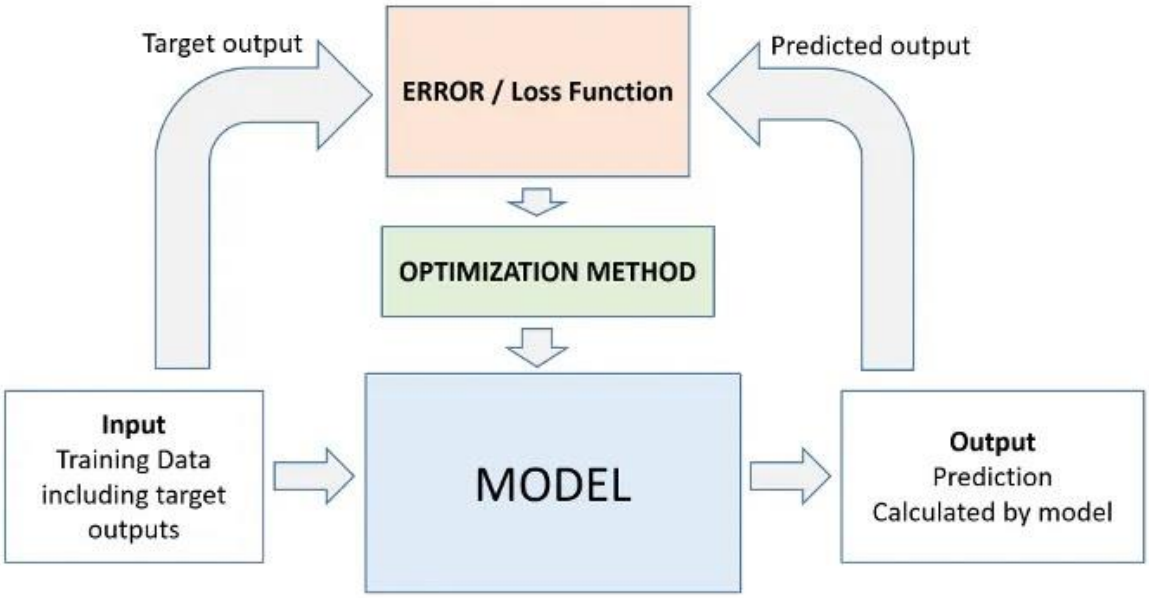
# Rapport du Projet académique





## Réalisé par :

SENNAH Marouane

FENNICH Sofia

HAFSI Siham

KHALIL Chaimaa

## Encadré par :

Pr. Abdelghani GHAZDALI

Pr. Hamza KHALFI

# *Sommaire :*

# Remerciements

Après avoir rendu grâce à Dieu le tout puissant nous tenons à remercier vivement nos chers professeurs Pr. Abdelghani GHAZDALI et Pr. Hamza KHALFI pour leur parrainage, leur assistance leurs recommandations, et conseils qui ont aidé à réaliser ce projet.

# I. Introduction

Le Machine Learning est une branche de l'intelligence artificielle qui permet aux ordinateurs d'apprendre à partir de données sans être explicitement programmés.

Notre projet académique consiste à utiliser le Machine Learning pour développer un modèle de prédiction des prix des ordinateurs en utilisant **la régression linéaire multiple**.

Pour atteindre cet objectif, deux algorithmes d'**optimisation** différents ont été comparés :

La descente de gradient (**GD**) et la descente de gradient stochastique (**SGD**).

L'optimisation avec ces algorithmes permettra de trouver les coefficients optimaux du modèle de la régression linéaire multiple.

Ces coefficients représentent les poids associés à chaque caractéristique de l'ordinateur qui permettent de prédire son prix.

# II Concepts clé

## Machine Learning

Le Machine Learning peut être défini comme étant une technologie d'intelligence artificielle permettant aux machines d'apprendre sans avoir été au préalablement programmées spécifiquement à cet effet.

Mais comment apprendre ?

Pour donner à un ordinateur la capacité d'apprendre, on utilise des méthodes d'apprentissage qui sont fortement inspirées de la façon dont nous, les êtres humains, apprenons à faire des choses. Parmi ces méthodes, on compte :

- L'apprentissage supervisé (Supervised Learning)
- L'apprentissage non supervisé (Unsupervised Learning)
- L'apprentissage par renforcement (Reinforcement Learning)

Dans le cas du Machine Learning supervisé, un programmeur se charge d'attribuer une étiquette à chaque « input » d'entraînement injecté au système.

Dans le cas du Machine Learning non supervisé, en revanche, le modèle reçoit uniquement des données d'entrée sans étiquette explicite. Il doit explorer les données pour trouver la structure cachée ou les relations entre elles.

Enfin, l'apprentissage par renforcement utilise un système de récompenses et de punition pour guider l'ordinateur à résoudre un problème par lui-même. L'intervention humaine est limitée au strict minimum, à savoir les modifications d'environnement et du système de récompenses.

Ces différentes techniques d'apprentissage partagent une caractéristique commune, à savoir qu'elles ont recours à des algorithmes pour apprendre à partir de données, sans qu'il soit nécessaire de programmer explicitement une tâche spécifique. Cependant, ces méthodes présentent des différences quant à la façon dont les données sont présentées à l'algorithme, ainsi que la manière dont l'algorithme apprend à partir des données.

Pour maîtriser les bases du Machine Learning, il faut absolument comprendre et connaitre ces 4 notions essentielles :

1. **Le Dataset**

   En Machine Learning, tout démarre d'un Dataset qui contient les données.

2. **Le modèle et ses paramètres**

   A partir de ce Dataset, on crée un modèle, qui n'est autre qu'une fonction mathématique. Les coefficients de cette fonction sont les paramètres du modèle.

3. **La Fonction Coût**

   Lorsqu'on teste notre modèle sur le Dataset, celui-ci nous donne des erreurs. L'ensemble de ces erreurs, c'est ce qu'on appelle la Fonction Coût.

4. **L'Algorithme d'apprentissage**

   L'idée centrale du Machine Learning, c'est de laisser la machine trouver quels sont les paramètres de notre modèle qui minimisent la Fonction Coût

## Régression linéaire multiple

En machine Learning, la régression linéaire multiple est utilisée comme méthode d'apprentissage supervisé pour prédire des valeurs continues à partir d'un ensemble de données d'entraînement. Elle est couramment utilisée pour la prédiction de prix, de quantités, de scores, etc. Elle est également utilisée dans des domaines tels que la finance, l'économie, la sociologie, la biologie, la physique, l'ingénierie, etc.

Le processus d'apprentissage dans la régression linéaire multiple consiste à trouver les coefficients optimaux de l'équation linéaire en minimisant l'erreur quadratique moyenne entre les valeurs prédites et les valeurs réelles dans l'ensemble de données d'entraînement.

Une fois que le modèle a été entraîné sur l'ensemble de données d'entraînement, il peut être utilisé pour faire des prédictions sur de nouvelles données.

Etapes pour développer un programme de Régression Linéaire multiple :

## 1. Récolter des données ($X$, $y$)

En Machine Learning, on compile ces exemples ($x$, $y$) dans un tableau que l'on appelle Dataset :

- La variable $y$ porte le nom de target (la cible). C'est la valeur que l'on cherche à prédire.
- On regroupe les features ($x1$, $x2$, … ) dans notre Dataset dans une matrice $X$.

## 2. Donner à la machine un modèle linéaire

Fonction mathématique qui associe $X$ à $y$, telle que $f(X) = y$. Un bon modèle doit être une bonne généralisation, c'est-à-dire qu'il doit fournir de petites erreurs entre $f(x)$ et $y$

On note $\theta$ le vecteur qui contient les paramètres de notre modèle. Pour une régression

Linéaire multiple, la formulation matricielle de notre modèle devient : $F(X) = X.\theta$

### 3. Créer la Fonction Coût

La Fonction Coût $\boldsymbol{J(\theta)}$ mesure l'ensemble des erreurs entre le modèle et le Dataset.

$$J(\theta) = \frac{1}{2m} \sum (F(X) - y)^2$$

Chaque prédiction s'accompagne d'une erreur, on a donc $\boldsymbol{m}$ erreurs.

On définit la Fonction Coût $\boldsymbol{J(\theta)}$ comme étant la moyenne de toutes les erreurs.

### 4. Calculer le gradient et utiliser l'algorithme de Gradient Descent

*Répeter en boucle*:

$$\theta = \theta - \alpha \times \frac{\partial J(\theta)}{\partial \theta}$$

$$\textbf{Gradient}: \frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} X^T . (F(X) - Y)$$

Le *Learning rate* $\alpha$ prend le nom d'hyper-paramètre de par son influence sur la performance finale du modèle (s'il est trop grand où trop petit, la fonction le Gradient Descent ne converge pas).

Le **gradient Descent** est un algorithme de minimisation de la Fonction Coût.

Il existe beaucoup de variante de cet algorithme :

- Mini Batch Gradient Descent
- Stochastic Gradient Descent
- Momentum
- RMSProp
- Adam

On peut résumer les étapes de la régression linéaire multiple de la manière suivante :

$$\boldsymbol{Dataset}: (X, y)\ avec\ X, y \in\ \mathbb{R}^{m \times n}$$

$$\boldsymbol{Modèle}: F(X) = X.\theta$$

$$\boldsymbol{Fonction\ Coût}: J(\theta) = \frac{1}{2m} \sum (F(X) - y\ )^2$$

$$\boldsymbol{Gradient}: \frac{\partial\ J(\theta)}{\partial\ \theta} = \frac{1}{m}\ X^T.(\ F(X) - y)$$

$$\boldsymbol{Gradient\ Descent}: \theta =\ \theta -\ \alpha \times \frac{\partial\ J(\theta)}{\partial\ \theta}$$

## *Optimisation :*

Dans la régression linéaire multiple, l'optimisation joue un rôle crucial dans la sélection des paramètres optimaux pour le modèle. L'objectif de l'optimisation est de minimiser l'erreur de prédiction du modèle en trouvant les valeurs optimales pour les coefficients de régression.

Dans le domaine de l'apprentissage automatique, la descente de gradient (Gradient Descent, GD) et la descente de gradient stochastique (Stochastic Gradient Descent, SGD) sont deux algorithmes d'optimisation couramment utilisés pour trouver les paramètres optimaux d'un modèle.

La descente de gradient est une méthode d'optimisation qui ajuste progressivement les paramètres du modèle en utilisant la dérivée de la fonction de coût par rapport à chaque paramètre. L'algorithme commence avec une estimation initiale des paramètres et se déplace progressivement dans la direction de la pente négative de la fonction de coût pour minimiser l'erreur de prédiction. Le pas de mise à jour des paramètres est déterminé par un paramètre appelé taux d'apprentissage (learning rate), qui contrôle la taille des pas de mise à jour.

La descente de gradient stochastique (SGD) est une variante de la descente de gradient qui calcule la dérivée de la fonction de coût pour un échantillon aléatoire de données à la fois, plutôt que pour l'ensemble des données. L'algorithme effectue donc des mises à jour de paramètres plus fréquentes et peut converger plus rapidement que la descente de gradient classique. Cependant, cette méthode peut être plus instable en raison de la variabilité introduite par l'échantillonnage aléatoire.

Le choix entre GD et SGD dépend du contexte et des caractéristiques des données. La descente de gradient est plus appropriée lorsque l'ensemble des données est relativement petit et que le calcul de la dérivée de la fonction de coût est réalisable en un temps raisonnable. En revanche, la descente de gradient stochastique est souvent préférable pour les ensembles de données volumineux, car elle peut être calculée plus rapidement sur des échantillons de données plus petits.

En conclusion, GD et SGD sont deux algorithmes d'optimisation couramment utilisés dans l'apprentissage automatique pour trouver les paramètres optimaux d'un modèle. Ils ont chacun leurs avantages et leurs inconvénients, et le choix entre les deux dépend du contexte et des caractéristiques des données.

Pour plus de détails sur le fonctionnement de chaque algorithme veuillez consulter la section « cours » dans la plateforme qu’on a créé.

# III. Conception

Le projet utilise la régression linéaire multiple pour prédire le prix d'un ordinateur à partir de ses caractéristiques, en comparant l'efficacité de deux algorithmes d'optimisation : la descente de gradient et la descente de gradient stochastique. L'objectif est de prédire le prix de l'ordinateur en se basant sur ses caractéristiques.

## Dataset :

Le dataset "laptop-datacsv" est un ensemble de données disponible sur Kaggle, qui contient des informations sur des ordinateurs portables de différentes marques, modèles et configurations. Les données ont été collectées à partir de différentes sources en ligne, telles que des sites d'e-commerce et des sites de fabricants d'ordinateurs portables.

Le dataset comprend 14 colonnes :

1. Company : le nom de la marque de l'ordinateur portable

2. Product : le nom du modèle de l'ordinateur portable

3. TypeName : le type de l'ordinateur portable, comme par exemple "gaming", "ultrabook", "netbook", etc.

4. Inches : la taille de l'écran en pouces

5. ScreenResolution : la résolution de l'écran

6. CPU : le nom du processeur

7. RAM : la quantité de mémoire vive (RAM) en gigaoctets (Go)

8. Memory : la capacité de stockage en gigaoctets (Go)

9. GPU : le nom de la carte graphique

10. OpSys : le système d'exploitation préinstallé sur l'ordinateur portable

11. Weight : le poids de l'ordinateur portable en kilogrammes

12. Price : le prix de l'ordinateur portable en euros

13. dtype : le type de stockage, tel que "SSD" ou "HDD"

14. Speed : la vitesse du processeur en gigahertz (GHz)

Ces données peuvent être utilisées pour différents types d'analyses, telles que la comparaison des caractéristiques des ordinateurs portables de différentes marques et modèles, l'analyse de la relation entre le prix et les caractéristiques de l'ordinateur portable, etc.

| | Unnamed: 0 | Company | TypeName | Inches | ScreenResolution | Cpu | Ram | Memory | Gpu | OpSys | Weight | Price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Apple | Ultrabook | 13.3 | IPS Panel Retina Display 2560x1600 | Intel Core i5 2.3GHz | 8GB | 128GB SSD | Intel Iris Plus Graphics 640 | macOS | 1.37kg | 71378.6832 |
| 1 | 1 | Apple | Ultrabook | 13.3 | 1440x900 | Intel Core i5 1.8GHz | 8GB | 128GB Flash Storage | Intel HD Graphics 6000 | macOS | 1.34kg | 47895.5232 |
| 2 | 2 | HP | Notebook | 15.6 | Full HD 1920x1080 | Intel Core i5 7200U 2.5GHz | 8GB | 256GB SSD | Intel HD Graphics 620 | No OS | 1.86kg | 30636.0000 |
| 3 | 3 | Apple | Ultrabook | 15.4 | IPS Panel Retina Display 2880x1800 | Intel Core i7 2.7GHz | 16GB | 512GB SSD | AMD Radeon Pro 455 | macOS | 1.83kg | 135195.3360 |
| 4 | 4 | Apple | Ultrabook | 13.3 | IPS Panel Retina Display 2560x1600 | Intel Core i5 3.1GHz | 8GB | 256GB SSD | Intel Iris Plus Graphics 650 | macOS | 1.37kg | 96095.8080 |

Après le processus de nettoyage nous allons travailler avec ces champs :

| | Ram | Weight | Touchscreen | ppi | HDD | SSD |
|---|---|---|---|---|---|---|
| 0 | 8 | 1.37 | 0 | 226.983005 | 0 | 128 |
| 1 | 8 | 1.34 | 0 | 127.677940 | 0 | 0 |
| 2 | 8 | 1.86 | 0 | 141.211998 | 0 | 256 |
| 3 | 16 | 1.83 | 0 | 220.534624 | 0 | 512 |
| 4 | 8 | 1.37 | 0 | 226.983005 | 0 | 256 |
| ... | ... | ... | ... | ... | ... | ... |
| 1297 | 4 | 1.80 | 1 | 157.350512 | 0 | 128 |
| 1298 | 16 | 1.30 | 1 | 276.053530 | 0 | 512 |
| 1299 | 2 | 1.50 | 0 | 111.935204 | 0 | 0 |
| 1300 | 6 | 2.19 | 0 | 100.454670 | 1000 | 0 |
| 1301 | 4 | 2.20 | 0 | 100.454670 | 500 | 0 |

1302 rows × 6 columns

## Implémentation du modèle :

```
def model(x, w, b):
    f_wb = np.dot(x,w) + b
    return f_wb
```

## On a utilisé l'erreur quadratique :

```
def cost(x, y, w, b):
    m = x.shape[0]
    total_cost = 0.0
    for i in range(m):
        f_wb_i = np.dot(x[i], w) + b
        total_cost += (f_wb_i - y[i])**2
    cost = total_cost / (2*m)
    return cost
```

## La descente du gradient classique :

```
def compute_gradient(X, y, w, b):
    m,n = X.shape
    dj_dw = np.zeros((n,))
    dj_db = 0.

    for i in range(m):
        err = (np.dot(X[i], w) + b) - y[i]
        for j in range(n):
            dj_dw[j] = dj_dw[j] + err * X[i, j]
        dj_db = dj_db + err
    dj_dw = dj_dw / m
    dj_db = dj_db / m

    return dj_db, dj_dw

# Descente de gradient
alpha = 0.000000005
tolerance = 1e-6
max_iter = 1000000
costs = []
iteration = 0
while iteration < max_iter:
    dj_db, dj_dw = compute_gradient(X, y, w, b)
    w = w - alpha * dj_dw
    b = b - alpha * dj_db
    cost_i = cost(X, y, w, b)
    costs.append(cost_i)
    if len(costs) > 1 and np.abs(costs[-1] - costs[-2]) < tolerance:
        break
    iteration += 1
```

Ce code implémente la descente de gradient pour un modèle de régression linéaire multiple. La fonction compute_gradient calcule le gradient de la fonction de coût pour tous les échantillons d'entraînement, qui sont stockés dans la matrice X et les étiquettes

cibles y. Les poids du modèle sont stockés dans le vecteur w et le biais est stocké dans le scalaire b.

La descente de gradient est effectuée dans une boucle while où, à chaque itération, les poids et le biais sont mis à jour en fonction de leur gradient calculé dans la fonction compute_gradient. L'hyperparamètre alpha définit la taille du pas de mise à jour des poids et est communément appelé le taux d'apprentissage. Les hyperparamètres tolerance et max_iter sont utilisés pour définir les critères d'arrêt de la descente de gradient. La descente de gradient continue jusqu'à ce que le nombre d'itérations maximales max_iter soit atteint ou que la différence de coût entre deux itérations successives soit inférieure à la tolérance tolerance.

Le coût à chaque itération est stocké dans la liste

costs, ce qui permet de visualiser la convergence de la fonction de coût au fil du temps. La boucle while s'arrête lorsque la convergence est atteinte ou que le nombre maximum d'itérations est atteint.

En somme, ce code effectue une descente de gradient pour apprendre les poids et le biais d'un modèle de régression linéaire multiple, en utilisant les données

d'entraînement pour minimiser la fonction de coût du modèle.

La descente du gradient Stochastique :

```python
def compute_gradient_stochastique(X, y, w, b, i):
    dj_dw = np.zeros((len(w),))
    dj_db = 0.
    err = (np.dot(X[i], w) + b) - y[i]
    for j in range(len(w)):
        dj_dw[j] = err * X[i, j]
    dj_db = err

    return dj_db, dj_dw

# Descente de gradient stochastique
alpha =  0.000000005
tolerance = 1e-6
max_iter = 10000
costs = []
iteration = 0
while iteration < max_iter:
    shuffled_indices = list(range(len(y)))
    random.shuffle(shuffled_indices)
    for i in shuffled_indices:
        dj_db, dj_dw = compute_gradient_stochastique(X, y, w, b, i)
        w = w - alpha * dj_dw
        b = b - alpha * dj_db
    cost_i = cost(X, y, w, b)
    costs.append(cost_i)
    if len(costs) > 1 and np.abs(costs[-1] - costs[-2]) < tolerance:
        break
    iteration += 1
```

Ce deuxième code implémente la descente de gradient stochastique (SGD), une variante de la descente de gradient classique (GD) qui utilise un échantillon aléatoire à chaque itération pour calculer le gradient et mettre à jour les paramètres du modèle. Contrairement à GD, qui calcule le gradient en utilisant l'ensemble de données entier à chaque itération, SGD ne considère qu'un seul exemple d'entraînement à chaque itération, ce qui peut être plus rapide pour les ensembles de données volumineux.

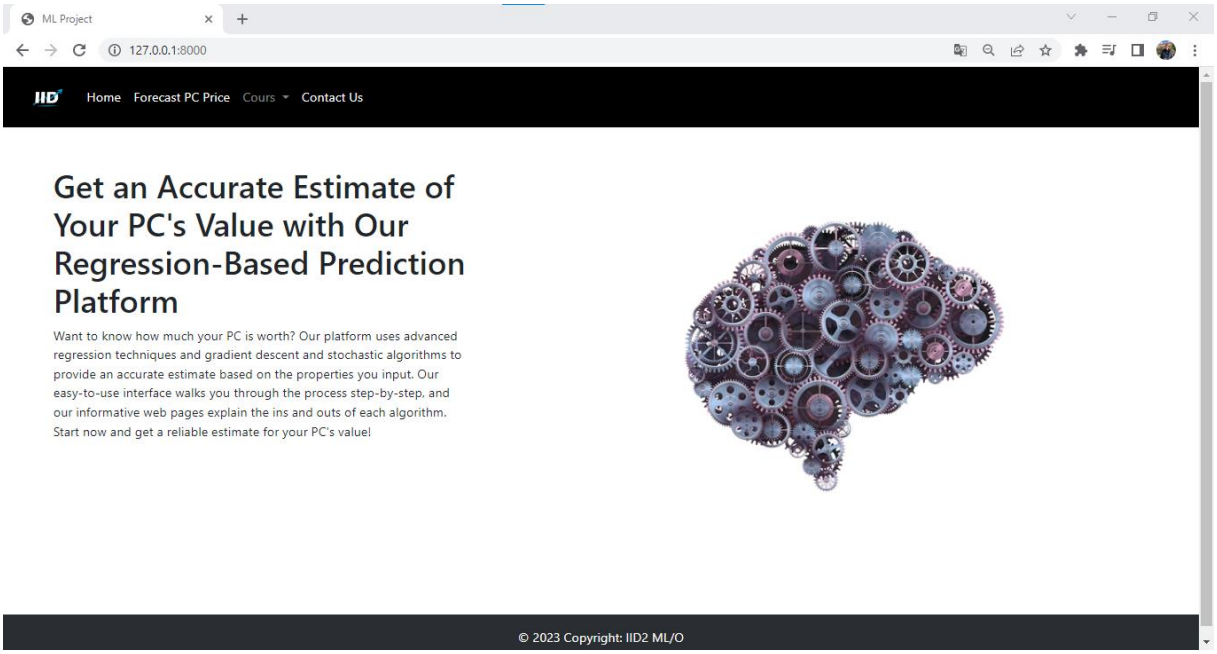
La fonction compute_gradient_stochastique calcule le gradient pour un exemple d'entraînement donné. Les paramètres X et y sont les données d'entraînement, w et b sont les poids et le biais du modèle, et i est l'indice de l'exemple d'entraînement. La fonction utilise cet exemple pour calculer le gradient et retourne les valeurs correspondantes pour le biais et les poids.

Le reste du code est similaire à la descente de gradient classique, mais à chaque itération, il mélange les indices des exemples d'entraînement (shuffled_indices) et itère sur chacun d'entre eux pour mettre à jour les poids et le biais à l'aide de la fonction compute_gradient_stochastique. Ensuite, il calcule le coût total en utilisant la fonction cost, stocke la valeur dans la liste costs et vérifie si la différence entre le coût actuel et le précédent est inférieure à la tolérance fixée. Si c'est le cas, l'algorithme s'arrête.

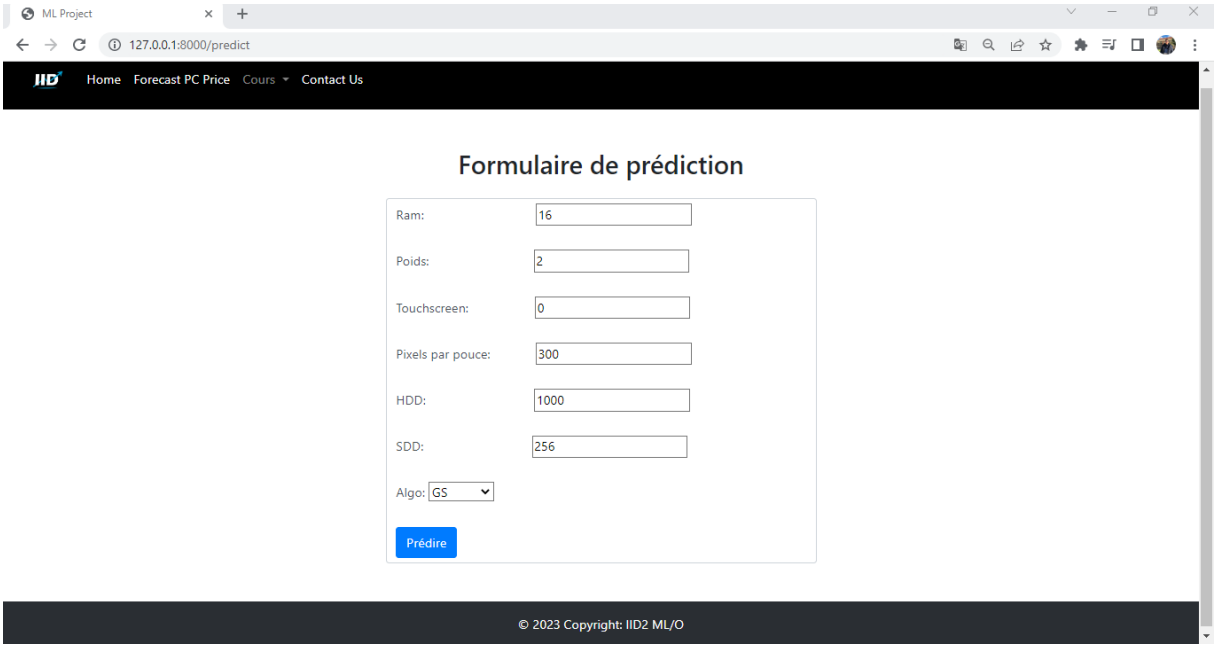
On a exploité les résultats de chaque algorithme pour créer un formulaire de prédiction des prix d’achat des ordinateurs.

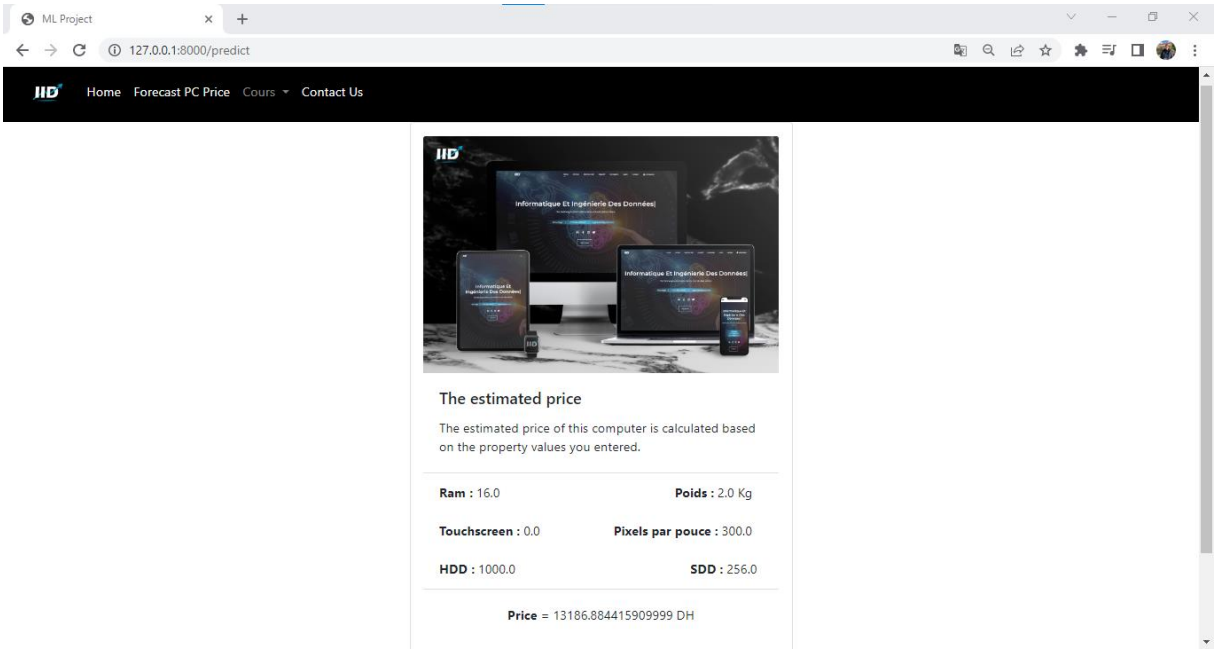
# IV. Plateforme

Dans la plateforme qu'on a créée, on donne aux utilisateurs la possibilité de prédire le prix d'un ordinateur à partir des caractéristiques voulues, Découvrir la partie théorie de chaque algorithme utilisé pour la prédiction :

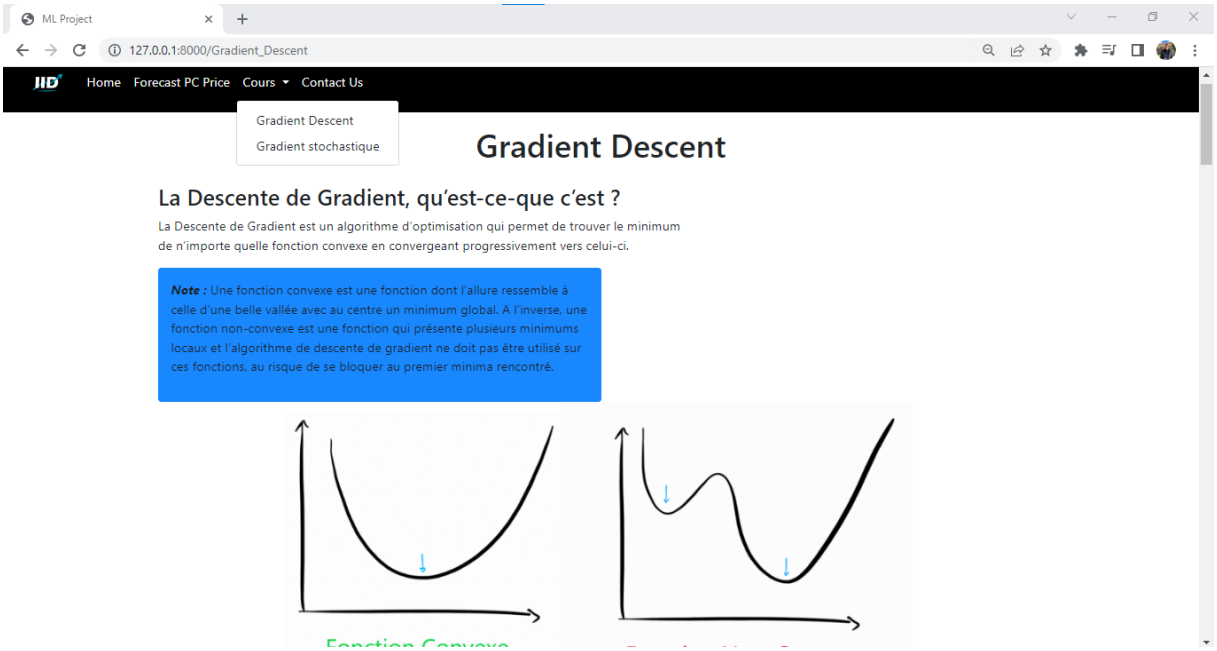


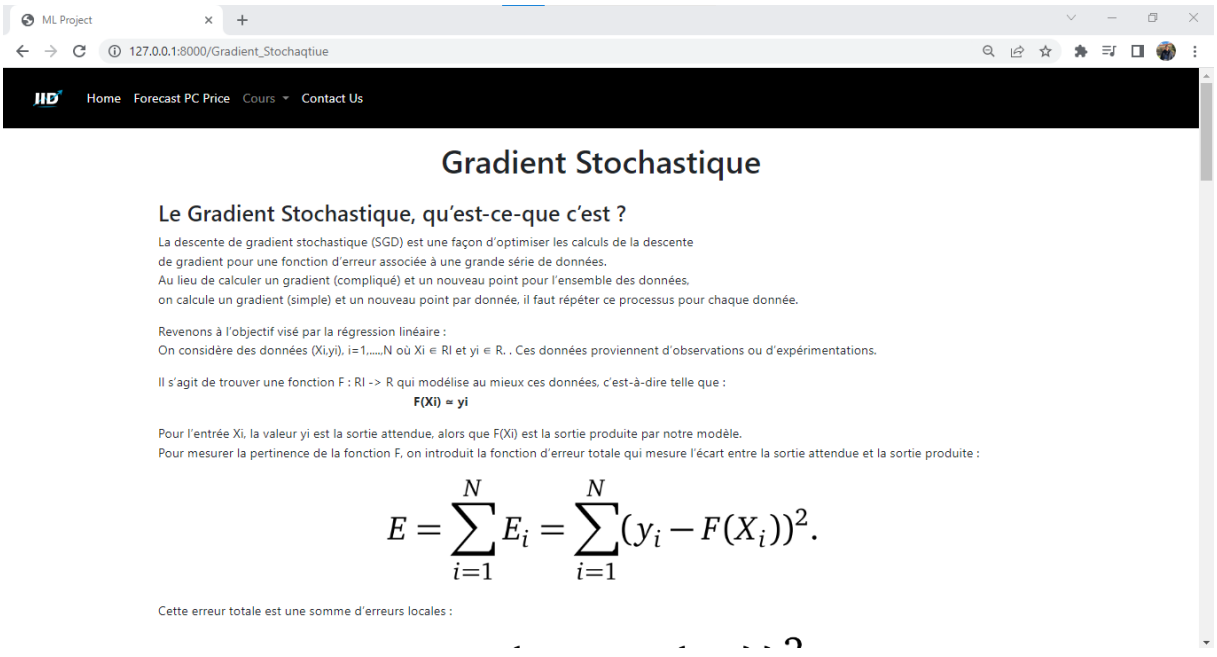
# Formulaire de prédiction :



S’il veut consulter la partie théorique de chaque algorithme :



# V. Conclusion

En conclusion, l'utilisation de différents algorithmes d'optimisation peut avoir un impact significatif sur la performance de la régression linéaire multiple pour la prédiction de variables continues telles que les prix des ordinateurs.

Dans ce cas, la descente de gradient stochastique a donné des résultats comparables à ceux de la descente de gradient standard, mais avec une convergence plus rapide.

Cependant, il est important de noter que le choix de l'algorithme d'optimisation dépendra de la nature des données et de la précision souhaitée, et qu'il est toujours recommandé d'expérimenter avec différentes méthodes pour déterminer celle qui convient le mieux à un cas d'utilisation spécifique.

FIN