In [1]:
```
import numpy as np
import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
import category_encoders as ce
import warnings

from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import recall_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.feature_selection import SequentialFeatureSelector

warnings.simplefilter(action="ignore", category=FutureWarning)
```

Data Preparation

After our initial exploration and fine tuning of the business understanding, it is time to construct our final dataset prior to modeling. Here, we want to make sure to handle any integrity issues and cleaning, the engineering of new features, any transformations that we believe should happen (scaling, logarithms, normalization, etc.), and general preparation for modeling with `sklearn`.

In [2]:
```
vehicles = pd.read_csv('data/vehicles.csv')
```

In [3]:
```
vehicles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426880 entries, 0 to 426879
Data columns (total 18 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            426880 non-null  int64
 1   region        426880 non-null  object
 2   price         426880 non-null  int64
 3   year          425675 non-null  float64
 4   manufacturer  409234 non-null  object
 5   model         421603 non-null  object
 6   condition     252776 non-null  object
 7   cylinders     249202 non-null  object
 8   fuel          423867 non-null  object
 9   odometer      422480 non-null  float64
 10  title_status  418638 non-null  object
 11  transmission  424324 non-null  object
 12  VIN           265838 non-null  object
 13  drive         296313 non-null  object
 14  size          120519 non-null  object
 15  type          334022 non-null  object
 16  paint_color   296677 non-null  object
 17  state         426880 non-null  object
dtypes: float64(2), int64(2), object(14)
memory usage: 58.6+ MB
```

In [4]:
```
vehicles = vehicles.convert_dtypes()
original_row_count = vehicles.shape[0]
```

In [5]:
```
# CALC: % of null values
vehicles.isnull().sum()/vehicles.shape[0]*100
```

Out[5]:
```
id               0.000000
region           0.000000
price            0.000000
year             0.282281
manufacturer     4.133714
model            1.236179
condition       40.785232
cylinders       41.622470
fuel             0.705819
odometer         1.030735
title_status     1.930753
transmission     0.598763
VIN             37.725356
drive           30.586347
size            71.767476
type            21.752717
paint_color     30.501078
state            0.000000
dtype: float64
```

In [6]:
```
# remove a few features (columns) that are not relavent to the analysis
vehicles.drop(columns = ['id','region','VIN','state'], axis=1, inplace = Tru
```

In [7]:
```
# before dropping NaN's
px.imshow(vehicles.isnull())
```

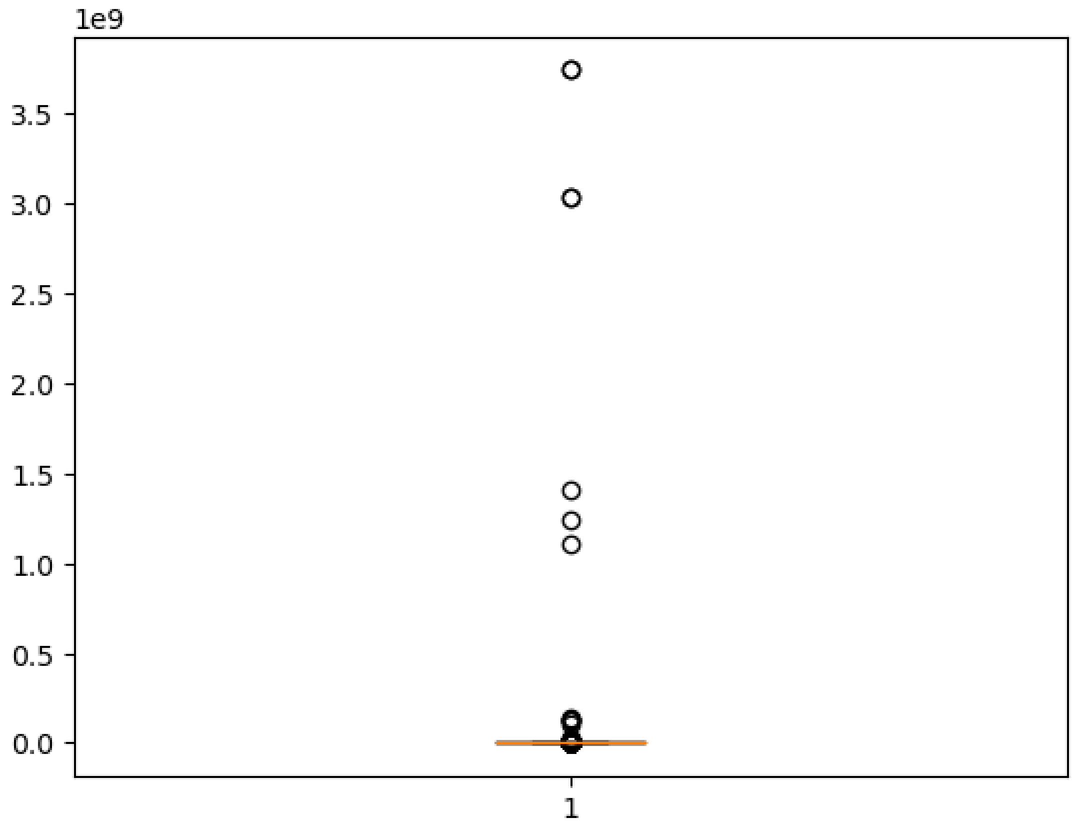
In [8]:
```
vehicles.select_dtypes(['Int64','float']).columns
```
Out[8]:
```
Index(['price', 'year', 'odometer'], dtype='object')
```
In [9]:
```
num_cols=['price', 'year', 'odometer']
```
In [10]:
```
vehicles.select_dtypes(['string']).columns
```
Out[10]:
```
Index(['manufacturer', 'model', 'condition', 'cylinders', 'fuel',
       'title_status', 'transmission', 'drive', 'size', 'type', 'paint_colo
r'],
      dtype='object')
```
In [11]:
```
obj_cols=['manufacturer', 'model', 'condition', 'cylinders', 'fuel',
        'title_status', 'transmission', 'drive', 'size', 'type', 'paint_color
```

Cleanup & Outlier Analysis

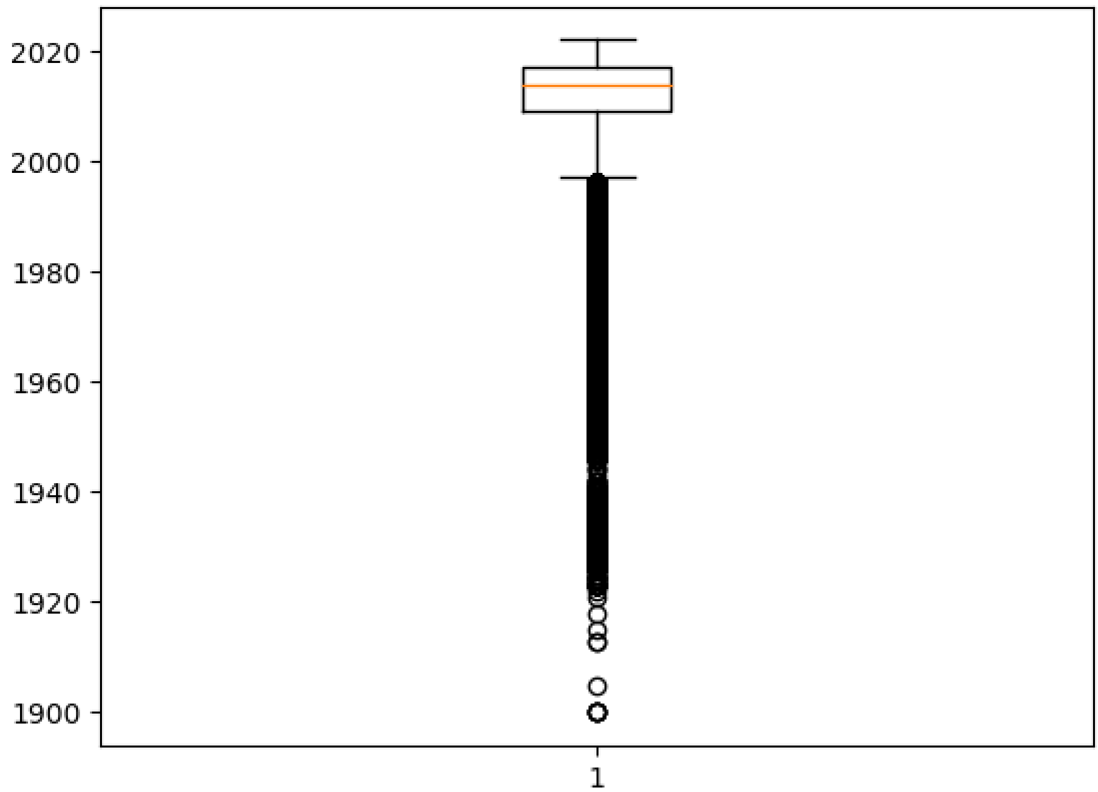
In [12]:
```
def remove_NaN_df(df, cols):
    for col in cols:
        df = df[df[col].notna()]

    return df
```
In [13]:
```
# removing NaN's from columns that dont carry a significant amount of NaN's
# improperly guessed, hence will remove those entires ( after careful analys
# errors - e.g. fuel or title_status
cols = ['year','odometer','manufacturer','model','fuel','title_status']
vehicles = remove_NaN_df(vehicles, cols)
```

Visualizations to understand current outliers

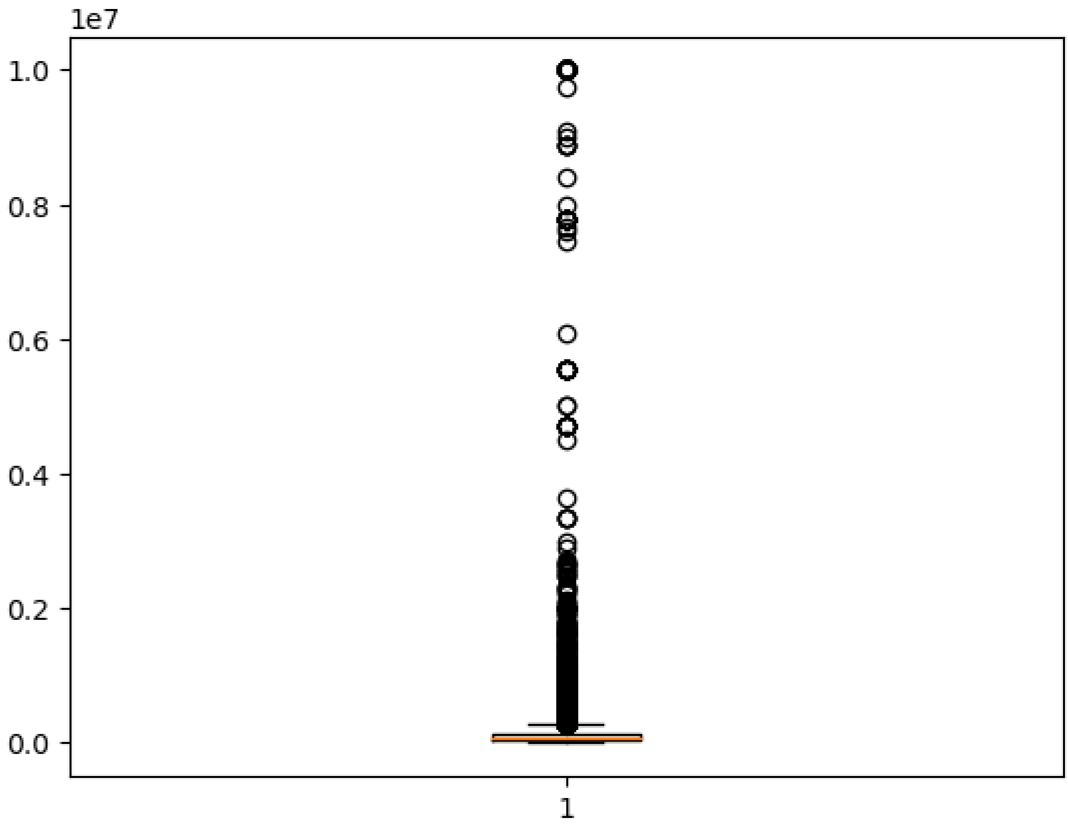
In [14]:
```
plt.boxplot(data=vehicles, x='price')
plt.show()
```



In [15]:
```
plt.boxplot(data=vehicles, x='year')
plt.show()
```



In [16]:
```
plt.boxplot(data=vehicles, x='odometer')
plt.show()
```



In [17]:
```
vehicles_df = vehicles.copy()
```

In [18]:
```
def find_boundaries(df, variable, distance):
    IQR = df[variable].quantile(0.75) - df[variable].quantile(0.25)
    lower_boundary = df[variable].quantile(0.25) - (IQR*distance)
    upper_boundary = df[variable].quantile(0.75) + (IQR*distance)

    return lower_boundary, upper_boundary
```

In [19]:
```
lo, up = find_boundaries(vehicles_df, 'price', 1.5)
outliers_p = np.where(vehicles_df['price'] > up, True,
                     np.where(vehicles_df['price'] < lo, True, False))
```

In [20]:
```
vehicles_df=vehicles_df.loc[~outliers_p]
```

In [21]:
```
lo, up = find_boundaries(vehicles_df, 'odometer', 1.5)
outliers_o = np.where(vehicles_df['odometer'] > up, True,
                     np.where(vehicles_df['odometer'] < lo, True, False))
```

In [22]:
```
vehicles_df=vehicles_df.loc[~outliers_o]
```

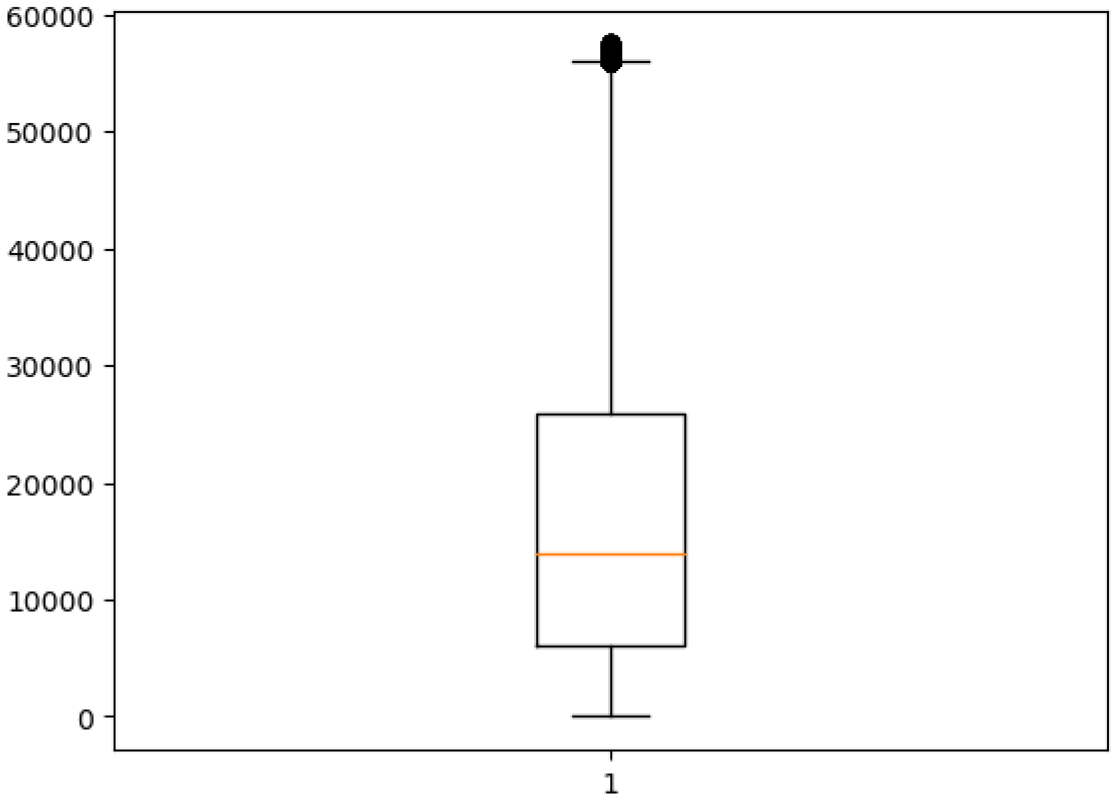
In [23]:
```
lo, up = find_boundaries(vehicles_df, 'odometer', 1.5)
outliers_y = np.where(vehicles_df['year'] > up, True,
                     np.where(vehicles_df['year'] < lo, True, False))
```

In [24]:
```
vehicles_df=vehicles_df.loc[~outliers_y]
```

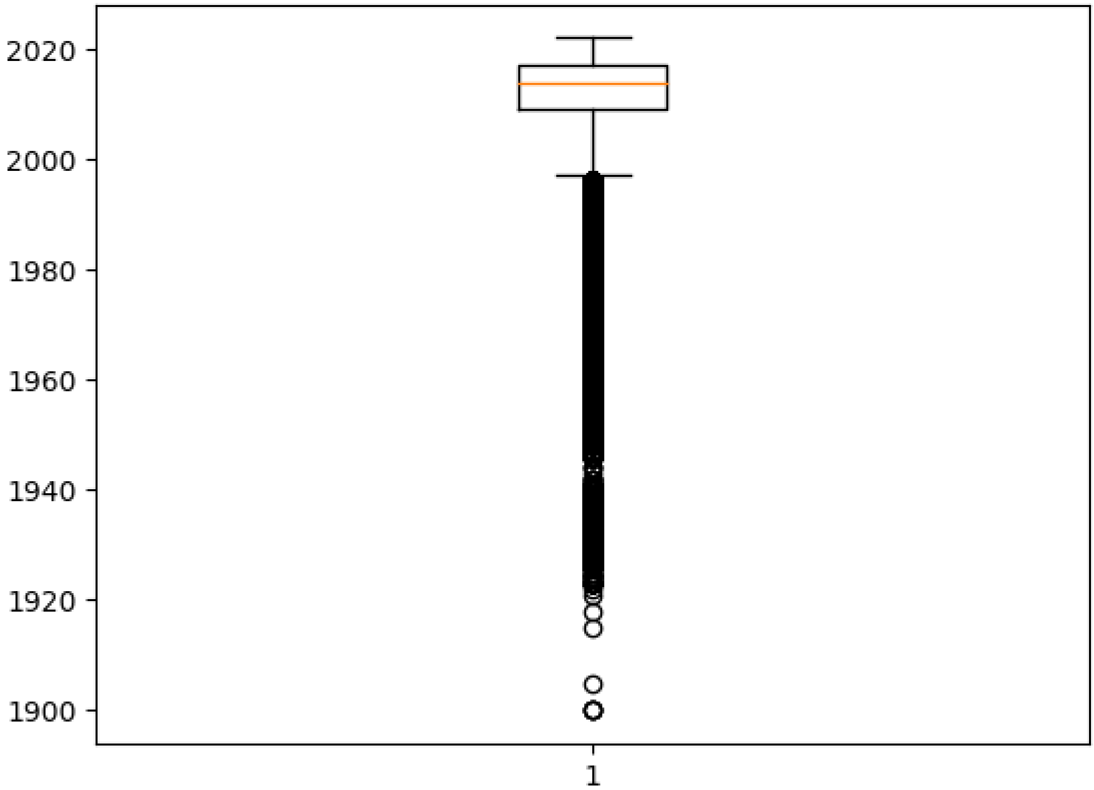
In [25]:
```
# remove 'parts only' from the title_status because this category offers no
title_status_values = ['parts only']
vehicles_df = vehicles_df[vehicles_df.title_status.isin(title_status_values)
```

Visualizations after adjusting outliers

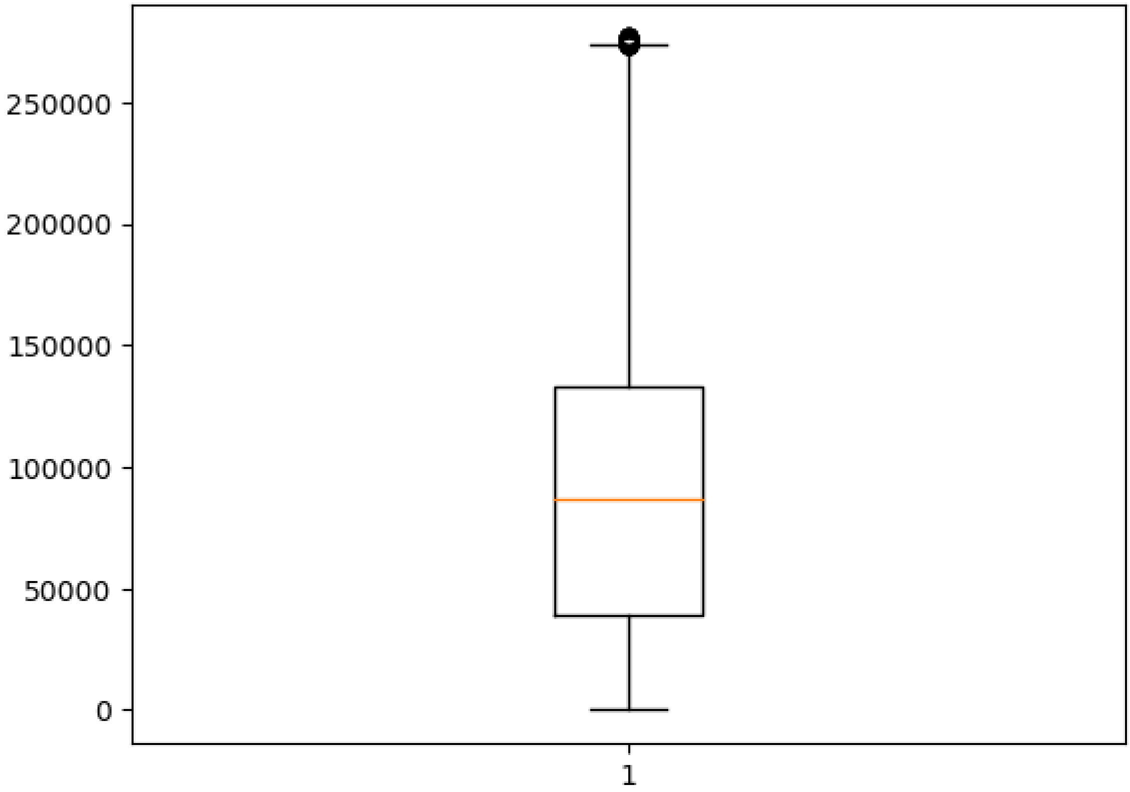
In [26]:
```
plt.boxplot(data=vehicles_df, x='price')
plt.show()
```



In [27]:
```
plt.boxplot(data=vehicles_df, x='year')
plt.show()
```



In [28]:
```
plt.boxplot(data=vehicles_df, x='odometer')
plt.show()
```



How much % of data removed?

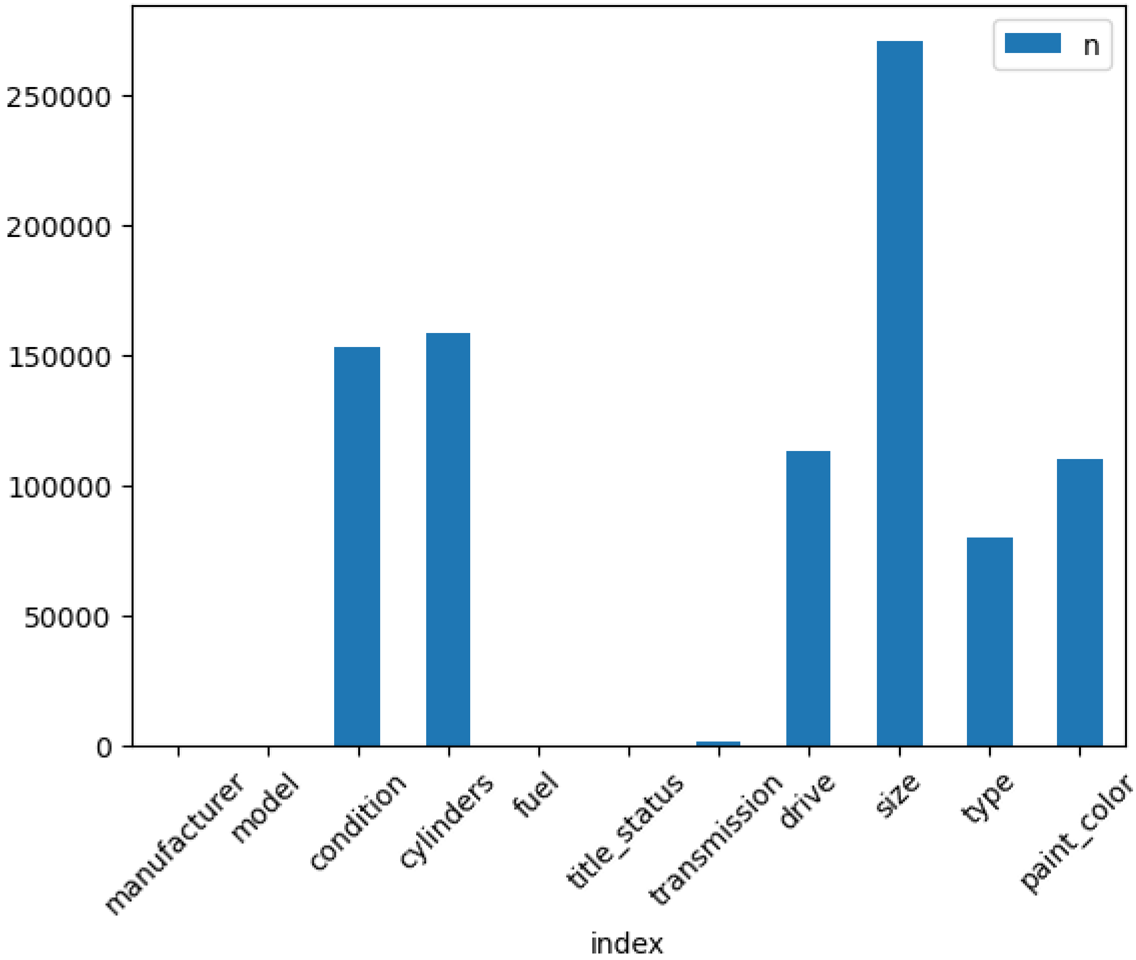
In [29]:
```
print('% of data removed ===>',((original_row_count-vehicles_df.shape[0])/(c
```

```
% of data removed ===> 10.72151424287856
```

Impute missing categorical values

In [30]:
```
# How many NaN's are in each categorical feature
dummy_df = vehicles_df[obj_cols].copy()
dummy_df.isna().sum().reset_index(name="n").plot.bar(x='index', y='n', rot=4

print(dummy_df.isna().sum().reset_index(name="n"))
```

```
           index       n
0   manufacturer       0
1          model       0
2      condition  153230
3      cylinders  158169
4           fuel       0
5   title_status       0
6   transmission    1476
7          drive  113227
8           size  270937
9           type   79912
10   paint_color  109917
```



In [31]:

```
# After all NaN's are removed, what's left need to be imputed.
px.imshow(vehicles_df.isnull())
```

In [32]:
```
# Use encoder to encode categorical features.
cols_to_enc = ['manufacturer','model','condition','cylinders','fuel','title_
X = vehicles_df.drop(columns=['price'], axis=1)
y = vehicles_df['price']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

encoder = ce.JamesSteinEncoder(cols=cols_to_enc)
X_train_enc = encoder.fit_transform(X_train, y_train)
X_test_enc = encoder.transform(X_test)
```

In [33]:
```
# final EDA on the cleaned data for insights - before moving on to modelling
vehicles[obj_cols].describe()
```

Out[33]:

	manufacturer	model	condition	cylinders	fuel	title_status	transmission	dr
count	391144	391144	232341	228432	391144	391144	389604	2754
unique	41	21892	6	8	5	6	3	
top	ford	f-150	good	6 cylinders	gas	clean	automatic	4
freq	68165	7821	115016	86855	330956	378675	309260	1239

Modeling

With your (almost?) final dataset in hand, it is now time to build some models. Here, you should build a number of different regression models with the price as the target. In building your models, you should explore different parameters and be sure to cross-validate your findings.

A Simple Linear Regression - with all features

In [34]:
```
%%time
all_features_linreg = ''
linreg_mse = ''

# keeping the intercept term to false
linreg_pipe = Pipeline([('scaler', StandardScaler()),
                        ('lreg', LinearRegression())]).fit(X_train_enc, y_tr
train_preds = linreg_pipe.predict(X_train_enc)
test_preds = linreg_pipe.predict(X_test_enc)

train_mse = mean_squared_error(y_train, train_preds)
test_mse = mean_squared_error(y_test, test_preds)

print(f'Linear Regression Train MSE: {np.around(train_mse,2)}')
print(f'Linear Regression Test MSE: {np.around(test_mse,2)}')

lr_coef = linreg_pipe.named_steps['lreg'].coef_
lr_intercept = linreg_pipe.named_steps['lreg'].intercept_
print(f'Intercept: {np.around(lr_intercept,2)}')

list_lr_coef = list((zip(linreg_pipe.named_steps['scaler'].get_feature_names
lr_coef_df = pd.DataFrame(list_lr_coef, columns = [' Features', 'Coefficient
lr_coef_df.sort_values(by='Coefficients', ascending=False, key=abs)
```

```
Linear Regression Train MSE: 69658567.49
Linear Regression Test MSE: 75504875.81
Intercept: 16644.24
CPU times: user 1.36 s, sys: 112 ms, total: 1.47 s
Wall time: 683 ms
```

Out[34]:

	Features	Coefficients
2	model	7622.099721
6	odometer	-3070.291758
0	year	1456.474888
8	transmission	-1162.837167
11	type	814.109935
9	drive	630.819201
5	fuel	602.289175
7	title_status	343.353569
1	manufacturer	329.942694
4	cylinders	197.824047
10	size	153.179612
12	paint_color	134.375861
3	condition	-37.639819

Observation-Simple Linear Regression

fit_intercept is false:

1. Train MSE: 351884283.44
2. Test MSE: 353533750.09
3. Intercept: 0.0

fit_intercept is True:

1. Train MSE: 66852359.27
2. Test MSE: 71854716.44
3. Intercept: 16882.89

Theory: *A positive coefficient indicates that as the value of the independent variable increases, the mean of the dependent variable also tends to increase. A negative coefficient suggests that as the independent variable increases, the dependent variable tends to decrease*

At this stage we can draw a quick inference by looking at the coefficients that ones that have a negative affect on the price are odometer, transmission & condition. The more the odometer, the cheaper is the car & so goes with the condition (old is less expensive). Model has the most impact on the price followed by the year of the car. Newer makes are more expensive

Ridge Regression using GridSearchCV

In [35]:
```python
ridge_pipe = Pipeline([('scaler', StandardScaler()), ('ridge', Ridge())])
param_dict = {'ridge__alpha': [0.001, 0.1, 1.0, 10.0, 100.0, 1000.0]}
```

In [36]:
```python
%%time
r_grid = ''
ridge_train_mse = ''
ridge_test_mse = ''
ridge_best_alpha = ''

r_grid = GridSearchCV(ridge_pipe, param_grid=param_dict).fit(X_train_enc, y_

train_preds = r_grid.predict(X_train_enc)
test_preds = r_grid.predict(X_test_enc)

ridge_train_mse = mean_squared_error(y_train, train_preds)
ridge_test_mse = mean_squared_error(y_test, test_preds)
ridge_best_alpha = r_grid.best_params_

print(f'Ridge Regression Train MSE: {np.around(ridge_train_mse,2)}')
print(f'Ridge Regression Test MSE: {np.around(ridge_test_mse,2)}')
print(f'Best Alpha: {list(ridge_best_alpha.values())[0]}')
```

```
Ridge Regression Train MSE: 69658567.63
Ridge Regression Test MSE: 75504662.48
Best Alpha: 10.0
CPU times: user 30.4 s, sys: 2.8 s, total: 33.2 s
Wall time: 14.4 s
```

Observation-Ridge Regression

1. *Ridge Regression Train MSE: 66852359.42*
2. *Ridge Regression Test MSE: 71854450.24*
3. *Best Alpha: 10.0*

In [37]:
```python
ridge_coef_list = []

# for best alpha = 10 find out all the coeffs ( captured in the ridge_best_a
ridge_pipe_4best_alpha = Pipeline([('scaler', StandardScaler()), ('ridge', R
ridge_pipe_4best_alpha.fit(X_train_enc, y_train)

ridge_coef_list.append(list(ridge_pipe_4best_alpha.named_steps['ridge'].coef
len(ridge_coef_list)
print('For alpha = 10 we have the following coefficients:')
list(zip(X_train_enc.columns, ridge_coef_list[-1]))

ridge_coef_df = pd.DataFrame(list(zip(X_train_enc.columns, ridge_coef_list[-
ridge_coef_df.sort_values(by='Coefficients', ascending=False, key=abs)
```

```
For alpha = 10 we have the following coefficients:
```

Out[37]:

	Features	Coefficients
2	model	7621.598761
6	odometer	-3070.233381
0	year	1456.499020
8	transmission	-1162.557522
11	type	814.154869
9	drive	630.882620
5	fuel	602.298276
7	title_status	343.347012
1	manufacturer	330.043513
4	cylinders	197.862226
10	size	153.196668
12	paint_color	134.395734
3	condition	-37.612925

At this stage, with the best alpha (10), Ridge Regression gives us almost similar results as a simple linear regression. We can draw a quick inference by looking at the coefficients that ones that have a negative affect on the price are odometer, transmission & condition, similar to LR model above. Model & Year have positive affect on the price of the used car vehicle

LASSO Regression

In [38]:
```python
%%time
lasso_grid = ''
lasso_train_mse = ''
lasso_test_mse = ''
lasso_coefs = ''

lasso_pipe = Pipeline([('scaler', StandardScaler()),
                       ('lasso', Lasso(random_state = 42))]).fit(X_train_enc


train_preds = lasso_pipe.predict(X_train_enc)
test_preds = lasso_pipe.predict(X_test_enc)
```

```python
lasso_train_mse = mean_squared_error(y_train, train_preds)
lasso_test_mse = mean_squared_error(y_test, test_preds)
lasso_coefs = lasso_pipe.named_steps['lasso'].coef_

feature_names = X_train_enc.columns
lasso_df = pd.DataFrame({'feature': feature_names, 'Coefficients': lasso_coe

print(f'LASSO Train MSE: {np.around(lasso_train_mse,2)}')
print(f'LASSO Test MSE: {np.around(lasso_test_mse,2)}')

lasso_df.sort_values(by='Coefficients', ascending=False, key=abs)
```

```
LASSO Train MSE: 69658578.85
LASSO Test MSE: 75505613.41
CPU times: user 1.86 s, sys: 111 ms, total: 1.97 s
Wall time: 685 ms
```

Out[38]:

	feature	Coefficients
2	model	7621.356737
6	odometer	-3068.927061
0	year	1455.584808
8	transmission	-1160.433138
11	type	813.623405
9	drive	630.761472
5	fuel	601.373002
7	title_status	342.383635
1	manufacturer	329.691349
4	cylinders	197.276948
10	size	152.374815
12	paint_color	133.661073
3	condition	-36.411221

Observation-LASSO

1. *LASSO Train MSE: 69658578.85*
2. *LASSO Test MSE: 75505613.41*

LASSO Regression gives us the same results as the previous 2 regression models with respect to the behvior of the best features with the target

SFS - To identify a list of features that have the most influence on the price

In [39]:
```python
sfs_lr_pipe = Pipeline([('scaler', StandardScaler()),
                        ('selector', SequentialFeatureSelector(LinearRegress
                        ('lr_model', LinearRegression())])
```

In [40]:
```python
%%time
param_dict = {}
sfs_lr_grid = ''
sfs_lr_train_mse = ''
```

```
sfs_lr_test_mse = ''

param_dict = {'selector__n_features_to_select': [4, 5, 6]}
sfs_lr_grid = GridSearchCV(sfs_lr_pipe, param_grid=param_dict).fit(X_train_e

train_preds = sfs_lr_grid.predict(X_train_enc)
test_preds = sfs_lr_grid.predict(X_test_enc)

sfs_lr_train_mse = mean_squared_error(y_train, train_preds)
sfs_lr_test_mse = mean_squared_error(y_test, test_preds)

print(f'Minimum Train MSE is : {np.around(sfs_lr_train_mse,2)}')
print(f'Minimum Test MSE is: {np.around(sfs_lr_test_mse,2)}')
```

```
Minimum Train MSE is : 70350171.77
Minimum Test MSE is: 76482069.99
CPU times: user 8min 46s, sys: 30.4 s, total: 9min 16s
Wall time: 3min 3s
```

In [41]:
```
best_estimator = ''
best_selector = ''
best_model = ''
feature_names = ''
coefs = ''

best_estimator = sfs_lr_grid.best_estimator_
best_selector = best_estimator.named_steps['selector']
best_model = sfs_lr_grid.best_estimator_.named_steps['lr_model']
feature_names = X_train_enc.columns[best_selector.get_support()]
coefs = best_model.coef_

print(best_estimator)
print(f'Features from best selector: {feature_names}.')
print('Coefficient values: ')
print('===================')
pd.DataFrame([coefs.T], columns = feature_names, index = ['lr_model'])
```

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('selector',
                 SequentialFeatureSelector(estimator=LinearRegression(),
                                           n_features_to_select=6)),
                ('lr_model', LinearRegression())])
Features from best selector: Index(['year', 'model', 'odometer', 'transmissi
on', 'drive', 'type'], dtype='object').
Coefficient values: 
===================
```

Out[41]:

	year	model	odometer	transmission	drive	type
lr_model	1487.65941	7933.031114	-2983.766478	-1062.939404	770.644025	943.642679

Prepare encoded data down to the list of top 6 features identified above.

In [42]:
```
top_features = ['year','model','odometer','transmission','drive','type']

X_top_train_enc = X_train_enc[top_features]
X_top_test_enc = X_test_enc[top_features]

X_top_train_enc.shape, X_top_test_enc.shape
```

Out[42]:
```
((285834, 6), (95278, 6))
```

Polynomial Degree & Linear Regression --- To identify the best degree for the features identified above

In [43]:
```
%%time
polyd_lr_train_mses = []
polyd_lr_test_mses = []

best_polyd = ''

for i in range(1, 3):
    pipe = Pipeline([('pfeat', PolynomialFeatures(degree = i, include_bias=F
                    ('scale', StandardScaler()),
                    ('linreg', LinearRegression())]).fit(X_top_train_enc, y

    train_preds = pipe.predict(X_top_train_enc)
    test_preds = pipe.predict(X_top_test_enc)
    polyd_lr_train_mses.append(mean_squared_error(y_train, train_preds))
    polyd_lr_test_mses.append(mean_squared_error(y_test, test_preds))

best_polyd_test = polyd_lr_test_mses.index(min(polyd_lr_test_mses)) + 1

print(f'Train MSE is: {np.around(polyd_lr_train_mses,2)}')
print(f'Test MSE is: {np.around(polyd_lr_test_mses,2)}')
best_polyd_train = polyd_lr_train_mses.index(min(polyd_lr_train_mses)) + 1
best_polyd_test = polyd_lr_test_mses.index(min(polyd_lr_test_mses)) + 1

print(f'Best TRAIN performing degree model : {best_polyd_train}')
print(f'Best TEST performing degree model : {best_polyd_test}')
```

```
Train MSE is: [70350171.77 66300546.84]
Test MSE is: [76482069.99 72068495.72]
Best TRAIN performing degree model : 2
Best TEST performing degree model : 2
CPU times: user 2.71 s, sys: 164 ms, total: 2.87 s
Wall time: 1.08 s
```

Polynomial with Degree = 2 (best degree) & Ridge Regression (to identify best alpha ... will it change?)

In [44]:
```
%%time
pd_ridge_pipe = Pipeline([('poly_features', PolynomialFeatures(degree = 2, i
                          ('scaler', StandardScaler()),
                          ('ridge', Ridge())])
param_dict = {'ridge__alpha': [0.001, 0.1, 1.0, 10.0, 100.0, 1000.0]}

pd_ridge_grid = ''
pd_ridge_train_mse = ''
pd_ridge_test_mse = ''
pd_ridge_best_alpha = ''

pd_ridge_grid = GridSearchCV(pd_ridge_pipe, param_grid=param_dict).fit(X_top

train_preds = pd_ridge_grid.predict(X_top_train_enc)
test_preds = pd_ridge_grid.predict(X_top_test_enc)

pd_ridge_train_mse = mean_squared_error(y_train, train_preds)
pd_ridge_test_mse = mean_squared_error(y_test, test_preds)
pd_ridge_best_alpha = pd_ridge_grid.best_params_

print(f'Polynomial with Degree =2 & Ridge Regression Train MSE: {np.around(p
print(f'Polynomial with Degree =2 & Ridge Regression Test MSE: {np.around(pc
print(f'Best Alpha: {list(pd_ridge_best_alpha.values())[0]}')
```

```
Polynomial with Degree =2 & Ridge Regression Train MSE: 66300547.22
Polynomial with Degree =2 & Ridge Regression Test MSE: 72068615.15
Best Alpha: 0.001
CPU times: user 27.5 s, sys: 3.09 s, total: 30.6 s
Wall time: 12.4 s
```

LASSO Regression with Degree = 2

In [45]:

```
pd_lasso_pipe = Pipeline([('polyfeatures', PolynomialFeatures(degree = 2, in
                          ('scaler', StandardScaler()),
                          ('lasso', Lasso(random_state = 42))]).fit(X_top_tr

train_preds = pd_lasso_pipe.predict(X_top_train_enc)
test_preds = pd_lasso_pipe.predict(X_top_test_enc)

lasso_train_mse = mean_squared_error(y_train, train_preds)
lasso_test_mse = mean_squared_error(y_test, test_preds)
lasso_coefs = pd_lasso_pipe.named_steps['lasso'].coef_

pd_lasso_coefs = pd_lasso_pipe.named_steps['lasso'].coef_
feature_names = X_train_enc.columns

print(f'LASSO Train MSE: {np.around(lasso_train_mse,2)}')
print(f'LASSO Test MSE: {np.around(lasso_test_mse,2)}')

list_lasso_coeff = list((zip(pd_lasso_pipe.named_steps['polyfeatures'].get_f
                             pd_lasso_pipe.named_steps['lasso'].coef_)))
pd_lasso_df = pd.DataFrame(list_lasso_coeff, columns = [' Features', 'Lasso
pd_lasso_df.sort_values(by='Lasso Coefficients', ascending=False, key=abs)
```

```
/Users/vandavilli/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_m
odel/_coordinate_descent.py:647: ConvergenceWarning:

Objective did not converge. You might want to increase the number of iterati
ons, check the scale of the features or consider increasing regularisation.
Duality gap: 6.964e+12, tolerance: 4.921e+09
```

```
LASSO Train MSE: 68705604.07
LASSO Test MSE: 74851193.62
```

Out[45]:

	Features	Lasso Coefficients
1	model	6727.396960
4	drive	-6614.846803
24	drive^2	5747.610678
25	drive type	5099.018127
3	transmission	4186.530353
23	transmission type	-4161.744861
2	odometer	-4084.277669
6	year^2	3171.288505
8	year odometer	2842.250411
22	transmission drive	-2746.414512
15	model drive	2723.363969
19	odometer drive	-2475.445868
18	odometer transmission	1748.972547
0	year	-1720.900867
12	model^2	-1430.966379
13	model odometer	-1207.786945
16	model type	1174.460232
20	odometer type	-1144.787618
17	odometer^2	941.801750
21	transmission^2	-338.182407
9	year transmission	304.217410
14	model transmission	-285.556073
5	type	-253.131590
26	type^2	133.669882
11	year type	0.000000
10	year drive	-0.000000
7	year model	0.000000

In [46]:

```
feature_names = pd_lasso_pipe.named_steps['polyfeatures'].get_feature_names_
coefs = pd_lasso_pipe.named_steps['lasso'].coef_

print(best_estimator)
print('Coefficient values: ')
print('===================')
errors = pd.DataFrame([coefs.T], columns = feature_names, index = ['lr_model
errors[errors.columns[(abs(errors) > 0.000001).any()]]
```

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('selector',
                 SequentialFeatureSelector(estimator=LinearRegression(),
                                           n_features_to_select=6)),
                ('lr_model', LinearRegression())])
Coefficient values: 
===================
```

Out[46]:

	year	model	odometer	transmission	drive	type
lr_model	-1720.900867	6727.39696	-4084.277669	4186.530353	-6614.846803	-253.13159

1 rows × 24 columns

Evaluation

With some modeling accomplished, we aim to reflect on what we identify as a high quality model and what we are able to learn from this. We should review our business objective and explore how well we can provide meaningful insight on drivers of used car prices. Your goal now is to distill your findings and determine whether the earlier phases need revisitation and adjustment or if you have information of value to bring back to your client.

best fitting model - (Linear Regression degree 2)

In [47]:

```
from sklearn.inspection import permutation_importance
from sklearn import metrics
from sklearn.inspection import permutation_importance
pipe = Pipeline([('pfeat', PolynomialFeatures(degree = 2, include_bias=False
                      ('scale', StandardScaler()),
                      ('linreg', LinearRegression())]).fit(X_top_train_enc, y
train_preds = pipe.predict(X_top_train_enc)
test_preds = pipe.predict(X_top_test_enc)

metrics.mean_squared_error(y_test, test_preds, squared = False)
```

Out[47]: 8489.316563785678

Permutation Feature Importance with best performing model

In [48]:

```
r = permutation_importance(pipe, X_top_test_enc, y_test,
                           random_state=123)
pd.DataFrame({"Variables":X_top_test_enc.columns,"Score":r.importances_mean}
```

Out[48]:

	Variables	Score
1	model	0.595140
0	year	0.092802
2	odometer	0.058063
4	drive	0.021898
3	transmission	0.020292
5	type	0.017435

Recommendations

Now that we've settled on our models and findings, it is time to deliver the information to the client. You should organize your work as a basic report that details your primary *findings. Keep in mind that your audience is a group of used car dealers interested in fine tuning their inventory.*

Visualizations on the top selected features {model, year, odometer, drive, transmission, type}

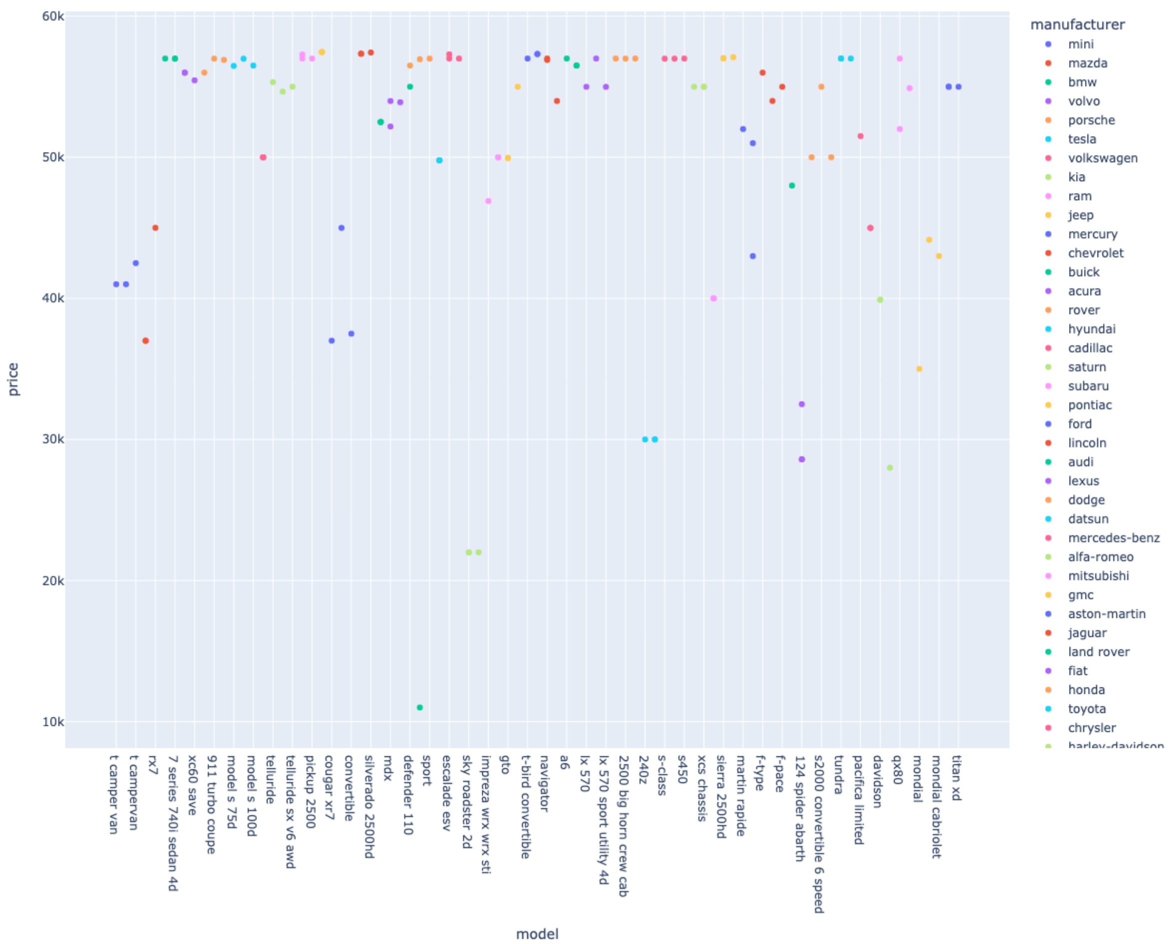
In [49]:
```
top_cols=['model','manufacturer','drive','transmission','type']
vehicles_df[top_cols].describe()
```

Out[49]:

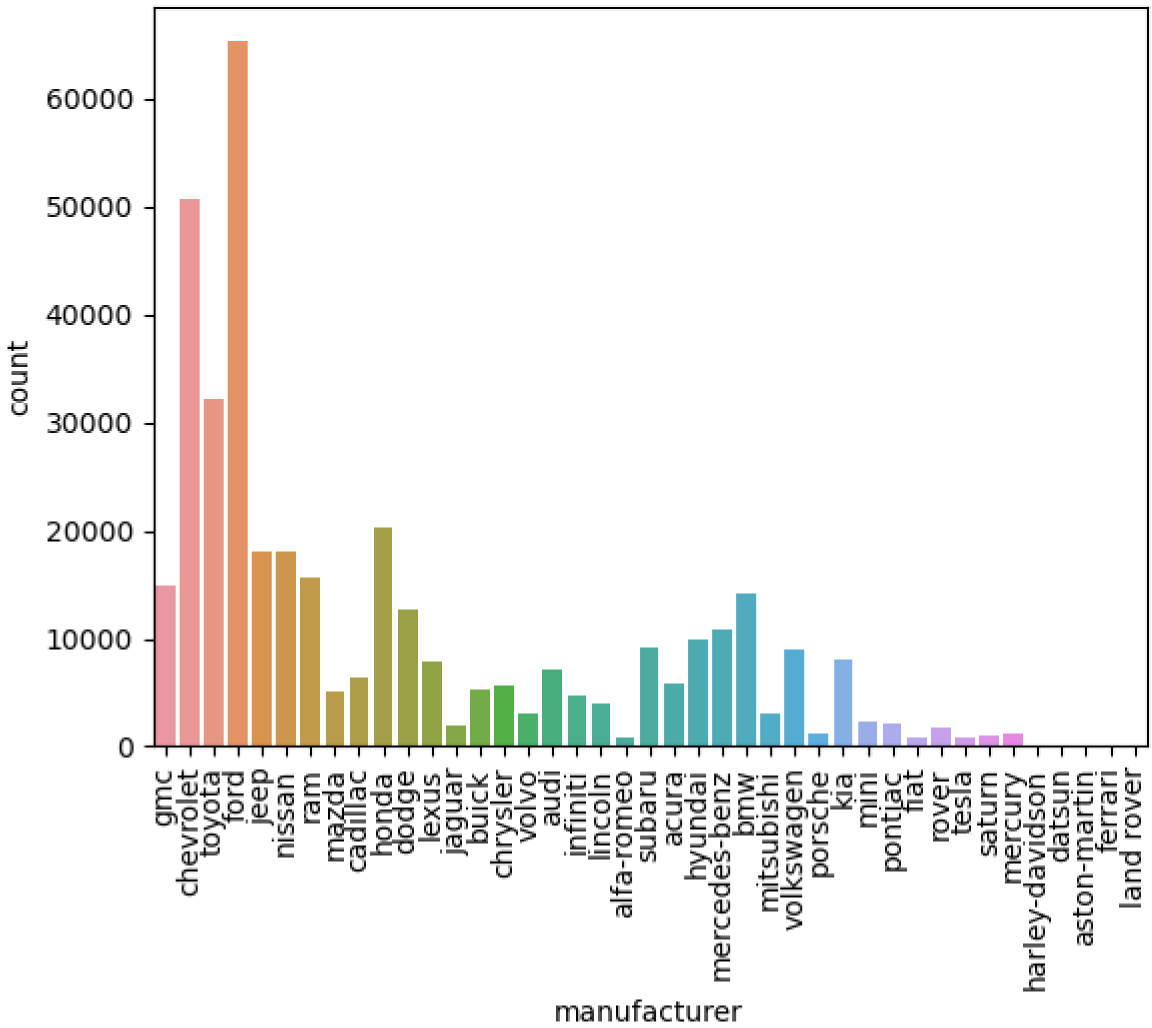
	model	manufacturer	drive	transmission	type
count	381112	381112	267885	379636	301200
unique	21107	41	3	3	13
top	f-150	ford	4wd	automatic	sedan
freq	7649	65219	118274	300466	80844

In [50]:
```
# for top 3 models among each manufacturer
N = 3
msk = vehicles_df.groupby('manufacturer')['price'].rank(method='first', asce
models_df = vehicles_df[msk]
```

In [51]:
```
fig = px.scatter(models_df,x='model',y='price',color='manufacturer',width=12
fig.show("png")
```

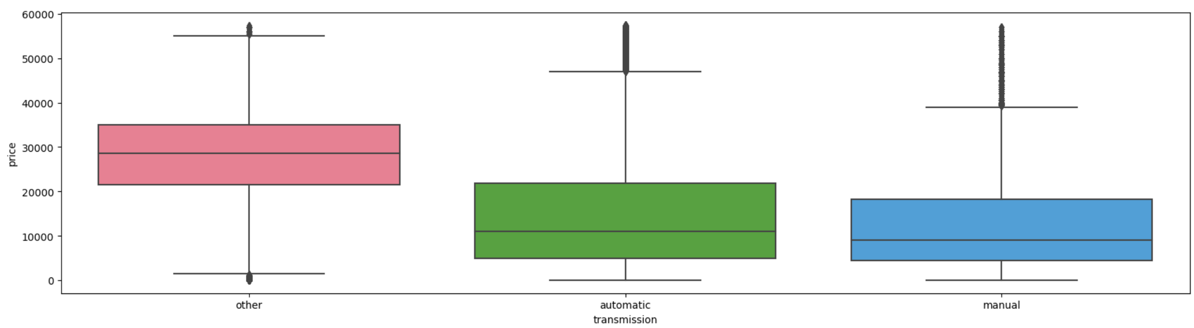


In [52]:
```
#Inventory layout
sns.countplot(data = vehicles_df, x = "manufacturer")
plt.xticks(rotation = 90);
```



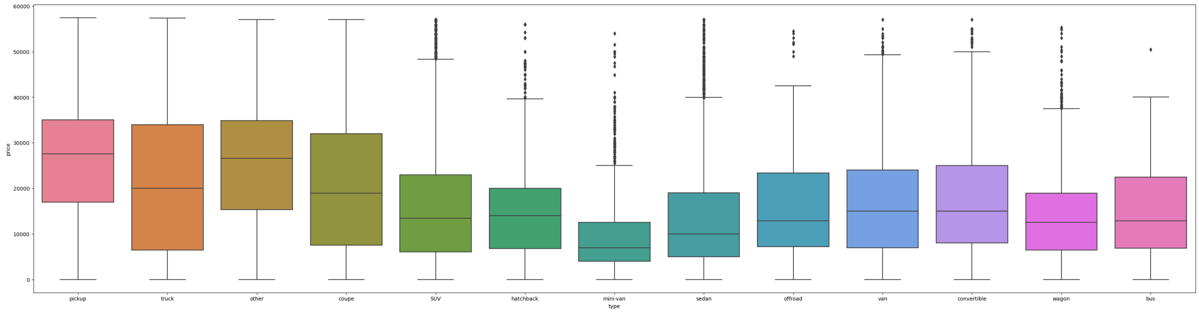
In [53]:
```
# transmission
plt.figure(figsize=(20,5))
sns.boxplot(x = vehicles_df['transmission'], y = vehicles_df['price'], palet
```

Out[53]: <AxesSubplot:xlabel='transmission', ylabel='price'>



In [54]:
```
# type
plt.figure(figsize=(40,10))
sns.boxplot(x = vehicles_df['type'], y = vehicles_df['price'], palette = 'hu
```

Out[54]: <AxesSubplot:xlabel='type', ylabel='price'>



price
60000
50000
40000
30000
20000
10000
0
pickup
truck
other
coupe
SUV
hatchback
mini-van
sedan
offroad
van
convertible
wagon
bus
type