

*Explainable AI*

## Predictive Modeling for Diabetes Detection using Machine Learning

### Abstract

This project aims to develop a machine learning-based predictive model for diabetes detection. The project includes interpretability analysis using LIME, Shapley, and Counterfactual explanations. Additionally, a user-friendly interface is implemented using Streamlit for easy interaction and real-time predictions..

IBA



Hammad Hadi Khan - 14278  
Hassan Hadi Khan - 12837

# Table of Contents

---

Introduction .....	3
Data preprocessing .....	4
Code Snippet: .....	4
Select Machine Learning Algorithms using PyCaret.....	5
Code Snippet: .....	5
Results: .....	5
Standard Machine Learning Algorithms with Parameter Tuning .....	8
ADA Boost Parameter Tuning.....	8
Code Snippet: .....	8
Result: .....	9
LightGBM Parameter Tuning.....	10
Code Snippet: .....	10
Result: .....	11
XGBoost Parameter Tuning .....	12
Code Snippet: .....	12
Result: .....	13
Catboost Parameter Tuning .....	14
Code Snippet: .....	14
Result: .....	15
Gradient Boosting Classifier .....	16
Code Snippet: .....	16
Result: .....	17
Ensemble Model .....	18
Code Snippet: .....	18
Result: .....	19
Neural Networks .....	20
Code Snippet: .....	20
Result: .....	21
Using LIME and SHAP with Machine Learning Models .....	22
LIME .....	22
Code Snippet (for AdaBoost only, all codes are in the python notebook) .....	22
Result: .....	22
SHAPLEY (for AdaBoost only, all codes are in the python notebook) .....	23
Code Snippet: .....	23
Result: .....	24
Counter Factual Explanations (for AdaBoost only, all codes are in the python notebook) ..	24
Code Snippet: .....	24

Result: .....	25
Streamlit WebApp.....	26
Conclusion.....	28
References .....	28

# Introduction

---

Diabetes is a serious health condition that affects millions of people around the world. It's important to diagnose and manage diabetes early to prevent complications like heart disease, stroke, and kidney failure. That's where machine learning comes in. By analyzing data and detecting patterns, machine learning algorithms can help healthcare professionals predict and diagnose diabetes more accurately.

In this project, we're using a diabetes dataset to develop a machine learning model that can predict whether a person has or is at risk of developing diabetes. The dataset includes information from 10,000 patients, such as their age, BMI, blood pressure, and glucose levels. The target variable is a binary outcome indicating whether the patient has diabetes.

Our goal is to create a machine learning model that can make accurate predictions based on the patient's input features. To do this, we'll be using preprocessing techniques to clean and prepare the data, and then training three standard machine learning algorithms to see which one performs best. We used PyCaret library to predict which model will perform best out of all the machine learning models. We'll also be building an ensemble model to combine the strengths of the individual algorithms.

To evaluate the performance of our model, we'll be using three sample scenarios with different input values. We'll then use LIME, Shapley, and Counterfactual to generate explanations for the model's predictions. Finally, we'll describe the programming interface we've built using Streamlit, which allows users to interact with the model and see its predictions in real-time.

# Data preprocessing

---

The diabetes dataset used in this project required some preprocessing to ensure the data was clean and ready for analysis. The following techniques were applied to the dataset:

**Removing records where the age of the patient is less than 2 and changing data type of age to INT:** After examining the data, it was found that some records had an age value less than 2. As this is not a realistic age for a patient, these records were removed from the dataset. Additionally, the age feature was originally stored as a string, but it was converted to an integer to make it easier to work with.

**Merging and cleaning multiple similar values in smoking history:** The smoking history feature had multiple similar values that needed to be merged to reduce redundancy. For example, "not current" was merged into category of "former", "ever" was merged into category of "current". Additionally, some values were cleaned and standardized.

**Converting BMI of people over 60:** The weight of people over 60 was sometimes recorded in pounds instead of kilograms. To ensure consistency, the weight was converted to kilograms for all patients over the age of 60.

After these preprocessing techniques were applied, the dataset was mostly clean and ready for analysis. Further, one-hot encoding was applied to avoid the issue of cardinality and converting string data to categorical data.

## Code Snip pet:

```
#removing records where age of the patient is less than 2, and changing data
type of age to INT
data2 = data2[data2['age'] >= 2]
data2['age'] = data2['age'].astype(int)

#merging and cleaning multiple similar values in smoking history
smoking_history_mapping = {'not current' : 'former', 'ever' : 'current', 'No
Info' : 'unrevealed'}
data2['smoking_history'] =
data2['smoking_history'].replace(smoking_history_mapping)
data2['smoking_history'] = data2['smoking_history'].apply(lambda x:
x.split()[0])

#converting bmi of people over 60, because the weight can be in pounds which
needs to be converted into kgs for homogeneity
bmi_validity = data2['bmi'] > 60
data2.loc[bmi_validity, 'bmi'] = data2.loc[bmi_validity, 'bmi'] / 2.205

#applying one-hot feature encoding to the data
enc_data = pd.get_dummies(data2)
```



# Select Machine Learning Algorithms using PyCaret

For this project, we used the PyCaret library to automate the machine learning process and select the best performing algorithm for our preprocessed diabetes dataset. We trained and evaluated four popular gradient boosting algorithms: Gradient Boosting Classifier, LightGBM, XGBoost, and CatBoost.

After running the Pycaret setup and comparison functions, we found that all four algorithms performed well on the diabetes dataset, with an accuracy score of over 97%. However, Gradient Boosting and LightGBM had slightly higher accuracy scores than XGBoost and CatBoost.

We then used the PyCaret tune model function to further optimize the hyperparameters of the Gradient boosting classifier model. The final optimized models had accuracy score of 97.16% which was not an improvement of original model, so we used the original model for further parametric tuning.

Overall, the gradient boosting algorithms performed well on the preprocessed diabetes dataset, with Gradient Boosting Classifier and LightGBM being the top performers. The PyCaret library provided a convenient and efficient way to compare and optimize multiple machine learning algorithms, allowing us to quickly select the best model for our dataset.

## Code Snippet:

```
from pycaret.classification import *
Pycaretmodel = setup(data2, target='diabetes')
best_model = compare_models()
tuned_model = tune_model(best_model)
```

## Results:

	Description	Value
0	Session id	4228
1	Target	diabetes
2	Target type	Binary
3	Original data shape	(97899, 9)
4	Transformed data shape	(97899, 14)
5	Transformed train set shape	(68529, 14)
6	Transformed test set shape	(29370, 14)
7	Numeric features	6
8	Categorical features	2
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	mode
13	Maximum one-hot encoding	25
14	Encoding method	None
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	34d0

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.9717	0.9784	0.6844	0.9852	0.8076	0.7928	0.8083	1.6810
ada	Ada Boost Classifier	0.9714	0.9785	0.6882	0.9748	0.8067	0.7918	0.8058	0.6770
lightgbm	Light Gradient Boosting Machine	0.9712	0.9777	0.6877	0.9717	0.8053	0.7902	0.8041	0.5380
xgboost	Extreme Gradient Boosting	0.9707	0.9766	0.6941	0.9563	0.8043	0.7889	0.8008	1.6410
catboost	CatBoost Classifier	0.9703	0.9769	0.6934	0.9517	0.8022	0.7866	0.7982	10.3540
rf	Random Forest Classifier	0.9695	0.9594	0.6881	0.9460	0.7965	0.7805	0.7921	1.0810
et	Extra Trees Classifier	0.9660	0.9509	0.6872	0.8975	0.7783	0.7602	0.7683	0.9650
lr	Logistic Regression	0.9596	0.9609	0.6289	0.8696	0.7299	0.7086	0.7195	2.4450
lda	Linear Discriminant Analysis	0.9560	0.9532	0.5916	0.8582	0.7002	0.6774	0.6911	0.2960
knn	K Neighbors Classifier	0.9528	0.8745	0.5341	0.8737	0.6628	0.6390	0.6615	0.4330
dt	Decision Tree Classifier	0.9509	0.8574	0.7430	0.7074	0.7246	0.6977	0.6980	0.3530
svm	SVM - Linear Kernel	0.9447	0.0000	0.7086	0.7166	0.6948	0.6653	0.6754	0.4330
ridge	Ridge Classifier	0.9398	0.0000	0.3097	0.9904	0.4718	0.4490	0.5360	0.2750
dummy	Dummy Classifier	0.9132	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.3080
nb	Naive Bayes	0.8762	0.9168	0.7978	0.3947	0.5281	0.4660	0.5044	0.2550
qda	Quadratic Discriminant Analysis	0.7704	0.7115	0.4694	0.1743	0.2281	0.1473	0.1813	0.2910

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
Fold							
0	0.9701	0.9715	0.6571	0.9974	0.7923	0.7769	0.7966
1	0.9713	0.9757	0.6706	0.9975	0.8020	0.7872	0.8052
2	0.9724	0.9739	0.6840	0.9975	0.8116	0.7972	0.8138
3	0.9736	0.9747	0.6975	0.9976	0.8210	0.8072	0.8223
4	0.9704	0.9750	0.6605	0.9975	0.7947	0.7795	0.7988
5	0.9711	0.9763	0.6672	1.0000	0.8004	0.7855	0.8042
6	0.9745	0.9785	0.7092	0.9953	0.8283	0.8149	0.8286
7	0.9727	0.9694	0.6874	0.9976	0.8139	0.7997	0.8159
8	0.9696	0.9725	0.6504	1.0000	0.7882	0.7726	0.7934
9	0.9701	0.9645	0.6571	0.9974	0.7923	0.7769	0.7966
Mean	0.9716	0.9732	0.6741	0.9978	0.8045	0.7898	0.8075
Std	0.0016	0.0038	0.0185	0.0013	0.0129	0.0136	0.0114

Fitting 10 folds for each of 10 candidates, totalling 100 fits  
 Original model was better than the tuned model, hence it will be returned. NOTE:  
 The display metrics are for the tuned model (not the original one).



# Standard Machine Learning Algorithms with Parameter Tuning

---

To enhance the performance of the machine learning algorithms, we performed parametric tuning using randomized search cross-validation. We focused on our four best gradient boosting algorithms: AdaBoost, LightGBM, XGBoost, Catboost, and Gradient Boosting Classifier. Moreover, we also used Neural Networks and Ensemble model with soft voting to train and predict our model.

## ADA Boost Parameter Tuning

We utilized randomized search CV to explore various combinations of hyperparameters and identify the optimal configuration for the ADA Boost model. The hyperparameters considered for tuning included learning rate, and number of estimators. The randomized search CV technique helped us efficiently search through a wide range of parameter values and find the optimal set that maximized the model's performance.

### Code Snippet:

```
# Define the AdaBoost classifier
AdaBoost = AdaBoostClassifier()

start_time = time.time()

# Define the hyperparameter grid to search over
params = {
    'n_estimators': np.arange(50, 200, 10),
    'learning_rate': np.logspace(-3, 0, 4),
}

# Define the cross-validation strategy
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Perform the randomized search with cross-validation
rs = RandomizedSearchCV(
    AdaBoost, param_distributions=params, n_iter=50, cv=cv, verbose=False,
    random_state=42,
    scoring='roc_auc' # Set AUC as the metric to optimize
)

# Fit the randomized search on the training data
rs.fit(X_train, y_train)

# Print the best hyperparameters and score
print("Best Hyperparameters:", rs.best_params_)
print("Best Score (AUC):", rs.best_score_)

# Use the best hyperparameters to train the model
best_params = rs.best_params_
model = AdaBoostClassifier(**best_params)
model.fit(X_train, y_train)

# Generate predictions for the test set
```

```

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

# Calculate the false positive rate (FPR), true positive rate (TPR), and
thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC score
auc_score = auc(fpr, tpr)

# Generate the AUC curve with color
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='green', label='AUC = {:.2f}'.format(auc_score))
plt.fill_between(fpr, tpr, color='lightgreen', alpha=0.3)
plt.plot([0, 1], [0, 1], 'k--') # Plot the random guessing line
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

end_time = time.time()
time_taken = end_time - start_time
print("Time taken: {:.2f} seconds".format(time_taken))

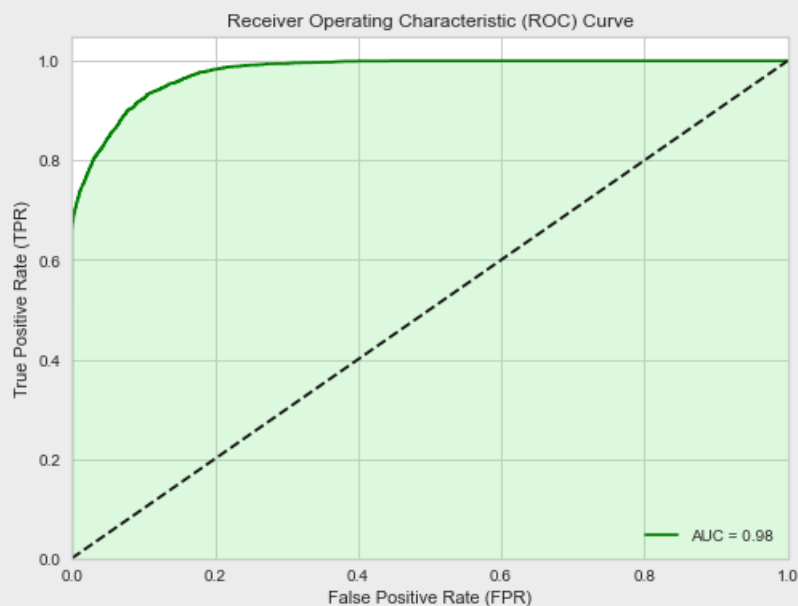
```

## Result:

Best Hyperparameters: {'n\_estimators': 70, 'learning\_rate': 1.0}

Best Score (AUC): **0.979181598381925**

Time taken: 1052.74 seconds



# LightGBM Parameter Tuning

We utilized randomized search CV to explore various combinations of hyperparameters and identify the optimal configuration for the LightGBM model. The hyperparameters considered for tuning included learning rate, maximum depth, number of leaves, minimum child samples, and others. The randomized search CV technique helped us efficiently search through a wide range of parameter values and find the optimal set that maximized the model's performance.

## Code Snippet:

```
import lightgbm as lgb
import time
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import RandomizedSearchCV, KFold

start_time = time.time()

# Define the LightGBM classifier
LGBM = lgb.LGBMClassifier()

# Define the hyperparameter grid to search over
params = {
    'num_leaves': np.arange(10, 100, 10),
    'max_depth': np.arange(3, 12, 1),
    'learning_rate': np.logspace(-3, 0, 4),
    'min_child_samples': np.arange(10, 60, 10),
    'subsample': np.arange(0.1, 1.1, 0.1),
    'colsample_bytree': np.arange(0.1, 1.1, 0.1),}

# Define the cross-validation strategy
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Perform the randomized search with cross-validation
rs = RandomizedSearchCV(
    LGBM, param_distributions=params, n_iter=50, cv=cv, verbose=False,
    random_state=42,
    scoring='roc_auc' # Set AUC as the metric to optimize)

# Fit the randomized search on the training data
rs.fit(X_train, y_train)

# Print the best hyperparameters and score
print("Best Hyperparameters:", rs.best_params_)
print("Best Score (AUC):", rs.best_score_)

# Use the best hyperparameters to train the model
best_params = rs.best_params_
model = lgb.LGBMClassifier(**best_params)
model.fit(X_train, y_train)

# Generate predictions for the test set
```

```

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

# Calculate the false positive rate (FPR), true positive rate (TPR), and
thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC score
auc_score = auc(fpr, tpr)

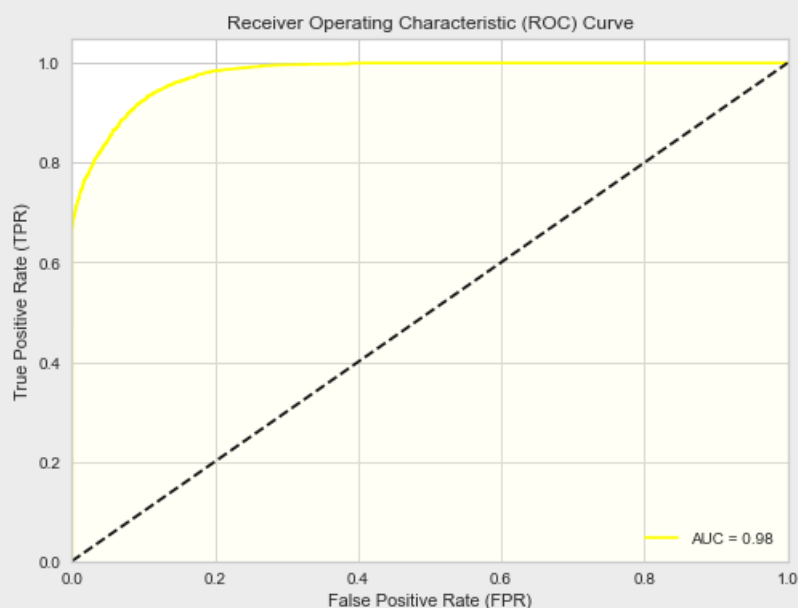
# Generate the AUC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='AUC = {:.2f}'.format(auc_score))
plt.plot(fpr, tpr, color='purple', label='AUC = {:.2f}'.format(auc_score))
plt.fill_between(fpr, tpr, color='lightblue', alpha=0.3)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

end_time = time.time()
time_taken = end_time - start_time
print("Time taken: {:.2f} seconds".format(time_taken))

```

## Result:

Best Hyperparameters: {'subsample': 1.0, 'num\_leaves': 80, 'min\_child\_samples': 20, 'max\_depth': 5, 'learning\_rate': 0.1, 'colsample\_bytree': 1.0}  
 Best Score (AUC): **0.9791296045708127**



## XGBoost Parameter Tuning

We used the randomized search CV technique to tune the parameters of the XGBoost model. The parameters included learning rate, maximum depth, and number of parameters. By searching over various combinations of these parameters, we aimed to find the configuration that maximized the accuracy of the XGBoost model on the diabetes dataset.

### Code Snippet:

```
start_time = time.time()
param_grid = {'n_estimators': [50, 100, 200, 300], 'max_depth': [2, 3, 4, 5], 'learning_rate': [0.1, 0.01, 0.001]}

xgb_model = xgb.XGBClassifier()

random_search = RandomizedSearchCV(estimator=xgb_model,
param_distributions=param_grid, n_iter=10, cv=3)

# Perform the randomized search
random_search.fit(X_train, y_train)

# Print the best parameters and the best score
print("Best parameters:", random_search.best_params_)
print("Best score:", random_search.best_score_)

# Generate predictions for the test set
y_pred_proba = random_search.predict_proba(X_test)[: , 1]

# Calculate the false positive rate (FPR), true positive rate (TPR), and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC score
auc_score = auc(fpr, tpr)

# Generate the AUC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='AUC = {:.2f}'.format(auc_score))
plt.plot(fpr, tpr, color='purple', label='AUC = {:.2f}'.format(auc_score))
plt.fill_between(fpr, tpr, color='lightblue', alpha=0.3)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

end_time = time.time()
time_taken = end_time - start_time
```



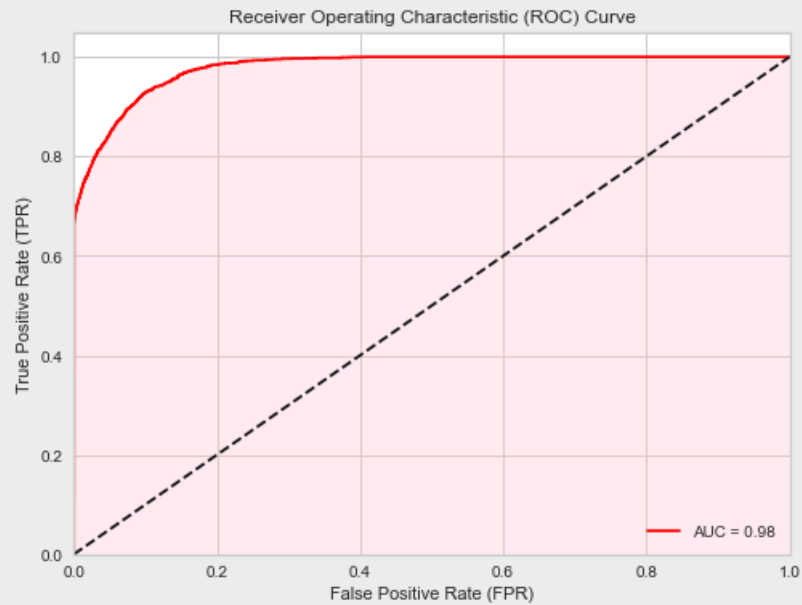
```
print("Time taken: {:.2f} seconds".format(time_taken))
```

## Result:

Best parameters: {'n\_estimators': 500, 'max\_depth': 2, 'learning\_rate': 0.1}

Best score: **0.9716762246640108**

Time taken: 32.13 seconds



## Catboost Parameter Tuning

Like LightGBM and XGBoost, we used randomized search CV to identify the optimal combination of hyperparameters for the Catboost model. The parameters considered for tuning included learning rate, maximum depth, l2 regularization, feature importance boosting type, and others. The randomized search CV allowed us to explore a wide range of parameter values and find the best configuration for the Catboost model.

### Code Snippet:

```
import catboost as cb
import time
CatB = cb.CatBoostClassifier()

start_time = time.time()

# Define the hyperparameters and their possible values for grid search
param_grid = {
    'iterations': [100, 200, 300],
    'learning_rate': [0.1, 0.01, 0.001],
    'depth': [4, 6, 8],
    'l2_leaf_reg': [1, 3, 5]}

# Perform grid search using GridSearchCV
grid_search = GridSearchCV(estimator=CatB, param_grid=param_grid, cv=3)
grid_search.fit(enc_data2, y, verbose=False)

# Get the best hyperparameter values
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
print("Best Score (AUC):", grid_search.best_score_)

# Generate predictions for the test set
y_pred_proba = model.predict_proba(X_test)[: , 1]

# Calculate the false positive rate (FPR), true positive rate (TPR), and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC score
auc_score = auc(fpr, tpr)

# Generate the AUC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='AUC = {:.2f}'.format(auc_score))
plt.plot(fpr, tpr, color='purple', label='AUC = {:.2f}'.format(auc_score))
plt.fill_between(fpr, tpr, color='lightblue', alpha=0.3)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
```

```
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

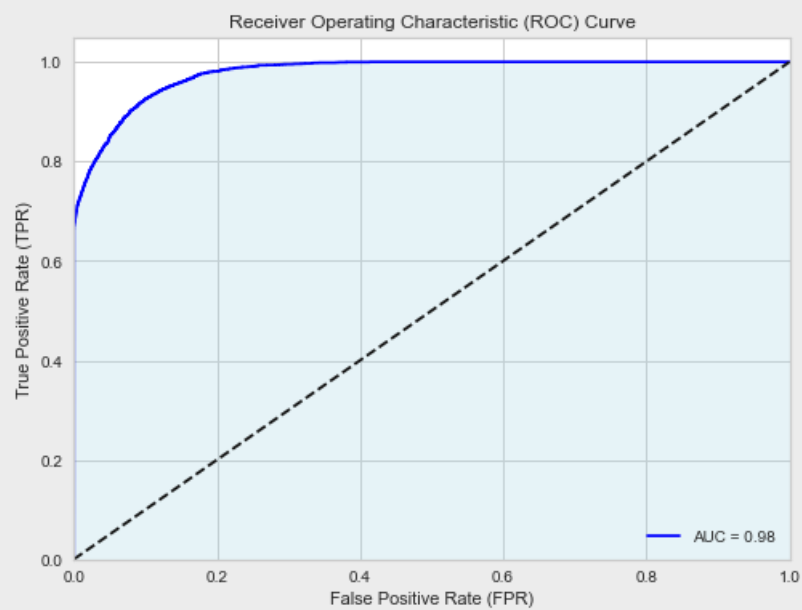
end_time = time.time()
time_taken = end_time - start_time
print("Time taken: {:.2f} seconds".format(time_taken))
```

## Result:

Best Parameters: {'depth': 4, 'iterations': 100, 'l2\_leaf\_reg': 3, 'learning\_rate': 0.1}

Best Score (AUC): **0.9715829579464549**

Time taken: 854.68 seconds



# Gradient Boosting Classifier

In addition to the boosting algorithms, we used Randomized search CV to tune the hyperparameters of the Gradient Boosting Classifier. The parameters included learning rate, maximum depth, subsample ratio, loss function, and others. By searching over different combinations of these parameters, we aimed to find the optimal configuration for the Gradient Boosting Classifier model.

## Code Snippet:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from scipy.stats import randint as sp_randint

start_time = time.time()
param_dist = {"learning_rate": [0.05, 0.1, 0.15, 0.2],
              "max_depth": sp_randint(2, 10),
              "max_features": sp_randint(2, 10),
              "n_estimators": sp_randint(100, 1000)}

# Create the classifier object
gb_clf = GradientBoostingClassifier()

# Create the randomized search object
rand_search = RandomizedSearchCV(gb_clf, param_distributions = param_dist,
n_iter=10, cv=5)

# Fit the randomized search object to the data
rand_search.fit(X_train, y_train)

# Print the best hyperparameters and best score
print("Best hyperparameters found: ", rand_search.best_params_)
print("Best Score (AUC):", rand_search.best_score_)

# Generate predictions for the test set
y_pred_proba = gb_clf.predict_proba(X_test)[: , 1]

# Calculate the false positive rate (FPR), true positive rate (TPR), and
thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC score
auc_score = auc(fpr, tpr)

# Generate the AUC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='AUC = {:.2f}'.format(auc_score))
plt.plot(fpr, tpr, color='purple', label='AUC = {:.2f}'.format(auc_score))
plt.fill_between(fpr, tpr, color='lightblue', alpha=0.3)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0, 1])
plt.ylim([0, 1.05])
```

```
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

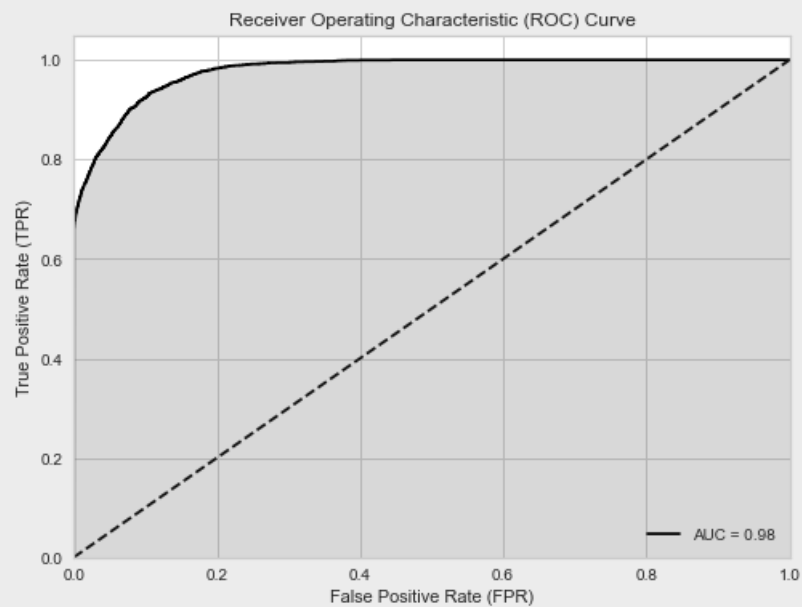
end_time = time.time()
time_taken = end_time - start_time
print("Time taken: {:.2f} seconds".format(time_taken))
```

## Result:

Best hyperparameters found: {'learning\_rate': 0.05, 'max\_depth': 3, 'max\_features': 3, 'n\_estimators': 293}

Best Score (AUC): **0.9714865098746156**

Time taken: 2293.50 seconds





# Ensemble Model

We implemented an ensemble model combining LightGBM, XGBoost, AdaBoost, Gradient Boosting Classifier, and CatBoost. We trained the ensemble model using the training data and evaluated its performance on the test data. To further analyze the model's performance, we generated an AUC curve. The ensemble model accuracy for all models combined was the least, when compared to each individual model.

## Code Snippet:

```
from sklearn.ensemble import VotingClassifier
import lightgbm as lgb
import xgboost as xgb
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from catboost import CatBoostClassifier
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Create instances of each individual model
lgb_model = lgb.LGBMClassifier()
xgb_model = xgb.XGBClassifier()
ada_model = AdaBoostClassifier()
gb_model = GradientBoostingClassifier()
cat_model = CatBoostClassifier()

# Define the ensemble model by combining the individual models using the
VotingClassifier
ensemble_model = VotingClassifier(estimators=[('lgb', lgb_model), ('xgb',
xgb_model), ('ada', ada_model), ('gb', gb_model), ('cat',
cat_model)], voting='soft')

# Train the ensemble model
ensemble_model.fit(X_train, y_train)

# Make predictions using the ensemble model
y_pred = ensemble_model.predict(X_test)

# Calculate and print the accuracy of the ensemble model
accuracy = accuracy_score(y_test, y_pred)
print("Ensemble Model Accuracy:", accuracy)

# Get the predicted probabilities for the positive class from the ensemble
model
y_pred_proba = ensemble_model.predict_proba(X_test)[: , 1]

# Calculate the false positive rate (FPR), true positive rate (TPR), and
thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the AUC score
auc_score = auc(fpr, tpr)
```

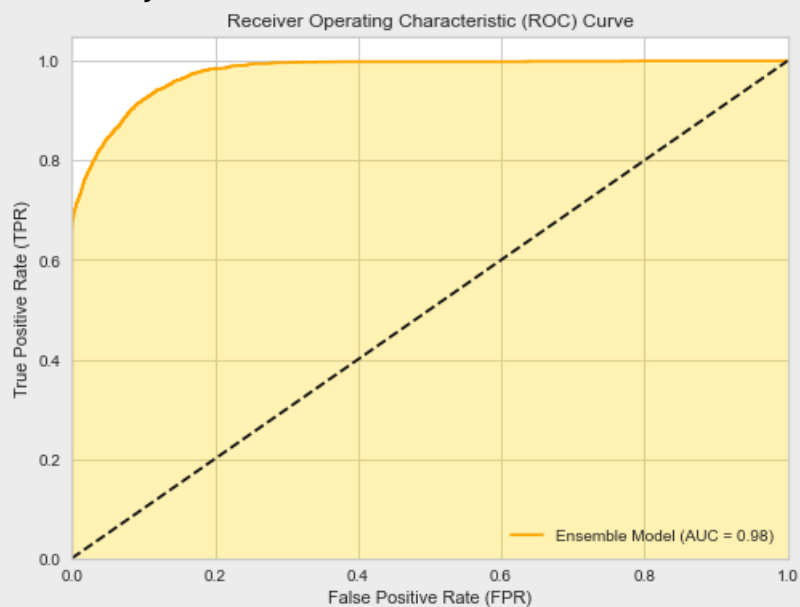
```

# Plot the AUC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label='Ensemble Model (AUC = {:.2f})'.format(auc_score))
plt.fill_between(fpr, tpr, color='lightblue', alpha=0.3)
plt.plot([0, 1], [0, 1], 'k--') # Plot the random guessing line
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

```

## Result:

Ensemble Model Accuracy: **0.9713653387810691**



# Neural Networks

We implemented a neural network model using Keras to predict the presence or absence of diabetes. We standardized the input features and constructed a neural network with multiple layers, including a combination of dense and activation layers. The model was compiled with a binary cross-entropy loss function and the Adam optimizer. We then trained the model on the training data, iterating over a specified number of epochs.

To evaluate the model's performance, we made predictions on the test set and generated a confusion matrix to visualize the results. Additionally, we produced a classification report, providing key metrics such as precision, recall, and F1-score for both positive and negative classes.

## Code Snippet:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

# Standardize the input features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convert the target variable to binary labels
y_train_binary = np.where(y_train > np.mean(y_train), 1, 0)
y_test_binary = np.where(y_test > np.mean(y_train), 1, 0)

# Build the neural network model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(X_train, y_train_binary, epochs=100, batch_size=32, verbose=1)

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test_binary, verbose=0)
print("Loss:", loss)
print("Accuracy:", accuracy)

# Make predictions on the test set and Convert predictions to binary labels
y_pred = model.predict(X_test)
y_pred_binary = np.where(y_pred > 0.5, 1, 0)

# Generate the confusion matrix
```

```

cm = confusion_matrix(y_test_binary, y_pred_binary)

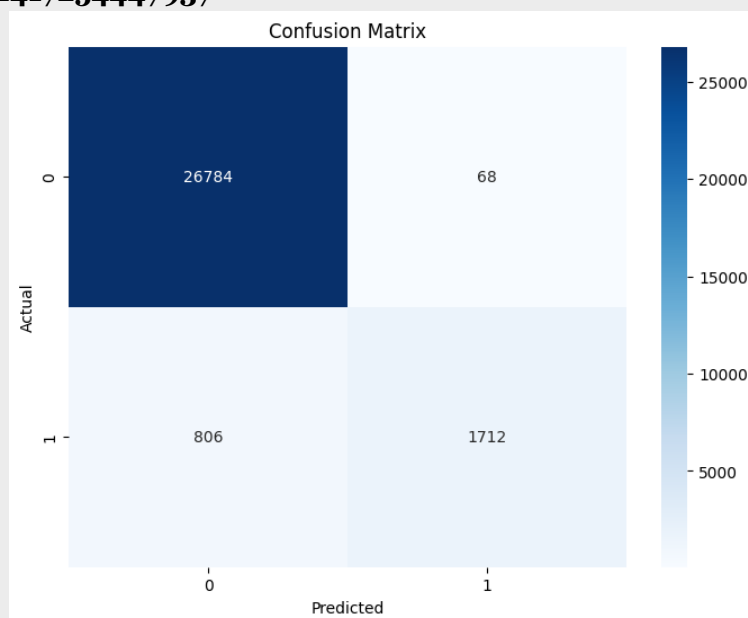
# Create a heatmap of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Generate the classification report
classification_rep = classification_report(y_test_binary, y_pred_binary)
print("Classification Report:\n", classification_rep)

```

## Result:

Epoch 1/100  
 2142/2142 [=====] - 5s 2ms/step - loss: 0.1307 - accuracy: 0.9532  
 Epoch 2/100  
 2142/2142 [=====] - 5s 2ms/step - loss: 0.1073 - accuracy: 0.9637  
 ...  
 Epoch 100/100  
 2142/2142 [=====] - 4s 2ms/step - loss: 0.0766 - accuracy: 0.9733  
 Loss: **0.09086526930332184**  
 Accuracy: **0.9702417254447937**



Classification Report	Precision	Recall	F1-score	Support
<b>0</b>	0.97	1.00	0.98	26852
<b>1</b>	0.96	0.68	0.80	2518
<b>Accuracy</b>			0.97	29370
<b>Macro Avg</b>	0.97	0.84	0.89	29370
<b>Weighted Avg</b>	0.97	0.97	0.97	29370

# Using LIME and SHAP with Machine Learning Models

To gain a deeper understanding of the predictions made by our ensemble model, we employed three explanation methods: LIME, Shapley, and Counterfactual. These methods provide insights into how the features contribute to the model's predictions and help interpret the decision-making process.

## LIME

First, we used LIME to generate explanations for the three sample scenarios we selected, namely AdaBoost, LightGBM, and Gradient Boosting. LIME creates interpretable explanations by approximating the complex model locally. For each scenario, LIME highlighted the important features and their contributions to the prediction. By visualizing the explanations, we were able to identify which features had the most significant impact on the model's decision.

### Code Snippet (for AdaBoost only, all codes are in the python notebook)

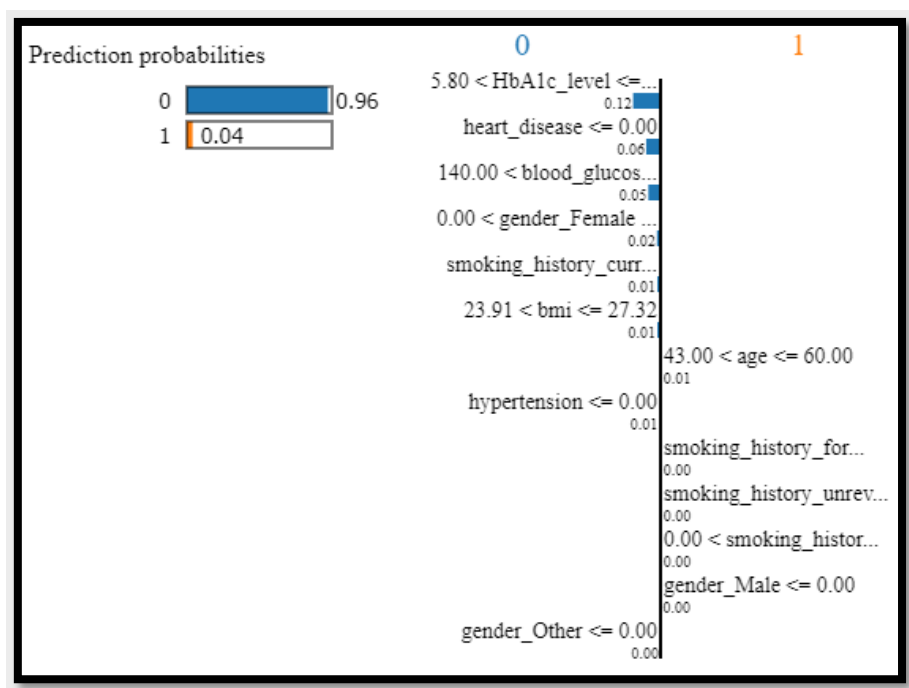
```
# Initialize the LIME explainer
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,
feature_names=X_train.columns, class_names=['0', '1'])

# Select an instance from the test data for explanation
instance = X_test.iloc[3]

# Explain the prediction for the selected instance
explanation = explainer.explain_instance(instance.values,
GradBoost.predict_proba, num_features=len(X_train.columns))

# Print the explanation
explanation.show_in_notebook()
```

### Result:





Feature	Value
HbA1c_level	6.00
heart_disease	0.00
blood_glucose_level	155.00
gender_Female	1.00
smoking_history_current	0.00
bmi	27.32
age	51.00
hypertension	0.00
smoking_history_former	0.00
smoking_history_unrevealed	0.00
smoking_history_never	1.00
gender_Male	0.00
gender_Other	0.00

### SHAPLEY (for AdaBoost only, all codes are in the python notebook)

Next, we utilized Shapley values to gain a better understanding of feature importance in our ensemble model. Shapley assigns values to each feature based on its contribution to the prediction. By analyzing the Shapley explanations for our sample scenarios, we could determine the relative importance of different features. This information helped us identify key factors driving the predictions and provided insights into the decision-making process of the ensemble model.

### Code Snippet:

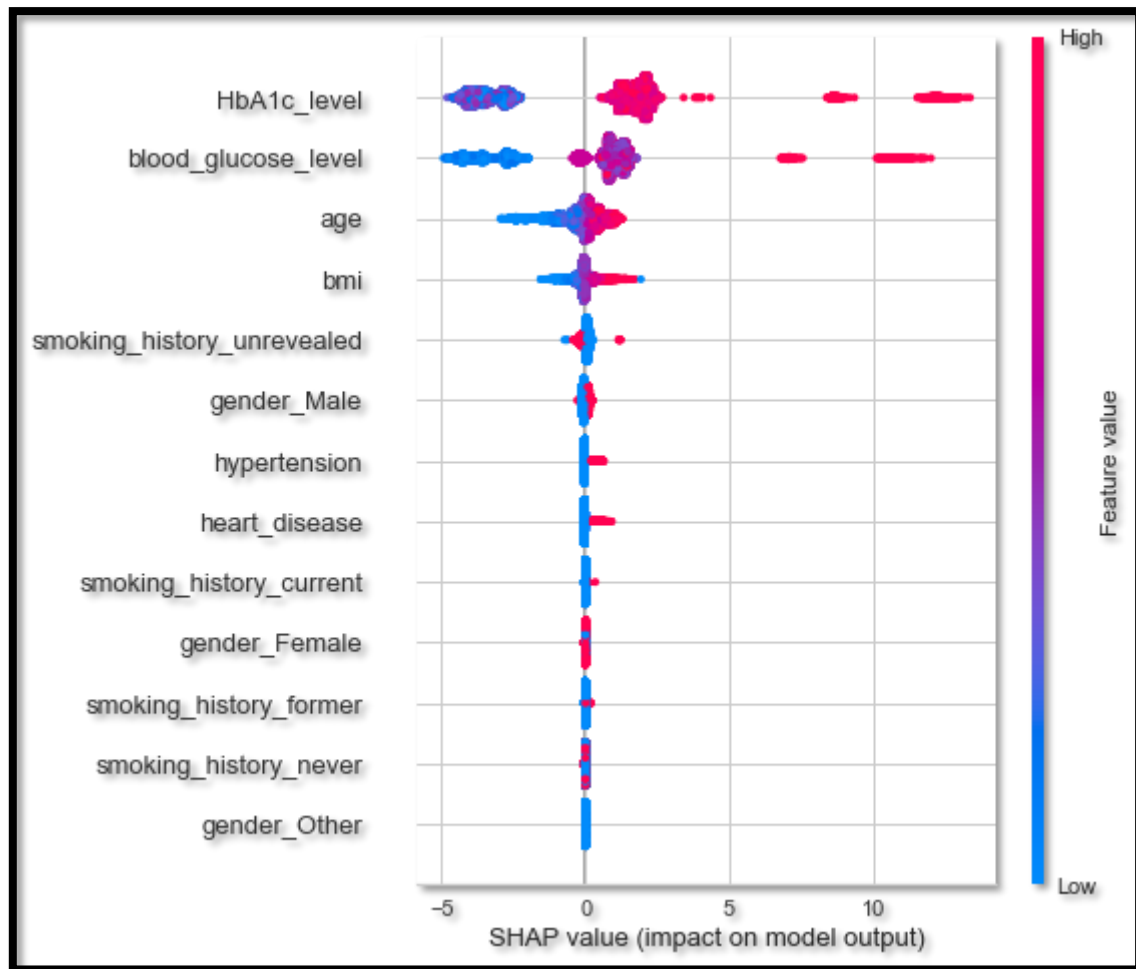
```
# Convert the AdaBoost model to a scikit-learn compatible decision tree-based model
ADABOOST_as_tree = ADABOOST.estimators_[50]

# Initialize the SHAP explainer using the converted decision tree-based model
explainer = shap.Explainer(ADABOOST_as_tree)

# Calculate SHAP values for the test data
shap_values = explainer.shap_values(X_test)

# Visualize the SHAP values
shap.summary_plot(shap_values[1], X_test, plot_type='dot')
```

Result:



### Counter Factual Explanations (for AdaBoost only, all codes are in the python notebook)

Lastly, we employed Counterfactual explanations to explore the "what-if" scenarios. Counterfactual explanations generate alternative feature values that would lead to different predictions. By manipulating the feature values for our sample scenarios, we were able to observe how the ensemble model responded to different inputs. We selected only certain features to vary, and locked features like age, and gender. This analysis helped us identify the thresholds or boundaries of the model's decision-making process and provided valuable insights into potential interventions or changes to achieve desired outcomes.

### Code Snippet:

# Dataset for training an ML model

```
d=dice_ml.Data(dataframe= enc_data,
               continuous_features=['bmi', 'blood_glucose_level',
                                   'HbA1c_level'],
               discrete_features = ['gender', 'smoking_history', 'age'],
               outcome_name='diabetes')

m=dice_ml.Model(model=ADABOOST, backend='sklearn')

exp=dice_ml.Dice(d,m, method='random')
```

```


input_data = X_test[30:31]
input_data

features_to_vary=['bmi', 'blood_glucose_level','HbA1c_level']
permitted_range={'bmi':[15,45], 'blood_glucose_level':[80,250]}

CounterFactualEXP = exp.generate_counterfactuals(input_data, total_CFs=3,
desired_class="opposite", permitted_range=permitted_range,
features_to_vary=features_to_vary)
# Visualize counterfactual explanation
CounterFactualEXP.visualize_as_dataframe(show_only_changes=True)

```

## Result:

100% 1/1 [00:00<00:00, 1.82it/s]  
Query instance (original outcome : 0)

### Input and Output layers respectively:

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	gender_Female	gender_Male	gender_Other	smoking_history_current	smoking_history_former	sr
0	52	0	0	25.9	4.8	160	0	1	0	0	1	

Diverse Counterfactual set (new outcome: 1.0)

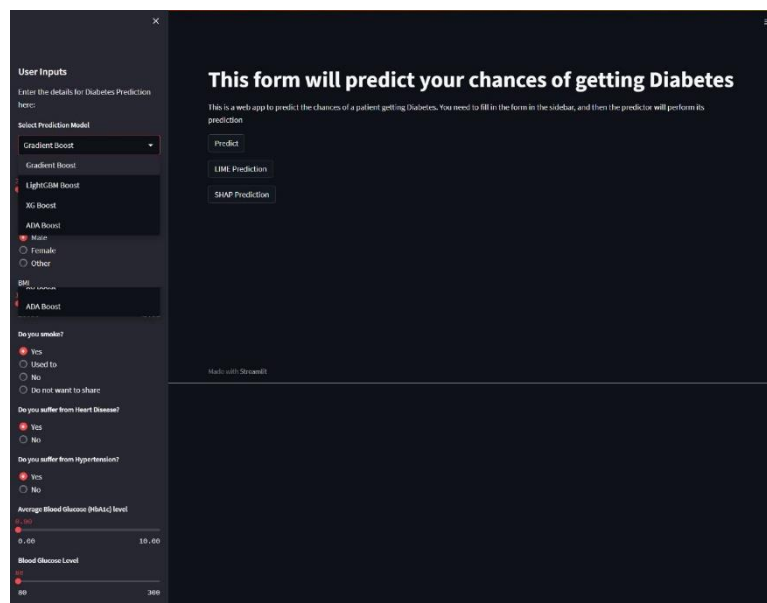
	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	gender_Female	gender_Male	gender_Other	smoking_history_current	smoking_history_former	sr
0	52.0	0.0	0.0	26.0000000000000014	6.9	193.0	0.0	1.0	0.0	0.0	0.0	
1	52.0	0.0	0.0	18.01	8.1	-	0.0	1.0	0.0	0.0	0.0	
2	52.0	0.0	0.0	26.0000000000000014	8.8	150.0	0.0	1.0	0.0	0.0	0.0	

# Streamlit WebApp

For the implementation of our machine learning model and its explanations, we use the Streamlit programming interface. Streamlit provided us with an interactive platform to showcase our model, which enables users to input their data and select the desired model for prediction.

We divided the interface into two main sections: input and model selection, and prediction visualization.

In the input and model selection section, users must manually enter their feature values or select them from predefined options. For model selection, we provided a dropdown menu where users can choose the desired machine learning model for prediction. The available options include the four standard machine learning algorithms used in our project (Light GBM, AdaBoost, XGBoost, and Gradient Boosting).

The screenshot shows a Streamlit web application titled "This form will predict your chances of getting Diabetes". The interface is split into two main sections. The left sidebar, titled "User Inputs", contains a form for entering details for a diabetes prediction. It includes a "Select Prediction Model" dropdown menu with options: Gradient Boost, LightGBM Boost (selected), XG Boost, and Ada Boost. Below this are radio buttons for "Male" (selected), "Female", and "Other". There are also radio buttons for "Do you smoke?" with options "Yes" (selected), "Used to", "No", and "Do not want to share". Further down are radio buttons for "Do you suffer from Heart Disease?" (selected "Yes") and "Do you suffer from Hypertension?" (selected "Yes"). At the bottom of the sidebar are two sliders: "Average Blood Glucose (HbA1c) level" ranging from 0 to 10.00, and "Blood Glucose Level" ranging from 0 to 200. The main content area on the right has a heading "This form will predict your chances of getting Diabetes" and a subtext "This is a web app to predict the chances of a patient getting Diabetes. You need to fill in the form in the sidebar, and then the predictor will perform its prediction." Below the subtext are three buttons: "Predict", "LIME Prediction", and "SHAP Prediction".

Once the user has provided the input and selected the model, in the prediction visualization section, the program processes the data and generates predictions for the likelihood of having diabetes. Then the user may select Predict, Lime or Shapley option to generate respective predictions.

When the user clicks on Predict, the program will generate an output statement of whether the user will have diabetes or not.

User Inputs

Enter the details for Diabetes Prediction here:

Select Prediction Model

ADA Boost

Age

2

Gender

Male

Female

Other

BMI

22.58

Do you smoke?

Yes

Used to

No

Do not want to share

Do you suffer from Heart Disease?

Yes

No

Do you suffer from Hypertension?

Yes

No

Average Blood Glucose (HbA1c) level

4.78

Blood Glucose Level

191

This form will predict your chances of getting Diabetes

This is a web app to predict the chances of a patient getting Diabetes. You need to fill in the form in the sidebar, and then the predictor will perform its prediction

Predict

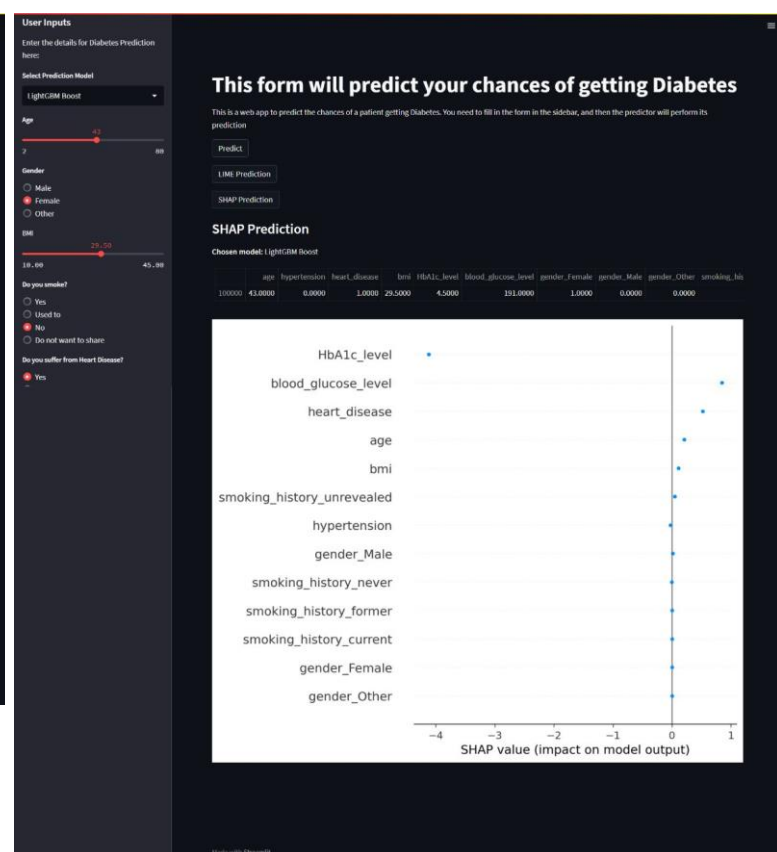
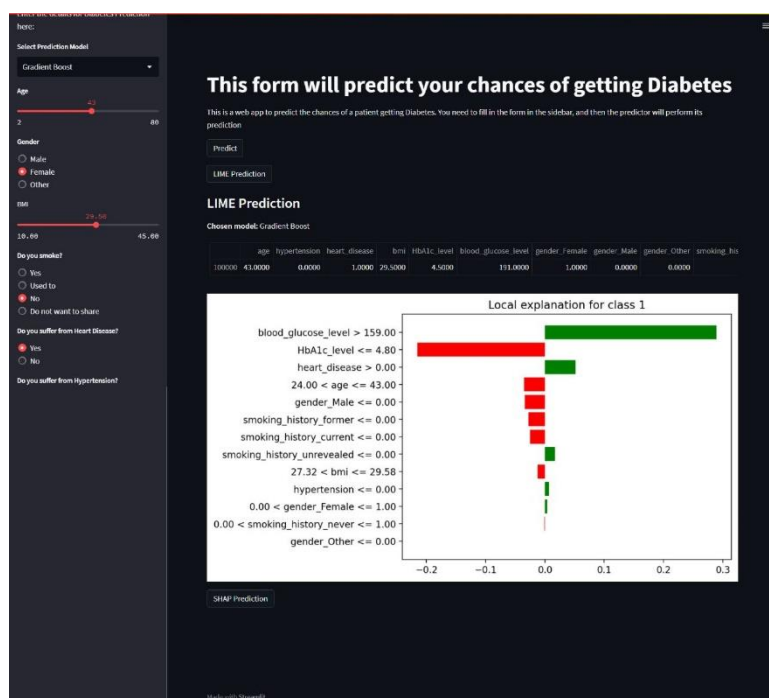
age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	gender_Female	gender_Male	gender_Other	smoking_history_curr
43.0000	0.0000	1.0000	29.5000	4.5000	191.0000	1.0000	0.0000	0.0000	0.0

According to the ADA Boost model, you are less likely to have diabetes

LIME Prediction

SHAP Prediction

Similarly, when the user will choose Lime or Shapley option, the program will apply Lime or Shapley methods to highlight the features contributing to the prediction and their respective importance.





# Conclusion

---

In our machine learning project, we successfully developed an ensemble model to predict the likelihood of having diabetes. By selecting a relevant medical data set and applying preprocessing techniques, we improved the quality of the data and enhanced the performance of the models. We used PyCaret library to find the optimum models, evaluated based on their accuracy and used them individually.

We also created an ensemble model of the best 5 models, but the top individual models outperformed the ensemble model, so we used the individual models in our project instead.

To enhance interpretation, we used explanation methods such as LIME and Shapley. These techniques helped us understand the model's decision-making process and identify key features influencing the predictions.

We developed an interactive programming interface using Streamlit, allowing users to input their data, select a model, and visualize predictions alongside explanations. This interface aims to make the model accessible and provide a better understanding of the results.

Nonetheless, our project has limitations, such as dataset representativeness and limitedness. Improvements in getting additional data sources and advanced explanation methods will make our model predictions more effective. Adding more features and getting insights from domain experts may further improve our model predictions.

# References

---

Dataset: <https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset>

GitHub Repository: <https://github.com/HHadiKhan/ML2023.git>