Université des Sciences et de la Technologie
Houari Boumediene

Faculté d'Informatique
Master 1 - Systèmes Informatiques Intelligents
Module Réseaux neurones

Rapport de Projet
Classification et Régression

Étudiants
Athmane Mohamed Anis
212131088659
BOUROU Melissa
212131049198
NEBBALI Khalida Meriem
202031079174

Enseignante
Mme MOULAI Hadjer

Table des matières

Chapitre 1
Classification

1.1 Introduction

La classification est une méthode d'apprentissage supervisé dans laquelle la variable de sortie à prédire est de nature discrète ou catégorique. Il existe deux types de classification

a- Binaire : la valeur prédite est tiré d'un ensemble de deux valeurs (classes) possibles.

b- Multi-classe : la variable cible peut prendre plus de deux valeurs parmi un ensemble de classes distinctes.

1.2 Objectifs de la classification

La santé mentale des étudiants est un enjeu de plus en plus préoccupant, notamment face à la pression académique, l'isolement social ou encore les incertitudes liées à l'avenir. Détecter précocement les signes de dépression permettrait d'intervenir plus rapidement et de proposer un accompagnement adapté. Dans le cadre de ce projet, Nous appliquons des techniques de classification sur notre dataset afin de prédire si un étudiant est atteint de dépression, en construisant un modèle capable de généraliser efficacement sur de nouvelles données.

1.3 Présentation du Dataset

Ce dataset regroupe un large éventail d'informations visant à comprendre, analyser et prédire les niveaux de dépression chez les étudiants. Il est conçu pour la recherche en psychologie, en science des données et en éducation, en fournissant des informations sur les facteurs qui contribuent aux troubles de la santé mentale des étudiants et en aidant à la mise en place de stratégies d'intervention précoce.

1.3.1 Description détaillée du dataset

- **Nom :** " Student Depression Dataset "
- **Source :** Le dataset est obtenu via Kaggle
- **Définition du problème :** On cherche à prédire l'attribut *" Depression "* étant un attribut catégoriel pouvant être (Oui "1"/ Non "0")
- **Contenu :** On a un ensenble de **27.901** instances sans prétraitement.
- **Description des données :** Ce dataset contient 18 colonnes distinctes dont 10 numériques et 8 catégorielles, voici quelques colonnes phares :

TABLE 1.1 – Les colonnes du dataset pour la classification

Colonne	**Type**	**Description**
id	Numeric	Un identifiant unique attribué à chaque enregistrement d'étudiant.
Gender	String	Le genre de l'étudiant (ex : Masculin, Féminin, Autre).
Age	Numeric	L'âge de l'étudiant en années.
City	String	La ville ou région de résidence de l'étudiant.
Profession	String	Le domaine de travail ou d'étude de l'étudiant.
Academic pressure	Float	Une mesure indiquant le niveau de pression académique.
Work pressure	Float	Une mesure de la pression liée au travail.
CGPA	Float	La moyenne cumulative des notes de l'étudiant.
Study satisfaction	Float	Niveau de satisfaction concernant les études.
Job satisfaction	Float	Satisfaction de l'étudiant vis-à-vis de son travail.
Sleep duration	String	Durée moyenne de sommeil quotidienne.
Dietary habits	String	Habitudes alimentaires et nutritionnelles.
Degree	String	Diplôme ou programme académique poursuivi.
Suicidal thoughts	Binary (Oui/Non)	Pensées suicidaires chez l'étudiant.
Work/Study hours	Numeric	Heures quotidiennes de travail/étude.
Financial stress	String	Stress dû à des préoccupations financières.
Family mental illness	Binary (Oui/Non)	Antécédents familiaux de maladie mentale.
Depression	Numeric	Classe cible : '0' (non), '1' (oui).

1.3.2 Justification du choix

Le dataset a été choisi pour la tâche de classification car il présente plusieurs qualités qui le rendent particulièrement adapté à un projet de machine learning.

- **Richesse des variables :** il contient des informations variées(18 attributs) et liées au mode de vie permettant d'entraîner des modèles complets et réalistes (27.898 lignes).
- **Pertinence sociale :** le sujet est d'actualité et important surtout pour la prévention de la santé mentale chez les étudiants.
- **Accessibilité et qualité :** le dataset est anonyme bien structuré et facilement accessible facilitant son utilisation dans un cadre académique.

1.4 Prétraitement des Données

1.4.1 Visualisation des données

Selon la distribution ci-dessous, on distingue deux classes de la variable cible dépression : la classe majoritaire "Non" (absence de dépression) représente environ 16 000 instances, tandis que la classe "Oui" (présence de dépression) en compte environ 12 000. Avec un total de 27.898 instances, cela indique une répartition eviron 57% (Non) vs 43% (Oui) montrant un bon équilibre entre les deux.

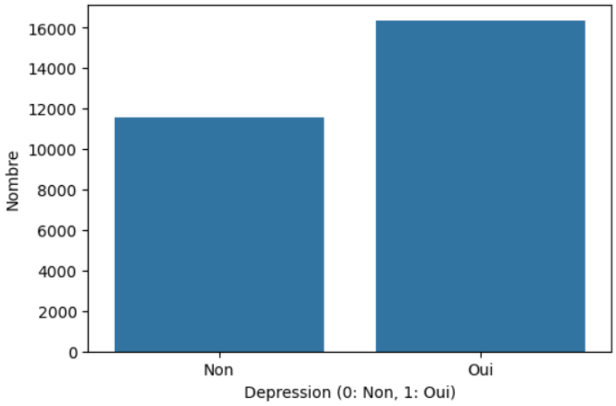


FIGURE 1.1 – Distribution de la variable cible

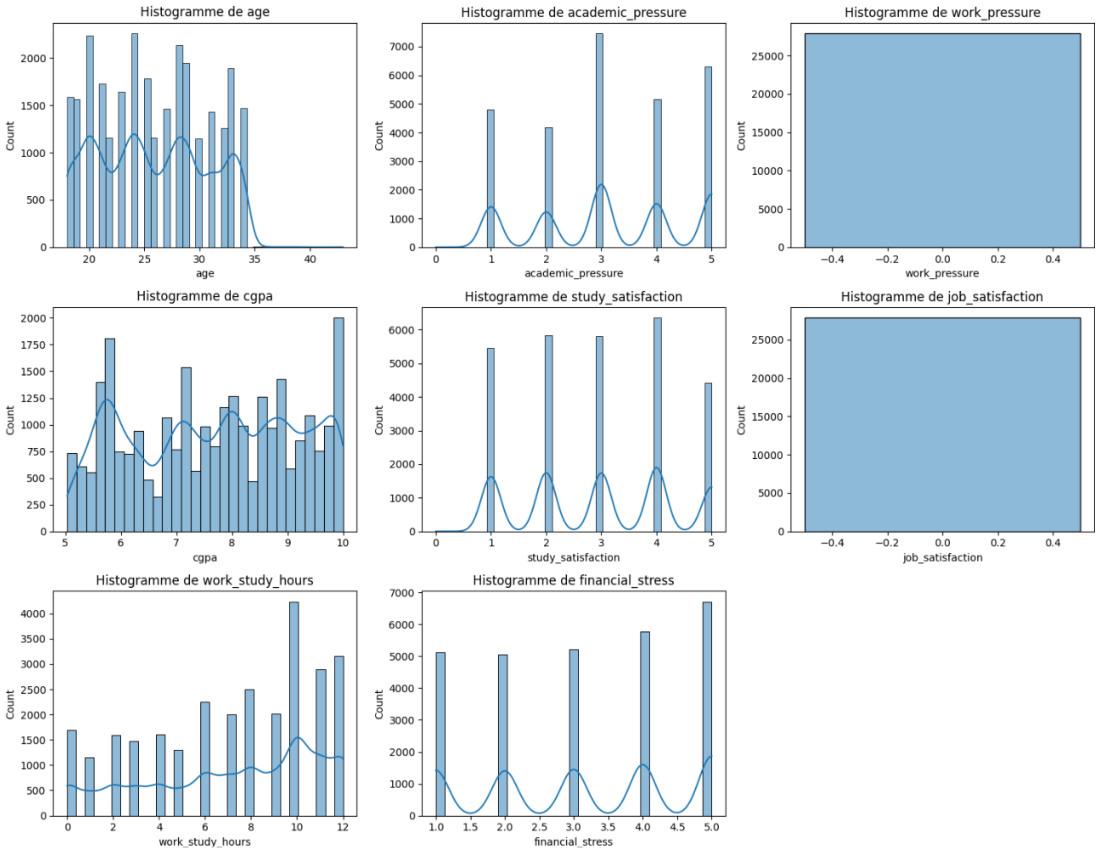


FIGURE 1.2 – Histogrammes des valeurs numériques

1.4.2 Répartition des variables catégorielles importantes

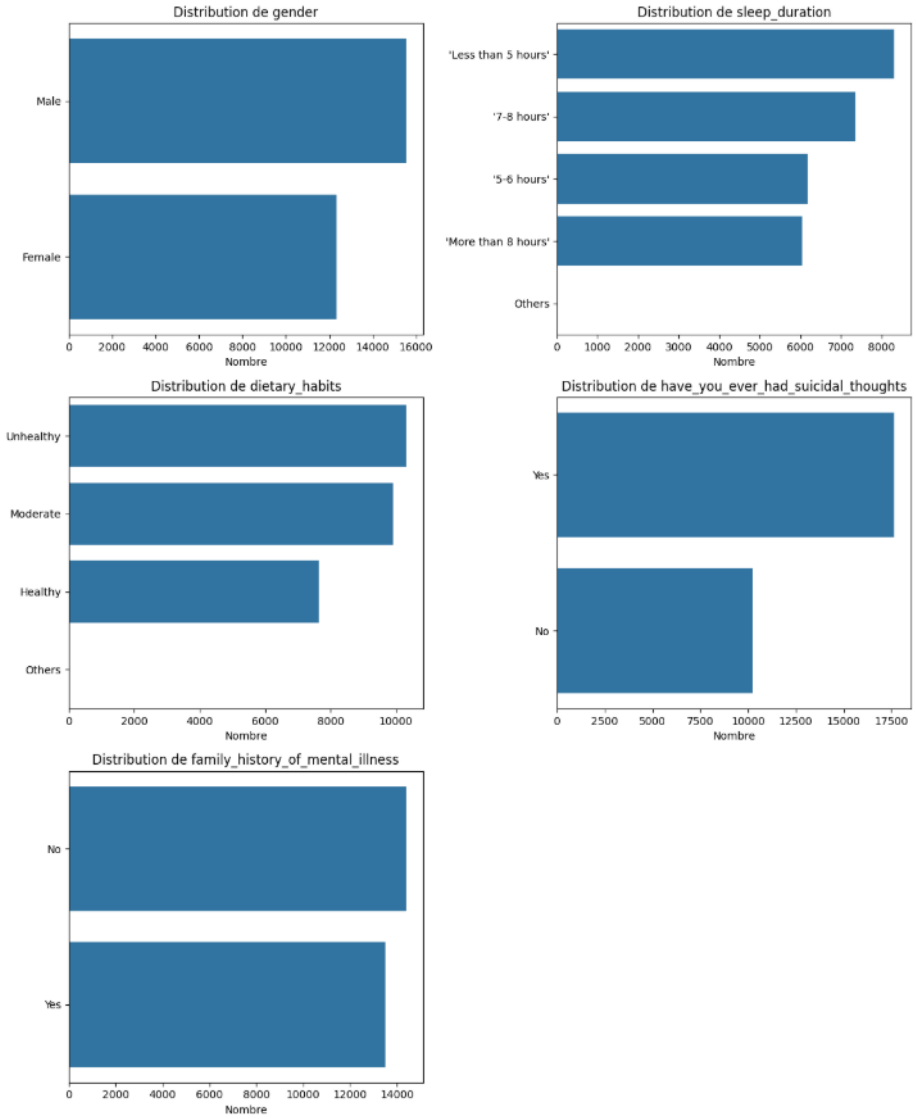


FIGURE 1.3 – Distribution des variables catégorielles

1.4.3 Matrice de corrélation

La matrice de corrélation révèle les informations suivants :

- Les variables les plus liées à la dépression (cible) sont `academic_pressure` (corrélation modérée de 0.47) et `financial_stress (0.36)`, suggérant que ces facteurs pourraient être des prédicteurs clés.
- À l'inverse, age (-0.23) et `study_satisfaction` (-0.17) montrent une relation négative, indiquant un effet potentiellement protecteur.
- Les variables comme `work_pressure`, `cgpa` et `job_satisfaction` semblent neutres (corrélations proches de 0).

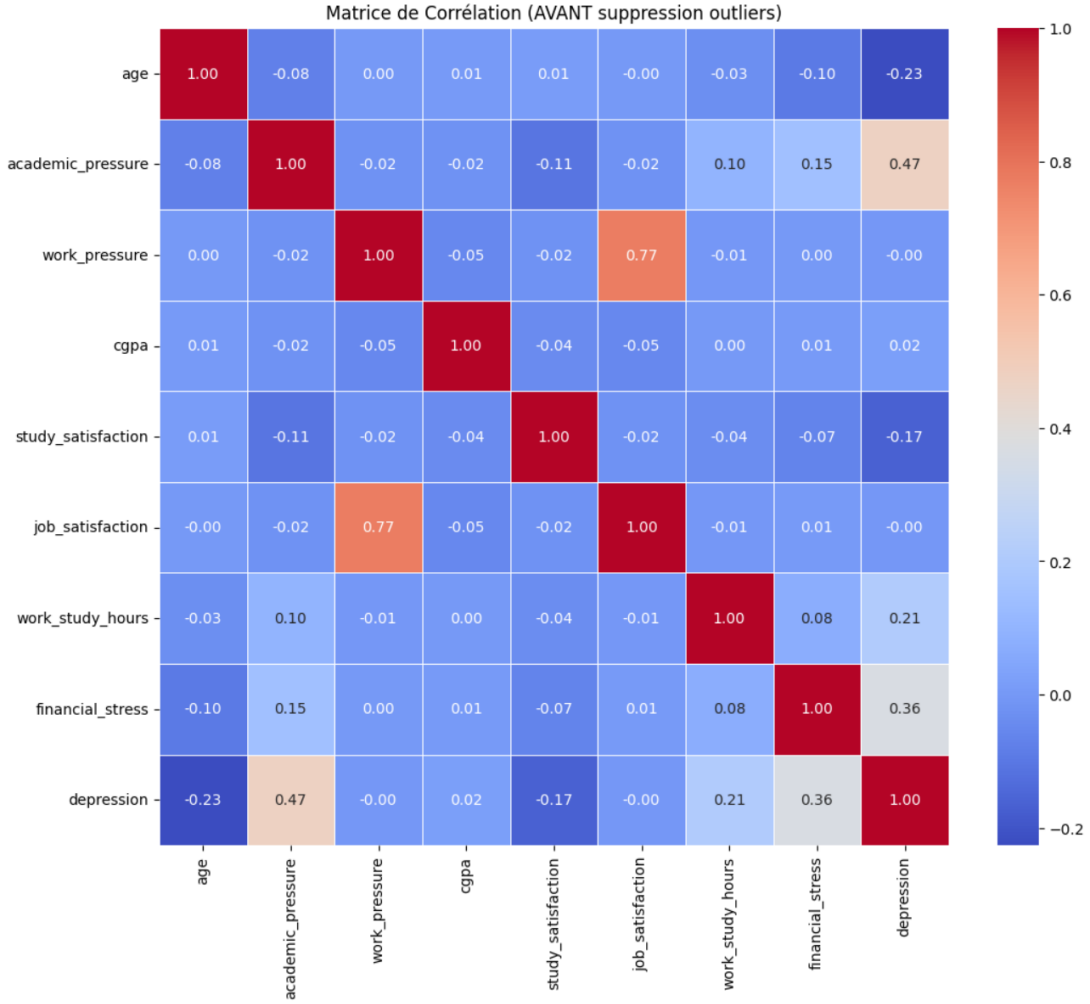


FIGURE 1.4 – Matrice de corrélation avant traitement d'outliers

1.4.4 Nettoyage de données

Gestion des valeurs manquantes

Après inspection du dataset 3 lignes contenant des valeurs null, on les supprime car c'est vraiment très peu et ça n'aura pas un impact.

```
print(f"Nombre de valeurs manquantes avant suppression:\n{df.isnull().sum()}")
nan_count_before = df.isnull().sum().sum()
```

Listing 1.1 – Affichage des valeurs manquantes avant traitement

```
df_cleaned = df.dropna()
print(f"Dimensions apres suppression des NaN: {df_cleaned.shape}")
nan_removed = nan_count_before - df_cleaned.isnull().sum().sum()
print(f"{nan_removed} lignes contenant des NaN ont ete supprimees.")
```

Listing 1.2 – Traitement des valeurs manquantes

Suppression des doublons

Une vérification des doublons a aussi été effectuée et aucun doublon n'a été trouvé dans les données. Aucune suppression n'a été requise.

```
duplicates_before = df_cleaned.duplicated().sum()
print(f"\nNombre de lignes dupliquees avant suppression: {duplicates_before}")
```

Listing 1.3 – Détection des doublons

Standardisation des noms de variables

Remplacer les espaces et caractères spéciaux par '_' et mettre le tout en minuscule.

```
# Remplacer les espaces et caracteres speciaux par '_' et mettre en minuscule
df.columns = [re.sub(r'\W+', '_', col).lower() for col in df.columns]

# Gerer specifiquement le '?' dans le nom de colonne
df.columns = df.columns.str.replace('_$', '', regex=True) # Enlever '_' a la fin
renamed_columns = df.columns.tolist()
print("\nNoms de colonnes nettoyes:")
for old, new in zip(original_columns, renamed_columns):
    if old != new:
        print(f"'{old}' -> '{new}'")
```

Listing 1.4 – Nettoyage des noms de colonnes

Suppression de Colonnes

Les colonnes *ID, âge, city, profession, degree, gender, work_pressure, job_satisfaction* ont été supprimées pour l'entrainement du modèle car elles sont non pertinentes pour la classification ou ont la même valeur pour toutes les lignes.

```
X = df_processed.drop(target_column, axis=1)
y = df_processed[target_column]

cols_to_drop_for_training = ['id','city', 'profession', 'degree', 'work_pressure', '
    job_satisfaction']
X_train_ready = X.drop(columns=[col for col in cols_to_drop_for_training if col in X.
    columns])

print(f"Colonnes utilisees pour l'entrainement: {X_train_ready.columns.tolist()}")
```

Listing 1.5 – Préparation des données pour l'entraînement

Gestion des outliers

Les outliers sont des valeurs numériques anormalement élevées ou basses qui peuvent fausser les analyses statistiques et la performance des modèles. Ici, on identifie les colonnes numériques pour appliquer la méthode *IQR* qui permet de détecter et supprimer ces valeurs extrêmes. On a trouvé les outliers suivant :

- 12 outliers détectés dans 'age'
- 3 outliers détectés dans 'work_pressure'
- 6 outliers détectés dans 'cgpa'
- 2 outliers détectés dans 'job_satisfaction'

Nombre total de lignes supprimées à cause des outliers est 23. Le nombre de ligne devient 27.875.

```
print("\nSuppression des outliers (methode IQR)...")
numerical_cols_conf = df_cleaned.select_dtypes(include=np.number).columns.tolist()
```

Listing 1.6 – Détection des outliers par méthode IQR

Encodage

1. **Features catégorielles à encoder**

   On applique `OneHotEncoder()` sur les colonnes catégorielles(chaîne de caractères) pour les convertir en variables binaires (encodage), tout en ignorant les catégories inconnues et en évitant la redondance avec `drop='first'`.

2. **Features numériques à normaliser**

   Avec `StandardScaler()` les colonnes numériques sont normalisées en les centrant (moyenne = 0) et en les réduisant (écart-type = 1).

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore', drop='first'), categorical_features
    )],
    remainder='passthrough')

print("\nPreprocesseur (StandardScaler + OneHotEncoder) defini pour l'integration dans le
    Pipeline.")
```

Listing 1.7 – Application d'encodage + normalisation

1.5 Modélisation

1.5.1 Définition variable cible

```
# Separer les features (X) et la target (y) du df_processed
target_column = 'depression'
if target_column not in df_processed.columns:
    raise ValueError(f"La colonne cible '{target_column}' n'existe pas dans le DataFrame
    traite.")

X = df_processed.drop(target_column, axis=1)
y = df_processed[target_column]
```

Listing 1.8 – Séparation des features et de la target

1.5.2 Séparation des données

```
X_train, X_test, y_train, y_test = train_test_split(
    X_train_ready,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

Listing 1.9 – Division des donnees en ensembles d'entraînement et de test

1.5.3 Algorithmes de Classification utilisés

Pour ce projet, plusieurs algorithmes de classification ont été utilisés afin de comparer leurs performances et sélectionner le plus adapté au problème de prédiction :

- **Régression Logistique (Logistic Regression)** : un modèle linéaire utilisé pour la classification binaire, efficace et rapide, surtout en présence de données linéairement séparables.

```
LogisticRegression (
    solver='liblinear',
    random_state=42,
    max_iter=1000  )
```

Listing 1.10 – Régression Logistique

- **Random Forest (RandomForestClassifier)** : un ensemble d'arbres de décision permettant de réduire le surapprentissage et d'améliorer la robustesse.

```
RandomForestClassifier( random_state=42 )
```

Listing 1.11 – Random Forest

- **Support Vector Classifier (SVC)** : particulièrement adapté aux espaces de grande dimension, en utilisant un noyau RBF dans notre configuration.

```
SVC (probability=True, random_state=42)
```

Listing 1.12 – SVM Classifier

- **XGBoost (XGBClassifier)** : un algorithme de boosting performant, largement utilisé pour les compétitions de machine learning.

```
XGBClassifier(
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42
)
```

Listing 1.13 – XGBoost

- **LightGBM (LGBMClassifier)** : une alternative rapide à XGBoost, utilisant une croissance des arbres feuille à feuille.

```
LGBMClassifier(random_state=42)
```

Listing 1.14 – LightGBM

- **HistGradientBoosting (HistGradientBoostingClassifier)** : un algorithme de boosting récent intégré à `scikit-learn`, optimisé pour les grandes bases de données.

```
HistGradientBoostingClassifier( random_state=42 )
```

Listing 1.15 – HistGradientBoosting

1.5.4 Paramétrage des modèles d'entraînement

Chaque modèle a été entraîné au sein d'un **pipeline**(outil permettant de chaîner plusieurs étapes de traitement en ML en un seul objet cohérent.) comprenant un prétraitement (normalisation, encodage, etc.) suivi du classifieur. L'ajustement des hyperparamètres a été réalisé à l'aide de **GridSearchCV** avec *validation croisée* (technique essentielle en machine learning pour évaluer la performance d'un modèle de manière robuste) à 3 plis (cv=3) et une métrique d'évaluation basée sur l'*accuracy*. Le calcul parallèle a été activé aussi pour accélérer l'entraînement.

Classifieur	**Paramètre**	**Valeurs**
LogisticRegression	classifier__C	[0.01, 0.1, 1, 10]
	classifier__penalty	['l1', 'l2']
RandomForestClassifier	classifier__n_estimators	[50, 100]
	classifier__max_depth	[None, 10]
	classifier__min_samples_split	[2, 5]
SVC	classifier__C	[0.1, 1]
	classifier__gamma	['scale', 'auto']
	classifier__kernel	['rbf']
XGBoost	classifier__n_estimators	[100, 200]
	classifier__max_depth	[3, 5, 7]
	classifier__learning_rate	[0.01, 0.1, 0.2]
	classifier__subsample	[0.8, 1.0]
LightGBM	classifier__n_estimators	[50, 100]
	classifier__max_depth	[3, 5]
	classifier__learning_rate	[0.01, 0.1]
	classifier__num_leaves	[31, 63]
	classifier__min_child_samples	[20, 50]
HistGradientBoosting	classifier__max_iter	[100, 200]
	classifier__learning_rate	[0.01, 0.1]
	classifier__max_depth	[3, 5]
	classifier__min_samples_leaf	[20, 50]

1.5.5 Meilleurs paramètres

Après exécution de **GridSearchCV** pour chaque modèle voici la meilleur combinaison de paramètres trouvée pour chaque modèle :

TABLE 1.2 – Résumé des meilleurs paramètres, durée d'entraînement et accuracy

Modèle	Meilleurs paramètres	Durée (s)	Accuracy
Logistic Regression	`C=10, penalty='l1'`	6.80	0.8477
Random Forest	`n_estimators=100, max_depth=10, min_samples_split=2`	11.12	0.8850
SVC	`C=0.1, gamma='scale', kernel='rbf'`	338.48	0.8504
XGBoost	`learning_rate=0.1, max_depth=3, n_estimators=200, subsample=0.8`	18.60	0.8558
LightGBM	`learning_rate=0.1, max_depth=5, n_estimators=100, num_leaves=31, min_child_samples=50`	16.76	0.8512
HistGradientBoosting	`learning_rate=0.1, max_depth=5, max_iter=100, min_samples_leaf=20`	14.71	0.8643

1.6 Analyse des résultats

1.6.1 Métriques d'Évaluation en Classification

Les performances ont été mesurées sur l'ensemble de test à l'aide des métriques suivantes : précision (Accuracy), F1-Score pondéré. Les résultats sont présentés dans le tableau 1.3.

TABLE 1.3 – Performances comparées sur l'ensemble de test

Modèle	Accuracy	F1-Score (Pondéré)	Temps (s)
Régression Logistique	0.8477	0.85	6.80
Forêt Aléatoire	0.8850	0.88	11.12
SVC	0.8504	0.85	338.48
XGBoost	0.8558	0.86	18.60
LightGBM	0.8512	0.85	16.76
HistGradientBoosting	0.8643	0.86	14.71

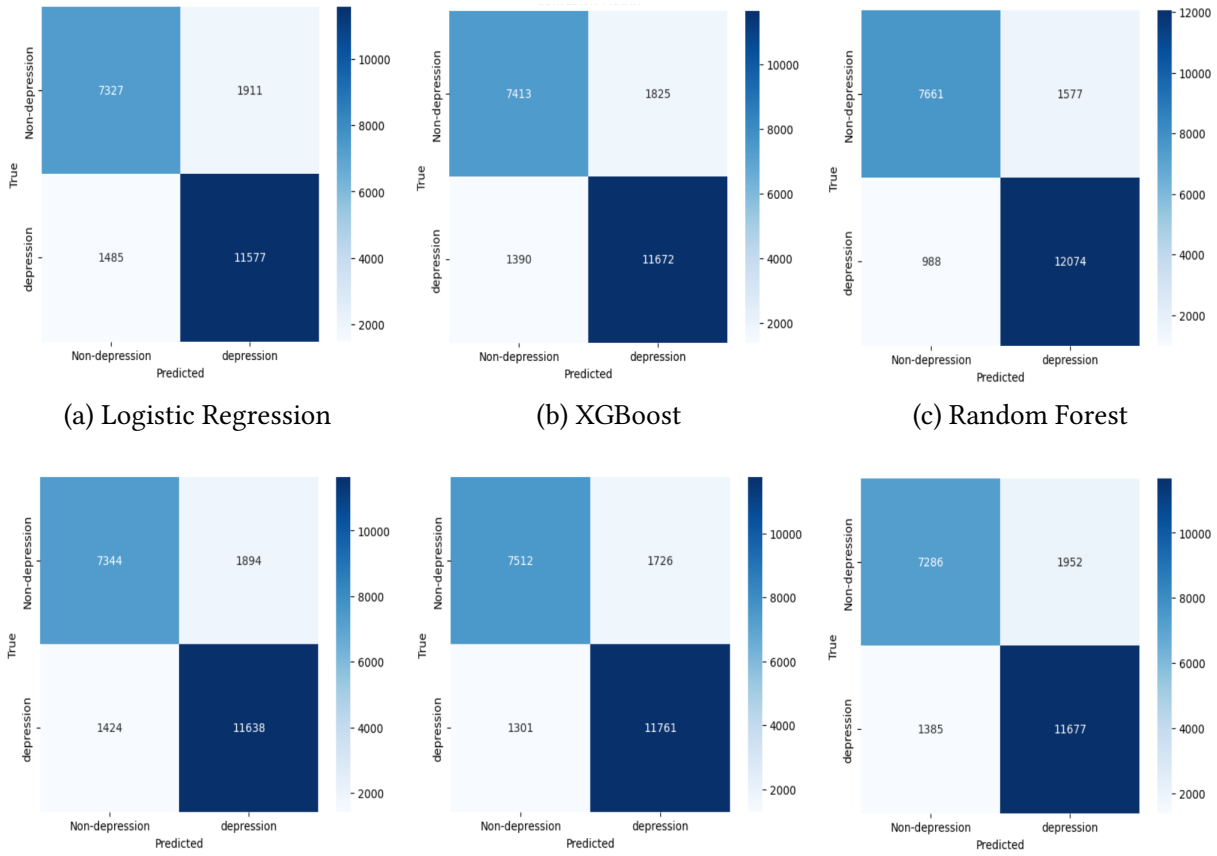
1.6.2 Matrices de confusion

Les matrices de confusion comparées mettent en évidence des performances variables selon les modèles, avec un enjeu crucial : la minimisation des faux négatifs (individus dépressifs incorrectement classés comme non dépressifs 'FN'). Cette erreur est particulièrement grave dans le contexte clinique, car elle peut retarder une prise en charge nécessaire.

Parmi les algorithmes, XGBoost (figure 1.5b) et LightGBM (figure 1.5d) se distinguent par un meilleur équilibre entre sensibilité (détection des vrais dépressifs) et spécificité (rejet des vrais non-dépressifs), tout en limitant les faux négatifs (taux très faible). La Régression Logistique (figure 1.5a) et la Forêt Aléatoire (figure 1.5c) donnent des résultats corrects mais moins optimaux. À l'inverse, le SVC (figure 1.5f) montre des lacunes marquées avec un taux de faux négatifs un peu plus élevé que les autres, ce qui le rend moins adapté à cette application.

Les matrices de confusion révèlent que, globalement, tous les modèles fournissent des résultats acceptables, avec des Vrais Positifs (TP) et Vrais Négatifs (TN) très élevés dans l'en-

semble. Cela indique une bonne capacité générale à classer correctement les cas de dépression et de non-dépression.



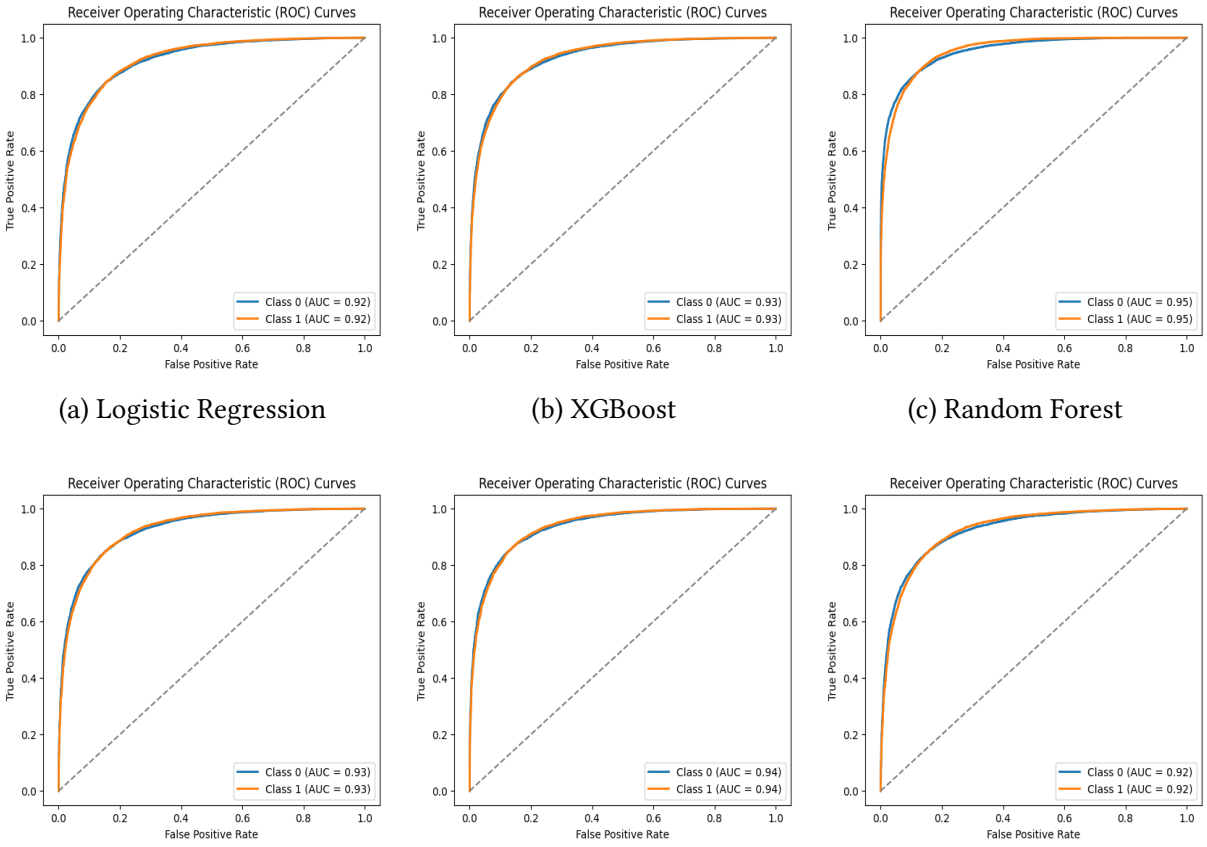
(a) Logistic Regression (b) XGBoost (c) Random Forest

(d) LightGBM (e) Hist Gradient Boosting (f) SVC

FIGURE 1.5 – Performance des matrices de confusion pour les différents modèles

1.6.3 Courbes ROC

Les courbes ROC démontrent que tous les modèles obtiennent des résultats acceptables (AUC >= 0.92), montrés dans la figure 1.6 pour les deux classes pour discriminer les cas dépressifs et non-dépressifs. Ces résultats, combinés aux analyses précédentes des matrices de confusion, confirment que les méthodes de boosting (*LightGBM, XGBoost, Hist Gradient Boosting*) constituent les choix les plus optimaux pour cette classification.



(a) Logistic Regression (b) XGBoost (c) Random Forest

(d) LightGBM (e) Hist Gradient Boosting (f) SVC

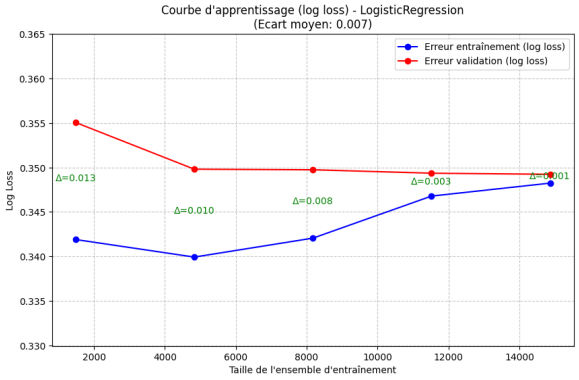
FIGURE 1.6 – Courbe ROC pour les différents modèles

1.6.4 Courbe d'apprentissage - learning curve -

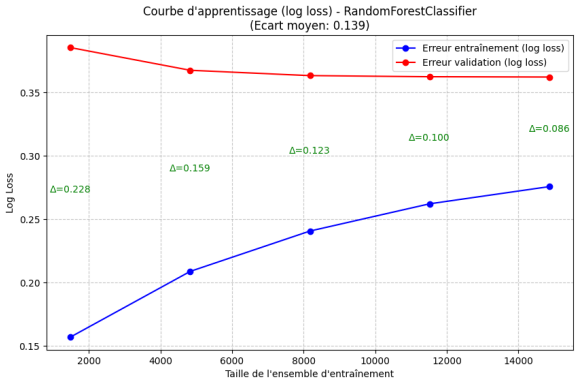
Les courbes d'apprentissage montrent l'évolution de l'erreur d'entraînement et de l'erreur de validation en fonction de la taille de l'ensemble d'entraînement. Elles permettent de diagnostiquer si un modèle souffre de biais (underfitting) ou de variance (overfitting).

Ces courbes d'apprentissage montrent que tous les modèles bénéficient d'une augmentation de la taille de l'ensemble d'entraînement, la perte de validation diminuant à mesure que plus de données sont ajoutées.

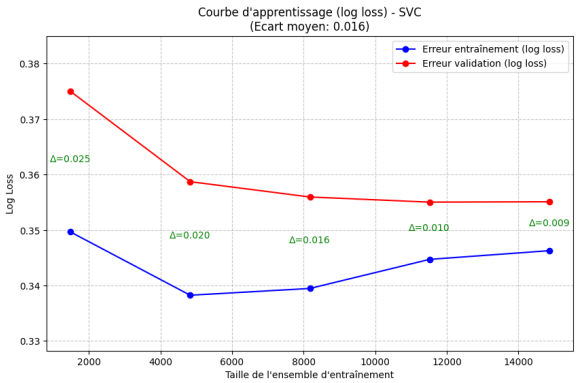
En analysant les courbe, les modèles présentent des comportements distincts face au sur-apprentissage ; la Régression Logistique se distingue avec un écart final d'environ 0.001 entre les courbes d'entraînement et de validation (et un écart moyen faible de 0.007), ce qui est très correct et indique un faible sur-apprentissage, suggérant qu'elle apprend de manière stable, elle pourrait être considérée comme la "meilleure" en termes d'équilibre généralisation/ajustement simple. À l'opposé, des modèles comme XGBoost affichent un écart important entre les pertes d'entraînement et de validation, confirmant qu'il est en sur-apprentissage.



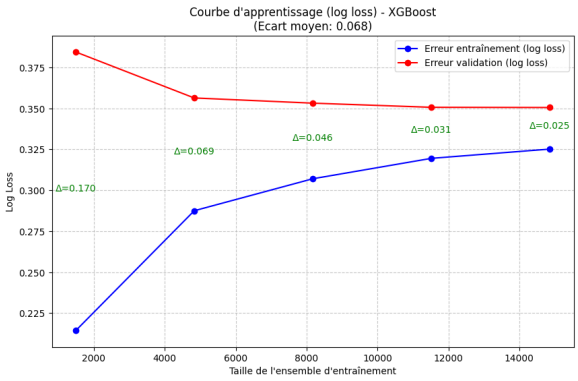
Logistic Regression



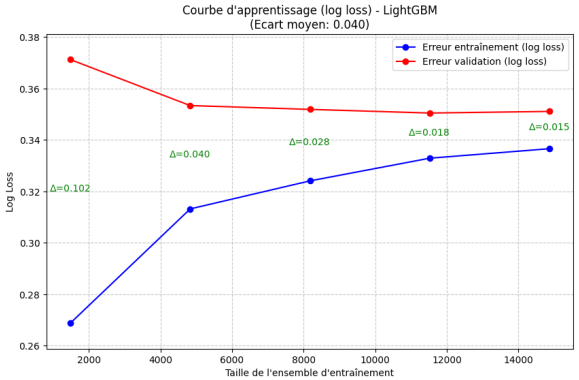
Random Forest



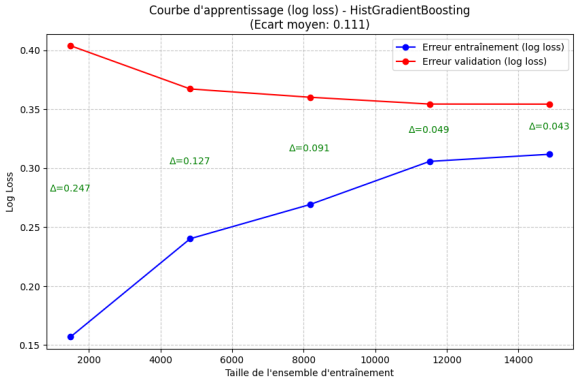
SVC



XGBoost



LightGBM



HistGradientBoosting

FIGURE 1.7 – Courbes d'apprentissage pour les différents modèles d'entraînement

1.6.5 Comparaison globale des modèles

Dans cette étude comparative, plusieurs modèles de classification ont été évalués pour détecter les cas de dépression, en combinant un prétraitement de données, recherche d'hyperparamètres et l'entraînement avec de différents modèles.

Globalement, les méthodes de boosting telles que *XGBoost, LightGBM et HistGradientBoosting* se démarquent nettement. *XGBoost* offre le meilleur compromis entre l'accuracy (0.8558), faible taux de faux négatifs et robustesse, ce qui en fait un choix privilégié dans un contexte clinique. *HistGradientBoosting*, avec une accuracy de 0.8643 et un temps d'entraînement réduit (14.71 s), représente une alternative très efficace. *LightGBM* affiche également de bonnes performances, bien qu'un peu en retrait sur l'accuracy.

À l'inverse, le *SVC*, malgré une précision correcte, souffre de temps d'entraînement extrêmement longs (338 s) et d'un taux de faux négatifs plus élevé, ce qui le rend moins adapté à ce type d'application. *Logistic Regression* et *Random forest* fournissent des résultats corrects, rapides et interprétables, mais restent moins performantes face aux algorithmes de boosting.

Ainsi, pour des applications sensibles comme le dépistage de la dépression, les méthodes de boosting sont les plus recommandées.

1.7 Perspectives futures

- **Enrichissement des données :** Intégrer davantage de variables cliniques, comportementales ou issues de capteurs (données physiologiques, historiques médicaux, etc.) pourrait améliorer la précision et la sensibilité des modèles.
- **Optimisation des hyperparamètres :** Bien que GridSearchCV ait permis d'optimiser les modèles, l'utilisation d'autres méthodes d'optimisation pourrait aboutir à de meilleurs réglages.
- **Méthodes d'apprentissage automatique plus avancées :** Tester des architectures de deep learning comme les réseaux de neurones profonds pourrait conduire à des gains de performance, notamment si les données deviennent plus complexes ou volumineuses.
- **Optimisation en contexte réel : :** Une prochaine étape consisterait à intégrer ces modèles dans une application clinique (ou mobile) avec des tests sur des données en temps réel, pour évaluer la robustesse et l'impact dans des conditions d'utilisation concrètes.

1.8 Conclusion

Dans cette étude, nous avons mené une analyse comparative de plusieurs algorithmes de classification dans le but de détecter les cas de dépression à partir de données prétraitées. En appliquant des techniques de normalisation, d'encodage, ainsi qu'une recherche d'hyperparamètres via GridSearchCV avec validation croisée, chaque modèle a été évalué selon des critères à la fois quantitatifs (accuracy, F1-score, AUC) et cliniquement pertinents.

Les résultats obtenus mettent en lumière la supériorité des méthodes de boosting telles que XGBoost, LightGBM qui offrent un bon compromis entre précision, robustesse et temps d'entraînement. En particulier, XGBoost se distingue comme le modèle le plus équilibré, tandis que HistGradientBoosting combine rapidité et haute performance.

Cette étude souligne également l'importance de choisir des modèles adaptés aux enjeux réels de l'application, notamment dans un contexte médical où les erreurs de classification peuvent avoir des conséquences critiques.

En conclusion, les approches basées sur le boosting apparaissent comme les plus appropriées pour une tâche aussi sensible que le dépistage automatisé de la dépression, alliant précision, robustesse et potentiel d'implémentation dans des environnements réels.

Chapitre 2
Régression

2.1 Introduction

La régression est une méthode statistique fondamentale utilisée pour modéliser et analyser les relations entre une variable dépendante (ou variable à expliquer) et une ou plusieurs variables indépendantes (ou variables explicatives). Elle permet de prédire des valeurs continues, d'identifier des tendances et de comprendre l'impact des différents facteurs sur un phénomène observé.

Parmi les types de régression les plus courants, on trouve la régression linéaire simple (pour une seule variable explicative) et la régression linéaire multiple (pour plusieurs variables). D'autres modèles plus avancés, comme la régression polynomiale, la régression logistique (pour les problèmes de classification) ou les méthodes de régularisation (comme la régression Ridge et Lasso), étendent ses applications à des cas plus complexes.

Que ce soit en économétrie, en machine learning, en biologie ou en sciences sociales, la régression reste un outil indispensable pour l'analyse prédictive et la prise de décision basée sur les données. Dans ce contexte, nous explorerons ses principes, ses hypothèses et ses applications pratiques.

2.2 Objectifs de la Régression

L'objectif de la régression dans ce projet est de prédire le prix des ordinateurs en fonction de différentes caractéristiques, telles que la mémoire RAM, la capacité du disque dur, la marque, le processeur ou d'autres spécifications techniques. En analysant ces variables, le modèle de régression permettra d'identifier les facteurs qui influencent le plus le prix et d'établir une relation mathématique pour estimer le coût d'un ordinateur en fonction de ses composants. Cette prédiction peut aider les acheteurs à évaluer le rapport qualité-prix, les fabricants à fixer des tarifs compétitifs ou les revendeurs à optimiser leur stratégie de pricing. Grâce à des algorithmes comme la régression linéaire, polynomiale ou par forêts aléatoires, nous pouvons construire un modèle précis et interprétable pour anticiper les prix du marché.

2.3 Présentation du Dataset

Notre dataset contient des informations sur différentes configurations d'ordinateurs, avec leurs caractéristiques techniques et leurs prix. L'objectif est d'analyser ces données pour comprendre quels facteurs influencent le coût des machines et construire un modèle de prédiction

fiable.

2.3.1 Description détaillée du dataset

- **Nom :** " Laptop Price Prediction using specifications "
- **Source :** Le dataset est obtenu via Kaggle
- **Définition du problème :** Ce dataset contient des informations détaillées sur différentes configurations d'ordinateurs avec leurs prix en Roupie Indienne (INR). L'objectif principal est d'analyser l'impact des composants matériels sur le prix.
- **Contenu :** On a un ensenble de **1302** instances sans prétraitement.
- **Description des données :** Ce dataset contient 13 colonnes distinctes, voici quelques colonnes phares :

TABLE 2.1 – Description des caractéristiques du dataset des ordinateurs

Colonne	Type	Description
Manufacturer	object	Fabricant de l'ordinateur (Dell, HP, Lenovo, etc.)
Model Name	object	Nom du modèle spécifique du produit
Category	object	Type d'ordinateur (Portable, Bureau, Gaming, etc.)
Screen Size	object	Taille de l'écran en pouces (ex : "15.6")
Screen	object	Détails de l'affichage (Résolution, technologie : Full HD, IPS, etc.)
CPU	object	Processeur (ex : "Intel Core i7-1165G7", "AMD Ryzen 5 5600U")
RAM	object	Mémoire RAM (ex : "8GB", "16GB DDR4")
Storage	object	Stockage (ex : "512GB SSD", "1TB HDD + 256GB SSD")
GPU	object	Carte graphique (ex : "NVIDIA GeForce RTX 3050", "Intel Iris Xe")
Operating System	object	Système d'exploitation principal (Windows, Linux, macOS)
Operating System Version	object	Version du système d'exploitation (ex : "Windows 11 Home", "macOS Monterey")
Weight	object	Poids de l'appareil (ex : "1.8 kg", "3.5 lbs")
Price	float	Prix en INR (Roupie indienne)

2.3.2 Justification du choix

1. **Données Complètes et Exploitables :** Nous avons choisi ce dataset de 1 302 ordinateurs pour sa richesse en caractéristiques techniques (CPU, RAM, GPU, stockage) et sa pertinence pour la prédiction de prix. Avec 13 variables explicatives claires et une variable cible continue (prix en INR).
2. **Idéal pour la Prédiction de Prix (Régression)**
   — Variable cible claire : Prix en INR (nombre continu, parfait pour la régression).
   — Relations évidentes : Par exemple, plus la RAM ou le GPU est performant, plus le prix augmente.

3. **Source Fiable et Facile d'Accès**
   — Kaggle : Plateforme reconnue pour les datasets de qualité.
   — Format propre : CSV prêt à l'analyse, sans nettoyage complexe nécessaire.
4. **Applications Concrètes**
   — Aide aux acheteurs à estimer un prix juste.
   — Permet aux vendeurs de fixer des tarifs compétitifs.

2.4 Prétraitement des Données

2.4.1 Visualisation des données

La distribution des prix montre une asymétrie marquée vers les valeurs basses, avec la majorité des ordinateurs concentrés dans les gammes de prix les plus accessibles.

On observe également plusieurs valeurs extrêmes significativement plus élevées que le reste des données, ce qui indique la présence d'outliers dans les hautes gammes de prix.

L'étendue importante des prix et la distribution inégale des observations à travers les différentes gammes de valeurs sont des caractéristiques notables de ce jeu de données.

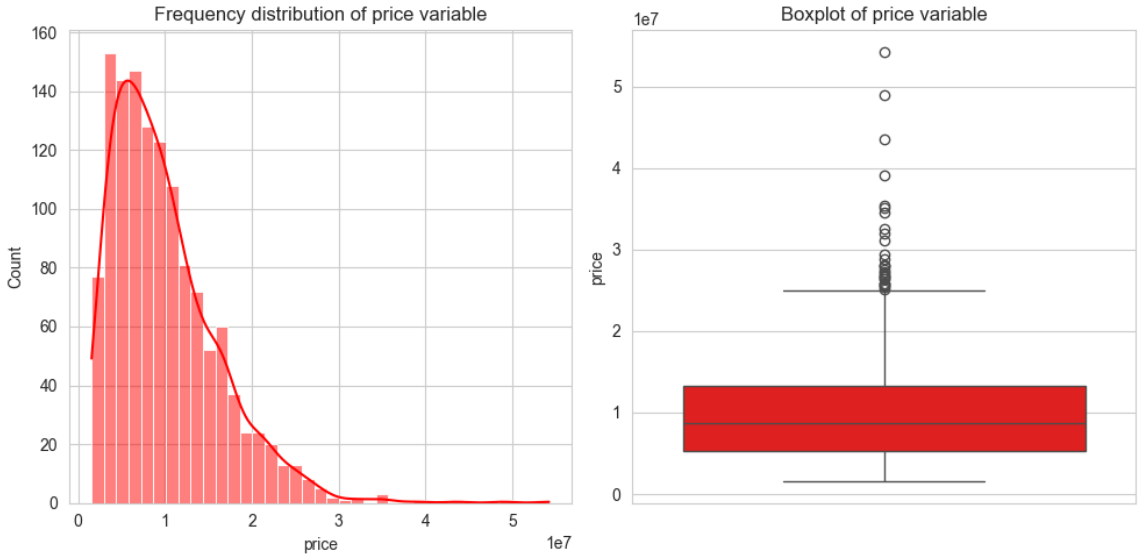


FIGURE 2.1 – Distribution de la variable cible

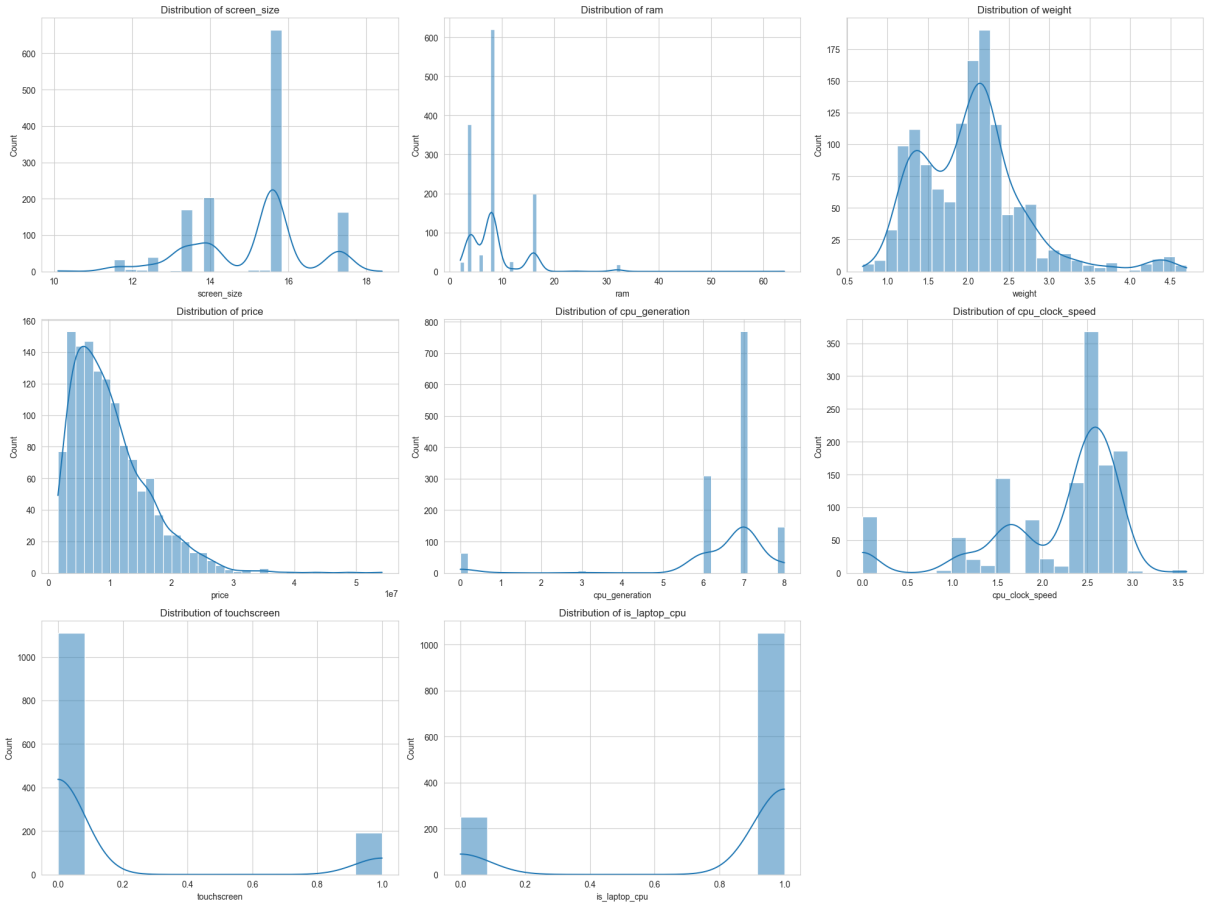


FIGURE 2.2 – Histogrammes des valeurs numériques

2.4.2 Répartition des variables catégorielles importantes

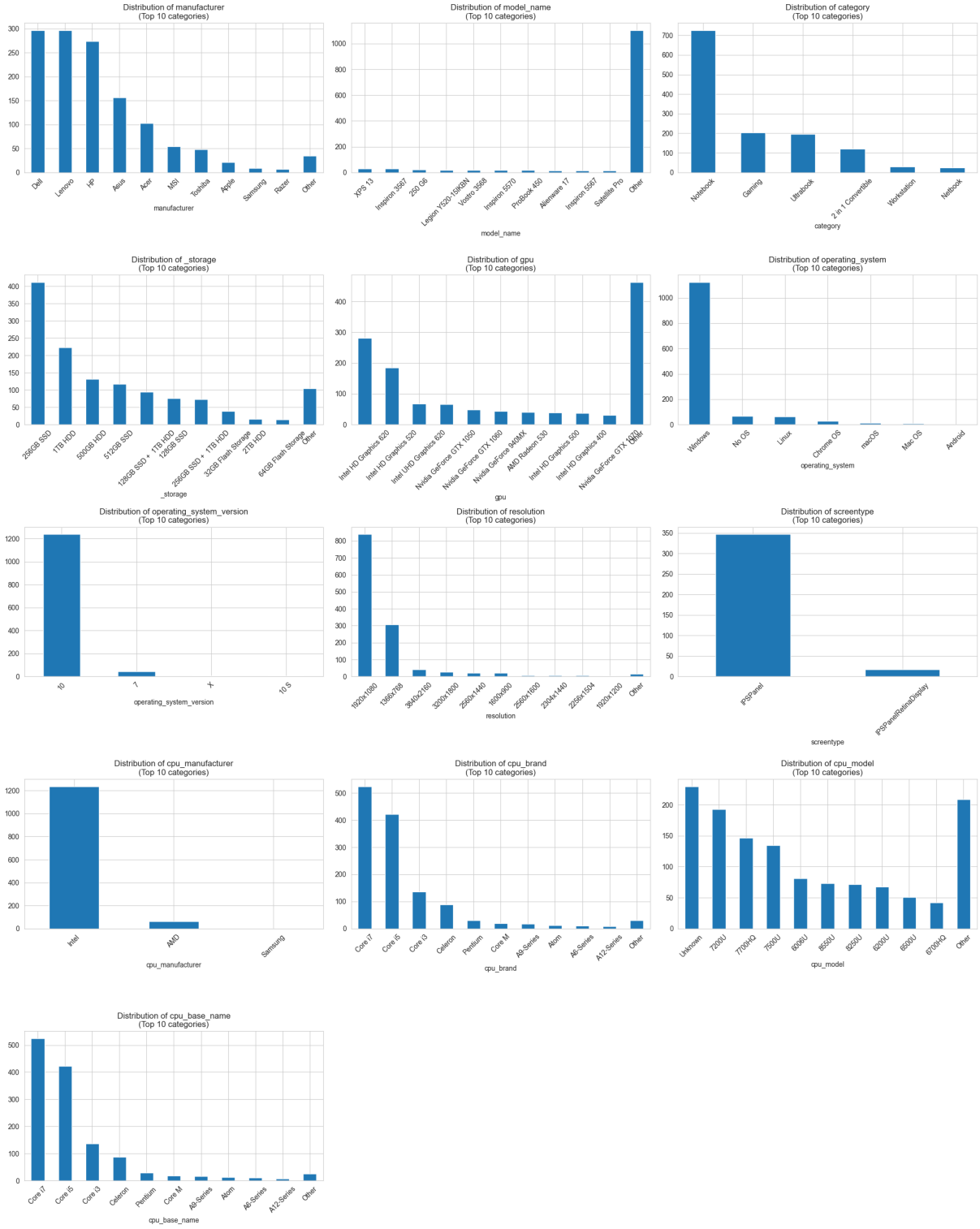


FIGURE 2.3 – Distribution des variables catégorielles

2.4.3 Matrice de corrélation

Points clés de la matrice de corrélation :

- **RAM-Prix (0.74) :** La corrélation la plus forte, indiquant que la mémoire est le facteur principal influençant le prix.
- Pour prédire le prix, la RAM et les caractéristiques CPU sont plus pertinentes que la taille d'écran ou l'écran tactile. La forte corrélation écran-poids pourrait justifier le retrait d'une de ces variables pour éviter la redondance.
- La génération du CPU (r= -0.091) et l'écran tactile (r= -0.3) montrent une corrélation négligeable avec le prix, indiquant que : L'ancienneté du processeur (génération) n'influence pas significativement le coût et La technologie tactile n'ajoute pas de valeur perçue justifiant une augmentation de prix.

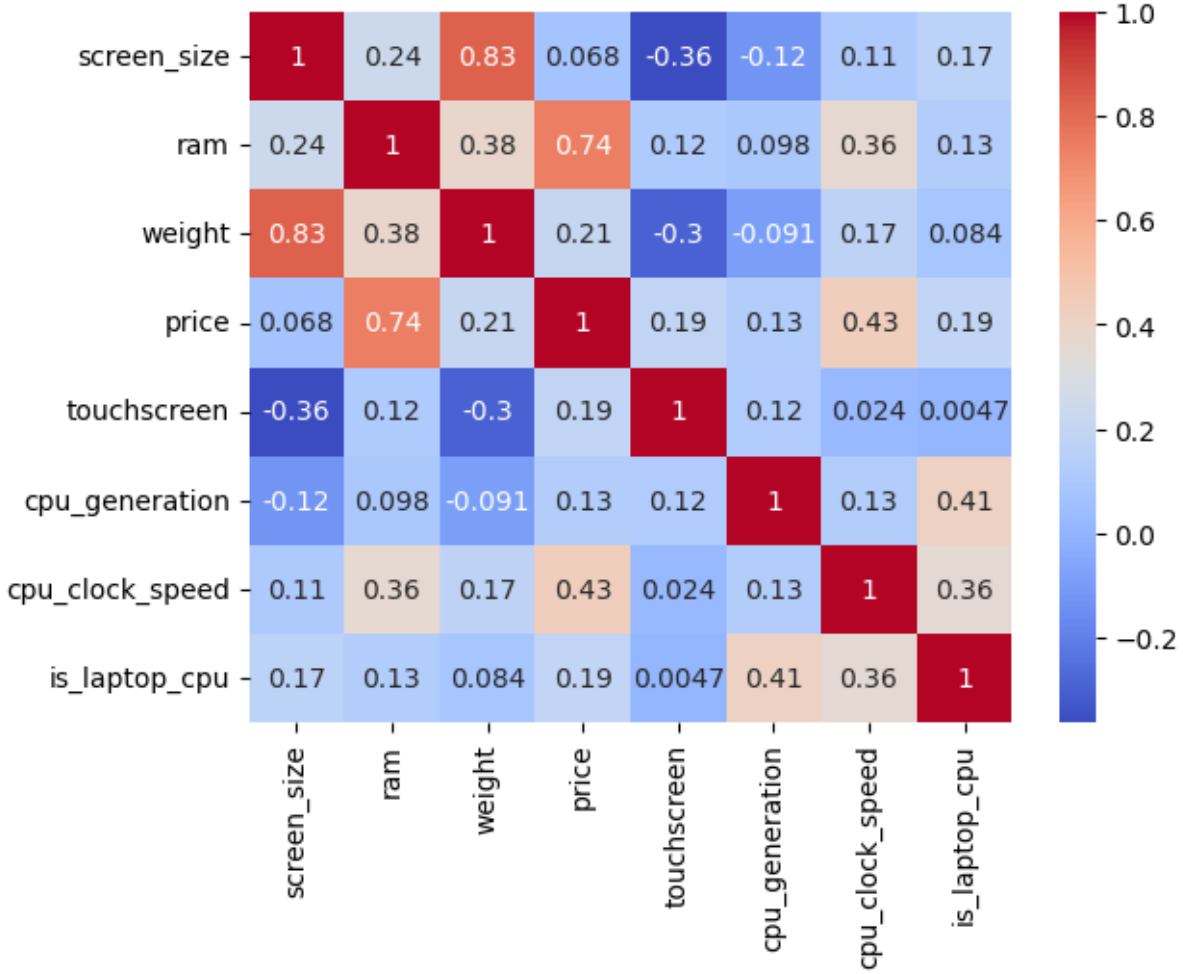


FIGURE 2.4 – Matrice de corrélation

2.4.4 Nettoyage de données

Élimination des Caractères d'Unité dans les Colonnes de Données :

Pour garantir l'exploitabilité des données dans les analyses, un traitement préalable a été effectué sur les colonnes screen_size (taille d'écran) et ram (mémoire vive) du DataFrame à l'aide de la bibliothèque pandas en Python. Ce processus, détaillé ci-dessous, vise à supprimer les caractères non numériques et à convertir les valeurs en types numériques adaptés.

```
# Suppression du caractere '"'
df['screen_size'] = df['screen_size'].str.replace('"', '')

# Conversion en type float
df['screen_size'] = df['screen_size'].astype(float)
```

Listing 2.1 – Nettoyage et conversion des données de taille d'écran

```
# Suppression des caracteres 'GB'
df['ram'] = df['ram'].str.replace('GB', '')

# Conversion en type entier
df['ram'] = df['ram'].astype(int)
```

Listing 2.2 – Nettoyage et conversion des données de RAM

```
# Suppression des caracteres 'kg'
df['weight'] = df['weight'].str.replace('kg', '')
```

Listing 2.3 – Nettoyage des données de poids

Nettoyage des Données de Poids et Suppression des Valeurs Aberrantes :

Pour garantir l'intégrité des données, un traitement spécifique a été appliqué à la colonne *weight* (poids) du DataFrame à l'aide de la bibliothèque pandas en Python. Ce processus vise à identifier et supprimer une valeur aberrante non numérique, puis à convertir les valeurs en type numérique adapté.

```
# Recherche et identification de la valeur aberrante
not_weight_index = df.query('weight == "4s"').index[0]
print("Index de la ligne avec '4s' dans la colonne weight:", not_weight_index)

# Suppression de cette ligne specifique
df = df.drop(not_weight_index, axis=0)

# Conversion finale en type float
df['weight'] = df['weight'].astype(float)
```

Listing 2.4 – Nettoyage des données de poids et suppression des valeurs aberrantes

2.4.5 Détection et Suppression des Valeurs Aberrantes dans la Colonne Prix

Pour améliorer la qualité du jeu de données et réduire l'impact des valeurs extrêmes sur les analyses ou les modèles prédictifs, les valeurs aberrantes (outliers) dans la colonne price ont été identifiées et supprimées à l'aide de la méthode de l'écart interquartile (IQR) avec la bibliothèque pandas en Python.

```
# Calcul des quartiles et de l'IQR
Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
IQR = Q3 - Q1

# Definition des bornes pour les outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
    # Filtrage du dataframe
    df_clean = df[(df['price'] >= lower_bound) &
                  (df['price'] <= upper_bound)]

    print(f"Nombre d'outliers supprimes : {len(df) - len(df_clean)}")
    print(f"Pourcentage de donnees conservees : {len(df_clean)/len(df):.1%}")
```

Listing 2.5 – Détection et suppression des outliers par méthode IQR

Décomposition des Caractéristiques Composites de l'Écran et du Processeur :

Les colonnes screen (écran) et cpu (processeur) du DataFrame contiennent des informations composites sous forme de chaînes de caractères complexes (par exemple, résolution, type d'écran, fabricant du processeur, fréquence). Pour rendre ces données exploitables dans des analyses ou des modèles, deux fonctions Python ont été développées à l'aide de la bibliothèque pandas et du module re (expressions régulières). Ces fonctions décomposent les informations en colonnes distinctes, facilitant leur utilisation.

```
    def extract_screen_features(train):
        """Normalise et extrait les caracteristiques techniques de l'ecran"""

        # Normalisation des noms de colonnes
        train = train.rename(columns=str.lower)

        # Extraction de la resolution (format NxN)
        train["resolution"] = train["screen"].str.extract(r'(\d+x\d+)')

        # Extraction du type d'ecran (nettoyage progressif)
        train["screentype"] = train["screen"].replace(r'(\d+x\d+)','', regex=True)
        train['screentype'] = train['screentype'].replace(
            r'(Full HD|Quad HD|\+|/|4K Ultra HD)', '', regex=True)

        # Detection des ecrans tactiles
        train['touchscreen'] = train['screentype'].str.extract(r'(Touchscreen)')
        train['screentype'] = train['screentype'].replace(r'(Touchscreen)','', regex=True)
        train['touchscreen'] = train['touchscreen'].replace('Touchscreen', 1)
        train['touchscreen'] = train['touchscreen'].replace(np.nan, 0)

        # Nettoyage final
        train['screentype'] = train['screentype'].replace(r' ','', regex=True)
        train['screentype'] = train['screentype'].replace(r'^\s*$', np.nan, regex=True)

        # Suppression de la colonne originale
        train = train.drop("screen", axis=1)

        return train
```

Listing 2.6 – Extraction des caractéristiques d'écran

```
    def extract_cpu_features(cpu_strings):
        """Extrait les caracteristiques techniques des processeurs a partir de strings
    descriptifs"""
        data = []
        for cpu in cpu_strings:
            # Valeurs par defaut
            entry = {
                'cpu_manufacturer': None,
                'cpu_brand': None,
                'cpu_model': None,
                'cpu_generation': None,
                'cpu_clock_speed': None,
                'is_laptop_cpu': 0,
                'cpu_base_name': None
            }

            # Extraction du fabricant (Intel/AMD/Samsung)
            if 'Intel' in cpu:
                entry['cpu_manufacturer'] = 'Intel'
            elif 'AMD' in cpu:
                entry['cpu_manufacturer'] = 'AMD'
            elif 'Samsung' in cpu:
                entry['cpu_manufacturer'] = 'Samsung'

            # Extraction de la frequence (GHz)
            speed_match = re.search(r'(\d+\.\d+)GHz', cpu)
            if speed_match:
                entry['cpu_clock_speed'] = float(speed_match.group(1))

            # Traitement specifique Intel
            if entry['cpu_manufacturer'] == 'Intel':
                brand_match = re.search(r'(Core i[3579]|Xeon|Celeron|Pentium|Atom|Core M)'
    , cpu)
                if brand_match:
                    entry['cpu_brand'] = brand_match.group(0)

                model_match = re.search(r'(?:\s|[-/])(\d{4}[A-Za-z]*)', cpu)
                if model_match:
                    entry['cpu_model'] = model_match.group(1)
                    entry['cpu_generation'] = int(entry['cpu_model'][0]) if entry['
    cpu_model'][0].isdigit() else None

                    if any(suffix in entry['cpu_model'] for suffix in ['U', 'HQ', 'Y']):
                        entry['is_laptop_cpu'] = 1

                entry['cpu_base_name'] = entry['cpu_brand']

            # Traitement specifique AMD
            elif entry['cpu_manufacturer'] == 'AMD':
                series_match = re.search(r'(Ryzen|A[0-9]|E[0-9]|FX)\s?[\w-]*', cpu)
                if series_match:
                    entry['cpu_brand'] = series_match.group(0).strip()
                    entry['cpu_base_name'] = entry['cpu_brand'].split()[0] if ' ' in entry
    ['cpu_brand'] else entry['cpu_brand']

            data.append(entry)

        return pd.DataFrame(data)
```

Listing 2.7 – Extraction des caracteristiques CPU

Vérification et Imputation des Valeurs Manquantes :

Pour assurer la qualité des données, des étapes de vérification et de traitement des valeurs manquantes ont été appliquées au DataFrame. Ces opérations permettent d'identifier les données manquantes et de les imputer de manière appropriée afin de garantir l'exploitabilité du jeu de données.

Par ailleurs, nous avons analysé la matrice de corrélation et supprimé les colonnes présentant à la fois :

- Une faible corrélation avec la variable cible (le prix)
- Un taux élevé de valeurs manquantes

```
# Verification des valeurs manquantes
print("Valeurs manquantes :\n", df.isnull().sum())
missing_percent = (df.isnull().sum() / len(df)) * 100
missing_report = missing_percent[missing_percent > 0].sort_values(ascending=False)

print("Colonnes avec valeurs manquantes (%):\n", missing_report)
```

Listing 2.8 – Vérification des valeurs manquantes

Encodage

On applique deux méthodes d'encodage pour transformer des variables catégorielles en données numériques exploitables par des algorithmes de machine learning. La première méthode utilise l'encodage One-Hot, qui crée une colonne binaire pour chaque modalité (sauf la première, grâce à `drop_first=True`) et affiche la dimension finale du DataFrame après transformation.

La seconde méthode utilise LabelEncoder pour attribuer un entier unique à chaque catégorie de chaque colonne listée dans cols. Un aperçu des résultats encodés est ensuite affiché pour les trois premières colonnes, permettant de vérifier l'effet de l'encodage.

```
# Caleur a encoder
cols = [
    'manufacturer', 'model_name', 'category', 'screentype',
    'cpu_manufacturer', 'resolution', 'cpu_base_name',
    'cpu_model', 'cpu_brand', '_storage', 'gpu',
    'operating_system', 'operating_system_version'
]

# Encodage One-Hot (pour comparaison)
print('Dimension apres One-Hot Encoding : ',
      pd.get_dummies(df, columns=cols, drop_first=True).shape)

# Encodage par LabelEncoder
en = LabelEncoder()
for col in cols:
    df[col] = en.fit_transform(df[col])

# Verification
print("\nExemple de valeurs encodees:")
print(df[cols[:3]].head())  #Affiche les 3 premieres colonnes
```

Listing 2.9 – Encodage des variables

2.5 Modélisation

2.5.1 Algorithmes de régression utilisés

Pour ce projet, plusieurs algorithmes de classification ont été utilisés afin de comparer leurs performances et sélectionner le plus adapté au problème de prédiction :

- **Régression Logistique (Logistic Regression)**
- **Random Forest (RandomForestClassifier)**
- **Support Vector Regression (SVR)**
- **XGBoost (XGBClassifier)**
- **Réseaux de neurones (RNN) :** est un type de réseau de neurones conçu pour traiter des données séquentielles (comme du texte, de l'audio ou des séries temporelles). Contrairement aux réseaux classiques, les RNN ont des connexions récurrentes qui leur permettent de mémoriser des informations sur les états précédents, ce qui les rend utiles pour modéliser des dépendances temporelles.
- **ElasticNet :** ElasticNet est une méthode de régression qui combine les pénalités de Ridge (L2) et Lasso (L1). Elle permet à la fois de réduire le surapprentissage et de sélectionner les variables importantes, surtout utile quand les variables sont nombreuses et corrélées.

2.5.2 Optimisation du XGBoost

Pour optimiser le modèle de *XGBoost*, nous allons utiliser un **GridSearchCV** sur quelques-uns de ces hyperparamètres :

```
params = {
    'n_estimators': [100, 200],        # Nombre d'arbres
    'max_depth': [3, 5],               # Profondeur max des arbres
    'learning_rate': [0.05, 0.1],      # Taux d'apprentissage
    'subsample': [0.8, 0.9]            # Fraction d'echantillons utilises
}
```

Listing 2.10 – Dictionnaire d'hyperparamètres pour XGBoost

Les meilleurs hypers paramètres pour notre XGBoost sont donc :

```
Best Parameters: {
    'learning_rate': 0.05,
    'max_depth': 5,
    'n_estimators': 200,
    'subsample': 0.8
}
```

Listing 2.11 – Meilleurs hyperparamètres obtenus par optimisation

Après l'optimisation des paramètres de XGBoost, nous remarquons une amélioration des résultats dont une baisse de la RMSE ainsi qu'une légère augmentation du score R^2 montrés dans la figure 2.5.

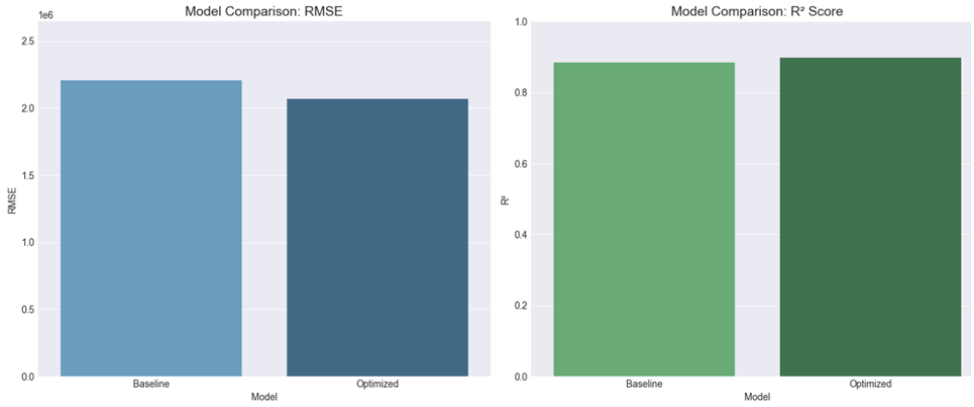


FIGURE 2.5 – Améliorations des métriques de XGBoost après optimisation

2.6 Analyse des résultats

2.6.1 Métriques d'évaluation

Mean Absolute Error (MAE)

Moyenne des écarts absolus entre les valeurs réelles et les valeurs prédites.
→ Mesure l'erreur moyenne sans tenir compte du sens de l'erreur.

Mean Squared Error (MSE)

Moyenne des carrés des écarts entre les valeurs réelles et prédites.
→ Pénalise davantage les grandes erreurs que le MAE.

R^2 Score

Proportion de la variance des données expliquée par le modèle.
→ Plus il est proche de 1, meilleur est le modèle.

Modèle	MAE	RMSE	R^2
XGBoost	0.134916	0.186896	0.912044
Random Forest	0.139180	0.194342	0.904896
SVR	0.174160	0.241314	0.853367
Linear Regression	0.256898	0.328414	0.728412
RNN	0.423782	0.531843	0.287747
ElasticNet	0.513001	0.630534	-0.001113

TABLE 2.2 – Comparaison des performances des modèles de Machine Learning

Les résultats du tableau 2.2 montrent des performances très variables selon les modèles testés. *XGBoost* et *Random Forest* obtiennent les meilleurs scores, avec des valeurs de MAE (0.135 et 0.139) et RMSE (0.187 et 0.194) faibles, ainsi qu'un R^2 élevé (0.912 et 0.905), indiquant une capacité à expliquer la variance des données.

Le *SVR* et *Linear regression* présente des performances légèrement inférieures mais reste acceptable.

En revanche, le *RNN* et *ElasticNet* affichent des résultats médiocres, particulièrement *ElasticNet* dont le R^2 négatif (-0.001) suggère que le modèle est inadapté aux données. Enfin, Basilabular (probablement une coquille pour ElasticNet ou RNN d'après le tableau) apparaît comme le moins performant, avec des barres quasi nulles pour le R^2, ce qui corrobore l'idée d'un modèle inadapté.

La visualisation en 2.6 renforce l'analyse : les modèles basés sur les arbres (XGBoost, Random Forest) sont optimaux pour ces données, tandis que les approches linéaires ou plus simples échouent à capturer la complexité du problème.

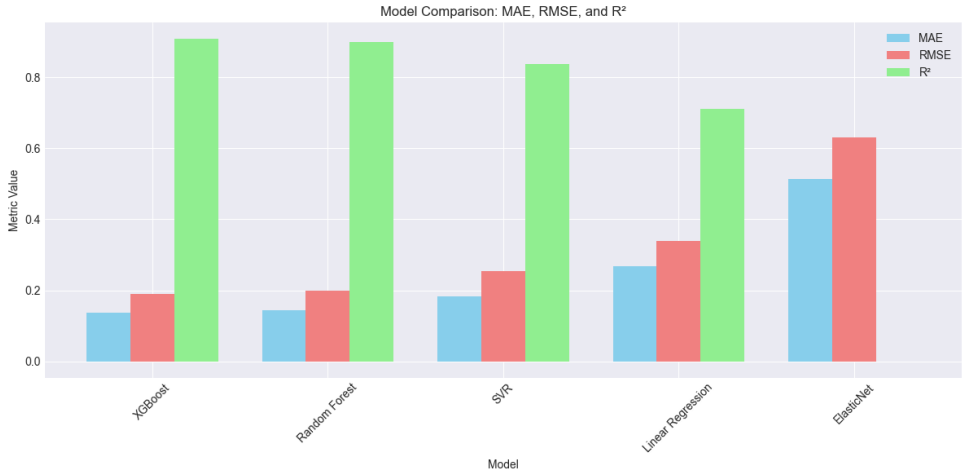


FIGURE 2.6 – Comparaison des performance via les metriques dévaluation

2.6.2 Comparaison des Performances de modèles pour la prédiction de valeurs

Les graphiques montrent une comparaison entre les valeurs réelles et prédites pour cinq modèles de régression : **Régression Linéaire**, **Random Forest**, **XGBoost**, **SVR** et **ElasticNet**.

Random Forest ($R^2 = 0{,}989$) et **XGBoost** ($R^2 = 0{,}909$) affichent les meilleures performances, avec des prédictions très proches des valeurs réelles, comme en témoigne leur forte corrélation et leur proximité avec la ligne diagonale ($y = x$).

SVR ($R^2 = 0{,}837$) performe également bien, mais montre un peu plus de dispersion.

Régression Linéaire ($R^2 = 0{,}712$) présente une corrélation modérée, avec des prédictions moins précises, surtout pour les valeurs extrêmes.

ElasticNet ($R^2 = 0{,}001$) échoue complètement, avec des prédictions constantes ($y \approx 15{,}94$), indiquant qu'il ne capture aucune relation dans les données.

En résumé, **Random Forest** et **XGBoost** sont les plus performants, tandis qu'**ElasticNet** est inadapté pour ce problème.

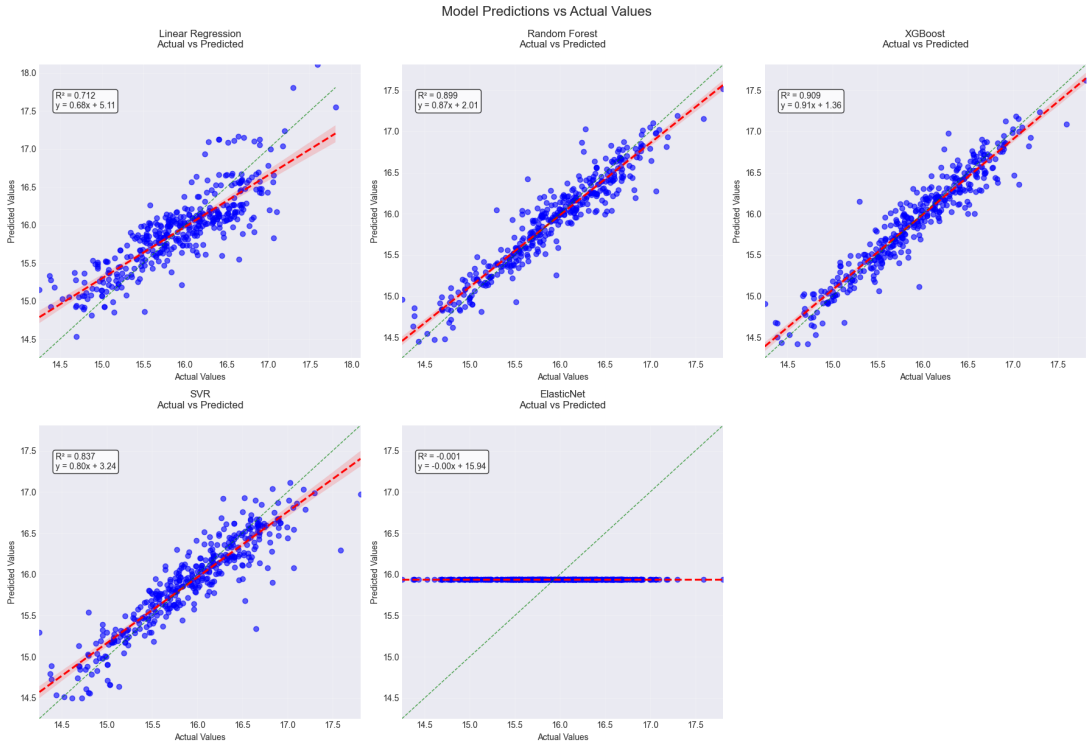


FIGURE 2.7 – Enter Caption

2.6.3 Caractéristiques importantes du XGBoost

Les caractéristiques les plus importantes selon la figure 2.8 pour la prédiction sont :

1. Modèle du *laptop*
2. Poids
3. Mémoire RAM
4. Type de stockage

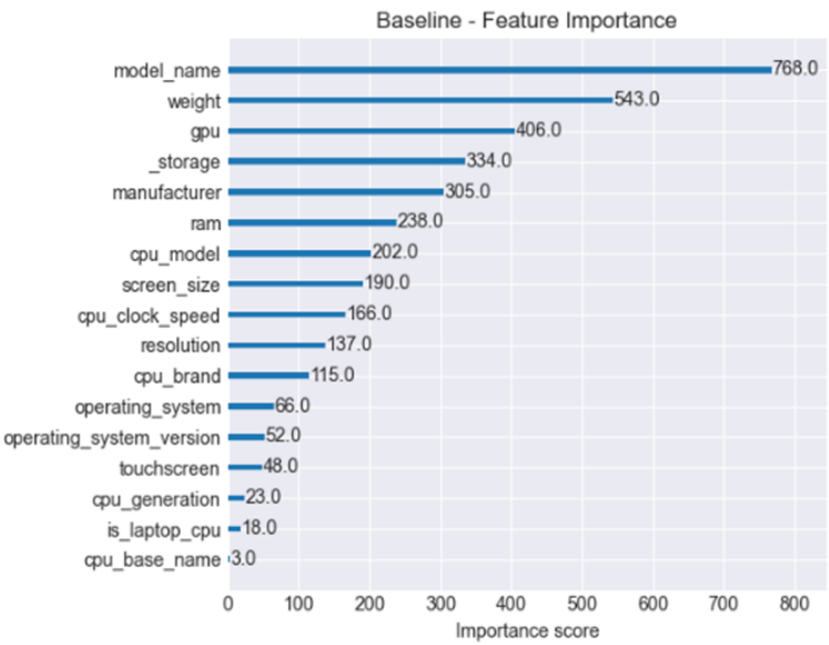


FIGURE 2.8 – Caractéristiques les plus importantes du XGBoost

2.6.4 Analyse et Comparaison globale

Les écarts trouvés dans la partie des métriques d'évaluation soulignent la supériorité des méthodes ensemblistes (*XGBoost, Random Forest*) pour cette tâche, tandis que les approches linéaires simples ou mal ajustées peinent à capturer les relations complexes dans les données, de ce là on s'est intéressé plus au XGBoost.

XGBoost s'est révélé être le modèle le plus performant pour cette tâche de prédiction de prix. On peut recommander les choses suivants :

1. Enrichir le jeu de données avec plus de modèles récents
2. Explorer des techniques d'optimisation hyper-paramétrique plus poussées

Présentons quelques avantages et inconvénients pour XGBoost :
- **Avantages** : Meilleure précision, capture bien les relations non-linéaires
- **Inconvénients** : Temps d'entraînement plus long, plus complexe

2.7 Limites rencontrées

- Données limitées à certains fabricants.
- Prix non actualisés dans le temps.
- Certaines caractéristiques techniques sont difficiles à standardiser.

2.8 Perspectives

Pour améliorer d'avantage le modèle :
- Intégrer des données de marché en temps réel
- Ajouter des caractéristiques comme les avis clients
- Tester des architectures de *deep learning*

2.9 Conclusion

Dans cette partie du projet, on compare plusieurs modèles de régression pour la prédiction de prix, en évaluant leurs performances à l'aide du MAE, du RMSE et du R^2. Les résultats montrent que les méthodes ensemblistes, notamment XGBoost et Random Forest, sont nettement supérieures, avec des scores élevés en précision et en capacité explicative. À l'inverse, des modèles comme ElasticNet et RNN s'avèrent peu performants.

XGBoost ressort comme le meilleur modèle, notamment grâce à sa capacité à capturer des relations non linéaires complexes. Les variables les plus influentes identifiées sont le modèle, le poids, la RAM et le type de stockage. L'analyse souligne cependant certaines limites (jeu de données restreint, informations techniques incomplètes) et propose d'enrichir les données et d'optimiser davantage les hyperparamètres à l'avenir.