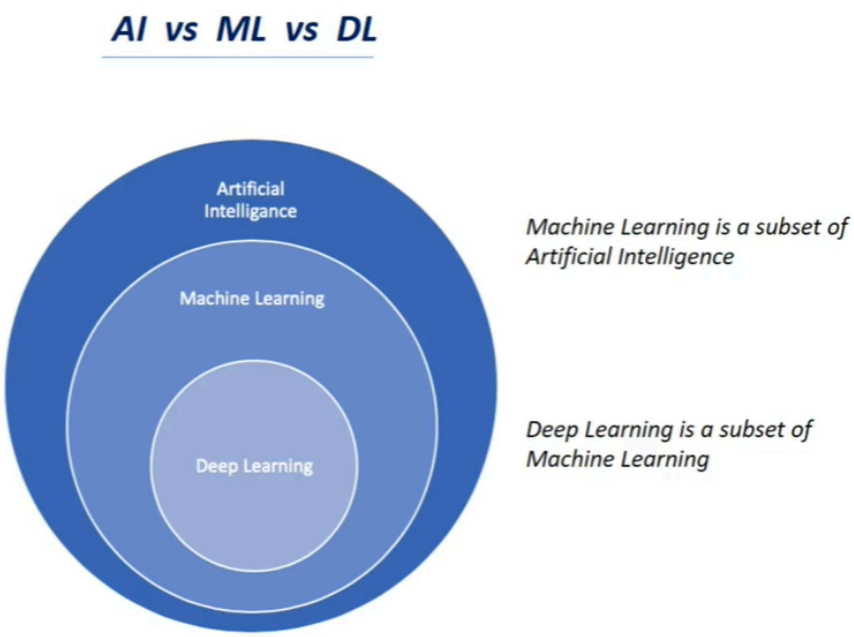
Machine Learning

Module - 1: Machine Learning Basics

Introduction

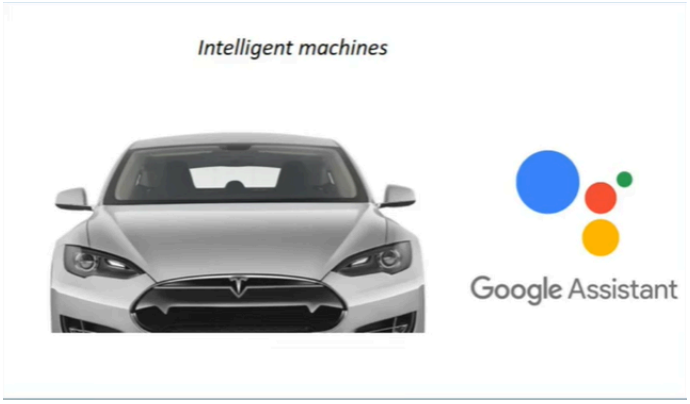
AI vs ML vs DL



ML is a SUBSET of AI and DL is a SUBSET of ML.

What is Artificial Intelligence?

Artificial Intelligence is a branch of Computer Science that is concerned with building smart and intelligent machines.



What is Machine Learning?

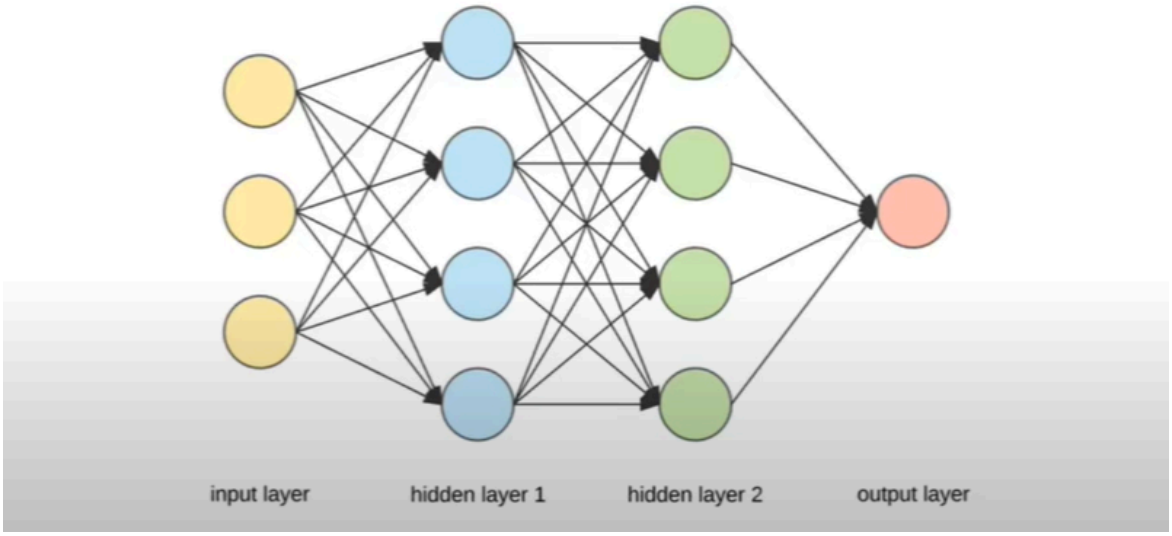
Machine Learning is a technique to implement AI that can learn from data by themselves without being explicitly programmed.

Iron Man

Captain America

What is Deep Learning?

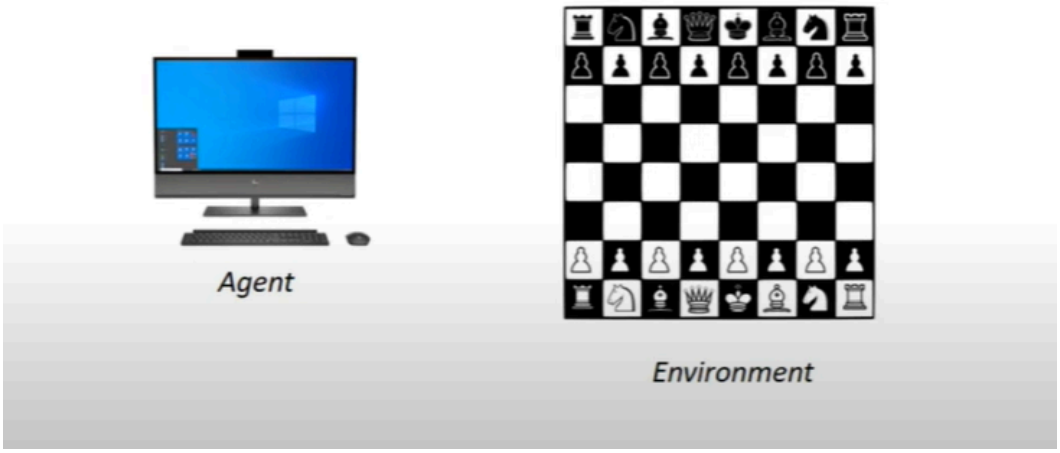
Deep Learning is a subfield of Machine Learning that uses Artificial Neural Networks to learn from the data.



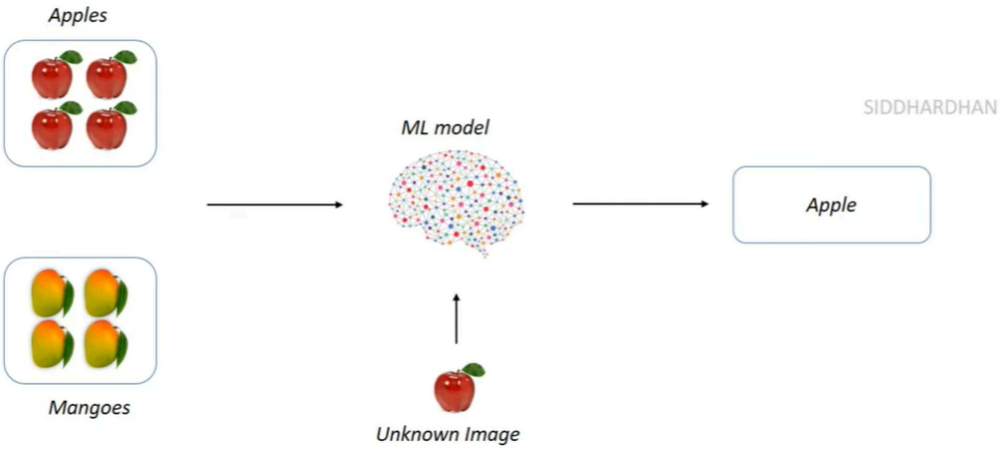
Types of Machine Learning

Machine Learning is a technique to implement AI that can learn from data by themselves without being explicitly programmed.

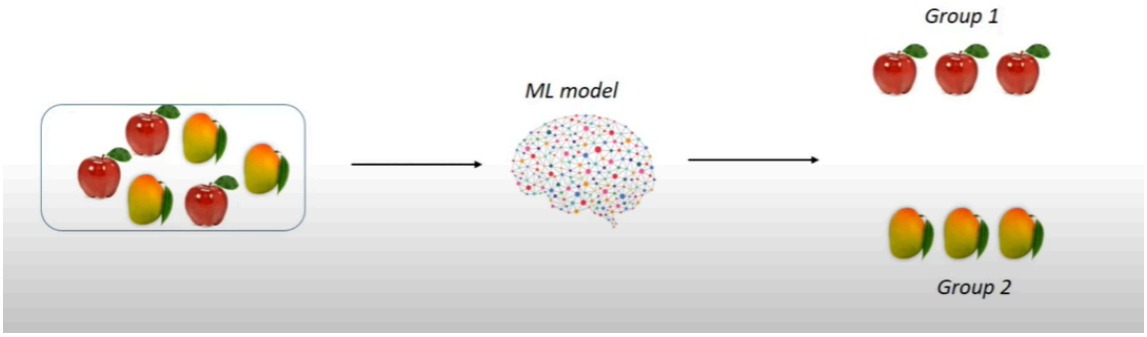
- **Reinforcement** - Reinforcement learning is an area of ML concerned with how intelligent agent takes actions in an environment to maximize their rewards.
    - Aspects of Reinforcement Learning:-
        - Environment
        - Agent
        - Action
        - Reward



- **Supervised** - In Supervised Learning, the ML Algorithm learns from labeled data.



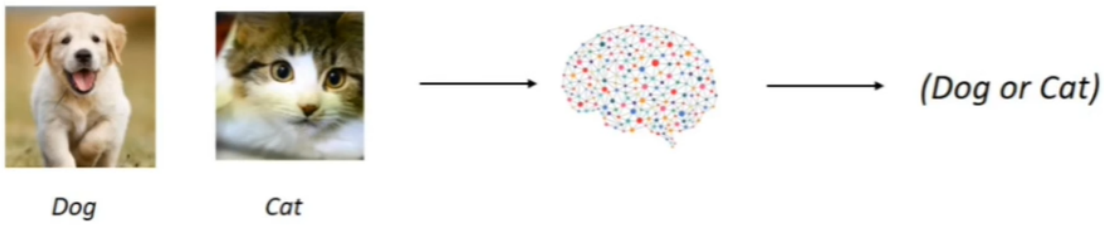
- **Unsupervised** - In Unsupervised Learning, the ML Algorithm learns from unlabeled data.



Supervised Learning & its Types

- In Supervised Learning, the ML Algorithm learns from labeled data.
    - **Types of Supervised Learning:**
        - **Classification** - Classification is about predicting a class or discrete values.

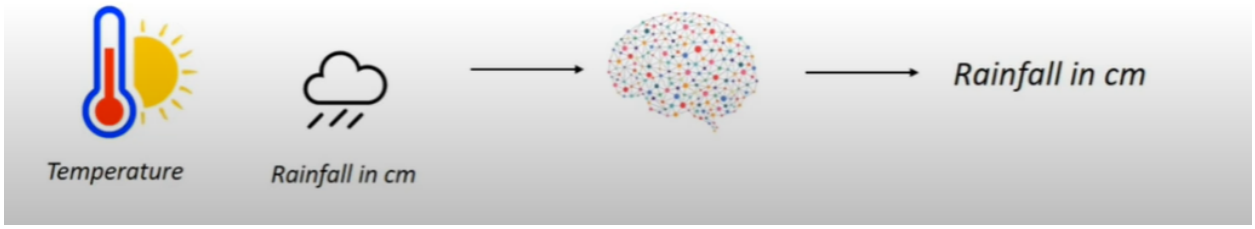
          Ex. - Male or Female, True or False
            - **Classification Algorithms:**
                - **Decision Tree Classification**
                - **Random Forest Classification**
                - **K-nearest Neighbour**



- **Regression** - Regression is about predicting a continuous value or quantity.

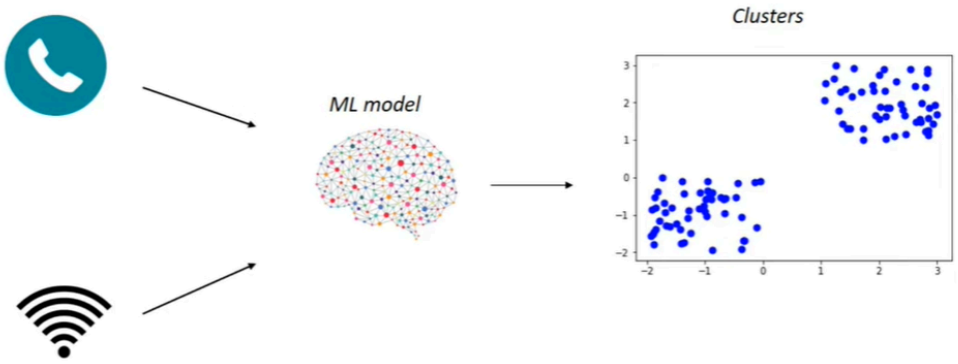
  Ex - Salary, Age, Price

  - **Regression Algorithms:**
    - **Logistic Regression**
    - **Polynomial Regression**
    - **Support Vector Machines**



Unsupervised Learning & its Types

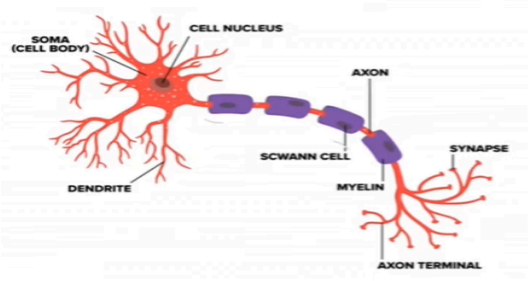
- In Unsupervised Learning, the ML algorithm learns from unlabeled data.
  - **Types Of Unsupervised Learning:**
    - **Clustering -** Clustering is an unsupervised task that involves grouping of similar data points.
      - **Clustering Algorithms:**
        - **K-Means Clustering**
        - **Hierarchical Clustering**

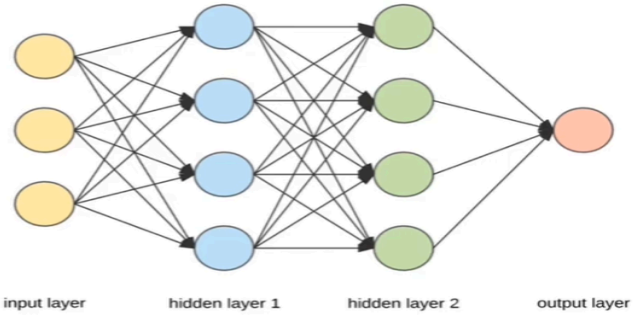


- **Association -** Association is an unsupervised task that is used to find important relationships between data points. Ex. - OTT Platforms, Shopping Sites.
    - **Association Algorithms:**
        - **Apriori**
        - **Eclat**

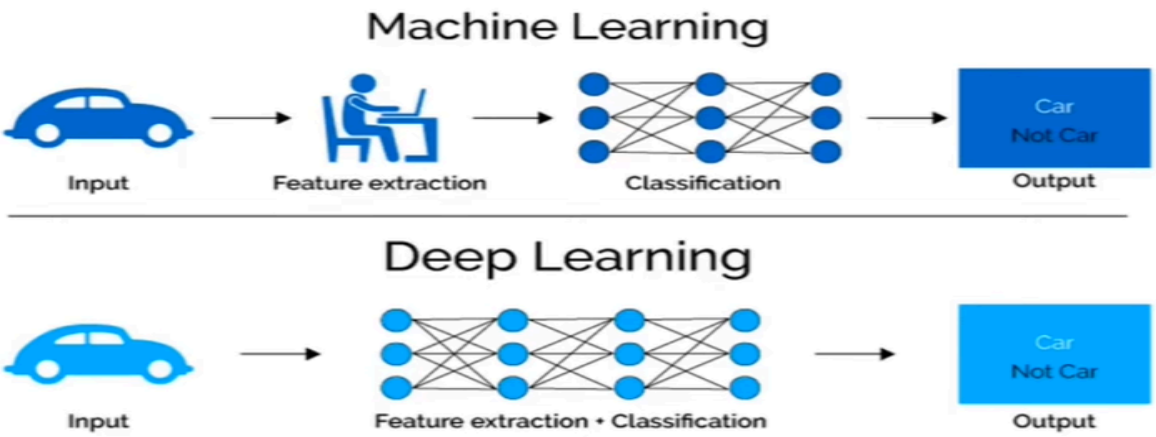
Deep Learning

Deep Learning is a subfield of Machine Learning that uses Artificial Neural Networks to learn from the data.





Machine Learning V/S Deep Learning



How Deep Learning Beacme so Famous?

DeepMind DL Model VS 5 Times World Go Champion Lee Sedol around 2010

DeepMind DL Model won the 5-game tournament by 4-1.

<u>Google Colaboratory - basics</u>

Module - 2: Python Basics for Machine Learning

Basic Data Types in Python:

- **Numeric Type:**
  - **int:** Integer values without decimal points (e.g., 5, -10, 1000).
  - **float:** Floating-point numbers with decimal points (e.g., 3.14, -0.5, 2.0).

- **Boolean Type:**
  - **bool:** Represents boolean values, which can be either True or False.

- **Sequence Type:**
  - **str:** Represents strings of characters (e.g., "hello", 'Python', "123").
  - **tuple:** Ordered and immutable collections of items (e.g., (1, 2, 3), ('apple', 'banana', 'cherry')).
  - **list:** Ordered and mutable collections of items (e.g., [1, 2, 3], ['a', 'b', 'c']).
  - **range:** Represents a range of numbers (e.g., range(0, 10) generates numbers from 0 to 9).

- **Mapping Type:**
  - **dict:** Represents key-value pairs (e.g., {'name': 'Alice', 'age': 30}).

- **Set Type:**
  - **set:** Unordered collections of unique items (e.g., {1, 2, 3}, {'apple', 'orange', 'banana'}).
  - **frozenset:** Immutable version of set (e.g., frozenset({1, 2, 3})).

- **None Type:**
  - **none:** Represents the absence of a value or null.

List, Tuple, Set, Dictionary

- **List:** Lists in Python are ordered collections of items that can hold heterogeneous data types such as integers, floats, strings, and even other lists. Lists are mutable, meaning you can modify, add, or remove elements from them.

```
my_list = [1, 2, 3, 'hello', True]
```

- **Tuple:** A tuple in Python is an ordered collection of elements, similar to a list. However, unlike lists, tuples are immutable, meaning once a tuple is created, its elements cannot be changed, added, or removed. Tuples are typically used to store heterogeneous data types and are useful for representing fixed collections of items.

```
my_tuple = (1, 'hello', 3.14, True)
```

- **Set:** A set in Python is an unordered collection of unique elements. Sets are mutable, meaning you can add or remove elements from them, but they do not allow duplicate elements. Sets are useful for tasks that involve checking membership, performing set operations (such as union, intersection, and difference), and removing duplicates from lists.

```
my_set = {1, 2, 3, 4}
```

- **Dictionary:** A dictionary in Python is an unordered collection of key-value pairs. It is a versatile data structure that allows you to store and retrieve values based on keys. Keys in a dictionary are unique, immutable objects (such as strings, numbers, or tuples), while values can be of any data type, including lists, tuples, sets, dictionaries, and more.

```
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}
```

Operators in Python

- **Arithmetic Operators:**
  - Addition: +
  - Subtraction: -
  - Multiplication: *
  - Division: /
  - Floor Division: // (returns the integer part of the division)
  - Modulus (Remainder): %
  - Exponentiation: **
- **Comparison Operators:**
  - Equal to: ==
  - Not equal to: !=

    - Greater than: >
    - Less than: <
    - Greater than or equal to: >=
    - Less than or equal to: <=
- **Logical Operators:**
    - AND: and (returns True if both conditions are true)
    - OR: or (returns True if at least one condition is true)
    - NOT: not (inverts the result, returns True if the condition is false)
- **Assignment Operators:**
    - Assignment: =
    - Addition assignment: +=
    - Subtraction assignment: -=
    - Multiplication assignment: *=
    - Division assignment: /=
    - Modulus assignment: %=
    - Floor division assignment: //=
    - Exponentiation assignment: **=
- **Identity Operators:**
    - is: is (returns True if two variables refer to the same object)
    - is not: is not (returns True if two variables refer to different objects)

- **Membership Operators:**
    - in: in (returns True if a value exists in the sequence)
    - not in: not in (returns True if a value does not exist in the sequence)
- **Bitwise Operators:**
    - AND: &
    - OR: |
    - XOR: ^
    - NOT: ~

If Else Statement in Python

- If statement
- If-else statement

- If-elif-else statement
- Nested if statements
- Ternary conditional operator (Conditional Expression)
- Short-circuit evaluation

Loops in Python

- for loop
- while loop
- break statement within a loop
- continue statement within a loop
- else clause with loops
- Nested loops
- pass statement within a loop

Functions in Python

- **Types Of Functions:**
    - **Built-in Functions:** These are functions that are built into the Python interpreter and are available for use without needing to define them. Examples include print(), len(), range(), type(), sum(), max(), min(), etc.
    - **User-Defined Functions:** These are functions defined by the user to perform specific tasks. They help in organizing code, improving reusability, and making the code more modular.
    - **Anonymous Functions (Lambda Functions):** Lambda functions are small, anonymous functions defined using the lambda keyword. They are typically used for short, one-liner functions.
    - **Recursive Functions:** Recursive functions are functions that call themselves within their definition. They are useful for solving problems that can be broken down into smaller, similar subproblems.
    - **Higher-Order Functions:** These are functions that take other functions as arguments or return functions as results. They are used extensively in functional programming paradigms.
    - **Generator Functions:** Generator functions use the yield keyword to generate a series of values one at a time, rather than returning them all at once like a normal function. They are memory efficient for generating large sequences of values.

Module - 3: Python Libraries for ML (NumPy, Pandas, Matplotlib, Seaborn)

Numpy

NumPy is a Python library that stands for "Numerical Python." It's used extensively in scientific computing and data analysis because it provides support for arrays, matrices, and mathematical functions to operate on these data structures efficiently. NumPy is often used alongside other libraries like SciPy, Matplotlib, and Pandas to create a powerful environment for data manipulation, analysis, and visualization.

Some of the key features of NumPy include:

- **Arrays**: NumPy provides the ndarray data structure, which is a multidimensional array that allows efficient storage and manipulation of homogeneous data.
- **Mathematical Functions**: NumPy includes a wide range of mathematical functions that operate element-wise on arrays, making it easy to perform complex mathematical operations on large datasets.
- **Broadcasting**: NumPy's broadcasting feature allows operations between arrays of different shapes and sizes, which can greatly simplify code and improve performance.
- **Linear Algebra**: NumPy provides functions for linear algebra operations such as matrix multiplication, eigenvalues, eigenvectors, and solving linear equations.
- **Random Number Generation**: NumPy includes tools for random number generation, which is useful for simulations and statistical applications.

Pandas:

Pandas is a powerful Python library that is widely used for data manipulation and analysis. It provides data structures and functions specifically designed to work with structured and time-series data.

Here are some key aspects of pandas:

- **DataFrame**: The central data structure in pandas is the DataFrame, which is a two-dimensional, labeled data structure with columns of potentially different data types. Think of it as a spreadsheet or SQL table in Python.

- **Series**: A Series is a one-dimensional labeled array capable of holding any data type. It can be seen as a single column within a DataFrame.
- **Data Alignment**: Pandas allows for automatic data alignment based on the labels of the data structures, making it easy to perform operations on data even when they are of different shapes.
- **Data Cleaning**: Pandas provides functions and methods for handling missing data, converting data types, removing duplicates, and other data cleaning tasks.
- **Data Selection and Indexing**: Pandas supports various methods for indexing and selecting data from DataFrames, including label-based indexing (using column names and index labels) and position-based indexing.
- **Grouping and Aggregation**: Pandas enables grouping data based on one or more keys and then applying aggregate functions (like sum, mean, count, etc.) to the grouped data.
- **Time Series Data**: Pandas has robust support for working with time series data, including date/time indexing, resampling, time zone handling, and date range generation.
- **Input/Output**: Pandas can read data from and write data to various file formats such as CSV, Excel, SQL databases, JSON, and more.

Matplotlib

Matplotlib is a widely used Python library for creating static, animated, and interactive visualizations. It provides a high-level interface for generating 2D plots and graphs, making it an essential tool for data visualization in scientific computing, data analysis, and machine learning.

Here are some key features and capabilities of Matplotlib:

- **Plotting Functions:** Matplotlib offers a variety of plotting functions to create line plots, scatter plots, bar plots, histograms, pie charts, box plots, and more.
- **Customization:** Users can extensively customize the appearance of plots, including colors, line styles, markers, labels, titles, legends, axis scales, ticks, and annotations, to create visually appealing and informative visualizations.
- **Subplots and Layouts:** Matplotlib supports creating multiple plots within a single figure using subplots. Users can arrange subplots in various configurations to compare different datasets or aspects of data in one visual.

- **Exporting and Saving:** Matplotlib allows saving plots in various image formats such as PNG, JPEG, PDF, SVG, and more. It also supports interactive plotting in Jupyter Notebooks and exporting plots for inclusion in publications and reports.
- **Mathematical Expressions:** Matplotlib integrates with LaTeX for rendering mathematical expressions and symbols within plot labels, titles, and annotations, providing support for mathematical notation in visualizations.
- **Animations:** Matplotlib includes functionalities for creating animated visualizations, allowing users to animate plots over time or iterate through data to visualize dynamic processes.
- **Interactivity:** Matplotlib can be combined with interactive backends like Qt, Tkinter, or Jupyter widgets to create interactive plots and dashboards, enabling users to interactively explore data and adjust plot parameters.
- **Integration with Pandas:** Matplotlib seamlessly integrates with Pandas DataFrames, allowing users to directly plot data stored in Pandas data structures without extensive data manipulation.

Seaborn

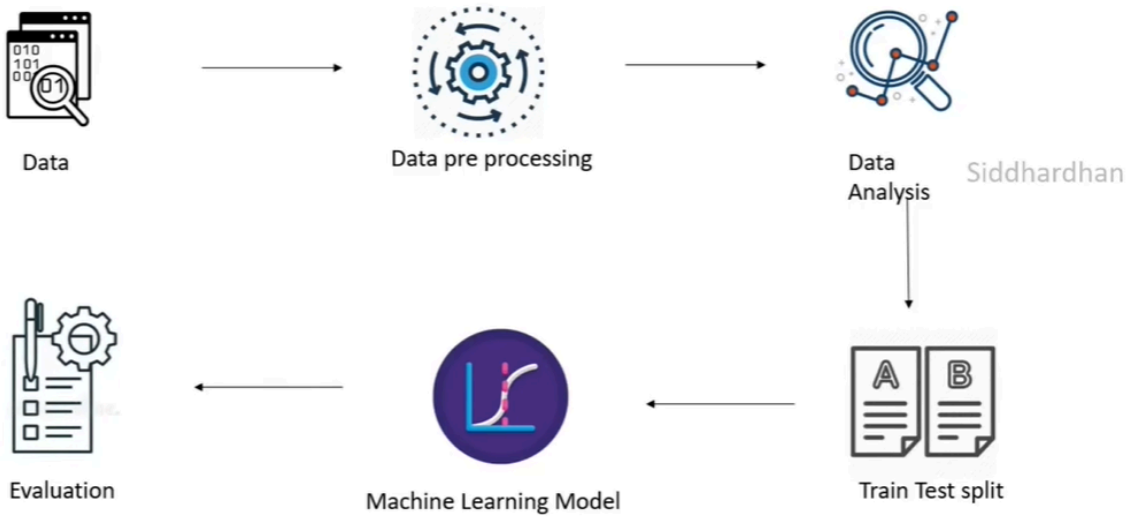
Seaborn is a statistical data visualization library built on top of Matplotlib that provides a high-level interface for creating attractive and informative statistical graphics. While Seaborn works well with pandas DataFrames, it is built to complement Matplotlib rather than pandas directly.

Here are some key features and advantages of using Seaborn with pandas:

- **Statistical Visualization:** Seaborn simplifies the creation of complex statistical visualizations by providing easy-to-use functions for plotting common statistical plots such as bar plots, box plots, violin plots, heatmaps, scatter plots, pair plots, and more.
- **Integration with Pandas:** Seaborn seamlessly integrates with Pandas DataFrames, allowing users to directly pass DataFrame columns as data inputs to Seaborn plotting functions. This integration simplifies data manipulation and plotting tasks.
- **Automatic Styling:** Seaborn comes with built-in themes and color palettes that automatically style plots to make them visually appealing and suitable for publication or presentation. Users can easily customize themes and palettes to match their preferences.
- **Statistical Aggregation:** Seaborn provides functions for statistical aggregation and summarization of data before plotting. For example, the sns.barplot() function can automatically calculate and display the mean or other summary statistics of a numerical variable grouped by a categorical variable.

- **Complex Visualizations:** Seaborn supports creating complex visualizations such as facet grids and pair plots, which allow users to explore relationships between multiple variables and create conditional plots based on subsets of data.
- **Categorical Data:** Seaborn offers specialized functions for visualizing categorical data, including swarm plots, strip plots, and categorical scatter plots, which are useful for analyzing distributions and relationships within categorical variables.
- **Time Series Plotting:** While Seaborn primarily focuses on statistical graphics, it also includes support for time series data visualization, making it suitable for analyzing temporal trends and patterns.
- **Seamless Matplotlib Integration:** Although Seaborn builds on Matplotlib, it simplifies many aspects of creating complex plots by providing higher-level functions and intelligent defaults, allowing users to create sophisticated visualizations with minimal code.

Module - 4: Data Collection and Pre-processing



Data Collection for ML

- **Importance of Data in ML**
    - **Training:** Data teaches machine learning algorithms.
    - **Performance:** Better data means better model performance.
    - **Generalization:** Good data helps models work in new situations.
    - **Features:** Data helps find important patterns (features).
    - **Bias and Fairness:** Data affects fairness and bias in models.

    - **Continuous Improvement:** New data helps models get better over time.
- **From Where to Collect Data?**
    - Kaggle
    - UCI Machine Learning Repository
    - Google Dataset Search
- **Demonstration of Data Collection**

Importing datasets through Kaggle API (Application Program Interface)

Handling Missing Values

- **Imputation**
    - **Central Tendencies**
        - **Mean:** The mean is the average of a set of numbers. To find the mean, you add up all the numbers in the set and then divide by the total count of numbers.
        - **Median:** The median is the middle value in a sorted list of numbers. If there's an even number of values, the median is the average of the two middle numbers.
        - **Mode:** The mode is the number that appears most frequently in a set of numbers. A set of numbers can have one mode (unimodal), two modes (bimodal), or more than two modes (multimodal).
- **Dropping:** Dropping/Deleting the brown which consists of null or empty values.

Data Standardization

sklearn.utils._bunch.Bunch is a specific data structure used in scikit-learn **to represent datasets and similar objects**. It's a class that **behaves like a dictionary** but allows attribute-style access to its elements.

- **Data Structure:** It's a data structure used in scikit-learn.
- **Dictionary-Like:** Allows access to elements using keys like a dictionary.
- **Attribute-Style Access:** Also supports accessing elements using dot notation.

- **Common Usage:** Typically used in scikit-learn to represent datasets and similar objects.

The standard score, also known as the z-score, is a measure used in statistics to standardize and compare data points in a dataset. It tells you how many standard deviations a data point is from the mean of the dataset. The formula for calculating the z-score of a sample x is:

z = (x - u) / s

Where:

- z = z is the standard score (z-score) of the sample x.
- x = x is the individual data point you want to standardize.
- u = u is the mean (average) of the dataset.
- s = s is the standard deviation of the dataset.

StandardScaler in machine learning is a preprocessing technique used to **standardize features** by **removing the mean and scaling them to unit variance**. This preprocessing step is crucial for many machine learning algorithms because it **ensures that different features have the same scale**, preventing certain features from dominating the learning process due to their larger magnitude.

Standardizing data before and after splitting it into training and testing sets:

Aspect	Standardizing Before Splitting	Standardizing After Splitting
Merits		
Information Leakage	Avoids information leakage between sets.	Prevents data leakage during scaling.
Consistent Scaling	Ensures consistent scaling across sets.	Reflects real-world scenario for scaling.
Simplifies Code	Applies scaler once to entire dataset.	Simplifies code for separate sets.
Demerits		
Risk Of Data Leakage	Risk of accidental data leakage.	Eliminates risk of data leakage.
Limited To Batch Processing	Not feasible for very large datasets.	More flexible for large datasets.
Inconsistent Scaling	Scaling factors may vary slightly.	Ensures consistent scaling per set.
Additional Code Complexity	Slightly simpler code.	Requires separate scaling code.

Label Encoding

Label encoding is a process in machine learning used to convert categorical data into numerical format. Categorical data refers to data that represents categories or labels, such as colors (red, blue, green), sizes (small, medium, large), or types (cat, dog, bird).

Label encoding assigns a unique numerical value to each category in a categorical feature, essentially transforming the data into a format that machine learning algorithms can understand and work with.

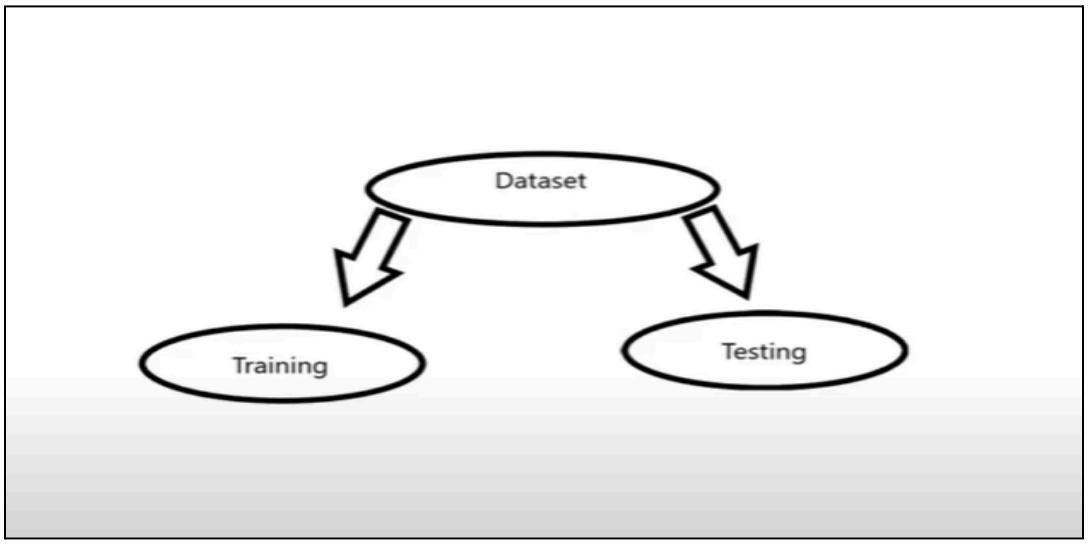
How label encoding typically works:

- **Assigning Numerical Labels:** Each unique category in a categorical feature is assigned a numerical label. For example, if you have a feature "Color" with categories "Red," "Blue," and "Green," label encoding might assign 0 to "Red," 1 to "Blue," and 2 to "Green."
- **Transforming Data:** The categorical feature is transformed into a numerical format using the assigned labels. So, "Red" becomes 0, "Blue" becomes 1, and "Green" becomes 2.
- **Machine Learning Input:** The transformed numerical data can then be used as input to machine learning algorithms for training and prediction.

While label encoding is a straightforward way to convert categorical data to numerical format, it has some considerations:

- **Ordinality:** Label encoding assumes an ordinal relationship between categories, meaning that the numerical labels imply an order or ranking. However, this may not always be appropriate for all categorical features.
- **Impact on Algorithms:** Some machine learning algorithms may misinterpret the numerical labels as ordinal values and treat them accordingly, leading to incorrect results.

Train Test Split



Splitting the data into training data and test data is a crucial step in machine learning model development.

- **Training Data**: This is a subset of the dataset that you use to train your machine-learning model. The model learns patterns, relationships, and features from this data to make predictions or classifications.
- **Test Data:** This is another subset of the dataset that you set aside and do not use during training. After training the model on the training data, you evaluate its performance using the test data. This helps you understand how well the model generalizes to new, unseen data.
- By splitting the data into training and test sets, you can assess the model's performance on data it hasn't seen before. This process helps to detect overfitting (when the model performs well on the training data but poorly on new data) and ensures that the model can make accurate predictions or classifications on real-world data.

Handling Imbalanced Dataset

Imbalanced Dataset: A dataset with an unequal class distribution.

Feature Extraction of Text data

The Mapping from textual data to real-valued vectors is called feature extraction.
Feature extraction in text data for machine learning involves converting text documents into numerical feature vectors that machine learning algorithms can work with.

1. **Tokenization**: Breaking text documents into smaller units called tokens, which could be words, phrases, or characters. This step also involves removing punctuation and special characters.
2. **Lowercasing**: Converting all text to lowercase to ensure consistency in word representation (e.g., "Hello" and "hello" are treated the same).
3. Stop Words Removal: Removing common stop words (e.g., "the," "is," "and") that do not contribute much to the meaning of the text.
4. **Stemming or Lemmatization**: Normalizing words to their root form to reduce variations (e.g., "running" and "ran" to "run").
5. **Vectorization**:
    a. **Bag of Words (BoW):** Represents text as a matrix where rows are documents and columns are unique words (or n-grams). Each cell contains the frequency of the word in the document.
    b. **TF-IDF (Term Frequency-Inverse Document Frequency)**: Assigns weights to words based on their frequency in the document and across the corpus. It emphasizes rare words that are more informative.
        i. **TF(Term Frequency): (**No. of times term **t** appears in a doc**)/(**No. Of terms in the doc**)**
        ii. **IDF(Inverse Document Frequency):** log(N/n),
            - **N** = No. of documents
            - **n** = No. of documents, the term **t** has appeared in
        iii. The IDF Value of a rare word is high whereas the IDF value of a common word is low.
        iv. **TF-IDF Value of a term = TF * IDF**
6. **Word Embeddings (Optional):** Representing words as dense vectors in a continuous space using techniques like Word2Vec, GloVe, or FastText. This captures semantic relationships between words.
7. **Feature Selection (Optional):** Selecting the most relevant features (words or n-grams) based on criteria like frequency, importance, or statistical tests.
8. **Sparse Matrix Representation:** The final feature matrix is typically sparse since most text data contains many zeros (due to the vast vocabulary and sparse word occurrences).

Numerical dataset processing

Common steps for pre-processing numerical data for machine learning:

- **Handling Missing Values:**
    - Identify and handle missing values by imputation (e.g., mean, median, mode) or deletion.
- **Feature Scaling:**
    - Standardize numerical features to have mean 0 and variance 1 (e.g., using StandardScaler).
    - Normalize features to a specified range (e.g., min-max scaling).
- **Outlier Detection and Treatment:**
    - Identify outliers using statistical methods (e.g., IQR, Z-score).
    - Treat outliers by clipping, winsorizing, or transforming them.
- **Encoding Categorical Variables (if applicable):**
    - Convert categorical variables to numerical format using techniques like one-hot encoding or label encoding.
- **Feature Engineering:**
    - Create new features based on domain knowledge or transformations of existing features (e.g., polynomial features).
- **Dimensionality Reduction (if needed):**
    - Reduce dimensionality using techniques like PCA (Principal Component Analysis) or LDA (Linear Discriminant Analysis) to capture essential information.
- **Data Splitting:**
    - Split the pre-processed data into training and testing sets for model training and evaluation.

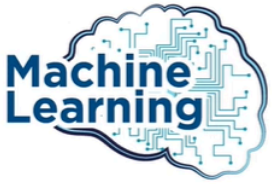
Textual Data Processing

Sure, here's an explanation of each library used in your code:

- **re (Regular Expressions):**
    - **import re:** This library provides functions and methods for working with regular expressions (regex) in Python.
    - Regular expressions are powerful tools for pattern matching and manipulation of strings.
    - Examples of usage include searching for patterns, replacing substrings based on patterns, and splitting strings based on patterns.

- **nltk (Natural Language Toolkit):**
    - **import nltk:** This library is a popular suite of libraries and programs for natural language processing (NLP) tasks in Python.
    - It provides tools for tokenization, stemming, lemmatization, part-of-speech tagging, and more.
    - Examples of usage include text preprocessing, feature extraction, sentiment analysis, and language modeling.
- **nltk.corpus.stopwords (Stopwords):**
    - **from nltk.corpus import stopwords:** This part of NLTK provides a collection of common stop words for various languages.
    - Stop words are words that are considered non-informative and are often removed during text preprocessing to improve text analysis and modeling.
- **nltk.stem.porter.PorterStemmer (Stemming):**
    - **from nltk.stem.porter import PorterStemmer:** This module in NLTK provides the Porter stemming algorithm.
    - Stemming is the process of reducing words to their root or base form, removing suffixes and prefixes.
    - Example: "running" and "ran" would both be stemmed to "run" using the PorterStemmer.
- **sklearn.feature_extraction.text.TfidfVectorizer (TF-IDF Vectorization):**
    - **from sklearn.feature_extraction.text import TfidfVectorizer:** This is a part of scikit-learn (sklearn) library for machine learning in Python.
    - TF-IDF (Term Frequency-Inverse Document Frequency) vectorization is a technique used to convert text documents into numerical feature vectors.
    - It calculates the importance of a word in a document relative to a collection of documents, considering both the frequency of the word in the document and its rarity across the collection.
    - TF-IDF vectorization is commonly used for text classification, clustering, and information retrieval tasks in machine learning.

Machine Learning Models



Data

Machine Learning Models

A machine learning model is a mathematical representation or algorithm that learns patterns and relationships from data. It's like a virtual brain that can make predictions or decisions based on the patterns it has learned. These models are trained using large amounts of data and are used in various applications such as image recognition, natural language processing, predictive analytics, and more.

In machine learning (ML), models are used to predict or understand the relationship between variables in a dataset. These variables are often categorized as dependent and independent variables.

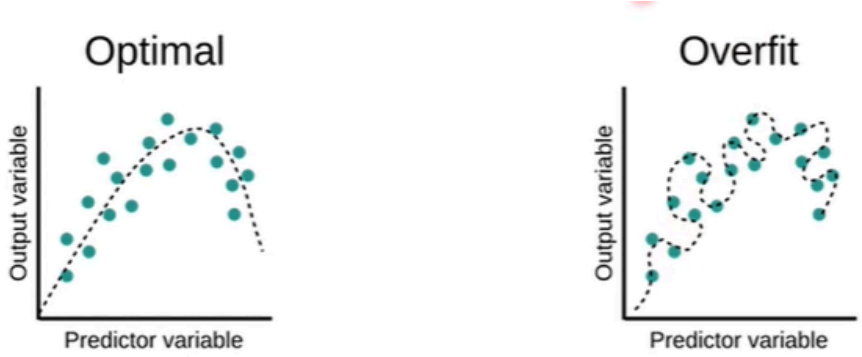
- **Independent Variables:** These are the input variables or features that are used to make predictions or observations.
    - For example, in a housing price prediction model, independent variables could include features like the size of the house, the number of bedrooms, location, etc.
- **Dependent Variables:** Also known as the target variable, this is what the model aims to predict or explain based on the independent variables.
    - In the housing price prediction example, the dependent variable would be the price of the house.

Model Selection

Models can be selected based on :

1. **Type of Data available:**
    a. Images & Videos - CNN
    b. Text data or Speech data - RNN
    c. Numerical data - SVM, Logistic Regression, Decision trees, etc.
2. **2. Based on the task we need to carry out:**
    a. Classification tasks - SVM, Logistic Regression, Decision trees, etc.
    b. Regression tasks - Linear regression, Random Forest, Polynomial regression, etc.
    c. Clustering tasks - K-Means Clustering, Hierarchical Clustering

Overfillting



Overfitting refers to a model that models the training data too well. Overfitting happens when a model learns the detail and noise in the training dataset to the extent that it negatively impacts the performance of the model.

Sign that the model has Overfitted: High Training data Accuracy & very low Test data Accuracy

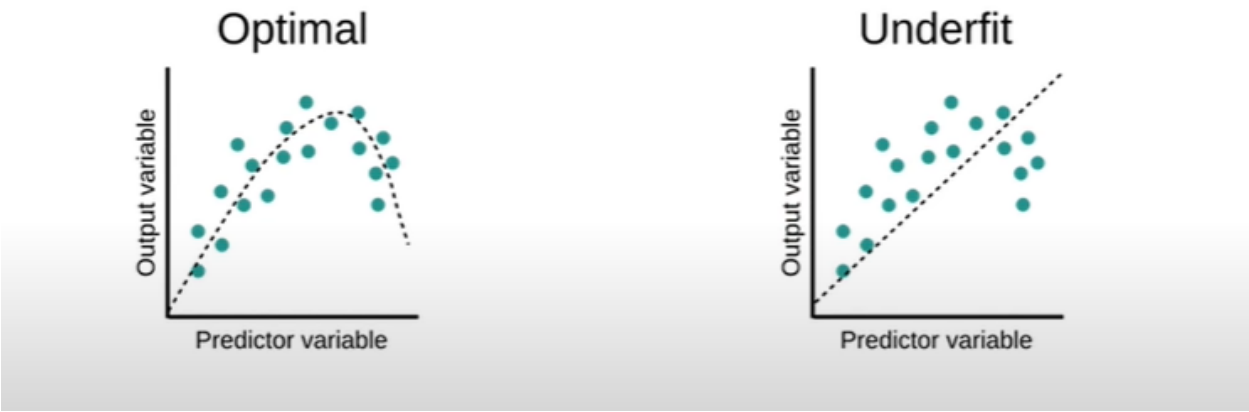
Causes for Overfitting:

- Less Data
- Increased Complexity of the model
- More number of layers in Neural Network

Preventing Overfitting by:

- Using more data
- Reducing the number of layers in the Neural network
- Early Stopping
- Bias - Variance Tradeoff

Underfitting



Underfitting happens when the model does not learn enough from the data. Underfitting occurs when a machine-learning model cannot capture the underlying trend of the data.

Causes for Underfitting:

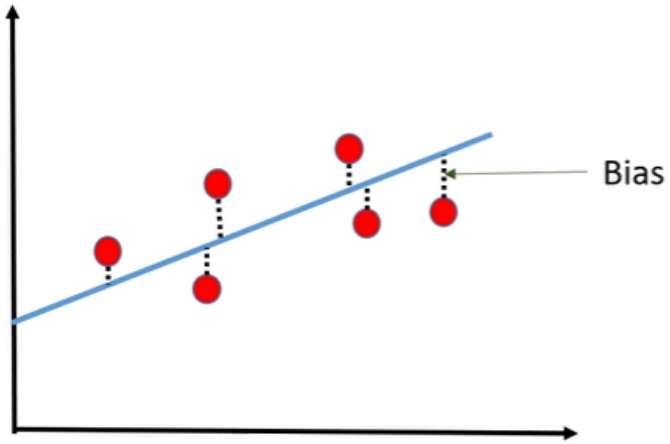
- Choosing the wrong model
- Less complexity of the model
- Less variance but high bias

Prevent Underfitting by:

- Choosing the correct model appropriate for the problem
- Increasing the complexity of the model
- More number of parameters to the model
- Bias - Variance Tradeoff

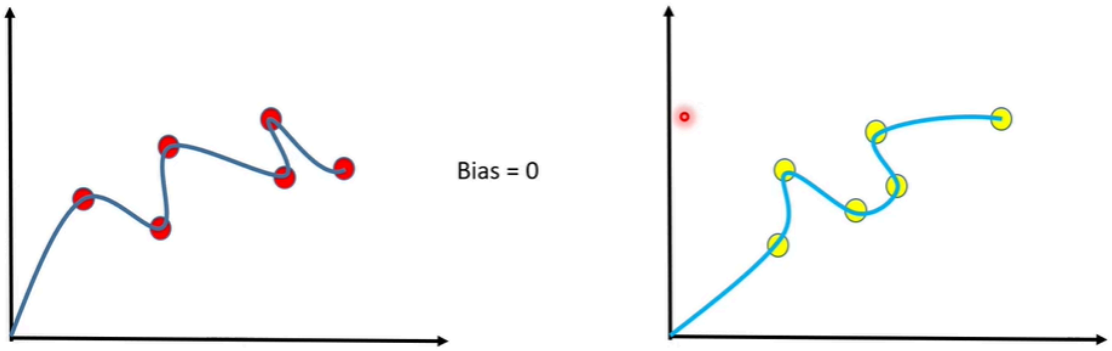
Bias-Variance Tradeoff

Bias - Bias is the difference between the average prediction of our model and the correct value which we are trying to predict.



- **High Bias (Underfitting):** This occurs when a model is too simple to capture the underlying patterns in the data. It often leads to high errors in both the training and testing data.
- **How to Address:** To reduce bias and prevent underfitting, use more complex models, increase the number of features, or use more sophisticated algorithms.

Variance - Variance is the amount that the estimate of change if different training data was used.



- **High Variance (Overfitting):** Occurs when a model is overly complex and fits too closely to the training data, capturing noise and random fluctuations as meaningful patterns. This leads to low errors on the training data but high errors on new, unseen data.
- **How to Address:** To reduce variance and prevent overfitting, use techniques such as regularization, feature selection, cross-validation, and early stopping.

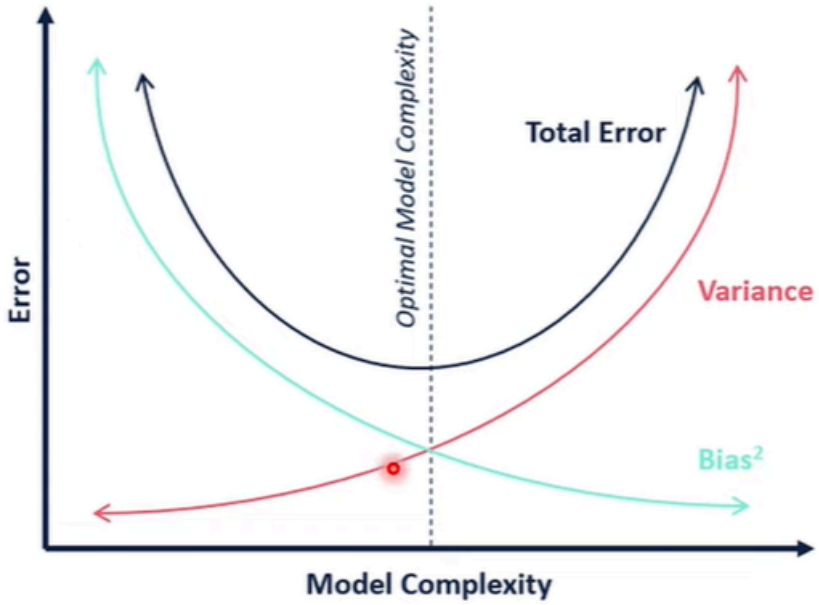
Bias-Variance Tradeoff:

- **Increasing Model Complexity:** Increasing model complexity reduces bias but increases variance. This means the model can capture more complex patterns but may also overfit the training data.
- **Decreasing Model Complexity:** Decreasing model complexity reduces variance but increases bias. This means the model is less likely to overfit but may underfit by oversimplifying the data.

Preventing Overfitting and Underfitting:

- **Regularization:** Use techniques like L1 or L2 regularization to penalize complex models, reducing variance and preventing overfitting.
- **Feature Selection:** Choose only the most relevant features to reduce model complexity and prevent overfitting.
- **Cross-Validation:** Use k-fold cross-validation to evaluate model performance on multiple subsets of the data, helping to identify and prevent overfitting.
- **Early Stopping:** Stop training the model when performance on a validation set starts to degrade, preventing overfitting.
- **Ensemble Methods:** Use ensemble methods like random forests or gradient boosting to combine multiple models, reducing overfitting and improving generalization.

Bias – Variance Tradeoff



Machine Learning Algorithms

Linear Regression

Linear regression is a fundamental machine learning technique used for modeling the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the input variables and the output variable. The goal of linear regression is to find the best-fitting linear equation that predicts the dependent variable based on the independent variables.

The general form of a linear regression model with one independent variable can be written as

y = b0 + b1 x + e

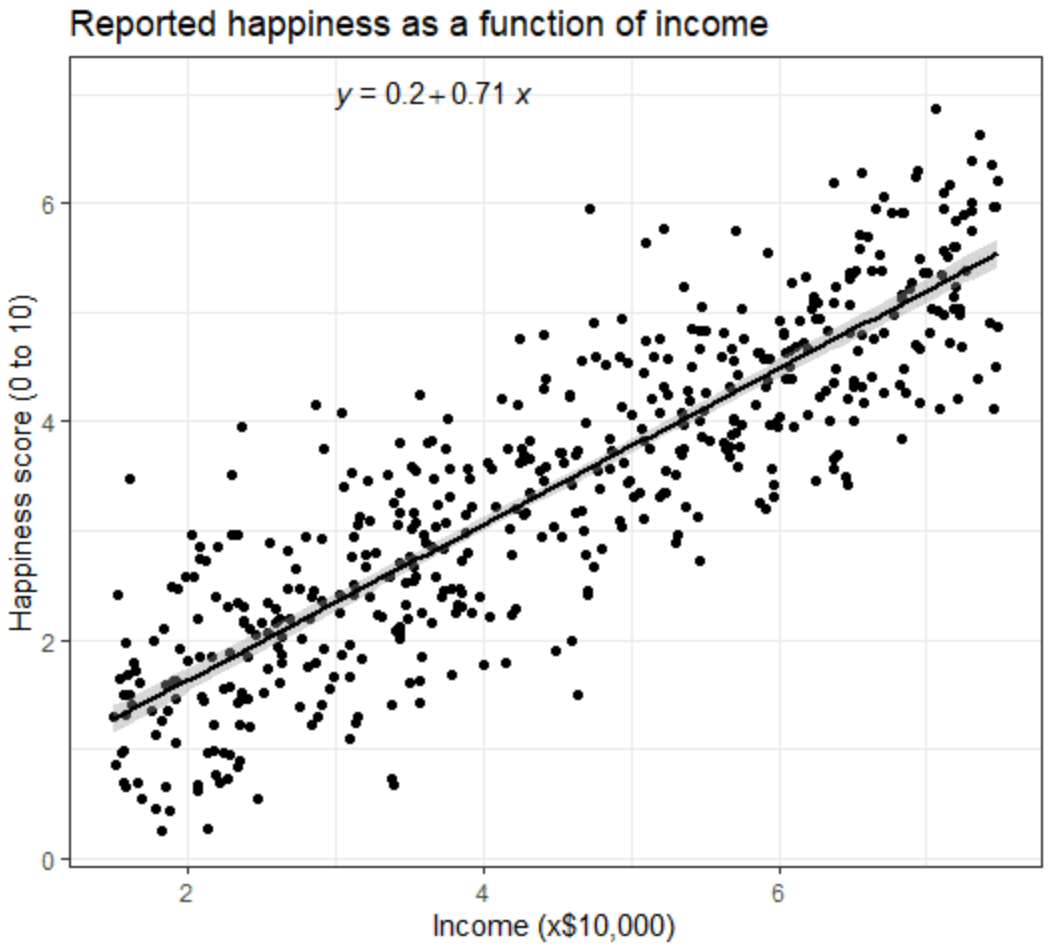
Where:

- **y** is the dependent variable (the variable we want to predict).
- **x** is the independent variable (the input feature).
- **b0** is the intercept (the value of **y** when **x** is zero).
- **b1** is the slope (the change in **y** for a one-unit change in **x**.
- **e** is the error term (the difference between the predicted **y** and the actual **y**.

The linear regression model finds the values of **b0** and **b1** that minimize the sum of squared errors between the predicted values and the actual values in the training data. This is typically done using techniques like **Ordinary Least Squares (OLS)** or **gradient descent.**

Linear regression is commonly used for tasks such as:

- Predicting house prices based on features like size, number of bedrooms, etc.
- Predicting sales based on advertising spending.
- Forecasting stock prices based on historical data.



Logistic Regression

Logistic regression is a classification algorithm in machine learning used to predict the probability of a binary outcome (e.g., yes/no, 1/0, true/false). Despite its name, logistic regression is primarily used for classification tasks rather than regression tasks.

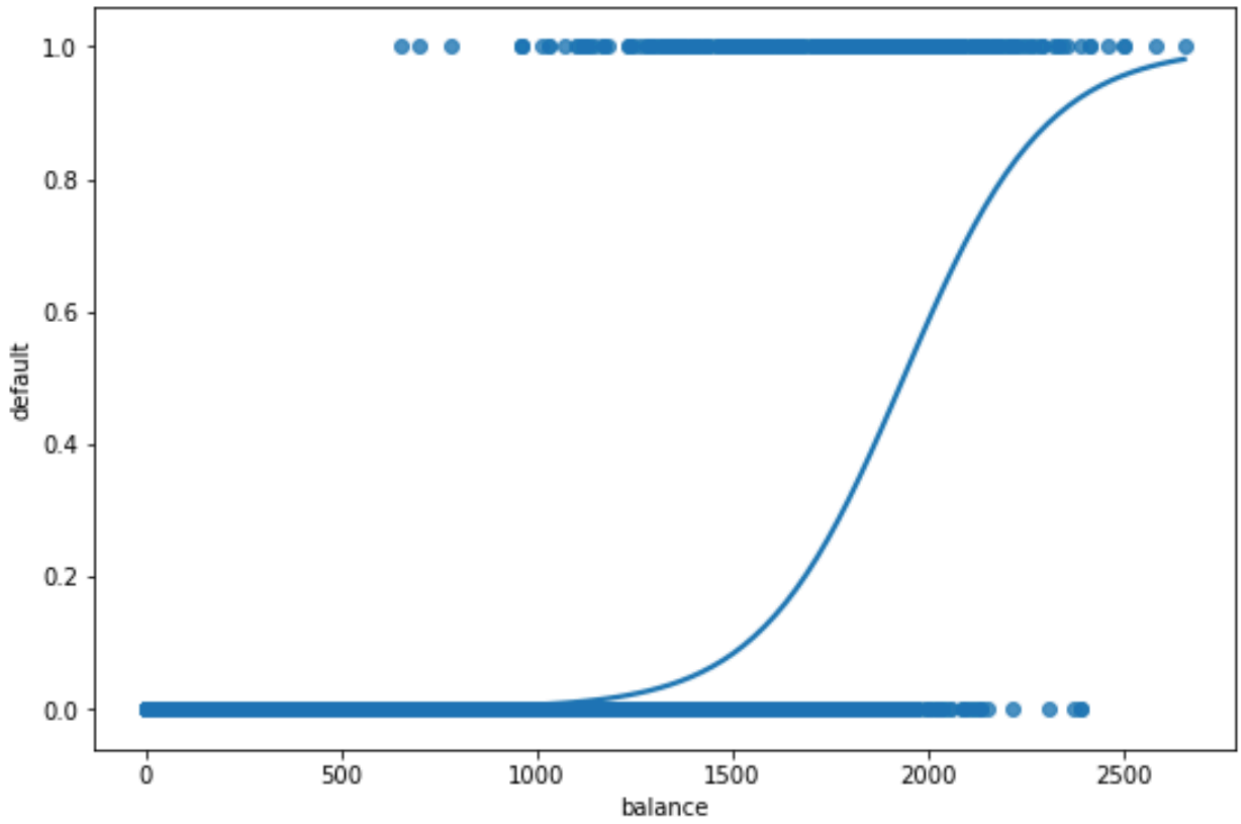
Key points about logistic regression:

- **Binary Classification:** Logistic regression is well-suited for binary classification problems where the target variable has two possible classes.
- **Sigmoid Function:** In logistic regression, the output is transformed using the sigmoid function (also known as the logistic function), which maps any real-valued number to the range [0, 1]. The sigmoid function is defined as:
    - σ(z)= 1/{1+e^(-z)}

    - where z is the linear combination of input features and model coefficients.
- **Linear Decision Boundary:** Logistic regression models a linear decision boundary between the classes in feature space. The decision boundary is determined by the coefficients learned during training.
- **Probabilistic Interpretation:** Logistic regression predicts the probability that an instance belongs to a particular class. If the predicted probability is greater than a threshold (usually 0.5), the instance is classified into one class; otherwise, it's classified into the other class.
- **Loss Function:** The logistic regression model is trained by minimizing a loss function, typically the logistic loss (also known as cross-entropy loss), which measures the difference between the predicted probabilities and the actual class labels.
- **Regularization:** Similar to linear regression, logistic regression can also use regularization techniques (e.g., L1 or L2 regularization) to prevent overfitting and improve generalization.

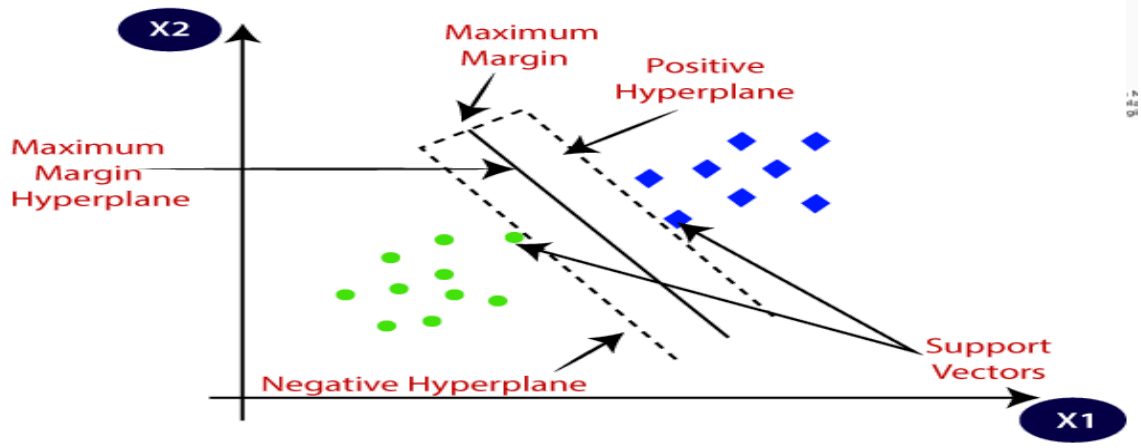
Logistic regression is widely used in various applications, including:

- Spam email detection (classifying emails as spam or not spam)
- Medical diagnosis (predicting whether a patient has a disease or not)
- Credit risk assessment (predicting the likelihood of default on a loan)
- Customer churn prediction (predicting whether a customer will leave a service or not)



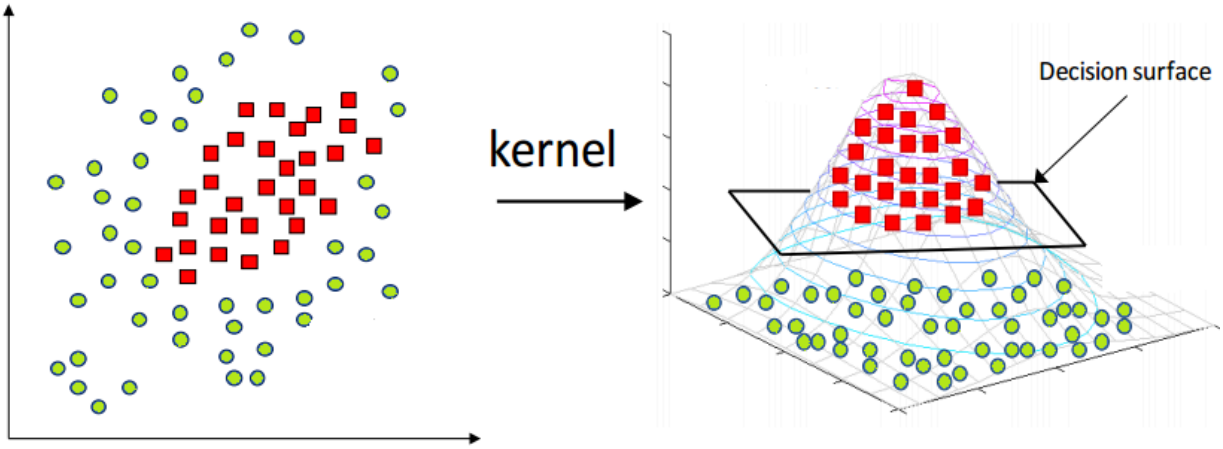
Support Vector Machine

A Support Vector Machine (SVM) is a powerful supervised learning algorithm used for both classification and regression tasks in machine learning. SVMs are particularly effective in high-dimensional spaces and are widely used for tasks such as the classification of images, text, and biological data.



Key points about Support Vector Machines (SVMs):

- **Binary Classification:** SVMs are primarily used for binary classification, where the algorithm classifies data points into two classes based on their features.
- **Margin Maximization:** The goal of SVM is to find the optimal hyperplane that separates the data into classes while maximizing the margin, which is the distance between the hyperplane and the nearest data points (support vectors). This maximization of the margin helps in achieving better generalization and robustness of the model.
- **Kernel Trick:** SVMs can handle linearly separable data using a linear kernel, but they can also handle non-linearly separable data by using kernel functions such as polynomial, radial basis function (RBF), and sigmoid. These kernel functions map the input data into higher-dimensional spaces where the data may become linearly separable.
- **Support Vectors:** Support vectors are the data points that lie closest to the decision boundary (hyperplane). These points play a crucial role in defining the optimal hyperplane and determining the margin.
- **Regularization:** SVMs use regularization parameters (C parameter in scikit-learn's SVM implementation) to control the trade-off between achieving a larger margin and minimizing classification errors. A smaller C value encourages a larger margin (more regularization), while a larger C value allows more classification errors (less regularization).
- **Sensitivity to Outliers:** SVMs are sensitive to outliers as they can affect the position of the hyperplane and the margin. Proper preprocessing and outlier handling techniques are often necessary for robust SVM models.
- **Kernel Selection:** The choice of kernel function can significantly impact the performance of an SVM model. It's important to experiment with different kernel functions and tune hyperparameters for optimal results.



XGBoost Regressor

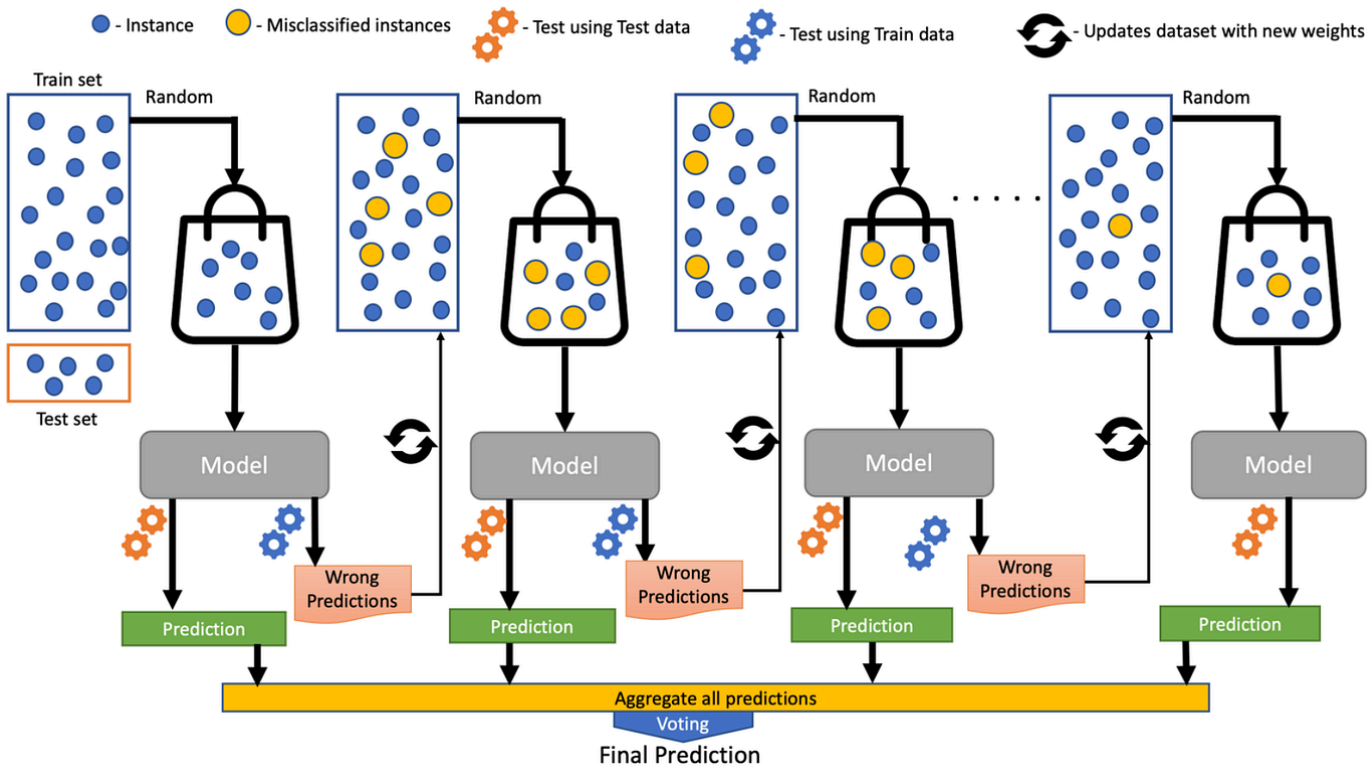
XGBoost (Extreme Gradient Boosting) is a powerful and widely used machine learning algorithm that belongs to the ensemble learning family. Specifically, XGBoost is an implementation of gradient boosting machines, which is a technique for building predictive models by combining multiple weak learners (typically decision trees) into a strong learner. Key points about the XGBoost Regressor model:

- **Gradient Boosting:** XGBoost uses gradient boosting to iteratively train weak learners (decision trees by default) in a sequential manner. Each new tree corrects errors made by the previous ones, leading to a strong ensemble model.
- **Objective Function:** XGBoost minimizes a specified loss function during training. Common loss functions for regression tasks include mean squared error (MSE) and mean absolute error (MAE).
- **Regularization:** XGBoost incorporates regularization techniques to prevent overfitting. It includes L1 (Lasso) and L2 (Ridge) regularization terms in the objective function to penalize complexity.
- **Tree Pruning:** XGBoost performs tree pruning during training to remove splits that do not contribute significantly to improving the model's performance. This helps in reducing overfitting and improving generalization.
- **Feature Importance:** XGBoost provides feature importance scores, indicating the contribution of each feature in the model's predictions. This can be useful for feature selection and understanding the model's behavior.

- **Parallel Processing:** XGBoost is designed for efficiency and supports parallel processing, making it scalable and capable of handling large datasets.
- **Handling Missing Values:** XGBoost has built-in capabilities to handle missing values in the dataset, reducing the need for preprocessing.
- **Hyperparameter Tuning:** XGBoost offers a wide range of hyperparameters that can be tuned to optimize model performance, including tree-specific parameters, regularization parameters, and learning rate.

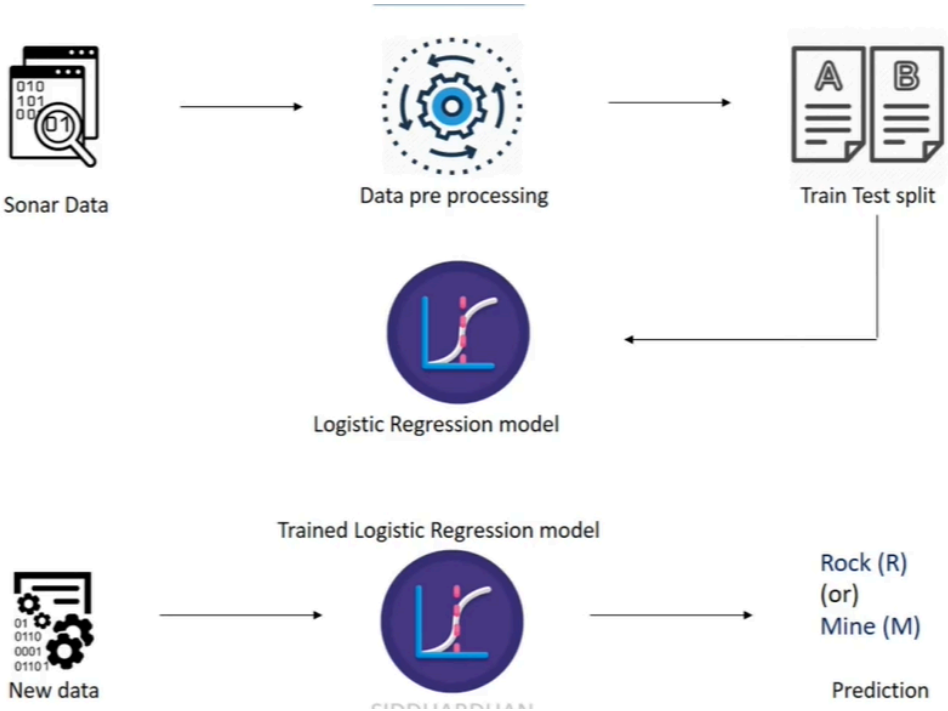
XGBoost is widely used in various regression tasks, including:

- Predicting house prices based on features like size, location, etc.
- Forecasting sales or demand for products/services.
- Predicting stock prices or financial indicators.
- Regression tasks in data science competitions like Kaggle.



ML Use Cases / Projects

ML Use Case 1: Rock vs Mine Prediction → Logistic Regression → Supervised Learning



ML Use Case 2: Diabetes Prediction → Support Vector Machine → Supervised Learning

ML Use Case 3: Spam Mail Prediction → Logistic Regression → Supervised Learning

ML Use Case 4: Loan Status Prediction → Support Vector Machine → Supervised Learning

ML Use Case 5: Heart Disease Prediction → Logistic Regression → Supervised Learning

ML Use Case 6: House Price Prediction → XGBoost Regressor → Supervised Learning