

DATA SCIENCE – PART 2 | MIAS M1

Idriss JAIRI

Idriss.jairi@univ-lille.fr

COURSE PLAN

Session 1: Introduction to Data Science

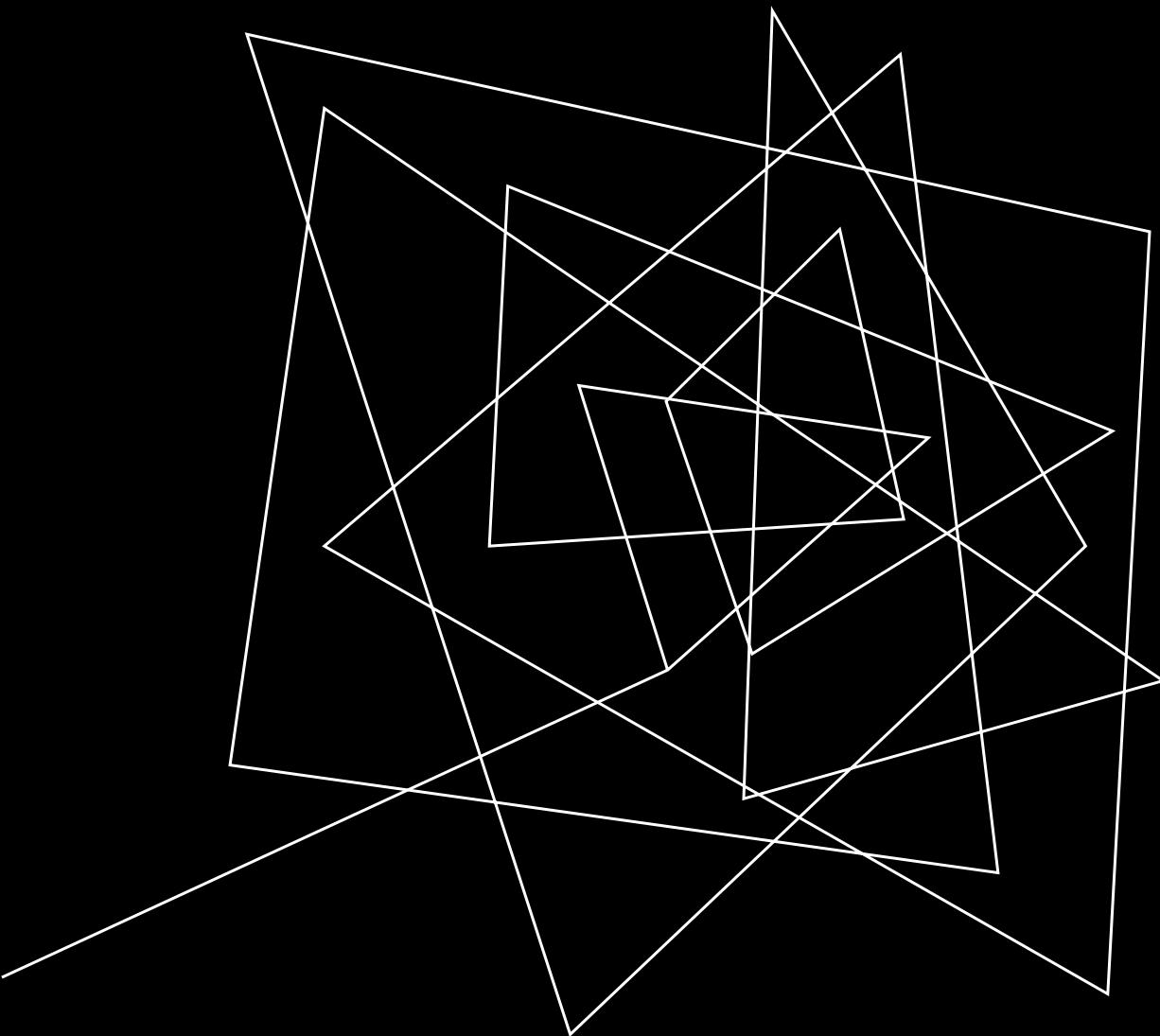
Session 2: Data Visualization and EDA

Session 3: Machine Learning Fundamentals

Session 4: Feature Engineering and Model Evaluation

Session 5: Introduction to Deep Learning

Session 6: Advanced Topics (Time series)



SESSION 1: INTRO TO DATA SCIENCE

Introduction to Data Science, Introduction to Python,
Numerical Computing with NumPy and Data
Manipulation with Pandas

Introduction to Data Science: What is Data Science?

Data science is the study of data to extract meaningful insights for business. It is a **multidisciplinary** approach that combines principles and practices from the fields of **mathematics, statistics, artificial intelligence, and computer engineering** to **analyze** large amounts of **data**. This analysis helps data scientists to ask and answer questions like **what happened, why it happened, what will happen, and what can be done with the results**. This **multidisciplinary** field has the primary goal of identifying **trends, patterns, connections** and **correlations** in large data sets.



Introduction to Data Science: What is Data Science?

Using tools from **coding (computer science)**, **statistics and math**, and **domain knowledge** to work **creatively** and **efficiently** with **DATA**.

GOAL: Getting insights about the **DATA**.

Math and Statistics



Computer Science and IT



Domains/Business Knowledge



Introduction to Data Science: Why Data Science is Important?



Fig. Types of data.
Source: wikipedia.org

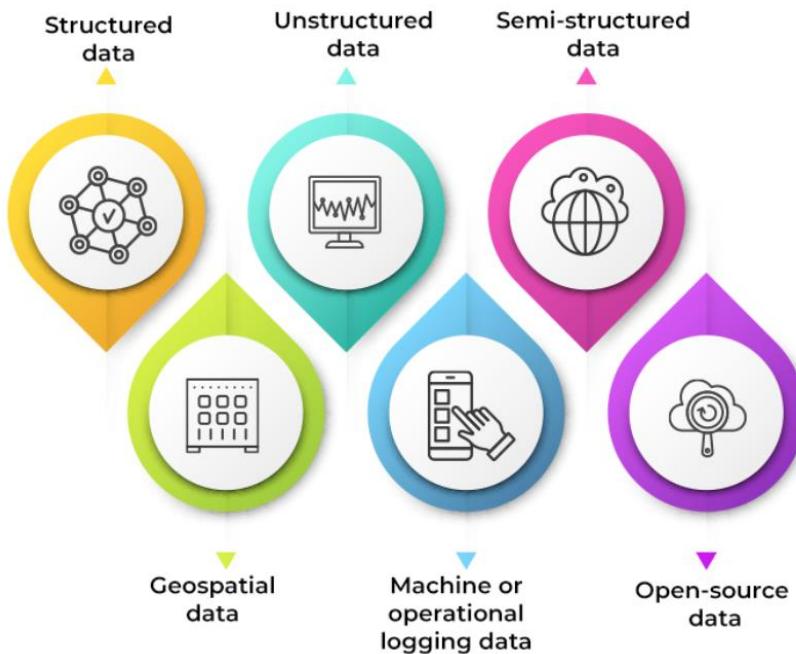


Fig. Types of big data.
Source: Spiceworks.com

Introduction to Data Science: Why Data Science is Important?

Data science is important because it combines tools, methods, and technology to generate meaning from data. Modern organizations are inundated with data; there is a proliferation of devices that can automatically collect and store information. Online systems and payment portals capture more data in the fields of e-commerce, medicine, finance, and every other aspect of human life. We have text, audio, video, and image data available in vast quantities.



Introduction to Data Science: Why Data Science is Important? <Some Key Reasons>

Informed Decision-Making

Data science enables organizations to make informed decisions by analyzing and interpreting large volumes of data. This leads to better strategies and improved decision outcomes.

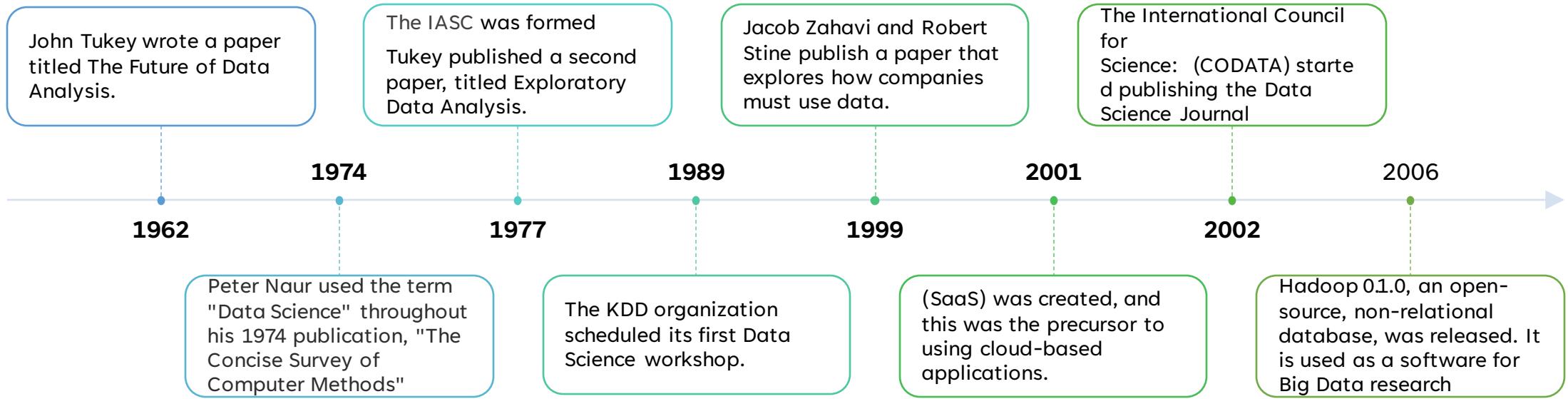
Business Intelligence

Data science helps businesses gain insights into customer behavior, market trends, and competitive landscapes. This intelligence is valuable for developing effective business strategies.

Predictive Analytics

Data science allows for the development of predictive models that can forecast future trends and outcomes. This is particularly beneficial in areas like sales forecasting, demand planning, and risk management.

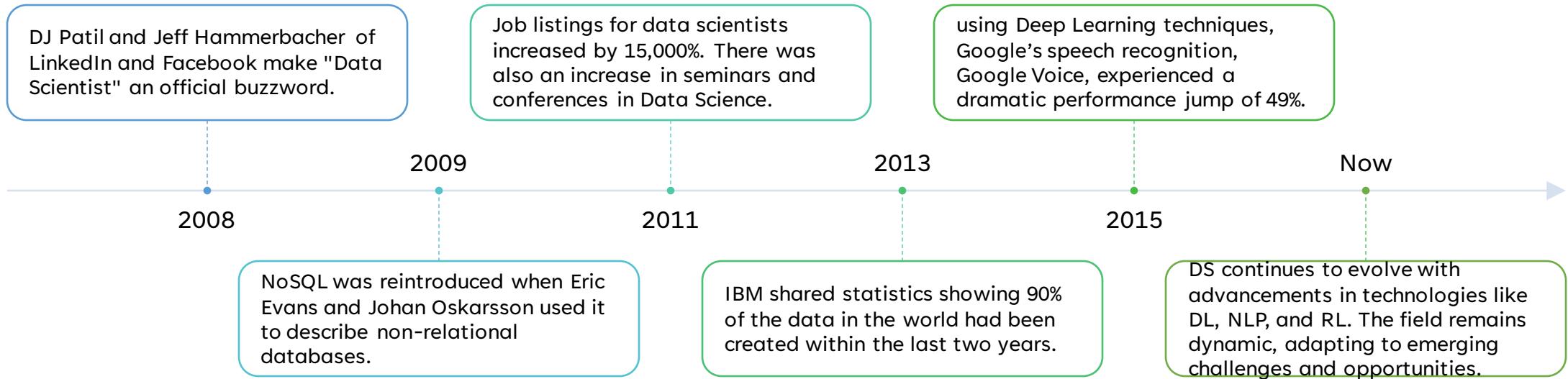
Introduction to Data Science: A Brief History



Source 1: [The History Of Data Science and Pioneers You Should Know](#)

Source 2: [A Brief History of Data Science](#)

Introduction to Data Science: A Brief History

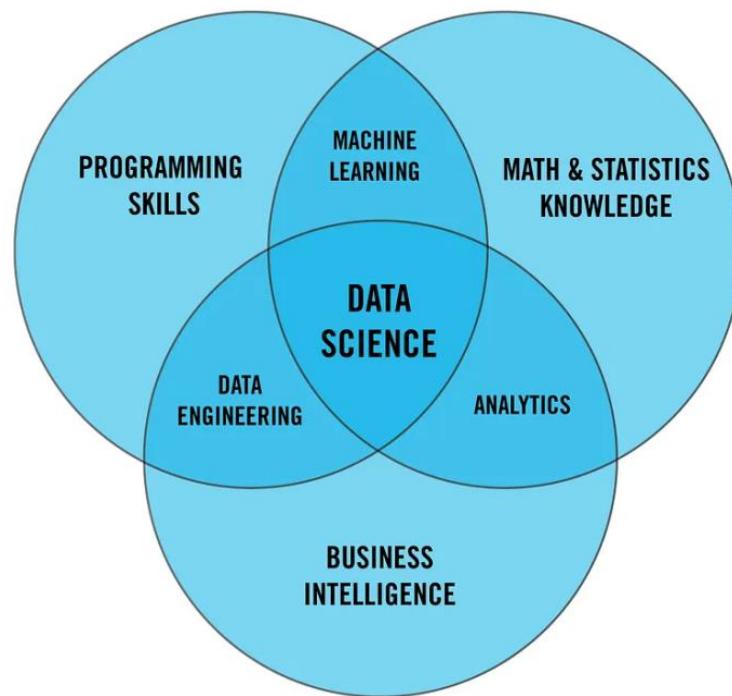


Source 1: [The History Of Data Science and Pioneers You Should Know](#)

Source 2: [A Brief History of Data Science](#)

Introduction to Data Science: Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

The roles of **data scientists**, **data analysts**, **data engineers**, and **machine learning engineers** are distinct, yet they share some commonalities.



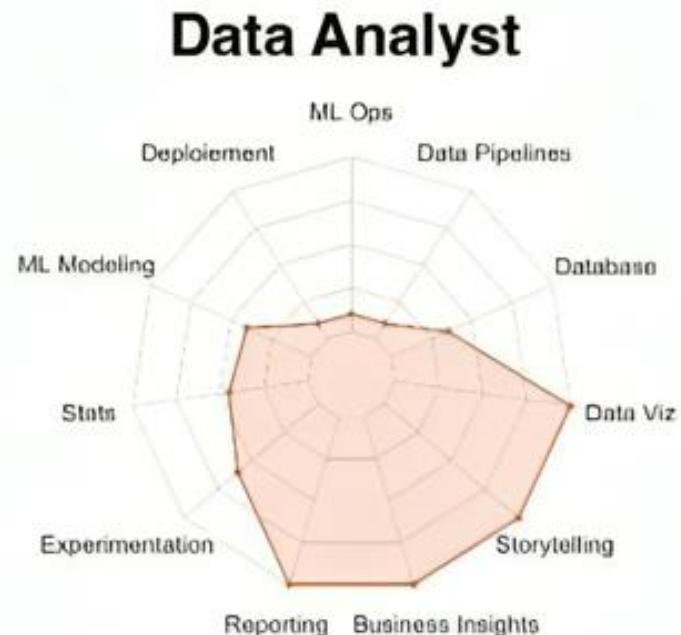
Introduction to Data Science: Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

- **Data Scientist:**
 - **Focus:** Extracting insights and knowledge from data.
 - **Responsibilities:**
 - Analyzing and interpreting complex data sets.
 - Developing and implementing machine learning models.
 - Communicating findings to non-technical stakeholders.
 - Identifying trends, patterns, and correlations in data.
 - **Skills:**
 - Strong statistical and mathematical background.
 - Proficiency in programming languages (e.g., Python, R).
 - Data visualization and storytelling skills.
 - Machine learning expertise.



Introduction to Data Science: Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

- **Data Analyst:**
 - **Focus:** Examining and interpreting data to provide insights.
 - **Responsibilities:**
 - Cleaning and processing data.
 - Creating reports and dashboards.
 - Identifying trends and patterns.
 - Conducting statistical analysis.
 - **Skills:**
 - Data cleaning and analysis using tools like SQL, Excel, or Python.
 - Data visualization skills.
 - Basic statistical knowledge.
 - Strong attention to detail.



Introduction to Data Science: Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

- **Data Engineer:**
 - **Focus:** Designing, constructing, and maintaining the systems and architecture for data generation.
 - **Responsibilities:**
 - Building and optimizing data pipelines.
 - Developing, constructing, testing, and maintaining databases.
 - Collaborating with data scientists to implement models into production.
 - **Skills:**
 - Proficiency in programming languages (e.g., Python, Java, Scala).
 - Knowledge of big data technologies (e.g., Hadoop, Spark).
 - Database design and management skills.
 - ETL (Extract, Transform, Load) processes.



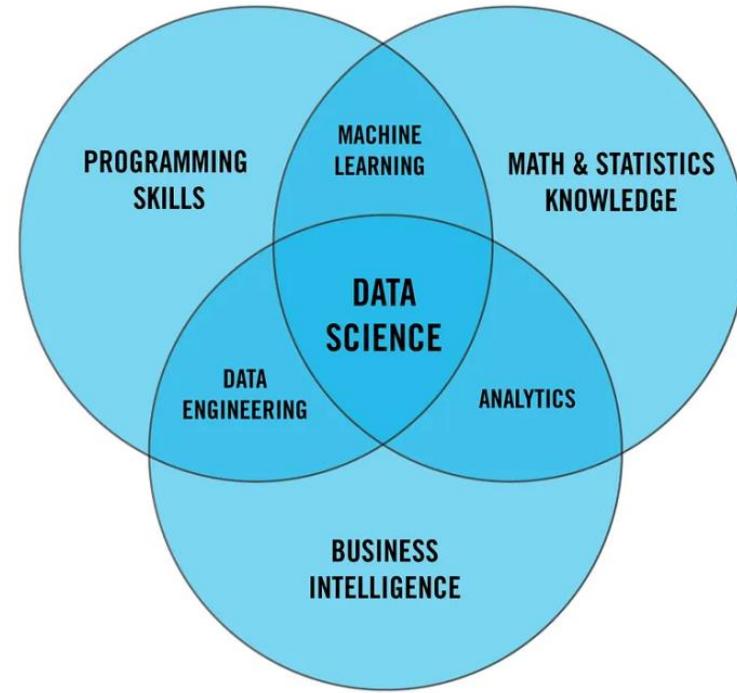
Introduction to Data Science: Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

- **Machine Learning Engineer:**
 - **Focus:** Designing and implementing machine learning models into production systems.
 - **Responsibilities:**
 - Developing and deploying machine learning models.
 - Collaborating with data scientists and software engineers.
 - Ensuring scalability and performance of machine learning solutions.
 - **Skills:**
 - Strong programming skills (e.g., Python, Java).
 - In-depth knowledge of machine learning algorithms and frameworks.
 - Experience with model deployment and productionization.
 - Understanding of software engineering principles.



Introduction to Data Science: Data Scientist vs. Data Analyst vs. Data Engineer vs. Machine Learning Engineer?

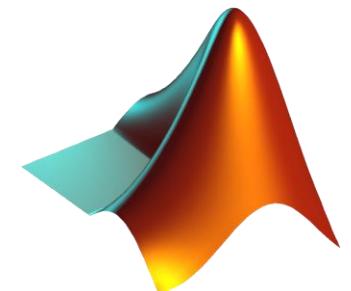
Note: While these roles have distinct responsibilities, in some organizations or smaller teams, individuals might wear multiple hats, and the lines between these roles can blur. Moreover, the field is dynamic, and the specific skills and responsibilities can vary based on the industry, company size, and evolving technologies.



Introduction to Data Science: What Programming Languages Are Used in Data Science?

Data science involves a variety of programming languages, each serving different purposes in the data analysis workflow. The choice of programming language often depends on the specific task, preference, and the existing ecosystem.

The choice of programming language depends on factors such as the specific task, available libraries, ease of use, and the preferences of the data scientist or analyst. Many data scientists are proficient in multiple languages and choose the best tool for the job at hand.



Introduction to Data Science: What Programming Languages Are Used in Data Science?

Python: Python is widely used in the data science community due to its simplicity, readability, and a vast ecosystem of libraries and tools. Popular Python libraries for data science include NumPy, Pandas, Matplotlib, Seaborn, Scikit-learn, TensorFlow, and PyTorch.



R: R is a statistical programming language specifically designed for data analysis and visualization. It has a strong community in academia and is commonly used for statistical modeling, data exploration, and visualization.



SQL (Structured Query Language): SQL is essential for working with relational databases. It is used to retrieve, manipulate, and analyze data stored in databases. Knowledge of SQL is crucial for extracting and cleaning data from databases.



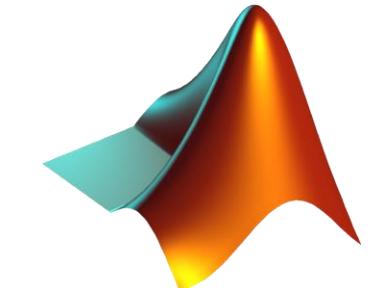
Introduction to Data Science: What Programming Languages Are Used in Data Science?

Julia: Julia is gaining popularity in the data science community for its performance and ease of use. It is designed to be fast and efficient for numerical and scientific computing, making it suitable for data-intensive tasks.

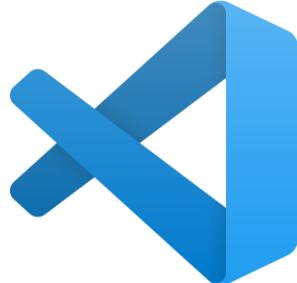
Scala: Scala, along with Apache Spark, is used for distributed data processing. Spark provides a fast and general-purpose cluster computing system, and Scala is the primary language for developing Spark applications.

Java: Java is commonly used in big data processing frameworks, such as Apache Hadoop. It's not as common in data science analysis but plays a significant role in handling large-scale data processing.

MATLAB: MATLAB is widely used in academia and industry for numerical computing and is popular for applications in engineering, physics, and finance.



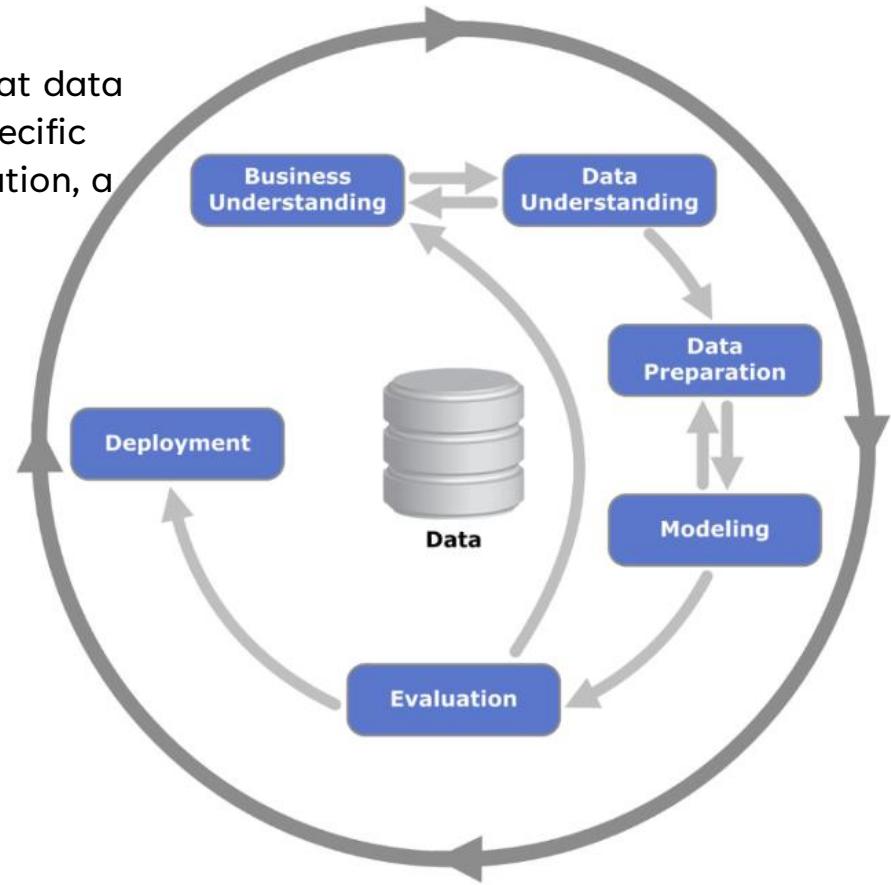
Introduction to Data Science: What Software Are Used in Data Science?



Introduction to Data Science: Data Science Workflow

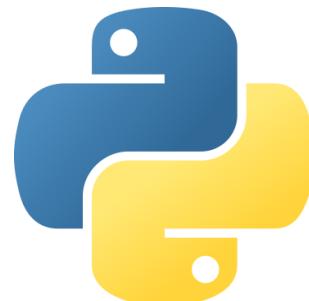
The data science workflow refers to the series of steps or stages that data scientists follow to analyze and derive insights from data. While specific workflows may vary depending on the project, domain, or organization, a typical data science workflow often includes the following stages:

1. Define the Problem
2. Collect Data
3. Data Cleaning and Preprocessing
4. Exploratory Data Analysis (EDA)
5. Feature Engineering
6. Model Selection
7. Model Training
8. Model Evaluation
9. Interpretability and Explainability
10. Deployment
11. Monitoring and Maintenance
12. Documentation



Introduction to Python: Overview

- **Brief History:**
 - Created by **Guido van Rossum** in the late 1980s
 - First released in 1991 (Python 1.0)
- **Key Features:**
 - Readability: Emphasizes code readability with a clean and easy-to-understand syntax.
 - Versatility: Supports both procedural and object-oriented programming paradigms.
 - Extensive Libraries: Offers a rich standard library and a vast ecosystem of third-party libraries.
 - Community Support: Active and supportive global community.
- **Applications:**
 - Web Development: Django, Flask.
 - Data Science: NumPy, Pandas, SciPy.
 - Artificial Intelligence: TensorFlow, PyTorch.
 - Automation: Scripting, task automation.



Guido van Rossum

Introduction to Python: Basic Syntax and Data Types

- Python uses indentation to define code blocks.
- Data Types:
 - Integers, Floats, Strings, Booleans.
 - Dynamic typing - no need to declare variable types explicitly.

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



Introduction to Python: Variables, Data Structures, and Control Flow

- **Variables:**
 - **Declaration:** variable_name = value
 - **Example:** x = 10
- **Data Structures:**
 - **Lists:** Ordered, mutable, can contain different data types. Created using square brackets "[]"
 - **Tuples:** Ordered, immutable collections of elements. Created using parentheses "()"
 - **Dictionaries:** Unordered key-value pairs for efficient data retrieval. Created using curly braces "{}"
 - **Sets:** Unordered collections of unique elements. Created using curly braces "{}" or "set()" constructor

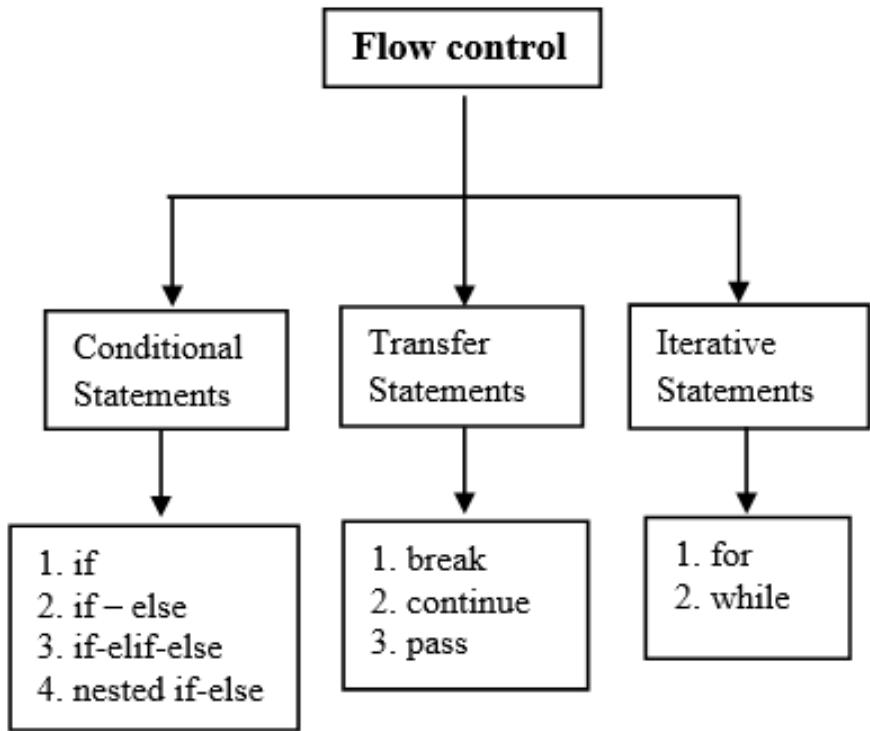
```
my_list = [1, 2, "three", 4.0]
```

```
my_tuple = (1, 2, "three", 4.0)
```

```
my_dict = {"name": "John", "age": 25}
```

```
my_set = {1, 2, 3, 4, 5}
```

Introduction to Python: Variables, Data Structures, and Control Flow



```
if condition:  
    # code block  
else:  
    # code block
```

```
for item in my_list:  
    # code block
```

```
while condition:  
    # code block
```

Introduction to Python: Learning Resources



Link: <https://www.python.org/>



Link: <https://www.w3schools.com/python/>

Introduction to Python: DEMO

DEMO: [Session 1 – Intro to Data Science](#)



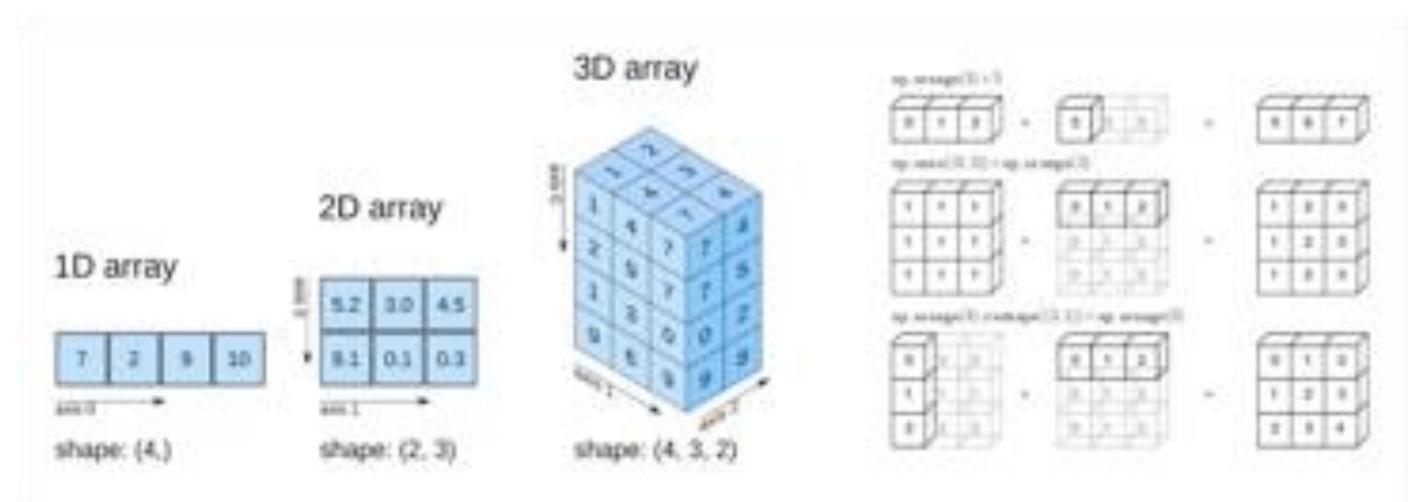
Numerical Computation with NumPy: <Definition>

- **Definition:** NumPy, short for Numerical Python, is a fundamental library for numerical computing in Python. Its core feature is the ndarray object, which allows efficient manipulation of large, multi-dimensional arrays and matrices. NumPy plays a pivotal role in scientific computing, data analysis, and machine learning, making it an indispensable tool for researchers, engineers, and data scientists.
- **Why choose NumPy for numerical computing?** NumPy excels in handling large datasets efficiently, providing optimized mathematical operations on arrays, and offering a broad range of mathematical functions. Its ability to seamlessly integrate with other libraries, such as SciPy for scientific computing and Matplotlib for data visualization, makes NumPy a cornerstone in the Python data science ecosystem."

Numerical Computation with NumPy: <Features>

N-dimensional Arrays

NumPy's main feature is the `numpy.ndarray` data structure, which is a multi-dimensional array. It allows you to represent and manipulate arrays of any dimensionality efficiently.



Numerical Computation with NumPy: <Features>

Broadcasting

NumPy supports broadcasting, a powerful mechanism that allows operations between arrays of different shapes and sizes. Broadcasting automatically expands smaller arrays to match the shape of larger ones, making element-wise operations more flexible.

$$\begin{array}{c} \text{(3,3)} \\ \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} \end{array} \otimes \begin{array}{c} \text{(3,) or (1,3)} \\ \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix} \end{array} = \begin{array}{c} \text{(3,3)} \\ \begin{matrix} -1 & 0 & 3 \\ -4 & 0 & 6 \\ -7 & 0 & 9 \end{matrix} \end{array}$$

multiplying several columns at once

$$\begin{array}{c} \text{(3,3)} \\ \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} \end{array} / \begin{array}{c} \text{(3,1)} \\ \begin{matrix} 3 & 3 & 3 \\ 6 & 6 & 6 \\ 9 & 9 & 9 \end{matrix} \end{array} = \begin{array}{c} \text{(3,3)} \\ \begin{matrix} .3 & .7 & 1. \\ .6 & .8 & 1. \\ .8 & .9 & 1. \end{matrix} \end{array}$$

row-wise normalization

$$\begin{array}{c} \text{(3,) or (1,3)} \\ \begin{matrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{matrix} \end{array} * \begin{array}{c} \text{(3,1)} \\ \begin{matrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{matrix} \end{array} = \begin{array}{c} \text{(3,3)} \\ \begin{matrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{matrix} \end{array}$$

outer product

Numerical Computation with NumPy: <Features>

Universal Functions (ufuncs)

- NumPy provides a set of universal functions that operate element-wise on arrays. These include mathematical, trigonometric, bitwise, and other operations. Ufuncs allow for efficient array computations without the need for explicit loops.
- **Why use ufuncs?** ufuncs are used to implement vectorization in NumPy which is way faster than iterating over elements. They also provide broadcasting and additional methods like reduce, accumulate etc. that are very helpful for computation.

```
import numpy as np

x = [1, 2, 3, 4]
y = [4, 5, 6, 7]

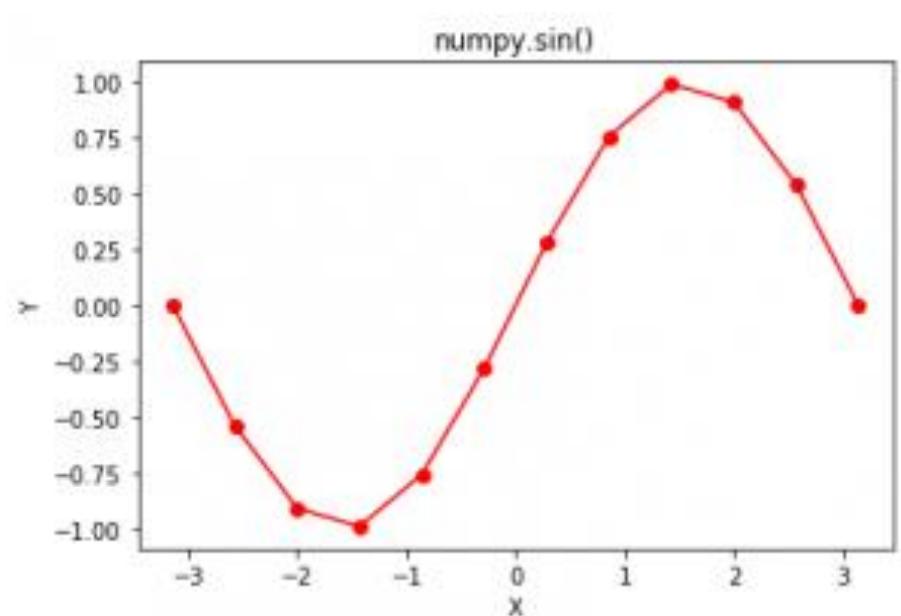
z = np.add(x, y)

print(z)
```

Numerical Computation with NumPy: <Features>

Mathematical Functions

NumPy includes a wide range of mathematical functions for linear algebra, Fourier analysis, random number generation, statistical analysis, and more. Examples include `np.sin()`, `np.cos()`, `np.exp()`, `np.mean()`, and many others.



Numerical Computation with NumPy: <Features>

Indexing Arrays

Array indexing is the same as accessing an array element.

- You can access an array element by referring to its index number.
- The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.
- **Note:** Use negative indexing to access an array from the end.

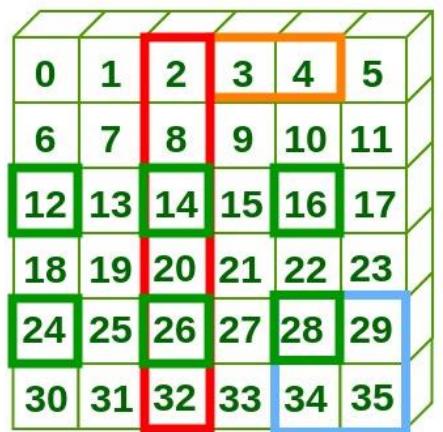
| | | | | |
|---------------------|----|----|----|----|
| 1 | 3 | 5 | 7 | 9 |
| index → 0 | 1 | 2 | 3 | 4 |
| negative index → -5 | -4 | -3 | -2 | -1 |

```
>>> a[0:3:5]
array( [3,4] )

>>> a[4:, 4:]
array( [ 28, 29],
      [ 34, 35] )

>>> a[ :, 2]
array( [2, 8, 14, 20, 26, 32] )

>>> a[2 :: 2, :: 2]
array( [ 12, 14, 16],
      [ 24, 26, 28] )
```



Numerical Computation with NumPy: <Features>

Indexing Arrays

```
import numpy as np

arr_1d = np.array([1, 2, 3, 4, 5])

# Accessing a single element
print(arr_1d[2]) # Output: 3

# Negative indexing
print(arr_1d[-1]) # Output: 5
```

1D Array

```
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Accessing a single element
print(arr_2d[1, 2]) # Output: 6

# Accessing an entire row
print(arr_2d[1]) # Output: [4 5 6]

# Accessing an entire column
print(arr_2d[:, 1]) # Output: [2 5 8]
```

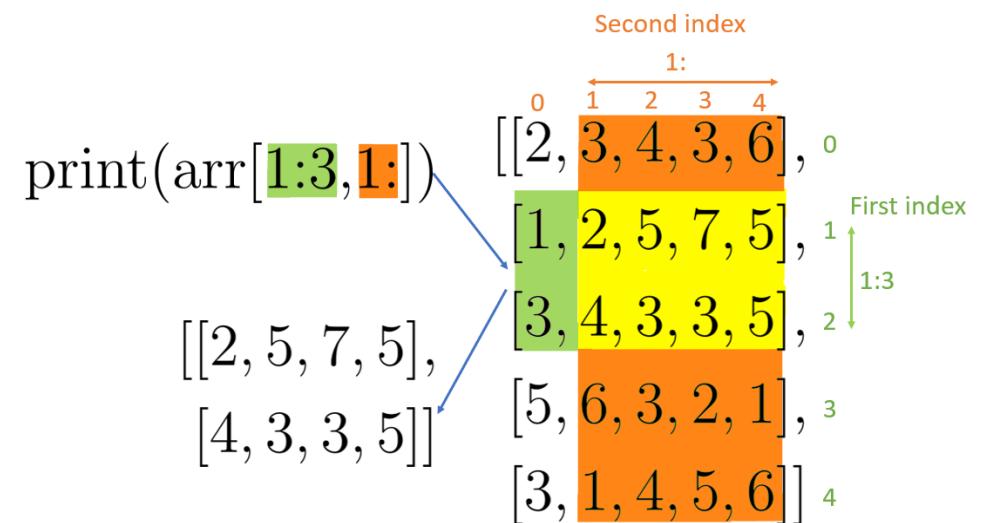
2D Array

Numerical Computation with NumPy: <Features>

Slicing Arrays

Slicing in python means taking elements from one given index to another given index.

- We pass slice instead of index like this: [start:end].
- We can also define the step, like this: [start:end:step].
- If we don't pass start it's considered 0
- If we don't pass end it's considered length of array in that dimension
- If we don't pass step it's considered 1
- **Note:** The result includes the start index, but excludes the end index.



Numerical Computation with NumPy: <Features>

Slicing Arrays

```
arr_1d = np.array([1, 2, 3, 4, 5])

# Slicing a subarray
subarray = arr_1d[1:4] # Output: [2 3 4]

# Slicing with a step
subarray_step = arr_1d[::-2] # Output: [1 3 5]
```

1D Arrays

```
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Slicing a subarray
subarray = arr_2d[0:2, 1:3]
# Output:
# [[2 3]
#  [5 6]]

# Slicing with a step
subarray_step = arr_2d[:, ::2]
# Output:
# [[1 3]
#  [4 6]
#  [7 9]]
```

2D Arrays

Numerical Computation with NumPy: <Features>

Linear Algebra Operations:

NumPy provides a comprehensive set of linear algebra operations, including matrix multiplication (np.dot()), determinant calculation (np.linalg.det()), eigenvalue computation (np.linalg.eig()), and more.

```
import numpy as np

# Define two matrices
matrix1 = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])

matrix2 = np.array([[9, 8, 7],
                    [6, 5, 4],
                    [3, 2, 1]])

# Perform matrix multiplication
result_matrix = np.dot(matrix1, matrix2)
```

Numerical Computation with NumPy: <NumPy Installation and Importing>

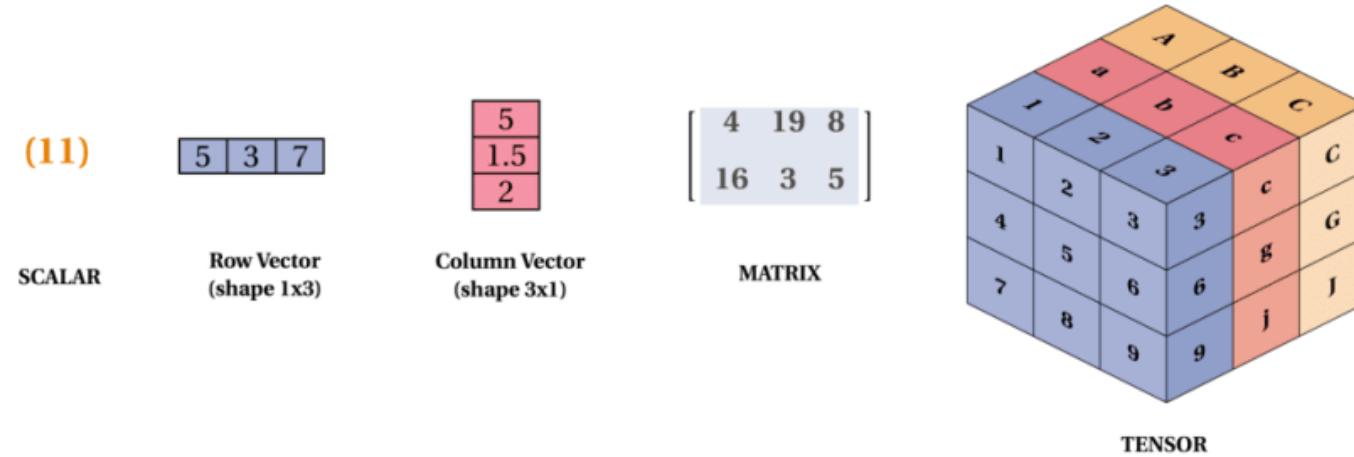
■ Anaconda Prompt (anaconda3)

```
(base) C:\Users\jairiidriss>pip install numpy
```

In [50]: `import numpy as np`

Numerical Computation with NumPy: <Review of Linear Algebra>

- Vectors and vector operations
- Matrix and matrix operations



Source: <https://web.stanford.edu/class/cs246/>

Numerical Computation with NumPy: <Review of Linear Algebra>

Why Linear Algebra?



Paolo Perrone • Following

I help Silicon Valley tech-founders build their audience on LinkedIn
2h • 

•••

Linear Algebra is to Machine Learning as Flour is to Bakery

Not the only ingredient, but no model bakes up without that essential base

Here's an intro to linear algebra for applied ML with Python

Through intuitive lessons and Python code you'll quickly grasp key concepts like matrix math, eigenvectors and independence.

No need to slog through dry-dense textbooks!

This guide is your fast-track to understanding the true foundation of ML

100% free  <https://lnkd.in/g3jn2s3V>

Link: [Introduction to Linear Algebra for Applied Machine Learning with Python](https://lnkd.in/g3jn2s3V)

Numerical Computation with NumPy: <Review of Linear Algebra>

Vectors and vector operations: Column vs. Row vector

A vector is a mathematical object that has both magnitude and direction. It is often represented as an ordered list of numbers or coordinates.

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} (n * 1)$$

Column vector

$$V = [v_1 \ v_2 \ \dots \ v_n] : (1 * n)$$

Row vector

Numerical Computation with NumPy: <Review of Linear Algebra>

**Vectors and vector operations: Vector
Multiplication (dot product)**

$$u \cdot v = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = u_1 \cdot v_1 + u_2 \cdot v_2 + \dots + u_n \cdot v_n = \sum_{i=1}^n u_i \cdot v_i$$

Numerical Computation with NumPy: <Review of Linear Algebra>

Vectors and vector operations: Norm of a vector

The norm of a vector is a mathematical concept that represents the size or length of the vector. There are different ways to calculate the norm, and the choice of norm depends on the context and application. The two most common norms are the Euclidean norm (or 2-norm) and the Manhattan norm (or 1-norm).

- Euclidean Norm (L2 Norm): The Euclidean norm of a vector v in n -dimensional space is denoted as $\|v\|$ or $\|v\|_2$
- Manhattan Norm (L1 Norm): The Manhattan norm of a vector v is denoted as $\|v\|_1$, and it is calculated as the sum of the absolute values of its components

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \dots + v_n^2}$$

$$\|v\|_1 = |v_1| + |v_2| + \dots + |v_n|$$

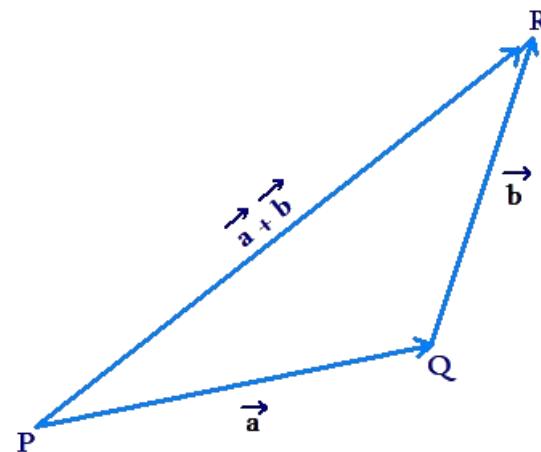
Numerical Computation with NumPy: <Review of Linear Algebra>

Vectors and vector operations: Triangle Inequality

For two vectors u and v , we have:

$$\|u + v\| \leq \|u\| + \|v\|$$

$$\|u - v\| \geq \left| \|u\| - \|v\| \right|$$



Numerical Computation with NumPy: <Review of Linear Algebra>

Matrix and Matrix Operations

- A matrix is a rectangular array of numbers, symbols, or expressions, arranged in rows and columns. The size of a matrix is given by its number of rows and columns.
- Matrices are denoted by uppercase letters. For example, an $m \times n$ matrix A could be represented as:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Numerical Computation with NumPy: <Review of Linear Algebra>

Matrix Operations: Addition

Matrix addition is defined for matrices of the same dimension. Matrices are added component-wise:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

Numerical Computation with NumPy: <Review of Linear Algebra>

Matrix Operations: Multiplication

Matrices can be multiplied like so:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \times 5 + 2 \times 7 & 1 \times 6 + 2 \times 8 \\ 3 \times 5 + 4 \times 7 & 3 \times 6 + 4 \times 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

You can also multiply non-square matrices, but the dimensions have to match(i.e. the number of columns of the first matrix has to equal the number of rows of the second matrix).

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 4 & 1 \cdot 2 + 2 \cdot 5 & 1 \cdot 3 + 2 \cdot 6 \\ 3 \cdot 1 + 4 \cdot 4 & 3 \cdot 2 + 4 \cdot 5 & 3 \cdot 3 + 4 \cdot 6 \\ 5 \cdot 1 + 6 \cdot 4 & 5 \cdot 2 + 6 \cdot 5 & 5 \cdot 3 + 6 \cdot 6 \end{bmatrix} = \begin{bmatrix} 9 & 12 & 15 \\ 19 & 26 & 33 \\ 29 & 40 & 51 \end{bmatrix}$$

In general, If matrix A is multiplied by matrix B, we have $(AB)_{ij} = \sum_k A_{i,k} * B_{k,j}$ for all entries (i,j)of the matrix product.

$$(AB)_{ij} = \sum_k A_{ik}B_{kj}$$

Numerical Computation with NumPy: <Review of Linear Algebra>

Matrix Operations: Multiplication

Note: Matrix multiplication is **associative**, i.e. $(AB)C=A(BC)$. It is also **distributive**, i.e. $A(B+C)=AB+AC$. However, it is **not commutative**. That is, AB does not have to equal BA . Note that if you multiply a 1-by- n matrix with an n -by-1 matrix, that is the same as taking the dot product of the corresponding vectors.

Numerical Computation with NumPy: <Review of Linear Algebra>

Matrix Operations: Matrix Transpose

The transpose operation switches a matrix's rows with its columns, so:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

In other words, we define A^T by $(A^T)_{ij} = A_{ji}$.

Properties:

- $(A^T)^T = A$
- $(AB)^T = B^T A^T$
- $(A+B)^T = A^T + B^T$

Numerical Computation with NumPy: <Review of Linear Algebra>

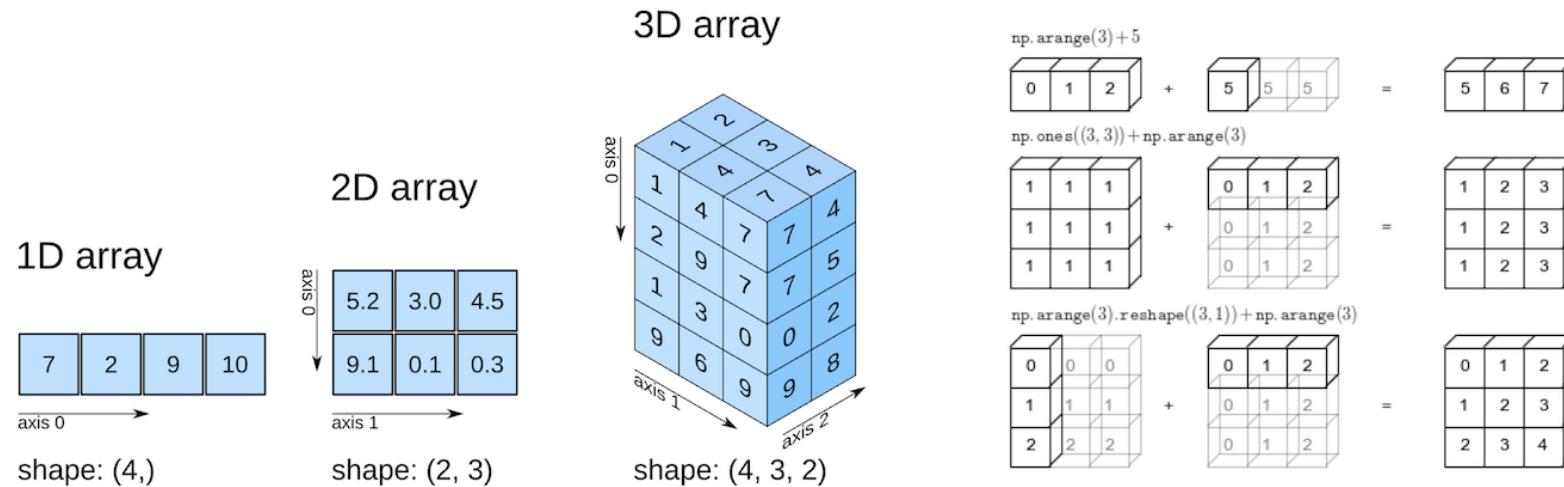
Matrix Operations: Identity Matrix

The identity matrix I_n is an n -by- n matrix with all 1's on the diagonal, and 0's everywhere else. It is usually abbreviated I , when it is clear what the dimensions of the matrix are. It has the property that when you multiply it by any other matrix, you get that matrix. In other words, if A is an m -by- n matrix, then $A * I_n = I_m * A = A$.

$$I_1 = [1], I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \dots, I_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Numerical Computation with NumPy: <DEMO>

Demo: Session 1 – Introduction to Data Science



Data Manipulation With Pandas: Introduction

- **Definition 1:** Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.
- **Definition 2:** Pandas stands as a cornerstone in the realm of data manipulation within the Python ecosystem. It is a robust and versatile library designed to facilitate the handling and analysis of structured data seamlessly.
- **Programming languages:** Python, C, Cython
- **Developer:** Wes McKinney
- **Initial release:** 11 January 2008; 15 years ago



Data Manipulation With Pandas: Brief History and Purpose of Pandas

The story of **Pandas** begins with its creation by **Wes McKinney** in **2008** while working in the financial industry. Faced with the need for an efficient tool to manipulate and analyze financial data,

McKinney developed Pandas, which was later open-sourced in **2009**. Since then, it has evolved into one of the most essential libraries for data manipulation and analysis in Python.

The primary purpose of Pandas is to provide easy-to-use data structures and functions, enabling users to manipulate, analyze, and visualize structured data seamlessly. It caters to a diverse range of applications, from cleaning and preprocessing datasets to exploratory data analysis and statistical modeling.



Wes McKinney

Data Manipulation With Pandas: Key Features <DataFrame and Series>

At the core of Pandas lie two fundamental data structures: the **DataFrame** and the **Series**.

- The **DataFrame** is a two-dimensional, tabular data structure resembling a spreadsheet, where data is organized into rows and columns. It allows for efficient manipulation of structured data and is well-suited for tasks ranging from data cleaning to complex analytics.
- The **Series** is a one-dimensional array-like object that can hold any data type. It serves as the building block for constructing DataFrames and offers powerful indexing and data manipulation capabilities.

Understanding these key features is pivotal for unleashing the full potential of Pandas in your data manipulation endeavors.

| Series | | Series | | DataFrame | |
|---------------|--------|---------------|---------|------------------|--------|
| | apples | | oranges | | apples |
| 0 | 3 | 0 | 0 | 0 | 3 |
| 1 | 2 | 1 | 3 | 1 | 2 |
| 2 | 0 | 2 | 7 | 2 | 0 |
| 3 | 1 | 3 | 2 | 3 | 1 |

+

Data Manipulation With Pandas: Loading and Exploring Datasets

- Importing Pandas and Common Aliases
 - import pandas as pd
- Loading Data from Various Sources
 - Using "pd.read_*" function
 - pd.read_csv() for CSV files.
 - pd.read_excel() for Excel spreadsheets.
 - pd.read_sql() for SQL databases.

```
python
```

```
import pandas as pd
```

```
import pandas as pd
```

```
# Read a CSV file into a DataFrame  
df = pd.read_csv('your_file.csv')
```

```
# Display the DataFrame  
print(df)
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Exploration Methods>

Basic Exploration Methods: **head()**, **tail()**, **info()**, **describe()**.

Pandas offers a suite of quick and powerful methods to gain a snapshot of your dataset:

- **head()**: Displays the first few rows of your DataFrame, allowing you to glimpse at the data's initial structure.
- **tail()**: Provides a view of the last few rows, aiding in understanding the tail-end of your dataset.
- **info()**: Offers a concise summary of the DataFrame, including the data types, non-null counts, and memory usage.
- **describe()**: Generates descriptive statistics, giving a statistical overview of the dataset's numeric columns.

```
# Display the first 5 rows
print(df.head())
```

```
# Display the last 5 rows
print(df.tail())
```

```
# Display DataFrame information
print(df.info())
```

```
# Display summary statistics
print(df.describe())
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Accessing Columns and Rows>

```
# Access a specific column by name  
column_data = df['column_name']
```

- **Accessing Columns:**

- By Column Name
- Multiple Columns

- **Accessing Rows:**

- By Index (loc)
- By Integer Position (iloc)
- Slicing Rows

```
# Access multiple columns  
selected_columns = df[['column1', 'column2']]
```

```
# Access a specific row by index label  
row_data = df.loc[row_index]
```

```
# Access a specific row by integer position  
row_data = df.iloc[row_position]
```

```
# Access a range of rows using slicing  
sliced_data = df[2:5] # Rows 2 to 4
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Accessing Columns and Rows>

Accessing Both Rows and Columns

- By Index and Column Name (loc)
- By Integer Position and Column Position (iloc)

```
# Access a specific element by row index and column name
element_data = df.loc[row_index, 'column_name']
```

```
# Access a specific element by integer position and column position
element_data = df.iloc[row_position, column_position]
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Checking for Missing Values>

```
# Check for missing values in the entire DataFrame
print(df.isnull().sum())
```

```
# Check for missing values in the entire DataFrame
print(df.isna().sum())
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Cleaning>

Identifying and Handling Missing Data: **dropna()** and **fillna()**:

- '**dropna()- '**fillna()****

```
# Drop rows with any missing values
df_cleaned = df.dropna()
```

```
# Drop columns with any missing values
df_cleaned = df.dropna(axis=1)
```

```
# Fill missing values with a specific value (e.g., 0)
df_filled = df.fillna(0)
```

```
# Fill missing values with the mean of the column
df_filled = df.fillna(df.mean())
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Cleaning>

Removing Duplicates: **drop_duplicates()**

Duplicate entries can distort analyses and lead to misleading results. Pandas provides a simple solution to eliminate these redundancies.

- **drop_duplicates():** This method removes duplicate rows, retaining only the first occurrence.

```
# Remove duplicate rows based on all columns
df_no_duplicates = df.drop_duplicates()
```

```
# Remove duplicates based on specific columns
df_no_duplicates = df.drop_duplicates(subset=['column1', 'column2'])
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Cleaning>

Renaming Columns for Clarity

Column names play a vital role in understanding and communicating the content of a dataset. Pandas facilitates easy renaming of columns.

- `rename()`: Use this method to rename one or more columns, providing a clear and concise representation of the data.

```
# Rename a single column
df.rename(columns={'old_name': 'new_name'}, inplace=True)
```

```
# Rename multiple columns
df.rename(columns={'old_name1': 'new_name1', 'old_name2': 'new_name2'})
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Manipulation>

- Filtering Data with Boolean Indexing, Boolean indexing is a powerful method for selectively extracting data from a DataFrame based on conditions.
- Filter data with boolean indexing using 'and' (&) and 'or' (|) conditions.

```
# Filtering data based on a condition  
filtered_data = df[df['column_name'] > threshold_value]
```

```
# Filtering data with 'and' condition  
filtered_and_data = df[(df['Age'] > 25) & (df['Salary'] > 55000)]
```

```
# Filtering data with 'or' condition  
filtered_or_data = df[(df['Department'] == 'IT') | (df['Salary'] > 60000)]
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Manipulation>

Selecting Specific Columns and Rows using **loc** and **iloc**: Pandas offers two primary methods for precise selection of data: **loc** and **iloc**.

- **'loc'**: This method allows selection by label. We can specify both the rows and columns we want to extract.
- **'iloc'**: This method facilitates integer-location based indexing, providing a powerful tool for numeric selection.

```
# Select specific rows and columns by label
selected_data = df.loc[row_labels, ['column1', 'column2']]
```

```
# Select specific rows and columns by integer position
selected_data = df.iloc[row_positions, column_positions]
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Manipulation>

Adding and removing columns:

- Adding columns
- Removing columns

```
# Add a new column with a calculated value  
df['new_column'] = df['column1'] + df['column2']
```

```
# Remove a column  
df.drop(['column_to_remove'], axis=1, inplace=True)
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Manipulation>

Applying Functions to Columns using apply()

The `apply()` function in Pandas is a powerful tool for transforming data within columns. It allows the application of custom or predefined functions to each element or column of a DataFrame.

```
# Applying a function to a column
df['new_column'] = df['existing_column'].apply(lambda x: custom_function(x))
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Manipulation>

Grouping and Aggregating Data with groupby()

- Grouping and aggregation are essential steps in data analysis, enabling us to draw meaningful conclusions from our datasets.
- The groupby() function allows us to group data based on specific criteria and perform aggregate operations, such as calculating means, counts, or custom functions, on each group.

```
# Grouping data by a column and calculating aggregate statistics
grouped_data = df.groupby('grouping_column').agg({'numeric_column': 'mean', 'other_column': 'count'})
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Manipulation>

Merging and Concatenating DataFrames: Combining data from multiple sources is a common requirement in real-world data analysis. Pandas provides methods for both merging and concatenating DataFrames.

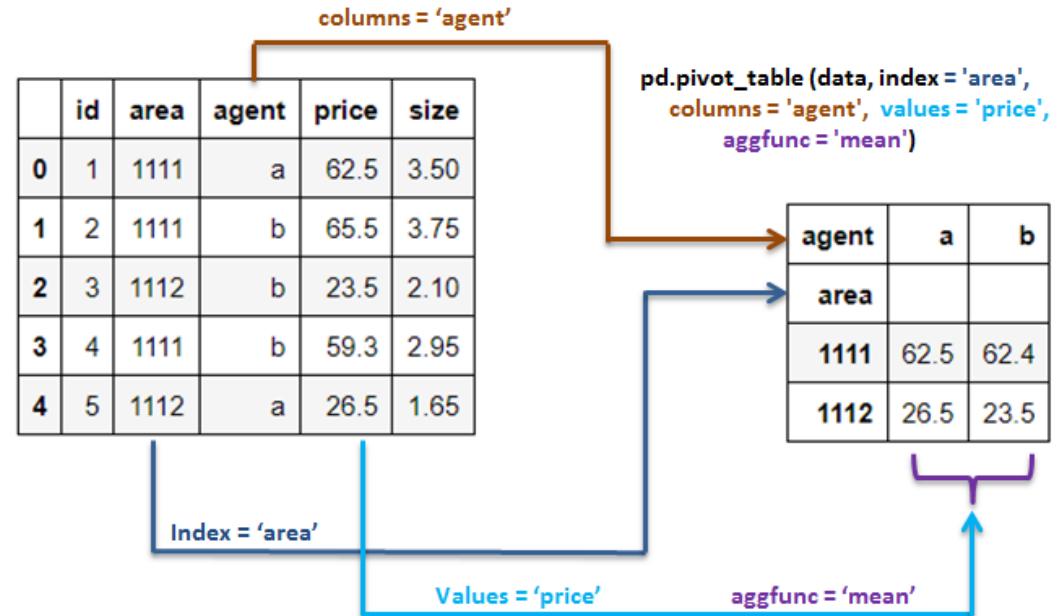
```
# Merge two DataFrames based on a common column
merged_data = pd.merge(df1, df2, on='common_column', how='inner')
```

```
# Concatenate DataFrames along a specific axis (rows or columns)
concatenated_data = pd.concat([df1, df2], axis=0)
```

Data Manipulation With Pandas: Loading and Exploring Datasets <Basic Data Manipulation>

Introduction to Pivot Tables

Pivot tables are a powerful tool for reshaping and summarizing data, providing a concise and structured view of information.



Data Manipulation With Pandas: Learning Resources



Link: <https://pandas.pydata.org/>



Link: <https://www.w3schools.com/python/pandas/>



Data Manipulation with Pandas: <DEMO>

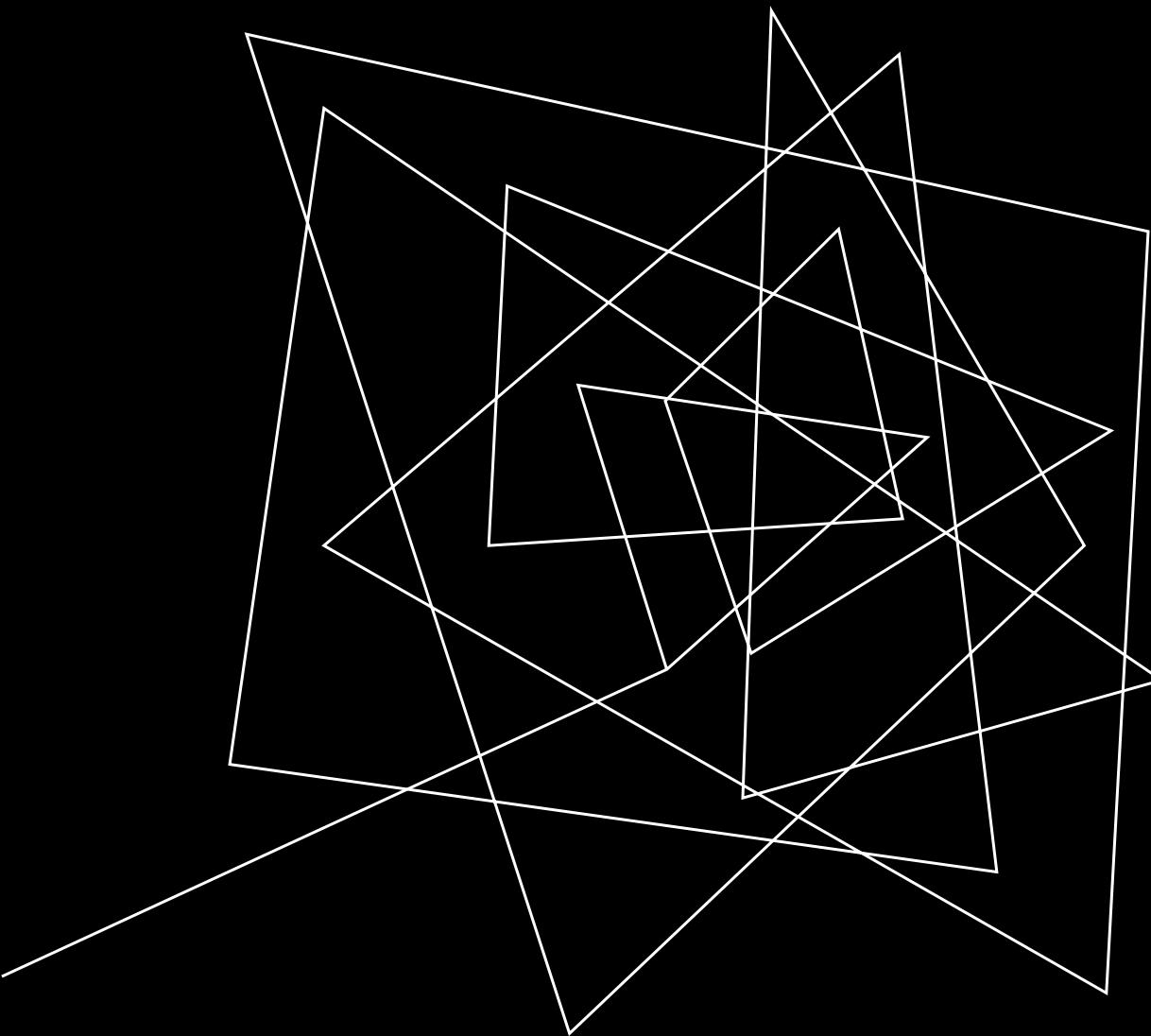
Demo: [Session 1 – Introduction to Data Science](#)



Course's Practical Example: Titanic Disaster

Demo: [Session 1 – Introduction to Data Science](#)



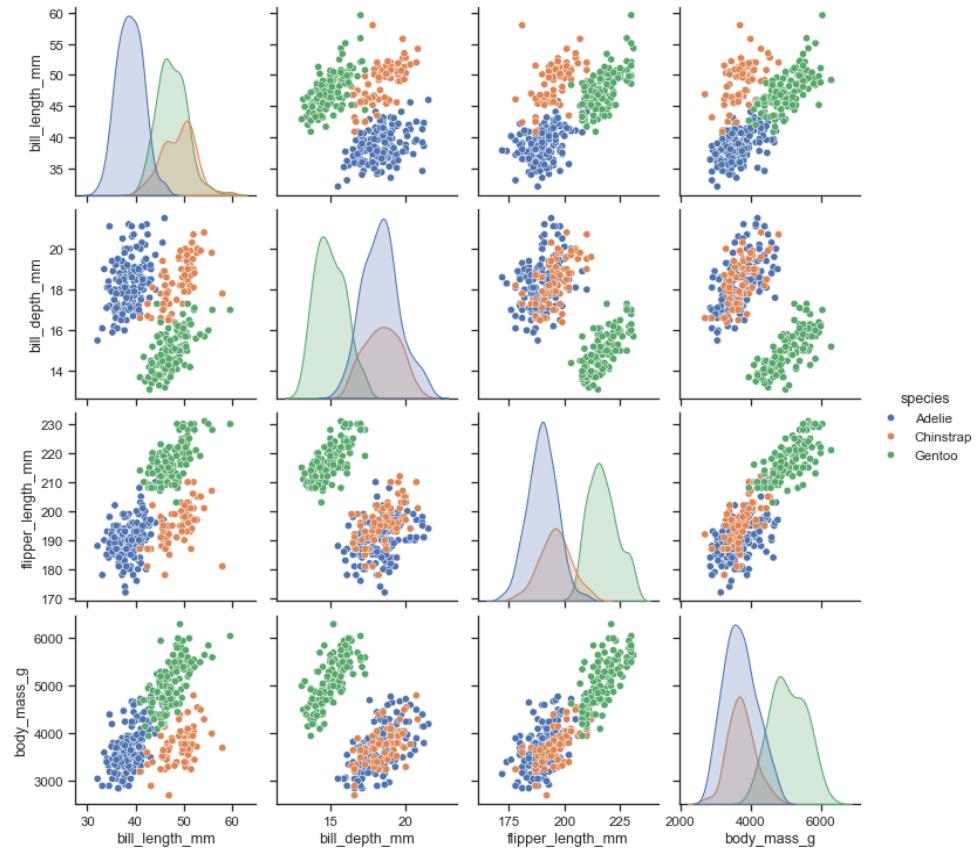


SESSION 2: DATA VISUALIZATION AND EDA

Understand the importance of data visualization and learn exploratory data analysis techniques.

Data Visualization and EDA

Data visualization and exploratory data analysis (EDA) play crucial roles in the field of data science, providing essential tools for understanding and extracting insights from complex datasets.



Data Visualization and EDA

Data Visualization:

Data visualization involves representing data graphically to gain insights. Visualization is a powerful tool for communication and exploration. Common types of visualizations include:

- **Scatter plots:** Show the relationship between two continuous variables.
- **Histograms:** Display the distribution of a single variable.
- **Box plots:** Summarize the distribution of a variable and identify outliers.
- **Line charts:** Show trends or patterns over time.
- **Heatmaps:** Visualize patterns in a matrix of data.
- **Bar charts:** Display the distribution of a categorical variable.
- **Pie charts:** Show the proportion of different categories in a whole.

Data Visualization and EDA:

<Importance of Data Visualization>

1. Understanding Patterns and Trends

Data visualization allows for the graphical representation of data, making it easier to identify patterns, trends, and relationships. Visualizations can reveal insights that may be challenging to discern from raw data alone.

2. Communication of Complex Information

Visualizations provide an effective means of communicating complex information to a diverse audience. Charts, graphs, and dashboards can convey insights quickly and clearly, enabling stakeholders to make informed decisions.

3. Identification of Outliers and Anomalies

Visualizations help in the identification of outliers or anomalies in the data. Anomalies may not be apparent in tabular data but can be easily spotted in visual representations, aiding in data cleaning and preprocessing.

Data Visualization and EDA: <Importance of Data Visualization>

4. Storytelling with Data

Visualization facilitates storytelling with data, allowing data scientists to create compelling narratives that highlight key findings and support data-driven conclusions. This is particularly valuable in conveying the significance of the results to non-technical stakeholders.

5. Comparisons and Benchmarking

Visualizations enable effective comparisons between different variables or datasets. Whether it's comparing performance over time or benchmarking against a standard, visual representations enhance the understanding of data relationships.

6. Enhanced Decision-Making

Decision-makers often rely on visualizations to grasp the implications of various choices. Visual data exploration supports evidence-based decision-making by providing a clear and intuitive understanding of the data.

Data Visualization: Matplotlib and Seaborn Libraries

- **Matplotlib** is a library in Python that enables users to generate visualizations like histograms, scatter plots, bar charts, pie charts and much more.
- **Seaborn** is a visualization library that is built on top of Matplotlib. It provides data visualizations that are typically more aesthetic and statistically sophisticated.
- **Matplotlib** and **Seaborn** are two popular Python libraries for data visualization, each serving different purposes while complementing each other.



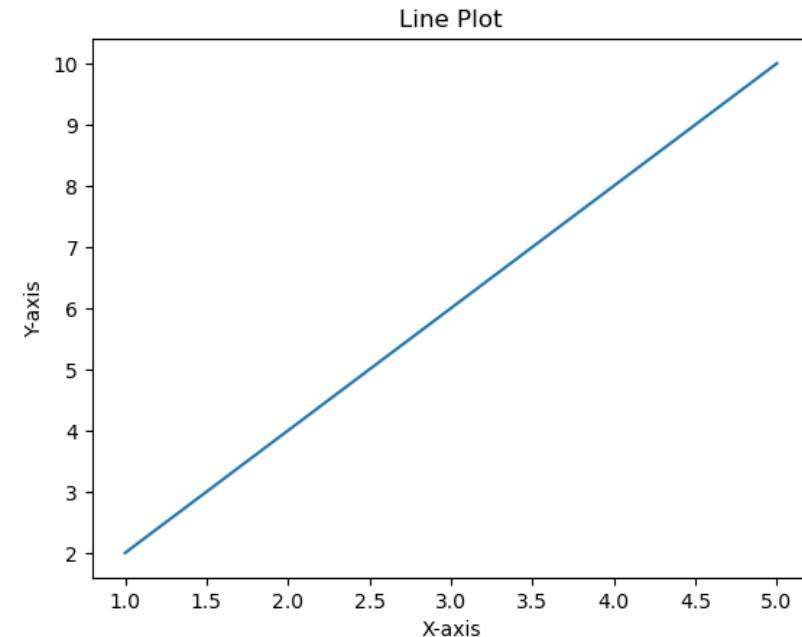
Data Visualization: Matplotlib Plots

<Line Plot>

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.plot(x, y)
plt.title('Line Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



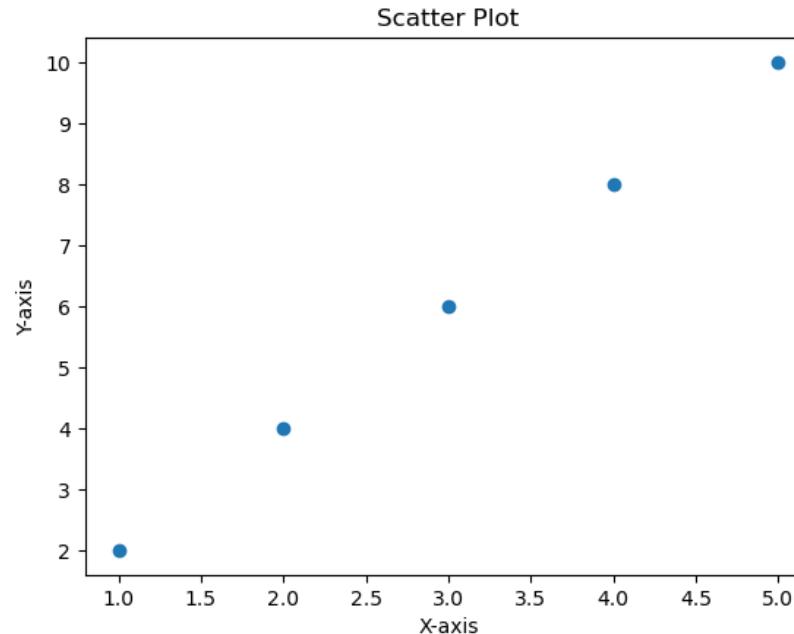
Data Visualization: Matplotlib Plots

<Scatter Plot>

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.scatter(x, y)
plt.title('Scatter Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



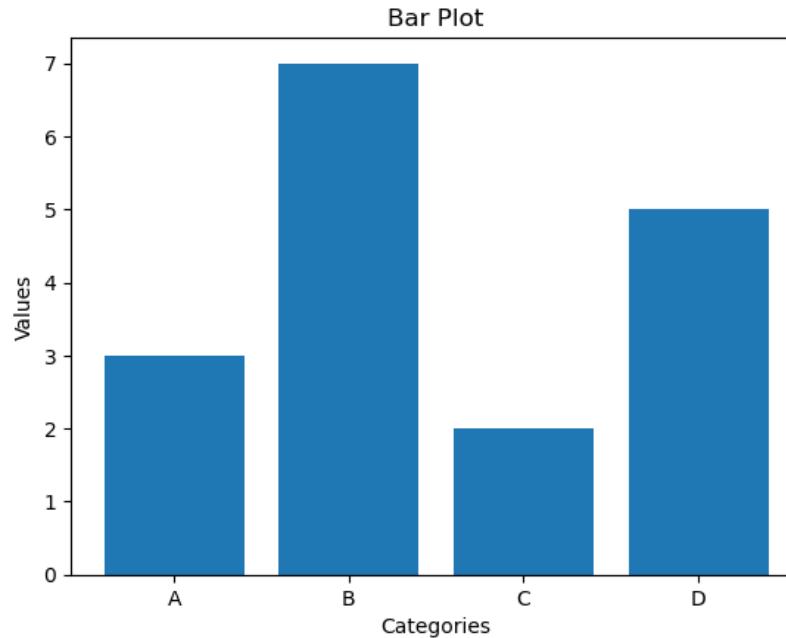
Data Visualization: Matplotlib Plots

<Bar Plot>

```
import matplotlib.pyplot as plt

categories = ['A', 'B', 'C', 'D']
values = [3, 7, 2, 5]

plt.bar(categories, values)
plt.title('Bar Plot')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.show()
```

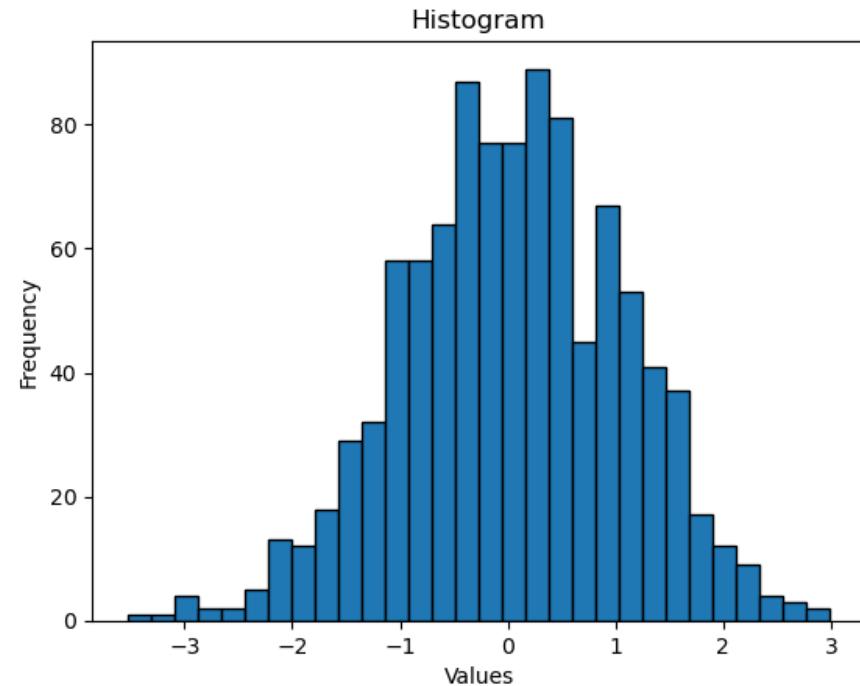


Data Visualization: Matplotlib Plots

<Histogram>

```
import matplotlib.pyplot as plt
import numpy as np

data = np.random.randn(1000) # Example data
plt.hist(data, bins=30, edgecolor='black')
plt.title('Histogram')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
```



Data Visualization: Matplotlib Plots

<Box Plot>

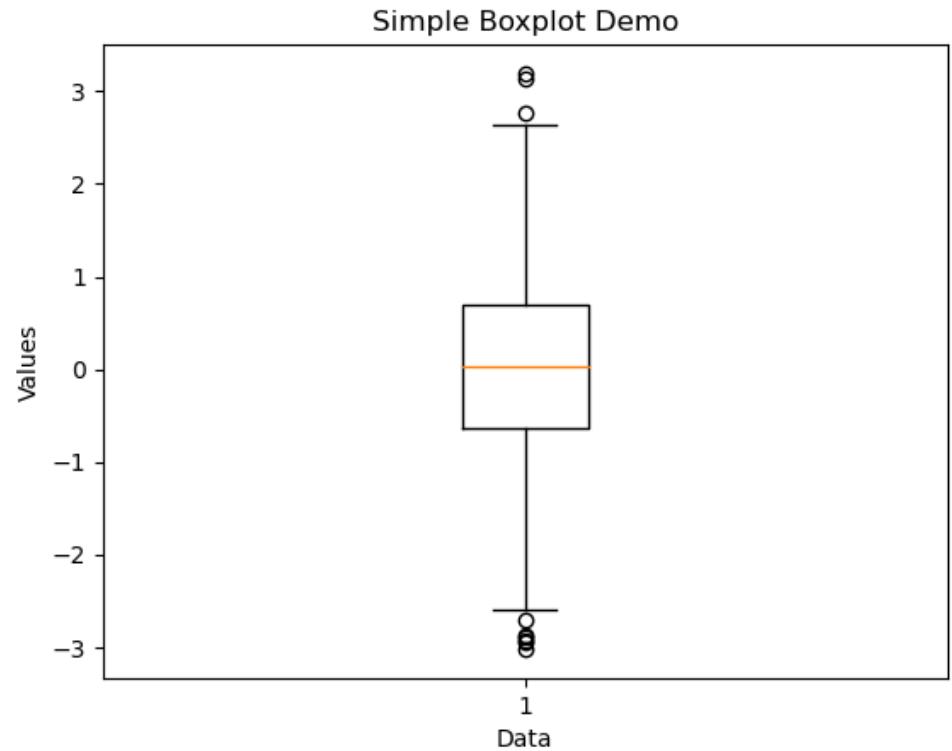
```
import matplotlib.pyplot as plt
import numpy as np

# Generate some random data
data = np.random.normal(0, 1, 100)

# Create a boxplot
plt.boxplot(data)

# Add labels and title
plt.title('Simple Boxplot Demo')
plt.xlabel('Data')
plt.ylabel('Values')

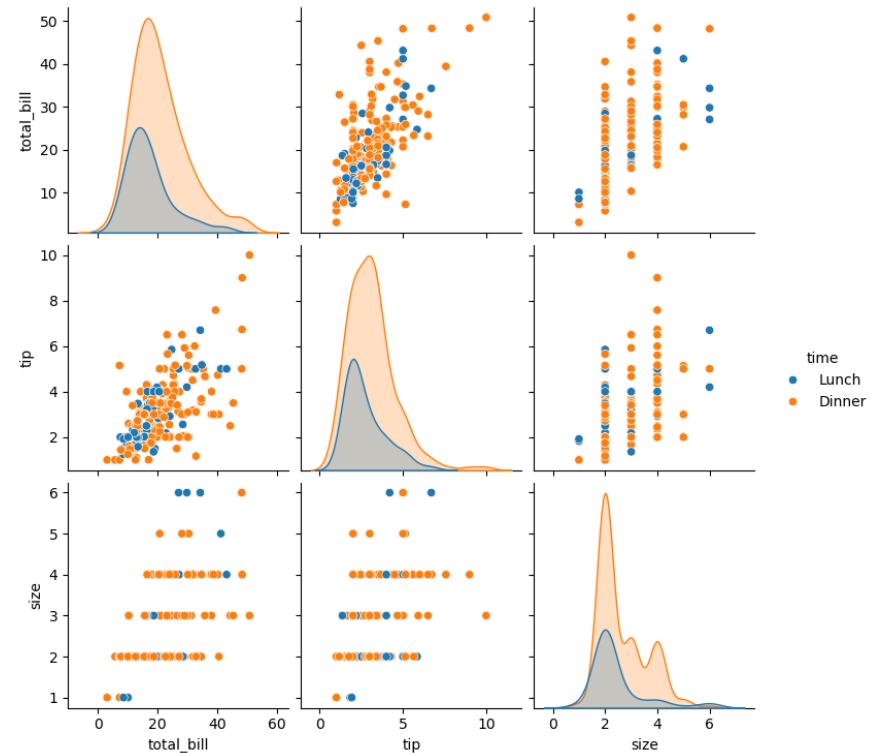
# Show the plot
plt.show()
```



Data Visualization: Seaborn Plots

<Pair Plot>

```
import seaborn as sns  
  
import matplotlib.pyplot as plt  
  
tips = sns.load_dataset('tips')  
sns.pairplot(tips, hue='time')  
plt.title('Pair Plot with Seaborn')  
plt.show()
```

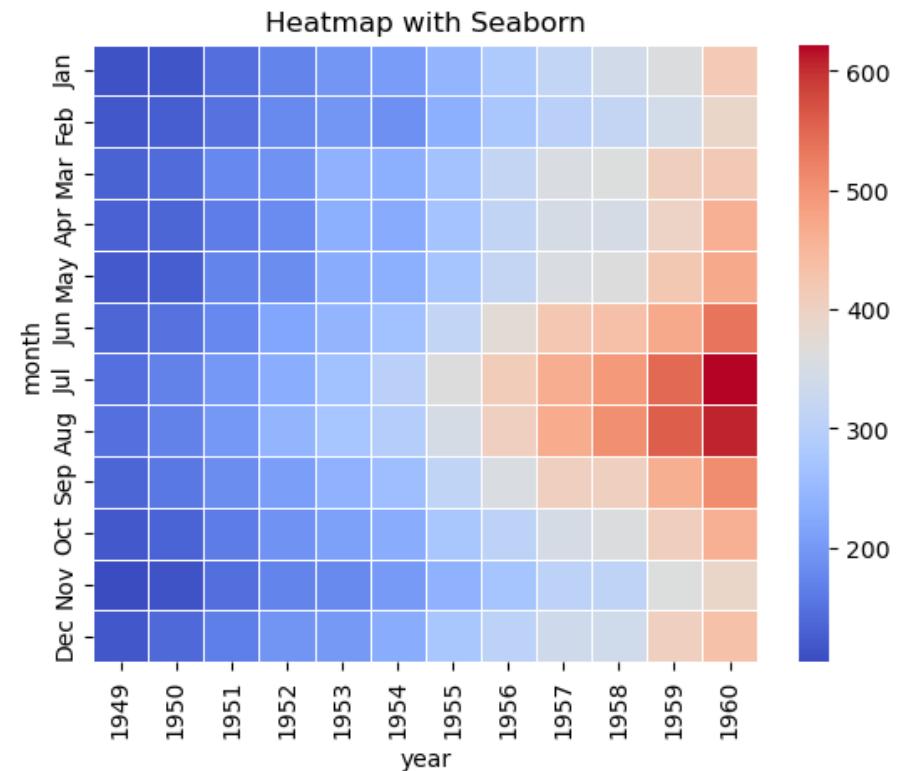


Data Visualization: Seaborn Plots

<Heatmap>

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

data = sns.load_dataset('flights')
sns.heatmap(data, cmap='coolwarm', annot=True)
plt.title('Heatmap with Seaborn')
plt.show()
```

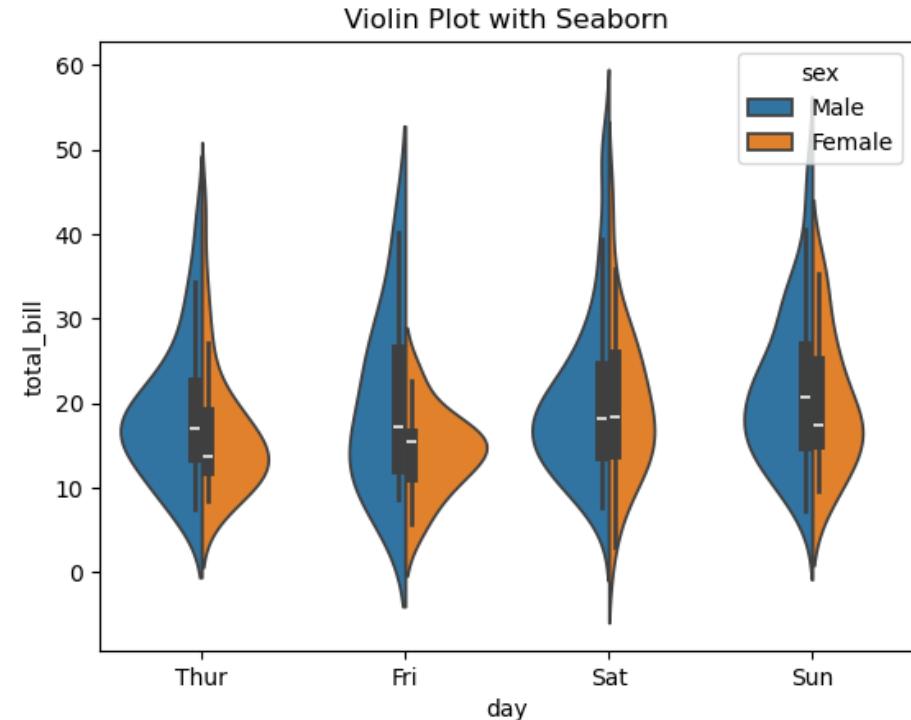


Data Visualization: Seaborn Plots

<Violin Plot>

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

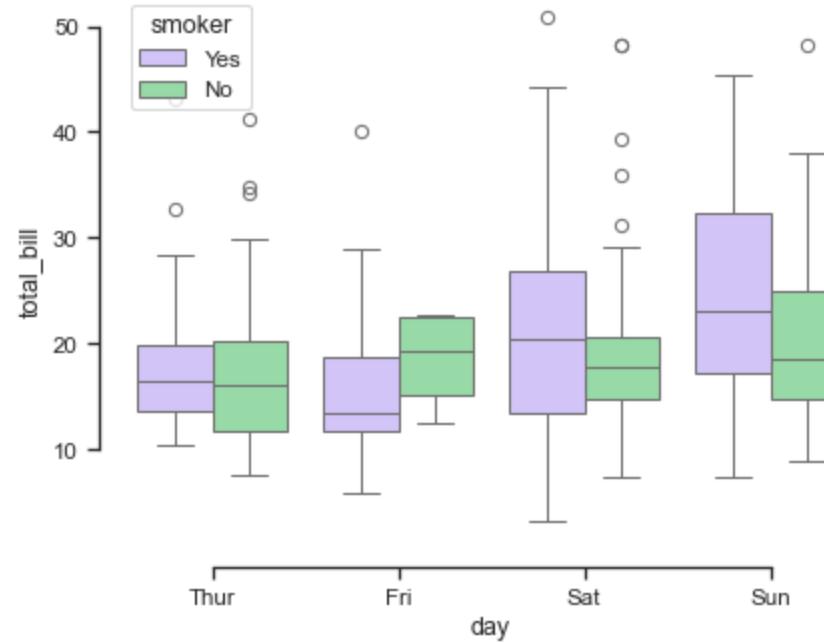
tips = sns.load_dataset('tips')
sns.violinplot(x='day', y='total_bill', data=tips)
plt.title('Violin Plot with Seaborn')
plt.show()
```



Data Visualization: Matplotlib and Seaborn Plots



- For more Matplotlib plots:
 - <https://matplotlib.org/>
- For more Seaborn plots:
 - <https://seaborn.pydata.org/>



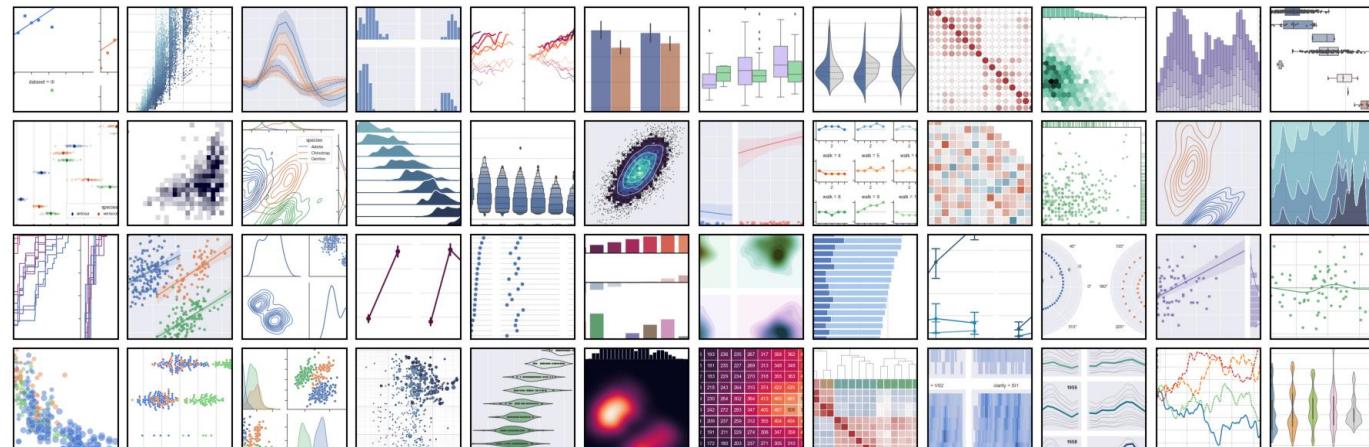
Data Visualization: Matplotlib and Seaborn Plots <DEMO>

Demo: Session 2 – Data Visualization and EDA



Data Visualization and EDA

Exploratory Data Analysis (EDA) is a **crucial** step in the data analysis process, allowing you to understand the main characteristics of your dataset. EDA involves examining and visualizing the data to identify patterns, relationships, and potential outliers. Univariate, bivariate, and multivariate analyses are key components of EDA, and they can be performed using both graphical and non-graphical methods.



Data Visualization and EDA: <Importance of EDA>

1. Data Understanding and Familiarity

EDA is a critical initial step in the data science workflow. It involves summarizing and familiarizing oneself with the main characteristics of the dataset, including its structure, distributions, and key statistics.

2. Data Cleaning and Preprocessing

Through EDA, data scientists can identify and address issues such as missing values, outliers, and inconsistencies in the dataset. This is crucial for ensuring the quality and reliability of the data before moving on to more advanced analyses.

3. Feature Engineering and Selection

EDA aids in the identification of relevant features for modeling. Data scientists can uncover patterns and relationships that inform decisions about which features to include or exclude in the model-building process.

Data Visualization and EDA: <Importance of EDA>

4. Hypothesis Generation

EDA helps in generating hypotheses about the relationships within the data. By visualizing and exploring data distributions and correlations, data scientists can form initial hypotheses that guide further analysis.

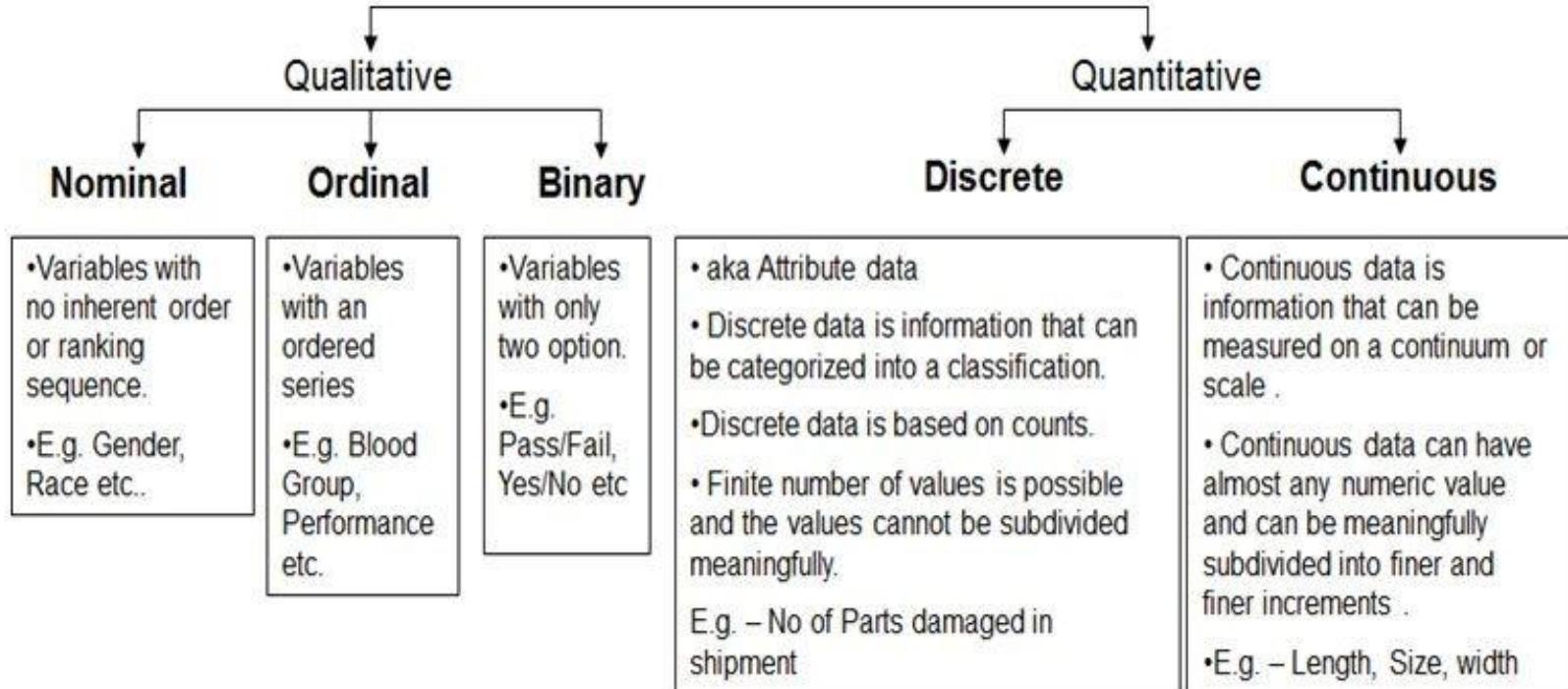
5. Model Assumptions and Validations

EDA assists in validating assumptions made during the modeling process. Understanding the underlying assumptions of statistical models and verifying their applicability to the dataset ensures the reliability of the analysis.

6. Communication with Stakeholders

EDA provides insights that can be effectively communicated to stakeholders. Whether it's presenting initial findings or explaining the nuances of the dataset, EDA supports clear and transparent communication.

Data Visualization and EDA: <Numerical vs. Categorical Data>

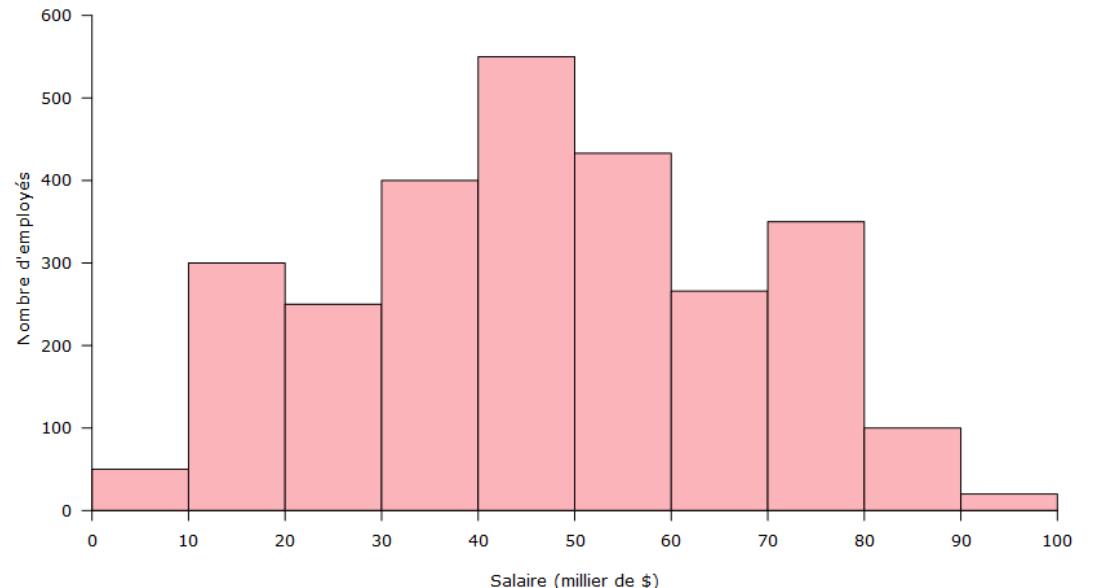


Data Visualization and EDA: <Numerical vs. Categorical Data>

In **data science**, data can be broadly categorized into **numerical** and **categorical** types based on their nature and characteristics:

- **Numerical Data:**
 - **Definition:** Numerical data consists of numbers and represents measurable quantities. It can be further categorized into discrete and continuous data.
 - **Examples:**
 - **Discrete Numerical Data:** Count of items, number of people, etc.
 - **Continuous Numerical Data:** Height, weight, temperature, etc.
 - **Analysis Techniques:**
 - **Descriptive statistics:** Mean, median, mode, range, etc.
 - **Inferential statistics:** Regression analysis, hypothesis testing, etc.
 - **Visualization:** Histograms, box plots, scatter plots, etc.

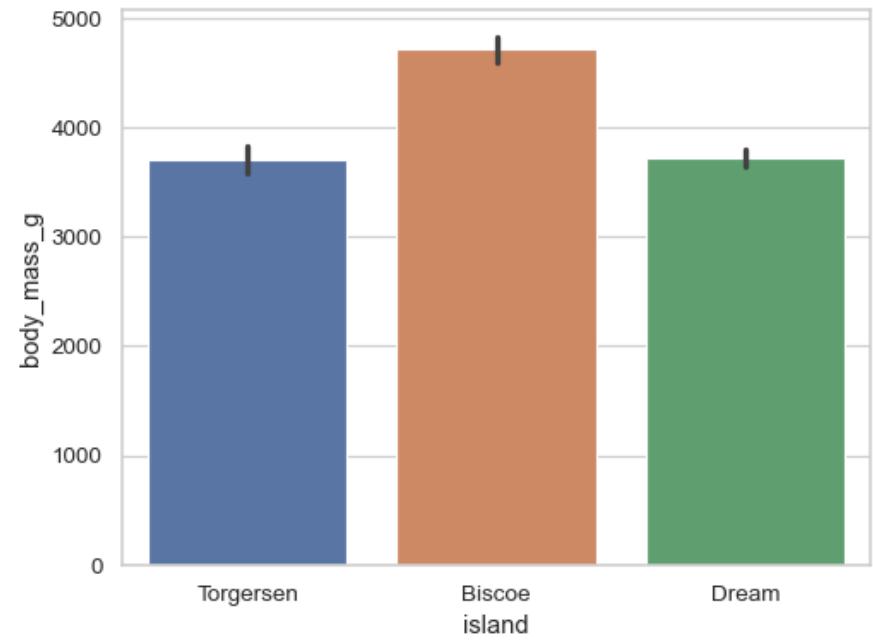
Graphique 5.7.1
 Distribution des salaires des employés de la société ABC



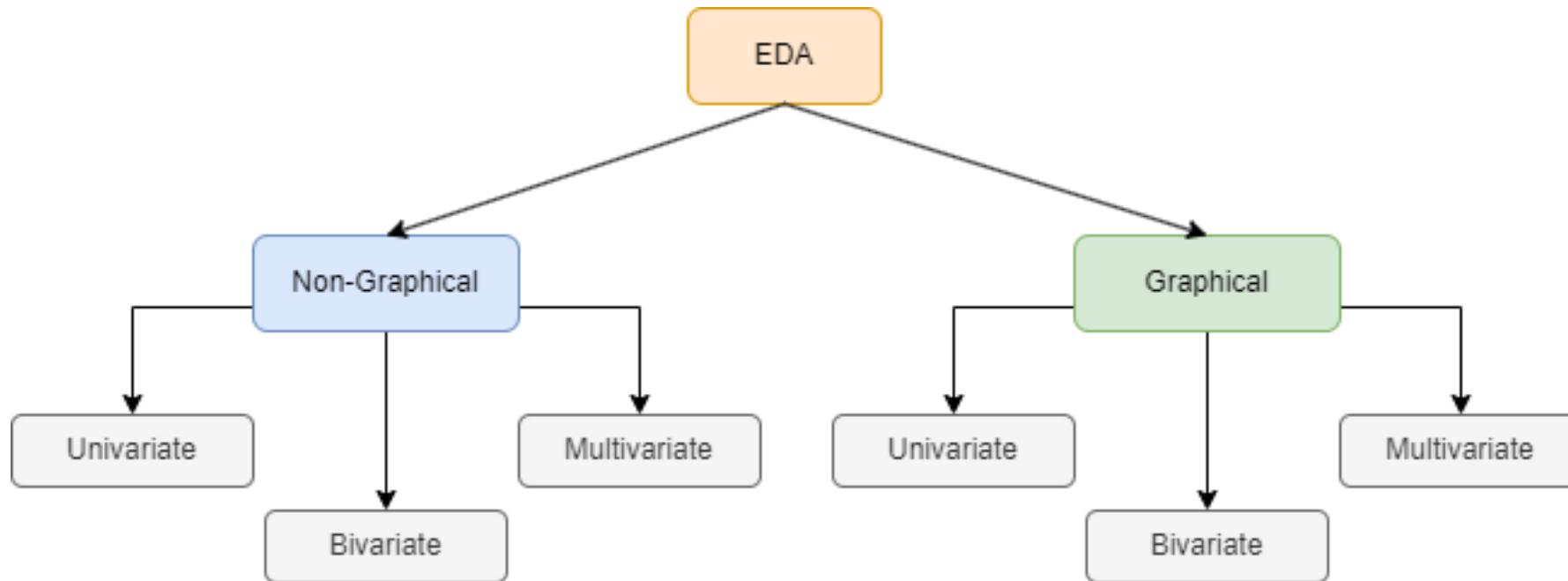
Data Visualization and EDA: <Numerical vs. Categorical Data>

In **data science**, data can be broadly categorized into **numerical** and **categorical** types based on their nature and characteristics:

- **Categorical Data:**
 - **Definition:** Categorical data represents categories or labels and cannot be measured in a numerical sense. It can be nominal or ordinal.
 - **Examples:**
 - **Nominal Categorical Data:** Colors, gender, country, etc.
 - **Ordinal Categorical Data:** Education levels, customer satisfaction ratings, etc.
 - **Analysis Techniques:**
 - Frequency counts and proportions.
 - Cross-tabulations and contingency tables.
 - **Visualization:** Bar charts, pie charts, stacked bar charts, etc.



Data Visualization and EDA: Univariate, Bivariate, and Multivariate Analysis

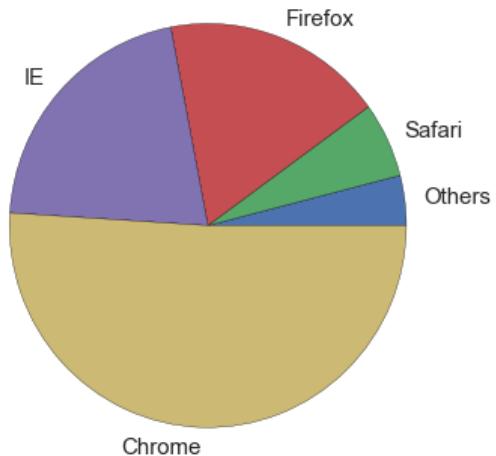


Data Visualization and EDA: Univariate, Bivariate, and Multivariate Analysis <Categorical Data>

- **Univariate Analysis:**

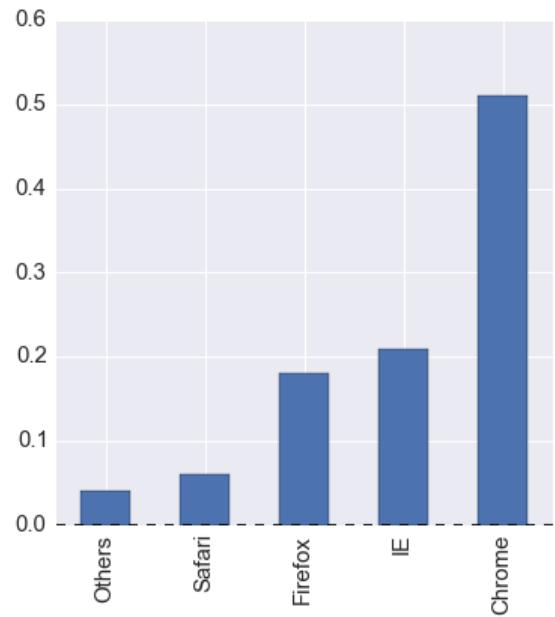
- **Graphical Methods:**

- **Bar charts:** Display the frequency or proportion of each category.
 - **Pie charts:** Show the relative contribution of each category to the whole.



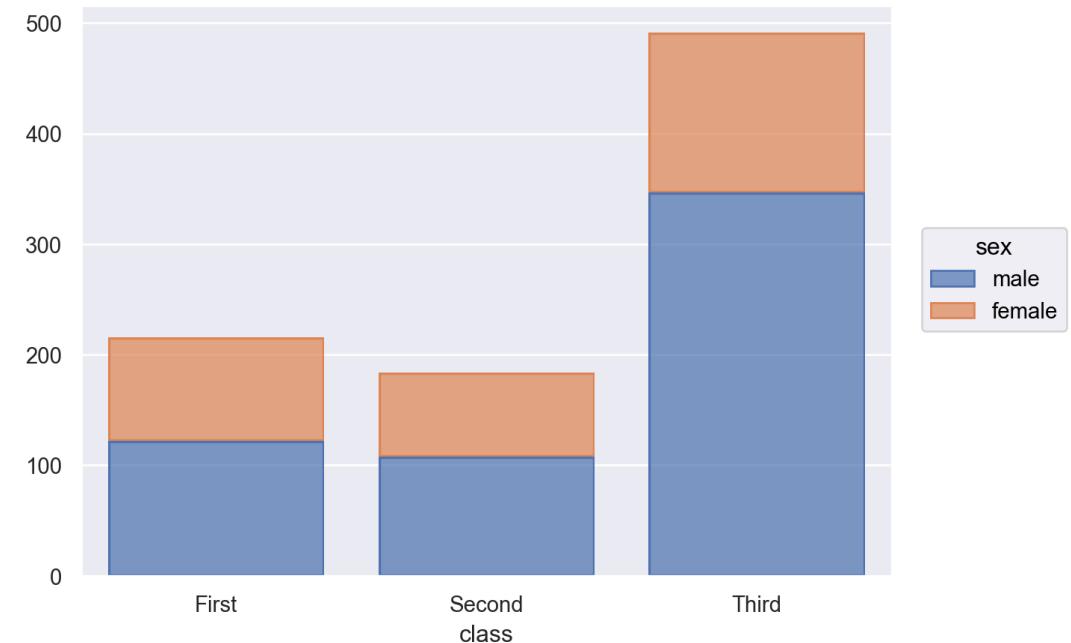
- **Non-graphical Methods:**

- **Frequency tables:** Summarize the count of each category.
 - **Mode:** Identify the most frequently occurring category.



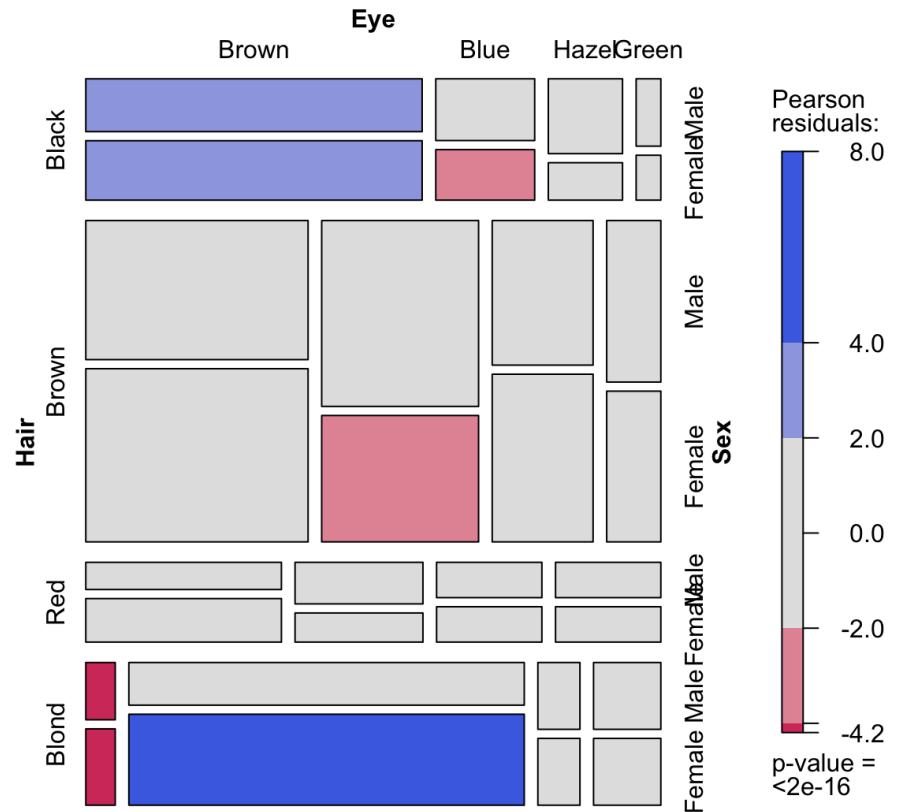
Data Visualization and EDA: Univariate, Bivariate, and Multivariate Analysis <Categorical Data>

- **Bivariate Analysis:**
 - **Graphical Methods:**
 - **Clustered bar charts:** Compare the distribution of one categorical variable across different levels of another categorical variable.
 - **Stacked bar charts:** Illustrate the composition of one categorical variable relative to another.
 - **Non-graphical Methods:**
 - **Chi-square test:** Assess the independence between two categorical variables.



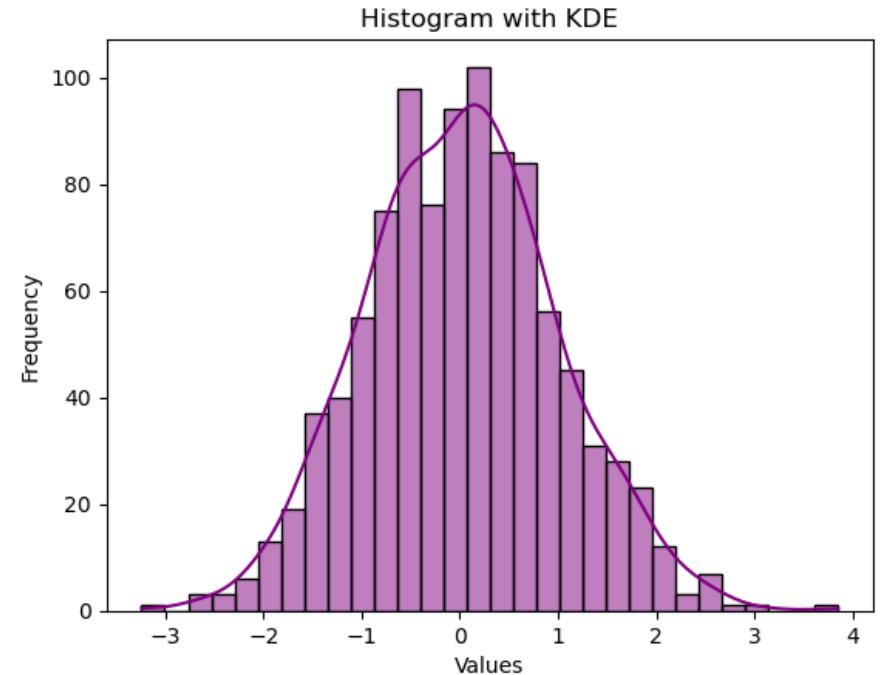
Data Visualization and EDA: Univariate, Bivariate, and Multivariate Analysis <Categorical Data>

- **Multivariate Analysis:**
 - **Graphical Methods:**
 - **Mosaic plots:** Visualize the relationship between three categorical variables.
 - **Clustered stacked bar charts:** Represent multiple categorical variables simultaneously.
 - **Non-graphical Methods:**
 - **Multinomial logistic regression:** Examine the relationship between multiple categorical variables.



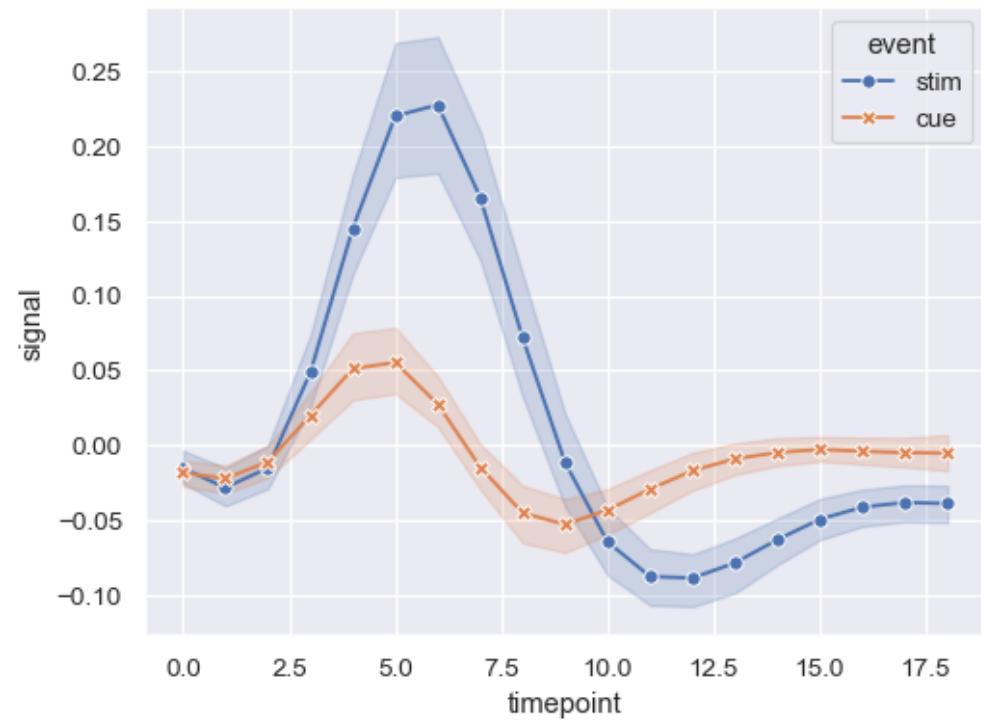
Data Visualization and EDA: Univariate, Bivariate, and Multivariate Analysis <Numerical Data>

- **Univariate Analysis:**
 - **Graphical Methods:**
 - Histograms: Display the distribution of numerical values.
 - Box plots: Show summary statistics and identify outliers.
 - Kernel Density Plots: Estimate the probability density function
 - **Non-graphical Methods:**
 - Descriptive statistics: Mean, median, mode, variance, standard deviation.
 - Percentiles and quantiles: Identify values below which a given percentage of observations fall.
 - Skewness and kurtosis: Measure of asymmetry and tailedness.



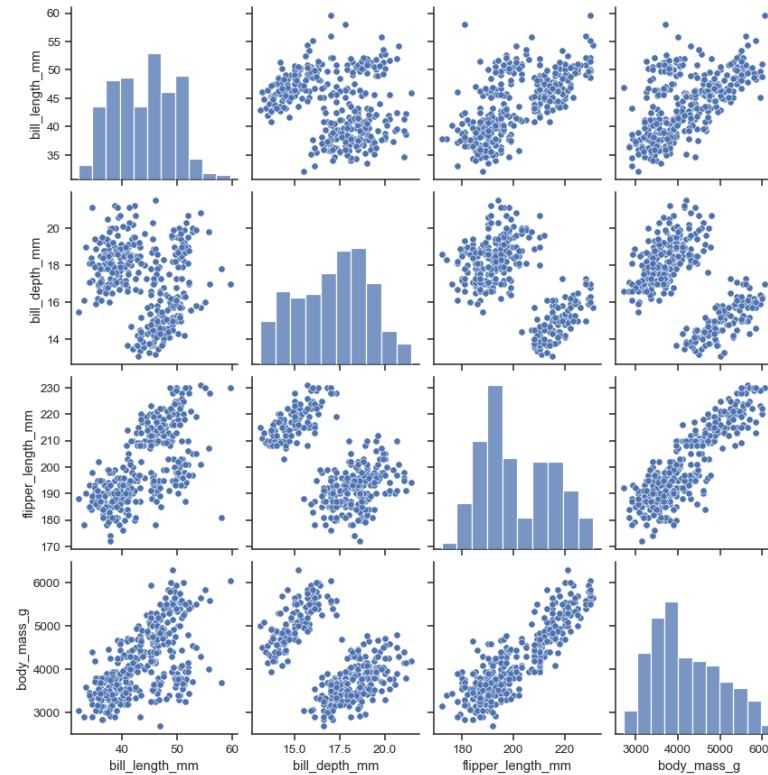
Data Visualization and EDA: Univariate, Bivariate, and Multivariate Analysis <Numerical Data>

- **Bivariate Analysis:**
 - **Graphical Methods:**
 - Scatter plots: Explore the relationship between two numerical variables.
 - Line plots: Show trends over time or across a variable.
 - Heatmaps: Visualize the correlation matrix between two numerical variables.
 - **Non-graphical Methods:**
 - Correlation coefficient: Quantify the strength and direction of a linear relationship.
 - Covariance: Measure the joint variability of two numerical variables.



Data Visualization and EDA: Univariate, Bivariate, and Multivariate Analysis <Numerical Data>

- **Multivariate Analysis:**
 - **Graphical Methods:**
 - 3D Scatter plots: Extend scatter plots to three dimensions for multiple numerical variables.
 - Pair plots: Combine scatter plots and histograms for multiple numerical variables.
 - **Non-graphical Methods:**
 - Multiple regression analysis: Assess the relationship between one dependent numerical variable and multiple independent numerical variables.
 - Principal Component Analysis (PCA): Reduce dimensionality while preserving as much variance as possible.



Data Visualization and EDA: General Tips

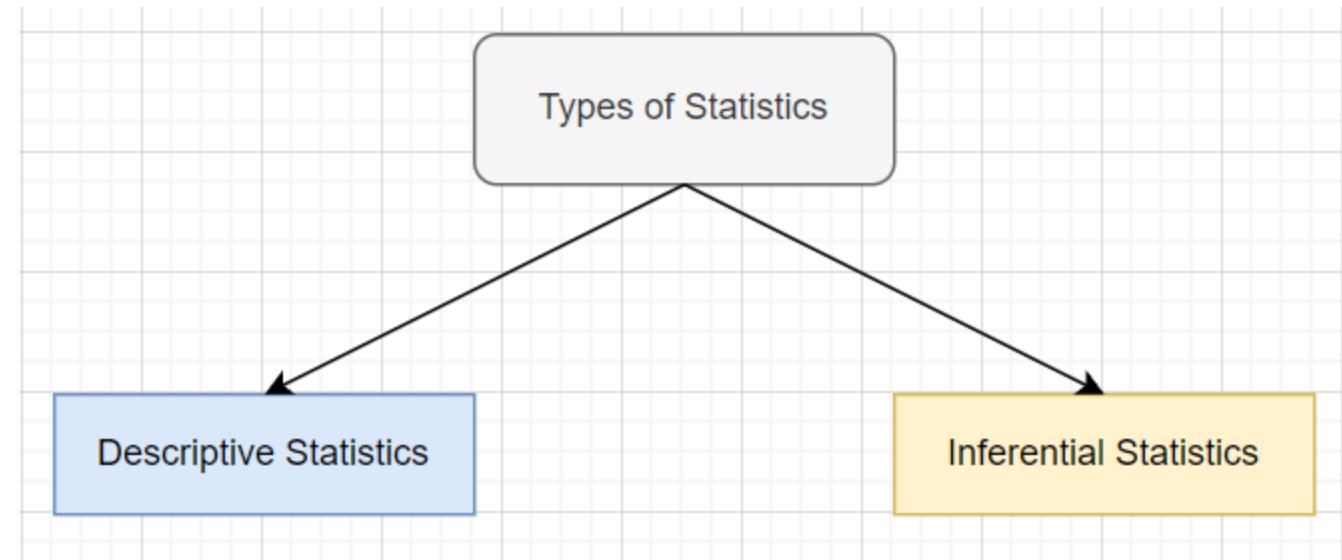
- **Data Cleaning:** You can perform EDA before and after cleaning the data by handling missing values, outliers, and ensuring data consistency.
- **Visualization Libraries:** Use tools like Matplotlib, Seaborn, and Plotly for graphical analysis in Python. R has ggplot2, lattice, and other packages.
- **Interpretation:** Always interpret the results of your analyses, considering the context of the data and potential implications for further analysis or decision-making.
- **Note:** Remember, EDA is an iterative process, and the insights gained can guide further analysis or modeling efforts.

Data Visualization and EDA: Statistics Fundamentals

- **Definition 1:** Statistics is a branch of mathematics that involves collecting, analyzing, interpreting, presenting, and organizing data. It provides methods for making inferences and decisions in the face of uncertainty. The goal of statistics is to extract meaningful patterns and insights from data, helping to understand and describe complex phenomena.
- **Definition 2:** Statistics is the science of learning from data. It involves collecting, analyzing, interpreting, presenting, and organizing data. It provides methods for drawing reliable conclusions and making informed decisions in the face of uncertainty.

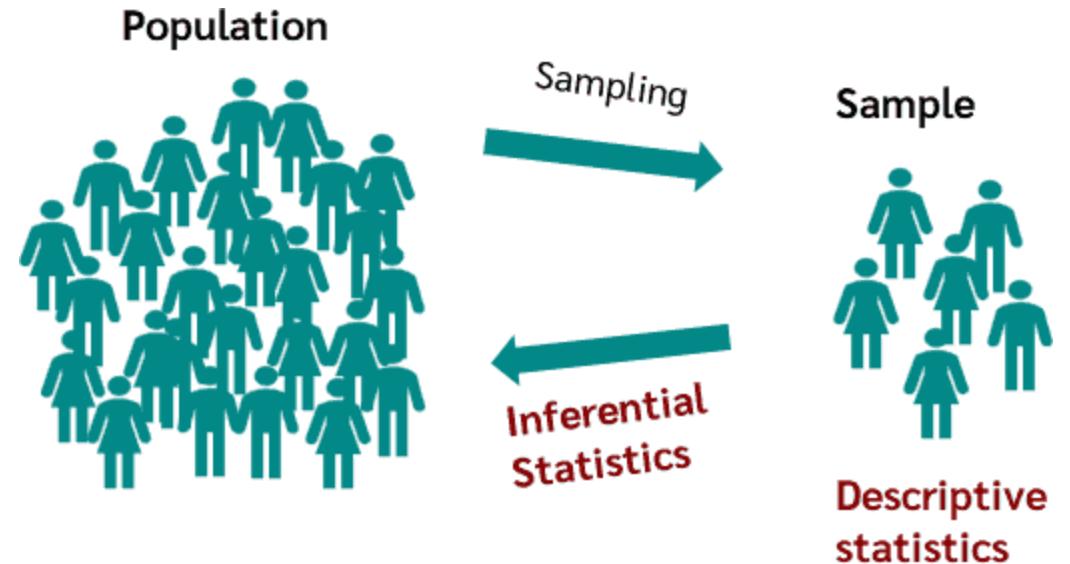


Data Visualization and EDA: Statistics Fundamentals <Key Components>



Data Visualization and EDA: Statistics Fundamentals <Key Components>

Descriptive statistics and **inferential** statistics, along with exploratory statistics, are the main areas of statistics. **Descriptive** statistics provides tools to **describe** a sample. Starting from the sample, **inferential** statistics can now be used to make a statement about the population.

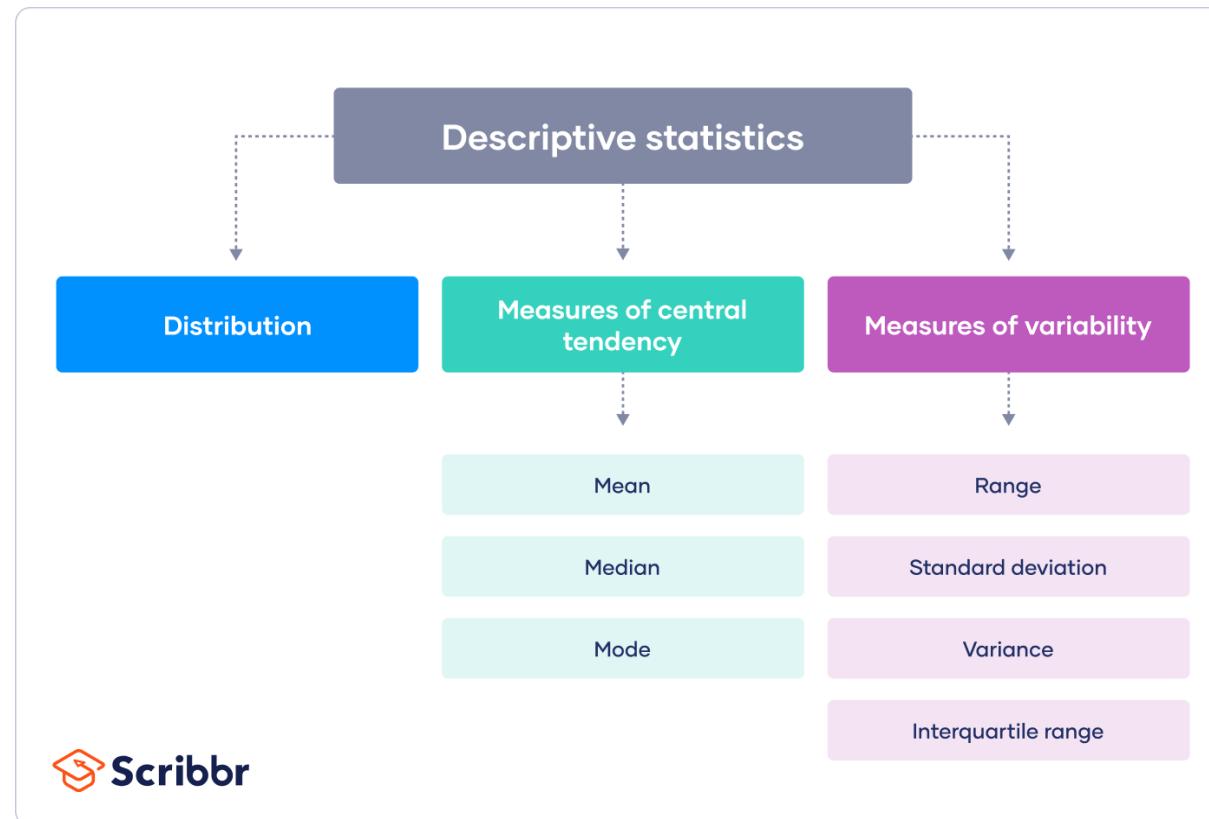


Source: [Datalab.net](https://databl.net)

Data Visualization and EDA: Statistics Fundamentals <Key Components>

- **Descriptive Statistics:** This branch involves summarizing and describing the main features of a dataset. Descriptive statistics include measures such as mean, median, mode, standard deviation, and various graphical representations like histograms and scatter plots. These methods help in organizing and simplifying large amounts of data to make it more understandable.
- **Definition 2:** The term descriptive statistics covers statistical methods for describing data using statistical characteristics, charts, graphics or tables.

Data Visualization and EDA: Statistics Fundamentals <Key Components>



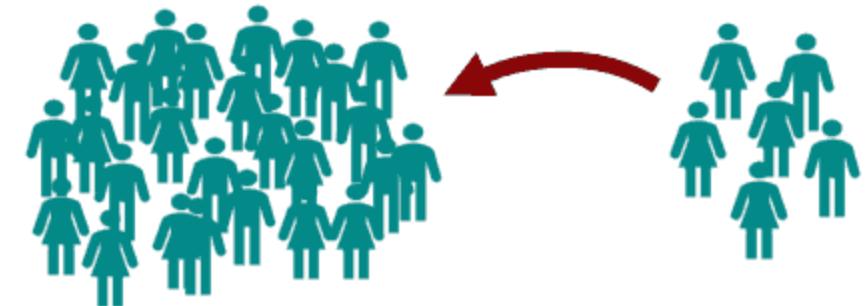
Source: [Scribbr.com](https://www.scribbr.com)

Data Visualization and EDA: Statistics Fundamentals <Key Components>

- **Inferential Statistics:** This branch uses statistical methods to make predictions or inferences about a population based on a sample of data taken from that population. Inferential statistics include hypothesis testing, confidence intervals, regression analysis, and more. These methods help researchers draw conclusions from a limited set of observations.
- **Definition 2:** What's inferential statistics? In contrast to descriptive statistics, inferential statistics want to make a statement about the population. However, since it is almost impossible in most cases to survey the entire population, a sample is used, i.e. a small data set originating from the population. With this sample a statement about the population can be made. An example would be if a sample of 1,000 citizens is taken from the population of all Canadian citizens.

Inferential statistics:

Testing statements about the population on the basis of sample characteristics.



Source: [Datalab.net](https://www.datalab.net)

Data Visualization and EDA: Statistics Fundamentals <Key Components>

Depending on which statement is to be made about the population or which question is to be answered about the population, different statistical methods or hypothesis tests are used. The best known are the hypothesis tests with which a group difference can be tested, such as the t-test, the chi-square test or the analysis of variance. Then there are the hypothesis tests with which a correlation of variables can be tested, such as correlation analysis and regression.

Simple test procedures

- t-Test
- Binomial Test
- Chi-square test
- Mann-Whitney U Test
- Wilcoxon-Test
- ...

Correlation analysis

- Pearson Correlation analysis
- Spearman Rank Correlation
- ...

Regression Analysis

- Simple linear regression
- Multiple regression
- Logistic regression
- ...

ANOVA

- Single factorial ANOVA
- Two factorial ANOVA
- ANOVA with measurement repetitions
- ...

Source: [Datalab.net](https://www.datalab.net)

Data Visualization and EDA: Statistics Fundamentals <Key Components>

Difference Between Descriptive and Inferential Statistics

Descriptive statistics provide a **summary** of the features or attributes of a **dataset**, while **inferential** statistics enable **hypothesis testing** and evaluation of the applicability of the data to a larger population.

Data Visualization and EDA: Statistics Fundamentals <Key Components>

Difference Between Descriptive and Inferential Statistics

| | Descriptive Statistics | Inferential Statistics |
|------------------------------|---|--|
| Purpose | Describe and Summarize data | Make inferences and draw conclusions about a population based on sample data |
| Data Analysis | Analyzes and interprets the characteristics of a dataset | Uses sample data to make generalizations or predictions about a larger population |
| Population vs. Sample | Focuses on the entire population or a dataset | Focuses on a subset of the population (sample) to draw conclusions about the entire population |
| Measurements | Provides measures of central tendency and dispersion | Estimates parameters, tests hypotheses, and determines the level of confidence or significance in the results |
| Examples | Mean, median, mode, standard deviation, range, frequency tables | Hypothesis testing, confidence intervals, regression analysis, ANOVA (analysis of variance), chi-square tests, t-tests, etc. |

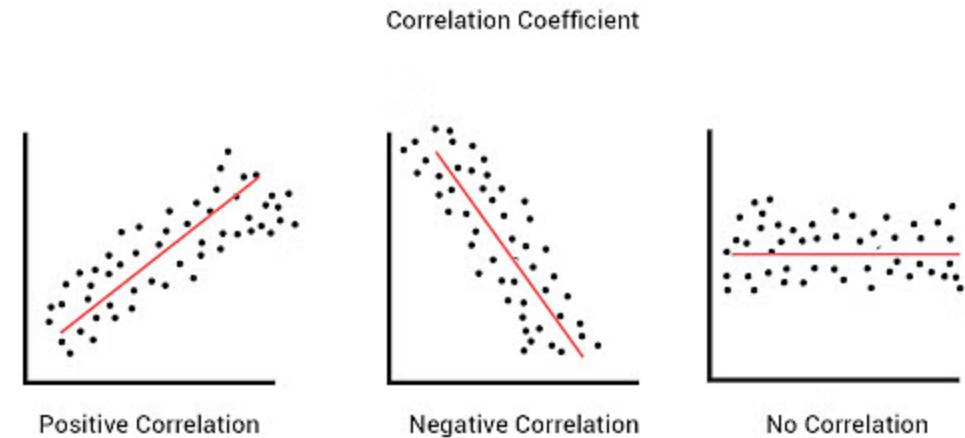
Data Visualization and EDA: Statistics Fundamentals <Key Components>

Difference Between Descriptive and Inferential Statistics

| | Descriptive Statistics | Inferential Statistics |
|----------------------------------|-------------------------------------|--|
| Goal | Summarize organize and present data | Generalize findings to a larger population, make predictions, test hypotheses, evaluate relationships, and support decision-making |
| Population Parameters | Not typically estimated | Estimated using sample statistics (e.g., sample mean as an estimate of population mean) |
| Sample Representativeness | Not required | Crucial; the sample should be representative of the population to ensure accurate inferences |

Data Visualization and EDA: Statistics Fundamentals <Correlation>

Correlation in statistics refers to the statistical relationship or association between two or more variables. It measures the degree to which changes in one variable are associated with changes in another. In other words, correlation helps to quantify the strength and direction of a linear relationship between two variables. The two most common measures of correlation are the Pearson correlation coefficient and the Spearman rank correlation coefficient.



Data Visualization and EDA: Statistics Fundamentals <Pearson Correlation Coefficient>

Pearson Correlation Coefficient:

- Denoted by r , the Pearson correlation coefficient ranges from -1 to 1.
- $r=1$ indicates a perfect positive linear relationship.
- $r=-1$ indicates a perfect negative linear relationship.
- $r=0$ indicates no linear relationship.
- Positive values of r signify a positive association, while negative values indicate a negative association.

Note: It cannot determine the nonlinear relationships between variables.

Formula

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

r = correlation coefficient

x_i = values of the x-variable in a sample

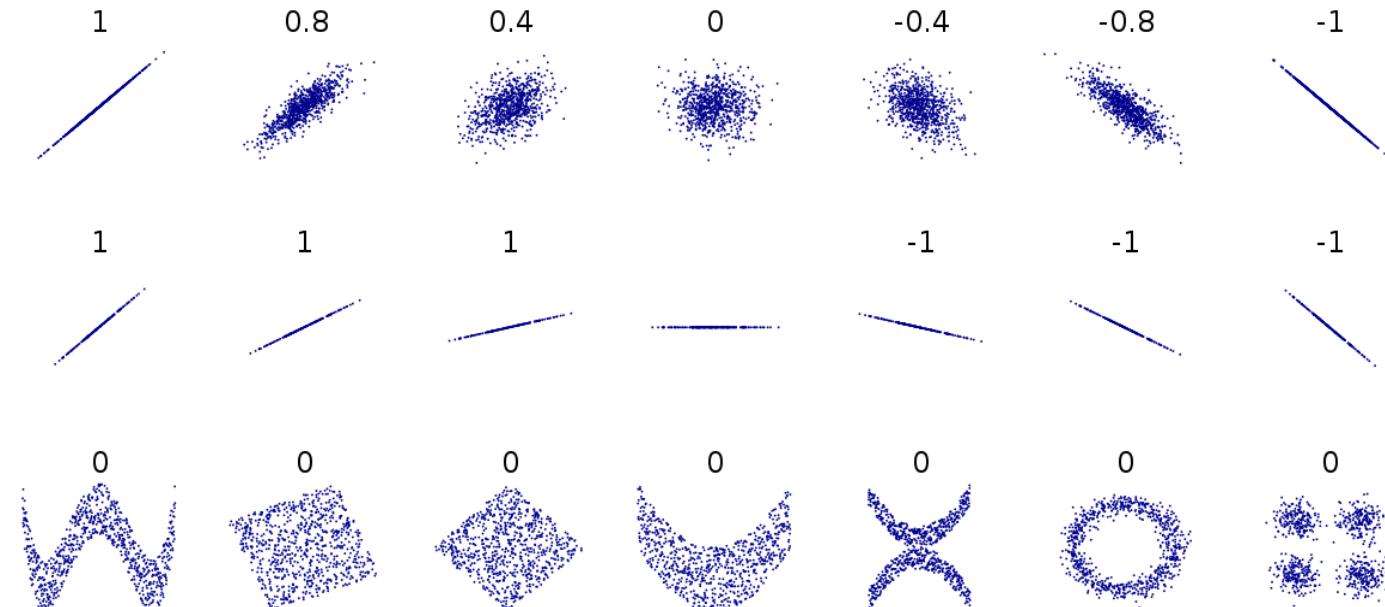
\bar{x} = mean of the values of the x-variable

y_i = values of the y-variable in a sample

\bar{y} = mean of the values of the y-variable

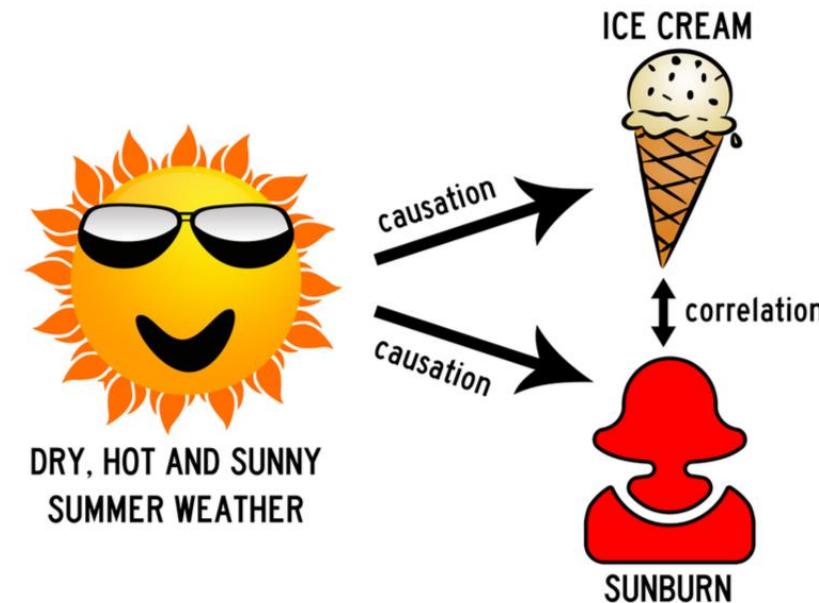
Data Visualization and EDA: Statistics Fundamentals <Pearson Correlation Coefficient>

Note: It cannot determine the nonlinear relationships between variables.



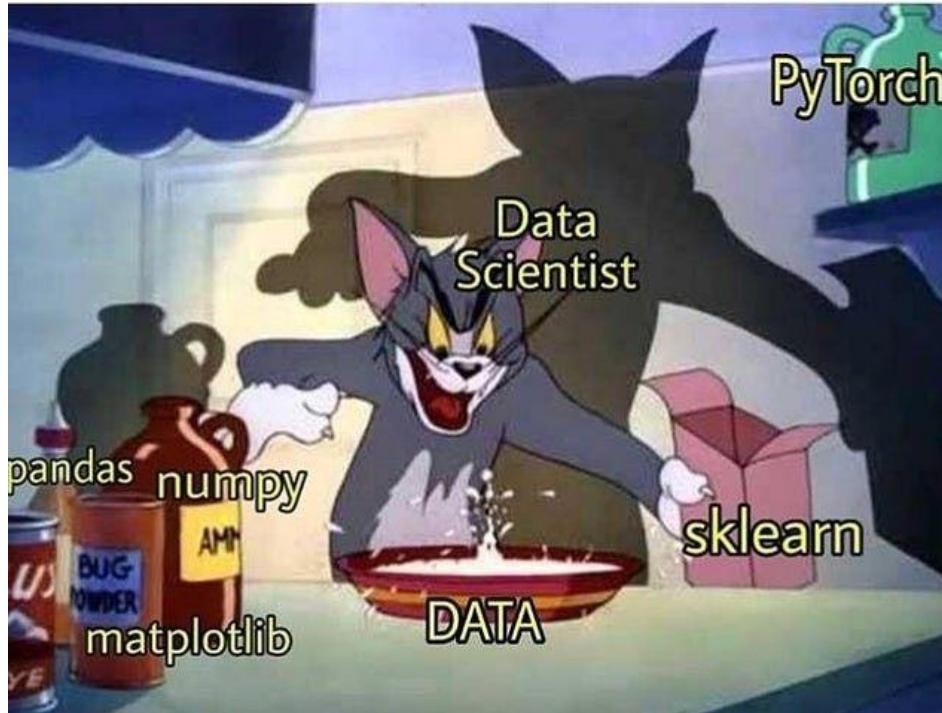
Data Visualization and EDA: Statistics Fundamentals <Correlation != Causation>

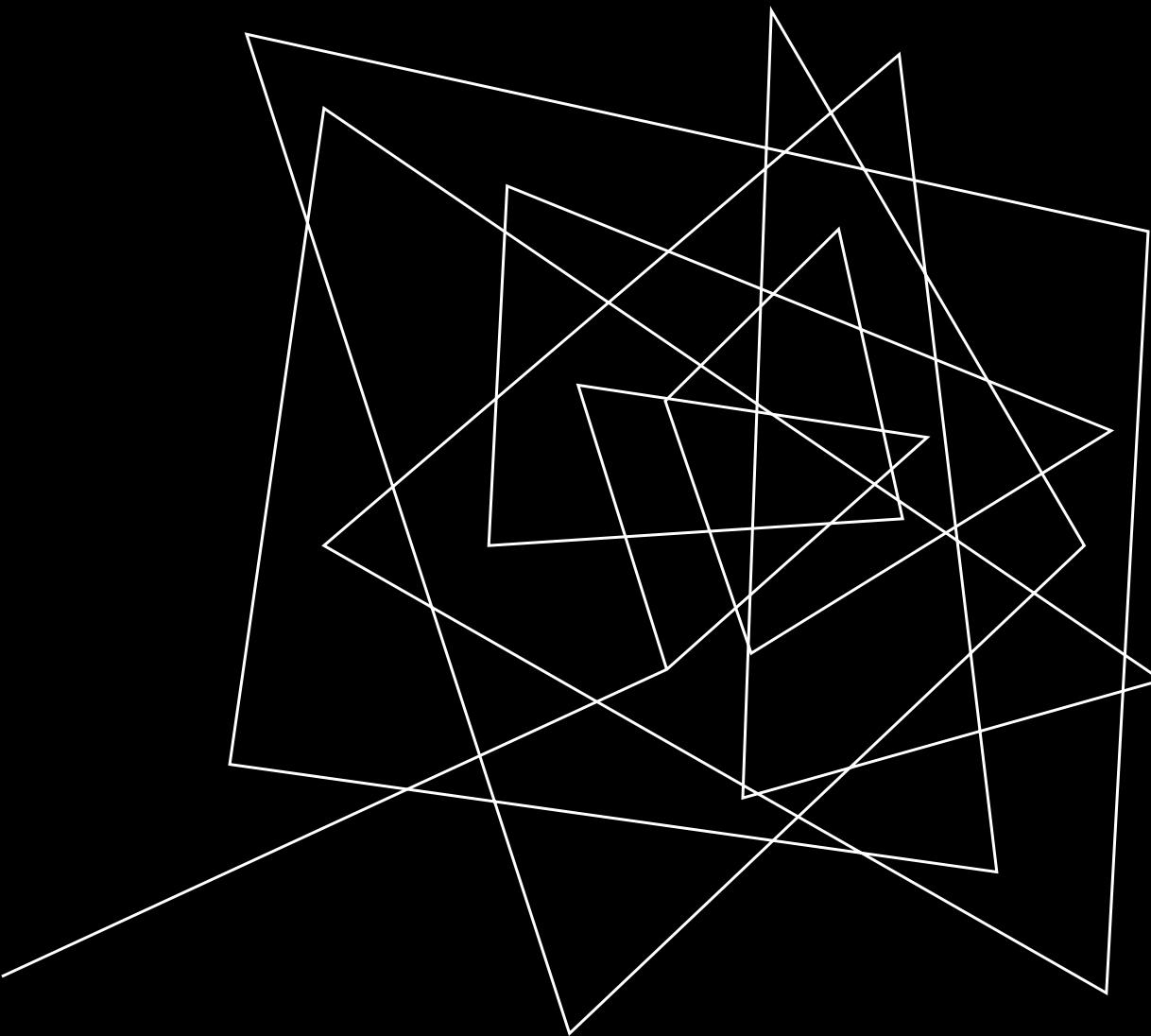
Correlation tests for a relationship between two variables. However, seeing two variables moving together does not necessarily mean we know whether one variable causes the other to occur. This is why we commonly say “correlation does not imply causation.”



Data Visualization and EDA

Demo: [Session 2 – Data Visualization and EDA](#)

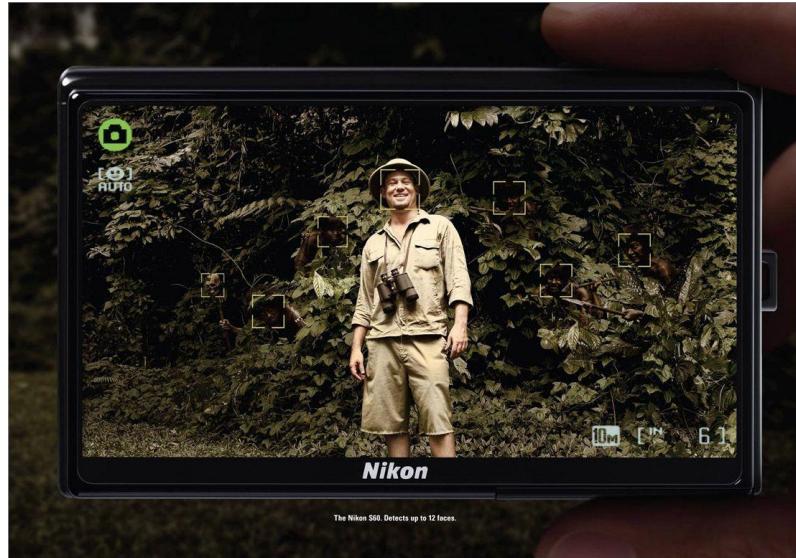




SESSION 3: MACHINE LEARNING FUNDAMENTALS

Overview of machine learning concepts and introduce popular machine learning algorithms.

Introduction: Machine learning is everywhere!



Face Recognition



Object recognition (e.g., for self-driving cars)

Introduction: Machine learning is everywhere!



Figure 1. Examples of retinal fundus photographs that are taken to screen for DR. The image on the left is of a healthy retina (A), whereas the image on the right is a retina with referable diabetic retinopathy (B) due to a number of hemorrhages (red spots) present.

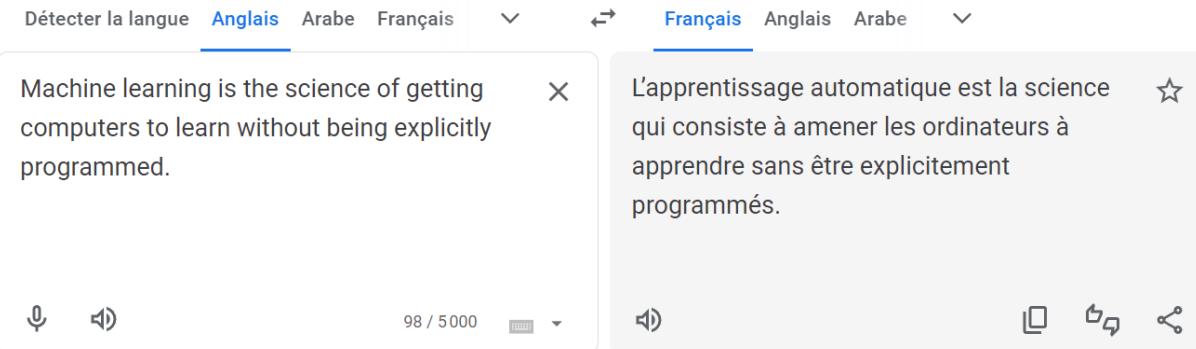
Analysis of medical images

Source: [A Practitioner's Guide to Machine Learning](#)



Conversational agents (i.e., speech recognition...)

Introduction: Machine learning is everywhere!



Détecer la langue **Anglais** Arabe Français ▾ ↔ **Français** Anglais Arabe ▾

Machine learning is the science of getting computers to learn without being explicitly programmed.

L'apprentissage automatique est la science qui consiste à amener les ordinateurs à apprendre sans être explicitement programmés.

98 / 5000

Machine Translation

**You**

What neural network architecture are you built on?

**ChatGPT**

I'm built on the GPT-3.5 architecture, which stands for "Generative Pre-trained Transformer 3.5." The underlying architecture is a transformer neural network developed by OpenAI. The transformer architecture is known for its effectiveness in natural language processing tasks and has been widely used in various language models.

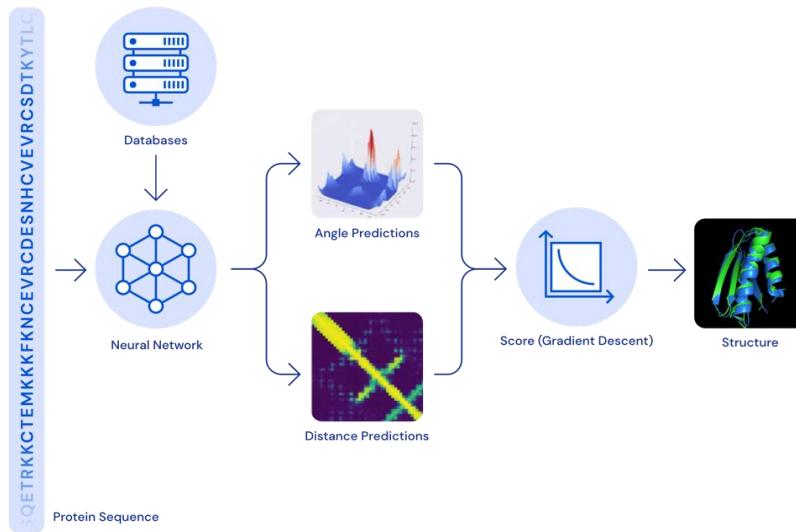
GPT-3.5 is one of the latest versions of the GPT series, featuring 175 billion parameters, making it one of the largest language models created by OpenAI as of my last knowledge update in January 2022. If there have been any updates or new models since then, I may not be aware of them.

Advanced Chatbots

Introduction: Machine learning is everywhere!

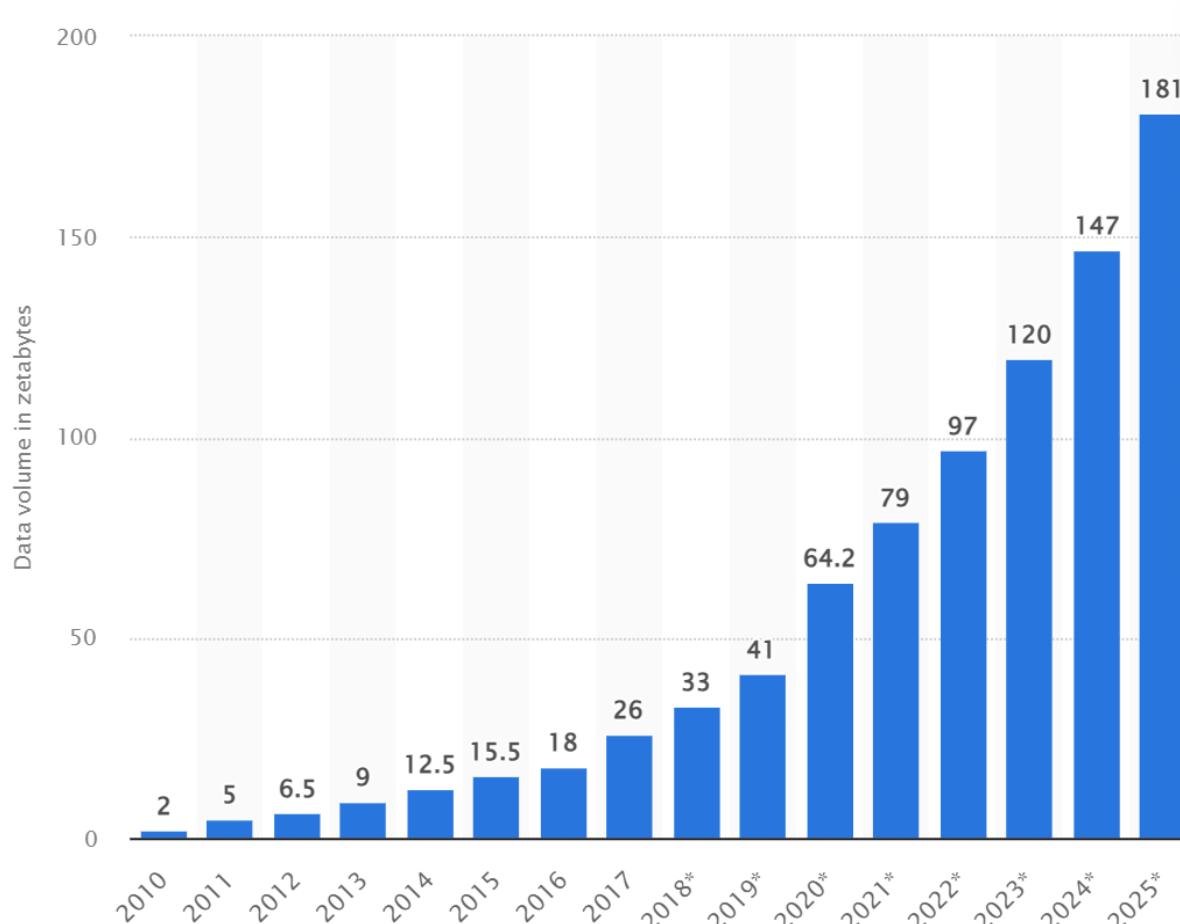


AlphaGo beats Lee Sedol



Solving protein folding challenge
(AlphaFold)

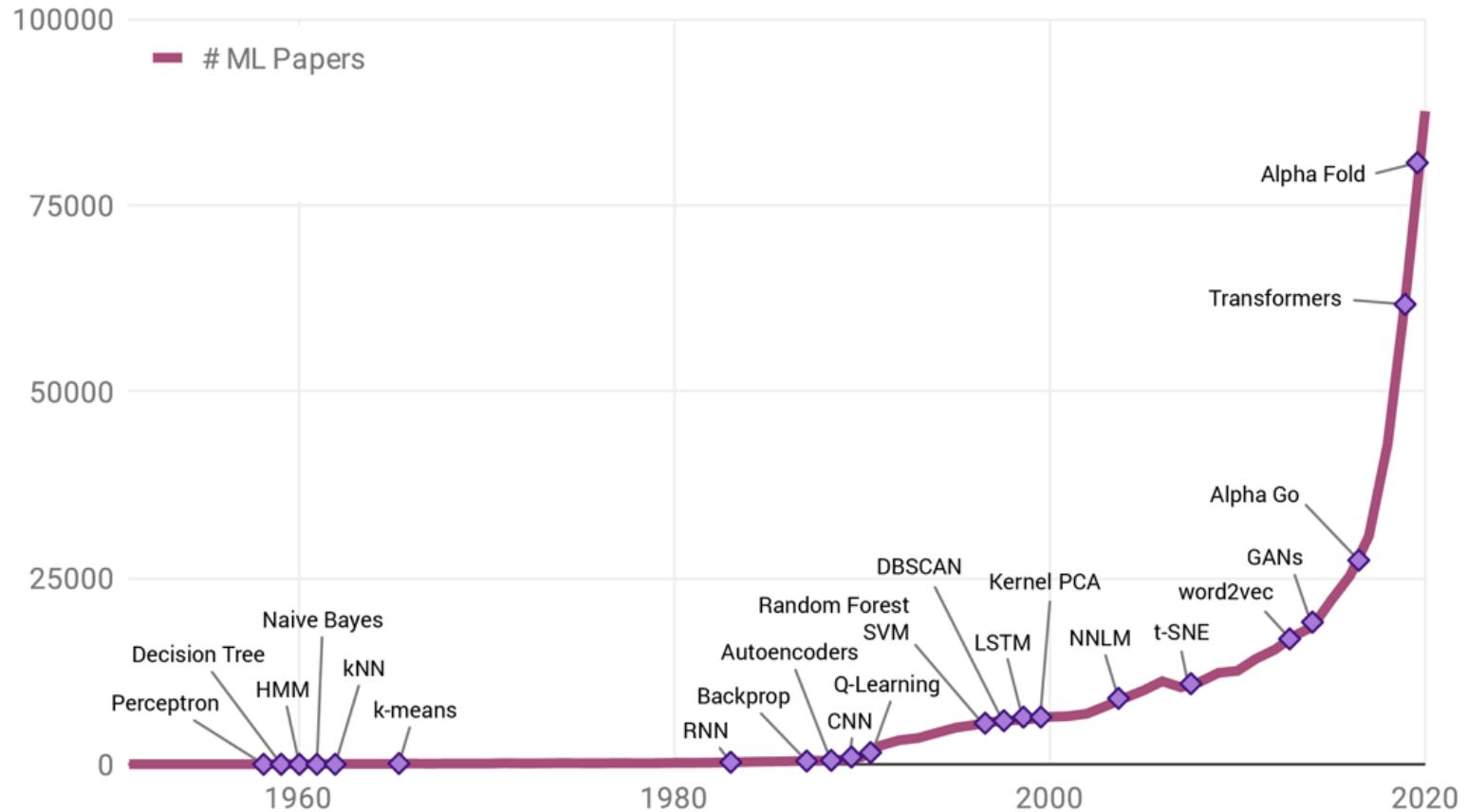
Introduction: Why is ML so popular? <Increased volumes of data>



Source: [Statista.com](https://www.statista.com)

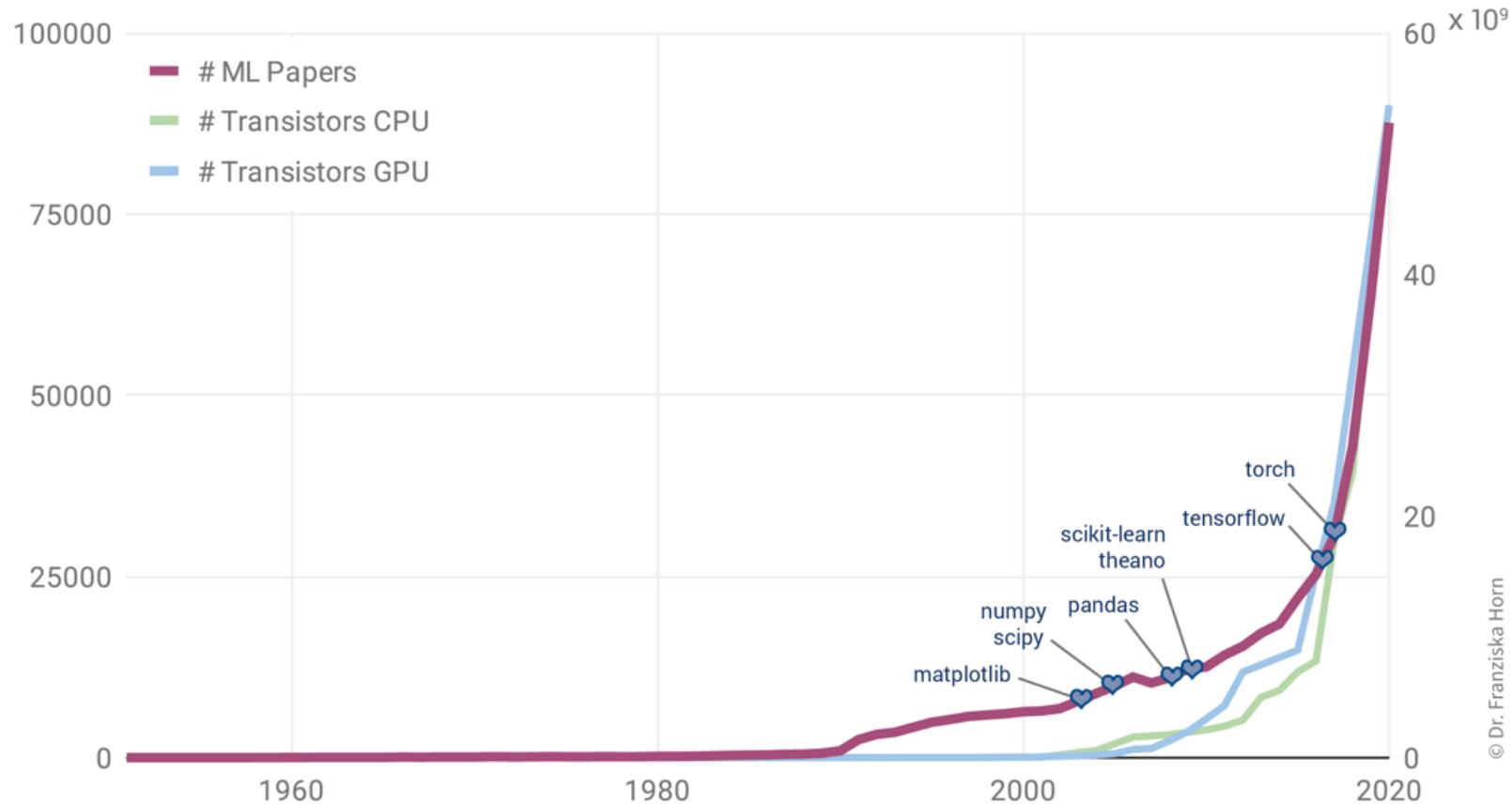
Introduction: Why is ML so popular?

<Advanced Algorithms>



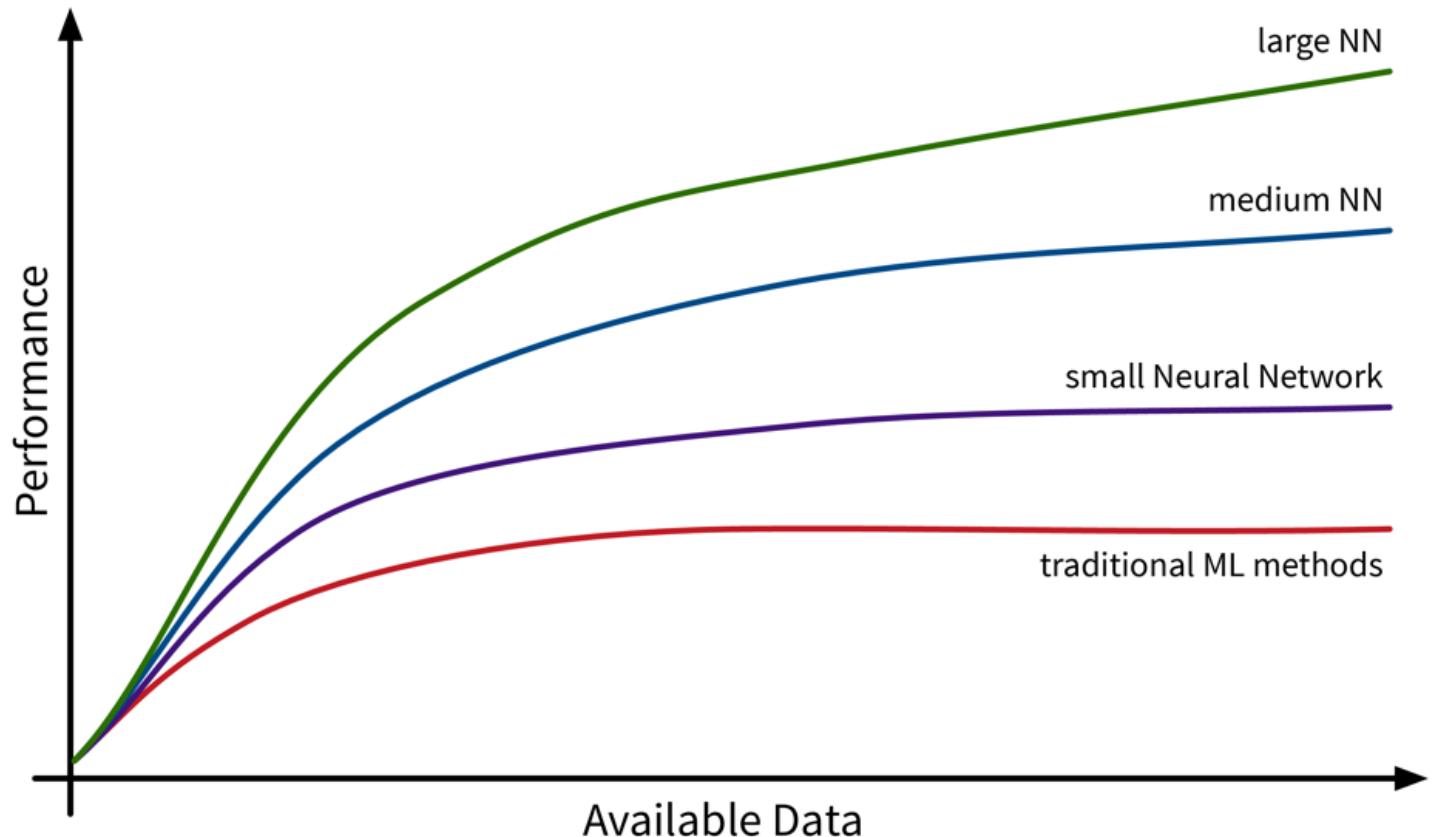
© Dr. Franziska Horn

Introduction: Why is ML so popular? <Advanced Computational Resources>



© Dr. Franziska Horn

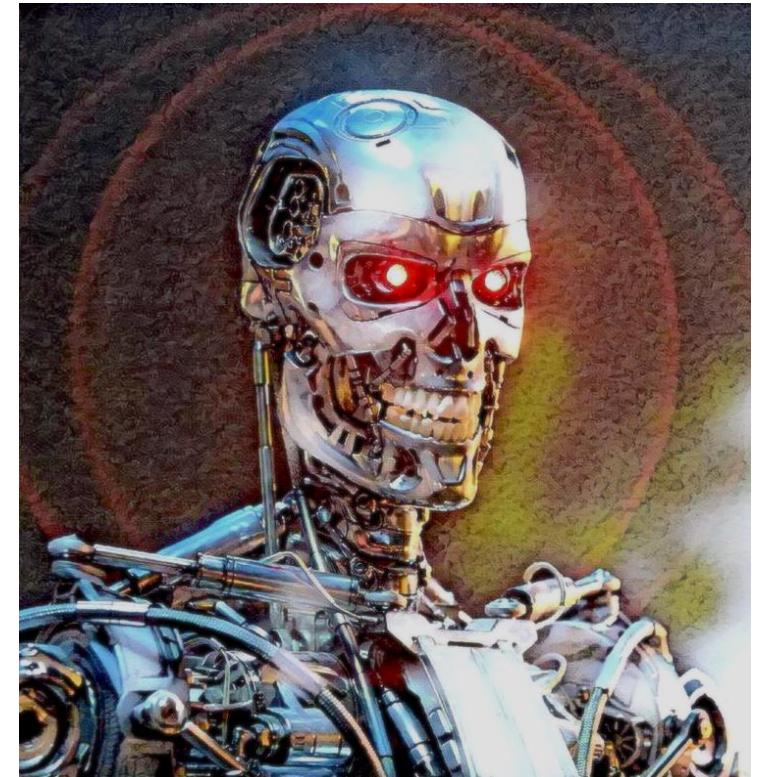
Introduction: Why is ML so popular? <Available Data vs. Performance>



Introduction: Is AGI Coming?

Artificial General Intelligence (AGI) refers to a type of artificial intelligence that possesses the ability to understand, learn, and apply knowledge across a wide range of tasks, similar to the broad cognitive abilities of humans. AGI is often contrasted with Narrow AI, which is designed for specific tasks and lacks the versatility of human-like intelligence.

Wikipedia says: The timeline for AGI development remains a subject of ongoing debate among researchers and experts. Some argue that it may be possible in years or decades; others maintain it might take a century or longer; and a minority believe it may never be achieved



Source: [A Practitioner's Guide to Machine Learning](#)

Basics: Data is the New Oil!

It all started with the data, "**data is the new oil**" is a metaphor that has been used to highlight the increasing importance of data in the modern economy.

The reason why oil is considered valuable is because we have important use cases for it: powering our cars, heating our homes, and producing plastics or fertilizers. Similarly, our data is only as valuable as what we make of it. **So, what can we use data for?**

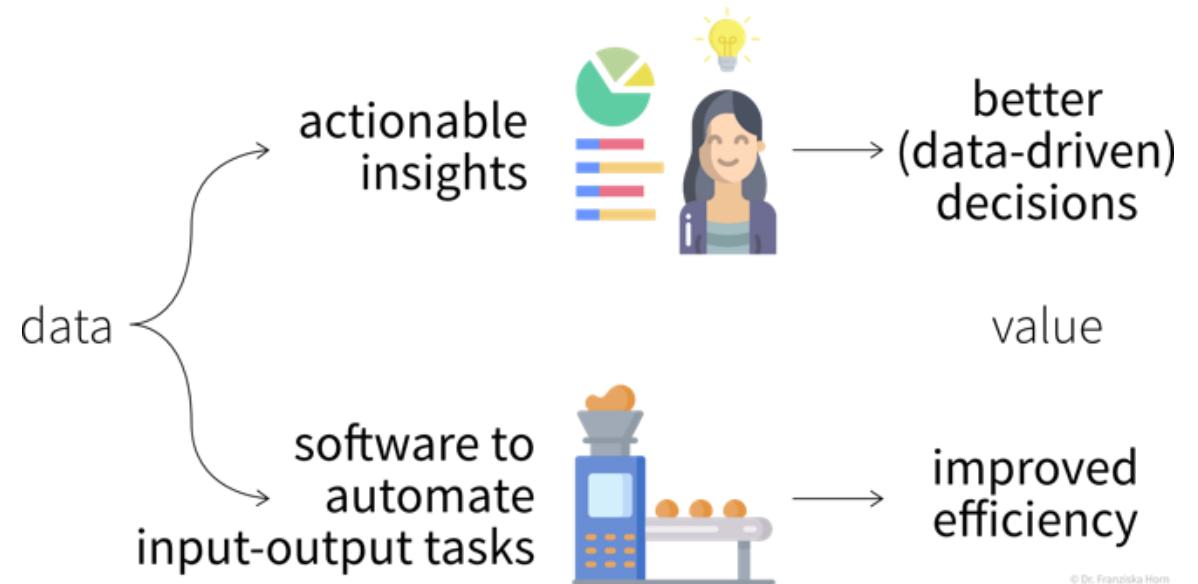


Source: economist.com

Basics: Data is the New Oil!

The main use cases belong to one of two categories:

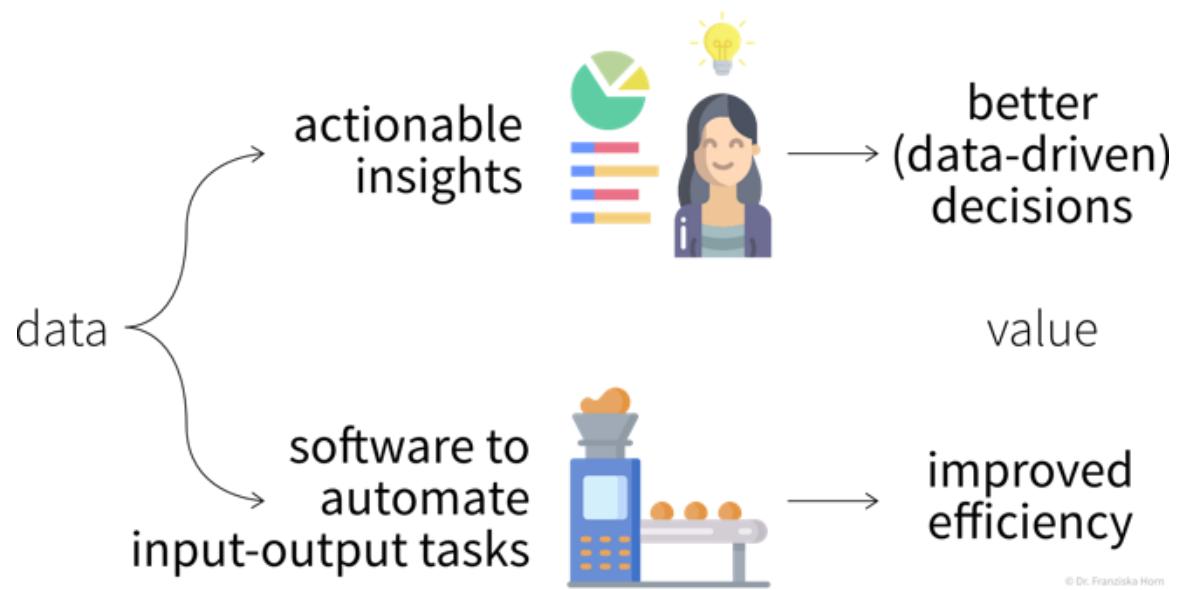
- **Insights:** We can generate insights either through continuous monitoring (“Are we on track?”) or a deeper analysis (“What’s wrong?”). By visualizing important variables or Key Performance Indicators (KPIs) in reports or dashboards, we increase the transparency of the current status and quantify our progress toward some goal.
- **Automation:** machine learning models can be used to automate ‘input → output’ tasks otherwise requiring a human (expert). These tasks are usually easy for an (appropriately trained) human, for example: Translating texts from one language into another, Sorting out products with scratches when they pass a checkpoint on the assembly line, Recommending movies to a friend. For this to work, the ML models need to be trained on a lot of historical data (e.g., texts in both languages, images of products with and without scratches, information about different users and which movies they watched).



© Dr. Franziska Horn

Basics: Data is the New Oil!

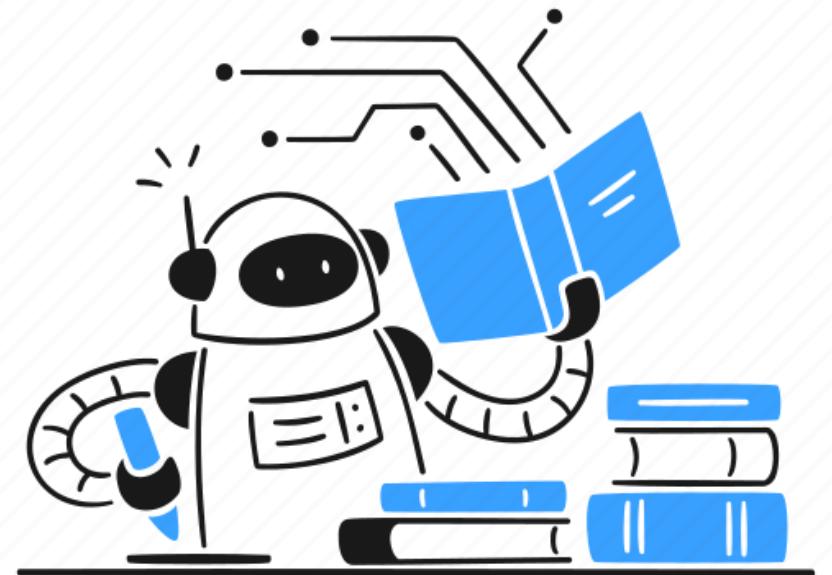
Note: These categories are not mutually exclusive, and many applications involve a combination of insights and automation. For example, data insights derived from analytics might inform the development of automation systems, and automated processes generate more data that can be analyzed for continuous improvement.



Source: [A Practitioner's Guide to Machine Learning](#)

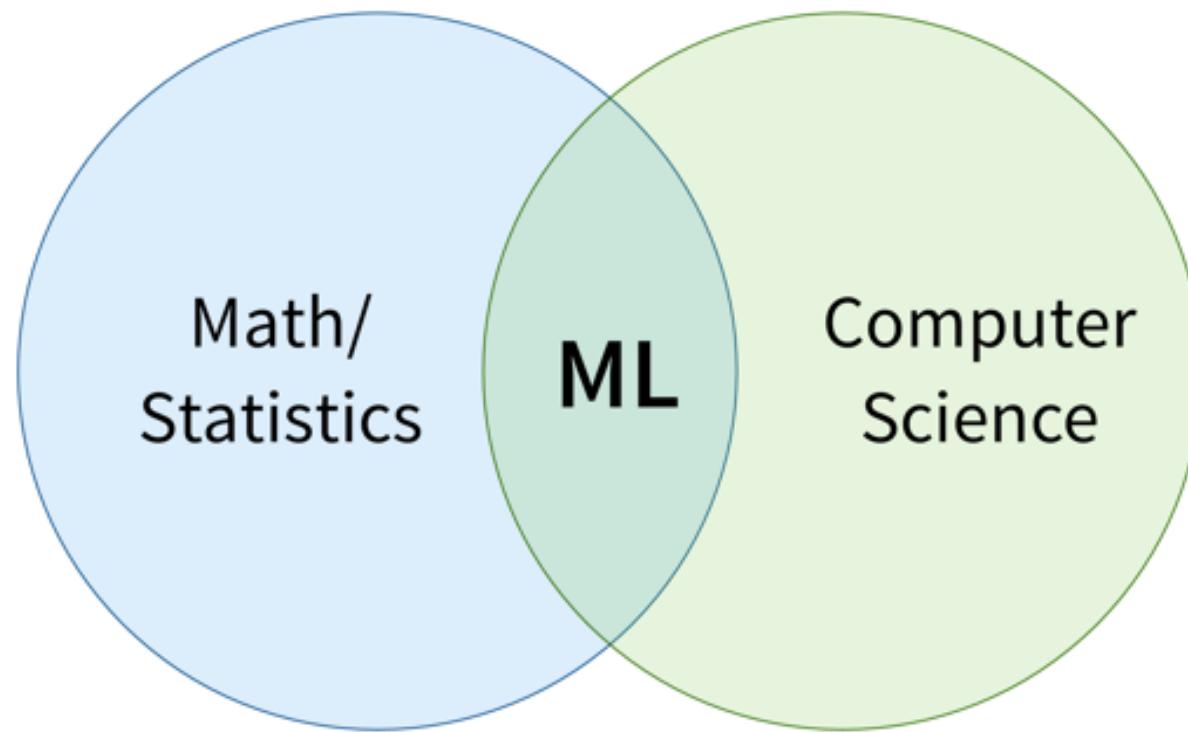
Basics: What is Machine Learning?

- **Definition 1:** Machine learning is the science of getting computers to act without being explicitly programmed
- **Definition 2:** A subset of artificial intelligence (AI) and computer science, machine learning (ML) deals with the study and use of data and algorithms that mimic how humans learn. This helps machines gradually improve their accuracy. ML allows software applications to improve their prediction accuracy without being specifically programmed to do so. It estimates new output values by using historical data as input.



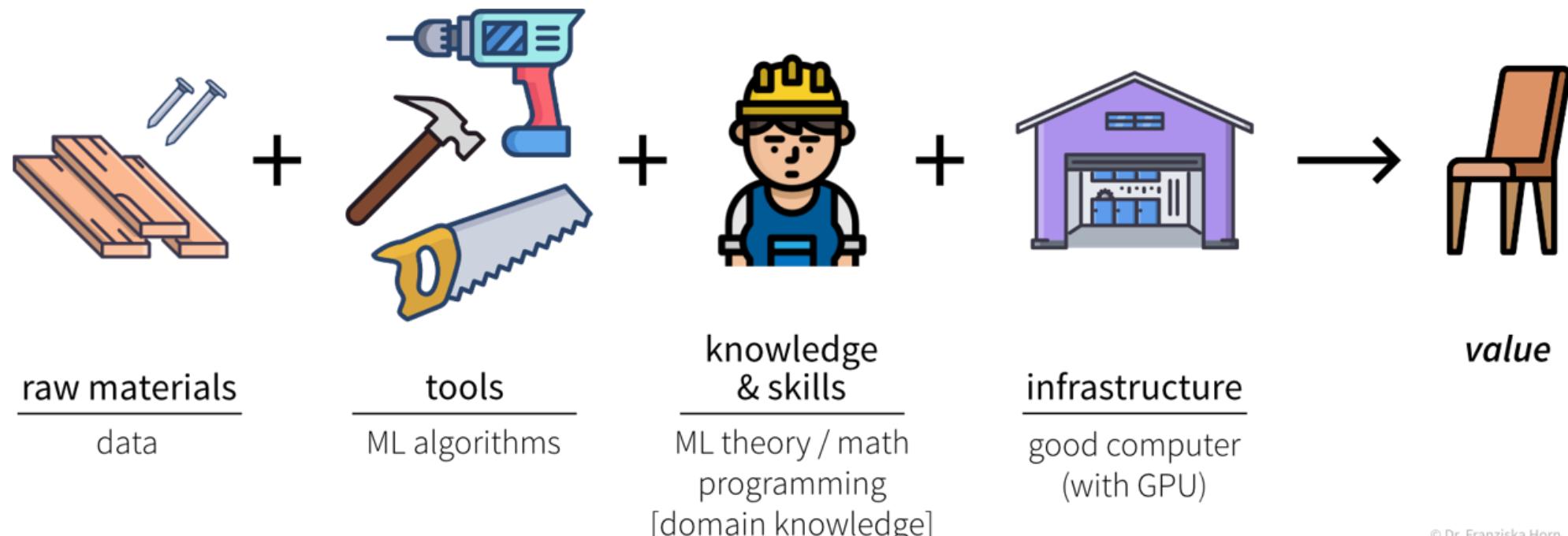
Basics: What is Machine Learning?

ML is an area of research in the field of theoretical computer science, i.e., at the intersection of mathematics and computer science



Basics: What is Machine Learning?

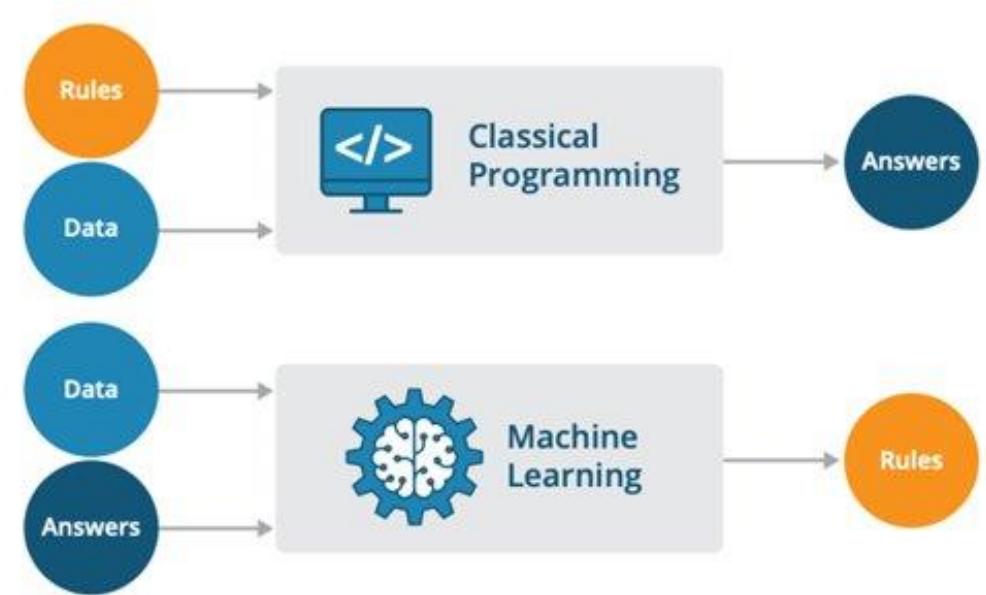
Machine learning algorithms are considered as tools to generate the value from the data



Source: [A Practitioner's Guide to Machine Learning](#)

Basics: Machine Learning vs. Traditional Software

While traditional software solutions are used to automate tasks that can be formulated as a fixed, predefined sequence of actions, executed according to some hard-coded rules (e.g., “a gate should open if an object passes through a photoelectric barrier and 20 seconds later the gate should close again”), machine learning can be used to automate “input → output” tasks for which it would otherwise be difficult to come up with such rules.



Basics: Avoid "Solution In Search of a Problem" Syndrome

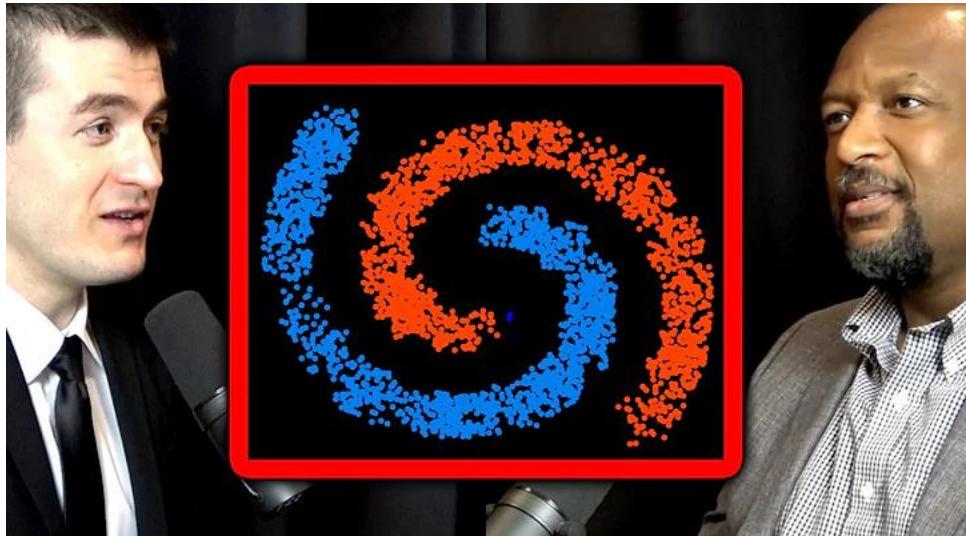
"IT WOULD BE COOL TO USE <PLACE LATEST ML TECHNOLOGY/ALGORITHM HERE"

- **Answer the question:** What problem are you solving?
 - If you can't answer this question, it is probably not worth building an ML
- **ML is overkill if:**
 - a manually defined set of rules or mechanistic (white box) model can solve the problem



Source: [reddit.com](https://www.reddit.com)

Basics: Is Machine Learning Just Statistics?



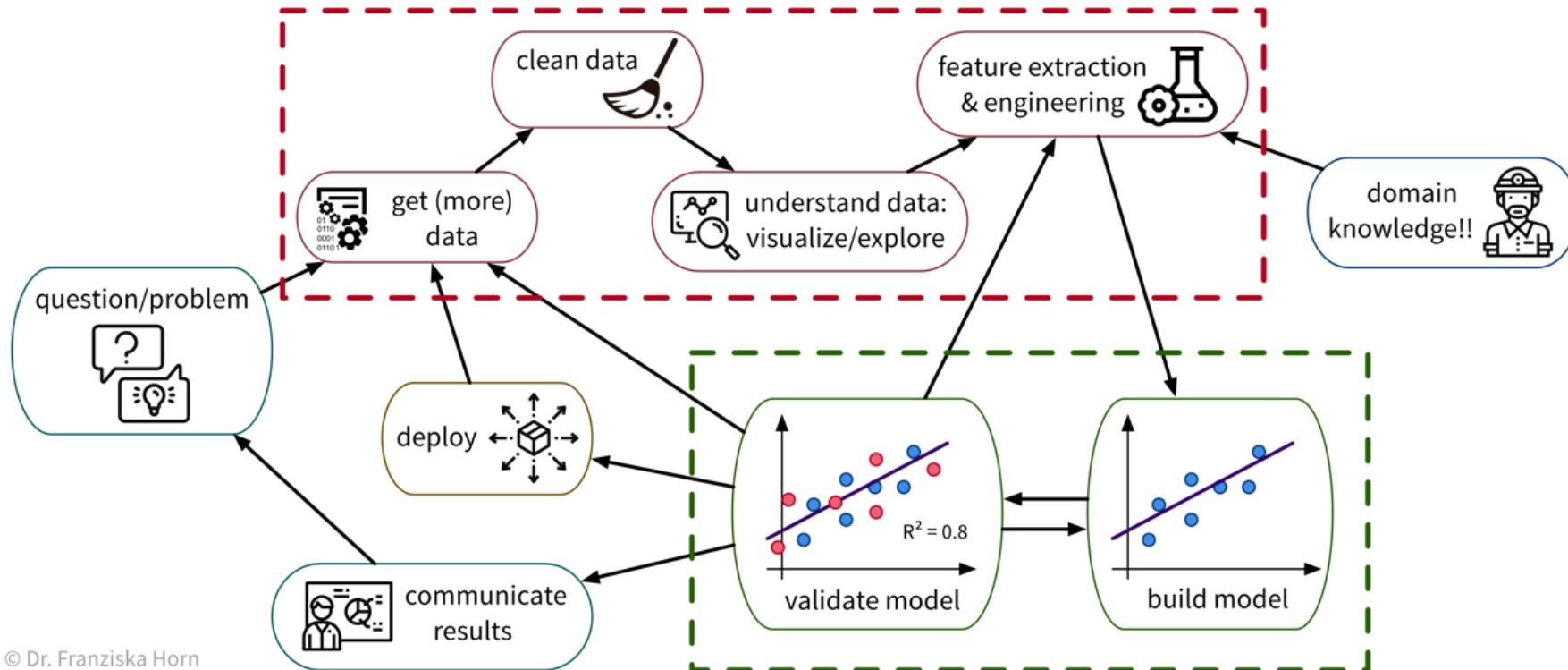
[Is machine learning just statistics? | Charles Isbell and Michael Littman and Lex Fridman](#)



[Is AI just statistics? | Yann LeCun and Lex Fridman](#)

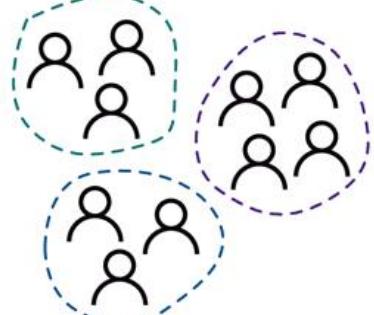
Basics: Solving Problems with ML

<Workflow>



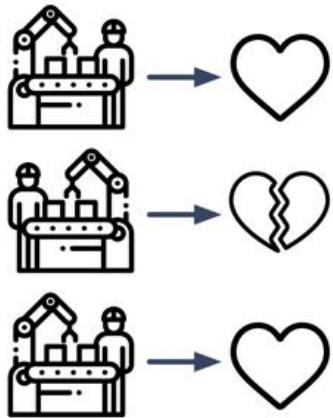
Types of Machine Learning

Unsupervised Learning



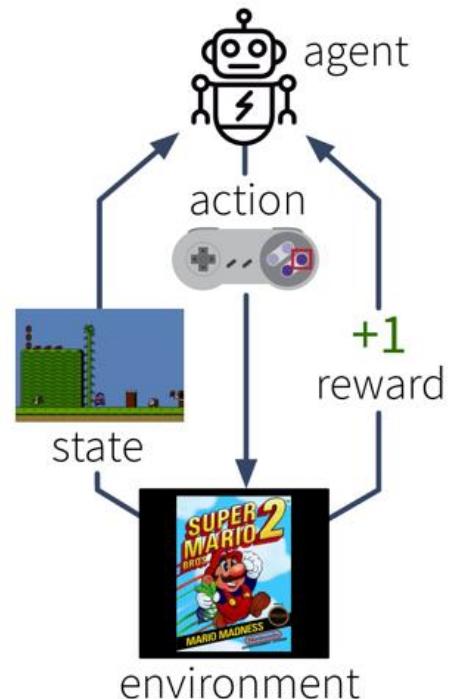
discover patterns
in the data

Supervised Learning



a model generates
a specific output
given some input

Reinforcement Learning



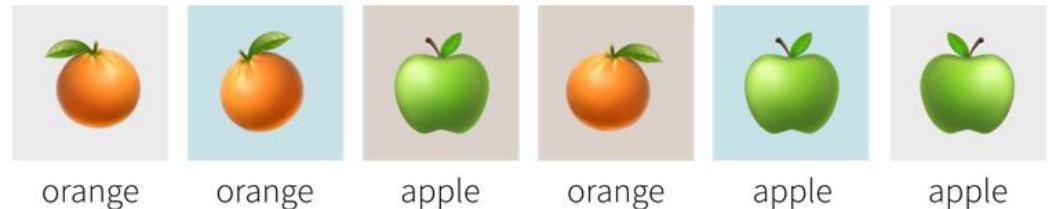
© Dr. Franziska Horn

Types of Machine Learning

Unsupervised Learning: a dataset with examples



Supervised Learning: a dataset with labeled examples



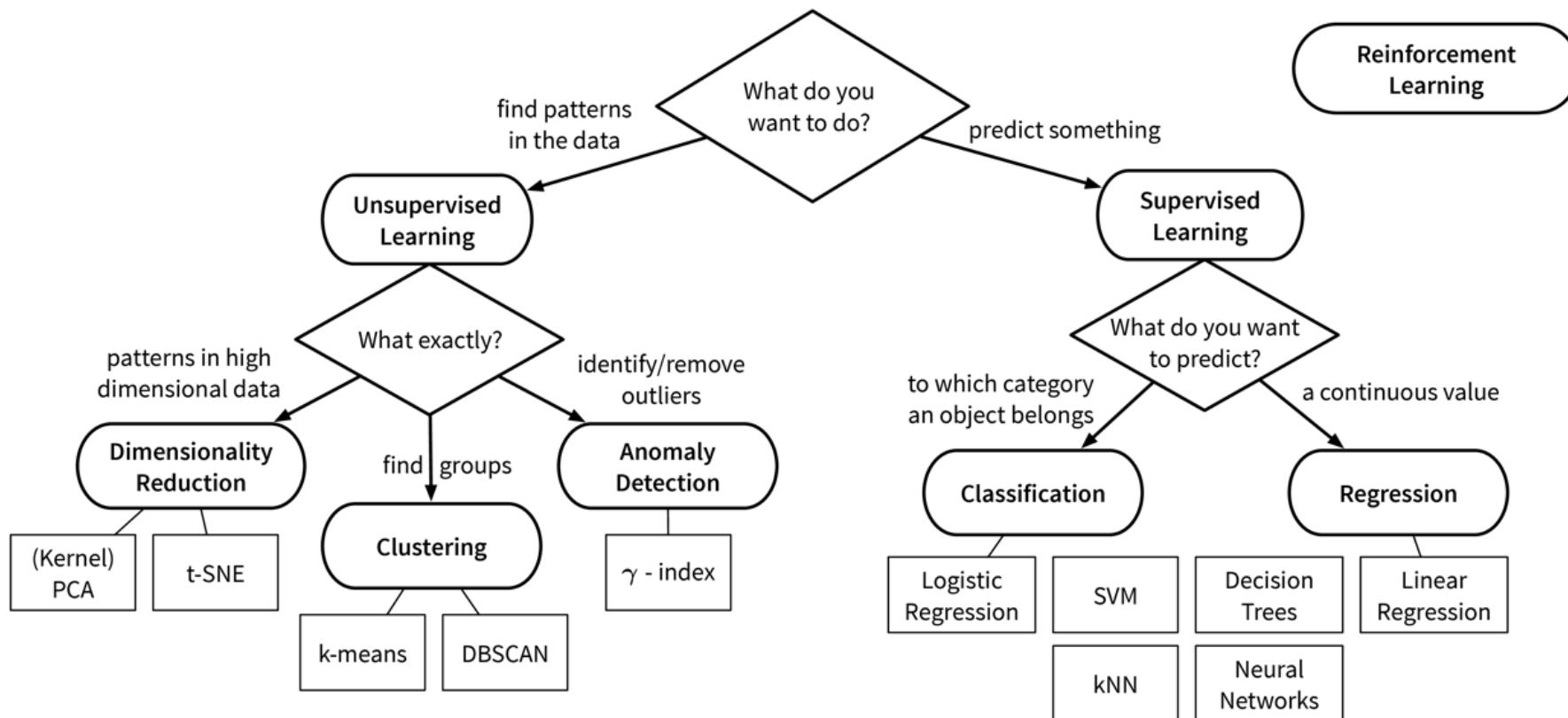
Reinforcement Learning: a (simulation) environment that generates data (i.e., reward + new state) in response to the agent's actions



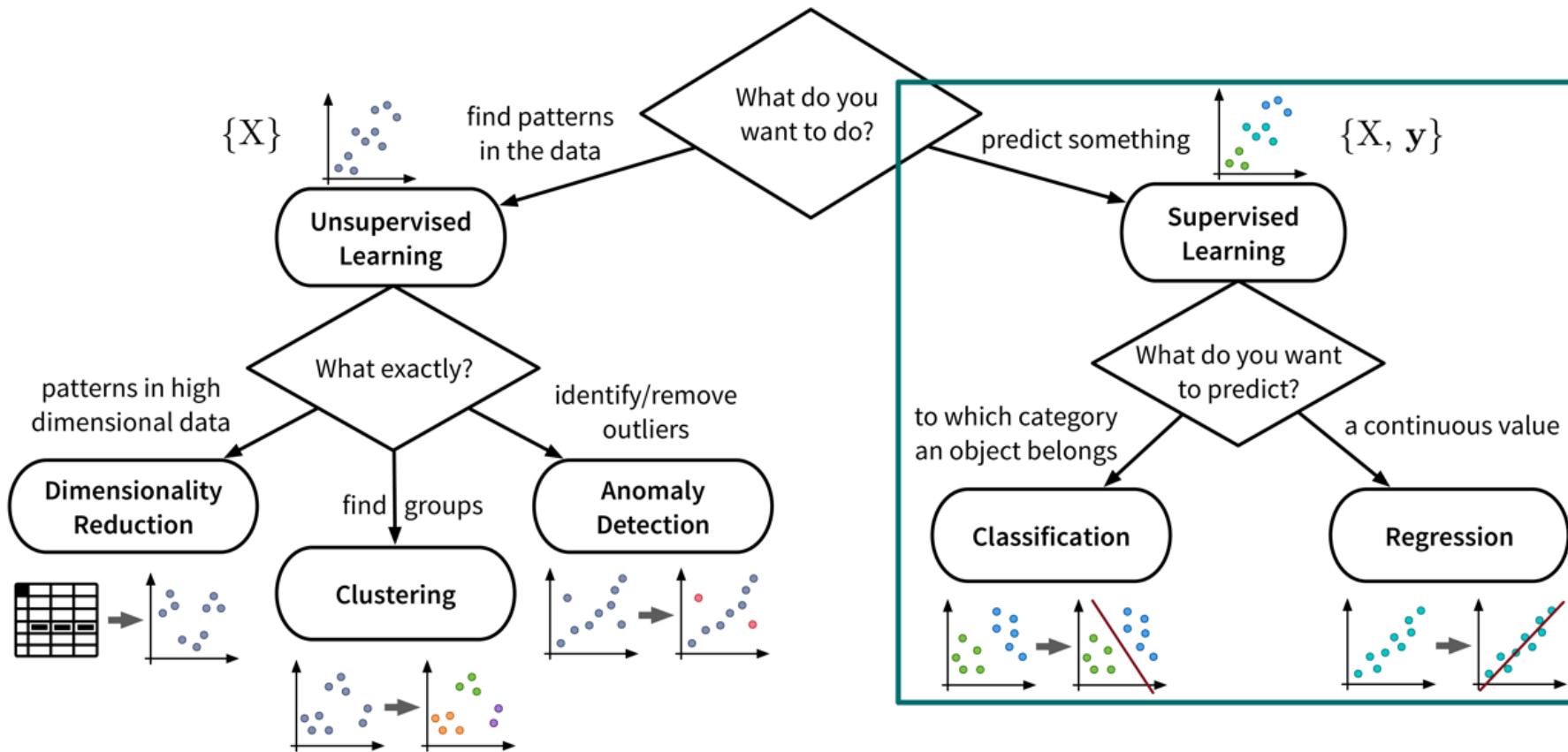
© Dr. Franziska Horn

Source: [A Practitioner's Guide to Machine Learning](#)

Types of Machine Learning



Supervised Learning Basics



© Dr. Franziska Horn

Supervised Learning Basics



Regression



What will be the temperature tomorrow?



Classification



Will it be hot or cold tomorrow?



Source: enjoyalgorithms.com

Supervised Learning Basics: Different Types of Models

The most important task of a data scientist is to select an appropriate model for solving a problem. Three considerations when choosing a supervised learning model:

1. Problem type

What kind of problem are you trying to solve: regression or classification?

→ Depends on the type of target variable, i.e., if it has continuous or discrete values.

2. Problem complexity

How complicated is the relationship between the input features and target variable: linear or nonlinear?

→ Depends on the available data, i.e., how easily the target can be predicted from the inputs.

3. Algorithmic approach

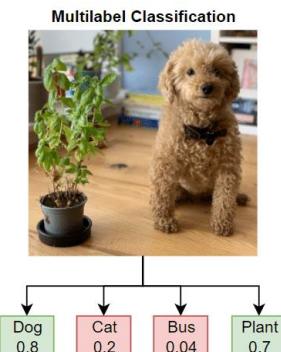
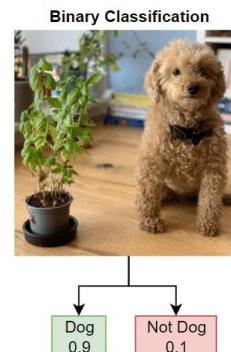
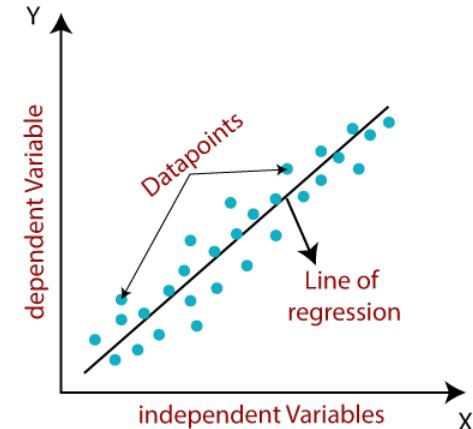
Which type of model works best for this dataset size & complexity: features-based or similarity-based?

→ Depends on the model you choose, i.e., it either learns according to the first or second strategy.

Different Types of Models: Problem Type <Regression vs. Classification>

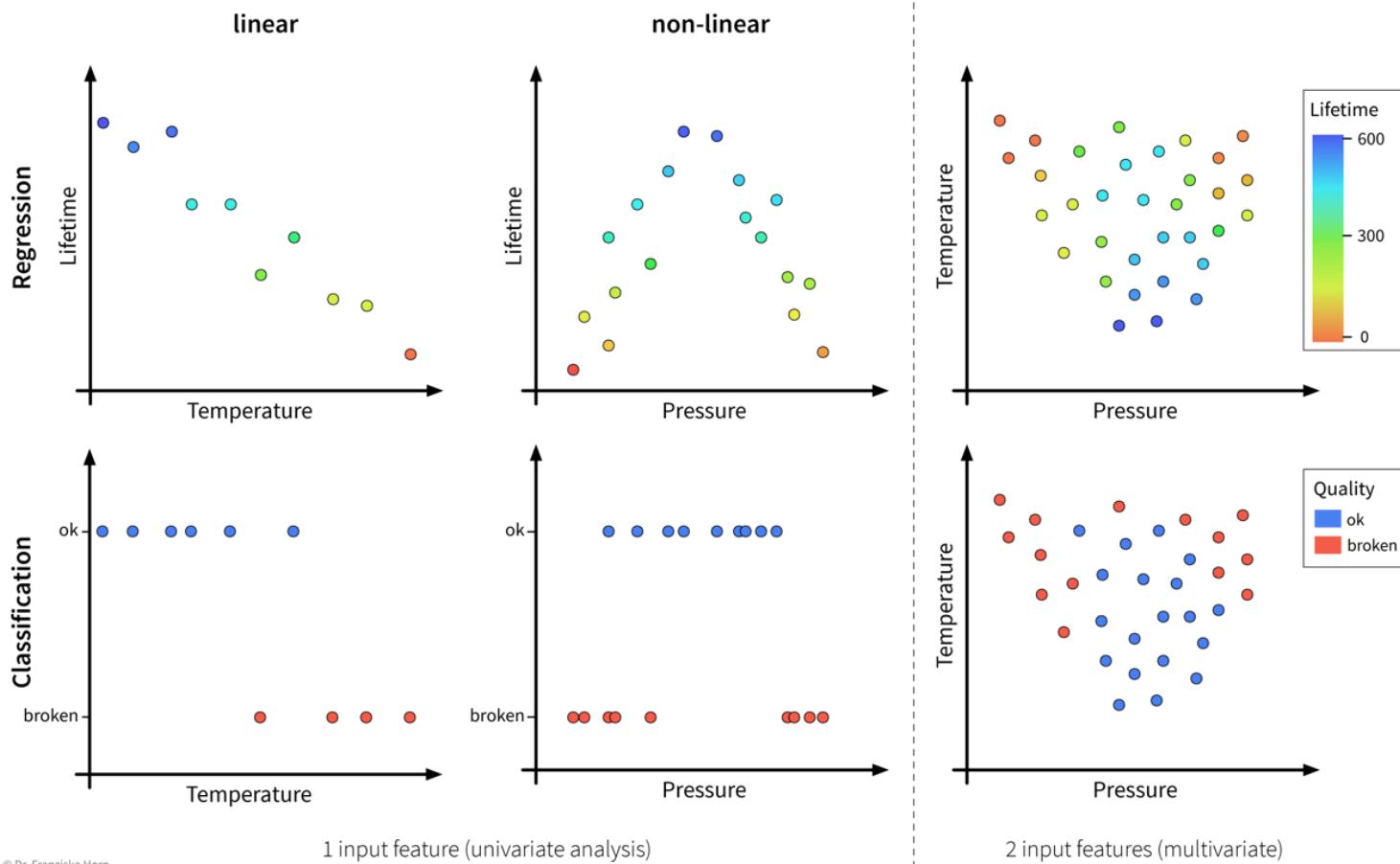
The type of the target variable that we want to predict determines whether we are dealing with a regression or classification problem.

- **Regression:** Prediction of continuous value(s) (e.g., price, number of users, etc.).
- **Classification:**
 - Prediction of discrete values:
 - binary (e.g., product will be faulty: yes/no)
 - multi-class (e.g., picture displays cat/dog/house/car/...)
 - multi-label (e.g., picture may display multiple objects)



Source: mathworks.com

Different Types of Models: Problem Complexity <Linear or nonlinear>



© Dr. Franziska Horn

Source: [A Practitioner's Guide to Machine Learning](#)

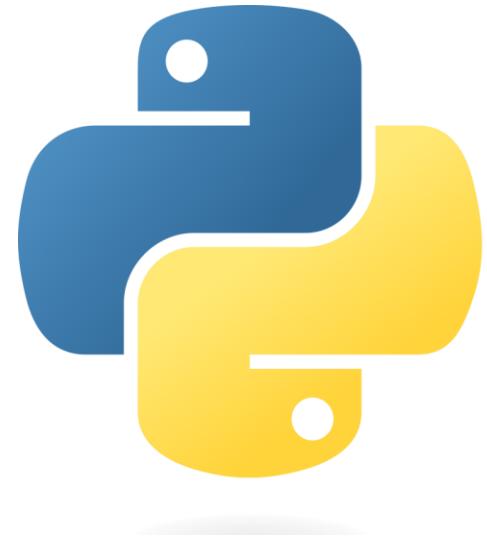
Machine Learning with Python

Why Python?

- free & open source (unlike, e.g., MatLab)
- easy; fast prototyping
- general purpose language (unlike, e.g., R): easy to incorporate ML into regular applications or web apps
- fast: many numerical operations are backed with C libraries
- a lot of open-source ML libraries with a very active community

How?

- regular scripts (i.e., normal text files ending in .py), especially useful for function definitions that can be reused in different projects
- iPython shell: interactive console to execute code
- Jupyter Notebooks (i.e., special files ending in .ipynb): great for experimenting & sharing work with others (also works with other programming languages: Jupyter stands for Julia, Python, and R; you can even mix languages in the same notebook)

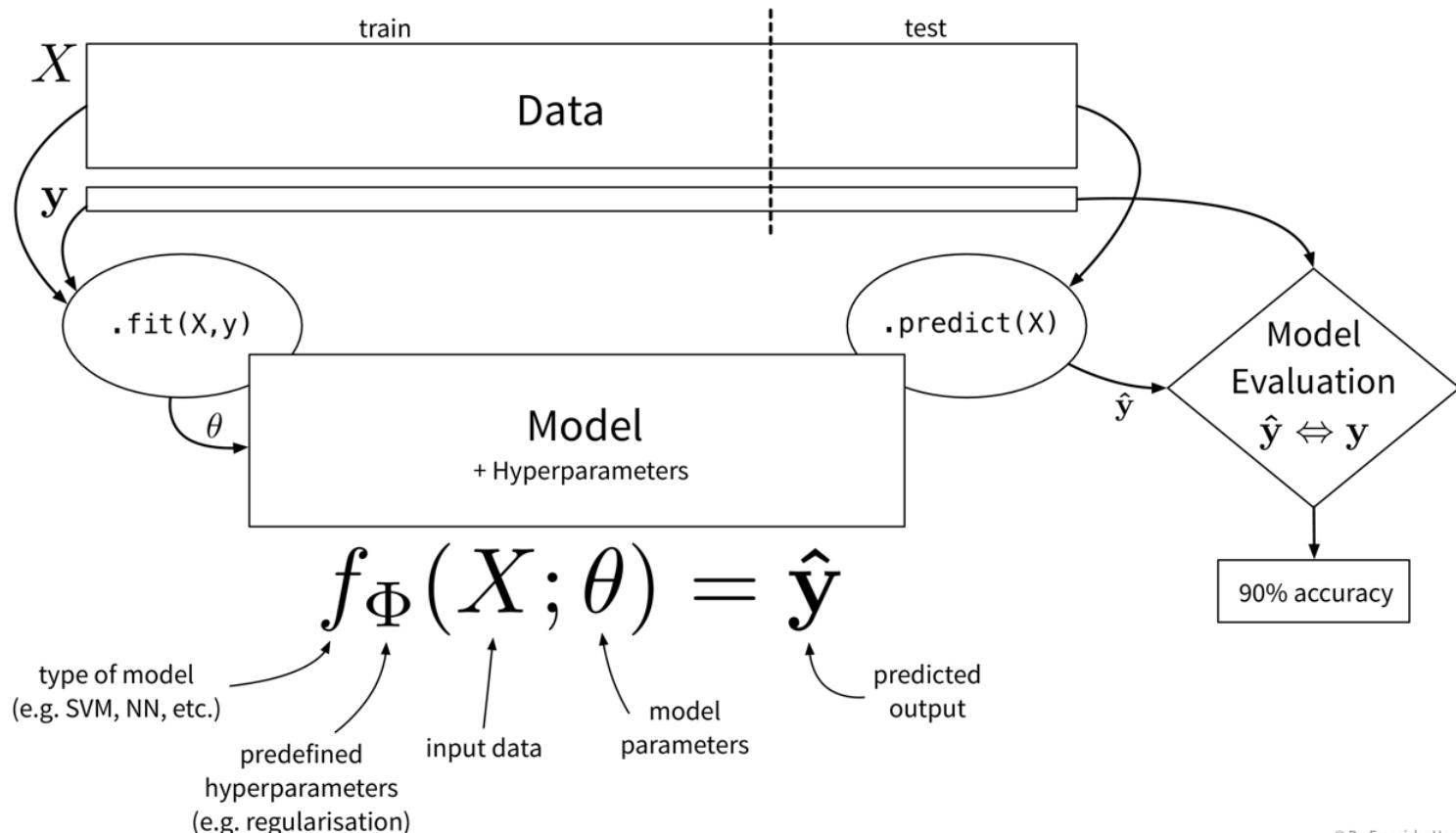


Machine Learning with Python: Scikit-learn Library

- **Scikit-learn** (also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
- It includes a lot of (**non-deep learning**) machine learning algorithms, preprocessing tools, and evaluation functions with an unified interface, i.e., all models (depending on their type) have these `.fit()`, `.transform()`, and/or `.predict()` methods, which makes it very easy to switch out models in the code by just changing the line where the model was initialized
- scikit-learn integrates well with many other Python libraries, such as Matplotlib and plotly for plotting, NumPy for array vectorization, Pandas dataframes, SciPy, and many more.



Machine Learning with Python: Scikit-learn Library



© Dr. Franziska Horn

Machine Learning with Python: Scikit-learn Library

```
# import the model class from the specific submodule
from sklearn.xxx import Model
from sklearn.metrics import accuracy_score

# initialize the model (usually we also set some parameters here)
model = Model()

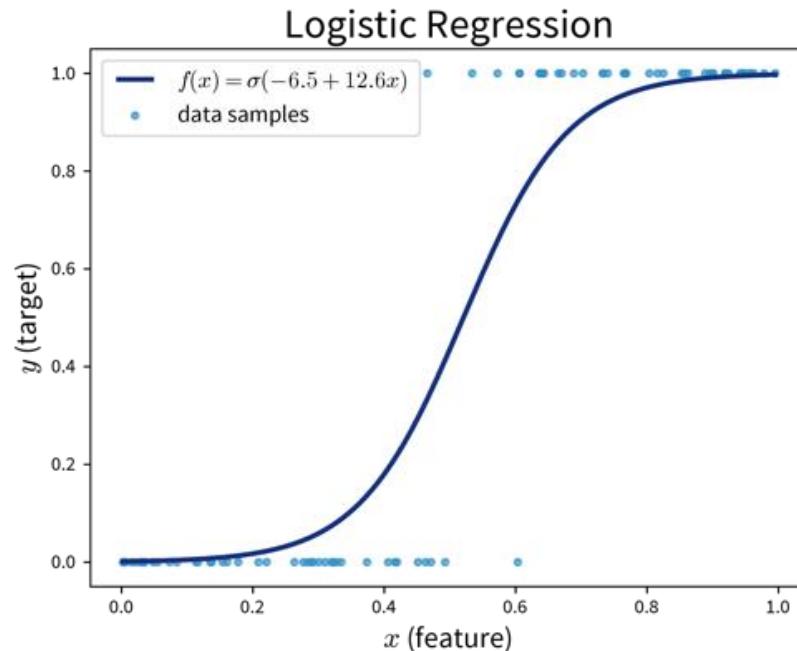
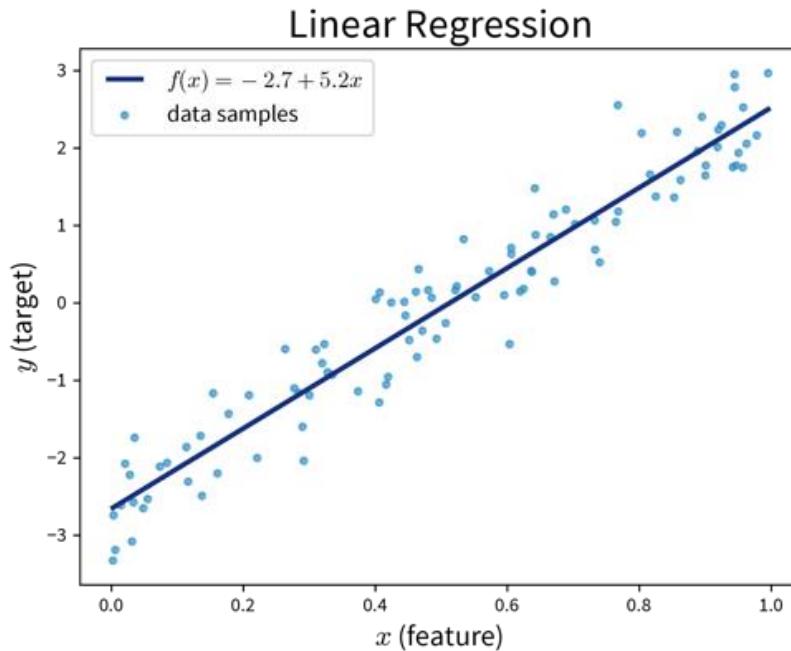
# preprocessing/unsupervised learning methods:
model.fit(X) # only pass feature matrix X
X_transformed = model.transform(X) # e.g., the StandardScaler would return a scaled feature matrix

# supervised learning methods:
model.fit(X, y) # pass features and labels for training
y_pred = model.predict(X_test) # generate predictions for new points
# evaluate the model (the internal score function uses the model's preferred evaluation metric)
print("The model is this good:", model.score(X_test, y_test)) # .score() internally calls .predict()
print("Equivalently:", accuracy_score(y_test, y_pred))
```

Supervised Learning Models

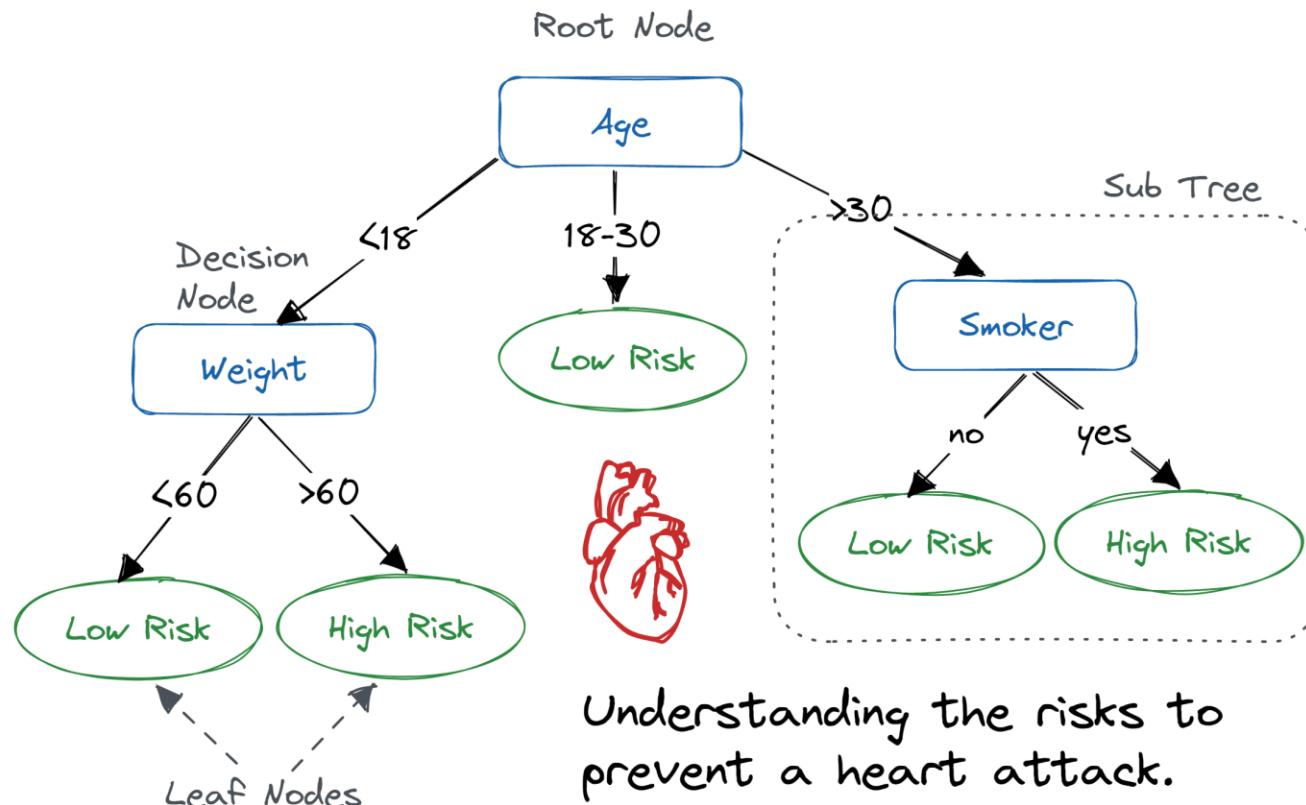
<Linear Models>

The first type of supervised learning model that we'll look at in more detail are linear models, which are a type of features-based model that are very efficient (i.e., can be used with large datasets), but, as the name suggests, are only capable of describing linear relationships between the input and target variables.



Supervised Learning Models <Decision Trees>

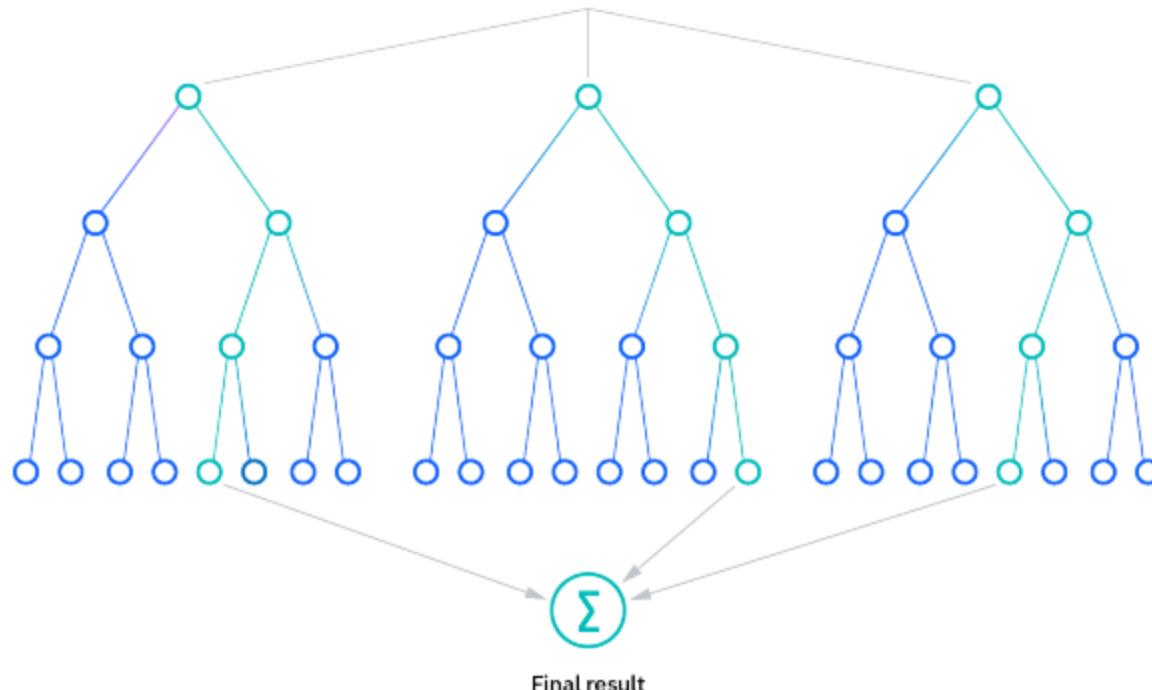
Creating a tree-like model to make decisions based on input features.



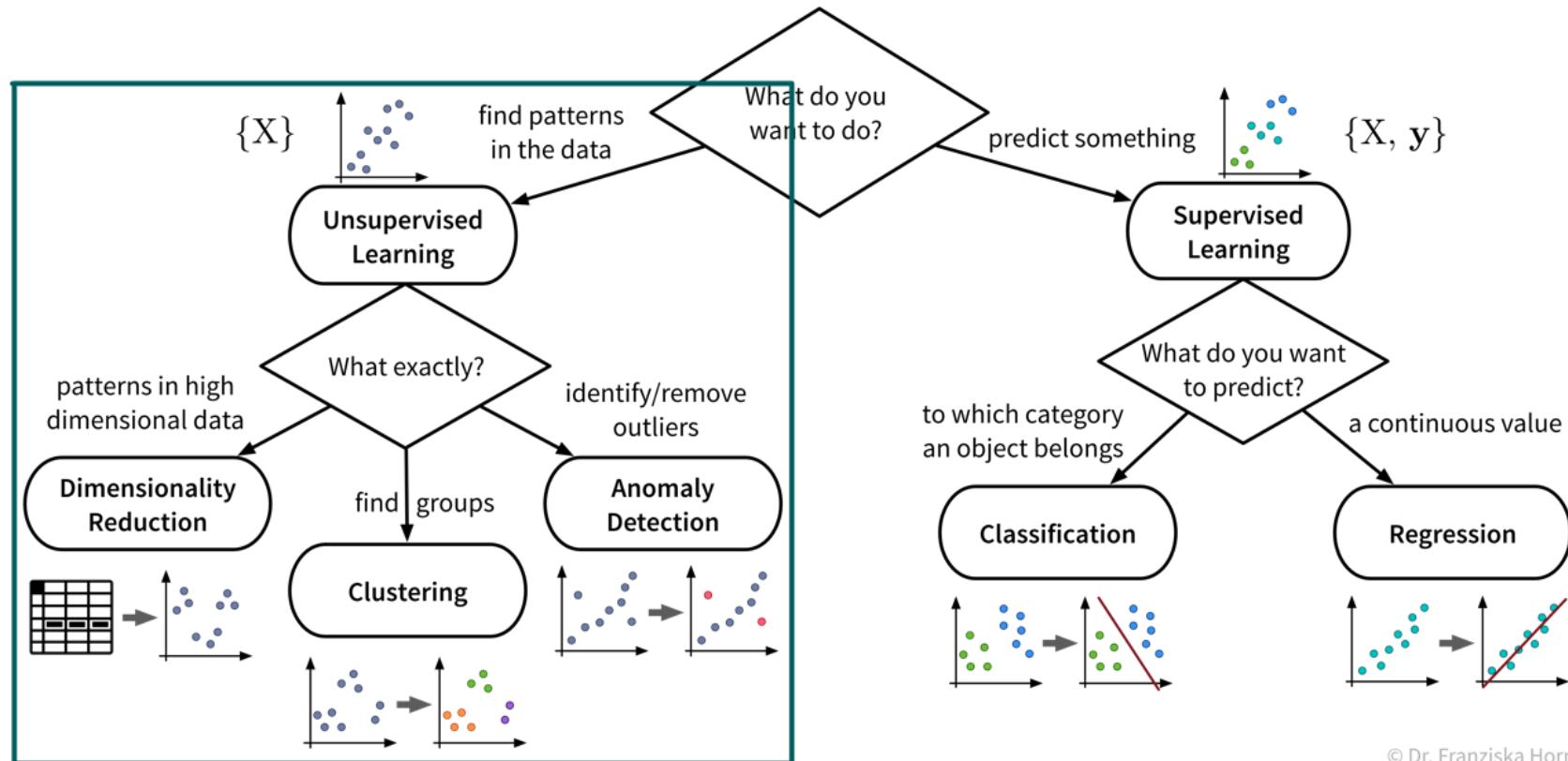
Supervised Learning Models <Ensemble Models: Random Forest>

What is better than one model? Multiple models!

Train multiple models & combine their predictions (regression: average; classification: most frequent class).



Unsupervised Learning



© Dr. Franziska Horn

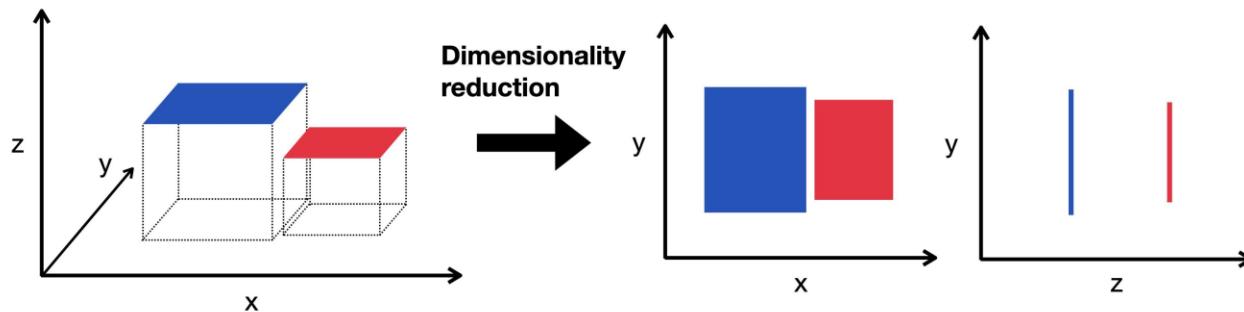
Source: [A Practitioner's Guide to Machine Learning](#)

Unsupervised Learning: Dimensionality Reduction

Goal: Reduce the number of features without losing relevant information.

Advantages:

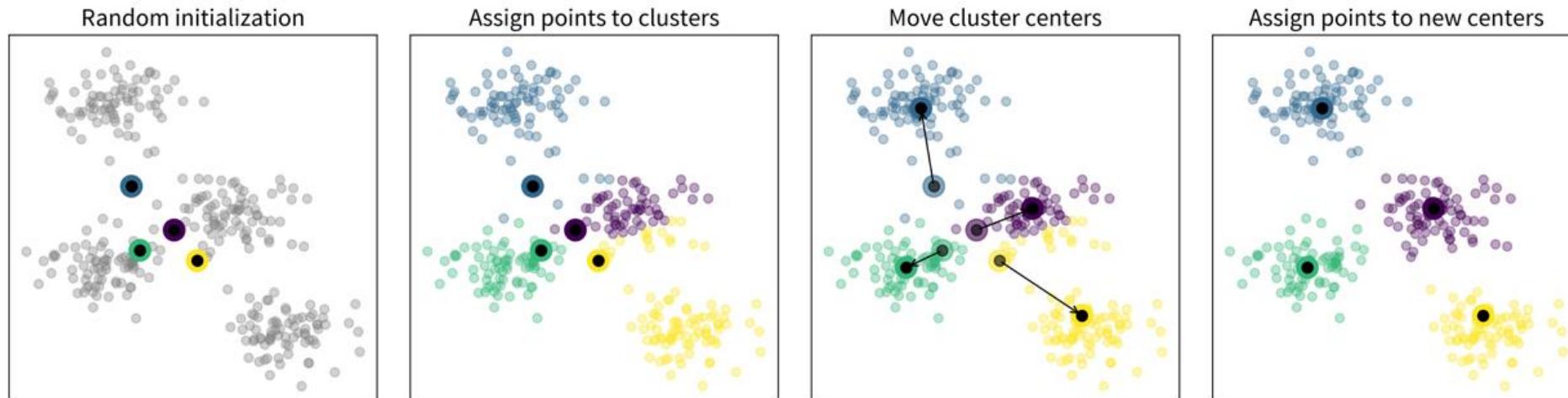
- Reduced data needs less memory (usually not that important anymore today)
- Noise reduction (by focusing on the most relevant signals)
- Create a visualization of the dataset (what we are mostly using these algorithms for)



Unsupervised Learning: Clustering

Clustering is a type of unsupervised learning technique in machine learning where the goal is to group similar data points together based on certain characteristics or features. Unlike supervised learning, where the algorithm is trained on labeled data with predefined classes, clustering algorithms work on unlabeled data.

The main objective of clustering is to discover hidden patterns or structures within the data, allowing for the identification of natural groupings or clusters. Each cluster ideally contains data points that are more similar to each other than to those in other clusters.



Unsupervised Learning: Anomaly/Outlier Detection

Anomaly or outlier detection in unsupervised learning is the process of identifying instances in a dataset that deviate significantly from the norm, without relying on labeled data. In other words, the goal is to discover patterns or observations that are different from the majority of the data. Anomalies, also referred to as outliers, can represent unexpected events, errors, or rare occurrences in the dataset.



Source: [A Practitioner's Guide to Machine Learning](#)

Unsupervised Learning: Anomaly/Outlier Detection

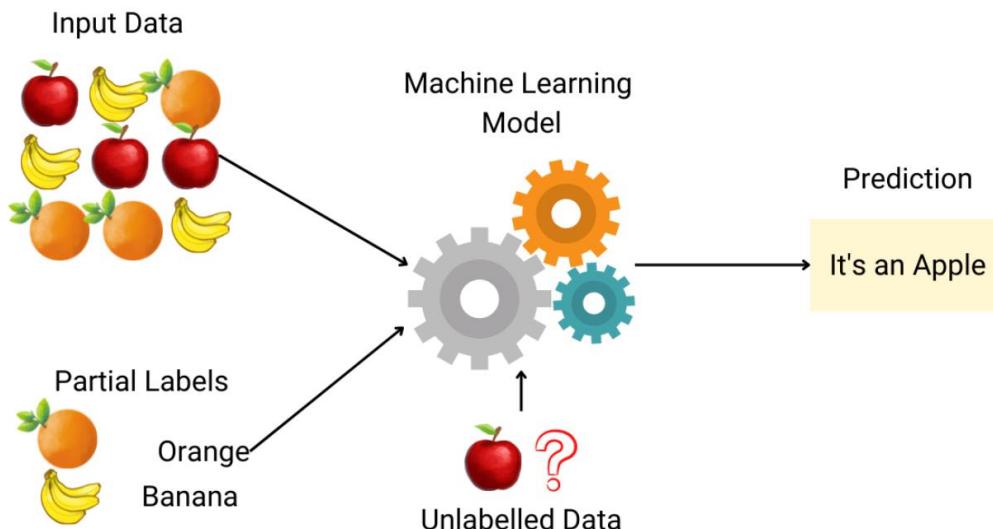
Different methods for outlier/anomaly detection:

There is no one-size-fits-all solution, but depending on the characteristics of the dataset you might want to experiment with some of these approaches:

- **Detect outliers in individual feature dimensions:**
 - Compute the interquartile range (IQR), i.e., the difference between the 75th and 25th percentiles of the data, as $IQR=Q3-Q1$, where Q1 and Q3 are the lower and upper quartiles. All data points with values outside of the range $[Q1-1.5 \cdot IQR, Q3+1.5 \cdot IQR]$ are considered outliers.
- **Detect multidimensional outlier patterns**, especially in datasets with independent data points:
 - γ -index (Gamma Index)
 - Local Outlier Factor
 - Isolation Forest
 - One-class SVM
 - DBSCAN (a clustering algorithm that allows for noise, which would be considered outliers)

Other Types of Machine Learning: Semi-Supervised Learning

Semi-supervised learning is a branch of machine learning that combines supervised and unsupervised learning, using both labeled and unlabeled data to train artificial intelligence (AI) models for classification and regression tasks.



Other Types of Machine Learning: Semi-Supervised Learning

Definition:

Uses a combination of labeled and unlabeled data for training.

Motivation:

Labeled data is often expensive and time-consuming to acquire, while unlabeled data is more abundant.

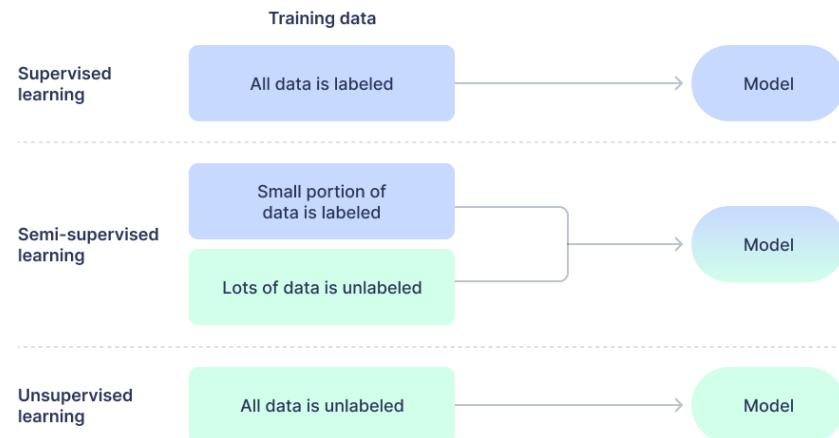
Approaches:

- **Self-training:** Uses a model to make predictions on unlabeled data and adds confidently predicted instances to the labeled set.
- **Co-training:** Trains multiple models on different views of the data, each providing information on the unlabeled instances.

Applications:

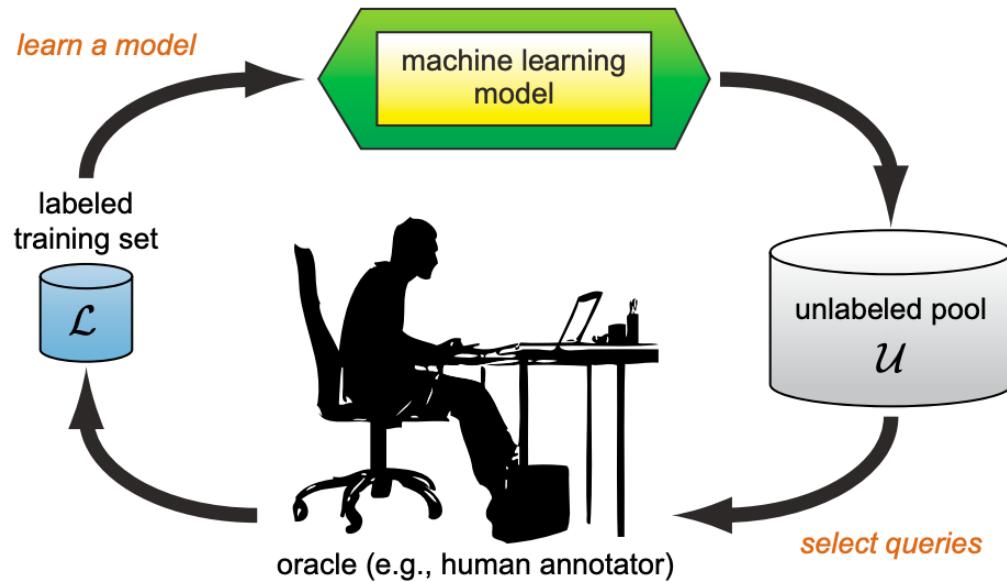
Often used in scenarios where acquiring labeled data is challenging, such as in medical imaging or natural language processing.

Supervised learning vs Semi-supervised learning vs Unsupervised learning



Other Types of Machine Learning: Active Learning

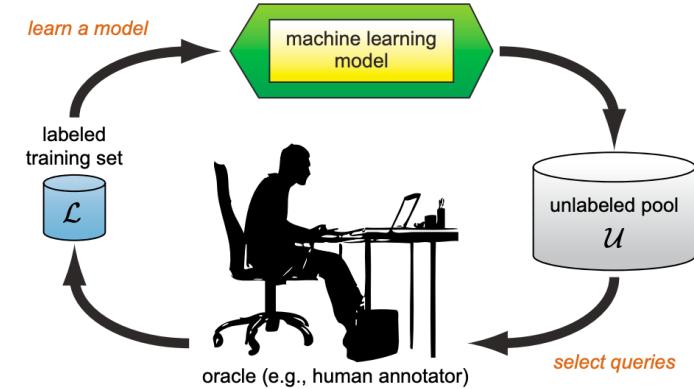
Active learning is a machine learning paradigm that involves an iterative process of selecting the most informative or uncertain data points for labeling and adding them to the training set. The goal is to improve the performance of a model with minimal labeled data. Instead of passively relying on a fixed dataset, active learning enables the model to actively query the user or a labeling oracle for the labels of the most valuable instances.



Other Types of Machine Learning: Active Learning

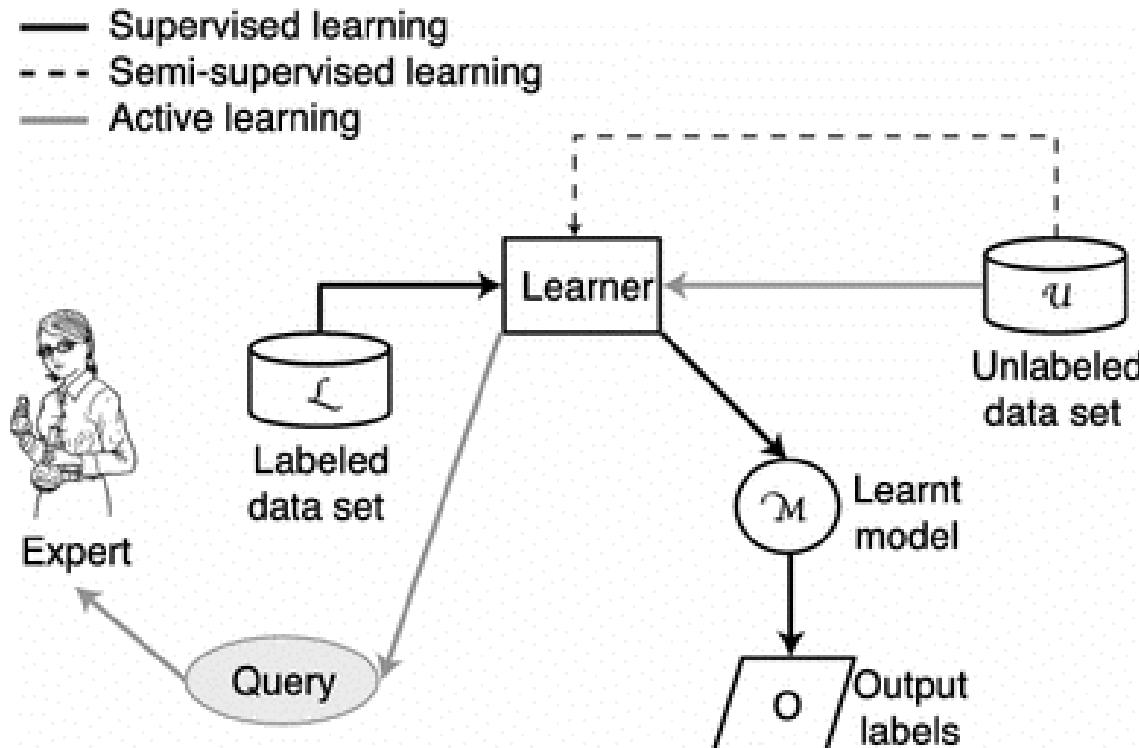
The key steps in an active learning process are as follows:

- **Initial Model Training:** Train an initial model using a small set of labeled examples.
- **Querying:** Identify the instances in the unlabeled dataset that are expected to provide the most valuable information for improving the model. These instances are typically chosen based on some measure of uncertainty or informativeness.
- **Labeling:** Request labels for the selected instances either from human annotators or some other means.
- **Updating the Model:** Incorporate the newly labeled instances into the training set and retrain the model.
- **Iteration:** Repeat the process by going back to step 2 until the model achieves satisfactory performance or resources are exhausted.



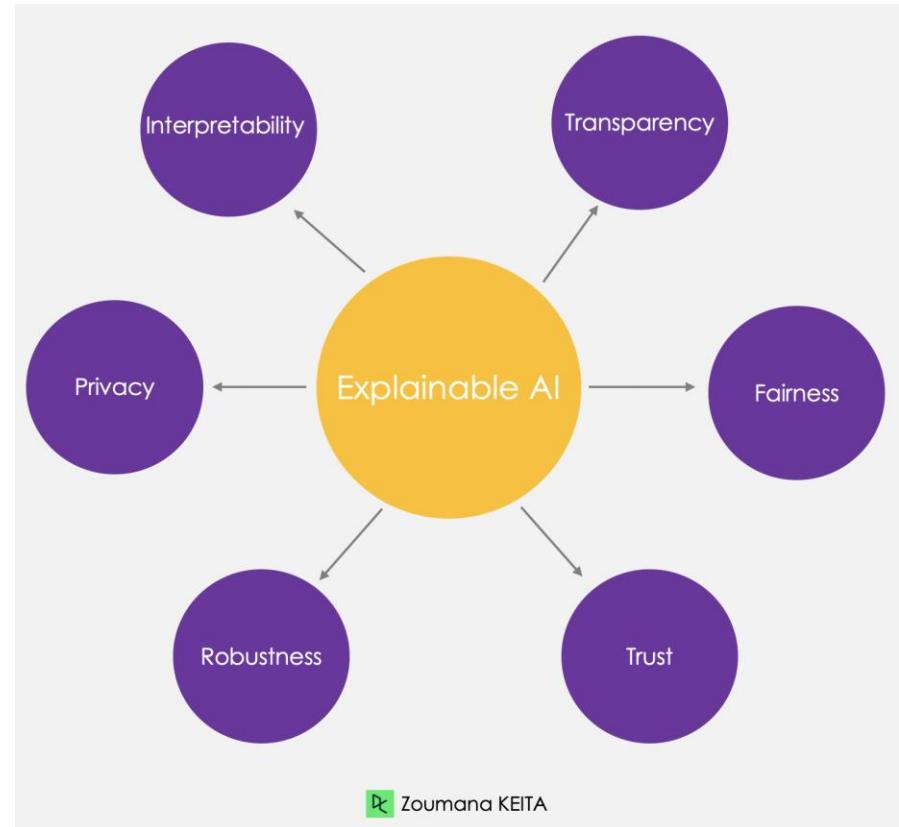
Semi-Supervised and Active Learning <DEMO>

Demo: [Session 3 – Machine Learning Fundamentals](#)



eXplainable Artificial Intelligence (XAI)

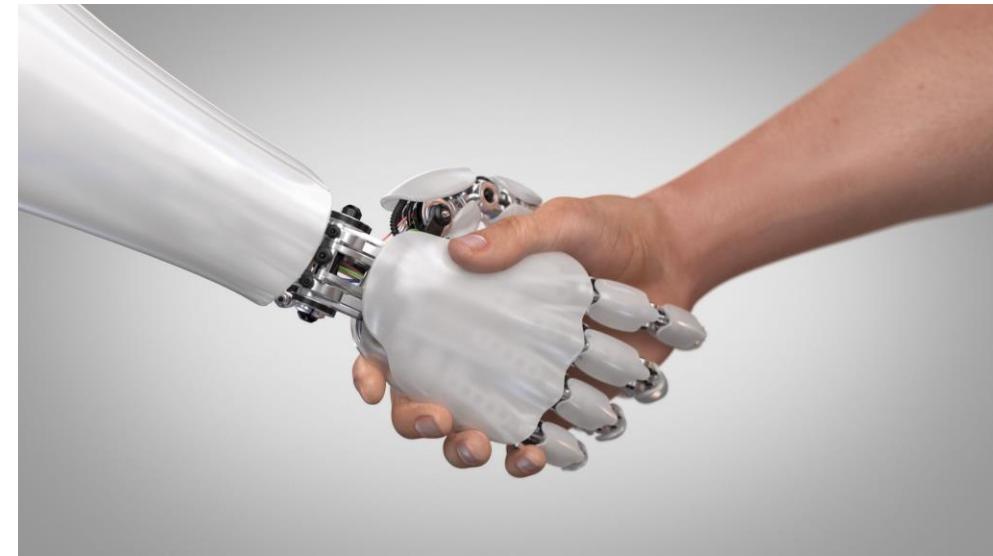
Explainable AI refers to a set of processes and methods that aim to provide a clear and human-understandable explanation for the decisions generated by AI and machine learning models. Integrating an explainability layer into these models, Data Scientists and Machine Learning practitioners can create more trustworthy and transparent systems to assist a wide range of stakeholders such as developers, regulators, and end-users.



Goal of XAI: The goal of XAI is to make AI systems more transparent and interpretable, enabling users to comprehend the rationale behind the AI's output.

Why eXplainable AI (XAI)?

1. **Trust and Accountability:** In many applications, especially those with significant consequences such as healthcare, finance, and legal systems, users need to trust AI systems. Understanding how and why a model reaches a particular decision fosters trust and accountability.
2. **Legal and Ethical Compliance:** Regulations and ethical guidelines often require organizations to provide explanations for automated decisions. This is particularly crucial in situations where AI impacts individuals' rights and well-being.
3. **Bias and Fairness:** Explainability helps identify and address biases in AI models. It allows stakeholders to evaluate whether the model is making fair and unbiased decisions, and if not, take corrective measures.
4. **Human Understanding:** Explainable AI is essential for users who may not be experts in machine learning. It enables them to understand the decision-making process and ensures collaboration between AI systems and human users.



Why should I trust you?

Techniques To Achieve explainability in AI

Interpretable Models: Choosing models that inherently provide transparency, such as decision trees or linear regression, can make it easier to understand how inputs contribute to outputs.

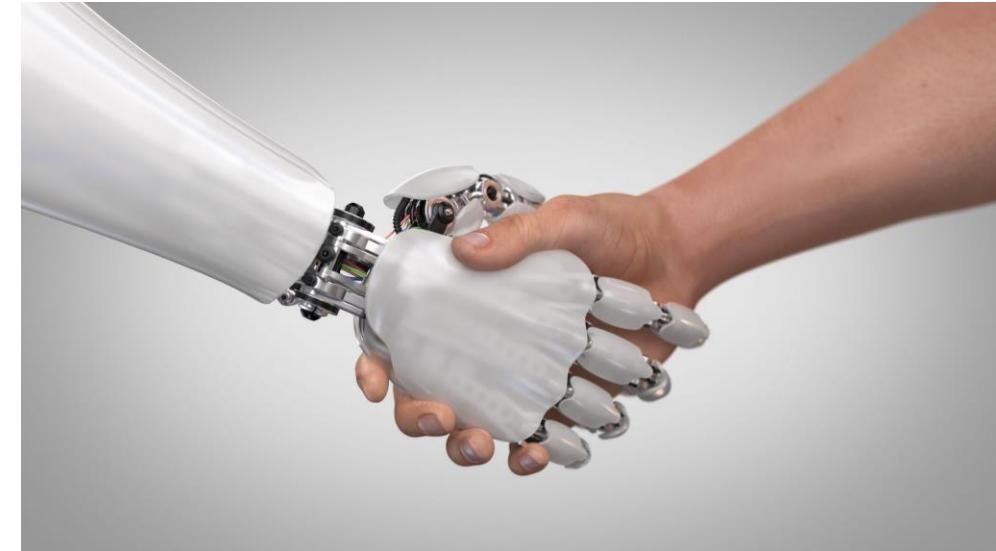
Feature Importance: Analyzing the importance of different features in the model's decision-making process helps users understand which inputs have the most significant impact.

Local Explanations: Providing explanations for individual predictions rather than the entire model output allows users to understand specific instances of decision-making.

LIME (Local Interpretable Model-agnostic Explanations): LIME is a technique that approximates the behavior of a black-box model using a simpler, interpretable model for specific instances.

SHAP (SHapley Additive exPlanations): SHAP values allocate contributions of each feature to the prediction, offering a comprehensive and theoretically grounded explanation.

Visualization: Graphical representations, such as heatmaps or saliency maps, can help users visually understand the parts of input data that influenced the model's decision.

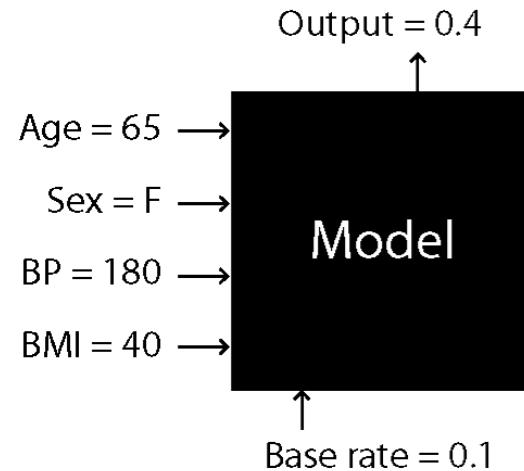


Why should I trust you?

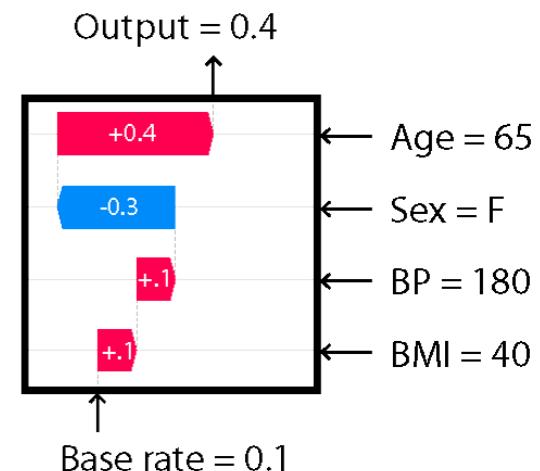
Techniques To Achieve explainability in AI <SHAP>



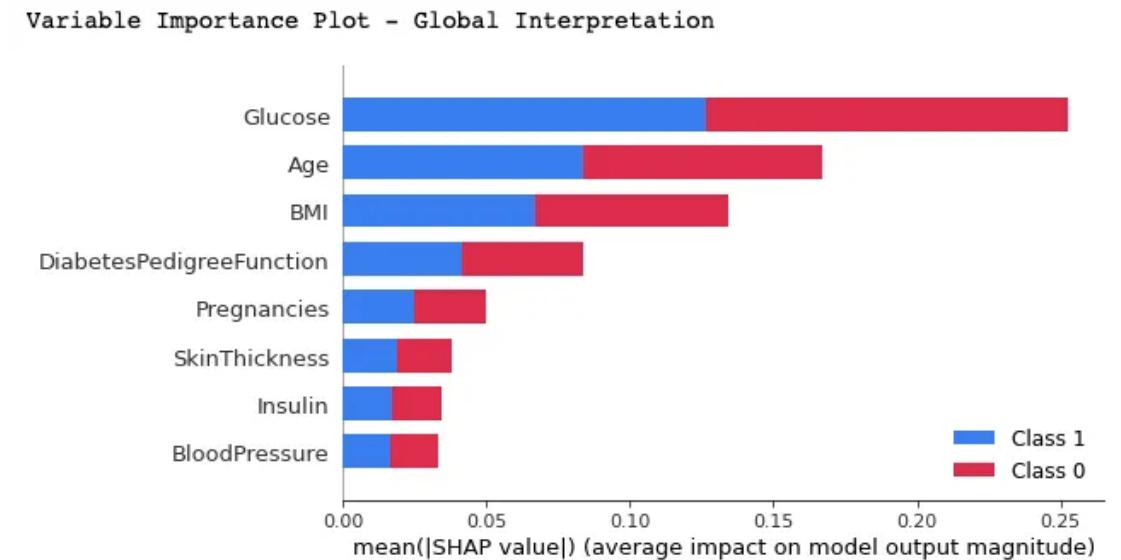
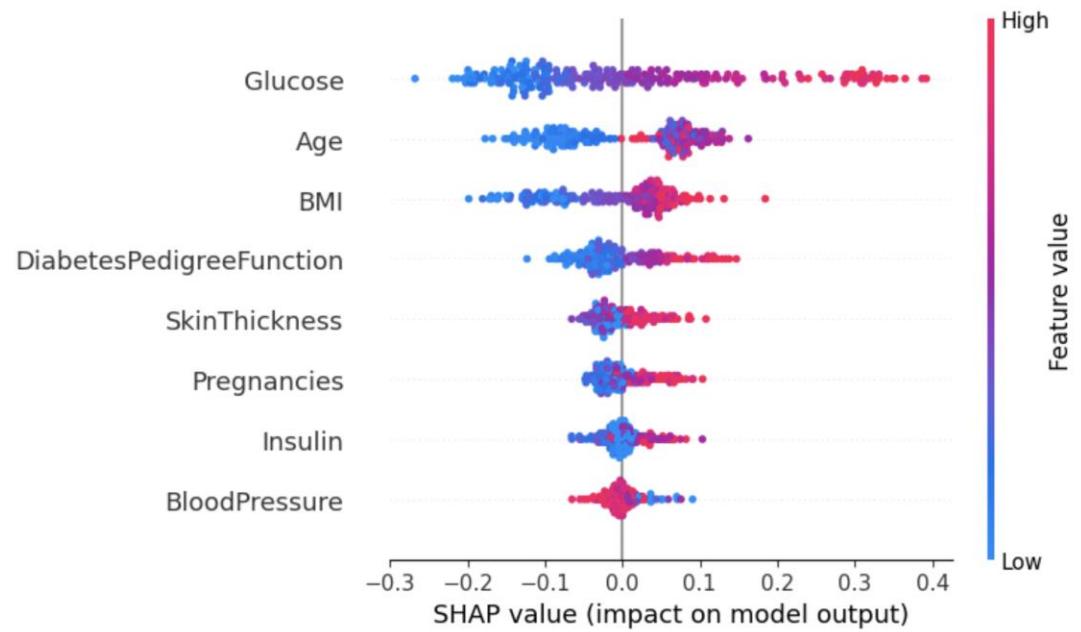
SHAP



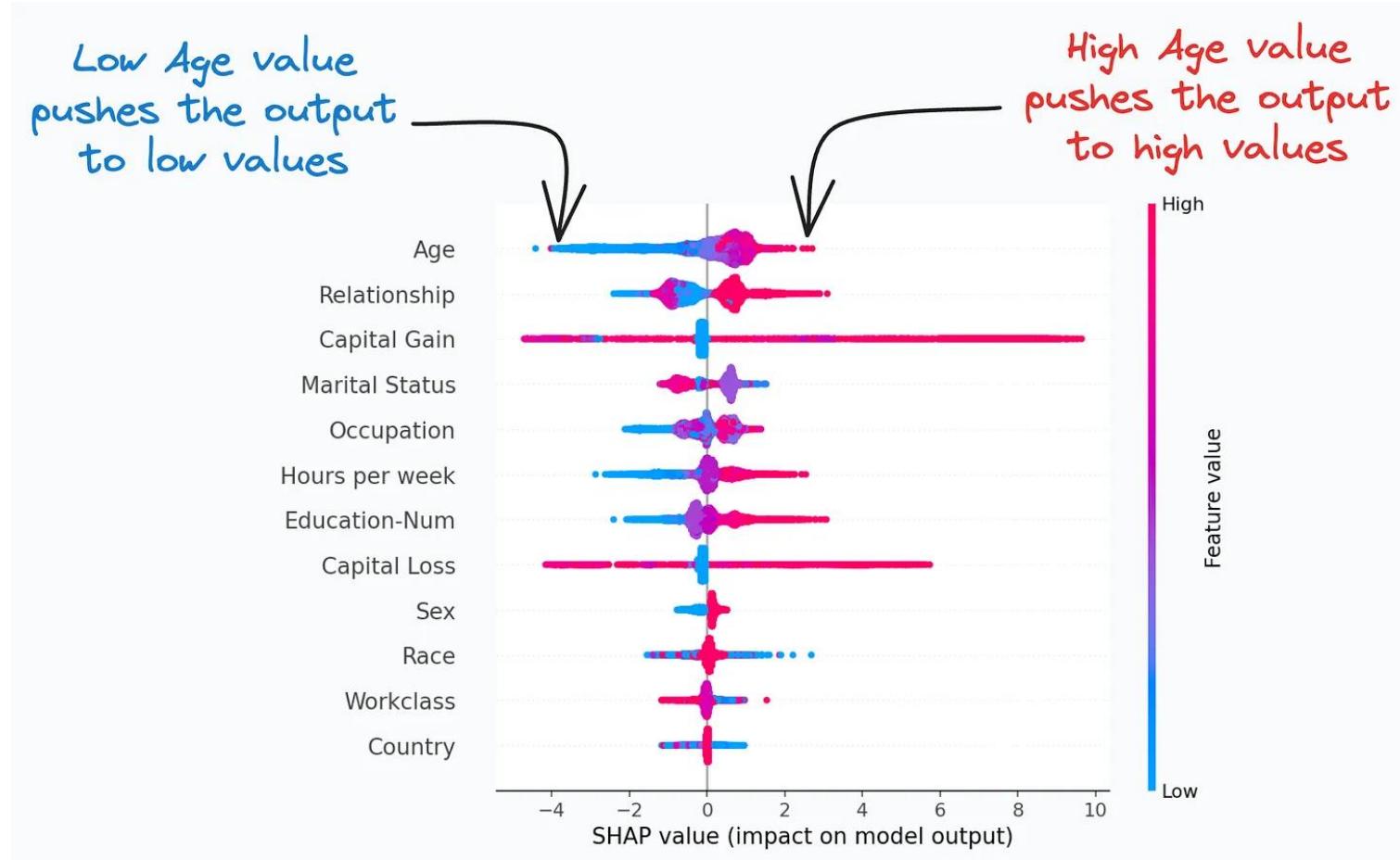
Explanation →



Techniques To Achieve explainability in AI <SHAP>

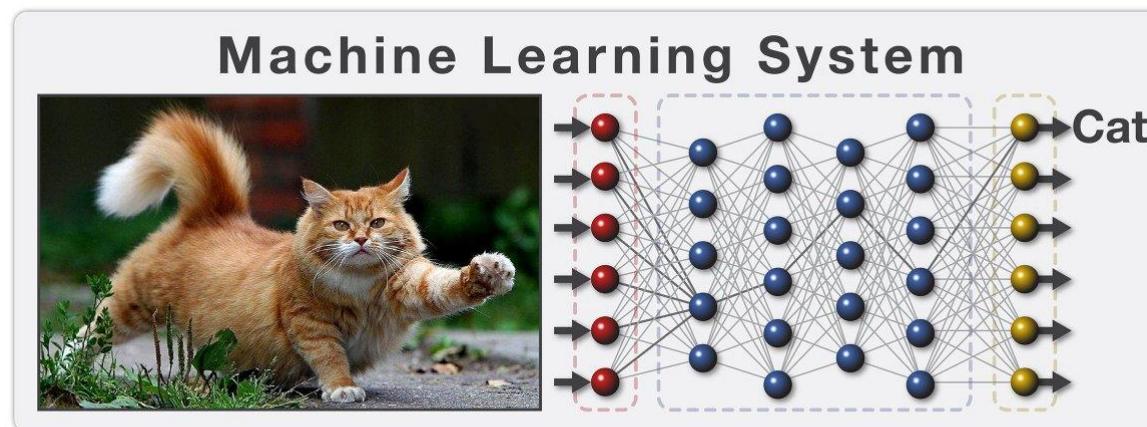


Techniques To Achieve explainability in AI <SHAP>



eXplainable AI using SHAP < DEMO>

Demo: [Session 3 – Machine Learning Fundamentals](#)



This is a cat.

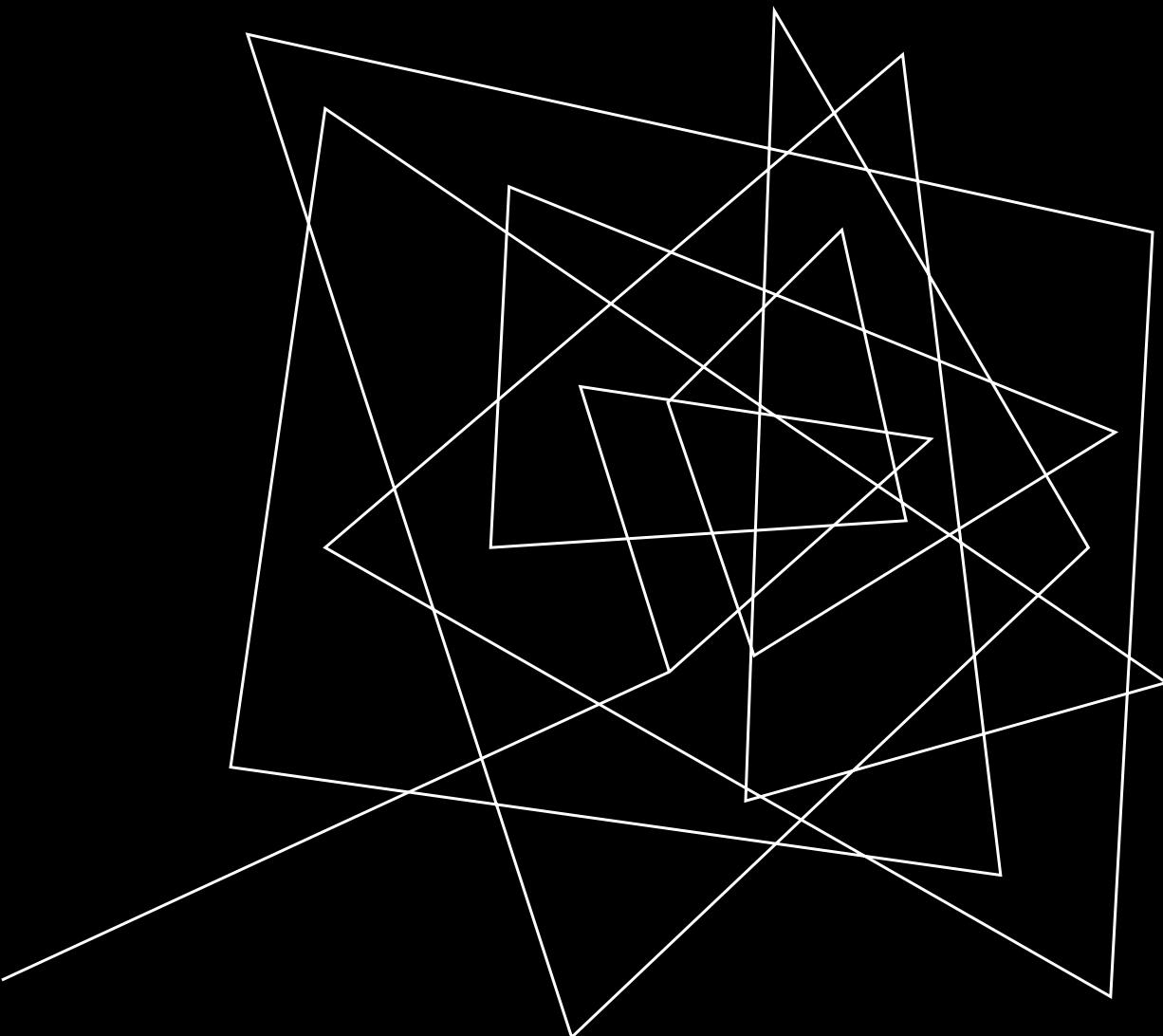
Current Explanation

This is a cat:

- It has fur, whiskers, and claws.
- It has this feature:



XAI Explanation

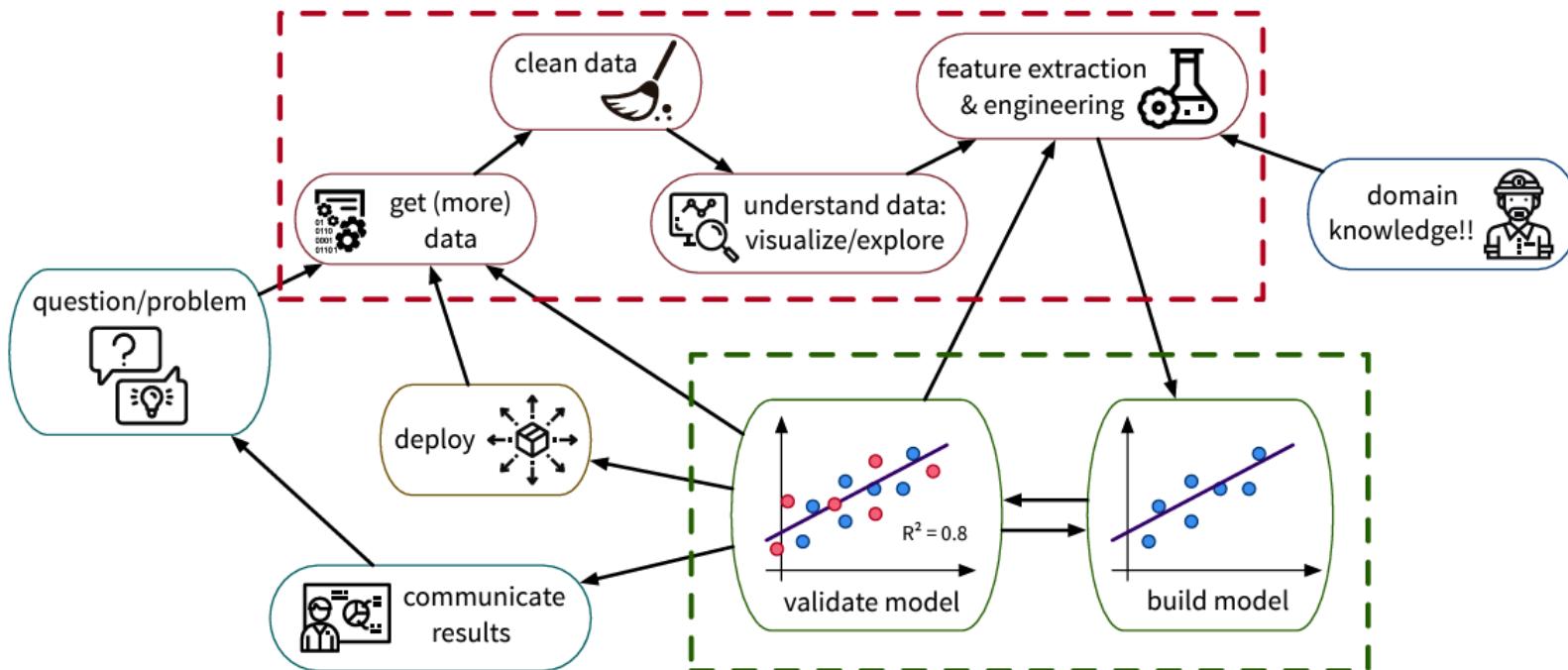


SESSION 4: FEATURE ENGINEERING AND MODEL EVALUATION

Feature engineering, types of feature engineering, and supervised ML evaluation metrics

Feature Engineering in Machine Learning

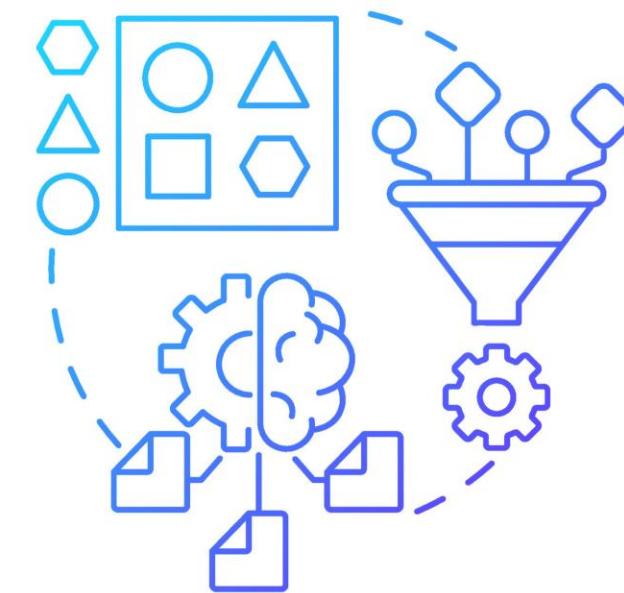
Feature engineering plays a pivotal role in the success of machine learning models. It involves the strategic refinement of input data to maximize a model's predictive capabilities. Essentially, it is the **art and science of crafting the right set of features to empower models** to uncover meaningful patterns and relationships within the data.



Feature Engineering in Machine Learning

Feature engineering is the process of meticulously selecting, transforming, and creating features to amplify the effectiveness and efficiency of machine learning models. It's similar to preparing the raw ingredients before cooking a gourmet dish – the quality and arrangement of ingredients significantly influence the final outcome. In the realm of machine learning, features are the **ingredients** and **feature engineering** is the cooking expertise that brings out the best flavors in our models.

By carefully engineering features, practitioners can enhance the model's ability to discern/extract/understand patterns, capture relevant information, and ultimately make more accurate predictions.



Why Feature Engineering? <Impact of Features on Model Performance>

Impact of Features on Model Performance: The features used in a machine learning model serve as the foundation upon which the entire predictive framework is built. The choice and quality of these features have a profound impact on the model's ability to generalize patterns from the data. In essence, the features act as the informational building blocks that guide the model in understanding the underlying relationships within the dataset.



Data Cleaning

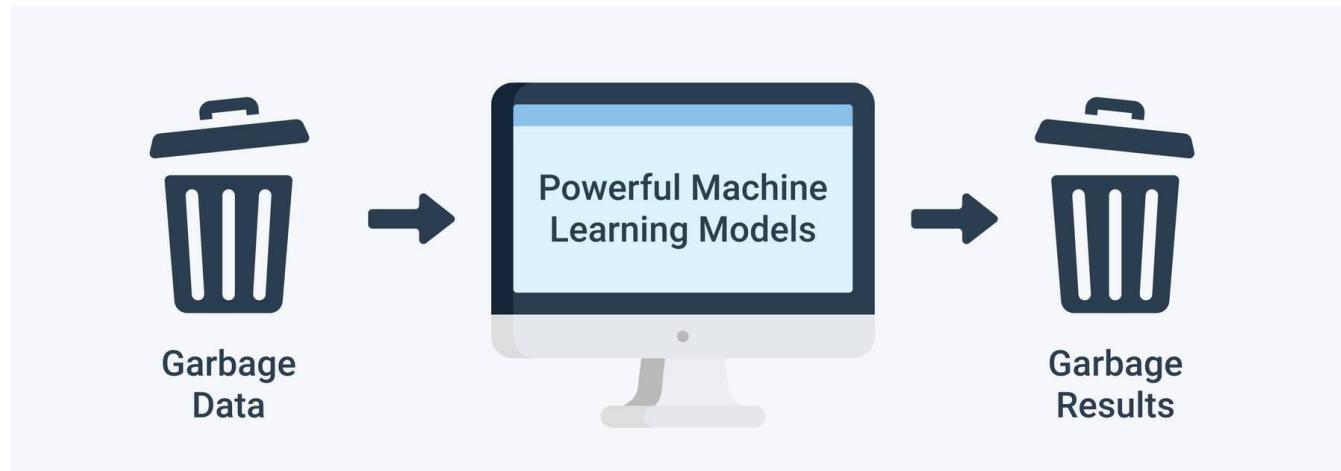
**Feature
Engineering**

Why Feature Engineering? <"Garbage In, Garbage Out" Principle>

"Garbage In, Garbage Out" Principle: A fundamental principle in the realm of machine learning is the "Garbage In, Garbage Out" principle. This concisely/briefly captures the essence of the relationship between data quality and model outcomes. If the features provided to a model are of low quality or lack relevance to the prediction task, the model's predictions are likely to be inaccurate or unreliable.

Note 1: Both quality and quantity matter

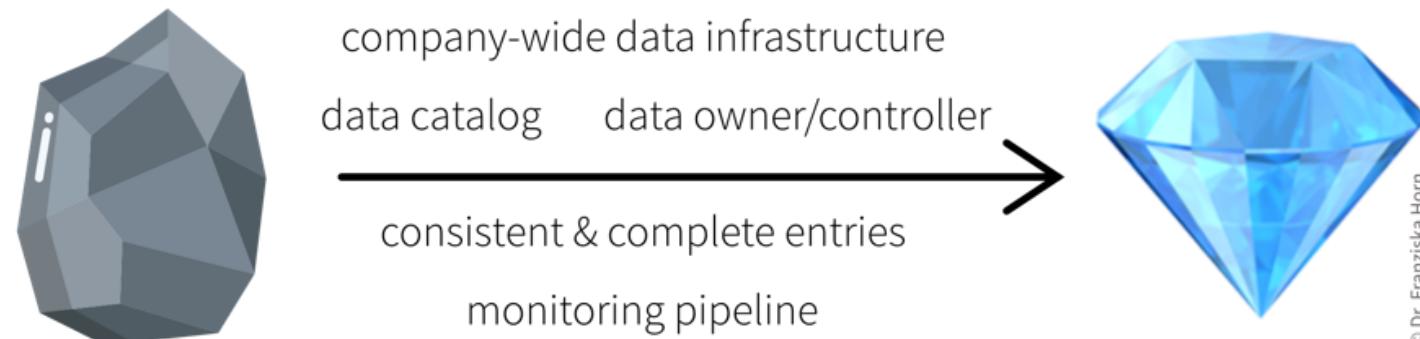
Note 2: It is not a one-off task, but rather a continuous process



Why Feature Engineering?

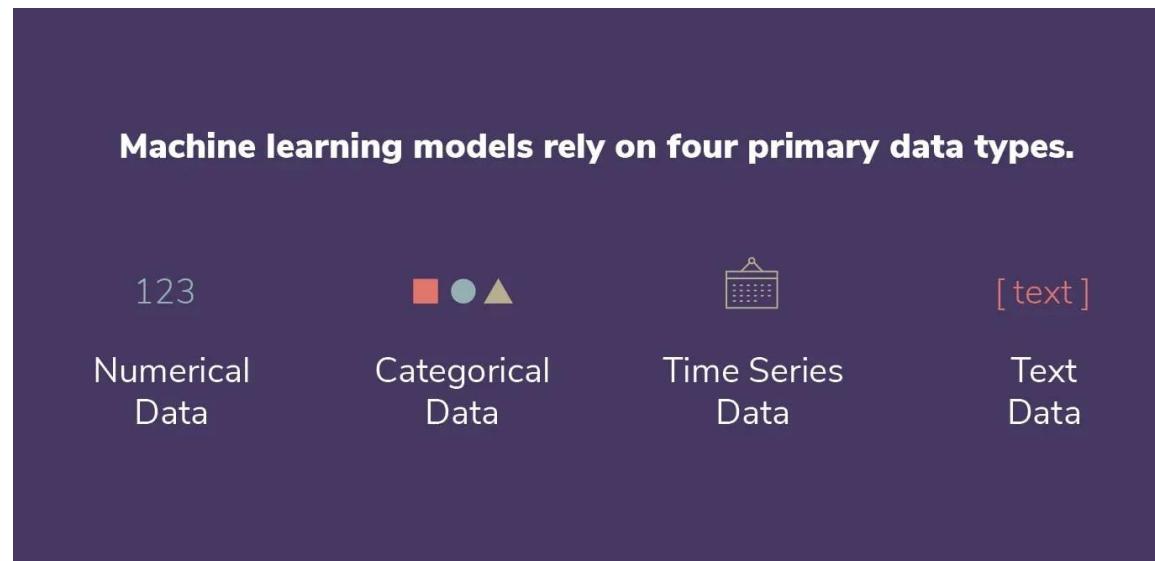
Quality of Features Directly Affects Predictions: The quality of predictions is directly tied to the quality of features. Well-crafted features can illuminate intricate patterns in the data, allowing the model to make informed decisions. Conversely, inadequate features can obscure meaningful information or introduce noise, hindering the model's ability to discern relevant patterns.

Note: keep in mind that the care and consideration given to features significantly influence the overall performance and reliability of machine learning models.



Types of Features

In the diverse landscape of machine learning, data comes in various forms, each requiring unique handling and consideration. Understanding the types of features is fundamental to effective feature engineering.

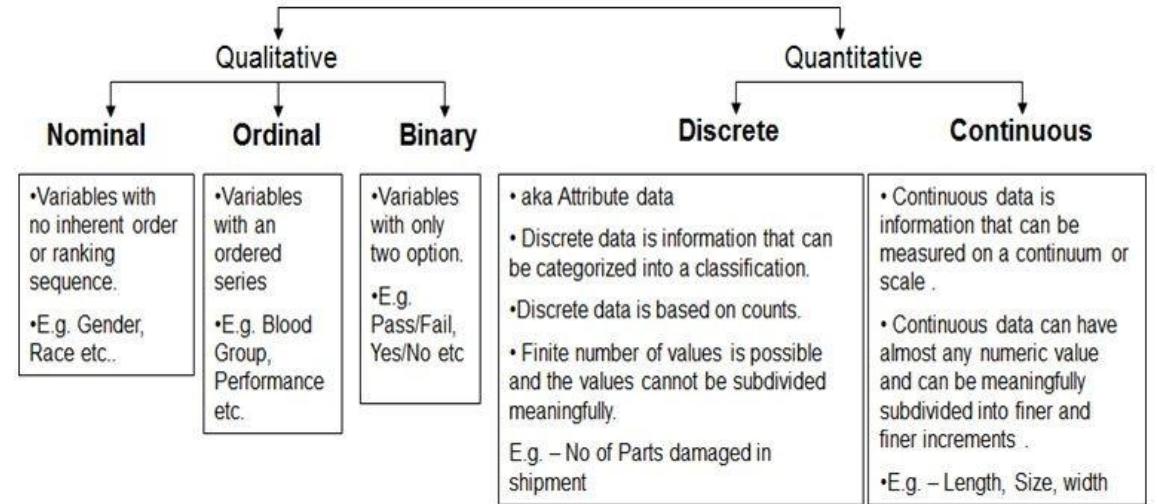


Source: [datarobot.com](https://www.datarobot.com)

Types of Features: Categorical and Numerical Data

1. Categorical Features: Categorical features represent qualitative data and can take on discrete values that often denote categories or labels. Examples include colors, gender, or types of products. Proper encoding techniques, such as one-hot encoding or label encoding, are essential to translate categorical features into a format understandable by machine learning algorithms.

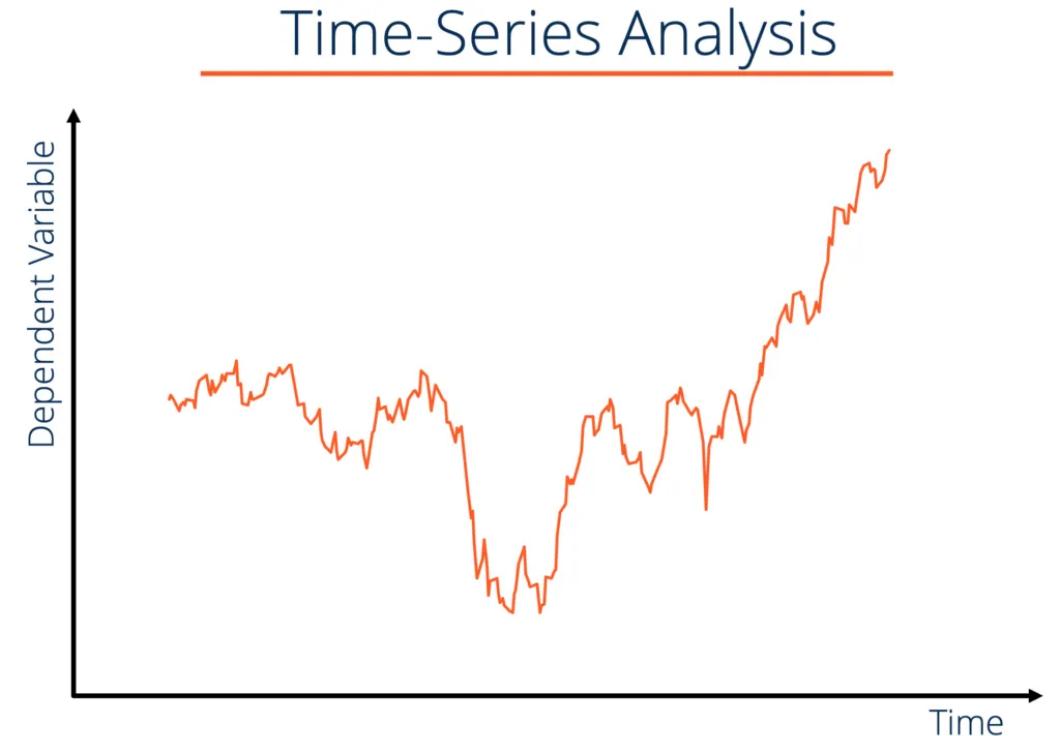
2. Numerical Features: Numerical features represent quantitative data and take on numeric values. These could include measurements such as height, weight, or temperature. Numerical features may further be divided into two subtypes: discrete (countable) and continuous (infinitely divisible). Scaling and normalization are common preprocessing steps for numerical features to ensure their comparable impact on the model.



Types of Features: Text and Time-series Features

3. Text Features: Text features involve unstructured data, such as sentences, paragraphs, or entire documents. NLP (Natural Language Processing) techniques are often employed to extract meaningful information from text features. Methods like tokenization, TF-IDF (Term Frequency-Inverse Document Frequency), and word embeddings are commonly used to convert text into a format suitable for machine learning models.

4. Time Series Features: Time series features involve data points collected or recorded over time. Examples include stock prices, temperature readings, or daily sales figures. Handling time series features requires specialized techniques like lag features, rolling statistics, and time-based aggregations to capture temporal patterns and trends effectively.



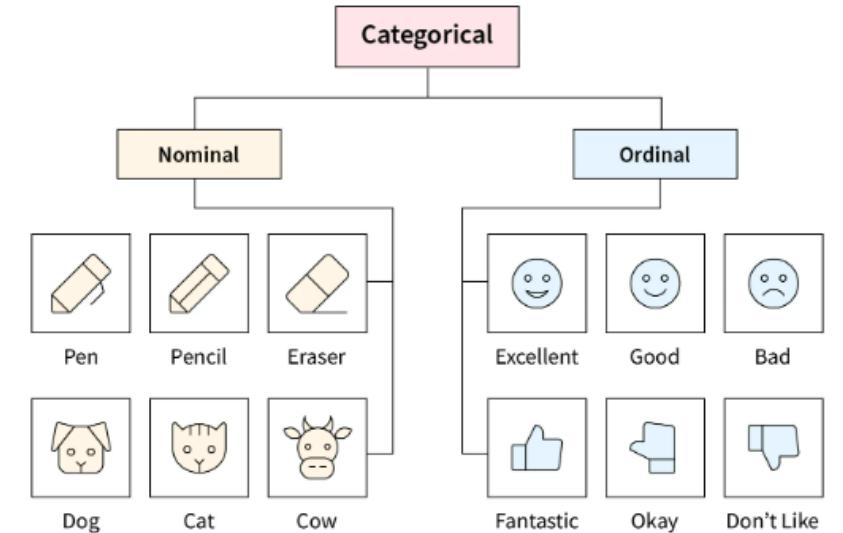
Feature Engineering: Missing Data

Techniques for Handling Missing Data:

- **Imputation:** Imputation involves filling in missing values with estimated or calculated values. Common imputation methods include mean, median, or mode imputation, where missing values are replaced with the mean, median, or mode of the observed values. This helps maintain the overall distribution of the feature.
- **Deletion:** Deletion involves removing instances or features with missing values. While straightforward, this approach can lead to a loss of valuable information, especially if the missing values are not randomly distributed. Careful consideration is needed to assess the impact of deletion on model performance.
- **Advanced Methods - Predictive Modeling for Imputation:** Leveraging advanced techniques, such as predictive modeling, offers a more sophisticated approach to handling missing data. This involves using the non-missing features to predict and impute missing values. Techniques like regression imputation or machine learning algorithms can be employed to create accurate predictions for missing values.

Feature Engineering: Handling/Encoding Categorical Data

- Definition:** Categorical encoding is the process of converting categorical data, which represents qualitative variables with discrete categories, into a numerical format that can be used by machine learning algorithms. Since many machine learning models require numerical input, encoding categorical variables is essential to ensure compatibility and effectiveness in the training process.
- Question:** Why encoding categorical columns/data?
- Answer:** Since most machine learning models only accept numerical variables, preprocessing the categorical variables becomes a necessary step. We need to convert these categorical variables to numbers such that the model is able to understand and extract valuable information.



Source: medium.com

Feature Engineering: Handling/Encoding Categorical Data <One Hot Encoding>

One-Hot Encoding is a popular technique for converting categorical variables into a binary matrix. Each category is transformed into a binary column, and only one of these columns is marked as "1" for the respective category while the others are set to "0". This method ensures that each category is represented independently, preventing the model from incorrectly assuming ordinal relationships between categories.

One-Hot Encoding

datagy.io

| Island | Biscoe | Dream | Torgensen |
|-----------|--------|-------|-----------|
| Biscoe | 1 | 0 | 0 |
| Torgensen | 0 | 0 | 1 |
| Dream | 0 | 1 | 0 |

Source: [datagy.io](https://datagy.io/one-hot-encoding/)

Feature Engineering: Handling/Encoding Categorical Data <Dummy Encoding>

Dummy encoding technique which slightly differs from one-hot-encoding, dummy encoding uses $N-1$ features to represent N labels/categories.

DUMMY

~~One Hot Encoding~~

datagy.io

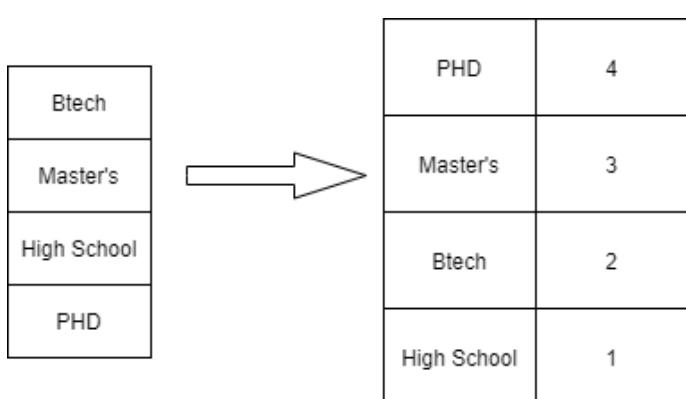
| Island | Biscoe | Dream | Torgensen |
|-----------|--------|-------|-----------|
| Biscoe | 1 | 0 | 0 |
| Torgensen | 0 | 0 | 1 |
| Dream | 0 | 1 | 0 |

Source: datagy.io

Feature Engineering: Handling/Encoding Categorical Data <Label Encoding>

Label Encoding involves assigning a unique numerical label to each category. This method is suitable when there is an ordinal relationship between categories, as the numerical values can represent the order.

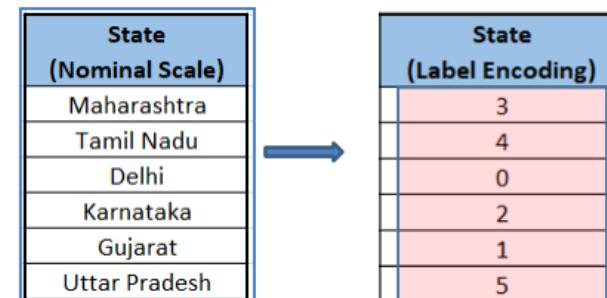
However, caution is needed when using Label Encoding for non-ordinal categorical variables, as it might mislead the model into assuming a meaningful order.



The diagram illustrates the process of Label Encoding. On the left, a vertical stack of four rectangular boxes contains the categorical labels: 'Btech', 'Master's', 'High School', and 'PHD'. An arrow points from this stack to a table on the right. The table has two columns: the first column lists the categorical labels, and the second column lists their corresponding numerical values: PHD (4), Master's (3), Btech (2), and High School (1).

| | |
|-------------|---|
| PHD | 4 |
| Master's | 3 |
| Btech | 2 |
| High School | 1 |

Source: ai-ml-analytics.com



The diagram illustrates the process of Label Encoding for a nominal scale. On the left, a vertical stack of six rectangular boxes contains the categorical labels: 'State (Nominal Scale)', 'Maharashtra', 'Tamil Nadu', 'Delhi', 'Karnataka', 'Gujarat', and 'Uttar Pradesh'. An arrow points from this stack to a table on the right. The table has two columns: the first column lists the categorical labels, and the second column lists their corresponding numerical values: Maharashtra (3), Tamil Nadu (4), Delhi (0), Karnataka (2), Gujarat (1), and Uttar Pradesh (5).

| | |
|-----------------------|---|
| State (Nominal Scale) | |
| Maharashtra | 3 |
| Tamil Nadu | 4 |
| Delhi | 0 |
| Karnataka | 2 |
| Gujarat | 1 |
| Uttar Pradesh | 5 |

Source: mygreatlearning.com

Feature Engineering: Handling/Encoding Categorical Data <Target Encoding (Mean Encoding)>

Target Encoding (or Mean Encoding) involves replacing each category with the mean of the target variable for that category. This method is particularly useful when dealing with **high-cardinality categorical features***, as it provides a smooth mapping between the categorical variable and the target variable. However, it is crucial to avoid **data leakage** by calculating target means using only the training data.

| id | job | job_mean | target |
|-----------|------------|-----------------|---------------|
| 1 | Doctor | 0,50 | 1 |
| 2 | Doctor | 0,50 | 0 |
| 3 | Doctor | 0,50 | 1 |
| 4 | Doctor | 0,50 | 0 |
| 5 | Teacher | 1 | 1 |
| 6 | Teacher | 1 | 1 |
| 7 | Engineer | 0,50 | 0 |
| 8 | Engineer | 0,50 | 1 |
| 9 | Waiter | 1 | 1 |
| 10 | Driver | 0 | 0 |

Source: [towardsdatascience.com](https://towardsdatascience.com/target-encoding-mean-encoding-in-machine-learning-10f3e0a2a2)

High-cardinality categorical features are those that have a **large number of unique values**, such as **product IDs**, **zip codes**, or **names**. These features can pose challenges for data encoding, as they can create a large number of dummy variables, increase memory usage, and reduce model performance.

Feature Engineering: Handling/Encoding Categorical Data <DEMO>

Demo: [Session 4 – Feature Engineering and Model Evaluation](#)

| id | color |
|----|-------|
| 1 | red |
| 2 | blue |
| 3 | green |
| 4 | blue |



| id | color_red | color_blue | color_green |
|----|-----------|------------|-------------|
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 |

Source: [datacamp.com](https://www.datacamp.com)

Feature Engineering: Feature Scaling

Feature scaling is a crucial preprocessing step in machine learning, ensuring that numerical features contribute equally to model training. It involves transforming the range of features to a standardized scale, preventing certain features from dominating due to their inherent scale.

| | Age | EstimatedSalary |
|--------------|------------|-----------------|
| count | 400.000000 | 400.000000 |
| mean | 37.655000 | 69742.500000 |
| std | 10.482877 | 34096.960282 |
| min | 18.000000 | 15000.000000 |
| 25% | 29.750000 | 43000.000000 |
| 50% | 37.000000 | 70000.000000 |
| 75% | 46.000000 | 88000.000000 |
| max | 60.000000 | 150000.000000 |

Before Min-Max Scaling

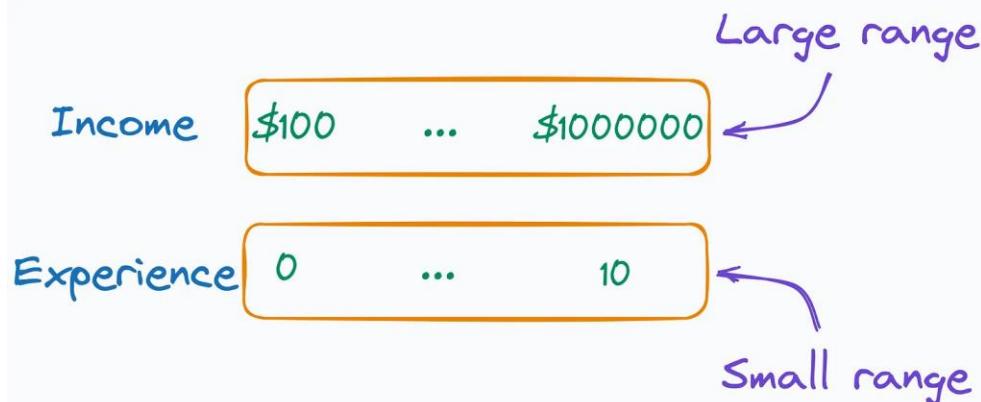
| | Age | EstimatedSalary |
|--------------|------------|-----------------|
| count | 400.000000 | 400.000000 |
| mean | 0.467976 | 0.405500 |
| std | 0.249592 | 0.252570 |
| min | 0.000000 | 0.000000 |
| 25% | 0.279762 | 0.207407 |
| 50% | 0.452381 | 0.407407 |
| 75% | 0.666667 | 0.540741 |
| max | 1.000000 | 1.000000 |

After Min-Max Scaling

Feature Engineering: Why Feature Scaling?

Feature scaling is commonly used to improve the performance and stability of ML models.

This is because it **scales** the data to a **standard range**. This **prevents** a specific **feature** from having a **strong influence** on the model's **output**.

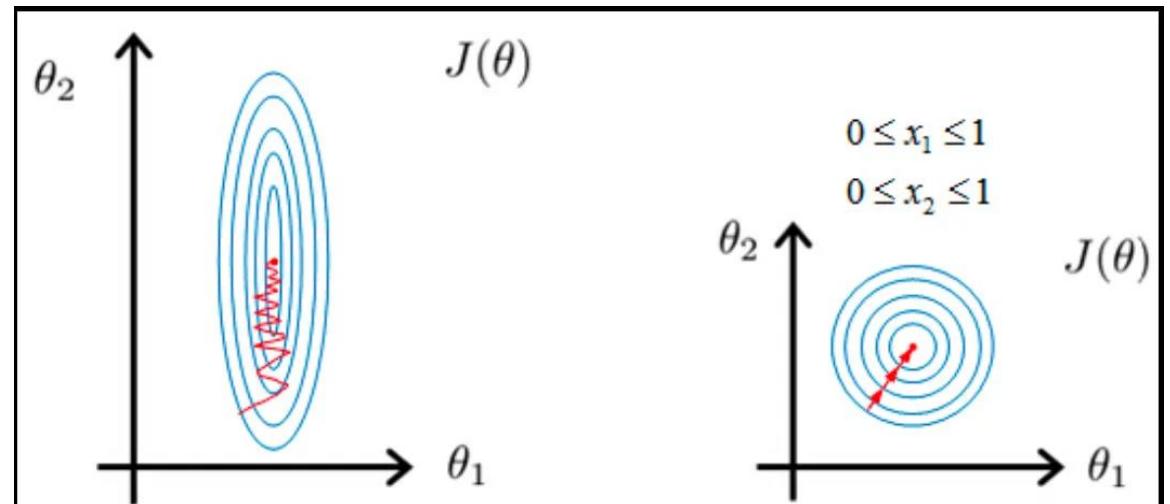


Source: dailydoseofds.com

For instance, in the image above, the scale of Income could massively impact the overall prediction. Scaling both features to the same range can mitigate this and improve the model's performance.

Feature Engineering: Why Feature Scaling?

Feature scaling is essential for algorithms that rely on distance measures, such as k-nearest neighbors or support vector machines. It helps in achieving better convergence for gradient-based optimization algorithms and ensures that the model is not biased towards features with larger magnitudes.



Gradient Descent: Attaining Global minimum before and after scaling

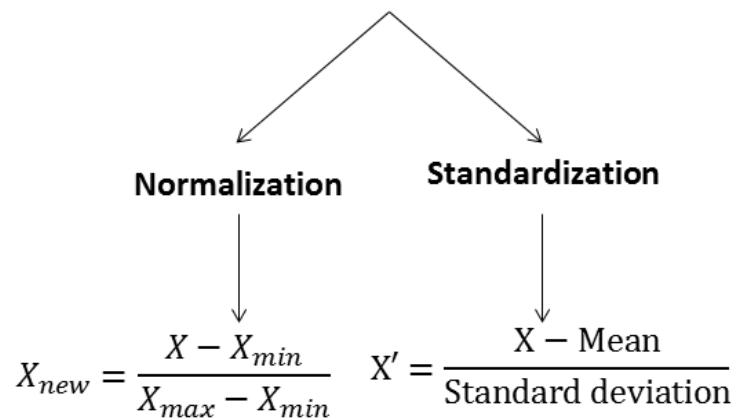
Source: medium.com

Feature Engineering: Feature Scaling <Normalization vs. Standardization>

Two common techniques for feature scaling are **normalization** and **standardization**, which include methods such as **Min-Max Scaling** and **Z-score Standardization**.

- **Normalization:** Scales the features to a range between 0 and 1. It's particularly useful when the features have varying ranges, and you want to bring them to a uniform scale. Normalization is performed using the formula: $X_{scaled} = (X - X_{min}) / (X_{max} - X_{min})$
- **Standardization:** Standardizes the features to have a mean of 0 and a standard deviation of 1. This is effective when the features follow a Gaussian distribution. Standardization is performed using the formula: $X_{scaled} = (X - \text{mean}(X)) / (\text{std}(X))$

Feature scaling



Feature Engineering: Feature Scaling <Normalization: Min-Max Scaling>

Min-Max Scaling, a type of normalization, transforms the features to a specific range, usually between 0 and 1. This is achieved by using the formula:

$$X' = \frac{X - \min(X)}{\max(X) - \min(X)}$$

Note: Min-Max Scaling is sensitive to outliers, and it's important to handle extreme values appropriately.

Feature Engineering: Feature Scaling <Standardization: Z-Score>

Z-score Standardization transforms the features to have a mean of 0 and a standard deviation of 1. This method is effective for features that follow a Gaussian distribution. The formula for Z-score Standardization is:

$$X' = \frac{X - \text{mean}(X)}{\text{std}(X)}$$

Note: Z-score Standardization is less sensitive to outliers compared to Min-Max Scaling.

Feature Engineering: Feature Scaling is Not Always Necessary

Feature Scaling is NOT Always Necessary



| Algorithm | Test Set Classification Performance | | |
|---------------------------|-------------------------------------|----------------------|-------------------|
| | Without Feature Scaling | With Feature Scaling | Scaling Required? |
| Logistic Regression | 0.53 | 0.70 | YES |
| Support Vector Classifier | 0.72 | 0.94 | YES |
| MLP Classifier | 0.73 | 0.89 | YES |
| kNN Classifier | 0.66 | 0.93 | YES |
| Decision Tree | 0.83 | 0.83 | NO |
| Random Forest | 0.91 | 0.91 | NO |
| Gradient Boosting | 0.86 | 0.86 | NO |
| Naive Bayes | 0.75 | 0.75 | NO |

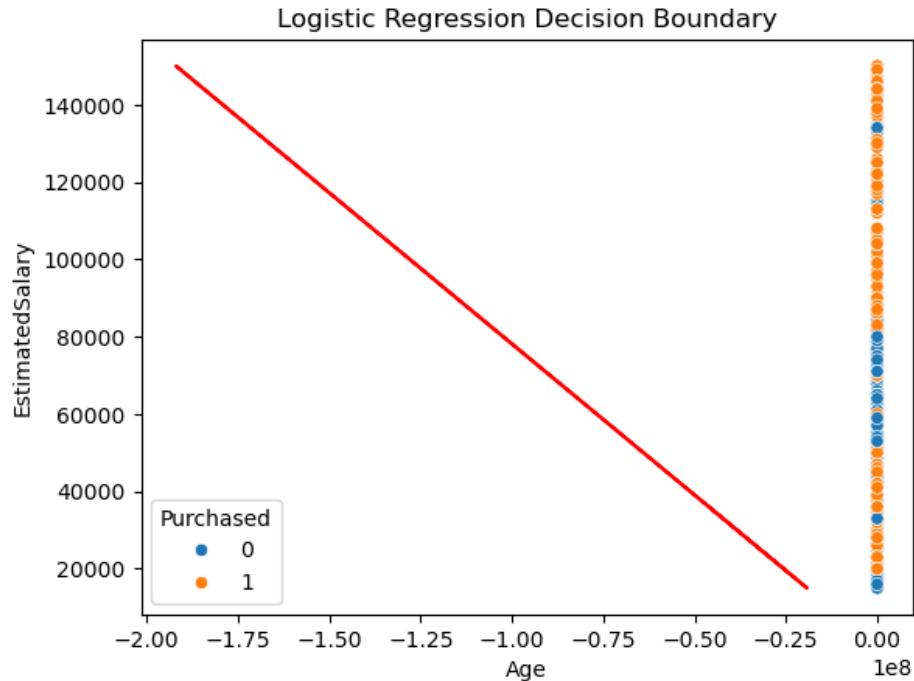
Better performance with feature scaling

Same performance irrespective of feature scaling

Feature Engineering: Feature Scaling

<DEMO>

Demo: Session 4 – Feature Engineering and Model Evaluation



Before Min-Max Scaling



After Min-Max Scaling

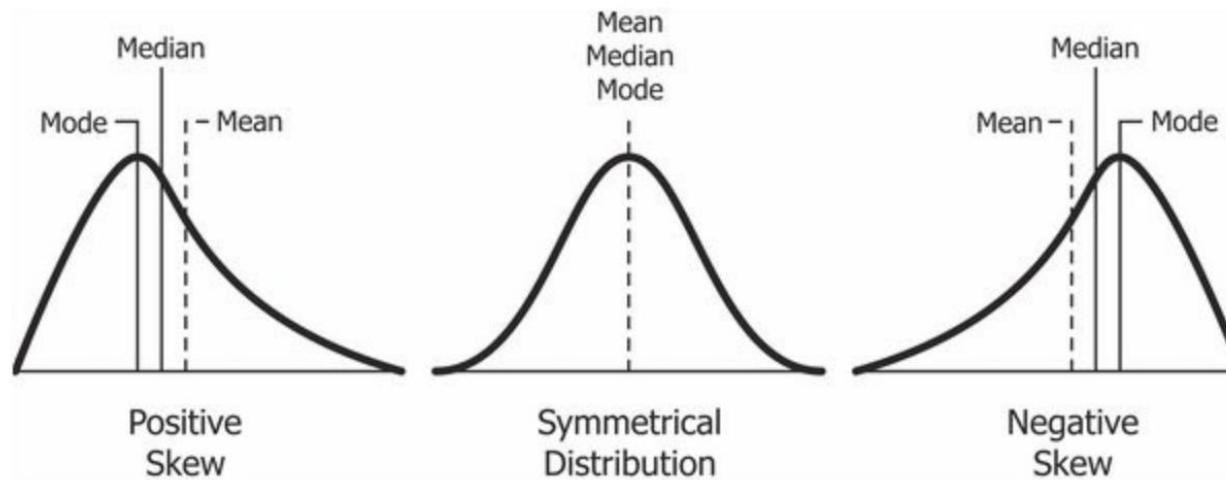
Feature Engineering: Feature Transformation

<Why Applying Feature Transformation>

Skewed Data Problem

One problem that can arise when working with datasets is skewed data. Skewed data occurs when the distribution of a variable is not evenly distributed, resulting in an unbalanced data set. This can negatively affect the performance of machine learning algorithms because they cannot learn patterns from data effectively.

Note: Most of ML algorithms assume that the data is normally distributed!

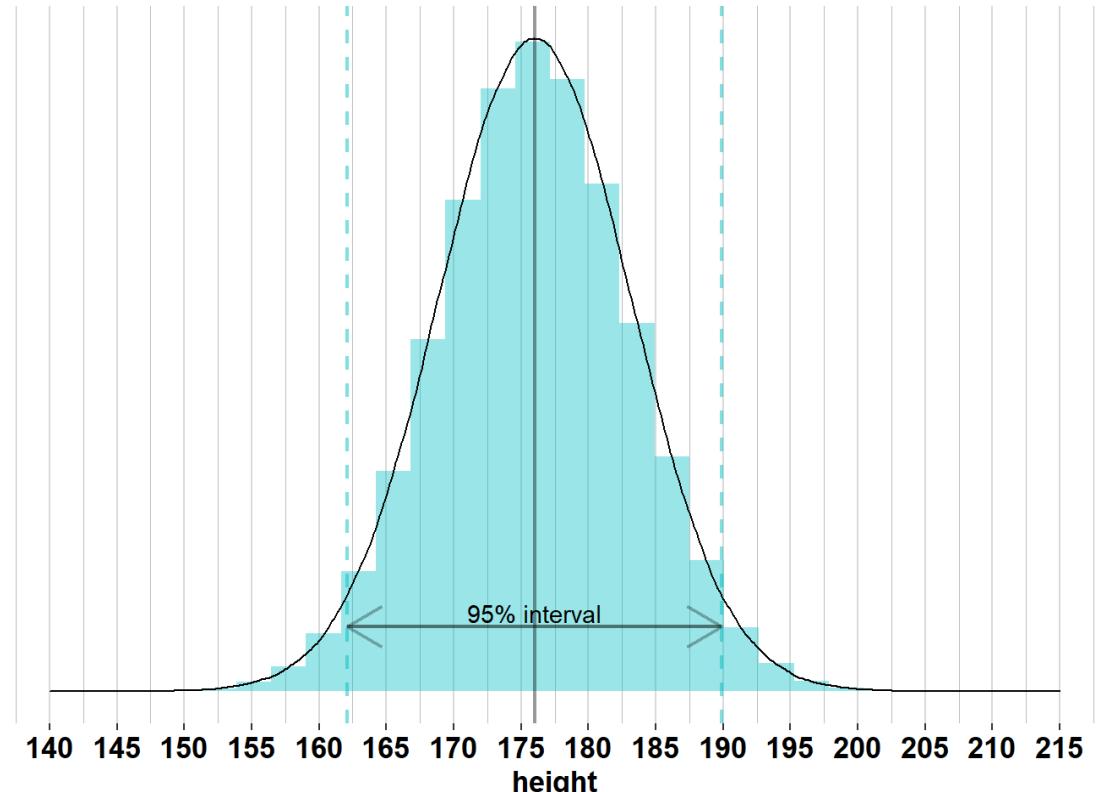


Source: [wikipedia.com](https://en.wikipedia.org/wiki/Skewness)

Feature Engineering: Feature Transformation

<Why Applying Feature Transformation>

Like **Linear** and **Logistic regression**, some data science models assume that the variables follow a **normal distribution**. More likely, variables in real datasets will follow a skewed distribution. By applying some transformations to these skewed variables, we can map this skewed distribution to a normal distribution to increase the performance of our models. As we know, **Normal Distribution** is a very important distribution in Statistics, which is key to many statisticians for solving problems in statistics. Usually, the data distribution in Nature follows a Normal distribution like - age, income, height, weight, etc. But the features in the real-life data are not normally distributed. However, it is the best approximation when we are unaware of the underlying distribution pattern.



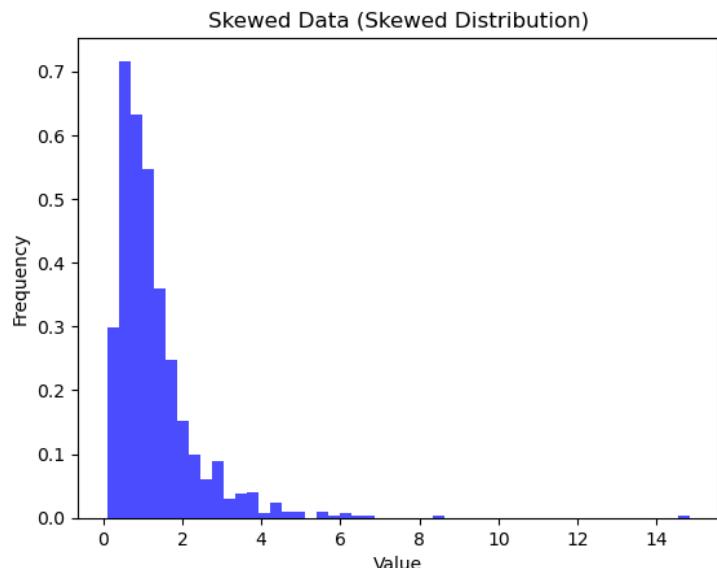
Source: javatpoint.com

Source: bookdown.org

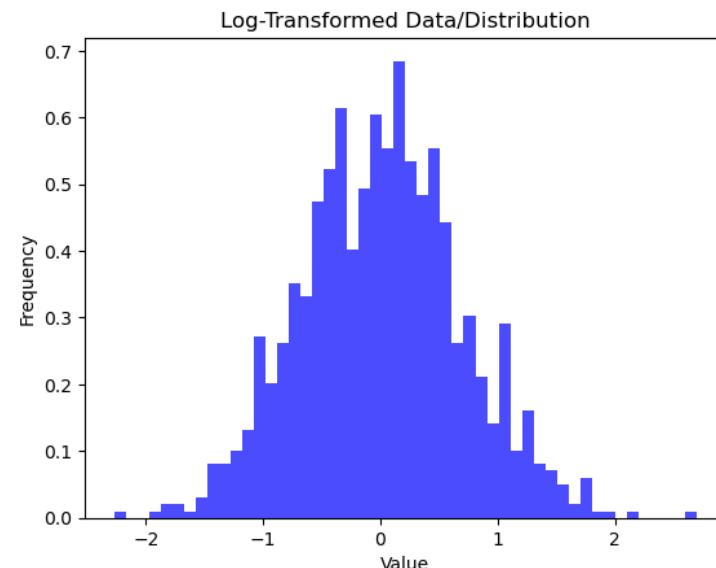
Feature Engineering: Feature Transformation <Logarithmic Transformation>

Logarithmic transformation is applied to features to reduce the impact of extreme values and make the distribution more symmetric. This is especially beneficial when dealing with skewed data. Logarithmic transformation is implemented using the logarithmic function.

Note: This transformation is not applied to those features which have negative values



Before Log-Transformation

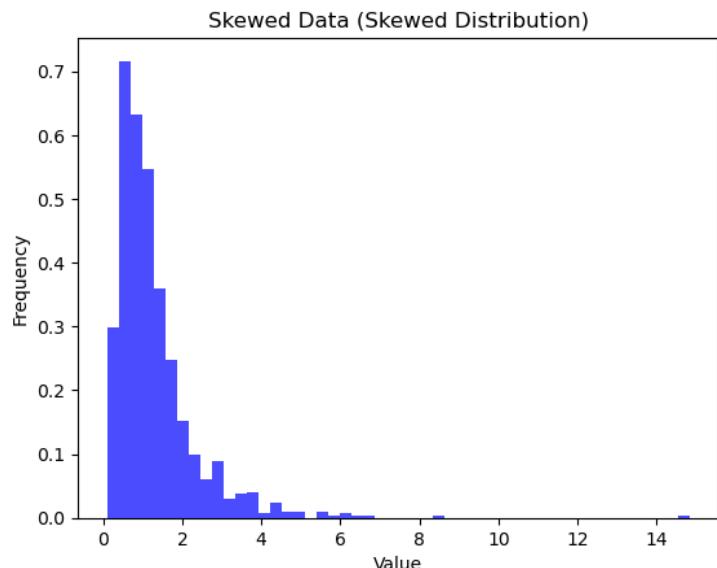


After Log-Transformation

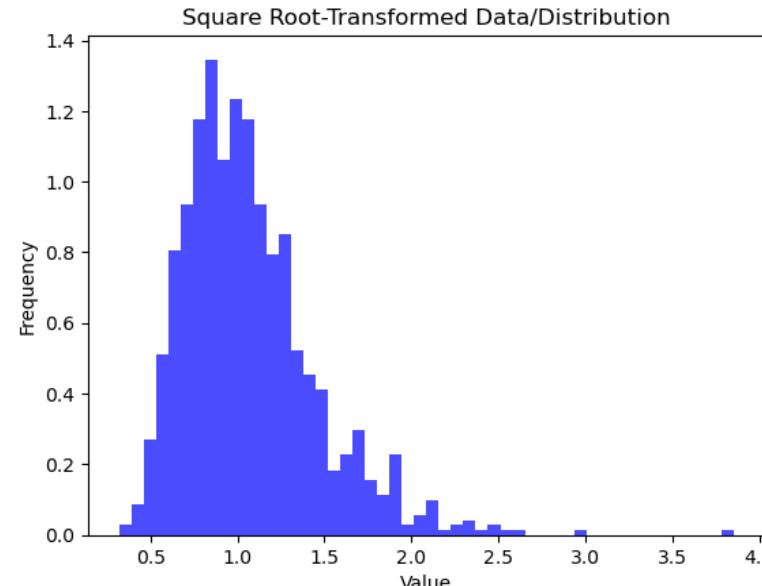
Feature Engineering: Feature Transformation

<Square Root Transformation>

Square Root Transformation: This transformation is defined **only** for **positive numbers**. This can be used for reducing the skewness of right-skewed data. This transformation is weaker than Log Transformation.



Before Sqrt-Transformation

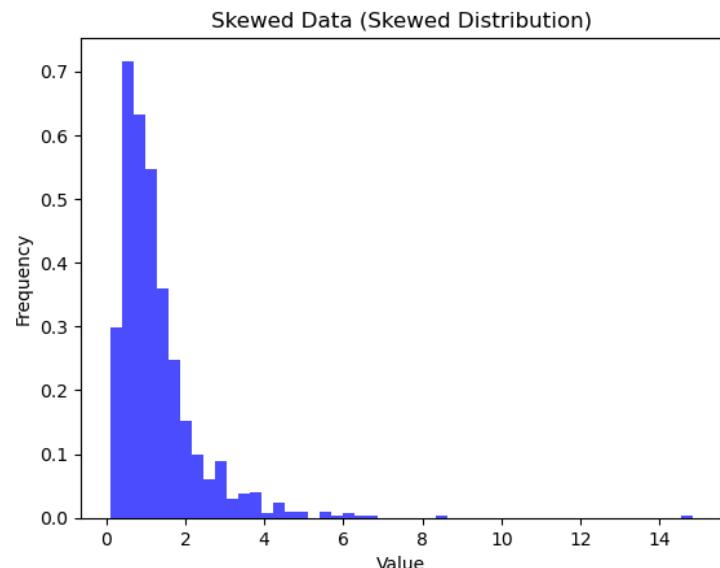


After Sqrt-Transformation

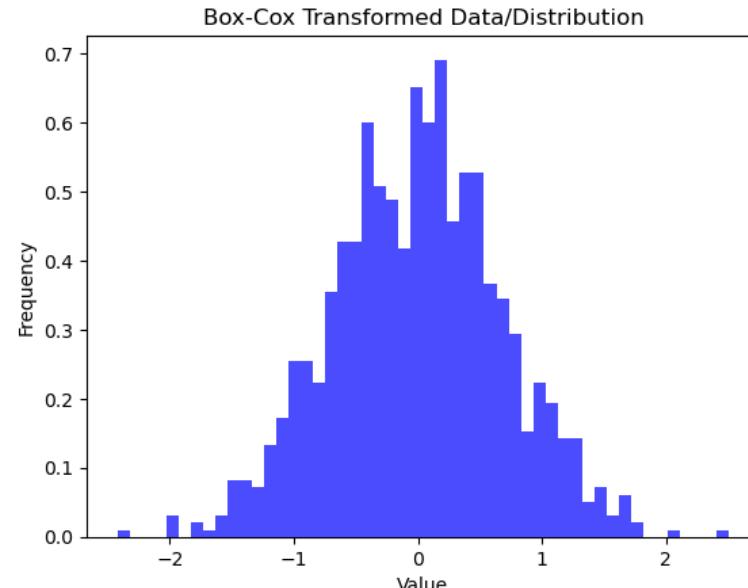
Feature Engineering: Feature Transformation <Box-Cox Transformation>

The Box-Cox transformation is a family of power transformations that includes logarithmic transformation as a special case. It is useful for stabilizing variance and making the data more closely approximate a normal distribution. The Box-Cox transformation requires the data to be strictly positive; if the data contains zeros or negative values, alternative transformations might be considered.

$$y_i^{(\lambda)} = \begin{cases} \frac{y_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln(y_i) & \text{if } \lambda = 0, \end{cases}$$



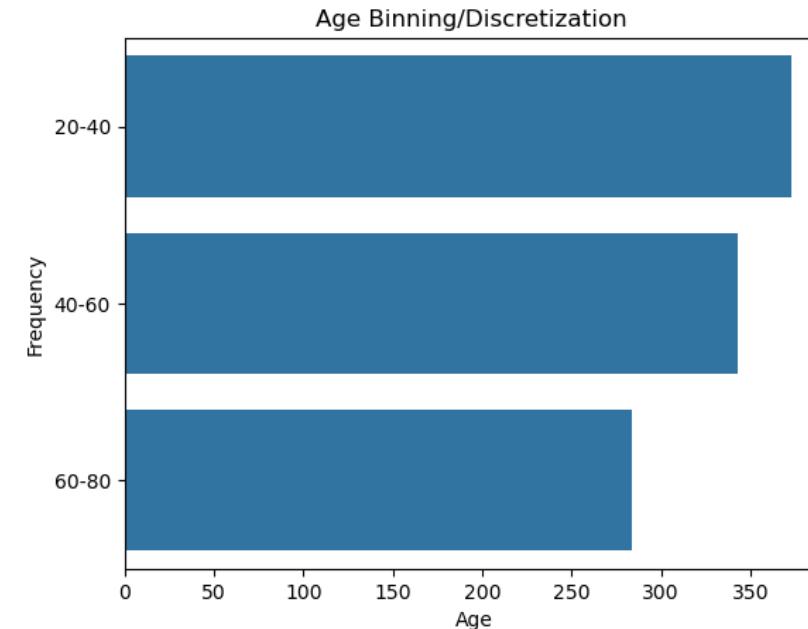
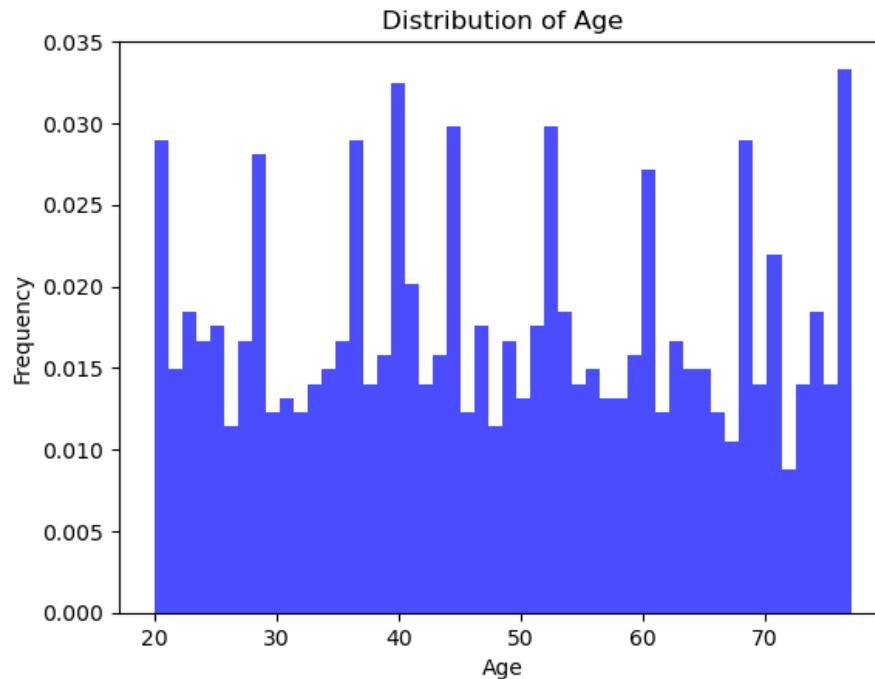
Before Box-Cox-Transformation



After Box-Cox-Transformation

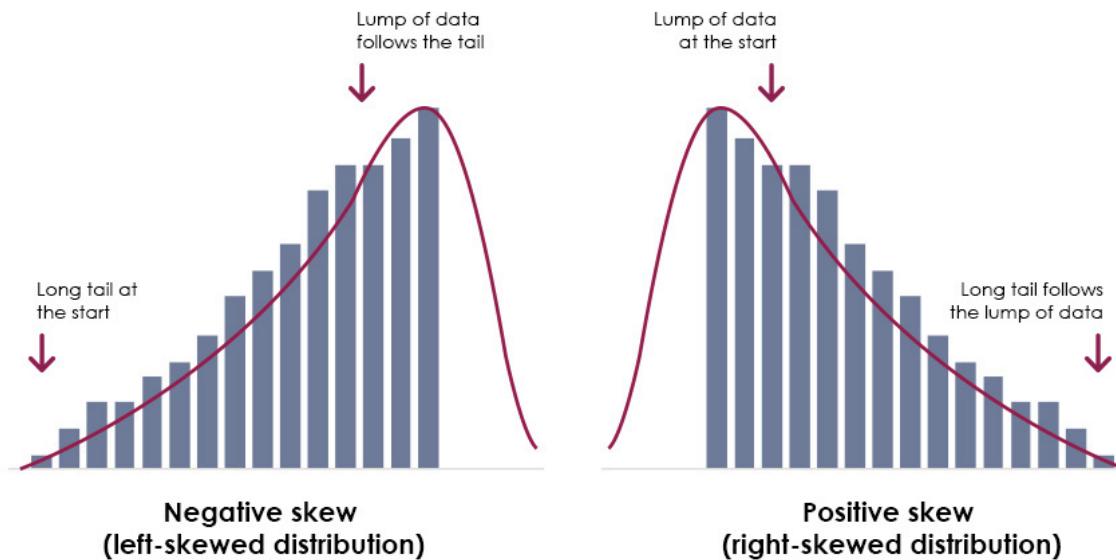
Feature Engineering: Feature Transformation <Feature Binning>

Feature binning, also known as discretization or bucketing, is a data preprocessing technique that involves grouping continuous or numerical features into discrete bins or intervals. The primary goal of feature binning is to convert numerical data into categorical data by partitioning the range of values into a set of predefined bins.



Feature Engineering: Feature Transformation <DEMO>

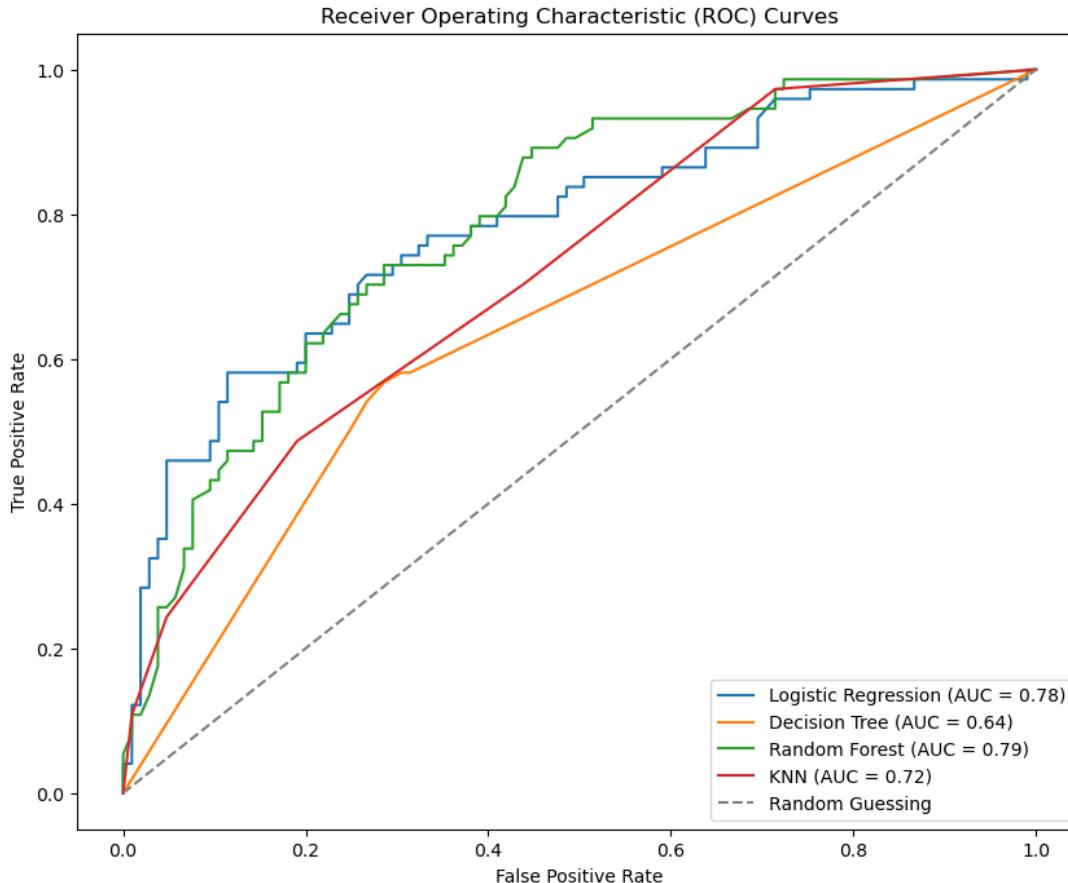
Demo: [Session 4 – Feature Engineering and Model Evaluation](#)



Source: cambridgemaths.org

Model Evaluation Metrics for Supervised Learning Models

Since in supervised learning problems, we know the ground truth (the actual/true outputs), we can objectively evaluate different models and benchmark them against each other.



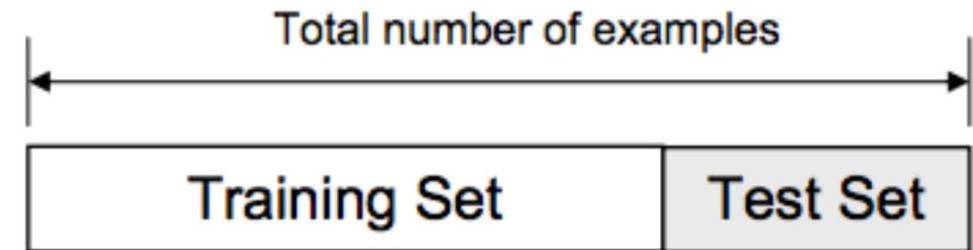
Model Evaluation Metrics for Supervised Learning Models

Is this a good model for the task?

I.e., does the model generate reliable predictions for new data points?

- Split the data into training and test sets to be able to get a reliable estimate of how the model will later perform when applied to new data points that it wasn't trained on.
- Quantify the quality of the model's predictions on the test set with a suitable evaluation metric (depending on the problem type, i.e., classification or regression).

Note: Compare the model to a '**stupid baseline**' that predicts the mean (→ regression) or most frequent class (→ classification).



Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Accuracy>

| Actual Output | Predicted Output |
|---------------|------------------|
| 0 | 1 |
| 0 | 0 |
| 1 | 1 |
| 1 | 1 |
| 0 | 0 |
| 1 | 0 |
| 1 | 0 |

The simplest way to evaluate this ML model is to compute the fraction of the correct predictions out of the number of predictions, as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Model Evaluation Metrics for Supervised Learning Models:

<Classification Evaluation Metrics: Accuracy>

Confusion Matrix: A **confusion matrix** is a **table** used in **classification** to evaluate the performance of a machine learning model. It provides a summary of the predicted and actual classifications, breaking down the results into four categories: **true positives** (correctly predicted positive instances), **true negatives** (correctly predicted negative instances), **false positives** (incorrectly predicted as positive), and **false negatives** (incorrectly predicted as negative). This matrix is particularly useful for assessing the accuracy and effectiveness of a classification model by revealing the types and frequencies of classification errors.

| Predicted Label | True Label | |
|-----------------|---------------------|---------------------|
| | + | - |
| + | True Positive (TP) | False Positive (FP) |
| - | False Negative (FN) | True Negative (TN) |

$$\frac{TP + TN}{TP + TN + FP + FN} = \frac{\boxed{TP + TN}}{\boxed{n}}$$

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Accuracy>



Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Accuracy>

Some potential **problems** with relying only on **accuracy**

| Problem | Description | Examples |
|---------------------------|---|---|
| Imbalanced Datasets | In datasets where one class significantly outweighs the others (class imbalance), a model can achieve high accuracy by simply predicting the majority class most of the time. | In a medical dataset where only 5% of patients have a rare disease, a model that always predicts "no disease" would still achieve 95% accuracy. |
| Misleading in Rare Events | In applications where detecting rare events is crucial (e.g., fraud detection, disease diagnosis), accuracy can be misleading. A high accuracy score may not reflect the model's ability to correctly identify these critical cases. Accuracy may not reflect the model's ability to detect rare, but important, occurrences. | In fraud detection, a model that misses rare instances of fraud (false negatives) could have high accuracy but would be ineffective. |

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Recall and Precision>

Recall and precision are two important metrics used in the evaluation of classification models. They are often used together, especially when dealing with imbalanced datasets.

Recall (Sensitivity or True Positive Rate):

- **Definition:** Recall measures the ability of a classification model to capture and correctly identify all relevant instances in a dataset. It is the ratio of true positives to the sum of true positives and false negatives.
- **Formula:** Recall = True Positives / (True Positives + False Negatives)
- **Interpretation:** A high recall indicates that the model is good at identifying most of the positive instances in the dataset. However, it may also lead to a higher number of false positives.

$$\text{Recall} = \frac{\text{True Positive}(TP)}{\text{True Positive}(TP) + \text{False Negative}(FN)}$$

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: Recall and Precision>

Recall and precision are two important metrics used in the evaluation of classification models. They are often used together, especially when dealing with imbalanced datasets.

Precision (Positive Predictive Value):

- **Definition:** Precision measures the accuracy of the positive predictions made by a model. It is the ratio of true positives to the sum of true positives and false positives.
- **Formula:** Precision = True Positives / (True Positives + False Positives)
- **Interpretation:** A high precision indicates that when the model predicts a positive instance, it is likely to be correct. However, it may miss some positive instances, resulting in a lower recall.

$$\text{Precision} = \frac{\text{True Positive}(TP)}{\text{True Positive}(TP) + \text{False Positive}(FP)}$$

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: F1-Score>

- **Recall** focuses on minimizing false negatives, making sure that all actual positive instances are identified by the model.
- **Precision** focuses on minimizing false positives, ensuring that the positive predictions made by the model are accurate.
- The trade-off between recall and precision is often application-dependent. In some cases, you may want to prioritize recall over precision, and vice versa. The **F1 Score**, which is the harmonic mean of precision and recall, is a metric that combines both and is useful for finding a balance between the two when needed:

$$\mathbf{F1\ Score} = \frac{2}{\left(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}} \right)}$$

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

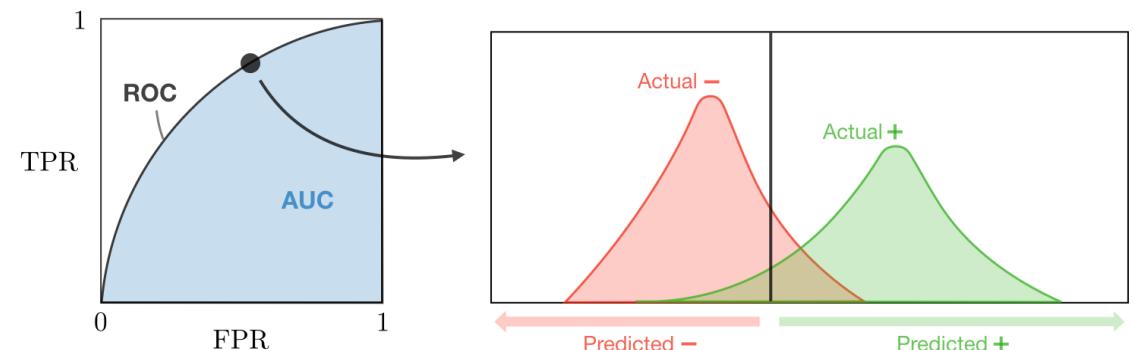
Model Evaluation Metrics for Supervised Learning Models:

<Classification Evaluation Metrics: ROC-AUC>

ROC: The receiver operating curve, also noted as ROC, is the plot of TPR versus FPR by varying the threshold. These metrics are summed up below:

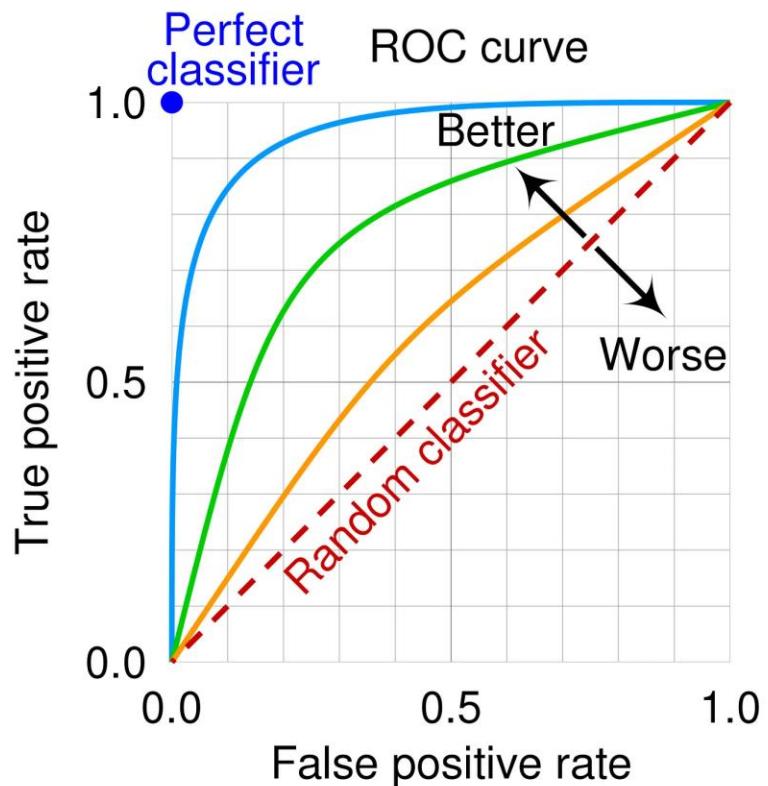
| Metric | Formula | Equivalent |
|----------------------------|----------------------|---------------------|
| True Positive Rate TPR | $\frac{TP}{TP + FN}$ | Recall, sensitivity |
| False Positive Rate FPR | $\frac{FP}{TN + FP}$ | 1-specificity |

AUC: The area under the receiving operating curve, also noted AUC or AUROC, is the area below the ROC as shown in the following figure:



Source: <https://stanford.edu/>

Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: ROC-AUC>



Model Evaluation Metrics for Supervised Learning Models: <Classification Evaluation Metrics: DEMO>

Example Scenario:

Let's say we have a binary classification problem for a medical test that identifies a disease:

- Out of 100 actual cases of the disease:
 - The test correctly identifies 80 as positive (True Positives).
 - The test incorrectly identifies 20 as negative (False Negatives).
- Out of 100 non-cases (healthy individuals):
 - The test correctly identifies 90 as negative (True Negatives).
 - The test incorrectly identifies 10 as positive (False Positives).

Find the confusion matrix and compute the accuracy, recall, precision, and specificity!!

Model Evaluation Metrics for Supervised Learning Models: <Regression Evaluation Metrics>

These regression metrics serve different purposes in evaluating the performance of regression models. **MAE**, **MSE**, and **RMSE** focus on the accuracy of predictions, while **R² (R-squared)** assesses the overall goodness of fit of the model. Depending on the specific characteristics of the problem, different metrics may be more appropriate for evaluation.

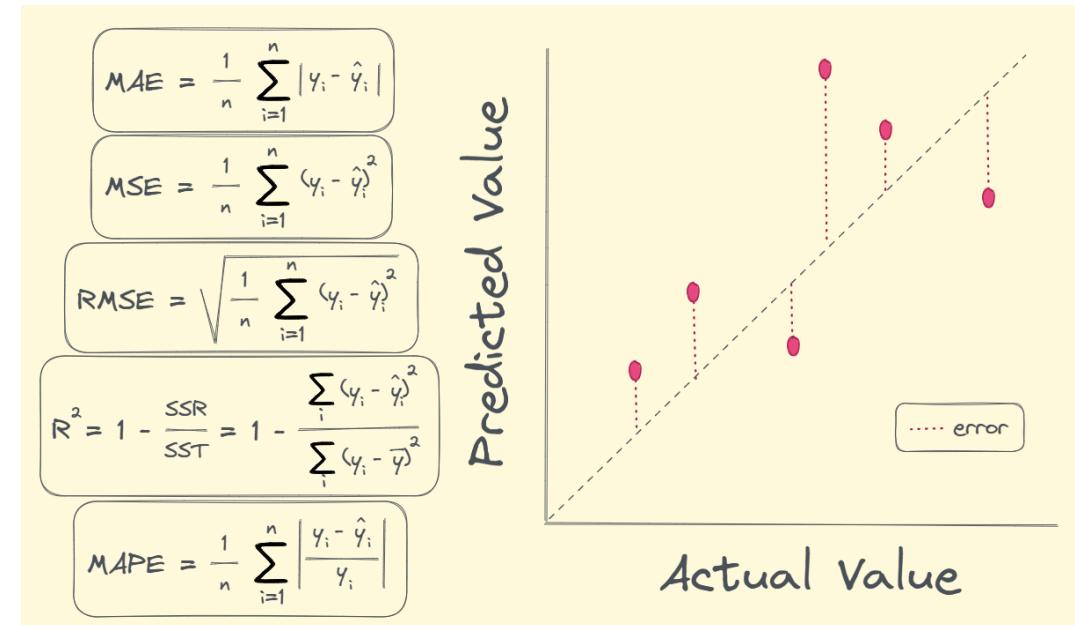
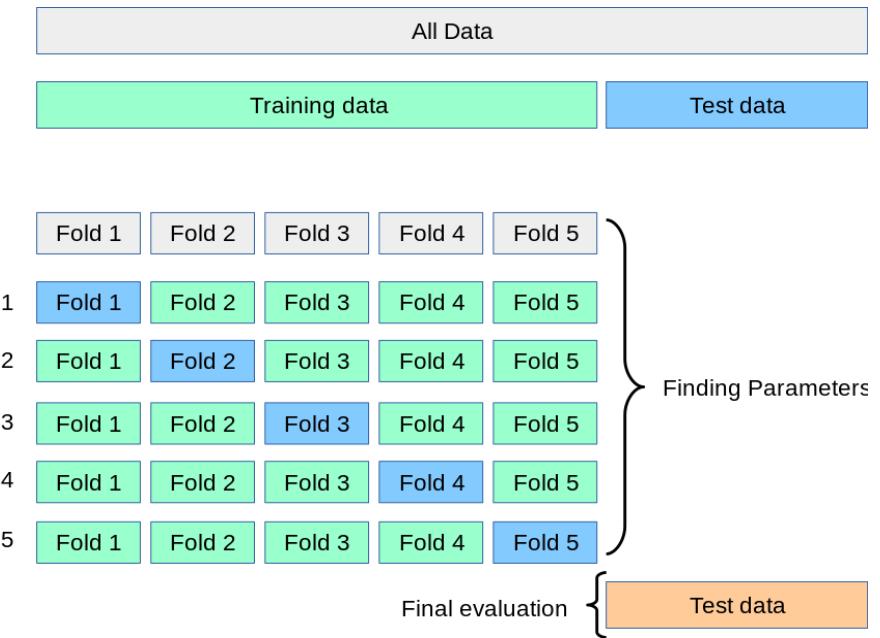


Fig. Regression evaluation metrics

Model Evaluation Metrics for Supervised Learning Models

K-fold cross-validation: One of the techniques to evaluate machine learning models on limited data samples and one of the employed techniques to detect overfitting and how well a model will generalize to new, unseen data.

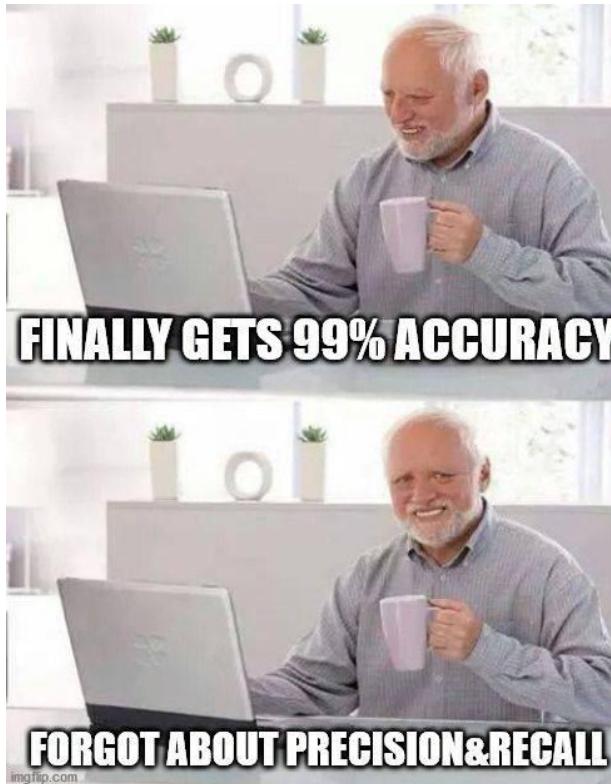
- **Idea:** The k-fold cross-validation (CV) refers to the **k** number of the folds or subset division of the dataset (The model is then trained and evaluated **k** times, using each fold as the testing set once and the remaining **k-1** folds as the training set). It is generally a technique used in machine learning to evaluate the performance of the models on the entire dataset, it ensures that every data point from the dataset has the chances of appearing in both training and testing sets, and thus it is one of the best approaches to evaluate the models' performance.

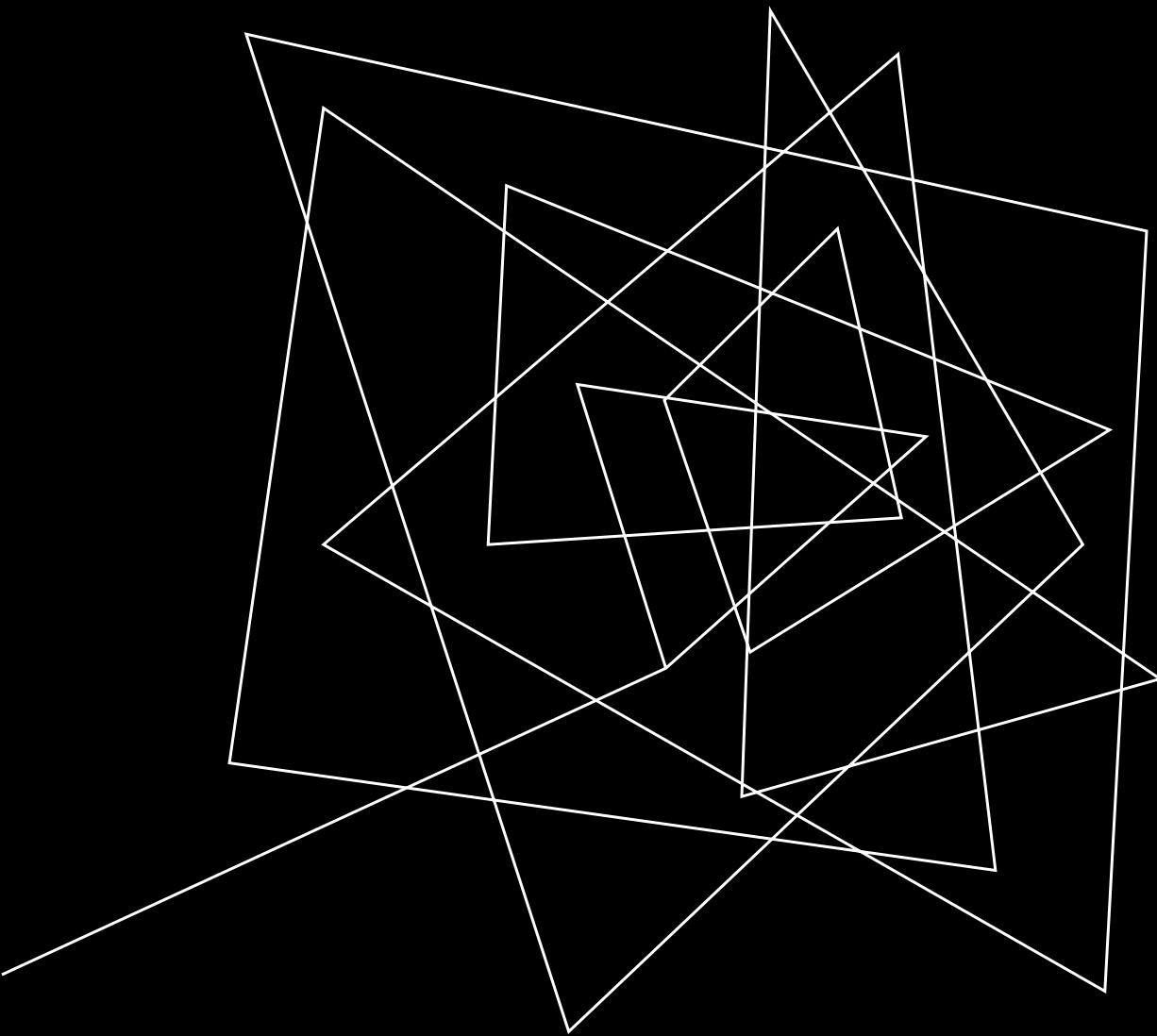


Model Evaluation Metrics for Supervised Learning Models

<DEMO>

Demo: [Session 4 – Feature Engineering and Model Evaluation](#)



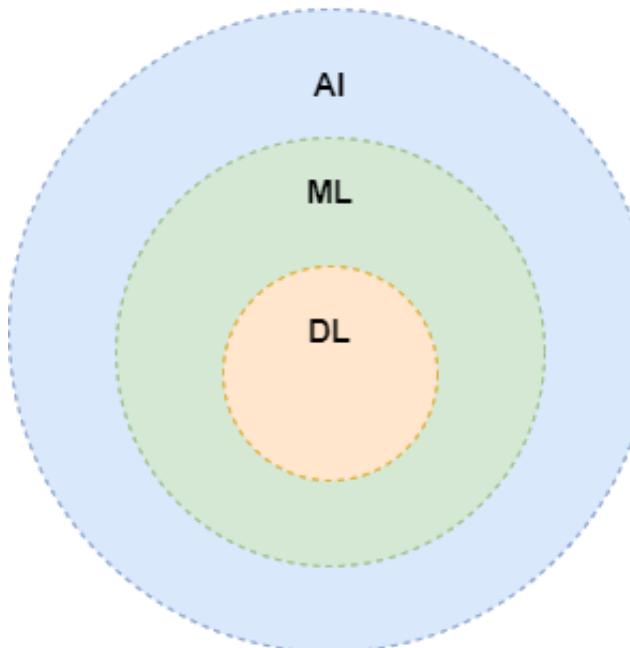


SESSION 5: INTRODUCTION TO DEEP LEARNING

Introduction to deep learning and different types of architectures

Introduction: Why Deep Learning?

The first question someone can ask after knowing about "classical" machine learning (linear regression, logistic regression, decision tree, KNN, ...etc.) is, **why do we need deep learning?**



Artificial Intelligence

Class of problems we can solve when computers think/act like humans

Machine Learning

The science of getting computers to learn without being explicitly programmed. Data + Algorithms

Deep Learning

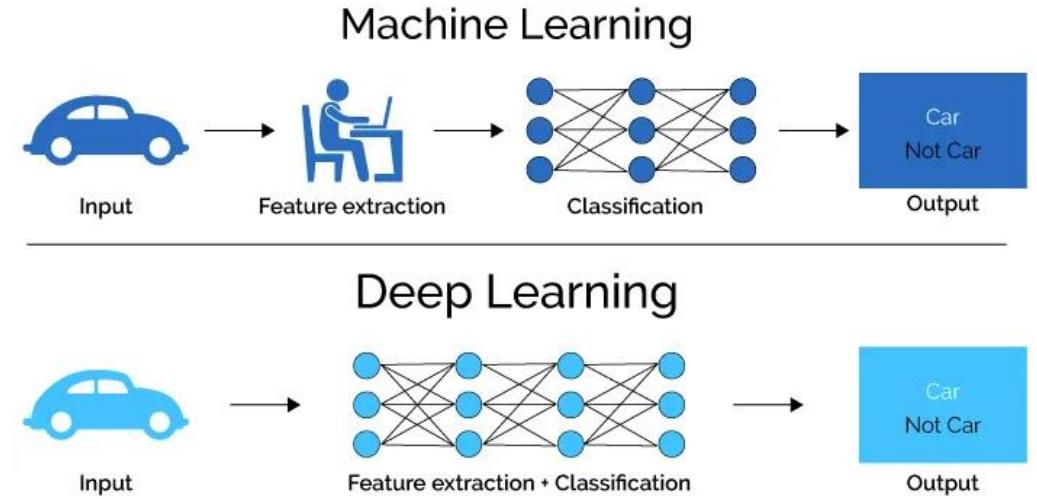
Subset of machine learning that is based on neural networks to mimic the human brain. Generally outperforms classical machine learning on unstructured data

Introduction: Why Deep Learning?

<Automatic Feature Extraction>

Automatic Feature Extraction:

- Deep learning algorithms automatically learn relevant features during training, alleviating the burden of manually selecting and engineering features. This is especially advantageous when dealing with high-dimensional data.
- Feature extraction (aka feature engineering) is the process of putting domain knowledge into the creation of feature extractors to reduce the complexity of the data and make patterns more visible to learning algorithms. This process is difficult and expensive in terms of time and expertise.



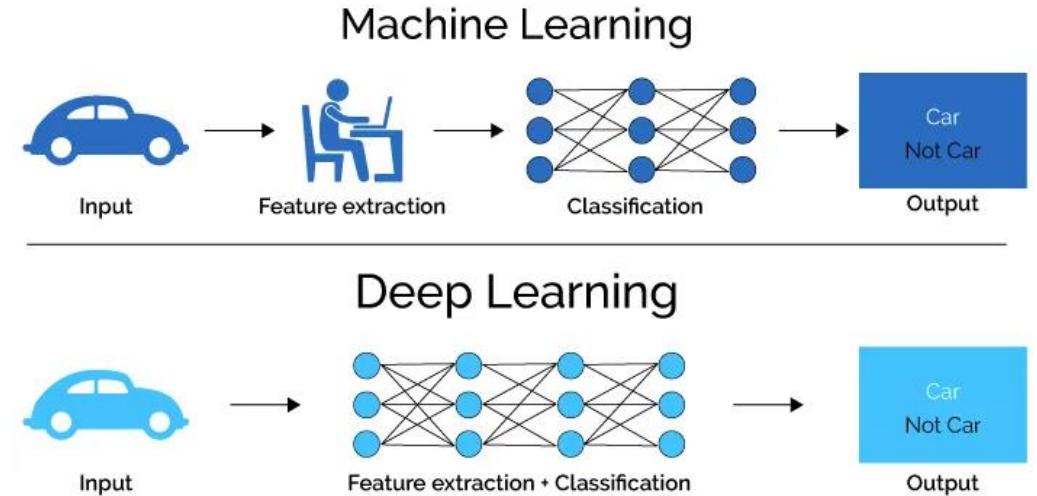
Feature extraction: Machine learning vs Deep learning

Introduction: Why Deep Learning?

<Automatic Feature Extraction>

This is best explained via an example:

- Assume you are building a system that will learn to classify images as either Car or Not Car.
- In classical ML, the algorithmic approach will use data to learn whether the image is Car or Not Car. To help this along, it might have been possible for a human to label constituent features indicative of Car (e.g. wheels) in the images thereby providing extra features with which the system can assess.
- By contrast, a DL solution will also attempt to determine which parts of the image make up the car, e.g. wheels, wing mirrors, headlamps, windscreens etc.
- As a result, DL can reduce the amount of hard coding humans have to apply to define features in datasets. This is the difference between having to label the image as Car vs. Not Car and having to do that plus label other data about the image such as wheels, windscreens, wing mirror etc that indicate Car or Not Car.



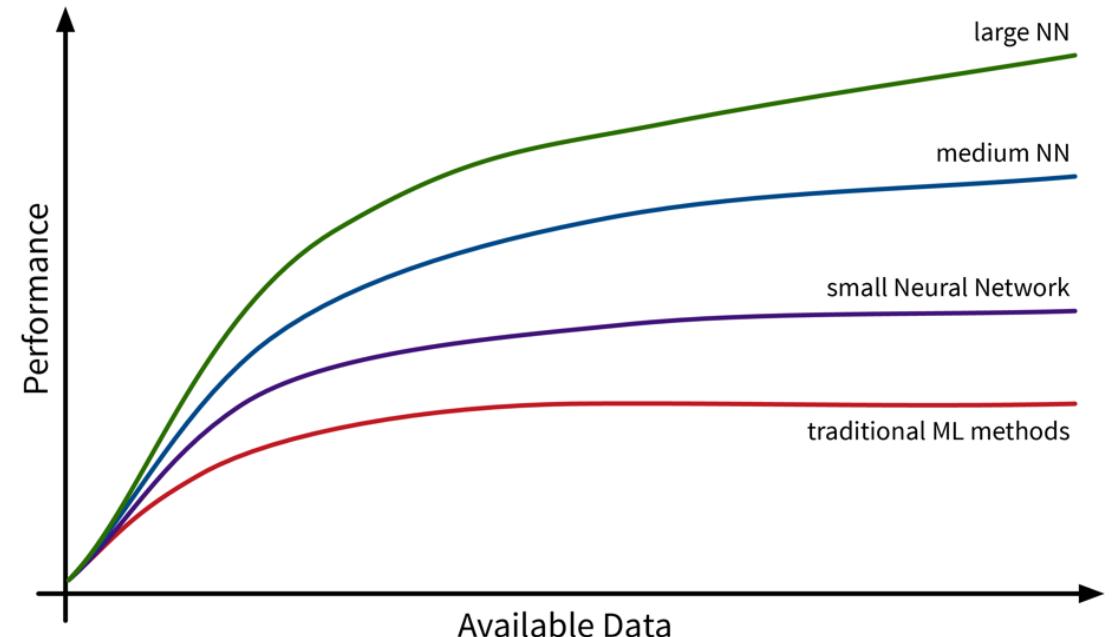
Feature extraction: Machine learning vs Deep learning

Introduction: Why Deep Learning?

<Handling Big Data>

Handling Big Data:

- Deep learning algorithms thrive on large datasets. They can effectively utilize massive amounts of data to generalize well and make accurate predictions, whereas traditional machine learning models might struggle with scalability.
- Another factor driving the spread of ML is the availability of (digital) data. Companies like Google, Amazon, and Meta have had a head start here, as their business model was built around data from the start, but other companies are starting to catch up. While traditional ML models do not benefit much from all this available data, large neural network models with many degrees of freedom can now show their full potential by learning from all the texts and images posted every day on the Internet



Introduction: Why Deep Learning?

- 1. Representation Learning:** Deep learning excels at learning hierarchical representations of data. It automatically learns features from raw data, eliminating the need for manual feature engineering, which is common in traditional machine learning.
- 2. Complex Pattern Recognition:** Deep learning is particularly effective in handling complex patterns and relationships within data. The multiple layers in neural networks enable the model to capture intricate and abstract features.
- 3. Handling Big Data:** Deep learning algorithms thrive on large datasets. They can effectively utilize massive amounts of data to generalize well and make accurate predictions, whereas traditional machine learning models might struggle with scalability.
- 4. Feature Abstraction:** Deep learning models automatically abstract and extract relevant features from the input data. This allows the model to focus on the most important aspects, leading to better generalization.
- 5. Improved Performance in Unstructured Data:** For tasks involving unstructured data like images, audio, and text, deep learning models, especially Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have demonstrated superior performance compared to traditional methods.

Introduction: Why Deep Learning?

1. **End-to-End Learning:** Deep learning models can learn end-to-end, meaning they can take raw input data and produce a final output without the need for intermediate processing steps. This simplifies the modeling process and can lead to more accurate results.
2. **Adaptability to Varied Tasks:** Deep learning models are versatile and can be applied to a wide range of tasks, from image and speech recognition to natural language processing. Their adaptability makes them suitable for a variety of domains.
3. **Automatic Feature Extraction:** Deep learning algorithms automatically learn relevant features during training, alleviating the burden of manually selecting and engineering features. This is especially advantageous when dealing with high-dimensional data.
4. **Continuous Improvement with Data:** Deep learning models can continuously improve with more data. As the amount of available data increases, the model's performance often improves, making it a scalable solution for evolving datasets.
5. **State-of-the-Art Performance:** In many domains, deep learning models have achieved state-of-the-art performance, outperforming traditional machine learning methods. This makes them the go-to choice for cutting-edge applications.

Introduction: Why Deep Learning? <Classical ML vs. Deep Learning>

| Aspect | Classical machine learning | Deep learning |
|---|--|--|
| Data size | Can perform well with smaller datasets. | Often requires large amounts of data for effective training. |
| Feature Engineering | Often requires domain knowledge for manual feature creation. | Automatically learns hierarchical features. |
| Computation | Computationally less intensive compared to deep learning. | Requires powerful hardware, like GPUs, for efficient training. |
| NLP + Computer vision + Audio/Speech recognition | Limited and less effective in these tasks | Highly effective and represents state-of-the-art for these tasks |
| Human-like Learning | Relies on human-engineered features and rules. | Can automatically learn complex patterns like humans. |

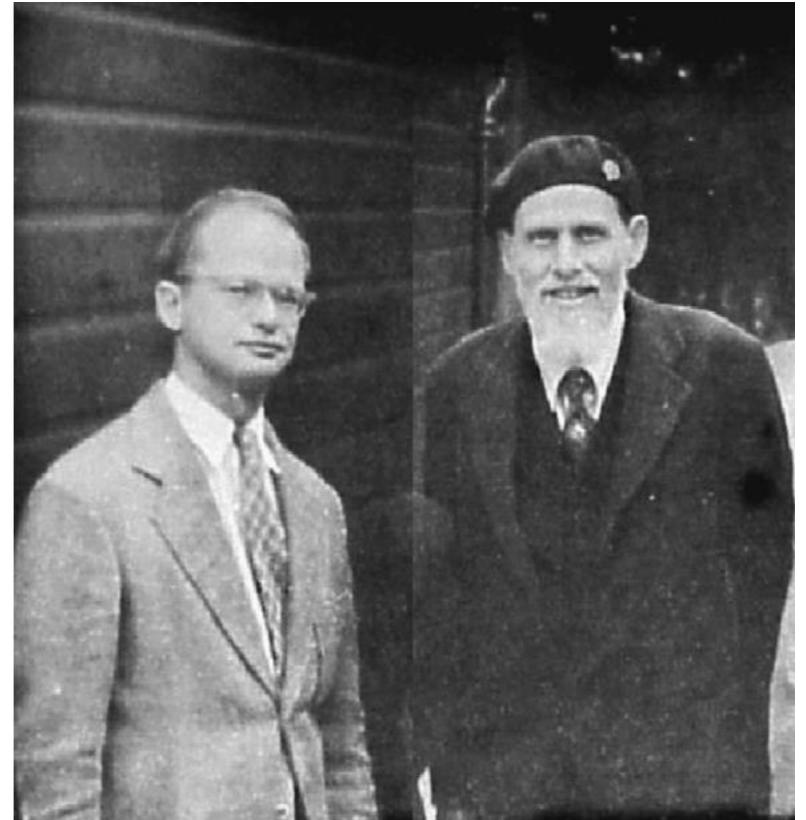
Table. The comparative table between classical machine learning (ML) and deep learning (DL) based on various aspects

History of Deep Learning: 1943: Warren McCulloch and Walter Pitts

McCulloch-Pitts Neuron (1943): The concept of an artificial neuron was first introduced by Warren McCulloch and Walter Pitts. They proposed a mathematical model of a simplified neuron, capable of binary decision-making based on weighted inputs.

Paper title: [A logical calculus of the ideas immanent in nervous activity](#)

In the paper, they explain how they were able to program artificial neurons inspired by the functioning of biological neurons



McCulloch (right) and Pitts (left) in 1949
Image Source: www.semanticscholar.org

History of Deep Learning: 1943: Warren McCulloch and Walter Pitts

Neurons are excitable cells connected to each other and whose role is to transmit information in our nervous system

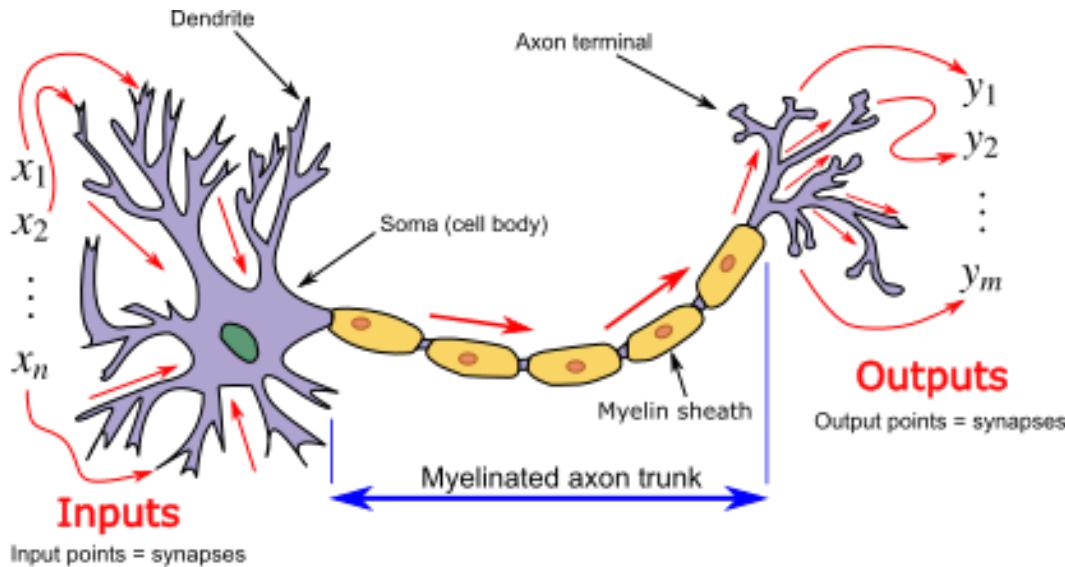


Fig. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal is a short electrical pulse called action potential or 'spike'.

Source: [Wikipedia.org](https://en.wikipedia.org)

History of Deep Learning: 1943: Warren McCulloch and Walter Pitts

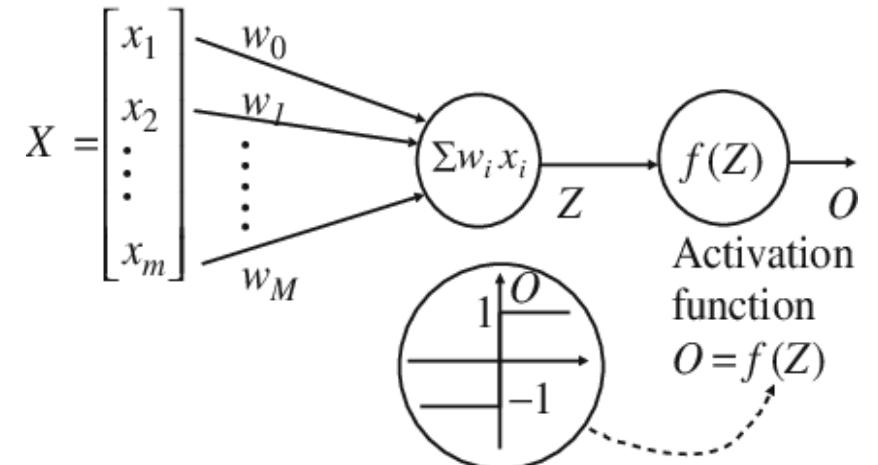
<Threshold Logic Unit>

Mathematical Definition:

McCulloch and Pitts developed a mathematical formulation known as linear threshold gate (or threshold logic unit), which describes the activity of a single neuron with two states, firing or not-firing. In its simplest form, the mathematical formulation is as follows:

$$Sum = \sum_{i=1}^N I_i W_i$$

$$y(Sum) = \begin{cases} 1, & \text{if } Sum \geq T \\ 0, & \text{otherwise} \end{cases}$$



History of Deep Learning: 1943: Warren McCulloch and Walter Pitts

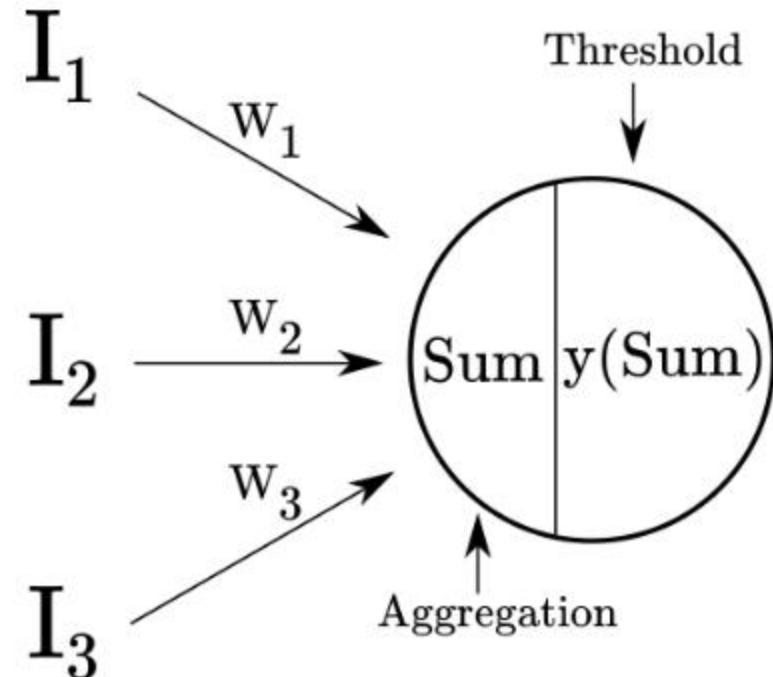
<Threshold Logic Unit>

Mathematical Definition:

Where I_1, I_2, \dots, I_N are binary input values $\in \{0,1\}$; W_1, W_2, \dots, W_N are weights associated with each input $\in \{-1,1\}$; Sum is the weighted sum of inputs; and T is a predefined threshold value for the neuron activation (i.e., firing).

$$Sum = \sum_{i=1}^N I_i W_i$$

$$y(Sum) = \begin{cases} 1, & \text{if } Sum \geq T \\ 0, & \text{otherwise} \end{cases}$$



History of Deep Learning: 1943: Warren McCulloch and Walter Pitts

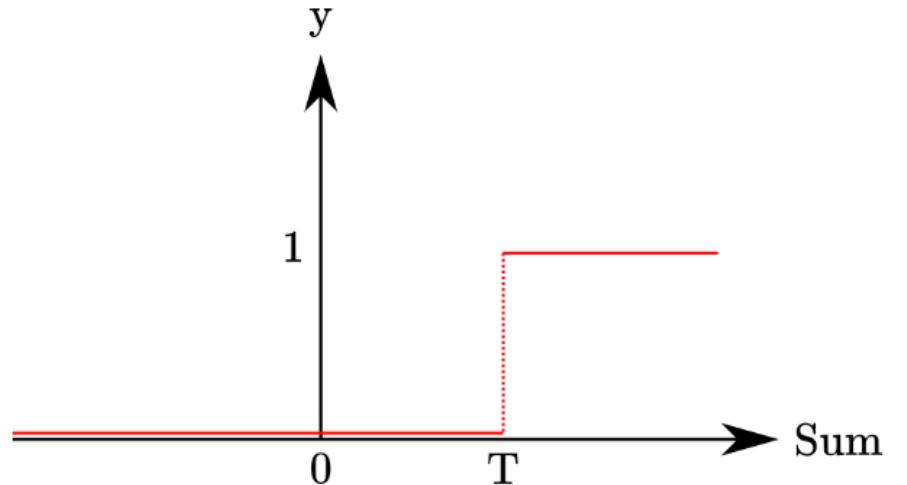
<Threshold Logic Unit>

Mathematical Definition:

An input is considered **excitatory** when its contribution to the weighted sum is **positive**, for instance, $I_1 \cdot W_1 = 1 \cdot 1 = 1$; whereas an input is considered **inhibitory** when its contribution to the weighted sum is **negative**, for instance, $I_1 \cdot W_1 = 1 \cdot -1 = -1$. If the value of Sum is $\geq T$, the neuron fires, otherwise, it does not.

$$Sum = \sum_{i=1}^N I_i W_i$$

$$y(Sum) = \begin{cases} 1, & \text{if } Sum \geq T \\ 0, & \text{otherwise} \end{cases}$$

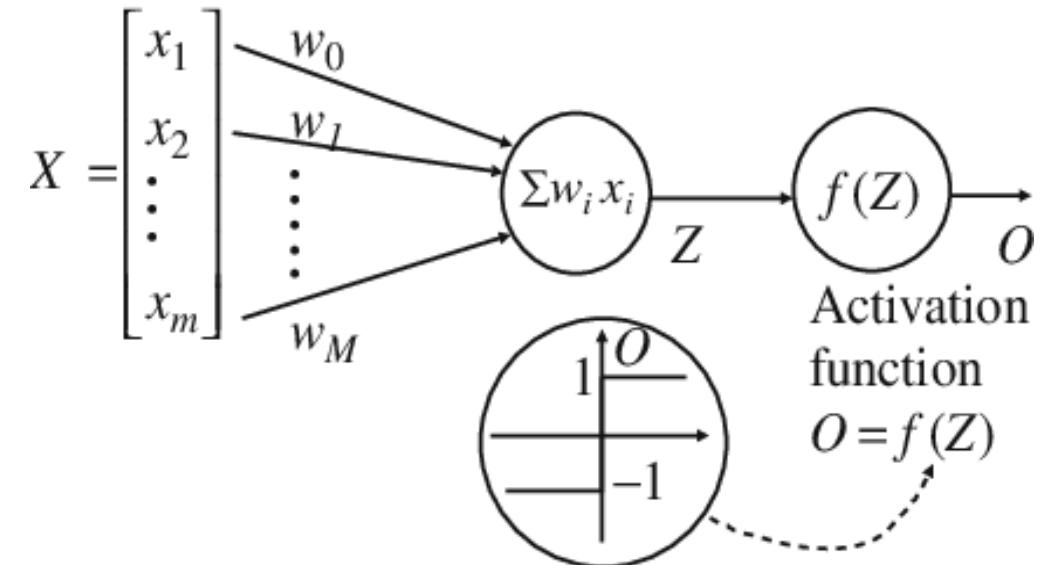


This is known as a step-function, where the y-axis encodes the activation-state of the neuron, and the x-axis encodes the output of the weighted sum of inputs.

History of Deep Learning: 1943: Warren McCulloch and Walter Pitts

<Threshold Logic Unit>

In the McCulloch-Pitts neuron model, the weights are typically set manually by the designer or researcher. The weights represent the strength of the connections between the inputs and the neuron. Each input is multiplied by its corresponding weight, and the sum of these weighted inputs is compared to a threshold. Setting the weights involves some trial and error or knowledge of the problem domain. The choice of weights determines the neuron's sensitivity to each input feature. Adjusting the weights allows you to influence how much influence each input has on the neuron's decision.



History of Deep Learning: 1957: Frank Rosenblatt's Perceptron

Frank Rosenblatt's perceptron is a type of artificial neural network and a fundamental concept in the history of artificial intelligence and machine learning. It was introduced by Frank Rosenblatt in 1957. The perceptron is a simple algorithm designed for binary classification problems.

Who is Frank Rosenblatt? Frank Rosenblatt (July 11, 1928 – July 11, 1971) was an American psychologist notable in the field of artificial intelligence. He is sometimes called the father of deep learning for his pioneering work on neural networks.

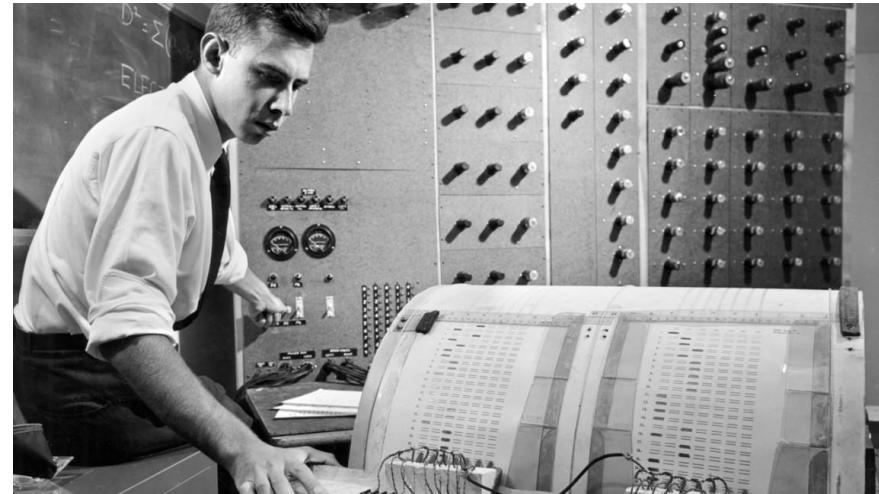


Fig. Frank Rosenblatt '50, Ph.D. '56, works on the “perceptron” – what he described as the first machine “capable of having an original idea.”

Source: News.cornell.edu

History of Deep Learning: 1957: Frank Rosenblatt's Perceptron

Here are the basic components and working principles of the perceptron:

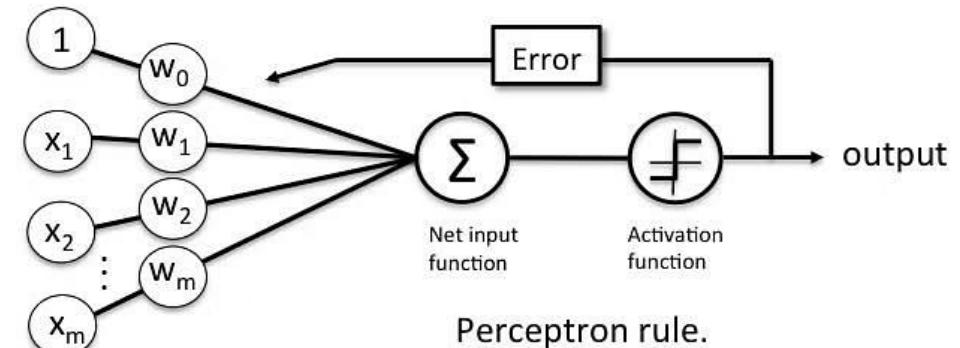
Inputs: The perceptron takes multiple binary inputs (0 or 1). Each input is multiplied by a corresponding weight.

Weights: Each input is associated with a weight, which is a parameter that the algorithm learns during training. The weights determine the strength of the influence of the corresponding input.

Summation: The weighted inputs are summed up, and a bias term is added. The bias is another parameter that the perceptron learns during training.

Activation Function: The summed value is then passed through an activation function. The traditional perceptron uses a step function as its activation function. If the result is above a certain threshold, the perceptron outputs 1; otherwise, it outputs 0.

Learning: The perceptron is trained using a supervised learning approach. During training, it adjusts its weights and bias based on the error in its predictions compared to the true output. This process is often referred to as the perceptron learning rule or the delta rule.



History of Deep Learning: 1957: Frank Rosenblatt's Perceptron

The perceptron learning rule, also known as the perceptron update rule or perceptron training rule, is used to adjust the weights and bias of a perceptron during the training process. The goal is to minimize the error in the perceptron's predictions. The formula for updating the weights and bias for a single training example is as follows:

For each weight w_i and the bias b , the update rule is:

$$w_i \leftarrow w_i + \text{learning rate} \times (\text{target} - \text{prediction}) \times \text{input}_i$$

Here:

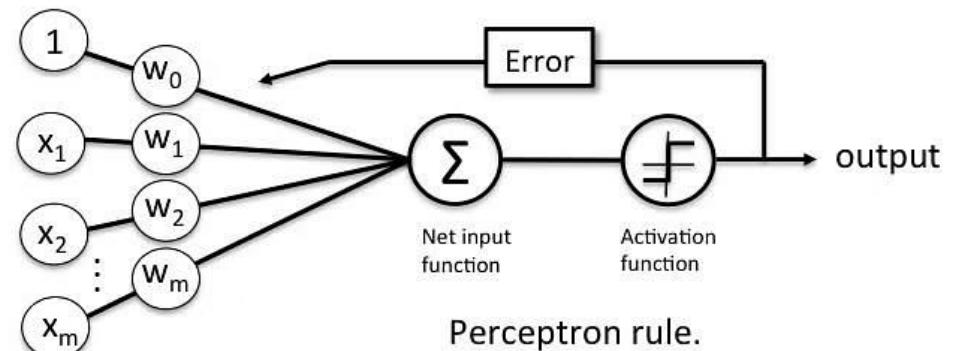
w_i is the weight associated with the i -th input feature.

b is the bias.

The learning rate is a hyperparameter that determines the step size of the update.

The target is the true class label for the training example.

The prediction is the output of the perceptron for the given input.



McCulloch-Pitts Artificial Neuron Model and Rosenblatt's Perceptron <DEMO>

Demo: [Session 5 – Introduction to Deep Learning](#)



[Perceptron Research from the 50's & 60's, clip](#)



[The Man who forever changed Artificial Intelligence](#)

Rosenblatt's Perceptron: The New York Times Reported;

WASHINGTON, July 7, 1958 (UPI) -- "The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

Link: nytimes.com

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo
 of Computer Designed to
 Read and Grow Wiser

WASHINGTON, July 7 (UPI) — The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's \$2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of \$100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human be-

ings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

1958 New York
 Times...

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000 responsive cells, including 100,000,000 connections with the eyes.

Rosenblatt's Perceptron: XOR Problem

The XOR problem and its implications for perceptrons were first highlighted by Marvin Minsky and Seymour Papert in their book "Perceptrons," published in 1969. Minsky and Papert demonstrated that a single-layer perceptron (with a linear activation function) was unable to learn and represent certain types of logical functions, specifically those that were not linearly separable. The XOR function is a classic example of a logical operation that cannot be represented by a single-layer perceptron.

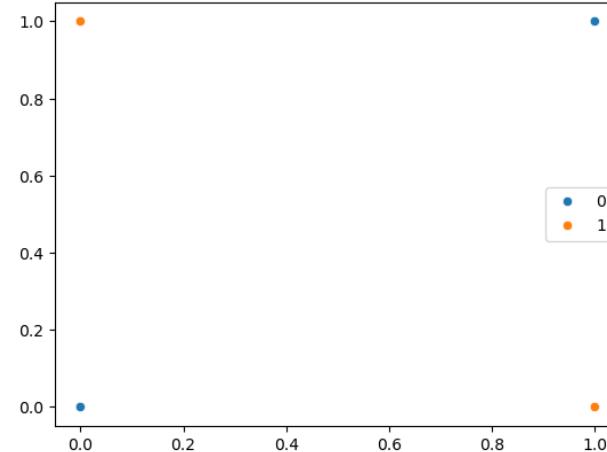
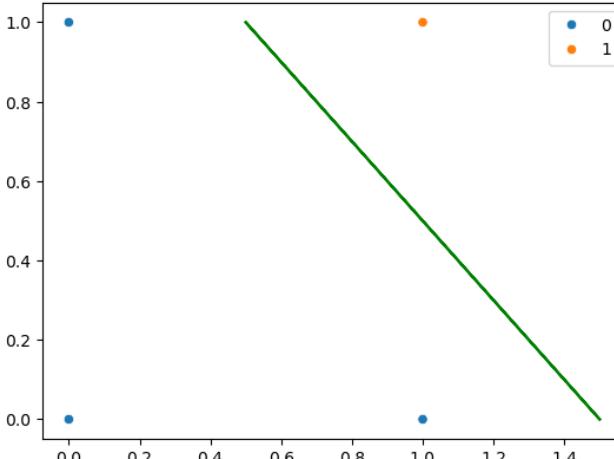
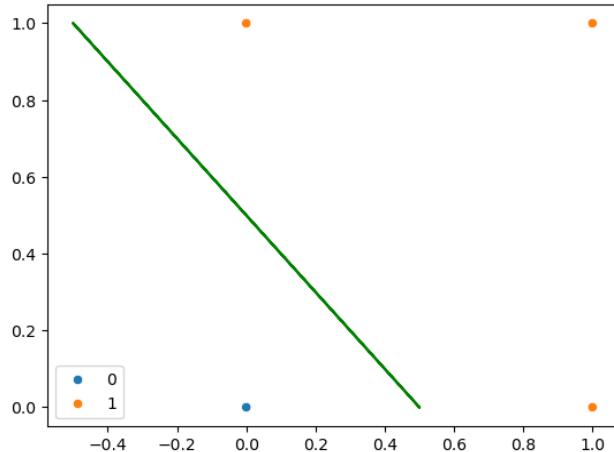


Fig. Marvin Minsky (left) and Seymour Papert (right) in 1971.

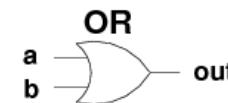
Source: npr.org

Rosenblatt's Perceptron: XOR Problem

The XOR (exclusive OR) function takes two binary inputs and outputs 1 if the inputs are different and 0 if they are the same. The problem with representing XOR using a single-layer perceptron arises because the decision boundary for XOR is not a straight line; it requires a non-linear decision boundary.



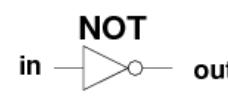
| a | b | out |
|---|---|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



| a | b | out |
|---|---|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



| a | b | out |
|---|---|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



| in | out |
|----|-----|
| 0 | 1 |
| 1 | 0 |

Rosenblatt's Perceptron: XOR Problem

Minsky and Papert's work was influential because it highlighted the limitations of the perceptron model and led to a period of reduced interest (**dark age of AI** also called **AI winter**) in neural networks during the 1970s and 1980s. It wasn't until the development of multi-layer perceptrons and the backpropagation algorithm in the 1980s that neural networks regained popularity and were shown to be capable of learning more complex, non-linear patterns, including XOR.



Fig. Marvin Minsky (left) and Seymour Papert (right) in 1971.

Source: npr.org

Rosenblatt's Perceptron: XOR Problem

<What is The Solution?>

Well, the solution is to use multi-layer perceptron instead of a single perceptron

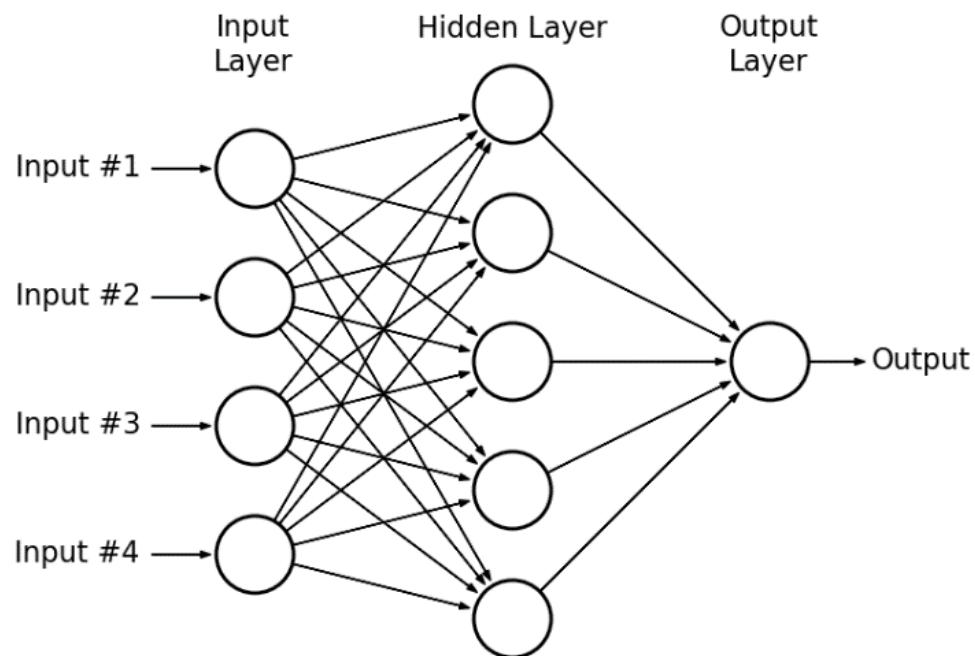
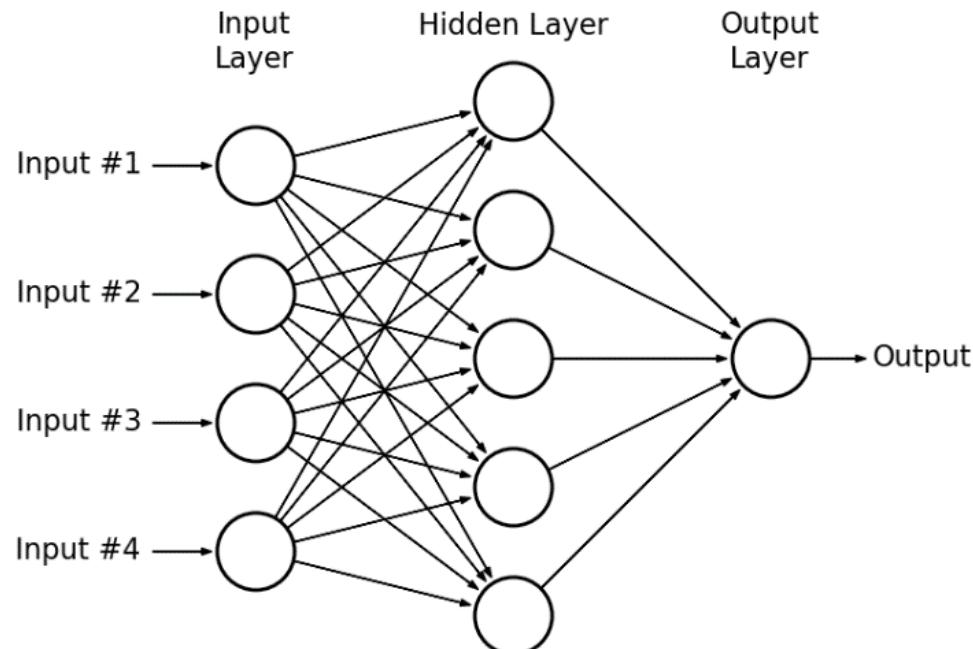


Fig. Marvin Minsky (left) and Seymour Papert (right) in 1971.

Source: npr.org

Multi-Layer Perceptron (MLP): Another Challenge!

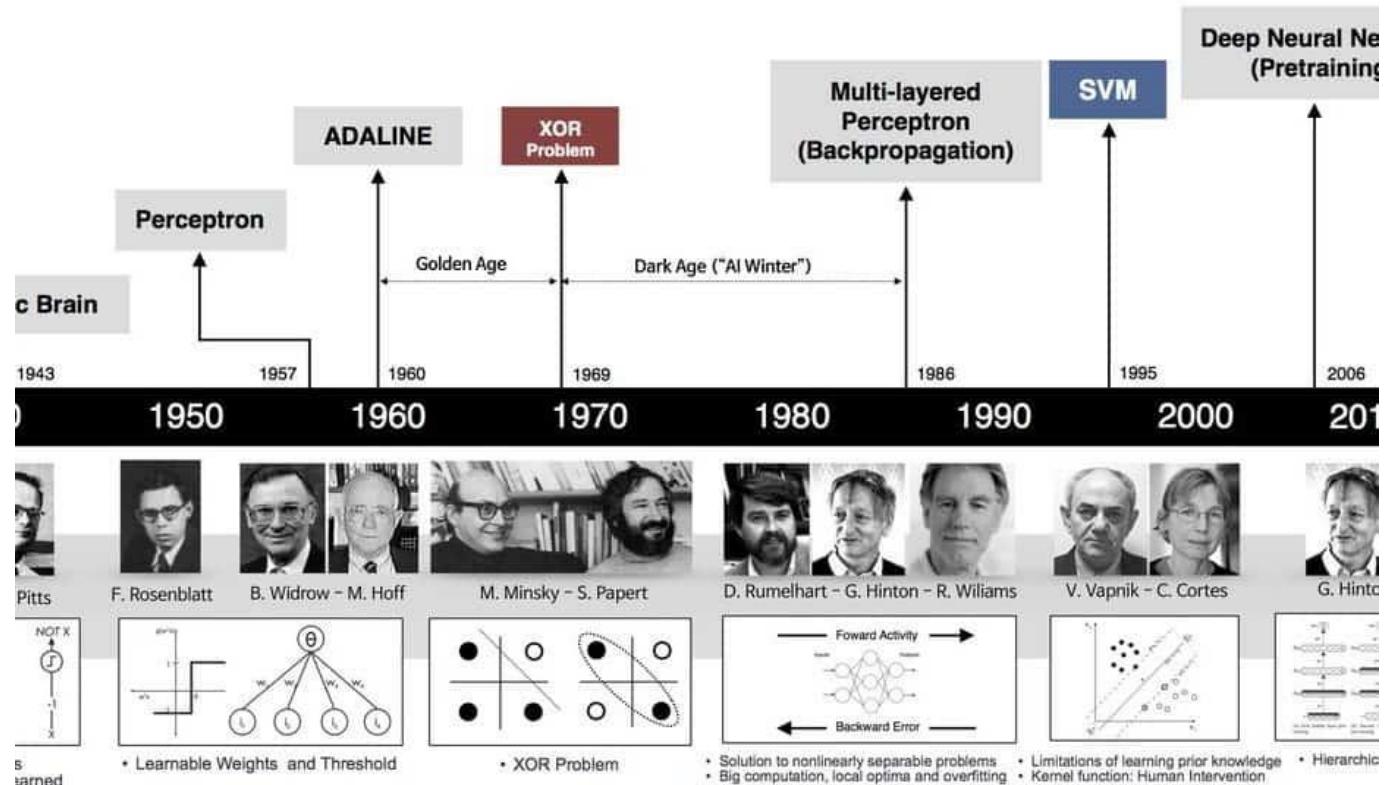
At that time, there was no training algorithm
that can be used for such architecture!



Multi-Layer Perceptron (MLP): Problem Solved and Beginning of New Era!

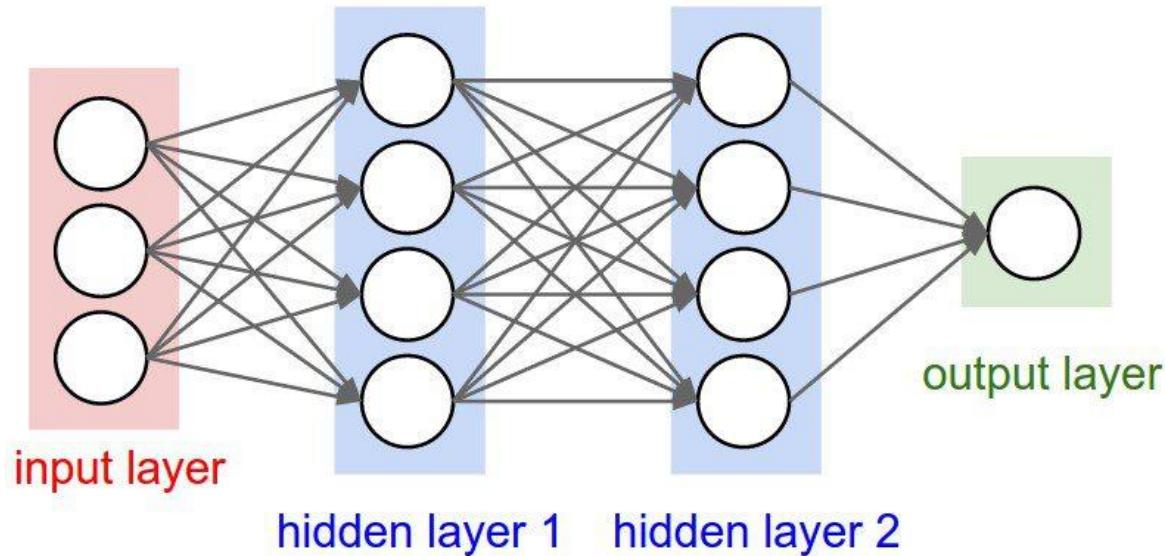
It took almost 17 years (1969 to 1986) until a training algorithm was re-discovered, known as back-propagation.

Paper Link: [Learning representations by back-propagating errors](#)



Multi-Layer Perceptron (MLP)

Like in the human brain, the power and robustness of the biological neurons is when they are fully connected to each other, it starts to become more and more complex by adding/stacking more and more hidden layers, at this level we are talking about **Multi-layer perceptron. (DEEP LEARNING)**



Multi-Layer Perceptron (MLP): <DEMO>

Demo: [Session 5 - Introduction To Deep Learning](#)



Advanced Neural Network Architectures: Convolutional Neural Network (CNN)

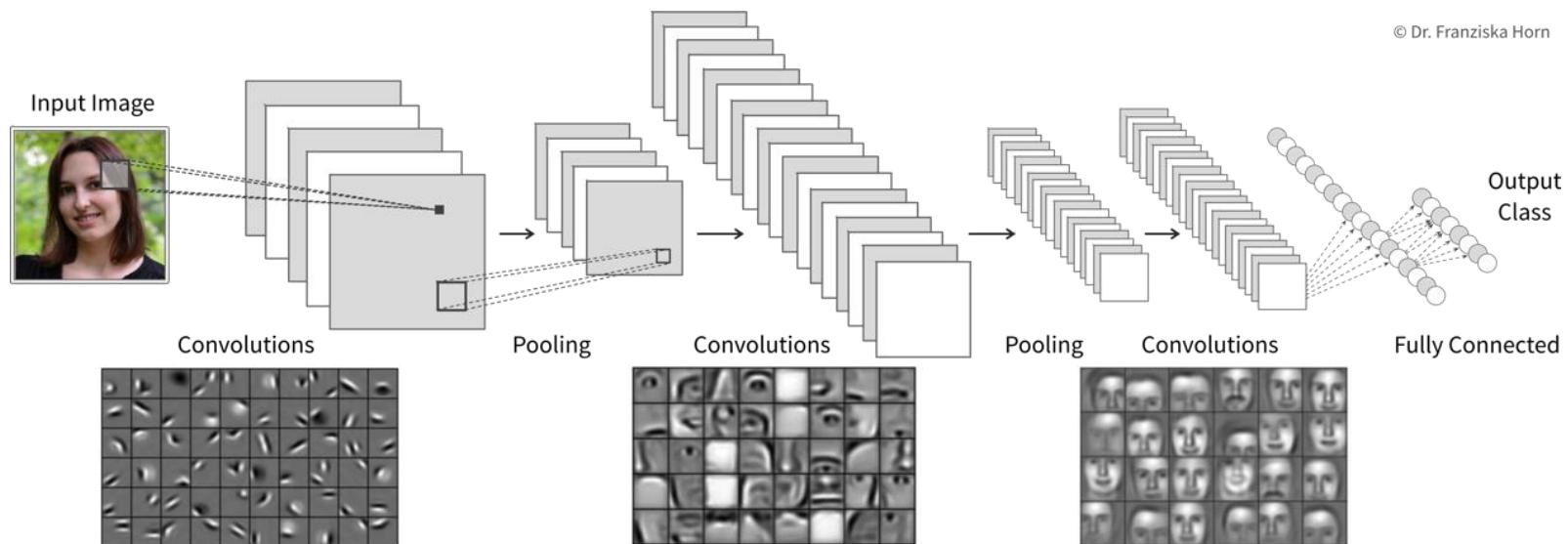
Manual feature engineering for computer vision tasks is incredibly difficult. While humans recognize a multitude of objects in images without effort, it is hard to describe why we can identify what we see, e.g., which features allow us to distinguish a cat from a small dog. Deep learning had its first breakthrough success in this field, because neural networks, in particular CNNs, manage to learn meaningful feature representations of visual information through a hierarchy of layers.



Source: [istockphoto.com](https://www.istockphoto.com)

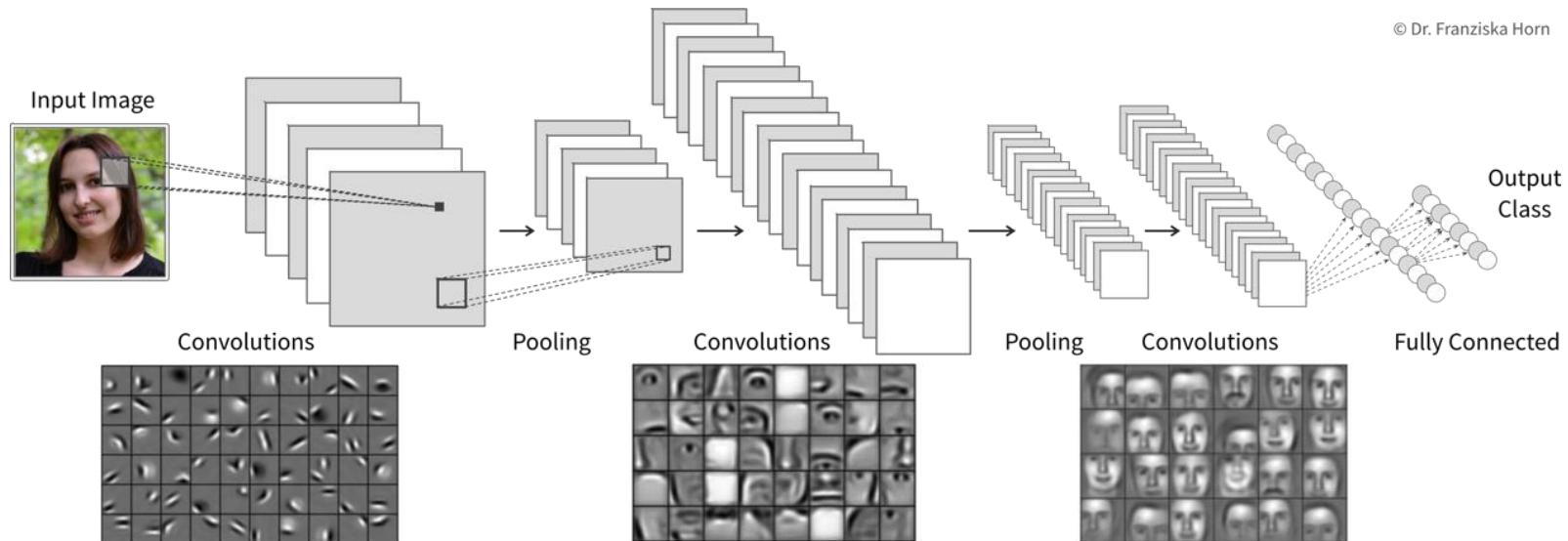
Advanced Neural Network Architectures: Convolutional Neural Network (CNN)

Convolutional neural networks are very well suited for processing visual information, because they can operate on the 2D images directly and do not need the input to be flattened into a vector. Furthermore, they utilize the fact that images are composed of a lot of local information (e.g., eyes, nose, and mouth are all localized components of a face).

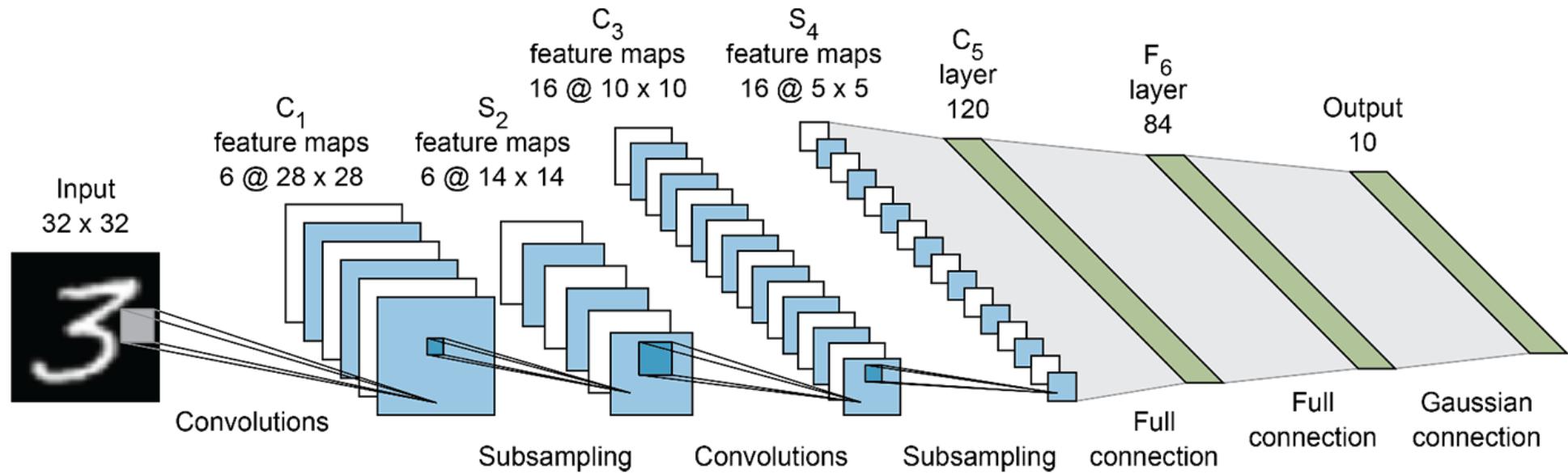


Advanced Neural Network Architectures: Convolutional Neural Network (CNN)

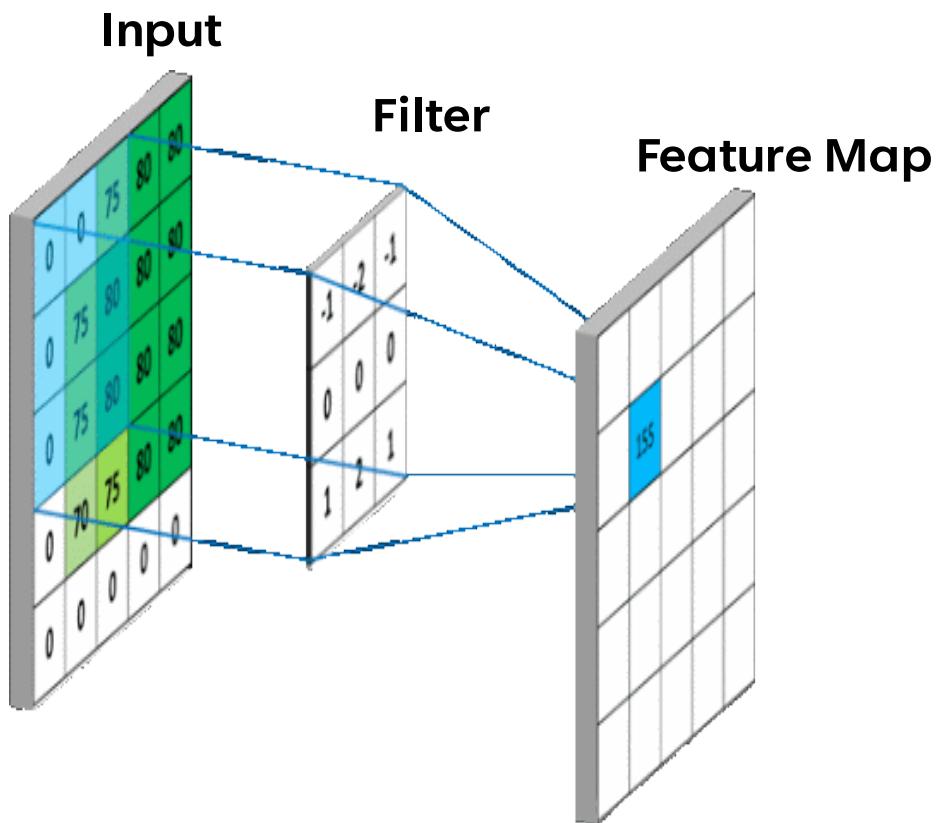
Compared to the dense / fully-connected layers in FFNNs, which consist of one huge matrix mapping from one layer to the next, the filter patches used in convolutional layers are very small, i.e., there are less parameters that need to be learned. Furthermore, the fact that the filters are applied at every position in the image has a regularizing effect, since the filters need to be general enough capture relevant information in multiple areas of the images.



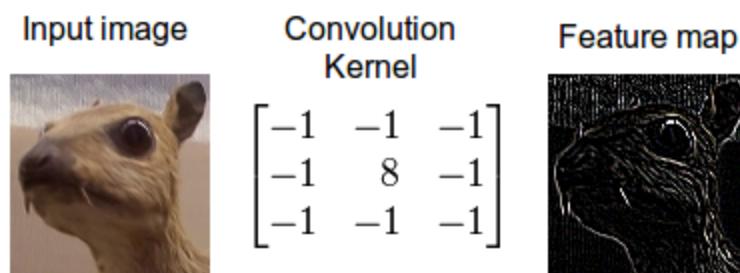
Convolutional Neural Networks (CNNs)



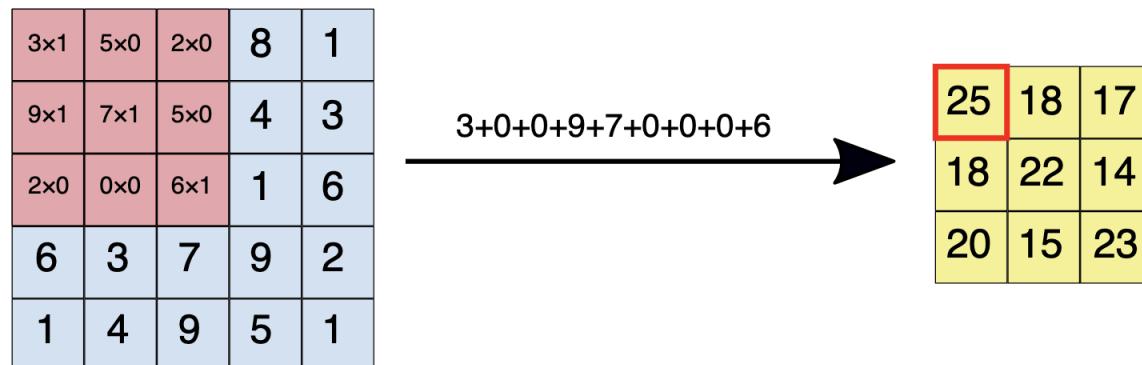
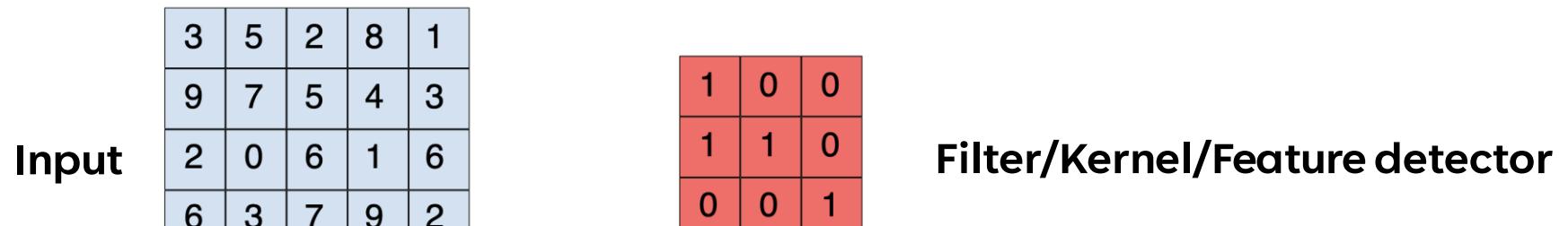
Convolutional Neural Networks (CNNs)



- The term convolution refers to **the mathematical combination of two functions to produce a third function**. It merges two sets of information. In the case of a CNN, the convolution is performed on the input data with the use of a filter or kernel (these terms are used interchangeably) to then produce a feature map.
- It is a very important technique to find patterns in images and image processing



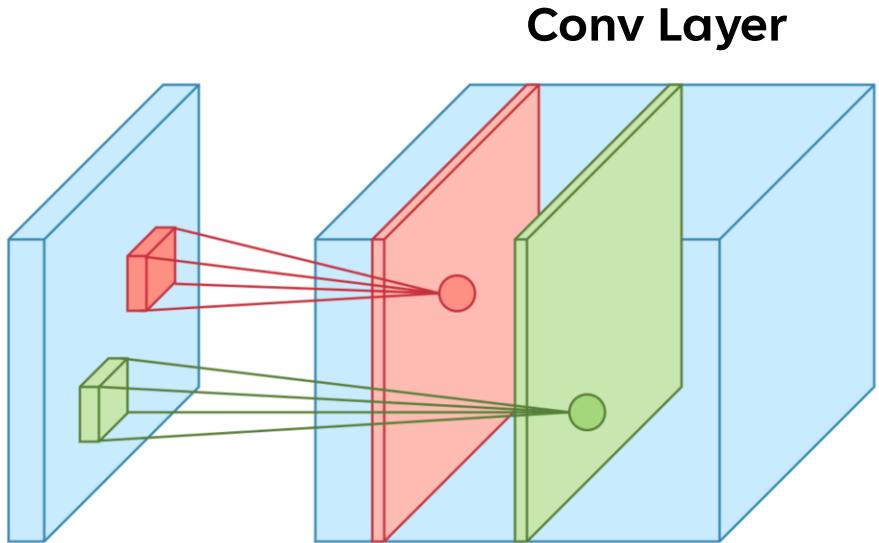
Convolutional Neural Networks (CNNs): Filter



Result: Feature map

Note: Feature map output size = [size of the input - size of the kernel/filter] + 1

Convolutional Neural Networks (CNNs): ConvLayers

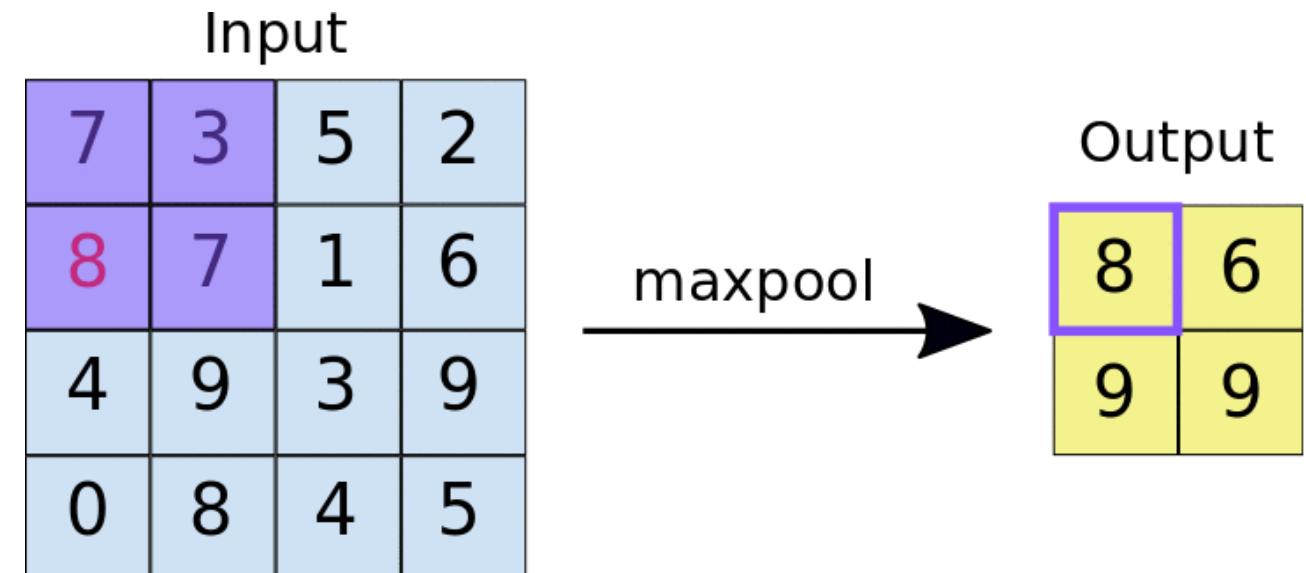


- Convolutional neural networks are a type of neural network in which we are trying to **find the filters** and **biases** that **minimize** the **loss function**.
- **Conv Layer** contains different filters/kernels with different lengths and widths.
- The filters are **computed** by the **CNN**, and that is what makes the **CNN** more **powerful**.

Convolutional Neural Networks (CNNs): Pooling

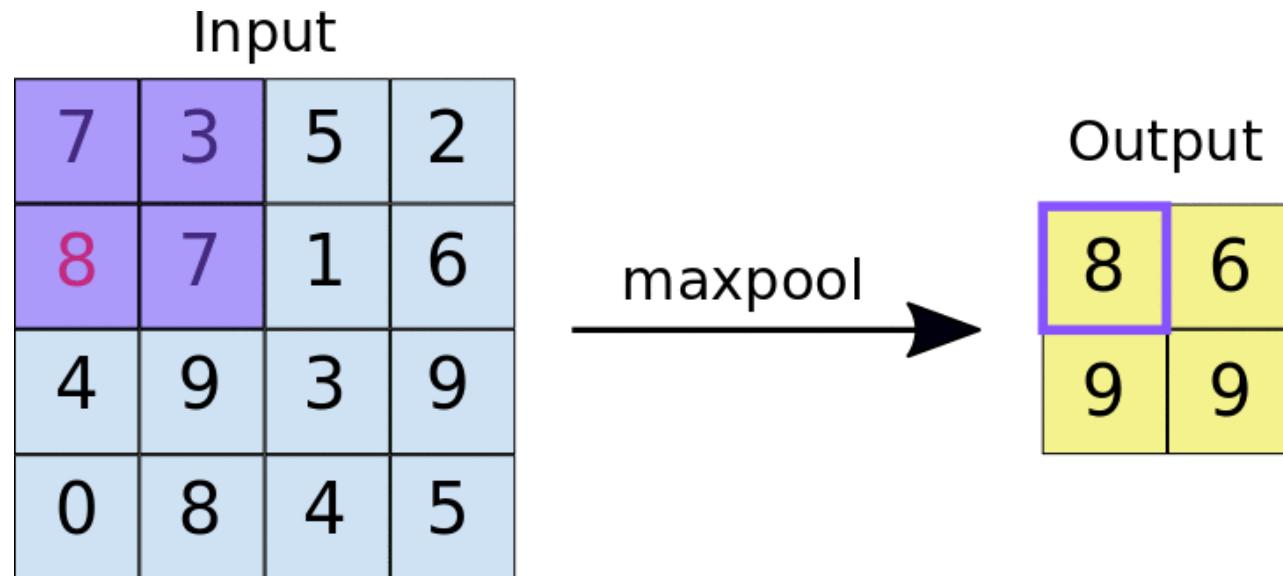
Some used techniques in ConvNets: Pooling

- **Pooling** is required to down sample the detection of features in feature maps.
- **Pooling layers** provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map.
- Two common pooling methods are **average pooling** and **max pooling**



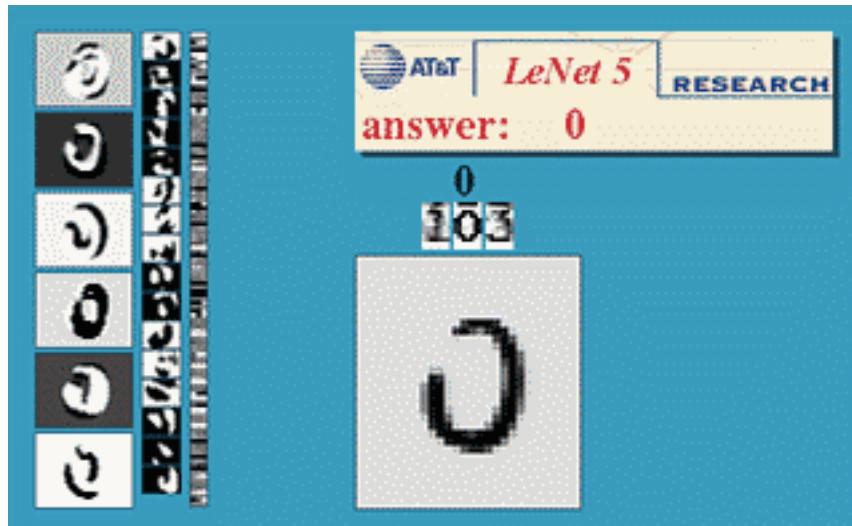
Convolutional Neural Networks (CNNs): MaxPooling

Some used techniques in ConvNets: MaxPooling



Convolutional Neural Networks (CNNs): LeNet5

LeNet-5 is a convolutional neural network architecture designed for handwritten digit recognition. It was introduced by **Yann LeCun**, Léon Bottou, Yoshua Bengio, and Patrick Haffner in the paper titled "**Gradient-Based Learning Applied to Document Recognition**" published in **1998**.



LeNet5 (1998)



Yann LeCun

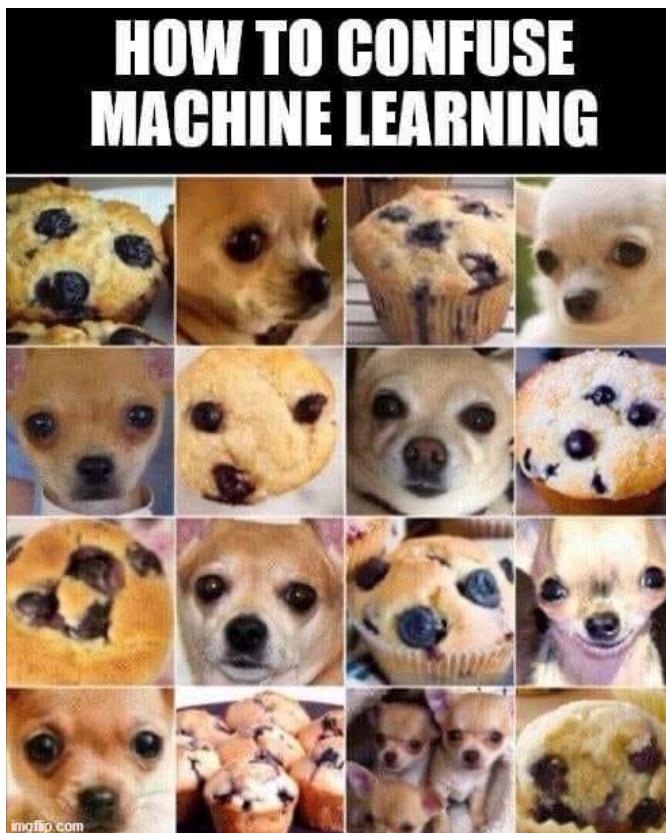
Convolutional Neural Networks (CNNs): LeNet5

LeNet5 Architecture(1998)

1. **Convolutional Layers:** LeNet-5 consists of seven layers, including two convolutional layers. The first convolutional layer applies six filters with a 5×5 receptive field, followed by a 2×2 max-pooling operation.
2. **Activation Functions:** In LeNet-5, the hyperbolic tangent function (\tanh) is used as the activation function.
3. **Subsampling Layers:** Following the first convolutional layer, LeNet-5 has a subsampling layer (max-pooling) to reduce the spatial dimensions of the feature maps.
4. **Second Convolutional Layer:** The second convolutional layer applies sixteen 5×5 filters to the subsampled feature maps from the first convolutional layer. This is followed by another 2×2 max-pooling operation.
5. **Fully Connected Layers:** After the convolutional layers, there are three fully connected layers. The first fully connected layer has 120 units, the second has 84 units, and the final output layer has 10 units (corresponding to the 10 possible digits).
6. **Flatten Operation:** Between the convolutional and fully connected layers, there is a flatten operation to convert the 3D volume output from the convolutional layers into a 1D vector.
7. **Softmax Activation:** The output layer uses a softmax activation function, which turns the raw scores (logits) into class probabilities.

Convolutional Neural Networks (CNNs): <DEMO>

Demo: [Session 5 - Introduction To Deep Learning](#)



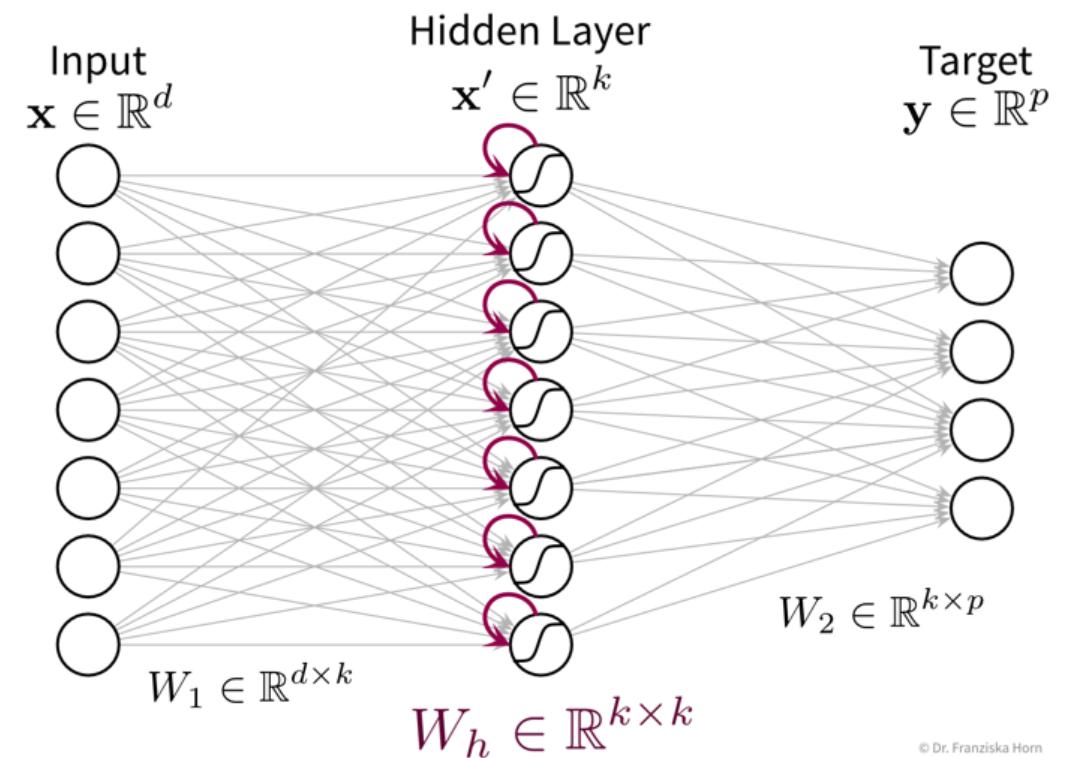
Advanced Neural Network Architectures: Recurrent Neural Network (RNN)

Suited for Sequential Data

Recurrent Neural Networks (RNNs) are a class of neural networks uniquely designed to handle sequential data, making them powerful tools for tasks such as time series analysis, natural language processing, and speech recognition.

Recurrent neural networks are great for sequential data such as time series data or text (i.e., a sequence of words).

In its simplest form, a RNN is like a FFNN, but with additional recurrent connections W_h in the hidden layer to create a memory of the past:



© Dr. Franziska Horn

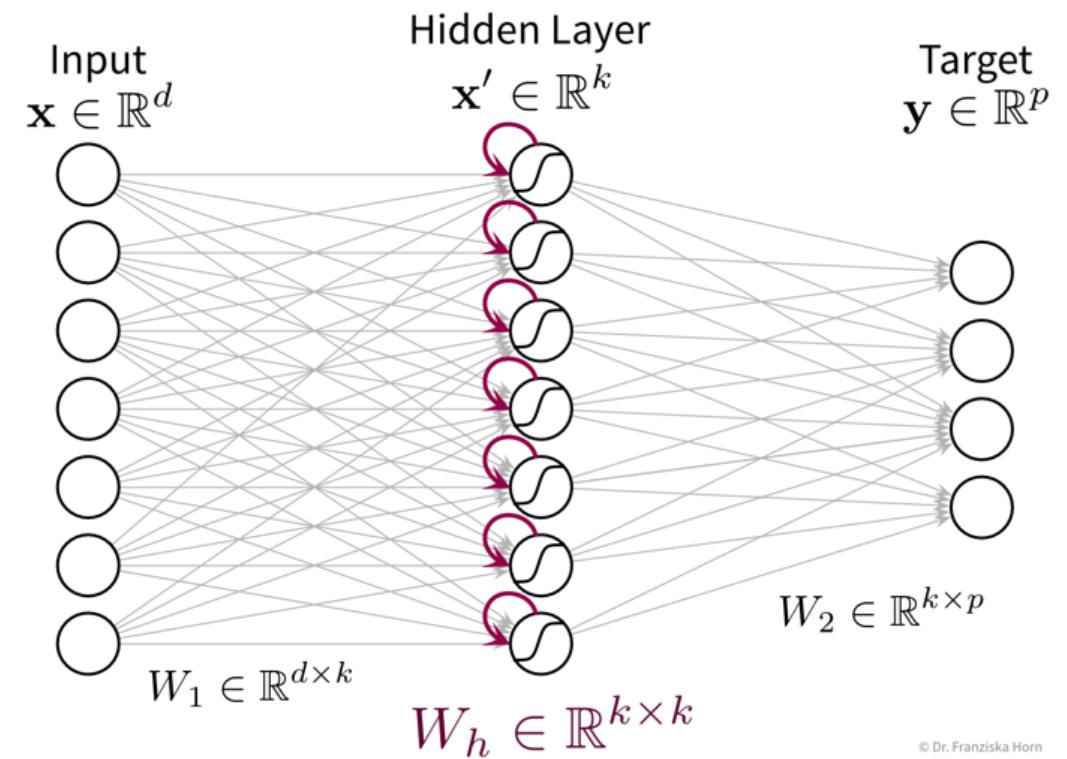
Advanced Neural Network Architectures: Recurrent Neural Network (RNN)

Definition:

A Recurrent Neural Network (RNN) is a type of neural network architecture designed to handle sequential data by maintaining internal memory. Unlike traditional feedforward networks, RNNs have connections that loop back on themselves, allowing them to maintain a sense of context or memory across input sequences.

Processing Sequential Data:

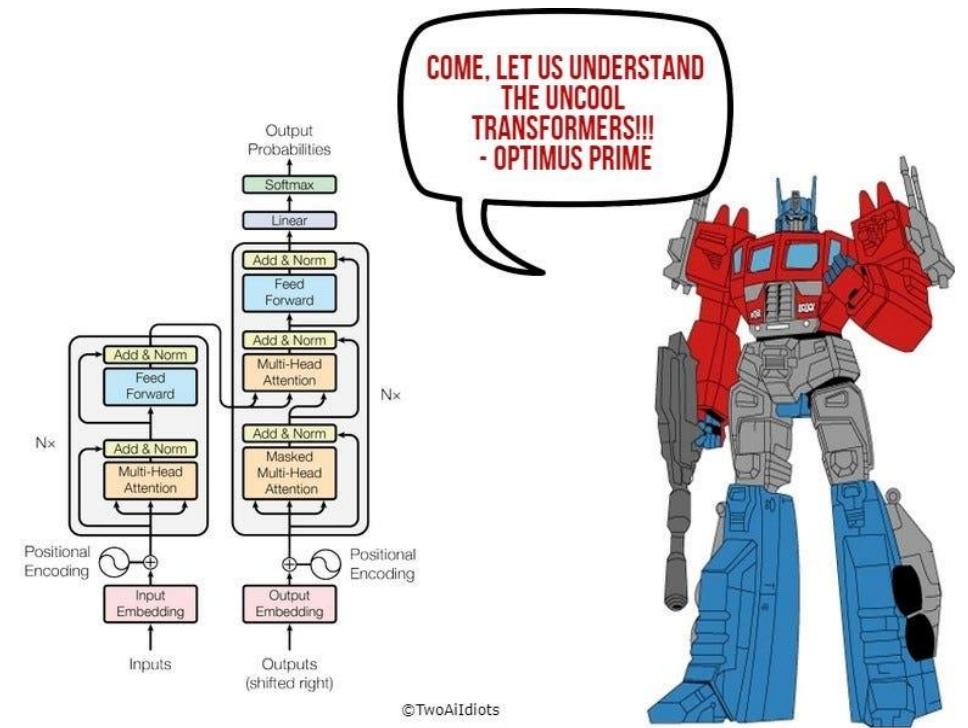
Key Feature: The defining feature of RNNs is their ability to process sequential data. This is achieved by allowing information to persist in the network's hidden state, which is updated at each time step. This internal memory enables RNNs to capture dependencies and patterns in sequential information.



© Dr. Franziska Horn

Advanced Neural Network Architectures: Transformers

Transformers are a type of deep learning model architecture that has gained significant popularity and success in various natural language processing (NLP) tasks. They were introduced in the paper "[Attention is All You Need](#)" by Vaswani et al. in 2017. Since then, transformers have become a cornerstone in the field of deep learning and have been extended to various domains beyond NLP.



Advanced Neural Network Architectures: Transformers

Key components of transformers include:

Self-Attention Mechanism:

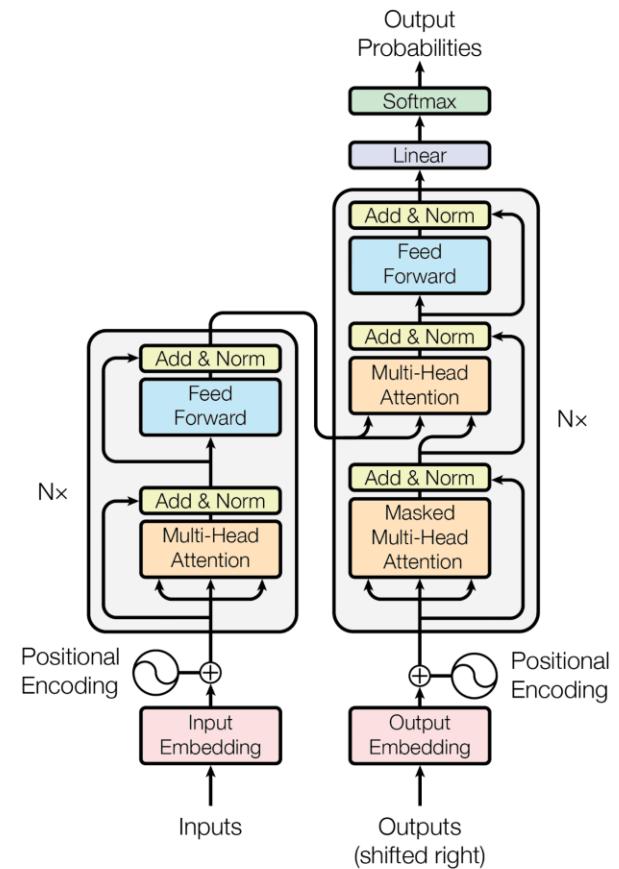
The self-attention mechanism allows the model to weigh the importance of different words in a sequence when processing each word. This helps in capturing long-range dependencies in the data.

Multi-Head Attention:

Transformers use multiple self-attention mechanisms, called attention heads, in parallel. This enables the model to capture different aspects of the relationships between words.

Positional Encoding:

Since transformers don't inherently understand the order of elements in a sequence, positional encodings are added to the input embeddings to provide information about the positions of words in a sentence.



Advanced Neural Network Architectures: Transformers

Key components of transformers include:

Encoder-Decoder Architecture:

Transformers are commonly used in sequence-to-sequence tasks where the input and output sequences can be of different lengths. The model consists of an encoder and a decoder, with each having its own stack of layers.

Feedforward Neural Networks:

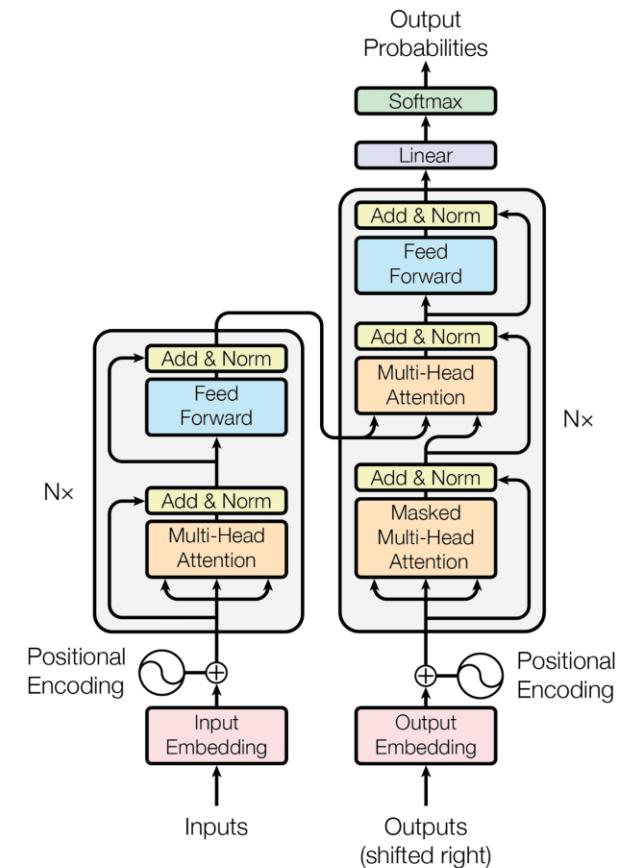
After the self-attention mechanism, there is a feedforward neural network for further processing and non-linearity.

Layer Normalization and Residual Connections:

Each sub-layer (like self-attention or feedforward layer) is followed by layer normalization and connected with a residual connection. This helps in stabilizing the training process.

Attention Masking:

In some applications, attention masking is used to prevent the model from attending to future tokens in the sequence during training.

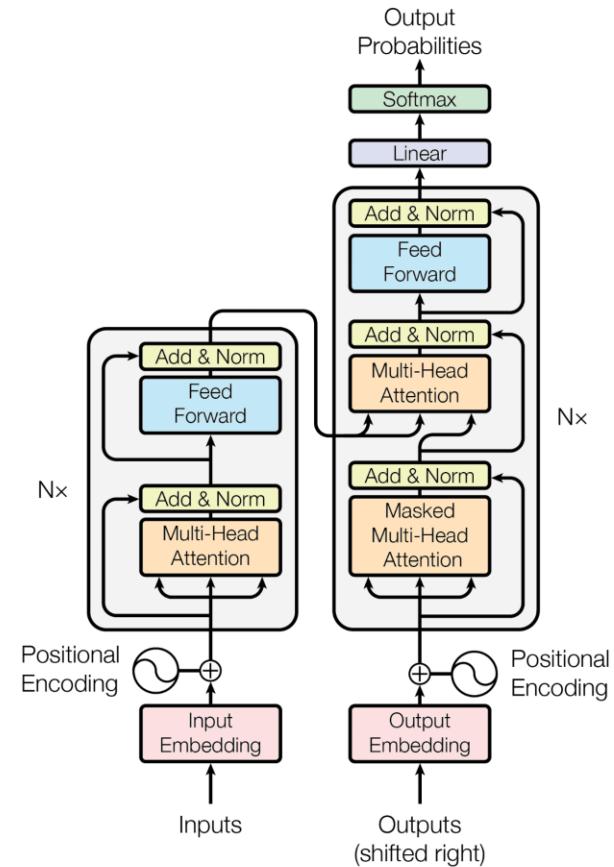


Advanced Neural Network Architectures: Transformers

Applications of transformers extend beyond NLP and include:

- BERT (Bidirectional Encoder Representations from Transformers): Used for various NLP tasks, such as question answering and sentiment analysis.
- GPT (Generative Pre-trained Transformer): Employed for language modeling and text generation.
- T5 (Text-to-Text Transfer Transformer): Generalized transformer model capable of performing various tasks with a unified text-to-text approach.
- Vision Transformers (ViTs): Applying transformers to computer vision tasks, replacing the traditional convolutional neural networks (CNNs) in some scenarios.

Transformers have shown state-of-the-art performance in many tasks and have become a fundamental building block for a wide range of deep learning applications. Researchers continue to explore and refine transformer architectures for different domains and tasks.



Advanced Neural Network Architectures: Transformers

LET'S BUILD GPT.
FROM SCRATCH.
IN CODE.
SPELLED OUT.



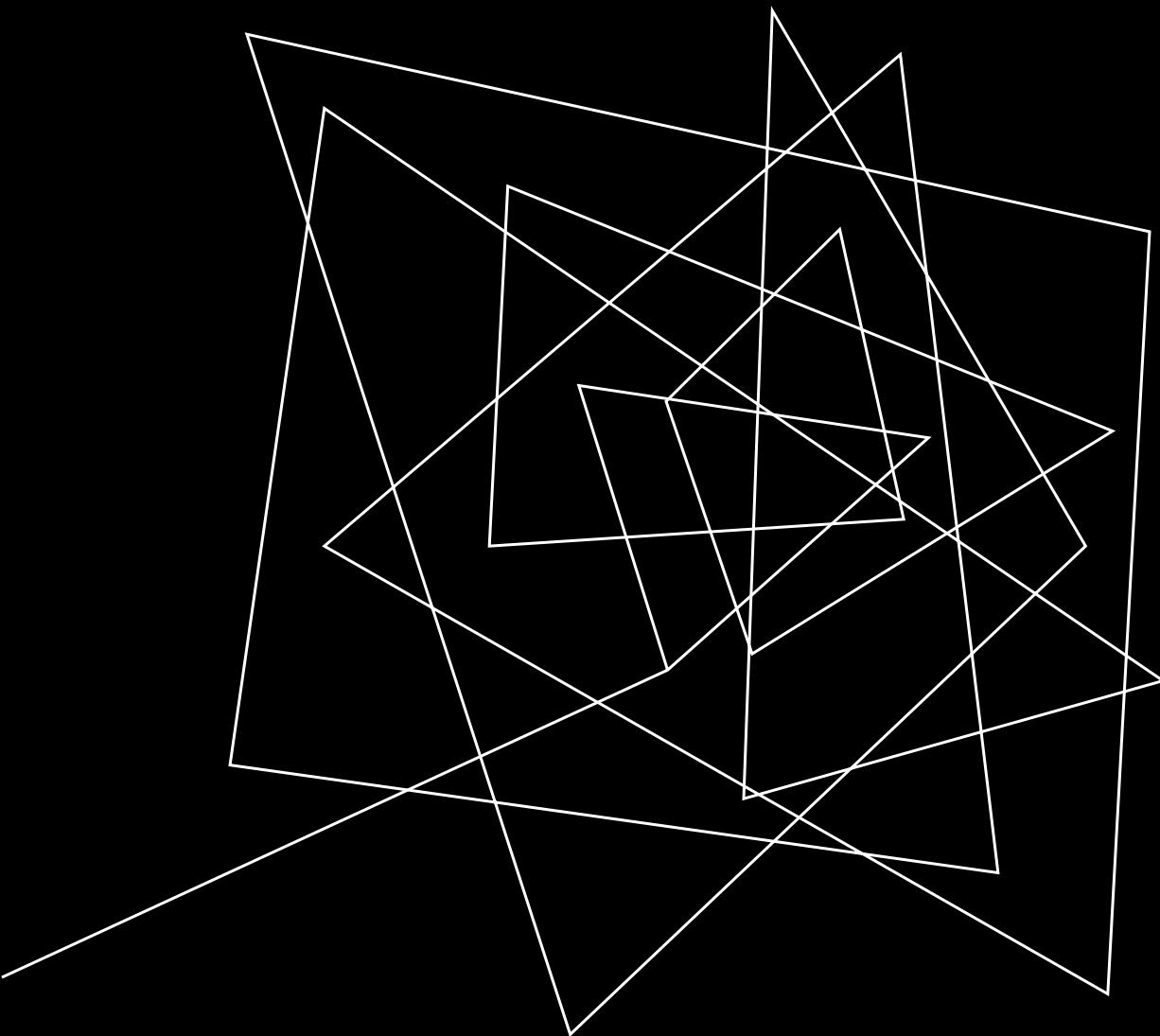
[Let's build GPT: from scratch, in code, spelled out.](#)

Transformer Neural Networks... *iVamos!*



Let's go!
...Clearly Explained!!!

[Transformer Neural Networks, ChatGPT's foundation,
Clearly Explained!!!](#)

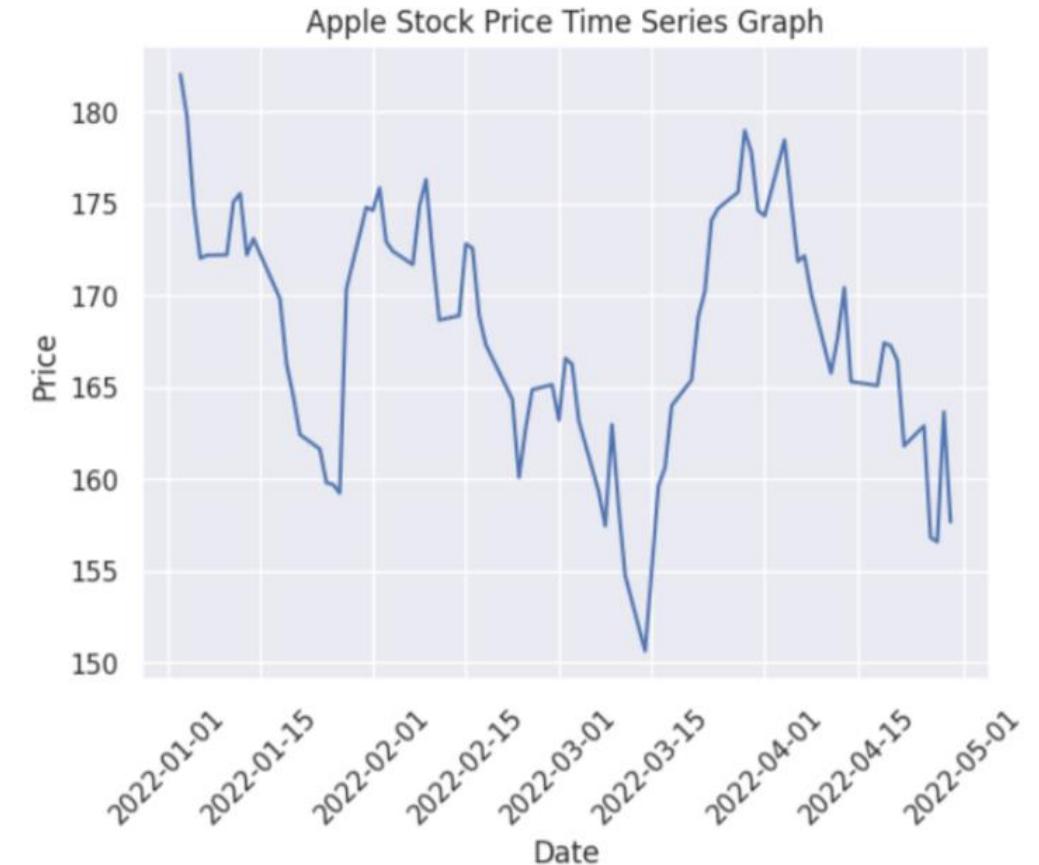


SESSION 6: TIME SERIES ANALYSIS

Introduction to time series analysis and forecasting,
understanding trends, patterns, and predictions

Introduction: What is Time Series?

- **Time Series:** A Sequence Unveiling Insights Over Time
- **Definition 1:** A time series is a sequential arrangement of data points, where each point corresponds to a specific moment in time. This ordered structure allows us to explore and analyze how data evolves, uncovering patterns, trends, and anomalies that might be hidden in traditional datasets.
- **Definition 2:** a time series is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus, it is a sequence of discrete-time data.

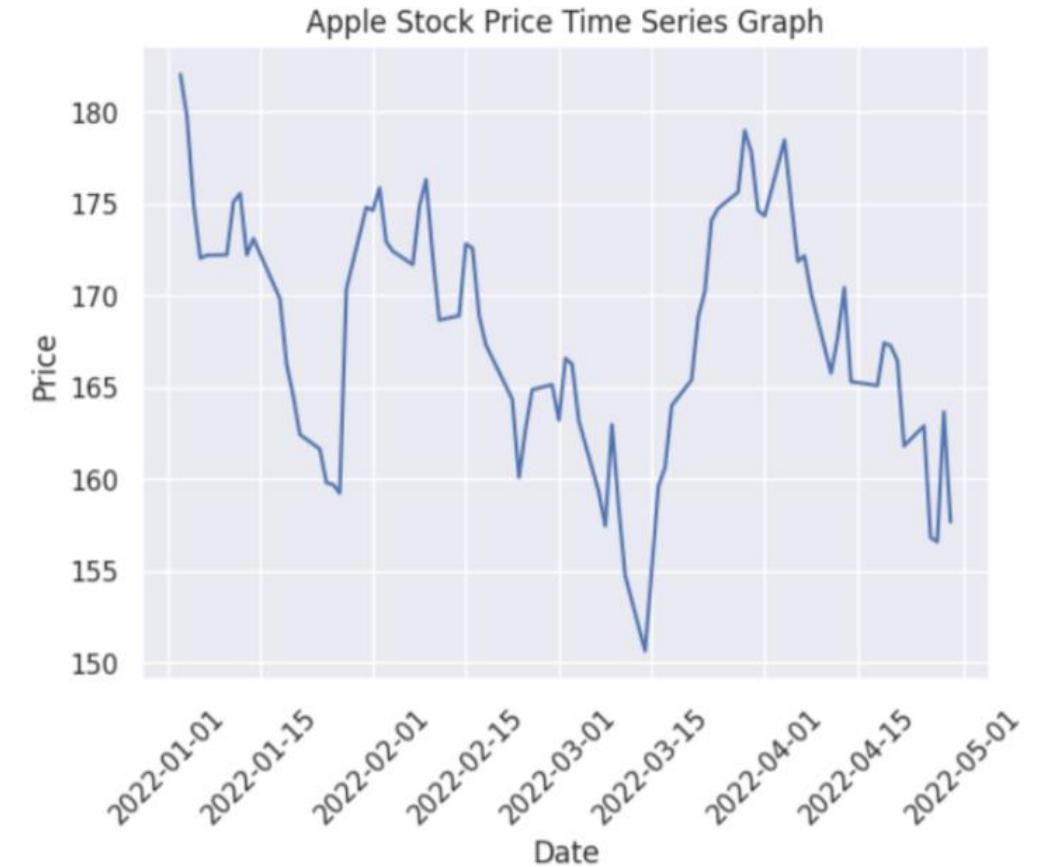


Source: timescale.com

Introduction: Examples of Time Series

Examples of Time Series:

- **Stock Prices:** Daily closing prices of stocks represent a classic time series, revealing the historical performance of a company's shares over time.
- **Temperature Readings:** Hourly, daily, or seasonal temperature data forms a time series, aiding meteorologists in understanding climate patterns and trends.
- **Sales Data:** Monthly sales figures provide a time series perspective, offering insights into product demand variations and seasonal trends.



Source: timescale.com

Time Series Components: Trend

What is Trend?

The trend component of a time series represents a persistent, long-term change in the mean of the series. The trend is the slowest-moving part of a series, the part representing the largest time scale of importance. In a time series of product sales, an increasing trend might be the effect of a market expansion as more people become aware of the product year by year.

Trend:

- **Definition:** A long-term movement or direction in the data.
- **Example:** The steady increase in annual sales over several years.

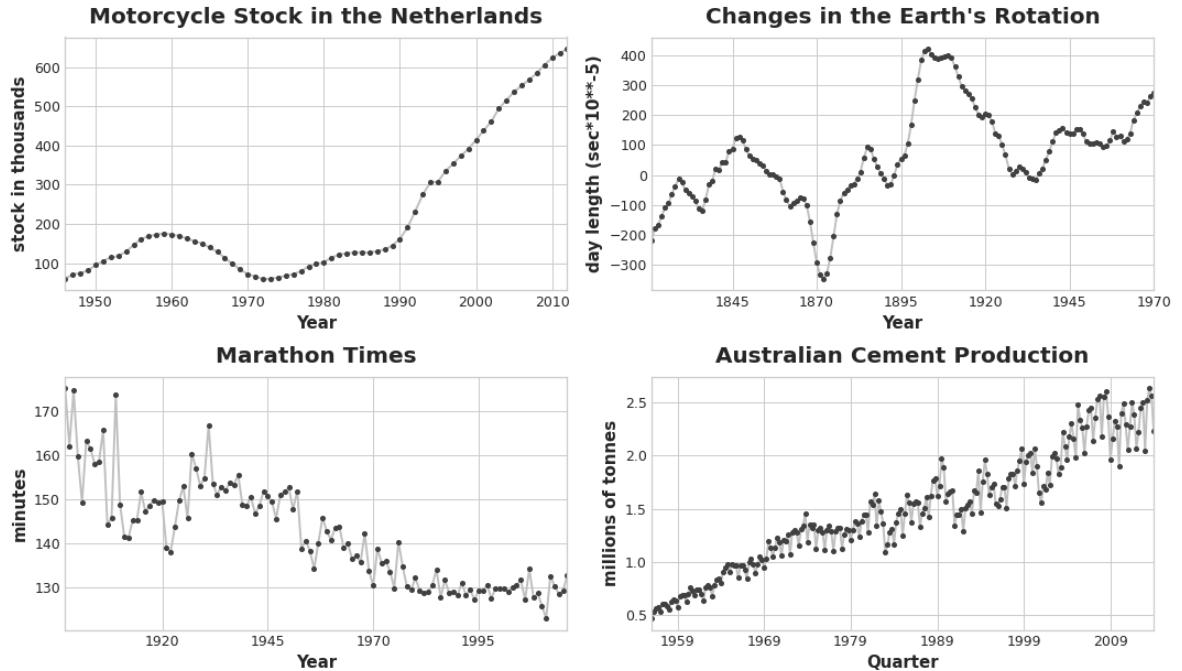


Fig. Trend patterns in four time series.

Time Series Components: Trend

A trend in time series data does not always imply an increase. A trend represents the long-term movement or direction in the data, and it can take various forms, including increases, decreases, or even remain relatively flat. Here are different types of trends:

Increasing Trend:

Characterized by a consistent upward movement in the data over time.

Decreasing Trend:

Characterized by a consistent downward movement in the data over time.

Flat (Constant) Trend:

Characterized by a relatively stable and constant level of the data over time.

Non-linear Trend:

The trend may exhibit more complex patterns, such as curves or waves, indicating a non-linear relationship.

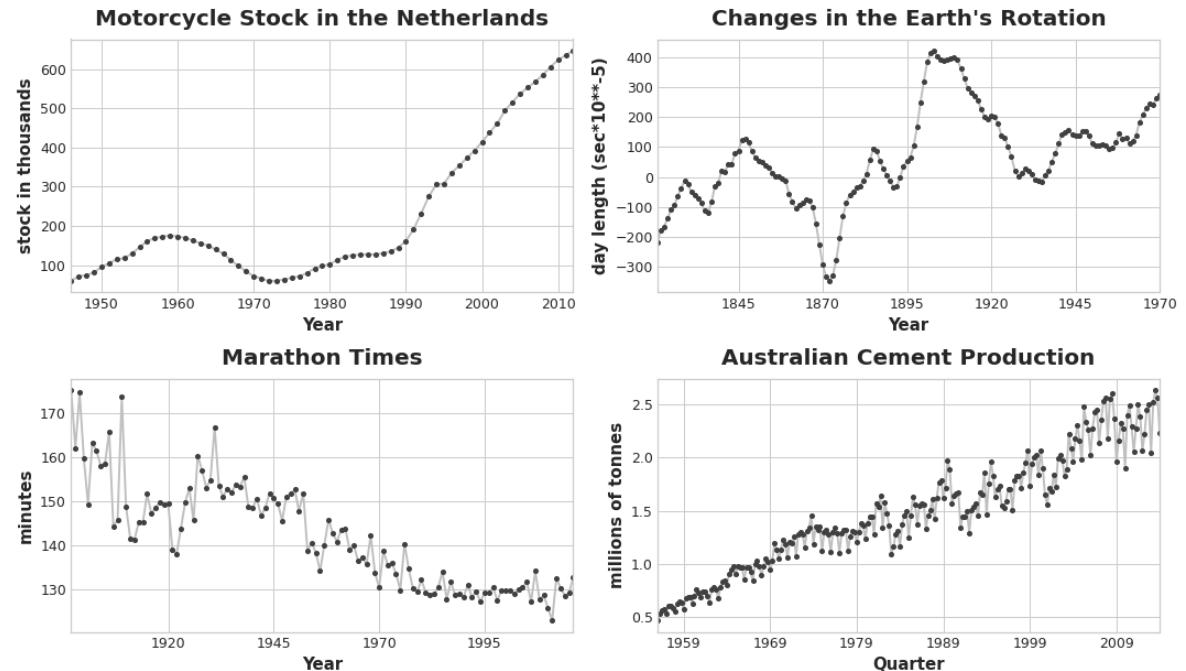


Fig. Trend patterns in four time series.

Time Series Components: Seasonality

What is Seasonality?

We say that a time series exhibits seasonality whenever there is a regular, periodic change in the mean of the series. Seasonal changes generally follow the clock and calendar -- repetitions over a day, a week, or a year are common. Seasonality is often driven by the cycles of the natural world over days and years or by conventions of social behavior surrounding dates and times.

Seasonality:

Definition: Regular and predictable fluctuations within a specific time frame.

Example: Monthly variations in ice cream sales due to seasonal demand.

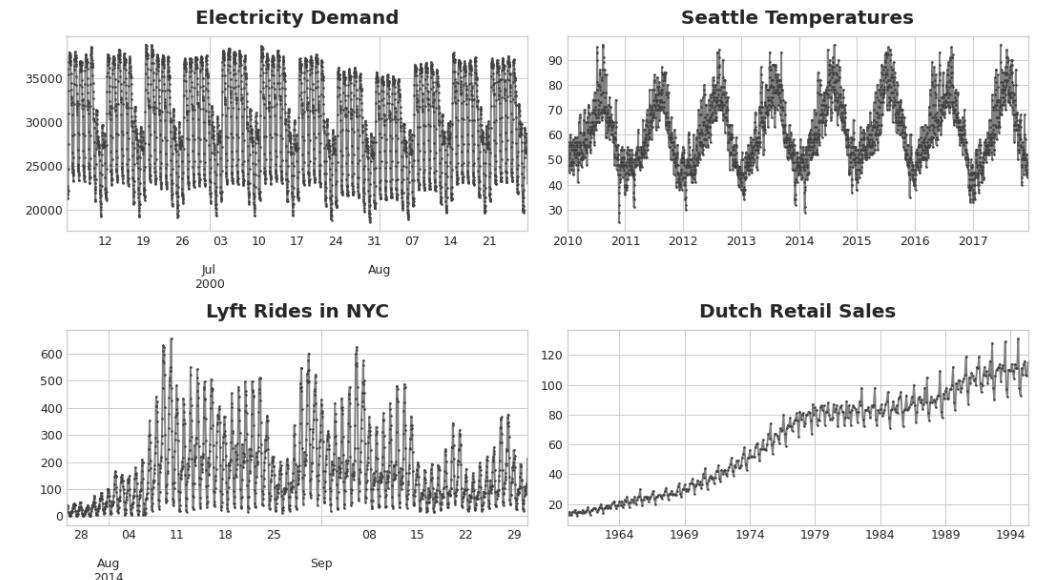


Fig. Seasonality patterns in four time series.

Time Series Analysis Methods: Exploratory Data Analysis

Exploratory Data Analysis (EDA) involves a set of techniques aimed at gaining insights into time series data before applying forecasting models. By visually and statistically exploring the characteristics of the data, analysts can make informed decisions about the appropriate modeling techniques.

Visual Inspection for Pattern Identification

Importance of Visualization:

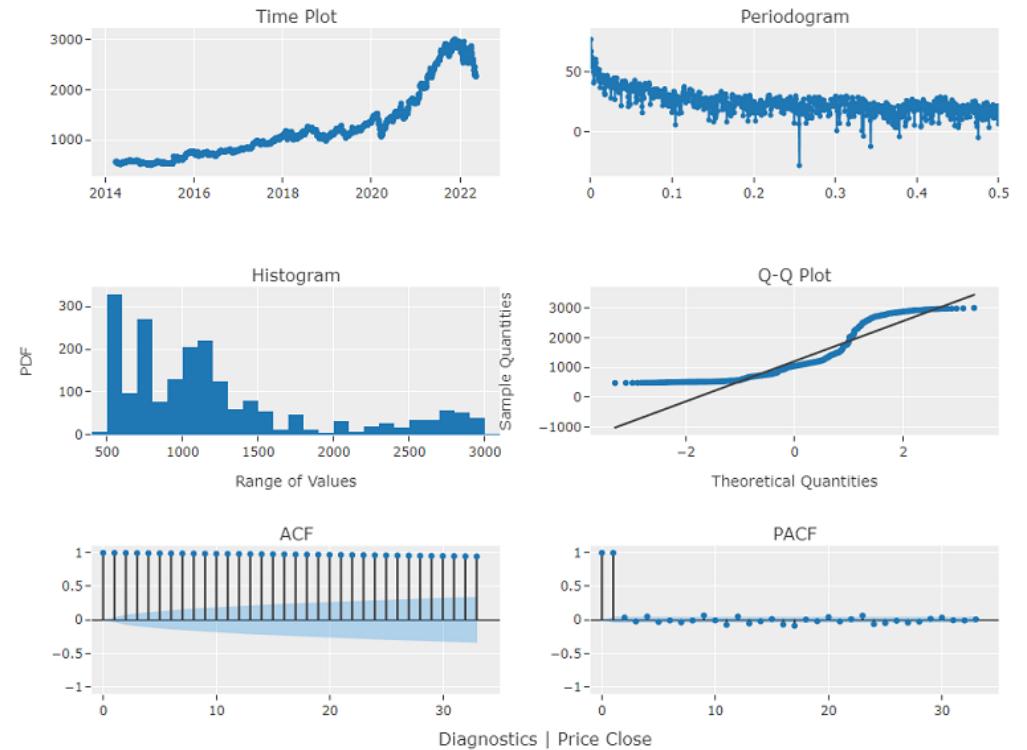
Visualizing time series data is crucial for understanding its underlying patterns and trends.

Graphical representations, such as line charts and seasonal decomposition plots, provide a clear view of the data's dynamics.

Pattern Identification:

Detecting patterns involves recognizing recurring structures or behaviors in the data.

Patterns may include trends (long-term movements), seasonality (repeating cycles), and cyclic patterns.



Source: developers.lseg.com

Time Series Analysis Methods: Exploratory Data Analysis

Outlier Detection for Data Quality Assurance

Outliers in Time Series:

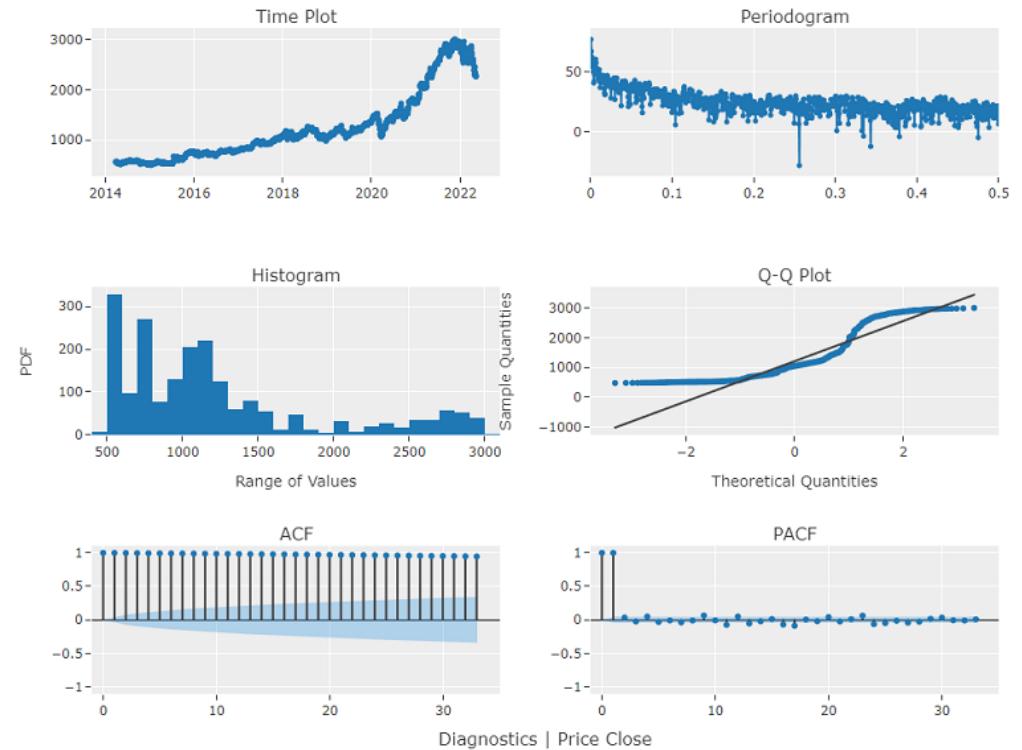
Outliers are data points that deviate significantly from the general pattern of the time series.

Identification of outliers is essential for ensuring data quality and preventing misleading forecasting results.

Techniques for Outlier Detection:

Statistical methods, such as z-scores and interquartile range (IQR), help identify data points deviating from the norm.

Visual techniques, like box plots, can highlight data points lying outside expected ranges.



Source: developers.lseg.com

Forecasting Techniques: Moving Average

Concept: Moving averages smooth out variations in data by calculating the average of a fixed-size window as it moves through the time series.

Application: Widely used for trend identification and noise reduction in time series data.

$$\text{Moving Average} = \frac{\text{Sum of Values in Window}}{\text{Number of Values in Window}}$$

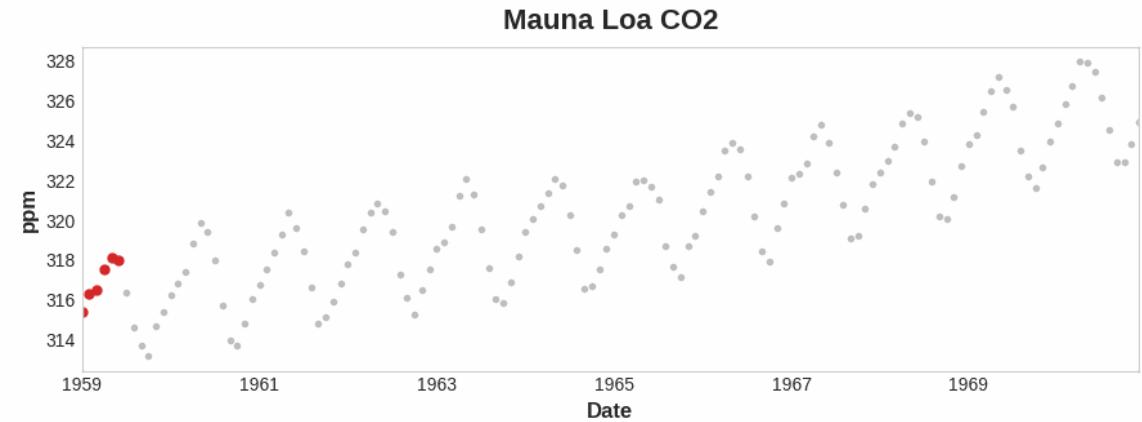


Fig. A moving average plot illustrating a linear trend. Each point on the curve (blue) is the average of the points (red) within a window of size 12.

Source: [kaggle.com](https://www.kaggle.com)

Forecasting Techniques: Moving Average

```

import pandas as pd
import matplotlib.pyplot as plt

time_series_column = 'Hardcover'

# Choose the window size for the moving average (e.g., 7 for a 7-day moving average)
window_size = 7

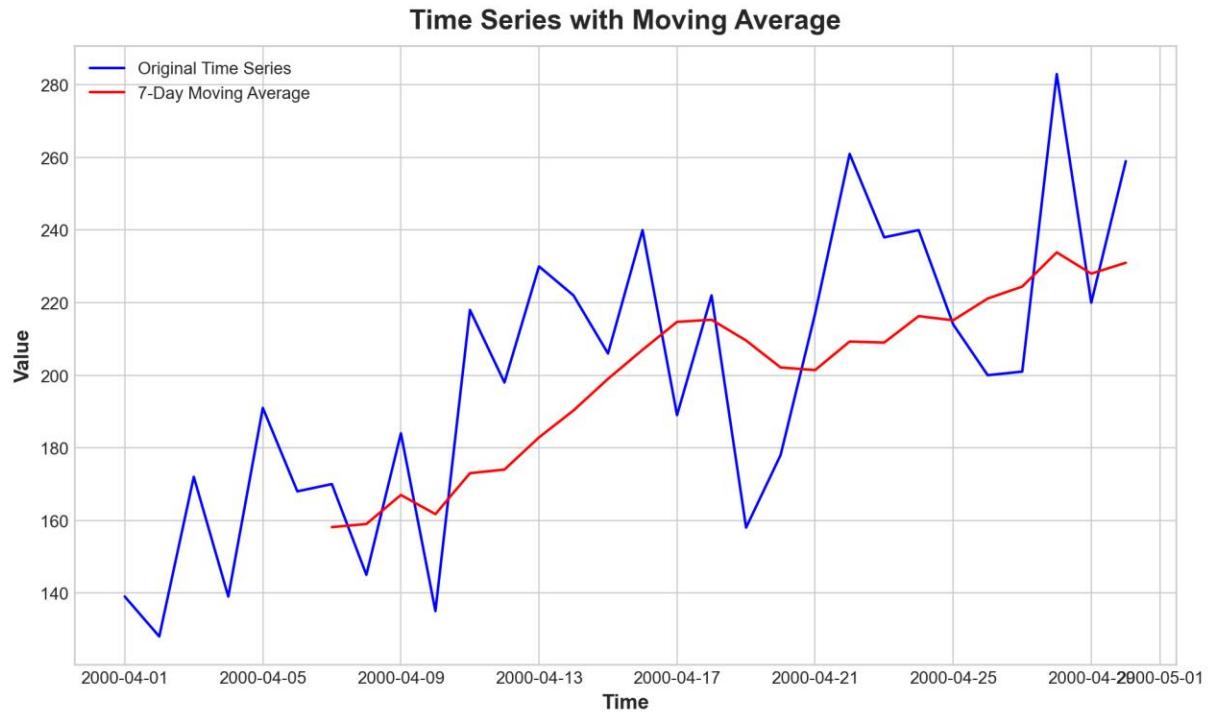
# Calculate the moving average using the rolling function
df['moving_average'] = df[time_series_column].rolling(window=window_size).mean()

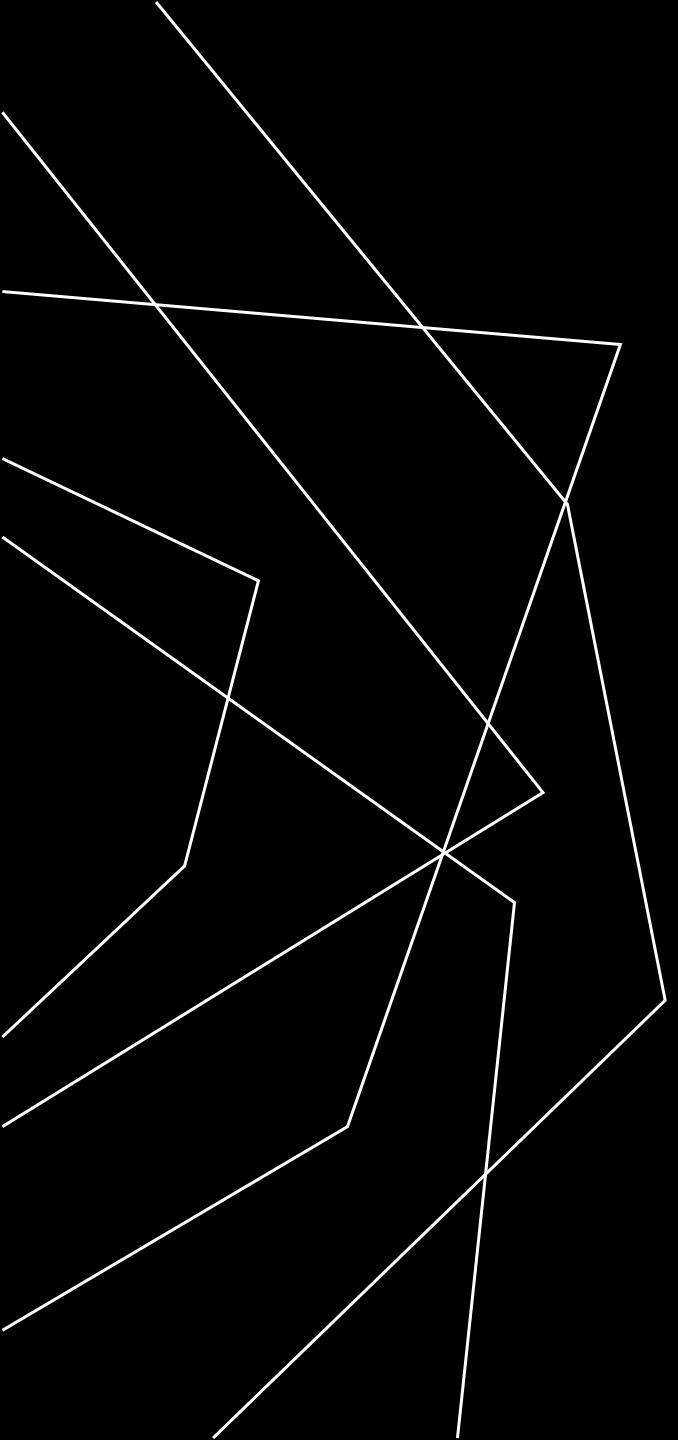
# Plot the original time series and the moving average
plt.figure(figsize=(10, 6))
plt.plot(df[time_series_column], label='Original Time Series', color='blue')
plt.plot(df['moving_average'], label=f'{window_size}-Day Moving Average', color='red')

# Add labels and a legend
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Time Series with Moving Average')
plt.legend()

# Show the plot
plt.show()

```





THANK YOU

Idriss JAIRI

Idriss.jairi@univ-lille.fr