```
In [12]: # Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (confusion_matrix, classification_report, roc_curve, auc, accuracy_score)
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings("ignore")
```

```
In [3]: # Set visualization style
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)
```

Data Loading

```
In [4]: df = pd.read_csv('diabetes.csv')
print("Dataset shape:", df.shape)
print("First five rows:")
print(df.head())
```

```
Dataset shape: (768, 9)
First five rows:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

Data Overview

```
In [5]: print("\nDataset info:")
print(df.info())
```

```
Dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

```
In [6]: print("\nSummary statistics:")
print(df.describe())
```

```
Summary statistics:
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

```
In [7]: # Check for missing values
print("\nMissing values:")
print(df.isnull().sum())
```

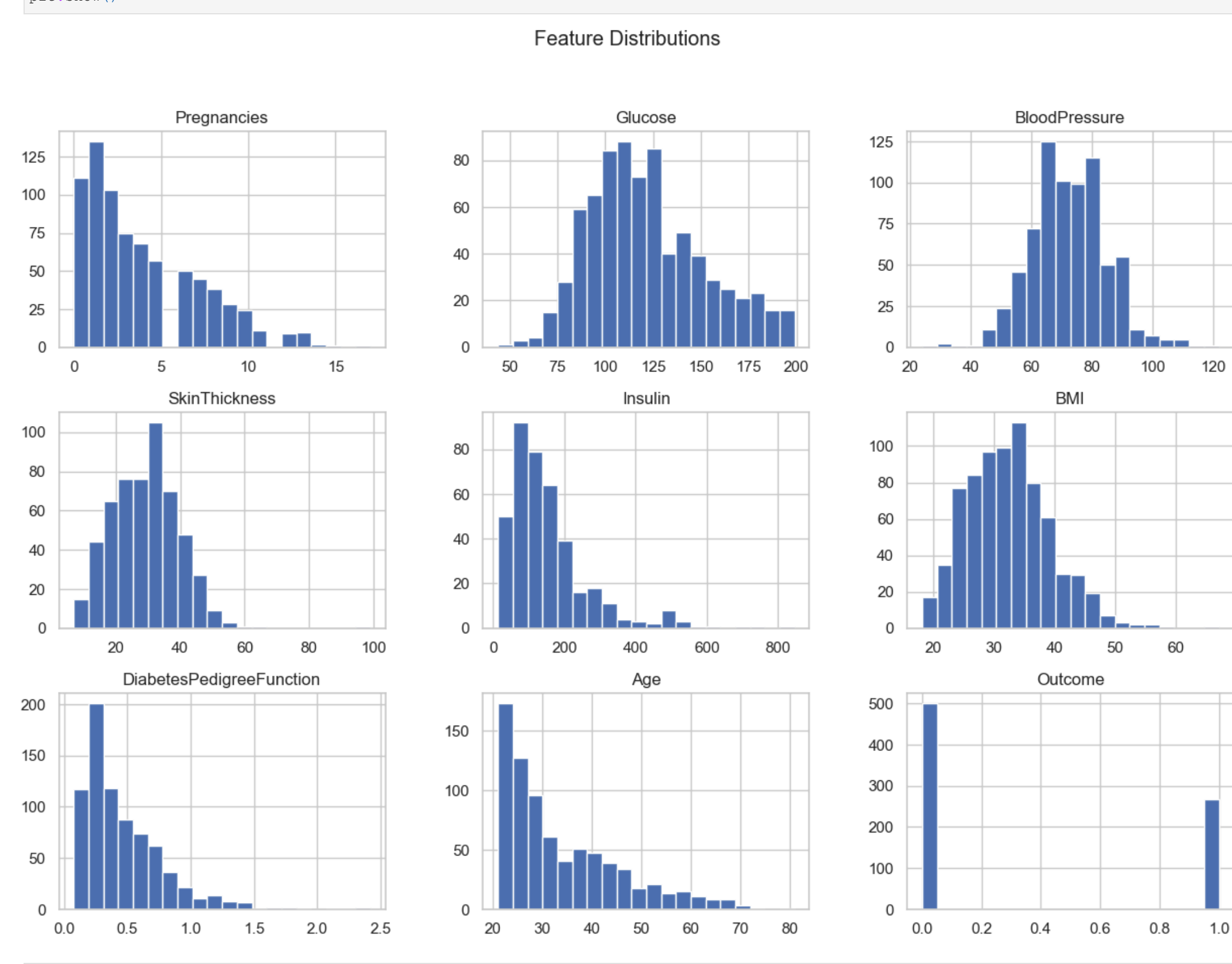
```
Missing values:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
In [8]: # Replace zeros with NaN for these columns and then impute later.
cols_with_zero = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
df[cols_with_zero] = df[cols_with_zero].replace(0, np.NaN)
print("\nMissing values after replacing zeros:")
print(df.isnull().sum())
```

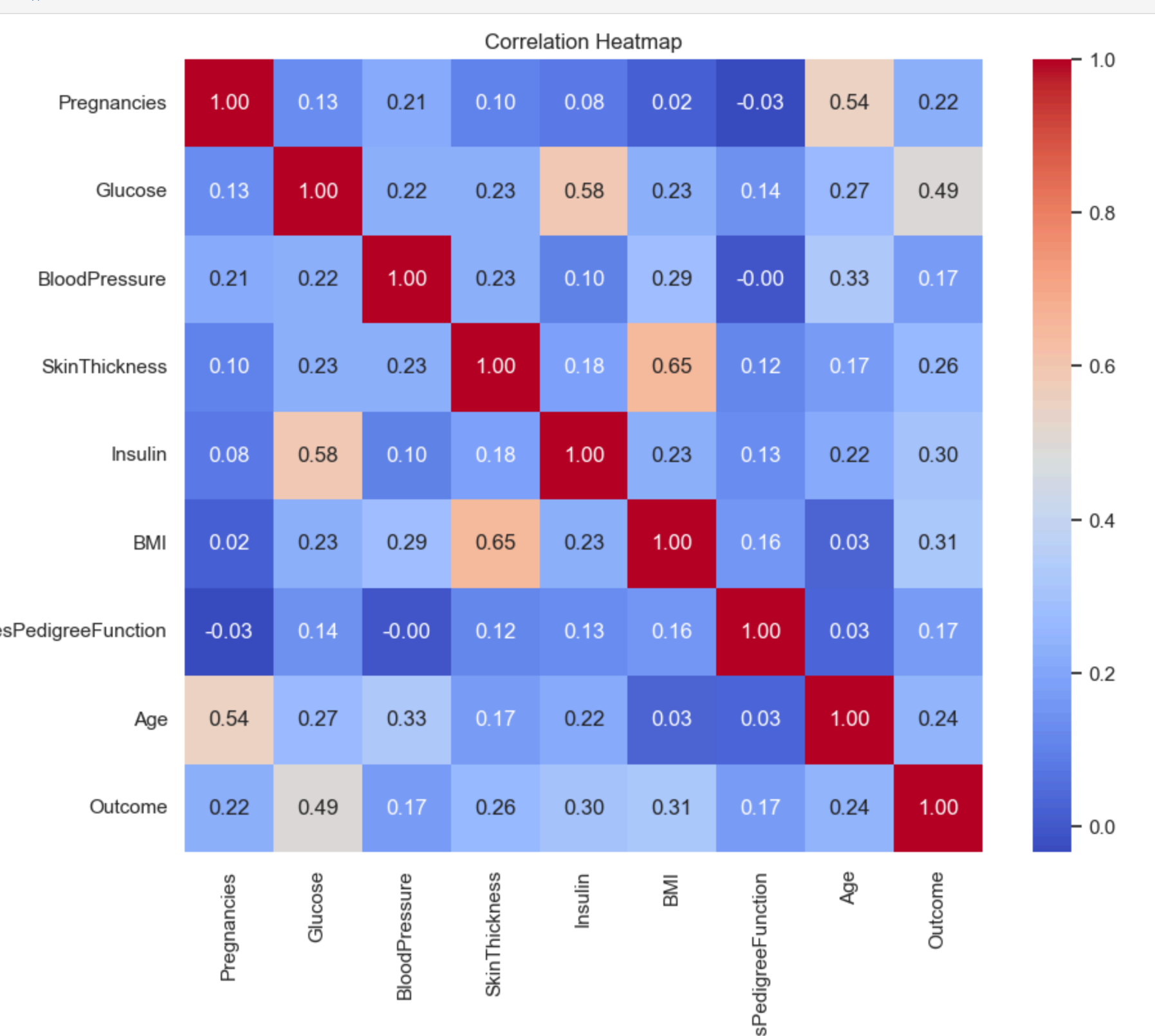
```
Missing values after replacing zeros:
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

2. Exploratory Data Analysis (EDA)

```
In [9]: # Visualizations: Histograms of features
df.hist(bins=20, figsize=(15,10))
plt.suptitle("Feature Distributions")
plt.show()
```




```
In [10]: # Correlation heatmap
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



3. Data Preprocessing

```
In [13]: for col in cols_with_zero:
    median_val = df[col].median()
    df[col].fillna(median_val, inplace=True)

# Verify no missing values remain
print("\nMissing values after imputation:")
print(df.isnull().sum())
```

```
Missing values after imputation:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
In [14]: # Separate features and target
X = df.drop('Outcome', axis=1)
y = df['Outcome']
```

```
In [15]: # Split data into training and testing sets (80/20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
print("\nTraining set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

```
Training set shape: (614, 8)
Test set shape: (154, 8)
```

```
In [16]: # Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

4. Model Building and Training

```
In [17]: # a) Logistic Regression
lr_model = LogisticRegression(max_iter=1000, random_state=42)
lr_model.fit(X_train_scaled, y_train)
lr_preds = lr_model.predict(X_test_scaled)
lr_probs = lr_model.predict_proba(X_test_scaled)[:, 1]
```

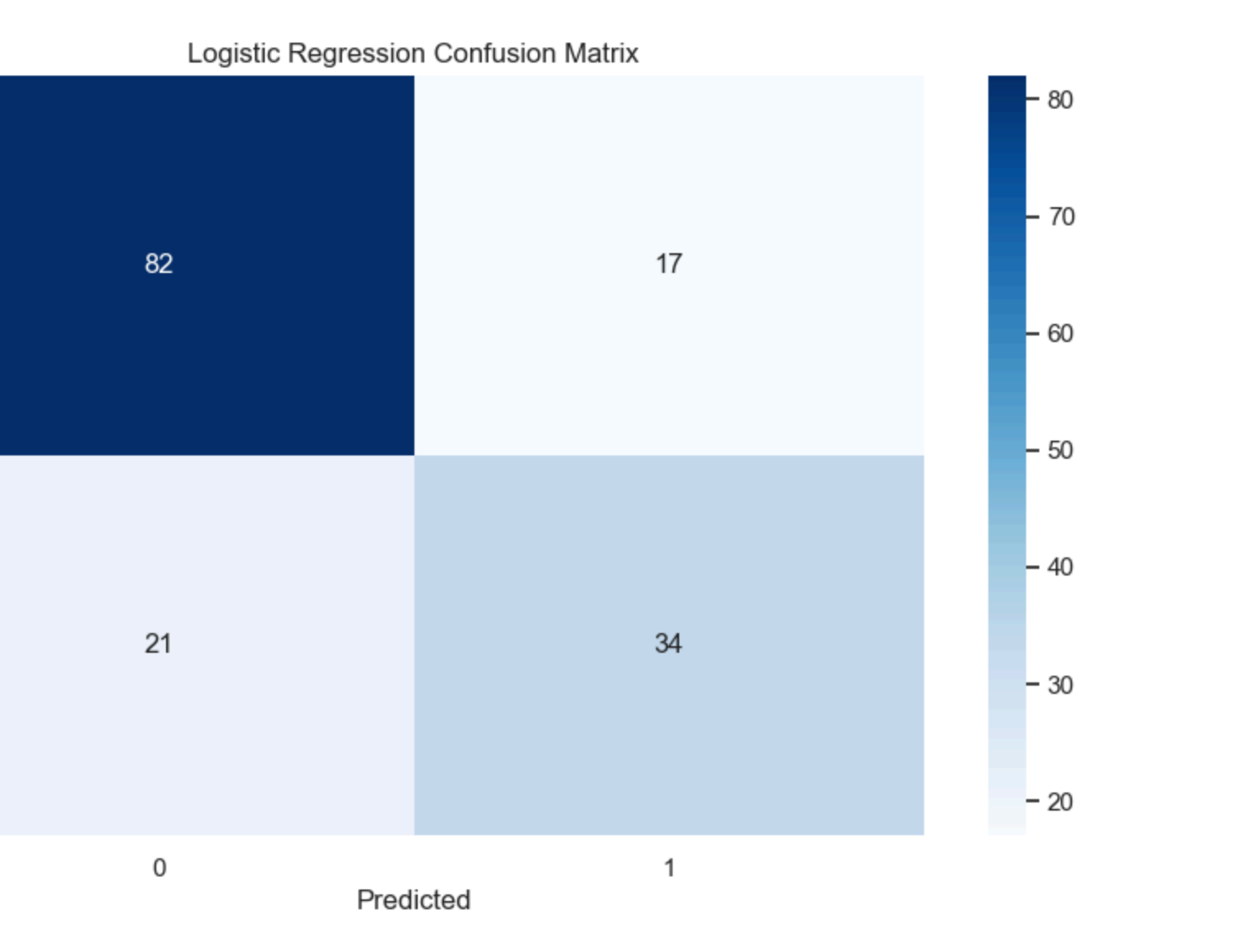
```
In [18]: # b) Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)  # RF can work without scaling
rf_preds = rf_model.predict(X_test)
rf_probs = rf_model.predict_proba(X_test)[:, 1]
```

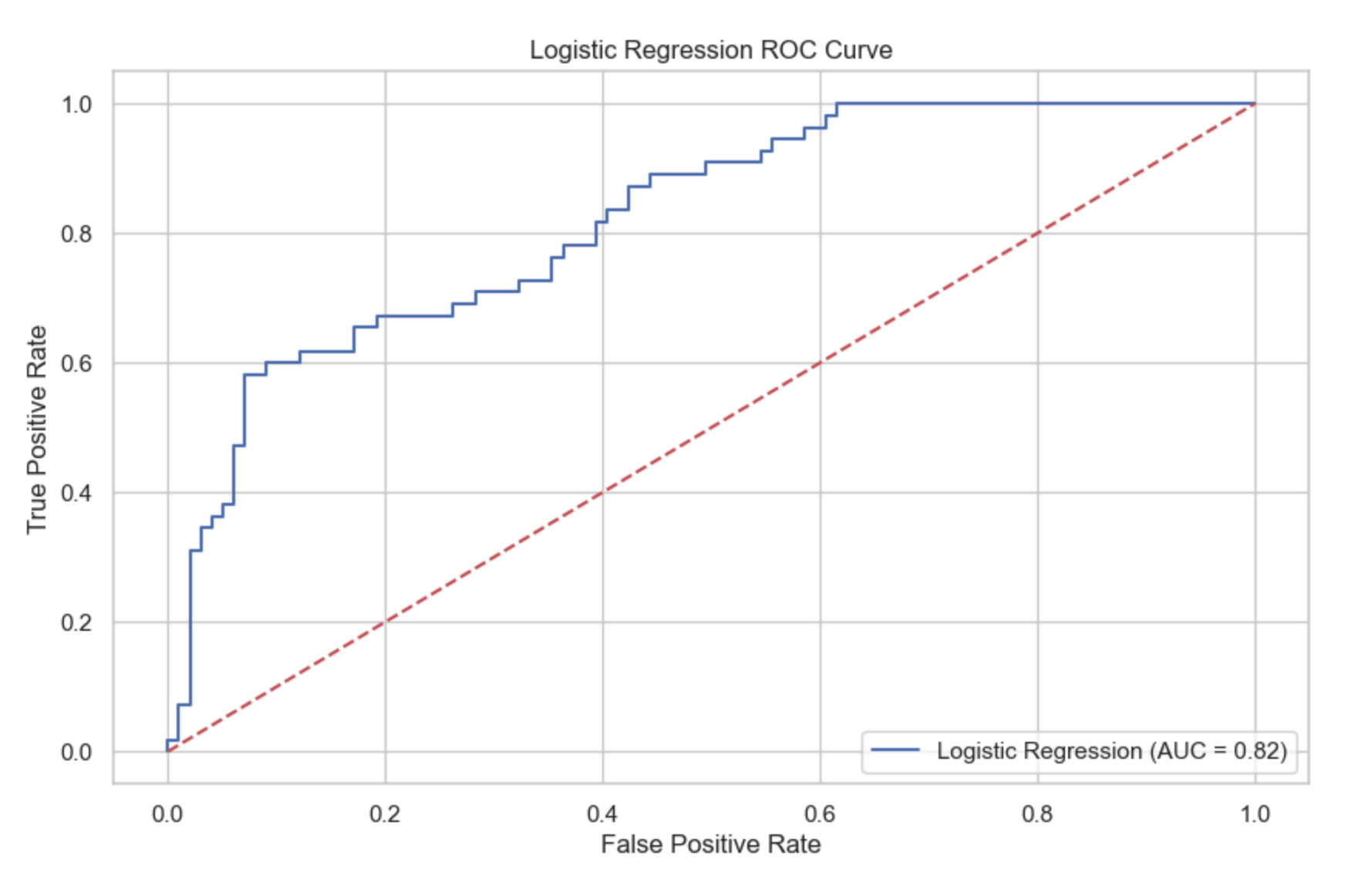
5. Model Evaluation

```
In [19]: def evaluate_model(y_test, preds, probs, model_name):
    print(f"\n--- {model_name} Evaluation ---")
    print("Accuracy:", accuracy_score(y_test, preds))
    print("Classification Report:")
    print(classification_report(y_test, preds))

    # Confusion Matrix
    cm = confusion_matrix(y_test, preds)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'{model_name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    # ROC Curve
    fpr, tpr, thresholds = roc_curve(y_test, probs)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')
    plt.plot([0,1], [0,1],'r--')
    plt.title(f'{model_name} ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.show()
```

```
In [20]: # Evaluate Logistic Regression
evaluate_model(y_test, lr_preds, lr_probs, "Logistic Regression")
```

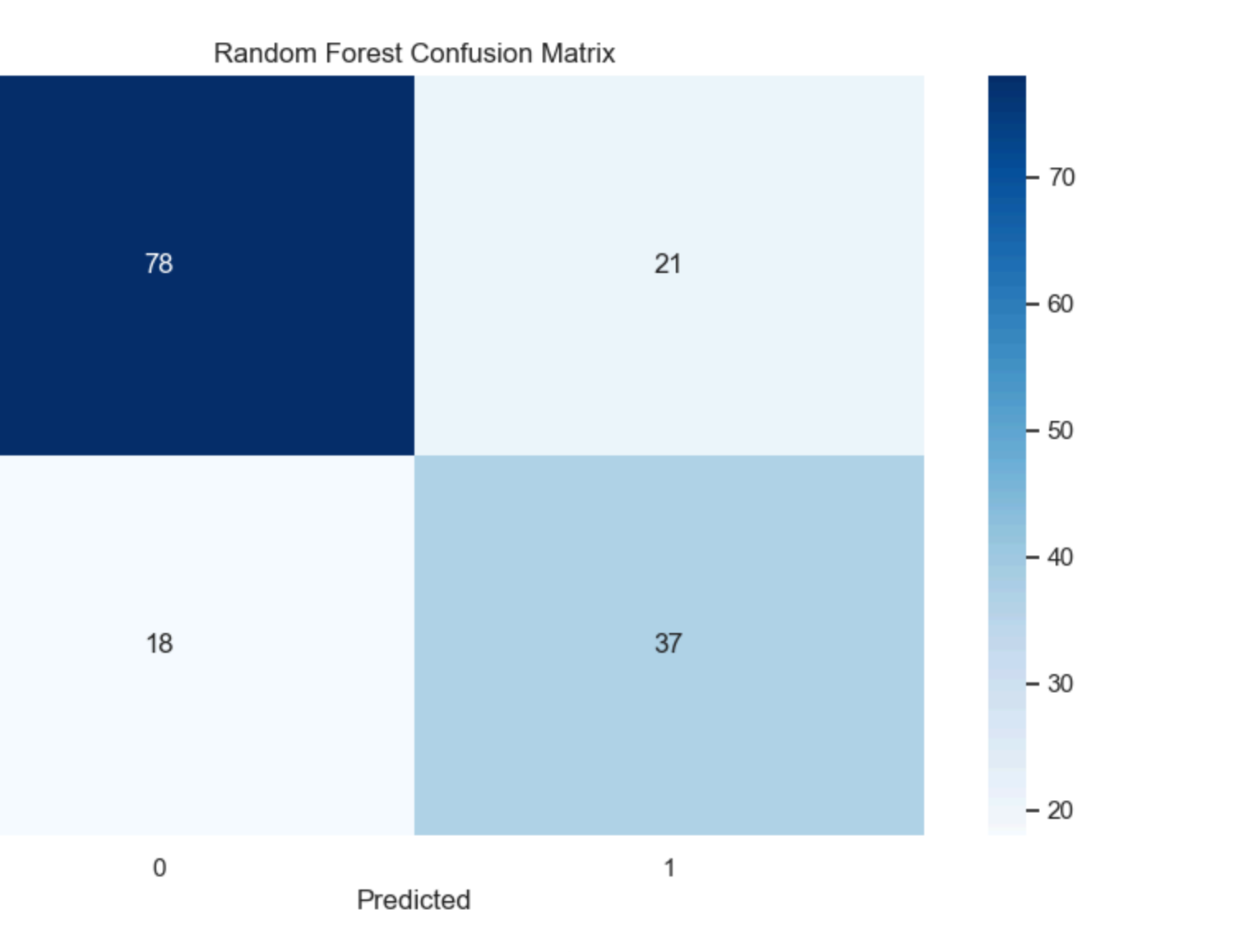
```
--- Logistic Regression Evaluation ---
Accuracy: 0.7532467532467533
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.81        99
           1       0.67      0.62      0.64        55

    accuracy                           0.75       154
   macro avg       0.73      0.72      0.73       154
weighted avg       0.75      0.75      0.75       154
```

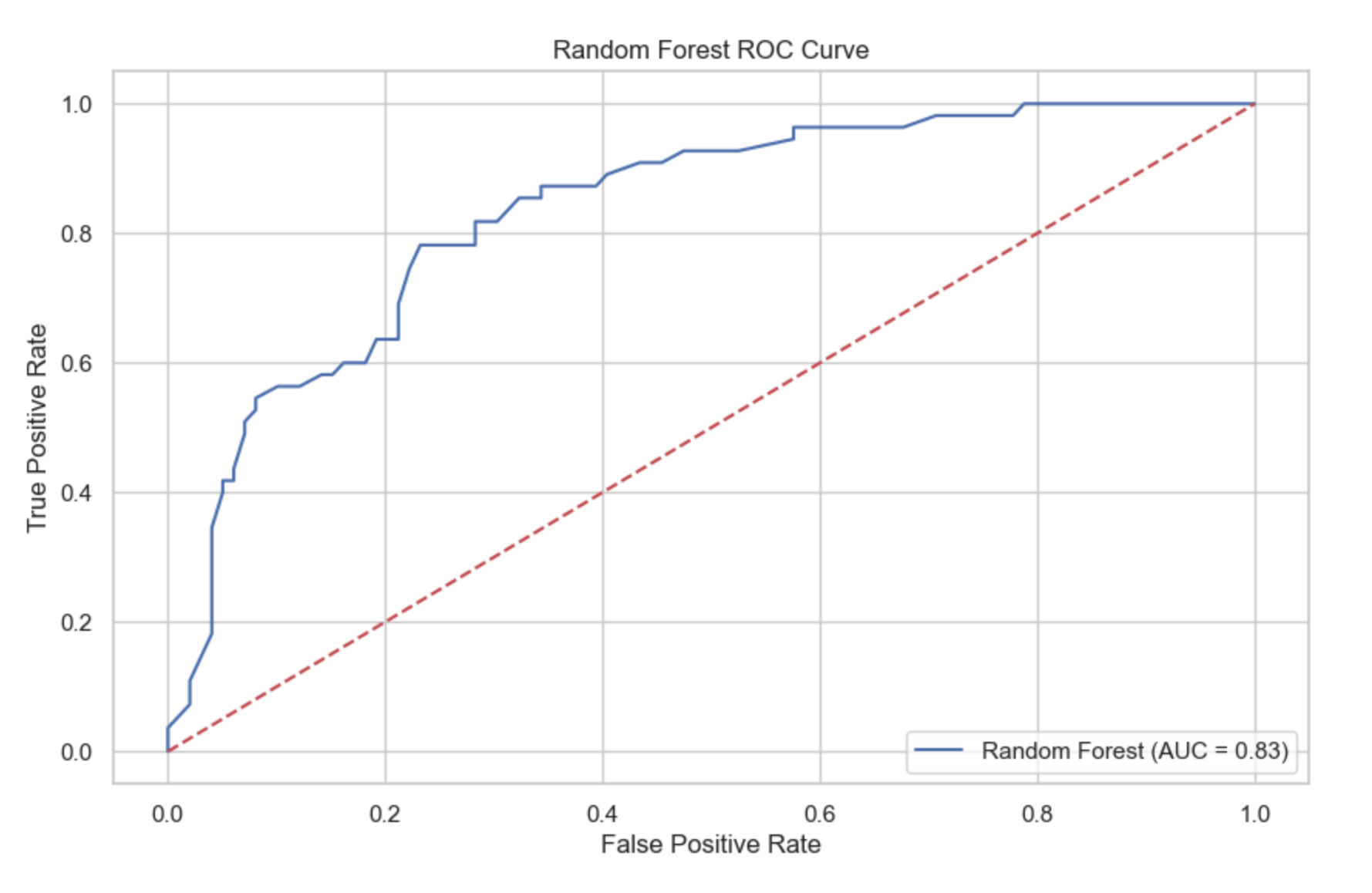





```
In [21]: # Evaluate Random Forest
evaluate_model(y_test, rf_preds, rf_probs, "Random Forest")
```

```
--- Random Forest Evaluation ---
Accuracy: 0.7467532467532467
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.79      0.80        99
           1       0.64      0.67      0.65        55

    accuracy                           0.75       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.75      0.75      0.75       154
```

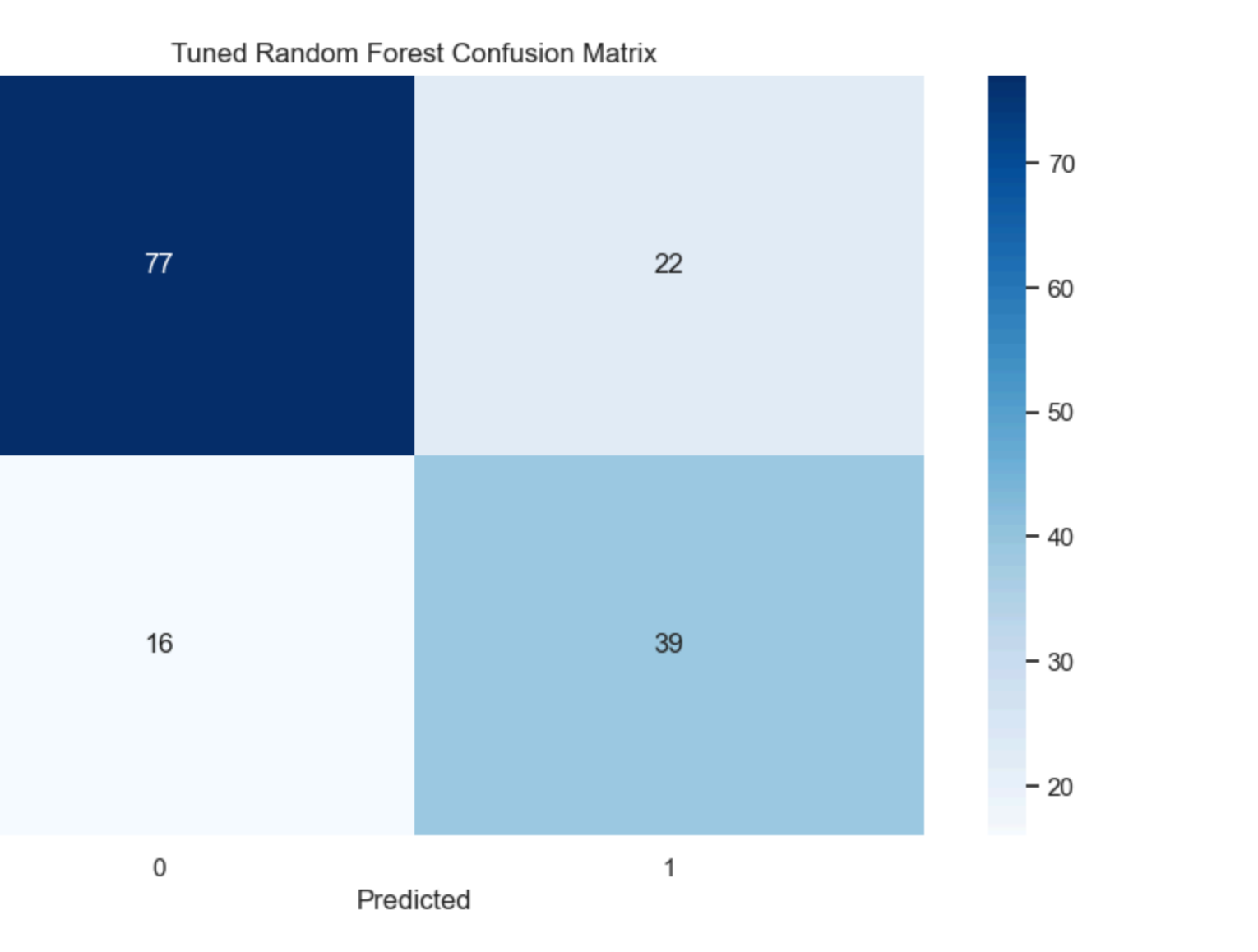


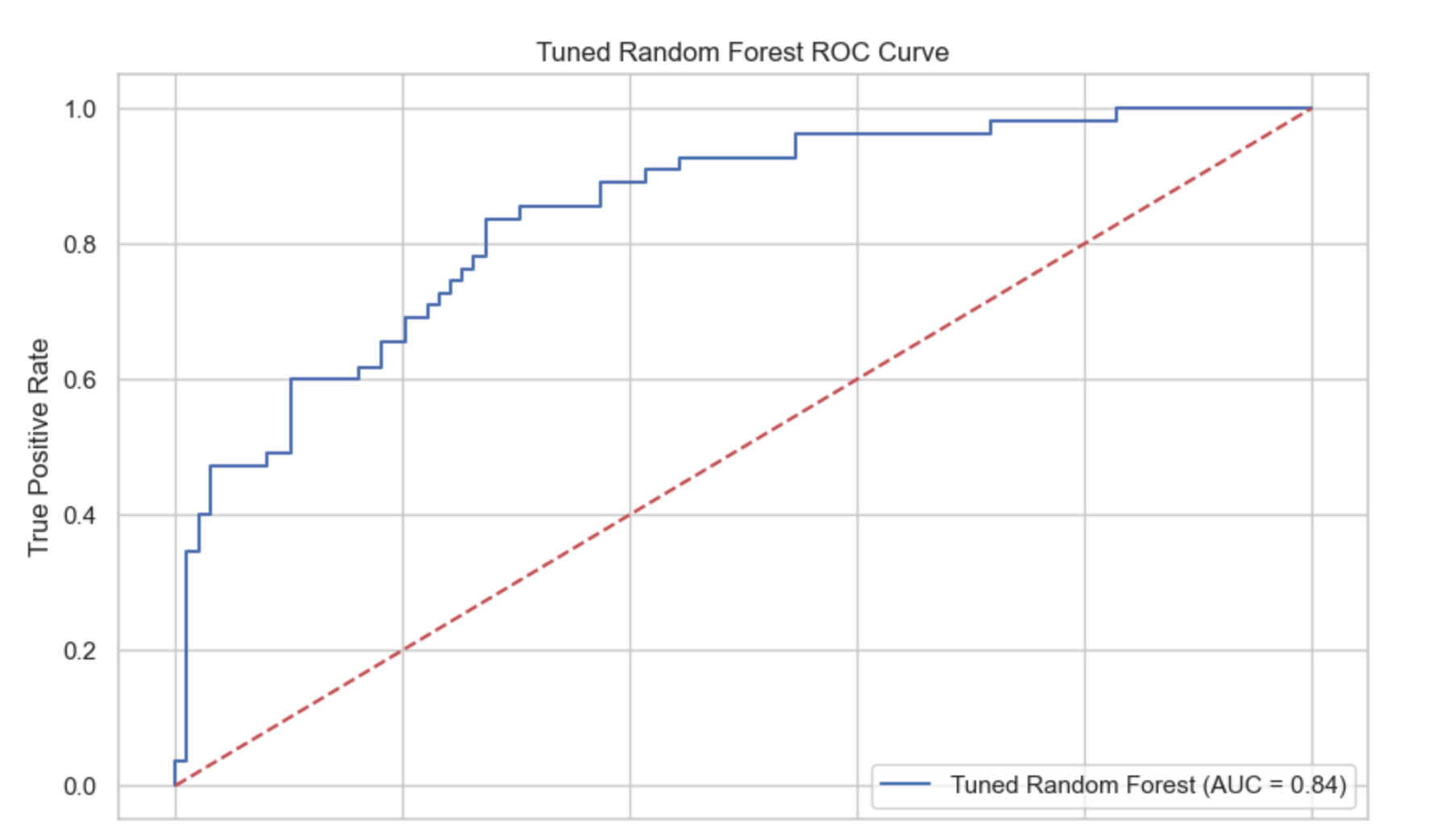


6. Hyperparameter Tuning for Random Forest

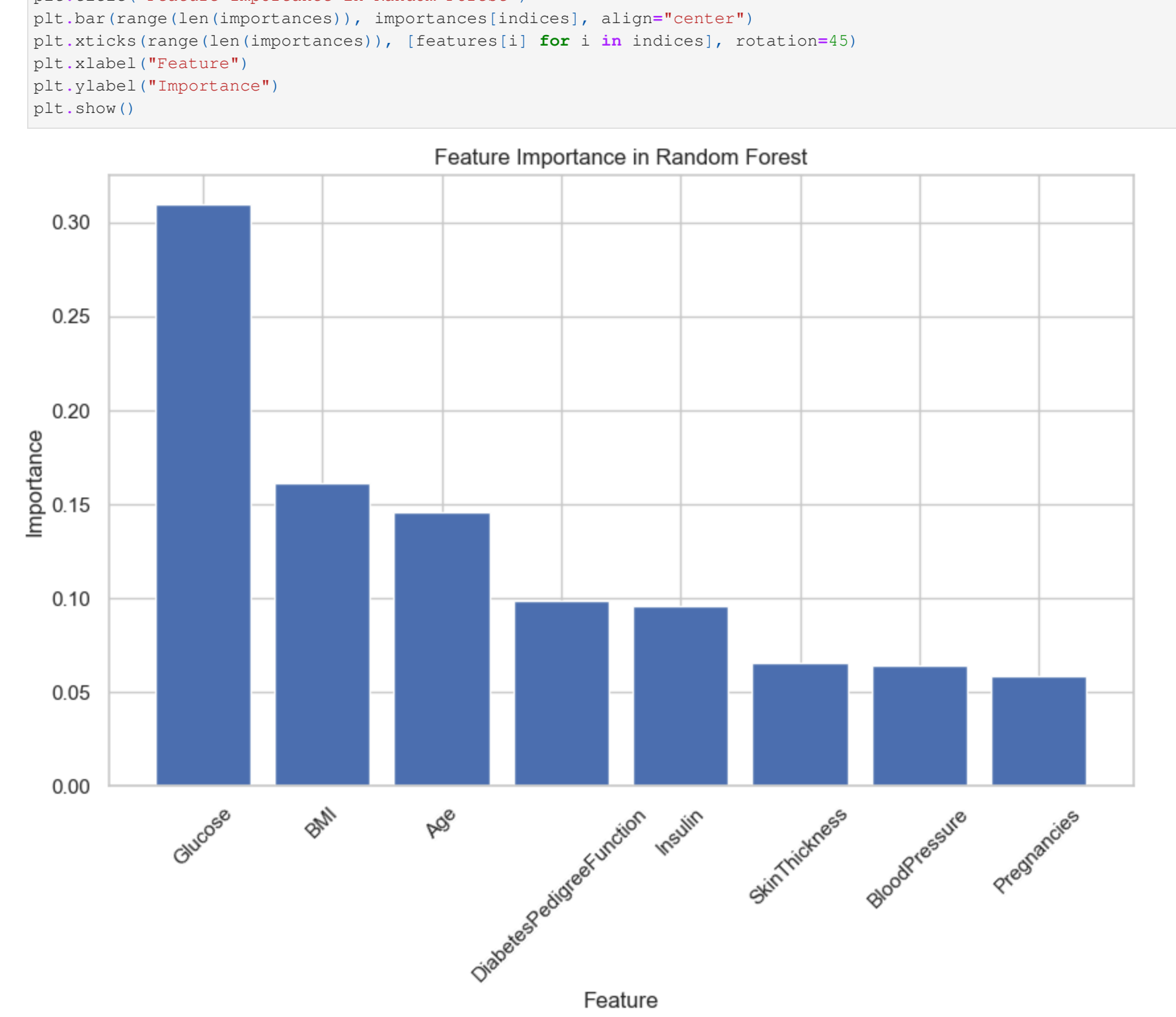
```
In [22]: param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, n_jobs=-1, scoring='accuracy')
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
best_rf_model = grid_search.best_estimator_
best_rf_preds = best_rf_model.predict(X_test)
best_rf_probs = best_rf_model.predict_proba(X_test)[:, 1]
evaluate_model(y_test, best_rf_preds, best_rf_probs, "Tuned Random Forest")
```

```
Best parameters: {'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 50}

--- Tuned Random Forest Evaluation ---
Accuracy: 0.7532467532467533
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.78      0.80        99
           1       0.64      0.71      0.67        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.74       154
weighted avg       0.76      0.75      0.76       154
```

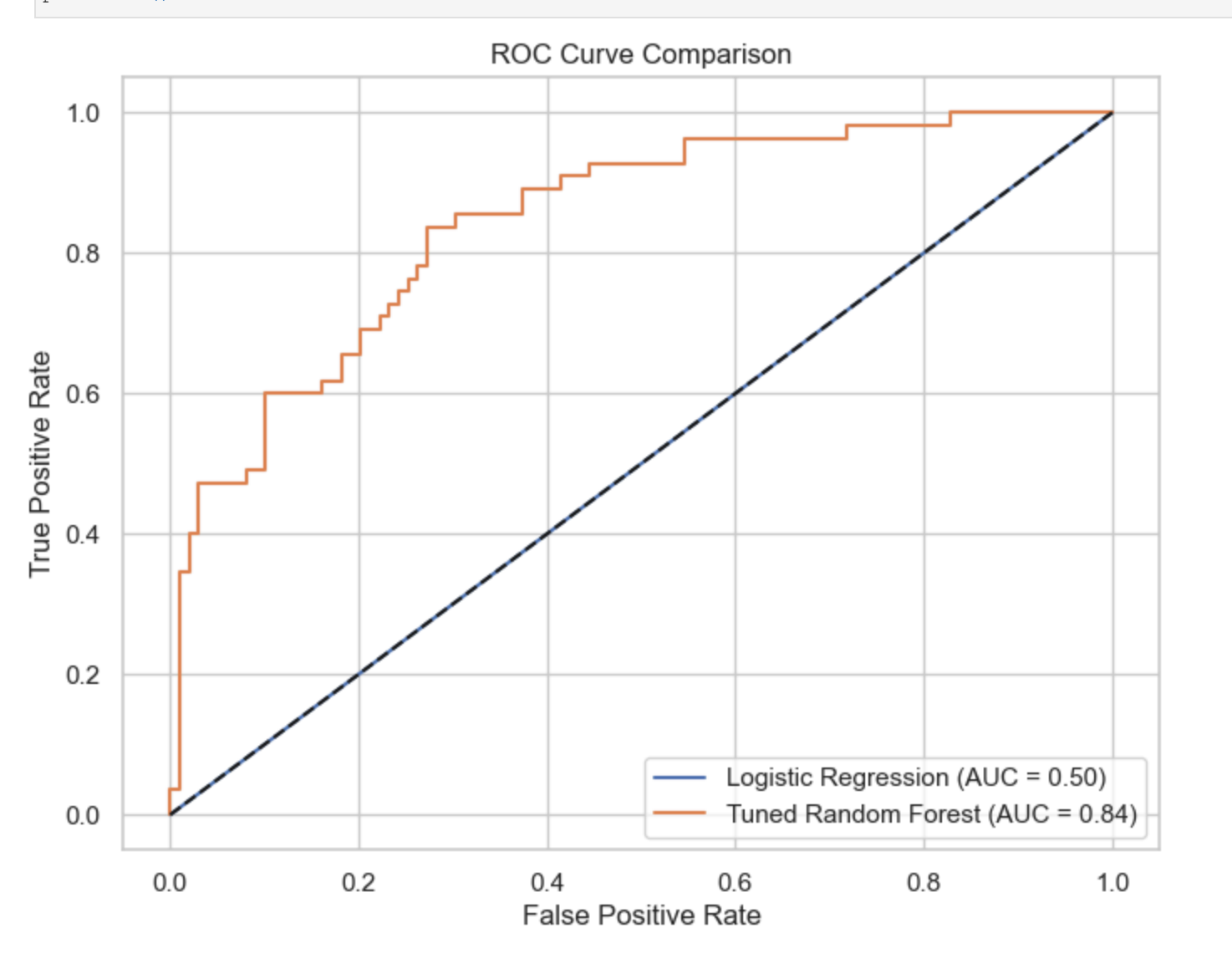




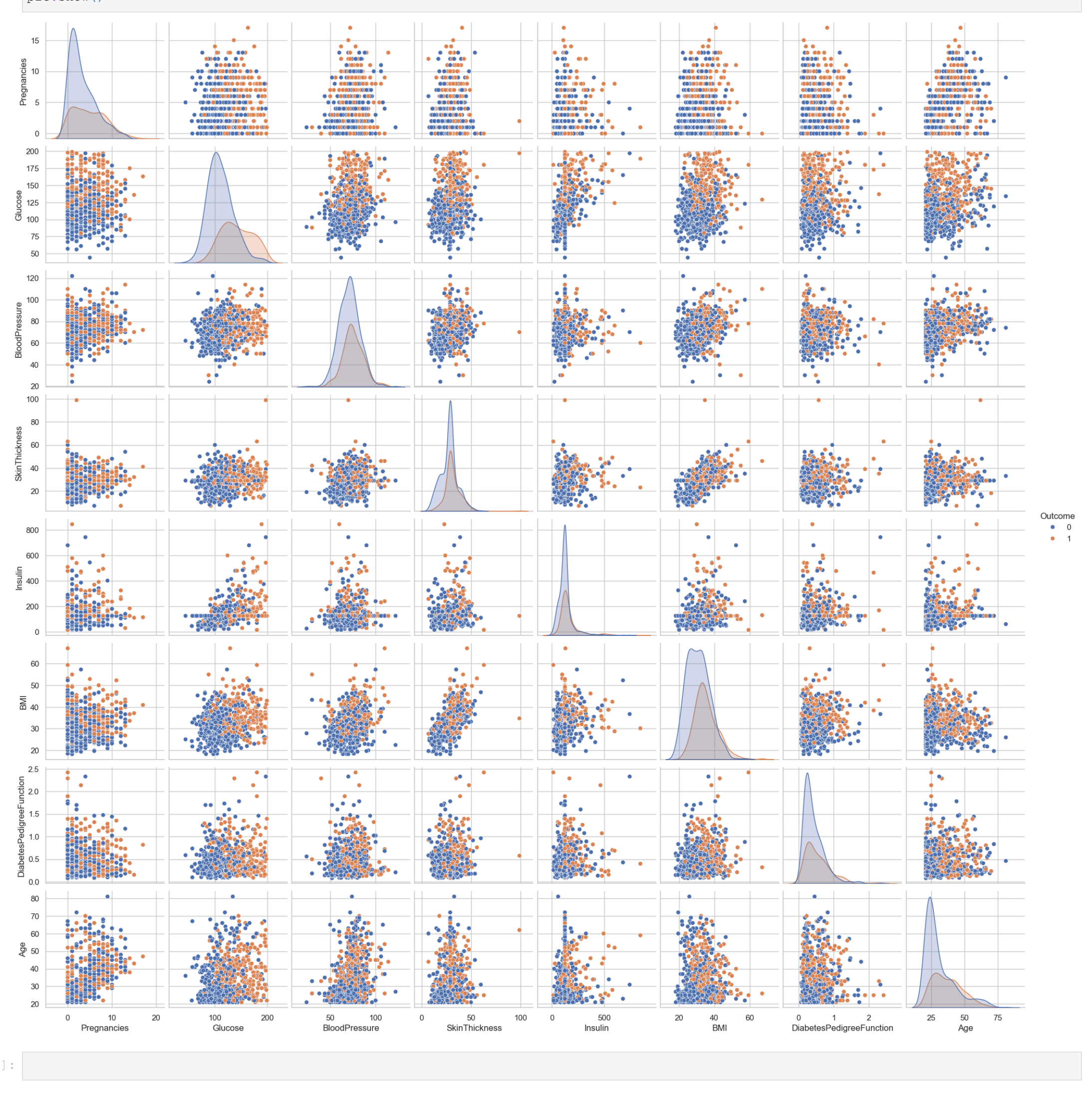

```
In [23]: #Feature Importance Plot
importances = best_rf_model.feature_importances_
indices = np.argsort(importances)[::-1]
features = df.columns[:-1]

plt.figure(figsize=(10,6))
plt.title("Feature Importance in Random Forest")
plt.bar(range(len(importances)), importances[indices], align="center")
plt.xticks(range(len(importances)), [features[i] for i in indices], rotation=45)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.show()
```




```
In [25]: # ROC Curve for Model Comparison
plt.figure(figsize=(8,6))

models = {"Logistic Regression": lr_model, "Tuned Random Forest": best_rf_model}

for name, model in models.items():
    y_probs = model.predict_proba(X_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_probs)
    plt.plot(fpr, tpr, label=f"{name} (AUC = {auc(fpr, tpr):.2f})")

plt.plot([0,1], [0,1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.show()
```




```
In [26]: #Pair Plot
sns.pairplot(df, hue="Outcome", diag_kind="kde")
plt.show()
```




```
In [ ]:
```