car_ad_EDA

December 31, 2020

1 Research on car sales ads

1.0.1 Author: Xia Cui

1.0.2 Introduction

The report presents the exploratory analysis of a dataset of 51525 free advertisements for vehicles on Crankshaft list, posted from 2018-05-01 to 2019-04-19.

At the data preprocessing stage, we will address the issues of missing values and wrong data types, and also add columns where it fits in order to maximize the information it can present.

The analysis that follows first of all examines the pattern of the advertisements in a range of factors such as price, model year, cylinders, and mileage. Then we will look at what are the characteristics of cars that tend to be gone quickly, as well as of those that stay on the list longer than most. Finally, we will take a closer look at car prices, and investigate which factors influence the price of a vehicle, using the two most popular vehicle types as examples.

1.0.3 Table of Content

Part 1: data importing

Part 2: data preprocessing

Part 3: adding new columns

Part 4: exploratory data analysis

Part 5: conclusion

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display
import seaborn as sns
%matplotlib inline
```

1.0.4 Part 1: Data importing

[2]:
```python
vehicles = pd.read_csv('https://code.s3.yandex.net/datasets/vehicles_us.csv')
vehicles.head()
```

[2]:
```
   price  model_year           model  condition  cylinders fuel  odometer  \
0   9400      2011.0          bmw x5       good        6.0  gas  145000.0
1  25500         NaN      ford f-150       good        6.0  gas   88705.0
2   5500      2013.0  hyundai sonata   like new        4.0  gas  110000.0
3   1500      2003.0      ford f-150       fair        8.0  gas       NaN
4  14900      2017.0    chrysler 200  excellent        4.0  gas   80903.0

  transmission    type paint_color  is_4wd date_posted  days_listed
0    automatic     SUV         NaN     1.0  2018-06-23           19
1    automatic  pickup       white     1.0  2018-10-19           50
2    automatic   sedan         red     NaN  2019-02-07           79
3    automatic  pickup         NaN     NaN  2019-03-22            9
4    automatic   sedan       black     NaN  2019-04-02           28
```

[3]:
```python
vehicles.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
```

[4]:
```python
vehicles.shape
```

[4]:
```
(51525, 13)
```

Let's also take a look at the summary data of the numeric variables.

[5]:
```
vehicles.describe()
```

[5]:
```
               price   model_year     cylinders       odometer     is_4wd  \
count   51525.000000  47906.000000  46265.000000   43633.000000  25572.0
mean    12132.464920   2009.750470      6.125235  115553.461738      1.0
std     10040.803015      6.282065      1.660360   65094.611341      0.0
min         1.000000   1908.000000      3.000000       0.000000      1.0
25%      5000.000000   2006.000000      4.000000   70000.000000      1.0
50%      9000.000000   2011.000000      6.000000  113000.000000      1.0
75%     16839.000000   2014.000000      8.000000  155000.000000      1.0
max    375000.000000   2019.000000     12.000000  990000.000000      1.0

       days_listed
count  51525.00000
mean      39.55476
std       28.20427
min        0.00000
25%       19.00000
50%       33.00000
75%       53.00000
max      271.00000
```

1.0.5 Part 1 conclusion

The dataset contains 51525 rows and 13 columns. a quick look at its first 5 rows and the general information reveals a number of issues that need to be addressed in the data preprocessing stage next. These include missing values, data type (e.g. date_posted), and some suspicious values (e.g. the 1 dollar car price). There are also some extremely large values in, for example, car price, and odometers. These could potentially be outliers that might skew the data. We will further examine these in the next section.

1.0.6 Part 2: Data preprocessing

Addressing missing values

[6]:
```
vehicles.isnull().sum(axis = 0)
```

[6]:
```
price               0
model_year       3619
model               0
condition           0
cylinders        5260
fuel                0
odometer         7892
transmission        0
type                0
paint_color      9267
is_4wd          25953
```

```
date_posted         0
days_listed         0
dtype: int64
```

We can see that, out of the 13 columns, 5 columns have missing values. To have a better idea of the impact of these, let's calculate the percentage of missing values in the dataset.

[7]:
```
100*vehicles.isnull().sum(axis = 0) / vehicles.shape[0]
```

[7]:
```
price            0.000000
model_year       7.023775
model            0.000000
condition        0.000000
cylinders       10.208637
fuel             0.000000
odometer        15.316836
transmission     0.000000
type             0.000000
paint_color     17.985444
is_4wd          50.369723
date_posted      0.000000
days_listed      0.000000
dtype: float64
```

Missing values for 'is_4wd'

It seems that none of the missing values can be considered trivial in the five columns,especially for the 'is_4wd' variable. However, a closer look at the data description earlier shows that all the values in this column is '1', indicating the car is 4_wd. It is reasonable to believe that the missing values should have been '0', indicating that the car is not 4_wd. So the missing values for this variable is relatively easier to solve. We can simply replace the NaN with '0'.

[8]:
```
vehicles['is_4wd'] = vehicles['is_4wd'].fillna(0)
vehicles['is_4wd'].value_counts()
```

[8]:
```
0.0    25953
1.0    25572
Name: is_4wd, dtype: int64
```

Before we decide what to do with the missing values in the other four columns, let's examine them first.

Missing values for 'model_year'

[9]:
```
vehicles[vehicles['model_year'].isna()].head()
```

[9]:
```
    price  model_year            model  condition  cylinders fuel  odometer  \
1   25500         NaN       ford f-150       good        6.0  gas   88705.0
20   6990         NaN  chevrolet tahoe  excellent        8.0  gas  147485.0
```

```
65  12800          NaN       ford f-150  excellent        6.0  gas  108500.0
69   7800          NaN       ford f-150   like new        8.0  gas   97510.0
72   3650          NaN   subaru impreza  excellent        NaN  gas   74000.0

   transmission    type paint_color  is_4wd date_posted  days_listed
1     automatic  pickup       white     1.0  2018-10-19           50
20    automatic     SUV      silver     1.0  2018-08-05           28
65    automatic  pickup       white     0.0  2018-09-23           15
69    automatic   truck       white     1.0  2019-02-20           39
72    automatic   sedan        blue     1.0  2018-08-07           60
```

Is model_year dependant on model? let's have a look.

[19]:
```python
vehicles.groupby('model')['model_year'].value_counts()
```

[19]:
```
model              model_year
acura tl           2005.0        31
                   2007.0        30
                   2008.0        28
                   2012.0        27
                   2006.0        24
                                 ..
volkswagen passat  2009.0         3
                   1995.0         1
                   1999.0         1
                   2002.0         1
                   2018.0         1
Name: model_year, Length: 2226, dtype: int64
```

It might be tricky to replace the missing values in 'model_year'. Each model category has several different model years and is not really helpful in filling the missing values.

While it is reasonable to believe that, quite often, the more odometers a car has, the older it could be, to fill in the model year based on its odometer would be over generalizing. Moreover, model_year potentially could be a key parameter in its sale price, therefore we need to be especially careful how to replace these, or whether to replace these at all. For now, let's keep the column as it is and decide what to do with it later.

Missing value for 'cylinders'

[20]:
```python
vehicles[vehicles['cylinders'].isna()].head()
```

[20]:
```
    price  model_year              model  condition  cylinders fuel  odometer  \
9    9200      2008.0        honda pilot  excellent        NaN  gas  147191.0
36  10499      2013.0       chrysler 300       good        NaN  gas   88042.0
37   7500      2005.0      toyota tacoma       good        NaN  gas  160000.0
59   5200      2006.0  toyota highlander       good        NaN  gas  186000.0
63  30000      1966.0       ford mustang  excellent        NaN  gas   51000.0
```

```
   transmission         type paint_color  is_4wd date_posted  days_listed
9     automatic          SUV        blue     1.0  2019-02-15           17
36    automatic        sedan         NaN     0.0  2018-05-05           22
37    automatic       pickup         NaN     0.0  2018-07-22           44
59    automatic          SUV       green     0.0  2018-12-20            2
63       manual  convertible         red     0.0  2019-01-23           17
```

It is possible that the number of cylinders is dependant on the car model. Let's check using the first two in dataset above.

[25]:
```
vehicles[vehicles['model']=='honda pilot']['cylinders'].value_counts()
```

[25]:
```
6.0    269
4.0      2
8.0      2
5.0      1
Name: cylinders, dtype: int64
```

[26]:
```
vehicles[vehicles['model']=='chrysler 300']['cylinders'].value_counts()
```

[26]:
```
6.0    216
8.0     58
4.0      5
Name: cylinders, dtype: int64
```

Our assumption seems to hold some truth. For each car model, there is a dominant cylinder number. Although it would not be entirely accurate, it's relatively safe to use the dominant cylinder number of each model to replace the missing values in the cylinder column.

[27]:
```
vehicles["cylinders"] = vehicles.groupby("model")[
    "cylinders"].transform(lambda x: x.fillna(x.mode()[0]))
```

Let's check if the null values have been filled.

[28]:
```
vehicles["cylinders"].isna().sum()
```

[28]:
```
0
```

It's all done for cylinders. Now let's continue with 'odometer'.

Missing values for 'odometer'

[335]:
```
vehicles[vehicles['odometer'].isna()].head()
```

[335]:
```
    price  model_year           model  condition  cylinders fuel  odometer  \
3    1500      2003.0      ford f-150       fair        8.0  gas       NaN
15  17990      2013.0        ram 1500  excellent        8.0  gas       NaN
23   7500      2004.0   jeep wrangler       good        6.0  gas       NaN
24   3950      2009.0    chrysler 200  excellent        4.0  gas       NaN
```

```
25   11499      2017.0  chevrolet malibu   like new        4.0  gas       NaN

   transmission    type paint_color  is_4wd date_posted  days_listed
3     automatic  pickup         NaN     0.0  2019-03-22            9
15    automatic  pickup         red     1.0  2018-05-15          111
23    automatic     SUV         red     1.0  2018-05-17           39
24    automatic   sedan         red     0.0  2018-06-11           40
25    automatic   sedan         NaN     0.0  2018-07-26           43
```

It is possible to fill up the missing values in odometer using the median values based on 'model_year'and 'condition'. However, we know that there are missing values in 'model_year' as well. So we need to check how many have missing values in both columns.

[29]:
```python
vehicles[(vehicles['odometer'].isna()) & (vehicles['model_year'].isna())]
```

[29]:
```
       price  model_year                       model  condition  cylinders  \
159    23300         NaN  nissan frontier crew cab sv       good        6.0
260    14975         NaN              toyota 4runner       good        6.0
370     4700         NaN                    kia soul       good        4.0
586    26000         NaN                 toyota rav4   like new        4.0
659     8400         NaN           volkswagen jetta       good        4.0
…      …          …                            …         …          …
51195  21999         NaN                    ram 2500       good        6.0
51222   1000         NaN                    acura tl       good        6.0
51257   6500         NaN              toyota corolla       good        4.0
51295   3850         NaN             hyundai elantra  excellent        4.0
51399   4400         NaN                 kia sorento  excellent        6.0

         fuel  odometer transmission    type paint_color  is_4wd date_posted  \
159       gas       NaN        other  pickup        grey     1.0  2018-07-24
260       gas       NaN    automatic     SUV      silver     0.0  2018-05-13
370       gas       NaN       manual   sedan       white     0.0  2019-01-14
586       gas       NaN    automatic     SUV         NaN     0.0  2018-08-09
659    diesel       NaN       manual   wagon         NaN     0.0  2018-10-22
…       …       …          …       …       …         …       …
51195  diesel       NaN    automatic   truck       white     1.0  2018-05-10
51222     gas       NaN    automatic   sedan        grey     0.0  2018-12-09
51257     gas       NaN    automatic   sedan       white     0.0  2018-10-16
51295     gas       NaN    automatic   sedan      silver     0.0  2019-03-16
51399     gas       NaN    automatic     SUV      silver     0.0  2018-08-21

       days_listed
159             73
260             57
370             50
586             29
659             37
```

```
…              …
51195          35
51222          23
51257          75
51295          83
51399          23

[549 rows x 13 columns]
```

There are 549 rows that have missing values in both 'model_year' and 'odometer'. For these 549 rows, we can use the median odometers by condition only to fill in the missing values. For the rest, we can use both 'model_year' and 'condition' to do so.

[31]:
```
vehicles.loc[vehicles['model_year'].notna(), "odometer"] = vehicles[
    vehicles['model_year'].notna()].groupby(["model_year", "condition"])[
    "odometer"].transform(lambda x: x.fillna(x.median()))
```

[32]:
```
vehicles.loc[vehicles['model_year'].isna(), "odometer"] = vehicles[
    vehicles['model_year'].isna()].groupby("condition")[
    "odometer"].transform(lambda x: x.fillna(x.median()))
```

Now let's check how many missing values are left in the 'odometer' column.

[34]:
```
vehicles['odometer'].isna().sum()
```

[34]: 7

Let's found out who these are.

[35]:
```
vehicles[vehicles['odometer'].isna()]
```

[35]:
```
       price  model_year               model  condition  cylinders fuel  \
21421   4500      1974.0  chevrolet corvette       fair        8.0  gas
28009  65000      1960.0  chevrolet corvette   like new        8.0  gas
31806   1700      1996.0        ford mustang    salvage        6.0  gas
33257   4500      1963.0    chevrolet impala       fair        6.0  gas
33907  12995      1908.0   cadillac escalade  excellent        8.0  gas
45694  18000      1929.0          ford f-150       good        8.0  gas
46911  22300      2003.0  chevrolet corvette        new        8.0  gas

       odometer transmission         type paint_color  is_4wd date_posted  \
21421       NaN    automatic        sedan         red     0.0  2018-12-15
28009       NaN       manual        coupe         NaN     0.0  2018-11-03
31806       NaN       manual  convertible       white     0.0  2019-03-31
33257       NaN    automatic        sedan         NaN     0.0  2019-03-17
33907       NaN    automatic          SUV       white     0.0  2018-06-24
45694       NaN       manual        other      silver     0.0  2018-11-18
46911       NaN       manual  convertible       black     0.0  2018-11-08
```

```
       days_listed
21421           18
28009           41
31806           46
33257           38
33907           25
45694           59
46911           23
```

These cars have both model year and conditions, but why aren't their values filled? Let's take a look at the first year condition combo.

[36]:
```
vehicles[(vehicles['model_year'] == 1974.0) & (vehicles['model'] == 'chevrolet␣
↪corvette')]
```

[36]:
```
       price  model_year               model condition  cylinders fuel  \
4708   10500      1974.0  chevrolet corvette      good        8.0  gas
16723   4950      1974.0  chevrolet corvette      good        8.0  gas
21421   4500      1974.0  chevrolet corvette      fair        8.0  gas

       odometer transmission   type paint_color  is_4wd date_posted  \
4708     4133.0    automatic  coupe         red     0.0  2018-09-16
16723   29000.0    automatic  coupe        blue     0.0  2018-07-21
21421       NaN    automatic  sedan         red     0.0  2018-12-15

       days_listed
4708            74
16723          103
21421           18
```

The result above shows that the reason the missing odometer is not filled is because the other two are both in good conditions and there is no median value for the fair condition car from the same year.

There are only 7 left. We can leave these for now.

Missing values in 'paint_color'

[37]:
```
vehicles[vehicles['paint_color'].isna()].head()
```

[37]:
```
    price  model_year         model  condition  cylinders fuel  odometer  \
0    9400      2011.0        bmw x5       good        6.0  gas  145000.0
3    1500      2003.0    ford f-150       fair        8.0  gas  193850.0
8   11500      2012.0   kia sorento  excellent        4.0  gas  104174.0
12  18990      2012.0      ram 1500  excellent        8.0  gas  140742.0
21   5250      2007.0  toyota rav4        good        6.0  gas  154000.0

   transmission    type paint_color  is_4wd date_posted  days_listed
```

```
0     automatic     SUV          NaN     1.0   2018-06-23           19
3     automatic   pickup         NaN     0.0   2019-03-22            9
8     automatic     SUV          NaN     1.0   2018-07-16           19
12    automatic   pickup         NaN     1.0   2019-04-02           37
21    automatic     SUV          NaN     0.0   2018-08-22            8
```

Let's first have a look at how the colors are distributed.

[38]:
```python
vehicles['paint_color'].value_counts()
```

[38]:
```
white     10029
black      7692
silver     6244
grey       5037
blue       4475
red        4421
green      1396
brown      1223
custom     1153
yellow      255
orange      231
purple      102
Name: paint_color, dtype: int64
```

We can either fill the missing values with the dominant car color, 'white', or with the dominate car color of same model. Let's take the second approach so white wouldn't become even more dominant.

[39]:
```python
vehicles["paint_color"] = vehicles.groupby("model")[
    "paint_color"].transform(lambda x: x.fillna(x.mode()[0]))
```

[40]:
```python
vehicles['paint_color'].isna().sum()
```

[40]:
```
0
```

Now we've filled up all the NA values that we could. Let's check what's there left.

[41]:
```python
vehicles.isnull().sum(axis = 0)
```

[41]:
```
price              0
model_year      3619
model              0
condition          0
cylinders          0
fuel               0
odometer           7
transmission       0
type               0
```

```
paint_color        0
is_4wd             0
date_posted        0
days_listed        0
dtype: int64
```

While we are here, let's also check if there are any duplicated rows.

[42]:
```
vehicles.duplicated().sum()
```

[42]:
```
0
```

No duplicated row!!

All the missing values left are the 'model_year' missing values, and those 'odometer' missing values where the 'model_year' is also missing. 3619 rows repesent over 7% of the total dataset and is not insignificant. However, 'model_year' is a key parameter for car price and keeping these in the dataset or replace them with any other values will potentially skew the data and impact the results. let's drop these and save the dataset with a new name.

Drop NA values

[43]:
```
vehicles_new = vehicles.dropna().reset_index(drop = True)
vehicles_new.shape
```

[43]:
```
(47899, 13)
```

The new dataset has 47899 rows and 13 columns. Let's double check to see if we missed anything.

[45]:
```
vehicles_new.isnull().sum()
```

[45]:
```
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
dtype: int64
```

Now there are no NA values in the dataset. We can move on to change the data types and add new columns. First, let's have another look at the datatypes. **Changing data types**

[46]:
```
vehicles_new.dtypes
```

[46]:
```
price              int64
model_year       float64
model             object
condition         object
cylinders        float64
fuel              object
odometer         float64
transmission      object
type              object
paint_color       object
is_4wd           float64
date_posted       object
days_listed        int64
dtype: object
```

'date_posted' should be datetime data type in order to calculate the day of the week, month and year later on.

cylinders, and is_4wd should be integer type.

'model_year' is float. We can leave it be because later the extracted year of the ad posting date will be integers and we can still calcuate the age of the cars using those two columns.

[47]:
```
vehicles_new['model_year'] = vehicles_new['model_year'].astype(int)
vehicles_new.head()
```

[47]:
```
   price  model_year           model  condition  cylinders fuel  odometer  \
0   9400        2011          bmw x5       good        6.0  gas  145000.0
1   5500        2013  hyundai sonata   like new        4.0  gas  110000.0
2   1500        2003      ford f-150       fair        8.0  gas  193850.0
3  14900        2017    chrysler 200  excellent        4.0  gas   80903.0
4  14990        2014    chrysler 300  excellent        6.0  gas   57954.0

  transmission    type paint_color  is_4wd date_posted  days_listed
0    automatic     SUV       black     1.0  2018-06-23           19
1    automatic   sedan         red     0.0  2019-02-07           79
2    automatic  pickup       white     0.0  2019-03-22            9
3    automatic   sedan       black     0.0  2019-04-02           28
4    automatic   sedan       black     1.0  2018-06-20           15
```

[48]:
```
vehicles_new['date_posted'] = pd.to_datetime(vehicles_new['date_posted'],␣
↪format = '%Y-%m-%d')
```

[49]:
```
vehicles_new['cylinders'] = vehicles_new['cylinders'].astype(int)
```

[50]:
```
vehicles_new['is_4wd'] = vehicles_new['is_4wd'].astype(int)
```

[51]:

```
vehicles_new.dtypes
```

[51]:

```
price                       int64
model_year                  int64
model                      object
condition                  object
cylinders                   int64
fuel                       object
odometer                  float64
transmission               object
type                       object
paint_color                object
is_4wd                      int64
date_posted        datetime64[ns]
days_listed                 int64
dtype: object
```

Now we have the data in the types exactly what we want. Let's move on to add some new columns.

1.0.7 Part 3 : Make calculations and add them to the table

Add a column of the day of the week when the ads were posted

[52]:

```
vehicles_new['day_of_week_posted'] = vehicles_new['date_posted'].dt.weekday
```

Add column of the month the ads were placed

[53]:

```
vehicles_new['month_posted'] = pd.DatetimeIndex(vehicles_new['date_posted']).
 ↪month
```

Add a column of the year the ads were posted

[54]:

```
vehicles_new['year_posted'] = pd.DatetimeIndex(vehicles_new['date_posted']).year
```

Add a column of the vehicle's age (in years) when the ads were placed

[55]:

```
vehicles_new['vehicle_age'] = vehicles_new['year_posted] -␣
 ↪vehicles_new['model_year']
```

Add a column of the vehicle's average mileage per year

[56]:

```
vehicles_new['mileage_per_year'] = vehicles_new['odometer'] /␣
 ↪vehicles_new['vehicle_age']
```

In the condition column, replace string values with a numeric scale

[57]:

```
vehicles_new['condition'] = vehicles_new[
    'condition'].replace(['salvage', 'fair', 'good', 'excellent', 'like new',␣
 ↪'new'],
```

```
                    [0, 1, 2, 3, 4, 5])
vehicles_new.head()
```

[57]:

```
   price  model_year           model  condition  cylinders fuel  odometer  \
0   9400        2011          bmw x5          2          6  gas  145000.0
1   5500        2013  hyundai sonata          4          4  gas  110000.0
2   1500        2003      ford f-150          1          8  gas  193850.0
3  14900        2017    chrysler 200          3          4  gas   80903.0
4  14990        2014    chrysler 300          3          6  gas   57954.0

  transmission    type paint_color  is_4wd date_posted  days_listed  \
0    automatic     SUV       black       1  2018-06-23           19
1    automatic   sedan         red       0  2019-02-07           79
2    automatic  pickup       white       0  2019-03-22            9
3    automatic   sedan       black       0  2019-04-02           28
4    automatic   sedan       black       1  2018-06-20           15

   day_of_week_posted  month_posted  year_posted  vehicle_age  \
0                   5             6         2018            7
1                   3             2         2019            6
2                   4             3         2019           16
3                   1             4         2019            2
4                   2             6         2018            4

   mileage_per_year
0      20714.285714
1      18333.333333
2      12115.625000
3      40451.500000
4      14488.500000
```

The mileage_per_year column has many digits after the decimal point. Let's round these to the nearest whole number up.

[59]:

```
vehicles_new['mileage_per_year'] = vehicles_new['mileage_per_year'].apply(np.
 ↪ceil)
```

1.0.8 Part 3 Conclusion

Now the dataset is more telling, including which weekday, month, and year the advertisements were placed, as well as vehicle's age and their mileage per year. We can easily draw on these addition columns in the exploratory analysis next.

1.0.9 Part 4: Exploratory data analysis

The exploratory analysis will first examine the following parameters: price, vehicle's age when the ad was placed, mileage, number of cylinders, and condition. A histogram or bar graph will be plotted for each of these parameters.

Such informaton, together with the descriptive statistics, will allow us to identify outliers in the data, and understand how they might affect the form and readability of the graphs. Then we will decide what outliers to be filtered out of the data, and plot another set of graphs to show the data distribution along the above mentioned parameters.

Then, we will have a look at how many days advertisements were displayed. The histogram and descriptive statistics of this parameter will allow us to understand the typical lifetime of an ad.

Last, we'll analyze the number of ads and the average price for each type of vehicle. The two types with the greatest number of ads will then be further examined to find out which factors impact on their car prices most.

4.1: Distribution of car price, age, mileage, cylinders, and condition

4.2: Removing outliers

4.3: Plotting new graphs

4.4: Studying the lifespan of advertisements

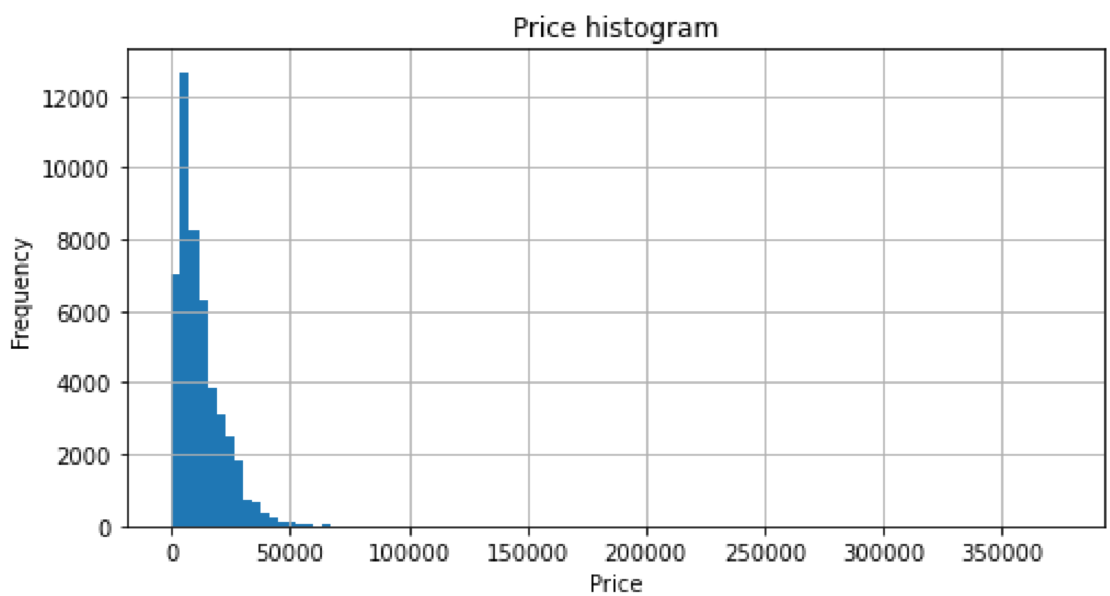
4.5: Number of ads and mean prices by car type

4.6: Factors in car prices**

1.0.10 4.1 Distribution of car price, age, mileage, cylinders, and condition

Price distribution

[108]:
```
vehicles_new.hist('price', bins = 100, grid = True, figsize = (8, 4))
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Price histogram')
plt.show()
```



[61]:
```
vehicles_new['price'].describe()
```

[61]:
```
count     47899.000000
mean      12159.549281
std       10079.922175
min           1.000000
25%        5000.000000
50%        9000.000000
75%       16900.000000
max      375000.000000
Name: price, dtype: float64
```

Both the histogram and statistics data of the 'price' column shows that there are outliers on both sides of the distribution. It's clear that 75% of the cars are priced 16900 and under, and the number of cars priced over 75000 is approching zero. However, it's not clear what the distribution is like on the cheaper end. To find out, let's have a look at the 10 cheapest cars.

[62]:
```
vehicles_new.sort_values('price').head(10)
```

[62]:
```
       price  model_year                  model  condition  cylinders fuel  \
10951      1        2015               ram 1500          3         10  gas
8804       1        2015             ford edge           3          6  gas
11438      1        2014     chevrolet silverado          4          8  gas
38692      1        2002       volkswagen jetta          2          4  gas
8702       1        2014             gmc sierra          3          8  gas
8664       1        2014       chevrolet camaro          3          6  gas
8663       1        2016          jeep wrangler          3          6  gas
46210      1        2007  chevrolet trailblazer          3          8  gas
8662       1        2012           ford mustang          3         10  gas
8661       1        2017               ram 3500          3         10  gas

       odometer transmission   type paint_color  is_4wd date_posted  \
10951   59980.0        other  truck      silver       1  2018-07-25
8804    24897.0    automatic    SUV       black       1  2018-06-21
11438      42.0    automatic  truck       black       1  2018-06-28
38692  167062.0    automatic  sedan        blue       0  2019-02-08
8702   147470.0    automatic  truck       brown       0  2018-12-29
8664    51550.0       manual  coupe       black       1  2018-07-31
8663    56000.0    automatic    SUV         red       1  2018-05-20
46210  137000.0    automatic    SUV       black       1  2018-08-06
8662    41469.0    automatic  coupe      orange       1  2018-08-13
8661    57482.0        other  truck       white       1  2019-03-17

       days_listed  day_of_week_posted  month_posted  year_posted  \
10951            4                   2             7         2018
```

```
8804           83                3           6        2018
11438          60                3           6        2018
38692          12                4           2        2019
8702            9                5          12        2018
8664           36                1           7        2018
8663           44                6           5        2018
46210          28                0           8        2018
8662           20                0           8        2018
8661            3                6           3        2019

       vehicle_age  mileage_per_year
10951            3           19994.0
8804             3            8299.0
11438            4              11.0
38692           17            9828.0
8702             4           36868.0
8664             4           12888.0
8663             2           28000.0
46210           11           12455.0
8662             6            6912.0
8661             2           28741.0
```

It's interesting that there are so many 1 dollar cars, and these are not particularly old cars or cars have high mileage. Could that be a mistake when entering the data? Can we safely keep these out? Let's find out the proportion of these cars.

[63]:
```
len(vehicles_new.query('price == 1')) / len(vehicles_new)
```

[63]:
```
0.015574437879705213
```

The cars priced exactly 1 dollar represents slightly over 1 percent of the total cars. Therefore we can consider it safe to remove these outliers on the cheaper end of the price range. Now. Let's have a look at how many cars more priced over 50000, and 75000 respectively.

[64]:
```
len(vehicles_new.query('price >= 50000'))/ len(vehicles_new)
```

[64]:
```
0.004551243240986242
```

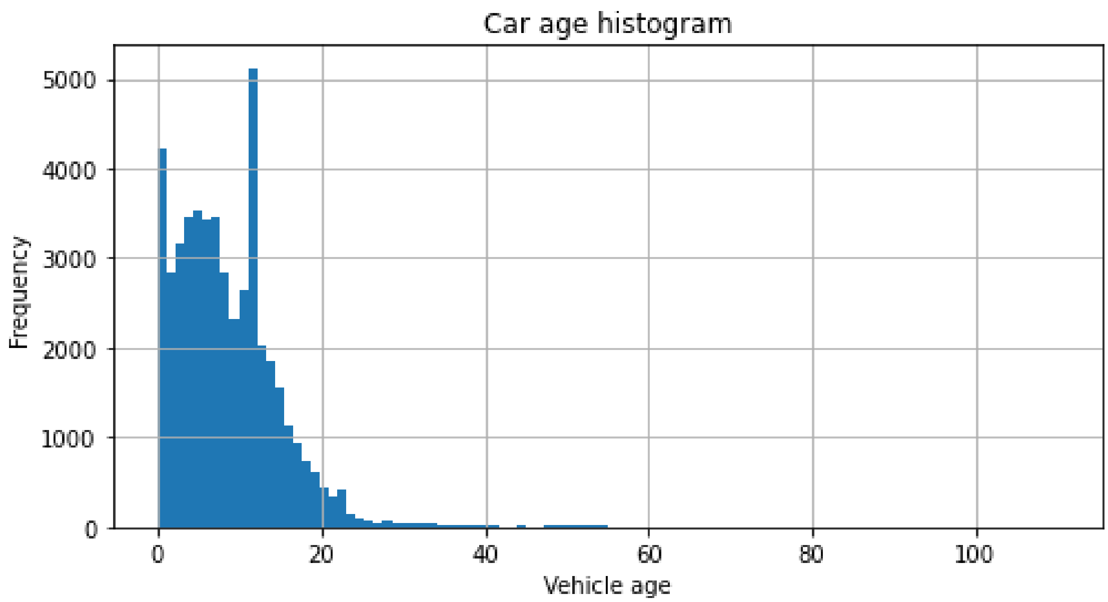
[65]:
```
len(vehicles_new.query('price >=75000')) / len(vehicles_new)
```

[65]:
```
0.0005219315643332846
```

They both represent a very small percentage of the total cars. While it would be interesting to study these outliers, they also skew the data towards the more expensive end. For now, let's arbitrarily use 50000 as the upper threshhold for the data. Please note that this is still higher than the upper whisker (3rd quantile + 1.5 * IQR), which is 34750 if we used a boxplot to show the price distribution.

Vehicle age distribution

[110]:
```
vehicles_new.hist('vehicle_age', bins = 100, grid = True, figsize = (8,4))
plt.xlabel('Vehicle age')
plt.ylabel('Frequency')
plt.title('Car age histogram')
plt.show()
```



[67]:
```
vehicles_new['vehicle_age'].describe()
```

[67]:
```
count    47899.000000
mean         8.549970
std          6.257531
min          0.000000
25%          4.000000
50%          7.000000
75%         12.000000
max        110.000000
Name: vehicle_age, dtype: float64
```

The distribution of the 'vehicle_age' shows two peaks, one at 0, meaning there are lots of cars on sale which are quite new, and one around 10 years. 75% of the cars are aged 12 years and under, but there are some outliers on the older side of the distribution, the oldest one being 110 years old. The distribution seems to be thinnning out to 0 after 40 years. Let's have a look at the proportion of cars older than 40 years.

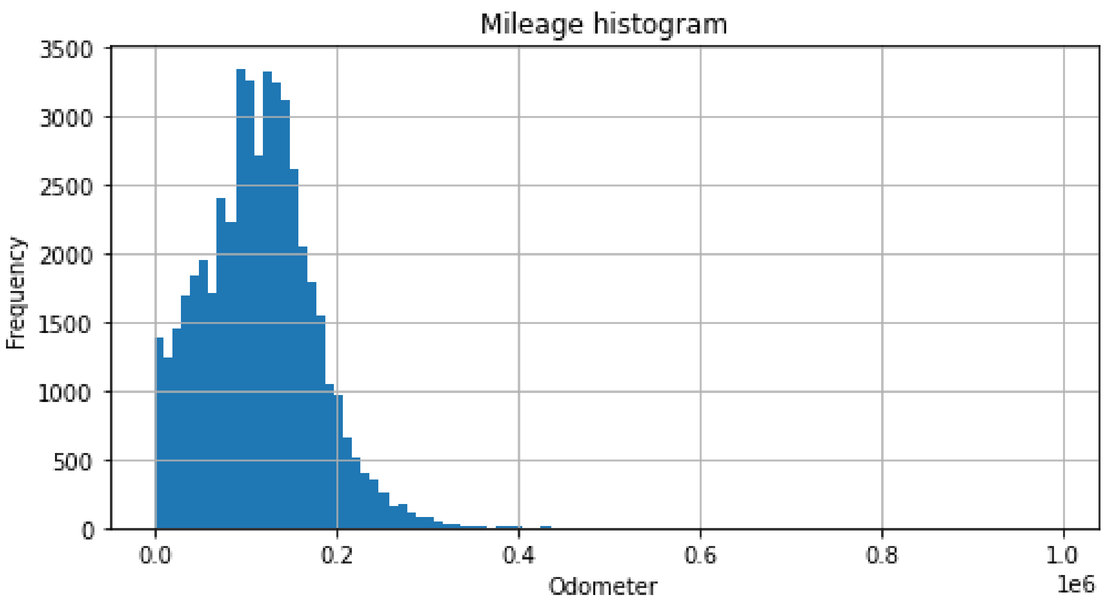
[68]:
```
len(vehicles_new.query('vehicle_age > 40'))/len(vehicles_new)
```

[68]:
```
0.0032986074865863587
```

Cars older than 40 years only represents a very small proportion. It's safe to remove these.

Odometer distribution

[111]:
```
vehicles_new.hist('odometer', bins = 100, grid = True, figsize = (8, 4))
plt.xlabel('Odometer')
plt.ylabel('Frequency')
plt.title('Mileage histogram')
plt.show()
```



[112]:
```
vehicles_new['odometer'].describe()
```

[112]:
```
count     47899.000000
mean     115081.371135
std       62424.773267
min           0.000000
25%       72757.000000
50%      114524.000000
75%      152869.000000
max      990000.000000
Name: odometer, dtype: float64
```

As show in the graph and descriptive statistics, 75% of the cars have odometers less than 153000. After 300000, the numbers is approaching zero. There are clearly some outliers on the high odometers' end. Let's find out the proportion of cars having over 300000 odometers (3rd quantile + 1.5 * IQR is 273000).

[113]:
```
len(vehicles_new.query('odometer > 300000')) / len(vehicles_new)
```

[113]: 0.005490720056786154

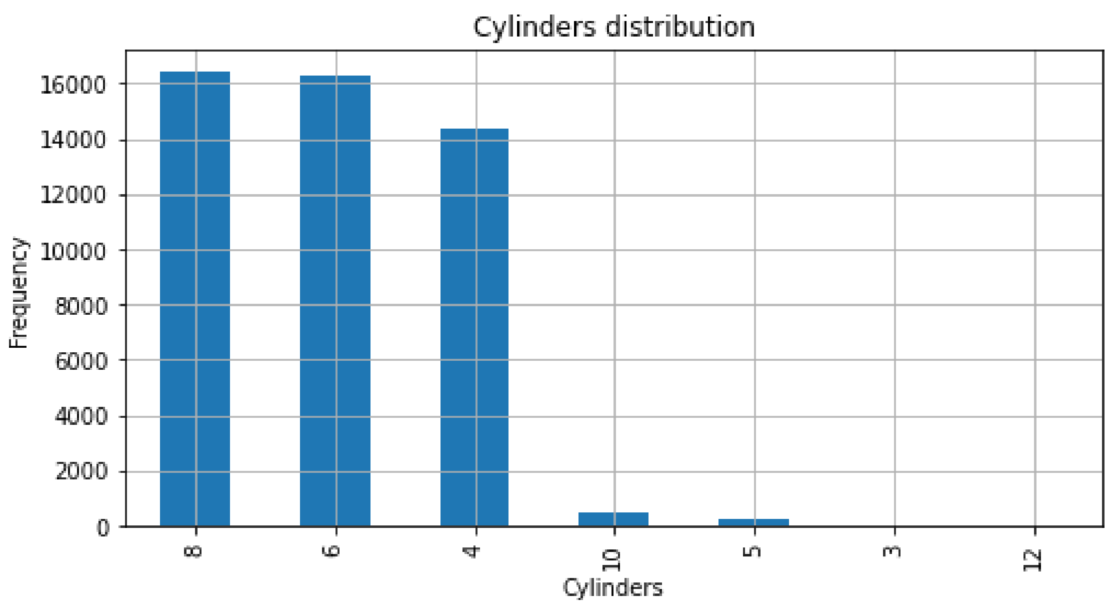
[114]:
```
vehicles_new.query('odometer > 300000')['vehicle_age'].describe()
```

[114]:
```
count    263.000000
mean      15.433460
std        6.102366
min        0.000000
25%       12.000000
50%       15.000000
75%       19.000000
max       54.000000
Name: vehicle_age, dtype: float64
```

It's a very small proportion. A quick look at the age distribution of these high odometer cars also shows that these are not exclusively old cars only. We can consider removing these outliers.

Cylinders

[115]:
```
vehicles_new['cylinders'].value_counts().plot(kind = 'bar', grid = True,␣
 ↪figsize = (8, 4))
plt.xlabel('Cylinders')
plt.ylabel('Frequency')
plt.title('Cylinders distribution')
plt.show()
```



As shown in the graph, most of the cars have 4, 6, or 8 cylinders. The bar for 3 and 12 is very low and it's really hard to see. Let's find out how many cars have 3 or 12 cylinders.

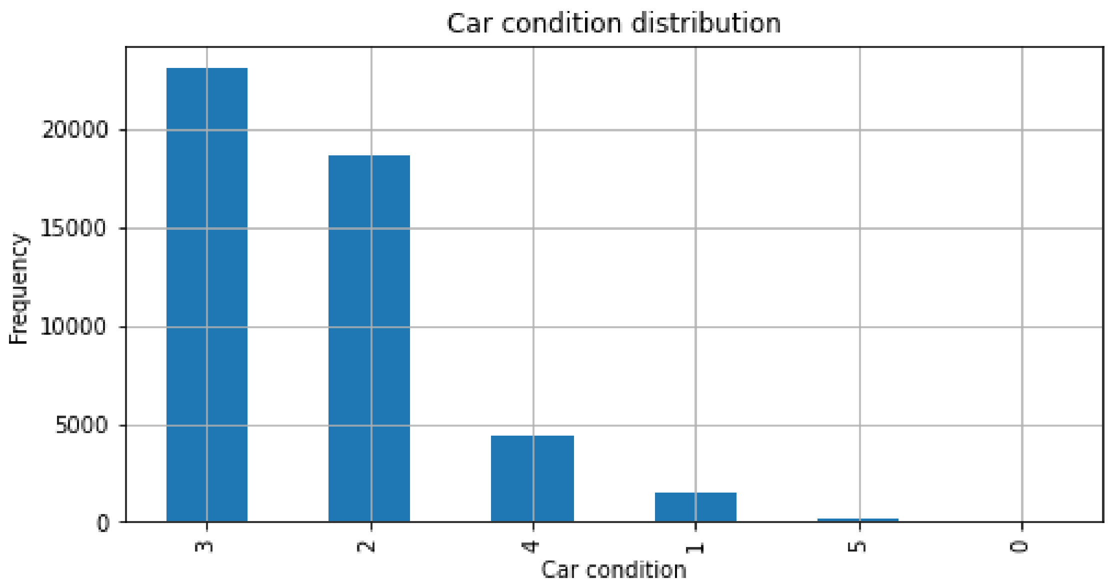
[116]:
```
len(vehicles_new.query('cylinders in (3, 12)'))
```

[116]: 35

Only 35!

Car conditions

[117]:
```
vehicles_new['condition'].value_counts().plot(kind = 'bar',grid = True, figsize␣
↪= (8, 4))
plt.xlabel('Car condition')
plt.ylabel('Frequency')
plt.title('Car condition distribution')
plt.show()
```



As shown above, most of the cars' condition are 3, or 2, excellent and good respectively. There are some cars that are new (5), and barely any are salvage(0)

1.0.11 4.2 Removing outliers

Given the information shown in the graphs above, let's remove the cars that are priced 1 or over 50000, older than 40 years, and have odometers of more than 300000. In order not to lose any data, we will keep the outliers in a separate dataset.

Slicing data

[118]:
```
vehicles_filtered = vehicles_new.query(
    'price != 1 & price <= 50000 & vehicle_age <= 40 & odometer <= 300000')
```

[119]:
```
len(vehicles_filtered)/len(vehicles_new)
```

[119]: 0.971398150274536

Slightly less than 3 percent of the data is filtered out. It's reasonable. We can proceed to save the data that is filtered out in a separate dataset and use the filtered data for the following exploratory analysis.

[120]:
```
vehicles_filtered_out = vehicles_new.query(
    'price == 1 | price > 50000 | vehicle_age > 40 | odometer > 300000')
```

[121]:
```
len(vehicles_filtered_out)/len(vehicles_new)
```

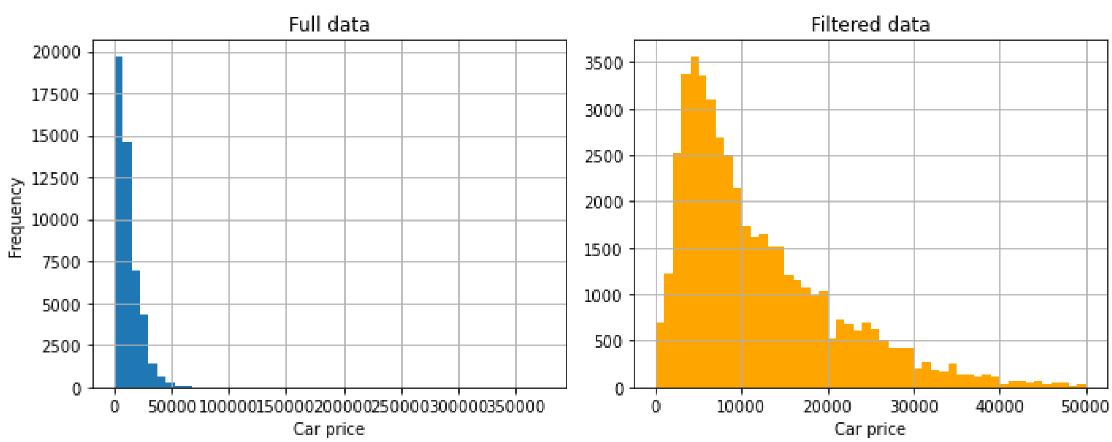
[121]: 0.028601849725463997

1.0.12 4.3 New graphs

Let's use the filtered data to plot new histograms. To see the comparison, we'll plot the two graphs next to each other.

Car price

[124]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_new['price'].hist(bins = 50, grid = True, ax = ax1)
ax1.set_title('Full data')
ax1.set_ylabel('Frequency')
ax1.set_xlabel('Car price')
vehicles_filtered['price'].hist(bins = 50, grid = True, color = 'orange',ax =␣
↪ax2)
ax2.set_title('Filtered data')
ax2.set_xlabel('Car price')

plt.tight_layout()
```

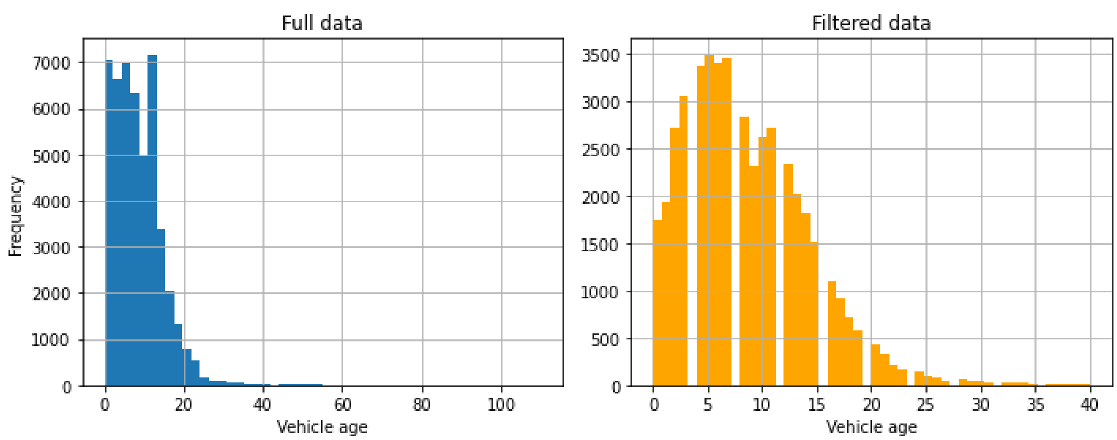


Now the histogram for the filtered data is more readable. It shows the peak of the car prices just under 5000, and then the number descreases as the price goes up.

Let's plot the histograms for the vehicles' age next.

Vehicle age

[125]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_new['vehicle_age'].hist(bins = 50, grid = True, ax = ax1)
ax1.set_title('Full data')
ax1.set_ylabel('Frequency')
ax1.set_xlabel('Vehicle age')

vehicles_filtered['vehicle_age'].hist(bins = 50, grid = True, color = 'orange',␣
 ↪ax = ax2)
ax2.set_title('Filtered data')
ax2.set_xlabel('Vehicle age')
plt.tight_layout()
```



The full data and filtered data exhibit different peaks. While there are 3 peaks for the full data, close 0, 8, and 12 year, the peak for the filtered data center around 4-7 years old. Did the removal of these 1 dollar cars or more expensive cars do that to the data? Let's have a look.

[126]:
```
vehicles_filtered_out['vehicle_age'].describe()
```

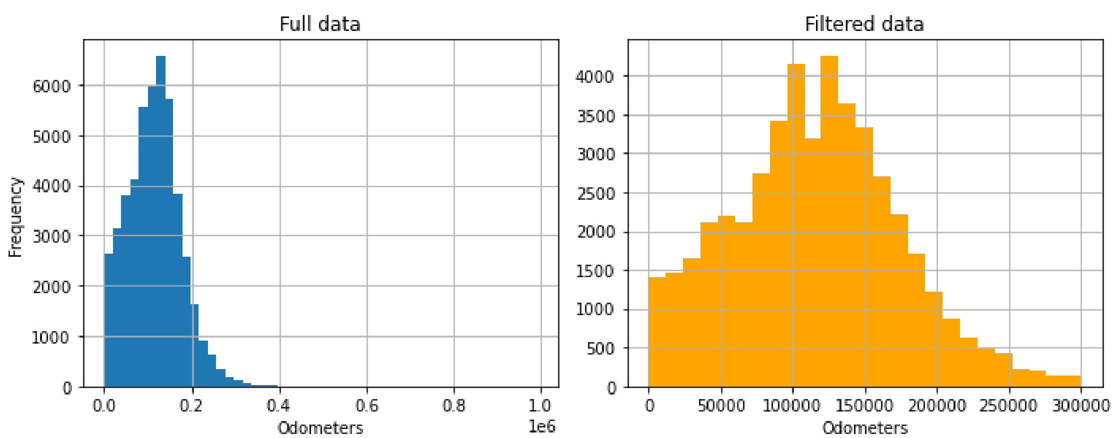
[126]:
```
count    1370.000000
mean       10.332117
std        15.875011
min         0.000000
25%         0.000000
50%         3.000000
75%        14.000000
max       110.000000
```

```
Name: vehicle_age, dtype: float64
```

Indeed, in the filtered out data, the mean age of the cars is 10 years old. Taking these away from the data, the vehicle age distribution shows different peaks from before. We are not going to do anything for the moment but it's good to keep this in mind.

Odometer

[129]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_new['odometer'].hist(bins = 50, grid = True, ax = ax1)
ax1.set_title('Full data')
ax1.set_ylabel('Frequency')
ax1.set_xlabel('Odometers')
vehicles_filtered['odometer'].hist(bins = 25, grid = True, color = 'orange', ax␣
↪= ax2)
ax2.set_title('Filtered data')
ax2.set_xlabel('Odometers')
plt.tight_layout()
```

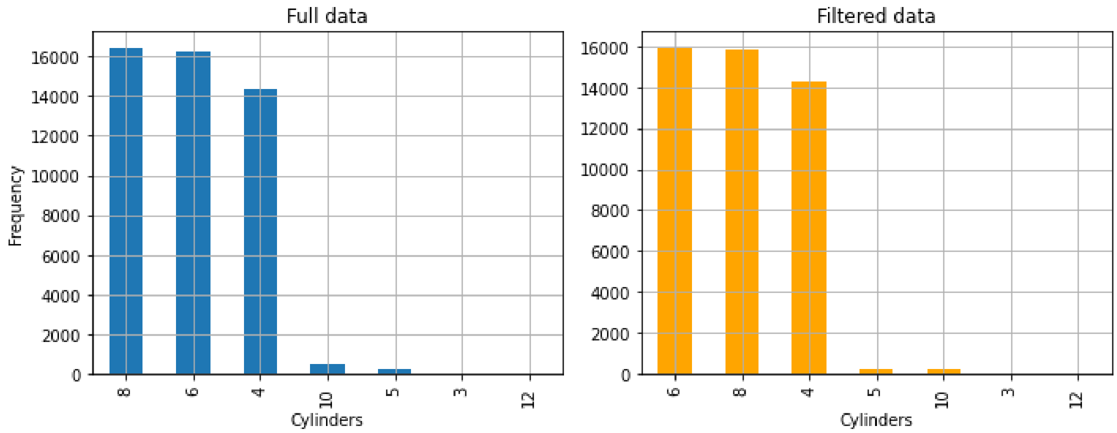


The histogram of the mileage from the filtered data is much more readable and informative. It shows a peak around 125000, and right skewed.

Cylinders

[130]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_new['cylinders'].value_counts().plot(kind = 'bar', grid = True, ax =␣
↪ax1)
ax1.set_title('Full data')
ax1.set_ylabel('Frequency')
ax1.set_xlabel('Cylinders')

vehicles_filtered['cylinders'].value_counts().plot(
    kind = 'bar', grid = True, color = 'orange', ax = ax2)
```

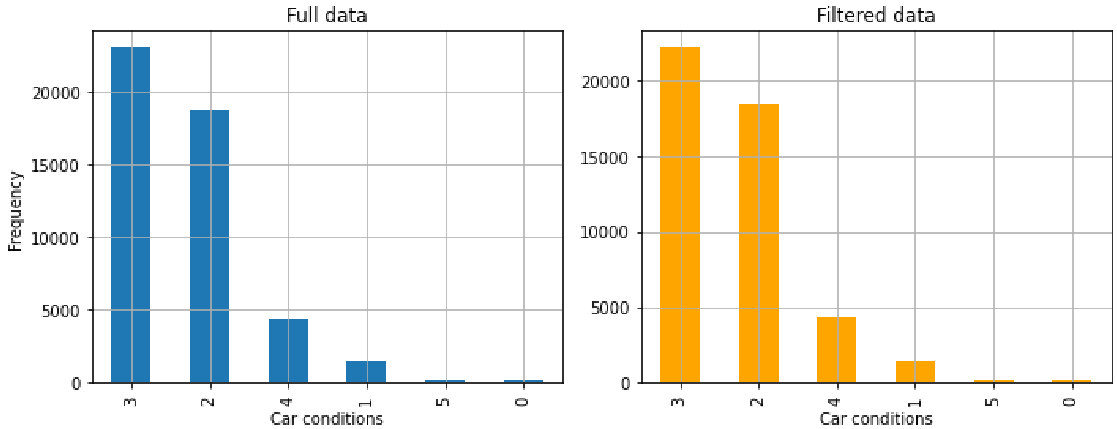
```
ax2.set_title('Filtered data')
ax2.set_xlabel('Cylinders')
plt.tight_layout()
```



Filtering the data changed the ranking postion of 6 and 8, as well as 5 and 10. This indicates that most of the cars filtered out have 8 cylinders.

Car condition

[131]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_new['condition'].value_counts().plot(kind = 'bar', grid = True, ax =␣
↪ax1)
ax1.set_title('Full data')
ax1.set_ylabel('Frequency')
ax1.set_xlabel('Car conditions')

vehicles_filtered['condition'].value_counts().plot(
    kind = 'bar', grid = True, color = 'orange', ax = ax2)
ax2.set_title('Filtered data')
ax2.set_xlabel('Car conditions')
plt.tight_layout()
```



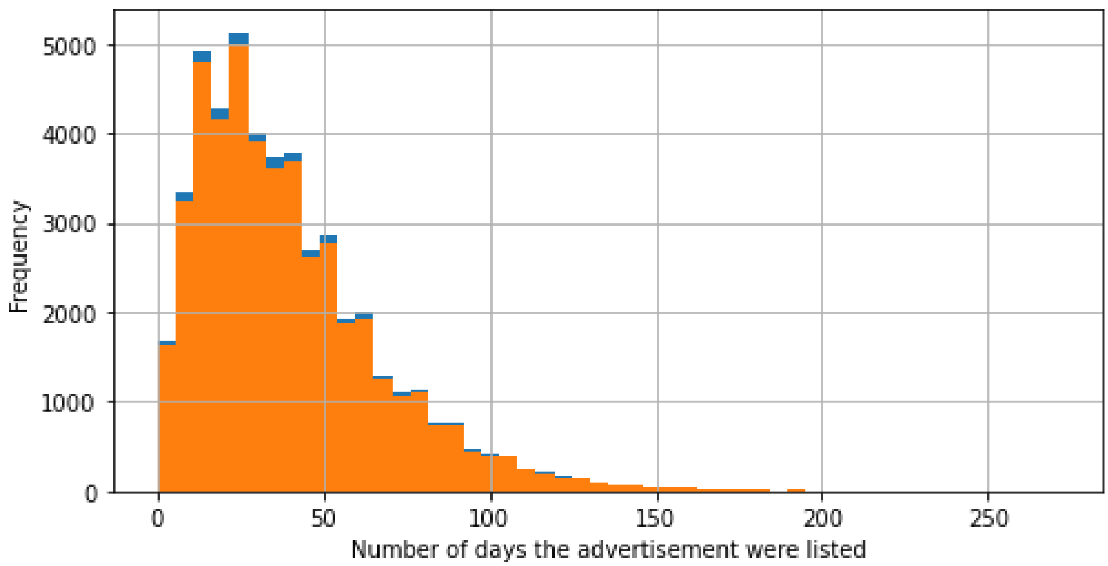
The distribution of the car conditions didn't change much, which is great! The cars that dominate are still **good** and **excellent**!

1.0.13 4.4 Life span of car advertisements

In this section, we will study how many days advertisements were displayed (days_listed) and describe the typical lifetime of an ad. We will also take a closer look at what kind of cars were removed quickly, and what kind were listed for an abnormally long time.

Just to be sure that, by filtering the data, we didn't change the distribution of this parameter dramatically, let's plot two histograms for 'days_listed', using the full dataset, and the filtered dataset.

[132]:
```
vehicles_new['days_listed'].hist(bins = 50, figsize = (8, 4))
vehicles_filtered['days_listed'].hist(bins = 50, figsize = (8, 4))
plt.xlabel('Number of days the advertisement were listed')
plt.ylabel('Frequency')
plt.show()
```



Filtering the data didn't seem to change the distribution of 'days_listed' much, which is good news.

[133]:
```
vehicles_filtered['days_listed'].describe()
```

[133]:
```
count    46529.000000
mean        39.571901
std         28.200632
min          0.000000
25%         19.000000
50%         33.000000
75%         53.000000
max        271.000000
Name: days_listed, dtype: float64
```

Using the filtered dataset, the mean for days listed is approximately 39 days, and the median 33 days. 75% of the ads were listed 53 days and under. This indicates that the typical life of a car ad is slightly over a month but less than 2 months.

Taking a look at the extreme ends of the ad life: the longest ad life is 271 days, whereas there seems to be a lot of ads which lasted less than one day! From the graph we can also see that the numbers start to thin out fast after 150.

Let's find out which cars have the days listed as 0, which ones have less than 7 days, which ones were listed over 150 days, and which car has the longest listing days.

Zero day listings

[134]:
```
vehicles_filtered.query('days_listed == 0').head()
```

[134]:
```
      price  model_year                     model  condition  cylinders  \
1165  14995        2008  chevrolet silverado 1500          3          8
1854  14000        1999                 ford f250          3          8
2707   4000        2004                  ram 1500          3          8
3702  16750        1985       chevrolet corvette          4          8
4254   5000        2007            toyota corolla          2          4

        fuel  odometer transmission       type paint_color  is_4wd  \
1165     gas   93300.0    automatic      truck        grey       1
1854  diesel  137500.0    automatic      truck         red       1
2707     gas  250000.0    automatic      truck       brown       1
3702     gas   24540.0    automatic  hatchback       white       0
4254     gas  223000.0       manual      sedan      silver       0

     date_posted  days_listed  day_of_week_posted  month_posted  year_posted  \
1165  2018-05-15            0                   1             5         2018
1854  2018-09-27            0                   3             9         2018
2707  2018-08-13            0                   0             8         2018
3702  2018-10-14            0                   6            10         2018
4254  2018-07-11            0                   2             7         2018

      vehicle_age  mileage_per_year
1165           10            9330.0
1854           19            7237.0
2707           14           17858.0
3702           33             744.0
4254           11           20273.0
```

[135]:
```python
len(vehicles_filtered.query('days_listed == 0'))
```

[135]: 49

There doesn't seem to be any particular pattern among the car ads that didn't last even for a day. Could this be a mistake?

In a real life situation, this would need to be brought to attention to the team who provided the data for further information. Fow the purpose of this report, let's have a look at ads that lastes more than 0 but gone within a week. We will artitrarily decide these are the ones that are gone quickly.

Cars with short advertisement life

[136]:
```python
vehicles_filtered.query('0 < days_listed <= 7')
```

[136]:
```
        price  model_year                     model  condition  cylinders  \
30       9499        2015              nissan altima          4          4
37       8000        2009                 ford f-150          2          8
57       5200        2006          toyota highlander          2          6
70       6950        2005            chevrolet tahoe          3          8
```

```
111    33900        2018  chevrolet silverado 1500 crew         2          8
…      …         …                             …        …        …
47792  12990        2013                  honda accord          3          6
47830   4700        2007                toyota corolla          3          4
47858   7300        2016                ford fusion se          3          4
47870   9500        2012            chevrolet traverse          2          6
47879  20481        2018                  toyota camry          4          4

      fuel  odometer transmission    type paint_color  is_4wd date_posted  \
30     gas   51848.0    automatic   sedan        grey       0  2018-11-12
37     gas  234000.0    automatic   truck       black       1  2019-03-31
57     gas  186000.0    automatic     SUV       green       0  2018-12-20
70     gas  186021.0    automatic     SUV       black       1  2018-10-30
111    gas   11315.0        other  pickup       white       1  2019-03-01
…    …        …           …      …          …      …         …
47792  gas  118659.0    automatic   coupe         red       0  2018-05-02
47830  gas  137000.0    automatic   sedan        grey       0  2018-07-17
47858  gas  106212.0    automatic   sedan        grey       0  2019-03-10
47870  gas  144500.0    automatic     SUV      silver       1  2019-03-05
47879  gas   38590.0    automatic   sedan      silver       0  2018-12-06

       days_listed  day_of_week_posted  month_posted  year_posted  \
30               7                   0            11         2018
37               1                   6             3         2019
57               2                   3            12         2018
70               3                   1            10         2018
111              2                   4             3         2019
…            …                  …            …           …
47792            6                   2             5         2018
47830            6                   1             7         2018
47858            6                   6             3         2019
47870            1                   1             3         2019
47879            4                   3            12         2018

       vehicle_age  mileage_per_year
30               3           17283.0
37              10           23400.0
57              12           15500.0
70              13           14310.0
111              1           11315.0
…            …                 …
47792            5           23732.0
47830           11           12455.0
47858            3           35404.0
47870            7           20643.0
47879            0               inf
```

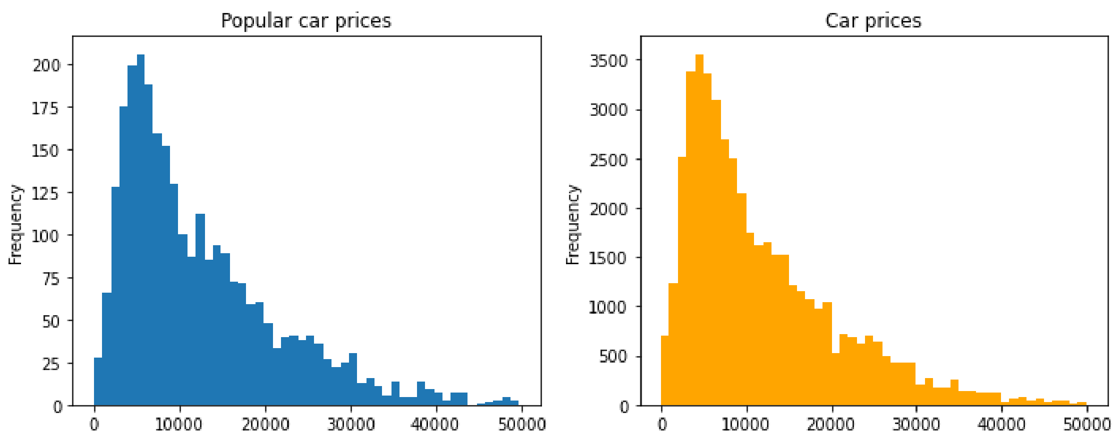
[2767 rows x 18 columns]

[137]:
```
vehicles_filtered.query('0 < days_listed <= 7').describe()
```

[137]:
```
              price   model_year    condition    cylinders       odometer  \
count   2767.000000  2767.000000  2767.000000  2767.000000     2767.00000
mean   12239.842429  2009.870618     2.659198     6.089989   115186.95627
std     9076.218664     5.729176     0.725933     1.631508    58478.01055
min        5.000000  1978.000000     0.000000     3.000000        0.00000
25%     5495.000000  2006.000000     2.000000     4.000000    73661.50000
50%     9500.000000  2011.000000     3.000000     6.000000   116510.00000
75%    16987.500000  2014.000000     3.000000     8.000000   153000.00000
max    49750.000000  2019.000000     5.000000    10.000000   300000.00000

             is_4wd  days_listed  day_of_week_posted  month_posted  \
count   2767.000000  2767.000000         2767.000000   2767.000000
mean       0.491507     4.790025            2.946151      6.728948
std        0.500018     1.835321            1.989577      3.506255
min        0.000000     1.000000            0.000000      1.000000
25%        0.000000     3.000000            1.000000      4.000000
50%        0.000000     5.000000            3.000000      7.000000
75%        1.000000     6.000000            5.000000     10.000000
max        1.000000     7.000000            6.000000     12.000000

        year_posted  vehicle_age  mileage_per_year
count   2767.000000  2767.000000            2767.0
mean    2018.295266     8.424648               inf
std        0.456245     5.726536               NaN
min     2018.000000     0.000000               0.0
25%     2018.000000     4.000000           11129.0
50%     2018.000000     7.000000           15657.0
75%     2019.000000    12.000000           22000.0
max     2019.000000    40.000000               inf
```

There are 2767 car ads which are gone within a week. Let's check out the characteristics of these cars in terms of 'price', 'model_year', 'model', 'odometer', 'condition', 'transmission' and 'color', in comparision with the whole data.

[138]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_filtered.query('0 < days_listed <= 7')['price'].plot(
    kind = 'hist', bins = 50, title = 'Popular car prices', ax=ax1)
vehicles_filtered['price'].plot(kind = 'hist', bins = 50, color = 'orange',
                                title = 'Car prices',ax=ax2)
plt.tight_layout()
```

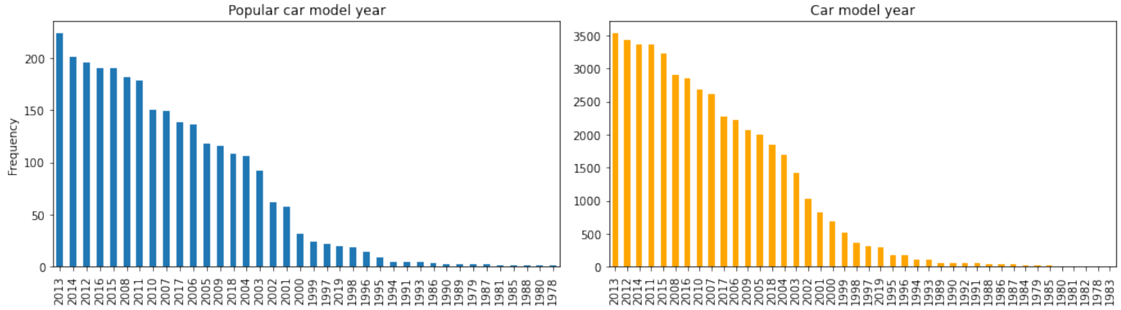


[139]:
```
print('Popular car price median:',
      vehicles_filtered.query('0 < days_listed <= 7')['price'].median())
print('Popular car price mean:',
      vehicles_filtered.query('0 < days_listed <= 7')['price'].mean())
print('Car price median:', vehicles_filtered['price'].median())
print('Car price mean:', vehicles_filtered['price'].mean())
```

```
Popular car price median: 9500.0
Popular car price mean: 12239.842428623058
Car price median: 9495.0
Car price mean: 12109.173805583614
```

The price distribution of the popular cars is almost identical with that of all cars. There is not much difference in the mean and median price for popular cars and for all cars either. Price doesn't seem to be a deciding factor for car popularity!

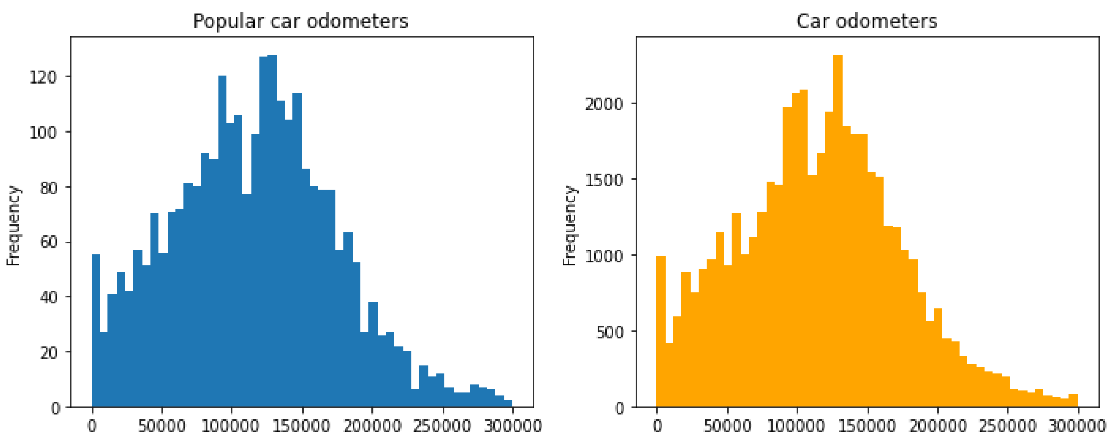
[140]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(14,4))
vehicles_filtered.query('0 < days_listed <= 7')['model_year'].value_counts().
 ↪plot(
    kind = 'bar', title = 'Popular car model year', ax=ax1)
ax1.set_ylabel('Frequency')
vehicles_filtered['model_year'].value_counts().plot(
    kind = 'bar', title = 'Car model year', color = 'orange', ax=ax2)
plt.tight_layout()
```



The distribution is almost identical. The top three popular car model years are 2013, 2014, and 2012, which are the same as the top three year models among all cars, just in slightly different oder.

It seems that how old a car is doesn't affect its popularity. Let's have a look at the odometer next.

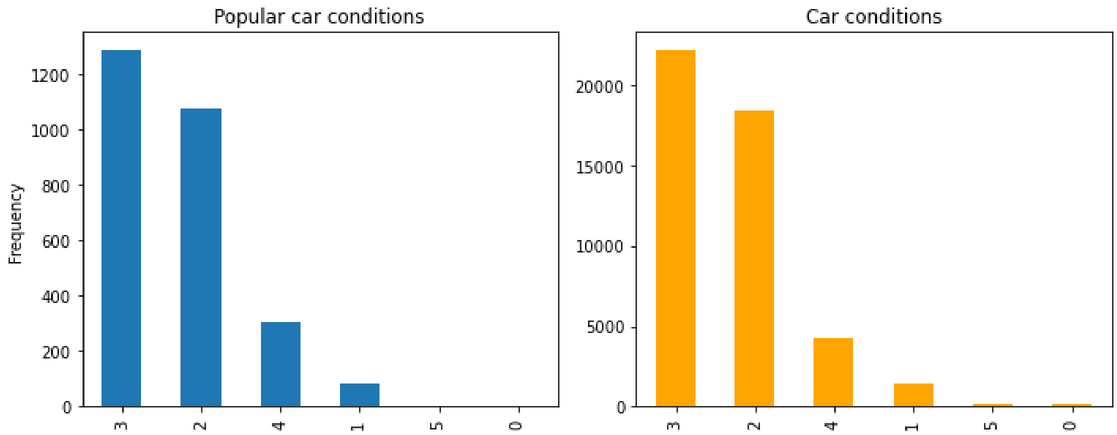
[141]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_filtered.query('0 < days_listed <= 7')['odometer'].plot(
    kind = 'hist', bins = 50, title = 'Popular car odometers', ax=ax1)
vehicles_filtered['odometer'].plot(kind = 'hist', bins = 50,
                                   title = 'Car odometers', color = 'orange',␣
 ↪ax=ax2)
plt.tight_layout()
```



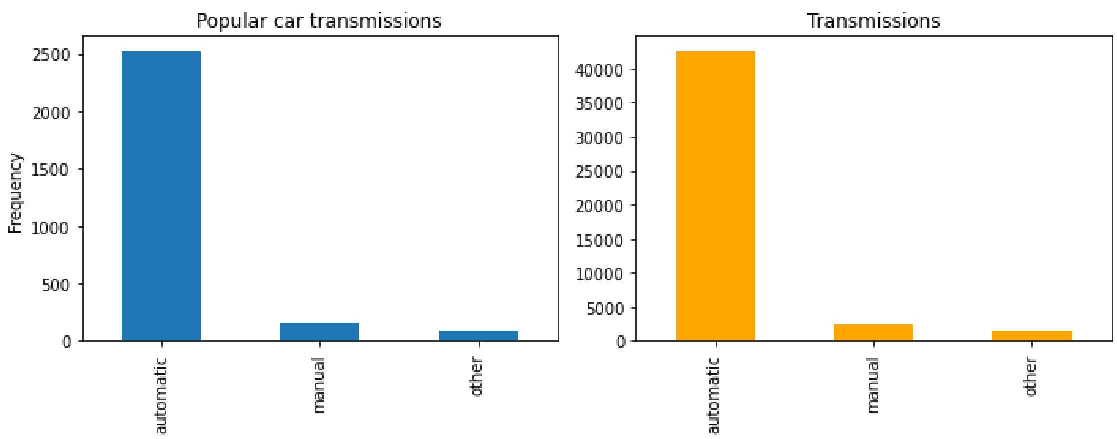
Interestingly, odometer also doesn't seem to be impacting factor. Let's have a look at condition then.

[142]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_filtered.query('0 < days_listed <= 7')['condition'].value_counts().
 ↪plot(
    kind = 'bar', title = 'Popular car conditions', ax=ax1)
ax1.set_ylabel('Frequency')
```

```
vehicles_filtered['condition'].value_counts().plot(
    title = 'Car conditions', kind = 'bar', color = 'orange', ax=ax2)
plt.tight_layout()
```

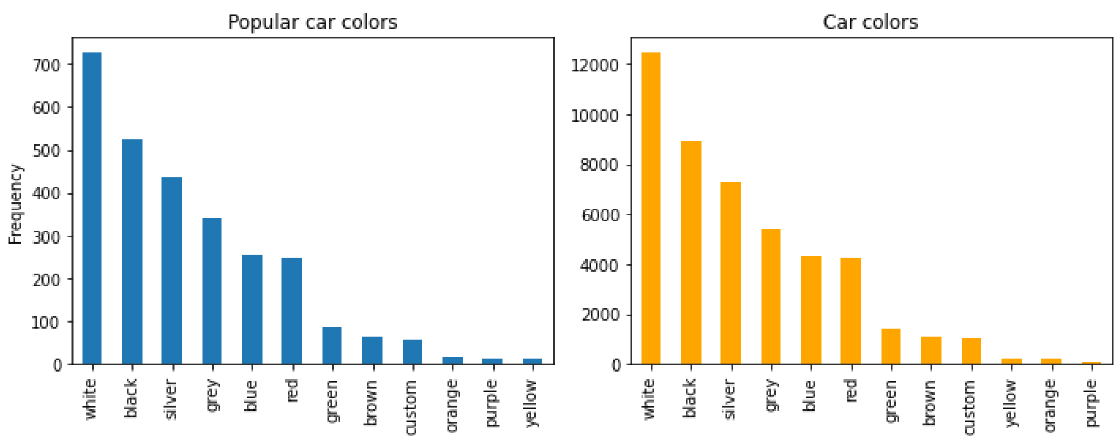


[143]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_filtered.query('0 < days_listed <= 7')['transmission'].value_counts().
 ↪plot(
    title = 'Popular car transmissions', kind = 'bar', ax=ax1)
ax1.set_ylabel('Frequency')
vehicles_filtered['transmission'].value_counts().plot(
    kind = 'bar', title = 'Transmissions', color = 'orange', ax=ax2)
plt.tight_layout()
```



[145]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
```

```
vehicles_filtered.query('0 < days_listed <= 7')['paint_color'].value_counts().
↪plot(
    kind = 'bar', title= 'Popular car colors', ax=ax1)
ax1.set_ylabel('Frequency')
vehicles_filtered['paint_color'].value_counts().plot(
    kind = 'bar', color = 'orange', title = 'Car colors', ax=ax2)
plt.tight_layout()
```



All the graphing for not much! There is indeed nothing standing out for those cars that were quickly removed!

Now let's have a look at the car ad that have stayed for over 150 days.

Cars of long advertisement life

[146]:
```
vehicles_filtered.query('days_listed > 150')
```

[146]:

	price	model_year	model	condition	cylinders	\
49	3800	2012	ford focus	2	4	
83	18800	2015	chevrolet camaro lt coupe 2d	2	6	
211	8795	2014	honda civic	3	4	
647	26995	2016	chevrolet silverado	4	8	
797	8595	2014	dodge charger	4	6	
…	…	…	…	…	…	
47338	9995	2012	toyota tacoma	3	6	
47465	8495	2013	hyundai elantra	2	4	
47711	3500	2005	toyota camry	3	4	
47864	1200	2005	volkswagen jetta	1	5	
47877	7995	2011	chevrolet equinox	4	4	

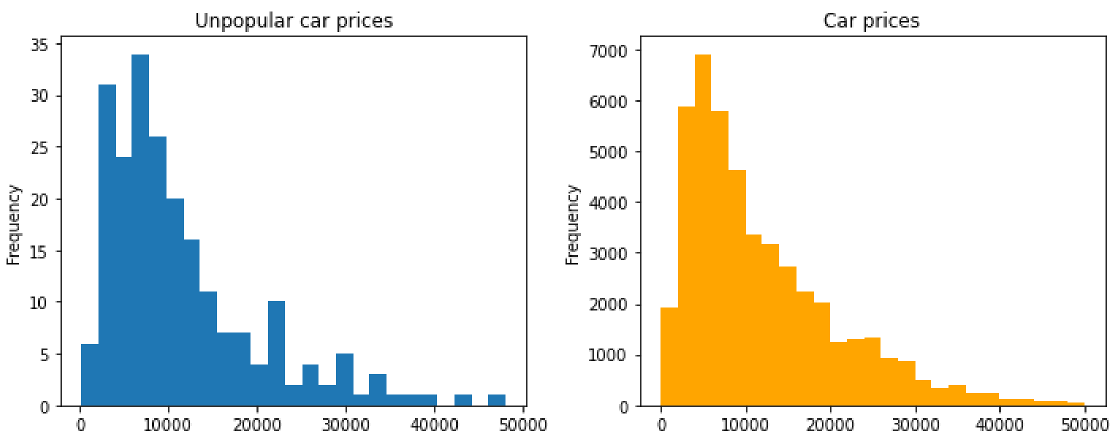
	fuel	odometer	transmission	type	paint_color	is_4wd	date_posted	\
49	gas	130323.0	automatic	sedan	black	0	2018-11-29	
83	gas	33926.0	other	coupe	grey	0	2019-01-16	

```
211     gas   85452.0    automatic   sedan        grey        0  2018-09-11
647     gas   36645.0    automatic  pickup       white        1  2018-09-01
797     gas  100004.0    automatic   sedan        blue        0  2018-10-14
…     …     …          …     …      …       …        …
47338   gas  172695.0    automatic   truck        grey        0  2018-09-19
47465   gas   55262.0    automatic   sedan        blue        0  2018-06-30
47711   gas  208299.0    automatic   sedan       green        0  2018-06-07
47864   gas  185000.0    automatic   sedan        grey        0  2018-10-10
47877   gas  111088.0    automatic     SUV       black        0  2019-04-01

       days_listed  day_of_week_posted  month_posted  year_posted  \
49             261                   3            11         2018
83             152                   2             1         2019
211            164                   1             9         2018
647            152                   5             9         2018
797            154                   6            10         2018
…            …                  …           …          …
47338          162                   2             9         2018
47465          158                   5             6         2018
47711          159                   3             6         2018
47864          158                   2            10         2018
47877          175                   0             4         2019

       vehicle_age  mileage_per_year
49               6           21721.0
83               4            8482.0
211              4           21363.0
647              2           18323.0
797              4           25001.0
…            …                …
47338            6           28783.0
47465            5           11053.0
47711           13           16023.0
47864           13           14231.0
47877            8           13886.0

[218 rows x 18 columns]
```

There are 218 car ads fitting the criterion.

Now let's compare the parameters between those cars and all cars.

[147]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_filtered.query('days_listed >150')['price'].plot(
    kind = 'hist', bins = 25, title = 'Unpopular car prices', ax=ax1)
vehicles_filtered['price'].plot(kind = 'hist', bins = 25, color = 'orange',
                                title = 'Car prices', ax=ax2)
plt.tight_layout()
```



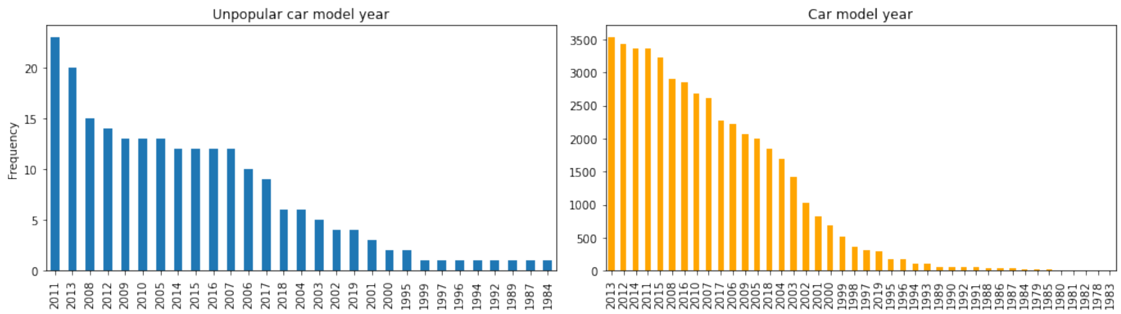
Compared to the price distribution for all cars, there seems to be a peak for not so popular car prices around 3000, then again around 6000. Let's take a look at the mean and median prices for both groups.

[148]:
```
print('Unpopular car price median:',
      vehicles_filtered.query('days_listed > 150')['price'].median())
print('Unpopular car price mean:',
      vehicles_filtered.query('days_listed > 150')['price'].mean())
print('Car price median:', vehicles_filtered['price'].median())
print('Car price mean:', vehicles_filtered['price'].mean())
```

```
Unpopular car price median: 8745.0
Unpopular car price mean: 11441.188073394496
Car price median: 9495.0
Car price mean: 12109.173805583614
```

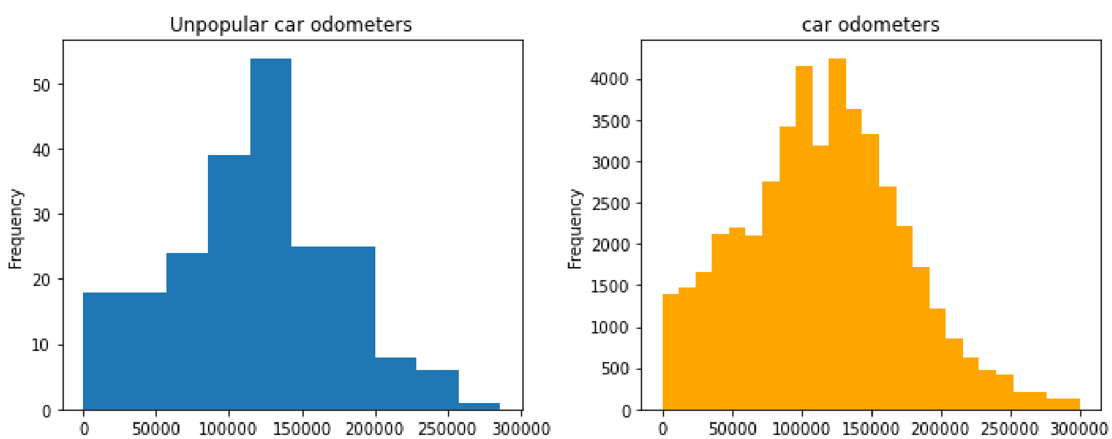
While the price distributions for both groups are similar, the unpopular cars have a slightly lower median and mean prices compared with these for all the cars.

[149]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(14,4))
vehicles_filtered.query('days_listed >150')['model_year'].value_counts().plot(
    kind = 'bar', title = 'Unpopular car model year', ax=ax1)
ax1.set_ylabel('Frequency')
vehicles_filtered['model_year'].value_counts().plot(
    kind = 'bar', color = 'orange', title = 'Car model year', ax=ax2)
plt.tight_layout()
```



The top 5 unpopular car model years are 2011, 2013, 2008, 2012, and 2009, slightly different from the top 5 car model years: 2013, 2012, 2014, 2011, and 2015.

[150]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_filtered.query('days_listed > 150')['odometer'].plot(
    kind = 'hist', bins = 10, title = 'Unpopular car odometers', ax=ax1)
vehicles_filtered['odometer'].plot(kind = 'hist', bins = 25,
                                   title = 'car odometers', color = 'orange', ax=ax2)
plt.tight_layout()
```



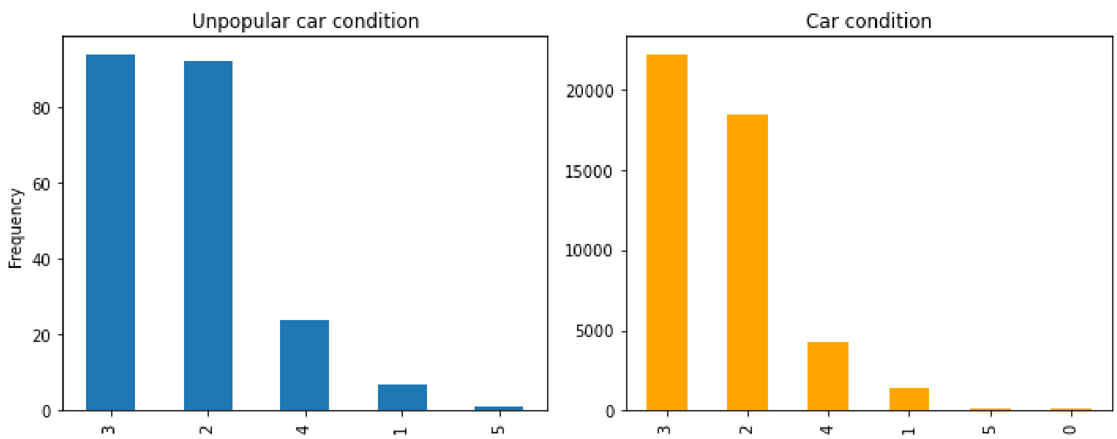
[151]:
```
print('Unpopular car odometer median:',
      vehicles_filtered.query('days_listed > 150')['odometer'].median())
print('Unpopular car odometer mean:',
      vehicles_filtered.query('days_listed > 150')['odometer'].mean())
print('Car odometer median:', vehicles_filtered['odometer'].median())
print('Car odometer mean:', vehicles_filtered['odometer'].mean())
```

```
Unpopular car odometer median: 118610.0
Unpopular car odometer mean: 116514.9495412844
```

```
Car odometer median: 116000.0
Car odometer mean: 115287.79697607944
```

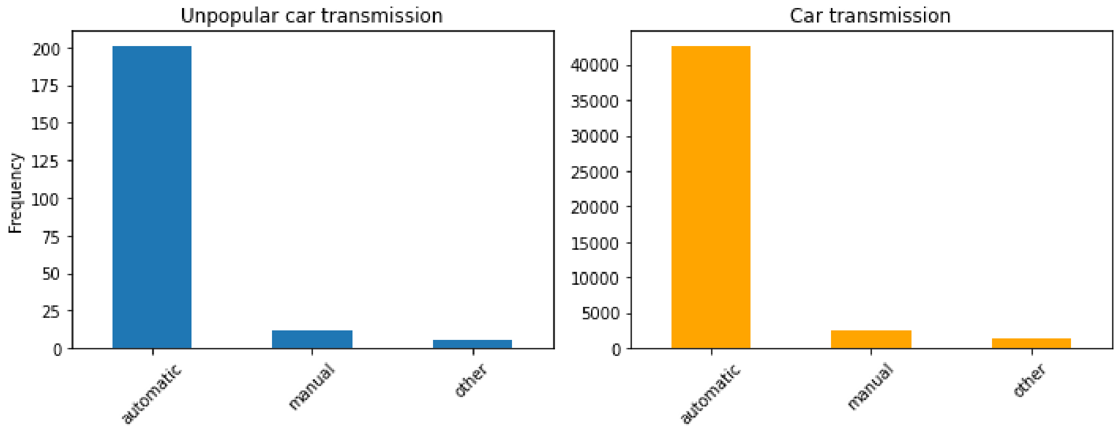
As show in both graphs and the median and mean statistics, the odometers of the unpopular cars do not differ much from cars in general.

[152]:
```python
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_filtered.query('days_listed >150')['condition'].value_counts().plot(
    kind = 'bar', title = 'Unpopular car condition', ax=ax1)
ax1.set_ylabel('Frequency')
vehicles_filtered['condition'].value_counts().plot(
    kind = 'bar', color = 'orange', title = 'Car condition', ax=ax2)
plt.tight_layout()
```



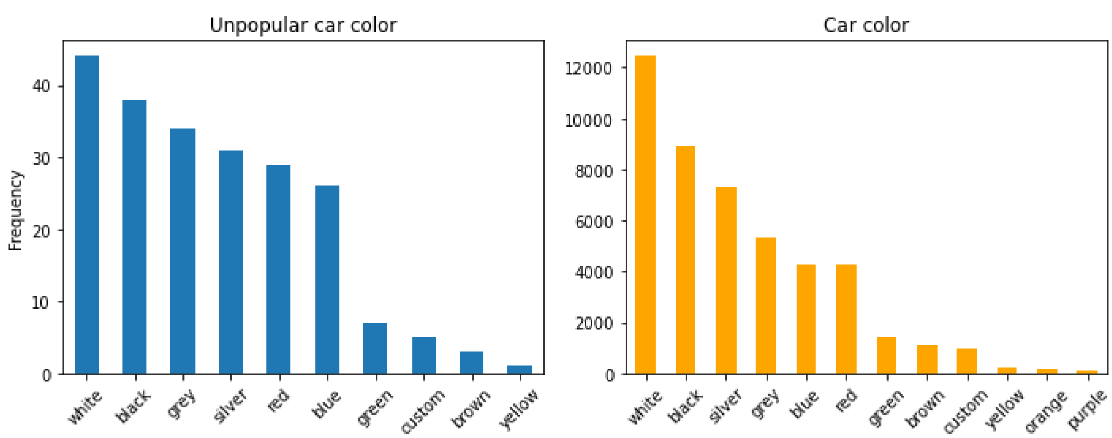
The top 2 conditions are 3 and 2, excellent and good, which is the same to distribution of car conditions over all. The only difference is that there is no condition '0' in the unpopular cars, which is the worse conditio of all. It's interesting to see that the car conditions doesn't impact much on the car popularity either.

[158]:
```python
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_filtered.query('days_listed >150')['transmission'].value_counts().plot(
    kind = 'bar', title = 'Unpopular car transmission', ax=ax1)
ax1.set_ylabel('Frequency')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation= 45)

vehicles_filtered['transmission'].value_counts().plot(
    kind = 'bar', color = 'orange', title = 'Car transmission', ax=ax2)
ax2.set_xticklabels(ax2.get_xticklabels(), rotation= 45)
plt.tight_layout()
```



Both groups show almost identitical distribution for car transmissions! Let's have a look at the last variable, color!

[159]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))
vehicles_filtered.query('days_listed >150')['paint_color'].value_counts().plot(
    kind = 'bar', title = 'Unpopular car color', ax=ax1)
ax1.set_xticklabels(ax1.get_xticklabels(), rotation= 45)
ax1.set_ylabel('Frequency')

vehicles_filtered['paint_color'].value_counts().plot(
    kind = 'bar', color = 'orange', title = 'Car color', ax=ax2)
ax2.set_xticklabels(ax2.get_xticklabels(), rotation= 45)

plt.tight_layout()
```



White and black dominate both groups, with grey and silver swapping places.

1.0.14 Conclusion

After we sorted out the cars by the number of days they were listed, we plotted the variables for both the popular ones (those that are gone within a week), and the unpopular ones(those have stayed over 150 days), and compared the pattern in these with all cars.

Suprisingly, or not, no particular pattern really stands out for both the popular and unpopular cars, except that there seems to be a lot of cars around the price of 3000 that stay longer on the listing than others.

Well, at least we now know that it is not likely that we can predict the length of time that a car ad stays on the list based on its characteristics, but what about price? Can we predict that? We will address this query in the last section of this report.

1.0.15 4.5 Number of ads and mean price by car type

In this section, we will analyze the number of ads and the average price for each type of vehicle. A graph will be plotted to show the dependence of the number of ads on the vehicle type. We will also find out which two types have the greatest number of ads.

Let's first use pivot_table to calcuate the number of ads and mean price for each type of car. We will sort the table in ascending order of the number of ads.

[160]:
```
vehicles_type = vehicles_filtered.pivot_table(
    index = 'type', values = 'price', aggfunc = ['count', 'mean'])
vehicles_type.columns = ('number_of_ads', 'mean_price')
vehicles_type = vehicles_type.sort_values('number_of_ads')
vehicles_type
```

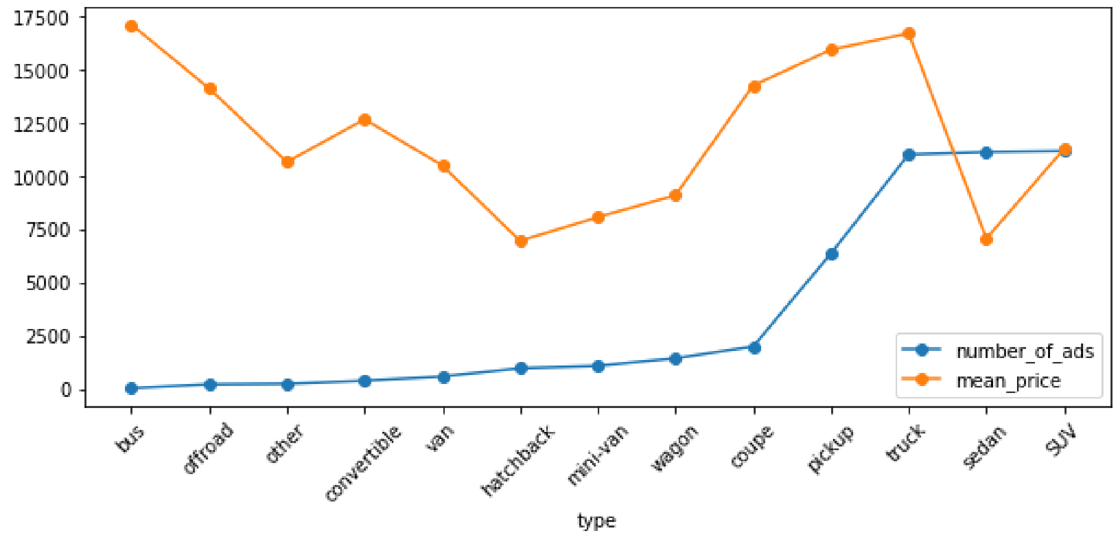
[160]:

	number_of_ads	mean_price
type		
bus	24	17135.666667
offroad	201	14129.855721
other	229	10661.296943
convertible	374	12674.647059
van	573	10504.680628
hatchback	962	6957.200624
mini-van	1069	8067.755847
wagon	1423	9095.635278
coupe	1982	14268.399092
pickup	6355	15948.229583
truck	11017	16720.881184
sedan	11136	7063.282058
SUV	11184	11285.182761

Now let's plot the number and price.

[161]:
```
vehicles_type.plot(style = 'o-', figsize = (10, 4))
plt.xticks(rotation=45)
plt.xticks(np.arange(len(vehicles_type.index)), vehicles_type.index)
```

```
plt.show()
```



As shown in the graph above, *SUV* tops the list in its number of ads, followed by *sedan*. Now let's have a look at what factors impact their car prices most.

1.0.16 4.6 Factors in car prices

In this section, we will first generate the correlation efficients for price and the numerical variables including age, mileage, and condition. Then we will make a scatterplot for each to show their relationship with prices.

For categorical variables (transmission type and color), a boxplot will be made to show their relationship with price.

Correlation coefficients

[162]:
```
vehicles_filtered.query('type == "SUV"')[[
    'price','vehicle_age', 'odometer', 'condition']].corr()
```

[162]:
```
                price  vehicle_age  odometer  condition
price        1.000000    -0.648791 -0.612848   0.294974
vehicle_age -0.648791     1.000000  0.632784  -0.332079
odometer    -0.612848     0.632784  1.000000  -0.347473
condition    0.294974    -0.332079 -0.347473   1.000000
```

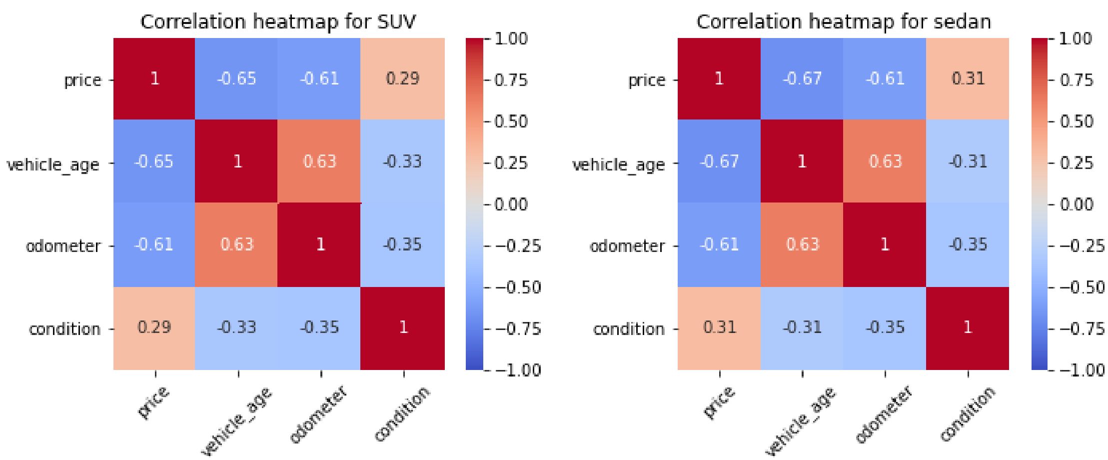
[164]:
```
vehicles_filtered.query('type == "sedan"')[[
    'price','vehicle_age', 'odometer', 'condition']].corr()
```

[164]:
```
                price  vehicle_age  odometer  condition
price        1.000000    -0.671924 -0.608063   0.311434
vehicle_age -0.671924     1.000000  0.629729  -0.310252
```

```
odometer     -0.608063      0.629729  1.000000  -0.348089
condition     0.311434     -0.310252 -0.348089   1.000000
```

Heatmap to show the correlations

```
[173]: fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,4))

sns.heatmap(vehicles_filtered.query('type == "SUV"')[[
    'price','vehicle_age', 'odometer', 'condition']].corr(),
            annot = True, fmt='.2g', vmin=-1, vmax=1,
            center= 0,cmap = 'coolwarm', square = True, ax = ax1)
ax1.set_title('Correlation heatmap for SUV')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation= 45)

sns.heatmap(vehicles_filtered.query('type == "sedan"')[[
    'price','vehicle_age', 'odometer', 'condition']].corr(),
            annot = True, fmt='.2g', vmin=-1, vmax=1,
            center= 0,cmap = 'coolwarm', square = True, ax = ax2)
ax2.set_title('Correlation heatmap for sedan')
ax2.set_xticklabels(ax2.get_xticklabels(), rotation= 45)

plt.tight_layout()
```

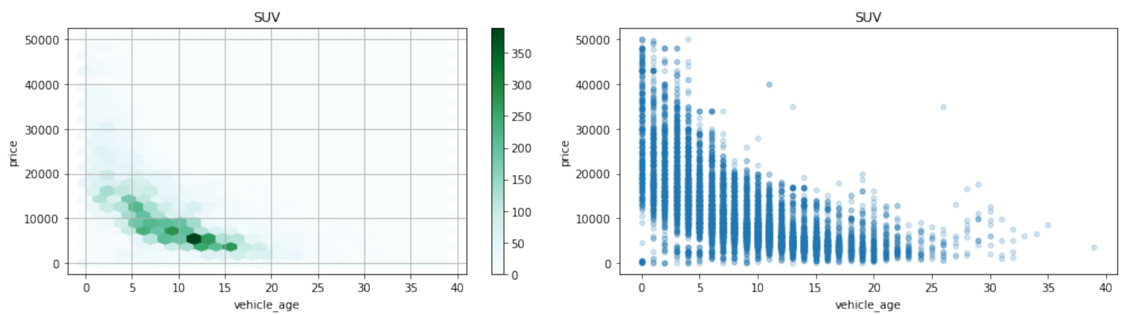


The above numbers show that, for both SUVs and sedans, car prices have relatively strong negative relationship with their age, and total mileage, whereas their relationship with car's condition is not as strong. Let's take a closer look at the scatterplot of each of these pairs.

Price and vehicle age

Give the large number of data, let's use both *hexbin* and *scatterplot* to depict the relationship among the data.

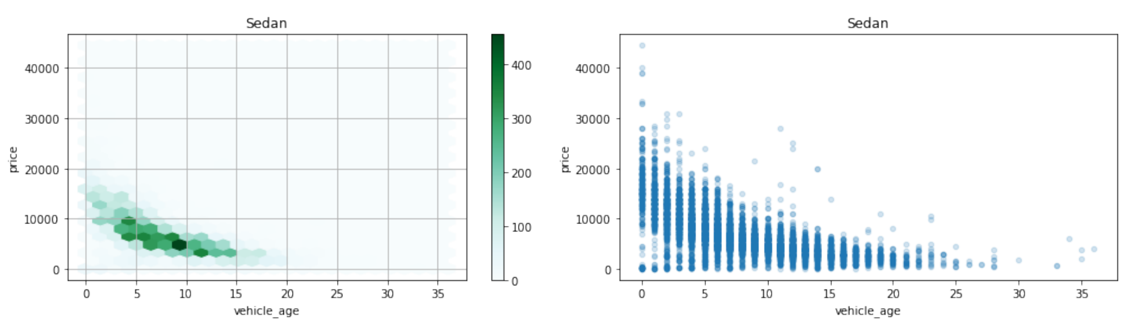
[174]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(14,4))
vehicles_filtered.query('type == "SUV"').plot(
    x = 'vehicle_age', y = 'price', kind = 'hexbin',
    gridsize=25, sharex=False, grid=True, title = 'SUV', ax = ax1)
vehicles_filtered.query('type == "SUV"').plot(
    x = 'vehicle_age', y = 'price', kind = 'scatter', title = 'SUV', alpha = 0.
↪2, ax = ax2)
plt.tight_layout()
```



The graphs above show that, as SUV cars get older, not only the price overall decreases, the range of prices also become smaller.

SUV of approximately 11 years and having the price around 7000 seem to be the most popular among the ads. Now let's have a look at the sedans.

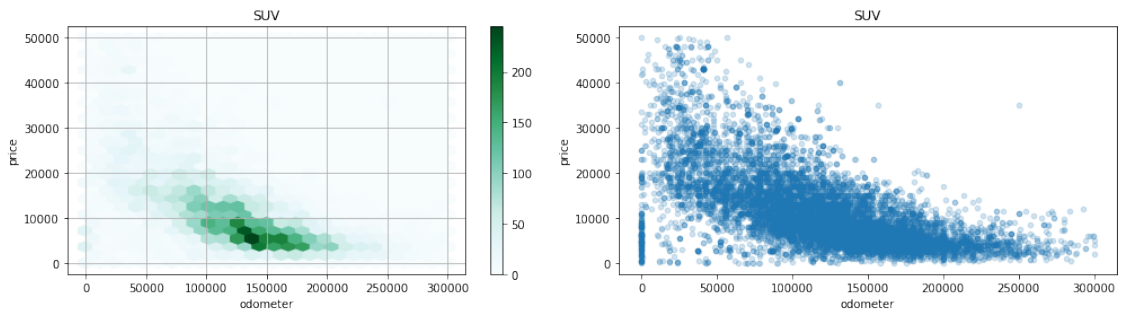
[175]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(14,4))
vehicles_filtered.query('type == "sedan"').plot(
    x = 'vehicle_age', y = 'price', kind = 'hexbin',
    gridsize=25, sharex=False, grid=True, title = 'Sedan', ax = ax1)
vehicles_filtered.query('type == "sedan"').plot(
    x = 'vehicle_age', y = 'price', kind = 'scatter', title = 'Sedan', alpha =␣
↪0.2, ax = ax2)
plt.tight_layout()
```



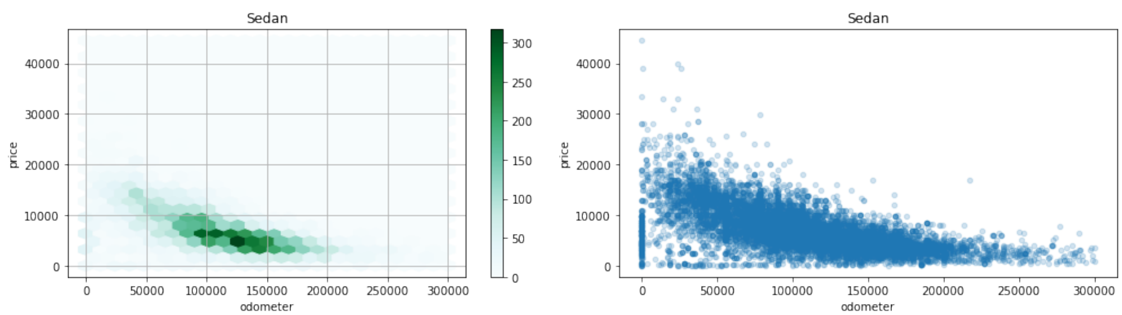
Sedans show a similar patter as SUV, with overall cheaper prices, and younger car age.

Price and odometers

[176]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(14,4))
vehicles_filtered.query('type == "SUV"').plot(
    x = 'odometer', y = 'price', kind = 'hexbin',
    gridsize=25, sharex=False, grid=True, title = 'SUV', ax = ax1)
vehicles_filtered.query('type == "SUV"').plot(
    x = 'odometer', y = 'price', kind = 'scatter', title = 'SUV', alpha = 0.2, ax = ax2)
plt.tight_layout()
```



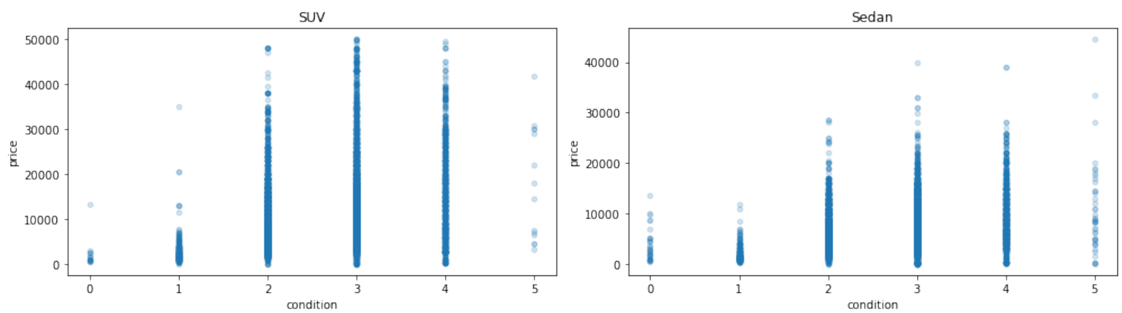
[177]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(14,4))
vehicles_filtered.query('type == "sedan"').plot(
    x = 'odometer', y = 'price', kind = 'hexbin',
    gridsize=25, sharex=False, grid=True, title = 'Sedan', ax = ax1)
vehicles_filtered.query('type == "sedan"').plot(
    x = 'odometer', y = 'price', kind = 'scatter', title = 'Sedan', alpha = 0.2, ax = ax2)
plt.tight_layout()
```



For both SUV and sedan cars, the graphs show a moderately strong negative correlation between prices and their total mileage. This makes sense, the older a car is, the cheaper it might sell.

Price and car conditions

[178]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(14,4))
vehicles_filtered.query('type == "SUV"').plot(
    x = 'condition', y = 'price', kind = 'scatter', title = 'SUV', alpha = 0.2,␣
 ↪ax = ax1)
vehicles_filtered.query('type == "sedan"').plot(
    x = 'condition', y = 'price', kind = 'scatter', title = 'Sedan', alpha = 0.
 ↪2, ax = ax2)
plt.tight_layout()
```



The graphs show that cars of both **SUV** and **sedan** type cluster around condition 2, 3, 4, which correspond to **good, excellent, like new**, and the prices for these car have a wide range.

Cars of condition 0 and 1, which are salvage and poor, are at the lower end of the price range. There are not as many new cars and but their prices seem to vary.

Price and transmission types

Before we plot the prices for different transmission types and colors, let's check how many ads there are in each category.

[179]:
```
vehicles_filtered.query('type == "SUV"')['transmission'].value_counts()
```

[179]:
```
automatic    10629
manual         468
other           87
Name: transmission, dtype: int64
```

[180]:
```
vehicles_filtered.query('type == "sedan"')['transmission'].value_counts()
```

[180]:
```
automatic    10518
manual         547
other           71
Name: transmission, dtype: int64
```

[181]:
```
vehicles_filtered.query('type == "SUV"')['paint_color'].value_counts()
```

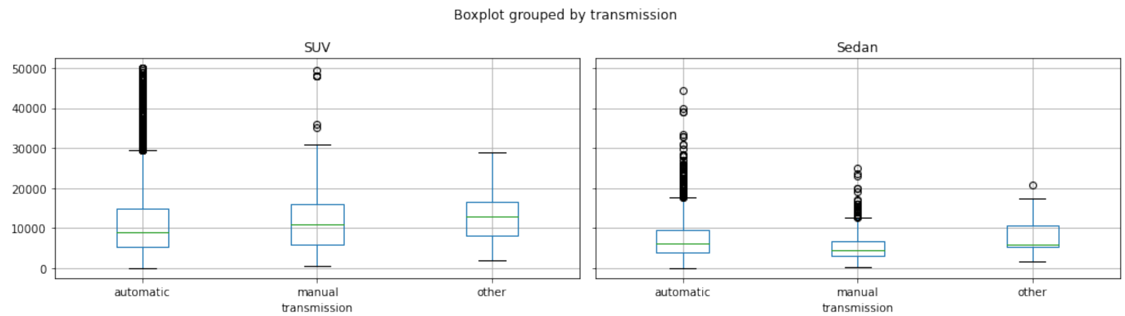
[181]:
```
black     3020
white     2004
silver    1865
grey      1254
blue      1034
red        836
green      432
brown      326
custom     259
orange      73
yellow      44
purple      37
Name: paint_color, dtype: int64
```

[182]:
```
vehicles_filtered.query('type == "sedan"')['paint_color'].value_counts()
```

[182]:
```
silver    2769
black     2259
grey      1953
white     1378
blue      1189
red        861
custom     246
brown      236
green      191
purple      25
yellow      21
orange       8
Name: paint_color, dtype: int64
```

For SUVs, yellow and purple cars have less than 50 in number, and for sedans, purple, yellow, and orange has less than 50. The boxplot wouldn't work for these categories and therefore let's remove these before the plotting.

[183]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(14,4), sharey = True)
vehicles_filtered.query('type == "SUV"').boxplot(
    column = 'price', by = 'transmission', ax = ax1)
ax1.set_title('SUV')
vehicles_filtered.query('type == "sedan"').boxplot(
    column = 'price', by = 'transmission', ax = ax2)
ax2.set_title('Sedan')
plt.tight_layout()
```

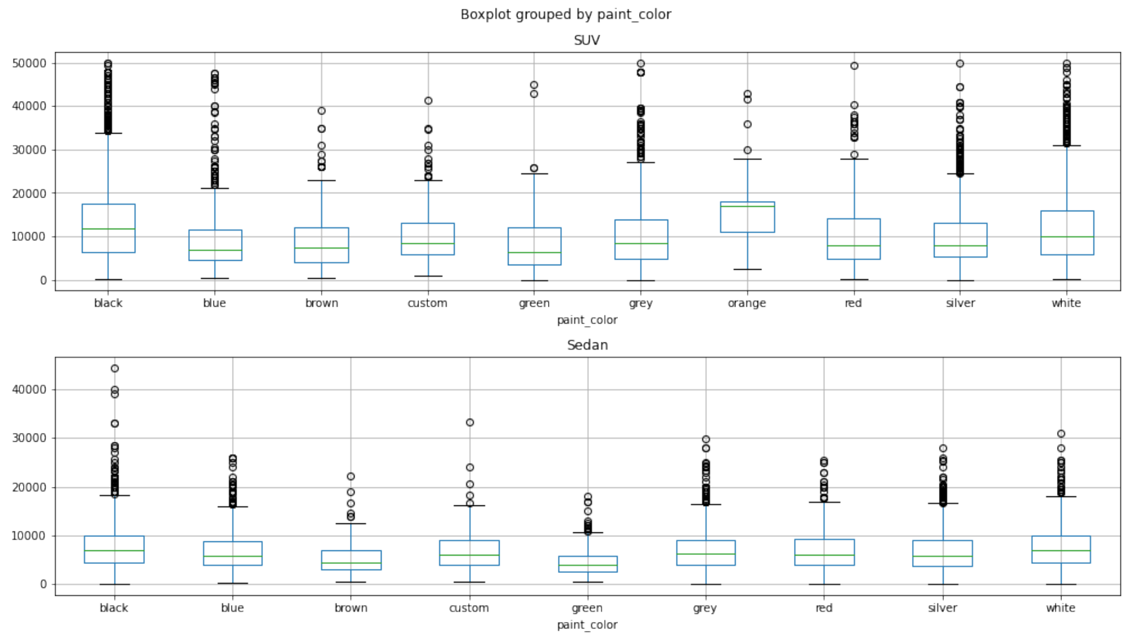


As shown in the graphs above, for the **SUV** type, manual cars have slightly higher median prices than automatic cars, whereas *other* have higher median prices than manual SUVs. It would be interesting to find out what *other* means. Automatic SUVs also have a lot of outliers towards the higher end of the price range.

In comparison, the median prices for automatic **sedans** are more expensive then manual sedans, with both transmission types having a rather dense distribution of outliers towards the higher price end. Similar to SUVs, *other* transmission type also has a higher median price than the other two types.

The IQR for sedans, shown as the boxes in the boxplots, are narrower than that for SUVs across all transmission types. This indicates that sedans prices have less dispersion than SUV prices.

[184]:
```
fig, (ax1,ax2) = plt.subplots(2,1, figsize=(14,8))
vehicles_filtered.query(
    'type == "SUV" & paint_color not in ("purple", "yellow")').boxplot(
    column = 'price', by = 'paint_color', ax = ax1)

vehicles_filtered.query(
    'type == "sedan" & paint_color not in ("orange", "purple", "yellow")').
 ↪boxplot(
    column = 'price', by = 'paint_color', ax = ax2)
ax1.set_title('SUV')
ax2.set_title('Sedan')
plt.tight_layout()
```



For **SUVs**, orange cars have the highest median price, followed by black and white cars. However it needs to be pointed out that there are only a total of 73 orange cars. Green SUVs have the lowest median price. Black and white SUVs also have quite densely distributed outliers over the upper whisker. Overall the car prices are right skewed.

For **sedans**, black, custom, grey, red, silder and white have relatively higher median prices than the others. Green sedans have the lowest median price. Similar to SUVs, the prices for all colors are right skewed. However, the sedans do not have as wide dispersion as SUVs, as indicated by their narrower boxes.

1.0.17 Part 5: Overall conclusion

The exploratory analysis in this report examined a dataset of free advertisements for vehicles on Crankshaft list over slightly less than a year.

The data was preprocessed by having the rows containing missing model years removed. The outliers as identified by car price, age, and mileage have also been moved to a separate dataset.

Using the preprocessed data, we examined the distribution of price, model year, cylinders, and mileage. The dominant price range seems to be around 5000, age 4 to 7 years, mileage around 125,000. The most popular cylinder categories are 6, 8, and 4, and the most popular conditions are 'good' and 'excellent'.

Examining the number of days the advertisements are listed, we discoverd those that were gone with a week, and those that have stayed for over 5 months! However, a closer look at each of the parameters of those quick gone and 'permanent resident' cars doesn't really show any particular patterns.

We also found that the types of cars that have the most advertisements are SUVs and sedans, whose prices are most negatively impacted by their mileage and age.

Finally, it would also be interesting to examine the outliers, depending on what questions we are asking. However, this is beyond the scope of this report. There are also a couple of issues that might need attention of the team who provided the data, such as the missing model year values, and the 1 dollar car price.

1.0.18 Project completion checklist

- ☒ file opened
- ☒ files explored (first rows printed, info() method)
- ☒ missing values determined
- ☒ missing values filled in
- ☒ clarification of the discovered missing values provided
- ☒ data types converted
- ☒ explanation of which columns had the data types changed and why
- ☒ calculated and added to the table: day of the week, month, and year the ad was placed
- ☒ calculated and added to the table: the vehicle's age (in years) when the ad was placed
- ☒ calculated and added to the table: the vehicle's average mileage per year
- ☒ the following parameters investigated: price, vehicle's age when the ad was placed, mileage, number of cylinders, and condition
- ☒ histograms for each parameter created
- ☒ task completed: "Determine the upper limits of outliers, remove the outliers and store them in a separate DataFrame, and continue your work with the filtered data."
- ☒ task completed: "Use the filtered data to plot new histograms. Compare them with the earlier histograms (the ones that included outliers). Draw conclusions for each histogram."
- ☒ task completed: "Study how many days advertisements were displayed (days_listed). Plot a histogram. Calculate the mean and median. Describe the typical lifetime of an ad. Determine when ads were removed quickly, and when they were listed for an abnormally long time."
- ☒ task completed: "Analyze the number of ads and the average price for each type of vehicle. Plot a graph showing the dependence of the number of ads on the vehicle type. Select the two types with the greatest number of ads."
- ☒ task completed: "What factors impact the price most? Take each of the popular types you detected at the previous stage and study whether the price depends on age, mileage, condition, transmission type, and color. For categorical variables (transmission type and color), plot box-and-whisker charts, and create scatterplots for the rest. When analyzing categorical variables, note that the categories must have at least 50 ads; otherwise, their parameters won't be valid for analysis."
- ☒ each stage has a conclusion
- ☒ overall conclusion drawn