Detect Thyroid Cancer Reoccurrence using patient data.

Background and Problem Statement

In healthcare, predicting the likelihood of a patient's relapse is a critical task, as it directly influences the treatment and care plan provided to the patient. Early prediction of relapse can help healthcare professionals take proactive measures to mitigate the risks and improve patient outcomes.

The aim of this project is to develop a machine learning model that can predict whether a patient is likely to experience a relapse, based on a variety of medical and demographic features. The dataset provided contains information such as age, medical history, test results, and other features relevant to the prediction of relapse. This project involves analyzing the dataset, preprocessing the data, training multiple machine learning models, and evaluating their performance.

The models being evaluated include Random Forest, XGBoost, Support Vector Machine (SVM), Gradient Boosting, and Logistic Regression. This project will focus on building a reliable model with a high degree of accuracy and interpretability that can be used for future relapse prediction in a clinical setting.

Step 1: Data Preprocessing and Exploration

The first stage of the project involves exploring the dataset. The goal is to understand the data, identify any inconsistencies such as missing values or duplicates, and check the structure of the data. This allows for effective cleaning and preparation of the data, which is a crucial step for building a good model.

The categorical variables need to be encoded properly to be usable by machine learning algorithms, and numerical features will be standardized. We also remove any unnecessary columns that do not contribute to the prediction task.

Example Code for Data Exploration and Cleaning:

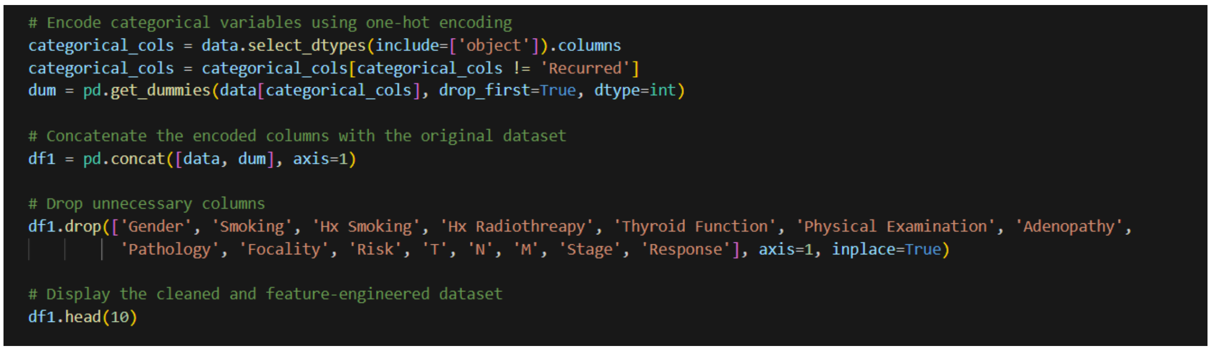
```python
import pandas as pd
import numpy as np

# Load the dataset
data = pd.read_csv("dataset.csv")

# Display the first few rows of the dataset
data.head()

# Check for missing values
data.isnull().sum()

# Check for duplicates
data_duplicate = data.copy()
data_duplicate.drop_duplicates(inplace=True)

# Identify numerical columns
numerical_cols = data.select_dtypes(include=['int64', 'float64']).columns
print("Numerical Columns:", numerical_cols)
```

After cleaning, the next step is to handle missing values and duplicates. If any missing data is present, it can be imputed or removed. Duplicates, if found, are also dropped.

Step 2: Feature Engineering

Once the data is cleaned, we move to feature engineering. This involves transforming the data into a format that is suitable for machine learning. Categorical columns are encoded using one-hot encoding, which converts them into binary vectors that machine learning models can understand. Additionally, numerical features are scaled using standardization to ensure that all features are on a similar scale, which improves the performance of many machine learning models.

Example Code for Feature Engineering:



```python
# Encode categorical variables using one-hot encoding
categorical_cols = data.select_dtypes(include=['object']).columns
categorical_cols = categorical_cols[categorical_cols != 'Recurred']
dum = pd.get_dummies(data[categorical_cols], drop_first=True, dtype=int)

# Concatenate the encoded columns with the original dataset
df1 = pd.concat([data, dum], axis=1)

# Drop unnecessary columns
df1.drop(['Gender', 'Smoking', 'Hx Smoking', 'Hx Radiothreapy', 'Thyroid Function', 'Physical Examination', 'Adenopathy',
          'Pathology', 'Focality', 'Risk', 'T', 'N', 'M', 'Stage', 'Response'], axis=1, inplace=True)

# Display the cleaned and feature-engineered dataset
df1.head(10)
```

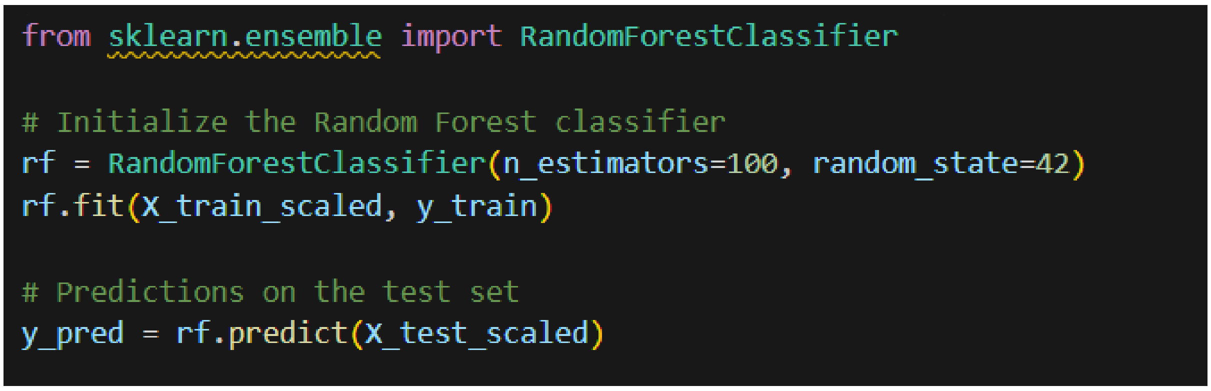
Here, we encode the categorical columns and remove features that do not contribute to the prediction task.

Step 3: Model Training and Testing

With the cleaned and prepared data, the next step is model building. Multiple machine learning algorithms are employed to predict whether a patient will experience a relapse. These algorithms include Random Forest, XGBoost, Support Vector Machine (SVM), Gradient Boosting, and Logistic Regression. We split the dataset into training and testing sets, with 30% allocated for testing, and 70% for training.

Each model is trained on the training data and then evaluated on the test data.

Example Code for Model Training:



```python
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train_scaled, y_train)

# Predictions on the test set
y_pred = rf.predict(X_test_scaled)
```

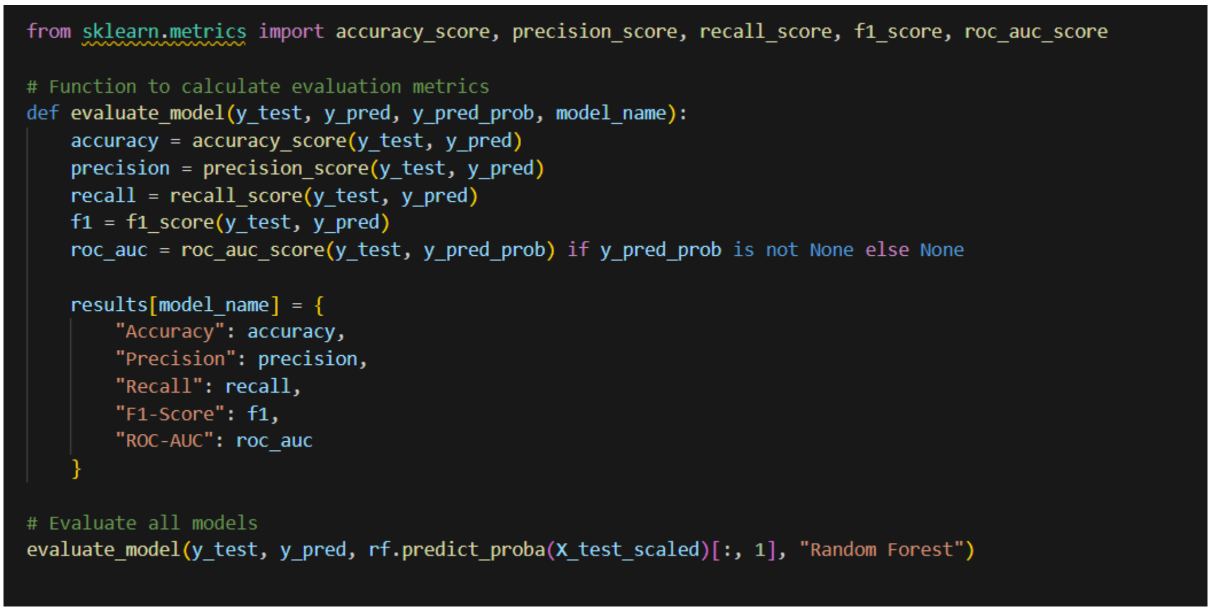
We train all five models and store the predictions for the evaluation step.

Step 4: Model Evaluation

To evaluate the performance of the models, we use multiple metrics: Accuracy, Precision, Recall, F1-Score, and ROC-AUC. These metrics give us insights into how well each model performs, especially in terms of identifying both relapse and non-relapse cases.

Accuracy is the percentage of correct predictions, Precision evaluates how many of the predicted relapses were true, Recall tells us how many of the actual relapses were predicted correctly, and the F1-Score provides a balance between Precision and Recall. ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) helps in understanding the trade-off between sensitivity and specificity.

Example Code for Model Evaluation:

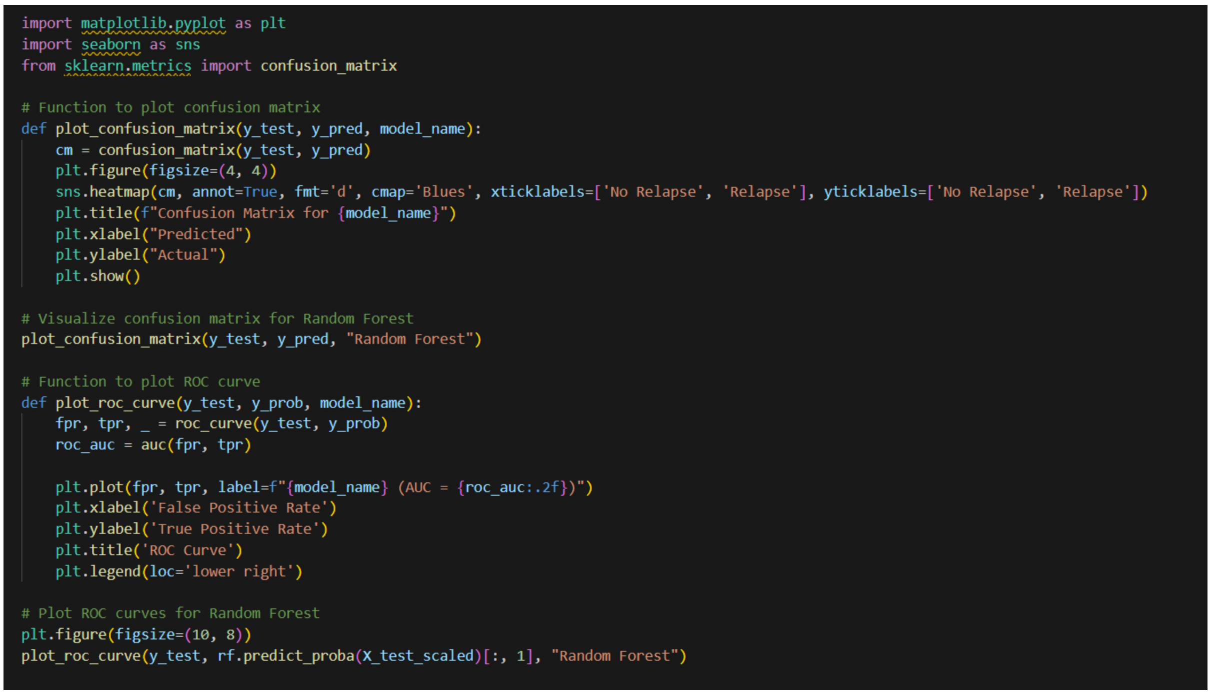


```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Function to calculate evaluation metrics
def evaluate_model(y_test, y_pred, y_pred_prob, model_name):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred_prob) if y_pred_prob is not None else None

    results[model_name] = {
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1-Score": f1,
        "ROC-AUC": roc_auc
    }

# Evaluate all models
evaluate_model(y_test, y_pred, rf.predict_proba(X_test_scaled)[:, 1], "Random Forest")
```

Step 5: Visualization

Visualizations are created to help us interpret the model results better. The confusion matrix is plotted to show the number of correct and incorrect predictions for each class (relapse vs. non-relapse). Additionally, ROC curves are plotted for each model to visualize how well each model performs across different thresholds of classification.

Example Code for Visualization:



```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Function to plot confusion matrix
def plot_confusion_matrix(y_test, y_pred, model_name):
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Relapse', 'Relapse'], yticklabels=['No Relapse', 'Relapse'])
    plt.title(f"Confusion Matrix for {model_name}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

# Visualize confusion matrix for Random Forest
plot_confusion_matrix(y_test, y_pred, "Random Forest")

# Function to plot ROC curve
def plot_roc_curve(y_test, y_prob, model_name):
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, label=f"{model_name} (AUC = {roc_auc:.2f})")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc='lower right')

# Plot ROC curves for Random Forest
plt.figure(figsize=(10, 8))
plot_roc_curve(y_test, rf.predict_proba(X_test_scaled)[:, 1], "Random Forest")
```

These visualizations give a clear picture of how well the models are performing, and which models are best suited for the task of relapse prediction.

Conclusion and Next Steps

The evaluation metrics and visualizations provide a clear picture of the models' performance. Based on the results, we can conclude which model is the most effective for predicting relapse. The Random Forest and XGBoost models, with high ROC-AUC and F1-Score, are considered the best candidates.

These models can now be deployed in a clinical setting to help healthcare professionals identify patients at risk of relapse and take preventive measures. Additionally, further hyperparameter tuning could be conducted to improve model performance.

Future steps could also include integrating these models into a user-friendly application that allows doctors to input patient data and receive relapse predictions in real-time.