

## Title: Explainable AI for Communicable Disease Prediction and Sustainable Living: An Intelligent Healthcare Prototype

**Abstract:** Communicable diseases spread rapidly and can infect large populations quickly. With rising cases worldwide 🌍, early detection 🧑🏻 and continuous monitoring of patients' health is critical. This project develops an intelligent healthcare prototype 🏥 using Explainable AI 🧠 to identify risk factors for communicable diseases. The prototype monitors parameters like temperature 🌡️, oxygen saturation 🩺, etc. using medical sensors and analyzes data locally using the Explainable Stack model. Experiments demonstrate the prototype's efficiency, with the stack model (Logistic Regression, XGBoost, Random Forest 🌲) achieving 87.4% accuracy in predicting infection risk. The Explainability of Stacked model via SHAP, LIME, and ELI5 increases trustworthiness 👍.

### Introduction:

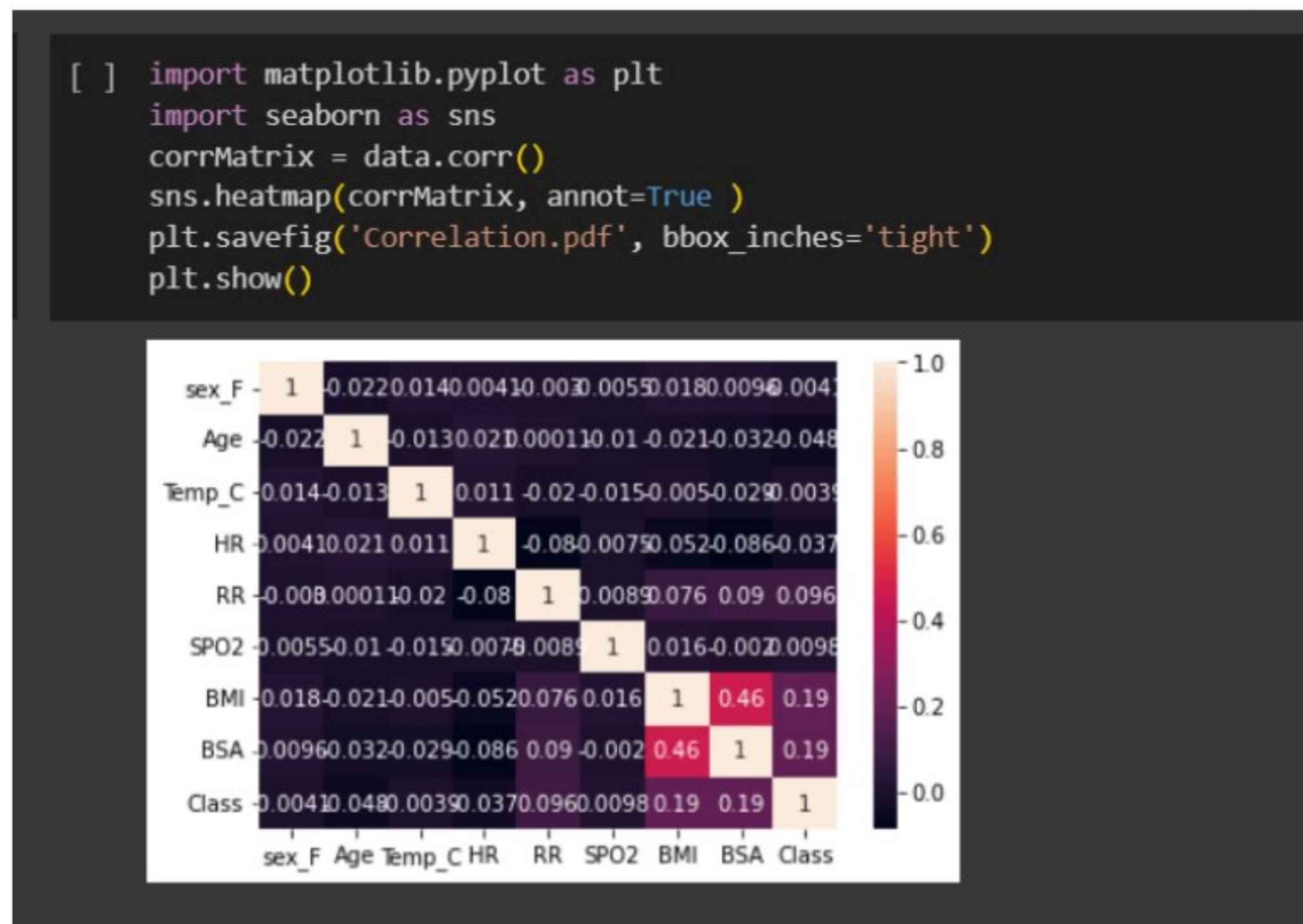
1. **Background and motivation** - Rising communicable diseases, need for early identification, remote monitoring benefits.
2. **Limitations of black-box AI models in healthcare** - Lack of trust, hard to interpret by doctors. Explainable AI can increase model understanding and accountability.
3. **Objectives and contributions of the prototype** - Continuous monitoring, XStacked Model for prediction, mobile app for tracking, edge computing for sustainability.

### Proposed Prototype and Methodology:

1. **System architecture** - Sensors, local device, edge device, cloud storage, mobile app.
2. **Data collection through sensors and pre-processing techniques:**
  - **Filtering:** The dataset is cleaned by handling missing values in the 'BMI' and 'BSA' columns. The `fill_na` function is used to replace these missing values with the mean of the respective column.
  - **Feature Selection:** Certain columns ('race\_White', 'race\_AA', 'race\_Other', 'ethnicity\_Hispanic\_YN', 'patient\_class', 'encounter\_type', 'reason\_for\_visit', 'SBP', 'DBP', and 'Month') are dropped from the dataset using the `drop` function. This is done to remove features that may not contribute to the model's performance or could introduce bias.



- **Correlation Analysis Of Health Parameters:** This analysis focuses on the correlation between health parameters for disease prediction. A correlation matrix is used to identify redundant parameters. If the correlation value between two parameters is greater than 0.55, the redundant parameter is dropped. This approach reduces redundancy and improves the efficiency of the predictive model. 👍



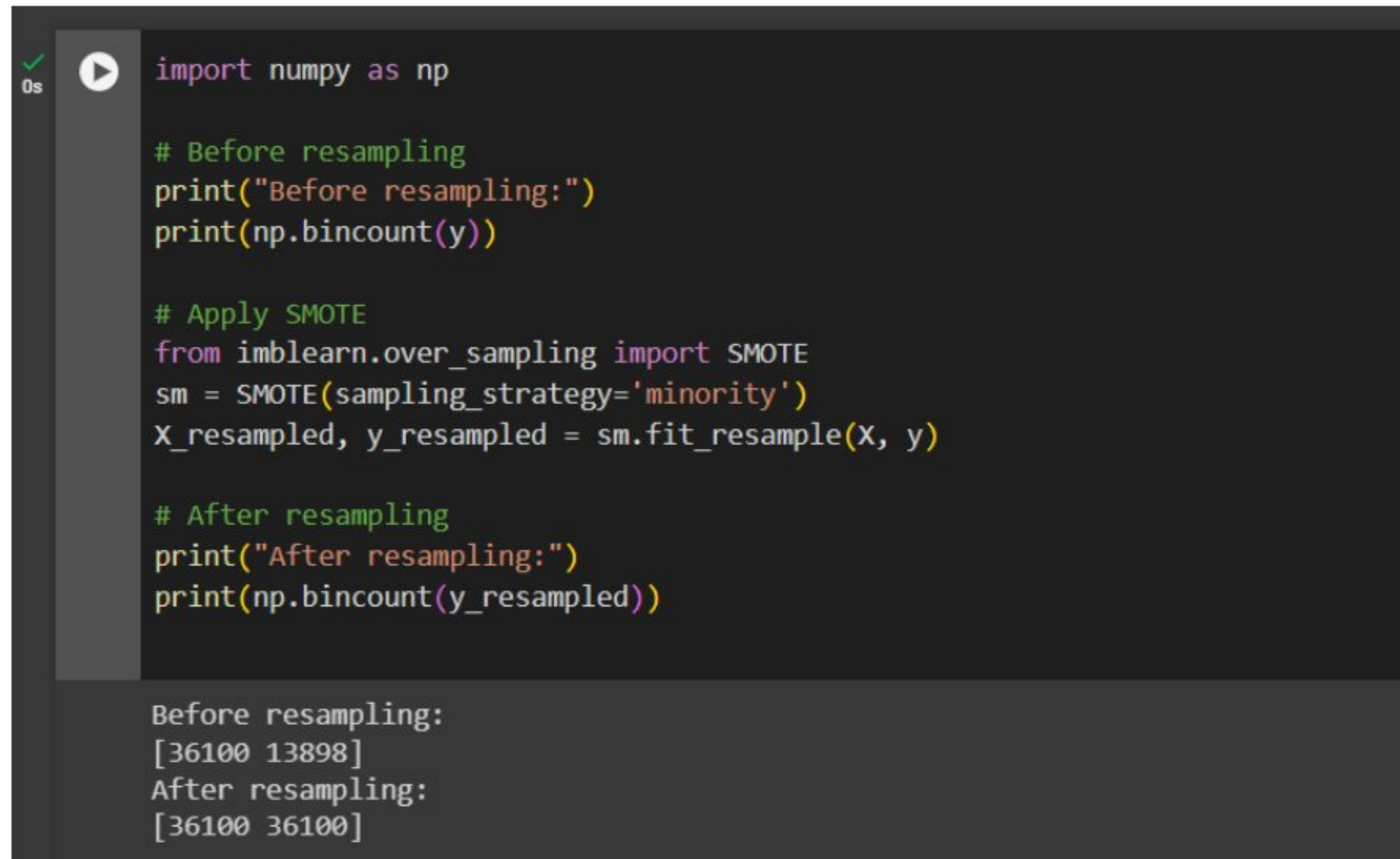
- **Outlier Removal:** The **drop\_duplicates** function is used to remove duplicate rows, ensuring each data point is unique. Additionally, specific rows (40003 and 40004) are removed from the dataset using the **drop** function, possibly to eliminate outliers

```
[ ] # Reading dataset
data = pd.read_csv('CnegvsCpos.csv')

[ ] # Preprocessing
data = data.drop(columns=['race_white', 'race_AA', 'race_Other',
                        'ethnicity_Hispanic_YN', 'patient_class',
                        'encounter_type', 'reason_for_visit', 'SBP',
                        'DBP', 'Month'])
data["BMI"].fillna(data["BMI"].mean(), inplace=True)
data["BSA"].fillna(data["BSA"].mean(), inplace=True)
data.drop_duplicates(inplace=True)
data.drop([40003, 40004], axis=0, inplace=True)
```



- **Over-sampling of Minority Classes: SMOTE (Synthetic Minority Over-sampling Technique)** is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.



```
import numpy as np

# Before resampling
print("Before resampling:")
print(np.bincount(y))

# Apply SMOTE
from imblearn.over_sampling import SMOTE
sm = SMOTE(sampling_strategy='minority')
X_resampled, y_resampled = sm.fit_resample(X, y)

# After resampling
print("After resampling:")
print(np.bincount(y_resampled))
```

Before resampling:  
[36100 13898]  
After resampling:  
[36100 36100]

### 3. Implementation of XStacked Model for Disease Prediction

The code provided is implementing a **XStacked Model**, a powerful stacked machine learning model designed for disease prediction. This model is a robust combination of three base models: **Logistic Regression**, **Random Forest Regressor**, and **XGBoost Classifier**.

The process initiates with the importation of necessary libraries and the division of resampled data into training and testing sets. Following this, the three base models are trained using the training data. Each model, equipped with its unique strengths, generates probability predictions on the test data.

These predictions are subsequently stacked together, forming the foundation for the next level of the model. A **meta-model**, which in this case is a **Random Forest Classifier**, is trained using these stacked predictions. The meta-model makes the final predictions by considering the predictions of all the base models, thereby leveraging the strengths of multiple models.

The performance of this XStacked Model is then evaluated using **accuracy**, **F1 score**, **precision**, and **recall** metrics. The results are printed for review, providing valuable insights into the model's prediction performance.



This approach of using an XStacked Model not only enhances prediction performance but also provides a robust and comprehensive method to tackle disease prediction. 👍

```
[10] # Import necessary libraries
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

# Assuming you have X_train, X_test, y_train, and y_test
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.1, random_state=0)

# Train base models
# Logistic Regression model
logreg = LogisticRegressionCV(cv=5, scoring='accuracy', max_iter=500, random_state=42)
logreg.fit(X_train, y_train)

# RandomForestRegressor model
rf = RandomForestRegressor(n_estimators=1000, max_depth=20, random_state=42, min_samples_split=5, min_samples_leaf=2)
rf.fit(X_train, y_train)

# XGBoost model
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', n_estimators=500, learning_rate=0.01, max_depth=10,
                    subsample=0.8, colsample_bytree=0.8, gamma=0.5, random_state=42)
xgb.fit(X_train, y_train)

# Make predictions (probabilities instead of hard predictions)
logreg_pred_probs = logreg.predict_proba(X_test)[:, 1]
rf_pred_probs = rf.predict(X_test)
xgb_pred_probs = xgb.predict_proba(X_test)[:, 1]

# Meta model
blender = RandomForestClassifier(n_estimators=500, max_depth=10, random_state=42)
blender.fit(np.column_stack([logreg_pred_probs, rf_pred_probs, xgb_pred_probs]), y_test)
```

#### 4. Evaluation of XStacked Model for Disease Prediction:

```
# Predictions
stack_pred_probs = blender.predict_proba(np.column_stack([logreg_pred_probs, rf_pred_probs, xgb_pred_probs]))[:, 1]
stack_pred = np.round(stack_pred_probs).astype(int) # Use np.round to get binary predictions

# Evaluate stacked model
accuracy = accuracy_score(y_test, stack_pred)
f1 = f1_score(y_test, stack_pred)
precision = precision_score(y_test, stack_pred)
recall = recall_score(y_test, stack_pred)

# Display evaluation metrics
print("Stacked Model Evaluation:")
print("Accuracy: ", accuracy)
print("F1 Score: ", f1)
print("Precision: ", precision)
print("Recall: ", recall)
```

```
Stacked Model Evaluation:
Accuracy: 0.8742382271468144
F1 Score: 0.8732551647124511
Precision: 0.8851160158460668
Recall: 0.8617079889807162
```



## 5. Explainability of Predictions

### 1. Eli5:

- Eli5 is a **Python library** that allows you to **visualize and debug** various Machine Learning models using a unified API.
- It can give explanations for any given supervised learning model by treating it as a **'black box'**.
- It provides **locally faithful explanations** around the vicinity of the instance being explained.

```
# Import ELI5
import eli5

# Interpret Meta model (RandomForestClassifier)
print("Meta Model (RandomForestClassifier) Interpretation:")
eli5.show_prediction(blender, np.column_stack([logreg_pred_probs, rf_pred_probs, xgb_pred_probs])[27],
                    feature_names=['logreg', 'rf', 'xgb'], show_feature_values=True)
```

Meta Model (RandomForestClassifier) Interpretation:  
y=1 (probability 0.575) top features

Contribution?	Feature	Value
+0.503	<BIAS>	1.000
+0.041	rf	0.607
+0.017	logreg	0.579
+0.014	xgb	0.609

### 2. DeepShap:

- DeepShap or Deep SHAP is an explainability technique that can be used for models with a **neural network-based architecture**.
- It is a **game-theoretic approach** to explain the output of any machine learning model.
- It connects optimal credit allocation with local explanations using the classic **Shapley values** from game theory and their related extensions.



✓  
7m



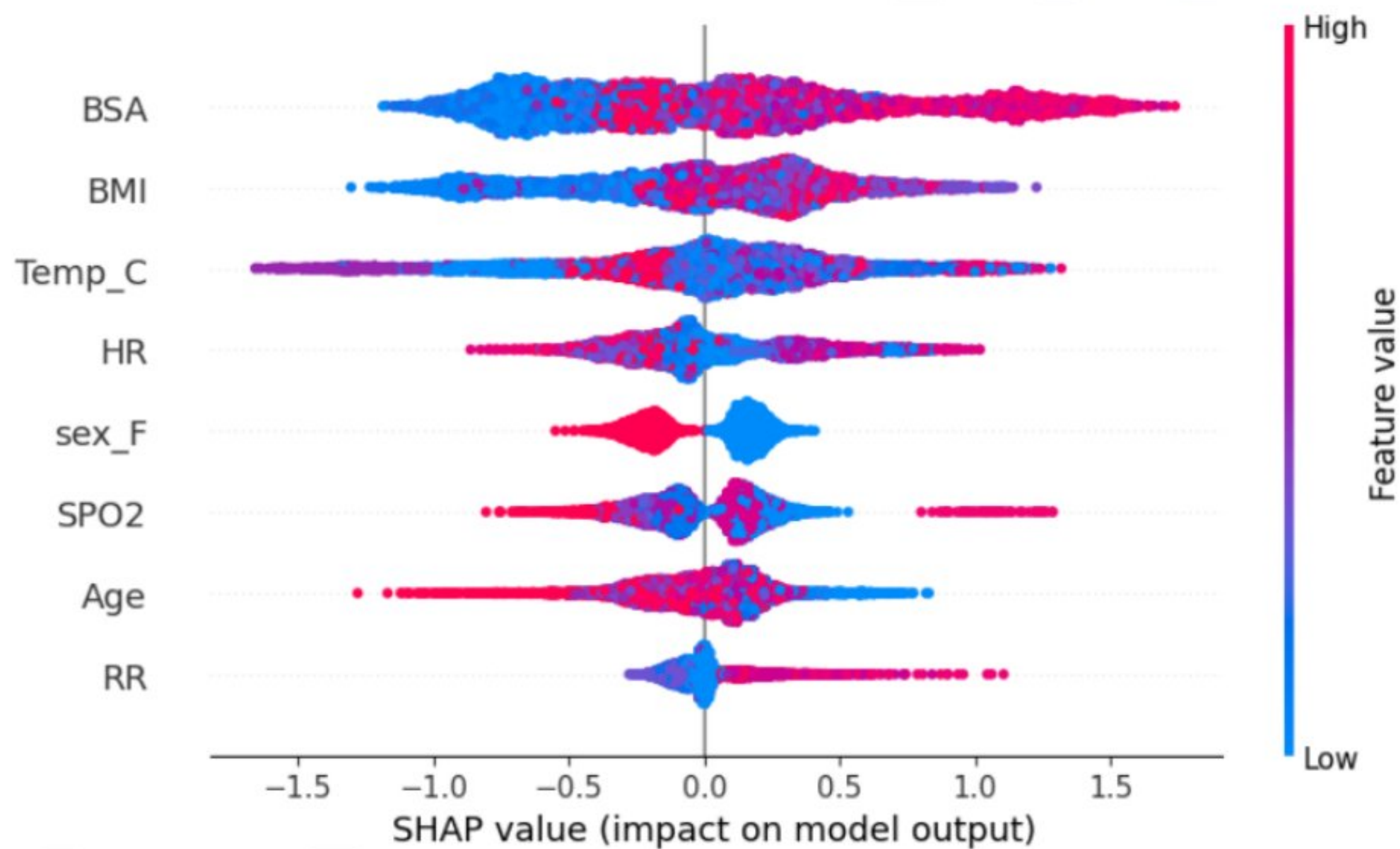
```
# Import necessary libraries
import shap

# Create a TreeExplainer for the XGBoost model
shap_explainer = shap.TreeExplainer(xgb) # Use the XGBoost model

# Calculate SHAP values
shap_values = shap_explainer.shap_values(X_test) # Use your original test data

# Plot force plot for a test instance
test_instance = X_test.iloc[27] # Use a row from your original test data
shap.force_plot(shap_explainer.expected_value, shap_values[27,:], test_instance)

# Plot summary plot
shap.summary_plot(shap_values, X_test) # Use your original test data
```



### 3. Lime:

- Lime is an explainable AI method that helps **illuminate a machine learning model** and make its predictions individually comprehensible.

- It can give explanations for any given supervised learning model by treating it as a **'black box'**.
- Local explanations mean that Lime gives explanations that are **locally faithful** within the surroundings or vicinity of the observation/sample being explained

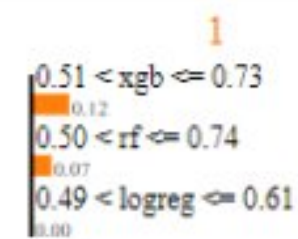
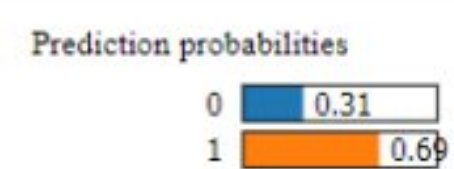
SHREEERAJ



```
[21] # Import necessary libraries
import lime
from lime.lime_tabular import LimeTabularExplainer

# Create a LimeTabularExplainer
explainer = LimeTabularExplainer(training_data=np.column_stack([logreg_pred_probs, rf_pred_probs, xgb_pred_probs]),
                                feature_names=['logreg', 'rf', 'xgb'],
                                mode='classification')

# Explain instance using LIME
exp = explainer.explain_instance(data_row=np.column_stack([logreg_pred_probs, rf_pred_probs, xgb_pred_probs])[27],
                                predict_fn=blender.predict_proba)
exp.show_in_notebook(show_table=True)
```



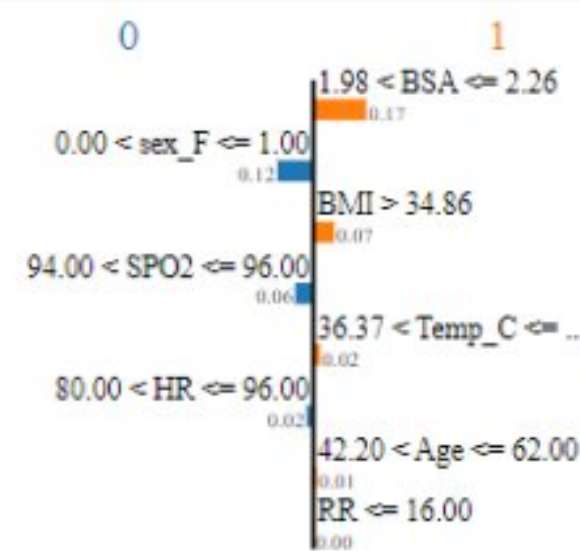
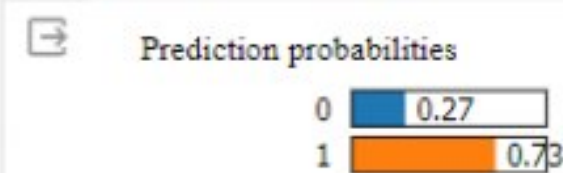
Feature Value

xgb	0.73
rf	0.72
logreg	0.58

```
# Import necessary libraries
import lime
from lime.lime_tabular import LimeTabularExplainer

# Create a LimeTabularExplainer
explainer = LimeTabularExplainer(training_data=X_train.values, # Use your original training data
                                feature_names=X_train.columns, # Use the column names from your original data
                                mode='classification')

# Explain instance using LIME
exp = explainer.explain_instance(data_row=X_test.values[27], # Use a row from your original test data
                                predict_fn=xgb.predict_proba) # Use the predict_proba method of the XGBoost model
exp.show_in_notebook(show_table=True)
```



Feature Value

BSA	2.24
sex_F	1.00
BMI	38.17
SPO2	96.00
Temp_C	36.40
HR	90.00
Age	44.00
RR	16.00



These techniques are part of a larger field known as **Explainable AI (XAI)**, which refers to methods that help explain a given AI model's decision-making process. They are particularly useful in scenarios where understanding the reasoning behind a model's prediction is crucial. For example, in healthcare, if a model predicts a patient has a certain disease, doctors need to understand why the model made that prediction before making treatment decisions.

## 6. ROC Curve:

- The **ROC (Receiver Operating Characteristic) curve** is a binary classification performance metric that depicts the model's capability to distinguish between classes at different thresholds.
- It is a **probability curve**, with the true positive rate plotted against the false positive rate for various threshold settings.
- The **Area Under the ROC curve (AUC)** is a measure of the degree of separability. It tells how much the model is capable of distinguishing between classes. The higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s.



✓  
2s

```
# Import necessary libraries
from sklearn.metrics import roc_curve, auc

# Define a result_table to store the fpr, tpr, and auc
result_table = pd.DataFrame(columns=['classifiers', 'fpr', 'tpr', 'auc'])

# Calculate the ROC curve and AUC for each model
for cls, pred_probs in zip(['Logistic Regression', 'Random Forest', 'XGBoost', 'Stacked Model'],
                           [logreg_pred_probs, rf_pred_probs, xgb_pred_probs, stack_pred_probs]):
    fpr, tpr, _ = roc_curve(y_test, pred_probs)
    auc_score = auc(fpr, tpr)
    result_table = result_table.append({'classifiers': cls,
                                       'fpr': fpr,
                                       'tpr': tpr,
                                       'auc': auc_score}, ignore_index=True)

# Set name of the classifiers as index labels
result_table.set_index('classifiers', inplace=True)

# Plot the ROC curve
fig = plt.figure(figsize=(8,6))

for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(i, result_table.loc[i]['auc']))

plt.plot([0,1], [0,1], color='orange', linestyle='--')

plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("False Positive Rate", fontsize=15)

plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)

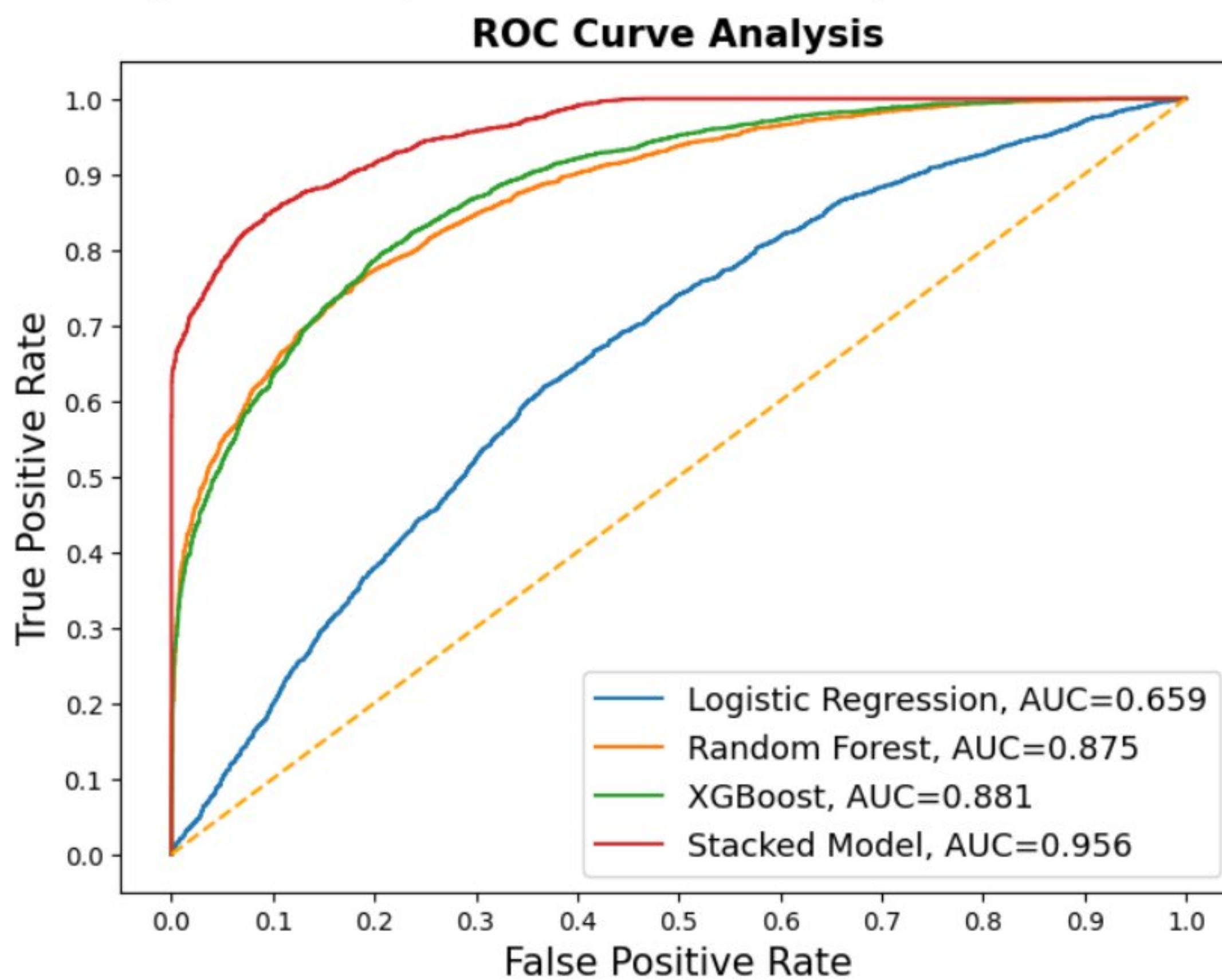
plt.title('ROC Curve Analysis', fontweight='bold', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')

plt.savefig('ROC.pdf', bbox_inches='tight')
plt.show()
```

- In the given context, the AUC-ROC curve shows that the **XStacked model outperforms the existing ML models** with an AUC of **0.956**. This is higher than the AUCs of the existing methods, indicating that the XStacked model is more capable of distinguishing between classes.
- This high AUC validates that the proposed XStacked model has the capability to identify communicable disease infection and the level of risk factor in a patient.



Remember, an ideal model has an AUC close to 1, which means it has a good measure of separability and is capable of making accurate predictions. A poor model, on the other hand, has an AUC close to 0, which means it has a low measure of separability and may make less accurate predictions. A model with an AUC of 0.5 is no better than random guessing. Therefore, the AUC is a crucial metric for evaluating the performance of a binary classification model.



Written By ([M<>ShreeRaj](#))

[Github Link Of Implementation](#)

[Linkedin](#)