Exercice4 : Évaluation et amélioration de modèles prédictifs

1) Modèle de base

1. Charger le jeu de données `Breast Cancer` depuis `sklearn.datasets`.
2. Séparer les données en un ensemble d'entraînement (70%) et un ensemble de test (30%) avec `random_state=42`.
3. Entraîner un modèle de **régression logistique** avec les paramètres par défaut.
4. Évaluer les performances du modèle sur l'ensemble de test en calculant : Accuracy, Precision, Recall, F1-score, ROC-AUC, Matrice de confusion

In [1]:
```
from sklearn.datasets import load_breast_cancer
import pandas as pd

data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

X.head()
```

Out[1]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20

5 rows × 30 columns

In [2]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

In [3]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix

model = LogisticRegression(max_iter=5000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:,1]

print(classification_report(y_test, y_pred))
print('ROC-AUC:', roc_auc_score(y_test, y_prob))
print('Confusion matrix:\n', confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        63
           1       0.98      0.98      0.98       108

    accuracy                           0.98       171
   macro avg       0.97      0.97      0.97       171
weighted avg       0.98      0.98      0.98       171

ROC-AUC: 0.9976484420928865
Confusion matrix:
 [[ 61   2]
 [  2 106]]
```

L'accuracy est-elle une métrique suffisante ?

Le jeu de données est-il équilibré ?

L'accuracy seule n'est pas suffisante si les classes sont déséquilibrées.

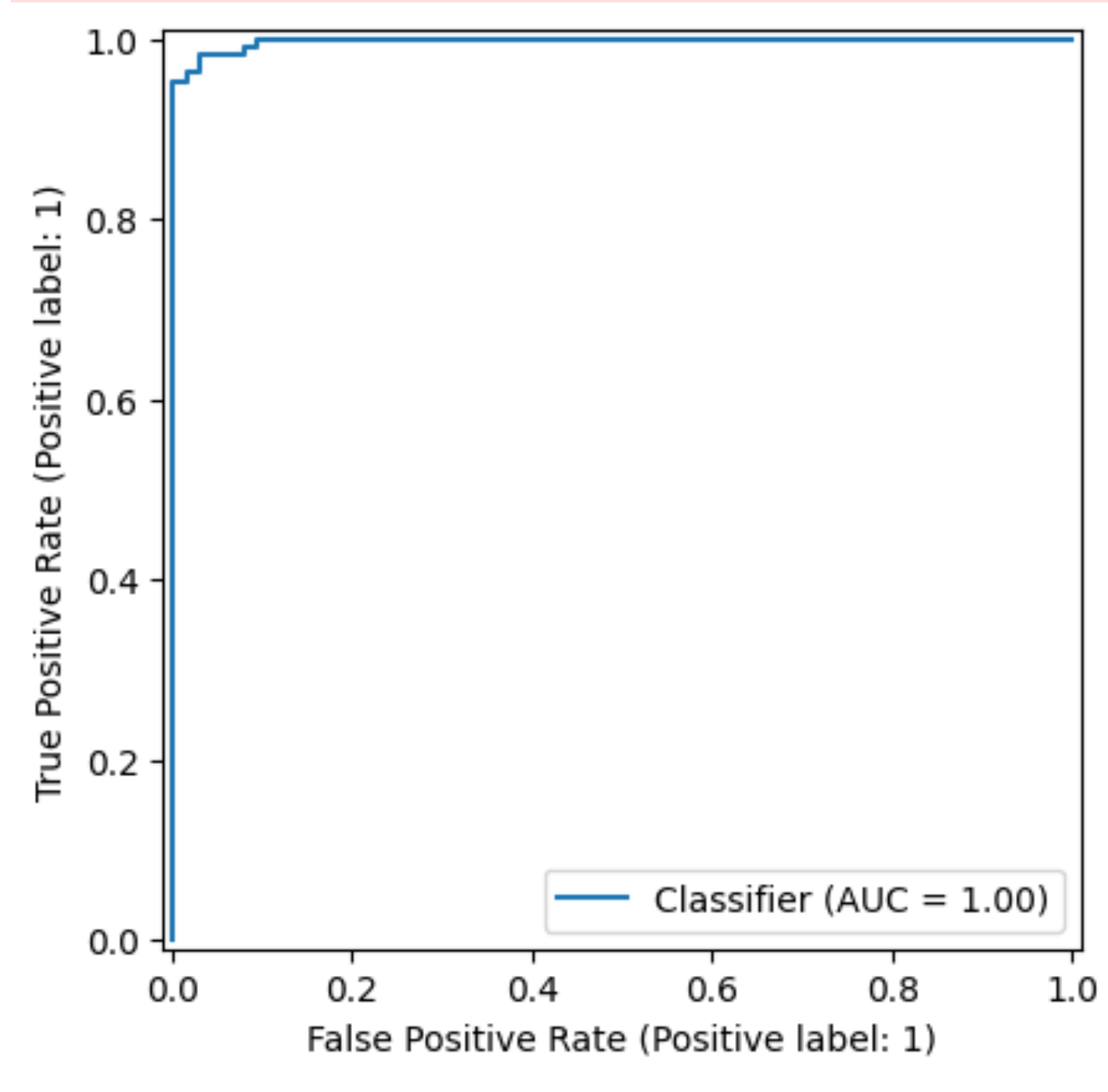
Le dataset est relativement équilibré.

2) Évaluation du modèle

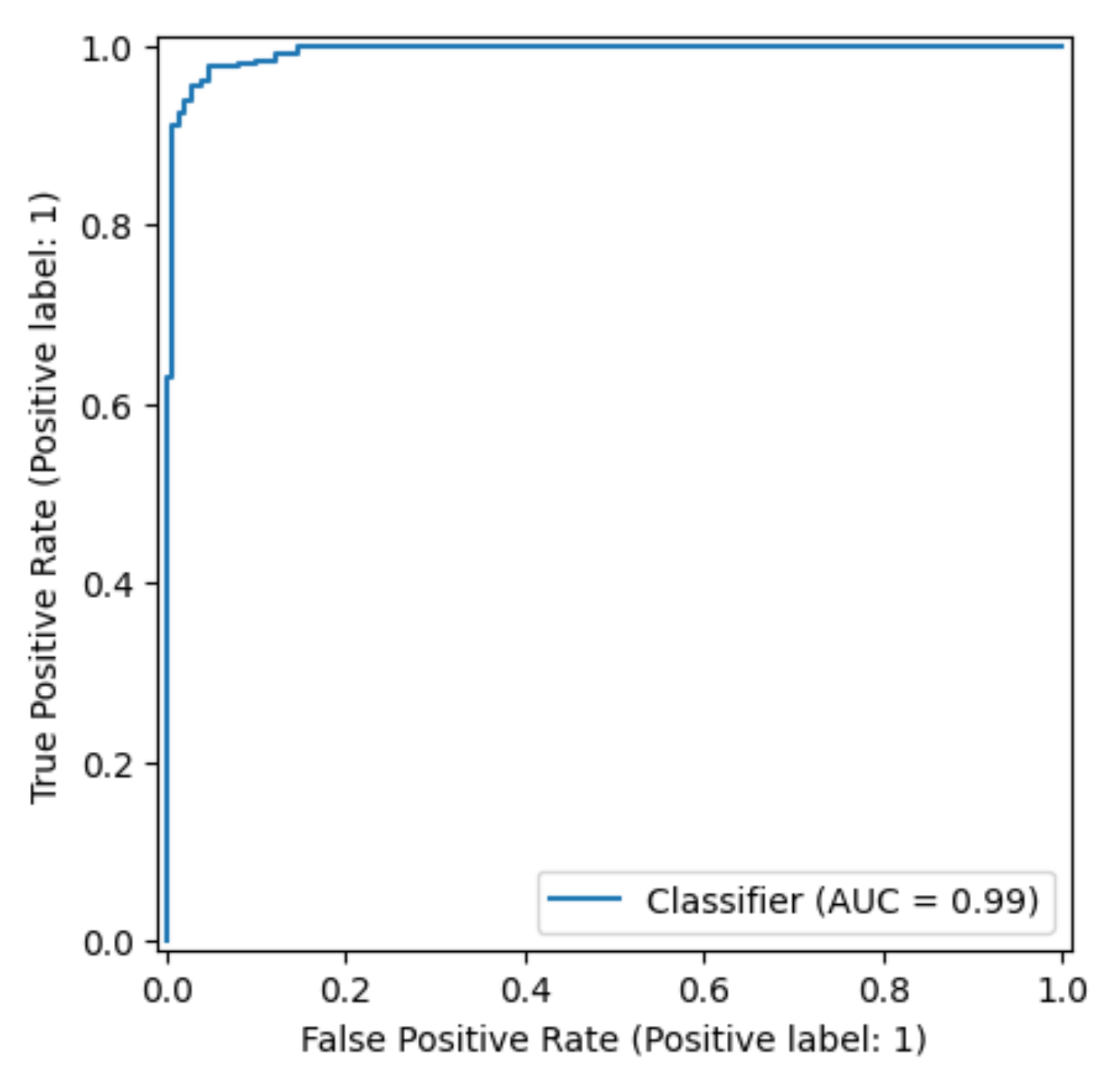
1. Tracer la courbe ROC.
2. Comparer les performances sur l'ensemble d'entraînement et de test.
3. Utiliser une validation croisée (5-fold cross-validation).

In [4]:
```
import numpy as np
from sklearn.metrics import roc_curve, RocCurveDisplay
fpr, tpr, thresholds = roc_curve(y_test, y_prob, pos_label=2)

import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
RocCurveDisplay.from_predictions(y_test, y_prob)
plt.show()
```

```
/Users/ouaa2003/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_ranking.py:1183: UndefinedMetricWarning: No po
sitive samples in y_true, true positive value should be meaningless
  warnings.warn(
```



In [5]:
```
y_train_prob = model.predict_proba(X_train)[:,1]
RocCurveDisplay.from_predictions(y_train, y_train_prob)
plt.show()
```



In [6]:
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5, scoring='roc_auc')
print('ROC-AUC scores:', scores)
print('ROC-AUC moyenne:', scores.mean())
```

```
ROC-AUC scores: [0.99377661 0.99344907 0.99801587 0.97949735 0.99765258]
ROC-AUC moyenne: 0.9924782978664407
```

Observe-t-on un surapprentissage (overfitting) ?

Les performances varient-elles fortement selon les folds ?

Pas de surapprentissage car on n a pas train >> test

Les scores sont légèrement variables selon folds.

3) Amélioration du modèle

3.1) Normalisation des variables

Utiliser `StandardScaler` dans un `Pipeline`. Réentraîner le modèle. Comparer les performances.

3.2) Optimisation des hyperparamètres

Utiliser `GridSearchCV` pour ajuster le paramètre `C` de la régression logistique. Tester plusieurs valeurs (ex : 0.01, 0.1, 1, 10, 100). Identifier la meilleure configuration.

In [7]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', LogisticRegression(max_iter=5000))
])

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98        63
           1       0.99      0.98      0.99       108

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

In [ ]:

In [8]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {'model__C': [0.01, 0.1, 1, 10, 100]}

grid = GridSearchCV(pipeline, param_grid, cv=5, scoring='roc_auc')
grid.fit(X_train, y_train)

print('Meilleur C:', grid.best_params_)
print('Meilleur score:', grid.best_score_)
```

```
Meilleur C: {'model__C': 1}
Meilleur score: 0.9928839784189538
```

La normalisation améliore la convergence.

L'optimisation du paramètre C ajuste la régularisation.

3.3) Comparaison avec d'autres modèles

RandomForestClassifier (ou GradientBoostingClassifier, SVC (avec `probability=True`))

Comparer les modèles à l'aide de la ROC-AUC et du F1-score.

Quel modèle généralise le mieux ?

Lequel semble surapprendre ?

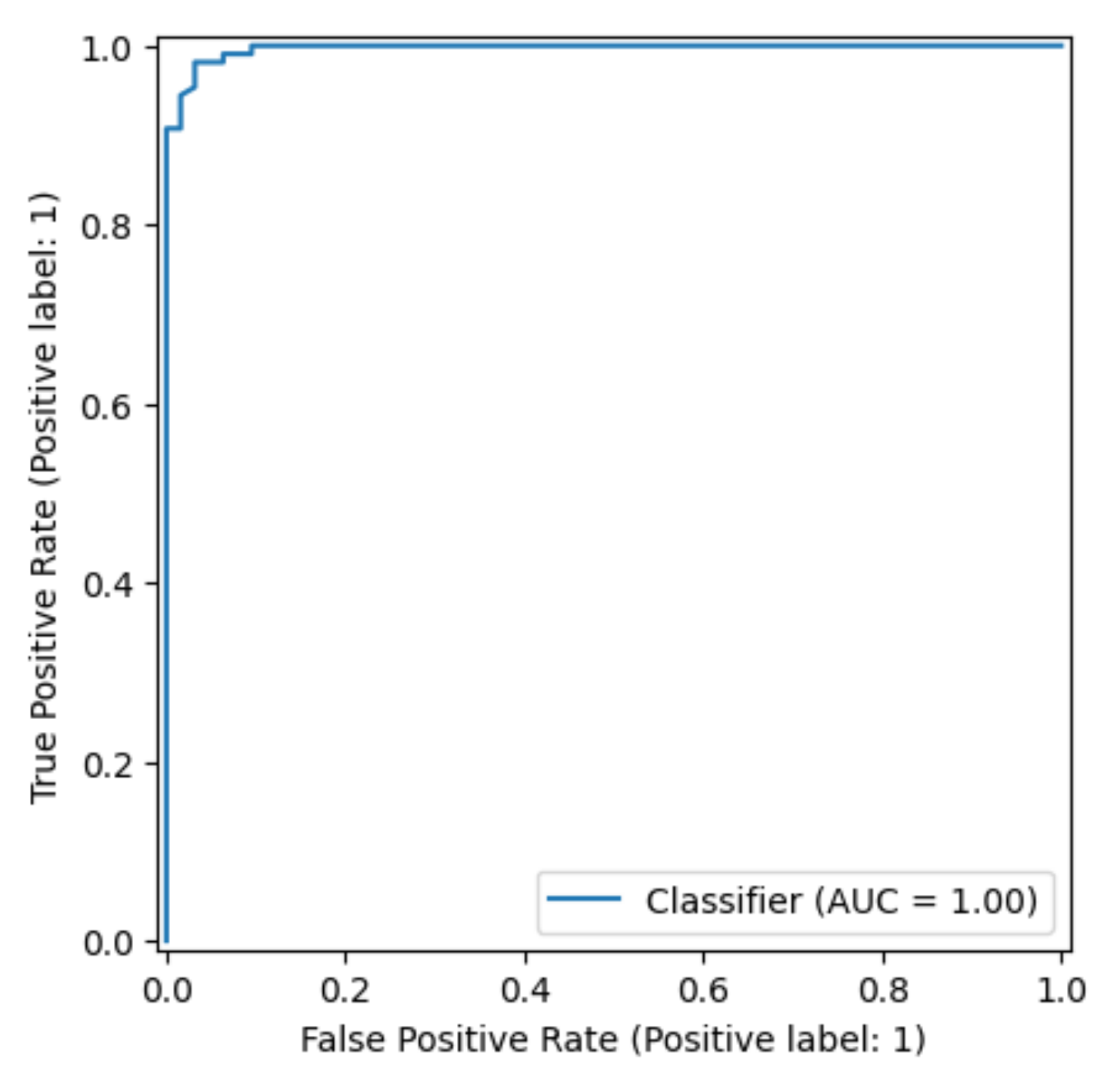
In [9]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

y_prob_rf = rf.predict_proba(X_test)[:,1]
print('ROC-AUC RF:', roc_auc_score(y_test, y_prob_rf))
```

```
ROC-AUC RF: 0.9968400940623163
```

Les modèles arbres capturent des non-linéarités.

Cependant, ils ont tendance à surapprendre.

In [10]:
```
fpr, tpr, _ = roc_curve(y_test, y_prob_rf)

RocCurveDisplay.from_predictions(y_test, y_prob_rf)
plt.show()
```

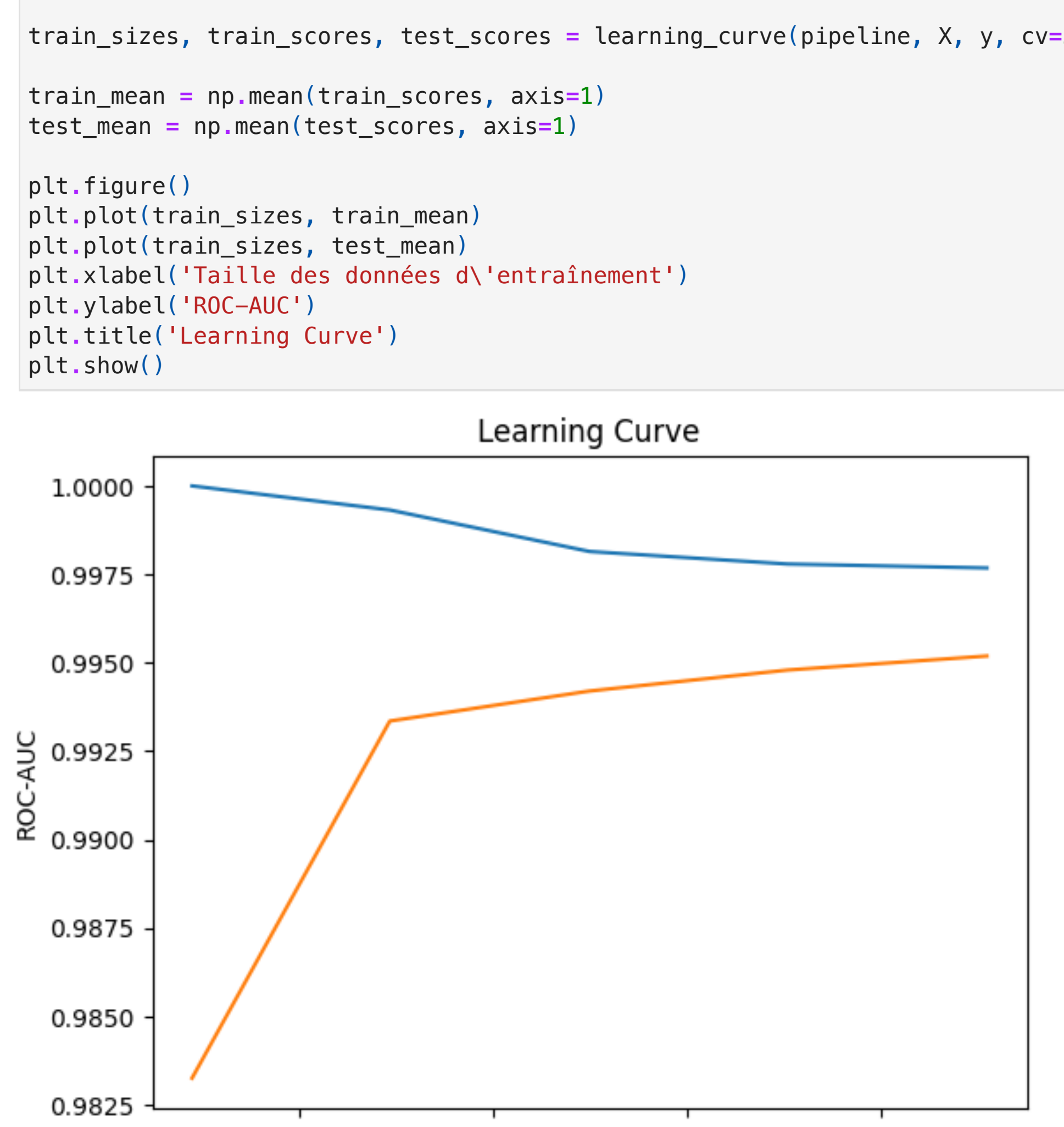


4) Analyse avancée

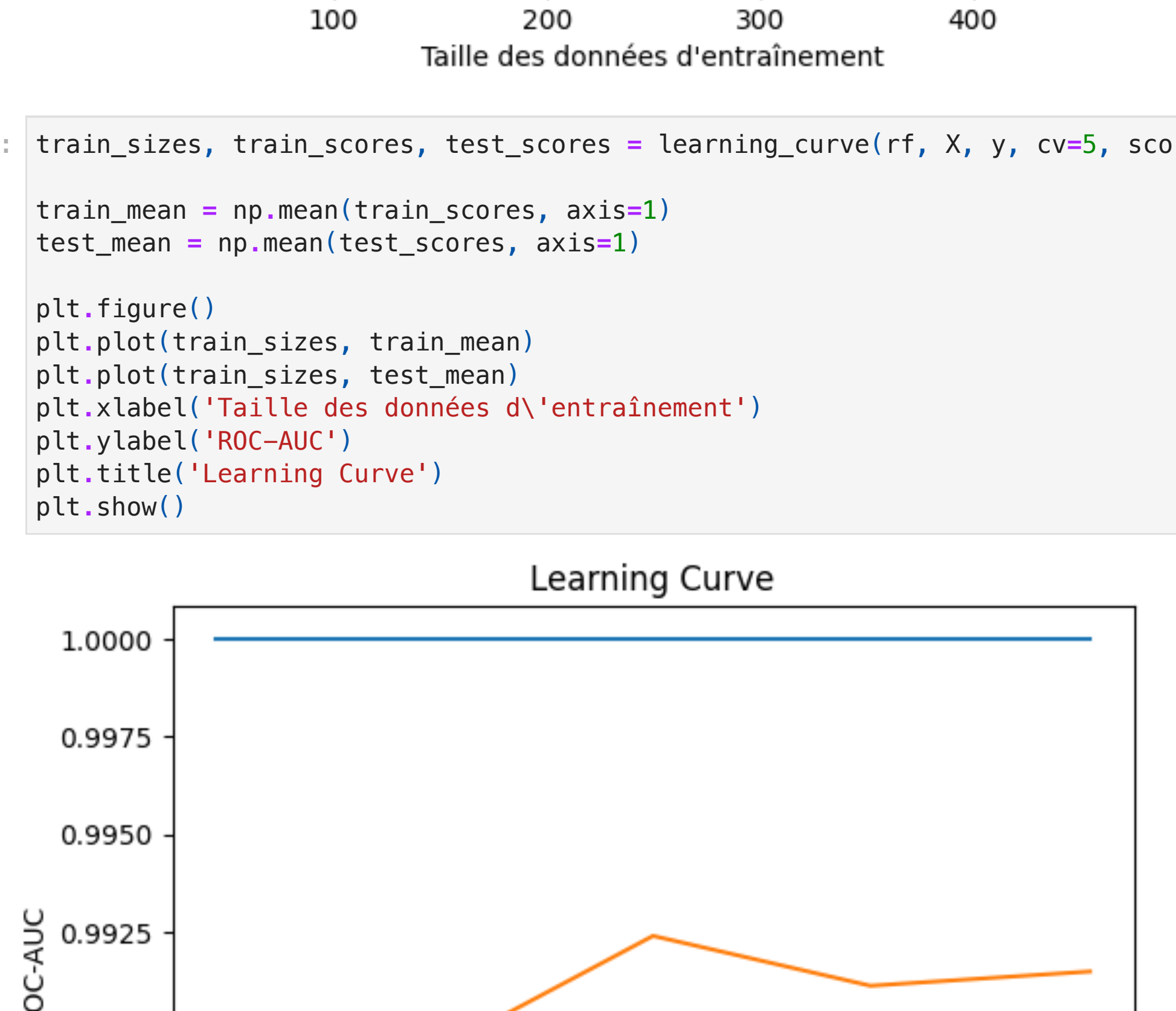
1. Tracer des learning curves.
2. Examiner l'importance des variables pour les modèles basés sur les arbres.
3. Appliquer une réduction de dimension (PCA) en conservant 95% de la variance.
4. Comparer les performances avant et après PCA.

- Le modèle souffre-t-il de biais élevé ou de variance élevée ?
- La réduction de dimension améliore-t-elle les performances ? Pourquoi ?

In [11]:
```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(pipeline, X, y, cv=5, scoring='roc_auc')

train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)

plt.figure()
plt.plot(train_sizes, train_mean)
plt.plot(train_sizes, test_mean)
plt.xlabel('Taille des données d\'entraînement')
plt.ylabel('ROC-AUC')
plt.title('Learning Curve')
plt.show()
```



In [12]:
```
train_sizes, train_scores, test_scores = learning_curve(rf, X, y, cv=5, scoring='roc_auc')

train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)

plt.figure()
plt.plot(train_sizes, train_mean)
plt.plot(train_sizes, test_mean)
plt.xlabel('Taille des données d\'entraînement')
plt.ylabel('ROC-AUC')
plt.title('Learning Curve')
plt.show()
```



On a un écart important entre train et test donc la variance est élevée. Le biais n'est pas élevé car les scores sont forts des deux côtés. La convergence dans le cas du pipeline indique un bon compromis.

In [13]:
```
from sklearn.decomposition import PCA

pca_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95)),
    ('model', LogisticRegression(max_iter=5000))
])

pca_pipeline.fit(X_train, y_train)
y_pred_pca = pca_pipeline.predict(X_test)
print(classification_report(y_test, y_pred_pca))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        63
           1       0.99      1.00      1.00       108

    accuracy                           0.99       171
   macro avg       1.00      0.99      0.99       171
weighted avg       0.99      0.99      0.99       171
```

L'optimisation des hyperparamètres a le plus d'impact.

La validation croisée est plus robuste.

In [ ]: