in the price and odometer columns and converting the columns to numeric as below:

In [13]:
```
autos["price"].unique()
```

Out[13]:
```
array(['$5,000 ', '$8,500 ', '$8,990 ', ..., '$385 ', '$22,200 ',
       '$16,995 '], dtype=object)
```

I will remove the "$" sign and the "," in the unique values of the price shown above and convert to numeric

In [15]:
```
autos["price"] = (autos["price"]).str.replace("$" , "") .str.replace("," , "")
.astype(int)
```

In [16]:
```
autos["price"].unique()
```

Out[16]:
```
array([ 5000,  8500,  8990, ...,   385, 22200, 16995])
```

The prices are stored as above now.

For the Odometer, lets investigate the unique values:

In [18]:
```
autos["odometer"].unique()
```

Out[18]:
```
array(['150,000km', '70,000km', '50,000km', '80,000km', '10,000km',
       '30,000km', '125,000km', '90,000km', '20,000km', '60,000km',
       '5,000km', '100,000km', '40,000km'], dtype=object)
```

I will remove the "km" and "," and covert to numeric

In [19]:
```
autos["odometer"] = (autos["odometer"].str.replace("km" , "") .str.replace("," 
, "") .astype(int))
```

In [20]:
```
autos["odometer"].unique()
```

Out[20]:
```
array([150000,  70000,  50000,  80000,  10000,  30000, 125000,  90000,
        20000,  60000,   5000, 100000,  40000])
```

The odometer is stored as shown above now. Just so we do not lose important description whick is the kilometers, I will rename the odometer column to odometer_km

In [21]:
```
autos.rename(columns = {"odometer" : "odometer_km"}, inplace = True)
```

I will examine how the first few rows of my dataset looks like now

In [22]:
```
autos.head()
```

Out[22]:

	date_crawled	name	price	abtest	vehicle_type	r
0	3/26/2016 17:47	Peugeot_807_160_NAVTECH_ON_BOARD	5000	control	bus	
1	4/4/2016 13:38	BMW_740i_4_4_Liter_HAMANN_UMBAU_Mega_Optik	8500	control	limousine	
2	3/26/2016 18:57	Volkswagen_Golf_1.6_United	8990	test	limousine	
3	3/12/2016 16:58	Smart_smart_fortwo_coupe_softouch/F1/Klima/Pan...	4350	control	kleinwagen	
4	4/1/2016 14:38	Ford_Focus_1_6_Benzin_TÜV_neu_ist_sehr_gepfleg...	1350	test	kombi	

Analysing the price and odometer_km columns

In [23]:
```
autos["price"].unique().shape
```

Out[23]:
```
(2357,)
```

In [24]:
```
autos["price"].describe()
```

Out[24]:
```
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 [28]:
```
autos["price"].value_counts().sort_index().head(10) # to show the value counts
```

Out[28]:
```
0     1421
1      156
2        3
3        1
5        2
8        1
9        1
10       7
11       2
12       3
Name: price, dtype: int64
```

In [35]:
```
autos["price"].value_counts().sort_index(ascending = False).head(20)  # to sho
w the highest values of counts in descending order
```

Out[35]:
```
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 [30]:
```
autos["price"].value_counts().head(10)
```

Out[30]:
```
0       1421
500      781
1500     734
2500     643
1000     639
1200     639
600      531
800      498
3500     498
2000     460
Name: price, dtype: int64
```

From observation, there are 1421 entries with price list of 0, so it might be be worth removing this. Also listing can start from $1, however looking at the prices, I realized that the increase in listing after 350,000 increases from the thousands to millions which shows outliers. So I will be considering listing between 1 and 350,000

In [36]:
```
autos = autos[autos["price"].between(1, 350000)]
```

Now lets look at my description of the price column below:

In [38]:
```
autos["price"].describe()
```

Out[38]:
```
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
```

Now let me investigate the odometer_km column:

In [41]:
```
autos["odometer_km"].describe()
```

Out[41]:
```
count     48565.000000
mean     125770.101925
std       39788.636804
min        5000.000000
25%      125000.000000
50%      150000.000000
75%      150000.000000
max      150000.000000
Name: odometer_km, dtype: float64
```

In [40]:
```
autos["odometer_km"].value_counts().sort_index(ascending = False)
```

Out[40]:
```
150000    31414
125000     5057
100000     2115
90000      1734
80000      1415
70000      1217
60000      1155
50000      1012
40000       815
30000       780
20000       762
10000       253
5000        836
Name: odometer_km, dtype: int64
```

Vehicles with minimum mileage of 5000 km and maximum mileage of 150,000 km were listed. Higher mileages vehicles were most listed.

Exploring the date columns

From the auto.info(), I realized that the date_crawled, last-seen, and ad_created are all strings as recognised by pandas, while registration_year and registration_month are numeric.

Now let's look at the dates represented as strings:

In [42]:
```
autos[["date_crawled","ad_created","last_seen"]].head()
```

Out[42]:

	date_crawled	ad_created	last_seen
0	3/26/2016 17:47	3/26/2016 0:00	4/6/2016 6:45
1	4/4/2016 13:38	4/4/2016 0:00	4/6/2016 14:45
2	3/26/2016 18:57	3/26/2016 0:00	4/6/2016 20:15
3	3/12/2016 16:58	3/12/2016 0:00	3/15/2016 3:16
4	4/1/2016 14:38	4/1/2016 0:00	4/1/2016 14:38

The dates are represented as MM/DD/YYYY and are the first 10 characters.

Now selecting the first 10 characters in each of the three columns of dates above and sort from earliest to latest:

In [44]:
```
autos["date_crawled"].str[:10].value_counts(normalize=True, dropna=False).sort_values(ascending=False)
```

Out[44]:
```
3/20/2016     0.037887
3/21/2016     0.037373
3/12/2016     0.036920
3/14/2016     0.036549
3/28/2016     0.034860
                ...
4/4/2016 4    0.000021
4/7/2016 6    0.000021
4/1/2016 3    0.000021
4/4/2016 5    0.000021
3/6/2016 5    0.000021
Name: date_crawled, Length: 131, dtype: float64
```

Entries were made in the months of March and April 2016.

In [45]:
```
autos["ad_created"].str[:10].value_counts(normalize=True, dropna=False).sort_index()
```

Out[45]:
```
1/10/2016     0.000041
1/13/2016     0.000021
1/14/2016     0.000021
1/16/2016     0.000021
1/22/2016     0.000021
                ...
4/6/2016 0    0.003253
4/7/2016 0    0.001256
6/11/2015     0.000021
8/10/2015     0.000021
9/9/2015 0    0.000021
Name: ad_created, Length: 76, dtype: float64
```

Ad creation dates ranges from June 2015 to April 2016.

In [47]:
```
autos["last_seen"].str[:10].value_counts(normalize=True, dropna=False).sort_index()
```

Out[47]:
```
3/10/2016     0.010666
3/11/2016     0.012375
3/12/2016     0.023783
3/13/2016     0.008895
3/14/2016     0.012602
                ...
4/7/2016 5    0.009575
4/7/2016 6    0.009389
4/7/2016 7    0.009204
4/7/2016 8    0.008875
4/7/2016 9    0.008731
Name: last_seen, Length: 130, dtype: float64
```

The last seen dates tells us the date the listing was last seen, meaning the listing was removed because the car was sold on that day.

I will now look at the vehicle registration_year:

In [48]:
```
autos["registration_year"].describe()
```

Out[48]:
```
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 minimum registration year is 1000, this was way before cars were invented and Maximum is 9999 (many years in the future). Because cars are first registered before being put up for sales, and all the ads were created on or before 2016, any car registration year beyond 2016 is inaccurate.

For the minimum value, this is more difficult to know the minimum year and I will count the number of listings outside 1900 - 2016 and see if it is safe to eliminate the rest.

In [50]:
```
autos.loc[autos["registration_year"]<1950,"registration_year"]
```

Out[50]:
```
2221     1934
2573     1934
3679     1910
10556    1800
11047    1948
11246    1931
11585    1943
13963    1941
21416    1927
21421    1937
22101    1929
22316    1000
22659    1910
23804    1937
24511    1111
24855    1939
25792    1941
26103    1938
26607    1937
28693    1910
30781    1910
32585    1800
39725    1937
45157    1910
49283    1001
Name: registration_year, dtype: int64
```

I will proceed to assume valid listing to be from 1910 - 2016 and remove the rest of the entries.

In [52]:
```
autos = autos[autos["registration_year"].between(1910,2016)]
autos["registration_year"].value_counts(normalize=True).sort_values(ascending=
False)
```

Out[52]:
```
2000    0.067608
2005    0.062895
1999    0.062060
2004    0.057904
2003    0.057818
          ...
1931    0.000021
1929    0.000021
1943    0.000021
1953    0.000021
1952    0.000021
Name: registration_year, Length: 78, dtype: float64
```

Most vehicles on the listings were registered in the early 2000's

Exploring Sales by Brand

In [54]:
```
autos["brand"].value_counts(normalize=True)
```

Out[54]:
```
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
```

Volkswagen is the most popular brand on the list. I will look at brands that contribute more than 5% of the total listings values:

In [59]:
```
brand_greater_5 = autos["brand"].value_counts(normalize=True)
popular_brands = brand_greater_5[brand_greater_5 > 0.05].index

brand_mean_prices = {}
for brand in popular_brands:
    rows = autos[autos["brand"]==brand]
    mean_price = rows["price"].mean()
    brand_mean_prices[brand] = int(mean_price)

brand_mean_prices
```

Out[59]:
```
{'volkswagen': 5402,
 'bmw': 8332,
 'opel': 2975,
 'mercedes_benz': 8628,
 'audi': 9336,
 'ford': 3749}
```

Looking at these figures, Volkswagen, opel and ford are cheaper while BMW, Mercedes and Audi are expensive. It can be deduced that Volkwagen os most popular due to his performance compares to its price.

Finding a link between the mileage and the price

In [62]:
```
bmp_series = pd.Series(brand_mean_prices)
df = pd.DataFrame(bmp_series, columns=['mean_price'])
```

In [64]:
```
brand_mean_mileage = {}
for brand in popular_brands:
    rows = autos[autos["brand"]==brand]
    mean_mileage = rows["odometer_km"].mean()
    brand_mean_mileage[brand] = int(mean_mileage)

mean_mileage = pd.Series(brand_mean_mileage).sort_values(ascending=False)
mean_prices = pd.Series(brand_mean_prices).sort_values(ascending=False)
popularity = pd.Series(autos["brand"].value_counts(normalize=True).sort_values
(ascending=False))
```

In [65]:
```
brand_info = pd.DataFrame(mean_mileage,columns=["mean_mileage"])
brand_info["popularity"] = popularity
brand_info["mean_price"] = mean_prices
brand_info
```

Out[65]:

	mean_mileage	popularity	mean_price
bmw	132572	0.110045	8332
mercedes_benz	130788	0.096463	8628
opel	129310	0.107581	2975
audi	129157	0.086566	9336
volkswagen	128707	0.211264	5402
ford	124266	0.069900	3749

BMW nad Mercedes with higher prices has the higher mileages. With all the popular brands also the showing up as the ones with highest mileages, it can be deduce that popularity of the cars is a fuction of the mileages.