Exploratory Data Analysis Project

Factors influencing the price of a vehicle

Table of Contents

Goal

Prepare a report for the Crankshaft List to determine whether age, mileage, condition, transmission type, and color influence the price of a vehicle based on the data collected over the last few years.

Hypothesis

Based on common sense, we predict the following tendencies:

1. The older a vehicle, the lower the price;
2. The higher the mileage, the lower the price;
3. A vehicle with the condition 'new' should be more expensive than one with condition 'salvage';
4. A transmission type is probably customer dependent and should not have a clear connection to the price;
5. Color is also a preference of a particular customer but still some colors might be more popular than the others, so the price of a vehicle may be higher for a more popular color.

Description of the data

The dataset contains the following fields:

- `price`
- `model_year`
- `model`
- `condition`
- `cylinders`
- `fuel` — gas, diesel, etc.
- `odometer` — the vehicle's mileage when the ad was published
- `transmission`
- `paint_color`
- `is_4wd` — whether the vehicle has 4-wheel drive (Boolean type)
- `date_posted` — the date the ad was published
- `days_listed` — from publication to removal

Imports

In [111]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

import sys
import warnings
if not sys.warnoptions:
       warnings.simplefilter("ignore")

# pd.set_option('display.max_rows', None)

print("Setup Complete")
```

```
Setup Complete
```

Input data

In [112]:

```
try:
    df = pd.read_csv('vehicles_us.csv')
except:
    df = pd.read_csv('/datasets/vehicles_us.csv')
```

Overview

In [113]:

```
df.head(10)
```

Out[113]:

	price	model_year	model	condition	cylinders	fuel	odometer	transmission	type	pa
0	9400	2011.0	bmw x5	good	6.0	gas	145000.0	automatic	SUV	
1	25500	NaN	ford f-150	good	6.0	gas	88705.0	automatic	pickup	
2	5500	2013.0	hyundai sonata	like new	4.0	gas	110000.0	automatic	sedan	
3	1500	2003.0	ford f-150	fair	8.0	gas	NaN	automatic	pickup	
4	14900	2017.0	chrysler 200	excellent	4.0	gas	80903.0	automatic	sedan	
5	14990	2014.0	chrysler 300	excellent	6.0	gas	57954.0	automatic	sedan	
6	12990	2015.0	toyota camry	excellent	4.0	gas	79212.0	automatic	sedan	
7	15990	2013.0	honda pilot	excellent	6.0	gas	109473.0	automatic	SUV	
8	11500	2012.0	kia sorento	excellent	4.0	gas	104174.0	automatic	SUV	
9	9200	2008.0	honda pilot	excellent	NaN	gas	147191.0	automatic	SUV	

In [114]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51525 entries, 0 to 51524
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   price         51525 non-null  int64  
 1   model_year    47906 non-null  float64
 2   model         51525 non-null  object 
 3   condition     51525 non-null  object 
 4   cylinders     46265 non-null  float64
 5   fuel          51525 non-null  object 
 6   odometer      43633 non-null  float64
 7   transmission  51525 non-null  object 
 8   type          51525 non-null  object 
 9   paint_color   42258 non-null  object 
 10  is_4wd        25572 non-null  float64
 11  date_posted   51525 non-null  object 
 12  days_listed   51525 non-null  int64  
dtypes: float64(4), int64(2), object(7)
memory usage: 5.1+ MB
```

From the table above we can make the following conclusions:

- The data set includes 51525 observations and 13 columns;
- The `price` column is the target variable and the other 12 columns are features;
- 4 features are numerical, the other 9 are categorical;
- Float data type should be changed to integer;
- `is_4wd` is a float but it's a categorical (binary) variable is essence (True = 1, False = 0);
- `date_posted` column should be converted to a date_time format;
- Values are missing in the `model_year` , `cylinders` , `odometer` , `paint_color` , `is_4wd` (almost a half is missing);
- The column names seem to be correct.

In [115]:

```
df.describe()
```

Out[115]:

	price	model_year	cylinders	odometer	is_4wd	days_listed
count	51525.000000	47906.000000	46265.000000	43633.000000	25572.0	51525.00000
mean	12132.464920	2009.750470	6.125235	115553.461738	1.0	39.55476
std	10040.803015	6.282065	1.660360	65094.611341	0.0	28.20427
min	1.000000	1908.000000	3.000000	0.000000	1.0	0.00000
25%	5000.000000	2006.000000	4.000000	70000.000000	1.0	19.00000
50%	9000.000000	2011.000000	6.000000	113000.000000	1.0	33.00000
75%	16839.000000	2014.000000	8.000000	155000.000000	1.0	53.00000
max	375000.000000	2019.000000	12.000000	990000.000000	1.0	271.00000

From the table above we can make the following conclusions:

- The minimum value of the `price` variable is 1, so there is probably an error that should be fixed;
- In the `is_4wd` variable there is only one value category = '1.0'. It confirms our hypothesis that it's actually a binary feature, where '1' is 'True' and '0' is 'False'. All the missing values should therefore be replaced with '0';
- For the `price` variable we see that the mean is higher than the median value suggesting that the distribution has a long tail of large values.

Preprocessing

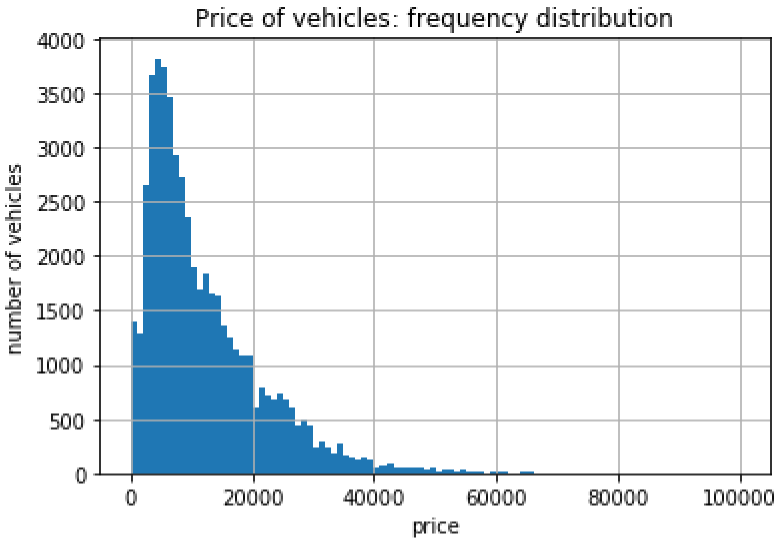
Artifacts

Price

First, let's look at our target variable `price` , see its distribution and clean up any artifacts.

In [116]:

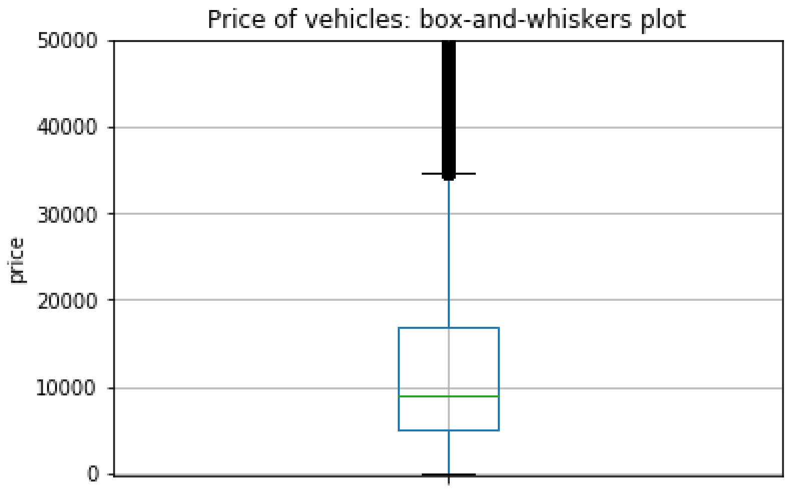
```
df.hist('price', bins=100, range=(0,100000))
plt.title('Price of vehicles: frequency distribution')
plt.xlabel('price')
plt.ylabel('number of vehicles');
```



We can see that the distribution is positively skewed with the peak at around 5000. Let's create a box plot for this variable to have a look at the interquartile range and measure its dispersion.

In [117]:

```
df.boxplot('price')
plt.ylim(-350, 50000)
plt.title('Price of vehicles: box-and-whiskers plot')
plt.xticks([1], [''])
plt.ylabel('price');
```

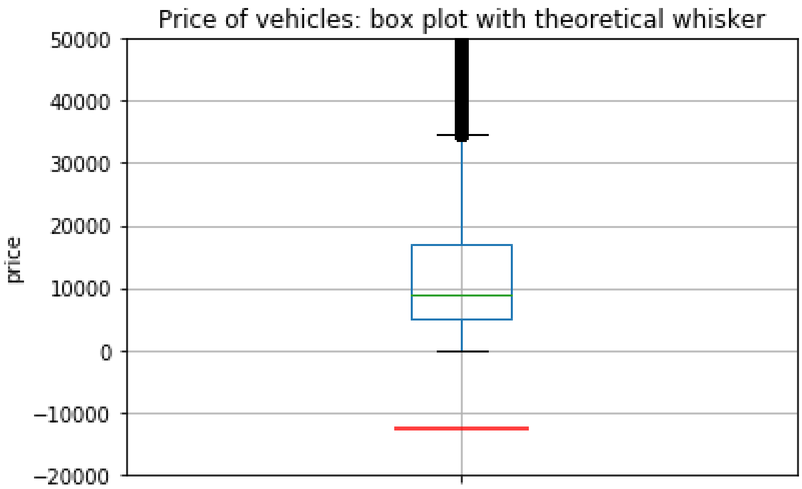


We can see that most values lie in the range between 5 000 and 17 000 with the median of around 9000. The whiskers turned out to be asymmetrical with respect to the box: the lower whisker is 0, while the upper whisker is 1.5 IQR above the upper limit of the box. Let's add quartiles and theoretical whisker values to the plot.

In [118]:

```
Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
IQR = Q3 - Q1
lower_whisker = Q1 - 1.5 * IQR
plt.ylim(-20000, 50000)
df.boxplot('price')
plt.hlines(y=lower_whisker, xmin=0.9, xmax=1.1, color='red')

plt.title('Price of vehicles: box plot with theoretical whisker')
plt.xticks([1], [''])
plt.ylabel('price');
```



We see that the theoretical value of the lower whisker is smaller than the minimum, that's why it was rendered at the minimum value before. Usually, anything above the upper whisker (and below the lower whisker) is considered an outlier, but since there are so many of them (the black line is thick), they are not outliers but rather a feature of this dataset. So these must be some super luxury cars, let's take a look at the car with the highest price in this data set.

In [119]:

```
df[df['price'] == df['price'].max()]
```

Out[119]:

	price	model_year	model	condition	cylinders	fuel	odometer	transmission	type
12504	375000	1999.0	nissan frontier	good	6.0	gas	115000.0	automatic	pickup

When googling the price of this particular model, it seems to be 10 times lower in reality (around 37 500 usd). This fact suggests that there has been some kind of a mistake (human or technical) in the data preparation.

Car brand

In this section we are going to identify observations with unusually high prices. It seems reasonable to assume that the price range is dependent on the car type and also on the car brand. First, we will extract the car brand from the model name and then group the data based on both type and brand.

In [120]:

```
df['model'].value_counts().head()
```

Out[120]:

```
ford f-150                  2796
chevrolet silverado 1500    2171
ram 1500                    1750
chevrolet silverado         1271
jeep wrangler               1119
Name: model, dtype: int64
```

We see that the first word in the model name is the car brand. Let's create a function that will extract the brand name.

In [121]:

```
def brand(row):
    """
    Takes in a vehicle's model and returns its brand
    """

    model = row['model']
    model_split = model.split(' ')
    brand = model_split[0]
    return brand
```

In [122]:

```
df['brand'] = df.apply(brand, axis=1)
```

In [123]:

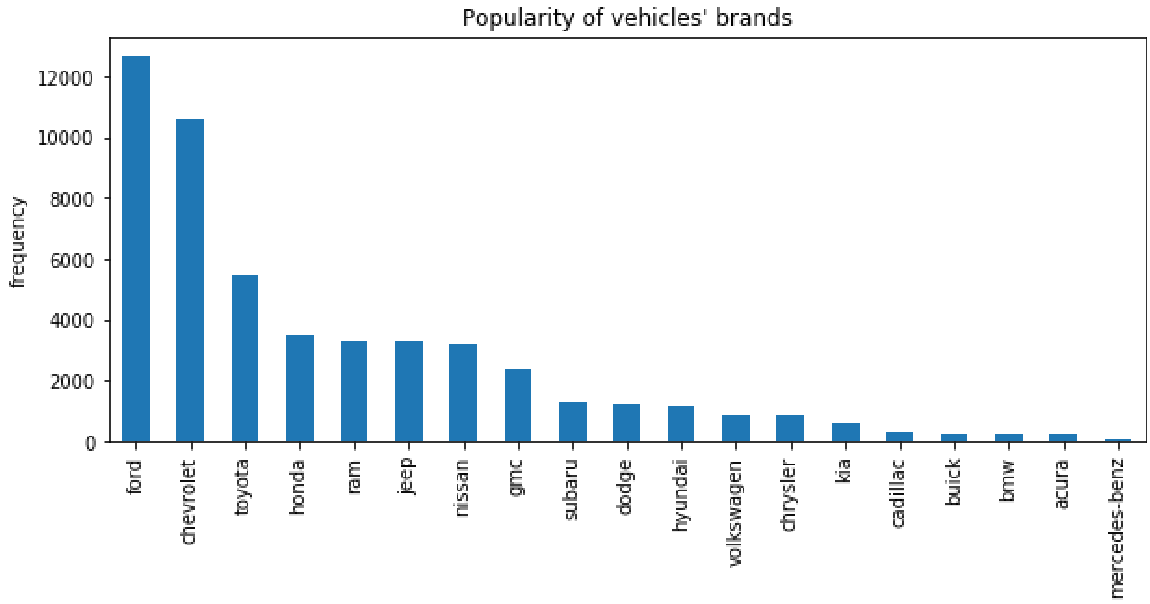
```
df['brand'].value_counts()
```

Out[123]:

```
ford             12672
chevrolet        10611
toyota            5445
honda             3485
ram               3316
jeep              3281
nissan            3208
gmc               2378
subaru            1272
dodge             1255
hyundai           1173
volkswagen         869
chrysler           838
kia                585
cadillac           322
buick              271
bmw                267
acura              236
mercedes-benz       41
Name: brand, dtype: int64
```

In [124]:

```
df['brand'].value_counts().plot(kind='bar', figsize=(10,4))
plt.title("Popularity of vehicles' brands")
plt.xlabel('brand')
plt.ylabel('frequency');
```



Ford, Chevrolet and Toyota are the most popular brands in this dataset. Mercedes-benz is the least popular brand in this dataset, probably because it's a luxury brand, so not as many people are able to afford it, compared to Ford, for instance.

Unusually high values

Let's now group the data based on both type and brand of a car. We will display the median price for each category, price of the 95th and 99th quantile and also count how many cars we have in these categories.

In [125]:

```
def q95(x):
    return np.quantile(x, 0.95)
def q99(x):
    return np.quantile(x, 0.99)
def over_q95_cnt(x):
    return sum(x > np.quantile(x, 0.95))
def over_q99_cnt(x):
    return sum(x > np.quantile(x, 0.99))
df.groupby(['brand','type']).agg(q50=('price', 'median'),
                               price_q95=('price', q95),
                               over_q95_cnt=('price', over_q95_cnt),
                               price_q99=('price', q99),
                               over_q99_cnt=('price', over_q99_cnt),
                               price_max=('price', 'max')).head()
```

Out[125]:

		q50	price_q95	over_q95_cnt	price_q99	over_q99_cnt	price_max
brand	**type**						
acura	**SUV**	4950.0	4950.0	0	4950.00	0	4950
	other	4595.0	5400.5	1	5472.10	1	5490
	sedan	5950.0	12995.0	10	13845.00	3	14498
bmw	**SUV**	9925.0	23999.0	11	36030.68	3	50000
	hatchback	6000.0	6000.0	0	6000.00	0	6000

Based on the above table, we will assume that all cars with their prices over the 98th quantile in the same group are unusual cases. They are either overpriced, or some luxury cases, or were just incorrectly inputted. We see that their number is minor and not impacting much this analysis, so we are going to exclude them.

In [126]:

```
q99 = df.groupby(['brand','type']).agg(price_q99=('price', q99))
q99_dict = q99.to_dict()
```

In [127]:

```
def filter_price_artifacts(row):
    """
    Takes in a row of a data frame, if the price of this row lower than limit pr
ice
    for this type and brand from the q99_dict, returns True, else returns False.
    """
    if row['price'] < q99_dict['price_q99'][(row['brand'], row['type'])]:
        result = True
    else:
        result = False
    return result
```

In [128]:

```
decently_priced = df.apply(filter_price_artifacts, axis=1)
df = df[decently_priced]
```

In [129]:

```
df.reset_index(drop=True, inplace=True)
```

In [130]:

```
df.shape
```

Out[130]:

```
(50802, 14)
```

723 observations were removed.

Other artifacts

While looking at the `price` we noticed another probable mistake: price = 12345. Let's correct it by replacing this value with the medians for the model and type.

In [131]:

```
df[df['price'] == 123456]
```

Out[131]:

price	model_year	model	condition	cylinders	fuel	odometer	transmission	type	paint_cc

In [132]:

```
df.loc[(df.index == 29810) | (df.index == 36822), 'price'] = df.query('model ==
 "chevrolet suburban" and type == "truck"')['price'].median()
```

In [133]:

```
df.loc[df.index == 42853, 'price'] = df.query('model == "chevrolet suburban" and
type == "SUV"')['price'].median()
```

Same issue in the `odometer` column. Let's also correct it with the median.

In [134]:

```
df[df['odometer'] == 123456]
```

Out[134]:

price	model_year	model	condition	cylinders	fuel	odometer	transmission	type	paint_cc

In [135]:

```
df.loc[df.index == 29810, 'odometer'] = df.query('model == "chevrolet suburban"
 and type == "truck"')['odometer'].median()
```

In [136]:

```
df.shape
```

Out[136]:

```
(50802, 14)
```

Missing values

First, let's drop all the rows with a NaN in each column.

In [137]:

```
df = df.dropna(how='all', axis=0)
```

In [138]:

```
df.shape
```

Out[138]:

```
(50802, 14)
```

There has been 720 such observations.

Is_4wd

Based on the above conclusions, let's replace all the missing values in this column with 0.

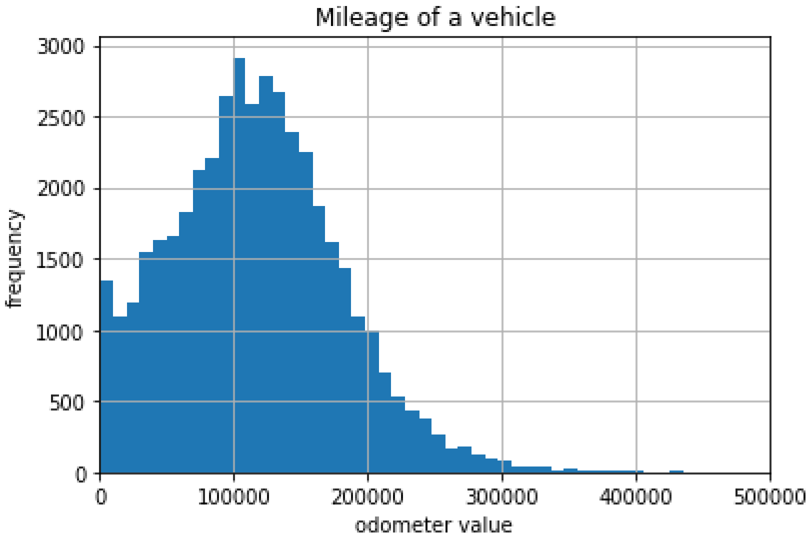
In [139]:

```
df['is_4wd'] = df['is_4wd'].fillna(0)
```

Odometer

In [140]:

```
df['odometer'].dropna().astype('int').hist(bins=100)
plt.xlim(0,500000)
plt.title("Mileage of a vehicle")
plt.xlabel('odometer value')
plt.ylabel('frequency');
```



The distribution looks more or less normal, a little positively skewed. We see a peak at 0, let's take a closer look at these rows.

In [141]:

```
df[df['odometer'] == 0].head(10)
```

Out[141]:

	price	model_year	model	condition	cylinders	fuel	odometer	transmission	
347	7997.0	2009.0	gmc yukon	excellent	8.0	gas	0.0	automatic	
805	2995.0	1999.0	ford f-150	good	6.0	gas	0.0	manual	
1356	5888.0	NaN	toyota 4runner	good	6.0	gas	0.0	automatic	
1442	1000.0	1992.0	gmc sierra 1500	good	8.0	gas	0.0	automatic	
1935	10988.0	2000.0	ford f-250 sd	good	8.0	diesel	0.0	automatic	
2014	30000.0	1969.0	chevrolet corvette	excellent	8.0	other	0.0	automatic	co
2042	11888.0	2010.0	chevrolet silverado 1500	good	8.0	gas	0.0	automatic	
2463	3000.0	2006.0	honda civic	good	NaN	gas	0.0	automatic	
3581	4200.0	NaN	nissan murano	good	6.0	gas	0.0	automatic	
4117	11888.0	2010.0	chevrolet silverado 1500	good	8.0	gas	0.0	automatic	

Based on these vehicle's model year, we can see that they are not new cars, the condition is sometimes only 'fair', so we will assume that the 0 value in the `odometer` column is also an error. We will replace it with 'NaN' and then fill all the 'NaN' values with the median based on grouping by `model` and `type` .

In [142]:

```
df.loc[df['odometer'] == 0, 'odometer'] = np.nan
```

In [143]:

```
df['odometer'] = df.groupby(['model','type'])['odometer'].apply(lambda x: x.fill
na(x.median()))
```

In [144]:

```
df['odometer'].isnull().sum()
```

Out[144]:

```
13
```

These 13 observations are unique in this data set (only 1 row with the model and type combination), so for them we don't have proper medians to fill in the `odometer` column. We are forced to exclude them from the data frame.

In [145]:

```
df = df.dropna(subset=['odometer'], axis=0)
df.reset_index(drop=True, inplace=True)
```

In [146]:

```
df.shape
```

Out[146]:

```
(50789, 14)
```

Model_year

First, we will find a median value in `odometer` column per year for each model and type. For missing `model_year` values we divide `odometer` value by median odometer per year value. It gives the number of years in exploitation. Then subtracting from the posting year the number of exploitation years gives us the missing `model_year` value.

Let's create a column `year_posted` first. For that we'll need to convert `date_posted` to datetime format.

In [147]:

```
df['date_posted']= pd.to_datetime(df['date_posted'])
df['year_posted'] = df['date_posted'].dt.year
```

Next, we'll subtract `model_year` from `year_posted` .

In [148]:

```
df['exploitation_years'] = df['year_posted'] - df['model_year']
```

Now let's calculate odometer per year value for each observation.

In [149]:

```
df['odometer_per_year'] = df['odometer'] / df['exploitation_years']
```

Next step is to find a median `odometer_per_year` value for each type and model combination.

In [150]:

```
odometer_per_year_dict = df.groupby(['model','type'])['odometer_per_year'].media
n().to_dict()
```

For each row where `model_year` is NaN, we will take `odometer` value and divide it by median `odometer_per_year` for this model and type combination and save this value in the `exploitation_years` column, then subtract `exploitation_years` from the `year-posted` column and save this value to the `model_year` column.

In [151]:

```
def fill_in_model_year(row):
    """
    Takes in a row and if the model_year of this row is NaN, takes `odometer` va
lue and
    divides it by median `odometer_per_year` for this model and type combination
    and saves this value in the `exploitation_years` column,
    then subtracts `exploitation_years` from the `year-posted` column and saves
 this value to the `model_year` column.
    """
    if np.isnan(row['model_year']):
        row['exploitation_years'] = row['odometer'] / odometer_per_year_dict[(ro
w['model'], row['type'])]
        row['model_year'] = row['year_posted'] - row['exploitation_years']
    return row
```

In [152]:

```
df = df.apply(fill_in_model_year, axis=1)
```

In [153]:

```
df['model_year'].isnull().sum()
```

Out[153]:

```
7
```

These 7 observations are unique in this data set (only 1 row with the model and type combination), so for them we don't have proper medians to fill in the `model_year` column. We are forced to exclude them from the data frame.

In [154]:

```
df = df.dropna(subset=['model_year'], axis=0)
df.reset_index(drop=True, inplace=True)

df.shape
```

Out[154]:

```
(50782, 17)
```

Finally, let's change the data type to integer.

In [155]:

```
df['model_year'] = df['model_year'].astype('int')
```

We don't need the `odometer_per_year` and `exploitation_years` columns anymore, so let's remove them.

In [156]:

```
df = df.drop(['odometer_per_year','exploitation_years'], axis=1)
df.reset_index(drop=True, inplace=True)

df.shape
```

Out[156]:

```
(50782, 15)
```

Cylinders

Missing `cylinders` values will be filled with the median value of a respective group based on the model and type of a vehicle. We assume that cars with the same model (e.g. cadillac escalade) but different types (e.g. SUV and pickup) have different number of cylinders because their engines were designed for different purposes.

In [157]:

```
df['cylinders'] = df.groupby(['model','type'])['cylinders'].apply(lambda x: x.fi
llna(x.median()))
```

In [158]:

```
df['cylinders'].isnull().sum()
```

Out[158]:

```
10
```

These 10 observations are unique in this data set (only 1 row with the model and type combination), so for them we don't have proper medians to fill in the `cylinders` column. We are forced to exclude them from the data frame.

In [159]:

```
df = df.dropna(subset=['cylinders'], axis=0)
df.reset_index(drop=True, inplace=True)

df.shape
```

Out[159]:

```
(50772, 15)
```

Paint color

In [160]:

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

Out[160]:

```
white     9844
black     7526
silver    6177
grey      4969
blue      4412
red       4372
green     1388
brown     1218
custom    1142
yellow     250
orange     224
purple     102
Name: paint_color, dtype: int64
```

In [161]:

```
df['paint_color'] = df['paint_color'].fillna('missing')
```

In [162]:

```
df.isnull().sum()
```

Out[162]:

```
price           0
model_year      0
model           0
condition       0
cylinders       0
fuel            0
odometer        0
transmission    0
type            0
paint_color     0
is_4wd          0
date_posted     0
days_listed     0
brand           0
year_posted     0
dtype: int64
```

All the missing values have been filled.

Data type change

Let's change all the float data types (except for `price` as it is supposed to be a float) into integers.

In [163]:

```
for column in ['price','odometer','cylinders','is_4wd','days_listed']:
    df[column] = df[column].astype('int')
```

In [164]:

```
df.dtypes
```

Out[164]:

```
price                     int64
model_year                int64
model                    object
condition                object
cylinders                 int64
fuel                     object
odometer                  int64
transmission             object
type                     object
paint_color              object
is_4wd                    int64
date_posted      datetime64[ns]
days_listed               int64
brand                    object
year_posted               int64
dtype: object
```

Duplicates

Let's check if any rows are duplicated.

In [165]:

```
df.duplicated().sum()
```

Out[165]:

0

Calculations

Month and DOW when the ad was posted

In [166]:

```
df['month_posted'] = df['date_posted'].dt.month
df['dow_posted'] = df['date_posted'].dt.dayofweek
```

Age in years

Next, we'll subtract calculate the age in years of each vehicle. To be more accurate, we will first convert the `model_year` column into datetime format, so that each year will be formated 01.01.year. Then we will convert the number of days past from the purchase of a car until the posting of an ad into years.

In [167]:

```
df['age_in_years'] = (df['date_posted'] - pd.to_datetime(df.model_year, format=
'%Y'))/ np.timedelta64(1, 'Y')
```

Average mileage per year

In [168]:

```
df['avg_miles_per_year'] = df['odometer']/df['age_in_years']
```

Condition

In [169]:

```
condition_dict = {5:'new', 4:'like new', 3:'excellent', 2:'good', 1:'fair', 0:'s
alvage'}
```

In [170]:

```
def get_key(val):
    """
    If the val can be found in the dictinary.values() list,
    returns the key of the dictionary item in which the val was found.
    """

    for key, value in condition_dict.items():
        if val in value:
            return key
```

In [171]:

```
def condition_num(row):
    """
    Takes in a string describing the vehicle's condition and returns an integer
 according to the condition_dict.
    """
    if row['condition'] in condition_dict.values():
        return get_key(row['condition'])
```

In [172]:

```
df['condition'] = df.apply(condition_num, axis=1)
df['condition'].value_counts()
```

Out[172]:

```
3    24399
2    20010
4     4534
1     1602
0      115
5      112
Name: condition, dtype: int64
```

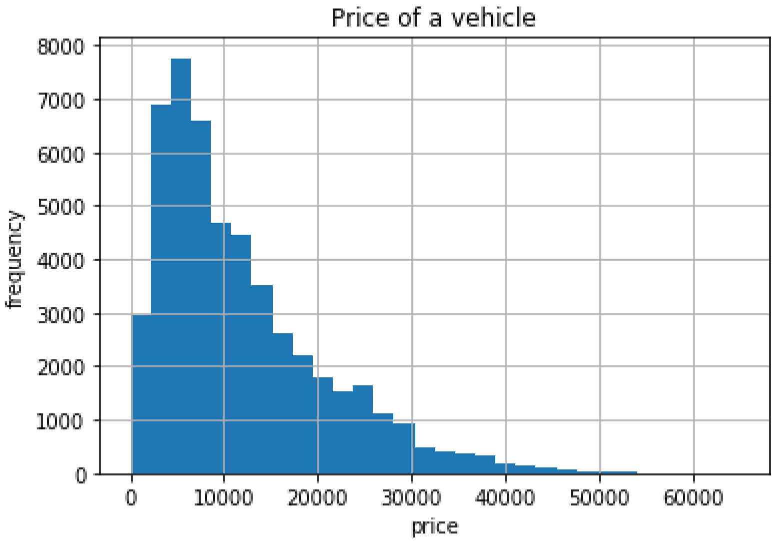
EDA

Histograms

Price

In [173]:

```
df.hist('price', bins=30)
plt.title("Price of a vehicle")
plt.xlabel('price')
plt.ylabel('frequency');
```



The `price` distribution looks normal, positively skewed due to some luxury models but overall sensible. We have got rid of abnormally high prices earlier, now let's look at abnormally low prices.

In [174]:

```
len(df[df['price'] == 1])
```

Out[174]:

```
798
```

There are 798 vehicles with the price of 1 USD. We will assume it's an error. Let's replace them with the median price for the type and brand combination.

In [175]:

```
df.loc[df['price'] == 1, 'price'] = np.nan
```

In [176]:

```
df['price'] = df.groupby(['model','type'])['price'].apply(lambda x: x.fillna(x.m
edian()))
```

In [177]:

```
df['price'].isnull().sum()
```

Out[177]:

```
1
```

This 1 observation is unique in this data set (only 1 row with the model and type combination), so we don't have a proper median to fill it in. We are forced to exclude this row from the data frame.

In [178]:

```
df = df.dropna(subset=['price'], axis=0)
df.reset_index(drop=True, inplace=True)

df.shape
```

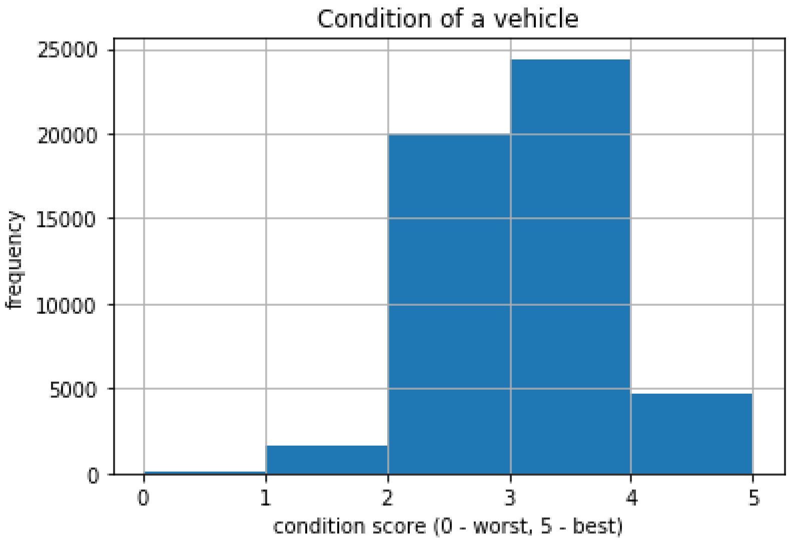
Out[178]:

```
(50771, 19)
```

Condition

In [179]:

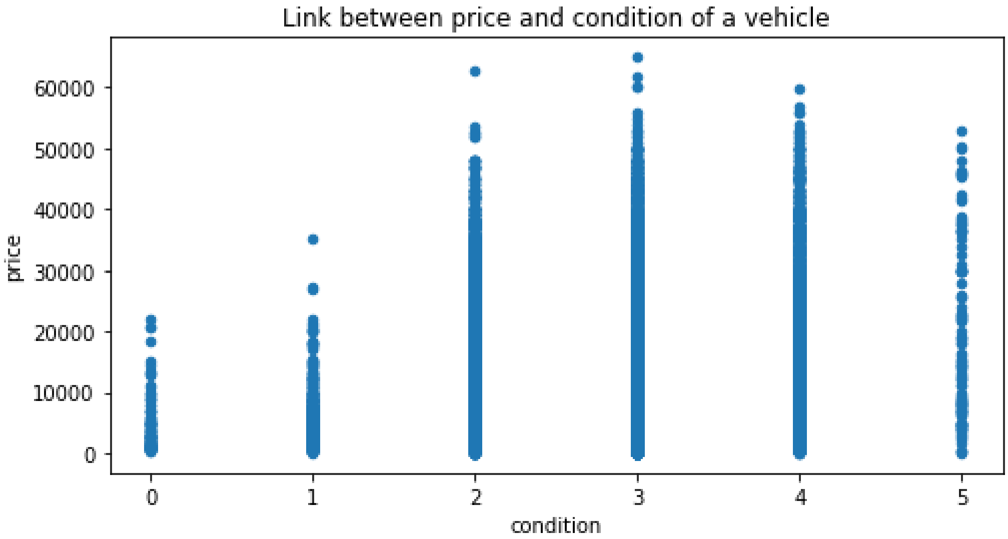
```
df.hist('condition', bins=5)
plt.title("Condition of a vehicle")
plt.xlabel('condition score (0 - worst, 5 - best)')
plt.ylabel('frequency');
```



Most vehicles are in "like new" and "excellent". Just a few are in a "salvage" condition. Let's see if there is a link between price and condition.

In [180]:

```
df.plot(x='condition', y='price', kind='scatter', figsize=(8,4))
plt.title('Link between price and condition of a vehicle');
```

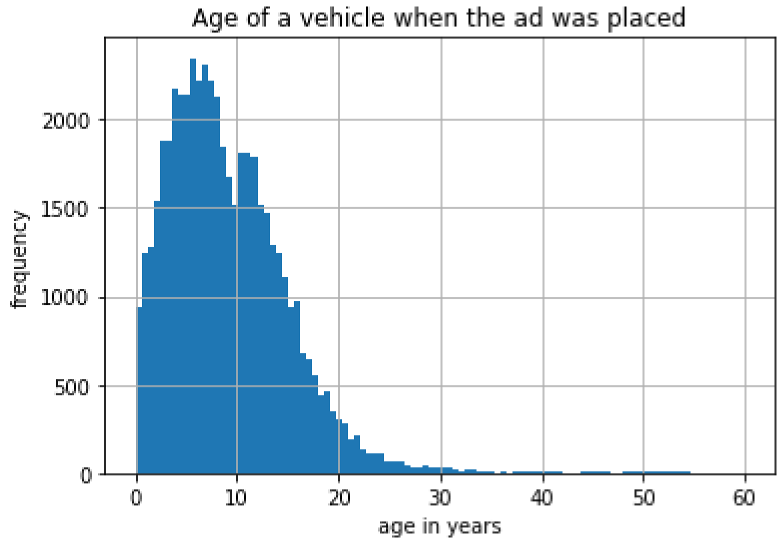


Although the price is in fact lower for vehicles with worse condition, we don't see an obvious tendency that the better the condition, the higher the price. It is probably due to other vehicle's features like number of cylinders, model, brand, type and so on.

Age of a vehicle when the ad was placed

In [181]:

```
df.hist('age_in_years', bins=100, range=(0,60))
plt.title("Age of a vehicle when the ad was placed")
plt.xlabel('age in years')
plt.ylabel('frequency');
```



In [182]:

```
print('{:0.2f}'.format(df['age_in_years'].mean()))
```

9.12

In [183]:

```
print('{:0.2f}'.format(df['age_in_years'].median()))
```

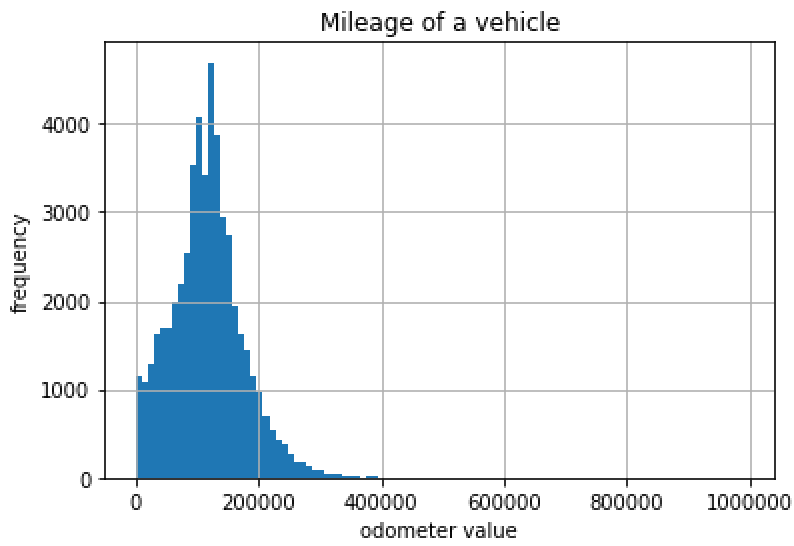
8.08

The average and median age of a vehicle in this data set is similar - around 7-8 years. We also see quite a long tail of large values, we will remove those that we consider outliers in the next section.

Odometer

In [184]:

```
df.hist('odometer', bins=100)
plt.title("Mileage of a vehicle")
plt.xlabel('odometer value')
plt.ylabel('frequency');
```

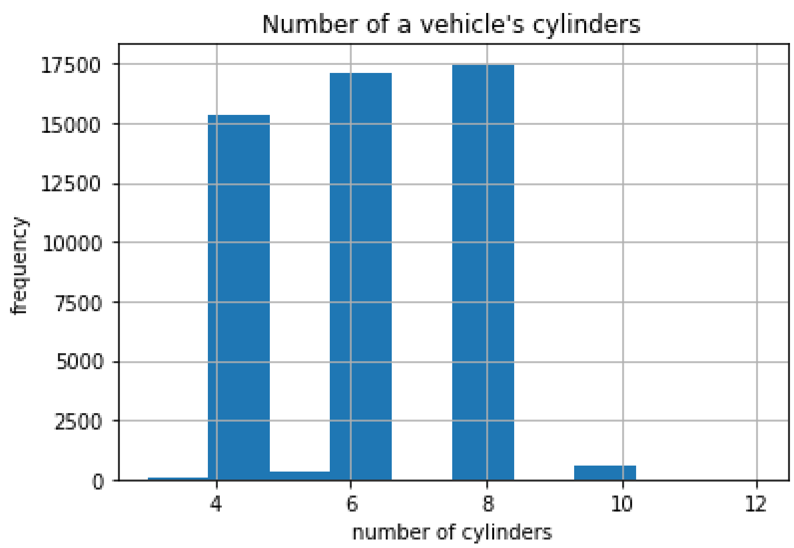


Again, there is a long tail of large values that skew our distribution, we will identify and remove outliers in the next section.

Cylinders

In [185]:

```
df.hist('cylinders', bins=10)
plt.title("Number of a vehicle's cylinders")
plt.xlabel('number of cylinders')
plt.ylabel('frequency');
```

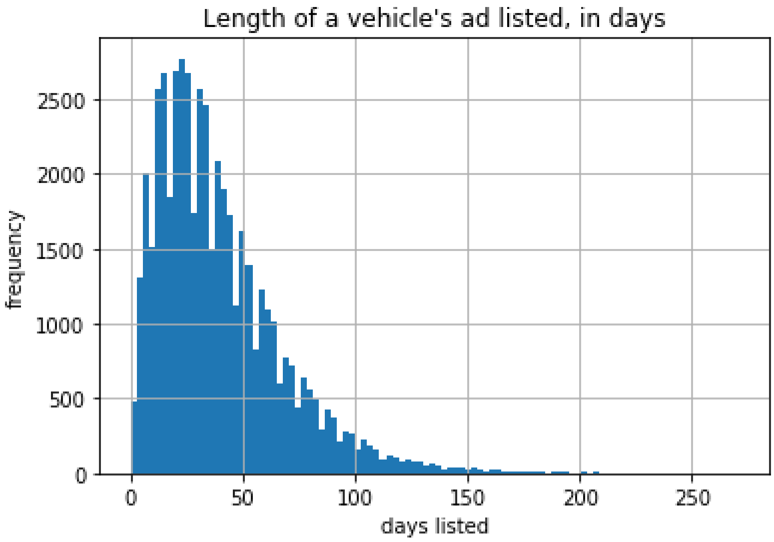


This variable looks good: there are just a few cars with number of cylinders less than 4 (probably some old cars) and also a few with more than 8 cylinders (probably some bigger cars like trucks and pickups). Most vehicles have 4 to 8 cylinders, which sounds reasonable. No visible outliers here.

Days_listed

In [186]:

```
df['days_listed'].hist(bins=100)
plt.title("Length of a vehicle's ad listed, in days")
plt.xlabel('days listed')
plt.ylabel('frequency');
```



In [187]:

```
print('{:0.2f}'.format(df['days_listed'].mean()))
```

39.54

In [188]:

```
print('{:0.2f}'.format(df['days_listed'].median()))
```

33.00

A lifetime of an ad in this dataset ranges from 0 up to around 250 days, the distribution is skewed towards large positive values. A typical ad is placed for around 30-40 days, so around 1 month.

Vehicles with higher values can be considered outliers - they are probably just inadequately priced and that's the reason they couldn't have been sold for a long time. We will identify the upper limit and remove these values in the next section.

Now let's have a look at the ads that were removed too quickly, meaning rows where `days_listed` is 0.

In [189]:

```
len(df[df['days_listed'] == 0])
```

Out[189]:

51

In [190]:

```
df[df['days_listed'] == 0].head()
```

Out[190]:

	price	model_year	model	condition	cylinders	fuel	odometer	transmission	
1232	14995.0	2008	chevrolet silverado 1500	3	8	gas	93300	automatic	
1948	14000.0	1999	ford f250	3	8	diesel	137500	automatic	
2832	4000.0	2004	ram 1500	3	8	gas	250000	automatic	
3900	16750.0	1985	chevrolet corvette	4	8	gas	24540	automatic	hat
4490	5000.0	2007	toyota corolla	2	4	gas	223000	manual	

There are 51 ads of this type and there is no visible pattern or issue with them. However the fact that they were removed the same day as posted makes these observations suspicious. Maybe it was a technical error. We are going to replace them with the median value of a respective model and type group.

In [191]:

```
df.loc[df['days_listed'] == 0, 'days_listed'] = np.nan
df['days_listed'] = df.groupby(['model','type'])['days_listed'].apply(lambda x:
x.fillna(x.median()))
```

In [192]:

```
df['days_listed'].isnull().sum()
```

Out[192]:

0

Removing outliers

In the previous sections we have already got rid of outliers in the `price` variable. From our histogram analysis above both `cylinders` and `condition` variables' distribution look normal, with no issues. We therefore need to remove outliers only for `odometer` and `age_in_years` columns.

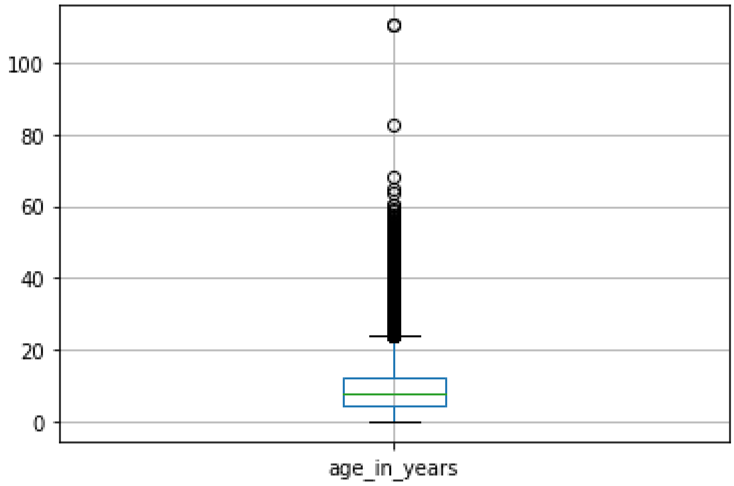
Based on this [article (https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/)](https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/), if we know that the distribution of values in the sample is Gaussian or Gaussian-like, we can use the standard deviation of the sample as a cut-off for identifying outliers. A good statistic for summarizing a non-Gaussian distribution sample of data is the Interquartile Range.

First, let's test whether our two variables have Gaussian distribution. There are [several ways (https://towardsdatascience.com/6-ways-to-test-for-a-normal-distribution-which-one-to-use-9dcf47d8fa93)](https://towardsdatascience.com/6-ways-to-test-for-a-normal-distribution-which-one-to-use-9dcf47d8fa93) to do that, we are going to use box plots and qqplots for that.

Identifying type of a distribution

In [193]:

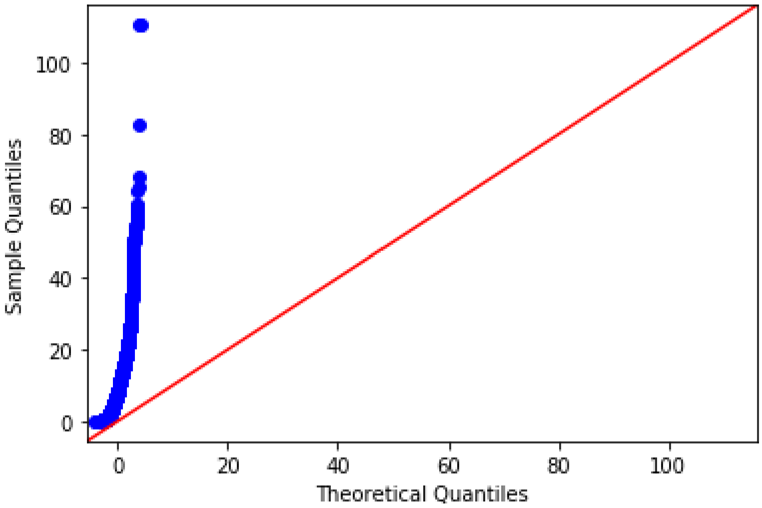
```
df.boxplot('age_in_years');
```



In [194]:

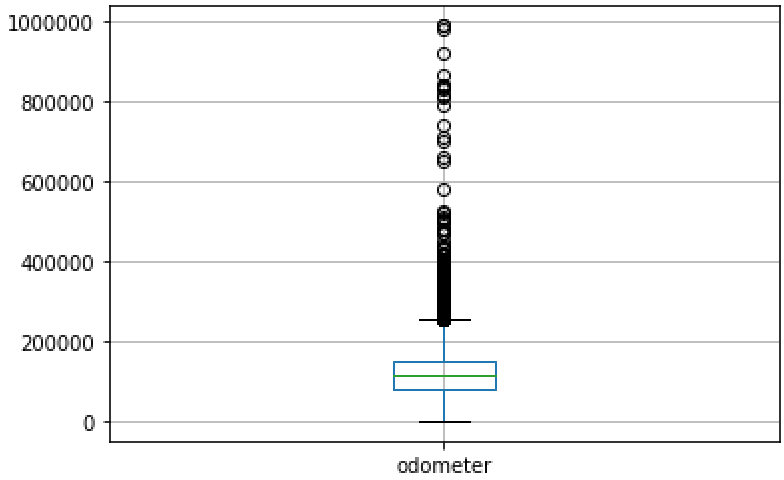
```
import statsmodels.api as sm
from scipy.stats import norm
import pylab

sm.qqplot(df['age_in_years'], line='45')
pylab.show()
```



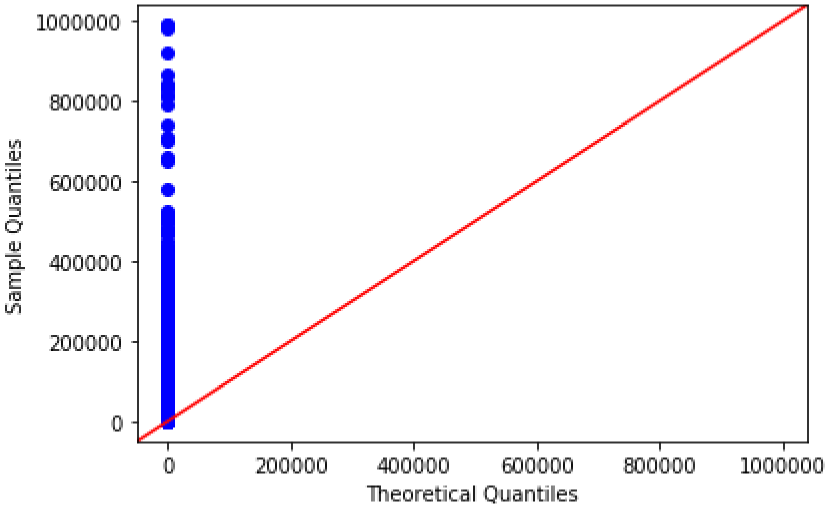
In [195]:

```
df.boxplot('odometer');
```



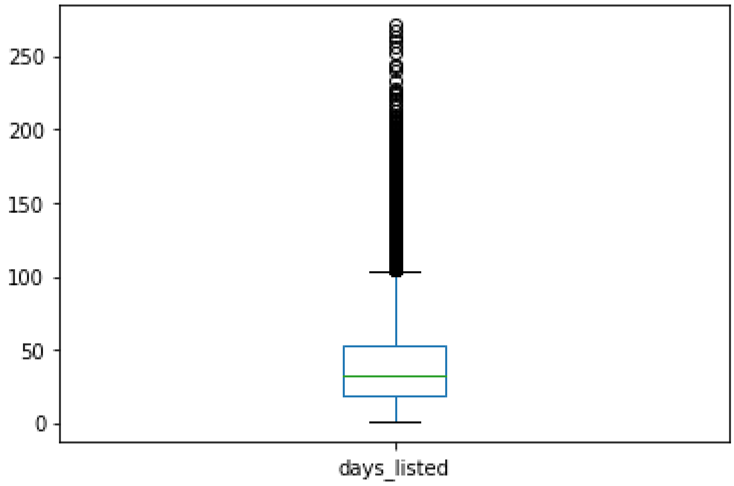
In [196]:

```
sm.qqplot(df['odometer'], line='45')
pylab.show()
```



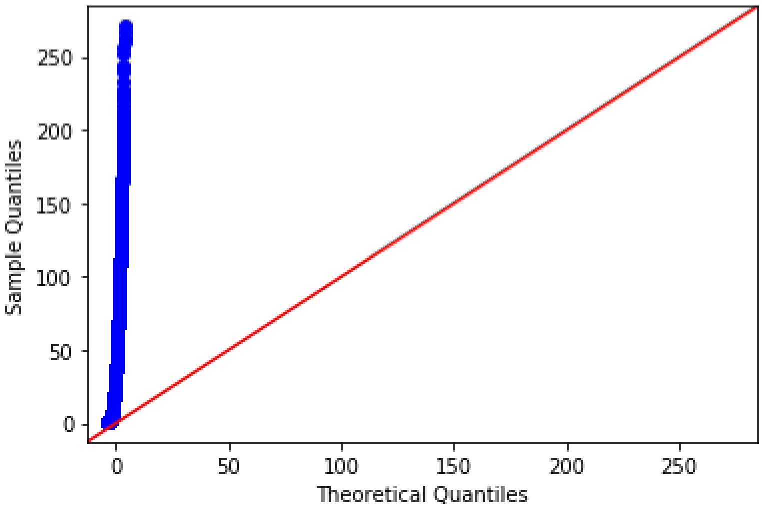
In [197]:

```
df['days_listed'].plot(kind='box');
```



In [198]:

```
sm.qqplot(df['days_listed'], line='45')
pylab.show()
```



According to above box and qq plots we can see that all 3 variables do not have Gaussian distribution, they deviate quite strongly from the theoretical bell curve distribution (red line). Based on that, we are going to use IQR to identify limit values in order to remove outliers.

Using IQR to remove outliers

In [199]:

```
filter = np.zeros(len(df), dtype=bool) + True
for feature in ['age_in_years','odometer','days_listed']:
    q25 = df[feature].quantile(0.25)
    q75 = df[feature].quantile(0.75)
    iqr = q75 - q25
    # calculate the outlier cutoff and upper limit
    cut_off = iqr * 1.5
    upper = q75 + cut_off
    filter[np.where(df[feature]>upper)] = False
```

In [200]:

```
df_filtered = df[filter]
df_filtered.dropna(how='all', inplace=True)
df_filtered.reset_index(drop=True, inplace=True)
```

In [201]:

```
df_filtered.shape
```

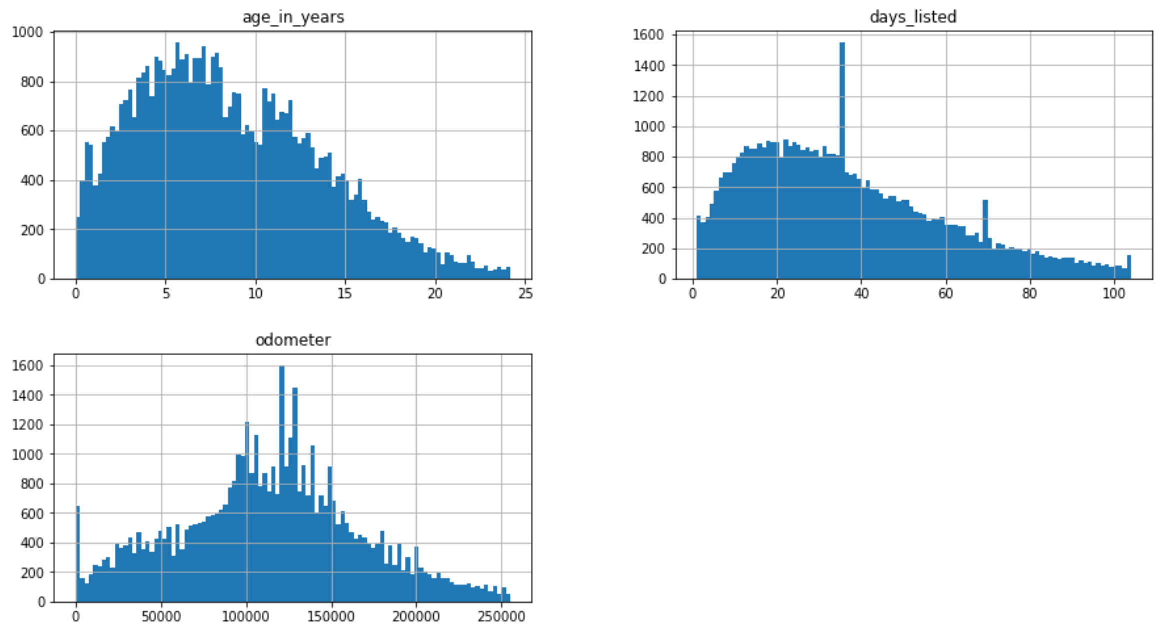
Out[201]:

```
(47494, 19)
```

Filtered data histograms

In [202]:

```
df_filtered[['age_in_years', 'odometer', 'days_listed']].hist(bins=100, figsize=(15,8));
```



We no longer see long tails of large values for these two features, so their distributions look more normal and not as skewed as before.

Type of vehicles

Let's analyze the number of ads and the average price for each type of vehicle.

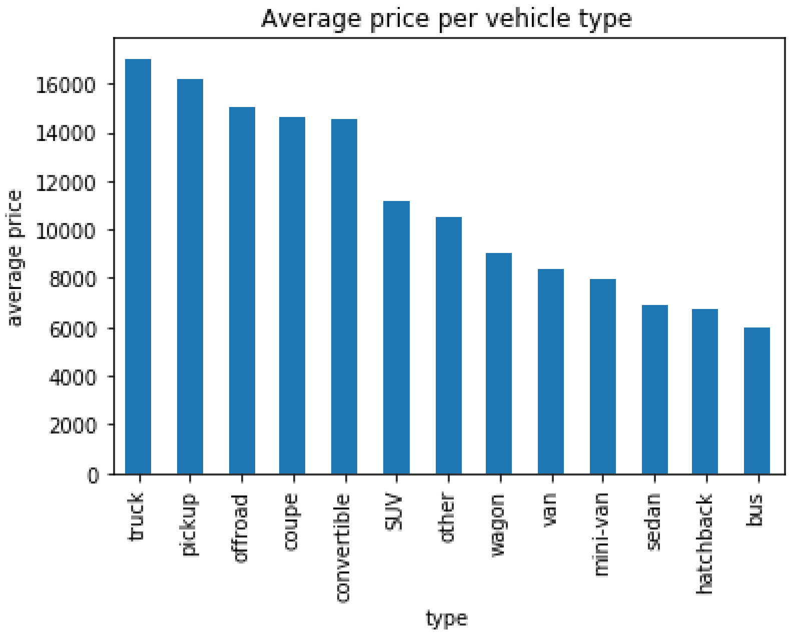
In [203]:

```
type_grouped = df_filtered.pivot_table(index='type', values='price', aggfunc=['c
ount','mean'])
```

First, let's see what type of a vehicle has the largest price, on average.

In [204]:

```
type_grouped['mean'].sort_values(by='price', ascending=False).plot(kind='bar', l
egend=False)
plt.title('Average price per vehicle type')
plt.ylabel('average price');
```

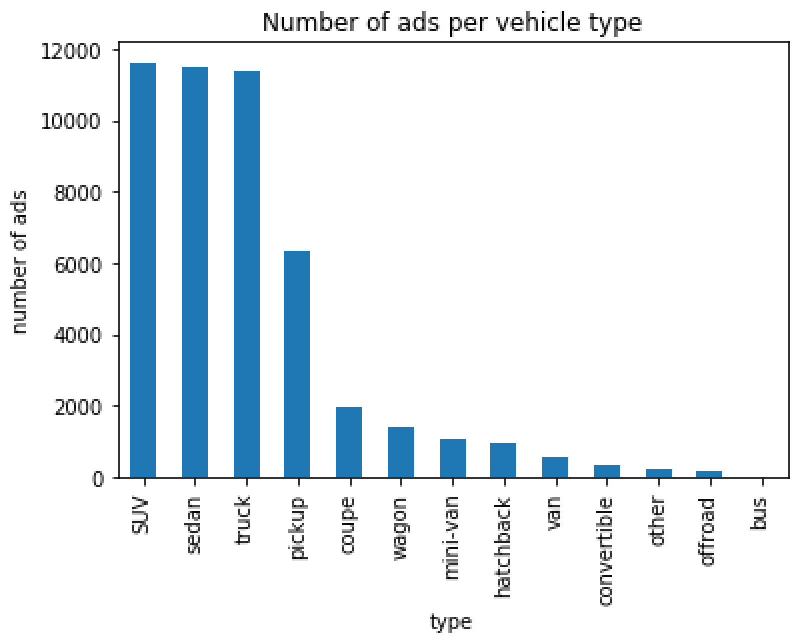


According to the above bar plot, trucks and pickups are the most expensive on average and it seems logical as they are the biggest also.

Next, we'll plot a graph showing the dependence of the number of ads on the vehicle type and identify the two types with the greatest number of ads.

In [205]:

```
type_grouped['count'].sort_values(by='price', ascending=False).plot(kind='bar',
legend=False)
plt.title('Number of ads per vehicle type')
plt.ylabel('number of ads');
```



From the bar plot above we see that 'SUV' and 'Sedan' are the two most popular types of vehicles in this data set. Let's subset our data frame based on these 2 types.

In [206]:

```
popular_types = df_filtered[df_filtered['type'].isin(['SUV', 'sedan'])]
```

Factors influencing price

Condition vs price

First, let's check whether each category has at least 50 ads.

In [207]:

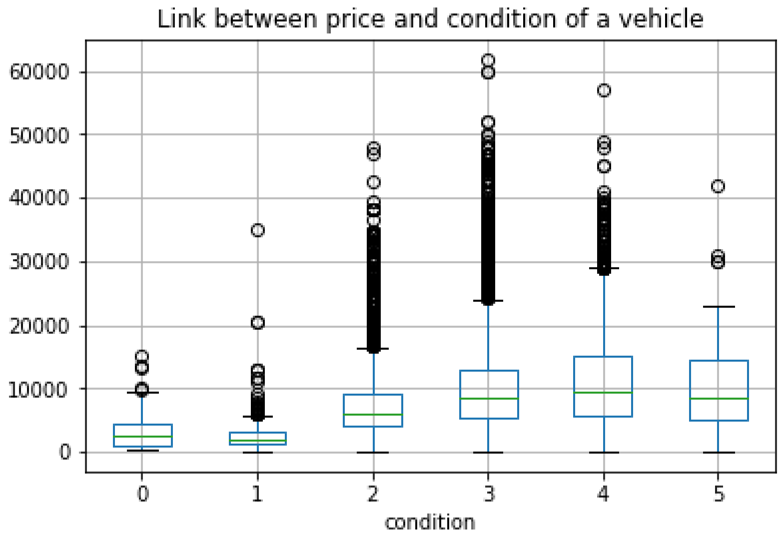
```
popular_types.condition.value_counts()
```

Out[207]:

```
3    12046
2     8129
4     2304
1      522
5       57
0       55
Name: condition, dtype: int64
```

In [208]:

```
popular_types.boxplot(by='condition', column='price')
plt.suptitle('')
plt.title('Link between price and condition of a vehicle');
```



We see quite a clear connection: on average, the better the condition of a vehicle, the higher its price.

Transmission type vs price

First, let's check whether each category has at least 50 ads.

In [209]:

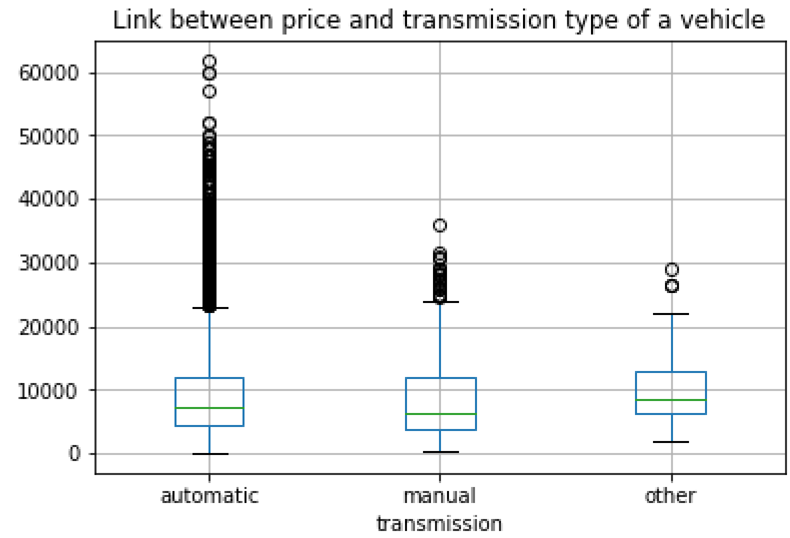
```
popular_types.transmission.value_counts()
```

Out[209]:

```
automatic    21838
manual         992
other          283
Name: transmission, dtype: int64
```

In [210]:

```
popular_types.boxplot(by='transmission', column='price')
plt.suptitle('')
plt.title('Link between price and transmission type of a vehicle');
```



Vehicles with the 'other' type of transmission tend to be slightly more expensive. It is probably explained by the fact these other types give more flexibility and comfort to the driver and more efficient in terms of fuel usage.

Paint color vs price

First, let's check whether each category has at least 50 ads.

In [211]:

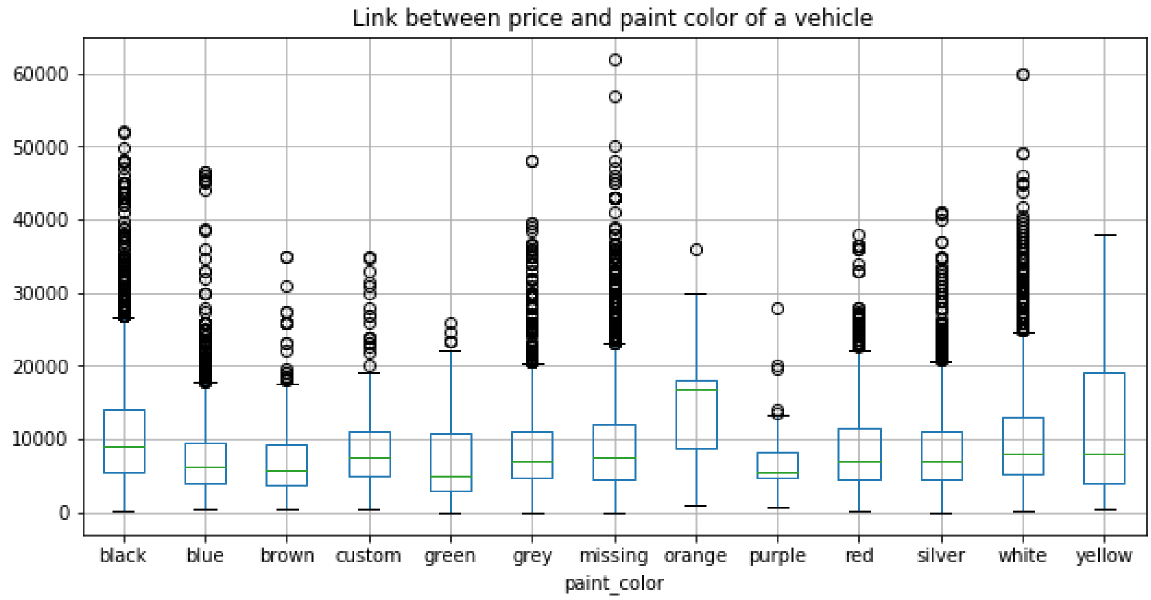
```
popular_types.paint_color.value_counts()
```

Out[211]:

```
missing    4167
black      3748
silver     3435
white      3246
grey       2677
blue       2135
red        1743
green       592
brown       581
custom      577
orange       88
yellow       65
purple       59
Name: paint_color, dtype: int64
```

In [212]:

```
popular_types.boxplot(by='paint_color', column='price', figsize=(10,5))
plt.suptitle('')
plt.title('Link between price and paint color of a vehicle');
```

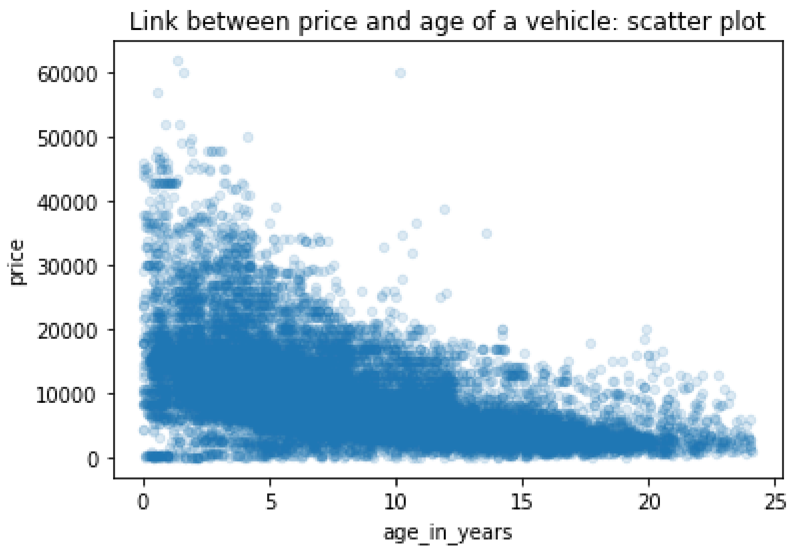


There is no clear connection between price and paint color. It makes sense because paint color depends mostly on customers' preferences and not the price of a car. Interestingly, orange cars seem to be the most expensive but their number is quite low in this data set (only 88 cars). Most cars are black and silver.

Age vs price

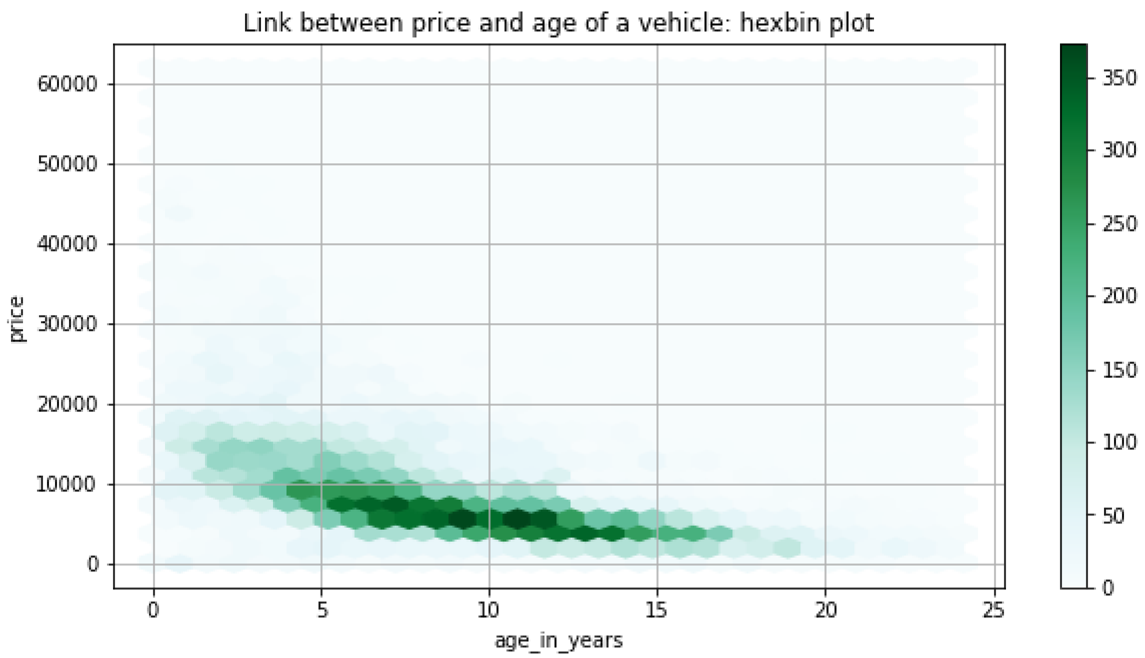
In [213]:

```
popular_types.plot.scatter(x='age_in_years', y='price', alpha=.15)
plt.title('Link between price and age of a vehicle: scatter plot');
```



In [214]:

```
popular_types.plot(x='age_in_years', y='price', kind='hexbin', gridsize=30, figs
ize=(10, 5), sharex=False, grid=True)
plt.title('Link between price and age of a vehicle: hexbin plot');
```

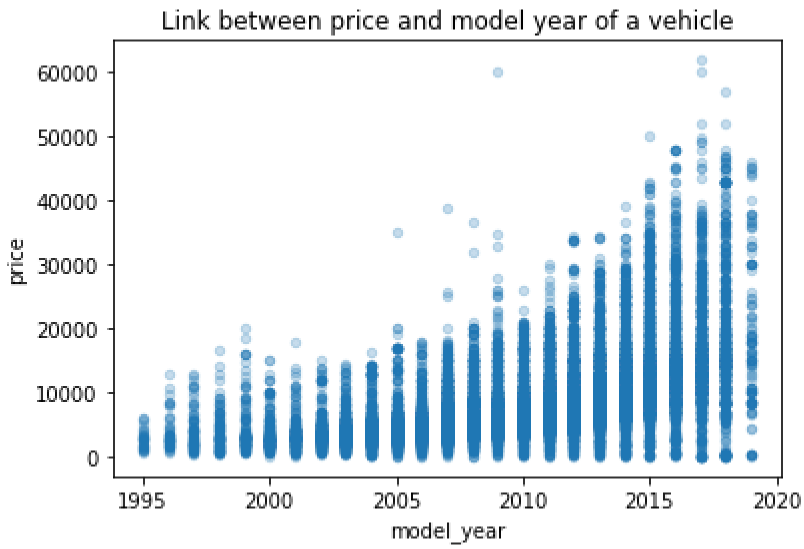


We see that, on average, the older a vehicle is, the lower its price, which is quite reasonable.

Let's see if this conclusion holds for the `model_year` feature.

In [215]:

```
popular_types.plot.scatter(x='model_year', y='price', alpha=.25)
plt.title('Link between price and model year of a vehicle');
```

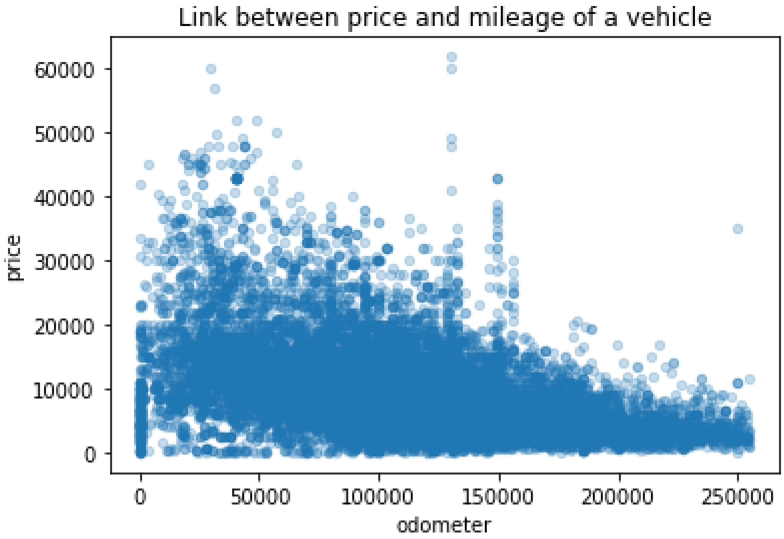


This tendency is also confirmed by the `model_year` variable. The data reflects real life situation: the more recent a car's model, the higher its price.

Mileage vs price

In [216]:

```
popular_types.plot.scatter(x='odometer', y='price', alpha=.25)
plt.title('Link between price and mileage of a vehicle');
```



Based on the above scatter plot, as the `odometer` increases, `price` of a vehicle tends to decrease. It makes sense as the higher the mileage, the worse, on average, the condition of a car, hence lower the price.

Conclusion

In this report we have analyzed different features of various types of vehicles in order to determine whether age, mileage, condition, transmission type, and color influence the price of a vehicle.

First of all, we have familiarized ourselves with the data by performing the descriptive statistics. Then, we examined our target variable `price` and found a few artifacts in the data set that we have corrected: rows with abnormally high prices per type and brand group, observations with `price =` 1 and `price =` 123456.

Next step was to deal with missing values:

- Missing values in the `is_4wd` column were filled with 0 for those vehicles that do not have 4 weels;
- Missing `odometer` values were filled with the median value of a respective group based on the model and type of a vehicle. We assumed that cars with the same model (e.g. cadillac escalade) but different types (e.g. SUV and pickup) have different odometer values;
- Missing `model_year` values were filled based on the median number of years in exploitation of a respective group based on the model and type of a vehicle;
- Missing `cylinders` values were filled with the median value of a respective group based on the model and type of a vehicle. We assumed that cars with the same model (e.g. cadillac escalade) but different types (e.g. SUV and pickup) have different number of cylinders because their engines were designed for different purposes;
- Missing `paint_color` values were filled with a string 'missing' as it mostly depends on customer preferences and we don't have any additional information to fill in this column.

Then we performed a few auxiliary calculations:

- We extracted day of the week, month, and year when an ad was placed;
- We calculated a vehicle's age (in years) when the ad was placed;
- We calculated a vehicle's average mileage per year;
- In the condition column, we replaced string values with a numeric scale to make further analysis easier.

Lastly, we have performed exploratory data analysis:

1. We have plotted histograms for the main features in order to find outliers. `price` , `condition` and `cylinders` columns had no visible outliers while `age_in_years` , `odometer` and `days_listed` all had long tails of large values that we had to be filtered out as they skewed our data. First, we tested whether these variables had normal (Gaussian) distribution. All of them deviated strongly from the theoretical bell curve distribution. Based on that, we have decided to use interquartile range to identify limit values for removal of outliers. After filtering, histograms looked more normal and less skewed as before;
2. Lastly, we have analyzed what factors influenced the price of a vehicle the most. We have conducted this analysis for the 2 most popular types of vehicles in terms of the number of ads (SUV and sedan). Based on our analysis, some of the predicted tendencies were correct while others were not:
    - Correct, on average, the older a vehicle, the lower the price;
    - Correct, on average, the higher the mileage, the lower the price;
    - Correct, on average, the better the condition of a vehicle, the higher the price;
    - Incorrect, vehicles with the 'other' type of transmission tend to be more expensive. It is probably explained by the fact these other types give more flexibility and comfort to the driver and more efficient in terms of fuel usage.
    - Incorrect, there is no clear connection between price and paint color, it depends mostly on customers' preferences. Interestingly, orange cars seem to be the most expensive, although their number is quite low in this data set (only 88 cars). Most cars are black and silver but they are not the most expensive.