Université Paris Dauphine

Projet Master
Machine Learning Project
Rapport

Master Statistiques et Big Data

Élèves :
Natacha Babalola

Enseignant :
Fabrice Rossi

16 mai 2024

Table des matières

Introduction

Pour ce projet, nous avons entrepris une approche supervisée en utilisant les ensembles de données fournis. Nous avons d'abord effectué un prétraitement des données, comprenant l'analyse des valeurs manquantes, suivi d'une fusion pour obtenir un fichier final pour les ensembles d'entraînement et de test. Avec un total de 99 985 données, nous avons pris soin d'évaluer chaque modèle en fonction de sa pertinence et de sa capacité à absorber les variances, en tenant compte de la quantité de données.

Pour exploiter au maximum nos données, nous avons procédé à un encodage afin d'utiliser la quasi-totalité des colonnes pour l'entraînement du modèle. Nous avons exploré plusieurs modèles, notamment la régression linéaire, le Random Forest, le K-Nearest Neighbors, le Support Vector Machine, l'approche par Ridge, un ajustement via un modèle polynomiale, et enfin le GradientBoostingRegressor.

Le modèle final obtenu à l'aide du GradientBoosting, avec les meilleurs hyperparamètres ajustés, a abouti à un MSE approchant 1474.40, alors que nous avons commencé avec un MSE de 20 197.93.

À la fin du projet, nous pourrons exporter deux fichiers de prévisions : le premier, Prediction_merge, contiendra les résultats de chaque modèle sur l'ensemble d'entraînement, tandis que le fichier Prevision.CSV regroupera les prédictions du modèle sur l'ensemble de test.

Pour les prédictions sur l'ensemble de Test, notre objectif est d'obtenir un MSE proche de celui obtenu lors de la phase d'entraînement, ainsi qu'un R2 de proche des 92% obtenu en utilisant votre modèle final de GradientBoosting sur l'ensemble de test de notre ensemble d'entraînement. Ce coefficient de dertermination signifie que votre modèle a bien performé sur ces données spécifiques. Cependant, le fait d'obtenir un R2 élevé sur l'ensemble de test de l'ensemble d'entraînement ne garantit pas nécessairement que nous obtiendrons le même R2 sur de nouvelles données.

Lorsque nous appliquons notre modèle à de nouvelles données (ensemble de test final pour lequel nous n'avons pas les vraies valeurs), il est possible que la relation entre nos variables indépendantes et la variable cible soit différente. Par conséquent, il est difficile de prédire exactement quel R2 nous obtiendrons sur ces nouvelles données. Nous attendons à ce que notre modèle performe relativement bien sur de nouvelles données similaires à celles sur lesquelles il a été entraîné.

1 Chargement des fichiers

La première chose à faire pour tous les projets d'analyse de données est d'importer les modules et les packages pertinents :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
```

```
import pandas as pd

# Charger les fichiers CSV
learn_dataset = pd.read_csv("learn_dataset.csv")
test_dataset = pd.read_csv("test_dataset.csv")
learn_dataset_job = pd.read_csv("learn_dataset_job.csv")
test_dataset_job = pd.read_csv("test_dataset_job.csv")
learn_dataset_emp = pd.read_csv("learn_dataset_emp.csv")
test_dataset_emp = pd.read_csv("test_dataset_emp.csv")
learn_dataset_CLUB = pd.read_csv("learn_dataset_sport.csv")
test_dataset_CLUB = pd.read_csv("test_dataset_sport.csv")
```

2 Analyse Descriptive

2.1 Traiter les données manquantes et les valeurs abbérantes

On remplace les données manquantes par la moyenne des variables présente : :

```
# Afficher un résumé des données
summary = learn_dataset.describe()

# Afficher le résumé
print(summary)
```

3 Fusion des données

3.1 Données d'apprentissages

La première chose à faire pour tous les projets d'analyse de données est d'importer les modules et les packages pertinents :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#!pip install pandas_profiling
#!pip install --upgrade pip
#! pip install lightgbm
import plotly.express as px
import plotly.graph_objects as go
from ipywidgets import interact,widgets
import folium
```

3.2 Tables utilisées

Les données étudiées dans ce projet sont stockées dans plusieurs fichiers CSV documentés ci-dessous. Ils concernent 99985 Français (de France métropolitaine).

L'objectif du projet est de construire un modèle prédictif pour la variable cible. Cette variable représente une caractéristique non divulguée liée au statut socio-économique de la personne pris au sens large.

L'ensemble de données a été divisé en un ensemble d'apprentissage comptant 49 993 personnes et un ensemble de tests comptant 49 992 personnes. Chaque ensemble est décrit par quatre fichiers csv : :

- learn_dataset.csv et test_dataset.csv contiennent la description principale des personnes dans l'ensemble de données, avec la variable cible uniquement dans learn_dataset.csv ;

- learn_datataset_job.csv et test_datataset_job.csv décrivent les emplois des personnes ayant un statut de salarié ;

- learn_dataset_emp.csv et test_dataset_emp.csv donnent le type d'emploi de chaque personne occupant un emploi (y compris les non-salariés) ;

- learn_dataset_CLUB.csv et test_dataset_CLUB.csv contiennent des informations sur les personnes inscrites dans des clubs sportifs.

```
# Vérifier s'il y a des doublons dans la colonne 'Uid' de chaque DataFrame
print("Nombre de doublons dans learn_dataset : ", learn_dataset.duplicated(subset=['U
print("Nombre de doublons dans learn_datataset_job : ", learn_dataset_job.duplicated(
print("Nombre de doublons dans learn_dataset_emp : ", learn_dataset_emp.duplicated(su
print("Nombre de doublons dans learn_dataset_CLUB : ", learn_dataset_CLUB.duplicated(

# Supprimer les doublons dans la colonne 'Uid' de chaque DataFrame
learn_dataset = learn_dataset.drop_duplicates(subset=['Uid'])
learn_dataset_job = learn_dataset_job.drop_duplicates(subset=['Uid'])
learn_dataset_emp = learn_dataset_emp.drop_duplicates(subset=['Uid'])
learn_dataset_CLUB = learn_dataset_CLUB.drop_duplicates(subset=['Uid'])

# Fusionner les DataFrames sans doublons au niveau de la colonne de jointure
learn_dataset_merge = pd.merge(learn_dataset, learn_dataset_job, on='Uid', how='outer
learn_dataset_merge = pd.merge(learn_dataset_merge, learn_dataset_emp, on='Uid', how=
learn_dataset_merge = pd.merge(learn_dataset_merge, learn_dataset_CLUB, on='Uid', how

# Supprimer les doublons de la colonne 'Uid' après la fusion
learn_dataset_merge = learn_dataset_merge.drop_duplicates(subset=['Uid'])

# Afficher les informations sur le DataFrame fusionné
print(learn_dataset_merge.info())
```

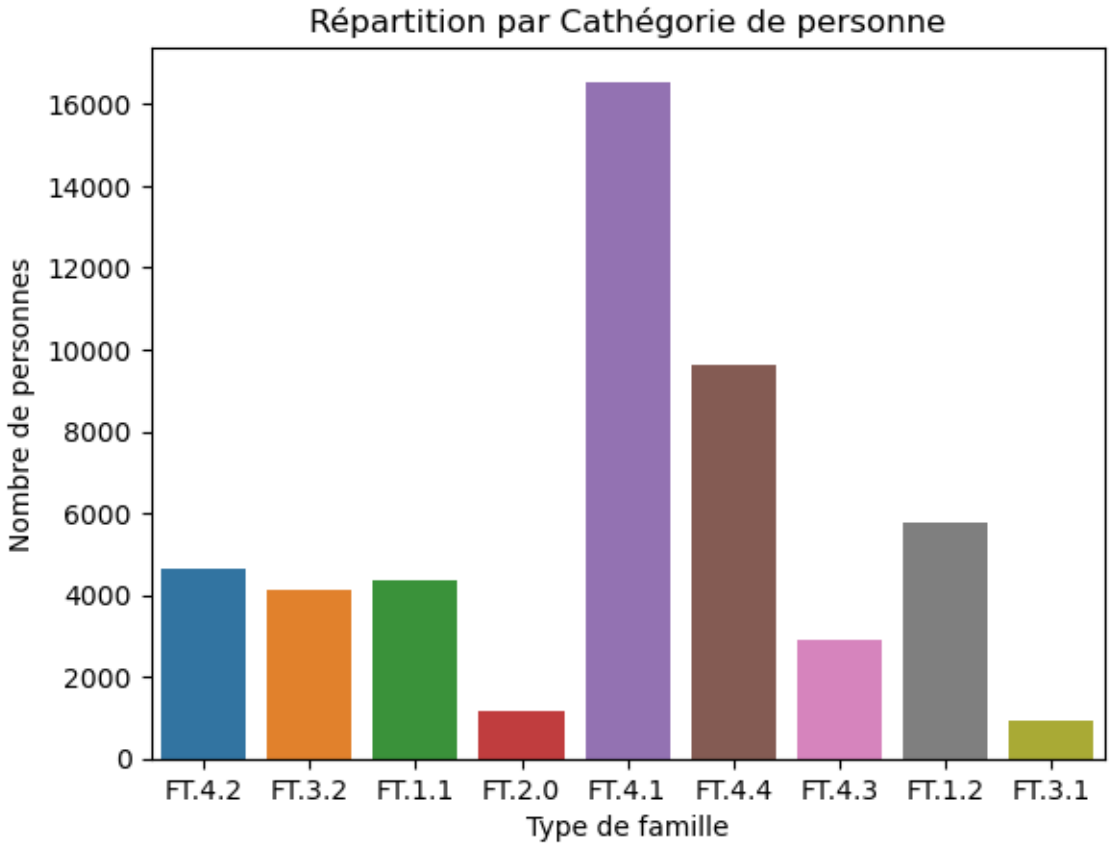
3.3 Données de test

```
# Vérifier s'il y a des doublons dans la colonne 'Uid' de chaque DataFrame
print("Nombre de doublons dans test_dataset : ", test_dataset.duplicated(subset=['Uid
print("Nombre de doublons dans test_dataset_job : ",test_dataset_job.duplicated(subse
print("Nombre de doublons dans test_dataset_emp : ", test_dataset_emp.duplicated(subs
print("Nombre de doublons dans test_dataset_CLUB : ", test_dataset_CLUB.duplicated(su

# Supprimer les doublons dans la colonne 'Uid' de chaque DataFrame
test_dataset = test_dataset.drop_duplicates(subset=['Uid'])
test_dataset_job = test_dataset_job.drop_duplicates(subset=['Uid'])
test_dataset_emp = test_dataset_emp.drop_duplicates(subset=['Uid'])
test_dataset_CLUB = test_dataset_CLUB.drop_duplicates(subset=['Uid'])

# Fusionner les DataFrames sans doublons au niveau de la colonne de jointure
test_dataset_merge = pd.merge(test_dataset, test_dataset_job, on='Uid', how='outer')
test_dataset_merge = pd.merge(test_dataset_merge, test_dataset_emp, on='Uid', how='ou
test_dataset_merge = pd.merge(test_dataset_merge, test_dataset_CLUB, on='Uid', how='o

# Supprimer les doublons de la colonne 'Uid' après la fusion
test_dataset_merge = test_dataset_merge.drop_duplicates(subset=['Uid'])

# Afficher les informations sur le DataFrame fusionné
print(test_dataset_merge.info())
```

4 Exploration des données

4.1 Analyse exploratoire des données

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
sns.countplot(x='Familty_type', data=learn_dataset)
plt.xlabel('Type de famille')
plt.ylabel('Nombre de personnes')
plt.title('Répartition par Cathégorie de personne')
plt.show()
```

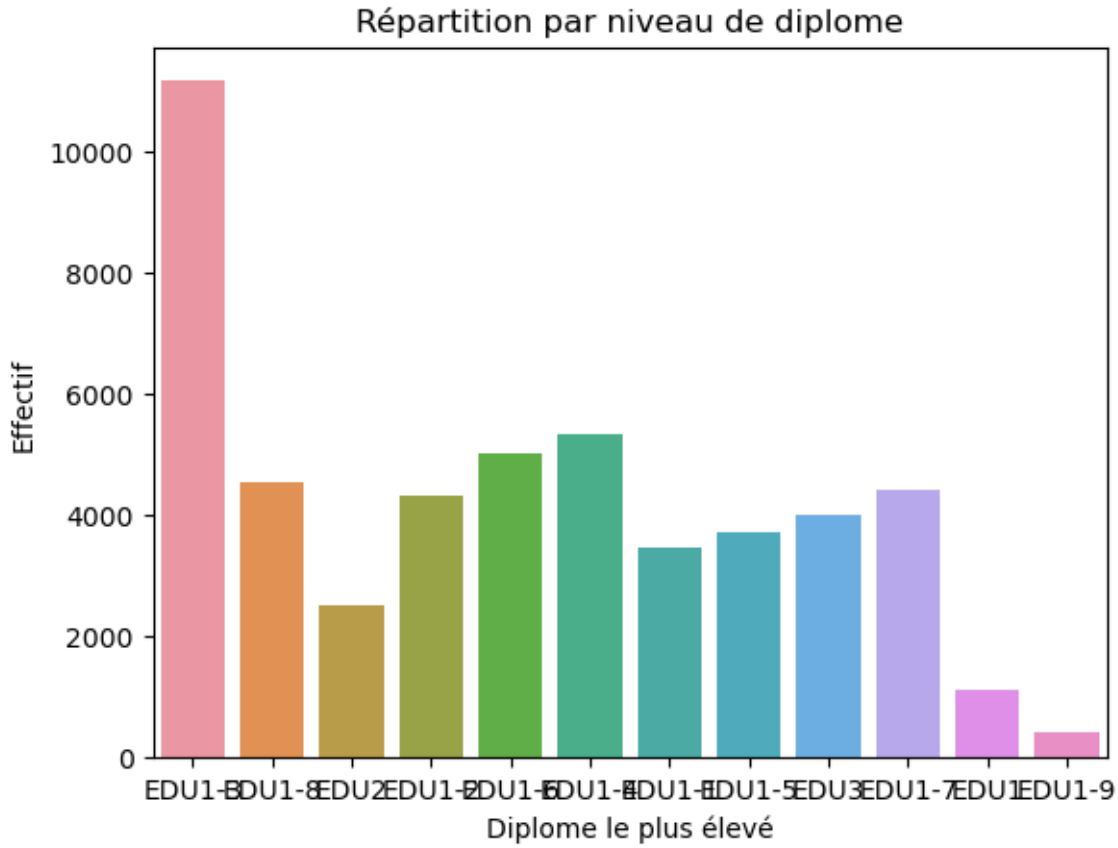



```
#Dimension des matrices ( Tables) de données
print(df_death.shape)
print(df_conf.shape)
print(etat_df.shape)
```

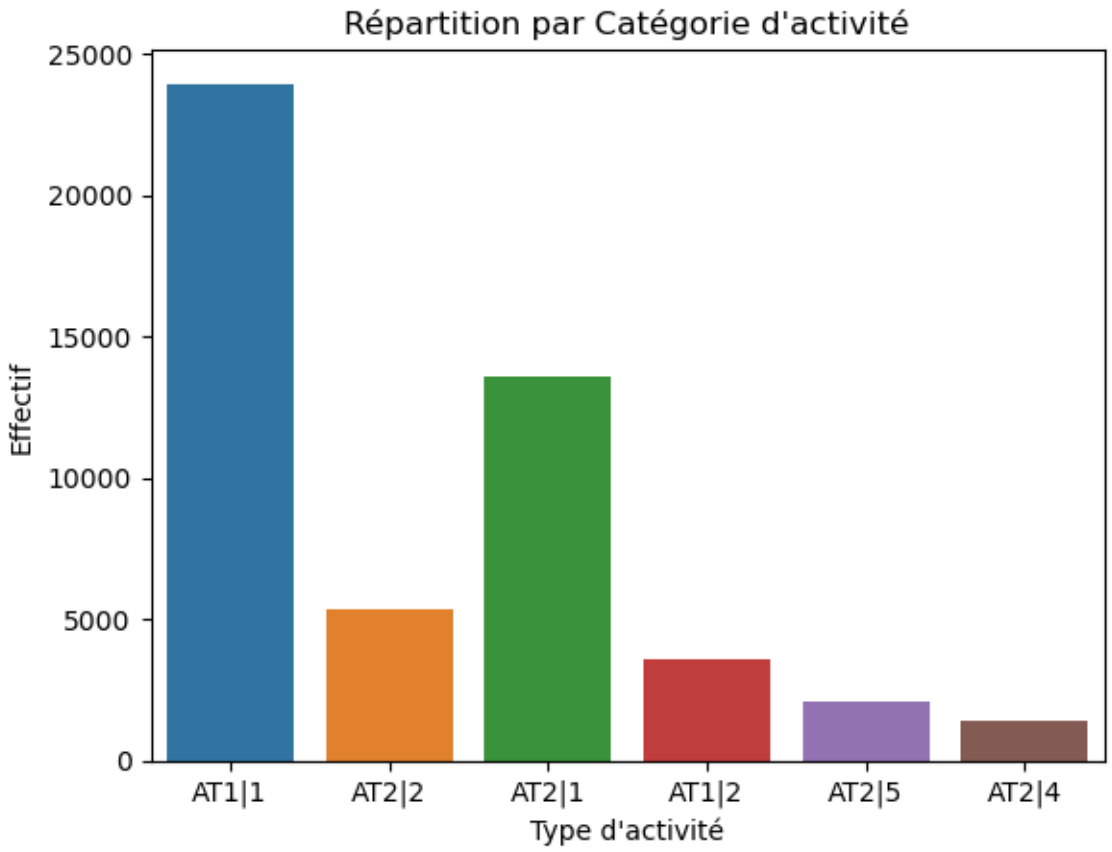
Commentaire

Notre jeu de données est composé principalement sinon en majorité de Famille formé d'un couple de 2 "actifs ayant un emploi" et de Famille formé d'un couple d'aucun "actif ayant un emploi". Ils forment à eux 2 près de 50% de notre jeu de données.

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

sns.countplot(x='Highest_diploma', data=learn_dataset)
plt.xlabel('Diplome le plus élevé')
plt.ylabel('Effectif')
plt.title('Répartition par niveau de diplome')
plt.show()
```



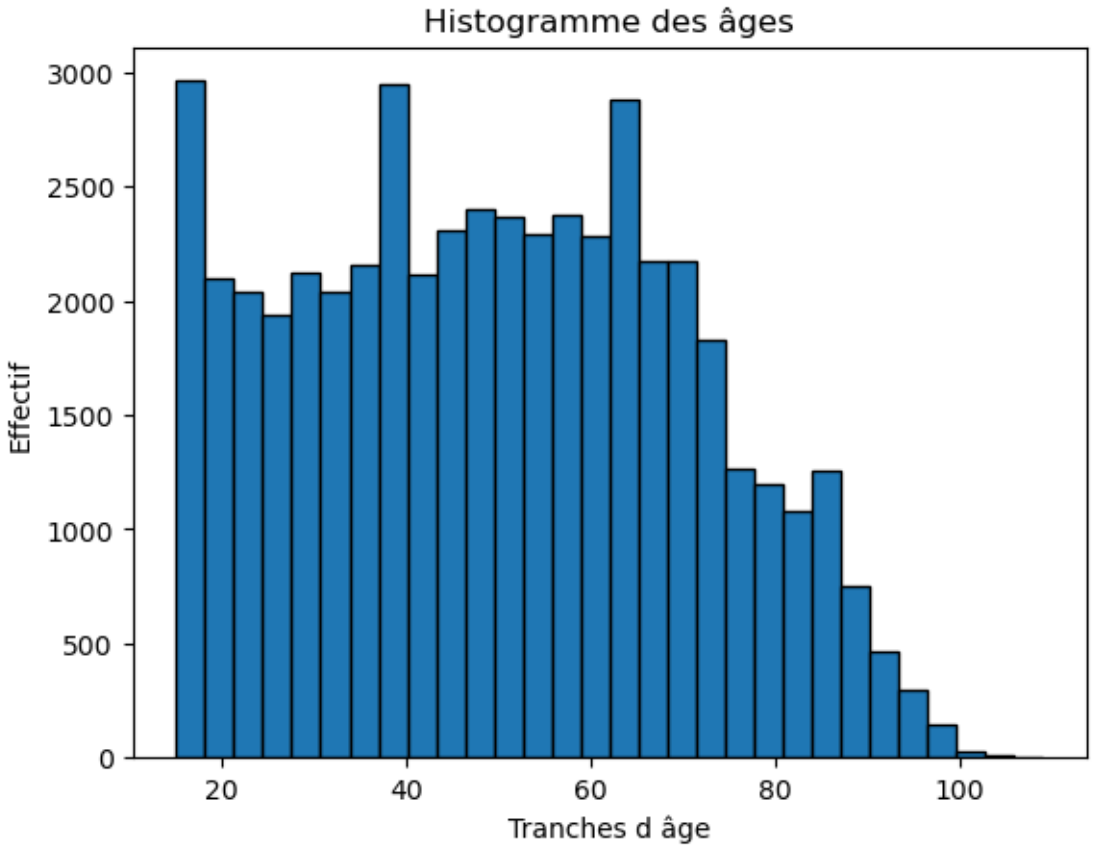

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

sns.countplot(x='ACT', data=learn_dataset)
plt.xlabel('Type d\'activité')
plt.ylabel('Effectif')
plt.title('Répartition par Catégorie d\'activité')
plt.show()
```



Commentaire

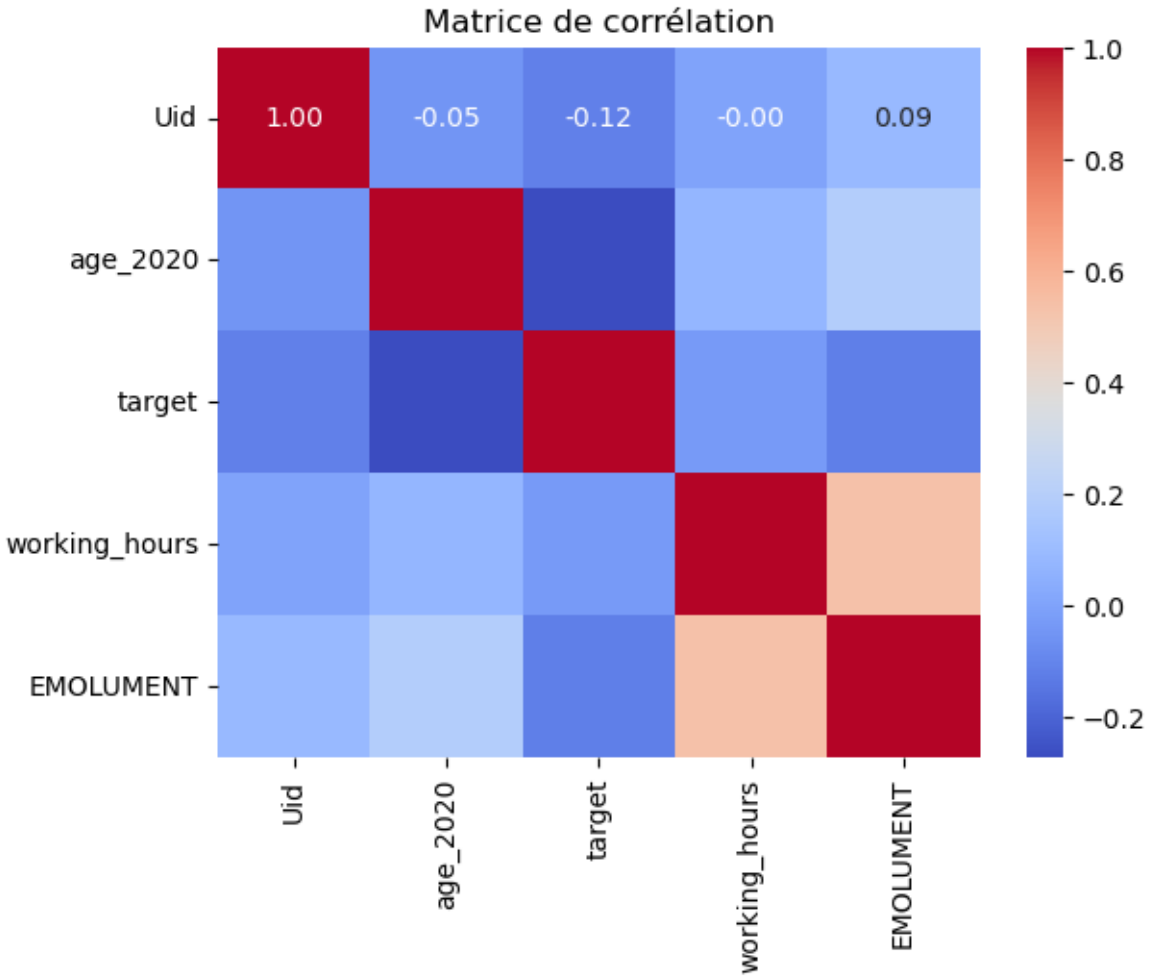
La population est majoritairement constituée de AT1/1 (personnes actifs ayant un emploi, y compris sous apprentissage ou en stage) et de AT2/1 (personnes retraités ou préretraités)

```
import numpy as np
import matplotlib.pyplot as plt

# Générer des données aléatoires
#data = np.random.randn(1000)

# Calculer l'histogramme
# Convertir la série 'age_2020' en un tableau NumPy
age_array = learn_dataset['age_2020'].values
hist, bins = np.histogram(age_array, bins=30)

# Afficher l'histogramme
plt.hist(age_array, bins=bins, edgecolor='black')
plt.xlabel("Tranches d âge")
plt.ylabel('Effectif')
plt.title('Histogramme des âges')
plt.show()
```



4.2 Visualisation des données

```
# Sélectionner uniquement les colonnes numériques
numeric_columns = learn_dataset_merge.select_dtypes(include=[np.number])

# Calculer la matrice de corrélation
correlation_matrix = numeric_columns.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Matrice de corrélation')
plt.show()
```



Commentaires :

Nous observons une corrélation négative modérée entre nos deux variables âge et la varibale cible (-0,27) :

- **Valeur négative** : Le signe négatif indique une relation inverse entre les variables. Cela signifie que lorsque la valeur de l'une des variables augmente, la valeur de l'autre variable diminue, et vice versa.

- **Modérée** : : Une corrélation de -0,27 n'est pas très forte, mais elle indique tout de même une relation significative entre les deux variables. Elle suggère qu'il y a une tendance pour que lorsque l'âge augmente, la valeur de la variable cible diminue légèrement, et vice versa.

Dans le contexte de notre analyse, cela pourrait signifier que l'âge a une influence modérée sur la variable cible. Par exemple, dans un problème de prédiction de revenu, une corrélation négative modérée entre l'âge et le revenu pourrait signifier que les personnes plus âgées ont tendance à avoir des revenus légèrement plus faibles, Dans le cas espèce, nous ne pouvons pas aller plus loin, la variable cible étant inconnu.

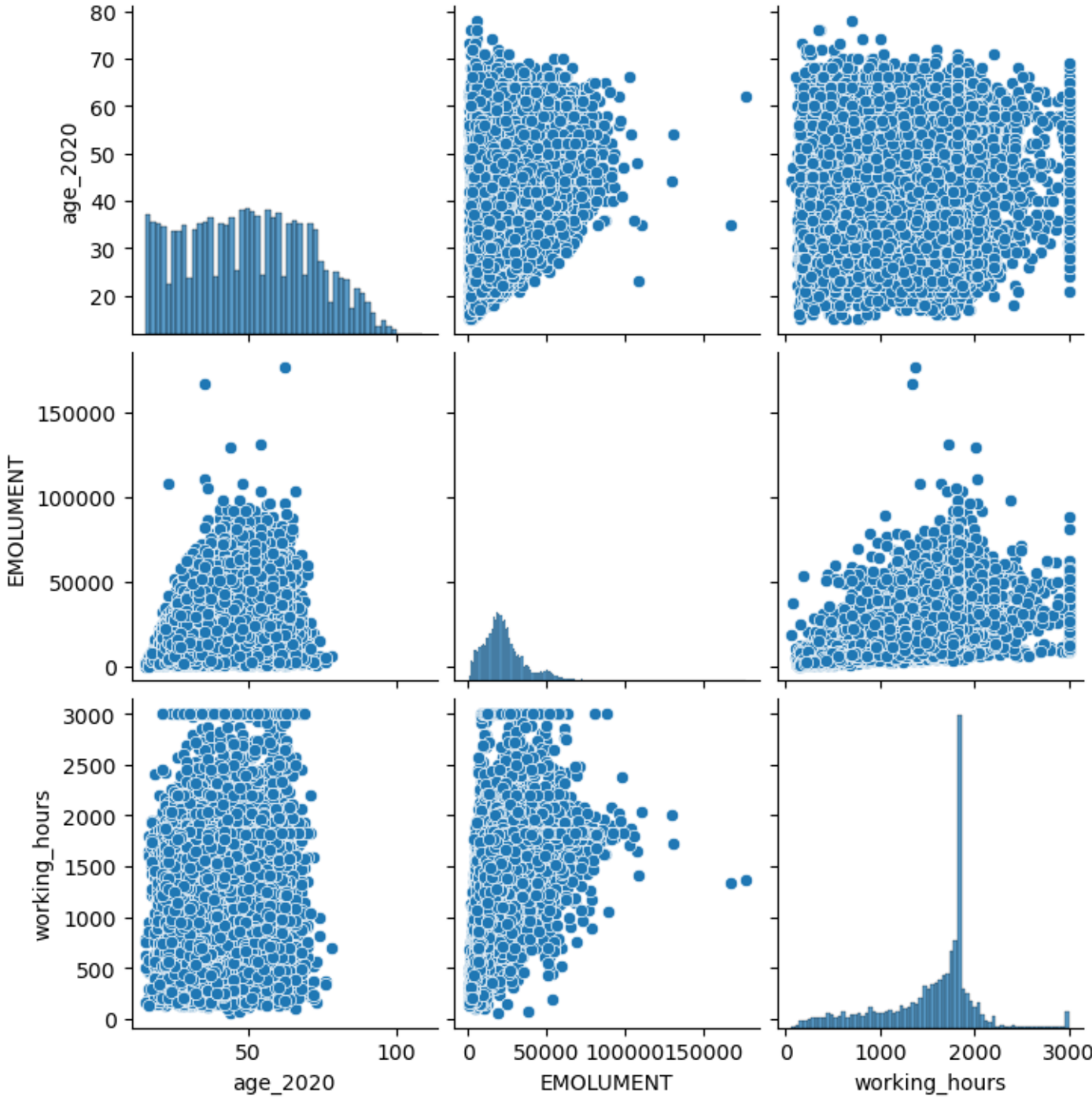
Nous observons également une corrélation positive entre nos deux variables Emolument et Working_hours (0,53) :

- **Valeur positive** : Le signe positif indique une relation linéaire entre les variables. Cela signifie que lorsque la valeur de l'une des variables augmente, la valeur de l'autre variable augmente également, et vice versa.

- **Modérée** : Une corrélation de 0,53 n'est pas très forte, mais elle indique tout de même une relation significative entre les deux variables.

Dans ce cas, une corrélation de 0,53 entre le salaire annuel et le nombre total d'heures de travail suggère que, en général, les personnes gagnant des salaires annuels plus élevés ont tendance à travailler un plus grand nombre d'heures par an.

```
sns.pairplot(learn_dataset_merge[['age_2020', 'EMOLUMENT', 'working_hours']])
plt.show()
```



Commentaire

On comprend que le niveau de rémuneration augmente lorsque le niveau de rémunération augmente avec l'âge

4.3 Encodage des colonnes

```
from sklearn.preprocessing import LabelEncoder

# Initialiser un objet LabelEncoder
le = LabelEncoder()

# Appliquer le LabelEncoder à la colonne 'animal'


learn_dataset_merge['WORK_CONDITION'] = le.fit_transform(learn_dataset_merge['WORK_CO
learn_dataset_merge['Job_42'] = le.fit_transform(learn_dataset_merge['Job_42'])
learn_dataset_merge['ACT'] = le.fit_transform(learn_dataset_merge['ACT'])
learn_dataset_merge['CLUB'] = le.fit_transform(learn_dataset_merge['CLUB'])
learn_dataset_merge['job_category'] = le.fit_transform(learn_dataset_merge['job_categ
learn_dataset_merge['TERMS_OF_EMP'] = le.fit_transform(learn_dataset_merge['TERMS_OF_
learn_dataset_merge['sex'] = le.fit_transform(learn_dataset_merge['sex'])
learn_dataset_merge['emp'] = le.fit_transform(learn_dataset_merge['emp'])
learn_dataset_merge['Job_desc'] = le.fit_transform(learn_dataset_merge['Job_desc'])
learn_dataset_merge['Familty_type'] = le.fit_transform(learn_dataset_merge['Familty_t
learn_dataset_merge['Highest_diploma'] = le.fit_transform(learn_dataset_merge['Highes
learn_dataset_merge['Eco_sect'] = le.fit_transform(learn_dataset_merge['Eco_sect'])
learn_dataset_merge['Employer_category'] = le.fit_transform(learn_dataset_merge['Empl
learn_dataset_merge['employee_count'] = le.fit_transform(learn_dataset_merge['employe


# Afficher le résultat
#print(learn_dataset_merge)
```

```
from sklearn.preprocessing import LabelEncoder

# Initialiser un objet LabelEncoder
le = LabelEncoder()


test_dataset_merge['WORK_CONDITION'] = le.fit_transform(test_dataset_merge['WORK_COND
test_dataset_merge['Job_42'] = le.fit_transform(test_dataset_merge['Job_42'])
test_dataset_merge['ACT'] = le.fit_transform(test_dataset_merge['ACT'])
test_dataset_merge['CLUB'] = le.fit_transform(test_dataset_merge['CLUB'])
test_dataset_merge['job_category'] = le.fit_transform(test_dataset_merge['job_categor
test_dataset_merge['TERMS_OF_EMP'] = le.fit_transform(test_dataset_merge['TERMS_OF_EM
test_dataset_merge['sex'] = le.fit_transform(test_dataset_merge['sex'])
test_dataset_merge['emp'] = le.fit_transform(test_dataset_merge['emp'])
test_dataset_merge['Job_desc'] = le.fit_transform(test_dataset_merge['Job_desc'])
test_dataset_merge['Familty_type'] = le.fit_transform(test_dataset_merge['Familty_typ
test_dataset_merge['Highest_diploma'] = le.fit_transform(test_dataset_merge['Highest_
test_dataset_merge['Eco_sect'] = le.fit_transform(test_dataset_merge['Eco_sect'])
test_dataset_merge['Employer_category'] = le.fit_transform(test_dataset_merge['Employ
test_dataset_merge['employee_count'] = le.fit_transform(test_dataset_merge['employee_


# Afficher le résultat
#print(test_dataset_merge)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

```
Learn_encoded_num.head()
```

	Uid	Familty_type	Highest_diploma	insee_code	Job_42	ACT	sex	Studying	age_2020	target	...	working_hours	JOB_DEP	Eco_sect	Employer_(
0	1	FT.4.2	EDU1-3	01002	csp_5_2	AT1|1	Male	False	56	334.988013	...	NaN	NaN	NaN	
1	3	FT.3.2	EDU1-3	01004	csp_6_8	AT1|1	Male	True	19	385.059215	...	607.0	69	GZ	
2	4	FT.1.1	EDU1-3	01004	csp_5_4	AT1|1	Male	False	47	311.387009	...	1792.0	71	FZ	
3	6	FT.4.2	EDU1-8	01004	csp_4_6	AT1|1	Male	False	28	281.922903	...	1824.0	73	GZ	
4	10	FT.2.0	EDU2	01004	csp_8_4	AT2|2	Male	True	16	602.748256	...	NaN	NaN	NaN	

5 rows × 22 columns

5 Sélection des algorithmes d'apprentissage supervisé

5.1 Modèle Linéaire

```
model_linear = LinearRegression()
```

5.2 Random Forest

```
model_rf = RandomForestRegressor()
```

5.3 Plus proches voisins

```
model_knn = KNeighborsRegressor()
```

5.4 SVM

```
model_svm = SVR()
```

5.5 Entrainons les modèles

```
predictions_linear = model_linear.predict(x_test)
```

```
predictions_linear
```

```
array([375.71076586, 453.9763946 , 326.90908233, ..., 591.33693411,
       623.59337937, 218.0634448 ])
```

```
predictions_rf = model_rf.predict(x_test)
```

```
predictions_rf
```

```
array([468.81280481, 533.92955234, 440.39929562, ..., 710.07137928,
       754.8267908 , 385.64144606])
```

```
predictions_knn = model_knn.predict(x_test)
```

```
predictions_knn
```

```
array([399.47604719, 389.04754746, 408.10262965, ..., 672.24720647,
       633.08304384, 461.03692416])
```

```
predictions_svm = model_svm.predict(x_test)
```

```
predictions_svm
```

```
array([447.27277197, 420.56190798, 431.83775253, ..., 446.52237203,
       436.83789582, 409.36406393])
```

5.6 Évaluations des performances des modèles

```
mse_linear = mean_squared_error(y_test, predictions_linear)
mse_rf = mean_squared_error(y_test, predictions_rf)
mse_knn = mean_squared_error(y_test, predictions_knn)
mse_svm = mean_squared_error(y_test, predictions_svm)
```

5.7 Coefficient de détermination des modèles

```
from sklearn.metrics import r2_score
print("le R2 pour le model_linear est : ", r2_score(y_test, predictions_linear))
print("le R2 pour le random forest est : ", r2_score(y_test, predictions_rf))
print("le R2 pour le Knn model est : ", r2_score(y_test, predictions_knn))
print("le R2 pour le Svm model est : ", r2_score(y_test, predictions_svm))
```

```
le R2 pour le model_linear est :  0.47006075466044817
le R2 pour le random forest est :  0.8094992482276947
le R2 pour le Knn model est :  0.4034202300066212
le R2 pour le Svm model est :  0.032121128200017446
```

Sur l'ensemble de test

Le Random forest apparait avec le le meilleur coefficient de détermination, il explique environ 80% de la variance dans les données de test. Un R2 de 0.032 n'est pas très élevé, ce qui indique que le modèle Svm ne parvient pas à capturer une grande partie de la

variance des données de test. Cela peut être dû à une sous-ajustement (underfitting) du modèle, où le modèle est trop simple pour capturer la complexité des données.

Sur l'ensemble d'entraînement

```
print("le R2 pour le model_linear est : ",r2_score(y_train, model_linear.predict(x_tr
print("le R2 pour le random forest est : ",r2_score(y_train, model_rf.predict(x_train
print("le R2 pour le Knn linear est : ",r2_score(y_train, model_knn.predict(x_train))
print("le R2 pour le Svm linear est : ",r2_score(y_train, model_svm.predict(x_train))
```

```
le R2 pour le model_linear est :  0.46310029102899775
le R2 pour le random forest est :  0.9736370129389976
le R2 pour le Knn linear est :  0.6158522316544622
le R2 pour le Svm linear est :  0.0325167284977641
```

Le même modèle explique environ 97,3% de la variance dans les données d'entraînement. Un R2 de 0.97 pour l'ensemble d'entraînement est plus élevé que celui pour l'ensemble de test, ce qui est souvent le cas. Cela peut indiquer un certain surajustement (overfitting) du modèle aux données d'entraînement.

Le modèle peut être trop complexe et avoir appris le bruit présent dans les données d'entraînement. Même remarque avec le Knn model ou nous passons de 37% à 60%. Les deux autres apparaissent raisonnables.

5.8 Comparaison ses performances des différents modèles choisis

```
performances = {'Linear Regression': mse_linear, 'Random Forest': mse_rf, 'K-Nearest
print(performances)
```

```
{'Linear Regression': 11058.921598765703, 'Random Forest': 3975.423403499752, 'K-Near
```

Interpretation

Nos résultats montrent les performances des différents modèles en termes d'erreur quadratique moyenne (MSE) sur un ensemble de données de test. Ci-après une interpretation des résultats :

- **Linear Regression** : L'erreur quadratique moyenne obtenue avec la régression linéaire est d'environ 11050.77.

- **Random Forest** : L'erreur quadratique moyenne obtenue avec le modèle de forêt aléatoire est d'environ 3992.39.

- **K-Nearest Neighbors** : L'erreur quadratique moyenne obtenue avec le modèle de k-plus proches voisins est d'environ 13100.18.

- **Support Vector Machine** : L'erreur quadratique moyenne obtenue avec le modèle de machine à vecteurs de support est d'environ 20197.93.

En général, des valeurs de MSE plus faibles indiquent de meilleures performances du modèle, car cela signifie que les prédictions du modèle sont plus proches des vraies valeurs. Ainsi, dans ce cas, le modèle Random Forest semble être le plus performant, suivi par la régression linéaire, les k-plus proches voisins et enfin la machine à vecteurs de support.

Cependant les résultas du random Forest nous semble malgré tout assez élevé, nous souhaitons donc d'utiliser d'autres approches plus robuste afin d'améliorer les résultats de prédiction.

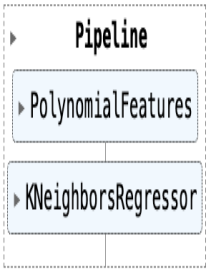
6 Approche complexe

6.1 Approche polynomiale

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Créer un pipeline avec PolynomialFeatures pour ajouter des interactions et des term
model_poly = make_pipeline(PolynomialFeatures(degree=2), KNeighborsRegressor())
```

6.2 Entrainement du modèle

```
model_poly.fit(x_train, y_train)
```



6.3 Prédiction

6.4 Entrainement du modèle

```
prediction_poly = model_poly.predict(x_test)
```

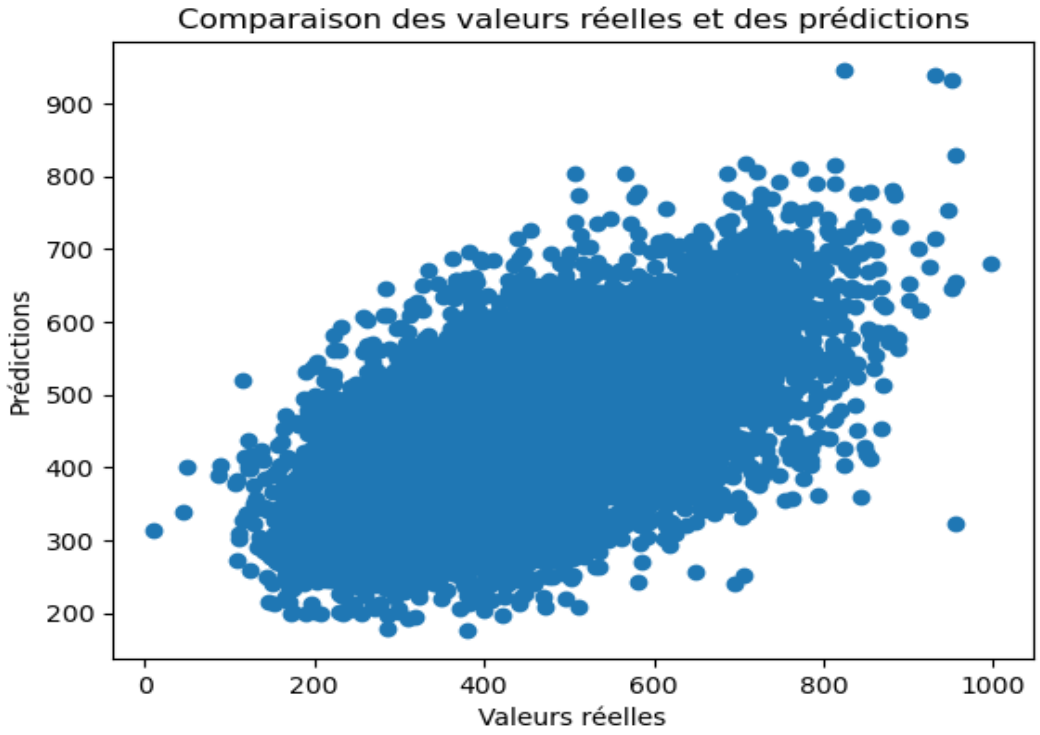
6.5 Évaluation du modèle

```
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, prediction_poly)
r2 = r2_score(y_test, prediction_poly)

print("MSE :", mse)
print("R2 Score :", r2)
```

```
MSE : 13299.646325422014
R2 Score : 0.362686092488103
```

Ces résultats sont plus élevés que ceux obtenus avec le random forest plus haut.

```
# Visualisation des prédictions par rapport aux valeurs réelles
plt.scatter(y_test, prediction_poly)
plt.xlabel("Valeurs réelles")
plt.ylabel("Prédictions")
plt.title("Comparaison des valeurs réelles et des prédictions")
plt.show()
```



Interpretation :

On remarque une énorme dispersion entre les valeurs réelles et les valeurs prédictes

6.6 Utilisons des techniques de régularisation : un modèle Ridge

```
model_ridge.fit(x_train, y_train)
prediction_ridge = model_ridge.predict(x_test)
mse = mean_squared_error(y_test, prediction_ridge)
r2 = r2_score(y_test, prediction_ridge)
mse_ridge : 11058.918878376611
R2_ridge : 0.4700608850204412
```

```
mse_ridge : 11058.918878376611
R2_ridge : 0.4700608850204412
```

Interpretation des résultats obtenu avec le modèle ridge :

Notre **MSE** de **11050.79** indique que les prédictions du modèle, en moyenne, s'écartent de la vraie valeur cible d'environ 11050.79 unités

Notre **R2** de **0.47** signifie que le modèle explique environ 47% de la variance totale des données de test.

Notre modèle Ridge semble avoir des performances raisonnables, mais il y a encore de la place pour l'amélioration, surtout si nous cherchons à réduire l'erreur de prédiction et

à augmenter la proportion de variance expliquée par le modèle.

6.7 Analyse des coefficients du modèle

```
# Obtenir les coefficients de régression
coefficients = best_ridge_model.coef_

# Afficher les coefficients
for feature, coef in zip(x_train.columns, coefficients):
    print(feature, ":", coef)
```

```
Uid : -0.0005266174024338466
Familty_type : -3.0327784065246988
Highest_diploma : -5.27219410451327
Job_42 : 0.08136971330461966
ACT : 34.22575605817072
sex : 28.297580835199543
Studying : -7.509424265152476
age_2020 : -2.840972022446897
WORK_CONDITION : 8.328986296819764
TERMS_OF_EMP : 25.417215339010944
working_hours : -0.00027460207902695543
Eco_sect : 0.2424178138519283
Employer_category : 2.8143516125297854
job_category : 25.83887528979055
Job_desc : 0.4327389324698308
EMOLUMENT : 0.0011687093354819976
employee_count : 4.811255653533696
emp : -3.58352037864475
CLUB : -0.041182024170218906
```

Interpretation :

Pour analyser les coefficients du modèle, Nous pouvons considérer les points suivants :

1. **Importance des caractéristiques** :

Les coefficients indiquent l'importance relative de chaque caractéristique dans la prédiction de la variable cible. Les caractéristiques avec des coefficients plus importants ont un impact plus important sur les prédictions. Dans notre cas : ACT, job_category, sex, TERMS_OF_EMP et Studying.

2. **Interprétation de signe** :

Le signe du coefficient indique la direction de la relation entre la caractéristique et la variable cible. Un coefficient positif signifie qu'une augmentation de la valeur de la caractéristique est associée à une augmentation de la variable cible, (ici **ACT** , **sex** , : **les types d'activités**, **le sexe des individus**) tandis qu'un coefficient négatif signifie l'opposé (ici job_category, Studying : les variables encodées des cathégories de jobs et la variable qui renseigne si l'individu est étudiant ou pas) .

3. **Magnitude des coefficients** :

Plus la valeur absolue d'un coefficient est grande, plus l'impact de la caractéristique correspondante sur la variable cible est important. Les coefficients avec une magnitude élevée peuvent indiquer des relations fortes entre les caractéristiques et la variable cible.

4. **Comparaison avec d'autres coefficients** :

Comparons les coefficients entre eux pour évaluer leur importance relative. Les caractéristiques avec des coefficients plus importants peuvent être considérées comme ayant un plus grand pouvoir prédictif.

En analysant nos coefficients nous pouvons conclure :

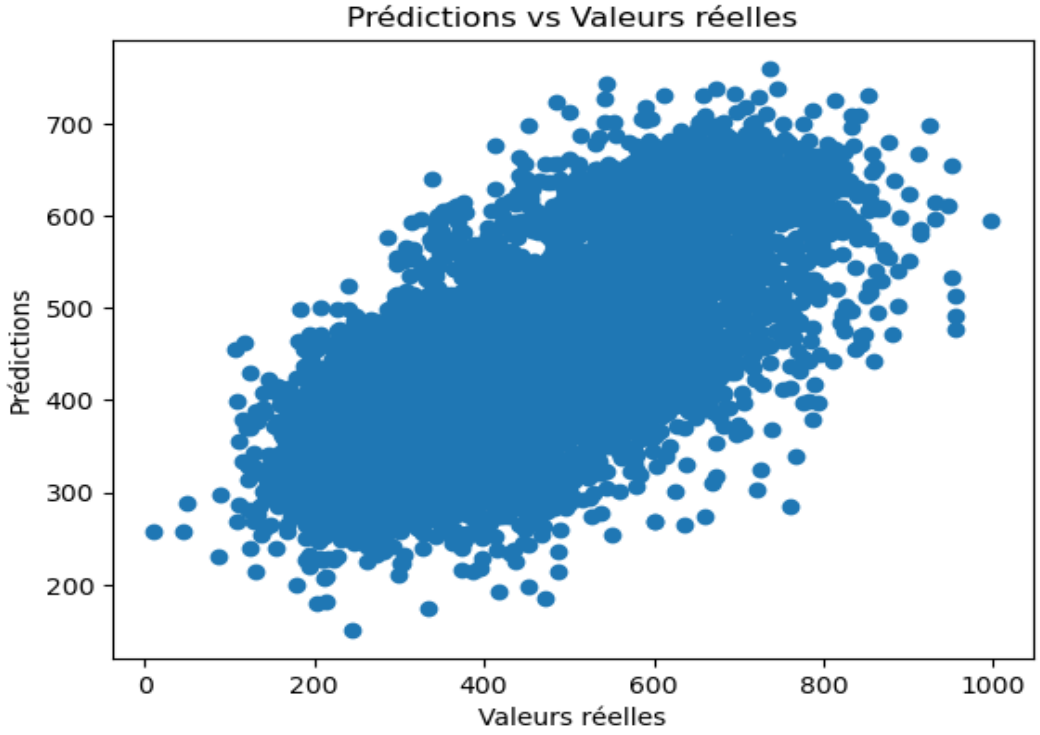
— Les caractéristiques avec des coefficients négatifs (comme 'Uid', 'Studying', 'age_2020', etc.) ont une relation négative avec la variable cible. Cela signifie que lorsque ces caractéristiques augmentent, la variable cible a tendance à diminuer, et vice versa.

— À l'inverse, les caractéristiques avec des coefficients positifs (comme 'ACT', 'sex', 'TERMS_OF_EMP', etc.) ont une relation positive avec la variable cible. Cela signifie qu'une augmentation de ces caractéristiques est associée à une augmentation de la variable cible, et vice versa.

— Les caractéristiques avec des coefficients proches de zéro ont moins d'impact sur la variable cible.

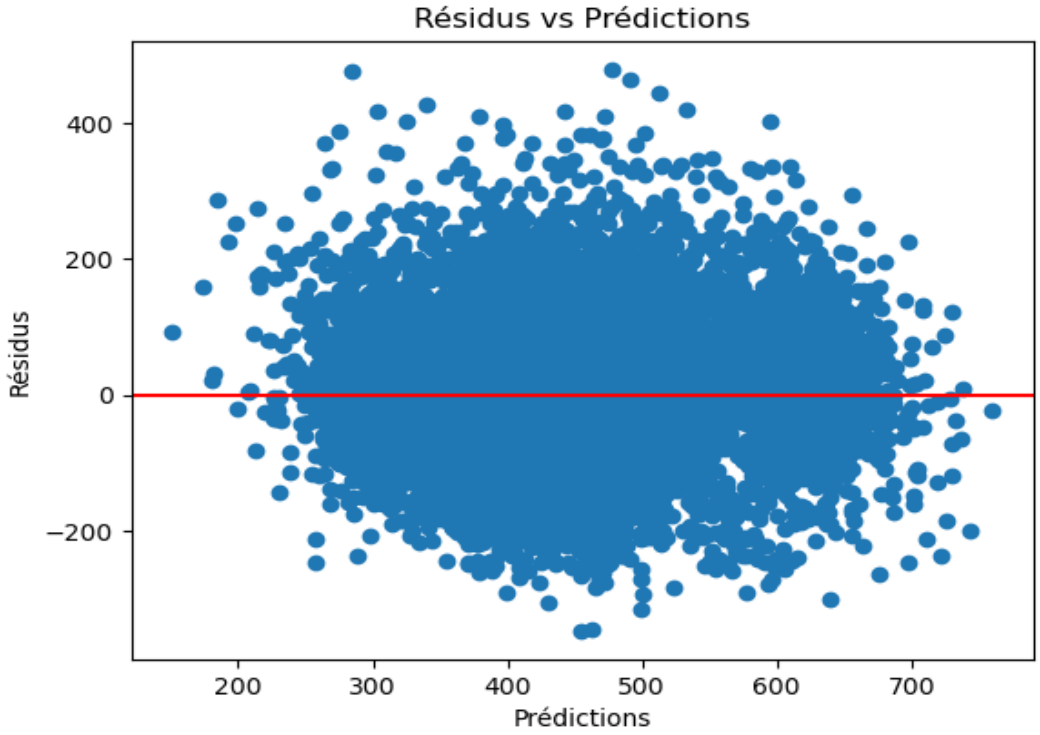
— La magnitude des coefficients peut être utilisée pour comparer l'importance relative des caractéristiques. Par exemple, 'ACT' a un coefficient relativement élevé, ce qui suggère qu'il a un impact plus important sur la variable cible par rapport aux autres caractéristiques.

Visualisations les prédictions du modèle



Nous observons une dispersion entre les valeurs réels de notre ensemmble d'entrainement et celles qui ont été prédicte via le modèle de rigde

Analyse des résidus



6.8 Utiliser les meilleures valeurs d'hyperparamètres : KNeighborsRegressor

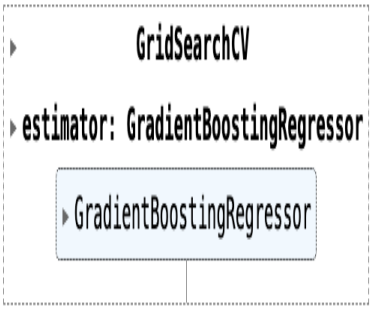
```
from sklearn.model_selection import GridSearchCV

# Utiliser GridSearchCV pour effectuer une recherche sur une grille de paramètres
param_grid = {'n_neighbors': [5, 10, 15]}  # Spécifiez les hyperparamètres à recherch
grid_search = GridSearchCV(KNeighborsRegressor(), param_grid, cv=5)
grid_search.fit(x_train, y_train)
best_model = grid_search.best_estimator_

best_model
```

```
MSE: 12449.594857876906
R2: 0.4034202300066212
```

6.9 Gradient Boosting

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor

# Définition du modèle
model = GradientBoostingRegressor()

# Définition de la grille des hyperparamètres à tester
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.5],
    'max_depth': [3, 5, 7]
}
```

Analyse des résidus




```
# Utilisez les meilleurs paramètres trouvés pour GradientBoostingRegressor
best_params_gb = {'n_estimators': 200, 'learning_rate': 0.5, 'max_depth': 5}

# Initialisez le modèle GradientBoostingRegressor avec les meilleurs paramètres
final_model = GradientBoostingRegressor(**best_params_gb)


# Entraînez le modèle final
final_model.fit(x_train, y_train)
```

Interprétation des résultats :

Learning_rate : 0.5 :
indique la vitesse à laquelle le modèle d'apprentissage s'ajuste aux erreurs. Une valeur plus élevée signifie un ajustement plus rapide, mais cela peut rendre le modèle instable.

Max_depth : 5 :
indique la profondeur maximale de chaque arbre de décision dans l'algorithme de boosting. Une valeur plus élevée permet au modèle de capturer des relations plus complexes dans les données, mais cela peut entraîner un surapprentissage.

N_estimators : 200 :
C'est le nombre d'estimateurs (ou arbres) dans l'algorithme de boosting. Une valeur plus élevée peut améliorer les performances du modèle, mais cela nécessite également plus de temps de calcul. Ces valeurs sont celles qui ont donné la meilleure performance sur l'ensemble d'entraînement pendant la recherche des hyperparamètres.

Nous utiliserons ces valeurs pour entraîner notre modèle final afin d'obtenir de meilleures performances sur de nouveaux ensembles de données.

```
MSE: 1547.4058916382537
R2: 0.9258489082208272
```

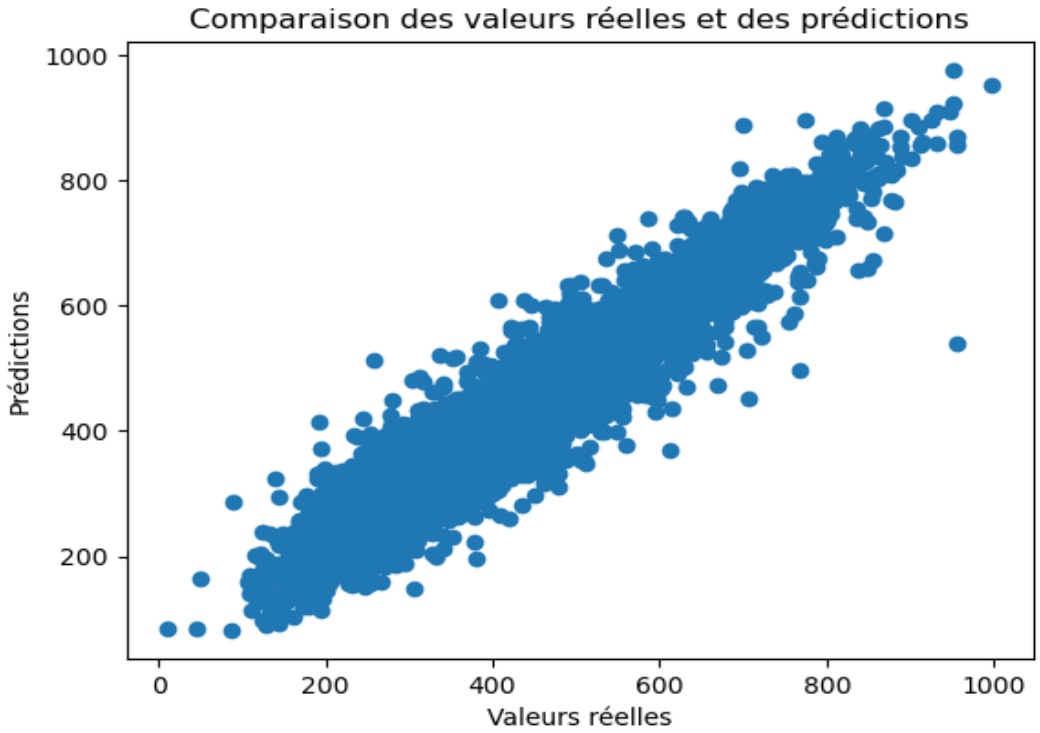
Interpretation des résultats

Notre nouveau **MSE** est de **1547.40** (est assez basse) , ce qui suggère que le modèle a une bonne capacité à prédire les valeurs cibles sur l'ensemble de test.

Notre coefficient de détermination **R$\hat{2}$** est de **92** % ce qui indique que le modèle a une bonne capacité à capturer la relation entre les variables d'entrée et la variable cible.

En somme, les résultats indiquent que le modèle avec les meilleurs hyperparamètres identifiés a une très bonne performance de prédiction sur l'ensemble de test, avec une faible erreur moyenne et une grande capacité à expliquer la variance des données. Cela suggère que le modèle est bien généralisé et peut être utilisé avec confiance pour faire des prédictions sur de nouvelles données.

Analyse des résidus

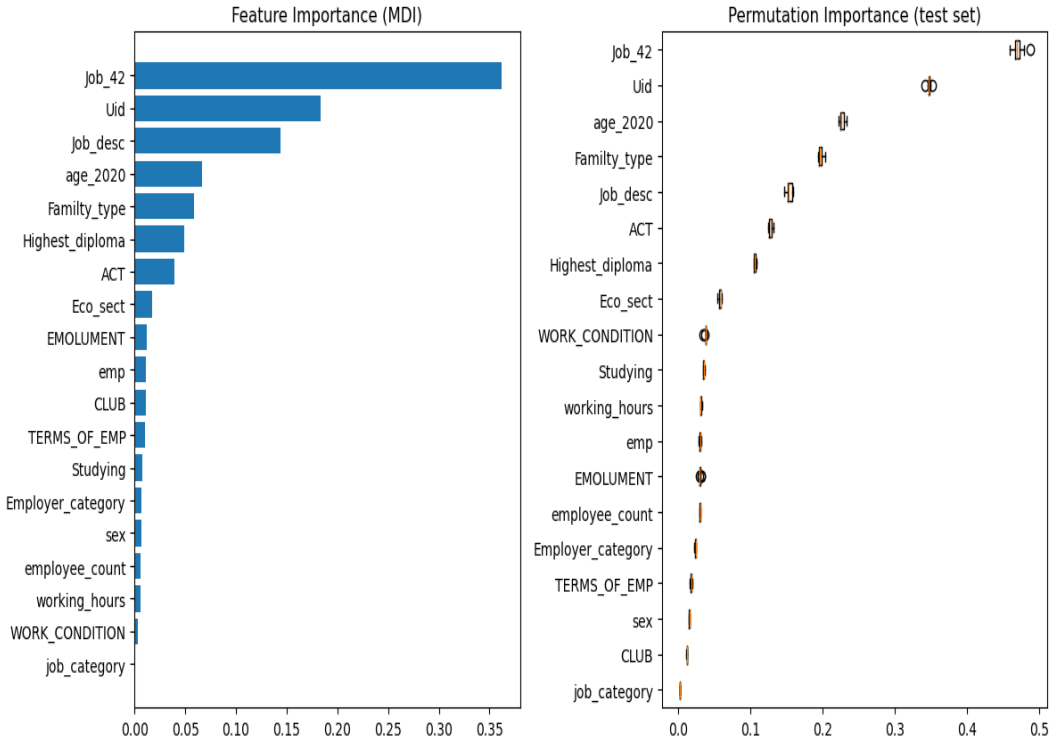


Interpretation :

On remarque une dispersion plus fine entre les valeurs réelles et les valeurs prédictes. Les valeurs prédictes se rapprochent des prédictions. En comparaison avec le modèle de

Ridge.

Analyse des résidus



Interpretation :

Les barres les plus longues (caractéristiques de l'emploi, leur age, le type de famille) correspondent aux variables les plus importantes pour la prédiction du modèle.

ces dernières assez éloignés de l'origine sont considérées comme plus importantes car leur permutation aléatoire entraîne une plus grande baisse de performance du modèle. Ces points indiquent la sensibilité du modèle aux changements dans ces caractéristiques.

7 Prédiction avec le modèle final sur l'ensemble de Test

```
Y_pred_final = final_model.predict(X_test)
```

7.1 Exportons les prédictions dans un fichier Csv

```
predictions_merge = pd.DataFrame({'Uid' : Uid_test,'Linear Regression': predictions_l
predictions_merge.to_csv('predictions_merge.csv', index=False)
```

```
Prediction = pd.DataFrame({"Uid" : Uid, "Target": Y_pred_final})
Prediction.to_csv("Prediction.csv", index=False)
```

Conclusion

Au terme de nos différentes analyses, nous pouvons dire au vu de tout ce qui a été réalisé que nous somme satisfait du travail fourni car nous avons atteint nos différents objectifs, analyser visuellement des données de 99985 Français (de France métropolitaine) et appliquer des méthodes de machine learning sur nos échantillons.

En effet, ce projet nous a permis d'utiliser et d'appliquer plusieurs tests sur python, cet interpreteur informatique très adapté à l'analyse de données.

Pour finir, nous espérons que notre modèle soit autant robuste face à de nouvelles données et qu'il puisse capter les différentes variances pouvant exister.

REFERENCES

Cours Machine Learning, Fabrice Rossi , 2024,

Introduction au Machine Learning, Chloé-Agathe Azencott, 228, 2019,
https ://https ://scikit-learn.org//