notebook

August 22, 2024

0.1 2. Machine Learning for Regression

[1]:
```
import pandas as pd
import numpy as np
```

0.2 2.2 Data preparation

[2]:
```
import requests

url = 'https://raw.githubusercontent.com/alexeygrigorev/mlbookcamp-code/master/
↪chapter-02-car-price/data.csv'
response = requests.get(url)

# Save the content to a file
with open('data.csv', 'wb') as file:
    file.write(response.content)
```

[3]:
```
df = pd.read_csv('data.csv')
```

[4]:
```
df.columns = df.columns.str.lower().str.replace(' ', '_')
```

[5]:
```
df['make'].str.lower().str.replace(' ', '_')
```

[5]:
```
0              bmw
1              bmw
2              bmw
3              bmw
4              bmw
          …
11909        acura
11910        acura
11911        acura
11912        acura
11913      lincoln
Name: make, Length: 11914, dtype: object
```

[6]:
```
strings = list(df.dtypes[df.dtypes == 'object'].index)
strings
```

[6]:
```
['make',
 'model',
 'engine_fuel_type',
 'transmission_type',
 'driven_wheels',
 'market_category',
 'vehicle_size',
 'vehicle_style']
```

[7]:
```python
for col in strings:
    df[col] = df[col].str.lower().str.replace(' ', '_')
```

[8]:
```python
df.dtypes
```

[8]:
```
make                  object
model                 object
year                   int64
engine_fuel_type      object
engine_hp            float64
engine_cylinders     float64
transmission_type     object
driven_wheels         object
number_of_doors      float64
market_category       object
vehicle_size          object
vehicle_style         object
highway_mpg            int64
city_mpg               int64
popularity             int64
msrp                   int64
dtype: object
```

0.3 2.3 Exploratory data analysis

[9]:
```python
for col in df.columns:
    print(col)
    print(df[col].unique()[:5])
    print(df[col].nunique())
    print()
```

```
make
['bmw' 'audi' 'fiat' 'mercedes-benz' 'chrysler']
48

model
['1_series_m' '1_series' '100' '124_spider' '190-class']
914
```

```
year
[2011 2012 2013 1992 1993]
28

engine_fuel_type
['premium_unleaded_(required)' 'regular_unleaded'
 'premium_unleaded_(recommended)' 'flex-fuel_(unleaded/e85)' 'diesel']
10

engine_hp
[335. 300. 230. 320. 172.]
356

engine_cylinders
[ 6.  4.  5.  8. 12.]
9

transmission_type
['manual' 'automatic' 'automated_manual' 'direct_drive' 'unknown']
5

driven_wheels
['rear_wheel_drive' 'front_wheel_drive' 'all_wheel_drive'
 'four_wheel_drive']
4

number_of_doors
[ 2.  4.  3. nan]
3

market_category
['factory_tuner,luxury,high-performance' 'luxury,performance'
 'luxury,high-performance' 'luxury' 'performance']
71

vehicle_size
['compact' 'midsize' 'large']
3

vehicle_style
['coupe' 'convertible' 'sedan' 'wagon' '4dr_hatchback']
16

highway_mpg
[26 28 27 25 24]
59

city_mpg
```

```
[19 20 18 17 16]
69

popularity
[3916 3105  819  617 1013]
48

msrp
[46135 40650 36350 29450 34500]
6049
```

[10]:
```
df
```

[10]:
```
          make       model  year             engine_fuel_type  engine_hp  \
0          bmw  1_series_m  2011  premium_unleaded_(required)      335.0
1          bmw    1_series  2011  premium_unleaded_(required)      300.0
2          bmw    1_series  2011  premium_unleaded_(required)      300.0
3          bmw    1_series  2011  premium_unleaded_(required)      230.0
4          bmw    1_series  2011  premium_unleaded_(required)      230.0
…            …         …     …                            …          …
11909    acura         zdx  2012  premium_unleaded_(required)      300.0
11910    acura         zdx  2012  premium_unleaded_(required)      300.0
11911    acura         zdx  2012  premium_unleaded_(required)      300.0
11912    acura         zdx  2013 premium_unleaded_(recommended)    300.0
11913  lincoln      zephyr  2006             regular_unleaded      221.0

       engine_cylinders transmission_type     driven_wheels  number_of_doors  \
0                   6.0            manual  rear_wheel_drive              2.0
1                   6.0            manual  rear_wheel_drive              2.0
2                   6.0            manual  rear_wheel_drive              2.0
3                   6.0            manual  rear_wheel_drive              2.0
4                   6.0            manual  rear_wheel_drive              2.0
…                     …               …                 …                …
11909               6.0         automatic   all_wheel_drive              4.0
11910               6.0         automatic   all_wheel_drive              4.0
11911               6.0         automatic   all_wheel_drive              4.0
11912               6.0         automatic   all_wheel_drive              4.0
11913               6.0         automatic front_wheel_drive              4.0

                            market_category vehicle_size vehicle_style  \
0      factory_tuner,luxury,high-performance      compact         coupe
1                         luxury,performance      compact   convertible
2                    luxury,high-performance      compact         coupe
3                         luxury,performance      compact         coupe
4                                     luxury      compact   convertible
…                                          …            …             …
```

```
11909             crossover,hatchback,luxury      midsize  4dr_hatchback
11910             crossover,hatchback,luxury      midsize  4dr_hatchback
11911             crossover,hatchback,luxury      midsize  4dr_hatchback
11912             crossover,hatchback,luxury      midsize  4dr_hatchback
11913                                 luxury      midsize          sedan

       highway_mpg  city_mpg  popularity   msrp
0               26        19        3916  46135
1               28        19        3916  40650
2               28        20        3916  36350
3               28        18        3916  29450
4               28        18        3916  34500
…             …       …         …     …
11909           23        16         204  46120
11910           23        16         204  56670
11911           23        16         204  50620
11912           23        16         204  50920
11913           26        17          61  28995

[11914 rows x 16 columns]
```

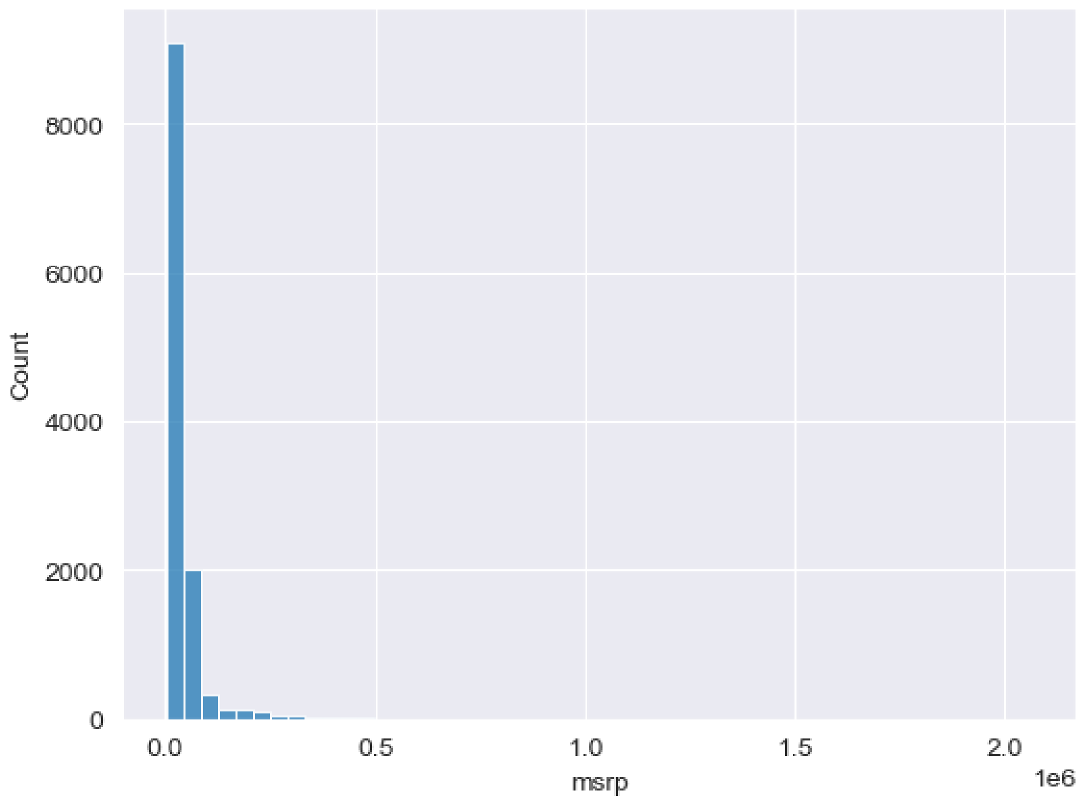
Distribution of price

[11]:
```
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

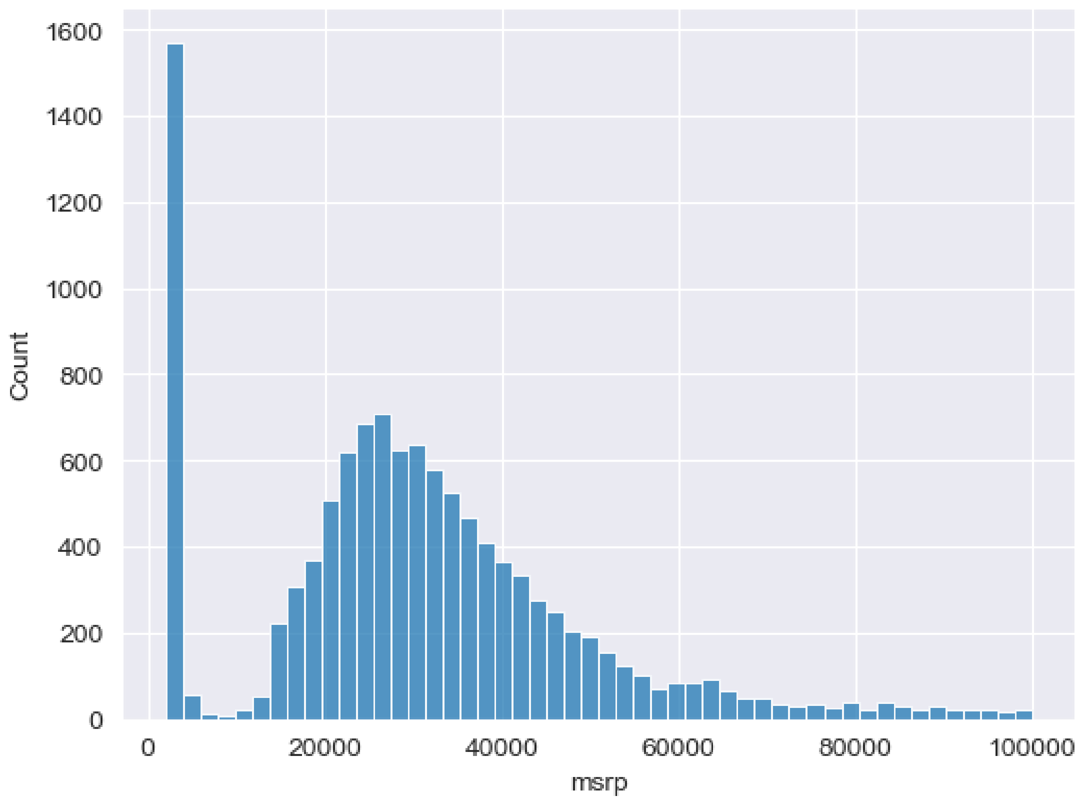
[12]:
```
sns.histplot(df.msrp, bins=50)
```

[12]:
```
<Axes: xlabel='msrp', ylabel='Count'>
```



[13]:
```
sns.histplot(df.msrp[df.msrp < 100000], bins=50)
```

[13]:
```
<Axes: xlabel='msrp', ylabel='Count'>
```




```
[14]: np.log1p([0, 1, 10, 1000, 100000])
```

```
[14]: array([ 0.        ,  0.69314718,  2.39789527,  6.90875478, 11.51293546])
```

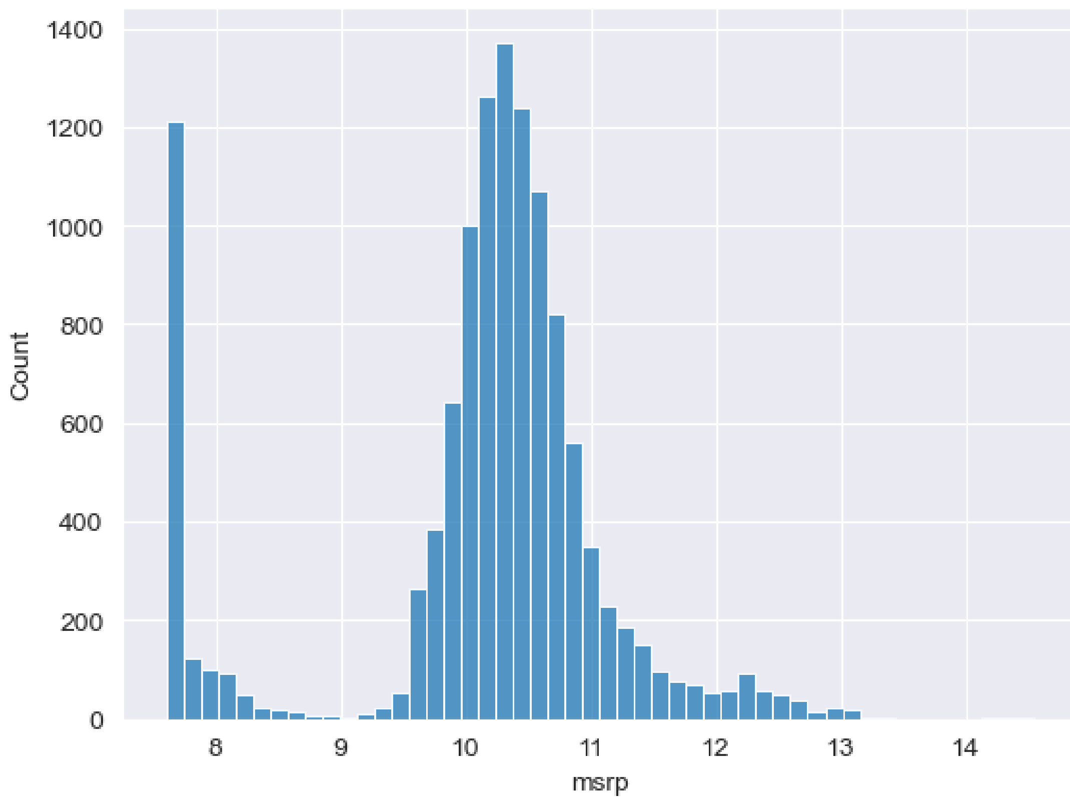
```
[15]: np.log([0 + 1, 1+ 1, 10 + 1, 1000 + 1, 100000])
```

```
[15]: array([ 0.        ,  0.69314718,  2.39789527,  6.90875478, 11.51292546])
```

```
[16]: price_logs = np.log1p(df.msrp)
```

```
[17]: sns.histplot(price_logs, bins=50)
```

```
[17]: <Axes: xlabel='msrp', ylabel='Count'>
```



Missing values

[18]:
```
df.isnull().sum()
```

[18]:
```
make                    0
model                   0
year                    0
engine_fuel_type        3
engine_hp              69
engine_cylinders       30
transmission_type       0
driven_wheels           0
number_of_doors         6
market_category      3742
vehicle_size            0
vehicle_style           0
highway_mpg             0
city_mpg                0
popularity              0
msrp                    0
dtype: int64
```

0.4 2.4 Setting up the validation framework

Let's draw it

[19]:
```
n = len(df)

n_val = int(n * 0.2)
n_test = int(n * 0.2)
n_train = n - n_val - n_test
```

[20]:
```
n
```

[20]:
```
11914
```

[21]:
```
n_val, n_test, n_train
```

[21]:
```
(2382, 2382, 7150)
```

[22]:
```
df.iloc[[10, 0, 3, 5]]
```

[22]:
```
   make       model  year             engine_fuel_type  engine_hp  \
10  bmw    1_series  2013  premium_unleaded_(required)      300.0
0   bmw  1_series_m  2011  premium_unleaded_(required)      335.0
3   bmw    1_series  2011  premium_unleaded_(required)      230.0
5   bmw    1_series  2012  premium_unleaded_(required)      230.0

    engine_cylinders transmission_type     driven_wheels  number_of_doors  \
10               6.0            manual  rear_wheel_drive              2.0
0                6.0            manual  rear_wheel_drive              2.0
3                6.0            manual  rear_wheel_drive              2.0
5                6.0            manual  rear_wheel_drive              2.0

                          market_category vehicle_size vehicle_style  \
10                 luxury,high-performance      compact         coupe
0   factory_tuner,luxury,high-performance      compact         coupe
3                       luxury,performance      compact         coupe
5                       luxury,performance      compact         coupe

    highway_mpg  city_mpg  popularity   msrp
10           28        20        3916  39600
0            26        19        3916  46135
3            28        18        3916  29450
5            28        18        3916  31200
```

[23]:
```
df_train = df.iloc[:n_train]
df_val = df.iloc[n_train:n_train+n_val]
df_test = df.iloc[n_train+n_val:]
```

[24]:
```python
idx = np.arange(n)
```

[25]:
```python
np.random.seed(2)
np.random.shuffle(idx)
```

[26]:
```python
df_train = df.iloc[idx[:n_train]]
df_val = df.iloc[idx[n_train:n_train+n_val]]
df_test = df.iloc[idx[n_train+n_val:]]
```

[27]:
```python
df_train.head()
```

[27]:
```
             make    model  year           engine_fuel_type  engine_hp  \
2735    chevrolet   cobalt  2008           regular_unleaded      148.0
6720       toyota   matrix  2012           regular_unleaded      132.0
5878       subaru  impreza  2016           regular_unleaded      148.0
11190  volkswagen  vanagon  1991           regular_unleaded       90.0
4554         ford    f-150  2017  flex-fuel_(unleaded/e85)      385.0

       engine_cylinders transmission_type     driven_wheels  number_of_doors  \
2735                4.0            manual  front_wheel_drive              2.0
6720                4.0         automatic  front_wheel_drive              4.0
5878                4.0         automatic    all_wheel_drive              4.0
11190               4.0            manual   rear_wheel_drive              3.0
4554                8.0         automatic   four_wheel_drive              4.0

      market_category vehicle_size      vehicle_style  highway_mpg  city_mpg  \
2735              NaN      compact              coupe           33        24
6720        hatchback      compact      4dr_hatchback           32        25
5878        hatchback      compact      4dr_hatchback           37        28
11190             NaN        large  passenger_minivan           18        16
4554        flex_fuel        large     crew_cab_pickup           21        15

       popularity   msrp
2735         1385  14410
6720         2031  19685
5878          640  19795
11190         873   2000
4554         5657  56260
```

[28]:
```python
len(df_train), len(df_val), len(df_test)
```

[28]:
```
(7150, 2382, 2382)
```

[29]:
```python
df_train = df_train.reset_index(drop=True)
df_val = df_val.reset_index(drop=True)
df_test = df_test.reset_index(drop=True)
```

[30]:
```
y_train = np.log1p(df_train.msrp.values)
y_val = np.log1p(df_val.msrp.values)
y_test = np.log1p(df_test.msrp.values)
```

[31]:
```
del df_train['msrp']
del df_val['msrp']
del df_test['msrp']
```

[32]:
```
len(y_train)
```

[32]: 7150

0.5 2.5 Linear regression

draw

[33]:
```
df_train.iloc[10]
```

[33]:
```
make                                     rolls-royce
model                          phantom_drophead_coupe
year                                            2015
engine_fuel_type         premium_unleaded_(required)
engine_hp                                      453.0
engine_cylinders                                12.0
transmission_type                          automatic
driven_wheels                       rear_wheel_drive
number_of_doors                                  2.0
market_category            exotic,luxury,performance
vehicle_size                                   large
vehicle_style                            convertible
highway_mpg                                       19
city_mpg                                          11
popularity                                        86
Name: 10, dtype: object
```

[34]:
```
xi = [453, 11, 86]
w0 = 7.17
w = [0.01, 0.04, 0.002]
```

[35]:
```
def linear_regression(xi):
    n = len(xi)

    pred = w0

    for j in range(n):
        pred = pred + w[j] * xi[j]

    return pred
```

[36]:
```
xi = [453, 11, 86]
w0 = 7.17
w = [0.01, 0.04, 0.002]
```

[37]:
```
linear_regression(xi)
```

[37]: 12.312

[38]:
```
np.expm1(12.312)
```

[38]: 222347.2221101062

[39]:
```
np.log1p(222347.2221101062)
```

[39]: 12.312

0.6 2.6 Linear regression vector form

[40]:
```
def dot(xi, w):
    n = len(xi)

    res = 0.0

    for j in range(n):
        res = res + xi[j] * w[j]

    return res
```

[41]:
```
def linear_regression(xi):
    return w0 + dot(xi, w)
```

[42]:
```
w_new = [w0] + w
```

[43]:
```
w_new
```

[43]: [7.17, 0.01, 0.04, 0.002]

[44]:
```
def linear_regression(xi):
    xi = [1] + xi
    return dot(xi, w_new)
```

[45]:
```
linear_regression(xi)
```

[45]: 12.312

[46]:
```
w0 = 7.17
w = [0.01, 0.04, 0.002]
w_new = [w0] + w
```

[47]:
```
x1  = [1, 148, 24, 1385]
x2  = [1, 132, 25, 2031]
x10 = [1, 453, 11, 86]

X = [x1, x2, x10]
X = np.array(X)
X
```

[47]:
```
array([[   1,  148,   24, 1385],
       [   1,  132,   25, 2031],
       [   1,  453,   11,   86]])
```

[48]:
```
def linear_regression(X):
    return X.dot(w_new)
```

[49]:
```
linear_regression(X)
```

[49]:
```
array([12.38 , 13.552, 12.312])
```

0.7 2.7 Training a linear regression model

[50]:
```
def train_linear_regression(X, y):
    pass
```

[51]:
```
X = [
    [148, 24, 1385],
    [132, 25, 2031],
    [453, 11, 86],
    [158, 24, 185],
    [172, 25, 201],
    [413, 11, 86],
    [38,  54, 185],
    [142, 25, 431],
    [453, 31, 86],
]

X = np.array(X)
X
```

[51]:
```
array([[ 148,   24, 1385],
       [ 132,   25, 2031],
       [ 453,   11,   86],
       [ 158,   24,  185],
       [ 172,   25,  201],
       [ 413,   11,   86],
       [  38,   54,  185],
       [ 142,   25,  431],
```

```
       [ 453,    31,    86]])
```

[52]:
```
ones = np.ones(X.shape[0])
ones
```

[52]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[53]:
```
X = np.column_stack([ones, X])
```

[54]:
```
y = [10000, 20000, 15000, 20050, 10000, 20000, 15000, 25000, 12000]
```

[55]:
```
XTX = X.T.dot(X)
XTX_inv = np.linalg.inv(XTX)
w_full = XTX_inv.dot(X.T).dot(y)
```

[56]:
```
w0 = w_full[0]
w = w_full[1:]
```

[57]:
```
w0, w
```

[57]:
```
(25844.754055766785, array([ -16.08906468, -199.47254894,   -1.22802883]))
```

[58]:
```
def train_linear_regression(X, y):
    ones = np.ones(X.shape[0])
    X = np.column_stack([ones, X])

    XTX = X.T.dot(X)
    XTX_inv = np.linalg.inv(XTX)
    w_full = XTX_inv.dot(X.T).dot(y)

    return w_full[0], w_full[1:]
```

[59]:
```
train_linear_regression(X, y)
```

[59]:
```
(8.127577566616782e+20,
 array([-8.12757757e+20,  2.60091371e+01,  7.02526108e+00, -4.63612378e+00]))
```

0.8 2.8 Car price baseline model

[60]:
```
df_train.columns
```

[60]:
```
Index(['make', 'model', 'year', 'engine_fuel_type', 'engine_hp',
       'engine_cylinders', 'transmission_type', 'driven_wheels',
       'number_of_doors', 'market_category', 'vehicle_size', 'vehicle_style',
       'highway_mpg', 'city_mpg', 'popularity'],
      dtype='object')
```

[61]:
```
base = ['engine_hp', 'engine_cylinders', 'highway_mpg',
        'city_mpg', 'popularity']

X_train = df_train[base].fillna(0).values

w0, w = train_linear_regression(X_train, y_train)

y_pred = w0 + X_train.dot(w)
```

[62]:
```
w0
```

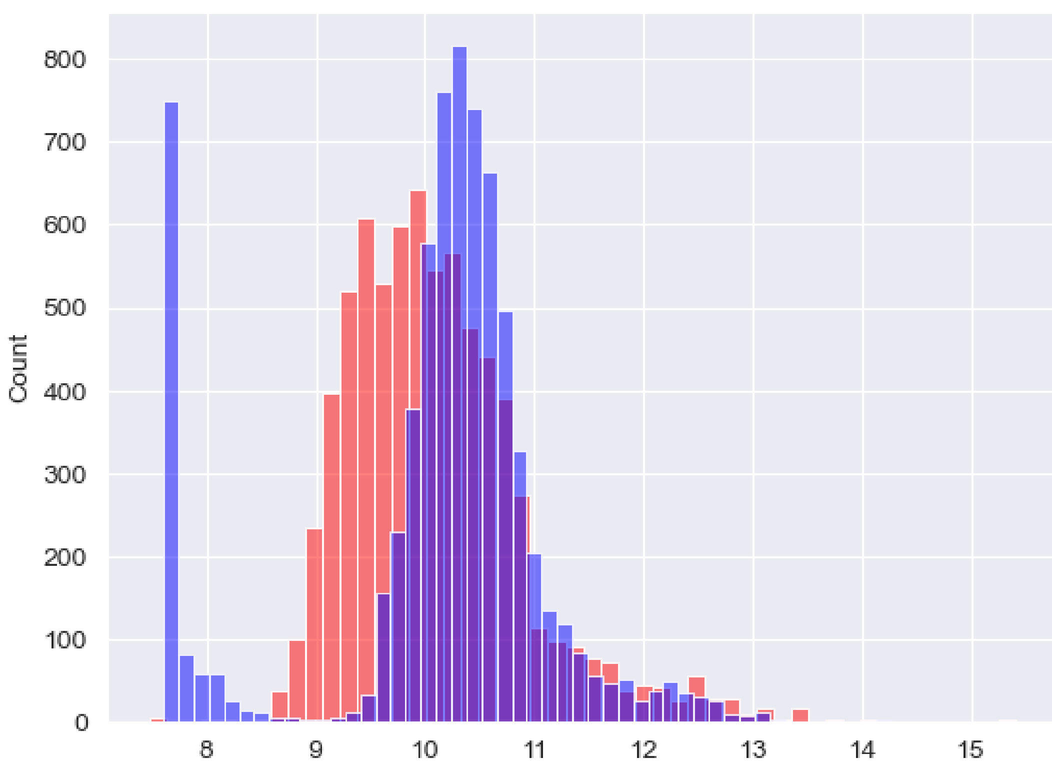
[62]: 7.927257388070112

[63]:
```
w
```

[63]:
```
array([ 9.70589522e-03, -1.59103494e-01,  1.43792133e-02,  1.49441072e-02,
       -9.06908672e-06])
```

[64]:
```
sns.histplot(y_pred, color='red', alpha=0.5, bins=50)
sns.histplot(y_train, color='blue', alpha=0.5, bins=50)
```

[64]: <Axes: ylabel='Count'>



0.9 2.9 RMSE

[65]:
```
def rmse(y, y_pred):
    se = (y - y_pred) ** 2
    mse = se.mean()
    return np.sqrt(mse)
```

[66]:
```
rmse(y_train, y_pred)
```

[66]: 0.7554192603920132

0.10 2.10 Validating the model

[67]:
```
def prepare_X(df):
    df_num = df[base]
    df_num = df_num.fillna(0)
    X = df_num.values
    return X
```

[68]:
```
X_train = prepare_X(df_train)
w0, w = train_linear_regression(X_train, y_train)

X_val = prepare_X(df_val)
y_pred = w0 + X_val.dot(w)
rmse(y_val, y_pred)
```

[68]: 0.7616530991301608

0.11 2.11 Simple feature engineering

[69]:
```
def prepare_X(df):
    df = df.copy()

    df['age'] = 2017 - df['year']
    features = base + ['age']

    df_num = df[features]
    df_num = df_num.fillna(0)
    X = df_num.values

    return X
```

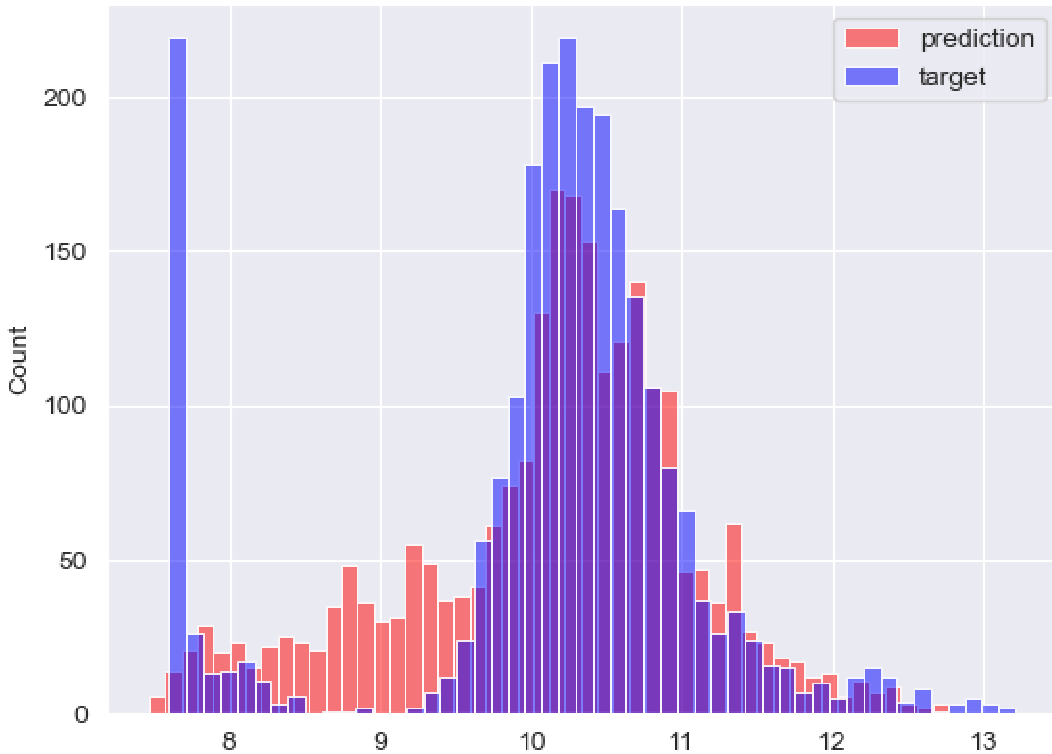
[70]:
```
X_train = prepare_X(df_train)
w0, w = train_linear_regression(X_train, y_train)

X_val = prepare_X(df_val)
y_pred = w0 + X_val.dot(w)
rmse(y_val, y_pred)
```

[70]: 0.5172055461058325

[71]:
```
sns.histplot(y_pred, label='prediction', color='red', alpha=0.5, bins=50)
sns.histplot(y_val, label='target', color='blue',  alpha=0.5, bins=50)
plt.legend()
```

[71]: <matplotlib.legend.Legend at 0x1413e4890>



0.12 2.12 Categorical variables

[72]:
```
categorical_columns = [
    'make', 'model', 'engine_fuel_type', 'driven_wheels', 'market_category',
    'vehicle_size', 'vehicle_style']

categorical = {}

for c in categorical_columns:
    categorical[c] = list(df_train[c].value_counts().head().index)
```

[73]:
```
def prepare_X(df):
    df = df.copy()
```

```
    df['age'] = 2017 - df['year']
    features = base + ['age']

    for v in [2, 3, 4]:
        df['num_doors_%d' % v] = (df.number_of_doors == v).astype(int)
        features.append('num_doors_%d' % v)

    for name, values in categorical.items():
        for value in values:
            df['%s_%s' % (name, value)] = (df[name] == value).astype(int)
            features.append('%s_%s' % (name, value))

    df_num = df[features]
    df_num = df_num.fillna(0)
    X = df_num.values

    return X
```

[74]:
```
X_train = prepare_X(df_train)
w0, w = train_linear_regression(X_train, y_train)

X_val = prepare_X(df_val)
y_pred = w0 + X_val.dot(w)
rmse(y_val, y_pred)
```

[74]:
```
1186.874474023634
```

[75]:
```
w0, w
```

[75]:
```
(2.2188634096833324e+16,
 array([-6.81651096e+00, -1.90869095e+01, -7.89425107e+01, -1.07770343e+02,
         2.38930533e-02, -6.94050130e+01, -1.14785375e+05, -1.15494870e+05,
        -1.14639299e+05,  8.54320819e+01, -1.67639211e+02, -6.76150963e+01,
         6.93313567e+02, -3.55053392e+01,  1.32882227e+02, -8.85080058e+02,
         2.25736344e+02,  1.79815398e+02, -2.86732774e+03, -4.83465025e+03,
        -4.34862174e+03, -4.64476730e+03, -4.98520840e+03, -4.63356569e+03,
        -2.21886341e+16, -2.21886341e+16, -2.21886341e+16, -2.21886341e+16,
        -1.35401806e+01, -1.07956356e+01,  6.00168522e+01,  2.59147889e+02,
         6.16557670e+01, -1.12572838e+02, -2.55658781e+02, -2.56718835e+02,
        -1.44115660e-01, -2.62579827e-02,  1.75913981e-01,  3.65037816e-01,
        -2.90235596e-01]))
```

0.13 2.13 Regularization

[76]:
```
X = [
    [4, 4, 4],
    [3, 5, 5],
```

```
    [5, 1, 1],
    [5, 4, 4],
    [7, 5, 5],
    [4, 5, 5.00000001],
]

X = np.array(X)
X
```

[76]:
```
array([[4.        , 4.        , 4.        ],
       [3.        , 5.        , 5.        ],
       [5.        , 1.        , 1.        ],
       [5.        , 4.        , 4.        ],
       [7.        , 5.        , 5.        ],
       [4.        , 5.        , 5.00000001]])
```

[77]:
```
y= [1, 2, 3, 1, 2, 3]
```

[78]:
```
XTX = X.T.dot(X)
XTX
```

[78]:
```
array([[140.        , 111.        , 111.00000004],
       [111.        , 108.        , 108.00000005],
       [111.00000004, 108.00000005, 108.0000001 ]])
```

[79]:
```
XTX_inv = np.linalg.inv(XTX)
```

[80]:
```
XTX_inv
```

[80]:
```
array([[ 3.85321698e-02,  1.20696657e+05, -1.20696686e+05],
       [ 1.20696640e+05, -2.74658839e+14,  2.74658839e+14],
       [-1.20696680e+05,  2.74658839e+14, -2.74658839e+14]])
```

[81]:
```
XTX_inv.dot(X.T).dot(y)
```

[81]:
```
array([ 8.39894892e-01,  3.44329390e+06, -3.44329299e+06])
```

[82]:
```
XTX = [
    [1, 2, 2],
    [2, 1, 1.0000001],
    [2, 1.0000001, 1]
]

XTX = np.array(XTX)
```

[83]:
```
np.linalg.inv(XTX)
```

[83]:

```
array([[-3.33333356e-01,  3.33333339e-01,  3.33333339e-01],
       [ 3.33333339e-01, -5.00000008e+06,  4.99999991e+06],
       [ 3.33333339e-01,  4.99999991e+06, -5.00000008e+06]])
```

[84]:

```
XTX = XTX + 0.01 * np.eye(3)
```

[85]:

```
np.linalg.inv(XTX)
```

[85]:

```
array([[ -0.33668908,   0.33501399,   0.33501399],
       [  0.33501399,  49.91590897, -50.08509104],
       [  0.33501399, -50.08509104,  49.91590897]])
```

[86]:

```
def train_linear_regression_reg(X, y, r=0.001):
    ones = np.ones(X.shape[0])
    X = np.column_stack([ones, X])

    XTX = X.T.dot(X)
    XTX = XTX + r * np.eye(XTX.shape[0])

    XTX_inv = np.linalg.inv(XTX)
    w_full = XTX_inv.dot(X.T).dot(y)

    return w_full[0], w_full[1:]
```

[87]:

```
X_train = prepare_X(df_train)
w0, w = train_linear_regression_reg(X_train, y_train, r=0.01)

X_val = prepare_X(df_val)
y_pred = w0 + X_val.dot(w)
rmse(y_val, y_pred)
```

[87]:

```
0.4608208286204368
```

0.14 2.14 Tuning the model

[88]:

```
for r in [0.0, 0.00001, 0.0001, 0.001, 0.1, 1, 10]:
    X_train = prepare_X(df_train)
    w0, w = train_linear_regression_reg(X_train, y_train, r=r)

    X_val = prepare_X(df_val)
    y_pred = w0 + X_val.dot(w)
    score = rmse(y_val, y_pred)

    print(r, w0, score)
```

```
0.0 2.2188634096833324e+16 1186.874474023634
1e-05 6.398609911018562 0.4608153059057127
0.0001 7.123460125218161 0.4608153639451674
```

```
0.001 7.130893282891831 0.4608158584430818
0.1 7.000232411796987 0.4608736549113679
1 6.2507478473706115 0.46158128382725866
10 4.72951258567708 0.4726098772668483
```

[89]:
```python
r = 0.001
X_train = prepare_X(df_train)
w0, w = train_linear_regression_reg(X_train, y_train, r=r)

X_val = prepare_X(df_val)
y_pred = w0 + X_val.dot(w)
score = rmse(y_val, y_pred)
score
```

[89]:
```
0.4608158584430818
```

0.15 2.15 Using the model

[90]:
```python
df_full_train = pd.concat([df_train, df_val])
```

[91]:
```python
df_full_train = df_full_train.reset_index(drop=True)
```

[92]:
```python
X_full_train = prepare_X(df_full_train)
```

[93]:
```python
X_full_train
```

[93]:
```
array([[148.,   4.,  33., …,   1.,   0.,   0.],
       [132.,   4.,  32., …,   0.,   0.,   1.],
       [148.,   4.,  37., …,   0.,   0.,   1.],
       …,
       [332.,   8.,  23., …,   0.,   0.,   0.],
       [148.,   4.,  34., …,   0.,   0.,   0.],
       [290.,   6.,  25., …,   0.,   0.,   0.]])
```

[94]:
```python
y_full_train = np.concatenate([y_train, y_val])
```

[95]:
```python
w0, w = train_linear_regression_reg(X_full_train, y_full_train, r=0.001)
```

[96]:
```python
X_test = prepare_X(df_test)
y_pred = w0 + X_test.dot(w)
score = rmse(y_test, y_pred)
score
```

[96]:
```
0.4600753970560443
```

[97]:
```python
car = df_test.iloc[20].to_dict()
car
```

[97]:
```
{'make': 'toyota',
 'model': 'sienna',
 'year': 2015,
 'engine_fuel_type': 'regular_unleaded',
 'engine_hp': 266.0,
 'engine_cylinders': 6.0,
 'transmission_type': 'automatic',
 'driven_wheels': 'front_wheel_drive',
 'number_of_doors': 4.0,
 'market_category': nan,
 'vehicle_size': 'large',
 'vehicle_style': 'passenger_minivan',
 'highway_mpg': 25,
 'city_mpg': 18,
 'popularity': 2031}
```

[98]:
```
df_small = pd.DataFrame([car])
df_small
```

[98]:
```
     make   model  year  engine_fuel_type  engine_hp  engine_cylinders  \
0  toyota  sienna  2015  regular_unleaded      266.0               6.0

  transmission_type      driven_wheels  number_of_doors  market_category  \
0         automatic  front_wheel_drive              4.0              NaN

  vehicle_size      vehicle_style  highway_mpg  city_mpg  popularity
0        large  passenger_minivan           25        18        2031
```

[99]:
```
X_small = prepare_X(df_small)
```

[100]:
```
y_pred = w0 + X_small.dot(w)
y_pred = y_pred[0]
y_pred
```

[100]:
```
10.632492501010727
```

[101]:
```
np.expm1(y_pred)
```

[101]:
```
41459.33645013401
```

[102]:
```
np.expm1(y_test[20])
```

[102]:
```
35000.00000000001
```

0.16 2.16 Next steps

- We included only 5 top features. What happens if we include 10?

Other projects

- Predict the price of a house - e.g. boston dataset
- https://archive.ics.uci.edu/ml/datasets.php?task=reg
- https://archive.ics.uci.edu/ml/datasets/Student+Performance

0.17 2.17 Summary

- EDA - looking at data, finding missing values
- Target variable distribution - long tail => bell shaped curve
- Validation framework: train/val/test split (helped us detect problems)
- Normal equation - not magic, but math
- Implemented it with numpy
- RMSE to validate our model
- Feature engineering: age, categorical features
- Regularization to fight numerical instability

[ ]:

```

```