AI BASED DIABETES PREDICTION SYSTEM

Phase 3: Development Part 1

Todo:

- Load the dataset
- Preprocess it
- Perform different analysis as needed
- Document all above steps

Step 1: Importing Libraries

We will start by importing all the necessary libraries needed.

```
# Importing necessary libraries

import numpy as
np import pandas
as pd
from sklearn.model_selection import
train_test_splitfrom sklearn.preprocessing import
StandardScaler import matplotlib.pyplot as plt
```

Step 2: Loading the Dataset

Next we will load the diabetes dataset from the provided Kaggle link into our Jupyter Notebook. This dataset contains relevant medical features and information about diabetes.

```
# Loading the diabetes dataset from the URL

dataset_url = "/content/dataset/diabetes.csv"
data = pd.read_csv(dataset_url)
```

Step 3: Data Exploration

We'll conduct initial data exploration to understand the structure and characteristics of the dataset. This includes examining data types, summary statistics, and detecting missing values.

```
# Displaying the first few rows of the dataset
data.head()

# Geting an overview of the dataset
```

```
data.info()

# Checking for missing values
data.isnull().sum()

# Summarizing statistics
data.describe()
```

```
<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
```

	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
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7cd6a7612e90>
```

```
import numpy as np
from google.colab import autoviz

def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,
figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(_df_0, *['Pregnancies'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,
figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(_df_1, *['Glucose'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,
figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(_df_2, *['BloodPressure'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz
```

```
def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,
figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(_df_3, *['SkinThickness'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7cd6a59d9510>
```

```
import numpy as np
from google.colab import autoviz

def scatter_plot(df, x_colname, y_colname, figscale=1, alpha=.8):
  from matplotlib import pyplot as plt
  plt.figure(figsize=(6 * figscale, 6 * figscale))
  df.plot(kind='scatter', x=x_colname, y=y_colname, s=(32 * figscale),
alpha=alpha)
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = scatter_plot(_df_4, *['Pregnancies', 'Glucose'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def scatter_plot(df, x_colname, y_colname, figscale=1, alpha=.8):
  from matplotlib import pyplot as plt
  plt.figure(figsize=(6 * figscale, 6 * figscale))
  df.plot(kind='scatter', x=x_colname, y=y_colname, s=(32 * figscale),
alpha=alpha)
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = scatter_plot(_df_5, *['Glucose', 'BloodPressure'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def scatter_plot(df, x_colname, y_colname, figscale=1, alpha=.8):
  from matplotlib import pyplot as plt
  plt.figure(figsize=(6 * figscale, 6 * figscale))
  df.plot(kind='scatter', x=x_colname, y=y_colname, s=(32 * figscale),
alpha=alpha)
```

```
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = scatter_plot(_df_6, *['BloodPressure', 'SkinThickness'], **{})
chart

import numpy as np
from google.colab import autoviz

def scatter_plot(df, x_colname, y_colname, figscale=1, alpha=.8):
  from matplotlib import pyplot as plt
  plt.figure(figsize=(6 * figscale, 6 * figscale))
  df.plot(kind='scatter', x=x_colname, y=y_colname, s=(32 * figscale),
alpha=alpha)
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = scatter_plot(_df_7, *['SkinThickness', 'Insulin'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7cd6a7472710>
```

```
import numpy as np
from google.colab import autoviz

def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale),
title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(_df_8, *['Pregnancies'], **{})
chart

import numpy as np
from google.colab import autoviz

def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale),
title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(_df_9, *['Glucose'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale),
title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(_df_10, *['BloodPressure'], **{})
chart

import numpy as np
from google.colab import autoviz

def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale),
title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(_df_11, *['SkinThickness'], **{})
chart
```

Step 4: Data Preprocessing

Data preprocessing is crucial to ensure the dataset is suitable for model development. Steps will include:

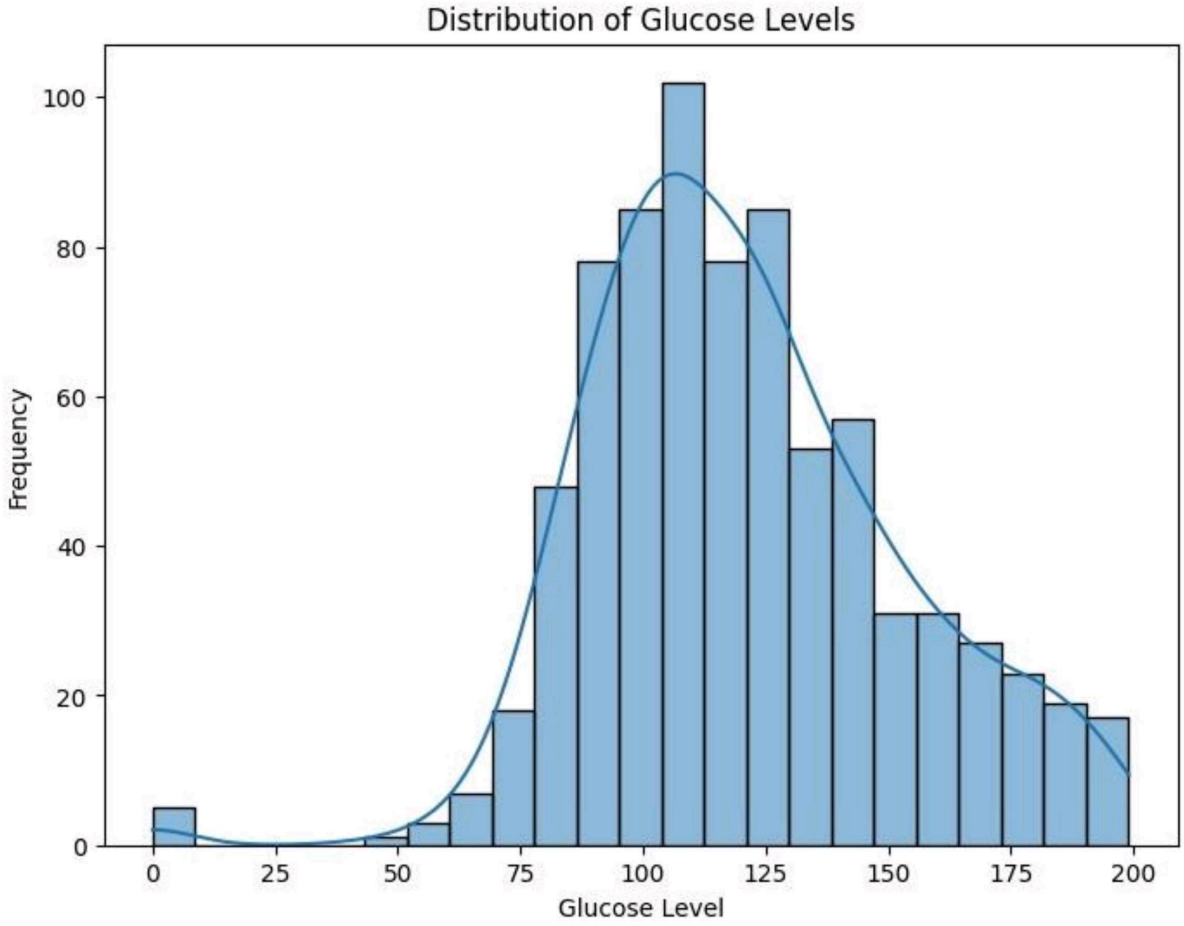
- *Handling missing data through imputation.*
- *Normalizing and scaling numerical features.*
- *Encoding categorical variables.*
- *Balancing class distribution, as diabetes prediction datasets are often imbalanced.*

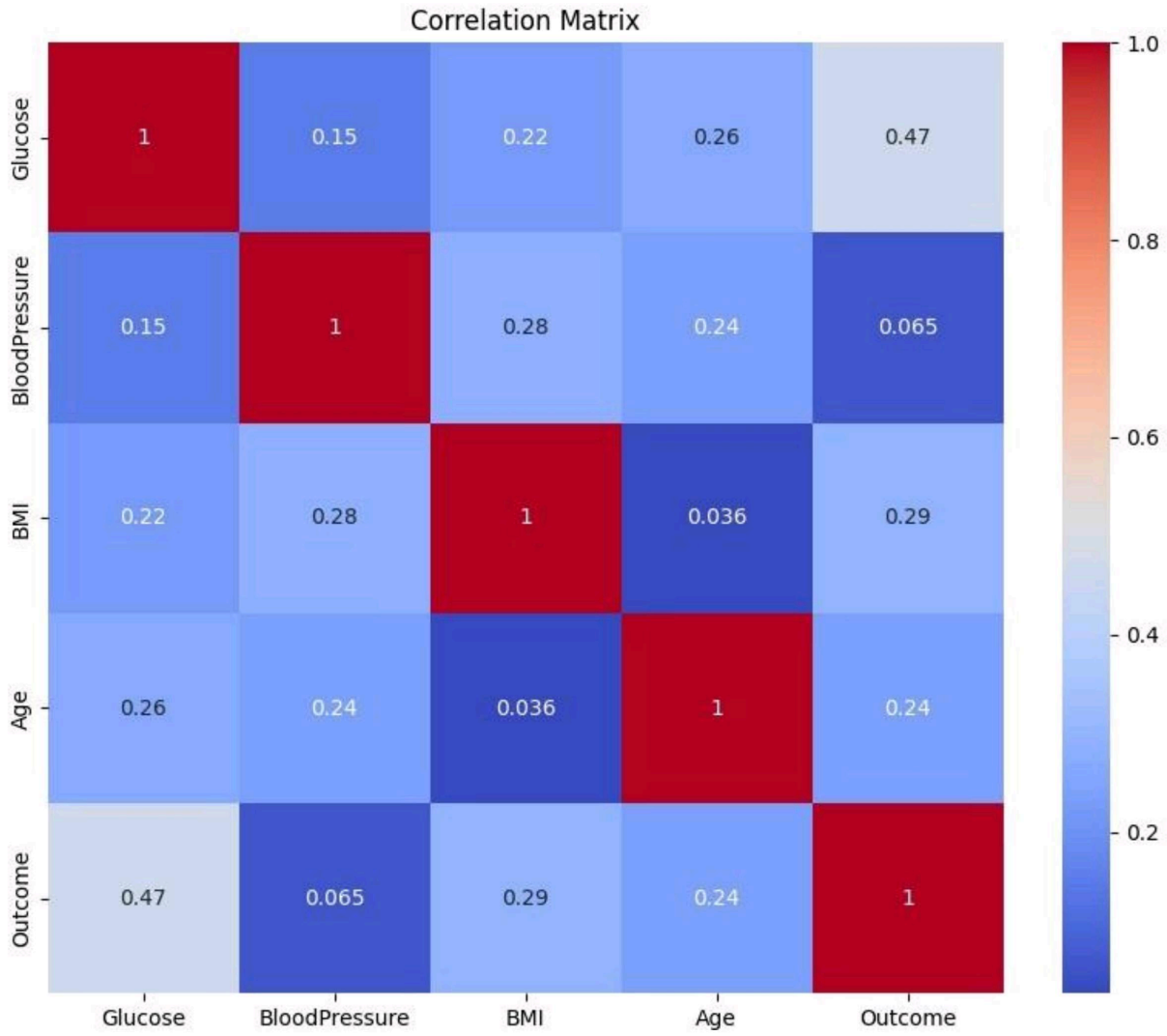
```
# Handle missing values (if any)
# Example: Replace missing values in a specific column (e.g.,
Glucose)with the mean of that column
data['Glucose'].fillna(data['Glucose'].mean(), inplace=True)

# Feature selection: Choose relevant features based on domain
knowledge and data analysis
# Example: Select relevant columns
selected_features = ['Glucose', 'BloodPressure', 'BMI', 'Age',
'Outcome']
data = data[selected_features]
```

Step 5: Data Visualization

We'll create data visualizations to gain insights into the relationships between features and the distribution of diabetes cases. This will help us understand feature importance.

```
# Visualizing the distribution of glucose
levelsplt.figure(figsize=(8, 6))
sns.histplot(data['Glucose'], kde=True)
plt.title('Distribution of Glucose Levels')
plt.xlabel('Glucose Level')
plt.ylabel('Frequency')
plt.show()

# Correlation matrix to see feature relationships
correlation_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```





Step 6: Feature Engineering

We may perform feature engineering to create new features or transform existing ones based on domain knowledge and insights from data visualization.

```
# Example: Creating a new feature for BMI categories
def
    categorize_bmi(bm
    i):if bmi < 18.5:
        return -1 #Underweight
      elif 18.5 <= bmi <
            24.9:
       return 0 #Normal
   elif 25 <= bmi < 29.9:
       return 1 #Overweight
```

```
data['BMI Category'] = data['BMI'].apply(categorize_bmi)
```

Step 7: Data Splitting

We will split the dataset into training and testing subsets to prepare for model development and evaluation.

```
from sklearn.model_selection import train_test_split

# Defining the features (X) and target (y)
X = data.drop('Outcome',
axis=1)y = data['Outcome']

# Spliting the data into training and testing sets (e.g., 80%
train,20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Checking the shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)

X_train shape: (614, 5)
X_test shape: (154, 5)
```

Phase 4: Development Part 2

Todo:

- Model Selection
- Model Training
- Model Evaluation

Step 1: Model Selection

Selecting an appropriate machine learning algorithm is crucial for building an accurate diabetes prediction model. Given that this is a binary classification task (predicting diabetes or not), For this we are going to use Random Forest classifier.

Step 2: Importing Libraries

We will start by importing all the necessary libraries needed.

```
# importing necessary libraries

from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score
from sklearn.metrics import confusion_matrix, roc_curve,
roc_auc_score
```

Step 3: Model Training

Now, we will train the selected machine learning model on the training dataset.

```
# Initializing the Random Forest classifier
clf = RandomForestClassifier(random_state=42)

# Training the Random Forest classifier on the training data
clf.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

Step 4: Model Evaluation

We will evaluate the model's performance using relevant evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

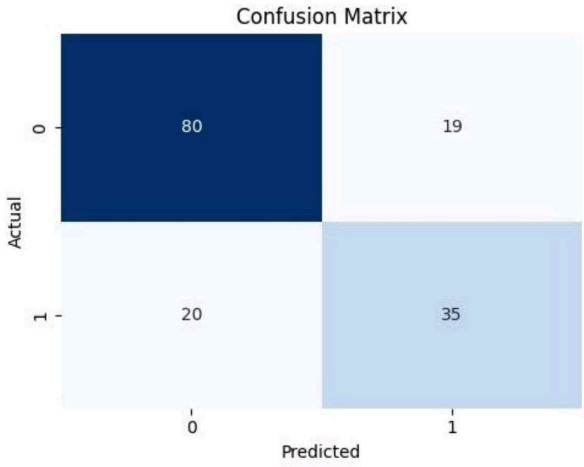
```
# Predicting outcomes on the test set
y_pred = clf.predict(X_test)

# Calculating evaluation metrics
accuracy = accuracy_score(y_test,
y_pred) precision =
precision_score(y_test, y_pred)recall =
recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1])

# Displaying the evaluation
metricsprint("Accuracy:",
accuracy) print("Precision:",
precision) print("Recall:",
recall)
print("F1 Score:", f1)
print("ROC AUC:",
roc_auc)
```

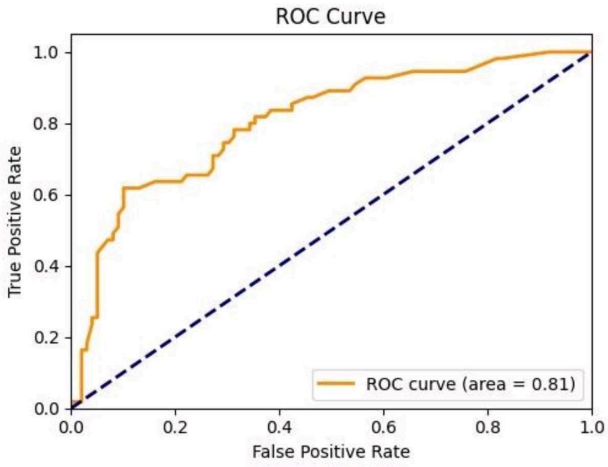
```
Accuracy: 0.7467532467532467
Precision: 0.6481481481481481
```

Step 5: Model Visualization

Visualizing the model's decision boundary can provide insights into how it separates the data points and how well the model is trained.

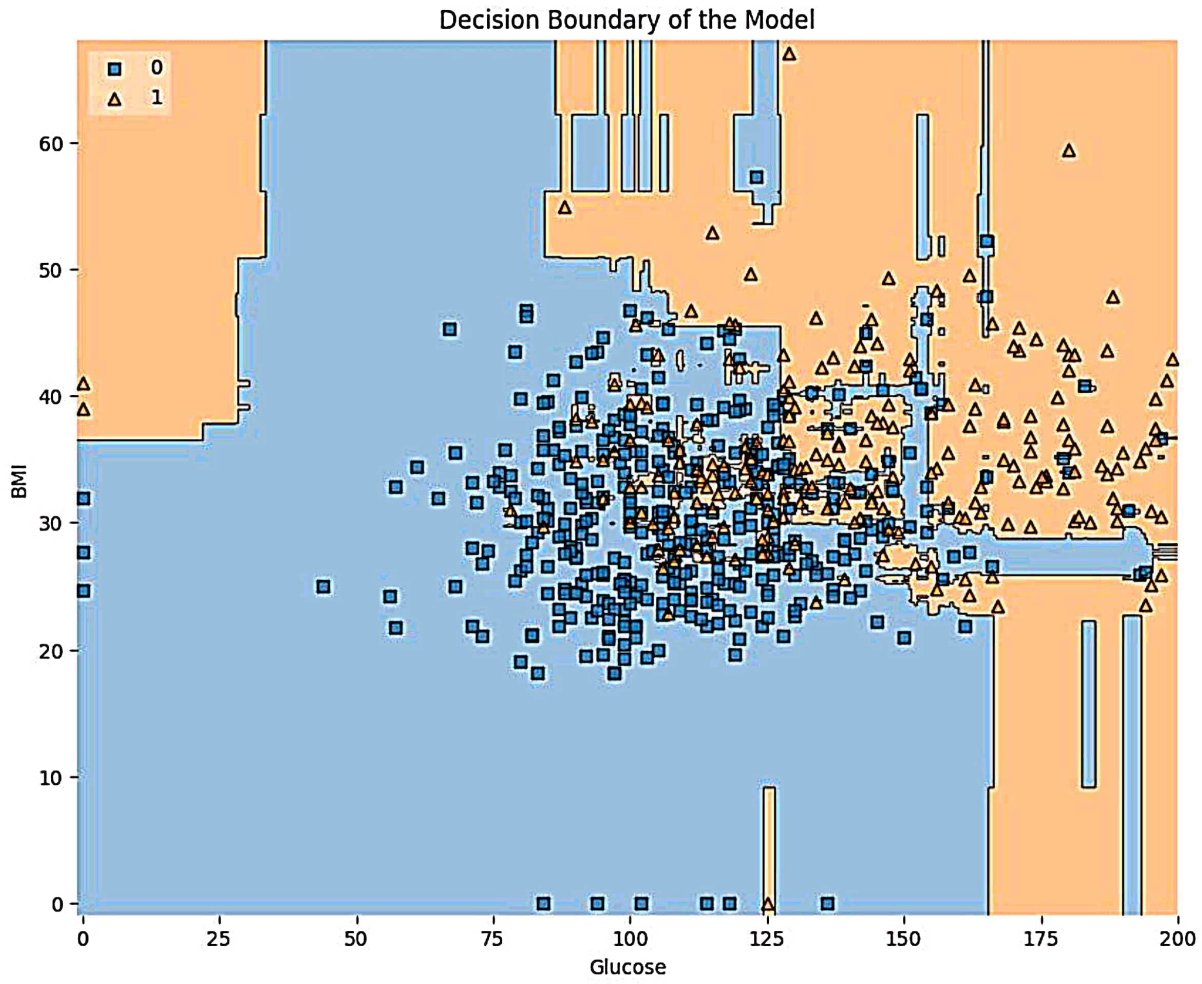
```
# Make predictions on the test data
y_pred = clf.predict(X_test)
y_pred_proba = clf.predict_proba(X_test)[:, 1]

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Plot Confusion Matrix
plt.figure(figsize=(10, 4))
plt.subplot(121)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

# Plot ROC Curve
plt.subplot(122)
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')

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






```
# Example: Visualize the decision boundary (works for binary
classification)
from mlxtend.plotting import plot_decision_regions

# Fit the model on a subset of features (2 features for
visualizationpurposes)
clf.fit(X_train[['Glucose', 'BMI']], y_train)

# Plot the decision boundary
plt.figure(figsize=(10, 8))
plot_decision_regions(X=np.array(X_train[['Glucose', 'BMI']]),
y=np.array(y_train), clf=clf, legend=2)
plt.title("Decision Boundary of the Model")
plt.xlabel("Glucose")
plt.ylabel("BM
I")plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
```



Jayasri A
Nandha college of technology