جامعة ابن طفيل
Ibn Tofaïl University
Faculté des Sciences

Rapport de Travaux Pratiques (TP)

Application de la Régression Linéaire et Logistique

Module : Apprentissage Automatique

Réalisé par : ACHENAN Ali

Présenté à : RANIA MAJDOUBI

Année Universitaire ; 2024-2025

Table des matières

Liste des figures

Introduction

Ce Travail Pratique (TP) a pour objectif d'appliquer les concepts fondamentaux de la régression linéaire et de la régression logistique pour modéliser des relations entre des variables. Conformément aux consignes, un dataset personnalisé a été créé pour l'exercice. Ce rapport détaille la démarche suivie, depuis la conception et la génération du dataset jusqu'à la construction, l'entraînement, l'évaluation et la visualisation des modèles de régression dans un notebook Python.

1. Création du Dataset Personnalisé

1.1. Sujet et Caractéristiques du Dataset

Pour ce TP, le sujet choisi est l'étude des **facteurs influençant la performance des étudiants**. Un dataset a été conceptualisé autour de ce thème, comprenant les variables suivantes :

- ✓ `Hours_Studied` : Nombre d'heures étudiées par semaine (Numérique)
- ✓ `Previous_Score` : Note obtenue lors d'évaluations précédentes (Numérique)
- ✓ `Attendance_Rate` : Taux de présence aux cours (Numérique)
- ✓ `Final_Exam_Score` : Note obtenue à l'examen final (Numérique - Cible pour Régression Linéaire)
- ✓ `Passed_Course` : Indique si l'étudiant a réussi le cours (1) ou échoué (0) (Binaire - Cible pour Régression Logistique)
- ✓ Ces variables ont été sélectionnées pour permettre l'application des deux types de régression : prédire un score continu (`Final_Exam_Score`) et prédire un résultat binaire (`Passed_Course`).

1.2. Méthodologie Initiale : Génération via Mockaroo

La génération du dataset a débuté avec l'outil en ligne Mockaroo, apprécié pour sa capacité à générer des données synthétiques avec des types et des plages spécifiés. L'idée initiale était d'utiliser la fonctionnalité "Formula" de Mockaroo pour générer les colonnes cibles (`Final_Exam_Score` et `Passed_Course`) en appliquant une formule mathématique (simulant une relation linéaire plus du bruit) qui dépendrait des colonnes prédicteurs déjà définies.

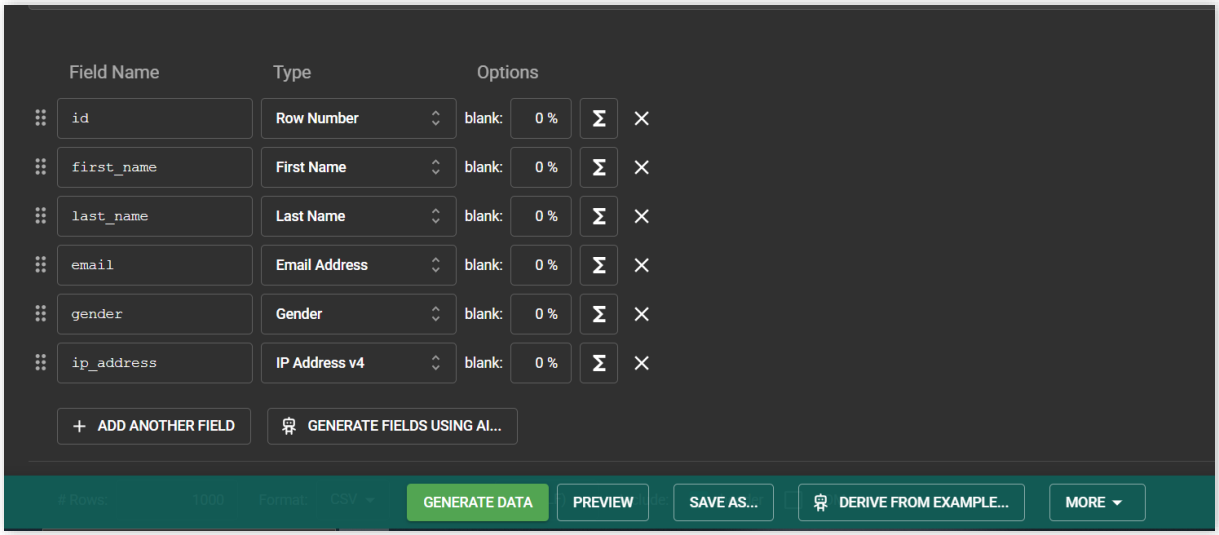


Figure 1 : Configuration initiale du dataset dans Mockaroo avec les colonnes définies.

1.3. Défis Rencontrés et Solution Adoptée (Script Python)

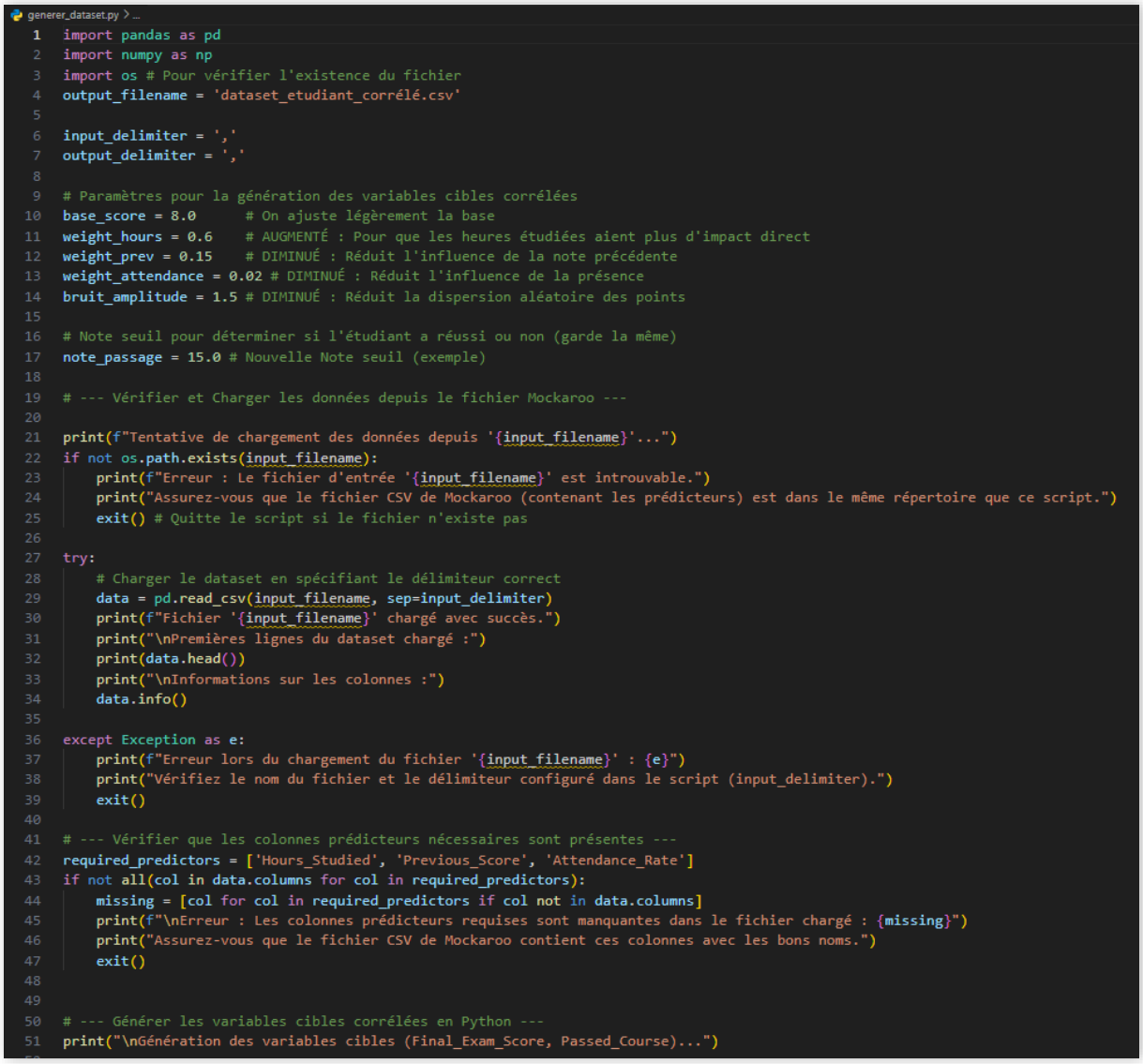
La génération des variables cibles avec des formules dans Mockaroo s'est avérée problématique. Des erreurs de syntaxe ou d'interprétation des fonctions Ruby (comme les méthodes `.max`, `.min` ou la structure `if/else`) par le moteur de formule de Mockaroo ont empêché la génération correcte de `Final_Exam_Score` et `Passed_Course`, affichant des messages d'erreur dans les données générées.

Pour contourner ce problème et garantir la création d'un dataset avec la corrélation souhaitée (en particulier une relation quasi-linéaire pour `Final_Exam_Score` et une distribution binaire pour `Passed_Course`), une approche en deux étapes a été adoptée :

a) Générer uniquement les colonnes prédicteurs (Hours_Studied, Previous_Score, Attendance_Rate) dans Mockaroo.

b) Utiliser un script Python séparé pour lire ce fichier CSV, calculer les colonnes Final_Exam_Score et Passed_Course en appliquant la formule de corrélation (somme pondérée des prédicteurs + bruit aléatoire) et les règles de seuil/bornes, puis exporter le dataset complet dans un nouveau fichier CSV.

Ce script Python a permis d'avoir un contrôle total sur la génération des variables cibles et d'assurer une relation logique entre les prédicteurs et les résultats.



```python
import pandas as pd
import numpy as np
import os # Pour vérifier l'existence du fichier
output_filename = 'dataset_etudiant_corrélé.csv'

input_delimiter = ','
output_delimiter = ','

# Paramètres pour la génération des variables cibles corrélées
base_score = 8.0      # On ajuste légèrement la base
weight_hours = 0.6    # AUGMENTÉ : Pour que les heures étudiées aient plus d'impact direct
weight_prev = 0.15    # DIMINUÉ : Réduit l'influence de la note précédente
weight_attendance = 0.02 # DIMINUÉ : Réduit l'influence de la présence
bruit_amplitude = 1.5 # DIMINUÉ : Réduit la dispersion aléatoire des points

# Note seuil pour déterminer si l'étudiant a réussi ou non (garde la même)
note_passage = 15.0 # Nouvelle Note seuil (exemple)

# --- Vérifier et Charger les données depuis le fichier Mockaroo ---

print(f"Tentative de chargement des données depuis '{input_filename}'...")
if not os.path.exists(input_filename):
    print(f"Erreur : Le fichier d'entrée '{input_filename}' est introuvable.")
    print("Assurez-vous que le fichier CSV de Mockaroo (contenant les prédicteurs) est dans le même répertoire que ce script.")
    exit() # Quitte le script si le fichier n'existe pas

try:
    # Charger le dataset en spécifiant le délimiteur correct
    data = pd.read_csv(input_filename, sep=input_delimiter)
    print(f"Fichier '{input_filename}' chargé avec succès.")
    print("\nPremières lignes du dataset chargé :")
    print(data.head())
    print("\nInformations sur les colonnes :")
    data.info()

except Exception as e:
    print(f"Erreur lors du chargement du fichier '{input_filename}' : {e}")
    print("Vérifiez le nom du fichier et le délimiteur configuré dans le script (input_delimiter).")
    exit()

# --- Vérifier que les colonnes prédicteurs nécessaires sont présentes ---
required_predictors = ['Hours_Studied', 'Previous_Score', 'Attendance_Rate']
if not all(col in data.columns for col in required_predictors):
    missing = [col for col in required_predictors if col not in data.columns]
    print(f"\nErreur : Les colonnes prédicteurs requises sont manquantes dans le fichier chargé : {missing}")
    print("Assurez-vous que le fichier CSV de Mockaroo contient ces colonnes avec les bons noms.")
    exit()


# --- Générer les variables cibles corrélées en Python ---
print("\nGénération des variables cibles (Final_Exam_Score, Passed_Course)...")
```

Figure 2: Script Python utilisé pour ajouter les variables cibles corrélées au dataset initial de prédicteurs

1	Hours_Studied	Previous_Score	Attendance_Rate	Final_Exam_Score	Passed_Course
2	24	11	98	20.0	1
3	19	18	11	20.0	1
4	22	13	57	20.0	1
5	7	9	11	13.949406732501776	0
6	8	10	54	15.199609774942976	1
7	28	10	16	20.0	1
8	22	14	19	20.0	1
9	20	10	46	20.0	1

Figure 3: Aperçu du dataset final chargé dans un éditeur de texte ou affiché dans le notebook, montrant les 5 colonnes prêtes pour l'analyse.

https://github.com/AliAch04/Linear-regression-at-logistic-regression-exercise

2. Régression Linéaire

2.1. Objectif

L'objectif de cette partie est de construire un modèle de régression linéaire simple pour prédire la **Note Finale (**`Final_Exam_Score`**)** en fonction d'une seule variable explicative : les **Heures Étudiées (**`Hours_Studied`**)**.

2.2. Démarche dans le Notebook

Le TP a été réalisé dans un environnement de notebook Python (Jupyter/VS Code) en suivant les étapes types d'un projet de modélisation.

2.2.1. Chargement et Exploration des Données

Le dataset final (`dataset_etudiant_corrélé.csv`) a été chargé dans un DataFrame pandas. Une première exploration a été réalisée pour comprendre la structure des données. Il a été important de spécifier le bon délimiteur (virgule) lors du chargement.

Charge ton fichier CSV généré par Mockaroo dans un DataFrame pandas. Assure-toi que le fichier se trouve dans le même répertoire que ton notebook, ou spécifie le chemin complet.

```
data = pd.read_csv('dataset_etudiant.csv')
```

Figure 4 : Chargement du dataset corrigé dans le notebook.

```
[65]: # Afficher les informations sur le dataset
      print("\nInformations sur le dataset :")
      data.info()

Informations sur le dataset :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Hours_Studied     1000 non-null   int64
 1   Previous_Score    1000 non-null   int64
 2   Attendance_Rate   1000 non-null   int64
 3   Final_Exam_Score  1000 non-null   float64
 4   Passed_Course     1000 non-null   int64
dtypes: float64(1), int64(4)
memory usage: 39.2 KB
```

Figure 5 : Exploration initiale des données chargées.

2.2.2. Préparation des Données (Variable Unique)

Pour la régression linéaire simple, la variable explicative X (`Hours_Studied`) et la variable cible y (`Final_Exam_Score`) ont été sélectionnées.

Ici, on choisit explicitement une seule variable explicative (`Hours_Studied`) et la variable cible (`Final_Exam_Score`).

```
[68]: # Sélectionner les variables explicatives (X) et la variable cible (y)
      X_lin = data[['Hours_Studied']]
      y_lin = data['Final_Exam_Score']
```

Figure 6 : Sélection de la variable explicative et de la variable cible pour la régression linéaire simple.

2.2.3. Séparation des Données

Le dataset a été divisé en ensembles d'entraînement (pour construire le modèle) et de test (pour évaluer sa performance sur de nouvelles données). Un ratio classique (ex: 80% entraînement, 20% test) a été utilisé.

```
# Séparer les données en training set et test set
# On utilise 80% pour l'entraînement et 20% pour le test
X_train_lin, X_test_lin, y_train_lin, y_test_lin = train_test_split(X_lin, y_lin, test_size=0.2, random_state=0)

print(f"\nDimensions des ensembles Training (Linéaire) : X={X_train_lin.shape}, y={y_train_lin.shape}")
print(f"Dimensions des ensembles Test (Linéaire) : X={X_test_lin.shape}, y={y_test_lin.shape}")
```

Figure 7 : Séparation du dataset en ensembles d'entraînement et de test.

2.2.4. Construction et Entraînement du Modèle

Un modèle `LinearRegression` de la bibliothèque scikit-learn a été instancié et entraîné sur l'ensemble d'entraînement.

```
# Créer une instance du modèle de Régression Linéaire
model_lin = LinearRegression()

# Entraîner le modèle sur les données d'entraînement
model_lin.fit(X_train_lin, y_train_lin)

print("\nModèle de Régression Linéaire entraîné.")
# Afficher les coefficients et l'intercept
print(f"Coefficients (pentes) : {model_lin.coef_}")
print(f"Intercept : {model_lin.intercept_}")
```

Figure 8 : Construction et entraînement du modèle de régression linéaire simple.

2.2.5. Prédiction et Évaluation

Le modèle entraîné a été utilisé pour faire des prédictions sur l'ensemble de test. La performance du modèle a été évaluée à l'aide de métriques telles que l'Erreur Absolue Moyenne (MAE), l'Erreur Quadratique Moyenne (MSE), la Racine Carrée de l'Erreur Quadratique Moyenne (RMSE) et le Coefficient de Détermination (R^2).

```
# Faire des prédictions sur l'ensemble de test
y_pred_lin = model_lin.predict(X_test_lin)

# Évaluer le modèle
mae = mean_absolute_error(y_test_lin, y_pred_lin)
mse = mean_squared_error(y_test_lin, y_pred_lin)
rmse = np.sqrt(mse) # La racine carrée du MSE
r2 = r2_score(y_test_lin, y_pred_lin) # Coefficient de détermination R²

print("\n--- Évaluation du Modèle de Régression Linéaire ---")
print(f"Erreur Absolue Moyenne (MAE) : {mae:.2f}")
print(f"Erreur Quadratique Moyenne (MSE) : {mse:.2f}")
print(f"Racine Carrée de l'Erreur Quadratique Moyenne (RMSE) : {rmse:.2f}")
print(f"Coefficient de Détermination (R²) : {r2:.2f}")
```

```
--- Évaluation du Modèle de Régression Linéaire ---
Erreur Absolue Moyenne (MAE) : 0.99
Erreur Quadratique Moyenne (MSE) : 1.51
Racine Carrée de l'Erreur Quadratique Moyenne (RMSE) : 1.23
Coefficient de Détermination (R²) : 0.67
```

Figure 9 : Prédiction sur l'ensemble de test et évaluation du modèle linéaire.

2.2.6. Visualisation

Pour visualiser la relation apprise par le modèle, un nuage de points a été tracé montrant les Heures Étudiées (`Hours_Studied`) par rapport à la Note Finale Réelle (`y_test_lin`) sur l'ensemble de test. La droite de régression apprise par le modèle a été superposée à ce nuage de points pour illustrer la tendance linéaire.

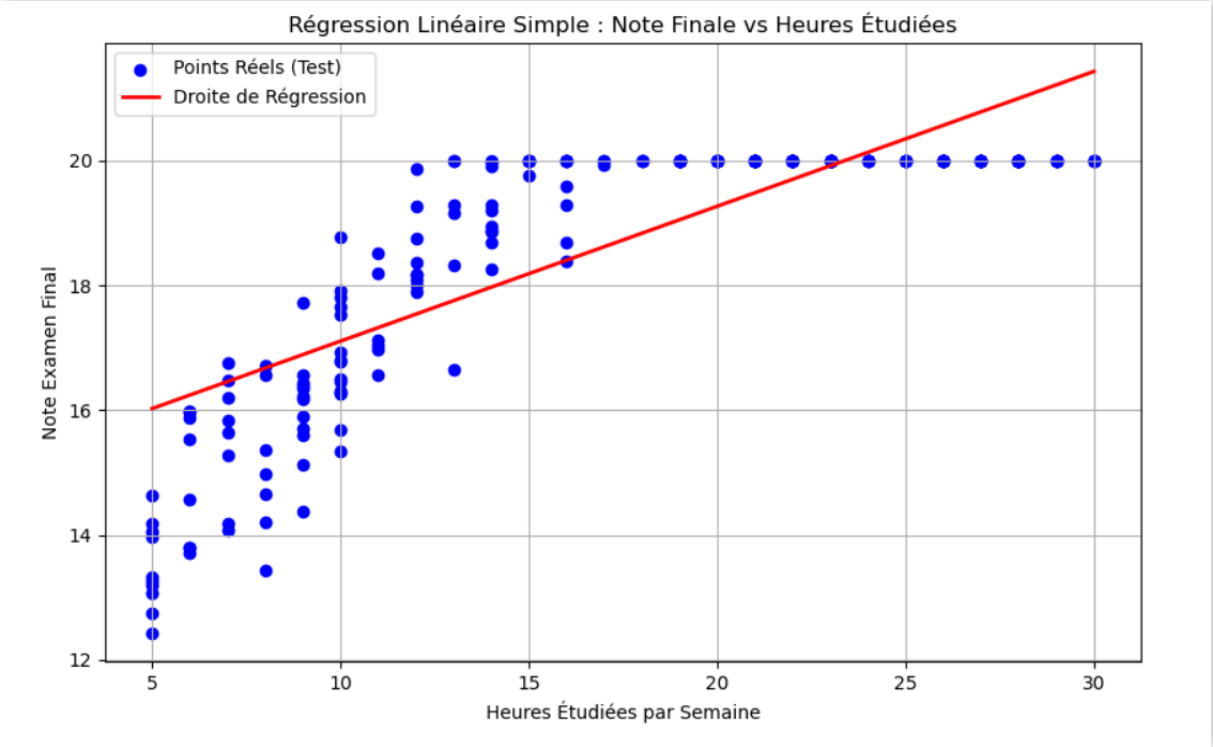


Figure 10 : Visualisation de la régression linéaire simple : points réels de l'ensemble de test et droite de régression apprise.

3. Régression Logistique

3.1. Objectif

L'objectif de cette partie est de construire un modèle de régression logistique pour prédire si un étudiant va **Réussir ou Échouer le Cours (**`Passed_Course`**)**, en utilisant potentiellement plusieurs variables explicatives.

3.2. Démarche dans le Notebook

3.2.1. Préparation des Données (Variables Multiples)

Les variables explicatives X (`Hours_Studied`, `Previous_Score`, `Attendance_Rate`) et la variable cible binaire y (`Passed_Course`) ont été sélectionnées pour cette tâche.

On utilise les mêmes variables explicatives (X) mais la variable cible change.

```
# Sélectionner les variables explicatives (X) et la nouvelle variable cible binaire (y)
X_log = data[['Hours_Studied', 'Previous_Score', 'Attendance_Rate']]
y_log = data['Passed_Course'] # 0 ou 1)
```

Figure 11 : Sélection des variables explicatives et de la variable cible pour la régression logistique.

3.2.2. Séparation des Données (Gestion du Problème de Classe Unique)

La division du dataset en ensembles d'entraînement et de test a initialement posé un problème : l'ensemble d'entraînement contenait parfois une seule classe (que des 0 ou que des 1), empêchant l'entraînement du modèle logistique (`ValueError: This solver needs samples of at least 2 classes...`). Il a été identifié que cela résultait d'un dataset généré n'ayant qu'une seule classe dans la colonne `Passed_Course` après l'application du seuil de réussite.

La solution a consisté à **ajuster le paramètre** `note_passage` **dans le script Python de génération des données** (par exemple, en le passant de 10.0 à 15.0) pour garantir que le dataset final contienne bien un mélange d'étudiants ayant "réussi" (1) et "échoué" (0) selon ce seuil.

De plus, lors de la séparation dans le notebook, le paramètre `stratify=y_log` a été ajouté à la fonction `train_test_split` pour assurer que la proportion des classes soit préservée dans les ensembles d'entraînement et de test.

```
# Cellule de vérification : Distribution des classes dans la variable cible (y_log)
print("\nDistribution des classes dans la variable cible 'Passed_Course' (y_log) dans le dataset complet :")
print(y_log.value_counts())
```

```
Distribution des classes dans la variable cible 'Passed_Course' (y_log) dans le dataset complet :
Passed_Course
1    908
0     92
Name: count, dtype: int64
```

Figure 12 : Vérification de la distribution des classes dans la variable cible binaire après ajustement des paramètres de génération.

```python
# Séparer les Données en training set et test set
X_train_log, X_test_log, y_train_log, y_test_log = train_test_split(X_log, y_log, test_size=0.2, random_state=0, stratify=y_log)

print(f"Dimensions ensembles Training (Logistique) : X={X_train_log.shape}, y={y_train_log.shape}")
print(f"Dimensions ensembles Test (Logistique) : X={X_test_log.shape}, y={y_test_log.shape}")
```

Figure 13 : Séparation stratifiée du dataset pour la régression logistique.

3.2.3. Construction et Entraînement du Modèle

Un modèle `LogisticRegression` a été instancié et entraîné sur l'ensemble d'entraînement (qui contient maintenant les deux classes grâce à `stratify`).

```python
# Créer une instance du modèle de Régression Logistique
model_log = LogisticRegression(solver='liblinear', random_state=0)

# Entraîner le modèle
model_log.fit(X_train_log, y_train_log)

print("\nModèle de Régression Logistique entraîné.")
```

Figure 14 : Construction et entraînement du modèle de régression logistique principal.

3.2.4. Prédiction et Évaluation

Les prédictions ont été faites sur l'ensemble de test. La performance du modèle a été évaluée à l'aide de métriques de classification pertinentes, telles que l'Accuracy, la Matrice de Confusion et le Rapport de Classification (incluant Précision, Rappel, F1-Score).

```
--- Évaluation du Modèle de Régression Logistique ---
Accuracy (Précision Globale) : 0.96

Matrice de Confusion :
[[ 13   5]
 [  2 180]]

Rapport de Classification :
              precision    recall  f1-score   support

           0       0.87      0.72      0.79        18
           1       0.97      0.99      0.98       182

    accuracy                           0.96       200
   macro avg       0.92      0.86      0.88       200
weighted avg       0.96      0.96      0.96       200
```

Figure 15 : Prédiction sur l'ensemble de test et évaluation du modèle logistique.

3.2.5. Visualisation

Bien que le modèle logistique principal utilise plusieurs prédicteurs, une visualisation 2D simple a été réalisée pour illustrer la relation entre la probabilité de réussite et un seul prédicteur (`Hours_Studied`). Un modèle logistique simple (un seul prédicteur) a été entraîné spécifiquement pour cette visualisation. Un nuage de points montrant les Heures Étudiées par rapport au résultat réel (0 ou 1) a été tracé, et la courbe sigmoïde représentant la probabilité prédite de réussite en fonction des heures étudiées a été superposée.

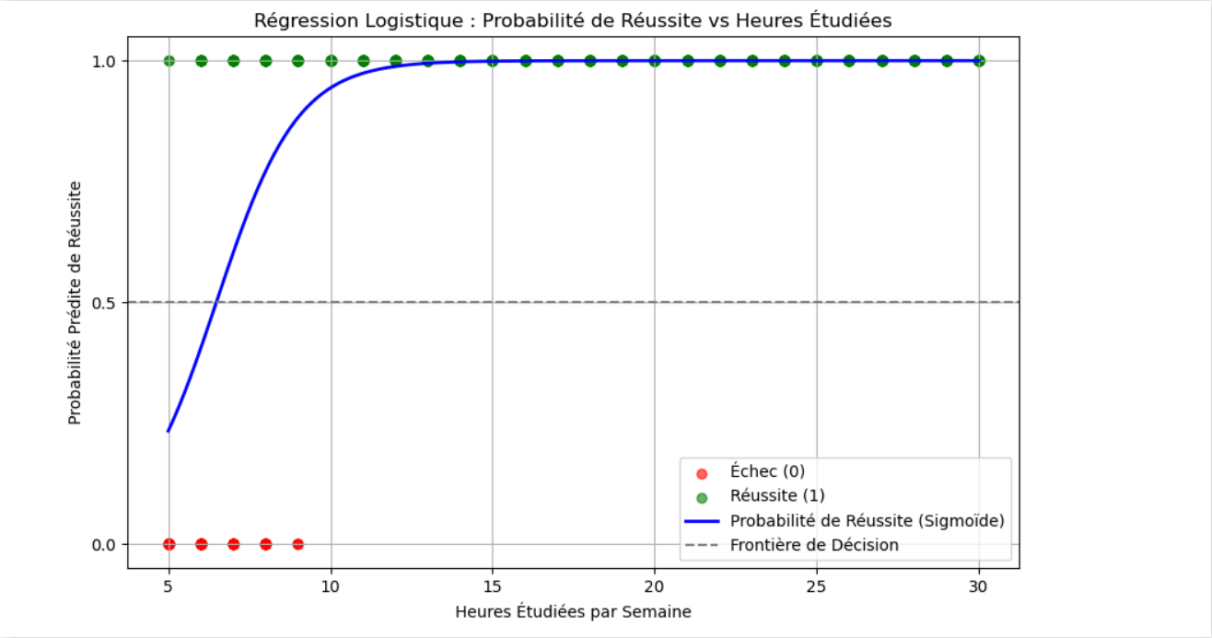


Figure 16 : Visualisation 2D de la régression logistique : points des classes et courbe de probabilité de réussite en fonction des heures étudiées.

Conclusion

Ce TP a permis de mettre en pratique les étapes clés de l'apprentissage supervisé en appliquant la régression linéaire et la régression logistique sur un dataset synthétique personnalisé. Les défis rencontrés lors de la génération des données ont souligné l'importance de la préparation et de la qualité du dataset pour le succès de la modélisation. L'exercice a démontré la capacité des modèles à prédire une variable continue (note finale) et une variable binaire (réussite/échec), ainsi que l'interprétation de leurs résultats et de leurs visualisations associées.