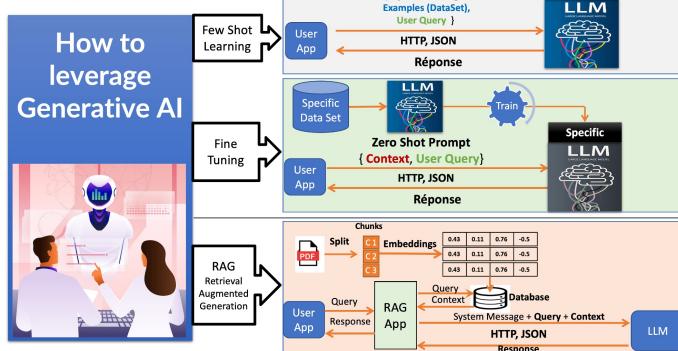


# Intelligence Artificielle & IA Générative

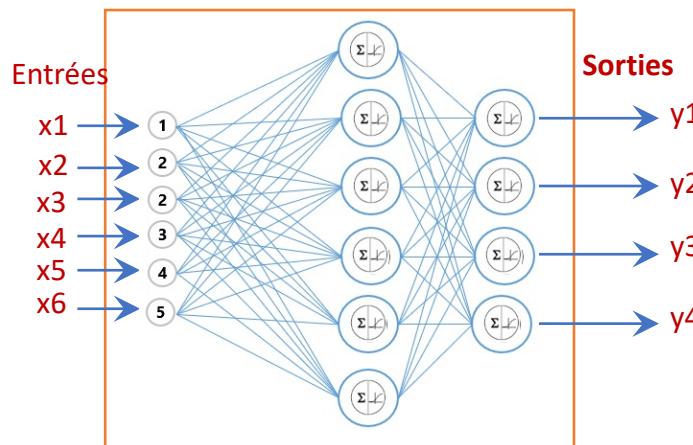


Mohamed Youssfi, Enseignant Chercheur, ENSET Mohammedia, Directeur Laboratoire Informatique, Intelligence Artificielle et Cyber Sécurité

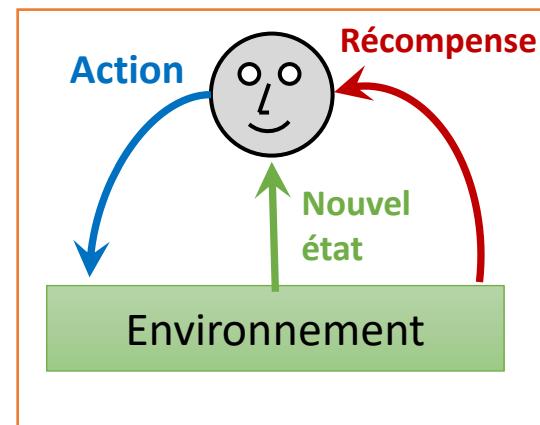
## Leverage Generative AI



## Apprentissage Supervisé



## Apprentissage Par Renforcement



## Artificial Intelligence Machine Learning

- Data-driven learning**
  - Supervised Learning
  - Unsupervised Learning
  - Self Supervised Learning
- Experiential learning**
  - Reinforcement Learning

## Deep Learning

Neural Network  
Computer Vision, NLP  
CNN, RNN, LSTM, GRU

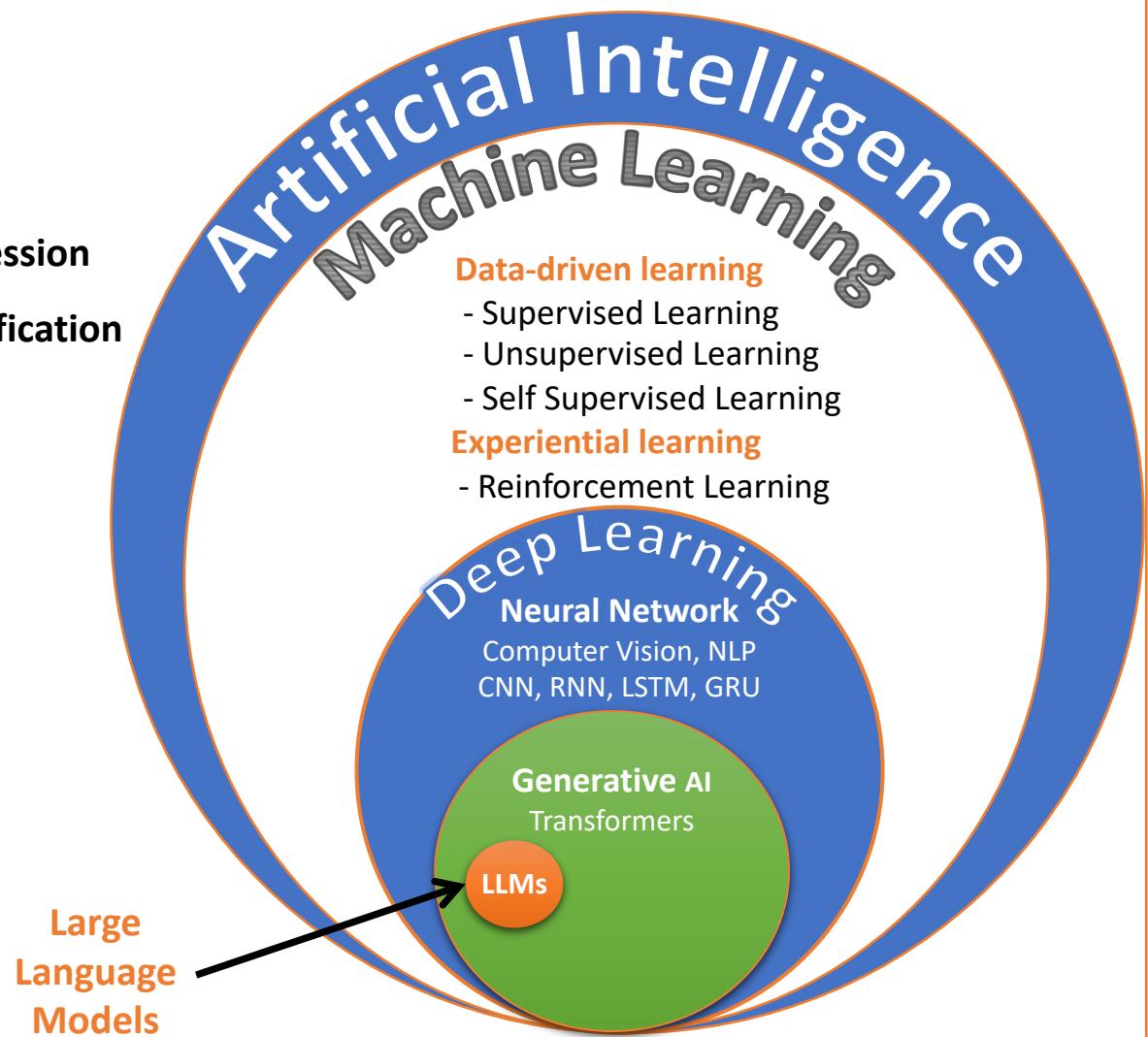
Generative AI  
Transformers

Large Language Models



# Intelligence Artificielle

- L'intelligence artificielle
- Intelligence Artificielle Distribuée =
  - IA : Pour des agents intelligent (Modéliser le savoir et le comportement)
  - + Distribuée : Modéliser leurs interactions => **Intelligence Collective**
- IA = **IA symbolique (5%) + Machine Learning (95%)**
- Techniques d'apprentissage :
  - Piloté par les données
    - Apprentissage Supervisé
    - Apprentissage Non Supervisé
    - Apprentissage Auto Supervisé
  - Piloté par l'expérience
- Deep Learning
- IA Générative



# Intelligence Artificielle

- L'intelligence artificielle
- Intelligence Artificielle Distribuée =
  - IA : Pour des agents intelligent (Modéliser le savoir et le comportement)
  - + Distribuée : Modéliser leurs interactions => **Intelligence Collective**
- IA = IA symbolique (5%) + Machine Learning (95%)
- Techniques d'apprentissage :
  - Piloté par les données
  - Piloté par l'expérience
- Deep Learning
- IA Générative



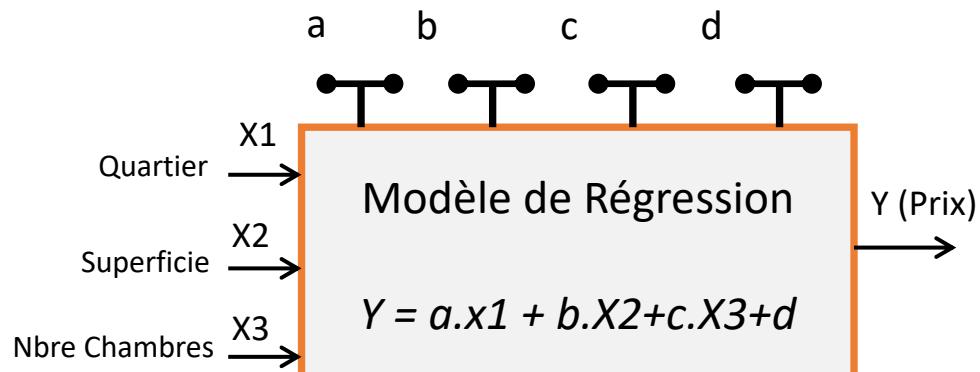
## Apprentissage Supervisé

### Régression

Exemple : Prédiction du prix d'un appartement

Data Set (Données étiquetées)

| Input       | Output        |               |              |
|-------------|---------------|---------------|--------------|
| Quartier X1 | Superficie X2 | Nbre chambres | Prix (MDH) Y |
| Maarif      | 100           | 3             | 2            |
| SBATA       | 120           | 3             | 1.1          |
| Maarif      | 100           | 4             | 2.1          |
| Bourgogne   | 80            | 3             | 1.44         |
| Maarif      | 200           | 5             | ???????      |



Modèle à 4 Paramètres a, b, c et d

Algorithmes : Linéaire, Polynomiale, Ridge, Lasso, etc..

# Intelligence Artificielle

- L'intelligence artificielle
- Intelligence Artificielle Distribuée =
  - IA : Pour des agents intelligent (Modéliser le savoir et le comportement)
  - + Distribuée : Modéliser leurs interactions => **Intelligence Collective**
- IA = IA symbolique (5%) + Machine Learning (95%)
- Techniques d'apprentissage :
  - Piloté par les données
  - Piloté par l'expérience
- Deep Learning
- IA Générative



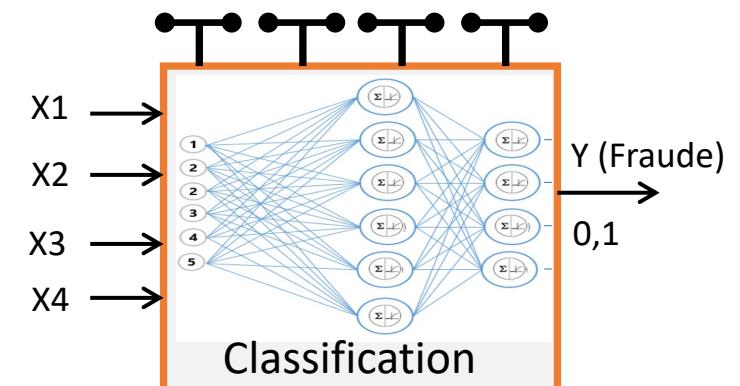
## Apprentissage Supervisé

### Classification

Exemple : Prédiction du prix d'un appartement

Data Set (Données étiquetées)

| Input    |            |         |        | Output  |
|----------|------------|---------|--------|---------|
| Heure X1 | Montant X2 | Long X3 | Lat X4 | Fraude  |
| 12:20    | 4500       | -1.2    | 3.2    | 0       |
| 01:45    | 1,60       | 0.6     | 4.3    | 1       |
| 10:08    | 100        | -2.45   | 1.3    | 0       |
| 11:55    | 80         | -1.2    | 3.2    | 1       |
| 00:00    | 3200       | -1      | 3      | ??????? |



Algorithmes : KNN, SVM, DT, RF, Réseaux de neurones, etc...

# Intelligence Artificielle

- L'intelligence artificielle
- Intelligence Artificielle Distribuée =
  - IA : Pour des agents intelligent (Modéliser le savoir et le comportement)
  - + Distribuée : Modéliser leurs interactions => **Intelligence Collective**
- IA = IA symbolique (5%) + Machine Learning (95%)
- Techniques d'apprentissage :
  - Piloté par les données
  - Apprentissage Supervisé
    - Régression
    - Classification
  - Apprentissage Non Supervisé
  - Apprentissage Auto Supervisé
  - Piloté par l'expérience
- Deep Learning
- IA Générative

## Apprentissage Non Supervisé

### Clustering

Data Set (Données Non étiquetées)



Algorithmes : K-Means

# Intelligence Artificielle

- L'intelligence artificielle
- Intelligence Artificielle Distribuée =
  - IA : Pour des agents intelligent (Modéliser le savoir et le comportement)
  - + Distribuée : Modéliser leurs interactions => **Intelligence Collective**
- IA = IA symbolique (5%) + Machine Learning (95%)
- Techniques d'apprentissage :
  - Piloté par les données
    - Apprentissage Supervisé
    - Apprentissage Non Supervisé
    - Apprentissage Auto Supervisé
  - Piloté par l'expérience
- Deep Learning
- IA Générative

## Apprentissage Auto Supervisé

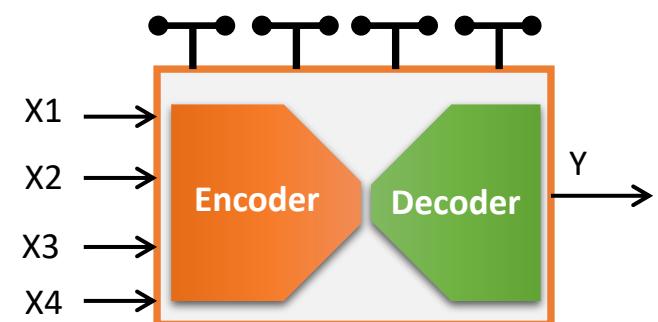
### Self-Supervised Learning

#### Encodeur - Décodeur

#### Data Set (Données Non étiquetées)

Cacher une partie des données et entraîner le modèle à prédire ce parties cachées en utilisant le principe des encodeurs et décodeurs

Pae exemple en NLP, On cache des mot du texte et on entraîne le modèle pour deviner ces mots cachés



Algorithmes : Transformers (BERT, GPT)

# Intelligence Artificielle

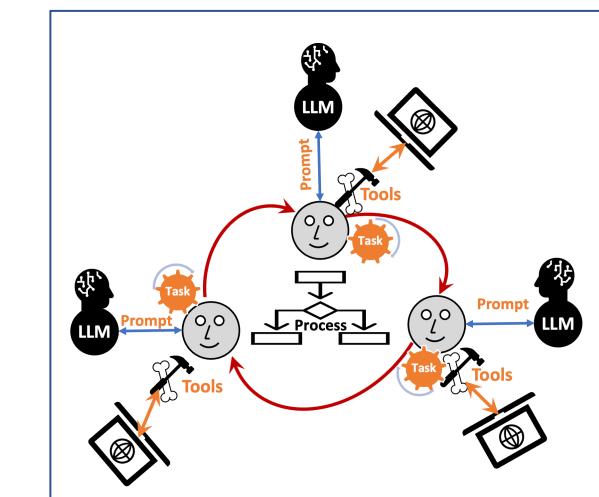
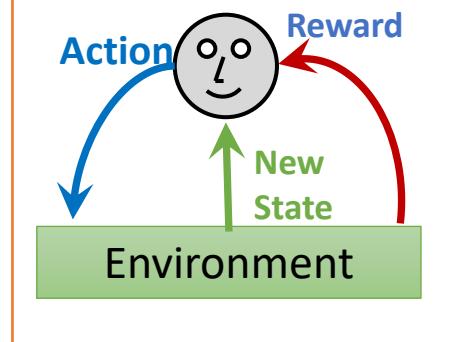
- L'intelligence artificielle
- Intelligence Artificielle Distribuée =
  - IA : Pour des agents intelligent (Modéliser le savoir et le comportement)
  - + Distribuée : Modéliser leurs interactions => **Intelligence Collective**
- IA = IA symbolique (5%) + Machine Learning (95%)
- Techniques d'apprentissage :
  - Piloté par les données
    - Apprentissage Supervisé
    - Régression
    - Classification
  - Apprentissage Non Supervisé
  - Apprentissage Auto Supervisé
- Piloté par l'expérience
- Deep Learning
- IA Générative

# Apprentissage Par Renforcement

## Reinforcement Learning

### Embodiment

Reinforcement learning



Algorithmes : Q-Learning, PPO

# Intelligence Artificielle

- L'intelligence artificielle
- Intelligence Artificielle Distribuée =
  - IA : Pour des agents intelligent (Modéliser le savoir et le comportement)
  - + Distribuée : Modéliser leurs interactions => **Intelligence Collective**
- IA = IA symbolique (5%) + Machine Learning (95%)
- Techniques d'apprentissage :
  - Piloté par les données
    - Apprentissage Supervisé
    - Classification
    - Régression
  - Apprentissage Non Supervisé
  - Apprentissage Auto Supervisé
- Piloté par l'expérience
- Deep Learning
- IA Générative

## Deep Learning



CNN : [ Convolution, RELU, MAX PULLING ] [ Fully Connected ]

RNN : [ Recurrent Neural Network ]

# Intelligence Artificielle

- L'intelligence artificielle
- Intelligence Artificielle Distribuée =
  - IA : Pour des agents intelligent (Modéliser le savoir et le comportement)
  - + Distribuée : Modéliser leurs interactions => **Intelligence Collective**

- IA = **IA symbolique (5%) + Machine Learning (95%)**

- Techniques d'apprentissage :

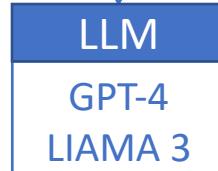


- Piloté par les données
- Piloté par l'expérience

- Deep Learning

- IA Générative

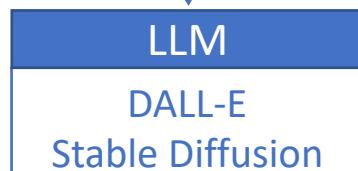
Text Prompt



(Text + Images) Prompt



(Text + Images) Prompt

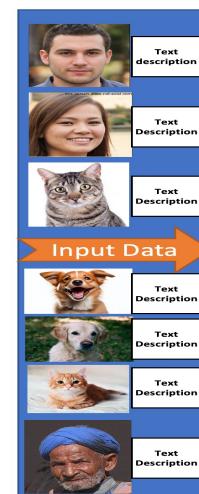


Text Generation

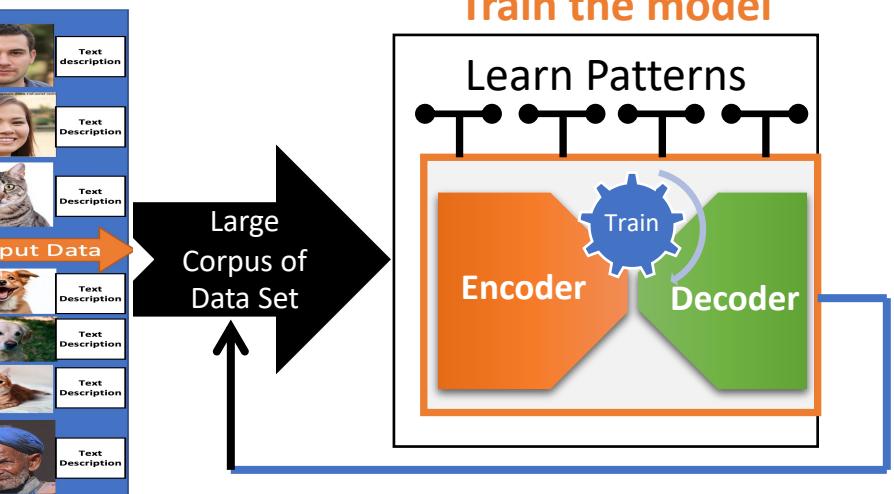
Text Generation

Image Generation

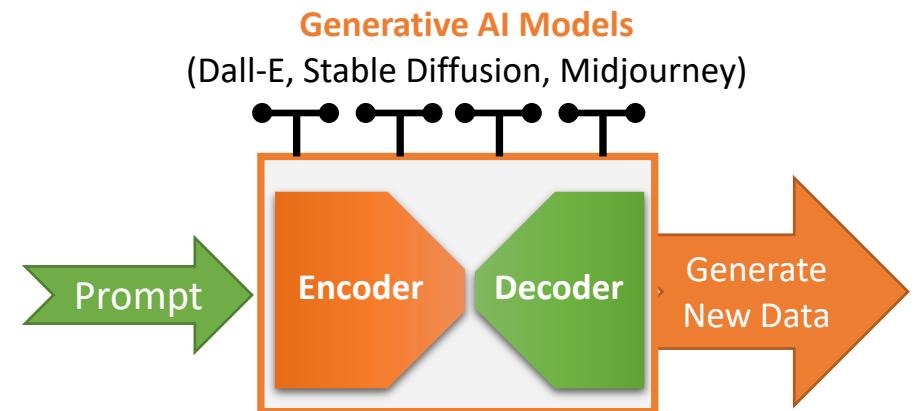
**L'IA Générative** est une catégorie de l'IA capable de générer des nouvelles données en conséquence d'un raisonnement conditionné par un prompt utilisateur en utilisant les LLMs (Large Language Models).



Large Corpus of Data Set



**Generative AI Models**  
(Dall-E, Stable Diffusion, Midjourney)



# Concepts fondamentaux de l'IA

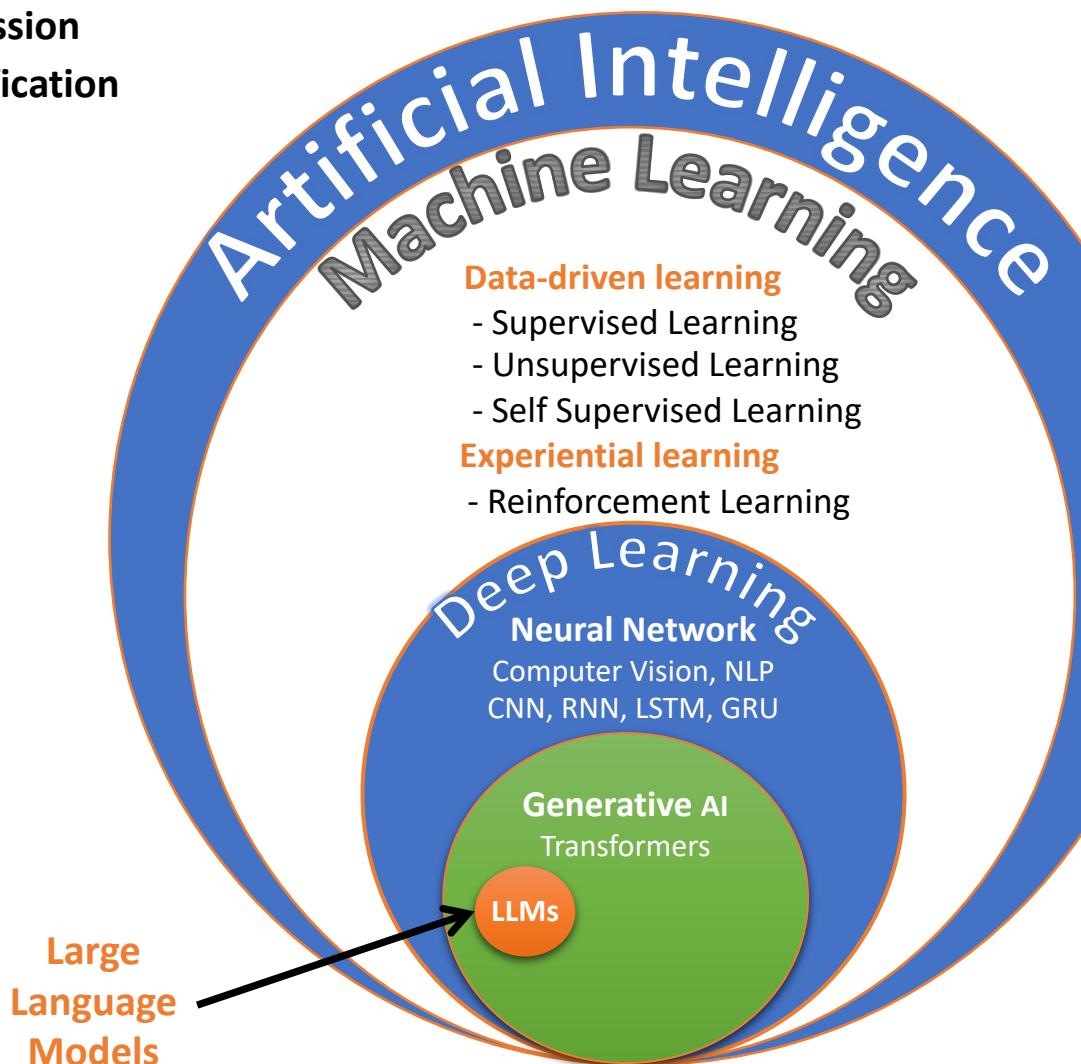
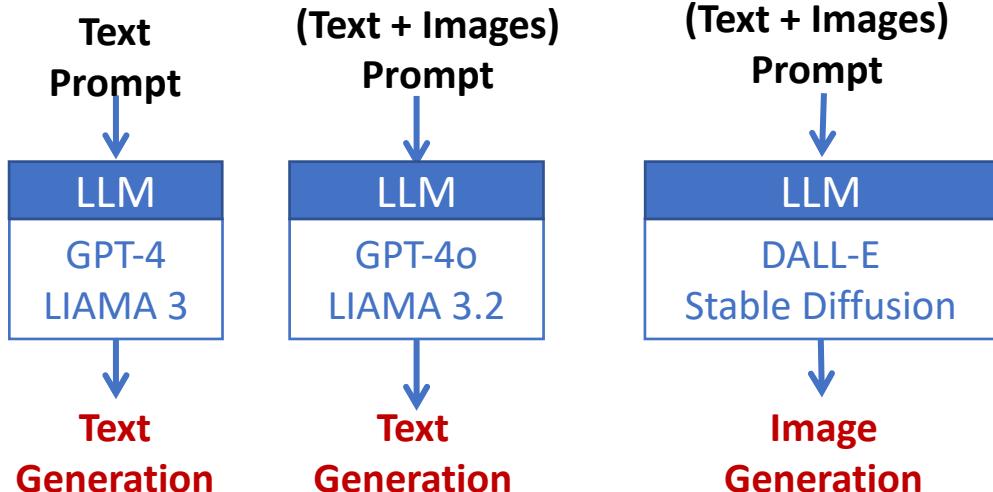
- L'intelligence artificielle
- IA = IA symbolique (5%) + Machine Learning (95%)
- Techniques d'apprentissage :
  - Piloté par les données
  - Piloté par l'expérience ( Apprentissage par Renforcement )



- Intelligence Artificielle Distribuée =
  - IA : Pour des agents intelligent (Modéliser le savoir et le comportement)
  - + Distribuée : Modéliser leurs interactions => **Intelligence Collective**

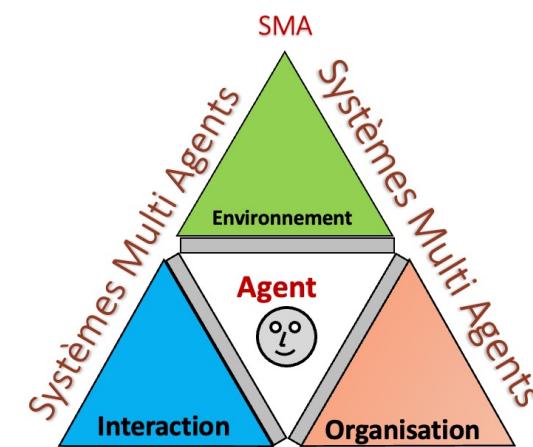
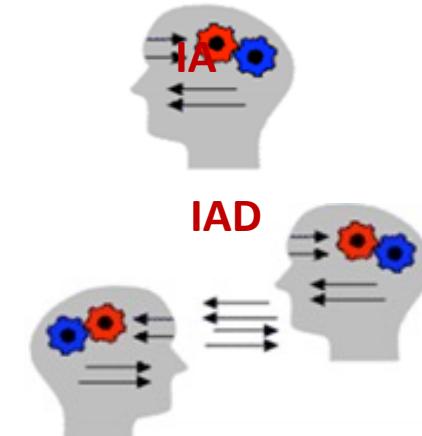
## Generative AI

### Large Language Models (LLMs) : Transformers (GPT, BERT)

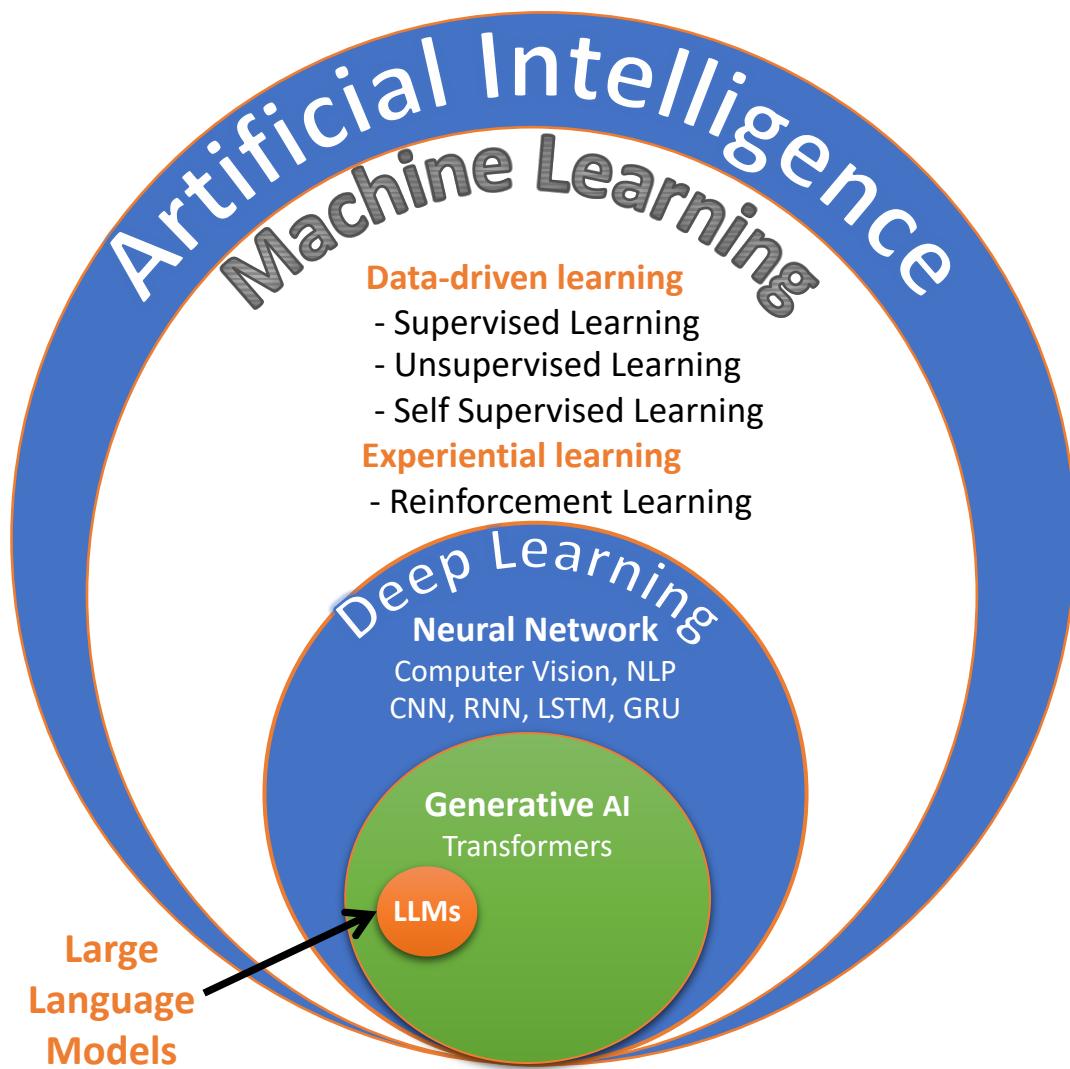


# Intelligence Artificielle Distribuée & AI Agents

- **L'intelligence artificielle** est une discipline qui cherche doter les systèmes informatiques avec des capacités intellectuelles semblables à celle des êtres humains et des animaux en utilisant des algorithmes
- **IA et IAD :**
  - L'Intelligence Artificielle permet de modéliser un penseur isolé en exploitant l'intelligence individuelle d'un agent.
  - L'Intelligence Artificielle Distribuée permet d'exploiter l'intelligence collective de plusieurs agents qui vont collaborer selon une planification et une organisation pour participer ensemble à résoudre des problèmes complexes. Cette discipline est connue par les Systèmes Multi Agents. Un Agent est une entité autonome entraînée pour prendre des décisions pour atteindre un but pour lequel il a été créé.

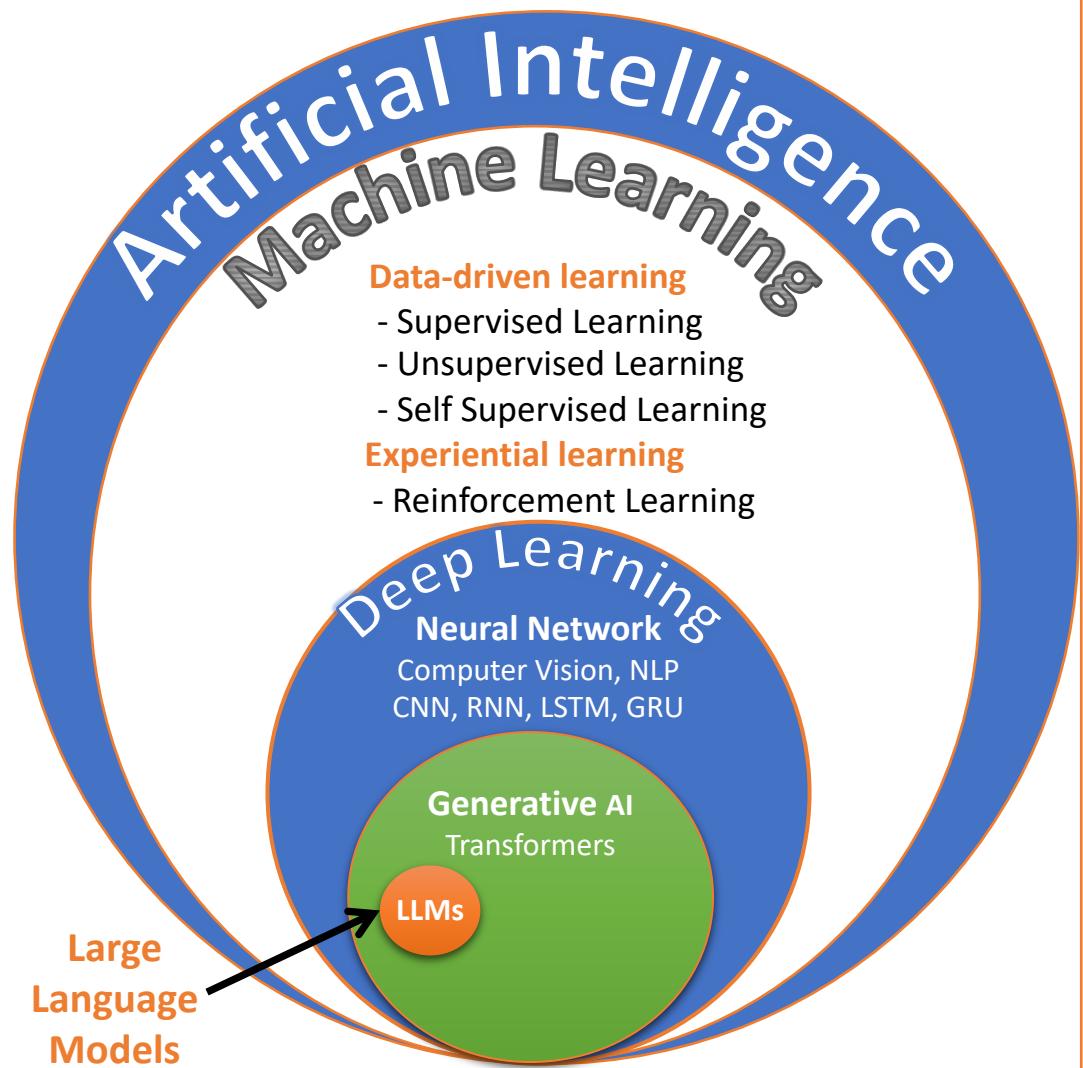


- Il existe deux **familles d'algorithmes de l'IA** :
  - l'IA Symbolique**: Des techniques qui n'ont pas pu décoller et donc rarement utilisées aujourd'hui dans les applications de l'IA. Elle couvre moins de 5% des applications industrielles.
  - Machine Learning** (Apprentissage automatique) (95%): Ce sont les techniques qui sont les plus exploitées aujourd'hui dans les applications de l'IA



# Machine Learning

- Dans le domaine du Machine Learning, il existe deux façons pour entraîner les algorithmes :
  - Apprentissage piloté par les données
  - Apprentissage piloté par l'expérience (Apprentissage par renforcement)
- Apprentissage piloté par les données :
  - Consiste à entraîner des algorithmes à effectuer des prédictions en utilisant un ensemble de données collectées du domaine réel étudié.
  - Il existe trois types d'apprentissage pilotés par les données\*\* :
    - Apprentissage supervisé
    - Apprentissage non supervisé (Clustering)
    - Apprentissage Auto Supervisé (Self Supervised Learning)
    - Apprentissage piloté par l'expérience (Apprentissage par renforcement)



# Apprentissage Supervisé

- Consiste à utiliser un data set étiqueté. C'est-à-dire des données dont on connaît les inputs et les outputs. Les outputs représentent des étiquettes ou des valeurs fournis par les experts du métier.
- Dans l'apprentissage supervisé on distingue deux types de problèmes :
  - **Régression** : Consiste à prédire, en sortie, une valeur continue comme le prix d'un appartement ou la durée de vie d'une pièce mécanique ou la durée de guérison d'un patient : Exemple Régression Linéaire
  - **Classification** : Consiste à prédire des classes d'appartenance parmi un ensemble de classe finies. Par exemple prédire si une transaction est frauduleuse ou encore prédire si un animal est un chien, un chat, un tigre ou un lapin.
  - Exemples d'algorithmes : Régression logistique, Support Vector Machine (SVM), Multi Layer Perceptron (MLP), KNN (K plus proches voisins), Decision Tree, Random Forest, etc.

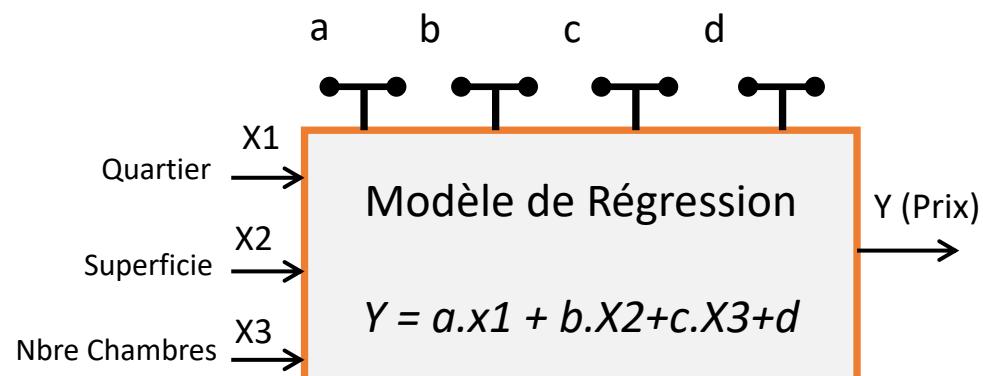
# Apprentissage Supervisé

## Régression

Exemple : Prédiction du prix d'un appartement

Data Set (Données étiquetées)

| Input       |               |               | Output       |
|-------------|---------------|---------------|--------------|
| Quartier X1 | Superficie X2 | Nbre chambres | Prix (MDH) Y |
| Maarif      | 100           | 3             | 2            |
| SBATA       | 120           | 3             | 1.1          |
| Maarif      | 100           | 4             | 2.1          |
| Bourgogne   | 80            | 3             | 1.44         |
| Maarif      | 200           | 5             | ???????      |



Modèle à 4 Paramètres a, b, c et d

Algorithmes : Linéaire, Polynomiale, Ridge, Lasso, etc..

# Apprentissage Supervisé

- Consiste à utiliser un data set étiqueté. C'est-à-dire des données dont on connaît les inputs et les outputs. Les outputs représentent des étiquettes ou des valeurs fournies par les experts du métier.
- Dans l'apprentissage supervisé on distingue deux types de problèmes :
  - Régression : Consiste à prédire, en sortie, une valeur continue comme le prix d'un appartement ou la durée de vie d'une pièce mécanique ou la durée de guérison d'un patient : Exemple Régression Linéaire
  - Classification : Consiste à prédire des classes d'appartenance parmi un ensemble de classes finies. Par exemple prédire sur une transaction est frauduleuse ou encore prédire si un animal est un chien, un chat, un tigre ou un lapin. Exemples : Régression logistique, Support Vector Machine (SVM), Multi Layer Perceptron (MLP), KNN (K plus proches voisins), Decision Tree, Random Forest, etc.

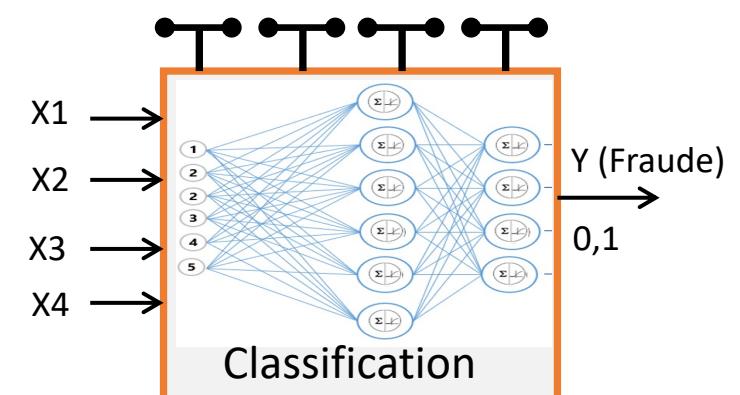
# Apprentissage Supervisé

## Classification

Exemple : Prédiction du prix d'un appartement

Data Set (Données étiquetées)

| Input       |               |            |           | Output |
|-------------|---------------|------------|-----------|--------|
| Heure<br>X1 | Montant<br>X2 | Long<br>X3 | Lat<br>X4 | Fraude |
| 12:20       | 4500          | -1.2       | 3.2       | 0      |
| 01:45       | 1,60          | 0.6        | 4.3       | 1      |
| 10:08       | 100           | -2.45      | 1.3       | 0      |
| 11:55       | 80            | -1.2       | 3.2       | 1      |
| 00:00       | 3200          | -1         | 3         | ?????? |



Algorithmes : KNN, SVM, DT, RF, Réseaux de neurones, etc...

# Apprentissage Non Supervisé

- **Apprentissage Non Supervisé :**

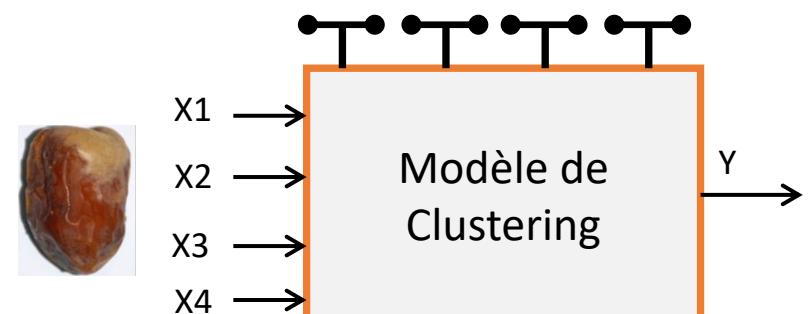
- Consiste à utiliser des données non étiquetées et utiliser des algorithmes de clustering qui vont fouiller dans les données pour chercher à les segmenter en un ensemble de groupes homogènes en se basant sur des mesures de similarités.

- Exemple : KMeans

# Apprentissage Non Supervisé

## Clustering

### Data Set (Données Non étiquetées)



Algorithmes : K-Means

# Apprentissage Auto Supervisé

- **Apprentissage Auto Supervisé :**

- Utilisé dans le domaine de NLP (Natural Language Processing),
- Ce type d'apprentissage consiste à exploiter des données non structurées et non étiquetées comme le texte d'un livre.
- Pendant l'entraînement d'un algorithme, on cache des parties du texte et puis on entraîner le modèle à prédire les parties cachées du texte en utilisant des encodeurs et des décodeurs qui sont basés sur un mécanisme d'attention qui est la base des Transformers qui sont les plus utilisés dans le domaine de l'IA générative sur les LLMs (Large Language Models) : Exemple BERT, GPT

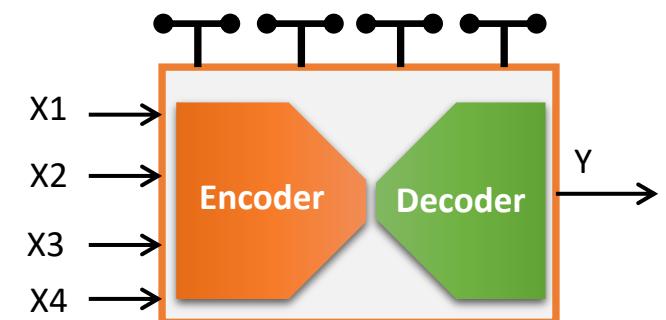
# Apprentissage Auto Supervisé

## Self-Supervised Learning

### Encodeur - Décodeur

#### Data Set (Données Non étiquetées)

*Cacher une partie des données et entraîner le modèle à prédire ces parties cachées en utilisant le principe des encodeurs et décodeurs*  
*Pae exemple en NLP, On cache des mots du texte et on entraîne le modèle pour deviner ces mots cachés*



Algorithmes : Transformers (BERT, GPT)

# Apprentissage Par renforcement

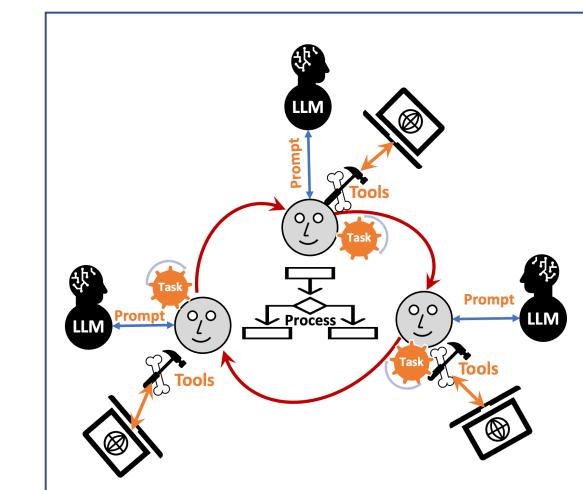
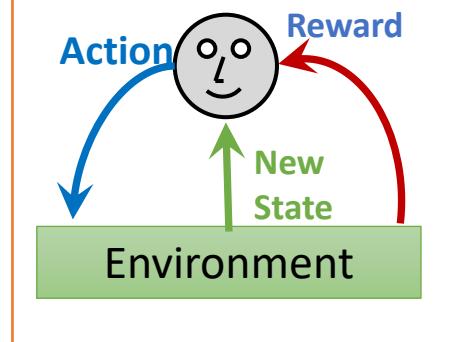
- **Apprentissage piloté par l'expérience (Apprentissage par renforcement) :**
  - Dans l'apprentissage par renforcement, on modélise l'environnement où un agent peut évoluer en agissant sur l'environnement en utilisant des actions.
  - Pendant la phase d'entraînement, l'agent qui possède un ensemble de Tools (Actions) explore l'environnement en agissant de manière itérative avec des actions disponibles.
  - Ensuite l'environnement lui procure des récompenses ou des observations. Ce qui permet de changer l'état de l'agent.
  - Ce processus répétitif d'exploration permet de construire une table de raisonnement.
  - Une fois que la phase d'entraînement et d'exploration est terminée, l'agent devient capable d'évoluer de manière autonome dans l'environnement en prenant les actions optimales selon son état actuel qui encapsule une abstraction et une représentation de l'environnement. Exemple : QLearning, PPO (Proximal Policy Optimisation)

# Apprentissage Par Renforcement

## Reinforcement Learning

### Embodiment

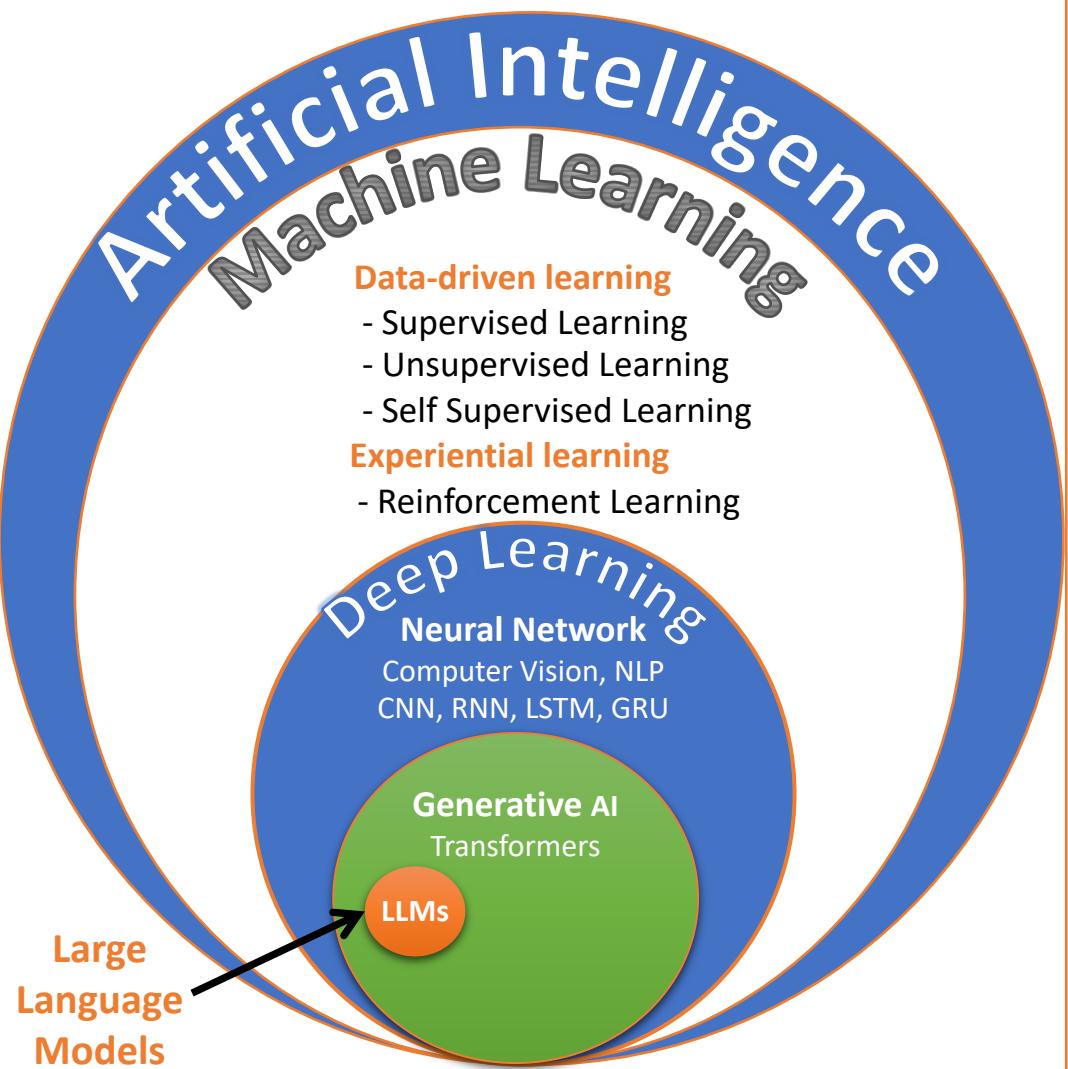
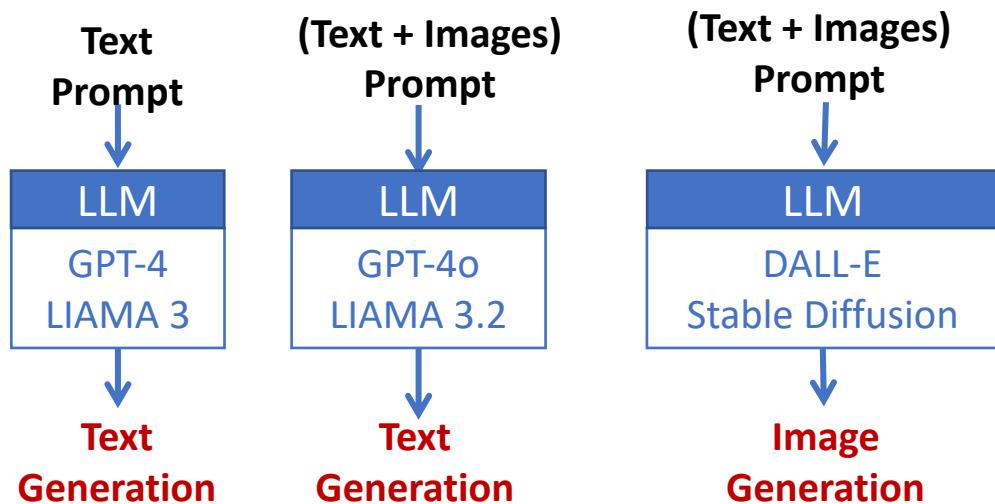
#### Reinforcement learning



Algorithmes : Q-Learning, PPO

# IA Générative

- L'IA Générative est une catégorie de l'IA capable de générer des nouvelles données en conséquence d'un raisonnement conditionné par un prompt utilisateur en utilisant les LLMs (Large Language Models).
- Ces LLMs, basés sur les algorithmes dits « Transformers » sont entraînés en utilisant un large corpus de données.
- Ces LLMs sont des réseaux de neurones de très grandes tailles avec des Billions de paramètres et sont capables de générer des nouvelles données n'ayant jamais été vues de différentes modalités : Texte, Images, Son, Vidéo, Musique. Exemples de LLMs : Gpt GPT-4o, DeepSeek, Claude, Gemini, GPT-03, Dall-e, CLIP, Stable Diffusion



# Apprentissage supervisé

## Régression

# Apprentissage Supervisé

- Consiste à utiliser un dataset étiqueté. C'est-à-dire des données dont on connaît les inputs et les outputs. Les outputs représentent des étiquettes ou des valeurs fournis par les experts du métier.
- Dans l'apprentissage supervisé on distingue deux types de problèmes :
  - **Régression** : Consiste à prédire, en sortie, une valeur continue comme le prix d'un appartement ou la durée de vie d'une pièce mécanique ou la durée de guérison d'un patient : Exemple Régression Linéaire
  - **Classification** : Consiste à prédire des classes d'appartenance parmi un ensemble de classe finies. Par exemple prédire si une transaction est frauduleuse ou encore prédire si un animal est un chien, un chat, un tigre ou un lapin.
  - Exemples d'algorithmes : Régression logistique, Support Vector Machine (SVM), Multi Layer Perceptron (MLP), KNN (K plus proches voisins), Decision Tree, Random Forest, etc.

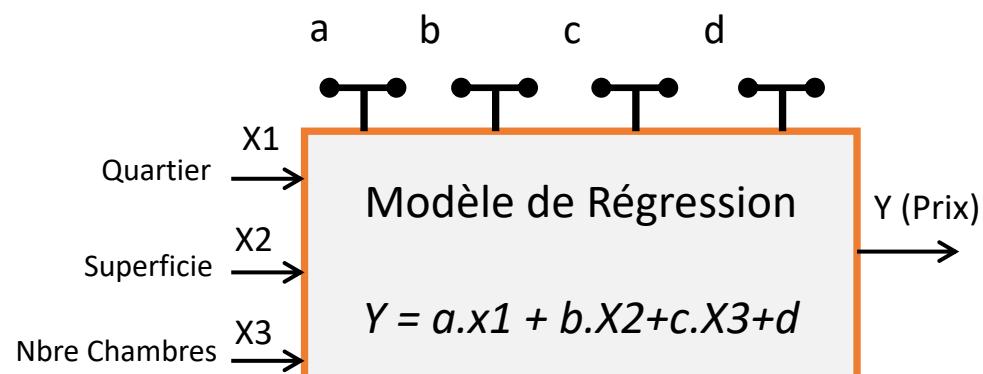
# Apprentissage Supervisé

## Régression

Exemple : Prédiction du prix d'un appartement

Data Set (Données étiquetées)

| Input       |               |               | Output       |
|-------------|---------------|---------------|--------------|
| Quartier X1 | Superficie X2 | Nbre chambres | Prix (MDH) Y |
| Maarif      | 100           | 3             | 2            |
| SBATA       | 120           | 3             | 1.1          |
| Maarif      | 100           | 4             | 2.1          |
| Bourgogne   | 80            | 3             | 1.44         |
| Maarif      | 200           | 5             | ???????      |



Modèle à 4 Paramètres a, b, c et d

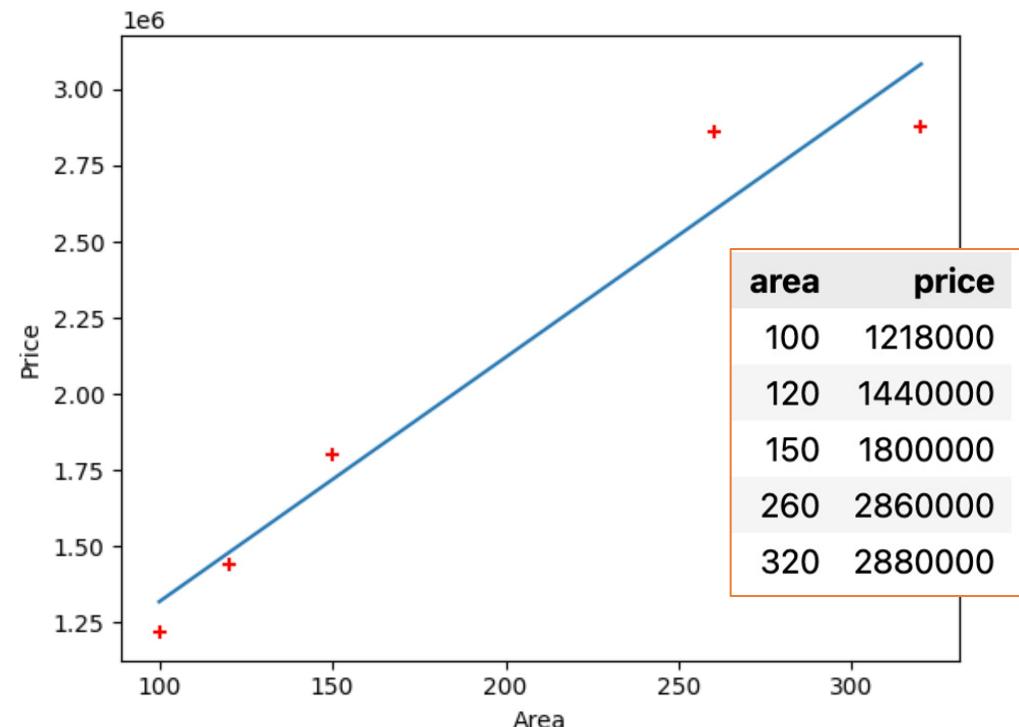
Algorithmes : Linéaire, Polynomiale, Ridge, Lasso, etc..

# Apprentissage Supervisé : Régression

- La régression consiste à déterminer une relation entre la ou les variables indépendantes et la variable dépendante.
- La régression linéaire suppose que la relation entre les variables peut être modélisée par une équation linéaire ou une équation de droite.
- La variable utilisée pour la prédiction est appelée **variable indépendante** (Features), tandis que la variable prédite est appelée **variable dépendante** (Target or Outcome).
- Dans le cas d'une régression linéaire avec une seule variable explicative, la combinaison linéaire peut s'exprimer comme suit :
- **Y = intercept + ( Coefficient \* X )**
  - X est la variable indépendante (Input)
  - Y est la variable dépendante (Output)

## Line of Best Fit

- L'objectif est de trouver la droite de régression qui s'ajuste le mieux aux données.
- Meilleur ajustement => que la ligne sera telle que la distance cumulative de tous les points par rapport à la ligne est minimisée.
- Mathématiquement, la ligne qui minimise la somme des carrés des erreurs résiduelles est appelée la droite de régression ou la ligne de meilleur ajustement.



# Apprentissage Supervisé : Evaluation de la Régression Linéaire

## • R-Squared :

- Mesure du % de variance dans la variable cible expliquée par le modèle
- Généralement la première métrique à examiner pour évaluer la performance d'un modèle linéaire
- Valeur entre 0 et 1. Plus elle est élevée, mieux c'est

## • Mean Absolute Error:

- Métrique la plus simple pour vérifier la précision des prédictions
- Même unité que la variable dépendante
- Non sensible aux valeurs aberrantes (outliers), c'est-à-dire que les erreurs n'augmentent pas trop en présence de valeurs aberrantes
- Difficile à optimiser d'un point de vue mathématique (logique purement mathématique)
- Plus la valeur est faible, mieux c'est

## • Root Mean Square Error:

- Une autre métrique pour mesurer la précision des prédictions
- Même unité que la variable dépendante
- Sensible aux valeurs aberrantes (outliers) : les erreurs sont amplifiées en raison de la fonction carrée
- Mais présente d'autres avantages mathématiques
- Plus la valeur est faible, mieux c'est

## Line of Best Fit

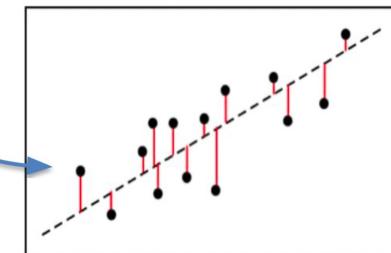
$$R^2 = 1 - \frac{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}{\frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2}$$

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

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

| Obs | Height in Inches, X | Act Weight in Pounds, Y | Predicted Weight $\hat{Y}$ | Residual / Error<br>$e_i = Y_i - \hat{Y}_i$ | Residual <sup>2</sup> / Error <sup>2</sup><br>$e_i^2 = (Y_i - \hat{Y}_i)^2$ |
|-----|---------------------|-------------------------|----------------------------|---|---|
| 1   | 63                  | 127                     | 120.1                      | 6.900                                       | 47.61   |
| 2   | 64                  | 121                     | 126.3                      | -5.300                                      | 28.09   |
| 3   | 66                  | 142                     | 138.5                      | 3.500                                       | 12.25   |
| 4   | 69                  | 157                     | 157.0                      | 0.000                                       | 0   |
| 5   | 69                  | 162                     | 157.0                      | 5.000                                       | 25  |
| 6   | 71                  | 156                     | 169.2                      | -13.200                                     | 174.24  |
| 7   | 71                  | 169                     | 169.2                      | -0.200                                      | 0.04  |
| 8   | 72                  | 165                     | 175.4                      | -10.400                                     | 108.16  |
| 9   | 73                  | 181                     | 181.5                      | -0.500                                      | 0.25  |
| 10  | 75                  | 208                     | 193.8                      | 14.200                                      | 201.64  |
|     |                     |                         |                            | 0.000                                       | 597.28  |

Sum of Squared Residuals :



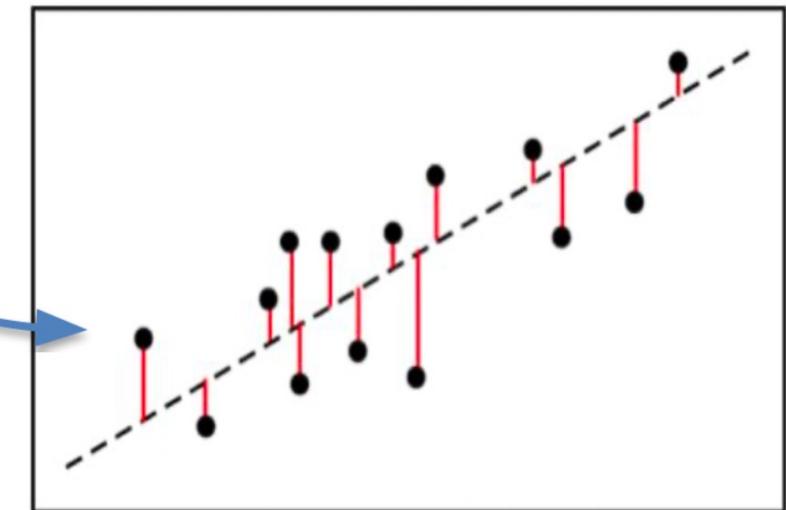
# Apprentissage Supervisé : Evaluation de la Régression Linéaire

| Obs | Height in Inches, X | Act Weight in Pounds, Y | Predicted Weight $\hat{Y}$ | Residual/Error<br>$e_i = Y_i - \hat{Y}_i$ | Residual <sup>2</sup> /Error <sup>2</sup><br>$e_i^2 = (\hat{Y}_i - Y_i)^2$ |
|-----|---------------------|-------------------------|----------------------------|---|--|
| 1   | 63                  | 127                     | 120.1                      | 6.900                                     | 47.61  |
| 2   | 64                  | 121                     | 126.3                      | -5.300                                    | 28.09  |
| 3   | 66                  | 142                     | 138.5                      | 3.500                                     | 12.25  |
| 4   | 69                  | 157                     | 157.0                      | 0.000                                     | 0  |
| 5   | 69                  | 162                     | 157.0                      | 5.000                                     | 25   |
| 6   | 71                  | 156                     | 169.2                      | -13.200                                   | 174.24   |
| 7   | 71                  | 169                     | 169.2                      | -0.200                                    | 0.04   |
| 8   | 72                  | 165                     | 175.4                      | -10.400                                   | 108.16   |
| 9   | 73                  | 181                     | 181.5                      | -0.500                                    | 0.25   |
| 10  | 75                  | 208                     | 193.8                      | 14.200                                    | 201.64   |
|     |                     |                         |                            | 0.000                                     | 597.28   |

Sum of Squared Residuals :

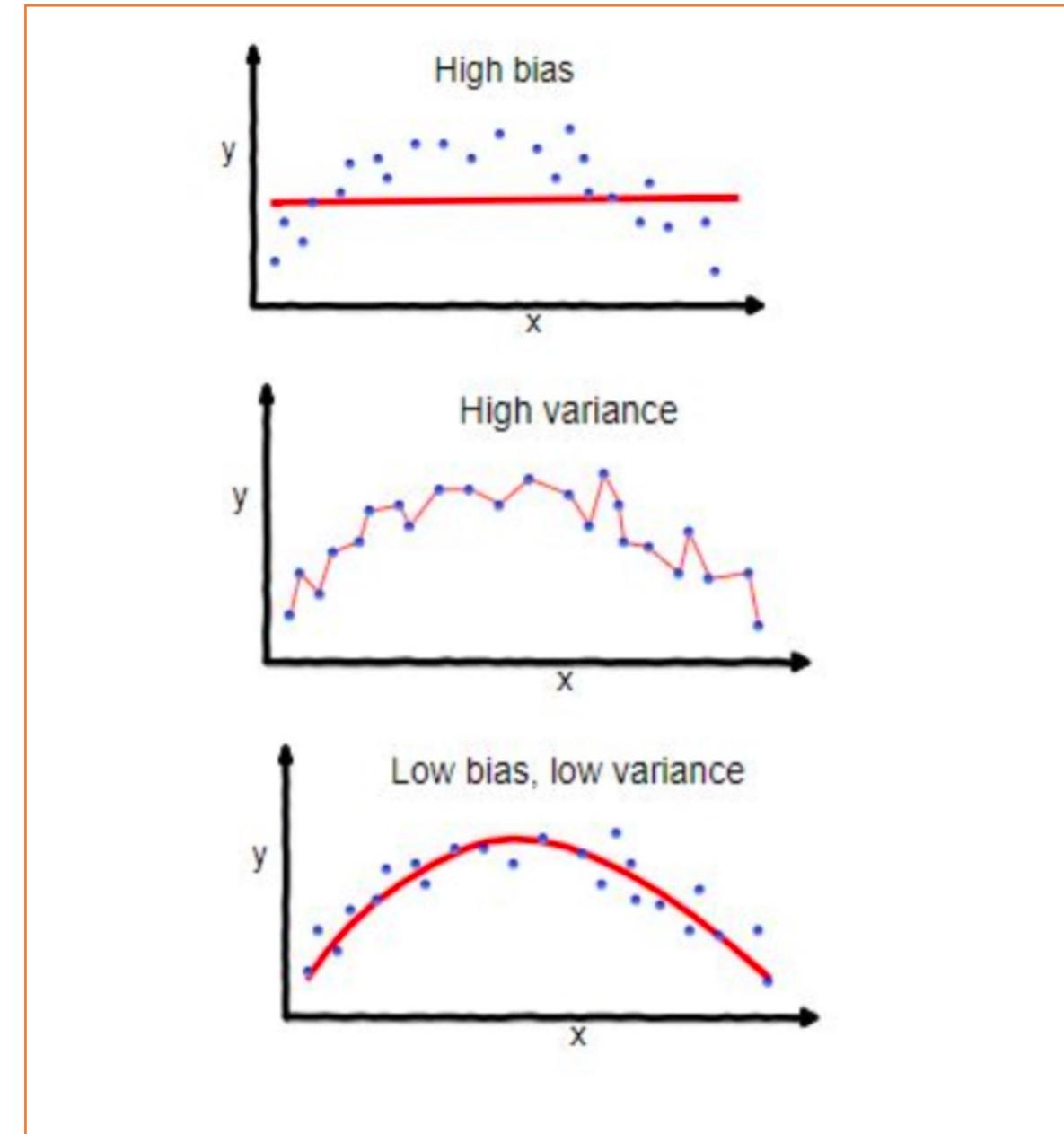
$$R^2 = 1 - \frac{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}{\frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2}$$

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



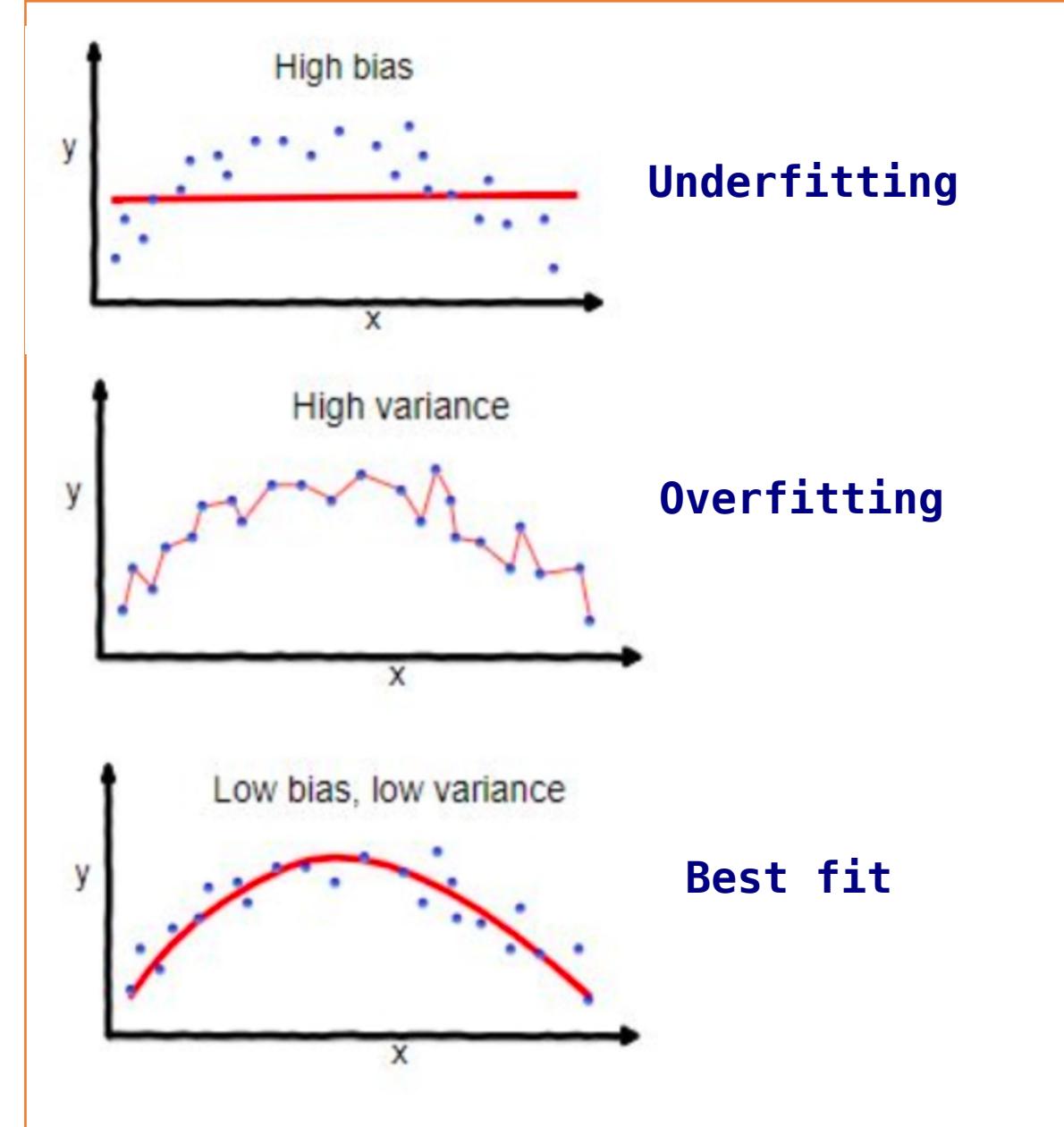
## Bias-Variance: Underfitting and Overfitting

- **Le biais et la variance** sont deux sources d'erreur clés dans les modèles d'apprentissage automatique qui impactent directement leurs performances et leur capacité de généralisation.
- **Biais** : Le biais est la différence entre la prédiction de notre modèle et la valeur correcte que nous essayons de prédire. Un modèle avec un biais élevé accorde moins d'attention aux données d'entraînement et sur-généralise, ce qui entraîne une erreur élevée sur les données d'entraînement et de test.
- **Variance** : La variance est la valeur qui indique la dispersion des données. Un modèle avec une variance élevée accorde beaucoup d'attention aux données d'entraînement en captant aussi le bruit. Ces modèles ne se généralisent pas bien sur les données de test. Par conséquent, ces modèles sont performant très bien sur les données d'entraînement mais ont une erreur élevée sur les données de test.



# Bias-Variance: Underfitting and Overfitting

- **Underfitting (Sous-ajustement)** : En apprentissage supervisé, le sous-ajustement se produit lorsqu'un modèle n'est pas capable de capturer la tendance sous-jacente des données. Ces modèles ont généralement un biais élevé et une faible variance.
- **Overfitting (Sur-ajustement)** : se produit lorsque notre modèle capture le bruit (ou les fluctuations aléatoires) en plus de la tendance sous-jacente des données. Ces modèles ont généralement un faible biais et une variance élevée.



# AI Tools

## • INSTALLATION DE PYTHON

The screenshot shows the Python.org Downloads page. At the top, there are tabs for Python, PSF, Docs, PyPI, Jobs, and Community. Below the tabs, the Python logo is displayed. A search bar with a magnifying glass icon and a 'GO' button are present. A yellow 'Donate' button is also visible. The main content area features a large illustration of two parachutes descending from the sky, carrying boxes. Text on the page includes 'Download the latest version for macOS' and 'Download Python 3.13.2'. It also mentions 'Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other' and 'Want to help test development versions of Python 3.14? Pre-releases, Docker images'.

## • INSTALLATION DE VISUAL STUDIO CODE (VS CODE)

The screenshot shows the Visual Studio Code download page. The top navigation bar includes links for Visual Studio Code, Docs, Updates, Blog, API, Extensions, FAQ, and GitHub Copilot. A search bar labeled 'Search Docs' and a 'Download' button are located at the top right. Below the navigation, there's a message about getting GitHub Copilot Free in VS Code. The main section is titled 'Download Visual Studio Code' with the subtext 'Free and built on open source. Integrated Git, debugging and extensions.' It features icons for Windows, Linux (Ubuntu), and macOS. Below each icon are download links for various package formats: Windows (User Installer, System Installer, .zip, CLI), Linux (.deb, .rpm, .tar.gz, Snap, CLI), and macOS (MacOS 10.15+, CLI). A note at the bottom states: 'User Installer x64 Arm64', 'System Installer x64 Arm64', '.zip x64 Arm64', 'CLI x64 Arm64', '.deb x64 Arm32 Arm64', '.rpm x64 Arm32 Arm64', '.tar.gz x64 Arm32 Arm64', 'Snap Snap Store', 'CLI x64 Arm32 Arm64', '.zip Intel chip Apple Silicon Universal', '.rpm Intel chip Apple Silicon', '.tar.gz Intel chip Apple Silicon', 'CLI Intel chip Apple Silicon'.

The screenshot shows the Jupyter.org/install page. The browser address bar displays 'jupyter.org/install'. The page has a header with the Jupyter logo and the title 'Installing Jupyter'. Below the title, it says 'Get up and running on your computer'. A paragraph explains that Project Jupyter's tools are available for installation via the Python Package Index. Another paragraph provides instructions for using pip, mentioning conda, mamba, pipenv, and Homebrew as alternatives for environment management.

The screenshot shows a Google Colab notebook titled 'Untitled9.ipynb'. The interface includes a toolbar with 'Fichier', 'Modifier', 'Affichage', 'Insérer', 'Exécution', 'Outils', and 'Aide'. A search bar at the top right contains the text 'Commencez à coder ou à générer avec l'IA.' Below the search bar, a button says 'Analyser des fichiers avec Gemini'. On the left side, there are icons for Commandes, Code, and Texte. The main workspace is currently empty.

# Application 1 – Régression linéaire

## ➤ Data set: `prix_maison.csv`

```
area,price  
100,1218000  
120,1440000  
150,1800000  
260,2860000  
320,2880000
```

## ➤ pip install pandas

```
import pandas as pd
```

```
[181] df = pd.read_csv('prix_maisons.csv')  
  
[182] df  
  
...  
area price  
0 100 1218000  
1 120 1440000  
2 150 1800000  
3 260 2860000  
4 320 2880000  
  
df.shape  
  
... (5, 2)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 5 entries, 0 to 4  
Data columns (total 2 columns):  
 #   Column  Non-Null Count  Dtype    
---  --      --          --  
 0   area    5 non-null      int64  
 1   price   5 non-null      int64  
dtypes: int64(2)  
memory usage: 212.0 bytes
```

## Resumé statistique

```
df.describe().T
```

|       | count | mean      | std           | min       | 25%       | 50%       | 75%       | max       |
|-------|-------|-----------|---------------|-----------|-----------|-----------|-----------|-----------|
| area  | 5.0   | 190.0     | 95.393920     | 100.0     | 120.0     | 150.0     | 260.0     | 320.0     |
| price | 5.0   | 2039600.0 | 786015.775923 | 1218000.0 | 1440000.0 | 1800000.0 | 2860000.0 | 2880000.0 |

- Les deux variables area et price sont de type numérique
- Il y a 5 valeurs pour chaque variable. Ce qui montre qu'il n'y a pas de valeurs manquante

- Le dataset contient 5 lignes et 2 colonnes

# Application 1 – Régression linéaire

## Vérification de l'existence des valeurs manquantes :

```
df.isna().sum()
```

✓ 0.0s

```
area      0  
price     0  
dtype: int64
```

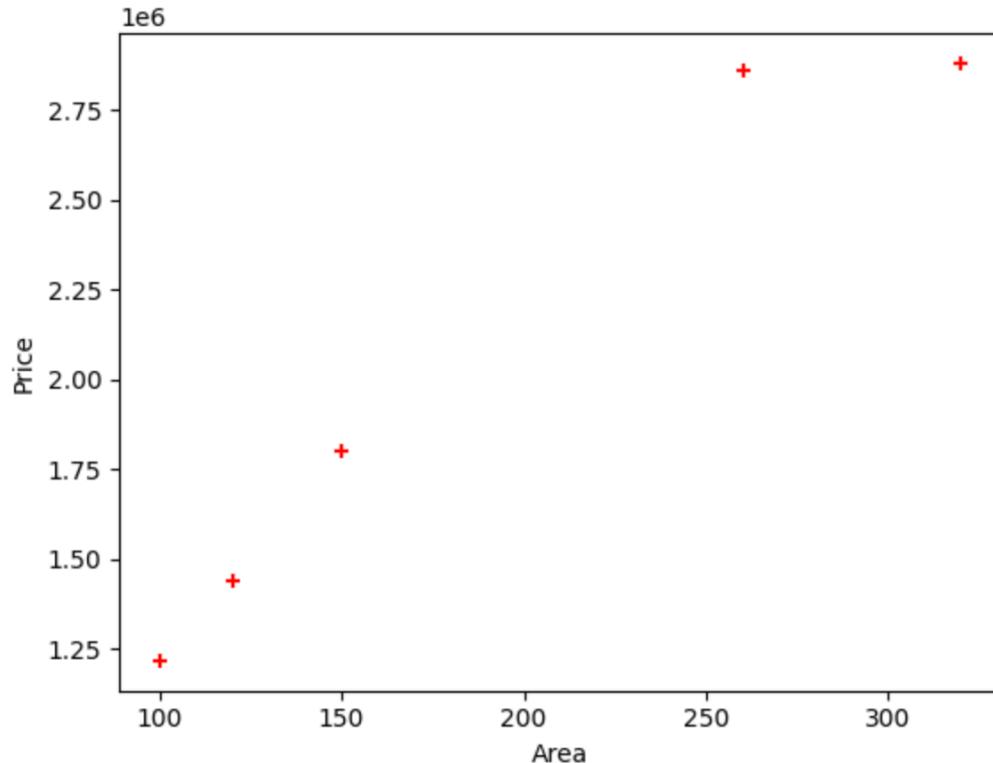
- Il n'y a aucune valeur manquante pour les deux variables

## Nuage de points avec Scatter Plot

```
plt.xlabel("Area")  
plt.ylabel("Price")  
plt.scatter(df['area'], df['price'], color='red', marker="+")
```

✓ 0.1s

```
<matplotlib.collections.PathCollection at 0x16dc3cd70>
```



# Application 1 – Régression linéaire

```
from sklearn import linear_model  
from sklearn.metrics import root_mean_squared_error, r2_score
```

## Création du modèle de régression linéaire

```
lr = linear_model.LinearRegression()
```

## Prédiction :

```
price = lr.predict([[2300]])  
print(price.tolist()[0])
```

✓ 0.0s

18976407.692307692

/Users/mohamedyoussfi/Documents/AI/Supervised Learning/Regression/.venv/lib/python3.13/s  
warnings.warn(

+ Code + Marquege

```
predicted = lr.predict(X)
```

9]

```
y = df['price']
```

```
X = df.drop(columns=['price'])
```

```
print("Predicted => ",predicted)  
print ("Target => ", y.tolist())
```

Predicted => [1317176.92307692 1477715.38461538 1718523.07692308 2601484.61538462  
3083100.]

Target => [1218000, 1440000, 1800000, 2860000, 2880000]

```
lr.fit(X,y)
```

0]

▼ LinearRegression ⓘ ?

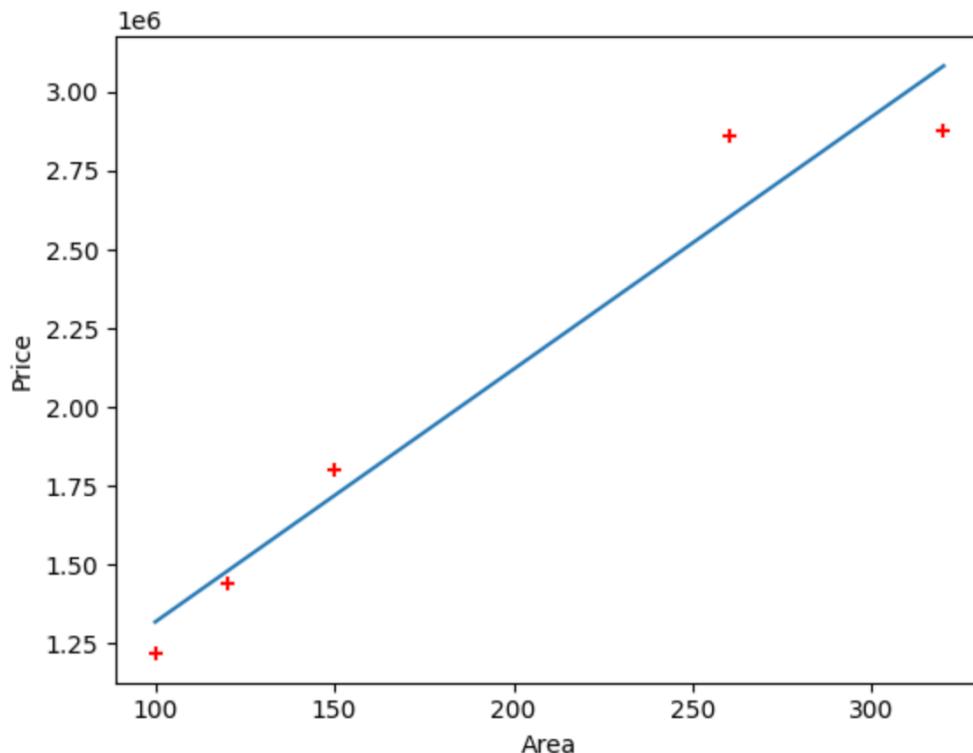
LinearRegression()

# Application 1 – Régression linéaire

## Visualiser le modèle avec les nuages de points

```
plt.xlabel("Area")
plt.ylabel("Price")
plt.scatter(df['area'], df['price'], color='red', marker="+")
plt.plot(df['area'], lr.predict(df[['area']]))

[<matplotlib.lines.Line2D at 0x29db72710>]
```



## Evaluation du modèle

```
r2_score(y, predicted)
```

```
[0] 0.9490237236328821
```

```
root_mean_squared_error(y, predicted)
```

```
[1] 158730.47305707596
```

## Expression mathématique du modèle

```
intercept= lr.intercept_
coef = lr.coef_
```

```
price1 = intercept + coef * 1000
price2 = lr.predict([[1000]])
print(price1)
print(price2)
```

```
[8541407.69230769]
[8541407.69230769]
```

# Application 2 – Régression linéaire

## ➤ Data set: prix\_maison2.csv

```
area,rooms,age,city,price  
100,3,4,Casablanca,1500000  
120,4,10,Casablanca,1680000  
260,,20,Casablanca,3380000  
190,5,1,Casablanca,3040000  
260,5,4,Casablanca,3900000  
60,2,3,Casablanca,1080000  
100,3,4,Marrakech,1400000  
120,4,10,Marrakech,1580000  
260,5,20,Marrakech,3280000  
190,5,1,Marrakech,2900000  
260,,4,Marrakech,3000000  
60,2,3,Marrakech,860000  
100,3,4,Tanger,1480000  
120,4,10,Tanger,1600000  
260,5,20,Tanger,3300000  
190,5,1,Tanger,3000000  
260,,4,Tanger,3000000  
60,2,3,Tanger,1000000
```

```
df2 = pd.read_csv('prix_maisons2.csv')
```

df2

|    | area | rooms | age | city       | price   |
|----|------|-------|-----|------------|---------|
| 0  | 100  | 3.0   | 4   | Casablanca | 1500000 |
| 1  | 120  | 4.0   | 10  | Casablanca | 1680000 |
| 2  | 260  | NaN   | 20  | Casablanca | 3380000 |
| 3  | 190  | 5.0   | 1   | Casablanca | 3040000 |
| 4  | 260  | 5.0   | 4   | Casablanca | 3900000 |
| 5  | 60   | 2.0   | 3   | Casablanca | 1080000 |
| 6  | 100  | 3.0   | 4   | Marrakech  | 1400000 |
| 7  | 120  | 4.0   | 10  | Marrakech  | 1580000 |
| 8  | 260  | 5.0   | 20  | Marrakech  | 3280000 |
| 9  | 190  | 5.0   | 1   | Marrakech  | 2900000 |
| 10 | 260  | NaN   | 4   | Marrakech  | 3000000 |
| 11 | 60   | 2.0   | 3   | Marrakech  | 860000  |
| 12 | 100  | 3.0   | 4   | Tanger     | 1480000 |
| 13 | 120  | 4.0   | 10  | Tanger     | 1600000 |
| 14 | 260  | 5.0   | 20  | Tanger     | 3300000 |
| 15 | 190  | 5.0   | 1   | Tanger     | 3000000 |
| 16 | 260  | NaN   | 4   | Tanger     | 3000000 |
| 17 | 60   | 2.0   | 3   | Tanger     | 1000000 |

df2.shape

(18, 5)

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 18 entries, 0 to 17  
Data columns (total 5 columns):  
 #   Column   Non-Null Count  Dtype    
---  --      --          --       --  
 0   area     18 non-null    int64  
 1   rooms    15 non-null    float64  
 2   age      18 non-null    int64  
 3   city     18 non-null    object  
 4   price    18 non-null    int64  
dtypes: float64(1), int64(3), object(1)  
memory usage: 852.0+ bytes
```

- 18 Lignes et 5 Colonnes

- Toutes les variables sont numériques sauf city

```
df2.describe()
```

|       | area       | rooms     | age       | price        |
|-------|------------|-----------|-----------|--------------|
| count | 18.000000  | 15.000000 | 18.000000 | 1.800000e+01 |
| mean  | 165.000000 | 3.800000  | 7.000000  | 2.276667e+06 |
| std   | 79.64997   | 1.207122  | 6.61549   | 9.947450e+05 |
| min   | 60.000000  | 2.000000  | 1.000000  | 8.600000e+05 |
| 25%   | 100.000000 | 3.000000  | 3.000000  | 1.485000e+06 |
| 50%   | 155.000000 | 4.000000  | 4.000000  | 2.290000e+06 |
| 75%   | 260.000000 | 5.000000  | 10.000000 | 3.030000e+06 |
| max   | 260.000000 | 5.000000  | 20.000000 | 3.900000e+06 |

## Application 2 – Régression linéaire

```
df2.isna().sum()
```

✓ 0.0s

```
area      0  
rooms     3  
age       0  
city      0  
price     0  
dtype: int64
```

- Il y a des valeurs manquantes pour la variable bedrooms
- Nous allons remplacer la valeur manquante par la médiane

```
median = df2['rooms'].median()  
print(median)
```

✓ 0.0s

4.0

```
df2['rooms']=df2['rooms'].fillna(median)
```

df2

|    | area | rooms | age | city       | price   |
|----|------|-------|-----|------------|---------|
| 0  | 100  | 3.0   | 4   | Casablanca | 1500000 |
| 1  | 120  | 4.0   | 10  | Casablanca | 1680000 |
| 2  | 260  | 4.0   | 20  | Casablanca | 3380000 |
| 3  | 190  | 5.0   | 1   | Casablanca | 3040000 |
| 4  | 260  | 5.0   | 4   | Casablanca | 3900000 |
| 5  | 60   | 2.0   | 3   | Casablanca | 1080000 |
| 6  | 100  | 3.0   | 4   | Marrakech  | 1400000 |
| 7  | 120  | 4.0   | 10  | Marrakech  | 1580000 |
| 8  | 260  | 5.0   | 20  | Marrakech  | 3280000 |
| 9  | 190  | 5.0   | 1   | Marrakech  | 2900000 |
| 10 | 260  | 4.0   | 4   | Marrakech  | 3000000 |
| 11 | 60   | 2.0   | 3   | Marrakech  | 860000  |
| 12 | 100  | 3.0   | 4   | Tanger     | 1480000 |
| 13 | 120  | 4.0   | 10  | Tanger     | 1600000 |
| 14 | 260  | 5.0   | 20  | Tanger     | 3300000 |
| 15 | 190  | 5.0   | 1   | Tanger     | 3000000 |
| 16 | 260  | 4.0   | 4   | Tanger     | 3000000 |
| 17 | 60   | 2.0   | 3   | Tanger     | 1000000 |

```
df2['city'].value_counts()
```

```
city  
Casablanca    6  
Marrakech    6  
Tanger        6  
Name: count, dtype: int64
```

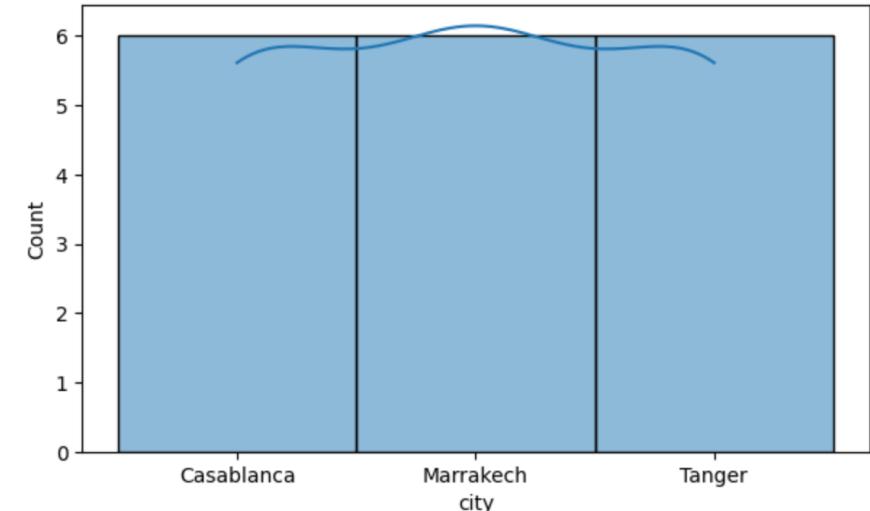
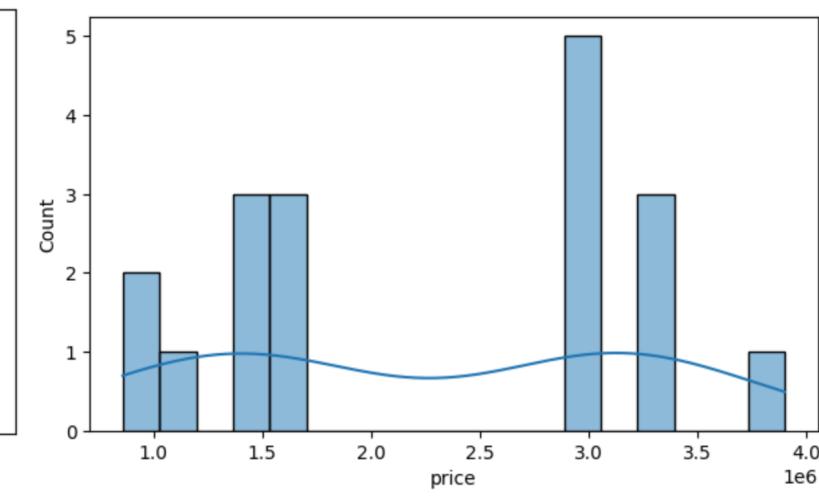
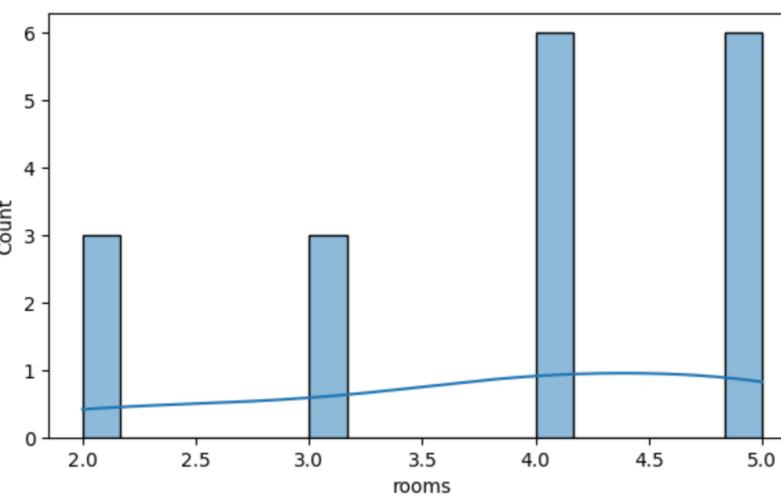
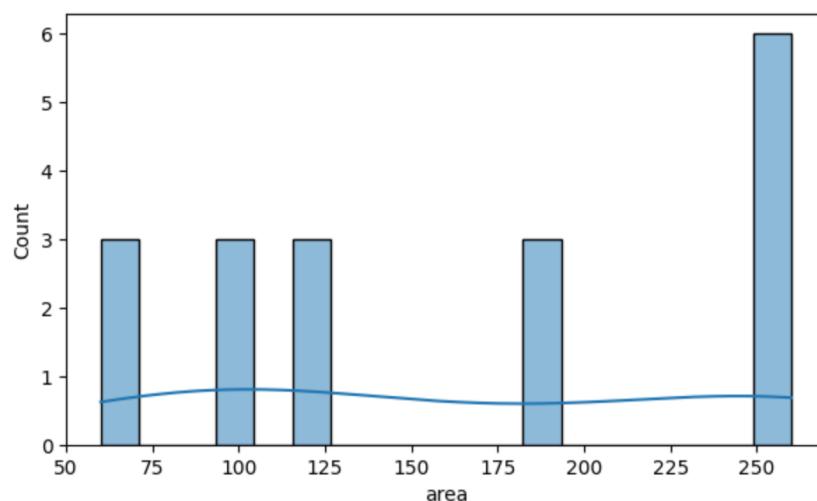
# Application 2 – Régression linéaire

```
# For Data Visualisation  
import seaborn as sns
```

## Univariate Analysis

### Check the distribution of the variables

```
[8] for col in df2.columns:  
    plt.figure(figsize = (7, 4))  
    sns.histplot(data = df2, x = col, kde = True, bins=18)  
    plt.show()
```



# Application 2 – Régression linéaire

## Categorical Variables Encoding

```
label_encoder = LabelEncoder()  
dummy_df = pd.get_dummies(df2['city'], drop_first=False, prefix='city')
```

```
#df2['city_label'] = label_encoder.fit_transform(df2['city'])  
df2 = pd.concat([df2, dummy_df], axis=1)
```

df2

|    | area | rooms | age | city       | price   | city_Casablanca | city_Marrakech | city_Tanger |
|----|------|-------|-----|------------|---------|-----------------|----------------|-------------|
| 0  | 100  | 3.0   | 4   | Casablanca | 1500000 | True            | False          | False       |
| 1  | 120  | 4.0   | 10  | Casablanca | 1680000 | True            | False          | False       |
| 2  | 260  | 4.0   | 20  | Casablanca | 3380000 | True            | False          | False       |
| 3  | 190  | 5.0   | 1   | Casablanca | 3040000 | True            | False          | False       |
| 4  | 260  | 5.0   | 4   | Casablanca | 3900000 | True            | False          | False       |
| 5  | 60   | 2.0   | 3   | Casablanca | 1080000 | True            | False          | False       |
| 6  | 100  | 3.0   | 4   | Marrakech  | 1400000 | False           | True           | False       |
| 7  | 120  | 4.0   | 10  | Marrakech  | 1580000 | False           | True           | False       |
| 8  | 260  | 5.0   | 20  | Marrakech  | 3280000 | False           | True           | False       |
| 9  | 190  | 5.0   | 1   | Marrakech  | 2900000 | False           | True           | False       |
| 10 | 260  | 4.0   | 4   | Marrakech  | 3000000 | False           | True           | False       |
| 11 | 60   | 2.0   | 3   | Marrakech  | 860000  | False           | True           | False       |
| 12 | 100  | 3.0   | 4   | Tanger     | 1480000 | False           | False          | True        |
| 13 | 120  | 4.0   | 10  | Tanger     | 1600000 | False           | False          | True        |
| 14 | 260  | 5.0   | 20  | Tanger     | 3300000 | False           | False          | True        |
| 15 | 190  | 5.0   | 1   | Tanger     | 3000000 | False           | False          | True        |
| 16 | 260  | 4.0   | 4   | Tanger     | 3000000 | False           | False          | True        |
| 17 | 60   | 2.0   | 3   | Tanger     | 1000000 | False           | False          | True        |

# Application 2 – Régression linéaire

## Bivariate Analysis

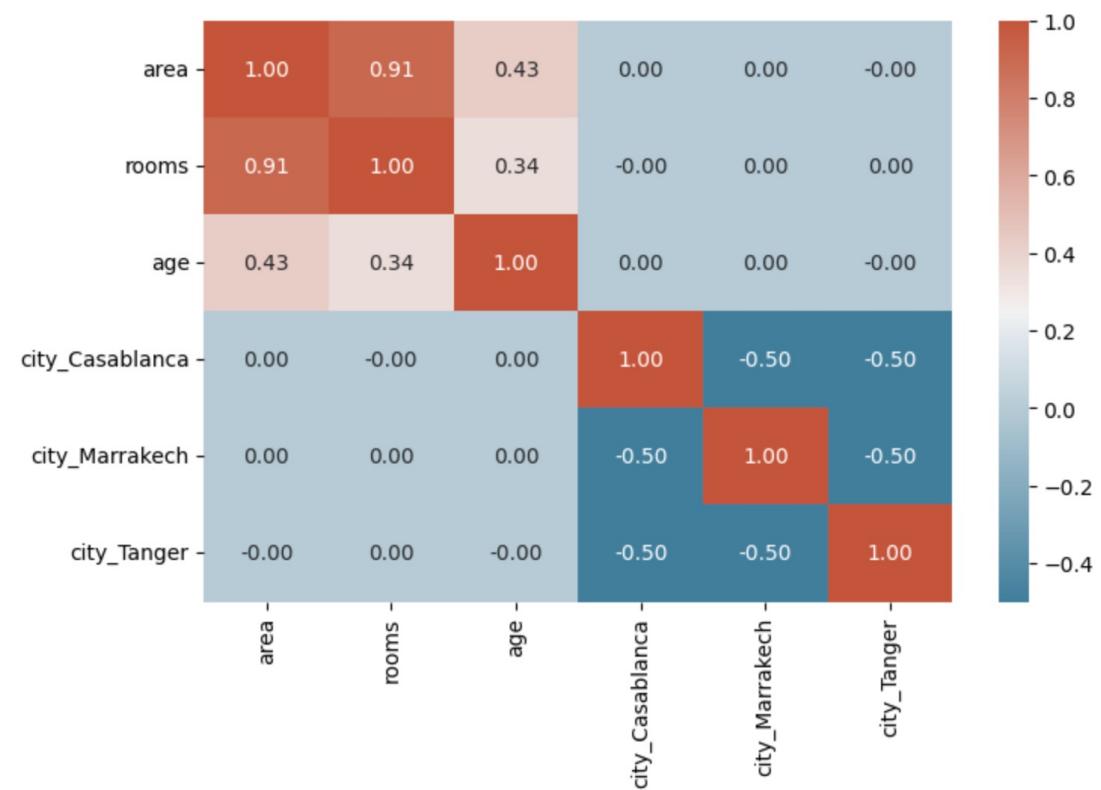
```
outcome = df2['price']
features = df2.drop(columns=['price', 'city'])
```

✓ 0.0s

Let's check the correlation using the heatmap

```
plt.figure(figsize = (8, 5))
cmap = sns.diverging_palette(230, 20, as_cmap = True)
sns.heatmap(features.corr(), annot = True, fmt = '.2f', cmap = cmap )
plt.show()
```

✓ 0.0s



- On note une forte corrélation entre area et bedrooms. Ce qui semble logique

# Application 2 – Régression linéaire

## Model Building - Approach

1. Data preparation
2. Partition the data into train and test set
3. Build model on the train data
4. Test the data on test set

### Split the dataset

Let's split the data into the dependent and independent variables and further split it into train and test set in a ratio of 70:30 for train and test set.

```
# Splitting the data  
X_train, X_test, y_train, y_test = train_test_split(features, outcome, test_size = 0.3, random_state = 1)
```

31]

Python

### Create the model

```
regressor = linear_model.LinearRegression()
```

32]

Python

### Train the model

```
regressor.fit(X_train, y_train)
```

33]

Python

▼ LinearRegression ⓘ ⓘ  
LinearRegression()

# Application 2 – Régression linéaire

## Prediction

```
preds_train = regressor.predict(X_train)
```

34]

## Model Evaluation

```
r2_score(y_train, preds_train)
```

35]

```
0.9760647063979758
```

```
preds_test = regressor.predict(X_test)
```

36]

```
r2_score(y_test, preds_test)
```

37]

```
0.9324022925175757
```

- Le R2 Score pour les données de'entrainement et pour les données de Tests sont proches. Ce qui signifie qu'il n'y a pas d'overfitting

```
regressor.coef_
```

38]

```
array([ 8626.11742769, 310819.43745123, -19758.08760919, 87415.85653553,
       -59418.96096589, -27996.89556964])
```

```
regressor.predict([[2600,3,20,False, False, True]])
```

```
regressor.intercept_
```

40]

```
np.float64(-196079.7804040648)
```

```
array([22741125.19617736])
```

# Apprentissage supervisé

## Classification

# Apprentissage supervisé : Classification

- La classification est une méthode d'apprentissage automatique supervisée dans laquelle le modèle tente de prédire l'étiquette correcte d'une donnée d'entrée.
- Dans la classification, le modèle est entièrement entraîné à l'aide des données d'apprentissage, puis il est évalué sur des données de test avant d'être utilisé pour effectuer des prédictions sur de nouvelles données inédites.
- Types de classification :
  - **Classification binaire:**
    - Exemple prédire si une transaction bancaire est frauduleuse ou non
  - **Classification multi classes :**
    - Exemple Prédire la classe d'appartenance d'un animal (Chien, Chat, Cheval)
  - **Classification multi-label :**
    - Exemple : Prédire la liste des objets qui se trouvent dans une photo ou un texte.

## Apprentissage Supervisé

### Classification

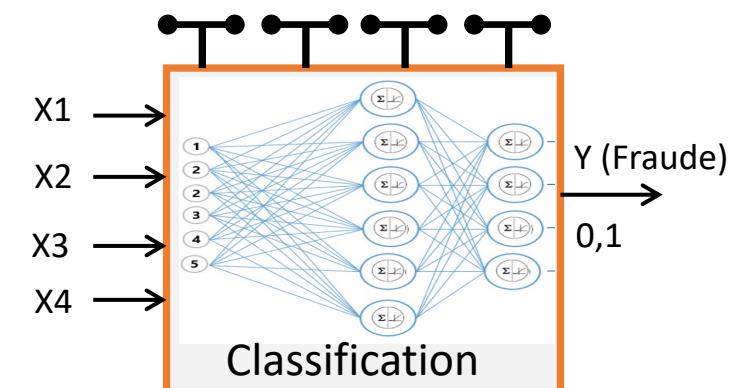
Exemple : Prédiction si une transaction est frauduleuse ou non

Data Set (Données étiquetées)

Input

Output

| Heure<br>X1 | Montant<br>X2 | Long<br>X3 | Lat<br>X4 | Fraude |
|-------------|---------------|------------|-----------|--------|
| 12:20       | 4500          | -1.2       | 3.2       | 0      |
| 01:45       | 1,60          | 0.6        | 4.3       | 1      |
| 10:08       | 100           | -2.45      | 1.3       | 0      |
| 11:55       | 80            | -1.2       | 3.2       | 1      |
| 00:00       | 3200          | -1         | 3         | ?????? |



Algorithmes : KNN, SVM, DT, RF, Réseaux de neurones, etc...

## Algorithmes de classification :

- Il existe deux types d'apprenants dans la classification de l'apprentissage automatique :
  - Les apprenants enthousiastes (Enthusiastic learners)**
    - Régression logistique.
    - Machine à vecteur de support.
    - Arbres de décision.
    - Réseaux neuronaux artificiels.
  - Les apprenants paresseux (Lazy learners)**
    - KNN (K-Nearest Neighbors) : K Voisins les plus proches

### Classification

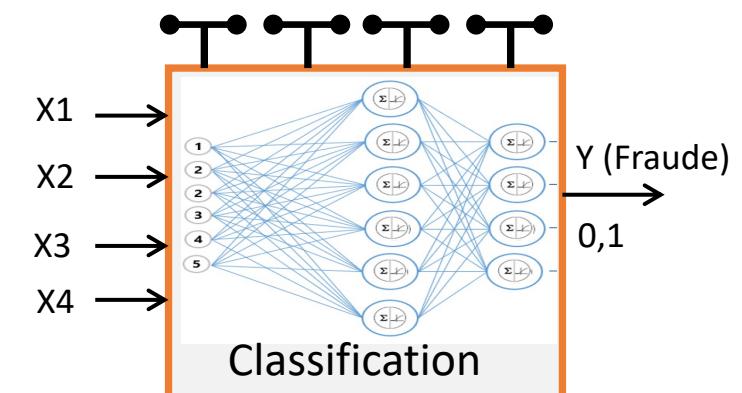
Exemple : Prédiction si une transaction est frauduleuse ou non

Data Set (Données étiquetées)

Input

Output

| Heure<br>X1 | Montant<br>X2 | Long<br>X3 | Lat<br>X4 | Fraude |
|-------------|---------------|------------|-----------|--------|
| 12:20       | 4500          | -1.2       | 3.2       | 0      |
| 01:45       | 1,60          | 0.6        | 4.3       | 1      |
| 10:08       | 100           | -2.45      | 1.3       | 0      |
| 11:55       | 80            | -1.2       | 3.2       | 1      |
| 00:00       | 3200          | -1         | 3         | ?????? |



Algorithmes : KNN, SVM, DT, RF, Réseaux de neurones, etc...

# Apprentissage supervisé : Classification

## Algorithmes de classification :

- Il existe deux types d'apprenants dans la classification de l'apprentissage automatique :
  - Les apprenants enthousiastes (Enthusiastic learners)**
  - Les apprenants paresseux (Lazy learners)**
- Les **apprenants enthousiastes** sont des algorithmes d'apprentissage automatique qui construisent d'abord un modèle à partir de l'ensemble de données d'apprentissage, puis faire des prédictions sur données les futurs.
  - Ils passent plus de temps au cours du processus d'entraînement en raison de leur volonté d'obtenir une meilleure généralisation, mais ils ont besoin de moins de temps pour faire des prédictions.
  - La plupart des algorithmes d'apprentissage automatique sont des apprenants enthousiastes, dont voici quelques exemples :
    - Régression logistique.**
    - Machine à vecteur de support.**
    - Arbres de décision.**
    - Réseaux neuronaux artificiels.**
- Les **apprenants paresseux** ou les apprenants basés sur les instances, en revanche, ne créent pas de modèle immédiatement à partir des données d'apprentissage, et c'est de là que vient l'aspect paresseux.
  - Ils se contentent de mémoriser les données d'apprentissage (Rapide)
  - Puis à chaque fois qu'il est nécessaire de faire une prédition, ils recherchent le plus proche voisin à partir de l'ensemble des données d'apprentissage, ce qui les rend **très lents lors de la prédiction**.
- Exemple :
  - KNN (K-Nearest Neighbors) : K Voisins les plus proches**

## Apprentissage Supervisé

### Classification

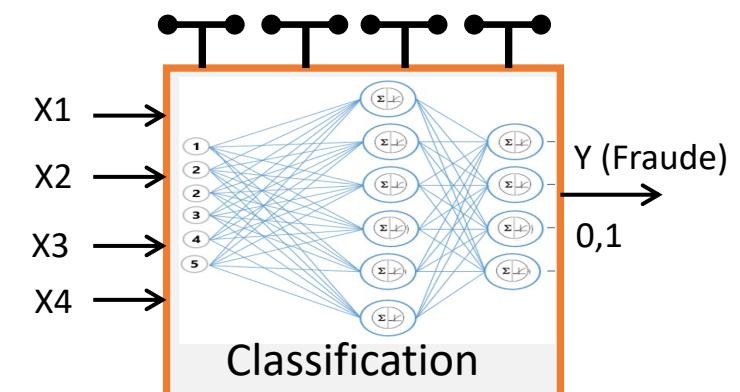
Exemple : Prédiction si une transaction est frauduleuse ou non

#### Data Set (Données étiquetées)

Input

Output

| Heure<br>X1 | Montant<br>X2 | Long<br>X3 | Lat<br>X4 | Fraude |
|-------------|---------------|------------|-----------|--------|
| 12:20       | 4500          | -1.2       | 3.2       | 0      |
| 01:45       | 1,60          | 0.6        | 4.3       | 1      |
| 10:08       | 100           | -2.45      | 1.3       | 0      |
| 11:55       | 80            | -1.2       | 3.2       | 1      |
| 00:00       | 3200          | -1         | 3         | ?????? |

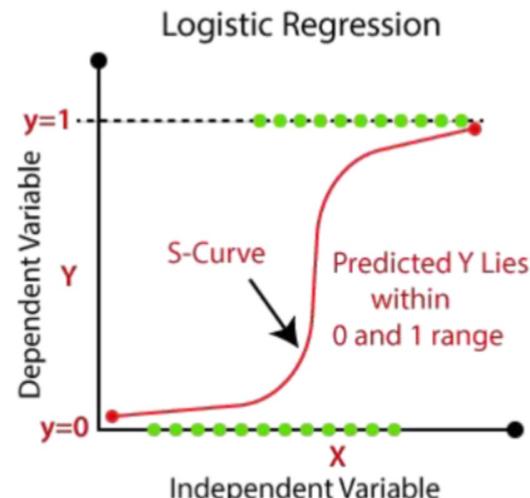
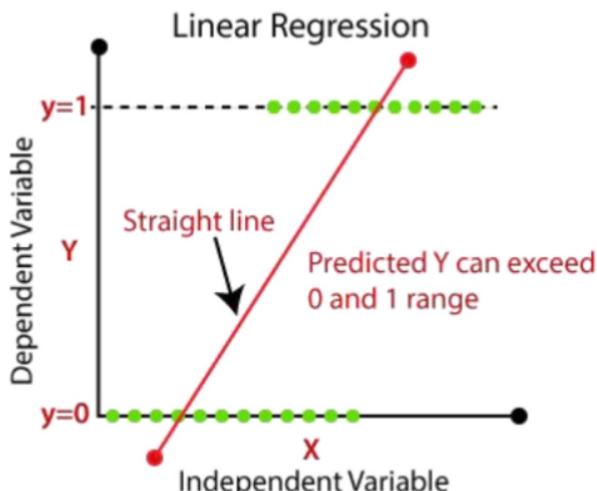


Algorithmes : KNN, SVM, DT, RF, Réseaux de neurones, etc...

# Apprentissage supervisé : Régression Logistique

- La régression logistique est un algorithme d'apprentissage supervisé utilisé pour les problèmes de classification binaire, c'est-à-dire lorsque la variable dépendante est catégorielle binaire
- Dans la régression logistique, nous utilisons la **fonction sigmoïde** pour **calculer la probabilité de la variable dépendante**.
- Exemples :
  - Prédire si un email est spam ou non
  - Prédire si un patient est malade ou non
  - Prédire si une transaction bancaire est frauduleuse ou pas

$$f(x) = \frac{1}{1 + e^{-x}}$$



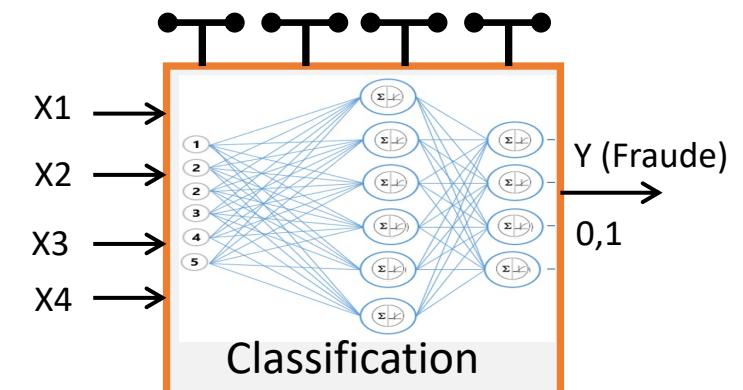
## Apprentissage Supervisé

### Classification

Exemple : Prédiction si une transaction est frauduleuse ou non

Data Set (Données étiquetées)

| Input    |            |         |        | Output  |
|----------|------------|---------|--------|---------|
| Heure X1 | Montant X2 | Long X3 | Lat X4 | Fraude  |
| 12:20    | 4500       | -1.2    | 3.2    | 0       |
| 01:45    | 1,60       | 0.6     | 4.3    | 1       |
| 10:08    | 100        | -2.45   | 1.3    | 0       |
| 11:55    | 80         | -1.2    | 3.2    | 1       |
| 00:00    | 3200       | -1      | 3      | ??????? |



Algorithmes : KNN, SVM, DT, RF, Réseaux de neurones, etc...

# Apprentissage supervisé : Classification (Evaluation)

- Matrice de confusion:**
  - La matrice de confusion est utilisée pour mesurer et évaluer les performances d'un algorithme de classification.
  - Elle permet de calculer les métriques suivantes :
    - Accuracy (Exactitude )** : Proportion des prédictions correctes par rapport au nombre total d'observations.
    - Accuracy =  $(TP + TN) / (TP + FP + FN + TN)$** 
      - TP** = Vrais Positifs (True Positives)
      - TN** = Vrais Négatifs (True Negatives)
      - FP** = Faux Positifs (False Positives)
      - FN** = Faux Négatifs (False Negatives)
    - Precision (Précision)** : Proportion de vrais positifs parmi toutes les prédictions positives, reflétant la **validité** des prédictions.
    - Precision =  $TP / (TP + FP)$**
    - Recall (Rappel)**: Proportion de vrais positifs identifiés parmi tous les cas positifs réels, mesurant l'**exhaustivité** des prédictions.
    - Recall =  $TP / (TP + FN)$**
    - F1 Score** : mesure la moyenne harmonique de la précision et du recall.
    - f1score =  $2 * precision * recall / (precision + recall)$**
  - Pour minimiser les False Positive, on se focalise sur Precision**
  - Pour minimiser les False Negative, on se focalise sur Recall**

| Input    |            |         |        |      | Output     |           |
|----------|------------|---------|--------|------|------------|-----------|
| Heure X1 | Montant X2 | Long X3 | Lat X4 | Réel | Prédiction |           |
| 12:20    | 4500       | -1.2    | 3.2    | 0    | 0          | <b>TN</b> |
| 01:45    | 1,60       | 0.6     | 4.3    | 1    | 1          | <b>TP</b> |
| 10:08    | 100        | -2.45   | 1.3    | 0    | 1          | <b>FP</b> |
| 11:55    | 80         | -1.2    | 3.2    | 1    | 0          | <b>FN</b> |
| 00:00    | 3200       | -1      | 3      | 0    | 0          | <b>TN</b> |
|          |            |         |        | 0    | 0          | <b>TN</b> |
|          |            |         |        | 1    | 1          | <b>TP</b> |

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} = \frac{2+3}{2+1+1+3} = \frac{5}{7} = 0,71 = 71\%$$

$$Precision = \frac{TP}{TP+FP} = \frac{2}{2+1} = \frac{2}{3} = 0,66 = 66\%$$

$$Recall = \frac{TP}{TP+FN} = \frac{2}{2+1} = \frac{2}{3} = 0,66 = 66\%$$

$$f1Score = \frac{2 * precision * recall}{precision + recall} = 0,68 = 68\%$$

**Matrice de confusion :**

|           |              | Actual Values |              |
|-----------|--------------|---------------|--------------|
|           |              | Positive (1)  | Negative (0) |
| Predicted | Positive (1) | 2<br>(TP)     | 1<br>(FP)    |
|           | Negative (0) | 1<br>(FN)     | 3<br>(TN)    |

# Apprentissage supervisé : Classification

## Application : Prédiction du Diabète :

- Le Jeu de données provient à l'origine du **National Institute of Diabetes and Digestive and Kidney Diseases\*\*** (États-Unis). L'objectif est de prédire, sur la base de mesures diagnostiques, si un patient est atteint de diabète.
- Contenu :** Les instances sélectionnées pour ce jeu de données respectent plusieurs critères tirés d'une base de données plus large :
  - Toutes les patientes sont des **femmes d'au moins 21 ans** d'origine **amérindienne Pima**.
- Variables :**
  - Pregnancies(Grossesses)** : Nombre de grossesses.
  - Glucose**: Concentration de glucose plasmatique (mesurée 2 heures après un test de tolérance au glucose oral).
  - BloodPressure(Pression artérielle)**: Pression artérielle diastolique (en mm Hg).
  - SkinThickness(Épaisseur de la peau)**: Épaisseur du pli cutané du triceps (en mm).
  - Insulin(Insuline)**: Taux d'insuline sérique à 2 heures (en mu U/ml).
  - BMI(IMC)**: Indice de masse corporelle (poids en kg / (taille en m)<sup>2</sup>).
  - DiabetesPedigreeFunction(Fonction pedigree du diabète)**: Score génétique reflétant l'hérédité liée au diabète.
  - Age(Âge)**: Âge (en années).
  - Outcome(Résultat)** : Variable cible (0 = non diabétique, 1 = diabétique).

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import numpy as np
```

```
# Charger les données
df = pd.read_csv('diabetes.csv')
✓ 0.0s
```

```
df.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
df.sample(10)
```

|     | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI  | DiabetesPedigreeFunction | Age   | Outcome |   |
|-----|-------------|---------|---------------|---------------|---------|------|--------------------------|-------|---------|---|
| 726 | 1           | 116     | 78            | 29            | 180     | 36.1 |                          | 0.496 | 25      | 0 |
| 227 | 3           | 162     | 52            | 38            | 0       | 37.2 |                          | 0.652 | 24      | 1 |
| 688 | 1           | 140     | 74            | 26            | 180     | 24.1 |                          | 0.828 | 23      | 0 |
| 46  | 1           | 146     | 56            | 0             | 0       | 29.7 |                          | 0.564 | 29      | 0 |
| 751 | 1           | 121     | 78            | 39            | 74      | 39.0 |                          | 0.261 | 28      | 0 |
| 69  | 4           | 146     | 85            | 27            | 100     | 28.9 |                          | 0.189 | 27      | 0 |
| 295 | 6           | 151     | 62            | 31            | 120     | 35.5 |                          | 0.692 | 28      | 0 |
| 489 | 8           | 194     | 80            | 0             | 0       | 26.1 |                          | 0.551 | 67      | 0 |
| 398 | 3           | 82      | 70            | 0             | 0       | 21.1 |                          | 0.389 | 25      | 0 |
| 668 | 6           | 98      | 58            | 33            | 190     | 34.0 |                          | 0.430 | 43      | 0 |

# Apprentissage supervisé : Classification

```
df.describe()
```

|       | Pregnancies | Glucose    | BloodPressure | SkinThickness | Insulin    | BMI        | DiabetesPedigreeFunction | Age        | Outcome    |
|-------|-------------|------------|---------------|---------------|------------|------------|--------------------------|------------|------------|
| count | 768.000000  | 768.000000 | 768.000000    | 768.000000    | 768.000000 | 768.000000 | 768.000000               | 768.000000 | 768.000000 |
| mean  | 3.845052    | 120.894531 | 69.105469     | 20.536458     | 79.799479  | 31.992578  | 0.471876                 | 33.240885  | 0.348958   |
| std   | 3.369578    | 31.972618  | 19.355807     | 15.952218     | 115.244002 | 7.884160   | 0.331329                 | 11.760232  | 0.476951   |
| min   | 0.000000    | 0.000000   | 0.000000      | 0.000000      | 0.000000   | 0.000000   | 0.078000                 | 21.000000  | 0.000000   |
| 25%   | 1.000000    | 99.000000  | 62.000000     | 0.000000      | 0.000000   | 27.300000  | 0.243750                 | 24.000000  | 0.000000   |
| 50%   | 3.000000    | 117.000000 | 72.000000     | 23.000000     | 30.500000  | 32.000000  | 0.372500                 | 29.000000  | 0.000000   |
| 75%   | 6.000000    | 140.250000 | 80.000000     | 32.000000     | 127.250000 | 36.600000  | 0.626250                 | 41.000000  | 1.000000   |
| max   | 17.000000   | 199.000000 | 122.000000    | 99.000000     | 846.000000 | 67.100000  | 2.420000                 | 81.000000  | 1.000000   |

```
cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
df[cols] = df[cols].replace(0, np.nan)
```

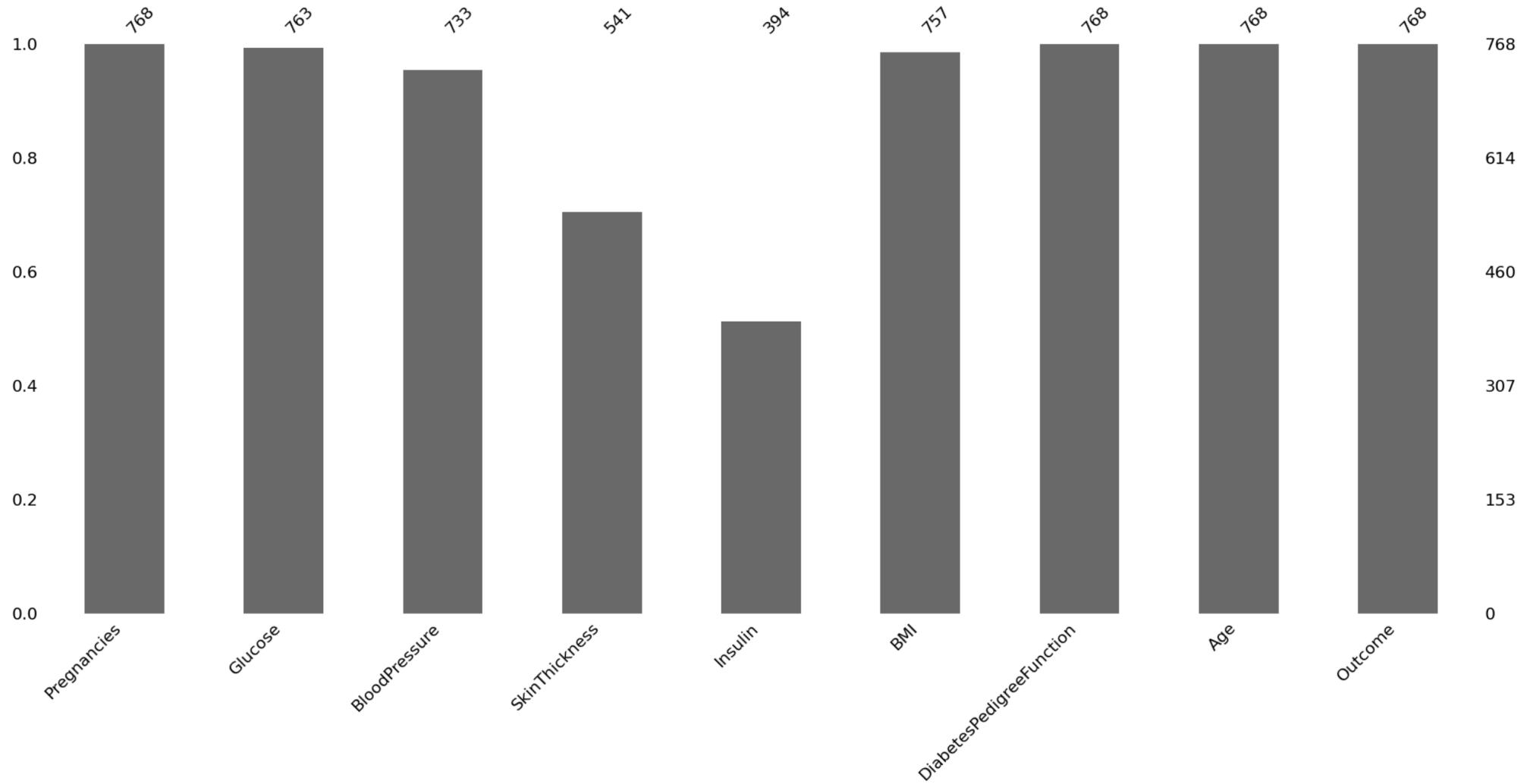
```
# Valeurs manquantes  
df.isnull().sum()
```

```
Pregnancies          0  
Glucose             5  
BloodPressure       35  
SkinThickness       227  
Insulin             374  
BMI                11  
DiabetesPedigreeFunction 0  
Age                0  
Outcome            0  
dtype: int64
```

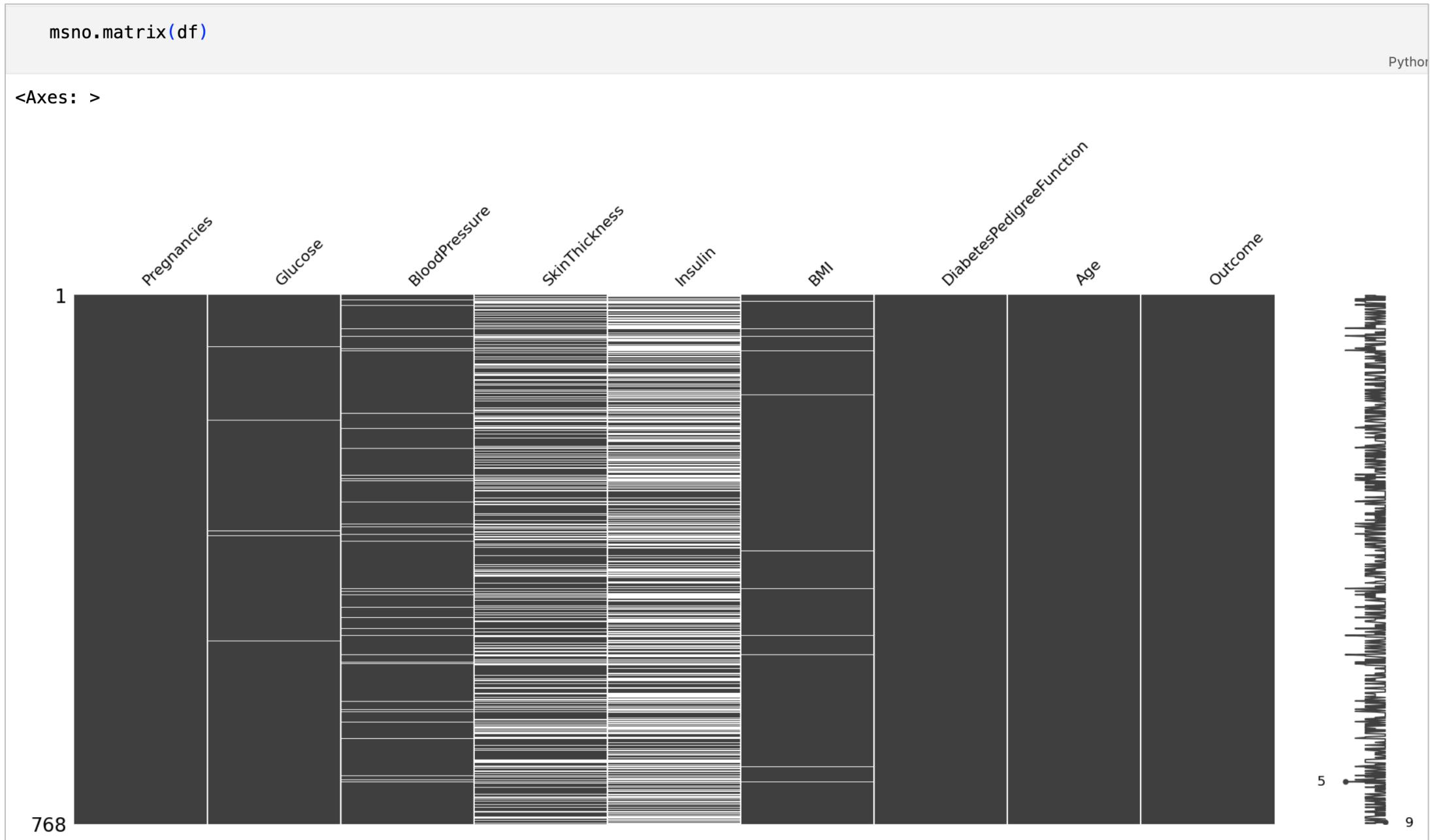
# Apprentissage supervisé : Classification

```
# Plotting  
import missingno as msno  
msno.bar(df);
```

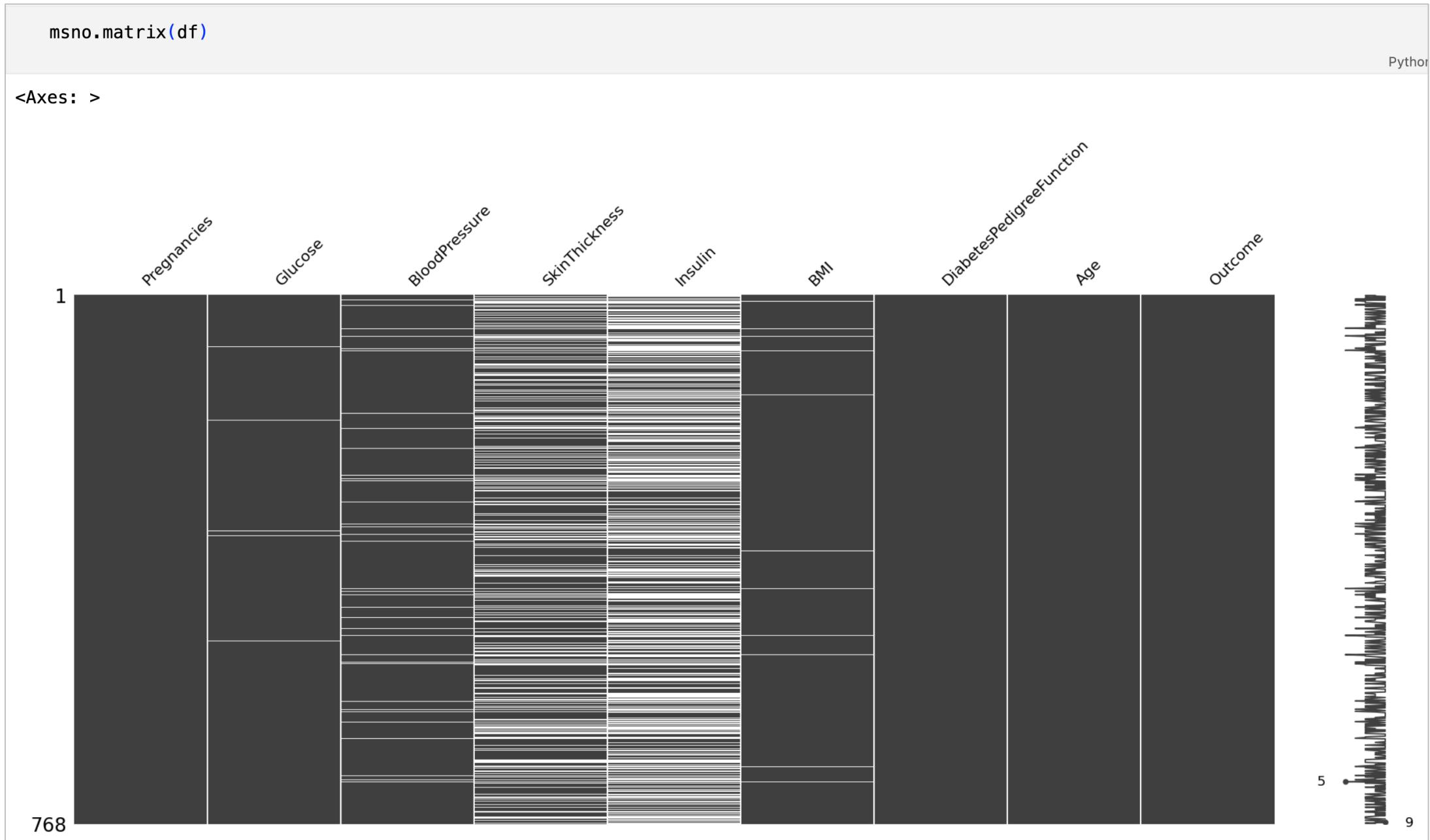
Python



# Apprentissage supervisé : Classification



# Apprentissage supervisé : Classification



# Apprentissage supervisé : Classification

```
# The values to be given for incomplete observations are given the median value of people who are not sick
# and the median values of people who are sick.
columns = df.columns
columns = columns.drop("Outcome")
for col in columns:
    median = df[col].median()
    df[col]=df[col].fillna(median)

df.describe()

Pregnancies      Glucose      BloodPressure      SkinThickness      Insulin      BMI      DiabetesPedigreeFunction      Age      Outcome
count    768.000000    768.000000    768.000000    768.000000    768.000000    768.000000          768.000000    768.000000    768.000000
mean     3.845052   121.656250    72.386719    29.108073   140.671875   32.455208          0.471876   33.240885   0.348958
std      3.369578   30.438286   12.096642     8.791221   86.383060    6.875177          0.331329   11.760232   0.476951
min      0.000000   44.000000   24.000000     7.000000   14.000000   18.200000          0.078000   21.000000   0.000000
25%     1.000000   99.750000   64.000000    25.000000   121.500000   27.500000          0.243750   24.000000   0.000000
50%     3.000000  117.000000   72.000000    29.000000   125.000000   32.300000          0.372500   29.000000   0.000000
75%     6.000000  140.250000   80.000000    32.000000   127.250000   36.600000          0.626250   41.000000   1.000000
max    17.000000  199.000000  122.000000    99.000000   846.000000   67.100000          2.420000   81.000000   1.000000

df.isnull().sum()

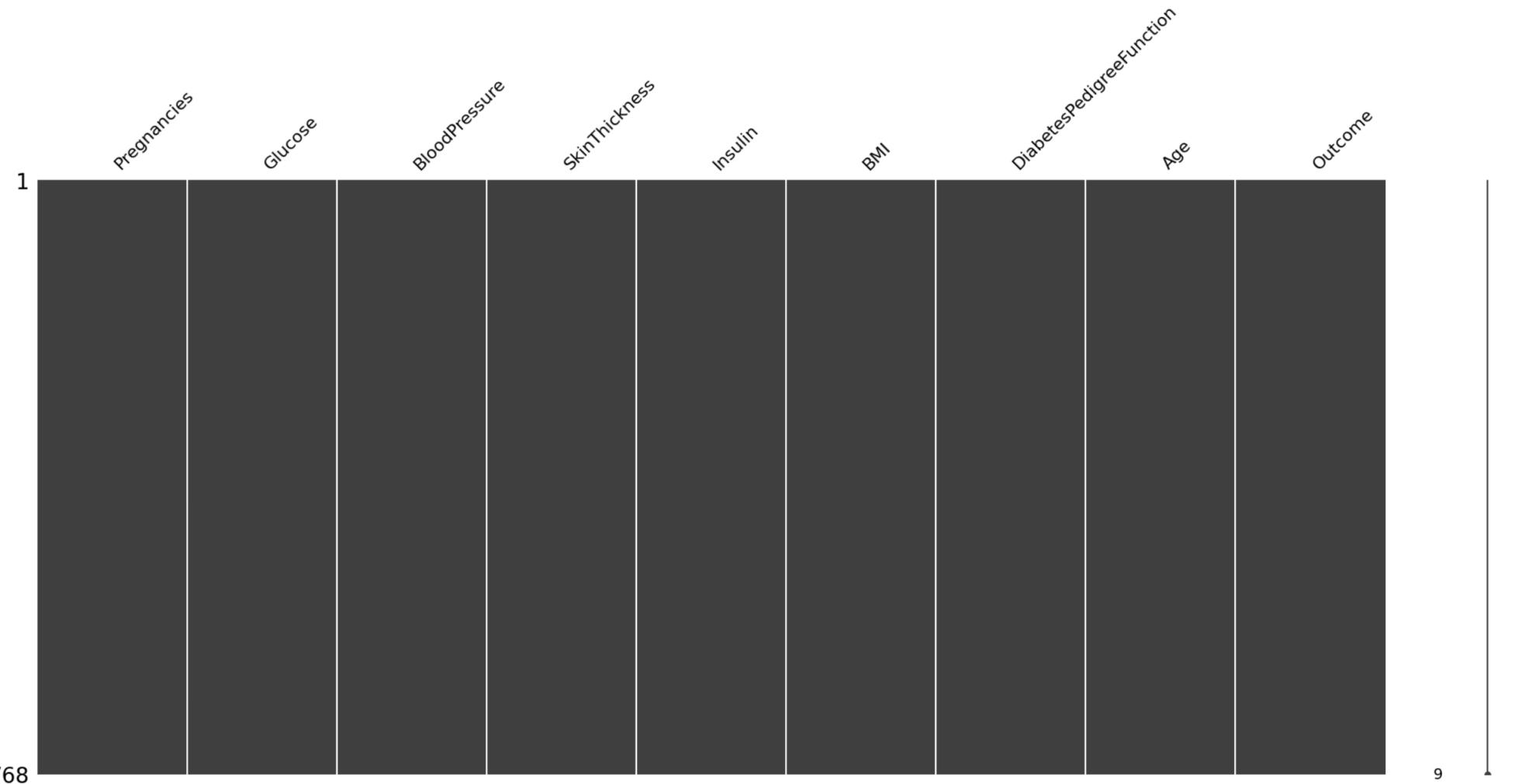
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
DiabetesPedigreeFunction 0
Age              0
Outcome          0
dtype: int64
```

# Apprentissage supervisé : Classification

```
msno.matrix(df)
```

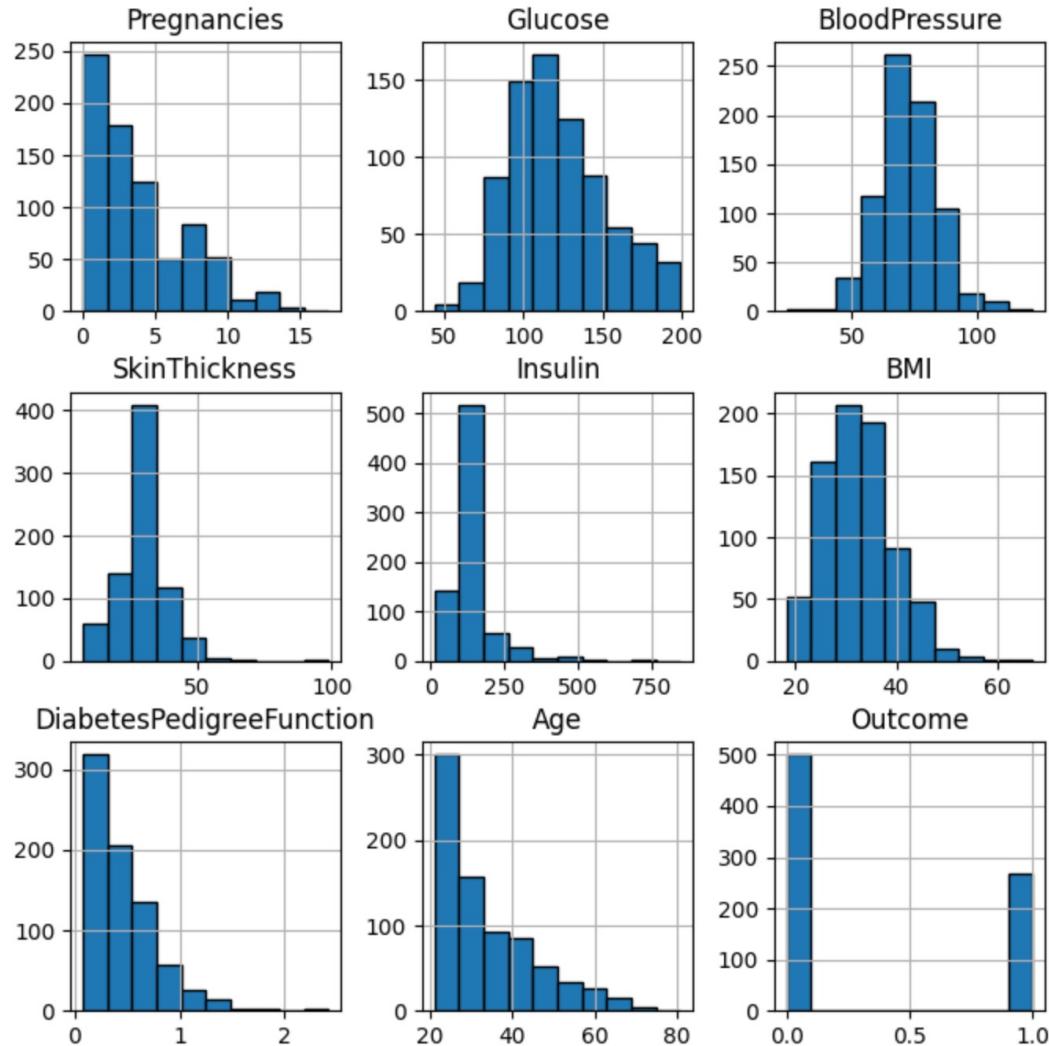
Python

<Axes: >

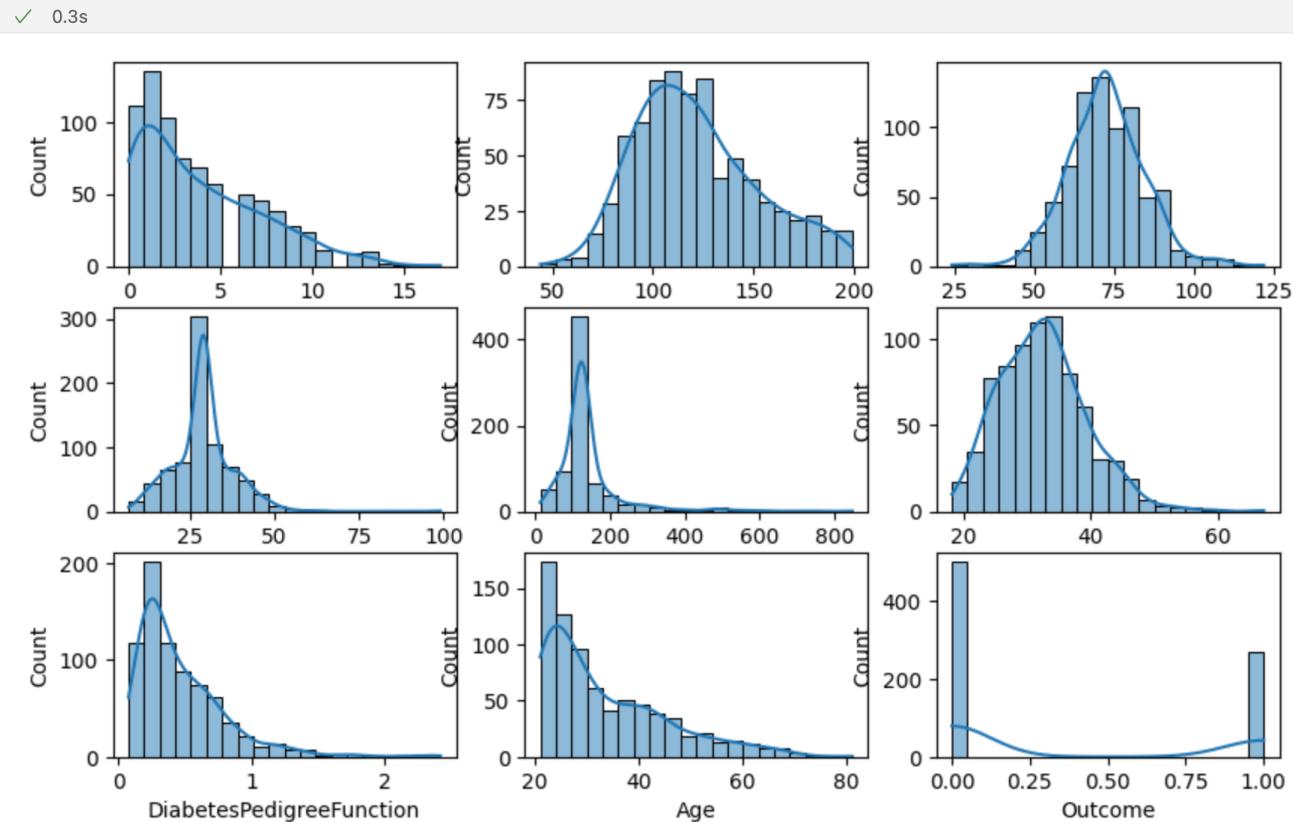


# Apprentissage supervisé : Classification

```
# Creating histograms
df.hist(figsize=(8,8), edgecolor = "black")
plt.show()
✓ 0.2s
```



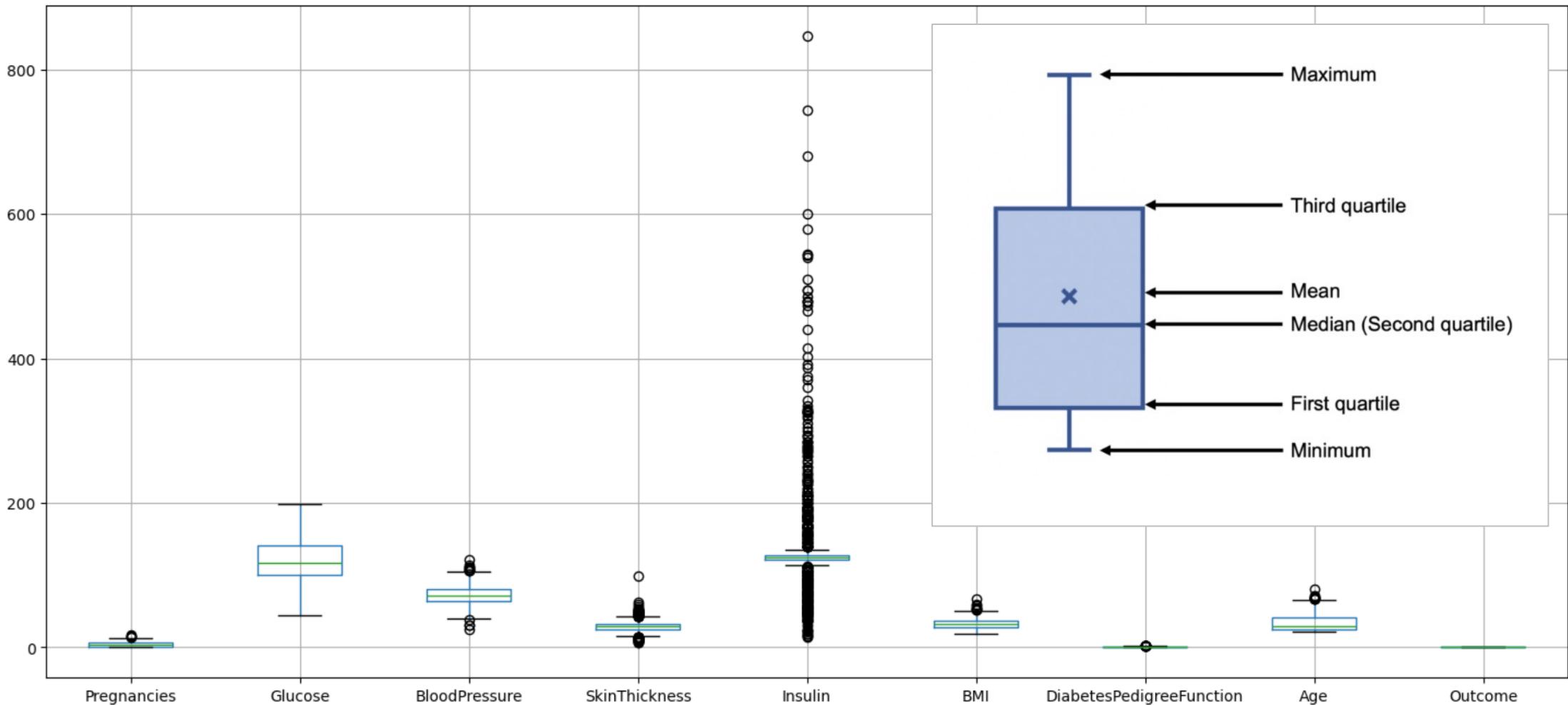
```
cols = df.columns
fig, ax = plt.subplots(3,3, figsize=(10,6))
row_index = 0
col_index = 0
for col in cols :
    sns.histplot(df[col], bins = 20, ax=ax[row_index,col_index], kde=True)
    col_index = col_index + 1
    if(col_index == 3):
        col_index=0
        row_index = row_index + 1
```



# Apprentissage supervisé : Classification

```
df.boxplot(figsize=(18,8))  
#plt.figure(figsize=(18,10))  
#sns.boxplot(df, orient='h')  
plt.show()
```

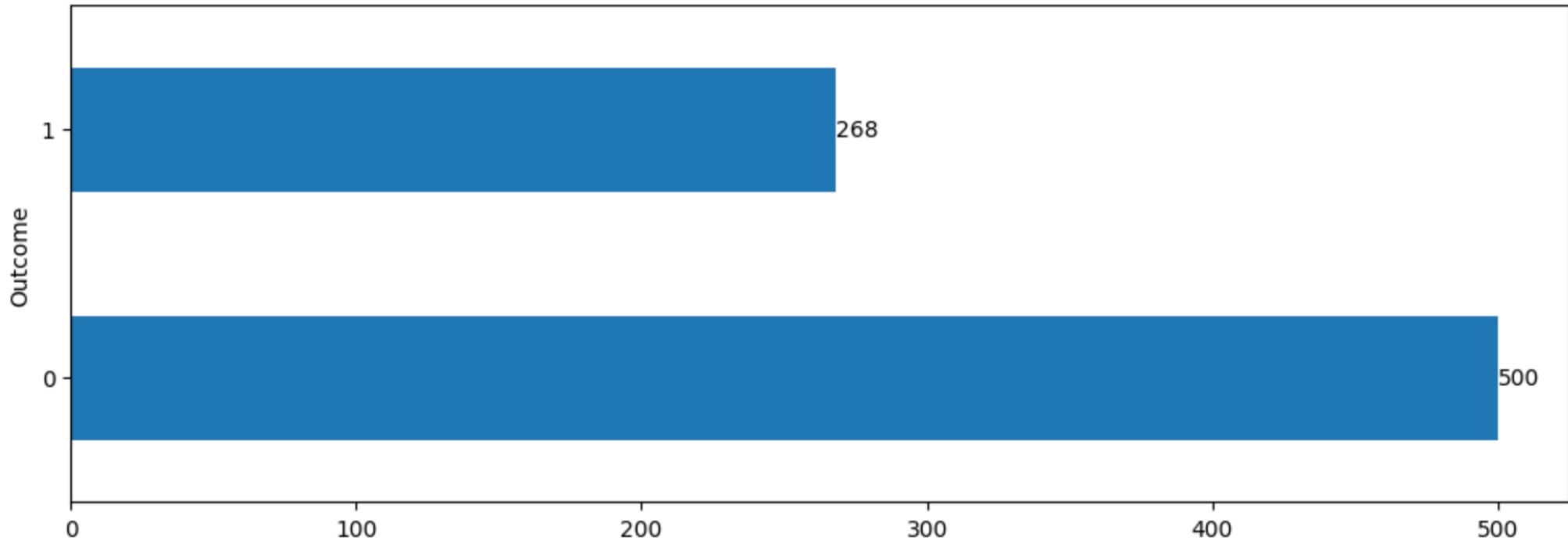
Python



# Apprentissage supervisé : Classification

```
bar=df['Outcome'].value_counts().plot.barh(figsize=(12,4))  
bar=bar.bar_label(bar.containers[0], fontsize=10)
```

✓ 0.0s



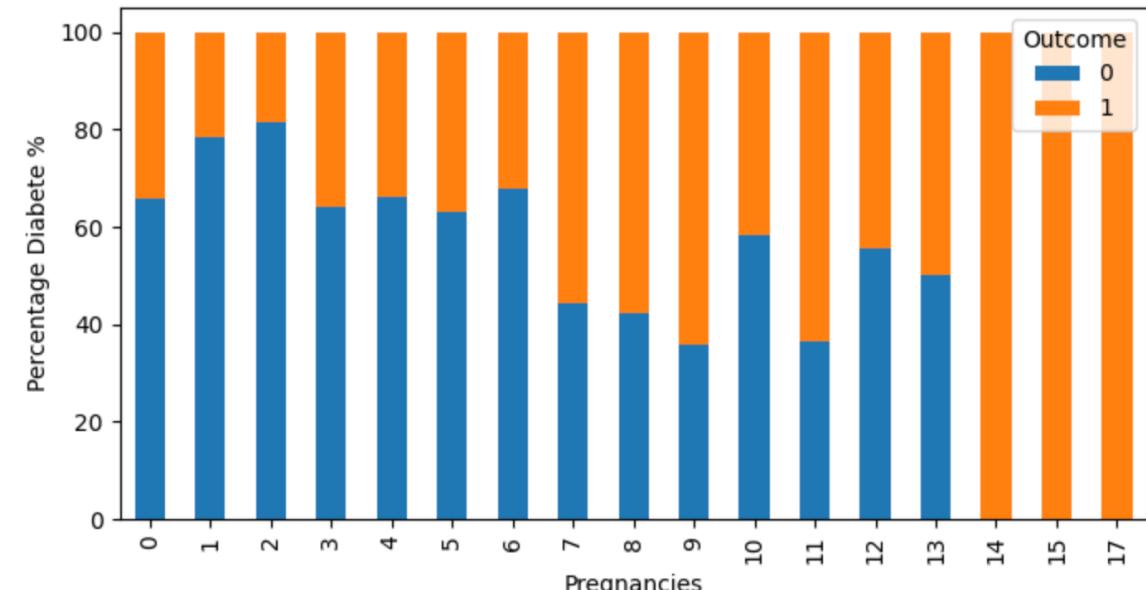
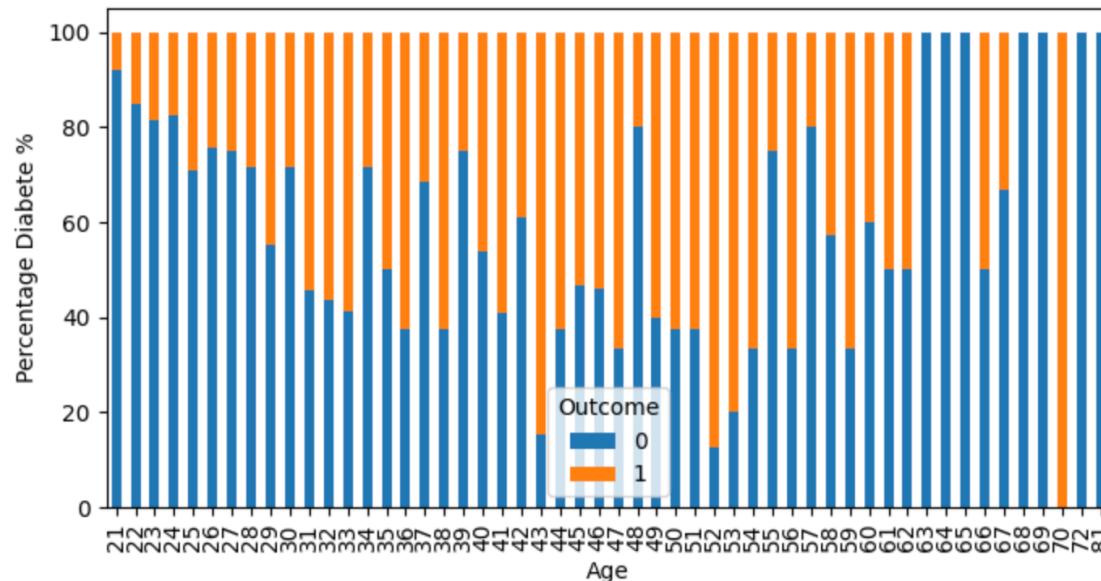
# Apprentissage supervisé : Classification

```
pd.crosstab(df['Age'],df['Outcome'], normalize='index')*100
```

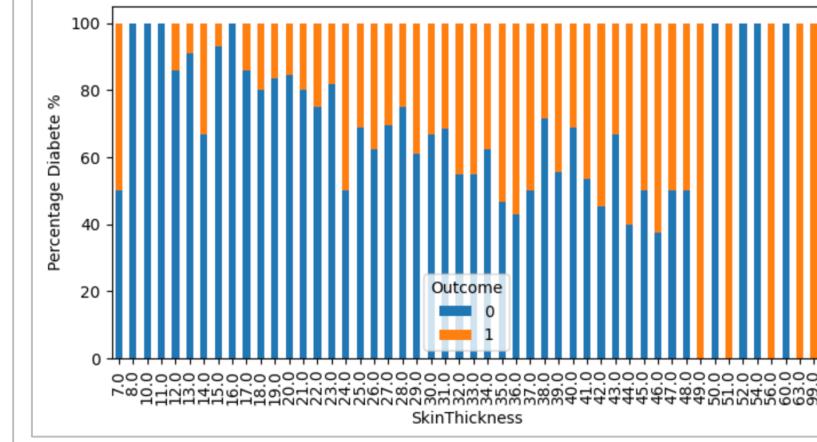
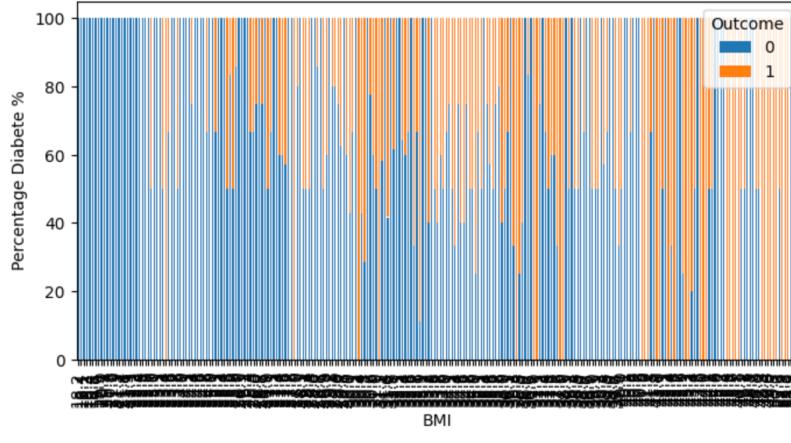
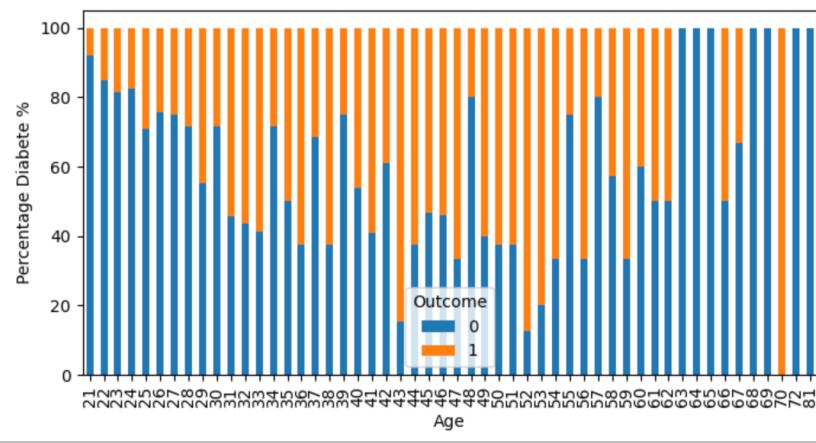
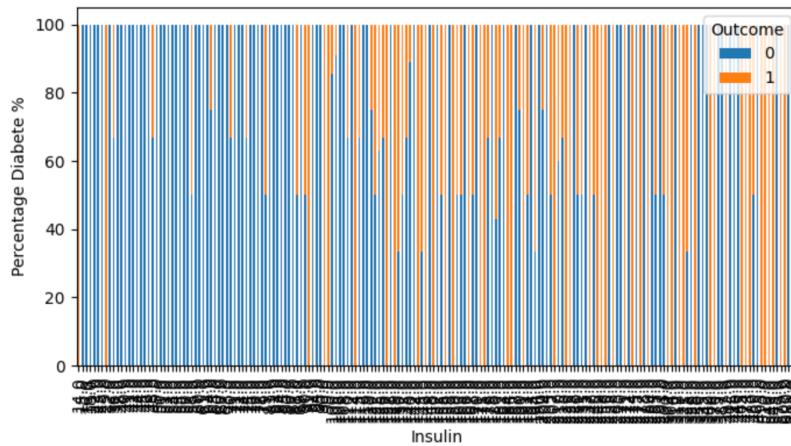
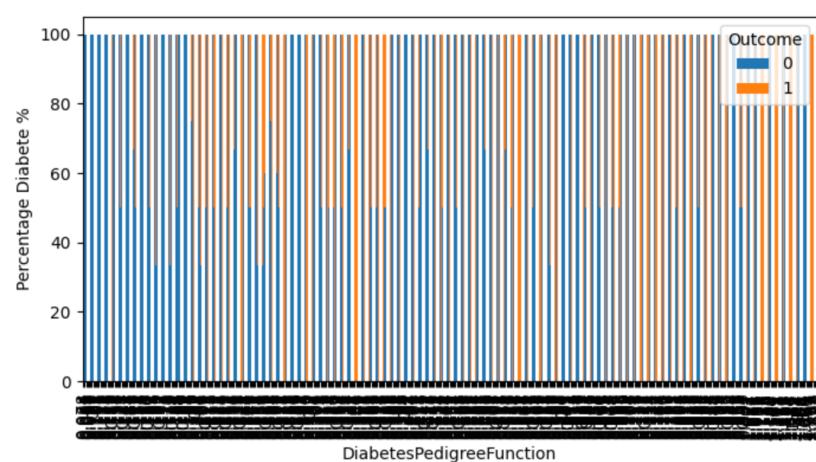
✓ 0.0s

| Outcome | 0         | 1         |
|---------|-----------|-----------|
| Age     |           |           |
| 21      | 92.063492 | 7.936508  |
| 22      | 84.722222 | 15.277778 |
| 23      | 81.578947 | 18.421053 |
| 24      | 82.608696 | 17.391304 |

```
for i in df.columns:  
    if i!='Outcome':  
        (pd.crosstab(  
            df[i],df['Outcome'],normalize='index')*100).plot(  
                kind='bar',  
                figsize=(8,4),stacked=True)  
plt.ylabel('Percentage Diabete %')
```



# Apprentissage supervisé : Classification



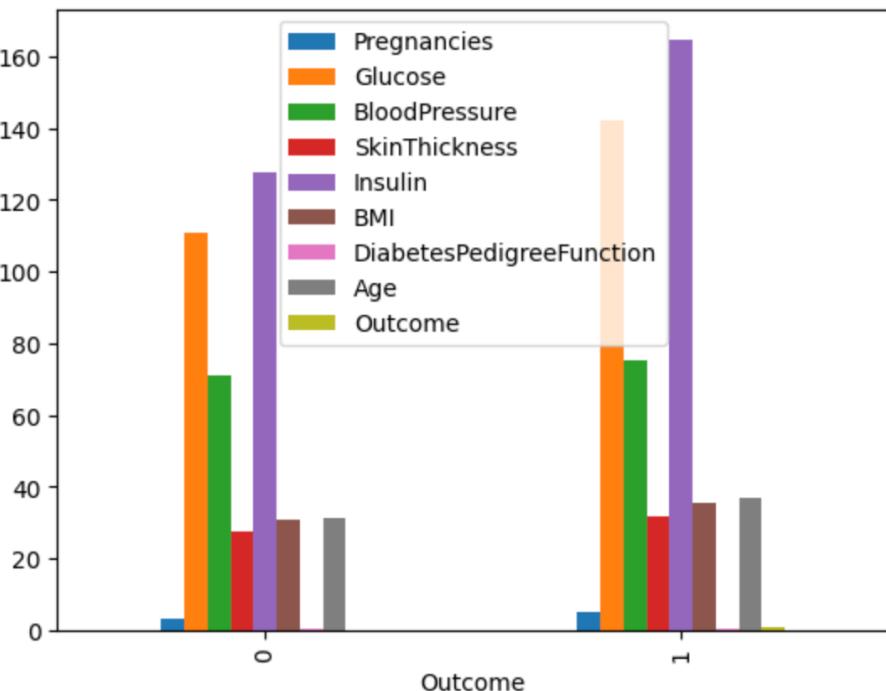
# Apprentissage supervisé : Classification

```
df.groupby(['Outcome']).mean()
```

|         | Pregnancies | Glucose    | BloodPressure | SkinThickness | Insulin    | BMI       | DiabetesPedigreeFunction | Age       | Outcome |
|---------|-------------|------------|---------------|---------------|------------|-----------|--------------------------|-----------|---------|
| Outcome |             |            |               |               |            |           |                          |           |         |
| 0       | 3.298000    | 110.682000 | 70.920000     | 27.726000     | 127.792000 | 30.885600 | 0.429734                 | 31.190000 | 0.0     |
| 1       | 4.865672    | 142.130597 | 75.123134     | 31.686567     | 164.701493 | 35.383582 | 0.550500                 | 37.067164 | 1.0     |

```
df.groupby(['Outcome']).mean().plot.bar()
```

<Axes: xlabel='Outcome'>



# Apprentissage supervisé : Classification

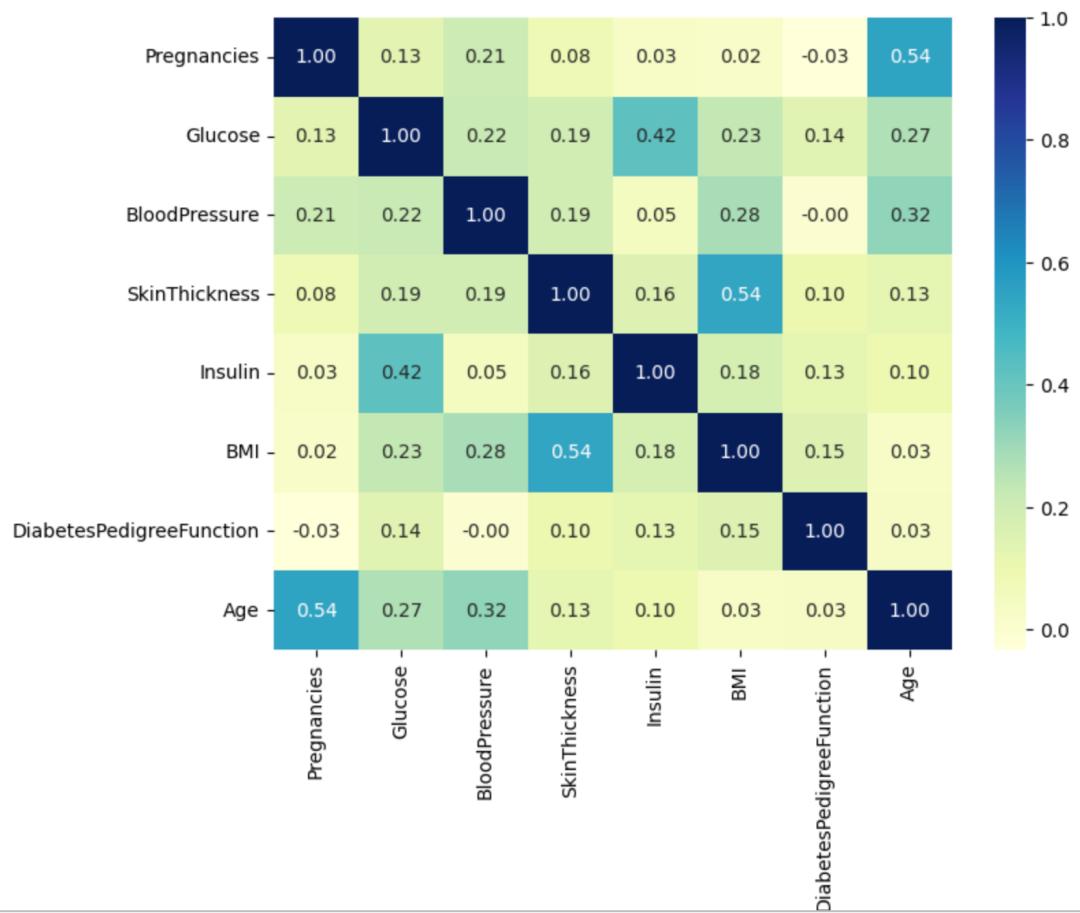
```
y = df['Outcome']
X= df.drop(columns=['Outcome'])

✓ 0.0s
```

```
# Plotting the correlation between numerical variables
plt.figure(figsize=(8,6))
sns.heatmap(X.corr(), annot=True, fmt='0.2f', cmap='YlGnBu')

✓ 0.1s
```

<Axes: >



# Apprentissage supervisé : Classification

## Splitting the data into 70% train and 30% test set

Some classification problems can exhibit a large imbalance in the distribution of the target classes: for instance there could be several times more negative samples than positive samples. In such cases it is recommended to use the **stratified sampling** technique to ensure that relative class frequencies are approximately preserved in each train and validation fold.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=16)
```

28]

Python

```
# Scaling the data
sc=StandardScaler()

# Fit_transform on train data
X_train_scaled=sc.fit_transform(X_train)
X_train_scaled=pd.DataFrame(X_train_scaled, columns=X.columns)

# Transform on test data
X_test_scaled=sc.transform(X_test)
X_test_scaled=pd.DataFrame(X_test_scaled, columns=X.columns)
```

29]

Python

# Apprentissage supervisé : Classification (Logistic Regression)

## Logistic Regression Model

- Logistic Regression is a supervised learning algorithm which is used for **binary classification problems** i.e. where the dependent variable is categorical and has only two possible values. In logistic regression, we use the sigmoid function to calculate the probability of an event  $y$ , given some features  $x$  as:

$$P(y) = \frac{1}{1 + e^{-x}}$$

```
model_lgr = LogisticRegression(max_iter=1000)
```

30] Python

```
model_lgr.fit(X_train_scaled,y_train)
```

31] Python

```
LogisticRegression(max_iter=1000)
```

```
y_pred_train = model_lgr.predict(X_train_scaled)
y_pred_test = model_lgr.predict(X_test_scaled)
```

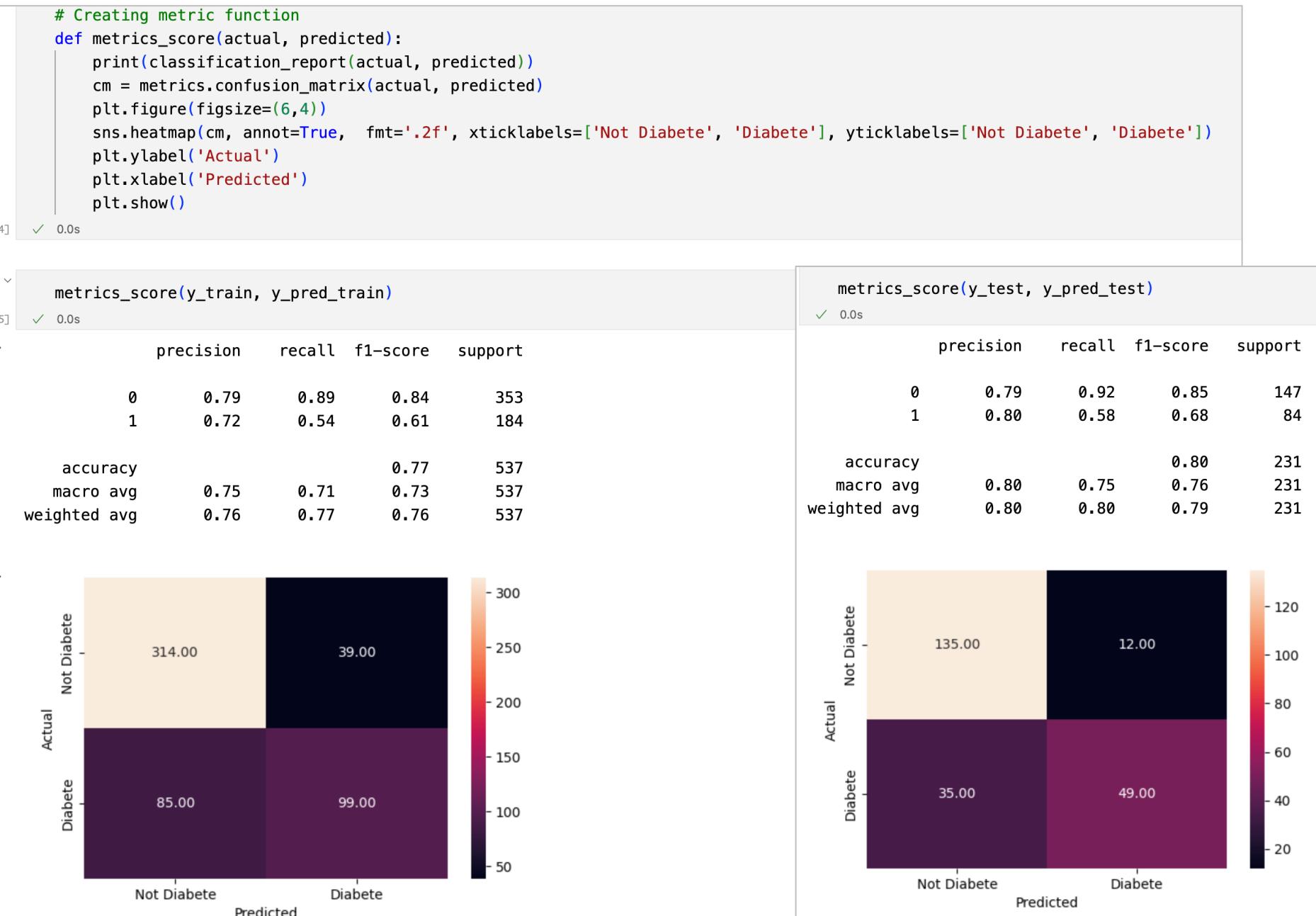
32] Python

```
print("Accuracy",metrics.accuracy_score(y_train, y_pred_train))
print("Precision",metrics.precision_score(y_train, y_pred_train))
print("Recall",metrics.recall_score(y_train, y_pred_train))
```

33] Python

```
Accuracy 0.7690875232774674
Precision 0.717391304347826
Recall 0.5380434782608695
```

# Apprentissage supervisé : Classification



# Apprentissage supervisé : Classification

```
cols=X.columns  
coef_lg=model_lgr.coef_  
coefs=pd.DataFrame(coef_lg,columns=cols).T.sort_values(by = 0,ascending = False)  
coefs
```

✓ 0.0s

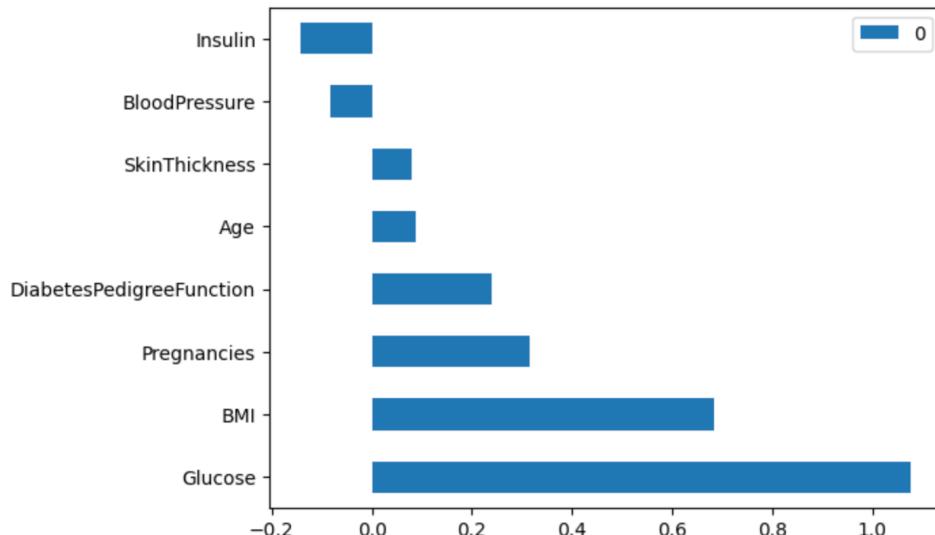
0

|                          |           |
|--------------------------|-----------|
| Glucose                  | 1.077539  |
| BMI                      | 0.683475  |
| Pregnancies              | 0.316472  |
| DiabetesPedigreeFunction | 0.238285  |
| Age                      | 0.088295  |
| SkinThickness            | 0.080405  |
| BloodPressure            | -0.084081 |
| Insulin                  | -0.143691 |

```
coefs.plot.bahr()
```

✓ 0.0s

<Axes: >



```
import numpy as np  
# Finding the odds  
odds = np.exp(model_lgr.coef_[0])
```

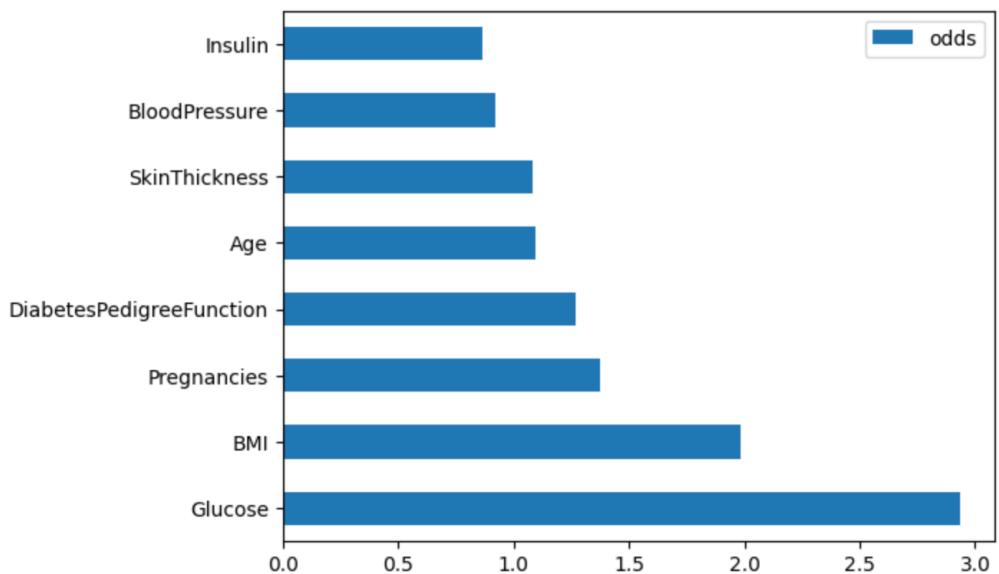
```
# Adding the odds to a dataframe and sorting the values  
df_odds=pd.DataFrame(odds, X_train_scaled.columns, columns = ['odds']).sort_values(by ='odds', ascending = False)  
df_odds
```

odds

| Feature                  | Odds     |
|--------------------------|----------|
| Glucose                  | 2.937441 |
| BMI                      | 1.980749 |
| Pregnancies              | 1.372278 |
| DiabetesPedigreeFunction | 1.269071 |
| Age                      | 1.092310 |
| SkinThickness            | 1.083725 |
| BloodPressure            | 0.919356 |
| Insulin                  | 0.866155 |

```
df_odds.plot.bahr()
```

<Axes: >

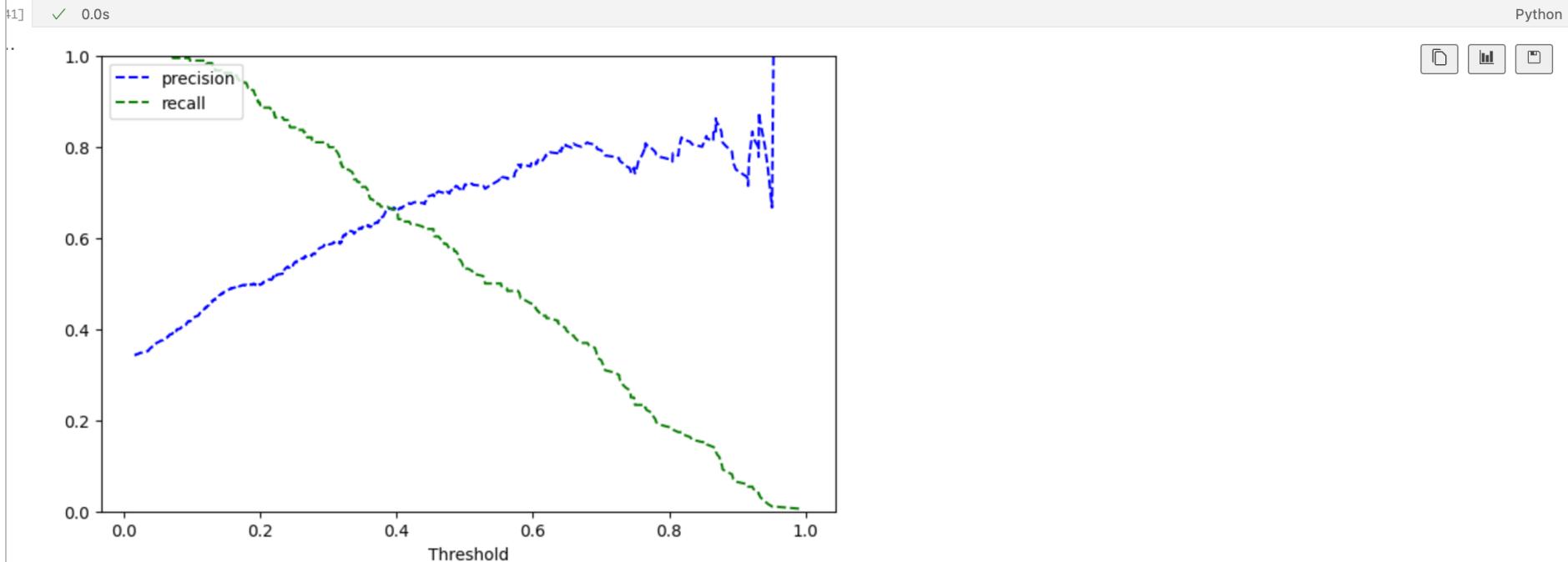


# Apprentissage supervisé : Classification

## Precision-Recall Curve for logistic regression

Precision-Recall curves summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.

```
# Predict_proba gives the probability of each observation belonging to each class
y_scores_lg=model_lgr.predict_proba(X_train_scaled)
precisions_lg, recalls_lg, thresholds_lg = metrics.precision_recall_curve(y_train, y_scores_lg[:,1])
# Plot values of precisions, recalls, and thresholds
plt.figure(figsize=(8,5))
plt.plot(thresholds_lg, precisions_lg[:-1], 'b--', label='precision')
plt.plot(thresholds_lg, recalls_lg[:-1], 'g--', label = 'recall')
plt.xlabel('Threshold')
plt.legend(loc='upper left')
plt.ylim([0,1])
plt.show()
```

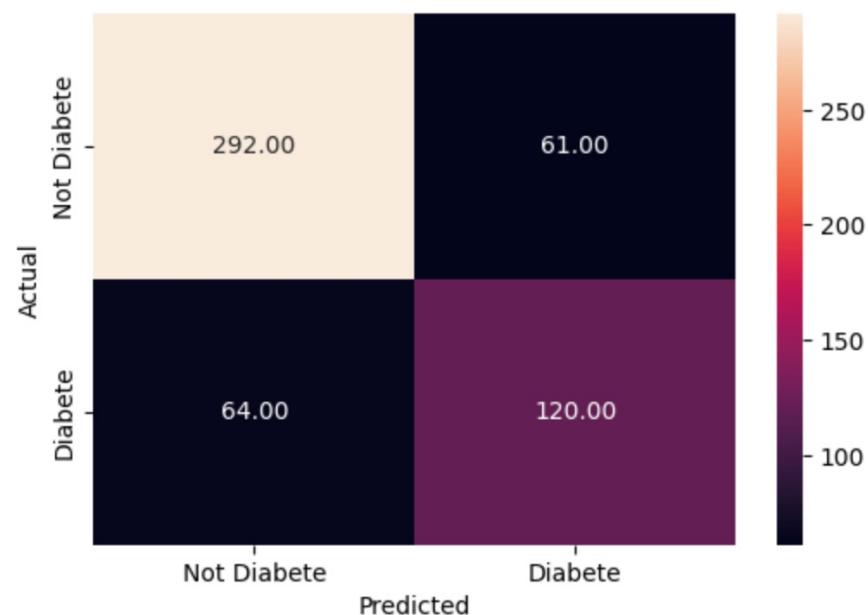


# Apprentissage supervisé : Classification

```
optimal_threshold=.4  
y_pred_train = model_lgr.predict_proba(X_train_scaled)  
metrics_score(y_train, y_pred_train[:,1]>optimal_threshold)
```

✓ 0.0s

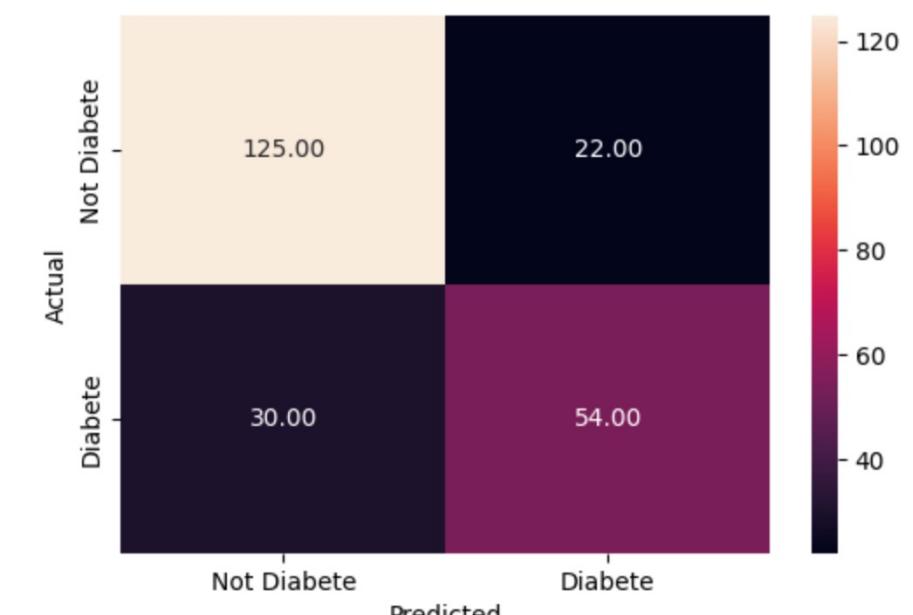
|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.82      | 0.83   | 0.82     | 353     |
| 1            | 0.66      | 0.65   | 0.66     | 184     |
| accuracy     |           |        | 0.77     | 537     |
| macro avg    | 0.74      | 0.74   | 0.74     | 537     |
| weighted avg | 0.77      | 0.77   | 0.77     | 537     |



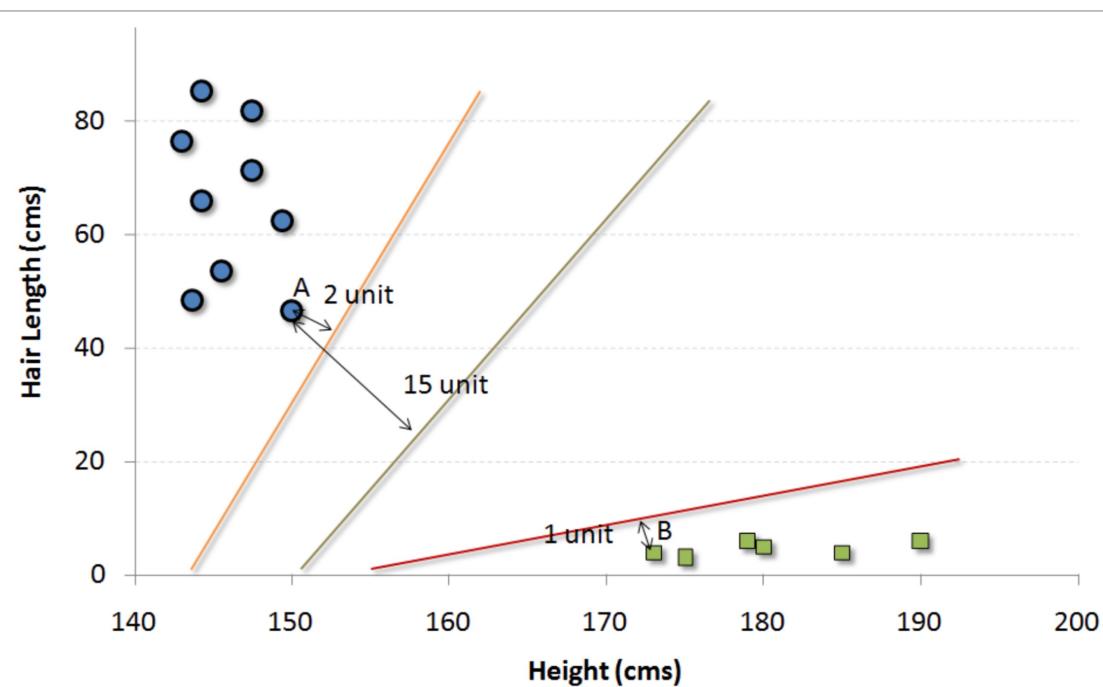
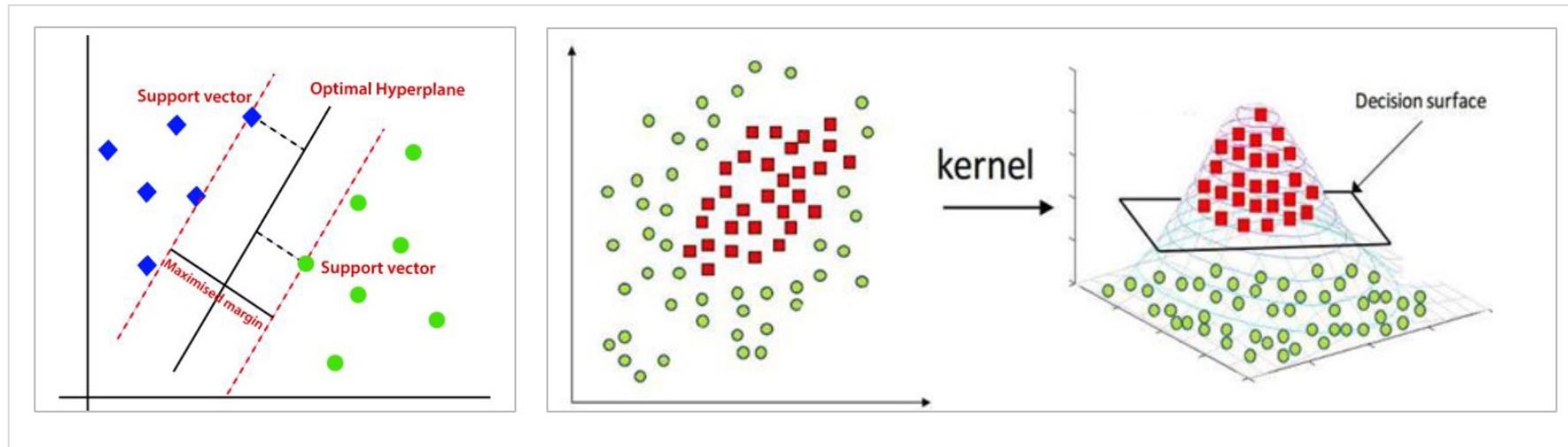
```
optimal_threshold=.4  
y_pred_test = model_lgr.predict_proba(X_test_scaled)  
metrics_score(y_test, y_pred_test[:,1]>optimal_threshold)
```

✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.81      | 0.85   | 0.83     | 147     |
| 1            | 0.71      | 0.64   | 0.68     | 84      |
| accuracy     |           |        | 0.77     | 231     |
| macro avg    | 0.76      | 0.75   | 0.75     | 231     |
| weighted avg | 0.77      | 0.77   | 0.77     | 231     |



# Apprentissage supervisé : Classification (Support Vector Machine)



- Les SVM sont des modèles de Machine Learning qui peuvent être utilisées à la fois pour les tâches de classification et de régression, mais Ils sont plus largement utilisés pour la classification.
- Un SVM vise à trouver un hyperplan dans un espace n-dimensionnel qui classe distinctement toutes les observations d'un ensemble de données
- Il est tout à fait possible d'avoir plusieurs hyperplans qui résolvent le problème. L'algorithme cherche à trouver celui avec la marge maximale.
- La dimension de l'hyperplan est déterminée par le nombre d'entrées
  - Pour 2 caractéristiques d'entrée : ligne
  - Pour 3 caractéristiques d'entrée : plan
- Toutes les données ne sont pas linéairement séparables. Ce qui rend la recherche de l'hyper plan complexe et couteuse en termes de calcul

# Apprentissage supervisé : Classification (Support Vector Machine)

## Exemple :

On dispose d'une population composée de 50% de femme et 50% d'hommes. En utilisant un échantillon de cette population, on veut créer un ensemble de règles qui nous guideront dans la classification de sexe pour le reste de la population.

On suppose que les deux facteurs de différenciation identifiés sont : la taille de l'individu et la longueur des cheveux. Voici un diagramme de dispersion de l'échantillon:

- Les cercles bleus dans le graphique représentent les femmes
- les carrés verts représentent les hommes.

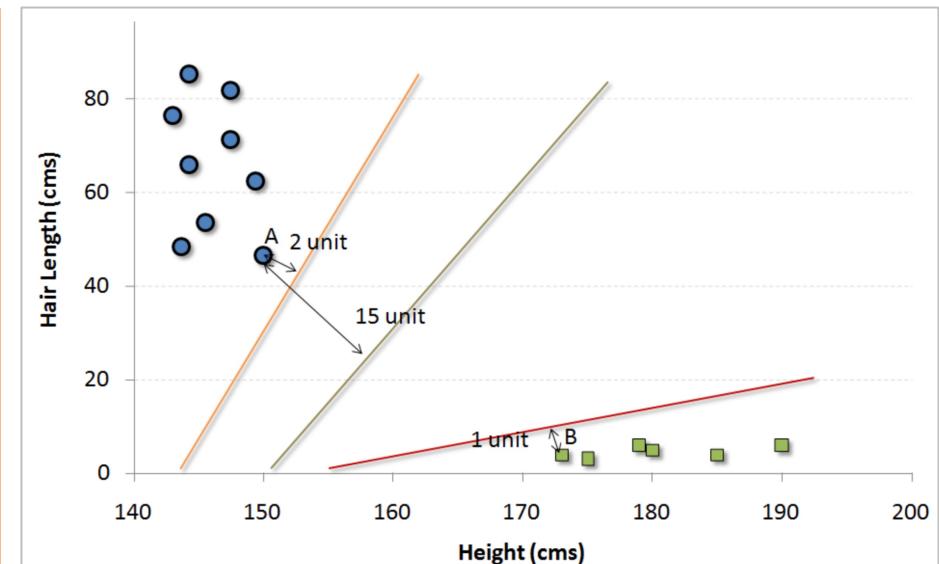
Visuellement, on peut distinguer ces deux classes comme suit :

- Les hommes de cette population ont une taille moyenne plus élevée.
- Les femmes de cette population ont des cheveux plus longs.

Supposons qu'on a un individu avec la hauteur 180 cm et la longueur des cheveux 4 cm qu'on devra classer. Intuitivement, on va le placer avec les hommes.

À l'aide d'un classificateur de la famille SVM cela est possible à travers un hyperplan séparateur bien choisi.

- On commence tout d'abord par projeter chaque élément de données dans l'espace n dimensionnel (où n représente le nombre de caractéristiques présentes dans le jeu de données) avec la valeur de chaque caractéristique étant la valeur d'une coordonnée particulière.
- Ensuite, on effectue la classification en trouvant l'hyperplan qui différencie très bien les deux classes. Dans le cas de la population ci-dessus, plusieurs choix se présentent.
- L'approche la plus adéquate dans ce cas est de trouver la distance minimale de la frontière par rapport à l'élément de la population la plus proche (cet élément peut appartenir à n'importe quelle classe).
- Par exemple, la frontière orange est la plus proche des cercles bleus. Et le cercle bleu le plus proche est à 2 unités de la frontière. Une fois que nous avons ces distances pour toutes les frontières, nous choisissons simplement la frontière avec la distance maximale (à partir du vecteur de support le plus proche).
- Sur les trois frontières indiquées, nous voyons que la frontière noire est la plus éloignée de l'élément le plus proche (c.-à-d. 15 unités).



# Apprentissage supervisé : Classification (Support Vector Machine)

## SVM

```
# Fitting SVM  
model2 = SVC(kernel='linear') # Linear kernal or linear decision boundary
```

45] ✓ 0.0s

```
model2.fit(X_train_scaled, y_train)
```

46] ✓ 0.0s

```
▼ SVC ⓘ ⓘ  
SVC(kernel='linear')
```

```
pred_train=model2.predict(X_train_scaled)  
pred_test = model2.predict(X_test_scaled)
```

47] ✓ 0.0s

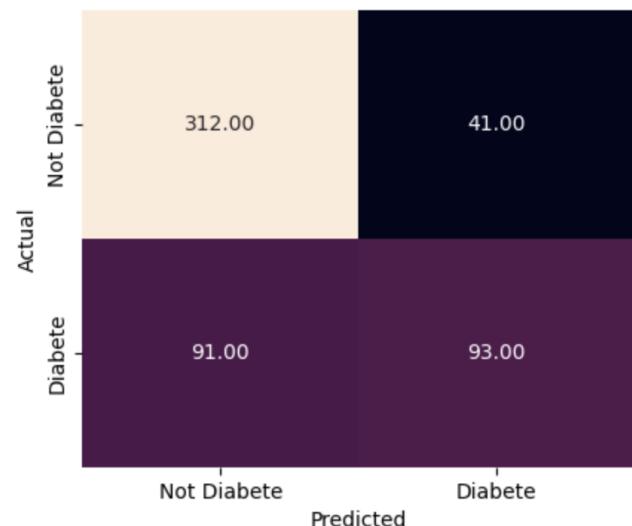
```
metrics_score(y_train, pred_train)
```

48] ✓ 0.0s

metrics\_score(y\_train, pred\_train)

✓ 0.0s

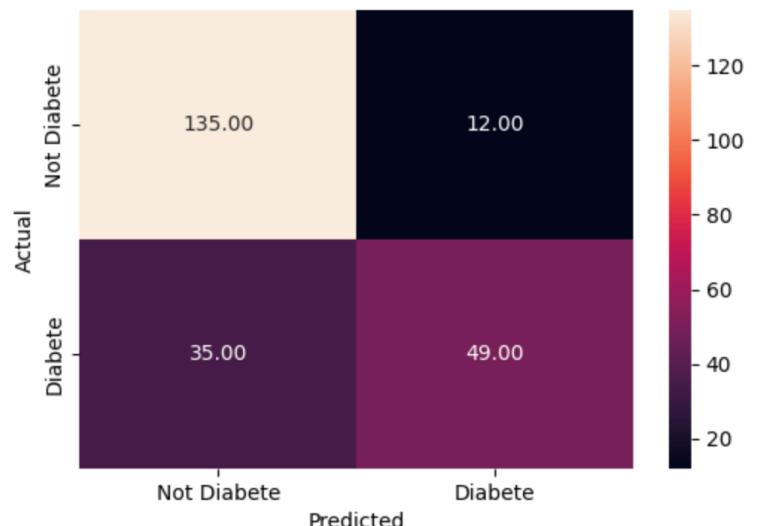
|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.77      | 0.88   | 0.83     | 353     |
| 1            | 0.69      | 0.51   | 0.58     | 184     |
| accuracy     |           |        | 0.75     | 537     |
| macro avg    | 0.73      | 0.69   | 0.71     | 537     |
| weighted avg | 0.75      | 0.75   | 0.74     | 537     |



metrics\_score(y\_test, pred\_test)

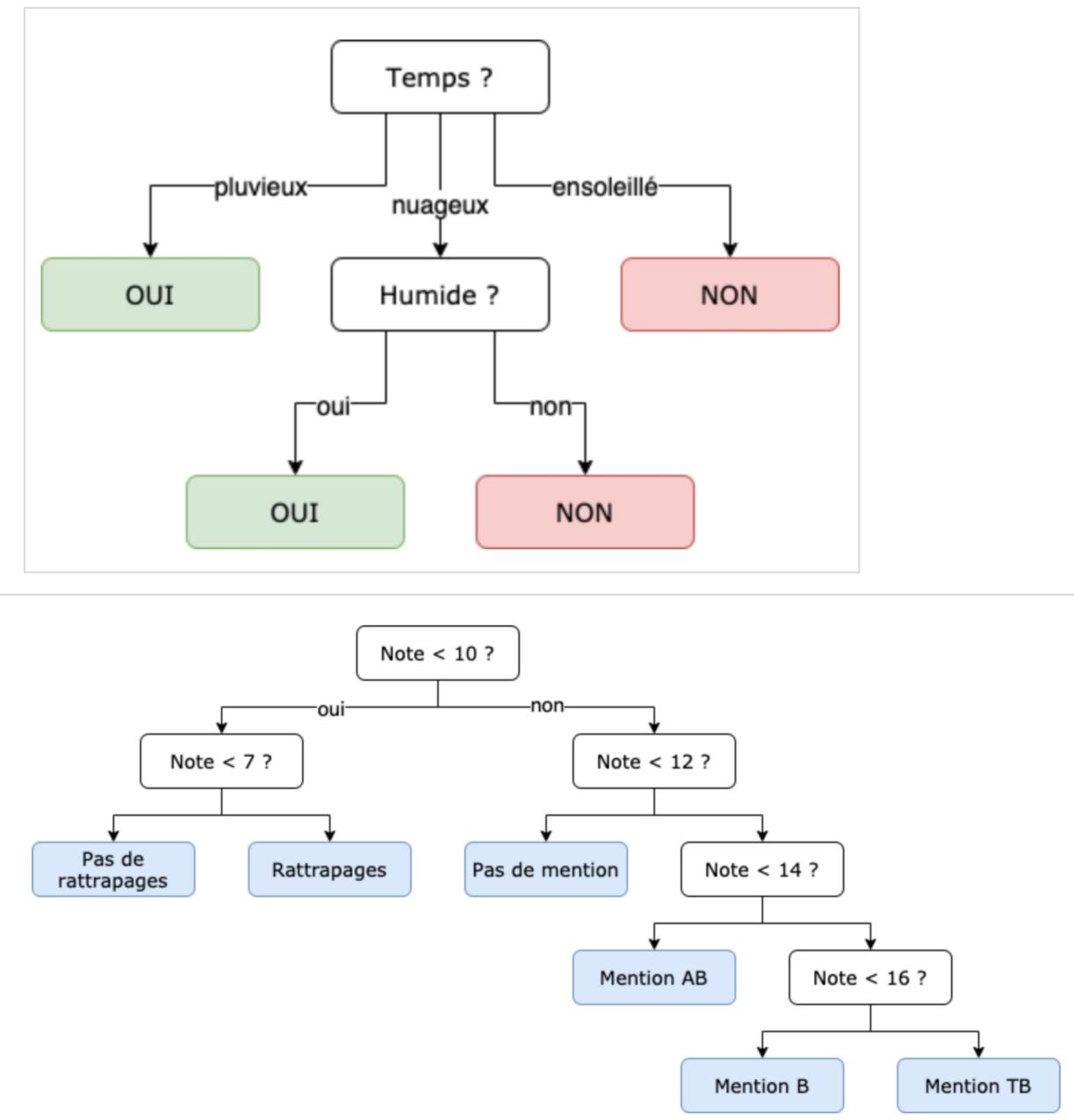
✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.79      | 0.92   | 0.85     | 147     |
| 1            | 0.80      | 0.58   | 0.68     | 84      |
| accuracy     |           |        | 0.80     | 231     |
| macro avg    | 0.80      | 0.75   | 0.76     | 231     |
| weighted avg | 0.80      | 0.80   | 0.79     | 231     |



# Apprentissage supervisé : Classification (Decision Tree)

- Un arbre de décisions est un algorithme d'apprentissage supervisé, utilisé à la fois pour les tâches de classification et de régression.
- Il possède une structure hiérarchique et arborescente, qui se compose d'un nœud racine, de branches, de nœuds internes et de nœuds feuille.
- Chaque nœud possède une condition qui amène à plusieurs réponses, ce qui dirige à un prochain nœud.
- Lorsqu'un nœud donne la réponse, on dit que le nœud est terminal.
- Exemple : Prenons l'arbre de décision suivant qui indique si l'on doit prendre un parapluie avec nous en fonction du temps.
- Dans cet exemple, un jour ensoleillé donnera directement la réponse NON, alors qu'un jour nuageux donnera la réponse OUI ou NON en fonction de l'humidité.
- Sur ce graphique, chaque nœud peut avoir aucune ou plusieurs possibilités: cela va dépendre de s'il est terminal ou non.
- Un arbre de décision binaire est un arbre où chaque nœud non terminal possède exactement deux possibilités (gauche et droite).
- Prenons l'arbre suivant qui indique la mention obtenue à un examen pour un étudiant, et s'il a le droit à des rattrapages en cas de note inférieure à 10.



# Apprentissage supervisé : Classification (Decision Tree)

## Decision Treee

```
# Building decision tree model  
model3 = DecisionTreeClassifier(class_weight = {0: 0.17, 1: 0.83}, random_state = 1)
```

✓ 0.0s

```
model3.fit(X_train_scaled, y_train)
```

✓ 0.0s

DecisionTreeClassifier

```
DecisionTreeClassifier(class_weight={0: 0.17, 1: 0.83},
```

Cellule de Markdown vide. Double-cliquez sur celle-ci, ou appuyez

```
pred_train=model3.predict(X_train_scaled)  
pred_test = model3.predict(X_test_scaled)
```

✓ 0.0s

metrics\_score(y\_train, pred\_train)

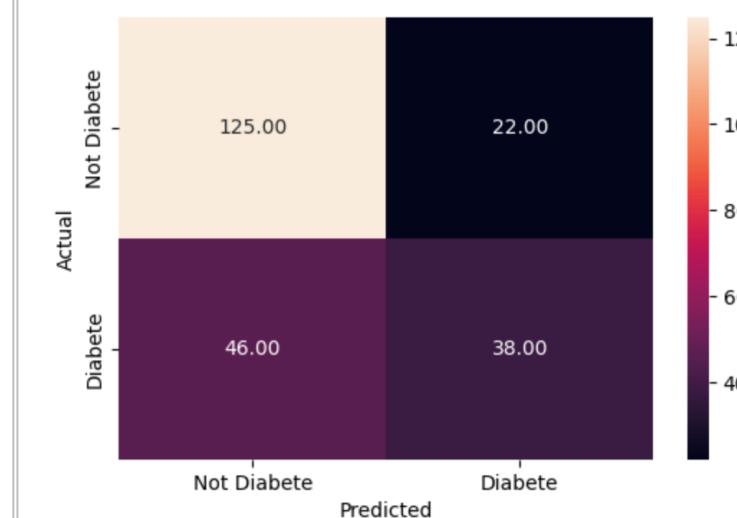
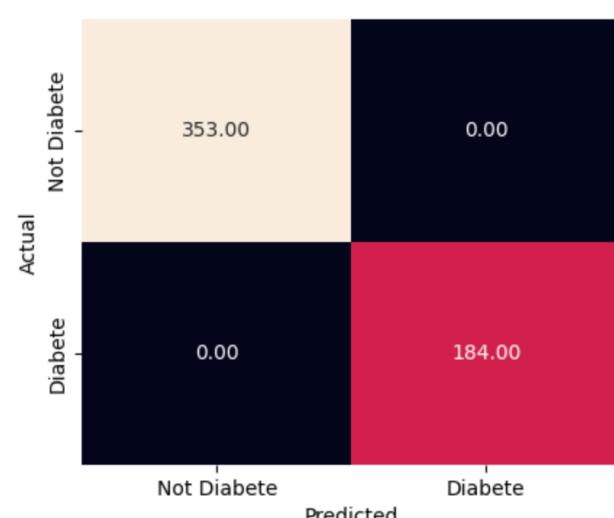
✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 1.00      | 1.00   | 1.00     | 353     |
| 1            | 1.00      | 1.00   | 1.00     | 184     |
| accuracy     |           |        | 1.00     | 537     |
| macro avg    | 1.00      | 1.00   | 1.00     | 537     |
| weighted avg | 1.00      | 1.00   | 1.00     | 537     |

metrics\_score(y\_test, pred\_test)

✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.73      | 0.85   | 0.79     | 147     |
| 1            | 0.63      | 0.45   | 0.53     | 84      |
| accuracy     |           |        | 0.71     | 231     |
| macro avg    | 0.68      | 0.65   | 0.66     | 231     |
| weighted avg | 0.70      | 0.71   | 0.69     | 231     |



# Apprentissage supervisé : Classification (Decision Tree)

```
# Plot the feature importance
```

```
importances = model3.feature_importances_
columns = X.columns
importance_df = pd.DataFrame(importances, index = columns, columns = ['Importance']).sort_values(by = 'Importance', ascending = False)
```

✓ 0.0s

```
importance_df
```

✓ 0.0s

|                          | Importance |
|--------------------------|------------|
| Glucose                  | 0.366708   |
| BMI                      | 0.187544   |
| DiabetesPedigreeFunction | 0.131999   |
| Age                      | 0.085482   |
| Insulin                  | 0.073709   |
| SkinThickness            | 0.060673   |
| BloodPressure            | 0.058156   |
| Pregnancies              | 0.035728   |

```
importance_df=importance_df.reset_index()
importance_df
```

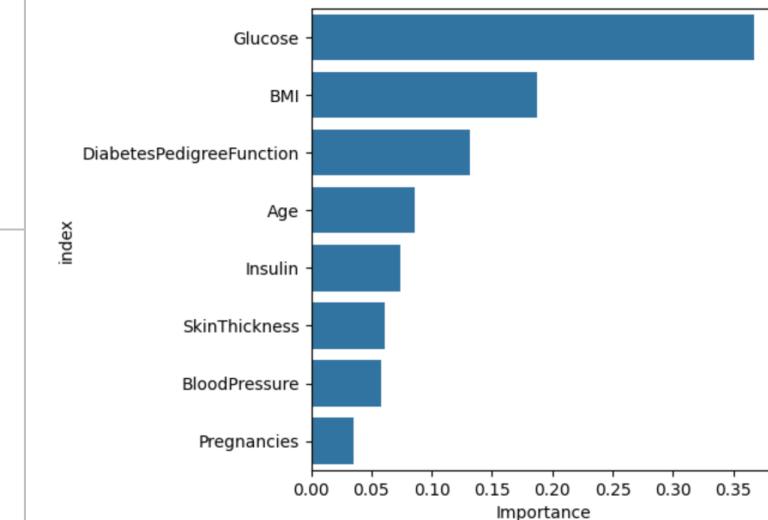
✓ 0.0s

| level_0 | index | Importance                        |
|---------|-------|-----------------------------------|
| 0       | 0     | Glucose 0.366708                  |
| 1       | 1     | BMI 0.187544                      |
| 2       | 2     | DiabetesPedigreeFunction 0.131999 |
| 3       | 3     | Age 0.085482                      |
| 4       | 4     | Insulin 0.073709                  |
| 5       | 5     | SkinThickness 0.060673            |
| 6       | 6     | BloodPressure 0.058156            |
| 7       | 7     | Pregnancies 0.035728              |

```
plt.figure(figsize = (5, 5))
sns.barplot(data=importance_df, y='index',x='Importance', orient='h')
```

✓ 0.0s

<Axes: xlabel='Importance', ylabel='index'>

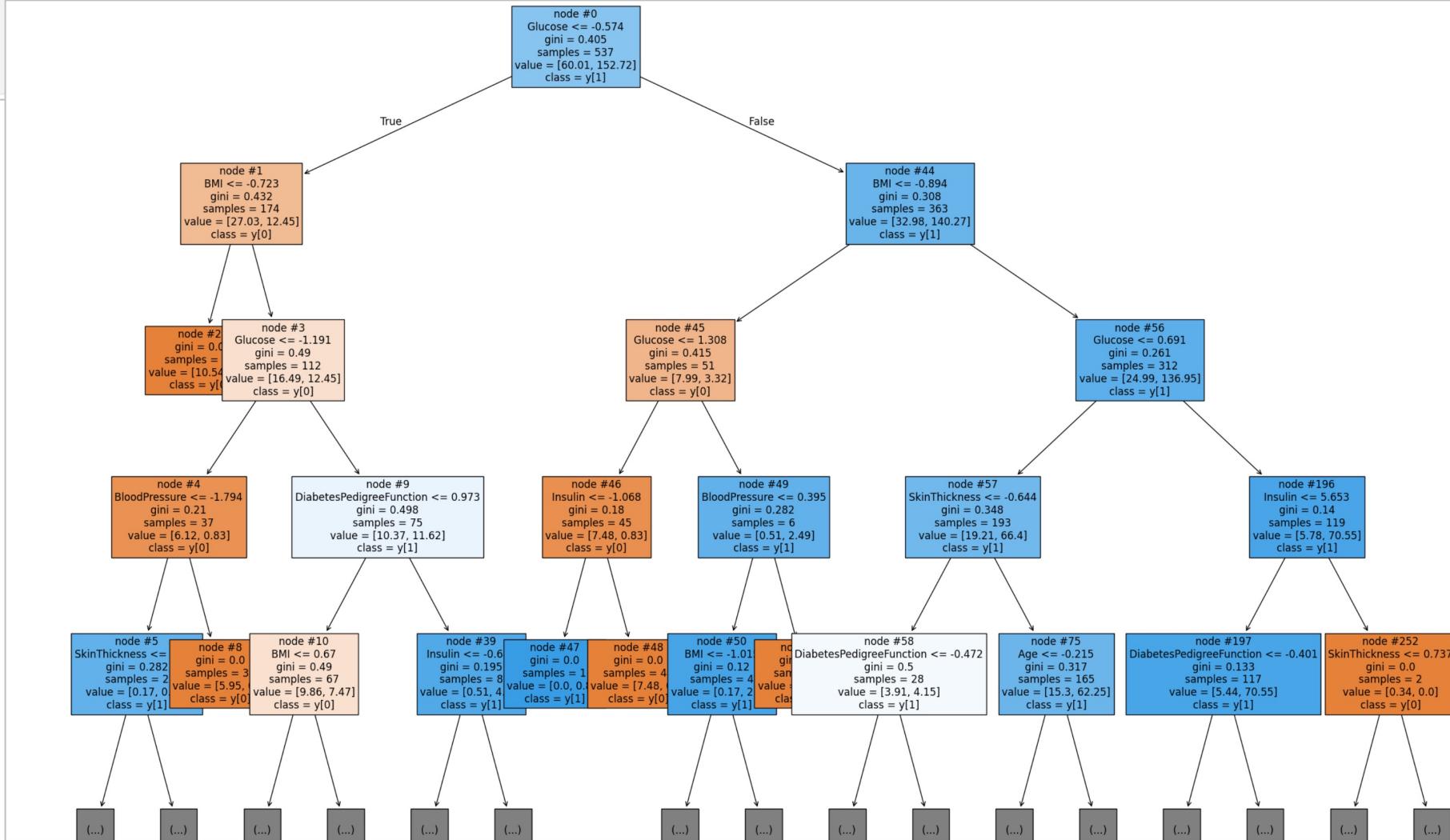


# Apprentissage supervisé : Classification (Decision Tree)

```
from sklearn import tree
features = list(X.columns)
plt.figure(figsize = (30, 20))
tree.plot_tree(model3, max_depth = 4, feature_names = features, filled = True, fontsize = 12, node_ids = True, class_names = True)
```

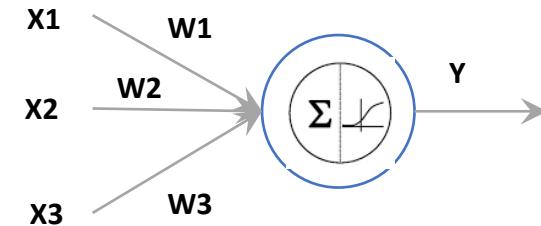
```
plt.show()
```

✓ 0.3s

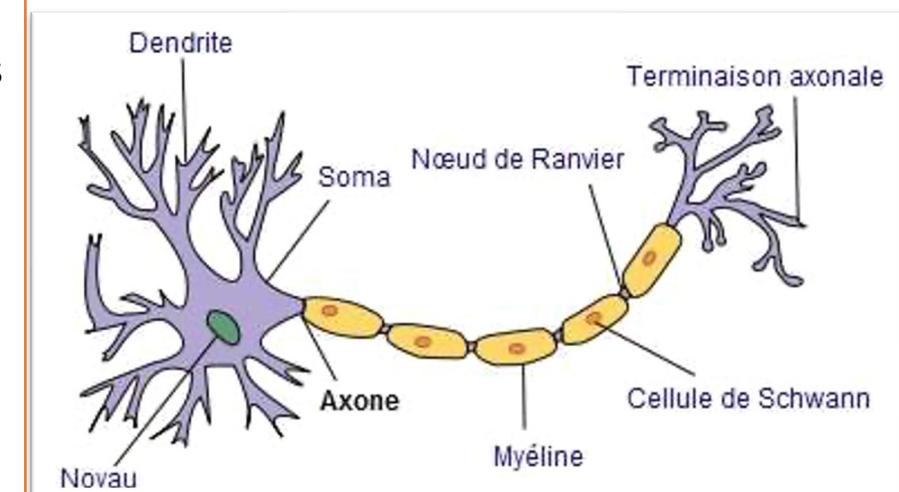


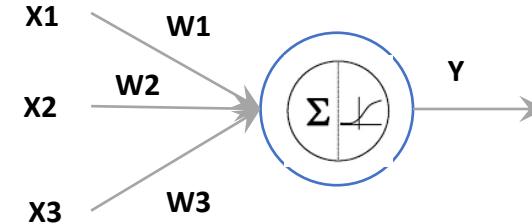
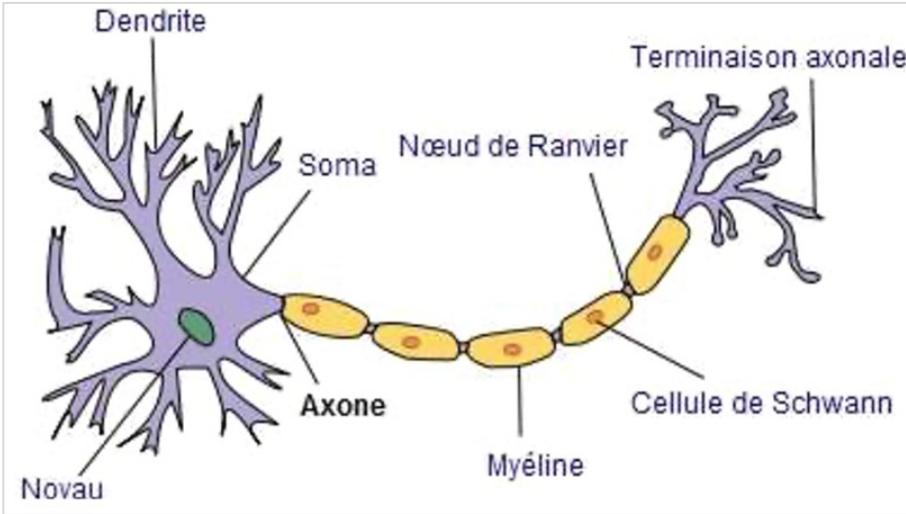
# Apprentissage supervisé : Classification (Neural Network)

- Un neurone artificiel (NA) est une unité de calcul élémentaire (fonction mathématique ) permet de mettre en relations des entrées  $X_i$  avec une sortie  $Y$
- Un neurone artificiel est représentation approximative d'un neurone biologique
  - Additionne ses entrées  $x_i$  pondérées par des poids  $w_i$ ,
  - compare la somme résultante à une valeur seuil selon un fonction d'activation,
  - répond en émettant un signal si cette somme est supérieure ou égale à ce seuil (modèle ultra-simplifié du fonctionnement d'un neurone biologique).
  - Ces neurones sont par ailleurs associés en réseaux dont la topologie des connexions est variable : réseaux proactifs, récurrents, etc.
  - Enfin, l'efficacité de la transmission des signaux d'un neurone à l'autre peut varier : on parle de « poids synaptique », et ces poids peuvent être modulés par des règles d'apprentissage (ce qui mime la plasticité synaptique des réseaux biologiques).
- Un neurone reçoit des signaux venant de d'autres neurones à travers ses dendrites(connexions)
- Emet un signal ou non à travers son Axone selon les entrées reçues et les poids synaptiques.

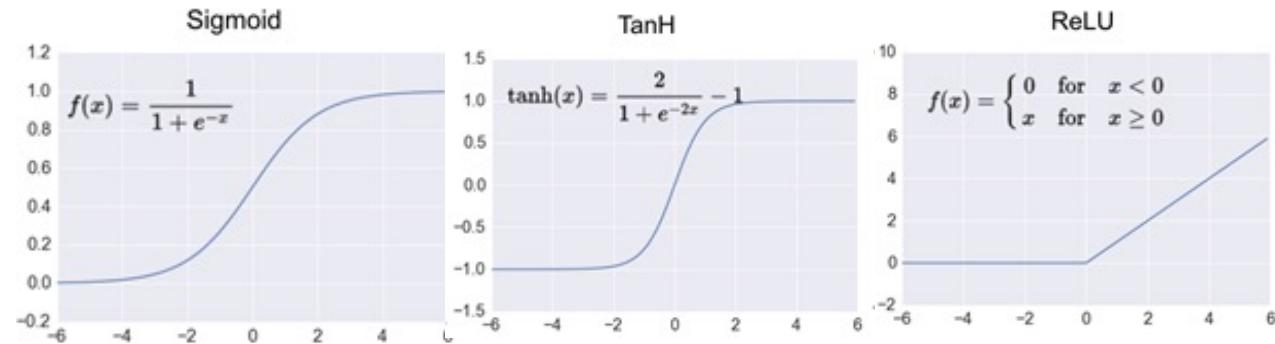
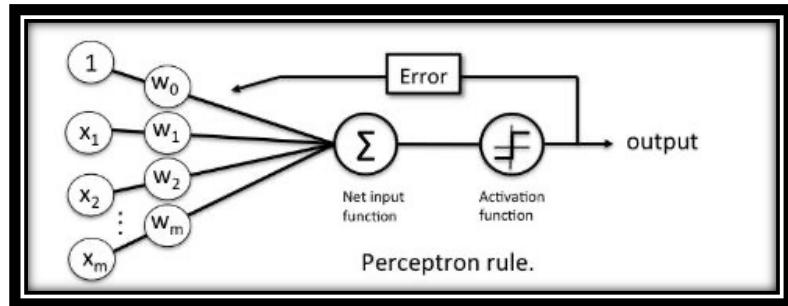


- $\text{Sum} = x_1 w_1 + x_2 w_2 + x_3 w_3$
- Si  $\text{Sum} > \text{Seuil} \Rightarrow Y=1$  Si non  $Y=0$





- $\text{Sum} = x_1w_1 + x_2w_2 + x_3w_3$
- Si  $\text{Sum} > \text{Seuil} \Rightarrow Y=1$  Si non  $Y=0$



# Apprentissage supervisé : Classification (Neural Network)

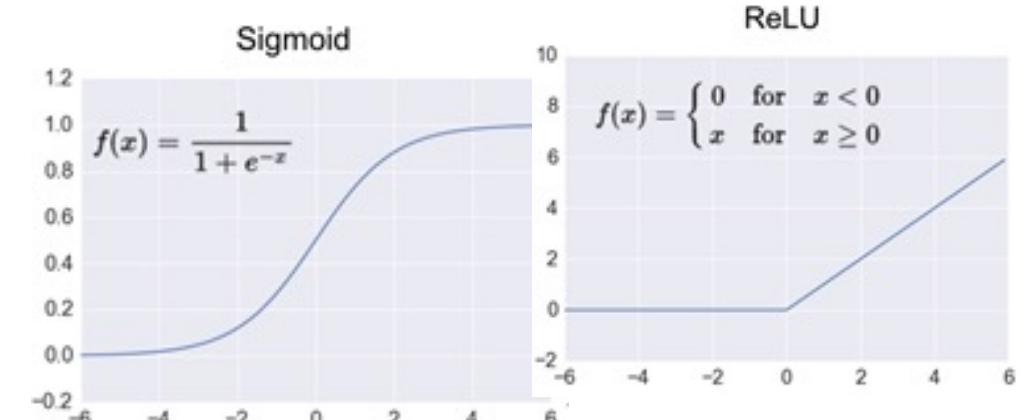
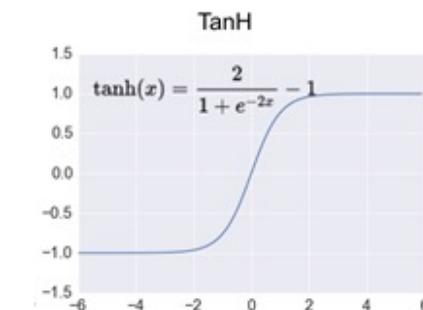
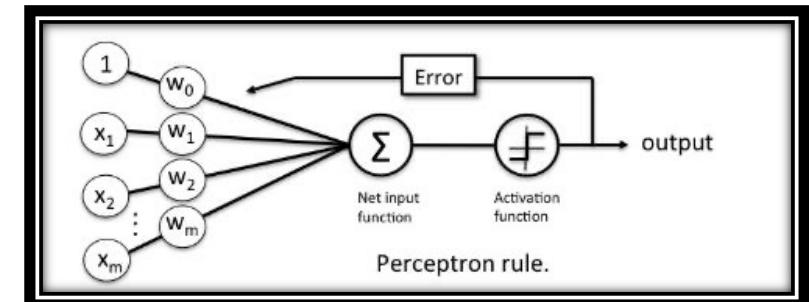
## Perceptron [Frank Rosenblatt ,1957]

- Modèle d'apprentissage supervisé, classifieurs binaires (séparant deux classes).
- Le perceptron reçoit un ensemble d'entrées pondérées par des poids synaptiques  $w_i$  et produit une sortie binaire en sortie.
- La sortie est produite en :
  - Additionnant les entrées  $x_i$  pondérées par les poids synaptiques du perceptron  $w_i$ 
    - $S = \sum_{i=1}^m x_i w_i$
  - Ensuite on applique à cette somme une **fonction d'activation non linéaire** qui produit une sortie binaire. : Exemple Seuillage

$$\bullet \quad Y = \begin{cases} 1 & \text{si } S \geq \theta \\ -1 & \text{si } S < \theta \end{cases}$$

Dans la phase d'apprentissage,

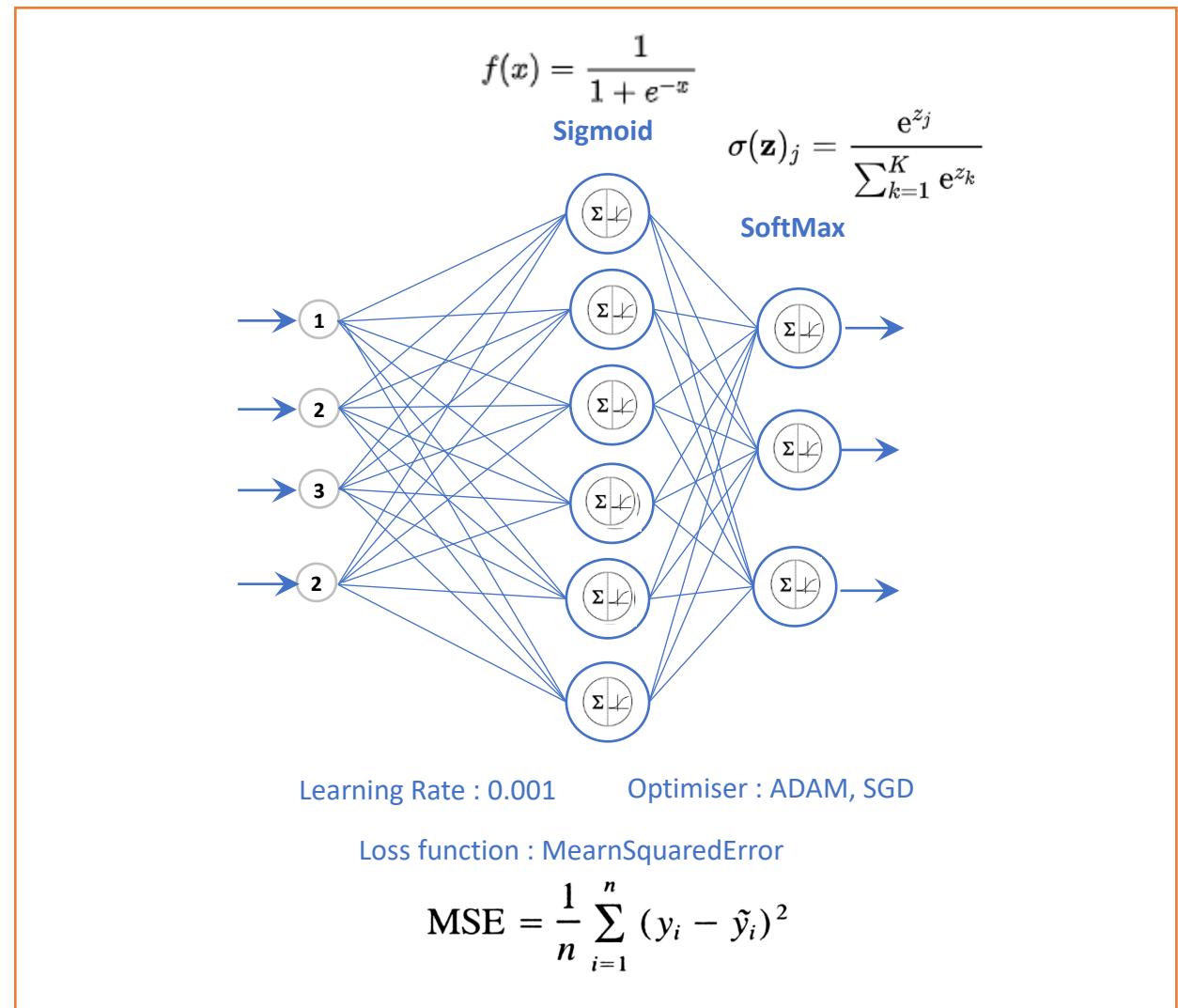
- $\text{err} = Y_p - Y_t$
- mise à jour des poids :  $w_i(t) = w_i(t-1) + \alpha \text{err}.x_i$
- $\alpha$  : vitesse d'apprentissage (entre 0 et 1)



# Apprentissage supervisé : Classification (Neural Network)

## Multi Layer Perceptron (MLP)

- Un Multi Layer Perceptron (MLP) est un type de réseau de neurones artificiels composé de plusieurs couches de neurones. Voici ses principales caractéristiques :
- **Couches** : Il comprend une couche d'entrée, une ou plusieurs couches cachées, et une couche de sortie.
- **Neurones** : Les neurones dans chaque couche sont connectés à tous les neurones de la couche précédente (connexion "fully connected").
- **Fonction d'activation** : Chaque neurone applique une fonction d'activation (comme ReLU, sigmoïde, ou tanh) pour introduire de la non-linéarité.
- **Entraînement par rétropropagation** : L'apprentissage du MLP se fait par rétropropagation de l'erreur à l'aide d'un algorithme d'optimisation, comme l'Adam ou le SGD.
- Un MLP est souvent utilisé pour des tâches de classification ou de régression et est une forme de réseau de neurones feedforward.



# Apprentissage supervisé : Classification (Neural Network)

## Neural Network : Multi Layer Perceptron

```
from sklearn.neural_network import MLPClassifier  
model4 = MLPClassifier(solver='lbfgs', alpha=1e-5,  
hidden_layer_sizes=(10,5,3), random_state=1, max_iter=2000)  
model4.fit(X= X_train_scaled, y = y_train)
```

✓ 0.8s

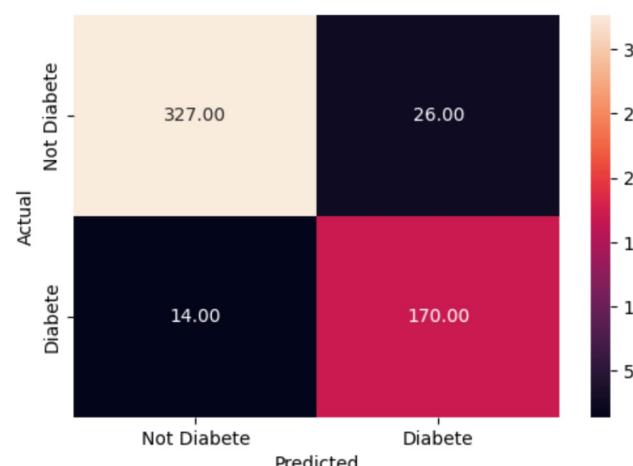
MLPClassifier

```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(10, 5, 3), max_iter=2000,  
random_state=1, solver='lbfgs')
```

```
pred_train=model4.predict(X_train_scaled)  
pred_test = model4.predict(X_test_scaled)
```

✓ 0.0s

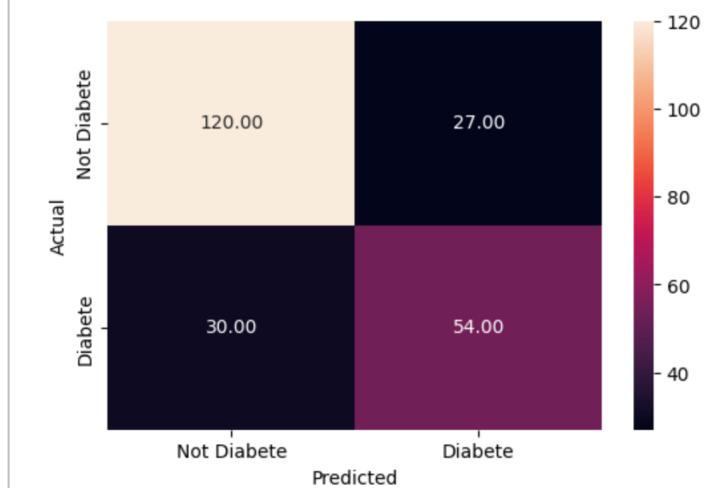
| metrics_score(y_train, pred_train) |           |        |          |         |
|------------------------------------|-----------|--------|----------|---------|
|                                    | precision | recall | f1-score | support |
| 0                                  | 0.96      | 0.93   | 0.94     | 353     |
| 1                                  | 0.87      | 0.92   | 0.89     | 184     |
| accuracy                           |           |        | 0.93     | 537     |
| macro avg                          | 0.91      | 0.93   | 0.92     | 537     |
| weighted avg                       | 0.93      | 0.93   | 0.93     | 537     |



```
metrics_score(y_test, pred_test)
```

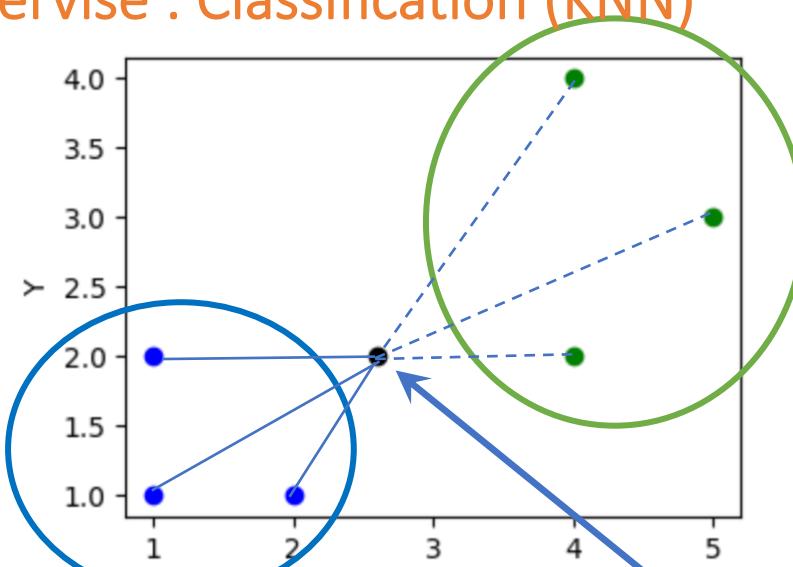
✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.80      | 0.82   | 0.81     | 147     |
| 1            | 0.67      | 0.64   | 0.65     | 84      |
| accuracy     |           |        | 0.75     | 231     |
| macro avg    | 0.73      | 0.73   | 0.73     | 231     |
| weighted avg | 0.75      | 0.75   | 0.75     | 231     |



# Apprentissage supervisé : Classification (KNN)

|   | x   | y   | Label |
|---|-----|-----|-------|
| 0 | 1.0 | 1.0 | A     |
| 1 | 2.0 | 1.0 | A     |
| 2 | 1.0 | 2.0 | A     |
| 3 | 4.0 | 2.0 | B     |
| 4 | 4.0 | 4.0 | B     |
| 5 | 5.0 | 3.0 | B     |



```
import numpy as np  
d['distance'] = np.sqrt((y-d['y'])**2+(x-d['x'])**2)  
✓ 0.0s
```

```
d  
✓ 0.0s
```

|   | x   | y   | Label | distance |
|---|-----|-----|-------|----------|
| 0 | 1.0 | 1.0 | A     | 1.886796 |
| 1 | 2.0 | 1.0 | A     | 1.166190 |
| 2 | 1.0 | 2.0 | A     | 1.600000 |
| 3 | 4.0 | 2.0 | B     | 1.400000 |
| 4 | 4.0 | 4.0 | B     | 2.441311 |
| 5 | 5.0 | 3.0 | B     | 2.600000 |

```
d.sort_values('distance')  
✓ 0.0s
```

|   | x   | y   | Label | distance |
|---|-----|-----|-------|----------|
| 1 | 2.0 | 1.0 | A     | 1.166190 |
| 3 | 4.0 | 2.0 | B     | 1.400000 |
| 2 | 1.0 | 2.0 | A     | 1.600000 |
| 0 | 1.0 | 1.0 | A     | 1.886796 |
| 4 | 4.0 | 4.0 | B     | 2.441311 |
| 5 | 5.0 | 3.0 | B     | 2.600000 |

- L'algorithme des K plus proches voisins ou K-nearest neighbors (kNN) est un algorithme de Machine Learning qui appartient à la classe des algorithmes d'apprentissage supervisé simple et facile à mettre en œuvre qui peut être utilisé pour résoudre les problèmes de classification et de régression

## Algorithme :

- On choisit une valeur pour k
- On calcule les distances d entre la nouvelle donnée N et ses voisins Xi déjà classifiés.
- Parmi les points Xi, les k plus proches de N sont retenus.
- On attribue à N le label majoritaire parmi les k plus proches voisins

x=2.6  
y=2

→ Classe A

$$\text{distance euclidienne}(xi) = \sqrt{(x - xi)^2 + (y - yi)^2}$$

$$\text{distance manhattan}(xi) = |x - xi| + |y - yi|$$

K=3

- Avantage : Phase d'entraînement très rapide (Pas d'apprentissage)
- Inconvénient : Phase de Prédiction très couteuse si le dataset d'entraînement est très grand

# Apprentissage supervisé : Classification (KNN)

```
from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n_neighbors=3)  
knn.fit(X=X_train_scaled, y=y_train)
```

✓ 0.0s

## KNeighborsClassifier

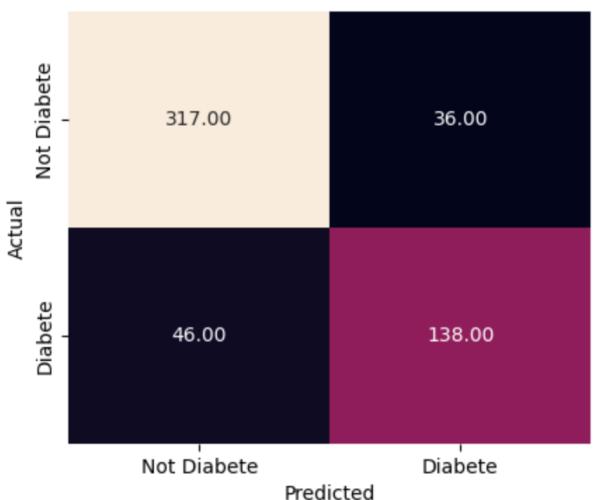
KNeighborsClassifier(n\_neighbors=3)

```
pred_train=knn.predict(X_train_scaled)  
pred_test = knn.predict(X_test_scaled)
```

## metrics\_score(y\_train, pred\_train)

✓ 0.0s

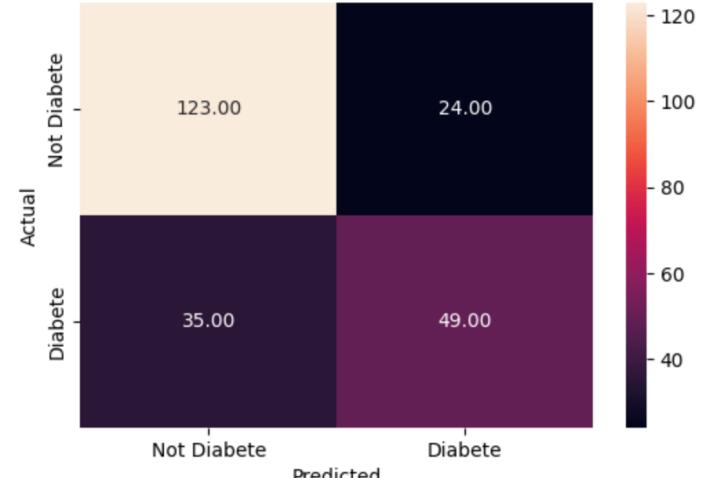
|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.87      | 0.90   | 0.89     | 353     |
| 1            | 0.79      | 0.75   | 0.77     | 184     |
| accuracy     |           |        | 0.85     | 537     |
| macro avg    | 0.83      | 0.82   | 0.83     | 537     |
| weighted avg | 0.85      | 0.85   | 0.85     | 537     |



## metrics\_score(y\_test, pred\_test)

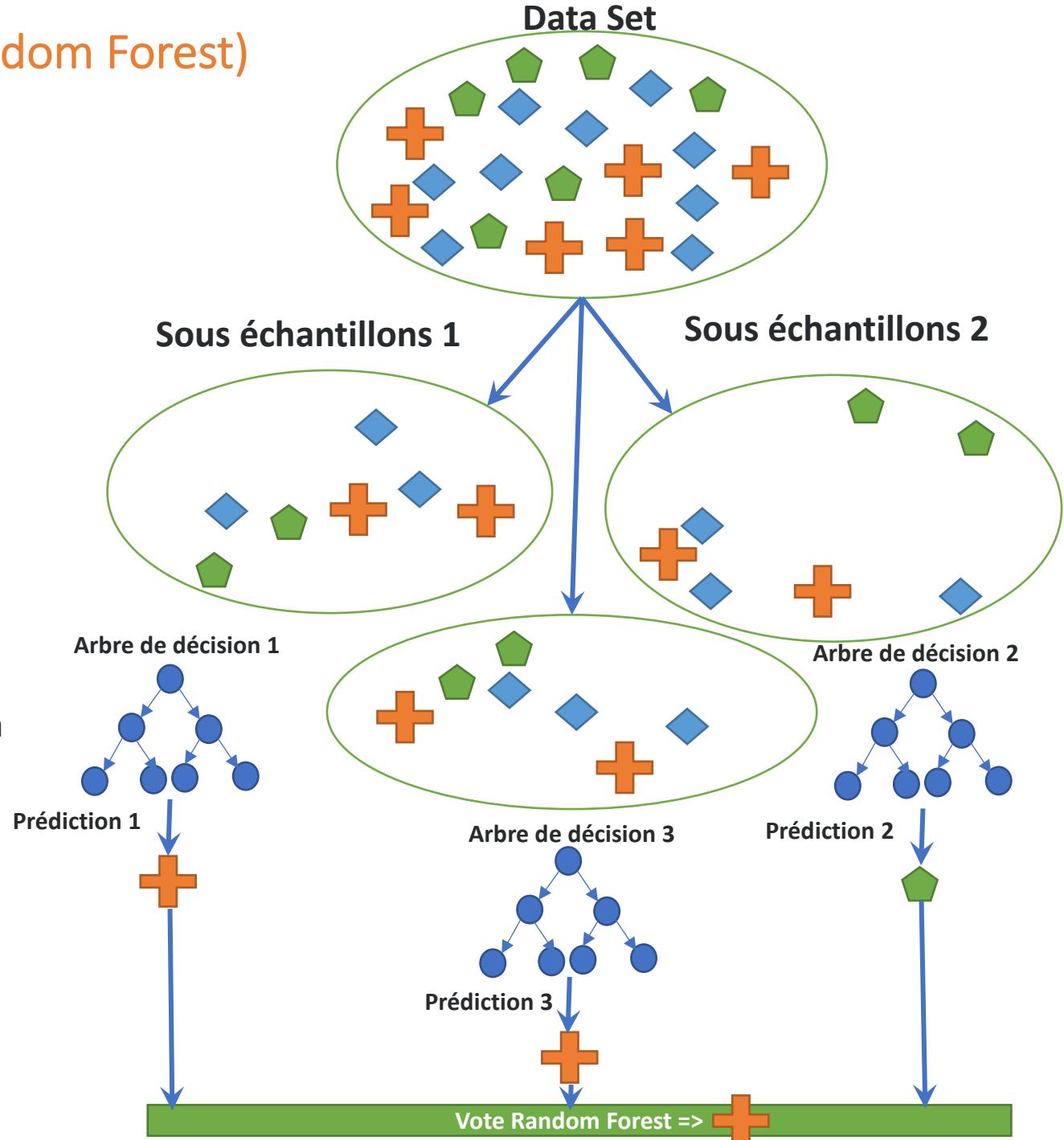
✓ 0.0s

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.78      | 0.84   | 0.81     | 147     |
| 1            | 0.67      | 0.58   | 0.62     | 84      |
| accuracy     |           |        | 0.74     | 231     |
| macro avg    | 0.72      | 0.71   | 0.72     | 231     |
| weighted avg | 0.74      | 0.74   | 0.74     | 231     |



## Apprentissage supervisé : Classification (Random Forest)

- Le Random Forest (qui signifie *forêt aléatoire*) est un **ensemble d'arbres de décision** utilisés pour prédire une quantité (Régression) ou une probabilité (classification)
- Principe de base de l'algorithme :
  - La première étape consiste à appliquer le principe du **bagging**, c'est-à-dire créer de nombreux sous-échantillons aléatoires de notre ensemble de données avec possibilité de sélectionner la même valeur plusieurs fois.
  - Des arbres de décision **individuels** sont ensuite construits **pour chaque échantillon**. Chaque arbre est entraîné sur une **portion aléatoire** afin de recréer une prédiction.
  - Enfin, chaque arbre va **prédir un résultat** (target). Le résultat avec **le plus de votes** (le plus fréquent) devient le résultat final de notre modèle. Dans le cas de régression, on prendra la moyenne des votes de tous les arbres.



# Apprentissage supervisé : Random Forest

## Random Forest

```
from sklearn.ensemble import RandomForestClassifier  
rf = RandomForestClassifier( random_state = 1)  
  
rf.fit(X_train, y_train)
```

79] ✓ 0.0s

```
    ▾ RandomForestClassifier ⓘ ?  
    RandomForestClassifier(random_state=1)
```

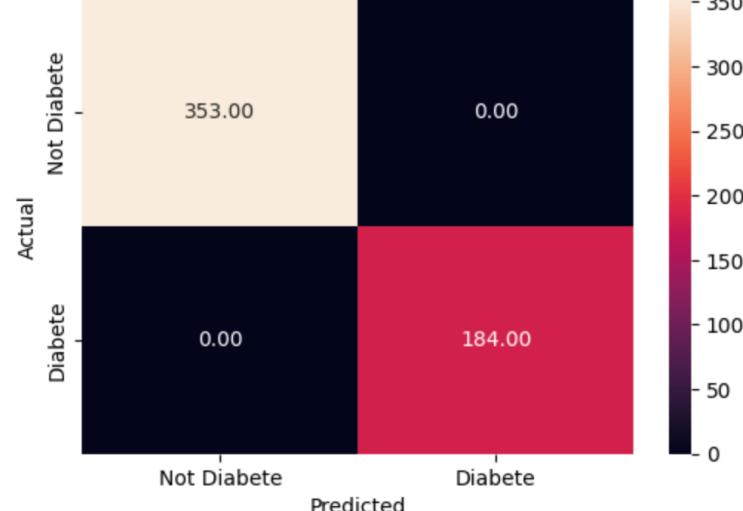
```
pred_train=rf.predict(X_train)  
pred_test = rf.predict(X_test)|
```

80] ✓ 0.0s

metrics\_score(y\_train, pred\_train)

✓ 0.1s

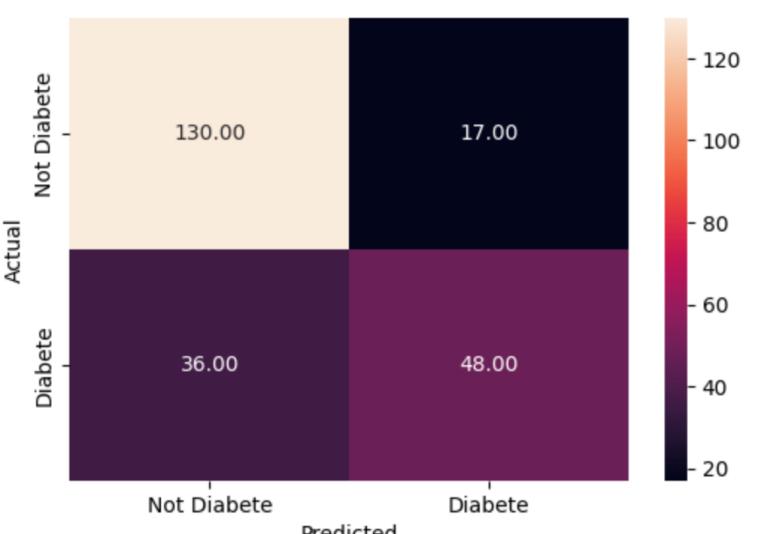
|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 1.00      | 1.00   | 1.00     | 353     |
| 1            | 1.00      | 1.00   | 1.00     | 184     |
| accuracy     |           |        | 1.00     | 537     |
| macro avg    | 1.00      | 1.00   | 1.00     | 537     |
| weighted avg | 1.00      | 1.00   | 1.00     | 537     |



metrics\_score(y\_test, pred\_test)

✓ 0.0s

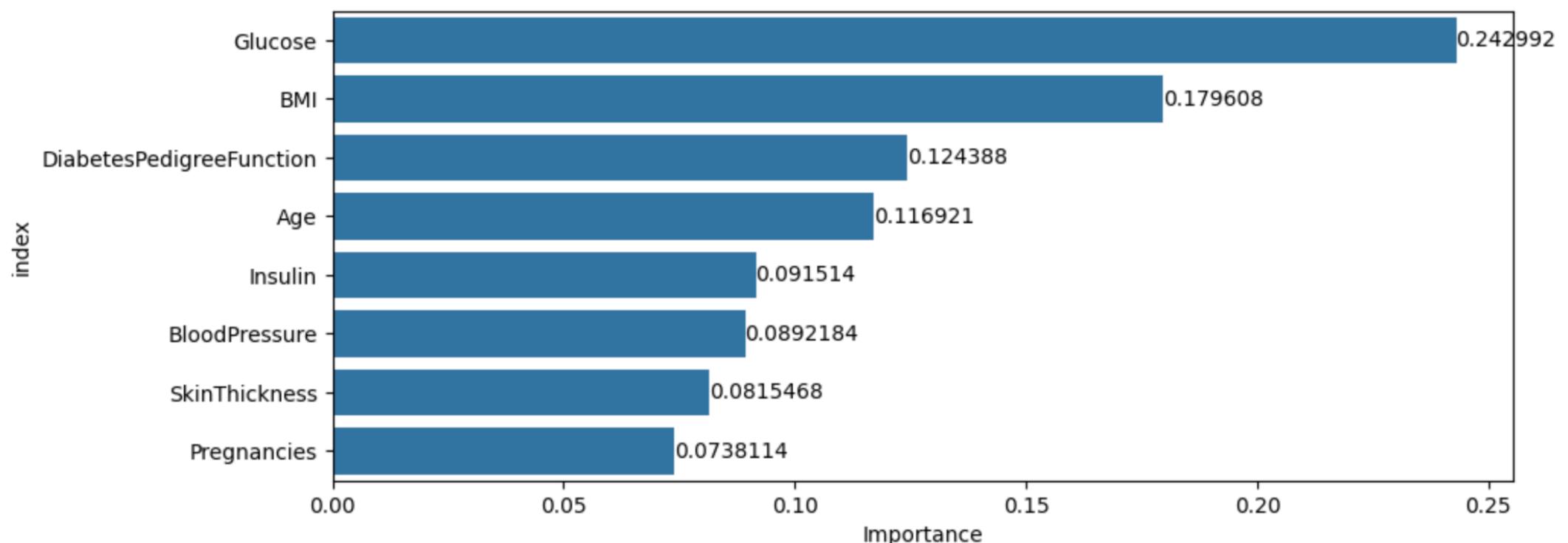
|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.78      | 0.88   | 0.83     | 147     |
| 1            | 0.74      | 0.57   | 0.64     | 84      |
| accuracy     |           |        | 0.77     | 231     |
| macro avg    | 0.76      | 0.73   | 0.74     | 231     |
| weighted avg | 0.77      | 0.77   | 0.76     | 231     |



# Apprentissage supervisé : Random Forest

```
importances = rf.feature_importances_
columns = X.columns
importance_df = pd.DataFrame(
    importances, index = columns,
    columns = ['Importance']).sort_values(by = 'Importance', ascending = False)
importance_df=importance_df.reset_index()
plt.figure(figsize = (10, 4))
bar =sns.barplot(data=importance_df,x='Importance', y='index', orient='h')
bar = bar.bar_label(bar.containers[0], fontsize=10)
```

✓ 0.0s

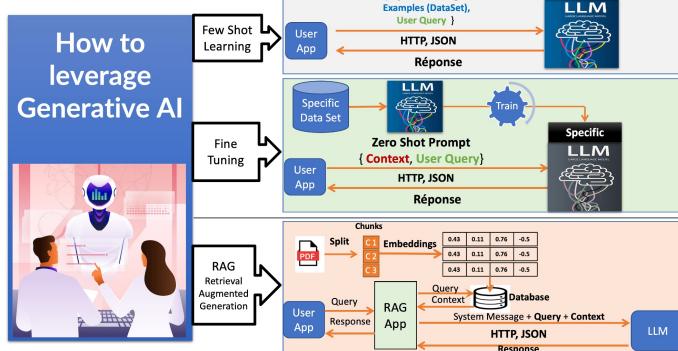


# Intelligence Artificielle & IA Générative

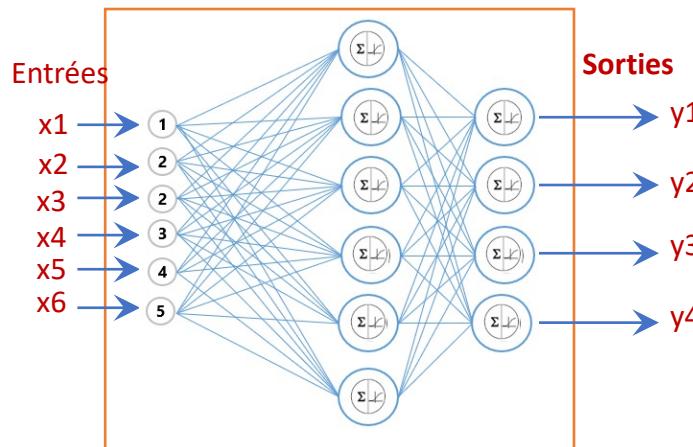


Mohamed Youssfi, Enseignant Chercheur, ENSET Mohammedia, Directeur Laboratoire Informatique, Intelligence Artificielle et Cyber Sécurité

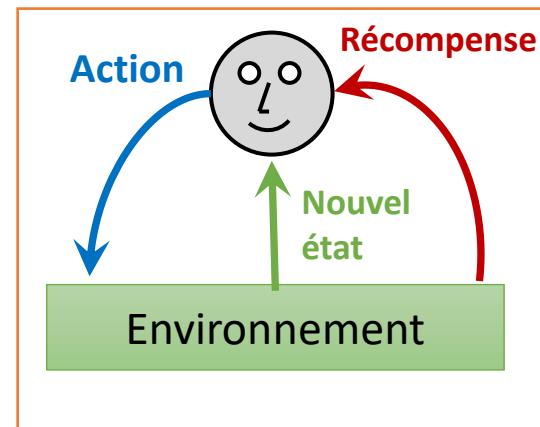
## Leverage Generative AI



## Apprentissage Supervisé



## Apprentissage Par Renforcement



## Artificial Intelligence Machine Learning

- Data-driven learning**
- Supervised Learning
  - Unsupervised Learning
  - Self Supervised Learning
- Experiential learning**
- Reinforcement Learning

## Deep Learning

Neural Network  
Computer Vision, NLP  
CNN, RNN, LSTM, GRU

Generative AI  
Transformers

Large Language Models



# Intelligence Artificielle

- L'intelligence artificielle
- Intelligence Artificielle Distribuée =
  - IA : Pour des agents intelligent (Modéliser le savoir et le comportement)
  - + Distribuée : Modéliser leurs interactions => **Intelligence Collective**
- IA = IA symbolique (5%) + Machine Learning (95%)
- Techniques d'apprentissage :
  - Piloté par les données
  - Apprentissage Supervisé
    - Régression
    - Classification
  - Apprentissage Non Supervisé
  - Apprentissage Auto Supervisé
  - Piloté par l'expérience
- Deep Learning
- IA Générative

## Apprentissage Non Supervisé

### Clustering

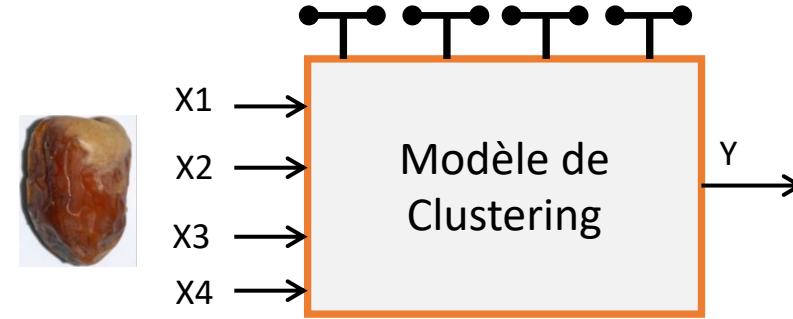
Data Set (Données Non étiquetées)



Algorithmes : K-Means

# Apprentissage non supervisé

Data Set (Données Non étiquetées)



Algorithmes : K-Means

# Apprentissage Non Supervisé

- **Apprentissage Non Supervisé :**

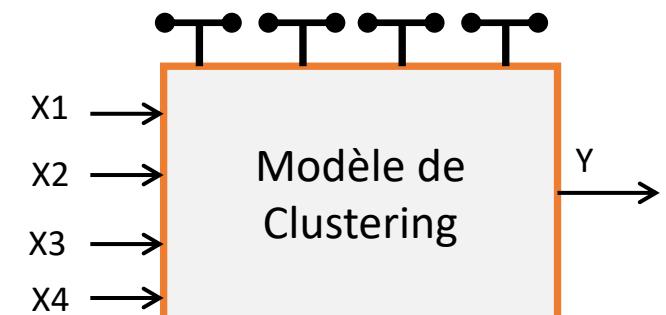
- Consiste à utiliser des données non étiquetées et utiliser des algorithmes de clustering qui vont fouiller dans les données pour chercher à les segmenter en un ensemble de groupes homogènes en se basant sur des mesures de similarités.
- Exemple d'algorithme de clustering :

- **KMeans**

# Apprentissage Non Supervisé

## Clustering

### Data Set (Données Non étiquetées)



Algorithmes : K-Means

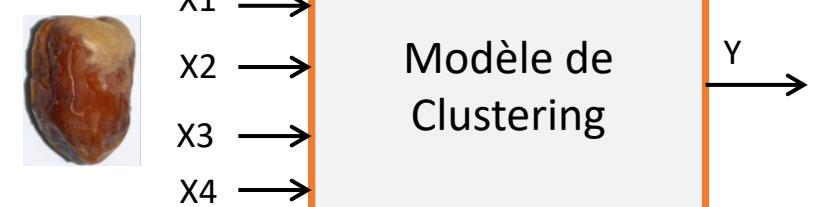
# Apprentissage Non Supervisé : K-means

- L'algorithme des **k-means** (ou bien *k-moyennes*) est l'un des algorithmes d'apprentissage non supervisé qui permet d'analyser un jeu de données afin de **regrouper les données « similaires » en groupes (ou clusters)**.
- Ce principe est applicable dans divers domaines de l'industrie comme le domaine du marketing pour la segmentation des clients, analyse des réseaux sociaux, segmentation des images médicales.
- Principe de Fonctionnement :
  1. Choisir le nombre de clusters **K**.
  2. Initialiser les centroïdes (Centres de classes)
  3. Calculer la distance aux centres de classes pour tous les points du data set
  4. Attribuer chaque point de données au centroïde le plus proche.
  5. Mettre à jour le centroïde en calculant la moyenne du cluster.
  6. Répéter les étapes 3, 4 et 5 jusqu'à convergence, c'est-à-dire jusqu'à ce qu'aucun changement discernable dans les centroïdes ne soit observé.

## Apprentissage Non Supervisé

### Clustering

#### Data Set (Données Non étiquetées)

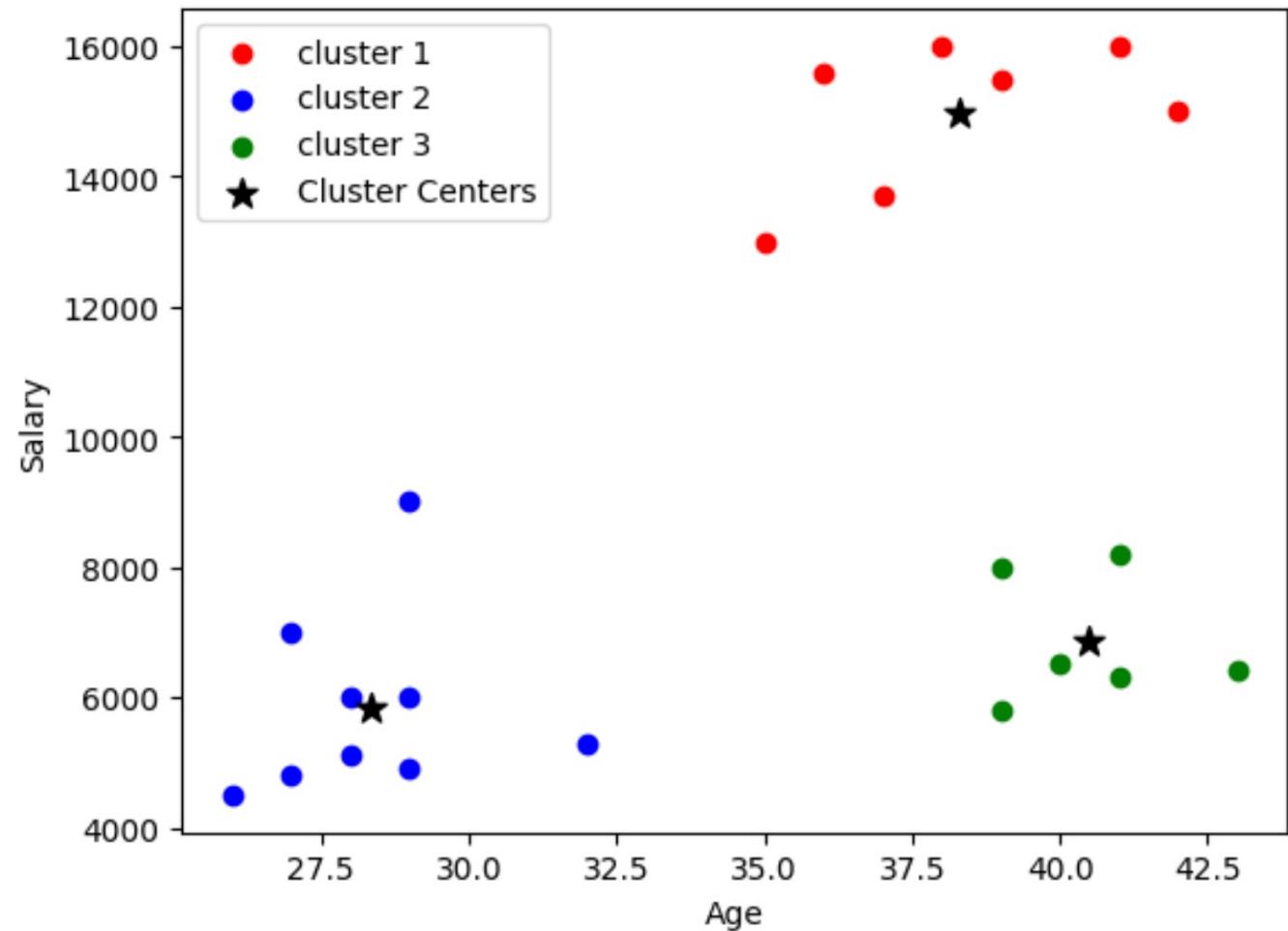


Algorithmes : K-Means

# Apprentissage Non Supervisé : K-means

- **Principe de Fonctionnement :**

1. Choisir le nombre de clusters K.
2. Initialiser les centroïdes (Centres de classes)
3. Calculer la distance aux centres de classes pour tous les points du data set
4. Attribuer chaque point de données au centroïde le plus proche.
5. Mettre à jour le centroïde en calculant la moyenne du cluster.
6. Répéter les étapes 3, 4 et 5 jusqu'à convergence, c'est-à-dire jusqu'à ce qu'aucun changement discernable dans les centroïdes ne soit observé.



# Apprentissage Non Supervisé : K-means

- Exemple simple : Considérant une population de différents âges.

|    |     | C1 | C2 |         |
|----|-----|----|----|---------|
| N° | Age | 18 | 30 | Cluster |
| 1  | 12  | 6  | 18 | C1      |
| 2  | 25  | 7  | 5  | C2      |
| 3  | 30  | 12 | 0  | C2      |
| 4  | 18  | 0  | 12 | C1      |
| 5  | 29  | 11 | 1  | C2      |
| 6  | 40  | 22 | 10 | C2      |
| 7  | 44  | 26 | 14 | C2      |
| 8  | 32  | 14 | 2  | C2      |
| 9  | 18  | 0  | 12 | C1      |
| 10 | 52  | 34 | 22 | C2      |

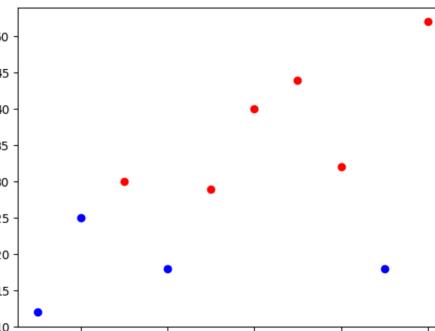
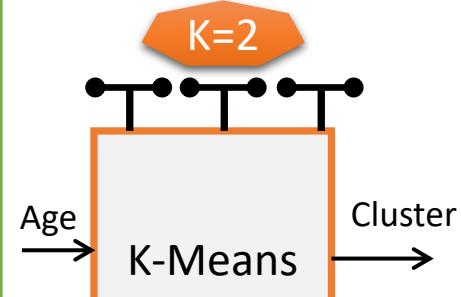
|    |     | C1 | C2 |         |
|----|-----|----|----|---------|
| N° | Age | 16 | 36 | Cluster |
| 1  | 12  | 4  | 24 | C1      |
| 2  | 25  | 9  | 11 | C1      |
| 3  | 30  | 14 | 6  | C2      |
| 4  | 18  | 2  | 18 | C1      |
| 5  | 29  | 13 | 7  | C2      |
| 6  | 40  | 24 | 4  | C2      |
| 7  | 44  | 28 | 8  | C2      |
| 8  | 32  | 16 | 4  | C2      |
| 9  | 18  | 2  | 18 | C1      |
| 10 | 52  | 36 | 16 | C2      |

|    |     | C1    | C2    |         |
|----|-----|-------|-------|---------|
| N° | Age | 18,25 | 37,83 | Cluster |
| 1  | 12  | 6,25  | 25,83 | C1      |
| 2  | 25  | 6,75  | 12,83 | C1      |
| 3  | 30  | 11,75 | 7,83  | C2      |
| 4  | 18  | 0,25  | 19,83 | C1      |
| 5  | 29  | 10,75 | 8,83  | C2      |
| 6  | 40  | 21,75 | 2,17  | C2      |
| 7  | 44  | 25,75 | 6,17  | C2      |
| 8  | 32  | 13,75 | 5,83  | C2      |
| 9  | 18  | 0,25  | 19,83 | C1      |
| 10 | 52  | 33,75 | 14,17 | C2      |

|    |     | C1 | C2 |         |
|----|-----|----|----|---------|
| N° | Age | 18 | 30 | Cluster |
| 1  | 12  | 6  | 18 | C1      |
| 4  | 18  | 0  | 12 | C1      |
| 9  | 18  | 0  | 12 | C1      |
|    | 16  |    |    |         |
|    |     | C1 | C2 |         |
| N° | Age | 18 | 30 | Culster |
| 2  | 25  | 7  | 5  | C2      |
| 3  | 30  | 12 | 0  | C2      |
| 5  | 29  | 11 | 1  | C2      |
| 6  | 40  | 22 | 10 | C2      |
| 7  | 44  | 26 | 14 | C2      |
| 8  | 32  | 14 | 2  | C2      |
| 10 | 52  | 34 | 22 | C2      |
|    | 36  |    |    |         |

|    |       | C1 | C2 |         |
|----|-------|----|----|---------|
| N° | Age   | 16 | 36 | Cluster |
| 1  | 12    | 4  | 24 | C1      |
| 2  | 25    | 9  | 11 | C1      |
| 4  | 18    | 2  | 18 | C1      |
| 9  | 18    | 2  | 18 | C1      |
|    | 18,25 |    |    |         |
|    |       | C1 | C2 |         |
| N° | Age   | 16 | 36 | Cluster |
| 3  | 30    | 14 | 6  | C2      |
| 5  | 29    | 13 | 7  | C2      |
| 6  | 40    | 24 | 4  | C2      |
| 7  | 44    | 28 | 8  | C2      |
| 8  | 32    | 16 | 4  | C2      |
| 10 | 52    | 36 | 16 | C2      |
|    | 37,83 |    |    |         |

|    |       | C1    | C2    |         |
|----|-------|-------|-------|---------|
| N° | Age   | 18,25 | 37,83 | Cluster |
| 1  | 12    | 6,25  | 25,83 | C1      |
| 2  | 25    | 6,75  | 12,83 | C1      |
| 4  | 18    | 0,25  | 19,83 | C1      |
| 9  | 18    | 0,25  | 19,83 | C1      |
|    | 18,25 |       |       |         |
|    |       | C1    | C2    |         |
| N° | Age   | 18,25 | 37,83 | Cluster |
| 3  | 30    | 11,75 | 7,83  | C2      |
| 5  | 29    | 10,75 | 8,83  | C2      |
| 6  | 40    | 21,75 | 2,17  | C2      |
| 7  | 44    | 25,75 | 6,17  | C2      |
| 8  | 32    | 13,75 | 5,83  | C2      |
| 10 | 52    | 33,75 | 14,17 | C2      |
|    | 37,83 |       |       |         |



# Apprentissage Non Supervisé : K-means : Performances de K-Means

- Pour évaluer la qualité de ces regroupements, nous utilisons deux métriques clés :
  - Distorsion
  - l'Inertie (Inertia).
- **Distorsion :**
  - La distorsion mesure la distance moyenne au carré entre chaque point de données et le centre de son cluster attribué. C'est une mesure de la qualité de représentation des données par les clusters. Une valeur de distorsion plus faible indique un meilleur clustering.

$$\text{Distortion} = \frac{1}{n} \sum_{i=1}^n \text{distance}(x_i, c_j)^2$$

- $(x_i)$  : le i-ème point de données
- $(c_j)$  : le centroïde du cluster auquel  $(x_i)$  appartient, c'est-à-dire la distance (point, centroïde) : distance euclidienne entre un point de données et le centre de cluster attribué.

- **Inertie (Initia):**

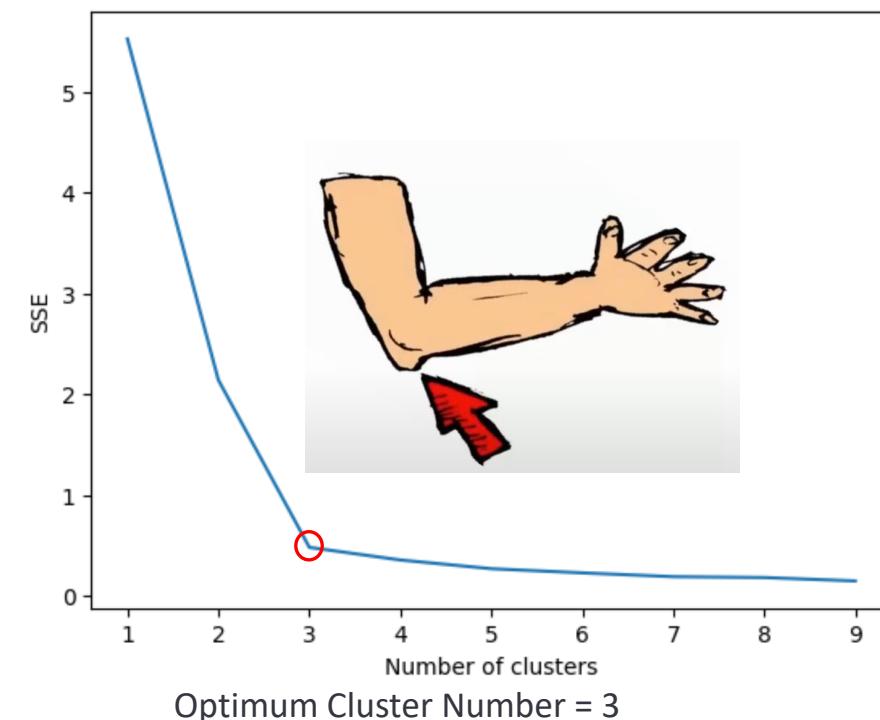
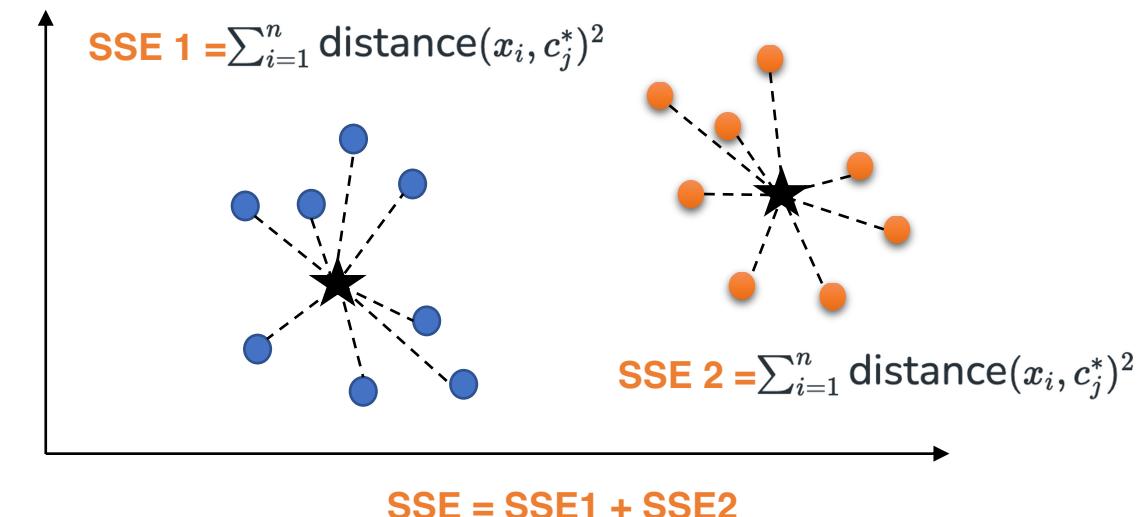
- L'inertie est la somme des distances au carré de chaque point de données à son centre de cluster le plus proche.
- C'est essentiellement l'erreur totale au carré du clustering.
- Comme la distorsion, une valeur d'inertie plus faible suggère un meilleur clustering.
- L'inertie est le numérateur de la formule de la Distorsion; la Distorsion est l'inertie moyenne par point de données.

$$\text{Inertia} = \sum_{i=1}^n \text{distance}(x_i, c_j^*)^2$$

---

## Apprentissage Non Supervisé : K-means : Nombre optimal de clusters (ELBOW)

- Pour K- means, nous avons besoin d'une **valeur fixe de K**, qui doit être connue avant d'effectuer le clustering.
- Pour trouver le nombre optimal de K, nous pouvons utiliser la méthode **ELBOW** (Coude).
- **Méthode Elbow :**
  - Dans cette méthode, nous prenons différentes valeurs de K, de 1 jusqu'à la plage requise.
  - Pour chaque valeur de K, nous calculons la somme des carrés intra-cluster, généralement appelée WCSS (Within Cluster Sum of Squares).
  - Nous calculons donc la somme des distances au carré (**SSE**) entre les centroïdes et les points pour chaque cluster et nous faisons une somme de tous les clusters.
  - Ensuite, nous traçons **SSE** en fonction de K - la forme du graphique ressemble à celle d'un coude.
  - Comme on le voit sur le graphique, à mesure que K augmente, le SSE diminue.
  - Le point optimal se trouve au niveau du coude - après cela, à mesure que K augmente, la diminution du SSE n'est pas si significative. Dans l'exemple donné, K=3semble être la valeur optimale de K.



# Apprentissage Non Supervisé : K-means : Nombre optimal clusters

```
from sklearn.cluster import KMeans  
import pandas as pd  
import matplotlib.pyplot as plt
```

✓ 8.7s

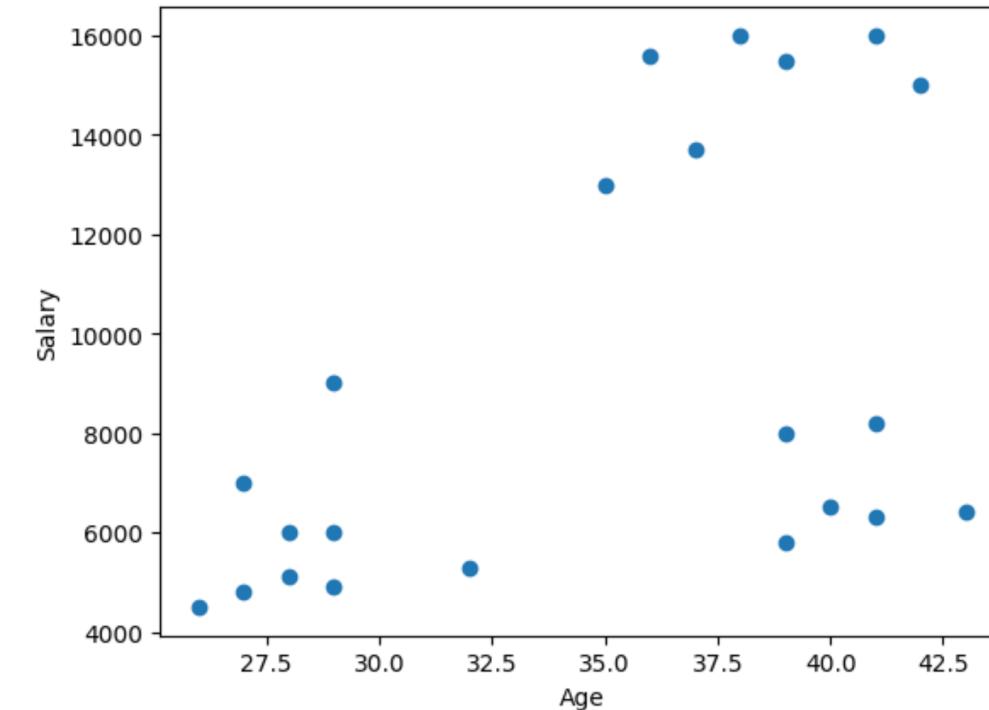
```
df = pd.read_csv('salaires.csv')
```

✓ 0.0s

|    | Name     | Age | Salary |
|----|----------|-----|--------|
| 0  | Ines     | 29  | 4900   |
| 1  | Omar     | 32  | 5300   |
| 2  | Sabrina  | 40  | 6500   |
| 3  | Souad    | 41  | 6300   |
| 4  | Nezha    | 43  | 6400   |
| 5  | Rabia    | 39  | 8000   |
| 6  | Soufiane | 41  | 8200   |
| 7  | Abdellah | 39  | 5800   |
| 8  | Mohamed  | 27  | 7000   |
| 9  | Yassine  | 29  | 9000   |
| 10 | Azizi    | 38  | 16000  |
| 11 | Ahmed    | 36  | 15600  |
| 12 | Yasmine  | 35  | 13000  |
| 13 | Aya      | 37  | 13700  |
| 14 | Zakaria  | 26  | 4500   |
| 15 | Fatima   | 27  | 4800   |
| 16 | Salim    | 28  | 5100   |
| 17 | Imane    | 29  | 6000   |
| 18 | Ismail   | 28  | 6000   |
| 19 | Malak    | 42  | 15000  |
| 20 | Hanane   | 39  | 15500  |
| 21 | Ibrahim  | 41  | 16000  |

```
plt.scatter(df['Age'], df['Salary'])  
plt.xlabel("Age")  
plt.ylabel("Salary")
```

✓ 0.0s  
Text(0, 0.5, 'Salary')



# Apprentissage Non Supervisé : K-means : Nombre optimal clusters

```
km = KMeans(n_clusters=3, random_state=0)
```

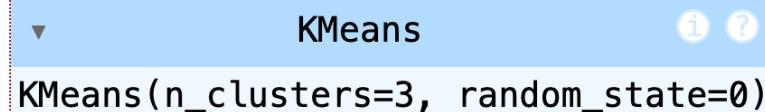
✓ 0.0s

```
X = df.drop(columns=['Name'])
```

✓ 0.0s

```
km.fit(X)
```

✓ 0.2s



```
km.labels_
```

✓ 0.0s

```
array([1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0],  
      dtype=int32)
```

```
predicted = km.predict(X)
```

✓ 0.0s

```
predicted
```

✓ 0.0s

```
array([1, 1, 1, 1, 1, 2, 2, 1, 2, 2, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0],  
      dtype=int32)
```

```
km.labels_==predicted
```

✓ 0.0s

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  
       True,  True,  True,  True,  True,  True,  True,  True,  
       True,  True,  True,  True])
```

# Apprentissage Non Supervisé : K-means : Nombre optimal clusters

```
df['cluster']= predicted
```

✓ 0.0s

df

✓ 0.0s

|  | Name | Age | Salary | cluster |
|--|------|-----|--------|---------|
|--|------|-----|--------|---------|

|    |          |    |       |   |
|----|----------|----|-------|---|
| 0  | Ines     | 29 | 4900  | 1 |
| 1  | Omar     | 32 | 5300  | 1 |
| 2  | Sabrina  | 40 | 6500  | 1 |
| 3  | Souad    | 41 | 6300  | 1 |
| 4  | Nezha    | 43 | 6400  | 1 |
| 5  | Rabia    | 39 | 8000  | 2 |
| 6  | Soufiane | 41 | 8200  | 2 |
| 7  | Abdellah | 39 | 5800  | 1 |
| 8  | Mohamed  | 27 | 7000  | 2 |
| 9  | Yassine  | 29 | 9000  | 2 |
| 10 | Azizi    | 38 | 16000 | 0 |
| 11 | Ahmed    | 36 | 15600 | 0 |
| 12 | Yasmine  | 35 | 13000 | 0 |
| 13 | Aya      | 37 | 13700 | 0 |
| 14 | Zakaria  | 26 | 4500  | 1 |
| 15 | Fatima   | 27 | 4800  | 1 |
| 16 | Salim    | 28 | 5100  | 1 |
| 17 | Imane    | 29 | 6000  | 1 |
| 18 | Ismail   | 28 | 6000  | 1 |
| 19 | Malak    | 42 | 15000 | 0 |
| 20 | Hanane   | 39 | 15500 | 0 |
| 21 | Ibrahim  | 41 | 16000 | 0 |

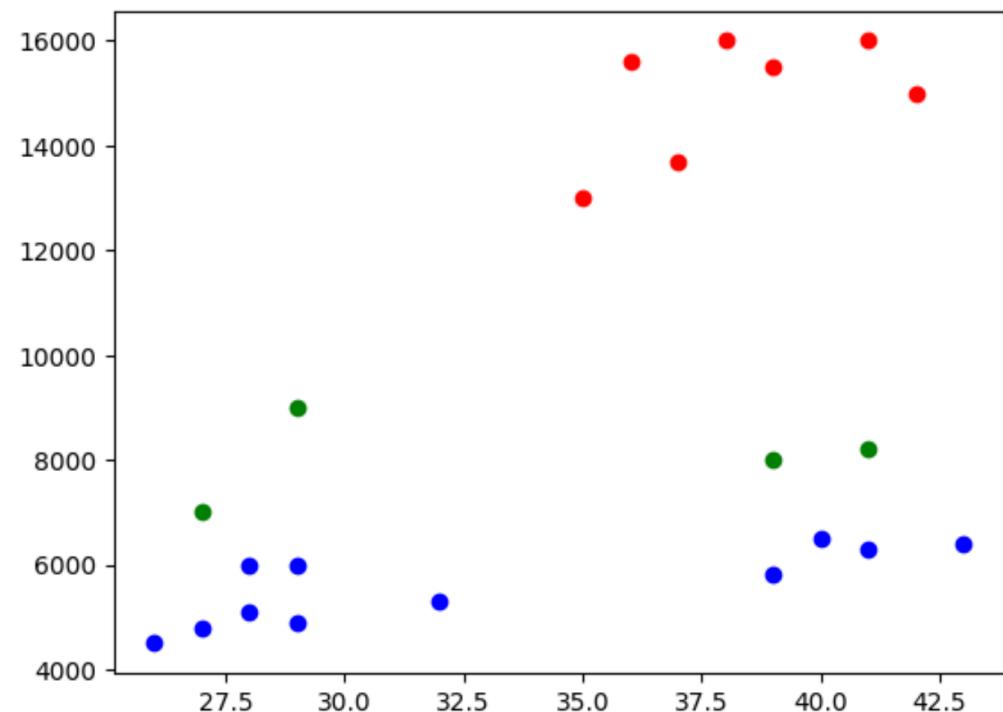
```
cluster1 = df[df['cluster']==0]  
cluster2 = df[df['cluster']==1]  
cluster3 = df[df['cluster']==2]
```

✓ 0.0s

```
plt.scatter(cluster1['Age'], cluster1['Salary'], color='red')  
plt.scatter(cluster2['Age'], cluster2['Salary'], color='blue')  
plt.scatter(cluster3['Age'], cluster3['Salary'], color='green')
```

✓ 0.0s

<matplotlib.collections.PathCollection at 0x29feaf390>



# Apprentissage Non Supervisé : Normalisation et standardisation des données

- La normalisation et la standardisation sont souvent utilisés pour décrire un ensemble de méthodes permettant d'effectuer une mise à échelle, mais ils sont aussi utilisés pour décrire une technique bien précise de redimensionnement de variables.
- 3 types de normalisation :
- **Normalisation Min-Max**

Le procédé de normalisation Min-Max utilise le minimum et le maximum de la variable.

L'idée est de ramener toutes les valeurs de la variable dans l'intervalle [0;1], tout en conservant le rapport des distances entre les valeurs.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- **Standardisation**

Pour ce qui est de la standardisation, la transformation a pour but de ramener la moyenne  $\mu$  à 0 et l'écart-type  $\sigma$  à 1. Encore une fois, le procédé est simple si on a à notre disposition la moyenne  $\mu$  et l'écart-type  $\sigma$  de la variable, la formule pour le faire est la suivante :

Ces deux techniques sont sensibles aux valeurs aberrantes, donc moins efficaces. Pour des variables ayant des valeurs aberrantes, il est préférable d'utiliser la normalisation robuste qui est peu sensible aux valeurs aberrantes.

$$X_{stand} = \frac{X - \mu}{\sigma}$$

- **Robuste**

Cette technique transforme chaque variable en étant peu sensible aux outliers (valeurs aberrantes). Le procédé est le suivant: on soustrait aux valeurs de la variable la médiane et on divise par l'écart interquartile IQR (IQR est une mesure de dispersion qui s'obtient en faisant la différence entre le troisième et le premier quartile)

$$X_{rob} = \frac{X - mediane}{IQR}$$

## Apprentissage Non Supervisé : K-means : Nombre optimal clusters

```
from sklearn.preprocessing import MinMaxScaler  
from sklearn.pipeline import Pipeline
```

✓ 0.0s

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

✓ 0.0s

```
scaler.inverse_transform([[0.88235294, 1.]]))
```

✓ 0.0s

```
array([[ 40.9999998, 16000. ]])
```

X\_scaled

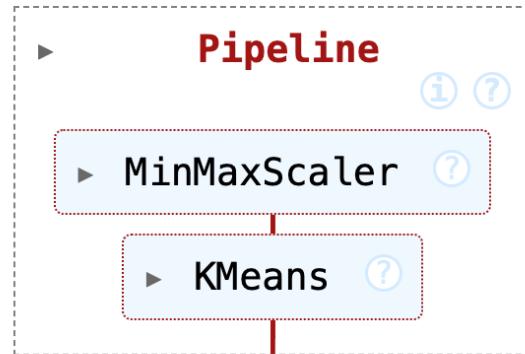
✓ 0.0s

```
array([[0.17647059, 0.03478261],  
[0.35294118, 0.06956522],  
[0.82352941, 0.17391304],  
[0.88235294, 0.15652174],  
[1. , 0.16521739],  
[0.76470588, 0.30434783],  
[0.88235294, 0.32173913],  
[0.76470588, 0.11304348],  
[0.05882353, 0.2173913 ],  
[0.17647059, 0.39130435],  
[0.70588235, 1. ],  
[0.58823529, 0.96521739],  
[0.52941176, 0.73913043],  
[0.64705882, 0.8 ],  
[0. , 0. ],  
[0.05882353, 0.02608696],  
[0.11764706, 0.05217391],  
[0.17647059, 0.13043478],  
[0.11764706, 0.13043478],  
[0.94117647, 0.91304348],  
[0.76470588, 0.95652174],  
[0.88235294, 1. ]])
```

## Apprentissage Non Supervisé : K-means : Nombre optimal clusters

```
estimator = Pipeline([
    ("MinMaxScaler", MinMaxScaler()),
    ("clustering", KMeans(n_clusters=3, random_state=0))
])
✓ 0.0s
```

```
estimator.fit(X)
✓ 0.0s
```



## Apprentissage Non Supervisé : K-means : Nombre optimal clusters

```
predicted2 = estimator.predict(X)
```

✓ 0.0s

```
df['cluster2'] = predicted2
```

✓ 0.0s

```
cluster1 = df[df['cluster2']==0]
```

```
cluster2 = df[df['cluster2']==1]
```

```
cluster3 = df[df['cluster2']==2]
```

✓ 0.0s

```
centers_scaled = estimator[-1].cluster_centers_
centers = estimator[0].inverse_transform(centers_scaled)
```

✓ 0.0s

```
centers_scaled
```

✓ 0.0s

```
array([[0.72268908, 0.91055901],
       [0.1372549 , 0.11690821],
       [0.85294118, 0.2057971 ]])
```

```
centers
```

✓ 0.0s

```
array([[ 38.28571429, 14971.42857143],
       [ 28.33333333,  5844.44444444],
       [ 40.5          ,  6866.66666667]])
```

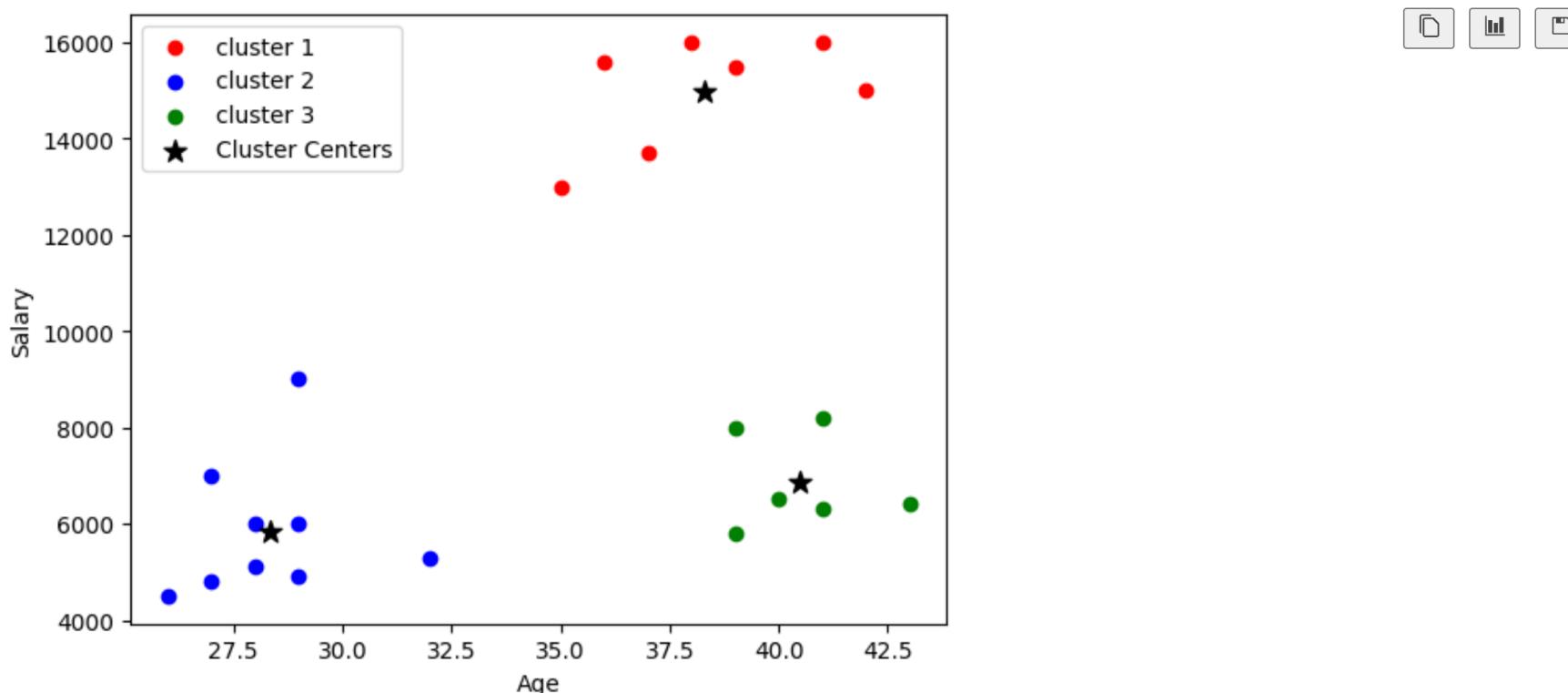
# Apprentissage Non Supervisé : K-means : Nombre optimal clusters

```
plt.scatter(cluster1['Age'], cluster1['Salary'], color='red', label = "cluster 1")
plt.scatter(cluster2['Age'], cluster2['Salary'], color='blue', label = "cluster 2")
plt.scatter(cluster3['Age'], cluster3['Salary'], color='green', label = "cluster 3")
plt.scatter(centers[:,0], centers[:,1], color="black", marker='*', s=100, label="Cluster Centers")
plt.xlabel("Age")
plt.ylabel("Salary")
plt.legend()
```

✓ 0.0s

Python

<matplotlib.legend.Legend at 0x2aa9b9d10>



## Apprentissage Non Supervisé : K-means : Nombre optimal clusters

```
from sklearn.metrics import silhouette_score
sse=[]
sil=[]
krange = range(1,10)
for k in krange:
    km = Pipeline([
        ("MinMaxScaler", MinMaxScaler()),
        ("clustering", KMeans(n_clusters=k, random_state=0))
    ])
    km.fit(X)
    sse.append(km[-1].inertia_)
    if(k==1):
        sil.append(0)
    else :
        sil.append(silhouette_score(X, km[-1].labels_))

✓ 0.0s
```

sse

✓ 0.0s

```
[5.522140987850921,
 2.139183706762151,
 0.4796830920366221,
 0.3546287126734011,
 0.26819371493732613,
 0.22690756427112166,
 0.18904083568265526,
 0.18057431161055554,
 0.14689918738539565]
```

sil

✓ 0.0s

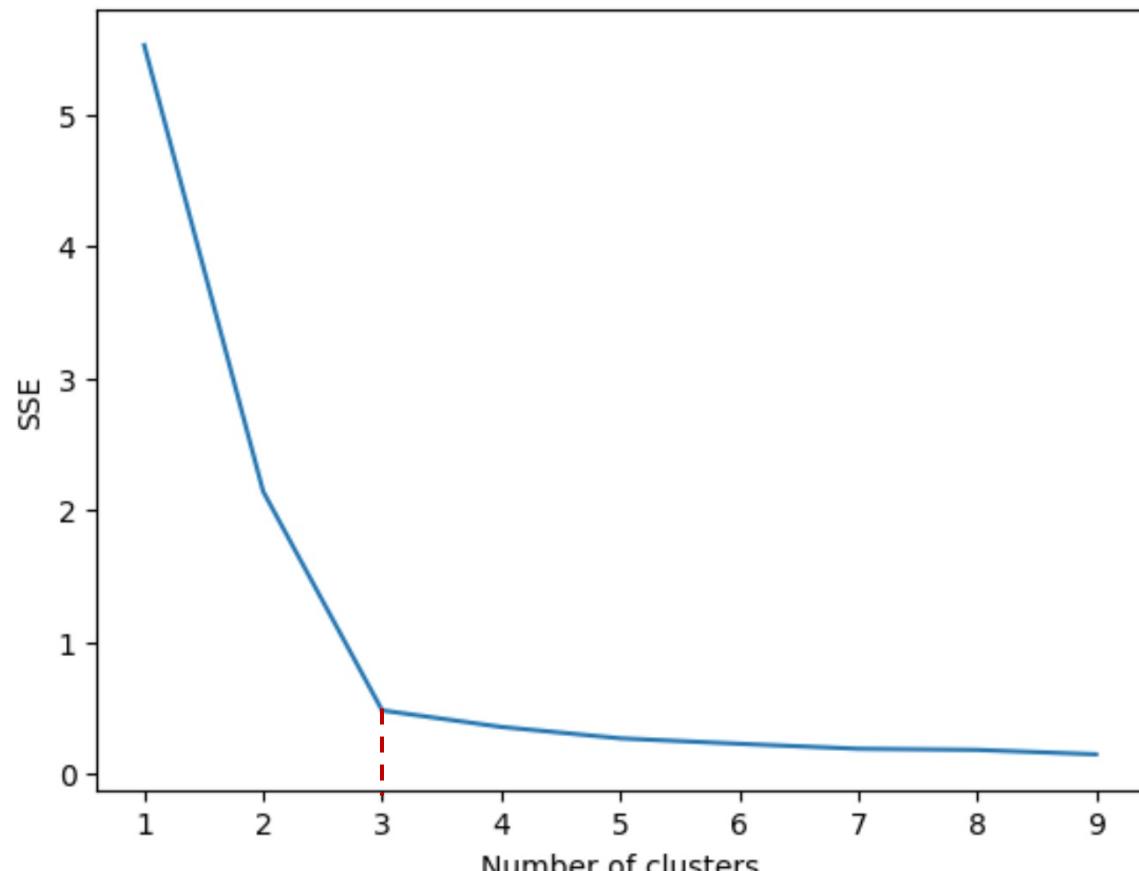
```
[0,
 np.float64(0.25657476200964113),
 np.float64(0.35398455359639963),
 np.float64(0.1995366438963641),
 np.float64(0.19160118123962888),
 np.float64(0.0694716270838743),
 np.float64(-0.020201042966170576),
 np.float64(0.1380407992851944),
 np.float64(0.2576952732350539)]
```

## Apprentissage Non Supervisé : K-means : Nombre optimal clusters

```
plt.plot(krange, sse,)  
plt.xlabel("Number of clusters")  
plt.ylabel("SSE")
```

✓ 0.0s

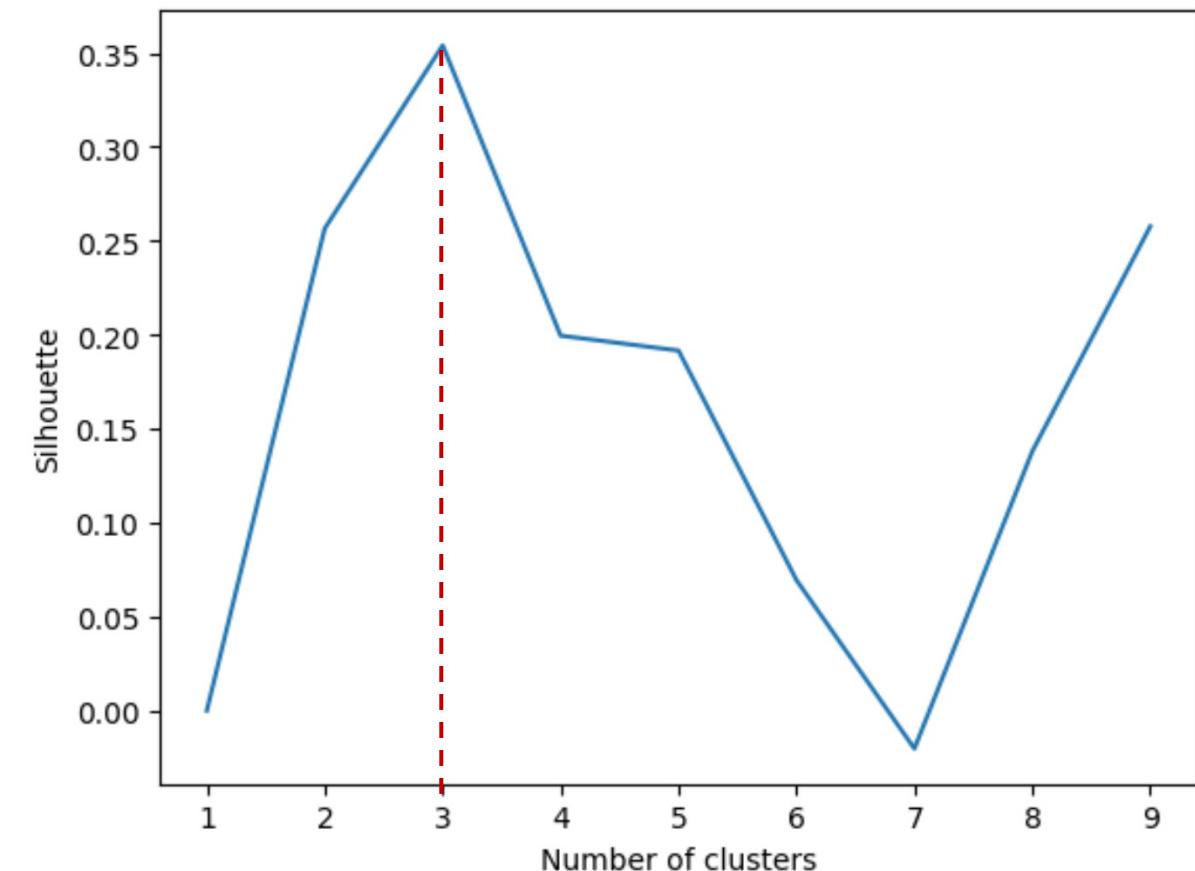
```
Text(0, 0.5, 'SSE')
```



```
plt.plot(krange, sil,)  
plt.xlabel("Number of clusters")  
plt.ylabel("Silhouette")
```

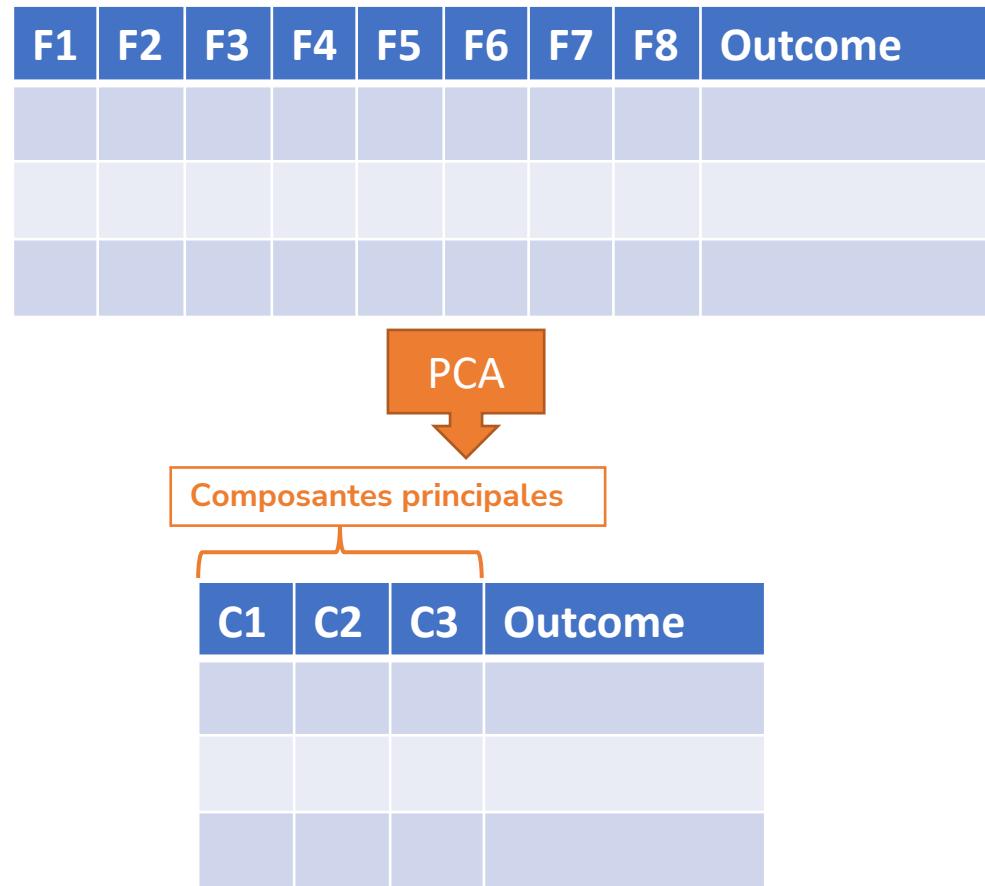
✓ 0.0s

```
Text(0, 0.5, 'Silhouette')
```



# Principal Component Analysis : PCA

- L'Analyse en Composantes Principales (PCA) est une méthode largement utilisée en statistique et en apprentissage automatique pour la réduction de dimensionnalité des données.
- **Réduction de dimensionnalité** : La PCA vise à réduire le nombre de variables d'un ensemble de données tout en préservant autant que possible la variance des données originales.
- **Composantes principales** : En PCA, les nouvelles variables, appelées composantes principales, sont des combinaisons linéaires des variables d'origine. Elles sont ordonnées en fonction de leur importance dans la description de la variance des données.
- **Corrélation** : PCA suppose que les variables sont linéairement corrélées. Elle cherche à trouver les axes de projection qui maximisent la variance des données.
- **Décomposition en valeurs singulières** : La méthode la plus couramment utilisée pour calculer la PCA est la décomposition en valeurs singulières (**SVD**) des données centrées. Elle permet de calculer les **vecteurs propres et les valeurs propres** de la matrice de covariance des données.
- **Interprétation des composantes principales** : Chaque composante principale représente une direction dans l'espace des variables d'origine. Elles sont souvent interprétées comme des patterns ou des combinaisons de variables.



```
from sklearn.datasets import load_digits
import pandas as pd

dataset = load_digits()
data = dataset.data
data.shape
```

✓ 0.0s

✓ 0.0s

dict\_keys(['data', 'target', 'frame', 'feature\_names', 'target\_names', 'images', 'DESCR'])

✓ 0.0s

(1797, 64)

```
data[0]
✓ 0.0s
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
       0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])

data[0].reshape(8,8)
✓ 0.0s
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

+ Code + Marquage

```
from matplotlib import pyplot as plt
```

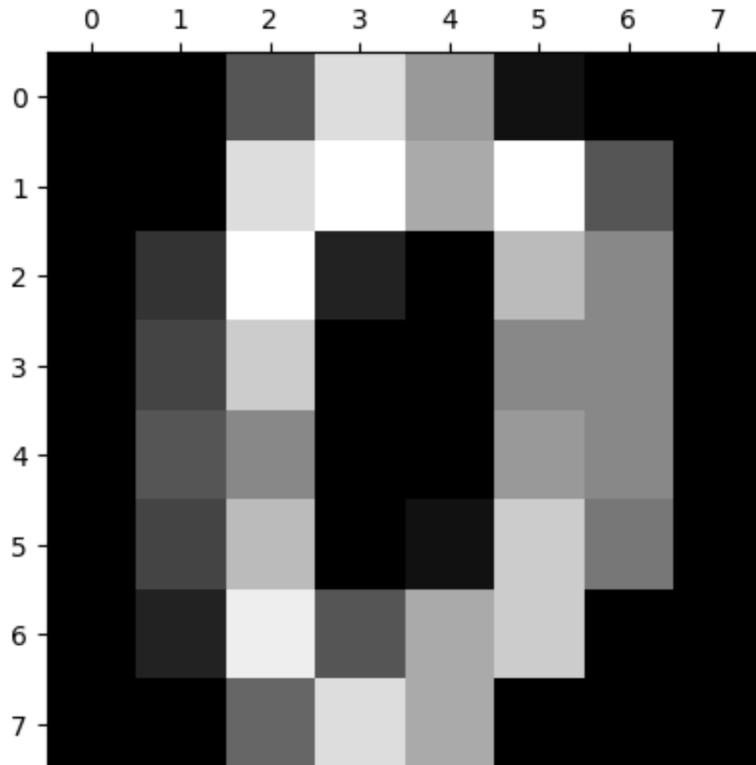
✓ 0.0s

```
plt.gray()  
plt.matshow(dataset.data[0].reshape(8,8))
```

✓ 0.0s

<matplotlib.image.AxesImage at 0x2aad07390>

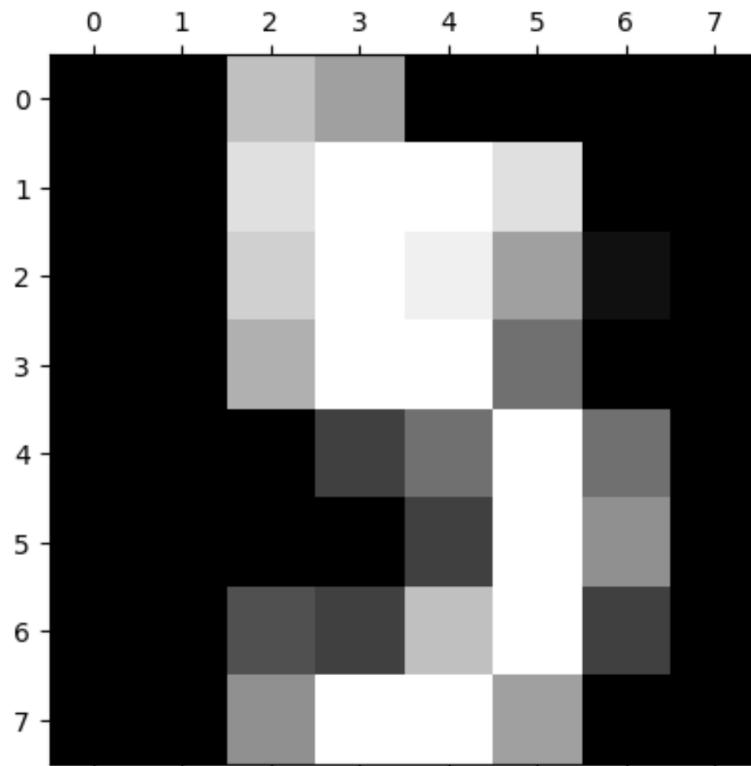
<Figure size 640x480 with 0 Axes>



```
plt.matshow(dataset.data[5].reshape(8,8))
```

✓ 0.0s

<matplotlib.image.AxesImage at 0x2aacdbd90>



```
dataset.target[:20]
```

✓ 0.0s

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
df = pd.DataFrame(data=data, columns=dataset['feature_names'])
```

✓ 0.0s

Python

```
df
```

✓ 0.0s

Python

|      | pixel_0_0 | pixel_0_1 | pixel_0_2 | pixel_0_3 | pixel_0_4 | pixel_0_5 | pixel_0_6 | pixel_0_7 | pixel_1_0 | pixel_1_1 | ... |
|------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----|
| 0    | 0.0       | 0.0       | 5.0       | 13.0      | 9.0       | 1.0       | 0.0       | 0.0       | 0.0       | 0.0       | ... |
| 1    | 0.0       | 0.0       | 0.0       | 12.0      | 13.0      | 5.0       | 0.0       | 0.0       | 0.0       | 0.0       | ... |
| 2    | 0.0       | 0.0       | 0.0       | 4.0       | 15.0      | 12.0      | 0.0       | 0.0       | 0.0       | 0.0       | ... |
| 3    | 0.0       | 0.0       | 7.0       | 15.0      | 13.0      | 1.0       | 0.0       | 0.0       | 0.0       | 8.0       | ... |
| 4    | 0.0       | 0.0       | 0.0       | 1.0       | 11.0      | 0.0       | 0.0       | 0.0       | 0.0       | 0.0       | ... |
| ...  | ...       | ...       | ...       | ...       | ...       | ...       | ...       | ...       | ...       | ...       | ... |
| 1792 | 0.0       | 0.0       | 4.0       | 10.0      | 13.0      | 6.0       | 0.0       | 0.0       | 0.0       | 1.0       | ... |
| 1793 | 0.0       | 0.0       | 6.0       | 16.0      | 13.0      | 11.0      | 1.0       | 0.0       | 0.0       | 0.0       | ... |
| 1794 | 0.0       | 0.0       | 1.0       | 11.0      | 15.0      | 1.0       | 0.0       | 0.0       | 0.0       | 0.0       | ... |
| 1795 | 0.0       | 0.0       | 2.0       | 10.0      | 7.0       | 0.0       | 0.0       | 0.0       | 0.0       | 0.0       | ... |
| 1796 | 0.0       | 0.0       | 10.0      | 14.0      | 8.0       | 1.0       | 0.0       | 0.0       | 0.0       | 2.0       | ... |

1797 rows × 64 columns

```
df.describe()
```

✓ 0.0s

|       | pixel_0_0 | pixel_0_1   | pixel_0_2   | pixel_0_3   | pixel_0_4   | pixel_0_5   | pixel_0_6   | pixel_0_7   |
|-------|-----------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| count | 1797.0    | 1797.000000 | 1797.000000 | 1797.000000 | 1797.000000 | 1797.000000 | 1797.000000 | 1797.000000 |
| mean  | 0.0       | 0.303840    | 5.204786    | 11.835838   | 11.848080   | 5.781859    | 1.362270    | 0.129661    |
| std   | 0.0       | 0.907192    | 4.754826    | 4.248842    | 4.287388    | 5.666418    | 3.325775    | 1.037383    |
| min   | 0.0       | 0.000000    | 0.000000    | 0.000000    | 0.000000    | 0.000000    | 0.000000    | 0.000000    |
| 25%   | 0.0       | 0.000000    | 1.000000    | 10.000000   | 10.000000   | 0.000000    | 0.000000    | 0.000000    |
| 50%   | 0.0       | 0.000000    | 4.000000    | 13.000000   | 13.000000   | 4.000000    | 0.000000    | 0.000000    |
| 75%   | 0.0       | 0.000000    | 9.000000    | 15.000000   | 15.000000   | 11.000000   | 0.000000    | 0.000000    |
| max   | 0.0       | 8.000000    | 16.000000   | 16.000000   | 16.000000   | 16.000000   | 16.000000   | 15.000000   |

8 rows × 64 columns

# Logistic Regression

```
X = df  
y = dataset.target  
[] ✓ 0.0s
```

```
from sklearn.preprocessing import StandardScaler  
from sklearn.model_selection import train_test_split  
from sklearn.linear_model import LogisticRegression  
from sklearn.decomposition import PCA  
[] ✓ 0.0s
```

```
scaler = StandardScaler()  
X_scaled = scaler.fit_transform(X)  
X_scaled  
[] ✓ 0.0s
```

```
array([[ 0.        , -0.33501649, -0.04308102, ..., -1.14664746,  
       -0.5056698 , -0.19600752],  
      [ 0.        , -0.33501649, -1.09493684, ...,  0.54856067,  
       -0.5056698 , -0.19600752],  
      [ 0.        , -0.33501649, -1.09493684, ...,  1.56568555,  
       1.6951369 , -0.19600752],  
      ...,  
      [ 0.        , -0.33501649, -0.88456568, ..., -0.12952258,  
       -0.5056698 , -0.19600752],  
      [ 0.        , -0.33501649, -0.67419451, ...,  0.8876023 ,  
       -0.5056698 , -0.19600752],  
      [ 0.        , -0.33501649,  1.00877481, ...,  0.8876023 ,  
       -0.26113572, -0.19600752]], shape=(1797, 64))
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=30)
```

```
✓ 0.0s
```

Python

```
model = LogisticRegression()  
model.fit(X_train, y_train)  
model.score(X_test, y_test)
```

```
✓ 0.6s
```

Python

0.972222222222222

# PCA

```
pca = PCA(0.95)
X_pca = pca.fit_transform(X)
X_pca.shape
```

5] ✓ 0.0s

(1797, 29)

```
evr=pca.explained_variance_ratio_
```

6] ✓ 0.0s

7] evr

8] ✓ 0.0s

```
array([0.14890594, 0.13618771, 0.11794594, 0.08409979, 0.05782415,
       0.0491691 , 0.04315987, 0.03661373, 0.03353248, 0.03078806,
       0.02372341, 0.02272697, 0.01821863, 0.01773855, 0.01467101,
       0.01409716, 0.01318589, 0.01248138, 0.01017718, 0.00905617,
       0.00889538, 0.00797123, 0.00767493, 0.00722904, 0.00695889,
       0.00596081, 0.00575615, 0.00515158, 0.0048954 ])
```

9] evr.sum()

✓ 0.0s

```
np.float64(0.9547965245651596)
```

```
pca.n_components_
```

✓ 0.0s

```
np.int64(29)
```

```
X_pca
```

✓ 0.0s

```
array([[ -1.25946645, -21.27488348,  9.46305462, ...,  3.67072108,
          0.9436689 ,  1.13250195],
       [ 7.9576113 ,  20.76869896, -4.43950604, ...,  2.18261819,
       0.51022719, -2.31354911],
       [ 6.99192297,  9.95598641, -2.95855808, ...,  4.22882114,
       -2.1576573 , -0.8379578 ],
       ...,
       [ 10.8012837 ,  6.96025223, -5.59955453, ..., -3.56866194,
       -1.82444444, -3.53885886],
       [-4.87210009, -12.42395362,  10.17086635, ...,  3.25330054,
       -0.95484174,  0.93895602],
       [ -0.34438963, -6.36554919, -10.77370849, ..., -3.01636722,
       -1.29752723, -2.58810313]], shape=(1797, 29))
```

```
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=30)
```

✓ 0.0s

```
model = LogisticRegression(max_iter=1000)
model.fit(X_train_pca, y_train)
model.score(X_test_pca, y_test)
```

✓ 0.0s

```
0.9722222222222222
```

```
pca2 = PCA(n_components=2)
X_pca2 = pca2.fit_transform(X)
X_pca2.shape
✓ 0.0s
(1797, 2)
```

```
X_pca2
✓ 0.0s
array([[ -1.25946645, -21.27488348],
       [  7.9576113 ,  20.76869896],
       [  6.99192297,   9.95598641],
       ...,
       [ 10.8012837 ,   6.96025223],
       [-4.87210009, -12.42395362],
       [-0.34438963, -6.36554919]], shape=(1797, 2))
```

```
pca2.explained_variance_ratio_
✓ 0.0s
array([0.14890594, 0.13618771])

pca2.explained_variance_ratio_.sum()
✓ 0.0s
np.float64(0.285093648236993)
```

```
X_train_pca2, X_test_pca2, y_train2, y_test2 = train_test_split(X_pca2, y, test_size=0.2, random_state=30)

model = LogisticRegression(max_iter=1000)
model.fit(X_train_pca2, y_train2)
model.score(X_test_pca2, y_test2)
✓ 0.1s
0.6083333333333333
```

# Quelques Rappels en statistiques

## Analyse de données statistiques : espérance, variance, écart-type

L'**espérance** d'une série statistique  $X = (x_i)_{i=1,\dots,N}$  est la moyenne des valeurs de cette série statistique :

$$E(X) = \bar{X} = \frac{x_1 + \cdots + x_N}{N}.$$

La **variance** et l'écart-type mesurent eux la dispersion des valeurs de cette série statistique autour de sa moyenne. La variance  $V(X)$  est définie par

$$V(X) = \frac{1}{N} ((x_1 - \bar{X})^2 + \cdots + (x_N - \bar{X})^2) = \frac{1}{N} \sum_{k=1}^N (x_k - \bar{X})^2.$$

Elle est donc égale à la moyenne des carrés des différences entre les observations  $x_i$  et leur moyenne  $\bar{X}$ .

L'écart-type, noté  $\sigma_X$ , est la racine carrée de la variance :

$$\sigma(X) = \sqrt{V(X)} = \sqrt{\frac{1}{N} \sum_{k=1}^N (x_k - \bar{X})^2}.$$

L'écart-type s'exprime dans les mêmes unités que  $X$  : si  $X$  désigne des longueurs en  $m$ , alors  $\sigma(X)$  s'exprime aussi en  $m$ .

Si la série statistique est donnée par un tableau statistique  $(x_i, n_i)$ , ce qui signifie que la valeur  $x_i$  est prise  $n_i$  fois, on peut directement calculer la variance par la formule :

$$V(X) = \frac{1}{n_1 + \cdots + n_N} \sum_{i=1}^N n_i (x_i - \bar{X})^2.$$

Voyons ce que signifie l'écart-type. Dans une classe de 25 élèves, à un devoir, on observe les notes suivantes :

- 5 notes à 8.
- 5 notes à 9.
- 5 notes à 10.
- 5 notes à 11.
- 5 notes à 12.

L'espérance (ou moyenne des notes!) est 10. La variance vaut

$$V = \frac{1}{25} (5 \times 2^2 + 5 \times 1^2 + 5 \times 0^2 + 5 \times 1^2 + 5 \times 2^2) = 2.$$

Elle est assez faible, les notes sont donc très centrées autour de la moyenne.

Dans une autre classe de 25 élèves, au même devoir, les élèves ont obtenu :

- 5 notes à 0.
- 5 notes à 5.
- 5 notes à 10.
- 5 notes à 15.
- 5 notes à 20.

L'espérance vaut toujours 10, mais pourtant il est clair que les deux classes sont très différentes! Calculons la variance : elle vaut

$$V = \frac{1}{25} (5 \times 10^2 + 5 \times 5^2 + 5 \times 0^2 + 5 \times 5^2 + 5 \times 10^2) = 50.$$

La variance est beaucoup plus grande que dans le premier cas, ce qui signifie que les notes sont très espacées!

# Corrélation des variables aléatoires

## Covariance de deux séries statistiques

La **covariance** de deux séries statistiques  $X = (X_i)_{i=1,\dots,N}$  et  $(Y_i)_{i=1,\dots,N}$  ayant le même nombre d'éléments est défini par

$$\text{Cov}(X, Y) = \sum_{i=1}^N \frac{(X_i - \bar{X})(Y_i - \bar{Y})}{N}.$$

Il s'agit donc de la moyenne des produits des écarts des valeurs à la moyenne de chaque série.

Le **coefficient de corrélation linéaire** de ces deux séries est défini par

$$\rho(X, Y) = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)} \sqrt{\text{Var}(Y)}}$$

où  $\text{Var}(X)$  et  $\text{Var}(Y)$  désignent respectivement les variances de  $X$  et de  $Y$ .

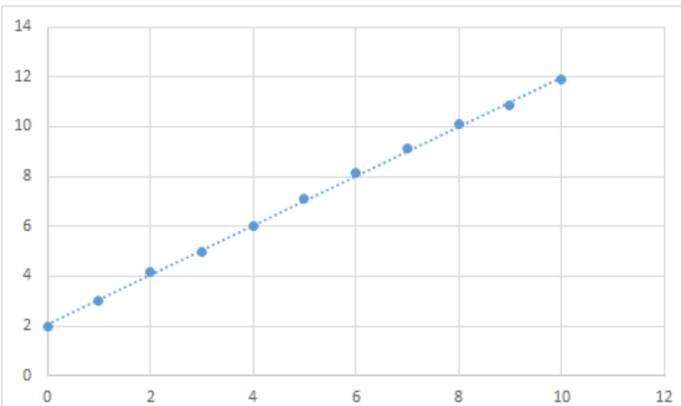
Le coefficient de relation linéaire quantifie la force du lien linéaire entre  $X$  et  $Y$ . Si  $r = 1$  ou  $r = -1$ , alors il existe deux réels  $a$  et  $b$  tels que  $Y = aX + b$  :  $Y$  dépend affinement de  $X$ . Si au contraire  $r$  est proche de 0, alors  $X$  et  $Y$  ne dépendent pas affinement l'un de l'autre.

Étudions trois exemples pour voir l'influence du coefficient de corrélation linéaire.

### Exemple 1 :

|   |      |      |      |      |      |      |      |      |       |       |       |
|---|------|------|------|------|------|------|------|------|-------|-------|-------|
| X | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8     | 9     | 10    |
| Y | 2,04 | 3,02 | 4,15 | 4,97 | 6,02 | 7,08 | 8,12 | 9,14 | 10,13 | 10,86 | 11,89 |

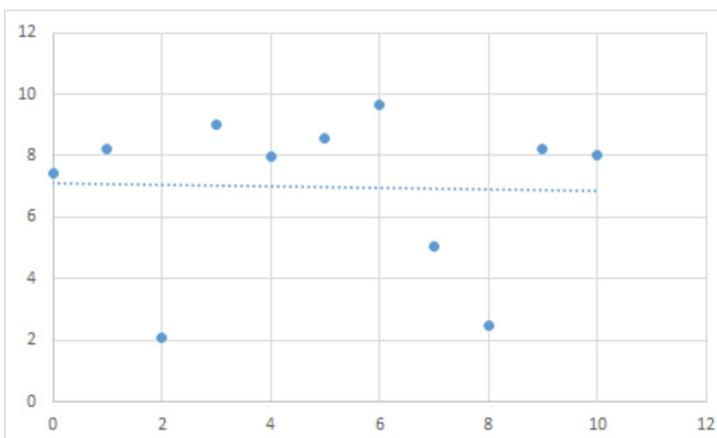
Le coefficient de corrélation linéaire est presque égal à 1. Les deux séries statistiques dépendent affinement l'une de l'autre, ce que confirme le nuage de points.



### Exemple 3 :

|   |      |      |      |      |      |      |      |      |      |      |      |
|---|------|------|------|------|------|------|------|------|------|------|------|
| X | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   |
| Y | 7,45 | 8,20 | 2,09 | 9,03 | 7,99 | 8,58 | 9,63 | 5,06 | 2,49 | 8,22 | 8,01 |

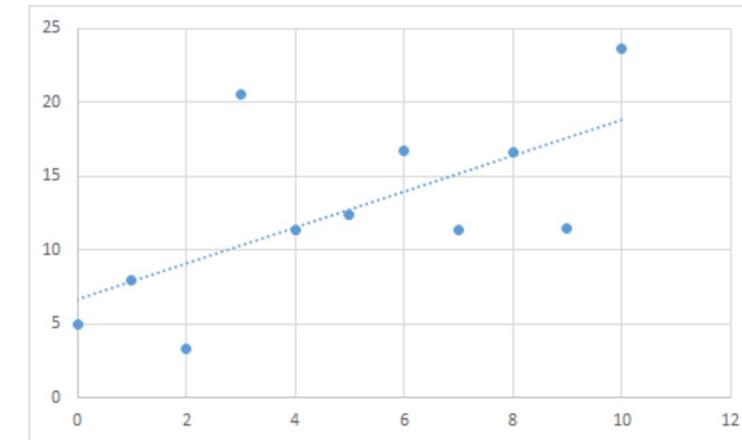
Le coefficient de corrélation linéaire est environ égal à  $-0,02$ , il est donc très faible. Les deux caractères étudiés ne semblent pas corrélés. Voici le nuage de points associé.



### Exemple 2 :

|   |      |      |      |       |       |       |       |       |       |       |       |
|---|------|------|------|-------|-------|-------|-------|-------|-------|-------|-------|
| X | 0    | 1    | 2    | 3     | 4     | 5     | 6     | 7     | 8     | 9     | 10    |
| Y | 5,01 | 7,93 | 3,27 | 20,50 | 11,33 | 12,41 | 16,74 | 11,39 | 16,61 | 11,48 | 23,59 |

Le coefficient de corrélation linéaire est environ égal à  $0,65$ . Il y a une certaine corrélation entre les deux séries statistiques mais elles ne dépendent pas affinement l'une de l'autre. Voici le nuage de points associé.



Si  $X$  et  $Y$  admettent une covariance et des variances non nulles, alors leur **coefficient de corrélation linéaire** est défini par :

$$\rho(X, Y) = \frac{\text{Cov}(X, Y)}{\sqrt{V(X)V(Y)}} = \frac{\text{Cov}(X, Y)}{\sigma(X)\sigma(Y)} \in [-1, 1].$$

Les variables aléatoires  $X$  et  $Y$  sont dites **non corrélées** si  $\text{Cov}(X, Y) = 0$ . Elles sont dites **corrélées** dans le cas contraire.

**Interprétation :** Le coefficient de corrélation linéaire mesure la dépendance affine de  $X$  et  $Y$ . Ainsi, si  $\rho(X, Y) = 1$ , il existe des réels  $a$  et  $b$  tels que  $Y = aX + b$ . Inversement, si  $X$  et  $Y$  sont indépendantes,  $\text{Cov}(X, Y) = 0$ .

Si  $(X_i)_{i=1,\dots,n}$  est une suite finie de variables aléatoires, la **matrice des variances/covariances** des  $(X_i)$  est la matrice carrée dont le coefficient en  $(i, j)$  est donné par :

$$c_{i,j} = \text{Cov}(X_i, X_j).$$

Par exemple, la matrice de covariance du couple  $(X, Y)$  est :

$$\begin{pmatrix} V(X) & \text{Cov}(X, Y) \\ \text{Cov}(X, Y) & V(Y) \end{pmatrix}.$$

Une matrice de covariance est toujours symétrique.

