```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import pandas as pd
```

```
df = pd.read_csv('/content/drive/MyDrive/Lab performance 2 /Pima Indians Diabetes Dataset.csv')
```

```
df.head(10)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

Next steps: Generate code with df | New interactive sheet

Handle Missing Values (including zero as missing)

```
columns_to_impute = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

for col in columns_to_impute:
    df[col] = df[col].replace(0, np.nan)
    df[col] = df[col].fillna(df[col].mean())

print("Missing values (zeros replaced by NaN and then imputed with mean):")
```

```
Missing values (zeros replaced by NaN and then imputed with mean):
```

```
print(df.isnull().sum())
```

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

Remove Duplicate Rows

```
initial_rows = len(df)
df.drop_duplicates(inplace=True)
rows_after_duplicates = len(df)

print(f'Initial number of rows: {initial_rows}')
print(f'Number of rows after removing duplicates: {rows_after_duplicates}')
```

```
Initial number of rows: 768
Number of rows after removing duplicates: 768
```

Correct Data Types

```
print("Current data types and non-null counts:")
df.info()
```

```
Current data types and non-null counts:
<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    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 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(6), int64(3)
memory usage: 54.1 KB
```

```
print("\nFirst few rows of the cleaned DataFrame:")
print(df.head())

print("\nInfo of the cleaned DataFrame:")
df.info()
```

```
First few rows of the cleaned DataFrame:
   Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin   BMI  \
0            6    148.0           72.0       35.00000  155.548223  33.6
1            1     85.0           66.0       29.00000  155.548223  26.6
2            8    183.0           64.0       29.15342  155.548223  23.3
3            1     89.0           66.0       23.00000   94.000000  28.1
4            0    137.0           40.0       35.00000  168.000000  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

Info of the cleaned DataFrame:
<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    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 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(6), int64(3)
memory usage: 54.1 KB
```

```
Start coding or generate with AI.
```

```
print("Current data types and non-null counts:")
df.info()
```

```
Current data types and non-null counts:
<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    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 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(6), int64(3)
memory usage: 54.1 KB
```

```
Start coding or generate with AI.
```

I take Complete help from AI for this portion.Couse, I don't find any duplicate row or column in this dataset

```
print("Info of the cleaned DataFrame:")
df.info()
```

```
Info of the cleaned DataFrame:
<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    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 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(6), int64(3)
memory usage: 54.1 KB
```

One-Hot Encoding

```
# Generic One-Hot Encoding for future datasets

df_future = df.copy()

print("Original DataFrame Info (before potential encoding):")
df_future.info()

# Identify categorical columns
categorical_cols = df_future.select_dtypes(include=['object', 'category']).columns

if len(categorical_cols) > 0:
    print(f"\nCategorical columns identified for One-Hot Encoding: {list(categorical_cols)}")
    # Application of One-Hot Encoding
    df_future_encoded = pd.get_dummies(df_future, columns=categorical_cols, drop_first=True)
    print("\nDataFrame after One-Hot Encoding (first 5 rows):")
    print(df_future_encoded.head())
    print("\nDataFrame Info (after potential encoding):")
    df_future_encoded.info()
else:
    print("\nNo 'object' or 'category' type columns found for One-Hot Encoding in this DataFrame.")
    print("\nDataFrame Info (no encoding performed):")
    df_future.info()
```

```
Original DataFrame Info (before potential encoding):
<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    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
```

```
 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(6), int64(3)
memory usage: 54.1 KB

No 'object' or 'category' type columns found for One-Hot Encoding in this DataFrame.

DataFrame Info (no encoding performed):
<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    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 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(6), int64(3)
memory usage: 54.1 KB
```

```
Start coding or generate with AI.
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Create a copy of the DataFrame to apply StandardScaler
df_standard_scaled = df.copy()

# Initialize the StandardScaler
scaler = StandardScaler()

# Identify columns to scale (all numerical columns except the 'Outcome' which is our target)
# Let's assume all columns except 'Outcome' should be scaled for now.
columns_to_scale = df_standard_scaled.columns.drop('Outcome')

# Apply StandardScaler to the selected columns
df_standard_scaled[columns_to_scale] = scaler.fit_transform(df_standard_scaled[columns_to_scale])

print("DataFrame after StandardScaler (first 5 rows):")
print(df_standard_scaled.head())

print("\nDescriptive statistics after StandardScaler:")
print(df_standard_scaled.describe())
```

```
DataFrame after StandardScaler (first 5 rows):
   Pregnancies   Glucose  BloodPressure  SkinThickness       Insulin  \
0     0.639947  0.865108      -0.033518   6.655021e-01 -3.345079e-16
1    -0.844885 -1.206162      -0.529859  -1.746338e-02 -3.345079e-16
2     1.233880  2.015813      -0.695306   8.087936e-16 -3.345079e-16
3    -0.844885 -1.074652      -0.529859  -7.004289e-01 -7.243887e-01
4    -1.141852  0.503458      -2.680669   6.655021e-01  1.465506e-01

        BMI  DiabetesPedigreeFunction       Age  Outcome
0  0.166292                  0.468492  1.425995        1
1 -0.852531                 -0.365061 -0.190672        0
2 -1.332833                  0.604397 -0.105584        1
3 -0.634212                 -0.920763 -1.041549        0
4  1.548980                  5.484909 -0.020496        1

Descriptive statistics after StandardScaler:
        Pregnancies       Glucose  BloodPressure  SkinThickness       Insulin  \
count  7.680000e+02  7.680000e+02   7.680000e+02   7.680000e+02  7.680000e+02
mean  -6.476301e-17 -3.561966e-16   6.915764e-16   7.956598e-16 -3.330669e-16
std    1.000652e+00  1.000652e+00   1.000652e+00   1.000652e+00  1.000652e+00
min   -1.141852e+00 -2.554131e+00  -4.004245e+00  -2.521670e+00 -1.665945e+00
25%   -8.448851e-01 -7.212214e-01  -6.953060e-01  -4.727737e-01 -4.007289e-01
50%   -2.509521e-01 -1.540881e-01  -1.675912e-02   8.087936e-16 -3.345079e-16
75%    6.399473e-01  6.103090e-01   6.282695e-01   3.240194e-01 -3.345079e-16
max    3.906578e+00  2.541850e+00   4.102655e+00   7.950467e+00  8.126238e+00
```

```
                BMI  DiabetesPedigreeFunction           Age     Outcome
count  7.680000e+02              7.680000e+02  7.680000e+02  768.000000
mean   3.515706e-16              2.451743e-16  1.931325e-16    0.348958
std    1.000652e+00              1.000652e+00  1.000652e+00    0.476951
min   -2.075119e+00             -1.189553e+00 -1.041549e+00    0.000000
25%   -7.215397e-01             -6.889685e-01 -7.862862e-01    0.000000
50%   -8.363615e-03             -3.001282e-01 -3.608474e-01    0.000000
75%    6.029301e-01              4.662269e-01  6.602056e-01    1.000000
max    5.042087e+00              5.883565e+00  4.063716e+00    1.000000
```

MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Create another copy of the DataFrame to apply MinMaxScaler
df_minmax_scaled = df.copy()

# Initialize the MinMaxScaler
scaler_minmax = MinMaxScaler()

# Identify columns to scale (all numerical columns except the 'Outcome')
columns_to_scale = df_minmax_scaled.columns.drop('Outcome')

# Apply MinMaxScaler to the selected columns
df_minmax_scaled[columns_to_scale] = scaler_minmax.fit_transform(df_minmax_scaled[columns_to_scale])

print("DataFrame after MinMaxScaler (first 5 rows):")
print(df_minmax_scaled.head())

print("\nDescriptive statistics after MinMaxScaler:")
print(df_minmax_scaled.describe())
```

```
DataFrame after MinMaxScaler (first 5 rows):
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.352941  0.670968       0.489796       0.304348  0.170130  0.314928
1     0.058824  0.264516       0.428571       0.239130  0.170130  0.171779
2     0.470588  0.896774       0.408163       0.240798  0.170130  0.104294
3     0.058824  0.290323       0.428571       0.173913  0.096154  0.202454
4     0.000000  0.600000       0.163265       0.304348  0.185096  0.509202

   DiabetesPedigreeFunction       Age  Outcome
0                  0.234415  0.483333        1
1                  0.116567  0.166667        0
2                  0.253629  0.183333        1
3                  0.038002  0.000000        0
4                  0.943638  0.200000        1

Descriptive statistics after MinMaxScaler:
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      0.226180    0.501205       0.493930       0.240798    0.170130
std       0.198210    0.196361       0.123432       0.095554    0.102189
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       0.058824    0.359677       0.408163       0.195652    0.129207
50%       0.176471    0.470968       0.491863       0.240798    0.170130
75%       0.352941    0.620968       0.571429       0.271739    0.170130
max       1.000000    1.000000       1.000000       1.000000    1.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean     0.291564                  0.168179    0.204015    0.348958
std      0.140596                  0.141473    0.196004    0.476951
min      0.000000                  0.000000    0.000000    0.000000
25%      0.190184                  0.070773    0.050000    0.000000
50%      0.290389                  0.125747    0.133333    0.000000
75%      0.376278                  0.234095    0.333333    1.000000
max      1.000000                  1.000000    1.000000    1.000000
```

```
Start coding or generate with AI.
```

Train-Test Split

```
from sklearn.model_selection import train_test_split
```

```
# For this example, let's use the df_standard_scaled DataFrame.
# You can choose df_minmax_scaled if you prefer.

X = df_standard_scaled.drop('Outcome', axis=1) # Features (all columns except 'Outcome')
y = df_standard_scaled['Outcome'] # Target variable ('Outcome')

# Split the data into training and testing sets
# test_size=0.20 means 20% of the data will be used for testing, and 80% for training.
# random_state is used to ensure reproducibility of the split. (You'll get the same split every time if you use the same number
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

print(f"Shape of X_train (training features): {X_train.shape}")
print(f"Shape of X_test (testing features): {X_test.shape}")
print(f"Shape of y_train (training target): {y_train.shape}")
print(f"Shape of y_test (testing target): {y_test.shape}")

print("\nFirst 5 rows of X_train:")
print(X_train.head())

print("\nFirst 5 rows of y_train:")
print(y_train.head())
```

```
Shape of X_train (training features): (614, 8)
Shape of X_test (testing features): (154, 8)
Shape of y_train (training target): (614,)
Shape of y_test (testing target): (154,)

First 5 rows of X_train:
     Pregnancies   Glucose  BloodPressure  SkinThickness       Insulin  \
60     -0.547919 -1.239039   1.175571e-15   8.087936e-16 -3.345079e-16
618     1.530847 -0.318475   7.937164e-01  -5.866013e-01 -3.345079e-16
346    -0.844885  0.569212  -2.184328e+00  -1.155739e+00 -8.538527e-01
294    -1.141852  1.292513  -1.853434e+00   8.087936e-16 -3.345079e-16
231     0.639947  0.404826   6.282695e-01   8.931573e-01  2.523979e+00

          BMI  DiabetesPedigreeFunction       Age
60   0.000000                 -0.507006 -1.041549
618 -0.619657                  2.446670  1.425995
346 -0.546884                  0.550035 -0.956462
294 -1.536598                 -0.658012  2.702312
231  2.000173                 -0.706334  1.085644

First 5 rows of y_train:
60     0
618    1
346    0
294    0
231    1
Name: Outcome, dtype: int64
```

```
Start coding or generate with AI.
```

Logistic Regression

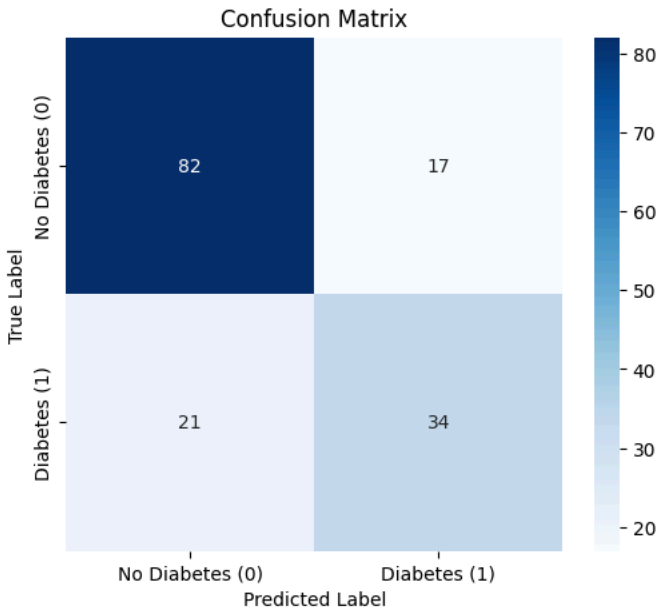
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# 1. Initialize the Logistic Regression model
model = LogisticRegression(random_state=42, solver='liblinear')
# 2. Train the model using training data

print("Training the Logistic Regression model...")
model.fit(X_train, y_train)
print("Model training complete!")

# 3. Make predictions on the test data
y_pred = model.predict(X_test)

# 4. Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"\nModel Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", report)
```

```
Training the Logistic Regression model...
Model training complete!

Model Accuracy: 0.75

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
```

```
Start coding or generate with AI.
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Diabetes (0)', 'Diabetes (1)'],
            yticklabels=['No Diabetes (0)', 'Diabetes (1)'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



Precision, Recall, F1-score

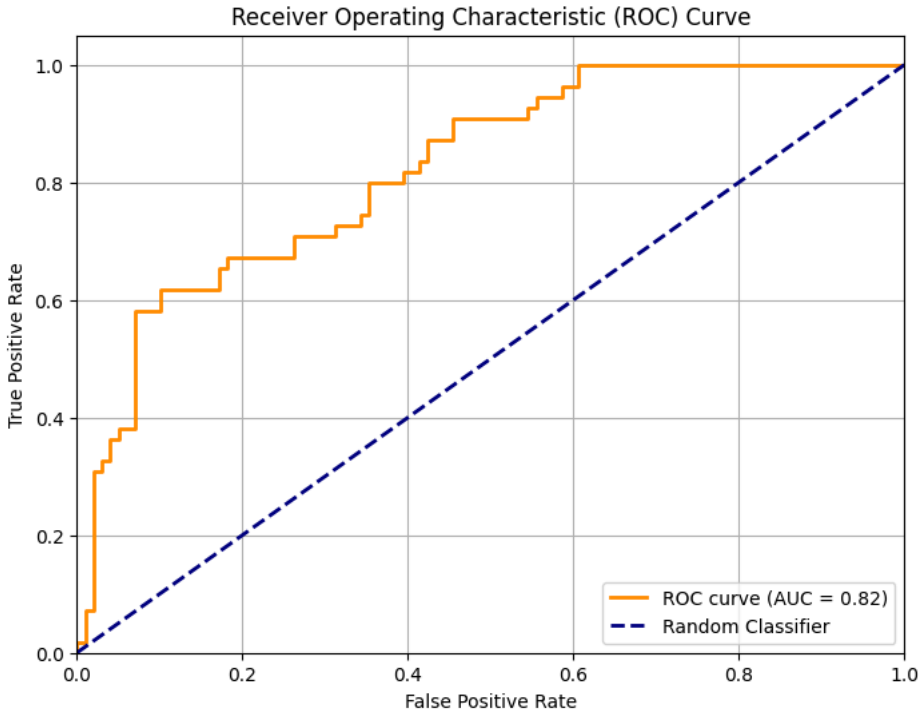
```
from sklearn.metrics import classification_report

print("Classification Report (reiterated for clarity):\n", classification_report(y_test, y_pred))
```

```
Classification Report (reiterated for clarity):
              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
```

ROC Curve

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

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

# Calculate ROC curve values
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate AUC score
auc = roc_auc_score(y_test, y_pred_proba)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```




```
Start coding or generate with AI.
```

K-Nearest Neighbors (KNN)

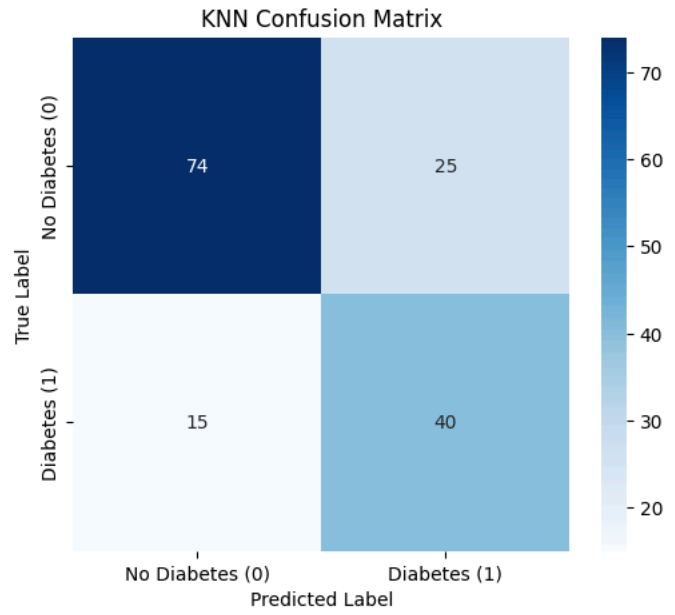
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# 1. Initialize the KNN model
model_knn = KNeighborsClassifier(n_neighbors=5)
```

```
# 2. Train the model using our training data
print("Training the K-Nearest Neighbors model...")
model_knn.fit(X_train, y_train)
print("KNN Model training complete!")

# 3. Make predictions on the test data
y_pred_knn = model_knn.predict(X_test)

# 4. Evaluate the KNN model's performance
accuracy_knn = accuracy_score(y_test, y_pred_knn)
report_knn = classification_report(y_test, y_pred_knn)

print(f"\nKNN Model Accuracy: {accuracy_knn:.2f}")
print("\nKNN Classification Report:\n", report_knn)
```

```
Training the K-Nearest Neighbors model...
KNN Model training complete!

KNN Model Accuracy: 0.74

KNN Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.75      0.79        99
           1       0.62      0.73      0.67        55

    accuracy                           0.74       154
   macro avg       0.72      0.74      0.73       154
weighted avg       0.75      0.74      0.74       154
```

.Confusion Matrix for KNN

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the confusion matrix for KNN
cm_knn = confusion_matrix(y_test, y_pred_knn)

# Visualize the confusion matrix for KNN
plt.figure(figsize=(6, 5))
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Diabetes (0)', 'Diabetes (1)'],
            yticklabels=['No Diabetes (0)', 'Diabetes (1)'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('KNN Confusion Matrix')
plt.show()
```



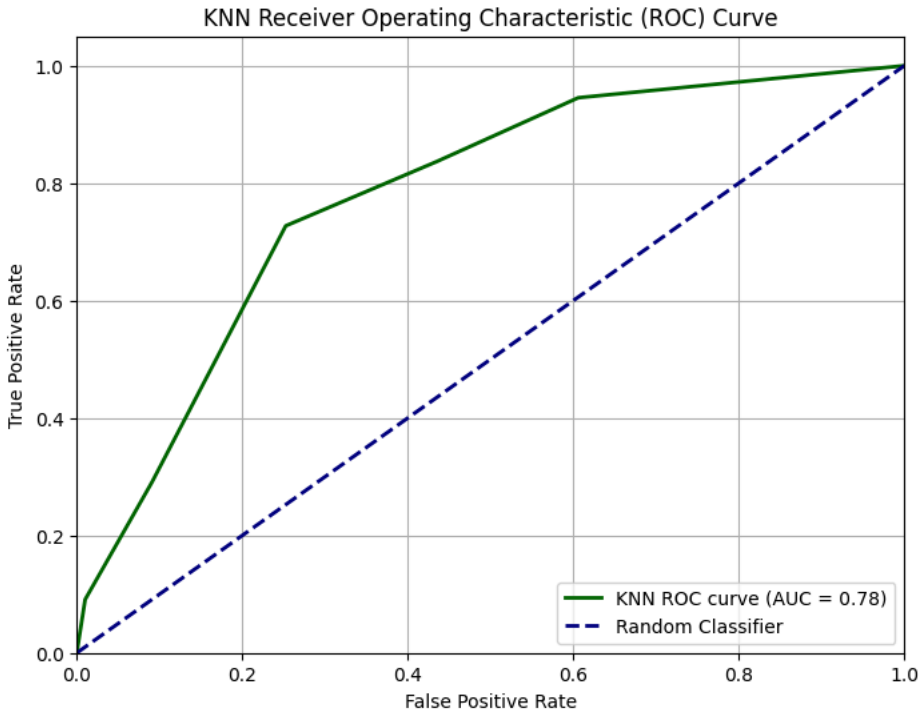
Precision, Recall, F1-score for KNN

```
from sklearn.metrics import classification_report

print("KNN Classification Report (reiterated for clarity):\n", classification_report(y_test, y_pred_knn))
```

```
KNN Classification Report (reiterated for clarity):
              precision    recall  f1-score   support

           0       0.83      0.75      0.79        99
           1       0.62      0.73      0.67        55

    accuracy                           0.74       154
   macro avg       0.72      0.74      0.73       154
weighted avg       0.75      0.74      0.74       154
```

ROC Curve for KNN

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Get the predicted probabilities for the positive class (class 1 - diabetes) for KNN
y_pred_proba_knn = model_knn.predict_proba(X_test)[:, 1]

# Calculate ROC curve values for KNN
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_pred_proba_knn)

# Calculate AUC score for KNN
auc_knn = roc_auc_score(y_test, y_pred_proba_knn)

# Plot the ROC curve for KNN
plt.figure(figsize=(8, 6))
plt.plot(fpr_knn, tpr_knn, color='darkgreen', lw=2, label=f'KNN ROC curve (AUC = {auc_knn:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('KNN Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



```
Start coding or generate with AI.
```

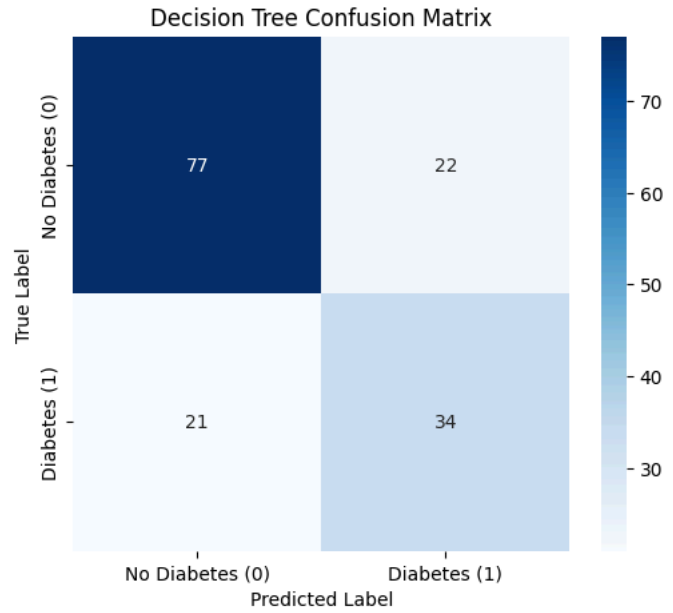
Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# 1. Initialize the Decision Tree model
# We'll use a `random_state` to make sure our results are consistent.
model_dt = DecisionTreeClassifier(random_state=42)

# 2. Train the model using our training data
print("Training the Decision Tree Classifier model...")
model_dt.fit(X_train, y_train)
print("Decision Tree Model training complete!")

# 3. Make predictions on the test data
y_pred_dt = model_dt.predict(X_test)

# 4. Evaluate the Decision Tree model's performance
accuracy_dt = accuracy_score(y_test, y_pred_dt)
report_dt = classification_report(y_test, y_pred_dt)

print(f"\nDecision Tree Model Accuracy: {accuracy_dt:.2f}")
print("\nDecision Tree Classification Report:\n", report_dt)
```

```
Training the Decision Tree Classifier model...
Decision Tree Model training complete!

Decision Tree Model Accuracy: 0.72

Decision Tree Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.78      0.78        99
           1       0.61      0.62      0.61        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154
```

Confusion Matrix for Decision Tree

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the confusion matrix for Decision Tree
cm_dt = confusion_matrix(y_test, y_pred_dt)

# Visualize the confusion matrix for Decision Tree
plt.figure(figsize=(6, 5))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Diabetes (0)', 'Diabetes (1)'],
            yticklabels=['No Diabetes (0)', 'Diabetes (1)'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Decision Tree Confusion Matrix')
plt.show()
```



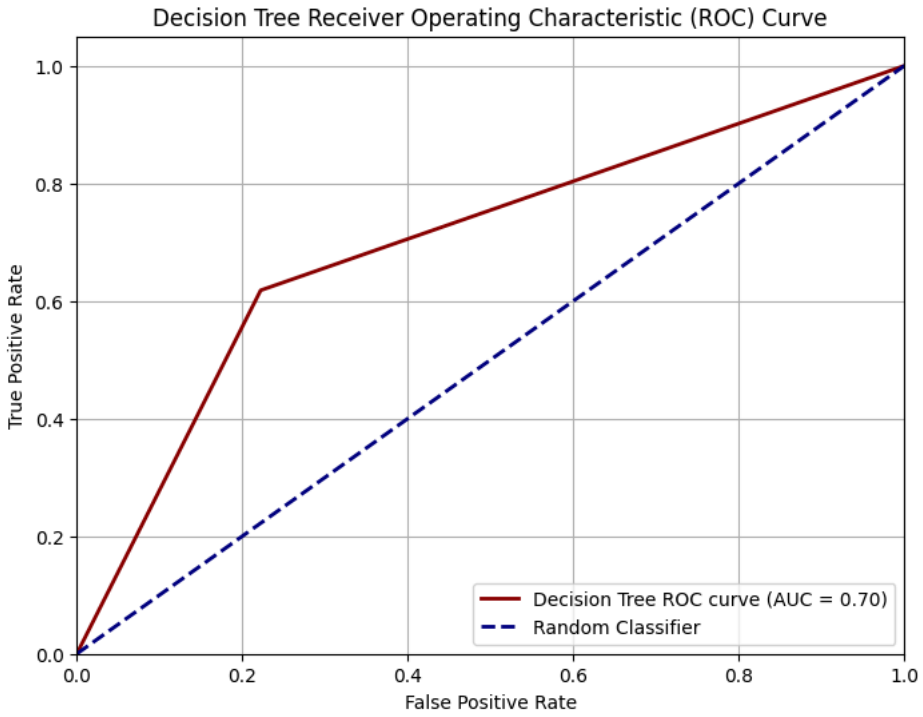
Precision, Recall, F1-score for Decision Tree

```
from sklearn.metrics import classification_report

print("Decision Tree Classification Report (reiterated for clarity):\n", classification_report(y_test, y_pred_dt))
```

```
Decision Tree Classification Report (reiterated for clarity):
              precision    recall  f1-score   support

           0       0.79      0.78      0.78        99
           1       0.61      0.62      0.61        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154
```

ROC Curve for Decision Tree

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Get the predicted probabilities for the positive class (class 1 - diabetes) for Decision Tree
y_pred_proba_dt = model_dt.predict_proba(X_test)[:, 1]

# Calculate ROC curve values for Decision Tree
fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_pred_proba_dt)

# Calculate AUC score for Decision Tree
auc_dt = roc_auc_score(y_test, y_pred_proba_dt)

# Plot the ROC curve for Decision Tree
plt.figure(figsize=(8, 6))
plt.plot(fpr_dt, tpr_dt, color='darkred', lw=2, label=f'Decision Tree ROC curve (AUC = {auc_dt:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Decision Tree Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```




```
Start coding or generate with AI.
```

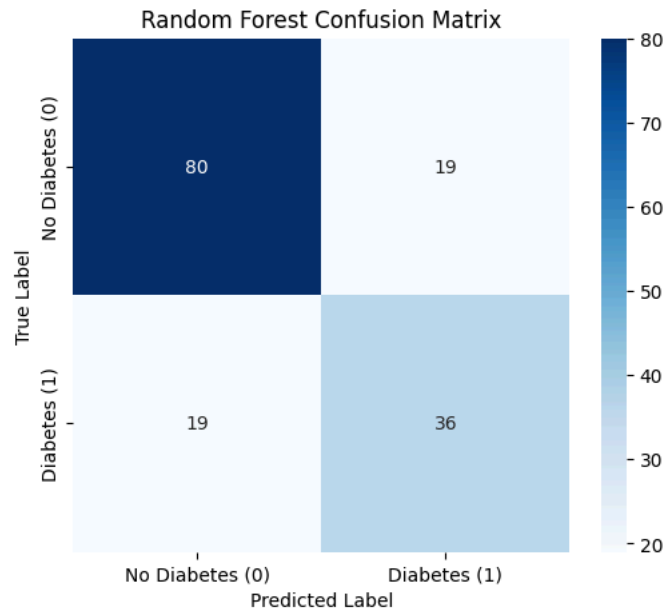
Model Training - Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# 1. Initialize the Random Forest model
# We'll use `n_estimators` (number of trees) and `random_state` for consistency.
model_rf = RandomForestClassifier(n_estimators=100, random_state=42) # A common starting point is 100 trees

# 2. Train the model using our training data
print("Training the Random Forest Classifier model...")
model_rf.fit(X_train, y_train)
print("Random Forest Model training complete!")

# 3. Make predictions on the test data
y_pred_rf = model_rf.predict(X_test)

# 4. Evaluate the Random Forest model's performance
accuracy_rf = accuracy_score(y_test, y_pred_rf)
report_rf = classification_report(y_test, y_pred_rf)

print(f"\nRandom Forest Model Accuracy: {accuracy_rf:.2f}")
print("\nRandom Forest Classification Report:\n", report_rf)
```

```
Training the Random Forest Classifier model...
Random Forest Model training complete!

Random Forest Model Accuracy: 0.75

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.81      0.81        99
           1       0.65      0.65      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
```

Confusion Matrix for Random Forest

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the confusion matrix for Random Forest
cm_rf = confusion_matrix(y_test, y_pred_rf)

# Visualize the confusion matrix for Random Forest
plt.figure(figsize=(6, 5))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Diabetes (0)', 'Diabetes (1)'],
            yticklabels=['No Diabetes (0)', 'Diabetes (1)'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Random Forest Confusion Matrix')
plt.show()
```



Precision, Recall, F1-score for Random Forest

```
from sklearn.metrics import classification_report

print("Random Forest Classification Report (reiterated for clarity):\n", classification_report(y_test, y_pred_rf))
```

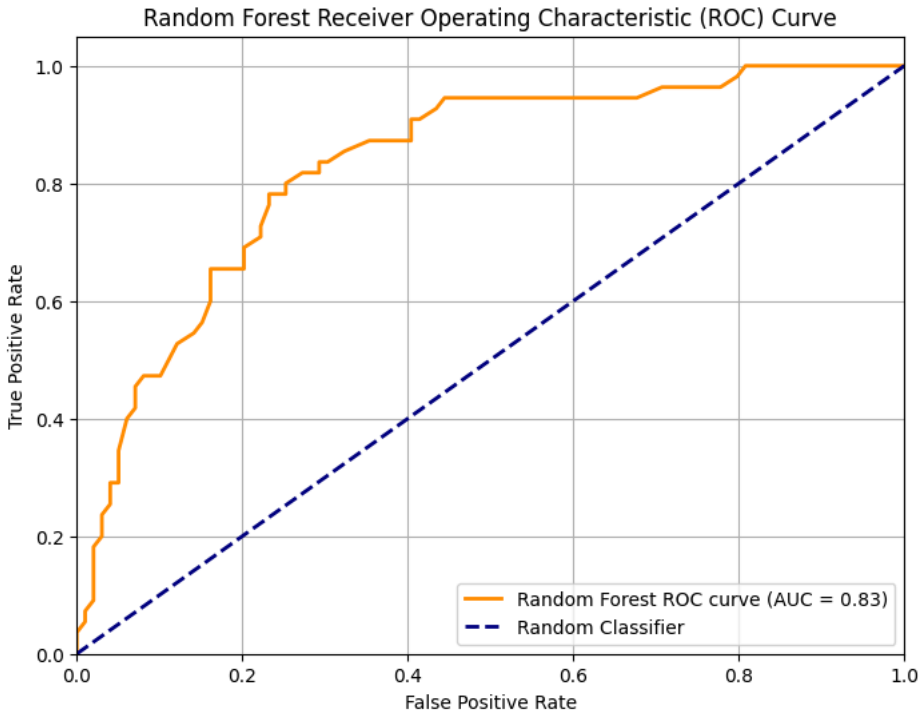
```
Random Forest Classification Report (reiterated for clarity):
               precision    recall  f1-score   support

           0       0.81      0.81      0.81        99
           1       0.65      0.65      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
```

ROC Curve for Random Forest

.

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Get the predicted probabilities for the positive class (class 1 - diabetes) for Random Forest
y_pred_proba_rf = model_rf.predict_proba(X_test)[:, 1]

# Calculate ROC curve values for Random Forest
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_pred_proba_rf)

# Calculate AUC score for Random Forest
```

```
auc_rf = roc_auc_score(y_test, y_pred_proba_rf)

# Plot the ROC curve for Random Forest
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2, label=f'Random Forest ROC curve (AUC = {auc_rf:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```




```
Start coding or generate with AI.
```

Naive Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# 1. Initialize the Naive Bayes model (GaussianNB is suitable for continuous data)
model_nb = GaussianNB()

# 2. Train the model using our training data
print("Training the Naive Bayes Classifier model...")
model_nb.fit(X_train, y_train)
print("Naive Bayes Model training complete!")

# 3. Make predictions on the test data
y_pred_nb = model_nb.predict(X_test)

# 4. Evaluate the Naive Bayes model's performance
accuracy_nb = accuracy_score(y_test, y_pred_nb)
report_nb = classification_report(y_test, y_pred_nb)

print(f"\nNaive Bayes Model Accuracy: {accuracy_nb:.2f}")
print("\nNaive Bayes Classification Report:\n", report_nb)
```

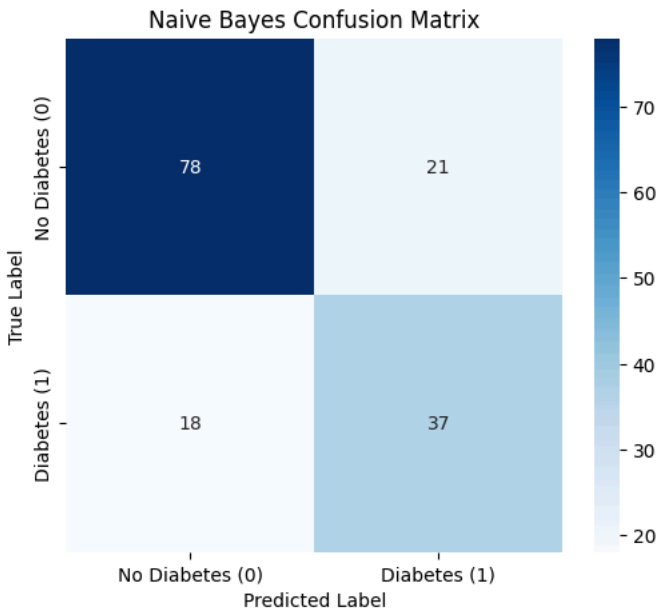
```
Training the Naive Bayes Classifier model...
Naive Bayes Model training complete!

Naive Bayes Model Accuracy: 0.75
```

```
Naive Bayes 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
```

1. Confusion Matrix for Naive Bayes

Let's see how our Naive Bayes model got its predictions right and wrong.

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the confusion matrix for Naive Bayes
cm_nb = confusion_matrix(y_test, y_pred_nb)

# Visualize the confusion matrix for Naive Bayes
plt.figure(figsize=(6, 5))
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Diabetes (0)', 'Diabetes (1)'],
            yticklabels=['No Diabetes (0)', 'Diabetes (1)'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Naive Bayes Confusion Matrix')
plt.show()
```



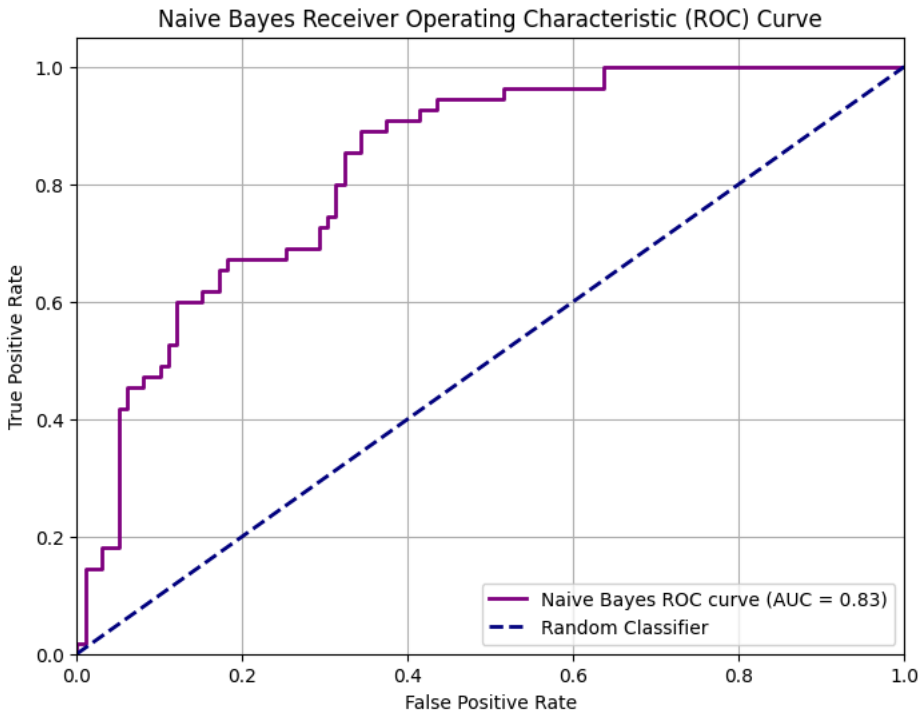
Precision, Recall, F1-score for Naive Bayes

```
from sklearn.metrics import classification_report

print("Naive Bayes Classification Report (reiterated for clarity):\n", classification_report(y_test, y_pred_nb))
```

```
Naive Bayes Classification Report (reiterated for clarity):
              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
```

ROC Curve for Naive Bayes

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Get the predicted probabilities for the positive class (class 1 - diabetes) for Naive Bayes
y_pred_proba_nb = model_nb.predict_proba(X_test)[:, 1]

# Calculate ROC curve values for Naive Bayes
fpr_nb, tpr_nb, thresholds_nb = roc_curve(y_test, y_pred_proba_nb)

# Calculate AUC score for Naive Bayes
auc_nb = roc_auc_score(y_test, y_pred_proba_nb)

# Plot the ROC curve for Naive Bayes
plt.figure(figsize=(8, 6))
plt.plot(fpr_nb, tpr_nb, color='purple', lw=2, label=f'Naive Bayes ROC curve (AUC = {auc_nb:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Naive Bayes Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

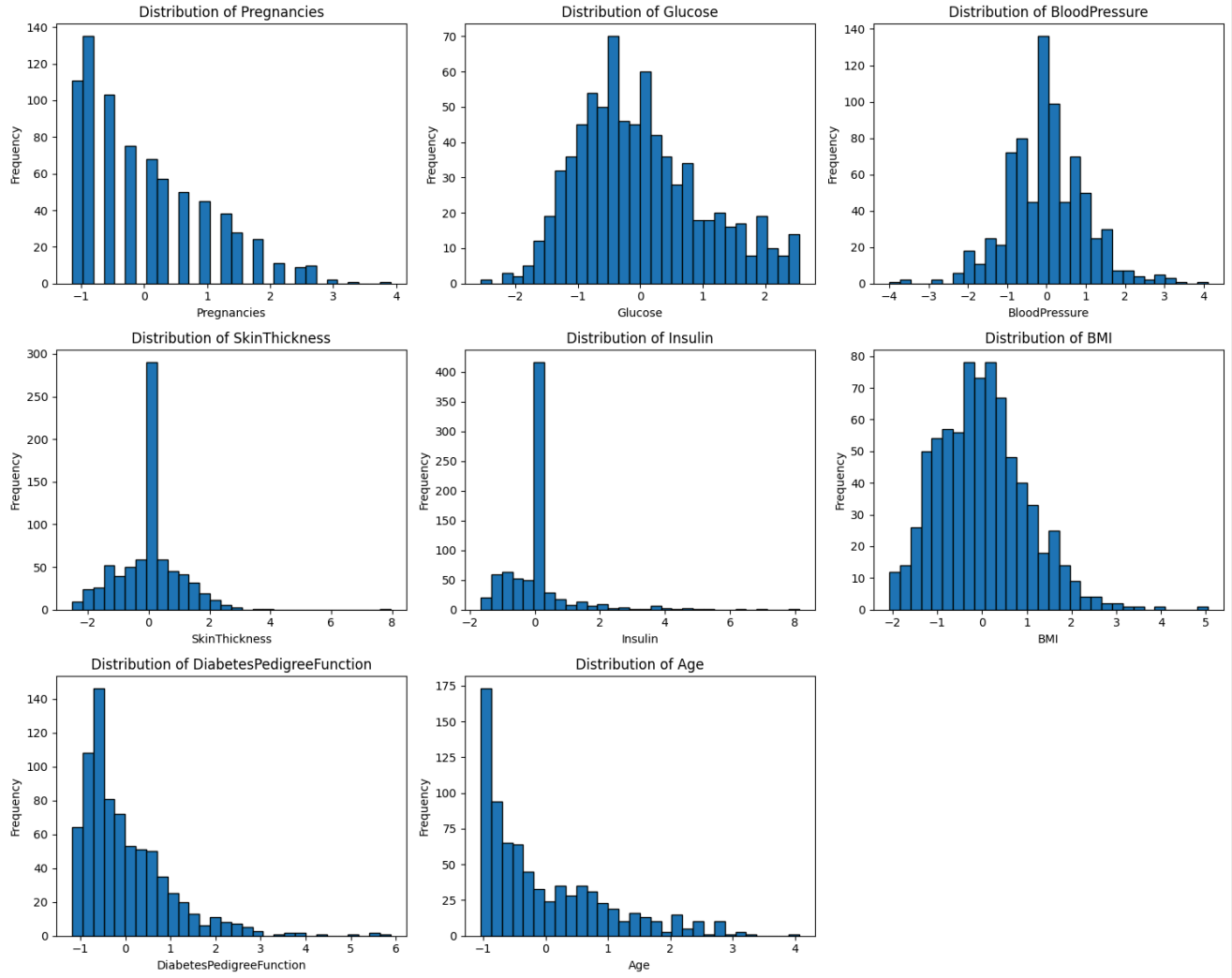


Start coding or generate with AI.

Histograms for Feature Distribution

```
import matplotlib.pyplot as plt

# Select all columns from df_standard_scaled that are not 'Outcome'
features = df_standard_scaled.drop('Outcome', axis=1).columns

# Determine the number of rows and columns for the subplot grid
n_features = len(features)
n_cols = 3  # You can adjust this number
n_rows = (n_features + n_cols - 1) // n_cols

plt.figure(figsize=(n_cols * 5, n_rows * 4))
```

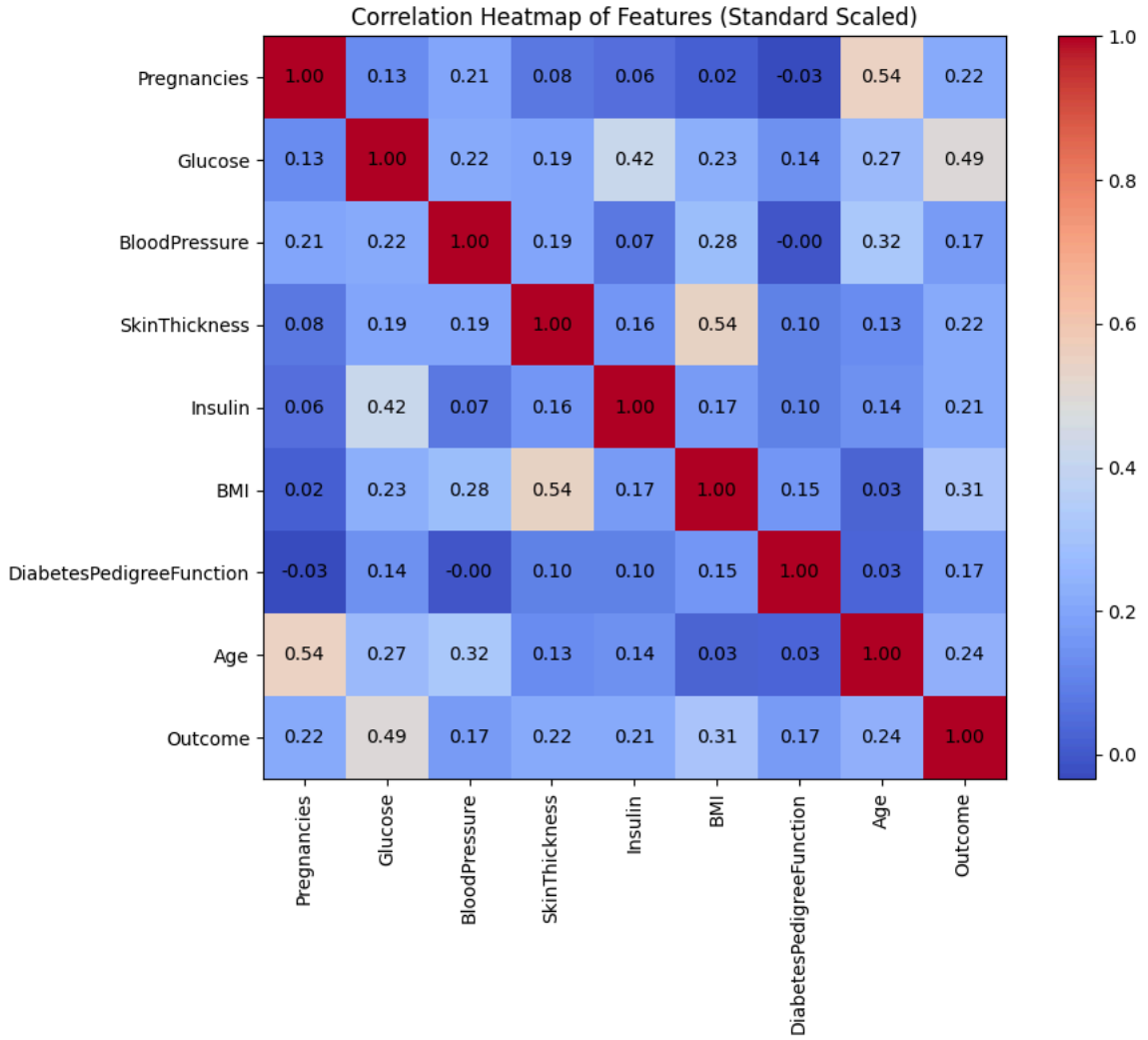
```
for i, col in enumerate(features):
    plt.subplot(n_rows, n_cols, i + 1)
    plt.hist(df_standard_scaled[col], bins=30, edgecolor='black') # Using plt.hist
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

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



Correlation Heatmap

```
import matplotlib.pyplot as plt
import numpy as np # Needed for np.ndenumerate

# Calculate the correlation matrix
correlation_matrix = df_standard_scaled.corr()

# Plot the correlation heatmap using imshow
plt.figure(figsize=(10, 8))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar() # Add a color bar
plt.xticks(range(len(correlation_matrix.columns)), correlation_matrix.columns, rotation=90)
plt.yticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)
plt.title('Correlation Heatmap of Features (Standard Scaled)')

# Annotate values on the heatmap
for (i, j), val in np.ndenumerate(correlation_matrix):
```

```
        plt.text(j, i, f"{val:.2f}", ha="center", va="center", color="black")
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

```
!sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic
```