Used_Vehicle_Data_Preprocessing

December 5, 2021

Author: Ekin Ugurel

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
# Read raw data
df = pd.read_csv('C:/Users/ekino/Downloads/vehicles.csv')
```

[3]:
```
df.columns
```

[3]:
```
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'],
      dtype='object')
```

[4]:
```
# Statistical Analysis to get a better understanding of database
df.describe()
```

[4]:
```
                 id         price           year      odometer  county  \
count  4.268800e+05  4.268800e+05  425675.000000  4.224800e+05     0.0
mean   7.311487e+09  7.519903e+04    2011.235191  9.804333e+04     NaN
std    4.473170e+06  1.218228e+07       9.452120  2.138815e+05     NaN
min    7.207408e+09  0.000000e+00    1900.000000  0.000000e+00     NaN
25%    7.308143e+09  5.900000e+03    2008.000000  3.770400e+04     NaN
50%    7.312621e+09  1.395000e+04    2013.000000  8.554800e+04     NaN
75%    7.315254e+09  2.648575e+04    2017.000000  1.335425e+05     NaN
max    7.317101e+09  3.736929e+09    2022.000000  1.000000e+07     NaN

                 lat           long
count  420331.000000  420331.000000
mean       38.493940     -94.748599
std         5.841533      18.365462
min       -84.122245    -159.827728
25%        34.601900    -111.939847
```

```
50%         39.150100     -88.432600
75%         42.398900     -80.832039
max         82.390818     173.885502
```

[5]:
```
# Check for missing values
null_count = pd.DataFrame({'Null': df.isnull().sum()})
# Check for percent of values missing
length=len(df)
percent_null = round((null_count['Null']/length)*100,1)
null_count['Percentage'] = percent_null
# Sort from highest percentage to lowest
null_count.sort_values(by='Null', ascending=False)
```

[5]:

	Null	Percentage
county	426880	100.0
size	306361	71.8
cylinders	177678	41.6
condition	174104	40.8
VIN	161042	37.7
drive	130567	30.6
paint_color	130203	30.5
type	92858	21.8
manufacturer	17646	4.1
title_status	8242	1.9
lat	6549	1.5
long	6549	1.5
model	5277	1.2
odometer	4400	1.0
fuel	3013	0.7
transmission	2556	0.6
year	1205	0.3
description	70	0.0
image_url	68	0.0
posting_date	68	0.0
url	0	0.0
price	0	0.0
state	0	0.0
region_url	0	0.0
region	0	0.0
id	0	0.0

[6]:
```
# Drop columns that have too many missing values or that are irrelevant
df.drop(['posting_date',␣
↪'county','VIN','url','region_url','image_url','id','lat','long','description'],␣
↪axis=1, inplace=True)
df.shape
```

[6]: (426880, 16)

[7]:
```
# Check for duplicates
df.duplicated().sum()
```

[7]: 56415

[8]:
```
# Drop duplicates and keep one of each
df = df.drop_duplicates(keep='first')
```

[9]:
```
# remove inconsistent data entry (e.g. spaces in the cell)
columns = ['manufacturer',␣
↪'condition','cylinders','fuel','title_status','transmission','drive','size','type','paint_c

for i in columns:
    df[i] = df[i].str.strip()
```

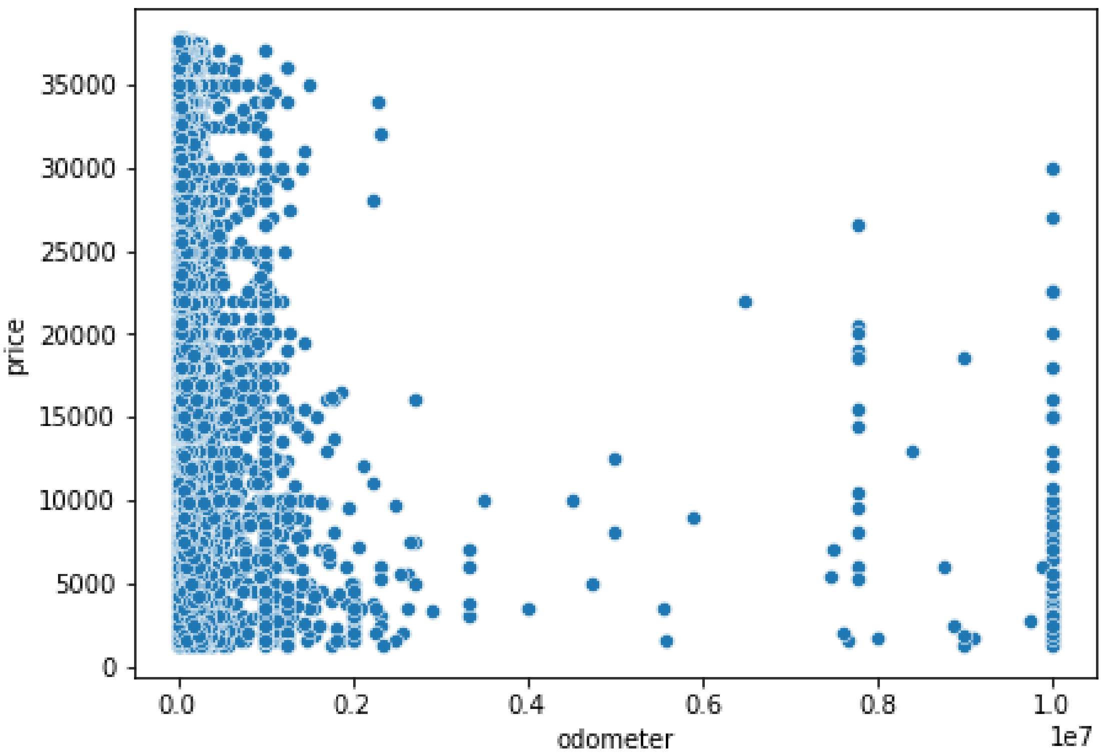
[11]:
```
# Drop 10% of each side on price (outliers)
sort = sorted(df['price'])
q1, q2 = np.percentile(sort, [10,90])
print(q1, q2)
```

```
1200.0 37740.0
```

[12]:
```
df = df[(df.price <= 37740.0) & (df.price >= 1200)]
df.shape
```

[12]: (296798, 16)

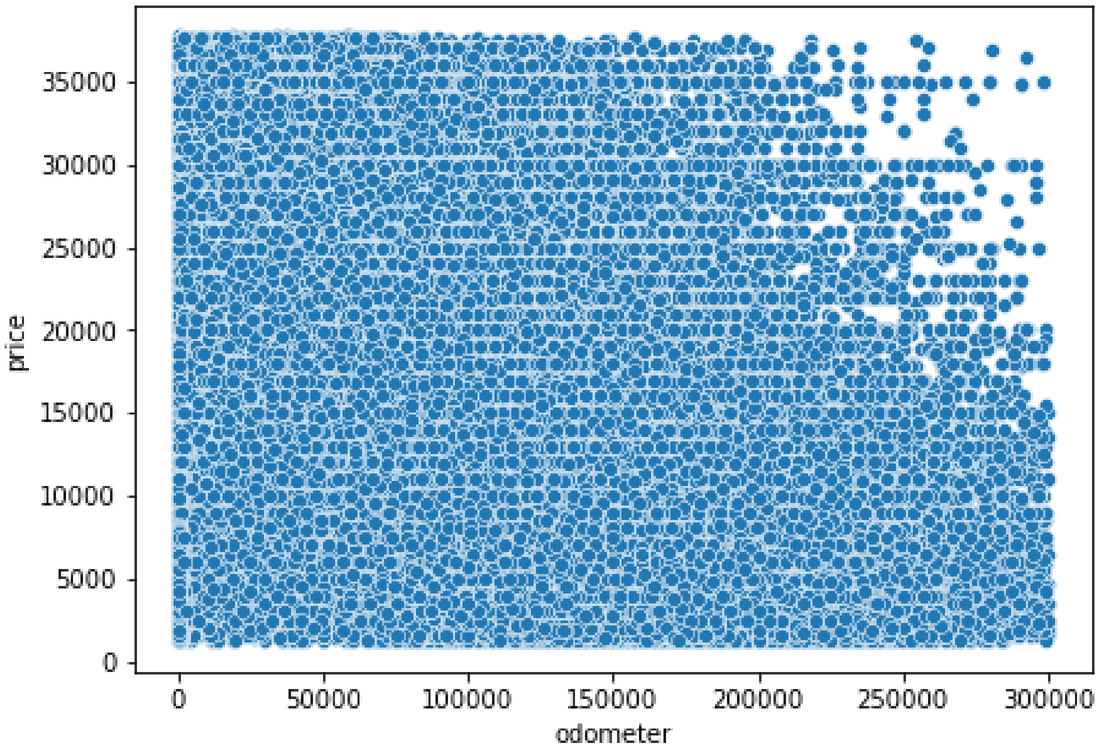
[14]:
```
# Visualize odometer distribution to check for outliers
plt.figure(figsize=[7,5])
odo = sns.scatterplot(x = df['odometer'], y=df['price'])
```



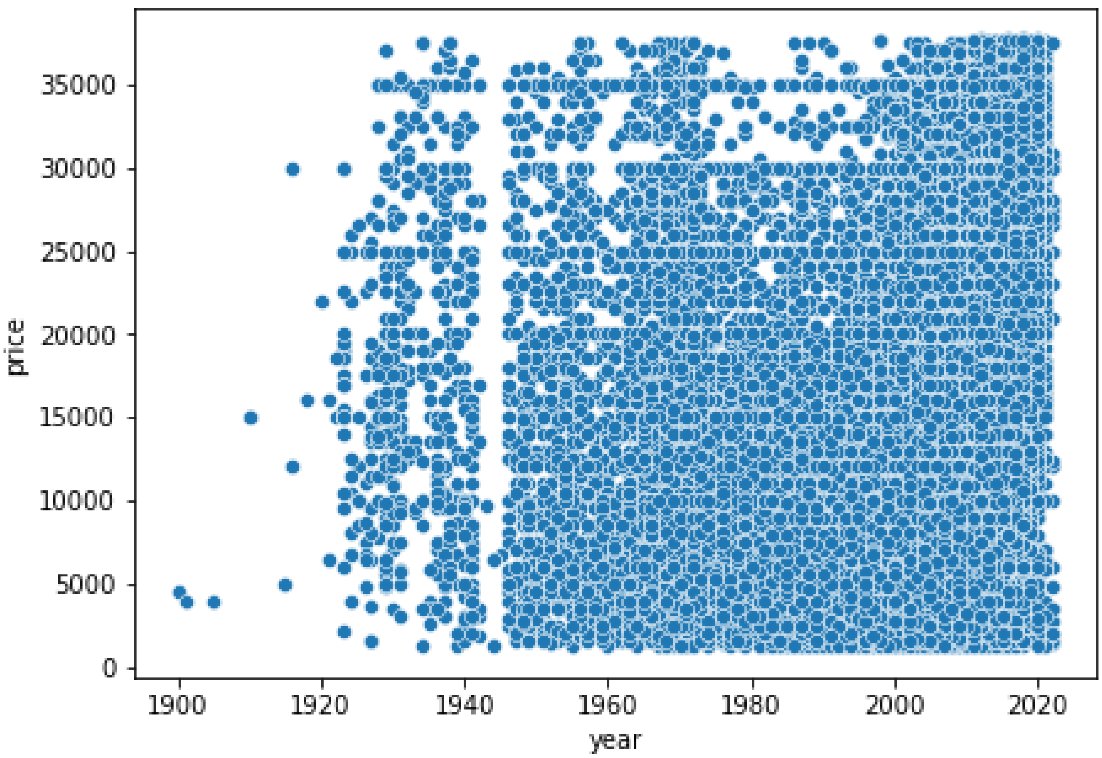
[15]:
```
# Since cars after 300,000 miles of use become almost obsolete, drop all values␣
↪greater than 3e5
df = df[(df.odometer < 3e5)]
```

[16]:
```
# Also drop odometers that equal 0, as we are concerned with used cars (not new)
df.drop(df[df['odometer']==0.0].index, inplace=True)
```

[17]:
```
plt.figure(figsize=[7,5])
odo = sns.scatterplot(x = df['odometer'], y=df['price'])
```

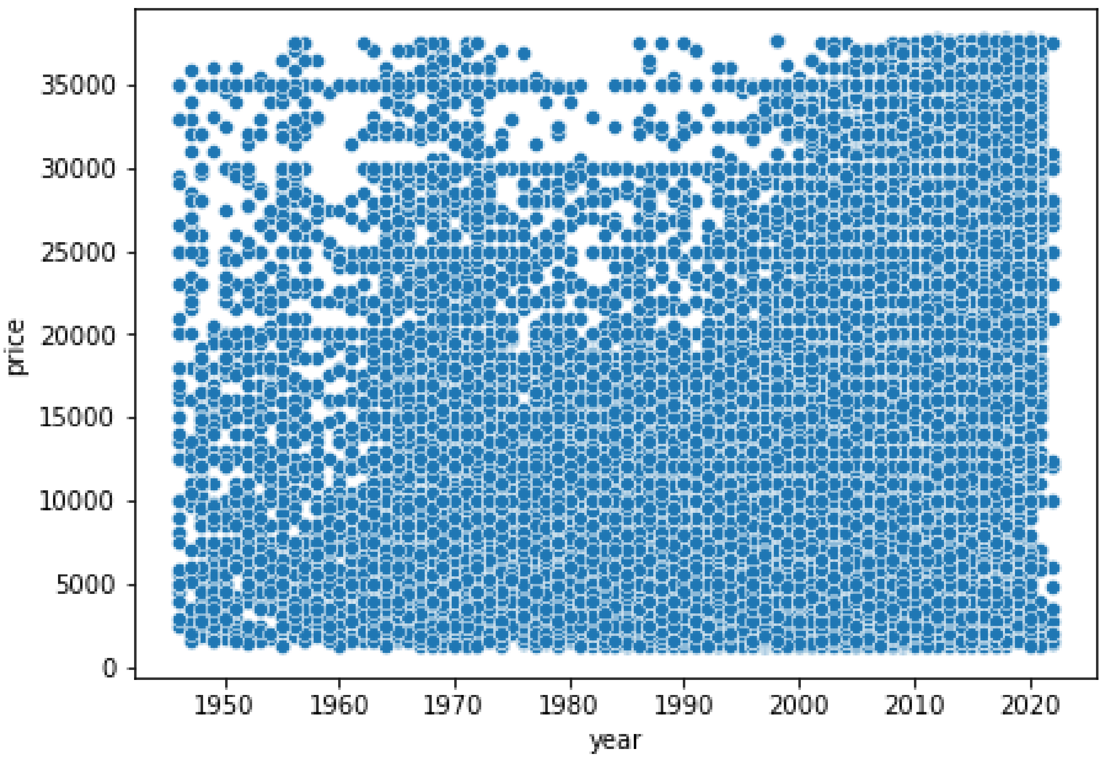


[18]:
```
# handle outliers for "year"
plt.figure(figsize=[7,5])
odo = sns.scatterplot(x = df['year'], y=df['price'])
```



[19]:
```
# Since outliers begin (roughly) before the year 1945, drop all such entries
df.drop(df[df['year'] <= 1945].index, inplace=True)
```

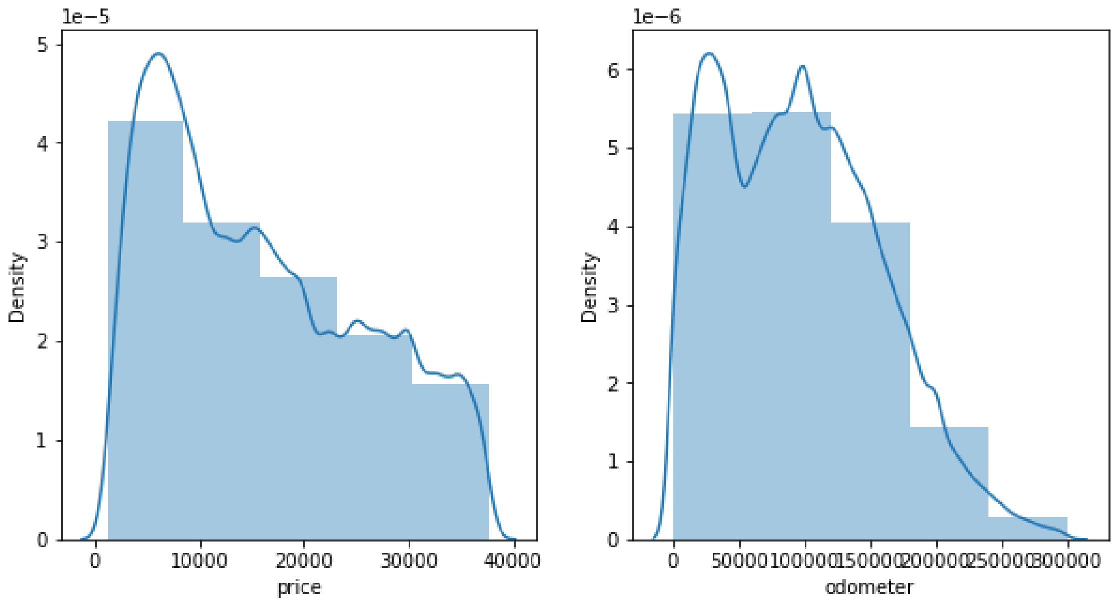
[20]:
```
# Check the scatter plot again
plt.figure(figsize=[7,5])
odo = sns.scatterplot(x = df['year'], y=df['price'])
```



[21]:
```
plt.figure(figsize=[10,5])
plt.subplot(121)
sns.distplot(df['price'], bins = 5)
plt.subplot(122)
sns.distplot(df['odometer'], bins=5)
```

```
C:\Users\ekino\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\ekino\anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[21]:
```
<AxesSubplot:xlabel='odometer', ylabel='Density'>
```




```
[22]: # Fill in NaN values for columns "condition" and "title status"
      bins = [0,30000,60000,90000,115000,150000,1000000]
      groups = df.groupby(['title_status', pd.cut(df.odometer,bins)])
      groups.size().unstack()
```

```
[22]: odometer      (0, 30000]  (30000, 60000]  (60000, 90000]  (90000, 115000]  \
      title_status
      clean              47388           42932           42568            39304
      lien                 127             166             261              192
      missing              130              53              63               85
      parts only            14              12               7               11
      rebuilt              981            1393            1424              898
      salvage              429             535             594              446

      odometer      (115000, 150000]  (150000, 1000000]
      title_status
      clean                    47848              55974
      lien                       210                204
      missing                     58                 78
      parts only                  11                 23
      rebuilt                    896                716
      salvage                    542                730
```

```
[23]: # Since an overwhelming majority of the entries have "clean" titles, this␣
       ↪column becomes insignificant. So we drop this as well.
      df.drop(['title_status'], axis = 1, inplace=True)
```

[24]:
```
# Now, let's check the distribution for condition
bins = [0,30000,60000,90000,115000,150000,1000000]
groups = df.groupby(['condition', pd.cut(df.odometer,bins)])
groups.size().unstack()
```

[24]:
```
odometer   (0, 30000]  (30000, 60000]  (60000, 90000]  (90000, 115000]  \
condition
excellent        4606            8272           13169            14048
fair              252             188             348              583
good            30487           20022           12201             8393
like new         2719            2670            2692             2374
new               233              69              82               53
salvage            29              40              38               50

odometer   (115000, 150000]  (150000, 1000000]
condition
excellent             16917              15101
fair                    948               2916
good                  10928              17649
like new               2563               1839
new                      75                 74
salvage                  63                135
```

[25]:
```
# Replace missing entries with the median condition for each odometer level
g1 = (df['odometer'] > 60000) & (df['odometer'] <= 150000)
g2 = (df['odometer'] <= 60000) | (df['odometer'] > 150000)

df.loc[g1,'condition']=df.loc[g1,'condition'].fillna('excellent')
df.loc[g2,'condition']=df.loc[g2,'condition'].fillna('good')
```

[26]:
```
# Check for missing values again
null_count = pd.DataFrame({'Null': df.isnull().sum()})
# Check for percent of values missing
length=len(df)
percent_null = round((null_count['Null']/length)*100,1)
null_count['Percentage'] = percent_null
# Sort from highest percentage to lowest
null_count.sort_values(by='Null', ascending=False)
```

[26]:
```
                Null  Percentage
size          203732        69.8
cylinders     115143        39.5
drive          88694        30.4
paint_color    80567        27.6
type           62264        21.3
manufacturer   10519         3.6
model           3076         1.1
```

```
fuel              1578         0.5
transmission      1078         0.4
year               471         0.2
region               0         0.0
price                0         0.0
condition            0         0.0
odometer             0         0.0
state                0         0.0
```

[27]:
```python
# Drop 'size' column, as it has too many missing values
df.drop('size', axis=1, inplace=True)
# For columns with less than 4% missing values, drop the empty rows
df = df.dropna(subset=['manufacturer','model','fuel','transmission','year'])
```

[28]:
```python
# Use "ffill" to propagate non-null values forward or backward for columns with␣
↪10-30% missing values
columns = ['drive','type','paint_color']
for i in columns:
    df[i] = df[i].fillna(method='ffill')
```

```
C:\Users\ekino\AppData\Local\Temp/ipykernel_52636/4242946631.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[i] = df[i].fillna(method='ffill')
```

[29]:
```python
# Drop the empty rows if there are any left
df = df.dropna(subset=['type'])
df = df.dropna(subset=['drive'])
```

[30]:
```python
# Check for missing values again
null_count = pd.DataFrame({'Null': df.isnull().sum()})
# Check for percent of values missing
total=len(df)
percent_null = round((null_count['Null']/total)*100,1)
null_count['Percentage'] = percent_null
# Sort from highest percentage to lowest
null_count.sort_values(by='Null', ascending=False)
```

[30]:
```
                Null  Percentage
cylinders     107802        39.1
region             0         0.0
price              0         0.0
year               0         0.0
manufacturer       0         0.0
```

```
model                 0          0.0
condition             0          0.0
fuel                  0          0.0
odometer              0          0.0
transmission          0          0.0
drive                 0          0.0
type                  0          0.0
paint_color           0          0.0
state                 0          0.0
```

[31]:

```
# Only column left to deal with is 'cylinders'.
# We can use the 'drive' column to make best-guesses about the missing values␣
 ↪in 'cylinder'
df.groupby(['drive','cylinders']).cylinders.count()
```

[31]:

```
drive  cylinders
4wd    10 cylinders      371
       12 cylinders        8
       3 cylinders       108
       4 cylinders     15361
       5 cylinders       335
       6 cylinders     30062
       8 cylinders     23839
       other             177
fwd    10 cylinders       38
       12 cylinders        4
       3 cylinders       247
       4 cylinders     35256
       5 cylinders       791
       6 cylinders     21727
       8 cylinders      3031
       other             245
rwd    10 cylinders      464
       12 cylinders       52
       3 cylinders        25
       4 cylinders      5031
       5 cylinders       134
       6 cylinders     13983
       8 cylinders     16506
       other             157
Name: cylinders, dtype: int64
```

[32]:

```
# Fill in the median value of "cylinders" for each type of "drive"
values = {'4wd': '6 cylinders', 'fwd': '4 cylinders', 'rwd': '8 cylinders'}
df.loc[df['cylinders'].isna(), 'cylinders'] = df.loc[df['cylinders'].
 ↪isna(),'drive'].map(lambda x: values[x])
```

[33]:
```
# Check for missing values again
null_count = pd.DataFrame({'Null': df.isnull().sum()})
# Check for percent of values missing
total=len(df)
percent_null = round((null_count['Null']/total)*100,1)
null_count['Percentage'] = percent_null
# Sort from highest percentage to lowest
null_count.sort_values(by='Null', ascending=False)
```

[33]:

	Null	Percentage
region	0	0.0
price	0	0.0
year	0	0.0
manufacturer	0	0.0
model	0	0.0
condition	0	0.0
cylinders	0	0.0
fuel	0	0.0
odometer	0	0.0
transmission	0	0.0
drive	0	0.0
type	0	0.0
paint_color	0	0.0
state	0	0.0

[34]:
```
# Since 'region' and 'state' are directly related to one another, join these␣
↪two columns
df['region'] = df['region'] + ' (' + df['state'] + ')'
df.drop(['state'], axis = 1, inplace=True)
```

[35]:
```
df.shape
```

[35]: (275754, 13)

[40]:
```
# change the order of columns such that 'price' is last
columns = ['region', 'year', 'manufacturer', 'model', 'condition',
                'cylinders','fuel','odometer',␣
↪'transmission','drive','type','paint_color','price']
df = df.reindex(columns=columns)
df.head(1)
```

[40]:

```
         region    year manufacturer      model  condition    cylinders fuel  \
31  auburn (al)  2013.0         ford  f-150 xlt  excellent  6 cylinders  gas

    odometer transmission drive   type paint_color  price
31  128000.0    automatic   rwd  truck       black  15000
```

[37]:
```
df.describe()
```

[37]:

	year	odometer	price
count	275754.000000	275754.000000	275754.000000
mean	2010.922514	96569.286777	16157.680465
std	8.243936	61415.561154	10013.385017
min	1946.000000	1.000000	1200.000000
25%	2008.000000	43213.000000	7450.000000
50%	2013.000000	93000.000000	14500.000000
75%	2016.000000	139594.000000	23999.000000
max	2022.000000	299999.000000	37740.000000

[41]:
```
# This dataset looks good. We will download this as a CSV and use it in our␣
↪analysis.
df.to_csv('preprocessedusedcars.csv')
```

[42]:
```
# However, we also want to check if there are highly correlated features using␣
↪a Correlation Matrix.
# If so, we would drop one of these features so that our analysis isn't␣
↪redundant
df2 = df.copy() # make a copy of dataframe
```

[44]:
```
# In order to check for correlation, we must convert all "categorical" features␣
↪into numerical features
# We use the LabelEncoder function from sklearn to do this
from sklearn.preprocessing import LabelEncoder
categories = ['region', 'manufacturer', 'model', 'cylinders', 'fuel',␣
↪'transmission',
                'drive', 'type', 'paint_color', 'condition']
encoder = LabelEncoder()
encoded = df[categories].apply(encoder.fit_transform)
df2.drop(categories, axis=1, inplace=True)
df2 = pd.concat([encoded, df2], axis=1)

df2.head(10)
```

[44]:

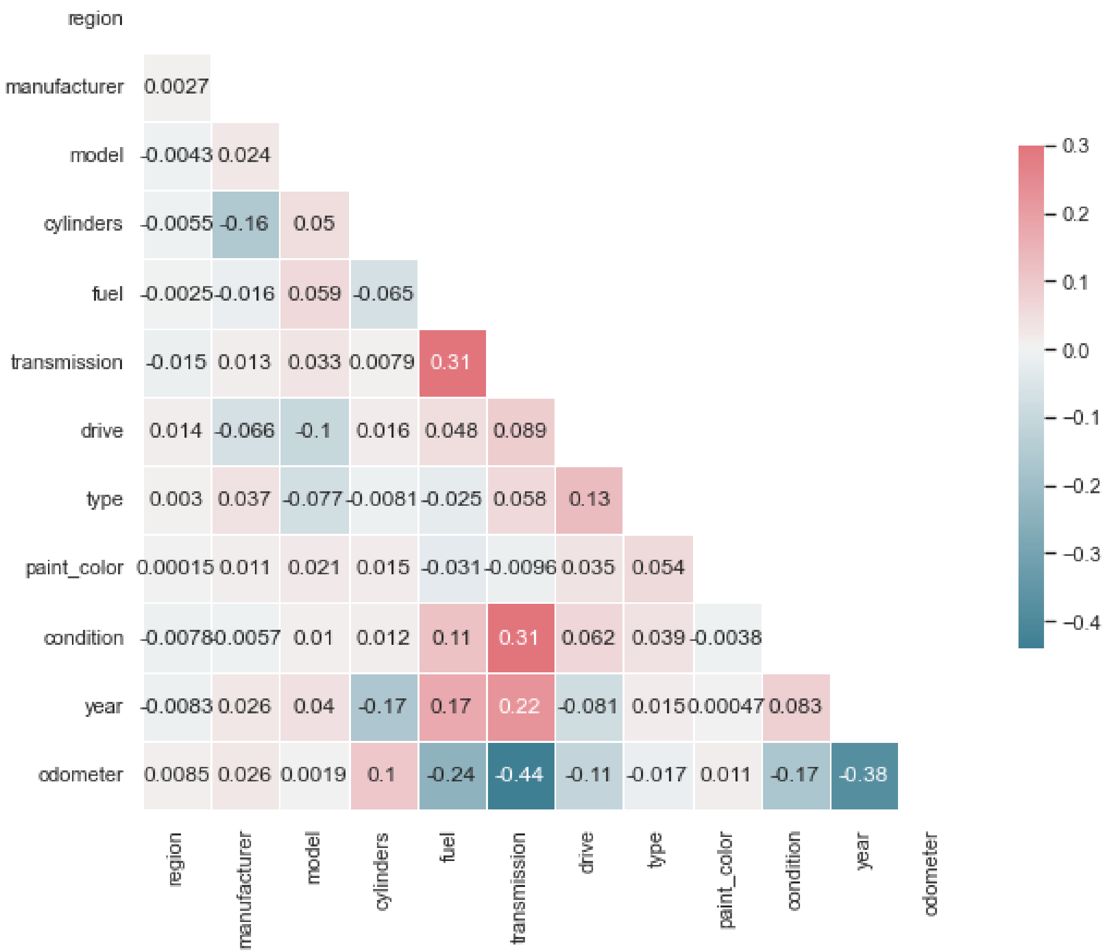
	region	manufacturer	model	cylinders	fuel	transmission	drive	type	\
31	18	13	7661	5	2	0	2	10	
32	18	14	15026	6	2	2	0	8	
33	18	7	15203	5	2	2	0	8	
34	18	38	16252	5	2	0	0	10	
35	18	7	4875	5	2	2	0	8	
37	18	20	4381	5	2	0	0	8	
38	18	20	17954	5	2	2	0	7	
39	18	7	15257	5	2	2	0	8	
40	18	7	4871	5	4	2	0	8	
41	18	38	16289	5	4	2	0	8	

```
    paint_color  condition    year  odometer  price
31            0          0  2013.0  128000.0  15000
32            0          2  2012.0   68696.0  27990
33            9          2  2016.0   29499.0  34590
34            5          0  2019.0   43000.0  35000
35            8          2  2016.0   17302.0  29990
37            8          0  1992.0  192000.0   4500
38            9          2  2017.0   30041.0  32990
39           10          2  2017.0   40784.0  24590
40            1          2  2016.0   34940.0  30990
41            8          2  2014.0   17805.0  27990
```

[52]:

```
# Correlation Heat Map
sns.set(style='whitegrid')
cor = df2.drop(columns=['price']).corr()
mask = np.zeros_like(corr,dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(10,10))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(cor, mask=mask, cmap=cmap, vmax=.3, center=0, square=True,␣
 ↪linewidths=.5, cbar_kws={"shrink":.5}, annot=True)
```

[52]: <AxesSubplot:>



[53]:
```
# No features are highly correlated, so we keep the set as is.
df2.to_csv('numericalpreprocesseddata.csv')
```