# Pattern Recognition Laboratory

## Mini Project
## Report


**Submitted By**

**Aswin Tom**
(97424607006)

**Hrishikesh M Jayakrishnan**
(97424607010)

**Mohammad Farhan Z**
(97424607012)

**MSc Computer Science with Specialization in Artificial Intelligence**

# Mini Project

## Aim:

To train a neural network classifier using the PIMA Indian Diabetes dataset by performing appropriate data preprocessing, model training, and performance evaluation, and to compare its results with at least two traditional machine learning models using metrics such as accuracy and confusion matrix.

## Program Code:

[1]:
```
import pandas as pd
import numpy as np

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report, roc_curve, auc, f1_score, precision_score,␣
↪recall_score

import copy
import torch
import torch.nn as nn
from torch.optim import NAdam
import torch.nn.functional as F

import seaborn as sns
import matplotlib.pyplot as plt

from imblearn.over_sampling import RandomOverSampler

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
Using device: cuda
```

[2]:
```
diabetes=pd.read_csv('diabetes.csv')
```

[3]:
```
diabetes.sample(5)
```

[3]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
135            2      125             60             20      140  33.8
677            0       93             60              0        0  35.3
```

```
568            4      154             72             29       126   31.3
356            1      125             50             40       167   33.3
360            5      189             64             33       325   31.2

     DiabetesPedigreeFunction  Age  Outcome
135                     0.088   31        0
677                     0.263   25        0
568                     0.338   37        0
356                     0.962   28        1
360                     0.583   29        1
```

[4]:
```python
diabetes.nunique()
```

[4]:
```
Pregnancies                  17
Glucose                     136
BloodPressure                47
SkinThickness                51
Insulin                     186
BMI                         248
DiabetesPedigreeFunction    517
Age                          52
Outcome                       2
dtype: int64
```

[5]:
```python
diabetes.isnull().sum()
```

[5]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[6]:
```python
diabetes_x = diabetes.drop("Outcome", axis=1)
diabetes_y = diabetes["Outcome"]
```

[7]:
```python
diabetes_x.describe()
```

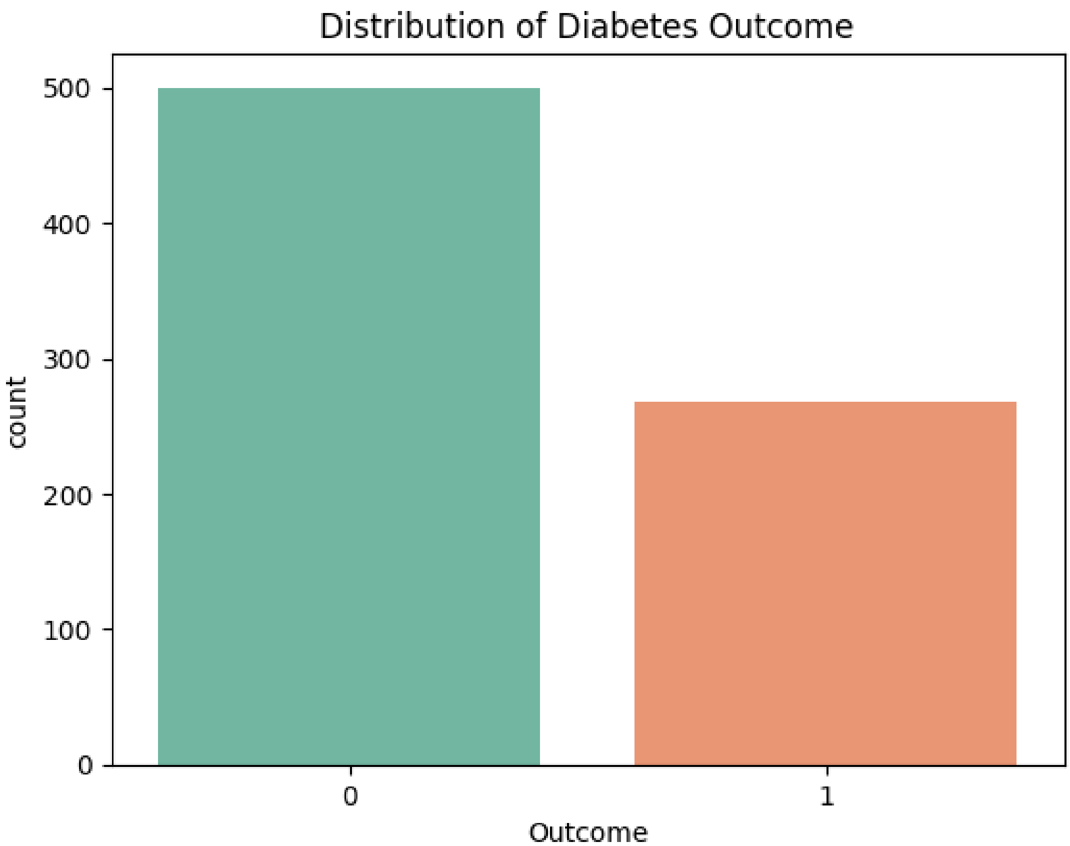
[7]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
```

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

[8]:

```python
sns.countplot(x='Outcome', data=diabetes, palette='Set2')
plt.title("Distribution of Diabetes Outcome")
plt.show()
```

```
/tmp/ipykernel_4973/2807747172.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed␣
↪in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the␣
↪same
effect.

  sns.countplot(x='Outcome', data=diabetes, palette='Set2')
```

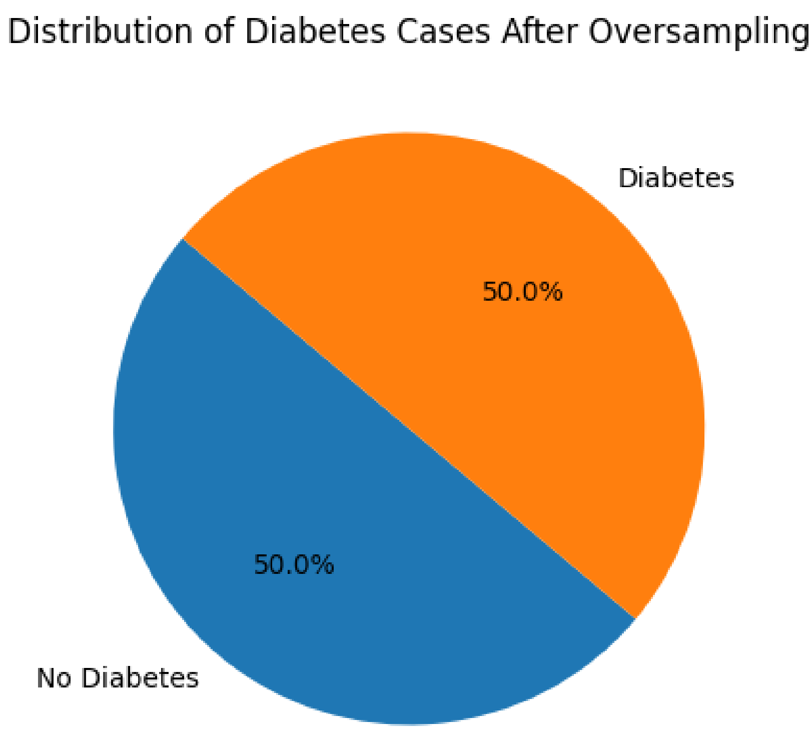



[9]:
```
diabetes_x_copy=diabetes_x.copy()
diabetes_y_copy=diabetes_y.copy()
```

[10]:
```
rand_sampler = RandomOverSampler(sampling_strategy=1.0, random_state=42)
```

[11]:
```
diabetes_x_resampled, diabetes_y_resampled = rand_sampler.
 ↪fit_resample(diabetes_x_copy, diabetes_y_copy)
```

[12]:
```
plt.figure(figsize=(9, 5))
plt.pie(diabetes_y_resampled.value_counts(), labels=["No Diabetes",␣
 ↪"Diabetes"], autopct="%1.1f%%", startangle=140)
plt.title("Distribution of Diabetes Cases After Oversampling")
plt.show()
```




[13]:
```
print("Balanced dataset:")
print(f"Positive cases:{diabetes_x_resampled[diabetes_y_resampled == 1].
↪shape[0]}\n Negative cases:{diabetes_x_resampled[diabetes_y_resampled␣
↪== 0].shape[0]}")
```

```
Balanced dataset:
Positive cases:500
 Negative cases:500
```

[14]:
```
diabetes_x_resampled
```

[14]:
```
     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
..           ...      ...            ...            ...      ...   ...
995            1      122             64             32      156  35.1
996            0      131              0              0        0  43.2
```

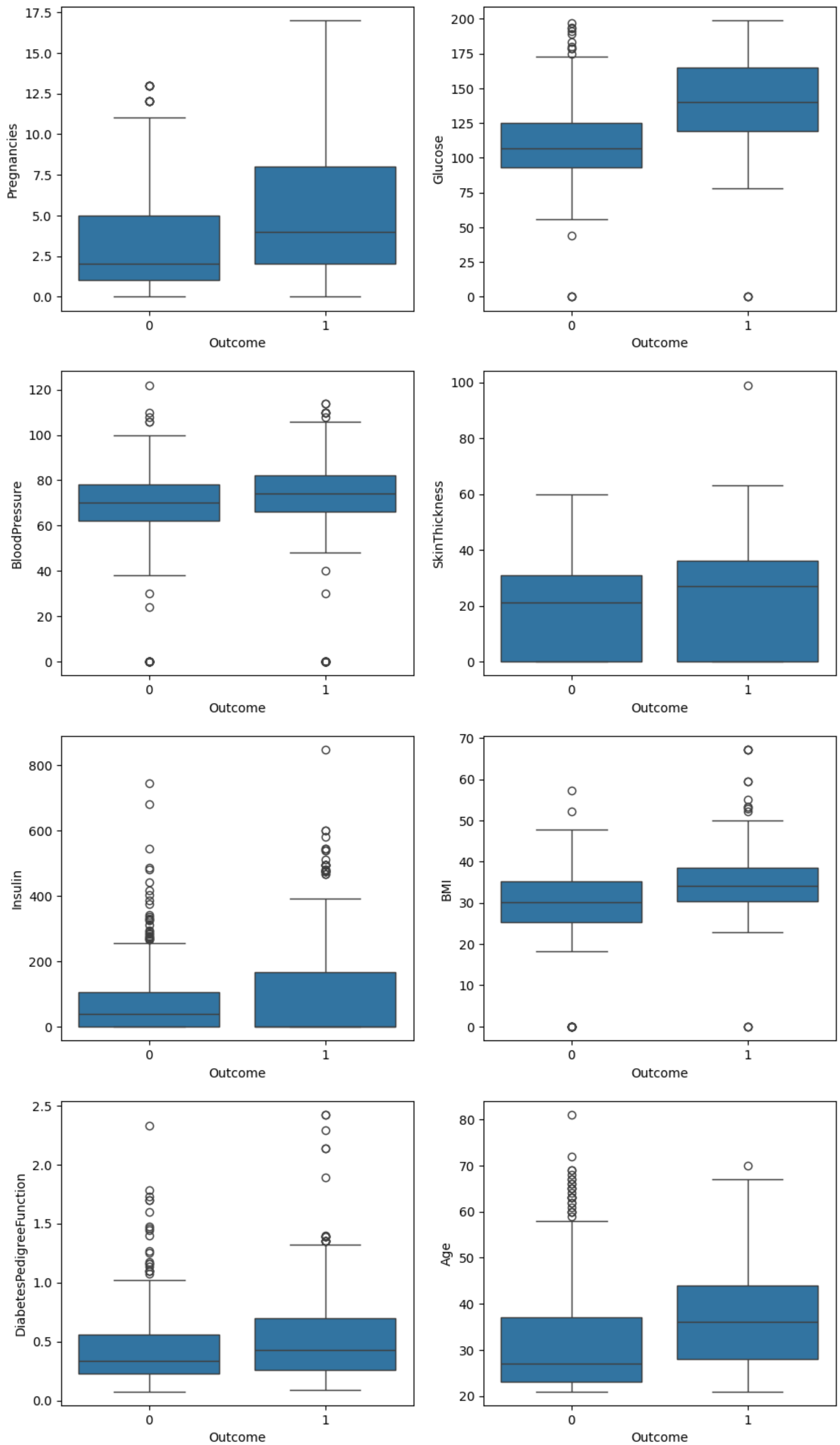
```
997             8     120             0             0        0  30.0
998             4     111            72            47      207  37.1
999            13     158           114             0        0  42.3

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
995                     0.692   30
996                     0.270   26
997                     0.183   38
998                     1.390   56
999                     0.257   44

[1000 rows x 8 columns]
```

[15]:

```
diabetes_combined=pd.
 ↪concat([diabetes_x_resampled,diabetes_y_resampled],axis=1)
```

[32]:

```
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(11, 20))
for i, col in enumerate(diabetes_combined.columns[:-1]):
    ax = axes[i // 2, i % 2]
    sns.boxplot(data=diabetes_combined, x="Outcome", y=col, ax=ax)
```


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
0
1

[20]:
```python
def remove_outliers(df, columns):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df[(df[col] < lower_bound)] = pd.NA
        df[(df[col] > upper_bound)] = pd.NA
    return df
```

[21]:
```python
diabetes_x_resampled = remove_outliers(diabetes_x_resampled,␣
 ↪diabetes_x_resampled.columns)
```

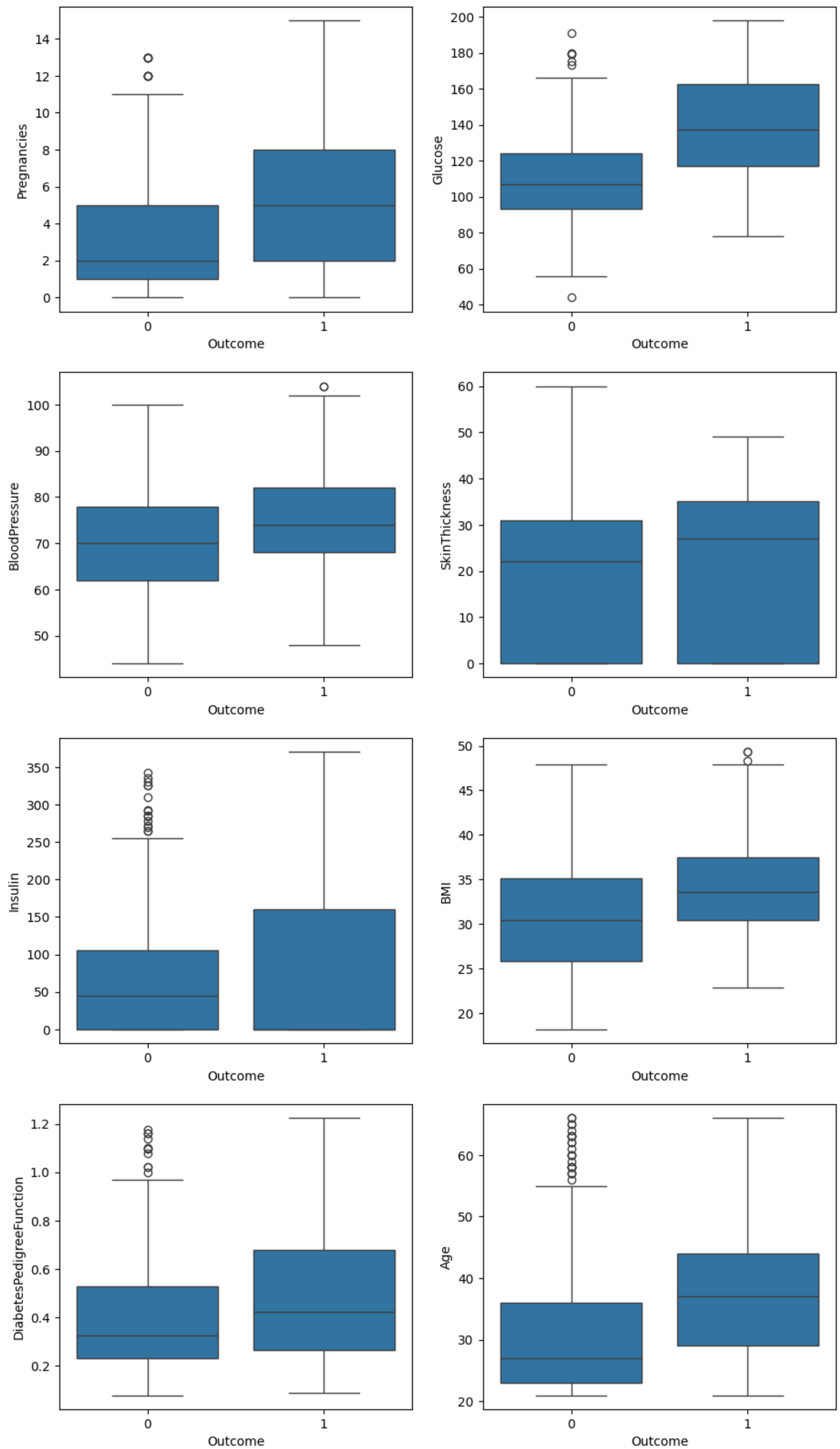
[22]:
```python
diabetes_x_copy = remove_outliers(diabetes_x_copy, diabetes_x_copy.columns)
```

[23]:
```python
combined_df = pd.concat([diabetes_x_resampled, diabetes_y_resampled],␣
 ↪axis=1)
combined_df_cp = pd.concat([diabetes_x_copy, diabetes_y_copy], axis=1)
```

[24]:
```python
combined_df.dropna(inplace=True)
combined_df_cp.dropna(inplace=True)
```

[25]:
```python
diabetes_x = combined_df.drop("Outcome", axis=1)
diabetes_y = combined_df["Outcome"]
diabetes_x_copy = combined_df_cp.drop("Outcome", axis=1)
diabetes_y_copy = combined_df_cp["Outcome"]
```

[33]:
```python
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(11, 20))
for i, col in enumerate(diabetes_x.columns):
    ax = axes[i // 2, i % 2]
    sns.boxplot(data=diabetes_x, x=diabetes_y, y=col, ax=ax)
```


Pregnancies
Outcome
Glucose
Outcome
BloodPressure
Outcome
SkinThickness
Outcome
Insulin
Outcome
BMI
Outcome
DiabetesPedigreeFunction
Outcome
Age
Outcome

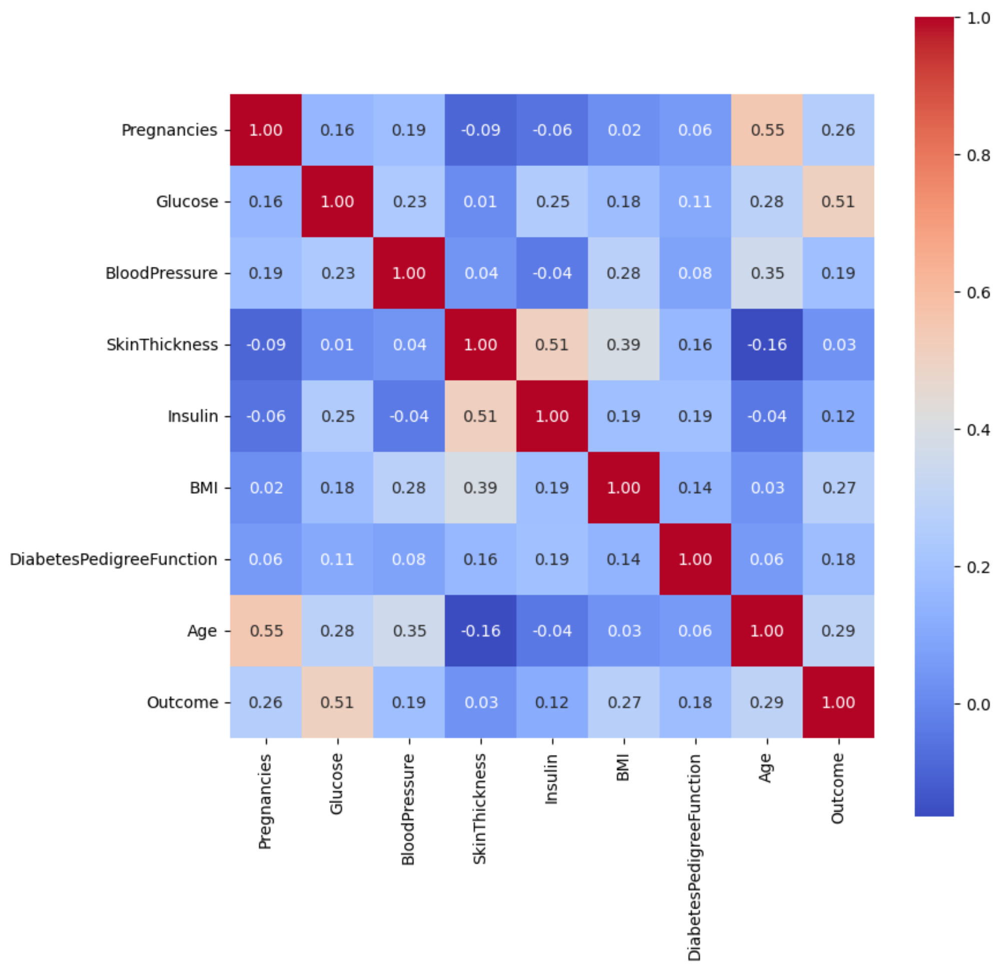
```
[34]: print(f"After removing outliers, there are {diabetes_x.shape[0]} samples␣
       ↪left.")
      print(f"There are {diabetes_x[diabetes_y == 1].shape[0]} positive cases␣
       ↪and {diabetes_x[diabetes_y== 0].shape[0]} negative cases")
```

```
After removing outliers, there are 835 samples left.
There are 394 positive cases and 441 negative cases
```

Correlation matrix of balanced dataset

```
[43]: corr = combined_df.corr()
      plt.figure(figsize=(9, 9))
      sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", square=True)
```

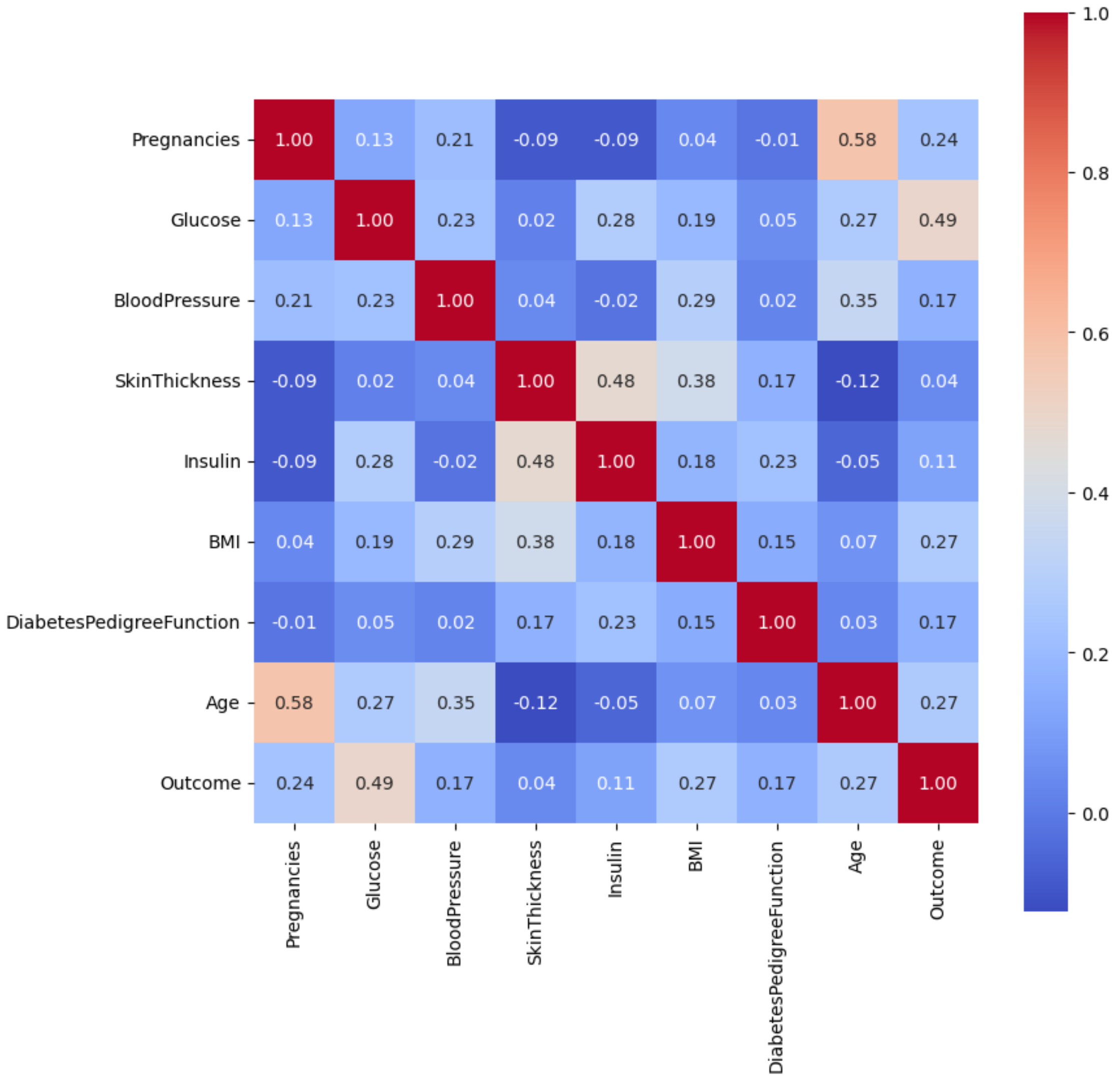
```
[43]: <Axes: >
```



Correlation matrix of unbalanced dataset

[45]:
```
corr_cp = combined_df_cp.corr()
plt.figure(figsize=(9, 9))
sns.heatmap(corr_cp, annot=True, fmt=".2f", cmap="coolwarm", square=True)
```

[45]: <Axes: >




[46]:
```
scaler = StandardScaler()
diabetes_X_scaled = scaler.fit_transform(diabetes_x)
diabetes_X_scaled_cp = scaler.fit_transform(diabetes_x_copy)
```

[47]:
```
X_train, X_test, y_train, y_test = train_test_split(diabetes_X_scaled,␣
↪diabetes_y, test_size=0.2, random_state=42)
```

[48]:
```
X_train_02, X_test_02, y_train_02, y_test_02 =␣
↪train_test_split(diabetes_X_scaled_cp, diabetes_y_copy, test_size=0.2,␣
↪random_state=42)
```

Training a Random Forest Classifier on the balanced dataset

[50]:
```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

[50]:
```
RandomForestClassifier(random_state=42)
```

[51]:
```
y_pred = rf_model.predict(X_test)
```

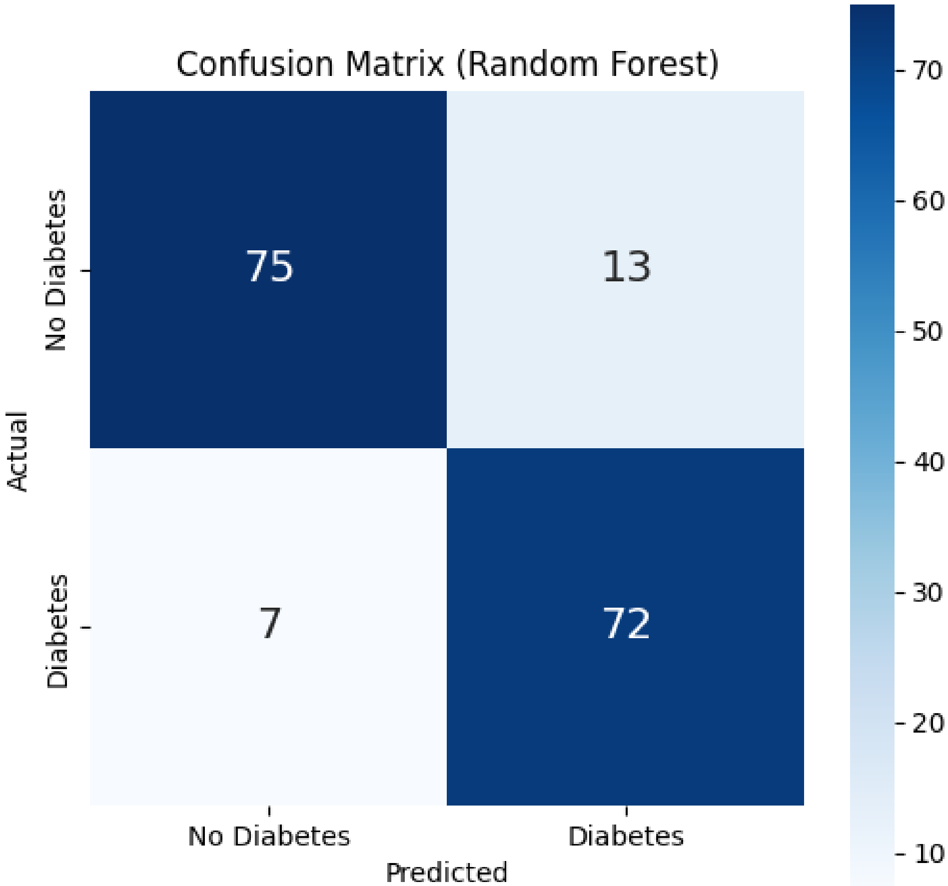
[52]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8802395209580839
```

[53]:
```
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", confusion_mat)
```

```
Confusion Matrix:
 [[75 13]
 [ 7 72]]
```

[56]:
```
plt.figure(figsize=(6, 6))
sns.heatmap(confusion_mat, annot=True, annot_kws={"size": 16}, fmt='d',␣
↪cmap='Blues', xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No␣
↪Diabetes", "Diabetes"], square=True)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix (Random Forest)")
plt.show()
```




[57]:
```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.85      0.88        88
           1       0.85      0.91      0.88        79

    accuracy                           0.88       167
   macro avg       0.88      0.88      0.88       167
weighted avg       0.88      0.88      0.88       167
```

Training a Random Forest Classifier on the unbalanced dataset

[59]:
```
rf_model_02 = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model_02.fit(X_train_02, y_train_02)
```

[59]: RandomForestClassifier(random_state=42)

[60]:
```
y_pred_02 = rf_model_02.predict(X_test_02)
```

[61]:
```
accuracy_02 = accuracy_score(y_test_02, y_pred_02)
print("Accuracy on unbalanced dataset:", accuracy_02)
```

```
Accuracy on unbalanced dataset: 0.7421875
```

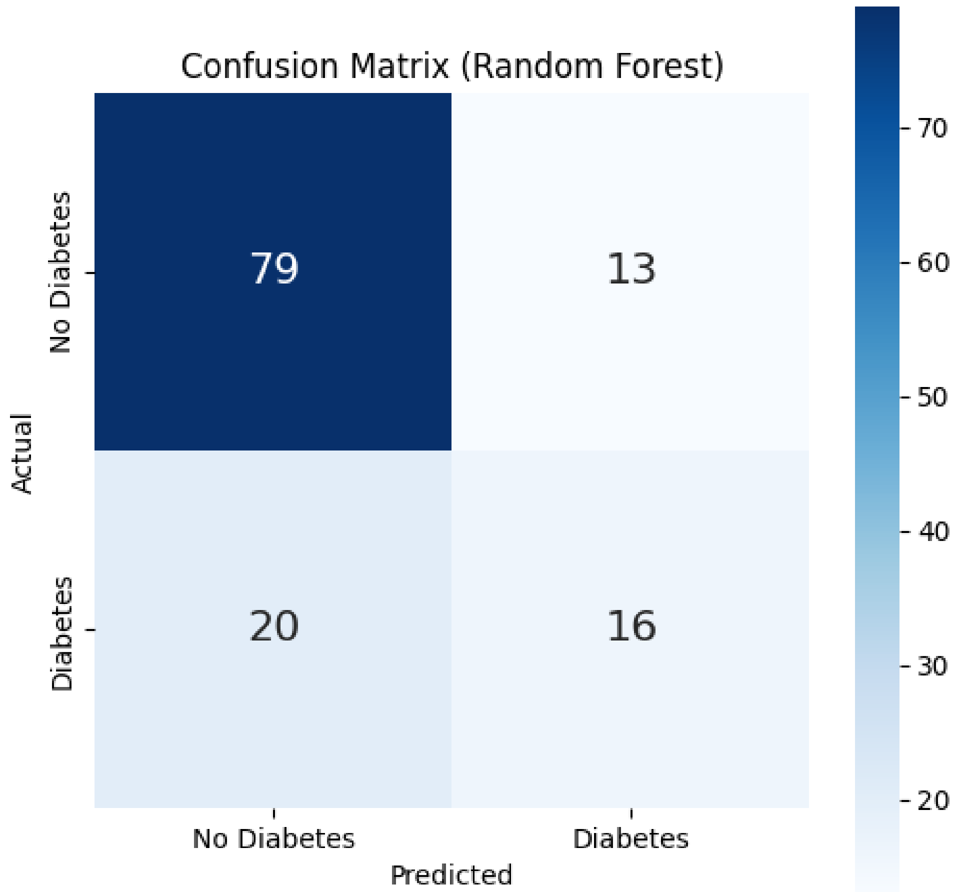
[62]:
```
y_test_02
```

[62]:
```
338    1
763    0
104    0
437    0
184    0
      ..
723    0
246    0
346    0
275    0
358    0
Name: Outcome, Length: 128, dtype: int64
```

[63]:
```
confusion_matrix_02 = confusion_matrix(y_test_02, y_pred_02)
print("Confusion Matrix (Unbalanced Dataset):")
print(confusion_matrix_02)
```

```
Confusion Matrix (Unbalanced Dataset):
[[79 13]
 [20 16]]
```

[64]:
```
plt.figure(figsize=(6,6))
sns.heatmap(confusion_matrix_02, annot=True, annot_kws={"size": 16},␣
 ↪fmt='d', cmap='Blues', xticklabels=["No Diabetes", "Diabetes"],␣
 ↪yticklabels=["No Diabetes", "Diabetes"], square=True)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix (Random Forest)")
plt.show()
```




[65]:
```
print("Classification Report:\n", classification_report(y_test_02,
    y_pred_02))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.86      0.83        92
           1       0.55      0.44      0.49        36

    accuracy                           0.74       128
   macro avg       0.67      0.65      0.66       128
weighted avg       0.73      0.74      0.73       128
```

Training a Multi-Layer Perceptron on the balanced dataset

[142]:
```
class DiabetesModel(nn.Module):
    def __init__(self, input_size=8, hidden_size=14, output_size=1):
```

```
        super(DiabetesModel, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = F.leaky_relu(self.fc1(x))
        x = F.leaky_relu(self.fc2(x))
        x = self.out(x)
        return x
```

[143]:
```
torch.manual_seed(42)
nn_model_01 = DiabetesModel()
nn_model_01
nn_model_01.to(device=device)
```

[143]:
```
DiabetesModel(
  (fc1): Linear(in_features=8, out_features=14, bias=True)
  (fc2): Linear(in_features=14, out_features=14, bias=True)
  (out): Linear(in_features=14, out_features=1, bias=True)
)
```

[144]:
```
torch.manual_seed(42)
nn_model_02 = DiabetesModel()
nn_model_02
nn_model_02.to(device=device)
```

[144]:
```
DiabetesModel(
  (fc1): Linear(in_features=8, out_features=14, bias=True)
  (fc2): Linear(in_features=14, out_features=14, bias=True)
  (out): Linear(in_features=14, out_features=1, bias=True)
)
```

We are using cross entropy loss for calculating the loss of the model, and NADAM for optimizing the weights and biases of the model

[145]:
```
loss_fn = nn.BCEWithLogitsLoss()
optimizer = NAdam(nn_model_01.parameters(), lr=0.01)
```

[146]:
```
loss_fn2 = nn.BCEWithLogitsLoss()
optimizer2 = NAdam(nn_model_02.parameters(), lr=0.01)
```

[147]:
```
X_train_tensor = torch.tensor(X_train, dtype=torch.float32, device=device)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32,␣
 ↪device=device).view(-1, 1)
```

```
X_test_tensor = torch.tensor(X_test, dtype=torch.float32, device=device)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32,␣
↪device=device).view(-1, 1)
```

[148]:
```
X_train_tensor1 = torch.tensor(X_train_02, dtype=torch.float32,␣
↪device=device)
y_train_tensor1 = torch.tensor(y_train_02.values, dtype=torch.float32,␣
↪device=device).view(-1, 1)
X_test_tensor1 = torch.tensor(X_test_02, dtype=torch.float32,␣
↪device=device)
y_test_tensor1 = torch.tensor(y_test_02.values, dtype=torch.float32,␣
↪device=device).view(-1, 1)
```

Training the neural network on the balanced dataset

[149]:
```
epochs = 1000
best_model_loss = float('inf')
best_model_weights = None
patience = 7

loss_list = []
accuracy_list = []

for i in range(epochs):
    # train_loss = 0.0
    nn_model_01.train()

    outputs = nn_model_01(X_train_tensor)
    loss = loss_fn(outputs, y_train_tensor)
    loss_list.append(loss.item())

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    nn_model_01.eval()
    with torch.no_grad():
        outputs = nn_model_01(X_test_tensor)
        test_loss = loss_fn(outputs, y_test_tensor)
    y_pred = torch.sigmoid(outputs).round()
    accuracy = (y_pred == y_test_tensor).float().mean().item()
    accuracy_list.append(accuracy)

    if test_loss < best_model_loss:
        best_model_loss = test_loss
```

```
        best_model_weights = copy.deepcopy(nn_model_01.state_dict())
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= patience:
            print(f"Early stopping at epoch {i+1}")
            break
    print(f"Epoch {i+1}/{epochs}, Train Loss: {loss.item():.4f}, Val Loss:␣
↪{test_loss.item():.4f}")
```

```
Epoch 1/1000, Train Loss: 0.6923, Val Loss: 0.6869
Epoch 2/1000, Train Loss: 0.6869, Val Loss: 0.6823
Epoch 3/1000, Train Loss: 0.6820, Val Loss: 0.6764
Epoch 4/1000, Train Loss: 0.6764, Val Loss: 0.6688
Epoch 5/1000, Train Loss: 0.6694, Val Loss: 0.6588
Epoch 6/1000, Train Loss: 0.6604, Val Loss: 0.6463
Epoch 7/1000, Train Loss: 0.6490, Val Loss: 0.6315
Epoch 8/1000, Train Loss: 0.6355, Val Loss: 0.6143
Epoch 9/1000, Train Loss: 0.6199, Val Loss: 0.5949
Epoch 10/1000, Train Loss: 0.6030, Val Loss: 0.5750
Epoch 11/1000, Train Loss: 0.5853, Val Loss: 0.5558
Epoch 12/1000, Train Loss: 0.5682, Val Loss: 0.5389
Epoch 13/1000, Train Loss: 0.5522, Val Loss: 0.5242
Epoch 14/1000, Train Loss: 0.5377, Val Loss: 0.5100
Epoch 15/1000, Train Loss: 0.5244, Val Loss: 0.4983
Epoch 16/1000, Train Loss: 0.5125, Val Loss: 0.4879
Epoch 17/1000, Train Loss: 0.5018, Val Loss: 0.4791
Epoch 18/1000, Train Loss: 0.4919, Val Loss: 0.4711
Epoch 19/1000, Train Loss: 0.4831, Val Loss: 0.4650
Epoch 20/1000, Train Loss: 0.4754, Val Loss: 0.4592
Epoch 21/1000, Train Loss: 0.4683, Val Loss: 0.4551
Epoch 22/1000, Train Loss: 0.4621, Val Loss: 0.4507
Epoch 23/1000, Train Loss: 0.4567, Val Loss: 0.4507
Epoch 24/1000, Train Loss: 0.4522, Val Loss: 0.4470
Epoch 25/1000, Train Loss: 0.4484, Val Loss: 0.4493
Epoch 26/1000, Train Loss: 0.4449, Val Loss: 0.4447
Epoch 27/1000, Train Loss: 0.4420, Val Loss: 0.4546
Epoch 28/1000, Train Loss: 0.4407, Val Loss: 0.4447
Epoch 29/1000, Train Loss: 0.4410, Val Loss: 0.4588
Epoch 30/1000, Train Loss: 0.4378, Val Loss: 0.4443
Epoch 31/1000, Train Loss: 0.4347, Val Loss: 0.4543
Epoch 32/1000, Train Loss: 0.4310, Val Loss: 0.4452
Epoch 33/1000, Train Loss: 0.4284, Val Loss: 0.4515
Epoch 34/1000, Train Loss: 0.4260, Val Loss: 0.4448
Epoch 35/1000, Train Loss: 0.4240, Val Loss: 0.4508
```

```
Epoch 36/1000, Train Loss: 0.4219, Val Loss: 0.4430
Epoch 37/1000, Train Loss: 0.4201, Val Loss: 0.4496
Epoch 38/1000, Train Loss: 0.4185, Val Loss: 0.4418
Epoch 39/1000, Train Loss: 0.4170, Val Loss: 0.4500
Epoch 40/1000, Train Loss: 0.4154, Val Loss: 0.4406
Epoch 41/1000, Train Loss: 0.4141, Val Loss: 0.4509
Epoch 42/1000, Train Loss: 0.4127, Val Loss: 0.4395
Epoch 43/1000, Train Loss: 0.4117, Val Loss: 0.4519
Epoch 44/1000, Train Loss: 0.4096, Val Loss: 0.4389
Epoch 45/1000, Train Loss: 0.4085, Val Loss: 0.4501
Epoch 46/1000, Train Loss: 0.4065, Val Loss: 0.4391
Epoch 47/1000, Train Loss: 0.4052, Val Loss: 0.4487
Epoch 48/1000, Train Loss: 0.4036, Val Loss: 0.4384
Epoch 49/1000, Train Loss: 0.4028, Val Loss: 0.4481
Epoch 50/1000, Train Loss: 0.4012, Val Loss: 0.4380
Epoch 51/1000, Train Loss: 0.4000, Val Loss: 0.4475
Epoch 52/1000, Train Loss: 0.3981, Val Loss: 0.4370
Epoch 53/1000, Train Loss: 0.3964, Val Loss: 0.4458
Epoch 54/1000, Train Loss: 0.3949, Val Loss: 0.4355
Epoch 55/1000, Train Loss: 0.3940, Val Loss: 0.4448
Epoch 56/1000, Train Loss: 0.3924, Val Loss: 0.4342
Epoch 57/1000, Train Loss: 0.3914, Val Loss: 0.4448
Epoch 58/1000, Train Loss: 0.3905, Val Loss: 0.4322
Epoch 59/1000, Train Loss: 0.3901, Val Loss: 0.4444
Epoch 60/1000, Train Loss: 0.3891, Val Loss: 0.4324
Epoch 61/1000, Train Loss: 0.3894, Val Loss: 0.4460
Epoch 62/1000, Train Loss: 0.3875, Val Loss: 0.4316
Epoch 63/1000, Train Loss: 0.3868, Val Loss: 0.4446
Epoch 64/1000, Train Loss: 0.3850, Val Loss: 0.4306
Epoch 65/1000, Train Loss: 0.3848, Val Loss: 0.4442
Epoch 66/1000, Train Loss: 0.3825, Val Loss: 0.4299
Epoch 67/1000, Train Loss: 0.3805, Val Loss: 0.4418
Epoch 68/1000, Train Loss: 0.3794, Val Loss: 0.4301
Epoch 69/1000, Train Loss: 0.3789, Val Loss: 0.4418
Epoch 70/1000, Train Loss: 0.3769, Val Loss: 0.4303
Epoch 71/1000, Train Loss: 0.3762, Val Loss: 0.4434
Epoch 72/1000, Train Loss: 0.3755, Val Loss: 0.4307
Early stopping at epoch 73
```

[150]:
```
nn_model_01.to(device="cpu")
nn_model_01.load_state_dict(best_model_weights)
nn_model_01.eval()

X_test_tensor = X_test_tensor.to(device="cpu")
```

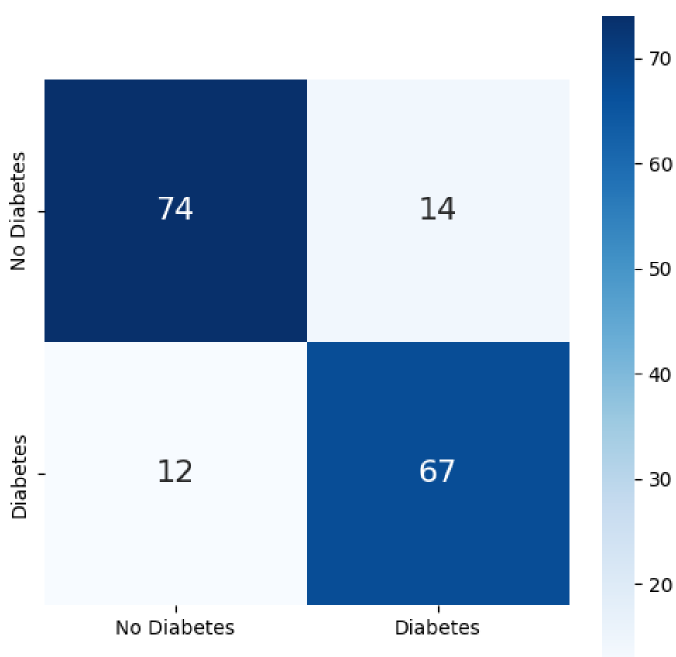
```
with torch.no_grad():
    outputs = nn_model_01(X_test_tensor)
    y_pred = torch.sigmoid(outputs).round().numpy()
```

[151]:
```
y_test_tensor = y_test_tensor.to(device="cpu")

print("Classification Report:")
print(classification_report(y_test_tensor, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.86      0.84      0.85        88
         1.0       0.83      0.85      0.84        79

    accuracy                           0.84       167
   macro avg       0.84      0.84      0.84       167
weighted avg       0.84      0.84      0.84       167
```

[152]:
```
conf_matrix = confusion_matrix(y_test_tensor, y_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(conf_matrix, annot=True, annot_kws={"size": 16}, fmt='d',␣
↪cmap='Blues', xticklabels=['No Diabetes', 'Diabetes'], yticklabels=['No␣
↪Diabetes', 'Diabetes'], square=True)
```

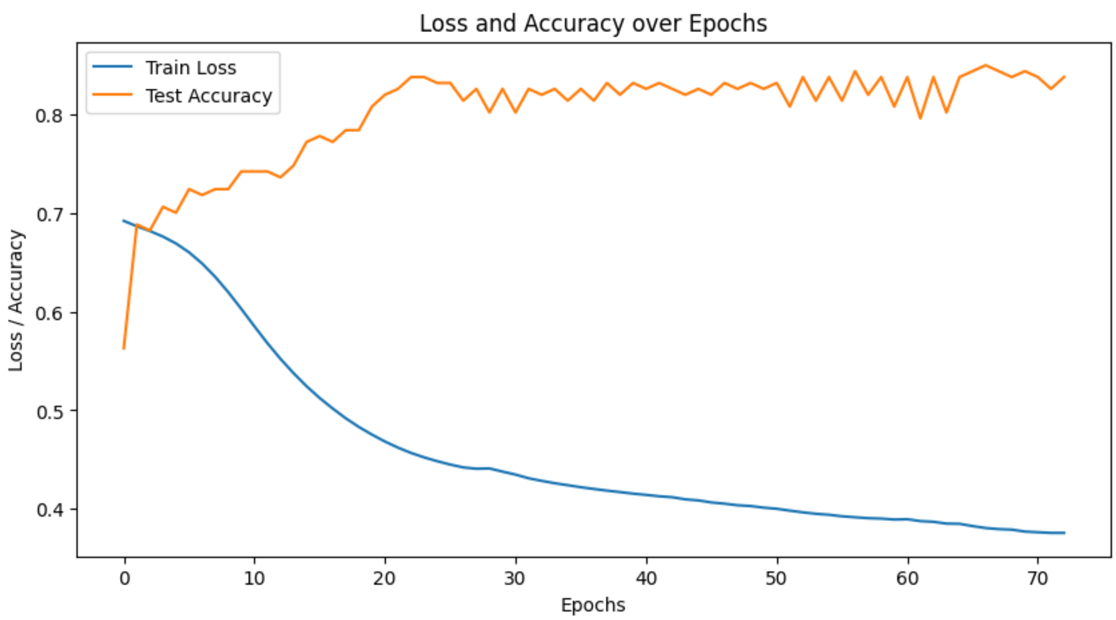
[152]: <Axes: >



[153]:
```
print(f"Accuracy: {accuracy_score(y_test_tensor, y_pred):.4f}")
```

```
Accuracy: 0.8443
```

[156]:
```
plt.figure(figsize=(10, 5))
sns.lineplot(x=range(len(loss_list)), y=loss_list, label='Train Loss')
sns.lineplot(x=range(len(accuracy_list)), y=accuracy_list, label='Test␣
 ↪Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss / Accuracy')
plt.title('Loss and Accuracy over Epochs')
plt.legend()
plt.show()
```




Training the neural network on the unbalanced dataset

[157]:
```
epochs = 400
best_model_loss1 = float('inf')
best_model_weights1 = None
patience = 7

loss_list1 = []
accuracy_list1 = []
```

```python
for i in range(epochs):
    # train_loss = 0.0
    nn_model_02.train()

    outputs1 = nn_model_02(X_train_tensor1)
    loss1 = loss_fn2(outputs1, y_train_tensor1)
    loss_list1.append(loss1.item())

    optimizer2.zero_grad()
    loss1.backward()
    optimizer2.step()

    nn_model_02.eval()
    with torch.no_grad():
        outputs1 = nn_model_02(X_test_tensor1)
        test_loss1 = loss_fn2(outputs1, y_test_tensor1)
    y_pred = torch.sigmoid(outputs1).round()
    accuracy = (y_pred == y_test_tensor1).float().mean().item()
    accuracy_list1.append(accuracy)

    if test_loss1 < best_model_loss1:
        best_model_loss1 = test_loss1
        best_model_weights1 = copy.deepcopy(nn_model_02.state_dict())
        patience_counter1 = 0
    else:
        patience_counter1 += 1
        if patience_counter1 >= patience:
            print(f"Early stopping at epoch {i+1}")
            break
    print(f"Epoch {i+1}/{epochs}, Train Loss: {loss1.item():.4f}, Val Loss:
 ↪ {test_loss1.item():.4f}")
```

```
Epoch 1/400, Train Loss: 0.6982, Val Loss: 0.6940
Epoch 2/400, Train Loss: 0.6881, Val Loss: 0.6849
Epoch 3/400, Train Loss: 0.6794, Val Loss: 0.6753
Epoch 4/400, Train Loss: 0.6702, Val Loss: 0.6646
Epoch 5/400, Train Loss: 0.6596, Val Loss: 0.6520
Epoch 6/400, Train Loss: 0.6469, Val Loss: 0.6371
Epoch 7/400, Train Loss: 0.6317, Val Loss: 0.6201
Epoch 8/400, Train Loss: 0.6138, Val Loss: 0.6022
Epoch 9/400, Train Loss: 0.5938, Val Loss: 0.5833
Epoch 10/400, Train Loss: 0.5725, Val Loss: 0.5659
Epoch 11/400, Train Loss: 0.5512, Val Loss: 0.5519
Epoch 12/400, Train Loss: 0.5309, Val Loss: 0.5404
```

```
Epoch 13/400, Train Loss: 0.5126, Val Loss: 0.5316
Epoch 14/400, Train Loss: 0.4969, Val Loss: 0.5252
Epoch 15/400, Train Loss: 0.4829, Val Loss: 0.5201
Epoch 16/400, Train Loss: 0.4703, Val Loss: 0.5167
Epoch 17/400, Train Loss: 0.4592, Val Loss: 0.5143
Epoch 18/400, Train Loss: 0.4492, Val Loss: 0.5122
Epoch 19/400, Train Loss: 0.4401, Val Loss: 0.5112
Epoch 20/400, Train Loss: 0.4317, Val Loss: 0.5104
Epoch 21/400, Train Loss: 0.4238, Val Loss: 0.5094
Epoch 22/400, Train Loss: 0.4161, Val Loss: 0.5070
Epoch 23/400, Train Loss: 0.4087, Val Loss: 0.5066
Epoch 24/400, Train Loss: 0.4025, Val Loss: 0.5068
Epoch 25/400, Train Loss: 0.3970, Val Loss: 0.5074
Epoch 26/400, Train Loss: 0.3921, Val Loss: 0.5081
Epoch 27/400, Train Loss: 0.3879, Val Loss: 0.5087
Epoch 28/400, Train Loss: 0.3842, Val Loss: 0.5107
Epoch 29/400, Train Loss: 0.3809, Val Loss: 0.5101
Early stopping at epoch 30
```

[158]:
```python
nn_model_02.load_state_dict(best_model_weights1)
nn_model_02.to(device="cpu")
X_test_tensor1 = X_test_tensor1.to(device="cpu")

nn_model_02.eval()
with torch.no_grad():
    outputs1 = nn_model_02(X_test_tensor1)
    y_pred_02 = torch.sigmoid(outputs1).round().numpy()
```

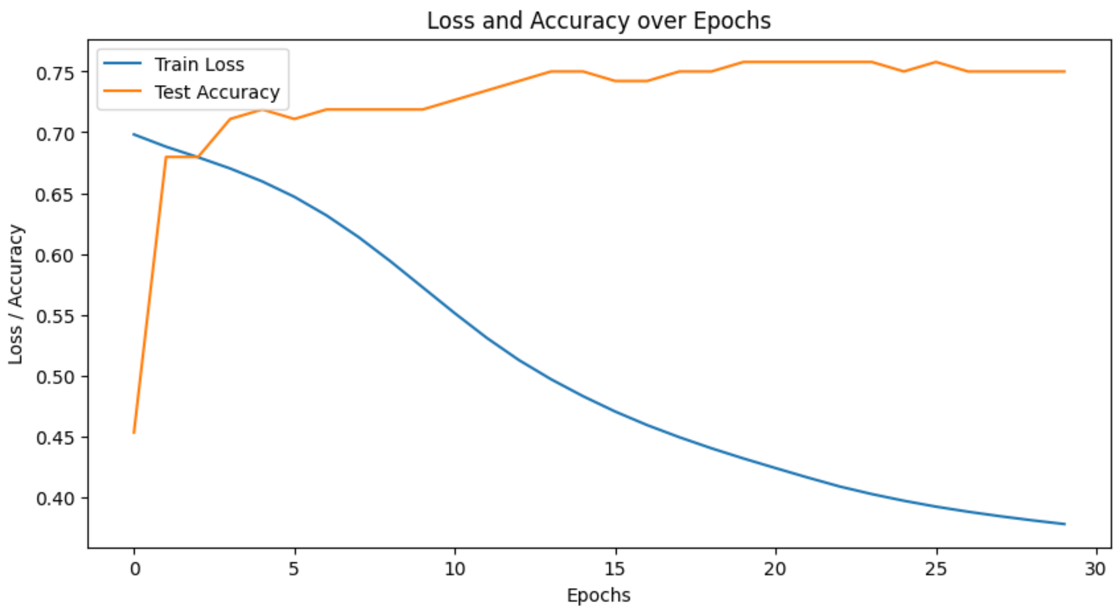
[159]:
```python
y_test_tensor1 = y_test_tensor1.to(device="cpu")

print("Classification Report:")
print(classification_report(y_test_tensor1, y_pred_02))
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.80      0.88      0.84        92
         1.0       0.59      0.44      0.51        36

    accuracy                           0.76       128
   macro avg       0.70      0.66      0.67       128
weighted avg       0.74      0.76      0.75       128
```

[160]:
```python
print(f"Accuracy: {accuracy_score(y_test_tensor1, y_pred_02):.4f}")
```

```
Accuracy: 0.7578
```

[161]:
```python
plt.figure(figsize=(10, 5))
sns.lineplot(x=range(len(loss_list1)), y=loss_list1, label='Train Loss')
sns.lineplot(x=range(len(accuracy_list1)), y=accuracy_list1, label='Test␣
↪Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss / Accuracy')
plt.title('Loss and Accuracy over Epochs')
plt.legend()
plt.show()
```




Training a Support Vector Machine on the balanced dataset

[162]:
```python
svc_model = SVC(random_state=42, kernel='rbf')
```

[163]:
```python
svc_model.fit(X_train, y_train)
```

[163]: SVC(random_state=42)

[164]:
```python
y_pred = svc_model.predict(X_test)
```

[165]:
```python
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.8084
```

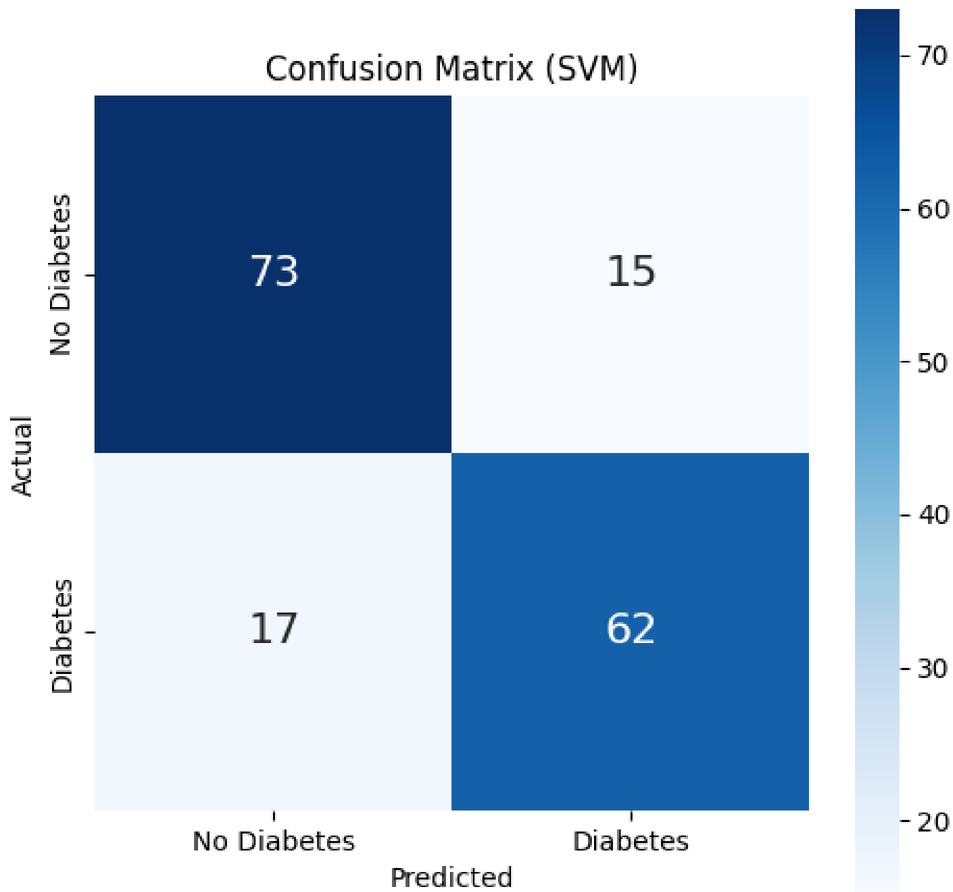
[166]:

```
confusion_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", confusion_mat)
```

```
Confusion Matrix:
 [[73 15]
 [17 62]]
```

[167]:

```
plt.figure(figsize=(6, 6))
sns.heatmap(confusion_mat, annot=True, annot_kws={"size": 16}, fmt='d',␣
 ↪cmap='Blues', xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No␣
 ↪Diabetes", "Diabetes"], square=True)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix (SVM)")
plt.show()
```




[168]:

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

```
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.83      0.82        88
           1       0.81      0.78      0.79        79

    accuracy                           0.81       167
   macro avg       0.81      0.81      0.81       167
weighted avg       0.81      0.81      0.81       167
```

Training a Support Vector Machine on the unbalanced dataset.

[169]:
```
svc_model_02 = SVC(random_state=42, kernel='rbf')
```

[170]:
```
svc_model_02.fit(X_train_02, y_train_02)
```

[170]:
```
SVC(random_state=42)
```

[171]:
```
y_pred_02 = svc_model_02.predict(X_test_02)
```

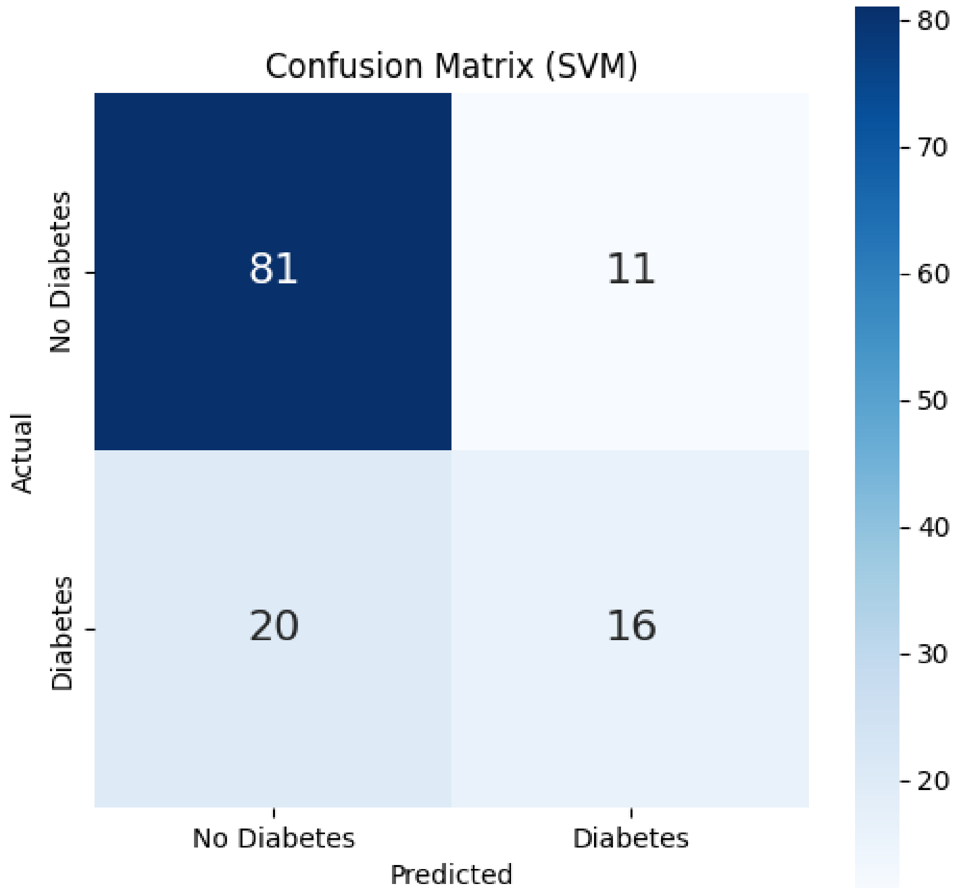
[172]:
```
accuracy_02 = accuracy_score(y_test_02, y_pred_02)
print(f"Accuracy: {accuracy_02:.4f}")
```

```
Accuracy: 0.7578
```

[173]:
```
confusion_mat = confusion_matrix(y_test_02, y_pred_02)
print("Confusion Matrix:\n", confusion_mat)
```

```
Confusion Matrix:
 [[81 11]
 [20 16]]
```

[174]:
```
plt.figure(figsize=(6, 6))
sns.heatmap(confusion_mat, annot=True, annot_kws={"size": 16}, fmt='d',␣
 ↪cmap='Blues', xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No␣
 ↪Diabetes", "Diabetes"], square=True)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix (SVM)")
plt.show()
```




[175]:
```
print("Classification Report:\n", classification_report(y_test_02,␣
↪y_pred_02))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.88      0.84        92
           1       0.59      0.44      0.51        36

    accuracy                           0.76       128
   macro avg       0.70      0.66      0.67       128
weighted avg       0.74      0.76      0.75       128
```

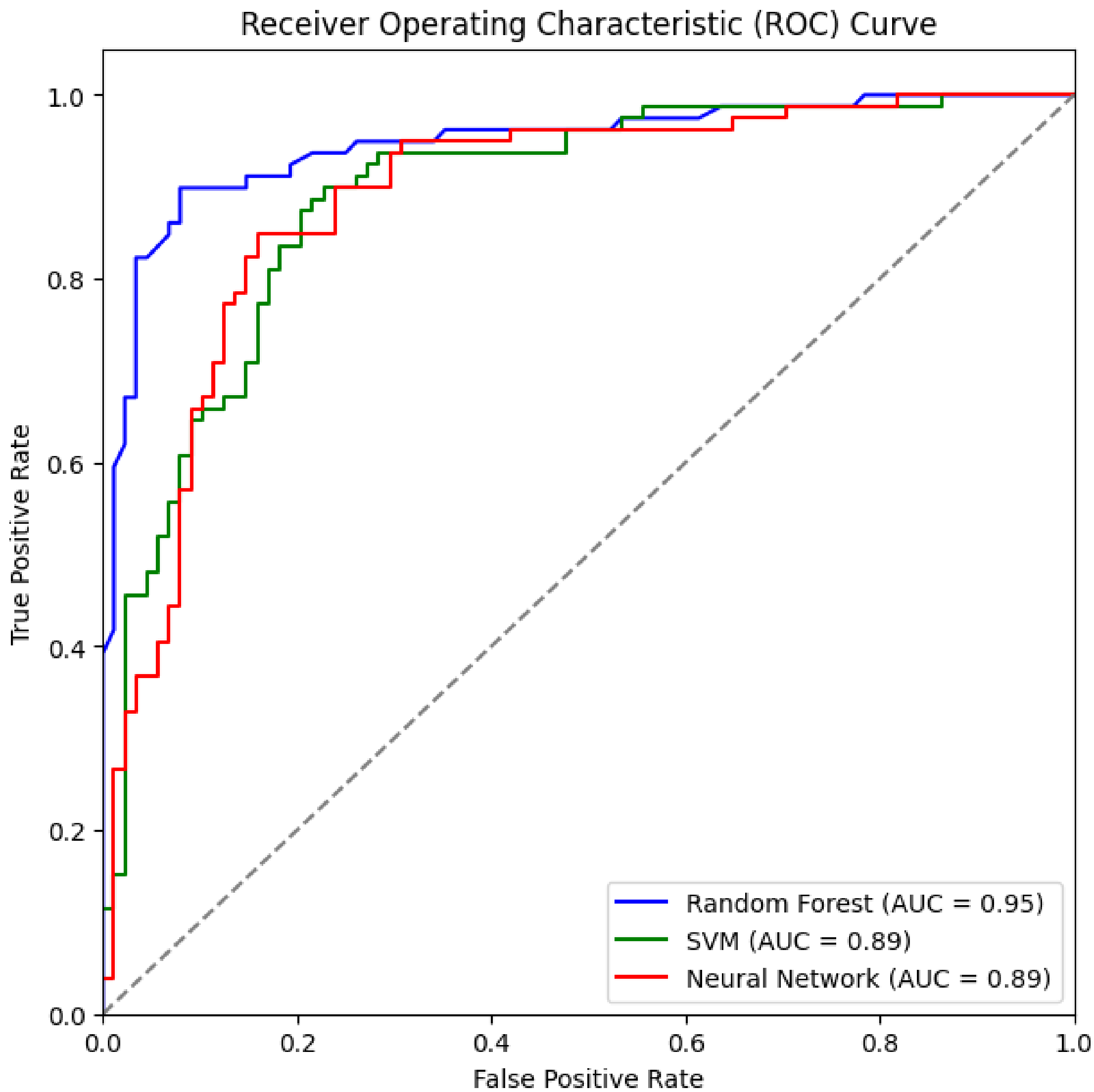
Plotting the ROC-AUC curves of the models trained on the balanced dataset

[176]:
```
y_pred_rf = rf_model.predict_proba(X_test)[:, 1]
y_pred_svc = svc_model.decision_function(X_test)
```

```
y_pred_nn = torch.sigmoid(nn_model_01(X_test_tensor)).detach().numpy()
```

[179]:

```
plt.figure(figsize=(7, 7))

fpr1, tpr1, _ = roc_curve(y_test, y_pred_rf)
fpr2, tpr2, _ = roc_curve(y_test, y_pred_svc)
fpr3, tpr3, _ = roc_curve(y_test, y_pred_nn)

roc_auc_rf = auc(fpr1, tpr1)
roc_auc_svc = auc(fpr2, tpr2)
roc_auc_nn = auc(fpr3, tpr3)

plt.plot(fpr1, tpr1, color='blue', label=f'Random Forest (AUC =␣
 ↪{roc_auc_rf:.2f})')
plt.plot(fpr2, tpr2, color='green', label=f'SVM (AUC = {roc_auc_svc:.2f})')
plt.plot(fpr3, tpr3, color='red', label=f'Neural Network (AUC =␣
 ↪{roc_auc_nn:.2f})')
plt.plot([0, 1], [0, 1], color='gray', 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('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




[180]:

```
nn_model_01.eval()
with torch.no_grad():
    outputs = nn_model_01(X_test_tensor)
    y_pred = torch.sigmoid(outputs).round().numpy()

results_df = pd.DataFrame({
    'Model': ['Random Forest', 'Neural Network', 'SVM'],
    'Accuracy': [
        accuracy_score(y_test, rf_model.predict(X_test)),
        accuracy_score(y_test, y_pred),
        accuracy_score(y_test, svc_model.predict(X_test)),
    ],
```

```
    'AUC': [roc_auc_rf, roc_auc_svc, roc_auc_nn],
    'F1-Score': [
        f1_score(y_test, rf_model.predict(X_test)),
        f1_score(y_test, y_pred),
        f1_score(y_test, svc_model.predict(X_test)),
    ],
    'Precision': [
        precision_score(y_test, rf_model.predict(X_test)),
        precision_score(y_test, y_pred),
        precision_score(y_test, svc_model.predict(X_test)),
    ],
    'Recall': [
        recall_score(y_test, rf_model.predict(X_test)),
        recall_score(y_test, y_pred),
        recall_score(y_test, svc_model.predict(X_test)),
    ]
})
```

[181]:

```
results_df
```

[181]:

| | Model | Accuracy | AUC | F1-Score | Precision | Recall |
|---|---|---|---|---|---|---|
| 0 | Random Forest | 0.880240 | 0.946778 | 0.878049 | 0.847059 | 0.911392 |
| 1 | Neural Network | 0.844311 | 0.889097 | 0.837500 | 0.827160 | 0.848101 |
| 2 | SVM | 0.808383 | 0.886939 | 0.794872 | 0.805195 | 0.784810 |