Predicting Continuous Target Variables with Regression Analysis

Hsi-Pin Ma 馬席彬

http://lms.nthu.edu.tw/course/40724

Department of Electrical Engineering

National Tsing Hua University

Outline

- Introduction to regression
- Exploring the Housing Dataset
- Implementing an ordinary least squares linear regression model
- Fitting a robust regression model using RANSAC
- Evaluating the performance of linear regression models
- Using regularized methods for regression
- Turning a linear regression model into a curve - polynomial regression

Introduction to Regression

Linear Regression

- To model the relationship between one or multiple features and a ***continuous*** target variable
  - A subcategory of supervised learning
  - discrete category (classification) vs. continuous target variable (regression)

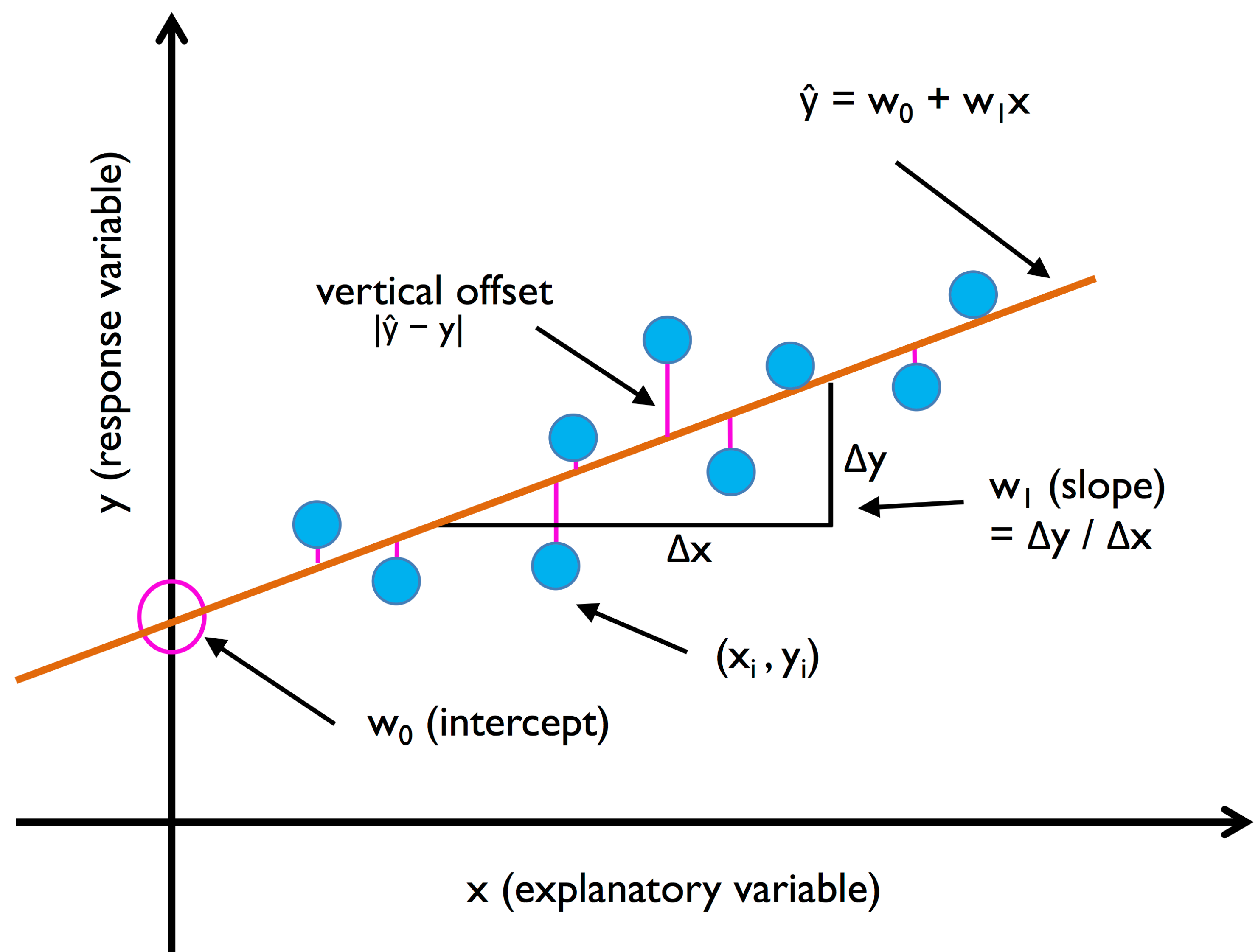
Simple (Univariate) Linear Regression

- To model the relationship between a single feature (explanatory variable ***x***) and a continuous valued response (target variable ***y***)

$$y = w_0 + w_1 x$$

$$\hat{y} = h(x) \equiv w_0 + w_1 x$$

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$$

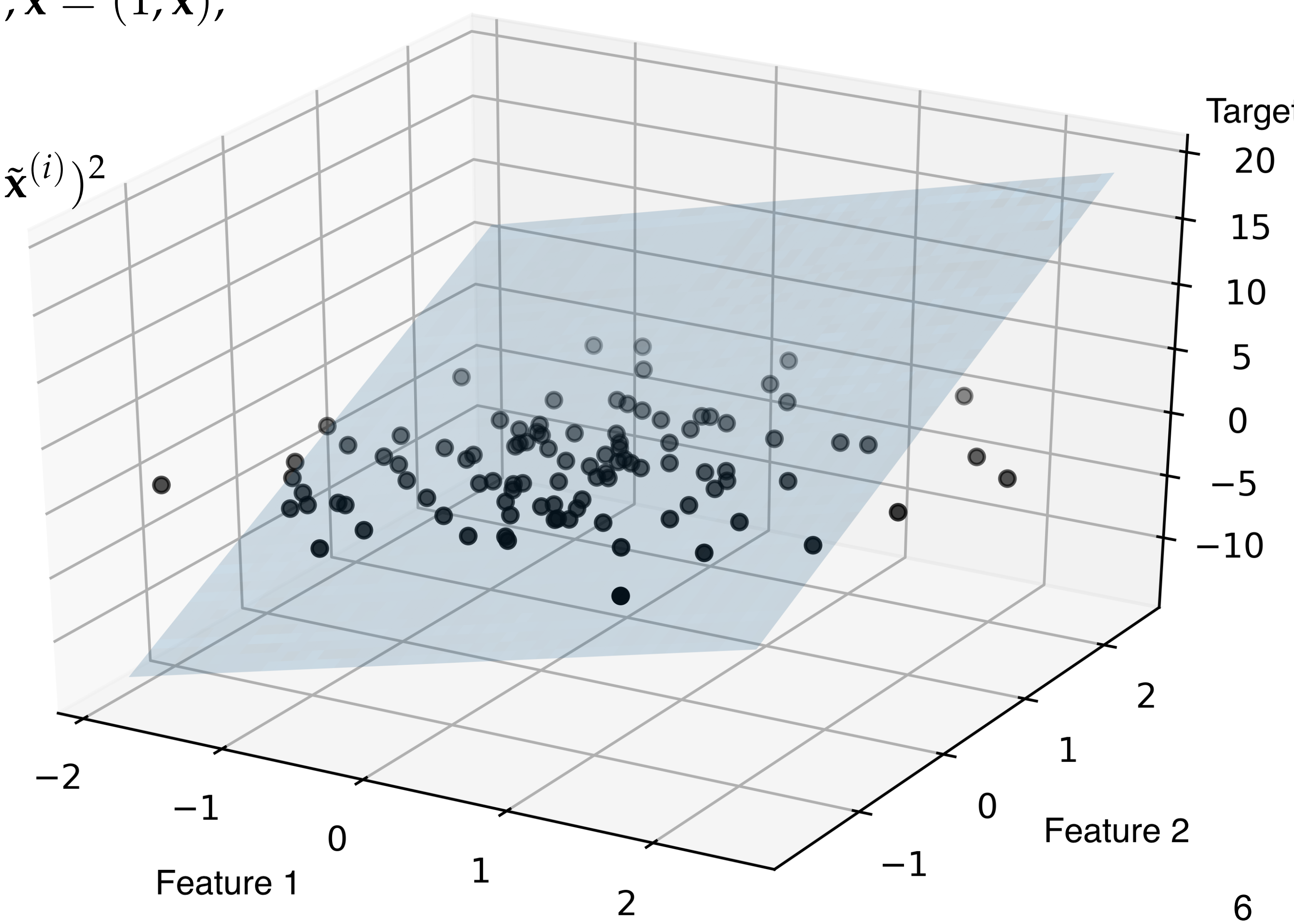


Multiple Linear Regression

- Generalize the model to multiple explanatory variables $y = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \sum_{i=0}^{m} w_i x_i = w^T x$

$$\begin{aligned} \hat{y} &= h(x_1, \ldots, x_m) \equiv w_0 + w_1 x_1 + \cdots + w_m x_m \\ &= \sum_{i=0}^{m} w_i x_i = \mathbf{w} \cdot \tilde{\mathbf{x}},\ \mathbf{w} = (w_0, w_1, \ldots, w_m), \tilde{\mathbf{x}} = (1, \mathbf{x}), \end{aligned}$$

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2 = \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - \mathbf{w} \cdot \tilde{\mathbf{x}}^{(i)})^2$$



Exploring the Housing Dataset

Housing Dataset

- 506 instance (houses), each with 13 features and the house price (MEDV) as the target variable

```
1. CRIM      per capita crime rate by town
2. ZN        proportion of residential land zoned for lots over
                 25,000 sq.ft.
3. INDUS     proportion of non-retail business acres per town
4. CHAS      Charles River dummy variable (= 1 if tract bounds
                 river; 0 otherwise)
5. NOX       nitric oxides concentration (parts per 10 million)
6. RM        average number of rooms per dwelling
7. AGE       proportion of owner-occupied units built prior to 1940
8. DIS       weighted distances to five Boston employment centres
9. RAD       index of accessibility to radial highways
10. TAX      full-value property-tax rate per $10,000
11. PTRATIO  pupil-teacher ratio by town
12. B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks
                 by town
13. LSTAT    % lower status of the population
14. MEDV     Median value of owner-occupied homes in $1000s
```

Load the Housing Dataset

```
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/rasbt/'
                 'python-machine-learning-book-2nd-edition'
                 '/master/code/ch10/housing.data.txt',
                 header=None,
                 sep='\s+')

df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS',
              'NOX', 'RM', 'AGE', 'DIS', 'RAD',
              'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
df.head()
```

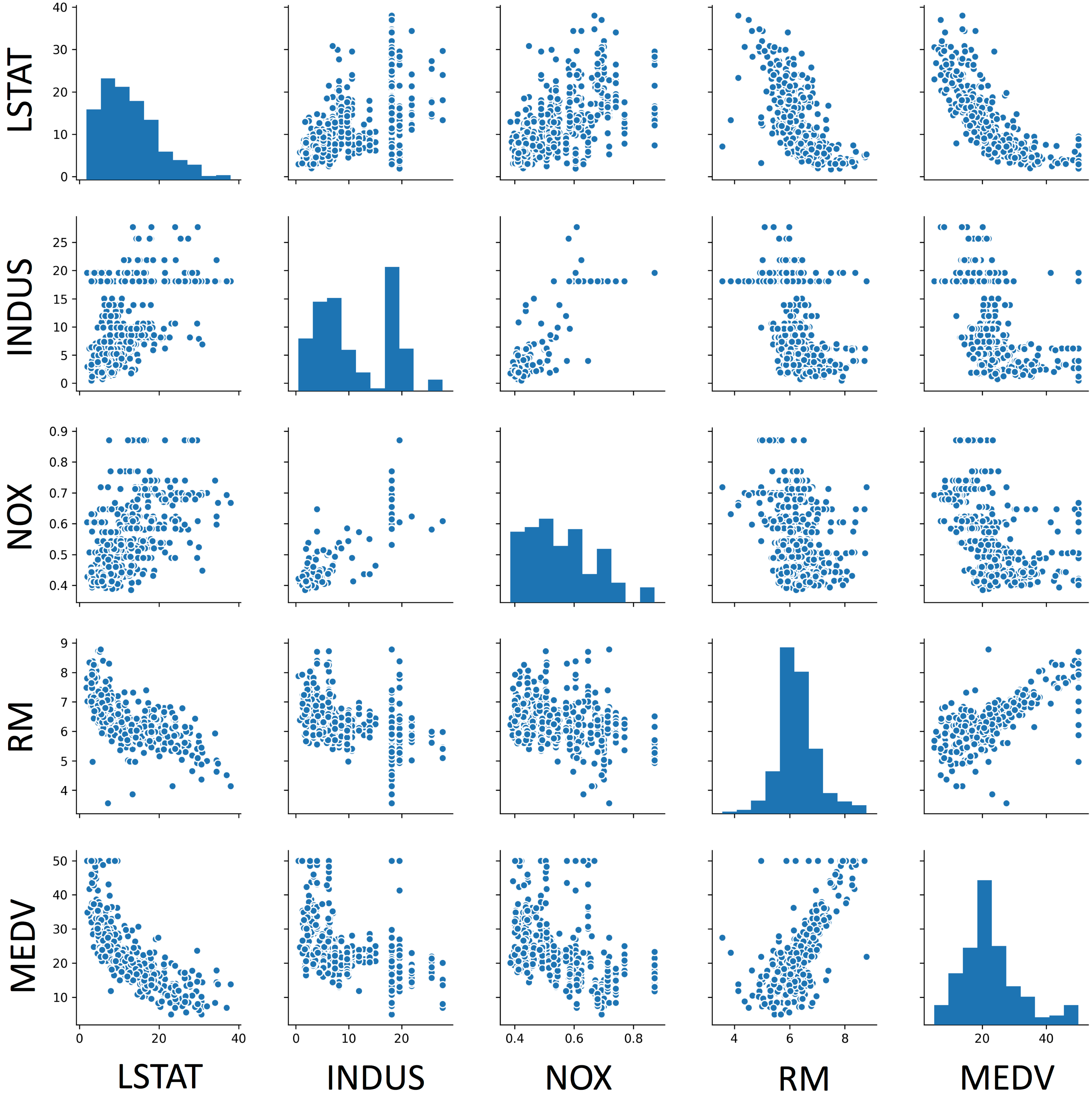
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	ME
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	2
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	2
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	3
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	3
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	3

Exploratory Data Analysis (EDA)

- Visualize the important characteristics of a dataset before training a model
  - Create a **scatterplot matrix** to visualize the pairwise correlations between the different features
  - Use **pairplot** function from Seaborn library
    - conda install seaborn or pip install seaborn

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
cols = ['LSTAT', 'INDUS', 'NOX', 'RM', 'MEDV']

sns.pairplot(df[cols], size=2.5)
plt.tight_layout()
# plt.savefig('images/10_03.png', dpi=300)
plt.show()
```


LSTAT
INDUS
NOX
RM
MEDV
LSTAT
INDUS
NOX
RM
MEDV

Correlation Matrix

- Use correlation matrix to quantify and summarize linear relationships between variables
  - Identical to a covariance matrix computed from standardized features
  - A square matrix that contains the Pearson product-moment correlation coefficient (Pearson's r), which measures the linear dependence between pairs of features

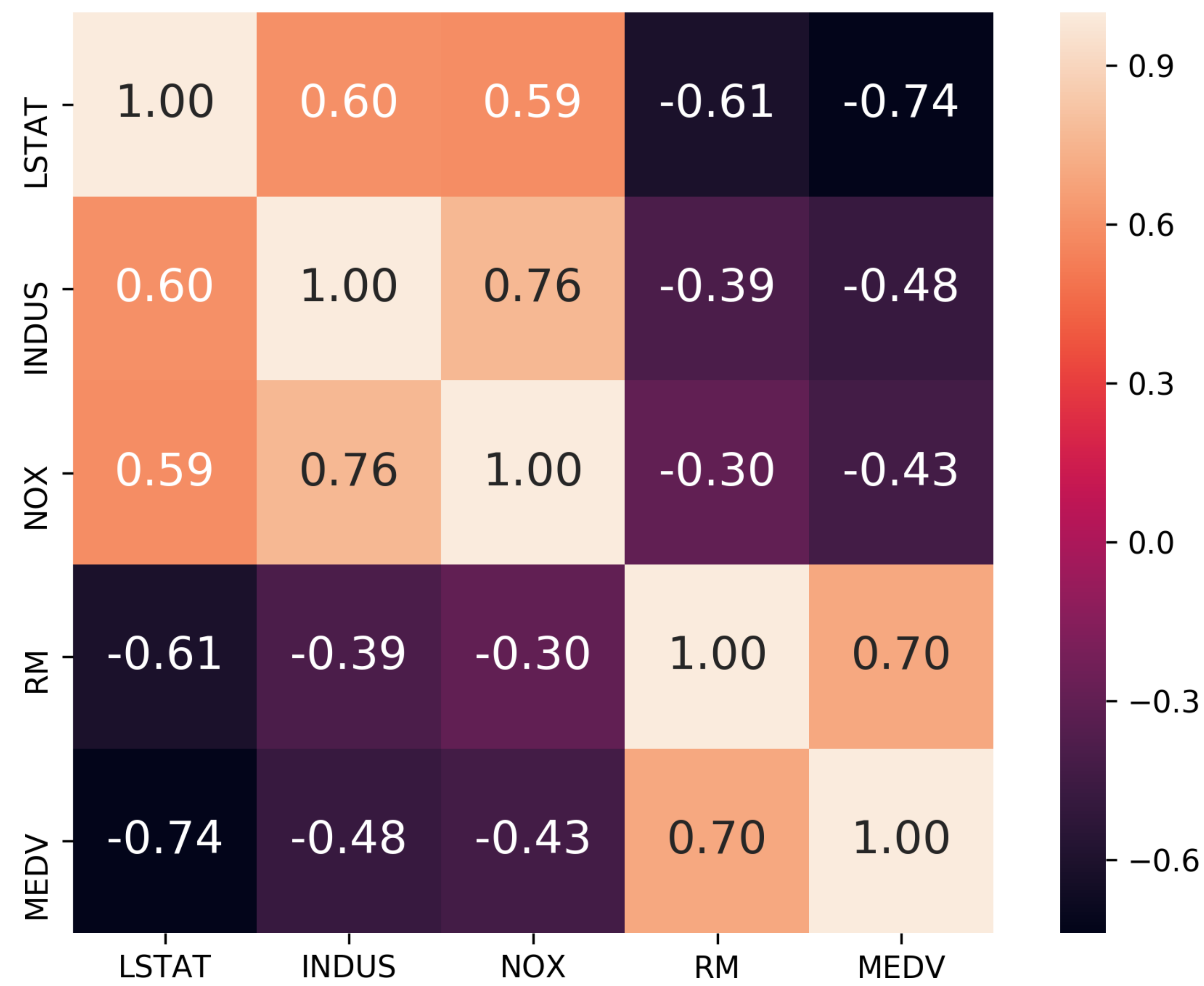
$$\sigma_{xy} = \frac{1}{n}\sum_{i}^{n}\left(x^{(i)} - \mu_x\right)\left(y^{(i)} - \mu_y\right)$$

$$r = \frac{\sum_{i=1}^{n}\left[\left(x^{(i)} - \mu_x\right)\left(y^{(i)} - \mu_y\right)\right]}{\sqrt{\sum_{i=1}^{n}\left(x^{(i)} - \mu_x\right)^2}\sqrt{\sum_{i=1}^{n}\left(y^{(i)} - \mu_y\right)^2}} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

$$-1 \le r_{x,y} \le +1.$$

Correlation Matrix

```python
import numpy as np


cm = np.corrcoef(df[cols].values.T)
#sns.set(font_scale=1.5)
hm = sns.heatmap(cm,
                 cbar=True,
                 annot=True,
                 square=True,
                 fmt='.2f',
                 annot_kws={'size': 15},
                 yticklabels=cols,
                 xticklabels=cols)

plt.tight_layout()
plt.savefig('images/10_04.png', dpi=300)
plt.show()
```



Implementing an Ordinary Least Squares (OLS) Linear Regression Model

Minimizing the Objective Function of Linear Regression

- Cost function J (Sum of Squared Errors, SSE)

$$J(w) = \frac{1}{2}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 \qquad \hat{y} = w^T x$$

  - Identical to the cost function of Adaline
  - OLS regression can be understood as Adaline without the unit step function, so we can obtain continuous target values
  - Use GD or SGD for optimization

Linear Regression GD

```
class LinearRegressionGD(object):

    def __init__(self, eta=0.001, n_iter=20):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []

        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        return self.net_input(X)
```

Training a Regressor

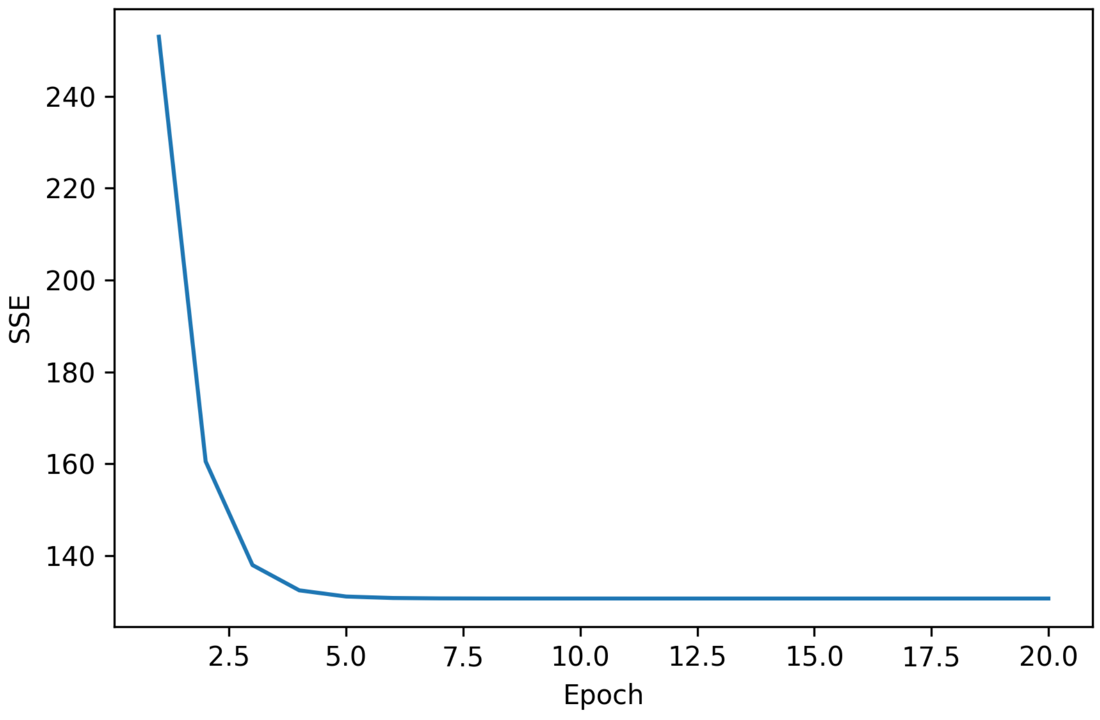
```
X = df[['RM']].values
y = df['MEDV'].values
```

```
from sklearn.preprocessing import StandardScaler



sc_x = StandardScaler()
sc_y = StandardScaler()
X_std = sc_x.fit_transform(X)
y_std = sc_y.fit_transform(y[:, np.newaxis]).flatten()
```

```
lr = LinearRegressionGD()
lr.fit(X_std, y_std)
```

```
<__main__.LinearRegressionGD at 0x1172ce780>
```

```
plt.plot(range(1, lr.n_iter+1), lr.cost_)
plt.ylabel('SSE')
plt.xlabel('Epoch')
#plt.tight_layout()
#plt.savefig('images/10_05.png', dpi=300)
plt.show()
```

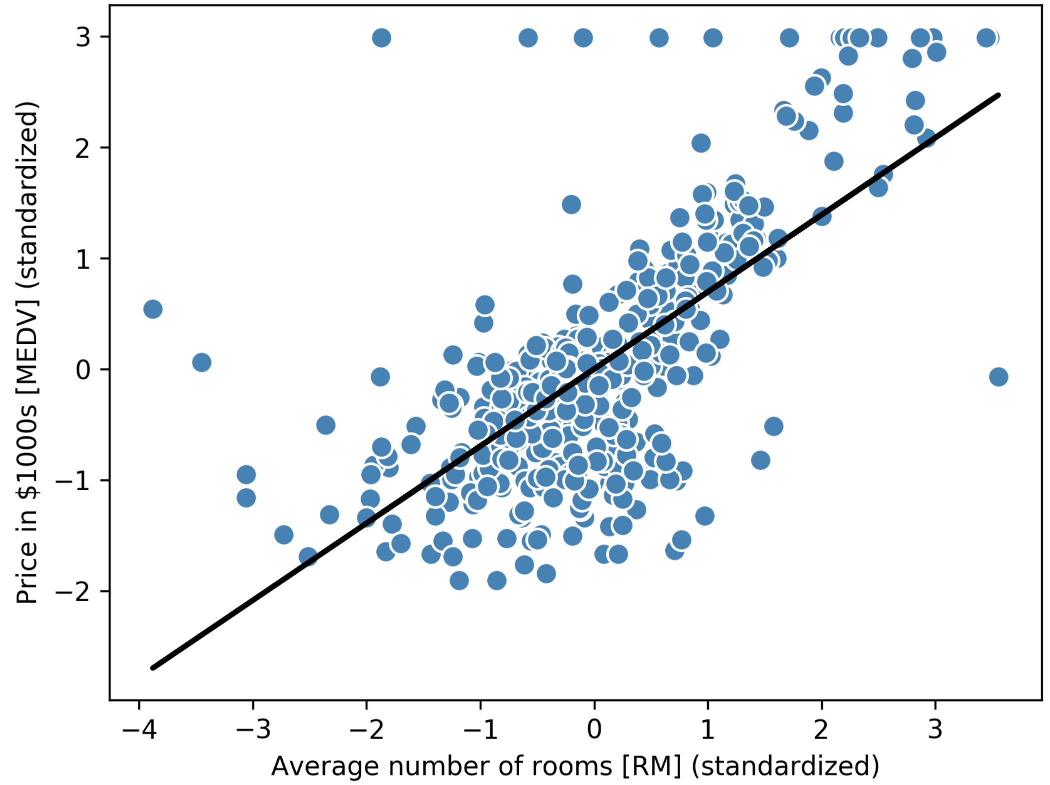


Laboratory for Reliable Computing

Visualize the Linear Regression

```
def lin_regplot(X, y, model):
    plt.scatter(X, y, c='steelblue', edgecolor='white', s=70)
    plt.plot(X, model.predict(X), color='black', lw=2)
    return
```

```
lin_regplot(X_std, y_std, lr)
plt.xlabel('Average number of rooms [RM] (standardized)')
plt.ylabel('Price in $1000s [MEDV] (standardized)')

plt.savefig('images/10_06.png', dpi=300)
plt.show()
```



Predicting with the Linear Regression Model

```
print('Slope: %.3f' % lr.w_[1])
print('Intercept: %.3f' % lr.w_[0])
```

```
Slope: 0.695
Intercept: -0.000
```

```
num_rooms_std = sc_x.transform(np.array([[5.0]]))
price_std = lr.predict(num_rooms_std)
print("Price in $1000s: %.3f" % sc_y.inverse_transform(price_std))
```

```
Price in $1000s: 10.840
```

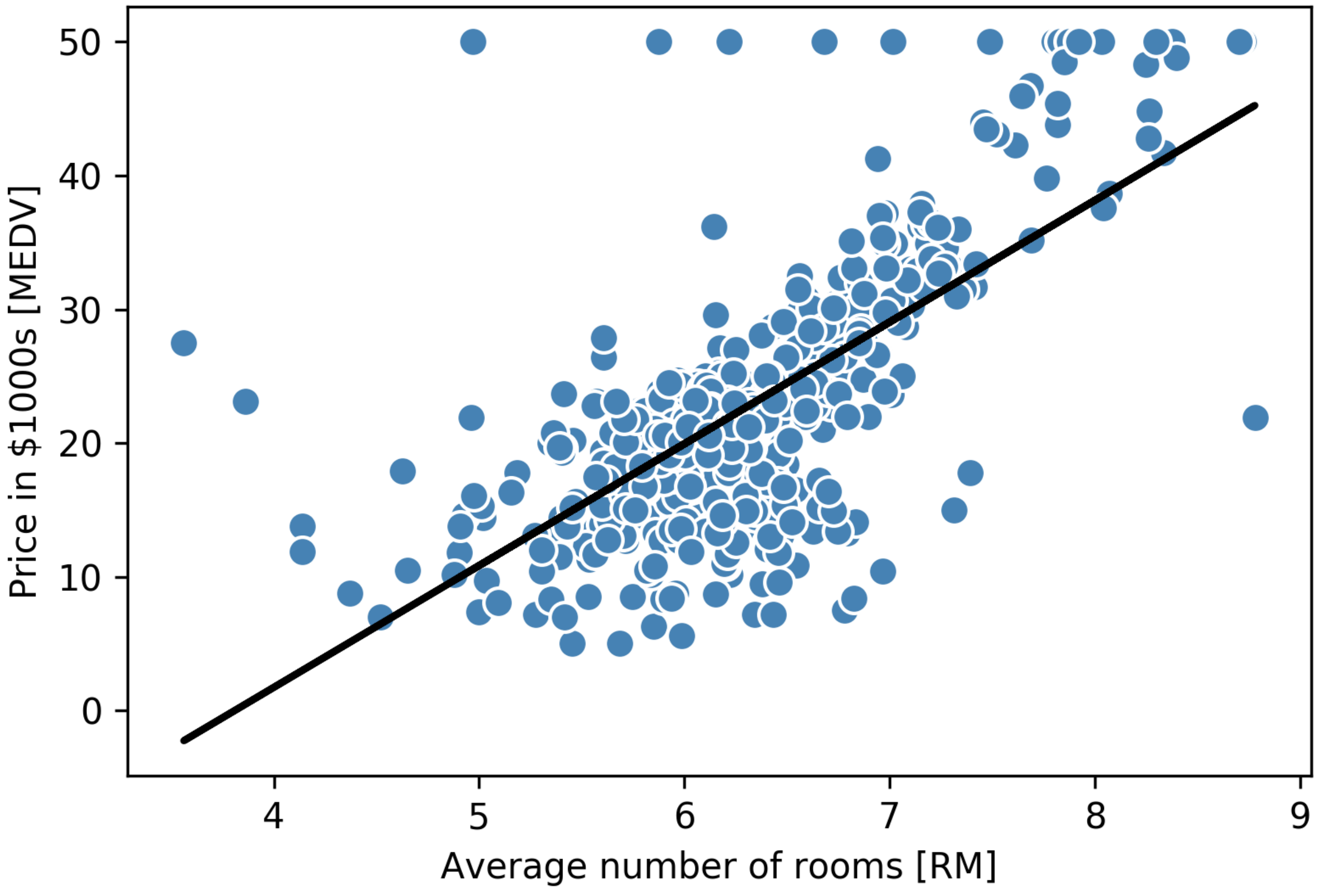
Estimating the Coefficient of a Regression Model via Scikit-learn

```
from sklearn.linear_model import LinearRegression
```

```
slr = LinearRegression()
slr.fit(X, y)
y_pred = slr.predict(X)
print('Slope: %.3f' % slr.coef_[0])
print('Intercept: %.3f' % slr.intercept_)
```

```
Slope: 9.102
Intercept: -34.671
```

```
lin_regplot(X, y, slr)
plt.xlabel('Average number of rooms [RM]')
plt.ylabel('Price in $1000s [MEDV]')

#plt.savefig('images/10_07.png', dpi=300)
plt.show()
```



Fitting a Robust Regression Model Using RANSAC

Outliers

- Linear regression can be heavily affected by the presence of "outliers"
- A very small subset of our data may have a big effect on the estimated coefficient
- In practice, removing ***outliers*** always requires our own judgement as well as domain knowledge
- An alternative to throwing away outliers, a robust method of regression using RANdom SAmple Consensus (RANSAC) algorithm fits a regression model to a subset of the data, called ***inliers***

The RANSAC Algorithm

- Select a random number of samples to be inliers and fit the model
- Test all other data points against the fitted model and add those points that fall within a user-given tolerance to the inliers
- Refit the model using all inliers
- Estimate the error of the fitted model versus inliers
- Terminate the algorithm if the performance meets a certain user-defined threshold or if a fixed number of iterations were reached; go back to step 1 otherwise

RANSAC using Scikit-learn

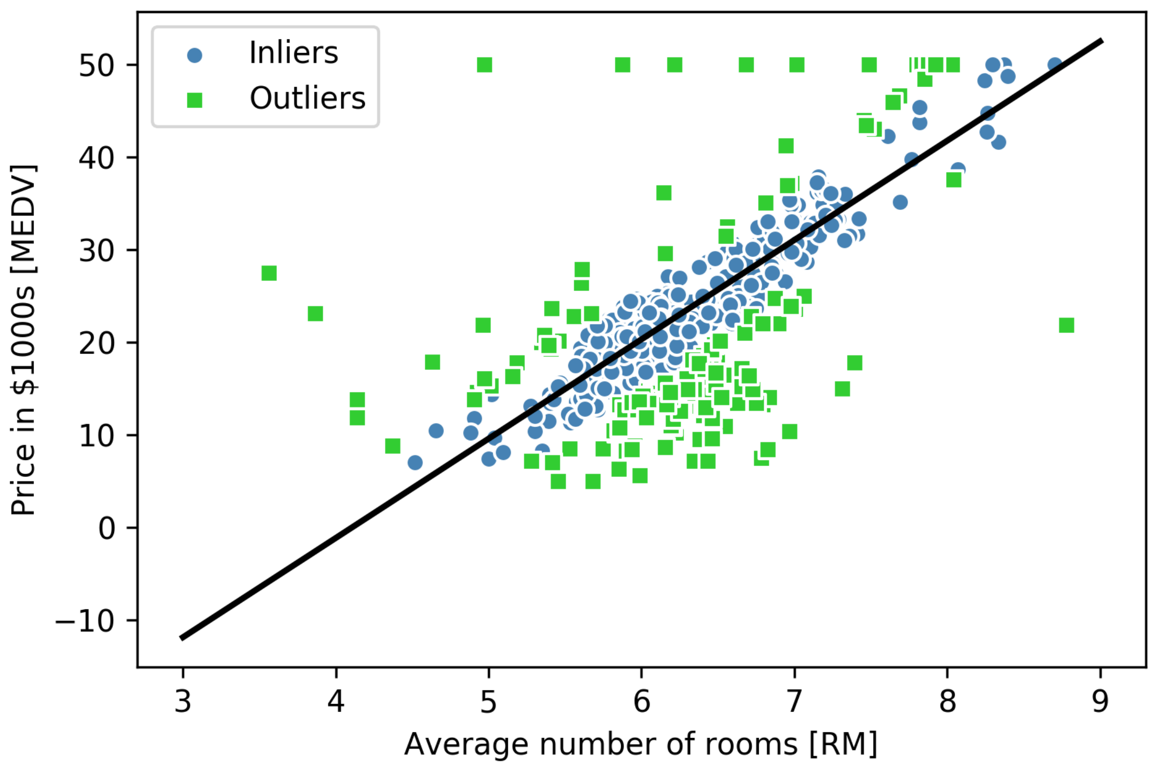
```
from sklearn.linear_model import RANSACRegressor

ransac = RANSACRegressor(LinearRegression(),
                         max_trials=100,
                         min_samples=50,
                         loss='absolute_loss',
                         residual_threshold=5.0,
                         random_state=0)



ransac.fit(X, y)
```

RANSAC using Scikit-learn

```
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

line_X = np.arange(3, 10, 1)
line_y_ransac = ransac.predict(line_X[:, np.newaxis])
plt.scatter(X[inlier_mask], y[inlier_mask],
            c='steelblue', edgecolor='white',
            marker='o', label='Inliers')
plt.scatter(X[outlier_mask], y[outlier_mask],
            c='limegreen', edgecolor='white',
            marker='s', label='Outliers')
plt.plot(line_X, line_y_ransac, color='black', lw=2)
plt.xlabel('Average number of rooms [RM]')
plt.ylabel('Price in $1000s [MEDV]')
plt.legend(loc='upper left')

#plt.savefig('images/10_08.png', dpi=300)
plt.show()
```

```
print('Slope: %.3f' % ransac.estimator_.coef_[0])
print('Intercept: %.3f' % ransac.estimator_.intercept_)
```

```
Slope: 10.735
Intercept: -44.089
```



Evaluating the Performance of Linear Regression Models

Performance Evaluation

- We will build a multiple linear regression model for the Housing dataset by using all features
- Evaluate the generalization performance by
  - Residual plot
  - Mean square error (MSE)
  - The coefficient of determination R^2

Train the Linear Regression Model

```
from sklearn.model_selection import train_test_split

X = df.iloc[:, :-1].values
y = df['MEDV'].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0)
```

```
slr = LinearRegression()

slr.fit(X_train, y_train)
y_train_pred = slr.predict(X_train)
y_test_pred = slr.predict(X_test)
```

Train the Linear Regression Model

```
import numpy as np
import scipy as sp

ary = np.array(range(100000))
```

```
%timeit np.linalg.norm(ary)
```

```
309 µs ± 11.4 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
%timeit sp.linalg.norm(ary)
```

```
307 µs ± 8.14 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
%timeit np.sqrt(np.sum(ary**2))
```

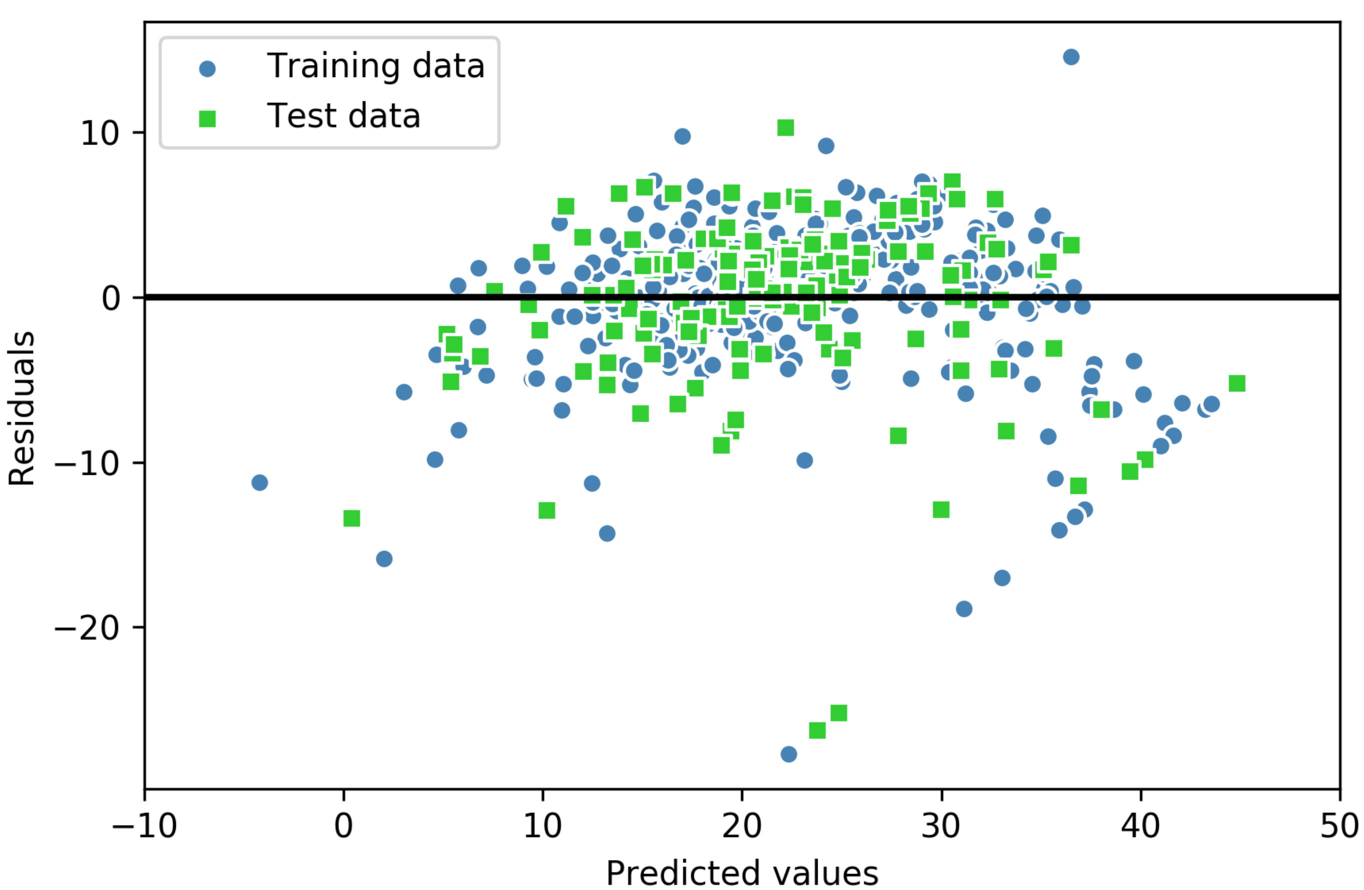
```
251 µs ± 8.93 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Residual Plot

- Plot the residuals versus the predicted values to diagnose the regression model
- For a good regression model, we would expect that the residuals should be randomly scattered around the centerline
- Residual plot can be used for detect outliers, which are represented by the points with a large deviation from the centerline

Residual Plot

```
plt.scatter(y_train_pred,  y_train_pred - y_train,
            c='steelblue', marker='o', edgecolor='white',
            label='Training data')
plt.scatter(y_test_pred,  y_test_pred - y_test,
            c='limegreen', marker='s', edgecolor='white',
            label='Test data')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=-10, xmax=50, color='black', lw=2)
plt.xlim([-10, 50])
plt.tight_layout()

# plt.savefig('images/10_09.png', dpi=300)
plt.show()
```



Mean Squared Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error

print('MSE train: %.3f, test: %.3f' % (
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_test, y_test_pred)))
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
```

```
MSE train: 19.958, test: 27.196
R^2 train: 0.765, test: 0.673
```

Coefficient of Determination R^2

R^2 is a rescaled version of the MSE

sum of squared errors

$$R^2 = 1 - \frac{SSE}{SST}$$

$$SST = \sum_{i=1}^{n} \left(y^{(i)} - \mu_y \right)^2$$

$$1 - \frac{\frac{1}{n}\sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2}{\frac{1}{n}\sum_{i=1}^{n} \left(y^{(i)} - \mu_y \right)^2}$$

$$1 - \frac{MSE}{Var(y)}$$

```
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
```

```
R^2 train: 0.765, test: 0.673
```

Using the Regularized Methods for Regression

Regularization for Linear Regression

- Regularization is commonly used to tackle the overfitting problem
- Regularization for linear regression is achieved by adding a term to the cost function which is proportional to a norm of the weighted vector
- Three popular approaches
  - Ridge regression
  - Least absolute shrinkage and selection operator (LASSO)
  - Elastic net

Ridge Regression

- Cost function $J(w)_{Ridge} = \sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda \| w \|_2^2$ $\quad L2: \quad \lambda \| w \|_2^2 = \lambda \sum_{j=1}^{m} w_j^{\,2}$
  - L2-panelized model
  - Does not regularize the intercept w_0

```python
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1.0)
```

Least Absolute Shrinkage and Selection Operator (LASSO)

- **Cost function** $J(w)_{LASSO} = \sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda \| w \|_1$ $\quad L1: \lambda \| w \|_1 = \lambda \sum_{j=1}^{m} |w_j|$
  - L1-panelized model
  - Does not regularize the intercept w_0
  - Can lead to sparse model
  - LASSO selects at most N coefficients to be nonzero if m>N

```python
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
y_train_pred = lasso.predict(X_train)
y_test_pred = lasso.predict(X_test)
print(lasso.coef_)
```

```python
print('MSE train: %.3f, test: %.3f' % (
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_test, y_test_pred)))
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
```

```
MSE train: 20.926, test: 28.876
R^2 train: 0.753, test: 0.653
```

```
[-0.11311792  0.04725111 -0.03992527  0.96478874 -0.          3.72289616
 -0.02143106 -1.23370405  0.20469    -0.0129439  -0.85269025  0.00795847
 -0.52392362]
```

Elastic Net

- Cost function $$J(w)_{ElasticNet} = \sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda_1 \sum_{j=1}^{m} w_j^{\ 2} + \lambda_2 \sum_{j=1}^{m} \left|w_j\right|$$
  - A compromise between ridge regression and LASSO as to have L1-penalty to generate sparsity and L2-penalty to overcome the limitation of the number of selected features

```
from sklearn.linear_model import ElasticNet
elanet = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

Turning a Linear Regression Model into a Curve - Polynomial Regression

Polynomial Regression

- When relation between explanatory and response variables are not linear

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_d x^d$$

  - ***x***: explanatory variable, ***y***: response variable, ***d***: degree of the polynomial

- Still considered a multiple linear regression model because of the linear regression coefficient ***w***.

Adding Polynomial Terms Using Scikit-learn

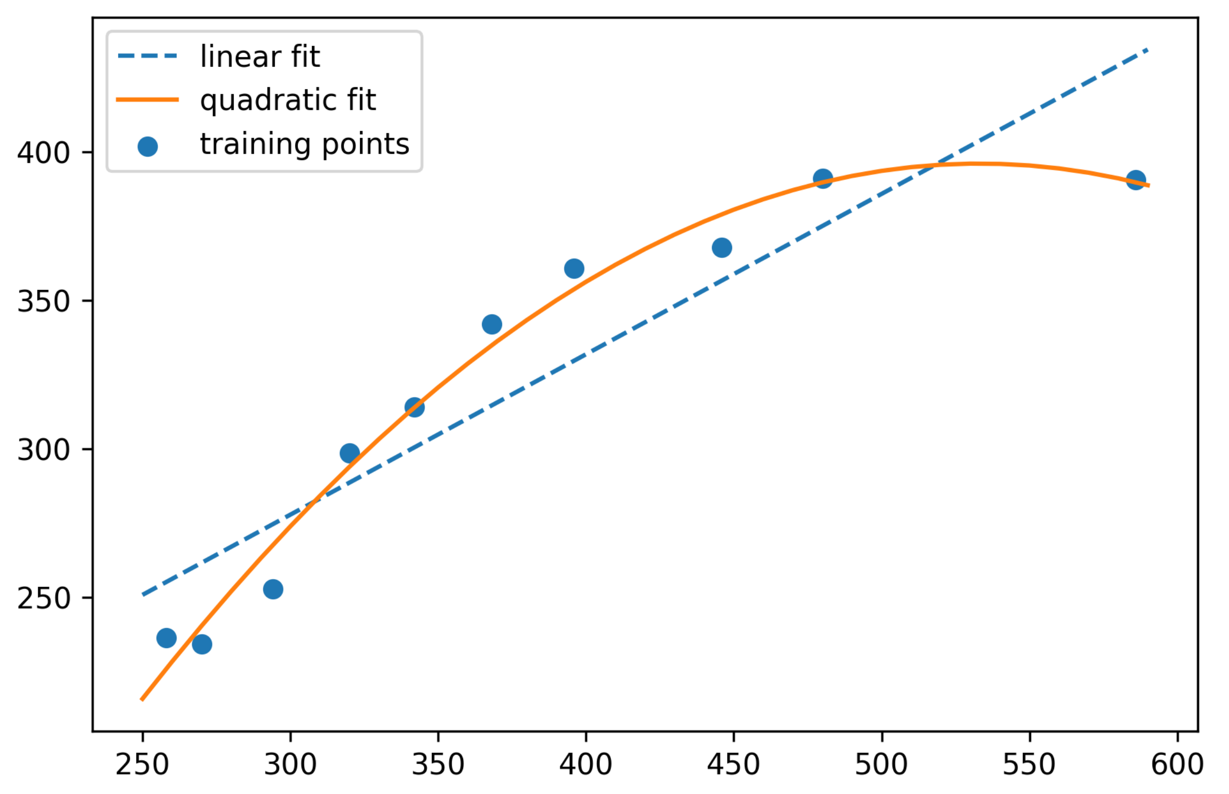
```
X = np.array([258.0, 270.0, 294.0,
              320.0, 342.0, 368.0,
              396.0, 446.0, 480.0, 586.0])\
             [:, np.newaxis]

y = np.array([236.4, 234.4, 252.8,
              298.6, 314.2, 342.2,
              360.8, 368.0, 391.2,
              390.8])
```

Add a second degree polynomial term

```
from sklearn.preprocessing import PolynomialFeatures

lr = LinearRegression()
pr = LinearRegression()
quadratic = PolynomialFeatures(degree=2)
X_quad = quadratic.fit_transform(X)
```

Adding Polynomial Terms Using Scikit-learn

```
# fit linear features
lr.fit(X, y)
X_fit = np.arange(250, 600, 10)[:, np.newaxis]
y_lin_fit = lr.predict(X_fit)

# fit quadratic features
pr.fit(X_quad, y)
y_quad_fit = pr.predict(quadratic.fit_transform(X_fit))

# plot results
plt.scatter(X, y, label='training points')
plt.plot(X_fit, y_lin_fit, label='linear fit', linestyle='--')
plt.plot(X_fit, y_quad_fit, label='quadratic fit')
plt.legend(loc='upper left')

plt.tight_layout()
#plt.savefig('images/10_10.png', dpi=300)
plt.show()
```

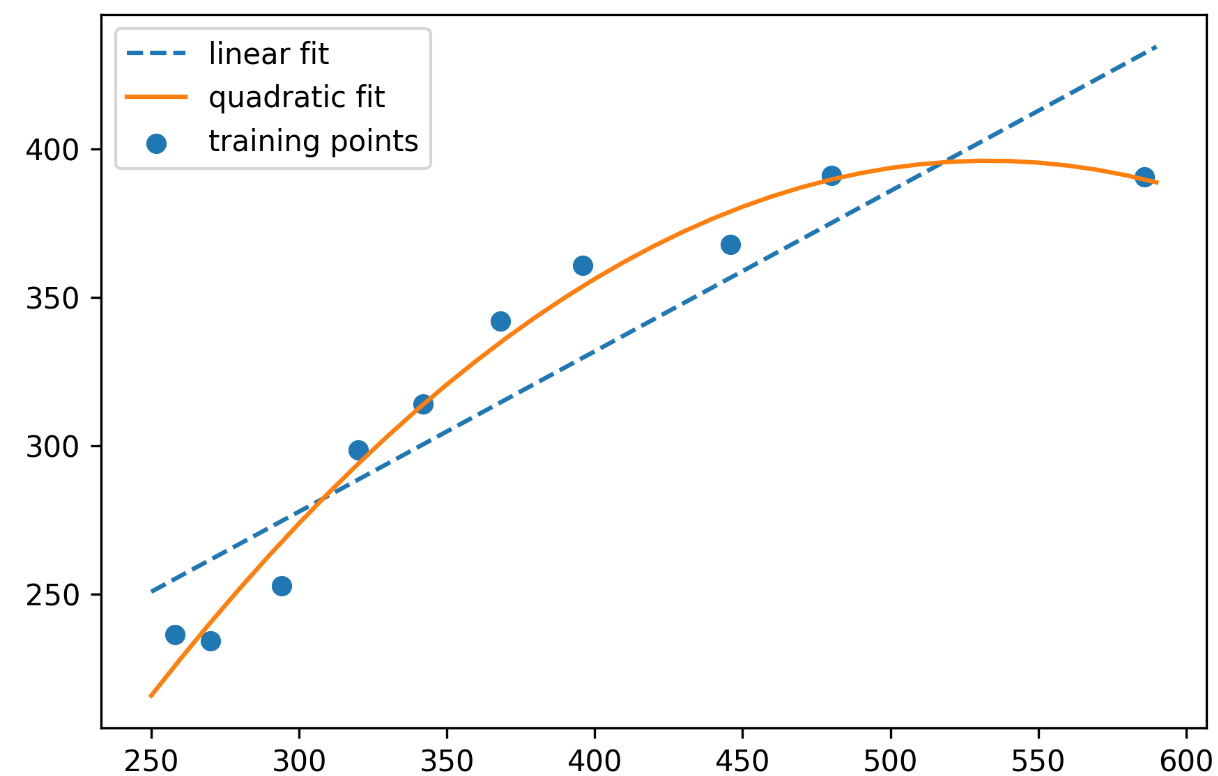


Adding Polynomial Terms Using Scikit-learn

```
y_lin_pred = lr.predict(X)
y_quad_pred = pr.predict(X_quad)
```

```
print('Training MSE linear: %.3f, quadratic: %.3f' % (
        mean_squared_error(y, y_lin_pred),
        mean_squared_error(y, y_quad_pred)))
print('Training R^2 linear: %.3f, quadratic: %.3f' % (
        r2_score(y, y_lin_pred),
        r2_score(y, y_quad_pred)))
```

```
Training MSE linear: 569.780, quadratic: 61.330
Training R^2 linear: 0.832, quadratic: 0.982
```

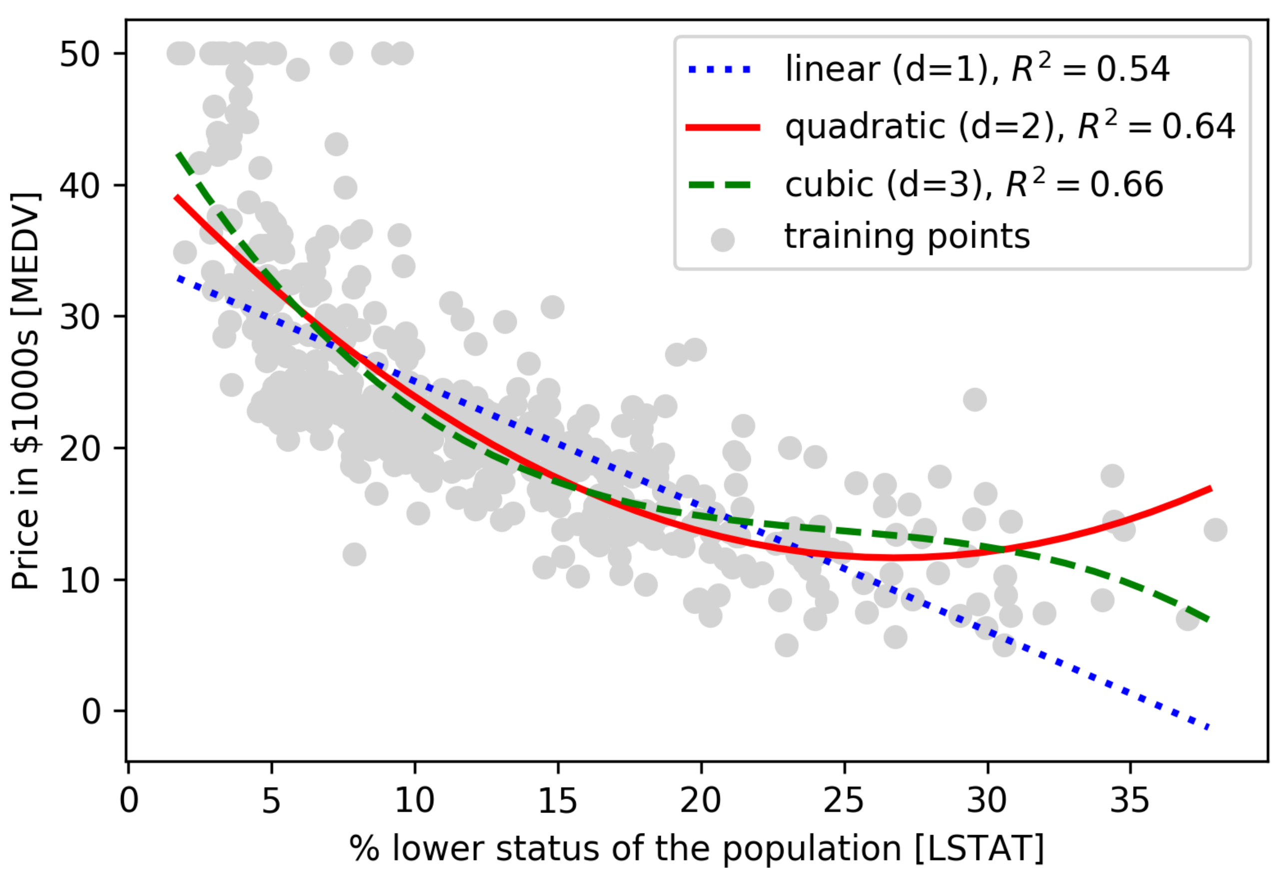


Modeling Nonlinear Relationships in the Housing Dataset

```
X = df[['LSTAT']].values
y = df['MEDV'].values

regr = LinearRegression()

# create quadratic features
quadratic = PolynomialFeatures(degree=2)
cubic = PolynomialFeatures(degree=3)
X_quad = quadratic.fit_transform(X)
X_cubic = cubic.fit_transform(X)

# fit features
X_fit = np.arange(X.min(), X.max(), 1)[:, np.newaxis]

regr = regr.fit(X, y)
y_lin_fit = regr.predict(X_fit)
linear_r2 = r2_score(y, regr.predict(X))

regr = regr.fit(X_quad, y)
y_quad_fit = regr.predict(quadratic.fit_transform(X_fit))
quadratic_r2 = r2_score(y, regr.predict(X_quad))
regr = regr.fit(X_cubic, y)
y_cubic_fit = regr.predict(cubic.fit_transform(X_fit))
cubic_r2 = r2_score(y, regr.predict(X_cubic))
```

Modeling Nonlinear Relationships in the Housing Dataset

```python
# plot results
plt.scatter(X, y, label='training points', color='lightgray')

plt.plot(X_fit, y_lin_fit,
         label='linear (d=1), $R^2=%.2f$' % linear_r2,
         color='blue',
         lw=2,
         linestyle=':')

plt.plot(X_fit, y_quad_fit,
         label='quadratic (d=2), $R^2=%.2f$' % quadratic_r2,
         color='red',
         lw=2,
         linestyle='-')

plt.plot(X_fit, y_cubic_fit,
         label='cubic (d=3), $R^2=%.2f$' % cubic_r2,
         color='green',
         lw=2,
         linestyle='--')
plt.xlabel('% lower status of the population [LSTAT]')
plt.ylabel('Price in $1000s [MEDV]')
plt.legend(loc='upper right')

#plt.savefig('images/10_11.png', dpi=300)
plt.show()
```



Transforming the Dataset

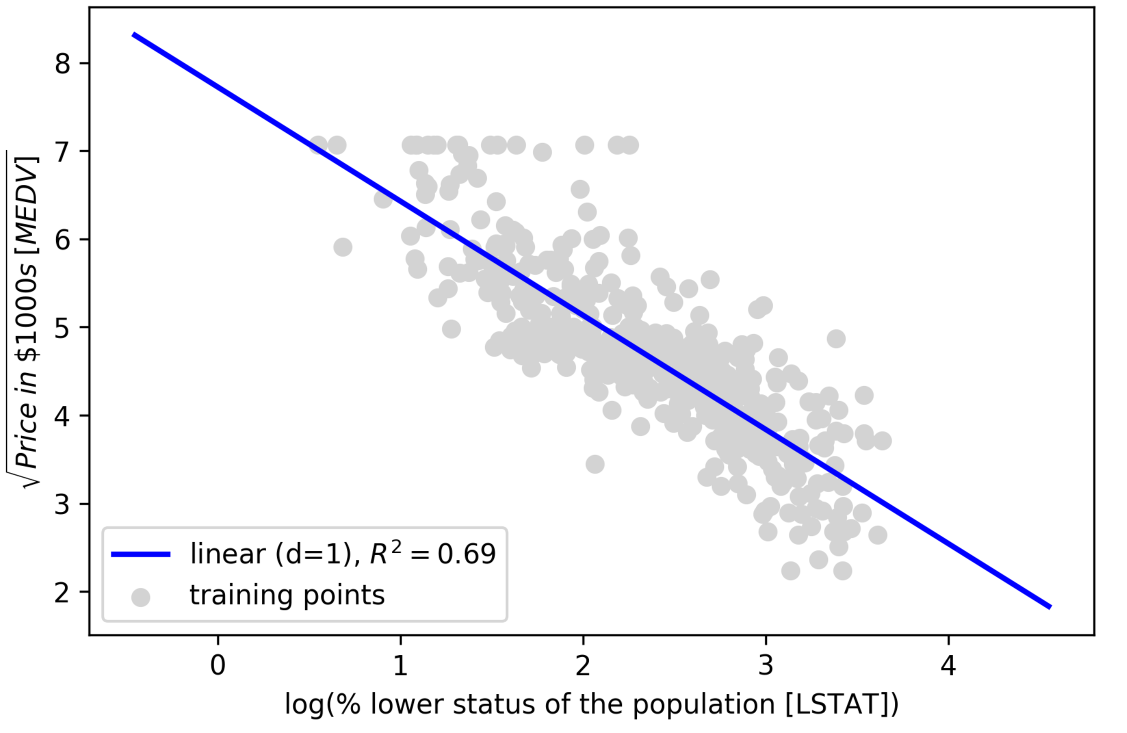
- However, polynomial features are not always the best choice for modeling nonlinear features
  - MEDV-LSTAT: may lead to the hypothesis that a log-transformation of the LSTAT feature variable and the square root of MEDV may project the data onto a linear feature space

```
X = df[['LSTAT']].values
y = df['MEDV'].values

# transform features
X_log = np.log(X)
y_sqrt = np.sqrt(y)

# fit features
X_fit = np.arange(X_log.min()-1, X_log.max()+1, 1)[:, np.newaxis]
```

$$\sqrt{y} = \ln x$$

Transforming the Dataset

```python
# fit features
X_fit = np.arange(X_log.min()-1, X_log.max()+1, 1)[:, np.newaxis]

regr = regr.fit(X_log, y_sqrt)
y_lin_fit = regr.predict(X_fit)
linear_r2 = r2_score(y_sqrt, regr.predict(X_log))

# plot results
plt.scatter(X_log, y_sqrt, label='training points', color='lightgray')

plt.plot(X_fit, y_lin_fit,
         label='linear (d=1), $R^2=%.2f$' % linear_r2,
         color='blue',
         lw=2)

plt.xlabel('log(% lower status of the population [LSTAT])')
plt.ylabel('$\sqrt{Price \; in \; \$1000s \; [MEDV]}$')
plt.legend(loc='lower left')

plt.tight_layout()
#plt.savefig('images/10_12.png', dpi=300)
plt.show()
```



Decision Tree Regression

- A (binary) decision tree will partition the feature space into disjoint regions through simple (binary) questions
- The predicted target value associated with a region is the average of the target values of the instances in the training dataset lying in the region
- The impurity metric used in the decision tree regression is the mean square error (MSE)

$$IG(D_p, x_i) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

true target value

predicted target value (sample mean)

$$I(t) = MSE(t) = \frac{1}{N_t} \sum_{i \in D_t} \left(y^{(i)} - \hat{y}_t \right)^2$$

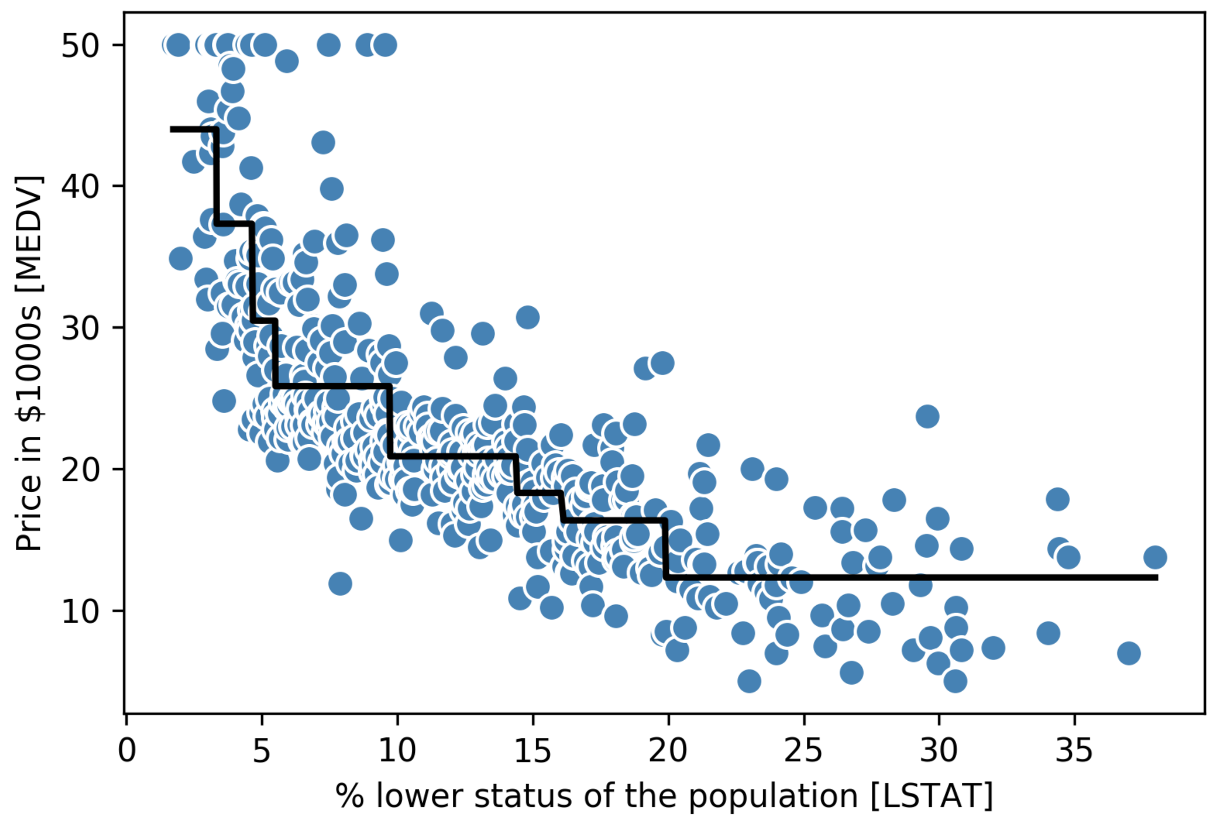
$$\hat{y}_t = \frac{1}{N_t} \sum_{i \in D_t} y^{(i)}$$

# of training samples at node t

training subset at node t

Decision Tree Regression

```
from sklearn.tree import DecisionTreeRegressor

X = df[['LSTAT']].values
y = df['MEDV'].values

tree = DecisionTreeRegressor(max_depth=3)
tree.fit(X, y)

sort_idx = X.flatten().argsort()

lin_regplot(X[sort_idx], y[sort_idx], tree)
plt.xlabel('% lower status of the population [LSTAT]')
plt.ylabel('Price in $1000s [MEDV]')
#plt.savefig('images/10_13.png', dpi=300)
plt.show()
```



Random Forest Regression

- The number of decision trees in a random forest is a hyperparameter
- We use MSE impurity reduction (i.e. variance reduction) to grow the individual decision tree
- The predicted target value is calculated as the average prediction over all decision trees
- In scikit-learn, the random forest regression is implemented in the **ensemble** module as the class **RandomForestRegressor**

Random Forest Regression

```
X = df.iloc[:, :-1].values
y = df['MEDV'].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=1)
```

```
from sklearn.ensemble import RandomForestRegressor

forest = RandomForestRegressor(n_estimators=1000,
                               criterion='mse',
                               random_state=1,
                               n_jobs=-1)
forest.fit(X_train, y_train)
y_train_pred = forest.predict(X_train)
y_test_pred = forest.predict(X_test)

print('MSE train: %.3f, test: %.3f' % (
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_test, y_test_pred)))
print('R^2 train: %.3f, test: %.3f' % (
        r2_score(y_train, y_train_pred),
        r2_score(y_test, y_test_pred)))
```

```
MSE train: 1.642, test: 11.052
R^2 train: 0.979, test: 0.878
```

Random Forest Regression

```
plt.scatter(y_train_pred,
            y_train_pred - y_train,
            c='steelblue',
            edgecolor='white',
            marker='o',
            s=35,
            alpha=0.9,
            label='training data')
plt.scatter(y_test_pred,
            y_test_pred - y_test,
            c='limegreen',
            edgecolor='white',
            marker='s',
            s=35,
            alpha=0.9,
            label='test data')

plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=-10, xmax=50, lw=2, color='black')
plt.xlim([-10, 50])
plt.tight_layout()

# plt.savefig('images/10_14.png', dpi=300)
plt.show()
```