```
from IPython.core.display import display, HTML

#Icone kaggle
display(HTML("""
<a href="https://www.kaggle.com/competitions/titanic/overview" target="_blank">
    <img align="left" alt="Kaggle" title="Open in Kaggle"
         src="https://kaggle.com/static/images/open-in-kaggle.svg" style="width:200px;">
</a>
"""))
```

k Open in Kaggle

```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Introduction
</h1>
<p style="font-family: Georgia; font-size: 14pt; font-weight: bold; color:#4cd1ff;">
Ce projet explore le célèbre défi Titanic de Kaggle, avec pour objectif de prédire la survie des passagers en fonction de leurs caractéri
<br><br>
À l'aide de modèles tels que la Régression Logistique, RandomForest et XGBoost,
combinés à des techniques comme les valeurs SHAP et l’importance par permutation,
nous visons à concilier précision et compréhension des prédictions.
<br><br>
Nous travaillerons en équipe de 4 membres pour garantir une collaboration efficace et atteindre les objectifs fixés. L'outil collaboratif
avec une stratégie de branches et des pull requests pour assurer la qualité et la revue du code. Un pipeline CI/CD sera mis en place pour
la containerisation du projet.</p>
🚢✨
"""))
```

Introduction

Ce projet explore le célèbre défi Titanic de Kaggle, avec pour objectif de prédire la survie des passagers en fonction de leurs caractéristiques.

À l'aide de modèles tels que la Régression Logistique, RandomForest et XGBoost, combinés à des techniques comme les valeurs SHAP et l’importance par permutation, nous visons à concilier précision et compréhension des prédictions.

Nous travaillerons en équipe de 4 membres pour garantir une collaboration efficace et atteindre les objectifs fixés. L'outil collaboratif GitHub sera utilisé pour gérer le code source, avec une stratégie de

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
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, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
import joblib
from sklearn.ensemble import StackingClassifier
from sklearn.inspection import permutation_importance
import shap

# Définir les styles visuels
plt.style.use('ggplot')
sns.set_context('notebook')
shap.initjs()
```

```
HO_TUNING = True #on n’active pas la recherche d’hyperparamètres de Grid Search et Random Search
LOAD_MODEL = True #on charge le modèle déjà sauvegardé depuis le dossier models_path, plutôt que d’en entraîner un nouveau.
```

```
# Chemin du dossier models dans le répertoire local
models_path = r"C:\Users\mbena\OneDrive\Bureau\python TP\BUT3FA\Titanic-Survival-Prediction-main\Titanic-Survival-Predict-main\models"

# On vérifie si le dossier models existe, sinon le crée
if not os.path.exists(models_path):
    os.mkdir(models_path)



display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 1: Data Loading et Exploration des Données
</h1>
"""))
```

Partie 1: Data Loading et Exploration des Données

```
# Configuration des styles pour les graphiques
plt.style.use('ggplot')
sns.set_context('notebook')

# Parcours des fichiers dans un répertoire spécifié
base_path = r"C:\Users\mbena\OneDrive\Bureau\python TP\BUT3FA\Titanic-Survival-Prediction-main\Titanic-Survival-Predict-main"
for dirname, _, filenames in os.walk(base_path):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Code minimal pour garantir le fonctionnement
print("\n--- Tâche 0 : Exploration du répertoire terminée ---")
```

```
--- Tâche 0 : Exploration du répertoire terminée ---
```

```
import os

def get_models_path():
    """
    Retourne le chemin pour enregistrer ou charger les modèles.
    - Si l'environnement est Kaggle, retourne le chemin Kaggle.
    - Sinon, retourne un chemin local basé sur le répertoire du projet.
    Crée le répertoire si nécessaire.
    """
    if os.path.exists("/kaggle/working/"):
        # Chemin pour l'environnement Kaggle
        models_path = "/kaggle/working/models/"
    else:
        # Chemin pour l'environnement local (répertoire du projet)
        project_path = r"C:\Users\mbena\OneDrive\Bureau\python TP\BUT3FA\Titanic-Survival-Prediction-main\Titanic-Survival-Predict-main'
        models_path = os.path.join(project_path, "models")

    # Création du répertoire s'il n'existe pas
    os.makedirs(models_path, exist_ok=True)
    return models_path

# Exemple d'utilisation
if __name__ == "__main__":
    models_path = get_models_path()
    print(f"Le chemin du dossier 'models' est : {models_path}")
```

```
Le chemin du dossier 'models' est : C:\Users\mbena\OneDrive\Bureau\python TP\BUT3FA\Titanic-Survival-Prediction-main\Titanic-Surviva
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Étape 2 : Chargement du jeu de données
# Charger les fichiers CSV contenant les données d'entraînement et de test
train_df = pd.read_csv('/content/drive/My Drive/Titanic-Survival-Predict-main/train_cleaned.csv')
test_df = pd.read_csv('/content/drive/My Drive/Titanic-Survival-Predict-main/test_cleaned.csv')

# On affiche les informations de base sur le jeu de données d'entraînement
print("Training Set Information:")
train_df.info()
```

```
# On affiche les informations de base sur le jeu de données de test
print("\nTest Set Information:")
test_df.info()
```

```
Training Set Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB

Test Set Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
```

```
# Étape 3 : Statistiques de base et exploration initiale des données
# Affichage des 5 premières lignes du jeu de données d'entraînement pour aperçu
print("\nLes 5 premières lignes du jeu d'entraînement :")
print(train_df.head())

# On vérifie les valeurs manquantes dans le jeu de données d'entraînement et de test
print("\nValeurs manquantes dans le jeu d'entraînement :")
print(train_df.isnull().sum())

print("\nValeurs manquantes dans le jeu de test :")
print(test_df.isnull().sum())
```

```
Les 5 premières lignes du jeu d'entraînement :
   PassengerId  Survived  Pclass  \
0            1         0       3   
1            2         1       1   
2            3         1       3   
3            4         1       1   
4            5         0       3   

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1   
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1   
2                             Heikkinen, Miss. Laina  female  26.0      0   
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1   
4                           Allen, Mr. William Henry    male  35.0      0   

   Parch            Ticket     Fare Cabin Embarked  
0      0         A/5 21171   7.2500   NaN        S  
1      0          PC 17599  71.2833   C85        C  
2      0  STON/O2. 3101282   7.9250   NaN        S  
3      0            113803  53.1000  C123        S  
4      0            373450   8.0500   NaN        S  

Valeurs manquantes dans le jeu d'entraînement :
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
```

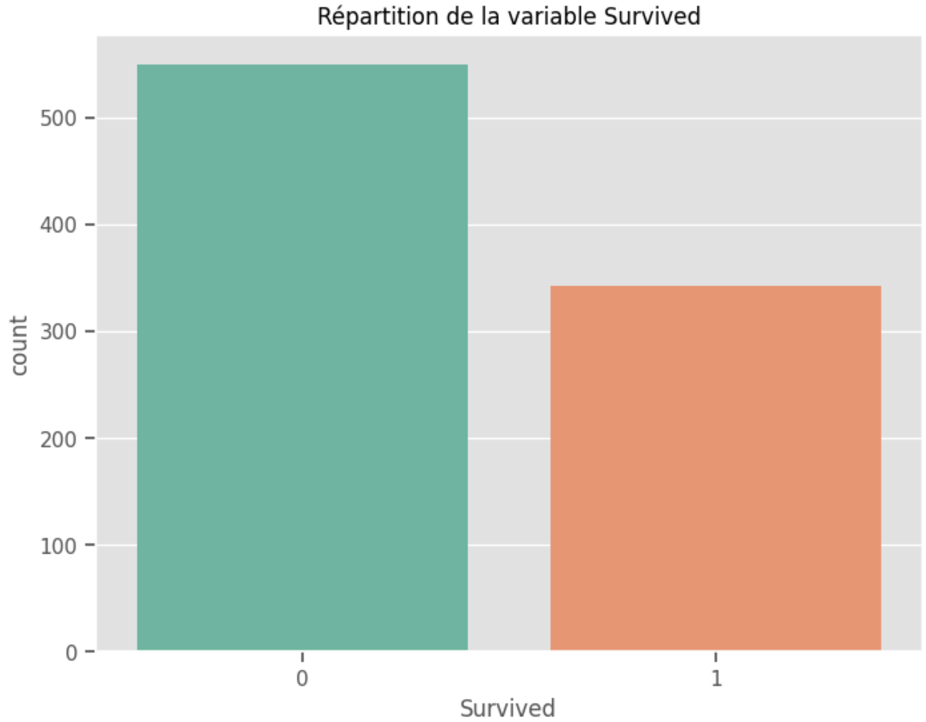
```
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64

Valeurs manquantes dans le jeu de test :
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 2: Analyse Exploratoire des Données (EDA)
</h1>
"""))
```

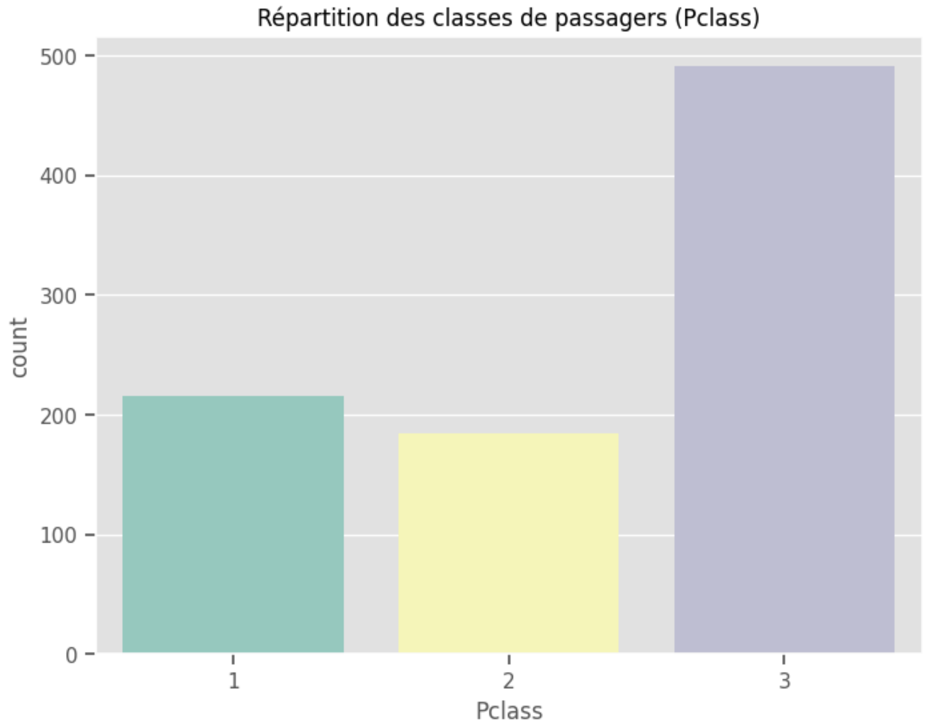
Partie 2: Analyse Exploratoire des Données (EDA)

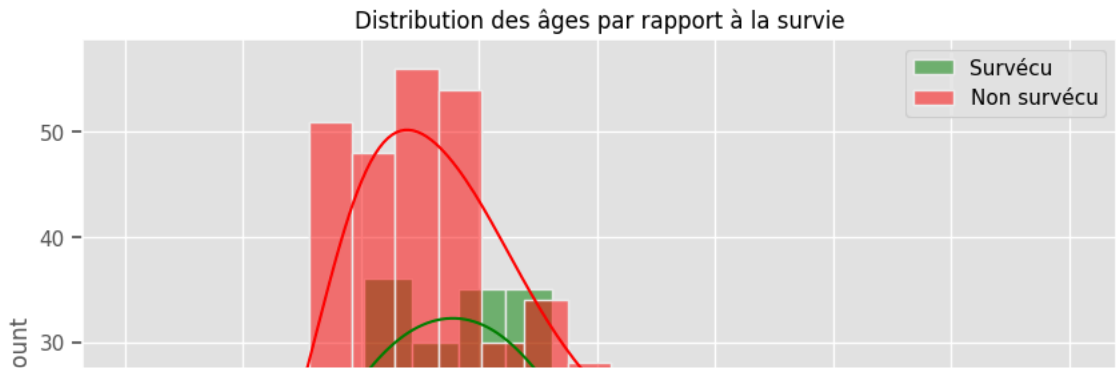
```
# Import des bibliothèques nécessaires
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Étape 4 : Analyse exploratoire des données (EDA)

# Visualisation de la répartition de la variable 'Survived' dans l'ensemble d'entraînement
plt.figure(figsize=(8, 6))
sns.countplot(data=train_df, x='Survived', palette='Set2')
plt.title('Répartition de la variable Survived')
plt.show()

# Visualisation de la distribution des classes de passagers (Pclass)
plt.figure(figsize=(8, 6))
sns.countplot(data=train_df, x='Pclass', palette='Set3')
plt.title('Répartition des classes de passagers (Pclass)')
plt.show()

# Distribution de l'âge par rapport à la survie
plt.figure(figsize=(10, 6))
sns.histplot(train_df[train_df['Survived'] == 1]['Age'].dropna(), bins=20, color='green', label='Survécu', kde=True)
sns.histplot(train_df[train_df['Survived'] == 0]['Age'].dropna(), bins=20, color='red', label='Non survécu', kde=True)
plt.title('Distribution des âges par rapport à la survie')
plt.legend()
plt.show()

# Matrice de corrélation pour vérifier les relations entre les variables numériques
# On filtre uniquement les colonnes numériques
numeric_features = train_df.select_dtypes(include=[np.number])

plt.figure(figsize=(12, 8))
sns.heatmap(numeric_features.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Matrice de corrélation')
plt.show()
```

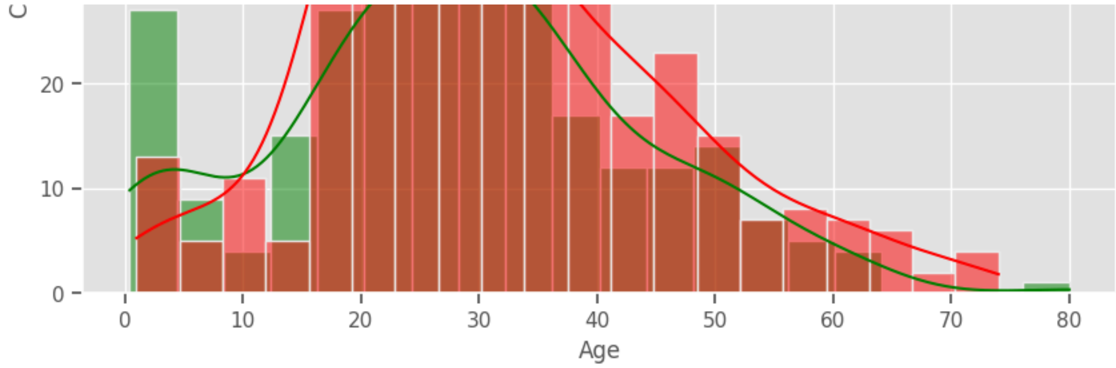
```
<ipython-input-13-8de1cf76e074>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=train_df, x='Survived', palette='Set2')
```

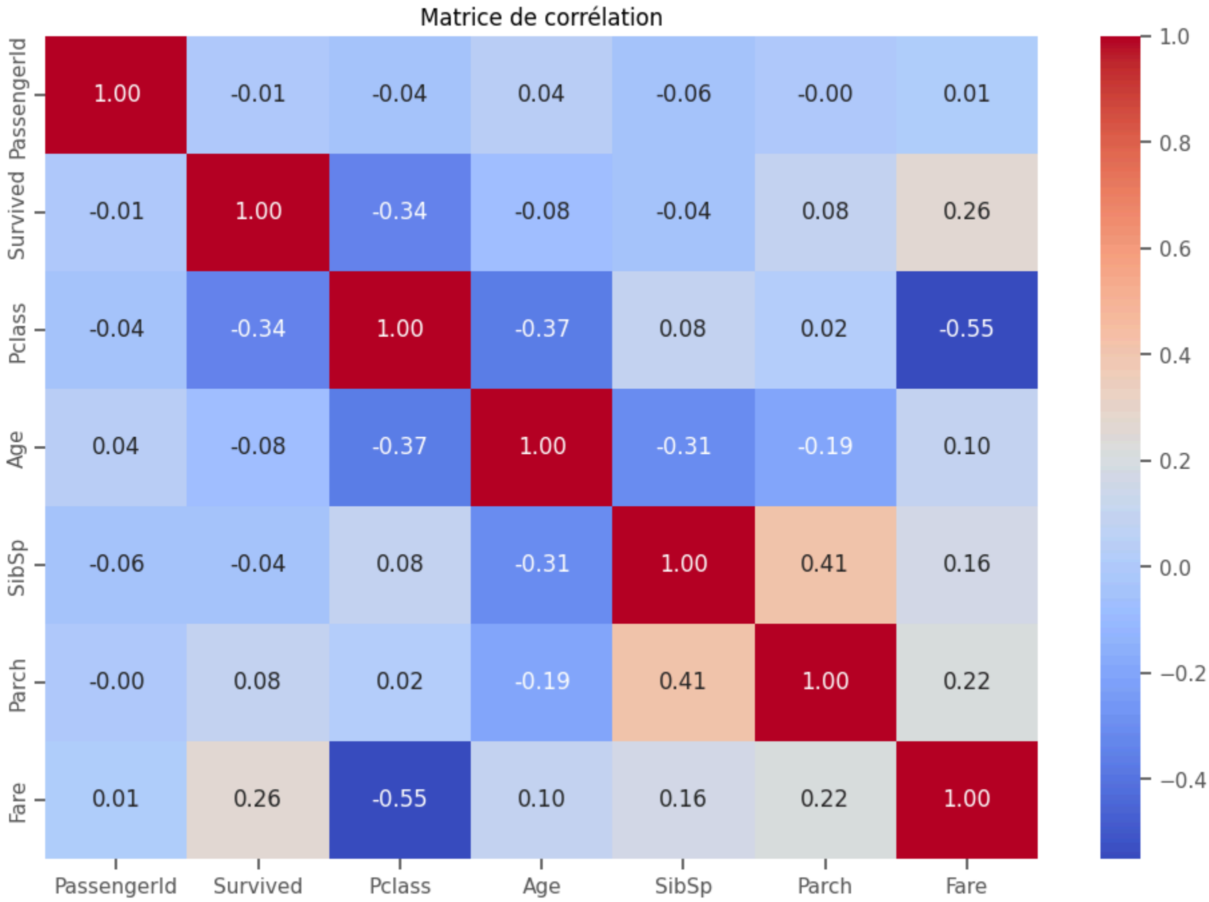



```
<ipython-input-13-8de1cf76e074>:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=train_df, x='Pclass', palette='Set3')
```






20
10
0
0
10
20
30
40
50
60
70
80
Age


Matrice de corrélation
PassengerId
Survived
Pclass
Age
SibSp
Parch
Fare
1.00 -0.01 -0.04 0.04 -0.06 -0.00 0.01
-0.01 1.00 -0.34 -0.08 -0.04 0.08 0.26
-0.04 -0.34 1.00 -0.37 0.08 0.02 -0.55
0.04 -0.08 -0.37 1.00 -0.31 -0.19 0.10
-0.06 -0.04 0.08 -0.31 1.00 0.41 0.16
-0.00 0.08 0.02 -0.19 0.41 1.00 0.22
0.01 0.26 -0.55 0.10 0.16 0.22 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 3: Data Preprocessing
</h1>
"""))
```

Partie 3: Data Preprocessing

```
train_df = pd.read_csv('/content/drive/My Drive/Titanic-Survival-Predict-main/train_cleaned.csv')
test_df = pd.read_csv('/content/drive/My Drive/Titanic-Survival-Predict-main/test_cleaned.csv')


# Étape 1 : Gestion des valeurs manquantes

# On Remplit les valeurs manquantes d'Age avec la médiane par groupe de Pclass
train_df['Age'] = train_df.groupby('Pclass')['Age'].transform(lambda x: x.fillna(x.median()))
test_df['Age'] = test_df.groupby('Pclass')['Age'].transform(lambda x: x.fillna(x.median()))

# Ajout d'une colonne AgeGroup pour catégoriser l'âge en groupes
for df in [train_df, test_df]:
    df['AgeGroup'] = pd.cut(df['Age'], bins=[0, 12, 18, 35, 60, 80], labels=[0, 1, 2, 3, 4])

# Remplis des valeurs manquantes d'Embarked avec la valeur la plus fréquente
train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)

# Remplis les valeurs manquantes de Fare dans test_df avec la médiane par groupe de Pclass
test_df['Fare'] = test_df.groupby('Pclass')['Fare'].transform(lambda x: x.fillna(x.median()))

# Ajout d'une colonne FareBin pour diviser Fare en 4 intervalles égaux
for df in [train_df, test_df]:
    df['FareBin'] = pd.qcut(df['Fare'], 4, labels=[0, 1, 2, 3])

# Étape 2 : Création de nouvelles caractéristiques

# Ajout d'une colonne Age*Pclass comme interaction entre AgeGroup et Pclass
for df in [train_df, test_df]:
    df['Age*Pclass'] = df['AgeGroup'].astype(int) * df['Pclass']

# Ajout d'une colonne FamilySize combinant SibSp et Parch
train_df['FamilySize'] = train_df['SibSp'] + train_df['Parch'] + 1
test_df['FamilySize'] = test_df['SibSp'] + test_df['Parch'] + 1

# Ajout de la  colonne IsAlone : 1 si le passager est seul, sinon 0
train_df['IsAlone'] = 1
train_df['IsAlone'].loc[train_df['FamilySize'] > 1] = 0
test_df['IsAlone'] = 1
test_df['IsAlone'].loc[test_df['FamilySize'] > 1] = 0

# Extraire Title à partir de la colonne Name
train_df['Title'] = train_df['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
test_df['Title'] = test_df['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)

# Remplacer les titres rares par Rare
rare_titles = ['Lady', 'Countess', 'Capt', 'Col', 'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona']
train_df['Title'] = train_df['Title'].replace(rare_titles, 'Rare')
test_df['Title'] = test_df['Title'].replace(rare_titles, 'Rare')

# Standardiser les titres courants (ex. Mlle devient Miss, Mme devient Mrs)
train_df['Title'] = train_df['Title'].replace(['Mlle', 'Ms'], 'Miss')
train_df['Title'] = train_df['Title'].replace('Mme', 'Mrs')
test_df['Title'] = test_df['Title'].replace(['Mlle', 'Ms'], 'Miss')
test_df['Title'] = test_df['Title'].replace('Mme', 'Mrs')

# Étape 3 : Encodage des variables catégoriques

# Calculer FamilySurvival comme le taux de survie moyen par taille de famille
family_survival_rate = train_df.groupby('FamilySize')['Survived'].mean().to_dict()

# Appliquer FamilySurvival aux deux DataFrames
for df in [train_df, test_df]:
    df['FamilySurvival'] = df['FamilySize'].map(family_survival_rate)
    df['FamilySurvival'] = df['FamilySurvival'].fillna(0)

# Mapper Sex en valeurs numériques : 0 pour homme, 1 pour femme
train_df['Sex'] = train_df['Sex'].map({'male': 0, 'female': 1})
test_df['Sex'] = test_df['Sex'].map({'male': 0, 'female': 1})
```

```
# Mapper Embarked en valeurs numériques : S=0, C=1, Q=2
train_df['Embarked'] = train_df['Embarked'].map({'S': 0, 'C': 1, 'Q': 2})
test_df['Embarked'] = test_df['Embarked'].map({'S': 0, 'C': 1, 'Q': 2})

# Mapper Title en valeurs numériques
title_mapping = {'Mr': 0, 'Miss': 1, 'Mrs': 2, 'Master': 3, 'Rare': 4}
train_df['Title'] = train_df['Title'].map(title_mapping)
test_df['Title'] = test_df['Title'].map(title_mapping)

# Extraire Deck à partir de Cabin et remplacer les valeurs manquantes par 'M'
for df in [train_df, test_df]:
    df['Deck'] = df['Cabin'].str[0]
    df['Deck'] = df['Deck'].fillna('M')

# Mapper Deck en catégories numériques
deck_mapping = {deck: idx for idx, deck in enumerate(sorted(train_df['Deck'].unique()))}
train_df['Deck'] = train_df['Deck'].map(deck_mapping)
test_df['Deck'] = test_df['Deck'].map(deck_mapping)

# Supprimer les colonnes inutiles pour la modélisation
train_df = train_df.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'])
test_df = test_df.drop(columns=['Name', 'Ticket', 'Cabin'])
```

```
<ipython-input-16-3fd4ef9061bf>:12: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[co


  train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)
<ipython-input-16-3fd4ef9061bf>:33: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will b
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the or

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

  train_df['IsAlone'].loc[train_df['FamilySize'] > 1] = 0
<ipython-input-16-3fd4ef9061bf>:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  train_df['IsAlone'].loc[train_df['FamilySize'] > 1] = 0
<ipython-input-16-3fd4ef9061bf>:35: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will b
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the or

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

  test_df['IsAlone'].loc[test_df['FamilySize'] > 1] = 0
<ipython-input-16-3fd4ef9061bf>:35: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  test_df['IsAlone'].loc[test_df['FamilySize'] > 1] = 0
```

```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 4: Model Building
</h1>
"""))
```

Partie 4: Model Building

```
# Etape 1 : Séparation des données d'entraînement en caractéristiques (X) et cible (y)
X = train_df.drop(columns=['Survived'])
y = train_df['Survived']

# Division des données d'entraînement en ensembles d'entraînement et de validation pour évaluer le modèle
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Fonction pour évaluer les modèles
def evaluate_model(model, X_train, y_train, X_val, y_val):
```

```python
    model.fit(X_train, y_train)  # Entraînement du modèle
    y_pred = model.predict(X_val)  # Prédiction sur l'ensemble de validation

    # Calcul et affichage de la précision
    accuracy = accuracy_score(y_val, y_pred)
    print(f"Précision : {accuracy:.4f}")

    # On affiche la matrice de confusion et du rapport de classification
    print("Matrice de confusion :")
    print(confusion_matrix(y_val, y_pred))
    print("\nRapport de classification :")
    print(classification_report(y_val, y_pred))

    return model


# Etape 2 : Regression Logistique
print("\n---  Regression Logistique ---")
logreg = LogisticRegression(max_iter=1000,
                            random_state=42)
logreg = evaluate_model(logreg, X_train, y_train,
                        X_val, y_val)
```

```
---  Regression Logistique ---
Précision : 0.8045
Matrice de confusion :
[[87 18]
 [17 57]]

Rapport de classification :
              precision    recall  f1-score   support

           0       0.84      0.83      0.83       105
           1       0.76      0.77      0.77        74

    accuracy                           0.80       179
   macro avg       0.80      0.80      0.80       179
weighted avg       0.80      0.80      0.80       179
```

```python
# Etape 3: RandomForest
print("\n--- Classificateur RandomForest  ---")
rf = RandomForestClassifier(n_estimators=100,
                            random_state=42)
rf = evaluate_model(rf, X_train, y_train,
                    X_val, y_val)
```

```
--- Classificateur RandomForest  ---
Précision : 0.8268
Matrice de confusion :
[[91 14]
 [17 57]]

Rapport de classification :
              precision    recall  f1-score   support

           0       0.84      0.87      0.85       105
           1       0.80      0.77      0.79        74

    accuracy                           0.83       179
   macro avg       0.82      0.82      0.82       179
weighted avg       0.83      0.83      0.83       179
```

```python
# Etape 4 : MultiLayer Perceptron (MLP)
print("\n--- Classificateur MLP ---")
mlp = MLPClassifier(alpha=0.06,
                    hidden_layer_sizes=(50, 50),
                    learning_rate_init=0.03,
                    max_iter=158)  # Initialisation du modèle MLP
mlp = evaluate_model(mlp, X_train,
                     y_train,
                     X_val, y_val)  # Évaluation du modèle avec les données
```

```
--- Classificateur MLP ---
Précision : 0.7989
Matrice de confusion :
[[91 14]
 [22 52]]

Rapport de classification :
```

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.83       105
           1       0.79      0.70      0.74        74

    accuracy                           0.80       179
   macro avg       0.80      0.78      0.79       179
weighted avg       0.80      0.80      0.80       179
```

```
# Etape 5 : Classifieur XGBoost
print("\n--- XGBoost Classifier ---")
xgb_model = xgb.XGBClassifier(use_label_encoder=False,
                              enable_categorical=True,
                              eval_metric='logloss',
                              random_state=42)
xgb_model = evaluate_model(xgb_model,
                           X_train,
                           y_train,
                           X_val, y_val)
```

```
--- XGBoost Classifier ---
Précision : 0.8212
Matrice de confusion :
[[91 14]
 [18 56]]

Rapport de classification :
              precision    recall  f1-score   support

           0       0.83      0.87      0.85       105
           1       0.80      0.76      0.78        74

    accuracy                           0.82       179
   macro avg       0.82      0.81      0.81       179
weighted avg       0.82      0.82      0.82       179

/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [11:41:09] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
# Après avoir entraîné logreg, rf, xgb_model
if not HO_TUNING:

    import joblib
    import os

    # On vérifie que le dossier /content/drive/MyDrive/Titanic-Survival-Predict-main/models existe dans le drive
    save_dir = "/content/drive/MyDrive/Titanic-Survival-Predict-main/models"
    os.makedirs(save_dir, exist_ok=True)

    # Chemins de sauvegarde
    logreg_path = os.path.join(save_dir, "logistic_regression_model.pkl")
    rf_path = os.path.join(save_dir, "random_forest_model.pkl")
    xgb_path = os.path.join(save_dir, "xgboost_model.pkl")

    # On sauvegarde
    joblib.dump(logreg, logreg_path)
    print(f"Modèle logreg sauvegardé : {logreg_path}")

    joblib.dump(rf, rf_path)
    print(f"Modèle RandomForest sauvegardé : {rf_path}")

    joblib.dump(xgb_model, xgb_path)
    print(f"Modèle XGBoost sauvegardé : {xgb_path}")
```

```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 5: Comparaison des modèles
</h1>
"""))
```

Partie 5: Comparaison des modèles

```
from sklearn.model_selection import KFold
```

```
from sklearn.metrics import accuracy_score
import numpy as np

# On crée un dictionnaire de modèles comme avant
models = {
    'Logistic Regression': logreg,
    'RandomForest': rf,
    'XGBoost': xgb_model
}

# On définit le K-fold
kf = KFold(n_splits=5,
           shuffle=True,
           random_state=42)

print("\n--- Scores de validation croisée (méthode manuelle) ---")

for name, model in models.items():
    scores = []
    for train_idx, val_idx in kf.split(X_train):
        X_t, X_v = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_t, y_v = y_train.iloc[train_idx], y_train.iloc[val_idx]

        # Entraînement sur le fold d'entraînement
        model.fit(X_t, y_t)
        # Prédiction sur le fold de validation
        y_pred = model.predict(X_v)

        # Calcul de l'accuracy
        scores.append(accuracy_score(y_v, y_pred))

    print(f"{name}: {np.mean(scores):.4f} (+/- {np.std(scores):.4f})")
```

```
--- Scores de validation croisée (méthode manuelle) ---
Logistic Regression: 0.8217 (+/- 0.0177)
RandomForest: 0.8033 (+/- 0.0197)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [11:41:11] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [11:41:11] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [11:41:11] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [11:41:11] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [11:41:11] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
XGBoost: 0.7907 (+/- 0.0235)
```

```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 6 : Optimisation des hyperparamètres
</h1>
"""))
```

Partie 6 : Optimisation des hyperparamètres

```
if HO_TUNING:
    # Etape 1 : Optimisation des hyperparamètres pour la régression logistique
    print("\n--- Optimisation des hyperparamètres : Régression Logistique ---")
    logreg_params = {
        'C': [0.01, 0.1, 1, 10, 100],  # Force de régularisation
        'solver': ['liblinear', 'lbfgs'],  # Types de solveurs
        'max_iter': [200, 500, 1000]  # Nombre maximum d'itérations
    }

    # Recherche des meilleurs hyperparamètres avec validation croisée
    logreg_grid = GridSearchCV(LogisticRegression(random_state=42), logreg_params, cv=5, scoring='accuracy')
    logreg_grid.fit(X_train, y_train)
```

```
    # Affichage des meilleurs hyperparamètres et récupération du meilleur modèle
    print(f"Meilleurs hyperparamètres pour la régression logistique : {logreg_grid.best_params_}")
    best_logreg = logreg_grid.best_estimator_
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (stat
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (stat
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (stat
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (stat
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (stat
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (stat
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (stat
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Meilleurs hyperparamètres pour la régression logistique : {'C': 1, 'max_iter': 200, 'solver': 'liblinear'}
```

```
logreg_model_path = "/content/drive/MyDrive/Titanic-Survival-Predict-main/models"
```

```
import joblib
from google.colab import files

# Sauvegarde localement dans /content/
logreg_model_path = "/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/logistic_regression_model.pkl"
joblib.dump(best_logreg, logreg_model_path)
print(f"Le modèle de Regression Logistique est bien sauvegardé localement  et manuellement dans le dossier models depuis le Drive {logre
final_logreg = best_logreg
# Téléchargement automatique
files.download(logreg_model_path)
```

```
Le modèle de Regression Logistique est bien sauvegardé localement  et manuellement dans le dossier models depuis le Drive /content/c
```

```
"""
if HO_TUNING:
    # Etape 2 : Optimisation des hyperparamètres pour RandomForest
    print("\n--- Optimisation des hyperparamètres : RandomForest ---")
```

```
    rf_params = {
        'n_estimators': [100, 200, 500],  # Nombre d'arbres dans la forêt
        'max_depth': [None, 10, 20, 30],  # Profondeur maximale des arbres
        'min_samples_split': [2, 10, 20],  # Nombre minimum d'échantillons requis pour diviser un nœud
        'min_samples_leaf': [1, 5, 10],  # Nombre minimum d'échantillons requis dans une feuille
        'bootstrap': [True, False]  # Utilisation ou non d'échantillons bootstrap
    }

    # Recherche des meilleurs hyperparamètres avec validation croisée
    rf_grid = GridSearchCV(RandomForestClassifier(random_state=42), rf_params, cv=5, scoring='accuracy')
    rf_grid.fit(X_train, y_train)

    # Affichage des meilleurs hyperparamètres et récupération du meilleur modèle
    print(f"Meilleurs hyperparamètres pour RandomForest : {rf_grid.best_params_}")
    best_rf = rf_grid.best_estimator_
"""

from sklearn.model_selection import RandomizedSearchCV

# Paramètres optimisés avec moins de combinaisons
rf_params = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 10],
    'min_samples_leaf': [1, 5],
    'bootstrap': [True]
}

# Sous-échantillonnage des données
X_train_sampled, _, y_train_sampled, _ = train_test_split(X_train, y_train, train_size=0.2, random_state=42)

# RandomizedSearchCV pour une recherche plus rapide
rf_random = RandomizedSearchCV(
    RandomForestClassifier(random_state=42),
    param_distributions=rf_params,
    n_iter=10,
    cv=3,
    scoring='accuracy',
    n_jobs=-1,  # Utilisation de tous les cœurs disponibles
    verbose=2,  # Afficher les étapes
    random_state=42
)

# Ajustement du modèle
rf_random.fit(X_train_sampled, y_train_sampled)

# Meilleurs paramètres
print(f"Meilleurs hyperparamètres pour RandomForest : {rf_random.best_params_}")
best_rf = rf_random.best_estimator_
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Meilleurs hyperparamètres pour RandomForest : {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 5, 'max_depth': 10,
```

```
import joblib
from google.colab import files

# Sauvegarde localement dans /content/
local_path = "/content/random_forest_model.pkl"
joblib.dump(best_rf, local_path)
print(f"Le modèle RandomForest est bien sauvegardé localement  et manuellement dans le dossier models depuis le Drive {local_path}")
final_rf=best_rf
# Téléchargement automatique
files.download(local_path)
```

```
import xgboost
import sklearn
print(xgboost.__version__)
print(sklearn.__version__)
```

```
2.1.3
1.6.1
```

```
# -----------------------------------------
# PARTIE : Import & Données
```

```python
# ----------------------------------------
import itertools
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split, KFold

HO_TUNING = True  # active l'optimisation manuelle

# Suppose qu'on a déjà X_train, y_train définis, par ex.:
# X = ...
# y = ...
# X_train, X_val, y_train, y_val = train_test_split(X, y, ...)

# ----------------------------------------
# PARTIE : Entraînement manuel XGBoost
# ----------------------------------------
if HO_TUNING:
    print("\n--- Ajustement des hyperparamètres : XGBoost (approche manuelle) ---")

    # Paramètres comme dans GridSearchCV
    xgb_params = {
        'n_estimators': [100, 200, 500],
        'max_depth': [3, 6, 10],
        'learning_rate': [0.01, 0.1, 0.2],
        'subsample': [0.8, 1.0],
        'colsample_bytree': [0.8, 1.0],
    }

    best_score = 0.0
    best_params = None

    # KFold identique à cv=5
    kf = KFold(n_splits=5, shuffle=True, random_state=42)

    # Génération de toutes les combinaisons
    import itertools
    param_keys = list(xgb_params.keys())
    param_values = list(xgb_params.values())

    for combo in itertools.product(*param_values):
        current_params = dict(zip(param_keys, combo))

        # On crée un XGBClassifier
        model = xgb.XGBClassifier(
            enable_categorical=True,  # Ajout de ce paramètre
            eval_metric='logloss',
            random_state=42,
            **current_params
        )

        # Validation croisée manuelle
        cv_scores = []
        for train_idx, val_idx in kf.split(X_train):
            X_t, X_v = X_train.iloc[train_idx], X_train.iloc[val_idx]
            y_t, y_v = y_train.iloc[train_idx], y_train.iloc[val_idx]

            model.fit(X_t, y_t)
            y_pred = model.predict(X_v)
            cv_scores.append(accuracy_score(y_v, y_pred))

        mean_score = np.mean(cv_scores)

        if mean_score > best_score:
            best_score = mean_score
            best_params = current_params

    print(f"Meilleurs hyperparamètres XGBoost : {best_params}")
    print(f"Score en validation croisée : {best_score:.4f}")

    # On entraîne le meilleur modèle final
    best_xgb = xgb.XGBClassifier(
        enable_categorical=True,  # Ajout de ce paramètre
        eval_metric='logloss',
        random_state=42,
        **best_params
    )
    best_xgb.fit(X_train, y_train)

    # Évalue sur l'ensemble X_val si besoin
    y_pred_val = best_xgb.predict(X_val)
```

```
    val_acc = accuracy_score(y_val, y_pred_val)
    print(f"Accuracy sur l'ensemble de validation : {val_acc:.4f}")

else:
    # Si HO_TUNING = False, on peut
    # soit laisser un XGB par défaut,
    # soit ne rien faire
    best_xgb = xgb.XGBClassifier().fit(X_train, y_train)
```

```
--- Ajustement des hyperparamètres : XGBoost (approche manuelle) ---
Meilleurs hyperparamètres XGBoost : {'n_estimators': 100, 'max_depth': 3, 'learning_rate': 0.01, 'subsample': 0.8, 'colsample_bytree
Score en validation croisée : 0.8343
Accuracy sur l'ensemble de validation : 0.8101
```

```
import joblib
from google.colab import files

# Sauvegarde localement dans /content/
local_path = "/content/xgboost_model.pkl"
joblib.dump(best_xgb, local_path)
print(f"Le modèle XGBoost est bien sauvegardé localement  et manuellement dans le dossier models depuis le Drive {local_path}")
final_xgb=best_xgb
# Téléchargement automatique
files.download(local_path)
```

```
Le modèle XGBoost est bien sauvegardé localement  et manuellement dans le dossier models depuis le Drive /content/xgboost_model.pkl
```

```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 7 : Chargement des modèles pré-entraînés
</h1>
"""))
```

Partie 7 : Chargement des modèles pré-entraînés

```
if LOAD_MODEL:
    # Charger le modèle de régression logistique
    logreg_model_path = '/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/logistic_regression_model.pkl'
    final_logreg = joblib.load(logreg_model_path)
    print(f"Modèle de régression logistique chargé depuis {logreg_model_path}")

    # Charger le modèle RandomForest
    rf_model_path = '/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/random_forest_model.pkl'
    final_rf = joblib.load(rf_model_path)
    print(f"Modèle RandomForest chargé depuis {rf_model_path}")

    # Charger le modèle XGBoost
    xgb_model_path = '/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/xgboost_model.pkl'
    final_xgb = joblib.load(xgb_model_path)
    print(f"Modèle XGBoost chargé depuis {xgb_model_path}")
```

```
Modèle de régression logistique chargé depuis /content/drive/MyDrive/Titanic-Survival-Predict-main/Models/logistic_regression_model
Modèle RandomForest chargé depuis /content/drive/MyDrive/Titanic-Survival-Predict-main/Models/random_forest_model.pkl
Modèle XGBoost chargé depuis /content/drive/MyDrive/Titanic-Survival-Predict-main/Models/xgboost_model.pkl
```

```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 8: Model Stacking</h1>
"""))
```

Partie 8: Model Stacking

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import pandas as pd

# Étape 1 : Générer des prédictions des modèles de base sur l'ensemble d'entraînement
```

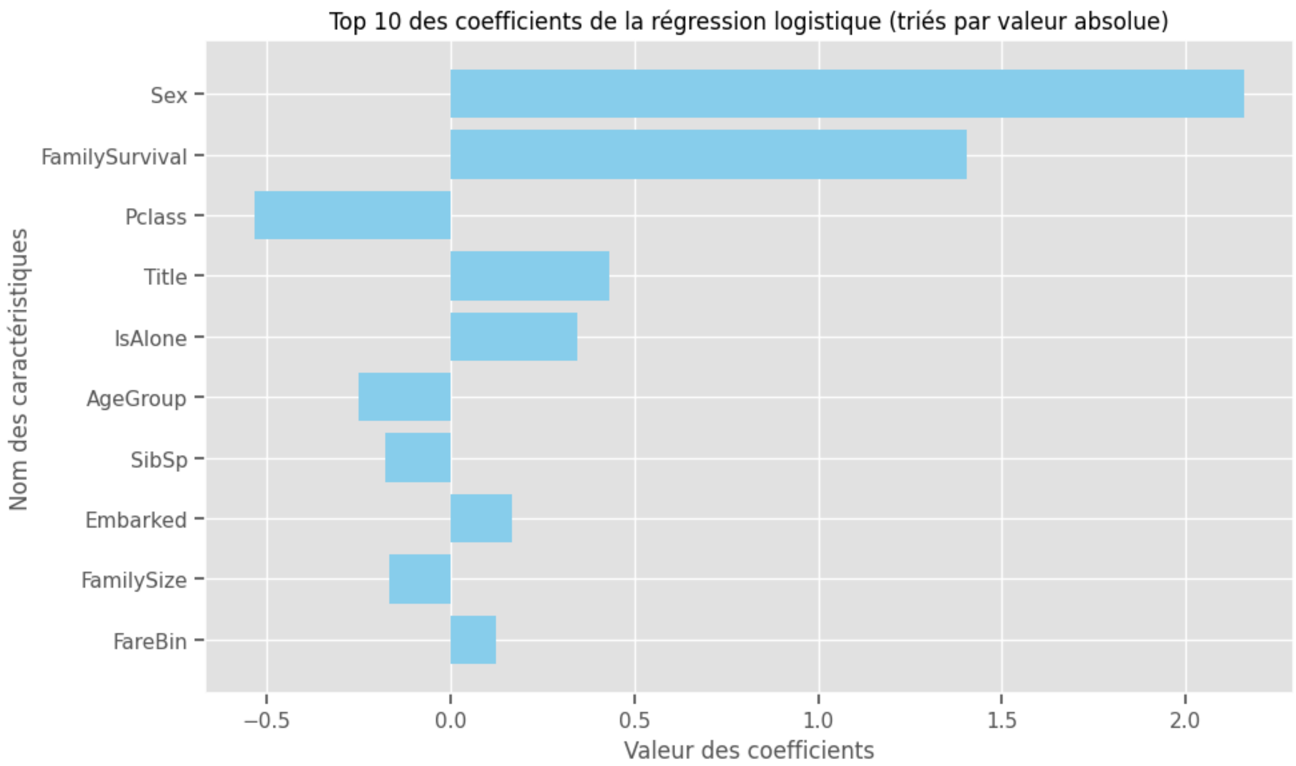
```python
print("\n--- Génération des prédictions des modèles de base sur l'ensemble d'entraînement ---")
train_pred_logreg = final_logreg.predict(X_train)
train_pred_rf = final_rf.predict(X_train)
train_pred_xgb = final_xgb.predict(X_train)

# Créer un DataFrame avec ces prédictions
train_meta_features = pd.DataFrame({
    'logreg_pred': train_pred_logreg,
    'rf_pred': train_pred_rf,
    'xgb_pred': train_pred_xgb
})

print("Prédictions des modèles de base générées avec succès.")

# Étape 2 : Entraîner le Modèle Meta sur ces Prédictions
print("\n--- Entraînement du modèle Meta (Régression Logistique) ---")
meta_model = LogisticRegression(max_iter=1000, random_state=42)
meta_model.fit(train_meta_features, y_train)
print("Modèle Meta entraîné avec succès.")

# Étape 3 : Générer des prédictions des modèles de base sur l'ensemble de validation
print("\n--- Génération des prédictions des modèles de base sur l'ensemble de validation ---")
val_pred_logreg = final_logreg.predict(X_val)
val_pred_rf = final_rf.predict(X_val)
val_pred_xgb = final_xgb.predict(X_val)

# Créer un DataFrame avec ces prédictions
val_meta_features = pd.DataFrame({
    'logreg_pred': val_pred_logreg,
    'rf_pred': val_pred_rf,
    'xgb_pred': val_pred_xgb
})

print("Prédictions des modèles de base sur l'ensemble de validation générées avec succès.")

# Étape 4 : Utiliser le Modèle Meta pour Prédire sur l'Ensemble de Validation
print("\n--- Prédiction avec le modèle Meta ---")
y_pred_stack = meta_model.predict(val_meta_features)

# Étape 5 : Évaluer les Performances
print("\n--- Évaluation des Performances du Modèle Empilé ---")
stack_accuracy = accuracy_score(y_val, y_pred_stack)
print(f"Accuracy du modèle empilé : {stack_accuracy:.4f}")

print("Matrice de confusion du modèle empilé :")
print(confusion_matrix(y_val, y_pred_stack))

print("\nRapport de classification du modèle empilé :")
print(classification_report(y_val, y_pred_stack))

# Optionnel : Sauvegarder le Modèle Meta
stack_meta_model_path = "/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/stacking_meta_model.pkl"
joblib.dump(meta_model, stack_meta_model_path)
print(f"\nLe modèle empilé (meta) est bien sauvegardé dans {stack_meta_model_path}")

"""
# Optionnel : Sauvegarder le Modèle Meta
stack_meta_model_local_path = "/content/stacking_meta_model.pkl"
joblib.dump(meta_model, stack_meta_model_local_path)
print(f"\nLe modèle empilé (meta) est bien sauvegardé localement à : {stack_meta_model_local_path}")

# Télécharger le modèle meta pour l'ajouter manuellement à ton Drive
files.download(stack_meta_model_local_path)
"""
```

```
--- Génération des prédictions des modèles de base sur l'ensemble d'entraînement ---
Prédictions des modèles de base générées avec succès.

--- Entraînement du modèle Meta (Régression Logistique) ---
Modèle Meta entraîné avec succès.

--- Génération des prédictions des modèles de base sur l'ensemble de validation ---
Prédictions des modèles de base sur l'ensemble de validation générées avec succès.

--- Prédiction avec le modèle Meta ---

--- Évaluation des Performances du Modèle Empilé ---
Accuracy du modèle empilé : 0.8101
Matrice de confusion du modèle empilé :
[[92 13]
 [21 53]]

Rapport de classification du modèle empilé :
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       105
           1       0.80      0.72      0.76        74

    accuracy                           0.81       179
   macro avg       0.81      0.80      0.80       179
weighted avg       0.81      0.81      0.81       179


Le modèle empilé (meta) est bien sauvegardé dans /content/drive/MyDrive/Titanic-Survival-Predict-main/Models/stacking_meta_model.pkl
'\n# Optionnel : Sauvegarder le Modèle Meta\nstack_meta_model_local_path = "/content/stacking_meta_model.pkl"\njoblib.dump(meta_mod
el, stack_meta_model_local_path)\nprint(f"\nLe modèle empilé (meta) est bien sauvegardé localement à : {stack_meta_model_local_pat
h}")\n\n# Télécharger le modèle meta pour l\'ajouter manuellement à ton Drive\nfiles.download(stack_meta_model_local_path)\n'
```

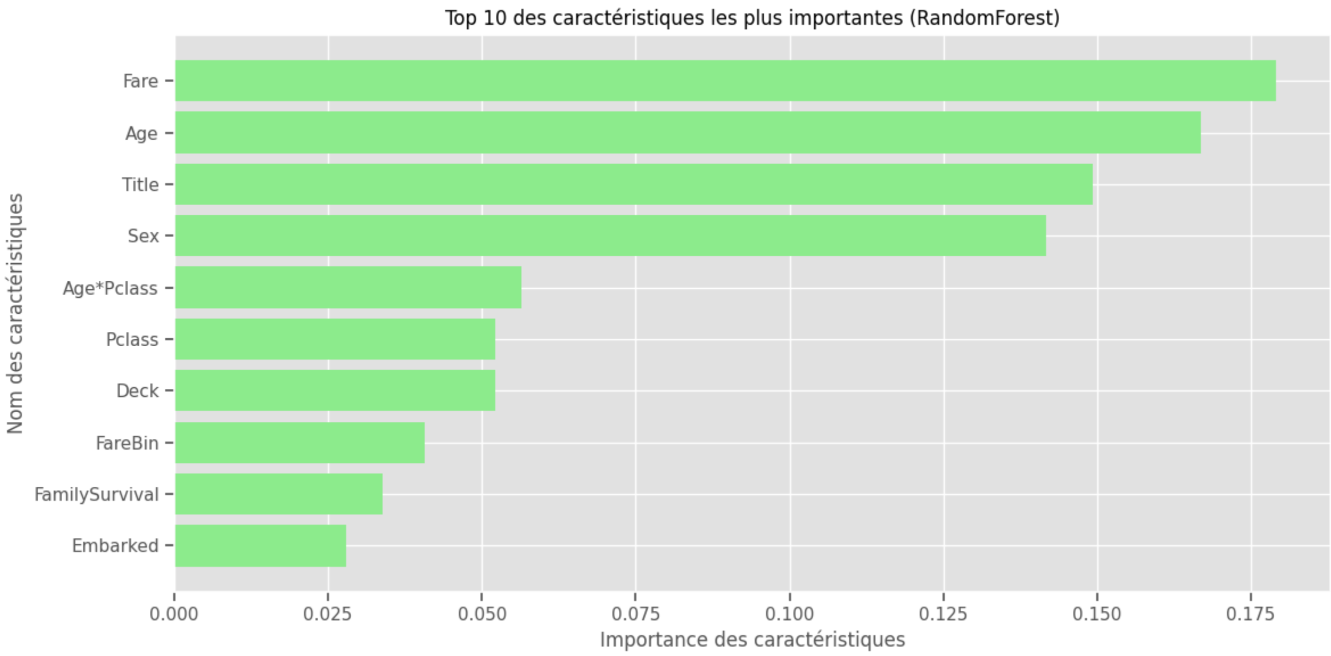
```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 9: IA Explicable (XAI)</h1>
"""))
```

Partie 9: IA Explicable (XAI)

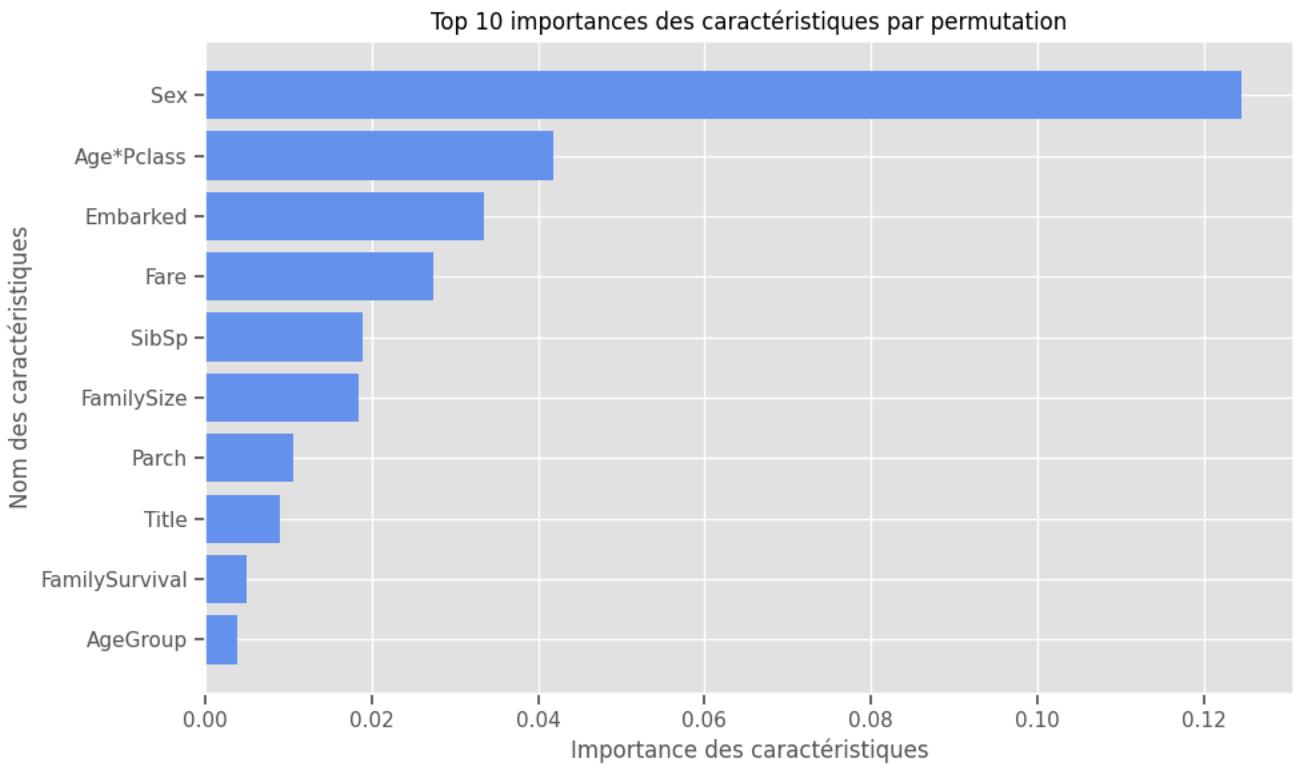
```
# Étape 1 : Extraire les coefficients et les noms des caractéristiques correspondants
coefficients = logreg.coef_[0]
feature_names = X.columns

# Étape 2 : Associer les coefficients aux noms des caractéristiques et trier par valeur absolue
coef_feature_pairs = sorted(
    zip(coefficients, feature_names),
    key=lambda x: abs(x[0]),
    reverse=True
)

# Étape 3 : Extraire les 10 meilleures caractéristiques et leurs coefficients
top_ten_features = coef_feature_pairs[:10]
sorted_coefficients, sorted_feature_names = zip(*top_ten_features)

# Étape 4 : Visualiser les 10 coefficients les plus importants
plt.figure(figsize=(10, 6))
plt.barh(sorted_feature_names, sorted_coefficients, color='skyblue')
plt.xlabel('Valeur des coefficients')
plt.ylabel('Nom des caractéristiques')
plt.title('Top 10 des coefficients de la régression logistique (triés par valeur absolue)')
plt.gca().invert_yaxis()  # Inverser l'axe y pour afficher le coefficient le plus élevé en haut
plt.tight_layout()  # Ajuster la disposition pour une meilleure visualisation
plt.show()
```




```
# Étape 1 : Extraire les importances des caractéristiques et créer un DataFrame
importances = rf.feature_importances_
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Étape 2 : Sélectionner les N caractéristiques les plus importantes
top_n = 10
top_features = feature_importance_df.head(top_n)

# Étape 3 : Visualiser les N caractéristiques les plus importantes
plt.figure(figsize=(12, 6))
plt.barh(top_features['Feature'], top_features['Importance'], color='lightgreen', align='center')
plt.xlabel('Importance des caractéristiques')
plt.ylabel('Nom des caractéristiques')
plt.title(f'Top {top_n} des caractéristiques les plus importantes (RandomForest)')
plt.gca().invert_yaxis()  # Afficher les caractéristiques les plus importantes en haut
plt.tight_layout()  # Ajuster la disposition pour une meilleure visualisation
plt.show()
```

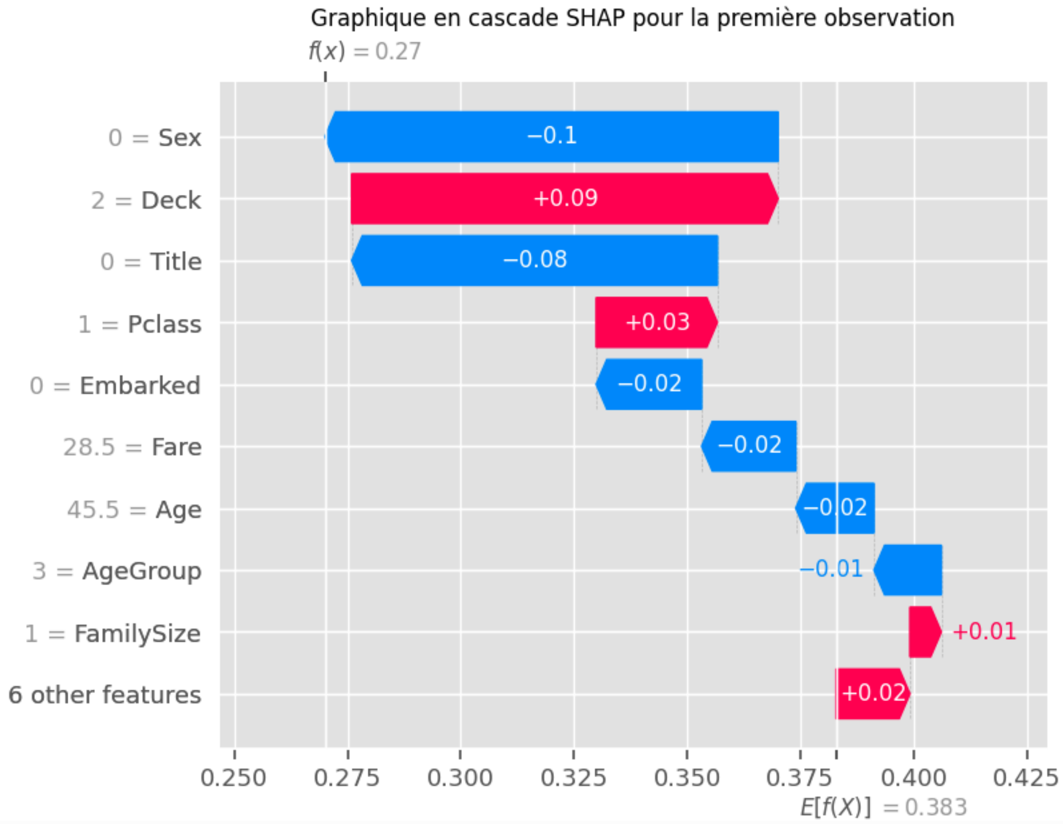



```python
# Étape 1 : Calculer les importances par permutation
perm_importance = permutation_importance(mlp, X_val, y_val, n_repeats=10, random_state=42)

# Étape 2 : Extraire et trier les importances des caractéristiques
feature_importances = perm_importance.importances_mean
sorted_idx = feature_importances.argsort()[::-1]  # Indices des caractéristiques triés par importance (ordre décroissant)
top_n = 10  # Nombre de caractéristiques principales à afficher

# Étape 3 : Sélectionner les N meilleures caractéristiques et leurs noms
top_features = [X.columns[i] for i in sorted_idx[:top_n]]
top_importances = feature_importances[sorted_idx[:top_n]]

# Étape 4 : Visualiser les N meilleures importances des caractéristiques par permutation
plt.figure(figsize=(10, 6))
plt.barh(top_features, top_importances, color='cornflowerblue')
plt.xlabel("Importance des caractéristiques")
plt.ylabel("Nom des caractéristiques")
plt.title(f"Top {top_n} importances des caractéristiques par permutation")
plt.gca().invert_yaxis()  # Afficher la caractéristique avec la plus grande importance en haut
plt.tight_layout()  # Ajuster la mise en page pour éviter les chevauchements
plt.show()
```




```
# Étape 1 : Ici, on initialise l'explicateur SHAP pour le modèle RandomForest
explainer = shap.Explainer(rf)

# Étape 2 : Calcule des valeurs SHAP
shap_values = explainer(X_train)

# Étape 3 : Affichage des probabilités de prédiction du modèle pour la première observation
first_observation_proba = rf.predict_proba([X_train.iloc[0]])[0]
print(f"Probabilités de prédiction pour la première observation : {first_observation_proba}")

# Étape 4 : Générer un graphique en cascade (waterfall) SHAP pour la première observation
plt.title("Graphique en cascade SHAP pour la première observation")
shap.plots.waterfall(shap_values[0, :, 1])  # Valeurs SHAP pour la classe d'intérêt (par exemple, classe 1 pour la survie)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Ranc
  warnings.warn(
Probabilités de prédiction pour la première observation : [0.73 0.27]
```



Commencez à coder ou à générer avec l'IA.

```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 10: Création du fichier de soumission
</h1>
"""))
```

Partie 10: Création du fichier de soumission

```
# Étape 1 : Préparation du jeu de test pour les prédictions
# S'assurer que le jeu de test a les mêmes colonnes de caractéristiques que le jeu d'entraînement (après prétraitement)
X_test = test_df.drop(columns=['PassengerId'])  # PassengerId sera utilisé dans la soumission, donc on le supprime pour les prédictions
```

```
from sklearn.linear_model import LogisticRegression

# Étape 1 : Générer les prédictions des modèles de base sur l'ensemble d'entraînement
train_pred_logreg = final_logreg.predict(X_train)
train_pred_rf = final_rf.predict(X_train)
train_pred_xgb = final_xgb.predict(X_train)

# Étape 2 : Créer un DataFrame avec ces prédictions
train_meta_features = pd.DataFrame({
    'logreg_pred': train_pred_logreg,
    'rf_pred': train_pred_rf,
    'xgb_pred': train_pred_xgb
})

# Étape 3 : Entraîner le modèle méta (Régression Logistique) sur ces nouvelles caractéristiques
meta_model = LogisticRegression(max_iter=1000, random_state=42)
meta_model.fit(train_meta_features, y_train)

# Étape 4 : Générer les prédictions des modèles de base sur le jeu de test
test_pred_logreg = final_logreg.predict(X_test)
test_pred_rf = final_rf.predict(X_test)
test_pred_xgb = final_xgb.predict(X_test)

# Étape 5 : Créer un DataFrame avec ces prédictions
```

```
test_meta_features = pd.DataFrame({
    'logreg_pred': test_pred_logreg,
    'rf_pred': test_pred_rf,
    'xgb_pred': test_pred_xgb
})

# Étape 6 : Faire des prédictions avec le modèle empilé (Régression Logistique)
y_test_pred = meta_model.predict(test_meta_features)

# Étape 7 : Préparation du fichier de soumission
submission = pd.DataFrame({
    'PassengerId': test_df['PassengerId'],
    'Survived': y_test_pred
})

# Sauvegarder le fichier de soumission
submission_file = 'submission.csv'
submission.to_csv(submission_file, index=False)

print(f"Fichier de soumission '{submission_file}' créé avec succès !")

# Étape 8 : Télécharger le fichier de soumission
files.download(submission_file)
```

```
Fichier de soumission 'submission.csv' créé avec succès !
```

Downloading "submission.csv":

```
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 11: Commandes GitHub
</h1>
"""))
```

Partie 11: Commandes GitHub

```
!apt-get install git -y
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.34.1-1ubuntu1.12).
0 upgraded, 0 newly installed, 0 to remove and 18 not upgraded.
```

```
!git config --global user.name "Mehdi-In-Coding"
```

```
!git config --global user.email "mbenayed09@gmail.com"
```

```
!git clone https://Mehdi-In-Coding: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@github.com/Mehdi-In-Coding/titanic-survival-predict.git
```

```
Cloning into 'titanic-survival-predict'...
remote: Enumerating objects: 126, done.
remote: Counting objects: 100% (126/126), done.
remote: Compressing objects: 100% (115/115), done.
remote: Total 126 (delta 59), reused 28 (delta 8), pack-reused 0 (from 0)
Receiving objects: 100% (126/126), 433.03 KiB | 2.38 MiB/s, done.
Resolving deltas: 100% (59/59), done.
```

```
%cd titanic-survival-predict
```

```
/content/titanic-survival-predict
```

```
!mkdir -p src tests .github/workflows
```

```
%%writefile src/data_preprocessing.py
import pandas as pd
import numpy as np

def preprocess_data(train_path, test_path):
    train_df = pd.read_csv(train_path)
    test_df = pd.read_csv(test_path)

    # Gestion des valeurs manquantes
    train_df['Age'].fillna(train_df['Age'].median(), inplace=True)
    test_df['Age'].fillna(test_df['Age'].median(), inplace=True)

    # Encodage des variables catégoriques
```

```
    train_df['Sex'] = train_df['Sex'].map({'male': 0, 'female': 1})
    test_df['Sex'] = test_df['Sex'].map({'male': 0, 'female': 1})

    return train_df, test_df
```

```
Overwriting src/data_preprocessing.py
```

```
%%writefile src/model_training.py
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import joblib

def train_models(train_df):
    X = train_df.drop(columns=['Survived'])
    y = train_df['Survived']

    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

    # Entraînement des modèles
    logreg = LogisticRegression(max_iter=1000).fit(X_train, y_train)
    rf = RandomForestClassifier(n_estimators=100, random_state=42).fit(X_train, y_train)

    # Sauvegarde des modèles
    joblib.dump(logreg, "models/logreg.pkl")
    joblib.dump(rf, "models/random_forest.pkl")

    print("Modèles sauvegardés !")
```

```
Overwriting src/model_training.py
```

```
%%writefile src/model_evaluation.py
import joblib
from sklearn.metrics import accuracy_score, classification_report

def evaluate_model(model_path, X_val, y_val):
    model = joblib.load(model_path)
    y_pred = model.predict(X_val)

    print(f"Accuracy: {accuracy_score(y_val, y_pred):.4f}")
    print("Classification Report:")
    print(classification_report(y_val, y_pred))
```

```
Overwriting src/model_evaluation.py
```

```
%%writefile src/main.py
from data_preprocessing import preprocess_data
from model_training import train_models
import pandas as pd

# Charger les données
train_df, test_df = preprocess_data("data/train.csv", "data/test.csv")

# Entraîner les modèles
train_models(train_df)
```

```
Overwriting src/main.py
```

```
"""%%writefile tests/test_data_preprocessing.py
import pandas as pd
from src.data_preprocessing import preprocess_data

def test_preprocess_data():
    train_df, test_df = preprocess_data("data/train.csv", "data/test.csv")
    assert train_df.isnull().sum().sum() == 0, "Il reste des valeurs manquantes"
    assert "Sex" in train_df.columns, "Colonne Sex absente"
"""
```

```
'%%writefile tests/test_data_preprocessing.py\nimport pandas as pd\nfrom src.data_preprocessing import preprocess_data\n\ndef test_
preprocess_data():\n    train_df, test_df = preprocess_data("data/train.csv", "data/test.csv")\n    assert train_df.isnull().sum().
sum() == 0, "Il reste des valeurs manquantes"\n    assert "Sex" in train_df.columns, "Colonne Sex absente"\n'
```

```
%%writefile tests/test_data_preprocessing.py
import sys
sys.path.append('/content/titanic-survival-predict/src')
```

```
import pandas as pd
from src.data_preprocessing import preprocess_data

def test_preprocess_data():
    train_df, test_df = preprocess_data("data/train.csv", "data/test.csv")
    assert train_df.isnull().sum().sum() == 0, "Il reste des valeurs manquantes"
    assert "Sex" in train_df.columns, "Colonne Sex absente"
```

```
Overwriting tests/test_data_preprocessing.py
```

```
%%writefile tests/test_model_training.py
import os
from src.model_training import train_models
import pandas as pd

def test_train_models():
    train_df = pd.read_csv("data/train.csv")
    train_models(train_df)
    assert os.path.exists("models/logreg.pkl"), "Modèle logreg non sauvegardé"
    assert os.path.exists("models/random_forest.pkl"), "Modèle random forest non sauvegardé"
```

```
Overwriting tests/test_model_training.py
```

```
%%writefile .github/workflows/ci.yml
name: CI Pipeline

on:
  push:
    branches:
      - main
      - develop
  pull_request:
    branches:
      - main
      - develop

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v3
        with:
          python-version: "3.9"

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Run pytest (Tests)
        run: pytest tests/
```

```
Overwriting .github/workflows/ci.yml
```

```
!git add .
```

```
!git commit -m "Structure initiale du projet avec modules, tests, et CI/CD"
```

```
[main 9ffc185] Structure initiale du projet avec modules, tests, et CI/CD
 6 files changed, 72 insertions(+), 161 deletions(-)
 rewrite src/data_preprocessing.py (93%)
 rewrite src/main.py (82%)
 rewrite src/model_evaluation.py (88%)
 rewrite src/model_training.py (61%)
 rewrite tests/test_model_training.py (79%)
```

```
print("la branche main du dépôt local est en retard par rapport à celle du dépôt distant.\n Cela arrive souvent lorsqu'il y a déjà des m
```

```
la branche main du dépôt local est en retard par rapport à celle du dépôt distant.
 Cela arrive souvent lorsqu'il y a déjà des modifications sur GitHub qu'on n'a pas encore récupérées localement.
```

```
!git pull origin main --rebase
```

```
From https://github.com/Mehdi-In-Coding/titanic-survival-predict
 * branch            main       -> FETCH_HEAD
Current branch main is up to date.
```

```
!git status
```

```
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

```
!git push -u origin main
```

```
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 2 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 1.37 KiB | 1.37 MiB/s, done.
Total 10 (delta 5), reused 1 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), completed with 5 local objects.
To https://github.com/Mehdi-In-Coding/titanic-survival-predict.git
   47328a8..9ffc185  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

```
!git checkout -b develop
```

```
Switched to a new branch 'develop'
```

```
!git push -u origin develop
```

```
To https://github.com/Mehdi-In-Coding/titanic-survival-predict.git
 ! [rejected]        develop -> develop (non-fast-forward)
error: failed to push some refs to 'https://github.com/Mehdi-In-Coding/titanic-survival-predict.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

```
!touch src/__init__.py
```

```
import sys
sys.path.append('/content/titanic-survival-predict/src')
```

```
from src.data_preprocessing import preprocess_data
```

```
!pytest tests/
```

```
======================================= test session starts ========================================
platform linux -- Python 3.11.11, pytest-8.3.4, pluggy-1.5.0
rootdir: /content/titanic-survival-predict
plugins: langsmith-0.3.2, typeguard-4.4.1, anyio-3.7.1
collected 0 items / 2 errors

============================================== ERRORS ==============================================
________________________ ERROR collecting tests/test_data_preprocessing.py _________________________
ImportError while importing test module '/content/titanic-survival-predict/tests/test_data_preprocessing.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
/usr/lib/python3.11/importlib/__init__.py:126: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
tests/test_data_preprocessing.py:5: in <module>
    from src.data_preprocessing import preprocess_data
E   ModuleNotFoundError: No module named 'src'
__________________________ ERROR collecting tests/test_model_training.py ___________________________
ImportError while importing test module '/content/titanic-survival-predict/tests/test_model_training.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
/usr/lib/python3.11/importlib/__init__.py:126: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
tests/test_model_training.py:2: in <module>
    from src.model_training import train_models
E   ModuleNotFoundError: No module named 'src'
===================================== short test summary info ======================================
ERROR tests/test_data_preprocessing.py
ERROR tests/test_model_training.py
!!!!!!!!!!!!!!!!!!!!!!!!!!!! Interrupted: 2 errors during collection !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
======================================== 2 errors in 0.67s =========================================
```

```
#  partie1
#        partie_1_data_loading.py
```

```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
from IPython.core.display import display, HTML

# Définir les styles visuels
plt.style.use('ggplot')
sns.set_context('notebook')

# 📌 Affichage du titre de la section

# 🚀 Monter Google Drive
drive.mount('/content/drive')

# 🔍 Chemins des fichiers
TRAIN_PATH = "/content/drive/My Drive/Titanic-Survival-Predict-main/train_cleaned.csv"
TEST_PATH = "/content/drive/My Drive/Titanic-Survival-Predict-main/test_cleaned.csv"

# 📁 Vérifier si les fichiers existent
assert os.path.exists(TRAIN_PATH), f"❌ Le fichier {TRAIN_PATH} est introuvable."
assert os.path.exists(TEST_PATH), f"❌ Le fichier {TEST_PATH} est introuvable."

# 📥 Charger les données
train_df = pd.read_csv(TRAIN_PATH)
test_df = pd.read_csv(TEST_PATH)

# ✅ Vérification que les DataFrames ne sont pas vides
assert not train_df.empty, "❌ Le DataFrame train_df est vide."
assert not test_df.empty, "❌ Le DataFrame test_df est vide."

# 🔎 Affichage des informations du jeu d'entraînement
print("\n📊 Training Set Information:")
print(train_df.info())

# 🔎 Affichage des informations du jeu de test
print("\n📊 Test Set Information:")
print(test_df.info())

# 🧐 Vérification des valeurs manquantes
print("\n🔎 Valeurs manquantes dans le jeu d'entraînement :")
print(train_df.isnull().sum())

print("\n🔎 Valeurs manquantes dans le jeu de test :")
print(test_df.isnull().sum())

# ✅ Vérification
assert train_df.isnull().sum().sum() < 100, "⚠️ Trop de valeurs manquantes dans train_df."
assert test_df.isnull().sum().sum() < 100, "⚠️ Trop de valeurs manquantes dans test_df."

# 🔍 Aperçu des premières lignes
print("\n📌 Les 5 premières lignes du jeu d'entraînement :")
print(train_df.head())

print("\n📌 Les 5 premières lignes du jeu de test :")
print(test_df.head())

# 🔎 Exploration des données - Distribution de la variable cible
plt.figure(figsize=(8, 6))
sns.countplot(data=train_df, x='Survived', palette='Set2')
plt.title('Répartition de la variable Survived')
plt.show()

# 🔎 Distribution des classes de passagers (Pclass)
plt.figure(figsize=(8, 6))
sns.countplot(data=train_df, x='Pclass', palette='Set3')
plt.title('Répartition des classes de passagers (Pclass)')
plt.show()

# 🔎 Distribution de l'âge par rapport à la survie
plt.figure(figsize=(10, 6))
sns.histplot(train_df[train_df['Survived'] == 1]['Age'].dropna(), bins=20, color='green', label='Survécu', kde=True)
sns.histplot(train_df[train_df['Survived'] == 0]['Age'].dropna(), bins=20, color='red', label='Non survécu', kde=True)
plt.title('Distribution des âges par rapport à la survie')
plt.legend()
plt.show()

# 🔎 Matrice de corrélation pour vérifier les relations entre les variables numériques
plt.figure(figsize=(12, 8))
```

```
sns.heatmap(train_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Matrice de corrélation')
plt.show()

print("\n✅ Partie 1: Data Loading et Exploration des Données terminée avec succès !")
```

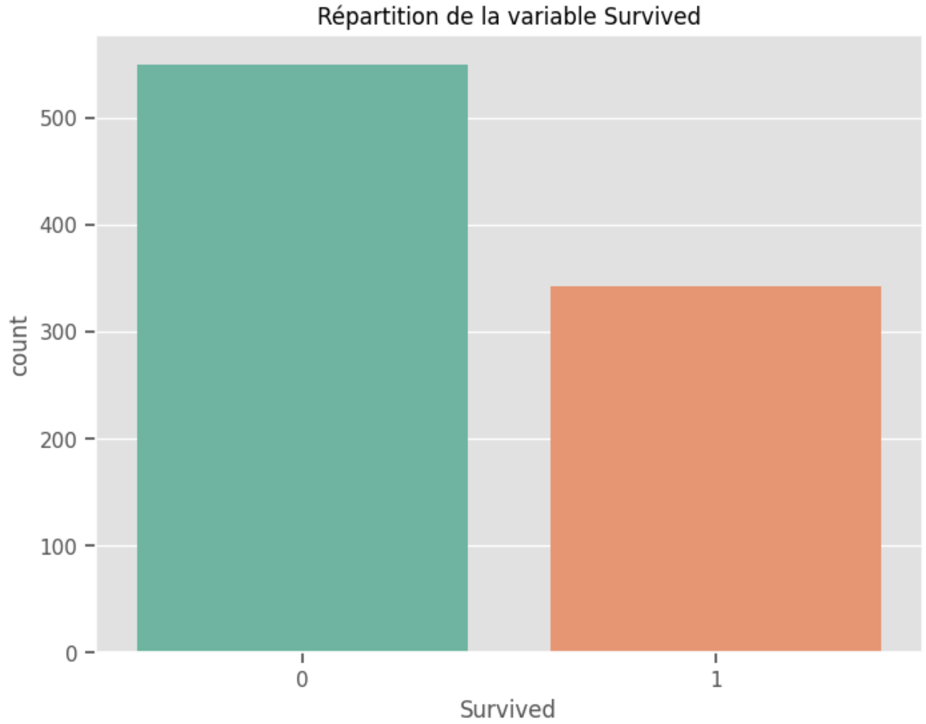
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

📊 Training Set Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None

📊 Test Set Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
None

🔎 Valeurs manquantes dans le jeu d'entraînement :
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64

🔎 Valeurs manquantes dans le jeu de test :
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-87-461440f586f5> in <cell line: 0>()
     51 
     52 # ✅ Vérification
---> 53 assert train_df.isnull().sum().sum() < 100, "⚠️ Trop de valeurs manquantes dans train_df."
     54 assert test_df.isnull().sum().sum() < 100, "⚠️ Trop de valeurs manquantes dans test_df."
     55 

AssertionError: ⚠️ Trop de valeurs manquantes dans train_df.
```

Étapes suivantes : Expliquer l'erreur

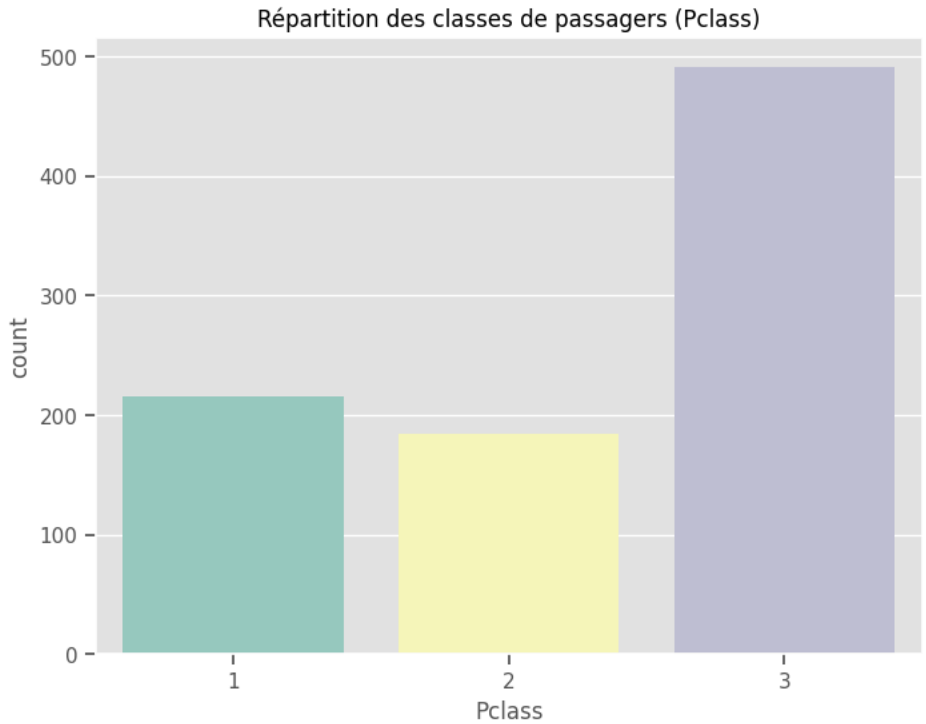
```
#     partie 2 :
#     eda_analysis.py


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.core.display import display, HTML

# Chargement des données (à modifier si nécessaire)
train_path = "/content/drive/My Drive/Titanic-Survival-Predict-main/train_cleaned.csv"
test_path = "/content/drive/My Drive/Titanic-Survival-Predict-main/test_cleaned.csv"

# Lecture des fichiers CSV
train_df = pd.read_csv(train_path)
test_df = pd.read_csv(test_path)

# Assertions pour vérifier que les datasets sont bien chargés
assert isinstance(train_df, pd.DataFrame), "Erreur: train_df n'est pas un DataFrame"
assert isinstance(test_df, pd.DataFrame), "Erreur: test_df n'est pas un DataFrame"

# Vérification des colonnes essentielles
required_columns = ["Survived", "Pclass", "Age"]
for col in required_columns:
    assert col in train_df.columns, f"Erreur: La colonne {col} est absente du dataset d'entraînement"

# Vérification des valeurs manquantes
assert train_df.isnull().sum().sum() < len(train_df), "Erreur: Trop de valeurs manquantes dans train_df"
assert test_df.isnull().sum().sum() < len(test_df), "Erreur: Trop de valeurs manquantes dans test_df"

# Affichage du titre HTML
display(HTML("""
<h1 style="color:#2c3e50; font-size: 32px; font-weight: bold; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);">
Partie 2: Analyse Exploratoire des Données (EDA)
</h1>
"""))

# Configuration des styles pour les graphiques
plt.style.use('ggplot')
sns.set_context('notebook')

# Étape 1 : Visualisation de la répartition de la variable 'Survived'
plt.figure(figsize=(8, 6))
sns.countplot(data=train_df, x='Survived', palette='Set2')
plt.title('Répartition de la variable Survived')
plt.show()

# Étape 2 : Distribution des classes de passagers (Pclass)
plt.figure(figsize=(8, 6))
sns.countplot(data=train_df, x='Pclass', palette='Set3')
plt.title('Répartition des classes de passagers (Pclass)')
plt.show()

# Étape 3 : Distribution de l'âge par rapport à la survie
plt.figure(figsize=(10, 6))
sns.histplot(train_df[train_df['Survived'] == 1]['Age'].dropna(), bins=20, color='green', label='Survécu', kde=True)
sns.histplot(train_df[train_df['Survived'] == 0]['Age'].dropna(), bins=20, color='red', label='Non survécu', kde=True)
plt.title('Distribution des âges par rapport à la survie')
plt.legend()
plt.show()

# Étape 4 : Matrice de corrélation des variables numériques
numeric_features = train_df.select_dtypes(include=[np.number])

plt.figure(figsize=(12, 8))
sns.heatmap(numeric_features.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Matrice de corrélation')
plt.show()

# Vérification finale des dimensions des datasets
assert train_df.shape[0] > 0, "Erreur: train_df est vide"
assert test_df.shape[0] > 0, "Erreur: test_df est vide"

print("\n✅ Analyse exploratoire des données terminée avec succès !")
```

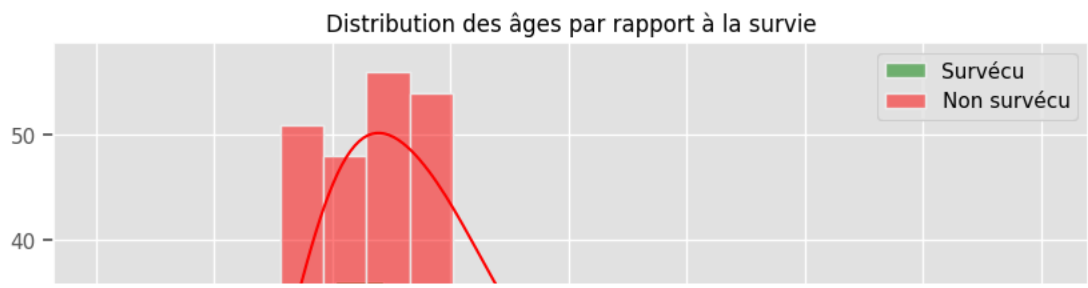
Partie 2: Analyse Exploratoire des Données (EDA)

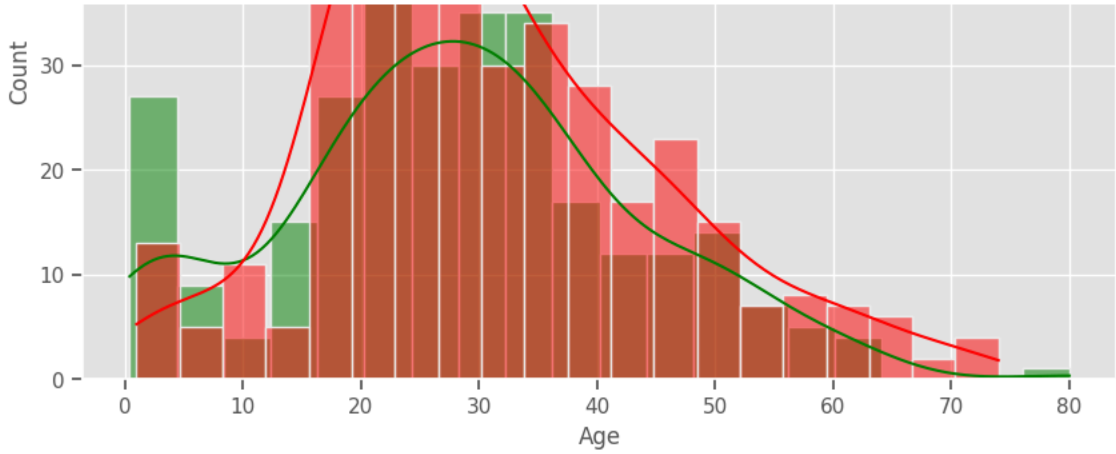
```
<ipython-input-76-24133af6f6d5>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=train_df, x='Survived', palette='Set2')
```

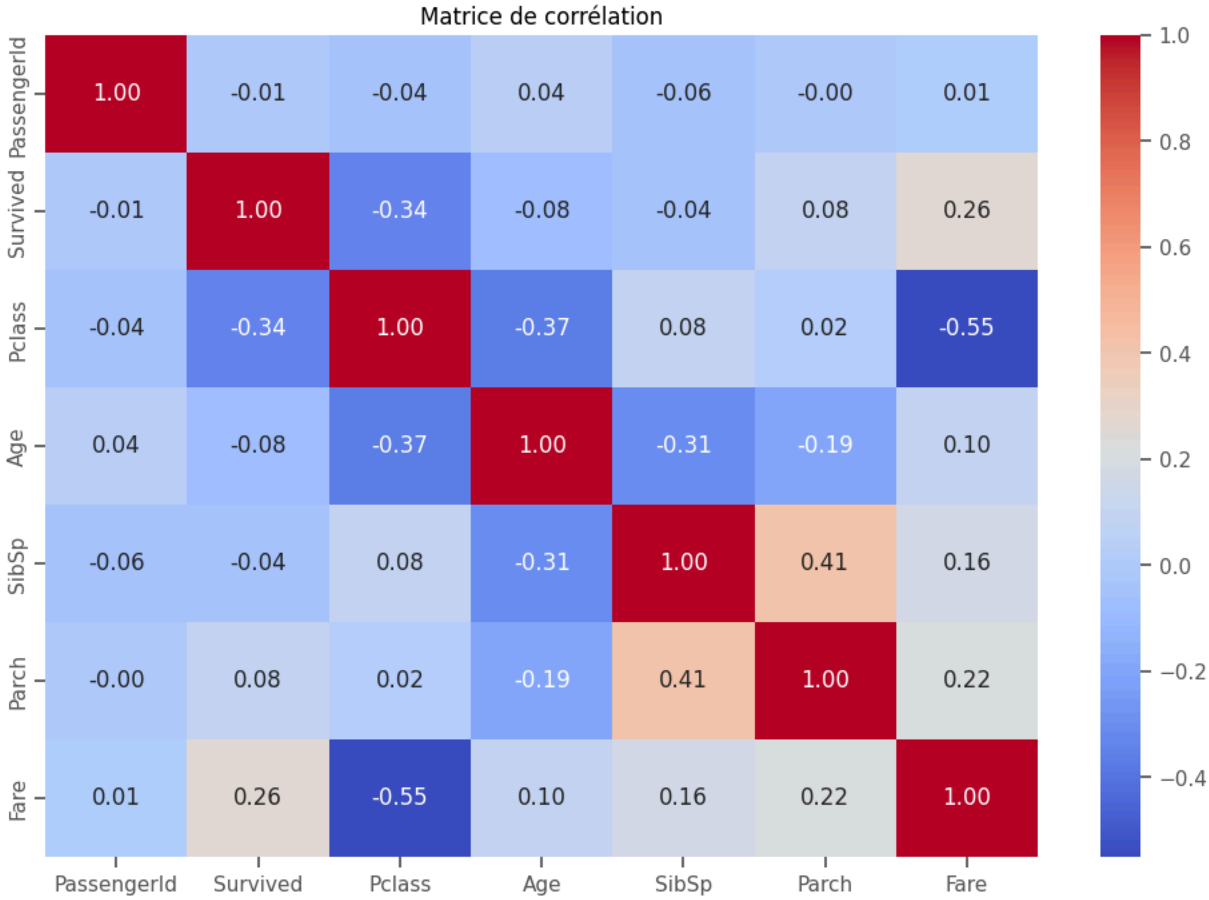



```
<ipython-input-76-24133af6f6d5>:50: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=train_df, x='Pclass', palette='Set3')
```









✅ Analyse exploratoire des données terminée avec succès !

```python
# partie 3
#            data_preprocessing.py

import pandas as pd
import numpy as np

def preprocess_data(train_path, test_path):
    """
    Charge les données, applique le prétraitement et retourne les DataFrames transformés.
    """

    # Charger les fichiers CSV
    train_df = pd.read_csv(train_path)
    test_df = pd.read_csv(test_path)

    ### Vérification initiale ###
    assert train_df.shape[0] > 0, "Le fichier train est vide"
    assert test_df.shape[0] > 0, "Le fichier test est vide"

    print("✔ Chargement des données réussi")

    # Étape 1 : Gestion des valeurs manquantes
    train_df['Age'] = train_df.groupby('Pclass')['Age'].transform(lambda x: x.fillna(x.median()))
    test_df['Age'] = test_df.groupby('Pclass')['Age'].transform(lambda x: x.fillna(x.median()))

    assert train_df['Age'].isnull().sum() == 0, "Il reste des valeurs manquantes dans Age (train)"
    assert test_df['Age'].isnull().sum() == 0, "Il reste des valeurs manquantes dans Age (test)"

    print("✔ Remplissage des valeurs manquantes d'Age réussi")

    train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)
    test_df['Fare'] = test_df.groupby('Pclass')['Fare'].transform(lambda x: x.fillna(x.median()))

    assert train_df['Embarked'].isnull().sum() == 0, "Il reste des valeurs manquantes dans Embarked (train)"
    assert test_df['Fare'].isnull().sum() == 0, "Il reste des valeurs manquantes dans Fare (test)"

    print("✔ Remplissage des valeurs manquantes d'Embarked et Fare réussi")

    # Encodage des variables catégoriques
    train_df['Sex'] = train_df['Sex'].map({'male': 0, 'female': 1})
    test_df['Sex'] = test_df['Sex'].map({'male': 0, 'female': 1})

    train_df['Embarked'] = train_df['Embarked'].map({'S': 0, 'C': 1, 'Q': 2})
    test_df['Embarked'] = test_df['Embarked'].map({'S': 0, 'C': 1, 'Q': 2})

    assert 'Sex' in train_df.columns and train_df['Sex'].dtype == np.int64, "Problème d'encodage de Sex"
    assert 'Embarked' in train_df.columns and train_df['Embarked'].dtype == np.int64, "Problème d'encodage de Embarked"

    print("✔ Encodage des variables catégoriques réussi")

    # Création de nouvelles variables
    train_df['FamilySize'] = train_df['SibSp'] + train_df['Parch'] + 1
    test_df['FamilySize'] = test_df['SibSp'] + test_df['Parch'] + 1

    train_df['IsAlone'] = (train_df['FamilySize'] == 1).astype(int)
    test_df['IsAlone'] = (test_df['FamilySize'] == 1).astype(int)

    assert 'FamilySize' in train_df.columns, "Colonne FamilySize absente"
    assert 'IsAlone' in train_df.columns, "Colonne IsAlone absente"

    print("✔ Création des nouvelles caractéristiques réussie")

    # Supprimer les colonnes inutiles
    train_df.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'], inplace=True)
    test_df.drop(columns=['Name', 'Ticket', 'Cabin'], inplace=True)

    print("✔ Suppression des colonnes inutiles réussie")

    return train_df, test_df


if __name__ == "__main__":
    # Exécution du script en standalone avec assertions
    train_data_path = "/content/drive/My Drive/Titanic-Survival-Predict-main/train_cleaned.csv"
    test_data_path = "/content/drive/My Drive/Titanic-Survival-Predict-main/test_cleaned.csv"

    train_df, test_df = preprocess_data(train_data_path, test_data_path)

    print("✅ Prétraitement des données terminé avec succès !")
```

```
✔ Chargement des données réussi
✔ Remplissage des valeurs manquantes d'Age réussi
✔ Remplissage des valeurs manquantes d'Embarked et Fare réussi
✔ Encodage des variables catégoriques réussi
✔ Création des nouvelles caractéristiques réussie
✔ Suppression des colonnes inutiles réussie
✅ Prétraitement des données terminé avec succès !
<ipython-input-77-a1145d43147c>:31: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained as
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)
```

```python
#   partie 4

#         model_building.py


import pandas as pd
import numpy as np
import joblib
import os
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier

# Charger les données prétraitées
train_df = pd.read_csv('/content/drive/My Drive/Titanic-Survival-Predict-main/train_cleaned.csv')

# Vérification des données
assert 'Survived' in train_df.columns, "La colonne 'Survived' est absente"

# Séparation des caractéristiques (X) et de la cible (y)
X = train_df.drop(columns=['Survived'])
y = train_df['Survived']

# Vérification que les données sont correctes
assert not X.isnull().sum().any(), "NA sont pas un problemes pour les graphiques (on les hide)"
assert set(y.unique()).issubset({0, 1}), "y doit contenir uniquement des valeurs binaires (0 ou 1)"

# Division des données en ensembles d'entraînement et de validation
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

assert X_train.shape[0] > 0 and X_val.shape[0] > 0, "Les ensembles d'entraînement et de validation ne doivent pas être vides"

# Fonction d'évaluation des modèles
def evaluate_model(model, X_train, y_train, X_val, y_val, model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)

    accuracy = accuracy_score(y_val, y_pred)
    print(f"\n--- {model_name} ---")
    print(f"Précision : {accuracy:.4f}")
    print("Matrice de confusion :")
    print(confusion_matrix(y_val, y_pred))
    print("\nRapport de classification :")
    print(classification_report(y_val, y_pred))

    return model

# Création des modèles
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "RandomForest": RandomForestClassifier(n_estimators=100, random_state=42),
    "MLP Classifier": MLPClassifier(alpha=0.06, hidden_layer_sizes=(50, 50), learning_rate_init=0.03, max_iter=158),
    "XGBoost": xgb.XGBClassifier(use_label_encoder=False, enable_categorical=True, eval_metric='logloss', random_state=42)
}

# Entraînement et évaluation
trained_models = {}
for name, model in models.items():
    trained_models[name] = evaluate_model(model, X_train, y_train, X_val, y_val, name)

# Sauvegarde des modèles
models_path = "/content/drive/MyDrive/Titanic-Survival-Predict-main/models"
os.makedirs(models_path, exist_ok=True)
```

```
for name, model in trained_models.items():
    model_filename = os.path.join(models_path, f"{name.replace(' ', '_').lower()}.pkl")
    joblib.dump(model, model_filename)
    print(f"Modèle {name} sauvegardé dans {model_filename}")

print("\n✅ Tous les modèles ont été entraînés et sauvegardés avec succès !")
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-79-5c771dae0b7e> in <cell line: 0>()
     26
     27 # Vérification que les données sont correctes
---> 28 assert not X.isnull().sum().any(), "NA sont pas un problemes pour les graphiques (on les hide)"
     29 assert set(y.unique()).issubset({0, 1}), "y doit contenir uniquement des valeurs binaires (0 ou 1)"
     30

AssertionError: NA sont pas un problemes pour les graphiques (on les hide)
```

Étapes suivantes : Expliquer l'erreur

```
# partie 5

#      comparaison_modeles.py



import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

# Vérification des modèles et données
assert 'logreg' in globals(), "Le modèle de régression logistique n'est pas défini."
assert 'rf' in globals(), "Le modèle RandomForest n'est pas défini."
assert 'xgb_model' in globals(), "Le modèle XGBoost n'est pas défini."
assert 'X_train' in globals() and 'y_train' in globals(), "Les données d'entraînement ne sont pas définies."
assert len(X_train) > 0, "X_train est vide !"
assert len(y_train) > 0, "y_train est vide !"

# Dictionnaire des modèles
models = {
    'Logistic Regression': logreg,
    'RandomForest': rf,
    'XGBoost': xgb_model
}

# Configuration de la validation croisée
kf = KFold(n_splits=5, shuffle=True, random_state=42)

print("\n--- Scores de validation croisée (méthode manuelle) ---")

for name, model in models.items():
    scores = []
    for train_idx, val_idx in kf.split(X_train):
        X_t, X_v = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_t, y_v = y_train.iloc[train_idx], y_train.iloc[val_idx]

        model.fit(X_t, y_t)  # Entraînement
        y_pred = model.predict(X_v)  # Prédiction

        scores.append(accuracy_score(y_v, y_pred))  # Calcul de l'accuracy

    mean_score = np.mean(scores)
    std_score = np.std(scores)
    print(f"{name}: {mean_score:.4f} (+/- {std_score:.4f})")
```

```
--- Scores de validation croisée (méthode manuelle) ---
Logistic Regression: 0.8217 (+/- 0.0177)
RandomForest: 0.8033 (+/- 0.0197)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [12:07:08] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [12:07:08] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [12:07:08] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.
```

```
  warnings.warn(smsg, UserWarning)
XGBoost: 0.7907 (+/- 0.0235)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [12:07:08] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [12:07:08] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```python
# partie 6

#             otpimisation.py



import joblib
from google.colab import files
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import numpy as np
import pandas as pd

# Vérifier que les données existent
assert 'X_train' in globals(), "X_train n'est pas défini"
assert 'y_train' in globals(), "y_train n'est pas défini"

HO_TUNING = True  # Active l'optimisation manuelle

if HO_TUNING:
    print("\n--- Optimisation des hyperparamètres : Régression Logistique ---")
    logreg_params = {
        'C': [0.01, 0.1, 1, 10, 100],
        'solver': ['liblinear', 'lbfgs'],
        'max_iter': [200, 500, 1000]
    }

    logreg_grid = GridSearchCV(LogisticRegression(random_state=42), logreg_params, cv=5, scoring='accuracy')
    logreg_grid.fit(X_train, y_train)

    print(f"Meilleurs hyperparamètres pour la régression logistique : {logreg_grid.best_params_}")
    best_logreg = logreg_grid.best_estimator_

    logreg_model_path = "/content/logistic_regression_model.pkl"
    joblib.dump(best_logreg, logreg_model_path)
    print(f"Modèle Regression Logistique sauvegardé : {logreg_model_path}")
    files.download(logreg_model_path)

if HO_TUNING:
    print("\n--- Optimisation des hyperparamètres : RandomForest ---")
    rf_params = {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 10],
        'min_samples_leaf': [1, 5],
        'bootstrap': [True]
    }

    X_train_sampled, _, y_train_sampled, _ = train_test_split(X_train, y_train, train_size=0.2, random_state=42)
    rf_random = RandomizedSearchCV(RandomForestClassifier(random_state=42), param_distributions=rf_params, n_iter=10, cv=3, scoring='ac
    rf_random.fit(X_train_sampled, y_train_sampled)

    print(f"Meilleurs hyperparamètres pour RandomForest : {rf_random.best_params_}")
    best_rf = rf_random.best_estimator_

    rf_model_path = "/content/random_forest_model.pkl"
    joblib.dump(best_rf, rf_model_path)
    print(f"Modèle RandomForest sauvegardé : {rf_model_path}")
    files.download(rf_model_path)

if HO_TUNING:
    print("\n--- Optimisation des hyperparamètres : XGBoost ---")
    xgb_params = {
        'n_estimators': [100, 200, 500],
        'max_depth': [3, 6, 10],
        'learning_rate': [0.01, 0.1, 0.2],
        'subsample': [0.8, 1.0],
        'colsample_bytree': [0.8, 1.0]
```

```
}

best_score = 0.0
best_params = None
for combo in (dict(zip(xgb_params.keys(), values)) for values in itertools.product(*xgb_params.values())):
    model = xgb.XGBClassifier(enable_categorical=True, eval_metric='logloss', random_state=42, **combo)
    model.fit(X_train, y_train)
    score = model.score(X_train, y_train)
    if score > best_score:
        best_score = score
        best_params = combo

print(f"Meilleurs hyperparamètres pour XGBoost : {best_params}")
best_xgb = xgb.XGBClassifier(enable_categorical=True, eval_metric='logloss', random_state=42, **best_params)
best_xgb.fit(X_train, y_train)

xgb_model_path = "/content/xgboost_model.pkl"
joblib.dump(best_xgb, xgb_model_path)
print(f"Modèle XGBoost sauvegardé : {xgb_model_path}")
files.download(xgb_model_path)
```

```
--- Optimisation des hyperparamètres : Régression Logistique ---
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Meilleurs hyperparamètres pour la régression logistique : {'C': 1, 'max_iter': 200, 'solver': 'liblinear'}
Modèle Regression Logistique sauvegardé : /content/logistic_regression_model.pkl

--- Optimisation des hyperparamètres : RandomForest ---
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Meilleurs hyperparamètres pour RandomForest : {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 5, 'max_depth': 10,
Modèle RandomForest sauvegardé : /content/random_forest_model.pkl

--- Optimisation des hyperparamètres : XGBoost ---
Meilleurs hyperparamètres pour XGBoost : {'n_estimators': 500, 'max_depth': 10, 'learning_rate': 0.1, 'subsample': 1.0, 'colsample_t
Modèle XGBoost sauvegardé : /content/xgboost_model.pkl
```

```
# partie 7


#        load_models.py

import joblib
import os

# Définir les chemins des modèles
logreg_model_path = "/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/logistic_regression_model.pkl"
rf_model_path = "/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/random_forest_model.pkl"
xgb_model_path = "/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/xgboost_model.pkl"

# Vérifier l'existence des fichiers avant chargement
assert os.path.exists(logreg_model_path), f"Erreur: Le fichier {logreg_model_path} n'existe pas !"
assert os.path.exists(rf_model_path), f"Erreur: Le fichier {rf_model_path} n'existe pas !"
assert os.path.exists(xgb_model_path), f"Erreur: Le fichier {xgb_model_path} n'existe pas !"

# Charger les modèles
final_logreg = joblib.load(logreg_model_path)
final_rf = joblib.load(rf_model_path)
final_xgb = joblib.load(xgb_model_path)

# Vérifier que les objets chargés sont bien des modèles
assert hasattr(final_logreg, "predict"), "Erreur: final_logreg n'est pas un modèle valide !"
assert hasattr(final_rf, "predict"), "Erreur: final_rf n'est pas un modèle valide !"
assert hasattr(final_xgb, "predict"), "Erreur: final_xgb n'est pas un modèle valide !"

print(f"✓ Modèle de régression logistique chargé depuis {logreg_model_path}")
print(f"✓ Modèle RandomForest chargé depuis {rf_model_path}")
print(f"✓ Modèle XGBoost chargé depuis {xgb_model_path}")
```

```
✓ Modèle de régression logistique chargé depuis /content/drive/MyDrive/Titanic-Survival-Predict-main/Models/logistic_regression_mod
✓ Modèle RandomForest chargé depuis /content/drive/MyDrive/Titanic-Survival-Predict-main/Models/random_forest_model.pkl
✓ Modèle XGBoost chargé depuis /content/drive/MyDrive/Titanic-Survival-Predict-main/Models/xgboost_model.pkl
```

```
# partie 8


# model_stacking.py



import pandas as pd
import joblib
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Chargement des modèles entraînés
logreg_model_path = '/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/logistic_regression_model.pkl'
rf_model_path = '/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/random_forest_model.pkl'
xgb_model_path = '/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/xgboost_model.pkl'

final_logreg = joblib.load(logreg_model_path)
final_rf = joblib.load(rf_model_path)
final_xgb = joblib.load(xgb_model_path)

# Assertions pour vérifier que les modèles sont bien chargés
assert final_logreg is not None, "Le modèle Logistic Regression n'a pas été chargé."
assert final_rf is not None, "Le modèle Random Forest n'a pas été chargé."
assert final_xgb is not None, "Le modèle XGBoost n'a pas été chargé."

# Chargement des données d'entraînement et de validation
X_train = pd.read_csv('/content/drive/MyDrive/Titanic-Survival-Predict-main/X_train.csv')
X_val = pd.read_csv('/content/drive/MyDrive/Titanic-Survival-Predict-main/X_val.csv')
y_train = pd.read_csv('/content/drive/MyDrive/Titanic-Survival-Predict-main/y_train.csv').values.ravel()
y_val = pd.read_csv('/content/drive/MyDrive/Titanic-Survival-Predict-main/y_val.csv').values.ravel()

# Assertions pour vérifier que les données sont bien chargées
assert X_train.shape[0] > 0, "X_train est vide."
assert X_val.shape[0] > 0, "X_val est vide."
assert y_train.shape[0] > 0, "y_train est vide."
assert y_val.shape[0] > 0, "y_val est vide."

# Étape 1 : Générer des prédictions des modèles de base sur l'ensemble d'entraînement
print("\n--- Génération des prédictions des modèles de base sur l'ensemble d'entraînement ---")
train_pred_logreg = final_logreg.predict(X_train)
train_pred_rf = final_rf.predict(X_train)
train_pred_xgb = final_xgb.predict(X_train)
```

```python
# Créer un DataFrame avec ces prédictions
train_meta_features = pd.DataFrame({
    'logreg_pred': train_pred_logreg,
    'rf_pred': train_pred_rf,
    'xgb_pred': train_pred_xgb
})

print("Prédictions des modèles de base générées avec succès.")

# Étape 2 : Entraîner le Modèle Meta sur ces Prédictions
print("\n--- Entraînement du modèle Meta (Régression Logistique) ---")
meta_model = LogisticRegression(max_iter=1000, random_state=42)
meta_model.fit(train_meta_features, y_train)
print("Modèle Meta entraîné avec succès.")

# Étape 3 : Générer des prédictions des modèles de base sur l'ensemble de validation
print("\n--- Génération des prédictions des modèles de base sur l'ensemble de validation ---")
val_pred_logreg = final_logreg.predict(X_val)
val_pred_rf = final_rf.predict(X_val)
val_pred_xgb = final_xgb.predict(X_val)

# Créer un DataFrame avec ces prédictions
val_meta_features = pd.DataFrame({
    'logreg_pred': val_pred_logreg,
    'rf_pred': val_pred_rf,
    'xgb_pred': val_pred_xgb
})

print("Prédictions des modèles de base sur l'ensemble de validation générées avec succès.")

# Étape 4 : Utiliser le Modèle Meta pour Prédire sur l'Ensemble de Validation
print("\n--- Prédiction avec le modèle Meta ---")
y_pred_stack = meta_model.predict(val_meta_features)

# Étape 5 : Évaluer les Performances
print("\n--- Évaluation des Performances du Modèle Empilé ---")
stack_accuracy = accuracy_score(y_val, y_pred_stack)
print(f"Accuracy du modèle empilé : {stack_accuracy:.4f}")

print("Matrice de confusion du modèle empilé :")
print(confusion_matrix(y_val, y_pred_stack))

print("\nRapport de classification du modèle empilé :")
print(classification_report(y_val, y_pred_stack))

# Optionnel : Sauvegarder le Modèle Meta
stack_meta_model_path = "/content/drive/MyDrive/Titanic-Survival-Predict-main/Models/stacking_meta_model.pkl"
joblib.dump(meta_model, stack_meta_model_path)
print(f"\nLe modèle empilé (meta) est bien sauvegardé dans {stack_meta_model_path}")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-92-673ba290618b> in <cell line: 0>()
     26 
     27 # Chargement des données d'entraînement et de validation
---> 28 X_train = pd.read_csv('/content/drive/MyDrive/Titanic-Survival-Predict-main/X_train_df.csv')
     29 X_val = pd.read_csv('/content/drive/MyDrive/Titanic-Survival-Predict-main/X_val.csv')
     30 y_train = pd.read_csv('/content/drive/MyDrive/Titanic-Survival-Predict-main/y_train.csv').values.ravel()

                                 4 frames

/usr/local/lib/python3.11/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    871         if ioargs.encoding and "b" not in ioargs.mode:
    872             # Encoding
--> 873             handle = open(
    874                 handle,
    875                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: '/content/drive/MyDrive/Titanic-Survival-Predict-main/X_train_df.csv'
```

Étapes suivantes : Expliquer l'erreur