

# Introduction to Machine Learning

**CX016-2.5-3-IML**

**Week 1**

Introduction and Module Overview

# Lecturer Information

- Lecturer Name: Mafas Raheem
- Email: raheem@apu.edu.my
- Consultation Hours: Refer to iConsult



## Experience:

3 years in the industry  
16 years academic

## Academic Background:

- ❖ PhD in IT (Reading)
  - ✓ AI
- ❖ MSc
  - ✓ Data Science & Business Analytics
- ❖ MBA
  - ✓ Business Administration
- ❖ BSc
  - ✓ Computer Science

## Prime Research Areas (but not limited to):

- Machine Learning, Deep Learning,
- Data Mining, Data Analytics, Business Analytics
- Text Analytics, Sentiment Analysis
- E-Commerce, Social Content Marketing, Social Media Analytics
- Big Data

# Pre-Requisites For This Module

- Nil.
- A knowledge in python programming language would be a definite plus.

# Outcomes Based Education

- OBE is education based on producing particular educational outcomes that:
  - Focus on what students can actually do after they are taught.
  - Expect all learners / students to successfully achieve particular (sometimes minimum) level of knowledge and abilities.
- It's NOT what We want to teach.
- It's WHAT You should learn.

# Aims of this Module

- This course aims to provide participants with essential skills in machine learning using Python.
- The syllabus focuses on foundational concepts, supervised learning (KNN, Linear Regression), unsupervised learning (K-Means), metrics evaluation, training processes, and cross-validation.
- Practical, hands-on sessions will empower participants to apply these concepts to real-world datasets.

# Module Learning Outcomes

CLO	Learning Outcomes	Assessment
1	Analyze the supervised and unsupervised learning techniques for a given field of study (C4, PLO2)	Examination
2	Demonstrate solutions obtained by applying appropriate machine learning models for various types of problems (A3, PLO6)	Assignment
3	Evaluate the performance of the proposed machine learning models (C5, PLO7)	Class Test

# Mapping of CLO with PLO

	PLO 1	PLO 2	PLO 3	PLO 4	PLO 5	PLO 6	PLO 7	PLO 8	PLO 9	PLO 10	PLO 11	PLO12
CLO1		✓										
CLO2						✓						
CLO3							✓					

The learning domains are:

**PLO1:** Knowledge and Understanding

**PLO2:** Cognitive Skills

**PLO3:** Practical Skills

# Student Learning Time

- Module Credit Value: 3
- Total Learning Hours: 56 per semester

Lecture	24 hours
Tutorial	32 hours
Independent Learning Time	46 hours
Assessment	18 hours
Total Learning Hours	120 hours per semester

# Module Content Outline

Week	Topic
1-2	Introduction to Machine Learning
3	Introduction to Python for Data Science
4	Introduction to Supervised Learning
5-6	K-Nearest Neighbors (KNN)
7-8	Linear regression
9	Introduction to Unsupervised Learning
10	K-Means Clustering (KMeans)
11	Introduction to hierarchical clustering
12	Metrics evaluation
13	Training process
14	Cross-validation

# Assessment Summary

(refer to module handbook and module descriptor)

Form of Assessment	Assessment Methods	Hand Out Date	Hand In Date	%
Continuous Assessment	Assignment	3 <sup>rd</sup> Week	12 <sup>th</sup> Week	30
	Class Test	8 <sup>th</sup> Week	8 <sup>th</sup> Week	20
Final Assessment	Final Examination			50

**Assessment requirement:** Include any specific requirement to pass the module (refer to module handbook for the information), such as:

- To pass the module, you must attempt every element of assessment and achieve at least 50% in the module overall.

# Expectations

1. Abide by ALL rules and regulations of APU.
2. Proper attire.
3. No speaking of dialects.
4. Attendance is compulsory. Valid Medical Certs must be supported in any absence from class.
5. Three cases of Late will be equal to 1 absence.
6. Use proper academic references – APA Referencing only.
7. Academic Dishonesty / Plagiarism is a serious offence. Any suspicions will be referred to the University's Academic Dishonesty Board.
8. Formal assessments must be submitted on time in the specified format given. Failure to meet deadlines will be treated as non-submission and no marks will be awarded. Incomplete submissions will be subjected to penalty of mark deductions or forfeit.

# Other Expectations

- State your expectation of what students need to do or deliver in class, as well as what they need to do out-of-class.
- If group projects are involved, clearly state what is to be expected from each individual (and not be dependent on the group leader)

# Achievement Requirements:

## Undergraduate (Diploma, Foundation, Degree)

Marks	Alphabetical Grade	Grading Point	Classification
80-100	A+	4.0	Distinction
75-79	A	3.7	
70-74	B+	3.3	Credit
65-69	B	3.0	
60-64	C+	2.7	Pass
55-59	C	2.3	
50-54	C-	2.0	
40-49	D	1.7	Fail (marginal)
30-39	F+	1.3	Fail
20-29	F	1.0	Fail
0-19	F-	0	Fail

# Reference Materials

## Course Materials available in Moodle

- Module handbook
- Module descriptor
- Lecture slides
- Tutorial/Lab materials
- Sample incourse questions & answers
- Sample exam questions & answers

## Essential and Further Readings

- Brett, L. (2019) Machine Learning with R : Expert techniques for predictive modeling, 3rd Edition. Packt Publishing Limited. ISBN-13: 9781788295864
- Campesato, O. (2020) Python 3 for machine learning. Mercury Learning & Information. ISBN-13: 9781683924937
- Gopal, M. (2019) Applied machine learning. McGraw-Hill Education. ISBN-13: 9781260456851
- Batchelor, A. (2020) Statistics in social work : An introduction to practical applications. Columbia University Press. ISBN-13: 9780231550222

\*Further readings will be assigned from time to time.

# Your Valuable Feedback

- You are welcome to discuss your views on this module at any point of time.
- Do fill in anonymous evaluation questionnaires in the student feedback form. There are two points - mid and end of the teaching semester.
- The feedbacks you provide will be constructive for improvement of teaching and module content development.

CX016-2.5-3-IML & Introduction to Machine Learning

## **Introduction to Machine Learning**

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understand Core Concepts and Terminologies.
2. Understand Supervised, unsupervised and reinforcement learning.
3. Identify different machine learning applications.

# Contents & Structure

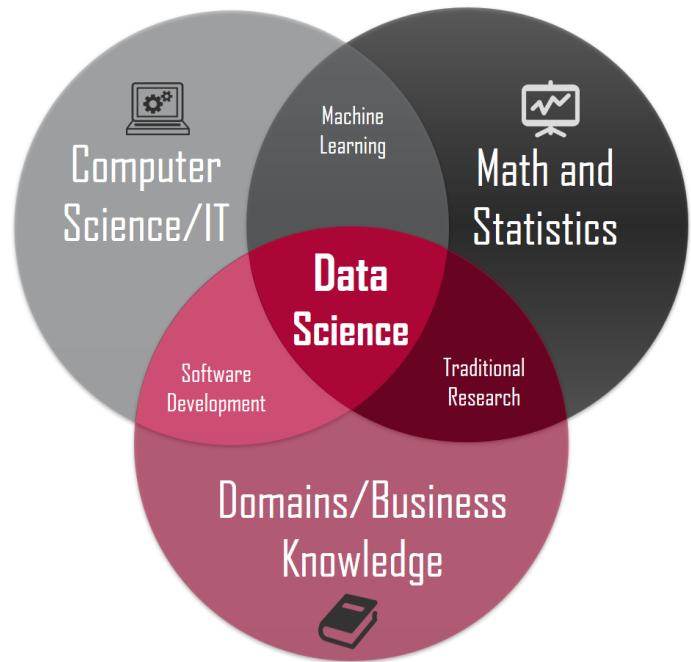
- Data Science
- Data Analytics
- Machine Learning
- Machine Learning - Methods & Applications

# Recap From Last Lesson

- Introduction

# Data Science

- Data science involves utilizing automated techniques to analyze vast amounts of data and extract valuable knowledge from it.
- Data science can transform the extensive data generated in the digital age into new insights and understanding.



# Data Analysis

- **Data Analysis** stands for human activities aimed at gaining some insight on a dataset.
- An analyst can use some Data Analytics tools to obtain desired results, but in principle, Data Analysis can be performed without special data processing.
- For example, a Forex trader can rely on his/her experience to open or close a trading position.

# Data Analytics

- **Data Analytics** is all about automating insights into a dataset and supposes the usage of queries and data aggregation procedures.
- It can represent various dependencies between input variables, but also it can use Data Mining techniques and tools to discover hidden patterns in the dataset under analysis.
- For example, not obvious associations between user purchases can be automatically discovered.

# Data Analytics

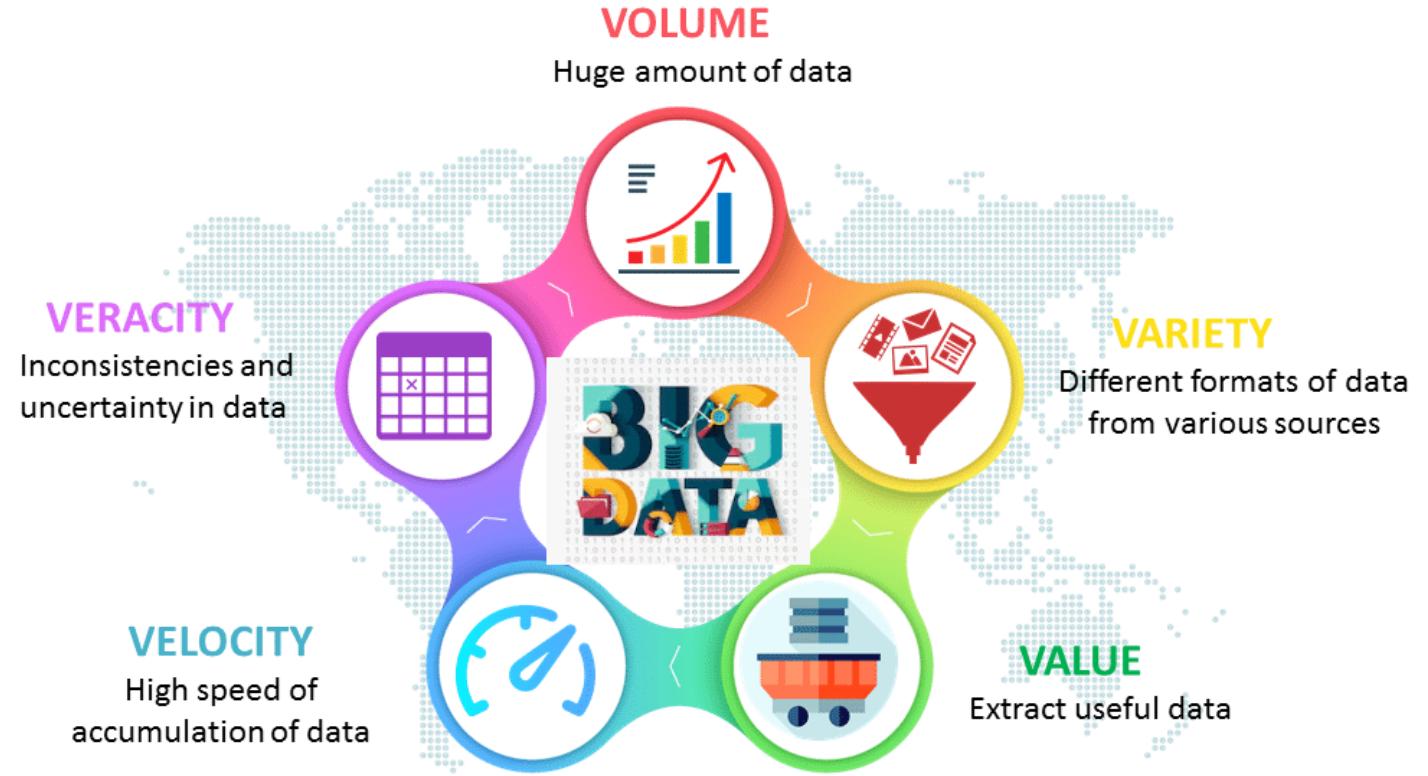
- **Big Data** refers to massive volumes of data that traditional applications cannot efficiently handle. Processing Big Data starts with raw, unaggregated data, which often cannot be stored in the memory of a single computer.
- **Machine learning (ML)** is an artificial intelligence technique widely used in data mining. ML utilizes a training dataset to create a model capable of predicting target variable values, such as future sales. Data mining leverages the predictive power of machine learning by applying various ML algorithms to Big Data.



# How Big is Your Data?

- Kilobyte (1000 bytes)
- Megabyte (1 000 000 bytes)
- Gigabyte (1 000 000 000 bytes)
- Terabyte (1 000 000 000 000 bytes)
- Petabyte (1 000 000 000 000 000 bytes)
- Exabyte (1 000 000 000 000 000 000 bytes)
- Zettabyte (1 000 000 000 000 000 000 000 bytes)
- Yottabyte (1 000 000 000 000 000 000 000 000 bytes)

# 5 Vs of Big Data



# Data Science

## Data Science

Field that determines the processes, systems, and tools needed to transform data into insights to be applied to various industries.

Skills needed:

- Statistics
- Data visualization
- Coding skills (Python/R)
- Machine learning
- SQL/NoSQL
- Data wrangling

## Machine Learning

Field of artificial intelligence (AI) that gives machines the human-like capability to learn and adapt through statistical models and algorithms.

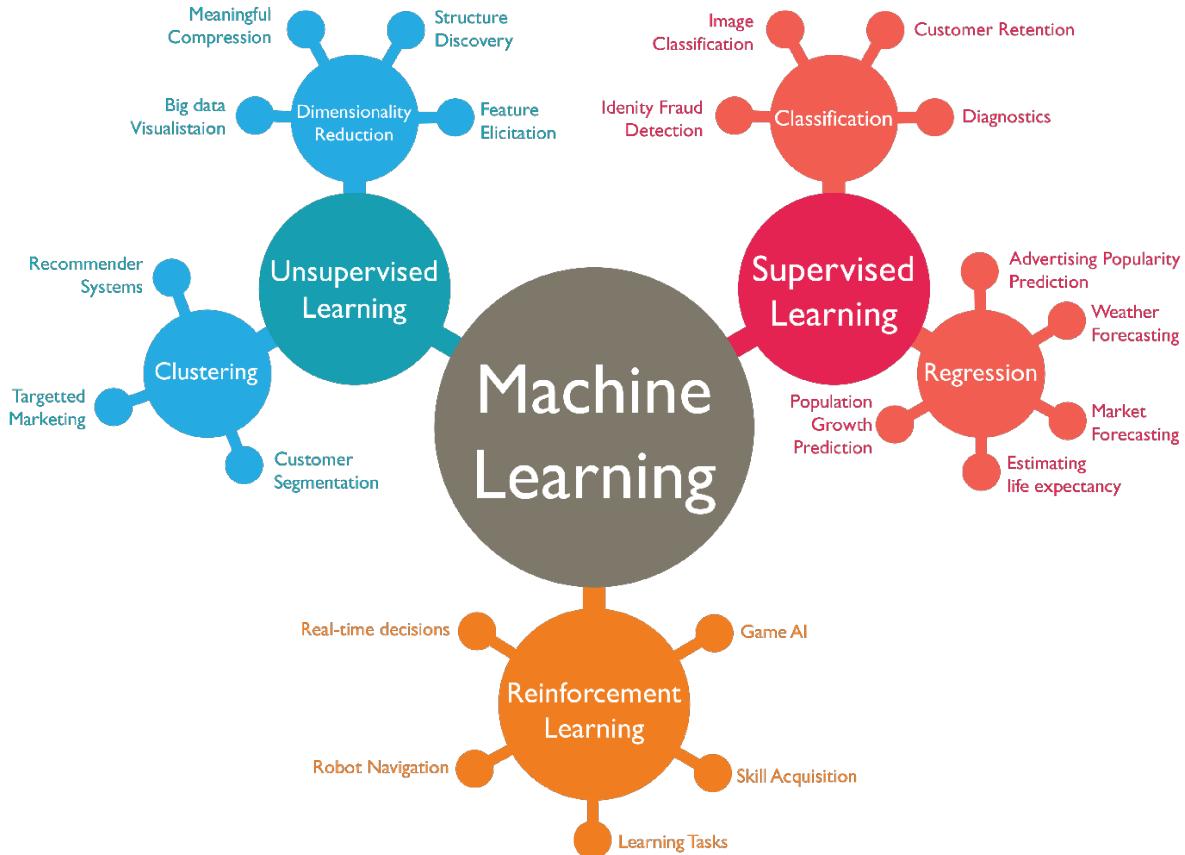
Skills needed:

- Math, statistics, and probability
- Comfortable working with data
- Programming skills

- Programming skills (Python, SQL, Java)
- Statistics and probability
- Prototyping
- Data modeling

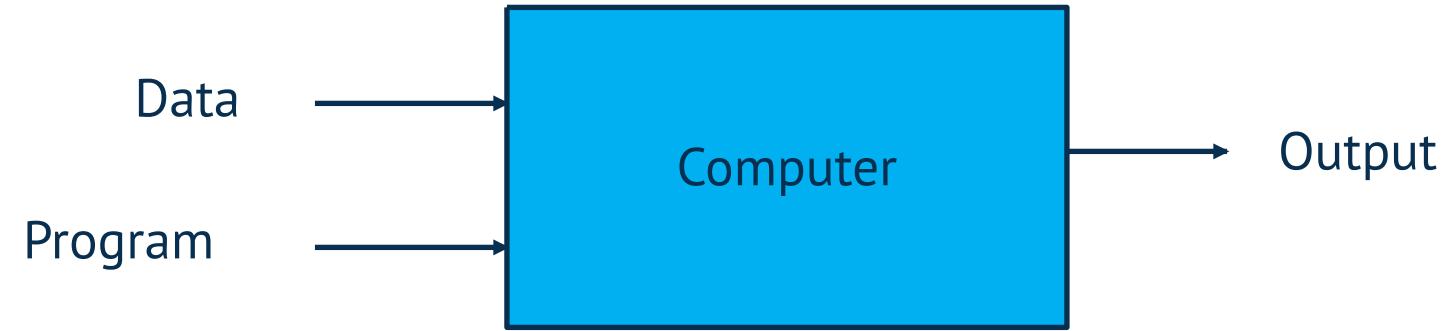
# Machine Learning

- Machine Learning is a subset of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to perform tasks without using explicit instructions.
- Instead, these systems rely on patterns and inference from data.

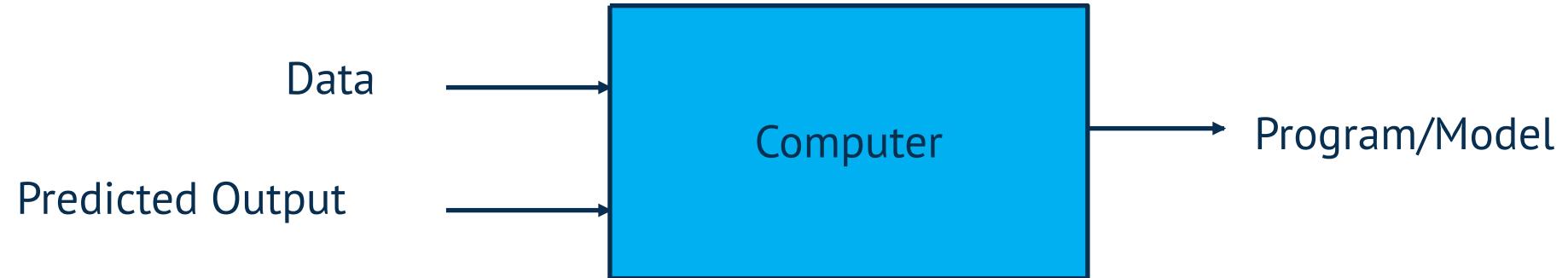


# Machine Learning

## Traditional Programming



## Machine Learning



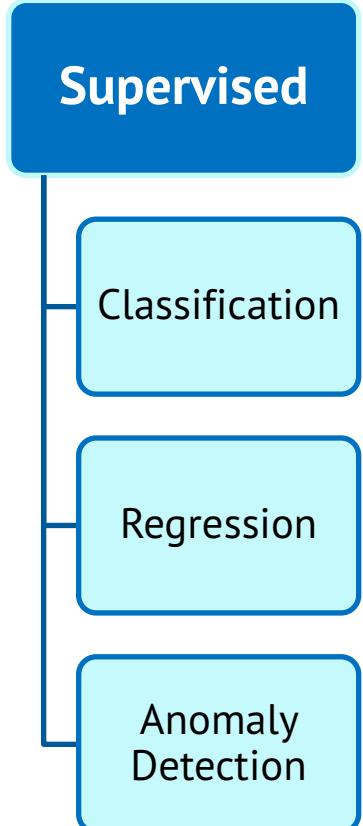
# Types of Machine Learning

- **Supervised learning:** given correct answers for each example (labeled data), learn the model to map input features to labels.
  - e.g.: learn to recognize cars by images of cars and non-cars
- **Unsupervised learning:** discover hidden patterns in data (unlabeled data)
  - Group toys of a given color or shape from a Lego set
  - e.g.: learn to categorize stars or genomic data when no labels are given (clustering, grouping)
- **Reinforcement learning:** occasional rewards\punishments after series of decisions
  - e.g.: learn to play tennis / table tennis

# Supervised Methods

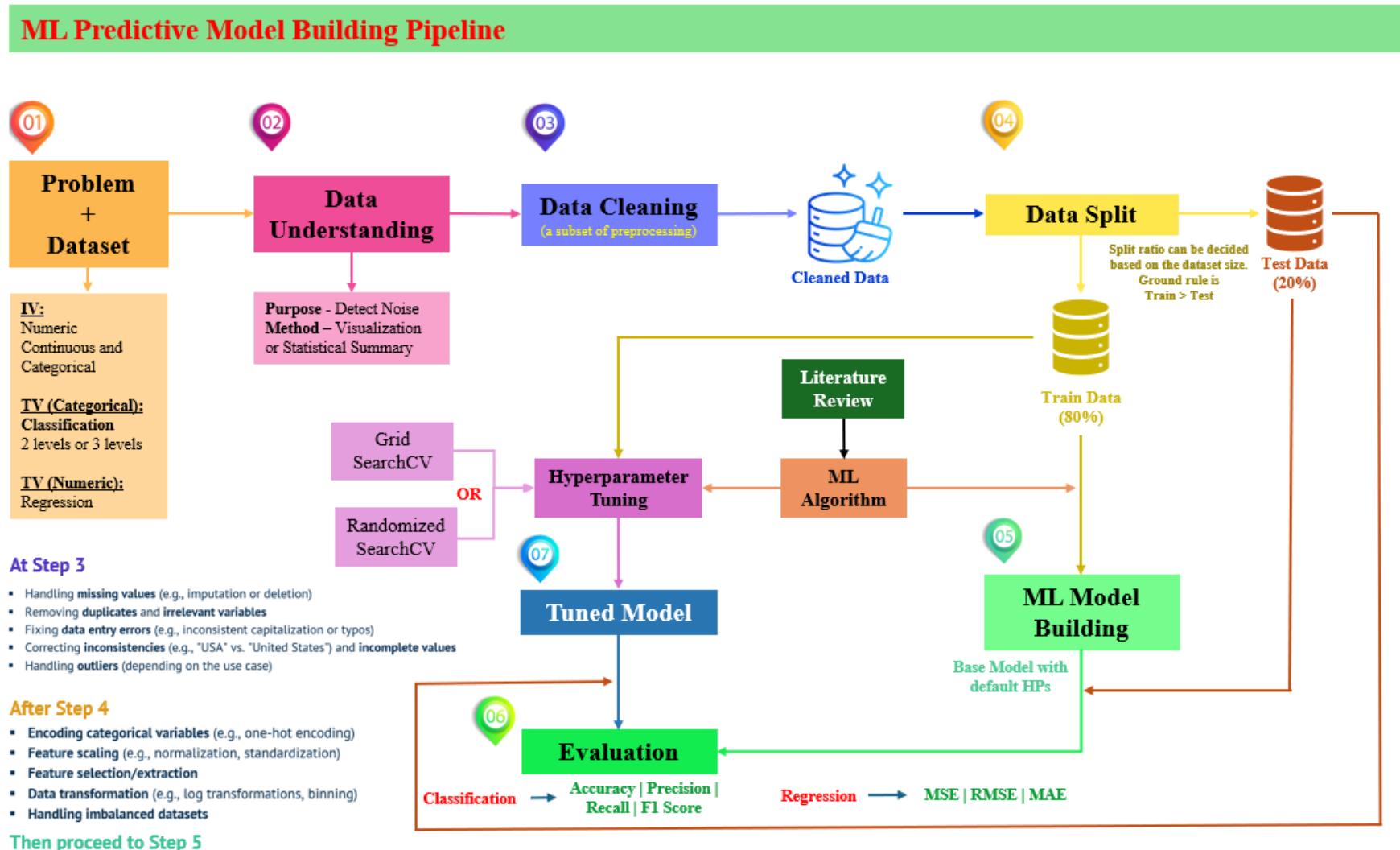
**Supervised Methods** (also called **Predictive**): Predict an unknown value(s) of a variable(s) from the values of some attributes

- **Classification:** predict the type/class of new cases
  - ✓ Spam Filtering, Handwriting Character Recognition, Patient Diagnosis
- **Regression:** predict a numerical value of new cases
  - ✓ Blood Pressure, Sales Amounts
- **Supervised Anomaly Detection:** identify items, events or observations deviating from expected patterns using data labeled as "normal" and "abnormal" (involves training a classifier)

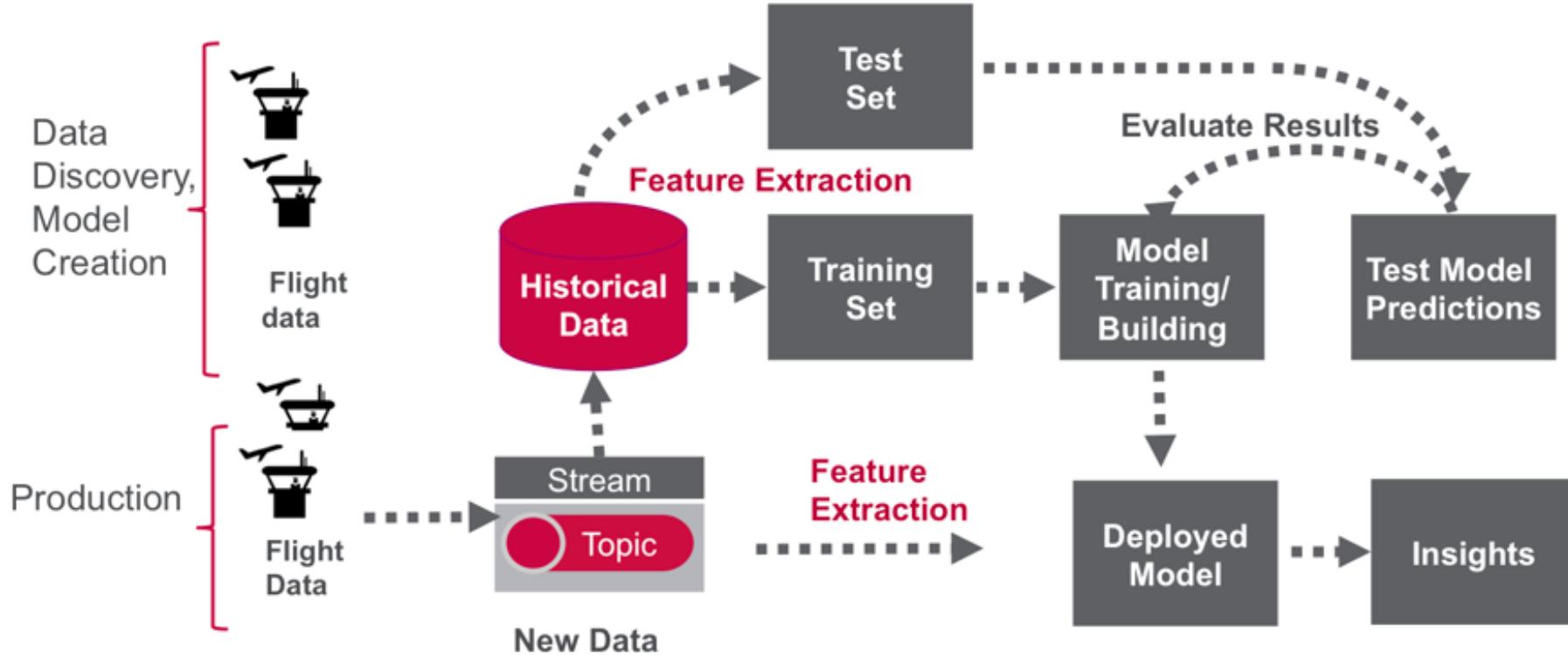


It is common to combine different methods such as clustering and classification (Hybrid methods)

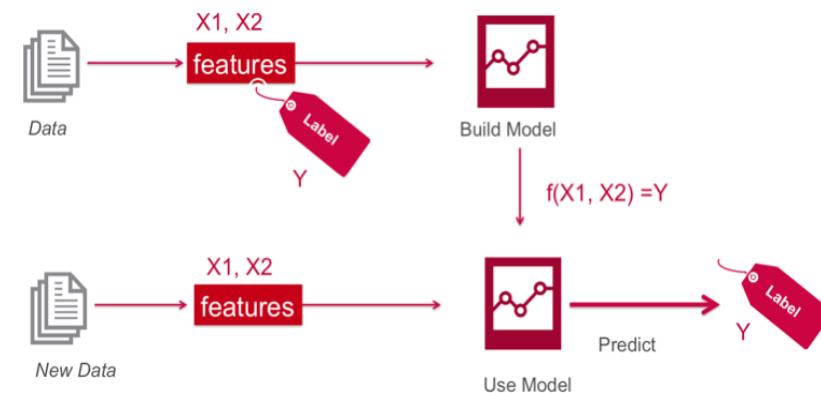
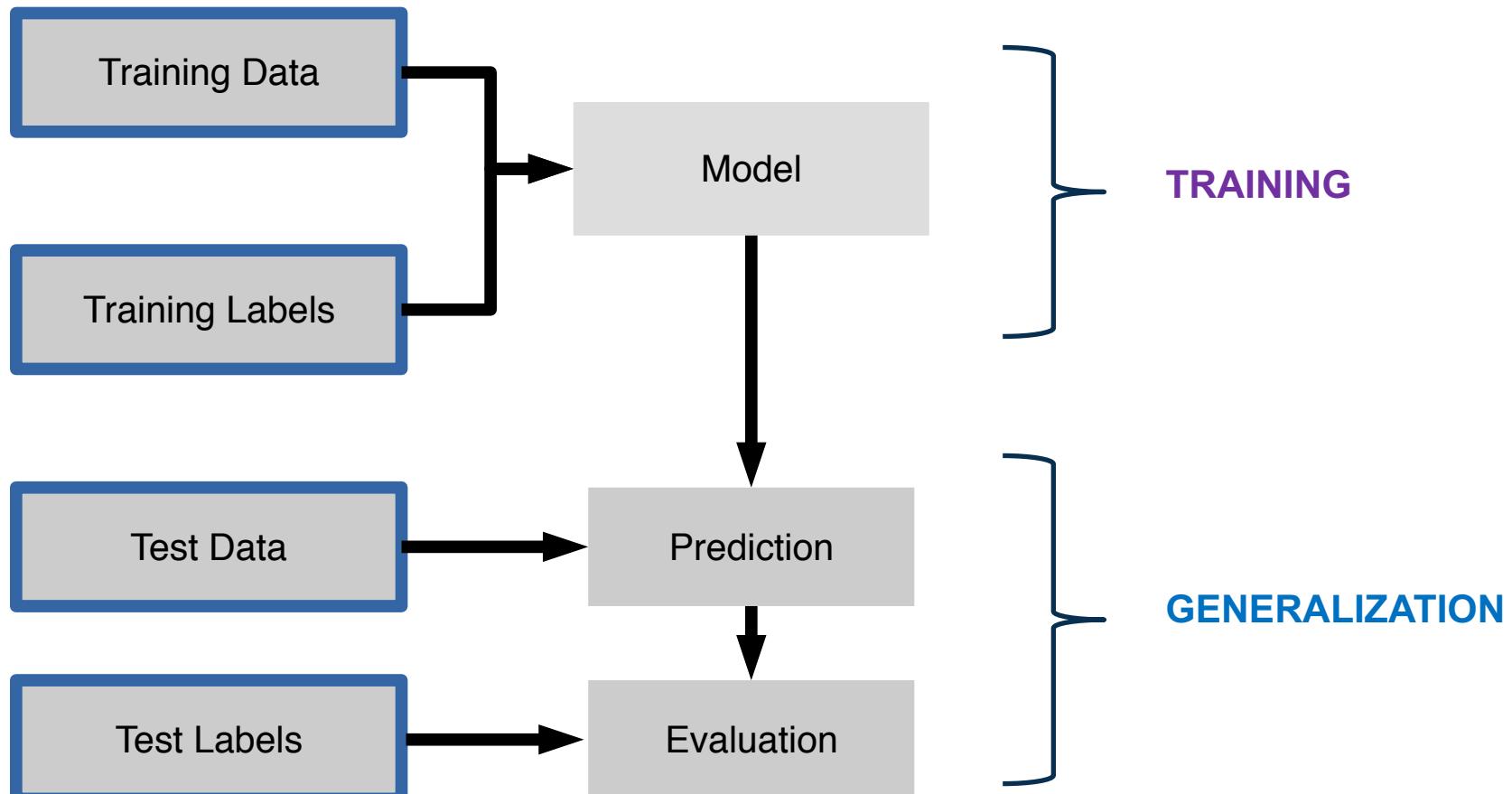
# Pipeline of Machine Learning Problem



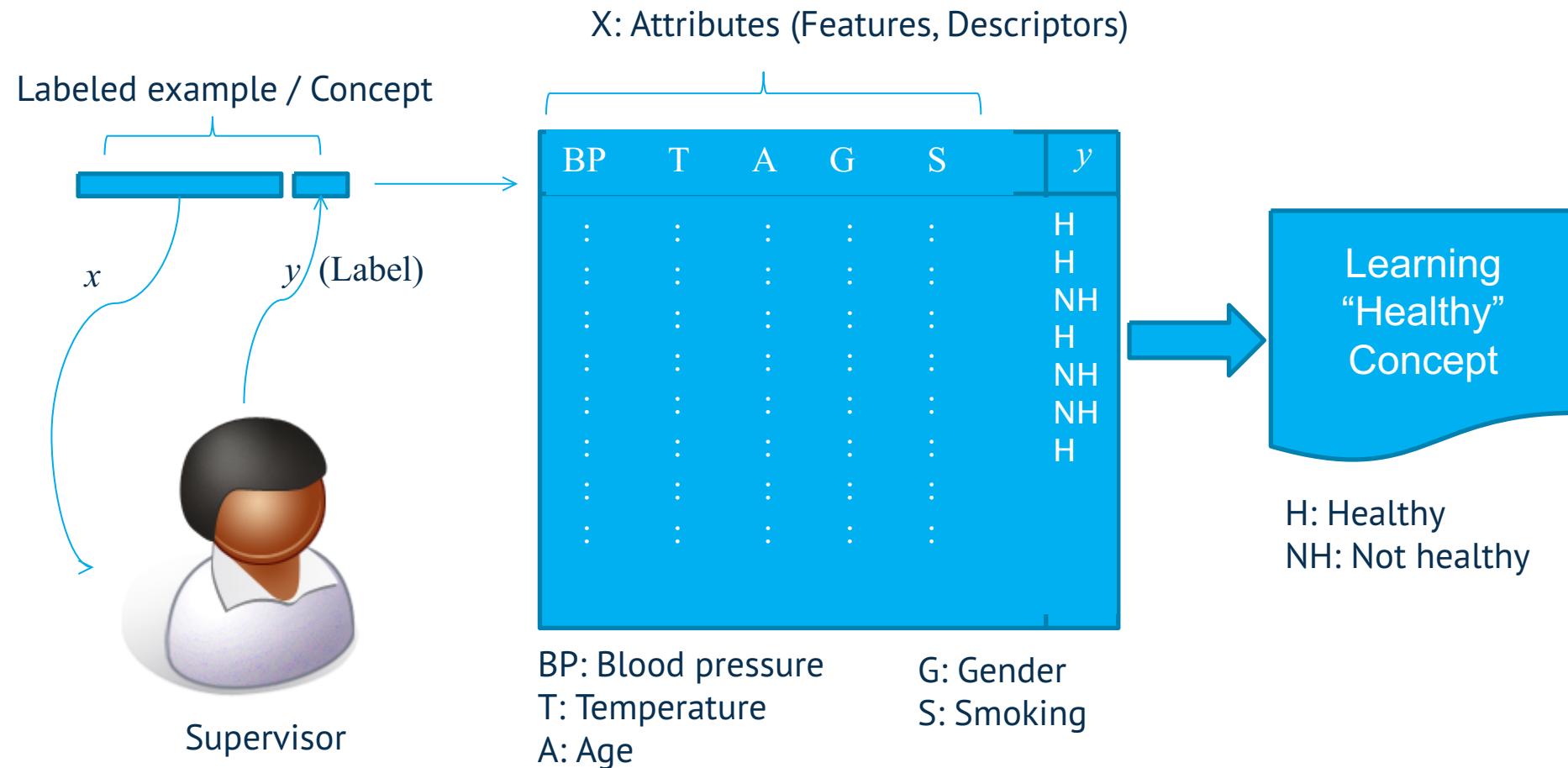
# Training and application Process of ML model



# Supervised Learning



# Supervised Learning



# Classification Example – Multilevel Target

## Iris Data Set

[Download: Data Folder](#), [Data Set Description](#)

**Abstract:** Famous database; from Fisher, 1936



X				Y
5.1	3.5	1.4	0.2	Setosa
7.0	3.2	1.4	0.2	Versicolor
6.3	3.3	6.0	2.5	Virginica
4.9	3.0	1.4	0.2	Setosa
7.7	3.8	6.7	2.2	Virginica
5.7	2.8	4.1	1.3	Versicolor

### Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm



### Label:

- 1- Iris Setosa
- 2- Iris Versicolour
- 3- Iris Virginica

# Classification Example – Binary Target

## Attribute Information:

1. X - x-axis spatial coordinate
2. Y - y-axis spatial coordinate
3. Month
4. Day
5. FFMC
6. DMC
7. DC
8. ISI
9. Temp
10. RH - relative humidity.
11. Wind - wind speed
12. Rain - outside rain

## Label:

Burned area or not

## Forest Fires Data Set

[Download](#): [Data Folder](#), [Data Set](#)  
[Description](#)



UC Irvine  
Machine Learning  
Repository



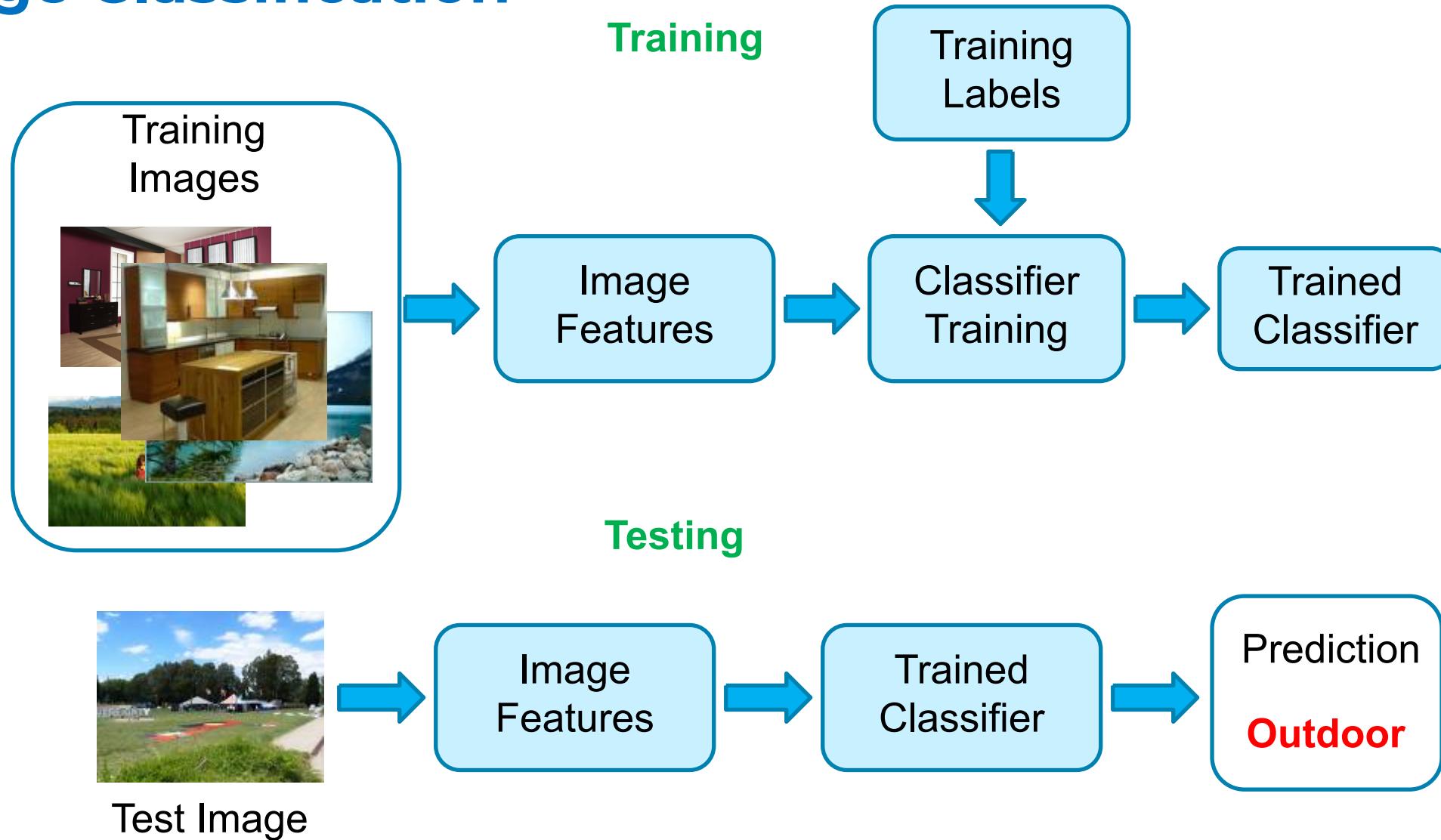
## Data:

X, Y, month, day, FFMC, DMC, DC, ISI, temp, RH, wind, rain	fire?
7, 5, Mar, Fri, 86.2, 26.2, 54.3, 5.1, 8.2, 51, 6.7, 0,	0
7, 4, Oct, Tue, 90.6, 35.4, 69.1, 6.7, 18, 33, 0.9, 0,	0
8, 6, Sep, Sat, 92.5, 121.1, 75, 8.6, 22, 56, 1.8, 0,	1

## Original problem is regression

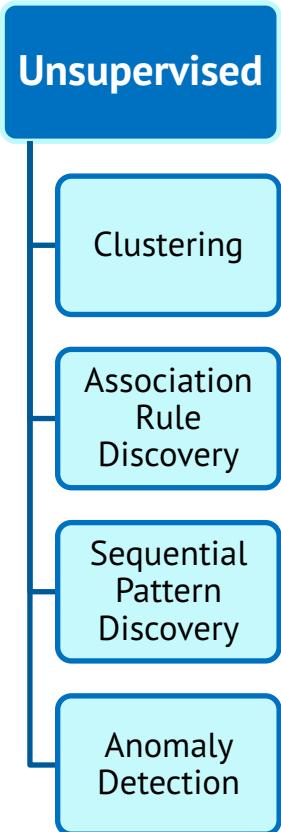
**Abstract:** This is a difficult regression task, where the aim is to predict the burned area of forest fires, in the northeast region of Portugal, by using meteorological and other data (see details at: [\[Web Link\]](#)).

# Image Classification

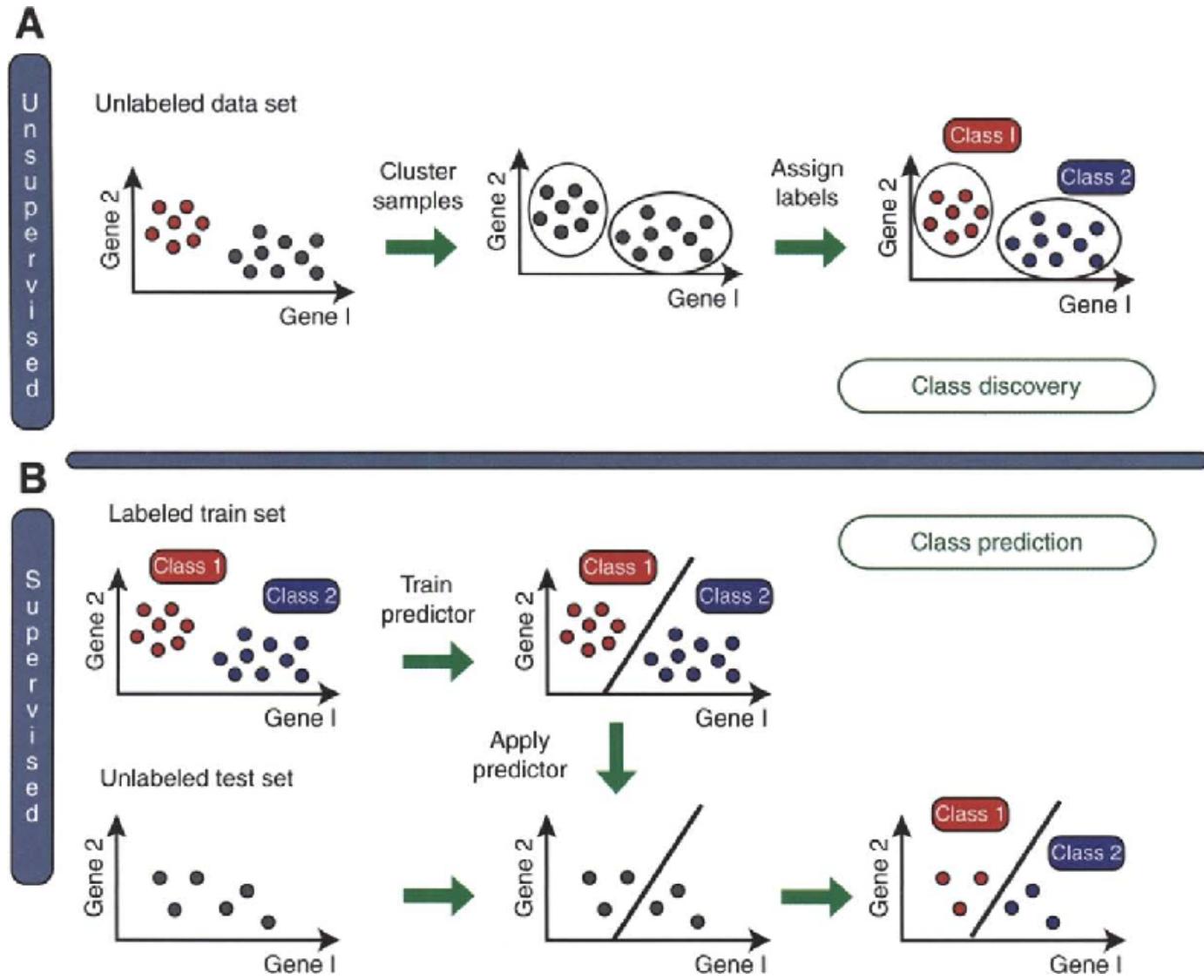


# Unsupervised Methods

- **Unsupervised Methods** (also called **Descriptive**): Try to find meaningful patterns in the data.
  - **Clustering**: group similar data into clusters
    - ✓ Market Segmentation, Document Clustering
  - **Association Rule Discovery**: find human interpretable patterns (associations)
    - ✓ Product Recommendations, Store Shelf Management
  - **Sequential Pattern Discovery**: describe the sequential dependencies among different events
    - ✓ Buying Patterns, Gene Sequencing
  - **Unsupervised anomaly detection**: to detect anomalies in unlabeled data under the assumption that the majority of the instances are normal
    - ✓ Fraud Detection, Network Intrusion Detection

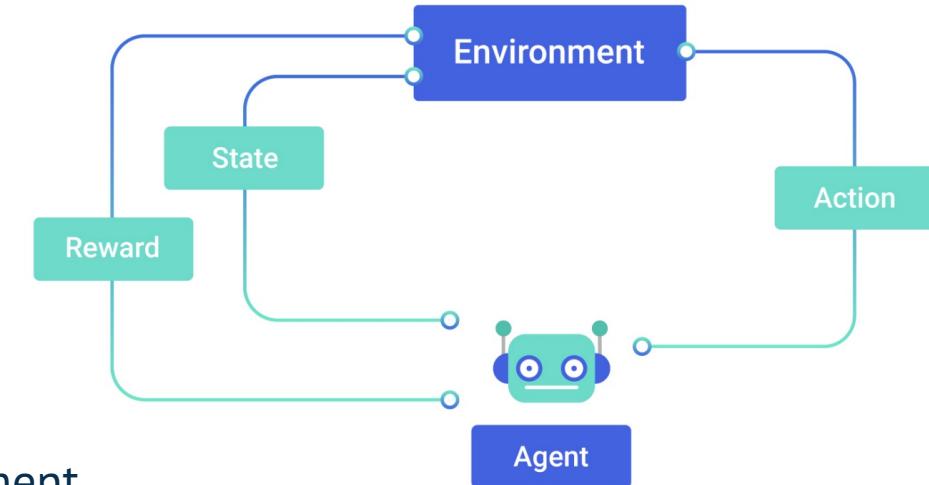


# Machine Learning Methods and Applications



# Reinforcement learning

- **Reinforcement learning (RL)** is an area of machine learning concerned with how software agents ought to take actions in an environment to maximize some notion of cumulative reward.
- The agent interacts with the environment in a trial-and-error manner, using feedback from its own actions and experiences to improve its performance over time.



**Agent:** The learner or decision maker that interacts with the environment.

**Environment:** The external system with which the agent interacts.

**State (s):** A representation of the current situation of the agent.

**Action (a):** The set of all possible moves the agent can make.

**Reward (r):** Feedback from the environment based on the action taken by the agent. It can be positive or negative.

# Some Applications

- **Retail:** Market basket analysis, Customer relationship management (CRM)
- **Finance:** Credit scoring, fraud detection
- **Manufacturing:** Optimization, troubleshooting
- **Medicine:** Medical diagnosis
- **Telecommunications:** Quality of service optimization
- **Web mining:** Search engines

# Statistical Modelling / Machine learning

- **Statistical modelling** is the formalization of relationships between variables in the form of mathematical equations.
- **Machine learning** is an algorithm that can learn from data without relying on rules-based programming.

# Statistical Modelling / Machine learning

## Assumptions in Statistical Models

Statistical modelling work on a number of assumptions. For instance, a linear regression assumes:

- Linear relation between independent and dependent variable
- Homoscedasticity (the error term is same across all values of the independent variables)
- Mean of error at zero for every dependent value
- Independence of observations
- Error should be normally distributed for each value of dependent variable

# Statistical Modelling / Machine learning

## ML/Statistical Models

- Lesser assumptions in a predictive model, higher will be the predictive power.
- **Machine Learning** as the name suggest needs minimal human effort.
  - Machine learning works on iterations where computer tries to find out patterns hidden in data.
  - Because machine does this work on comprehensive data and is independent of all the assumption, predictive power is generally very strong for these models.
- **Statistical model** are mathematics intensive and based on coefficient estimation.
  - It requires the modeler to understand the relation between variable before putting it in.

# Summary / Recap of Main Points

- The pipeline of machine learning involves several stages.
- Various types of methods and models are utilized in this process.
- One common approach is supervised learning.

# What To Expect Next Week

## In Class

- Introduction to Python for Data Science

## Preparation for Class

- Python for Data Science

CX016-2.5-3-IML & Introduction to Machine Learning

## **Introduction to Python for Data Science**

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understanding Python Basics and for Data Science.
2. Data Manipulation and Analysis.
3. Implementing Basic Data Science Techniques

# Contents & Structure

Introduction to Python Programming

Working with Data Structures

Introductions to Data Science libraries

Data Manipulation with pandas

Data Visualization with Matplotlib and Seaborn

Introduction to Basic Statistical Analysis

Data Pre-processing

Scikit-Learn library and its functions

# Recap From Last Lesson

- What are the major types of ML?
- What are the applications of ML?
- What are the benefits of predictive ML models?



## Installation Guides

# Python

- Python is a programming language that lets you work more quickly and integrate your systems more effectively.
- Key Features:
  - Readable and Simple Syntax: Python's syntax is clear and easy to understand.
  - Interpreted Language: Python code is executed line-by-line, which makes debugging easier.
  - Dynamically Typed: You don't need to declare the type of variable; it is inferred at runtime.
  - Extensive Standard Library: Python comes with a vast standard library.
  - Cross-Platform: Python can run on various operating systems.
  - Open Source: Python is free to use and distribute, even for commercial purposes.

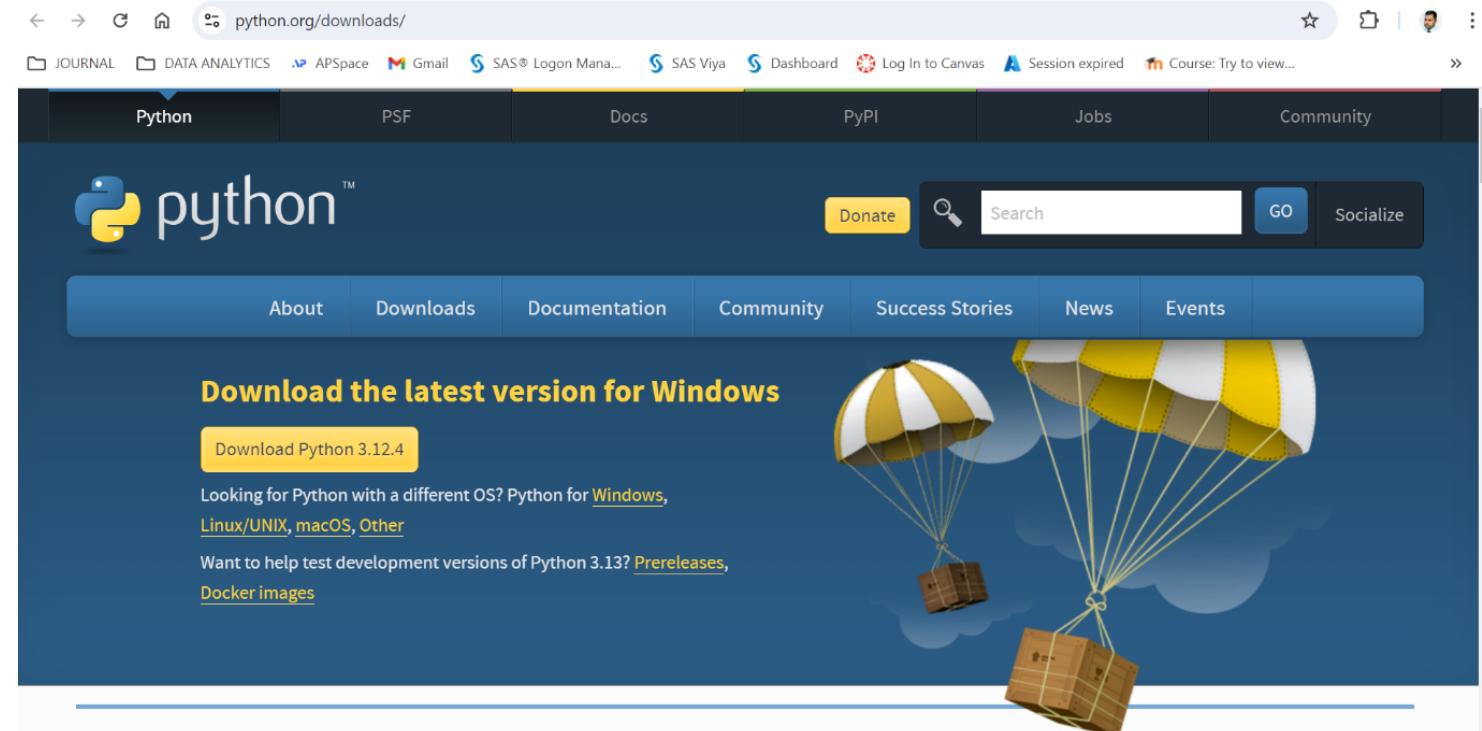
# Python

- Python is a programming language that lets you work more quickly and integrate your systems more effectively.
- Common Uses:
  - Web Development
  - Data Science and Machine Learning
  - Automation and Scripting
  - Software Development
  - Artificial Intelligence
  - Scientific Computing

# Installation Guide - Python

## Python

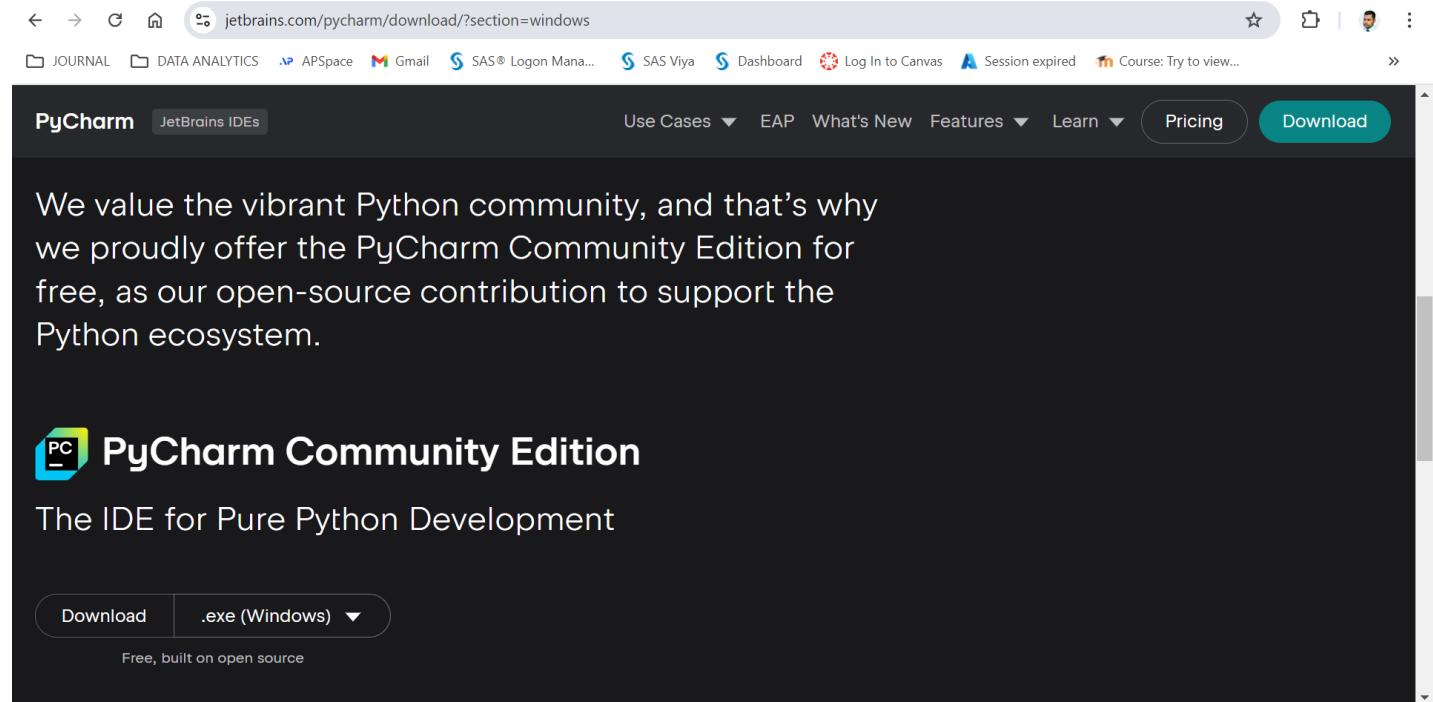
- Download and Install Python via <https://www.python.org/downloads/>
- Select the right version of Python according to the operating system of your laptop/desktop.
- Can use IDLE to write and execute python programs.
- Also, you can use either PyCharm or Visual Code as IDEs for your Python programming requirements.



# Installation Guide - PyCharm



- Download and Install PyCharm via
- <https://www.jetbrains.com/pycharm/download/#section=windows>
- PyCharm Community Edition – Free IDE
- You may also get specific version according to the operating system you use via
- <https://www.jetbrains.com/pycharm/download/other.html>

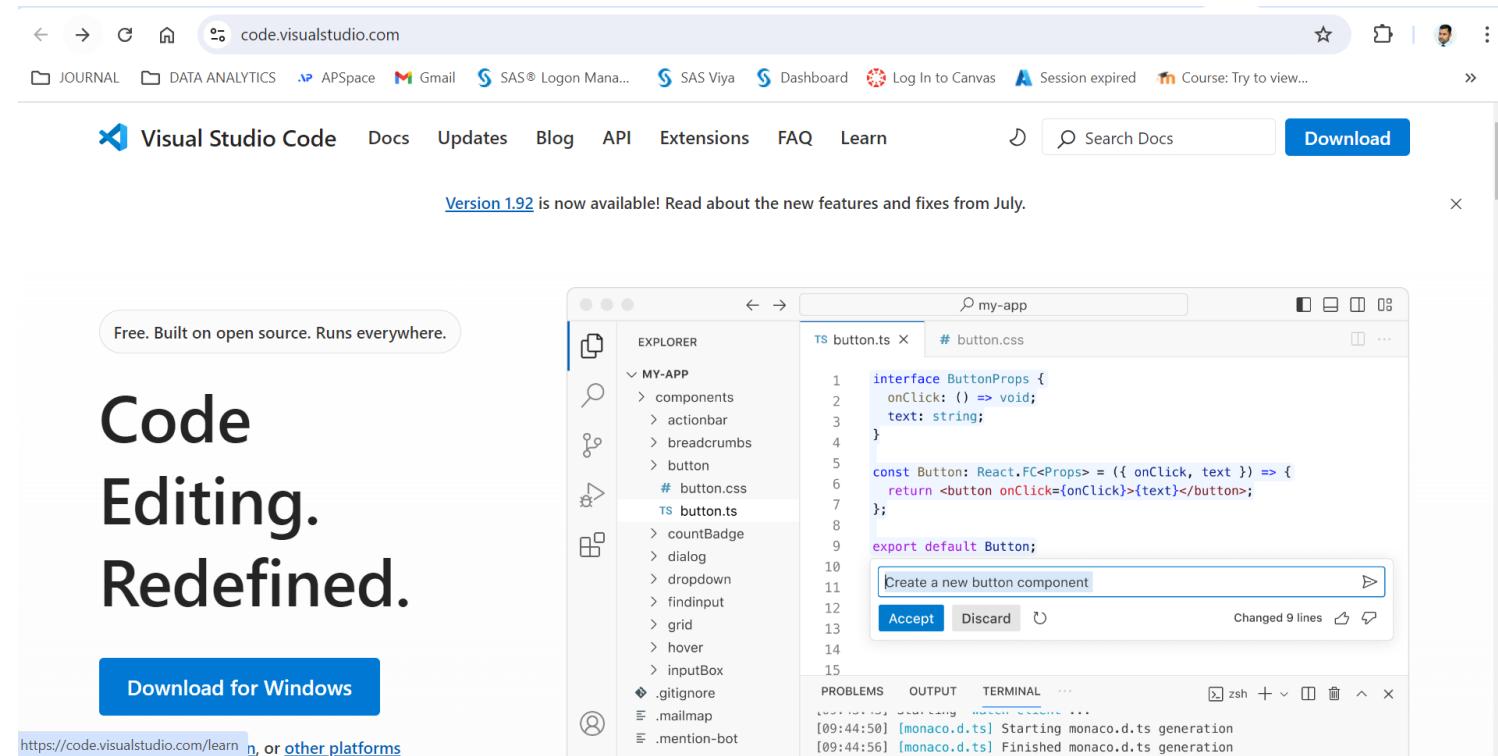


The screenshot shows a web browser displaying the JetBrains PyCharm download page for Windows. The URL in the address bar is [jetbrains.com/pycharm/download/?section=windows](https://jetbrains.com/pycharm/download/?section=windows). The page header includes the PyCharm logo and "JetBrains IDEs". Navigation links include "Use Cases", "EAP", "What's New", "Features", "Learn", "Pricing", and a prominent "Download" button. The main content area features a message about valuing the Python community and offering the PyCharm Community Edition for free. Below this, there is a section for the "PyCharm Community Edition" featuring its logo, the text "The IDE for Pure Python Development", and download links for "Download" and ".exe (Windows)". A note at the bottom states "Free, built on open source".

# Installation Guide - Visual Code

## Visual Code

- Download and Install Visual Code via
- <https://code.visualstudio.com/>
- Select the specific one according to the operating system

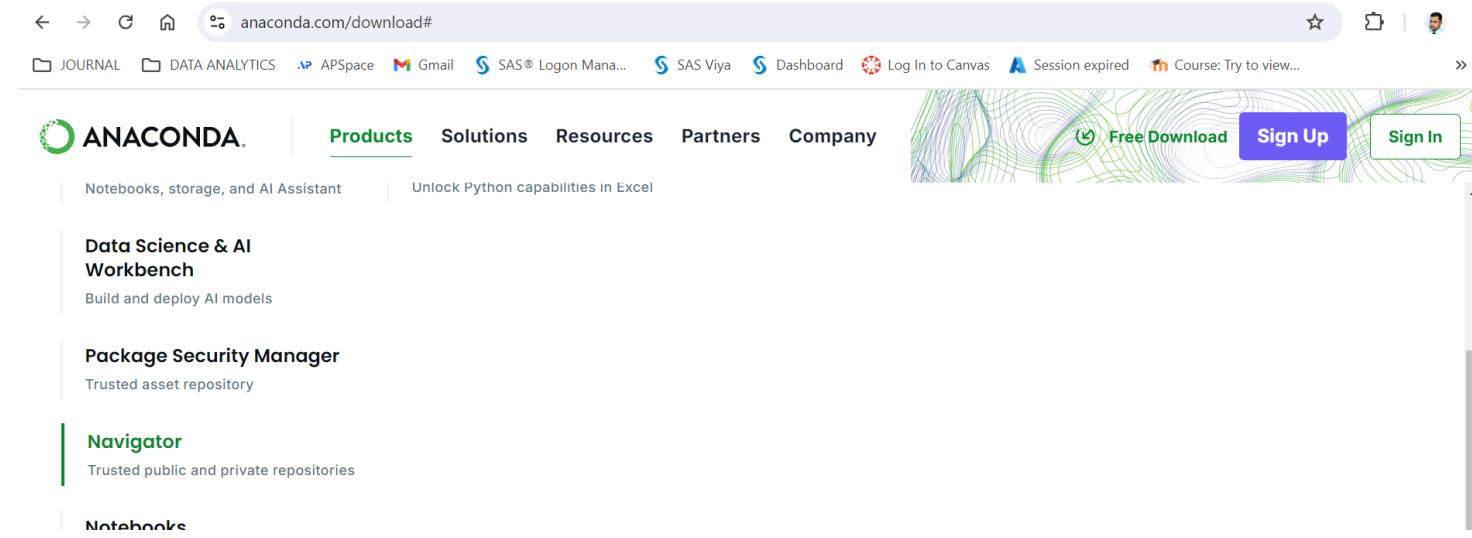


The image contains two parts. On the left, the official Visual Studio Code website homepage is shown. It features the title "Code Editing. Redefined.", a "Download for Windows" button, and a "my-app" code editor window displaying a TypeScript file (button.ts) with code for a button component. On the right, a screenshot of the Visual Studio Code application itself is displayed. The interface shows the "EXPLORER" sidebar with a "MY-APP" folder containing various UI components like "components", "actionbar", "breadcrumbs", "button", etc. The main editor area shows the same "button.ts" file with syntax highlighting. A floating "Create a new button component" dialog box is open over the code. The bottom of the screen shows the "PROBLEMS", "OUTPUT", and "TERMINAL" tabs, with some terminal logs visible.

# Installation Guide - Anaconda



- Download and Install Anaconda via
- <https://www.anaconda.com/>
- Download Navigator and Install.
- After the installation, you will get an Anaconda Navigator and can load the Jupyter Lab or Jupyter Notebook to work with python programming.

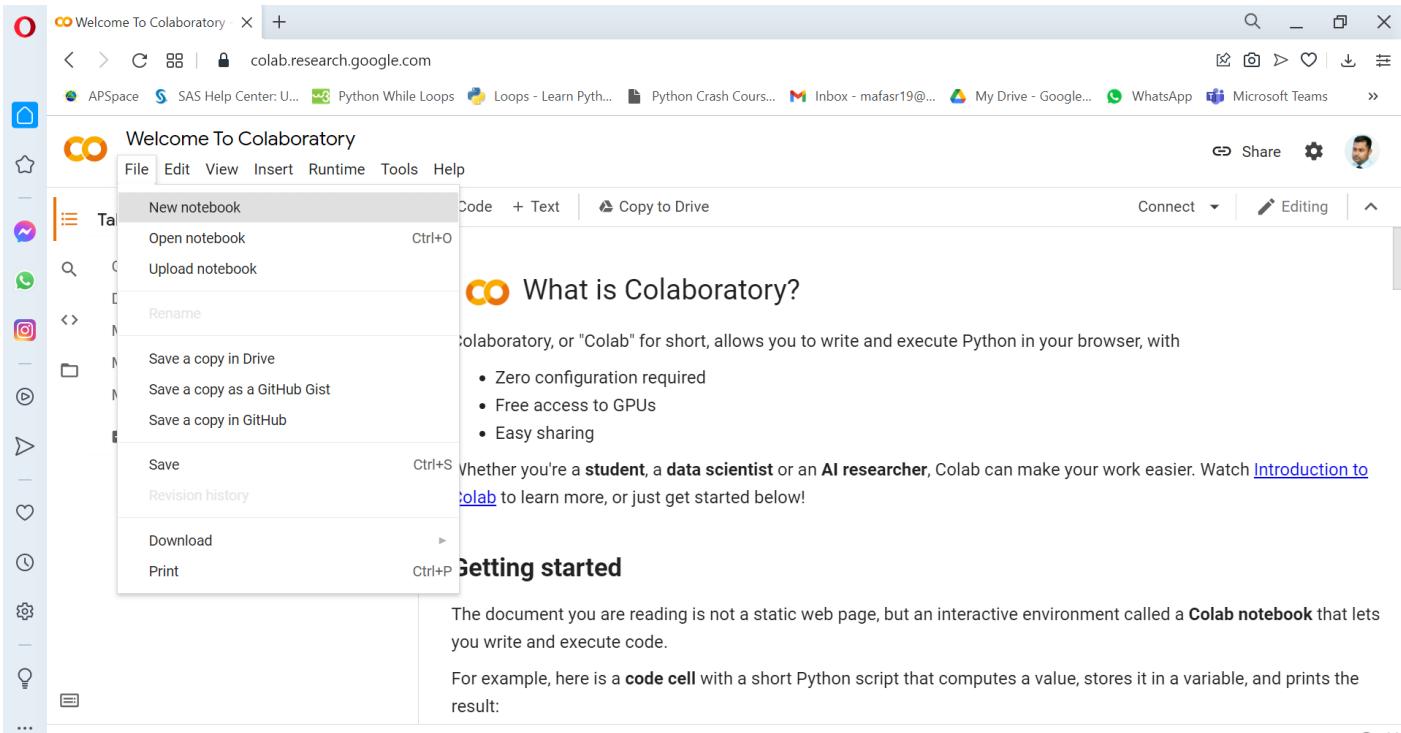


The screenshot shows the Anaconda website at [anaconda.com/download#](https://anaconda.com/download#). The header includes links for JOURNAL, DATA ANALYTICS, APSpace, Gmail, SAS Logon Manager, SAS Viya, Dashboard, Log In to Canvas, Session expired, and Course: Try to view... A search bar is also present. The main navigation menu has 'Products' selected, along with 'Solutions', 'Resources', 'Partners', and 'Company'. Below the menu, there are sections for Notebooks, storage, and AI Assistant; Data Science & AI Workbench (with a sub-section for Build and deploy AI models); Package Security Manager (with a sub-section for Trusted asset repository); Navigator (with a sub-section for Trusted public and private repositories); and Notebooks. On the right side, there is a 'Free Download' button and links for 'Sign Up' and 'Sign In'.

# Installation Guide - Colab



- Use Colab via  
<https://colab.research.google.com>
- You need to have a Gmail account to use this facility.
- Colab is an IDE maintained by Google.



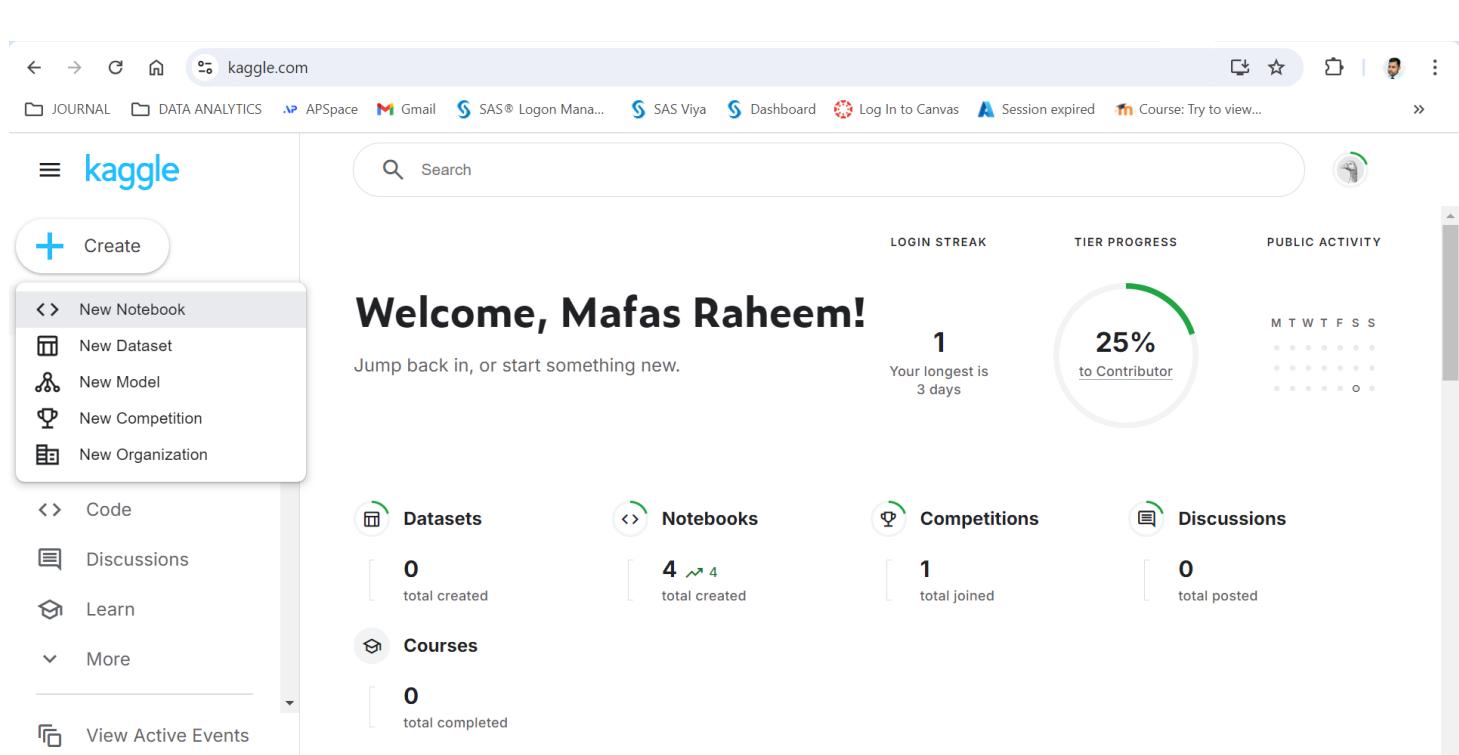
The screenshot shows the Google Colab interface. At the top, there's a browser-like header with tabs, a search bar, and various icons. Below it is the main Colab interface. On the left, there's a sidebar with icons for file operations like New notebook, Open notebook, and Save. The main content area displays the 'Welcome To Colaboratory' page. It features a heading 'What is Colaboratory?' followed by a list of benefits: Zero configuration required, Free access to GPUs, and Easy sharing. Below that, it says 'Whether you're a student, a data scientist or an AI researcher, Colab can make your work easier. Watch [Introduction to Colab](#) to learn more, or just get started below!'. At the bottom, there's a section titled 'Getting started' with a note about the document being an interactive environment and a code cell example.

# Installation Guide - Kaggle

**Kaggle**

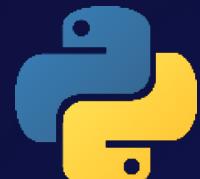


- Use Kaggle via  
<https://www.kaggle.com/>
- You need to have an account to use this facility. Can use Gmail too.



The screenshot shows the Kaggle homepage with a user profile for "Mafas Raheem". The top navigation bar includes links for JOURNAL, DATA ANALYTICS, APSPACE, Gmail, SAS Logon Manager, SAS Viya, Dashboard, Log In to Canvas, Session expired, and Course: Try to view... The main area features a "Welcome, Mafas Raheem!" message and a summary of activity metrics. A sidebar on the left provides navigation options like "New Notebook", "Code", "Discussions", "Learn", and "More".

Category	Value
Datasets	0 total created
Notebooks	4 total created
Competitions	1 total joined
Discussions	0 total posted



PYTHON

## Basics

# Python Syntax vs. other programming languages

- Check the python version in Anaconda Jupyter or Colab

```
from platform import python_version  
python_version()
```

- A simple Syntax is followed when writing python codes.
- Indentation refers to the spaces at the beginning of a code line and is very important in python.
- Python uses indentation to indicate a block of code.
- Example:  
`print("Hello, World!")`

# Python Comments

- Comments are used to explain python code.
- It makes the code more readable.
- It prevents execution of code line(s) when testing code.

## Example:

- `# This is a comment - Single line comment`  
`print("Hello, World!")`

- Inline comment:

`print("Hello World") # This code prints "Hello World"`

- multiple line comments use `"""` or `'''` on either end.

`"""`

*This type of comment spans multiple lines.*

*These are mostly used for documentation of functions, classes and modules.* `"""`

# Variables

- Variables are data value storing containers in a programming language.
- In Python a variable is created when a value is assigned.

## • Example:

```
a = 10
```

```
name = 'John' or name = "John"
```

- Can find the variable type/data type by using

```
type(name)
```

- Variable names are case-sensitive.

## A variable name

- Must start with a letter or the underscore character.
- Cannot start with a number.
- Can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ ).
- Is case-sensitive (name, Name and NAME are three different variables).

# Variable – Input and Output

`print()` function → can be used for  
single variable

```
a = "Python is awesome"  
print(a)
```

multiple variables

```
x = "Python"  
y = "is"  
z = "awesome"  
print(x, y, z)
```

```
a = input("Enter the value:")  
print(a)
```

Can use + operator too →

```
x = "Python "  
y = "is "  
z = "awesome"  
print(x + y + z)
```

# Operator Precedence

Operators	Meaning
<code>()</code>	Parentheses
<code>**</code>	Exponent
<code>+x, -x, ~x</code>	Unary plus, Unary minus, Bitwise NOT
<code>*, /, //, %</code>	Multiplication, Division, Floor division, Modulus
<code>+, -</code>	Addition, Subtraction
<code>&lt;&lt;, &gt;&gt;</code>	Bitwise shift operators

# Operator Precedence

Operators	Meaning
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
<b>==, !=, &gt;, &gt;=, &lt;, &lt;=, is, is not, in, not in</b>	Comparisons, Identity, Membership operators
<b>not</b>	Logical NOT
<b>and</b>	Logical AND
<b>or</b>	Logical OR

# Collections in Python

List	Tuple	Set	Dictionary
ordered	ordered	unordered	ordered
changeable	unchangeable	unchangeable	changeable
Allows duplicate	Allows duplicate	No duplicate	No duplicate
		unindexed	

# Collections in Python

## LIST

- Example:
- `list = ["apple", "banana", "cherry"]`

## TUPLE

- Example:
- `mytuple = ("apple", "banana", "cherry")`

## SET

- Example:
- `my_set = {1, 2, 3, 4, 5}`

## DICTIONARY

Example:

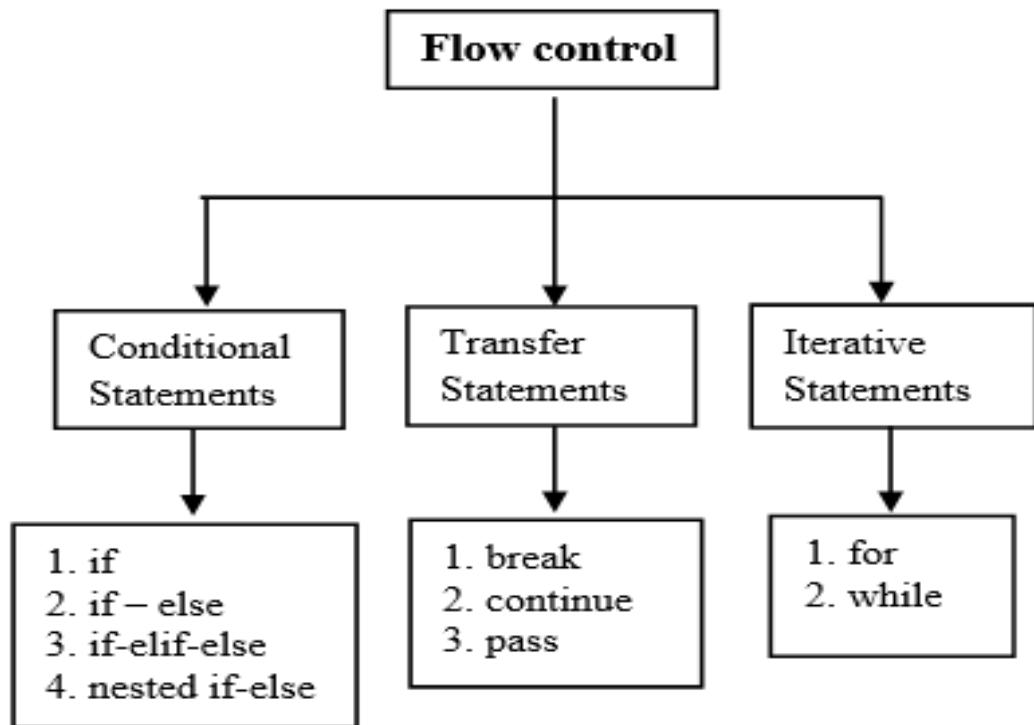
```
marks = {  
    "maths" : 86,  
    "science" : 82,  
    "english" : 93  
}
```

`marks["maths"]`

# Control Flows in Python

The flow controls in python are divided into three types:

- Conditional statements
- Iterative statements.
- Transfer statements



# Control Flows - Conditional statements

## IF →

- Is the simplest form and evaluates a condition to either True or False.

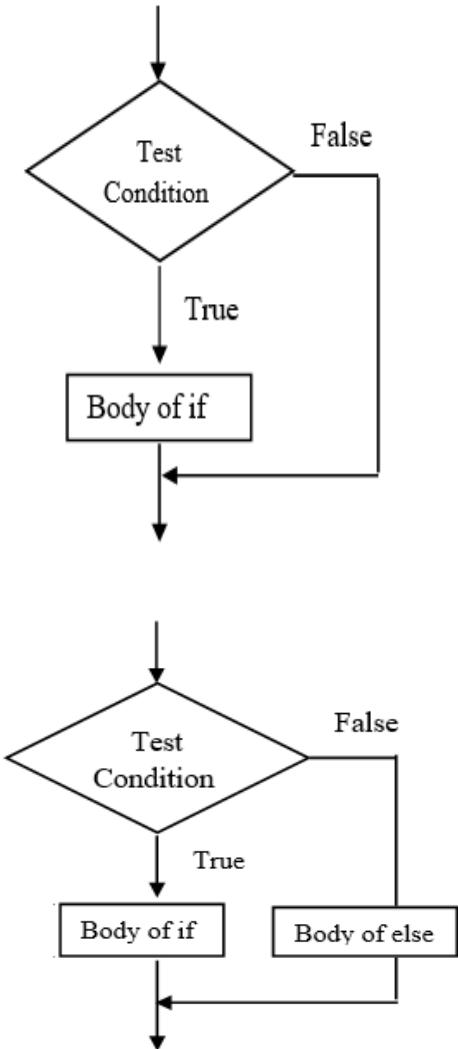
Syntax:

if condition:

```
    statement 1
    statement n
```

## Example:

```
number = 6
if number > 5:
    # Calculate square
    print(number * number)
print('Next lines of code')
```



## IF - ELSE →

Checks the condition and executes the if block of code when the condition is **True** and executes the else block of code if the condition is **False**.

## Example:

```
password =
input('Enter password:
')
if password ==
"PYnative@#29":
    print("Correct
password")
else:
    print("Incorrect
Password")
```

## Syntax:

```
if condition:
    statement 1
else:
    statement 2
```

# Control Flows - Iterative Statements

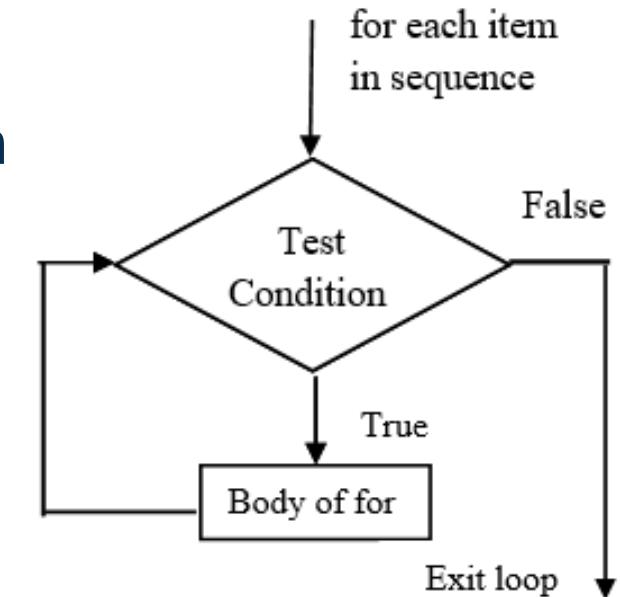
## FOR loop

- Can iterate any sequence or iterable variable. The sequence can be string, list, dictionary, set, or tuple.
- Syntax:
- `for i in range/sequences:`
- statement 1
- statement 2
- statement n

**Example:**

```
for i in range(1, 11):
    print(i)
```

```
~~~~~
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
    for y in fruits:
        print(x, y)
```



**Loop Through a List, Tuple, Dictionary and Tuple is possible**

# Control Flows - Iterative Statements

## WHILE loop

### Syntax

```
while condition:  
    body of while loop
```

Repeatedly executes a code block while a particular condition is true.

Can also use ELSE

```
while condition: # until this condition is true  
    statement 1
```

.

.

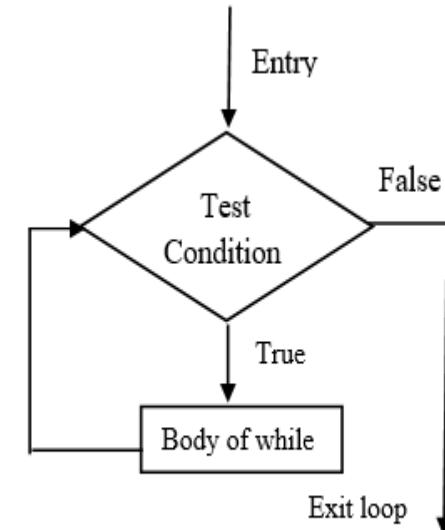
statement n

else:

statement(s)

### Example:

```
num = 10  
sum = 0  
i = 1  
while i <= num:  
    sum = sum + i  
    i = i + 1  
print("Sum of first 10 number is:", sum)
```



### Body of while loop

Gets executed as long as the condition is true

### Else block is optional

Gets executed when while loop is executed normally

# User defined function in Python

A function is a block of code which runs when it is called. A function can accept parameters and arguments and can return value as a result.

In Python a function is defined using the **def** keyword:

```
def my_function():
    print("My name is Mafas")
my_function()
```

**Example:**

```
def my_function(name1, name2):
    print("My name is", name1, name2)
```

```
my_function(name1 = "Mafas", name2 = "Raheem")
```

**Example:**

```
def my_function(name = "Mafas"):
    print("My name is", name)
```

```
my_function()
```

```
my_function("Aahil")
```

```
my_function(name1 = "Mafas", name2 = "Raheem")
```



## Libraries



# Python Libraries

## Required for Data Science Projects

- **Pandas** 
  - Data Loading | Reading | Data Management | Basic Visualization
- **Matplotlib** 
  - 2D Visualization
- **Seaborn** 
  - 2D & 3D Visualization
- **TensorFlow**   

  - Pre-processing | Data Split | Modelling | Evaluation
- **NumPy** 
  - 2D & 3D Visualization
- **Scikit Learn** 
  - Pre-processing | Data Split | Modelling | Evaluation
- **Keras** 
  - Pre-processing | Data Split | Modelling | Evaluation



Pandas Library



# Pandas

## Data Management using Pandas



### CREATE A DATAFRAME

```
import pandas as pd  
  
df = pd.DataFrame({ "Name" : ["Ann", "Barack", "Charles"],  
                    "Age" : [25, 65, 70]}, index = [1, 2, 3])  
  
df
```

### LOAD A DATASET

```
import pandas as pd  
  
data = pd.read_csv("C:/APU/MATERIALS/DATASETS/Churn_Modelling.csv")  
  
data
```

**Suitable read functions of Pandas library must be used to read the specific file format of the data.**

# Pandas

## Data Management using Pandas



### LOAD A DATASET

Load the data using the Pandas library. Pandas library can read/write the following data file formats.

Format				
Type	Data Description	Reader	Writer	
text	CSV	<a href="#">read_csv</a>	<a href="#">to_csv</a>	
text	Fixed-Width Text File	<a href="#">read_fwf</a>		
text	JSON	<a href="#">read_json</a>	<a href="#">to_json</a>	
text	HTML	<a href="#">read_html</a>	<a href="#">to_html</a>	
text	LaTeX		<a href="#">Styler.to_latex</a>	
text	XML	<a href="#">read_xml</a>	<a href="#">to_xml</a>	
text	Local clipboard	<a href="#">read_clipboard</a>	<a href="#">to_clipboard</a>	
binary	MS Excel	<a href="#">read_excel</a>	<a href="#">to_excel</a>	

# Pandas

## Data Management using Pandas



### LOAD A DATASET

Load the data using the Pandas library. Pandas library can read/write the following data file formats.

binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle

# Pandas

## Data Management using Pandas



### LOAD A DATASET

Load the data using the Pandas library. Pandas library can read/write the following data file formats.

```
data = pd.read_csv('file path')
```

```
data = pd.read_excel('file
path', sheet_name = 'Sheet1')
```

SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

### VIEW DATA

```
# display the first 5 rows
data.head()
```

```
# display the last 5 rows
data.tail()
```

# Pandas

## Data Management using Pandas



### UNDERSTAND DATA

`data.shape` → dimension of the data

`len(data)` → display the no of records

`data.columns` → list out the columns

`data.dtypes` → column names and the respective data types

# Pandas

## Data Management using Pandas



### UNDERSTAND DATA

```
# to get a summary of the categorical data  
data['Gender'].value_counts() → any categorical variable can be used to get  
the details of it.
```

```
# to get the mode  
data['Gender'].mode()
```

```
# to get the total number of records  
data['Gender'].count()
```

# Pandas

## Data Management using Pandas



### UNDERSTAND DATA

```
# to get the descriptive statistics
```

```
data['EstimatedSalary'].describe() → any numeric continuous variable can  
be used to get the details of it. The details will be:
```

```
count, mean, std, min, 25%, 50%, 75%, max
```

The following methods can also be exclusively used to get the details such as  
`count()`, `mean()`, `std()`, `min()`, `max()`

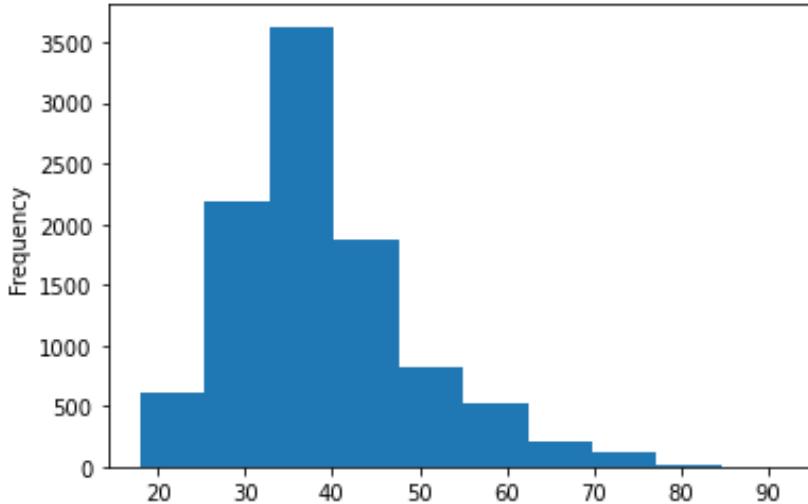
# Pandas

## Data Management using Pandas

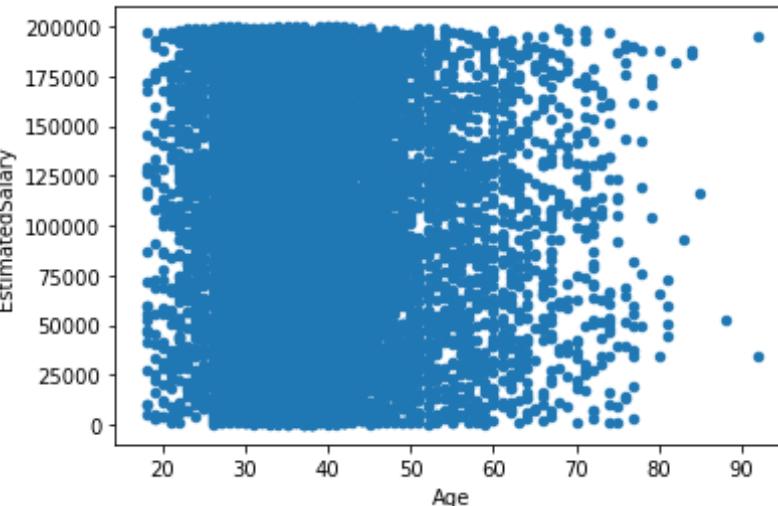


### VISUALIZE DATA

```
# to visualize → offered by PANDAS  
library  
data[ 'Age' ].plot.hist() # univariate
```



```
data.plot.scatter(x = 'Age', y =  
'EstimatedSalary') # bivariate
```



# Pandas

## Data Management using Pandas



### TAKE SAMPLE OF THE DATA

```
# take sample of the dataset
data_1 = data.sample(frac=0.5, random_state=1) # percentage to get
the sample in a random manner
data_2 = data.sample(n=100, random_state=1)

# to export as csv
data_2.to_csv("C:/Users/raheem/OneDrive - Asia Pacific
University/Desktop/data_2.csv")
```

# Pandas

## Data Management using Pandas



### MISSING DATA

```
# detect missing values  
data_new.isnull().sum()
```

```
# drop the rows which contain  
missing values  
  
data_new.dropna()
```

### MANAGE DATA

```
# dropping unwanted variables  
= part of preprocessing  
  
data_3 =  
data.drop('RowNumber',  
axis=1)  
  
  
data_new =  
data.drop(data.columns[[0, 1,  
2]], axis=1)
```

# Pandas

## Data Management using Pandas



### MANAGE DATA

```
# convert the data type of a variable  
data['Balance'] = data['Balance'].astype(int)
```

```
# create a new variable  
data['Balance_new'] = data['Balance'].astype(int)
```

# Pandas

## Data Management using Pandas



### MANAGE DATA

```
# group by → Descriptive analytics  
data.groupby(['Geography', 'Gender'])['Geography'].count()  
  
data.groupby(['Geography', 'Exited'])['Exited'].count()  
  
data.groupby(['Geography', 'Age', 'Exited'])['Exited'].count()
```

# Pandas

## Data Management using Pandas



### MERGE DATA

- “Merging” is the process of bringing two datasets together into one and aligning the rows from each based on common attributes or columns.
- The words “merge” and “join” are used relatively interchangeably in Pandas and other languages and they both do the similar things.

`pandas.DataFrame.merge #`

```
DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None,  
left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'),  
copy=True, indicator=False, validate=None)
```

[source]

Merge DataFrame or named Series objects with a database-style join.

# Pandas

## Data Management using Pandas



### MERGE DATA

```
import pandas as pd  
  
data_1 = pd.DataFrame(  
    {"X1" : ['A', 'B', 'C'],  
     "X2" : [1, 2, 3]}  
)  
  
data_1
```

### MERGE DATA

```
data_2 = pd.DataFrame(  
    {"X1" : ['A', 'B', 'D'],  
     "X3" : ['T', 'F', 'T']}  
)  
  
data_2  
  
pd.merge(data_1 , data_2)
```

# Pandas

## Data Management using Pandas



### MERGE DATA

```
import pandas as pd
```

```
user_usage = pd.read_csv('C:/Users/raheem/OneDrive - Asia Pacific  
University/Desktop/Merge/user_usage.csv')
```

```
user_device = pd.read_csv('C:/Users/raheem/OneDrive - Asia Pacific  
University/Desktop/Merge/user_device.csv')
```

```
# identify a common variable among the datasets
```

```
result_1 = pd.merge(user_usage, user_device, on = 'use_id')
```

# Pandas

## Data Management using Pandas



### MERGE DATA

- **Inner Merge**
- Pandas uses “inner” merge by default. This keeps only the common values in both the left and right dataframes for the merged data.

```
# Inner Merge/Join
result_inner = pd.merge(user_usage, user_device[['use_id',
'platform', 'device']], on = 'use_id', how = 'inner')
```

# Pandas

## Data Management using Pandas



### MERGE DATA

- **Left Merge**
- Keep every row in the left data frame. Where there are missing values of the “on” variable in the right data frame, add empty / NaN values in the result.

```
# Left
result_left = pd.merge(user_usage, user_device[['use_id',
'platform', 'device']], on = 'use_id', how = 'left')
```

# Pandas

## Data Management using Pandas



### MERGE DATA

- **Right Merge**
- Keep all the rows in the right data frame using the right parameter for the method how.

```
# Right
result_right = pd.merge(user_usage, user_device[['use_id',
'platform', 'device']], on = 'use_id', how = 'right')
```

# Pandas

## Data Management using Pandas



### MERGE DATA

- Outer Merge
- The “outer” merge combines all the rows for left and right dataframes with NaN when there are no matched values in the rows.

```
# Outer
result_outer = pd.merge(user_usage, user_device[['use_id',
'platform', 'device']], on = 'use_id', how = 'outer')
```

# Pandas

## Data Management using Pandas



### Concatenating objects

- The concat() function (in the main pandas namespace) does all of the heavy lifting of performing concatenation operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.
- `pd.concat(objs, axis=0, join="outer", ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, copy=True)`

<https://pandas.pydata.org/docs/reference/api/pandas.concat.html>

# Pandas

## Data Management using Pandas



### pd.concat vs pd.merge

- Concat function concatenates data frames along rows or columns. We can think of it as stacking up multiple data frames. Merge combines data frames based on values in shared columns. Merge function offers more flexibility compared to concat function because it allows combinations based on a condition.

<https://pandas.pydata.org/docs/reference/api/pandas.concat.html>



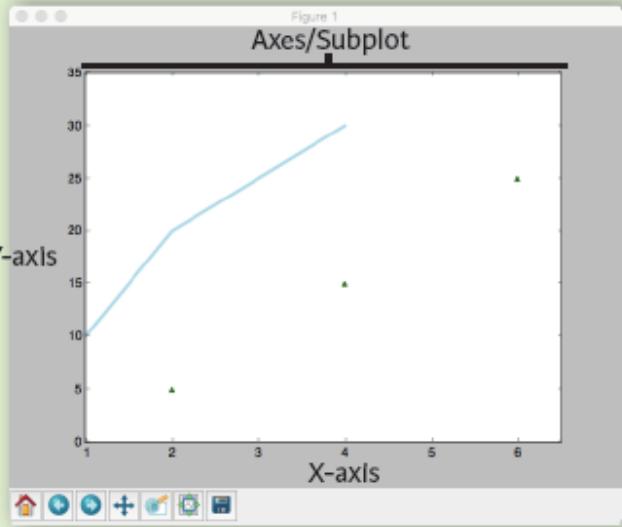
## Matplotlib and Seaborn Libraries

# Matplotlib

# matplotlib

## Plot Anatomy & Workflow

### Plot Anatomy



### Workflow

The basic steps to creating plots with matplotlib are:

- 1 Prepare data
- 2 Create plot
- 3 Plot
- 4 Customize plot
- 5 Save plot
- 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                                Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                            Step 2
>>> ax = fig.add_subplot(111)                      Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)  Step 3, 4
>>> ax.scatter([2,4,6],
              [5,15,25],
              color='darkgreen',
              marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')                         Step 6
>>> plt.show()
```

Figure

# Matplotlib



- Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

## 2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

### Figure

```
>>> fig = plt.figure()  
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

### Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()  
>>> ax1 = fig.add_subplot(221) # row-col-num  
>>> ax3 = fig.add_subplot(212)  
>>> fig3, axes = plt.subplots(nrows=2, ncols=2)  
>>> fig4, axes2 = plt.subplots(ncols=3)
```

# Matplotlib

# matplotlib

## 3 Plotting Routines

### 1D Data

```
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
>>> axes[0,0].bar([1,2,3],[3,4,5])
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them  
 Draw unconnected points, scaled or colored  
 Plot vertical rectangles (constant width)  
 Plot horizontal rectangles (constant height)  
 Draw a horizontal line across axes  
 Draw a vertical line across axes  
 Draw filled polygons  
 Fill between y-values and 0

### 2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,
                  cmap='gist_earth',
                  interpolation='nearest',
                  vmin=-2,
                  vmax=2)
```

Colormapped or RGB arrays

### Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)
>>> axes[1,1].quiver(y,z)
>>> axes[0,1].streamplot(X,Y,U,V)
```

Add an arrow to the axes  
 Plot a 2D field of arrows  
 Plot 2D vector fields

### Data Distributions

```
>>> ax1.hist(y)
>>> ax3.boxplot(y)
>>> ax3.violinplot(z)
```

Plot a histogram  
 Make a box and whisker plot  
 Make a violin plot

```
>>> axes2[0].pcolor(data2)
>>> axes2[0].pcolormesh(data)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2]= ax.clabel(CS)
```

Pseudocolor plot of 2D array  
 Pseudocolor plot of 2D array  
 Plot contours  
 Plot filled contours  
 Label a contour plot

# Matplotlib

# matplotlib

## 4 Customize Plot

### Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                  cmap='seismic')
```

### Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

### LineStyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

### Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowsyle="->",
                               connectionstyle="arc3"),)
```

### Mathtext

```
>>> plt.title(r'$\sigma_i=15$', fontsize=20)
```

### Limits, Legends & Layouts

#### Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])
>>> ax.set_xlim(0,10.5)
```

#### Legends

```
>>> ax.set(title='An Example Axes',
            ylabel='Y-Axis',
            xlabel='X-Axis')
>>> ax.legend(loc='best')
```

#### Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
                  ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',
                  direction='inout',
                  length=10)
```

#### Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,
                        hspace=0.3,
                        left=0.125,
                        right=0.9,
                        top=0.9,
                        bottom=0.1)
>>> fig.tight_layout()
```

#### Axis Spines

```
>>> ax1.spines['top'].set_visible(False)
>>> ax1.spines['bottom'].set_position(('outward',10))
```

Add padding to a plot

Set the aspect ratio of the plot to 1

Set limits for x-and y-axis

Set limits for x-axis

Set a title and x-and y-axis labels

No overlapping plot elements

Manually set x-ticks

Make y-ticks longer and go in and out

Adjust the spacing between subplots

Fit subplot(s) in to the figure area

Make the top axis line for a plot invisible

Move the bottom axis line outward

## 5 Save Plot

### Save figures

```
>>> plt.savefig('foo.png')
```

### Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

## 6 Show Plot

```
>>> plt.show()
```

## Close & Clear

```
>>> plt.cla()
```

```
>>> plt.clf()
```

```
>>> plt.close()
```

Clear an axis

Clear the entire figure

Close a window

# Seaborn

seaborn

## Statistical Data Visualization With Seaborn

The Python visualization library Seaborn is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")
>>> sns.set_style("whitegrid")           Step 1
>>> g = sns.lmplot(x="tip",
                    y="total_bill",
                    data=tips,
                    aspect=2)                         Step 2
>>> g = (g.set_axis_labels("Tip", "Total bill(USD)") .
        set(xlim=(0,10), ylim=(0,100)))      Step 3
>>> plt.title("title")                  Step 4
>>> plt.show(g)                       Step 5
```

## 1 Data

Also see [Lists, NumPy & Pandas](#)

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({ 'x':np.arange(1,101),
                           'y':np.random.normal(0,4,100) })
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

## 2 Figure Aesthetics

Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5, 6))
```

Create a figure and one subplot

### Seaborn styles

```
>>> sns.set()
>>> sns.set_style("whitegrid")
>>> sns.set_style("ticks",
                 {"xtick.major.size":8,
                  "ytick.major.size":8})
>>> sns.axes_style("whitegrid")
```

(Re)set the seaborn default  
Set the matplotlib parameters  
Set the matplotlib parameters  
Return a dict of params or use with  
with to temporarily set the style

### Context Functions

```
>>> sns.set_context("talk")
>>> sns.set_context("notebook",
                   font_scale=1.5,
                   rc={"lines.linewidth":2.5})
```

Set context to "talk"  
Set context to "notebook",  
scale font elements and  
override param mapping

### Color Palette

```
>>> sns.set_palette("husl", 3)
>>> sns.color_palette("husl")
>>> flatui = ["#9b59b6", "#3498db", "#95a5a6", "#e74c3c", "#34495e", "#2ecc71"]
>>> sns.set_palette(flatui)
```

Define the color palette  
Use with `with` to temporarily set palette  
Set your own color palette

## 3) Plotting With Seaborn

### Axis Grids

```
>>> g = sns.FacetGrid(titanic,
                      col="survived",
                      row="sex")
>>> g = g.map(plt.hist, "age")
>>> sns.factorplot(x="pclass",
                     y="survived",
                     hue="sex",
                     data=titanic)
>>> sns.lmplot(x="sepal_width",
                 y="sepal_length",
                 hue="species",
                 data=iris)
```

Subplot grid for plotting conditional relationships

Draw a categorical plot onto a Facetgrid

Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris)
>>> i = sns.JointGrid(x="x",
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
                  sns.distplot)
>>> sns.jointplot("sepal_length",
                  "sepal_width",
                  data=iris,
                  kind='kde')
```

Subplot grid for plotting pairwise relationships  
Plot pairwise bivariate distributions  
Grid for bivariate plot with marginal univariate plots

Plot bivariate distribution

# Seaborn

seaborn

## Categorical Plots

### Scatterplot

```
>>> sns.stripplot(x="species",
                   y="petal_length",
                   data=iris)
>>> sns.swarmplot(x="species",
                   y="petal_length",
                   data=iris)
```

### Bar Chart

```
>>> sns.barplot(x="sex",
                 y="survived",
                 hue="class",
                 data=titanic)
```

### Count Plot

```
>>> sns.countplot(x="deck",
                   data=titanic,
                   palette="Greens_d")
```

### Point Plot

```
>>> sns.pointplot(x="class",
                   y="survived",
                   hue="sex",
                   data=titanic,
                   palette={"male":"g",
                            "female":"m"},
                   markers=["^", "o"],
                   linestyles=["-", "--"])
```

### Boxplot

```
>>> sns.boxplot(x="alive",
                 y="age",
                 hue="adult_male",
                 data=titanic)
>>> sns.boxplot(data=iris,orient="h")
```

### Violinplot

```
>>> sns.violinplot(x="age",
                   y="sex",
                   hue="survived",
                   data=titanic)
```

Scatterplot with one categorical variable

Categorical scatterplot with non-overlapping points

Show point estimates and confidence intervals with scatterplot glyphs

Show count of observations

Show point estimates and confidence intervals as rectangular bars

Boxplot

Boxplot with wide-form data

Violin plot

## Regression Plots

```
>>> sns.regplot(x="sepal_width",
                 y="sepal_length",
                 data=iris,
                 ax=ax)
```

Plot data and a linear regression model fit

## Distribution Plots

```
>>> plot = sns.distplot(data.y,
                        kde=False,
                        color="b")
```

Plot univariate distribution

## Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1)
```

Heatmap

# Seaborn

seaborn

## 4 Further Customizations

[Also see Matplotlib](#)

### Axisgrid Objects

```
>>> g.despine(left=True)
>>> g.set_ylabels("Survived")
>>> g.set_xticklabels(rotation=45)
>>> g.set_axis_labels("Survived",
                      "Sex")
>>> h.set(xlim=(0,5),
          ylim=(0,5),
          xticks=[0,2.5,5],
          yticks=[0,2.5,5])
```

Remove left spine  
Set the labels of the y-axis  
Set the tick labels for x  
Set the axis labels  
Set the limit and ticks of the x-and y-axis

### Plot

```
>>> plt.title("A Title")
>>> plt.ylabel("Survived")
>>> plt.xlabel("Sex")
>>> plt.ylim(0,100)
>>> plt.xlim(0,10)
>>> plt.setp(ax, yticks=[0,5])
>>> plt.tight_layout()
```

Add plot title  
Adjust the label of the y-axis  
Adjust the label of the x-axis  
Adjust the limits of the y-axis  
Adjust the limits of the x-axis  
Adjust a plot property  
Adjust subplot params

## 5 Show or Save Plot

[Also see Matplotlib](#)

```
>>> plt.show()
>>> plt.savefig("foo.png")
>>> plt.savefig("foo.png",
               transparent=True)
```

Show the plot  
Save the plot as a figure  
Save transparent figure

### Close & Clear

[Also see Matplotlib](#)

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```

Clear an axis  
Clear an entire figure  
Close a window



Scikit Learn Library



# Scikit Learn

## Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

### A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```



## Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10, 5))
>>> y = np.array(['M', 'M', 'F', 'F', 'M', 'F', 'M', 'M', 'F', 'F', 'F'])
>>> X[X < 0.7] = 0
```

## Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
y,
random_state=0)
```

## Preprocessing The Data

### Standardization

```
>>> from sklearn.preprocessing import StandardScaler  
>>> scaler = StandardScaler().fit(X_train)  
>>> standardized_X = scaler.transform(X_train)  
>>> standardized_X_test = scaler.transform(X_test)
```

### Normalization

```
>>> from sklearn.preprocessing import Normalizer  
>>> scaler = Normalizer().fit(X_train)  
>>> normalized_X = scaler.transform(X_train)  
>>> normalized_X_test = scaler.transform(X_test)
```

### Binarization

```
>>> from sklearn.preprocessing import Binarizer  
>>> binarizer = Binarizer(threshold=0.0).fit(X)  
>>> binary_X = binarizer.transform(X)
```

### Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder  
>>> enc = LabelEncoder()  
>>> y = enc.fit_transform(y)
```

### Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer  
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)  
>>> imp.fit_transform(X_train)
```

### Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures  
>>> poly = PolynomialFeatures(5)  
>>> poly.fit_transform(X)
```

# Scikit Learn



## Create Your Model

### Supervised Learning Estimators

#### Linear Regression

```
>>> from sklearn.linear_model import LinearRegression  
>>> lr = LinearRegression(normalize=True)
```

#### Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC  
>>> svc = SVC(kernel='linear')
```

#### Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB  
>>> gnb = GaussianNB()
```

#### KNN

```
>>> from sklearn import neighbors  
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

### Unsupervised Learning Estimators

#### Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA  
>>> pca = PCA(n_components=0.95)
```

#### K Means

```
>>> from sklearn.cluster import KMeans  
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

### Model Fitting

#### Supervised learning

```
>>> lr.fit(X, y)  
>>> knn.fit(X_train, y_train)  
>>> svc.fit(X_train, y_train)
```

#### Unsupervised Learning

```
>>> k_means.fit(X_train)  
>>> pca_model = pca.fit_transform(X_train)
```

Fit the model to the data

Fit the model to the data

Fit to data, then transform it

### Prediction

#### Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))  
>>> y_pred = lr.predict(X_test)  
>>> y_pred = knn.predict_proba(X_test)
```

#### Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test)
```

Predict labels

Predict labels

Estimate probability of a label

Predict labels in clustering algos

## Classification Metrics

### Accuracy Score

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Estimator score method  
Metric scoring functions

### Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Precision, recall, f1-score  
and support

### Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

## Regression Metrics

### Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

### Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

### R<sup>2</sup> Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

## Clustering Metrics

### Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

### Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

### V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

## Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

## Tune Your Model

### Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

### Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```



TensorFlow – Keras Library



# TensorFlow - Keras



## Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

### A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

## Data

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

### Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                           mnist,
                           cifar10,
                           imdb
>>> (x_train,y_train), (x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2), (x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3), (x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4), (x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

### Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"), delimiter=",")
>>> X = data[:,0:8]
>>> y = data[:,8]
```

## Preprocessing

### Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

### One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Also see NumPy & Scikit-Learn

### Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
...                                                    y,
...                                                    test_size=0.33,
...                                                    random_state=42)
```

### Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

## Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

## Multilayer Perceptron (MLP)

### Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

### Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

### Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

## Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

## Recurrent Neural Network (RNN)

```
>>> from keras.layers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

# TensorFlow - Keras



## Inspect Model

```
>>> model.output_shape  
>>> model.summary()  
>>> model.get_config()  
>>> model.get_weights()
```

Model output shape  
Model summary representation  
Model configuration  
List all weight tensors in the model

## Compile Model

### MLP: Binary Classification

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

### MLP: Multi-Class Classification

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

### MLP: Regression

```
>>> model.compile(optimizer='rmsprop',  
                  loss='mse',  
                  metrics=['mae'])
```

### Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',  
                  optimizer='adam',  
                  metrics=['accuracy'])
```

## Model Training

```
>>> model3.fit(x_train4,  
              y_train4,  
              batch_size=32,  
              epochs=15,  
              verbose=1,  
              validation_data=(x_test4,y_test4))
```

## Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,  
                           y_test,  
                           batch_size=32)
```

## Prediction

```
>>> model3.predict(x_test4, batch_size=32)  
>>> model3.predict_classes(x_test4,batch_size=32)
```

## Save/ Reload Models

```
>>> from keras.models import load_model  
>>> model3.save('model_file.h5')  
>>> my_model = load_model('my_model.h5')
```

## Model Fine-tuning

### Optimization Parameters

```
>>> from keras.optimizers import RMSprop  
>>> opt = RMSprop(lr=0.0001, decay=1e-6)  
>>> model2.compile(loss='categorical_crossentropy',  
                    optimizer=opt,  
                    metrics=['accuracy'])
```

### Early Stopping

```
>>> from keras.callbacks import EarlyStopping  
>>> early_stopping_monitor = EarlyStopping(patience=2)  
>>> model3.fit(x_train4,  
                y_train4,  
                batch_size=32,  
                epochs=15,  
                validation_data=(x_test4,y_test4),  
                callbacks=[early_stopping_monitor])
```

# Review Questions

- What are python-based libraries suitable for a data science project?
- Explore different functions to handle dataset.
- Explore the functions available for data preprocessing.

# Summary / Recap of Main Points

- Python is a widely used programming language for data science.
- It is Readable and Simple Syntax, Interpreted Language, Dynamically Typed, Extensive Standard Library, Cross-Platform, and Open-Source programming language.
- Various IDEs available.
- Wide range of libraries available for an effective data science project.

# What To Expect Next Week

## In Class

- Introduction to Supervised Learning

## Preparation for Class

- Introduction to Supervised Learning

CX016-2.5-3-IML - Introduction to Machine Learning

## **Introduction to Supervised Learning**

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understand Supervised Learning
2. Understand Types of Supervised Learning
3. Applications of Supervised Learning
4. Steps in Supervised Learning Process

# Contents & Structure

Supervised Learning.

Types of Supervised Learning Algorithms

Applications of Supervised Learning

Steps in Supervised Learning Process

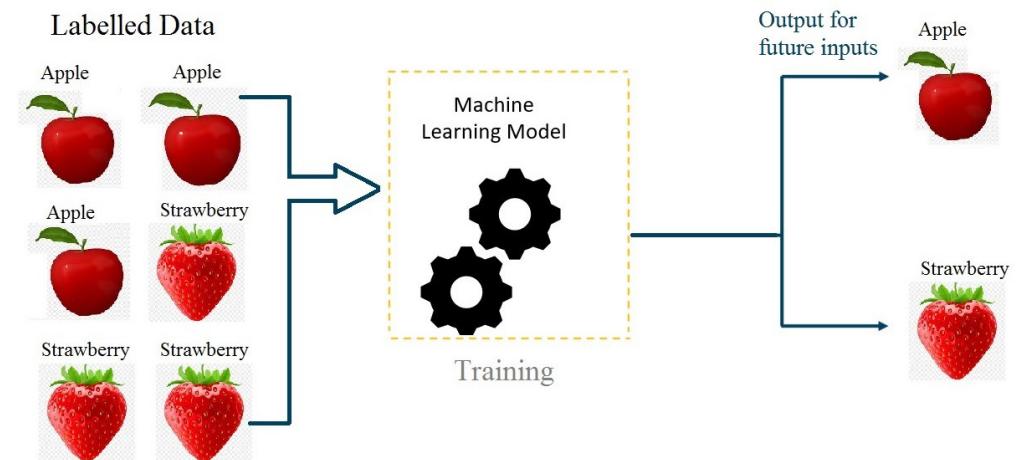
# Recap From Last Lesson

- What are python-based libraries suitable for a data science project?
- Explore different functions to handle dataset.
- Explore the functions available for data preprocessing.

# Supervised Learning

- A type of machine learning where the model is trained on labeled data.
- It learns a mapping from inputs to outputs.
- Key Components:
  - Inputs (Features): Data used to make predictions.
  - Outputs (Labels): The desired prediction outcomes.
  - Example:
    - Predicting house prices based on features like size, location, etc.
    - Classify Apples and Strawberries

**It's like teaching a child to recognize animals by showing them pictures of dogs, cats, and birds with their names written below.**

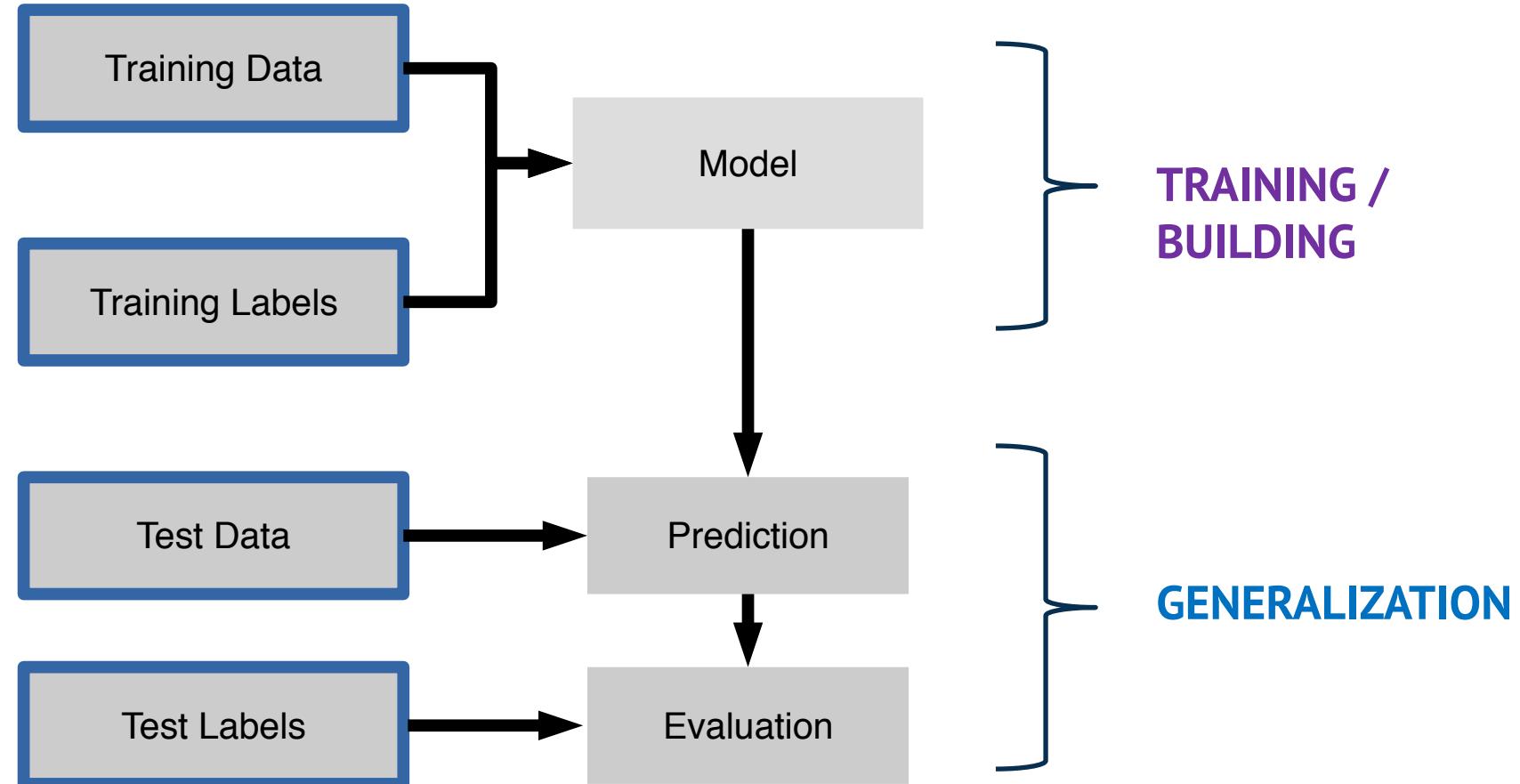


# Supervised Learning

**Supervised Learning** (also called **Predictive**): Predict an unknown value(s) of a variable(s) from the values of some attributes

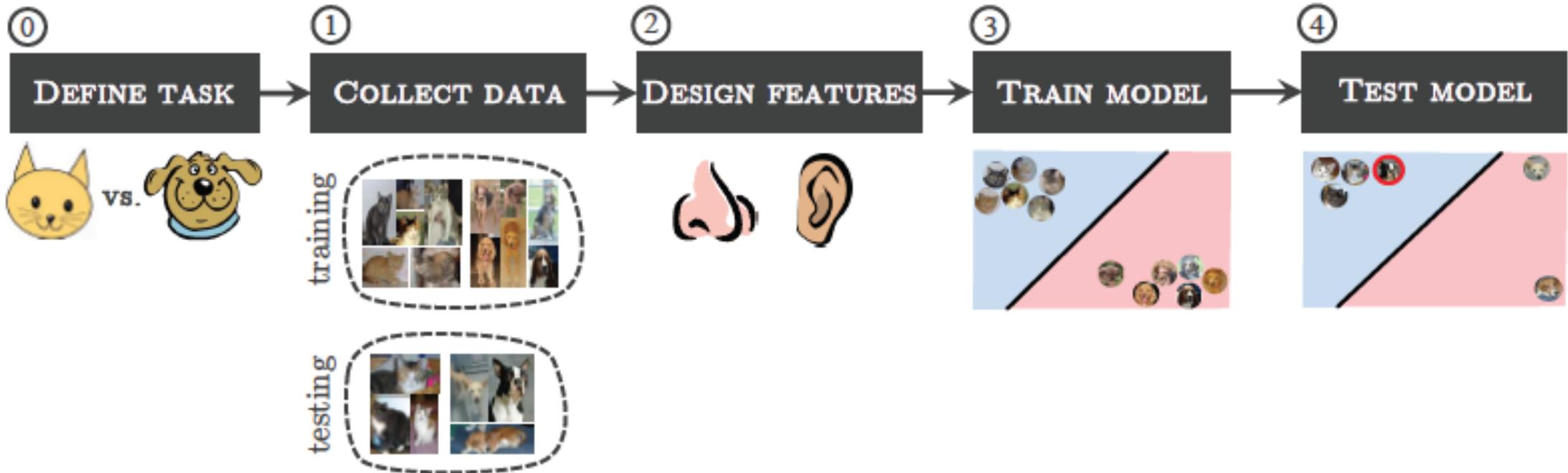
- **Classification:** predict the type/class of new cases
  - ✓ Spam Filtering, Handwriting Character Recognition, Patient Diagnosis
  - ✓ Example ML Algorithms: K-Nearest Neighbors
- **Regression:** predict a numerical value of new cases
  - ✓ Blood Pressure, Sales Amounts
  - ✓ Examples: Linear Regression
- **Supervised Anomaly Detection:** identify items, events or observations deviating from expected patterns using data labeled as "normal" and "abnormal" (involves training a classifier)
  - ✓ Fraud Detection

# Supervised Learning



Fit model on the new data after evaluation

# Supervised Learning



The learning pipeline of the cat versus dog classification problem.  
The same general pipeline is used for essentially all machine learning problems.

# Key Concepts and Terminologies

- Training Data: The dataset used to train the model.
- Test Data: The dataset used to evaluate the model.
- Features: Input variables.
- Labels: Output variables.
- Model: The mathematical representation of the learning process.
- Overfitting: When the model learns noise and details in the training data to an extent that it negatively impacts performance on new data.
- Underfitting: When the model is too simple to capture the underlying patterns in the data.

# Applications of Supervised Learning

- Spam Detection: Classifying emails as spam or not spam.
- Medical Diagnosis: Predicting diseases based on patient data.
- Sentiment Analysis: Analyzing customer reviews to determine sentiment.
- Fraud Detection: Identifying fraudulent transactions.
- Image Classification: Recognizing objects in images.

# Steps in Supervised Learning Process

- Data Collection: Gather and collect data.
- Data Preprocessing: Clean and prepare the data.
- Feature Selection: Identify and select relevant features.
- Model Selection: Choose an appropriate algorithm.
- Training: Train the model on the training data.
- Evaluation: Evaluate the model using test data.
- Hyperparameter Tuning: Optimize the model parameters.
- Deployment: Implement the model in a real-world environment.

# Evaluation Metrics

- Regression Metrics:
  - Mean Absolute Error (MAE)
  - Mean Squared Error (MSE)
  - R-squared ( $R^2$ )
- Classification Metrics:
  - Accuracy
  - Precision
  - Recall
  - F1 Score
  - Confusion Matrix

# Challenges and Limitations

- Data Quality: Quality of data significantly affects model performance.
- Bias and Variance: Balancing underfitting and overfitting.
- Computational Cost: Training models can be resource-intensive.
- Interpretability: Some models are complex and hard to interpret.
- Ethical Concerns: Ensuring fair and unbiased predictions.

# Review Questions

1. What is Supervised Learning?
2. What are types of Supervised Learning?
3. What are the applications of Supervised Learning?
4. What are the steps in Supervised Learning Process?

# Summary / Recap of Main Points

- Supervised learning is a type of machine learning where the model is trained on labeled data, meaning the input data is paired with the correct output, with the goal of learning a mapping from inputs to outputs to predict the output for new, unseen data.
- There are two main types of supervised learning algorithms: classification, which is used to predict discrete labels such as spam or not spam, and regression, which is used to predict continuous values such as the price of a house.
- Supervised learning is used in various applications such as email filtering, fraud detection, medical diagnosis, and stock price prediction, and it is essential in areas where labeled data is available and predictions need to be made based on historical data.

# What To Expect Next Week

## In Class

- K-Nearest Neighbors (KNN)

## Preparation for Class

- K-Nearest Neighbors (KNN)

CX016-2.5-3-IML - Introduction to Machine Learning

## **K-Nearest Neighbors (KNN)**

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understand KNN Algorithm
2. Explain the characteristics of KNN Algorithm
3. List out the pros and cons of KNN Algorithm
4. Implement KNN Algorithm using Python

# Contents & Structure

KNN Algorithm

Characteristics of KNN Algorithm

Pros and cons of KNN Algorithm

Implement KNN Algorithm using Python

# Recap From Last Lesson

- What is Supervised Learning?
- What are types of Supervised Learning?
- What are the applications of Supervised Learning?
- What are the steps in Supervised Learning Process?

# K-Nearest Neighbors (KNN)

- K-Nearest Neighbors (KNN) is a simple and effective algorithm used for classification and regression tasks in machine learning.
- This algorithm is used to solve the classification model problems. K-nearest neighbor or K-NN algorithm basically creates an imaginary boundary to classify the data.
- When new data points come in, the algorithm will try to predict that to the nearest of the boundary line.
- Therefore, larger k value means smoother curves of separation resulting in less complex models. Whereas smaller k value tends to overfit the data and resulting in complex models.
- **Note:** It's very important to have the right k-value when analyzing the dataset to avoid overfitting and underfitting of the dataset.

# K-Nearest Neighbors (KNN)

## Characteristics

- Lazy Learning: KNN is considered a lazy learning algorithm because it doesn't learn a discriminative function from the training data but rather memorizes the training dataset.
- Instance-Based: Predictions are made based on specific instances of the data rather than a generalized model.
- Non-Parametric: KNN makes no assumptions about the underlying data distribution.

# K-Nearest Neighbors (KNN)

## Characteristics

Choosing K involves balancing these two errors:

- Small K: High variance, low bias (overfitting).
- Large K: Low variance, high bias (underfitting).

The best K is where the sum of bias and variance is minimized, which results in the smallest overall prediction error.

# K-Nearest Neighbors (KNN)

## Pros and Cons

### Pros:

- Simple to understand and implement.
- No training phase (besides storing the training data), making it quick to set up.
- Versatile, can be used for both classification and regression.

### Cons:

- Computationally expensive during prediction, especially with large datasets.
- Performance depends heavily on the choice of  $k$  and the distance metric.
- Sensitive to irrelevant or redundant features.
- Can be affected by the curse of dimensionality.

# K-Nearest Neighbors (KNN)

## Use Cases

### Classification:

- Handwriting recognition
- Image recognition
- Recommendation systems

### Regression:

- Predicting house prices
- Stock prices

# K-Nearest Neighbors (KNN)

## How KNN Works

- **Training Phase:** The algorithm doesn't explicitly train a model. Instead, it simply stores the training data points.
- **Prediction Phase:**
  - Classification:
    - Choose a value for  $k$ , the number of nearest neighbors.
    - Calculate the distance (using Euclidean, Manhattan, or other distance metrics) between the query point and all points in the training set.
    - Identify the  $k$  nearest neighbors to the query point.
    - Assign the class label that is most common among these  $k$  neighbors to the query point.
  - Regression:
    - The same steps as classification are followed, but instead of voting for a class label, the average of the target values of the  $k$  nearest neighbors is taken as the predicted value.

# K-Nearest Neighbors (KNN)

## Detailed Steps of KNN:

### Data Preparation:

- **Dataset:** Prepare a dataset consisting of input features and their corresponding labels.
- **Normalization:** Optionally, normalize or standardize the data to ensure that features with larger ranges do not dominate the distance calculations.

### Choosing $k$ :

- The value of  $k$  (the number of nearest neighbors) is selected. This is a crucial hyperparameter that can be chosen using methods like cross-validation.

# K-Nearest Neighbors (KNN)

## Detailed Steps of KNN:

### Distance Metric:

- Decide on a distance metric to measure the similarity between instances. Common choices include:
  - Euclidean Distance:**  $\text{Euclidean}(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$
  - Manhattan Distance:**  $\text{Manhattan}(x, y) = \sum_{i=1}^n |x_i - y_i|$
  - Minkowski Distance:** Generalized form that includes Euclidean and Manhattan distances.

$$D(x, y) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

where:

- $D(x, y)$  is the distance between points  $x$  and  $y$ .
- $x_i$  and  $y_i$  are the  $i$ -th coordinates of points  $x$  and  $y$ .
- $n$  is the number of dimensions (features).
- $p$  is the order parameter.

# K-Nearest Neighbors (KNN)

## Detailed Steps of KNN:

### Prediction Phase:

- **Classification:**
  - **Calculate Distance:** For a new input instance (query point), calculate the distance between this instance and all the points in the training dataset.
  - **Find Nearest Neighbors:** Identify the  $k$  training instances that are closest to the query point based on the chosen distance metric.
  - **Voting:** Among these  $k$  nearest neighbors, count the occurrences of each class label. The class label with the highest count (majority vote) is assigned to the query point.

### Prediction Phase:

- **Regression:**
  - **Calculate Distance:** Calculate the distance between the query point and all points in the training dataset.
  - **Find Nearest Neighbors:** Identify the  $k$  nearest neighbors.
  - **Average:** Compute the average of the target values of these  $k$  nearest neighbors. This average is the predicted value for the query point.

# K-Nearest Neighbors (KNN)

## Calculation

### Dataset:

Feature 1	Feature 2	TV (Class)
2	3	A
1	1	B
4	2	A
6	5	B
7	8	B

### Query Point:

- Need to classify a new data point:  
**(5,4)**

### Euclidean Distance

#### Calculate Distance:

- Distance between (5, 4) and (2, 3):  $\sqrt{(5 - 2)^2 + (4 - 3)^2} = \sqrt{9 + 1} = \sqrt{10} \approx 3.16$
- Distance between (5, 4) and (1, 1):  $\sqrt{(5 - 1)^2 + (4 - 1)^2} = \sqrt{16 + 9} = \sqrt{25} = 5$
- Distance between (5, 4) and (4, 2):  $\sqrt{(5 - 4)^2 + (4 - 2)^2} = \sqrt{1 + 4} = \sqrt{5} \approx 2.24$
- Distance between (5, 4) and (6, 5):  $\sqrt{(5 - 6)^2 + (4 - 5)^2} = \sqrt{1 + 1} = \sqrt{2} \approx 1.41$
- Distance between (5, 4) and (7, 8):  $\sqrt{(5 - 7)^2 + (4 - 8)^2} = \sqrt{4 + 16} = \sqrt{20} \approx 4.47$

### Find Nearest Neighbors:

- Let's choose  $k = 3$ . The three nearest neighbors to (5, 4) are:
  - (6, 5) with distance 1.41 (Label B)
  - (4, 2) with distance 2.24 (Label A)
  - (2, 3) with distance 3.16 (Label A)

#### Voting:

- Among the 3 nearest neighbors, labels are: B, A, A.
- Majority label is A.

### Result:

The new point (5, 4) is classified as A.

# K-Nearest Neighbors (KNN)

## Python Implementation - Classification

### Import library

```
from sklearn.neighbors import KNeighborsClassifier
```

### Perform required Data Preprocessing

### Data Split

### Implementation

```
knn = KNeighborsClassifier()  
knn.fit(X_train, y_train)  
y_pred = knn.predict(X_test)  
acc = accuracy_score(y_test, y_pred)*100  
print("KNN - Accuracy: {:.3f}.".format(acc))
```

**Conduct HP Tuning to improve the performance and to get reliable model**

# K-Nearest Neighbors (KNN)

## Python Implementation - Regression

### Import library

```
from sklearn.neighbors import KNeighborsRegressor
```

### Perform required Data Preprocessing

### Data Split

### Implementation

```
knn = KNeighborsRegressor ()  
knn.fit(X_train, y_train)  
y_pred = knn.predict(X_test)  
print("Mean Squared Error: ", mean_squared_error(y_test, y_pred).round(2))  
print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y_test, y_pred)).round(2))  
print('Mean Absolute Error: ', mean_absolute_error(y_test, y_pred).round(2))
```

**Conduct HP Tuning to improve the performance and to get reliable model**

# Review Questions

1. What is KNN Algorithm?
2. What are the characteristics of KNN Algorithm?
3. What are the pros and cons of KNN Algorithm?
4. How to implement KNN Algorithm using Python?

# Summary / Recap of Main Points

- Fundamentals of the KNN algorithm, including its core concepts, how it works, and its underlying logic.
- Exploring the key properties and features of the KNN algorithm, such as its simplicity, non-parametric nature, and reliance on distance metrics.
- Advantages and disadvantages of using the KNN algorithm, including its ease of implementation, sensitivity to noise, and computational efficiency.

# What To Expect Next Week

## In Class

- K-Nearest Neighbors (KNN)

## Preparation for Class

- K-Nearest Neighbors (KNN)

CX016-2.5-3 - Introduction to Machine Learning

# Regression

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understand the idea of regression
2. Recognize various types of linear regression
3. Perform numeric prediction

# Contents & Structure

Simple Linear Regression

Multiple Linear Regression

Polynomial Linear Regression

Evaluation – R Coefficient

Evaluation - Statistics

Implement Linear Regression using Python

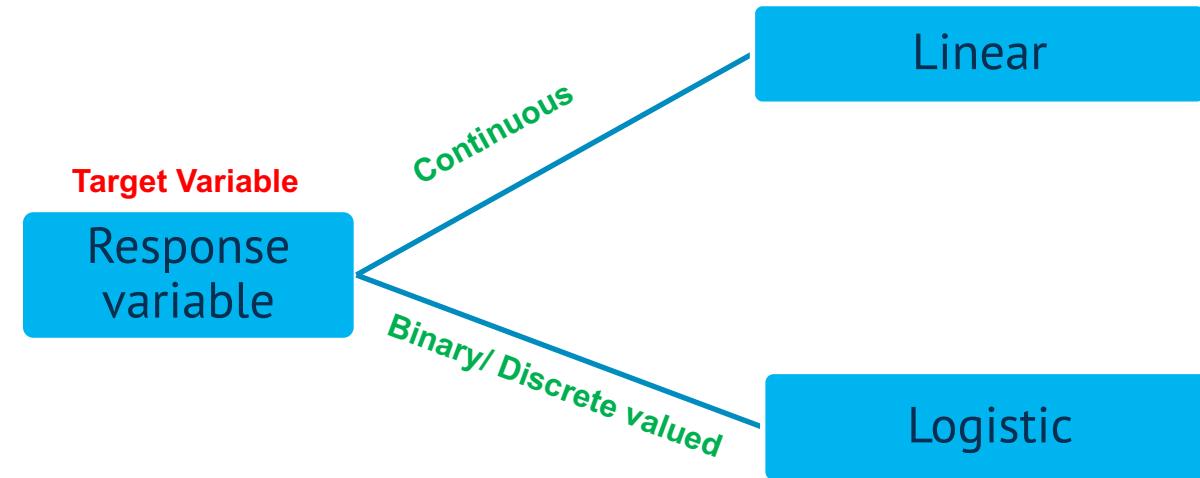
# Recap From Last Lesson

- What is KNN Algorithm?
- What are the characteristics of KNN Algorithm?
- What are the pros and cons of KNN Algorithm?
- How to implement KNN Algorithm using Python?

# Regression

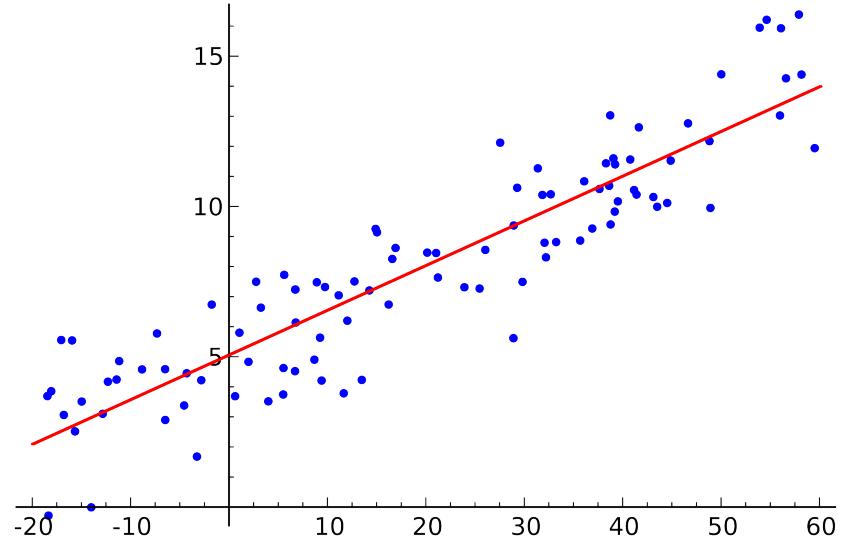
## Regression:

- Regression is a technique used to predict the value of a response (dependent) variable, from one or more predictor (independent) variables, where the variable is either categorical or numeric.
  - Estimation
    - Linear Regression
  - Classification/Decision
    - Logistic Regression

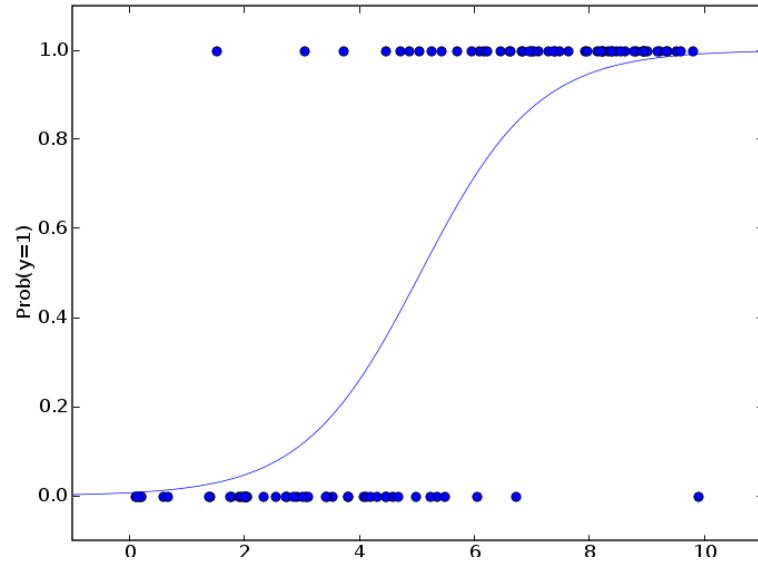


# Regression

Linear



Logistic



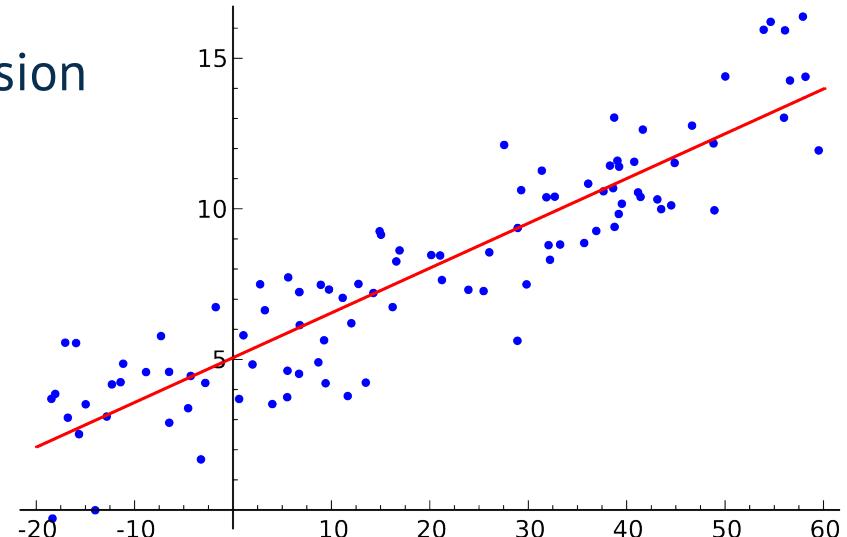
# Linear Regression



# Linear Regression

## Linear Regression:

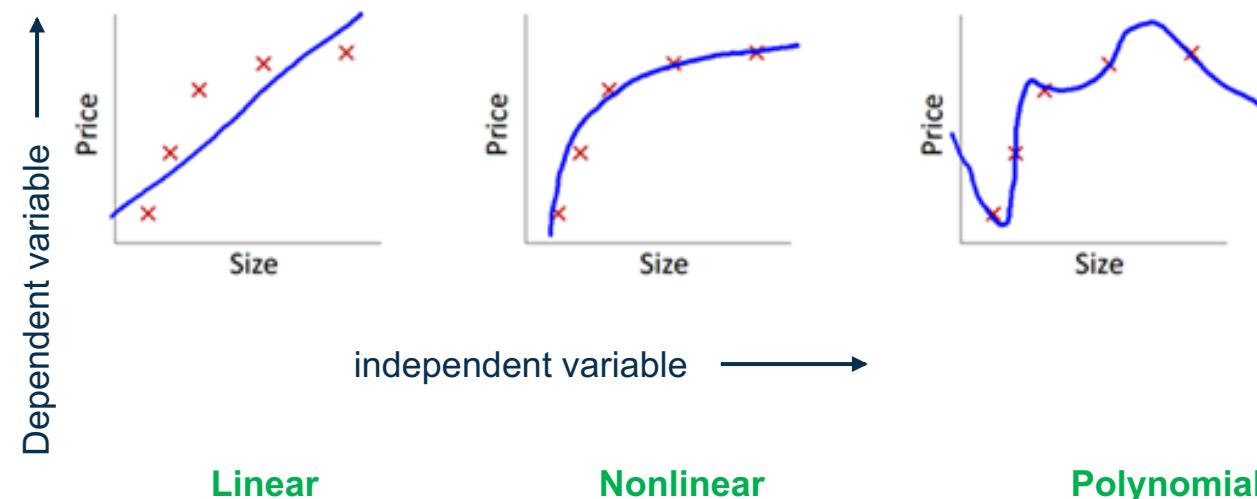
- Regression is a technique used to predict the value of a response (dependent) variable, from one or more predictor (independent) variables, where the variable is numeric.
  - Estimation
    - Linear Regression



# Linear Regression

## Types of Linear Regression:

- Regression (linear and polynomial):
  - for Estimation (prediction)



Linear

Nonlinear

Polynomial

polynomial regression is a form of regression analysis in which the relationship between the independent variable  $x$  and the dependent variable  $y$  is modelled as an  $n$ th degree polynomial

# Linear Regression

## Variations in Linear Regression

- Independent variable (input / predictor variable) -  $x$
- Dependent variable (output variable/response variable) -  $y$ 
  - Simple Linear Regression - assumes an input  $x$  – a single variable ; one output
  - Multiple linear regression assumes an input vector  $x=(x_1, x_2, \dots x_n)$
  - Multivariate linear regression assumes an output vector  $y=(y_1, y_2, \dots y_n)$

# Linear Regression

## Simple Linear Regression

Analytical solution to linear regression with 2 variables:

**Regression Model →  $y = w_0 + w_1x$**

$$\text{RSS} = \sum_{i=1}^N (y_i - (w_0 + w_1 x_i))^2$$

**RSS** → Residual sum of squares, also known as the sum of squared residuals or the sum of squared estimate of errors. It is a measure of the discrepancy between the data and an estimation model.

**GLM – generalized Linear Regression Model – No assumption of Gaussian distribution**

# Linear Regression

## Multiple Linear Regression

Multiple Linear Regression with multiple inputs and single output:

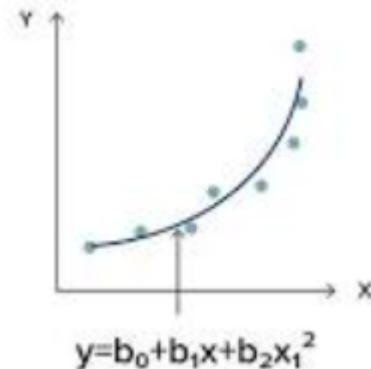
$$\text{Regression Model} \rightarrow y = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$$

Mathematical solution to finding multiple linear regression parameters/ weights requires matrix inversion, etc.

# Linear Regression

## Polynomial regression

- Is a form of regression analysis where the relationship between the independent variable (x) and dependent variable (y) is modeled as an nth degree polynomial.
- Unlike linear regression, which fits a straight line to the data, polynomial regression can capture more complex patterns by fitting a curve.
- When the relationship between variables is not linear, polynomial regression can effectively model the curve.



# Linear Regression

## Assumptions of Linear Regression

- Linearity: The relationship between the dependent and independent variables is linear.
- Independence: The residuals (errors) are independent.
- Homoscedasticity: The residuals have constant variance.
- Normality: The residuals of the model are normally distributed.

# Linear Regression

## Assessing the Model

The least squares method will produce a regression line whether there is a linear relationship between  $x$  and  $y$ .

Consequently, it is important to assess how well the linear model fits the data.

# Linear Regression

## Coefficient of Determination - R<sup>2</sup>

- The **coefficient of determination (R<sup>2</sup>)** is a key output of regression analysis.
- **R-squared** is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination.
- It is interpreted as the proportion of the variance in the dependent variable that is predictable from the independent variable.

**R-squared = Explained variation / Total variation**

$$R^2 = 1 - \frac{RSS}{\sum (y_i - \bar{y})^2}$$

# Linear Regression

## Coefficient of Determination - $R^2$

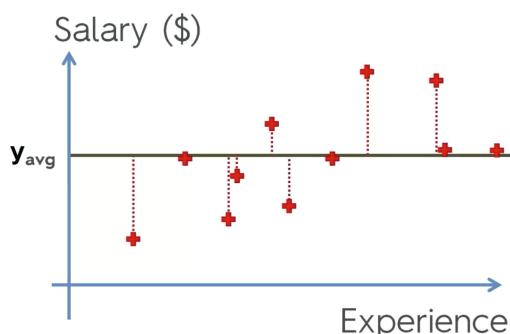
- $0 \leq R^2 \leq 1 \rightarrow$  Range of values
- $R^2 = 0 \rightarrow$  means that the dependent variable cannot be predicted from the independent variable.
- $R^2 = 1 \rightarrow$  means the dependent variable can be predicted without error from the independent variable.
- $0 < R^2 < 1$  indicates the extent to which the dependent variable is predictable.
- An  $R^2$  of 0.10 means that 10 percent of the variance in Y is predictable from X;
- an  $R^2$  of 0.20 means that 20 percent is predictable; and so, on

# Linear Regression

## Adjusted R<sup>2</sup>

- Adjusted R<sup>2</sup> (Adjusted R-squared) is a statistical measure that modifies the R-squared value to account for the number of predictors in a regression model.
- While R-squared indicates the proportion of the variance in the dependent variable that is predictable from the independent variables, Adjusted R-squared provides a more accurate measure when multiple predictors are used.

Simple Linear Regression:



$$SS_{res} = \text{SUM } (y_i - \hat{y}_i)^2$$

$$SS_{tot} = \text{SUM } (y_i - y_{avg})^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

$$\text{Adj } R^2 = 1 - (1 - R^2) \frac{n - 1}{n - p - 1}$$

p - number of regressors

n - sample size

**A higher Adjusted R-squared indicates a better fit of the model.**

# Linear Regression

## F-statistic:

- Tests the overall significance of the model.
- The F-statistic in linear regression is a measure used to determine if there is a significant relationship between the dependent variable and the set of independent variables.

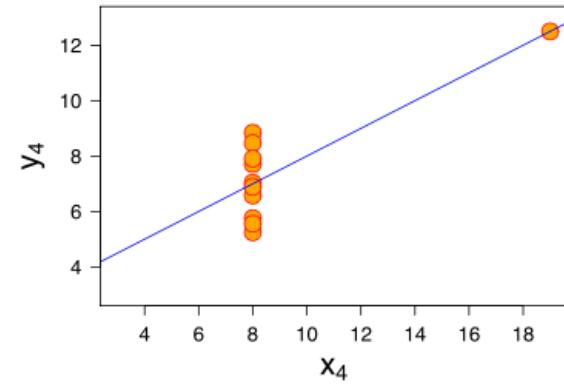
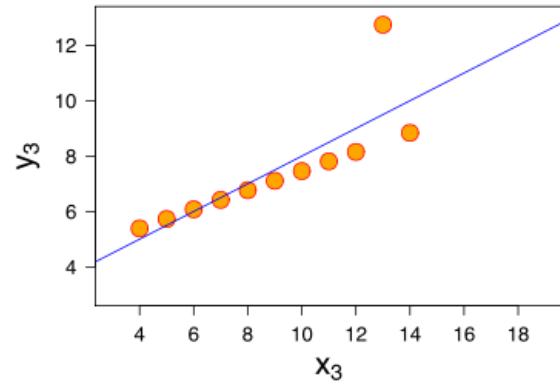
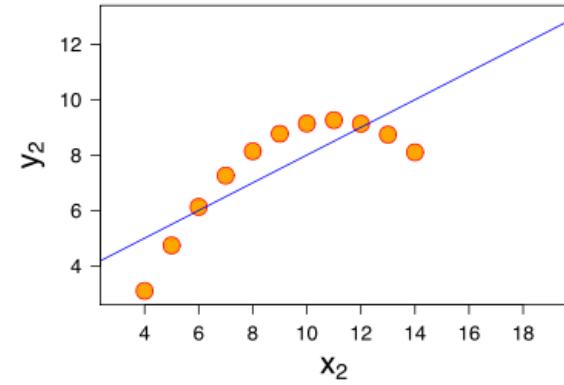
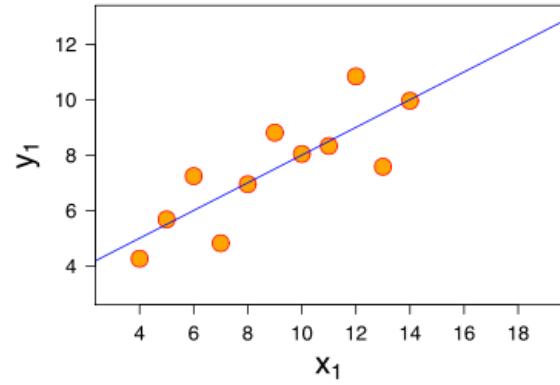
# Linear Regression

## Hypothesis Testing

- **t-tests:** Used to determine the significance of individual regression coefficients.
- **p-value:** Indicates the probability that the coefficient is different from zero.

# Linear Regression

## Where Linear Regression isn't suitable:



# Linear Regression

## Python Implementation

### Import library

```
from sklearn.linear_model import LinearRegression
```

### Perform required Data Preprocessing

#### Data Split

### Implementation

```
Linear_Regression = LinearRegression()  
Linear_Regression.fit(x_train, y_train)  
print("Mean Squared Error: ", mean_squared_error(y_test, y_pred).round(2))  
print("Mean Absolute Error: ", mean_absolute_error(y_test, y_pred).round(2))
```

Conduct HP Tuning to improve the performance and to get reliable model

# Gradient Descent

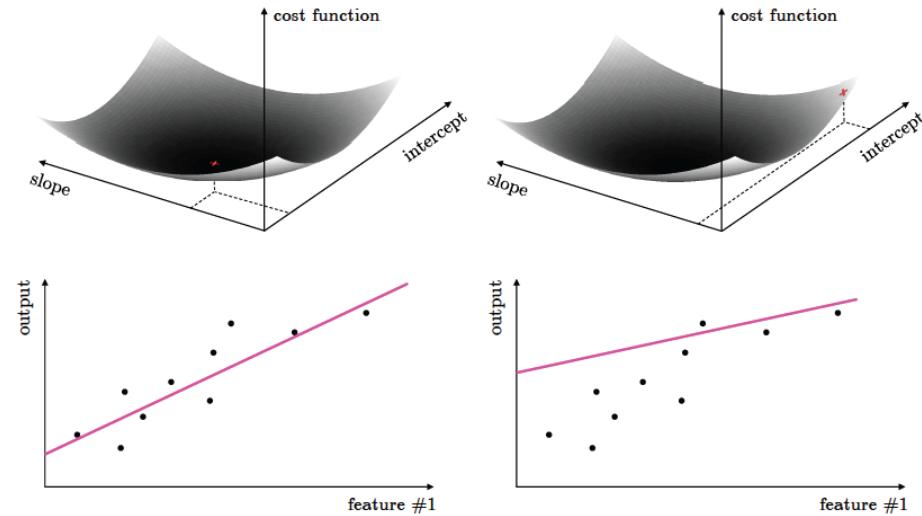


# Gradient Descent

- Gradient Descent is an optimization algorithm used to minimize a function, commonly applied in machine learning for minimizing the **cost function** in models like Linear Regression.
- In Linear Regression, we try to fit a straight line to a set of data points. The equation of the line is:  $y = mx + b$
- The goal of linear regression is to minimize the difference between the predicted  $\hat{y}$  and the actual  $y$ . We measure this difference using a cost function (usually Mean Squared Error, MSE)

# Gradient Descent

## 2D Error/lost/cost function of a linear model



Cost function as function of the slope and intercept parameters of a linear model on a toy dataset

Two different sets of parameter values (solutions:

- (left) at the minimum of the cost function
- (right) at a point with larger cost function value.

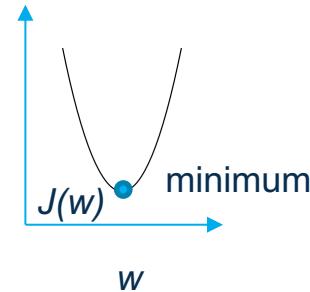
The set of parameters resulting in the best performance are found at the minimum of the cost surface.

# Gradient Descent

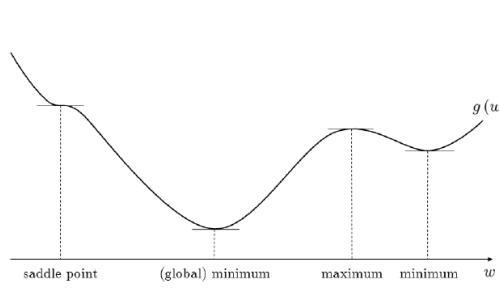
## Finding minima

- If the problem is deterministic and can be solved mathematically, it may be possible to find an optimal solution.
- However real-world problems are complex and may not have suitable mathematical model.
- Also, there may be several minimas. As an example, see Figure:

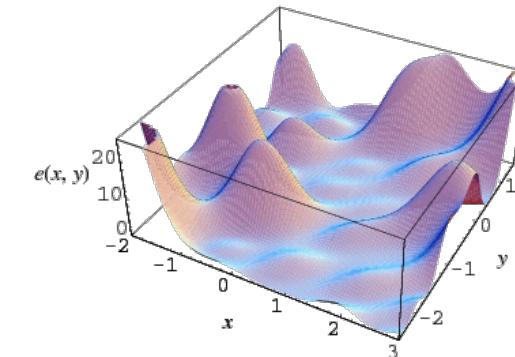
### Sample Error/loss functions



1 - Parameter



1 - Parameter



2 - Parameters

**Goal: to find the model/solution with minimum loss/error**

# Gradient Descent

## An Alternate view of Machine Learning

Imagine there are many different types of models each with several sets of parameters. A set of all these is the “solution space”.

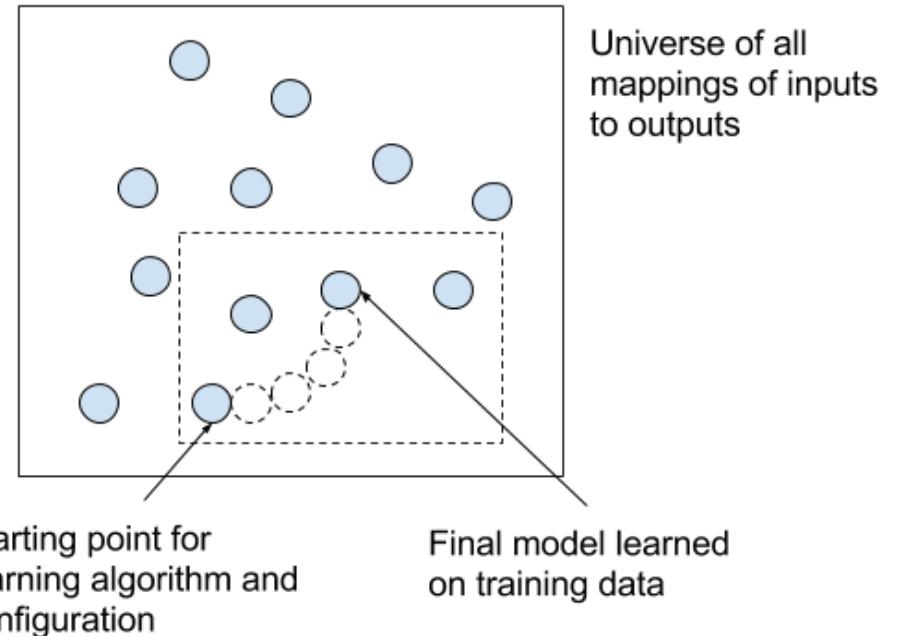
An optimum solution can be found in two ways:

Deterministic:

if the problem can be modeled mathematically

Nondeterministic:

Iterative – **search** for an optimum solution



# Gradient Descent

## Generalization

- The tuning of these parameters require the *minimization of a cost function* (or **searching** for the minimum cost function).
- It can be formally written as follows:
  - For a generic function  $g(w)$  taking in a general  $N$  dimensional input  $w$  the problem of finding the particular point  $v$  where  $g$  attains its smallest value is written formally as

$\underset{w}{\text{minimize}} \ g(w)$ . —————> Read as *minimize  $g(w)$  over  $w$*

# Gradient Descent

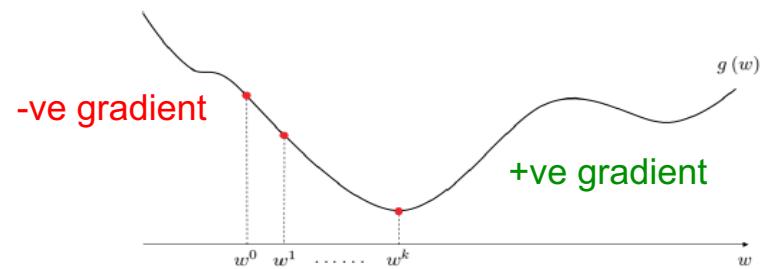
## Variations in learning

- Epoch – One set of presentation of all training samples to the model
- Batch Mode – present all training samples in each epoch
- Minibatch mode – present a small set of training samples in each epoch and decide on the weight update.
- Update all weights in every epoch
- Update weights one at a time, in each epoch

# Gradient Descent

## Moving towards minima – Gradient Descent (GD)

- Gradient towards minimum is increasing.
- On the left of the minimum, gradient is **negative** and on the right it is **positive**.
- Hence to move /descend towards minimum,
  - if the gradient is -ve, increase the value of  $w$
  - Else if the gradient is +ve decrease the value of  $w$
- That is update the parameter value in negative of the gradient direction
- Based on this principle, the **Gradient Descent algorithm** is formulated.



# Gradient Descent

## Moving towards minima – Gradient Descent (GD)

- Finding the next  $w$  in moving towards minima

---

**Algorithm 2.1** Gradient descent (with fixed step length)

---

**Input:** differentiable function  $g$ , fixed step length  $\alpha$ , and initial point  $w^0$

$k = 1$

Repeat until stopping condition is met:

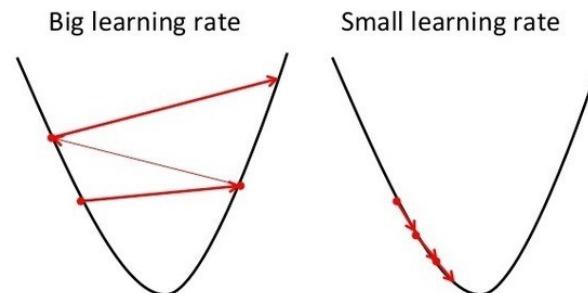
$$w^k = w^{k-1} - \alpha \nabla g(w^{k-1})$$

$$k \leftarrow k + 1$$


---

- Moving in the negative of gradient

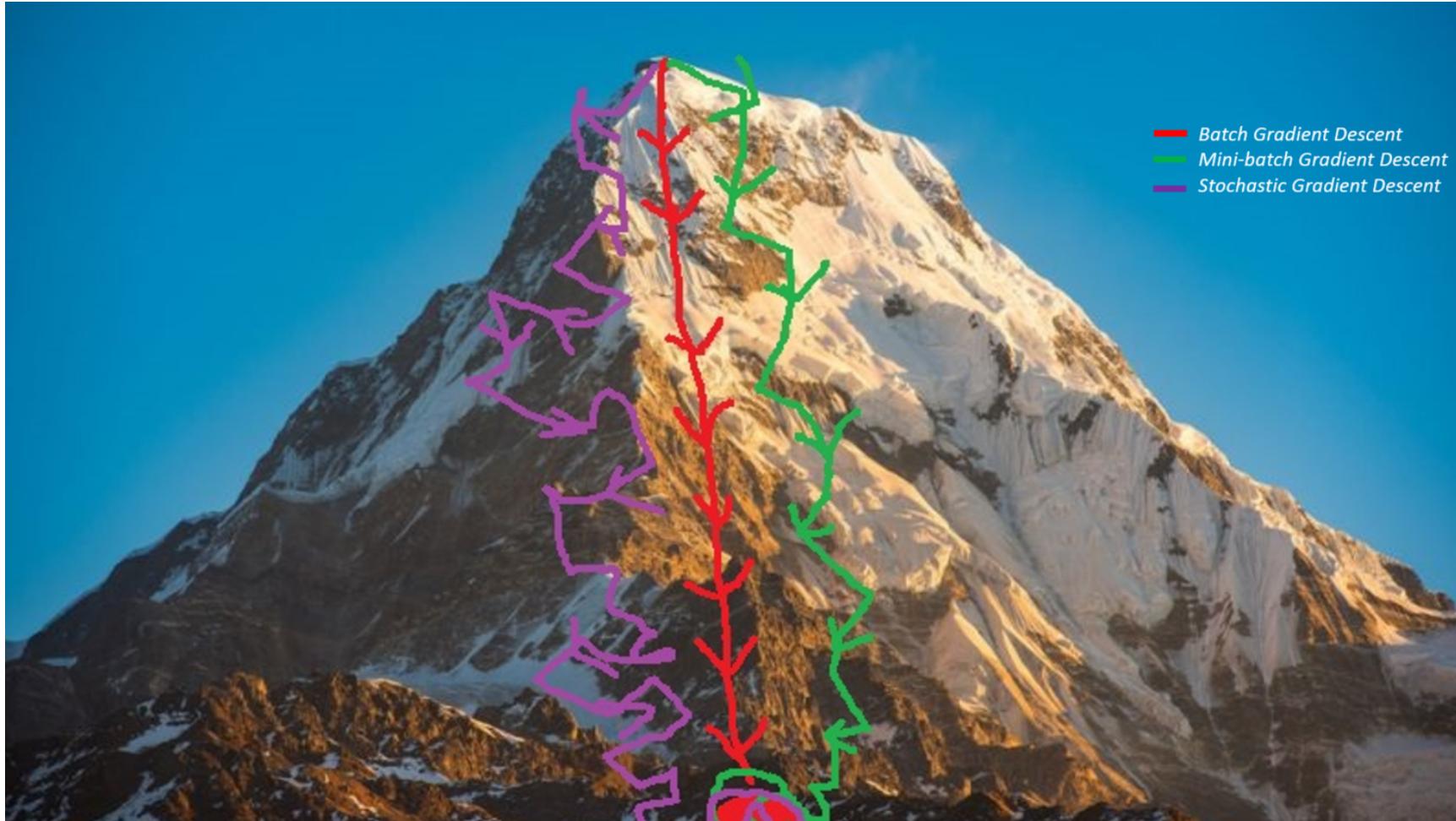
Gradient Descent



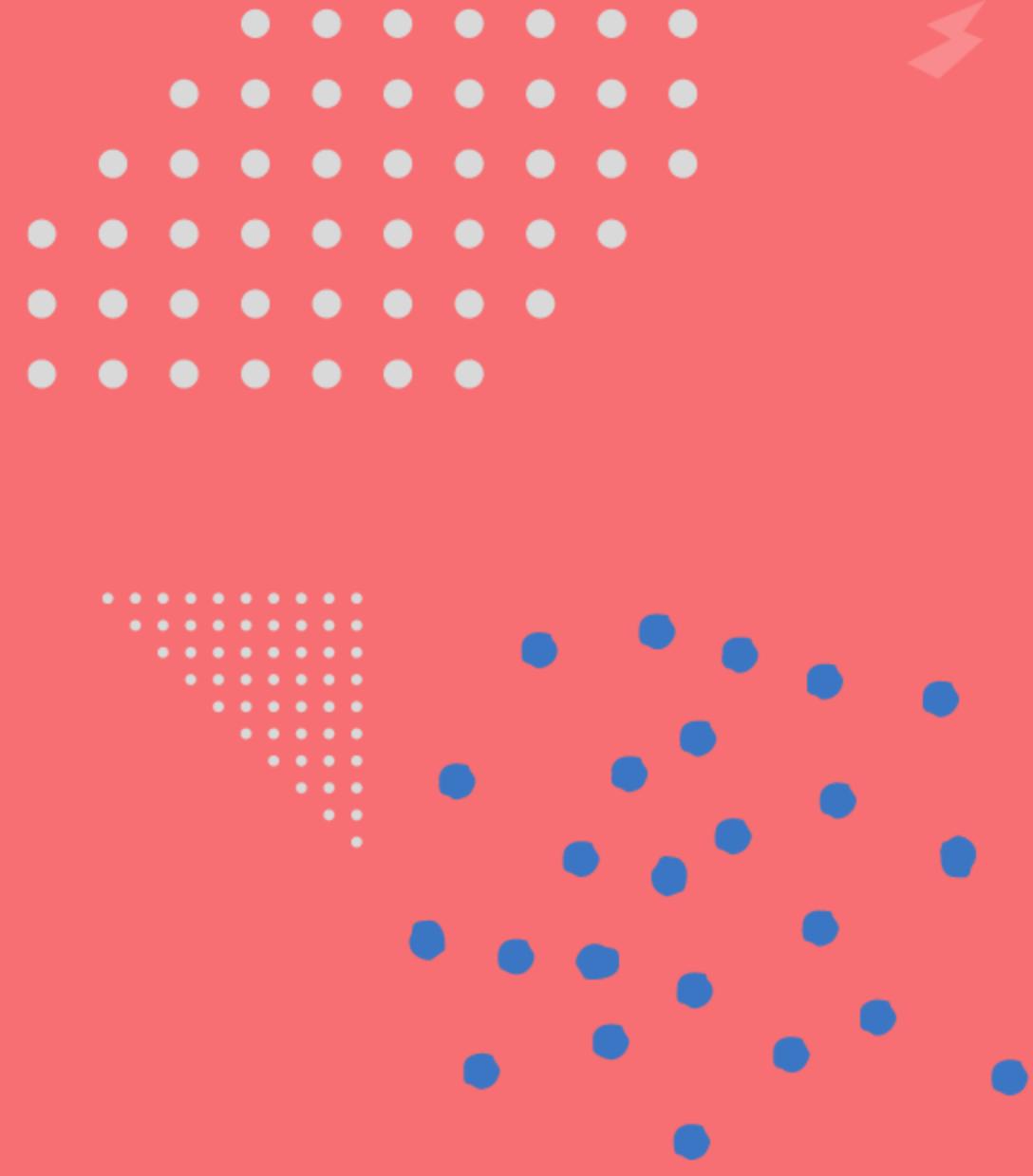
- Learning Rate →

# Gradient Descent

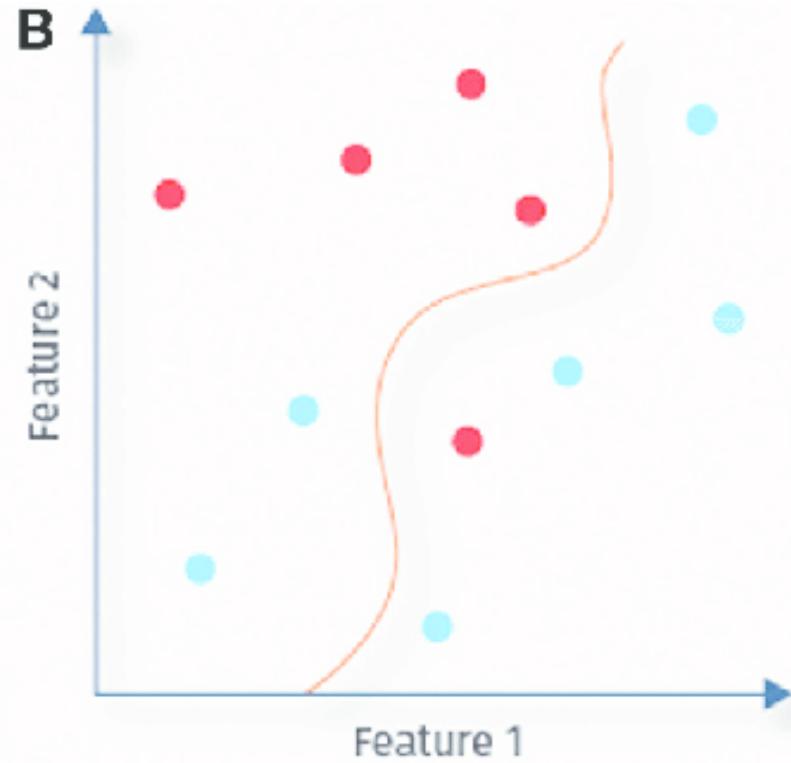
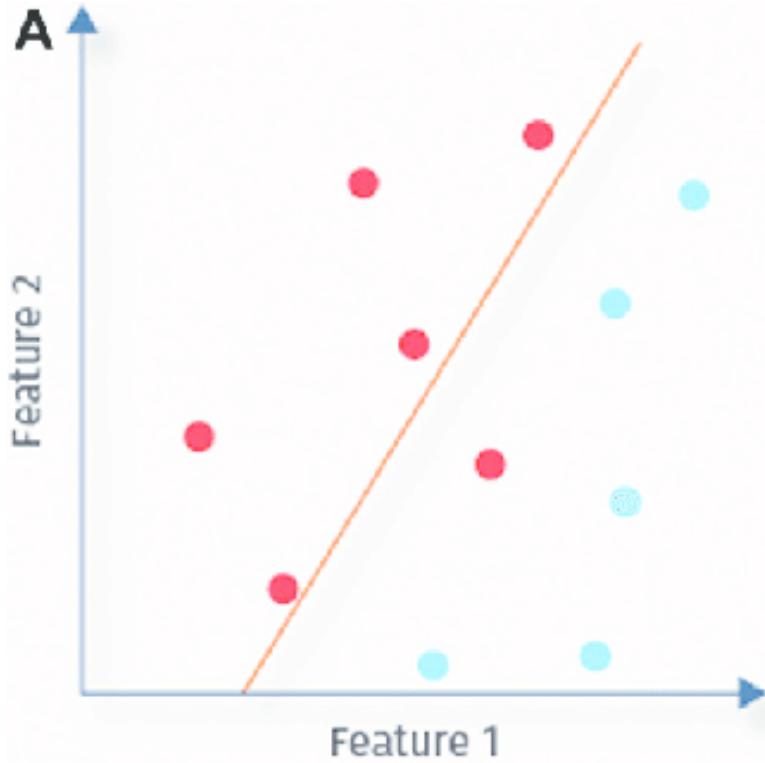
## Moving towards minima – Gradient Descent (GD)



# Logistic Regression



# Logistic Regression

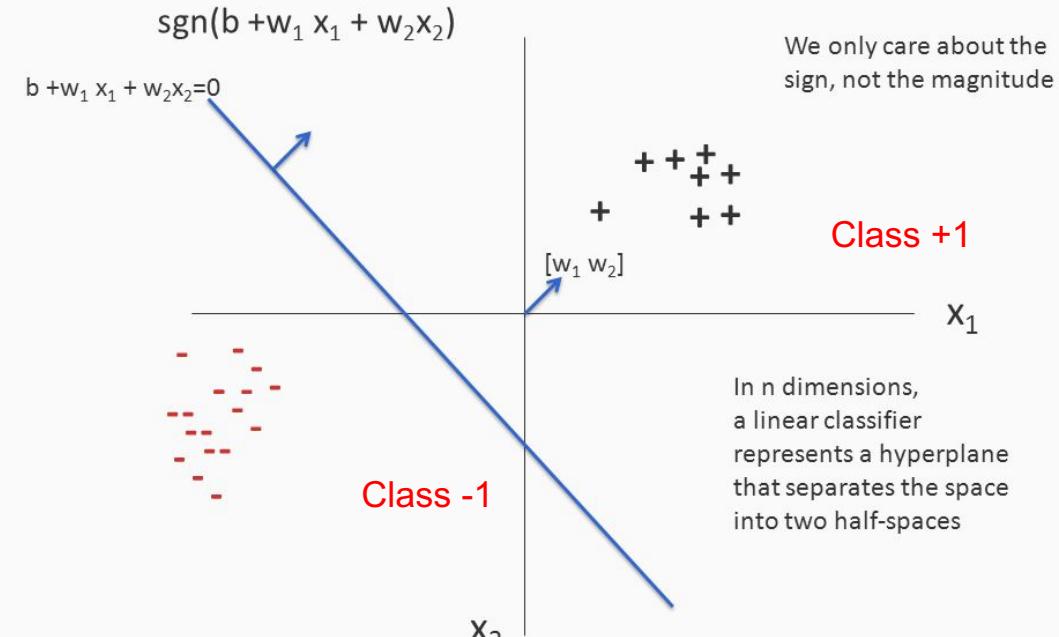


● Class 1  
● Class 2

# Logistic Regression

## Decision Boundary

### The geometry of a linear classifier

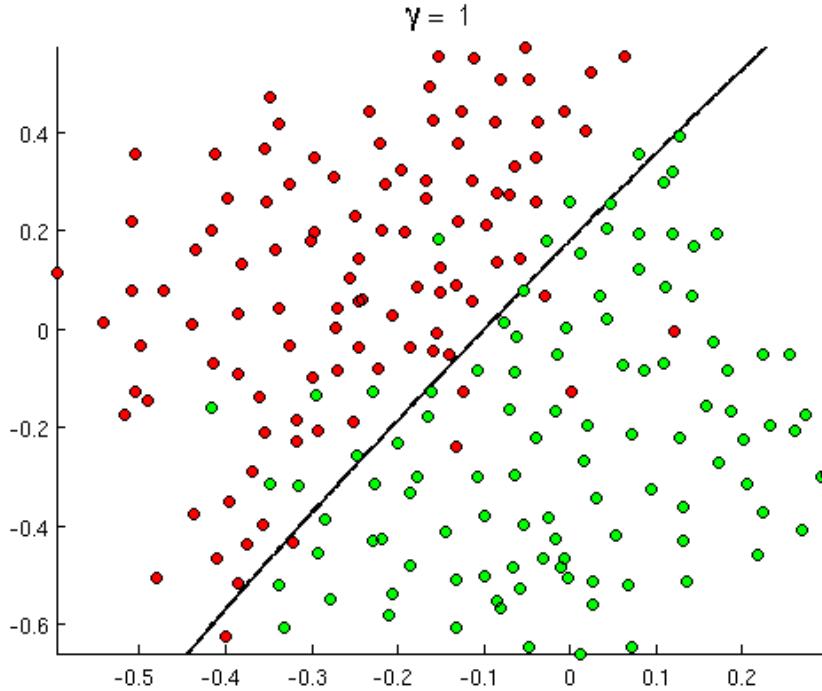


14

In deciding the class, we are only interested in the sign of the “score”:  $w_0 + w_1 x_1 + w_2 x_2$

# Logistic Regression

## Overlapping Classes



In this case there is a possibility that the points close to the boundary may belong to either class.

This ambiguity of class assignment is expressed through **probabilities**.

# Logistic Regression

## Magnitude of the “score”

$$\text{Score} = w_0 + w_1x_1 + w_2x_2$$

In deciding on the **class** of an instance, we are only **interested in the sign** of the “score”.

But **how sure** are we of the class?

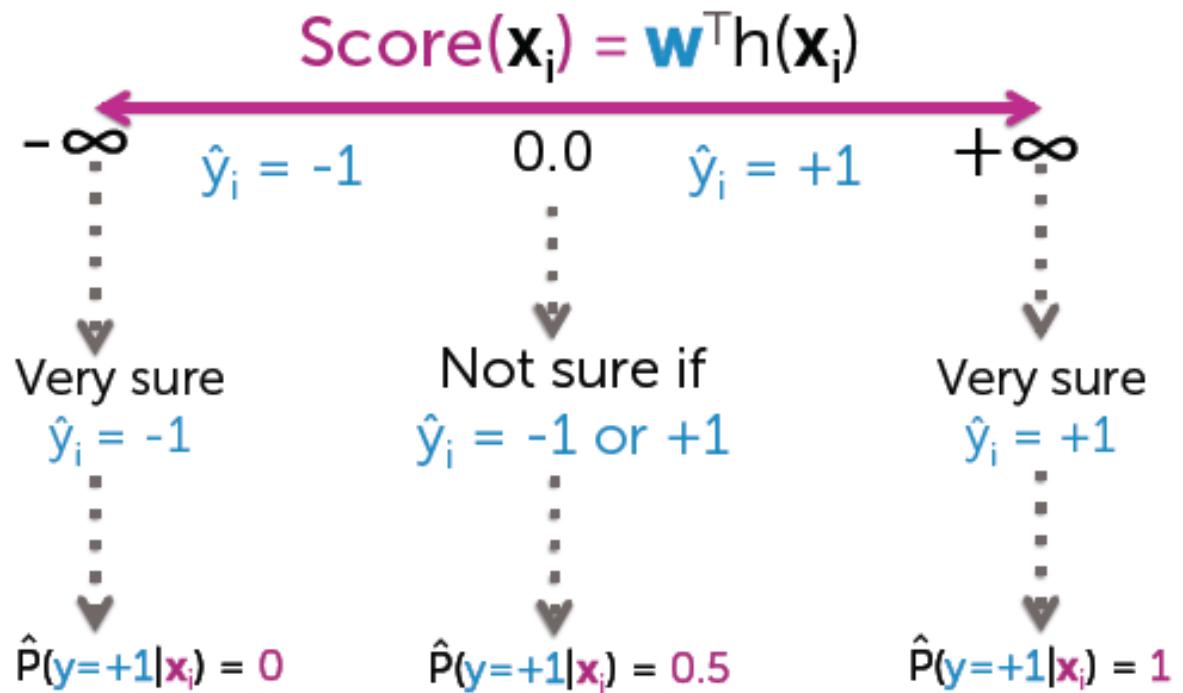
This depends on the distance of each instance from the decision boundary. Larger the distance surer is the class.

Hence , we are interested in the **magnitude of the score**, which is a measure of how far is the instance from the boundary.

# Logistic Regression

## Interpreting the score

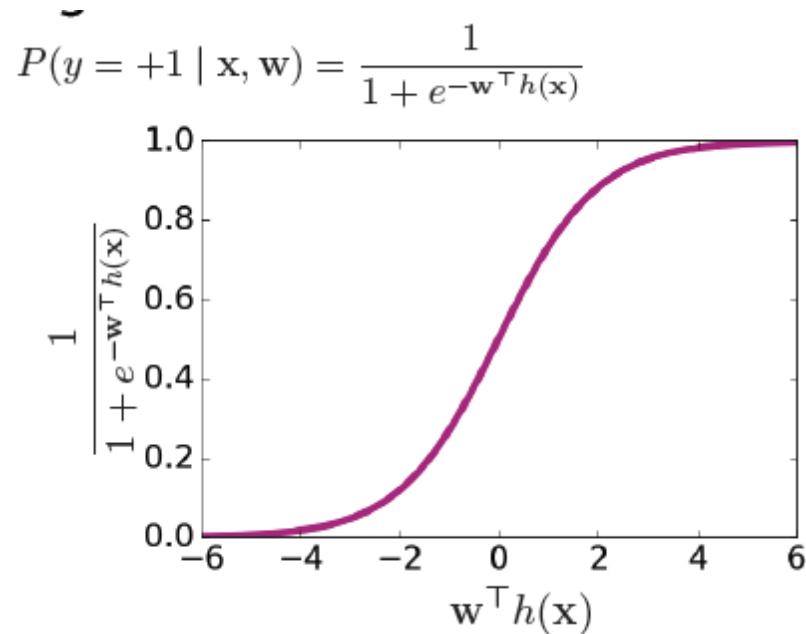
### Interpreting Score( $\mathbf{x}_i$ )



This leads us to the formulation of logistic regression and its training

# Logistic Regression

## Logistic Regression Model



### SIGMOID Function

Fit linear boundary with  $\mathbf{w}^T \mathbf{h}(\mathbf{x}) = 0$ .

For each instance, transform the scores  $\mathbf{w}^T \mathbf{h}(\mathbf{x})$ , to probabilities using the **sigmoid** function.

Learn model parameters  $\mathbf{w}$ , through training the model.

# Logistic Regression

## Objective function of Logistic Regression (LR)

- **The central idea in machine learning is:**
  - An objective function that reflects the performance of the model is defined.
  - Parameters that optimize this objective function are determined.
    - This is also known as Parameter tuning.
- **In logistic regression various objective functions are mentioned:**
  - Maximum Likelihood
  - Log Likelihood
  - Cross Entropy

**Effectively, all these functions mean the same.**

# Logistic Regression

## Python Implementation

### Import library

```
from sklearn.linear_model import LogisticRegression
```

### Perform required Data Preprocessing

#### Data Split

### Implementation

```
log_reg = LogisticRegression ()  
log_reg.fit(x_train, y_train)  
print('Accuracy: ', round(accuracy_score(y_test, y_pred), 2)*100, '%')  
print('Precision: ', round(precision_score(y_test, y_pred, average = 'micro'), 2))
```

Conduct HP Tuning to improve the performance and to get reliable model

# Review Questions

1. What is simple linear regression, and how does it differ from multiple linear regression?
2. Explain the concept of polynomial linear regression and its applications.
3. What is the R coefficient, and how is it used to evaluate the goodness of fit of a regression model?
4. What are the key assumptions of linear regression, and how can they be checked?
5. What is the objective function in Logistic regression?
6. How do you explain decision boundary in Logistic regression?

# Summary / Recap of Main Points

- Simple linear regression involves one independent variable, whereas multiple linear regression involves two or more independent variables.
- Polynomial regression uses polynomial terms of the independent variable(s) to capture non-linear relationships. It's useful for modeling curved trends in data.
- $R^2$  measures the proportion of variance in the dependent variable explained by the independent variable(s). Higher R-squared values indicate a better fit.
- Logistic Regression is used for Classification.
- Various objectives functions available.

# What To Expect Next Week

## In Class

- Introduction to Unsupervised Learning

## Preparation for Class

- Introduction to Unsupervised Learning

CX016-2.5-3-IML - Introduction to Machine Learning

## **Introduction to Unsupervised Learning**

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understand Unsupervised Learning
2. Explore Common Unsupervised Learning Algorithms
3. Explore the applications of Unsupervised Learning Techniques

# Contents & Structure

1. Unsupervised Learning
2. Common Unsupervised Learning Algorithms
3. Applications of Unsupervised Learning Techniques

# Recap From Last Lesson

- What is simple linear regression, and how does it differ from multiple linear regression?
- Explain the concept of polynomial linear regression and its applications.
- What is the R coefficient, and how is it used to evaluate the goodness of fit of a regression model?
- What are the key assumptions of linear regression, and how can they be checked?

# Unsupervised Learning

- Unsupervised learning is a type of machine learning where the algorithm is trained on unlabeled data.
- This means that the system is given input data without explicit instructions on what to do with it.
- The goal of unsupervised learning is to find hidden patterns or intrinsic structures in the input data.
- It then explores the data, identifying similarities, differences, and underlying structures.

# Unsupervised Learning

## Key Concepts

- Unlabeled Data:
  - Unsupervised learning deals with data that has no labels. The algorithm must learn to understand the structure of the data on its own.
- Dimensionality Reduction:
  - Techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) are used to reduce the number of variables under consideration, simplifying the data while preserving its essential characteristics.
- Clustering:
  - This involves grouping a set of objects in such a way that objects in the same group (called a cluster) are more like each other than to those in other groups. Common clustering algorithms include K-Means, Hierarchical Clustering, and DBSCAN.
- Association Rule:
  - This method is used to find relationships between variables in large datasets. Association rule learning is often used in market basket analysis, where the goal is to find items that frequently co-occur in transactions.

# Unsupervised Learning

## Common Techniques

- Clustering
  - K-Means: Divides data into K clusters based on similarity.
  - Hierarchical Clustering: Creates a hierarchy of clusters.
  - DBSCAN: Discovers clusters of arbitrary shape.
- Association Rule Learning:
  - Apriori: Finds frequent item sets and generates association rules.
  - Eclat: Efficient algorithm for finding frequent item sets.
- Dimensionality Reduction:
  - Principal Component Analysis (PCA): Reduces dimensionality while preserving variance.
  - t-Distributed Stochastic Neighbor Embedding (t-SNE): Effective for visualizing high-dimensional data.
- Anomaly Detection:
  - Isolation Forest: Isolates anomalies by randomly partitioning data.
  - One-Class SVM: Defines a boundary around normal data points.

# Unsupervised Learning

## Applications

- **Market Basket Analysis:** Understanding the purchase behavior of customers by finding associations between different products.
- **Customer Segmentation:** Dividing a customer base into distinct groups of individuals that share similar characteristics.
- **Anomaly Detection:** Identifying fraudulent transactions or unusual patterns that do not conform to expected behavior.
- **Data Compression:** Reducing the dimensionality of data to save storage space and speed up computation.
- **Recommendation Systems:** Providing recommendations based on user behavior patterns.

# Unsupervised Learning

## Challenges

- **Evaluation:** Since there are no labels, it's challenging to evaluate the performance of an unsupervised learning model.
- **Interpretability:** The patterns or structures identified may not always be easy to interpret or actionable.
- **Scalability:** Processing large datasets can be computationally intensive and require efficient algorithms.
- **Model Selection:** Choosing the right algorithm for a specific problem can be complex.

# Unsupervised Learning

## Key Steps

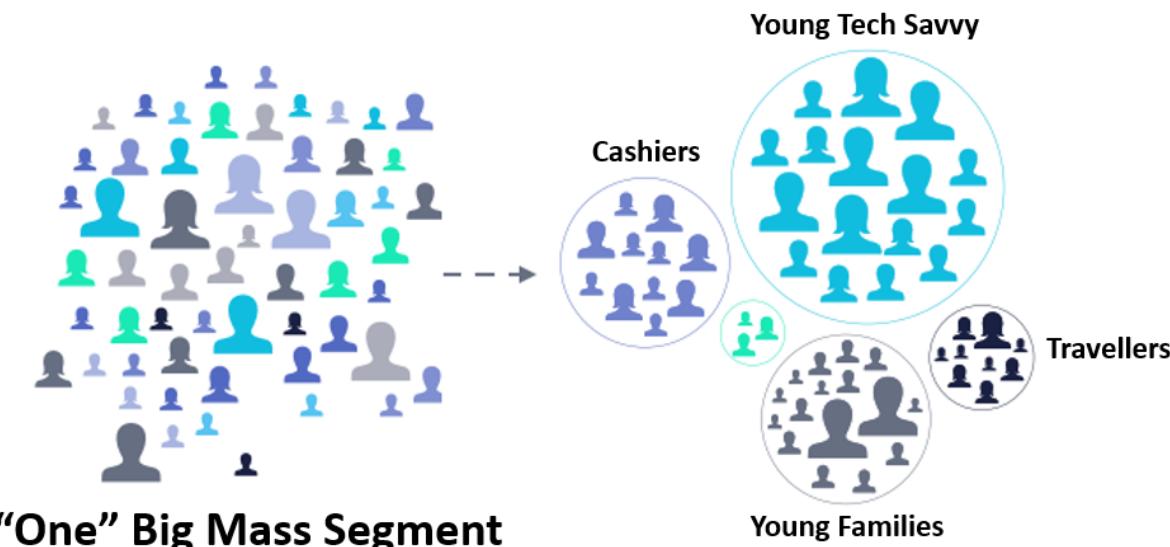
- Data Preparation: The data is cleaned, preprocessed, and formatted to be suitable for the algorithm.
- Model Selection: An appropriate unsupervised learning algorithm is chosen based on the problem and data characteristics.
- Training: The algorithm analyzes the data to discover patterns and relationships.
- Evaluation: The quality of the discovered patterns is assessed using appropriate metrics.

# Unsupervised Learning

## Example

### Example: Customer Segmentation

- A retail company can use unsupervised learning to segment its customers into different groups based on purchasing behavior, demographics, and other relevant factors.
- This information can help the company tailor marketing campaigns, product recommendations, and customer service strategies for each segment.



# Review Questions

1. What are the fundamental concepts of unsupervised learning, and how does it differ from supervised learning?
2. Explain the key differences between clustering and dimensionality reduction algorithms within the realm of unsupervised learning?
3. How can unsupervised learning be applied to customer segmentation in a retail setting?
4. What are some of the challenges associated with implementing unsupervised learning algorithms on large datasets?

# Summary / Recap of Main Points

- Unsupervised learning is a powerful tool for making sense of complex and unlabeled data, revealing hidden patterns and structures that can provide valuable insights and inform decision-making across various domains.
- Unlike supervised learning, there's no predefined output or target variable.
- Instead, the model independently discovers patterns, structures, and relationships within the data.
- Unsupervised learning techniques like customer segmentation, anomaly detection, image and pattern recognition, and feature engineering are essential for extracting valuable insights from complex datasets.

# What To Expect Next Week

## In Class

- K-Means Clustering (KMeans)

## Preparation for Class

- K-Means Clustering (KMeans)

CX016-2.5-3-IML - Introduction to Machine Learning

## **K-Means Clustering (KMeans)**

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understand K-Means Clustering
2. Applications and Limitations of K-Means Clustering
3. Implementing K-Means Clustering

# Contents & Structure

1. K-Means Clustering
2. Applications and Limitations of K-Means Clustering
3. Implementing K-Means Clustering using Python

# Recap From Last Lesson

- What are the fundamental concepts of unsupervised learning, and how does it differ from supervised learning?
- Explain the key differences between clustering and dimensionality reduction algorithms within the realm of unsupervised learning?
- How can unsupervised learning be applied to customer segmentation in a retail setting?
- What are some of the challenges associated with implementing unsupervised learning algorithms on large datasets?

Machine Learning Concepts

# K-MEANS CLUSTERING

Incremental K-means and Batch K-means



# Incremental K-means and Batch K-means

- Incremental K-means and Batch K-means are variations of the K-means clustering algorithm, differing mainly in how they update cluster centroids.
- When to Use:
  - Batch K-means: If the dataset is static and you can afford to process all data at once, and memory is not an issue.
  - Incremental K-means: When working with large datasets, streaming data, or when memory is limited.

# Batch K-means (Standard K-means)

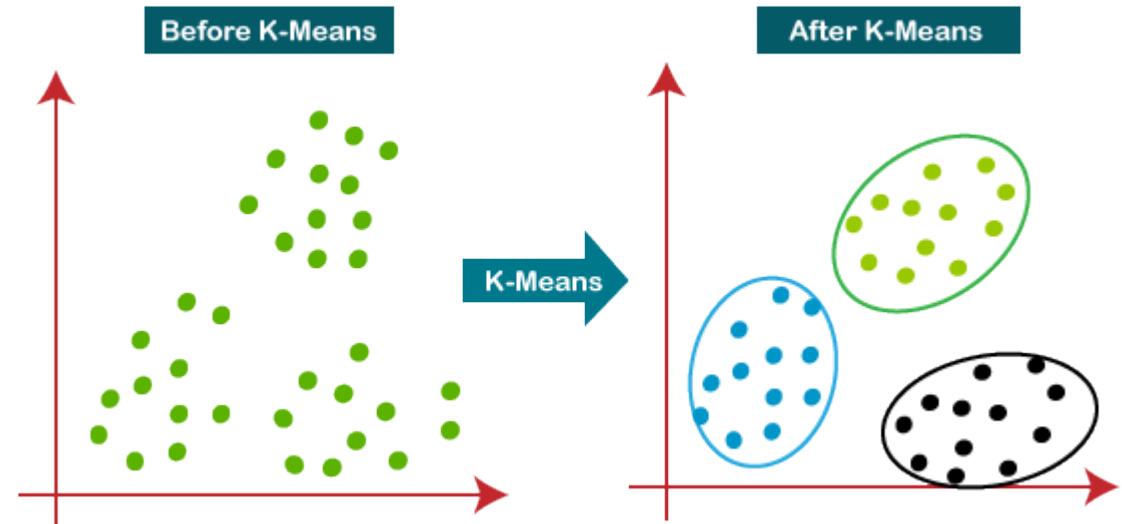
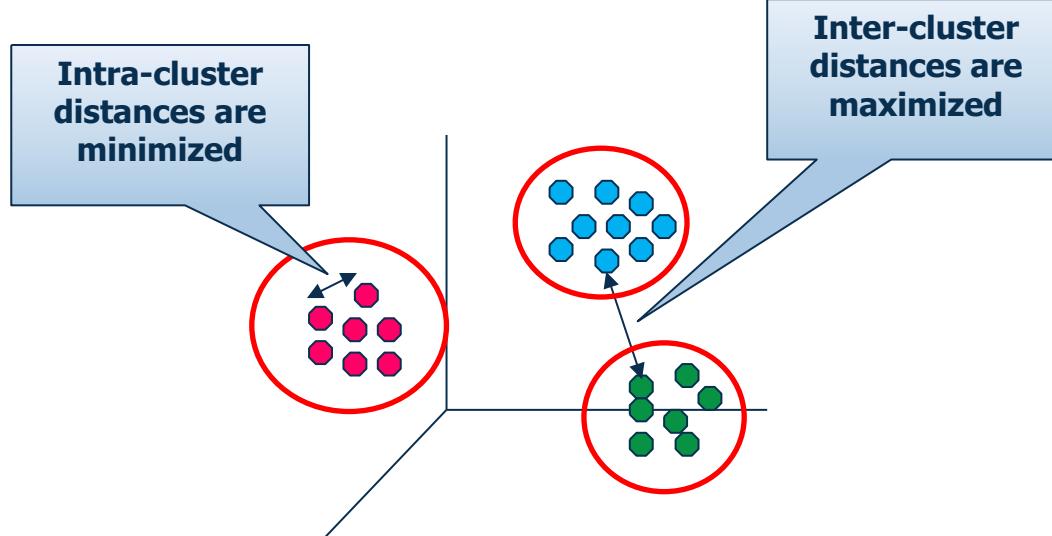
- Process:
  - This is the traditional K-means algorithm where all data points are processed together in a batch.
  - It iteratively updates the cluster centroids by assigning all data points to the nearest centroids and then recalculating the centroids based on the assigned points.
- Algorithm:
  - Initialize K centroids randomly.
  - Assign each data point to the nearest centroid.
  - Recalculate the centroid of each cluster.
  - Repeat the assignment and centroid update until convergence.

# K-MEANS CLUSTERING



# K-Means Clustering (K-Means)

- K-Means clustering is a popular unsupervised machine learning algorithm used to partition a dataset into distinct groups or clusters.
- The goal of the algorithm is 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 Clustering (KMeans)

## Steps

- **Initialization:** Choose the number of clusters, K. Randomly initialize K cluster centroids.
- **Assignment Step:** Assign each data point to the closest centroid. This step forms K clusters.
- **Update Step:** Recalculate the centroids as the mean of all data points assigned to each cluster.
- **Repeat:** Repeat the assignment and update steps until the centroids no longer change (convergence) or for a fixed number of iterations.

# K-Means Clustering (KMeans)

## Objective Function

- The objective of K-Means is to minimize the **within-cluster sum of squares (WCSS)**, also known as inertia:

$$\text{Inertia} = \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2$$

where:

- $C_i$  is the set of points in cluster  $i$ .
- $\mu_i$  is the centroid of cluster  $i$ .

# K-Means Clustering (KMeans)

## The K-Means Clustering Method

K-means algorithm is implemented in 5 steps:

- **Step 1:** Ask the user how many clusters  $K$  the data set should be partitioned into.
- **Step 2:** Randomly assign  $k$  records to be the initial cluster center locations.
- **Step 3:** For each record, find the nearest cluster center. Thus, in a sense, each cluster center “owns” a subset of the records, thereby representing a partition of the data set. We therefore have  $k$  clusters,  $C_1, C_2, \dots, C_k$ .
- **Step 4:** For each of the  $k$  clusters, find the cluster centroid, and update the location of each cluster center to the new value of the centroid.
- **Step 5:** Repeat steps 3 to 5 until convergence or termination.

# K-Means Clustering (KMeans)

## The K-Means Clustering Method

**Manhattan Equation** to calculate the nearest value to the center of cluster.

$$d = \sum_{i=1}^n |x_i - y_i|$$

**Can also use:**

**Euclidean:** to calculate the nearest value to the center of cluster.

$$d_{\text{Euclidean}}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_i (x_i - y_i)^2}$$

**Example:** Consider the following dataset consisting of the ratings of two variables on each of seven movies.

Movie	A	B
M1	1.0	1.0
M2	1.5	2.0
M3	3.0	4.0
M4	5.0	7.0
M5	3.5	5.0
M6	4.5	5.0
M7	3.5	4.5

# K-Means Clustering (KMeans)

## The K-Means Clustering Method

**Steps 1 and 2:** Lets choose two seeds in random

**Steps 3 & 4:** Compute the distances using the two attributes and using the sum of absolute difference for simplicity (K-means method) – using **Manhattan Equation**

Movie	A	B
M1	1.0	1.0
M4	5.0	7.0

# K-Means Clustering (KMeans)

## The K-Means Clustering Method

	A	B	DISTANCE FROM CLUSTERS			ALLOCATION TO NEAREST CLUSTER
C1	1	1				
C2	5	7	C1		C2	
M1	1	1	0		10	C1
M2	1.5	2	1.5		8.5	C1
M3	3	4	5		5	C1, C2
M4	5	7	10		0	C2
M5	3.5	5	6.5		3.5	C2
M6	4.5	5	7.5		2.5	C2
M7	3.5	4.5	6		4	C2

# K-Means Clustering (KMeans)

## The K-Means Clustering Method

### Step 5: Find new centroids

	<b>A</b>	<b>B</b>
<b>C1</b>	1.83	2.33
<b>C2</b>	3.9	5.1
<b>SEED1</b>	1	1
<b>SEED2</b>	5	7



**Cluster 1 → M1, M2**  
**Cluster 2 → M3, M4, M5, M6, M7**

			<b>DISTANCE FROM CLUSTERS</b>		<b>ALLOCATION TO THE NEAREST CLUSTER</b>	
<b>C1</b>	<b>1.83</b>	<b>2.33</b>	<b>FROM</b>			
<b>C2</b>	<b>3.9</b>	<b>5.1</b>	<b>C1</b>		<b>C2</b>	
<b>M1</b>	1	1	<b>2.16</b>		7	<b>C1</b>
<b>M2</b>	<b>1.5</b>	2	<b>0.66</b>		5.5	<b>C1</b>
<b>M3</b>	3	4	<b>2.84</b>		2	<b>C2</b>
<b>M4</b>	5	7	<b>7.84</b>		3	<b>C2</b>
<b>M5</b>	<b>3.5</b>	5	<b>4.34</b>		0.5	<b>C2</b>
<b>M6</b>	4.5	5	<b>5.34</b>		0.5	<b>C2</b>
<b>M7</b>	3.5	4.5	<b>3.84</b>		1	<b>C2</b>

# K-Means Clustering (KMeans)

## Python Implementation

### Import library

```
from sklearn.cluster import Kmeans
```

### Fit K-Means

```
kmeans = KMeans(n_clusters = 3)  
kmeans.fit(X)
```

### Get cluster assignments and centroids

```
labels = kmeans.labels_  
centroids = kmeans.cluster_centers_
```

# Review Questions

1. What are the main steps involved in the K-Means clustering algorithm, and how does the algorithm ensure that clusters are formed?
2. Describe at least two real-world applications of K-Means clustering and explain why this algorithm is suitable for these applications.
3. What are some limitations of K-Means clustering, and how might these limitations affect the results of clustering in certain datasets?

# Summary / Recap of Main Points

- K-Means Clustering is a popular unsupervised learning algorithm used to partition a dataset into  $K$  distinct, non-overlapping subsets or clusters. Each data point belongs to the cluster with the nearest mean, serving as a prototype of the cluster.
- Widely used in market segmentation, document clustering, image segmentation, and pattern recognition. It's efficient for large datasets and provides straightforward interpretability.
- K-Means assumes clusters to be spherical and of similar size, which may not always be true. It can be sensitive to initial placement of centroids and requires the number of clusters ( $K$ ) to be specified in advance.

# What To Expect Next Week

## In Class

- Introduction to hierarchical clustering

## Preparation for Class

- Introduction to hierarchical clustering

CX016-2.5-3-IML - Introduction to Machine Learning

## **Introduction to hierarchical clustering**

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understand the basics of Hierarchical Clustering
2. Applications and Limitations of Hierarchical Clustering
3. Implementing Hierarchical Clustering

# Contents & Structure

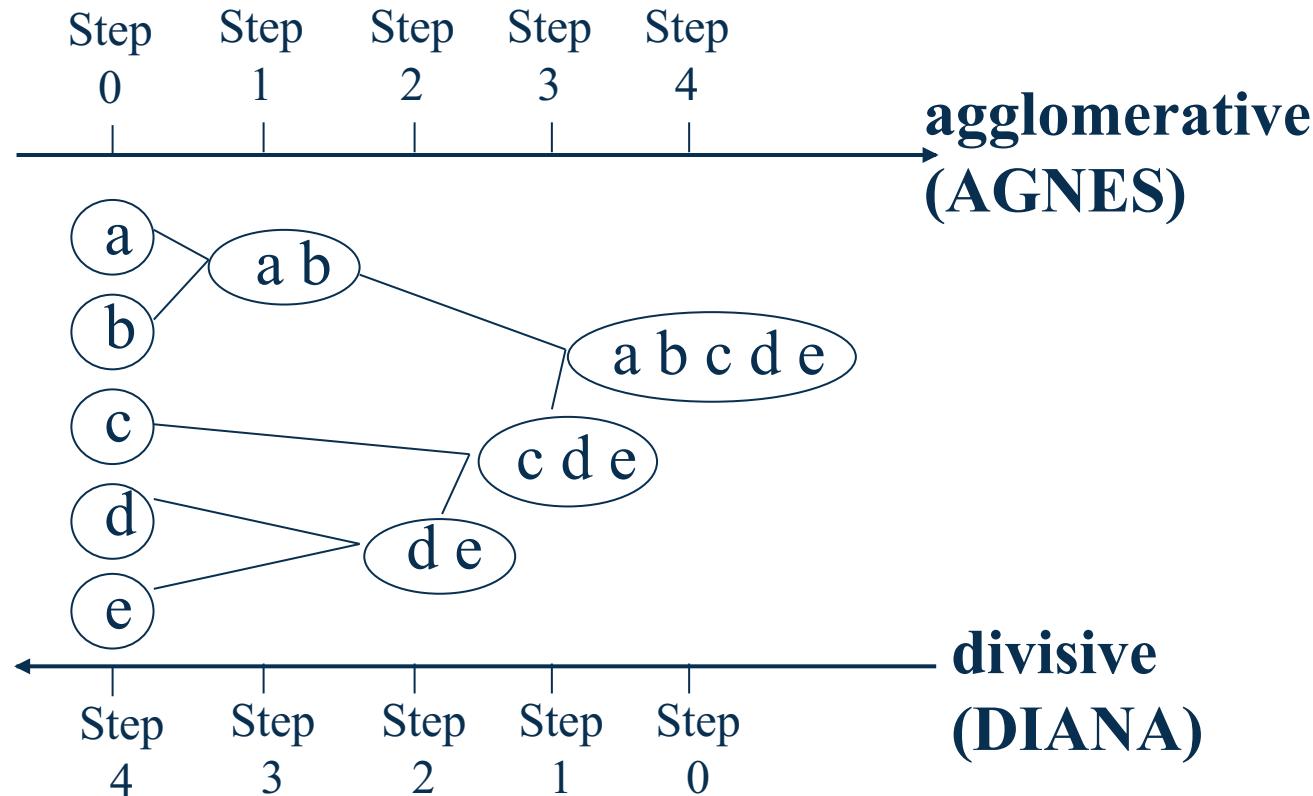
1. Hierarchical Clustering Clustering
2. Applications and Limitations of Hierarchical Clustering
3. Implementing K-Means Clustering using Python

# Recap From Last Lesson

- What are the main steps involved in the K-Means clustering algorithm, and how does the algorithm ensure that clusters are formed?
- Describe at least two real-world applications of K-Means clustering and explain why this algorithm is suitable for these applications.
- What are some limitations of K-Means clustering, and how might these limitations affect the results of clustering in certain datasets?

# Introduction to hierarchical clustering

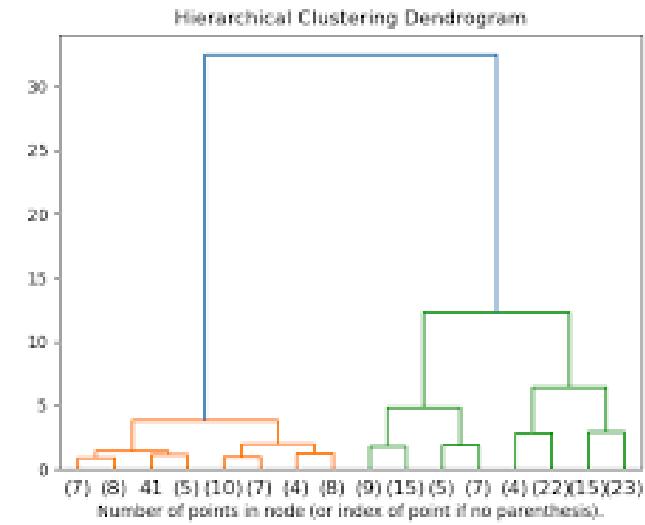
- Hierarchical clustering is a type of unsupervised learning used to group similar objects into clusters.
- This method involves creating a tree-like structure (dendrogram) that represents the nested grouping of similar objects.



# Introduction to hierarchical clustering

## Types of Hierarchical Clustering

- **Agglomerative (Bottom-Up) Clustering:**
  - Start: Each data point is considered a single cluster.
  - Process: Iteratively merges the closest pair of clusters until all data points belong to one cluster.
  - Visualization: The results are typically visualized in a dendrogram, which shows the hierarchical relationships between clusters.
  
- **Divisive (Top-Down) Clustering:**
  - Start: All data points are in one cluster.
  - Process: Iteratively splits the cluster into two sub-clusters until each data point is in its own cluster.
  - Visualization: Also represented by a dendrogram.



While divisive clustering is theoretically possible, it's less commonly used in practice due to computational complexity.

# Introduction to hierarchical clustering

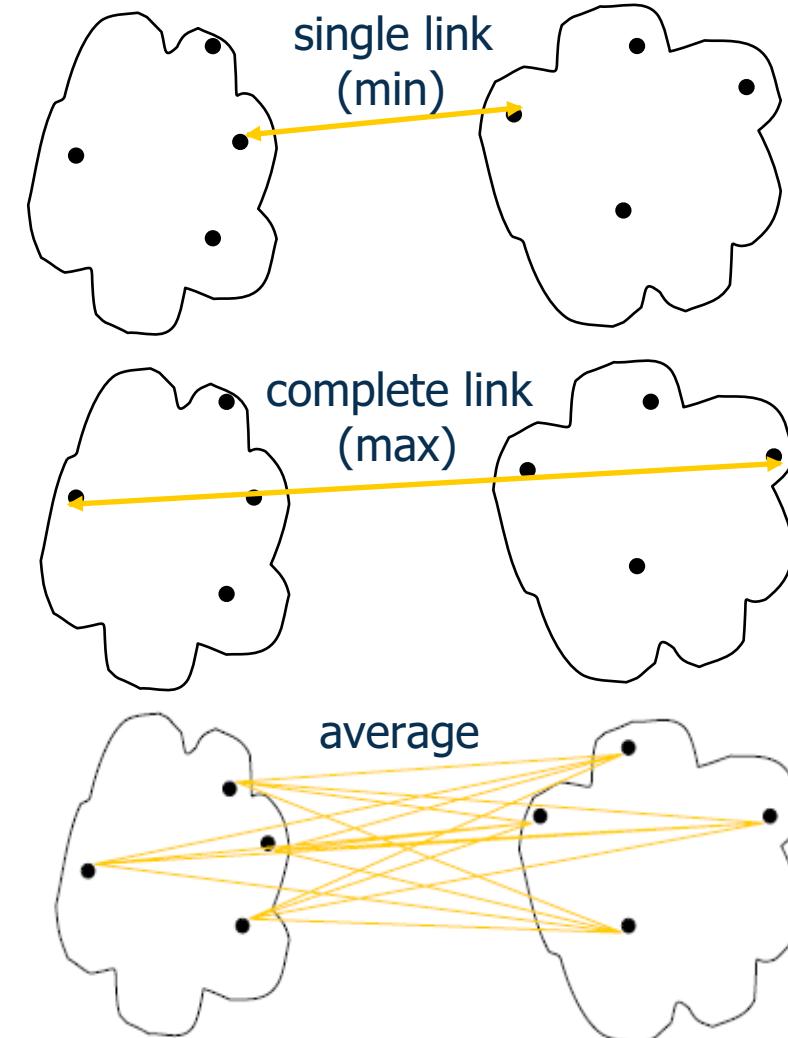
## Key Concepts

- **Linkage:** Determines the distance between clusters. Common linkage methods include:
  - Single linkage: Distance between the closest points of two clusters.
  - Complete linkage: Distance between the farthest points of two clusters.
  - Average linkage: Average distance between all pairs of points from two clusters.
  - Ward's method: Minimizes the sum of squared distances within clusters.
- **Dendrogram:** A tree-like diagram that visualizes the hierarchical relationships between clusters.
- **Cutting the dendrogram:** To determine the optimal number of clusters, the dendrogram can be cut at a specific height.

# Introduction to hierarchical clustering

## Cluster Distance Measures

- **Linkage:** Determines the distance between clusters. Common linkage methods include:
  - Single linkage: Distance between the closest points of two clusters.
  - Complete linkage: Distance between the farthest points of two clusters.
  - Average linkage: Average distance between all pairs of points from two clusters.
  - Ward's method: Minimizes the sum of squared distances within clusters.



# Introduction to hierarchical clustering

## Cluster Distance Measures

**Example:** Given a data set of five objects characterized by a single continuous feature, assume that there are two clusters: C1: {a, b} and C2: {c, d, e}.

	a	b	c	d	e
Feature	1	2	4	5	6

1. Calculate the distance matrix.

	a	b	c	d	e
a	0	1	3	4	5
b	1	0	2	3	4
c	3	2	0	1	2
d	4	3	1	0	1
e	5	4	2	1	0

2. Calculate three cluster distances between C1 and C2.  
Single link

$$\begin{aligned}\text{dist}(C_1, C_2) &= \min\{d(a, c), d(a, d), d(a, e), d(b, c), d(b, d), d(b, e)\} \\ &= \min\{3, 4, 5, 2, 3, 4\} = 2\end{aligned}$$

Complete link

$$\begin{aligned}\text{dist}(C_1, C_2) &= \max\{d(a, c), d(a, d), d(a, e), d(b, c), d(b, d), d(b, e)\} \\ &= \max\{3, 4, 5, 2, 3, 4\} = 5\end{aligned}$$

Average

$$\begin{aligned}\text{dist}(C_1, C_2) &= \frac{d(a, c) + d(a, d) + d(a, e) + d(b, c) + d(b, d) + d(b, e)}{6} \\ &= \frac{3 + 4 + 5 + 2 + 3 + 4}{6} = \frac{21}{6} = 3.5\end{aligned}$$

# Introduction to hierarchical clustering

## Steps in Hierarchical Clustering

- **Compute Distance Matrix:** Calculate the distance between every pair of data points using a distance metric (e.g., Euclidean distance).
- **Merge Clusters:** Merge the two closest clusters based on the distance matrix.
- **Update Distance Matrix:** Recalculate the distances between the new cluster and the remaining clusters.
- **Repeat:** Repeat the merge and update steps until all points are in a single cluster.

# Introduction to hierarchical clustering

## Advantages and Disadvantages

### Advantages

- No need to specify the number of clusters in advance.
- Dendrogram provides a visual representation of the data's structure.
- Can capture complex nested structures.

### Disadvantages

- Computationally intensive, especially for large datasets.
- Sensitive to noise and outliers.
- The choice of linkage method can significantly affect the results.

# Introduction to hierarchical clustering

## Application

Biology: Clustering genes or species.

Marketing: Customer segmentation.

Image processing: Image segmentation.

Social network analysis: Community detection.

# Introduction to hierarchical clustering

## Manual Method

**Problem:** Clustering analysis with agglomerative algorithm

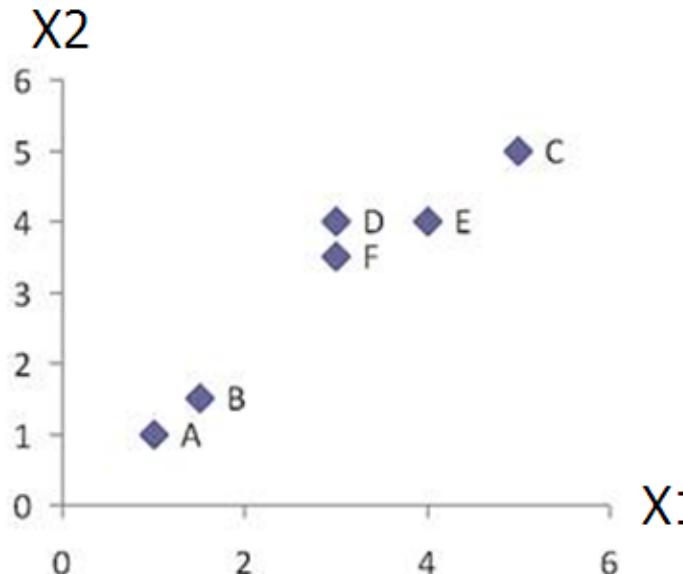
### How it Works:

- 1. Initialization:** Each data point is considered an individual cluster.
- 2. Distance Calculation:** A distance matrix is computed, representing the similarity between all pairs of clusters.
- 3. Merging:** The two closest clusters based on a chosen linkage criterion are merged into a new cluster.
- 4. Update:** The distance matrix is updated to reflect the newly formed cluster.
- 5. Iteration:** Steps 3 and 4 are repeated until all data points belong to a single cluster.

# Introduction to hierarchical clustering

## Example

**Problem:** clustering analysis with agglomerative algorithm



Euclidean :

$$d(x, y) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$$

$$d_{AB} = \sqrt{(1-1.5)^2 + (1-1.5)^2} = \sqrt{\frac{1}{2}} = 0.7071$$

$$d_{DF} = \sqrt{(3-3)^2 + (4-3.5)^2} = 0.5$$

**Euclidean distance**

	X1	X2
A	1	1
B	1.5	1.5
C	5	5
D	3	4
E	4	4
F	3	3.5

**Data Matrix**

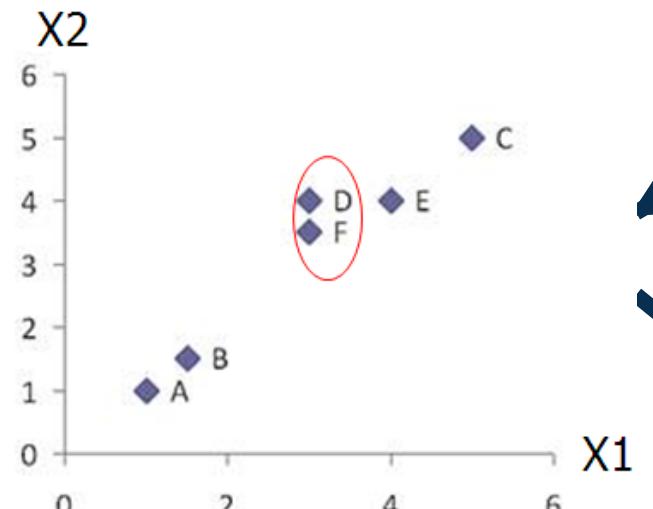
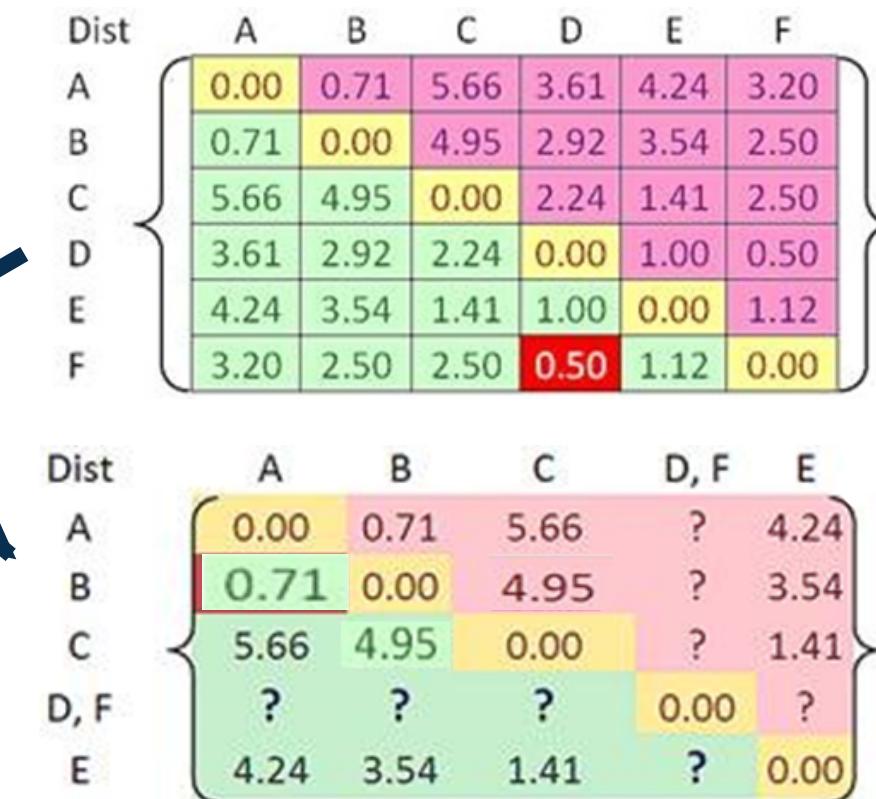
Dist	A	B	C	D	E	F
A	0.00	0.71	5.66	3.61	4.24	3.20
B	0.71	0.00	4.95	2.92	3.54	2.50
C	5.66	4.95	0.00	2.24	1.41	2.50
D	3.61	2.92	2.24	0.00	1.00	0.50
E	4.24	3.54	1.41	1.00	0.00	1.12
F	3.20	2.50	2.50	0.50	1.12	0.00

**Distance Matrix**

# Introduction to hierarchical clustering

## Example

- Merge two closest clusters (iteration 1)

Distance matrices illustrating the hierarchical clustering process:

Initial Distance Matrix (Dist):

	A	B	C	D	E	F
A	0.00	0.71	5.66	3.61	4.24	3.20
B	0.71	0.00	4.95	2.92	3.54	2.50
C	5.66	4.95	0.00	2.24	1.41	2.50
D	3.61	2.92	2.24	0.00	1.00	0.50
E	4.24	3.54	1.41	1.00	0.00	1.12
F	3.20	2.50	2.50	0.50	1.12	0.00

Merge Step 1 (Dist):

	A	B	C	D, F	E
A	0.00	0.71	5.66	?	4.24
B	0.71	0.00	4.95	?	3.54
C	5.66	4.95	0.00	?	1.41
D, F	?	?	?	0.00	?
E	4.24	3.54	1.41	?	0.00

# Introduction to hierarchical clustering

## Example

- Update distance matrix (iteration 1)

Dist	A	B	C	D	E	F
A	0.00	0.71	5.66	3.61	4.24	3.20
B	0.71	0.00	4.95	2.92	3.54	2.50
C	5.66	4.95	0.00	2.24	1.41	2.50
D	3.61	2.92	2.24	0.00	1.00	0.50
E	4.24	3.54	1.41	1.00	0.00	1.12
F	3.20	2.50	2.50	0.50	1.12	0.00

$d_{(D,F) \rightarrow A} = \min(d_{DA}, d_{FA}) = \min(3.61, 3.20) = 3.20$   
 $d_{(D,F) \rightarrow B} = \min(d_{DB}, d_{FB}) = \min(2.92, 2.50) = 2.50$   
 $d_{(D,F) \rightarrow C} = \min(d_{DC}, d_{FC}) = \min(2.24, 2.50) = 2.24$   
 $d_{B \rightarrow (D,F)} = \min(d_{BD}, d_{BF}) = \min(1.00, 1.12) = 1.00$

Min Distance (Single Linkage)

Dist	A	B	C	D, F	E
A	0.00	0.71	5.66	?	4.24
B	0.71	0.00	4.95	?	3.54
C	5.66	4.95	0.00	?	1.41
D, F	?	?	?	0.00	?
E	4.24	3.54	1.41	?	0.00

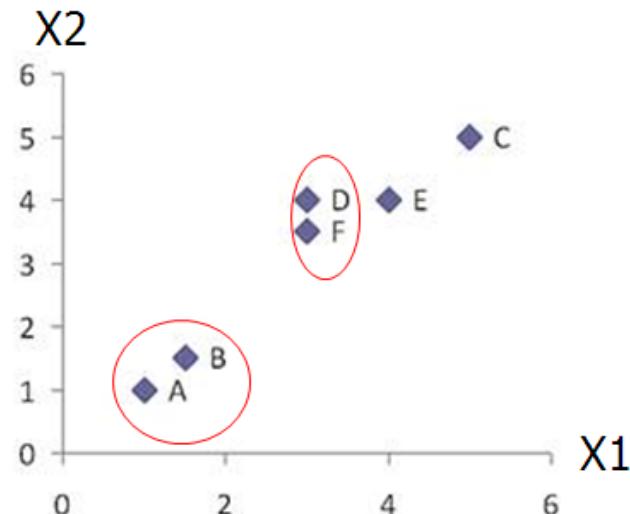
Dist

	A	B	C	D, F	E
A	0.00	0.71	5.66	3.20	4.24
B	0.71	0.00	4.95	2.50	3.54
C	5.66	4.95	0.00	2.24	1.41
D, F	3.20	2.50	2.24	0.00	1.00
E	4.24	3.54	1.41	1.00	0.00

# Introduction to hierarchical clustering

## Example

- Merge two closest clusters (iteration 2)



Min Distance (Single Linkage)

Dist	A	B	C	D, F	E
A	0.00	0.71	5.66	3.20	4.24
B	0.71	0.00	4.95	2.50	3.54
C	5.66	4.95	0.00	2.24	1.41
D, F	3.20	2.50	2.24	0.00	1.00
E	4.24	3.54	1.41	1.00	0.00

Dist	A,B	C	(D, F)	E
A,B	0	?	?	?
C	?	0	2.24	1.41
(D, F)	?	2.24	0	1.00
E	?	1.41	1.00	0

# Introduction to hierarchical clustering

## Example

- Update distance matrix (iteration 2)

Min Distance (Single Linkage)

Dist	A	B	C	D, F	E
A	0.00	0.71	5.66	3.20	4.24
B	0.71	0.00	4.95	2.50	3.54
C	5.66	4.95	0.00	2.24	1.41
D, F	3.20	2.50	2.24	0.00	1.00
E	4.24	3.54	1.41	1.00	0.00

$d_{C \rightarrow (A,B)} = \min(d_{CA}, d_{CB}) = \min(5.66, 4.95) = 4.95$   
 $d_{(D,F) \rightarrow (A,B)} = \min(d_{DA}, d_{DB}, d_{FA}, d_{FB}) = \min(3.61, 2.92, 3.20, 2.50) = 2.50$   
 $d_{E \rightarrow (A,B)} = \min(d_{EA}, d_{EB}) = \min(4.24, 3.54) = 3.54$

Dist	A,B	C	(D, F)	E
A,B	0	?	?	?
C	?	0	2.24	1.41
(D, F)	?	2.24	0	1.00
E	?	1.41	1.00	0

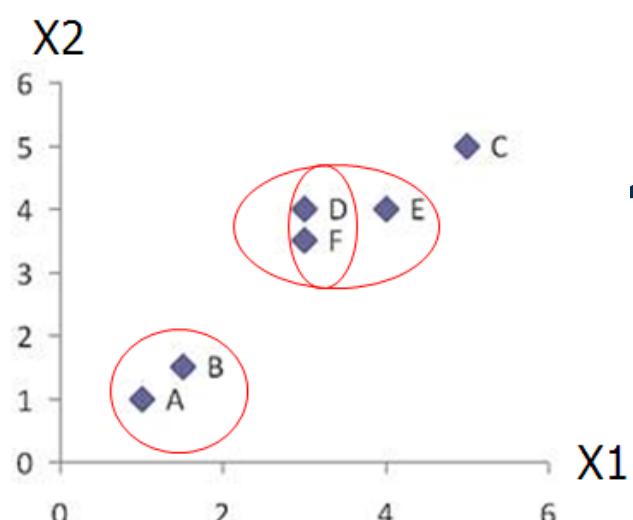
Min Distance (Single Linkage)

Dist	A,B	C	(D, F)	E
A,B	0	4.95	2.50	3.54
C	4.95	0	2.24	1.41
(D, F)	2.50	2.24	0	1.00
E	3.54	1.41	1.00	0

# Introduction to hierarchical clustering

## Example

- Merge two closest clusters/update distance matrix (iteration 3)



Min Distance (Single Linkage)

Dist	A,B	C	(D, F)	E
A,B	0	4.95	2.50	3.54
C	4.95	0	2.24	1.41
(D, F)	2.50	2.24	0	1.00
E	3.54	1.41	1.00	0

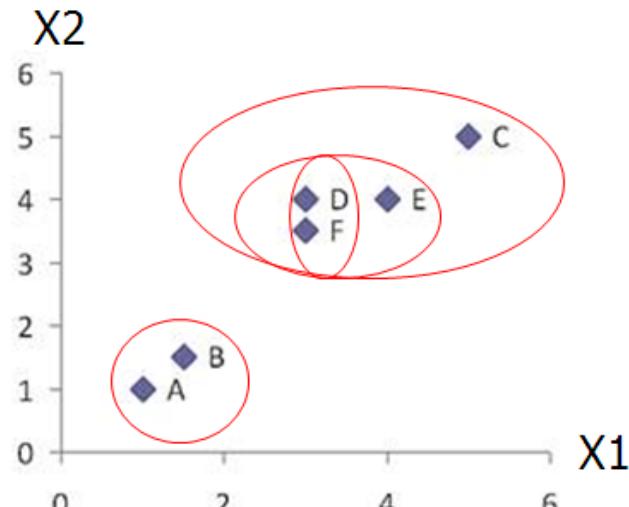
Min Distance (Single Linkage)

Dist	(A,B)	C	(D, F), E
(A,B)	0.00	4.95	2.50
C	4.95	0.00	1.41
(D, F), E	2.50	1.41	0.00

# Introduction to hierarchical clustering

## Example

- Merge two closest clusters/update distance matrix (iteration 4)



Min Distance (Single Linkage)

Dist	(A,B)	C	(D,F), E
(A,B)	0.00	4.95	2.50
C	4.95	0.00	1.41
(D,F), E	2.50	1.41	0.00

Min Distance (Single Linkage)

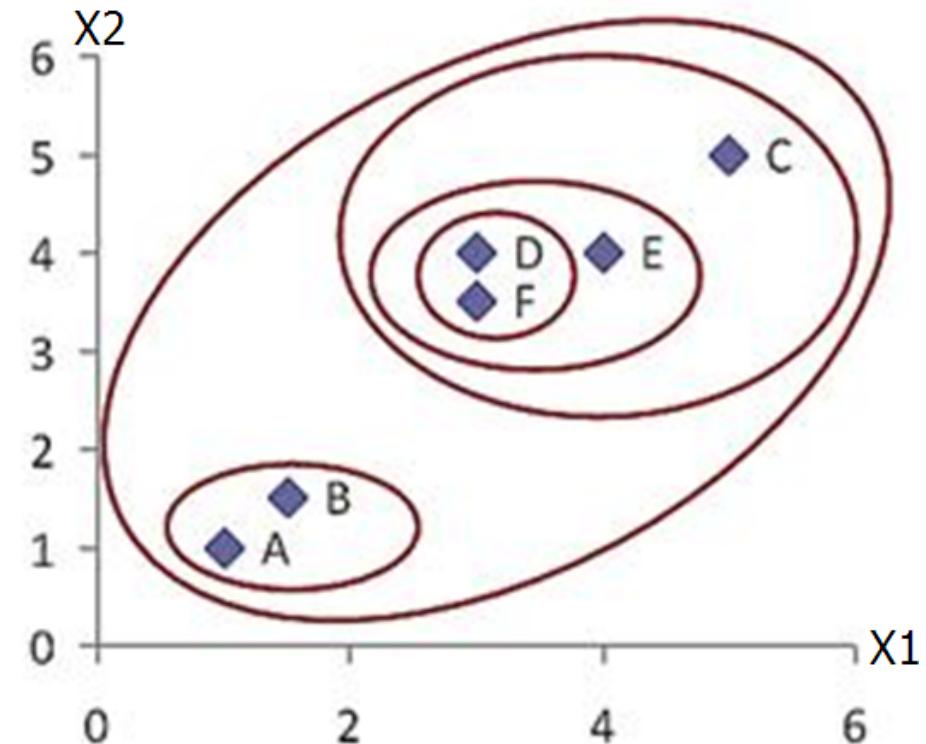
Dist	(A,B)	((D,F), E), C
(A,B)	0.00	2.50
((D,F), E), C	2.50	0.00

# Introduction to hierarchical clustering

## Example

- Result (meeting termination condition)

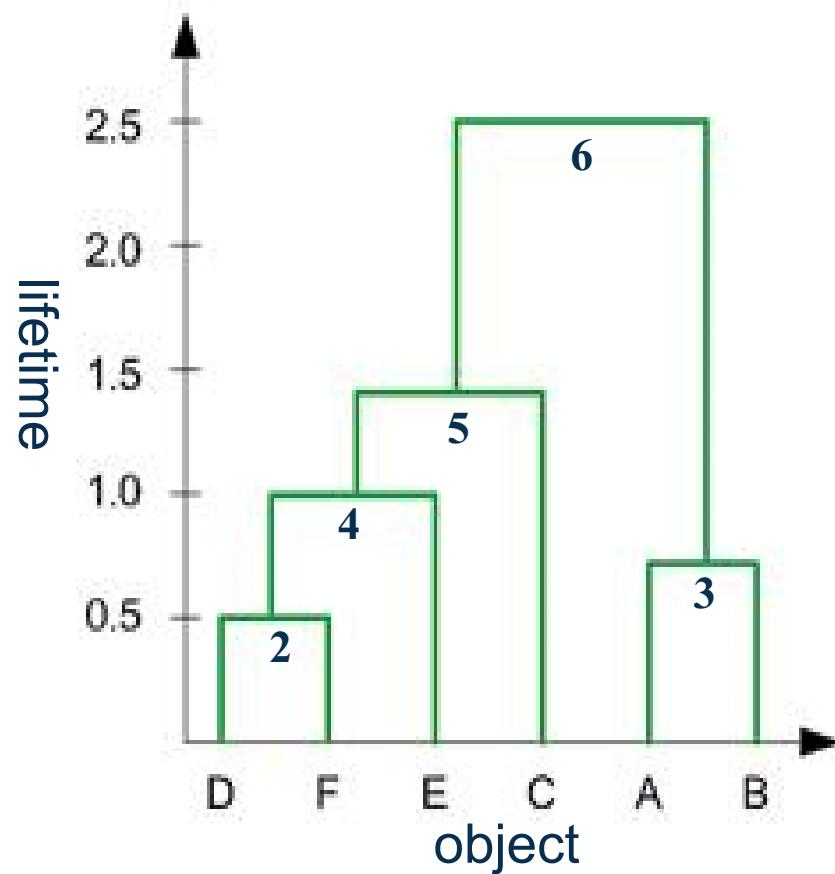
	X1	X2
A	1	1
B	1.5	1.5
C	5	5
D	3	4
E	4	4
F	3	3.5



# Introduction to hierarchical clustering

## Example

- Dendrogram tree representation



1. In the beginning we have 6 clusters: A, B, C, D, E and F
2. We merge clusters D and F into cluster (D, F) at distance 0.50
3. We merge cluster A and cluster B into (A, B) at distance 0.71
4. We merge clusters E and (D, F) into ((D, F), E) at distance 1.00
5. We merge clusters ((D, F), E) and C into (((D, F), E), C) at distance 1.41
6. We merge clusters (((D, F), E), C) and (A, B) into ((((D, F), E), C), (A, B)) at distance 2.50
7. The last cluster contain all the objects, thus conclude the computation

# Hierarchical Clustering

## Python Implementation

### Import library

```
from scipy.cluster.hierarchy import dendrogram, linkage  
from scipy.cluster.hierarchy import fcluster
```

Or

```
from sklearn.cluster import AgglomerativeClustering
```

### # Perform hierarchical clustering using Agglomerative Clustering

```
agg_clustering = AgglomerativeClustering(n_clusters=3, metric =  
'euclidean', linkage = 'ward')  
clusters = agg_clustering.fit_predict(data_scaled)
```

# Review Questions

1. Explain the difference between agglomerative and divisive hierarchical clustering. Provide an example use case for each.
2. Discuss the concept of linkage methods (single, complete, average, centroid) in hierarchical clustering. When might you choose one over another?
3. What are the primary advantages and disadvantages of hierarchical clustering compared to other clustering algorithms (e.g., k-means)?
4. Outline the steps involved in implementing hierarchical clustering on a given dataset. What factors should be considered when selecting distance metrics and linkage methods?

# Summary / Recap of Main Points

- Hierarchical clustering creates a hierarchy of clusters through either an agglomerative (bottom-up) or divisive (top-down) approach.
- Dendograms visually represent the clustering process, aiding interpretation.
- Linkage methods (single, complete, average, centroid) determine how to measure distance between clusters and influence clustering results.
- Hierarchical clustering is valuable for exploratory data analysis, uncovering hierarchical relationships, and applications like customer segmentation and document clustering.
- The process involves calculating distances, merging or splitting clusters based on linkage criteria, and creating a dendrogram.

# What To Expect Next Week

## In Class

- Metrics evaluation

## Preparation for Class

- Metrics evaluation

CX016-2.5-3-IML - Introduction to Machine Learning

## **Metrics evaluation**

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understand the suitable metrics/evaluation measures
2. Perform evaluation measures suitably.

# Contents & Structure

1. Suitable metrics/evaluation measures
2. Perform evaluation measures suitably.

# Recap From Last Lesson

1. Explain the difference between agglomerative and divisive hierarchical clustering. Provide an example use case for each.
2. Discuss the concept of linkage methods (single, complete, average, centroid) in hierarchical clustering. When might you choose one over another?
3. What are the primary advantages and disadvantages of hierarchical clustering compared to other clustering algorithms (e.g., k-means)?
4. Outline the steps involved in implementing hierarchical clustering on a given dataset. What factors should be considered when selecting distance metrics and linkage methods?

# Metrics evaluation

- Evaluation measures for classification and regression models are critical for assessing the performance of these models.
- Choosing the right metric depends on the specific problem, the desired outcome, and the characteristics of the data.

## Classification (TV = Binary / Multilevel)

- Accuracy, Error Rate, Confusion matrix etc.
- Precision
- Recall
- F1 Score, ROC, AUC values

## Regression (TV = Numeric Continuous)

- R<sup>2</sup>
- MSE – Mean Square Error
- RMSE – Root Mean Square Error
- MAE – Mean Absolute Error

# Metrics evaluation

## Classification Models

- **Accuracy:**
  - Definition: The ratio of correctly predicted instances to the total instances.
  - Formula: Accuracy =  $\frac{TP+TN}{TP+TN+FP+FN}$
  - Usage: Best when the classes are balanced.

**ERR - Error rate**

**ACC - Accuracy**

Calculate using  
Confusion Matrix



```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)*100, '%'
```

$$ERR = \frac{FP + FN}{FP + FN + TP + TN} = 1 - ACC$$

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$

# Metrics evaluation

## Classification Models

- **Precision:**
  - Definition: The ratio of true positive predictions to the total positive predictions (true positives and false positives).  
  
**% of selected items that are correct. This tells when you predict something, how many times they were actually correct.**
  - Formula: Precision =  $\frac{TP}{TP+FP}$
  - Usage: Important when the cost of false positives is high. **Precision is more important than Recall**

Precision →

```
from sklearn.metrics import precision_score,  
recall_score, f1_score  
precision_score(y_test, y_pred, average='micro')
```

# Metrics evaluation

## Classification Models

- **Recall:**

- Definition: The ratio of true positive predictions to the total actual positives.

- Formula: Accuracy =  $\frac{TP}{TP+FN}$

**% of correct items that are selected. This tells out of actual data, how many times you predicted correctly.**

- Usage: Important when the cost of false negatives is high.

Recall →

```
from sklearn.metrics import precision_score,  
recall_score, f1_score  
recall_score(y_test, y_pred, average='micro')
```

# Metrics evaluation

## Classification Models

- **F1-Score:**
  - Definition: The harmonic mean of precision and recall.
  - Formula: Precision =  $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$
  - Usage: Useful when you need to balance precision and recall.

		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

```
from sklearn.metrics import precision_score,
recall_score, f1_score
f1_score(y_test, y_pred, average='micro')
```

**F1 Score** is the weighted average of **Precision** and **Recall**. Therefore, this score takes both false positives and false negatives into account.

# Metrics evaluation

## Classification Models

- **Confusion Matrix:**

- Definition: A matrix that shows the number of true positives, true negatives, false positives, and false negatives.
- Usage: Provides a detailed breakdown of prediction results.

		Predicted class		Actual Values	
		P	N	Positive (1)	Negative (0)
Actual Class	P	True Positives (TP)	False Negatives (FN)	TP	FP
	N	False Positives (FP)	True Negatives (TN)	FN	TN

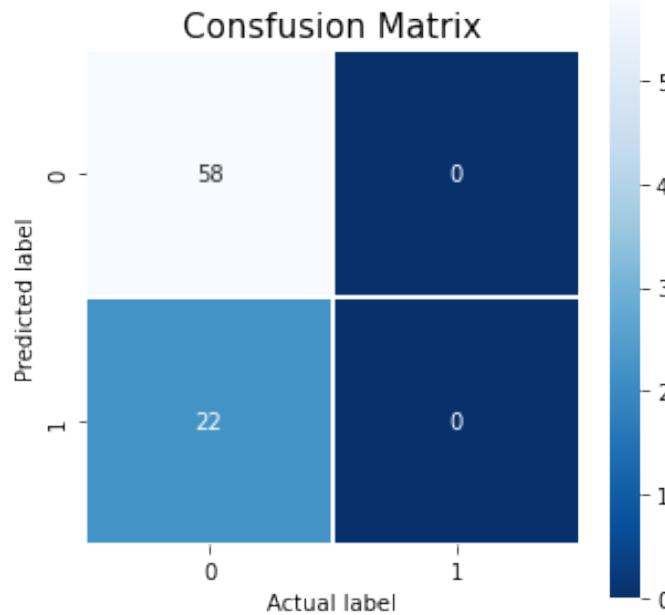
### Confusion matrix

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

# Metrics evaluation

## Classification Models

- Confusion Matrix:



```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, linewidths=.5, square = True, cmap = 'Blues_r');
plt.xlabel('Actual label');
plt.ylabel('Predicted label');
plt.title("Confusion Matrix", size = 15);
```

# Metrics evaluation

## Classification Models

- Classification Report:

```
from sklearn.metrics import classification_report
classification_report(y_test, y_pred)
```

Classification Report:				
	precision	recall	f1-score	support
0	0.72	1.00	0.84	58
1	0.00	0.00	0.00	22
accuracy			0.73	80
macro avg	0.36	0.50	0.42	80
weighted avg	0.53	0.72	0.61	80

# Metrics evaluation

## Classification Models

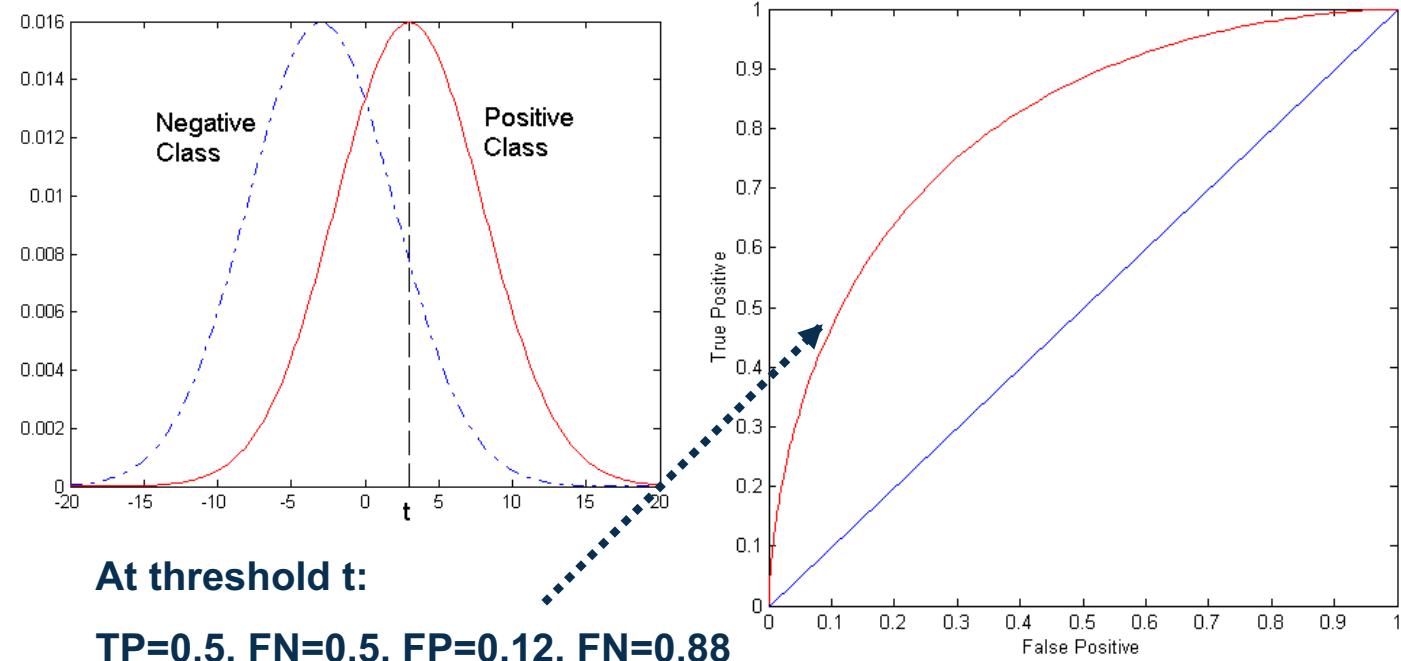
- **AUC-ROC (Area Under the Receiver Operating Characteristic Curve):**
  - Definition: A plot of the true positive rate (recall) against the false positive rate (1 - specificity) at various threshold settings.
  - Usage: Evaluates the model's ability to discriminate between positive and negative classes.
  - Characterize the trade-off between positive hits and false alarms
  - ROC curve plots TP (on the y-axis) against FP (on the x-axis)
  - Performance of each classifier represented as a point on the ROC curve
  - changing the threshold of algorithm, sample distribution or cost matrix changes the location of the point

# Metrics evaluation

## Classification Models

- **AUC-ROC (Area Under the Receiver Operating Characteristic Curve):**

- 1-dimensional data set containing 2 classes (positive and negative)
- any points located at  $x > t$  is classified as positive



# Metrics evaluation

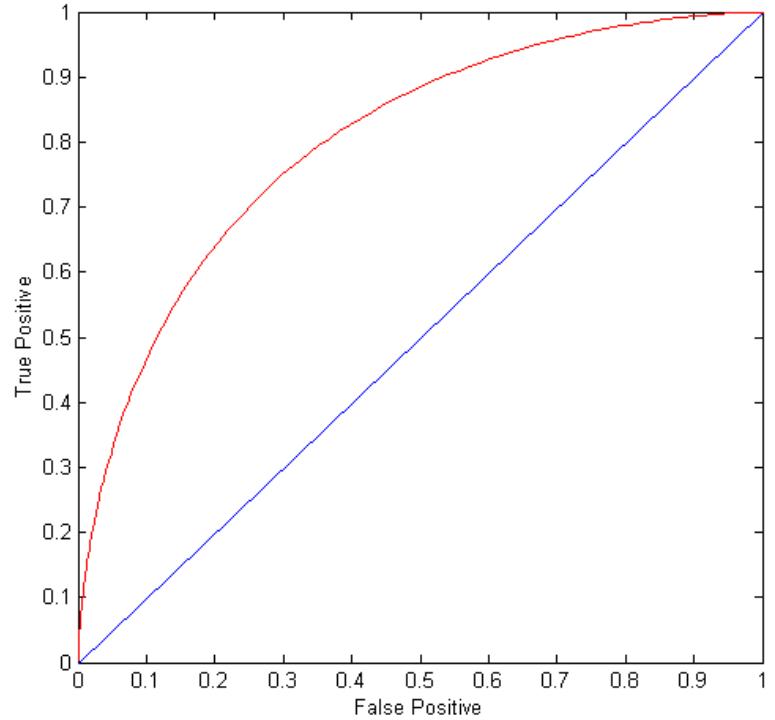
## Classification Models

- **AUC-ROC (Area Under the Receiver Operating Characteristic Curve):  
(TP,FP):**

- (0,0) : declare everything to be negative class
- (1,1) : declare everything to be positive class
- (1,0) : ideal

- **Diagonal line:**

- Random guessing
- Below diagonal line:
  - prediction is opposite of the true class



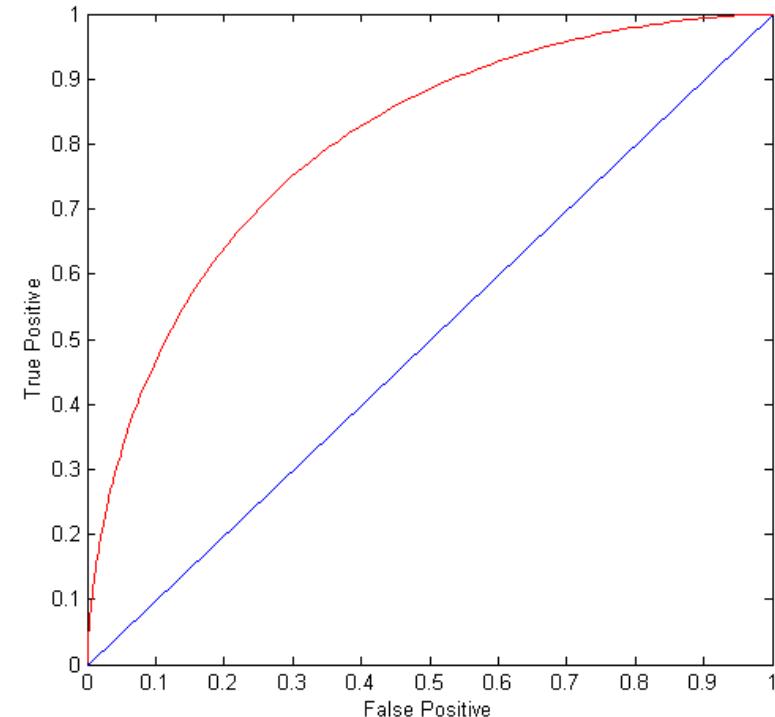
# Metrics evaluation

## Classification Models

- **AUC-ROC (Area Under the Receiver Operating Characteristic Curve):**

```
def plot_roc (fpr, tpr, threshold):
    auc = roc_auc_score(y_te, y_pred_2)
    plt.figure(1)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr, label='(area = {:.3f})'.format(auc))
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title('ROC curve')
    plt.legend(loc='best')
    plt.show()

fpr, tpr, thresholds = roc_curve(y_te, y_pred_2)
plot_roc(fpr, tpr, thresholds)
```



# Metrics evaluation

## Regression Models

- **Mean Absolute Error (MAE):**
  - Definition: The average of the absolute differences between predicted and actual values.
  - Formula:  $\text{MAE} = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$
  - Usage: Provides a straightforward interpretation of prediction errors.
- **Mean Squared Error (MSE):**
  - Definition: The average of the squared differences between predicted and actual values.
  - Formula:  $\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$
  - Usage: Penalizes larger errors more heavily.

# Metrics evaluation

## Regression Models

- **Root Mean Squared Error (RMSE):**
  - Definition: The square root of the mean squared error.
  - Formula:  $\text{RMSE} = \sqrt{\text{MSE}}$
  - Usage: Maintains the same units as the target variable, making it more interpretable.
- **R-squared ( $R^2$ ):**
  - Definition: The proportion of variance in the dependent variable that is predictable from the independent variables.
  - Formula:  $1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2}$
  - Usage: Indicates how well the independent variables explain the variability of the dependent variable.

# Metrics evaluation

## Regression Models

- **Adjusted R-squared:**
  - Definition: Adjusts the  $R^2$  for the number of predictors in the model.
  - Formula:  $1 - \frac{(1-R^2)(N-1)}{N-k-1}$
  - Usage: Useful for comparing models with different numbers of predictors.

# Metrics evaluation

## Regression Models

R<sup>2</sup>

MSE – Mean Square Error

RMSE – Root Mean Square Error

MAE – Mean Absolute Error

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
mean_squared_error(y_test, y_pred)
mean_absolute_error(y_test, y_pred)
np.sqrt(mean_squared_error(y_test, y_pred))
```

# Review Questions

1. What is the importance of understanding suitable metrics/evaluation measures before performing them?
2. Elaborate on the different types of metrics/evaluation measures that can be used in various contexts.

# Summary / Recap of Main Points

- Relevance: Choose metrics that directly align with the problem's objectives and the desired outcomes.
- Diversity: Employ a variety of metrics to comprehensively assess model performance from different angles.
- Contextualization: Consider the specific characteristics of the data and the problem domain when selecting metrics.

# What To Expect Next Week

## In Class

- Training process

## Preparation for Class

- Training process

CX016-2.5-3-IML - Introduction to Machine Learning

## **Training (Model Building) Process**

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understand the model building/training process
2. Recognize and perform model building process

# Contents & Structure

1. Model building/training process
2. Recognize and perform model building process

# Recap From Last Lesson

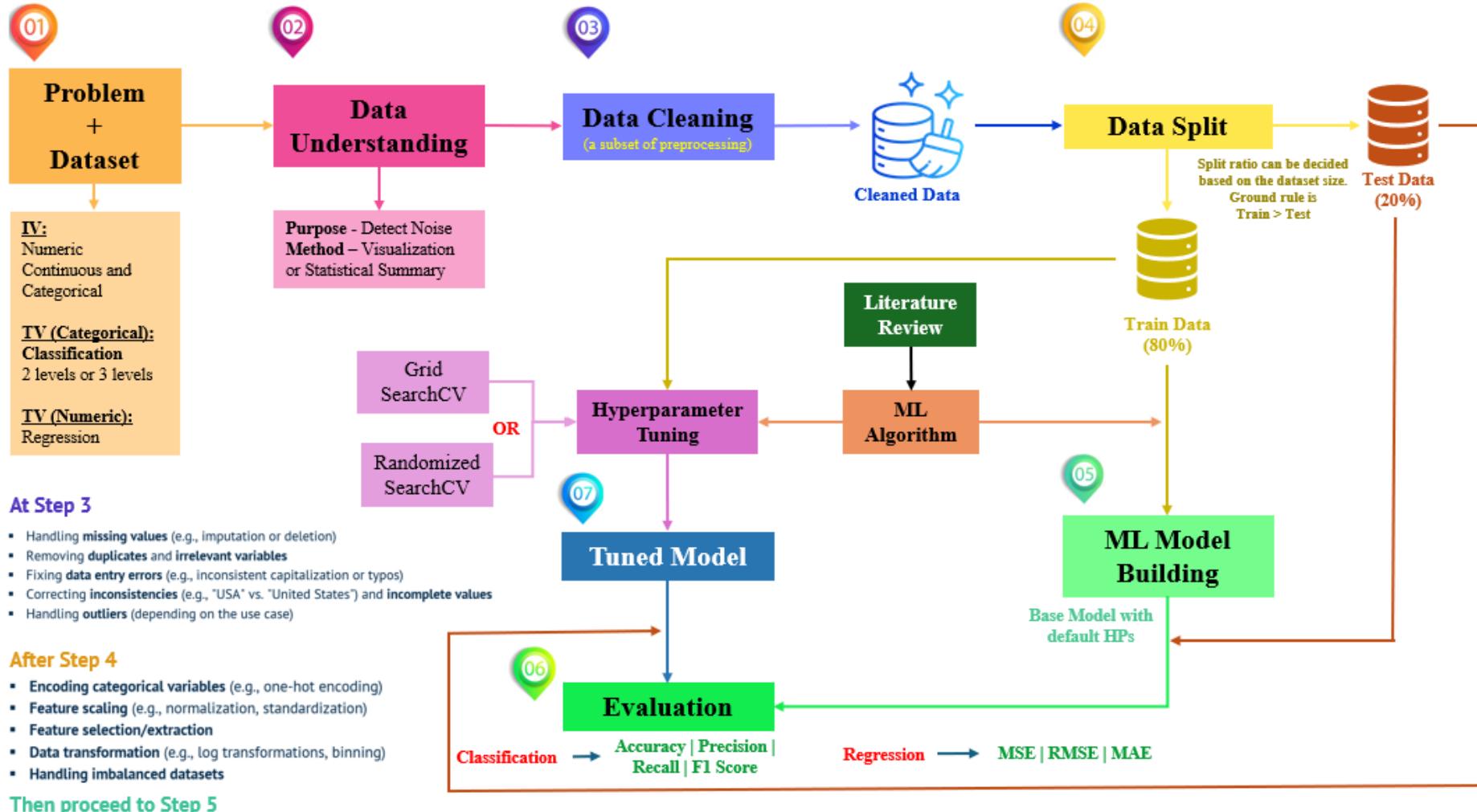
1. What is the importance of understanding suitable metrics/evaluation measures before performing them?
2. Elaborate on the different types of metrics/evaluation measures that can be used in various contexts.

# Training Process

- Building a machine learning model is a multi-step process that involves several key stages, from data collection to model deployment.

# Model Building Process

## ML Predictive Model Building Pipeline



# Model Building Process

## Problem definition/understanding

- The business problem should be clearly defined or understood to plan out the data science project.
- The problem definition/understanding should also be supported with the suitable set of data.
- The data should be finalized in terms of suitable variables, no of variables, no of records etc.

# Model Building Process

## Data loading

Load the data using the Pandas library. Pandas library can read/write the following data file formats.

Format				
Type	Data Description	Reader	Writer	
text	CSV	<a href="#">read_csv</a>	<a href="#">to_csv</a>	
text	Fixed-Width Text File	<a href="#">read_fwf</a>		
text	JSON	<a href="#">read_json</a>	<a href="#">to_json</a>	
text	HTML	<a href="#">read_html</a>	<a href="#">to_html</a>	
text	LaTeX			<a href="#">Styler.to_latex</a>
text	XML	<a href="#">read_xml</a>	<a href="#">to_xml</a>	
text	Local clipboard	<a href="#">read_clipboard</a>	<a href="#">to_clipboard</a>	
binary	MS Excel	<a href="#">read_excel</a>	<a href="#">to_excel</a>	

# Model Building Process

## Data loading

Load the data using the Pandas library. Pandas library can read/write the following data file formats.

binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle

# Model Building Process

## Data loading

Load the data using the Pandas library. Pandas library can read/write the following data file formats.

```
data = pd.read_csv('file path')
```

```
data = pd.read_excel('file path', sheet_name = 'Sheet1')
```

SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

# Model Building Process

## Data understanding

The data understanding can be done by using different libraries such as

1. Pandas
2. Matplotlib / Seaborn

The purpose of this task is to understand the data and to detect **noise** such as

1. Missing values
2. Outliers
3. Normal / not normal distribution
4. Duplicate values etc.
5. Incomplete and inconsistent data

# Model Building Process

## Data understanding

Also, the data can be explored using the visualization libraries of python. This would facilitate studying towards a business perspective.

The Exploratory Data Analysis / Descriptive Analytics will always help to explore / reveal the hidden patterns / insights of the data which will be useful for the decision makers.

The results of the data understanding would lead to data preprocessing.

# Model Building Process

## Data preprocessing

- This is a crucial task of any data science project which would consume 80% of the total project time.
  - This starts with the data cleaning tasks.
  - The issues detected during the data understanding phase will help to decide what kind of preprocessing activities to carry out.
1. Missing values → Must be imputed
  2. Outliers → Must be treated → either be removed or transformed
  3. Normal / not normal distribution → Normalized or Transformed
  4. Duplicate values, etc. → Must be removed
  5. Incomplete and inconsistent data → Must be handled
  6. Categorical data → Must be encoded
  7. Feature Selection → Correlation, Chi Square Test, R Square Test
  8. Class Balancing → Under sampling or Oversampling

**Preprocessing is done only if needed.**  
**Follow the order of tasks as per order provided in LAB 3 - Data Preprocessing**

# Model Building Process

## Data split

- The preprocessed data must be split (partitioned) as Train and Test subsets.
- The proportion of the Train and Test subsets can be as follows:
- The proportion is normally considered in percentages.
- Sklearn library has a function to perform this split while considering the random sampling aspect.

Train	Test
50	50
60	40
70	30
80	20

# Model Building Process

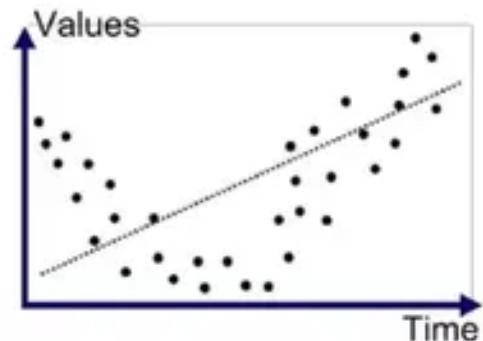
## Model building

- Sklearn library has several machine learning algorithms. The most suitable machine learning algorithm(s) can be selected to perform the model building.
- The algorithm selection can be done based on the Literature Review and Past work experience.
- Usually more than one machine learning predictive models are built and evaluated to select the best one suitable for the requirement.

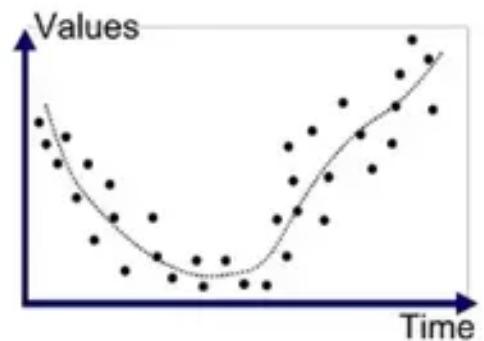
# Model Building Process

## Model building

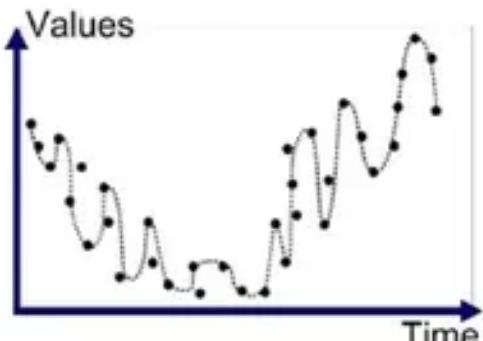
The machine learning problem can then be formalized as minimizing the error on the training set, say the squared error, while constraining the model to be simple.



Underfitted



Good Fit/Robust



Overfitted

# Model Building Process

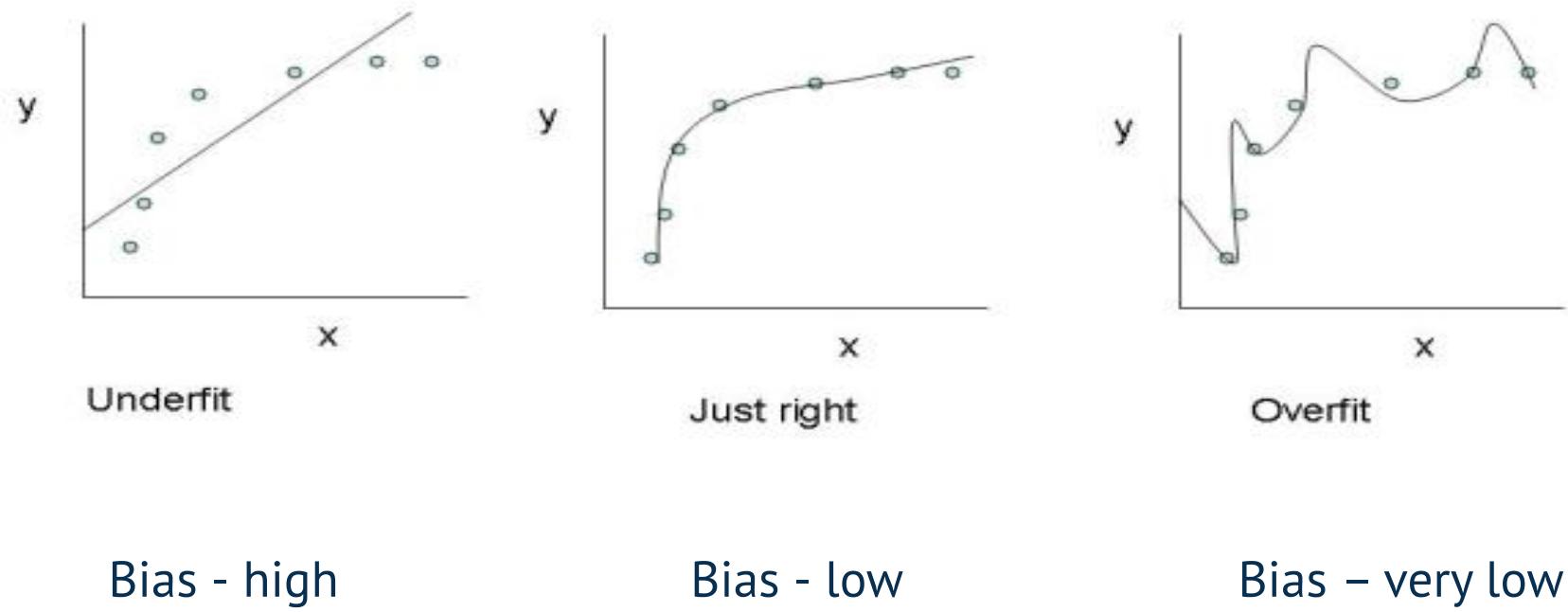
## Model building

- The concept in ML is:
  - use training set, a **sample** from a **population**
  - we are interested in **fitting a model to the population**
  - The **model** relates the attributes to a response/target variable.
- The model fit is to the population using sample data
- Hence, the model should be generic enough to represent the population.
  - However, this may result in an **underfit** (loose) between sample data and the model.
- However, if the model is **overfit** (tight fit) to the sample, there is a danger that it may not represent the population well.
  - Hence there is a need to relax it. This process is called **Regularization**.

# Model Building Process

## Model building

In a case of overfit, for the same population, different sample training sets produce different models.



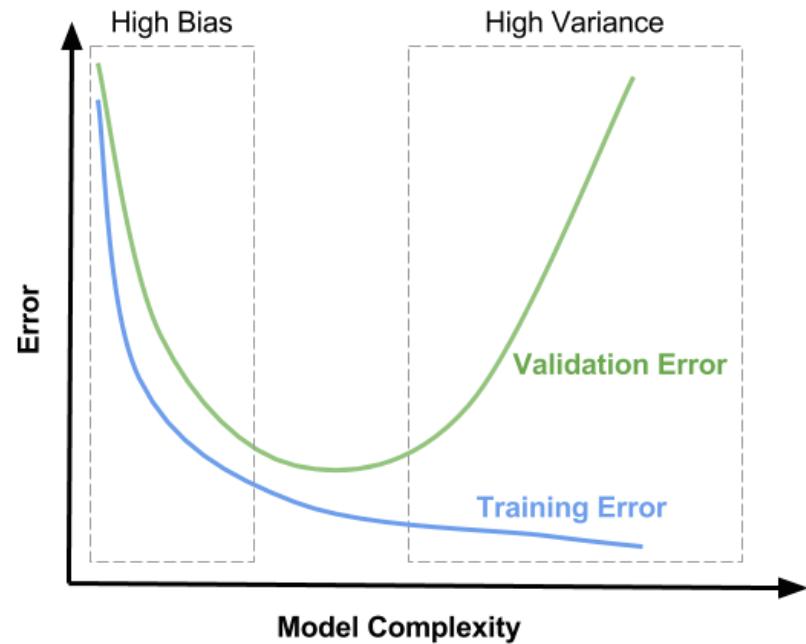
Variance is the measure of variation among the fits of different samples training sets on the same population

**Bias** – is the error in fitting a model

# Model Building Process

## Model building

- Increasing model complexity can decrease bias
  - However, it increases the variance
- A model that has very good fit on the training data:
  - This model may give minimum error on the training set and hence has low bias
  - However, it may not generalize well to the population.
  - Hence it gives higher error on validation set.



**There is a tradeoff between bias and variance**

# Model Building Process

## Model building

- **A high bias problem has the following characteristics**
  1. High training error.
  2. Validation/test error is similar in magnitude to the training error.
- **How to detect a high variance problem?**
  - A high variance problem on the other hand has the following characteristics
    - 1.Low training error
    - 2.Very high Validation/test error

# Model Building Process

## Model building

### To decrease bias

- More training data – may help
- increase model complexity:
  - Increase **number of features** so that the model has more information as in Linear regression
  - Increase model complexity through **model selection**:  
Nonlinear models,  
Increase number hidden layers in ANN

# Model Building Process

## Model evaluation

The predictive model must be evaluated to make the selection. The evaluation will be done based on the type of the predictive model.

Model → Classification (Target Variable is Categorical → Binary or Multilevel)

Accuracy, Precision, Recall, F1 Score, ROC value, AUC Value

Model → Regression (Target Variable is numeric continuous)

Mean Squared Error, Root Mean Squared Error,  
Mean Absolute Error

# Review Questions

1. What is the importance of understanding model training process?
2. Elaborate on the steps of model training process that can be used.

# Summary / Recap of Main Points

- Data Preparation: Data is cleaned, normalized, and augmented to enhance diversity and prepare it for model training.
- Training and Optimization: The model adjusts its parameters using optimization techniques like SGD or Adam to minimize a loss function over multiple iterations.
- Evaluation and Tuning: The model is evaluated on validation data, and techniques like cross-validation and hyperparameter tuning are used to improve performance before testing on unseen data.

# What To Expect Next Week

## In Class

- Cross-validation

## Preparation for Class

- Cross-validation

CX016-2.5-3-IML - Introduction to Machine Learning

## **Cross-validation**

# TOPIC LEARNING OUTCOMES

At the end of this topic, you should be able to:

1. Understand Cross Validation
2. Recognize different cross validation techniques

# Contents & Structure

1. Understand Cross Validation
2. Recognize different cross validation techniques

# Recap From Last Lesson

1. What is the importance of understanding model training process?
2. Elaborate on the steps of model training process that can be used.

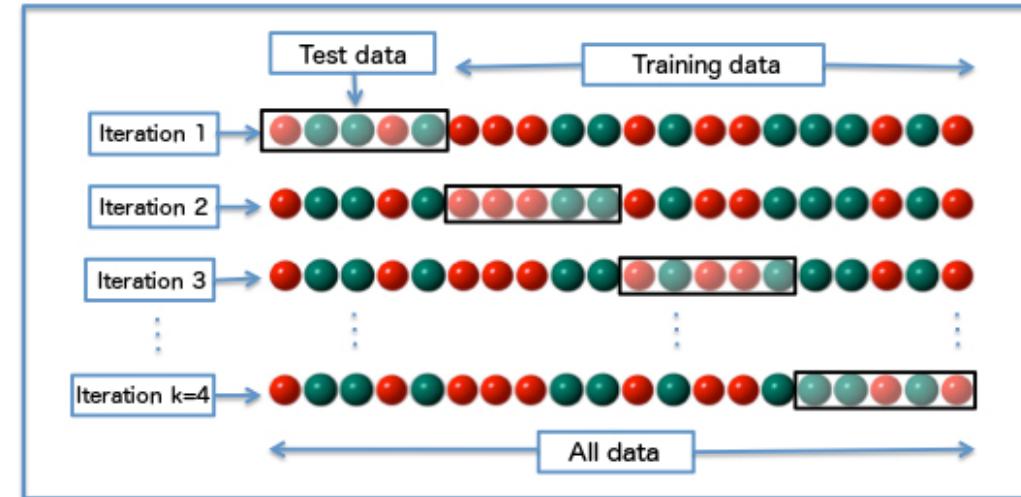
# Cross-validation

- Cross-validation assesses a model's generalizability by testing it on different data subsets, offering a reliable accuracy estimate.
- It helps balance bias and variance, reducing overfitting or underfitting, leading to a more robust model.
- K-Fold Cross-Validation: The dataset is split into k folds; the model is trained on  $k-1$  folds and tested on the remaining one, with results averaged for a comprehensive performance metric.

# Cross-validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

- **Cross-validation** is a technique to evaluate predictive models.
  - It partitions the original sample into a training set to train the model, and a test set to evaluate it.
- **Stratified Cross Validation** is used for **classification** problems.  
 Stratified k-fold cross-validation selects the folds so that each fold contains roughly the same proportions of class labels.



# Cross-validation

## Revisiting machine learning:

Main Objective of Machine Learning is to learn the relationship between input variables and the output response, for the universe of “population”

“Population” is the set of all combinations of inputs and outputs.

## Learning with samples

In practice it is not possible to have all these combinations available, to learn their relationship. Hence, we use samples from the population, to learn.

If there are sufficiently large number of samples, the learned relationship (model), closely represents the population.

# Cross-validation

However, in most cases we may not have large number of samples.

**Cross Validation** methods are commonly used to make best use of the available samples such that the learned model represents the true population, as closely as possible.

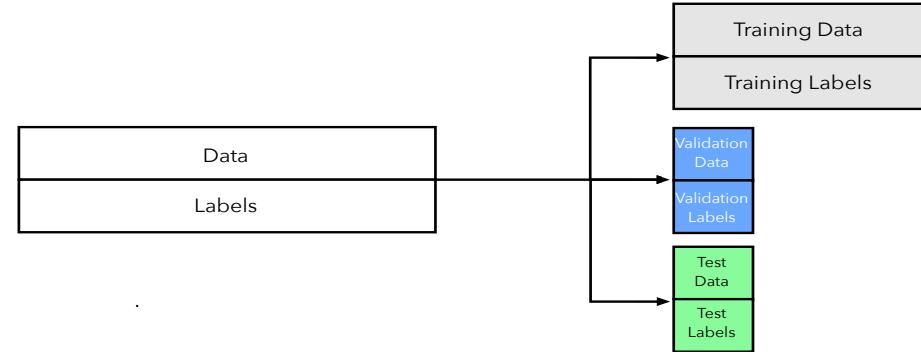
## Cross Validation Techniques

- Holdout cross validation
- K-fold cross validation
- Nested/Repeated Cross Validation

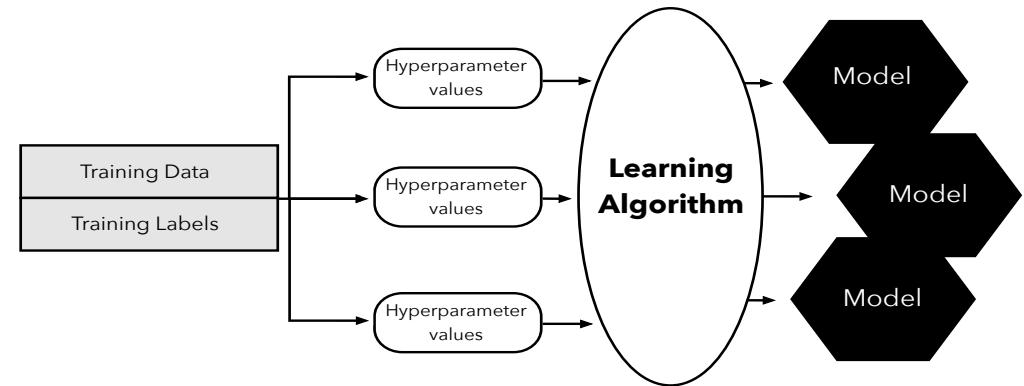
# Cross-validation

## Holdout Cross Validation

1



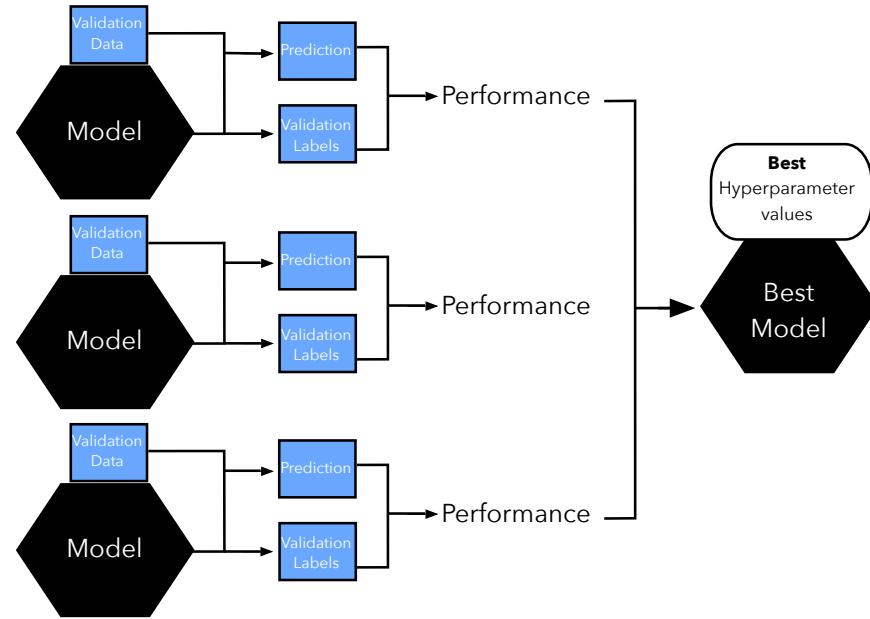
2



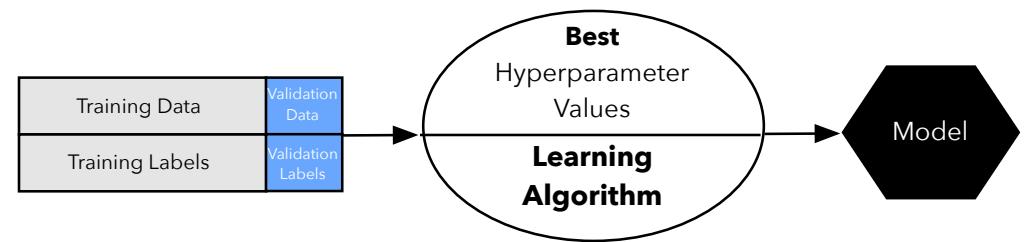
# Cross-validation

## Holdout Cross Validation

3



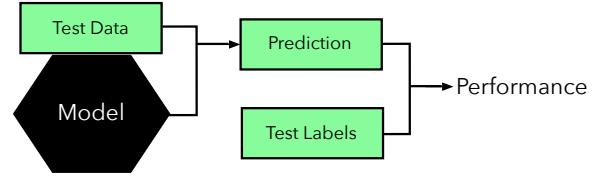
4



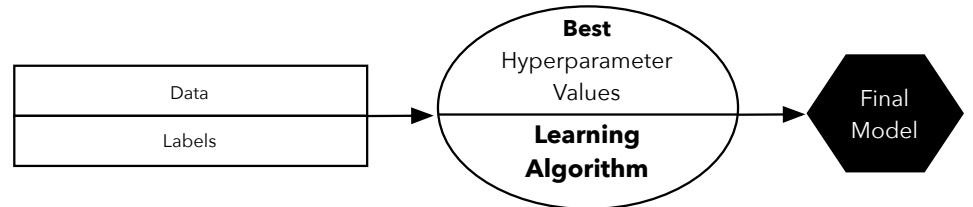
# Cross-validation

## Holdout Cross Validation

5



6

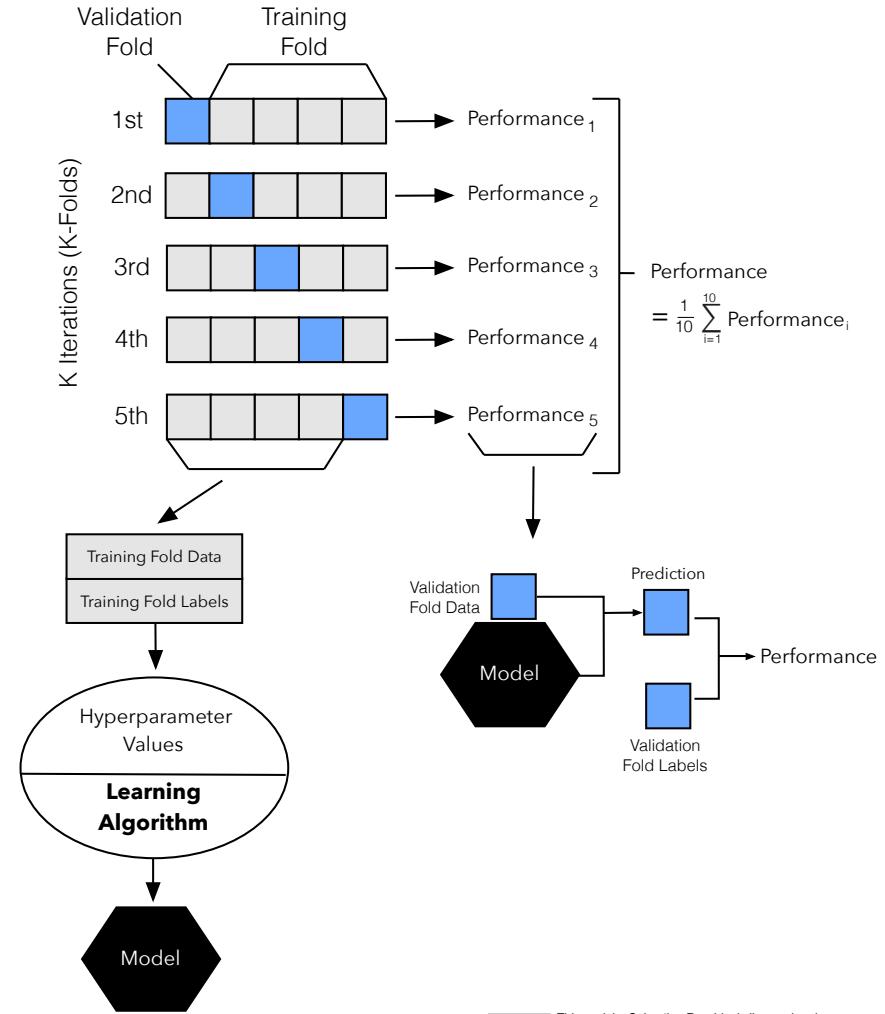


 This work by Sebastian Raschka is licensed under a Creative Commons Attribution 4.0 International License.

# Cross-validation

## K-Fold Cross Validation

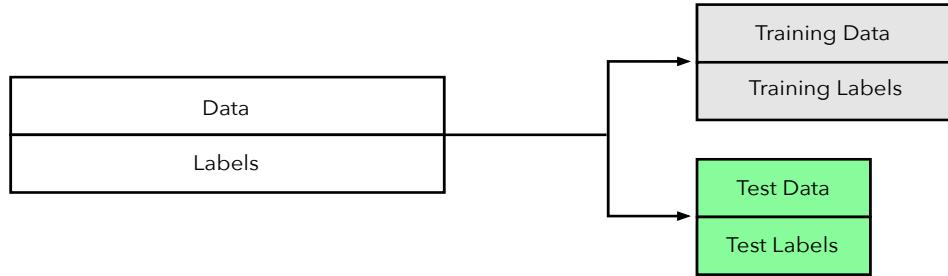
- Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.
- The parameter called k refers to the number of groups that a given data sample is to be split into.
- When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k = 5 becoming 5-fold cross-validation.



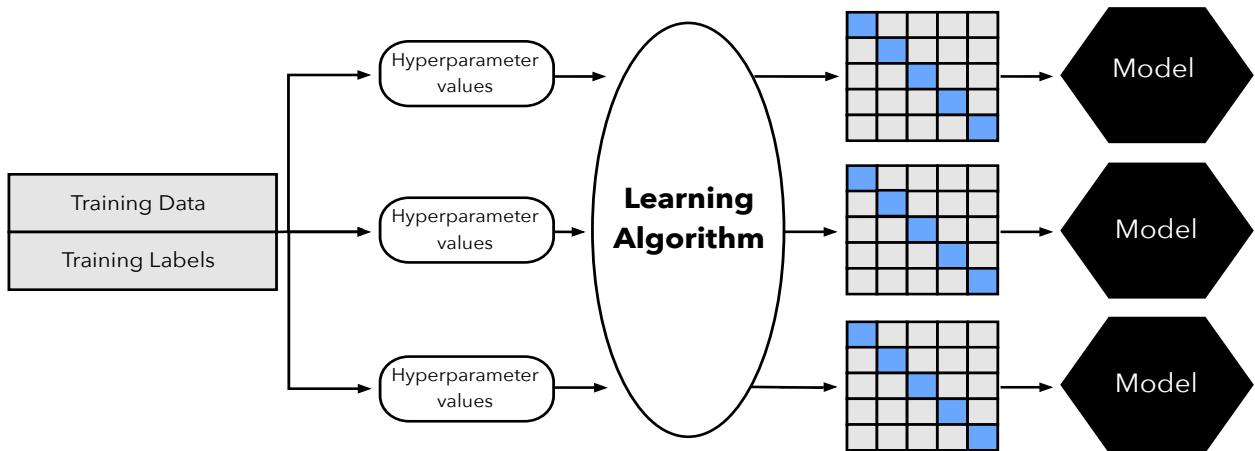
# Cross-validation

## K-Fold Cross Validation

1



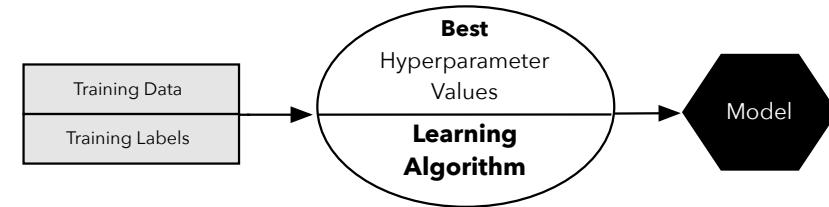
2



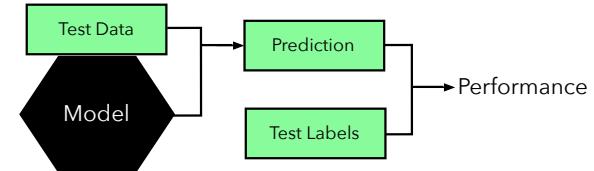
# Cross-validation

## K-Fold Cross Validation

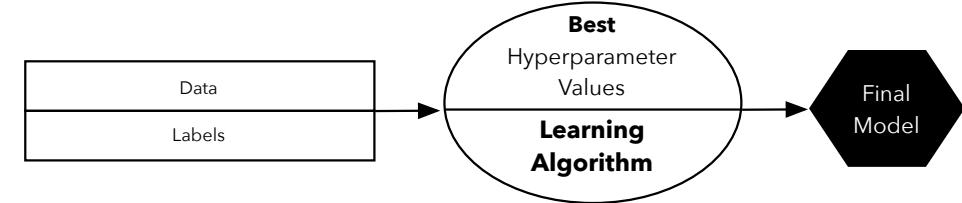
3



4



5



# Review Questions

1. What is the importance of Cross-Validation?
2. Elaborate on the steps of K-Fold Cross Validation.

# Summary / Recap of Main Points

- Model Validation: Cross-validation assesses a model's generalizability by testing it on different data subsets, offering a reliable accuracy estimate.
- Bias-Variance Balance: It helps balance bias and variance, reducing overfitting or underfitting, leading to a more robust model.
- K-Fold Cross-Validation: The dataset is split into  $k$  folds; the model is trained on  $k-1$  folds and tested on the remaining one, with results averaged for a comprehensive performance metric.

# What To Expect Next Week

## In Class

- End

## Preparation for Class

- End