Data Analysis and Machine-Learning

Chapter 10.4.

ML Modelling Applications (3)



*Persistence Guarantees that results are inevitable.*

*- Paramahansa Yogananda*

1. Introduction

So far we have covered the algorithmic concepts and mathematical calculations regarding various ML models, including Linear Models, Generalized Linear Models, Support Vector Machine, K-Nearest Neighbor, Decision-Tree Model, Random Forest Model, etc. We have also covered various data-science techniques for effective analytics and modelling, including feature selection methods (FS, BE, SS), Multicollinearity/Variance Inflation Factor processing, Variable Weight Evaluation Methodologies (e.g., eli5, Permutation Importance, Shap), scaling, stochastic modelling, regularizations (L1, L2), etc.

In other words, we are now equipped with the necessary skillsets to perform actual predictions using data. In actual applications, EDA (Exploratory Data Analysis), descriptive statistics of variables, visualizations, and preprocessing consist a big part in prior to the actual modelling process, as deeply learning about the data per se is essential to generate an effective model with best accuracy. Focusing on such aspects together with the purpose of demonstrating various modelling methodologies, this chapter will be focusing on various applications of ML analysis & models via data coding. In this subchapter in particular, we will be predicting the used car prices using various types of structured and unstructured data.

2. EDA (Exploratory Data Analysis) and Visualizations of Variables

#Import Essential Libraries

import warnings

warnings.filterwarnings('always')

warnings.filterwarnings('ignore')

import numpy as np

import pandas as pd

from IPython.display import display

pd.options.display.max\_columns = 10000

pd.options.display.max\_colwidth = 1000

pd.options.display.max\_rows = 10000

pd.options.display.precision = 15

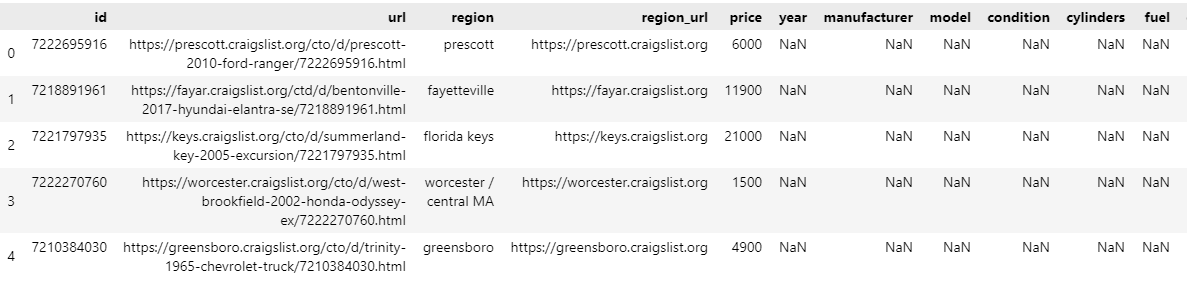
import matplotlib.pyplot as plt

import seaborn as sns

#Importing Data

rawData = pd.read\_csv('Your File Path\\vehicles.csv')

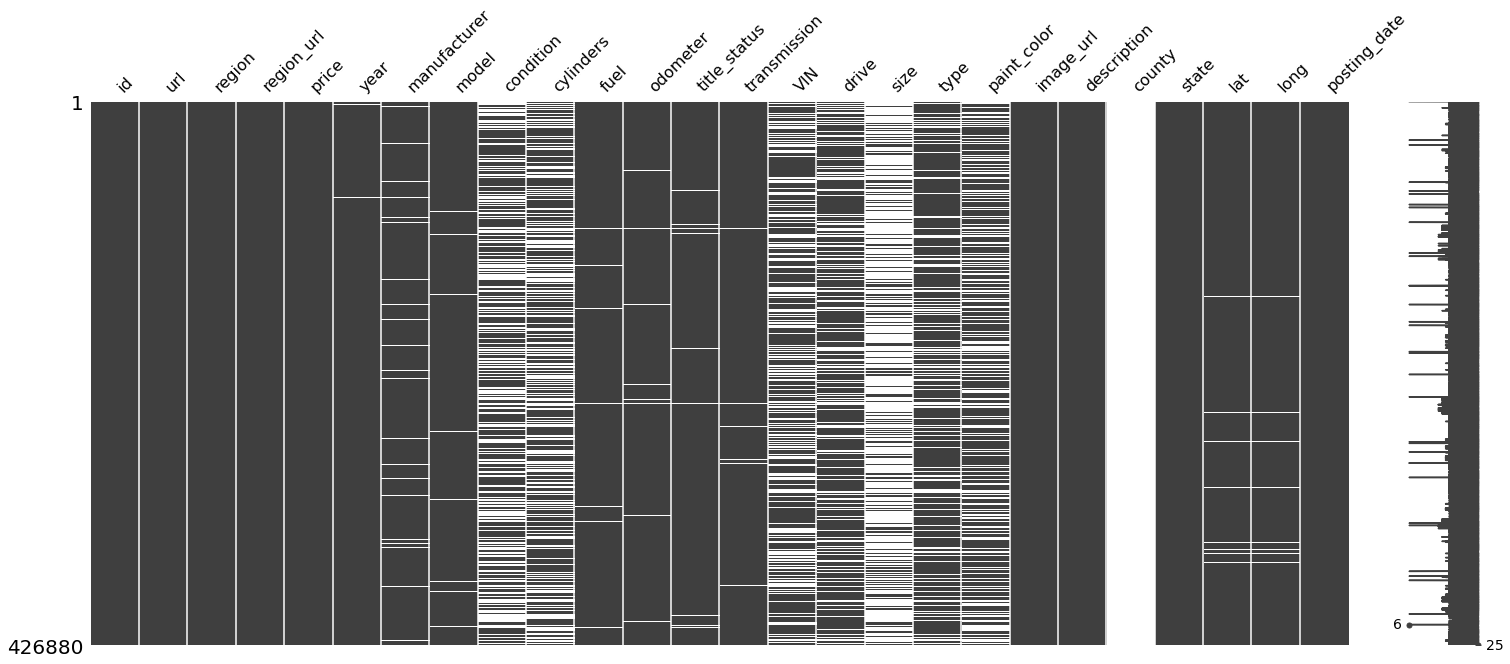
rawData.head()



#Drop Unnecessary Columns

import missingno

missingno.matrix(rawData)



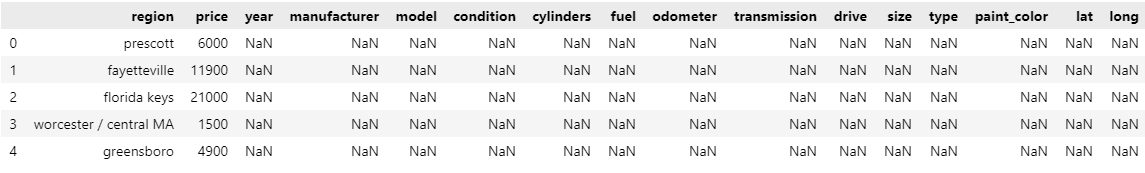
drop\_columns = [

    'id', 'url', 'region\_url', 'VIN', 'image\_url', 'description', 'county', 'state', 'posting\_date', 'title\_status'

]

rawData.drop(columns=drop\_columns, inplace=True)

rawData.head()



2.1. Categorical Data

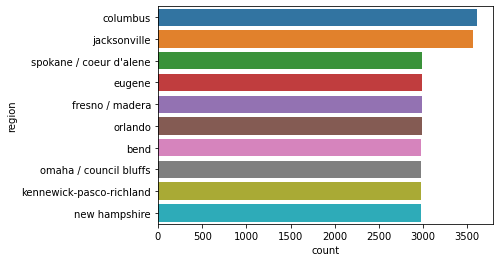
len(rawData['region'].unique())

rawData['region'].value\_counts()



#Top 10

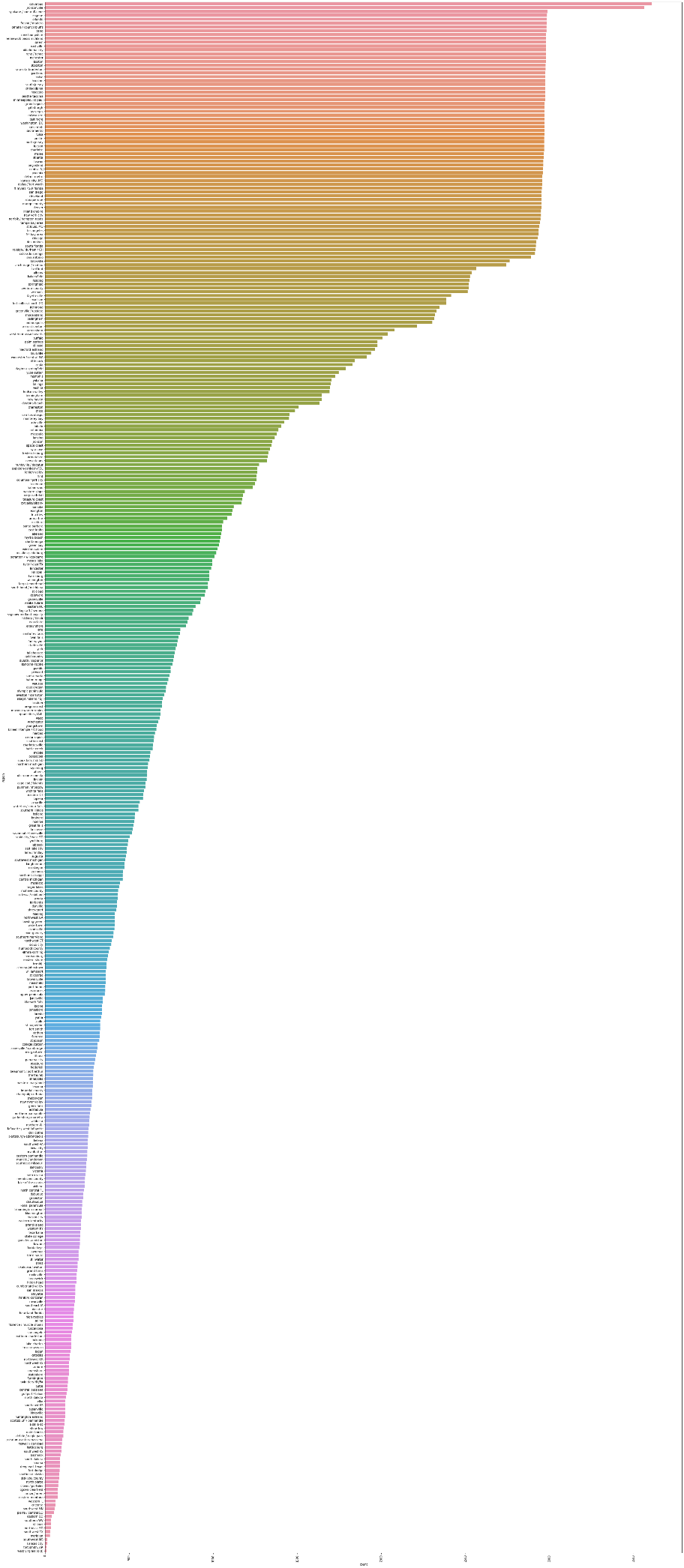
sns.countplot(data=rawData, y='region', order=rawData['region'].value\_counts().iloc[:10].index)



plt.figure(figsize=(10,100))

sns.countplot(data=rawData, y='region', order=rawData['region'].value\_counts().index)

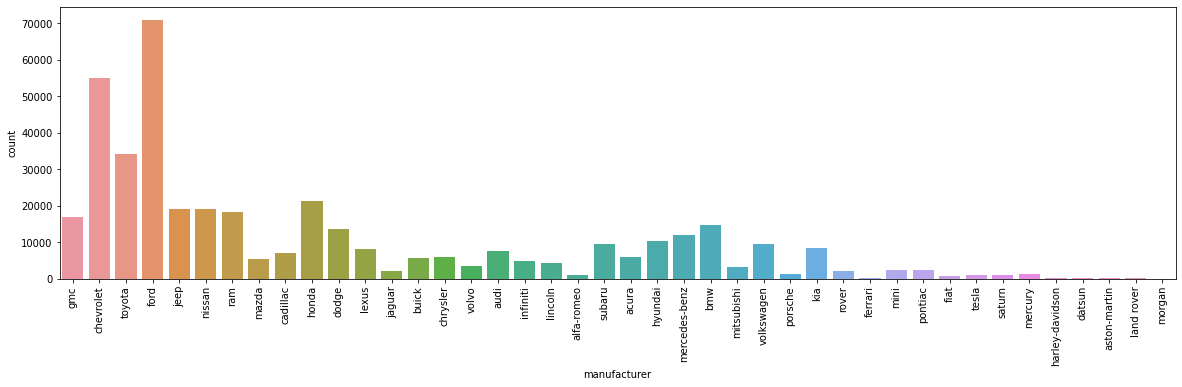
plt.xticks(rotation=90)



plt.figure(figsize=(20,5))

plt.xticks(rotation=90)

sns.countplot(data=rawData, x='manufacturer')



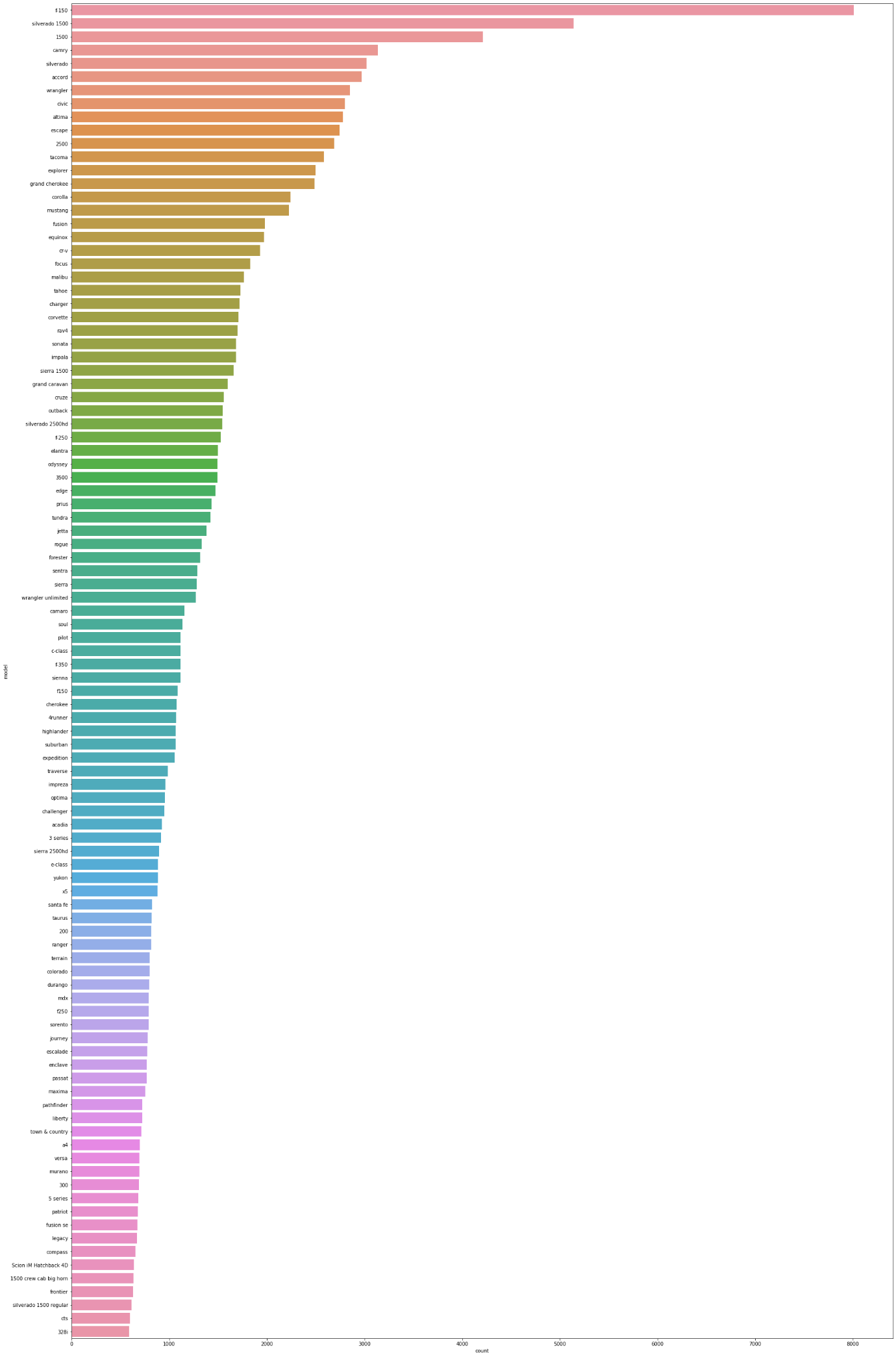
len(rawData['model'].unique())

OUTPUT:

29668

plt.figure(figsize=(30,50))

sns.countplot(data=rawData, y='model', order=rawData['model'].value\_counts().iloc[:100].index)

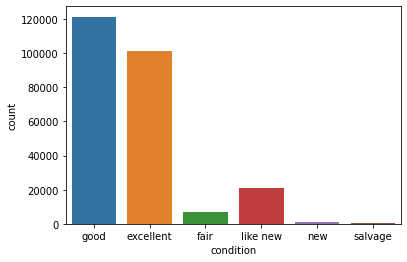


plt.figure(figsize=(30,50))

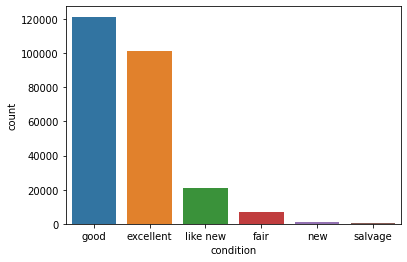
sns.countplot(data=rawData, y='model', order=rawData['model'].value\_counts().iloc[29500:].index)



sns.countplot(data=rawData, x='condition')

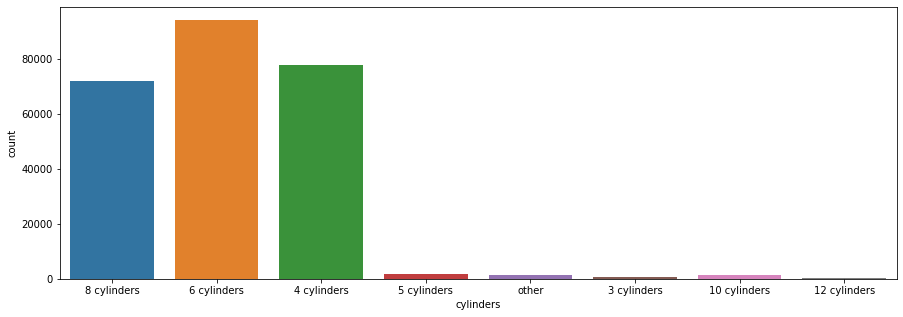


sns.countplot(data=rawData, x='condition', order=rawData['condition'].value\_counts().index)



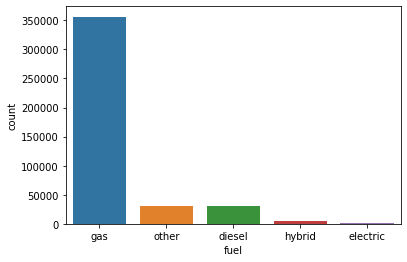
plt.figure(figsize=(15,5))

sns.countplot(data=rawData, x='cylinders')

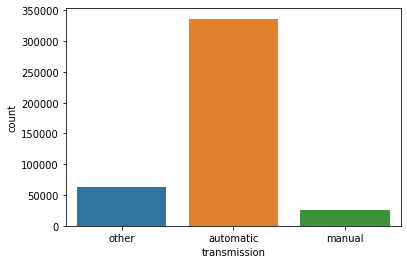


rawData[rawData['cylinders']=='12 cylinders']

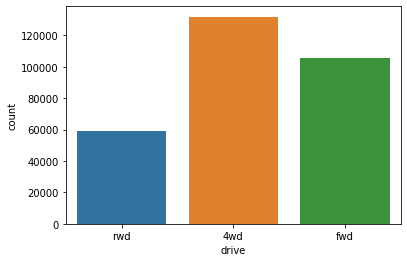
sns.countplot(data=rawData, x='fuel')



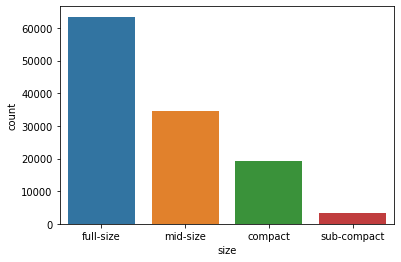
sns.countplot(data=rawData, x='transmission')



sns.countplot(data=rawData, x='drive')

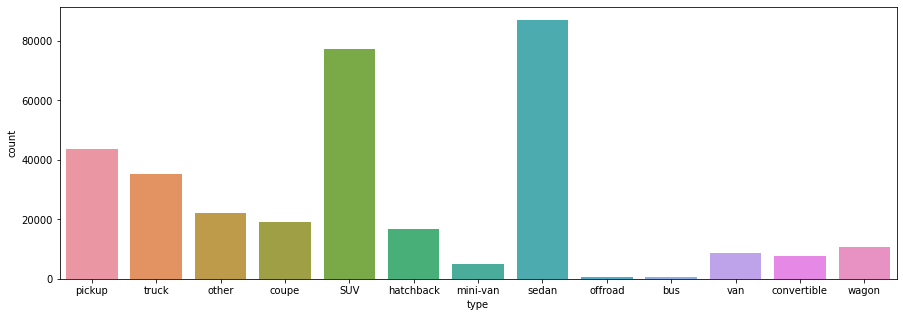


sns.countplot(data=rawData, x='size')



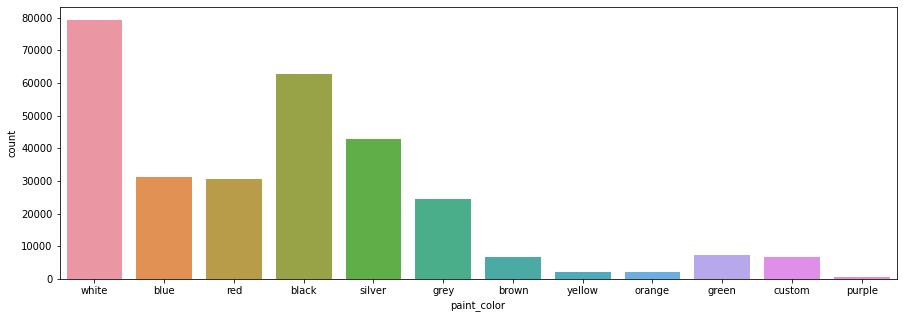
plt.figure(figsize=(15,5))

sns.countplot(data=rawData, x='type')



plt.figure(figsize=(15,5))

sns.countplot(data=rawData, x='paint\_color')



2.2. Continuous Data

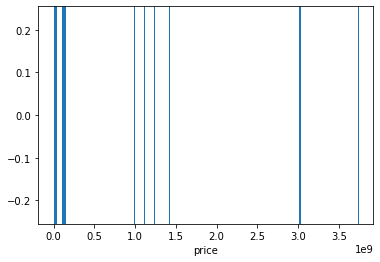
rawData.columns

OUTPUT:

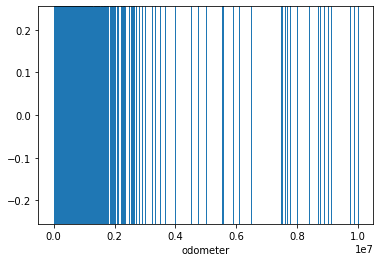
Index(['region', 'price', 'year', 'manufacturer', 'model', 'condition', 'cylinders', 'fuel', 'odometer', 'transmission', 'drive', 'size', 'type', 'paint\_color', 'lat', 'long'], dtype='object')

#Rugplots can be useful when the size of continuous data is too large to visualize with histograms

sns.rugplot(data=rawData, x='price', height=1)



sns.rugplot(data=rawData, x='odometer', height=1)



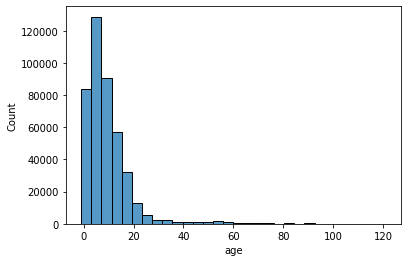
#Generating a New Column for Car Age

current\_year = 2021

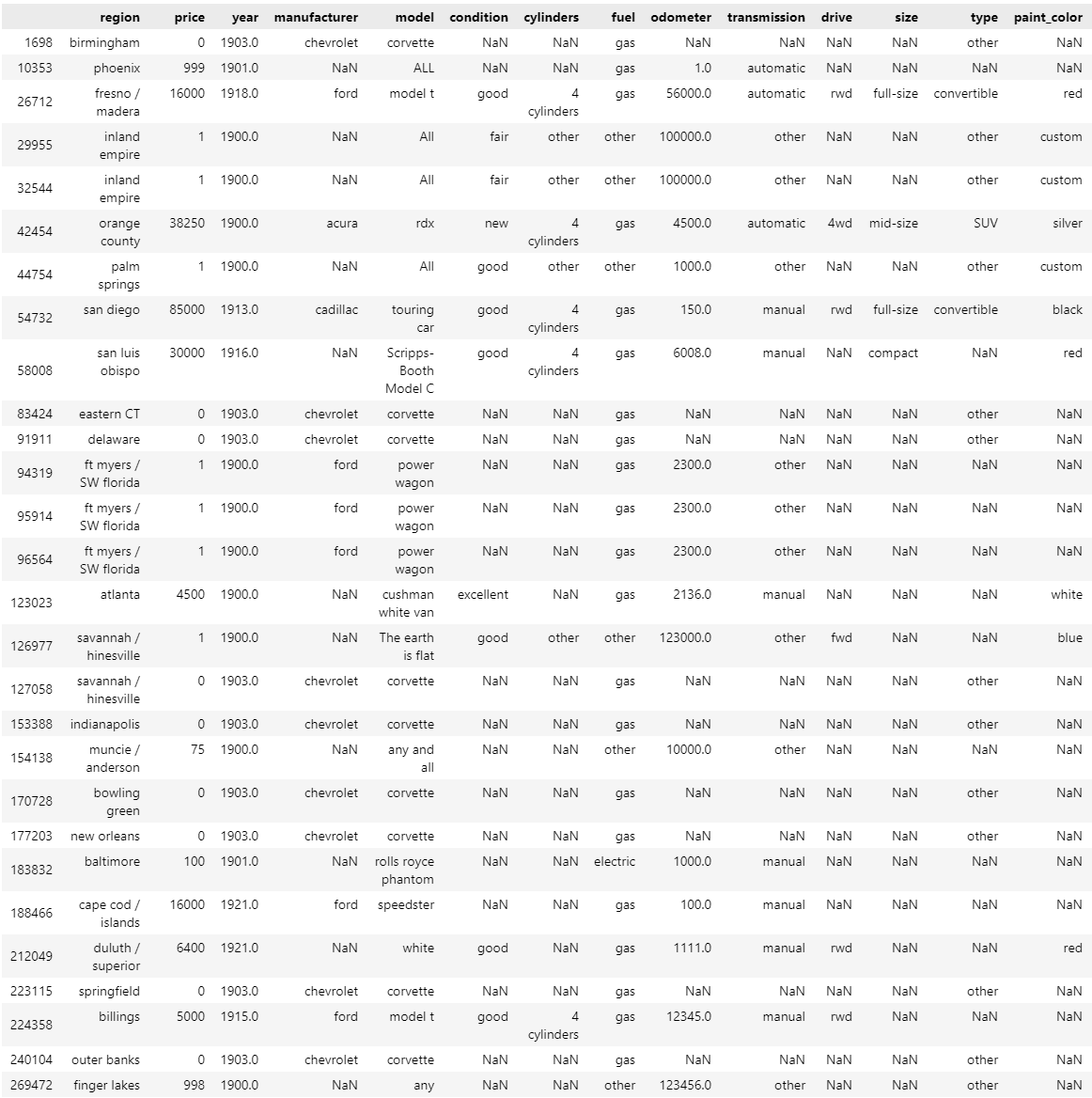
rawData['age'] = current\_year - rawData['year']

sns.histplot(data=rawData, x='age', bins=30)

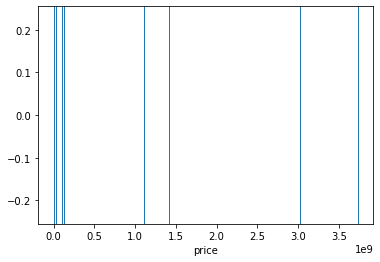
sns.histplot(data=rawData, x='age', bins=30)



rawData[rawData['age']>=100]



sns.rugplot(data=rawData[ (rawData['age'] >=20) & (rawData['age']<=40) ], x='price', height=1)



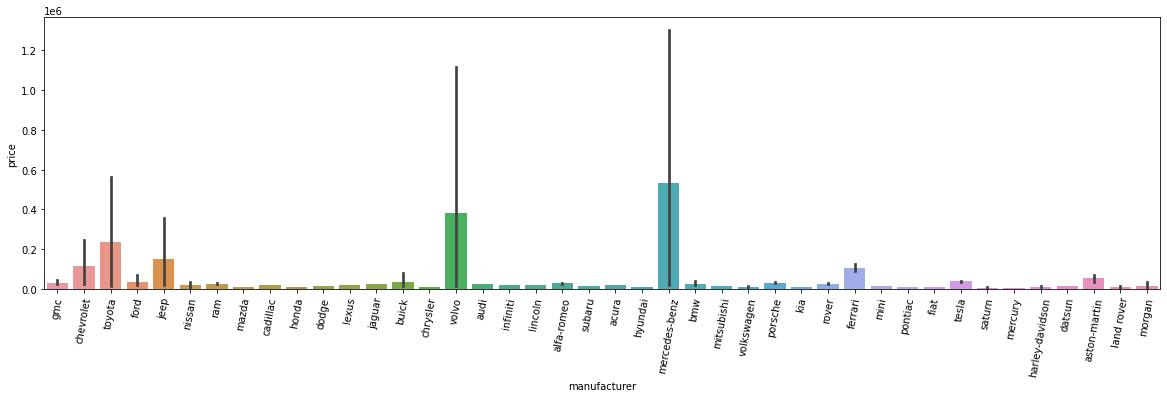
2.3. Relationship with the Target Variable

rawData[ (rawData['age'] >40) & (rawData['age']<100) ]

plt.figure(figsize=(20,5))

plt.xticks(rotation=80)

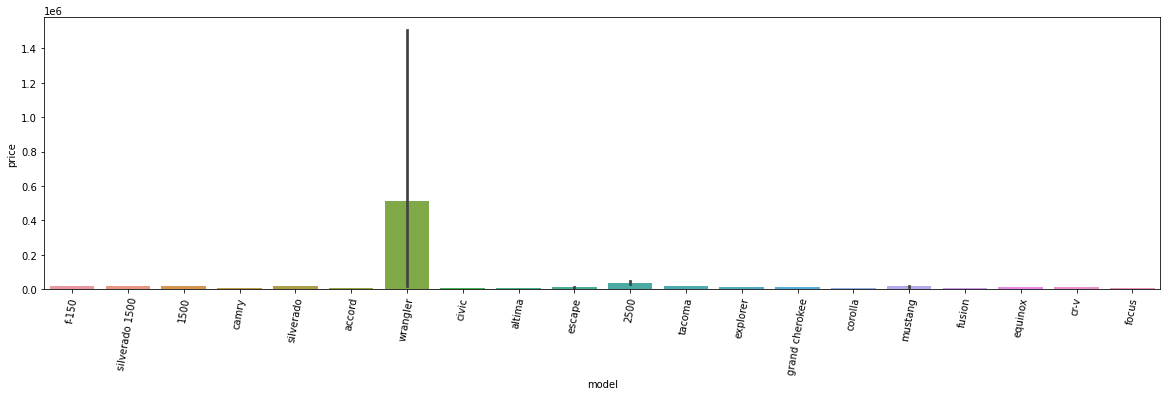
sns.barplot(data=rawData, x='manufacturer', y='price')



plt.figure(figsize=(20,5))

plt.xticks(rotation=80)

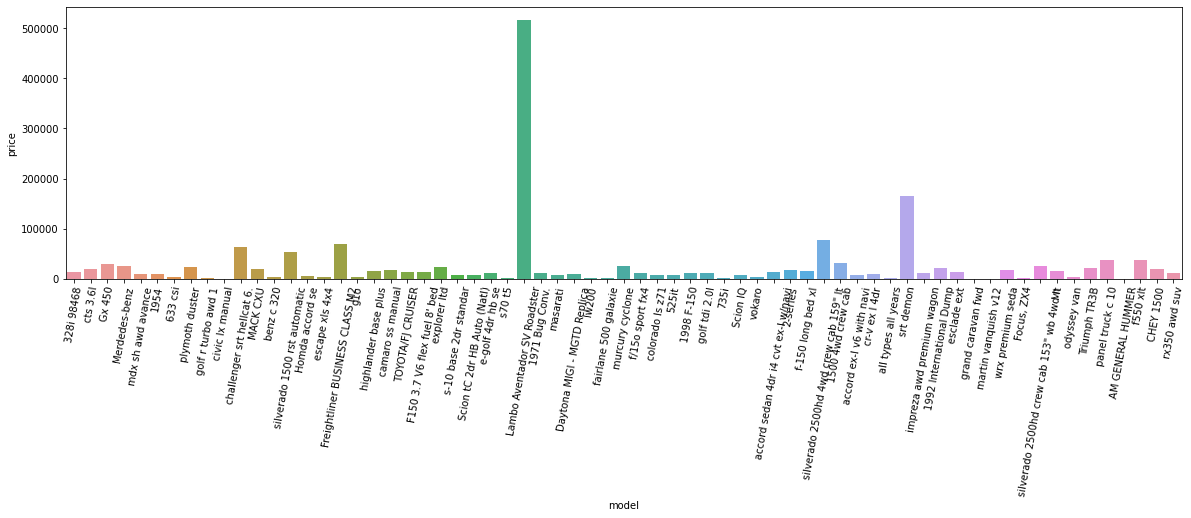
sns.barplot(data=rawData, x='model', y='price', order=rawData['model'].value\_counts().iloc[:20].index)



plt.figure(figsize=(20,5))

plt.xticks(rotation=80)

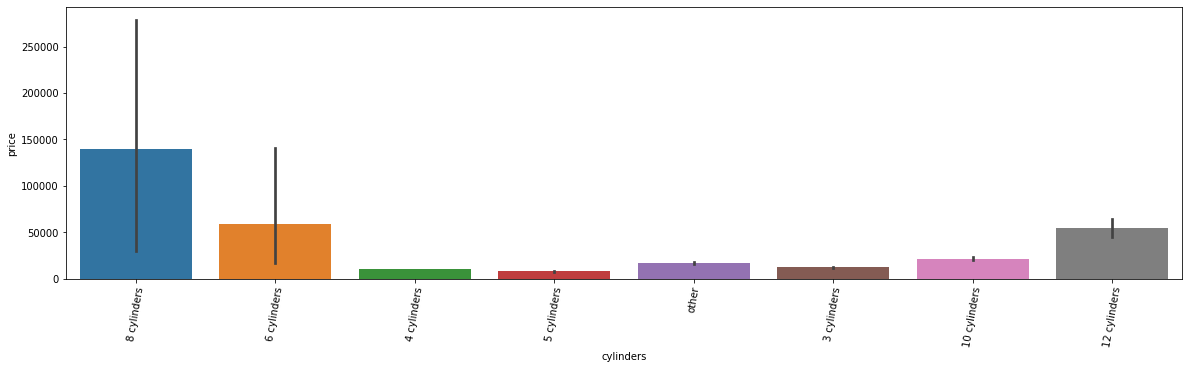
sns.barplot(data=rawData, x='model', y='price', order=rawData['model'].value\_counts().iloc[29600:].index)



plt.figure(figsize=(20,5))

plt.xticks(rotation=80)

sns.barplot(data=rawData, x='cylinders', y='price')



display(rawData[rawData['cylinders']=='8 cylinders'].price.mean())

display(rawData[rawData['cylinders']=='6 cylinders'].price.mean())

OUTPUT:

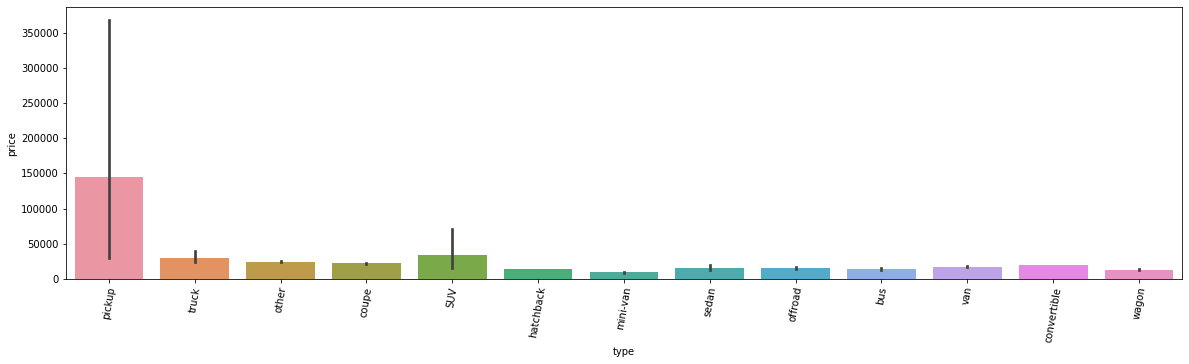
139595.72214204434

58696.31705763043

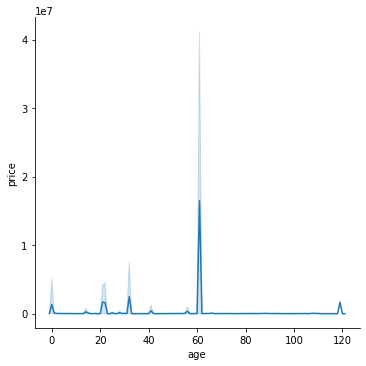
plt.figure(figsize=(20,5))

plt.xticks(rotation=80)

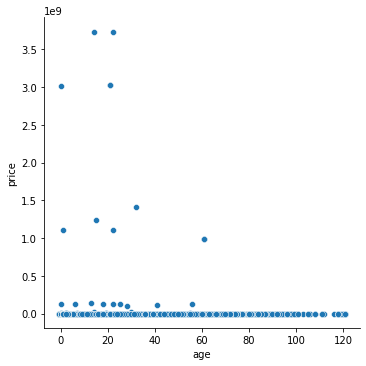
sns.barplot(data=rawData, x='type', y='price')



sns.relplot(data=rawData, x='age', y='price', kind='line')



sns.relplot(data=rawData, x='age', y='price')



3. Preprocessing

3.1. Processing Outliers

high = rawData['price'].quantile(0.99)

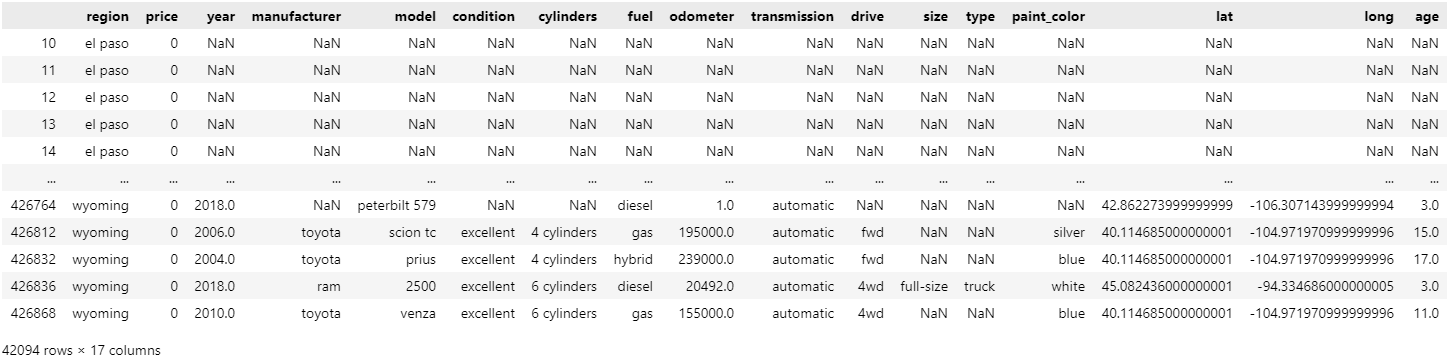
high

rawData[rawData['price']>high]

low = rawData['price'].quantile(0.1)

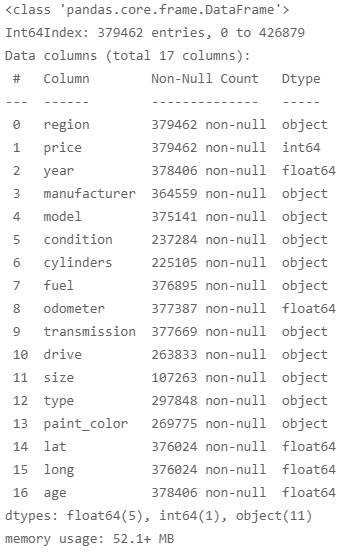
low

rawData[rawData['price']<low]



rawData = rawData[ (high > rawData['price']) & (rawData['price'] > low)]

rawData.info()



high = rawData['odometer'].quantile(0.99)

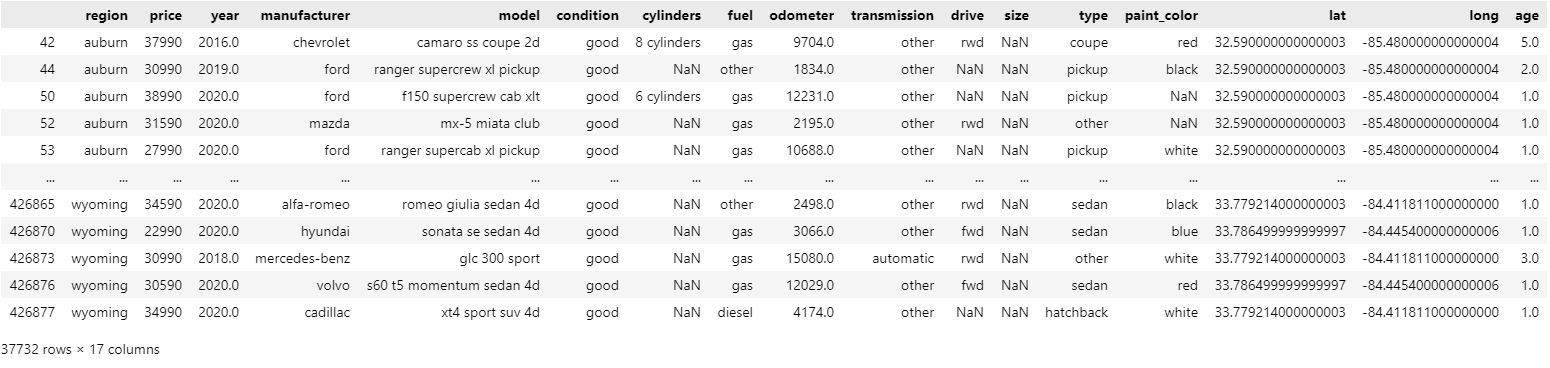
high

rawData[rawData['odometer']>high]

low = rawData['odometer'].quantile(0.1)

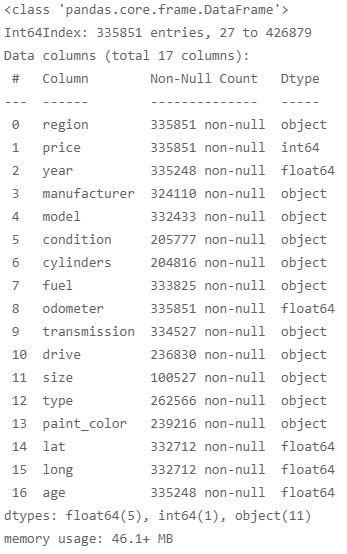
low

rawData[rawData['odometer']<low]



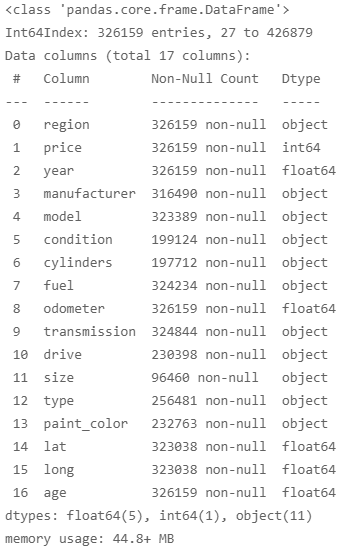
rawData = rawData[ (high>rawData['odometer']) & (rawData['odometer']>low)]

rawData.info()



rawData = rawData[rawData['age'] <= 30]

rawData.info()

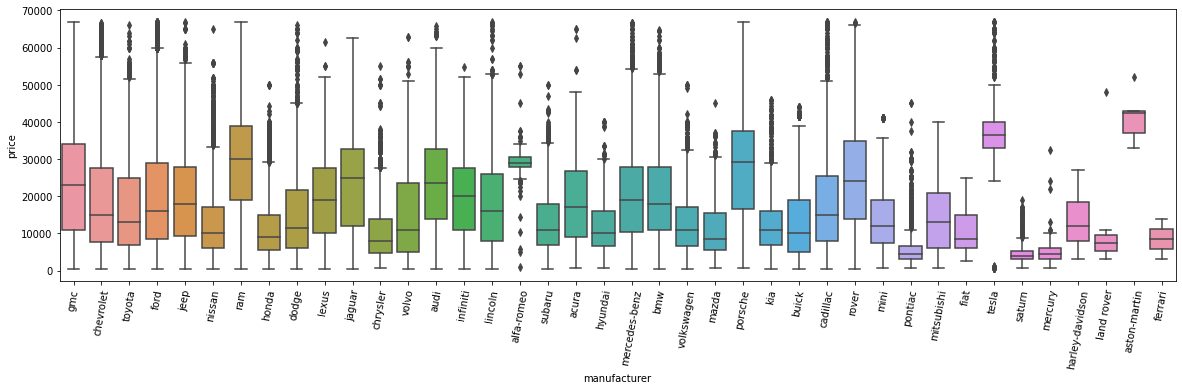


3.2. Revalidation with Visualization after Outlier Processing

plt.figure(figsize=(20,5))

plt.xticks(rotation=80)

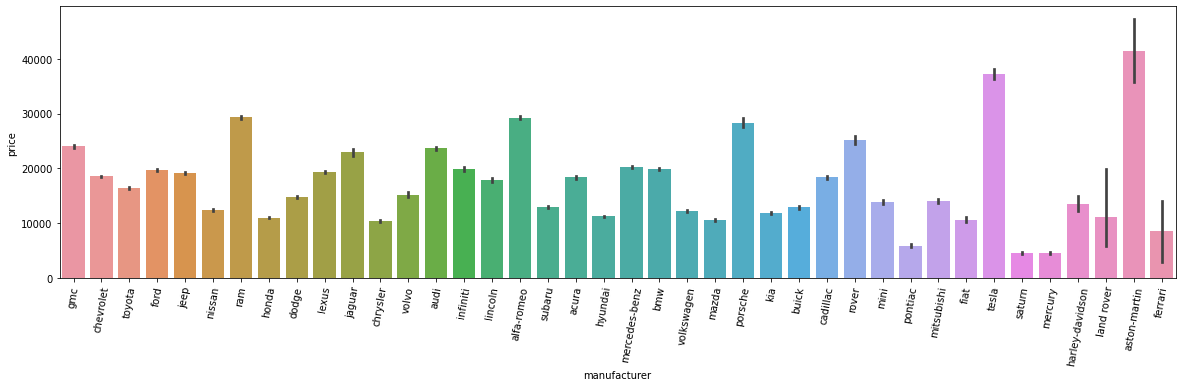
sns.boxplot(data=rawData, x='manufacturer', y='price')



plt.figure(figsize=(20,5))

plt.xticks(rotation=80)

sns.barplot(data=rawData, x='manufacturer', y='price')



display(rawData[rawData['manufacturer']=='ferrari'].price.mean())

display(rawData[rawData['manufacturer']=='toyota'].price.mean())

OUTPUT:

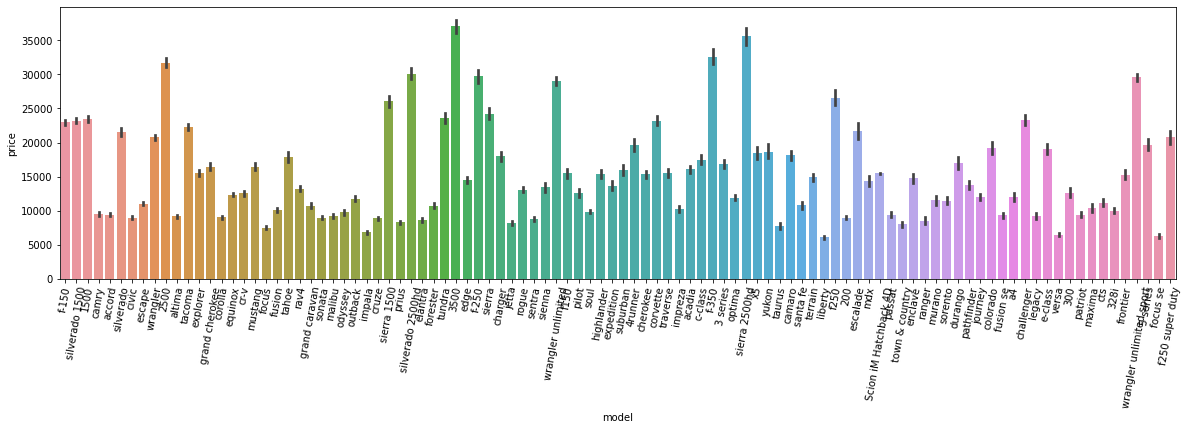
8493.0

16423.587427536233

plt.figure(figsize=(20,5))

plt.xticks(rotation=80)

sns.barplot(data=rawData, x='model', y='price', order=rawData['model'].value\_counts().iloc[:100].index)



len(rawData['model'].unique())

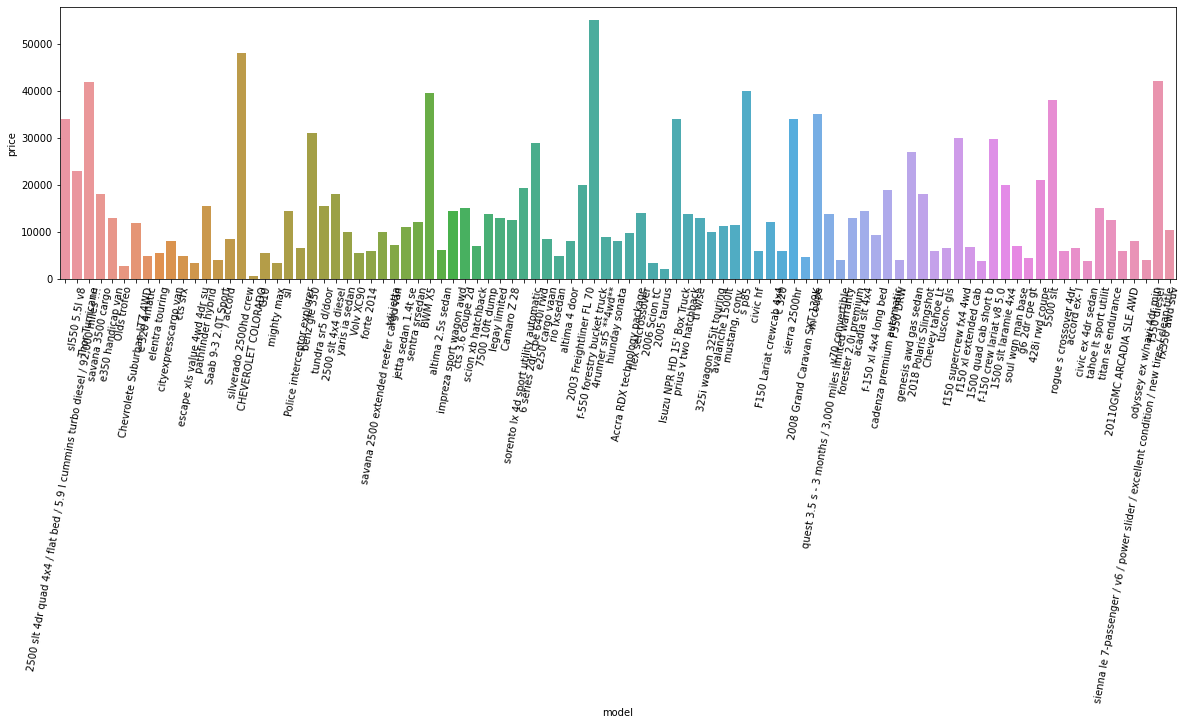
OUTPUT:

22096

plt.figure(figsize=(20,5))

plt.xticks(rotation=80)

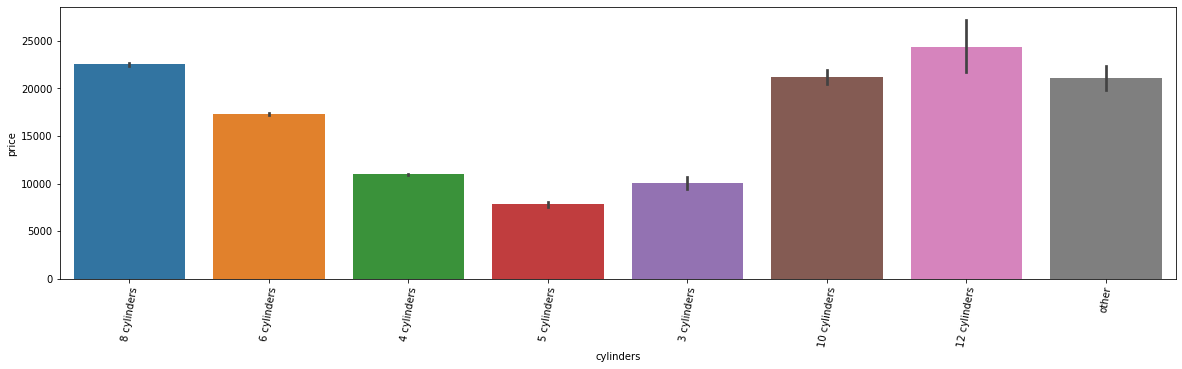
sns.barplot(data=rawData, x='model', y='price', order=rawData['model'].value\_counts().iloc[22000:].index)



plt.figure(figsize=(20,5))

plt.xticks(rotation=80)

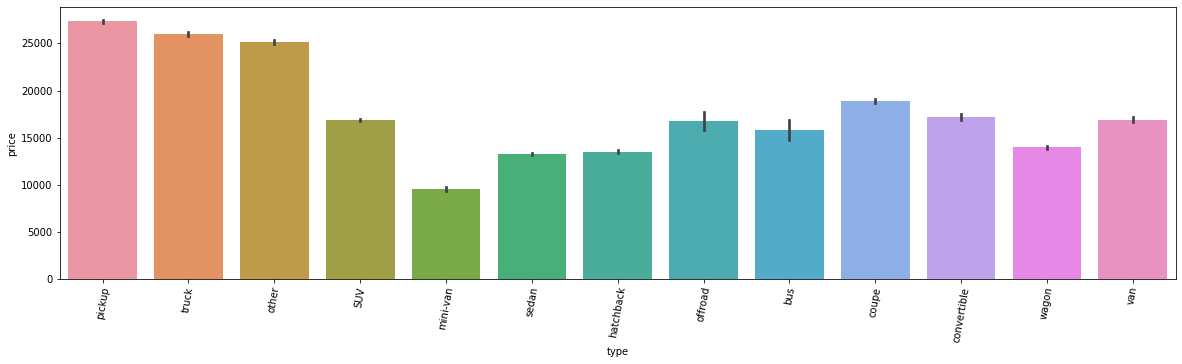
sns.barplot(data=rawData, x='cylinders', y='price')



plt.figure(figsize=(20, 5))

plt.xticks(rotation=80)

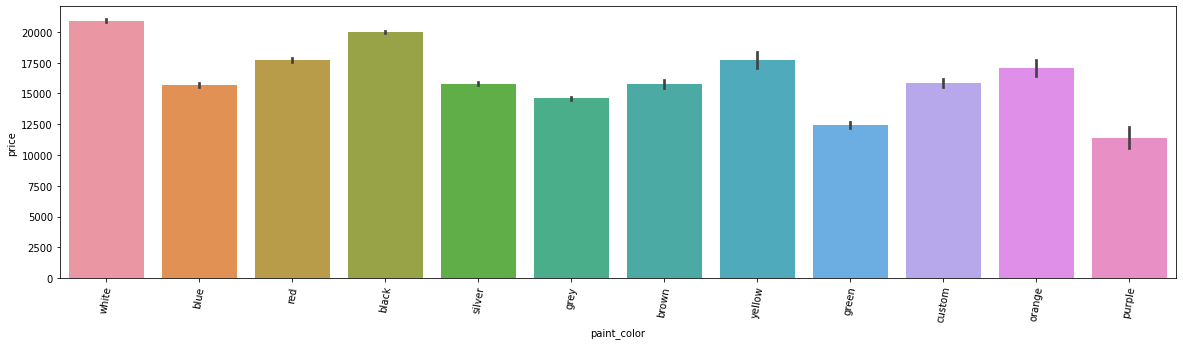
sns.barplot(data=rawData, x='type', y='price')



plt.figure(figsize=(20, 5))

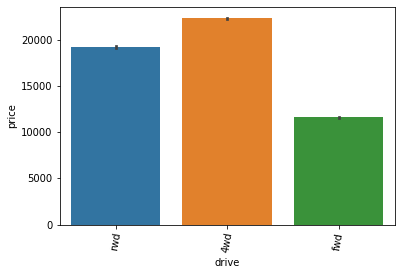
plt.xticks(rotation=80)

sns.barplot(data=rawData, x='paint\_color', y='price')



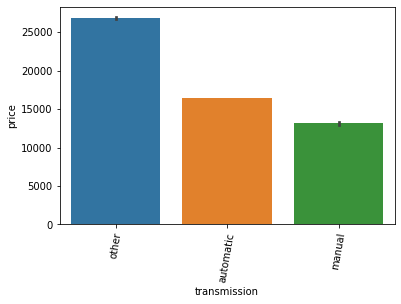
plt.xticks(rotation=80)

sns.barplot(data=rawData, x='drive', y='price')

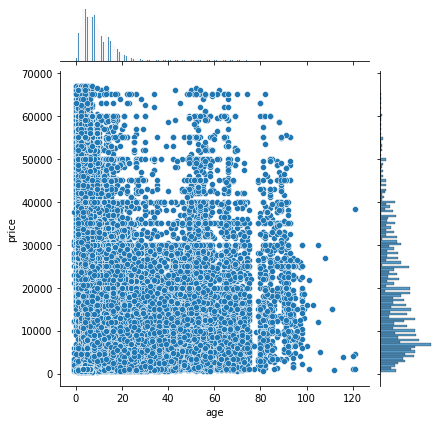


plt.xticks(rotation=80)

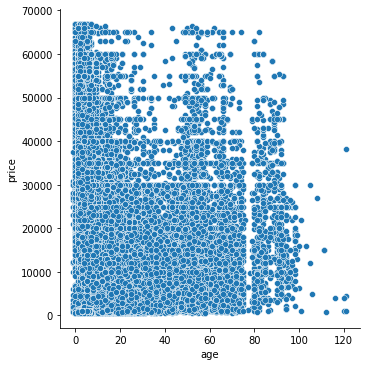
sns.barplot(data=rawData, x='transmission', y='price')



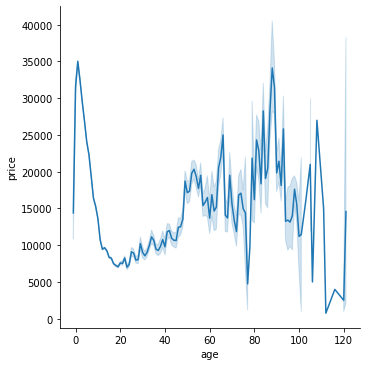
sns.jointplot(data=rawData, x='age', y='price')



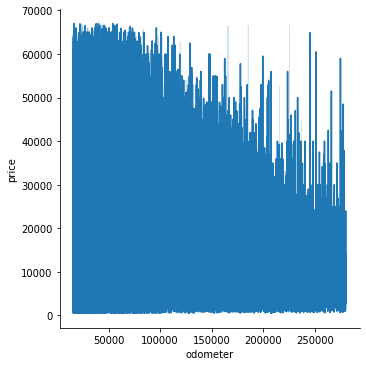
sns.relplot(data=rawData, x='age', y='price')

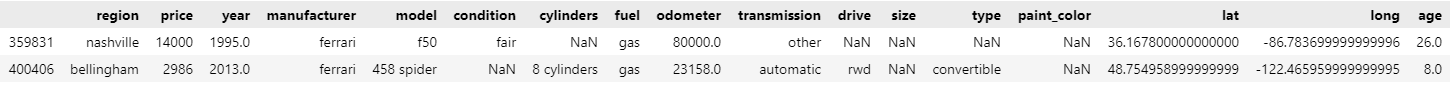


sns.relplot(data=rawData, x='age', y='price', kind='line')

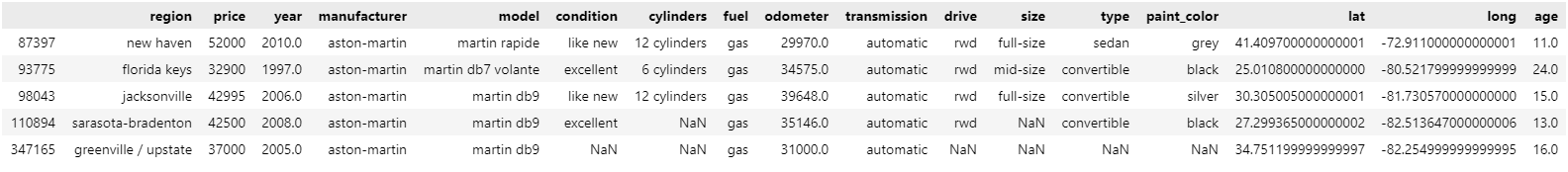


sns.relplot(data=rawData, x='odometer', y='price', kind='line')

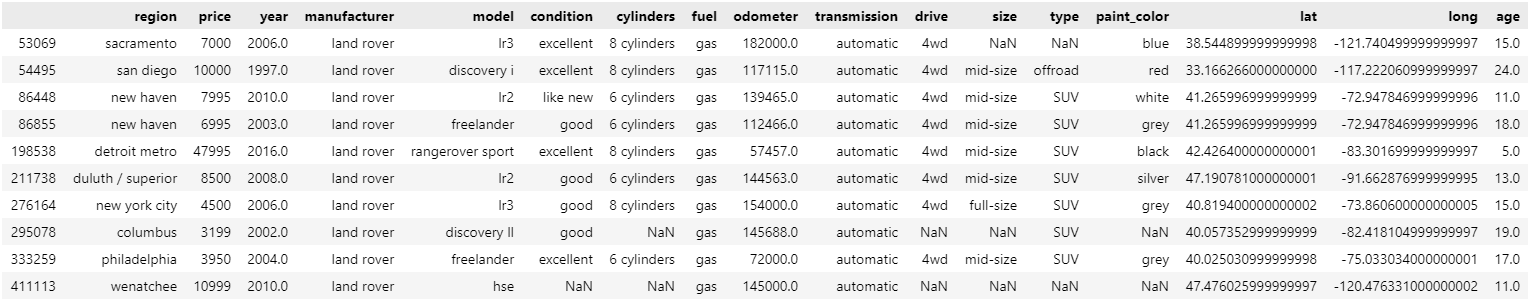
  
rawData[ rawData['manufacturer'] == 'ferrari' ]



rawData[ rawData['manufacturer'] == 'aston-martin' ]



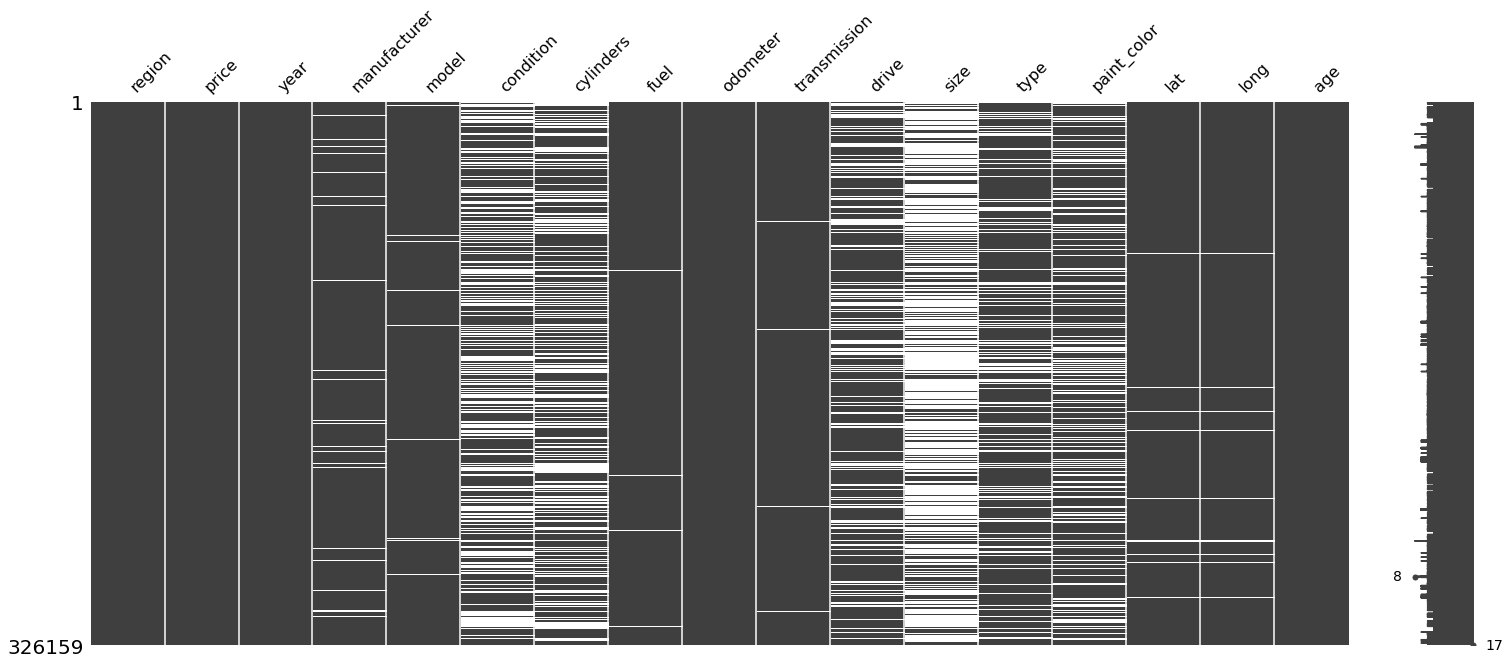
rawData[ rawData['manufacturer'] == 'land rover' ]



As implied from the results, certain values typically associated with used car prices such as the brand values, car ages, short odometer ranges are not being reflected on the actual used car prices. Rather, the type of cars (e.g., trucks, SUV, Sedan) depict some tendencies towards prices. Larger cars (including trucks, SUVs, Vans) have general advantage in terms of price defense compared to other types.

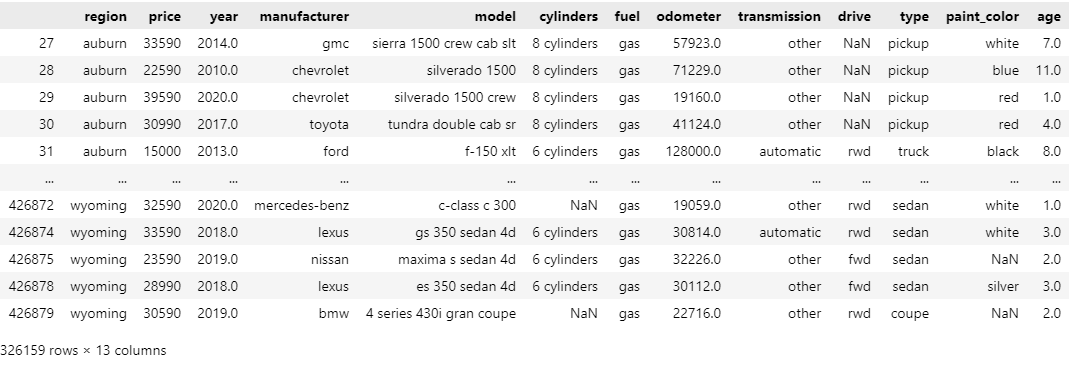
3.3. Imputation of Missing Values

missingno.matrix(rawData)

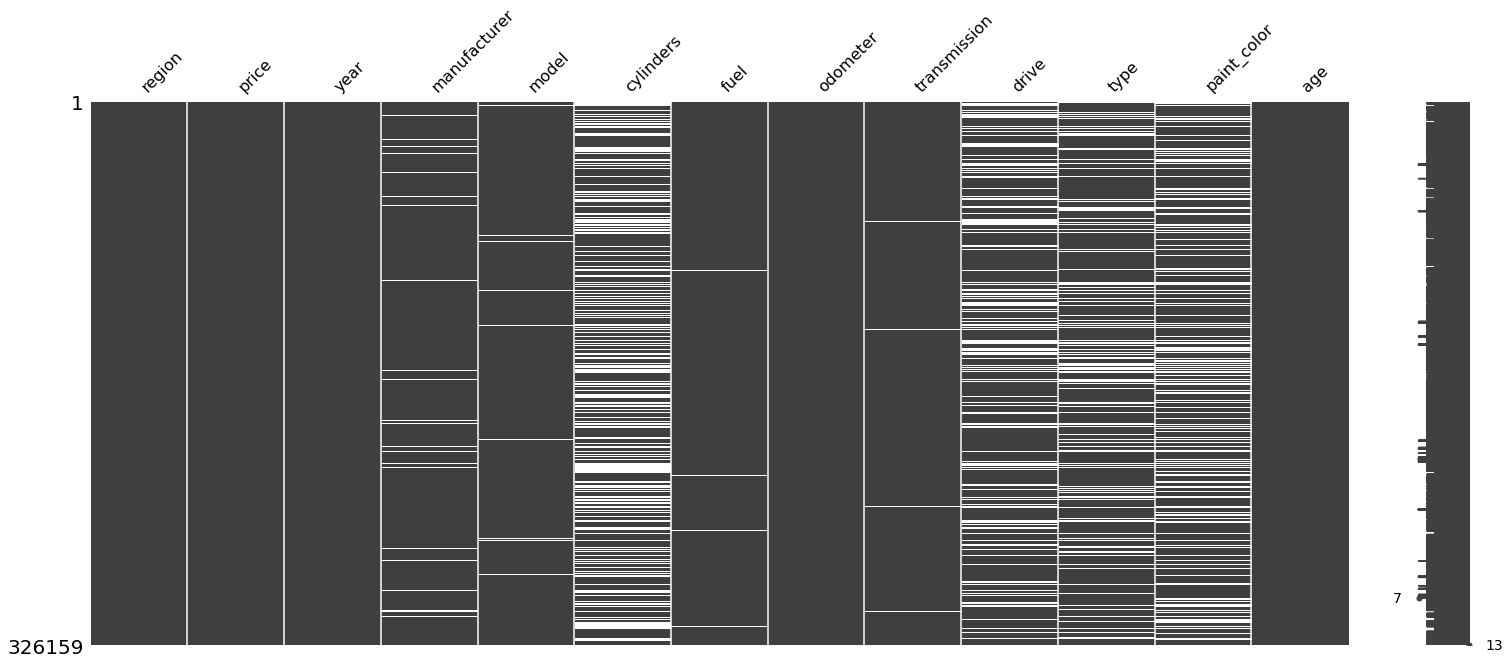


rawData.drop(columns=['size', 'condition', 'lat', 'long'], inplace=True)

rawData

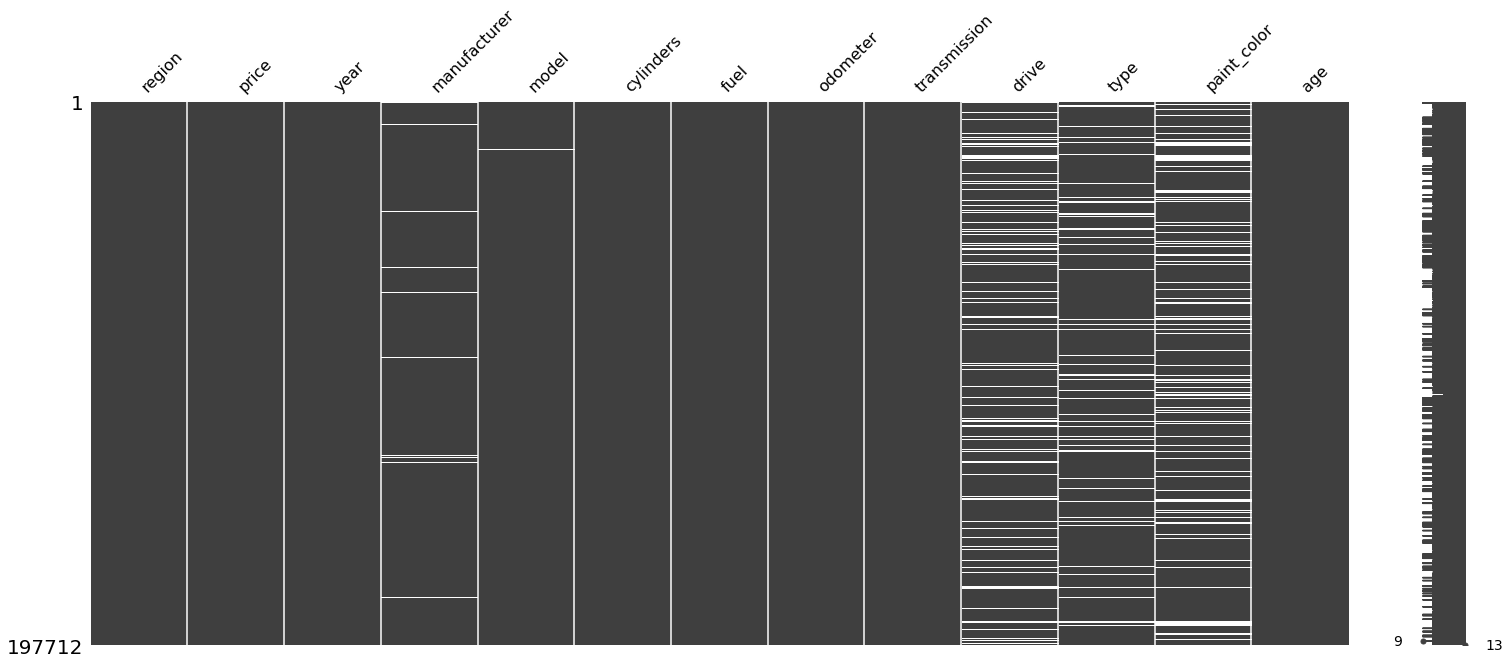


missingno.matrix(rawData)

drop\_index = rawData[ rawData['cylinders'].isna() ].index

rawData.drop(drop\_index, inplace=True)

missingno.matrix(rawData)



drop\_index = rawData[ rawData['paint\_color'].isna() ].index

rawData.drop( drop\_index, inplace=True )

drop\_index = rawData[ rawData['drive'].isna() ].index

rawData.drop( drop\_index, inplace=True )

drop\_index = rawData[ rawData['type'].isna() ].index

rawData.drop( drop\_index, inplace=True )

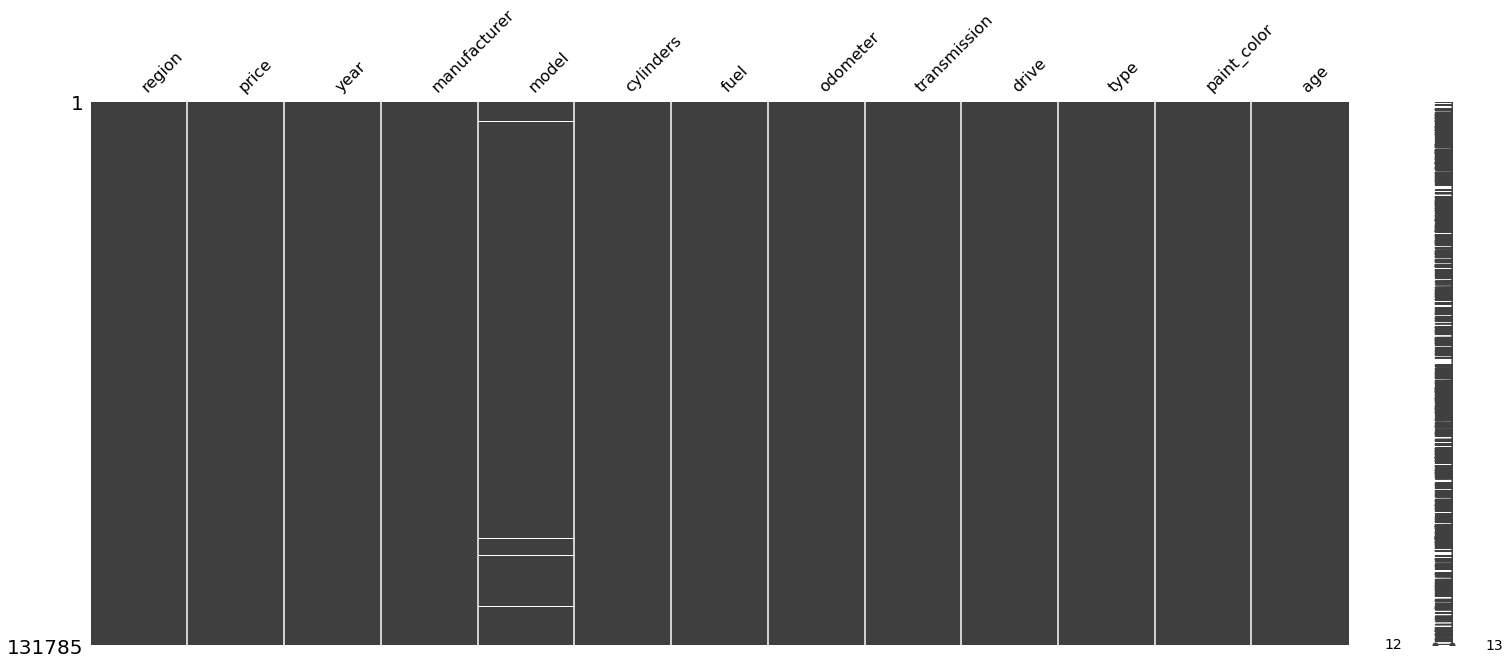
drop\_index = rawData[ rawData['manufacturer'].isna() ].index

rawData.drop( drop\_index, inplace=True )

drop\_index = rawData[ rawData['fuel'].isna() ].index

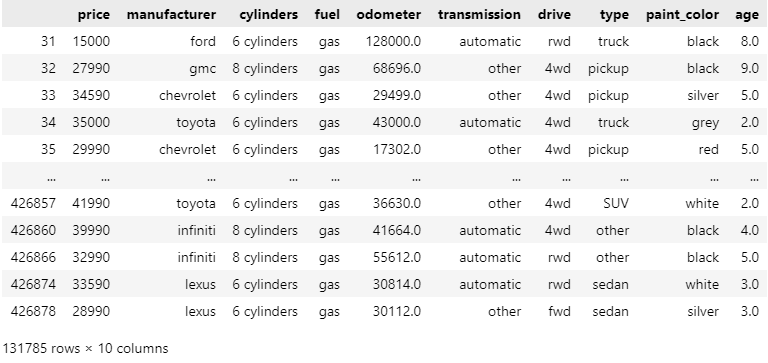
rawData.drop( drop\_index, inplace=True )

missingno.matrix(rawData)

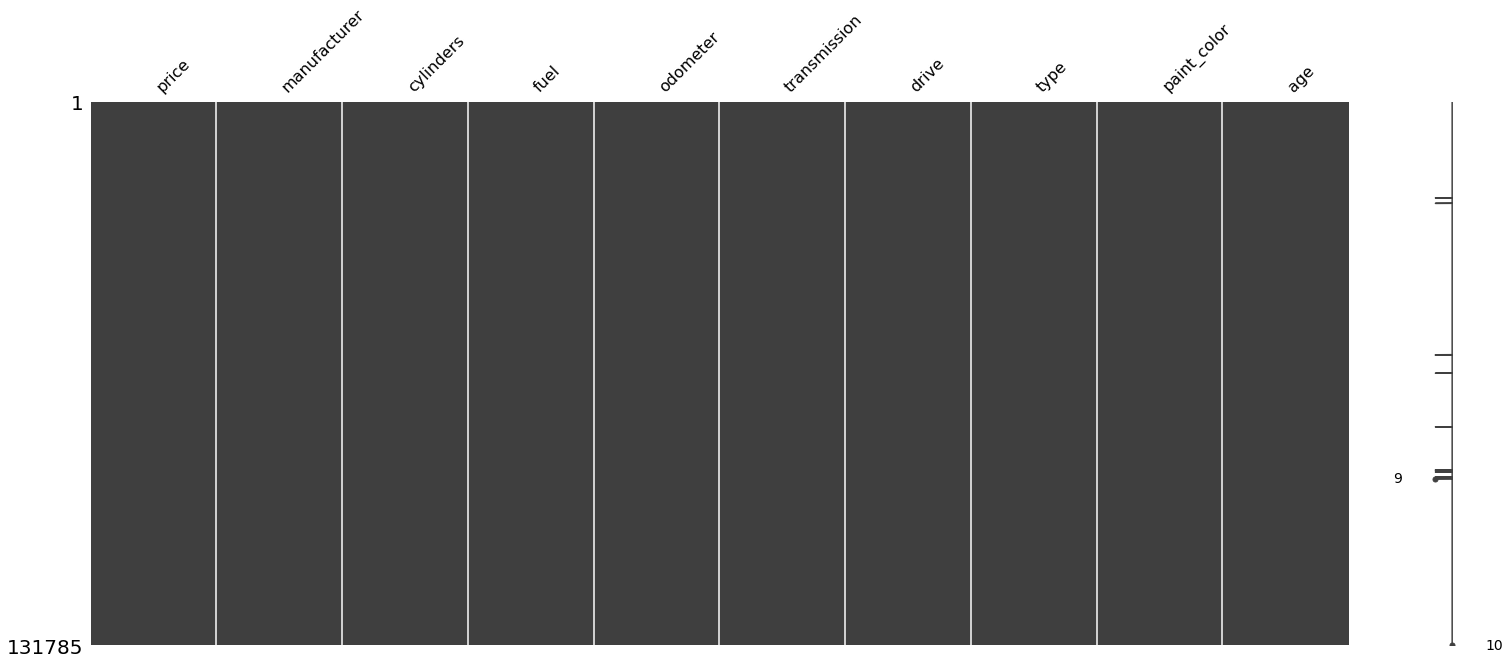


rawData.drop(columns=['region','year','model'], inplace=True)

rawData



missingno.matrix(rawData)



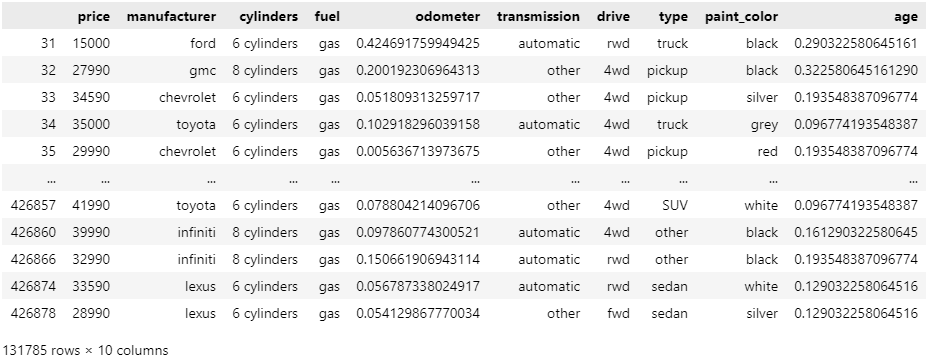
3.4. Scaling

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature\_range=(0,1))

rawData[['odometer','age']]=scaler.fit\_transform(rawData[['odometer','age']])

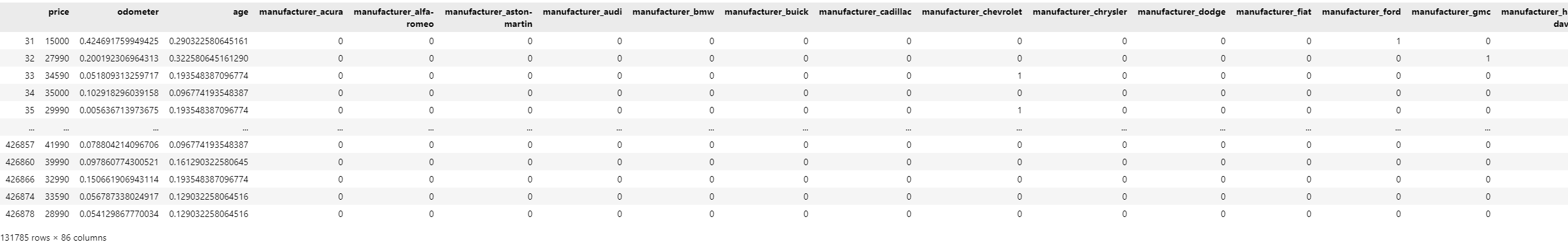
rawData



3.5. Generation of Dummy Variables

rawData = pd.get\_dummies(data=rawData, columns=['manufacturer','cylinders','fuel','transmission','drive','type','paint\_color'])

rawData



4. Modelling

4.1. Base Model

from sklearn.model\_selection import train\_test\_split, GridSearchCV, KFold

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

from sklearn.linear\_model import LinearRegression, ElasticNet

x = rawData.drop(columns=['price'])

y= rawData['price']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3)

model = LinearRegression().fit(x\_train, y\_train)

yhat\_train = model.predict(x\_train)

train\_score = r2\_score(y\_train, yhat\_train)

train\_mse = mean\_squared\_error(y\_train, yhat\_train)

train\_mae = mean\_absolute\_error(y\_train, yhat\_train)

display(train\_score)

display(train\_mse)

display(train\_mae)

OUTPUT:

0.7556546901399812

35646950.61895522

4309.206387061106

yhat\_test = model.predict(x\_test)

test\_score = r2\_score(y\_test, yhat\_test)

test\_mse = mean\_squared\_error(y\_test, yhat\_test)

test\_mae = mean\_absolute\_error(y\_test, yhat\_test)

display(test\_score)

display(test\_mse)

display(test\_mae)

OUTPUT:

0.7550252489403264

35698125.567457505

4295.233989275597

pd.concat([pd.DataFrame(model.coef\_, columns=['coef']), pd.DataFrame(x\_train.columns, columns=['feature'])], axis=1)



4.2. Regularization of the Coefficients

folds = KFold(n\_splits=10, shuffle=True)

model = ElasticNet()

params = dict(

    l1\_ratio = [0, 0.3, 0.5, 0.8, 1],

    alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

)

search = GridSearchCV(estimator=model, param\_grid=params, cv=folds, scoring='r2')

search.fit(x\_train, y\_train)

model = ElasticNet(l1\_ratio=1, alpha=0.1).fit(x\_train, y\_train)

yhat\_train = model.predict(x\_train)

train\_score = r2\_score(y\_train, yhat\_train)

train\_mse = mean\_squared\_error(y\_train, yhat\_train)

train\_mae = mean\_absolute\_error(y\_train, yhat\_train)

display(train\_score)

display(train\_mse)

display(train\_mae)

OUTPUT:

0.7556870845955048

35642224.66961311

4309.748900258853

yhat\_test = model.predict(x\_test)

test\_score = r2\_score(y\_test, yhat\_test)

test\_mse = mean\_squared\_error(y\_test, yhat\_test)

test\_mae = mean\_absolute\_error(y\_test, yhat\_test)

display(test\_score)

display(test\_mse)

display(test\_mae)

OUTPUT:

0.7550592926235883

35693164.6656723

4295.157656459854

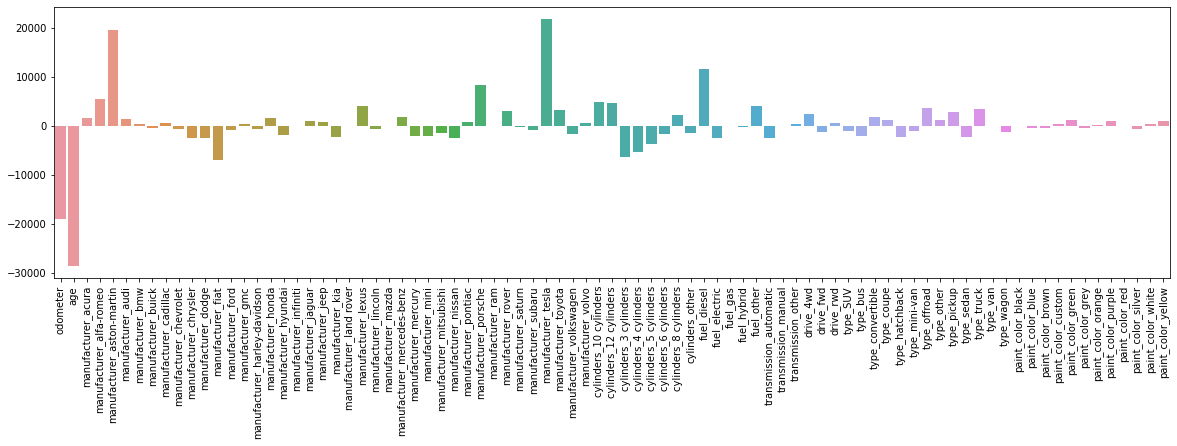
pd.concat( [ pd.DataFrame( model.coef\_.T, columns=['coef'] ), pd.DataFrame( x\_train.columns, columns=['feature']) ], axis=1 )



plt.figure(figsize=(20,5))

plt.xticks(rotation=90)

sns.barplot(x = x.columns, y = model.coef\_)



4.3. Feature Selection

4.1. Stepwise Selection

search = dt.stepwise\_selection(x\_train, y\_train)

len(search.columns)

OUTPUT:

71

search.score

OUTPUT:

0.7562533554944528

print( search.columns )

OUTPUT:

['age', 'drive\_fwd', 'odometer', 'fuel\_diesel', 'cylinders\_4 cylinders', 'cylinders\_8 cylinders', 'manufacturer\_toyota', 'transmission\_automatic', 'type\_sedan', 'type\_truck', 'type\_pickup', 'manufacturer\_honda', 'manufacturer\_lexus', 'fuel\_other', 'manufacturer\_porsche', 'drive\_4wd', 'cylinders\_10 cylinders', 'manufacturer\_nissan', 'type\_hatchback', 'manufacturer\_tesla', 'manufacturer\_mercedes-benz', 'manufacturer\_dodge', 'type\_SUV', 'type\_wagon', 'type\_mini-van', 'manufacturer\_jeep', 'manufacturer\_rover', 'manufacturer\_chrysler', 'manufacturer\_fiat', 'type\_van', 'manufacturer\_gmc', 'paint\_color\_silver', 'manufacturer\_acura', 'manufacturer\_kia', 'manufacturer\_hyundai', 'cylinders\_12 cylinders', 'manufacturer\_ford', 'paint\_color\_green', 'manufacturer\_volkswagen', 'paint\_color\_grey', 'paint\_color\_blue', 'cylinders\_3 cylinders', 'manufacturer\_chevrolet', 'manufacturer\_mini', 'manufacturer\_subaru', 'manufacturer\_mitsubishi', 'manufacturer\_mercury', 'manufacturer\_alfa-romeo', 'type\_offroad', 'type\_convertible', 'type\_bus', 'manufacturer\_aston-martin', 'paint\_color\_brown', 'paint\_color\_black', 'manufacturer\_audi', 'manufacturer\_jaguar', 'paint\_color\_red', 'manufacturer\_buick', 'paint\_color\_white', 'manufacturer\_lincoln', 'cylinders\_5 cylinders', 'fuel\_electric', 'manufacturer\_pontiac', 'manufacturer\_bmw', 'manufacturer\_volvo', 'manufacturer\_cadillac', 'paint\_color\_yellow', 'transmission\_other', 'manufacturer\_ram', 'type\_coupe', 'cylinders\_6 cylinders']

4.2. Recursive Feature Elimination

from sklearn.feature\_selection import RFECV

folds = KFold(n\_splits=10, shuffle=True)

model = ElasticNet(l1\_ratio=1, alpha=0.1)

result = RFECV(model, cv=folds, scoring='r2')

result.fit(x\_train, y\_train)

print( result.support\_.sum() )

print( x\_train.loc[ :, result.support\_ ].columns )

OUTPUT:

77

Index(['odometer', 'age', 'manufacturer\_acura', 'manufacturer\_alfa-romeo',

'manufacturer\_aston-martin', 'manufacturer\_audi', 'manufacturer\_bmw',

'manufacturer\_buick', 'manufacturer\_cadillac', 'manufacturer\_chevrolet',

'manufacturer\_chrysler', 'manufacturer\_dodge', 'manufacturer\_fiat',

'manufacturer\_ford', 'manufacturer\_gmc', 'manufacturer\_honda',

'manufacturer\_hyundai', 'manufacturer\_jaguar', 'manufacturer\_jeep',

'manufacturer\_kia', 'manufacturer\_land rover', 'manufacturer\_lexus',

'manufacturer\_lincoln', 'manufacturer\_mercedes-benz',

'manufacturer\_mercury', 'manufacturer\_mini', 'manufacturer\_mitsubishi',

'manufacturer\_nissan', 'manufacturer\_pontiac', 'manufacturer\_porsche',

'manufacturer\_ram', 'manufacturer\_rover', 'manufacturer\_saturn',

'manufacturer\_subaru', 'manufacturer\_tesla', 'manufacturer\_toyota',

'manufacturer\_volkswagen', 'manufacturer\_volvo',

'cylinders\_10 cylinders', 'cylinders\_12 cylinders',

'cylinders\_3 cylinders', 'cylinders\_4 cylinders',

'cylinders\_5 cylinders', 'cylinders\_6 cylinders',

'cylinders\_8 cylinders', 'cylinders\_other', 'fuel\_diesel',

'fuel\_electric', 'fuel\_hybrid', 'fuel\_other', 'transmission\_automatic',

'transmission\_other', 'drive\_4wd', 'drive\_fwd', 'drive\_rwd', 'type\_SUV',

'type\_bus', 'type\_convertible', 'type\_coupe', 'type\_hatchback',

'type\_mini-van', 'type\_offroad', 'type\_other', 'type\_pickup',

'type\_sedan', 'type\_truck', 'type\_wagon', 'paint\_color\_blue',

'paint\_color\_brown', 'paint\_color\_custom', 'paint\_color\_green',

'paint\_color\_grey', 'paint\_color\_orange', 'paint\_color\_purple',

'paint\_color\_silver', 'paint\_color\_white', 'paint\_color\_yellow'],

dtype='object')

import shap

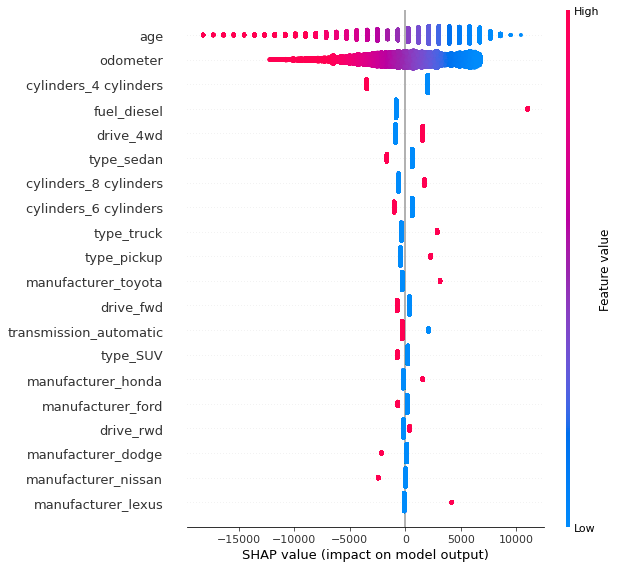
model = ElasticNet(l1\_ratio=1, alpha=0.1)

model.fit(x\_train, y\_train)

explainer = shap.LinearExplainer(model, x\_train)

shap\_value = explainer.shap\_values(x\_train)

shap.summary\_plot(shap\_value, x\_train)



4.4. Non-Linear Models

4.4.1. Support Vector Machine

from sklearn.svm import SVR

folds = KFold(n\_splits=10, shuffle=True)

model = SVR()

params = dict(

    C=[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],

    gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

)

search = GridSearchCV(estimator=model, param\_grid=params, cv=folds, scoring='neg\_mean\_squared\_error')

search.fit(x\_train, y\_train)

search.best\_params\_

search.best\_score\_

4.4.2. Random Forest

Main parameters:

n\_estimators: number of trees

max\_depth: max depth of the trees

* Pruning
* Divides until the value is lower than the min\_samples\_split
* Deeper depth increases the likelihood of overfit

min\_samples\_split: minimum number of samples for node division of each tree

* Larger number of dividing nodes (which means smaller number of minimum samples split) may result in overfits

min\_samples\_leaf: minimum number of samples for the terminal node

* Tuning for data imbalance
* If the total number of data for certain class is too small, it would be better to adjust this parameter by giving smaller values

max\_features: maximum number of features (variables)

max\_leaf\_nodes: maximum number of terminal nodes

* For pruning purpose: either use max\_depth or this parameter

from sklearn.ensemble import RandomForestRegressor

folds = KFold(n\_splits=10, shuffle=True)

model = RandomForestRegressor()

params = dict(

    n\_estimators = [100, 300, 500, 1000],

    max\_depth = [None, 3, 5, 7, 11, 17, 19, 21],

    min\_samples\_split = [2, 3, 5, 7, 9, 11],

    min\_samples\_leaf = [1, 2, 3, 5, 7, 9, 11],

    max\_features = ['auto' 'sqrt', 'log2'],

)

search = GridSearchCV(estimator=model, param\_grid=params, cv=folds, scoring='neg\_mean\_squared\_error')

search.fit(x\_train, y\_train)

search.best\_params\_

search.best\_score\_

4.4.3. XGB (Boosting)

Main parameters:

max\_depth: maximum depth of the tree (default is set to 6)

colsample\_bytree, colsample\_bylevel, colsample\_bynoe: sampling ratio for each tree/level/node (default is set to 1)

lambda: L2 regularization

alpha: L1 regularization

rate\_drop: dropout ratio (default = 0)

num\_leaves: total number of terminal nodes

n\_estimators: total number of trees

import xgboost as xgb

folds = KFold(n\_splits=10, shuffle=True)

model = xgb.XGBRegressor()

params = dict(

  n\_estimators = [100, 300, 500, 1000],

  max\_depth = [None, 3, 5, 7, 11, 17, 19, 21],

  colsample\_bytree = [0.2, 0.5, 0.7, 1],

  reg\_lambda = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],

)

search = GridSearchCV(estimator=model, param\_grid=params, cv=folds, scoring='neg\_mean\_squared\_error')

search.fit(x\_train, y\_train)

search.best\_params\_

search.best\_score\_

4.4.4. LightGBM (Boosting)

import lightgbm as lgb

folds = KFold(n\_splits=10, shuffle=True)

model = lgb.LGBMRegressor()

params = dict(

  n\_estimators = [100, 300, 500, 1000],

  num\_leaves = [8, 16, 32, 64, 128, 256, 512],

  colsample\_bytree = [0.2, 0.5, 0.7, 1],

  min\_data\_in\_leaf = [2, 3, 5, 7, 9, 11,]

)

search = GridSearchCV(estimator=model, param\_grid=params, cv=folds, scoring='neg\_mean\_squared\_error')

search.fit(x\_train, y\_train)

search.best\_params\_

search.best\_score\_