Hello Ameer!

My name is Olga. I'm happy to review your project today.

When I will see mistake at the first time, I will just point it out. I let you find it and fix it by yourself. I'm trying to prepare you to work as an Data Analyst. SO, at a real job, your team lead will do the same. But if you can't handle this task yet, I will give you a more accurate hint at the next iteration.

Below you will find my comments - **please do not move, modify or delete them**.

You can find my comments in green, yellow or red boxes like this:

Reviewer's comment

Success. Everything is done succesfully.

Reviewer's comment

Remarks. Some recommendations.

Reviewer's comment

Needs fixing. The block requires some corrections. Work can't be accepted with the red comments.

You can answer me by using this:

Student answer.

Text here.

Overallv reviewer's comment

Ameer, thank you for sending your project. You've done a really good job on it!

While there's room for improvement, on the whole, your project is impressive good.I like your filling missing numbers in your project. Also you ploted excellent graphs! You did a great job! I am really impressed!

There are few things that need to be corrected in your project. They're mostly minor issues that are easy to fix. I wrote comments, please, check them.

Every issue with our code is a chance for us to learn something new.

Good luck! :)

Overallv reviewer's comment. V.2.

Hello Ameer! I really appreciate the corrections you sent in! Thanks for taking the time to do so.

I'm glad to say that your project has been accepted.Congratulations!

Keep up the good work, and good luck on the next sprint! :)

For your future projects

- We should always add a title, axis labels and grid on graph. It help to understand our graph correctly.

1 What sells a car?

I'm an analyst at Crankshaft List. Hundreds of free advertisements for vehicles are published on the site every day. I need to study data collected over the last few years and determine which factors influence the price of a vehicle.

Purpose of Project: The purpose of the project is to study and analyze the data we have in hand about an advertising website for selling vehicles of all types. We need to find the correlation between several factors such as; vehicle color, vehicle model , vehicle type etc, to the price and conclude whether those factors have an impact on it (price) or not.

Plan of Work: We are going to start by inspecting the data we were given, look for errors, duplicates, missing values etc. Once we are done with these steps, we proceed into fixing these problematic data by removing unnecessary trash data, correcting all the values that can be corrected and retaining missing values where possible. Then we move on with our clean data and start studying the correlations between the factors we have, in-order to conclude whether any of them really affects the price the vehicle was advertisied for.

Reviewer's comment

Thank you for the introduction!

1.1 Initialization

We start by loading the required libraries for our project:

In [1]:

```
# Loading all the libraries
import pandas as pd
import numpy as np
import datetime as dt
```

Reviewer's comment

Good idea to import all libraries here!

1.1.1 Loading data

Next we load the files of data that we have:

In [2]:

```
# Loading the data file into a DataFrame
vehicle_data = pd.read_csv('/datasets/vehicles_us.csv')
```

In [3]:

```
# Taking a look at our dataframe
vehicle_data
```

Out[3]:

	price	model_year	model	condition	cylinders	fuel	odometer	transmission	type
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
...	...	...	...	...	...	...	...	...	...
51520	9249	2013.0	nissan maxima	like new	6.0	gas	88136.0	automatic	sedan
51521	2700	2002.0	honda civic	salvage	4.0	gas	181500.0	automatic	sedan
51522	3950	2009.0	hyundai sonata	excellent	4.0	gas	128000.0	automatic	sedan
51523	7455	2013.0	toyota corolla	good	4.0	gas	139573.0	automatic	sedan
51524	6300	2014.0	nissan altima	good	4.0	gas	NaN	automatic	sedan

51525 rows × 13 columns

1.1.2 Explore initial 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

Next we are going to look at the dataframe information/describition:

In [4]:

```
# print the general/summary information about the DataFrame
vehicle_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51525 entries, 0 to 51524
Data columns (total 13 columns):
price           51525 non-null int64
model_year      47906 non-null float64
model           51525 non-null object
condition       51525 non-null object
cylinders       46265 non-null float64
fuel            51525 non-null object
odometer        43633 non-null float64
transmission    51525 non-null object
type            51525 non-null object
paint_color     42258 non-null object
is_4wd          25572 non-null float64
date_posted     51525 non-null object
days_listed     51525 non-null int64
dtypes: float64(4), int64(2), object(7)
memory usage: 5.1+ MB
```

In [5]:

```
vehicle_data.describe()
```

Out[5]:

	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

We haven't started yet and we can already see some errors in our data just by looking at the information and the dataframe itself. The *info()* method provided us with information about missing values in columns like; model_year, cylinders, odometer, paint_color and is_4wd. the *describe()* method provided us with information about the values we have in our dataframes, here we can see some problematic values as well such as; **minimum price is 1**.

Now that we know that we have some problematic values/missing values, lets move on and find them:

In [6]:

```
# Find how many missing values we got
vehicle_data.isnull().sum()
```

Out[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
```

As we mentioned above, we have some columns that include alot of missing values. It is interesting to find the distribution of missing values in every column. Below we check this matter:

In [7]:

```
for col in vehicle_data:
    col_missing = vehicle_data[col].isna().sum()
    print('{} : {:.0%}'.format(col, col_missing/len(vehicle_data)))
```

```
price : 0%
model_year : 7%
model : 0%
condition : 0%
cylinders : 10%
fuel : 0%
odometer : 15%
transmission : 0%
type : 0%
paint_color : 18%
is_4wd : 50%
date_posted : 0%
days_listed : 0%
```

Looking at the percentages of missing values in each column, they all are greater than 5% (which is ~2500 values) therefore we have to work on getting them back as correctly as humanily possible using all the data we have in hand and using the mean/median of each column when needed.

1.1.3 Conclusions and further steps

So far, we only looked at the data from the outside (using *info()* and *describe()* methods). We already calculated how much missing values we have and which columns excatly has them. Surprisingly the **price** has no missing values, hoo-ray! But on the other hand, some other columns have some massive amounts of missing values like the **is_4wd** column and **paint_color** column. As bad as that looks, there is still hope that we still can find the main factors that may have an impact on the prices, because for instance; missing values in **cylinders**

column can be retrieved using data from different rows with similar values (similar models have similar cylinders) and other values can be filled with mean/median. Besides, it is safe to assume that paint color or wheel-drive type aren't really among those factors and it is more likely to be the **condition**, **odometer**, **model** and **model_year**.

Next: we are going to treat the missing values we have found in this phase of the project and then correct any values that need to be corrected (or removed when necessary). Then removing any duplicates (Although our data doesn't have a unique identifier for every row, but it is still safe to assume that two rows with identical values and identical **date_posted** are possibly duplicates, *BUT ITS EVEN BETTER IF WE FIND OUT THAT THERE'S NO DUPLICATES ANYWAY*)

Reviewer's comment

So, we see basic information about our data. Too much missing values here. Let's do something!:)

1.2 Treating missing values (if any)

We'll start with filling the values of the **model_year** column since it has the least amount of missing values:

Since we can't really guess what years each of these missing values were therefore we are going to be using the mean or median to fill in the missing values.

In [8]:

```
# Check how many missing values we have again
vehicle_data['model_year'].isna().sum()
```

Out[8]:

```
3619
```

In [9]:

```
# Checking for mean and median
vehicle_data['model_year'].describe()
```

Out[9]:

```
count    47906.000000
mean      2009.750470
std          6.282065
min       1908.000000
25%       2006.000000
50%       2011.000000
75%       2014.000000
max       2019.000000
Name: model_year, dtype: float64
```

In [10]:

```
# filling model year missing values with the median
vehicle_data['model_year'] = vehicle_data.groupby('model')['model_year'].transform(lambda g
```

In [11]:

```
# checking if we still have missing values
vehicle_data['model_year'].isna().sum()
```

Out[11]:

```
0
```

~~**Reviewer's comment**~~

~~We should use model and model year for filling missing numbers.It show us more real numbers.~~

Student answer. Thanks for the tip! Although I don't think this method will help us get more accurate results but I agree with you about that this may give us more real numbers (For instance, old models with missing values will now have old model years since I am comparing them with their group now and not with the whole dataset, we might not end up with more accurate numbers, but we're defenitely ending with more real numbers)

(FIXED)

Reviewer's comment. V.2.

Here we need mean results, so, this method is good. Well done! :)

We filled the missing values of the **model_year** column with the median because the mean was pulled down due to the minimum model year being very low (1908).

Next we'll treat missing values in the **cylinders** column:

In [12]:

```
# Check how many missing values we have again
vehicle_data['cylinders'].isna().sum()
```

Out[12]:

```
5260
```

In [13]:

```
# filling cylinder's missing values using data we already have
vehicle_data['cylinders'] = vehicle_data.groupby('model')['cylinders'].transform(lambda grp
```

In [14]:

```
# Checking if we still have missing values
vehicle_data['cylinders'].isnull().sum()
```

Out[14]:

```
0
```

We are aware that some models of similar types have different cylinders number (due to an error possibily) therefore we grouped the data by the **model** and filled in the missing values for vehicles according to the median of the group it belongs to. (Assuming that the majority of vehicles of each model have the correct cylinders number, then the median is the right choice here!)

Next we we'll treat missing values in the **odometer** column:

In [15]:

```
# Check how many missing values we have again
vehicle_data['odometer'].isna().sum()
```

Out[15]:

```
7892
```

In [16]:

```
# filling missing values in odometer with the help of grouping by model_year
vehicle_data['odometer'] = vehicle_data.groupby('model_year')['odometer'].transform(lambda
```

Reviewer's comment

Tha same here about model. :)

Student answer. Hello Olga, here I grouped by the **model_year** only because , on average, people drive more or less the same mileage per year (the average driver) so I wanted to base my work on that. I don't think that the **model** is actually a factor to be considered here because mileage driven is after all related to the vehicle owner/driver and not to the model. For instance, you can have two Skodas (same model), one belongs to someone that travels 30km to work, and the other Skoda belongs to someone else that travels 100km for work, although both are the same **model** but that doesn't mean they travel the same amount of miles/kms just because they are the same model. I hope I made myself clear! If you meant something else above then I will be glad to know about that in your response. Thank you by the way!

Reviewer's comment. V.2.

You explanation is very good. Than you! Here in model coulmn we have full name: bmw x5, hyundai sonata,toyota corolla,honda civic, etc. So, we can say, for example bmw x5 can have bigger odometrs value than honda civic. Because this car is SUV and perhaps owner use it often. We cannot use type because we have very different models.

By the way, only year is a great idea. :)

In [17]:

```
# Checking if we still have missing values
vehicle_data['odometer'].isnull().sum()
```

Out[17]:

```
0
```

What we did above is; we filled the missing values of the **odometer** column with the help of groups of the same **model_year**. We divided the vehicles into groups of the same model year and filled in missing values according to the median of each group.

Next we'll treat missing values in the **paint_color** column:

In [18]:

```
# Check how many missing values we have again
vehicle_data['paint_color'].isna().sum()
```

Out[18]:

```
9267
```

In [19]:

```
# Filling the missing values in the paint column
vehicle_data['paint_color'] = vehicle_data['paint_color'].fillna("unknown")
```

In [20]:

```
# checking for value counts
vehicle_data['paint_color'].value_counts()
```

Out[20]:

```
white      10029
unknown     9267
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
```

Regarding missing values in the **paint_color** column, we decided to fill them with the value 'unknown' due to the fact that there's no median/mean for non-numeric values and we don't have a way to guess each vehicles colors. Besides, missing values in this column could've been intentional due to vehicle owners not filling in this information.

Finally, we'll treat the missing values in the column with most missing values which is **is_4wd**:

In [21]:

```
# Check how many missing values we have again
vehicle_data['is_4wd'].isna().sum()
```

Out[21]:

```
25953
```

In [22]:

```
# we fill missing values of the 4wd
vehicle_data['is_4wd'] = vehicle_data['is_4wd'].fillna(0)
```

~~**Reviewer's comment**~~

~~is 4wd is a bool column. We should fill by 0 here.~~

Student answer. Hey again! Thanks for the tip, I didn't bother to check the values of this column at the start. But when I read your message, I checked for them and found out that all of the values that weren't missing were 1's that means it is safe to assume that the ones that are missing are actually/possibly zeros.

(FIXED)

Reviewer's comment. V.2.

Good practice to check columns at the beginning. :)

In [23]:

```
# Check if have any missing values still
vehicle_data['is_4wd'].isnull().sum()
```

Out[23]:

```
0
```

We had alot of missing values in the **is_4wd** column. But one thing that we noticed was that all the values we had were 1's, that means it is pretty reasonable to assume that all of the missing values are actually 0's and they were missing due to a systematic error. Therefore we decided to fill all the missing values with zeros.

Thusfar, we filled all the missing values we detected at the start, to make sure of that we can run another test:

In [24]:

```
# one last check to check if we still have missing values
vehicle_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51525 entries, 0 to 51524
Data columns (total 13 columns):
price           51525 non-null int64
model_year      51525 non-null float64
model           51525 non-null object
condition       51525 non-null object
cylinders       51525 non-null float64
fuel            51525 non-null object
odometer        51525 non-null float64
transmission    51525 non-null object
type            51525 non-null object
paint_color     51525 non-null object
is_4wd          51525 non-null float64
date_posted     51525 non-null object
days_listed     51525 non-null int64
dtypes: float64(4), int64(2), object(7)
memory usage: 5.1+ MB
```

All columns have 51525 non-null values. We don't have any missing values.

1.3 Fixing data types and values

Looking at the dataframe we have, we have some other issues to deal with.

First, it is good to cast the **date_posted** column from str to datetime since datetime has alot more features as a date than a normal str.

In [25]:

```
# Changing the type of date_posted to dt
vehicle_data['date_posted'] = pd.to_datetime(vehicle_data['date_posted'], format='%Y-%m-%d'
vehicle_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51525 entries, 0 to 51524
Data columns (total 13 columns):
price           51525 non-null int64
model_year      51525 non-null float64
model           51525 non-null object
condition       51525 non-null object
cylinders       51525 non-null float64
fuel            51525 non-null object
odometer        51525 non-null float64
transmission    51525 non-null object
type            51525 non-null object
paint_color     51525 non-null object
is_4wd          51525 non-null float64
date_posted     51525 non-null datetime64[ns]
days_listed     51525 non-null int64
dtypes: datetime64[ns](1), float64(4), int64(2), object(6)
memory usage: 5.1+ MB
```

In [26]:

```
# Adding 3 new columns for the day of the week, month and year the advert was posted on
vehicle_data['day_posted'] = pd.to_datetime(vehicle_data['date_posted']).dt.weekday
vehicle_data['month_posted'] = pd.to_datetime(vehicle_data['date_posted']).dt.month
vehicle_data['year_posted'] = pd.to_datetime(vehicle_data['date_posted']).dt.year
```

In [27]:

```
vehicle_data.head()
```

Out[27]:

	price	model_year	model	condition	cylinders	fuel	odometer	transmission	type	paint
0	9400	2011.0	bmw x5	good	6.0	gas	145000.0	automatic	SUV	ur
1	25500	2011.0	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	161397.0	automatic	pickup	ur
4	14900	2017.0	chrysler 200	excellent	4.0	gas	80903.0	automatic	sedan	

Now that we have convereted **date_posted** from str to datetime, there's one other change we can do to our dateframe, which is changing the type of the **condition** from str to a numerical type (out of 5 for example, 5 for best condition, 0 for worst).

In [28]:

```
# It might help to replace the condition values with something that can be manipulated more
conditions = ['salvage', 'fair', 'good', 'excellent', 'like new', 'new']
for index, cond in enumerate(conditions):
    vehicle_data['condition'] = vehicle_data['condition'].replace([cond],index)

vehicle_data['condition'].value_counts()
```

Out[28]:

```
3    24773
2    20145
4     4742
1     1607
5      143
0      115
Name: condition, dtype: int64
```

Next we are going to dive more in our dataframe, check for more values that needs to be added or changed!

1.4 Enrich data

For easier analysis, it is always good to get as much information as possible from the data we have and add this information into the dataframe for later use if it was of importance. For example, we can add vehicle **age** to our dataframe, this can help us determine whether the age of a vehicle has an impact on the price it was advertisied for.

In [29]:

```
# Adding the vehicle's age when the ad was placed
vehicle_data['model_year'].astype('int64').dtypes
vehicle_data['age'] = vehicle_data['year_posted'] - vehicle_data['model_year'] + 1
```

~~**Reviewer's comment**~~

~~For age we need to use: vehicle_data['age'] = vehicle_data['year_posted'] - vehicle_data['model_year'] + 1~~

~~It help us to avoid nulls. :)~~

Student answer. I fixed this one. Thanks! Though, I didn't quite understand how we could end up with a null after a simple arithmetic, would love to know more about this for future projects!

(FIXED)

Reviewer's comment. V.2.

It is not real arithmetic. It is arithmetic from car dealers. Thay said: "you bought a car and it is not new. 1 years old.".

Perhas, I bought a carin 1st of January. And I want to sell it now. Februrary. Really, not full year, but we should write 1. The also reason that our age column is int and it cannot be 0 for mile_per_year. :)

In [30]:

```
# checking that we have filled the column correctly
vehicle_data['age'].max()
```

Out[30]:

```
111.0
```

In [31]:

```
vehicle_data['age'].min()
```

Out[31]:

```
1.0
```

In [32]:

```
# Checking if we missed any values
vehicle_data['age'].isna().sum()
```

Out[32]:

```
0
```

Another example would be, adding vehicle's average mile per year (according to a vehicle's age & odometer) as done below:

In [33]:

```
# Adding the vehicle's average mileage per year
vehicle_data['mile_per_year'] = vehicle_data['odometer'] // vehicle_data['age']
vehicle_data.loc[vehicle_data['age'] == 0, 'mile_per_year'] = vehicle_data['odometer'] # be
```

In [34]:

```
# Checking if we have any missing values
vehicle_data['mile_per_year'].isna().sum()
```

Out[34]:

```
0
```

We created a new column **mile_per_year** and filled it with the average total miles/age for any vehicle with age above 1 year(s). For vehicles with 1 or 0 year(s) we simply copied the data of the vehicle's odometer.

Now we have two new columns, **age** and **mile_per_year** that will help us in our further investigation!

Lets take a look at the dataframe after the changes:

In [35]:

```
vehicle_data.head(20)
```

Out[35]:

	price	model_year	model	condition	cylinders	fuel	odometer	transmission	type	pa
0	9400	2011.0	bmw x5	2	6.0	gas	145000.0	automatic	SUV	
1	25500	2011.0	ford f-150	2	6.0	gas	88705.0	automatic	pickup	
2	5500	2013.0	hyundai sonata	4	4.0	gas	110000.0	automatic	sedan	
3	1500	2003.0	ford f-150	1	8.0	gas	161397.0	automatic	pickup	
4	14900	2017.0	chrysler 200	3	4.0	gas	80903.0	automatic	sedan	
5	14990	2014.0	chrysler 300	3	6.0	gas	57954.0	automatic	sedan	
6	12990	2015.0	toyota camry	3	4.0	gas	79212.0	automatic	sedan	
7	15990	2013.0	honda pilot	3	6.0	gas	109473.0	automatic	SUV	
8	11500	2012.0	kia sorento	3	4.0	gas	104174.0	automatic	SUV	
9	9200	2008.0	honda pilot	3	6.0	gas	147191.0	automatic	SUV	
10	19500	2011.0	chevrolet silverado 1500	3	8.0	gas	128413.0	automatic	pickup	
11	8990	2012.0	honda accord	3	4.0	gas	111142.0	automatic	sedan	
12	18990	2012.0	ram 1500	3	8.0	gas	140742.0	automatic	pickup	
13	16500	2018.0	hyundai sonata	3	4.0	gas	22104.0	automatic	sedan	
14	12990	2009.0	gmc yukon	3	8.0	gas	132285.0	automatic	SUV	
15	17990	2013.0	ram 1500	3	8.0	gas	99840.0	automatic	pickup	
16	14990	2010.0	ram 1500	3	8.0	gas	130725.0	automatic	pickup	
17	13990	2014.0	jeep cherokee	3	6.0	gas	100669.0	automatic	SUV	
18	12500	2013.0	chevrolet traverse	3	6.0	gas	128325.0	automatic	SUV	
19	13990	2018.0	hyundai elantra	3	4.0	gas	31932.0	automatic	sedan	

1.5 Checking clean data

Before we move on to the analysis part, we take a final look at our data, at a deep level, to make sure we have our data as clean as possible. This makes our analysis much more accurate and easier.

First we want to check for any zeros in places it shouldn't be at:

In [36]:

```
for i in vehicle_data: #checking for any zeros.
    print(i, len(vehicle_data[vehicle_data[i] == 0]))
```

```
price 0
model_year 0
model 0
condition 115
cylinders 0
fuel 0
odometer 185
transmission 0
type 0
paint_color 0
is_4wd 25953
date_posted 0
days_listed 54
day_posted 7339
month_posted 0
year_posted 0
age 0
mile_per_year 220
```

Looking at the list generated by the code above, we can see that; price, model_year, model, cylinders, fuel, transmission, type, ..., year_posted has 0 zeros, which is good and expected. while condition, odometer, is_4wd, days_listed, days_posted actually have zeros, but again this is very fine since these columns are supposed to have zeros in them (For instance, 0 on day_posted means 'Monday' and 0 in condition means 'Salvage' etc).

Next we look at the general values for some of our coulmns:

In [37]:

```
vehicle_data['fuel'].value_counts()
```

Out[37]:

```
gas         47288
diesel       3714
hybrid        409
other         108
electric        6
Name: fuel, dtype: int64
```

In [38]:

```
vehicle_data['transmission'].value_counts()
```

Out[38]:

```
automatic    46902
manual        2829
other         1794
Name: transmission, dtype: int64
```

In [39]:

```
vehicle_data['type'].value_counts()
```

Out[39]:

```
SUV            12405
truck          12353
sedan          12154
pickup          6988
coupe           2303
wagon           1541
mini-van        1161
hatchback       1047
van              633
convertible      446
other            256
offroad          214
bus               24
Name: type, dtype: int64
```

In [40]:

```
vehicle_data['model_year'].describe()
```

Out[40]:

```
count    51525.000000
mean      2009.793954
std          6.099296
min       1908.000000
25%       2007.000000
50%       2011.000000
75%       2014.000000
max       2019.000000
Name: model_year, dtype: float64
```

In [41]:

```
vehicle_data['model'].describe()
```

Out[41]:

```
count          51525
unique           100
top       ford f-150
freq            2796
Name: model, dtype: object
```

In [42]:

```
vehicle_data['odometer'].describe()
```

Out[42]:

```
count     51525.000000
mean     115199.667559
std       62082.598563
min           0.000000
25%       73500.000000
50%      114072.000000
75%      152862.000000
max      990000.000000
Name: odometer, dtype: float64
```

None of the columns seem to have any values that it shouldn't have, everything seems fine.

In [43]:

```
# Checking for dirty values in price
vehicle_data['price'].describe()
```

Out[43]:

```
count     51525.000000
mean      12132.464920
std       10040.803015
min           1.000000
25%        5000.000000
50%        9000.000000
75%       16839.000000
max      375000.000000
Name: price, dtype: float64
```

In [44]:

```
# Checking how many cases of price below $100 there is
vehicle_data[vehicle_data['price'] <= 100]
```

Out[44]:

	price	model_year	model	condition	cylinders	fuel	odometer	transmission	t
405	1	2014.0	chevrolet camaro	3	6.0	gas	71310.0	automatic	co
3063	1	1998.0	chevrolet silverado	2	8.0	gas	164000.0	automatic	pic
3808	1	2007.0	chevrolet tahoe	2	8.0	gas	200.0	automatic	S
3902	1	1996.0	ford f-150	1	8.0	gas	163000.0	manual	tr
4140	1	2004.0	chevrolet silverado	3	8.0	diesel	83000.0	automatic	pic
...	...	...	...	...	...	...	...	...	
50245	1	1986.0	chevrolet silverado	2	8.0	gas	90420.0	automatic	tr
50393	1	2003.0	gmc sierra 2500hd	2	8.0	diesel	212300.0	automatic	pic
50430	5	2011.0	toyota sienna	2	6.0	gas	123025.0	automatic	S
50971	10	2012.0	toyota prius	3	4.0	hybrid	101000.0	automatic	hatchb
51256	1	2012.0	honda civic lx	3	4.0	gas	71262.0	automatic	se

863 rows × 18 columns

In [45]:

```
# Changing the values of prices below $100 to x100
vehicle_data.loc[vehicle_data['price'] <= 100, 'price'] = vehicle_data['price']*100
```

In [46]:

```
# Checking the price column again
vehicle_data['price'].describe()
```

Out[46]:

```
count     51525.000000
mean      12140.555924
std       10033.169588
min         100.000000
25%        5000.000000
50%        9000.000000
75%       16839.000000
max      375000.000000
Name: price, dtype: float64
```

The minimum price was 1 dollar, after checking how many vehicles were advertisied for less than 100 dollar we found out only 18 cases. We can drop these cases or we can change their values to something that makes more sense (it doesn't really matter, it is a very small portion of our data). We decided to multiply all values below 100 by 100, assuming people misconfigurated the prices when filling the details.

With this behind ourbacks, we are left with one last task and it is to check for duplicates, and drop them if there's any:

In [47]:

```
# Check for dups
vehicle_data.duplicated().sum() # checking for duplicates
```

Out[47]:

```
0
```

Our data is very clean and it is ready for analysis!

Reviewer's comment.

So, we are ready for the next part!

1.6 Study core parameters

Below we'll study some parameters using histograms, find possible outliers and see how these outliers affect the form/readability of these histograms.

The parameters are

- Price
- The vehicle's age when the ad was placed
- Mileage
- Number of cylinders
- Condition

In [48]:

```
#
params = {'price' : (0,100000), 'age' : (0,75) , 'odometer' : (0, 500000) , 'mile_per_year'
params2 = {'price' : (0,50000), 'age' : (0,35) , 'odometer' : (0, 300000) , 'mile_per_year'

for param in params:
    ax = vehicle_data.hist(param, figsize=(10,7.5), bins = 20, range=params[param])
    vehicle_data.hist(param, figsize=(10,7.5), bins = 20, range=params2[param], ax=ax)
```

If we are to find any outliers, it is definetly going to be among the columns with the highest range of values, such as **price** or **odometer**. Below we continue to check how much outliers does each column has:

In [49]:

```
#
for param in params2:
    outliers = vehicle_data[vehicle_data[param] > params2[param][1]]
    print(param, ":", outliers.shape[0])
    print(outliers[param].mean(), '\n')
```

```
price : 227
66888.34801762115

age : 233
47.772532188841204

odometer : 287
375554.88850174216

mile_per_year : 115
89988.33913043479

cylinders : 0
nan

condition : 0
nan
```

As expected, cylinders and condition doesn't have any outliers. While on the other hand, price, odometer and mile_per_year have some. Those outliers appear as a small lines far away from the condensed area of values in the histogram, and including them in our plots will only make our histogram less readable and less ideal. Below is our histograms without these outliers:

According to the histograms above, all outliers were an upper limit outlier meaning that all the outliers we spotted were on the very right side of the histograms. The histograms that showed outliers were the histograms of **odometer, price, age & mile_per_year**. The histograms were more pleasnt to read without the outliers (when they were out of range). The outliers don't only affect the histograms, they also impact the mean of a certain column leading into incorrect results. But in our case, the outliers are a very very small portion of our data contributing in a very very small average pull up (for instance around 300 miles for **odometer**, that has average of around 115,000 which is considered insignificant change).

Next we are going to study ads lifetime:

Reviewer's comment

So, we compared old and new data. Good choice of outliries! Now we know more infromation and can move on.

Reviewer's comment

Also we can show 2 graphs nearby or compare old and new data on the one graph. :)

Student answer. Thanks for the tip, now it is easier to compare between each 2 graphs!

(FIXED)

Reviewer's comment. V.2.

Excellent!

1.7 Ads lifetime

Different ads had different lifetime, which is determined by the values of the **days_listed** column. Lets take a look at the describition of this column. Is there a reason behind ads being removed very quickly? or being listed for an abnormally long time?

In [50]:

```
# Checking the days_listed column
vehicle_data['days_listed'].describe()
```

Out[50]:

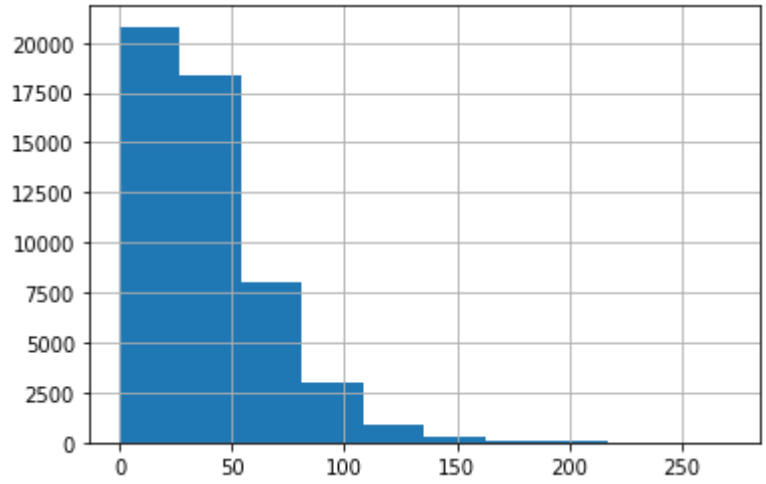
```
count    51525.00000
mean        39.55476
std         28.20427
min          0.00000
25%         19.00000
50%         33.00000
75%         53.00000
max        271.00000
Name: days_listed, dtype: float64
```

In [51]:

```
vehicle_data['days_listed'].hist()
```

Out[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f255fc48b10>
```



Assuming 2-4 days is quick enough and 100+ abnormally long, we continue to check:

In [52]:

```
# We dont have alot of values here
vd_quick_list = vehicle_data.query('days_listed <= 4')
vd_quick_list.shape[0]
```

Out[52]:

```
1308
```

In [53]:

```
# Checking whether vehicle condition has anything to do with how quick an ad was removed
vd_quick_list['condition'].value_counts()
```

Out[53]:

```
3    634
2    500
4    131
1     33
5      8
0      2
Name: condition, dtype: int64
```

In [54]:

```
# Checking if the vehicle's type has anything to do with how quick an ad was removed
vd_quick_list['type'].value_counts()
```

Out[54]:

```
truck          308
sedan          306
SUV            299
pickup         193
wagon           55
coupe           47
hatchback       32
mini-van        32
van             12
convertible     12
offroad          7
other            5
Name: type, dtype: int64
```

In [55]:

```
# Checking whether a vehicle's age had an impact on the ad lifetime.
vd_quick_list['age'].describe()
```

Out[55]:

```
count    1308.000000
mean        9.455275
std         6.056615
min         1.000000
25%         5.000000
50%         8.000000
75%        13.000000
max        56.000000
Name: age, dtype: float64
```

In [56]:

```
# Checking whether vehicle prices has an impact on the ad lifetime
vd_quick_list['price'].describe()
```

Out[56]:

```
count     1308.000000
mean     12488.420489
std       9846.851150
min        100.000000
25%       5000.000000
50%       9814.500000
75%      16995.000000
max      66000.000000
Name: price, dtype: float64
```

In [57]:

```
# Checking whether vehicle odometer has an impact on the ad lifetime
vd_quick_list['odometer'].describe()
```

Out[57]:

```
count      1308.000000
mean     115173.226682
std       66181.998454
min           0.000000
25%       70849.000000
50%      114671.500000
75%      152314.000000
max      990000.000000
Name: odometer, dtype: float64
```

By the data collected above, we can say that the **condition** could've been the reason for those ads to quickly get removed since the majority of these ads are related to vehicles of good or excellent conditions! The rest of the factors don't seem to be a reason for the ads to be removed quickly since they are very close to the total average.

As for the other part of this research, the abnormally long time:

In [58]:

```
vd_slow_list = vehicle_data.query('days_listed >= 100')
vd_slow_list.shape[0]
```

Out[58]:

```
2036
```

In [59]:

```
# Checking whether vehicle condition has anything to do with how quick an ad was removed
vd_slow_list['condition'].value_counts()
```

Out[59]:

```
3    1000
2     785
4     184
1      57
0       6
5       4
Name: condition, dtype: int64
```

In [60]:

```
# Checking if the vehicle's type has anything to do with how quick an ad was removed
vd_slow_list['type'].value_counts()
```

Out[60]:

```
SUV            516
sedan          481
truck          458
pickup         244
coupe           98
wagon           76
mini-van        53
hatchback       41
van             35
convertible     18
offroad         10
other            6
Name: type, dtype: int64
```

In [61]:

```
# Checking whether a vehicle's age had an impact on the ad lifetime.
vd_slow_list['age'].describe()
```

Out[61]:

```
count    2036.000000
mean        9.763016
std         6.337746
min         1.000000
25%         6.000000
50%         9.000000
75%        13.000000
max        56.000000
Name: age, dtype: float64
```

In [62]:

```
# Checking whether vehicle prices has an impact on the ad lifetime
vd_slow_list['price'].describe()
```

Out[62]:

```
count      2036.000000
mean      11989.168468
std       10168.972843
min         100.000000
25%        5146.250000
50%        8998.000000
75%       15995.000000
max      189000.000000
Name: price, dtype: float64
```

In [63]:

```
# Checking whether vehicle odometer has an impact on the ad lifetime
vd_slow_list['odometer'].describe()
```

Out[63]:

```
count      2036.000000
mean     116466.953094
std       60365.889107
min           0.000000
25%       76000.000000
50%      117716.500000
75%      153108.000000
max      500000.000000
Name: odometer, dtype: float64
```

From the results above, there doesn't appear to be a specific reason for why those ads took so long to be removed. The **odometer, price, age** are very close to average, meaning they should take just as long as an average ad for them to get removed. Looking at the **condition** values counts, most of the values are in the good/excellent category, meaning that the condition wasn't the reason behind them taking a long time to get removed. We are left with **type**, with SUV, sedan and truck at the top. Maybe people don't like these types of vehicles? we don't think so, those same types were at the top among the quickest ads as well, so it doesn't seem like it.

Reviewer's comment

Now we have more information about days. Very good that you plot many graphs and tables!

Describe is great variant to show all information.

1.8 Average price per each type of vehicle

Now we are going to analyze the number of ads and the average price for each type of vehicle, to check the dependence of the number of ads on the vehicle type.

In [64]:

```
new_vd_grouped = vehicle_data.pivot_table(index = 'type', values = 'price', aggfunc=['count
new_vd_grouped.columns = ['Count', 'Median', 'Mean']
new_vd_grouped
```

Out[64]:

type	Count	Median	Mean
SUV	12405	8900	11160.836276
truck	12353	14995	16738.693678
sedan	12154	5995	6981.014234
pickup	6988	14200	16058.090441
coupe	2303	12950	14359.418150
wagon	1541	7900	9088.134328
mini-van	1161	6495	8193.347976
hatchback	1047	5980	6873.714422
van	633	7990	10560.235387
convertible	446	12250	14580.764574
other	256	8995	10990.101562
offroad	214	11650	14306.172897
bus	24	10500	17135.666667

In [65]:

```
new_vd_grouped.plot( y = 'Count', kind = 'bar', grid = True, figsize = (10, 5))
```

Out[65]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f255fbcf6d0>
```

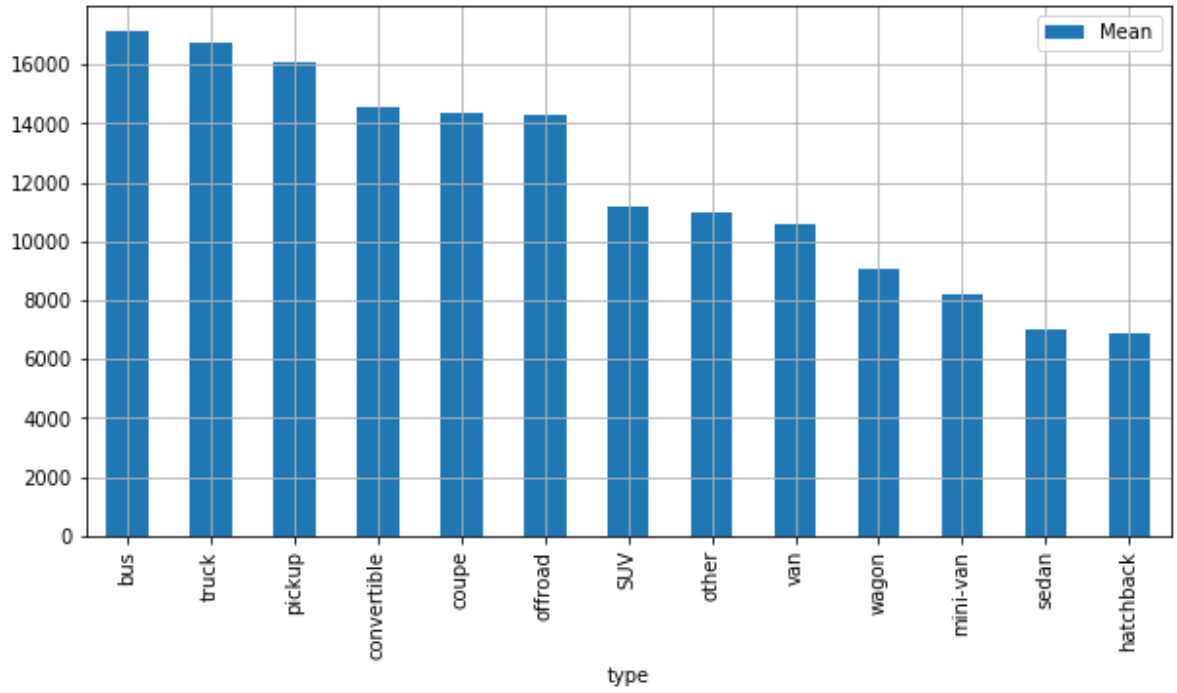
Looking at the histogram, we can see that SUVs and trucks had the majority of ads on the website. Below we have of the average price per each vehicle type:

In [66]:

```
new_vd_grouped.sort_values(ascending = False, by = 'Mean').plot( y = 'Mean', kind = 'bar',
```

Out[66]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f255fac5bd0>
```

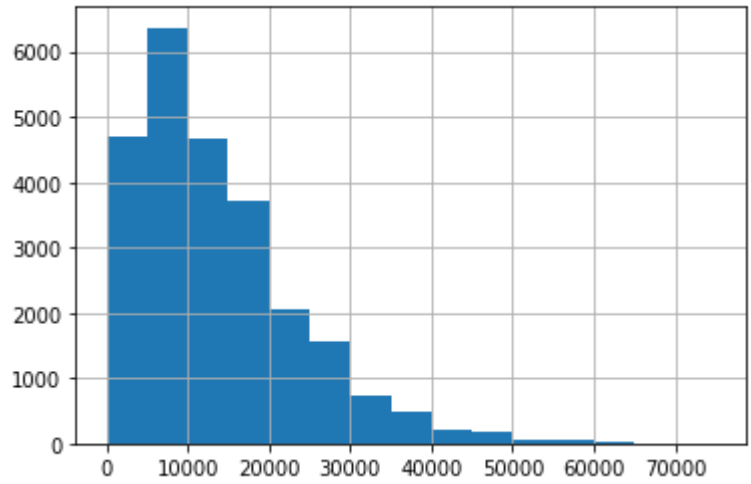


In [67]:

```
# Checking the distribution of the prices on our filtered dataframe
vehicle_data.query('type == "SUV" or type == "truck"')['price'].hist(range=(0,75000), bins
```

Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f255fa12490>
```



Earlier we found out that SUVs and trucks had the most ads among all other types. On the second histogram we see that bus and truck vehicle types had the highest price average. On the third histogram, we check the distribution of the prices among the new filtered dataframe that only includes our most popular vehicle types

and the distribution is pretty similar to the distribution we seen earlier for the whole dataframe.

Next we move on to calculate the impact of factors like age, odomter etc on the price of the vehicle types SUV and truck (top vehicle types with according to number of ads on the website)

~~**Reviewer's comment**~~

~~So, we choosed 2 most popular types. After that we need:~~

- ~~For categorical variables (transmission type and color) plot box-and-whisker charts. For each type. **Note that the categories must have at least 50 ads; otherwise, their parameters won't be valid for analysis.**~~
- ~~For age, mileage, condition create scatterplots. For each type.~~

~~Could you change your code below?~~

Student's comment Thanks for the tips again! I have seperated the graphs for each type and changed the graphs into scatterplots & boxplots and changed my conclusions respectively! Please note that I didn't include any conclusion about paint colors because almost 50% of the data were missing when we started our research. This means, it doesn't matter how much we try to reach for a conclusion here, the chances that it is a wrong conclusion is 50% and that's way too high (The unknown category could include data about any of the

```
(FIXED)
```

1.9 Price factors

In this part of the project, we are going to check for dependencies between the price and the other factors; **age**, **odometer**, **transmission** and **condition**. We need to decide whether those factors really impacted the price of sale. Does higher age mean higher price, or lower price? Does the condition really affect the price as expected? we'd expect to have higher prices for better conditions and so on. We'll find out below!

We'll start with the correlation between price and age:

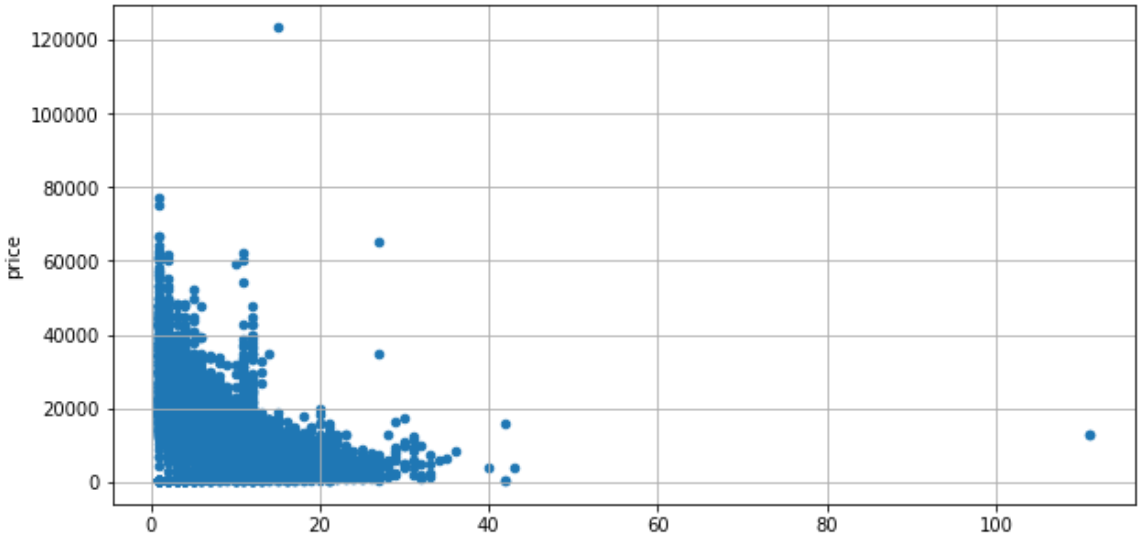
In [68]:

```
# creating a new dataframe that includes only the filtered data (by type)
new_vd_top1 = vehicle_data.query('type == "SUV"')
new_vd_top2 = vehicle_data.query('type == "truck"')

new_vd_top1.plot.scatter(x='age', y = 'price', figsize = (10,5), grid = True)
new_vd_top2.plot.scatter(x='age', y = 'price', figsize = (10,5), grid = True)
```

Out[68]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f255f9586d0>
```



The scatter plots of the age x price are almost perfect. We can see that the older the vehicle is, the lower it's average price becomes and the lower the max price for that age. This isn't displayed accurately in the first plot but in the second plot (which belongs to the 'truck' type) we can clearly see this relation between the age and the price!

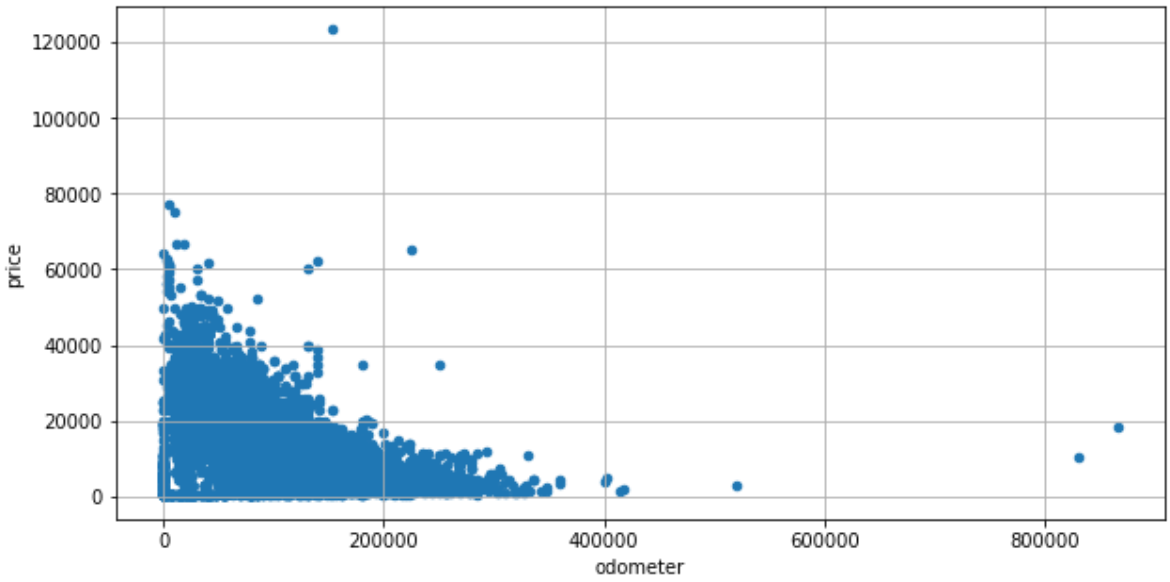
Next we'll investigate odometer x price:

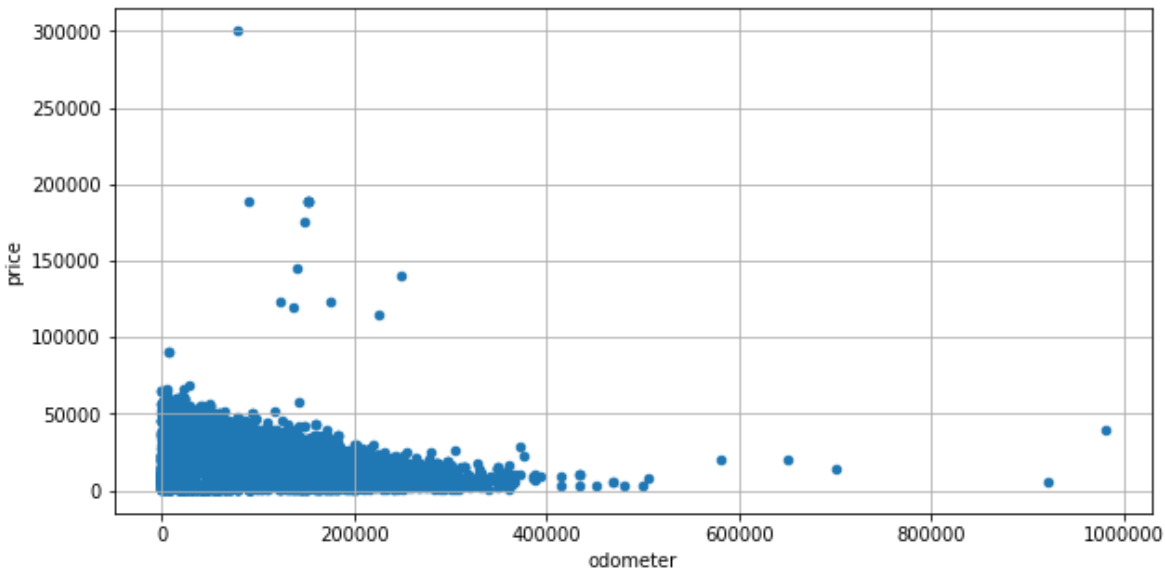
In [69]:

```
new_vd_top1.plot.scatter(x='odometer', y = 'price', figsize = (10,5), grid = True)
new_vd_top2.plot.scatter(x='odometer', y = 'price', figsize = (10,5), grid = True)
```

Out[69]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f255f8ae2d0>
```





Looking at the scatterplot above, we can see that we have a negative correlation between median price and the odometer (The higher the odometer gets, the lower the mean and max price gets) on both vehicle types. This is pretty normal and very reasonable, vehicles with higher odometers tend to have more problems than other vehicles because of how long it was on the streets, therefore people will be less interested in such vehicles and prices drop down!

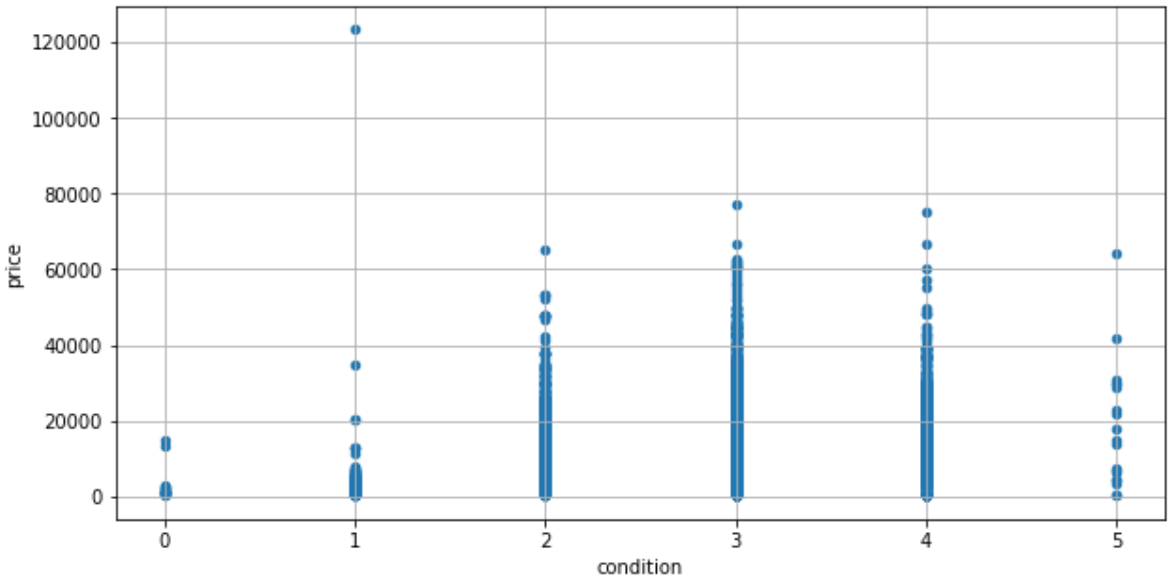
Next we'll investigate the correlation between the price and the condition:

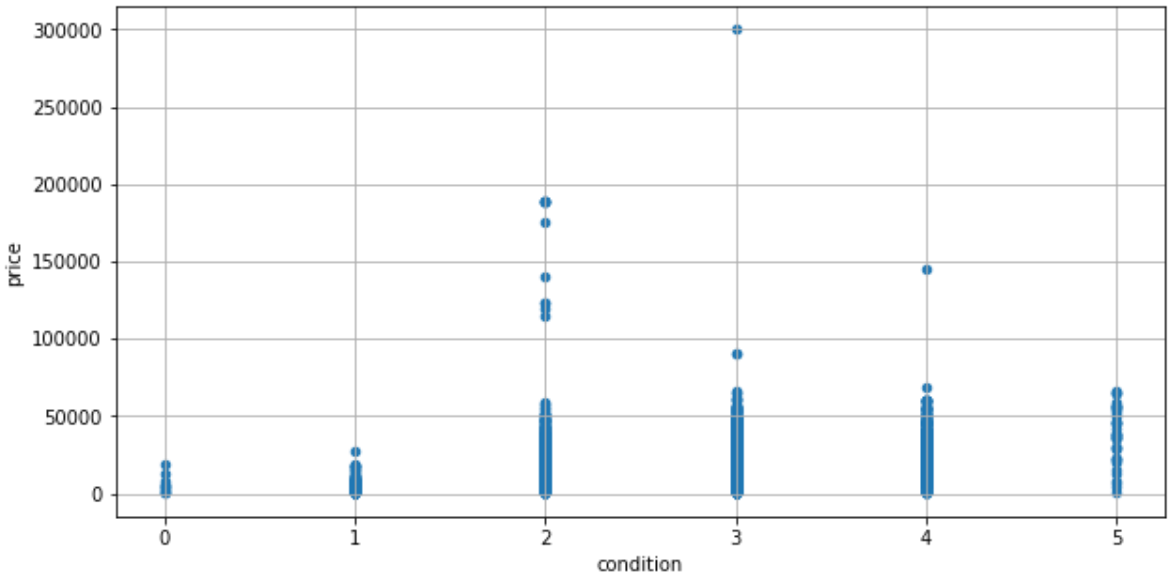
In [70]:

```
new_vd_top1.plot.scatter(x='condition', y = 'price', figsize = (10,5), grid = True)
new_vd_top2.plot.scatter(x='condition', y = 'price', figsize = (10,5), grid = True)
```

Out[70]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f255f8e80d0>
```





The plot above is very reasonable. We'd expect that the better the condition, the higher the price, and that's exactly the case here. We can see higher conditions (3,4 & 5) have the highest average's and maxes on each

type. So there's defenitely a good correlation between the condition and the price!

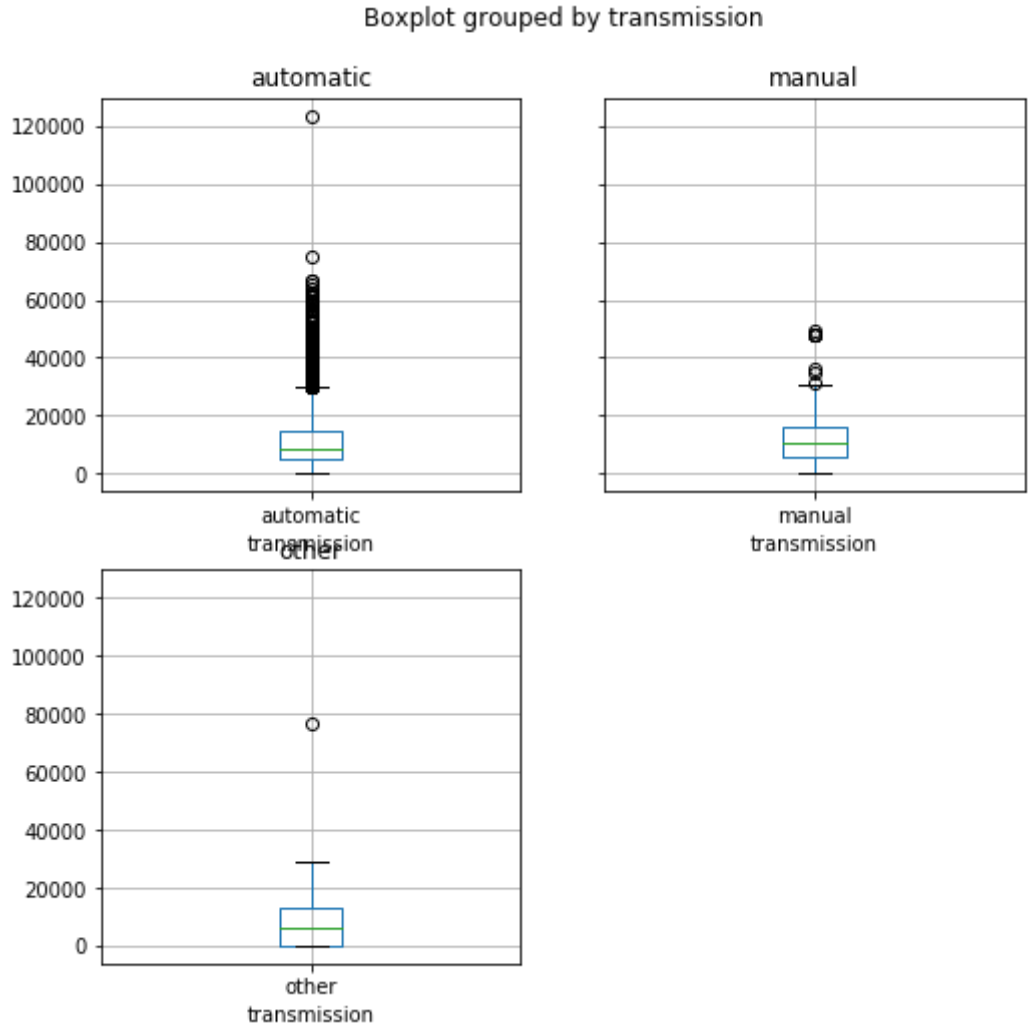
Next we'll check the correlation between price and transmission:

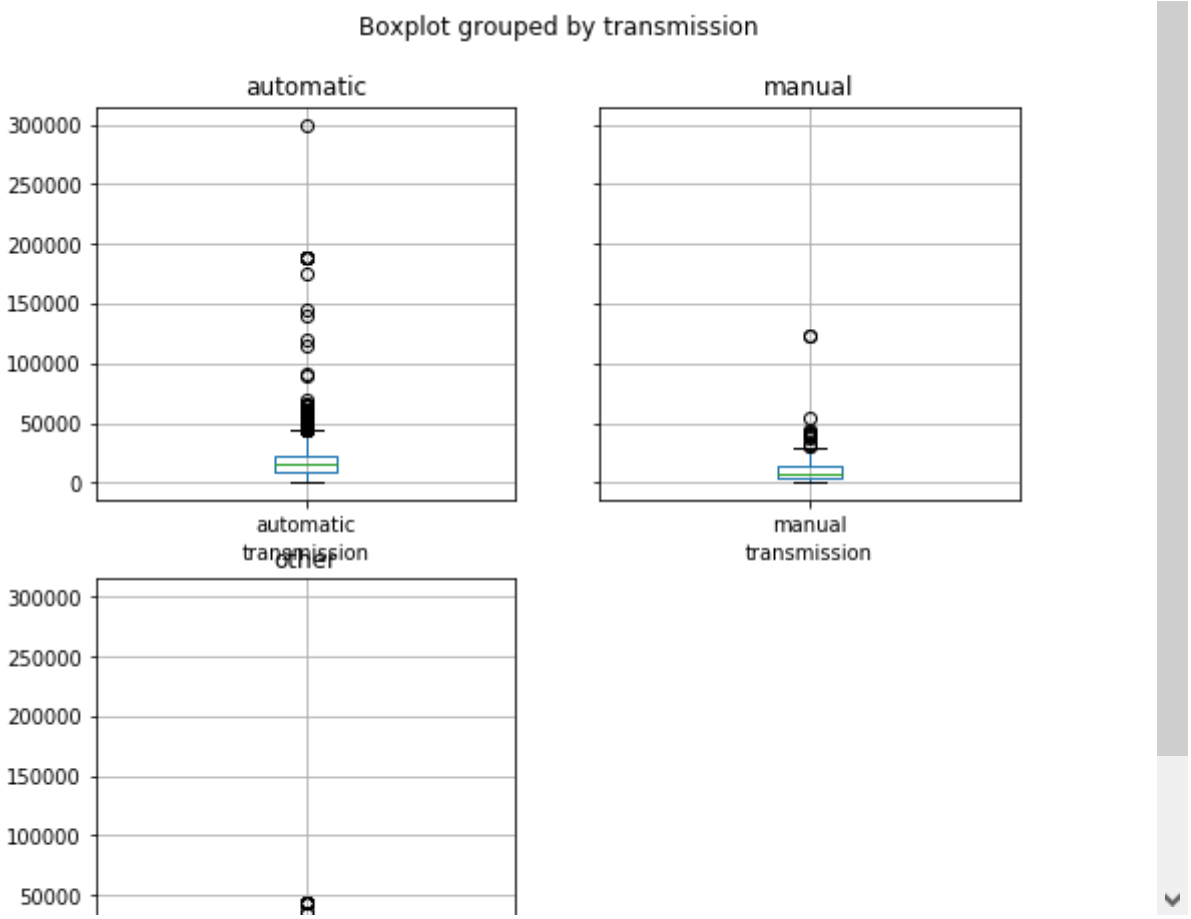
In [71]:

```
new_vd_top1.groupby('transmission').boxplot(by = 'transmission', column = 'price', figsize=
new_vd_top2.groupby('transmission').boxplot(by = 'transmission', column = 'price', figsize=
```

Out[71]:

```
automatic         AxesSubplot(0.1,0.559091;0.363636x0.340909)
manual       AxesSubplot(0.536364,0.559091;0.363636x0.340909)
other                 AxesSubplot(0.1,0.15;0.363636x0.340909)
dtype: object
```





Finally, to check the relation between the transmission and the price we used a 'box' plot, showing us that 'automatic' vehicles had a higher max and average price than the manual transmission and other transmissions on each category. This implies that transmission type has an impact on the price, and automatic vehicles cost more than manual vehicles of the same model/type.

Reviewer's comment. V.2.

We should compare transmissions type on the one graph.And color on the another. I create graphs for SUV. Coud you check it?

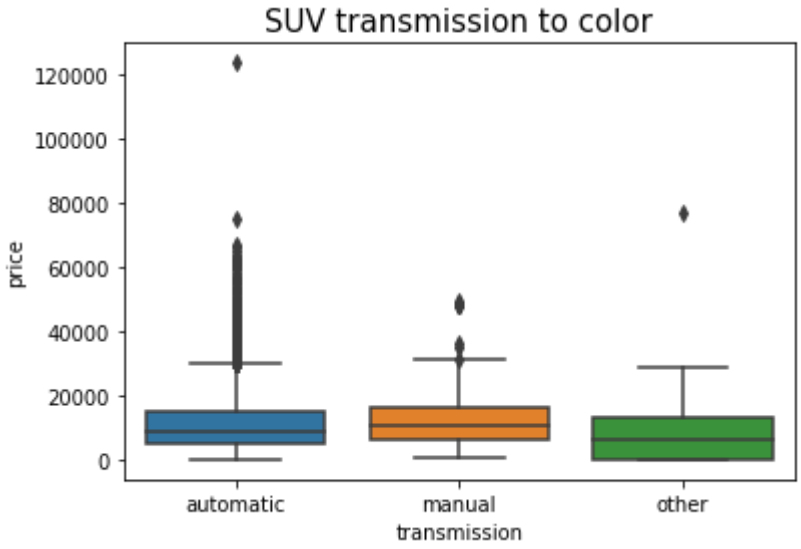
I hope my example will be helpful. :)

In [80]:

```
import seaborn as sns
import matplotlib.pyplot as plt
```

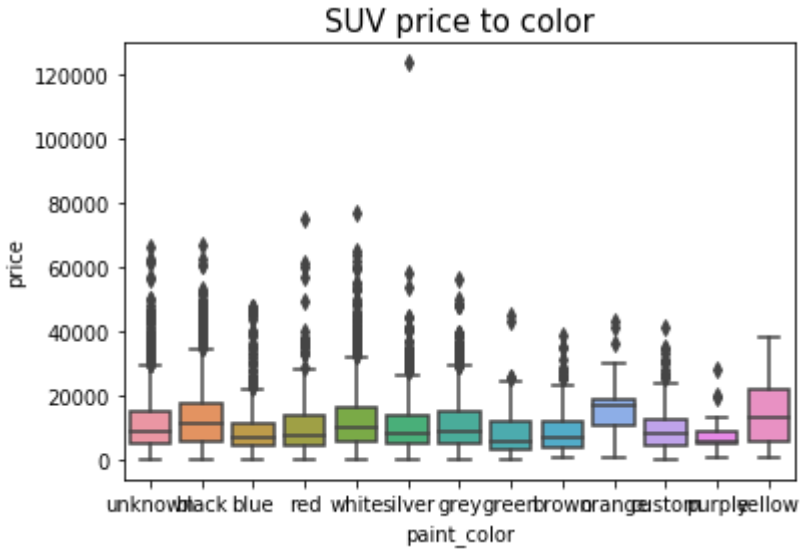
In [82]:

```
sns.boxplot(x ='transmission', y ='price', data = new_vd_top1)
plt.title('SUV transmission to color', fontsize = 15);
```



In [83]:

```
sns.boxplot(x ='paint_color', y ='price', data = new_vd_top1)
plt.title('SUV price to color', fontsize = 15);
```



1.10 General conclusion

To summarize everything in one place: **(1)** We started our investigation with **looking for outliers** by looking at the data and the histograms representing these data. We expected to find outliers among the columns with the greatest ranges of values like **odometer** or **price** or **age** and less or close to nothing on columns like **condition** and **transmission**. We ended up being correct:

price with 227 outliers with the average of 66888.34, **age** with 225 outliers with the average of 47.19, **odometer** with 287 outliers with the average of 375554.88 and **mile_per_year** with 677 outliers with the average of 82173.85. While **cylinders or condition** had 0 outliers.

After checking the histograms with the outliers and without the outliers, we decided that the histograms without the outliers were more accurate and more representitative of our data and the outliers only contributed to an unclear version of the histograms that is harder to read.

(2) Then after we were finished with outliers, we started investigating the lifetimes of ads, some of them were removed quickly and the some actually took a long time to be removed. After studying these ads, we found out that the first part had one thing in common, which is a **Good+** condition, something that could've lead to the quick removal of these ads regardless of the rest of the data. While on the other hand, the second part (ads that lasted long), didn't have much in common, judging by the first part of the investigation, we assumed that vehicles that lasted very long are probably vehicles with bad condition, but that wasn't the case here and most of the ads that last longed were actually ads of vehicles that surprisingly were in a **Good+** condition as well.

(3) Next we started investigating the relation between the vehicles' price and the number of ads to continue our research. Here we found out that the most popular vehicle ads were ads of **SUVs, trucks and sedans**, through the histograms we also came to the conclusion that the most popular vehicle types represent the rest of the dataframe ideally and can be used for our further investigations instead of using the whole dataframe.

(4) Finally we are in the final phase of the research, after all the investigations we have made, we move on to check for dependencies between the price and the rest of the factors. Is there a dependcy? For some of the factors that was a positive, for the rest, not really.

- When we checked for dependcy of price with age we found an interesting result; the newest and the oldest vehicles are the vehicles with the highest prices. And as we explained earlier, this is very reasonable because new cars tend to be good cars with good conditions and old cars tend to be rare which helps the price go up and up. The rest of the ads didn't have any specific order and it was scattered randomly implying that there's no relation between the age of the vehicles they presented and their price.
- When we checked the impact of odometer on the price, we ended up with; The higher the odometer > the lower the price. This means there is a dependncy between those two factors. In my opinion this is very logical as well, since vehicles with higher odometers tend to have more problems due to the time it spent running/on the roads.
- When we checked for dependcy of price with **condition** we found out the pretty obvious: vehicles with better conditions had higher prices. There's nothing to explain here, that's what any reasonable person would've expected and that's what we got. So there is a dependcy between those two factors.
- And finally when we checked for the relation between transmission and vehicles' median price, we ended up with the conclusion that automatic vehicles are more expensive than manual vehicles. This is pretty normal, automatic vehicles have higher prices because they include more components to help automate the gear shifting process thus making the vehicle itself more expensive.
- The paint color column had alot of missing values at the start of the research and we had to fill them with uncertain values instead. Therefore, we decided to exclude this factor from our research. Beside that, paint color shouldn't really be of any impact on the price considering normal paint colors have similar prices.. (unless your painting your vehicle with gold eh?)

To sum up once and for all, what factors really had an impact on the price over the research? Mainly the vehicle condition, odometer and age at some level. Is that what we would've expected before we started? It is safe to

Reviewer's comment

Very good final conclusion! We desribe our results with details. It is informative for clients. They often read only this part. :)