

INSTITUT DE MATHEMATIQUES ET DES
SCIENCES PHYSIQUES

Master 1 TIC
2024-2024

Rapport TP Optimisation de l'apprentissage automatique

SOUS LA SUPERVISION DE :

ADJAHOU HOUÉFA SARA ODILE
MAMA ABDOU RAOUFOU

SOUS LA SUPERVISION DE :

DR BAH HABIB SIDI

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.

A. 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

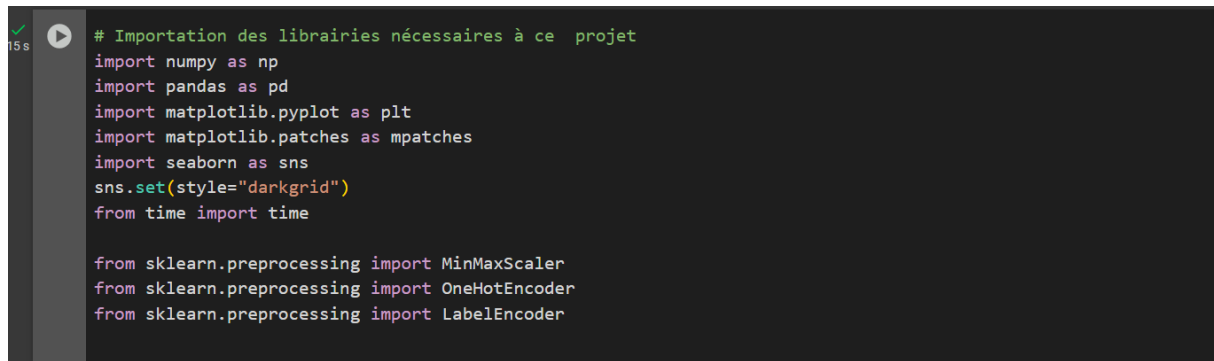
B. Tâches menées

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.

A screenshot of a Jupyter Notebook cell. On the left, there is a green checkmark icon and a play button icon, with '15 s' indicating execution time. The cell contains Python code for importing various libraries: numpy, pandas, matplotlib.pyplot, matplotlib.patches, seaborn, and time. It also shows the setting of a dark grid style for seaborn and the import of MinMaxScaler, OneHotEncoder, and LabelEncoder from sklearn.preprocessing.

```
# Importation des librairies nécessaires à ce projet
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
sns.set(style="darkgrid")
from time import time

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
```

Figure 1

2. 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'entraînement 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'entraînements** : □ 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.

```
# Données d'entraînement
# Ajoutons des noms de colonnes aux données
columns = ['age', 'classemetier', 'fnlwtgt', 'education', 'education-rang', 'status-marital', 'occupation',
           'relation', 'race', 'genre', 'gain-surcapital', 'perte-surcapital', 'heure-par-semaine', 'pays-origine', 'salaire']

# Lecture des données d'entraînement
adult_train = pd.read_csv('adult.data', header=None, names=columns, skipinitialspace=True)

# Suppression de la colonne sur le poids fnlwtgt non utilisé dans les analyses à suivre
adult_train = adult_train.drop('fnlwtgt', axis=1)

# Affichage de lignes et dimensions du jeux de données d'entraînements.
display(adult_train.head())
display(adult_train.shape)
```

	age	classemetier	education	education-rang	status-marital	occupation	relation	race	genre	gain-surcapital	perte-surcapital	heure-par-semaine	pays-origine	salaire
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

Figure 2

```
# Enlever '.' de la colonne salaire
adult_test['salaire'] = adult_test['salaire'].apply(lambda x: '>50K' if x=='>50K.' else '<=50K')

# Affichage de lignes et dimensions du jeux de données d'entraînements.
display(adult_test.head())
display(adult_test.shape)
```

	age	classemetier	education	education-rang	status-marital	occupation	relation	race	genre	gain-surcapital	perte-surcapital	heure-par-semaine	pays-origine	salaire
0	25	Private	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	?	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K

(16281, 14)

Figure 3

3. 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.

```

# Examiner si des valeurs sont manquantes
adult_test.info()

# Vérifier la présence du code de valeurs manquantes et convertir en NaNs
# Object récupère les colonnes dont les types objets sont renseignés comme valeurs. '?' par exemple
object_col = adult_train.select_dtypes(include=object).columns.tolist()
for col in object_col:
    print(adult_train[col].value_counts(dropna=False)/adult_train.shape[0], '\n')

# Convertir '?' en NaNs et supprimer les entrées avec des valeurs NaN
for col in object_col:
    adult_train.loc[adult_train[col]=='?', col] = np.nan
    adult_test.loc[adult_test[col]=='?', col] = np.nan

# Evaluer le taux de données manquantes dans le dataset.
col_missing_pct = adult_train.isna().sum()/adult_train.shape[0]
col_missing_pct.sort_values(ascending=False)

# Supprimer les entrées avec des données à valeurs manquantes
adult_train = adult_train.dropna(axis=0, how='any')
adult_test = adult_test.dropna(axis=0, how='any')

# Affichage des valeurs pour les jeux de données
print(f"Après suppression des valeurs manquantes : \ntrain : {adult_train.shape[0]} lignes \ntest : {adult_test.shape[0]} lignes")

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16281 entries, 0 to 16280
Data columns (total 14 columns):
#   Column      Non-Null Count  Dtype
---  -
0   age          16281 non-null   int64
1   classemetier 16281 non-null   object
2   education    16281 non-null   object

```

Figure 4

4. 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.
- 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.

```

# Nombre des entrées où les individus ont un salaire supérieur à $50,000
n_greater_50k = np.sum(adult_data.salaire >= 50000)

# Nombre des entrées où les individus ont un salaire inférieur à $50,000
n_at_most_50k = np.sum(adult_data.salaire <= 50000)

# Pourcentage d'individus avec un salaire de plus de $50,000
greater_percentage = round(np.mean(adult_data.salaire >= 50000) * 100.00, 2)

# Print the results
print("Total du nombre des entrées: {}".format(n_records))
print("Individus ayant plus de $50,000 de salaire: {}".format(n_greater_50k))
print("Individus ayant au plus $50,000 de salaire: {}".format(n_at_most_50k))
print("Pourcentage d'individus avec un salaire de plus de $50,000: {:.2f}%".format(greater_percentage))

```

Figure 5

5. Préparation des données

a) 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.

Distributions asymétriques de caractéristiques de données continues du recensement

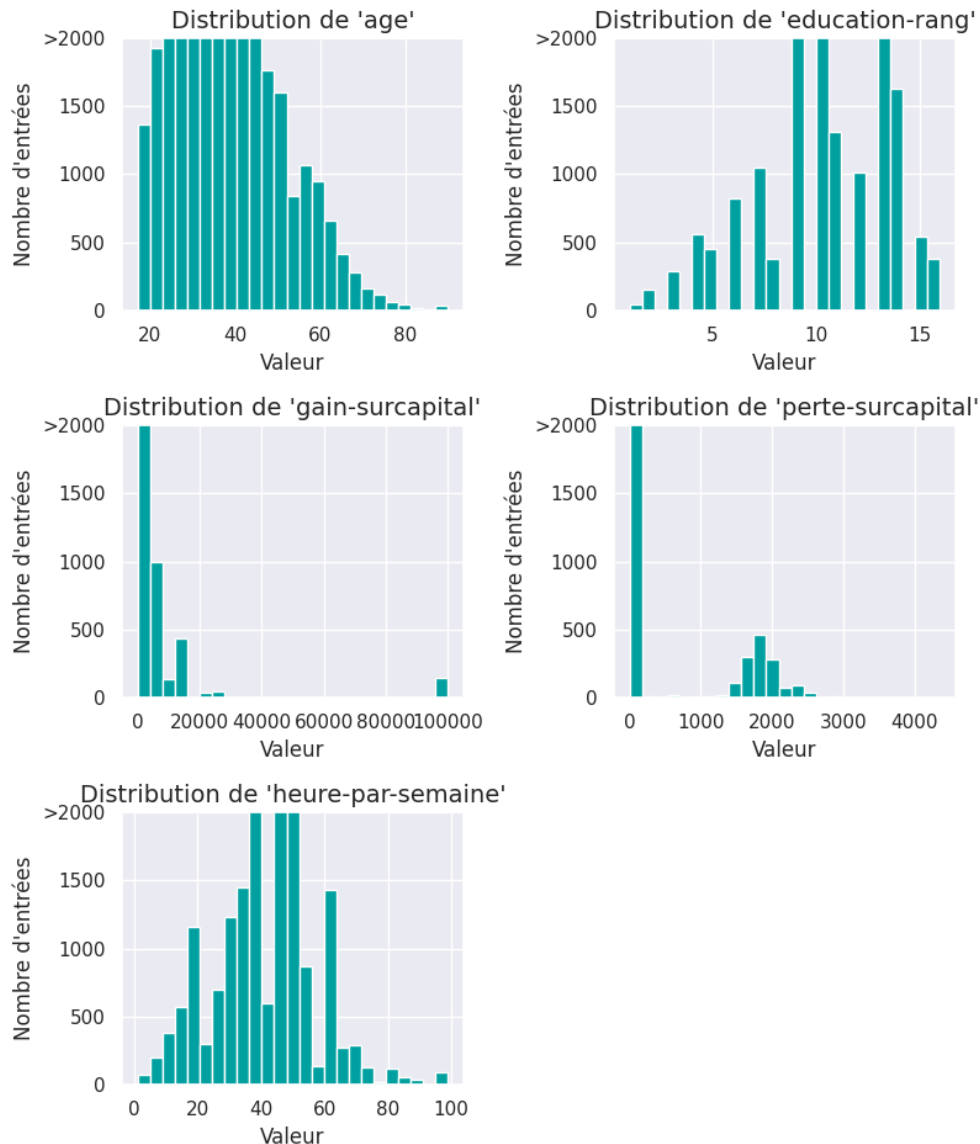


Figure 6

b) Normalisation des caractéristiques 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.

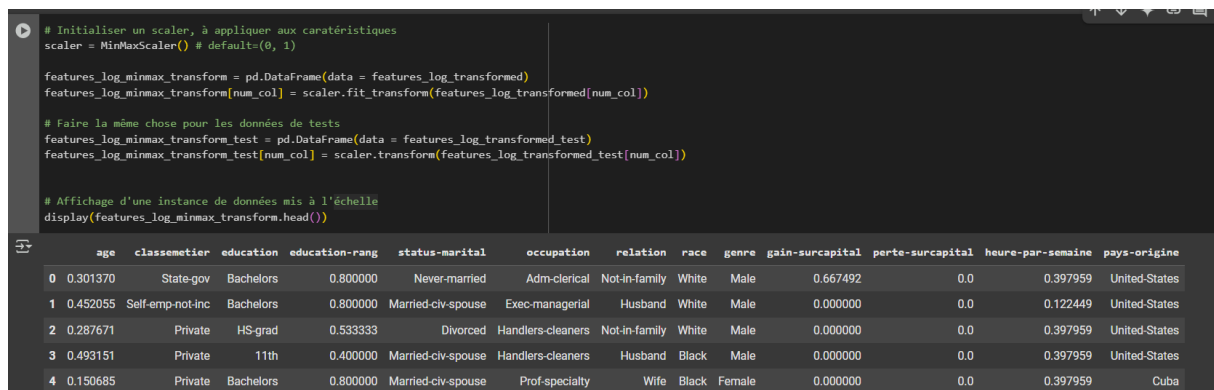


Figure 7

c) 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.


```

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Encodage One-hot des données 'features_log_minmax_transform' avec sklearn.OneHotEncoder

# Liste des données catégorielles
cat_feats = features_log_minmax_transform.dtypes[features_log_minmax_transform.dtypes=='object'].index.tolist()
cat_idx = [features_log_minmax_transform.columns.get_loc(col) for col in cat_feats]

labelencoder_feats = []
encoded_cat_features = []
# Encodage des labels.
for i, col in enumerate(cat_feats):
    labelencoder_feats.append(LabelEncoder(cat_feats[i]))
    if i == 0:
        encoded_cat_features = labelencoder_feats[i].fit_transform(features_log_minmax_transform.loc[:,col])
    else:
        encoded_cat_features = np.vstack([encoded_cat_features, labelencoder_feats[i].fit_transform(features_log_minmax_transform.loc[:,col])])

# création de l'encodeur
encoder = OneHotEncoder(handle_unknown='error', sparse_output=False)

# fit et transform de l'encodeur sur les caractéristiques numériques
encoded_cat_feats = encoder.fit_transform(np.array(encoded_cat_features).T)

# extractions des noms des caractéristiques à encoder
cat_col_name = features_log_minmax_transform.columns[cat_idx].tolist()
encoded_cat_feats_name = encoder.get_feature_names_out(cat_col_name)

# Génération du dataframe ONE pour concaténation avec le dataframe numérique par la suite.
encoded_cat_feats_df = pd.DataFrame(encoded_cat_feats, columns=encoded_cat_feats_name)
encoded_cat_feats_df.head()

[54] # Encodage des labels.
encoded_cat_features_tests = [labelencoder_feats[i].transform(features_log_minmax_transform_test.loc[:,col]) for i, col in enumerate(cat_feats)]

# Fit et transform de l'encodeur sur les caractéristiques numériques
encoded_cat_feats_test = encoder.transform(np.array(encoded_cat_features_tests).T)

# extractions des noms des caractéristiques à encoder
cat_col_name = features_log_minmax_transform.columns[cat_idx].tolist()
encoded_cat_feats_name = encoder.get_feature_names_out(cat_col_name)

# Génération du dataframe ONE pour concaténation avec le dataframe numérique par la suite.
encoded_cat_feats_df_test = pd.DataFrame(encoded_cat_feats_test, columns=encoded_cat_feats_name)
encoded_cat_feats_df_test.head()

```

Figure 8

d) 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 :
 $>50K \rightarrow 1,$
 $\leq 50K \rightarrow 0,$
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.

```

# Extraction du dataframe avec uniquement les données numériques.
num_feats_df = features_log_mimex_transform(num_col).reset_index()

# Concaténation des données numériques avec les données catégorielles encodées.
X_train = pd.merge(num_feats_df, encoded_cat_feats_df, left_index=True, right_index=True).drop('index', axis=1)

# Encoder les 'salaire_brut' en valeurs numériques
y_train = salaire_brut.apply(lambda x: 1 if x == '>50K' else 0)

print(f"Caractéristiques au total après l'encodage one-hot: {len(X_train.columns)}")

# Affichage de quelques lignes du dataframe prétraité.
X_train.head()

```

	age	education- rang	gain- surcapital	perte- surcapital	heure-par- semaine	classometier_0	classometier_1	classometier_2	classometier_3	classometier_4	...	pays- origine_11	pays- origine_12	pays- origine_13	pays- origine_14	pays- origine_15	pays- origine_16	pays- origine_17	pays- origine_18	
0	0.301370	0.800000	0.667492	0.0	0.397959	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.452065	0.800000	0.000000	0.0	0.122449	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	0.287671	0.520333	0.000000	0.0	0.397959	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0.493151	0.400000	0.000000	0.0	0.397959	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	0.150685	0.800000	0.000000	0.0	0.397959	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows x 103 columns

```

# Extract the dataframe with only numerical features
num_feats_df_test = features_log_mimex_transform_test(num_col).reset_index()

# Concatenate numerical and encoded categorical features together
X_test = pd.merge(num_feats_df_test, encoded_cat_feats_df_test, left_index=True, right_index=True).drop('index', axis=1)

# Encode the 'income_raw' to numerical values
y_test = salaire_brut_test.apply(lambda x: 1 if x == '>50K' else 0)

print(f"Caractéristiques au total après l'encodage one-hot: {len(X_test.columns)}")

# Display several rows of processed dataframe
X_test.head()

```

103 caractéristiques au total après l'encodage one-hot.

Figure 9

6. 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 ($\beta = 0.5$). La performance est donc faible mais attendue pour un modèle naïf.

```

# Calcul de l'accuracy, la précision et le recall
accuracy = np.sum(y_train) / y_train.count()
recall = np.sum(y_train) / np.sum(y_train)
precision = np.sum(y_train) / y_train.count()

print(recall)
print(accuracy)
print(precision)

# Calcul de F-score avec beta = 0.5 et avec les valeurs des précision et recall.
fscore = (1 + 0.5*0.5) * precision * recall / (0.5*0.5*precision + recall)
print(fscore)

```

1.0

Figure 10

7. Creation d'un Pipeline pour l'Entraînement 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.

```
def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    """
    inputs:
    - learner: l'algorithme d'apprentissage à entraîner et à prédire
    - sample_size: taille des données (nombre d'instances) à échantillonner des données d'apprentissage
    - X_train: caractéristiques pour les données d'apprentissage
    - y_train: salaire données d'apprentissage
    - X_test: caractéristiques pour les données de tests
    - y_test: salaire données de tests
    """
    results = {}

    # Adapter l'algorithme d'apprentissage sur des tranches de données avec la variable 'sample_size'
    start = time() # Temps de démarrage de l'apprentissage
    learner = learner.fit(X_train[:sample_size], y_train[:sample_size])
    end = time() # Temps de fin

    # Calcul de la durée d'apprentissage
    results['train_time'] = end - start

    # Effectuer les prédictions sur les données de test (X_test),
    # et sur les premières 300 instances d'entraînement (X_train) avec la fonction .predict()
    start = time() # Temps de démarrage
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train[:300])
    end = time() # Temps de fin

    # Calcul de la durée de prédiction
    results['pred_time'] = end - start

    # Calcul de l'accuracy sur les 300 premières instances d'entraînement y_train[:300]
    results['acc_train'] = accuracy_score(y_train[:300], predictions_train)

    # Calcul de l'accuracy sur les données de tests avec la fonction accuracy_score()
    results['acc_test'] = accuracy_score(y_test, predictions_test)

    # Calcul du F-score sur les 300 premières instances d'entraînement avec fbeta_score()
    results['f_train'] = fbeta_score(y_train[:300], predictions_train, average = 'binary', beta = 0.5)

    # Calcul du F-score sur les données de tests y_test
    results['f_test'] = fbeta_score(y_test, predictions_test, average = 'binary', beta = 0.5)

    # Avancements
    print("{} entraîné sur {} instances.".format(learner.__class__.__name__, sample_size))

    # Retourner les résultats
    return results
```

Figure 11

8. 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.

9. 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 :

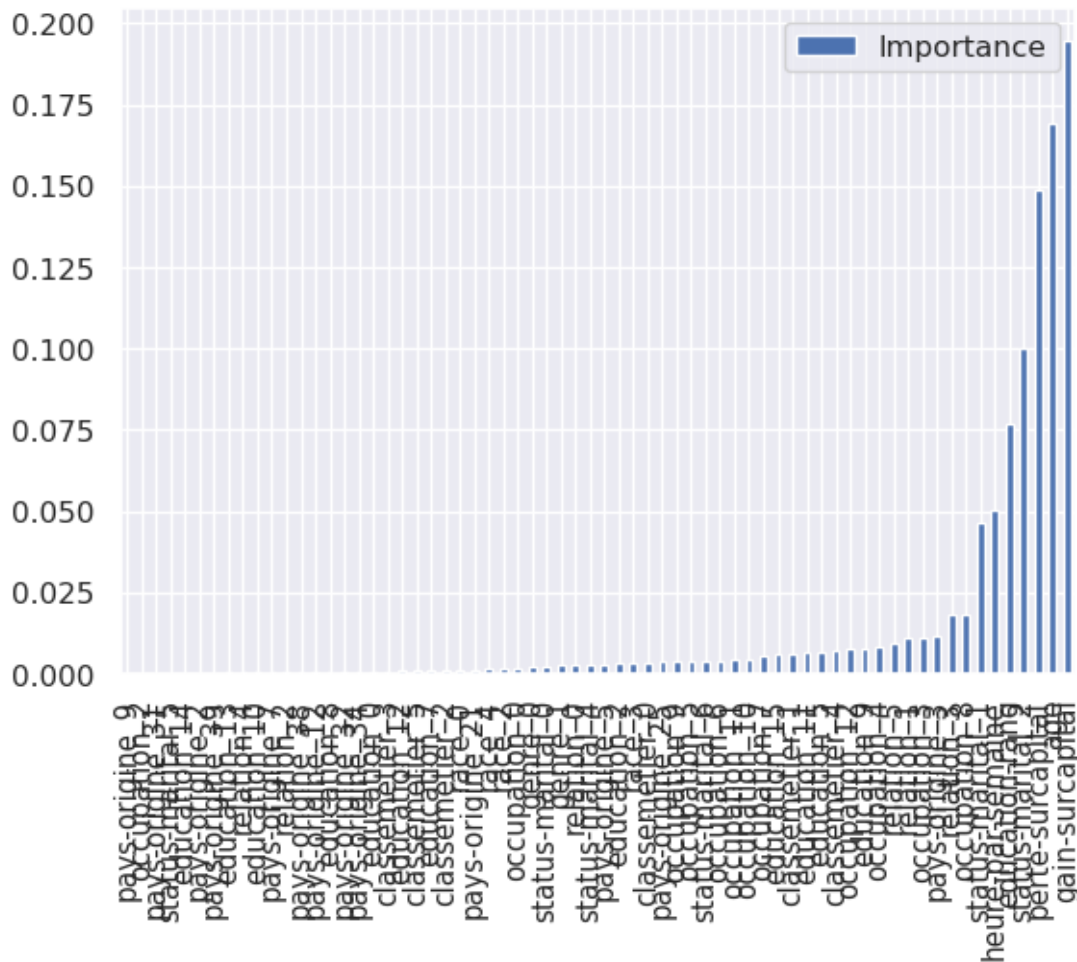


Figure 12

C. 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 ($\beta = 0.5$), 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.

D. 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>