w3-ML-regressions

March 10, 2021

1 Big Data for Public Policy

1.1 Machine Learning - Regressions

1.2 Malka Guillot

1.3 ETH Zürich | 860-0033-00L

Regression belongs like classification to the field of **supervised learning**.

Regression predicts numerical values in contrast to classification which predicts categories.

Other differences are:

Accuracy in measured differently

Other algorithms

[1]:
```
# Common imports
import numpy as np
import os
import pandas as pd

# To plot pretty figures
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import matplotlib as mpl
import matplotlib.pyplot as plt
#%matplotlib notebook
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

import seaborn as sns
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=DeprecationWarning)
warnings.filterwarnings = lambda *a, **kw: None
```

```python
# to make this notebook's output identical at every run
np.random.seed(42)
```

1.4 General ML Procedure

0. Look at the data
1. Select a ML method (eg. LASSO)
2. Draw randomly a hold-out sample from the data
3. Estimate the ML model using different hyperparameters
4. Select the optimal hyperparameters
5. Predict $\hat{Y}$ using hyperparameters and extrapolated the fitted values to the retarded hold-out-sample
6. Evaluation the prediction power of the ML in the hold-out-sample

1.5 Scikit-Learn Design Overview

1.5.1 Estimator: an object that can estimate parameters

- e.g. `linear_models.LinearRegression`
- Estimation performed by `fit()` method
- Exogenous parameters (provided by the researcher) are called `hyperparameters`

1.5.2 Transformer (preprocessor): An object that transforms a data set.

- e.g. `preprocessing.StandardScaler`
- Transformation is performed by the `transform()` method.
- The convenience method `fit_transform()` both fits an estimator and returns the transformed input data set.

1.5.3 Predictor: An object that forms a prediction from an input data set.

- e.g. `LinearRegression`, after training
- The `predict()` method forms the predictions.
- It also has a `score()` method that measures the quality of the predictions given a test set.

1.5.4 Miscellaneous

- **Inspection**: Hyperparameters and parameters are accessible. Learned parameters have an underscore suffix (e.g.`lin_reg.coef_`)
- **Non-proliferation of classes**: Use native Python data types; existing building blocks are used as much as possible.
- **Sensible defaults**: Provides reasonable default values for hyperparameters – easy to get a good baseline up and running

1.6 Set up and load data

1.7 Boston housing data

[2]:
```python
# Scikit-Learn  0.20 is required
import sklearn
```

We use as an example the **Boston housing data** (from `sklearn`), which contains 13 attributes of housing markets around Boston. The data was collected in 1978 and each of the 506 entries represent aggregated data about 14 features for homes from various suburbs in Boston, Massachusetts.

The objective is to **predict the value of prices** of the house using the given features

[3]:
```python
from sklearn.datasets import load_boston
data = load_boston() # object is a dictionary
data.keys()
```

[3]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

Data Set Characteristics:

[4]:
```python
print(data['DESCR'])
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
```

```
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by
town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
…', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that
address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

1.7.1 Create X and y

[5]:
```python
X_full, y_full = data.data, data.target
n_samples = X_full.shape[0]
n_features = X_full.shape[1]
```

[6]:
```python
X_df=pd.DataFrame(X_full, columns=data['feature_names']) # to dataframe format
X_df.head()
```

[6]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
```

```
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

1.7.2 Look for null values in the dataset

[7]:
```
X_df.isnull().sum()
```

[7]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
dtype: int64
```

There is none

1.8 Exploratory Data Analysis

1.8.1 Quantity to predict= price (target or y)

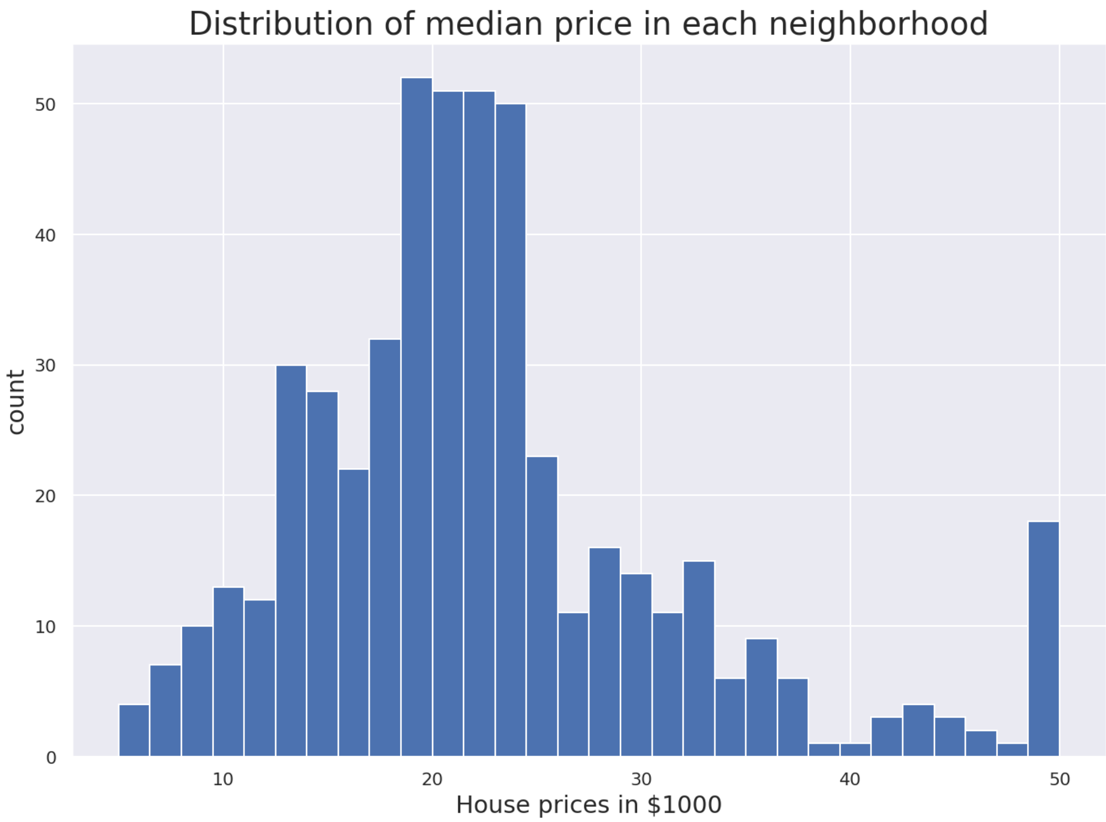
Before the regression, let us inspect the features and their distributions.

[8]:
```
y_full.shape
```

[8]:
```
(506,)
```

[9]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
plt.hist(y_full, bins=30)
plt.xlabel("House prices in $1000", size=15)
```

```
plt.ylabel('count', size=15)
plt.title('Distribution of median price in each neighborhood', size=20)
plt.show()
```



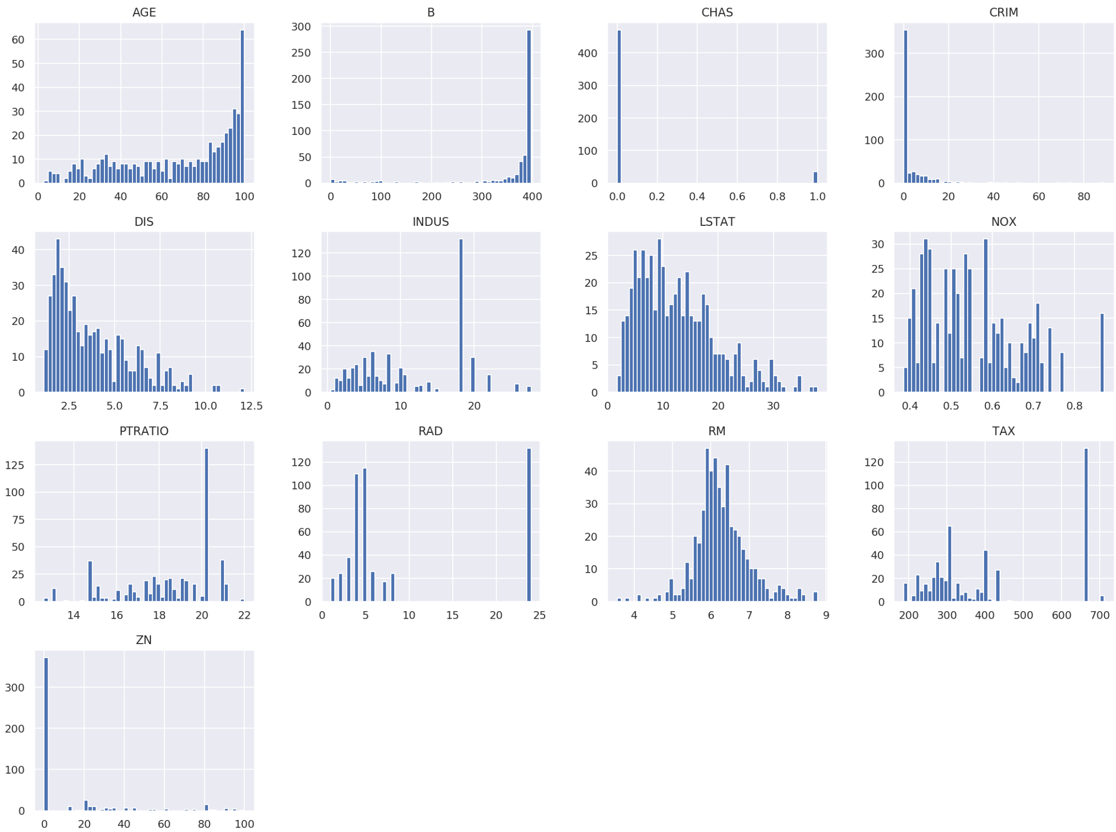
1.8.2 Features (X) used for prediction

[10]:
```
X_full.shape
```

[10]: (506, 13)

Distributions **Histogram plots** to look at the distribution

[11]:
```
X_df.hist(bins=50, figsize=(20,15))
plt.show()
```

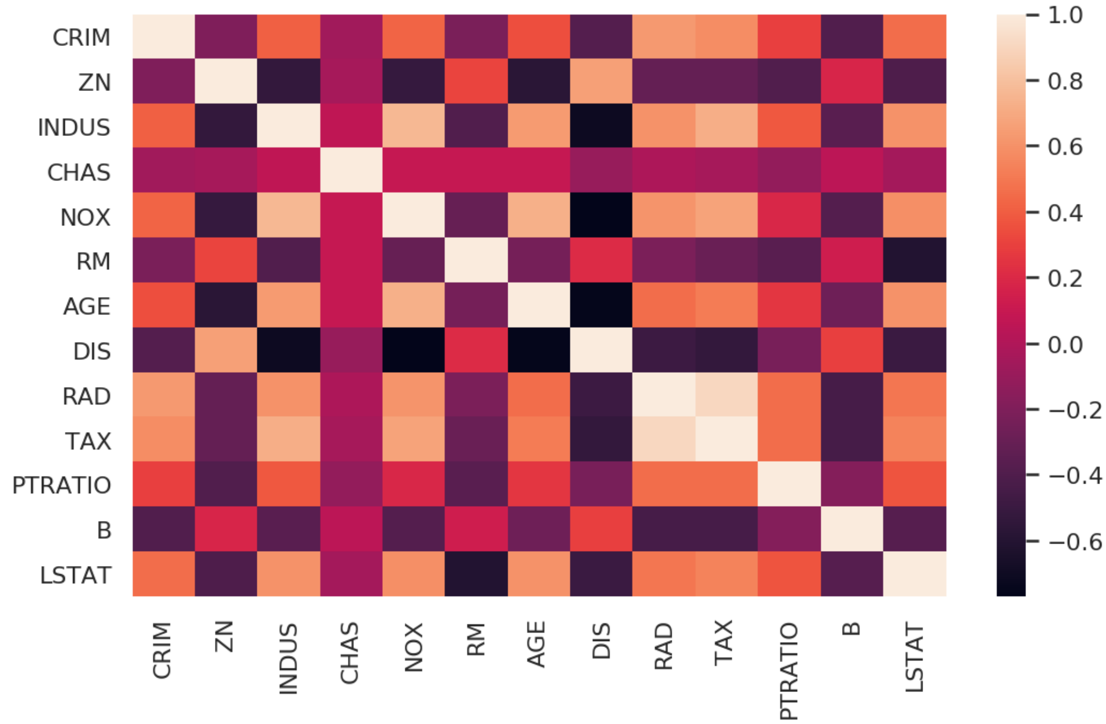


1.8.3 Correlations

Boston Correlation Heatmap Example with Seaborn

The seaborn package offers a heatmap that will allow a two-dimensional graphical representation of the Boston data. The heatmap will represent the individual values that are contained in a matrix are represented as colors.

[12]:
```
import pandas as pd
import matplotlib.pyplot as plt
sns.set(rc={'figure.figsize':(8.5,5)})
correlation_matrix = X_df.corr().round(2)
sns.heatmap(correlation_matrix) #annot=True
plt.show()
```



1.8.4 Check for multicolinearity

An important point in selecting features for a linear regression model is to check for **multicolinearity**.

The features RAD, TAX have a correlation of 0.91. These feature pairs are strongly correlated to each other. This can affect the model.

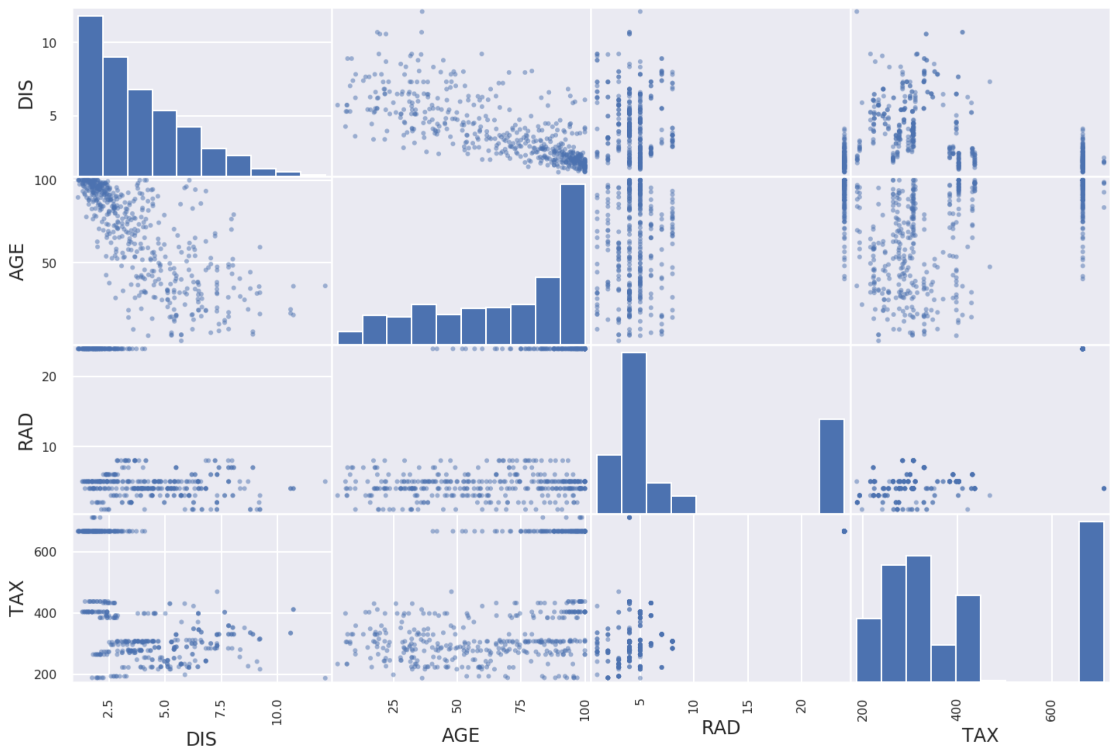
Same goes for the features DIS and AGE which have a correlation of -0.75.

1.8.5 Correlation plots

[13]:
```
from pandas.plotting import scatter_matrix
scatter_matrix(X_df[['DIS', 'AGE','RAD', 'TAX']], figsize=(12, 8))
```

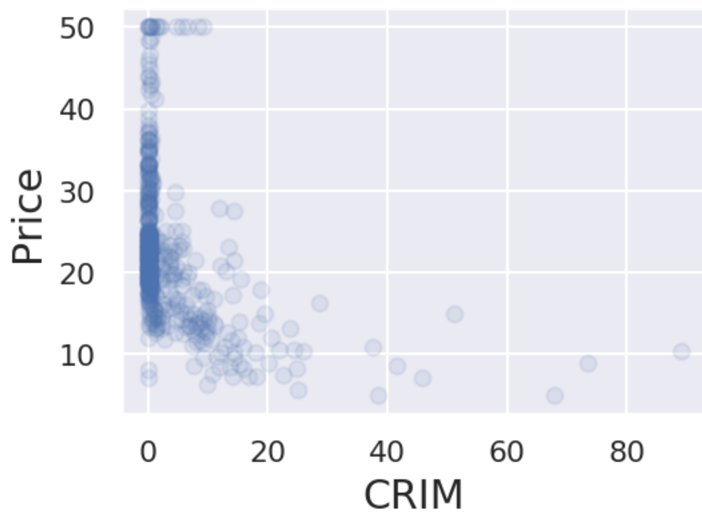
[13]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f428f905940>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f8690f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f896940>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f850320>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f428f7ffcc0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f7b86a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f773080>,
```

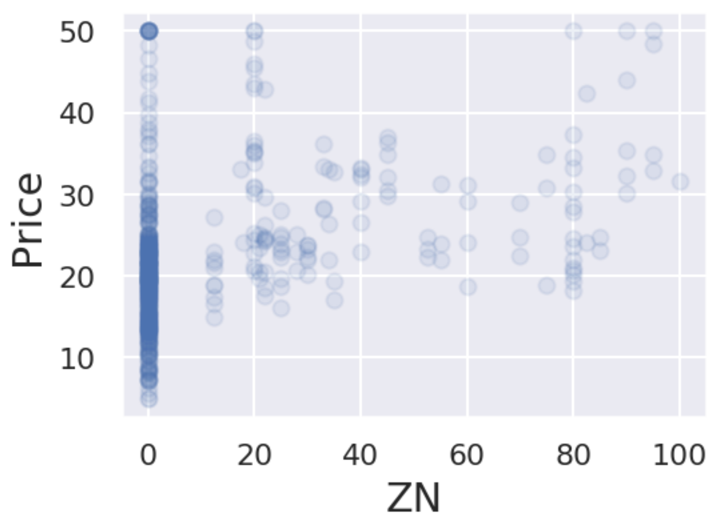
```
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f721a58>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f428f721a90>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f70bda0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f6c5780>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f67f160>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f428f62db00>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f5e64e0>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f617e80>,
 <matplotlib.axes._subplots.AxesSubplot object at 0x7f428f5ce860>]],
dtype=object)
```

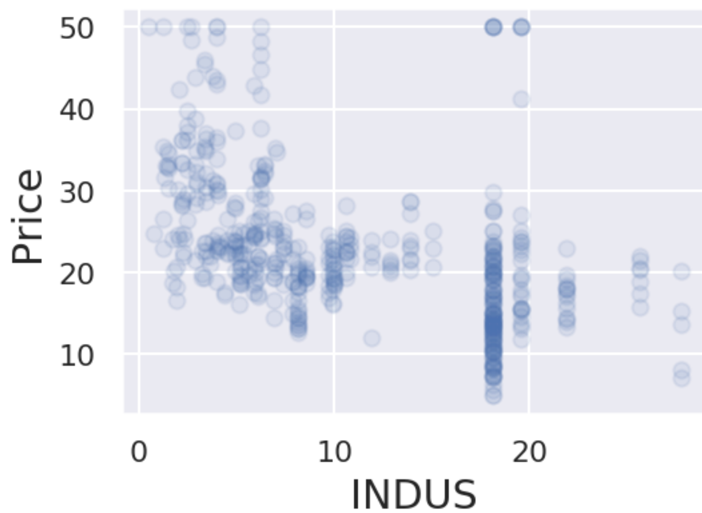


1.8.6 Scatter plot relative to the target (price)

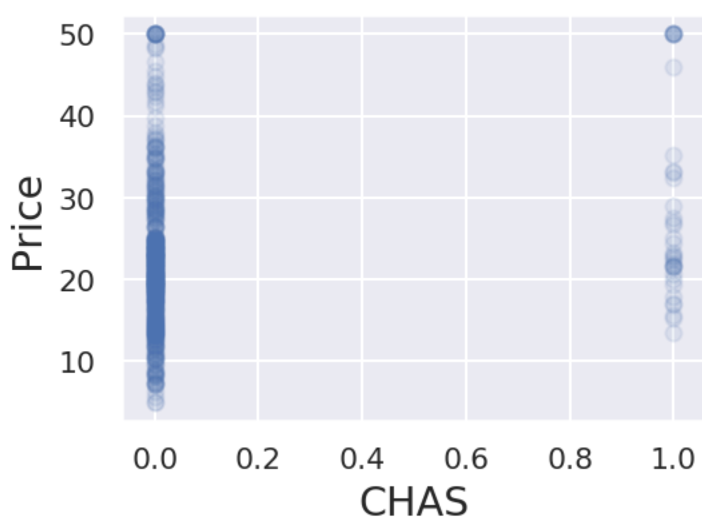
[14]:
```
for feature_name in X_df.columns:
    plt.figure(figsize=(4, 3))
    plt.scatter(X_df[feature_name], y_full, alpha=0.1)
    plt.ylabel('Price', size=15)
    plt.xlabel(feature_name, size=15)
    plt.tight_layout()
```


Price
CRIM
50
40
30
20
10
0
20
40
60
80

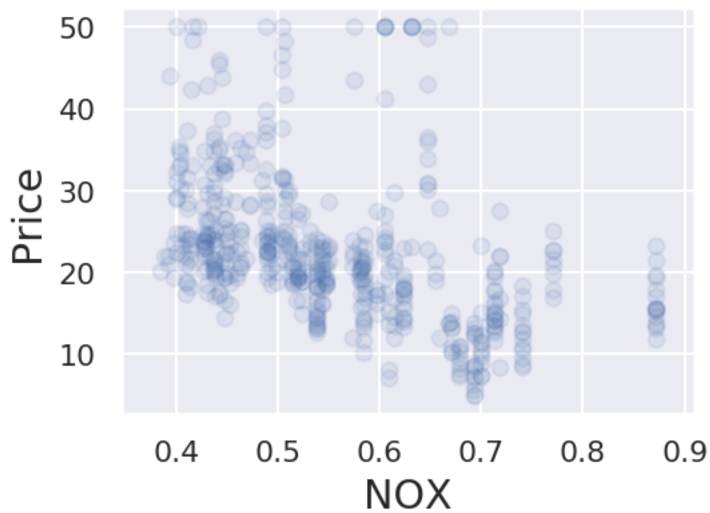

Price
ZN
50
40
30
20
10
0
20
40
60
80
100

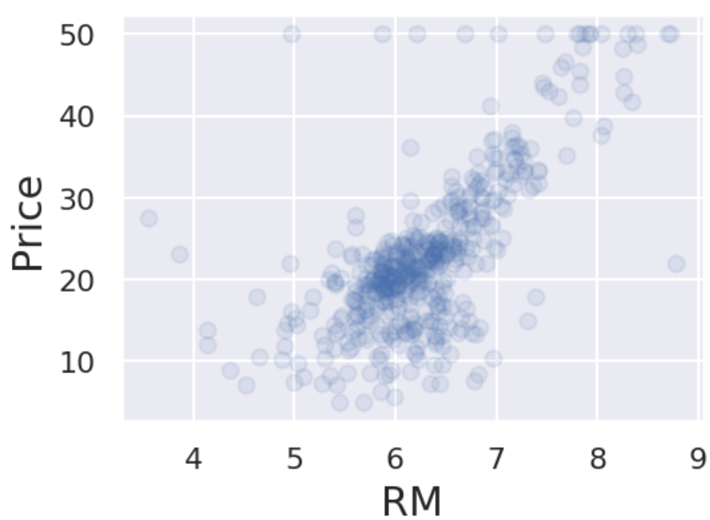


Price
50
40
30
20
10
0
10
20
INDUS

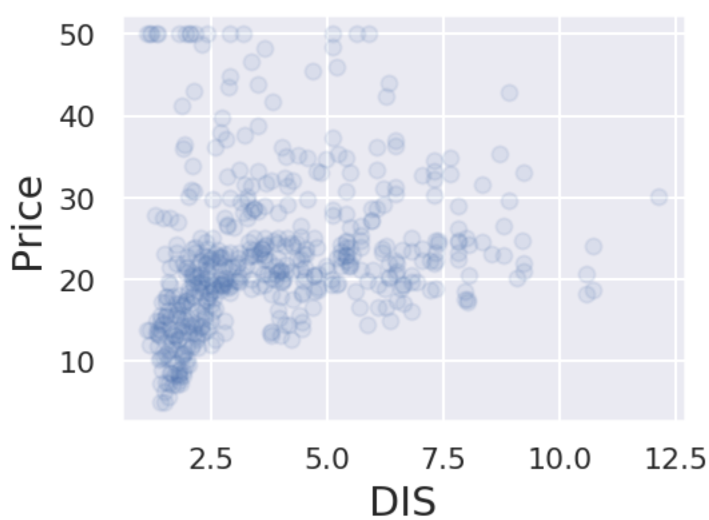


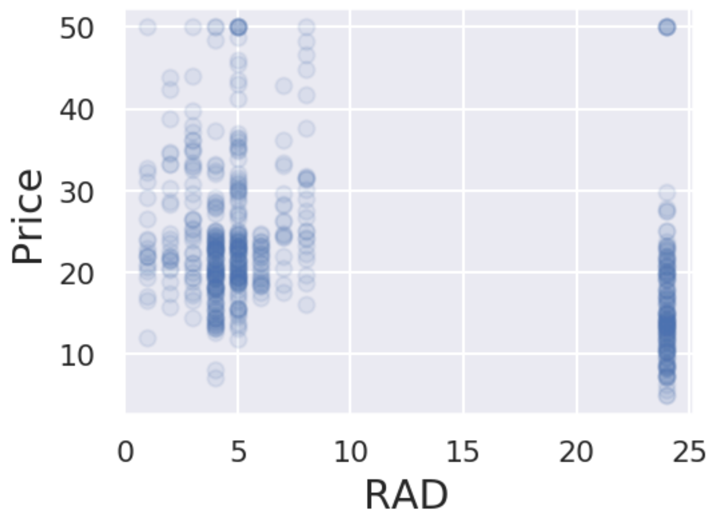
Price
50
40
30
20
10
0.0
0.2
0.4
0.6
0.8
1.0
CHAS


50
40
30
20
10
Price
0.4
0.5
0.6
0.7
0.8
0.9
NOX

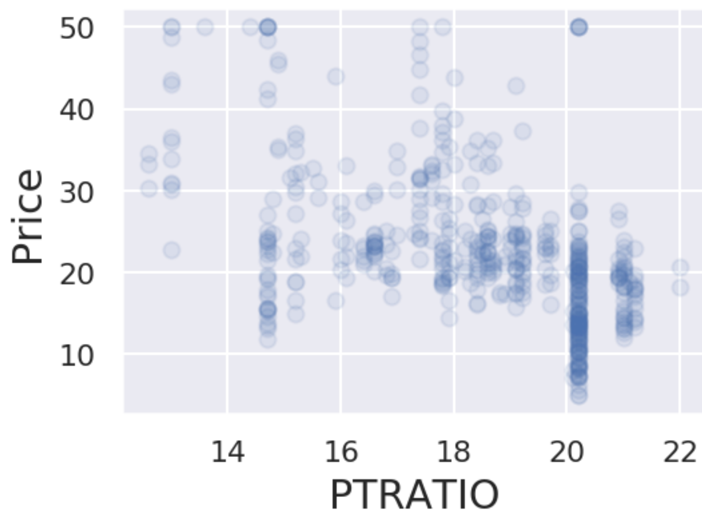

50
40
30
20
10
Price
4
5
6
7
8
9
RM

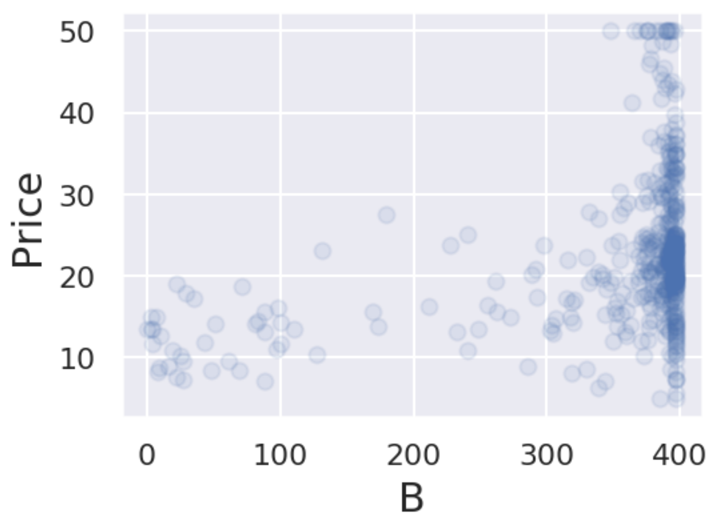

Price
AGE
0
20
40
60
80
100
10
20
30
40
50

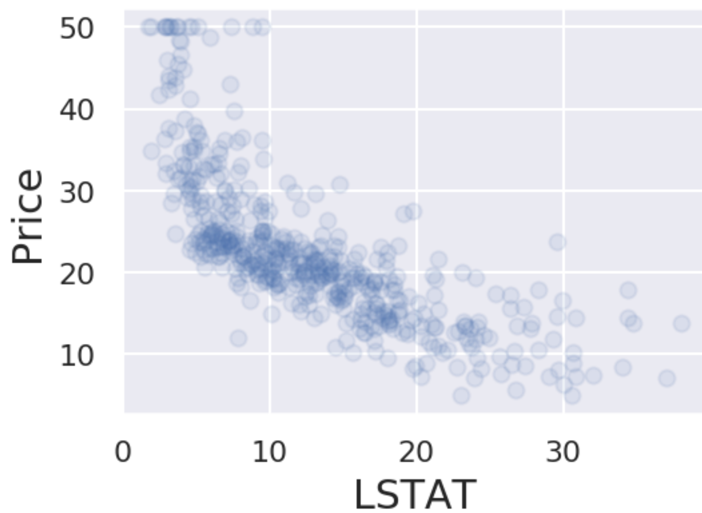

Price
DIS
2.5
5.0
7.5
10.0
12.5
10
20
30
40
50


Price
RAD
50
40
30
20
10
0
5
10
15
20
25


Price
TAX
50
40
30
20
10
200
300
400
500
600
700


Price
PTRATIO
50
40
30
20
10
14
16
18
20
22


Price
B
50
40
30
20
10
0
100
200
300
400



1.8.7 What can we say ?

- The prices increase as the value of RM increases linearly. There are few outliers and the data seems to be capped at 50.
- The prices tend to decrease with an increase in LSTAT. Though it doesn't look to be following exactly a linear line.

1.9 Prepare the data for ML algorithms

1.9.1 Drop some labeled observations:

Drop the observations with price >=50 (because of the right censure)

[15]:
```
mask=y_full<50

y_full=y_full[mask==True]
X_full=X_full[mask==True]
X_df=X_df[mask==True]
```

1.10 Split train test sets

1.10.1 Using `train_test_split`

Pure ramdomness of the sampling method

[16]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_full, y_full,test_size=0.
 ↪2, random_state=1)
print("train data", X_train.shape, y_train.shape)
print("test data", X_test.shape,  y_test.shape)
```

```
train data (392, 13) (392,)
test data (98, 13) (98,)
```

1.10.2 Data cleaning

The missing features should be: 1. dropped 2. imputed to some value (zero, the mean, the median…)

1.10.3 Feature Scaling

Most common scaling methods: - **standardization**= normalization by substracting the mean and dividing by the standard deviation (values are not bounded) - **Min-max scaling**= normalization by substracting the minimum and dividing by the maximum (values between `0` and `1`)

[17]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

1.11 Select and Train a Model

Regression algorithm (we consider firs the `LinearRegression`, more algorithms will be discussed later): ### First algorithm: Simple Linear Regression

[18]:
```
# our first machine learning model
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
```

`<i class="fa fa-warning"></i> <strong><code>scikit-learn</code> API</strong>`

In scikit-learn all regression algorithms have:

a fit() method to learn from data, and

and a subsequent predict() method for predicting numbers from input features.

[19]:
```
lin_reg.fit(X_train, y_train)
print("R-squared for training dataset:{}".
      format(np.round(lin_reg.score(X_train, y_train), 2)))
```

```
R-squared for training dataset:0.79
```

[20]:
```
lin_reg.fit(X_train_scaled, y_train)
print("R-squared for training dataset & scaled features:{}".
      format(np.round(lin_reg.score(X_train_scaled, y_train), 2)))
```

```
R-squared for training dataset & scaled features:0.79
```

1.11.1 Coefficients of the linear regression

[21]:
```
features = list(X_df.columns)

print('The coefficients of the features from the linear model:')
print(dict(zip(features, [round(x, 2) for x in lin_reg.coef_])))
```

```
The coefficients of the features from the linear model:
{'CRIM': -0.82, 'ZN': 0.96, 'INDUS': -0.54, 'CHAS': 0.19, 'NOX': -1.5, 'RM':
2.16, 'AGE': -0.55, 'DIS': -2.79, 'RAD': 2.17, 'TAX': -2.22, 'PTRATIO': -1.86,
'B': 0.72, 'LSTAT': -2.7}
```

1.12 Metrics / error measures

`scikit-learn` offers the following metrics for measuring regression quality:

1.12.1 Mean absolute error

Taking absolute values before adding up the deviatons assures that deviations with different signs can not cancel out.

`<i class="fa fa-info-circle"></i>  <strong>mean absolute error</strong> is defined as`

$$\frac{1}{n}\left(\,|y_1 - \hat{y}_1| \,+\, |y_2 - \hat{y}_2| \,+\, \ldots \,+\, |y_n - \hat{y}_n|\,\right)$$

`neg_mean_absolute_error` in `scikit-learn`.

1.12.2 Mean squared error

Here we replace the absolute difference by its squared difference. Squaring also insures positive differeces.

`<i class="fa fa-info-circle"></i>  <strong>mean squared error</strong> is defined as`

$$\frac{1}{n}\left(\,(y_1 - \hat{y}_1)^2 \,+\, (y_2 - \hat{y}_2)^2 \,\ldots \,+\, (y_n - \hat{y}_n)^2\,\right)$$

This measure is more sensitive to **outliers**: A few larger differences contribute more significantly to a larger mean squared error.

`neg_mean_squared_error` in `scikit-learn`.

1.12.3 Median absolute error

Here we replace mean calculation by median.

`<i class="fa fa-info-circle"></i>  <strong>median absolute error</strong> is defined as`

$$\text{median}\left(\,|y_1 - \hat{y}_1|,\ |y_2 - \hat{y}_2|,\ \ldots,\ |y_n - \hat{y}_n|\,\right)$$

This measure is less sensitive to outliers: A few larger differences will not contribute significantly to a larger error value.

`neg_median_absolute_error` in `scikit-learn`.

1.12.4 Mean squared log error

The formula for this metric can be found here.

This metric is recommended **when your target values are distributed over a huge range of values**, like popoluation numbers.

The previous error metrics would put a larger weight on large target values.

The name is `neg_mean_squared_log_error`

1.12.5 In-sample performance with MSE

[22]:
```
from sklearn.metrics import mean_squared_error

y_train_pred = lin_reg.predict(X_train_scaled)
train_mse = mean_squared_error(y_train, y_train_pred)
train_rmse = np.sqrt(train_mse)
print("RMSE: %s" % train_rmse) # = np.sqrt(np.mean((predicted - expected) **
→2))
```

```
RMSE: 3.6792311563556526
```

Your turn

Compute:

    the out-of-sample mean squarred error

     mean absolute error (using $mean\ absolute\ error$ from $sklearn.metrics$: you need

[23]:
```python
#1. Out-of-sample performance
y_test_pred = lin_reg.predict(X_test_scaled)
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)
print("RMS: %s" % test_rmse)
```

```
RMS: 3.8824350594341417
```

[24]:
```python
# 2. mean  absolute error
from sklearn.metrics import mean_absolute_error

lin_mae = mean_absolute_error(y_test, y_test_pred)
lin_rmae = np.sqrt(lin_mae)
print("RMAE: %s" % lin_rmae)
```

```
RMAE: 1.6979249376874554
```

1.12.6 Explained variance and R^2-score

Two other scores to mention are *explained variance* and R^2-score. For both larger values indicate better regression results.

The R^2-score corresponds to **the proportion of variance (of y) that has been explained by the independent variables in the model**. It takes values in the range 0..1. The name within `scikit-learn` is R2.

`<i class="fa fa-info-circle"></i>  <strong>$R^2$</strong> is defined as`

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

The formula for explained variance, the score takes values up to 1. The name within `scikit-learn` is `explained_variance`.

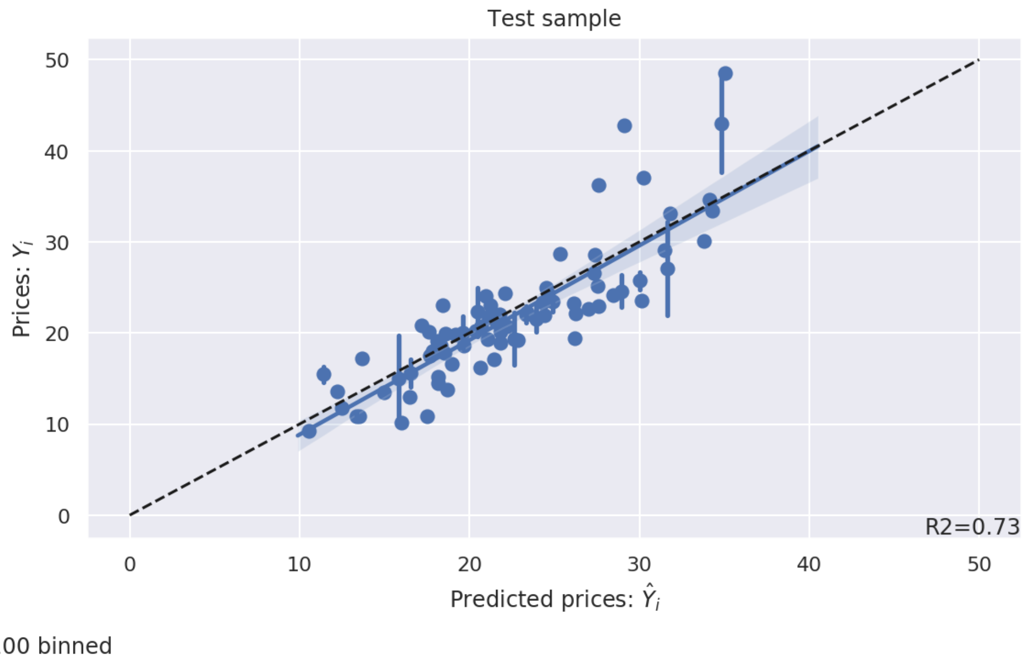
[25]:
```python
from sklearn.metrics import r2_score
r2=round(r2_score(y_test, y_test_pred), 2)
print("R2: %s" % r2)
```

```
R2: 0.73
```

1.12.7 Binned Regression Plots

[26]:
```python
# Regplot
g=sns.regplot(x= y_test_pred, y=y_test, x_bins=100)
g=g.set_title("Test sample")

plt.xlabel("Predicted prices: $\hat{Y}_i$")
```

```python
plt.ylabel("Prices: $Y_i$")
plt.annotate('R2={}'.format(r2),
             xy=(1, 0),  xycoords='axes fraction',
             horizontalalignment='right',
             verticalalignment='bottom')
plt.annotate('Notes: 100 binned',
             xy=(0, 0),  xycoords='figure fraction',
             horizontalalignment='left',
             verticalalignment='bottom')
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.tight_layout()
plt.show(g)
```

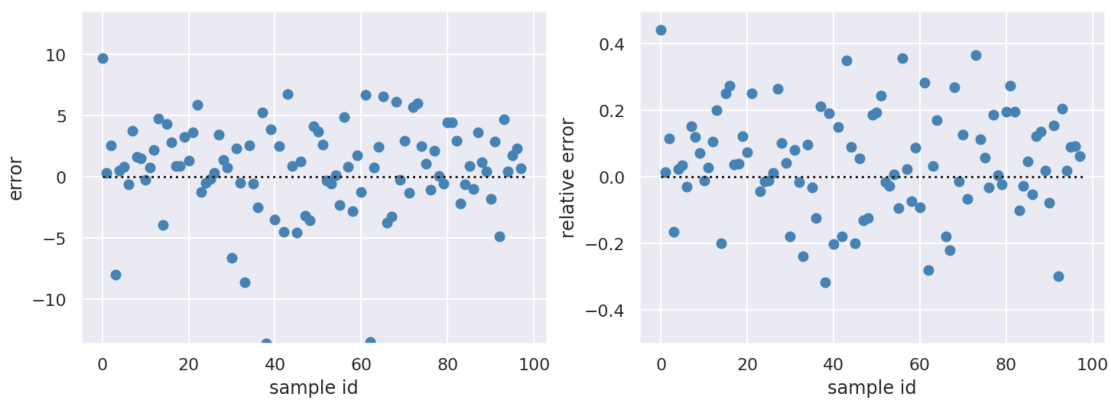


1.12.8 Plotting Regression Residuals

[27]:

```python
#Let us plot how good given and predicted values match on the training data set␣
↪(sic !).
def plot_fit_quality(values_test, predicted):

    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)

    x = np.arange(len(predicted))
```

```
    plt.scatter(x, predicted - values_test, color='steelblue', marker='o')

    plt.plot([0, len(predicted)], [0, 0], "k:")

    max_diff = np.max(np.abs(predicted - values_test))
    plt.ylim([-max_diff, max_diff])

    plt.ylabel("error")
    plt.xlabel("sample id")

    plt.subplot(1, 2, 2)

    plt.scatter(x, (predicted - values_test) / values_test, color='steelblue',␣
 ↪marker='o')
    plt.plot([0, len(predicted)], [0, 0], "k:")
    plt.ylim([-.5, .5])

    plt.ylabel("relative error")
    plt.xlabel("sample id")

plot_fit_quality(y_test, y_test_pred)
```



Your turn

`Train a ridge model and look at the goodeness of fit`

[28]:
```
from sklearn.linear_model import Ridge
```

[29]:
```
ridge_reg=Ridge(alpha=2)
ridge_reg.fit(X_train, y_train)

y_train_pred=ridge_reg.predict(X_train)
y_test_pred = ridge_reg.predict(X_test)
```

```
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)
print("train RMS: %s" % train_rmse)
print("test RMS: %s" % test_rmse)
print("train R2: %s" % round(r2_score(y_train, y_train_pred), 2))
print("test R2: %s" % round(r2_score(y_test, y_test_pred), 2))
```

```
train RMS: 3.6792311563556526
test RMS: 3.8991529766060626
train R2: 0.78
test R2: 0.73
```

[30]:
```
print('The coefficients of the features from the Ridge model:')
print(dict(zip(features, [round(x, 2) for x in ridge_reg.coef_])))
```

```
The coefficients of the features from the Ridge model:
{'CRIM': -0.09, 'ZN': 0.04, 'INDUS': -0.12, 'CHAS': 0.63, 'NOX': -4.79, 'RM':
3.34, 'AGE': -0.03, 'DIS': -1.21, 'RAD': 0.23, 'TAX': -0.01, 'PTRATIO': -0.8,
'B': 0.01, 'LSTAT': -0.38}
```

Polynomial regression

[31]:
```
from sklearn.preprocessing import PolynomialFeatures
poly_features=PolynomialFeatures(degree=2)
X_train_poly=poly_features.fit_transform(X_train)
X_test_poly=poly_features.fit_transform(X_test)

lin_reg = LinearRegression()
lin_reg.fit(X_train_poly, y_train)

y_test_pred = lin_reg.predict(X_test_poly)
test_rmse = mean_squared_error(y_test,y_test_pred)
test_rmse = np.sqrt(test_rmse)
print("test RMS: %s" % test_rmse)
print("train R2: %s" % round(r2_score(y_train, y_train_pred), 2))
print("test R2: %s" % round(r2_score(y_test, y_test_pred), 2))
```

```
test RMS: 3.7051965810729564
train R2: 0.78
test R2: 0.76
```

1.12.9 Lasso regression

[32]:
```
from sklearn.linear_model import Lasso
lasso_reg=Lasso(alpha=1)
lasso_reg.fit(X_train, y_train)
```

```
y_test_pred = lasso_reg.predict(X_test)
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)
print("test RMS: %s" % test_rmse)
print("train R2: %s" % round(r2_score(y_train, y_train_pred), 2))
print("test R2: %s" % round(r2_score(y_test, y_test_pred), 2))
```

```
test RMS: 4.579918320353294
train R2: 0.78
test R2: 0.63
```

1.12.10 Lasso regression – with scaled X

[33]:
```
from sklearn.linear_model import Lasso
lasso_reg=Lasso(alpha=1)
lasso_reg.fit(X_train_scaled, y_train)

y_test_pred = lasso_reg.predict(X_test_scaled)
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)
print("test RMS: %s" % test_rmse)
print("train R2: %s" % round(r2_score(y_train, y_train_pred), 2))
print("test R2: %s" % round(r2_score(y_test, y_test_pred), 2))
```

```
test RMS: 4.342235733030307
train R2: 0.78
test R2: 0.66
```

[34]:
```
print('The coefficients of the features from the Lasso model:')
print(dict(zip(features, [round(x, 2) for x in lasso_reg.coef_])))
```

```
The coefficients of the features from the Lasso model:
{'CRIM': -0.0, 'ZN': 0.0, 'INDUS': -0.23, 'CHAS': 0.0, 'NOX': -0.0, 'RM': 2.02,
'AGE': -0.0, 'DIS': 0.0, 'RAD': -0.0, 'TAX': -0.64, 'PTRATIO': -1.17, 'B': 0.06,
'LSTAT': -2.88}
```

Elastic Net

[35]:
```
from sklearn.linear_model import ElasticNet
elanet_reg=ElasticNet(random_state=0)
elanet_reg.fit(X_train_scaled, y_train)

y_test_pred = elanet_reg.predict(X_test_scaled)
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)
print("test RMS: %s" % test_rmse)
print("train R2: %s" % round(r2_score(y_train, y_train_pred), 2))
```

```
print("test R2: %s" % round(r2_score(y_test, y_test_pred), 2))
```

```
test RMS: 4.363821963597824
train R2: 0.78
test R2: 0.66
```

[36]:
```
print('The coefficients of the features from the Lasso model:')
print(dict(zip(features, [round(x, 2) for x in elanet_reg.coef_])))
```

```
The coefficients of the features from the Lasso model:
{'CRIM': -0.36, 'ZN': 0.07, 'INDUS': -0.62, 'CHAS': 0.0, 'NOX': -0.27, 'RM':
1.82, 'AGE': -0.19, 'DIS': -0.0, 'RAD': -0.0, 'TAX': -0.6, 'PTRATIO': -1.15,
'B': 0.36, 'LSTAT': -1.78}
```

1.13 Setting the regularization parameter: generalized Cross-Validation.

[37]:
```
alphas=np.logspace(-6, 6, 13)
```

[38]:
```
from sklearn import linear_model
lassocv_reg = linear_model.LassoCV(alphas=alphas)
lassocv_reg.fit(X_train, y_train)
alpha=lassocv_reg.alpha_
print("Best alpha", alpha)
```

```
Best alpha 0.001
```

1.13.1 Then re-run the model using the best α

[39]:
```
lasso_reg=Lasso(alpha=alpha)

lasso_reg.fit(X_train_scaled, y_train)

y_train_pred=lasso_reg.predict(X_train_scaled)
y_test_pred = lasso_reg.predict(X_test_scaled)
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)
print("test RMS: %s" % test_rmse)
print("train R2: %s" % round(r2_score(y_train, y_train_pred), 2))
print("test R2: %s" % round(r2_score(y_test, y_test_pred), 2))
```

```
test RMS: 3.881814011244504
train R2: 0.79
test R2: 0.73
```

1.14 Fine-tuning of the Model

1.14.1 Model Evaluation using Cross-Validation

<i class="fa fa-info-circle"></i> cross_val_score expect a utility function rather than a

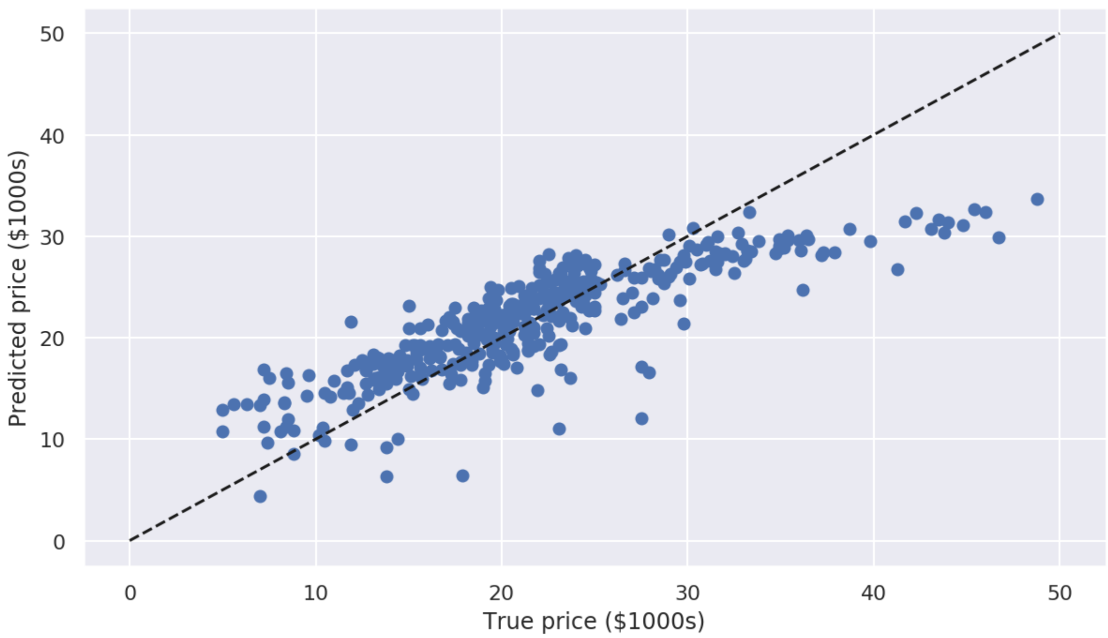
[40]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
# Perform 6-fold cross validation
scores = cross_val_score(elanet_reg, X_train_scaled, y_train,
                         scoring="neg_mean_squared_error", cv=5)
scores
```

[40]:
```
array([-14.96383737, -20.02416214, -20.20072418, -22.69197907,
       -19.66651518])
```

1.14.2 Make cross validated predictions

[41]:
```
y_train_pred_cv = cross_val_predict(elanet_reg, X_train_scaled, y_train, cv=5)
```

[42]:
```
## plt.figure(figsize=(4, 3))
plt.scatter(y_train, y_train_pred_cv)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.xlabel('True price ($1000s)')
plt.ylabel('Predicted price ($1000s)')
plt.tight_layout()
```



```
[43]: accuracy =r2_score(y_train, y_train_pred_cv)
print('Cross-Predicted Accuracy:', accuracy)
```

```
Cross-Predicted Accuracy: 0.6910345977768511
```

1.14.3 Hyperparameters tuning

```
[44]: from sklearn.model_selection import GridSearchCV
param_grid = [
  {'alpha': [0.0001, 0.001, 0.01, 0.1 ,1, 10],
      'l1_ratio':[.1,.5,.9,1]}

]
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
grid_search = GridSearchCV(elanet_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(X_train, y_train)
```

```
[44]: GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True,
                                  l1_ratio=0.5, max_iter=1000, normalize=False,
                                  positive=False, precompute=False,
                                  random_state=0, selection='cyclic',
                                  tol=0.0001, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10],
                          'l1_ratio': [0.1, 0.5, 0.9, 1]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='neg_mean_squared_error', verbose=0)
```

1.14.4 The best hyperparameter combination found:

```
[45]: grid_search.best_params_
```

```
[45]: {'alpha': 0.001, 'l1_ratio': 0.1}
```

```
[46]: grid_search.best_estimator_
```

```
[46]: ElasticNet(alpha=0.001, copy_X=True, fit_intercept=True, l1_ratio=0.1,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=0, selection='cyclic', tol=0.0001, warm_start=False)
```

1.14.5 Score of each hyperparameter combination tested during the grid search

[47]:
```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
3.8449782001198876 {'alpha': 0.0001, 'l1_ratio': 0.1}
3.8453507713983592 {'alpha': 0.0001, 'l1_ratio': 0.5}
3.8457594440788605 {'alpha': 0.0001, 'l1_ratio': 0.9}
3.8458675635883437 {'alpha': 0.0001, 'l1_ratio': 1}
3.8427979953843545 {'alpha': 0.001, 'l1_ratio': 0.1}
3.8429934252783204 {'alpha': 0.001, 'l1_ratio': 0.5}
3.8448097234318506 {'alpha': 0.001, 'l1_ratio': 0.9}
3.8456764681537368 {'alpha': 0.001, 'l1_ratio': 1}
3.8679356235567086 {'alpha': 0.01, 'l1_ratio': 0.1}
3.861942472619534 {'alpha': 0.01, 'l1_ratio': 0.5}
3.850373456685524 {'alpha': 0.01, 'l1_ratio': 0.9}
3.848624033246271 {'alpha': 0.01, 'l1_ratio': 1}
3.9119804095695105 {'alpha': 0.1, 'l1_ratio': 0.1}
3.9033448282073366 {'alpha': 0.1, 'l1_ratio': 0.5}
3.894063537112958 {'alpha': 0.1, 'l1_ratio': 0.9}
3.892576340848957 {'alpha': 0.1, 'l1_ratio': 1}
4.164310214998727 {'alpha': 1, 'l1_ratio': 0.1}
4.208846012935031 {'alpha': 1, 'l1_ratio': 0.5}
4.318715780226241 {'alpha': 1, 'l1_ratio': 0.9}
4.363200905580823 {'alpha': 1, 'l1_ratio': 1}
4.785822705650883 {'alpha': 10, 'l1_ratio': 0.1}
5.07285841622499 {'alpha': 10, 'l1_ratio': 0.5}
5.165833870770178 {'alpha': 10, 'l1_ratio': 0.9}
5.196973525805875 {'alpha': 10, 'l1_ratio': 1}
```

1.14.6 In a DataFrame

[48]:
```
df_cvres=pd.DataFrame(cvres)
df_cvres['mean_test_score_pos_sqrt']=df_cvres['mean_test_score'].apply(lambda x:
 ↪ np.sqrt(-x))
df_cvres['log_param_alpha']=df_cvres['param_alpha'].apply(lambda x: np.log(x))
df_cvres.head()
```

[48]:

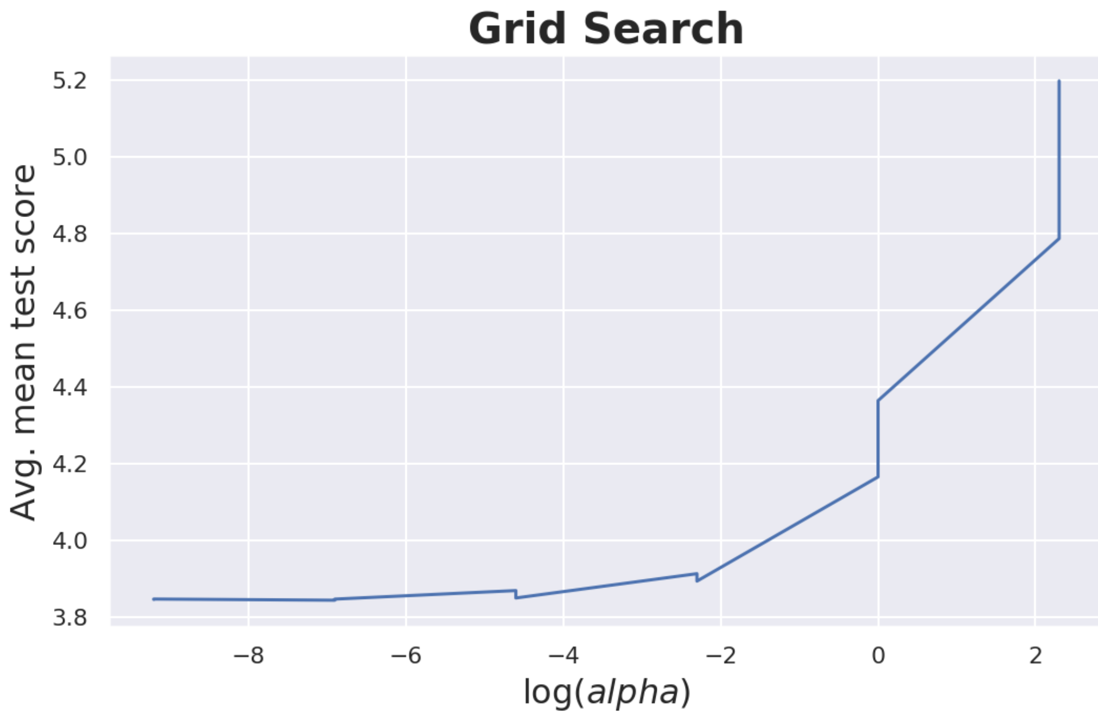
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha \
0	0.002857	0.000264	0.000778	0.000059	0.0001
1	0.002818	0.000329	0.000882	0.000147	0.0001
2	0.002827	0.000276	0.000876	0.000115	0.0001
3	0.002734	0.000088	0.000781	0.000074	0.0001
4	0.001955	0.000629	0.000514	0.000172	0.001

```
  param_l1_ratio                                  params  split0_test_score  \
0            0.1  {'alpha': 0.0001, 'l1_ratio': 0.1}          -7.724733
1            0.5  {'alpha': 0.0001, 'l1_ratio': 0.5}          -7.730899
2            0.9  {'alpha': 0.0001, 'l1_ratio': 0.9}          -7.737379
3              1    {'alpha': 0.0001, 'l1_ratio': 1}          -7.739050
4            0.1   {'alpha': 0.001, 'l1_ratio': 0.1}          -7.643106

   split1_test_score  split2_test_score  …  rank_test_score  \
0         -15.470735         -14.729138  …                4
1         -15.469364         -14.726713  …                5
2         -15.468092         -14.724325  …                7
3         -15.467791         -14.723734  …                8
4         -15.517337         -14.785278  …                1

   split0_train_score  split1_train_score  split2_train_score  \
0          -15.094455          -13.209017          -13.312006
1          -15.094138          -13.208753          -13.311775
2          -15.093978          -13.208618          -13.311657
3          -15.093964          -13.208606          -13.311646
4          -15.125461          -13.234697          -13.335068

   split3_train_score  split4_train_score  mean_train_score  std_train_score  \
0          -12.774886          -12.640780        -13.406229         0.881179
1          -12.774378          -12.640655        -13.405940         0.881169
2          -12.774126          -12.640587        -13.405793         0.881164
3          -12.774105          -12.640580        -13.405780         0.881164
4          -12.821180          -12.653476        -13.433976         0.882646

   mean_test_score_pos_sqrt  log_param_alpha
0                  3.844978        -9.210340
1                  3.845351        -9.210340
2                  3.845759        -9.210340
3                  3.845868        -9.210340
4                  3.842798        -6.907755

[5 rows x 24 columns]
```

1.14.7 Vizualize the grid search results

[49]:
```python
_, ax = plt.subplots(1,1)
plt.plot(df_cvres["log_param_alpha"], df_cvres["mean_test_score_pos_sqrt"])
ax.set_title("Grid Search", fontsize=20, fontweight='bold')
ax.set_xlabel("$\log (alpha)$", fontsize=16)
ax.set_ylabel('Avg. mean test score', fontsize=16)
```

[49]:
```
Text(0, 0.5, 'Avg. mean test score')
```



1.14.8 Other possibility: for randomized search of hyperparameters

[50]:
```python
from sklearn.model_selection import RandomizedSearchCV
```

1.15 What is not covered today:

- more advanced regression algorithms (gradient boosting, random forest)
- classification algorithm
- pipelines