In [2]:
```
#The vector a holds the squares of integers 0 to n; for instance, if n is equal to 3,
#pt 1 how to solve with python
#pt 2 how to solve with numpy


#Python method below

#value of 'n'?
n = 3

#A & B vectors
#vector a = sqaures
#vector b = cubes
a = [i ** 2 for i in range(n + 1)]
b = [i ** 3 for i in range(n + 1)]
print("vector 'a':", a)
print("vector 'b':", b)
```

```
vector 'a': [0, 1, 4, 9]
vector 'b': [0, 1, 8, 27]
```

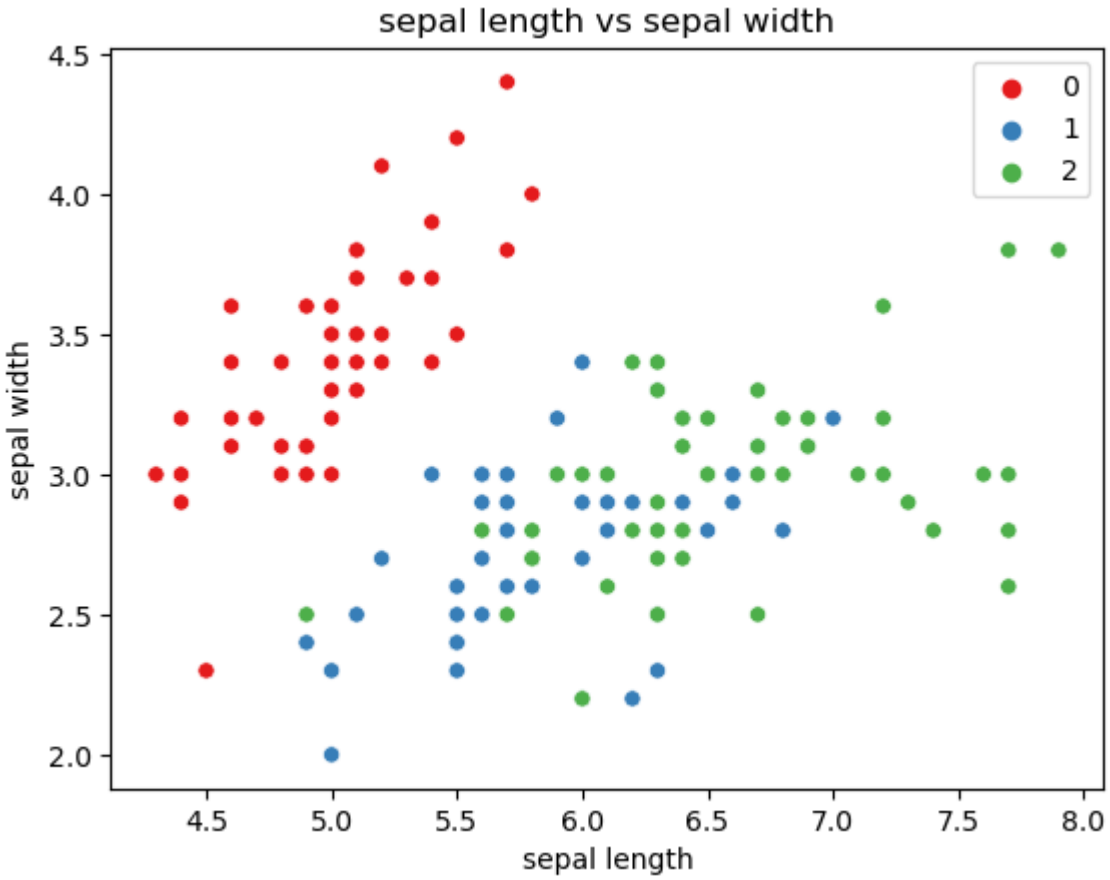
In [5]:
```
#pt 2, numpy method
import numpy as np

#value of 'n'
n = 3

#np.arrays
#a = sqaures
#b = cubes
a = np.array(n + 1) ** 2
b = np.array(n + 1) ** 3
print("numpy array 'a':", a)
print("numpy array 'b':", b)
```

```
numpy array 'a': 16
numpy array 'b': 64
```

In [3]:
```
#Load the iris dataset, print a description of the dataset, and plot column 1 (sepal l

#load sns, plt, sklearn and iris
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

#iris datab
iris = load_iris()

#print a description of the dataset (iris dataset?)
print("iris dataset:")
print(iris.DESCR)

#scatterplot? set (sepal length as x) & (sepal width as y)
sns.scatterplot(x=iris.data[:, 0], y=iris.data[:, 1], hue=iris.target, palette='Set1')
plt.title("sepal length vs sepal width")
plt.xlabel("sepal length")
plt.ylabel("sepal width")
plt.show()
```

```
iris dataset:
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
```

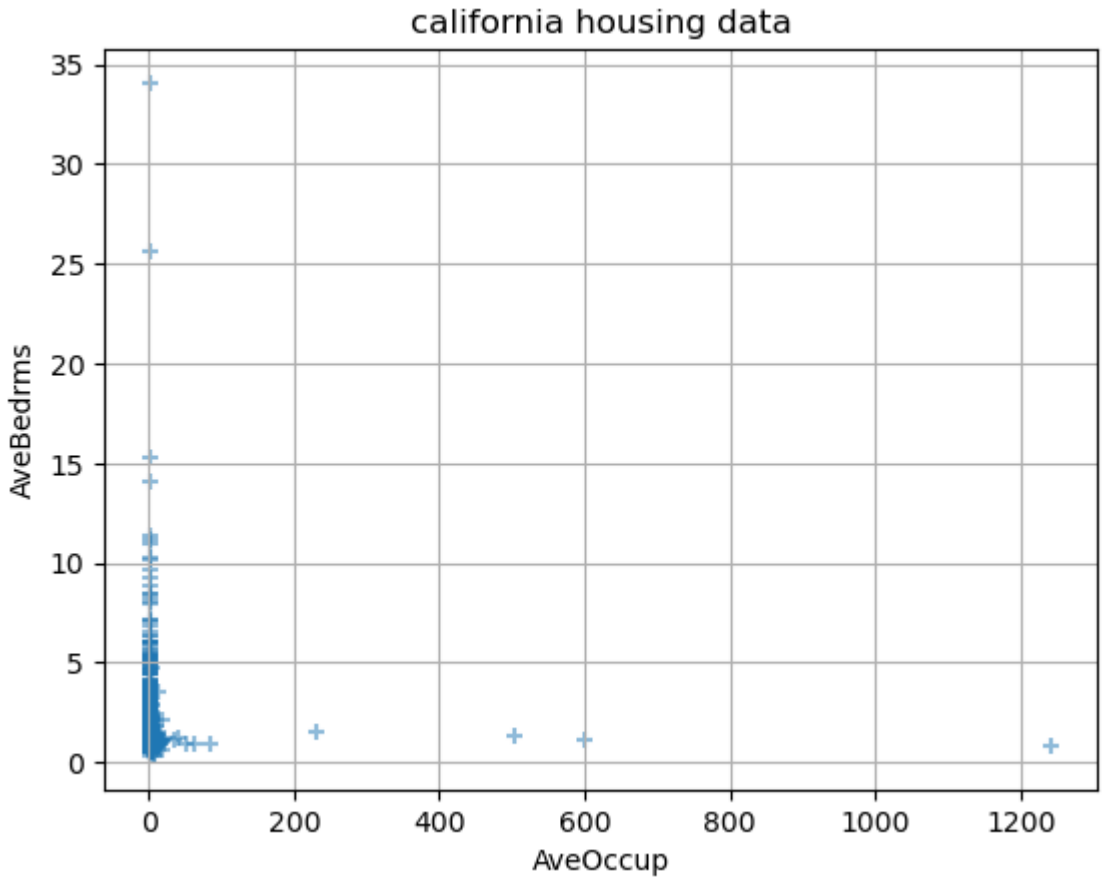
```
  on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
  conceptual clustering system finds 3 classes in the data.
- Many, many more ...
```



In [17]:

```
#Load the boston dataset, print the description of the dataset and plot column 3 (prop

##WAITING ON INSTRUCTOR FEEDBACK FOR DATA SAMPLE BOSTON
#just use california if i cant get boston to work
#boston didnt work , attempting california data set it recommended in the error messag
#use suggested code in error message  from sklearn.datasets import fetch_california_ho

#plt, sklearn, boston or california
#boston didnt work
#california: import pd, plt, sklearn, and california
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

#california dataset
housing = fetch_california_housing(as_frame=True)
housing_df = pd.DataFrame(data=housing.data, columns=housing.feature_names)
print(housing.DESCR)

#(column 5 averooms y) and (column 3 ave0ccups as x)
plt.scatter(housing_df['AveOccup'], housing_df['AveBedrms'], marker='+', alpha=0.5)
plt.xlabel('AveOccup')
plt.ylabel('AveBedrms')
plt.title('california housing data')
plt.grid(True)
plt.show()
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block group
        - HouseAge      median house age in block group
        - AveRooms      average number of rooms per household
        - AveBedrms     average number of bedrooms per household
        - Population    block group population
        - AveOccup      average number of household members
        - Latitude      block group latitude
        - Longitude     block group longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

A household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surprisingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```



In [18]:
```
#Create a multi-dimensional array (Hint: See GitHub repo for the book, Chapter02 > ch-

import numpy as np

#2D array 9=(3 rows/3 columns)
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print("2D array:/n", matrix)
```

```
2D array:/n [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [20]:
```
#Produce an array of single precision floats (Hint: See GitHub repo for the book, Chap

#import np
import numpy as np

#random values?
values = [1.23, 4.56, 7.89, 0.12, 3.45]
#ch 2 dtype=float32)

single_precision_array = np.array(values, dtype=np.float32)
print(single_precision_array)
```

```
[1.23 4.56 7.89 0.12 3.45]
```

In [22]:
```
#Produce an array of complex numbers (Hint: See GitHub repo for the book, Chapter02 >

#do i really have to keep saying this note (import numpy as np)
import numpy as np
```

```
complex_numbers_list = [1 +2j, 3 - 4j, 5 + 6j, 7 - 8j]
#dtype complex 64
complex_numbers_array = np.array(complex_numbers_list, dtype=np.complex64)
print(complex_numbers_array)
```

```
[1.+2.j 3.-4.j 5.+6.j 7.-8.j]
```

In [25]:
```
#Define an array containing the numbers 0, 1, 2, and so on up to and including 8. Sele
#np
import numpy as np

#array from 0 to 8 (go 1 up)
#arange not arrange
original_array = np.arange(9)

#index 3 to 7, extracts the elements 3 to 6
selected_part = original_array[3:7]
#index of 0 to 7 with increment of 2
incremented_part = original_array[0:8:2]

#reverse array
#[::-1]
reversed_array = selected_part[::-1]
#prints
print("original array:", original_array)
print("selected_part:", selected_part)
print("incremented part:", incremented_part)
print("reversed array:", reversed_array)
```

```
original array: [0 1 2 3 4 5 6 7 8]
selected_part: [3 4 5 6]
incremented part: [0 2 4 6]
reversed array: [6 5 4 3]
```

In [28]:
```
#starting on page 34 of your text, create an array and perform the following functions
#import numpy as np
import numpy as np
original_array = np.random.rand(3, 4)

#Ravel
raveled_array = original_array.ravel()
#Flatten
flattened_array = original_array.flatten()
#Setting the shape with a tuple
reshaped_array = original_array.reshape((2, 6))
#Transpose
transposed_array = original_array.T
#Resize
resized_array = np.resize(original_array, (2, 2))

print("original array:", original_array)
print("raveled array:", raveled_array)
print("flattened array:", flattened_array)
print("reshaped array:", reshaped_array)
print("transposed array:", transposed_array)
print("resized array", resized_array)
```

```
original array: [[6.48742904e-01 4.55766958e-02 3.06688834e-01 3.35338651e-01]
 [4.32714245e-01 2.13411282e-01 6.42093027e-01 9.25509755e-02]
 [5.39430148e-04 2.82892716e-01 6.24870482e-01 5.25106678e-01]]
raveled array: [6.48742904e-01 4.55766958e-02 3.06688834e-01 3.35338651e-01
 4.32714245e-01 2.13411282e-01 6.42093027e-01 9.25509755e-02
 5.39430148e-04 2.82892716e-01 6.24870482e-01 5.25106678e-01]
flattened array: [6.48742904e-01 4.55766958e-02 3.06688834e-01 3.35338651e-01
 4.32714245e-01 2.13411282e-01 6.42093027e-01 9.25509755e-02
 5.39430148e-04 2.82892716e-01 6.24870482e-01 5.25106678e-01]
reshaped array: [[6.48742904e-01 4.55766958e-02 3.06688834e-01 3.35338651e-01
  4.32714245e-01 2.13411282e-01]
 [6.42093027e-01 9.25509755e-02 5.39430148e-04 2.82892716e-01
  6.24870482e-01 5.25106678e-01]]
transposed array: [[6.48742904e-01 4.32714245e-01 5.39430148e-04]
 [4.55766958e-02 2.13411282e-01 2.82892716e-01]
 [3.06688834e-01 6.42093027e-01 6.24870482e-01]
 [3.35338651e-01 9.25509755e-02 5.25106678e-01]]
resized array [[0.6487429  0.0455767 ]
 [0.30668883 0.33533865]]
```

In [30]:

```python
#create an array and find the following properties (Hint: See GitHub repo for the book

import numpy as np

random_array = np.random.rand(3, 4, 5)

#Number of dimensions
num_dimensions = random_array.ndim

#Count of elements
num_elements = random_array.size

#Count of bytes
element_size_bytes = random_array.itemsize

#Full count of bytes
total_bytes = random_array.nbytes

print("number of dimensions:", num_dimensions)
print("count of elements:", num_elements)
print("count of bytes:", element_size_bytes)
print("full count of bytes:", total_bytes)
```

```
number of dimensions: 3
count of elements: 60
count of bytes: 8
full count of bytes: 480
```

In [31]:

```python
#convert an array to a Python list (random data)
import numpy as np

#random data
random_array = np.random.rand(5)

#convert array to python
python_list = random_array.tolist()
print("python list:", python_list)
```

```
python list: [0.8416756440019337, 0.5684677542958868, 0.40420658735219883, 0.94827190
24637927, 0.055545859711147005]
```

In [ ]: