```
This is a project to develop a machine learning model to predict used car prices.  The dataset used
is from craigslist and was obtained from kaggle.
```

In [61]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv("Used Vehicle Prices.csv")
```

In [63]:
```
data
```

Out[63]:

	id	url	region	region_url	price	year	manufac
0	7222695916	https://prescott.craigslist.org/cto/d/prescott...	prescott	https://prescott.craigslist.org	6000	NaN	
1	7218891961	https://fayar.craigslist.org/ctd/d/bentonville...	fayetteville	https://fayar.craigslist.org	11900	NaN	
2	7221797935	https://keys.craigslist.org/cto/d/summerland-k...	florida keys	https://keys.craigslist.org	21000	NaN	
3	7222270760	https://worcester.craigslist.org/cto/d/west-br...	worcester / central MA	https://worcester.craigslist.org	1500	NaN	
4	7210384030	https://greensboro.craigslist.org/cto/d/trinit...	greensboro	https://greensboro.craigslist.org	4900	NaN	
...	...	...	...	...	...	...	
371012	7315083667	https://dallas.craigslist.org/sdf/ctd/d/north-...	dallas / fort worth	https://dallas.craigslist.org	10991	2015.0	d
371013	7315082729	https://dallas.craigslist.org/dal/ctd/d/lewisv...	dallas / fort worth	https://dallas.craigslist.org	0	2019.0	
371014	7315082647	https://dallas.craigslist.org/dal/ctd/d/richar...	dallas / fort worth	https://dallas.craigslist.org	21999	2013.0	
371015	7315082545	https://dallas.craigslist.org/dal/cto/d/irving...	dallas / fort worth	https://dallas.craigslist.org	2100	2008.0	chev
371016	7315082276	https://dallas.craigslist.org/sdf/ctd/d/north-...	dallas / fort worth	https://dallas.craigslist.org	16998	2014.0	a

371017 rows × 28 columns

In [64]:
```
#Get basic information about dataset-shape, name of columns, print first five rows, and number of
#unique values in each of the columns.

print(data.shape)
```

```
(371017, 28)
```

In [65]:
```
print(data.columns)
```

```
Index(['id', 'url', 'region', 'region_url', 'price', 'year', 'manufacturer',
       'model', 'condition', 'cylinders', 'fuel', 'odometer', 'title_status',
       'transmission', 'VIN', 'drive', 'size', 'type', 'paint_color',
       'image_url', 'description', 'county', 'state', 'lat', 'long',
       'posting_date', 'Column1', 'Column2'],
      dtype='object')
```

In [66]:
```
data.head()
```

Out[66]:

	id	url	region	region_url	price	year	manufacturer	m
0	7222695916	https://prescott.craigslist.org/cto/d/prescott...	prescott	https://prescott.craigslist.org	6000	NaN	NaN	
1	7218891961	https://fayar.craigslist.org/ctd/d/bentonville...	fayetteville	https://fayar.craigslist.org	11900	NaN	NaN	
2	7221797935	https://keys.craigslist.org/cto/d/summerland-k...	florida keys	https://keys.craigslist.org	21000	NaN	NaN	
3	7222270760	https://worcester.craigslist.org/cto/d/west-br...	worcester / central MA	https://worcester.craigslist.org	1500	NaN	NaN	
4	7210384030	https://greensboro.craigslist.org/cto/d/trinit...	greensboro	https://greensboro.craigslist.org	4900	NaN	NaN	

5 rows × 28 columns

In [67]:
```
#Check for how many unique values there are in the dataset for each column
#axis=0 counts values by rows, where as axis=1 checks for unique values by columns
data.nunique(axis=0)
```

Out[67]:
```
id              371017
url             371017
region             350
region_url         357
price            14692
year               113
manufacturer        42
model            27082
condition            6
cylinders            8
fuel                 5
odometer         95730
title_status         6
transmission         3
VIN             103960
drive                3
size                 4
type                13
paint_color         12
image_url       212439
description     316080
county               0
state               46
lat              46798
long             47371
posting_date    333516
Column1              0
Column2              0
dtype: int64
```

In [68]:
```python
#Drop columns that don't help with the analysis
data = data.drop(["region_url", "Column1", "Column2",'VIN', 'image_url', 'url', 'county', 'id',
                  'lat', 'long', 'state'], axis = 1)
```

In [69]:
```python
data.shape
```

Out[69]: (371017, 17)

In [70]:
```python
data.describe()
```

Out[70]:

	price	year	odometer
count	3.710170e+05	369936.000000	3.672260e+05
mean	8.363187e+04	2011.158833	9.813305e+04
std	1.306718e+07	9.531748	2.164597e+05
min	0.000000e+00	1900.000000	0.000000e+00
25%	5.900000e+03	2008.000000	3.772500e+04
50%	1.390000e+04	2013.000000	8.600000e+04
75%	2.599100e+04	2017.000000	1.340000e+05
max	3.736929e+09	2022.000000	1.000000e+07

In [72]:
```python
#Get data types of each of the columns
#Data types are all appropriate given the columns of data we're working with
data.dtypes
```

Out[72]:
```
region           object
price             int64
year            float64
manufacturer     object
model            object
condition        object
cylinders        object
fuel             object
odometer        float64
title_status     object
transmission     object
drive            object
size             object
type             object
paint_color      object
description      object
posting_date     object
dtype: object
```

In [73]:
```
#As can be seen, the size column has the largest number of null values.
missing_values = pd.DataFrame({'Null': data.isnull().sum()})
total = len(data)
missing_values_percentage = round((missing_values['Null']/total)*100,1)
missing_values['Percentage'] = missing_values_percentage
missing_values.sort_values(by='Null', ascending=False)
```

Out[73]:

	Null	Percentage
size	265464	71.6
cylinders	153549	41.4
condition	148350	40.0
drive	115007	31.0
paint_color	113090	30.5
type	81264	21.9
manufacturer	15573	4.2
title_status	6820	1.8
model	4689	1.3
odometer	3791	1.0
fuel	2434	0.7
transmission	2197	0.6
year	1081	0.3
description	66	0.0
posting_date	65	0.0
price	0	0.0
region	0	0.0

In [74]:
```
data.duplicated().sum()
```

Out[74]: 16

In [75]:
```
data = data.drop_duplicates(keep = 'first')
```

In [76]:
```
#Check to see if dataset of duplicated values has been deleted
data.duplicated().sum()
```

Out[76]: 0

In [77]:
```
#Get count of unique values in all columns with NaN values
#Columns with less than 14 unique values will be filled with the mode of each respective column, where
#be filled based on text from the description column.
print("Number of Categories in: ")
for ColName in data[['size','cylinders','condition', 'drive', 'paint_color', 'type', 'manufacturer',
                    'odometer', 'fuel', 'transmission', 'year']]:
    print("{} = {}".format(ColName,       len(data[ColName].unique())))
```

```
Number of Categories in:
size = 5
cylinders = 9
condition = 7
drive = 4
paint_color = 13
type = 14
manufacturer = 43
title_status = 7
model = 27083
odometer = 95731
fuel = 6
transmission = 4
year = 114
```

In [78]:
```
data
```

Out[78]:

	region	price	year	manufacturer	model	condition	cylinders	fuel	odometer	title_status	transmission	driv
0	prescott	6000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
1	fayetteville	11900	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2	florida keys	21000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
3	worcester / central MA	1500	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
4	greensboro	4900	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
...	...	...	...	...	...	...	...	...	...	...	...	.
371012	dallas / fort worth	10991	2015.0	dodge	dart	NaN	4 cylinders	gas	46658.0	clean	automatic	fw
371013	dallas / fort worth	0	2019.0	ford	transit	NaN	NaN	gas	NaN	clean	automatic	Nal
371014	dallas / fort worth	21999	2013.0	ram	1500	NaN	NaN	gas	110739.0	clean	automatic	4w
371015	dallas / fort worth	2100	2008.0	chevrolet	cobalt	NaN	NaN	gas	178000.0	rebuilt	automatic	Nal
371016	dallas / fort worth	16998	2014.0	acura	rdx	NaN	6 cylinders	gas	89204.0	clean	automatic	4w

371001 rows × 17 columns

In [79]:
```
#Fill cylinders, condition, drive, title_status, fuel, transmission, paint_color, and type with most
#frequently occuring values
#These columns all have 14 or less unique values
#drop size column because of high number of NaN values
df_names = ['cylinders', 'condition', 'drive', 'title_status', 'fuel', 'transmission', 'paint_color',
df_clean = data.apply(lambda x: x.fillna(x.value_counts().index[0]) if x.name in df_names else x)
df_clean = df_clean.drop(['size'], axis = 1)
```

In [80]:
```
#Extract information from description column to fill empty values in the manufacturer, type, and paint
import re

manufacturer = '(gmc | hyundai | toyota | mitsubishi | ford | chevrolet | ram | buick | jeep | dodge
| honda | chrysler | mini | pontiac | mercedes-benz | cadillac | bmw | kia | volvo | volkswagen | jagu
mercury | lincoln | infiniti | ferrari | fiat | tesla | land rover | harley-davidson | datsun | alfa-r
| hennessey)'
type_ = '(sedan | truck | SUV | mini-van | wagon | hatchback | coupe | pickup | convertible | van | bu
paint_color = '(red | grey | blue | white | custom | silver | brown | black | purple | green | orange
keys =    ['manufacturer','type', 'paint_color']
columns = [ manufacturer, type_, paint_color]

for i,column in zip(keys,columns):
    df_clean[i] = df_clean[i].fillna(
      df_clean['description'].str.extract(column, flags=re.IGNORECASE, expand=False)).str.lower()

df_clean.drop(['description', 'posting_date'], axis=1, inplace= True)
```

In [81]:
```
df_clean
```

Out[81]:

	region	price	year	manufacturer	model	condition	cylinders	fuel	odometer	title_status	transmission	drive
0	prescott	6000	NaN	NaN	NaN	good	6 cylinders	gas	NaN	clean	automatic	4wc
1	fayetteville	11900	NaN	NaN	NaN	good	6 cylinders	gas	NaN	clean	automatic	4wc
2	florida keys	21000	NaN	NaN	NaN	good	6 cylinders	gas	NaN	clean	automatic	4wc
3	worcester / central MA	1500	NaN	NaN	NaN	good	6 cylinders	gas	NaN	clean	automatic	4wc
4	greensboro	4900	NaN	NaN	NaN	good	6 cylinders	gas	NaN	clean	automatic	4wc
...	...	...	...	...	...	...	...	...	...	...	...	..
371012	dallas / fort worth	10991	2015.0	dodge	dart	good	4 cylinders	gas	46658.0	clean	automatic	fwc
371013	dallas / fort worth	0	2019.0	ford	transit	good	6 cylinders	gas	NaN	clean	automatic	4wc
371014	dallas / fort worth	21999	2013.0	ram	1500	good	6 cylinders	gas	110739.0	clean	automatic	4wc
371015	dallas / fort worth	2100	2008.0	chevrolet	cobalt	good	6 cylinders	gas	178000.0	rebuilt	automatic	4wc
371016	dallas / fort worth	16998	2014.0	acura	rdx	good	6 cylinders	gas	89204.0	clean	automatic	4wc

371001 rows × 14 columns

In [82]:
```
#Since manufacturer, model, and odometer all have less than 3% of null values, these null values will
missing_values = pd.DataFrame({'Null': df_clean.isnull().sum()})
total = len(data)
missing_values_percentage = round((missing_values['Null']/total)*100,1)
missing_values['Percentage'] = missing_values_percentage
missing_values.sort_values(by='Null', ascending=False)
```

Out[82]:

	Null	Percentage
manufacturer	10671	2.9
model	4679	1.3
odometer	3781	1.0
year	1071	0.3
region	0	0.0
price	0	0.0
condition	0	0.0
cylinders	0	0.0
fuel	0	0.0
title_status	0	0.0
transmission	0	0.0
drive	0	0.0
type	0	0.0
paint_color	0	0.0

In [83]:
```
#remove remaining null rows from manufacturer, model, odometer, and year columns. Dataset is now clea
df_clean.dropna(axis=0, how='any', inplace=True)
```

In [84]:
```
#Find mode of transmission column.  This will be used to fill the other values in the tranmission col
#they're aren't many unique values for transmission

df_clean.mode(axis=0, numeric_only=False)
```

Out[84]:

	region	price	year	manufacturer	model	condition	cylinders	fuel	odometer	title_status	transmission	drive	type
0	columbus	0	2018.0	ford	f-150	good	6 cylinders	gas	100000.0	clean	automatic	4wd	sedan

In [85]:
```
df_clean['transmission'] = df_clean['transmission'].replace(['other'], 'automatic')
```

In [86]:
```
data2 = df_clean
```

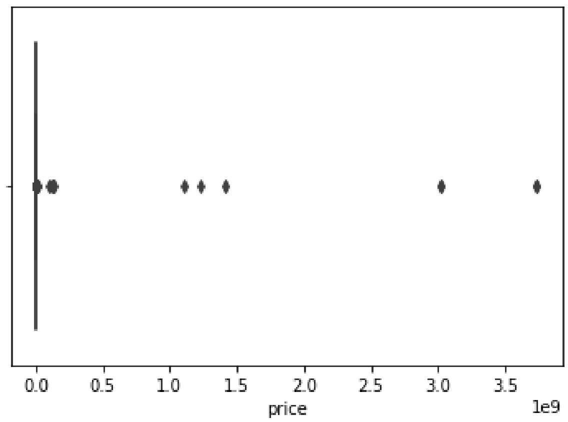
In [87]:
```
#Take care of inconsistent data entry- ex: 'awd' and 'awd '
columns = ['manufacturer', 'condition', 'cylinders', 'fuel', 'title_status', 'transmission', 'drive',
           'type', 'paint_color']
for i in columns:
    data2[i] = data2[i].str.strip()
```

Determine if there are outliers in the dataset and handle for price, odometer, and year by using a boxplot and histogram.

In [88]:
```
sns.boxplot(data2.price)
```

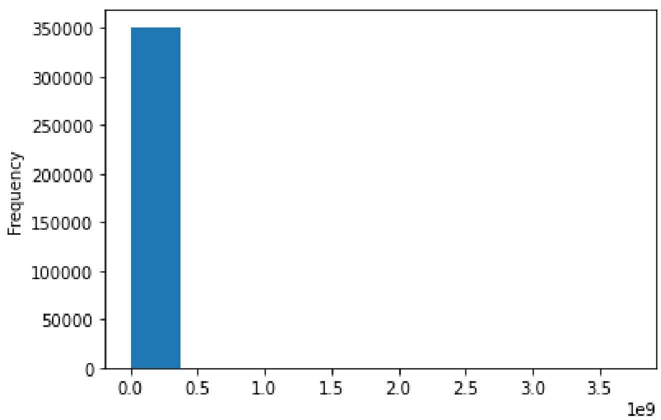
```
C:\Users\office\Anaconda 2022\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the f
ollowing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinterp
retation.
  warnings.warn(
```

Out[88]: <AxesSubplot:xlabel='price'>



In [89]:
```
data2.price.plot.hist()
print(len(data2.price >= 500000000))
```

```
351136
```



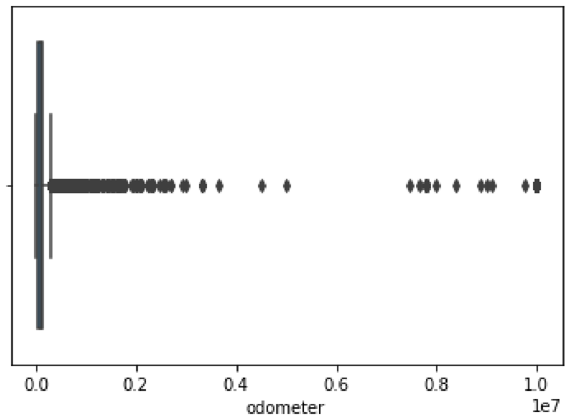
In [90]:
```
#Since I am assumming cars have some value, $0 would skew the analysis.  So I will only consider price
#10th percentile.  Also, since there is a big difference between the 99th and 95th percentile, I will
#of values.
print(data2.price.quantile([0, 0.05, 0.1, 0.25, 0.5, 0.70, 0.75, 0.9, 0.95, 0.99, 1.0]))
data2 = data2[data2.price >= 969].copy()
data2 = data2[data2.price <= 42995].copy()
```

```
0.00    0.000000e+00
0.05    0.000000e+00
0.10    8.000000e+02
0.25    5.995000e+03
0.50    1.399000e+04
0.70    2.299900e+04
0.75    2.599500e+04
0.90    3.697700e+04
0.95    4.299500e+04
0.99    6.399500e+04
1.00    3.736929e+09
Name: price, dtype: float64
```

In [91]:
```
sns.boxplot(data2.odometer)
```

```
C:\Users\office\Anaconda 2022\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the f
ollowing variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinterp
retation.
  warnings.warn(
```

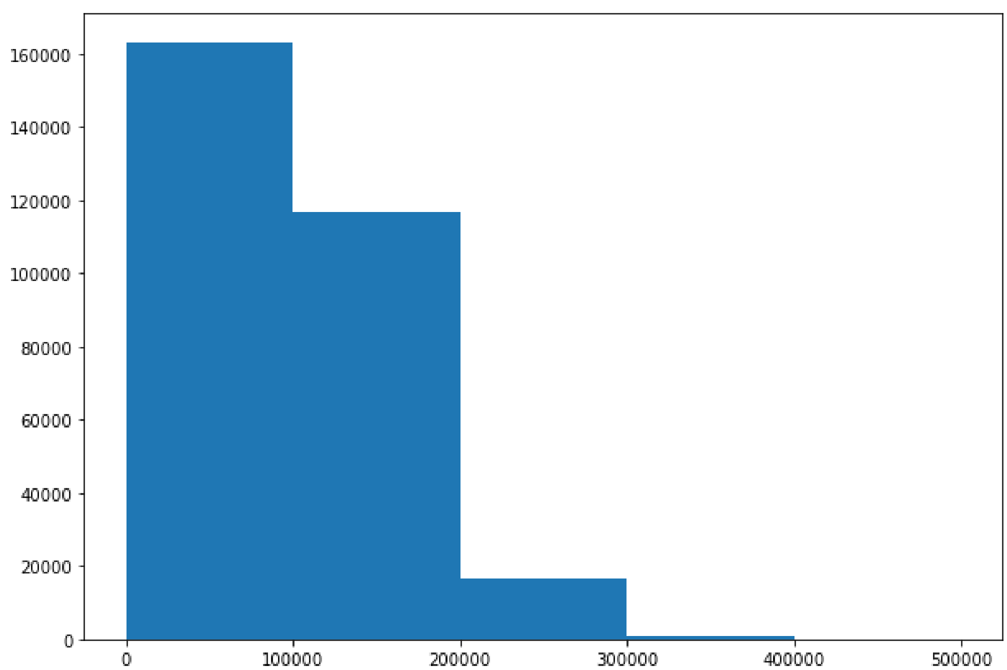
Out[91]:
```
<AxesSubplot:xlabel='odometer'>
```



In [92]:
```
#As can be seen below in the histogram, most values for mileage are less than or equal to 300,000 mile
#300,000 will therefore be deleted, as well as values equal to 0.
print(len(data2[data2.odometer >= 300000]))
fig, ax = plt.subplots(figsize = (10,7))
ax.hist(data2.odometer, bins = [0, 100000, 200000, 300000, 400000, 500000])
```

```
1631
```

Out[92]:
```
(array([1.62968e+05, 1.16630e+05, 1.66360e+04, 1.00200e+03, 1.32000e+02]),
 array([     0, 100000, 200000, 300000, 400000, 500000]),
 <BarContainer object of 5 artists>)
```



In [93]:
```
data2 = data2[data2.odometer < 300000].copy()
data2= data2[data2.odometer > 0].copy()
```

In [94]:
```
data2.count()
```

Out[94]:
```
region          295518
price           295518
year            295518
manufacturer    295518
model           295518
condition       295518
cylinders       295518
fuel            295518
odometer        295518
title_status    295518
transmission    295518
drive           295518
type            295518
paint_color     295518
dtype: int64
```

In [95]:
```
#As we started out with 371,017 values, we have deleted almost 100000 values that are outliers,
#or about 25% of the original data
print(data2.year.quantile([0, 0.05, 0.1, 0.25, 0.5, 0.70, 0.75, 0.9, 0.95, 0.99, 1.0]))
print(len(data2))
```

```
0.00    1900.0
0.05    1999.0
0.10    2003.0
0.25    2008.0
0.50    2013.0
0.70    2016.0
0.75    2017.0
0.90    2018.0
0.95    2019.0
0.99    2020.0
1.00    2022.0
Name: year, dtype: float64
295518
```

In [96]:
```
#Since the typical consumer doesn't buy antique cars, I will cut off the bottom 5% of cars, and only c
#cars with a year 1999 and newer

data2 = data2[data2.year >= 1999].copy()
len(data2)
```

Out[96]:
```
281594
```

In [97]:
```
#I will also combine condition and title status into one column since they are related
data2['status'] = data2['condition'] + '&' + data2['title_status']
data2.drop(['condition', 'title_status'], axis=1, inplace=True)
```

In [98]:
```
#I will also combine manufacturer and model
data2['manufacturer'] = data2['manufacturer'] + ' ' + data2['model']
data2.drop(['model'], axis=1, inplace=True)
data2
```

	region	price	year	manufacturer	cylinders	fuel	odometer	transmission	drive	type	paint_color	s
27	auburn	33590	2014.0	gmc sierra 1500 crew cab slt	8 cylinders	gas	57923.0	automatic	4wd	pickup	white	good8
28	auburn	22590	2010.0	chevrolet silverado 1500	8 cylinders	gas	71229.0	automatic	4wd	pickup	blue	good8
29	auburn	39590	2020.0	chevrolet silverado 1500 crew	8 cylinders	gas	19160.0	automatic	4wd	pickup	red	good8
30	auburn	30990	2017.0	toyota tundra double cab sr	8 cylinders	gas	41124.0	automatic	4wd	pickup	red	good8
31	auburn	15000	2013.0	ford f-150 xlt	6 cylinders	gas	128000.0	automatic	rwd	truck	black	excellent8
...	...	...	...	...	...	...	...	...	...	...	...	
371011	dallas / fort	22922	2013.0	chevrolet	8	gas	72215.0	automatic	rwd	coupe	custom	good8

Categorical Variable Encoding and Random Forest Regression Model

In [99]:
```
#Export dataframe to excel before doing categorical variable encoding. Spreadsheet available in files
data2.to_excel("used_car_data2.xlsx")
```

In [100]:
```
#Label Encoding is used by assigning each categorical value within each categorical variable a number,
#variables can be plugged into the machine learning algorithm. The number values of price, year, and o


from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
cat_features = ['region', 'manufacturer', 'cylinders', 'fuel', 'transmission', 'drive', 'type', 'pain
encoder=LabelEncoder()
encoded=data2[cat_features].apply(encoder.fit_transform)
data2.drop(cat_features, axis=1, inplace=True)
data2=pd.concat([encoded, data2], axis=1)
data2
```

	region	manufacturer	cylinders	fuel	transmission	drive	type	paint_color	status	price	year	odometer
27	15	7972	6	2	0	0	7	10	12	33590	2014.0	57923.0
28	15	3301	6	2	0	0	7	1	12	22590	2010.0	71229.0
29	15	3339	6	2	0	0	7	8	12	39590	2020.0	19160.0
30	15	16874	6	2	0	0	7	8	12	30990	2017.0	41124.0
31	15	5977	5	2	0	2	10	0	0	15000	2013.0	128000.0
...	...	...	...	...	...	...	...	...	...	...	...	...
371011	69	2607	6	2	0	2	2	3	12	22922	2013.0	72215.0
371012	69	4581	3	2	0	1	8	8	12	10991	2015.0	46658.0
371014	69	14238	5	2	0	0	7	0	12	21999	2013.0	110739.0
371015	69	2712	5	2	0	0	8	10	16	2100	2008.0	178000.0
371016	69	101	5	2	0	0	9	8	12	16998	2014.0	89204.0

281594 rows × 12 columns

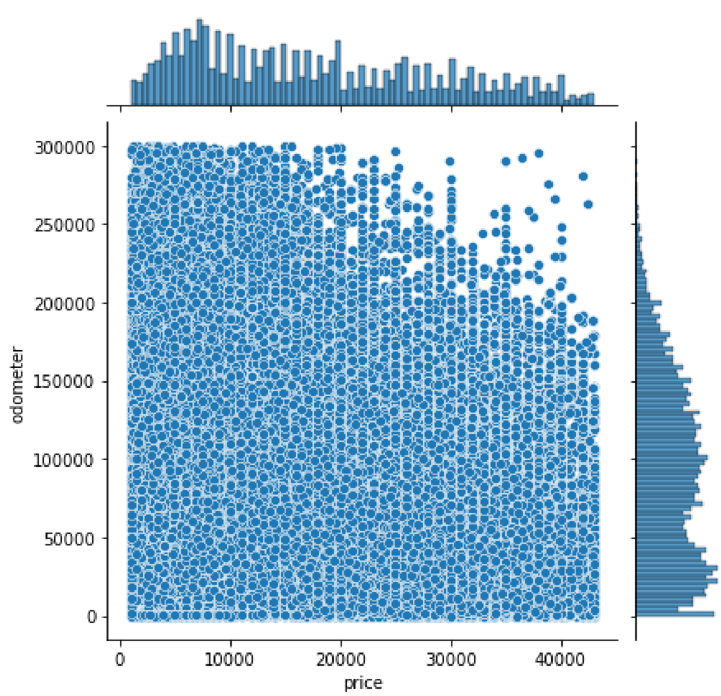
In [101]:
```
corr_matrix = data2[['price', 'year', 'odometer']].corr()
sn.heatmap(corr_matrix, annot=True)
plt.show()
```

Perform bivariate analysis to see how the price is affected by the odometer reading and the year. The data is too scattered to draw any sort of conclusion about correlation between price and odometer (see below). Possibly segmenting the dataset into smaller groups may help with the analysis, such as type of vehicle.

In [102]:
```
sns.jointplot(data2.price, data2.odometer)
```

```
C:\Users\office\Anaconda 2022\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the f
ollowing variables as keyword args: x, y. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result in an error or misint
erpretation.
  warnings.warn(
```

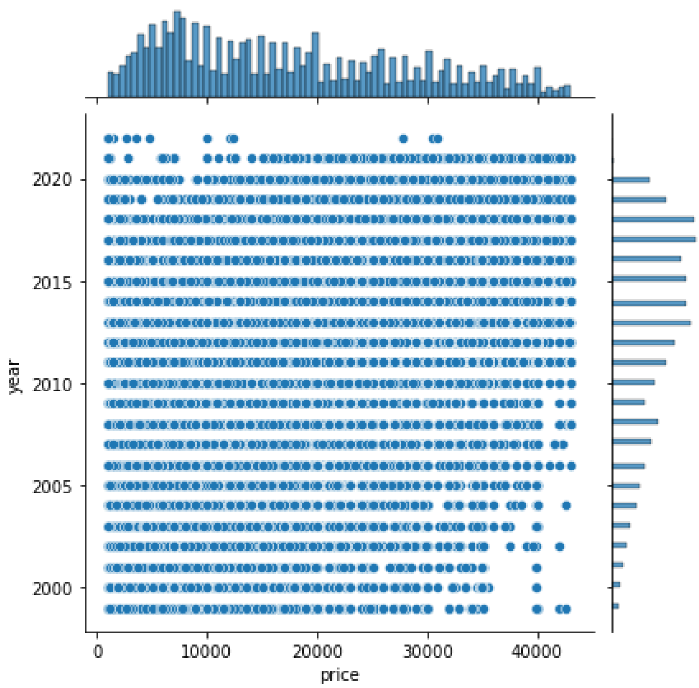
Out[102]: <seaborn.axisgrid.JointGrid at 0x1dc0c42de20>



In [103]:
```
#Again, no conclusion can be drawn
sns.jointplot(data2.price, data2.year)
```

```
C:\Users\office\Anaconda 2022\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the f
ollowing variables as keyword args: x, y. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result in an error or misint
erpretation.
  warnings.warn(
```

Out[103]: <seaborn.axisgrid.JointGrid at 0x1dc2d1580d0>



In [104]:
```
#Change the order of the dataframe to make price last, so it's easy to separate the dataframe into x a
data2 = data2[['region', 'manufacturer', 'cylinders', 'fuel', 'transmission', 'drive', 'type', 'paint_
             'odometer', 'price']]
data2
```

Out[104]:

	region	manufacturer	cylinders	fuel	transmission	drive	type	paint_color	status	year	odometer	price
27	15	7972	6	2	0	0	7	10	12	2014.0	57923.0	33590
28	15	3301	6	2	0	0	7	1	12	2010.0	71229.0	22590
29	15	3339	6	2	0	0	7	8	12	2020.0	19160.0	39590
30	15	16874	6	2	0	0	7	8	12	2017.0	41124.0	30990
31	15	5977	5	2	0	2	10	0	0	2013.0	128000.0	15000
...	...	...	...	...	...	...	...	...	...	...	...	...
371011	69	2607	6	2	0	2	2	3	12	2013.0	72215.0	22922
371012	69	4581	3	2	0	1	8	8	12	2015.0	46658.0	10991
371014	69	14238	5	2	0	0	7	0	12	2013.0	110739.0	21999
371015	69	2712	5	2	0	0	8	10	16	2008.0	178000.0	2100
371016	69	101	5	2	0	0	9	8	12	2014.0	89204.0	16998

281594 rows × 12 columns

In [105]:
```
x=data2.iloc[:, :-1]
y=data2.iloc[:, -1:]

#convert target variable columns to array
#y = np.array(y)
#x = np.array(x)
print(x)
print(y)
```

```
        region  manufacturer  cylinders  fuel  transmission  drive  type  \
27          15          7972          6     2             0      0     7
28          15          3301          6     2             0      0     7
29          15          3339          6     2             0      0     7
30          15         16874          6     2             0      0     7
31          15          5977          5     2             0      2    10
...        ...           ...        ...   ...           ...    ...   ...
371011      69          2607          6     2             0      2     2
371012      69          4581          3     2             0      1     8
371014      69         14238          5     2             0      0     7
371015      69          2712          5     2             0      0     8
371016      69           101          5     2             0      0     9

        paint_color  status    year  odometer
27               10      12  2014.0   57923.0
28                1      12  2010.0   71229.0
29                8      12  2020.0   19160.0
30                8      12  2017.0   41124.0
31                0       0  2013.0  128000.0
...             ...     ...     ...       ...
371011            3      12  2013.0   72215.0
371012            8      12  2015.0   46658.0
371014            0      12  2013.0  110739.0
371015           10      16  2008.0  178000.0
371016            8      12  2014.0   89204.0

[281594 rows x 11 columns]
        price
27      33590
28      22590
29      39590
30      30990
31      15000
...       ...
371011  22922
371012  10991
371014  21999
371015   2100
371016  16998

[281594 rows x 1 columns]
```

In [106]:
```
#Split the data into test and training sets
from sklearn.model_selection import train_test_split

train_x, test_x, train_y, test_y = train_test_split(x, y, test_size = 0.2, random_state=0)

#Verify data split appropriately
print('Train X Shape: ', train_x.shape)
print('Test X Shape: ', test_x.shape)
print('Train Y Shape: ', train_y.shape)
print('Test Y Shape: ', test_y.shape)
```

```
Train X Shape:  (225275, 11)
Test X Shape:  (56319, 11)
Train Y Shape:  (225275, 1)
Test Y Shape:  (56319, 1)
```

In [107]:
```
#convert train_y to 1-D array using ravel()
train_y = np.ravel(train_y)
```

In [108]:
```
#Find R^2 value to access model accuracy
#Two different n_estimator values were used to access the model accuracy-50 & 100
#The 100 n_estimator was slightly more accurate than 50 (90.82% vs. 90.67%)

from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor

scores=[]

# Instantiate model with 100 decision trees
randfor = RandomForestRegressor(n_estimators = 100, random_state = 42, max_features="auto")
acc=cross_val_score(randfor, train_x, train_y, scoring='r2', cv=5)
scores.append(round(acc.mean()*100,2))
```

In [109]:
```
#R^2 represents the goodness of the fit of the random forest model we created. This means the model is
#can be considered good

results=pd.DataFrame({
    'Metrics' : ['R2'],
    'Accuracy': scores})
results
```

Out[109]:

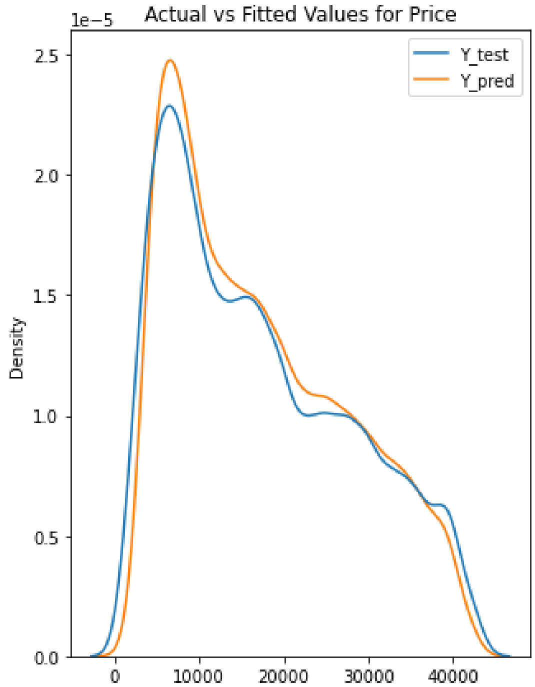
	Metrics	Accuracy
0	R2	92.28

In [110]:
```
#Compare predicted values to test_y values
randfor.fit(train_x, train_y)
y_pred = randfor.predict(test_x)
#y_pred = y_pred.reshape(56788,1)

test_y['Y_pred'] = y_pred.tolist()

#rename column price to y_test
test_y.rename(columns = {'price': 'Y_test'}, inplace = True)
test_y
```

Out[110]:

	Y_test	Y_pred
297962	22990	23064.66
278304	19990	20050.00
362494	10250	14080.46
6328	20991	20818.87
196834	6568	6577.32
...	...	...
49370	21990	21990.00
214832	19590	19416.10
193896	19322	18462.84
251239	32995	29022.29
267466	18000	21106.69

In [130]:
```
#For a visual to access accuracy, the y_pred and test_y will be visualized together
#As can be seen from the plot, the model is pretty accurate
#The least accurate data price point is around 7k, where there is the biggest gap between the graphs
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(5, 7))
sns.kdeplot(data=test_y)
plt.title('Actual vs Fitted Values for Price')
plt.show()
```



x_new1 is an array in which each variable entered is used to predict the target variable (price). These values are all numeric since the these values were encoded to be able to be used for the machine learning algorithm.

Any values can be used in the randfor.predict() function, assuming the categorical variables have values entered that are valid numeric values for each variable.

randfor.pred() will predict the price of the data entered within 92% accuracy (see R^2 value).

In [132]:
```
#x_new1 is an actual value in the dataset. The actual price was $33,590.  So the predicted value of $3

x_new1 = [[15, 7746, 6, 2, 0, 0, 7, 10, 12, 2014, 57923]]

#x_new1_encoded = [['auburn', 'gmc sierra 1500 crew cab slt', '8 cylinders', 'gas', 'automatic', '4wd
# 'good&clean', '33590', '2014', '57923']]

randfor.predict(x_new1)
```

```
C:\Users\office\Anaconda 2022\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have va
lid feature names, but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

Out[132]:
```
array([32874.08])
```