



UNIVERSITÉ LUMIÈRE LYON2

MASTER 2 SISE  
RAPPORT

---

# Développement d'un Package R pour la Régression Logistique Multinomiale Stochastique

---

*Étudiants :*

Hugo COLLIN  
Falonne KPAMEGAN  
Lucile PERBET

*Enseignant :*

Ricco RAKOTOMALALA

1<sup>er</sup> décembre 2024

# Table des matières

<b>Introduction</b>	<b>2</b>
<b>1 Vue d'ensemble du programme</b>	<b>2</b>
1.1 Architecture du projet . . . . .	2
<b>2 Regression Logistique multinomiale stochastique</b>	<b>3</b>
2.1 Fonction <code>handle_missing_values</code> . . . . .	3
2.2 Fonction <code>prepare_data</code> . . . . .	4
2.3 Fonction <code>fit</code> . . . . .	4
2.3.1 Probabilités . . . . .	5
2.3.2 Fonction de coût : log-vraisemblance négative . . . . .	5
2.3.3 Optimisation par descente de gradient stochastique (SGD) . . . . .	5
2.3.4 Traitement par lots (batch) et seuil de tolérance . . . . .	6
2.3.5 Pseudo-code . . . . .	6
2.4 Fonction <code>predict</code> . . . . .	8
2.5 Fonction <code>predict_proba</code> . . . . .	8
2.6 Fonction <code>var_importance</code> . . . . .	9
2.7 Fonction <code>var_select</code> . . . . .	9
2.8 Fonction <code>generate_confusion_matrix</code> . . . . .	10
2.9 Fonctions <code>summary</code> et <code>print</code> . . . . .	10
2.10 Autres fonctions . . . . .	11
2.10.1 Fonction <code>target_select</code> . . . . .	11
<b>3 Application R SHINY</b>	<b>12</b>
3.1 Onglet Data . . . . .	13
3.2 Onglet Modeling . . . . .	13
<b>4 Performance du modèle</b>	<b>14</b>
4.1 Comparaison par rapport à scikit-learn sur Python et nnet sur R . . . . .	14
<b>Conclusion et perspectives</b>	<b>16</b>
<b>Références</b>	<b>16</b>

# Introduction

Ce projet a été réalisé dans le cadre d'un travail académique visant à développer et évaluer un modèle de régression logistique multinomiale stochastique pour la classification de données multiclasse. Le modèle, implémenté à l'aide du langage R, repose sur des techniques avancées de machine learning et a été intégré dans une application interactive avec Shiny, permettant une manipulation facile des données et une visualisation claire des résultats.

Dans ce rapport, nous présentons d'abord le contexte et les objectifs du projet, suivis de la description technique du modèle développé. Les différentes étapes, y compris le prétraitement des données, l'optimisation du modèle, et les tests de performance, sont ensuite détaillées. Enfin, nous comparons les résultats obtenus avec notre modèle et ceux d'autres implémentations existantes sur des jeux de données standards, dans le but de démontrer l'efficacité et la pertinence de notre approche.

## 1 Vue d'ensemble du programme

L'architecture du programme repose sur plusieurs étapes clés. Dans un premier temps, les données sont préparées grâce à des fonctions intégrées qui automatisent des opérations essentielles comme le partitionnement en ensembles d'entraînement et de test, l'encodage one-hot, et la gestion des variables numériques. Ensuite, le modèle ajuste ses paramètres grâce à une descente de gradient stochastique optimisée pour minimiser une fonction de coût basée sur la log-vraisemblance négative. Une fois le modèle entraîné, les fonctions prédictives permettent d'obtenir à la fois les classes prédites et les probabilités associées.

Une particularité notable de ce programme est l'utilisation d'une gestion interne des jeux de données d'entraînement et de test (`X_train`, `X_test`, `y_train`, `y_test`), rendant l'utilisation des fonctions comme `fit`, `predict`, ou `predict_proba` beaucoup plus intuitive. L'utilisateur n'a pas besoin de spécifier explicitement ces jeux de données à chaque appel, contrairement à des outils comme `scikit-learn`. Cette approche favorise une prise en main rapide tout en réduisant la complexité opérationnelle.

Le programme inclut également des outils d'évaluation comme la matrice de confusion, la précision, le rappel, et le F1-score, ainsi qu'une analyse de l'importance des variables. Ces fonctionnalités permettent une interprétation approfondie des résultats et une validation rigoureuse des performances du modèle. Enfin, l'application Shiny associée offre une interface interactive pour explorer les données, ajuster les paramètres du modèle, et visualiser les résultats de manière conviviale.

Ce projet se distingue par son équilibre entre une implémentation rigoureuse des algorithmes et une expérience utilisateur simplifiée, répondant ainsi aux besoins à la fois pédagogiques et analytiques.

### 1.1 Architecture du projet

Le dossier principal est composé de trois dossiers : `data`, `docs` et `package` et d'un fichier R `interface.r`. Le dossier `data` contient quelques jeu de données pour tester notre package. Le package proprement dit se trouve dans le dossier `package` composé des dossiers `man` pour la documentation et `R` qui contient le modèle de regression logistique

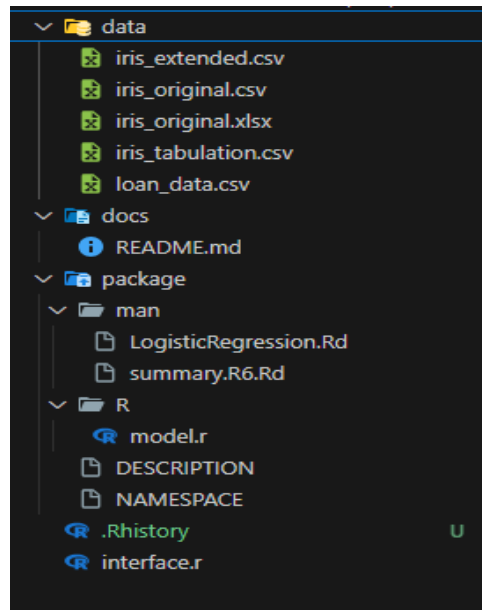


FIGURE 1 – Architecture du projet

## 2 Regression Logistique multinomiale stochastique

### 2.1 Fonction `handle_missing_values`

Elle permet de traiter les valeurs manquantes dans le jeu de données en fonction des méthodes spécifiées pour les variables numériques et catégorielles.

- **Vérification de l'état du modèle** : La fonction commence par vérifier que le modèle a bien été initialisé. Si ce n'est pas le cas, un message d'erreur est renvoyé.
- **Vérification des arguments** : Les arguments `num_method` et `cat_method` sont validés à l'aide de `match.arg`, afin de s'assurer que l'utilisateur a choisi une méthode valide parmi les options proposées (par exemple, "mean", "median", "mode", etc.).
- **Gestion des valeurs manquantes pour les variables numériques** : La fonction vérifie chaque variable numérique et applique la méthode spécifiée par l'utilisateur pour gérer les valeurs manquantes. Les options disponibles sont :
  - "none" : ne rien faire,
  - "mean" : remplacer les valeurs manquantes par la moyenne,
  - "median" : remplacer les valeurs manquantes par la médiane,
  - "mode" : remplacer les valeurs manquantes par la valeur la plus fréquente,
  - "remove" : supprimer les lignes contenant des valeurs manquantes.
- **Gestion des valeurs manquantes pour les variables catégorielles** : De même, pour les variables catégorielles, les méthodes suivantes sont proposées pour gérer les valeurs manquantes :
  - "none" : ne rien faire,
  - "mode" : remplacer les valeurs manquantes par la modalité la plus fréquente,
  - "remove" : supprimer les lignes contenant des valeurs manquantes.
- **Calcul du nombre de valeurs manquantes** : Après avoir traité les valeurs manquantes, la fonction calcule le nombre total de valeurs manquantes ainsi que le

pourcentage de ces valeurs par rapport au jeu de données complet. Ces informations sont stockées dans les attributs `missing_values` et `missing_values_percent`.

- **Mise à jour des données** : Les données traitées sont stockées dans l'objet du modèle, prêtes pour les étapes suivantes de l'analyse ou de l'entraînement du modèle.

La fonction `handle_missing_values` est essentielle pour garantir que les données utilisées par le modèle ne contiennent pas de valeurs manquantes, ce qui pourrait entraîner des erreurs ou de mauvaises performances du modèle.

## 2.2 Fonction `prepare_data`

La fonction `prepare_data` est responsable du prétraitement des données, ce qui inclut la gestion des colonnes à supprimer, l'encodage des variables catégoriques, la normalisation des variables numériques, et la séparation des données en ensembles d'entraînement et de test. Elle permet de s'assurer que toutes les variables sont prêtes et normalisées pour l'apprentissage. Cette fonction est scindée en plusieurs étapes-clés :

- **Vérification de l'état du modèle** : La fonction commence par vérifier que le modèle a bien été initialisé. Si ce n'est pas le cas, un message d'erreur est renvoyé.
- **Sélection des colonnes** : La fonction récupère les colonnes de prédiction en excluant la colonne cible et les colonnes spécifiées pour suppression. Un contrôle est effectué pour s'assurer que la colonne cible n'est pas incluse dans les colonnes à supprimer.
- **Encodage de la variable cible** : Si la variable cible (`target`) n'est pas déjà un facteur, elle est convertie en facteur.
- **Encodage des variables catégoriques** : Pour chaque variable catégorique (facteur ou caractère), un encodage one-hot est appliqué. Cela crée une nouvelle colonne binaire pour chaque niveau de la variable. Les anciennes colonnes catégoriques sont ensuite supprimées.
- **Normalisation des variables numériques** : Les variables numériques sont normalisées (centrées et réduites) à l'aide de la fonction `scale`, ce qui garantit que toutes les variables ont une échelle comparable.
- **Séparation des données en ensembles d'entraînement et de test** : Les données sont séparées de manière aléatoire en deux ensembles : un ensemble d'entraînement et un ensemble de test, en fonction du paramètre `test_size`.
- **Stockage des données préparées** : Les matrices de données (`X`, `y`, `X_train`, `y_train`, `X_test`, `y_test`) sont stockées dans l'objet du modèle pour être utilisées lors de l'entraînement.
- **Mise à jour de l'état du modèle** : Après la préparation des données, l'état du modèle est mis à jour pour indiquer que les données sont prêtes à être utilisées pour l'entraînement.

## 2.3 Fonction `fit`

La méthode `fit` implémente l'entraînement d'un modèle de régression logistique multinomiale en utilisant une descente de gradient stochastique (SGD). L'objectif est de minimiser la fonction coût en utilisant la log-vraisemblance négative.

### 2.3.1 Probabilités

Dans le cadre de la régression logistique multinomiale, le **logit** est utilisé pour modéliser la probabilité qu'une observation appartienne à une classe particulière parmi  $K$  classes possibles. Pour une observation donnée  $\mathbf{x}_i$ , le logit pour la classe  $k$  est défini comme une combinaison linéaire des prédicteurs pondérés par un vecteur de coefficients spécifique à la classe :

$$z_{i,k} = \mathbf{w}_k^\top \mathbf{x}_i,$$

où  $\mathbf{w}_k$  est le vecteur des poids associés à la classe  $k$ , et  $\mathbf{x}_i$  est le vecteur des caractéristiques (ou prédicteurs) pour l'observation  $i$ . Le logit  $z_{i,k}$  représente une mesure non normalisée de l'association entre les prédicteurs et la classe  $k$ . Ces logits servent ensuite à calculer les probabilités prédites via une fonction softmax, garantissant que les probabilités pour toutes les classes  $k = 1, \dots, K$  soient positives et se somment à 1.

Les probabilités prédictives pour chaque classe  $k$  sont obtenues via la fonction softmax :

$$P(y_i = k | \mathbf{x}_i) = \frac{\exp(z_{i,k})}{\sum_{j=1}^K \exp(z_{i,j})}$$

### 2.3.2 Fonction de coût : log-vraisemblance négative

La fonction de coût utilisée dans la régression logistique multinomiale est la log-vraisemblance négative, qui mesure l'écart entre les probabilités prédites et les observations réelles. Elle est définie comme suit :

$$\mathcal{L}(\mathbf{W}) = -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K y_{i,k} \log P(y_i = k | \mathbf{x}_i, \mathbf{W}),$$

où  $N$  est le nombre total d'observations,  $K$  le nombre de classes,  $y_{i,k}$  est une variable indicatrice prenant la valeur 1 si l'observation  $i$  appartient à la classe  $k$ , et  $P(y_i = k | \mathbf{x}_i, \mathbf{W})$  est la probabilité prédite que l'observation  $i$  appartienne à la classe  $k$ , calculée via la fonction softmax. Cette fonction de coût pénalise fortement les prédictions incorrectes en attribuant un poids élevé aux écarts pour des classes effectivement observées. L'objectif de l'apprentissage consiste à minimiser  $\mathcal{L}(\mathbf{W})$  afin d'optimiser les paramètres  $\mathbf{W}$  du modèle.

### 2.3.3 Optimisation par descente de gradient stochastique (SGD)

L'optimisation des paramètres du modèle de régression logistique multinomiale repose sur la méthode de la descente de gradient stochastique (SGD). Cette méthode permet de minimiser la fonction de coût en mettant à jour les poids  $\mathbf{w}_k$  associés à chaque classe  $k$  dans la direction opposée au gradient de la log-vraisemblance négative. Le gradient pour un poids  $\mathbf{w}_k$  s'exprime comme suit :

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_k} = \frac{1}{N} \sum_{i=1}^N (P(y_i = k | \mathbf{x}_i) - y_{i,k}) \mathbf{x}_i,$$

où  $P(y_i = k | \mathbf{x}_i)$  est la probabilité prédite pour la classe  $k$ , et  $y_{i,k}$  est une variable indicatrice valant 1 si l'observation  $i$  appartient à la classe  $k$ , sinon 0. La mise à jour des poids s'effectue selon la règle suivante :

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}_k},$$

où  $\eta$  représente le taux d'apprentissage. L'utilisation du SGD permet une mise à jour progressive des poids après évaluation sur un petit lot (ou une seule observation), rendant l'algorithme adapté aux grands ensembles de données.

### 2.3.4 Traitement par lots (batch) et seuil de tolérance

Dans notre implémentation de la descente de gradient stochastique (SGD), nous utilisons un traitement par **batch**, qui consiste à diviser l'ensemble des données en petits sous-ensembles appelés *batches*. Chaque batch contient un nombre réduit d'observations, permettant de calculer les gradients et de mettre à jour les poids du modèle de manière itérative.

Le fonctionnement est décrit comme suit :

- **Division des données** : Les données d'entraînement sont séparées en plusieurs batches de taille  $B$  (un hyperparamètre défini par l'utilisateur).
- **Mise à jour par batch** : Pour chaque batch, les gradients de la fonction de coût sont calculés uniquement sur les observations de ce lot. Les poids du modèle sont ensuite mis à jour en fonction de ces gradients.
- **Itération sur plusieurs époques** : Une *époque* correspond au passage complet sur tous les batches. Ce processus est répété sur plusieurs époques pour optimiser les poids.

L'utilisation du traitement par batch présente plusieurs avantages :

- Elle réduit les besoins en mémoire, car seules les données du batch courant sont chargées.
- Elle accélère les calculs en exploitant les architectures modernes comme les GPU ou CPU multi-cœurs.
- Elle combine la précision des gradients calculés sur plusieurs observations avec la rapidité de la mise à jour fréquente des poids.

Pour arrêter l'optimisation, nous utilisons un **seuil de tolérance**, le paramètre `tol` de notre fonction. La descente de gradient s'arrête lorsque la variation de la log-vraisemblance entre deux époques successives devient inférieure à ce seuil, indiquant que le modèle a convergé. Cela permet d'éviter un nombre excessif d'itérations tout en garantissant une solution optimale.

### 2.3.5 Pseudo-code

L'algorithme d'entraînement de la régression logistique multinomiale est décrit dans le pseudocode suivant.

---

**Algorithm 1** Entraînement du modèle de régression logistique multinomiale

---

```
1: Entrée : taux d'apprentissage (learning_rate), nombre d'itérations maximales  
   (max_iter), taille du lot (batch_size), tolérance de convergence (tol)  
2: Sortie : coefficients du modèle  
3: if les données ne sont pas préparées then  
4:   Stop "Les données doivent être préparées avant l'entraînement."  
5: end if  
6: Récupérer les données d'entraînement :  $X$  et  $y$   
7: Initialiser les coefficients du modèle  
8: Encoder les classes de  $y$  en format One-Hot  
9: Initialiser la log-vraisemblance (prev_log_likelihood à  $-\text{Inf}$ )  
10: for  $i = 1$  à max_iter do  
11:   Mélanger les indices des données  
12:   for chaque lot de données de taille batch_size do  
13:     Extraire le sous-ensemble de données  $X_{\text{batch}}$  et  $y_{\text{batch}}$   
14:     Calculer les scores :  $\text{scores} = X_{\text{batch}} \times \text{coefficients}$   
15:     Appliquer la fonction Softmax pour obtenir les probabilités  
16:     Calculer le gradient :  $\text{gradient} = X_{\text{batch}}^T \times (\text{probabilités} - y_{\text{batch}}) / \text{taille du lot}$   
17:     Mettre à jour les coefficients :  $\text{coefficients} = \text{coefficients} - \text{learning\_rate} \times$   
       gradient  
18:   end for  
19:   Calculer la log-vraisemblance pour l'ensemble des données  
20:   if la log-vraisemblance est invalide ou infinie then  
21:     Stop "Log-vraisemblance invalide, arrêter l'entraînement."  
22:   end if  
23:   if convergence atteinte, i.e.,  $|\log\_likelihood - \text{prev\_log\_likelihood}| < \text{tol}$  then  
24:     Stop "Convergence atteinte à l'itération  $i$ ."  
25:   end if  
26:   Mettre à jour prev_log_likelihood  
27:   if  $i \% 100 == 0$  then  
28:     Afficher la log-vraisemblance à l'itération  $i$   
29:   end if  
30: end for  
31: Mettre à jour les coefficients finaux du modèle  
32: Mettre à jour l'état du modèle : modèle prêt pour prédiction
```

---



## 2.4 Fonction predict

La fonction `predict` est utilisée pour effectuer des prédictions sur les données de test en se basant sur le modèle de régression logistique multinomial entraîné avec la fonction `fit()`. Elle retourne la performance de la prédiction.

### Détails des étapes :

- **Vérification de l'état du modèle** : Avant toute prédiction, la fonction s'assure que le modèle a été entraîné (`fit`). Si ce n'est pas le cas, un message d'erreur informatif est levé : "You must fit the model before making predictions by calling the 'fit' method."
- **Calcul des scores logistiques (*logits*)** : Une multiplication matricielle est effectuée entre les données de test (avec le biais ajouté) et les coefficients du modèle (`self$coefficients`).

$$\text{scores} = X \times \text{coefficients}$$

Chaque ligne de la matrice `scores` représente les scores calculés pour une observation, et chaque colonne correspond à une classe.

- **Application de la fonction Softmax** : La fonction Softmax est appliquée pour transformer les scores en probabilités, garantissant que la somme des probabilités pour chaque observation est égale à 1 :

$$\text{softmax}(z_i) = \frac{\exp(z_i - \max(z))}{\sum_j \exp(z_j - \max(z))}$$

Ici,  $\max(z)$  est utilisé pour éviter les problèmes numériques dus à des valeurs exponentielles très élevées.

- **Détermination des classes prédictives** : Pour chaque observation, la classe ayant la probabilité la plus élevée est sélectionnée comme prédiction. Cela se fait via :

$$\text{predicted\_class} = \arg \max(\text{softmax\_probs})$$

Ces classes prédictives sont stockées dans `self$predicted_targets`.

- **Évaluation de la performance** : Une comparaison entre les classes prédictives (`predicted_targets`) et les vraies classes (`y_test`) permet de calculer la précision globale (`accuracy`) :

$$\text{accuracy} = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total d'observations}}$$

- **Mise à jour de l'état du modèle** : Après avoir effectué les prédictions, l'état interne du modèle est mis à jour pour indiquer qu'il est maintenant en phase "prédictions réalisées" (`private$state = "predicted"`).

## 2.5 Fonction predict\_proba

La fonction `predict_proba` calcule les probabilités d'appartenance de chaque observation de l'ensemble de test (`X_test`) à chacune des classes cibles. Elle utilise les scores prédits par le modèle (*logits*) et applique la fonction softmax pour transformer ces scores en probabilités.

### Étapes principales :

- Une colonne d'intercept est ajoutée aux données d'entrée pour inclure le biais dans les calculs.
- Les scores (*logits*) pour chaque classe sont calculés en multipliant les données d'entrée augmentées par la matrice des coefficients du modèle.
- La fonction softmax est appliquée aux scores pour normaliser les valeurs et obtenir des probabilités. Cela comprend le calcul des exponentielles des scores (après soustraction du maximum pour éviter les problèmes de dépassement numérique) et la normalisation de ces exponentielles par la somme des valeurs pour chaque observation.
- Les probabilités pour chaque observation et chaque classe sont retournées sous la forme d'une matrice, où chaque ligne correspond à une observation et chaque colonne à une classe.

## 2.6 Fonction `var_importance`

La fonction `var_importance` calcule et retourne l'importance relative des variables explicatives dans le modèle de régression logistique multinomial. Elle permet de quantifier la contribution de chaque variable dans la prédiction des classes.

### Détails des étapes :

- **Extraction des coefficients (sans l'intercept) :** Les coefficients associés aux variables explicatives sont extraits de la matrice des coefficients (`self$coefficients`), en excluant le biais ou intercept (la première colonne). Cela permet de concentrer l'analyse uniquement sur les contributions directes des variables.
- **Calcul des scores d'importance :** La contribution de chaque variable est mesurée par la somme des valeurs absolues de ses coefficients sur toutes les classes.

$$\text{importance\_score}_j = \sum_{k=1}^K |\text{coef}_{jk}|$$

où  $j$  représente une variable et  $k$  les différentes classes cibles.

- **Association des scores aux variables :** Les scores normalisés sont associés aux noms des variables explicatives `data.frame`. Il contient donc le nom de la variable explicative et son score d'importance trié par ordre décroissant.

Voici un exemple de sortie de la fonction `var_importance` :

	Variable	Importance
1	Var_A	0.25
2	Var_B	0.20
3	Var_C	0.15
4	Var_D	0.10

## 2.7 Fonction `var_select`

Cette fonction permet de sélectionner les variables les plus importantes dans un ensemble de données d'entraînement et de test, en fonction des scores d'importance calculés par la méthode `var_importance`. Elle peut conserver un nombre fixe de variables

(`num_vars`) ou filtrer selon un seuil d'importance (`threshold`).

L'attribut `threshold` (*par défaut : 0.05*), spécifie un seuil minimal pour le score d'importance. Les variables ayant un score supérieur ou égal à ce seuil sont conservées. Si `num_vars` est défini, cet argument est ignoré. L'attribut `num_vars` (*par défaut : NULL*) spécifie le nombre de variables les plus importantes à conserver, triées par ordre décroissant d'importance. Si défini, remplace le critère basé sur `threshold`.

Cette fonction ne retourne pas de valeur, par contre, les ensembles `X_train` et `X_test` sont mis à jour pour ne conserver que les variables sélectionnées. La liste des prédicteurs sélectionnés est sauvegardée dans `self$predictors`.

Cette fonction est particulièrement utile pour :

- Réduire la dimensionnalité des données et améliorer les performances.
- Identifier les variables ayant un impact significatif sur les prédictions.
- Optimiser les modèles en éliminant les caractéristiques non pertinentes.

L'idéal après l'utilisation de cette fonction serait de faire un `fit()` et de mesurer la performance du modèle.

## 2.8 Fonction `generate_confusion_matrix`

Cette fonction génère une matrice de confusion basée sur les labels réels et les prédictions effectuées par le modèle. Elle calcule également des métriques de performance importantes telles que la précision globale (*accuracy*), la précision par classe (*precision*), le rappel (*recall*) et le F1-score pour évaluer les performances du modèle. La fonction retourne une liste contenant :

- `confusion_matrix` : Une matrice  $K \times K$  (où  $K$  est le nombre de classes), montrant les correspondances entre les labels réels et prédits.
- `accuracy` : Un score entre 0 et 1 représentant la proportion de prédictions correctes.
- `precision` : Un vecteur contenant la précision pour chaque classe.
- `recall` : Un vecteur contenant le rappel pour chaque classe.
- `f1_score` : Un vecteur contenant le F1-score pour chaque classe.

## 2.9 Fonctions `summary` et `print`

**Fonction `summary` :** Elle fournit un résumé détaillé du modèle de régression logistique multinomiale. Elle affiche les informations suivantes dans la console :

- Le nombre d'observations dans les ensembles d'entraînement (`X_train`) et de test (`X_test`).
- Le nombre de prédicteurs utilisés dans le modèle.
- Le nombre de classes cibles, ainsi que leurs étiquettes (`class_labels`).
- Les fréquences des classes dans l'ensemble d'entraînement.
- Les cinq premiers coefficients du modèle, si le modèle a été ajusté (`fit`).
- L'accuracy sur les données de test, si des prédictions ont été effectuées.

Pour adapter la fonction `summary` de R à notre classe R6, nous avons ajouté une méthode `summary.R6` à l'extérieur de notre classe.

**Fonction `print` :** La fonction `print` offre un résumé concis des principales caractéristiques du modèle. Les informations affichées incluent :

- Le nombre de classes cibles et leurs étiquettes.

- Le nombre de prédicteurs dans le modèle.
- Les cinq premiers coefficients du modèle, si celui-ci a été ajusté (`fit`).

Notons que ces fonctions affichent directement les résultats dans la console et ne retournent aucune valeur. Pour accéder à ces fonctions, deux méthodes s'offrent à vous :

```
model$summary() ou summary(model)
model$print() ou print(model)
```

## 2.10 Autres fonctions

En complément des fonctions principales du projet, nous avons également développé plusieurs fonctionnalités annexes, tout aussi importantes pour enrichir et simplifier l'expérience utilisateur. Ces fonctionnalités ont été conçues pour offrir une meilleure ergonomie, automatiser des tâches récurrentes, et rendre l'interaction avec le modèle plus intuitive. Parmi celles-ci, on peut citer :

### 2.10.1 Fonction `target_select`

La fonction `target_select` permet de sélectionner automatiquement la meilleure colonne cible parmi les colonnes catégoriques d'un jeu de données. Elle repose sur deux critères principaux : l'entropie des classes et la corrélation moyenne avec d'autres colonnes catégoriques. Cette automatisation facilite le choix de la variable cible dans les projets de classification. Elle est particulièrement utile dans des contextes où le jeu de données contient plusieurs colonnes catégoriques, rendant difficile le choix manuel d'une cible pertinente. En automatisant cette tâche, elle permet de gagner du temps tout en garantissant une sélection objective et basée sur des critères bien définis.

#### Étapes principales :

1. **Filtrage initial des colonnes** : Les colonnes catégoriques sont conservées si elles respectent les deux critères suivants :
  - Elles contiennent au moins deux catégories.
  - Elles ont moins de 10% de valeurs manquantes.
2. **Calcul de l'entropie des classes** : L'entropie mesure la dispersion des classes dans une colonne. Les colonnes présentant une entropie inférieure à un seuil (`entropy_threshold`) sont éliminées, car elles sont considérées comme insuffisamment informatives.
3. **Mesure de la corrélation des colonnes (*Cramer's V*)** : La corrélation entre les colonnes catégoriques restantes est calculée à l'aide du coefficient de corrélation de Cramer. Une corrélation élevée peut indiquer une redondance entre les colonnes.
4. **Normalisation et combinaison des scores** :
  - L'entropie est normalisée entre 0 et 1.
  - La corrélation moyenne est également normalisée et inversée pour privilégier les colonnes peu corrélées.
  - Un score combiné est calculé en pondérant l'entropie et la corrélation selon des poids (`weight_entropy` et `weight_correlation`).
5. **Sélection de la meilleure colonne cible** :
  - Les colonnes dont la corrélation moyenne est inférieure à un seuil (`correlation_threshold`) sont privilégiées.

- Parmi ces colonnes, celle ayant le score combiné le plus élevé est choisie comme cible.
- Si aucune colonne ne respecte les seuils de corrélation, la colonne avec le meilleur score combiné global est sélectionnée.

#### Sous-fonctions associées :

- `cramers_v(x, y)` : Cette fonction calcule le coefficient de corrélation de Cramer pour mesurer l'association entre deux variables catégoriques.
- `compute_entropy(y)` : Elle calcule l'entropie des classes pour évaluer la dispersion d'une variable catégorique.

## 3 Application R SHINY

L'application R Shiny a été développée pour fournir une interface utilisateur interactive et intuitive autour du package R dédié à la régression logistique multinomiale stochastique. Elle vise à simplifier l'analyse des données multiclasse en combinant des fonctionnalités avancées avec une visualisation conviviale. L'interface de l'application est divisée trois parties :

1. **Les onglets** : qui permettent de paramétrer les fonctions du package.
2. **La barre d'info** : un espace dédié à la communication des messages importants tels que les avertissements, les erreurs ou les confirmations des actions effectuées.
3. **La visualisation** : elle présente les résultats des manipulations et analyses sous forme de graphiques clairs et interactifs, ainsi que des tableaux détaillés pour explorer les sorties du modèle.

The screenshot shows the R Shiny application interface for Multinomial Logistic Regression. The interface is divided into three main sections, each highlighted with a red circle and a number:

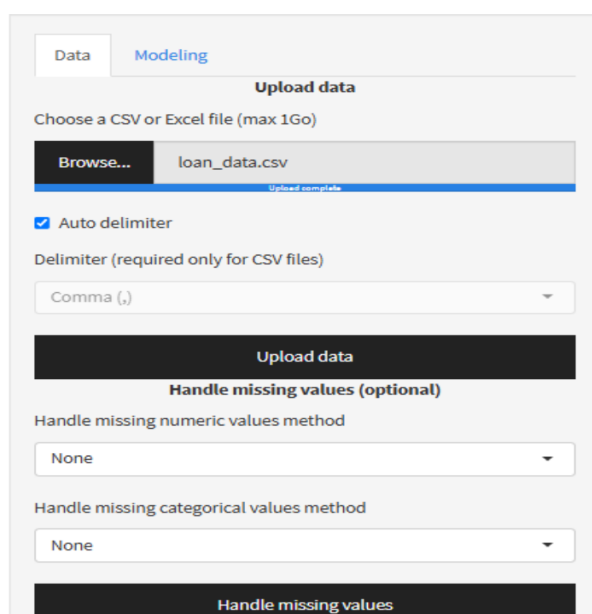
- 1. Data preparation (left):** This section contains two tabs: 'Data' and 'Modeling'. Under 'Modeling', there are two steps: 'Step 1: Prepare data' and 'Step 2: Fit the model'. In 'Step 1', the 'Auto target variable' checkbox is checked, and the 'Target variable (required)' is set to 'loan\_intent'. There is a 'Test size (%)' slider set to 0.3. In 'Step 2', the 'Learning rate' is set to 0.01.
- 2. Information bar (top right):** This section displays messages. It shows '[INFO] The data has been successfully configured.' and '[DATA INFO] Number of missing values : 0' and '[DATA INFO] Percentage of missing values : 0 %'.
- 3. Data preview (bottom right):** This section shows a 'Data preview' table with columns: 'person\_age', 'person\_gender', 'person\_education', 'person\_income', 'person\_emp\_exp', 'person\_home\_ownership', and 'loan'. The table displays 8 rows of data.

FIGURE 2 – Interface R Shiny

Regardons un peu plus en détails la section des **Onglets**.

### 3.1 Onglet Data

L'onglet Data est dédié à la préparation des données, une étape essentielle pour garantir la qualité des analyses. Il permet d'**importer des fichiers** CSV ou Excel (jusqu'à 1 Go), avec une **détection automatique ou manuelle des délimiteurs** pour les fichiers CSV. Cet onglet inclut également des options pour **traiter les valeurs manquantes** : les variables numériques peuvent être corrigées en remplaçant les valeurs manquantes par la moyenne, la médiane ou le mode, ou en supprimant les observations concernées ; pour les variables catégorielles, des méthodes similaires sont proposées, avec la possibilité d'ignorer ou de supprimer les valeurs manquantes. Une fois ces étapes finalisées, les données sont prêtes pour l'analyse prédictive.



The screenshot displays the 'Data' tab interface. At the top, there are two tabs: 'Data' (active) and 'Modeling'. Below the tabs, the section is titled 'Upload data'. It prompts the user to 'Choose a CSV or Excel file (max 1Go)' and provides a 'Browse...' button. A file named 'loan\_data.csv' is shown as uploaded, with a progress bar indicating 'Upload complete'. Below this, there is a checked checkbox for 'Auto delimiter' and a dropdown menu for 'Delimiter (required only for CSV files)' set to 'Comma (,)'. A large black button labeled 'Upload data' is positioned below these options. Underneath, the section 'Handle missing values (optional)' contains two dropdown menus: 'Handle missing numeric values method' set to 'None' and 'Handle missing categorical values method' also set to 'None'. At the bottom of this section is another large black button labeled 'Handle missing values'.

FIGURE 3 – Onglet Data

### 3.2 Onglet Modeling

L'onglet Modeling, quant à lui, accompagne l'utilisateur dans l'application des modèles de régression logistique multinomiale. Il commence par une phase de préparation où l'utilisateur sélectionne la variable cible à prédire, spécifie les colonnes à supprimer si nécessaire, et ajuste la proportion des données réservées au test. Une fois les données préparées via le bouton dédié, l'étape suivante consiste à paramétrer le modèle, notamment en définissant le taux d'apprentissage, avant de lancer l'entraînement.

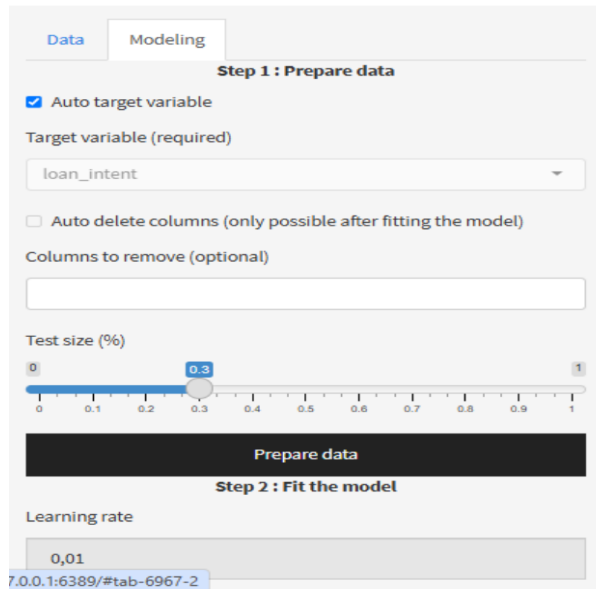


FIGURE 4 – Onglet Modeling

## 4 Performance du modèle

Nous avons évalué notre modèle sur le jeu de données *iris\_extended.csv* qui compte 1200 lignes et 21 colonnes. Le but de cette section est de montrer la performance de notre modèle basé sur le traitement par batch.

Étape	Temps (batch size 1)	Temps (batch size 50)
Temps de traitement	0.06108212 sec	0.04612303 sec
Temps d'entraînement	1.051246 min	8.739952 sec
Temps de prédiction	0.01797605 sec	0.01947784 sec
Temps total	1.060877 min	9.393789 sec

TABLE 1 – Tableau des temps d'exécution du modèle

Le temps de traitement de notre algorithme, sans prendre en compte le `batch_size`, c'est-à-dire, fixé à 1, est beaucoup plus long que lorsque le paramètre est optimisé (50), comme nous pouvons le voir sur la dernière colonne.

### 4.1 Comparaison par rapport à scikit-learn sur Python et nnet sur R

Pour obtenir la comparaison la plus juste, les deux packages ont été testé sur le même jeu de données *iris\_extended.csv* et avec les mêmes traitements implémentés : les variables catégoriques encodées avec la méthode du one-hot, les variables numériques sont normalisées, le jeu de données est séparé en train/test avec un `test_size` de 0.3 et une graine de 123.

Au vu de cette comparaison des performances, il apparaît que, bien que le jeu de données ne soit pas particulièrement volumineux, notre modèle présente des temps de

```

Temps de traitement : 0.0225 secondes
Temps d'entraînement : 0.0169 secondes

```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	132
versicolor	0.99	0.99	0.99	112
virginica	0.99	0.99	0.99	116
accuracy			0.99	360
macro avg	0.99	0.99	0.99	360
weighted avg	0.99	0.99	0.99	360

```

Temps de prédiction : 0.0135 secondes
Temps total : 0.1443 secondes

```

FIGURE 5 – Performance avec scikit-learn sur Python

```

Temps d'entraînement du modèle : 0.124 secondes
Temps de prédiction : 0.004 secondes
Précision du modèle : 98.06 %
Matrice de confusion :
Précision globale : 0.9758065
Rappel global : 0.9918033
F1-score global : 0.9837398

```

FIGURE 6 – Performance avec nnet sur R

```

> print(results$accuracy)
[1] 0.9916667
> print(results$precision)
      0      1      2
1.0000000 0.9915966 0.9831933
> print(results$recall)
      0      1      2
1.0000000 0.9833333 0.9915254
> print(results$f1_score)
      0      1      2
1.0000000 0.9874477 0.9873418

```

FIGURE 7 – Performance de notre package

calcul plus longs, même avec des tailles de batch élevées. Cependant, en termes d'accuracy, le modèle se positionne très favorablement par rapport à Scikit-learn et nnet. En effet, les scores obtenus sont remarquablement similaires, ce qui témoigne de la solidité du modèle en termes de performance prédictive.

## Conclusion et perspectives

En conclusion, le modèle de régression logistique multinomiale stochastique développé dans le cadre de ce projet a démontré son efficacité pour traiter des problématiques de classification multiclasse. Sa conception, basée sur une approche par descente de gradient stochastique, offre une solution flexible et adaptée à des ensembles de données de tailles variées, tout en permettant une convergence rapide grâce à une gestion par lots et un seuil de tolérance défini. L'intégration de l'ensemble des étapes de traitement, de l'encodage des variables catégoriques à la normalisation des variables numériques, renforce la cohérence des prédictions et simplifie son utilisation pratique. Néanmoins, le modèle



présente certaines limites. Il est particulièrement sensible au choix des hyperparamètres, tels que le taux d'apprentissage ou la taille des lots, et peut rencontrer des difficultés face à des données fortement déséquilibrées ou contenant des classes rares. De plus, bien qu'il soit robuste sur des jeux de données standardisés, son implémentation nécessite une connaissance préalable des étapes de prétraitement.

En termes de perspectives, plusieurs axes d'amélioration peuvent être envisagés. Par exemple, l'intégration d'une validation croisée automatisée pour optimiser les hyperparamètres pourrait renforcer la performance du modèle. De même, l'ajout de régularisations L1 ou L2 pourrait permettre de mieux gérer la colinéarité entre les variables prédictives. Enfin, l'extension de l'application Shiny pour intégrer d'autres méthodes de classification ou des visualisations plus interactives offrirait une expérience utilisateur encore plus riche et adaptée aux besoins des analystes.

## Références

- [Wikipedia Stochastic gradient descent](#)
- [Gradient Descent pour le machine learning](#)
- [Interpret Log likelihood](#)
- [Logistic model Maximum likelihood](#)
- [Derivative gradient descent logistic regression - LinkedIn](#)