CodeX: Machine Learning Syllabus!

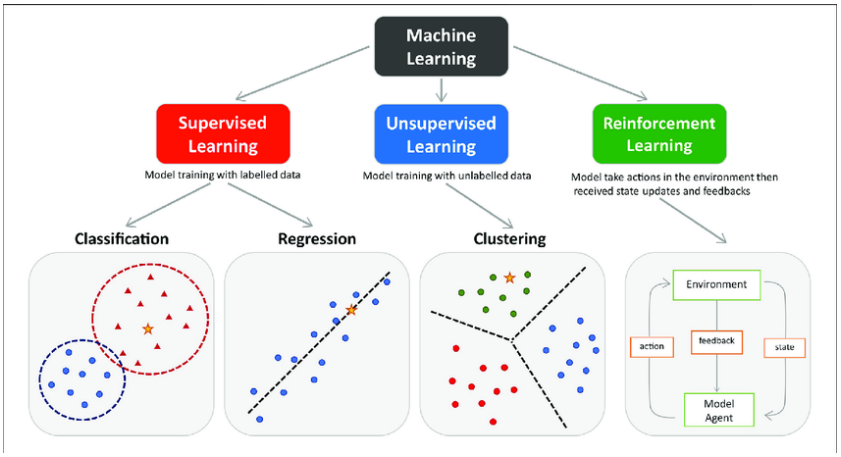
Overview of Machine Learning (ML) and its applications

What is ML?

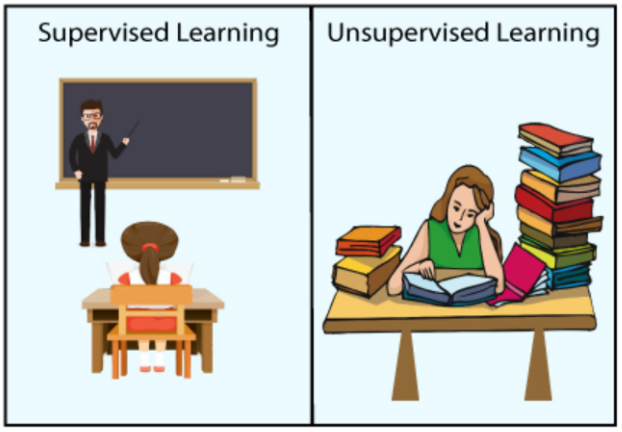
💡 Machine Learning (ML) is a really cool branch of artificial intelligence that focuses on **teaching computers to learn from** data and **make** smart decisions without needing to be told exactly what to do. It's like having a super smart friend who can analyze information and predict things without any explicit instructions. ML is used in a bunch of different areas like finance, healthcare, marketing, and even robotics to solve complex problems and help us make better decisions based on patterns and trends in data.

🔥 Add more fun looking stuff!
Make it relatable with famous example.

https://www.youtube.com/watch?v=f_uwKZIAeM0&ab_channel=OxfordSparks



Understanding supervised and unsupervised learning

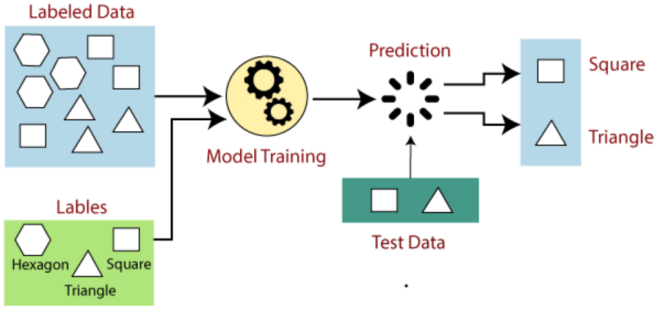


Supervised Learning

> 🔑 Supervised learning is a machine learning method in which **models are trained using labeled data**. The labelled data means some input data is already tagged with the correct output. The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y)**.

In the real-world, supervised learning can be used for **Risk Assessment, Image classification, Fraud Detection, spam filtering**, etc.

The working of Supervised learning can be easily understood by the below example and diagram:



Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

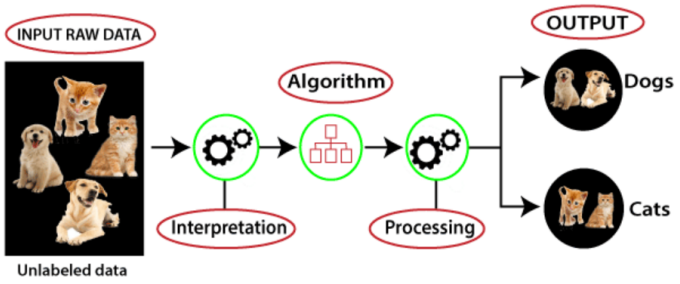
- If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
- If the given shape has three sides, then it will be labelled as a **triangle**.
- If the given shape has six equal sides then it will be labelled as **hexagon**.

Now, after training, we test our model using the test set, and the task of the model is to identify the given shape. The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

Unsupervised Learning

🔑 Unsupervised learning is another machine learning method in which **patterns inferred from the unlabeled input data**. The goal of unsupervised learning is to find the structure and patterns from the input data. Unsupervised learning **does not need any supervision**. Instead, it **finds patterns from the data by its own**.

Working of unsupervised learning can be understood by the below diagram:



Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

Comparison

Aspect	Supervised Learning	Unsupervised Learning
Objective	Predicts outcomes based on **labeled data**	Extracts patterns and structures from **unlabeled data**
Task	Classification, Regression, Ranking	Clustering, Dimensionality Reduction, Association
Use Cases	Predictive Modeling, Image Recognition, Speech Recognition	Customer Segmentation, Anomaly Detection, Topic Modeling
Advantages	1. **Precise** predictions with labeled data	1. Ability to **discover hidden** patterns and structures
	2. Well-understood, **easier to evaluate**	2. Effective in **exploratory data** analysis
	3. **Robust** and widely applicable	3. No need for labeled data, useful for **unknown patterns**
Disadvantages	1. Requires labeled data which can be **costly**	1. **Lack of clear evaluation** metrics in some cases
	2. **Terrible performance** in **new/unseen data.**	2. **Interpretability challenges** in complex models

Aspect	Supervised Learning	Unsupervised Learning
	3. **Sensitive** to outliers and **noise** in data	3. **Dependency** on the choice of **algorithms**

Session 1:Basic Programming (30 minute)

Wait, I don’t know how to code in the first place????!!?!!!!!??

> 💡 That's okay! You don't need to worry. **Learning to code is not as difficult** as it may seem at first. There are plenty of resources available online that can help you get started. You can begin by learning the basics of a programming language like Python, which is commonly used in machine learning. With practice and dedication, you'll be able to grasp the foundations of coding and start your journey into the exciting world of machine learning! We will teach you here in this epic crash course that will be beneficial for your rapport.

Introduction to Google Colab

When I stepped into the world of programming, the first thing I had to do was a lot of installation. This was quite an overwhelming process since I had to download a huge number of libraries(pre written code). These libraries would take both time and space on your local machine. Also, many data science and machine learning algorithms need a dedicated GPU to run the programs efficiently

Then I came across a platform called “Google Colab”. Google Colab which stands for Google Colaboratory is a cloud platform provided by Google to run Python Notebook’s without need to install anything on your local machine. Isn’t it cool? (yes I’m a nerd)

> Visit colab.google and try to follow through this introduction written by Google Teams. For more info, visit https://colab.research.google.com/notebooks/basic_features_overview.ipynb#

Features of Google Colab

Features of Google Colab	Description
Zero Setup Required	No need for software installation or setup on your local machine. Just open a web browser and start working.
Free GPU Access	Provides free access to GPU (Graphics Processing Units) and sometimes TPU (Tensor Processing Units).
Easy Collaboration	Facilitates seamless collaboration by allowing real-time sharing, viewing, and editing of notebooks.
Pre-installed Libraries	Comes with essential data science and machine learning libraries pre-installed, saving setup time.

Features of Google Colab	Description
Version Control and Integration	Easily integrates with GIT for version control. Can be linked to Google Drive for cloud-based storage.

Creating a New Notebook

Click on the “File” menu, then select “New Notebook” to create a new Python 3 notebook. You can rename your notebook by clicking on the current name at the top and entering a new one.

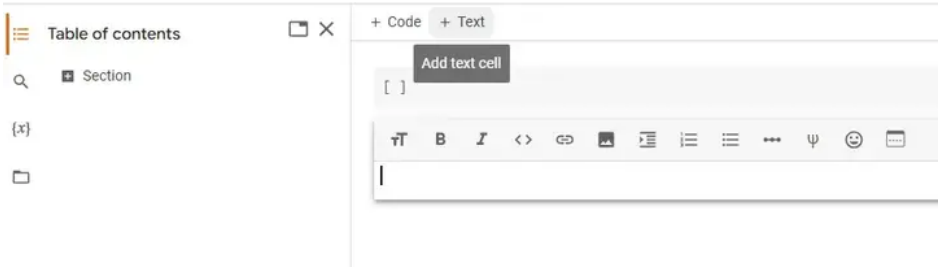
Working with Cells

> Cell’s are the building blocks of the Notebook. Google Colab notebooks also consist of cells. There can be two types of cells.

1. Code cells
2. Text (Markdown) cells.

Code cells are where you write and run your Python code. To add a new code cell, click the “+ Code” button on the top or use keyboard shortcuts (e.g., Ctrl+M B for).

Text (Markdown) cells are used for adding explanations, documentation, or formatted text. To add a new code cell, click the “+ Text” button on the top.



Running Code

Now let’s see, how to execute a code cell once we have added them to the notebook. As you can see below, you need click on the play button to run the code once you have added the python code in the code cell.

After the run is successful, you can see the green tick indicating the code run is successful and also you can see the results below the code cell.

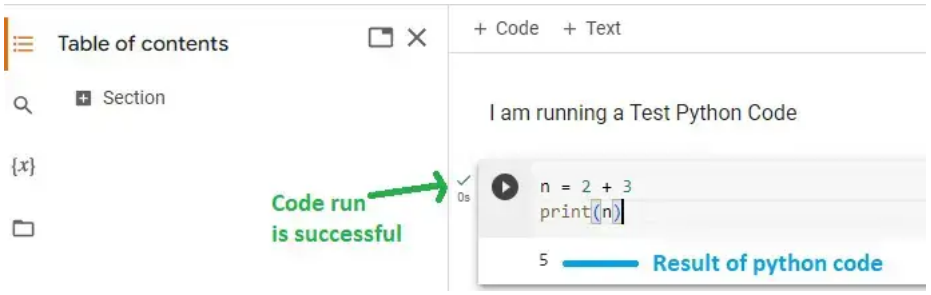
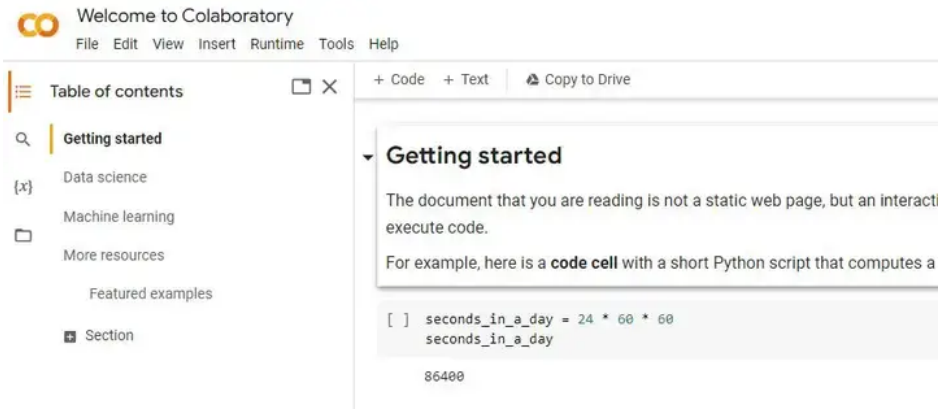


Table of Contents

Google colab allows you to use sections in the notebooks. This allows you to organize your notebook into distinct sections which makes it easier to navigate and scroll through as it grows in size. This can be particularly helpful when working with long or complex notebooks.



Managing Notebooks(Save your progress!)

Save your work by clicking “File” > “Save” or use Ctrl+S (Cmd+S on Mac). You can organize your notebooks into folders on Google Drive by clicking “File” > “Save a copy in Drive.”

Using GPU/TPU

To utilize Google Colab’s free GPU or TPU resources, go to “Runtime” > “Change runtime type” and select “GPU” or “TPU” as the hardware accelerator. Note that GPU/TPU resources are allocated per session and may have usage limits.

Collaboration

Share your notebook with collaborators by clicking “Share” in the upper right corner. You can grant various access levels, such as viewing, commenting, or editing permissions.

Importing Libraries

To import libraries that are not present, simply run the below commands followed by the library name, Libraries will get installed in the notebook virtual environment without taking up any space on your local machine.

```
!pip install <LIBRARY_NAME>
#Example --> pip install pygame
```

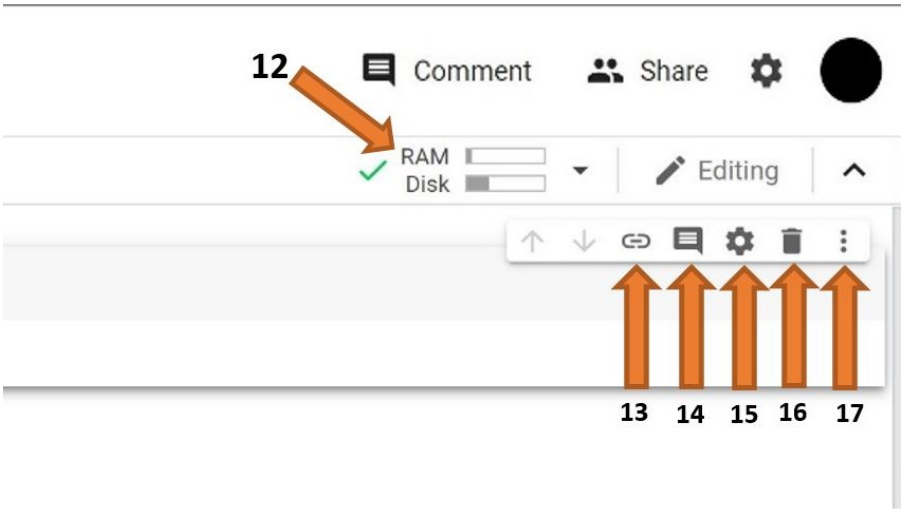
The UI

Create a new Colab notebook following the steps provided in the above section.

Before we start getting into the coding, let’s familiarize ourselves with the user interface (UI) of Google Colab.

What the different buttons mean:

1. **Files:** Here you will be able to upload datasets and other files from both your computer and Google Drive
2. **Code Snippets:** Here you will be able to find prewritten snippets of code for different functionalities like adding new libraries or referencing one cell from another.
3. **Run Cell:** This is the run button. Clicking this will run any code that is inserted in the cell beside it. You can use the shortcut shift+enter to run the current cell and exit to a new one.
4. **Table of Contents:** Here you will be able to create and traverse different sections inside of your notebook. Sections allow you to organize your code and improve readability.
5. **Menu Bar:** Like in any other application, this menu bar can be used to manipulate the entire file or add new files. Look over the different tabs and familiarize yourself with the different options. In particular, make sure you know how to upload or open a notebook and download the notebook (all of these options are under “File”).
6. **File Name:** This is the name of your file. You can click on it to change the name. Do not edit the extension (.ipynb) while editing the file name as this might make your file unopenable.
7. **Insert Code Cell:** This button will add a code cell below the cell you currently have selected.
8. **Insert Text Cell:** This button will add a text cell below the cell you currently have selected.
9. **Cell:** This is the cell. This is where you can write your code or add text depending on the type of cell it is.
10. **Output:** This is the output of your code, including any errors, will be shown.
11. **Clear Output:** This button will remove the output.



1. **Ram and Disk:** All of the code you write will run on Google's computer, and you will only see the output. This means that even if you have a slow computer, running big chunks of code will not be an issue. Google only allots a certain amount of Ram and Disk space for each *user*, so be mindful of that as you work on larger projects.
2. **Link to Cell:** This button will create a URL that will link to the cell you have selected.
3. **Comment:** This button will allow you to create a comment on the selected cell. Note that this will be a comment on (about) the cell and not a comment in the cell.
4. **Settings:** This button will allow you to change the Theme of the notebook, font type, and size, indentation width, etc.
5. **Delete Cell:** This button will delete the selected cell.
6. **More Options:** Contains options to cut and copy a cell as well as the option to add form and hide code.

Learn Programming with Python

💡 All of this is in 1 hour

Hello world!

Here is a sample line of code that can be executed in Python:

```
print("Hello, World!")
```

You can just as easily store a string(we will talk later) as a variable and then print it to the terminal:

```
teks = "Hello, World!"
print(teks)
```

The above code will print `Hello, World!` on your screen. Try it yourself!! Congratulations on your first code.

In Python, the `print()` function is used to display or output information to the standard output, which is typically the console or terminal.

In this case, the `print()` function is passed the string “Hello, World!” as its argument (the empty bracket). A string is a sequence of characters enclosed in quotation marks. The quotation marks can be single (`'`) or double (`"`), and in this case, double quotation marks are used.

Try changing the text into other thing:

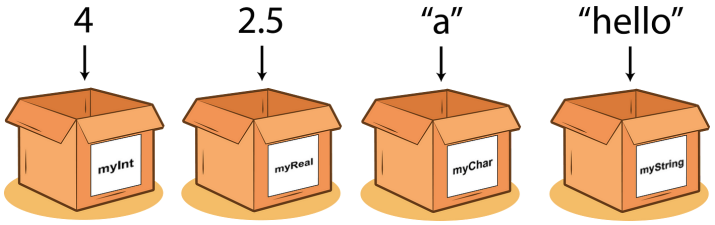
```
name=input()
print(name +' do not like waking up early')
```

```
Ali
Ali do not like waking up early
```

Except from the terrible grammar above , this example get the user input using `input()` code and store it into a variable named ‘name’. We will learn more about variables in next section.

Variables concept and naming convention

💡 In Python, a **variable** is a container for storing data values. It is like a labeled box where you can put information and give it a name. This name is used to refer to the stored value later in the program.



Rules for naming the variable:

1. Variable names can only contain letters (a-z, A-Z), numbers (0-9), and underscores (_).
2. They cannot start with a number.

3. Variable names are case-sensitive (`age` and `Age` would be different).
4. Avoid using Python reserved words like `print`, `if`, `else`, etc., as variable names.

Good Practices:

- Choose descriptive names that convey the purpose of the variable.
- Use lowercase letters with underscores for readability (e.g., `my_variable`).

Example:

```
# Assigning values to variables ---> this is a comment block which will not be detected by th
name, age, height= "Alice", 17, 1.65
is_student=True

# Displaying initial information using variables
print("Initial Information:")
print("Name:", name)
print("Age:", age)
print("Height:", height)
print("Is a Student:", is_student)

# Modifying variables
age = age + 1  # Increase age by 1
height = height + 0.05  # Increase height by 0.05

# Displaying updated information
print("\nUpdated Information:") #the \n is used to skip a line during printing.
print("Name:", name)
print("Age:", age)
print("Height:", height)

# Copying variables
copied_age, copied_height = age, height  # Multiple assignments
copied_age-=1 #reduce by 2
print("\nCopied Information:")
print("Copied Age:", copied_age)
print("Copied Height:", copied_height)

# Parsing and changing variable types
height_as_string = "1.75"
height = float(height_as_string)  # Parsing string to float
print("\nHeight as String:", height_as_string)
print("Height as Float:", round(height,1))

# Function that accepts parameters
def greet_person(person_name, person_age):
    print("\nGreeting Function:")
    print("Hello, {}! You are {} years old.".format(person_name, person_age))

# Passing variables as parameters to a function
greet_person(name, age)

# Modifying variables inside a function
def make_younger(person_age):
    person_age -= 1
    return person_age

# Displaying modified age
```

```
new_age = make_younger(age)
print("\nNew Age after Modification:", new_age)
```

Output:

```
Initial Information:
Name: Alice
Age: 17
Height: 1.65
Is a Student: True

Updated Information:
Name: Alice
Age: 18
Height: 1.7

Copied Information:
Copied Age: 17
Copied Height: 1.7

Height as String: 1.75
Height as Float: 1.8

Greeting Function:
Hello, Alice! You are 18 years old.

New Age after Modification: 17
```

Datatypes(focus also in calculation and operations)

💡 A data type is a classification or categorization that specifies which type of value a variable can hold in computer programming. It defines the characteristics of data that includes how it can be stored, manipulated, and the types of operations that can be performed on it.

Data Type	Example	Role	Purpose	Common Operations
int	`age = 25`	Represents integers.	Arithmetic operations on whole numbers.	`age + 5` returns `30`
float	`height = 1.75`	Represents floating-point numbers.	Precision calculations requiring decimals.	`height * 2` returns `3.5`
str	`name = "Alice"`	Represents text.	Working with textual information.	`name + " is a student"` returns `"Alice is a student"`
bool	`is_student = True`	Represents logical values.	Control flow and decision-making.	`not is_student` returns `False`
list	`grades = [90, 85, 92]`	Represents a sequence of elements.	Handling multiple values in one variable.	`grades[1]` returns `85`
tuple	`coordinates = (3, 4)`	Similar to a list but cannot be modified.	Storing data that should not be changed.	`coordinates[0]` returns `3`
dict	`person = {'name': 'John', 'age': 25}`	Represents a mapping of keys to values.	Efficient data retrieval using keys.	`person['age']` returns `25`
set	`unique_numbers = {1, 2, 3, 4}`	Represents a set in mathematics.	Storing and manipulating unique elements.	`unique_numbers.add(5)` modifies the set
NoneType	`result = None`	Represents the absence of a value.	Indicating the lack of a specific value.	`result is None` returns `True`

Notes : **Orange color are data structure**, is not a datatype but behave almost like container for other single data type! Datatype are like truck, and data are stored in uniquely named box (variable), where the truck have certain rules of storing box

```
# Machine Learning Data Types and Operations Example

# Numeric Operations
age = 25
height = 1.75
new_age = age + 5
adjusted_height = height * 1.1

# String Operations
text = "Machine Learning"
uppercase_text = text.upper()
substring = text[0:7]

# List Operations
data_points = [3, 7, 11, 5, 9]
average = sum(data_points) / len(data_points)
sorted_data = sorted(data_points)

# Dictionary Operations
student_info = {'name': 'Alice', 'age': 20, 'grade': 'A'}
student_info['grade'] = 'B'
keys = list(student_info.keys())

# Set Operations
unique_numbers = {1, 2, 3, 4, 4, 5}
unique_numbers.add(6)
intersection_set = unique_numbers.intersection({3, 4, 5, 6})

# Displaying Results
print("Numeric Operations:")
print("New Age:", new_age)
print("Adjusted Height:", adjusted_height)

print("\nString Operations:")
print("Uppercase Text:", uppercase_text)
print("Substring:", substring)

print("\nList Operations:")
print("Average:", average)
print("Sorted Data:", sorted_data)

print("\nDictionary Operations:")
print("Updated Grade:", student_info['grade'])
print("Dictionary Keys:", keys)

print("\nSet Operations:")
print("Updated Set:", unique_numbers)
print("Intersection Set:", intersection_set)
```

List

💡 In machine learning, effective handling and manipulation of data are crucial for building and training models. Different data structures are employed to organize and represent datasets. Here are some fundamental data structures commonly used in machine learning:

List is a versatile and mutable data structure that allows you to store and manipulate a collection of items. Here's an overview of lists and some common operations:

Creating a List:

You can create a list by enclosing elements within square brackets `[]`.

```
fruits = ['apple', 'banana', 'orange', 'grape']
```

Accessing Elements:

You can access individual elements in a list using their index. **Indexing starts from 0.**

```
# Accessing Elements
first_fruit = fruits[0]
print("First Fruit:", first_fruit)
last_fruit=fruits[-1]
print("Last fruit:",last_fruit)
```

Slicing:

Slicing allows you to extract a portion of the list.

```
# Slicing
subset_of_fruits = fruits[1:3]
print("Subset of Fruits:", subset_of_fruits)
```

Modifying Elements:

Lists are mutable, so you can modify elements by assigning new values.

```
# Modifying Elements
fruits[2] = 'mango'
print("Modified Fruits List:", fruits)
```

Adding Elements:

You can add elements to the end of the list using the `append()` method.

```
# Adding Elements
fruits.append('kiwi')
print("Updated Fruits List:", fruits)
```

Removing Elements:

You can remove elements by value or index.

```
# Removing Elements
fruits.remove('banana')
print("Fruits List after Removal:", fruits)

# Remove by Index
removed_fruit = fruits.pop(2)
print("Removed Fruit:", removed_fruit)
```

List Length:

You can find the number of elements in a list using the `len` function.

```
# List Length
num_fruits = len(fruits)
print("Number of Fruits:", num_fruits)
```

Checking Element Existence:

You can check if an element exists in a list using the `in` operator.

```
# Checking Existence
is_apple_present = 'apple' in fruits
print("Is Apple Present?", is_apple_present)
```

Concatenating Lists:

You can concatenate two lists using the `+` operator.

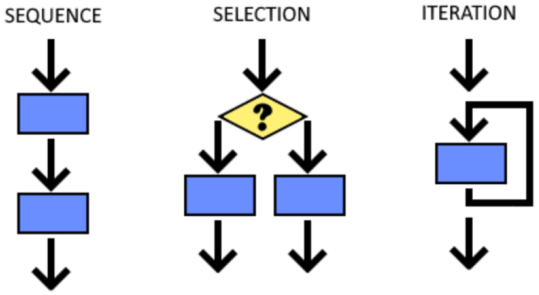
```
# Concatenating Lists
more_fruits = ['pineapple', 'watermelon']
combined_fruits = fruits + more_fruits
print("Combined Fruits List:", combined_fruits)
```

Sort lists:

You can sort list using `sort`() method

```
numbers = [5, 2, 8, 1, 6]
# Using sort() method
numbers.sort()
print("Sorted List (in-place):", numbers)
# Sorting in descending order
numbers.sort(reverse=True)
sorted_numbers_desc = sorted(numbers, reverse=True) # other style
print("Sorted List (Descending, in-place):", numbers)
print("Sorted List (Descending, new):", sorted_numbers_desc)
```

Control Sequence



- **Sequential**

> 💡 Statements are executed one after the other, in sequence, from top to bottom.

```
# Import necessary libraries
#pip install scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

# Generate some sample data
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Plot the results
plt.scatter(X_test, y_test, color='black', label='Actual Data')
plt.plot(X_test, y_pred, color='blue', linewidth=3, label='Linear Regression Model')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.legend()
plt.show()
```

Here is example of a sequential machine learning code. I know this looks intimidating but we will further discuss this code.In this example, we generate some synthetic data with a linear relationship, split it into training and testing sets, create a linear regression model, train the model using the training data, make predictions on the test data, and finally evaluate the

model using Mean Squared Error. The results are also plotted to visualize the linear regression line compared to the actual data. **Don’t bother to understand this now**, just understand the flow of the code

- **Selection**

> 💡 Allows the execution of different statements based on whether a condition is true or false(boolean)

1. If

```
# Example 1: If statement
age = int(input("Enter your age: "))
if age >= 13:
    print("You are eligible to join the school club!")
```

1. If-else

```
# Example 2: If-else statement
age = int(input("Enter your age: "))
if age >= 13:
    print("You are eligible to join the school club!")
else:
    print("Sorry, you are not eligible to join the school club.")
```

1. If-elif-else

```
# Example 3: If-elif statement
age = int(input("Enter your age: "))
if age < 13:
    print("Sorry, you are too young to participate.")
elif age <= 18:
    print("Congratulations! You are eligible to participate in the school event.")
else:
    print("Sorry, you are too old to participate. The event is for high school students only.
```

- **Looping + activity**

> 💡 **Allows repeatition** execution of a block of code while a **condition is true**.

1. **FOR loop**

Print numbers in a range.

```
# Example 1: For loop to print numbers in a range
 for i in range(5):
     print(i)
 for x in range(9,0,-1): #(start=9, stop=0, step=-1)
     print('x is ' + str(x))
```

Iterate over items in a list

```
# Example 2: For loop to iterate over a list
 fruits = ["apple", "banana", "cherry"]
 for fruit in fruits:
     print(fruit)
```

2. **While loop**

Print numbers until a certain condition is met.

```
# Example 3: While loop to print numbers until a condition is met
 count = 0
 while count < 5:
     print(count)
     count += 1
```

User input validation using a while loop.

```
# Example 4: While loop for user input validation
 user_input = ""
 while user_input.lower() != "yes":
     user_input = input("Do you want to continue? (yes/no): ")
```

3. **Nested loop**

```
# Example 5: Nested loops to create a multiplication table
 for i in range(1, 6):
     for j in range(1, 11):
         print(f"{i} x {j} = {i*j}")
```

4. **Loop control**

Break

```
# Example 6: Using break to exit a loop
 for i in range(10):
     if i == 5:
         break
     print(i)
```

Continue

```
# Example 7: Using continue to skip an iteration
 for i in range(10):
     if i == 5:
         continue
     print(i)
```

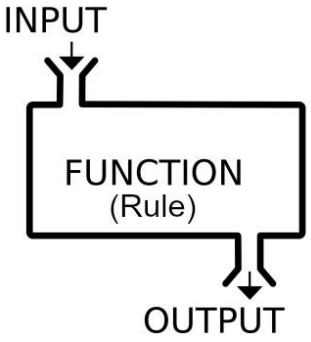
Fun Activity (Yes I want to drag your attention)

Here is code that looks like donut, try to run this code using Thonny!

```
while ('o_' in dir()) or (A := (0)
                ) or (B := 0) or print((
             "\x1b[2J")) or not ((sin := ((
          __import__('math'))).sin)) or not (
       cos := __import__('math').cos) or (o_ := (
      True)): [pr() for b in [[(func(), b) for ((z
     )) in [[0 for _ in range(1760)]] for b in [[ (
    "\n") if ii % 80 == 79 else " " for ii in range(
   1760)]] for o, D, N in [(o, D, N) for j in range((
  0), 628, 7) for i in range(0, 628, 2) for (c, d, e,(
 f), g, l, m, n) in [(sin(     i / 100), cos(j / 100),(
 sin(A)), sin(j / 100),          cos(A), cos(i / 100) ,
 cos(B), sin(B))] for              (h) in [d + 2] for (
```

```
D, t) in [ (1 / ( c                * h * e + f * g + (5
)), c * h * g - f *                 e)] for (x, y) in [
(int( 40 + 30 * D * (               l * h * m - t * n)),
int( 12 + 15 * D * ( l            * h * n + t * (m))))]
 for (o, N) in [(x + 80 *     y,int(8 * (( f * e -  c
  * d * g) * m - c * d * e - f * g - l * d * n)))] if
  0 < x < 80 and 22 > y > 0] if D > z[o] for func in
   [lambda: (z.pop((o))), lambda: z.insert((o), (D)
    ),(lambda: (b.pop( o))), lambda: b.insert((o),
     ".,-~:;=!*#$@"[ N if N > 0 else 0])]][ 0][1]
       ] for pr in [lambda: print("\x1b[H"),
         lambda: print("".join(b))] if (A :=
            A + 0.02) and ( B := B + 0.02)]
               #..-~~Ali Azeem CodeX:~~--.#
                    #...,2023,...#
```

Function and procedure (grasp understanding on parameters)



💡 A function in Python is a block of reusable code that performs a specific task. Functions can take parameters as input, and they can return a value

Feature	Function	Procedure
Return Value	Can return a value using the `return` statement.	Does not return a value.
Usage of return	Can have multiple `return` statements to return different values based on conditions.	Typically does not use `return` or uses it only for control flow, not for returning values.
Example	`def add(a, b):` `return a + b`	`def print_sum(a, b): print(a + b)`

Function

```
# Function with parameters
def greet(name):
    print(f"Hello, {name}!")

# Call the function with an argument
greet("Alice")
greet(name="Alice")
```

Procedure

```
# Procedure with parameters
def print_square(number):
```

```
    square = number ** 2
    print(f"The square of {number} is {square}.")

# Call the procedure with an argument
print_square(4)
```

Parameters:

- **Parameters** are variables that you define in the function or procedure definition to receive input values.
- **Arguments** are the actual values that you pass to the function or procedure when calling it.

In the examples above, `name` and `number` are parameters. When you call the function or procedure with specific values (like "Alice" or 4), these values are the arguments.

Understanding parameters is crucial because they allow you to make your functions and procedures more flexible. They enable you to reuse the same piece of code with different inputs.

Libraries

→In Python, a library (or module) is a collection of pre-written code and functionalities that can be reused by other programs. Libraries provide a set of tools and functions that help programmers perform common tasks without having to write the code from scratch.

Useful Libraries in machine learning:

> You can do your own self research to determine useful and perfect libraries to used in your ML project:

Library	Description
NumPy	Provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.
SciPy	Built on NumPy, it offers additional functionality for optimization, integration, interpolation, eigenvalue problems, and more.
Pandas	A powerful library for data manipulation and analysis, offering data structures like DataFrame for easy cleaning, preprocessing, and analysis of datasets.
Matplotlib	A 2D plotting library for creating static, animated, and interactive visualizations.
Seaborn	Built on Matplotlib, it provides a high-level interface for drawing attractive and informative statistical graphics.
Scikit-learn	A comprehensive machine learning library with tools for classification, regression, clustering, dimensionality reduction, and model selection.
TensorFlow	Developed by Google, an open-source deep learning library widely used for building and training neural networks.
PyTorch	Developed by Facebook, another popular deep learning library known for its dynamic computation graph and ease of use.
Keras	A high-level neural networks API that runs on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit, simplifying the process of building and training deep learning models.
Scrapy	An open-source and collaborative web crawling framework for extracting data from websites, often used for creating datasets.
NLTK (Natural Language Toolkit)	A library for working with human language data, providing tools for tokenization, stemming, tagging, parsing, and more.

Library	Description

Installation

Before using a library, you need to make sure it is installed(colab will auto install for you). You can use the `pip` command to install libraries. Open your terminal or command prompt and type:

```
pip install library_name
```

Import and libraries utilization

Once the library is installed, you can use the `import` statement to make its functionality available in your Python script.

```
# Standard import
import math

# Import with alias
import pandas as pd

# Importing specific functions/classes
from datetime import datetime

# Importing multiple functions/classes
from random import randint, choice

# Using the imported functions and classes
result = math.sqrt(25)
df = pd.DataFrame()
current_time = datetime.now()
random_number = randint(1, 10)
random_choice = choice(['apple', 'banana', 'cherry'])

# Displaying the results
print("Square Root:", result)
print("Pandas DataFrame:", df)
print("Current Time:", current_time)
print("Random Number and Choice:", random_number, random_choice)
```

When you import a library in Python, you often use the **dot** `.` notation to access elements (such as functions, classes, or constants) within that library. The dot notation is used to navigate the library's structure.

File handling in Python

> 💡 In both machine learning and general programming, reading and writing to files are essential tasks. In machine learning, these operations enable the loading, preprocessing, and feature engineering of datasets, as well as the storage and retrieval of trained models. We will use pandas to

In this course, we will focus on using **Comma-Seperated Value** (CSV)file type for handling our data.It is a simple and widely used file format for storing tabular data (numbers and text) in plain text form, where each line of the file represents a row of the table, and the values in each row are separated by commas (or other delimiters).

Below is example of CSV file:

```
Name,Age,Salary
Alice,25,50000
Bob,30,60000
Charlie,22,45000
```

It is equivalent to this in spreadsheet(like Microsoft Excel):

	A	B	C
1	Name	Age	Salary
2	Alice	25	50000
3	Bob	30	60000
4	Charlie	22	45000
5			

Imagine a CSV file like a digital table, kind of like a spreadsheet you might use in class. Each row in the table represents information about something (let's say people, like classmates or friends), and each column has a specific type of information (like names, ages, and salaries).

Read CSV file using pandas

```
import pandas as pd

# Reading a CSV file into a DataFrame
file_path = 'machine_learning_data.csv'
try:
    df = pd.read_csv(file_path)
    print("Dataset Loaded Successfully:")
    print(df.head())
except FileNotFoundError:
    print(f"The file '{file_path}' does not exist.")
except Exception as e:
    print(f"An error occurred: {e}")
```

Write CSV file using pandas

```
# Creating a simple DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 22],
        'Salary': [50000, 60000, 45000]}

df_to_write = pd.DataFrame(data)

# Writing the DataFrame to a CSV file
output_file_path = 'output_machine_learning_data.csv'
try:
    df_to_write.to_csv(output_file_path, index=False)
    print(f"Data has been written to '{output_file_path}'.")
except Exception as e:
    print(f"An error occurred: {e}")
```

Extra Pandas note for reference

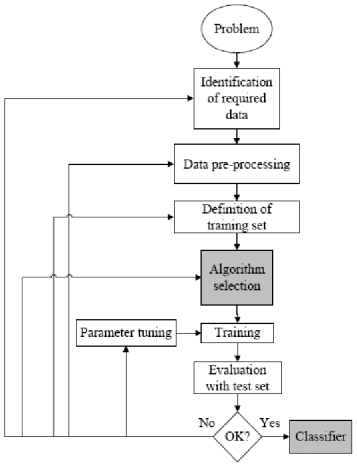
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# ----------------------1. Loading Data----------------------
# Reading a CSV file
df = pd.read_csv('your_dataset.csv')

# Displaying the first few rows
print(df.head())

# ----------------------2. Exploratory Data Analysis (EDA)----------------------
# Display basic statistics of the data
print(df.describe())

# Checking for missing values
print(df.isnull().sum())

# Count unique values in a column
print(df['column_name'].value_counts())

# Selecting specific columns
selected_columns = df[['column1', 'column2']]

# ----------------------3. Handling Missing Values----------------------
# Drop rows with missing values
df_cleaned = df.dropna()

# Fill missing values with a specific value
df_filled = df.fillna(value)

# Interpolate missing values
df_interpolated = df.interpolate()

# ----------------------4. Filtering Data----------------------
# Filtering rows based on a condition
df_filtered = df[df['column'] > threshold]

# Using multiple conditions
df_filtered = df[(df['column1'] > threshold1) & (df['column2'] < threshold2)]

# ----------------------5. Handling Categorical Data----------------------
# One-hot encoding categorical variables
df_encoded = pd.get_dummies(df, columns=['categorical_column'])

# Label encoding
label_encoder = LabelEncoder()
df['categorical_column'] = label_encoder.fit_transform(df['categorical_column'])

# ----------------------6. Feature Engineering----------------------
# Creating a new feature
df['new_feature'] = df['feature1'] * df['feature2']

# Binning numerical data
df['binned_feature'] = pd.cut(df['numerical_feature'], bins=3, labels=['low', 'medium', 'high

# ----------------------7. Grouping Data----------------------
# Grouping data and calculating statistics
```

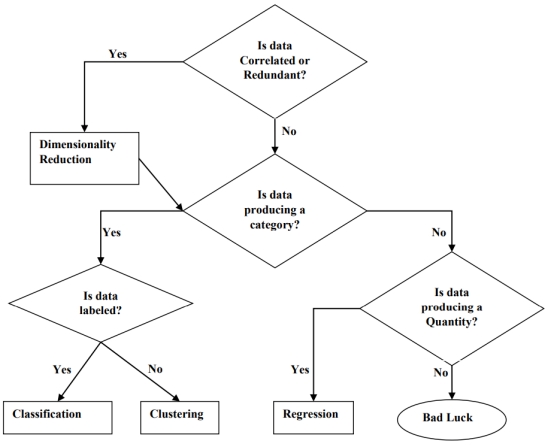
```
grouped_data = df.groupby('grouping_column')['numerical_column'].mean()

# ---------------------8. Merging DataFrames---------------------
# Concatenating DataFrames
df_concatenated = pd.concat([df1, df2], axis=0)

# Merging DataFrames
merged_df = pd.merge(df1, df2, on='common_column', how='inner')

# ---------------------9. Dropping Columns---------------------
# Dropping columns
df_dropped = df.drop(['column1', 'column2'], axis=1)
```

Section 2: Supervised Machine Learning (1 hour 30 minute)

📖 "Introduction to Machine Learning with Python" by Andreas C. Müller & Sarah Guido

GO to Example first then cross refer with note

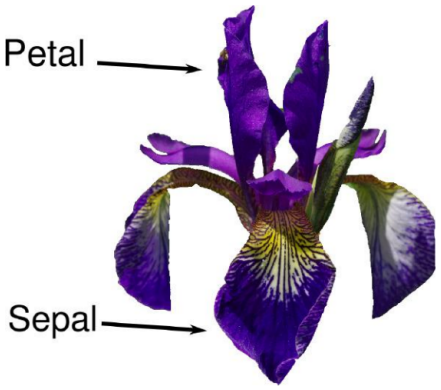




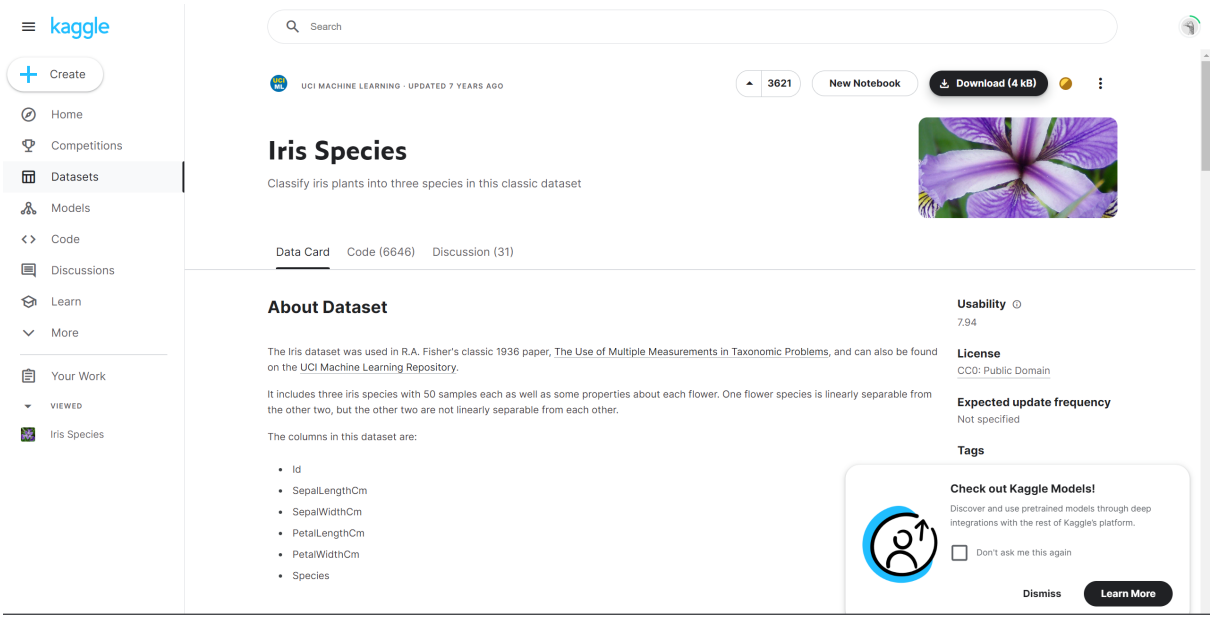
‘Hello world’ to ML: Iris (Classification ML)

Let’s assume that a hobby botanist is interested in distinguishing the species of some iris flowers that she has found.

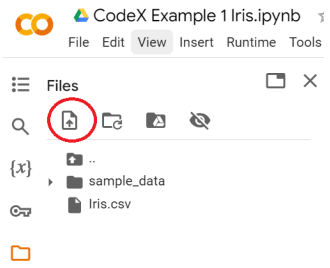
- She has collected some measurements associated with each iris: the length and width of the petals and the length and width of the sepals, all measured in centimeters.
- She also has the measurements of some irises that have been previously identified by an expert botanist as belonging to the species setosa, versicolor, or virginica.
- For these measurements, she can be certain of which species each iris belongs to. Let’s assume that these are the only species our hobby botanist will encounter in the wild.
- Our goal is to build a machine learning model that can learn from the measurements of these irises whose species is known, so that we can predict the species for a new iris.



Download the CSV from Kaggle : https://www.kaggle.com/datasets/uciml/iris/

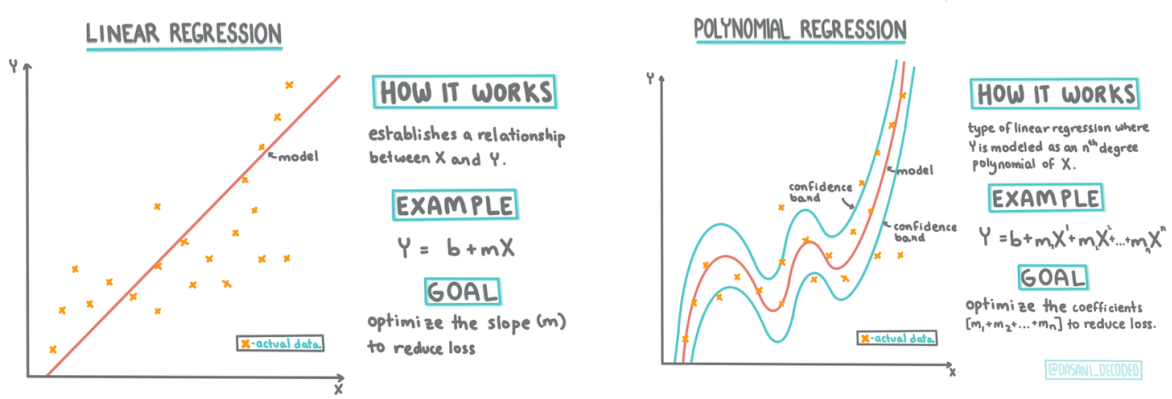


Load CSV file to Google Colab



VIEW THE PROCESS HERE: https://colab.research.google.com/drive/1lkiA-pLayzo8a-KoDAsQ8_kZiIqADxCZ?usp=sharing

Example 2: Linear Regression ML



> 💡 This type of ML is **far more complex than classification** due to **intricate statmath** foundation needed to be understand to obtain a good result. I recommend for further reading from online sources to learn more about different type of regression and the theory behind it to improve your result in the hackathon.

Download the data here:https://drive.google.com/file/d/1iu60hiAg6SxznOjVF4VP1_CYRWumLw2r/view?usp=sharing

View Implementation here:

https://colab.research.google.com/drive/1lXenJJnwB9spYXatMgjAMgRLVEdK5LuL?usp=sharing

Bias and Variance

Cheat Sheet – Bias-Variance Tradeoff

What is Bias? $bias = \mathbb{E}[f'(x)] - f(x)$

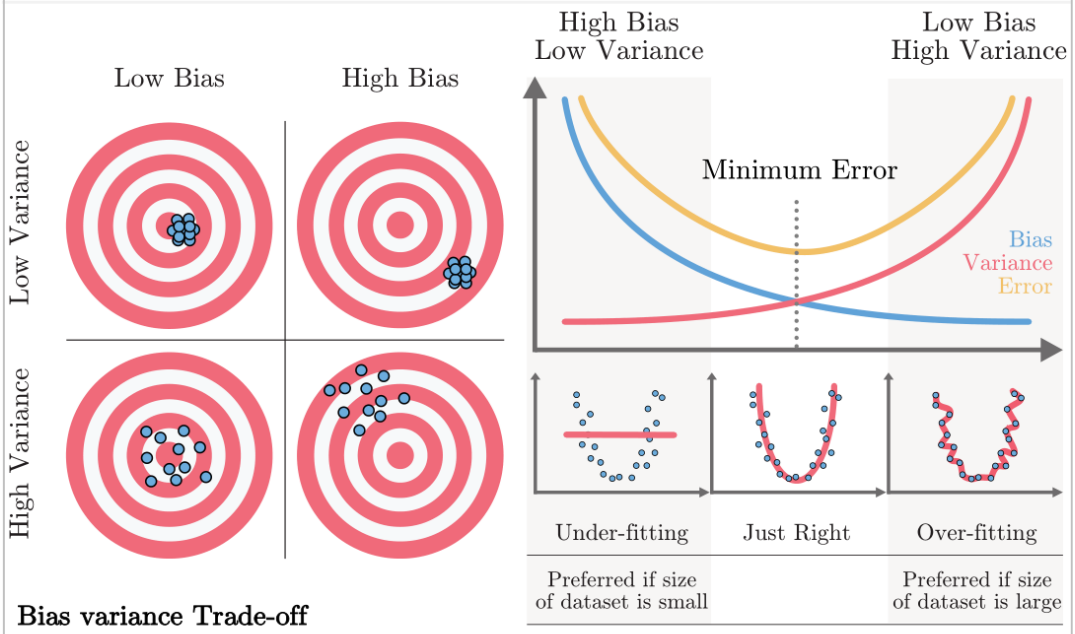
- Error between average model prediction and ground truth
- The bias of the estimated function tells us the capacity of the underlying model to predict the values

What is Variance? $variance = \mathbb{E}\left[\left(f'(x) - \mathbb{E}[f'(x)]\right)^2\right]$

- Average variability in the model prediction for the given dataset
- The variance of the estimated function tells you how much the function can adjust to the change in the dataset

High Bias
- → Overly-simplified Model
- → Under-fitting
- → High error on both test and train data

High Variance
- → Overly-complex Model
- → Over-fitting
- → Low error on train data and high on test
- → Starts modelling the noise in the input

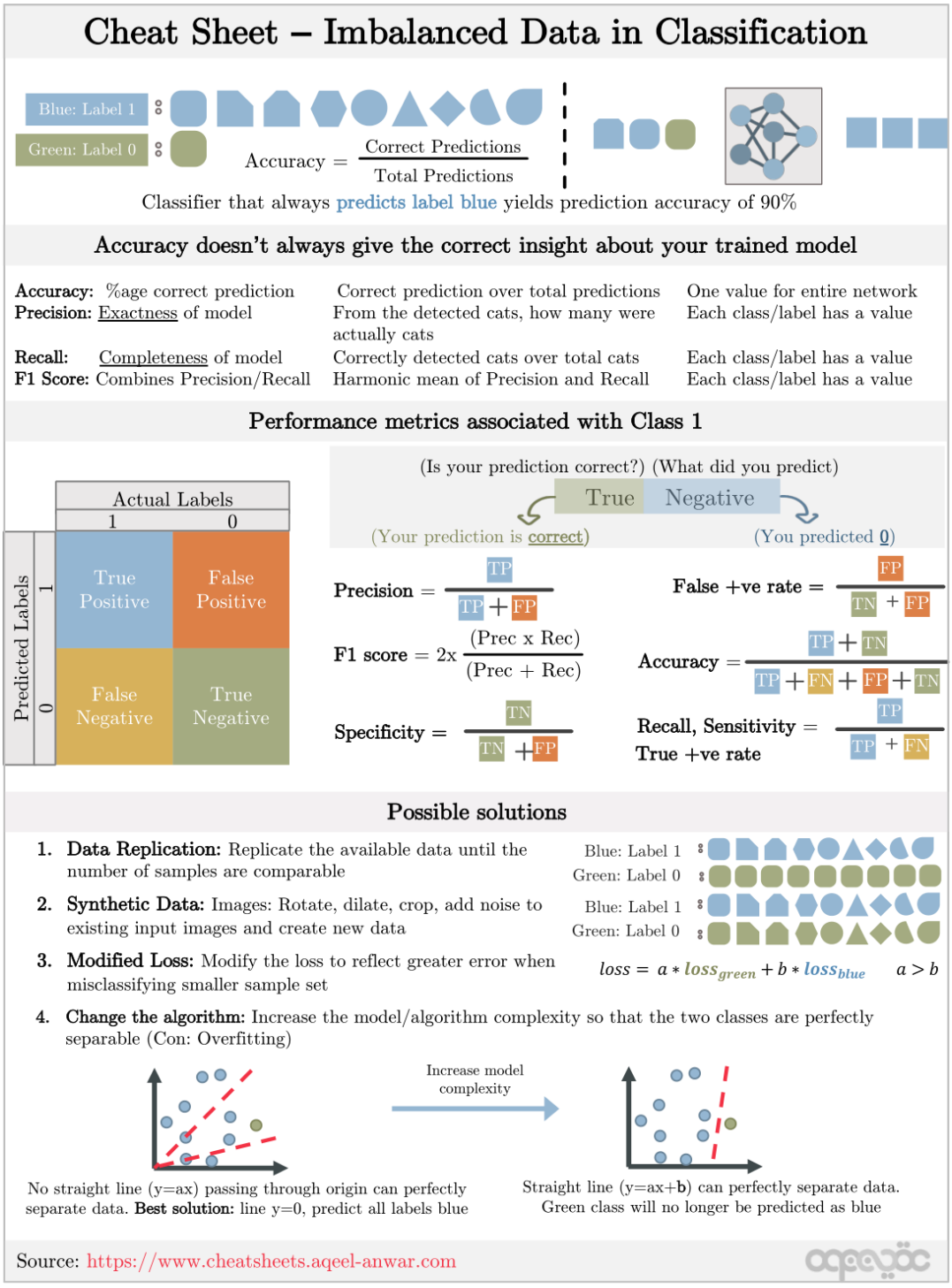


Bias variance Trade-off

- Increasing bias reduces variance and vice-versa
- Error = bias^2 + variance +irreducible error
- The best model is where the error is reduced.
- Compromise between bias and variance

Source: https://www.cheatsheets.aqeel-anwar.com

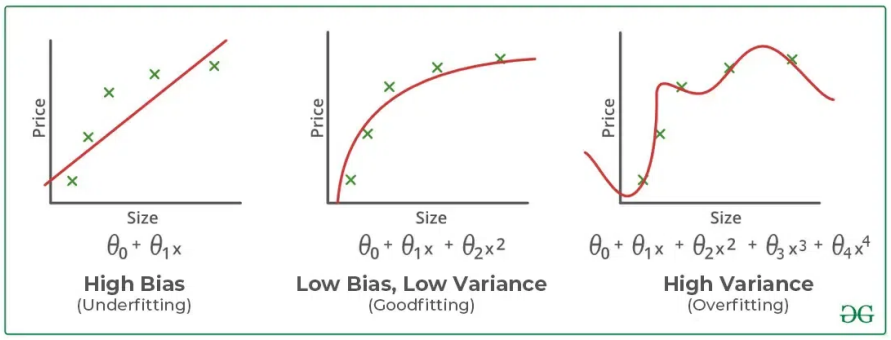
Imbalanced Data Classification



Overfitting vs Underfitting

🔥 **Underfitting** occurs when a model is too simple to capture the underlying patterns in the data. It fails to learn the training data and performs poorly on both the training and testing sets.

Overfitting occurs when a model is too complex and learns the training data too well, including its noise. As a result, the model performs well on the training set but poorly on unseen data.



Data Scaling

let's say you measure heights in centimeters and weights in kilograms. When you plot these points on your graph, you'll notice that the numbers on the height axis are much larger than the numbers on the weight axis. This difference in scale can make it challenging to see any patterns clearly.

Data scaling is like making a fair comparison. It's as if you decide to convert everyone's height and weight to a scale where the highest number is 1 and the lowest is 0. This way, you're not biased towards one measurement just because it naturally has larger values.

Method 1:**Standard Scaling (Z-score normalization):**

- It transforms the data to have a mean of 0 and a standard deviation of 1.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Method 2: Min-Max Scaling

- It scales the data to a specific range, usually [0, 1].

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

Handling Null values

Method 1: Just delete null-containing record

```
# Drop rows with null values
df.dropna(axis=0, inplace=True)
# Drop columns with null values
```

```
df.dropna(axis=1, inplace=True)
# df is panda dataframe
```

Method 2 : Imputation

Replace null values with a suitable estimate. Common methods include using the mean, median, or mode for numerical features or filling with the most frequent category for categorical features

```
# Impute null values with mean for numerical columns
df.fillna(df.mean(), inplace=True)
# Impute null values with mode for categorical columns
df['categorical_column'].fillna(df['categorical_column'].mode()[0], inplace=True)
```

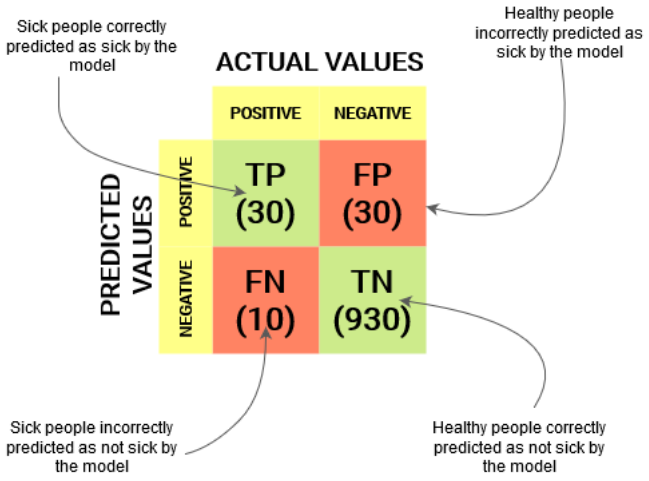
Classification ML Evaluation metrics

Accuracy

> 💡 It represents the percentage of correctly predicted instances out of the total instances. Accuracy is a straightforward measure and is often used for balanced datasets where the classes are approximately equally distributed. However, it might not be the best metric for imbalanced datasets, where one class significantly outnumbers the other

```
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_prediction)
print(f"Accuracy: {acc}")
```

Confusion Matrix (Classification ML)




```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate predictions
predictions = model.predict(test_X)

# Create a confusion matrix
conf_matrix = confusion_matrix(test_y, predictions)
```

```
# Display the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=iris_dataset['Species'].unique(),
            yticklabels=iris_dataset['Species'].unique())
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Explanation:

- **True Positive (TP):** The number of instances of a class correctly predicted as that class.
- **True Negative (TN):** The number of instances not of a class correctly predicted as not that class.
- **False Positive (FP):** The number of instances not of a class incorrectly predicted as that class.
- **False Negative (FN):** The number of instances of a class incorrectly predicted as not that class.

In the confusion matrix heatmap:

- The diagonal elements represent the correct predictions.
- Off-diagonal elements represent misclassifications.
- The rows represent the actual classes.
- The columns represent the predicted classes.

Classification Report (Classification ML)

```
from sklearn.metrics import classification_report

# Generate predictions
predictions = model.predict(test_X)

# Create a classification report
class_report = classification_report(test_y, predictions, target_names=iris_dataset['Species'

# Display the classification report
print('Classification Report:\n', class_report)
```

Precision:

Precision is a measure of how many correctly predicted instances of a class truly belong to that class. It is calculated as the ratio of true positives to the sum of true positives and false positives.

A high precision value indicates that when the model predicts a particular class, it is likely to be correct.

Recall (Sensitivity):

Recall, also known as sensitivity or true positive rate, measures the ability of the model to capture all instances of a particular class. It is calculated as the ratio of true positives to the sum of true positives and false negatives.

A high recall value indicates that the model is effective at identifying all instances of a class.

F1-Score:

The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall, especially in situations where one metric might be more important than the other.

The F1-score ranges between 0 and 1, where a higher value indicates a better balance between precision and recall.

Support:

Support represents the number of actual occurrences of each class in the specified dataset. It gives an idea of how many instances are available for each class

Regression Evaluation Metrics

✔ Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

- **Mean Absolute Error** (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Mean Squared Error** (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Root Mean Squared Error** (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

📌 Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_true, y_pred)
print(f"Mean Absolute Error (MAE): {mae}")

# Mean Squared Error (MSE)
mse = mean_squared_error(y_true, y_pred)
print(f"Mean Squared Error (MSE): {mse}")

# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

R^2 Analysis

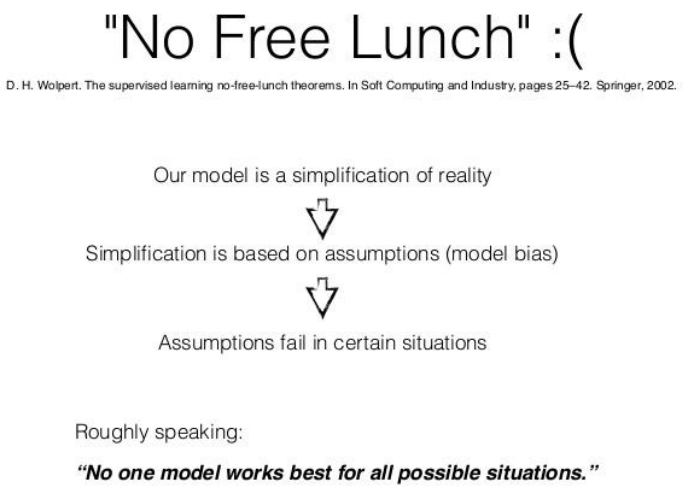
If $R^2 = 0$: the dependent variable Y can not be predicted from the independent variable **X**

If $R^2 = 1$: the dependent variable Y can be predicted from the independent variable **X**

If $0 < R^2 < 1$: Indicates the percentage at which the dependent variable Y is predictable by **X**

```
from sklearn.metrics import r2_score

r2=r2_score(y_test,y_pred)
```

Section 3:Hackathon Walkthrough(1 hour)



No Free lunch theorem

💡 When it comes to data modeling, the beginner's question is always, "what is the best machine learning algorithm?" To this the beginner must learn, the No Free Lunch Theorem (NFLT) of Machine Learning. In short, NFLT states, there is no super algorithm, that works best in all situations, for all datasets. So the best approach is to try multiple MLAs, tune them, and compare them for your specific scenario

Tweaking Parameter

💡 Tweaking parameters in machine learning is an essential part of the model development process. The process involves adjusting hyperparameters to improve the performance of a machine learning model. Hyperparameters are configuration settings external to the model itself that cannot be learned from training data but can significantly impact the model's performance

- Grid Search ('GridSearchCV' module from scikitlearn)
- Randomized search
- Manual Adjustment

- Validation Curves

> 📖 Brief hackathon, show brief example taken from kaggle(https://www.kaggle.com/code/nadintamer/titanic-survival-predictions-beginner) if we got time.

Hackathon Prompt: AI for Social Impact: Leveraging Supervised Machine Learning for Positive Change

Rubric: CodeX judge rubric (judges are needed to download to fill)

Description:
Embark on a journey to apply supervised machine learning techniques to address societal challenges and contribute to positive social impact. Participants are encouraged to choose a domain of interest such as education, health, environment, poverty alleviation, or any other area where machine learning can play a role in making a meaningful difference.

Key Tasks Needed to be included in presentation :

1. **Problem Definition:**
   - Clearly define a societal problem that can benefit from a supervised machine learning solution.
   - Articulate the potential impact on individuals or communities.
2. **Data Exploration and Collection:**
   - Explore existing datasets related to the chosen societal problem.
   - Discuss the data, exploratory analysis and data scrubbing are much appreciated.
3. **Model Development:**
   - Select and justify the choice of supervised machine learning algorithms.
   - Implement models to address the defined problem.
   - Explain the models: theoratically and practically based on your dataset.
4. **Ethical Considerations:**
   - Consider the ethical implications of the machine learning solution.
5. **Interpretability and Explainability:**
   - Strive for model interpretability and explainability to ensure transparency(white box)
   - Discuss how end-users can understand and trust the model's predictions.
6. **Scalability and Sustainability:**
   - Consider the scalability of the solution for broader impact.
   - Discuss how the solution can be sustained and adapted for long-term success.
7. **Presentation Format:**
   - **Introduction (10%):**
     - Introduce the chosen societal problem and its relevance.
     - Emphasize the potential positive impact of a machine learning solution.
   - **Data Exploration and Collection (25%):**
     - Share insights gained from data exploration(transformation process).
     - Discuss the availability and suitability of datasets.
   - **Model Development (20%):**
     - Present the choice of machine learning algorithms and their application.
     - Showcase the process of model development and optimization.

- **Ethical Considerations (15%):**
  - Discuss potential ethical challenges and considerations in the chosen domain.
  - Outline steps taken to address biases and ensure fairness.
- **Interpretability and Explainability (15%):**
  - Explain how the models are designed to be interpretable.
  - Discuss the importance of transparency in the context of social impact.
- **Scalability and Sustainability (10%):**
  - Discuss how the solution can be scaled for broader impact.
  - Outline plans for sustaining the solution over time.
- **Q&A and Discussion (5%):**
  - Engage with the audience through questions and discussions.
  - Encourage participants to share thoughts on extending the solution's impact.