Machine Learning with PyTorch and Scikit-Learn

-- Code Examples

Package version checks

Add folder to path in order to load from the check_packages.py script:

In [1]:
```
import sys
sys.path.insert(0, '..')
```

這段程式碼將當前目錄的父目錄（即上一層目錄）添加到 Python 模組的搜尋路徑中。這樣做的目的是使 Python 解釋器能夠找到位於父目錄中的自定義模組或套件，從而可以在當前工作目錄下引用它們。

通常情況下，這樣做是因為當前工作目錄不在 Python 模組的搜尋路徑中，但我們需要引用父目錄中的自定義模組或套件。這樣的情況可能發生在組織較大的專案或多層次的目錄結構中。

Check recommended package versions:

In [2]:
```
from python_environment_check import check_packages


d = {
    'numpy': '1.21.2',
    'mlxtend': '0.19.0',
    'matplotlib': '3.4.3',
    'sklearn': '1.0',
    'pandas': '1.3.2',
}
check_packages(d)
```

```
[OK] Your Python version is 3.9.7 | packaged by conda-forge | (default, Sep 29 2021, 19:24:02)
[Clang 11.1.0 ]
[OK] numpy 1.22.1
[OK] mlxtend 0.19.0
[OK] matplotlib 3.5.1
[OK] sklearn 1.0.2
[OK] pandas 1.4.0
```

這段程式碼主要是用來檢查特定Python套件的版本是否符合指定的要求。具體來說，它引入了一個自定義的 `check_packages` 函式，該函式會接收一個字典 `d` 作為參數，字典中包含了需要檢查的套件名稱和版本號。程式碼會檢查每個套件是否已經安裝，並且檢查其版本是否符合字典中指定的版本號。

在這個特定的例子中， `d` 字典指定了以下套件和版本：

- `numpy` 版本需為 `1.21.2`
- `mlxtend` 版本需為 `0.19.0`
- `matplotlib` 版本需為 `3.4.3`
- `sklearn` （即Scikit-learn）版本需為 `1.0`
- `pandas` 版本需為 `1.3.2`

函式會遍歷字典中的每個套件，檢查其版本是否符合要求。如果符合，則顯示訊息指出套件版本正確；如果不符合，則顯示相應的錯誤訊息或警告。這有助於確保你的環境中的套件版本是符合預期的，以確保程式碼的正確運作。

Chapter 09 - Predicting Continuous Target Variables with Regression Analysis

Overview

- Introducing regression
  - Simple linear regression
- Exploring the Ames Housing Dataset
  - Loading the Ames Housing dataset into a data frame
  - Visualizing the important characteristics of a dataset

- Implementing an ordinary least squares linear regression model
  - Solving regression for regression parameters with gradient descent
  - Estimating the coefficient of a regression model via scikit-learn
- 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
  - Modeling nonlinear relationships in the Ames Housing dataset
  - Dealing with nonlinear relationships using random forests
    - Decision tree regression
    - Random forest regression
- Summary

In [3]:
```
from IPython.display import Image
%matplotlib inline
```

這兩行程式碼是用於在Jupyter Notebook中顯示圖片並將Matplotlib圖形內嵌到Notebook中的。讓我來解釋一下每行的作用：

1. `from IPython.display import Image`：
   - 這個語句從 `IPython.display` 模組中導入了 `Image` 類。 `IPython.display` 模組提供了在Jupyter Notebook中顯示各種內容的功能，包括圖片、音頻、影片等。
   - `Image` 類可以用於顯示圖片文件，例如PNG、JPEG等格式的圖片。
2. `%matplotlib inline`：
   - 這是一個魔術命令（Magic Command），用於設置Matplotlib庫的後端，將Matplotlib圖形直接嵌入到Jupyter Notebook中，而不是打開一個新的窗口顯示。
   - `%matplotlib inline` 命令會將圖形嵌入到Notebook的輸出中，這樣你可以直接在Notebook中查看和操作圖形，而不需要額外的操作。

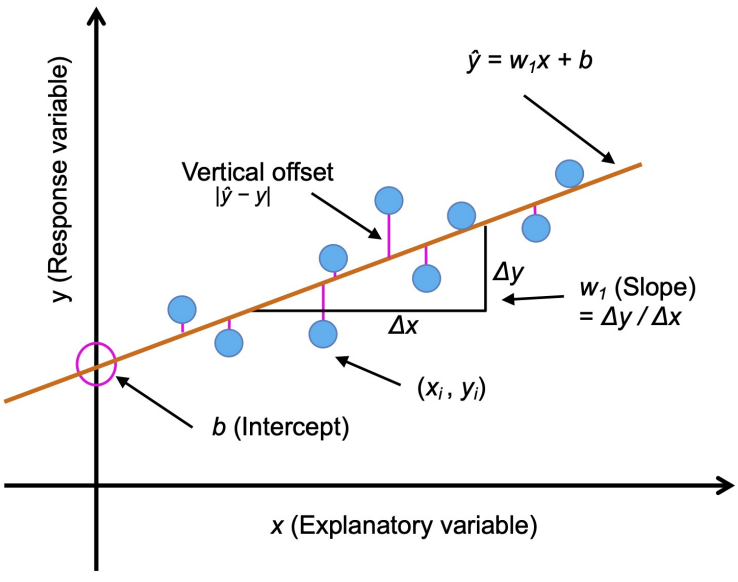
綜合起來，這兩行程式碼確保了在Jupyter Notebook中可以方便地顯示圖片並將Matplotlib圖形直接嵌入到Notebook的輸出中，使得數據分析和可視化更加便捷和直觀。

Introducing linear regression

Simple linear regression

In [4]:
```
Image(filename='figures/09_01.png', width=500)
```

Out[4]:

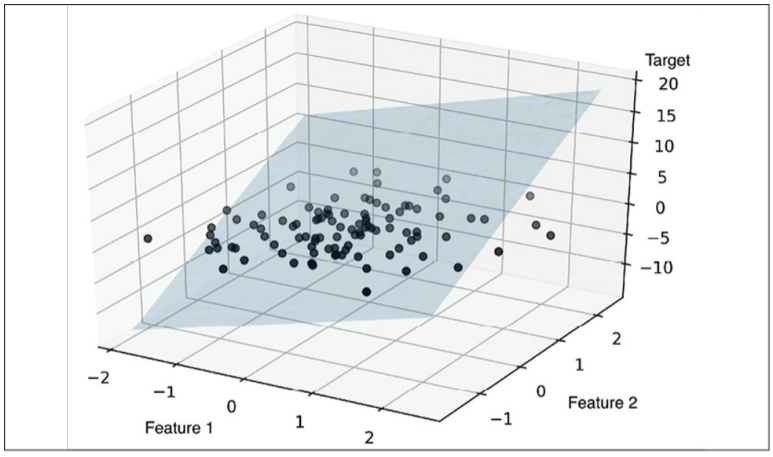


這段代碼是用來顯示一個圖片。這裡的 `Image` 是從 `IPython.display` 中引入的類別，用於在IPython環境中顯示圖片。 `filename` 參數指定了圖片的文件路徑，這裡假設圖片文件名為 'figures/09_01.png'。 `width` 參數則指定了圖片的顯示寬度為 500 像素。

Multiple linear regression

In [5]:
```
Image(filename='figures/09_01_2.png', width=500)
```

Out[5]:



這段代碼使用了IPython的`Image`類，該類用於在Jupyter Notebook中顯示圖像。具體來說：

- `Image(filename='figures/09_01_2.png', width=500)`：這行代碼創建了一個`Image`對象，指定了要顯示的圖像文件名為`figures/09_01_2.png`，並設置顯示寬度為500像素。

這行代碼的作用是在Jupyter Notebook中顯示一張圖像，但需要注意的是，圖像文件`figures/09_01_2.png`必須存在於指定的路徑中，否則將無法正確顯示。

Exploring the Ames Housing dataset

Loading the Ames Housing dataset into a data frame

- Dataset source: http://jse.amstat.org/v19n3/decock/AmesHousing.txt
- Dataset documentation: http://jse.amstat.org/v19n3/decock/DataDocumentation.txt
- Dataset write-up: http://jse.amstat.org/v19n3/decock.pdf

- `'Overall Qual'`: Rates the overall material and finish of the house

```
10      Very Excellent
9       Excellent
8       Very Good
7       Good
6       Above Average
5       Average
4       Below Average
3       Fair
2       Poor
1       Very Poor
```

- `'Overall Cond'`: Rates the overall condition of the house

```
10      Very Excellent
9       Excellent
8       Very Good
7       Good
6       Above Average
5       Average
4       Below Average
3       Fair
2       Poor
1       Very Poor
```

- `'Gr Liv Area'`: Above grade (ground) living area square feet
- `'Central Air'`: Central air conditioning

```
N       No
Y       Yes
```

- `'Total Bsmt SF'`: Total square feet of basement area
- `'SalePrice'`: Sale price $$

In [6]:
```
import pandas as pd


columns = ['Overall Qual', 'Overall Cond', 'Gr Liv Area',
           'Central Air', 'Total Bsmt SF', 'SalePrice']

df = pd.read_csv('http://jse.amstat.org/v19n3/decock/AmesHousing.txt',
                 sep='\t',
                 usecols=columns)

df.head()
```

Out[6]:

	Overall Qual	Overall Cond	Total Bsmt SF	Central Air	Gr Liv Area	SalePrice
0	6	5	1080.0	Y	1656	215000
1	5	6	882.0	Y	896	105000
2	6	6	1329.0	Y	1329	172000
3	7	5	2110.0	Y	2110	244000
4	5	5	928.0	Y	1629	189900

這段程式碼的目的是從指定的URL（http://jse.amstat.org/v19n3/decock/AmesHousing.txt%EF%BC%89%E8%AE%80%E5%8F%96%E6%88%BF%E5%B1%8B%E6%95%B8 Qual', 'Overall Cond', 'Gr Liv Area', 'Central Air', 'Total Bsmt SF', 'SalePrice'）。然後，使用Pandas庫的 `read_csv` 函數將這些數據讀取到DataFrame對象中，並顯示前幾行數據。

- `pd.read_csv('http://jse.amstat.org/v19n3/decock/AmesHousing.txt', sep='\t', usecols=columns)`：此函數調用將指定URL中的數據讀取到DataFrame中。 `sep='\t'` 表示使用制表符作為字段分隔符。 `usecols=columns` 指定只讀取 `columns` 列表中列出的列。
- `df.head()`：顯示DataFrame `df` 的前幾行數據，默認顯示前5行。

In [7]:
```
df.shape
```

Out[7]:
```
(2930, 6)
```

這段程式碼執行後，會顯示DataFrame `df` 的形狀（shape），即行數和列數。

解釋：

- `df.shape`：返回一個元組，其中第一個元素是DataFrame的行數，第二個元素是列數。例如，如果這段程式碼執行後返回 `(2930, 6)`，則表示DataFrame `df` 包含2930行和6列的數據。

In [8]:
```
df['Central Air'] = df['Central Air'].map({'N': 0, 'Y': 1})
```

這行程式碼的作用是將 DataFrame 中的 `'Central Air'` 列中的值從字母形式（'N' 和 'Y'）映射為數字形式（0 和 1）。

解釋：

- `df['Central Air']`：選擇 DataFrame `df` 中的 `'Central Air'` 列。
- `.map({'N': 0, 'Y': 1})`：使用字典將列中的值進行映射。具體來說，'N' 將被映射為 0，'Y' 將被映射為 1。這樣做的目的通常是將具有類別性質的文本數據轉換為可以在模型中使用的數值形式，例如將二元類別變量轉換為二進制數據。

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Overall Qual     0
Overall Cond     0
Total Bsmt SF    1
Central Air      0
Gr Liv Area      0
SalePrice        0
dtype: int64
```

這行程式碼用於計算 DataFrame `df` 中每個列的空值數量。

解釋：

- `df.isnull()`：這會生成一個與 `df` 具有相同形狀的布林值 DataFrame，其中每個元素如果是空值則為 `True`，否則為 `False`。
- `.sum()`：對每一列進行求和操作，因為布林值會被隱式轉換為整數（`True` 變成 1，`False` 變成 0），所以求和後，結果就是每列中空值的總數。

結果將是一個顯示每列空值總數的 Series。

In [10]:
```
# remove rows that contain missing values

df = df.dropna(axis=0)
df.isnull().sum()
```

Out[10]:
```
Overall Qual     0
Overall Cond     0
Total Bsmt SF    0
Central Air      0
Gr Liv Area      0
SalePrice        0
dtype: int64
```

這段程式碼用於從 DataFrame `df` 中刪除包含缺失值的行，並檢查刪除後是否還有空值。

解釋：

1. `df.dropna(axis=0)`：這會從 DataFrame 中刪除包含任何空值的行。參數 `axis=0` 表示操作將應用於行，也就是對行進行操作。
2. `df.isnull().sum()`：這段程式碼用來檢查刪除操作後的 DataFrame `df` 中是否還有空值。如果 `isnull()` 返回的 DataFrame 中所有元素的總和為零，則意味著 DataFrame 中已經沒有空值。

如果執行後的結果顯示每個欄位的空值總數都是零，那麼 `df` 中就已經沒有空值了。

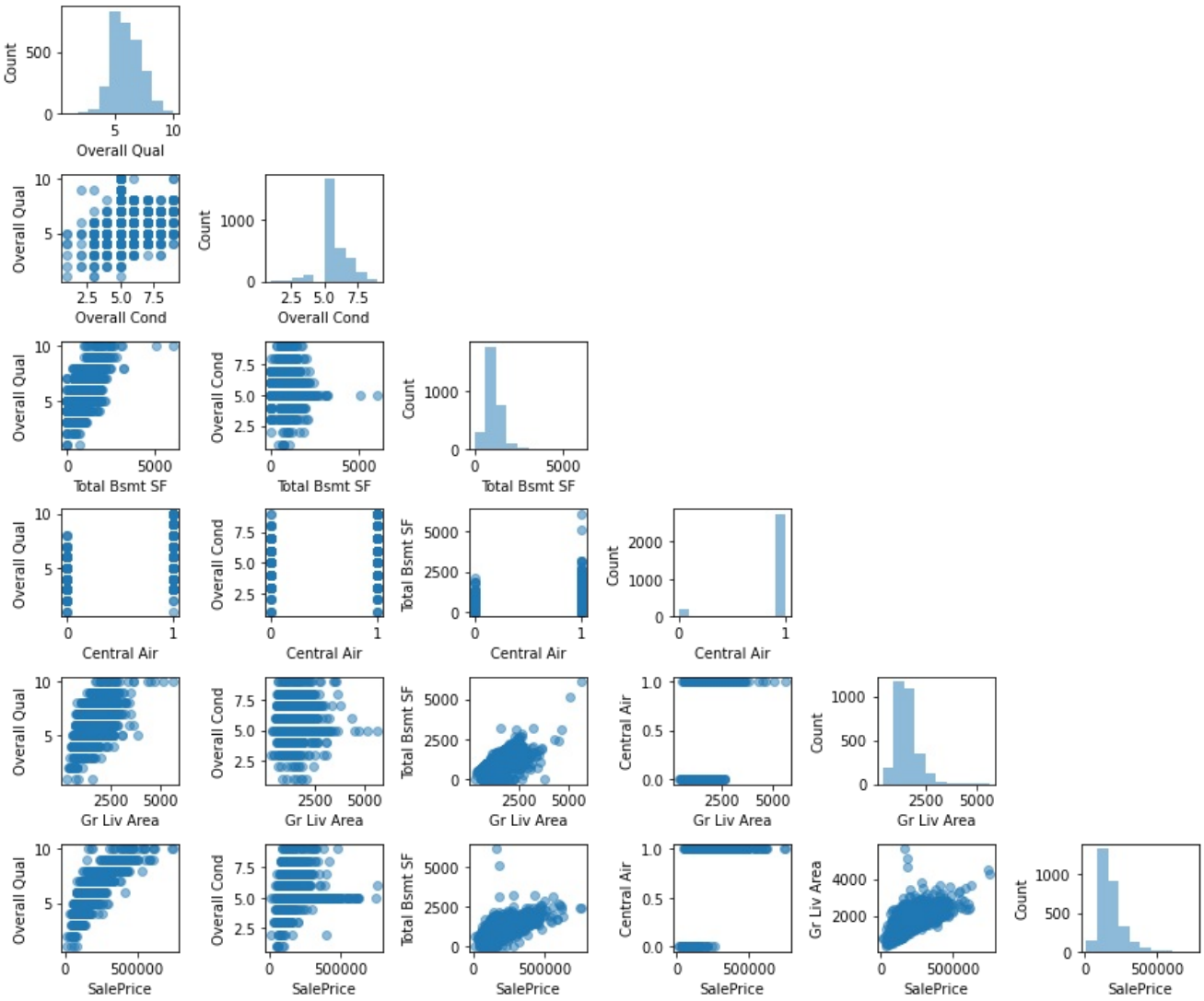
Visualizing the important characteristics of a dataset

In [11]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import scatterplotmatrix
```

這段程式碼的目的是從 `mlxtend.plotting` 模組中導入 `scatterplotmatrix` 函式，並導入 `matplotlib.pyplot` 模組中的 `plt` 別名。

- `matplotlib.pyplot` 是 Python 中流行的繪圖庫 Matplotlib 的子模組，用於創建各種類型的圖表和圖形。
- `mlxtend.plotting.scatterplotmatrix` 是 `mlxtend` 庫中用於繪製散點圖矩陣的函式。散點圖矩陣通常用於展示多個變數之間的相關性和分佈情況。

In [12]:
```
scatterplotmatrix(df.values, figsize=(12, 10),
                  names=df.columns, alpha=0.5)
plt.tight_layout()
#plt.savefig('figures/09_04.png', dpi=300)
plt.show()
```

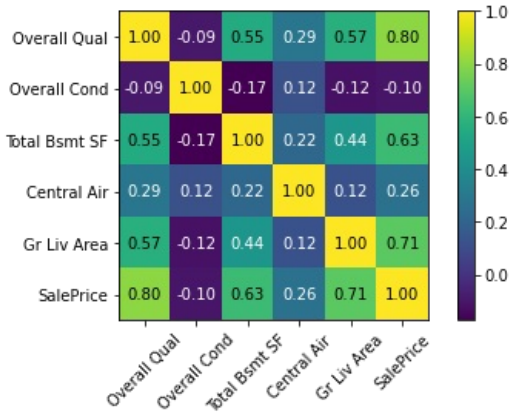


這段程式碼會繪製一個散點圖矩陣，展示資料集 `df` 中各個變數之間的相關性和分佈情況。讓我們逐步解釋：

- `scatterplotmatrix(df.values, figsize=(12, 10), names=df.columns, alpha=0.5)`：
  - `scatterplotmatrix` 函式用來繪製散點圖矩陣。
  - `df.values` 取得 DataFrame `df` 的所有資料作為繪圖資料。
  - `figsize=(12, 10)` 設置圖形的尺寸為寬度 12 單位、高度 10 單位。
  - `names=df.columns` 將 DataFrame 的欄位名稱用作散點圖矩陣中每個子圖的標題。
  - `alpha=0.5` 設置點的透明度為 0.5，使得點的顏色較淡。
- `plt.tight_layout()`：自動調整子圖間距，以免重疊。
- `plt.show()`：顯示圖形。

最後兩行程式碼被註解掉了，這些程式碼通常用於保存圖形為圖片文件，但在這裡被註解掉了，因此不會執行。

In [13]:
```
import numpy as np
from mlxtend.plotting import heatmap


cm = np.corrcoef(df.values.T)
hm = heatmap(cm, row_names=df.columns, column_names=df.columns)

plt.tight_layout()
#plt.savefig('figures/09_05.png', dpi=300)
plt.show()
```



這段程式碼使用 `mlxtend` 庫中的 `heatmap` 函式來繪製相關係數矩陣的熱度圖，展示各個變數之間的相關性。讓我們逐步解釋：

- `cm = np.corrcoef(df.values.T)`：
  - `np.corrcoef` 函式計算了資料集 `df` 的轉置版本（`.T`）的相關係數矩陣。
  - `cm` 是計算得到的相關係數矩陣。
- `hm = heatmap(cm, row_names=df.columns, column_names=df.columns)`：
  - `heatmap` 函式用來繪製熱度圖。
  - `cm` 是相關係數矩陣。
  - `row_names=df.columns, column_names=df.columns` 設置熱度圖的行和列的名稱為 DataFrame `df` 的欄位名稱。
- `plt.tight_layout()`：自動調整子圖間距，以免重疊。
- `plt.show()`：顯示繪製的熱度圖。

最後兩行程式碼被註解掉了，這些程式碼通常用於保存圖形為圖片文件，但在這裡被註解掉了，因此不會執行。

Implementing an ordinary least squares linear regression model

...

Solving regression for regression parameters with gradient descent

In [14]:
```
class LinearRegressionGD:
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01, size=X.shape[1])
        self.b_ = np.array([0.])
        self.losses_ = []

        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_ += self.eta * 2.0 * X.T.dot(errors) / X.shape[0]
            self.b_ += self.eta * 2.0 * errors.mean()
            loss = (errors**2).mean()
            self.losses_.append(loss)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_) + self.b_

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

這段程式碼定義了一個使用梯度下降法（Gradient Descent）訓練的線性回歸模型 `LinearRegressionGD`。讓我們逐步解釋每個部分的功能：

1. **`__init__` 方法：**
   - 初始化模型的學習率 `eta`、迭代次數 `n_iter` 和隨機種子 `random_state`。

- `eta` 控制每次更新權重和偏差的步長。
- `n_iter` 是訓練過程中的迭代次數。
- `random_state` 是用於初始化隨機數生成器的種子，確保每次運行結果一致。

2. **`fit` 方法：**
   - 接收特徵矩陣 `X` 和目標向量 `y`，並進行模型訓練。
   - 使用隨機數生成器 `rgen` 初始化權重 `self.w_` 為小數值的隨機向量。
   - 初始化偏差 `self.b_` 為零。
   - 儲存每次迭代的損失值於 `self.losses_` 中。
   - 迭代 `n_iter` 次，計算模型的輸出、誤差、更新權重和偏差，計算並儲存損失。
3. **`net_input` 方法：**
   - 計算輸入特徵 `X` 的預測結果，使用權重 `self.w_` 和偏差 `self.b_` 進行加權線性組合。
4. **`predict` 方法：**
   - 接收特徵矩陣 `X`，通過 `net_input` 方法計算並返回預測的目標向量。

在 `fit` 方法中，主要的計算是使用梯度下降來最小化均方誤差（MSE）。每次迭代時，計算輸出與真實值的誤差，根據誤差的梯度更新權重 `self.w_` 和偏差 `self.b_`。這樣模型就可以逐步提升，使得預測越來越準確。

這是一個簡單但有效的線性回歸實現方式，通過設置不同的 `eta` 和 `n_iter` 可以調整模型的學習速度和準確性。

In [15]:
```
X = df[['Gr Liv Area']].values
y = df['SalePrice'].values
```

這段代碼從 DataFrame `df` 中選擇了名為 `'Gr Liv Area'` 的單個特徵作為自變量 `X`，並將名為 `'SalePrice'` 的列作為因變量 `y`。

具體來說：

- `df[['Gr Liv Area']]`：使用 DataFrame `df` 的索引操作，選取 `'Gr Liv Area'` 這一列。這裡使用雙重方括號 `[['Gr Liv Area']]` 是為了確保 `X` 是一個二維數組（或矩陣），而不是一維數組。
- `.values`：將選取的 DataFrame 列轉換為 NumPy 數組。這麼做是為了能夠在後續的數學計算中使用這些數據，因為許多機器學習算法要求輸入是 NumPy 數組形式。

因此，`X` 現在是一個二維數組，其中每行包含一個樣本的 `'Gr Liv Area'` 特徵值。而 `y` 是一維數組，包含了對應每個樣本的 `'SalePrice'` 值，這是我們希望預測的目標值或因變量。

In [16]:
```
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()
```

這段程式碼使用了 `StandardScaler` 類別從 `sklearn.preprocessing` 模組，用來對特徵 `X` 和目標 `y` 進行標準化處理。

具體來說：

1. **特徵 `X` 的標準化：**

   `X_std = sc_x.fit_transform(X)`
   - `sc_x` 是一個 `StandardScaler` 物件，用於對特徵進行標準化。
   - `sc_x.fit_transform(X)` 調用了 `StandardScaler` 物件的 `fit_transform` 方法，該方法首先計算 `X` 中每個特徵的平均值和標準差，然後將 `X` 中的每個特徵獨立地進行標準化處理。標準化後的數據存儲在 `X_std` 中，它是一個 NumPy 陣列。
2. **目標 `y` 的標準化：**

   `y_std = sc_y.fit_transform(y[:, np.newaxis]).flatten()`
   - `sc_y` 是另一個 `StandardScaler` 物件，用於對目標進行標準化。
   - `y[:, np.newaxis]` 將 `y` 轉換為列向量形式，這是因為 `StandardScaler` 的 `fit_transform` 方法要求輸入是二維數組。
   - `sc_y.fit_transform(...)` 調用了 `StandardScaler` 物件的 `fit_transform` 方法，同樣計算了 `y` 的平均值和標準差，並將 `y` 標準化處理後的結果存儲在 `y_std` 中。`.flatten()` 方法將結果展平為一維數組。

總結來說，這段程式碼使用 `StandardScaler` 將 `X` 和 `y` 分別進行標準化處理，使得特徵和目標的數值範圍在平均值為0，標準差為1的標準正態分佈內。這是許多機器學習算法的常見前置步驟，可以改善模型的收斂速度和預測性能。

In [17]:
```
lr = LinearRegressionGD(eta=0.1)
lr.fit(X_std, y_std)
```

Out[17]: <__main__.LinearRegressionGD at 0x13a60faf0>

這段程式碼建立了一個 `LinearRegressionGD` 的物件 `lr`，並使用標準化後的特徵 `X_std` 和目標 `y_std` 來訓練該模型。

具體來說：

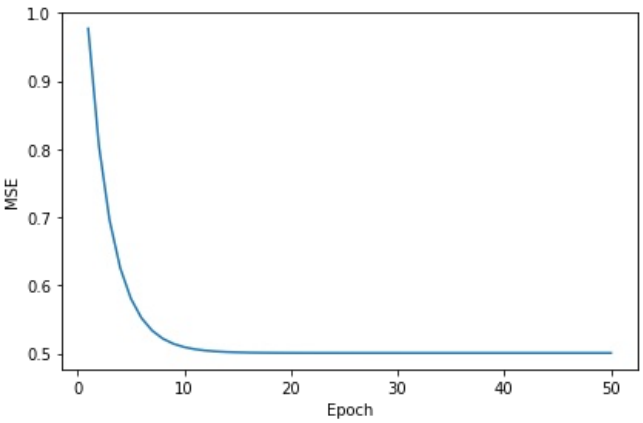
1. **建立 `LinearRegressionGD` 物件：**

   ```
   lr = LinearRegressionGD(eta=0.1)
   ```
   - 創建了一個 `LinearRegressionGD` 的物件 `lr`，設置了學習速率 `eta` 為 `0.1`。
   - `LinearRegressionGD` 是一個自訂的線性回歸模型，它使用梯度下降算法來最小化均方誤差。
2. **訓練模型：**

   ```
   lr.fit(X_std, y_std)
   ```
   - 調用 `lr` 物件的 `fit` 方法，傳入標準化後的特徵 `X_std` 和目標 `y_std` 進行模型訓練。
   - 在 `fit` 方法內部，模型使用梯度下降算法進行迭代更新，以學習適合於給定數據的權重 `w_` 和偏差 `b_`。

這段程式碼的目的是使用自訂的梯度下降線性回歸模型（`LinearRegressionGD`）對標準化後的數據進行訓練，以擬合出適合的線性關係模型，使得模型能夠對未見過的數據進行準確預測。

In [18]:
```
plt.plot(range(1, lr.n_iter+1), lr.losses_)
plt.ylabel('MSE')
plt.xlabel('Epoch')
plt.tight_layout()
#plt.savefig('figures/09_06.png', dpi=300)
plt.show()
```



這段程式碼使用 Matplotlib 繪製了梯度下降過程中每個 epoch 的均方誤差（MSE）隨著 epoch 數量變化的折線圖。

具體來說：

1. **繪製折線圖：**

   ```
   plt.plot(range(1, lr.n_iter+1), lr.losses_)
   ```
   - `plt.plot` 函數用於繪製折線圖，其中 `range(1, lr.n_iter+1)` 提供了 x 軸的數據（即 epoch 數量），`lr.losses_` 提供了 y 軸的數據（即每個 epoch 的 MSE）。
2. **設置標籤和標題：**

   ```
   plt.ylabel('MSE')
   plt.xlabel('Epoch')
   ```
   - `plt.ylabel` 設置 y 軸的標籤為 'MSE'，表示均方誤差。
   - `plt.xlabel` 設置 x 軸的標籤為 'Epoch'，表示迭代的輪次或 epoch 數量。
3. **調整圖形佈局並顯示圖片：**

   ```
   plt.tight_layout()
   plt.show()
   ```
   - `plt.tight_layout()` 確保圖形元素不重疊，使得圖形更加美觀。
   - `plt.show()` 顯示圖形。

這段程式碼的目的是展示隨著模型訓練過程中 epoch 數量增加，均方誤差（MSE）如何變化的過程。通過這樣的折線圖，可以觀察到模型在訓練過程中學習的效果，以及是否達到了收斂。

In [19]:
```
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
```

這個函式 `lin_regplot` 用於繪製線性回歸模型的散點圖和預測直線。以下是對函式內部每個步驟的解釋：

1. **散點圖：**

```
plt.scatter(X, y, c='steelblue', edgecolor='white', s=70)
```

- `plt.scatter` 函式用於繪製散點圖，其中 `X` 和 `y` 分別是特徵和目標變數的數據。 `c='steelblue'` 設置散點的顏色為鋼藍色， `edgecolor='white'` 設置散點的邊緣顏色為白色， `s=70` 設置散點的大小為 70 像素。

2. **預測直線：**

```
plt.plot(X, model.predict(X), color='black', lw=2)
```

- `plt.plot` 函式用於繪製直線， `X` 是特徵變數的數據， `model.predict(X)` 是使用線性回歸模型 `model` 對 `X` 進行預測後得到的目標變數的預測值。 `color='black'` 設置直線的顏色為黑色， `lw=2` 設置直線的線寬為 2。

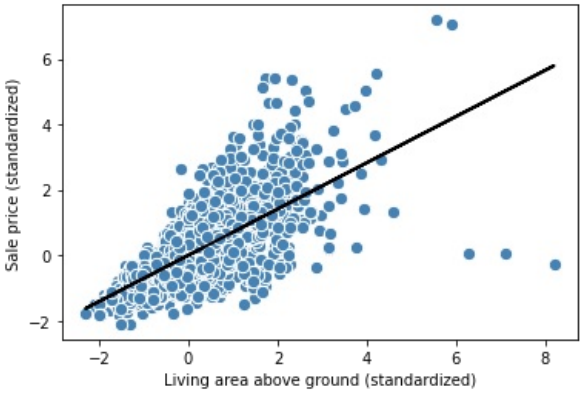
3. **返回值：**

```
return
```

- 函式沒有明確的返回值，這意味著它主要用於在圖中繪製散點圖和預測直線，而不是生成某個數據結構或計算結果。

這個函式的用途是快速地將數據集的散點圖和線性回歸模型的預測直線一起顯示出來，以便直觀地了解模型如何擬合數據。

In [20]:
```
lin_regplot(X_std, y_std, lr)
plt.xlabel('Living area above ground (standardized)')
plt.ylabel('Sale price (standardized)')

#plt.savefig('figures/09_07.png', dpi=300)
plt.show()
```



這段程式碼是繪製一個散點圖和回歸線，用來顯示經過標準化後的房屋地上面積（X_std）與銷售價格（y_std）之間的關係，並且將訓練好的線性回歸模型（lr）的預測結果添加到圖中。

具體解釋如下：

- `plt.scatter(X, y, c='steelblue', edgecolor='white', s=70)`：繪製散點圖，X 軸是標準化後的房屋地上面積，Y 軸是標準化後的銷售價格。這裡使用了鋼藍色作為點的顏色，白色作為邊緣顏色，點的大小為 70。
- `plt.plot(X, model.predict(X), color='black', lw=2)`：繪製回歸線，X 軸是標準化後的房屋地上面積，Y 軸是使用模型 `lr` 預測的銷售價格。這裡設置了線的顏色為黑色，線寬為 2。
- `plt.xlabel('Living area above ground (standardized)')`：設置 x 軸的標籤為 "Living area above ground (standardized)"，表示房屋地上面積的標準化數據。
- `plt.ylabel('Sale price (standardized)')`：設置 y 軸的標籤為 "Sale price (standardized)"，表示銷售價格的標準化數據。
- `plt.show()`：顯示圖表。

這段程式碼的目的是展示經過線性回歸模型擬合後的預測結果，以及觀察標準化後的特徵與目標變量之間的關係趨勢。

In [21]:
```
feature_std = sc_x.transform(np.array([[2500]]))
target_std = lr.predict(feature_std)
target_reverted = sc_y.inverse_transform(target_std.reshape(-1, 1))
print(f'Sale price: ${target_reverted.flatten()[0]:.2f}')
```

```
Sale price: $292507.07
```

這段程式碼的目的是將標準化後的特徵值（房屋面積，2500平方英尺）轉換為預測的房價。這裡是如何達成這個目標的：

1. `feature_std = sc_x.transform(np.array([[2500]]))`：使用 `StandardScaler` 對2500平方英尺的房屋面積進行標準化。 `sc_x.transform()` 方法將2500轉換為標準化後的值，以便能夠在模型中進行預測。
2. `target_std = lr.predict(feature_std)`：使用訓練好的線性回歸模型 `lr` 來預測標準化後的目標變量（房價）。 `lr.predict()` 方法將標準化後的特徵值 `feature_std` 作為輸入，返回預測的標準化房價。
3. `target_reverted = sc_y.inverse_transform(target_std.reshape(-1, 1))`：將預測的標準化房價逆轉換為原始的房價。 `sc_y.inverse_transform()` 方法將標準化後的房價 `target_std` 轉換回原始數據空間，使其恢復為原始的美元表示。

4. `print(f'Sale price: ${target_reverted.flatten()[0]:.2f}')`：最後，將恢復後的房價以美元格式化輸出。這樣可以顯示預測的房價，保留小數點後兩位，並添加“Sale price: $”前綴以表明其含義。

In [22]:
```
print(f'Slope: {lr.w_[0]:.3f}')
print(f'Intercept: {lr.b_[0]:.3f}')
```

```
Slope: 0.707
Intercept: -0.000
```

這段程式碼用於印出線性回歸模型的斜率和截距。以下是對這段程式碼的解釋：

1. `print(f'Slope: {lr.w_[0]:.3f}')`：
   - `lr.w_[0]`：這是線性回歸模型在訓練過程中學習到的斜率（或權重）。
   - `: .3f`：這部分格式化字串表示要顯示的數字格式。在這裡，`.3f` 表示顯示到小數點後三位。

   所以，這行程式碼會印出線性回歸模型的斜率，並保留到小數點後三位。

2. `print(f'Intercept: {lr.b_[0]:.3f}')`：
   - `lr.b_[0]`：這是線性回歸模型在訓練過程中學習到的截距（或偏差）。
   - `: .3f`：同樣，這部分格式化字串表示要顯示的數字格式，將截距顯示到小數點後三位。

   這行程式碼會印出線性回歸模型的截距，同樣保留到小數點後三位。

這樣，這兩行程式碼一起印出了線性回歸模型的斜率和截距，提供了對模型訓練後的基本理解。

Estimating the coefficient of a regression model via scikit-learn

In [23]:
```
from sklearn.linear_model import LinearRegression
```

這行程式碼導入了 `sklearn` 中的 `LinearRegression` 線性回歸模型。線性回歸是一種用來探索變數之間線性關係的統計模型，它假設因變數（或目標變數）與自變數之間存在著線性關係。在機器學習中，線性回歸通常用於解決迴歸問題，即預測連續型變數的值。

這個模型在 `sklearn` 中的實作提供了一個方法來擬合數據，找到最適合的線性關係，使得模型預測的目標變數值與實際觀察到的值之間的誤差最小化。

In [24]:
```
slr = LinearRegression()
slr.fit(X, y)
y_pred = slr.predict(X)
print(f'Slope: {slr.coef_[0]:.3f}')
print(f'Intercept: {slr.intercept_:.3f}')
```

```
Slope: 111.666
Intercept: 13342.979
```

這段程式碼使用 `LinearRegression` 模型來擬合 `X` 和 `y`，其中 `X` 是特徵（這裡指的是 'Gr Liv Area'），`y` 是目標變量（即 'SalePrice'）。以下是程式碼的解釋：

1. **LinearRegression 實例化：**

   ```
   slr = LinearRegression()
   ```
   使用 `LinearRegression()` 創建了一個線性回歸模型的實例 `slr`。

2. **擬合模型：**

   ```
   slr.fit(X, y)
   ```
   通過調用 `fit(X, y)` 方法，將特徵 `X` 和目標變量 `y` 用於訓練模型。這個步驟會計算最適合數據的回歸線，以最小化觀測值和預測值之間的平方誤差。

3. **預測：**

   ```
   y_pred = slr.predict(X)
   ```
   使用 `predict(X)` 方法根據輸入的特徵 `X` 來預測目標變量 `y` 的值。

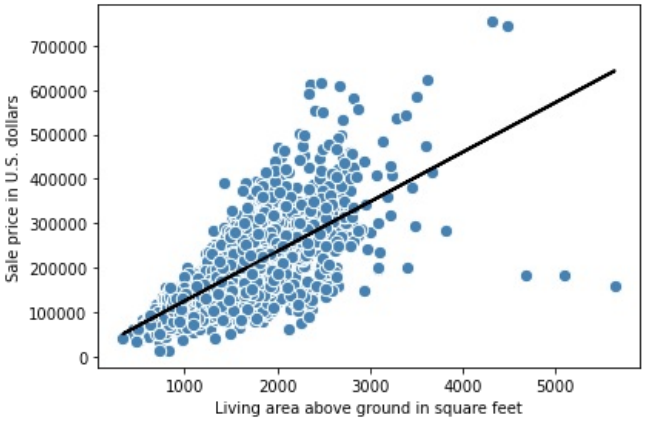
4. **輸出斜率和截距：**

   ```
   print(f'Slope: {slr.coef_[0]:.3f}')
   print(f'Intercept: {slr.intercept_:.3f}')
   ```
   - `slr.coef_[0]` 返回回歸模型的斜率。
   - `slr.intercept_` 返回回歸模型的截距。

這段程式碼的目的是利用線性回歸模型對 'Gr Liv Area' 和 'SalePrice' 之間的關係進行建模，並且顯示回歸線的斜率和截距。

In [ ]:
```

```

In [25]:
```
lin_regplot(X, y, slr)
plt.xlabel('Living area above ground in square feet')
plt.ylabel('Sale price in U.S. dollars')

plt.tight_layout()
#plt.savefig('figures/09_08.png', dpi=300)
plt.show()
```



這段程式碼主要用於繪製一個散點圖和線性回歸模型的擬合線，以下是每行的解釋：

- `lin_regplot(X, y, slr)`：這裡呼叫了 `lin_regplot` 函式，該函式接受三個參數：X（特徵值），y（目標值），以及 `LinearRegression` 模型 `slr`。`lin_regplot` 函式用來顯示散點圖和線性回歸模型的擬合結果。
- `plt.xlabel('Living area above ground in square feet')`：設置 x 軸的標籤為 "Living area above ground in square feet"，表示地面以上的生活面積（單位：平方英尺）。
- `plt.ylabel('Sale price in U.S. dollars')`：設置 y 軸的標籤為 "Sale price in U.S. dollars"，表示房屋售價（單位：美元）。
- `plt.tight_layout()`：該函式用於自動調整子圖或子軸的間距，以便更好地適應圖形區域。
- `plt.show()`：顯示圖形，如果在 Jupyter Notebook 中執行，則會直接顯示在輸出區域。

這段程式碼的主要作用是展示了線性回歸模型在特徵（Living area above ground）和目標（Sale price）之間的關係，並顯示了模型的擬合效果。

Normal Equations alternative:

In [26]:
```
# adding a column vector of "ones"
Xb = np.hstack((np.ones((X.shape[0], 1)), X))
w = np.zeros(X.shape[1])
z = np.linalg.inv(np.dot(Xb.T, Xb))
w = np.dot(z, np.dot(Xb.T, y))

print(f'Slope: {w[1]:.3f}')
print(f'Intercept: {w[0]:.3f}')
```

```
Slope: 111.666
Intercept: 13342.979
```

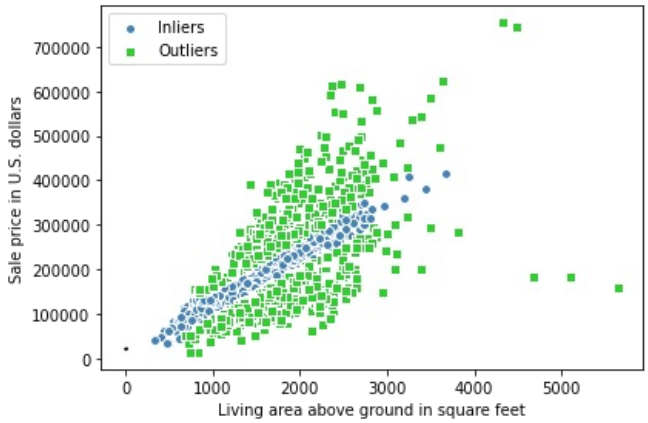
這段程式碼執行了多元線性回歸的計算過程，以下是每行的解釋：

- `Xb = np.hstack((np.ones((X.shape[0], 1)), X))`：在原始特徵矩陣 `X` 的左側添加一列全為1的列向量，這是為了處理截距項。`X.shape[0]` 是矩陣 `X` 的行數，`np.ones((X.shape[0], 1))` 創建一個形狀為 `(行數, 1)` 的全1矩陣，然後 `np.hstack` 函式將這個全1矩陣和原始特徵矩陣 `X` 水平拼接起來。
- `w = np.zeros(X.shape[1])`：創建一個形狀為 `(特徵數, )` 的全0向量 `w`，用來存儲回歸係數。
- `z = np.linalg.inv(np.dot(Xb.T, Xb))`：計算 `Xb` 的轉置矩陣 `Xb.T` 和 `Xb` 的內積，然後對結果進行求逆操作，得到矩陣 `Xb` 的逆矩陣 `z`。
- `w = np.dot(z, np.dot(Xb.T, y))`：用求得的逆矩陣 `z`，乘以 `Xb` 的轉置 `Xb.T` 和目標變量 `y` 的內積，最終計算出回歸係數 `w`。
- `print(f'Slope: {w[1]:.3f}')`：輸出多元線性回歸模型的斜率（係數），這裡 `w[1]` 是在 `X` 矩陣加上截距後的第二個元素，即第一個原始特徵的回歸係數。
- `print(f'Intercept: {w[0]:.3f}')`：輸出多元線性回歸模型的截距，這裡 `w[0]` 是在 `X` 矩陣加上截距後的第一個元素，即截距項的回歸係數。

這段程式碼展示了使用矩陣運算求解多元線性回歸模型的方法，包括添加截距項、計算回歸係數，並輸出模型的斜率和截距。

Fitting a robust regression model using RANSAC

In [27]:
```
from sklearn.linear_model import RANSACRegressor


ransac = RANSACRegressor(LinearRegression(),
                         max_trials=100, # default
                         min_samples=0.95,
                         loss='absolute_error', # default
                         residual_threshold=None, # default
                         random_state=123)


ransac.fit(X, y)

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('Living area above ground in square feet')
plt.ylabel('Sale price in U.S. dollars')
plt.legend(loc='upper left')

plt.tight_layout()
#plt.savefig('figures/09_09.png', dpi=300)
plt.show()
```



這段程式碼使用了RANSAC（Random Sample Consensus）回歸模型來估計一條最適合的線性回歸線，同時排除了數據中的離群值。以下是程式碼的解釋：

- `ransac = RANSACRegressor(LinearRegression(), max_trials=100, min_samples=0.95, loss='absolute_error', residual_threshold=None, random_state=123)`：創建了一個RANSAC回歸器。這裡使用了`LinearRegression()`作為基礎估計器。參數`max_trials`設置最大迭代次數，`min_samples`設置每次迭代所需的最小樣本數比例，`loss`指定損失函數（這裡使用絕對誤差），`residual_threshold`設置殘差閾值（預設為None表示沒有限制），`random_state`設置隨機種子以確保結果的可重現性。
- `ransac.fit(X, y)`：使用RANSAC模型擬合數據。它將自動擬合並找出擬合後的內部和外部數據點（inliers和outliers）。
- `inlier_mask = ransac.inlier_mask_`：得到內部數據點的布爾遮罩（boolean mask）。
- `outlier_mask = np.logical_not(inlier_mask)`：使用`logical_not`函數得到外部數據點的遮罩。
- `line_X = np.arange(3, 10, 1)`：創建了一系列在3到10之間間隔為1的數據點，這將用於繪製回歸線。
- `line_y_ransac = ransac.predict(line_X[:, np.newaxis])`：使用RANSAC模型預測這些數據點的回歸結果。
- `plt.scatter(X[inlier_mask], y[inlier_mask], c='steelblue', edgecolor='white', marker='o', label='Inliers')`：繪製內部數據點，顏色為鋼藍色。
- `plt.scatter(X[outlier_mask], y[outlier_mask], c='limegreen', edgecolor='white', marker='s',`

`label='Outliers')`：繪製外部數據點，顏色為橙綠色。

- `plt.plot(line_X, line_y_ransac, color='black', lw=2)`：繪製RANSAC模型預測的回歸線，顏色為黑色，線寬為2。
- `plt.xlabel('Living area above ground in square feet')`：設置x軸標籤為“Living area above ground in square feet”。
- `plt.ylabel('Sale price in U.S. dollars')`：設置y軸標籤為“Sale price in U.S. dollars”。
- `plt.legend(loc='upper left')`：添加圖例，顯示內部和外部數據點的標籤。
- `plt.tight_layout()`：自動調整圖形佈局，以防止標籤重疊。
- `plt.show()`：顯示繪製的圖形。

這段程式碼展示了如何使用RANSAC回歸模型擬合數據，並且可以有效地處理存在離群值的數據集。

In [28]:
```
print(f'Slope: {ransac.estimator_.coef_[0]:.3f}')
print(f'Intercept: {ransac.estimator_.intercept_:.3f}')
```

```
Slope: 106.348
Intercept: 20190.093
```

這段程式碼用於印出RANSAC模型中基礎估計器（`LinearRegression`）的斜率和截距。以下是程式碼的解釋：

- `ransac.estimator_.coef_[0]`：使用`ransac.estimator_`屬性可以訪問到RANSAC模型中使用的基礎估計器（`LinearRegression`）的斜率（係數）。
- `ransac.estimator_.intercept_`：同樣，使用`ransac.estimator_`屬性可以訪問到基礎估計器的截距。

這兩行程式碼將列印出RANSAC模型找到的擬合直線的斜率和截距，這些值是基於模型擬合後的數據集得出的。

In [38]:
```
def median_absolute_deviation(data):
    return np.median(np.abs(data - np.median(data)))

median_absolute_deviation(y)
```

Out[38]: 37000.0

這段程式碼定義了一個函式`median_absolute_deviation(data)`，用來計算給定數據集的絕對中位數偏差（Median Absolute Deviation, MAD）。

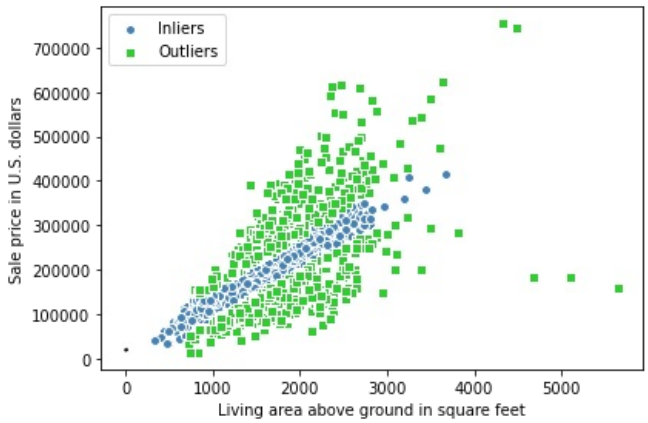
解釋如下：

1. `np.median(data)`：使用NumPy庫的`median()`函式計算數據集`data`的中位數。
2. `np.abs(data - np.median(data))`：計算數據集中每個數據點與中位數之間的絕對差值。
3. `np.median(np.abs(data - np.median(data)))`：使用`median()`函式再次計算上一步計算得到的絕對差值的中位數，從而得到絕對中位數偏差（MAD）。

最後，`median_absolute_deviation(y)`呼叫這個函式並計算數據集`y`的絕對中位數偏差。

In [39]:
```
ransac = RANSACRegressor(LinearRegression(),
                         max_trials=100, # default
                         min_samples=0.95,
                         loss='absolute_error', # default
                         residual_threshold=65000, # default
                         random_state=123)

ransac.fit(X, y)

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('Living area above ground in square feet')
plt.ylabel('Sale price in U.S. dollars')
plt.legend(loc='upper left')

plt.tight_layout()
```

```
#plt.savefig('figures/09_10.png', dpi=300)
plt.show()
```



這段程式碼示範了如何使用 RANSACRegressor 進行隨機樣本一致性估計（RANSAC）來擬合具有離群值的線性回歸模型。

解釋如下：

1. `ransac = RANSACRegressor(LinearRegression(), max_trials=100, min_samples=0.95, loss='absolute_error', residual_threshold=65000, random_state=123)`
   - `RANSACRegressor` 是 Scikit-learn 中的一種回歸模型，用於處理包含離群值的數據集。這裡使用 `LinearRegression()` 作為基本估計器。
   - `max_trials=100`：最大試驗次數，指定RANSAC算法嘗試擬合模型的最大次數。
   - `min_samples=0.95`：每次試驗所需的最小樣本百分比，這裡表示至少要使用數據集的95%來擬合模型。
   - `loss='absolute_error'`：損失函數設置為絕對誤差。
   - `residual_threshold=65000`：殘差閾值，超過這個閾值的點將被視為離群值。
   - `random_state=123`：隨機數生成器的種子，用於可重現性。
2. `ransac.fit(X, y)`
   - 使用 RANSAC 擬合模型到輸入的特徵 `X` 和目標 `y` 上。
3. 接下來的程式碼用來繪製結果：
   - `inlier_mask` 和 `outlier_mask` 分別標識出內部點和離群點。
   - `line_X` 用於定義要繪製RANSAC擬合線的範圍。
   - `line_y_ransac` 計算 `line_X` 範圍內的預測值。
   - 使用 `plt.scatter()` 繪製內部點和離群點的散點圖，顏色分別是藍色和綠色，並添加標籤。
   - 使用 `plt.plot()` 繪製RANSAC擬合線，顏色設置為黑色並設置線寬為2。
   - 設置標籤和標題，最後使用 `plt.tight_layout()` 來調整圖形佈局並顯示圖形。

這段程式碼演示了如何使用 RANSAC 擬合線性回歸模型，以處理具有離群值的數據集，並使用 Matplotlib 來視覺化結果。

In [31]:
```
print(f'Slope: {ransac.estimator_.coef_[0]:.3f}')
print(f'Intercept: {ransac.estimator_.intercept_:.3f}')
```

```
Slope: 105.631
Intercept: 18314.587
```

這段程式碼的目的是打印RANSAC估計器的斜率和截距。解釋如下：

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

- `ransac.estimator_` 是RANSAC中使用的基本線性回歸模型。這裡的 `coef_` 和 `intercept_` 分別代表線性回歸模型的係數（斜率）和截距。
- `ransac.estimator_.coef_[0]` 提取斜率（係數），使用 `print(f'Slope: {ransac.estimator_.coef_[0]:.3f}')` 將其格式化為三位小數並打印。
- `ransac.estimator_.intercept_` 提取截距，使用 `print(f'Intercept: {ransac.estimator_.intercept_:.3f}')` 將其格式化為三位小數並打印。

這些打印語句的輸出結果會顯示RANSAC模型的斜率和截距，幫助你了解模型的擬合參數。

Evaluating the performance of linear regression models

In [32]:
```
from sklearn.model_selection import train_test_split


target = 'SalePrice'
features = df.columns[df.columns != target]

X = df[features].values
y = df[target].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=123)
```

這段程式碼使用了Scikit-Learn中的 `train_test_split` 函數，將數據集劃分為訓練集和測試集。以下是這段程式碼的解釋：

1. **目標與特徵定義**：
   - `target = 'SalePrice'`：指定了目標變量，即我們希望預測的房屋銷售價格。
   - `features = df.columns[df.columns != target]`：定義了特徵變量，這些是除了目標變量外數據集中的所有列。
2. **提取特徵和目標**：
   - `X = df[features].values`：將所有特徵變量的值提取為 `X`，這些特徵將用於建立模型。
   - `y = df[target].values`：將目標變量（銷售價格）的值提取為 `y`，這是我們希望模型預測的值。
3. **訓練集和測試集劃分**：
   - `train_test_split(X, y, test_size=0.3, random_state=123)`：這是劃分函數的調用。
     - `X` 和 `y` 是要劃分的特徵和目標數據。
     - `test_size=0.3` 表示將數據集劃分為30%的測試集和70%的訓練集。
     - `random_state=123` 是隨機數生成器的種子，確保每次運行結果的一致性，即確保隨機劃分的可重現性。
4. **返回值**：
   - `X_train, X_test, y_train, y_test`：這四個變量分別包含訓練集和測試集的特徵和目標值。

這樣的劃分方法可以確保在機器學習模型訓練和評估中，使用獨立的數據集來驗證模型的泛化能力和效果。

In [33]:
```
slr = LinearRegression()

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

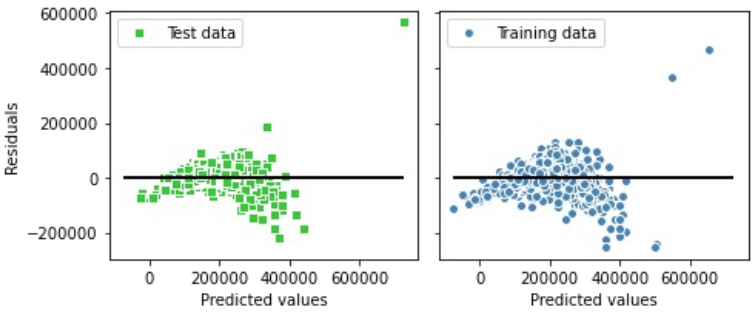
這段程式碼使用了 `LinearRegression` 模型來訓練和預測房屋銷售價格。以下是程式碼的解釋：

1. **建立和訓練模型**：
   - `slr = LinearRegression()`：建立一個線性回歸模型的實例。
   - `slr.fit(X_train, y_train)`：使用訓練集 `X_train` 和對應的目標變量 `y_train` 來訓練線性回歸模型。模型通過最小化平方回歸損失函數來擬合訓練數據，找到最佳的斜率和截距參數。
2. **進行預測**：
   - `y_train_pred = slr.predict(X_train)`：使用訓練好的模型對訓練集 `X_train` 進行預測，得到預測的銷售價格 `y_train_pred`。
   - `y_test_pred = slr.predict(X_test)`：使用訓練好的模型對測試集 `X_test` 進行預測，得到預測的銷售價格 `y_test_pred`。

這樣，我們就可以利用 `LinearRegression` 模型來預測房屋銷售價格，並使用訓練集和測試集來評估模型的性能和泛化能力。

In [34]:
```
x_max = np.max([np.max(y_train_pred), np.max(y_test_pred)])
x_min = np.min([np.min(y_train_pred), np.min(y_test_pred)])

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3), sharey=True)

ax1.scatter(y_test_pred, y_test_pred - y_test,
            c='limegreen', marker='s', edgecolor='white',
            label='Test data')
ax2.scatter(y_train_pred, y_train_pred - y_train,
            c='steelblue', marker='o', edgecolor='white',
            label='Training data')
ax1.set_ylabel('Residuals')

for ax in (ax1, ax2):
    ax.set_xlabel('Predicted values')
    ax.legend(loc='upper left')
    ax.hlines(y=0, xmin=x_min-100, xmax=x_max+100, color='black', lw=2)
```

```
plt.tight_layout()

#plt.savefig('figures/09_11.png', dpi=300)
plt.show()
```



這段程式碼用於繪製模型預測值與殘差之間的散佈圖，用來評估線性回歸模型的表現。以下是程式碼的解釋：

1. **建立圖表**：
   - `fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3), sharey=True)`：建立一個包含兩個子圖（`ax1` 和 `ax2`）的圖表，每個子圖的大小為 (7, 3)。
2. **繪製散佈圖**：
   - `ax1.scatter(y_test_pred, y_test_pred - y_test, c='limegreen', marker='s', edgecolor='white', label='Test data')`：在第一個子圖 `ax1` 中繪製測試集的預測值（`y_test_pred`）與殘差（`y_test_pred - y_test`）的散佈圖。殘差是預測值與實際值之間的差異。
   - `ax2.scatter(y_train_pred, y_train_pred - y_train, c='steelblue', marker='o', edgecolor='white', label='Training data')`：在第二個子圖 `ax2` 中繪製訓練集的預測值（`y_train_pred`）與殘差（`y_train_pred - y_train`）的散佈圖。
3. **設置圖表屬性**：
   - `ax1.set_ylabel('Residuals')`：設置第一個子圖 `ax1` 的 y 軸標籤為 "Residuals"，表示殘差。
   - `for ax in (ax1, ax2):`：對每個子圖執行以下操作：
     - `ax.set_xlabel('Predicted values')`：設置 x 軸標籤為 "Predicted values"，表示預測值。
     - `ax.legend(loc='upper left')`：顯示圖例，位置為左上角。
     - `ax.hlines(y=0, xmin=x_min-100, xmax=x_max+100, color='black', lw=2)`：繪製一條 y=0 的水平線，用來標示殘差為0的位置。
4. **調整圖表布局和顯示**：
   - `plt.tight_layout()`：調整子圖之間的間距，使得圖表更加美觀。
   - `plt.show()`：顯示圖表。

這樣的圖表可以幫助我們觀察模型的預測效果，特別是對於預測值的偏差情況，以及殘差的分布是否符合模型假設。

In [35]:
```
from sklearn.metrics import mean_squared_error


mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
print(f'MSE train: {mse_train:.2f}')
print(f'MSE test: {mse_test:.2f}')
```

```
MSE train: 1497216245.85
MSE test: 1516565821.00
```

這段程式碼用於計算並顯示線性回歸模型在訓練集和測試集上的均方誤差（Mean Squared Error, MSE）。以下是程式碼的解釋：

1. **匯入套件**：
   - `from sklearn.metrics import mean_squared_error`：從 `sklearn.metrics` 模組中匯入 `mean_squared_error` 函數，用於計算均方誤差。
2. **計算均方誤差**：
   - `mse_train = mean_squared_error(y_train, y_train_pred)`：計算訓練集上的均方誤差。`y_train` 是訓練集的實際目標值，`y_train_pred` 是模型對訓練集的預測值。
   - `mse_test = mean_squared_error(y_test, y_test_pred)`：計算測試集上的均方誤差。`y_test` 是測試集的實際目標值，`y_test_pred` 是模型對測試集的預測值。
3. **顯示均方誤差**：

- `print(f'MSE train: {mse_train:.2f}')`：顯示訓練集上的均方誤差，保留兩位小數。
- `print(f'MSE test: {mse_test:.2f}')`：顯示測試集上的均方誤差，保留兩位小數。

這樣的均方誤差評估可以幫助我們評估模型在訓練集和測試集上的預測精度，MSE 值越小表示模型的預測能力越好。

In [36]:
```
from sklearn.metrics import mean_absolute_error


mae_train = mean_absolute_error(y_train, y_train_pred)
mae_test = mean_absolute_error(y_test, y_test_pred)
print(f'MAE train: {mae_train:.2f}')
print(f'MAE test: {mae_test:.2f}')
```

```
MAE train: 25983.03
MAE test: 24921.29
```

這段程式碼計算並顯示線性回歸模型在訓練集和測試集上的平均絕對誤差（Mean Absolute Error, MAE）。以下是程式碼的解釋：

1. **匯入套件**：
   - `from sklearn.metrics import mean_absolute_error`：從 `sklearn.metrics` 模組中匯入 `mean_absolute_error` 函數，用於計算平均絕對誤差。
2. **計算平均絕對誤差**：
   - `mae_train = mean_absolute_error(y_train, y_train_pred)`：計算訓練集上的平均絕對誤差。`y_train` 是訓練集的實際目標值，`y_train_pred` 是模型對訓練集的預測值。
   - `mae_test = mean_absolute_error(y_test, y_test_pred)`：計算測試集上的平均絕對誤差。`y_test` 是測試集的實際目標值，`y_test_pred` 是模型對測試集的預測值。
3. **顯示平均絕對誤差**：
   - `print(f'MAE train: {mae_train:.2f}')`：顯示訓練集上的平均絕對誤差，保留兩位小數。
   - `print(f'MAE test: {mae_test:.2f}')`：顯示測試集上的平均絕對誤差，保留兩位小數。

平均絕對誤差（MAE）是回歸模型評估指標之一，它度量了模型預測值與實際值之間的平均絕對偏差。MAE 值越小表示模型的預測精度越高。

In [37]:
```
from sklearn.metrics import r2_score


r2_train = r2_score(y_train, y_train_pred)
r2_test =r2_score(y_test, y_test_pred)
print(f'R^2 train: {r2_train:.2f}')
print(f'R^2 test: {r2_test:.2f}')
```

```
R^2 train: 0.77
R^2 test: 0.75
```

這段程式碼計算並顯示線性回歸模型在訓練集和測試集上的決定係數（R^2 score）。以下是程式碼的解釋：

1. **匯入套件**：
   - `from sklearn.metrics import r2_score`：從 `sklearn.metrics` 模組中匯入 `r2_score` 函數，用於計算決定係數。
2. **計算決定係數**：
   - `r2_train = r2_score(y_train, y_train_pred)`：計算訓練集上的決定係數。`y_train` 是訓練集的實際目標值，`y_train_pred` 是模型對訓練集的預測值。
   - `r2_test = r2_score(y_test, y_test_pred)`：計算測試集上的決定係數。`y_test` 是測試集的實際目標值，`y_test_pred` 是模型對測試集的預測值。
3. **顯示決定係數**：
   - `print(f'R^2 train: {r2_train:.2f}')`：顯示訓練集上的決定係數，保留兩位小數。
   - `print(f'R^2 test: {r2_test:.2f}')`：顯示測試集上的決定係數，保留兩位小數。

決定係數（R^2 score）是回歸模型評估指標之一，它表示模型解釋目標變量方差的比例。R^2 值越接近 1，表示模型的解釋能力越強；越接近 0，表示模型的解釋能力越差。

Using regularized methods for regression

In [38]:
```
from sklearn.linear_model import Lasso


lasso = Lasso(alpha=1.0)
lasso.fit(X_train, y_train)
y_train_pred = lasso.predict(X_train)
y_test_pred = lasso.predict(X_test)
print(lasso.coef_)
```

```
[26251.38276394   804.70816337    41.94651964 11364.80761309
    55.67855548]
```

這段程式碼使用了 Lasso 回歸模型來進行特徵選擇。以下是程式碼的解釋：

1. **匯入套件**：
   - `from sklearn.linear_model import Lasso`：從 `sklearn.linear_model` 模組中匯入 `Lasso` 類別，用於建立 Lasso 回歸模型。
2. **建立 Lasso 回歸模型**：
   - `lasso = Lasso(alpha=1.0)`：初始化 Lasso 回歸模型的物件。在這裡，`alpha` 是 L1 正則化項的係數，控制正則化的強度。較大的 `alpha` 值將使得模型趨向於將係數推向零，從而達到特徵選擇的效果。
3. **訓練模型**：
   - `lasso.fit(X_train, y_train)`：用訓練集 `X_train` 和對應的目標變量 `y_train` 來訓練 Lasso 回歸模型。
4. **進行預測**：
   - `y_train_pred = lasso.predict(X_train)`：對訓練集進行預測，得到預測值 `y_train_pred`。
   - `y_test_pred = lasso.predict(X_test)`：對測試集進行預測，得到預測值 `y_test_pred`。
5. **顯示係數**：
   - `print(lasso.coef_)`：顯示訓練後的 Lasso 模型的係數。這些係數反映了每個特徵對於目標變量的影響程度。由於 Lasso 通過正則化來推動係數向零，因此一些係數可能會被完全推到零，從而達到特徵選擇的目的。

Lasso 回歸模型是一種在線性回歸基礎上增加了 L1 正則化的變體，它不僅能用於預測，還能同時進行特徵選擇，特別是在具有多個特徵且存在共線性的數據集上效果顯著。

In [39]:
```
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
print(f'MSE train: {train_mse:.3f}, test: {test_mse:.3f}')

train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
print(f'R^2 train: {train_r2:.3f}, {test_r2:.3f}')
```

```
MSE train: 1497216262.014, test: 1516576825.348
R^2 train: 0.769, 0.752
```

這段程式碼計算了使用 Lasso 回歸模型進行訓練和測試集的均方誤差（MSE）和決定係數（R^2）。以下是程式碼的解釋：

1. **計算訓練集和測試集的均方誤差（MSE）**：
   - `train_mse = mean_squared_error(y_train, y_train_pred)`：使用 `mean_squared_error` 函數計算訓練集的均方誤差，即預測值 `y_train_pred` 與實際目標值 `y_train` 的差的平方的平均值。
   - `test_mse = mean_squared_error(y_test, y_test_pred)`：使用 `mean_squared_error` 函數計算測試集的均方誤差，即預測值 `y_test_pred` 與實際目標值 `y_test` 的差的平方的平均值。
   - `print(f'MSE train: {train_mse:.3f}, test: {test_mse:.3f}')`：將計算出的訓練集和測試集的均方誤差顯示為輸出。
2. **計算訓練集和測試集的決定係數（R^2）**：
   - `train_r2 = r2_score(y_train, y_train_pred)`：使用 `r2_score` 函數計算訓練集的決定係數，衡量模型對訓練集數據的擬合程度。
   - `test_r2 = r2_score(y_test, y_test_pred)`：使用 `r2_score` 函數計算測試集的決定係數，衡量模型對測試集數據的擬合程度。
   - `print(f'R^2 train: {train_r2:.3f}, {test_r2:.3f}')`：將計算出的訓練集和測試集的決定係數顯示為輸出。

這些指標是評估模型性能的重要指標：

- **MSE** 越小表示模型的預測誤差越小，模型的預測能力越好。
- **R^2** 越接近於1表示模型對數據的擬合越好，解釋方差越多。

通過計算並比較訓練集和測試集的指標，可以評估模型的泛化能力和是否存在過擬合或者欠擬合的問題。

Ridge regression:

In [40]:
```
from sklearn.linear_model import Ridge


ridge = Ridge(alpha=1.0)
```

這段程式碼使用了 scikit-learn 中的 Ridge 回歸模型來建立一個名為 `ridge` 的物件。Ridge 回歸是一種線性回歸模型，它加入了 L2 正規化，可以用來處理數據中的多重共線性問題，同時減少模型的過度擬合。

解釋：

1. **導入 Ridge 模型**：

   ```
   from sklearn.linear_model import Ridge
   ```
   這行程式碼導入了 scikit-learn 中的 Ridge 回歸模型。

2. **創建 Ridge 模型實例**：

   ```
   ridge = Ridge(alpha=1.0)
   ```
   - `Ridge(alpha=1.0)`：這段程式碼建立了一個 Ridge 回歸模型的實例。
   - `alpha=1.0`：是 Ridge 回歸模型的超參數，用來控制正規化的強度。在這裡，`alpha` 設置為 `1.0`，這意味著我們對模型的係數進行中等程度的正規化。

Ridge 回歸模型在應對具有高度相關特徵的數據集時特別有效，通常用於減少模型的方差，提高其泛化能力。

LASSO regression:

In [41]:
```
from sklearn.linear_model import Lasso


lasso = Lasso(alpha=1.0)
```

這段程式碼使用了 scikit-learn 中的 Lasso 回歸模型來建立一個名為 `lasso` 的物件。

解釋：

1. **導入 Lasso 模型**：

   ```
   from sklearn.linear_model import Lasso
   ```
   這行程式碼導入了 scikit-learn 中的 Lasso 回歸模型。

2. **創建 Lasso 模型實例**：

   ```
   lasso = Lasso(alpha=1.0)
   ```
   - `Lasso(alpha=1.0)`：這段程式碼建立了一個 Lasso 回歸模型的實例。
   - `alpha=1.0`：是 Lasso 回歸模型的超參數，用來控制正規化的強度。在這裡，`alpha` 設置為 `1.0`，這意味著我們對模型的係數進行中等程度的正規化。

Lasso 回歸模型與 Ridge 回歸模型類似，都是用來處理線性回歸中的過度擬合問題。不同的是，Lasso 回歸使用 L1 正規化，這使得一些係數可以變為零，從而實現特徵選擇的效果。

Elastic Net regression:

In [42]:
```
from sklearn.linear_model import ElasticNet


elanet = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

這段程式碼使用了 scikit-learn 中的 ElasticNet 模型來建立一個名為 `elanet` 的物件。

解釋：

1. **導入 ElasticNet 模型**：

   ```
   from sklearn.linear_model import ElasticNet
   ```
   這行程式碼導入了 scikit-learn 中的 ElasticNet 模型，它是一種同時具有 L1 和 L2 正規化的線性回歸模型。

2. **創建 ElasticNet 模型實例**：

   ```
   elanet = ElasticNet(alpha=1.0, l1_ratio=0.5)
   ```
   - `ElasticNet(alpha=1.0, l1_ratio=0.5)`：這段程式碼建立了一個 ElasticNet 模型的實例。
   - `alpha=1.0`：是 ElasticNet 模型的正規化項參數，用來控制正規化的強度。
   - `l1_ratio=0.5`：是 ElasticNet 模型中 L1 正規化的比例。當 `l1_ratio=0` 時，模型等同於 Ridge 回歸；當 `l1_ratio=1`

時，模型等同於 Lasso 回歸。在這裡，`l1_ratio=0.5` 表示 L1 和 L2 正規化的比例各佔一半。

ElasticNet 模型結合了 Lasso 和 Ridge 兩種正規化的優點，同時可以有效地處理具有高度共線性的資料和特徵選擇的問題。

Turning a linear regression model into a curve - polynomial regression

In [43]:
```
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])
```

這段程式碼定義了兩個 numpy 陣列 `X` 和 `y`，用來表示一組簡單的資料集。

解釋：

1. **定義 `X` 陣列**：

   ```
   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]
   ```
   - `X` 陣列包含了一組特徵資料，這裡是房屋面積（單位不詳），每個元素都是一個浮點數。
   - `np.newaxis` 用來在 `X` 陣列中增加一個新的維度，將原本的一維陣列轉換成列向量。
2. **定義 `y` 陣列**：

   ```
   y = np.array([236.4, 234.4, 252.8,
                 298.6, 314.2, 342.2,
                 360.8, 368.0, 391.2,
                 390.8])
   ```
   - `y` 陣列包含了目標變數（或稱標籤）的資料，這裡是對應的房價資料（單位不詳），每個元素也是一個浮點數。

這些資料通常用來示範或訓練機器學習模型，例如線性回歸或其他迴歸模型，用來預測 `X`（特徵）對應的 `y`（目標變數）。

In [44]:
```
from sklearn.preprocessing import PolynomialFeatures


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

這段程式碼使用了 `PolynomialFeatures` 類別來產生多項式特徵。以下是對程式碼的解釋：

解釋：

1. **匯入套件和類別**：

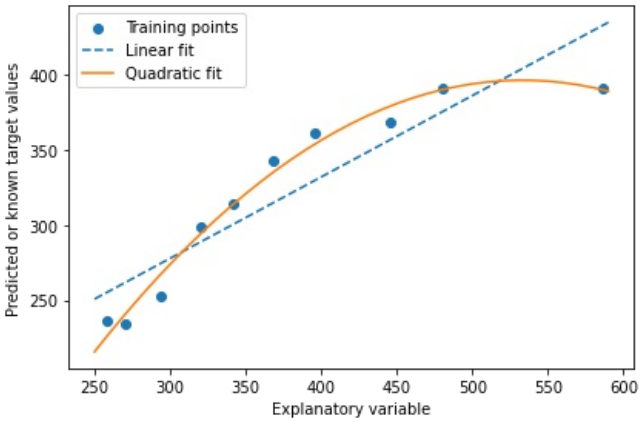
   ```
   from sklearn.preprocessing import PolynomialFeatures
   ```
   - `PolynomialFeatures` 是 Scikit-learn 套件中用來生成多項式特徵的類別。
2. **初始化模型**：

   ```
   lr = LinearRegression()
   pr = LinearRegression()
   ```
   - 初始化了兩個線性回歸模型，`lr` 和 `pr`。通常情況下，`lr` 用於比較基本的線性回歸，而 `pr` 則可能是用來測試或保留多項式特徵後的模型。
3. **生成多項式特徵**：

   ```
   quadratic = PolynomialFeatures(degree=2)
   X_quad = quadratic.fit_transform(X)
   ```
   - `PolynomialFeatures` 的 `degree` 參數設置為 `2`，這表示生成二次多項式特徵。
   - `fit_transform(X)` 方法將原始的特徵 `X` 轉換為包含原始特徵及其二次組合的新特徵矩陣 `X_quad`。
   - 例如，如果原始特徵 `X` 是一維的，比如 `X = [a, b, c]`，那麼 `X_quad` 會包含 `[1, a, a^2], [1, b, b^2], [1, c, c^2]` 等新的特徵。

生成這些多項式特徵後，通常會將 `X_quad` 用於線性回歸模型的訓練，以更好地擬合複雜的數據模式或非線性關係。

In [45]:
```
# 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.xlabel('Explanatory variable')
plt.ylabel('Predicted or known target values')
plt.legend(loc='upper left')

plt.tight_layout()
#plt.savefig('figures/09_12.png', dpi=300)
plt.show()
```



這段程式碼用來擬合包含線性和二次多項式特徵的模型，並將結果視覺化呈現。以下是對程式碼的解釋：

解釋：

1. **擬合線性特徵**：

   ```
   lr.fit(X, y)
   X_fit = np.arange(250, 600, 10)[:, np.newaxis]
   y_lin_fit = lr.predict(X_fit)
   ```
   - `lr.fit(X, y)` 使用線性回歸模型 `lr` 對原始特徵 `X` 和目標變量 `y` 進行擬合。
   - `np.arange(250, 600, 10)[:, np.newaxis]` 產生了從 250 到 600 的等間隔數字序列，並將其轉換為列向量。
   - `lr.predict(X_fit)` 使用訓練好的線性模型對新的特徵 `X_fit` 進行預測，得到 `y_lin_fit`，即線性擬合的預測結果。
2. **擬合二次多項式特徵**：

   ```
   pr.fit(X_quad, y)
   y_quad_fit = pr.predict(quadratic.fit_transform(X_fit))
   ```
   - `pr.fit(X_quad, y)` 使用線性回歸模型 `pr` 對二次多項式特徵 `X_quad` 和目標變量 `y` 進行擬合。
   - `quadratic.fit_transform(X_fit)` 將新的特徵 `X_fit` 轉換為包含一次和二次項的二次多項式特徵，這樣可以將其用於 `pr` 模型的預測。
   - `pr.predict(...)` 使用訓練好的二次多項式模型對轉換後的特徵進行預測，得到 `y_quad_fit`，即二次多項式擬合的預測結果。
3. **繪製結果**：

   ```
   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.xlabel('Explanatory variable')
   plt.ylabel('Predicted or known target values')
   plt.legend(loc='upper left')

   plt.tight_layout()
   #plt.savefig('figures/09_12.png', dpi=300)
   plt.show()
   ```
   - `plt.scatter(X, y, label='Training points')` 繪製原始的訓練數據點。
   - `plt.plot(X_fit, y_lin_fit, label='Linear fit', linestyle='--')` 繪製使用線性回歸模型擬合的線性擬合結果。
   - `plt.plot(X_fit, y_quad_fit, label='Quadratic fit')` 繪製使用線性回歸模型擬合的二次多項式擬合結果。
   - 其餘部分是設置圖表的標籤和圖例等。

這段程式碼的目的是比較線性擬合和二次多項式擬合在擬合結果上的差異，並將結果可視化以便於比較。

In [46]:
```
y_lin_pred = lr.predict(X)
y_quad_pred = pr.predict(X_quad)
```

這段程式碼執行了兩個預測操作：

1. **線性模型預測（`lr.predict(X)`）**：
   - 使用 `lr` 物件中的線性模型來對輸入特徵 `X` 進行預測，得到預測的目標值 `y_lin_pred`。
   - `lr.predict(X)` 返回的是根據訓練好的線性回歸模型預測的目標值。
2. **二次多項式模型預測（`pr.predict(X_quad)`）**：
   - 使用 `pr` 物件中的二次多項式模型來對多項式轉換後的特徵 `X_quad` 進行預測，得到預測的目標值 `y_quad_pred`。
   - `pr.predict(X_quad)` 返回的是根據訓練好的二次多項式回歸模型預測的目標值。

這些預測操作可以幫助我們評估模型在訓練數據上的擬合情況，並進行進一步的分析和比較。

In [47]:
```
mse_lin = mean_squared_error(y, y_lin_pred)
mse_quad = mean_squared_error(y, y_quad_pred)
print(f'Training MSE linear: {mse_lin:.3f}'
      f', quadratic: {mse_quad:.3f}')


r2_lin = r2_score(y, y_lin_pred)
r2_quad = r2_score(y, y_quad_pred)
print(f'Training R^2 linear: {r2_lin:.3f}'
      f', quadratic: {r2_quad:.3f}')
```

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

這段程式碼計算了兩個模型（線性模型和二次多項式模型）在訓練集上的均方誤差（MSE）和決定係數（R^2）。以下是對這段程式碼的解釋：

計算均方誤差 (MSE)

1. **線性模型 (Linear Model)**:

   `mse_lin = mean_squared_error(y, y_lin_pred)`
   - `mean_squared_error(y, y_lin_pred)`：使用 `y_lin_pred`（線性模型的預測值）和 `y`（實際目標變量）計算均方誤差。
   - `mse_lin`：儲存了線性模型在訓練集上的均方誤差。
2. **二次多項式模型 (Quadratic Polynomial Model)**:

   `mse_quad = mean_squared_error(y, y_quad_pred)`
   - `mean_squared_error(y, y_quad_pred)`：使用 `y_quad_pred`（二次多項式模型的預測值）和 `y`（實際目標變量）計算均方誤差。
   - `mse_quad`：儲存了二次多項式模型在訓練集上的均方誤差。

計算決定係數 (R^2)

1. **線性模型 (Linear Model)**:

   `r2_lin = r2_score(y, y_lin_pred)`
   - `r2_score(y, y_lin_pred)`：使用 `y_lin_pred`（線性模型的預測值）和 `y`（實際目標變量）計算決定係數（R^2）。
   - `r2_lin`：儲存了線性模型在訓練集上的R^2。
2. **二次多項式模型 (Quadratic Polynomial Model)**:

   `r2_quad = r2_score(y, y_quad_pred)`
   - `r2_score(y, y_quad_pred)`：使用 `y_quad_pred`（二次多項式模型的預測值）和 `y`（實際目標變量）計算決定係數（R^2）。
   - `r2_quad`：儲存了二次多項式模型在訓練集上的R^2。

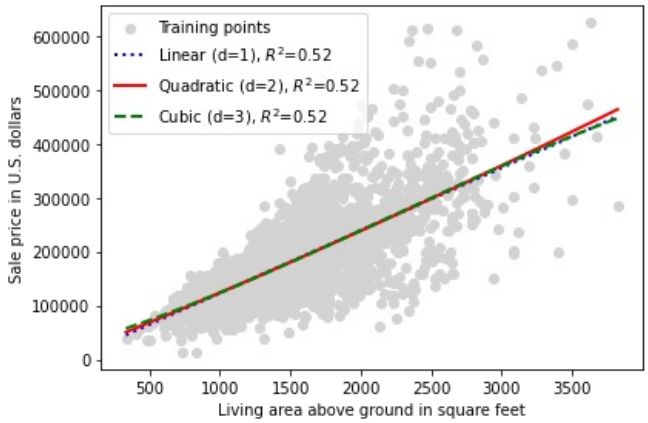
解釋

- **均方誤差 (MSE)**: 衡量模型預測值與實際觀測值之間的平均差異的平方。數值越小越好，代表模型預測的精確度越高。
- **決定係數 (R^2)**: 衡量模型對目標變量變異性的解釋程度。介於0到1之間，數值越接近1代表模型解釋力越強，越接近0則代表模型解釋力較弱。

這些指標用於評估不同模型在訓練集上的表現，有助於了解模型的預測準確度和適配程度。

Modeling nonlinear relationships in the Ames Housing dataset

In [48]:
```
X = df[['Gr Liv Area']].values
y = df['SalePrice'].values

X = X[(df['Gr Liv Area'] < 4000)]
y = y[(df['Gr Liv Area'] < 4000)]


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()-1, X.max()+2, 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))


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

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

plt.plot(X_fit, y_quad_fit,
         label=f'Quadratic (d=2), $R^2$={quadratic_r2:.2f}',
         color='red',
         lw=2,
         linestyle='-')

plt.plot(X_fit, y_cubic_fit,
         label=f'Cubic (d=3), $R^2$={cubic_r2:.2f}',
         color='green',
         lw=2,
         linestyle='--')


plt.xlabel('Living area above ground in square feet')
plt.ylabel('Sale price in U.S. dollars')
plt.legend(loc='upper left')

plt.tight_layout()
plt.savefig('figures/09_13.png', dpi=300)
plt.show()
```



這段程式碼執行了以下操作：

1. 從DataFrame `df` 中選取 'Gr Liv Area' 和 'SalePrice' 欄位作為特徵 `X` 和目標變量 `y`。
2. 篩選出 'Gr Liv Area' 小於 4000 的資料，同時更新 `X` 和 `y`。
3. 創建了三種不同多項式特徵：一次、二次和三次。
4. 分別用線性回歸模型（`LinearRegression`）擬合一次、二次和三次多項式特徵。
5. 計算每個模型的決定係數（R^2 score）。
6. 繪製了散點圖和每個模型的預測線，並在圖例中顯示了每個模型的R^2值。

解釋程式碼：

- **選擇特徵和目標變量**：

  ```
  X = df[['Gr Liv Area']].values
  y = df['SalePrice'].values
  ```
  從DataFrame `df` 中選取 'Gr Liv Area' 和 'SalePrice' 欄位，並轉換為NumPy數組。

- **篩選資料**：

  ```
  X = X[(df['Gr Liv Area'] < 4000)]
  y = y[(df['Gr Liv Area'] < 4000)]
  ```
  篩選出 'Gr Liv Area' 小於 4000 的資料點，同時更新 `X` 和 `y` 變量。

- **創建多項式特徵**：

  ```
  quadratic = PolynomialFeatures(degree=2)
  cubic = PolynomialFeatures(degree=3)
  X_quad = quadratic.fit_transform(X)
  X_cubic = cubic.fit_transform(X)
  ```
  使用 `PolynomialFeatures` 創建二次和三次多項式特徵。

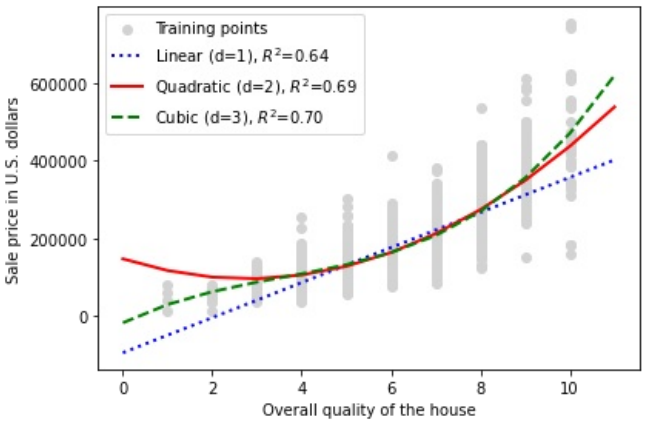
- **擬合模型並計算R^2**：

  ```
  regr = LinearRegression()

  # 線性模型
  regr = regr.fit(X, y)
  linear_r2 = r2_score(y, regr.predict(X))

  # 二次多項式模型
  regr = regr.fit(X_quad, y)
  quadratic_r2 = r2_score(y, regr.predict(X_quad))

  # 三次多項式模型
  regr = regr.fit(X_cubic, y)
  cubic_r2 = r2_score(y, regr.predict(X_cubic))
  ```
  使用 `LinearRegression` 分別對一次、二次和三次多項式特徵進行擬合，並計算每個模型的R^2值。

- **繪製圖表**：

  ```
  plt.scatter(X, y, label='Training points', color='lightgray')

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

  plt.plot(X_fit, y_quad_fit,
           label=f'Quadratic (d=2), $R^2$={quadratic_r2:.2f}',
           color='red',
           lw=2,
           linestyle='-')

  plt.plot(X_fit, y_cubic_fit,
           label=f'Cubic (d=3), $R^2$={cubic_r2:.2f}',
           color='green',
           lw=2,
           linestyle='--')

  plt.xlabel('Living area above ground in square feet')
  plt.ylabel('Sale price in U.S. dollars')
  plt.legend(loc='upper left')

  plt.tight_layout()
  plt.savefig('figures/09_13.png', dpi=300)
  plt.show()
  ```
  繪製了散點圖（標記為灰色的訓練點）和每個模型的預測線（藍色虛線為一次，紅色實線為二次，綠色虛線為三次），並在圖例中顯示了每個模型的R^2值。

In [49]:
```
X = df[['Overall Qual']].values
y = df['SalePrice'].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()-1, X.max()+2, 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))


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

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

plt.plot(X_fit, y_quad_fit,
         label=f'Quadratic (d=2), $R^2$={quadratic_r2:.2f}',
         color='red',
         lw=2,
         linestyle='-')

plt.plot(X_fit, y_cubic_fit,
         label=f'Cubic (d=3), $R^2$={cubic_r2:.2f}',
         color='green',
         lw=2,
         linestyle='--')


plt.xlabel('Overall quality of the house')
plt.ylabel('Sale price in U.S. dollars')
plt.legend(loc='upper left')

plt.tight_layout()
#plt.savefig('figures/09_14.png', dpi=300)
plt.show()
```



這段程式碼執行了以下操作：

1. 從DataFrame `df` 中選取 'Overall Qual' 和 'SalePrice' 欄位作為特徵 `X` 和目標變量 `y`。
2. 創建了三種不同多項式特徵：一次、二次和三次。
3. 分別用線性回歸模型（`LinearRegression`）擬合一次、二次和三次多項式特徵。

4. 計算每個模型的決定係數（R^2 score）。
5. 繪製了散點圖和每個模型的預測線，並在圖例中顯示了每個模型的R^2值。

解釋程式碼：

- **選擇特徵和目標變量**：

  ```
  X = df[['Overall Qual']].values
  y = df['SalePrice'].values
  ```
  從DataFrame `df` 中選取 'Overall Qual' 和 'SalePrice' 欄位，並轉換為NumPy數組。

- **創建多項式特徵**：

  ```
  quadratic = PolynomialFeatures(degree=2)
  cubic = PolynomialFeatures(degree=3)
  X_quad = quadratic.fit_transform(X)
  X_cubic = cubic.fit_transform(X)
  ```
  使用 `PolynomialFeatures` 創建二次和三次多項式特徵。

- **擬合模型並計算R^2**：

  ```
  regr = LinearRegression()

  # 線性模型
  regr = regr.fit(X, y)
  linear_r2 = r2_score(y, regr.predict(X))

  # 二次多項式模型
  regr = regr.fit(X_quad, y)
  quadratic_r2 = r2_score(y, regr.predict(X_quad))

  # 三次多項式模型
  regr = regr.fit(X_cubic, y)
  cubic_r2 = r2_score(y, regr.predict(X_cubic))
  ```
  使用 `LinearRegression` 分別對一次、二次和三次多項式特徵進行擬合，並計算每個模型的R^2值。

- **繪製圖表**：

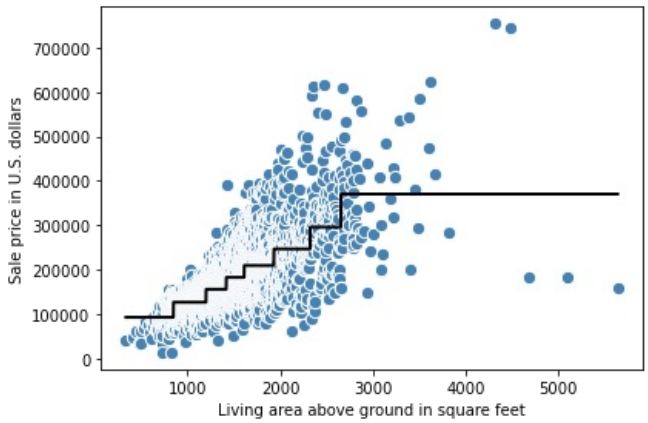
  ```
  plt.scatter(X, y, label='Training points', color='lightgray')

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

  plt.plot(X_fit, y_quad_fit,
           label=f'Quadratic (d=2), $R^2$={quadratic_r2:.2f}',
           color='red',
           lw=2,
           linestyle='-')

  plt.plot(X_fit, y_cubic_fit,
           label=f'Cubic (d=3), $R^2$={cubic_r2:.2f}',
           color='green',
           lw=2,
           linestyle='--')

  plt.xlabel('Overall quality of the house')
  plt.ylabel('Sale price in U.S. dollars')
  plt.legend(loc='upper left')

  plt.tight_layout()
  plt.savefig('figures/09_14.png', dpi=300)
  plt.show()
  ```
  繪製了散點圖（標記為灰色的訓練點）和每個模型的預測線（藍色虛線為一次，紅色實線為二次，綠色虛線為三次），並在圖例中顯示了每個模型的R^2值。

Dealing with nonlinear relationships using random forests

...

Decision tree regression

In [50]:
```
from sklearn.tree import DecisionTreeRegressor


X = df[['Gr Liv Area']].values
y = df['SalePrice'].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('Living area above ground in square feet')
plt.ylabel('Sale price in U.S. dollars')

plt.tight_layout()
#plt.savefig('figures/09_15.png', dpi=300)
plt.show()
```



這段程式碼利用了決策樹回歸器（`DecisionTreeRegressor`）來擬合 'Gr Liv Area' 和 'SalePrice' 這兩個特徵。以下是程式碼的解釋：

1. **匯入必要的函式庫和類別**：

   ```
   from sklearn.tree import DecisionTreeRegressor
   ```
2. **設定特徵和目標變量**：

   ```
   X = df[['Gr Liv Area']].values
   y = df['SalePrice'].values
   ```
   從DataFrame `df` 中選取 'Gr Liv Area' 作為特徵 `X`，並從 'SalePrice' 中選取目標變量 `y`。

3. **初始化和訓練決策樹回歸器**：

   ```
   tree = DecisionTreeRegressor(max_depth=3)
   tree.fit(X, y)
   ```
   創建了一個深度為3的決策樹回歸器（`DecisionTreeRegressor`），並用 `fit` 方法將特徵 `X` 和目標變量 `y` 進行訓練。

4. **準備繪製數據**：

   ```
   sort_idx = X.flatten().argsort()
   ```
   對特徵 `X` 進行扁平化（flatten）並排序，以便按照 'Gr Liv Area' 的大小對數據點進行排序。

5. **繪製數據**：

   ```
   lin_regplot(X[sort_idx], y[sort_idx], tree)
   plt.xlabel('Living area above ground in square feet')
   plt.ylabel('Sale price in U.S. dollars')
   ```
   使用 `lin_regplot` 函式繪製散點圖和決策樹回歸模型的預測線。標籤軸說明 'Living area above ground in square feet' 和 'Sale price in U.S. dollars' 分別表示 X 和 y 的含義。

6. **顯示圖表**：

   ```
   plt.tight_layout()
   #plt.savefig('figures/09_15.png', dpi=300)
   plt.show()
   ```
   調整圖表佈局並顯示圖表。可以根據需要將圖表保存為圖片文件。

這樣，該程式碼演示了如何使用決策樹回歸器擬合和可視化特徵 'Gr Liv Area' 對 'SalePrice' 的回歸關係。

In [51]:
```
tree_r2 = r2_score(y, tree.predict(X))
```

```
tree_r2
```

Out[51]:
```
0.5144569334885711
```

這段程式碼計算了決策樹回歸模型對目標變量 `y` 的 R^2 分數。這個分數顯示了模型解釋目標變量變異性的能力。以下是程式碼的解釋：

1. **計算 R^2 分數**：

   ```
   tree_r2 = r2_score(y, tree.predict(X))
   ```
   - `tree.predict(X)`：使用訓練好的決策樹模型 `tree` 對特徵 `X` 進行預測，得到預測的目標變量。
   - `r2_score(y, ...)`：計算實際目標變量 `y` 與模型預測結果之間的 R^2 分數。

   R^2 分數是一個介於0和1之間的值，越接近1表示模型對目標變量的解釋能力越強，越接近0表示模型的解釋能力越差。

2. **顯示 R^2 分數**：

   ```
   tree_r2
   ```
   打印或返回計算得到的 R^2 分數 `tree_r2`。

這段程式碼通過計算決策樹回歸模型的 R^2 分數來評估模型對於 'Gr Liv Area' 和 'SalePrice' 這兩個變量之間關係的解釋能力。

Random forest regression

In [52]:
```
target = 'SalePrice'
features = df.columns[df.columns != target]

X = df[features].values
y = df[target].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=123)
```

這段程式碼用於將特徵和目標變量準備成訓練和測試集，以便進行機器學習模型的訓練和評估。以下是程式碼的解釋：

1. **指定目標變量和特徵**：

   ```
   target = 'SalePrice'
   features = df.columns[df.columns != target]
   ```
   - `target = 'SalePrice'`：指定目標變量為 'SalePrice'，這是我們想要預測的目標。
   - `features = df.columns[df.columns != target]`：從數據框 `df` 中選擇所有不等於目標變量的列，即其他特徵列，並存儲在 `features` 中。
2. **準備特徵 `X` 和目標變量 `y`**：

   ```
   X = df[features].values
   y = df[target].values
   ```
   - `X = df[features].values`：從數據框 `df` 中選擇特徵 `features` 的值，並將其存儲在 `X` 中。這些特徵將用於訓練機器學習模型。
   - `y = df[target].values`：從數據框 `df` 中選擇目標變量 `target` 的值，並將其存儲在 `y` 中。這是我們希望模型預測的目標。
3. **拆分訓練和測試集**：

   ```
   X_train, X_test, y_train, y_test = train_test_split(
       X, y, test_size=0.3, random_state=123)
   ```
   - `train_test_split` 函數將數據集 `X` 和 `y` 分割為訓練集（`X_train`，`y_train`）和測試集（`X_test`，`y_test`）。
   - `test_size=0.3`：設置測試集的比例為總數據的30%。
   - `random_state=123`：設置隨機種子以確保每次運行時分割的結果相同，這對於結果的可重現性很重要。

這樣，`X_train` 和 `y_train` 將用於訓練模型，而 `X_test` 和 `y_test` 將用於評估模型在未見過的數據上的表現。

In [53]:
```
from sklearn.ensemble import RandomForestRegressor


forest = RandomForestRegressor(n_estimators=1000,
                               criterion='squared_error',
                               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)


mae_train = mean_absolute_error(y_train, y_train_pred)
mae_test = mean_absolute_error(y_test, y_test_pred)
print(f'MAE train: {mae_train:.2f}')
```

```
print(f'MAE test: {mae_test:.2f}')


r2_train = r2_score(y_train, y_train_pred)
r2_test =r2_score(y_test, y_test_pred)
print(f'R^2 train: {r2_train:.2f}')
print(f'R^2 test: {r2_test:.2f}')
```

```
MAE train: 8305.18
MAE test: 20821.77
R^2 train: 0.98
R^2 test: 0.85
```

這段程式碼用隨機森林回歸器訓練了一個模型，並對訓練集和測試集進行了預測和評估。以下是程式碼的解釋：

1. **引入隨機森林回歸器**：

   ```
   from sklearn.ensemble import RandomForestRegressor
   ```

2. **初始化隨機森林模型**：

   ```
   forest = RandomForestRegressor(n_estimators=1000,
                                  criterion='squared_error',
                                  random_state=1,
                                  n_jobs=-1)
   ```

   - `n_estimators=1000`：指定森林中樹木的數量為1000。
   - `criterion='squared_error'`：指定分割節點的準則為平方誤差。
   - `random_state=1`：設置隨機種子以確保每次運行時森林的結果相同。
   - `n_jobs=-1`：設置使用所有可用的CPU核心來加速訓練過程。

3. **訓練隨機森林模型**：

   ```
   forest.fit(X_train, y_train)
   ```

   - 使用 `X_train` 和 `y_train` 訓練隨機森林模型。

4. **進行預測**：

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

   - 使用訓練後的模型對訓練集 `X_train` 和測試集 `X_test` 進行預測，並將預測結果分別存儲在 `y_train_pred` 和 `y_test_pred` 中。

5. **評估模型性能**：

   ```
   mae_train = mean_absolute_error(y_train, y_train_pred)
   mae_test = mean_absolute_error(y_test, y_test_pred)
   print(f'MAE train: {mae_train:.2f}')
   print(f'MAE test: {mae_test:.2f}')
   ```

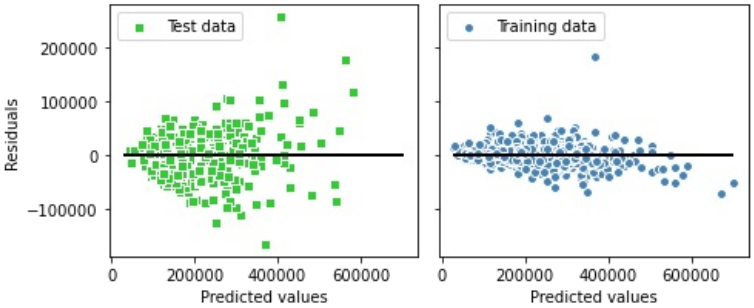
   - 計算並打印出訓練集和測試集的平均絕對誤差 (MAE)。

   ```
   r2_train = r2_score(y_train, y_train_pred)
   r2_test = r2_score(y_test, y_test_pred)
   print(f'R^2 train: {r2_train:.2f}')
   print(f'R^2 test: {r2_test:.2f}')
   ```

   - 計算並打印出訓練集和測試集的決定係數 (R^2)，它是模型擬合數據的度量。

這樣，該段程式碼演示了如何使用隨機森林回歸器在房價預測問題上進行模型訓練、預測和評估。

In [54]:
```
x_max = np.max([np.max(y_train_pred), np.max(y_test_pred)])
x_min = np.min([np.min(y_train_pred), np.min(y_test_pred)])

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 3), sharey=True)

ax1.scatter(y_test_pred, y_test_pred - y_test,
            c='limegreen', marker='s', edgecolor='white',
            label='Test data')
ax2.scatter(y_train_pred, y_train_pred - y_train,
            c='steelblue', marker='o', edgecolor='white',
            label='Training data')
ax1.set_ylabel('Residuals')

for ax in (ax1, ax2):
    ax.set_xlabel('Predicted values')
    ax.legend(loc='upper left')
    ax.hlines(y=0, xmin=x_min-100, xmax=x_max+100, color='black', lw=2)

plt.tight_layout()

#plt.savefig('figures/09_16.png', dpi=300)
plt.show()
```



這段程式碼的目的是比較隨機森林模型在訓練集和測試集上的預測結果。以下是對程式碼的解釋：

1. **設定數據範圍**：
   - `x_max` 和 `x_min` 變數分別計算了訓練集和測試集預測值的最大值和最小值。這些值用於確保圖表的 x 軸範圍足夠覆蓋所有數據點。
2. **創建子圖**：
   - 使用 `plt.subplots(1, 2, figsize=(7, 3), sharey=True)` 創建了一個一行兩列的子圖，每個子圖的大小是 (7, 3)。
   - `fig` 是整個圖形，`ax1` 和 `ax2` 是分別對應於左右兩個子圖的軸物件。
3. **繪製散點圖**：
   - `ax1.scatter` 和 `ax2.scatter` 分別在左右兩個子圖上繪製了散點圖。
   - `y_test_pred` 和 `y_train_pred` 是測試集和訓練集的預測值。
   - `y_test_pred - y_test` 和 `y_train_pred - y_train` 分別計算了測試集和訓練集的殘差。
4. **設置標籤和標題**：
   - `ax1.set_ylabel('Residuals')` 設置了左側子圖的 y 軸標籤為 "Residuals"。
   - `ax.set_xlabel('Predicted values')` 設置了兩個子圖的 x 軸標籤為 "Predicted values"。
   - `ax.legend(loc='upper left')` 添加了每個子圖的圖例，位置在左上角。
5. **繪製水平線**：
   - `ax.hlines(y=0, xmin=x_min-100, xmax=x_max+100, color='black', lw=2)` 在每個子圖上繪製了一條 y=0 的水平線，用於參考。
6. **調整佈局和顯示圖形**：
   - `plt.tight_layout()` 用於自動調整子圖的佈局，以防止重疊。
   - `plt.show()` 顯示了繪製好的圖形。

這段程式碼的最終目的是比較測試集和訓練集上的預測結果和殘差，以評估模型在不同數據集上的表現。

Summary

...

Readers may ignore the next cell.

In [1]:
```
! python ../.convert_notebook_to_script.py --input ch09.ipynb --output ch09.py
```

```
[NbConvertApp] WARNING | Config option `kernel_spec_manager_class` not recognized by `NbConvertApp`.
[NbConvertApp] Converting notebook ch09.ipynb to script
[NbConvertApp] Writing 20411 bytes to ch09.py
```

這段命令是用來將 Jupyter Notebook 轉換為 Python 腳本的工具命令。讓我們逐步解釋：

1. **命令部分**：
   - `!`：這個符號在 Jupyter Notebook 中表示要執行一個 shell 命令，而不是 Python 代碼。
2. **命令解釋**：
   - `python`：這是執行 Python 程式的命令。
   - `../.convert_notebook_to_script.py`：這是要執行的 Python 腳本文件的路徑。`..` 表示上級目錄，`.convert_notebook_to_script.py` 是腳本的文件名。
   - `--input ch09.ipynb`：這是 `convert_notebook_to_script.py` 腳本的參數，指定要轉換的 Jupyter Notebook 文件。`ch09.ipynb` 是要轉換的文件名，`.ipynb` 表示這是一個 Jupyter Notebook 文件。
   - `--output ch09.py`：這是 `convert_notebook_to_script.py` 腳本的參數，指定轉換後生成的 Python 腳本文件的名

稱。 `ch09.py` 是生成的 Python 腳本的文件名， `.py` 表示這是一個 Python 腳本文件。

3. **作用**：

   - 這個命令的作用是將名為 `ch09.ipynb` 的 Jupyter Notebook 文件轉換為名為 `ch09.py` 的 Python 腳本文件。轉換後的 Python 腳本文件可以在命令行或其他環境中運行，而不需要使用 Jupyter Notebook。

4. **注意事項**：

   - 在執行這個命令之前，確保你已經進入包含 `ch09.ipynb` 文件的目錄或提供了正確的文件路徑。
   - `convert_notebook_to_script.py` 可能是自定義的腳本，用於將 Jupyter Notebook 轉換為 Python 腳本。