# Diabetes-Healthcare Classifier

In [1]:
```
# Importing Libraries
import pandas as pd
import numpy as np
import dtale
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score, roc_curve, classification_report, f1_score, precision_score, recall_s
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import logging
logging.disable(logging.CRITICAL)
import optuna
from sklearn.model_selection import cross_val_score
```

In [2]:
```
df = pd.read_csv("Healthcare-Diabetes.csv")
```

In [3]:
```
df.head()
```

Out[3]:

| | Id | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 2 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 3 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| **3** | 4 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 5 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

In [4]:
```
df.shape
```

Out[4]:
```
(2768, 10)
```

In [5]:
```
# Checking Null Values --None
df.isnull().any()
```

Out[5]:
```
Id                          False
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

In [6]:
```
df.drop(columns = ['Id'],inplace = True)
```

In [7]:
```
# Uni/Bi/Multivariate- Analysis
dtale.show(df)
```

Out[7]:

In [8]:
```
df.columns
```

Out[8]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [9]:
```
# Standardization (Not required for Gradient Descent)
scaler = StandardScaler()
columns_to_standardize = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
                          'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
standardized_data = scaler.fit_transform(df[columns_to_standardize])

# Converting the standardized data back to a DataFrame
df_standardized = pd.DataFrame(standardized_data, columns=columns_to_standardize)

# Adding the target column to the standardized DataFrame
df_standardized["Outcome"] = df["Outcome"].values
```

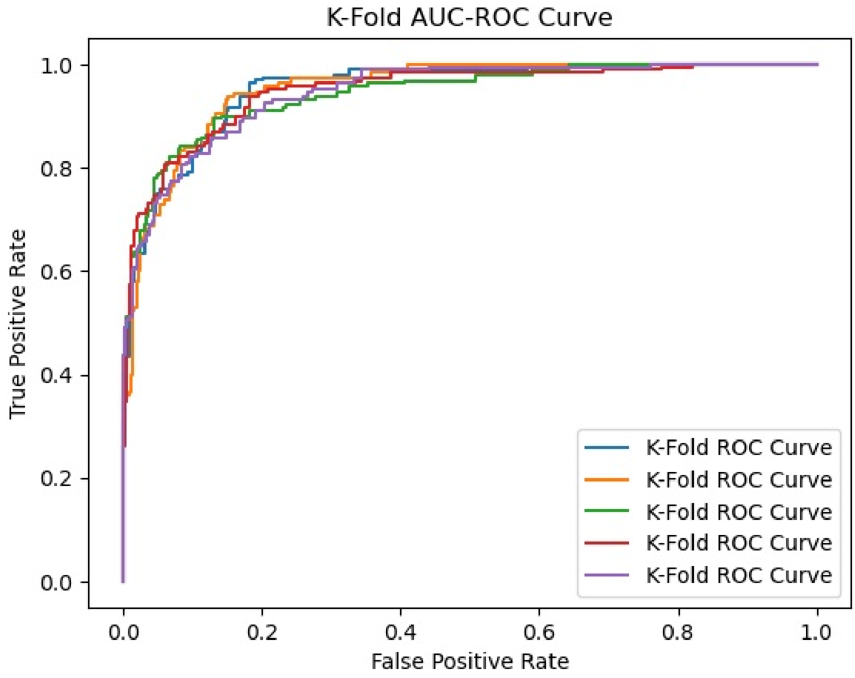
In [10]:
```
df1 = df_standardized.copy(deep = True)
```

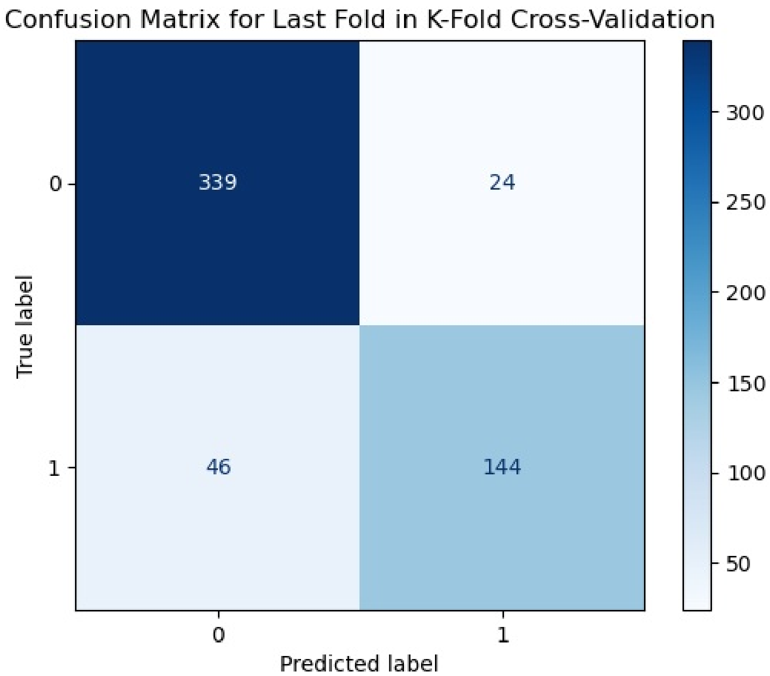
## K-FOLD BASE MODEL

In [11]:
```
X = df1.drop(columns=['Outcome'])  # Features
y = df1['Outcome']                # Target
```

In [12]:
```
# Initialize the Gradient Boosting Classifier
model = GradientBoostingClassifier(random_state=42)

# Stratified K-Fold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

f1_scores_kfold = []
precision_scores_kfold = []
recall_scores_kfold = []
auc_scores_kfold = []

# For Confusion Matrix Display
all_conf_matrices = []

for train_index, test_index in skf.split(X, y):
    # Splitting the data
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

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

    # Predictions
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]
```

```
    # Metrics
    f1_scores_kfold.append(f1_score(y_test, y_pred))
    precision_scores_kfold.append(precision_score(y_test, y_pred))
    recall_scores_kfold.append(recall_score(y_test, y_pred))
    auc_scores_kfold.append(roc_auc_score(y_test, y_pred_proba))

    # Save confusion matrix for the fold
    conf_matrix = confusion_matrix(y_test, y_pred)
    all_conf_matrices.append(conf_matrix)

    # ROC Curve for one fold
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    plt.plot(fpr, tpr, label=f"K-Fold ROC Curve")

# Plot AUC-ROC Curve for K-Fold
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("K-Fold AUC-ROC Curve")
plt.legend(loc="lower right")
plt.show()

# Plot Confusion Matrix for the last fold as an example
disp = ConfusionMatrixDisplay(confusion_matrix=all_conf_matrices[-1], display_labels=model.classes_)
disp.plot(cmap='Blues', values_format='d')
plt.title("Confusion Matrix for Last Fold in K-Fold Cross-Validation")
plt.show()

# Print Metrics
print(f"K-Fold F1 Scores: {f1_scores_kfold}")
print(f"K-Fold Precision Scores: {precision_scores_kfold}")
print(f"K-Fold Recall Scores: {recall_scores_kfold}")
print(f"K-Fold AUC Scores: {auc_scores_kfold}")
print(f"Mean F1 Score (K-Fold): {np.mean(f1_scores_kfold)}")
print(f"Mean AUC Score (K-Fold): {np.mean(auc_scores_kfold)}")
```



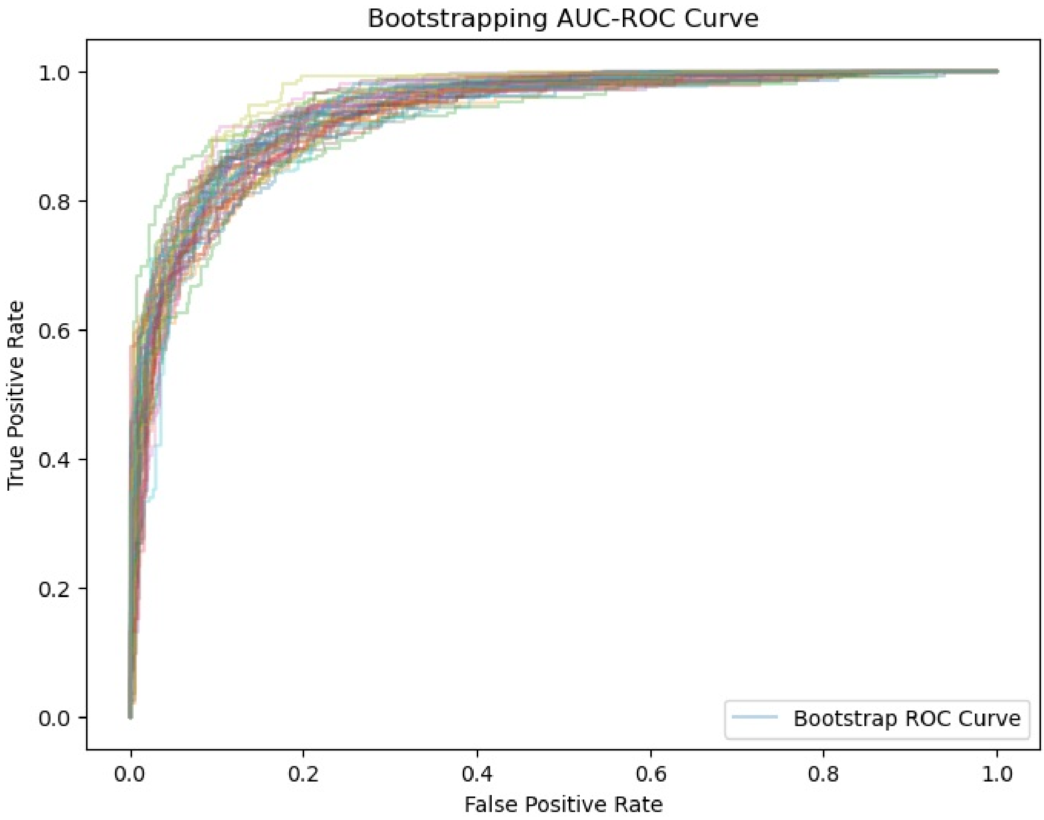


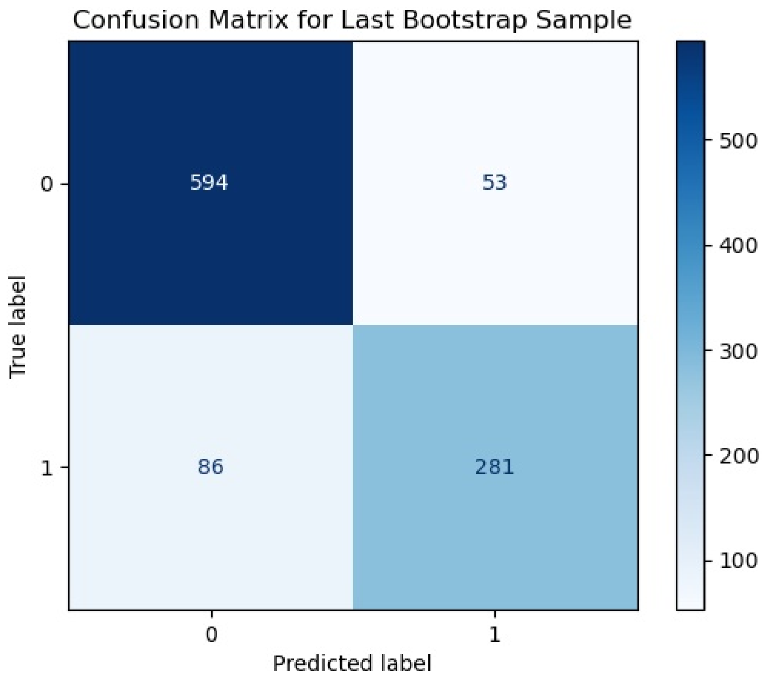

```
K-Fold F1 Scores: [0.807799442896936, 0.7897727272727273, 0.8364611260053619, 0.8245125348189415, 0.804469273743
0168]
K-Fold Precision Scores: [0.8630952380952381, 0.8633540372670807, 0.8524590163934426, 0.8757396449704142, 0.8571
428571428571]
K-Fold Recall Scores: [0.7591623036649214, 0.7277486910994765, 0.8210526315789474, 0.7789473684210526, 0.7578947
368421053]
K-Fold AUC Scores: [0.952605541372795, 0.953038235760749, 0.943984962406015, 0.9481078729882557, 0.9456140350877
194]
Mean F1 Score (K-Fold): 0.8126030209473967
Mean AUC Score (K-Fold): 0.9486701295231068
```

## Bootstrap Model

In [13]:
```
n_bootstrap_samples = 50
f1_scores_bootstrap = []
precision_scores_bootstrap = []
recall_scores_bootstrap = []
auc_scores_bootstrap = []

plt.figure(figsize=(8, 6))  # Set figure size for better visualization

# Store confusion matrices for visualization later
confusion_matrices = []

for i in range(n_bootstrap_samples):
    # Create Bootstrap Sample
    indices = np.random.choice(range(len(X)), size=len(X), replace=True)
    X_train, y_train = X.iloc[indices], y.iloc[indices]

    # Out-of-Bag (OOB) Data
    oob_indices = list(set(range(len(X))) - set(indices))
    if len(oob_indices) == 0 or len(y_train.unique()) < 2:
        continue  # Skip iteration if no OOB data or only one class
    X_test, y_test = X.iloc[oob_indices], y.iloc[oob_indices]
```

```python
    # Train the model
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    # Metrics
    f1_scores_bootstrap.append(f1_score(y_test, y_pred))
    precision_scores_bootstrap.append(precision_score(y_test, y_pred))
    recall_scores_bootstrap.append(recall_score(y_test, y_pred))
    auc_scores_bootstrap.append(roc_auc_score(y_test, y_pred_proba))

    # Save confusion matrix for the last bootstrap sample
    conf_matrix = confusion_matrix(y_test, y_pred)
    confusion_matrices.append(conf_matrix)

    # Plot ROC Curve only for the first iteration with a label
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    if i == 0:
        plt.plot(fpr, tpr, label="Bootstrap ROC Curve", alpha=0.3)
    else:
        plt.plot(fpr, tpr, alpha=0.3)

# Finalize and Display AUC-ROC Plot
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Bootstrapping AUC-ROC Curve")
plt.legend(loc="lower right")  # Show single legend
plt.show()

# Display Confusion Matrix for the last bootstrap sample
if confusion_matrices:
    disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrices[-1], display_labels=model.classes_)
    disp.plot(cmap='Blues', values_format='d')
    plt.title("Confusion Matrix for Last Bootstrap Sample")
    plt.show()

# Print Metrics
print(f"Bootstrapping F1 Scores: {f1_scores_bootstrap[:5]}...")  # Showing first 5 scores
print(f"Bootstrapping Precision Scores: {precision_scores_bootstrap[:5]}...")
print(f"Bootstrapping Recall Scores: {recall_scores_bootstrap[:5]}...")
print(f"Bootstrapping AUC Scores: {auc_scores_bootstrap[:5]}...")
print(f"Mean F1 Score (Bootstrapping): {np.mean(f1_scores_bootstrap)}")
print(f"Mean AUC Score (Bootstrapping): {np.mean(auc_scores_bootstrap)}")
```






```
Bootstrapping F1 Scores: [0.7856025039123631, 0.7717717717717718, 0.8612903225806452, 0.7678571428571429, 0.7981366459627329]...
Bootstrapping Precision Scores: [0.8311258278145696, 0.8210862619808307, 0.902027027027027, 0.819047619047619, 0.8371335504885994]...
Bootstrapping Recall Scores: [0.744807121661721, 0.7280453257790368, 0.8240740740740741, 0.7226890756302521, 0.7626112759643917]...
Bootstrapping AUC Scores: [0.9269081473266255, 0.9242285550672642, 0.9561955723428812, 0.918435459290099, 0.9466034417729629]...
Mean F1 Score (Bootstrapping): 0.8013186055213821
Mean AUC Score (Bootstrapping): 0.9379810966521147
```

## Finding Best Parameters Using Bayesian Optimization Hyperparameter Tuning

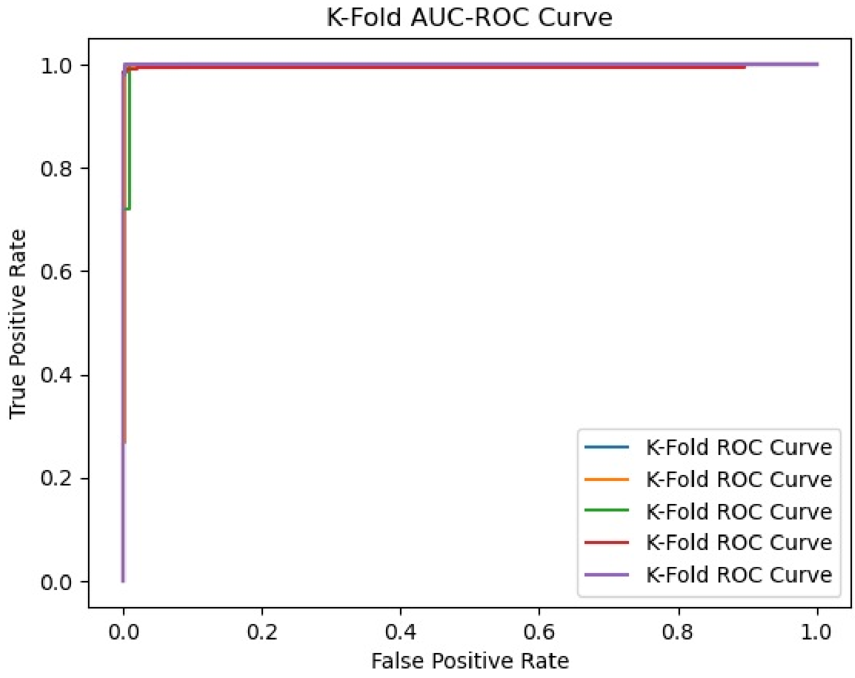
In [14]:
```
def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 50, 200),  # Reduced range
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.2),  # Reduced range
        'max_depth': trial.suggest_int('max_depth', 3, 7),  # Reduced range
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 5),  # Reduced range
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 4),  # Reduced range
        'subsample': trial.suggest_float('subsample', 0.5, 1.0)
    }
    model = GradientBoostingClassifier(random_state=42, **params)

    # Use reduced CV folds and smaller dataset for speed
    scores = cross_val_score(model, X, y, scoring='f1', cv=3, n_jobs=-1)
    return scores.mean()

# Optimize
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=20, timeout=600, show_progress_bar=True)  # Smaller trials with timeout

# Print best parameters
print(f"Best Parameters: {study.best_params}")
print(f"Best F1 Score: {study.best_value}")
```

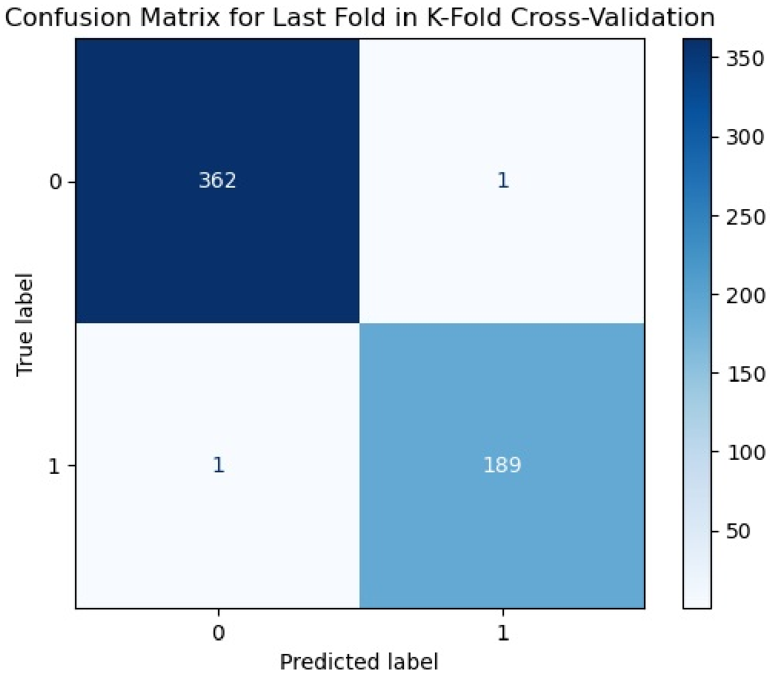
```
  0%|          | 0/20 [00:00<?, ?it/s]
Best Parameters: {'n_estimators': 96, 'learning_rate': 0.08020404204698642, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 2, 'subsample': 0.9877960074080036}
Best F1 Score: 0.9895500611222839
```

## Hyperparameter Tuned K-Fold Sampled Model

In [21]:
```
# Initialize the Gradient Boosting Classifier
model = GradientBoostingClassifier(
    random_state=42,
    n_estimators=96,
    learning_rate= 0.08020404204698642,
    max_depth=7,
    min_samples_split=5,
```

```
    min_samples_leaf=2,
    subsample=0.9877960074080036
)
# Stratified K-Fold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

f1_scores_kfold = []
precision_scores_kfold = []
recall_scores_kfold = []
auc_scores_kfold = []

# For Confusion Matrix Display
all_conf_matrices = []

for train_index, test_index in skf.split(X, y):
    # Splitting the data
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

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

    # Predictions
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    # Metrics
    f1_scores_kfold.append(f1_score(y_test, y_pred))
    precision_scores_kfold.append(precision_score(y_test, y_pred))
    recall_scores_kfold.append(recall_score(y_test, y_pred))
    auc_scores_kfold.append(roc_auc_score(y_test, y_pred_proba))

    # Save confusion matrix for the fold
    conf_matrix = confusion_matrix(y_test, y_pred)
    all_conf_matrices.append(conf_matrix)

    # ROC Curve for one fold
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    plt.plot(fpr, tpr, label=f"K-Fold ROC Curve")

# Plot AUC-ROC Curve for K-Fold
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("K-Fold AUC-ROC Curve")
plt.legend(loc="lower right")
plt.show()

# Plot Confusion Matrix for the last fold as an example
disp = ConfusionMatrixDisplay(confusion_matrix=all_conf_matrices[-1], display_labels=model.classes_)
disp.plot(cmap='Blues', values_format='d')
plt.title("Confusion Matrix for Last Fold in K-Fold Cross-Validation")
plt.show()

# Print Metrics
print(f"K-Fold F1 Scores: {f1_scores_kfold}")
print(f"K-Fold Precision Scores: {precision_scores_kfold}")
print(f"K-Fold Recall Scores: {recall_scores_kfold}")
print(f"K-Fold AUC Scores: {auc_scores_kfold}")
print(f"Mean F1 Score (K-Fold): {np.mean(f1_scores_kfold)}")
print(f"Mean AUC Score (K-Fold): {np.mean(auc_scores_kfold)}")
```



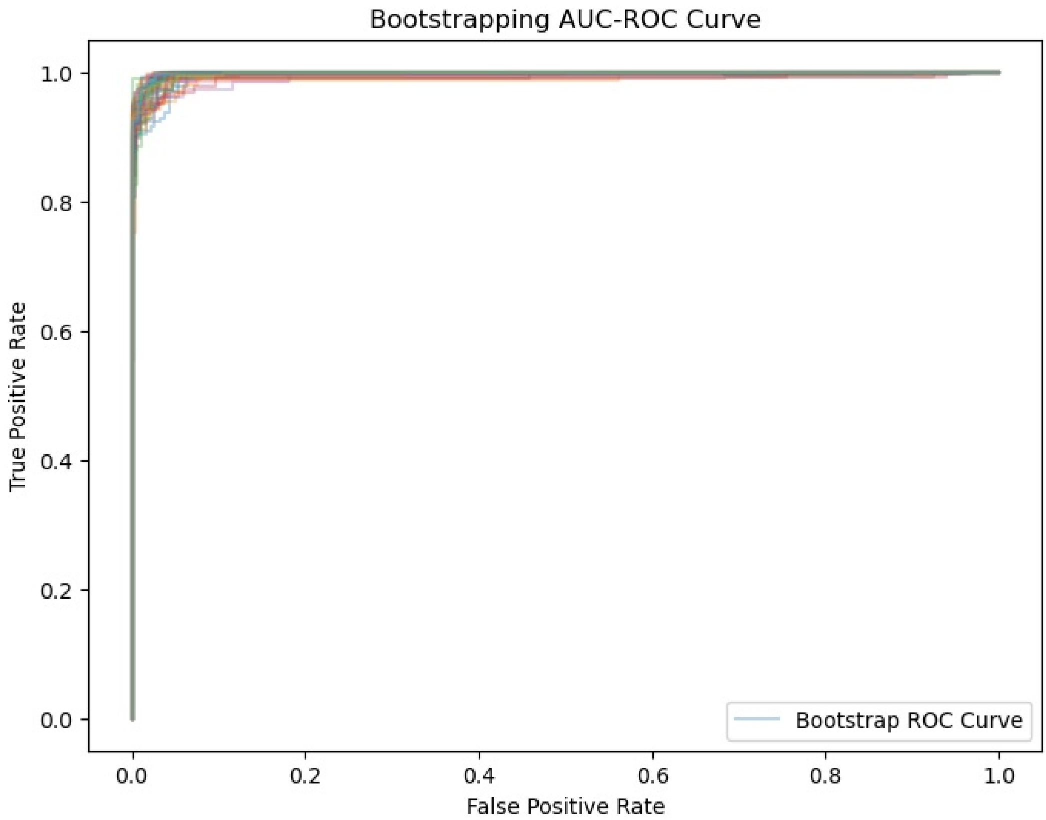



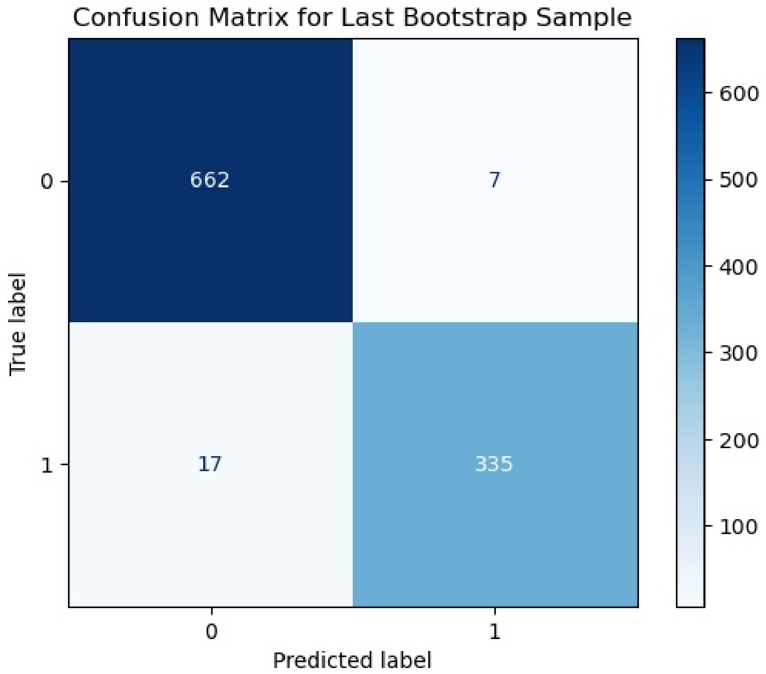

```
K-Fold F1 Scores: [0.9921259842519685, 0.9947643979057592, 0.9844559585492227, 0.9894736842105263, 0.9947368421052631]
K-Fold Precision Scores: [0.9947368421052631, 0.9947643979057592, 0.9693877551020408, 0.9894736842105263, 0.9947368421052631]
K-Fold Recall Scores: [0.9895287958115183, 0.9947643979057592, 1.0, 0.9894736842105263, 0.9947368421052631]
K-Fold AUC Scores: [0.9994374972956601, 0.9977788354751707, 0.9977009832272989, 0.9951573147745398, 0.999942003769755]
Mean F1 Score (K-Fold): 0.9911113734045479
Mean AUC Score (K-Fold): 0.9980033269084849
```

## Hyperparameter Tuned Bootstrap Sampled Model

In [16]:
```
n_bootstrap_samples = 50
f1_scores_bootstrap = []
precision_scores_bootstrap = []
recall_scores_bootstrap = []
auc_scores_bootstrap = []

plt.figure(figsize=(8, 6))  # Set figure size for better visualization

# Store confusion matrices for visualization later
confusion_matrices = []

for i in range(n_bootstrap_samples):
    # Create Bootstrap Sample
    indices = np.random.choice(range(len(X)), size=len(X), replace=True)
    X_train, y_train = X.iloc[indices], y.iloc[indices]

    # Out-of-Bag (OOB) Data
```

```python
    oob_indices = list(set(range(len(X))) - set(indices))
    if len(oob_indices) == 0 or len(y_train.unique()) < 2:
        continue  # Skip iteration if no OOB data or only one class
    X_test, y_test = X.iloc[oob_indices], y.iloc[oob_indices]

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

    # Predictions
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    # Metrics
    f1_scores_bootstrap.append(f1_score(y_test, y_pred))
    precision_scores_bootstrap.append(precision_score(y_test, y_pred))
    recall_scores_bootstrap.append(recall_score(y_test, y_pred))
    auc_scores_bootstrap.append(roc_auc_score(y_test, y_pred_proba))

    # Save confusion matrix for the last bootstrap sample
    conf_matrix = confusion_matrix(y_test, y_pred)
    confusion_matrices.append(conf_matrix)

    # Plot ROC Curve only for the first iteration with a label
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    if i == 0:
        plt.plot(fpr, tpr, label="Bootstrap ROC Curve", alpha=0.3)
    else:
        plt.plot(fpr, tpr, alpha=0.3)

# Finalize and Display AUC-ROC Plot
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Bootstrapping AUC-ROC Curve")
plt.legend(loc="lower right")  # Show single legend
plt.show()

# Display Confusion Matrix for the last bootstrap sample
if confusion_matrices:
    disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrices[-1], display_labels=model.classes_)
    disp.plot(cmap='Blues', values_format='d')
    plt.title("Confusion Matrix for Last Bootstrap Sample")
    plt.show()

# Print Metrics
print(f"Bootstrapping F1 Scores: {f1_scores_bootstrap[:5]}...")  # Showing first 5 scores
print(f"Bootstrapping Precision Scores: {precision_scores_bootstrap[:5]}...")
print(f"Bootstrapping Recall Scores: {recall_scores_bootstrap[:5]}...")
print(f"Bootstrapping AUC Scores: {auc_scores_bootstrap[:5]}...")
print(f"Mean F1 Score (Bootstrapping): {np.mean(f1_scores_bootstrap)}")
print(f"Mean AUC Score (Bootstrapping): {np.mean(auc_scores_bootstrap)}")
```






```
Bootstrapping F1 Scores: [0.9656160458452722, 0.9744318181818182, 0.9746478873239437, 0.949438202247191, 0.96718
97289586305]...
Bootstrapping Precision Scores: [0.9683908045977011, 0.9634831460674157, 0.969187675070028, 0.9602272727272727,
0.9741379310344828]...
Bootstrapping Recall Scores: [0.9628571428571429, 0.985632183908046, 0.9801699716713881, 0.9388888888888889, 0.9
603399433427762]...
Bootstrapping AUC Scores: [0.9987329931972789, 0.9971851665408172, 0.9991600237169775, 0.9894629094412332, 0.998
2609735984174]...
Mean F1 Score (Bootstrapping): 0.9642962344682645
Mean AUC Score (Bootstrapping): 0.9966428080006231
```

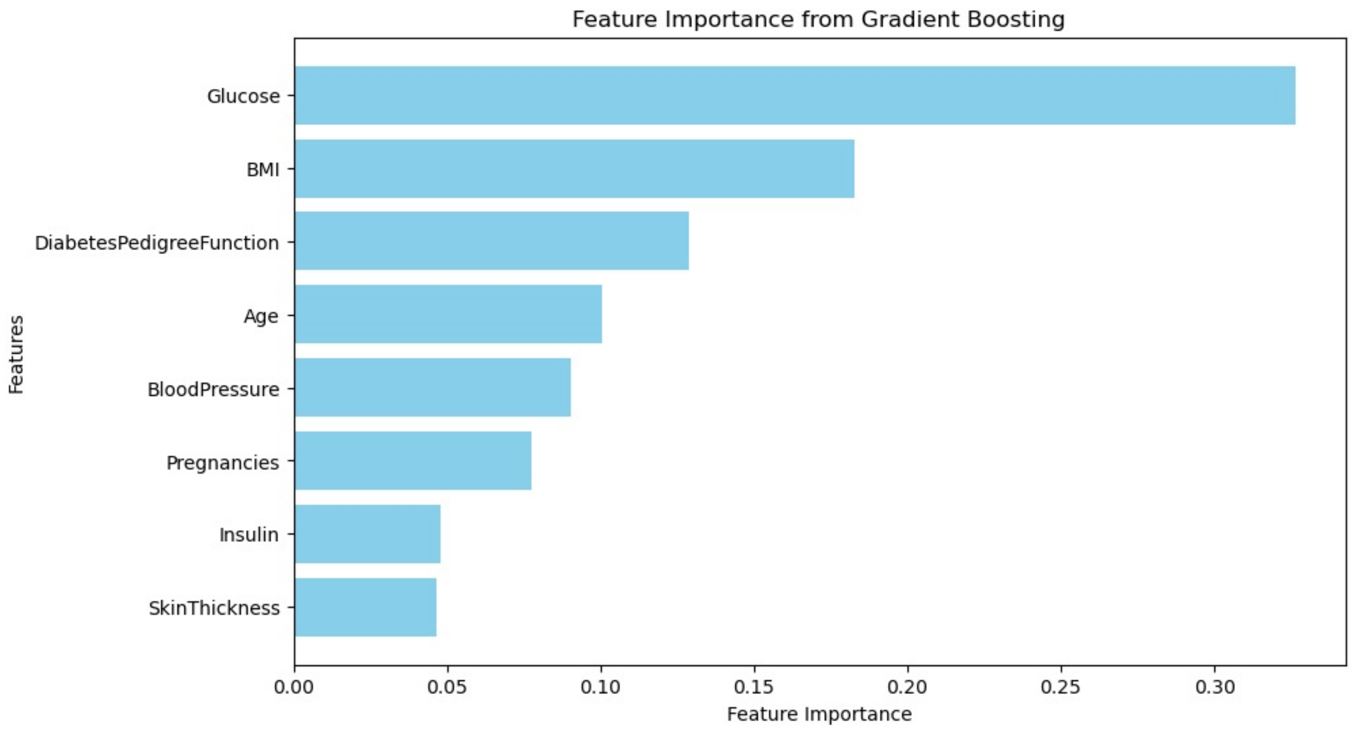
## Feature Importance

In [17]:
```python
feature_importance = model.feature_importances_

# Match importance scores with feature names
features = X.columns  # Assuming `X` is your input DataFrame with feature names
importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': feature_importance
}).sort_values(by='Importance', ascending=False)
```

In [18]:
```python
print(importance_df)
```

```
                    Feature  Importance
1                   Glucose    0.326323
5                       BMI    0.182679
6  DiabetesPedigreeFunction    0.128799
7                       Age    0.100334
2             BloodPressure    0.090225
0               Pregnancies    0.077337
4                   Insulin    0.047760
3             SkinThickness    0.046543
```

In [19]:
```python
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance from Gradient Boosting')
plt.gca().invert_yaxis()  # Invert y-axis to show the most important feature at the top
plt.show()
```


Feature Importance from Gradient Boosting
Glucose
BMI
DiabetesPedigreeFunction
Age
BloodPressure
Pregnancies
Insulin
SkinThickness
Features
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Feature Importance