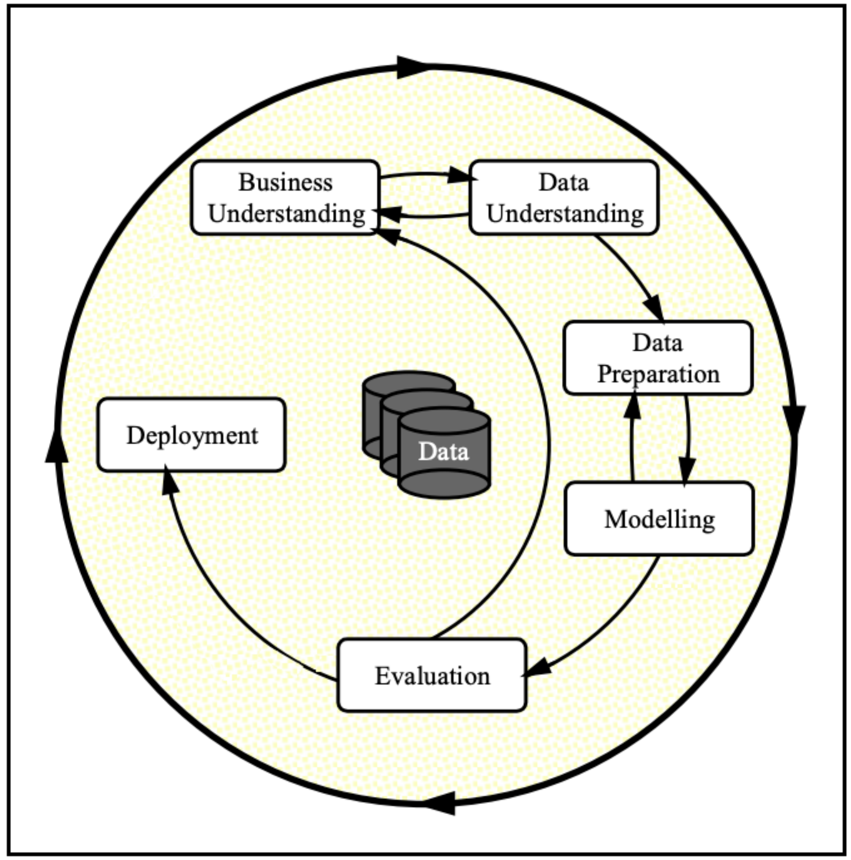
What drives the price of a car?

OVERVIEW

In this application, you will explore a dataset from kaggle. The original dataset contained information on 3 million used cars. The provided dataset contains information on 426K cars to ensure speed of processing. Your goal is to understand what factors make a car more or less expensive. As a result of your analysis, you should provide clear recommendations to your client -- a used car dealership -- as to what consumers value in a used car.

CRISP-DM Framework



To frame the task, throughout our practical applications we will refer back to a standard process in industry for data projects called CRISP-DM. This process provides a framework for working through a data problem. Your first step in this application will be to read through a brief overview of CRISP-DM [here (https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_11/readings_starter.zip)](https://mo-pcco.s3.us-east-1.amazonaws.com/BH-PCMLAI/module_11/readings_starter.zip). After reading the overview, answer the questions below.

Business Understanding

From a business perspective, we are tasked with identifying key drivers for used car prices. In the CRISP-DM overview, we are asked to convert this business framing to a data problem definition. Using a few sentences, reframe the task as a data task with the appropriate technical vocabulary.

1.1) Business Question

Given a dataset containing used cars attributes, identify the key drivers behind prices of used cars.

1.2) Understanding of Business

In order to estimate the price of used car, the deal would be interested in predicting the price of a car based on its attributes. Answers to the following questions may help the dealer determine the price of the used car.

1) Do used cars with less age cost more/less?

2) Do used cars with electric fuel cost more?

3) List top 5 states with highest used car sales.

4) Do used cars with automatic transmission cost more?

5) Which fuel type is preferred more compared to the other?

6) Do used cars with more cylinders cost more?

7) Which make and type are most predominant in the used car market?

8) Which transmission type is the most sold?

Data Understanding

After considering the business understanding, we want to get familiar with our data. Write down some steps that you would take to get to know the dataset and identify any quality issues within. Take time to get to know the dataset and explore what information it contains and how this could be used to inform your business understanding.

In order to understand the data, some required libraries are imported to run few basic stats about the data. Navigating through the dataset using shape to determine the number of rows and columns and using the describe and info function to determine the mean,std and count along with the type of columns will help in next steps.

In [ ]:

```

```

2.1) Data Collection - Imports

In [1]:
```
import pandas as pd
import numpy as np
import plotly.express as px
import matplotlib.pyplot     as plt
import seaborn               as sns
import matplotlib.image      as mpimg
import plotly.graph_objects as go
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, OrdinalEncoder
import category_encoders as ce
from sklearn.inspection import permutation_importance

from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,Ridge, Lasso
from sklearn.metrics import mean_squared_error, median_absolute_error, mean_absolute_error, mean_absolute_percen
from sklearn.pipeline import Pipeline

from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.compose import  make_column_transformer,TransformedTargetRegressor
from sklearn.model_selection import GridSearchCV, cross_val_score,KFold

from sklearn.feature_selection import RFE

from scipy.special import exp10

import warnings
warnings.filterwarnings("ignore")
```

2.2 Data Collection - Load Data

In [2]:
```
#Make a copy of the dataset to make it easier to reload the data in case there is a problem with the project.
df_vehicles = pd.read_csv('Data/vehicles.csv')
data_csv = df_vehicles.copy()
```

In [3]:
```
print('Exact number of entries/rows in vehicles: {}'.format(data_csv.shape[0]))
print('Number of features/columns in vehicles: {}'.format(data_csv.shape[1]))
print('Feature/Column-names in vehicles: {}'.format(data_csv.columns.values))
```

```
Exact number of entries/rows in vehicles: 426880
Number of features/columns in vehicles: 18
Feature/Column-names in vehicles: ['id' 'region' 'price' 'year' 'manufacturer' 'model' 'condition'
 'cylinders' 'fuel' 'odometer' 'title_status' 'transmission' 'VIN' 'drive'
 'size' 'type' 'paint_color' 'state']
```

2.3 Data Collection - Describe Data

In [4]:
```
data_csv.describe()
```

Out[4]:

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

2.4 Data Collection - Data Types

In [5]:
```
data_csv.dtypes
```

Out[5]:
```
id                 int64
region            object
price              int64
year             float64
manufacturer      object
model             object
condition         object
cylinders         object
fuel              object
odometer         float64
title_status      object
transmission      object
VIN               object
drive             object
size              object
type              object
paint_color       object
state             object
dtype: object
```

2.5 Data Collection - Data Dimensions

In [6]:
```
data_csv.shape
```

Out[6]:
```
(426880, 18)
```

2.6 Data Collection - Check NA

In [7]:
```
data_csv.isna().sum()
```

Out[7]:
```
id                   0
region               0
price                0
year              1205
manufacturer     17646
model             5277
condition       174104
cylinders       177678
fuel              3013
odometer          4400
title_status      8242
transmission      2556
VIN             161042
drive           130567
size            306361
type             92858
paint_color     130203
state                0
dtype: int64
```

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`.

3.1 Numerical and Categorical Attributes

In [8]:
```
num_attributes = data_csv.select_dtypes( include=['float64','int64'] )
cat_attributes = data_csv.select_dtypes( exclude=['float64', 'datetime64[ns]'] )
num_attributes.sample()
```

Out[8]:

	id	price	year	odometer
341626	7317041317	14500	2008.0	165000.0

In [9]:
```
cat_attributes.sample()
```

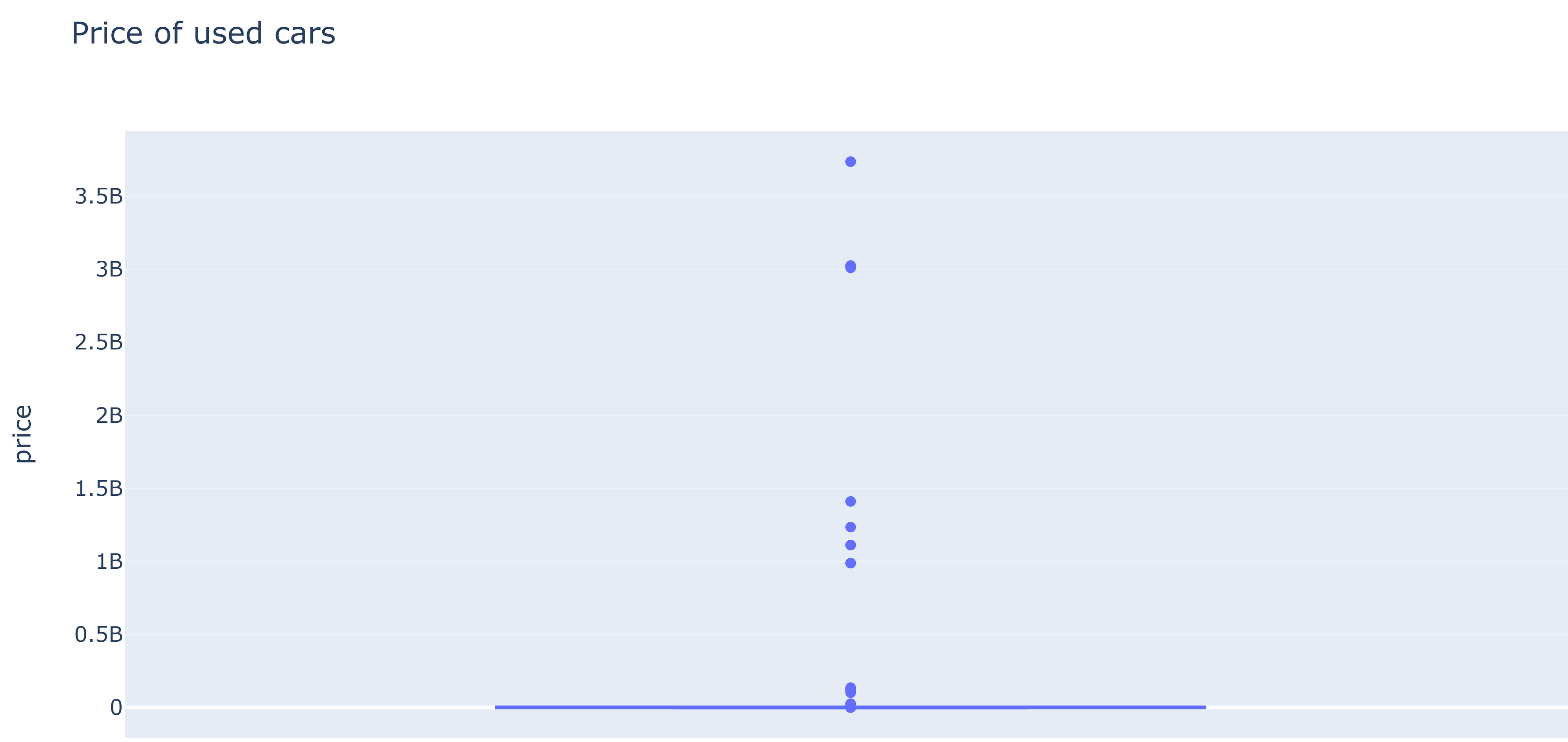
Out[9]:

	id	region	price	manufacturer	model	condition	cylinders	fuel	title_status	transmission	VIN	drive	size	type
327749	7307522928	salem	48	toyota	camry	NaN	6 cylinders	gas	clean	automatic	JT2BF28K4W0088786	fwd	NaN	sedan

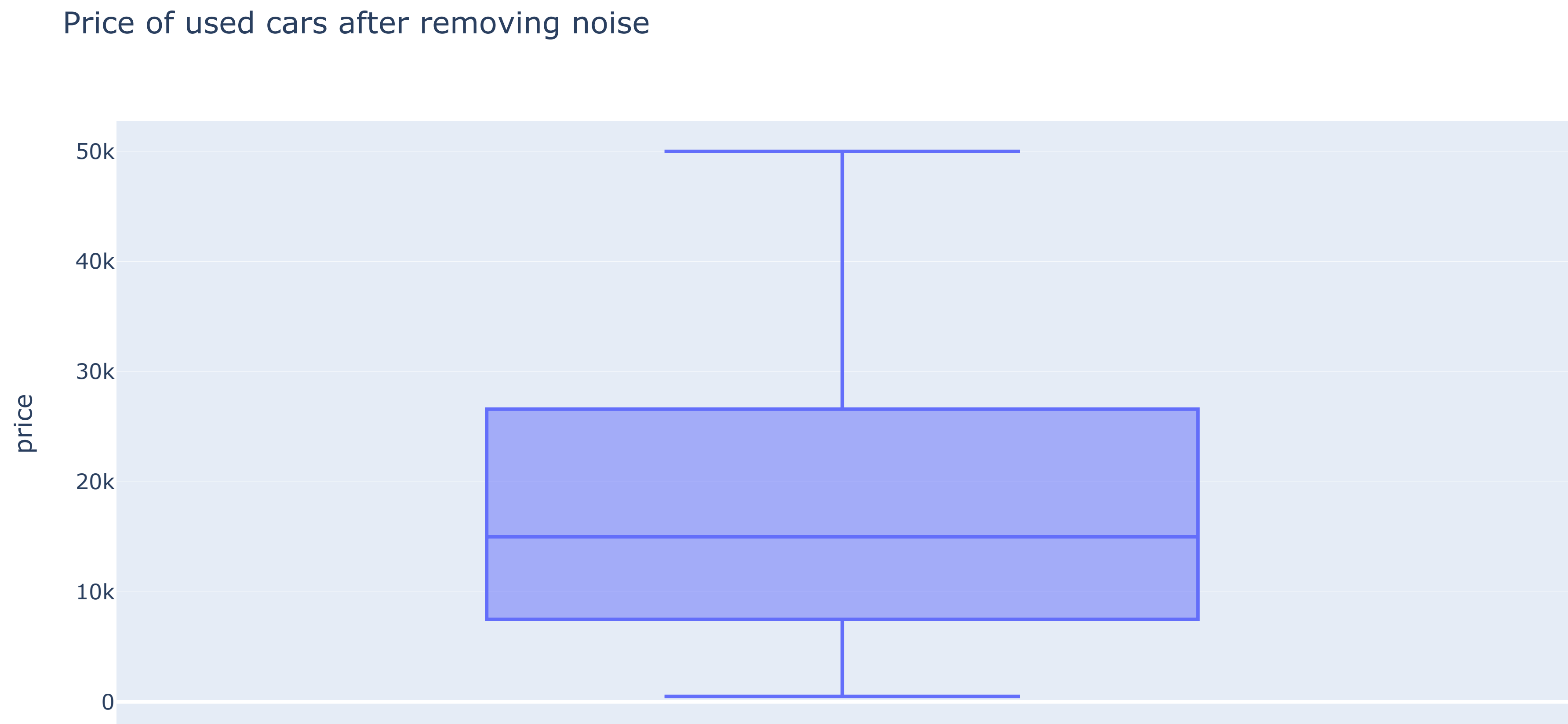
In [ ]:

3.2 Clean Dataset by dropping rows and columns

In [10]:
```
#sns.boxplot( data_csv['price'] )
px.box(data_csv, y="price", title="Price of used cars")
```



In [11]:
```
#From the above we see that outliers exist. For now, just remove the noise.
data_csv = data_csv[(data_csv['price']>500) & (data_csv['price']<50000)]
#sns.boxplot( data_csv['price'] )
px.box(data_csv, y="price", title="Price of used cars after removing noise")
```



In [12]:
```
cat_attributes.apply( lambda x: x.unique().shape[0])
```

Out[12]:
```
id              426880
region             404
price            15655
manufacturer        43
model            29650
condition            7
cylinders            9
fuel                 6
title_status         7
transmission         4
VIN             118247
drive                4
size                 5
type                14
paint_color         13
state               51
dtype: int64
```

After dropping the noise, lets check the shape

In [13]:
```
data_csv.shape
```

Out[13]:
```
(370724, 18)
```

In [14]:
```
data_csv.isna().sum()
```

Out[14]:
```
id                   0
region               0
price                0
year               863
manufacturer     14146
model             4045
condition       136790
cylinders       150132
fuel              2509
odometer          2009
title_status      6667
transmission      1730
VIN             144685
drive           113470
size            265007
type             79520
paint_color     106731
state                0
dtype: int64
```

From the above, we see a lot of Nan values. We will either drop these rows or replacing them with the mean values. We will start by evaluating the distinct values in categorical columns.The year, manufacturer,model and cylinders have Nan values. Without these the data will not be helpful.Dropping these rows with NAN values for those specific columns. Also, we will create a dictionary of model and other features to check for missing NAN values based on existing rows.

In [15]:
```
#drop rows and columns
data_csv.drop(index=data_csv[data_csv['year'].isna() & data_csv['model'].isna() & data_csv['manufacturer'].isna(
#drop all the nan values which are missing in the year column
data_csv.drop(index=data_csv[data_csv['year'].isna()].index,inplace=True)
data_csv.drop(['VIN'], axis=1, inplace=True )
data_csv.drop(['id'],axis = 1, inplace= True)
```

In [16]:

```
#apply lowercase first
data_csv['model'] = data_csv['model'].str.lower()
data_csv['type'] = data_csv['type'].str.lower()
data_csv.isna().sum()
```

Out[16]:

```
region               0
price                0
year                 0
manufacturer     13284
model             3984
condition       135927
cylinders       150065
fuel              2283
odometer          1947
title_status      6445
transmission      1669
drive           113207
size            264144
type             79366
paint_color     106612
state                0
dtype: int64
```

In [17]:

```
#drop rows that have 'model' and 'manufacturer' with nan values at the same time.
data_csv.drop(index=data_csv[data_csv['manufacturer'].isna() & data_csv['model'].isna()].index, inplace=True )
#fill the odometer nan values with mean
data_csv['odometer'] = data_csv['odometer'].fillna(data_csv.groupby('year')['odometer'].transform('mean'))
data_csv['title_status'].fillna( 'other', inplace=True )
data_csv['transmission'].fillna( 'other', inplace=True )
data_csv['paint_color'].fillna( 'custom', inplace=True )
data_csv['fuel'].fillna( 'other', inplace=True )

df_manufacturer = data_csv[data_csv['manufacturer'].notna()]
df_model_manufacturer = df_manufacturer[df_manufacturer['model'].notna()]
df1_cylinders = data_csv[data_csv['cylinders'].notna()]
df1_model_cylinders = df1_cylinders[df1_cylinders['model'].notna()]
df1_condition = data_csv[data_csv['condition'].notna()]
df1_model_condition = df1_condition[df1_condition['model'].notna()]
df1_drive = data_csv[data_csv['drive'].notna()]
df1_model_drive = df1_drive[df1_drive['model'].notna()]
df1_size = data_csv[data_csv['size'].notna()]
df1_model_size = df1_size[df1_size['model'].notna()]
df1_type = data_csv[data_csv['type'].notna()]
df1_model_type = df1_type[df1_type['model'].notna()]


condition_dict = {}
cylinders_dict = {}
manufacturer_dict = {}
drive_dict = {}
size_dict = {}
type_dict = {}


for model, manu in df_model_manufacturer[['model', 'manufacturer']].values:
    manufacturer_dict[model] = manu

for model, cylinder in df1_model_cylinders[['model', 'cylinders']].values:
    cylinders_dict[model] = cylinder

for model, condition in df1_model_condition[['model', 'condition']].values:
    condition_dict[model] = condition

for model, drive in df1_model_drive[['model', 'drive']].values:
    drive_dict[model] = drive

for model, size in df1_model_size[['model', 'size']].values:
    size_dict[model] = size
```

```
for model, types in df1_model_type[['model', 'type']].values:
    type_dict[model] = types



data_csv['manufacturer'] = data_csv.apply( lambda x: manufacturer_dict[x['model']] if x['model'] in manufacturer_
data_csv['cylinders'] = data_csv.apply( lambda x: x['cylinders'] if pd.isna( x['model'] ) or pd.notna( x['cylinde
data_csv['condition'] = data_csv.apply( lambda x: x['condition'] if pd.isna( x['model'] ) or pd.notna( x['conditi
data_csv['drive'] = data_csv.apply( lambda x: x['drive'] if pd.isna( x['model'] ) or pd.notna( x['drive'] ) else
data_csv['size'] = data_csv.apply( lambda x: x['size'] if pd.isna( x['model'] ) or pd.notna( x['size'] ) else si
data_csv['type'] = data_csv.apply( lambda x: x['type'] if pd.isna( x['model'] ) or pd.notna( x['type'] ) else ty
data_formatted = data_csv.copy()

print(data_formatted.shape)
data_formatted.dropna(inplace=True)
print(data_formatted.shape)
```

```
(369861, 16)
(281765, 16)
```

3.3 Rename Columns

In [18]:
```
data_formatted.rename( columns={"region": "city"}, inplace=True )
data_formatted.head()
```

Out[18]:

	city	price	year	manufacturer	model	condition	cylinders	fuel	odometer	title_status	transmission	drive	size	type	paint_color
28	auburn	22590	2010.0	chevrolet	silverado 1500	good	8 cylinders	gas	71229.0	clean	other	4wd	full-size	pickup	blue
30	auburn	30990	2017.0	toyota	tundra double cab sr	good	8 cylinders	gas	41124.0	clean	other	4wd	full-size	pickup	red
31	auburn	15000	2013.0	ford	f-150 xlt	excellent	6 cylinders	gas	128000.0	clean	automatic	rwd	full-size	truck	black
34	auburn	35000	2019.0	toyota	tacoma	excellent	6 cylinders	gas	43000.0	clean	automatic	4wd	compact	truck	grey
35	auburn	29990	2016.0	chevrolet	colorado extended cab	good	6 cylinders	gas	17302.0	clean	other	4wd	mid-size	pickup	red

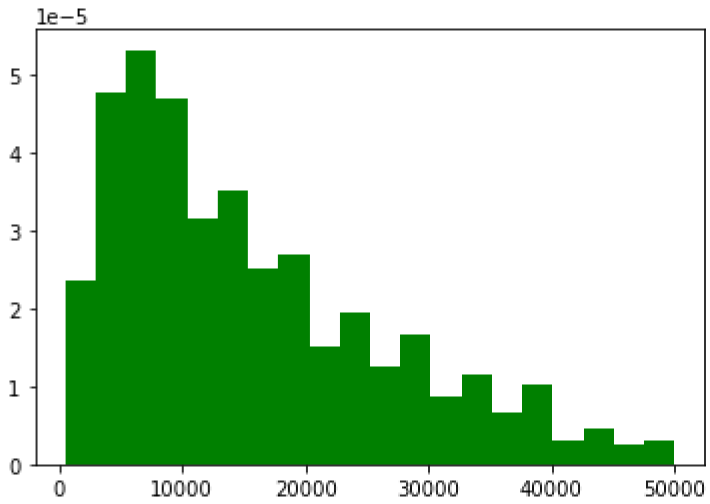
3.4 Data Analysis

3.4.1 Univariate Analysis (Descriptive Statistics)

3.4.1.1 Response Variable

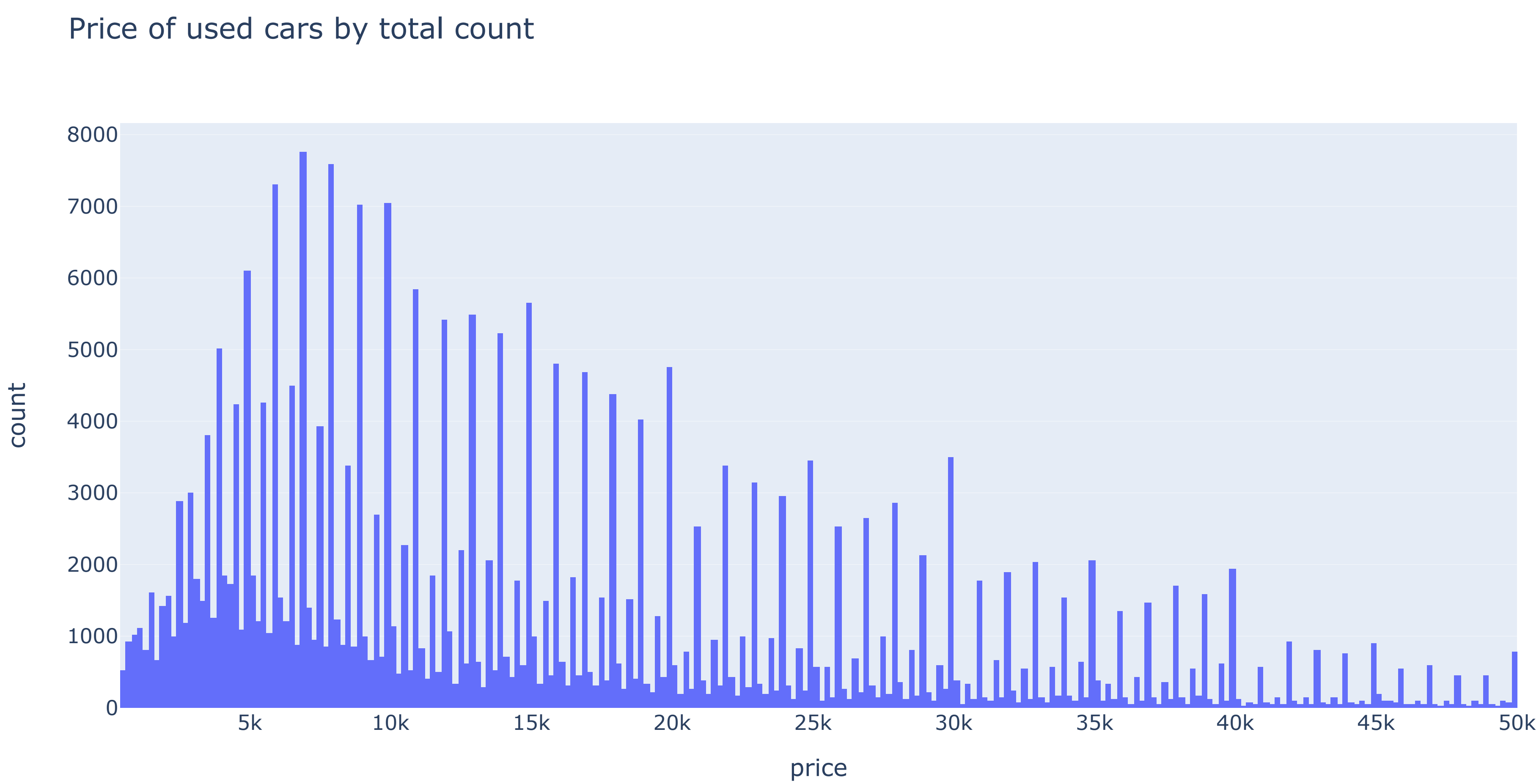
In [19]:
```python
data_formatted['year'] = data_formatted['year'].astype( 'int64' )
#sns.distplot( data_formatted['price'] )
n_bins = 20
colors = ['green']

plt.hist(data_formatted['price'], n_bins, density = True,
         histtype ='bar',
         color = colors,
         label = colors)
```

Out[19]:
```
(array([2.35709917e-05, 4.75678873e-05, 5.31189799e-05, 4.69928452e-05,
        3.16144115e-05, 3.49958314e-05, 2.51082614e-05, 2.68950507e-05,
        1.51690667e-05, 1.93219895e-05, 1.25505457e-05, 1.65199138e-05,
        8.73605166e-06, 1.15911861e-05, 6.57928513e-06, 1.03278142e-05,
        3.13190033e-06, 4.61754536e-06, 2.56259427e-06, 3.08457766e-06]),
 array([  501. ,  2975.9,  5450.8,  7925.7, 10400.6, 12875.5, 15350.4,
        17825.3, 20300.2, 22775.1, 25250. , 27724.9, 30199.8, 32674.7,
        35149.6, 37624.5, 40099.4, 42574.3, 45049.2, 47524.1, 49999. ]),
 <BarContainer object of 20 artists>)
```

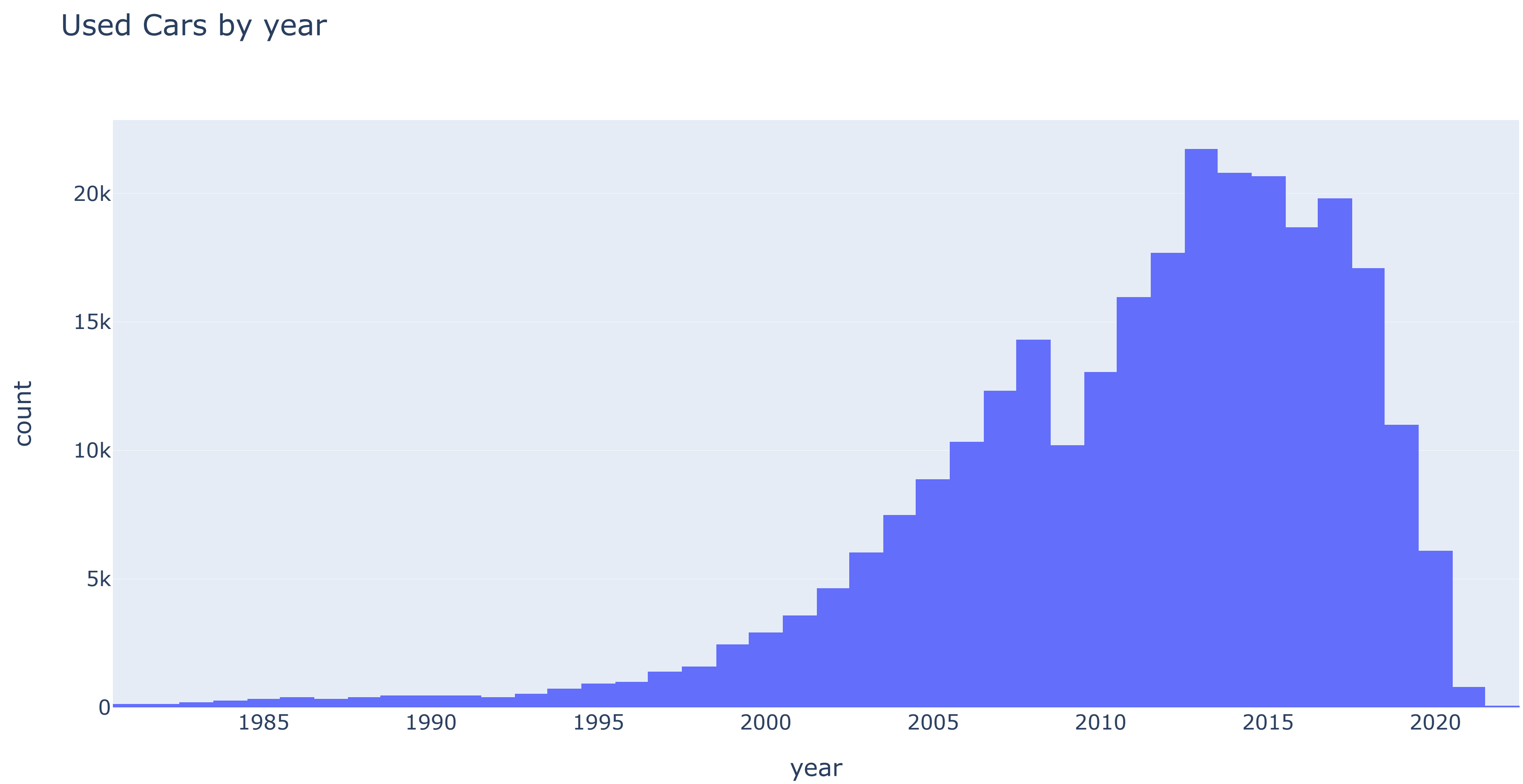


3.4.1.2 Numerical Variables

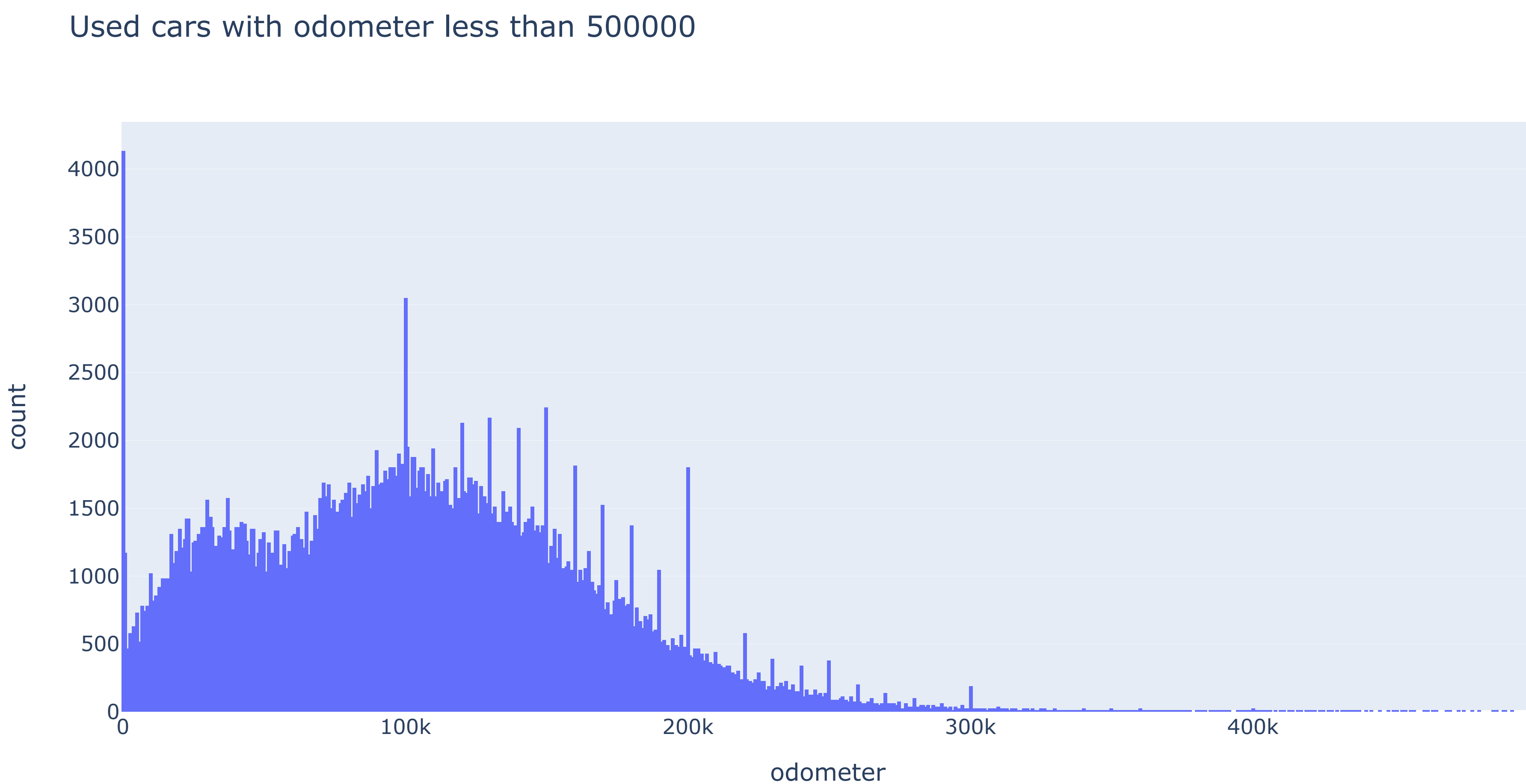
In [20]:
```
px.histogram(data_formatted,x="price", title='Price of used cars by total count',)
```



In [21]:
```
px.histogram(data_formatted[data_formatted['year'] > 1980],x="year",title="Used Cars by year")
```

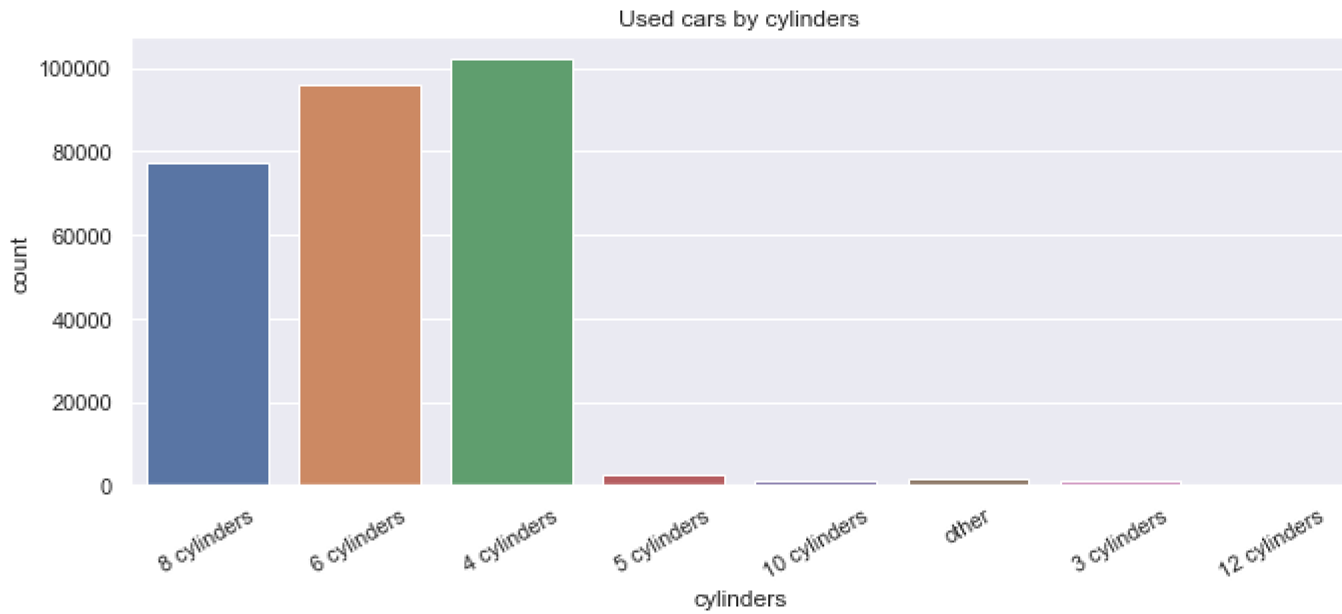


In [22]:
```
px.histogram(data_formatted[data_formatted['odometer'] < 500000],x="odometer", title='Used cars with odometer les
```



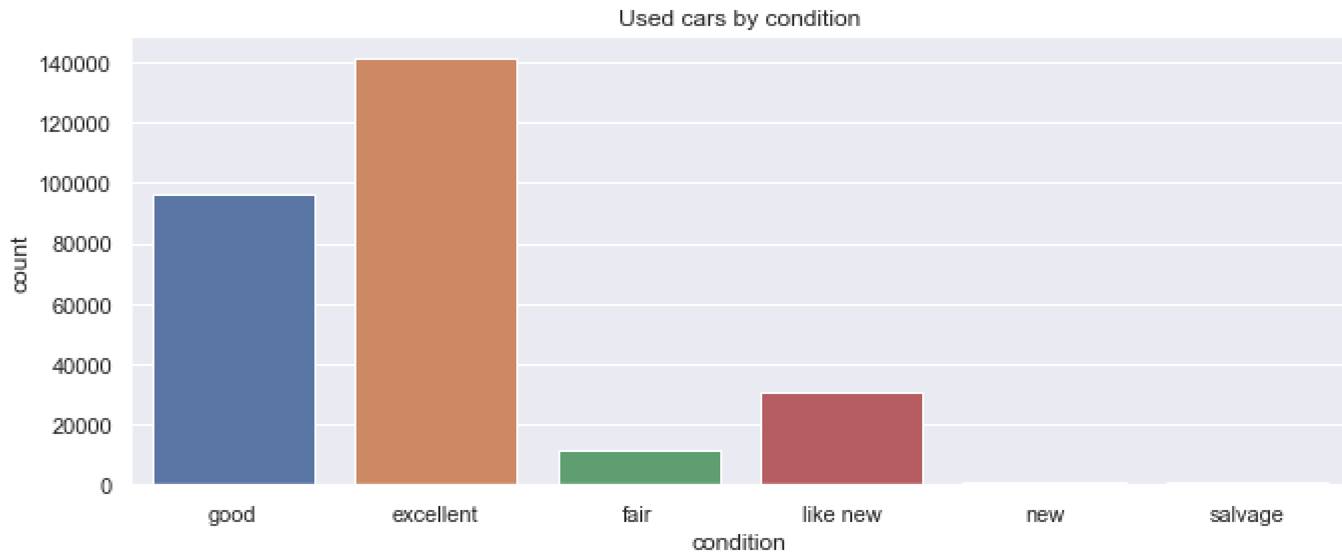
3.4.1.3 Categorical Variables

In [23]:
```
sns.set(rc={'figure.figsize':(24,14)})
plt.subplot( 3, 2, 5)
fig1 = sns.countplot( data_formatted['cylinders'] )
fig1.set_xticklabels(fig1.get_xticklabels(), rotation=30)
fig1.title.set_text('Used cars by cylinders')
plt.show()
```



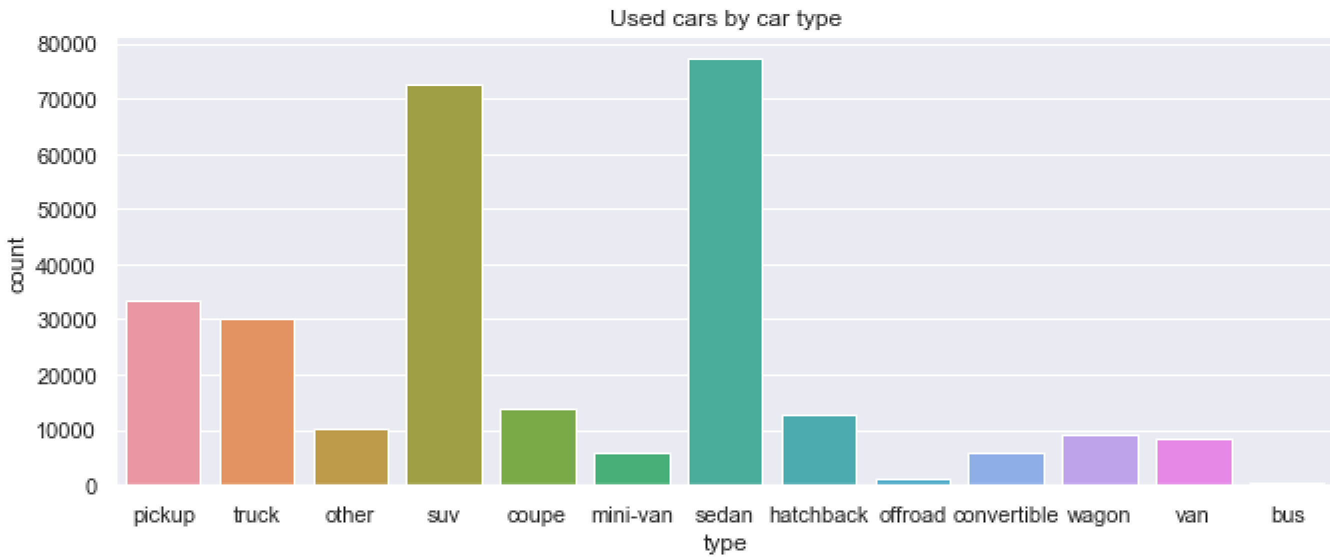
Cylinders do not have a direct. Correlation in determining the price.

In [24]:
```
sns.set(rc={'figure.figsize':(24,14)})
plt.subplot( 3, 2, 5)
fig2 = sns.countplot( data_formatted['condition'] )
fig2.title.set_text('Used cars by condition')
plt.show()
```



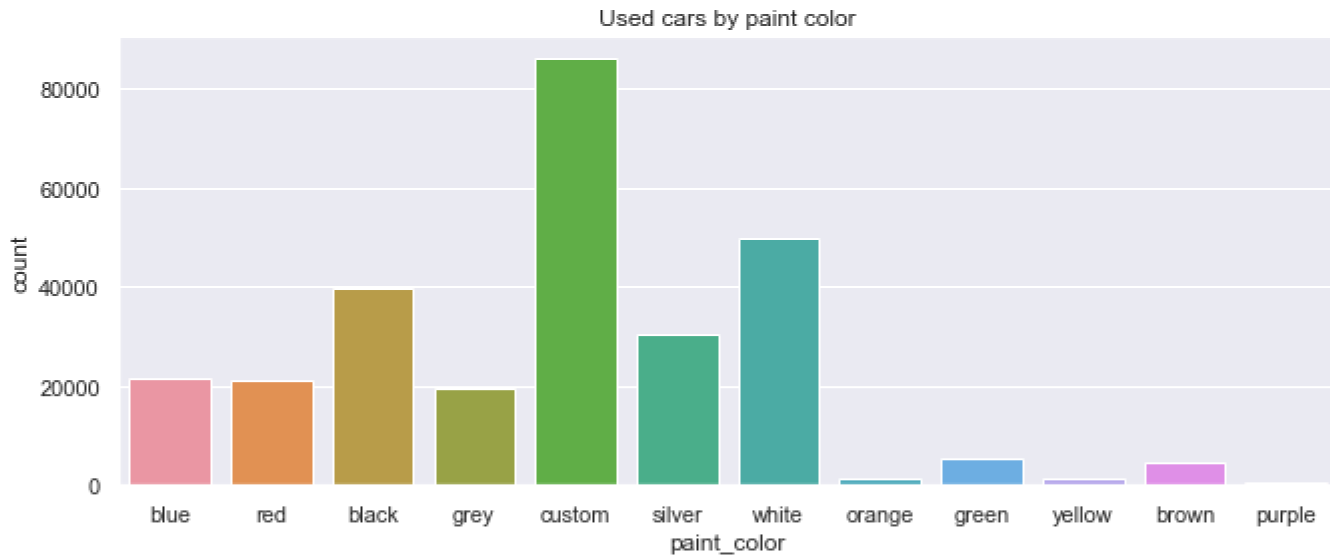
Cars with excellent condition cost more compared to other cars.

In [25]:
```
sns.set(rc={'figure.figsize':(24,14)})
plt.subplot( 3, 2, 3)
fig3 = sns.countplot( data_formatted['type'] )
fig3.title.set_text('Used cars by car type')
plt.show()
```



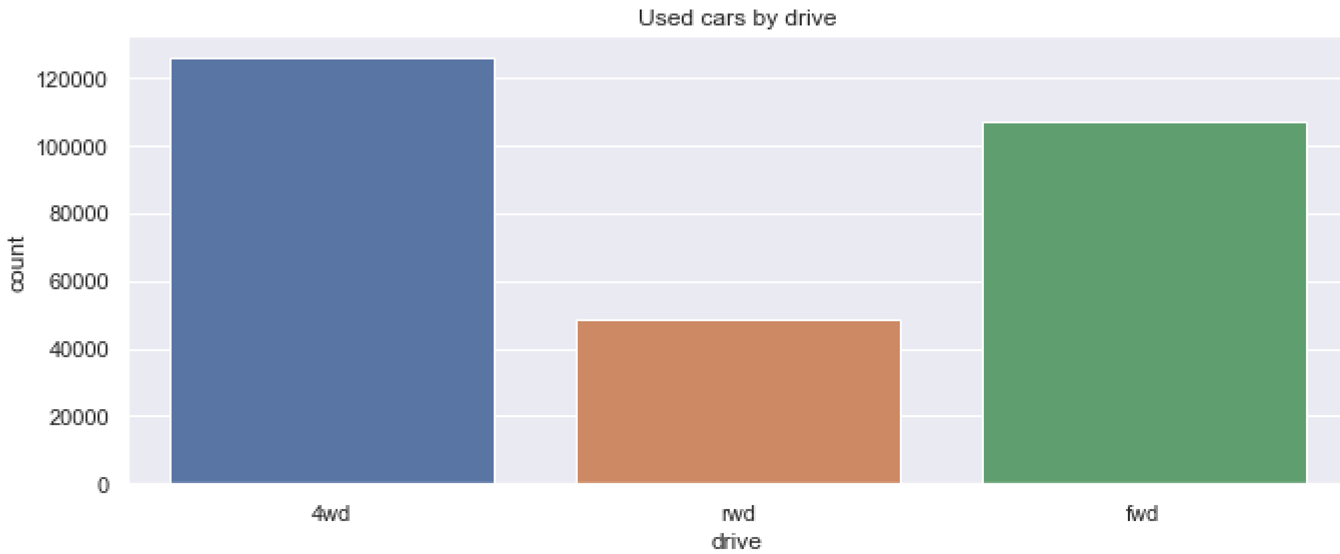
Cars of type SUV and sedan are sold more compared to other cars. The prices of these cars are driven by supply and demand.

In [26]:
```
sns.set(rc={'figure.figsize':(24,14)})
plt.subplot( 3, 2, 5)
fig4 = sns.countplot( data_formatted['paint_color'] )
fig4.title.set_text('Used cars by paint color')
plt.show()
```



Cars with white, black and silver are sold more compared to the other car types. The custom column in the above may involve Nan values too.

In [27]:
```
plt.subplot( 3, 2, 1)
fig5 = sns.countplot( data_formatted['drive'] )
fig5.title.set_text('Used cars by drive')
plt.show()
```



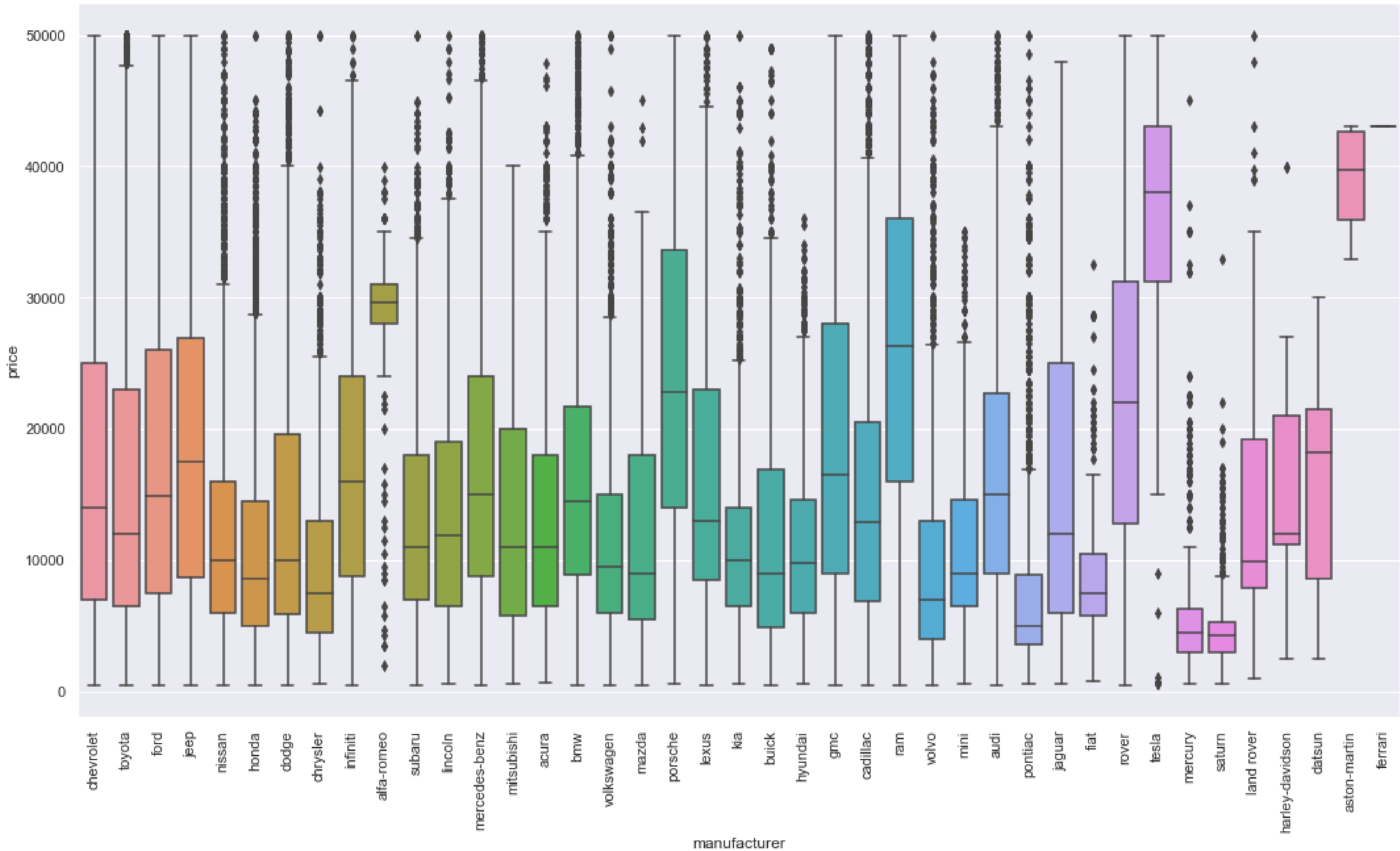
Cars with 4wd are priced higher compared to the other cars.

In [28]:
```
plt.subplot( 3, 2, 4)
fig6 = sns.boxplot(data=data_formatted, x='fuel', y='price')
fig6.title.set_text("Price by fuel")
plt.show()
```

Cars with fuel type diesel are priced higher compared to the other car types.

Cars with fuel type diesel are priced higher compared to the other car types.

In [29]:
```
# Visualization price vs manufacturer cars
fig = plt.figure(figsize=(18, 10))
sns.boxplot(data=data_formatted, x='manufacturer', y='price')
plt.xticks(rotation=90)
plt.show()
```



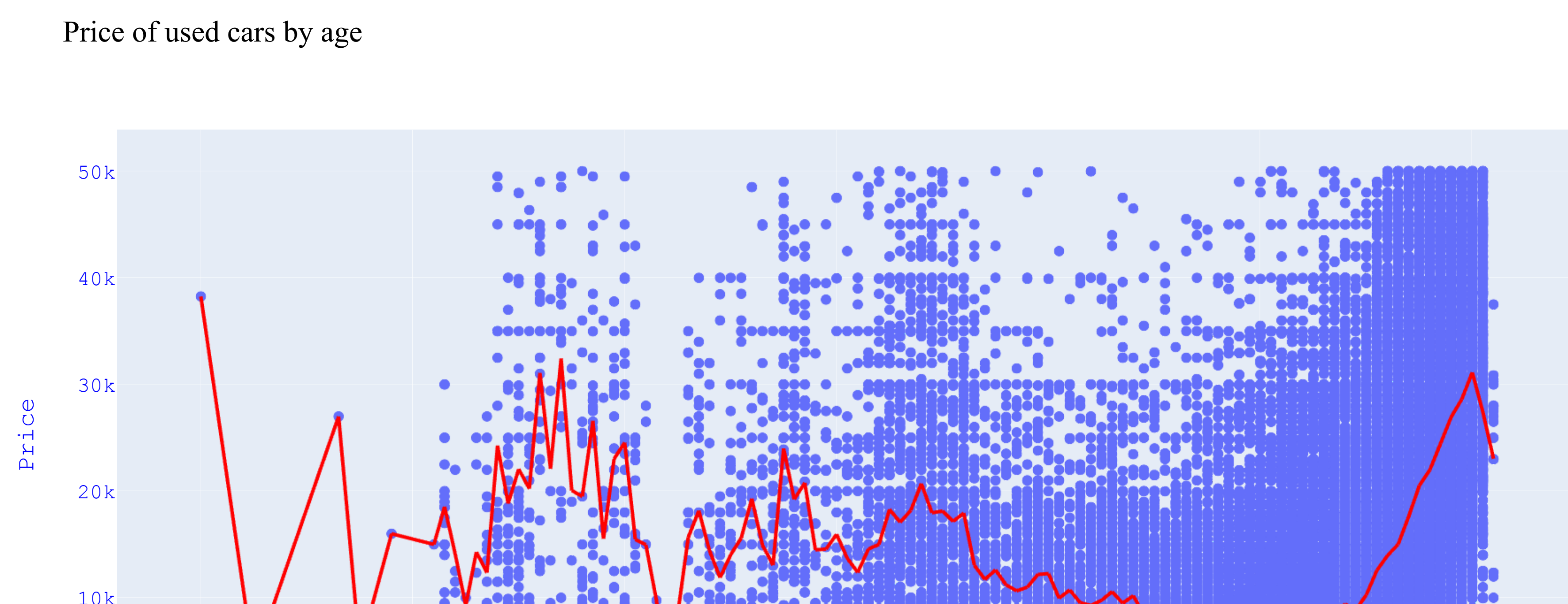
Tesla, Ferrari, Porsche and RAM are priced higher compared to other manufacturers.

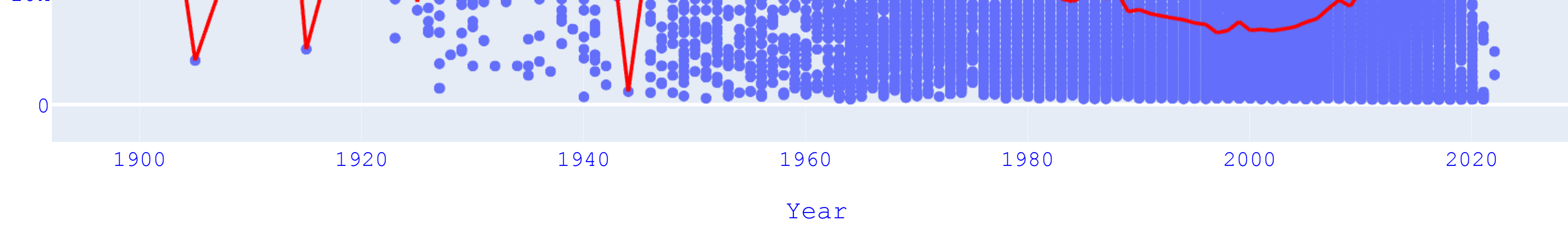
3.4.2 Bivariate analysis (inferential statistics)

We will be answering some of the questions from the business understanding steps.

1) Do used cars with less age should cost more/less?

In [30]:
```python
data_formatted['mean_price'] = data_formatted.groupby('year')['price'].transform('mean')
data_formatted = data_formatted.sort_values(by=['year'])

fig1 = px.scatter(data_formatted, x="year", y="price")

fig2 = px.line(data_formatted, x="year", y="mean_price", title='Mean Price')

fig3 = go.Figure(data=fig1.data + fig2.data, layout=go.Layout(
        title=go.layout.Title(text="Price of used cars by age")
    ))
fig3.update_traces(line=dict(color = 'red'))
fig3.update_layout(
    showlegend=True,
    font_family="Courier New",
    font_color="blue",
    title_font_family="Times New Roman",
    title_font_color="black",
    legend_title_font_color="green"
)
fig3.update_xaxes(title_text = 'Year')
fig3.update_yaxes(title_text = 'Price')
fig3.show()
```

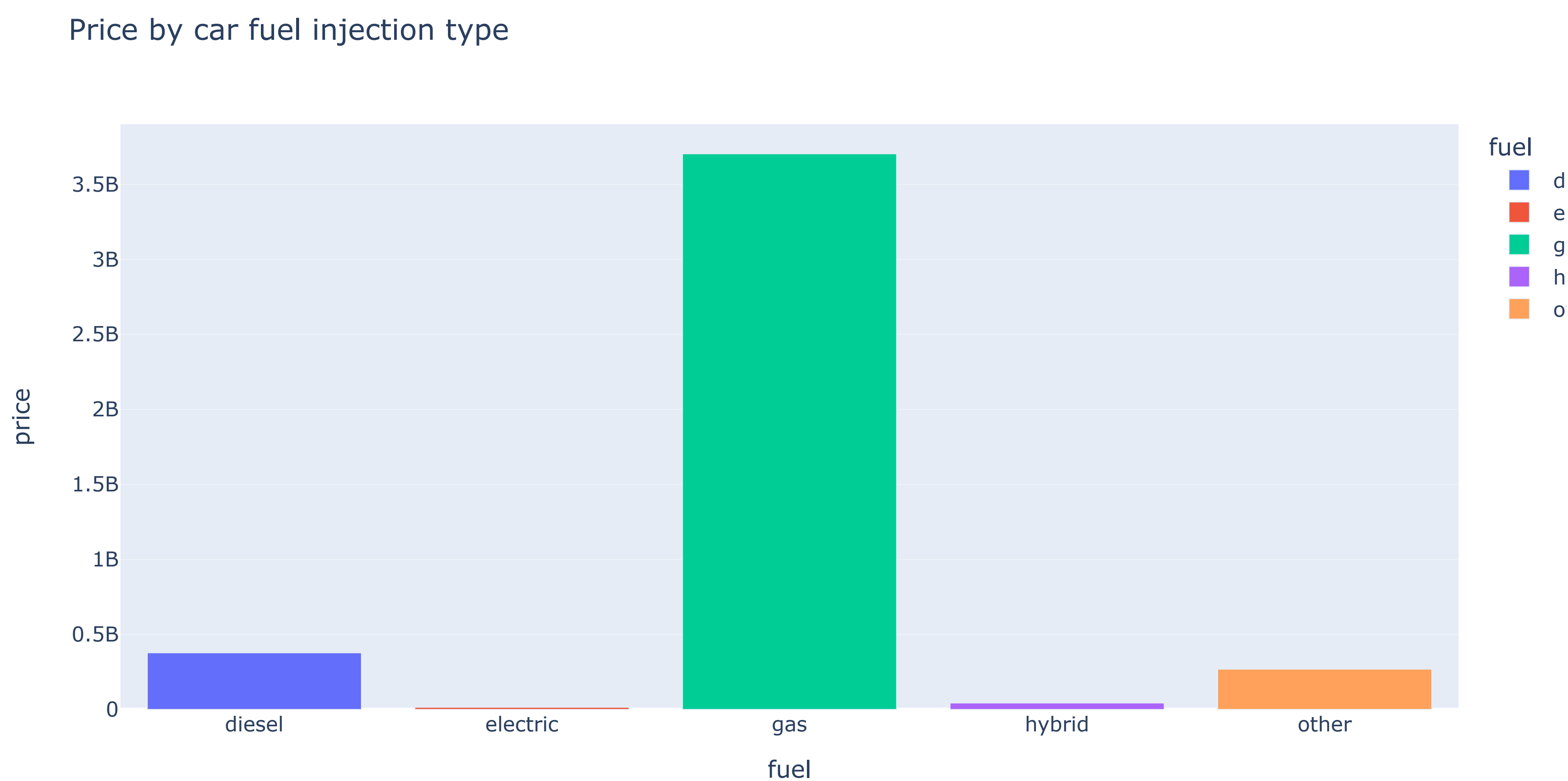




From the year 2000, we clearly see that the price of cars increases as age of the used car decreases.

2) Do used cars with electric fuel cost more?

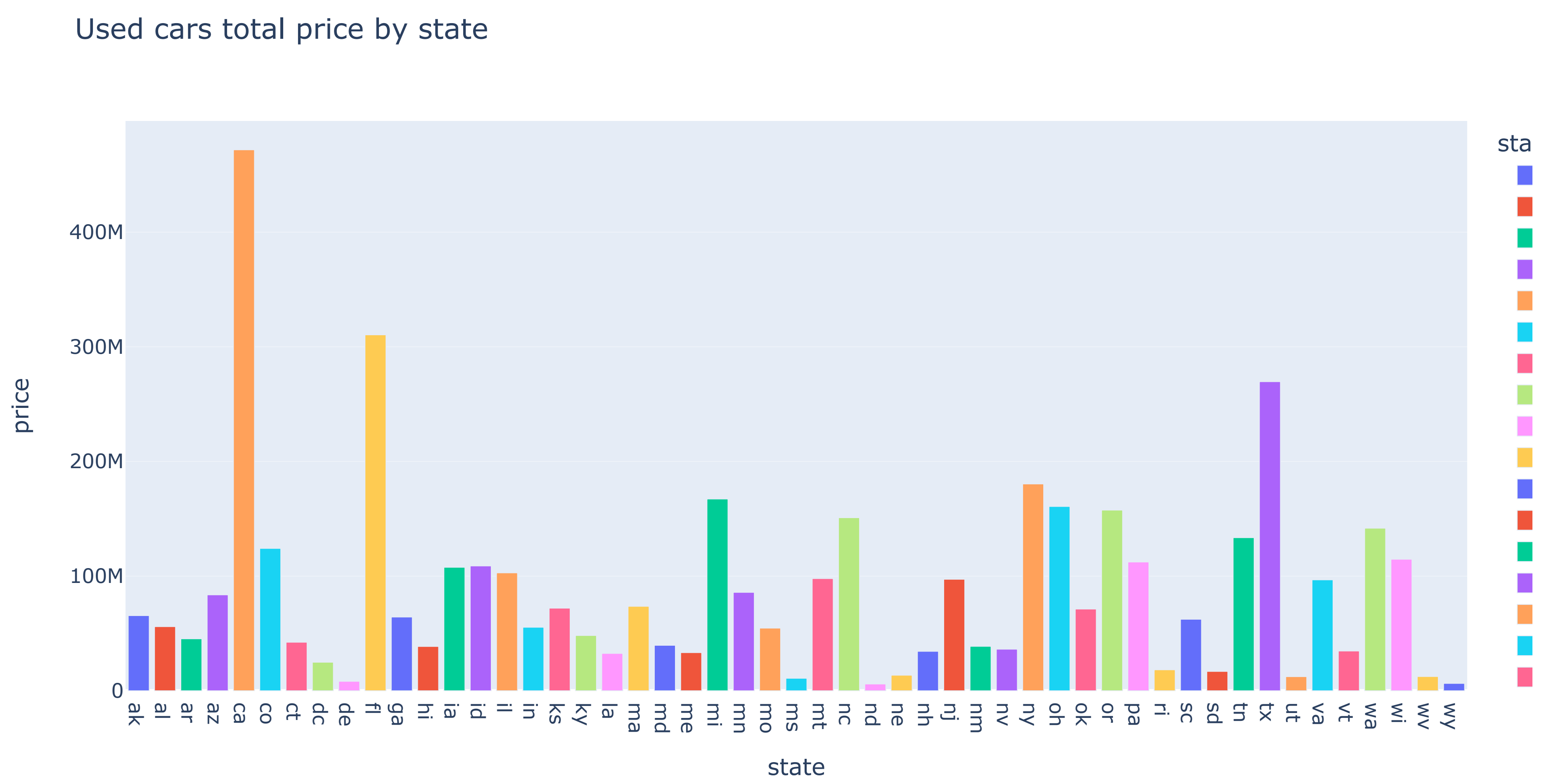
```
In [31]: df_electric = data_formatted[['fuel','price']].groupby('fuel').sum().reset_index()
         px.bar(df_electric,x='fuel', y='price',color="fuel", title="Price by car fuel injection type")
```



Based on the above plot, it is clear that used cars with fuel type gas are more sold compared to the other car types.

3) List top 5 states with highest used car sales.

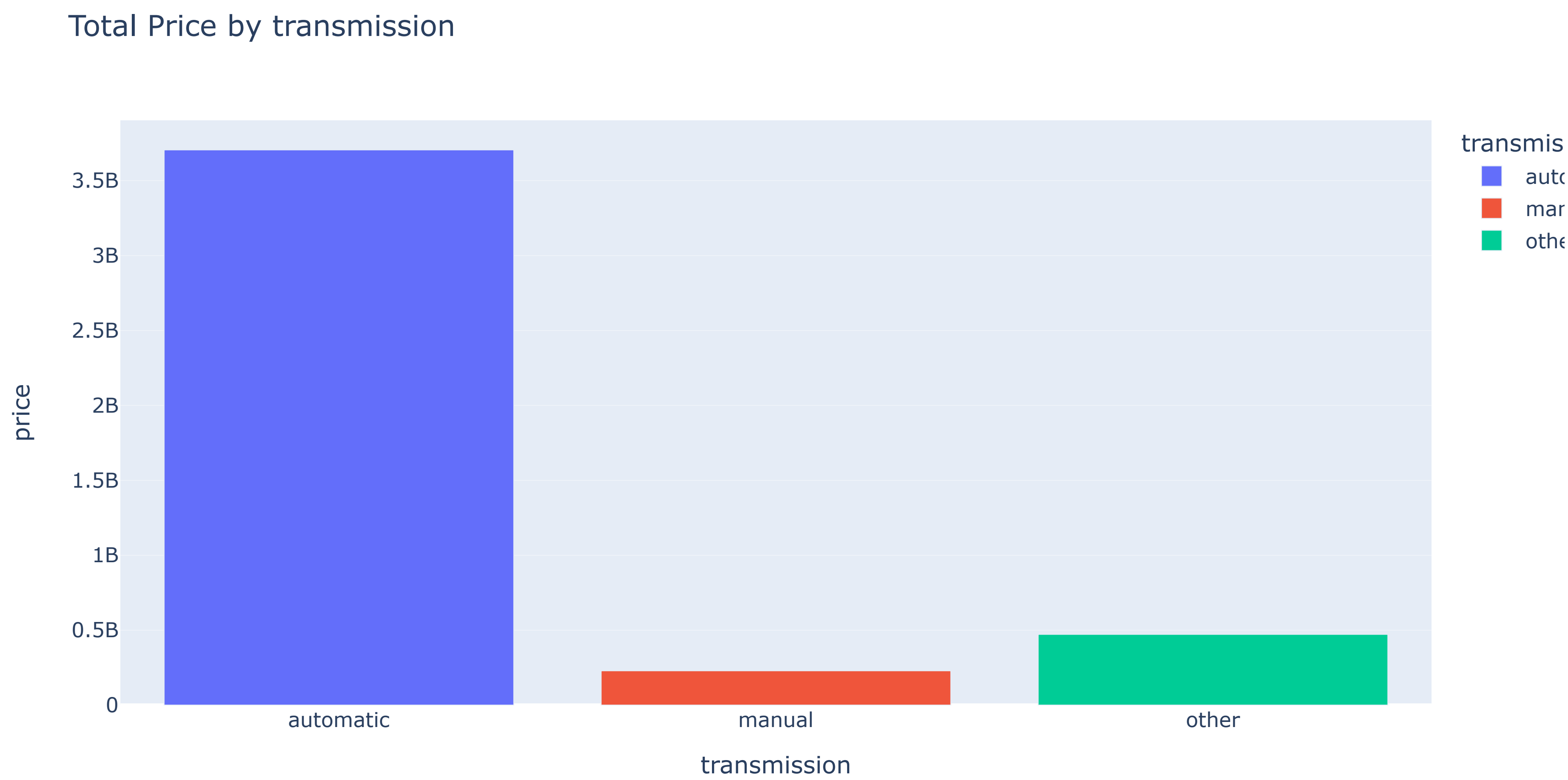
In [32]:
```
aux1 = data_formatted[['state', 'price']].groupby( 'state' ).sum().reset_index()
px.bar(aux1,x='state', y='price',color="state", title="Used cars total price by state")
```



Based on the above plot, we can clearly see th top 5 states by total price. CA, FL, TX, NY and MI

4) Do used cars with automatic transmission cost more?

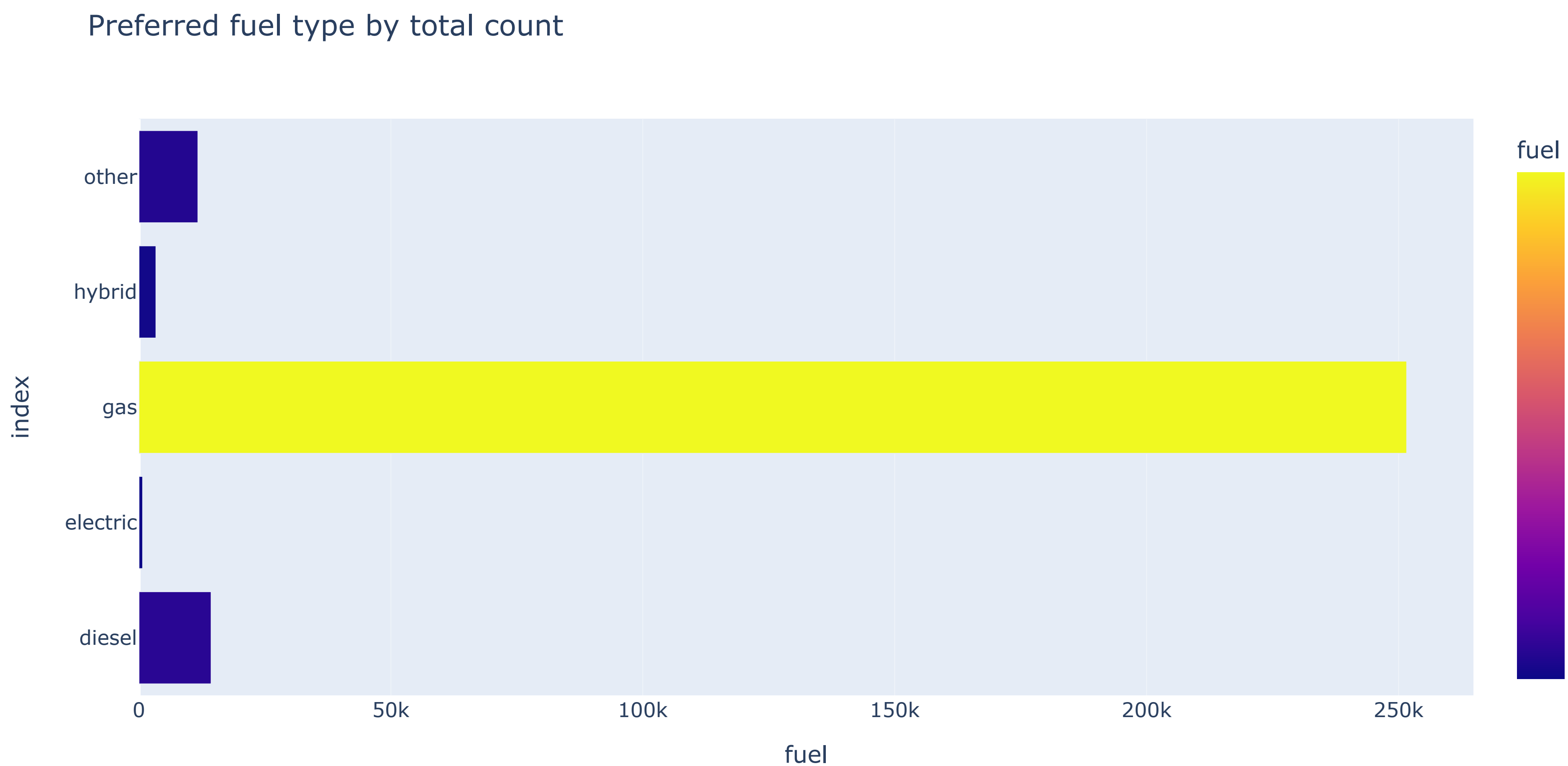
In [33]:
```
aux1 = data_formatted[['transmission', 'price']].groupby( 'transmission' ).sum().reset_index()
fig = px.bar(aux1,x='transmission', y='price',color="transmission", title="Total Price by transmission");
fig.show()
```



Based on the above plot, it clearly shows us that automatic transmission costs much more than manual and other transmissions.

5) Which fuel type is preferred more compared to the other?

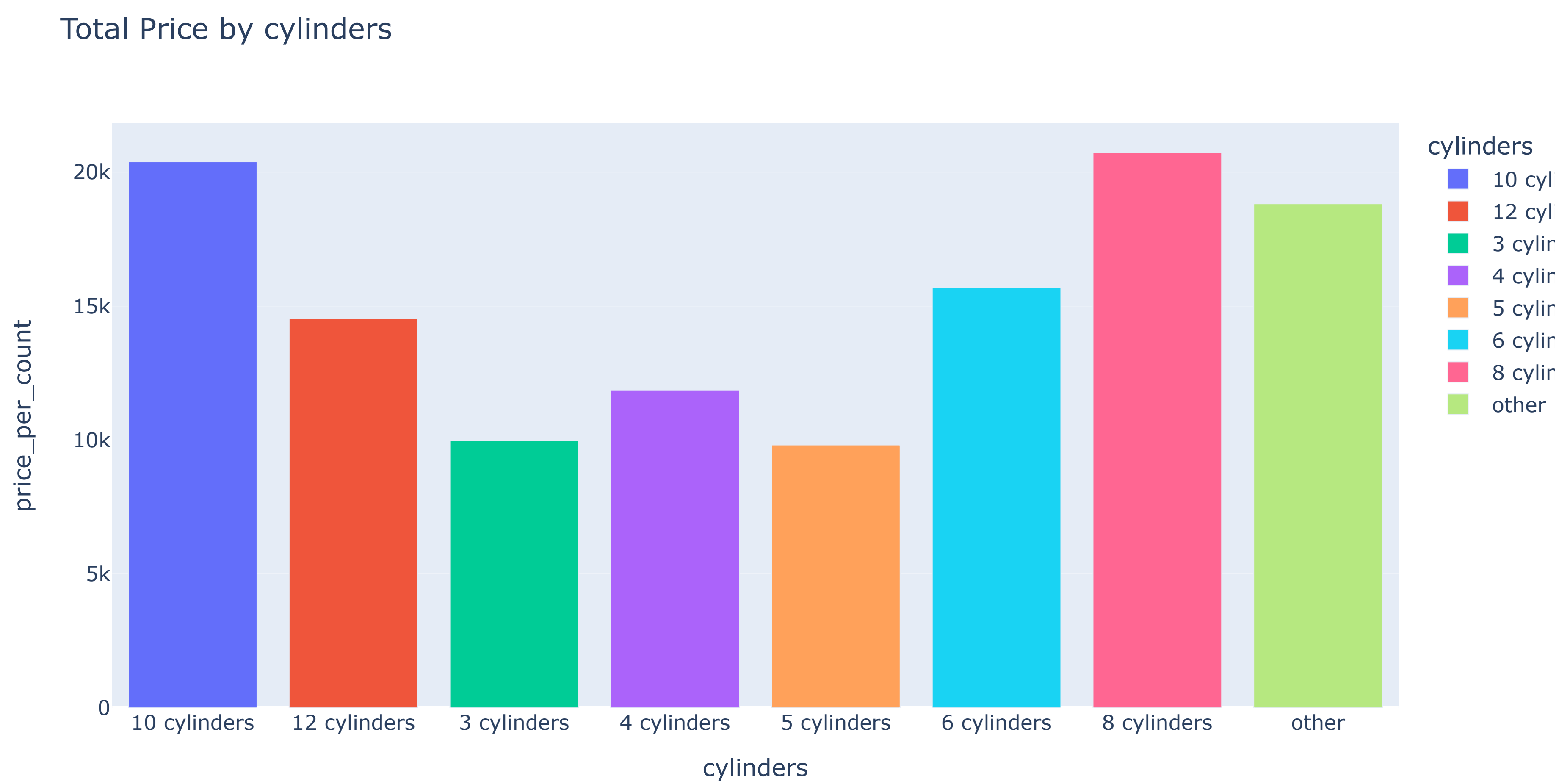
In [34]:
```
aux1 = data_formatted.groupby('fuel')['fuel'].count()
fig = px.bar(aux1, x='fuel', color="fuel", title="Preferred fuel type by total count")
fig.show()
```



From the above plot, we can see that the gas is most preferred fuel type compared to other fuel types.

6) Do used cars with more cylinders cost more?

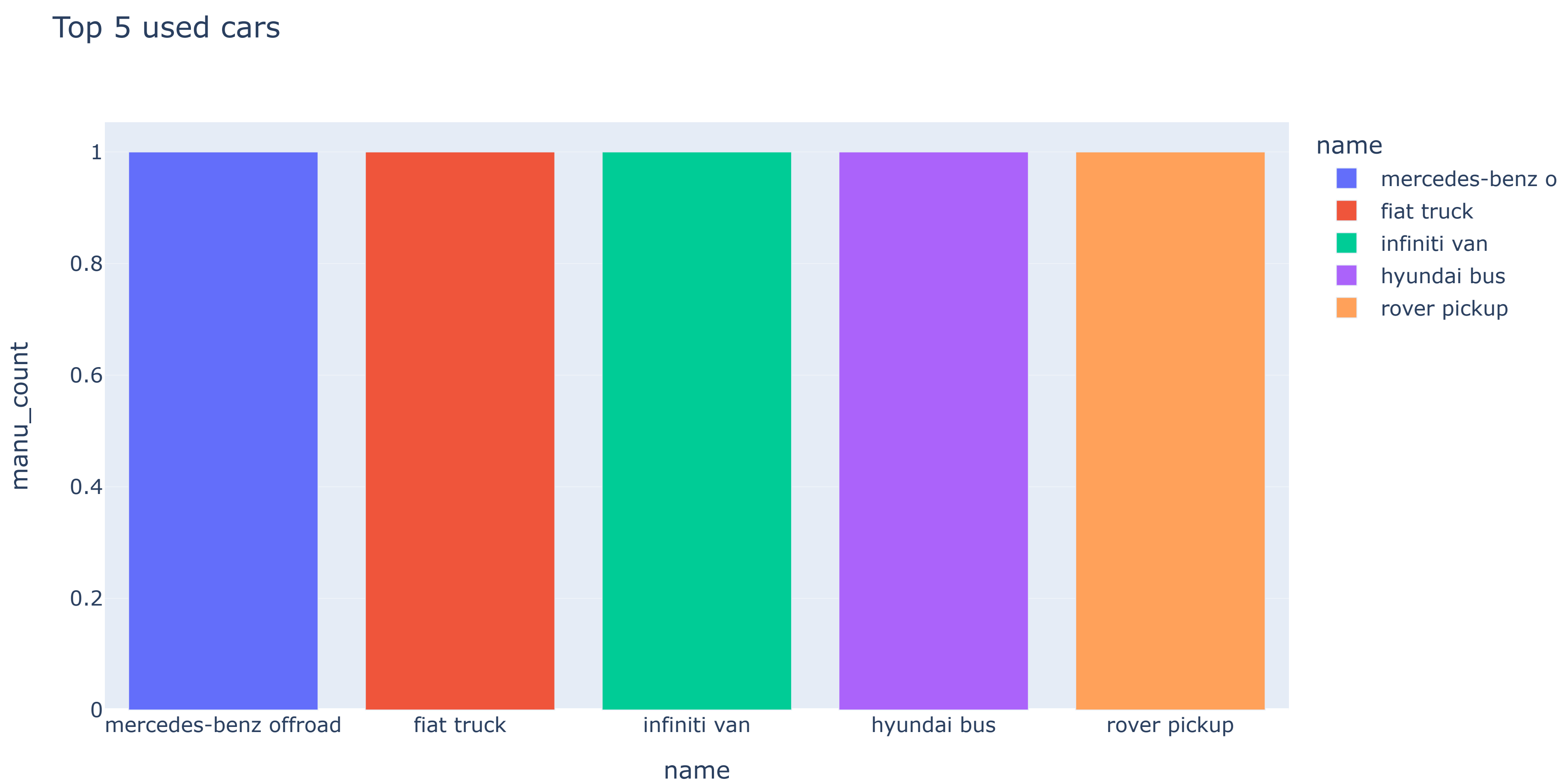
In [35]:
```
#In order to find out the true cost of cylinder, lets divide the total price of used car by cylinder and divide 
# number of rows by cylinder
aux1 = data_formatted[['cylinders', 'price']].groupby( 'cylinders' ).sum().reset_index()
cylinder_count = data_formatted.groupby('cylinders')['cylinders'].count()
cyl_count_frame = pd.Series.to_frame(cylinder_count)
cyl_count_frame.columns = ['count']
aux1 = aux1.set_index('cylinders')
aux1['count'] = cyl_count_frame['count']
aux1['price_per_count'] = aux1['price']/aux1['count']
aux1 = aux1.reset_index()
fig = px.bar(aux1,x='cylinders', y='price_per_count',color="cylinders", title="Total Price by cylinders");
fig.show()
```



Based on the information provided and from the above plot, it is pretty clear that the cost of cylinder does not increase with increase in number of cylinders.

7) Which make and type are most predominant in the used car market?

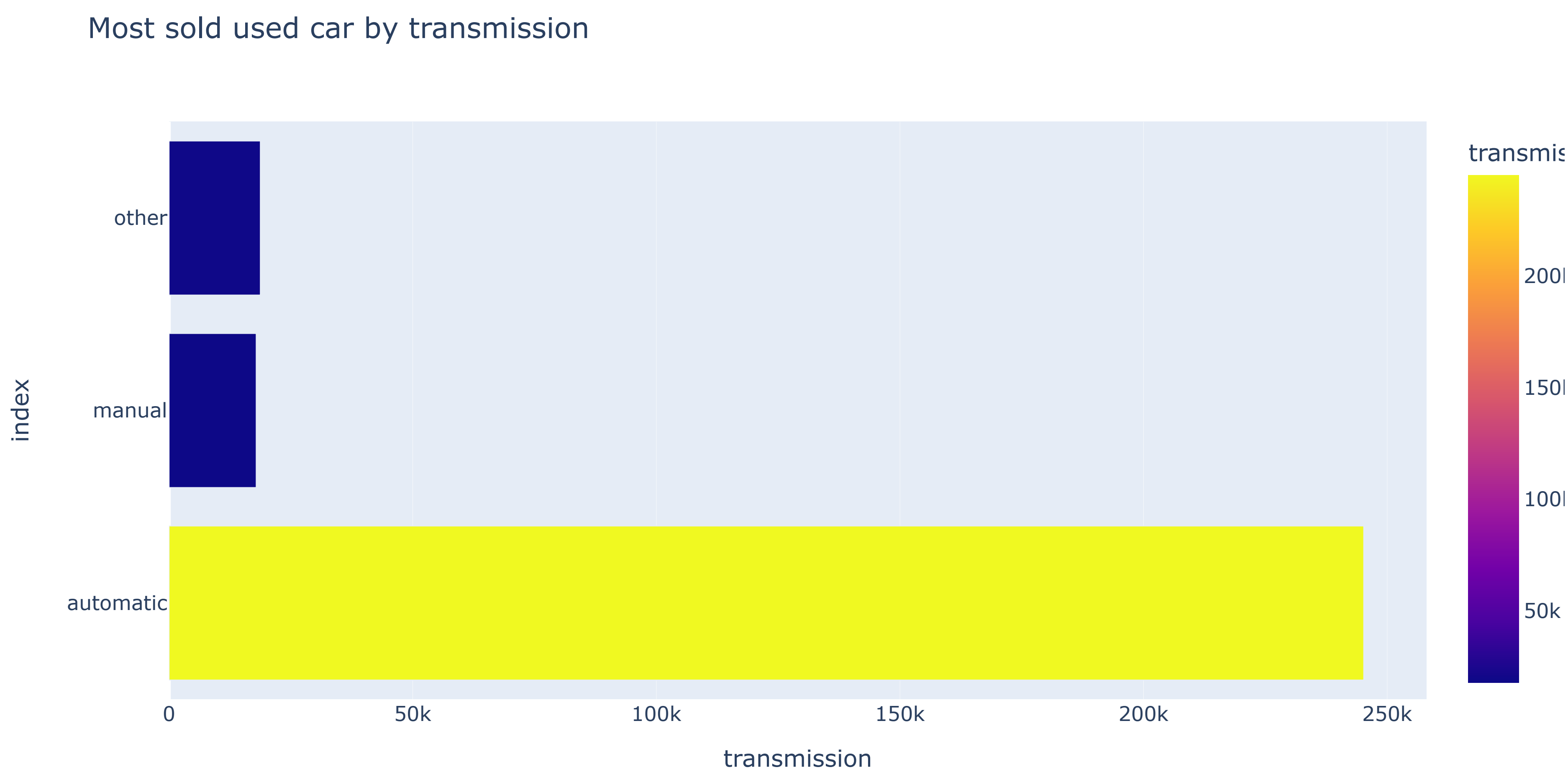
In [36]:
```
aux1 = data_formatted.groupby(['manufacturer', 'type'])['manufacturer', 'type'].count()
aux1.columns = ['manu_count','type_count']
aux1 = aux1.sort_values(by=['type_count']).head(5)
aux1 = aux1.reset_index()
aux1['name'] = aux1['manufacturer'] + " " + aux1['type']
fig = px.bar(aux1,x='name', y='manu_count',color='name', title="Top 5 used cars");
fig.show()
```



From the above, it is pretty clear that ford pick up is the most sold car.

9) Which transmission type is the most sold?

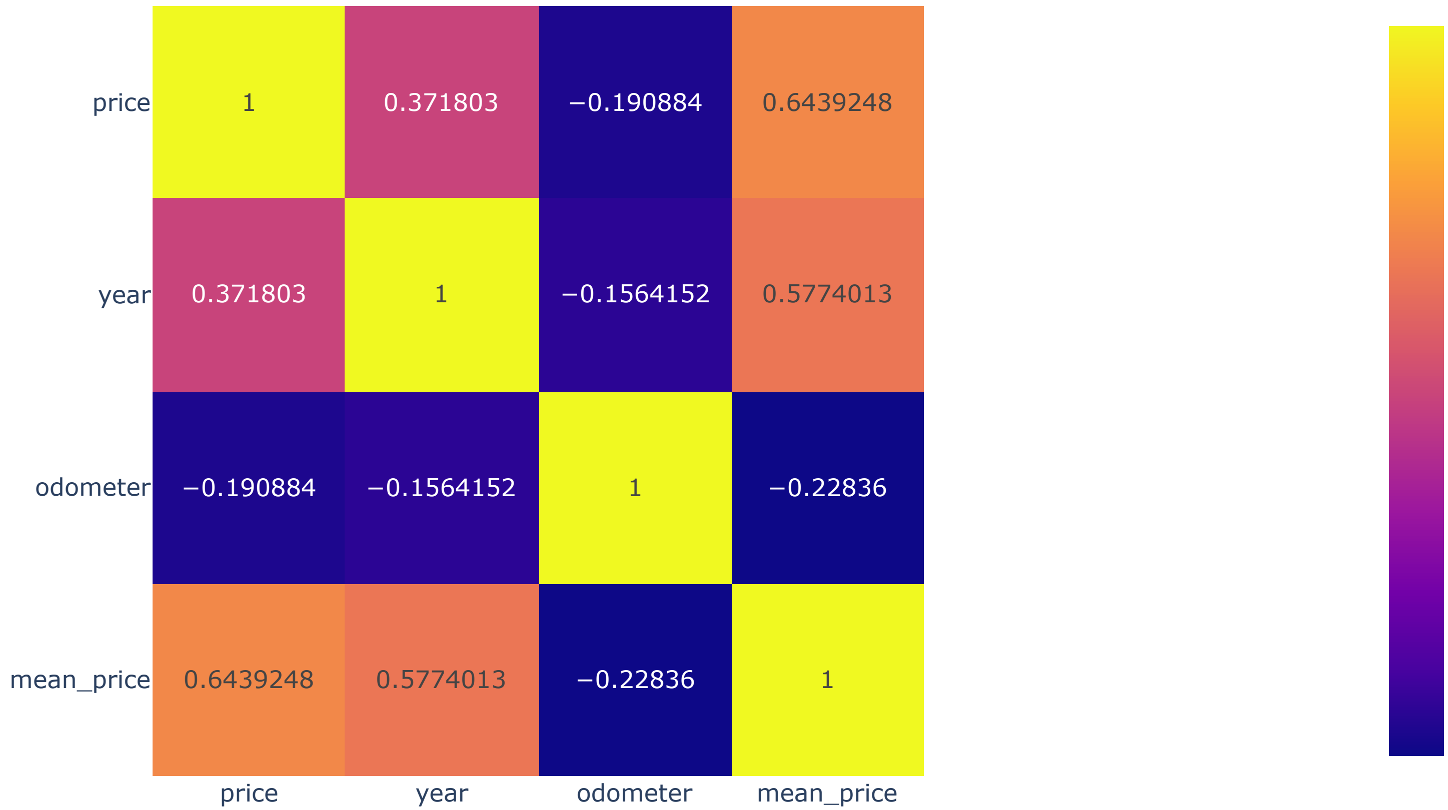
In [37]:
```
aux1 = data_formatted.groupby('transmission')['transmission'].count()
fig = px.bar(aux1, x='transmission', color="transmission", title="Most sold used car by transmission")
fig.show()
```



From the above plot, it is pretty clear that automatic transmission is the most sold among used cars.

3.4.2 Multivariate analysis

In [38]:
```python
#Lets take a look at the correlation matrix for numerical attributes
#data.corr()
fig = px.imshow(data_formatted.corr(),text_auto=True)
fig.show()
```



In [39]:
```python
#type(cat_attributes)
#d_cat_attributes = df_encoded.drop(['city', 'manufacturer','price', 'year','odometer','size', 'paint_color', 'd
#fig = px.imshow(d_cat_attributes.corr(),text_auto=True)
#fig.show()
```

In [ ]:

```

```

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.

In [ ]:

```

```

4.1 Rescaling

In [40]:

```
#Using Standard Scaler here to scale the numerical values.
ss = StandardScaler()
data_formatted['year'] = ss.fit_transform(data_formatted[['year']].values)
data_formatted['odometer'] = ss.fit_transform(data_formatted[['odometer']].values)
```

In [41]:

```
data_formatted.drop('mean_price', axis=1,inplace=True)
```

4.2 Transformation

Transformation of categorical values using One Hot Encoder, Ordinal Encoder. pd.get_dummies is not being used here as it is not a simple analyzation

4.2.1 Encoding

In [42]:

```
df_scaled = data_formatted.copy()
```

In [43]:

```
ohe = OneHotEncoder(sparse=False)
```

In [44]:

```
#Transforming the fuel column with One-Hot Encoder
ohe.fit(df_scaled[['fuel']])
temp_df = pd.DataFrame(data=ohe.transform(df_scaled[['fuel']]), columns=ohe.get_feature_names_out())
df_scaled.drop(columns=['fuel'], axis=1, inplace=True)
df_scaled = pd.concat([df_scaled.reset_index(drop=True), temp_df], axis=1)

#Transforming the transmission column
ohe.fit(df_scaled[['transmission']])
temp_df2 = pd.DataFrame(data=ohe.transform(df_scaled[['transmission']]), columns=ohe.get_feature_names_out())
df_scaled.drop(columns=['transmission'], axis=1, inplace=True)
df_scaled = pd.concat([df_scaled.reset_index(drop=True), temp_df2], axis=1)


#Transforming the drive column
ohe.fit(df_scaled[['drive']])
temp_df3 = pd.DataFrame(data=ohe.transform(df_scaled[['drive']]), columns=ohe.get_feature_names_out())
df_scaled.drop(columns=['drive'], axis=1, inplace=True)
df_scaled = pd.concat([df_scaled.reset_index(drop=True), temp_df3], axis=1)
```

In [45]:

```
#OrdinalEncoding the condition column
condition_dict = {'col':'condition','mapping':{'other': 0, 'new': 6, 'like new': 5, 'excellent': 4, 'good': 3, ':
ordinal_encoder1 = ce.OrdinalEncoder(cols=['condition'], return_df=True, mapping=[condition_dict])
df_scaled['condition'] = ordinal_encoder1.fit_transform(df_scaled[['condition']])

cylinders_dict = {'col':'cylinders','mapping':{'other': 0, '12 cylinders': 12, '10 cylinders': 10, '8 cylinders'
ordinal_encoder2 = ce.OrdinalEncoder(cols=['cylinders'], return_df=True, mapping=[cylinders_dict])
df_scaled['cylinders'] = ordinal_encoder2.fit_transform(df_scaled[['cylinders']])

title_status_dict = {'col':'title_status','mapping':{'other': 0, 'clean': 6, 'lien': 5, 'salvage': 4, 'rebuilt':
ordinal_encoder3 = ce.OrdinalEncoder(cols=['title_status'], return_df=True, mapping=[title_status_dict])
df_scaled['title_status'] = ordinal_encoder3.fit_transform(df_scaled[['title_status']])
```

In [46]:

```
#label encoding the remaining columns.
label_encoder = LabelEncoder()
df_scaled['size']= label_encoder.fit_transform(df_scaled['size'])
df_scaled['manufacturer'] = label_encoder.fit_transform( df_scaled['manufacturer'] )
df_scaled['type'] = label_encoder.fit_transform( df_scaled['type'] )
df_scaled['paint_color'] = label_encoder.fit_transform( df_scaled['paint_color'] )
df_scaled['state'] = label_encoder.fit_transform(df_scaled['state'] )
df_scaled['city'] = label_encoder.fit_transform(df_scaled['city'] )
#Lets take a look at the info to see what columns remain
```

In [47]:

```
#one last thing to do is to drop the model column as it is no more a determinant at this point.
df_encoded = df_scaled.drop('model', axis = 1)
```

4.2.2 Response Variable Transformation

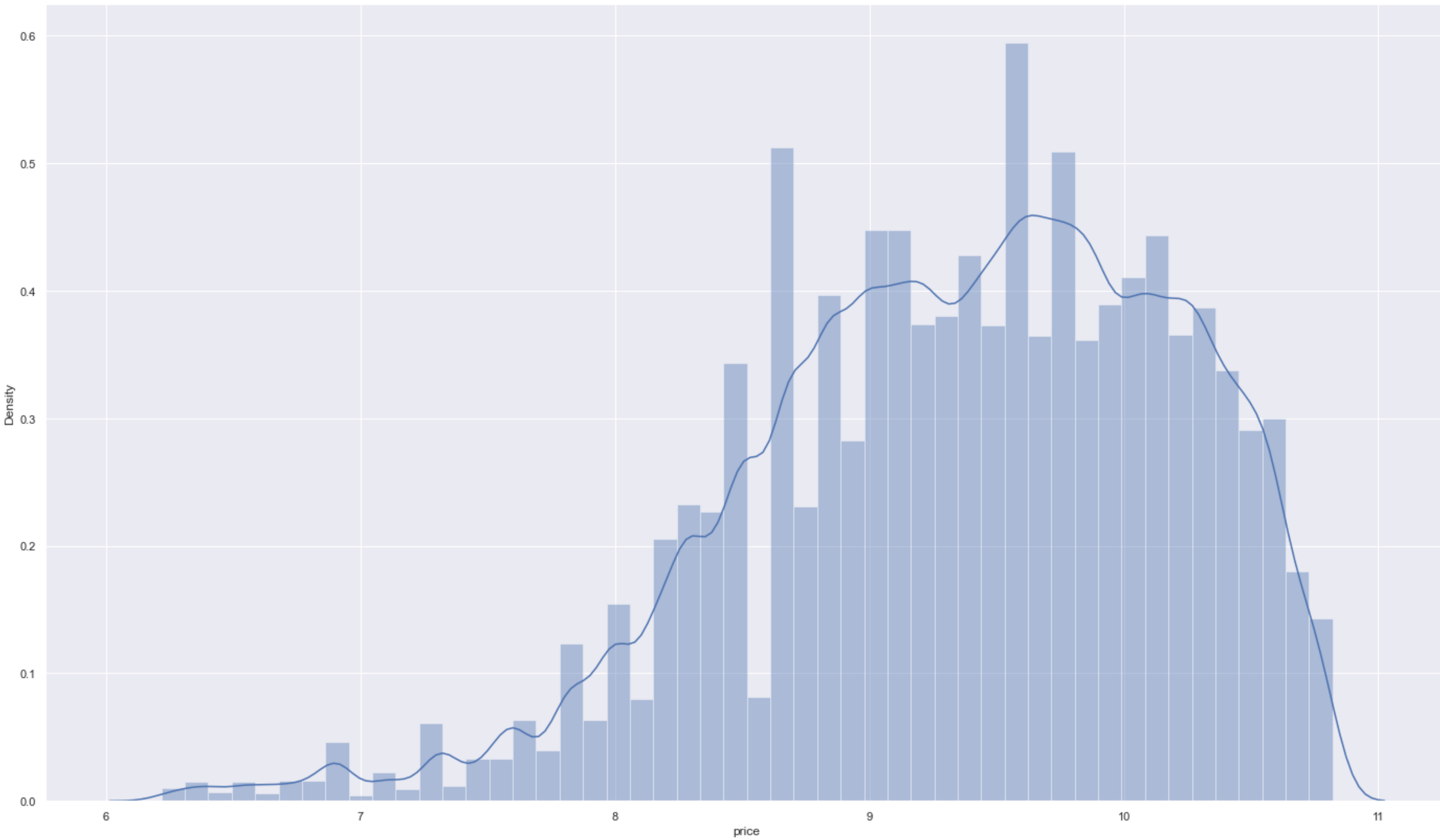
In [48]:

```
df_encoded['price'] = np.log1p( df_encoded['price'] )
```

In [ ]:

In [49]:
```
sns.distplot( df_encoded['price'] )
```

Out[49]: <AxesSubplot:xlabel='price', ylabel='Density'>



4.3 Feature Selection

4.3.1 Feature Selection using KNN and SFS

In [50]:

```
RANDOM_SEED = 42
knn = KNeighborsClassifier(n_neighbors=4)

X = df_encoded.drop('price', axis=1)
y = df_encoded['price']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

#Using KNN
sfs1 = SequentialFeatureSelector(knn,
           n_features_to_select=12,
           scoring='neg_mean_squared_error'
           )

sfs1 = sfs1.fit(X, y)
```

```
(197235, 22) (197235,)
(84530, 22) (84530,)
```

In [51]:

```
kcols = sfs1.get_feature_names_out()
```

4.3.2 Feature Selection using Permutation Importance

In [52]:
```
from sklearn.pipeline import make_pipeline

preprocessor = make_column_transformer(
    (StandardScaler(),['year','odometer']),
    remainder="passthrough",
    verbose_feature_names_out=False,  # avoid to prepend the preprocessor names
)

model = make_pipeline(
        preprocessor,
        TransformedTargetRegressor(
        regressor=Ridge(alpha=1e-10), func=np.log10, inverse_func=exp10
    )
)
model.fit(X_train, y_train)
y_pred = model.predict(X_train)
mae = median_absolute_error(y_train, y_pred)
string_score = f"MAE on training set: {mae:.2f} $/hour"
mse_train = mean_squared_error(y_train, y_pred)

y_pred = model.predict(X_test)
mae = median_absolute_error(y_test, y_pred)
string_score += f"\nMAE on testing set: {mae:.2f} $/hour"


mse_test = mean_squared_error(y_test,y_pred)
print(f'The mean squared error on the trained data is : {mse_train: .2f}')
print(f'The mean squared error on the test data is : {mse_test: .2f}')
```

```
The mean squared error on the trained data is :  0.42
The mean squared error on the test data is :  0.42
```

In [53]:
```
r = permutation_importance(model, X_test, y_test,n_repeats=30,random_state=RANDOM_SEED)
for i in r.importances_mean.argsort()[::-1]:
    if r.importances_mean[i] - 2 * r.importances_std[i] > 0:
        print(f"{X_test.columns[i]:<8}"
        f"  {r.importances_mean[i]:.3f}"
              f" +/- {r.importances_std[i]:.3f}")
```

```
drive_fwd  0.399 +/- 0.003
year      0.353 +/- 0.003
drive_4wd  0.087 +/- 0.001
cylinders  0.081 +/- 0.001
transmission_automatic  0.045 +/- 0.001
drive_rwd  0.039 +/- 0.001
odometer  0.038 +/- 0.001
condition  0.035 +/- 0.001
fuel_diesel  0.013 +/- 0.000
fuel_gas  0.009 +/- 0.000
transmission_manual  0.009 +/- 0.000
title_status  0.004 +/- 0.000
type      0.002 +/- 0.000
manufacturer  0.001 +/- 0.000
state     0.001 +/- 0.000
paint_color  0.001 +/- 0.000
transmission_other  0.001 +/- 0.000
fuel_electric  0.001 +/- 0.000
fuel_hybrid  0.000 +/- 0.000
size      0.000 +/- 0.000
city      0.000 +/- 0.000
```

In [54]:
```python
scoring = ['r2', 'neg_mean_absolute_percentage_error', 'neg_mean_squared_error']
r_multi = permutation_importance(model, X_test, y_test,n_repeats=30,random_state=RANDOM_SEED,scoring=scoring)
for metric in r_multi:
    print(f"-------------{metric}---------------")
    r = r_multi[metric]
    for i in r.importances_mean.argsort()[::-1]:
        if r.importances_mean[i] - 2 * r.importances_std[i] > 0:
            print(f"{X_test.columns[i]:<8}"
                  f"  {r.importances_mean[i]:.3f}"
                  f" +/- {r.importances_std[i]:.3f}")
```

```
-------------r2---------------
drive_fwd  0.399 +/- 0.003
year      0.353 +/- 0.003
drive_4wd  0.087 +/- 0.001
cylinders  0.081 +/- 0.001
transmission_automatic  0.045 +/- 0.001
drive_rwd  0.039 +/- 0.001
odometer  0.038 +/- 0.001
condition  0.035 +/- 0.001
fuel_diesel  0.013 +/- 0.000
fuel_gas  0.009 +/- 0.000
transmission_manual  0.009 +/- 0.000
title_status  0.004 +/- 0.000
type      0.002 +/- 0.000
manufacturer  0.001 +/- 0.000
state     0.001 +/- 0.000
paint_color  0.001 +/- 0.000
transmission_other  0.001 +/- 0.000
fuel_electric  0.001 +/- 0.000
fuel_hybrid  0.000 +/- 0.000
size      0.000 +/- 0.000
city      0.000 +/- 0.000
-------------neg_mean_absolute_percentage_error---------------
drive_fwd  0.020 +/- 0.000
year      0.018 +/- 0.000
cylinders  0.004 +/- 0.000
drive_4wd  0.004 +/- 0.000
drive_rwd  0.003 +/- 0.000
transmission_automatic  0.002 +/- 0.000
odometer  0.002 +/- 0.000
condition  0.002 +/- 0.000
fuel_diesel  0.001 +/- 0.000
transmission_manual  0.001 +/- 0.000
fuel_gas  0.001 +/- 0.000
title_status  0.000 +/- 0.000
```

```
transmission_other  0.000 +/- 0.000
paint_color  0.000 +/- 0.000
manufacturer  0.000 +/- 0.000
state     0.000 +/- 0.000
fuel_electric  0.000 +/- 0.000
fuel_hybrid  0.000 +/- 0.000
size      0.000 +/- 0.000
city      0.000 +/- 0.000
-------------neg_mean_squared_error---------------
drive_fwd  0.284 +/- 0.002
year      0.251 +/- 0.002
drive_4wd  0.062 +/- 0.001
cylinders  0.057 +/- 0.001
transmission_automatic  0.032 +/- 0.001
drive_rwd  0.028 +/- 0.000
odometer  0.027 +/- 0.000
condition  0.025 +/- 0.000
fuel_diesel  0.009 +/- 0.000
fuel_gas  0.007 +/- 0.000
transmission_manual  0.007 +/- 0.000
title_status  0.003 +/- 0.000
type      0.001 +/- 0.000
manufacturer  0.001 +/- 0.000
state     0.001 +/- 0.000
paint_color  0.001 +/- 0.000
transmission_other  0.001 +/- 0.000
fuel_electric  0.001 +/- 0.000
fuel_hybrid  0.000 +/- 0.000
size      0.000 +/- 0.000
city      0.000 +/- 0.000
```

Looking at the above we can say the following columns will have an impact on the prediction results final_columns = ['drive_rwd','drive_fwd','drive_4wd', 'fuel_gas','fuel_diesel', 'odometer','year', 'cylinders', 'condition','transmission_manual','type','title_status']

In [55]:
```
final_columns = ['drive_rwd','drive_fwd','drive_4wd', 'fuel_gas','fuel_diesel', 'odometer','year', 'cylinders',
final_columns_all = final_columns + ['price']
```

4.4 Models

In [56]:
```
def ml_error( model_name, y, yhat ):
    mae = mean_absolute_error( y, yhat )
    mape = mean_absolute_percentage_error( y, yhat )
    rmse = np.sqrt( mean_squared_error( y, yhat ) )
    mse = mean_squared_error(y, yhat)

    return pd.DataFrame( { 'Model Name': model_name,
                           'MAE': mae,
                           'MAPE': mape,
                           'MSE': mse,
                           'RMSE': rmse }, index=[0] )

def mean_percentage_error( y, yhat ):
    return np.mean( ( y - yhat ) / y )
```

In [57]:
```
# using final columns as features
X = df_encoded
y = np.log1p(df_encoded.price)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=RANDOM_SEED)
x_train = X_train[final_columns ]
x_test = X_test[final_columns ]


#using KNN columns as features
xk_train = X_train[kcols ]
xk_test = X_test[kcols ]
```

In [ ]:

4.4.1) Linear Regression

In [58]:
```
# In the beginning I use the Linear Model to see how linear or non-linear our dataset is.

# model
lr = LinearRegression().fit( x_train, y_train )

# prediction
yhat_lr = lr.predict( x_test )

# performance
lr_result1 = ml_error( 'Linear Regression', np.expm1( y_test ), np.expm1( yhat_lr ) )
lr_result1
```

Out[58]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Linear Regression	0.475208	0.052931	0.431789	0.657106

In [59]:
```
#using KNN
# model
lr = LinearRegression().fit( xk_train, y_train )

# prediction
yhat_lr = lr.predict( x_test )

# performance
lr_result2 = ml_error( 'Linear Regression with KNN Features', np.expm1( y_test ), np.expm1( yhat_lr ) )
lr_result2
```

Out[59]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Linear Regression with KNN Features	1.689028	0.19043	3.661722	1.913563

4.4.2) Linear Regression Model - Cross Validation

In [60]:
```
# k-fold CV (using all the 23 variables)
lm = LinearRegression()
scores = cross_val_score(lm, X_train, y_train, scoring='r2', cv=5)
scores
```

Out[60]:
```
array([0.99682226, 0.99681473, 0.99687368, 0.99675945, 0.9967948 ])
```

In [61]:
```
# create a KFold object with 5 splits
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
scores = cross_val_score(lm, X_train, y_train, scoring='r2', cv=folds)
scores
```

Out[61]:
```
array([0.99682007, 0.99676842, 0.9968159 , 0.9968659 , 0.99679319])
```

In [62]:
```
# can tune other metrics, such as MSE
scores = cross_val_score(lm, X_train, y_train, scoring='neg_mean_squared_error', cv=5)
scores
```

Out[62]:
```
array([-2.24672468e-05, -2.23684966e-05, -2.19326693e-05, -2.28778304e-05,
       -2.24255718e-05])
```

4.4.2.1) Hyperparameter tuning

A common use of cross-validation is for tuning hyperparameters of a model. The most common technique is what is called grid search cross-validation.

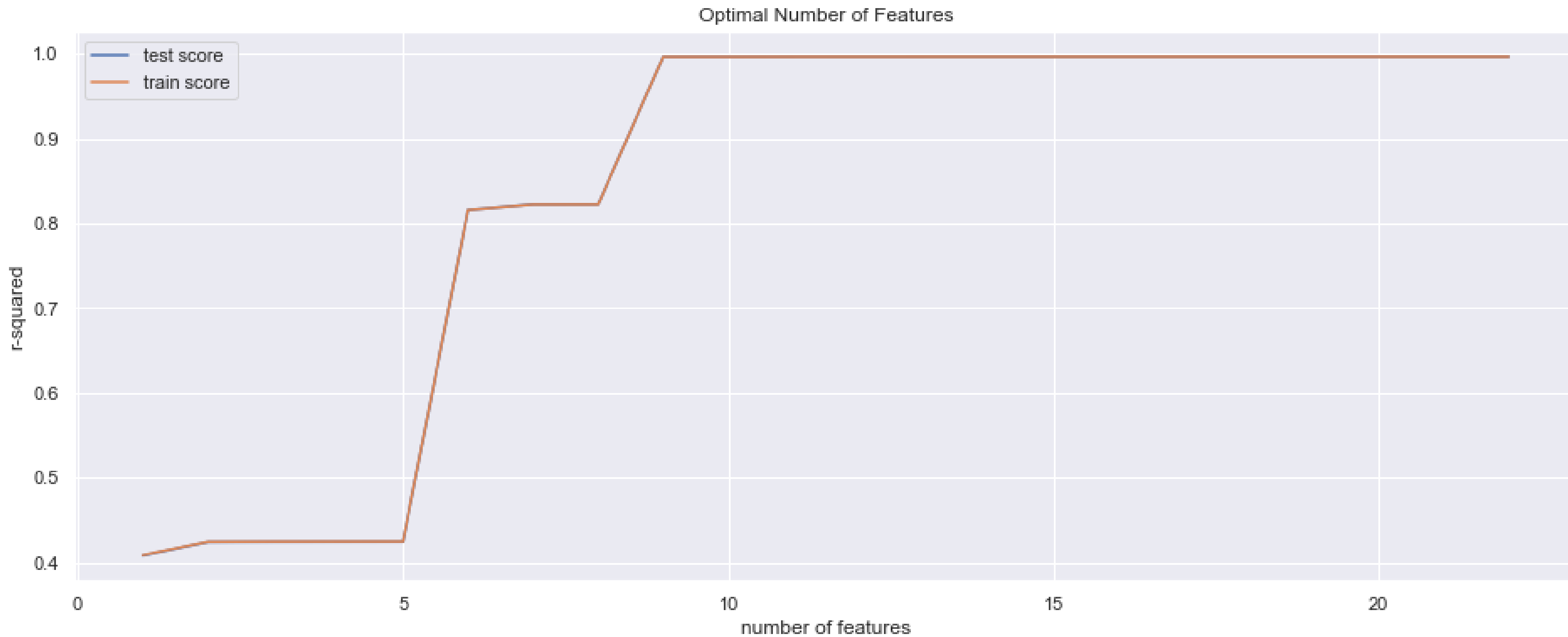
In [63]:
```python
# step-1: create a cross-validation scheme
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)

# step-2: specify range of hyperparameters to tune
hyper_params = [{'n_features_to_select': list(range(1, 23))}]


# step-3: perform grid search
# 3.1 specify model
lm = LinearRegression()
lm.fit(X_train, y_train)
rfe = RFE(lm)

# 3.2 call GridSearchCV()
model_cv = GridSearchCV(estimator = rfe,
                        param_grid = hyper_params,
                        scoring= 'r2',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 22 candidates, totalling 110 fits
```

Out[63]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=100, shuffle=True),
             estimator=RFE(estimator=LinearRegression()),
             param_grid=[{'n_features_to_select': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                   10, 11, 12, 13, 14, 15, 16,
                                                   17, 18, 19, 20, 21, 22]}],
             return_train_score=True, scoring='r2', verbose=1)
```

In [64]:
```
cv_results = pd.DataFrame(model_cv.cv_results_)
plt.figure(figsize=(16,6))

plt.plot(cv_results["param_n_features_to_select"], cv_results["mean_test_score"])
plt.plot(cv_results["param_n_features_to_select"], cv_results["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'], loc='upper left')
```

Out[64]:
```
<matplotlib.legend.Legend at 0x7f9a73d07be0>
```



Looking at the above plot, it is clear that optimal number of features is 9.

In [65]:
```
n_features_optimal = 9

lm = LinearRegression()
lm.fit(X_train, y_train)

rfe = RFE(lm, n_features_to_select=n_features_optimal)
rfe = rfe.fit(X_train, y_train)

# predict prices of X_test
y_pred = rfe.predict(X_test)
lr_result3 = ml_error( 'Linear Regression with with Kfold with 9 Features', np.expm1( y_test ), np.expm1( y_pred
lr_result3
```

Out[65]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Linear Regression with with Kfold with 9 Features	0.034345	0.003776	0.002141	0.046276

In [ ]:

4.4.3) Linear Regression - Lasso

In [66]:
```
# model
lrr = Lasso( alpha=0.00001 ).fit( x_train, y_train )

# prediction
yhat_lrr = lrr.predict( x_test )

# performance
lr_result4 = ml_error( 'Linear Regression - Lasso', np.expm1( y_test ), np.expm1( yhat_lrr ) )
lr_result4
```

Out[66]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Linear Regression - Lasso	0.475231	0.052934	0.43179	0.657107

In [67]:
```python
# model
lrr = Lasso( alpha=0.00001 ).fit( xk_train, y_train )

# prediction
yhat_lrr = lrr.predict( xk_test )

# performance
lr_result5 = ml_error( 'Linear Regression - Lasso with KNN features', np.expm1( y_test ), np.expm1( yhat_lrr ) )
lr_result5
```

Out[67]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Linear Regression - Lasso with KNN features	0.504285	0.056088	0.469406	0.685132

4.4.4) Ridge Regression

In [68]:
```python
rrr = Ridge()
rrr.fit(x_train,y_train)
yhat_rrr = rrr.predict(x_test)

lr_result6 = ml_error( 'Ridge Regression', np.expm1( y_test ), np.expm1( yhat_rrr ) )
lr_result6
```

Out[68]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Ridge Regression	0.475175	0.052928	0.431777	0.657097

In [69]:
```
rrr = Ridge()
rrr.fit(xk_train,y_train)
yhat_rrr = rrr.predict(xk_test)

lr_result7 = ml_error( 'Ridge Regression with KNN Features', np.expm1( y_test ), np.expm1( yhat_rrr ) )
lr_result7
```

Out[69]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Ridge Regression with KNN Features	0.504236	0.056083	0.46939	0.685121

4.4.5) Ridge Regression with GridSearchCV

In [70]:
```
ridge_param_dict = {'ridge__alpha': np.logspace(0, 10, 50) }
ridge_pipe = Pipeline([('scaler', StandardScaler()),
                       ('ridge',Ridge())
                      ])
ridge_grid = GridSearchCV(ridge_pipe, ridge_param_dict)
ridge_grid.fit(x_train, y_train)
ridge_test_preds = ridge_grid.predict(x_test)

lr_result8 = ml_error( 'Ridge Regression - Grid Search CV', np.expm1( y_test ), np.expm1( ridge_test_preds ) )
lr_result8
```

Out[70]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Ridge Regression - Grid Search CV	0.475269	0.052938	0.431805	0.657119

In [71]:
```
ridge_grid = GridSearchCV(ridge_pipe, ridge_param_dict)
ridge_grid.fit(xk_train, y_train)
ridge_test_preds = ridge_grid.predict(xk_test)

lr_result9 = ml_error( 'Ridge Regression - Grid Search CV with KNN features', np.expm1( y_test ), np.expm1( ridg
lr_result9
```

Out[71]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Ridge Regression - Grid Search CV with KNN fea...	0.504348	0.056095	0.469429	0.685149

4.5.6) Ridge Regression - Transformed Target Regressor

In [72]:

```
preprocessor = make_column_transformer(
    (StandardScaler(),['year','odometer']),
    remainder="passthrough",
    verbose_feature_names_out=False,  # avoid to prepend the preprocessor names
)

model = make_pipeline(
        preprocessor,
        TransformedTargetRegressor(
        regressor=Ridge(alpha=1e-10), func=np.log10, inverse_func=exp10
    )
)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

lr_result10 = ml_error( 'Ridge Regression - Transformed Target Regressor', np.expm1( y_test ), np.expm1( y_pred
lr_result10
```

Out[72]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Ridge Regression - Transformed Target Regressor	0.475604	0.052896	0.428134	0.654319

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.

5.1 Metric Evaluation

5.1.2 Identification of Evaluation Metric

MAE: Absolute value of the difference between the real and the predicted number and divides it by the number of predictions, that is, for each value that the model predicts it varies the MAE value on average (both for more and for less).Due to the way the MAE is calculated, it is not sensitive to outliers.

MAPE: The MAPE simply represents the percentage of the MAE, that is, how much the error that the model has means in percentage of the real value.

RMSE: This is the error most used to measure the performance of the model and its value serves as a parameter in the process of trying to decrease the model error within the project.This high use of RMSE is due to the fact that it is sensitive to outliers and this helps data scientists to be more rigorous with model errors.

MSE: measures the amount of error in statistical models. It assesses the average squared difference between the observed and predicted values. When a model has no error, the MSE equals zero.

5.1.2) Rationale behind use of evaluation metric

Since the projects contains outliers, we will be using RMSE to determine the right model.

In [73]:
```
modelling_result = pd.concat( [lr_result1, lr_result2, lr_result3, lr_result4, lr_result5, lr_result6, lr_result
modelling_result.sort_values( 'RMSE' )
```

Out[73]:

	Model Name	MAE	MAPE	MSE	RMSE
0	Linear Regression with with Kfold with 9 Features	0.034345	0.003776	0.002141	0.046276
0	Ridge Regression - Transformed Target Regressor	0.475604	0.052896	0.428134	0.654319
0	Ridge Regression	0.475175	0.052928	0.431777	0.657097
0	Linear Regression	0.475208	0.052931	0.431789	0.657106
0	Linear Regression - Lasso	0.475231	0.052934	0.431790	0.657107
0	Ridge Regression - Grid Search CV	0.475269	0.052938	0.431805	0.657119
0	Ridge Regression with KNN Features	0.504236	0.056083	0.469390	0.685121
0	Linear Regression - Lasso with KNN features	0.504285	0.056088	0.469406	0.685132
0	Ridge Regression - Grid Search CV with KNN fea...	0.504348	0.056095	0.469429	0.685149
0	Linear Regression with KNN Features	1.689028	0.190430	3.661722	1.913563

Linear Regression with K-fold seems to be the one with lowest RMSE. We will be taking that to move ahead with our evaulation

In [74]:
```
n_features_optimal = 9

lm = LinearRegression()
lm.fit(X_train, y_train)

rfe = RFE(lm, n_features_to_select=n_features_optimal)
rfe = rfe.fit(X_train, y_train)

# predict prices of X_test
y_pred = rfe.predict(X_test)

mpe = mean_percentage_error( np.expm1( y_test ), np.expm1(y_pred) )
mpe
```

Out[74]:
```
-2.995707850959188e-05
```

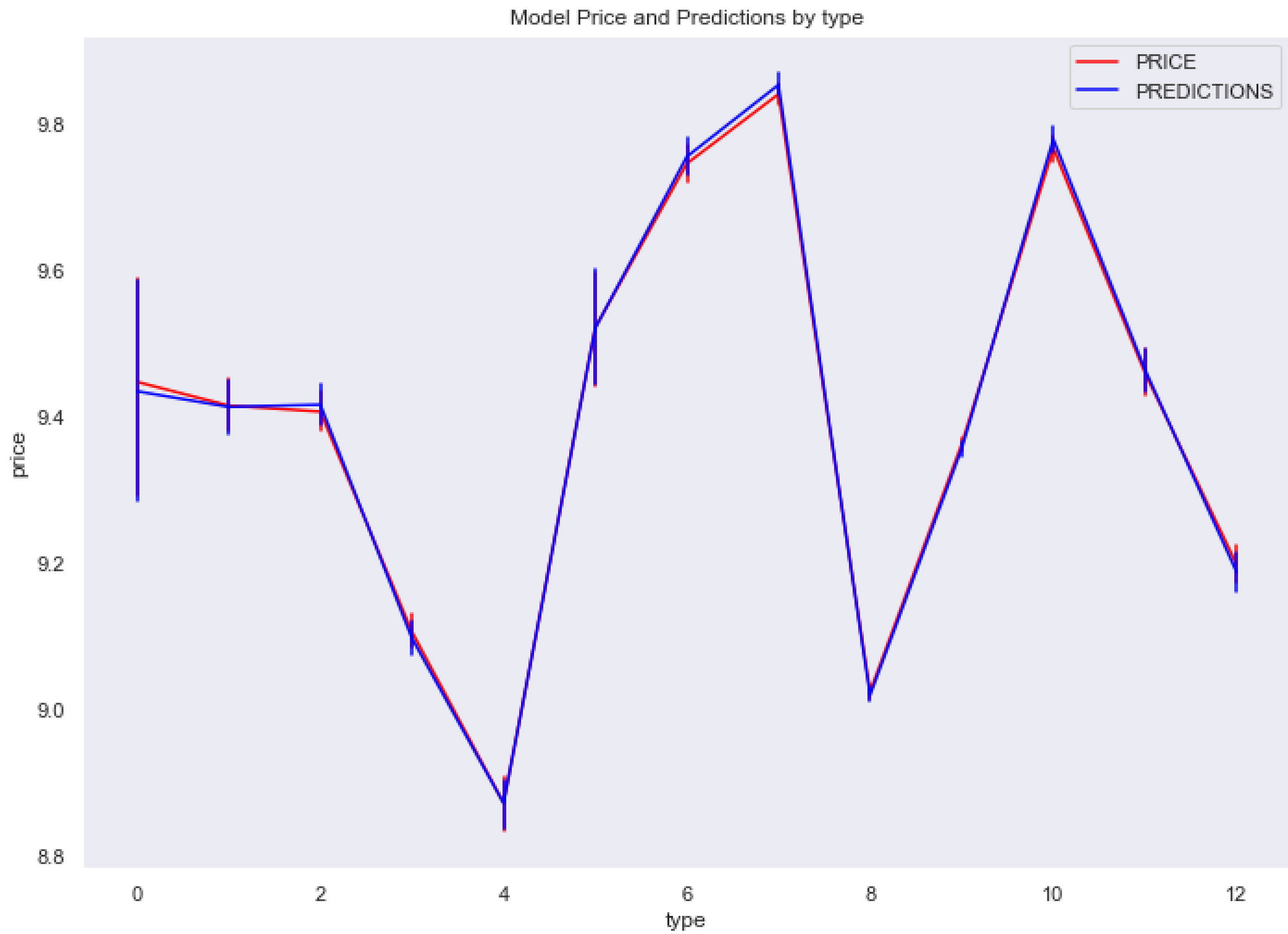
As the mean percentage error is closer to zero, it does not influence much.

In [75]:
```
df_final = X_test[ final_columns_all ]

# rescale
#df_final['price'] = np.expm1( df_final['price'] )
df_final['predictions'] = np.expm1( y_pred );
```

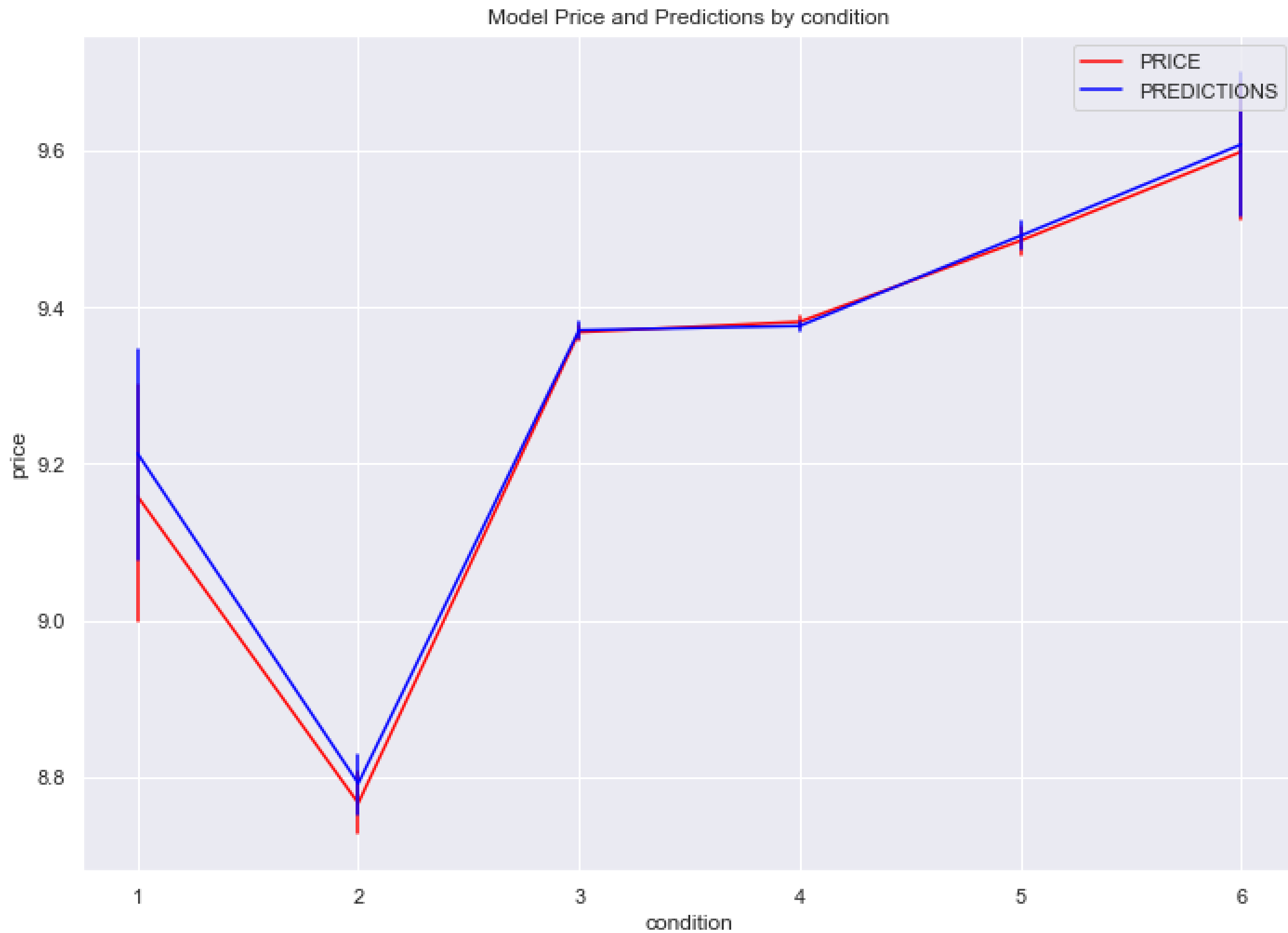
In [76]:
```
df_final['error'] = df_final['price'] - df_final['predictions']
df_final['error_rate'] = df_final['predictions'] / df_final['price']
```

In [ ]:
```

```

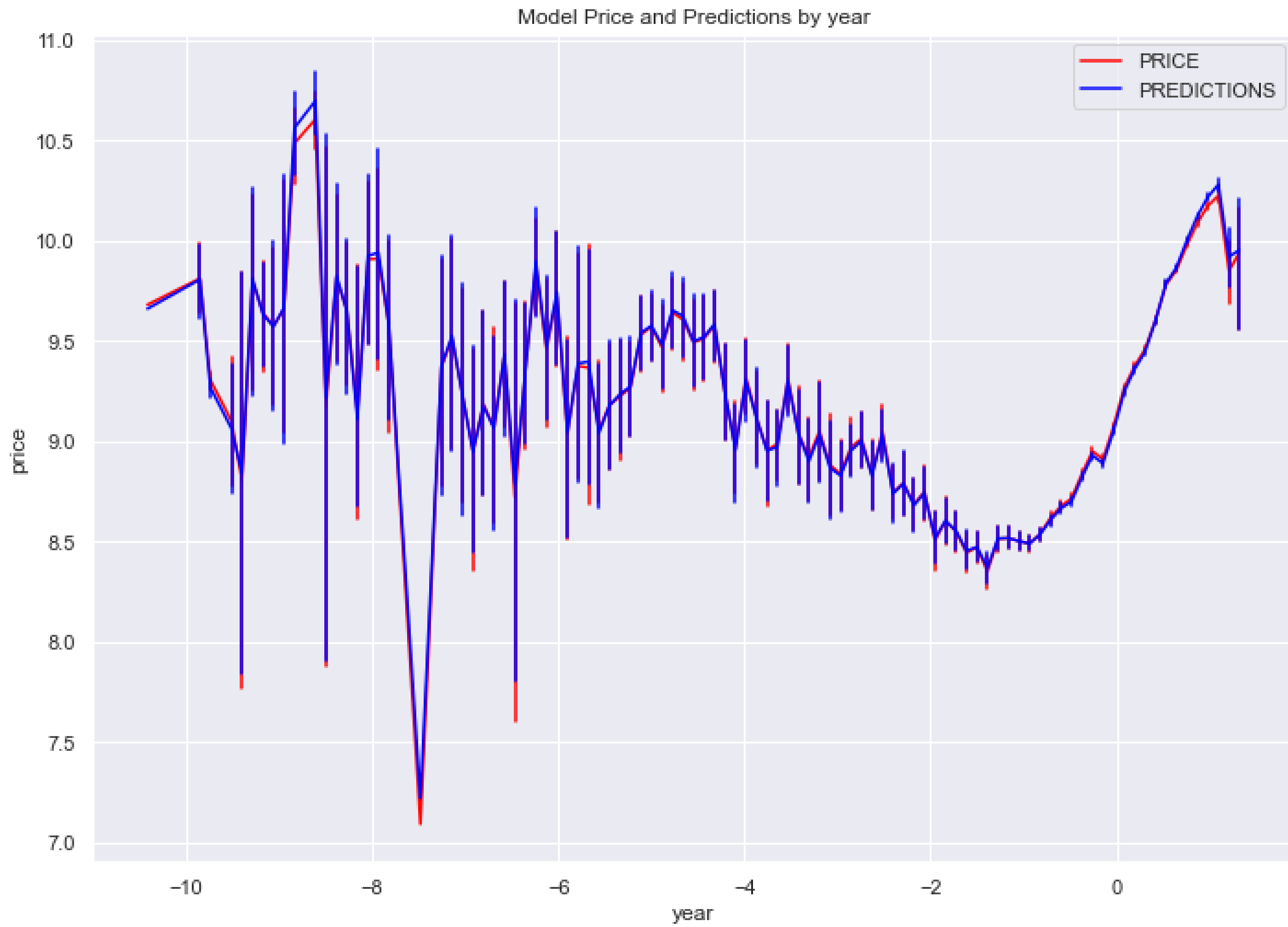
In [77]:
```
sns.set_style("whitegrid", {'axes.grid' : False})
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot( x='type', y='price', data=df_final, label='PRICE',color='red',err_style='bars' )
fig = sns.lineplot( x='type', y='predictions', data=df_final, label='PREDICTIONS',color='blue',err_style='bars'
fig.grid(False)
fig.title.set_text('Model Price and Predictions by type')
```



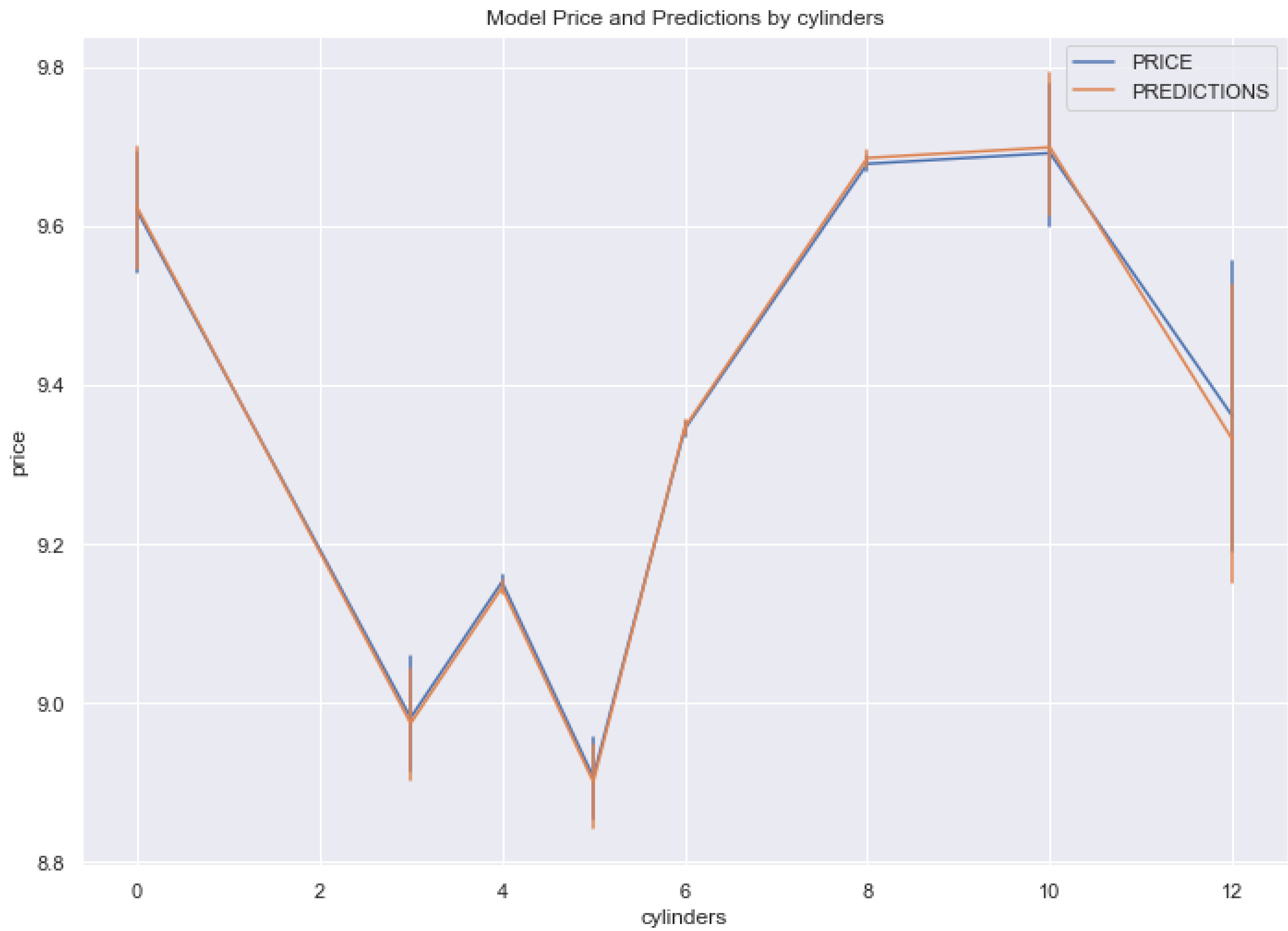
In [78]:
```
sns.set_style("whitegrid", {'axes.grid' : False})
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot( x='condition', y='price', data=df_final, label='PRICE', color='red',err_style='bars')
fig = sns.lineplot( x='condition', y='predictions', data=df_final, label='PREDICTIONS', color='blue',err_style='l
fig.title.set_text('Model Price and Predictions by condition')
```



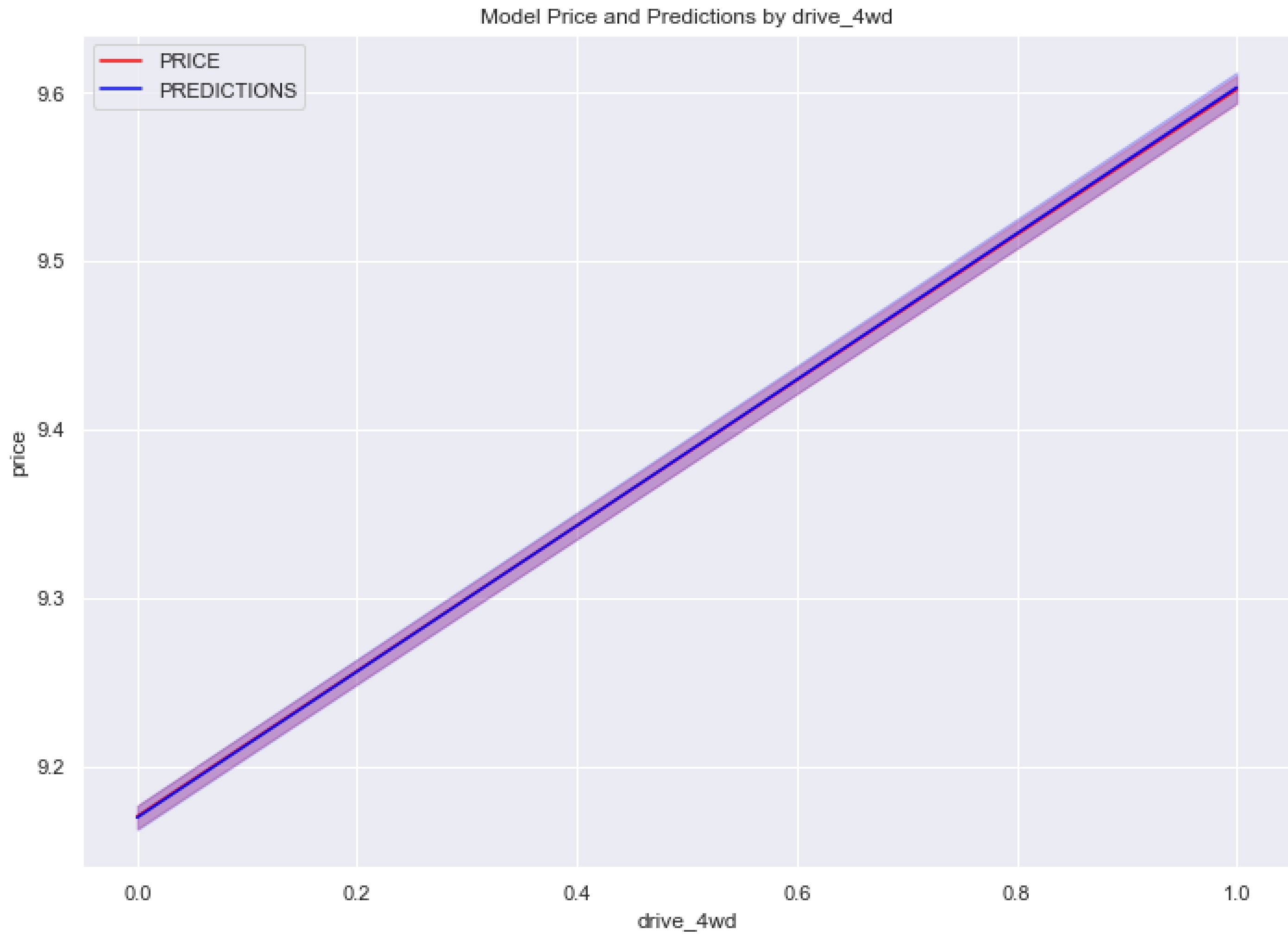
In [79]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.color_palette("husl", 9)
sns.lineplot( x='year', y='price', data=df_final, label='PRICE',color='red',err_style='bars')
fig = sns.lineplot( x='year', y='predictions', data=df_final, label='PREDICTIONS', color='blue',err_style='bars'
fig.title.set_text('Model Price and Predictions by year')
```



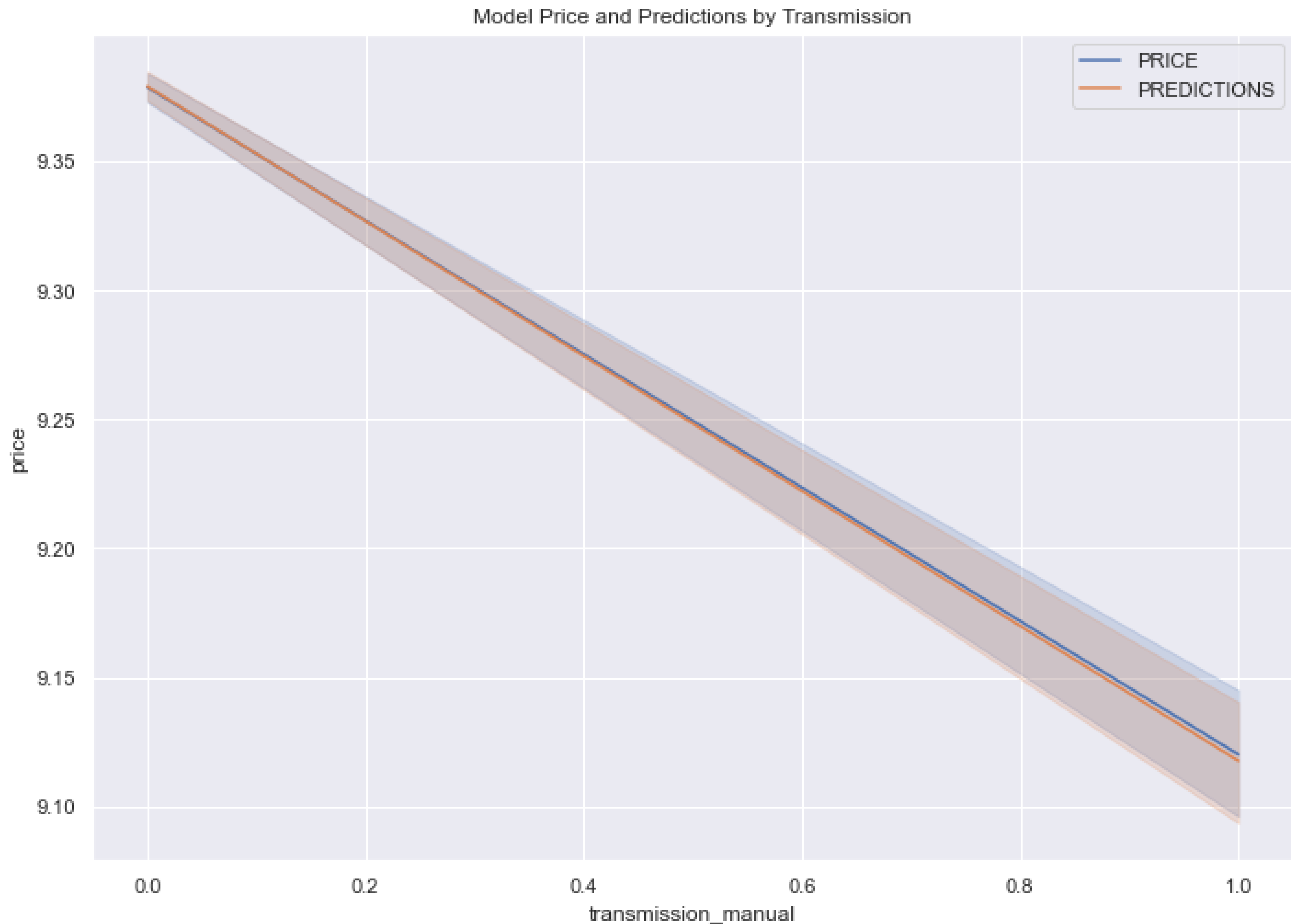
In [80]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot( x='cylinders', y='price', data=df_final, label='PRICE',err_style='bars' )
fig = sns.lineplot( x='cylinders', y='predictions', data=df_final, label='PREDICTIONS' ,err_style='bars')
fig.title.set_text('Model Price and Predictions by cylinders')
```



In [81]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot( x='drive_4wd', y='price', data=df_final, label='PRICE',color='red' )
fig = sns.lineplot( x='drive_4wd', y='predictions', data=df_final, label='PREDICTIONS',color='blue' )
fig.title.set_text('Model Price and Predictions by drive_4wd')
```



In [82]:
```
sns.lineplot( x='transmission_manual', y='price', data=df_final, label='PRICE' )
fig = sns.lineplot( x='transmission_manual', y='predictions', data=df_final, label='PREDICTIONS')
fig.title.set_text('Model Price and Predictions by Transmission')
```



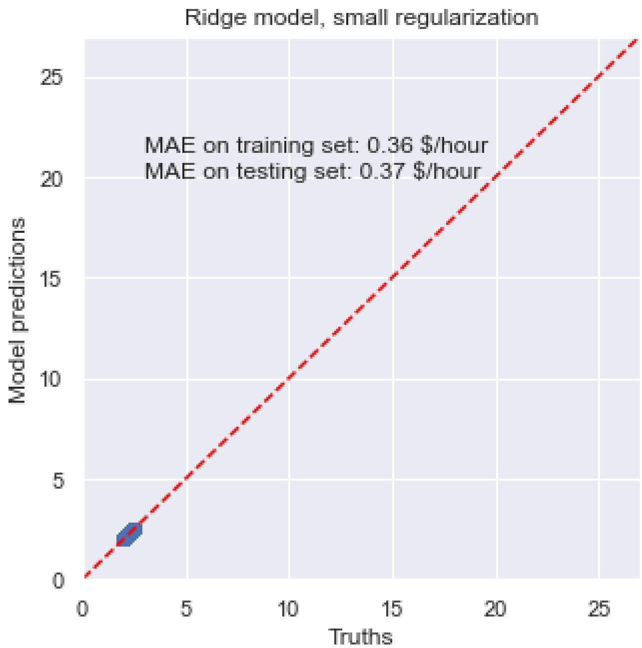
In [ ]:

In [ ]:

5.2 Interpretation of Coefficients

In [83]:

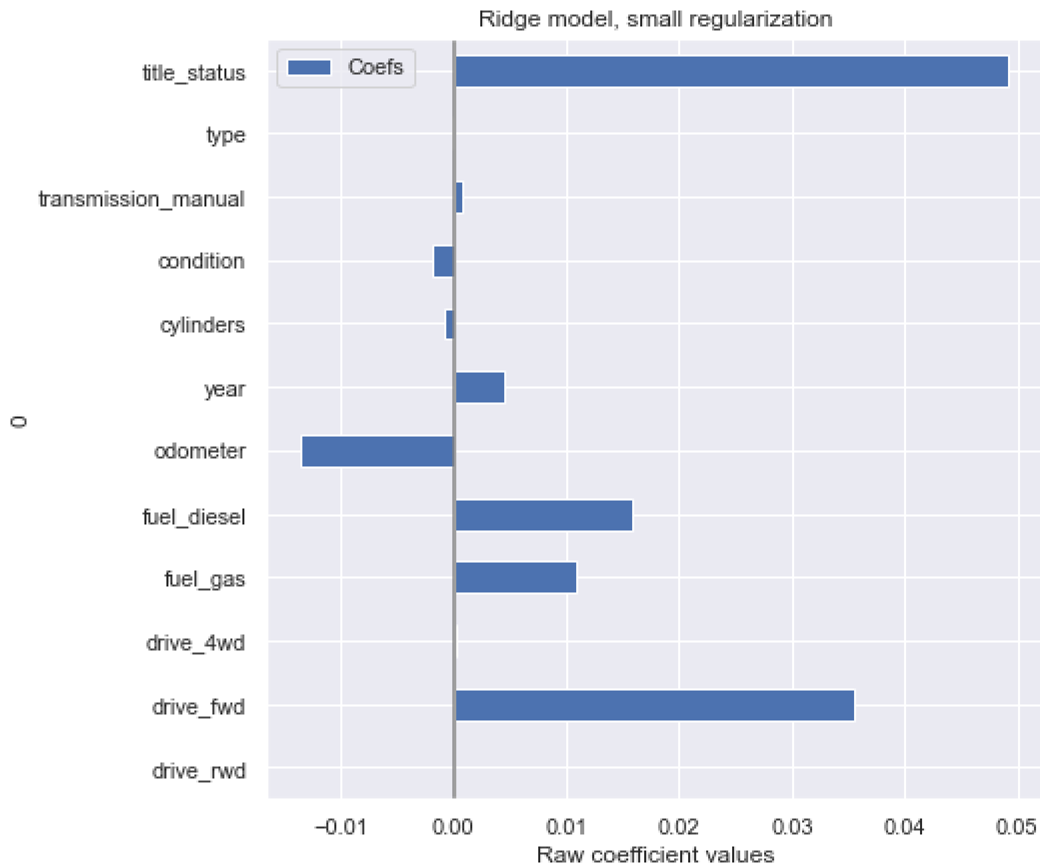
```
fig, ax = plt.subplots(figsize=(5, 5))
plt.scatter(y_test, y_pred)
ax.plot([0, 1], [0, 1], transform=ax.transAxes, ls="--", c="red")
plt.text(3, 20, string_score)
plt.title("Ridge model, small regularization")
plt.ylabel("Model predictions")
plt.xlabel("Truths")
plt.xlim([0, 27])
_ = plt.ylim([0, 27])
```



In [84]:
```
coef_table = pd.DataFrame(list(x_train.columns)).copy()
coef_table.insert(len(coef_table.columns),"Coefs",rrr.coef_.transpose())
coef_table = coef_table.set_index(0)
print(coef_table)
```

```
                       Coefs
0
drive_rwd          -0.000007
drive_fwd           0.035584
drive_4wd           0.000238
fuel_gas            0.010833
fuel_diesel         0.015801
odometer           -0.013522
year                0.004481
cylinders          -0.000766
condition          -0.001850
transmission_manual  0.000717
type               -0.000131
title_status        0.049182
```

In [85]:
```
coef_table.plot.barh(figsize=(9, 7))
plt.title("Ridge model, small regularization")
plt.axvline(x=0, color=".5")
plt.xlabel("Raw coefficient values")
plt.subplots_adjust(left=0.3)
plt.show()
```



From the plot above, the most important factor in determining the prices is year. Vehicles with fuel type gas also play an important role in driving the prices of used cars. Drive with type rwd, manual transmission and fwd are other factors that have a correlation with the overall used car price.

In [ ]:

```

```

Deployment

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.

The goal here is to make the prediction model accessible to anyone. To achieve this, an API is created.

The architecture of the model in production:

AWS SageMaker provides an HTTPS endpoint for your model, availing it to provide inferences in three steps:

1) Create the model in SageMaker, including the relevant S3 path and Docker registry path

2) Create an endpoint configuration for an HTTPS endpoint

3) Create an HTTPS endpoint

API -> is the part that receives the requests and plays for the other parts so that the data is processed and then brings everything together, returning the final answer.

Data Preparation -> all the treatments and modifications we made to the data will be kept inside. When the Handler receives the raw data it will throw it here within this list of treatment codes so that they are prepared so that they can be ready to be used within the Machine Learning model.

Model Training -> this is our trained model that has been saved and will be placed inside this folder in our production architecture. The Handler will take the data processed within Data Preparation and play it inside the model so that it provides the prediction.

At the end of the construction of all this architecture and being put into production, the way it will be visualized can be through an App, Dashboard or a website.

In [ ]:

```

```

6) Recommendations to the dealer

Summary

Detailed summary is located [here (https://github.com/spalakollu/Used-Cars/blob/main/Recommendations%20to%20the%20dealer.pdf)](https://github.com/spalakollu/Used-Cars/blob/main/Recommendations%20to%20the%20dealer.pdf)

Used vehicle market involves many factors when it comes to predicting the prices. Colors, vehicle type, manufacturer, model,price,transmission and vehicle ages are some of the significant factors we had a chance to analyze that affect the time a used vehicle stays in the lot. Few characteristics that are highly desirable among used vehicles that may help the dealer in determining the overall price.

1. Cars between 1980 and 2000 cost much lower compared to the cars from 2000.
2. Based on the data, the top 5 cars in the dataset are Chevrolet suv, ford truck, ford suv, jeep suv and ford pickup.
3. CA, NY and FL are the states where cars sales are more compared to the other states implying that price might be higher due to high demand.
4. Electric cars are expensive compared to gas, hybrid and diesel.
5. The price of cylinders is not directly correlated to the price of car.
6. Cars with color white, black, silver and grey are sold more compared to the other cars.
7. Cars with condition excellent are sold more compared to the cars good, new and like new.
8. SUVs and sedan-type vehicles are sold faster than Economy vehicles
9. Dealers should prefer to buy cars with low age with color black, while, or silver.
10. The top priced cars are tesla, Ferrari, ram and Porsche.

8) Next Steps

Future research is recommended to explore other factors that influence the sales period of a used vehicle. For example, the level of fuel-efficiency, level of discount from the original price. Incorporating these factors in the analysis can improve the accuracy to choose non-overage vehicles and have a positive impact on profit.

In [86]:
```
df_encoded.head()
```

Out[86]:

	city	price	year	manufacturer	condition	cylinders	odometer	title_status	size	type	...	fuel_electric	fuel_gas	fuel_hybrid	fuel_
0	258	10.551925	-12.447444	0	6	4	-0.588884	6	2	9	...	0.0	1.0	0.0	
1	59	8.291797	-11.883123	7	4	8	0.519987	6	1	11	...	0.0	1.0	0.0	
2	100	10.203629	-10.980210	13	5	4	55.369730	6	0	1	...	0.0	1.0	0.0	
3	27	8.517393	-10.754482	13	3	4	-0.544965	6	1	6	...	0.0	1.0	0.0	
4	119	9.680406	-10.415889	13	3	4	-0.300568	6	1	1	...	0.0	1.0	0.0	

5 rows × 23 columns

In [ ]:

In [ ]:

In [ ]:

In [ ]: