A Machine Learning Framework for Early Asthma Diagnosis

John Ali, Matthew Yoshida, Rebecca Haile, William Stern, Ty Wolber

Motivation:

Asthma is a chronic lung condition that impacts roughly 300 million people worldwide across all ages, genders, and ethnic groups. Disadvantaged communities tend to have higher rates of the disease, highlighting the structural inequality to healthcare access. Implementing proactive strategies to address this issue will enhance health coverage while also easing the economic burden of disease on the U.S. government. Each member of the team has friends or family affected by asthma, reinforcing the importance of the critical health issue. This project's goal is to create predictive models for early-onset asthma using social and socioeconomic factors in hopes to improve patient quality of life and reduce healthcare costs. We implement four machine learning algorithms: Logistic Regression for simple comprehension of potentially linear patterns, CART for capturing non-linear patterns, XGBoost for handling any missing data, and Random Forests for handling complex patterns. The dataset used in the models includes anonymized patient demographics, medical history, and environmental factors.

Data Cleaning:

The dataset (Figure 1) is composed of patient health records with features such as age, BMI, smoking habits, environmental exposures, and medical history. The target variable is an indicator for if the patient has asthma or not. The data was sourced from a Kaggle repository, processed to remove irrelevant features, like Patient ID and DoctorInCharge, and clinical and biological features, like Wheezing, Eczema, LungFunctionFVC, and HistoryOfAllergies and standardized using a scalar for model training. Missing values were removed and categorical variables were encoded to enhance the models' performance. Class balance and feature relationships were explored too in the datasets to ensure effective preprocessing.

Age	Gender	Ethnicity	EducationLevel	Smoking	PhysicalActivity	DietQuality	PollutionExposure	FamilyHistoryAsthma
0.965740	-0.986710	0.334986	-1.455673	-0.406355	-1.432099	0.160113	0.809355	1.523884
-0.747054	1.013469	1.349273	0.771363	-0.406355	0.291269	0.453069	-1.036866	-0.656218
0.687989	-0.986710	1.349273	-0.342155	-0.406355	0.581330	1.434458	-1.210374	1.523884
-0.098970	1.013469	1.349273	-0.342155	-0.406355	-1.256398	0.276233	-1.509757	-0.656218
0.873156	-0.986710	-0.679301	1.884880	-0.406355	-0.154081	-0.651625	-1.373822	-0.656218

Model Selection:

To select features for model creation we used Variance Inflation Factor (VIF) and Principal Component Analysis (PCA). Features with a VIF below 5 were retained, resulting in 20 features. Our team suspected certain variables played a larger role than others in asthma diagnosis, prior to retaining social features our team wanted to check if this was the case. We applied PCA to address this, but the scree plot appeared linear, suggesting that all features contributed significantly to the model. Biological and clinical features were removed to place larger emphasis on social and socioeconomic related features in the models.

Procedures:

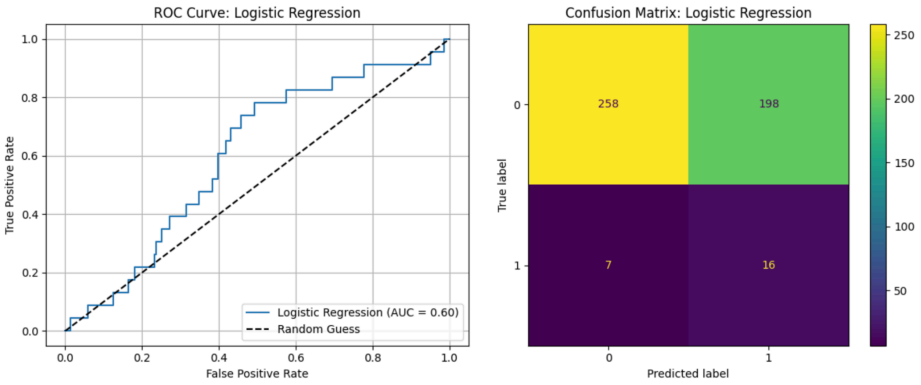
1. *Logistic Regression*

We decided to build a Logistic Regression model as it is the standard approach for binary classification tasks. It models the probability of the target variable (asthma diagnosis) being 1 (asthma) as a function of the independent variables, using a logistic function. Given the nature of our dataset and the binary classification problem, Logistic Regression was an appropriate starting point for our analysis.

Training Procedure:

The first step was to standardize the features using StandardScalar. This was necessary as logistic regression is sensitive to the scale of the input features. After scaling, we trained the LR model using the LogisticRegression function from sklearn using the class_weight='balanced' parameter to address the

imbalanced target variable, Diagnosis (456 - non-asthmatic, 23 - asthmatic). The IV's were all other features in the dataset based on the initial feature selection steps.



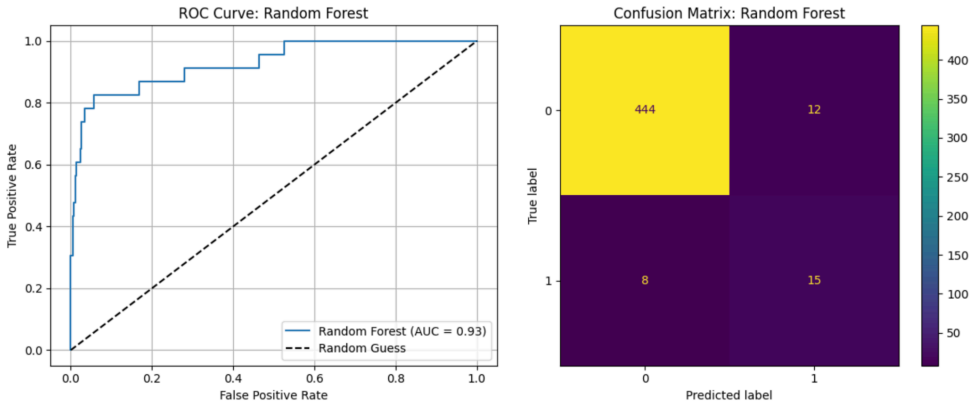
Sensitivity, specificity, F1-score and ROC curve were used as performance metrics for all models. They are all relevant in working with a classification problem, in particular sensitivity and F1-score allow us to evaluate the model's performance on the minority class, making it perfect for imbalance datasets like the one we are working with. The logistic regression got a sensitivity of 0.70 and specificity of 0.57, indicating that the model predicted the majority class more accurately, and the matrix reflects this. AUC was 0.6, so slightly better than random guessing, but has room for improvement. F1 for class 0 was 0.72, reflecting strong ability to identify non asthma cases, but F1 for class 1 was 0.14 showing that the model struggled with detecting minority cases due to imbalance in the dataset. To assess generalizability, K-fold was performed with 5 folds, and yielded a mean accuracy of 49.75% with SD of 0.0484. This showed very little variability in performance across different folds meaning the model generalizes well to unseen data, but the low accuracy indicates as our other metrics did that the model is weak.

2. *Random Forest Classifier*

The results from Logistic Regression led us to next build a Random Forests Classifier that combines multiple decision trees to improve classification performance. We hoped to capture an unseen complex relationship between features and target variables not found in Logistic Regression.

Training Procedure:

Similar to the logistic regression model, we standardized the features, and trained the model using the RandomForestClassifier from sklearn with class_weight='balanced', max_depth=4 to handle the imbalance dataset and to minimize the chance of overfitting.



The RF model achieved sensitivity of 0.65 and specificity of 0.97, indicating that the model has a tendency to predict the majority class. The confusion matrix provides a visual as to why sensitivity is much lower than specificity due to a very large imbalance in the data set. The AUC under the ROC curve (0.93) indicates that the model does much better than the baseline at every decision threshold. The high F1-score for 0 (0.98) and the lower F1-score for 1(0.6) indicates that the RF model struggles to work with the imbalance dataset even with the class weights and max_depth inputs. K-fold was performed with 5 folds yielding a mean accuracy of 89.76% with a SD

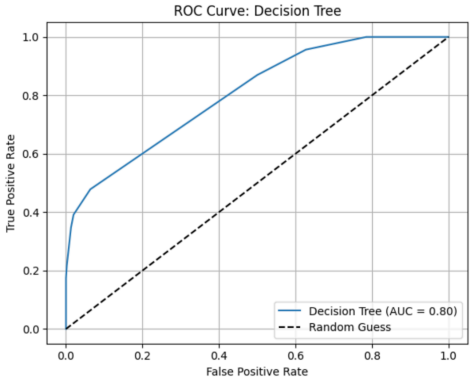
of 0.0206 indicating stable performance across folds, though the model still faces imbalance challenges, the higher mean accuracy indicates that RF did better than logistic regression.

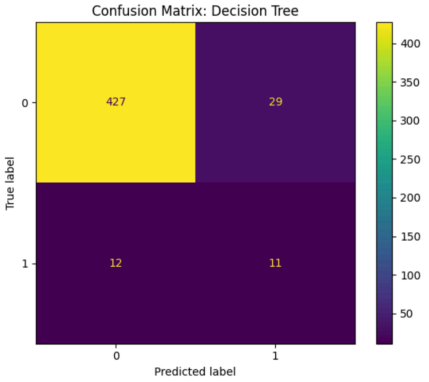
3. *Decision Tree (CART)*

The results from the previous models were great at handling the majority class, but struggled with the minority class, so we decided to choose CART as the next step as it might handle it differently by recursively splitting the data based on feature thresholds.

Training Procedure:

The model was trained using the DecisionTreeClassifier( ), the inputs class_weight='balanced', max_depth=6 were used to handle the imbalance dataset and to reduce the chance of overfitting. CART handled the imbalance dataset worse than RF with a sensitivity of 0.48 and a specificity of 0.93. The AOC of the ROC curve is 0.8 which is worse than the RF model. The F1-score provides comparable insights as the other performance metrics; the F1-score for 0 was 0.95 and 0.35 for 1. Overall the CART model was able to predict non-asthmatic status, the majority class, much better than asthmatic status.



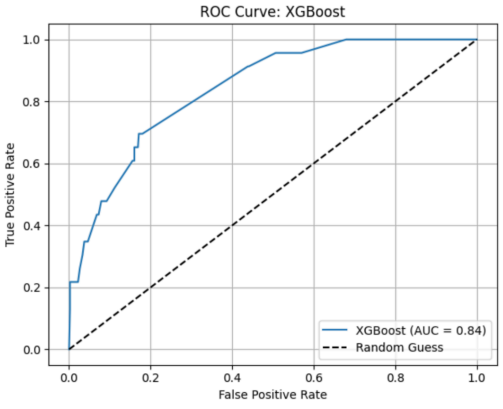


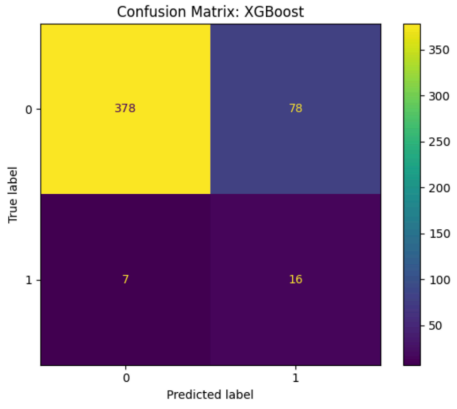
4. *XGBoost*

Finally, we chose XGBoost as it builds trees sequentially, where each new tree corrects the errors made by the previous one. For this reason, it was selected for its speed and efficiency with structured data, as well as its robust performance in predictive tasks.

Training Procedure:

The model was trained using the XGBClassifier( ) from XGBoost library, with a gamma of 25 for maximizing AUC and minimizing overfitting. As with the other tree based models, no feature scaling was required. XGBoost once again handles feature selection and identifies key predictors like LungFunction and BMI as important for the classification. The model was again evaluated using the same metrics as before. It got a sensitivity score of 0.7 and specificity of 0.83 showing the model correctly predicted class 0 but struggled with class 1. AUC was 0.84, so XGBoost performs worse than the RF model but better than CART in that regard.





Results:

Despite our models being more complex than the baseline that always predicts majority class 0 (non-asthma), the results suggest several models

Logistic Regression	0.95	0.14
Random Forests	0.95	0.6
Decision Tree	0.88	0.35
XGBoost	0.95	0.35

achieve accuracy comparable to the baseline's 95%. Throughout the debugging process we found that complex models do not always outperform the baseline, specifically in edge cases with significant class imbalance. Our test set contained far more cases of non-asthma than asthma, which limits the ability of the more complex models to surpass baseline's accuracy performance. The same logic can be applied to all models except CART which underperformed with an accuracy of 88%, suggesting it might have overfit the data or also simply struggled with the class imbalance. This behavior reinforces the need to use metrics beyond accuracy alone, as it can dilute the minority class predictions. This highlights the challenge posed by imbalanced datasets in model evaluation. Sensitivity and specificity are better performance metrics in this case, providing insights into true asthma predictions while reducing bias for non-asthma cases.

Discussion/Synthesis:

Our research developed a machine learning framework to predict early-onset asthma using social factors, comparing four predictive models: Logistic Regression, Random Forest, Decision Tree (CART), and XGBoost. Early on in our classification attempts, we found that our dataset may have a class imbalance which resulted in all our models beside CART having the exact same accuracy. In order to get a better understanding of the model performances we decided to dig deeper in specificity and sensitivity metrics. Logistic Regression established a baseline with moderate sensitivity of 0.70 and lower specificity of 0.57, demonstrating initial insights into feature relationships. Random Forest improved specificity to 0.97 but reduced sensitivity to 0.65, showcasing the trade-offs inherent in ensemble methods. The Decision Tree model offered granular data analysis but struggled with the lowest sensitivity of 0.47, while XGBoost achieved perfect specificity of 1.00 but critically low sensitivity of 0.09. Given these results, logistic regression may be the most suitable of these models with how balanced it is. Models such as Random Forest and XGboost show a high accuracy but have a hard time detecting true positives which may be detrimental in a medical setting. These results underscore the critical challenge in medical machine learning: balancing accurate detection of minority cases with overall predictive performance. In our case our classes were highly imbalanced which resulted in us having to dig deeper into the metrics such as specificity and sensitivity to get a better understanding of the model performance. However going forward we would look at ways to possibly address this model class imbalance by using techniques such as SMOTE. Moreover, we would put a heavier punishment on false negatives, as something as simple as a misclassification can cause a wide range of negative issues from panic to life altering effects or even death in some cases. Model performance is critical especially in the healthcare industry, where treatment could potentially save a life, given that placing a higher emphasis on punishing false negatives.

Impact:

This work has the potential to enhance asthma diagnosis accuracy and efficiency, enabling earlier identification of at risk individuals. This could improve disease management and elevate a patient's quality of care by supporting clinicians in interpreting complex data and prioritizing care effectively. This work is highly beneficial to government agencies and public health departments in allocating resources accurately. To enhance this model's utility, future efforts could include additional variables such as genetic predispositions or air quality indices to improve predictive power. Leveraging longitudinal data could help detect early patterns of asthma development, while tailoring a model to specific regions or demographics can enhance reliability and equity. Although, variations in environmental exposures, differing socioeconomic backgrounds, and access to healthcare, may affect its impact across subpopulations, introducing potential bias. Additionally, false negatives could delay treatment, while false positives might lead to unnecessary interventions. Underrepresentation in the training data also risks less accurate predictions for certain groups, potentially worsening health disparities. Mitigating these issues requires validation across diverse populations and regular data updates which ensures that the model complements clinical expertise.

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay, classification_report, f1_score, mean_absolute_error, mean_squared_error, roc_curv
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import warnings
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.decomposition import PCA
from sklearn.model_selection import KFold

warnings.filterwarnings("ignore")
```

In [ ]:
```
data = pd.read_csv('https://raw.githubusercontent.com/wstern1234/IE142/main/asthma_disease_data.csv')
data.head(10)
```

Out[ ]:

	PatientID	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	PhysicalActivity	DietQuality	SleepQuality	...	LungFunctionFEV1	LungFunctionFVC	Wheezing	ShortnessOfBreath
0	5034	63	0	1	0	15.848744	0	0.894448	5.488696	8.701003	...	1.369051	4.941206	0	0
1	5035	26	1	2	2	22.757042	0	5.897329	6.341014	5.153966	...	2.197767	1.702393	1	0
2	5036	57	0	2	1	18.395396	0	6.739367	9.196237	6.840647	...	1.698011	5.022553	1	1
3	5037	40	1	2	1	38.515278	0	1.404503	5.826532	4.253036	...	3.032037	2.300159	1	0
4	5038	61	0	0	3	19.283802	0	4.604493	3.127048	9.625799	...	3.470589	3.067944	1	1
5	5039	21	0	2	0	21.812975	0	0.470044	1.759118	9.549262	...	2.328191	5.898515	1	0
6	5040	45	1	1	1	30.245954	1	9.371784	7.030507	5.746128	...	2.995100	1.701512	1	1
7	5041	26	0	0	1	26.048416	1	8.344096	1.626484	6.431179	...	2.069343	4.012260	1	0
8	5042	49	1	1	2	32.676204	0	2.690256	3.920034	5.843645	...	1.761242	5.190931	1	1
9	5043	45	1	1	1	29.910298	0	2.895720	2.607700	7.234908	...	2.848420	5.771022	1	0

10 rows × 29 columns

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2392 entries, 0 to 2391
Data columns (total 29 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   PatientID              2392 non-null   int64  
 1   Age                    2392 non-null   int64  
 2   Gender                 2392 non-null   int64  
 3   Ethnicity              2392 non-null   int64  
 4   EducationLevel         2392 non-null   int64  
 5   BMI                    2392 non-null   float64
 6   Smoking                2392 non-null   int64  
 7   PhysicalActivity       2392 non-null   float64
 8   DietQuality            2392 non-null   float64
 9   SleepQuality           2392 non-null   float64
 10  PollutionExposure      2392 non-null   float64
 11  PollenExposure         2392 non-null   float64
 12  DustExposure           2392 non-null   float64
 13  PetAllergy             2392 non-null   int64  
 14  FamilyHistoryAsthma    2392 non-null   int64  
 15  HistoryOfAllergies     2392 non-null   int64  
 16  Eczema                 2392 non-null   int64  
 17  HayFever               2392 non-null   int64  
 18  GastroesophagealReflux 2392 non-null   int64  
 19  LungFunctionFEV1       2392 non-null   float64
 20  LungFunctionFVC        2392 non-null   float64
 21  Wheezing               2392 non-null   int64  
 22  ShortnessOfBreath      2392 non-null   int64  
 23  ChestTightness         2392 non-null   int64  
 24  Coughing               2392 non-null   int64  
 25  NighttimeSymptoms      2392 non-null   int64  
 26  ExerciseInduced        2392 non-null   int64  
 27  Diagnosis              2392 non-null   int64  
 28  DoctorInCharge         2392 non-null   object 
dtypes: float64(9), int64(19), object(1)
memory usage: 542.1+ KB
```

In [ ]:
```
data.dropna()
data.drop_duplicates(inplace=True)
```

In [ ]:
```
data.head()
```

Out[ ]:

	PatientID	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	PhysicalActivity	DietQuality	SleepQuality	...	LungFunctionFEV1	LungFunctionFVC	Wheezing	ShortnessOfBreath
0	5034	63	0	1	0	15.848744	0	0.894448	5.488696	8.701003	...	1.369051	4.941206	0	0
1	5035	26	1	2	2	22.757042	0	5.897329	6.341014	5.153966	...	2.197767	1.702393	1	0
2	5036	57	0	2	1	18.395396	0	6.739367	9.196237	6.840647	...	1.698011	5.022553	1	1
3	5037	40	1	2	1	38.515278	0	1.404503	5.826532	4.253036	...	3.032037	2.300159	1	0
4	5038	61	0	0	3	19.283802	0	4.604493	3.127048	9.625799	...	3.470589	3.067944	1	1

5 rows × 29 columns

In [ ]:
```
unique_counts = data.nunique()
print(unique_counts)
```

```
PatientID                2392
Age                        75
Gender                      2
Ethnicity                   4
EducationLevel              4
BMI                      2392
Smoking                     2
PhysicalActivity         2392
DietQuality              2392
SleepQuality             2392
PollutionExposure        2392
PollenExposure           2392
DustExposure             2392
PetAllergy                  2
FamilyHistoryAsthma         2
HistoryOfAllergies          2
Eczema                      2
HayFever                    2
GastroesophagealReflux      2
LungFunctionFEV1         2392
LungFunctionFVC          2392
Wheezing                    2
ShortnessOfBreath           2
ChestTightness              2
Coughing                    2
NighttimeSymptoms           2
ExerciseInduced             2
Diagnosis                   2
DoctorInCharge              1
dtype: int64
```

In [ ]:
```
data.columns
```

Out[ ]:
```
Index(['PatientID', 'Age', 'Gender', 'Ethnicity', 'EducationLevel', 'BMI',
       'Smoking', 'PhysicalActivity', 'DietQuality', 'SleepQuality',
       'PollutionExposure', 'PollenExposure', 'DustExposure', 'PetAllergy',
       'FamilyHistoryAsthma', 'HistoryOfAllergies', 'Eczema', 'HayFever',
       'GastroesophagealReflux', 'LungFunctionFEV1', 'LungFunctionFVC',
       'Wheezing', 'ShortnessOfBreath', 'ChestTightness', 'Coughing',
       'NighttimeSymptoms', 'ExerciseInduced', 'Diagnosis', 'DoctorInCharge'],
      dtype='object')
```

In [ ]:
```
data = data.drop(['PatientID', 'DoctorInCharge'], axis=1)
```

In [ ]:
```
correlation_matrix = data.corr()
target_correlation = correlation_matrix['Diagnosis'].sort_values(ascending=False)
target_correlation
```

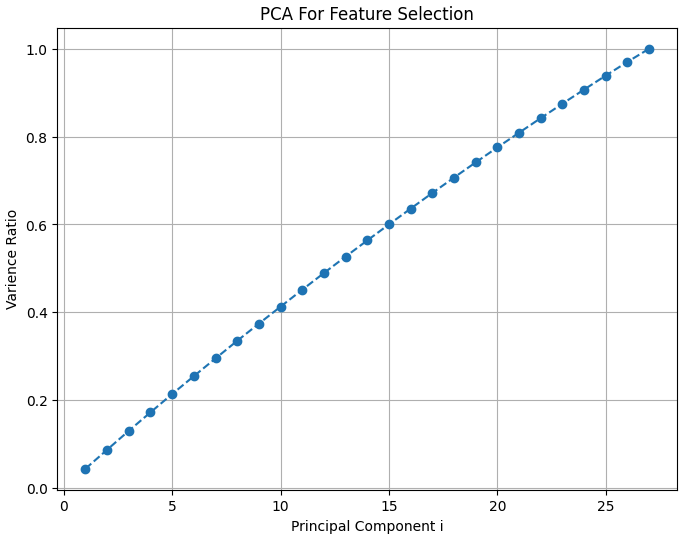
Out[ ]:

	Diagnosis
Diagnosis	1.000000
ExerciseInduced	0.053956
LungFunctionFVC	0.029629
Wheezing	0.027197
LungFunctionFEV1	0.023336
GastroesophagealReflux	0.022770
SleepQuality	0.018022
Ethnicity	0.017124
PollenExposure	0.015099
EducationLevel	0.008185
PhysicalActivity	0.005066
Gender	0.003128
FamilyHistoryAsthma	-0.001334
HistoryOfAllergies	-0.001951
DietQuality	-0.003149
PollutionExposure	-0.004535
Eczema	-0.008592
BMI	-0.012522
PetAllergy	-0.013078
Age	-0.015111
ShortnessOfBreath	-0.015281
HayFever	-0.019141
Smoking	-0.019321
NighttimeSymptoms	-0.021965
Coughing	-0.024193
DustExposure	-0.025972
ChestTightness	-0.039278

dtype: float64

In [ ]:
```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

pca = PCA()
pca_components = pca.fit_transform(data_scaled)

explained_variance = pca.explained_variance_ratio_
cumulative_variance = explained_variance.cumsum()

#scree plot to find out if some features are more important than others (prior to dropping biological features)
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(explained_variance) + 1), cumulative_variance, marker='o', linestyle='--')
```

```
plt.xlabel('Principal Component i')
plt.ylabel('Varience Ratio')
plt.title('PCA For Feature Selection')
plt.grid()
plt.show()
```



In [ ]:
```
data = data.drop(['ExerciseInduced','LungFunctionFVC','Wheezing','LungFunctionFEV1', 'GastroesophagealReflux','SleepQuality','PollenExposure','HistoryOfAllergies','Ecze
#Only keep social/socioeconomical features
```

In [ ]:
```
data = pd.get_dummies(data, drop_first=True)
data = data.replace([np.inf, -np.inf], np.nan).dropna()
data = data.drop(columns=["BMI"],axis=1)
#Features with high VIF(>5) are dropped

vif_data = pd.DataFrame()
vif_data["feature"] = data.columns

vif_data["VIF"] = [variance_inflation_factor(data.values, i) for i in range(len(data.columns))]
vif_data
```

Out[ ]:

	feature	VIF
0	Age	3.750296
1	Gender	1.866143
2	Ethnicity	1.439350
3	EducationLevel	2.736698
4	Smoking	1.155291
5	PhysicalActivity	3.420181
6	DietQuality	3.360988
7	PollutionExposure	3.225411
8	FamilyHistoryAsthma	1.406272
9	Diagnosis	1.052054

In [ ]:
```
data['Diagnosis'] = pd.to_numeric(data['Diagnosis'], errors='coerce')

scaler = StandardScaler()
features = data.drop('Diagnosis', axis=1)  # Excluding the target variable
scaled_features = scaler.fit_transform(features)

scaled_data = pd.DataFrame(scaled_features, columns=features.columns)
scaled_data['Diagnosis'] = data['Diagnosis']  # Adding the target variable back

scaled_data.head()
```

Out[ ]:

	Age	Gender	Ethnicity	EducationLevel	Smoking	PhysicalActivity	DietQuality	PollutionExposure	FamilyHistoryAsthma	Diagnosis
0	0.965740	-0.986710	0.334986	-1.455673	-0.406355	-1.432099	0.160113	0.809355	1.523884	0
1	-0.747054	1.013469	1.349273	0.771363	-0.406355	0.291269	0.453069	-1.036866	-0.656218	0
2	0.687989	-0.986710	1.349273	-0.342155	-0.406355	0.581330	1.434458	-1.210374	1.523884	0
3	-0.098970	1.013469	1.349273	-0.342155	-0.406355	-1.256398	0.276233	-1.509757	-0.656218	0
4	0.873156	-0.986710	-0.679301	1.884880	-0.406355	-0.154081	-0.651625	-1.373822	-0.656218	0

In [ ]:
```
# Separate features and target variable
X = scaled_data.drop('Diagnosis', axis=1)
y = scaled_data['Diagnosis']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X
```

Out[ ]:

	Age	Gender	Ethnicity	EducationLevel	Smoking	PhysicalActivity	DietQuality	PollutionExposure	FamilyHistoryAsthma
0	0.965740	-0.986710	0.334986	-1.455673	-0.406355	-1.432099	0.160113	0.809355	1.523884
1	-0.747054	1.013469	1.349273	0.771363	-0.406355	0.291269	0.453069	-1.036866	-0.656218
2	0.687989	-0.986710	1.349273	-0.342155	-0.406355	0.581330	1.434458	-1.210374	1.523884
3	-0.098970	1.013469	1.349273	-0.342155	-0.406355	-1.256398	0.276233	-1.509757	-0.656218
4	0.873156	-0.986710	-0.679301	1.884880	-0.406355	-0.154081	-0.651625	-1.373822	-0.656218
...	...	...	...	...	...	...	...	...	...
2387	0.039905	1.013469	-0.679301	0.771363	-0.406355	-0.699950	0.376978	-0.861740	-0.656218
2388	-1.117388	1.013469	-0.679301	-0.342155	-0.406355	0.259526	-0.218561	0.927074	-0.656218
2389	0.549114	-0.986710	2.363560	0.771363	-0.406355	-0.109067	1.096868	-0.755772	-0.656218
2390	0.178780	1.013469	-0.679301	0.771363	-0.406355	1.591768	0.804295	1.511361	1.523884
2391	-0.747054	1.013469	-0.679301	-1.455673	2.460904	-1.184528	0.821487	-0.606925	1.523884

2392 rows × 9 columns

In [ ]:
```
def plot_model_evaluation(model, X_test, y_test, model_name):
    y_pred_proba = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(12, 5))

    # Subplot for ROC Curve
    plt.subplot(1, 2, 1)
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve: {model_name}')
    plt.legend(loc="lower right")
    plt.grid()

    # Subplot for Confusion Matrix
    plt.subplot(1, 2, 2)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_)
    disp.plot(cmap='viridis', ax=plt.gca())
    plt.title(f'Confusion Matrix: {model_name}')

    plt.tight_layout()
    plt.show()
```

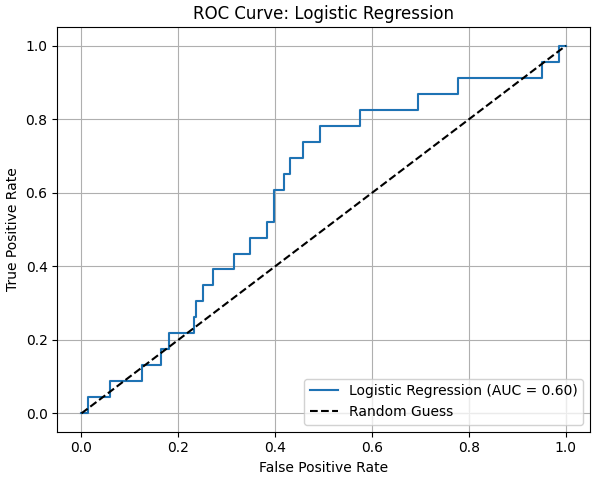
In [ ]:
```
def sensitivity(model, x_test, y_test, y_train):
    y_pred = model.predict(x_test)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    sensitivity = tp / (tp + fn)
    specificity = tn / (tn + fp)

    return sensitivity


def specificity(model, x_test, y_test, y_train):
    y_pred = model.predict(x_test)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    sensitivity = tp / (tp + fn)
    specificity = tn / (tn + fp)

    return specificity
```

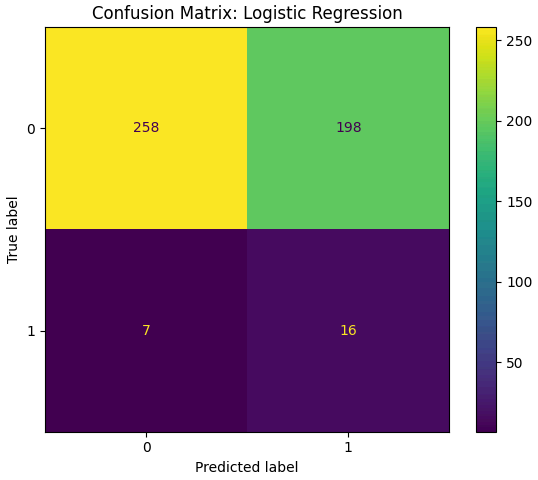
In [ ]:
```
#baseline model
print(y_test.value_counts())

#baseline model predicts zero
percentage_zeros = (y_test == 0).mean()
print(f'Accuracy: {percentage_zeros}')
```

```
Diagnosis
0    456
1     23
Name: count, dtype: int64
Accuracy: 0.9519832985386222
```

In [ ]:
```
log_reg = LogisticRegression(class_weight='balanced')

log_reg.fit(X_train, y_train)

y_pred = log_reg.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

plt.figure(figsize=(10, 8))
plot_model_evaluation(log_reg, X_test, y_test, "Logistic Regression")

print(f"\nSensitivity: {sensitivity(log_reg, X_test, y_test, y_train)}")
print(f"Specificity: {specificity(log_reg, X_test, y_test, y_train)}")
print(f"\nLogistic Regression Coefficients:\n{log_reg.coef_}\n")
print(f"\nF1 score: \n{f1_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))
```

```
<Figure size 1000x800 with 0 Axes>
```





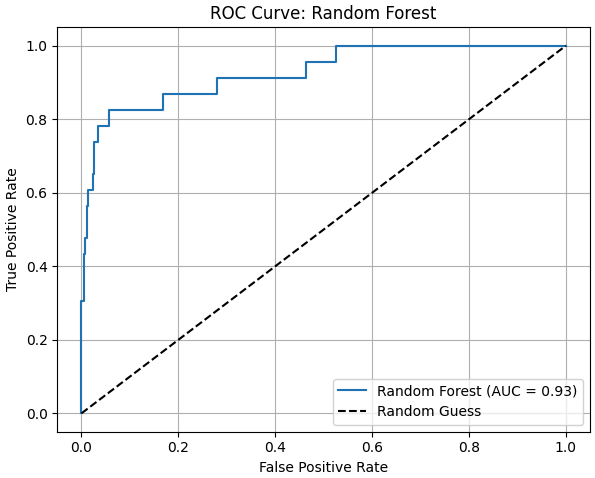

```
Sensitivity: 0.6956521739130435
Specificity: 0.5657894736842105

Logistic Regression Coefficients:
[[-0.12922326  0.20072222 -0.01307465  0.0920668  -0.04894893  0.04951638
   0.01232323 -0.04845033  0.05358788]]


F1 score:
0.1350210970464135
              precision    recall  f1-score   support

           0       0.97      0.57      0.72       456
           1       0.07      0.70      0.14        23

    accuracy                           0.57       479
   macro avg       0.52      0.63      0.43       479
weighted avg       0.93      0.57      0.69       479
```

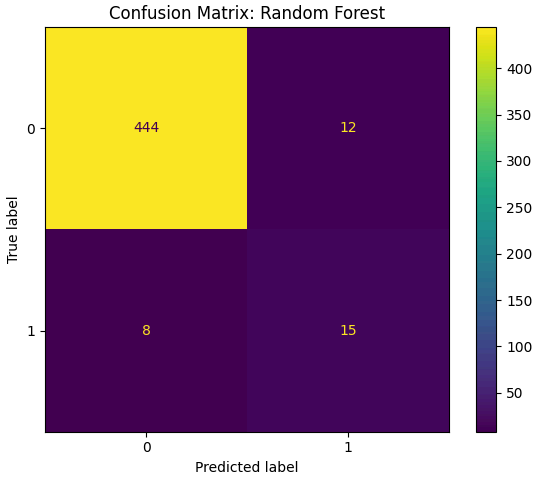
In [ ]:
```
kf = KFold(n_splits=5, shuffle=True, random_state=42)

scores = []

for train_index, val_index in kf.split(X):
    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]

    log_reg.fit(X_train, y_train)
    y_pred = log_reg.predict(X_val)

    accuracy = accuracy_score(y_val, y_pred)
    scores.append(accuracy)

scores = np.array(scores)

print(f"Cross-Validation Scores: {scores}")
print(f"Mean Accuracy: {scores.mean():.4f}")
print(f"Standard Deviation: {scores.std():.4f}")
```

```
Cross-Validation Scores: [0.58246347 0.44258873 0.5125523  0.46443515 0.48535565]
Mean Accuracy: 0.4975
Standard Deviation: 0.0484
```

In [ ]:
```
rf_classifier = RandomForestClassifier(random_state=6, class_weight='balanced', max_depth=4)

rf_classifier.fit(X_train, y_train)

y_pred_rf = rf_classifier.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)

plt.figure(figsize=(10, 8))
plot_model_evaluation(rf_classifier, X_test, y_test, "Random Forest")

print(f"F1 score: {f1_score(y_test, y_pred_rf)}")
print(f"\nSensitivity: {sensitivity(rf_classifier, X_test, y_test, y_train)}")
print(f"Specificity: {specificity(rf_classifier, X_test, y_test, y_train)}\n")
print(classification_report(y_test, y_pred_rf))
```

```
<Figure size 1000x800 with 0 Axes>
```





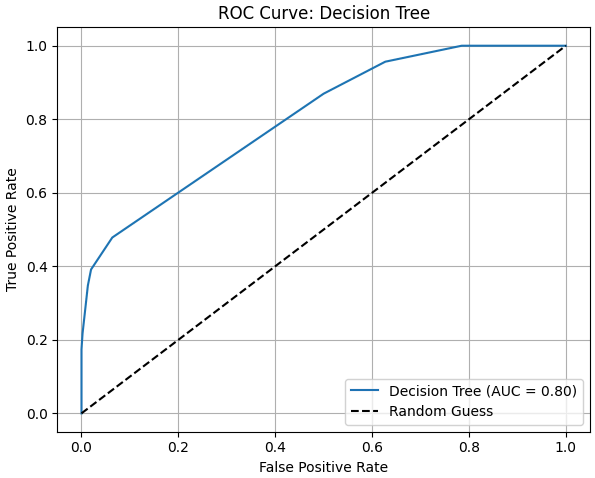

```
F1 score: 0.6

Sensitivity: 0.6521739130434783
Specificity: 0.9736842105263158

              precision    recall  f1-score   support

           0       0.98      0.97      0.98       456
           1       0.56      0.65      0.60        23

    accuracy                           0.96       479
   macro avg       0.77      0.81      0.79       479
weighted avg       0.96      0.96      0.96       479
```

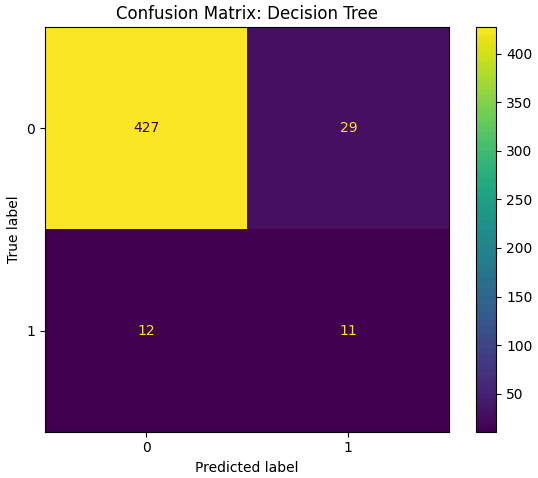
In [ ]:
```python
kf = KFold(n_splits=5, shuffle=True, random_state=42)

scores = []

for train_index, val_index in kf.split(X):
    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]

    rf_classifier.fit(X_train, y_train)
    y_pred = rf_classifier.predict(X_val)

    accuracy = accuracy_score(y_val, y_pred)
    scores.append(accuracy)

scores = np.array(scores)

print(f"Cross-Validation Scores: {scores}")
print(f"Mean Accuracy: {scores.mean():.4f}")
print(f"Standard Deviation: {scores.std():.4f}")
```

```
Cross-Validation Scores: [0.91858038 0.85803758 0.92887029 0.87866109 0.90376569]
Mean Accuracy: 0.8976
Standard Deviation: 0.0260
```

In [ ]:
```python
tree_model = DecisionTreeClassifier(random_state=42, class_weight='balanced', max_depth=6)

tree_model.fit(X_train, y_train)

y_pred = tree_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

plt.figure(figsize=(10, 8))
plot_model_evaluation(tree_model, X_test, y_test, "Decision Tree")

print(f"F1 score: {f1_score(y_test, y_pred)}")
print(f"\nSensitivity: {sensitivity(tree_model, X_test, y_test, y_train)}")
print(f"Specificity: {specificity(tree_model, X_test, y_test, y_train)}\n")
print(classification_report(y_test, y_pred))
```

```
<Figure size 1000x800 with 0 Axes>
```






```
F1 score: 0.3492063492063492

Sensitivity: 0.4782608695652174
Specificity: 0.9364035087719298

              precision    recall  f1-score   support

           0       0.97      0.94      0.95       456
           1       0.28      0.48      0.35        23

    accuracy                           0.91       479
   macro avg       0.62      0.71      0.65       479
weighted avg       0.94      0.91      0.93       479
```

In [ ]:
```
kf = KFold(n_splits=5, shuffle=True, random_state=42)

scores = []

for train_index, val_index in kf.split(X):
    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]

    tree_model.fit(X_train, y_train)
    y_pred = tree_model.predict(X_val)

    accuracy = accuracy_score(y_val, y_pred)
    scores.append(accuracy)

scores = np.array(scores)

print(f"Cross-Validation Scores: {scores}")
print(f"Mean Accuracy: {scores.mean():.4f}")
print(f"Standard Deviation: {scores.std():.4f}")
```

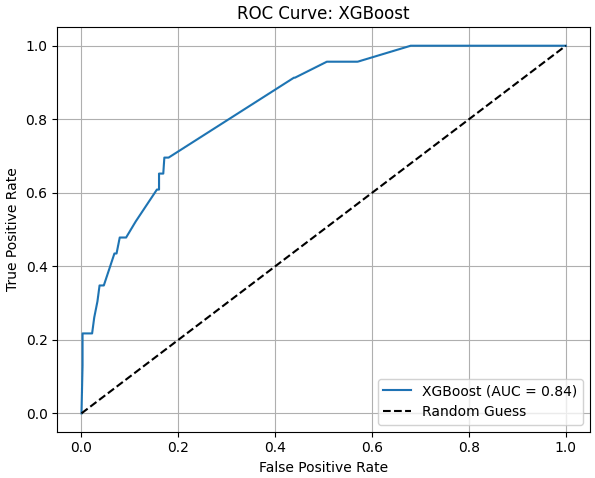
```
Cross-Validation Scores: [0.63465553 0.80584551 0.88075314 0.23640167 0.85774059]
Mean Accuracy: 0.6831
Standard Deviation: 0.2394
```

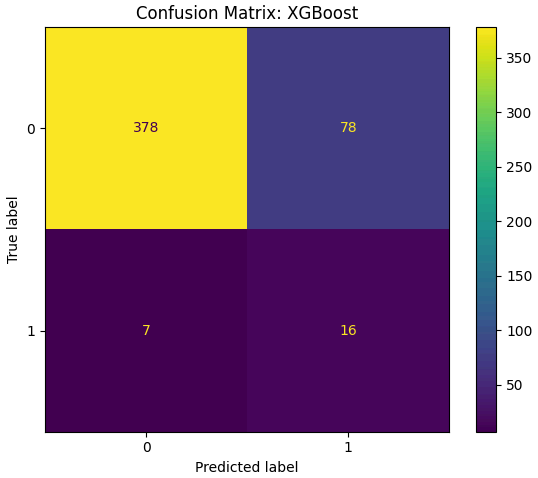
In [ ]:
```
print(type(y_test))
```

```
<class 'pandas.core.series.Series'>
```

In [ ]:
```
scaled_imbalanced = y_test.value_counts().get(0, 0) / y_test.value_counts().get(1, 0)
clf = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logloss', use_label_encoder=False, scale_pos_weight=scaled_imbalanced, gamma=25)

clf.fit(X_train, y_train)

y_pred_xgb = clf.predict(X_test)

plt.figure(figsize=(10, 8))
plot_model_evaluation(clf, X_test, y_test, "XGBoost")

accuracy = accuracy_score(y_test, y_pred_xgb)

print(f"F1 score: {f1_score(y_test, y_pred)}")
print(f"\nSensitivity: {sensitivity(clf, X_test, y_test, y_train)}")
print(f"Specificity: {specificity(clf, X_test, y_test, y_train)}\n")
print(classification_report(y_test, y_pred_xgb))
```

```
<Figure size 1000x800 with 0 Axes>
```






```
F1 score: 0.3492063492063492

Sensitivity: 0.6956521739130435
Specificity: 0.8289473684210527

              precision    recall  f1-score   support

           0       0.98      0.83      0.90       456
           1       0.17      0.70      0.27        23

    accuracy                           0.82       479
   macro avg       0.58      0.76      0.59       479
weighted avg       0.94      0.82      0.87       479
```

In [ ]:
```python
kf = KFold(n_splits=5, shuffle=True, random_state=42)

scores = []

for train_index, val_index in kf.split(X):
    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y.iloc[train_index], y.iloc[val_index]

    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_val)

    accuracy = accuracy_score(y_val, y_pred)
    scores.append(accuracy)

scores = np.array(scores)

print(f"Cross-Validation Scores: {scores}")
print(f"Mean Accuracy: {scores.mean():.4f}")
print(f"Standard Deviation: {scores.std():.4f}")
```

```
Cross-Validation Scores: [0.57620042 0.56784969 0.31171548 0.34100418 0.74267782]
Mean Accuracy: 0.5079
Standard Deviation: 0.1611
```