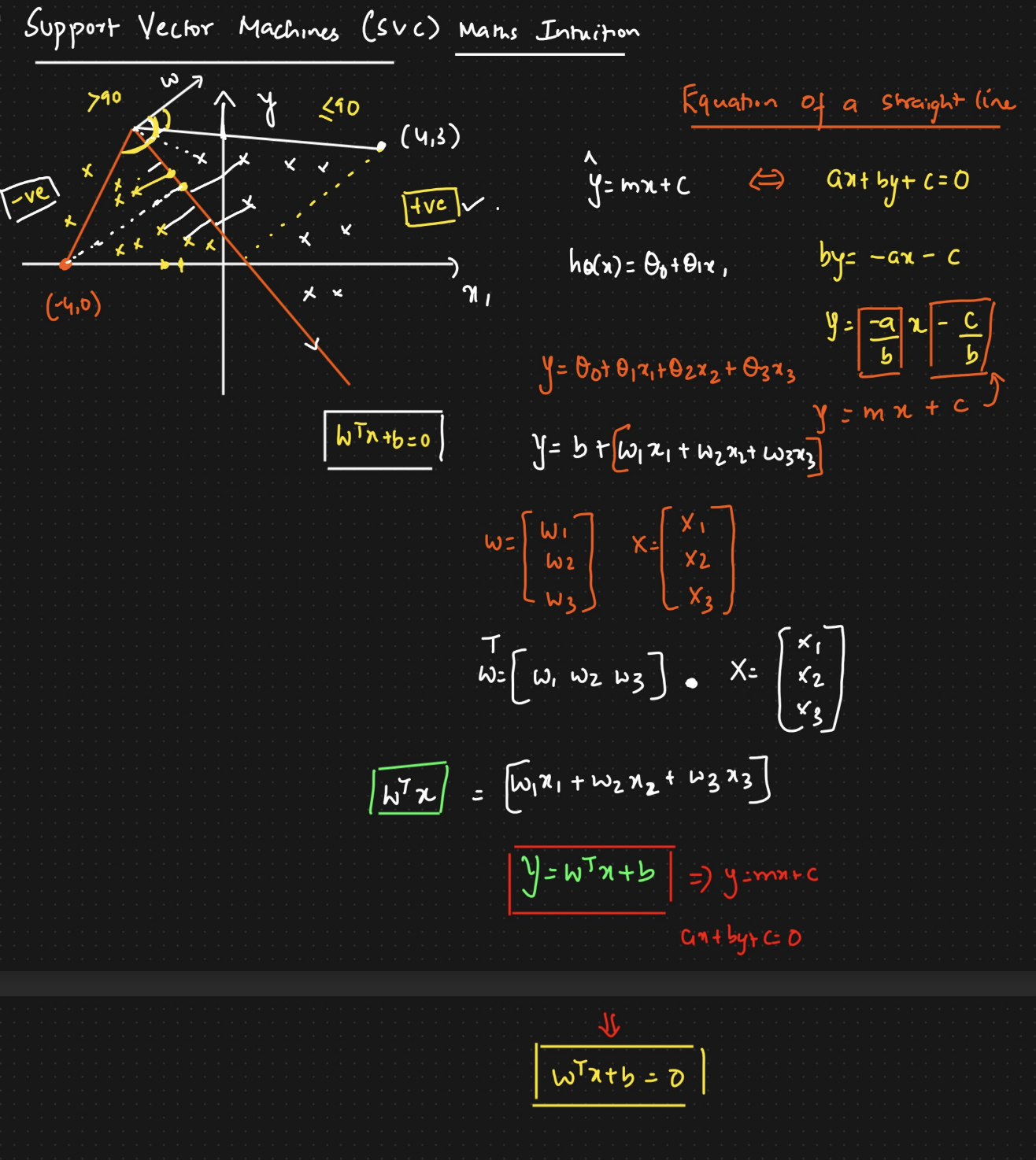
## **Q1. What is the mathematical formula for a linear SVM?**



The goal of a linear SVM is to find a hyperplane which seperates two classes and the formula is :

w T x + b = 0

Where:

* w is the weight vector (coefficients) that defines the orientation of the hyperplane.
* x is the input feature vector.
* b is the bias term (intercept).

The decision function for classifying a new input vector x is given by:

f(x) = sign(w \* x + b)

In this equation, "sign" is the sign function, which returns +1 if w T x + b is greater than or equal to 0 and -1 otherwise.

* Consider we have to find the distance from the yellow point on the line to -4,0 for finding this we extend the orange line to the point and on finding the angle between the vector (w) and line we find it to be above 90 degrees so the as per property the distance is termed as negative.
* We do the same for the point 4,3 and find the angle between the vector (w) and point to be less than 90 so the distance is positive.
* Now based on this the yellow points are classified as negative and the white points as positive

The objective of training a linear SVM is to find the optimal values of w and b that maximize the margin between the two classes while minimizing classification error.

## **Q2. What is the objective function of a linear SVM?**

The objective function of a linear Support Vector Machine (SVM) is to find the optimal values of the weight vector (w) and the bias term (b) that define the decision boundary (hyperplane) while maximizing the margin between the two classes and minimizing classification error. The objective function is typically formulated as a constrained optimization problem.

In the case of a linear SVM for binary classification, the objective function can be written as follows:

Minimize: 1/2 \* ||w||^2

Subject to the constraints: y\_i \* (w T x\_i + b) >= 1 for all training samples (x\_i, y\_i), where y\_i ∈ {-1, 1}

In this formulation:

* ||w|| represents the Euclidean norm (magnitude) of the weight vector w.
* The term 1/2 \* ||w||^2 is often referred to as the regularization term, and the objective is to minimize it. This term encourages the margin to be maximized.
* The constraints ensure that all training samples are correctly classified and lie on the correct side of the decision boundary. Specifically, it enforces that each sample (x\_i, y\_i) should be on or beyond the margin boundary, which is defined by y\_i \* (w \* x\_i + b) = 1.

The goal is to find the values of w and b that minimize the regularization term while satisfying the margin constraints.

The SVM objective function aims to strike a balance between maximizing the margin (which helps improve generalization) and minimizing classification error.

## **Q3. What is the kernel trick in SVM?**

The kernel trick is a fundamental concept in Support Vector Machines (SVMs) that allows SVMs to effectively handle non-linearly separable data by implicitly mapping the input features into a higher-dimensional space. It is a powerful technique that extends the capability of SVMs from linearly separable problems to non-linear ones.

Here's an explanation of the kernel trick:

1. **Linear Separability and Non-Linearity:**
   * In a traditional linear SVM, the algorithm seeks to find a hyperplane in the original feature space that best separates two classes of data.
   * However, many real-world datasets are not linearly separable, meaning a simple straight line or hyperplane cannot separate the classes.
2. **Kernel Functions:**
   * The kernel trick solves this problem by introducing kernel functions (also called Mercer kernels).
   * A kernel function is a mathematical function that computes the dot product between the mapped feature vectors in a higher-dimensional space without explicitly calculating the mapping itself. This allows SVMs to work in a potentially infinite-dimensional feature space without the need to store or compute the actual feature vectors in that space.
3. **Mapping to a Higher-Dimensional Space:**
   * The kernel function implicitly maps the original feature vectors into a higher-dimensional space where the data might become linearly separable.
   * Common kernel functions include the polynomial kernel (which introduces polynomial terms), the radial basis function (RBF) kernel (which creates a non-linear, Gaussian-like separation), and the sigmoid kernel, among others.
4. **Decision Boundary in the Higher-Dimensional Space:**
   * In the higher-dimensional space, the SVM aims to find a hyperplane that best separates the classes. This hyperplane can be non-linear in the original feature space but is linear in the higher-dimensional space.
5. **Predictions in Original Space:**
   * Although the SVM operates in the higher-dimensional space, it can make predictions in the original feature space by using the kernel function to compute the dot product between the new, implicitly mapped feature vectors and the decision boundary.

The types of kernel functions are :

1. Linear kernel
2. Polynomial kernel
3. Radial Basis Function (RPF) kernel
4. Sigmoid kernel

The choice of the kernel function and its parameters is essential and depends on the nature of the data. Different kernel functions have different properties, and selecting the right one can significantly impact the SVM's performance.

The kernel trick allows SVMs to handle complex decision boundaries and capture intricate relationships in the data, making them a versatile tool for classification and regression tasks, especially when the data is not linearly separable.

## **Q4. What is the role of support vectors in SVM Explain with example**

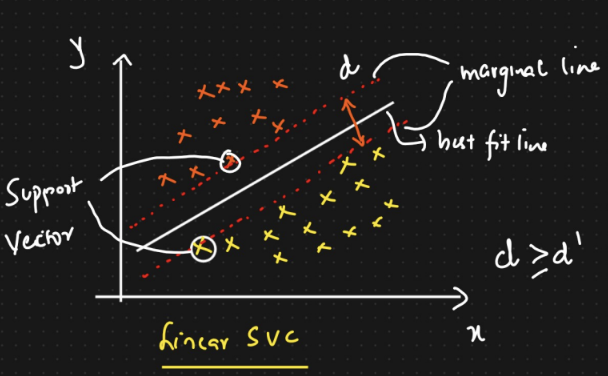
Support vectors play a crucial role in Support Vector Machines (SVMs) and are central to the SVM's ability to find an optimal hyperplane for classification or regression tasks. Support vectors are the data points that are closest to the decision boundary (hyperplane), and they have a significant influence on the placement and orientation of the hyperplane. Here's an explanation of the role of support vectors with an example:

**Role of Support Vectors:**

1. **Defining the Margin:**
   * Support vectors are the data points that lie on or within the margin of the hyperplane. The margin is the region between the two parallel hyperplanes that are equidistant from the decision boundary.
   * The support vectors define the width of the margin. The margin is maximized when the hyperplane is positioned such that it is as far away from the support vectors as possible.
2. **Supporting the Decision Boundary:**
   * The decision boundary (hyperplane) in SVM is determined by the support vectors. These vectors provide support to the hyperplane by being the closest data points from each class to it.
   * In the case of a linear SVM, the decision boundary is positioned to achieve the maximum margin while ensuring that it separates the support vectors of one class from the support vectors of the other class.

**Example:**

Let's consider a simple binary classification problem in a two-dimensional feature space. We have two classes, represented by red and blue points on a plane. The goal is to find a linear decision boundary (hyperplane) that separates these two classes.



In this example:

* The orange points and yellow points are the data points from two classes.
* The solid line represents the decision boundary (hyperplane) found by the SVM.
* The dashed lines parallel to the decision boundary represent the margin.
* The support vectors are the data points that are closest to the decision boundary and lie on or within the margin. In this case, these are the data points circled in the image.

The support vectors (circled points) play a critical role in determining the position and orientation of the decision boundary. The SVM aims to maximize the margin while ensuring that these support vectors are correctly classified. The other data points, which are not support vectors, have no influence on the placement of the decision boundary.

In summary, support vectors are the key data points that guide the SVM in finding an optimal hyperplane by defining the margin and providing support for the decision boundary. They are essential for the SVM's ability to generalize and make accurate predictions on new, unseen data.

## **Q5. Illustrate with examples and graphs of Hyperplane, Marginal plane, Soft margin and Hard margin in SVM?**

I'll provide examples and graphical representations of the concepts of Hyperplane, Marginal Plane, Soft Margin, and Hard Margin in Support Vector Machines (SVM) using a two-dimensional feature space for simplicity.

**1. Hyperplane:**

A hyperplane in SVM is a decision boundary that separates data points of different classes. In a two-dimensional feature space, a hyperplane is a straight line.

**2. Marginal Plane:**

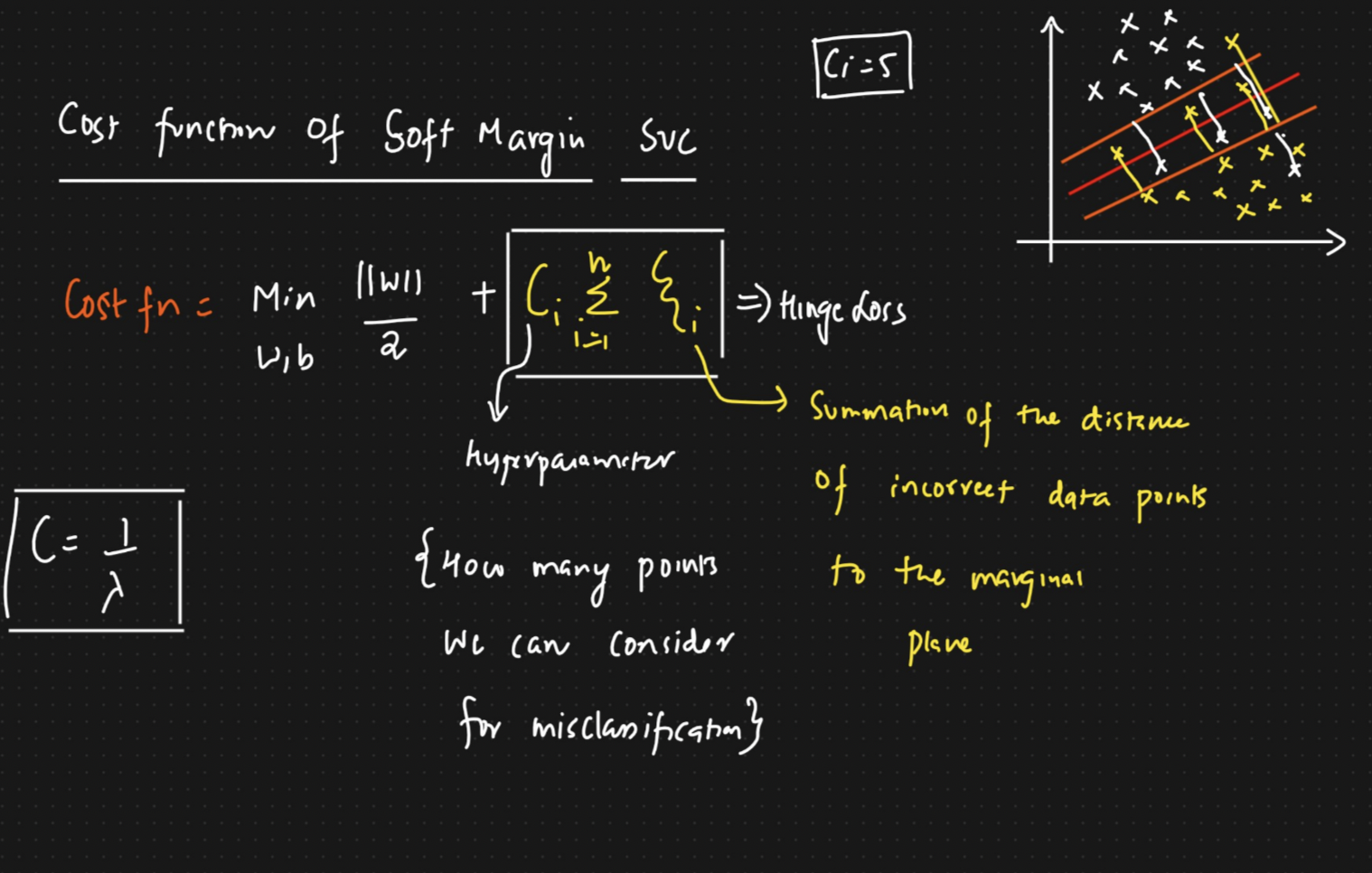
The marginal planes are the two parallel lines that run on each side of the hyperplane and define the margin in a Support Vector Machine. The margin is the region between these two planes where no data points should ideally exist. Here's an example:

**3. Hard Margin:**

In a Hard Margin SVM, the goal is to find a hyperplane that perfectly separates the two classes, with no data points allowed in the margin region.

**4. Soft Margin:**

In a Soft Margin SVM, some level of misclassification or allowing data points within the margin is permitted to find a more robust decision boundary when the data is not perfectly separable.



In real-world scenarios, data is often not perfectly separable, so Soft Margin SVMs are more commonly used. The parameter C controls the trade-off between maximizing the margin and minimizing the misclassification of data points. Smaller values of C result in a wider margin with potential misclassification, while larger values of C create a narrower margin but with fewer misclassifications. The choice of C depends on the specific problem and the nature of the data.

## **Q6. SVM Implementation through Iris dataset.**

* Load the iris dataset from the scikit-learn library and split it into a training set and a testing set
* Train a linear SVM classifier on the training set and predict the labels for the testing set
* Compute the accuracy of the model on the testing set
* Plot the decision boundaries of the trained model using two of the features
* Try different values of the regularisation parameter C and see how it affects the performance of the model.

Bonus task: Implement a linear SVM classifier from scratch using Python and compare its performance with the scikit-learn implementation.

**import** numpy **as** np

**import** pandas **as** pd

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**%matplotlib** inline

**from** sklearn.datasets **import** load\_iris

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.svm **import** SVC

**from** sklearn.metrics **import** accuracy\_score,confusion\_matrix

In [3]:

dataset **=** load\_iris()

In [5]:

print(dataset**.**DESCR)

.. \_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 [12]:

df **=** pd**.**DataFrame(dataset**.**data, columns**=**dataset**.**feature\_names)

In [14]:

df['target'] **=** dataset**.**target

In [15]:

df**.**head()

Out[15]:

|  | **sepal length (cm)** | **sepal width (cm)** | **petal length (cm)** | **petal width (cm)** | **target** |
| --- | --- | --- | --- | --- | --- |
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 |

In [16]:

df**.**tail()

Out[16]:

|  | **sepal length (cm)** | **sepal width (cm)** | **petal length (cm)** | **petal width (cm)** | **target** |
| --- | --- | --- | --- | --- | --- |
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | 2 |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | 2 |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | 2 |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 | 2 |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 | 2 |

In [20]:

*#Dependent and Independent Features*

X **=** df**.**iloc[:,:**-**1]

y **=** df**.**target

In [24]:

X**.**head()

Out[24]:

|  | **sepal length (cm)** | **sepal width (cm)** | **petal length (cm)** | **petal width (cm)** |
| --- | --- | --- | --- | --- |
| **0** | 5.1 | 3.5 | 1.4 | 0.2 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 |

In [25]:

y**.**head()

Out[25]:

0 0

1 0

2 0

3 0

4 0

Name: target, dtype: int64

In [26]:

X\_train,X\_test,y\_train,y\_test **=** train\_test\_split(X,y,test\_size**=**0.25,random\_state**=**42)

In [27]:

svm **=** SVC(kernel **=** 'linear')

In [28]:

svm**.**fit(X\_train,y\_train)

Out[28]:

SVC(kernel='linear')

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**

**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [29]:

y\_pred **=** svm**.**predict(X\_test)

In [32]:

accuracy\_score(y\_test,y\_pred)

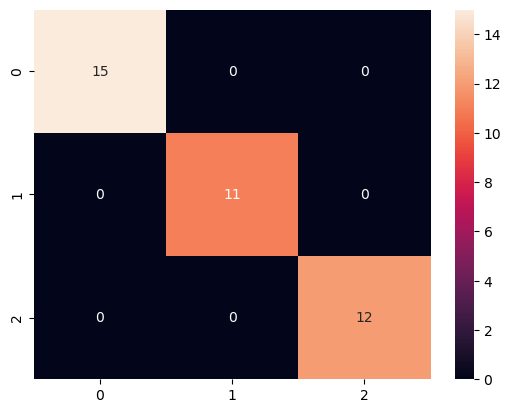
Out[32]:

1.0

In [36]:

cf **=** confusion\_matrix(y\_test,y\_pred)

sns**.**heatmap(cf,annot**=True**)

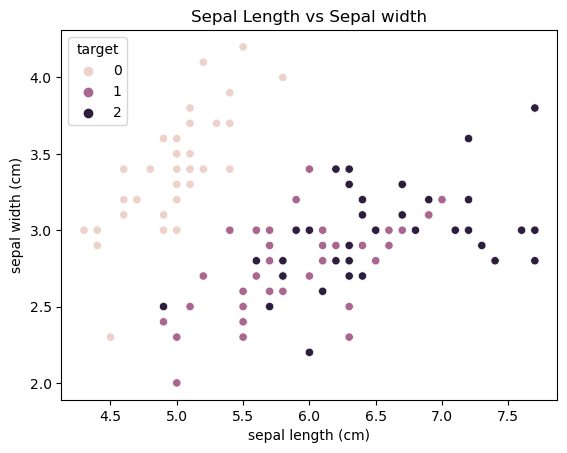


df\_train **=** pd**.**concat([X\_train,y\_train],axis**=**1)

sns**.**scatterplot(data **=** df\_train, x **=** 'sepal length (cm)',y**=**'sepal width (cm)',hue **=** y\_train)

plt**.**title('Sepal Length vs Sepal width')

plt**.**show()



**import** seaborn **as** sns

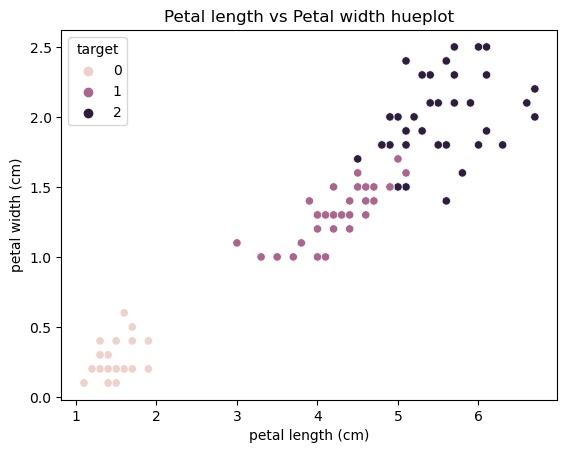
**import** matplotlib.pyplot **as** plt

df\_train **=** pd**.**concat([X\_train,y\_train],axis**=**1)

sns**.**scatterplot(data **=** df\_train, x **=** 'petal length (cm)',y**=**'petal width (cm)',hue**=** y\_train)

plt**.**title('Petal length vs Petal width hueplot')

plt**.**show()



**from** sklearn.metrics **import** classification\_report

C **=** [1,2,3,4,5,6,7,8,9,10]

**for** i **in** C:

model **=** SVC(kernel**=**'linear', C**=**i)

model**.**fit(X\_train,y\_train)

y\_pred **=** model**.**predict(X\_test)

print(f'C Value : {i}\n')

print(f'Accuracy : {accuracy\_score(y\_test,y\_pred)}\n')

print(classification\_report(y\_test,y\_pred))

print('\n-----------------------------------------\n')

C Value : 1

Accuracy : 1.0

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 1.00 1.00 11

2 1.00 1.00 1.00 12

accuracy 1.00 38

macro avg 1.00 1.00 1.00 38

weighted avg 1.00 1.00 1.00 38

-----------------------------------------

C Value : 2

Accuracy : 1.0

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 1.00 1.00 11

2 1.00 1.00 1.00 12

accuracy 1.00 38

macro avg 1.00 1.00 1.00 38

weighted avg 1.00 1.00 1.00 38

-----------------------------------------

C Value : 3

Accuracy : 1.0

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 1.00 1.00 11

2 1.00 1.00 1.00 12

accuracy 1.00 38

macro avg 1.00 1.00 1.00 38

weighted avg 1.00 1.00 1.00 38

-----------------------------------------

C Value : 4

Accuracy : 0.9736842105263158

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 0.91 0.95 11

2 0.92 1.00 0.96 12

accuracy 0.97 38

macro avg 0.97 0.97 0.97 38

weighted avg 0.98 0.97 0.97 38

-----------------------------------------

C Value : 5

Accuracy : 0.9736842105263158

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 0.91 0.95 11

2 0.92 1.00 0.96 12

accuracy 0.97 38

macro avg 0.97 0.97 0.97 38

weighted avg 0.98 0.97 0.97 38

-----------------------------------------

C Value : 6

Accuracy : 0.9736842105263158

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 0.91 0.95 11

2 0.92 1.00 0.96 12

accuracy 0.97 38

macro avg 0.97 0.97 0.97 38

weighted avg 0.98 0.97 0.97 38

-----------------------------------------

C Value : 7

Accuracy : 0.9736842105263158

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 0.91 0.95 11

2 0.92 1.00 0.96 12

accuracy 0.97 38

macro avg 0.97 0.97 0.97 38

weighted avg 0.98 0.97 0.97 38

-----------------------------------------

C Value : 8

Accuracy : 0.9736842105263158

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 0.91 0.95 11

2 0.92 1.00 0.96 12

accuracy 0.97 38

macro avg 0.97 0.97 0.97 38

weighted avg 0.98 0.97 0.97 38

-----------------------------------------

C Value : 9

Accuracy : 0.9736842105263158

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 0.91 0.95 11

2 0.92 1.00 0.96 12

accuracy 0.97 38

macro avg 0.97 0.97 0.97 38

weighted avg 0.98 0.97 0.97 38

-----------------------------------------

C Value : 10

Accuracy : 0.9736842105263158

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 0.91 0.95 11

2 0.92 1.00 0.96 12

accuracy 0.97 38

macro avg 0.97 0.97 0.97 38

weighted avg 0.98 0.97 0.97 38