Intel Student Developer Club, RCOEM

Machine Learning Bootcamp, 2022

Day - 1 : Introduction to Machine Learning 👋

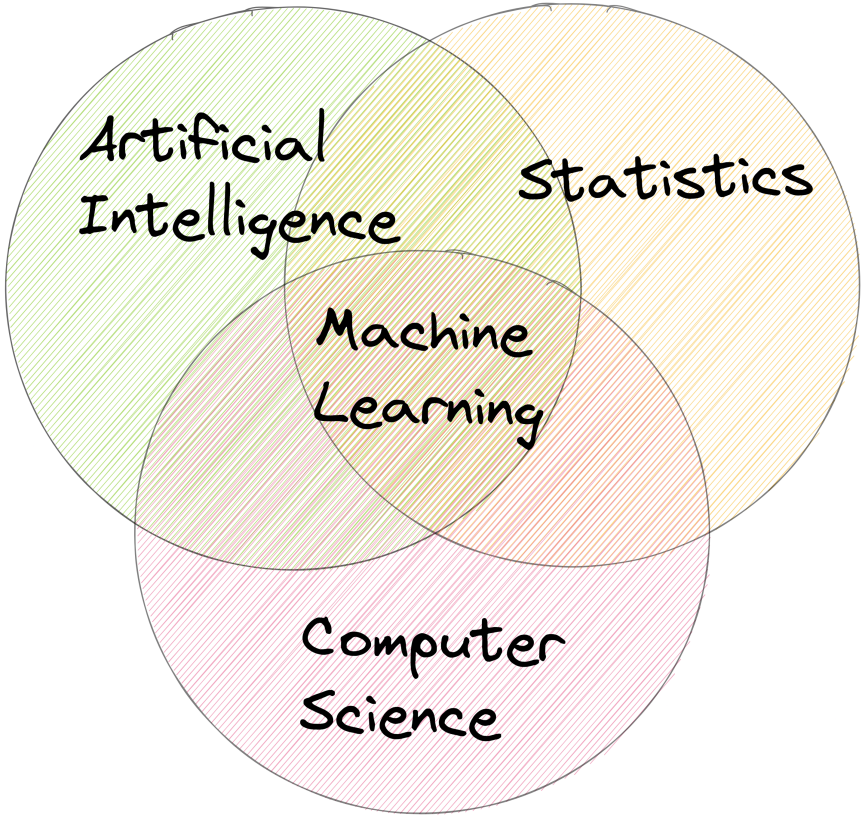
Machine Learning →

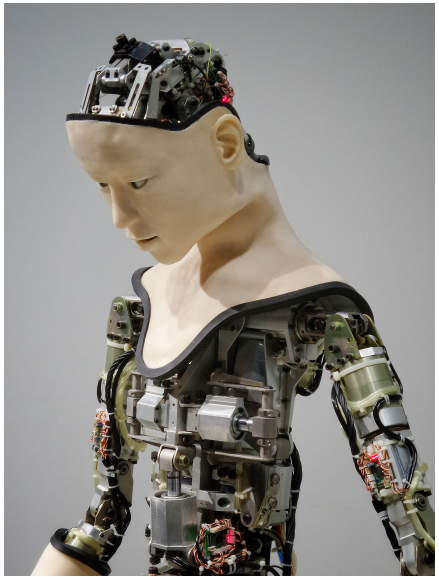
Machine learning has become one of the most important topics within development organizations that are looking for innovative ways to leverage data assets to help their business gain a new level of understanding.

> Machine learning is a form of AI that enables a system to learn from data rather than through explicit programming.
>
> ~Arthur Samuel (1959)

Like this, there are various other definitions of machine learning according to different researchers, scientists and authors.

> A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.
>
> ~ Tom Mitchell (1998)
>
> Machine Learning is about extracting knowledge from data.
>
> ~ Andreas Mueller (2012)


Artificial
Intelligence
Statistics
Machine
Learning
Computer
Science



Daily Use Cases of Machine Learning Algorithms :-

- Web search results are ranked using ML algorithms.
- Facebook's face recognition also works on machine learning models.
- ML also helps in distinguishing between spam and non-spam emails.

> The main agenda of machine learning is to develop algorithms that mimic near human neural networks / mimic how the human brain 🧠 works.

Artificial Intelligence vs. Machine Learning

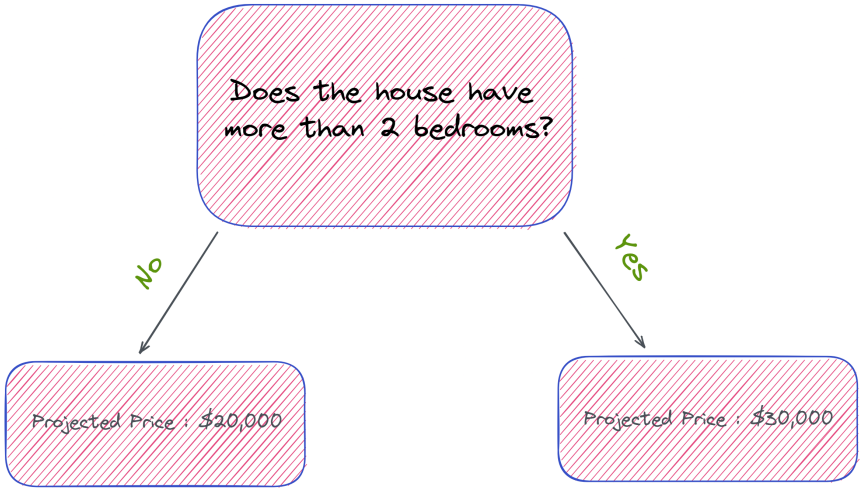
Artificial Intelligence	Machine Learning
Artificial Intelligence is a technology which enables a machine to simulate human behavior	Machine Learning is a subset of AI which allows a machine to 'learn' from past data without programming explicitly.
The goal of AI is to make a smart computer system like humans to solve complex problems	The goal of ML is to allow machines to learn from data so that they can give accurate predictions.
AI is working to create an intelligent system which can perform various complex tasks.	Machine Learning is working to create machines that can perform only those specific tasks for which they are trained.

History of ML :

Manually crafting decision rules is feasible for some applications, particularly for those in which humans have good understanding of the process to model. For example, "**spam filtering**"; in the early days of computing companies used to use hardcoded `if - else` conditions / decisions to mark an email as spam. It used to read the mail line by line and used to look for words like `sale, offers, apply now, buy now` etc., hard-coding this scenario was easy as it is human friendly.

`if - else` condition creates a decision tree model, and the algorithm computes the result based on those decisions.

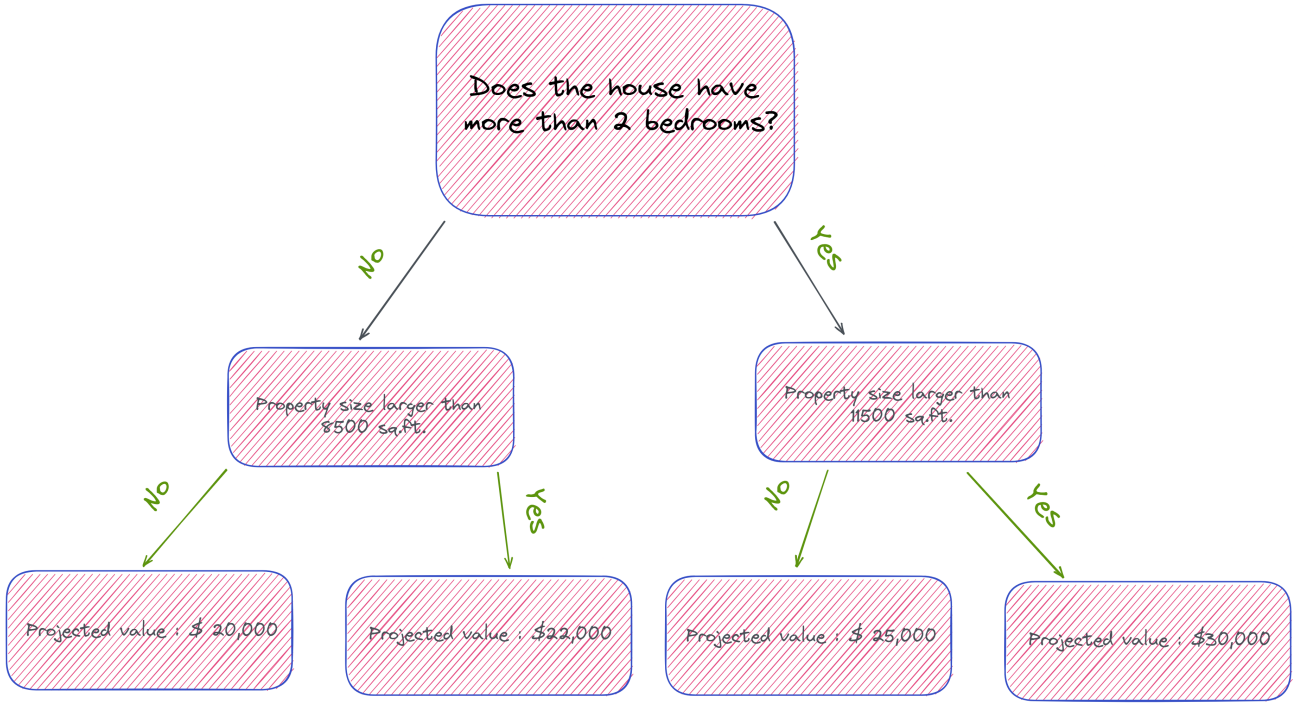
Consider a situation; you are thinking of buying a house and for that you compare lots and lots of properties, from which you infer that houses with 3 bedrooms are more expensive than a house with 2 bedrooms (well, duh!). But, there is one more thing you infer, for a property in your local area average price of a 3 BHK house is $30,000 and that of a 2 BHK is 20,000. Now let us create a decision tree based on this data.




```
if num_bedrooms() > 2:
    projected_price = 30,000
else :
    projected_price = 20,000
```

The step of capturing patterns from data is called fitting or training of a model, and the data used to fit the model is called the training data The above figure is a machine learning model in its rawest form.

Let's now upgrade our model by introducing a new feature and some new data, consider another pattern that deals with the size of the property



Types of Machine Learning Models :

1. ***Supervised Model*** : In this type of model data points have known outcome.
2. ***Unsupervised Model*** : In this type of model data points have unknown outcome.

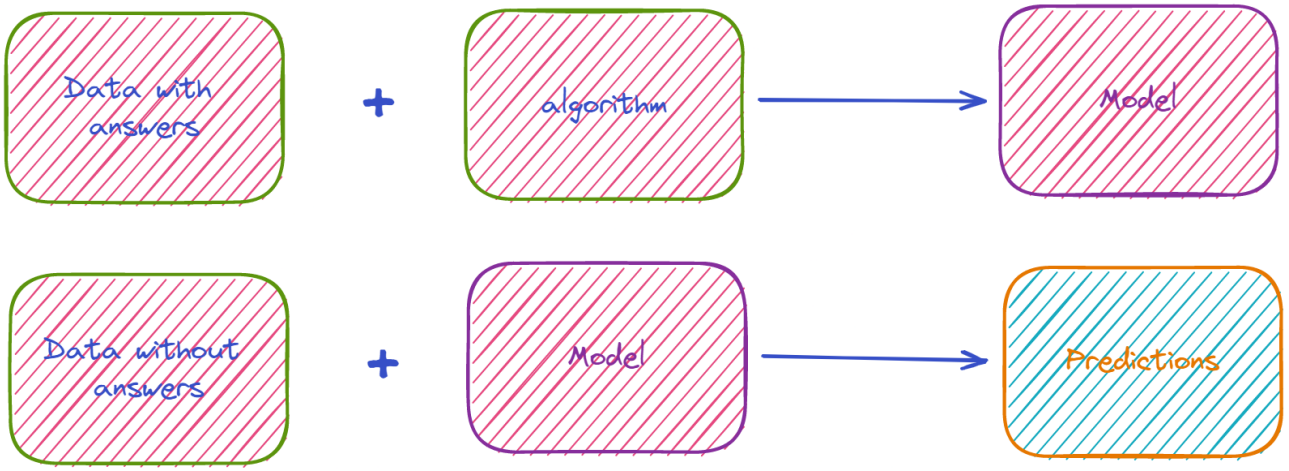
Further, there are two types of actions performed by supervised models :

1. *Classification* : In which the outcome is categorical (non-continuous)
2. *Regression* : In which the outcome is numerical (continuous)

Supervised Learning :

- One of the most commonly used and successful type of ML.
- Used whenever we want to predict a certain outcome from a given input which is new and unknown to the model.
- Our goal is to make accurate predictions, for newer, never before seen data.

⚠ Often requires human efforts to build the training set.



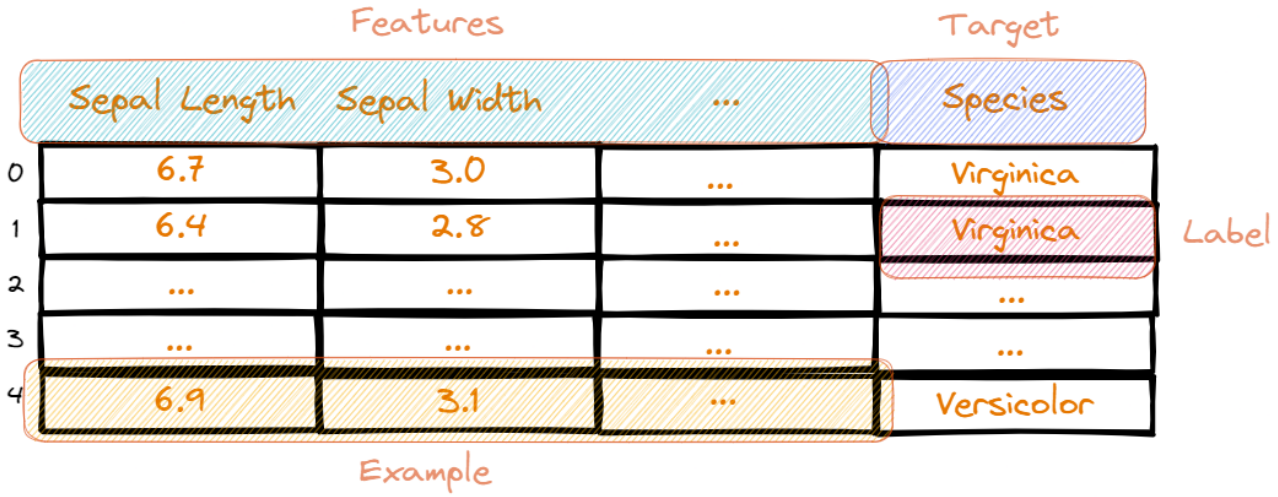
A. Classification

→ Goal is to predict a class label, which is a choice from a predefined list of possibilities.
Ex. Predicting the breed of a dog given its leg height, tail height, weight etc.

A. Regression

→ Goal is to predict a continuous number, or a floating point / real number.
Ex. Predicting a person's salary given his/her age, occupation, experience, education etc.

Machine Learning Vocabulary :



What is a model?

The above data has various values that corresponds to a specific target;

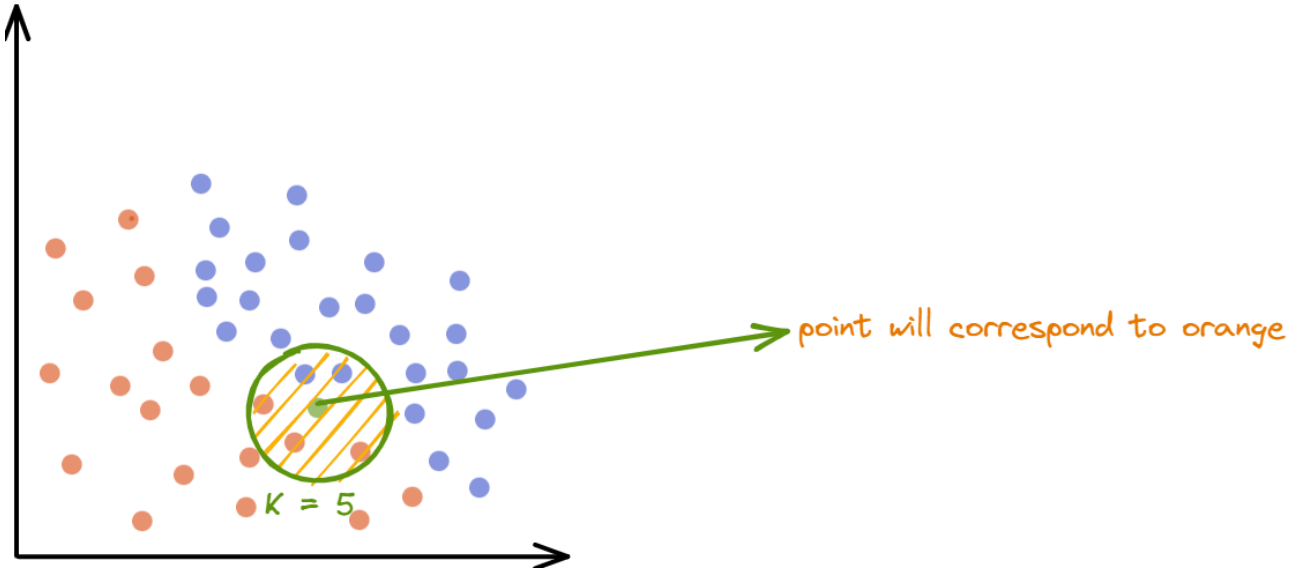
α sepal_length + β sepal_width + γ petal_length + δ petal_width = Label

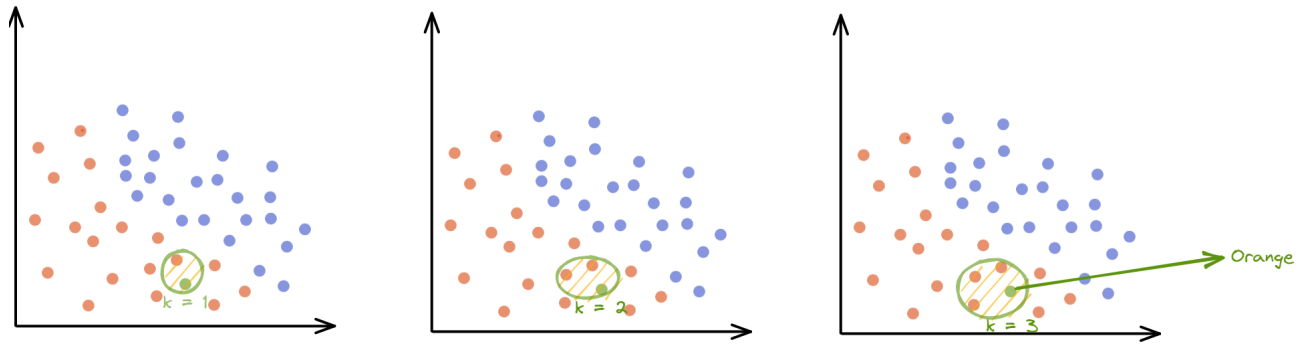
Training a model is nothing more than finding the values of α, β, γ and δ, then for any never-before seen value of features we can predict the label of the data.

Our Very First Machine Learning Model :

K - Nearest Neighbors (KNN) :

- In this model we take a new point whose labels are to be predicted and look for its nearest neighbors, `k` here is the number of neighbors we want to compare the new data point with. For ex, if there are data points which corresponds to two different classes then the new data point will correspond to the class which are more in number in the vicinity of the random point.
- We compare the new data point's distance from other points using the `Euclidian Distance Formula`;
  $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$





> ℹ A data point is classified by majority votes from its `k` nearest neighbors.

How to choose the value of k?

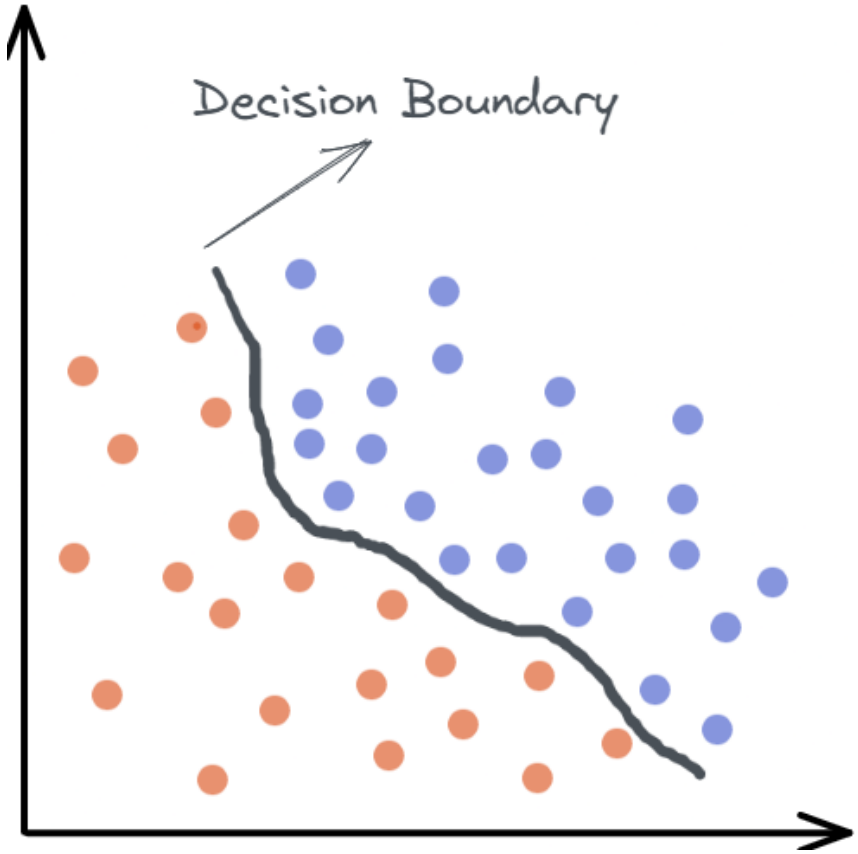
→ KNN is based on *feature similarity*, choosing the right values of `k` is a process called parameter tuning, and is important for better performance.

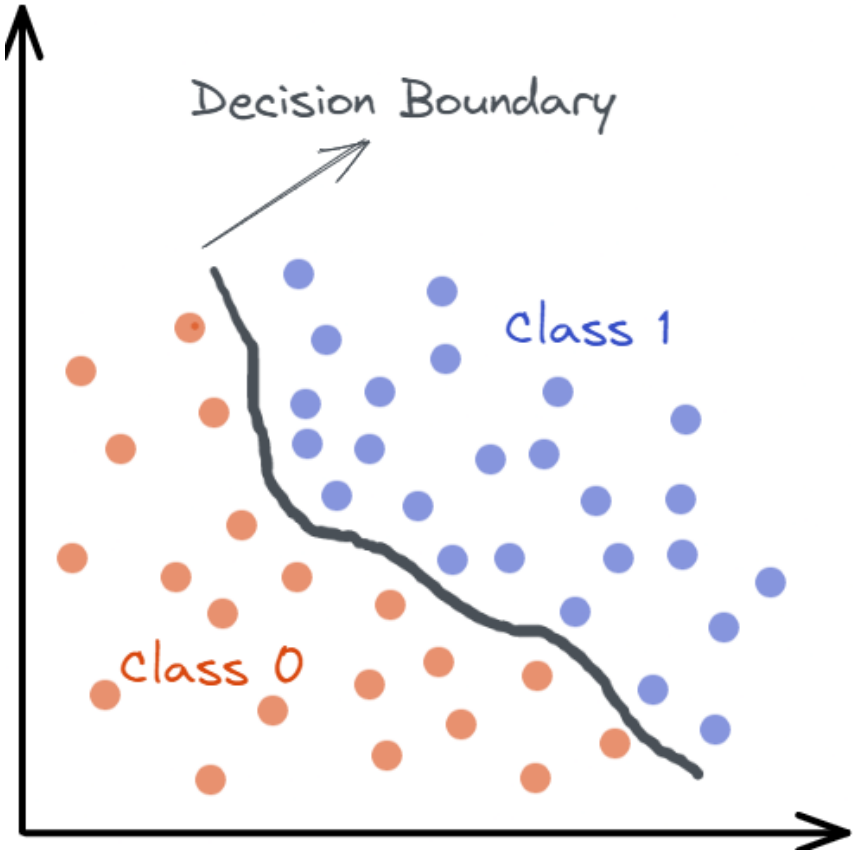
- If `k` is low, we can get skewed answers (if the data points are noisy, ie., there is no perfect cluster).
- If `k` is very large, then more number of calculations will be required to find nearest points.

Probable values of `k` can be :-

- $\sqrt{n}$, where n is the number of data points.
- Any respectable odd value, to avoid conflict or to break ties. If `k` = 2 and one neighbour is blue and other is orange, then new value can be anything. (Back to square 1!)

> Decision boundary is the line between where the algorithm will assign class 0 vs class 1.
>
> When we train the model for a particular value of `k` we are basically tracing out the decision boundary. It changes for different values of `k`.


Decision Boundary



Characteristics of KNN :

- Fast to create models as it simply stores the data.
- Slow to predict because many computations to find distance.
- Can require lots of memory if data is large.

KNN Summarised :

- A positive integer (preferably odd) is specified, along with a new sample.
- We select the `k` entries in our database which are closest to the new sample.
- We find the most common classification of these entries [Voting]
- This classification is given to the new sample.

Enough Theory, Hands-on Time!

Step 1 :

→ Importing the required libraries and data

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_iris  # Importing our dataset to create model
```

Step 2:

→ Exploring the Data

In [ ]:
```
iris = load_iris()  # Creating an object of the dataset

iris
```

Out[ ]:
```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
```

```
[5.4, 3.4, 1.7, 0.2],
[5.1, 3.7, 1.5, 0.4],
[4.6, 3.6, 1. , 0.2],
[5.1, 3.3, 1.7, 0.5],
[4.8, 3.4, 1.9, 0.2],
[5. , 3. , 1.6, 0.2],
[5. , 3.4, 1.6, 0.4],
[5.2, 3.5, 1.5, 0.2],
[5.2, 3.4, 1.4, 0.2],
[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
```

```
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
```

```
        [6.5, 3.2, 5.1, 2. ],
        [6.4, 2.7, 5.3, 1.9],
        [6.8, 3. , 5.5, 2.1],
        [5.7, 2.5, 5. , 2. ],
        [5.8, 2.8, 5.1, 2.4],
        [6.4, 3.2, 5.3, 2.3],
        [6.5, 3. , 5.5, 1.8],
        [7.7, 3.8, 6.7, 2.2],
        [7.7, 2.6, 6.9, 2.3],
        [6. , 2.2, 5. , 1.5],
        [6.9, 3.2, 5.7, 2.3],
        [5.6, 2.8, 4.9, 2. ],
        [7.7, 2.8, 6.7, 2. ],
        [6.3, 2.7, 4.9, 1.8],
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
```

```
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number
of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 numeric, predictive attributes and the clas
s\n    :Attribute Information:\n        - sepal length in cm\n        - sepal width in cm\n        - petal length in cm\n
- petal width in cm\n        - class:\n                - Iris-Setosa\n                - Iris-Versicolour\n
- Iris-Virginica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== ==============
======\n                    Min  Max   Mean    SD   Class Correlation\n    ============== ==== ==== ======= ===== =======
=============\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194
\n    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (h
igh!)\n    ============== ==== ==== ======= ===== ====================\n\n    :Missing Attribute Values: None\n    :Class
Distribution: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.
nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom F
isher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong
data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature.  Fisher\'s pa
per is a classic in the field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set
contains 3 classes of 50 instances each, where each class refers to a\ntype of iris plant.  One class is linearly separab
le from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher, R.
A. "The use of multiple measurements in taxonomic problems"\n     Annual Eugenics, 7, Part II, 179-188 (1936); also in "C
ontributions to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Clas
sification and Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy,
B.V. (1980) "Nosing Around the Neighborhood: A New System\n     Structure and Classification Rule for Recognition in Part
ially Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, N
o. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n     on Information Theor
y, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clust
ering system finds 3 classes in the data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'c:\\Python39\\lib\\site-packages\\sklearn\\datasets\\data\\iris.csv'}
```

In [ ]:

```
iris.DESCR  # We can get all the information about our dataset using this method
```

Out[ ]:

```
'.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instan
ces: 150 (50 in each of three classes)\n    :Number of Attributes: 4 numeric, predictive attributes and the class\n    :A
ttribute Information:\n        - sepal length in cm\n        - sepal width in cm\n        - petal length in cm\n        -
petal width in cm\n        - class:\n                - Iris-Setosa\n                - Iris-Versicolour\n                -
Iris-Virginica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== ================
====\n                    Min  Max   Mean    SD   Class Correlation\n    ============== ==== ==== ======= ===== =========
==========\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
```

```
\n    ============== ==== ==== ======= ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Distribution: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature.  Fisher\'s paper is a classic in the field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype of iris plant.  One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n     Structure and Classification Rule for Recognition in Partially Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n     on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classes in the data.\n   - Many, many more ...'
```

In [ ]:
```
iris.feature_names  # Features / columns of the dataset
```

Out[ ]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [ ]:
```
iris.target_names   # Target / label values of the dataset
```

Out[ ]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [ ]:
```
# <U10 -> 10 character unicode string
```

In [ ]:
```
iris.data   # All the column values of our dataset in the form of a 2D list
```

Out[ ]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
```

```
[4.9, 3.1, 1.5, 0.1],
[5.4, 3.7, 1.5, 0.2],
[4.8, 3.4, 1.6, 0.2],
[4.8, 3. , 1.4, 0.1],
[4.3, 3. , 1.1, 0.1],
[5.8, 4. , 1.2, 0.2],
[5.7, 4.4, 1.5, 0.4],
[5.4, 3.9, 1.3, 0.4],
[5.1, 3.5, 1.4, 0.3],
[5.7, 3.8, 1.7, 0.3],
[5.1, 3.8, 1.5, 0.3],
[5.4, 3.4, 1.7, 0.2],
[5.1, 3.7, 1.5, 0.4],
[4.6, 3.6, 1. , 0.2],
[5.1, 3.3, 1.7, 0.5],
[4.8, 3.4, 1.9, 0.2],
[5. , 3. , 1.6, 0.2],
[5. , 3.4, 1.6, 0.4],
[5.2, 3.5, 1.5, 0.2],
[5.2, 3.4, 1.4, 0.2],
[4.7, 3.2, 1.6, 0.2],
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
```

```
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
```

```
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
[6.7, 3.3, 5.7, 2.1],
[7.2, 3.2, 6. , 1.8],
[6.2, 2.8, 4.8, 1.8],
[6.1, 3. , 4.9, 1.8],
[6.4, 2.8, 5.6, 2.1],
[7.2, 3. , 5.8, 1.6],
[7.4, 2.8, 6.1, 1.9],
[7.9, 3.8, 6.4, 2. ],
[6.4, 2.8, 5.6, 2.2],
[6.3, 2.8, 5.1, 1.5],
[6.1, 2.6, 5.6, 1.4],
[7.7, 3. , 6.1, 2.3],
[6.3, 3.4, 5.6, 2.4],
[6.4, 3.1, 5.5, 1.8],
[6. , 3. , 4.8, 1.8],
[6.9, 3.1, 5.4, 2.1],
[6.7, 3.1, 5.6, 2.4],
[6.9, 3.1, 5.1, 2.3],
[5.8, 2.7, 5.1, 1.9],
[6.8, 3.2, 5.9, 2.3],
```

```
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

In [ ]:
```
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)

iris_df     # Creating a dataframe from the given raw data
```

Out[ ]:

	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
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [ ]:
```
iris_df['species'] = pd.DataFrame(iris.target)  # Creating a column that will contain the target values of all the entrie

iris_df
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
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
...	...	...	...	...	...
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

150 rows × 5 columns

In [ ]:

```
iris_df['flower name'] = iris_df.species.apply(lambda x: iris.target_names[x])  # giving names according to the target va

# 0 -> Iris - Setosa
# 1 -> Iris - Versicolor
# 2 -> Iris - Virginica

iris_df
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	flower name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2	virginica

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	flower name
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

150 rows × 6 columns

In [ ]:
```
iris_df.index = pd.Series(range(1, 151))    # changing index values for dataframe

iris_df
```

Out[ ]:

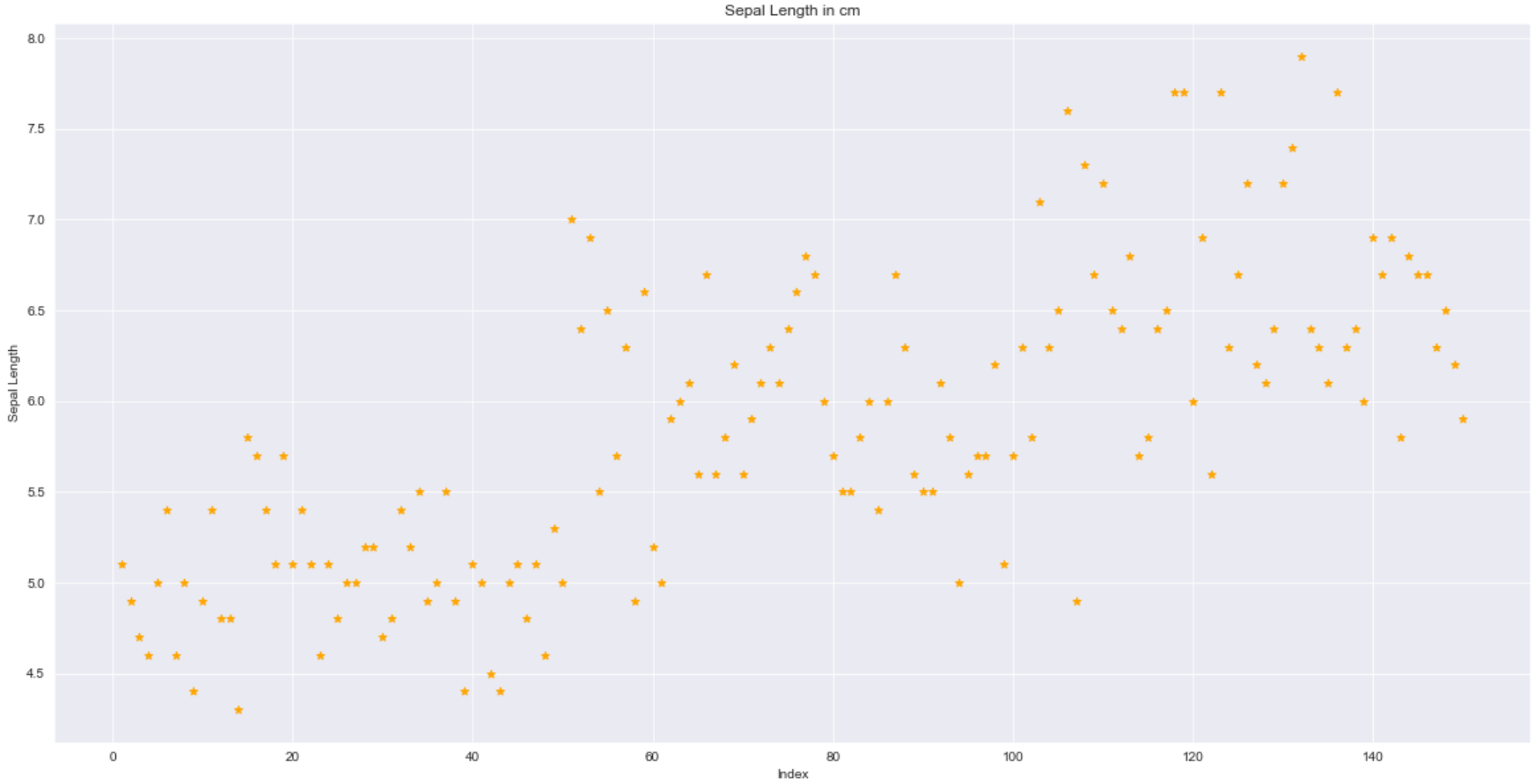
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	flower name
1	5.1	3.5	1.4	0.2	0	setosa
2	4.9	3.0	1.4	0.2	0	setosa
3	4.7	3.2	1.3	0.2	0	setosa
4	4.6	3.1	1.5	0.2	0	setosa
5	5.0	3.6	1.4	0.2	0	setosa
...	...	...	...	...	...	...
146	6.7	3.0	5.2	2.3	2	virginica
147	6.3	2.5	5.0	1.9	2	virginica
148	6.5	3.0	5.2	2.0	2	virginica
149	6.2	3.4	5.4	2.3	2	virginica
150	5.9	3.0	5.1	1.8	2	virginica

150 rows × 6 columns

Now that we have analysed the data and converted it into a dataframe to make it user-friendly, we can now try to visualize this data.

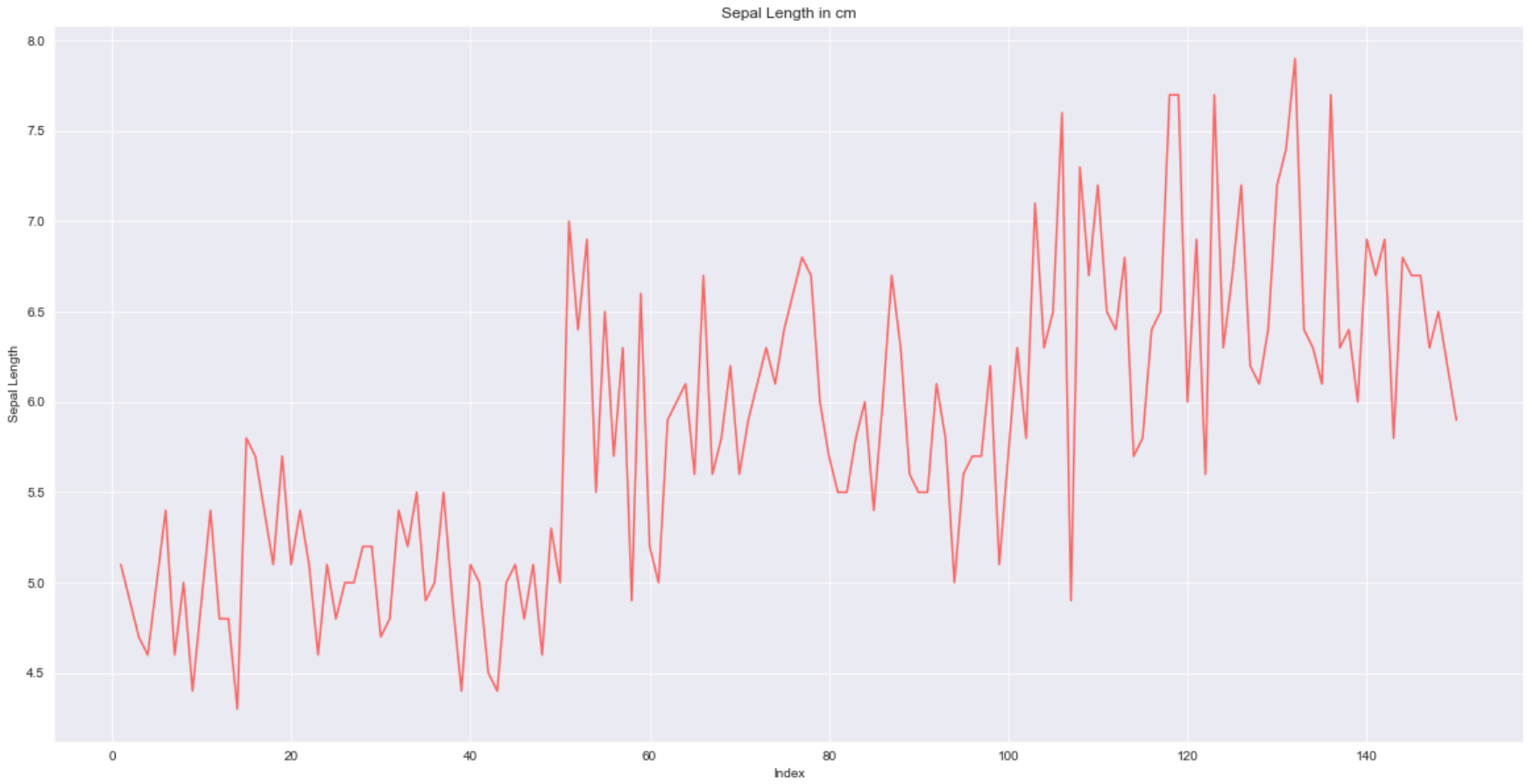
In [ ]:
```
sns.set_style('darkgrid')
colors = ['#003f5c', '#58508d', '#bc5090', '#ff6361', '#ffa600']
```

In [ ]:
```
plt.figure(figsize=(20,10))

plt.scatter(iris_df.index, iris_df['sepal length (cm)'], color = colors[4], marker='*')
plt.title('Sepal Length in cm')
plt.xlabel('Index')
plt.ylabel('Sepal Length')

plt.show()
```



In [ ]:
```
plt.figure(figsize=(20,10))
```

```
plt.plot(iris_df.index, iris_df['sepal length (cm)'], color = colors[3])
plt.title('Sepal Length in cm')
plt.xlabel('Index')
plt.ylabel('Sepal Length')

plt.show()
```

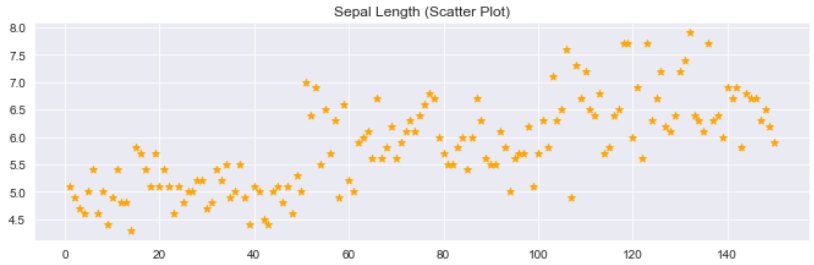


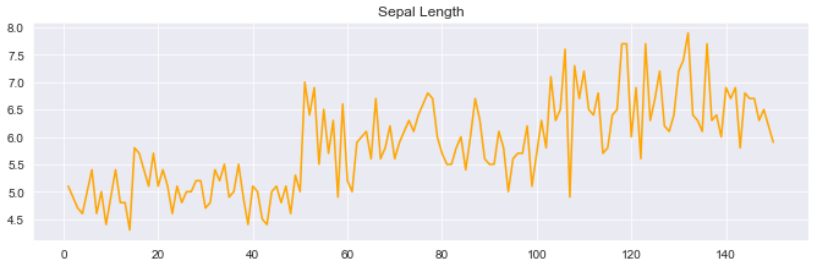
In [ ]:

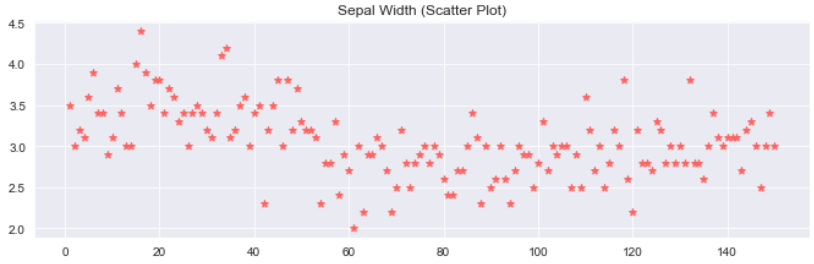
```
plt.figure(figsize=(20,10))

fig, ax = plt.subplots(4, 2, figsize=(25,15))

ax[0,0].scatter(iris_df.index, iris_df['sepal length (cm)'], color=colors[4], marker='*')
ax[0,0].set_title('Sepal Length (Scatter Plot)')

ax[0,1].plot(iris_df.index, iris_df['sepal length (cm)'], color=colors[4])
```

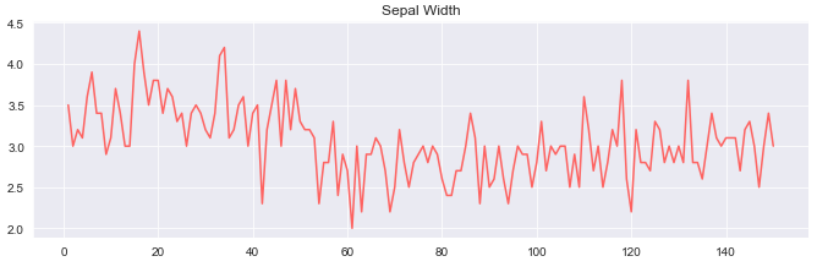
```
ax[0,1].set_title('Sepal Length')

ax[1,0].scatter(iris_df.index, iris_df['sepal width (cm)'], color=colors[3], marker='*')
ax[1,0].set_title('Sepal Width (Scatter Plot)')

ax[1,1].plot(iris_df.index, iris_df['sepal width (cm)'], color=colors[3])
ax[1,1].set_title('Sepal Width')

ax[2,0].scatter(iris_df.index, iris_df['petal length (cm)'], color=colors[2], marker='*')
ax[2,0].set_title('Petal Length (Scatter Plot)')

ax[2,1].plot(iris_df.index, iris_df['petal length (cm)'], color=colors[2])
ax[2,1].set_title('Petal Length')

ax[3,0].scatter(iris_df.index, iris_df['petal width (cm)'], color=colors[1], marker='*')
ax[3,0].set_title('Petal Width (Scatter Plot)')

ax[3,1].plot(iris_df.index, iris_df['petal width (cm)'], color=colors[1])
ax[3,1].set_title('Petal Width')

plt.show()
```

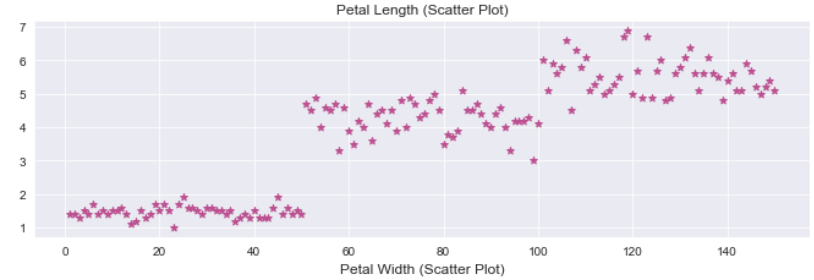
```
<Figure size 1440x720 with 0 Axes>
```

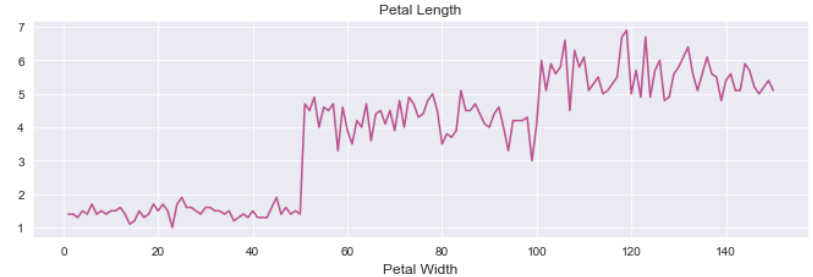


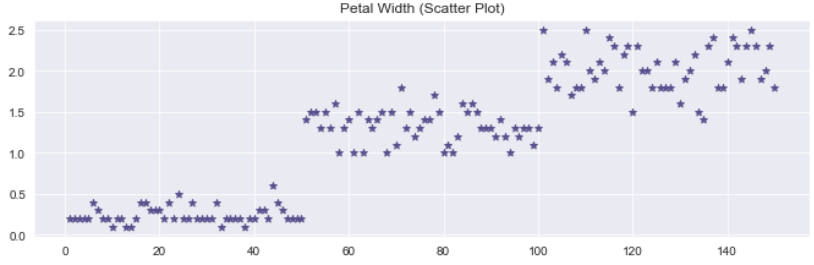


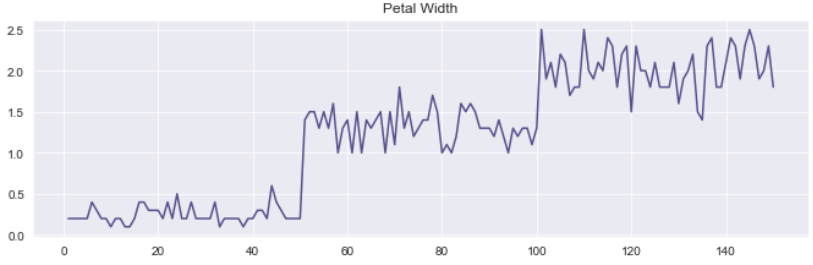












In [ ]:
```
iris_df.sample(n=10)
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	flower name
91	5.5	2.6	4.4	1.2	1	versicolor
7	4.6	3.4	1.4	0.3	0	setosa
72	6.1	2.8	4.0	1.3	1	versicolor
106	7.6	3.0	6.6	2.1	2	virginica

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	flower name
146	6.7	3.0	5.2	2.3	2	virginica
62	5.9	3.0	4.2	1.5	1	versicolor
46	4.8	3.0	1.4	0.3	0	setosa
29	5.2	3.4	1.4	0.2	0	setosa
80	5.7	2.6	3.5	1.0	1	versicolor
87	6.7	3.1	4.7	1.5	1	versicolor

Let's now train our first model!!!

In [ ]:

```
# First import the required tools

from sklearn.model_selection import train_test_split    # Since our data is sufficiently big we will use some of its part
from sklearn.neighbors import KNeighborsClassifier      # As this is a classification problem we will classifier not regr
```

In [ ]:

```
X = iris_df.drop(['species', 'flower name'], axis=1)
y = iris_df.species

print(f'Example Data : \n{X}\n-----------\nLabels : \n{y}')
```

```
Example Data :
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
1                  5.1               3.5                1.4               0.2
2                  4.9               3.0                1.4               0.2
3                  4.7               3.2                1.3               0.2
4                  4.6               3.1                1.5               0.2
5                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
146                6.7               3.0                5.2               2.3
147                6.3               2.5                5.0               1.9
148                6.5               3.0                5.2               2.0
149                6.2               3.4                5.4               2.3
150                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
-----------
Labels :
```

```
1      0
2      0
3      0
4      0
5      0
      ..
146    2
147    2
148    2
149    2
150    2
Name: species, Length: 150, dtype: int32
```

In [ ]:
```
# We will now split the data in training and testing part, training data will be used to train the model and testing data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)   # test_size=0.25 attribute te
```

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=3)   # n_neighbors assigns the value of `k`

knn.fit(X_train, y_train)    # Let us now fit the data and train our model
```

Out[ ]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [ ]:
```
# Let us now test the accuracy of the model for 3 neighbors

knn.score(X_test, y_test) * 100
```

Out[ ]:
```
97.36842105263158
```

In [ ]:
```
iris_df.sample()
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	flower name
96	5.7	3.0	4.2	1.2	1	versicolor

In [ ]:
```
# Now let us manually test the model with a unique data

y_prediction = knn.predict([[5.6321, 3.1232, 4.3399, 1.23423]])  # Instead of this value (which was taken while curating
```

```
print(f"Predicted species code : {y_prediction[0]}\nPredicted species name : {iris.target_names[y_prediction[0]]}")
```

```
Predicted species code : 1
Predicted species name : versicolor
```

In [ ]:
```
# There is another way to compute the accuracy of our model
from sklearn.metrics import accuracy_score

y_prediction = knn.predict(X_test)
accuracy_score(y_test, y_prediction)*100
```

Out[ ]:
```
97.36842105263158
```

For a different value of neighbors let us train and test our model

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=int(math.sqrt(150)))   # n_neighbors assigns the value of `k`
# We must take the recommended value, ie. sqrt(150)

knn.fit(X_train, y_train)    # Let us now fit the data and train our model
```

Out[ ]:
```
KNeighborsClassifier(n_neighbors=12)
```

In [ ]:
```
# Let us now test the accuracy of the model for 12 neighbors

knn.score(X_test, y_test) * 100
```

Out[ ]:
```
97.36842105263158
```

🎉 🎉 🎉 Congratulations!!! 🎉 🎉 🎉

You Just Created Your Own Machine Learning Model!!!

🎉 🎉 🎉

In [ ]: