

Breast cancer Prediction

Développement avec Flask, conteneurisation avec
Docker et déploiement sur GCP

Réalisé par :

AGJOUD Imad

ARAGOU Wassef

MALIH Mouad

Encadré par :

Mr.LAMRANI

TABLE OF CONTENTS

- **Introduction** 01
- **Modèle de prédiction** 02
- **Structure de l'application** 03
- **Conteneurisation avec Docker** 04
- **Déploiement sur GCP** 05
- **Conclusion** 06

Introduction

- Problématique
- Objectif
- Technologies utilisées



Problématique

Le cancer du sein constitue un défi majeur de santé publique, nécessitant une détection précoce, un diagnostic précis et une évaluation des coûts de traitement.

Ces processus, bien que cruciaux, sont souvent complexes et nécessitent des outils adaptés pour aider les professionnels de santé et les patients à prendre des décisions éclairées.

Objectif

Dans ce cadre, nous avons créé l'application **Cancer Prediction App** pour offrir une solution accessible et précise, permettant de prédire le risque de cancer du sein, son stade, et d'estimer les coûts de traitement, afin de faciliter la prise de décision pour les professionnels de santé et les patients.

Technologies utilisées



- Framework léger et flexible en Python, pour le développement de l'interface web et la gestion des interactions utilisateur, garantissant une expérience fluide et rapide.

Technologies utilisées



- HTML structure le contenu de la page.
- CSS assure un design moderne et responsive.
- JavaScript est utilisé pour gérer les interactions dynamiques et améliorer l'expérience utilisateur.

Technologies utilisées



- Créer un environnement isolé et reproductible pour son déploiement.
- Une gestion simplifiée des dépendances et un déploiement cohérent sur diverses plateformes.

Technologies utilisées



- Services cloud ,permet de déployer, gérer et scaler des applications de manière flexible et sécurisée
- GCP permet de gérer facilement les ressources et d'assurer une haute disponibilité.

Modèle de prédiction

- Prétraitement des Données
- construction du Modèle
- Evaluation du performance



Les modèles de prédiction:

- **Prédire le cancer et le coût du traitement :**
Utilise Elastic Net and Random forest models.
- **Prédire le type de tumeur :** Utilise le modèle Elastic Net pour prédire si une tumeur du sein est bénigne ou maligne.

Variables analysées : Facteurs médicaux variés

- Clump Thickness
- Uniformity of Cell Size
- Uniformity of Cell Shape
- Marginal Adhesion
- Single Epithelial Cell Size
- Bare Nuclei
- Bland Chromatin
- Normal Nucleoli
- Mitoses

breast_cancer_bd

- **À propos de ce jeu de données :** Ce jeu de données sur le cancer du sein a été obtenu des hôpitaux de l'Université du Wisconsin, à Madison, grâce au Dr William H. Wolberg.
- Les attributs de 1 à 10 sont utilisés pour représenter les instances. Chaque instance a l'une des 2 classes possibles : bénin ou malin.

Attribute	Domain
1. Sample code number	id number
2. Clump Thickness	1 - 10
3. Uniformity of Cell Size	1 - 10
4. Uniformity of Cell Shape	1 - 10
5. Marginal Adhesion	1 - 10
6. Single Epithelial Cell Size	1 - 10
7. Bare Nuclei	1 - 10
8. Bland Chromatin	1 - 10
9. Normal Nucleoli	1 - 10
10. Mitoses	1 - 10
11. Class (2 for benign, 4 for malignant)	

- **Class Distribution**
 - Benign:(65.5%)
 - Malignant:(34.5%)

breast_cancer_bd

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

Description de la base de donnees:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bland Chromatin	Normal Nucleoli	Mitoses	Class
count	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000
mean	4.417740	3.134478	3.207439	2.806867	3.216023	3.437768	2.866953	1.589413	2.689557
std	2.815741	3.051459	2.971913	2.855379	2.214300	2.438364	3.053634	1.715078	0.951273
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2.000000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.000000	2.000000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	3.000000	1.000000	1.000000	2.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	5.000000	4.000000	1.000000	4.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	4.000000

breast_cancer_bd

```
data = data.replace('?', np.nan)

data['Bare Nuclei'] = pd.to_numeric(data['Bare Nuclei'])
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Clump Thickness  699 non-null    int64  
 1   Uniformity of Cell Size  699 non-null    int64  
 2   Uniformity of Cell Shape 699 non-null    int64  
 3   Marginal Adhesion    699 non-null    int64  
 4   Single Epithelial Cell Size 699 non-null    int64  
 5   Bare Nuclei        683 non-null    float64 
 6   Bland Chromatin    699 non-null    int64  
 7   Normal Nucleoli   699 non-null    int64  
 8   Mitoses           699 non-null    int64  
 9   Class              699 non-null    int64  
dtypes: float64(1), int64(9)
memory usage: 54.7 KB
```

- Remplacement des valeurs manquantes : Il remplace d'abord les valeurs manquantes représentées par des "?" par NaN, facilitant ainsi leur identification.
- Conversion du type de données : Il convertit la colonne 'Bare Nuclei', qui contient des valeurs manquantes, en type numérique (float64), permettant de manipuler et traiter ces valeurs.

```
print(data.isnull().sum())
```

Clump Thickness	0
Uniformity of Cell Size	0
Uniformity of Cell Shape	0
Marginal Adhesion	0
Single Epithelial Cell Size	0
Bare Nuclei	16
Bland Chromatin	0
Normal Nucleoli	0
Mitoses	0
Class	0
dtype:	int64

Vérification des valeurs manquantes

: Le code affiche un résumé de chaque colonne (avec `data.info()`), suivi d'un comptage des valeurs manquantes pour chaque colonne, indiquant qu'il y a 16 valeurs manquantes dans la colonne 'Bare Nuclei'.

```
data['Bare Nuclei'] = data['Bare Nuclei'].fillna(data['Bare Nuclei'].median())
print(data.isnull().sum())
```

```
Clump Thickness      0
Uniformity of Cell Size    0
Uniformity of Cell Shape    0
Marginal Adhesion      0
Single Epithelial Cell Size 0
Bare Nuclei          0
Bland Chromatin        0
Normal Nucleoli       0
Mitoses              0
Class                0
dtype: int64
```

- **Imputation des valeurs manquantes :** Les valeurs manquantes dans la colonne 'Bare Nuclei' sont remplacées par la médiane de cette colonne, une méthode courante pour éviter la distorsion des données.
- **Vérification finale :** Enfin, il vérifie qu'il ne reste aucune valeur manquante, ce qui indique que le jeu de données est prêt pour une utilisation ultérieure.

```
X = data.drop('Class', axis=1)
y = data['Class']
scaler = StandardScaler()
# Train-test split and model training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **Séparation des variables d'entrée et de la cible :**

X = data.drop('Class', axis=1): Crée X, le sous-ensemble des données sans la colonne Class, qui représente les caractéristiques pour l'entraînement.

y = data['Class']: Crée y, la variable cible contenant les classes (benin ou malin) pour chaque échantillon.

- **Standardisation des données :**

Prépare un standardiseur, utile pour mettre les données sur la même échelle, ce qui améliore souvent la performance de certains algorithmes d'apprentissage.

- **Division des données en ensembles d'entraînement et de test :**

Diviser les données en ensembles d'entraînement (80%) et de test (20%), en maintenant un ordre aléatoire constant avec random_state=42 pour garantir la reproductibilité des résultats.

Prédiction du type de tumeur

Comparaison des modèles :

```
# Train-test split and model training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.linear_model import (
    LinearRegression, Ridge, Lasso, ElasticNet
)

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
# Initialize the models
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'LASSO Regression': Lasso(),
    'Elastic Net': ElasticNet(),
    'Random Forest': RandomForestRegressor(random_state=42),
}

# Train and evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = r2_score(y_test, y_pred)
    print(f"{name} R^2 score: {score:.4f}")
```

```
Linear Regression R^2 score: 0.8262
Ridge Regression R^2 score: 0.8262
LASSO Regression R^2 score: 0.6244
Elastic Net R^2 score: 0.7473
Random Forest R^2 score: 0.8459
```

Divers modèles de régression et de classification sont entraînés et évalués en utilisant le score R²

Optimisation du modèle “Elastic NET”:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
# Define the ElasticNet model
elastic_net = ElasticNet()

# Create a wider parameter grid for ElasticNet
param_grid = {
    'alpha': np.logspace(-10, 10, 100), # Wider range of alpha values
    'l1_ratio': np.linspace(0, 1, 11), # Mix of Lasso (1) and Ridge (0)
}

# Set up Grid Search with cross-validation
grid_search = GridSearchCV(estimator=elastic_net, param_grid=param_grid,
                           scoring='neg_mean_squared_error',
                           cv=5,
                           verbose=1,
                           n_jobs=-1) # Use all available cores

# Fit the model
grid_search.fit(X_train_scaled, y_train)

# Output best parameters and cross-validation score
print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation score (MSE): ", -grid_search.best_score_)

# Use the best estimator to make predictions
best_elastic_net = grid_search.best_estimator_
y_pred = best_elastic_net.predict(X_test_scaled)
# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Test set MSE: ", mse)

# Calculate R^2 score
test_score = r2_score(y_test, y_pred)
print("Test Set R^2 Score: ", test_score)
```

```
Fitting 5 folds for each of 1100 candidates, totalling 5500 fits
Best parameters found: {'alpha': 0.7924828983539186, 'l1_ratio': 0.9}
Best cross-validation score (MSE): 151325702.23363513
Test set MSE: 136692929.69417048
Test Set R^2 Score: 0.9080664805852825
```

- GridSearchCV est utilisé pour effectuer une recherche par validation croisée des meilleurs hyperparamètres alpha et l1_ratio pour le modèle ElasticNet. La grille de paramètres couvre un large éventail de valeurs possibles pour alpha et l1_ratio.
- La validation croisée est réalisée avec 5 plis, et la métrique utilisée est l'erreur quadratique moyenne négative (neg_mean_squared_error).

Optimisation du modèle “Elastic NET”:

Le modèle imprime les meilleurs paramètres, le MSE de validation croisée, le MSE sur l'ensemble de test, et le score R² pour évaluer la qualité de la prédiction.

```
Fitting 5 folds for each of 1100 candidates, totalling 5500 fits
Best parameters found: {'alpha': 0.04862601580065353, 'l1_ratio': 0.1}
Best cross-validation score (MSE): 0.15462709492758378
Test set MSE: 0.15273332598065525
Test Set R^2 Score: 0.8249372403964419
```

La précision du modèle est passée de 74.73% à 82.49%

Objectif : Estimer la probabilité de cancer

- **Calcul de la Probabilité :** Le modèle RandomForestRegressor utilise des données médicales pour générer une prédiction.
- **Prédiction Finale :** Le modèle considère le cancer comme bénin si la prédiction est inférieure à 3, sinon comme malin.
- **Utilité Clinique :** En fournissant un JSON avec le type de cancer probable et les probabilités, l'application aide les cliniciens à évaluer rapidement le risque et à décider d'une intervention rapide en cas de risque élevé.

Prédire le cancer et le coût du traitement :

Prediction du charge d'assurance

```
# Train-test split and model training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.linear_model import (
    LinearRegression, Ridge, Lasso, ElasticNet
)

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
# Initialize the models
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'LASSO Regression': Lasso(),
    'Elastic Net': ElasticNet(),
    'Random Forest': RandomForestRegressor(random_state=42),
}

# Train and evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = r2_score(y_test, y_pred)
    print(f"{name} R^2 score: {score:.4f}")
```

```
Linear Regression R^2 score: 0.9032
Ridge Regression R^2 score: 0.9032
LASSO Regression R^2 score: 0.9032
Elastic Net R^2 score: 0.9087
Random Forest R^2 score: 0.9098
```

Divers modèles de régression et de classification sont entraînés pour prédire la charge d'assurance et évalués en utilisant le score R²

optimisation du modèle:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
# Define the ElasticNet model
elastic_net = ElasticNet()

# Create a wider parameter grid for ElasticNet
param_grid = {
    'alpha': np.logspace(-10, 10, 100), # Wider range of alpha values
    'l1_ratio': np.linspace(0, 1, 11), # Mix of Lasso (1) and Ridge (0)
}

# Set up Grid Search with cross-validation
grid_search = GridSearchCV(estimator=elastic_net, param_grid=param_grid,
                           scoring='neg_mean_squared_error',
                           cv=5,
                           verbose=1,
                           n_jobs=-1) # Use all available cores

# Fit the model
grid_search.fit(X_train_scaled, y_train)

# Output best parameters and cross-validation score
print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation score (MSE): ", -grid_search.best_score_)

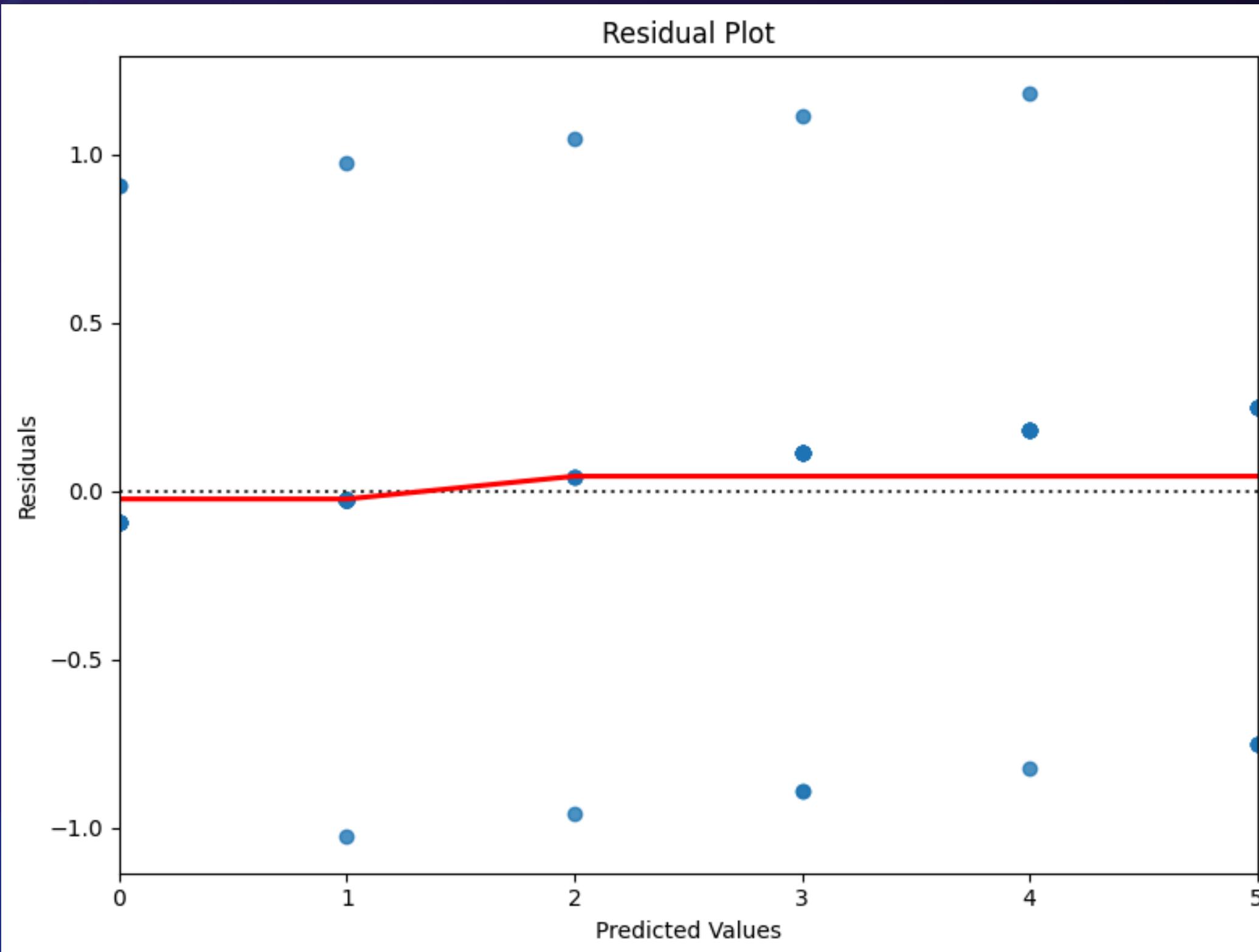
# Use the best estimator to make predictions
best_elastic_net = grid_search.best_estimator_
y_pred = best_elastic_net.predict(X_test_scaled)
# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Test set MSE: ", mse)

# Calculate R2 score
test_score = r2_score(y_test, y_pred)
print("Test Set R^2 Score:", test_score)
```

```
Fitting 5 folds for each of 1100 candidates, totalling 5500 fits
Best parameters found: {'alpha': 0.7924828983539186, 'l1_ratio': 0.9}
Best cross-validation score (MSE):  151325702.23363513
Test set MSE:  136692929.69417048
Test Set R^2 Score: 0.9080664805852825
```

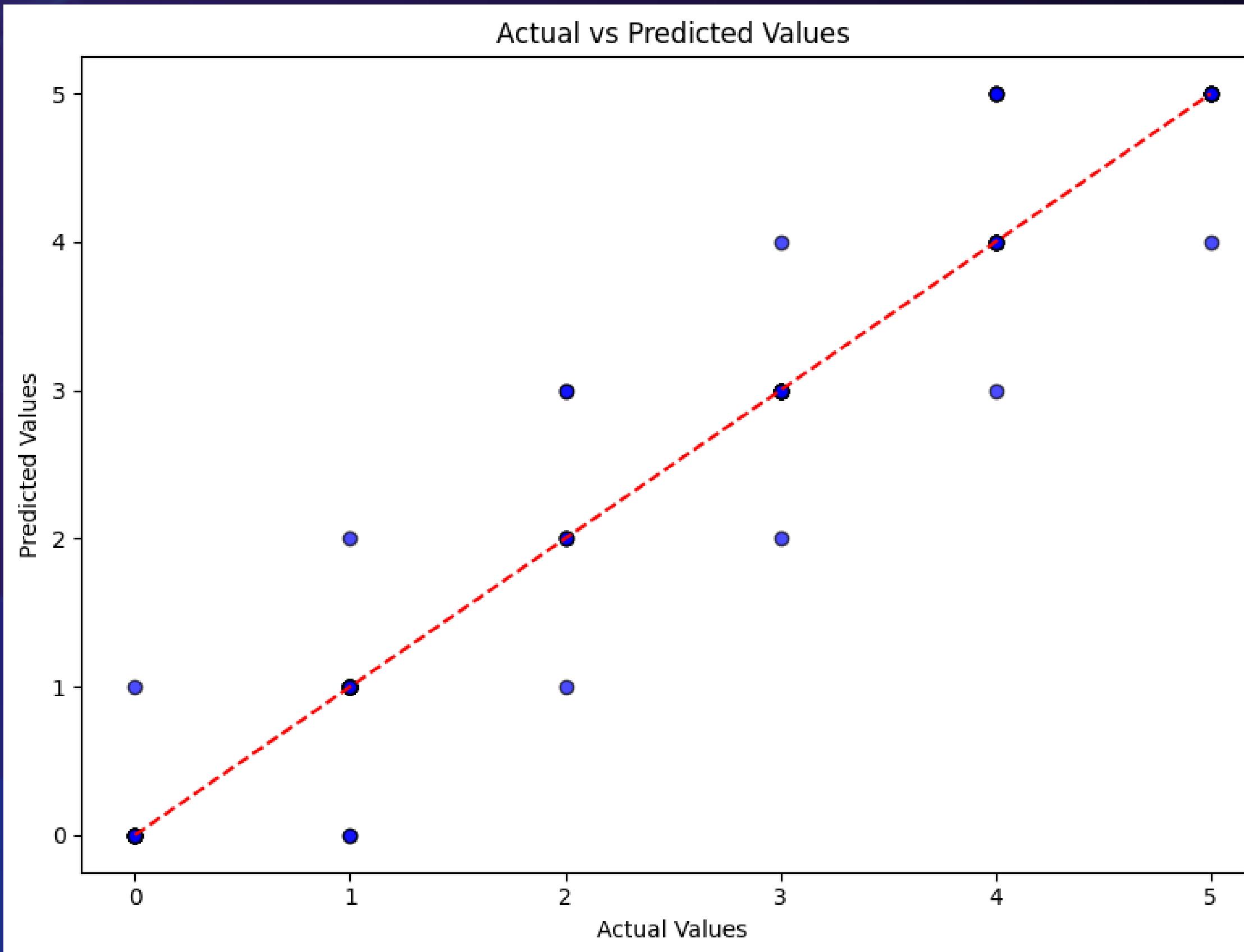
Ce code utilise ElasticNet et une recherche de grille pour optimiser les paramètres alpha et l1_ratio, en combinant les régularisations Lasso et Ridge. La recherche, via validation croisée (5 fois), teste 5500 combinaisons et choisit les paramètres minimisant l'erreur quadratique moyenne (MSE). Le meilleur modèle (avec alpha = 0,79 et l1_ratio = 0,9) obtient un MSE de 136,7 millions et un R² de 0,91 sur les données de test, indiquant une bonne précision du modèle.

Graphique des Résidus



- Un graphique des résidus aide à évaluer si le modèle présente un schéma dans ses erreurs. Si le modèle est performant, les résidus devraient être dispersés aléatoirement autour de zéro sans former de motifs particuliers.
- Ce graphique montrera si les résidus (erreurs) sont répartis uniformément autour de zéro, ce qui est idéal pour un bon modèle. Un motif dans les résidus suggère que le modèle pourrait ne pas capturer un aspect important des données.

Prediction vs Actual Plot



- Ce graphique compare les valeurs prédites aux valeurs réelles. Un modèle parfait aurait tous les points situés sur la ligne $y = x$
- La ligne en pointillés rouges représente le cas idéal où les valeurs prédites sont exactement les mêmes que les valeurs réelles. Les points proches de cette ligne indiquent de meilleures prédictions.

Prediction du stade du cancer:

```
> import pandas as pd  
from sklearn.model_selection import train_test_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.preprocessing import StandardScaler  
from sklearn.metrics import classification_report  
import joblib  
  
# Initialize the Random Forest Classifier  
clf = RandomForestClassifier(n_estimators=100, random_state=42)  
  
# Train the classifier  
clf.fit(X_train_scaled, y_train)  
  
# Test the model  
y_pred = clf.predict(X_test_scaled)  
print(classification_report(y_test, y_pred))
```

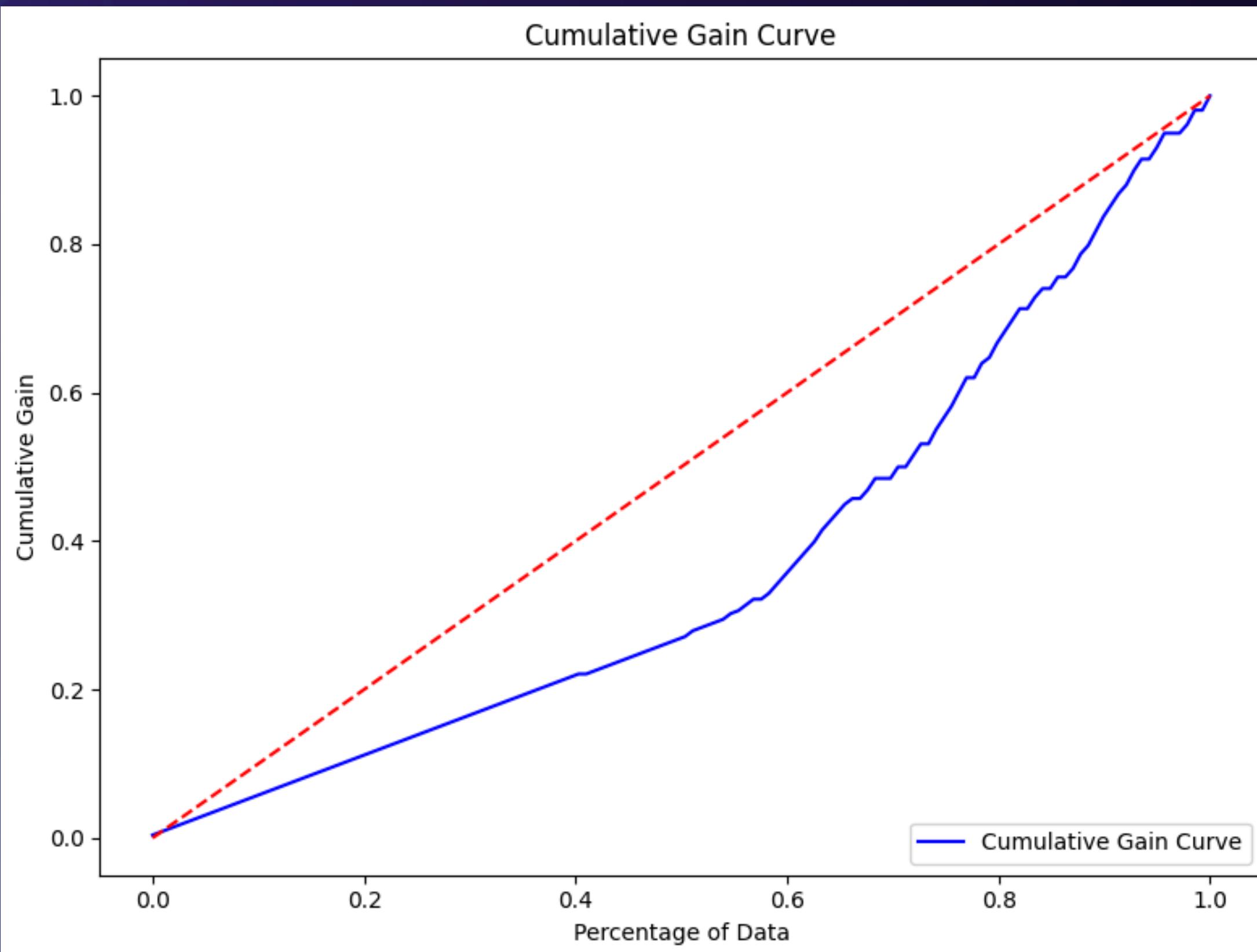
Ce code entraîne un modèle de classification (Random Forest) pour prédire le stade du cancer basé sur un ensemble de données de caractéristiques des tumeurs.

Prediction du stage du cancer:

	precision	recall	f1-score	support
0	0.88	0.93	0.90	15
1	0.97	0.96	0.97	75
2	0.67	0.57	0.62	7
3	0.81	0.87	0.84	15
4	0.82	0.56	0.67	16
5	0.65	0.92	0.76	12
accuracy			0.88	140
macro avg	0.80	0.80	0.79	140
weighted avg	0.88	0.88	0.88	140

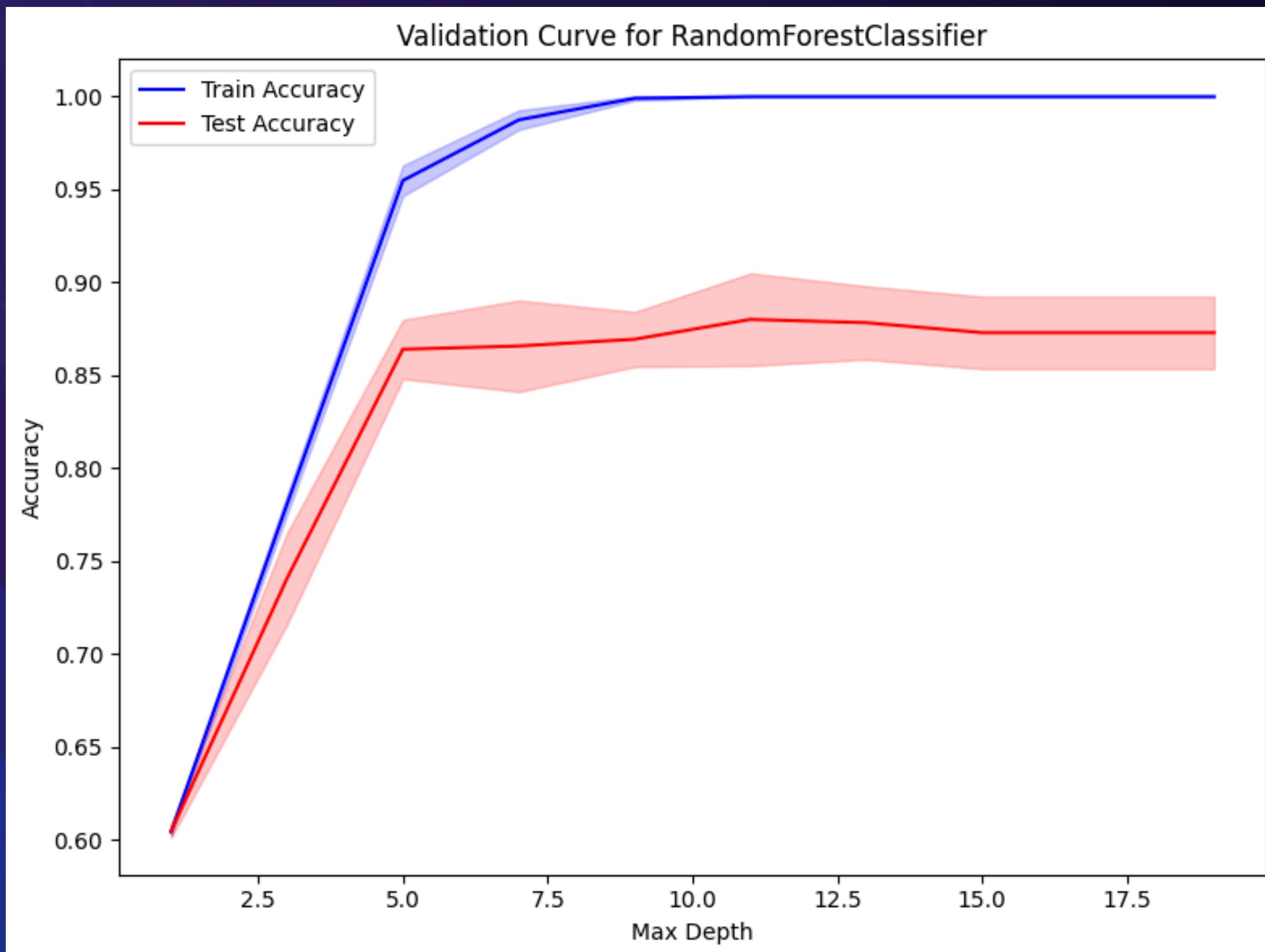
- Le rapport montre la précision par classe de stade du cancer ainsi qu'une moyenne pondérée et une macro moyenne pour évaluer l'ensemble de la performance du modèle.
- Dans cet exemple, l'exactitude globale (accuracy) est de 88 %.
- Ce modèle peut être utilisé pour prédire le stade du cancer en fournissant de nouvelles observations de caractéristiques

Cumulative Gain / Lift Curve



La courbe de gain cumulé montre le pourcentage d'instances positives capturées à mesure que l'on parcourt la liste des probabilités prédites. Cela est utile pour évaluer dans quelle mesure votre modèle classe correctement les instances par probabilité.

Validation Curve



Une courbe de validation est un graphique qui montre la relation entre la performance d'un modèle et la valeur d'un hyperparamètre spécifique. En faisant varier l'hyperparamètre et en traçant la performance du modèle à la fois sur l'ensemble d'entraînement et sur l'ensemble de validation,

Structure de l'application

- Back-End
 - Flask
- Front-End
 - Les interfaces



Chargement des Modèles et Scalers python Copy code

```
~> from flask import Flask, render_template, request, jsonify
    import joblib
    import pandas as pd
    app = Flask(__name__)

    model = joblib.load('models_files/best_elastic_net.joblib')
    scaler_loaded = joblib.load('models_files/scaler_Elastic_Net.joblib')
    # Load the pre-trained models and scalers
    price_model = joblib.load('models_files/price_model.joblib') # Elastic Net model for price
    price_scaler = joblib.load('models_files/price_scaler.joblib')

    stage_model = joblib.load('models_files/stage_model.joblib') # Classification model for stage
    stage_scaler = joblib.load('models_files/stage_scaler.joblib')
```

Les modèles pré-entraînés et les scalers sont chargés à partir de fichiers sauvegardés. price_model est utilisé pour les prédictions continues (par exemple, le prix), tandis que stage_model sert à classifier les stades du cancer. Les scalers correspondants normalisent les données d'entrée avant les prédictions.

Routes pour Rendre les Templates HTML

```
@app.route('/')
def home():
    return render_template('index.html')
@app.route('/explaine')
def explaine_page():
    return render_template('explaine.html')
@app.route('/predict')
def predict_page():
    return render_template('predict.html')

@app.route('/predict1')
def predict1_page():
    return render_template('predict1.html')

@app.route('/predict1', methods=['POST'])
```

Les routes suivantes renvoient des templates HTML pour différentes pages de l'application :

- @app.route('/') : Affiche la page d'accueil via le fichier index.html.
- @app.route('/explaine') et @app.route('/predict') : Redirigent vers les pages d'explication (explaine.html) et de prédiction (predict.html).
- Les autres routes comme /aboutus, /learnmore, et /ML redirigent vers des pages supplémentaires pour fournir plus d'informations aux utilisateurs.

predict 1:

```
def predict1():
    data = request.json

    # Extract features
    features = [
        data.get('Clump Thickness', 0),
        data.get('Uniformity of Cell Size', 0),
        data.get('Uniformity of Cell Shape', 0),
        data.get('Marginal Adhesion', 0),
        data.get('Single Epithelial Cell Size', 0),
        data.get('Bare Nuclei', 0),
        data.get('Bland Chromatin', 0),
        data.get('Normal Nucleoli', 0),
        data.get('Mitoses', 0)
    ]

    # Convert features to DataFrame
    features_df = pd.DataFrame([features], columns=[
        'Clump Thickness',
        'Uniformity of Cell Size',
        'Uniformity of Cell Shape',
        'Marginal Adhesion',
        'Single Epithelial Cell Size',
        'Bare Nuclei',
        'Bland Chromatin',
        'Normal Nucleoli',
        'Mitoses'
    ])

    # Scale features for price prediction
    features_scaled_price = price_scaler.transform(features_df)
    predicted_price = price_model.predict(features_scaled_price)[0]

    # Scale features for stage prediction
    features_scaled_stage = stage_scaler.transform(features_df)
    predicted_stage = stage_model.predict(features_scaled_stage)[0]

    # Create a response with both predictions
    response = {
        "predicted_value": round(predicted_price, 3),
        "predicted_stage": int(predicted_stage)
    }

    return jsonify(response)
```

predict 2:

```
def predict2():
    data = request.json

    # Extract and validate features to ensure values are between 1 and 10
    features = [
        data.get('Clump Thickness', 0),
        data.get('Uniformity of Cell Size', 0),
        data.get('Uniformity of Cell Shape', 0),
        data.get('Marginal Adhesion', 0),
        data.get('Single Epithelial Cell Size', 0),
        data.get('Bare Nuclei', 0),
        data.get('Bland Chromatin', 0),
        data.get('Normal Nucleoli', 0),
        data.get('Mitoses', 0)
    ]

    # Ensure each feature is within the range [1, 10]
    features = [min(max(f, 1), 10) for f in features]

    # Convert features to a DataFrame
    features_df = pd.DataFrame([features], columns=[
        'Clump Thickness',
        'Uniformity of Cell Size',
        'Uniformity of Cell Shape',
        'Marginal Adhesion',
        'Single Epithelial Cell Size',
        'Bare Nuclei',
        'Bland Chromatin',
        'Normal Nucleoli',
        'Mitoses'
    ])

    # Scale the features using the loaded scaler
    features_scaled = scaler_loaded.transform(features_df)

    # Get the prediction from the regressor
    prediction = model.predict(features_scaled)[0]

    # Calculate probabilities based on the prediction
    if prediction <= 2:
        benign_prob = 100
        malignant_prob = 0
    else:
        benign_prob = 0
        malignant_prob = 100
```

Prédiction de Prix et de Stade

- Endpoint /predict1 : Utilisé pour la prédiction de prix et de stade du cancer.
- Les données d'entrée sont reçues sous format JSON.
- Les caractéristiques sont extraites, puis mises sous forme de DataFrame.
- Les données sont mises à l'échelle avec les scalers price_scaler et stage_scaler avant de passer dans les modèles price_model et stage_model.
- Le prix prédit (predicted_price) et le stade du cancer (predicted_stage) sont renvoyés au format JSON.
- Endpoint /predict2 : Similaire à /predict1, mais ajoute une validation pour que chaque caractéristique soit dans l'intervalle [1, 10]. Ici, scaler_loaded est utilisé pour mettre à l'échelle les caractéristiques, et le modèle principal (model) est appliqué pour prédire si la tumeur est bénigne ou maligne, avec une probabilité associée

Exécution de l'Application

L'application écoute sur le port 8080 pour des connexions externes, ce qui est utile pour un déploiement en production.

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

Prédiction de Prix et de Stade

Le site web donne une expérience fluide où chaque page est un espace interactif et informatif, qui joue un rôle clé dans l'expérience utilisateur.

- ↳ [aboutus.html](#)
- ↳ [error.html](#)
- ↳ [explain.html](#)
- ↳ [explaine.html](#)
- ↳ [index.html](#)
- ↳ [learnmore.html](#)
- ↳ [MLhtml](#)
- ↳ [predict.html](#)
- ↳ [predict1.html](#)
- ↳ [predict2.html](#)
- ↳ [status.html](#)

Des pages d'introduction et d'information, comme [aboutus.html](#) et [learnmore.html](#), à celles de prédiction et de statut, telles que [predict.html](#) et [status.html](#), chaque section est conçue pour guider, éduquer et fournir des résultats, tout en assurant une navigation intuitive et cohérente.

développement front-end

```
function submitForm1() {
  const data = {
    'Clump Thickness': parseInt(document.getElementById('clump_thickness').value),
    'Uniformity of Cell Size': parseInt(document.getElementById('cell_size').value),
    'Uniformity of Cell Shape': parseInt(document.getElementById('cell_shape').value),
    'Marginal Adhesion': parseInt(document.getElementById('marginal_adhesion').value),
    'Single Epithelial Cell Size': parseInt(document.getElementById('epithelial_cell_size').value),
    'Bare Nuclei': parseInt(document.getElementById('bare_nuclei').value),
    'Bland Chromatin': parseInt(document.getElementById('bland_chromatin').value),
    'Normal Nucleoli': parseInt(document.getElementById('normal_nucleoli').value),
    'Mitoses': parseInt(document.getElementById('mitoses').value)
  };

  // Clear previous error message
  const errorMessage = document.getElementById('errorMessage');
  errorMessage.style.display = 'none';
  errorMessage.innerText = '';

  // Check for empty inputs
  for (const key in data) {
    if (isNaN(data[key]) || data[key] < 0) {
      errorMessage.innerText = 'Please enter valid numbers for all fields.';
      errorMessage.style.display = 'block'; // Show the error message
      return; // Exit the function if there's an invalid input
    }
  }

  fetch('/predict1', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  })
  .then(response => response.json())
  .then(data => {
    const resultModal = document.getElementById('resultModal');
    const resultContent = document.getElementById('resultContent');

    // Display the prediction result in the modal
    resultContent.innerText = `Prediction: ${data.prediction}`;
    resultModal.style.display = 'flex';
  })
  .catch(error => console.error('Error:', error));
}
```

Cette fonction JavaScript envoie des données de formulaire à un serveur pour obtenir une prédiction.

Elle récupère les valeurs saisies par l'utilisateur, les valide (en s'assurant qu'elles sont numériques et positives), puis envoie les données au serveur via une requête POST.

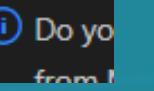
Si les données sont valides, elle affiche la prédiction dans une fenêtre modale. En cas d'erreur (entrée invalide ou problème de requête), un message d'erreur est montré à l'utilisateur.

Check API Status:

```
// Function to check the API status
function checkApiStatus() {
    // Show the modal and reset status message
    document.getElementById('resultModal').style.display = 'flex';
    const statusResult = document.getElementById('statusResult');
    statusResult.innerText = 'Checking status...';
    statusResult.classList.remove("status-online", "status-offline");

    // Send GET request to status endpoint
    fetch('/status')
        .then(response => {
            if (response.ok) {
                statusResult.innerHTML = '<p>API is Online :The system is <strong class="status-online">healthy</strong> and operational.</p>';
                statusResult.classList.add("status-online");
            } else {
                statusResult.innerHTML = '<p>API is Offline :The system is <strong class="status-offline">currently down</strong>.</p>';
                statusResult.classList.add("status-offline");
            }
        })
        .catch(() => {
            statusResult.innerHTML = '<p>The system is <strong class="status-offline">currently down</strong>.</p>';
            statusResult.classList.add("status-offline");
        });
}

function closeModal() {
    document.getElementById('resultModal').style.display = 'none';
}
```



Cette fonction vérifie l'état de l'API en envoyant une requête GET à l'endpoint /status. Si l'API est en ligne, elle affiche un message indiquant que le système est "healthy" et opérationnel, en appliquant un style spécifique. Si l'API est hors ligne ou si la requête échoue, un message d'erreur est affiché. Le tout se fait dans une fenêtre modale pour informer l'utilisateur de l'état du système.

Conteneurisation avec Docker

- Principe
- Script de Conteneurisation



Prédiction de Prix et de Stade

- La conteneurisation est une technologie qui permet de regrouper une application et toutes ses dépendances dans un environnement isolé appelé conteneur. Cela garantit une exécution cohérente sur n'importe quel système.
- **Avantages de Docker**
- **Portabilité** : Fonctionne partout où Docker est installé.
- **Isolation** : Chaque application est isolée, évitant les conflits.
- **Efficacité** : Moins de ressources utilisées qu'une machine virtuelle.



Créer le fichier requirements.txt : Ce fichier doit inclure toutes les bibliothèques nécessaires à l'exécution de l'application Flask.

```
Cancer_Predecture_App-main > └── requirements.txt
1   Flask
2   numpy
3   joblib
4   scikit-learn
5   pandas
6
```

Ce Dockerfile permet de créer une image Docker pour exécuter une application Flask avec toutes ses dépendances dans un environnement isolé.

```
1 # Use a lightweight Python image
2 FROM python:3.10-slim-buster
3
4 # Set working directory
5 WORKDIR /app
6
7 COPY requirements.txt requirements.txt
8 # Install dependencies
9
10 RUN pip3 install -r requirements.txt
11
12 COPY . .
13
14 # Start the Flask application on the required port
15 CMD ["python3", "app.py"]
16 |
```

- **FROM python:3.10-slim-buster** : Utiliser une image de base Python 3.10 légère.
- **WORKDIR /app** : Définir le répertoire de travail /app dans le conteneur.
- **COPY requirements.txt requirements.txt** : Copier le fichier requirements.txt dans le conteneur.
- **RUN pip3 install -r requirements.txt** : Installer les dépendances depuis requirements.txt.
- **COPY . .** : Copier tout le code source dans le conteneur.
- **CMD ["python3", "app.py"]** : Lancer l'application Flask avec python3 app.py.

Déploiement sur GCP

- Principe
- Script de déploiement



Script de déploiement

- **Étape 1 : Construire l'image Docker**

Naviguez vers le répertoire du serveur où se trouve votre fichier Dockerfile.

```
cd server
```

Construisez l'image Docker. Remplacez your-image-name par le nom de l'image que vous souhaitez créer.

```
docker build -t your-image-name
```

- **Étape 2 : Taguer l'image Docker**

Taguez l'image Docker pour Google Container Registry (GCR). Remplacez YOUR_PROJECT_ID par l'ID de votre projet Google Cloud.

```
docker tag your-image-name gcr.io/YOUR_PROJECT_ID/your-image-name
```

Script de déploiement

- **Étape 3 : Pousser l'image Docker vers Google Container Registry**

Authentifiez-vous auprès de Google Cloud :

```
gcloud auth login
```

Définissez votre projet :

```
gcloud config set project YOUR_PROJECT_ID
```

Poussez l'image Docker vers Google Container Registry :

```
docker push gcr.io/YOUR_PROJECT_ID/your-image-name
```

- **Étape 4 : Déployer l'image Docker sur Google Cloud Run**

Déployez l'image sur Cloud Run. Cette commande va créer un nouveau service nommé your-service-name à partir de l'image Docker. Remplacez le nom du service et celui de l'image selon vos besoins.

```
gcloud run deploy your-service-name --image gcr.io/YOUR_PROJECT_ID/your-image-name --platform managed
```

Script de déploiement

- **Étape 5 : Accéder à votre application déployée**

Une fois le déploiement terminé, vous recevrez une URL où votre application de prédiction du cancer sera hébergée. Vous pouvez y accéder en ouvrant cette URL dans votre navigateur

- **Nettoyage:**

Déployez l'image sur Cloud Run. Cette commande va créer un nouveau service nommé `your-service-name` à partir de l'image Docker. Remplacez le nom du service et celui de l'image selon vos besoins.

```
gcloud run services delete your-service-name --platform managed
```

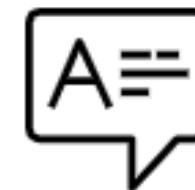
Simulation du site web

[https://cancer-prediction-app-
486638708818.europe-west1.run.app/](https://cancer-prediction-app-486638708818.europe-west1.run.app/)



Empowering Early Detection

Our AI-driven tool assists healthcare professionals in accurately identifying breast cancer, determining the tumor type, and assessing the stage of cancer. It also provides insights into potential treatment options and associated costs in the United States. Together, we can make a difference in early cancer detection and care.

[Start Prediction](#)

Looking For Answers?

Not knowing is scary. Start your research here and get help along the way.

[About Breast Cancer](#)

AI-Powered

Predict breast cancer probability with reliable, AI-driven insights for better outcomes.

[About Our Models](#)

Support The Cause

We need your help. From donations to volunteering, check Fondation Iala Salma.

[Get Involved](#)

Breast Cancer Prediction.

[Home](#)[Predict](#)[Status](#)[Team](#)

Predict Cancer Stage & Treatment Cost

Use the Elastic Net and Random Forest models to predict cancer stage and estimate treatment costs.

[Go to Prediction](#)

Choose one

Predict Tumor Type

Use the Elastic Net model to predict whether a breast tumor is benign or malignant.

[Go to Prediction](#)

Predict Tumor Type

Enter Tumor Characteristics

Detect signs of cancer

Clump Thickness
5

Uniformity of Cell Size
4

Uniformity of Cell Shape
4

Marginal Adhesion
5

Single Epithelial Cell Size
7

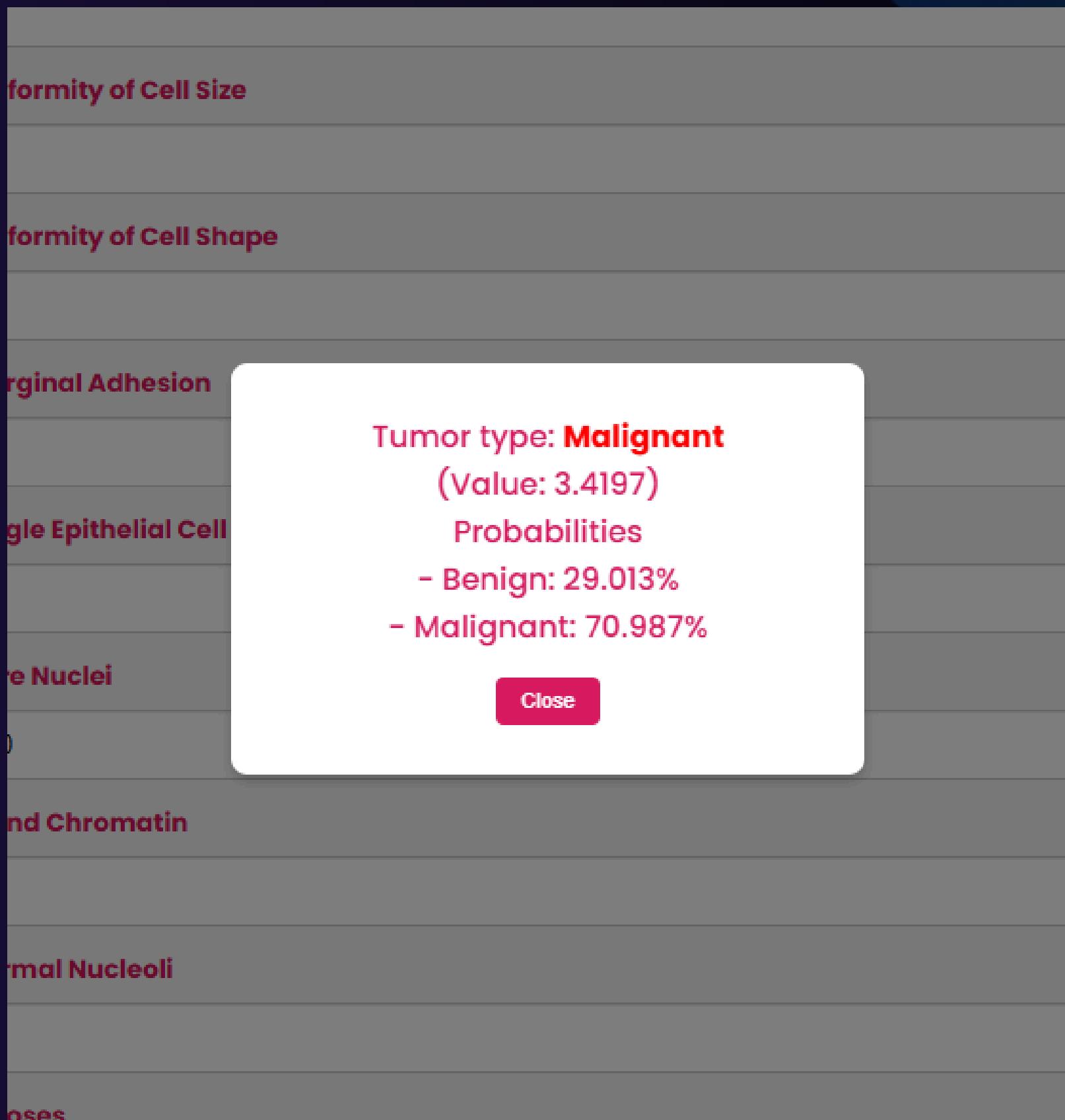
Bare Nuclei
10

Bland Chromatin
3

Normal Nucleoli
2

Mitoses
1

Predict



Predict Cancer stage & costs

Enter Tumor Characteristics

Detect signs of cancer

Clump Thickness
5

Uniformity of Cell Size
4

Uniformity of Cell Shape
4

Marginal Adhesion
5

Single Epithelial Cell Size
7

Bare Nuclei
10

Bland Chromatin
3

Normal Nucleoli
2

Mitoses
1

Predict

4

Uniformity of Cell Shape
4

Marginal Adhesion
5

Single Epithelial Cell Size
7

Bare Nuclei
10

Bland Chromatin
3

Normal Nucleoli
2

Mitoses
1

Treatment Cost : \$120190.917

Cancer stage : 3

[Close](#)

Detect signs of cancer

ion.

Home

BREAST CANCER DIAGNOSIS FEATURES

- **Clump Thickness:** Refers to the thickness of cell clusters. Higher values may indicate the formation of cell clumps, which can help detect signs of cancer.
- **Uniformity of Cell Size:** Cancerous cells often vary in size. This feature measures how uniform the sizes of the cells are, aiding in identifying possible malignancies.
- **Uniformity of Cell Shape:** Similar to cell size, cancerous cells often vary in shape. This feature assesses the consistency in the shape of the cells, assisting in the detection of abnormal growth.
- **Marginal Adhesion:** In healthy tissue, cells stick to each other. A decrease in adhesion may be a sign of malignant tissue, where cells lose their normal adhesion properties.
- **Single Epithelial Cell Size:** Measures the size of individual epithelial cells, which tend to be larger when signs of cancer are present.
- **Bare Nuclei:** Refers to the presence of nuclei without the surrounding cytoplasm. Bare nuclei are more commonly found in malignant tumors, helping to distinguish between benign and malignant cases.
- **Bland Chromatin:** Chromatin is the material that makes up chromosomes. In cancerous cells, chromatin can appear less structured, helping detect

About our models

Breast Cancer Prediction.

Home

Predict

Status

Team

Elastic Net Regressor for tumor type Prediction

The **Elastic Net Regressor** is a versatile machine learning model that combines the strengths of both L1 (Lasso) and L2 (Ridge) penalties, making it well-suited for various prediction tasks in healthcare. In our project, we applied Elastic Net to predict whether a breast tumor is benign or malignant, improving the accuracy and robustness of the prediction by addressing multicollinearity and avoiding overfitting.

How It Works

1. **L1 Penalty (Lasso):** Shrinks coefficients of less important features to zero, enabling feature selection.
2. **L2 Penalty (Ridge):** Shrinks the coefficients uniformly, helping to reduce model complexity.
3. **Combination:** Elastic Net balances bias and variance, offering more generalization power.



Accuracy

83%

Combining L1 and L2 penalties provides higher accuracy in predicting whether a tumor is benign or malignant.



Robustness

Effectively handles multicollinearity and prevents overfitting, leading to a more reliable model.



Versatility

Elastic Net is useful for feature selection in complex datasets, making it ideal for breast cancer diagnosis tasks.

Elastic Net Implementation in a Jupyter Notebook

Check API Status

System Health Status

Check API Status

API is Online :The system is healthy and operational.

Close

The system is currently down.

Close

Meet Our team



Imad Agjoud

ML/AI Engineer



WASSEF ARAGOU

front end developer



MOUAD MALIH

back end developer



Conclusion



Le déploiement de l'application de prédiction du cancer sur Google Cloud Platform (GCP) avec Docker et Cloud Run offre une solution évolutive et accessible. En utilisant des modèles de machine learning tels que l'ElasticNet pour prédire le coût et le type de tumeur, et le Random Forest Classifier pour le stade du cancer, l'application fournit des résultats précis. Docker et Cloud Run simplifient la gestion des déploiements et assurent une scalabilité et une disponibilité continues, démontrant l'efficacité de ces technologies pour créer des applications de santé intelligentes à grande échelle.

Thank You!