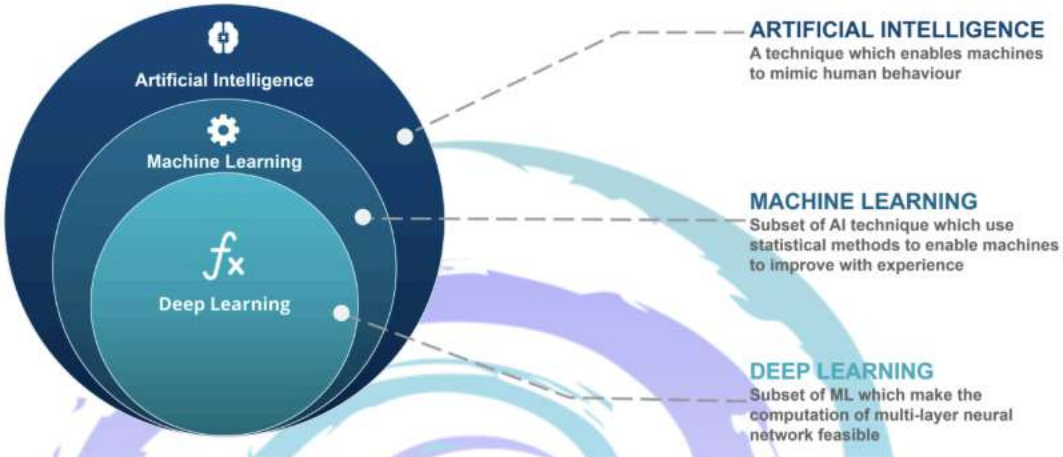
Machine Learning

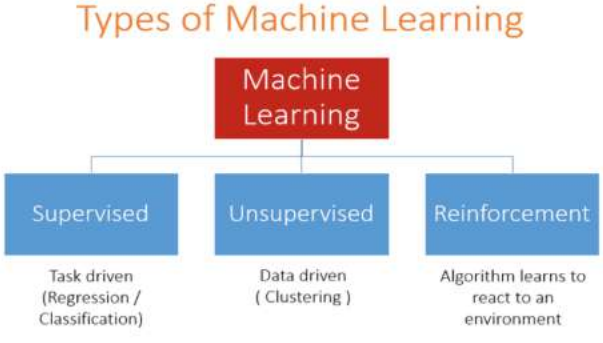
Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.
Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.



The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.
The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Some machine learning methods

Machine learning algorithms are often categorised as supervised or unsupervised.



Supervised machine learning :
Supervised machine learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events.
Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values.
The system is able to provide targets for any new input after sufficient training.
The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

Unsupervised machine learning :
In contrast, unsupervised machine learning algorithms are used when the information used to train is neither classified nor labeled.
Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data.
The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

Semi-supervised machine learning :
Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data.
The systems that use this method are able to considerably improve learning accuracy.
Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it.
Otherwise, acquiringunlabeled data generally doesn't require additional resources.

Reinforcement machine learning :
Reinforcement machine learning algorithms is a learning method that interacts with its environment by producing actions and discovers errors or rewards.
Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning.
This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance.
Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.
Machine learning enables analysis of massive quantities of data.
While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly.
Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

Data Science

Data Science

Data Science is the study of extracting **useful information** and **insights** from data using **mathematics, statistics, computer programming**, and domain knowledge.

Fields of Data Science

Data Science connects many different fields:

- **Statistics** – for analyzing numbers.
- **Machine Learning** – for making predictions.
- **Data Visualization** – for making charts and graphs.
- **Big Data** – for handling huge amounts of data.
- **Data Engineering** – for building systems to collect and store data.
- **Domain Knowledge** – understanding the field like finance, health, sports, etc.

Data :

Data means **raw facts or figures**. Data is meaningless unless we understand and process it.

Example:

- "21" is data. (Is it age, temperature, or score?)

Information :

When data is **organized and given meaning**, it becomes **information**.

Example:

- "Amit is 21 years old" — Now it's useful and meaningful = **information**.

Types of Data

Type	Description	Example
Structured	Organized in rows & columns	Excel sheet, databases
Unstructured	No fixed format	Images, videos, emails
Semi-structured	Some structure, not full table	JSON, XML files

Artificial Intelligence (AI) :

AI is the ability of a machine to **think, learn, and act** like humans.

Example:

- Alexa answering your questions.
- Google Maps giving directions.

Machine Learning (ML) :

Machine Learning is a part of AI that teaches machines to **learn from data and improve automatically without being told what to do every time**.

Simple example:

- If we show 1000 pictures of cats and dogs, a machine can **learn** to identify them on its own.

Dataset :

A dataset is a **collection of related data** used for analysis or training a machine.

Example:

Name	Age	Gender	Result
Ram	21	Male	Pass
Laxmi	23	Female	Pass

This table is a **dataset**.

Types of Datasets in ML

- **Training Dataset** – used to teach the machine.
- **Testing Dataset** – used to check the accuracy of the model.

Features and Labels

- **Features** – Input data used to predict something.
- **Label** – The output or result we want to predict.

Example:

Height (cm)	Weight (kg)	Gender
170	60	Male

- Features = Height, Weight
- Label = Gender

Types of Machine Learning

1. Supervised Learning

- You give input **and** the correct output (label).
- Machine learns from examples.

Example: Predicting house prices, classifying emails as spam.

2. Unsupervised Learning

- You give only input, **no correct answers**.
- Machine finds patterns.

Example: Grouping similar customers for marketing.

3. Reinforcement Learning

- Machine learns by **trial and error** using rewards and punishment.

Example: Self-driving cars

Supervised VS Unsupervised Machine Learning

Supervised learning:
Supervised learning is the learning of the model where with input variable (say, x) and an output variable (say, Y) and an algorithm to map the input to the output.

That is, Y = f(X)

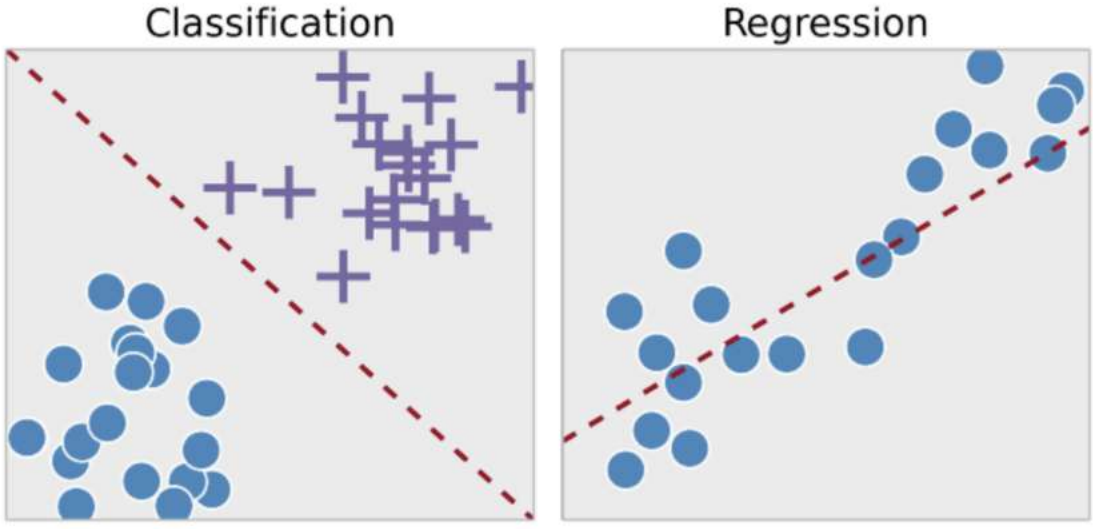
Why supervised learning?
The basic aim is to approximate the mapping function(mentioned above) so well that when there is a new input data (x) then the corresponding output variable can be predicted.
It is called supervised learning because the process of an learning(from the training dataset) can be thought of as a teacher who is supervising the entire learning process. Thus, the "learning algorithm" iteratively makes predictions on the training data and is corrected by the "teacher", and the learning stops when the algorithm achieves an acceptable level of performance(or the desired accuracy).

Example of Supervised Learning

Suppose there is a basket which is filled with some fresh fruits, the task is to arrange the same type of fruits at one place.
Also, suppose that the fruits are apple, banana, cherry, grape.
Suppose one already knows from their previous work (or experience) that, the shape of each and every fruit present in the basket so, it is easy for them to arrange the same type of fruits in one place.
Here, the previous work is called as training data in Data Mining terminology. So, it learns the things from the training data. This is because it has a response variable which says y that if some fruit has so and so features then it is grape, and similarly for each and every fruit.
This type of information is deciphered from the data that is used to train the model.
This type of learning is called Supervised Learning.
Such problems are listed under classical Classification Tasks.

Types of Supervised Machine Learning Techniques

Regression:



Regression technique predicts a single output value using training data.
Example: You can use regression to predict the house price from training data. The input variables will be locality, size of a house, etc.

Classification:
Classification means to group the output inside a class. If the algorithm tries to label input into two distinct classes, it is called binary classification. Selecting between more than two classes is referred to as multiclass classification.

Example: Determining whether or not someone will be a defaulter of the loan.

Strengths: Outputs always have a probabilistic interpretation, and the algorithm can be regularized to avoid overfitting.

Weaknesses: Logistic regression may underperform when there are multiple or non-linear decision boundaries. This method is not flexible, so it does not capture more complex relationships.

Unsupervised Learning:
Unsupervised learning is where only the input data (say, X) is present and no corresponding output variable is there.

Why Unsupervised Learning?
The main aim of Unsupervised learning is to model the distribution in the data in order to learn more about the data.
It is called so, because there is no correct answer and there is no such teacher(unlike supervised learning). Algorithms are left to their own devises to discover and present the interesting structure in the data.

Example of Unsupervised Learning

Again, Suppose there is a basket and it is filled with some fresh fruits. The task is to arrange the same type of fruits at one place.
This time there is no information about those fruits beforehand, its the first time that the fruits are being seen or discovered
So how to group similar fruits without any prior knowledge about those.
First, any physical characteristic of a particular fruit is selected. Suppose color.

Then the fruits are arranged on the basis of the color. The groups will be something as shown below:
RED COLOR GROUP: apples & cherry fruits.
GREEN COLOR GROUP: bananas & grapes.
So now, take another physical character say, size, so now the groups will be something like this.

RED COLOR AND BIG SIZE: apple.
RED COLOR AND SMALL SIZE: cherry fruits.
GREEN COLOR AND BIG SIZE: bananas.
GREEN COLOR AND SMALL SIZE: grapes.

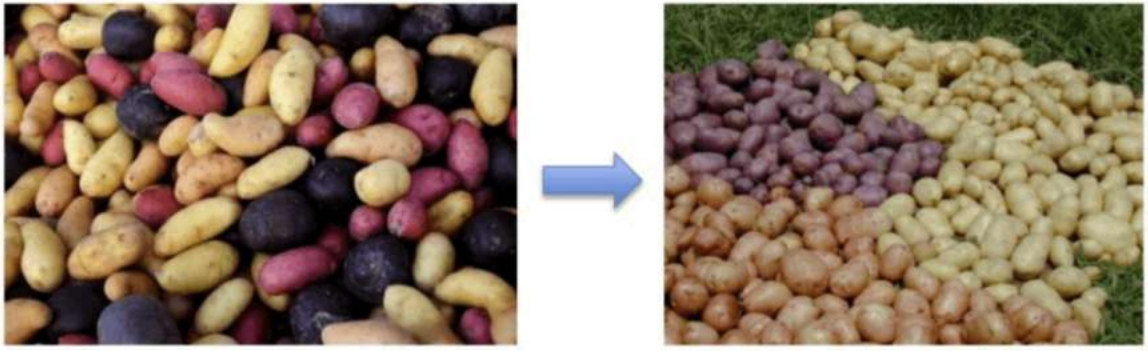
Here, there is no need to know or learn anything beforehand. That means, no train data and no response variable. This type of learning is known as Unsupervised Learning.

Types of Unsupervised Machine Learning Techniques

Unsupervised learning problems further grouped into clustering and association problems.

Clustering

Clustering is an important concept when it comes to unsupervised learning. It mainly



sample Cluster/group

deals with finding a structure or pattern in a collection of uncategorized data. Clustering algorithms will process your data and find natural clusters(groups) if they exist in the data. You can also modify how many clusters your algorithms should identify. It allows you to adjust the granularity of these groups.

Association

Association rules allow you to establish associations amongst data objects inside large databases. This unsupervised technique is about discovering exciting relationships between variables in large databases. For example, people that buy a new home most likely to buy new furniture.

Other Examples:

- A subgroup of cancer patients grouped by their gene expression measurements
- Groups of shopper based on their browsing and purchasing histories
- Movie group by the rating given by movies viewers

Difference between Supervised & Unsupervised Machine Learning

Parameters	Supervised machine learning technique	Unsupervised machine learning technique
Process	In a supervised learning model, input and output variables will be given.	In unsupervised learning model, only input data will be given
Input Data	Algorithms are trained using labeled data.	Algorithms are used against data which is not labeled
Algorithms Used	Support vector machine, Neural network, Linear and logistics regression, random forest, and Classification trees.	Unsupervised algorithms can be divided into different categories: like Cluster algorithms, K-means, Hierarchical clustering, etc.
Computational Complexity	Supervised learning is a simpler method.	Unsupervised learning is computationally complex
Use of Data	Supervised learning model uses training data to learn a link between the input and the outputs.	Unsupervised learning does not use output data.
Accuracy of Results	Highly accurate and trustworthy method.	Less accurate and trustworthy method.
Real Time Learning	Learning method takes place offline.	Learning method takes place in real time.
Number of Classes	Number of classes is known.	Number of classes is not known.
Main Drawback	Classifying big data can be a real challenge in Supervised Learning.	You cannot get precise information regarding data sorting, and the output as data used in unsupervised learning is labeled and not known.

Summary

- In Supervised learning, you train the machine using data which is well "labeled."
- Unsupervised learning is a machine learning technique, where you do not need to supervise the model.
- Supervised learning allows you to collect data or produce a data output from the previous experience.
- Unsupervised machine learning helps you to finds all kind of unknown patterns in data.
- For example, you will able to determine the time taken to reach back come base on weather condition, Times of the day and holiday.
- For example, Baby can identify other dogs based on past supervised learning.
- Regression and Classification are two types of supervised machine learning techniques.
- Clustering and Association are two types of Unsupervised learning.
- In a supervised learning model, input and output variables will be given while with unsupervised learning model, only input data will be given.

Pandas Python Library

Pandas is a software library written for the Python programming language for data manipulation and analysis.
In particular, it offers data structures and operations for manipulating numerical tables and time series.
It is free software released under the three-clause BSD license.
The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

This is the most important library for Machine Learning because we can deal with our data set by using Pandas.

Install Pandas library using pip as

sudo pip install pandas

Features of Pandas

- DataFrame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and subsetting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation[4] and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.
- Provides data filtration.

Pandas deals with the following three data structures –

- Series
- DataFrame
- Panel

These data structures are built on top of Numpy array, which means they are fast.

Dimension & Description

The best way to think of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure.
For example, DataFrame is a container of Series, Panel is a container of DataFrame.

Data Structure	Dimensions	Description
Series	1	1D labeled homogeneous array, sizeimmutable.
Data Frames	2	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	General 3D labeled, size-mutable array.

Series

Series is a one-dimensional array like structure with homogeneous data. For example, the following series is a collection of integers 10, 23, 56, …

Key Points

- Homogeneous data
- Size Immutable
- Values of Data Mutable

10	23	56	17	52	61	73	90	26	72

DataFrame

DataFrame is a two-dimensional array with heterogeneous data. For example,

Marvellous Infosystems Training Data set

Weight	Pattern	Label
35	Rough	Tennis
47	Rough	Tennis
90	Smooth	Cricket
48	Rough	Tennis
90	Smooth	Cricket
35	Rough	Tennis
92	Smooth	Cricket
35	Rough	Tennis
35	Rough	Tennis
35	Rough	Tennis
96	Smooth	Cricket
43	Rough	Tennis
110	Smooth	Cricket
35	Rough	Tennis
95	Smooth	Cricket

The table represents the data of a balls.
We are going to user to use this data set in machine learning application.
The data is represented in rows and columns. Each column represents an attribute and each row represents a person.
We consider data frame as our excel sheet.

Key Points

- Heterogeneous data
- Size Mutable
- Data Mutable

Panel

Panel is a three-dimensional data structure with heterogeneous data. It is hard to represent the panel in graphical representation. But a panel can be illustrated as a container of DataFrame.
We consider panel as excel file which contains multiple excel sheets (Data frame).

Key Points

- Heterogeneous data
- Size Mutable
- Data Mutable

Machine Learning Libraries in Python

1. NumPy (Numerical Python)

NumPy is the foundation for numerical computing in Python. It allows you to work with **arrays**, **matrices**, and perform **high-performance mathematical operations**.

Key Features:

- Multidimensional array support (ndarray)
- Fast mathematical functions (mean, median, std, etc.)
- Broadcasting and vectorized operations
- Linear algebra, Fourier transform, random number generation

All data in ML is ultimately numbers. NumPy helps perform fast mathematical operations during **data preprocessing**, **feature engineering**, and **model calculations**.

Example:

```
import numpy as np

arr = np.array([1, 2, 3, 4])
print("Mean:", np.mean(arr))
print("Standard Deviation:", np.std(arr))
```

2. Pandas (Python Data Analysis Library)

Pandas makes it easy to **load, analyze, clean, and manipulate data** in Python. It provides powerful **DataFrame** and **Series** structures similar to Excel tables.

Key Features:

- Load data from CSV, Excel, SQL, JSON, etc.
- Handle missing values, filter rows, group data
- Merge, join, and reshape datasets
- Excellent integration with NumPy and Matplotlib

Pandas is used for **data exploration**, **cleaning**, and **preprocessing**, which is the most time-consuming and critical part of the ML pipeline.

Example:

```
import pandas as pd

df = pd.read_csv("iris.csv")
print(df.head())  # Shows first 5 rows
print(df.describe())  # Statistical summary
```

3. Matplotlib

Matplotlib is the most widely used library for **basic data visualization** in Python.

Key Features:

- Create line plots, bar charts, histograms, scatter plots
- Full control over plot style, color, labels, and axes
- Supports exporting images in various formats (PNG, SVG)

Used to **visualize data** distributions, **loss curves**, and **performance metrics** during training.

Example:

```
import matplotlib.pyplot as plt

x = [1, 2, 3]
y = [10, 20, 30]

plt.plot(x, y)
plt.title("Sample Line Plot")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

4. Seaborn

Seaborn is a statistical data visualization library built on top of Matplotlib, offering **beautiful and informative plots**.

Key Features:

- Visualize distributions, correlations, and categorical data
- Built-in themes and color palettes
- Works seamlessly with Pandas DataFrames

Great for **exploratory data analysis (EDA)**—understanding the relationship between features and labels.

Example:

```
import seaborn as sns
import pandas as pd

df = sns.load_dataset('iris')
sns.pairplot(df, hue='species')
plt.show()
```

5. Scikit-learn (sklearn)

Scikit-learn is the **standard machine learning library** in Python. It provides tools for building, training, and evaluating ML models.

Key Features:

- Classification, regression, clustering algorithms
- Data preprocessing (scaling, encoding, splitting)
- Model evaluation tools (accuracy, confusion matrix)
- Pipeline for combining steps

It's used for everything from **model building** to **evaluation** in classical machine learning projects.

Example:

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# Train model
model = RandomForestClassifier()
model.fit(X_train, y_train)
print("Accuracy:", model.score(X_test, y_test))
```

6. TensorFlow

TensorFlow is a powerful open-source framework developed by **Google** for **building and training deep learning models**.

Key Features:

- Low-level control of neural networks
- Automatic differentiation and GPU acceleration
- Works for both research and production-level applications
- Supports mobile and embedded deployment (TensorFlow Lite)

Used for **deep learning**, especially where large data and powerful neural network architectures are required.

Example:

```
import tensorflow as tf

a = tf.constant([1, 2, 3])
print("Tensor:", a)
```

7. Keras (High-Level API for TensorFlow)

Keras is a **high-level neural network library** that simplifies the creation and training of deep learning models. It's now integrated within TensorFlow as `tf.keras`.

Key Features:

- Easy model building using Sequential or Functional API
- Abstracts complex TensorFlow code
- Supports custom layers and loss functions
- Fast prototyping of deep learning models

Used to **quickly build and train** deep learning models with very few lines of code.

Example:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Simple neural network
model = Sequential([
    Dense(8, activation='relu', input_shape=(4,)),
    Dense(3, activation='softmax')
])
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.summary()
```

Important Concepts of Machine Learning

Dataset :

A **dataset** is a structured collection of data used for analysis and training machine learning models.
In ML, each **row** typically represents an example (sample or instance), and each **column** represents a **feature (input)** or **label (output)**.

Example (Iris Dataset Sample)

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	Setosa

- **Features:** Sepal Length, Sepal Width, Petal Length, Petal Width
- **Label:** Species

Ways to Load Datasets in Python :

Method 1: Load Built-in Datasets (from sklearn)

```
from sklearn.datasets import load_iris
data = load_iris()
print(data.data)
```

Method 2: Load from a CSV file

```
import pandas as pd
df = pd.read_csv('iris.csv')
print(df.head())
```

Method 3: Load from a URL

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/
iris/iris.data"
df = pd.read_csv(url)
print(df.head())
```

Features, Labels, Independent & Dependent Variables

Features (Input Variables)

- These are the inputs to the model.
- Also called **independent variables**.
- Example: Age, Height, Weight, Income

Labels (Target Variable)

- This is the output we are trying to predict.
- Also called **dependent variable**.
- Example: Disease (Yes/No), House Price, Species

Train-Test Splitting

We split our dataset into **training** and **testing** parts to evaluate the model's performance.

- **Training Set:** Used to train the model
- **Testing Set:** Used to evaluate performance on unseen data

Typical Split:

- 80% training and 20% testing

Code Example:

```
from sklearn.model_selection import train_test_split

X = df.drop('species', axis=1)  # Features
y = df['species']               # Labels

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
```

Confusion Matrix & Related Terms

Used in classification problems to evaluate the performance of the model.

	Predicted: Positive	**Predicted: Negative**
Actual: Positive	True Positive (TP)	False Negative (FN)
Actual: Negative	False Positive (FP)	True Negative (TN)

Terms:

- **True Positive (TP):** Model correctly predicted Positive
- **True Negative (TN):** Model correctly predicted Negative
- **False Positive (FP):** Model incorrectly predicted Positive
- **False Negative (FN):** Model incorrectly predicted Negative

Accuracy and Other Evaluation Metrics

🔹 Accuracy:

Measures overall correctness of the model.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

🔹 Precision:

How many of the predicted positives are actually positive?

$$\text{Precision} = \frac{TP}{TP + FP}$$

🔹 Recall (Sensitivity):

How many actual positives were correctly predicted?

$$\text{Recall} = \frac{TP}{TP + FN}$$

🔹 F1 Score:

Harmonic mean of precision and recall

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Code Example: Confusion Matrix & Accuracy

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# 1. Load the Iris dataset
iris = load_iris()
X = iris.data        # Feature data (sepal & petal measurements)
y = iris.target      # Labels (species: 0, 1, 2)

# 2. Split the dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
# 3. Create and train the Decision Tree model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# 4. Predict using the test set
y_pred = model.predict(X_test)

# 5. Evaluate performance
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

Train-Test Split in Iris Dataset

When using machine learning, it's important to divide the data into two parts:

- **Training data**: used to train the model.
- **Testing data**: used to test how well the model performs on unseen data.

Variables Involved:

- **X** = Features (Input)
- **y** = Labels (Output / Target)
- **X_train** = Features for training the model
- **X_test** = Features for testing the model
- **y_train** = Labels used during training
- **y_test** = Actual labels used to test predictions

Iris Dataset Example with Explanation:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load dataset
iris = load_iris()
X = iris.data        # All feature columns (sepal/petal length/width)

y = iris.target      # Labels (0 = Setosa, 1 = Versicolor, 2 =
Virginica)

# Split into train and test sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# Print the shapes to verify
print("X_train shape:", X_train.shape)  # e.g., (120, 4)
```

```
print("X_test shape:", X_test.shape)    # e.g., (30, 4)
print("y_train shape:", y_train.shape)  # e.g., (120,)
print("y_test shape:", y_test.shape)    # e.g., (30,)
```

Variable	Meaning
`X_train`	80% of feature data used for training
`X_test`	20% of feature data used for testing
`y_train`	Labels corresponding to `X_train`
`y_test`	Labels corresponding to `X_test`, used for evaluation

Supervised Machine Learning

Supervised Machine Learning :

Supervised Machine Learning is a type of machine learning where the model is trained on a **labeled dataset** — which means each input (X) is associated with a correct output (Y). The algorithm learns the relationship between inputs and outputs and uses it to **predict outcomes for new, unseen data**.

Example:

If you're training a model to classify emails as *Spam* or *Not Spam*, you feed it emails with the correct labels. The model learns the pattern and then can predict future emails.

Types of Supervised Learning

Supervised learning is broadly categorized into:

1. **Classification**
2. **Regression**

1. Classification

In classification, the output is **categorical** (discrete labels or classes).

✅ **Examples:**

- Email Spam Detection (Spam / Not Spam)
- Disease Diagnosis (Positive / Negative)
- Handwriting Recognition (Digits 0–9)

Popular Algorithms:

Algorithm	Description
Logistic Regression	Used for binary and multi-class classification. Despite its name, it's a classification method.
K-Nearest Neighbors (KNN)	Classifies a data point based on the majority class among its 'K' nearest neighbors.
Decision Tree	A tree-like model of decisions, used for classification and regression.
Random Forest	An ensemble of decision trees, improves accuracy and reduces overfitting.
Support Vector Machine (SVM)	Finds the best boundary (hyperplane) that separates classes.

2. Regression

In regression, the output is **continuous** (numeric values).

✅ **Examples:**

- Predicting house prices
- Forecasting stock market trends
- Estimating car mileage

Popular Algorithms:

Algorithm	Description
Linear Regression	Models the relationship between dependent and independent variables with a straight line.
Polynomial Regression	Fits a polynomial curve instead of a straight line for non-linear relationships.

Supervised Learning Flow

```
1. Collect labeled data
2. Split into training and testing sets
3. Choose appropriate algorithm
4. Train the model on training data
5. Evaluate the model on test data (using metrics like accuracy, RMSE, etc.)
6. Use model for prediction on new data
```

Evaluation Metrics

Classification:

- Accuracy
- Precision
- Recall
- F1 Score
- Confusion Matrix

Regression:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)

- Root Mean Squared Error (RMSE)
- R^2 Score (Coefficient of Determination)

Supervised Machine Learning Algorithms

In machine learning, classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation.
This data set may simply be bi-class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class too.
Some examples of classification problems are: speech recognition, handwriting recognition, bio metric identification, document classification etc.

Here we have the types of classification algorithms in Machine Learning:

- Logistic Regression
- Naive Bayes Classifier
- Stochastic Gradient
- K Nearest Neighbour
- Decision Trees
- Random Forest
- Support Vector Machines

Structured Data Classification

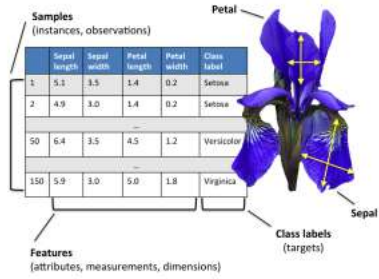
Classification can be performed on structured or unstructured data.
Classification is a technique where we categorise data into a given number of classes. The main goal of a classification problem is to identify the category/class to which a new data will fall under.

Few of the terminologies encountered in machine learning – classification:

Classifier:
An algorithm that maps the input data to a specific category.

Classification model:
A classification model tries to draw some conclusion from the input values given for training.
It will predict the class labels/categories for the new data.

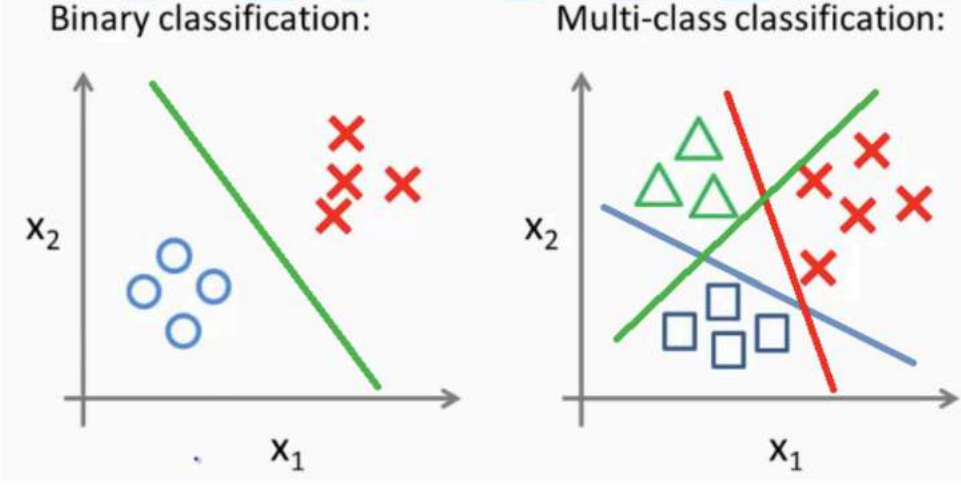
Feature:
A feature is an individual measurable property of a phenomenon being observed.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

Binary Classification:
Classification task with two possible outcomes. Eg: Gender classification (Male / Female)

Multi class classification:
Classification with more than two classes. In multi class classification each sample is assigned to one and only one target label. Eg: An animal can be cat or dog but not both at the same time



Multi label classification:
Classification task where each sample is mapped to a set of target labels (more than one class). Eg: A news article can be about sports, a person, and location at the same time.

The following are the steps involved in building a classification model

Step 1:
Initialise the classifier to be used.

Step 2:
Train the classifier:
All classifiers in scikit-learn uses a **fit**(X, y) method to fit the model(training) for the given train data X and train label y.

Step 3:
Predict the target:
Given an unlabeled observation X, the **predict**(X) returns the predicted label y.

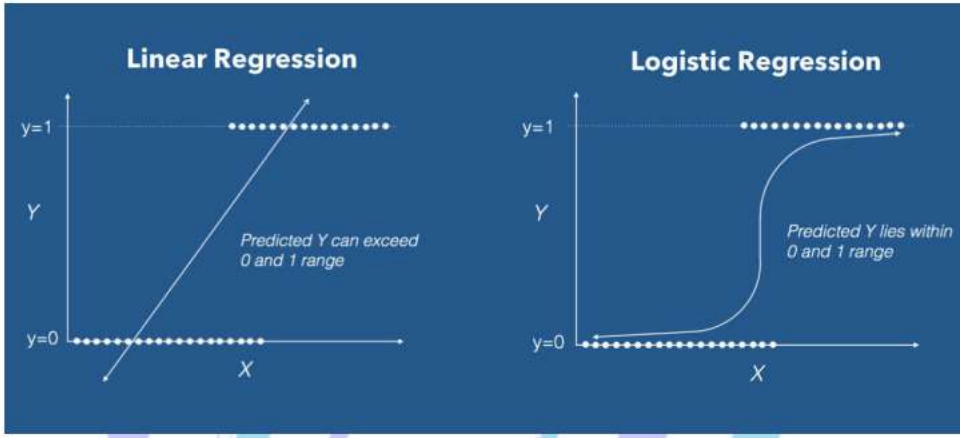
Step 4:
Evaluate the classifier model

Example :

```
from sklearn import tree

# Data Set
BalllsFeatures = [[35,1],[47,1],[90,0],[48,1],[90,0],[35,1],[92,0],[35,1],[35,1],[35,1],
[96,0],[43,1],[110,0],[35,1],[95,0]]

# Features
Names = [1,1,2,1,2,1,2,1,1,1,2,1,2,1,2]

# Step 1 Initialise the classifier
clf = tree.DecisionTreeClassifier()

# Step 2 Train the classifier
clf = clf.fit(BalllsFeatures,Names)

# Step 3 Predict the target
print(clf.predict([[44,1]]))
```

Classification Algorithms

Logistic Regression Algorithm (Predictive Learning Model)



Definition:

Logistic regression is a machine learning algorithm for classification. In this algorithm, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.
It is a statistical method for analysing a data set in which there are one or more independent variables that determine an outcome.
The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).
The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables.

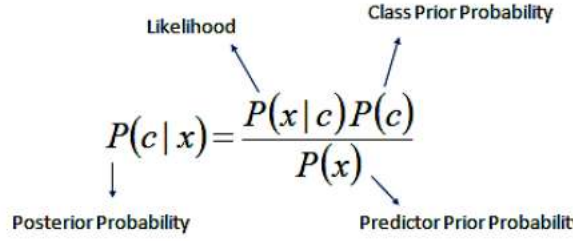
Advantages:

Logistic regression is designed for this purpose (classification), and is most useful for understanding the influence of several independent variables on a single outcome variable.

Disadvantages:

Works only when the predicted variable is binary, assumes all predictors are independent of each other, and assumes data is free of missing values.

Naïve Bayes Algorithm
(Generative Learning Model)



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Definition:

Naive Bayes algorithm based on Bayes' theorem with the assumption of independence between every pair of features.
Naive Bayes classifiers work well in many real-world situations such as document classification and spam filtering.
It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability.
Naive Bayes model is easy to build and particularly useful for very large data sets.
Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

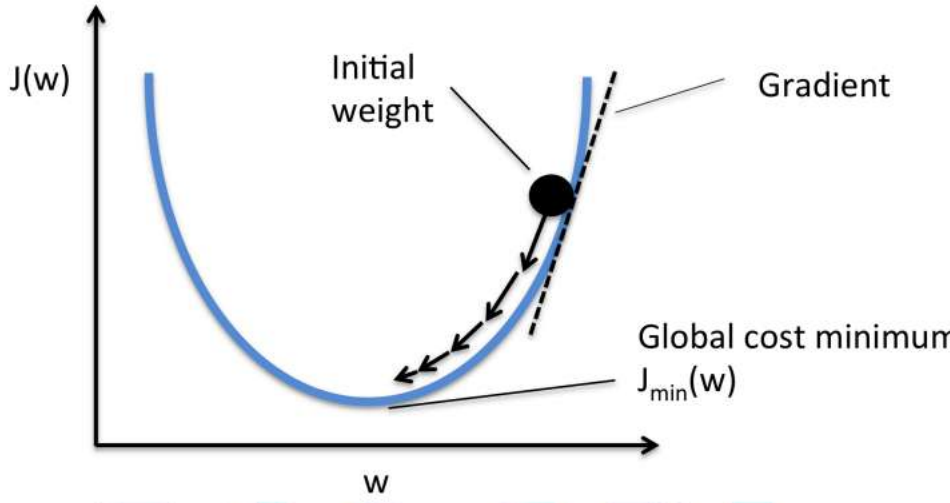
Advantages:

This algorithm requires a small amount of training data to estimate the necessary parameters. Naive Bayes classifiers are extremely fast compared to more sophisticated methods.

Disadvantages:

Naive Bayes is is known to be a bad estimator.

Stochastic Gradient Descent Algorithm



Definition:

Stochastic gradient descent is a simple and very efficient approach to fit linear models.
It is particularly useful when the number of samples is very large.
It supports different loss functions and penalties for classification.

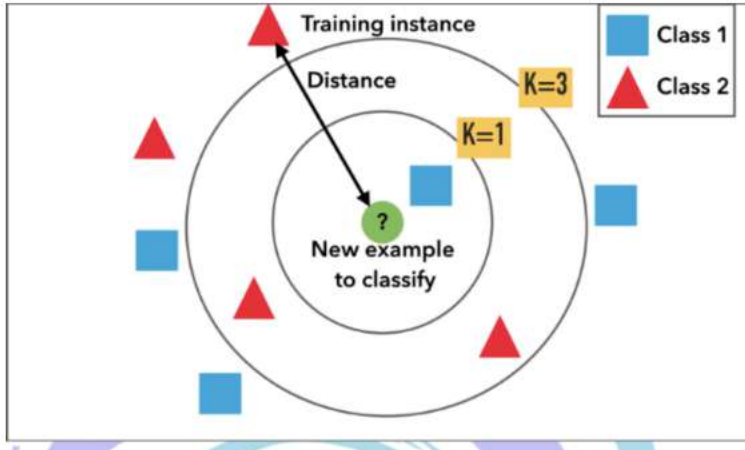
Advantages:

Efficiency and ease of implementation.

Disadvantages:

Requires a number of hyper-parameters and it is sensitive to feature scaling.

K-Nearest Neighbours Algorithm



Definition:

Neighbours based classification is a type of lazy learning as it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the k nearest neighbours of each point.
The k-nearest-neighbors algorithm is a classification algorithm, and it is supervised: it takes a bunch of labelled points and uses them to learn how to label other points.
To label a new point, it looks at the labelled points closest to that new point (those are its nearest neighbors), and has those neighbors vote, so whichever label the most of the neighbors have is the label for the new point (the "k" is the number of neighbors it checks).

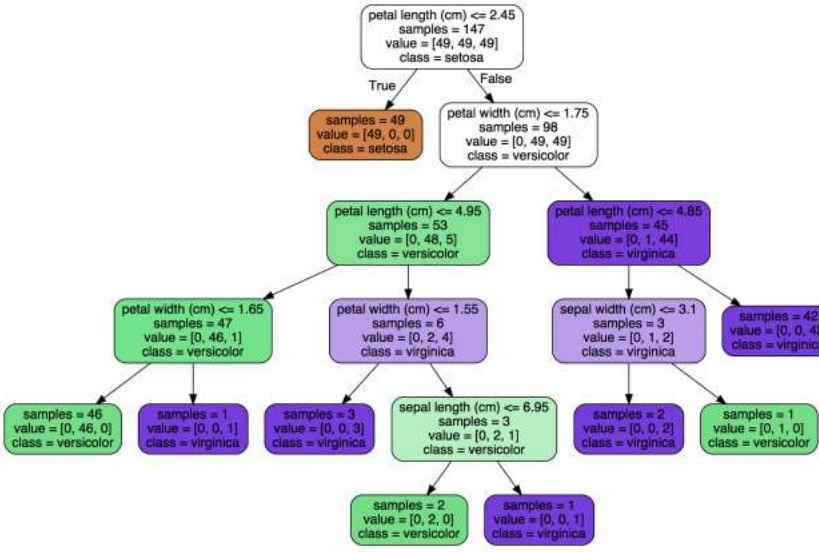
Advantages:

This algorithm is simple to implement, robust to noisy training data, and effective if training data is large.

Disadvantages:

Need to determine the value of K and the computation cost is high as it needs to computer the distance of each instance to all the training samples.

Decision Tree Algorithm



Definition:

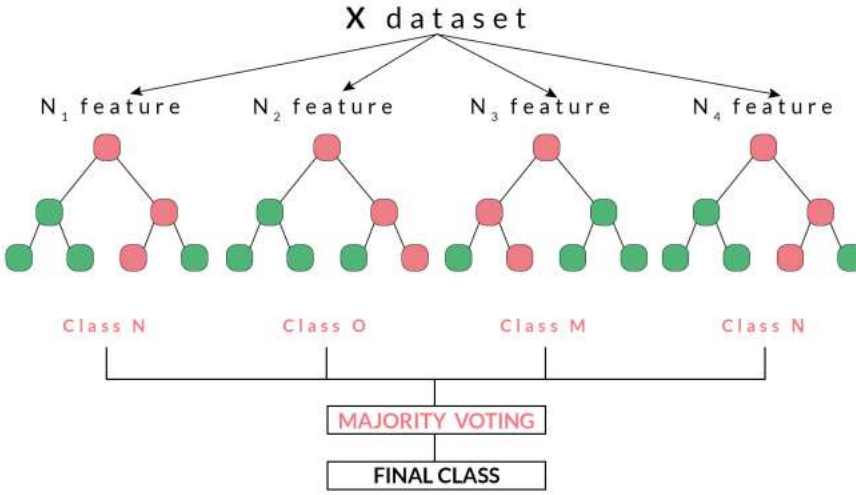
Given a data of attributes together with its classes, a decision tree produces a sequence of rules that can be used to classify the data.
Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed.
The final result is a tree with decision nodes and leaf nodes.
A decision node has two or more branches and a leaf node represents a classification or decision.
The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Advantages:

Decision Tree is simple to understand and visualise, requires little data preparation, and can handle both numerical and categorical data.

Disadvantages:
Decision tree can create complex trees that do not generalise well, and decision trees can be unstable because small variations in the data might result in a completely different tree being generated.

Random Forest Algorithm



Definition:

Random forest classifier is a meta-estimator that fits a number of decision trees on various sub-samples of datasets and uses average to improve the predictive accuracy of the model and controls over-fitting.
The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement.
Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.
Random decision forests correct for decision trees' habit of over fitting to their training set.

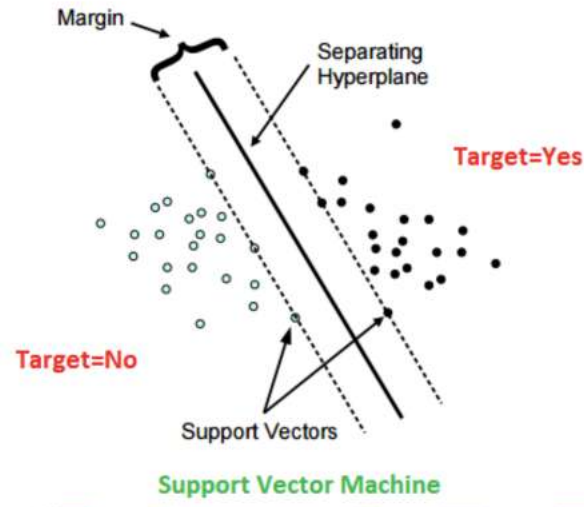
Advantages:

Reduction in over-fitting and random forest classifier is more accurate than decision trees in most cases.

Disadvantages:

Slow real time prediction, difficult to implement, and complex algorithm.

Support Vector Machine Algorithm



Definition:

Support vector machine is a representation of the training data as points in space separated into categories by a clear gap that is as wide as possible.
New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Advantages:

Effective in high dimensional spaces and uses a subset of training points in the decision function so it is also memory efficient.

Disadvantages:

The algorithm does not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

We can compare each algorithm by considering its Accuracy and F1-Score

Accuracy:

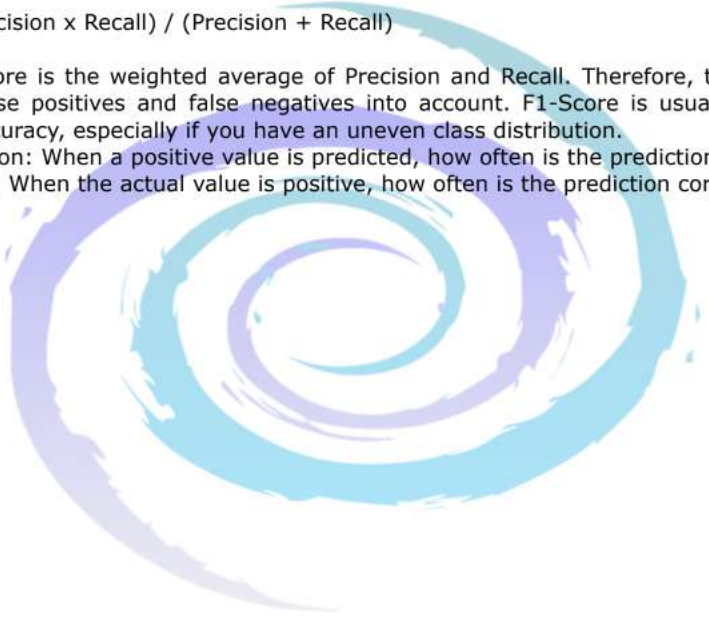
(True Positive + True Negative) / Total Population

• Accuracy is a ratio of correctly predicted observation to the total observations. Accuracy is the most intuitive performance measure.
• True Positive: The number of correct predictions that the occurrence is positive
• True Negative: The number of correct predictions that the occurrence is negative

F1-Score:

(2 x Precision x Recall) / (Precision + Recall)

• F1-Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. F1-Score is usually more useful than accuracy, especially if you have an uneven class distribution.
• Precision: When a positive value is predicted, how often is the prediction correct?
• Recall: When the actual value is positive, how often is the prediction correct?



Data Visualization using Seaborn & Matplotlib

In the world of **data science and machine learning**, understanding the structure, distribution, and relationships within data is extremely important.

Raw data in tables or spreadsheets can often be overwhelming and difficult to interpret.

This is where **data visualization** comes in — it allows us to **see patterns**, **spot outliers**, and **understand trends** clearly and effectively.

Two of the most powerful Python libraries used for creating such visualizations are:

- **Matplotlib** – the foundation library for static, animated, and interactive plots in Python.
- **Seaborn** – built on top of Matplotlib, it provides a high-level interface for creating attractive and informative statistical graphics.

1. Histogram using Matplotlib

File: `Histogram.py`

```
import pandas as pd
import matplotlib.pyplot as plt

def main():
    df = pd.read_csv("iris.csv")

    plt.hist(df['sepal.length'], bins=10, color="skyblue",
edgecolor='black')

    plt.xlabel("Sepal Length")
    plt.ylabel("Frequency")
    plt.title("Marvellous Histogram for IRIS")

    plt.show()

if __name__ == "__main__":
    main()
```

✅ Explanation:

- **Histogram** is used to visualize the **distribution** of a numerical feature.
- `plt.hist()` plots a histogram.
- `bins=10`: Divides data into 10 ranges.
- `color` and `edgecolor`: Used to enhance visibility.
- This chart shows how frequently each sepal length appears in the dataset.

2. Boxplot using Seaborn

File: `boxplot.py`

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def main():
    df = pd.read_csv("iris.csv")

    sns.boxplot(x="variety", y="petal.length", data=df)

    plt.title("Marvellous Boxplot for Petal length by variety")
    plt.show()

if __name__ == "__main__":
    main()
```

✅ **Explanation:**

- **Boxplot** is used to show the **spread and outliers** in the data.
- `x="variety"` groups the boxplots by flower variety.
- `y="petal.length"` shows the distribution of petal lengths.
- Very useful for **comparing distributions across categories**.

3. Pairplot using Seaborn

File: `pairplot.py`

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def main():
    df = pd.read_csv("iris.csv")

    sns.pairplot(df, hue="variety")

    plt.show()

if __name__ == "__main__":
    main()
```

✅ **Explanation:**

- **Pairplot** plots all possible **pairwise scatter plots** of features.
- `hue="variety"` colors the points based on flower type.
- Great for **understanding feature relationships** and **visualizing class separation**.
- Very helpful in **Exploratory Data Analysis (EDA)**.

4. 3D Scatter Plot using Matplotlib

File: `subplot.py`

```
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

def main():
    iris = load_iris()
    x = iris.data
    y = iris.target

    fig = plt.figure(figsize=(8,6))

    ax = fig.add_subplot(111, projection='3d')

    ax.scatter(x[:,2], x[:,3], x[:,0], c=y, cmap="viridis",
edgecolor='k')

    ax.set_xlabel("Petal length")
    ax.set_ylabel("Petal width")
    ax.set_zlabel("Sepal length")

    plt.title("Marvellous 3D visualisation for IRIS")

    plt.show()

if __name__ == "__main__":
    main()
```

✅ **Explanation:**

- This is a **3D scatter plot** using `mpl_toolkits.mplot3d`.
- `projection='3d'` enables 3D plotting.
- `ax.scatter()` plots the points with different colors for each class (`c=y`).
- `x[:,2], x[:,3], x[:,0]`: These are features chosen as axes.
- Useful for visualizing **multi-dimensional relationships**.

Confusion Matrix

N = 165	Predicted NO	Predicted YES	
Actual NO	TN = 50	FP = 10	60
Actual YES	FN = 5	TP = 100	105
	55	110	

- true positives (TP): These are cases in which we predicted yes (they have the disease), and they do have the disease.
- true negatives (TN): We predicted no, and they don't have the disease.
- false positives (FP): We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- false negatives (FN): We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

- Accuracy: How often is the classifier correct?
  - (TP+TN)/total = (100+50)/165 = 0.91
- Misclassification Rate: How often is it wrong?
  - (FP+FN)/total = (10+5)/165 = 0.09
  - equivalent to 1 minus Accuracy
  - also known as "Error Rate"
- True Positive Rate: When it's actually yes, how often does it predict yes? (Recall)
  - TP/actual yes = 100/105 = 0.95
  - also known as "Sensitivity" or "Recall"
- False Positive Rate: When it's actually no, how often does it predict yes?
  - FP/actual no = 10/60 = 0.17
- True Negative Rate: When it's actually no, how often does it predict no?
  - TN/actual no = 50/60 = 0.83
  - equivalent to 1 minus False Positive Rate
  - also known as "Specificity"
- Precision: When it predicts yes, how often is it correct?
  - TP/predicted yes = 100/110 = 0.91
- Prevalence: How often does the yes condition actually occur in our sample?
  - actual yes/total = 105/165 = 0.64

F1 Score

The F-score, also called the F1-score, is a measure of a model's accuracy on a dataset. It is used to evaluate binary classification systems, which classify examples into 'positive' or 'negative'.

F1 is calculated as follows:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

where:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

In "macro" F1 a separate F1 score is calculated for each `species` value and then averaged.

Linear Regression

X (Independent)	Y (Dependent)	X - X_Bar	Y - Y_Bar	(X-X_Bar)^2	(X - X_Bar) * (Y - Y_Bar)	Yp	(Yp - Y_Bar)	(Yp - Y_Bar) ^ 2	(Y - Y_Bar)^2
1	3	-2	-0.6	4	1.2	2.8	-0.8	0.64	0.36
2	4	-1	0.4	1	-0.4	3.2	-0.4	0.16	0.16
3	2	0	-1.6	0	0	3.6	0	0	2.56
4	4	1	0.4	1	0.4	4.0	0.4	0.16	0.16
5	5	2	1.4	4	2.8	4.4	0.8	0.64	1.96
X_Bar 3	**Y_Bar. 3.6**			**Sum = 10**	**Sum = 4.0**			**Sum = 1.6**	**Sum = 5.2**

Equation of line : Y = mX + C

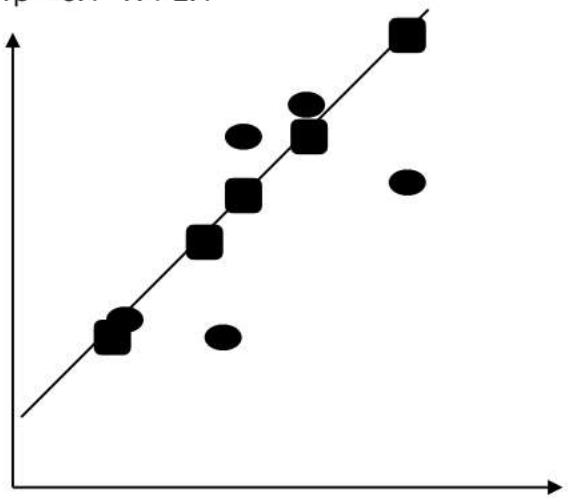
Y Dependent Variable
X Independent Variable
m Slope of line
c Y intercept of line

$$m = \frac{\Sigma (X - X\_Bar)(Y - Y\_Bar)}{\Sigma (X - X\_Bar)^2}$$

m = 4/10
m = 0.4

Y = mX + C
3.6 = 0.4 * 3 + C
3.6 = 1.2 + C
C = 3.6 - 1.2
C = 2.4

Yp = 0.4 * X + 2.4



R Square method

Distance (predicted - mean)
VS
Distance (actual - mean)

R_Square formula

$$\frac{\Sigma (Yp - Y\_Bar)^2}{\Sigma (Y - Y\_Bar)^2}$$

R^2 = 1.6 / 5.2
R^2 = 0.3

K-Nearest Neighbors (KNN)

KNN (K-Nearest Neighbors) is a simple, supervised machine learning algorithm used for both classification and regression tasks. However, it is more widely used for classification problems.

KNN works by:

- **Storing all available data** (no model training),
- And classifying a new data point based on **majority class of its K nearest neighbors** (for classification),
- Or **average of K nearest neighbor values** (for regression).

Step-by-Step Working of KNN:

1. **Choose the number K** (the number of neighbors to consider).
2. **Calculate the distance** between the new data point and all existing data points using a distance metric (typically Euclidean distance).
3. **Sort the distances** in ascending order and identify the K closest neighbors.
4. **Classify or Predict:**
    - For **classification**: Use **majority vote** among K nearest points.
    - For **regression**: Take the **mean (average)** of values of the K nearest points.

Mathematical Formula

🔷 Euclidean Distance:

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

For n-dimensions:

$$\text{Distance} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Problem Statement:

We want to classify a new point `P(3,3)` using **K = 3** based on the following training data:

Point	Coordinates (x, y)	Class
A	(1, 2)	Red
B	(2, 3)	Red
C	(3, 1)	Blue
D	(6, 5)	Blue

Step 1: Calculate Euclidean Distance

The **Euclidean distance formula** is:

Calculate distances between point `P(3,3)` and each training point:

- A(1,2): $\sqrt{(3-1)^2 + (3-2)^2} = \sqrt{4 + 1} = \sqrt{5} \approx$ **2.24**
- B(2,3): $\sqrt{(3-2)^2 + (3-3)^2} = \sqrt{1 + 0} = \sqrt{1} =$ **1.00**
- C(3,1): $\sqrt{(3-3)^2 + (3-1)^2} = \sqrt{0 + 4} = \sqrt{4} =$ **2.00**
- D(6,5): $\sqrt{(3-6)^2 + (3-5)^2} = \sqrt{9 + 4} = \sqrt{13} \approx$ **3.61**

Step 2: Sort and Select K Nearest Neighbors

Sort the distances in ascending order and pick **K = 3** nearest:

1. B (1.00) → Red
2. C (2.00) → Blue
3. A (2.24) → Red

Step 3: Majority Voting

From top 3 neighbors:

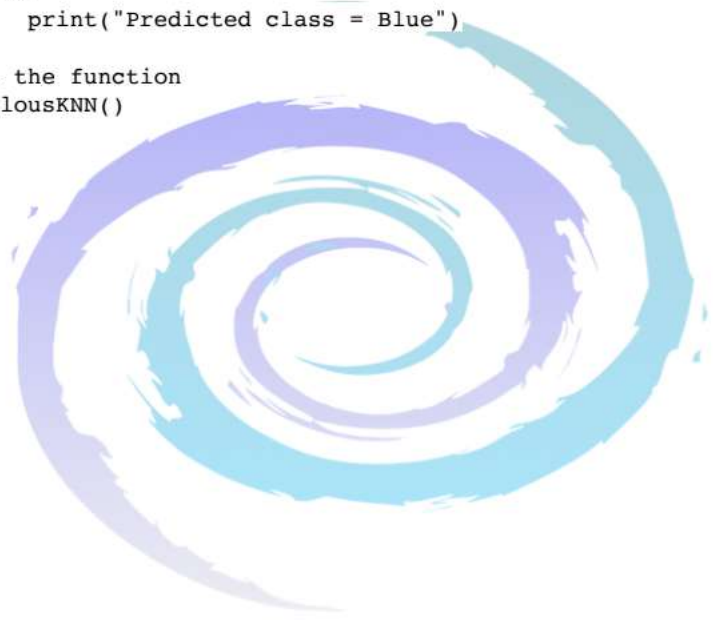
- **Red**: 2 votes (B, A)
- **Blue**: 1 vote (C)

Final **Predicted Class** for point **P(3,3)**: **Red**

Code Explanation

```
import numpy as np
import math

# Step 1: Euclidean distance function
def EucDistance(P1, P2):
    return math.sqrt((P1['x'] - P2['x'])**2 + (P1['y'] - P2['y'])**2)

# KNN Classification function
def MarvellousKNN():
    line = "-"*50

    # Step 0: Training data
    data = [
        {'point': 'A', 'x': 1, 'y': 2, 'label': 'Red'},
        {'point': 'B', 'x': 2, 'y': 3, 'label': 'Red'},
        {'point': 'C', 'x': 3, 'y': 1, 'label': 'Blue'},
        {'point': 'D', 'x': 6, 'y': 5, 'label': 'Blue'}
    ]

    print(line)
    print("Training data set:")
    for i in data:
        print(i)
    print(line)

    # New point to classify
    new_point = {'x': 3, 'y': 3}

    # Step 1: Calculate distances
    for d in data:
        d['distance'] = EucDistance(d, new_point)

    print("Calculated Distances:")
    for d in data:
        print(d)
    print(line)

    # Step 2: Sort by distance
    sorted_data = sorted(data, key=lambda item: item['distance'])

    print("Sorted Data by Distance:")
    for d in sorted_data:
        print(d)
    print(line)
```

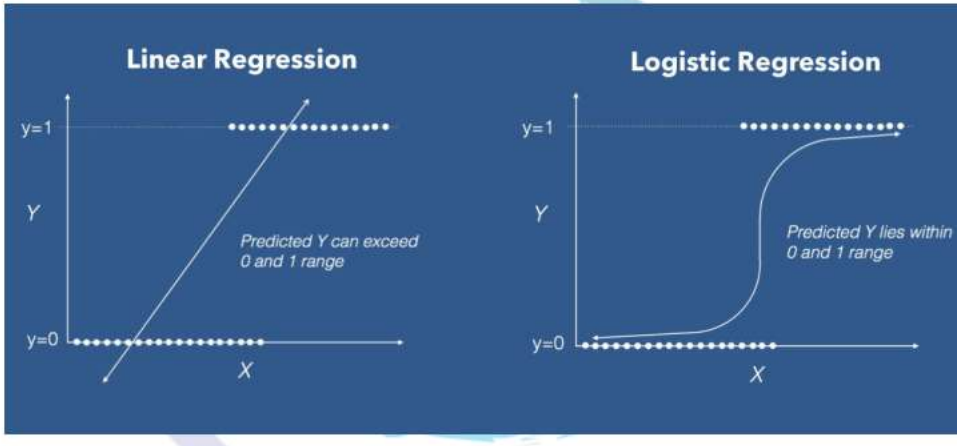
```
    # Step 3: Select top K = 3 neighbors
    K = 3
    k_nearest = sorted_data[:K]

    # Step 4: Majority voting
    red = sum(1 for i in k_nearest if i['label'] == 'Red')
    blue = sum(1 for i in k_nearest if i['label'] == 'Blue')

    print(f"Red votes = {red}, Blue votes = {blue}")
    if red > blue:
        print("Predicted class = Red")
    else:
        print("Predicted class = Blue")

# Call the function
MarvellousKNN()
```



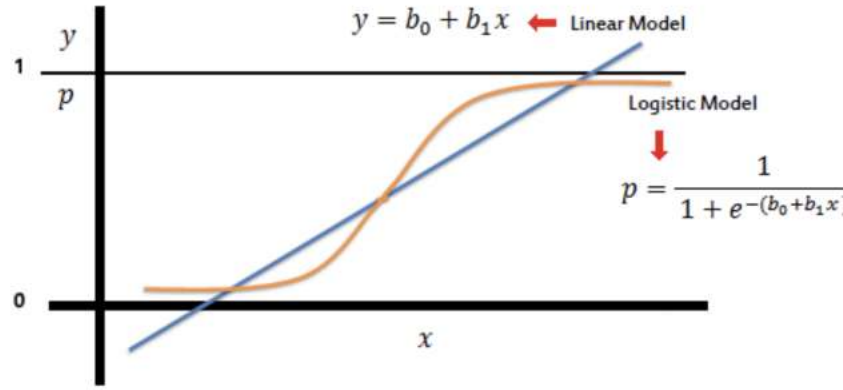
Logistic Regression

Logistic regression predicts the probability of an outcome that can only have two values (i.e. a dichotomy).
The prediction is based on the use of one or several predictors (numerical and categorical).
A linear regression is not appropriate for predicting the value of a binary variable for two reasons:

- A linear regression will predict values outside the acceptable range (e.g. predicting probabilities outside the range 0 to 1)
- Since the dichotomous experiments can only have one of two possible values for each experiment, the residuals will not be normally distributed about the predicted line.



On the other hand, a logistic regression produces a logistic curve, which is limited to values between 0 and 1.
Logistic regression is similar to a linear regression, but the curve is constructed using the natural logarithm of the "odds" of the target variable, rather than the probability. Moreover, the predictors do not have to be normally distributed or have equal variance in each group.



In the logistic regression the constant (b0) moves the curve left and right and the slope (b1) defines the steepness of the curve. By simple transformation, the logistic regression equation can be written in terms of an odds ratio.

$$\frac{p}{1-p} = \exp(b_0 + b_1 x)$$

Finally, taking the natural log of both sides, we can write the equation in terms of log-odds (logit) which is a linear function of the predictors. The coefficient (b1) is the amount the logit (log-odds) changes with a one unit change in x.

$$ln\left(\frac{p}{1-p}\right) = b_0 + b_1 x$$

As mentioned before, logistic regression can handle any number of numerical and/or categorical variables.

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p)}}$$

There are several analogies between linear regression and logistic regression. Just as ordinary least square regression is the method used to estimate coefficients for the best fit line in linear regression, logistic regression uses maximum likelihood estimation (MLE)

to obtain the model coefficients that relate predictors to the target. After this initial function is estimated, the process is repeated until LL (Log Likelihood) does not change significantly.

$$\boldsymbol{\beta^1 = \beta^0 + [X^T W X]^{-1}.X^T(y - \mu)}$$

$\boldsymbol{\beta}$ *is a vector of the logistic regression coefficients.*

$\boldsymbol{W}$ *is a square matrix of order N with elements* $n_i\pi_i(1-\pi_i)$ *on the diagonal and zeros everywhere else.*

$\boldsymbol{\mu}$ *is a vector of length N with elements* $\mu_i = n_i\pi_i$.

Logistic Regression – Concept + Mathematical Example

- Logistic Regression is a **supervised learning algorithm** used for **binary classification** problems.
- It predicts **probability** that a data point belongs to a particular class using a **sigmoid function**.
- Output is between **0 and 1** – typically converted to 0 or 1 using a **threshold** (commonly 0.5).

🔷 Sigmoid Function

The logistic function (sigmoid) is:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Where:

- $z = mx + c$ (linear equation)
- m is slope/weight
- c is intercept/bias

Example with Numerical Values

We have:

Age (X)	Result (Y)
50	1
30	0
20	0
40	1
60	1

Let's fit a logistic regression model on this data and use **gradient descent** to learn parameters `m` and `c`.

Step-by-Step Manual Calculation

Let's simplify with initial guess:

Assume m = 0.1 c=–4 (Just for calculation)

Predict probability using sigmoid:

$$z = mx + c = 0.1x - 4$$

Let's calculate $\hat{y}$ for each input (Age):

Age (x)	z = 0.1x - 4	$\hat{y} = \frac{1}{1+e^{-z}}$
50	1	$\frac{1}{1+e^{-1}} \approx 0.731$
30	-1	$\frac{1}{1+e^{1}} \approx 0.268$
20	-2	$\frac{1}{1+e^{2}} \approx 0.119$
40	0	$\frac{1}{1+e^{0}} = 0.5$
60	2	$\frac{1}{1+e^{-2}} \approx 0.881$

Prediction:

- **x = 15**:

$$z = 0.1 * 15 - 4 = -2.5 \Rightarrow \hat{y} \approx \frac{1}{1+e^{2.5}} \approx 0.075 \Rightarrow \text{Predicted Class} = 0$$

- **x = 45**:

$$z = 0.1 * 45 - 4 = 0.5 \Rightarrow \hat{y} \approx \frac{1}{1+e^{-0.5}} \approx 0.622 \Rightarrow \text{Predicted Class} = 1$$

Application 6

Supervised Machine Learning

User Defined K Nearest Neighbour Algorithm

- In this application we create our own algorithm for classified machine learning.
- We create our own K Nearest Neighbour algorithm.
- For user defined algorithm we design one class named as MarvellousKNN.
- This class contains 3 methods as fit, predict, closest method.
- There is one naked method euc() which calculate distance between two points using Euclidean distance algorithm.
- fit() method initialises training data and its targets inside class.
- predict() method creates one array as prediction which stores shortest distance between all test data and training data elements.
- predict() method calls closest method which returns the shortest distance.

Consider below characteristics of Machine Learning Application :

Classifier :	**User Defined K Nearest Neighbour**
DataSet :	**Iris Dataset**
Features :	**Sepal Width, Sepal Length, Petal Width, Petal Length**
Labels :	**Versicolor, Setosa , Virginica**
Training Dataset :	**75 Entries**
Testing Dataset :	**75 Entries**

```
1  from sklearn import tree
2  from scipy.spatial import distance
3  from sklearn.datasets import load_iris
4  from sklearn.metrics import accuracy_score
5  from sklearn.model_selection import train_test_split
6
7  def euc(a,b):
8      return distance.euclidean(a,b)
9
10 class MarvellousKNN():
11     def fit(self,TrainingData,TrainingTarget):
12         self.TrainingData = TrainingData
13         self.TrainingTarget = TrainingTarget
14
15     def predict(self,TestData):
16         predictions = []
17         for row in TestData:
18             lebel = self.closest(row)
19             predictions.append(lebel)
20         return predictions
21
```

```
    def closest(self,row):
        bestdistance = euc(row,self.TrainingData[0])
        bestindex = 0
        for i in range(1,len(self.TrainingData)):
            dist = euc(row,self.TrainingData[i])
            if dist < bestdistance:
                bestdistance = dist
                bestindex = i
        return self.TrainingTarget[bestindex]

def MarvellousKNeighbor():
    border = "-"*50

    iris = load_iris()

    data = iris.data
    target = iris.target

    print(border)
    print("Actual data set")
    print(border)

    for i in range(len(iris.target)):
        print("ID: %d, Label %s, Feature : %s" % (i,iris.data[i],iris.target[i]))
    print("Size of Actual data set %d"%(i+1))

    data_train, data_test, target_train, target_test = train_test_split(data,target,test_size=0.5)


    print(border)
    print("Training data set")
    print(border)
    for i in range(len(data_train)):
        print("ID: %d, Label %s, Feature : %s" % (i,data_train[i],target_train[i]))
    print("Size of Training data set %d"%(i+1))

    print(border)
    print("Test data set")
    print(border)
    for i in range(len(data_test)):
        print("ID: %d, Label %s, Feature : %s" % (i,data_test[i],target_test[i]))
    print("Size of Test data set %d"%(i+1))
    print(border)

    classifier = MarvellousKNN()

    classifier.fit(data_train,target_train)

    predictions = classifier.predict(data_test)

    Accuracy = accuracy_score(target_test,predictions)

    return Accuracy

def main():

    Accuracy = MarvellousKNeighbor()
    print("Accuracy of classification algorithm with K Neighbor classifier is",Accuracy*100,"%")

if __name__ == "__main__":
    main()
```

Output of above Application :

```
(base) MacBook-Pro-de-MARVELLOUS:iris marvellous$ p
ython3 MarvellousClassifier.py
-------------------------------------------------
Actual data set
-------------------------------------------------
ID: 0, Label [5.1 3.5 1.4 0.2], Feature : 0
ID: 1, Label [4.9 3.  1.4 0.2], Feature : 0
ID: 2, Label [4.7 3.2 1.3 0.2], Feature : 0
ID: 3, Label [4.6 3.1 1.5 0.2], Feature : 0
ID: 4, Label [5.  3.6 1.4 0.2], Feature : 0
ID: 5, Label [5.4 3.9 1.7 0.4], Feature : 0
ID: 6, Label [4.6 3.4 1.4 0.3], Feature : 0
ID: 7, Label [5.  3.4 1.5 0.2], Feature : 0
ID: 8, Label [4.4 2.9 1.4 0.2], Feature : 0
ID: 9, Label [4.9 3.1 1.5 0.1], Feature : 0
ID: 10, Label [5.4 3.7 1.5 0.2], Feature : 0
ID: 11, Label [4.8 3.4 1.6 0.2], Feature : 0
ID: 12, Label [4.8 3.  1.4 0.1], Feature : 0
ID: 13, Label [4.3 3.  1.1 0.1], Feature : 0
ID: 14, Label [5.8 4.  1.2 0.2], Feature : 0
ID: 146, Label [6.3 2.5 5.  1.9], Feature : 2
ID: 147, Label [6.5 3.  5.2 2. ], Feature : 2
ID: 148, Label [6.2 3.4 5.4 2.3], Feature : 2
ID: 149, Label [5.9 3.  5.1 1.8], Feature : 2
Size of Actual data set 150
-------------------------------------------------
Training data set
-------------------------------------------------
ID: 0, Label [5.4 3.  4.5 1.5], Feature : 1
ID: 1, Label [5.4 3.7 1.5 0.2], Feature : 0
ID: 2, Label [6.  2.7 5.1 1.6], Feature : 1
ID: 3, Label [5.  3.5 1.6 0.6], Feature : 0
ID: 4, Label [6.9 3.1 4.9 1.5], Feature : 1
ID: 5, Label [5.  2.  3.5 1. ], Feature : 1
ID: 6, Label [5.5 4.2 1.4 0.2], Feature : 0
ID: 7, Label [5.5 3.5 1.3 0.2], Feature : 0
ID: 8, Label [4.6 3.6 1.  0.2], Feature : 0
ID: 9, Label [4.9 3.1 1.5 0.2], Feature : 0
ID: 10, Label [6.4 2.9 4.3 1.3], Feature : 1
```

```
ID: 72, Label [5.   3.6 1.4 0.2], Feature : 0
ID: 73, Label [5.7 2.9 4.2 1.3], Feature : 1
ID: 74, Label [5.   3.4 1.6 0.4], Feature : 0
Size of Training data set 75
---------------------------------------------------
Test data set
---------------------------------------------------
ID: 0, Label [6.1 2.6 5.6 1.4], Feature : 2
ID: 1, Label [6.2 3.4 5.4 2.3], Feature : 2
ID: 2, Label [4.8 3.4 1.9 0.2], Feature : 0
ID: 3, Label [5.4 3.9 1.3 0.4], Feature : 0
ID: 4, Label [5.7 2.6 3.5 1. ], Feature : 1
ID: 5, Label [4.8 3.   1.4 0.1], Feature : 0
ID: 6, Label [5.5 2.6 4.4 1.2], Feature : 1
ID: 7, Label [7.2 3.   5.8 1.6], Feature : 2
ID: 8, Label [4.6 3.4 1.4 0.3], Feature : 0
ID: 9, Label [5.3 3.7 1.5 0.2], Feature : 0
ID: 10, Label [5.1 3.3 1.7 0.5], Feature : 0
ID: 11, Label [6.5 3.   5.2 2. ], Feature : 2
ID: 62, Label [6.9 3.2 5.7 2.3], Feature : 2
ID: 63, Label [7.2 3.2 6.   1.8], Feature : 2
ID: 64, Label [5.8 2.7 5.1 1.9], Feature : 2
ID: 65, Label [4.6 3.2 1.4 0.2], Feature : 0
ID: 66, Label [6.7 2.5 5.8 1.8], Feature : 2
ID: 67, Label [5.1 3.8 1.9 0.4], Feature : 0
ID: 68, Label [7.1 3.   5.9 2.1], Feature : 2
ID: 69, Label [4.3 3.   1.1 0.1], Feature : 0
ID: 70, Label [5.8 2.7 5.1 1.9], Feature : 2
ID: 71, Label [4.5 2.3 1.3 0.3], Feature : 0
ID: 72, Label [5.9 3.   5.1 1.8], Feature : 2
ID: 73, Label [4.6 3.1 1.5 0.2], Feature : 0
ID: 74, Label [7.7 2.6 6.9 2.3], Feature : 2
Size of Test data set 75
---------------------------------------------------
Accuracy of classification algorithm with K Neighbo
r classifier is 97.33333333333334 %
(base) MacBook-Pro-de-MARVELLOUS:iris marvellous$
```

Application 5

Supervised Machine Learning

Accuracy Calculation with Decision Tree & K Nearest Neighbour

- In this application we train iris data set with Decision Tree algorithm and K Nearest Neighbour algorithm.
- We divide iris data set into two equal parts as training data and test data.
- We apply training on training data and predict the result on test data.
- We calculate accuracy of both the algorithms by applying of test data.

Consider below characteristics of Machine Learning Application :

Classifier :	**Decision Tree & K Nearest Neighbour**
DataSet :	**Iris Dataset**
Features :	**Sepal Width, Sepal Length, Petal Width, Petal Length**
Labels :	**Versicolor, Setosa , Virginica**
Training Dataset :	**75 Entries**
Testing Dataset :	**75 Entries**

```
from sklearn import tree
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

def MarvellousCalculateAccuracyDecisionTree():
    iris = load_iris()

    data = iris.data
    target = iris.target

    data_train, data_test, target_train, target_test = train_test_split(data,target,test_size=0.5)

    classifier = tree.DecisionTreeClassifier()

    classifier.fit(data_train,target_train)

    predictions = classifier.predict(data_test)

    Accuracy = accuracy_score(target_test,predictions)

    return Accuracy

```

```
25
26 def MarvellousCalculateAccuracyKNeighbor():
27     iris = load_iris()
28
29     data = iris.data
30     target = iris.target
31
32     data_train, data_test, target_train, target_test = train_test_split(data,target,test_size=0.5)
33
34     classifier = KNeighborsClassifier()
35
36     classifier.fit(data_train,target_train)
37
38     predictions = classifier.predict(data_test)
39
40     Accuracy = accuracy_score(target_test,predictions)
41
42     return Accuracy
43
44 def main():
45     Accuracy = MarvellousCalculateAccuracyDecisionTree()
46     print("Accuracy of classification algorithm with Decision Tree  classifier is",Accuracy*100,"%")
47
48     Accuracy = MarvellousCalculateAccuracyKNeighbor()
49     print("Accuracy of classification algorithm with K Neighbor classifier is",Accuracy*100,"%")
50
51 if __name__ == "__main__":
52     main()
53
```

Output of above application

```
(base) MacBook-Pro-de-MARVELLOUS:iris marvellous$ p
ython3 irisAccuracy.py
Accuracy of classification algorithm with Decision
Tree  classifier is 94.66666666666667 %
Accuracy of classification algorithm with K Neighbo
r classifier is 97.33333333333334 %
(base) MacBook-Pro-de-MARVELLOUS:iris marvellous$
```

Accuracy with Decision Tree algorithm is 94% and with KNN is 97%.

Note : Accuracy may vary.

Clustering using K-Mean algorithm

In this case study we are generating the data set at run time randomly and apply user defined K-Mean algorithm.

```
import numpy as np
import pandas as pd
from copy import deepcopy
from matplotlib import pyplot as plt

def MarvellousKMean():
    print("_____");
    # Set three centers, the model should predict similar results
    center_1 = np.array([1,1])
    print(center_1)
    print("_____");
    center_2 = np.array([5,5])
    print(center_2)
    print("_____");
    center_3 = np.array([8,1])
    print(center_3)
    print("_____");
    # Generate random data and center it to the three centers
    data_1 = np.random.randn(7, 2) + center_1
    print("Elements of first cluster with size"+str(len(data_1)))
    print(data_1)
    print("_____");
    data_2 = np.random.randn(7,2) + center_2
    print("Elements of second cluster with size"+str(len(data_2)))
    print(data_2)
    print("_____");
    data_3 = np.random.randn(7,2) + center_3
    print("Elements of third cluster with size"+str(len(data_3)))
    print(data_3)
    print("_____");
    data = np.concatenate((data_1, data_2, data_3), axis = 0)
    print("Size of complete data set"+str(len(data)))
    print(data);
    print("_____");
    plt.scatter(data[:,0], data[:,1], s=7)
    plt.title('Marvellous Infosystems : Input Dataset')
    plt.show()
    print("_____");
    # Number of clusters
    k = 3

    # Number of training data
```

```
n = data.shape[0]
print("Total number of elements are",n)
print("_____");
# Number of features in the data
c = data.shape[1]
print("Total number of features are",c)
print("_____");
    # Generate random centers, here we use sigma and mean to ensure it
represent the whole data
mean = np.mean(data, axis = 0)
print("Value of mean",mean)
print("_____");
# Calculate standard deviation
std = np.std(data, axis = 0)
print("Value of std",std)
print("_____");
centers = np.random.randn(k,c)*std + mean
print("Random points are",centers)
print("_____");
# Plot the data and the centers generated as random
plt.scatter(data[:,0], data[:,1],c='r', s=7)
plt.scatter(centers[:,0], centers[:,1], marker='*', c='g', s=150)
plt.title('Marvellous Infosystems : Input Datase with random centroid *')
plt.show()
print("_____");

centers_old = np.zeros(centers.shape)   # to store old centers
centers_new = deepcopy(centers)         # Store new centers

print("Values of old centroids")
print(centers_old)
print("_____");

print("Values of new centroids")
print(centers_new)
print("_____");

data.shape
clusters = np.zeros(n)
distances = np.zeros((n,k))

print("Initial distances are")
print(distances)
print("_____");

error = np.linalg.norm(centers_new - centers_old)
print("value of error is ",error);
# When, after an update, the estimate of that center stays the same, exit loop
```

```
    while error != 0:
        print("value of error is ",error);
        # Measure the distance to every center
        print("Measure the distance to every center")
        for i in range(k):
            print("Iteration number ",i)
            distances[:,i] = np.linalg.norm(data - centers[i], axis=1)

        # Assign all training data to closest center
        clusters = np.argmin(distances, axis = 1)

        centers_old = deepcopy(centers_new)

        # Calculate mean for every cluster and update the center
        for i in range(k):
            centers_new[i] = np.mean(data[clusters == i], axis=0)
        error = np.linalg.norm(centers_new - centers_old)
    # end of while
    centers_new

    # Plot the data and the centers generated as random
    plt.scatter(data[:,0], data[:,1], s=7)
    plt.scatter(centers_new[:,0], centers_new[:,1], marker='*', c='g', s=150)
    plt.title('Marvellous Infosystems : Final data with Centroid')
    plt.show()

def main():
    print("---- Marvellous Infosystems by Piyush Khairnar-----")

    print("Unsuervised Machine Learning")

    print("Clustering using K Mean Algorithm")

    MarvellousKMean()

if __name__ == "__main__":
    main()
```

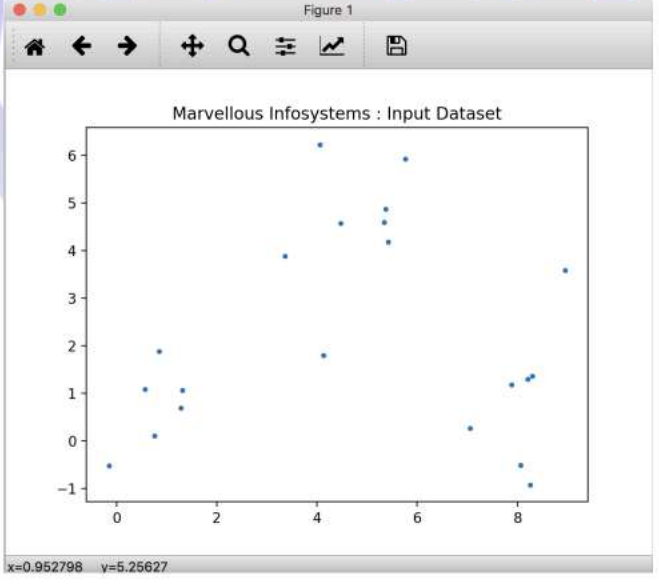
Output of above application :

```
---- Marvellous Infosystems by Piyush Khairnar——

Unsuervised Machine Learning
Clustering using K Mean Algorithm

_____________________________
[1 1]

_____________________________
[5 5]

_____________________________
[8 1]

_____________________________
Elements of first cluster with size7
[[ 0.76455047  0.1057342 ]
 [ 0.57413358  1.07954625]
 [-0.14432049 -0.52217058]
 [ 1.31367866  1.05880002]
 [ 0.85572402  1.87967134]
 [ 1.28881943  0.68895633]
 [ 4.12989909  1.80041537]]
_____________________________
Elements of second cluster with size7
[[5.34096455 4.5997487 ]
 [4.06005167 6.22535108]
 [5.42174515 4.17713132]
 [3.36119862 3.88051506]
 [4.4704562  4.57687409]
 [5.37072318 4.87360285]
 [5.76651027 5.92478869]]
_____________________________
Elements of third cluster with size7
[[ 8.29482744  1.36251521]
 [ 8.20703246  1.29568945]
 [ 8.06619888 -0.50865465]
 [ 8.94870609  3.58106472]
 [ 8.25722023 -0.9248197 ]
 [ 7.05976628  0.26199046]
 [ 7.88080164  1.17437363]]
_____________________________
Size of complete data set21
[[ 0.76455047  0.1057342 ]
 [ 0.57413358  1.07954625]
 [-0.14432049 -0.52217058]
 [ 1.31367866  1.05880002]
 [ 0.85572402  1.87967134]
 [ 1.28881943  0.68895633]
```

```
[ 4.12989909  1.80041537]
[ 5.34096455  4.5997487 ]
[ 4.06005167  6.22535108]
[ 5.42174515  4.17713132]
[ 3.36119862  3.88051506]
[ 4.4704562   4.57687409]
[ 5.37072318  4.87360285]
[ 5.76651027  5.92478869]
[ 8.29482744  1.36251521]
[ 8.20703246  1.29568945]
[ 8.06619888 -0.50865465]
[ 8.94870609  3.58106472]
[ 8.25722023 -0.9248197 ]
[ 7.05976628  0.26199046]
[ 7.88080164  1.17437363]]
```

Total number of elements are 21

Total number of features are 2



Value of mean [4.72803273 2.21862495]

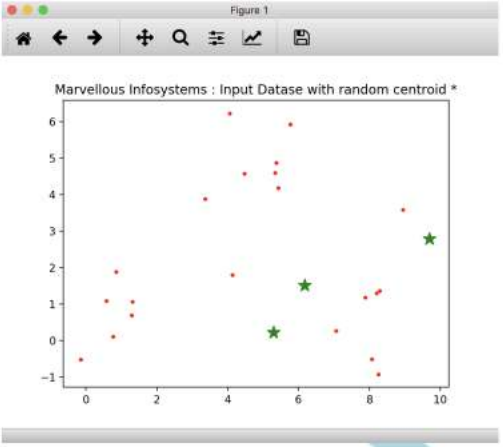
Value of std [2.94300319 2.15577759]

Random points are [[9.68991885 2.78162963]
 [6.17016879 1.5050397]
 [5.29720489 0.21581884]]

Values of old centroids
[[0. 0.]
 [0. 0.]
 [0. 0.]]

Values of new centroids
[[9.68991885 2.78162963]
 [6.17016879 1.5050397]
 [5.29720489 0.21581884]]

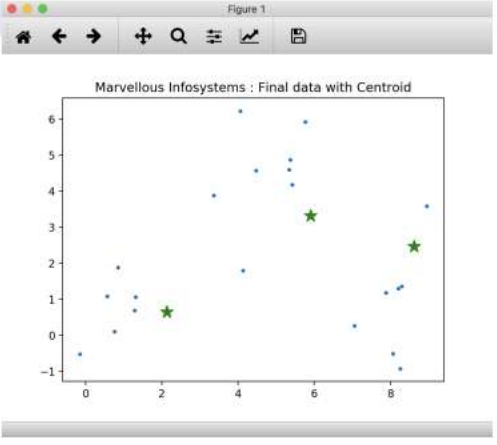
Initial distances are
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]



value of error is 13.04128351259481
value of error is 13.04128351259481

Measure the distance to every center
Iteration number 0
Iteration number 1
Iteration number 2
value of error is 3.847400069439621

Measure the distance to every center
Iteration number 0
Iteration number 1
Iteration number 2



Clustering using K-Mean algorithm

K-means clustering is a clustering algorithm that aims to partition n observations into k clusters.

There are 3 steps:

Step 1:
Initialisation – K initial "means" (centroids) are generated at random

Step 2:
Assignment – K clusters are created by associating each observation with the nearest centroid

Step 3:
Update – The centroid of the clusters becomes the new mean

Assignment and Update are repeated iteratively until convergence

The end result is that the sum of squared errors is minimised between points and their respective centroids.

```
#---------------------------------------------------------------------
import copy
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#---------------------------------------------------------------------

df = pd.DataFrame({
                'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61,
64, 69, 72],
                'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19,
7, 24]
                })

print("Step 1: Initialisation – K initial "means" (centroids) are generated at
random");

print("-----------------------------------------");
print("Data set for training");
print("-----------------------------------------");
print(df);
print("-----------------------------------------");
np.random.seed(200)
k = 3
# centroids[i] = [x, y]
```

```
centroids = {
    i+1: [np.random.randint(0, 80), np.random.randint(0, 80)]
    for i in range(k)
}
print("-------------------------------------------");
print("Random centroid generated");
print(centroids);
print("-------------------------------------------");

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color='k')

colmap = {1: 'r', 2: 'g', 3: 'b'}
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])

plt.title("Marvellous : Dataset with random centroid");

plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()

#---------------------------------------------------------------------

# Assignment – K clusters are created by associating each observation with the
nearest centroid

def assignment(df, centroids):

    for i in centroids.keys():
        # sqrt((x1 - x2)^2 - (y1 - y2)^2)
        df['distance_from_{}'.format(i)] = (
                                    np.sqrt(
                                        (df['x'] - centroids[i][0]) ** 2
                                        + (df['y'] - centroids[i][1]) ** 2
                                        )
                                    )

            centroid_distance_cols = ['distance_from_{}'.format(i) for i in
centroids.keys()]

    df['closest'] = df.loc[:, centroid_distance_cols].idxmin(axis=1)

    df['closest'] = df['closest'].map(lambda x: int(x.lstrip('distance_from_')))

    df['color'] = df['closest'].map(lambda x: colmap[x])
    return df
```

```
print("Step 2 : Assignment – K clusters are created by associating each observation with the nearest centroid");

print("Before assignment dataset");
print(df)
df = assignment(df, centroids)

print("First centroid : Red");
print("Second centroid : Green");
print("Third centroid : Blue");

print("After assignment dataset");
print(df)

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.title("Marvellous : Dataset with clustering & random centroid");
plt.show()

# ---------------------

old_centroids = copy.deepcopy(centroids)
print("Step 3:Update – The centroid of the clusters becomes the new mean Assignment and Update are repeated iteratively until convergence");

def update(k):
    print("Old values of centroids");
    print(k);

    for i in centroids.keys():
        centroids[i][0] = np.mean(df[df['closest'] == i]['x'])
        centroids[i][1] = np.mean(df[df['closest'] == i]['y'])

    print("New values of centroids");
    print(k);
    return k

centroids = update(centroids)

fig = plt.figure(figsize=(5, 5))
ax = plt.axes()
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
```

```
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)

for i in old_centroids.keys():
    old_x = old_centroids[i][0]
    old_y = old_centroids[i][1]
    dx = (centroids[i][0] - old_centroids[i][0]) * 0.75
    dy = (centroids[i][1] - old_centroids[i][1]) * 0.75
      ax.arrow(old_x, old_y, dx, dy, head_width=2, head_length=3, fc=colmap[i],
ec=colmap[i])

plt.title("Marvellous : Dataset with clustering and updated centroids");
plt.show()

#--------------------

## Repeat Assigment Stage
print("Before assignment dataset");
print(df)
df = assignment(df, centroids)
print("After assignment dataset");
print(df)

# Plot results
fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.title("Marvellous : Dataset with clustering and updated centroids");
plt.show()

# Continue until all assigned categories don't change any more
while True:
    closest_centroids = df['closest'].copy(deep=True)
    centroids = update(centroids)
    print("Before assignment dataset");
    print(df)
    df = assignment(df, centroids)
    print("After assignment dataset");
    print(df)
    if closest_centroids.equals(df['closest']):
        break

print("Final values of centroids");
print(centroids);
```

```
fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.title("Marvellous : Final dataset with set centroids");
plt.show()
```

Output of above application :

Step 1: Initialisation – K initial "means" (centroids) are generated at random

---

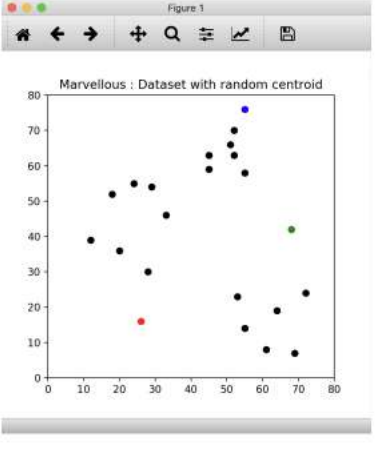
Data set for training

---

	x	y
0	12	39
1	20	36
2	28	30
3	18	52
4	29	54
5	33	46
6	24	55
7	45	59
8	45	63
9	52	70
10	51	66
11	52	63
12	55	58
13	53	23
14	55	14
15	61	8
16	64	19
17	69	7
18	72	24

Random centroid generated

{1: [26, 16], 2: [68, 42], 3: [55, 76]}



Step 2 : Assignment – K clusters are created by associating each observation with the nearest centroid

Before assignment dataset

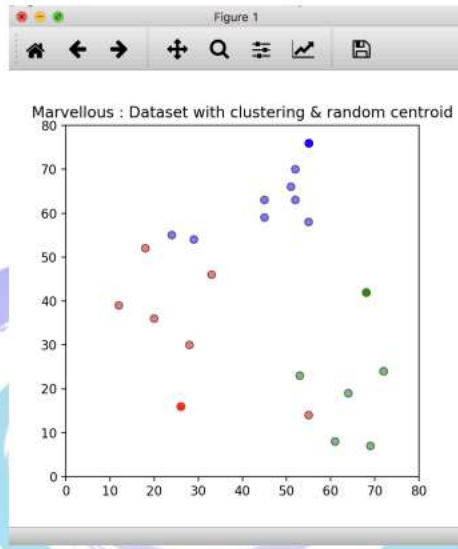
	x	y
0	12	39
1	20	36
2	28	30
3	18	52
4	29	54
5	33	46
6	24	55
7	45	59
8	45	63
9	52	70
10	51	66
11	52	63
12	55	58
13	53	23
14	55	14
15	61	8
16	64	19
17	69	7
18	72	24

First centroid : Red
Second centroid : Green
Third centroid : Blue

After assignment dataset

	x	y	distance_from_1	distance_from_2	distance_from_3	closest	color
0	12	39	26.925824	56.080300	56.727418	1	r
1	20	36	20.880613	48.373546	53.150729	1	r
2	28	30	14.142136	41.761226	53.338541	1	r
3	18	52	36.878178	50.990195	44.102154	1	r
4	29	54	38.118237	40.804412	34.058773	3	b
5	33	46	30.805844	35.227830	37.202150	1	r
6	24	55	39.051248	45.880279	37.443290	3	b
7	45	59	47.010637	28.600699	19.723083	3	b
8	45	63	50.695167	31.144823	16.401219	3	b
9	52	70	59.933296	32.249031	6.708204	3	b
10	51	66	55.901699	29.410882	10.770330	3	b
11	52	63	53.712196	26.400758	13.341664	3	b
12	55	58	51.039201	20.615528	18.000000	3	b
13	53	23	27.892651	24.207437	53.037722	2	g
14	55	14	29.068884	30.870698	62.000000	1	r
15	61	8	35.902646	34.713110	68.264193	2	g

16	64	19	38.118237	23.345235	57.706152	2	g
17	69	7	43.931765	35.014283	70.405966	2	g
18	72	24	46.690470	18.439089	54.708317	2	g



Step 3:Update – The centroid of the clusters becomes the new mean Assignment and Update are repeated iteratively until convergence
Old values of centroids
{1: [26, 16], 2: [68, 42], 3: [55, 76]}

New values of centroids
{1: [27.666666666666668, 36.166666666666664], 2: [63.8, 16.2], 3: [44.125, 61.0]}

Note : New centroids are mean of the generated X and Y coordinates of clustering members. We can use additions of distances to decide the error rate.

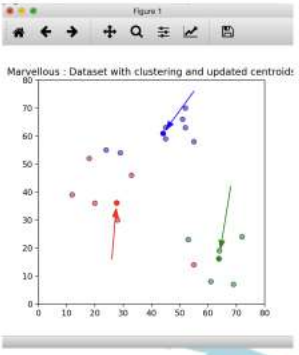
Example :
X for centroid 1 : (12+20+28+18+55) / 6 = 27.66
Y for centroid 2 : (39+36+30+52+14) / 6 = 36.16

Before assignment dataset

	x	y	distance_from_1	distance_from_2	distance_from_3	closest	color
0	12	39	26.925824	56.080300	56.727418	1	r
1	20	36	20.880613	48.373546	53.150729	1	r
2	28	30	14.142136	41.761226	53.338541	1	r
3	18	52	36.878178	50.990195	44.102154	1	r
4	29	54	38.118237	40.804412	34.058773	3	b
5	33	46	30.805844	35.227830	37.202150	1	r
6	24	55	39.051248	45.880279	37.443290	3	b
7	45	59	47.010637	28.600699	19.723083	3	b
8	45	63	50.695167	31.144823	16.401219	3	b
9	52	70	59.933296	32.249031	6.708204	3	b
10	51	66	55.901699	29.410882	10.770330	3	b
11	52	63	53.712196	26.400758	13.341664	3	b
12	55	58	51.039201	20.615528	18.000000	3	b
13	53	23	27.892651	24.207437	53.037722	2	g
14	55	14	29.068884	30.870698	62.000000	1	r
15	61	8	35.902646	34.713110	68.264193	2	g
16	64	19	38.118237	23.345235	57.706152	2	g
17	69	7	43.931765	35.014283	70.405966	2	g
18	72	24	46.690470	18.439089	54.708317	2	g

After assignment dataset

	x	y	distance_from_1	distance_from_2	distance_from_3	closest	color
0	12	39	15.920811	56.595760	38.936045	1	r
1	20	36	7.668478	48.067453	34.742130	1	r
2	28	30	6.175669	38.367695	34.943034	1	r
3	18	52	18.550981	58.131575	27.631786	1	r
4	29	54	17.883108	51.379763	16.666302	3	b
5	33	46	11.186549	42.856505	18.675268	1	r
6	24	55	19.186946	55.583091	21.000372	1	r
7	45	59	28.667151	46.746979	2.183031	3	b
8	45	63	31.944831	50.434909	2.183031	3	b
9	52	70	41.674999	55.078853	11.958914	3	b
10	51	66	37.874427	51.418674	8.500919	3	b
11	52	63	36.223458	48.264687	8.125000	3	b
12	55	58	34.982932	42.716273	11.281207	3	b
13	53	23	28.550637	12.762445	39.022630	2	g
14	55	14	35.191934	9.070832	48.241742	2	g
15	61	8	43.640259	8.664872	55.621629	2	g
16	64	19	40.184643	2.807134	46.465209	2	g
17	69	7	50.587932	10.567876	59.453895	2	g
18	72	24	45.972516	11.317243	46.325108	2	g



Old values of centroids
{1: [27.666666666666668, 36.166666666666664], 2: [63.8, 16.2], 3: [44.125, 61.0]}

New values of centroids
{1: [22.5, 43.0], 2: [62.333333333333336, 15.833333333333334], 3: [47.0, 61.857142857142854]}

Before assignment dataset

	x	y	distance_from_1	distance_from_2	distance_from_3	closest	color
0	12	39	15.920811	56.595760	38.936045	1	r
1	20	36	7.668478	48.067453	34.742130	1	r
2	28	30	6.175669	38.367695	34.943034	1	r
3	18	52	18.550981	58.131575	27.631786	1	r
4	29	54	17.883108	51.379763	16.666302	3	b
5	33	46	11.186549	42.856505	18.675268	1	r
6	24	55	19.186946	55.583091	21.000372	1	r
7	45	59	28.667151	46.746979	2.183031	3	b
8	45	63	31.944831	50.434909	2.183031	3	b
9	52	70	41.674999	55.078853	11.958914	3	b
10	51	66	37.874427	51.418674	8.500919	3	b
11	52	63	36.223458	48.264687	8.125000	3	b
12	55	58	34.982932	42.716273	11.281207	3	b
13	53	23	28.550637	12.762445	39.022630	2	g
14	55	14	35.191934	9.070832	48.241742	2	g
15	61	8	43.640259	8.664872	55.621629	2	g
16	64	19	40.184643	2.807134	46.465209	2	g
17	69	7	50.587932	10.567876	59.453895	2	g
18	72	24	45.972516	11.317243	46.325108	2	g

After assignment dataset

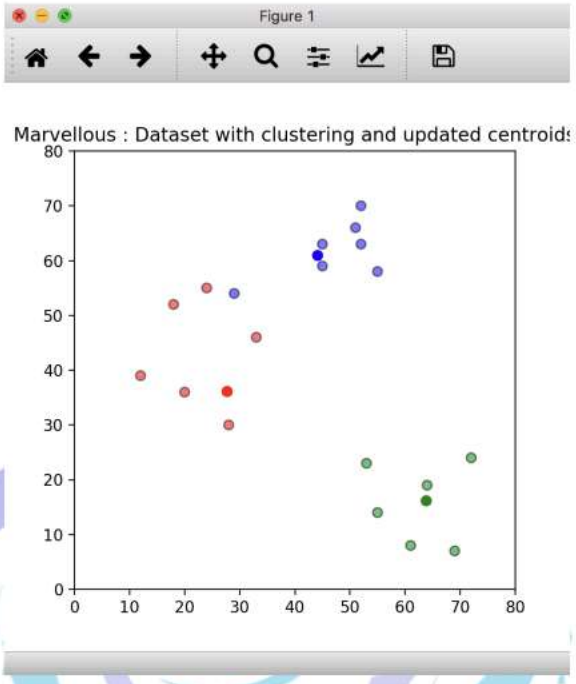
	x	y	distance_from_1	distance_from_2	distance_from_3	closest	color
0	12	39	11.236103	55.408834	41.802500	1	r
1	20	36	7.433034	46.891423	37.384380	1	r
2	28	30	14.115594	37.141247	37.092823	1	r
3	18	52	10.062306	57.214266	30.629451	1	r
4	29	54	12.776932	50.673519	19.640130	1	r
5	33	46	10.920165	42.076980	21.152990	1	r
6	24	55	12.093387	54.803943	24.000425	1	r
7	45	59	27.608875	46.516723	3.487587	3	b
8	45	63	30.103986	50.250760	2.303502	3	b
9	52	70	39.990624	55.143500	9.555424	3	b
10	51	66	36.623080	51.430914	5.758756	3	b
11	52	63	35.640567	48.285321	5.128949	3	b
12	55	58	35.794553	42.799598	8.881303	3	b
13	53	23	36.472592	11.767422	39.317649	2	g
14	55	14	43.557433	7.559027	48.521193	2	g
15	61	8	52.031241	7.945998	55.647029	2	g
16	64	19	47.940067	3.578485	46.105690	2	g
17	69	7	58.806887	11.066717	59.104197	2	g
18	72	24	53.021222	12.654600	45.366984	2	g

Old values of centroids

{1: [22.5, 43.0], 2: [62.333333333333336, 15.833333333333334], 3: [47.0, 61.857142857142854]}

New values of centroids

{1: [23.428571428571427, 44.57142857142857], 2: [62.333333333333336, 15.833333333333334], 3: [50.0, 63.166666666666664]}



Before assignment dataset

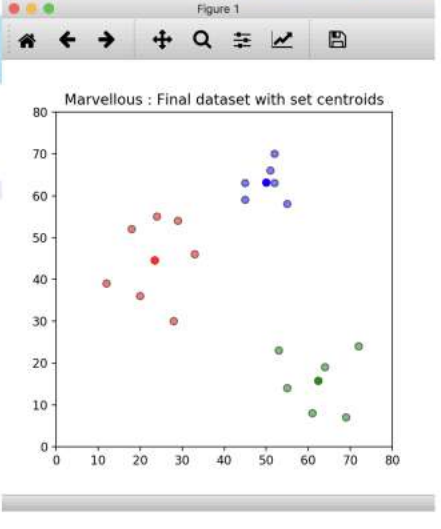
	x	y	distance_from_1	distance_from_2	distance_from_3	closest	color
0	12	39	11.236103	55.408834	41.802500	1	r
1	20	36	7.433034	46.891423	37.384380	1	r
2	28	30	14.115594	37.141247	37.092823	1	r
3	18	52	10.062306	57.214266	30.629451	1	r
4	29	54	12.776932	50.673519	19.640130	1	r
5	33	46	10.920165	42.076980	21.152990	1	r
6	24	55	12.093387	54.803943	24.000425	1	r
7	45	59	27.608875	46.516723	3.487587	3	b
8	45	63	30.103986	50.250760	2.303502	3	b
9	52	70	39.990624	55.143500	9.555424	3	b
10	51	66	36.623080	51.430914	5.758756	3	b
11	52	63	35.640567	48.285321	5.128949	3	b
12	55	58	35.794553	42.799598	8.881303	3	b
13	53	23	36.472592	11.767422	39.317649	2	g
14	55	14	43.557433	7.559027	48.521193	2	g
15	61	8	52.031241	7.945998	55.647029	2	g
16	64	19	47.940067	3.578485	46.105690	2	g
17	69	7	58.806887	11.066717	59.104197	2	g
18	72	24	53.021222	12.654600	45.366984	2	g

After assignment dataset

	x	y	distance_from_1	distance_from_2	distance_from_3	closest	color
0	12	39	12.714286	55.408834	45.033629	1	r
1	20	36	9.231711	46.891423	40.472556	1	r
2	28	30	15.271689	37.141247	39.799846	1	r
3	18	52	9.200710	57.214266	33.892395	1	r
4	29	54	10.951656	50.673519	22.913485	1	r
5	33	46	9.677451	42.076980	24.159769	1	r
6	24	55	10.444215	54.803943	27.252421	1	r
7	45	59	25.952075	46.516723	6.508541	3	b
8	45	63	28.371443	50.250760	5.002777	3	b
9	52	70	38.248383	55.143500	7.120003	3	b
10	51	66	34.919441	51.430914	3.004626	3	b
11	52	63	33.999100	48.285321	2.006932	3	b
12	55	58	34.308623	42.799598	7.189885	3	b
13	53	23	36.603223	11.767422	40.278544	2	g
14	55	14	43.947325	7.559027	49.420250	2	g
15	61	8	52.431685	7.945998	56.252654	2	g
16	64	19	47.957677	3.578485	46.332434	2	g
17	69	7	59.062402	11.066717	59.293292	2	g
18	72	24	52.748150	12.654600	44.922464	2	g

Final values of centroids

{1: [23.428571428571427, 44.57142857142857], 2: [62.333333333333336, 15.833333333333334], 3: [50.0, 63.166666666666664]}



K-Means Clustering

K-Means is an **unsupervised learning algorithm** used for **clustering** data into **K groups** based on feature similarity.

- It tries to **minimize intra-cluster variance** (data points within a cluster should be close to each other).
- It is iterative and works well with numeric data.

Algorithm Steps

1. **Select the value of K (number of clusters).**
2. **Initialize K centroids randomly.**
3. **Assign each data point to the closest centroid** (using distance like Euclidean).
4. **Recalculate the centroids** (mean of points in each cluster).
5. **Repeat steps 3 and 4 until centroids do not change** (or max iterations reached).

K-Means Clustering (2D Example)

We will work with **2D data points** to understand clustering better using **Euclidean distance**.

📊 Sample 2D Data Points:

Let's assume we have 6 points in 2D space:

```
P1 = (2, 10)
P2 = (2, 5)
P3 = (8, 4)
P4 = (5, 8)
P5 = (7, 5)
P6 = (6, 4)
```

We'll choose **K = 2 (2 clusters)**

Step 1: Initialization

Randomly select any **2 points** as **initial centroids**:

Let:

- C1 = P1 = (2, 10)
- C2 = P4 = (5, 8)

Step 2: Calculate Distance and Assign Cluster

Calculate distances of each point to centroids C1 and C2:

Point	Coordinates	d(C1) from (2,10)	d(C2) from (5,8)	Assigned Cluster
P1	(2, 10)	0.00	3.61	C1
P2	(2, 5)	5.00	3.61	C2
P3	(8, 4)	8.48	5.00	C2
P4	(5, 8)	3.61	0.00	C2
P5	(7, 5)	7.07	3.61	C2
P6	(6, 4)	7.21	4.47	C2

Step 3: Recalculate Centroids

Clusters:

- Cluster 1 (C1): P1 = (2, 10)
- Cluster 2 (C2): P2, P3, P4, P5, P6

New Centroid C1: mean of [(2,10)] = (2, 10) — unchanged

New Centroid C2: mean of P2–P6

x=(2+8+5+7+6)/5=28/5=5.6
y=(5+4+8+5+4)/5=26/5=5.2

New Centroid C2 = **(5.6, 5.2)**

Step 4: Reassign Points Using New Centroids

Point	Coordinates	d(C1: 2,10)	d(C2: 5.6, 5.2)	New Cluster
P1	(2, 10)	0.00	6.23	C1
P2	(2, 5)	5.00	3.61	C2
P3	(8, 4)	8.48	2.55	C2
P4	(5, 8)	3.61	2.87	C2
P5	(7, 5)	7.07	1.45	C2
P6	(6, 4)	7.21	1.62	C2

No change in cluster assignment. Algorithm **converged**.

Final Clusters:

- **Cluster 1**: (2, 10)
  Centroid: (2, 10)
- **Cluster 2**: (2, 5), (8, 4), (5, 8), (7, 5), (6, 4)
  Centroid: (5.6, 5.2)

Consider below Python Code

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans


# 2D points

X = np.array([[2,10], [2,5], [8,4], [5,8], [7,5], [6,4]])


# KMeans with 2 clusters

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)


# Plot

plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, s=100)

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],c='black', s=200, marker='X', label='Centroids')

plt.title("K-Means Clustering (2D)")

plt.xlabel("X")

plt.ylabel("Y")

plt.legend()

plt.grid(True)

plt.show()
```

K Mean Algorithm For Clustering

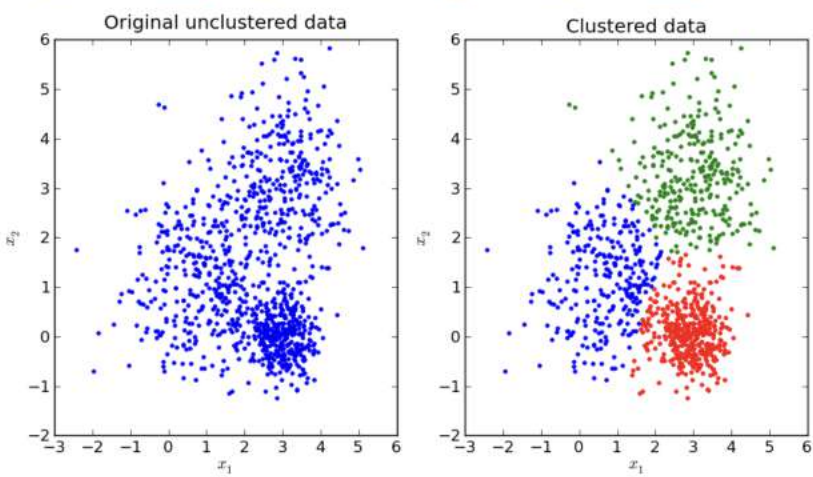
Clustering is a type of Unsupervised learning.
This is very often used when you don't have labeled data.
K-Means Clustering is one of the popular clustering algorithm.
The goal of this algorithm is to find groups(clusters) in the given data.

K Means Clustering is one of the most popular Machine Learning algorithms for cluster analysis in data mining. K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

K Means algorithm is an unsupervised learning algorithm, ie. it needs no training data, it performs the computation on the actual dataset. This should be apparent from the fact that in K Means, we are just trying to group similar data points into clusters, there is no prediction involved.

The K Means algorithm is easy to understand and to implement. It works well in a large number of cases and is a powerful tool to have in the closet.
The algorithm has a loose relationship to the k-nearest neighbor classifier, a popular machine learning technique for classification.

K-Means is a very simple algorithm which clusters the data into K number of clusters. The following image from PyPR is an example of K-Means Clustering.



The K Means algorithm is iterative based, it repeatedly calculates the cluster centroids, refining the values until they do not change much.

The k-means algorithm takes a dataset of 'n' points as input, together with an integer parameter 'k' specifying how many clusters to create(supplied by the programmer). The output is a set of 'k' cluster centroids and a labeling of the dataset that maps each of the data points to a unique cluster.

In the beginning, the algorithm chooses k centroids in the dataset. Then it calculates the distance of each point to each centroid. Each centroid represents a cluster and the points closest to the centroid are assigned to the cluster. At the end of the first iteration, the centroid values are recalculated, usually taking the arithmetic mean of all points in the cluster.

After the new values of centroid are found, the algorithm performs the same set of steps over and over again until the differences between old centroids and the new centroids are negligible.

K Mean Algorithm

Our algorithm works as follows, assuming we have inputs x1,x2,x3,…,xn and value of K

Step 1 -
Pick K random points as cluster centers called centroids.

Step 2 -
Assign each x_i to nearest cluster by calculating its distance to each centroid.

Step 3 -
Find new cluster center by taking the average of the assigned points.

Step 4 -
Repeat Step 2 and 3 until none of the cluster assignments change.

Detailed Explanation :

Step 1
We randomly pick K cluster centers(centroids). Let's assume these are
c_1, c_2, ..., c_k, and we can say that;
C = c_1, c_2,..., c_k
C is the set of all centroids.

Step 2
In this step we assign each input value to closest center. This is done by calculating Euclidean(L2) distance between the point and the each centroid.

$$\arg\min_{c_i \in C} dist(c_i, x)^2$$

Where dist(.) is the Euclidean distance.

Step 3
In this step, we find the new centroid by taking the average of all the points assigned to that cluster.

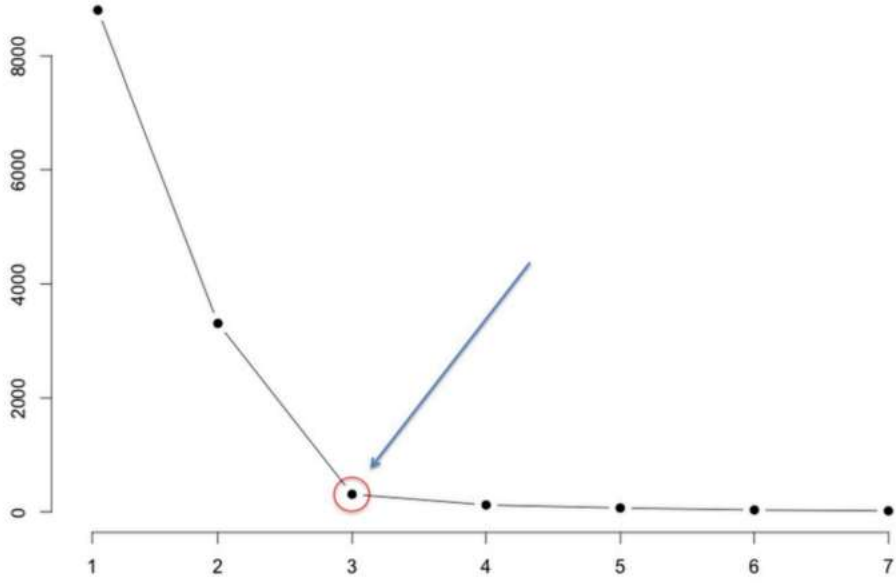
$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

Si is the set of all points assigned to the ith cluster.

Step 4
In this step, we repeat step 2 and 3 until none of the cluster assignments change. That means until our clusters remain stable, we repeat the algorithm.

Choosing the Value of K

We often know the value of K. In that case we use the value of K. Else we use the Elbow Method.



We run the algorithm for different values of K(say K = 10 to 1) and plot the K values against SSE(Sum of Squared Errors). And select the value of K for the elbow point as shown in the figure.

Ensemble Machine Learning

Ensemble learning is a machine learning technique where **multiple models (weak learners)** are trained and combined to solve the same problem in order to get **better accuracy, stability, and robustness**.

Types of Ensemble Methods

1. **Bagging (Bootstrap Aggregating)**
2. **Boosting**

Bagging (Bootstrap Aggregating)

Train **multiple models independently** on **random subsets** of the data and **average/vote their predictions**.

How it works:

1. Create **n random subsets** of data from the original dataset **with replacement** (bootstrap sampling).
2. Train a **separate model** (often the same algorithm, like Decision Tree) on each subset.
3. Combine the predictions:
    - **Classification** → Majority Voting
    - **Regression** → Averaging

Benefits:

- Reduces **variance**
- Avoids **overfitting**
- Works well with **unstable models** (e.g., Decision Trees)

Popular Bagging Algorithm:

- **Random Forest** – An ensemble of decision trees using bagging + feature randomness.

Diagram:

```
Original Dataset
     ↓   ↓   ↓   ↓
Bootstrap Sample 1 → Model 1
Bootstrap Sample 2 → Model 2
Bootstrap Sample 3 → Model 3
     ↓   ↓   ↓
    Voting / Averaging
         ↓
     Final Prediction
```

Boosting

Train models **sequentially**, where **each model learns from the mistakes** of the previous ones.

How it works:

1. Start with an initial model (usually weak like a shallow tree).
2. Evaluate errors → focus more on **incorrectly predicted data points**.
3. Train the next model on the updated data (adjusted weights).
4. Repeat for n iterations.
5. Combine all models using **weighted voting/averaging**.

Benefits:

- Reduces **bias**
- Improves performance on **complex datasets**
- Learns from mistakes

Drawbacks:

- Can **overfit** if not tuned properly
- **Slower** than bagging (sequential training)

Popular Boosting Algorithms:

Algorithm	Key Features
AdaBoost	Adjusts weights based on errors
Gradient Boosting	Trains new models to predict **residuals (errors)**
XGBoost	Optimized version of Gradient Boosting (fast, regularized)
LightGBM	Faster and more scalable boosting using histogram-based splits
CatBoost	Handles categorical features automatically

Bagging vs Boosting Comparison:

Feature	Bagging	Boosting
Model Training	**Parallel** (independent)	**Sequential** (one after another)
Focus	Reduce **variance**	Reduce **bias** and variance
Example	Random Forest	AdaBoost, Gradient Boosting
Overfitting	Less prone	More prone if not regularized
Speed	Faster (can train in parallel)	Slower (sequential training)

When to Use What :

Situation	Best Technique
High variance model (overfitting)	**Bagging**
High bias model (underfitting)	**Boosting**
Small to medium data	**Boosting**
Large-scale data	**Bagging or XGBoost**

Real-Life Example:

- **Bagging** (Random Forest): Useful in spam detection, credit scoring, etc.
- **Boosting** (XGBoost, LightGBM): Common in Kaggle competitions, customer churn prediction, fraud detection.

Ensemble Machine Learning application with heterogeneous algorithm technique

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
x = iris['data']
y = iris['target']

x_train, x_test, y_train, y_test = train_test_split(x, y,
random_state = 42, train_size = 0.85)

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
knn_clf = KNeighborsClassifier()

vot_clf = VotingClassifier(estimators = [('lr', log_clf), ('rnd',
rnd_clf), ('knn', knn_clf)], voting = 'hard')

vot_clf.fit(x_train, y_train)

pred = vot_clf.predict(x_test)

print("Testing accuracy is : ",accuracy_score(y_test,
pred)*100)
```

Ensemble Machine Learning application with Boosting technique

MNIST case study :

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn. ensemble import AdaBoostClassifier

data = pd.read_csv('mnist.csv')

df_x = data.iloc[:,1:]  # Labels
df_y = data.iloc[:,0]   # Pixels

x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2,
random_state=4)

obj = DecisionTreeClassifier(___, ___, ___);
adb = AdaBoostClassifier(obj,n_estimators =___, learning_rate = __);
adb = AdaBoostClassifier(DecisionTreeClassifier(),n_estimators = 100 , learning_rate =
1)
adb.fit(x_train,y_train)

print("Testing accuracy using bagging classifier : ",adb.score(x_test,y_test)*100)

print("Training accuracy using bagging classifier : ",adb.score(x_train,y_train)*100)
```

Iris Case study :

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets
# Import train_test_split function
from sklearn.model_selection import train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Load data
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training
and 30% test

# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=50,
                         learning_rate=1)
# Train Adaboost Classifer
model = abc.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = model.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

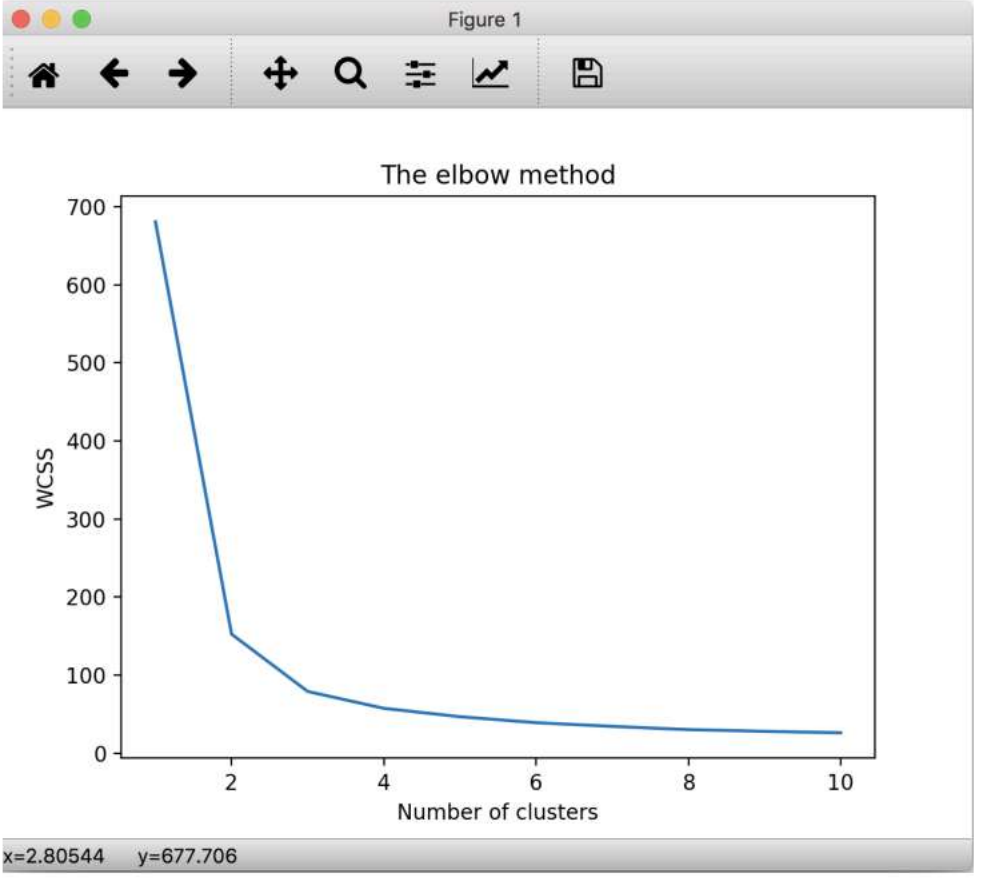
Clustering using K-Mean algorithm

In this case study we are using Iris dataset with K-Mean algorithm from sklearn.

```
#importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#importing the Iris dataset with pandas
dataset = pd.read_csv('iris.csv')
x = dataset.iloc[:, [0, 1, 2, 3]].values

#Finding the optimum number of clusters for k-means classification
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init
= 10, random_state = 0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)

#Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')  #within cluster sum of squares
plt.show()

#Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init =
10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)

#Visualising the clusters
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'red', label
= 'Iris-setosa')

plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'blue', label
= 'Iris-versicolour')

plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'green',
label = 'Iris-virginica')

#Plotting the centroids of the clusters
```

```
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100,
c = 'yellow', label = 'Centroids')

plt.legend()

plt.show()
```

Output of above application :



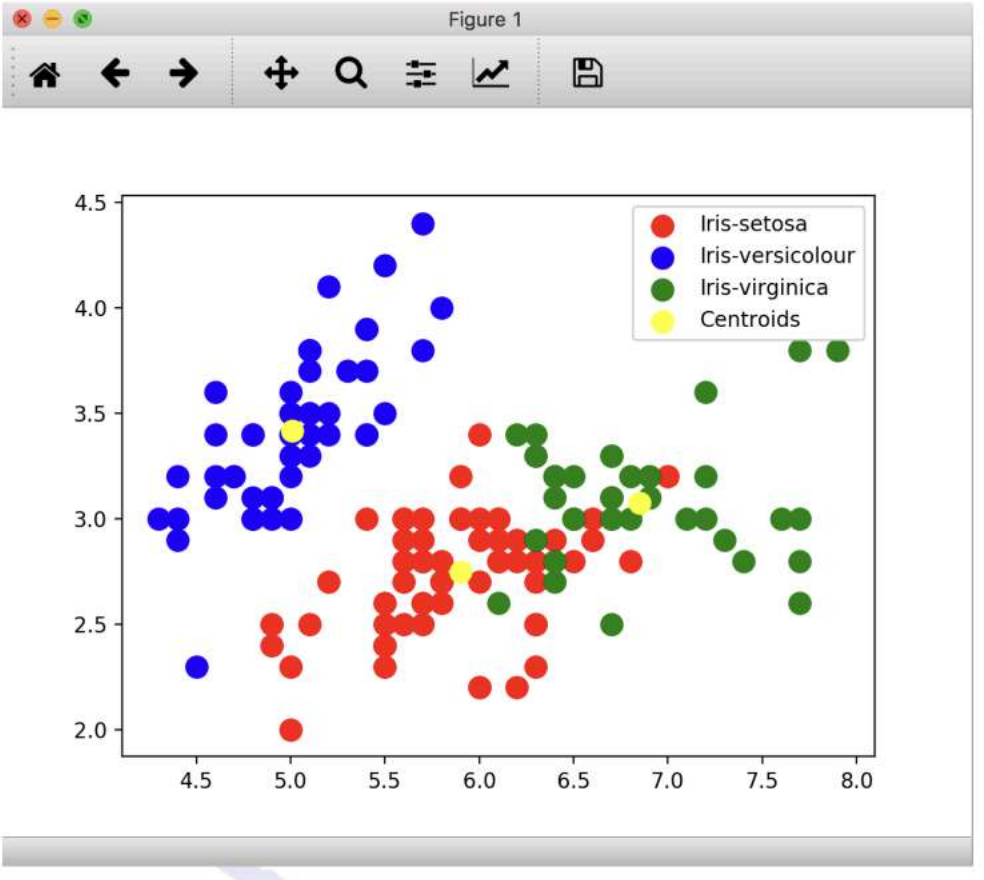


Figure 1
Iris-setosa
Iris-versicolour
Iris-virginica
Centroids
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

Bias, Variance, Overfitting & Underfitting

Bias

- **Meaning:** Error caused by a model making **oversimplified assumptions**, ignoring important details.
- **Effect:** Predictions are **consistently wrong** (model under-learns).
- **Example:**
  Predicting every student's exam score as **50 marks**, no matter how much they studied.

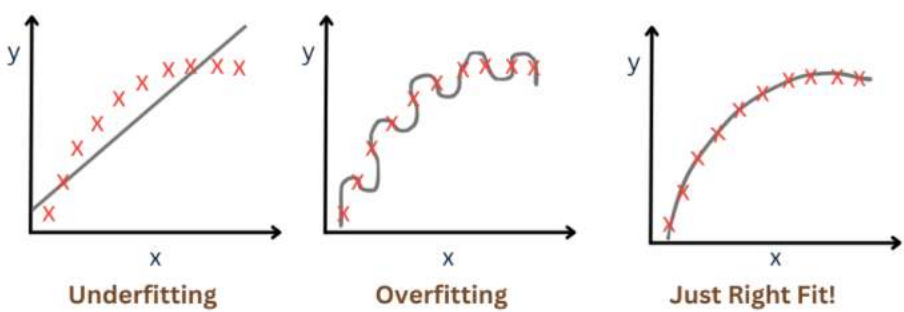
High Bias → Underfitting.

Variance

- **Meaning:** Error caused by a model being **too sensitive to training data**, learning **noise and random patterns** instead of real trends.
- **Effect:** Predictions are **unstable** – they change a lot for new data.
- **Example:**
  Model remembers every student's data exactly but **gives wrong results for new students**.

High Variance → Overfitting.

Model Underfitting, Overfitting and Right Fit



Bias	High	Low	Low
Variance	Low	High	Low

Underfitting

- **Definition:**
  Underfitting happens when a model is **too simple** to learn the data properly.
- **Cause:**
  - Few features used
  - Very basic algorithm
  - Less training
- **Effect:**
  - **High bias, low variance**
  - Bad accuracy on training and test data
- **Example:**
  Using a **straight line** to predict house prices when the relation is more complex.
- **Real-life analogy:**
  A **student not studying enough**, always performing poorly in exams.

Overfitting

- **Definition:**
  Overfitting happens when a model **learns too much from the training data**, including noise and unnecessary patterns.
- **Cause:**
  - Very complex model
  - Small or noisy dataset
  - No regularization
- **Effect:**
  - **Low bias, high variance**
  - Perfect accuracy on training data but poor on test data
- **Example:**
  A **zigzag curve** that passes through every training point but fails on new data.
- **Real-life analogy:**
  A **student memorizing answers word-for-word**, failing to answer new questions.

Best Fit Model

- **Goal:** Achieve **low bias and low variance**.
- **Effect:** Model learns the **right patterns**, ignoring noise, and works well on unseen data.
- **Example:**
  A **balanced study approach** – understanding concepts, not just memorizing.

Bias-Variance Trade-off

- **Rule:**

Total Error = Bias^2 + Variance + Noise

- Increasing model complexity:
  - Bias ↓ (model learns more)
  - Variance ↑ (model overfits easily)
- **Ideal Point:** Balanced error (best fit).

Archery Target Analogy

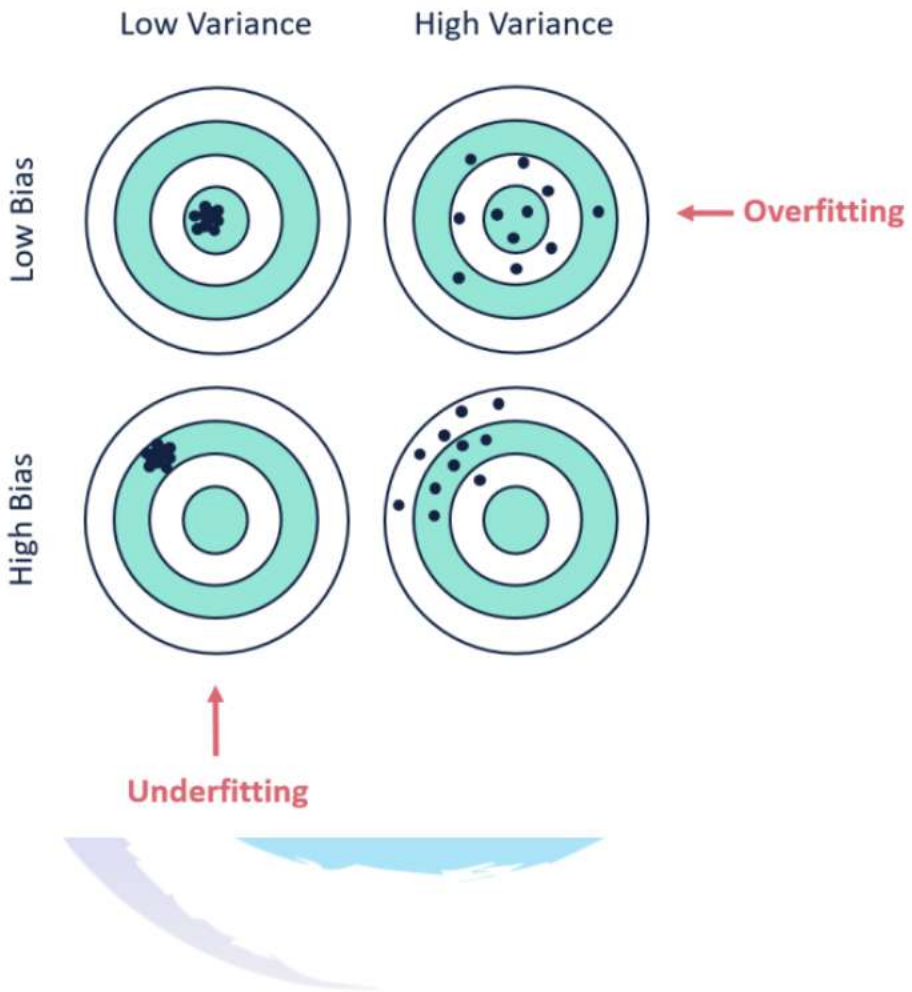
- **High Bias (Underfitting):** Arrows far from bullseye, close together → **Consistently wrong.**
- **High Variance (Overfitting):** Arrows spread all over → **Inconsistent.**
- **Best Fit:** Arrows close to bullseye → **Accurate and stable predictions.**

- **Underfitting → Model too simple → High Bias.**
- **Overfitting → Model too complex → High Variance.**
- **Best Fit → Balanced model → Low Bias + Low Variance.**

Definition of Noise

Noise = **Errors or variations in data** that occur due to:

- Measurement mistakes
- Random fluctuations
- Missing or incorrect data entries
- Factors that are **not part of the actual pattern** we are trying to learn.


Low Variance
High Variance
Low Bias
High Bias
Overfitting
Underfitting

Characteristics of Noise

- Noise is **unwanted information** in the dataset.
- It **cannot be predicted** by the model.
- It **adds complexity** and can cause **overfitting** if the model tries to learn it.
- Even the **best model cannot fully eliminate noise** (called **irreducible error**).

Examples of Noise

Example 1: House Price Prediction

- Input features: Size of house, location, number of rooms.
- **Noise:**
  - Typo in data: Price entered as **10,000,000 instead of 1,000,000**.
  - A sudden one-time **discount or premium** due to seller urgency.

Example 2: Student Exam Scores

- Inputs: Hours studied, practice tests taken.
- **Noise:**
  - A student **feels unwell on exam day** → unusually low score.
  - Random guessing of answers → unpredictable results.

Training Error and Testing Error

- **Training Error:**
  The error (difference between predicted and actual values) when the model is tested on **data it was trained on**.
  - Shows **how well the model learned the training data**.
- **Testing Error (Generalization Error):**
  The error when the model is tested on **new, unseen data**.
  - Shows **how well the model generalizes** to unknown data.

Role of Bias and Variance

- **Bias:**
  - Error due to **wrong assumptions** (model too simple).
  - Leads to **high training and testing error** (underfitting).
- **Variance:**
  - Error due to **over-sensitivity** to training data (model too complex).
  - Leads to **low training error but high testing error** (overfitting).
- **Noise:**
  - Random errors in data that no model can fix.

Training vs Testing Error Behavior

Model Complexity	Bias	Variance	Training Error	Testing Error
Underfitting (Too Simple)	High	Low	High	High
Overfitting (Too Complex)	Low	High	Very Low	High
Best Fit (Balanced)	Low	Low	Low	Low

Example: Student Exam Score Prediction

- **Underfitting:**
  Model says every student scores **50 marks**, ignoring all features.
  - Training error = **High**
  - Testing error = **High**
  - Reason = **High Bias**
- **Overfitting:**
  Model memorizes every student's exact data.
  - Training error = **Almost 0**
  - Testing error = **High** (fails on new students)
  - Reason = **High Variance**
- **Best Fit:**
  Model learns a general rule: *"More study hours = Higher score"*.
  - Training error = **Low**
  - Testing error = **Low**
  - Reason = **Balanced Bias and Variance**

Support Vector Machine

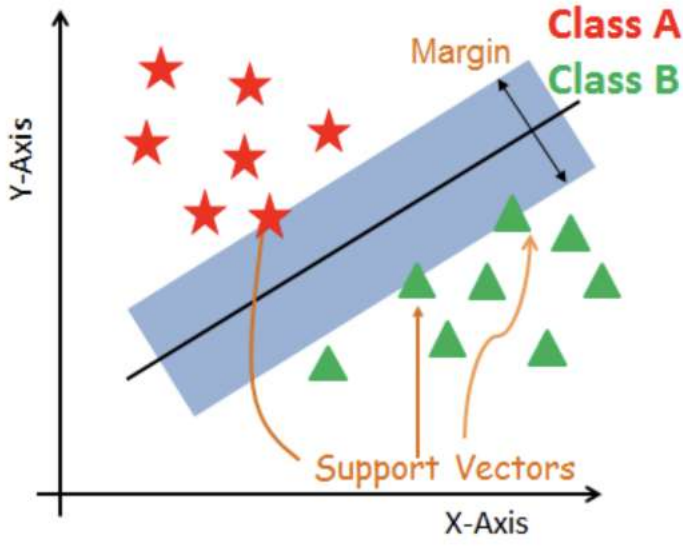
SVM offers very high accuracy compared to other classifiers such as logistic regression, and decision trees. It is known for its kernel trick to handle nonlinear input spaces. It is used in a variety of applications such as face detection, intrusion detection, classification of emails, news articles and web pages, classification of genes, and handwriting recognition.

Support Vector Machines

Support Vector Machines is considered to be a classification approach. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes.

SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error.

The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.



Support Vectors

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

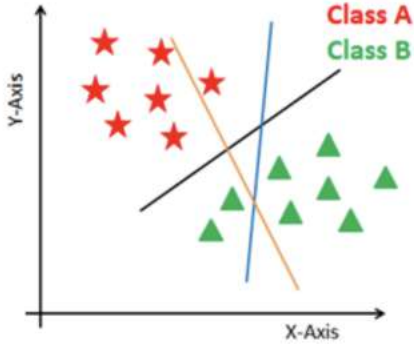
Hyperplane
A hyperplane is a decision plane which separates between a set of objects having different class memberships.

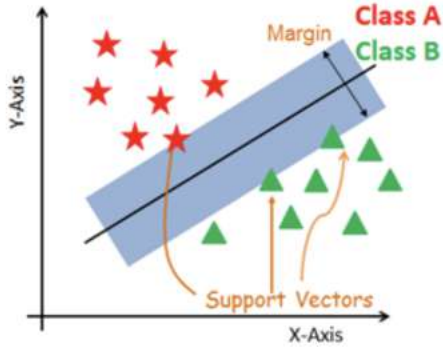
Margin
A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

How does SVM work?
The main objective is to segregate the given dataset in the best possible way. The distance between the either nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum marginal hyperplane in the following steps:

1. Generate hyperplanes which segregates the classes in the best way. Left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.

2. Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure.





Breast Cancer Dataset with Support Vector Machine

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics

def MarvellousSVM():
    #Load dataset
    cancer = datasets.load_breast_cancer()

    # print the names of the 13 features
    print("Features of the cancer dataset : ", cancer.feature_names)

    # print the label type of cancer('malignant' 'benign')
    print("Labels of the cancer dataset : ", cancer.target_names)

    # print data(feature)shape
    print("Shape of dataset is :",cancer.data.shape)

    # print the cancer data features (top 5 records)
    print("First 5 records are : ")
    print(cancer.data[0:5])

    # print the cancer labels (0:malignant, 1:benign)
    print("Target of dataset : ", cancer.target)

    # Split dataset into training set and test set
    X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
        test_size=0.3,random_state=109) # 70% training and 30% test

    #Create a svm Classifier
    clf = svm.SVC(kernel='linear') # Linear Kernel

    #Train the model using the training sets
    clf.fit(X_train, y_train)

    #Predict the response for test dataset
    y_pred = clf.predict(X_test)

    # Model Accuracy: how often is the classifier correct?
    print("Accuracy of the model is :",metrics.accuracy_score(y_test, y_pred)*100)

def main():
    print("_______ Marvellous Support Vector Machine _________")

    MarvellousSVM()

if __name__ == "__main__":
    main()
```

Preserving Machine Learning Models on HDD using joblib with Pipeline

When a machine learning model is trained, it learns from data and stores patterns/ parameters.
If you close the program, the trained parameters are lost unless you **save the model**.
Saving:

- Avoids retraining every time
- Saves computation time
- Enables deployment or sharing

joblib :

- Specially optimized for **large NumPy arrays** (common in ML models)
- Faster than pickle for ML models
- Easy syntax: just **dump** and **load**

Saving a Model to HDD

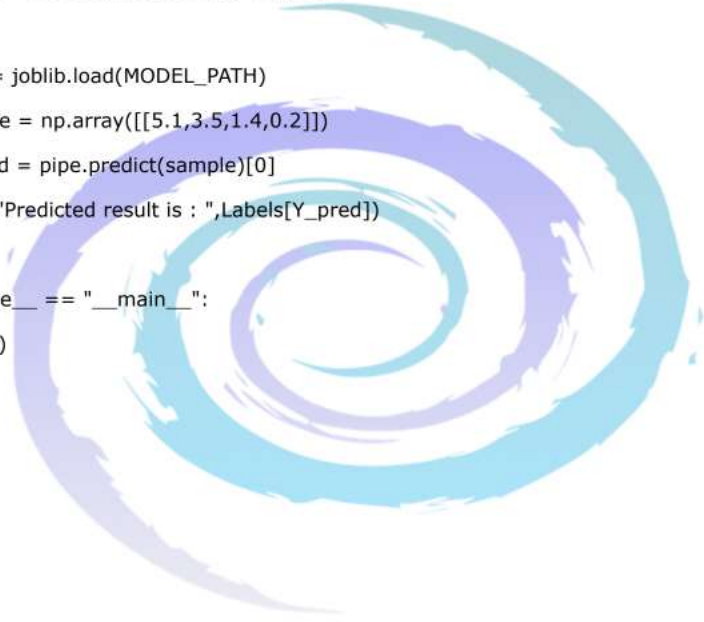
```
from pathlib import Path

import joblib

import numpy as np

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Configuration details

ARTIFACTS = Path("artifacts_sample")

ARTIFACTS.mkdir(exist_ok=True)

MODEL_PATH = ARTIFACTS / "iris_pipeline.joblib"

RANDOM_STATE = 42

TEST_SIZE = 0.2
```

```
def main():
    iris = load_iris()
    X =  iris.data
    Y = iris.target
    X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=TEST_SIZE,
random_state=RANDOM_STATE)
    pipe = Pipeline([
        ("scalar", StandardScaler()),
        ("clf", LogisticRegression(max_iter=1000))
    ])
    pipe.fit(X_train,Y_train)
    Y_pred = pipe.predict(X_test)

    print("Accuracy Score : ",accuracy_score(Y_test,Y_pred))

    joblib.dump(pipe,MODEL_PATH)

if __name__ == "__main__":
    main()
```

Loading a Model from HDD

```
from pathlib import Path
import joblib
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Configuration details
```

```
ARTIFACTS = Path("artifacts_sample")
ARTIFACTS.mkdir(exist_ok=True)
MODEL_PATH = ARTIFACTS / "iris_pipeline.joblib"
RANDOM_STATE = 42
TEST_SIZE = 0.2


def main():
    Labels = ['Setosa','Versicolor','Virginica']


    pipe = joblib.load(MODEL_PATH)
    sample = np.array([[5.1,3.5,1.4,0.2]])
    Y_pred = pipe.predict(sample)[0]
    print("Predicted result is : ",Labels[Y_pred])


if __name__ == "__main__":
    main()
```



Pipeline in Machine Learning

A **Pipeline** is a way to **chain preprocessing steps and a model together** into one object.

Advantages Pipeline :

- Keeps **data transformations & model** in one place
- Ensures the **same preprocessing** is applied during both training and prediction
- Reduces human error in applying transformations
- Makes saving/loading easier (one file instead of many)

Creating and Saving a Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
import joblib
from sklearn.datasets import load_iris

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Create pipeline: Scaling + Logistic Regression
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression())
])

# Train pipeline
pipeline.fit(X, y)

# Save pipeline to HDD
joblib.dump(pipeline, 'iris_pipeline.joblib')
print("Pipeline saved as iris_pipeline.joblib")
```

Loading and Using a Pipeline

```
# Load pipeline
loaded_pipeline = joblib.load('iris_pipeline.joblib')

# Predict with pipeline
sample_data = [[5.1, 3.5, 1.4, 0.2]]
prediction = loaded_pipeline.predict(sample_data)

print("Predicted class:", prediction)
```

Advantage:
We don't need to scale data manually — the pipeline remembers it.

Real-World Usage

- Web applications (Flask, FastAPI, Django)
- Batch predictions on new datasets
- Sharing models with team members
- Reproducibility in experiments

Industrial Way to handle Machine Learning Project

```
############################################################
# Required Python Packages
############################################################

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    roc_curve,
    auc,
    classification_report,
)

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
import joblib

############################################################
# File Paths
############################################################

INPUT_PATH = "breast-cancer-wisconsin.data"
OUTPUT_PATH = "breast-cancer-wisconsin.csv"
MODEL_PATH = "bc_rf_pipeline.joblib"

############################################################
# Headers
############################################################

HEADERS = [
    "CodeNumber", "ClumpThickness", "UniformityCellSize", "UniformityCellShape",
"MarginalAdhesion",
    "SingleEpithelialCellSize", "BareNuclei", "BlandChromatin", "NormalNucleoli", "Mitoses",
"CancerType"
]
```

```
#############################################################
# Function name :       read_data
# Description :         Read the data into pandas dataframe
# Inpt :                path of CSV file
# Output :              Gives the data
# Author :              Piyush Manohar Khairnar
# Date :                09/08/2025
#############################################################

def read_data(path):
    """Read the data into pandas dataframe"""
    data = pd.read_csv(path, header=None)
    return data

#############################################################
# Function name :       get_headers
# Description :         dataset headers
# Input :               dataset
# Output :              Returns the header
# Author :              Piyush Manohar Khairnar
# Date :                09/08/2025
#############################################################

def get_headers(dataset):
    """Return dataset headers"""
    return dataset.columns.values

#############################################################
# Function name :       add_headers
# Description :         Add the headers to the dataset
# Input :               dataset
# Output :              Updated dataset
# Author :              Piyush Manohar Khairnar
# Date :                09/08/2025
#############################################################

def add_headers(dataset, headers):
    """Add headers to dataset"""
    dataset.columns = headers
    return dataset
```

```
##############################################################
# Function name :       data_file_to_csv
# Input :               Nothing
# Output :              Write the data to CSV
# Author :              Piyush Manohar Khairnar
# Date :                09/08/2025
##############################################################

def data_file_to_csv():
    """Convert raw .data file to CSV with headers"""
    dataset = read_data(INPUT_PATH)
    dataset = add_headers(dataset, HEADERS)
    dataset.to_csv(OUTPUT_PATH, index=False)
    print("File saved ...!")

##############################################################
# Function name :       handel_missing_values
#  Description :        Filter missing values from the dataset
# Input :               Dataset with mising values
# Output :              Dataset by remocing missing values
# Author :              Piyush Manohar Khairnar
# Date :                09/08/2025
##############################################################

def handle_missing_values_with_imputer(df, feature_headers):
    """
    Convert '?' to NaN and let SimpleImputer handle them inside the Pipeline.
    Keep only numeric columns in features.
    """
    # Replace '?' in the whole dataframe
    df = df.replace('?', np.nan)

    # Cast features to numeric
    df[feature_headers] = df[feature_headers].apply(pd.to_numeric, errors='coerce')

    return df
```

```
##############################################################
# Function name :        split_dataset
# Description :          Split the dataset with train_percentage
# Input :                Dataset with related information
# Output :               Dataset after splitting
# Author :               Piyush Manohar Khairnar
# Date :                 09/08/2025
##############################################################

def split_dataset(dataset, train_percentage, feature_headers, target_header,
random_state=42):
    """Split dataset into train/test"""
    train_x, test_x, train_y, test_y = train_test_split(
        dataset[feature_headers], dataset[target_header],
        train_size=train_percentage, random_state=random_state,
stratify=dataset[target_header]
    )

return train_x, test_x, train_y, test_y

##############################################################
# Function name :        dataset_statistics
# Description :          Display the statistics
# Author :               Piyush Manohar Khairnar
# Date :                 09/08/2025
##############################################################

def dataset_statistics(dataset):
    """Print basic stats"""
    print(dataset.describe(include='all'))

##############################################################
# Function name :        build_pipeline
#  Description :         Build a Pipeline:
#  SimpleImputer:        replace missing with median
#                        RandomForestClassifier: robust baseline
# Author :               Piyush Manohar Khairnar
# Date :                 09/08/2025
##############################################################

def build_pipeline():

    pipe = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="median")),
        ("rf", RandomForestClassifier(
            n_estimators=300,
            random_state=42,
            n_jobs=-1,
```

```
        class_weight=None
    ))
])
return pipe

############################################################
# Function name :       train_pipeline
#  Description :        Train a Pipeline:
# Author :              Piyush Manohar Khairnar
# Date :                09/08/2025
############################################################

def train_pipeline(pipeline, X_train, y_train):
    pipeline.fit(X_train, y_train)
    return pipeline

############################################################
# Function name :       save_model
#  Description :        Save the model
# Author :              Piyush Manohar Khairnar
# Date :                09/08/2025
############################################################

def save_model(model, path=MODEL_PATH):
    joblib.dump(model, path)
    print(f"Model saved to {path}")

############################################################
# Function name :       load_model
#  Description :        Load the trained model
# Author :              Piyush Manohar Khairnar
# Date :                09/08/2025
############################################################

def load_model(path=MODEL_PATH):
    model = joblib.load(path)
    print(f"Model loaded from {path}")
    return model
```

```
##############################################################
# Function name :        plot_confusion_matrix_matshow
# Description :          Display Confusion Matrix
# Author :               Piyush Manohar Khairnar
# Date :                 09/08/2025
##############################################################

def plot_confusion_matrix_matshow(y_true, y_pred, title="Confusion Matrix"):
    cm = confusion_matrix(y_true, y_pred)
    fig, ax = plt.subplots()
    cax = ax.matshow(cm)
    fig.colorbar(cax)
    for (i, j), v in np.ndenumerate(cm):
        ax.text(j, i, str(v), ha='center', va='center')
    ax.set_xlabel("Predicted")
    ax.set_ylabel("Actual")
    ax.set_title(title)
    plt.tight_layout()
    plt.show()

##############################################################
# Function name :        plot_feature_importances
# Description :          Display the feture importance
# Author :               Piyush Manohar Khairnar
# Date :                 09/08/2025
##########################################

def plot_feature_importances(model, feature_names, title="Feature Importances (Random Forest)"):
    if hasattr(model, "named_steps") and "rf" in model.named_steps:
        rf = model.named_steps["rf"]
        importances = rf.feature_importances_
    elif hasattr(model, "feature_importances_"):
        importances = model.feature_importances_
    else:
        print("Feature importances not available for this model.")
        return

    idx = np.argsort(importances)[::-1]
    plt.figure(figsize=(8, 4))
    plt.bar(range(len(importances)), importances[idx])
    plt.xticks(range(len(importances)), [feature_names[i] for i in idx], rotation=45, ha='right')
    plt.ylabel("Importance")
    plt.title(title)
    plt.tight_layout()
    plt.show()
```

```
##############################################################
# Function name :        main
# Description :          Main function from where execution starts
# Author :               Piyush Manohar Khairnar
# Date :                 09/08/2025
##############################################################

def main():
    # 1) Ensure CSV exists (run once if needed)
    # data_file_to_csv()

    # 2) Load CSV
    dataset = pd.read_csv(OUTPUT_PATH)

    # 3) Basic stats
    dataset_statistics(dataset)

    # 4) Prepare features/target
    feature_headers = HEADERS[1:-1]   # drop CodeNumber, keep all features
    target_header = HEADERS[-1]       # CancerType (2=benign, 4=malignant)

    # 5) Handle '?' and coerce to numeric; imputation will happen inside Pipeline
    dataset = handle_missing_values_with_imputer(dataset, feature_headers)

    # 6) Split
    train_x, test_x, train_y, test_y = split_dataset(dataset, 0.7, feature_headers, target_header)

    print("Train_x Shape :: ", train_x.shape)
    print("Train_y Shape :: ", train_y.shape)
    print("Test_x Shape :: ", test_x.shape)
    print("Test_y Shape :: ", test_y.shape)

    # 7) Build + Train Pipeline
    pipeline = build_pipeline()
    trained_model = train_pipeline(pipeline, train_x, train_y)
    print("Trained Pipeline :: ", trained_model)

    # 8) Predictions
    predictions = trained_model.predict(test_x)

    # 9) Metrics
    print("Train Accuracy :: ", accuracy_score(train_y, trained_model.predict(train_x)))
    print("Test Accuracy  :: ", accuracy_score(test_y, predictions))
    print("Classification Report:\n", classification_report(test_y, predictions))
    print("Confusion Matrix:\n", confusion_matrix(test_y, predictions))
```

```
    # Feature importances (tree-based)
    plot_feature_importances(trained_model, feature_headers, title="Feature Importances (RF)")

    # 10) Save model (Pipeline) using joblib
    save_model(trained_model, MODEL_PATH)

    # 11) Load model and test a sample
    loaded = load_model(MODEL_PATH)
    sample = test_x.iloc[[0]]
    pred_loaded = loaded.predict(sample)
    print(f"Loaded model prediction for first test sample: {pred_loaded[0]}")

############################################################
# Application starter
############################################################

if __name__ == "__main__":
    main()
```

Breast Cancer Classification using Random Forest & Pipeline

This project demonstrates an **end-to-end machine learning pipeline** for predicting **breast cancer diagnosis** (Benign vs Malignant) using the **Breast Cancer Wisconsin dataset**.

It follows **industrial best practices** by:

- Automating preprocessing with **Pipelines**
- Handling missing values using **`SimpleImputer`**
- Using **Random Forest Classifier** as a robust baseline
- Saving & loading trained models using **Joblib**
- Providing **data visualization** for interpretability

Dependencies

Install the required Python packages before running the project:

```
pip install pandas numpy matplotlib scikit-learn joblib
```

Dataset Information

Source: UCI Machine Learning Repository - Breast Cancer Wisconsin Dataset

Features (10):

1. ClumpThickness
2. UniformityCellSize
3. UniformityCellShape
4. MarginalAdhesion
5. SingleEpithelialCellSize
6. BareNuclei
7. BlandChromatin
8. NormalNucleoli
9. Mitoses

Target:

- **2** = Benign
- **4** = Malignant

Workflow

Data Preparation

- Convert raw `.data` file to `.csv` with **headers**
- Replace **missing values (`?`)** with `NaN`
- Ensure numeric type conversion for all feature columns

Train-Test Split

- Split into **70% train** and **30% test**
- Use **stratified sampling** to preserve class balance

Pipeline Construction

- **Step 1:** `SimpleImputer(strategy="median")` for missing value handling
- **Step 2:** `RandomForestClassifier` with 300 estimators for classification

Model Training & Evaluation

- **Metrics:** Accuracy, Confusion Matrix, Classification Report
- **Feature Importance Plot:** Shows most influential features

Model Saving & Loading

- Save model with **Joblib**
- Load model for future predictions without retraining

Running the Project

Prepare CSV (Only Once)

```
from breast_cancer_pipeline import data_file_to_csv
data_file_to_csv()
```

Train & Evaluate Model

```
python breast_cancer_pipeline.py
```

Expected output:

```
Train Accuracy ::  1.0
Test Accuracy  ::  0.97
Classification Report:
              precision    recall  f1-score   support
...
Confusion Matrix:
[[...]]
Model saved to bc_rf_pipeline.joblib
Loaded model prediction for first test sample: 2
```

Visualizations

- **Feature Importance (Random Forest)**
- **Confusion Matrix** with Matplotlib

Model Storage

- Model is saved as **`bc_rf_pipeline.joblib`**
- Can be loaded anytime for prediction without retraining:

```
from breast_cancer_pipeline import load_model
model = load_model("bc_rf_pipeline.joblib")
```

Sample Prediction

```
sample = test_x.iloc[[0]]
pred = model.predict(sample)
print("Prediction:", pred[0])  # 2 (Benign) or 4 (Malignant)
```

Author

Piyush Manohar Khairnar

Date: 09/08/2025