Rapport TP Optimisation de l’apprentissage automatique

Master 1 TIC 2024-2024

INSTITUT DE MATHEMATIQUES ET DES SCIENCES PHYSIQUES

**Sous la supervision de** :

Dr Bah Habib SIDI

**Sous la supervision de** :

ADJAHO Houefa SARA ODILE

MAMA ABDOU RAOUFOU

Ce projet s’appuie sur un ensemble de données issues du **recensement américain**, disponible publiquement via le **UCI Machine Learning Repository**. Il vise à traiter l'ensemble du processus d'analyse et de prédiction, de la préparation des données jusqu'à l'évaluation des modèles.

1. **Objectif et contexte du projet**

* Prédire si un individu gagne plus ou moins de 50 000 $ par an.
* Identifier les facteurs importants qui influencent le revenu.
* Étudier les disparités selon le genre ou la race.
* Rechercher d’éventuels clusters sous-jacents aux données

1. **Tâches menées**

Afin d’atteindre ces objectifs, un ensemble d'étapes a été mis en place pour construire un processus complet d’analyse et de prédiction à partir de données réelles. Les principales actions menées sont présentées ci-dessous.

1. **Importation des bibliothèques essentielle**

Plusieurs bibliothèques incontournables ont été importées afin de faciliter la manipulation, la visualisation et le prétraitement des données.

* NumPy et Pandas ont permis de gérer efficacement les structures de données et d’effectuer les opérations de traitement nécessaires.
* Matplotlib et Seaborn ont été mobilisées pour créer des visualisations graphiques et explorer les relations entre les variables, avec une configuration esthétique du style via sns.set(style="darkgrid").
* Les outils de prétraitement de scikit-learn ont été utilisés pour transformer les données :
* MinMaxScaler pour normaliser les variables numériques,
* LabelEncoder pour encoder les variables catégorielles ordinales,
* OneHotEncoder pour convertir les variables catégorielles nominales en variables indicatrices.
* Enfin, le module time a servi à mesurer précisément les durées d’exécution lors de l’entraînement et de la prédiction des différents modèles.

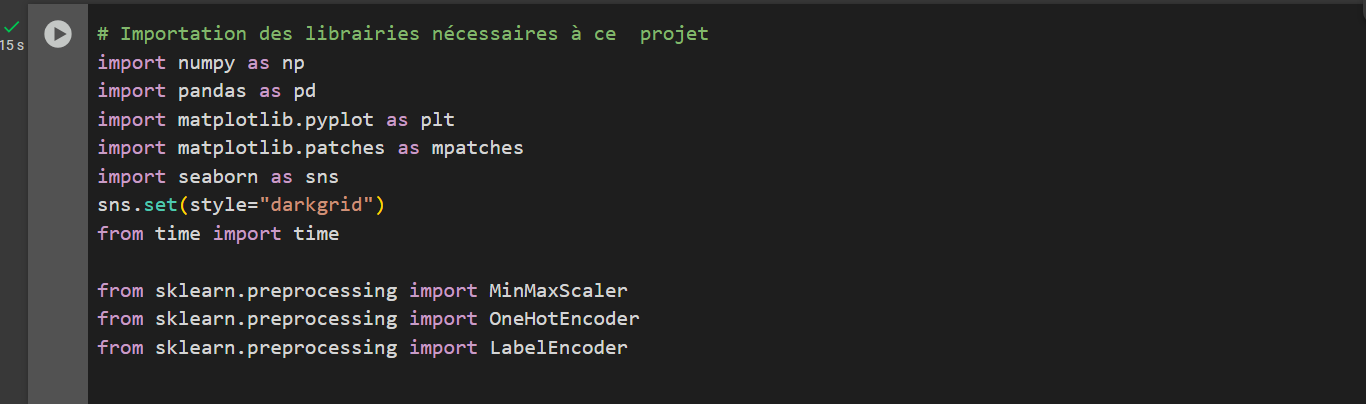


Figure 1

1. **Chargement des données et visualisation du format des données**

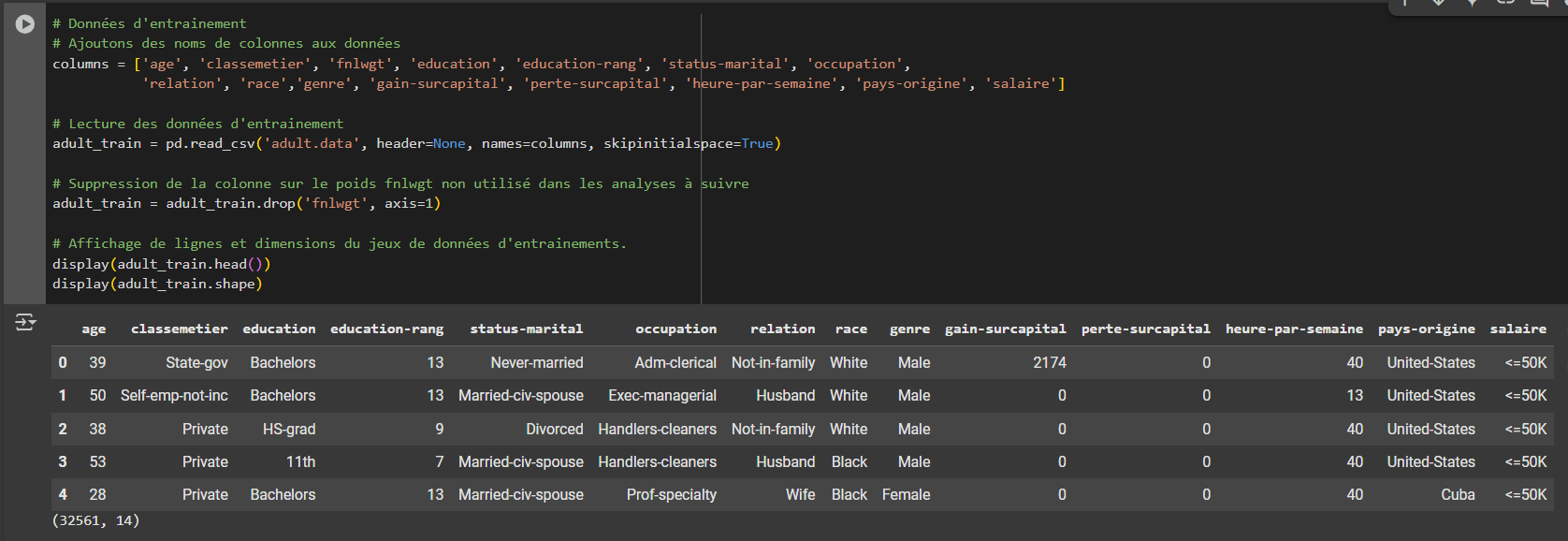
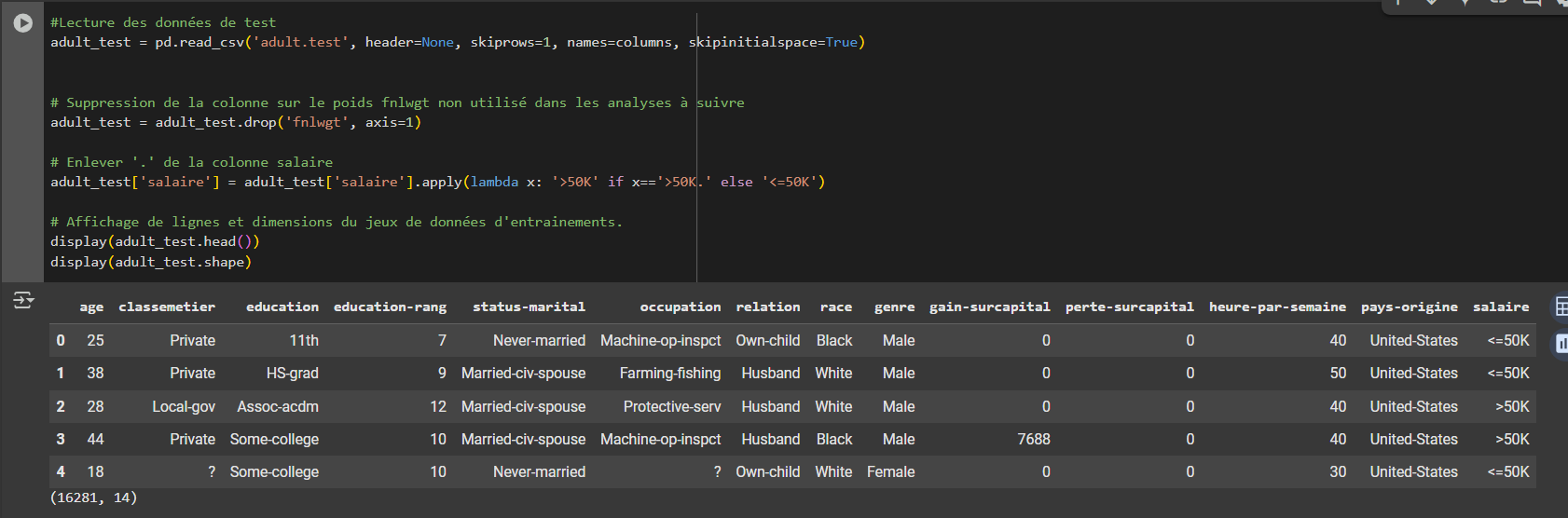
* **Ajout des noms de colonnes :** Pour faciliter l’analyse et la manipulation des données, des noms explicites ont été ajoutés aux colonnes des jeux de données adult.data (entraînement) et adult.test (test). Ces noms permettent de mieux comprendre la signification de chaque variable. La liste des colonnes ajoutées est la suivante :['age', 'classemetier', 'fnlwgt', 'education', 'education-rang', 'status-marital', 'occupation', 'relation', 'race','genre', 'gain-surcapital', 'perte-surcapital', 'heure-par-semaine', 'pays-origine', 'salaire']
* **Lecture du jeu de données d’entrainement et de test :** Les fichiers adult.data et adult.test, contenant respectivement les données d'entraînement et de test, ont été importés à l’aide de la bibliothèque pandas via la fonction read\_csv(). Comme ces fichiers ne contiennent pas d'en-tête, l'argument header=None a été utilisé pour éviter toute confusion dans l’interprétation des données. Les noms de colonnes ont ensuite été explicitement définis à l'aide du paramètre names=columns, permettant ainsi d'identifier clairement chaque variable. De plus, l'argument skipinitialspace=True a été spécifié pour ignorer les espaces inutiles après les virgules et garantir que les valeurs soient correctement formatées. Une particularité a été appliquée au fichier adult.test : l’argument skiprows=1 a été ajouté afin de sauter la première ligne du fichier, qui ne contient pas de données exploitables.
* **Suppression de la colonne fnlwgt :** La colonne fnlwgt (final weight) a été supprimée dans les deux jeux de données, car elle n’est pas utile dans les analyses et modèles statistiques envisagés ici.
* **Nettoyage de la colonne salaire (jeu de test uniquement)** : Dans le fichier adult.test, les valeurs de la colonne salaire contenaient un point final (>50K. ou <=50K.), ce qui les rendait incohérentes avec celles du fichier d'entraînement. Une opération de nettoyage a été effectuée pour uniformiser ces valeurs (>50K / <=50K).
* **Affichage de lignes et dimensions du jeux de données d'entrainements :**  Après nettoyage, un aperçu des premières lignes de chaque jeu a été affiché à l’aide de la fonction head() pour vérifier leur structure ainsi que leurs dimensions avec shape().  
  Le jeu de données d’entraînement contient 32 561 instances (lignes) et 14 variables, tandis que le jeu de données de test comprend 16 281 instances ****(lignes) et 14 variables également.****

Figure 2

Figure 3

1. **Examen et gestion des valeurs manquantes**

Afin de s'assurer que les jeux de données d’entraînement et de test sont prêts pour l’analyse, une vérification des valeurs manquantes a été effectuée.

* **Identification des valeurs manquantes :** La fonction info() a été utilisée pour examiner le jeu de données et identifier les colonnes de type objet. Ensuite, value\_counts(dropna=False) a permis de détecter les valeurs ?, utilisées pour indiquer des données manquantes.
* **Conversion des valeurs ? en NaNs :** Les valeurs ? ont été converties en NaN à l’aide de loc[], ce qui permet de remplacer ces valeurs dans les colonnes concernées des jeux de données adult\_train et adult\_test.
* **Suppression des lignes avec NaNs :** Les lignes contenant des valeurs NaN ont été supprimées avec la fonction dropna(axis=0, how='any'), qui permet de supprimer les lignes où au moins une valeur est manquante**.**
* **Résultat final :**Après suppression des valeurs manquantes :Le jeu d’entraînement contient désormais 30 162 instances, contre 32 561 instances avant nettoyage. Le jeu de test contient désormais 15 060 instances, contre 16 281 instances avant nettoyage**.**

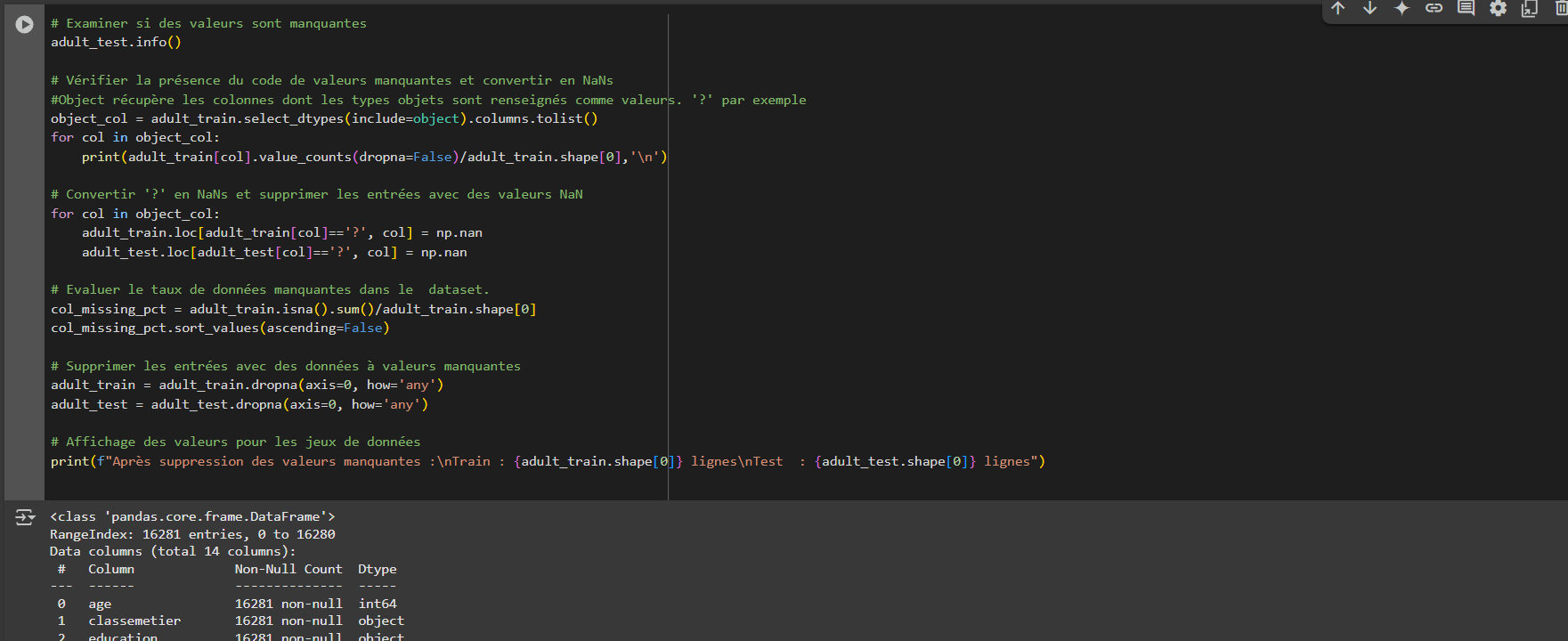


Figure 4

1. **Revue des données**

Dans cette section, les jeux de données d’entraînement et de test ont été combinés afin d'obtenir une distribution plus complète et générale des données. Cela permet d'avoir une vue d'ensemble du jeu de données global pour les analyses futures.

* **Combinaison des jeux de données :** Les jeux de données d’entraînement (adult\_train) et de test (adult\_test) ont été fusionnés à l’aide de la fonction pd.concat(), créant ainsi un jeu de données unique contenant à la fois les données d’entraînement et de test.
* **Calcul des statistiques** :  
  Les calculs suivants ont été effectués sur l’ensemble combiné des données (adult\_data) :
* Total du nombre d'entrées : Le jeu de données combiné contient 45 222 entrées.
* Individus ayant un salaire supérieur à $50,000 : Il y a 11 208 individus avec un salaire supérieur à $50,000.
* Individus ayant un salaire inférieur ou égal à $50,000 : 34 014 individus gagnent au plus $50,000.
* Une image contenant texte, logiciel, Logiciel multimédia, Logiciel de graphisme

  Le contenu généré par l’IA peut être incorrect.Pourcentage d'individus avec un salaire supérieur à $50,000 : 24,78% des individus dans le jeu de données ont un salaire supérieur à $50,000.

Figure 5

1. **Préparation des données**
2. **Transformer des données continues asymétriques (skewness)**

* **Vérification de l'asymétrie des données numériques :**Les variables numériques du jeu de données d’entraînement ont été examinées pour détecter d’éventuelles asymétries. Un graphique des distributions de chaque caractéristique numérique a été tracé à l’aide de la fonction hist(), permettant d’observer les distributions des données. Comme montré dans les graphiques, deux caractéristiques, 'gain-surcapital' et 'perte-surcapital', présentent des distributions asymétriques marquées, avec des pics importants dans la partie inférieure des distributions.
* **Mesure de l’asymétrie :** L’asymétrie des variables numériques a été mesurée à l’aide de la méthode skew() de pandas. Les résultats montrent que 'gain-surcapital' présente une asymétrie de 11.90 et 'perte-surcapital' une asymétrie de 4.53, ce qui confirme les observations visuelles de distributions asymétriques marquées. En revanche, les autres variables sont relativement proches d'une distribution normale, avec des asymétries plus faibles : 'age' avec 0.53, 'heure-par-semaine' avec 0.33, et 'education-rang' avec -0.31.
* **Transformation des données asymétriques :** Pour corriger l’asymétrie des variables 'gain-surcapital' et 'perte-surcapital', une transformation logarithmique a été appliquée à ces deux caractéristiques dans les jeux de données d’entraînement et de test. Cette transformation a été effectuée en utilisant np.log(x + 1), ce qui permet de réduire l’asymétrie et de rendre les distributions plus proches de la normale.

Les variables 'gain-surcapital' et 'perte-surcapital' nécessitant une transformation en raison de leur forte asymétrie, la transformation logarithmique a permis d'améliorer leur symétrie. Les autres variables ne nécessitaient pas de transformation, car leurs asymétries sont faibles et acceptables.

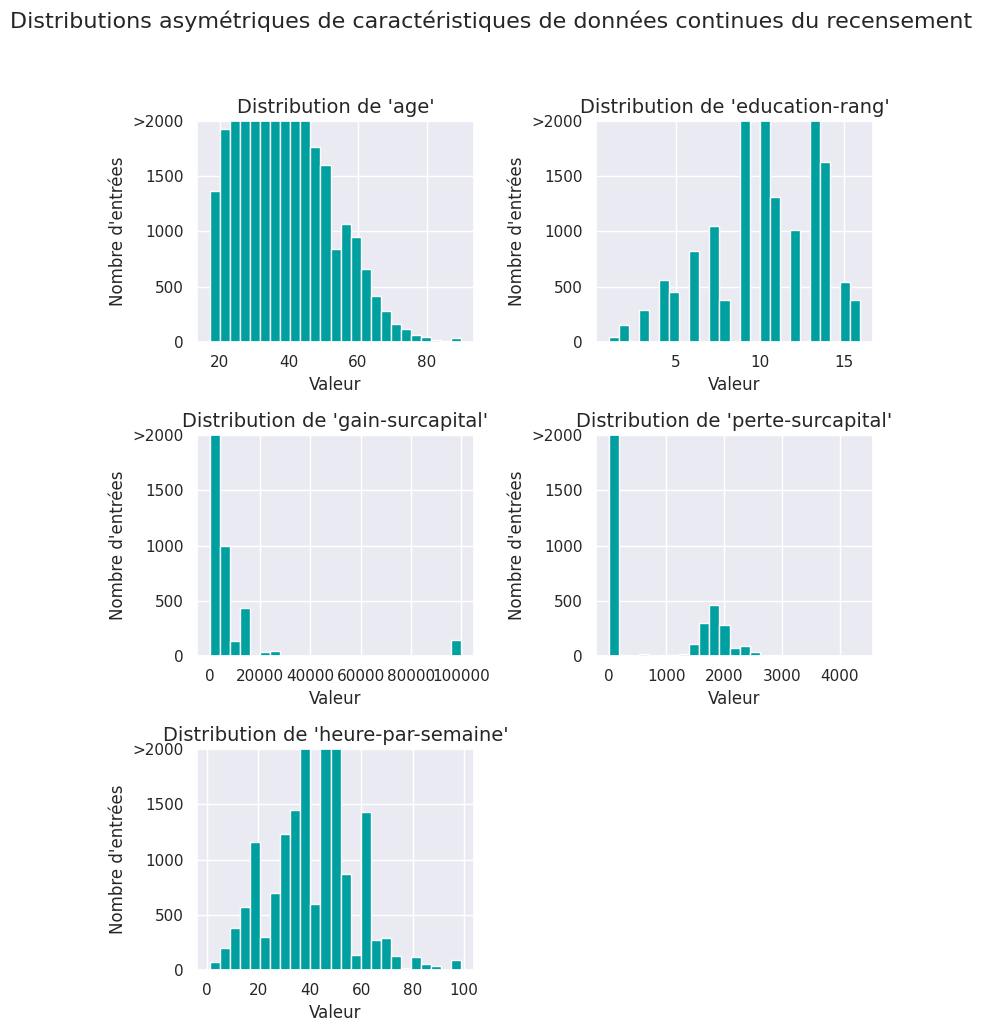


Figure 6

1. **Normalisation des caratéristiques numériques**

En complément de la transformation des variables fortement asymétriques, une mise à l’échelle (normalisation) a été appliquée aux variables numériques du jeu de données.

* **Méthode utilisée** : La mise à l’échelle a été réalisée à l’aide du MinMaxScaler de sklearn.preprocessing, qui transforme chaque valeur numérique pour qu’elle soit comprise dans l’intervalle **[0, 1]**. Cela permet de garantir une échelle uniforme entre les différentes variables numériques.
* **Application sur les données :** Le **scaler** a été ajusté (fit) sur les variables numériques du jeu d'entraînement transformé (features\_log\_transformed), puis appliqué (transform) aux deux ensembles de données. Cela a permis d’obtenir les ensembles mis à l’échelle suivants : features\_log\_minmax\_transform pour les données d'entraînement, et features\_log\_minmax\_transform\_test pour les données de test.



Figure 7

1. **Prétraitement des données – Encodage One-Hot et fusion**

* **Détection des variables catégorielles :**  
  Les colonnes de type object ont été identifiées dans le jeu de données features\_log\_minmax\_transform comme étant des variables à encoder.
* **Transformation avec LabelEncoder :**  
  Chaque variable catégorielle a d’abord été convertie en valeurs numériques entières à l’aide de LabelEncoder, afin de préparer les données pour un encodage ultérieur.
* **Encodage One-Hot via OneHotEncoder :**  
  Les valeurs numériques obtenues ont ensuite été encodées en variables binaires avec OneHotEncoder, générant une colonne distincte pour chaque modalité possible.
* **Nom des nouvelles colonnes :**  
  Les noms des colonnes générées ont été extraits à l’aide de get\_feature\_names\_out, assurant une bonne lisibilité des variables encodées.
* **Création des jeux de données encodés :**  
  Deux nouveaux DataFrames ont été produits :
* encoded\_cat\_feats\_df pour les données d’entraînement,
* encoded\_cat\_feats\_df\_test pour les données de test.  
  Ces DataFrames contiennent uniquement les variables catégorielles encodées de façon numérique.

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.

Figure 8

1. **Construction des caractéristiques finales du modèle**

* Les variables numériques ont été extraites du jeu de données transformé (features\_log\_minmax\_transform) pour former un DataFrame num\_feats\_df.
* Ces variables numériques ont ensuite été fusionnées avec les variables catégorielles encodées (encoded\_cat\_feats\_df) pour créer le jeu d'entraînement complet X\_train.
* La variable cible salaire\_brut a été convertie en valeurs binaires :  
  >50K → 1,  
  <=50K → 0,  
  donnant le vecteur cible y\_train.
* Le même traitement a été appliqué au jeu de test, produisant X\_test et y\_test.

Après l’encodage one-hot et la fusion des données, chaque individu est représenté par 103 caractéristiques prêtes pour l'entraînement des modèles.

Une image contenant capture d’écran, texte, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Figure 9

1. **Evaluation des Performances des Modèles (Prédicteur Naïf)**

Avant de comparer différents algorithmes d’apprentissage supervisé, un prédicteur naïf a été utilisé comme modèle de base. Ce modèle suppose qu’un individu gagne toujours plus de 50 000 $, autrement dit, il prévoit systématiquement la classe majoritaire (1) dans ce contexte d'évaluation orientée vers la précision des hauts revenus.

**Métriques obtenues:**

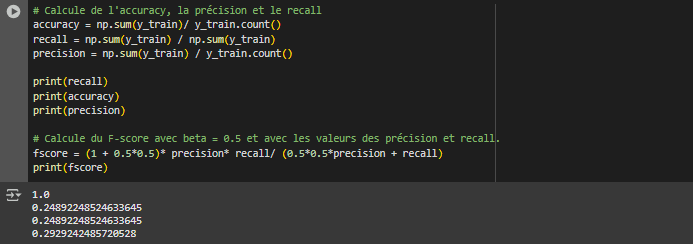
* **Accuracy (Exactitude)** : 24.89 %  
   Cela correspond à la proportion d’individus réellement gagnant plus de 50 000 $ dans l’ensemble d’entraînement.
* **Recall (Rappel)** : 1.0  
   Le modèle détecte **100 %** des individus gagnant plus de 50 000 $ (aucun faux négatif).
* **Precision (Précision)** : 24.89 %  
  Seulement un quart des prédictions « >50K » sont correctes, ce qui est attendu puisque le modèle prédit cette classe pour tous les individus.
* **F-score** : 29.29 %  
  Ce score combine la précision et le rappel, avec un poids plus important sur la précision (β = 0.5). La performance est donc faible mais attendue pour un modèle naïf.

Figure 10

1. **Creation d'un Pipeline pour l'Entrainement et la Prédiction**

Nous avons créé une fonction nommée train\_predict qui permet d’entraîner un modèle d’apprentissage supervisé sur un sous-ensemble des données, d’effectuer des prédictions, puis d’évaluer ses performances à l’aide de plusieurs métriques clés.

Cette fonction prend en entrée :

* Le modèle d’apprentissage choisi,
* Une taille d’échantillon des données d’entraînement,
* Les jeux de données d’apprentissage (X\_train, y\_train) et de test (X\_test, y\_test).

Elle réalise les étapes suivantes :

* Entraîne le modèle sur un sous-ensemble des données,
* Prédit les résultats sur les données de test et un petit échantillon d’entraînement,
* Mesure le temps d’entraînement et de prédiction,
* Évalue les performances à l’aide de la précision (accuracy) et du F-score sur l’entraînement et le test.

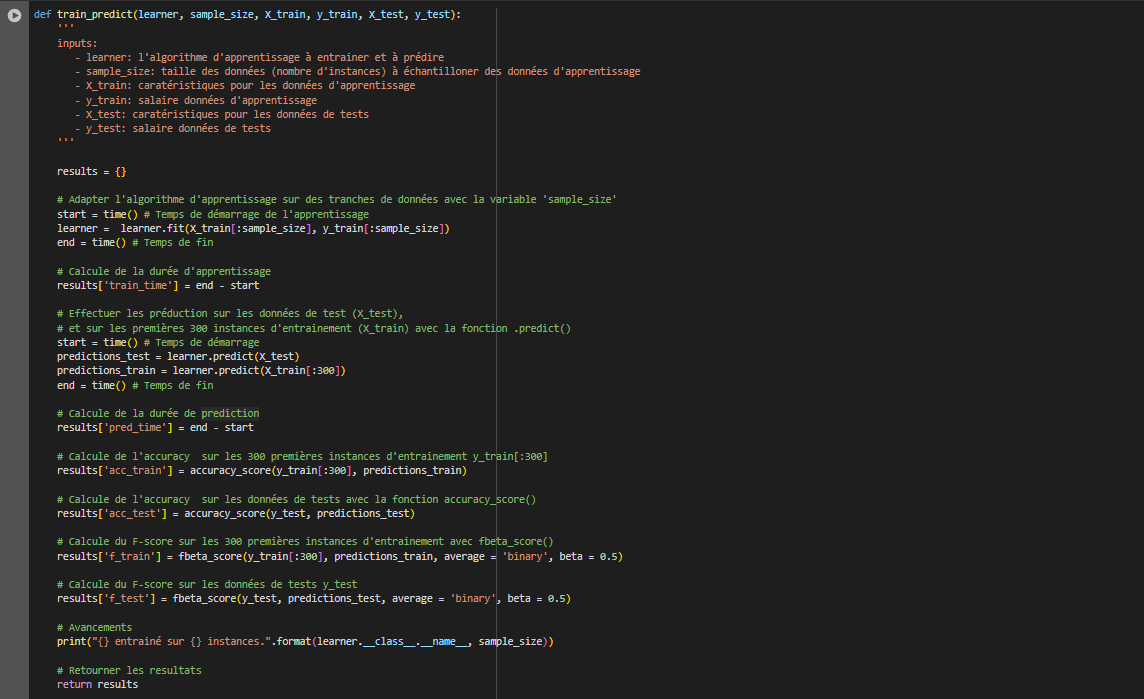
Les résultats sont retournés sous forme de dictionnaire pour une comparaison simple entre plusieurs modèles.

Figure 11

1. **Evaluation initiale des Modèles**

Afin d’identifier le modèle le plus performant pour prédire si un individu gagne plus ou moins de 50 000 dollars par an, nous avons évalué quatre algorithmes d’apprentissage supervisé : la régression logistique, le Random Forest, AdaBoost et le SVC (Support Vector Classifier). Ces modèles ont été testés selon plusieurs critères : la précision (accuracy), le F-score, ainsi que les temps d'entraînement et de prédiction. Une attention particulière a été portée à la capacité de généralisation de chaque modèle et à la vitesse d'exécution, critères essentiels dans un contexte applicatif réel.

Résultats Obtenus

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Modèle | Accuracy (Test) | F-score (Test) | Accuracy (Train) | F-score (Train) | Temps Entraînement (100% | Temps Prédiction (100%) | Remarques |
| Logistic Regression | 0.841 | 0.685 | 0.823 | 0.638 | 2.03 s | 0.025 s | Performances stables mais limitées |
| Random Forest | 0.840 | 0.679 | 0.980 | 0.952 | 6.32 s | 0.557 s | Très bon à l'entraînement, tendance au surapprentissage |
| AdaBoost | 0.849 | 0.704 | 0.840 | 0.674 | 1.71 s | 0.210 s | Excellent compromis entre performance et vitesse |
| SVC | 0.842 | 0.688 | 0.833 | 0.659 | 72.17 s | 29.28 s | Temps de calcul prohibitif |

À l’issue de cette évaluation, le modèle AdaBoost est retenu comme le meilleur compromis entre performance, généralisation et rapidité. Il surpasse les autres modèles tant sur les indicateurs de performance que sur l’efficacité computationnelle, ce qui en fait le choix le plus pertinent pour la suite du projet.

1. **Ajustement du Modèle**

L’optimisation du modèle AdaBoost a été réalisée grâce à une recherche exhaustive d’hyperparamètres (GridSearchCV). L’objectif était de maximiser le F-score à l’aide d’un estimateur de base de type DecisionTreeClassifier.

**Hyperparamètres testés:**

* n\_estimators : [50, 75, 100, 200]
* learning\_rate : [0.05, 0.1, 0.3, 1]
* estimator\_\_min\_samples\_split : [2, 4, 6]
* estimator\_\_max\_depth : [1, 2, 3]

Le F-score a été utilisé comme métrique d’évaluation à travers une validation croisée, via make\_scorer(fbeta\_score, beta=0.5).

La recherche a permis d’identifier le **meilleur classifieur** suivant :

AdaBoostClassifier(

estimator=DecisionTreeClassifier(max\_depth=3, min\_samples\_split=6),

n\_estimators=200,

learning\_rate=1,

random\_state=42

)

Ce modèle optimisé a été évalué sur les données de test :

| **Mesure** | **Avant optimisation** | **Après optimisation** |
| --- | --- | --- |
| Accuracy (test) | 0.849 | **0.861** |
| F-score (test) | 0.704 | **0.730** |
| Temps d’entraînement | 1.71 s | **15.78 s** |
| Temps de prédiction | 0.210 s | **0.68 s** |

On constate une amélioration significative du F-score, principal indicateur de performance ici, tout en conservant des temps de calcul raisonnables.

Une fois le modèle optimisé, nous avons examiné l’importance des différentes variables dans les décisions prises par l’algorithme. Les attributs les plus influents sont notamment :

* capital\_gain
* education\_num
* hours\_per\_week
* marital\_status
* relationship

Certaines variables ont une importance nulle ou négligeable, ce qui suggère qu’un nettoyage des attributs pourrait être envisagé pour simplifier le modèle sans perte de performance.

Un graphique ci-dessous illustre les variables ayant une importance strictement positive :

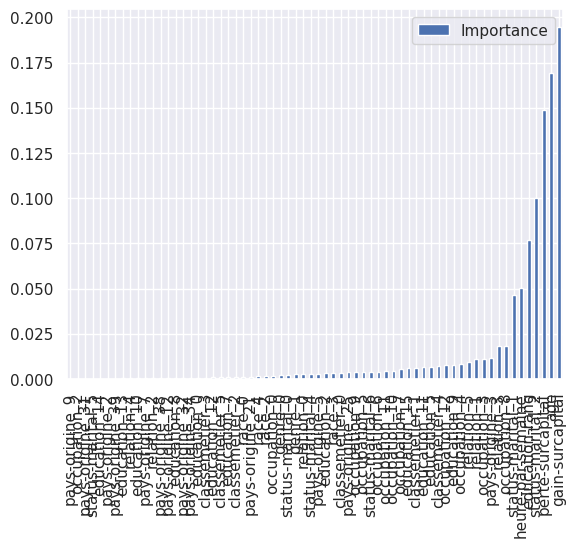


Figure 12

1. **Conclusion**

Au terme de cette étude, nous avons pu démontrer, à travers une démarche rigoureuse et progressive, la pertinence des techniques d’apprentissage supervisé dans la prédiction du niveau de revenu des individus à partir de données démographiques et socio-économiques.

Après un prétraitement minutieux du jeu de données (nettoyage, transformation, normalisation, encodage), plusieurs modèles ont été évalués selon des critères objectifs : précision, F-score (β = 0.5), temps d’entraînement et de prédiction. Parmi eux, le modèle AdaBoost s’est rapidement imposé comme le plus équilibré, offrant à la fois de bonnes performances de classification, une capacité de généralisation satisfaisante et une efficacité computationnelle adaptée à une mise en production.

L’optimisation fine de ce modèle via GridSearchCV a permis d’atteindre un F-score de 0.730 sur les données de test, contre 0.704 sans réglage, confirmant l’intérêt d’un ajustement hyperparamétrique précis. Le modèle final a également révélé les facteurs prédictifs les plus significatifs du revenu, tels que capital\_gain, education\_num et hours\_per\_week, apportant un éclairage complémentaire sur la dynamique socio-économique des individus.

1. **Lien vers le dépôt GitHub du projet :**  
   L’intégralité du projet (code source, traitements, visualisations et rapport) est disponible à l’adresse suivante : https://github.com/saraodile1/ML-Prevision-Revenus