PROJET AI28 - MACHINE LEARNING

Alexandre Bidaux
GI04
alexandre.bidaux@etu.utc.fr

Julie Chartier
GI04
julie.chartier@etu.utc.fr

Quentin Valakou
GI04
quentin.valakou@etu.utc.fr

ABSTRACT

Dans le cadre du projet de Machine Learning de l'UV AI28 à l'Université de Technologie de Compiègne, nous nous intéressons à la prédiction des revenus individuels à partir du jeu de données "Adult" du UC Irvine Machine Learning Repository. Ce dataset contient 48 842 instances et 14 attributs décrivant des caractéristiques socio-économiques. L'objectif est de classifier si un individu gagne plus ou moins de 50 000$ par an. Après une analyse exploratoire des données, nous avons testé plusieurs modèles supervisés, notamment la régression logistique, les arbres de décision et les forêts aléatoires. Nos résultats montrent que le modèle Extreme Gradient Boosting obtient la meilleure performance avec une précision de 87% sur les données de test. Ces résultats soulignent l'intérêt d'une bonne préparation des données et d'un choix judicieux de modèle pour ce type de tâche prédictive.

1 INTRODUCTION

Dans le cadre du projet de Machine Learning de l'UV AI28 dispensée à l'Université de Technologie de Compiègne par Elmokhtar ALAYA, nous utilisons les données "Adult" (ou "Census Income") du UC Irvine Machine Learning Repository datant de 1994. Dans ce travail, nous cherchons à prédire si le revenu annuel d'un individu est supérieur à 50000$ par an. Pour répondre à ce problème de classification, nous réalisons une analyse exploratoire des données (AED), un pré-traitement des données, une modélisation, une hyper-optimisation et une évaluation des modèles avant de conclure.

2 ANALYSE EXPLORATOIRE DES DONNÉES

2.1 JEU DE DONNÉES

Le jeu de données 'Adult Census Income'[1] contient cinq fichiers mais nous n'utiliserons principalement que les trois suivants.

- **adult.data** : jeu de données d'entrainement au format csv (séparateur de colonnes : la virgule)
- **adult.names** : fichier donnant du contexte et descriptions sur les données et les opérations ayant été réalisées dessus
- **adult.test** : jeu de données de test au format csv (séparateur de colonnes : la virgule)

Les données sont déjà séparées dans un jeu dédié à l'entrainement (adult.data) et dans un autre dédié au test du modèle (adult.test).

Le jeu de données contient 48842 instances (32561 instances pour l'entrainement et 16281 pour le test), 14 modalités et une variable cible. Afin de réaliser l'AED, nous concaténons les jeux d'entrainement et de test pour recréer un jeu de données complet.

[1] Source des données : https://archive.ics.uci.edu/dataset/2/adult

2.2 PROBLÈME

Dans ce travail, nous cherchons à prédire la variable *income*, c'est-à-dire si un individu a des revenus annuels inférieurs ou supérieurs à 50000$. Il s'agit ici d'un problème de classification, notre modèle devra prédire l'appartenance à une classe : 1 si l'individu gagne plus de 50000$ par an, 0 si l'individu gagne moins de 50000$ par an.

2.3 DESCRIPTION DES VARIABLES

2.3.1 VARIABLE CIBLE

La variable cible est celle que nous cherchons à prédire. Ici, elle est qualitative nominale (binaire), il s'agit de :

- **income** (variable cible) : si l'individu gagne ou non plus de 50000$ par an. (binaire, nous pré-traiterons les données de telle sorte que "0" vaut pour $\leq 50k$ et "1" pour $> 50k$).

2.3.2 VARIABLES QUANTITATIVES CONTINUES

- **age** : l'âge d'un individu (ici un entier supérieur à 0).
- **hours-per-week** : le nombre d'heures de travail par semaine rapporté par l'individu.

2.3.3 VARIABLES QUANTITATIVES DISCRÈTES

- **fnlwgt** : veut dire *final weight* (poids final en français), le nombre de personne que l'organisation pense que l'instance représente aux États-Unis.
- **capital-gain** : les gains de l'individu. (un entier supérieur ou égal à 0)
- **capital-loss** : les pertes de l'individu. (un entier supérieur ou égal à 0)

2.3.4 VARIABLES QUALITATIVES ORDINALES

- **education** : le niveau d'éducation de l'individu. (ordinale mais l'ordre n'est pas forcément évident)
  - 19 valeurs possibles : Bachelors, Some college, 11th, HS grad, Prof school, Assoc acdm, Assoc voc, 9th, 7th, 8th, 12th, Masters, 1st, 4th, 10th, Doctorate, 5th, 6th, Preschool.
- **education-num** : le niveau d'éducation dans sa forme numérique. (un entier supérieur à 0).

2.3.5 VARIABLES QUALITATIVES NOMINALES

- **workclass** : le statut d'emploi de l'individu.
  - 8 valeurs possibles : Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked
- **marital-status** : le statut marital de l'individu.
  - 7 valeurs possibles : Married civ spouse, Divorced, Never married, Separated, Widowed, Married spouse absent, Married AF spouse
- **occupation** : le type général de métier de l'individu.
  - 14 valeurs possibles : Tech support, Craft repair, Other service, Sales, Exec managerial, Prof specialty, Handlers cleaners, Machine op inspct, Adm clerical, Farming fishing, Transport moving, Priv house serv, Protective serv, Armed Forces
- **relationship** : ce que l'individu est par rapport à d'autres.
  - 6 valeurs possibles : Wife, Own child, Husband, Not in family, Other relative, etc.
- **race** : l'ethnie de l'individu.
  - 5 valeurs possibles : White, Asian Pac Islander, Amer Indian Eskimo, Other, Black

- **sex** : le sexe de l'individu. (nous pouvons la transformer en variable binaire car il n'y a que deux possibilités, "0" pour Female et "1" pour Male)
  - 2 valeurs possibles : Male, Female
- **native-country** : le pays de naissance de l'individu. (qualitative)
  - 42 valeurs possibles : United States, Cambodia, England, Puerto Rico, Canada, etc.

2.4 Valeurs manquantes

Le jeu de données contient des valeurs manquantes selon la description sur le site présentant la source de données. Nous confirmons cela dans notre analyse. En effet, les valeurs manquantes se situent dans les variables suivantes :

- **workclass** (1836 pour les données d'entrainement, 963 pour les données de test) : 5.7%
- **occupation** (1843 pour les données d'entrainement, 966 pour les données de test) : 5.8%
- **native-country** (583 pour les données d'entrainement, 274 pour les données de test) : 1.8%

Nous décidons de conserver ces trois variables car elles ont une proportion de valeurs manquantes inférieures à 6%, ce qui nous paraît correct car en-dessous de 10%.

2.5 Données dupliquées

Le jeu de données contient des données dupliquées. Des personnes ont exactement les mêmes profils. Cependant, la variable **fnlwgt** représentant le nombre de personnes que l'organisation pense que l'instance représente aux États-Unis, nous considérons que nous pouvons supprimer les duplicatas. Une fois les instances dupliquées supprimées, notre jeu de données contient 48789 instances. Il y en avait 53 de dupliquées.

2.6 Étude de la variable cible

Échantillon	$\leq 50k$	$> 50k$
Entrainement	24698	7839
Test	12429	3846
Total	37108 (76%)	11681 (24%)

Table 1: Distribution de la variable cible

Dans les résultats de la distribution de la variable cible (voir Table 1), nous remarquons que notre jeu de données est déséquilibré, c'est-à-dire qu'il contient une majorité de personnes ayant des revenus annuels inférieurs à 50000$ (76% de l'échantillon). Nous devrons prendre en compte cela dans la suite de l'étude.

2.7 Analyse univariée

Dans cette section, nous étudions chaque variable individuellement.

2.7.1 Variables catégorielles

Afin de récupérer plus facilement les différentes variables en fonction de leur nature, nous avons changé le type des variables quantitatives en *np.float64*. Les variables qualitatives sont donc facilement récupérables à l'aide des types *np.object* et *np.int64*.

Nous avons tracé des graphiques circulaires pour compter le nombre de chaque modalité pour chacune des variables qualitatives (voir Figure 1 ci-dessous puis Figure 7 et Figure 8 en Annexe A.1).

Nous observons que les variables *native-country* et *race* sont fortement déséquilibrées. Les variables *workclass*, *education*, *education-num*, *marital-status*, *relationship*, *sex* sont déséquilibrées. Aussi, la variable *sex* est binaire tandis que les autres ont toutes entre 5 et 42 valeurs possibles.

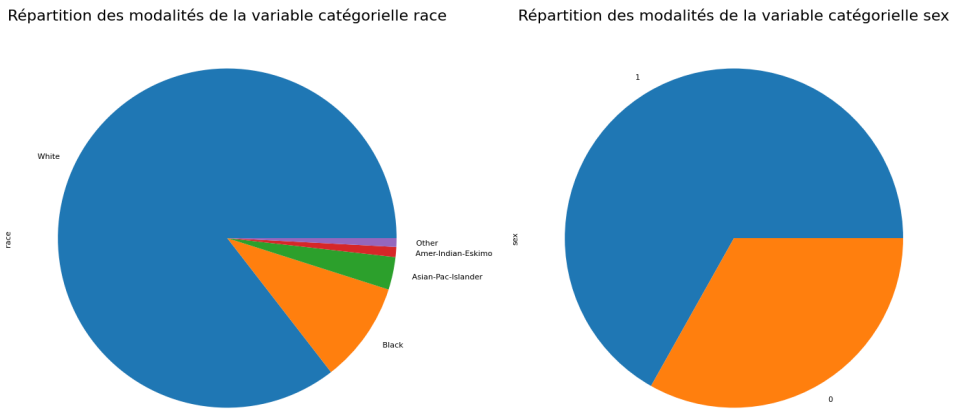

Figure 1: Répartition des modalités des variables *race* et *sex*

2.7.2 Variables numériques

Nous avons tracé des histogrammes pour observer la distribution de chacune des variables numériques (voir Figure 9 et Figure 10 en Annexe A.1). Dans la Figure 2, nous observons que la variable *hours-per-week* a une distribution très irrégulière avec le plus de données présentes dans 39. *fnlwgt* a une distribution irrégulière et a le plus de données dans 0.2e+6. Et *capital-gain* présente une asymétrie à droite.

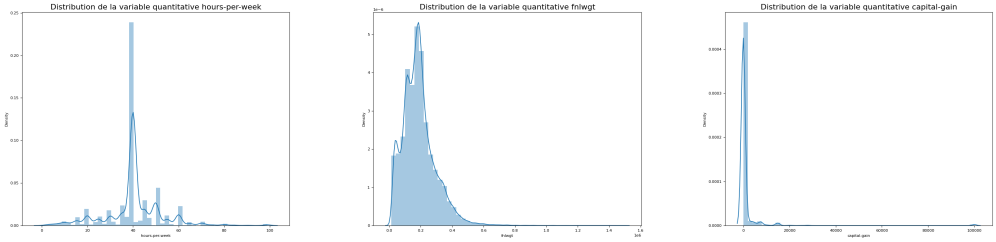

Figure 2: Distribution des variables numériques *hours-per-week*, *fnlwgt* et *capital-gain*

Lorsque nous traçons la matrice de corrélation pour les variables numériques, nous n'observons aucune corrélation entre les variables ou tellement faible que nous ne pouvons pas la retenir. Même chose lorsque nous traçons le pairplot, il n'y a aucune tendance particulière qui ressort (voir Figure 11 en Annexe A.1).

2.8 Analyse bivariée

2.8.1 Variables catégorielles

Nous effectuons une analyse bivariée entre la variable cible et les variables qualitatives. Nous utilisons à nouveau des matrices de corrélation sur le jeu de données total mais le déséquilibre de la plupart de ces variables rend le resultat non significatif. Nous retenons la seule variable (*occupation*) qui est à peu près équilibrée pour espérer optenir des relations pertinentes avec la variable cible (voir Figure 3).

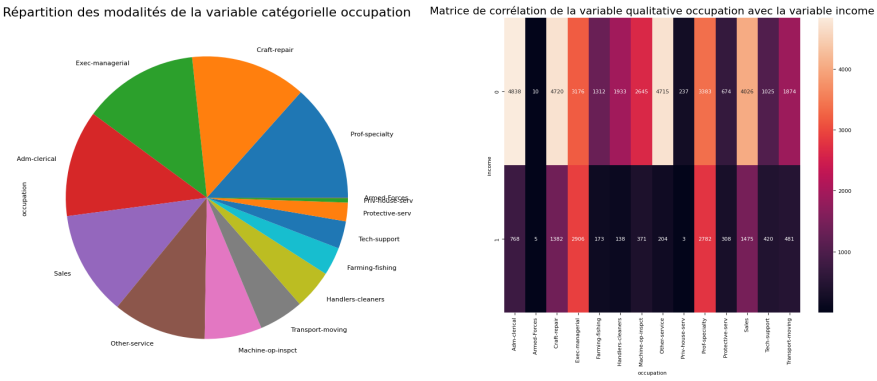

Figure 3: Répartition des modalité et matrice de corrélation de la variable *occupation*

Les relations qui semblent pertinentes sont celles de la variable cible à 0 avec les occupations : *Adm-clerical, Craft-repair, Other-service*. Les *occupation* qui semblent rapporter le plus sont : *Exec-Managerial* et *Prof-specialty*.

2.8.2 Variables numériques

Enfin, pour réaliser l'analyse bivariée avec les variables numériques, nous séparons le jeu de données total en deux sous-ensembles : celui des personnes qui gagnent plus de 50000$ annuellement et celui des personnes qui gagnent moins de 50000$ annuellement, conformément à la variable *income*.

Nous comparons la variable cible et les variables quantitatives. Nous observons que la variable *age* est plus élevée pour les personnes gagnant plus de 50000$ par an (voir Figure 4). Nous n'observons pas de différence avec les autres variables quantitatives (voir Figure 12 et Figure 13 en Annexe A.1).

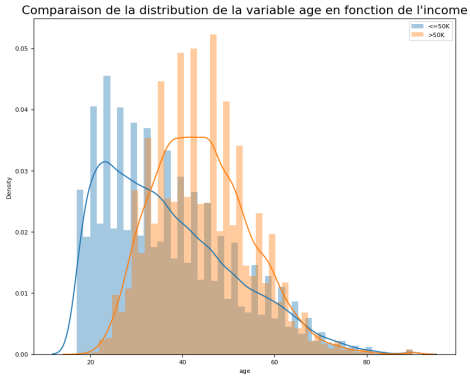

Figure 4: Comparaison des distributions de la variable age dans les deux sous-ensembles

3 Pré-traitement des données

3.1 Encodage des données

L'encodage des données est une étape essentielle dans la préparation des données. Comme vu en analyse exploratoire, notre jeu de données est composé de variables qualitatives ordinales et nominales. Dans la bibliothèque Scikit-Learn (`sklearn.preprocessing`), il existe plusieurs types d'encadoges disponibles :

- Le LabelEncoder : `sklearn.preprocessing.LabelEncoder()`,
- L'OrdinalEncoder : `sklearn.preprocessing.OrdinalEncoder()`

- L'encodage One Hot : `sklearn.preprocessing.OneHotEncoder()`

L'utilisation d'un encodage ou d'un autre dépend du type de variable que nous avons. Un encodage non adapté aux données peut significativement altérer la performance de notre modèle. Nous avions utilisé en premier lieu OrdinalEncoder() pour l'ensemble de nos variables catégorielles. Cependant, nous nous sommes rendus compte que ce dernier était inadapté pour bon nombre d'entre elles. L'OrdinalEncoder est pertinent si l'ensemble de nos variables catégorielles ont des modalités complètement ordonnées. Bien qu'il soit possible d'y voir une cohérence pour certains attributs comme *education*, cela devient moins pertinent pour les attributs comme *marital-status* ou *occupation*. Nous avons donc décidé d'utiliser l'encodage One Hot Encoder.

3.2 Normalisation des données

La normalisation des données s'applique à nos variables numériques. Ce procédés permet de donner à toutes nos features une échelle de comparaison cohérente pour que notre modèle ne donne pas plus d'importance à certains attributs qu'à d'autres. Comme pour l'encodage, la normalisation est possible par la bibliothèque python Scikit-Learn (`sklearn.preprocessing`) avec plusieurs possibilités :

- **Normalisation : transformation MinMax** (`preprocessing.MinMaxScaler()`). La transformation Min Max transforme chaque variable pour qu'elle soit comprise entre 0 et 1. Ainsi on maintient la distribution des données, sans perdre d'information. Cependant, cette normalisation est très sensibles aux valeurs extrêmes. Soit X la variable pre-normalisation, on nomme X_{scaled} la variable X post-normalisation : $X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$
- **Normalisation : standardisation** (`preprocessing.StandardScaler()`). La standardisation est le procédé de changement de la distribution d'une variable aléatoire X. Sa moyenne devient 0 et son écart-type est égal à 1. On note respectivement σ_X et μ_X l'écart-type et la moyenne de X. La formule de la standardisation devient : $X_{scaled} = \frac{X - \mu_X}{\sigma_X}$

Contrairement à l'encodage qui dépend de la variable à encoder, le choix d'une normalisation déprendra du modèle à entrainer. Si un modèle est sensible à l'échelle des variables, nous choisirons de préférence la **normalisation standardisation**. À l'inverse, si le modèle repose davantage sur des seuils de logiques ou des arbres de décisions, **la normalisation transformation min max** semble la plus adaptée.

3.3 Choix sur le pré-traitement des données

Afin de pouvoir maximiser la qualité d'apprentissage, nous avons fait des choix sur les données, notamment de supprimer la feature **education**. Il existe deux variables représentant le niveau d'éducation de la population : *education* et *education-num*. La variable *education* qualitative nominale. *education-num*, quand à elle, est une variable qualitative ordinale pouvant être considérée comme une variable quantitative discrète. Le problème s'est posé à partir de l'encodage de ces valeurs. En effet, il est pertinent de vouloir ordonner le niveau d'éducation. Cependant, garder deux features représentant la même donnée (le niveau d'éducation) risque de conduire à un overfitting. Nous avons donc décidé de nous décharger de la feature *education* et d'uniquement garder *education-num* déjà prête sans encodage. Nous avons vérifié qu'*education-num* est correctement ordonnée à l'aide des graphiques circulaires et en affichant leurs valeurs respectives côte-à-côte dans notre AED.

4 Modélisation

Nous avons affaire à un problème de classification binaire. Cela nous donne donc le choix parmi différents types de modèles de classification. Etant donné que nous avons choisi d'utiliser la librairie python **optuna**, nous pouvons faire des tests durant l'hyperoptimisation des paramètres pour déterminer quel modèle est le plus efficace dans notre cas.

Nous avons sélectionné les modèles de classification suivants pour nos tests. Nous les avons regroupés dans différentes sous-catégories :

- Méthodes à base d'arbres
  - Arbre de décision (`sklearn.tree.DecisionTreeClassifier`)
  - Random Forest (`sklearn.ensemble.RandomForestClassifier`)
- Méthodes de boosting
  - AdaBoost (`sklearn.ensemble.AdaBoostClassifier`)
  - Extreme gradient boosting (`xgboost.XGBClassifier`)
- Méthodes linéaires
  - Regression logistique (`sklearn.linear_model.LogisticRegression`)
  - Classifieur naïf de Bayes (`sklearn.naive_bayes.GaussianNB`)
- Méthodes à noyau et de marge
  - Machine à vecteurs de support (`sklearn.svm.SVC`)
- Méthodes à base de distances ou de densité
  - K-plus proches voisins (`sklearn.neighbors.KNeighborsClassifier`)
  - K-moyennes (`sklearn.cluster.KMeans`)
  - HDBSCAN (`sklearn.cluster.HDBSCAN`)
- Méthodes d'ensemble
  - Stacking avec random forest et extreme gradient boosting (`sklearn.ensemble.StackingClassifier`)
  - Stacking avec un classifieur naïf de Bayes et extreme gradient boosting (`sklearn.ensemble.StackingClassifier`)

5 HYPER-OPTIMISATION

Lors du traitement de nos données, nous avons encodé la variable d'intérêt de la façon suivante : $\leq 50\text{k} = 0 \quad \text{et} \quad > 50\text{k} = 1$

Afin d'optimiser nos paramètres, il nous faut une valeur à minimiser ou maximiser. Naïvement, nous pourrions opter pour une optimisation basée sur la maximisation de la précision. Cependant, rappelons que dans le cadre de la classification binaire, la formule de la précision est la suivante :

$$\frac{TP + TN}{TP + FP + TN + FN}$$

Or, comme vu dans l'analyse des données, nous avons une classe majoritaire représentant approximativement 76% des valeurs. Maximiser la précision pourrait résulter en un modèle moins performant, renvoyant quasi-systématiquement la classe majoritaire. Cela se traduirait également par un rappel très faible. Dans notre cas, la formule du rappel est :

$$\frac{TP}{TP + FN}$$

Rappelons que dans notre cas, la classe positive est "$>$ 50k", qui est également la classe minoritaire. En cherchant une maximisation du rappel, nous pouvons axer l'optimisation sur la bonne prédiction de la classe minoritaire plutôt que la bonne prédiction des valeurs en général. Cela devrait se traduire également par une hausse de la précision.

Concernant les hyperparamètres optimisés, voici une liste exhaustive de tous les hyperparamètres de chaque modèle tests et les valeurs pouvant être prises par ceux-ci :

Modèle	**Hyperparamètres**	**Valeurs prises**
Arbre de décision	`max_depth`	[2..32]
	`min_samples_split`	[2..10]
	`min_samples_leaf`	[1..4]
	`max_features`	['sqrt', 'log2']
	`criterion`	['gini', 'entropy']
Random Forest	`max_depth`	[2..32]
	`min_samples_split`	[2..10]
	`min_samples_leaf`	[1..4]
	`max_features`	['sqrt', 'log2']
	`criterion`	['gini', 'entropy']
	`n_estimators`	[50..200]
Regression logistique	`solver`	[newton-cg, lbfgs, liblinear, sag, saga]
	`penalty`	[None, l1, l2, elasticnet]
	`C`	[0.01; 10.0]
	`max_iter`	[50..500]
	`l1_ratio` (elasticnet)	[0.1; 0.5]
AdaBoost	`estimator`	[DecisionTree, RandomForest]
	`n_estimators`	[50..200]
	`learning_rate`	[0.01; 0.3]
Extreme gradient boosting	`max_depth`	[2..32]
	`min_samples_split`	[2..10]
	`min_samples_leaf`	[1..4]
	`max_features`	[sqrt, log2]
	`criterion`	[gini, entropy]
	`n_estimators`	[50..200]
	`learning_rate`	[0.01; 0.3]
K plus proches voisins	`n_neighbors`	[1..20]
	`weights`	[uniform, distance]
	`algorithm`	[auto, ball_tree, kd_tree, brute]
	`leaf_size`	[10..50]
	`p`	[1..2]
Classificateur naïf de Bayes	`var_smoothing`	$[1e^{-11}; 1e^{-6}]$
Machine à vecteurs de support	`C`	[0.01; 10]
	`kernel`	[linear, poly, rbf, sigmoid]
	`gamma`	[scale; auto]
	`degree`	[2..5]
	`class_weight`	[None, balanced]
	`n_estimator`	[10..200]
K-moyennes	`n_clusters`	2
	`init`	[k-means++, random]
	`n_init`	[10..50]
	`max_iter`	[100..500]
	`tol`	$[1e^{-6}; 1e^{-2}]$
HDBSCAN	`min_cluster_size`	[2..50]
	`min_samples`	[1..10]
	`cluster_selection_epsilon`	[0; 0.5]
Stacking (RF-XGB)	Voir Random Forest et Gradient Boosting	idem
Stacking (Bayes-XGB)	Voir Classificateur naïf de Bayes et Gradient Boosting	idem

Table 2: Liste de tous les hyperparamètres des différents classifieurs testés avec leurs valeurs prises

Les modèles sont optimisés à l'aide de **optuna**. Cette dernière est une librairie Python permettant de créer des études (**studies**) qui vont tenter de minimiser ou maximiser une valeur en jouant sur les hyperparamètres d'un modèle. Pour trouver ces hyperparamètres, on informe **optuna** des différents hyperparamètres et des valeurs qu'ils peuvent prendre et, à chaque essai (**trial**), la librairie va ajuster

ces paramètres pour maximiser ou minimiser une valeur que nous choisissons. Comme expliqué plus tôt, nous voulons maximiser le rappel donc nous renvoyons a chaque **trial** le rappel du modèle testé.

Certains modèles étant trop lents à optimiser, nous avions besoin de limiter le nombre d'essais pouvant être réalisés et le temps alloué à cette tâche. **optuna** permet de fixer un nombre de **trials** par **study** ainsi qu'un timeout au bout duquel l'hyperoptimisation s'arrête. Après de nombreux essais, nous avons fixé le nombre de **trials** à 150 et le timeout à 900 secondes (soit 15 minutes).

Comme vu précédemment, le jeu de données était livré avec un set de train (67% des données) et un set de test (33% des données). Afin de pouvoir tester les modèles sur des données inédites, nous avont décidé de séparer le train set en deux sous-sets : un set de train (80% des données de train original) et un set de validation (20% des données de train original). Cela nous donne la répartition finale suivante :

- Train set : 53.6%
- Validation set : 13.4%
- Test set : 33% (utilisé dans la partie évaluation)

6 ÉVALUATION

Afin d'éviter tout problème d'overfitting durant l'entraînement des modèles et l'optimisation de leurs hyperparamètres, nous avons à disposition un jeu de test contenant des données inédites aux différents modèles pour l'évaluation.

Pour rappel, notre classe négative représente 76% de nos données. Ainsi, nous attendons une précision supérieure à 0.76 (mieux que le hasard) et un rappel ne valant pas 0 (prédiction des deux classes et pas que d'une seule).

Après entraînement et optimisation des hyperparamètres, nous obtenons les statistiques suivantes pour nos différents modèles :

Modèle	Précision	Rappel	Rappel négatif	Score F1	MAE	RMSA
Arbre de décision	0.84	0.57	0.93	0.63	0.16	0.16
Random Forest	**0.86**	0.59	**0.95**	**0.67**	**0.14**	**0.14**
Regression logisitique	0.85	0.59	0.93	0.65	0.15	0.15
AdaBoost	**0.86**	**0.61**	**0.94**	**0.68**	**0.14**	**0.14**
Extreme gradient boosting	**0.87**	**0.64**	**0.94**	**0.70**	**0.13**	**0.13**
K-plus proches voisins	0.82	0.57	0.90	0.60	0.18	0.18
Classifieur naïf de bayes	0.49	**0.96**	0.34	0.47	0.51	0.51
Machine à vecteurs de support	0.83	0.56	0.92	0.62	0.17	0.17
K-moyennes	0.72	0.58	0.77	0.49	0.28	0.28
Stacking (RF-XGB)	0.87	0.64	0.94	0.70	0.13	0.13
Stacking (Bayes-XGB)	0.87	0.67	0.93	0.71	0.13	0.13

Table 3: Évaluation des différents modèles optimisés

MAE signifie "Mean Absolute Error" soit "Moyenne de l'erreur absolue".

RMSA signifie "Root Mean Squared Error" soit "Racine de l'erreur quadratique moyenne".

En gardant à l'esprit que notre jeu de données est désequilibré, nous nous concentrons sur 3 scores principaux :

- Le rappel (proportion de bonnes prédictions lors de prédictions positives)

- Le rappel négatif (proportion de bonnes prédictions lors de prédictions négatives)

- La précision (proportion de bonnes prédictions)

La modèle le plus approprié pour l'instant semble être **extreme gradient boosting**, suivi de **AdaBoost** et **Random Forest**.

Nous nous intéressons égalements aux différentes ROC et Précision-Rappel des différents modèles mais surtout aux scores **ROC-AUC** (aire sous la courbe ROC) et **PR-AUC** (aire sous la courbe Précision-Rappel). Nous cherchons à trouver le modèle qui maximise ces valeurs, plus particulièrement le **PR-AUC**, du fait que nos données sont désequilibrées.

Modèle	ROC-AUC	PR-AUC
Arbre de décision	0.87	0.71
Random Forest	**0.92**	**0.80**
Regression logisitique	0.90	0.76
Adaboost	**0.92**	**0.80**
Extreme gradient boosting	**0.93**	**0.82**
K-plus proches voisins	0.84	0.63
Classifieur naïf de bayes	0.71	0.35
Machine à vecteurs de support	0.89	0.71
Stacking (RF-XGB)	0.92	0.81
Stacking (Bayes-XGB)	0.92	0.81

Table 4: ROC-AUC et PR-AUC des différents modèles

Nous remarquons, encore une fois, que c'est le modèle **extreme gradient boosting** qui obtient les meilleurs résultats, suivi encore une fois par **AdaBoost** et **Random Forest**. Voici les courbes ROC et Précision-Rappel des meilleurs modèles.

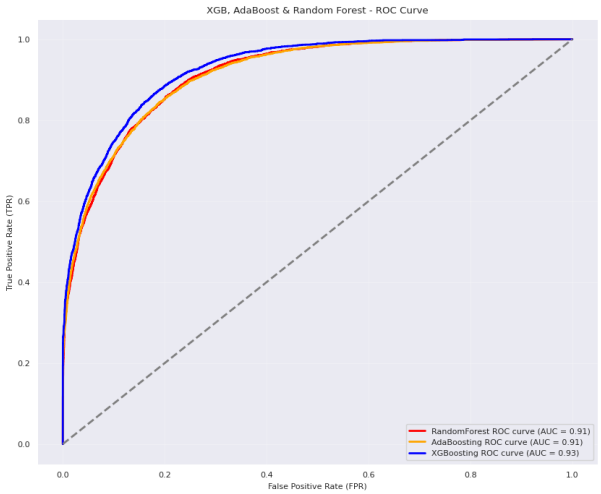

Figure 5: Courbes ROC des 3 meilleurs modèles.

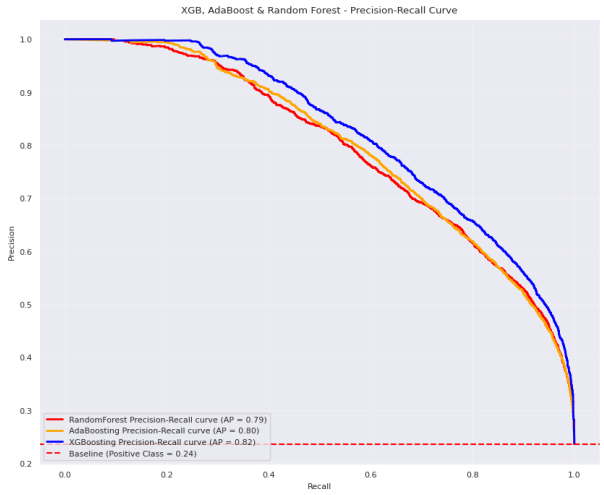

Figure 6: Courbes Précision-Rappel des 3 meilleurs modèles

En réalité, c'est en observant les résultats d'**extreme gradient boosting** et du **classifieur naïf de Bayes** que nous avons eu l'idée de créer un modèle de stacking les combinant. En effet, on observe que le classifieur naïf de Bayes est très performant en ce qui concerne la classification de la classe minoritaire (positive). Son rappel est bien supérieur à celui des autres modèles. Ainsi, nous avons pensé que le combiner au modèle le plus performant pourrait retourner des résultats intéressants. Finalement, en comparant les résultats du stacking à ceux d'extreme gradient boosting, nous observons en effet une légère amélioration des performances en ce qui concerne le rappel ($\approx +2\%$) mais au coût du rappel négatif ($\approx -2\%$).
En analysant les courbes ROC et Précision-Rappel du modèle, on constate un PR-AUC et un ROC-AUC diminué de $\approx 1\%$ par rapport à l'extreme gradient boosting. Cette solution est donc moins performante.

Nous avons également tenté de mélanger deux des meilleurs modèles appartenant à des catégories de modèles différentes. Nous avons donc fait un stacking **random forest** et **extreme gradient boosting**. Les performances obtenues pour le stacking RF-XGB sont exactement les mêmes que celles de extreme gradient boosting. Lors de l'analyse du ROC-AUC et du PR-AUC, on remarque néanmoins une perte de $\approx 1\%$ par rapport aux performances de extreme gradient boosting sur les deux métriques. Etant donné que random forest et extreme gradient boosting sont tous les deux basés sur des arbres de décision (l'un basé sur du bagging et l'autre du boosting), il n'est pas étonnant que nous n'ayons pas un gain ou une perte de performances significatif.

On en déduit que le meilleur modèle est l'**extreme gradient boosting**, avec une précision de 87%. Suivi de près par **AdaBoost** (86%) et **Random Forest** (86%), et des deux modèles de stacking (les deux utilisant extreme gradient boosting).

7 Conclusion

Dans ce projet, nous avons pu mener un projet de machine learning de bout en bout. La réalisation de l'analyse exploratoire de données du jeu "Adult" nous a permis de mettre en exergue différents traitement à réaliser pour affiner nos recherches de modèles. La surreprésentation de certaines données nous a exposé à un risque de tendre vers un over-fitting important. Cela nous a dirigé vers un prétraitement avec la suppression de la variable education, une normalisation type standardisation et un encodage One Hot. Malgré nos tentatives de combiner différents modèles via du stacking, aucun gain significatif de performance n'a été observé.

Les résultats montrent que l'Extreme Gradient Boosting reste le meilleur modèle, avec une précision de 87%. Cette stabilité pourrait s'expliquer par la similarité structurelle entre les modèles combinés, tous fondés sur les arbres de décisions.

7.1 Nos recommandations

À l'issue de ce travail, nous sommes en mesure de recommander le modèle d'**extreme gradient boosting** grâce à ses performances supérieures aux autres modèles que nous avons expérimenté.

7.2 Retour sur le projet

Ce projet nous a permis d'approcher de manière pratique un problème de machine learning. Nous avons pu faire notre prore analyse exploratoire des données afin de prévoir nos prétraitements pour ensuite entrainer des modèles et optimiser leurs hyperparamètres. Malheureusement, nous n'avons pas réussi à obtenir un modèle avec des performances supérieures à 87%, malgré de nombreuses tentatives au niveau du prétraitement des données et des paramètres d'entrainement du modèle. Nous avons parfois eu de la chance et obtenu une précision atteignant 89% sur certains essais mais ceux-ci étaient trop rares pour être utilisés comme la performance réelle de notre système. Globalement, nous sommes tout de mêmes contents de ce que nous avons pû accomplir.

7.3 Contributions des auteurs

Alexandre Bidaux Rédaction du rapport, pré-traitement des données (pour l'entraînement), entraînement et optimisation des modèles, évaluation des modèles.

Julie Chartier Rédaction du rapport, Analyse Exploratoire des Données, pré-traitement des données partiellement (traitements complémentaires à l'AED).

Quentin Valakou Rédaction du rapport, entraînement et optimisation des modèles, pré-traitement des données.

A Annexes

A.1 Figures complémentaires

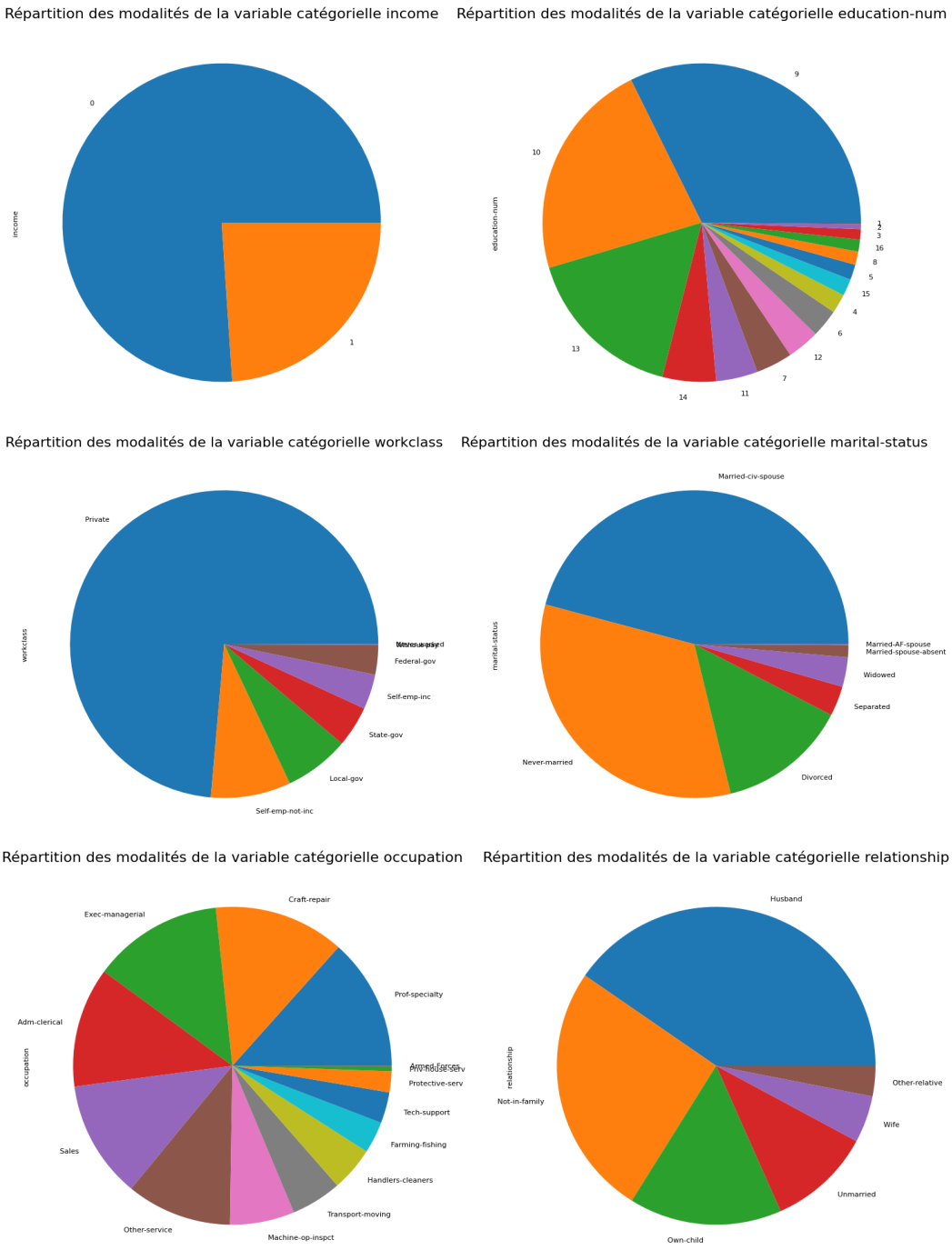

Figure 7: Graphiques circulaires des variables qualitatives

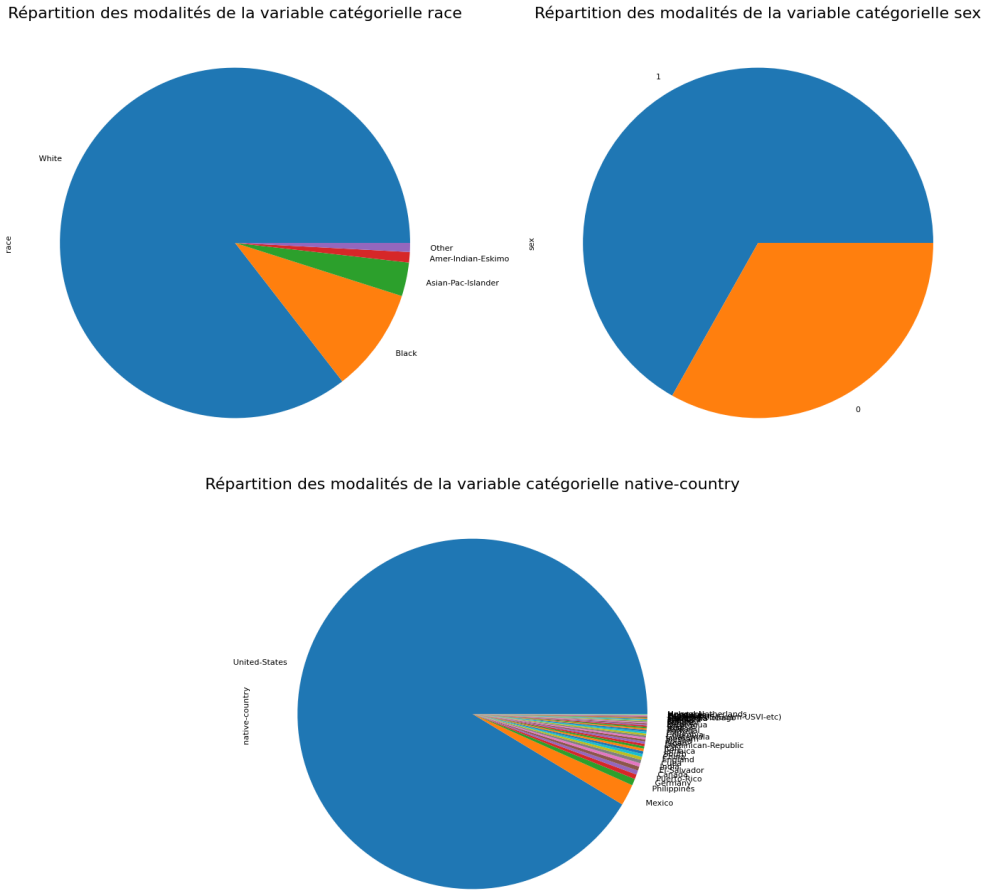

Figure 8: Deuxième partie des graphiques circulaires des variables qualitatives

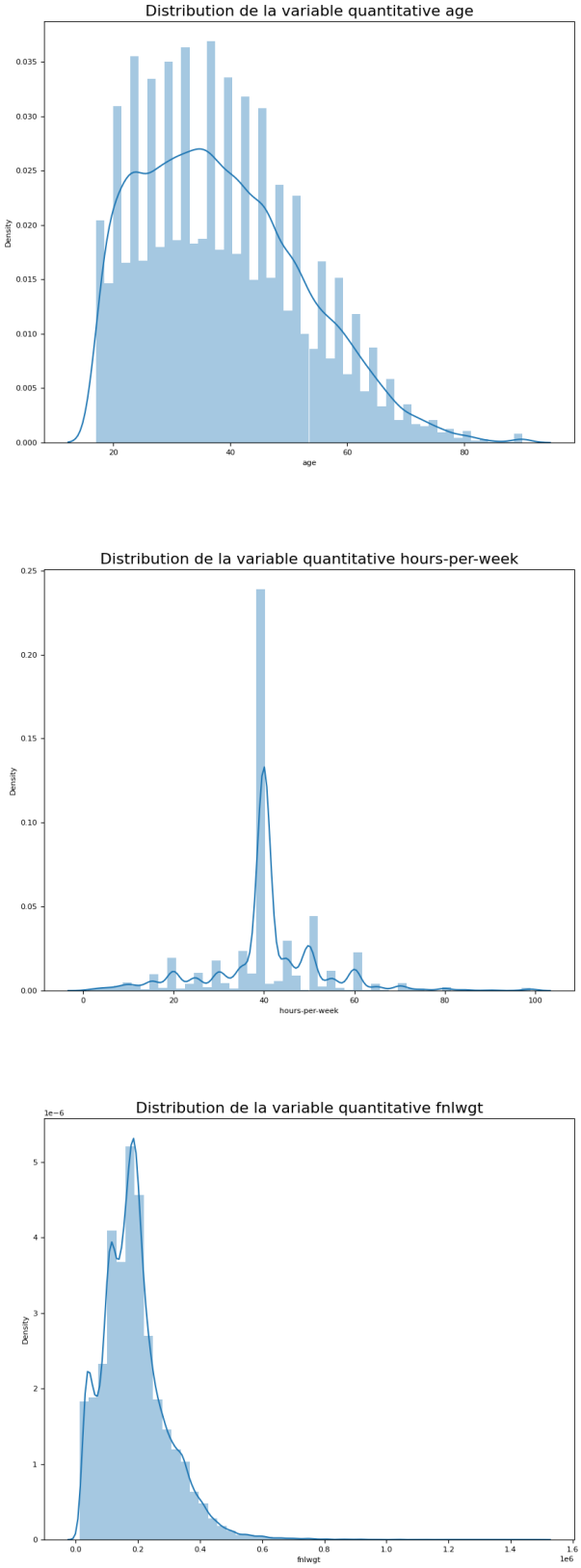

Figure 9: Histogrammes des variables numériques

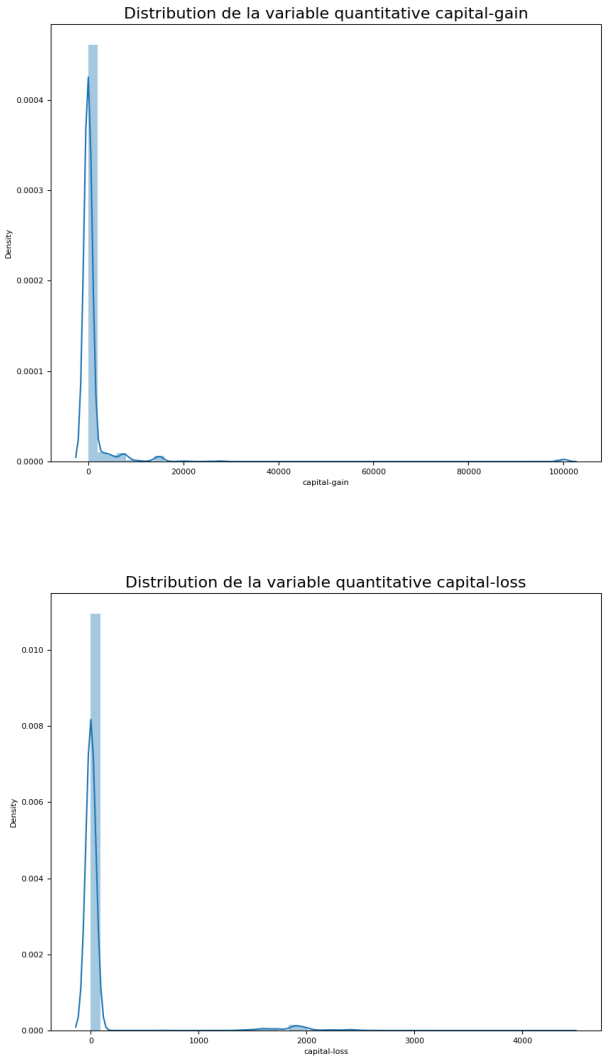

Figure 10: Suite des histogrammes des variables numériques

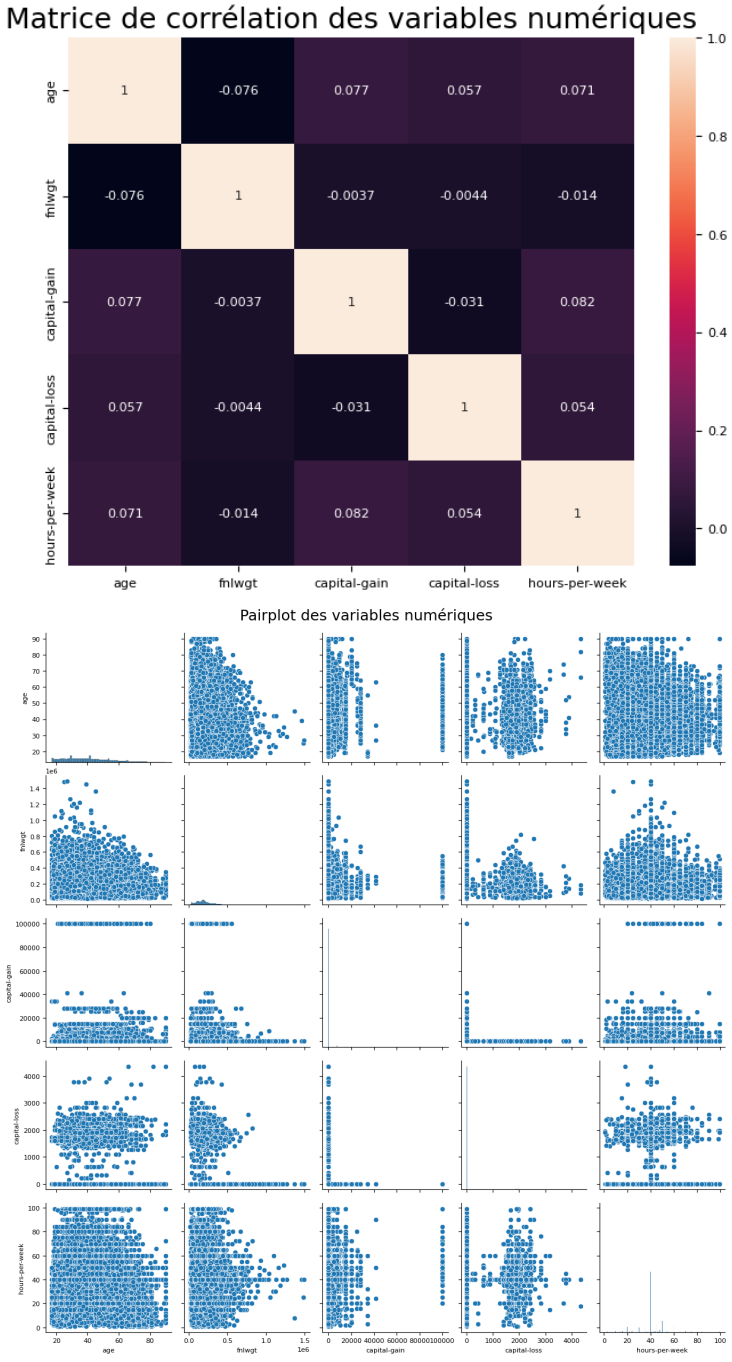

Figure 11: De haut en bas, matrice de corrélation et pairplot des variables numériques

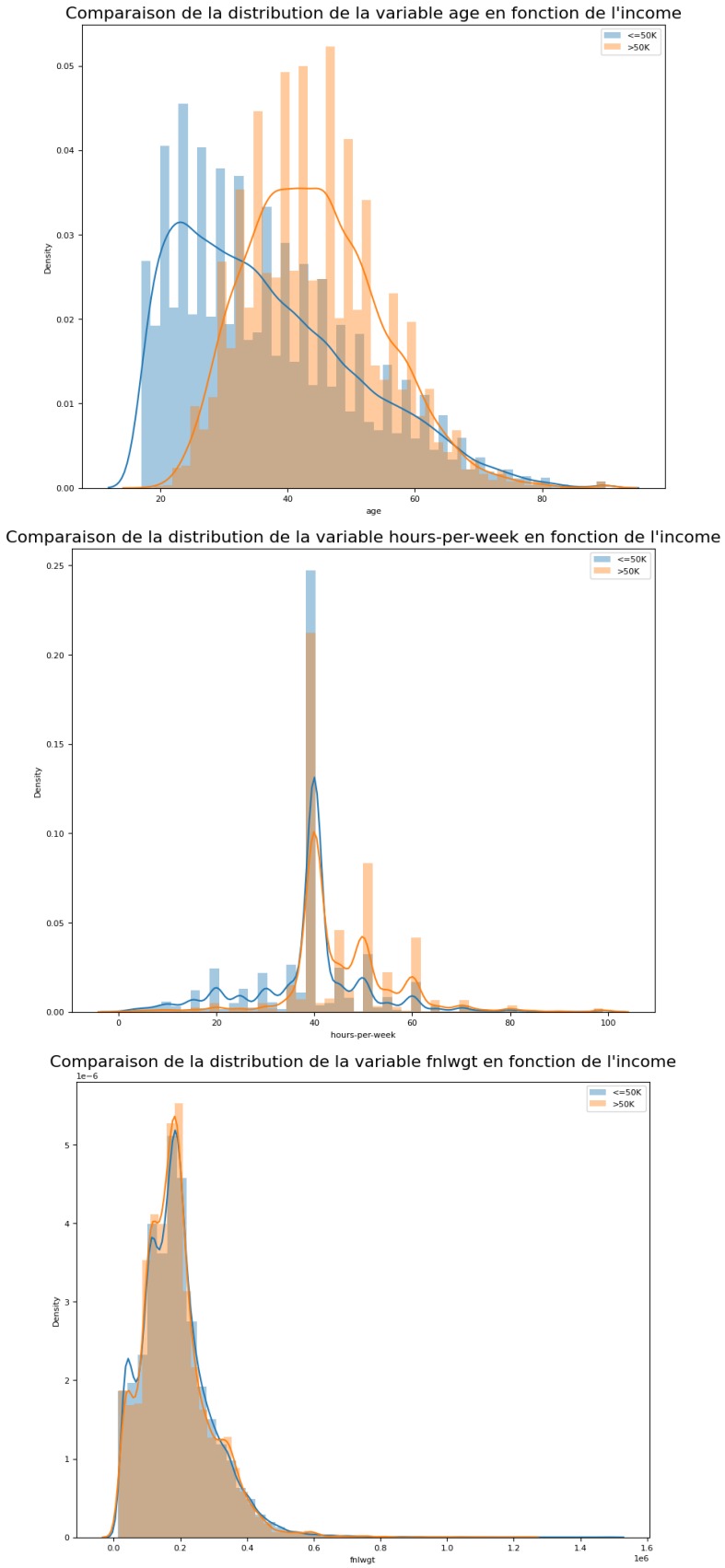

Figure 12: Histogrammes de comparaison des deux sous-ensembles de la variable cible avec les variables numériques

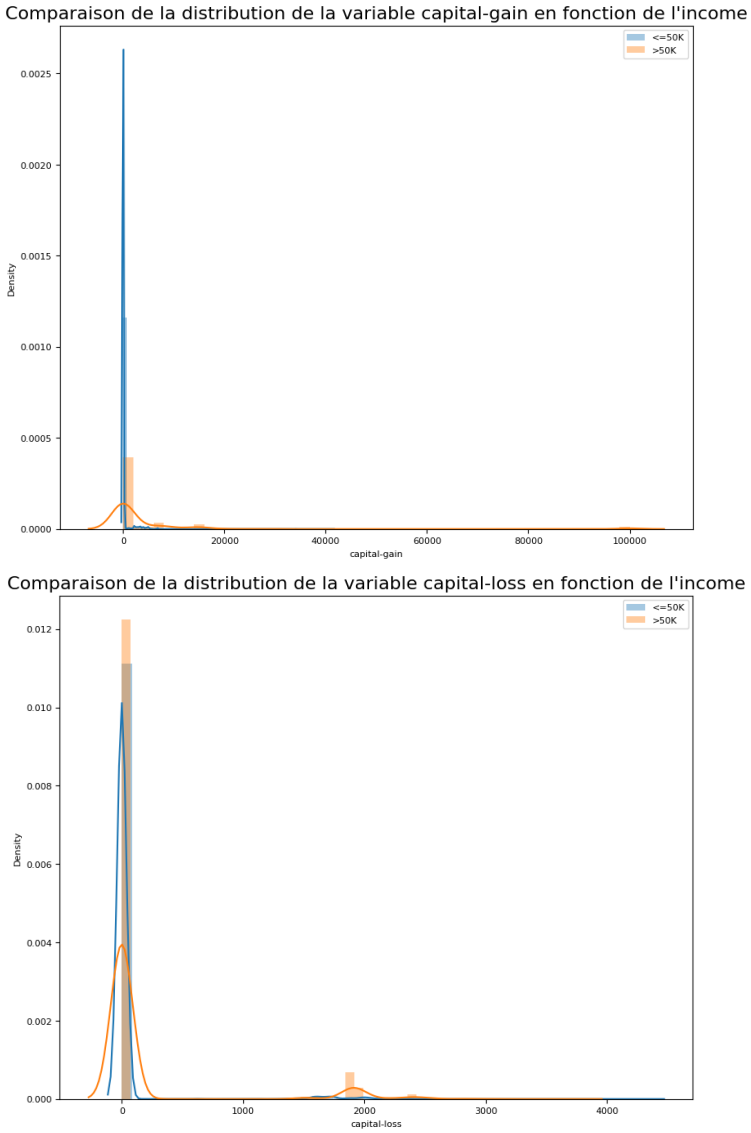

Figure 13: Deuxième partie des histogrammes de comparaison des deux sous-ensembles de la variable cible avec les variables numériques