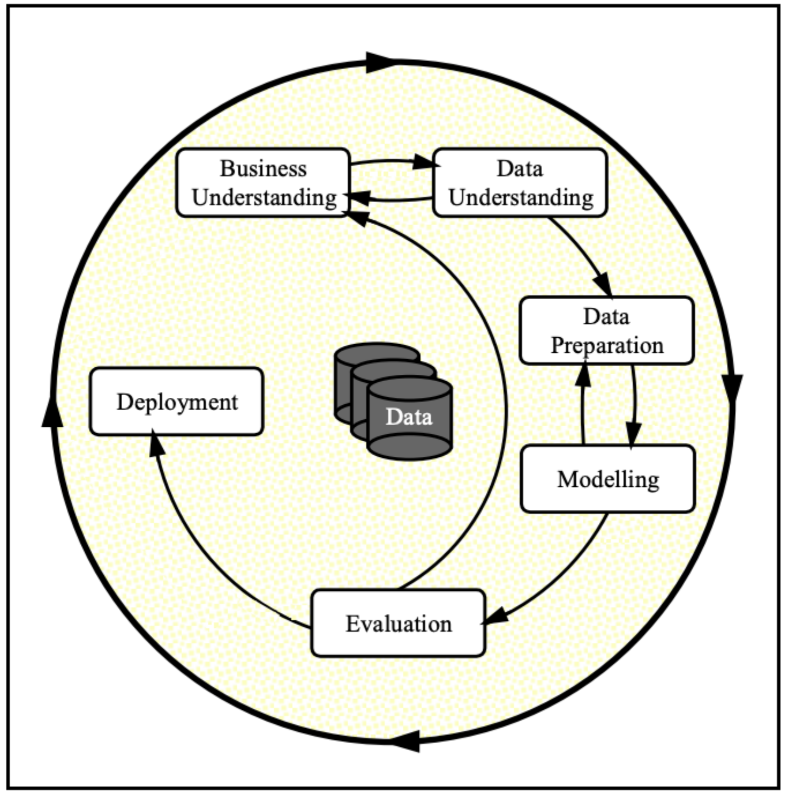
What drives the price of a car?

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.

Providing accurate prices to used cars can expedite business transactions, and guarantee fairness to both business owners and customers. There are many factors and the car's own features that determine the sale price of a used car. Based on the transactional data collected on historical car sales from different regions, we are expect to build a car sale price prediction model to accurately reflect the true value of a used car.

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.

The data were based on many used-car sales transactions that recorded itemized and detailed features of a car as well as the locations and its final sale price.

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 [1]:

```
# Import necessary libraries on models and tools.
from sklearn import set_config
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.feature_selection import (SelectFromModel,
                                       SequentialFeatureSelector)
from sklearn.linear_model import Lasso, LinearRegression, Ridge
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import (OneHotEncoder, OrdinalEncoder,
                                   PolynomialFeatures, StandardScaler,
                                   FunctionTransformer)
from sklearn.inspection import permutation_importance

set_config(display="diagram")

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import seaborn as sns
```

In [2]:

```
# Initial inspection of the data
pd.set_option('display.max_columns', None)
cars = pd.read_csv('data/vehicles.csv')
print(cars.head())
print(cars.info)
```

```
           id                  region  price  year manufacturer model  \
0  7222695916                prescott   6000   NaN          NaN   NaN
1  7218891961            fayetteville  11900   NaN          NaN   NaN
2  7221797935            florida keys  21000   NaN          NaN   NaN
3  7222270760  worcester / central MA   1500   NaN          NaN   NaN
4  7210384030              greensboro   4900   NaN          NaN   NaN

  condition cylinders fuel  odometer title_status transmission  VIN drive  \
0       NaN       NaN  NaN       NaN          NaN          NaN  NaN   NaN
1       NaN       NaN  NaN       NaN          NaN          NaN  NaN   NaN
2       NaN       NaN  NaN       NaN          NaN          NaN  NaN   NaN
3       NaN       NaN  NaN       NaN          NaN          NaN  NaN   NaN
4       NaN       NaN  NaN       NaN          NaN          NaN  NaN   NaN

  size type paint_color state
0  NaN  NaN         NaN    az
1  NaN  NaN         NaN    ar
2  NaN  NaN         NaN    fl
3  NaN  NaN         NaN    ma
4  NaN  NaN         NaN    nc
<bound method DataFrame.info of                 id                  region  price    year manufacturer  
\
0       7222695916                prescott   6000     NaN          NaN
1       7218891961            fayetteville  11900     NaN          NaN
2       7221797935            florida keys  21000     NaN          NaN
3       7222270760  worcester / central MA   1500     NaN          NaN
4       7210384030              greensboro   4900     NaN          NaN
...            ...                     ...    ...     ...          ...
426875  7301591192                 wyoming  23590  2019.0       nissan
426876  7301591187                 wyoming  30590  2020.0        volvo
426877  7301591147                 wyoming  34990  2020.0     cadillac
426878  7301591140                 wyoming  28990  2018.0        lexus
426879  7301591129                 wyoming  30590  2019.0          bmw

                           model condition    cylinders    fuel  odometer  \
0                            NaN       NaN          NaN     NaN       NaN
1                            NaN       NaN          NaN     NaN       NaN
```

```
2                           NaN       NaN          NaN     NaN       NaN
3                           NaN       NaN          NaN     NaN       NaN
4                           NaN       NaN          NaN     NaN       NaN
...                         ...       ...          ...     ...       ...
426875          maxima s sedan 4d      good  6 cylinders     gas   32226.0
426876  s60 t5 momentum sedan 4d      good          NaN     gas   12029.0
426877           xt4 sport suv 4d      good          NaN  diesel    4174.0
426878            es 350 sedan 4d      good  6 cylinders     gas   30112.0
426879  4 series 430i gran coupe      good          NaN     gas   22716.0

       title_status transmission                VIN drive size       type  \
0               NaN          NaN                NaN   NaN  NaN        NaN
1               NaN          NaN                NaN   NaN  NaN        NaN
2               NaN          NaN                NaN   NaN  NaN        NaN
3               NaN          NaN                NaN   NaN  NaN        NaN
4               NaN          NaN                NaN   NaN  NaN        NaN
...             ...          ...                ...   ...  ...        ...
426875        clean        other  1N4AA6AV6KC367801   fwd  NaN      sedan
426876        clean        other  7JR102FKXLG042696   fwd  NaN      sedan
426877        clean        other  1GYFZFR46LF088296   NaN  NaN  hatchback
426878        clean        other  58ABK1GG4JU103853   fwd  NaN      sedan
426879        clean        other  WBA4J1C58KBM14708   rwd  NaN      coupe

       paint_color state
0              NaN    az
1              NaN    ar
2              NaN    fl
3              NaN    ma
4              NaN    nc
...            ...   ...
426875         NaN    wy
426876         red    wy
426877       white    wy
426878      silver    wy
426879         NaN    wy

[426880 rows x 18 columns]>
```

There are 18 columns of features in each record.

- Some are irrelevant, such as ID and VIN, which can be dropped.
- Many are nominal data, such as fuel, state, drive, which are to be transformed using OneHotEncoder.

- Some are ordinal data, such as cylinders, condition, which are to be transformed using OrdinalEncoder.
- There are also records that are missing critical information, and should be eliminated from the modeling, such as rows with NaN values, or price==0, or odometer==1.
- EVs have no cylinders. Therefore, the cylinder value is turned to 0.
- Cars costing more than 100,000 is not considered because of insufficient data above that range.
- Cars that has 4 million miles odometer are considered outliers, and are removed as well.
- The values of 'region' is 403, and there is no proper way of converting a value of 'region' into a numeric value, therefore it is dropped.
- Similar to 'region', 'model' is also dropped.

In [3]:

```
print(cars.columns)
for i in cars.columns:
    print(f'Value of {i} is {len(cars[i].unique())}')
```

```
Index(['id', 'region', 'price', 'year', 'manufacturer', 'model', 'condition',
       'cylinders', 'fuel', 'odometer', 'title_status', 'transmission', 'VIN',
       'drive', 'size', 'type', 'paint_color', 'state'],
      dtype='object')
Value of id is 426880
Value of region is 404
Value of price is 15655
Value of year is 115
Value of manufacturer is 43
Value of model is 29650
Value of condition is 7
Value of cylinders is 9
Value of fuel is 6
Value of odometer is 104871
Value of title_status is 7
Value of transmission is 4
Value of VIN is 118247
Value of drive is 4
Value of size is 5
Value of type is 14
Value of paint_color is 13
Value of state is 51
```

In [4]:

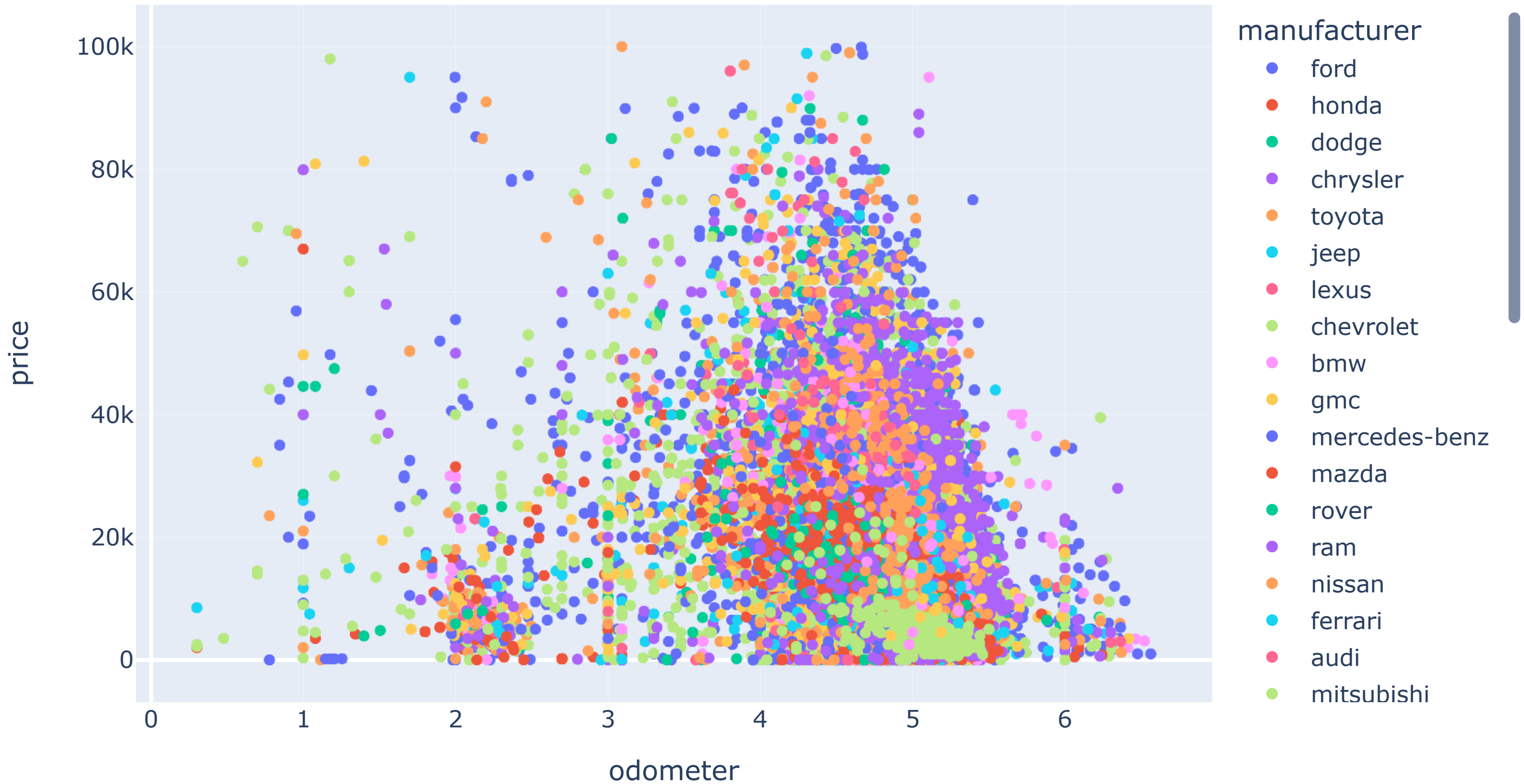
```
cars_no_id_nan = cars.drop(columns=['id', 'VIN', 'region', 'model']).dropna()
print(cars_no_id_nan.columns)
print(len(cars_no_id_nan))
cars_cleaned = cars_no_id_nan[(cars_no_id_nan['price'] > 0) &
                              (cars_no_id_nan['odometer'] > 1) &
                              (cars_no_id_nan['price'] < 100000) &
                              (cars_no_id_nan['odometer'] < 4000000)].copy()
cars_cleaned['cylinders'] = cars_cleaned['cylinders'].str.replace(' cylinders', '')
cars_cleaned['cylinders'] = cars_cleaned['cylinders'].str.replace('other', '0')
cars_cleaned['cylinders'] = cars_cleaned['cylinders'].astype(int)
cars_cleaned['odometer'] = np.log10(cars_cleaned['odometer'])

print(len(cars_cleaned))

# Convert string values to ordinal values.
ordinal_features = ['condition', 'size']
numeric_features = ['year', 'cylinders', 'odometer']
nominal_features = ['manufacturer', 'fuel', 'title_status',
                    'transmission', 'drive', 'type',
                    'paint_color', 'state']

#print(cars_cleaned.columns)
#help(OrdinalEncoder)
ord_condition = OrdinalEncoder(categories=[
    ['salvage', 'fair', 'good', 'like new', 'excellent', 'new'],
    ['sub-compact', 'compact', 'mid-size', 'full-size']
])
#print(cars_cleaned[['condition', 'size']].head())
cars_cleaned[['condition', 'size']] = ord_condition.fit_transform(
    cars_cleaned[['condition', 'size']])
cars_final = cars_cleaned
```

```
Index(['price', 'year', 'manufacturer', 'condition', 'cylinders', 'fuel',
       'odometer', 'title_status', 'transmission', 'drive', 'size', 'type',
       'paint_color', 'state'],
      dtype='object')
80170
76333
```

In [5]:
```
#print(len(cars_cleaned))
print(cars_final.iloc[0])
print(cars_final.columns)
```

```
price                15000
year                2013.0
manufacturer          ford
condition              4.0
cylinders                6
fuel                   gas
odometer           5.10721
title_status         clean
transmission     automatic
drive                  rwd
size                   3.0
type                 truck
paint_color          black
state                   al
Name: 31, dtype: object
Index(['price', 'year', 'manufacturer', 'condition', 'cylinders', 'fuel',
       'odometer', 'title_status', 'transmission', 'drive', 'size', 'type',
       'paint_color', 'state'],
      dtype='object')
```

In [6]:
```
px.scatter(cars_cleaned, x='odometer', y='price',
           color='manufacturer',
           width=800, height=500)
```



In [7]:
```
cars_cleaned['odometer'].unique()
```

Out[7]:
```
array([5.10720997, 4.94448267, 4.97772361, ..., 5.11602963, 5.24615623,
       5.23681679])
```

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 [8]:
```
X_train, X_test, y_train, y_test = train_test_split(
    cars_final.drop(columns=['price']), cars_final['price'],
    test_size=0.25, random_state=42, shuffle=True)
print(type(X_train), type(X_test), type(y_train), type(y_test))
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

```
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.frame.DataFrame'> <class 'pandas.core.serie
s.Series'> <class 'pandas.core.series.Series'>
57249 19084 57249 19084
```

In [ ]:
```

```

In [9]:
```
poly_features = ['condition', 'size', 'cylinders', 'odometer']
col_transformer = make_column_transformer(
    (StandardScaler(), poly_features),
    (PolynomialFeatures(degree=2, include_bias=False), poly_features),
    (OneHotEncoder(drop='if_binary'), nominal_features),
    remainder='passthrough')
```

In [10]:

```
pipeline1 = Pipeline([
    ('transform', col_transformer),
    ('ridge', Ridge())
])
param_grid={'ridge__alpha': [1, 10, 100]}
gridsearch = GridSearchCV(pipeline1, param_grid=param_grid,
                          scoring='neg_mean_squared_error')
gridsearch.fit(X_train, y_train)

y_predict_train = gridsearch.predict(X_train)
y_predict_test = gridsearch.predict(X_test)
print(f'Train MSE: {mean_squared_error(y_train, y_predict_train)}')
print(f'Test MSE: {mean_squared_error(y_test, y_predict_test)}')
print(gridsearch.best_params_)

r = permutation_importance(gridsearch, X_test,
                           y_test, n_repeats=20, random_state=42)
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]:<30}"
              f"{r.importances_mean[i]:.3f}"
              f" +/- {r.importances_std[i]:.3f}")
```

```
Train MSE: 56855628.84290065
Test MSE: 58062463.447030865
{'ridge__alpha': 10}
odometer                      64217385.085 +/- 1023765.400
fuel                          17064397.093 +/- 431285.817
year                          14619457.444 +/- 267702.373
type                          9704702.803 +/- 235763.137
cylinders                     7069078.524 +/- 343713.314
drive                         6256767.346 +/- 255347.642
condition                     5197772.622 +/- 301586.363
manufacturer                  3354563.337 +/- 89725.821
transmission                  2807826.565 +/- 118669.585
state                         2781727.188 +/- 151461.572
size                          2619337.647 +/- 201362.199
title_status                  784265.153 +/- 68011.137
paint_color                   489062.680 +/- 56671.728
```

In [11]:
```
pipeline2 = Pipeline([
    ('transform', col_transformer),
    ('selector', SequentialFeatureSelector(LinearRegression(),
                                           n_features_to_select=10)),
    ('linreg', LinearRegression())
])
pipeline2.fit(X_train, y_train)
print(pipeline2.named_steps['selector'].get_feature_names_out())

y_predict_train = pipeline2.predict(X_train)
y_predict_test = pipeline2.predict(X_test)
print(f'Train MSE: {mean_squared_error(y_train, y_predict_train)}')
print(f'Test MSE: {mean_squared_error(y_test, y_predict_test)}')

r = permutation_importance(pipeline2, X_test, y_test,
                           n_repeats=20, random_state=42)
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]:<30}"
              f"{r.importances_mean[i]:.3f}"
              f" +/- {r.importances_std[i]:.3f}")
```

```
['x3' 'x10' 'x13' 'x17' 'x59' 'x72' 'x74' 'x85' 'x86' 'x152']
Train MSE: 65373952.930877835
Test MSE: 66130608.32385149
odometer                      0.419 +/- 0.006
fuel                          0.128 +/- 0.003
year                          0.088 +/- 0.002
type                          0.062 +/- 0.002
cylinders                     0.046 +/- 0.002
drive                         0.036 +/- 0.002
condition                     0.023 +/- 0.001
transmission                  0.019 +/- 0.001
size                          0.009 +/- 0.001
```

In [12]:
```
print(y_test.iloc[101], y_predict_test[101])
```

```
1000 17314.43173201161
```

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.

In [13]:
```
gridsearch
```

Out[13]:
```
GridSearchCV
  transform: ColumnTransformer
    standardscaler      polynomialfeatures      onehotencoder      remainder
    StandardScaler      PolynomialFeatures      OneHotEncoder      passthrough
  Ridge
```

In [14]:
```
pipeline2
```

Out[14]:
```
Pipeline
  transform: ColumnTransformer
    standardscaler      polynomialfeatures      onehotencoder      remainder
    StandardScaler      PolynomialFeatures      OneHotEncoder      passthrough
  selector: SequentialFeatureSelector
    LinearRegression
  LinearRegression
```

Through this CRIPS-DM exercise, we got to familiarize ourselves with the complete process of data mining, and realized the importance of data preprocessing before exploring regression models to fit the data.

According to similar results of two slightly different regression and prediction models, we confirmed the validity and accuracy of each of the data analysis methods.

Going back to the original business drivers behind the data mining project, we can offer some valuable guidance in considerations of used car features in the determination of their sale prices, which include the following items in the order of decending weight:

- odometer reading
- fuel type
- body type
- age/year of production
- number of cylinders
- drive
- condition

With this simplified view of the data, we rebuild the data model and estimation methods, hoping getting a more accurate estimate of the car price.

In [15]:

```
# Input data.
cars = pd.read_csv('data/vehicles.csv')

# Remove invalid or irrelevant features.
cars_no_id_nan = cars.drop(columns=['id', 'VIN', 'region', 'model']).dropna()
cars_cleaned = cars_no_id_nan[(cars_no_id_nan['price'] > 0) &
                              (cars_no_id_nan['odometer'] > 1) &
                              (cars_no_id_nan['price'] < 100000) &
                              (cars_no_id_nan['odometer'] < 4000000)].copy()
cars_cleaned['cylinders'] = cars_cleaned['cylinders'].str.replace(' cylinders', '')
cars_cleaned['cylinders'] = cars_cleaned['cylinders'].str.replace('other', '0')
cars_cleaned['cylinders'] = cars_cleaned['cylinders'].astype(int)

# Convert ordinal values.
condition_vals = [['salvage', 'fair', 'good', 'like new', 'excellent', 'new']]
ord_transformer = make_column_transformer(
    (OrdinalEncoder(categories=condition_vals), ['condition']))
cars_cleaned['condition'] = ord_transformer.fit_transform(cars_cleaned)
cars_cleaned['odometer'] = np.log10(cars_cleaned['odometer'])

# Separate different types of features.
ordinal_features = ['condition']
numeric_features = ['year', 'cylinders', 'odometer']
nominal_features = ['fuel', 'drive', 'type']
all_features = ordinal_features+numeric_features+nominal_features

#print(type(cars_cleaned[all_features]), type(cars_cleaned['price']))

# Cross-validation split.
X_train, X_test, y_train, y_test = train_test_split(cars_cleaned[all_features],
    cars_cleaned['price'], test_size=0.25, random_state=42, shuffle=True)

# Polynomial expansion.
poly_features = ['condition', 'cylinders', 'odometer']
col_transformer = make_column_transformer(
    (StandardScaler(), poly_features),
    (PolynomialFeatures(degree=2, include_bias=False), poly_features),
    (OneHotEncoder(drop='if_binary'), nominal_features),
    remainder='passthrough')

# Pipeline for grid search.
pipeline3 = Pipeline([
```

```python
    ('transform', col_transformer),
    ('ridge', Ridge())
])
param_grid={'ridge__alpha': [1, 10, 100]}
gridsearch2 = GridSearchCV(pipeline3, param_grid=param_grid,
                           scoring='neg_mean_squared_error')
gridsearch2.fit(X_train, y_train)
```

Out[15]:

```
▸ GridSearchCV
  ▸ transform: ColumnTransformer
  ▸ standardscaler     ▸ polynomialfeatures     ▸ onehotencoder     ▸ remainder
  ▸ StandardScaler     ▸ PolynomialFeatures     ▸ OneHotEncoder     ▸ passthrough
  ▸ Ridge
```

In [16]:

```python
# Model evaluation.
y_predict_train = gridsearch2.predict(X_train)
y_predict_test = gridsearch2.predict(X_test)
print(f'Train MSE: {mean_squared_error(y_train, y_predict_train)}')
print(f'Test MSE: {mean_squared_error(y_test, y_predict_test)}')
print(gridsearch2.best_estimator_.named_steps['ridge'].coef_)
```

```
Train MSE: 62798434.80458184
Test MSE: 64558211.87038382
[ 7.24489502e+03  4.03295620e+03  1.78067651e+04  7.32557945e+03
  6.82581872e+03  7.42742979e+03 -9.42676979e+02  2.95280056e+02
 -1.95072370e+03 -6.90279070e+00 -1.80870061e+03 -5.18507551e+03
  6.63487032e+03  8.40563934e+03 -5.61237267e+03 -3.50595276e+03
 -5.92218423e+03  1.93503931e+03 -1.93406694e+03 -9.72367171e-01
 -7.23311676e+02 -1.82322049e+03  1.81956915e+03  1.11782211e+03
 -2.79241272e+03 -1.11961146e+03  2.73618433e+03 -5.96182514e+02
  2.34804537e+03 -2.93283839e+03  4.71072110e+03  1.81126050e+03
 -4.55602533e+03  2.52708297e+02]
```

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.

In [17]:
```
gridsearch.best_estimator_.get_params()
```

Out[17]:
```
{'memory': None,
 'steps': [('transform',
   ColumnTransformer(remainder='passthrough',
                     transformers=[('standardscaler', StandardScaler(),
                                    ['condition', 'size', 'cylinders',
                                     'odometer']),
                                   ('polynomialfeatures',
                                    PolynomialFeatures(include_bias=False),
                                    ['condition', 'size', 'cylinders',
                                     'odometer']),
                                   ('onehotencoder',
                                    OneHotEncoder(drop='if_binary'),
                                    ['manufacturer', 'fuel', 'title_status',
                                     'transmission', 'drive', 'type',
                                     'paint_color', 'state'])])),
  ('ridge', Ridge(alpha=10))],
 'verbose': False,
 'transform': ColumnTransformer(remainder='passthrough',
                   transformers=[('standardscaler', StandardScaler(),
                                  ['condition', 'size', 'cylinders',
                                   'odometer']),
                                 ('polynomialfeatures',
                                  PolynomialFeatures(include_bias=False),
                                  ['condition', 'size', 'cylinders',
                                   'odometer']),
                                 ('onehotencoder',
                                  OneHotEncoder(drop='if_binary'),
                                  ['manufacturer', 'fuel', 'title_status',
                                   'transmission', 'drive', 'type',
                                   'paint_color', 'state'])]),
 'ridge': Ridge(alpha=10),
 'transform__n_jobs': None,
 'transform__remainder': 'passthrough',
 'transform__sparse_threshold': 0.3,
 'transform__transformer_weights': None,
 'transform__transformers': [('standardscaler',
   StandardScaler(),
   ['condition', 'size', 'cylinders', 'odometer']),
  ('polynomialfeatures',
   PolynomialFeatures(include_bias=False),
   ['condition', 'size', 'cylinders', 'odometer']),
```

```
  ('onehotencoder',
   OneHotEncoder(drop='if_binary'),
   ['manufacturer',
    'fuel',
    'title_status',
    'transmission',
    'drive',
    'type',
    'paint_color',
    'state'])],
 'transform__verbose': False,
 'transform__verbose_feature_names_out': True,
 'transform__standardscaler': StandardScaler(),
 'transform__polynomialfeatures': PolynomialFeatures(include_bias=False),
 'transform__onehotencoder': OneHotEncoder(drop='if_binary'),
 'transform__standardscaler__copy': True,
 'transform__standardscaler__with_mean': True,
 'transform__standardscaler__with_std': True,
 'transform__polynomialfeatures__degree': 2,
 'transform__polynomialfeatures__include_bias': False,
 'transform__polynomialfeatures__interaction_only': False,
 'transform__polynomialfeatures__order': 'C',
 'transform__onehotencoder__categories': 'auto',
 'transform__onehotencoder__drop': 'if_binary',
 'transform__onehotencoder__dtype': numpy.float64,
 'transform__onehotencoder__handle_unknown': 'error',
 'transform__onehotencoder__sparse': True,
 'ridge__alpha': 10,
 'ridge__copy_X': True,
 'ridge__fit_intercept': True,
 'ridge__max_iter': None,
 'ridge__normalize': 'deprecated',
 'ridge__positive': False,
 'ridge__random_state': None,
 'ridge__solver': 'auto',
 'ridge__tol': 0.001}
```

Out of the two models explored, gridsearch and pipeline2, we found that gridsearch model provides lower MSEs in terms of the training data and the testing data. The parameters contained in the model can be exported to the backend Python library hosted on webservers, and be presented to car dealers with a simple API to get inputs regarding the critical parameters of the used car. According to the customer inputs, the model will output an estimated sale price as a reference for the dealer.

Of course, the used car sales also depend on supply and demand of the market, which was not modeled in this exercise.