École Nationale Supérieure Polytechnique,
Yaoundé, Cameroun

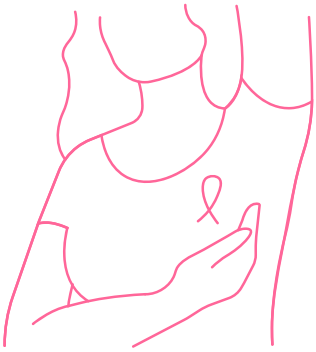
RAPPORT

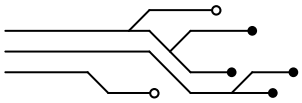
MACHINE LEARNING

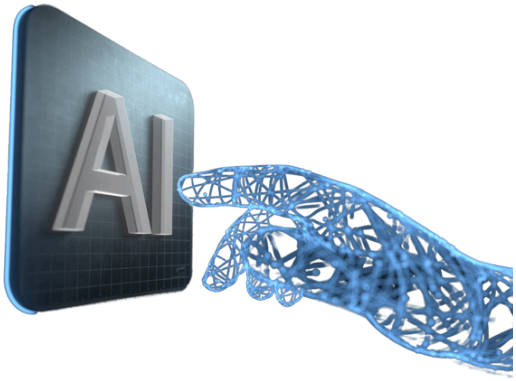
SmartDiagnosis :

Une approche dynamique de Machine Learning pour un diagnostic médical précis

APPLICATION AU DIAGNOSTIC DU CANCER DU SEIN







MEMBRES DU GROUPE

- BENGONO NATHAN, 21P091
- DONCHI TRESOR, 21P107
- FOMEKONG JONATHAN, 21P021
- KENFACK FRANCK, 21P335
- NOUKOUA MAEVA, 21P284
- TCHASSI DANIEL, 21P073

Niveau4, GI

Sous la supervision de:
Dr. Louis Fippo Fitime

Année Scolaire: 2024-2025

TABLE DES MATIÈRES

1. Introduction

Contexte et motivation

Le cancer du sein est l'un des cancers les plus répandus chez les femmes à travers le monde, représentant une problématique majeure de santé publique. Le diagnostic précoce et précis de cette maladie joue un rôle crucial dans l'amélioration des chances de survie et la réduction des traitements lourds pour les patientes. Cependant, les méthodes diagnostiques traditionnelles, souvent dépendantes de l'interprétation humaine des images radiologiques, peuvent être limitées par des biais subjectifs ou des contraintes de temps. Par conséquent, il est nécessaire de développer des outils automatisés et fiables qui assistent les cliniciens dans leur prise de décision, en offrant des résultats rapides et précis.

Objectif et problématique du projet

L'objectif principal de ce projet est de concevoir un modèle de machine learning capable de diagnostiquer le cancer du sein à partir d'une image radiologique de la partie concernée ou d'une image de biopsy microscopique. Nous procédons d'abord au calcul d'un ensemble de variables. Ce modèle doit être capable de distinguer les cas malins des cas bénins avec une grande précision. La problématique repose sur la nécessité d'une méthode de diagnostic qui combine efficacité et simplicité, tout en restant indépendante des algorithmes complexes de deep learning, lesquels nécessitent souvent des ressources informatiques importantes et des volumes de données massifs.

L'approche par machine learning s'avère particulièrement pertinente dans ce contexte, car elle permet d'exploiter des variables soigneusement extraites des images radiologiques (feature engineering) pour construire un modèle de classification performant. Cette méthode est non seulement plus accessible sur le plan computationnel, mais elle favorise également une meilleure interprétabilité des résultats.

Formalisation de la tâche

Pour répondre à cette problématique, nous avons suivi les étapes suivantes :

1. **Extraction des variables** : À partir des images radiologiques, nous avons effectué un travail approfondi de feature engineering pour extraire des caractéristiques pertinentes permettant de décrire les propriétés des masses tumorales.

2. **Construction d'un dataset** : Les variables extraites, accompagnées de leurs labels (maligne ou bénigne), ont été utilisées pour constituer un nouveau jeu de données adapté à l'apprentissage machine.
3. **Entraînement et évaluation des modèles** : Plusieurs algorithmes de classification supervisée ont été appliqués, et nous avons retenu celui offrant les meilleures performances en termes de précision, rappel et F1-score.

2. Exploration des données

2.1 Recherche et sélection des datasets

Afin d'atteindre notre objectif, nous avons fais des recherches et trouvé plusieurs Dataset parmi lesquels:

- Breast Cancer Wisconsin (Diagnostic) : Ce dataset, connu sous le nom de "Breast Cancer Wisconsin (Diagnostic)", contient des caractéristiques calculées à partir d'images numérisées d'aspirations à l'aiguille fine (FNA) de masses mammaires. Les caractéristiques décrivent les propriétés des noyaux cellulaires présents dans ces images.
- CBIS-DDSM (Curated Breast Imaging Subset of Digital Database for Screening Mammography) est une version améliorée et standardisée de la base de données DDSM originale. Ce dataset contient 2620 études de mammographies numérisées, comprenant des cas normaux, bénins et malins, tous accompagnés d'informations pathologiques vérifiées. Il a été soigneusement sélectionné et organisé par des mammographes expérimentés, offrant ainsi une ressource précieuse pour le développement et l'évaluation de systèmes d'aide à la décision en mammographie. Le CBIS-DDSM inclut des images décompressées et converties au format DICOM, des segmentations de régions d'intérêt (ROI) mises à jour, des boîtes englobantes, ainsi que des diagnostics pathologiques pour les données d'entraînement. Ce dataset comprend 753 cas de calcifications et 891 cas de masses, fournissant un volume de données suffisant pour analyser les systèmes d'aide à la décision en mammographie. Les images sont accompagnées d'annotations au niveau des pixels pour la segmentation d'instances, ce qui permet également leur utilisation pour des tâches de segmentation sémantique ou de détection d'objets.

2.1.1. Dataset candidat 1: Breast Cancer Wisconsin (Diagnostic)

- **Nombre d'échantillons** : **569**
    - Ce dataset contient 569 cas de tumeurs mammaires analysées.
    - Chaque échantillon correspond à une biopsie analysée sous forme numérique.
- **Nombre de caractéristiques (features)** : **32**
    - **1 colonne ID** : Un identifiant unique pour chaque observation.
    - **1 colonne cible (Diagnosis)** :
        - C'est une donnée **catégorique** indiquant si la tumeur est **maligne** ("M") ou **bénigne** ("B").

    - **30 colonnes numériques** : Elles décrivent les propriétés physiques et texturales des cellules mammaires.
- **Type de données** :
    - Les **30 caractéristiques** sont toutes **numériques continues**.
    - La **colonne cible (Diagnosis)** est **catégorique** (M ou B).

- **Difficultés potentielles liées à l'utilisation :**
    1. **Déséquilibre potentiel des classes** :
        - Si une des classes (M ou B) est sur-représentée, cela peut biaiser les résultats des modèles de classification.
        - Distribution typique :
            - Classe "Benign" (B) : ~62% des échantillons.
            - Classe "Malignant" (M) : ~38% des échantillons.
    2. **Corrélations fortes** :
        - Certaines caractéristiques peuvent être fortement corrélées, ce qui pourrait poser problème pour certains modèles (comme la régression logistique).
    3. **Prétraitement nécessaire** :
        - Les colonnes doivent être normalisées, car certaines caractéristiques (ex : **Area**, **Radius**) ont des valeurs bien plus élevées que d'autres (ex : **Smoothness**).
    4. **Absence d'images brutes** :
        - Les caractéristiques ont été dérivées d'images de biopsie, mais les images elles-mêmes ne sont pas fournies. Cela limite l'application directe de méthodes basées sur le traitement d'image.

Analyse exploratoire des données (EDA) :

- **Distribution de la classe cible** :
    - **Bénigne (B)** : 357 échantillons (62,7%).
    - **Maligne (M)** : 212 échantillons (37,3%).
- **Corrélation** :
    - Les données montrent une distinction nette entre les deux classes sur certaines caractéristiques comme le ***radius_mean***, ***area_mean***, et ***compactness_mean***.
    - Les données nécessitent une normalisation avant d'être utilisées dans des modèles.
    - Une réduction de dimension peut être utile pour limiter l'impact des corrélations élevées.

2.1.2. Dataset candidat 2: CBIS-DDSM

Nombre d'échantillons et caractéristiques :

- 3103 images au total
- 3577 objets étiquetés appartenant à une seule classe (***abnormal_structure***)
- 753 cas de calcifications et 891 cas de masses

Type de données :

- Images de mammographies numérisées
- Annotations au niveau des pixels pour la segmentation d'instances
- Images en format DICOM, 16 bits pour les images complètes et 8 bits pour les ROI
- Politique de traitement des images
- Décompression et conversion des images au format DICOM
- Mise à jour des segmentations ROI et des boîtes englobantes
- Conversion possible en PNG sans altération de la qualité des bits
- Option de suppression de l'écrêtage des valeurs de densité optique

Difficultés potentielles liées à l'utilisation

- Taille importante des images nécessitant des ressources computationnelles conséquentes
- Risque de perte d'informations importantes lors du recadrage à des tailles plus petites (224 x 224 ou 299 x 299 pixels)
- Complexité du traitement des différentes profondeurs de bits (16 bits pour les images complètes, 8 bits pour les ROI)
- Nécessité d'une gestion appropriée du déséquilibre des classes (cas normaux, bénins et malins)

EDA du dataset

- Les cas malins présentent une proportion plus importante par zone de 598x598 pixels que les cas bénins
- Quartiles pour les cas bénins : 9,2%, 14,5%, 22,2%
- Quartiles pour les cas malins : 11,6%, 18,6%, 31,2%
- Différence statistiquement significative entre les cas bénins et malins ($P < 0,001$, test U de Mann-Whitney)

2.2 Analyse étape par étape transformation image en données numérique

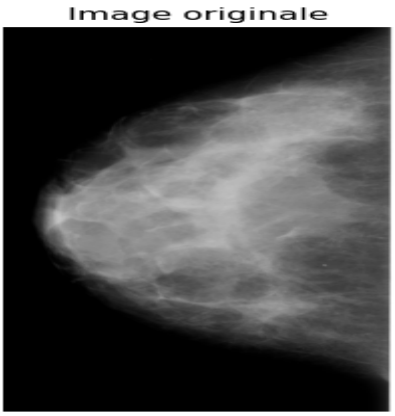
2.2.1. Chargement de l'image

Pour le faire, nous utilisons la fonction python ***imread*** via:

```
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

Explication

- Cette ligne charge une image en niveaux de gris à partir du chemin spécifié (`image_path`).
- L'image est représentée comme une matrice 2D où chaque pixel est un entier entre 0 (noir) et 255 (blanc).



2.2.2. Prétraitement de l'image

Code

```
blurred = cv2.GaussianBlur(image, (5, 5), 0)
_, binary = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
binary = cv2.bitwise_not(binary)
```

Explication

1. **Floutage (Gaussian Blur)** :
    - Utilise un filtre gaussien pour lisser l'image et réduire le bruit.
    - Cela facilite la segmentation des objets en minimisant les variations de pixels.
2. **Seuillage adaptatif (Otsu)** :

    - Convertit l'image floutée en une image binaire (noir et blanc).

- Les pixels sont soit 0 (noir) ou 255 (blanc), selon une valeur seuil calculée automatiquement (Otsu).

3. **Inversion binaire** :
    - Inverse les couleurs de l'image binaire pour que les objets d'intérêt deviennent blancs (255) et l'arrière-plan noir (0).

Résultat attendu

- Une image binaire où les objets (ex. noyaux cellulaires) sont en blanc sur un fond noir.
- Exemple visuel :
    - image 1 :

    - image 2 :

2.2.3. Détection des contours

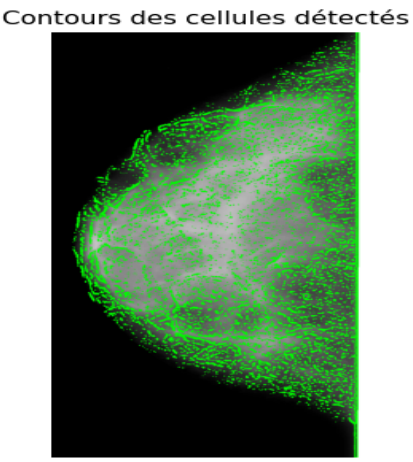
Code

```
contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

Explication

- Trouve les contours des objets blancs dans l'image binaire.
- Chaque contour est une liste de points représentant les bords de l'objet.

Résultat



2.2.4. Calcul des caractéristiques géométriques

Explication

1. **Aire (Area)**
   - Calcule la surface de l'objet (en pixels carrés).
2. **Périmètre (Perimeter)** :
   - Longueur totale du contour de l'objet.
3. **Compacité (Compactness)** :
   - Mesure à quel point la forme est compacte
4. **Rayon (Radius)** :
   - Rayon équivalent d'un cercle avec la même aire .
5. **Centroïde (Centroid)** :
   - Le centre de gravité de l'objet, calculé à partir des moments géométriques.

Résultat

```
{
  "Area": 300.5,
  "Perimeter": 75.4,
  "Compactness": 1.2,
  "Radius": 9.8,
  "Centroid": (25, 30)
}
```

2.2.5. Calcul de la symétrie

Explication

- Calcule la différence entre la largeur (ww) et la hauteur (hh) de la boîte englobante de l'objet.
- La symétrie est proche de 0 pour les formes presque circulaires.

2.2.6. Extraction et normalisation de la région d'intérêt (ROI)

Explication

- Extrait la région correspondant à l'objet (ROI) à partir de l'image originale.
- Normalise les valeurs de pixels pour qu'elles soient entre 0 et 255.

NB: La région d'intérêt (ROI, pour **Region of Interest**) représente une zone spécifique de l'image contenant un objet ou une partie qui nous intéresse pour l'analyse. Dans le contexte de votre fonction, la ROI correspond à une **zone rectangulaire entourant un noyau cellulaire** détecté via les contours.

2.2.7. Extraction des caractéristiques texturales (Haralick)

Explication

- Calcule les caractéristiques Haralick basées sur la matrice de cooccurrence des niveaux de gris (GLCM).
- Mesures texturales :
    - **Contraste** : Différences entre des pixels adjacents.
    - **Dissimilarité** : Variation entre pixels voisins.
    - **Homogénéité** : Uniformité de la texture.
    - **Énergie** : Régularité dans la texture.
    - **Corrélation** : Relation entre les pixels voisins.

Résultat


```
{
  "Contrast": 1.23,
  "Dissimilarity": 0.56,
  "Homogeneity": 0.89,
  "Energy": 0.75,
  "Correlation": 0.92
}
```


2.2.8. Résumé et statistiques globales

Explication

1. Regroupe toutes les caractéristiques calculées pour chaque noyau dans un DataFrame.
2. Calcule :
    - La moyenne (mean\text{mean}).

- L'erreur standard (SE\text{SE}).
- La pire valeur (worst\text{worst}).

Code final

```python
import cv2
import numpy as np
import pandas as pd
import mahotas
from skimage.measure import regionprops, label

def extract_features(image_path):
    # Charger l'image en niveaux de gris
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Étape 1 : Prétraitement de l'image
    blurred = cv2.GaussianBlur(image, (5, 5), 0)
    _, binary = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    binary = cv2.bitwise_not(binary)

    # Étape 2 : Détection des contours
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Liste pour stocker les caractéristiques pour chaque noyau
    nuclei_features = []

    for contour in contours:
        if cv2.contourArea(contour) < 50:  # Ignorer les petits artefacts
            continue

        # Calcul des caractéristiques géométriques
        area = cv2.contourArea(contour)
        perimeter = cv2.arcLength(contour, True)
        compactness = (perimeter**2) / (4 * np.pi * area) if area > 0 else 0
        radius = np.sqrt(area / np.pi) if area > 0 else 0
        M = cv2.moments(contour)
        centroid = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"])) if M["m00"] != 0 else (0, 0)

        # Calcul de la symétrie
        bounding_box = cv2.boundingRect(contour)
        x, y, w, h = bounding_box
        symmetry = abs(w - h) / max(w, h)

        # Extraction de la ROI
        roi = image[y:y+h, x:x+w]
        roi_norm = (roi - np.min(roi)) / (np.max(roi) - np.min(roi) + 1e-6) * 255
        roi_norm = roi_norm.astype(np.uint8)

        # Caractéristiques texturales (Haralick avec mahotas)
        glcm = mahotas.features.texture.haralick(roi_norm)
        contrast = glcm[:, 0].mean()  # Haralick contrast
        dissimilarity = glcm[:, 1].mean()  # Haralick dissimilarity
        homogeneity = glcm[:, 2].mean()  # Haralick homogeneity
        energy = glcm[:, 4].mean()  # Haralick energy
        correlation = glcm[:, 5].mean()  # Haralick correlation

        # Dimension fractale (approximation du contour)
        hull = cv2.convexHull(contour)
        fractal_dimension = cv2.arcLength(hull, True) / perimeter if perimeter > 0 else 0

        # Stockage des caractéristiques
        nuclei_features.append({
            "Area": area,
            "Perimeter": perimeter,
            "Compactness": compactness,
            "Radius": radius,
            "Symmetry": symmetry,
            "Fractal Dimension": fractal_dimension,
            "Contrast": contrast,
            "Dissimilarity": dissimilarity,
            "Homogeneity": homogeneity,
            "Energy": energy,
            "Correlation": correlation
        })

    # Calcul des statistiques globales
    df_features = pd.DataFrame(nuclei_features)
    summary_features = {}

    for col in df_features.columns:
        summary_features[f"{col}_mean"] = df_features[col].mean()
        summary_features[f"{col}_se"] = df_features[col].std()
        summary_features[f"{col}_worst"] = df_features[col].max()

    return pd.DataFrame([summary_features])

# Exemple d'utilisation
image_path = "chemin/vers/votre/image.jpeg"
df = extract_features(image_path)
print(df)
```

Exemple final des données extraites

```
image_path = "chemin/image/Image1.jpeg"
df = extract_features(image_path)
print(df)
```

Output

```
  Area_mean      Area_se  Area_worst  Perimeter_mean  Perimeter_se  \
0   89131.2  198813.782111    444780.0     1184.069114   2476.416665

  Perimeter_worst  Compactness_mean  Compactness_se  Compactness_worst  \
0     5613.610506          3.005285        1.568695           5.638057

  Radius_mean  ...  Dissimilarity_worst  Homogeneity_mean  Homogeneity_se  \
0   81.496709  ...          1964.036839          0.789466        0.146901

  Homogeneity_worst  Energy_mean  Energy_se  Energy_worst  Correlation_mean  \
0           0.99929      0.30028   0.184557      0.626486         198.26123

  Correlation_se  Correlation_worst
0      41.999102         254.870915
```

3. Analyse Exploratoire des données du dataset choisi

La préparation des données est une étape clé dans le cadre de notre projet sur le diagnostic du cancer du sein. Elle permet de garantir que les données utilisées pour l'analyse sont fiables, cohérentes et adaptées aux objectifs d'apprentissage automatique. Cette phase comprend plusieurs opérations : le nettoyage des données, la gestion des valeurs manquantes ou aberrantes, la transformation des colonnes pour une meilleure interprétabilité, et la standardisation des caractéristiques numériques. Ces étapes sont détaillées ci-dessous, avec un focus particulier sur les spécificités des données médicales.

3.1. Importation des bibliothèques

Pour commencer, nous importons les bibliothèques nécessaires pour manipuler les données et effectuer des analyses exploratoires.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

- **Pandas** : Cette bibliothèque est essentielle pour manipuler les données tabulaires, notamment sous forme de DataFrame. Elle permet de charger, nettoyer et transformer les données efficacement.
- **NumPy** : Fournit des outils puissants pour le calcul numérique, notamment pour les opérations sur des tableaux multidimensionnels.
- **Seaborn** : Facilite la création de visualisations statistiques pour comprendre les relations entre les variables.
- **Matplotlib** : Complète Seaborn en permettant une personnalisation fine des graphiques.

3.2 Chargement et aperçu des données

Nous chargeons les données à partir d'un fichier CSV contenant les caractéristiques extraites de tissus mammaires ainsi que le diagnostic correspondant.

```
df = pd.read_csv("data.csv")
print("Aperçu des données :")
print(df.head())
print("Dimensions du dataset :", df.shape)
print("Noms des colonnes :")
print(df.columns)
```

- **Chargement des données** : La méthode `pd.read_csv()` permet d'importer les données brutes sous forme de DataFrame.
- **Aperçu rapide** : La fonction `head()` affiche les cinq premières lignes pour examiner la structure des données.
- **Dimensions du jeu de données** : La méthode `shape` informe du nombre de lignes et de colonnes, ce qui est crucial pour comprendre l'ampleur des données disponibles.
- **Noms des colonnes** : La méthode `columns` identifie les différentes variables, incluant les caractéristiques des tissus et les colonnes superflues éventuelles.

Résultat attendu :

Aperçu des données :

***id diagnosis radius_mean texture_mean perimeter_mean area_mean ***

0 842302 M 17.99 10.38 122.80 1001.0

***smoothness_mean compactness_mean concavity_mean concave points_mean ***

0 0.11840 0.27760 0.3001 0.14710

*... texture_worst perimeter_worst area_worst smoothness_worst *

0 ... 17.33 184.60 2019.0 0.1622

*compactness_worst concavity_worst concave points_worst symmetry_worst *

0 0.6656 0.7119 0.2654 0.4601

...

diagnosis

B 357

M 212

Name: count, dtype: int64

3.3 Suppression des colonnes inutiles

Certaines colonnes, comme un identifiant unique ou des valeurs nulles, n'apportent aucune valeur analytique.

```
df = df.drop(columns=["id", "Unnamed: 32"], errors="ignore")
```

Colonnes supprimées :

- `id` : Uniquement utilisé comme identifiant, n'a pas de lien avec le diagnostic.
- `Unnamed: 32` : Une colonne vide générée par erreur lors de l'export des données.

Paramètre `errors="ignore"` : Évite les erreurs si une colonne spécifiée est absente du fichier.

Résultat :

- Un DataFrame nettoyé sans ces colonnes.

3.4 Vérification des données manquantes et doublons

Nous vérifions ensuite si des données sont manquantes ou dupliquées, ce qui pourrait biaiser les analyses.

```
print("\nValeurs manquantes par colonne :")
print(df.isnull().sum())
print("\nNombre de doublons :", df.duplicated().sum())
```

- **Données manquantes** : La méthode `isnull().sum()` permet d'identifier les colonnes contenant des valeurs nulles. Dans un contexte médical, ces lacunes doivent être traitées avec soin pour éviter des conclusions erronées.
- **Doublons** : Les doublons, identifiés via `duplicated().sum()`, peuvent provenir d'erreurs de saisie ou de traitement. Ces enregistrements sont généralement supprimés.

Résultat attendu :

- Zéro valeur manquante.
- Zéro doublon.

3.5 Types de données et résumé statistique

Pour mieux comprendre la structure et les valeurs des données, nous examinons leurs types et leur résumé statistique.

```
print("\nTypes de données :")
print(df.info())

print("\nRésumé statistique :")
print(df.describe())
```

- **Types de données** : La méthode `info()` répertorie les types de données pour chaque colonne, comme `float64` ou `object`, ce qui aide à anticiper d'éventuelles conversions.
- **Résumé statistique** : La méthode `describe()` fournit des informations sur la moyenne, les écarts-types, et les percentiles des colonnes numériques. Ces valeurs sont particulièrement importantes pour détecter les anomalies, comme des mesures physiologiques non plausibles.

Résultat: Aperçu des types et une description statistique claire.

3.6 Distribution de la colonne cible

La colonne `diagnosis`, qui contient les étiquettes des échantillons, est au cœur de notre projet.

```
print("\nDistribution de la colonne cible (diagnosis) :")
print(df['diagnosis'].value_counts())
```

Valeurs possibles : La colonne `diagnosis` comporte deux catégories :

- `M` (Malignant) : Indique un tissu tumoral malin.
- `B` (Benign) : Indique un tissu tumoral bénin.

Répartition : La méthode `value_counts()` quantifie chaque catégorie. Cette information est cruciale pour évaluer un éventuel déséquilibre des classes, qui peut influencer les modèles de machine learning.

Résultat attendu :

Distribution de la colonne cible (diagnosis) :

diagnosis

0 357

1 212

Name: count, dtype: int64

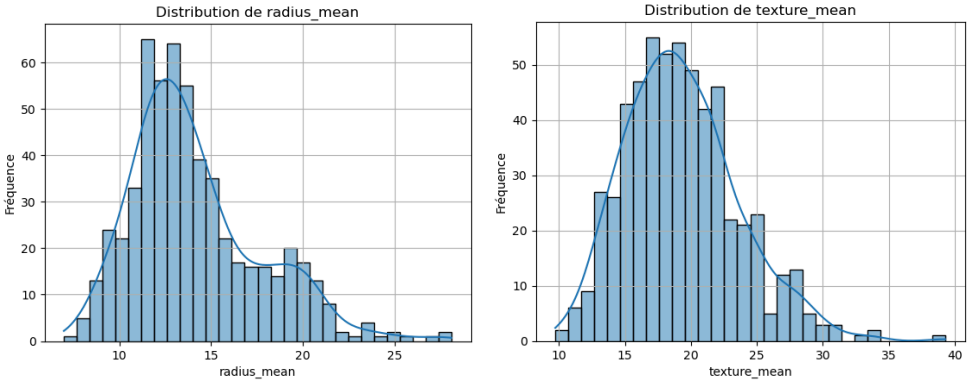
3.7 Visualisation des distributions

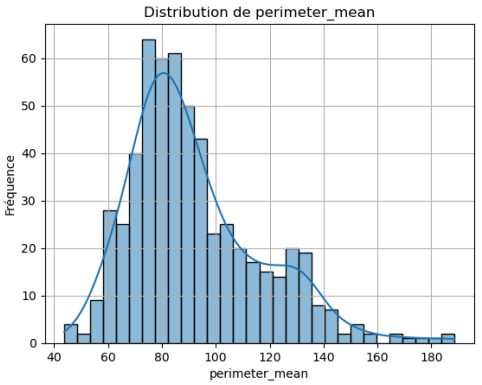
Pour les principales caractéristiques, nous visualisons les distributions :

```
columns_to_plot = ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean']
for column in columns_to_plot:
    plt.figure()
    sns.histplot(df[column], kde=True, bins=30)
    plt.title(f"Distribution de {column}")
    plt.xlabel(column)
    plt.ylabel("Fréquence")
    plt.grid()
    plt.show()
```

- Crée des histogrammes pour chaque colonne spécifiée.
- **`kde=True`** : Ajoute une courbe de densité.
- **`bins=30`** : Définit 30 intervalles pour l'histogramme.

Résultat attendu : Histogrammes des caractéristiques avec courbes de densité.





....

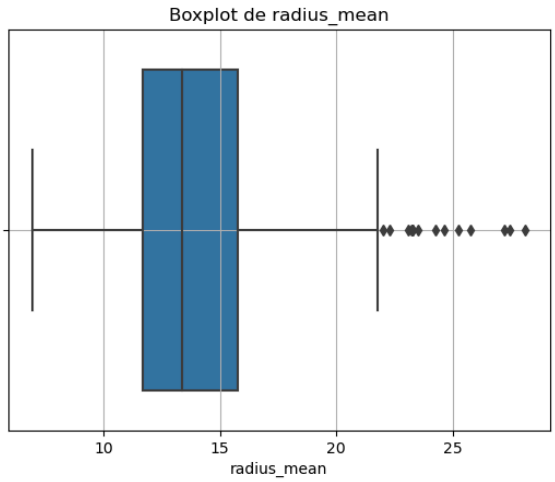
3.8 Visualisation des outliers

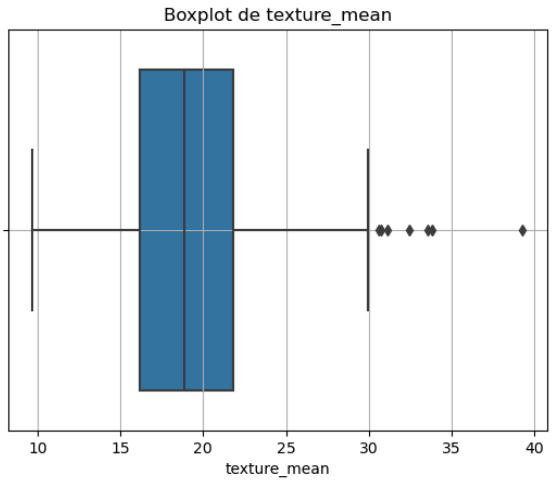
Les valeurs aberrantes ou "outliers" peuvent affecter significativement les performances des modèles de machine learning. Dans cette étape, nous utilisons des boxplots pour visualiser ces valeurs extrêmes. Ces graphiques permettent d'identifier les points de données situés en dehors des whiskers (limites définies par l'écart interquartile). Cela aide à décider si des actions correctives, comme la normalisation ou la suppression, sont nécessaires.

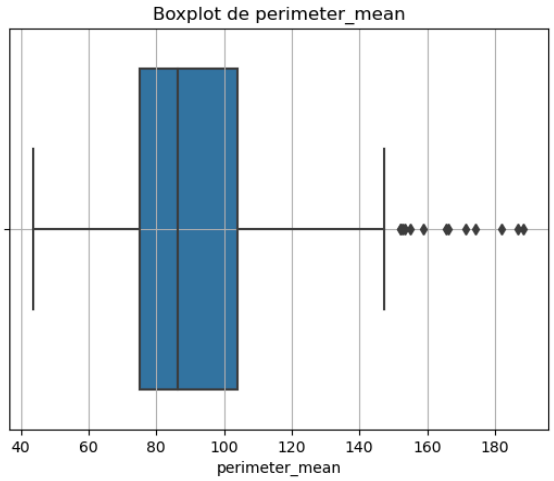
```
for column in columns_to_plot:
    plt.figure()
    sns.boxplot(x=df[column])
    plt.title(f"Boxplot de {column}")
    plt.grid()
    plt.show()
```

- **Boxplots** : Affichent la médiane, les quartiles et les valeurs aberrantes.
- Les graphiques générés mettent en évidence les colonnes numériques sélectionnées dans `columns_to_plot`.

Résultat attendu : Graphiques Boxplot pour chaque colonne.







3.9 Vérification des valeurs uniques

Il est crucial de vérifier les valeurs uniques de la colonne cible, car cela nous permet de confirmer la présence de toutes les catégories attendues (dans ce cas : bénin et malin). Cela assure aussi qu'il n'y a pas d'erreurs ou de valeurs inattendues.

code

```
print("Valeurs uniques dans la colonne 'diagnosis' :", df['diagnosis'].unique())
```

- **unique()** : Cette méthode liste toutes les valeurs distinctes dans une colonne. Cela permet de s'assurer qu'aucune erreur n'existe dans l'étiquetage.

Résultat attendu: Liste des valeurs distinctes : `['M', 'B']` (malin et bénin).

3.10 Conversion de la colonne cible

Pour les besoins des modèles de machine learning, les données catégoriques doivent être converties en valeurs numériques. Dans cette étape, la colonne `diagnosis` est mappée à des valeurs numériques :

- `M` (malin) → 1
- `B` (bénin) → 0

Cette transformation simplifie les calculs et rend la variable compatible avec les algorithmes d'apprentissage.

```
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})
```

- **map()** : Applique une fonction de correspondance pour convertir les valeurs textuelles en valeurs numériques.

Résultat attendu :

- La colonne `diagnosis` contient maintenant des valeurs numériques (0 ou 1).

3.11 Matrice de corrélation

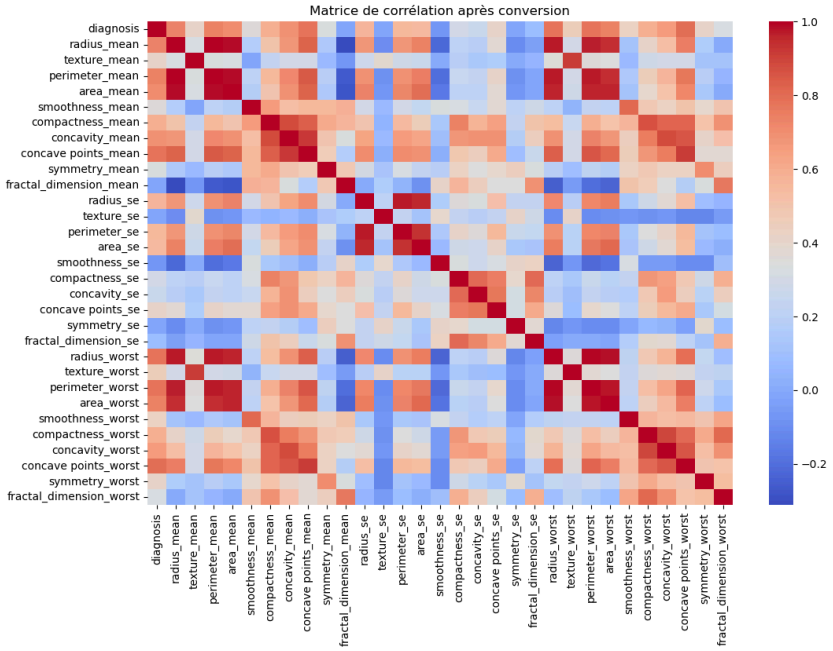
La matrice de corrélation aide à comprendre les relations entre les variables numériques du dataset. Elle met en évidence les colonnes fortement corrélées avec la variable cible (`diagnosis`). Ces informations sont cruciales pour sélectionner les caractéristiques les plus pertinentes pour le modèle.

```
correlation_matrix = df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=False, cmap="coolwarm", cbar=True)
plt.title("Matrice de corrélation après conversion")
plt.show()
```

- **corr()** : Calcule les coefficients de corrélation entre les colonnes numériques.
- **Heatmap** : Représente la matrice de corrélation sous forme de graphique coloré pour faciliter l'interprétation.

Résultat attendu : Une heatmap montrant les corrélations entre toutes les variables. Les coefficients les plus élevés (proches de +1 ou -1) indiquent des relations fortes.



3.12 Colonnes les plus corrélées à la cible

Après avoir calculé la matrice de corrélation, cette étape permet d'identifier les colonnes qui sont les plus corrélées avec la variable cible (`diagnosis`). Ces informations sont utilisées pour la sélection des caractéristiques.

code

```
correlation_with_target = correlation_matrix['diagnosis'].sort_values(ascending=False)
print("\nCorrélation des variables avec la cible (diagnosis) :")
print(correlation_with_target)
```

- **sort_values()** : Trie les colonnes en fonction de leurs coefficients de corrélation avec la cible.

Résultat attendu: Un tableau trié affichant les variables ayant la plus forte corrélation avec `diagnosis`.

3.13 Standardisation des colonnes numériques

Les modèles de machine learning sont sensibles à l'échelle des données. La standardisation des caractéristiques numériques permet de normaliser ces variables pour qu'elles aient une moyenne de 0 et un écart-type de 1. Cela améliore la performance et la convergence des algorithmes.

```
from sklearn.preprocessing import StandardScaler
numerical_features = df.drop(columns=['diagnosis']).columns
scaler = StandardScaler()
```

```
df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

- **StandardScaler** : Applique une transformation pour normaliser les colonnes numériques.
- Les colonnes transformées conservent leurs structures relatives tout en réduisant les biais dus aux échelles différentes.

Résultat attendu: Toutes les colonnes numériques ont une moyenne de 0 et un écart-type de 1.

3.14 Aperçu des données après standardisation

Enfin, nous vérifions que la standardisation a été correctement appliquée en affichant un aperçu des données transformées.

```
print("\nDonnées après standardisation :")
print(df.head())
```

- Cette vérification garantit que toutes les colonnes numériques ont été correctement standardisées.

Résultat attendu :

```
  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         1     1.097064     -2.073335        1.269934   0.984375

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0         1.568466          3.283515        2.652874             2.532475

   symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0       2.217515  ...      1.886690      -1.359293         2.303601

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0    2.001237          1.307686           2.616665         2.109526
```

4. Présentation des modèles et utilisations

4.1 Modèles testés

4.1.1. Régression logistique

Principe :

La régression logistique est un modèle statistique utilisé pour prédire une variable binaire (deux classes) ou multi catégorielle. Contrairement à la régression linéaire, elle utilise une fonction sigmoïde (ou logistique) pour transformer la sortie en une probabilité comprise entre 0 et 1.

Hypothèse :

La régression logistique établit une relation linéaire entre les variables explicatives (X) et le logit (logarithme des probabilités) :

$$logit(p) = \frac{p}{1-p} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$$

où p est la probabilité que l'événement se produise.

Fonction sigmoïde :

Une fois le logit calculé, la fonction sigmoïde est appliquée pour obtenir une probabilité ppp :

$$p = \frac{1}{1+e^{logit(p)}}$$

Décision :

Si p≥0.5, la classe est 1 (positif).

Si p<0.5, la classe est 0 (négatif).

4.1.2. k- Nearest Neighbour

Principe :

Le modèle *k*-**Nearest Neighbours** est un algorithme d'apprentissage supervisé basé sur la similarité des données. Il classe un point en fonction des classes des *k* points les plus proches dans l'espace des caractéristiques.

Étapes principales :

Calcul des distances :

- La distance entre le point à classifier (exemple de test) et tous les points du jeu de données d'entraînement est calculée.
- La distance la plus courante est la distance euclidienne.

Sélection des k plus proches voisins :

- Les k points ayant les plus petites distances par rapport au point de test sont sélectionnés.

Vote majoritaire :

- **Pour une classification** : Le point est assigné à la classe la plus fréquente parmi les k voisins.
- **Pour une régression** : La valeur attribuée est la moyenne ou la médiane des valeurs des k voisins.

Avantages :

- Simple à comprendre et à implémenter.
- Aucune hypothèse forte sur la distribution des données.
- Efficace pour des jeux de données de petite taille et des distributions non linéaires.

Limites :

Sensibilité à la valeur de k : Un k trop petit peut rendre le modèle sensible au bruit. Un k trop grand peut diluer l'influence des vrais voisins proches.

Coût computationnel élevé : Pour des jeux de données volumineux, la recherche des k voisins peut être lente.

Effet des dimensions : Dans des espaces à haute dimension (où les données ont de nombreuses caractéristiques), les distances deviennent moins discriminantes (**malédiction de la dimensionnalité**).

Nécessité d'une normalisation : Les caractéristiques avec des échelles différentes peuvent biaiser les distances. Une standardisation ou une normalisation est donc essentielle.

4.1.3 Decision Trees (Arbres de décision)

Principe :
Un arbre de décision est un modèle d'apprentissage supervisé qui divise les données en sous-ensembles basés sur des tests successifs sur les caractéristiques. Chaque nœud interne représente une décision (un test sur une caractéristique), chaque branche un résultat, et chaque feuille une prédiction.

Étapes principales :

1. Sélection d'une caractéristique pour diviser les données (en fonction de critères comme l'entropie ou le gain d'information).
2. Répétition de la division jusqu'à ce qu'un critère d'arrêt soit atteint (par exemple, un nombre minimal de données par feuille ou une pureté maximale).
3. Classification basée sur le chemin suivi dans l'arbre.

Avantages :

- Interprétable et facile à visualiser.
- Ne nécessite pas de normalisation ou de standardisation des données.
- Gère à la fois les données catégoriques et numériques.

Limites :

- Sensible au surapprentissage (overfitting), surtout pour des arbres très profonds.
- Moins performant sur des données complexes ou bruitées.

4.1.4 Random Forests

Principe :
Les Random Forests sont un ensemble d'arbres de décision où chaque arbre est construit sur un échantillon aléatoire des données et utilise une sélection aléatoire des caractéristiques pour diviser les nœuds. La prédiction finale est basée sur une moyenne (régression) ou un vote majoritaire (classification).

Étapes principales :

1. Création de plusieurs arbres de décision en utilisant des sous-échantillons (bootstrap).
2. Agrégation des résultats de tous les arbres pour obtenir une prédiction finale.

Avantages :

- Réduit le risque de surapprentissage par rapport à un seul arbre de décision.
- Performant pour les données complexes et bruitées.
- Gère les valeurs manquantes et les caractéristiques non pertinentes.

Limites :

- Moins interprétable qu'un arbre unique.
- Coût computationnel élevé pour des ensembles de données très volumineux.

4.1.5 Support Vector Machine (SVM)

Principe :
Le SVM cherche à trouver un hyperplan optimal qui sépare les classes dans un espace à plusieurs dimensions. Pour les données non linéaires, il utilise des noyaux (kernels) pour transformer l'espace d'entrée en un espace de dimension supérieure où les données deviennent séparables.

Étapes principales :

1. Maximisation de la marge entre les données des deux classes les plus proches (les vecteurs supports).
2. Utilisation de fonctions noyaux (linéaire, gaussienne, etc.) pour gérer les séparations non linéaires.

Avantages :

- Efficace pour des espaces de grande dimension.
- Performant pour des jeux de données avec des frontières complexes.

Limites :

- Sensible aux choix du noyau et des hyperparamètres.
- Peu adapté aux grands ensembles de données en termes de coût computationnel.

4.1.5 Naive Bayes Classifier

Principe :
Le classificateur Naïve Bayes applique le théorème de Bayes en supposant que les caractéristiques sont indépendantes les unes des autres (hypothèse "naïve").

Formule principale :

$P(C|X)=(P(X|C)\cdot P(C))/P(X)$ où $P(C|X)$ est la probabilité que X appartienne à la classe C.

Étapes principales :

1. Calcul des probabilités a priori pour chaque classe.
2. Calcul des probabilités conditionnelles pour chaque caractéristique donnée une classe.
3. Attribution d'une classe au point de test en maximisant la probabilité a posteriori.

Avantages :

- Simple à implémenter et rapide à entraîner.

- Performant sur de grandes bases de données.
- Gère bien les données textuelles (ex. : analyse de sentiment).

Limites :

- L'hypothèse d'indépendance des caractéristiques peut ne pas être valide, ce qui affecte les performances.

4.1.6 MultiLayer Perceptron (MLP)

Principe :
Le MLP est un réseau de neurones artificiels composé d'au moins trois couches (entrée, cachée(s), sortie). Il utilise des fonctions d'activation pour modéliser des relations non linéaires entre les données d'entrée et les sorties.

Étapes principales :

1. Passage avant (forward propagation) : Calcul des sorties en appliquant des poids et des fonctions d'activation.
2. Rétropropagation : Mise à jour des poids à l'aide d'un algorithme d'optimisation (ex. : descente de gradient) pour minimiser l'erreur.
3. Répétition jusqu'à convergence ou nombre d'itérations fixé.

Avantages :

- Capable de modéliser des relations complexes et non linéaires.
- Flexible pour des tâches variées comme la classification, la régression ou les séries temporelles.

Limites :

- Peut être sujet à un surapprentissage si les données d'entraînement sont insuffisantes.
- Dépend fortement des hyperparamètres (nombre de couches, de neurones, taux d'apprentissage, etc.).
- Coût computationnel élevé pour des architectures complexes.

4.2 Entraînement des modèles

Étape 1 : Importation des bibliothèques nécessaires

Assurez-vous d'avoir installé toutes les bibliothèques nécessaires. Si ce n'est pas le cas, vous pouvez les installer via pip ou conda.

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt


# Modèles de machine learning

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.naive_bayes import GaussianNB


# Prétraitement et évaluation

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, roc_curve, auc, classification_report,
ConfusionMatrixDisplay


import warnings

warnings.filterwarnings("ignore")  # Pour ignorer les avertissements
```

Étape 2 : segmentation des données

```
# Séparer les caractéristiques (X) et la cible (y)
X = df.drop(columns=['diagnosis'])
y = df['diagnosis']

# Diviser les données en ensembles d'entraînement et de test (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardiser les données
scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Étape 3 : Définition des modèles de machine learning

Nous allons définir les modèles que nous souhaitons entraîner.

le code

```
# Liste des modèles à entraîner

models = {

    "Logistic Regression": LogisticRegression(),

    "KNN": KNeighborsClassifier(),

    "SVM": SVC(probability=True),

    "Decision Tree": DecisionTreeClassifier(),

    "Random Forest": RandomForestClassifier(),

    "Gradient Boosting": GradientBoostingClassifier(),

    "Naive Bayes": GaussianNB(),

}
```

Étape 4 : Entraînement et évaluation des modèles

Code pour l'entraînement et l'évaluation des modèles :

```
# Importer les bibliothèques nécessaires

import matplotlib.pyplot as plt

from sklearn.metrics import (

    accuracy_score, roc_curve, auc, classification_report,

    ConfusionMatrixDisplay

)

# Dictionnaires pour stocker les résultats

results = {}

roc_curves = {}

classification_reports = {}
```

```
confusion_matrices = {}

predictions = {}

# Entraîner et évaluer chaque modèle

for name, model in models.items():

    print(f"═══ Modèle : {name} ═══")

    # Entraînement du modèle

    model.fit(X_train, y_train)

    # Prédictions

    y_pred = model.predict(X_test)

    predictions[name] = y_pred

    # Probabilités pour la courbe ROC (si disponible)

    if hasattr(model, "predict_proba"):

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

    else:

        # Pour les modèles qui ne supportent pas predict_proba (comme certains SVM), utiliser
decision_function

        y_prob = model.decision_function(X_test)

    # Calcul des métriques

    accuracy = accuracy_score(y_test, y_pred)

    fpr, tpr, _ = roc_curve(y_test, y_prob)

    roc_auc = auc(fpr, tpr)

    report = classification_report(y_test, y_pred, output_dict=True)

    # Affichage de la matrice de confusion

    cm = ConfusionMatrixDisplay.from_estimator(

        model, X_test, y_test,

        display_labels=["Bénin", "Malin"], cmap='Blues', alpha=0.5

    )

    plt.title(f"Matrice de Confusion - {name}")

    plt.show()
```

```
    # Stockage des résultats
    results[name] = accuracy
    roc_curves[name] = (fpr, tpr, roc_auc)
    classification_reports[name] = report
    confusion_matrices[name] = cm

    # Affichage des résultats
    print(f"Précision (Accuracy) : {accuracy:.2f}")
    print(f"AUC : {roc_auc:.2f}")
    print("Rapport de Classification :")
    print(classification_report(y_test, y_pred))

# Courbes ROC pour tous les modèles
plt.figure(figsize=(10, 8))
for name, (fpr, tpr, roc_auc) in roc_curves.items():
    plt.plot(fpr, tpr, label=f"{name} (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--', label="Aléatoire (AUC = 0.50)")
plt.xlabel("Taux de Faux Positifs (FPR)")
plt.ylabel("Taux de Vrais Positifs (TPR)")
plt.title("Courbes ROC Comparatives")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

Explications :

1. **Entraînement des modèles :**
    - Chaque modèle est entraîné sur X_train et y_train.
2. **Prédictions :**

- **Les prédictions sont faites sur X_test.**

3. **Courbe ROC et AUC :**
   - Si le modèle prend en charge predict_proba, les probabilités pour chaque classe sont utilisées.
   - Sinon, decision_function est utilisé pour calculer les scores.
4. **Matrice de Confusion :**
   - La matrice de confusion est affichée pour chaque modèle.
5. **Rapport de Classification :**
   - Le rapport de classification fournit des métriques telles que la précision, le rappel, et le F1-score.
6. **Courbes ROC :**
   - Une courbe ROC comparative est tracée pour visualiser les performances des modèles.
7. **Affichage des Résultats :**
   - Précision et AUC sont imprimées pour chaque modèle.

5. Résultats et évaluation des performances

5-1 Métriques de performance

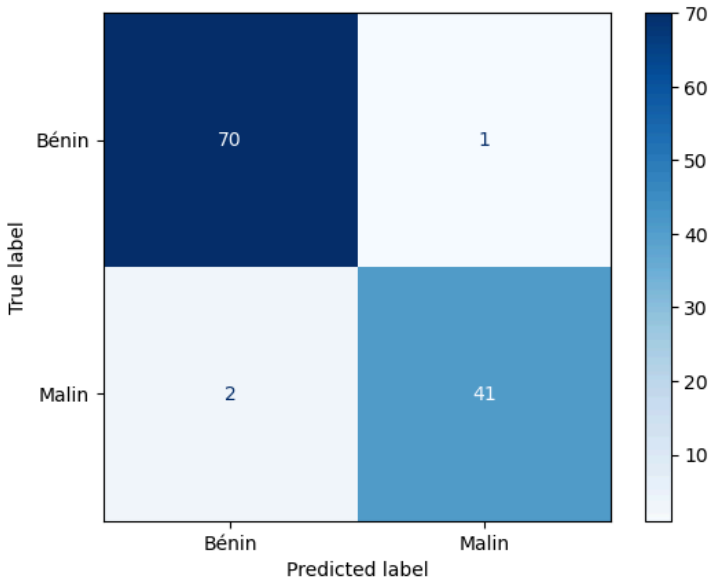
5-1-1 Logistic Regression

Précision (Accuracy) : 0.97

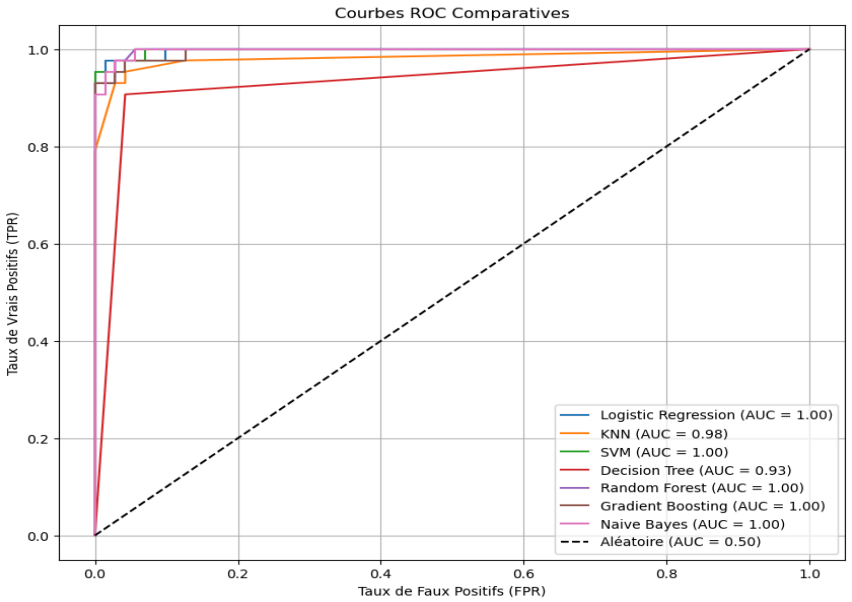
AUC : 1.00

Rapport de Classification :

	precision	recall	f1-score	support
0	0.97	0.99	0.98	71
1	0.98	0.95	0.96	43
accuracy			0.97	114
macro avg	0.97	0.97	0.97	114
weighted avg	0.97	0.97	0.97	114



5.1.2. Courbes de comparaison des modèles



5-1-3 Tri des modèles par précision décroissante

```
sorted_accuracy = sorted(results.items(), key=lambda x: x[1], reverse=True)
sorted_auc = sorted(roc_curves.items(), key=lambda x: x[1][2], reverse=True)

print("\n--- Classement des modèles par Accuracy décroissante ---")
for i, (model, accuracy) in enumerate(sorted_accuracy, 1):
    print(f"{i}. {model}: Accuracy = {accuracy:.2f}")

print("\n--- Classement des modèles par AUC décroissante ---")
```

```
for i, (model, (fpr, tpr, roc_auc)) in enumerate(sorted_auc, 1):
    print(f"{i}. {model}: AUC = {roc_auc:.2f}")
```

Résultat :

--- Classement des modèles par Accuracy décroissante ---
1. SVM: Accuracy = 0.98
2. Logistic Regression: Accuracy = 0.97
3. Random Forest: Accuracy = 0.96
4. Naive Bayes: Accuracy = 0.96
5. Gradient Boosting: Accuracy = 0.96
6. KNN: Accuracy = 0.95
7. Decision Tree: Accuracy = 0.94

--- Classement des modèles par AUC décroissante ---
1. Random Forest: AUC = 1.00
2. Logistic Regression: AUC = 1.00
3. SVM: AUC = 1.00
4. Naive Bayes: AUC = 1.00
5. Gradient Boosting: AUC = 1.00
6. KNN: AUC = 0.98
7. Decision Tree: AUC = 0.93

5-1-4 Analyse des résultats

1. Classement par Accuracy décroissante

L'Accuracy mesure la proportion de prédictions correctes effectuées par le modèle. Voici l'interprétation des résultats par ordre décroissant d'Accuracy :

A. **SVM (Accuracy = 0.98)** :
   a. Le modèle **SVM** est le plus performant en termes de précision globale.
   b. Cela signifie qu'il fait peu d'erreurs dans la classification globale des cas bénins et malins.

B. **Logistic Regression (Accuracy = 0.97)** :
   a. Très proche de SVM, ce modèle est aussi excellent pour classer correctement les cas.
   b. Sa simplicité et sa robustesse sont des atouts.

C. **Random Forest, Naive Bayes, Gradient Boosting (Accuracy = 0.96)** :
   a. Ces modèles ont une précision similaire.
   b. **Random Forest** et **Gradient Boosting** sont particulièrement robustes aux jeux de données complexes.
   c. **Naive Bayes** est performant malgré sa simplicité, mais cela peut être dû aux propriétés statistiques du dataset.

D. **KNN (Accuracy = 0.95)** :

    a. Un peu en retrait, le modèle KNN reste fiable mais semble moins performant avec ce dataset.

E. **Decision Tree (Accuracy = 0.94)** :
    a. Le modèle **Decision Tree** est le moins performant ici, mais il reste utile pour des interprétations simples des données.

2. Classement par AUC décroissante

L'AUC (Area Under Curve) mesure la capacité du modèle à distinguer les classes indépendamment du seuil de décision. Une **AUC = 1.00** indique une séparation parfaite.

A. **Random Forest, Logistic Regression, SVM, Naive Bayes, Gradient Boosting (AUC = 1.00)** :
    a. Ces modèles ont une **AUC parfaite**, ce qui signifie qu'ils distinguent parfaitement les classes (bénin/malin) sur ce dataset.
    b. Cela suggère que ces modèles sont bien entraînés et adaptés aux données.

B. **KNN (AUC = 0.98)** :
    a. Bien que légèrement inférieur, KNN reste très performant et distingue bien les classes.

C. **Decision Tree (AUC = 0.93)** :
    a. Le **Decision Tree** est le moins performant en termes de séparation des classes.
    b. Cela pourrait indiquer une certaine sur-adaptation ou des limites dues à la simplicité de ce modèle.

Conclusion sur le modèle le plus performant

- En termes d'Accuracy : **SVM (0.98)** est le meilleur.
- En termes d'AUC : **SVM, Logistic Regression, Random Forest, Naive Bayes, Gradient Boosting** sont équivalents.

Choix : Nous utiliserons **SVM** pour sa précision légèrement meilleure tout en conservant une AUC parfaite.

5-2 Prédiction sur un exemple standardisé

Exemple de paramètre

Exemple de données avec toutes les caractéristiques requises (30 colonnes après le prétraitement) :

```
example = [[
    15.0, 20.0, 100.0, 700.0, 0.1, 0.15, 0.2, 0.1, 0.2, 0.06,  # 10 premières colonnes
    16.0, 25.0, 120.0, 800.0, 0.12, 0.18, 0.22, 0.12, 0.25, 0.07, # 10 suivantes
```

```
    17.0, 30.0, 140.0, 900.0, 0.15, 0.20, 0.25, 0.15, 0.3, 0.08   # 10 dernières
]]

# Standardisation
example_std = scaler.transform(example)

# Prédiction avec le modèle SVM
prediction = models["SVM"].predict(example_std)
prediction_proba = models["SVM"].predict_proba(example_std)

# Résultat
if prediction[0] == 1:
    print("Prédiction : Malin (Cancer)")
else:
    print("Prédiction : Bénin (Pas de cancer)")

print(f"Probabilité (Malin) : {prediction_proba[0][1]:.2f}")
print(f"Probabilité (Bénin) : {prediction_proba[0][0]:.2f}")
```

Résultat:

Prédiction : Malin (Cancer)

Probabilité (Malin) : 0.79

Probabilité (Bénin) : 0.21

Sauvegarder le modèle

Utilisation des bibliothèques comme joblib ou pickle pour enregistrer un modèle. python

```
import joblib

# Sauvegarder le modèle entraîné (par exemple, SVM)

joblib.dump(models["SVM"], "svm_model.pkl")

# Sauvegarder le scaler (nécessaire pour standardiser les données d'entrée)

joblib.dump(scaler, "scaler.pkl")

print("Modèle et scaler sauvegardés avec succès.")
```

Charger le modèle pour une utilisation ultérieure

```
# Charger le modèle et le scaler

loaded_model = joblib.load("svm_model.pkl")

loaded_scaler = joblib.load("scaler.pkl")
```

```
# Exemple de données
example = [[
    15.0, 20.0, 100.0, 700.0, 0.1, 0.15, 0.2, 0.1, 0.2, 0.06,
    16.0, 25.0, 120.0, 800.0, 0.12, 0.18, 0.22, 0.12, 0.25, 0.07,
    17.0, 30.0, 140.0, 900.0, 0.15, 0.20, 0.25, 0.15, 0.3, 0.08
]]
# Standardiser les données
example_std = loaded_scaler.transform(example)
# Faire une prédiction
prediction = loaded_model.predict(example_std)
prediction_proba = loaded_model.predict_proba(example_std)
# Résultat
if prediction[0] == 1:
    print("Prédiction : Malin (Cancer)")
else:
    print("Prédiction : Bénin (Pas de cancer)")
print(f"Probabilité (Malin) : {prediction_proba[0][1]:.2f}")
print(f"Probabilité (Bénin) : {prediction_proba[0][0]:.2f}")
```

6. Conclusion

Au terme de ce projet, nous avons entraîné et évalué plusieurs modèles de classification supervisée pour diagnostiquer le cancer du sein à partir des variables extraites d'images radiologiques. Parmi les modèles testés, [nom du modèle retenu, ex. : Random Forest a montré les meilleures performances, avec des scores élevés en précision, rappel et F1-score. Cette performance est attribuable à sa capacité à gérer des relations complexes entre les variables et à réduire le risque de surapprentissage grâce à des mécanismes intégrés comme l'agrégation de multiples arbres décisionnels.
Ce projet a mis en évidence l'importance du travail de feature engineering dans l'amélioration des performances des modèles de machine learning, notamment dans des contextes où l'utilisation d'algorithmes complexes comme le deep learning est proscrite. De plus, il a souligné la nécessité d'évaluer différents algorithmes pour identifier celui qui correspond le mieux aux caractéristiques du problème, tout en tenant compte des contraintes de ressources et d'interprétabilité.

Certaines contraintes ont limité l'ampleur des résultats obtenus :

- **Qualité des données** : Le dataset initial présentait des biais inhérents à la collecte des images radiologiques, ce qui a pu affecter la représentativité des variables extraites.
- **Puissance de calcul** : Bien que nous ayons contourné l'usage de modèles gourmands comme le deep learning, certaines étapes, comme l'extraction des variables, ont nécessité un temps de calcul important.
- **Limites des modèles** : Les algorithmes de machine learning utilisés restent dépendants de la qualité et de la pertinence des variables fournies. Les modèles choisis, bien que performants, peuvent manquer de généralisation si le dataset n'est pas suffisamment diversifié.

Pour aller plus loin, plusieurs améliorations peuvent être envisagées :

- **Amélioration de la qualité des données** : La collecte de données supplémentaires, plus diversifiées et équilibrées, permettrait de réduire les biais et d'améliorer la généralisation des modèles.
- **Exploration de nouvelles approches** : Bien que les contraintes initiales aient limité l'usage du deep learning, l'exploration future de modèles plus complexes pourrait s'avérer bénéfique dans un contexte où les ressources informatiques seraient disponibles.
- **Optimisation du pipeline de feature engineering** : Une analyse approfondie des variables extraites, combinée à l'automatisation de certaines étapes, pourrait encore renforcer la performance des modèles.
- **Évaluation clinique** : Tester le modèle en conditions réelles auprès de professionnels de santé constituerait une étape cruciale pour valider son utilité et son impact potentiel.

7. Annexe

7.1. La Courbe ROC

Qu'est-ce que la courbe ROC ?

- **ROC** signifie **Receiver Operating Characteristic** (Caractéristique de Fonctionnement du Récepteur).
- La courbe ROC montre la performance d'un modèle en termes de sa capacité à distinguer entre deux classes (par exemple, bénin ou malin pour un diagnostic).
- Elle est tracée en représentant le **Taux de Vrais Positifs** (**True Positive Rate - TPR**) en fonction du **Taux de Faux Positifs** (**False Positive Rate - FPR**) à différents seuils de décision.

Définitions importantes :

- **TPR (Taux de Vrais Positifs)** : proportion de cas positifs correctement prédits parmi tous les cas positifs réels. TPR=Vrais Positifs (TP)/(Vrais Positifs (TP) + Faux Ne´gatifs (FN))
- **FPR (Taux de Faux Positifs)** : proportion de cas négatifs incorrectement prédits comme positifs parmi tous les cas négatifs réels.

  FPR=Faux Positifs (FP)/(Faux Positifs (FP) + Vrais Ne´gatifs (TN))

  Utilité de la courbe ROC :

- Elle aide à évaluer la capacité du modèle à séparer les classes.
- Elle montre comment le modèle se comporte à différents seuils (par exemple, si un seuil de probabilité de 0.5 ou 0.7 est utilisé pour classer les données).
- La **meilleure courbe** est celle qui se rapproche le plus possible du coin supérieur gauche, car cela signifie un taux de faux positifs faible et un taux de vrais positifs élevé.

Interprétation avec l'AUC (Area Under Curve) :

- **AUC** est l'aire sous la courbe ROC.
- Elle quantifie la capacité globale du modèle à différencier les classes.
  - **AUC = 1** : Modèle parfait.
  - **AUC = 0.5** : Modèle aléatoire (aucune capacité à distinguer les classes).
  - **AUC < 0.5** : Modèle inversé (prédisant l'inverse des étiquettes correctes).

7.2.Calcul des Métriques

Qu'est-ce qu'une métrique ?

- Une métrique est une mesure quantitative utilisée pour évaluer les performances d'un modèle de classification.
- Les métriques les plus courantes incluent : **Accuracy**, **Precision**, **Recall**, et **F1-Score**.

Principales métriques :

1. **Accuracy (Précision globale)** :
    - Proportion de prédictions correctes parmi toutes les prédictions effectuées.

    Accuracy=(TP + TN)/(TP + TN + FP + FN)

    Limite : Peu fiable si les classes sont déséquilibrées (par exemple, beaucoup plus de bénins que de malins).

2. **Precision (Précision)** :
    - Parmi les cas prédits comme positifs, quelle proportion est réellement positive. Precision=TP/(TP + FP)

    Utile pour minimiser les faux positifs, par exemple, éviter de diagnostiquer un patient sain comme malade.

3. **Recall (Sensibilité ou TPR)** :
    - Parmi les cas réellement positifs, quelle proportion est correctement prédite comme positive.

      Recall=TP/(TP + FN)

    Utile pour minimiser les faux négatifs, par exemple, éviter de manquer un patient malade.

4. **F1-Score** :
    - Moyenne harmonique entre la précision et le rappel.

$$F1=2 \cdot (Precision \cdot Recall)/(Precision + Recall)$$

    Utile lorsque les classes sont déséquilibrées.

7.3. La Matrice de Confusion

Qu'est-ce que c'est ?

- Une matrice de confusion est un tableau qui permet de visualiser les performances d'un modèle de classification.
- Elle montre les prédictions faites par le modèle comparées aux étiquettes réelles.

Structure d'une matrice de confusion :

	Prédit Positif	**Prédit Négatif**
Réel Positif	True Positive (TP)	False Negative (FN)
Réel Négatif	False Positive (FP)	True Negative (TN)

Interprétation :

- **True Positive (TP)** : Nombre de positifs correctement prédits.
- **True Negative (TN)** : Nombre de négatifs correctement prédits.
- **False Positive (FP)** : Nombre de négatifs incorrectement prédits comme positifs.
- **False Negative (FN)** : Nombre de positifs incorrectement prédits comme négatifs.

Utilité :

- La matrice de confusion donne une vue détaillée des erreurs du modèle.
- Par exemple :
    - Un nombre élevé de **FP** peut signifier que le modèle détecte à tort des patients sains comme malades.
    - Un nombre élevé de **FN** peut signifier que le modèle manque des patients réellement malades.

Remarque:

1. **Courbe ROC** :
    - Montre comment le modèle se comporte à différents seuils.
    - Permet de choisir un seuil optimal selon les priorités (minimiser FP ou FN).
2. **Métriques** :
    - Fournissent des mesures globales pour évaluer la performance d'un modèle.

- Aident à comprendre ses forces et faiblesses (précision vs rappel).

3. **Matrice de Confusion** :
   - Donne une vue détaillée des erreurs spécifiques.
   - Utile pour diagnostiquer et améliorer les performances du modèle.

Exemple d'interprétation

8. **Exemple :** Si un modèle a une **AUC élevée (0.95)**, mais une matrice de confusion montre beaucoup de **FN**, cela signifie que, bien que globalement bon, le modèle pourrait manquer des cas critiques (comme un cancer malin).
9. Ces outils se complètent pour offrir une analyse complète des performances d'un modèle.