Introduction to regression

Boston housing data

```
In [1]: boston = pd.read_csv('boston.csv')

In [2]: print(boston.head())
      CRIM    ZN  INDUS  CHAS     NX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296.0
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0

   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

Creating feature and target arrays

```
In [3]: X = boston.drop('MEDV', axis=1).values

In [4]: y = boston['MEDV'].values
```

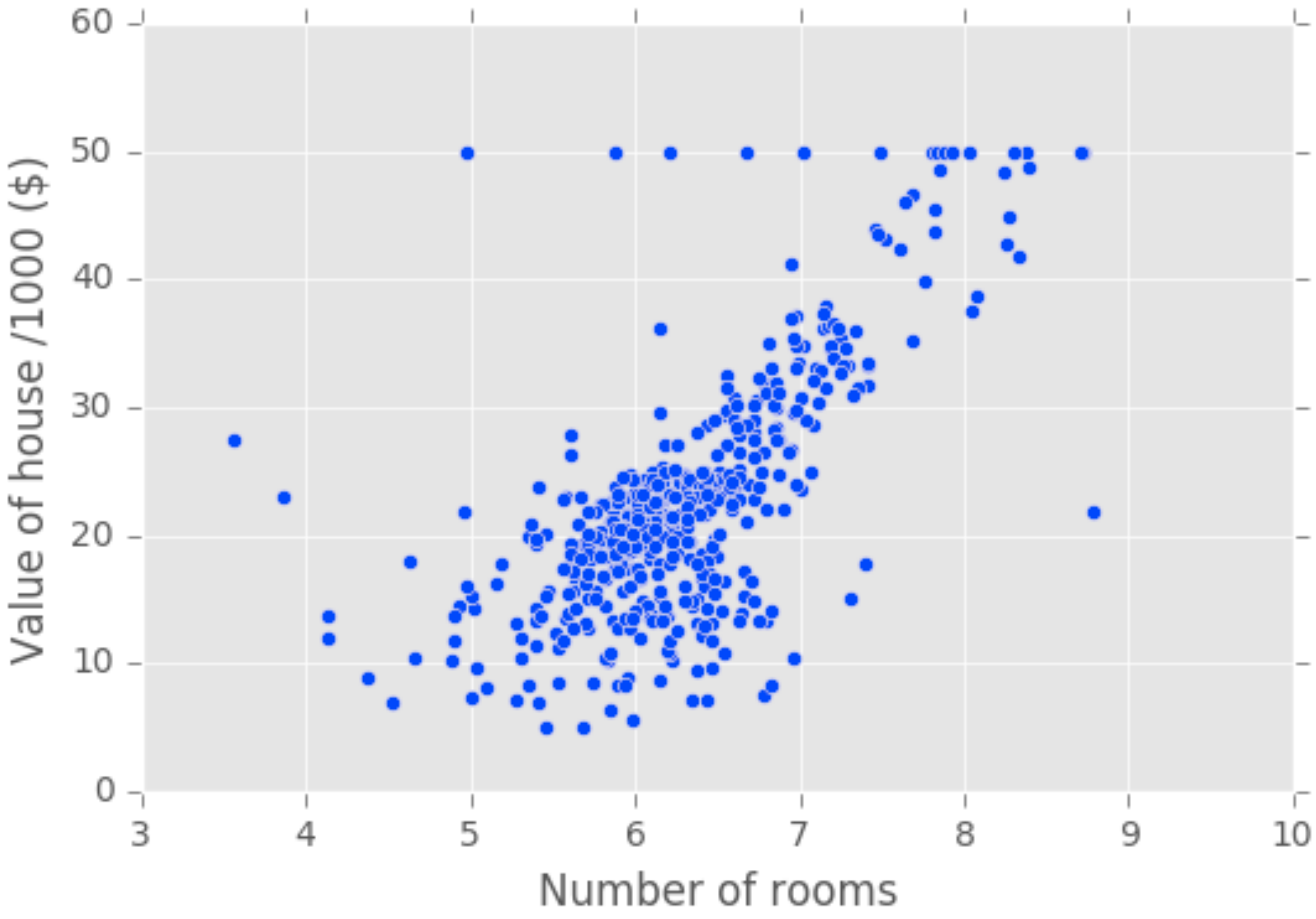
Predicting house value from a single feature

```
In [5]: X_rooms = X[:,5]

In [6]: type(X_rooms), type(y)
Out[6]: (numpy.ndarray, numpy.ndarray)

In [7]: y = y.reshape(-1, 1)

In [8]: X_rooms = X_rooms.reshape(-1, 1)
```

Plotting house value vs. number of rooms

```
In [9]: plt.scatter(X_rooms, y)

In [10]: plt.ylabel('Value of house /1000 ($)')

In [11]: plt.xlabel('Number of rooms')

In [12]: plt.show();
```

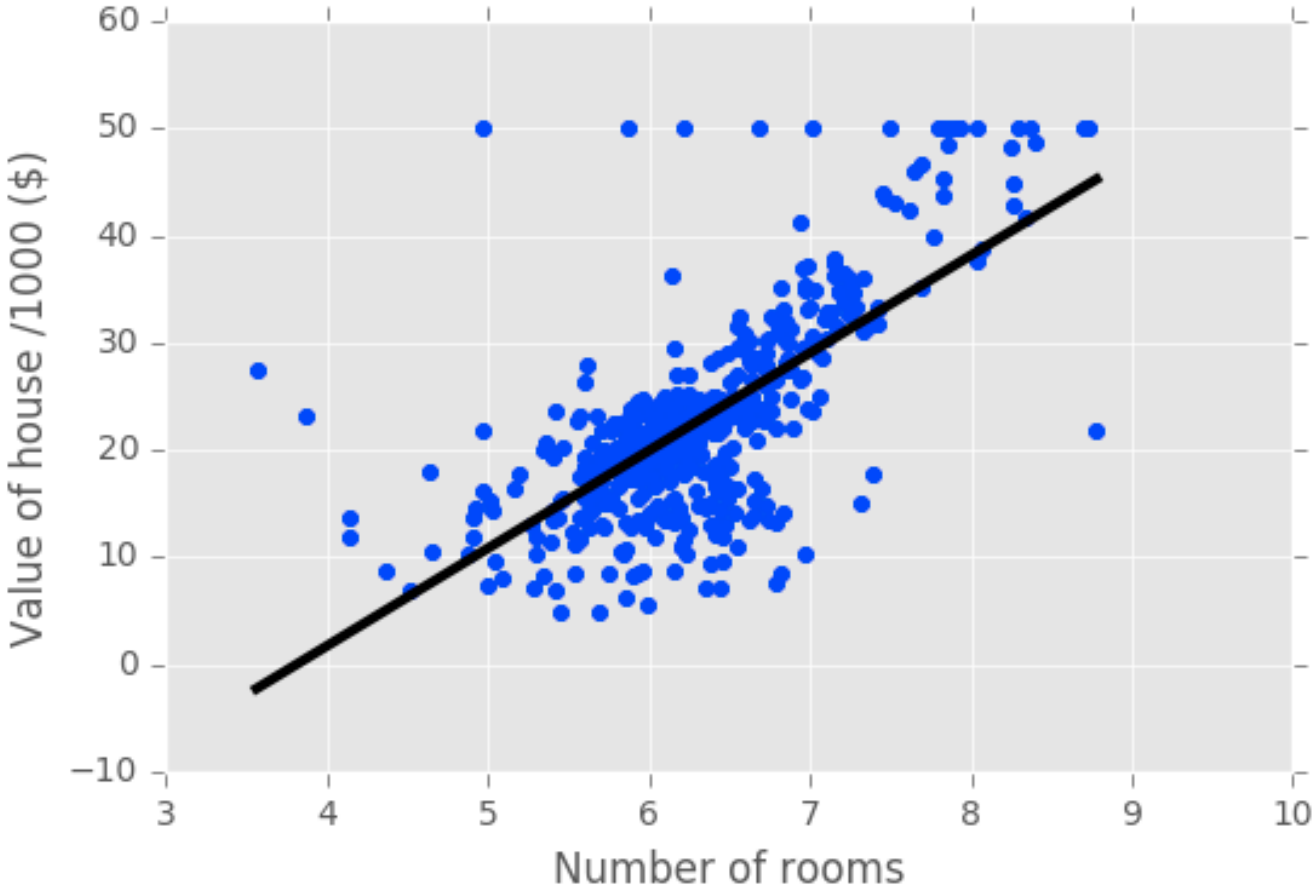
Plotting house value vs. number of rooms



Fitting a regression model

```
In [13]: import numpy as np

In [14]: from sklearn import linear_model

In [15]: reg = linear_model.LinearRegression()

In [16]: reg.fit(X_rooms, y)

In [17]: prediction_space = np.linspace(min(X_rooms),
    ...:                                max(X_rooms)).reshape(-1, 1)

In [18]: plt.scatter(X_rooms, y, color='blue')

In [19]: plt.plot(prediction_space, reg.predict(prediction_space),
    ...:          color='black', linewidth=3)

In [20]: plt.show()
```

Fitting a regression model



Let’s practice!

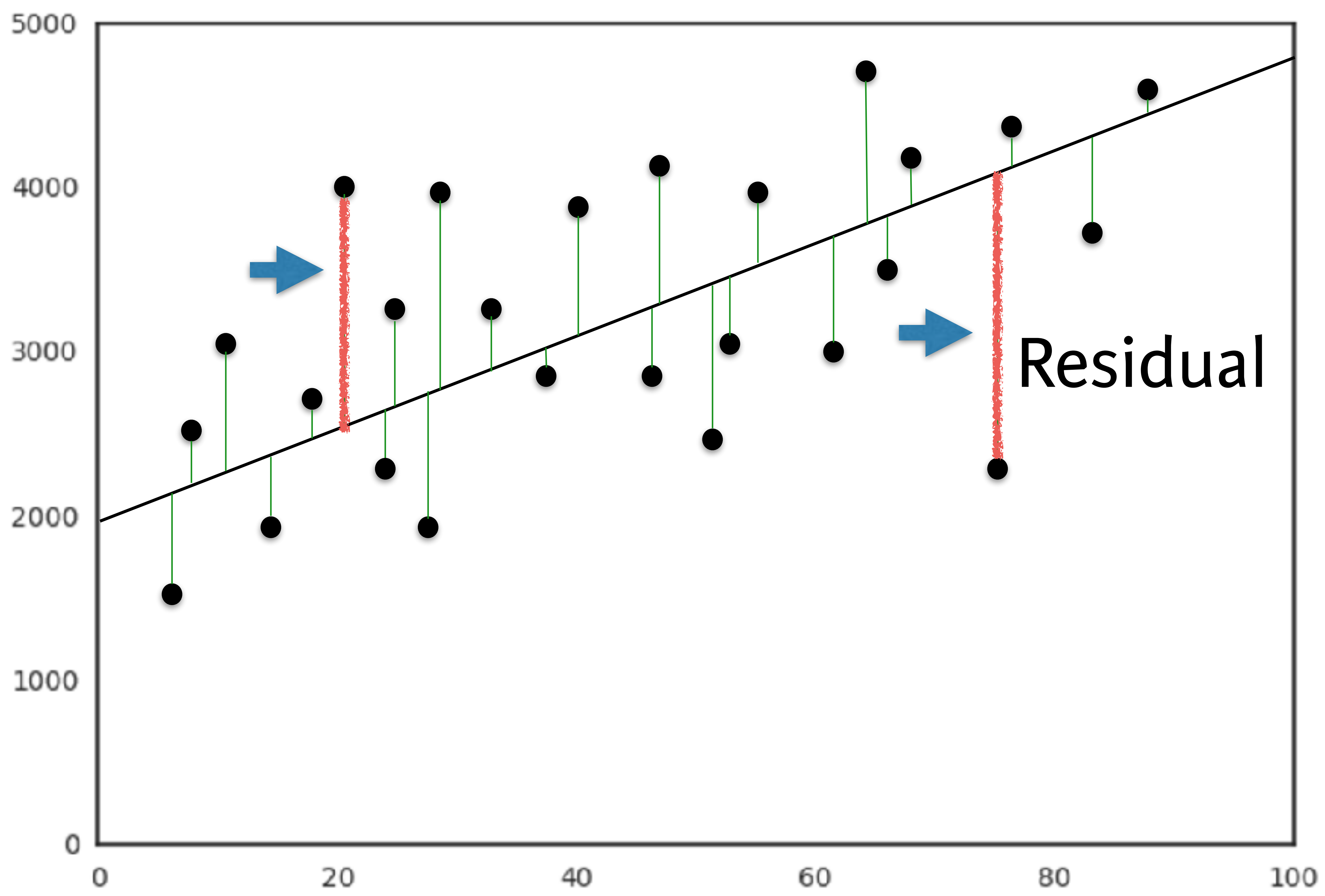
The basics of linear regression

Regression mechanics

- $y = ax + b$
  - y = target
  - x = single feature
  - a, b = parameters of model
- How do we choose a and b?
- Define an error function for any given line
  - Choose the line that minimizes the error function

The loss function

- Ordinary least squares (OLS): Minimize sum of squares of residuals



Linear regression in higher dimensions

$$y = a_1 x_1 + a_2 x_2 + b$$

- To fit a linear regression model here:
    - Need to specify 3 variables
- In higher dimensions:

$$y = a_1 x_1 + a_2 x_2 + a_3 x_3 + a_n x_n + b$$

- Must specify coefficient for each feature and the variable b
- Scikit-learn API works exactly the same way:
    - Pass two arrays: Features, and target

Linear regression on all features

```
In [1]: from sklearn.model_selection import train_test_split

In [2]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size = 0.3, random_state=42)

In [3]: reg_all = linear_model.LinearRegression()

In [4]: reg_all.fit(X_train, y_train)

In [5]: y_pred = reg_all.predict(X_test)

In [6]: reg_all.score(X_test, y_test)
Out[6]: 0.7112260057484926
```

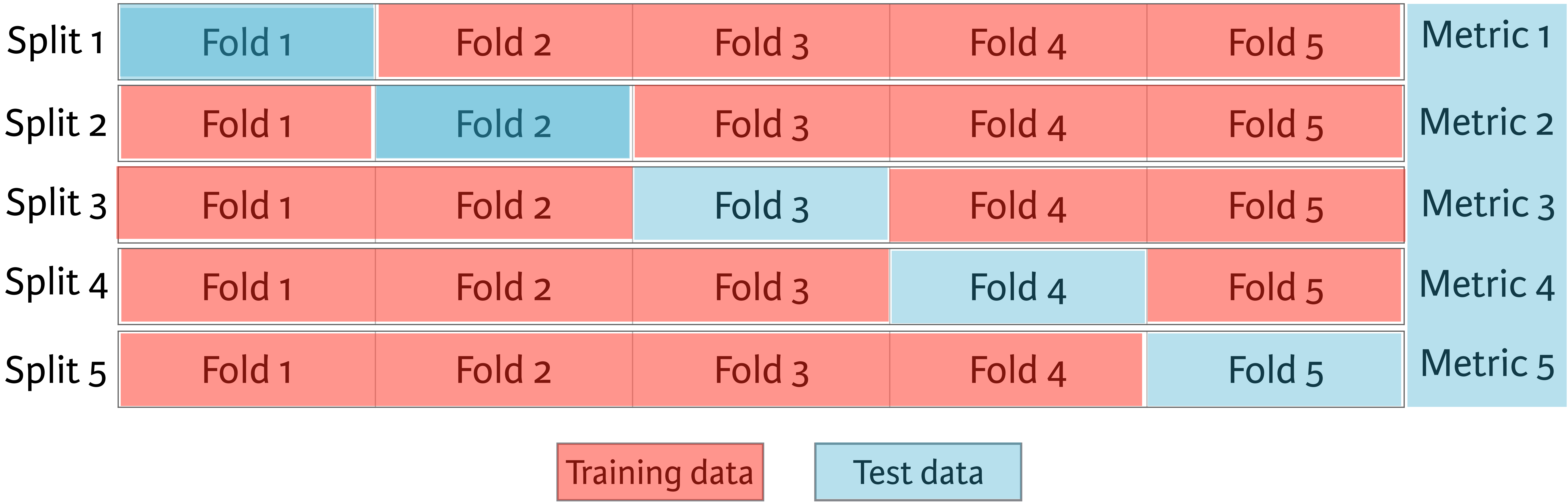
Let’s practice!

Cross-validation

Cross-validation motivation

- Model performance is dependent on way the data is split
- Not representative of the model's ability to generalize
- Solution: Cross-validation!

Cross-validation basics



Cross-validation and model performance

- 5 folds = 5-fold CV
- 10 folds = 10-fold CV
- k folds = k-fold CV
- More folds = More computationally expensive

Cross-validation in scikit-learn

```
In [1]: from sklearn.model_selection import cross_val_score

In [2]: reg = linear_model.LinearRegression()

In [3]: cv_results = cross_val_score(reg, X, y, cv=5)

In [4]: print(cv_results)
[ 0.63919994  0.71386698  0.58702344  0.07923081 -0.25294154]

In [5]: np.mean(cv_results)
Out[5]: 0.35327592439587058
```

Let's practice!

Regularized regression

Why regularize?

- Recall: Linear regression minimizes a loss function
- It chooses a coefficient for each feature variable
- Large coefficients can lead to overfitting
- Penalizing large coefficients: Regularization

Ridge regression

- Loss function = OLS loss function + $\alpha * \sum_{i=1}^{n} a_i^2$
- Alpha: Parameter we need to choose
- Picking alpha here is similar to picking k in k-NN
- Hyperparameter tuning (More in Chapter 3)
- Alpha controls model complexity
  - Alpha = 0: We get back OLS (Can lead to overfitting)
  - Very high alpha: Can lead to underfitting

Ridge regression in scikit-learn

```
In [1]: from sklearn.linear_model import Ridge

In [2]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size = 0.3, random_state=42)

In [3]: ridge = Ridge(alpha=0.1, normalize=True)

In [4]: ridge.fit(X_train, y_train)

In [5]: ridge_pred = ridge.predict(X_test)

In [6]: ridge.score(X_test, y_test)
Out[6]: 0.69969382751273179
```

Lasso regression

- Loss function = OLS loss function + $\alpha * \sum_{i=1}^{n} |a_i|$

Lasso regression in scikit-learn

```
In [1]: from sklearn.linear_model import Lasso

In [2]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size = 0.3, random_state=42)

In [3]: lasso = Lasso(alpha=0.1, normalize=True)

In [4]: lasso.fit(X_train, y_train)

In [5]: lasso_pred = lasso.predict(X_test)

In [6]: lasso.score(X_test, y_test)
Out[6]: 0.5950229535328551
```

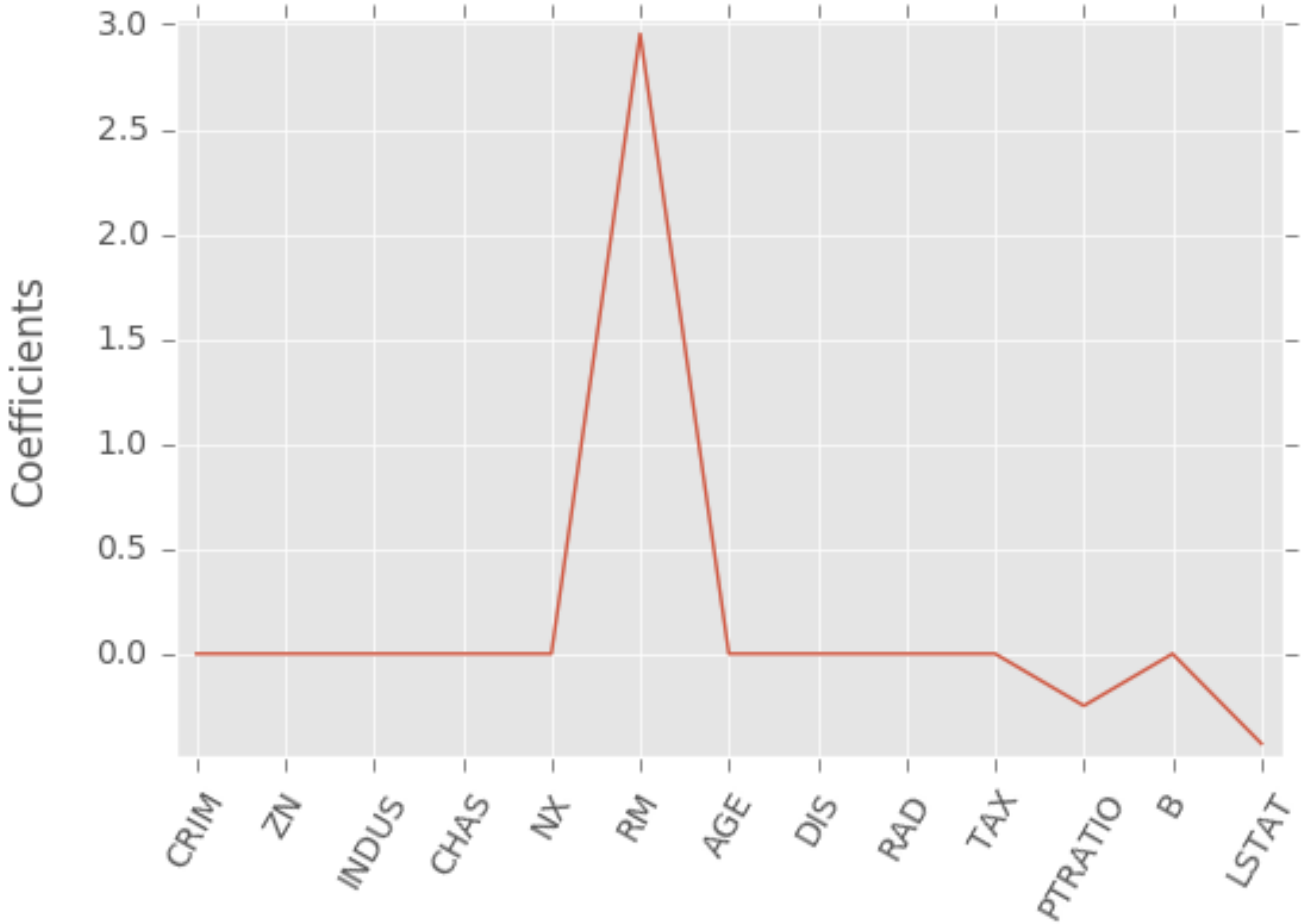
Lasso regression for feature selection

- Can be used to select important features of a dataset
- Shrinks the coefficients of less important features to exactly 0

Lasso for feature selection in scikit-learn

```
In [1]: from sklearn.linear_model import Lasso

In [2]: names = boston.drop('MEDV', axis=1).columns

In [3]: lasso = Lasso(alpha=0.1)

In [4]: lasso_coef = lasso.fit(X, y).coef_

In [5]: _ = plt.plot(range(len(names)), lasso_coef)

In [6]: _ = plt.xticks(range(len(names)), names, rotation=60)

In [7]: _ = plt.ylabel('Coefficients')

In [8]: plt.show()
```

Lasso for feature selection in scikit-learn



Let's practice!