In [9]:
```
import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,accuracy_score,classification_report, confusion_matrix
import matplotlib.pyplot as plt
import torch.nn.functional as F
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
import pandas
```

Importation of Dataset 2

In [10]:
```
# CHOOSE DATASET

# Binary classification dataset
diabetes =fetch_openml("diabetes", version = 1, as_frame=True)

X = diabetes.data.values
y = diabetes.target.values

y = np.where(y == "tested_positive", 1, 0)

print("Shape:", X.shape)

print(diabetes.data.head(), "\n \n")# first rows of features
print(diabetes.target.head()) # first rows of target
```

```
Shape: (768, 8)
   preg  plas  pres  skin  insu  mass   pedi  age
0     6   148    72    35     0  33.6  0.627   50
1     1    85    66    29     0  26.6  0.351   31
2     8   183    64     0     0  23.3  0.672   32
3     1    89    66    23    94  28.1  0.167   21
4     0   137    40    35   168  43.1  2.288   33


0    tested_positive
1    tested_negative
2    tested_positive
3    tested_negative
4    tested_positive
Name: class, dtype: category
Categories (2, object): ['tested_negative', 'tested_positive']
```

In [11]:
```
#train test spliting
test_size=0.2
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=test_size, random_state=42)
```

In [12]:
```
# Standardize features
scaler=StandardScaler()
Xtr= scaler.fit_transform(Xtr)
Xte= scaler.transform(Xte)
```

A fixed seed was added to this code to ensure the reproducibility of the analysis. This allowed for manual tuning of the hyperparameters to achieve better model training.

In [13]:
```
import random

seed = 42
torch.manual_seed(seed)
np.random.seed(seed)
random.seed(seed)

# Para GPU
torch.cuda.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
```

```
# Tornar CUDA determinístico
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

For this model, the number of neurons per layer was increased from 64 to 100. This change was made after observing that a higher number of neurons per layer could lead to improved model performance.

In [14]:
```
class MLP(nn.Module):
    def __init__(self, input_size, output_size=1, dropout_prob=0.5):
        super(MLP, self).__init__()

        a = 100

        self.fc1 = nn.Linear(input_size, a)
        self.fc2 = nn.Linear(a, a)
        self.fc3 = nn.Linear(a, a)
        self.fc4 = nn.Linear(a, a)
        self.out = nn.Linear(a, output_size)

        self.dropout = nn.Dropout(p=dropout_prob)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.dropout(x)

        x = F.relu(self.fc2(x))
        x = self.dropout(x)

        x = F.relu(self.fc3(x))
        x = self.dropout(x)

        x = F.relu(self.fc4(x))
        x = self.dropout(x)

        x = self.out(x)
        return x
```

In [15]:
```
num_epochs=150
lr=0.0001
dropout=0.1
batch_size=128
```

This model was trained on the GPU and then transferred to the CPU for use with NumPy. Given the low number of parameters (i.e., the model's low complexity), the time required on the CPU was similar to that on the GPU.

In [16]:
```
# Model, Loss, Optimizer
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#device = "cpu"  # force to use CPU
print(device)
```

```
cuda
```

In [17]:
```
Xtr = torch.tensor(Xtr, dtype=torch.float32).to(device)
ytr = torch.tensor(ytr, dtype=torch.float32).to(device)
Xte = torch.tensor(Xte, dtype=torch.float32).to(device)
yte = torch.tensor(yte, dtype=torch.float32).to("cpu")

# Wrap Xtr and ytr into a dataset
train_dataset = TensorDataset(Xtr, ytr)

# Create DataLoader
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
```

In [18]:
```
model = MLP(input_size=Xtr.shape[1], dropout_prob=dropout).to(device)
criterion = nn.BCEWithLogitsLoss()  # for binary classification
criterion = nn.MSELoss() #for regression
optimizer = optim.Adam(model.parameters(), lr=lr)
```

The model was implemented as a fully connected neural network (MLP) with four hidden layers of 64 neurons each, using ReLU activation functions. The network takes the input features of the dataset and outputs a single value for regression (Diabetes Progression).

In [19]:
```
# Training Loop
import time
start_time = time.time()
for epoch in range(num_epochs):
```

```
    model.train()
    epoch_loss = 0.0

    for batch_x, batch_y in train_dataloader:
        batch_x = batch_x.to(device)
        batch_y = batch_y.to(device)

        logits = model(batch_x)
        loss = criterion(logits, batch_y.view(-1, 1))

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

        epoch_loss += loss.item()

    avg_loss = epoch_loss / len(train_dataloader)
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}")

end_time = time.time()
print(f"Training time: {end_time - start_time:.2f} seconds")
```

```
Epoch [1/150], Loss: 0.3047
Epoch [2/150], Loss: 0.2965
Epoch [3/150], Loss: 0.2853
Epoch [4/150], Loss: 0.2748
Epoch [5/150], Loss: 0.2666
Epoch [6/150], Loss: 0.2612
Epoch [7/150], Loss: 0.2516
Epoch [8/150], Loss: 0.2430
Epoch [9/150], Loss: 0.2342
Epoch [10/150], Loss: 0.2262
Epoch [11/150], Loss: 0.2204
Epoch [12/150], Loss: 0.2118
Epoch [13/150], Loss: 0.2021
Epoch [14/150], Loss: 0.2008
Epoch [15/150], Loss: 0.1928
Epoch [16/150], Loss: 0.1910
Epoch [17/150], Loss: 0.1854
Epoch [18/150], Loss: 0.1833
Epoch [19/150], Loss: 0.1756
Epoch [20/150], Loss: 0.1753
Epoch [21/150], Loss: 0.1742
Epoch [22/150], Loss: 0.1699
Epoch [23/150], Loss: 0.1679
Epoch [24/150], Loss: 0.1621
Epoch [25/150], Loss: 0.1665
Epoch [26/150], Loss: 0.1668
Epoch [27/150], Loss: 0.1589
Epoch [28/150], Loss: 0.1595
Epoch [29/150], Loss: 0.1589
Epoch [30/150], Loss: 0.1579
Epoch [31/150], Loss: 0.1586
Epoch [32/150], Loss: 0.1594
Epoch [33/150], Loss: 0.1581
Epoch [34/150], Loss: 0.1524
Epoch [35/150], Loss: 0.1543
Epoch [36/150], Loss: 0.1496
Epoch [37/150], Loss: 0.1515
Epoch [38/150], Loss: 0.1545
Epoch [39/150], Loss: 0.1566
Epoch [40/150], Loss: 0.1530
Epoch [41/150], Loss: 0.1583
```

```
Epoch [42/150], Loss: 0.1496
Epoch [43/150], Loss: 0.1494
Epoch [44/150], Loss: 0.1520
Epoch [45/150], Loss: 0.1494
Epoch [46/150], Loss: 0.1473
Epoch [47/150], Loss: 0.1484
Epoch [48/150], Loss: 0.1517
Epoch [49/150], Loss: 0.1460
Epoch [50/150], Loss: 0.1500
Epoch [51/150], Loss: 0.1483
Epoch [52/150], Loss: 0.1478
Epoch [53/150], Loss: 0.1515
Epoch [54/150], Loss: 0.1521
Epoch [55/150], Loss: 0.1468
Epoch [56/150], Loss: 0.1484
Epoch [57/150], Loss: 0.1493
Epoch [58/150], Loss: 0.1454
Epoch [59/150], Loss: 0.1473
Epoch [60/150], Loss: 0.1443
Epoch [61/150], Loss: 0.1475
Epoch [62/150], Loss: 0.1423
Epoch [63/150], Loss: 0.1428
Epoch [64/150], Loss: 0.1438
Epoch [65/150], Loss: 0.1434
Epoch [66/150], Loss: 0.1442
Epoch [67/150], Loss: 0.1461
Epoch [68/150], Loss: 0.1430
Epoch [69/150], Loss: 0.1445
Epoch [70/150], Loss: 0.1458
Epoch [71/150], Loss: 0.1419
Epoch [72/150], Loss: 0.1426
Epoch [73/150], Loss: 0.1438
Epoch [74/150], Loss: 0.1436
Epoch [75/150], Loss: 0.1452
Epoch [76/150], Loss: 0.1464
Epoch [77/150], Loss: 0.1443
Epoch [78/150], Loss: 0.1388
Epoch [79/150], Loss: 0.1399
Epoch [80/150], Loss: 0.1416
Epoch [81/150], Loss: 0.1383
Epoch [82/150], Loss: 0.1432
```

```
Epoch [83/150], Loss: 0.1387
Epoch [84/150], Loss: 0.1417
Epoch [85/150], Loss: 0.1406
Epoch [86/150], Loss: 0.1373
Epoch [87/150], Loss: 0.1430
Epoch [88/150], Loss: 0.1435
Epoch [89/150], Loss: 0.1341
Epoch [90/150], Loss: 0.1422
Epoch [91/150], Loss: 0.1343
Epoch [92/150], Loss: 0.1393
Epoch [93/150], Loss: 0.1401
Epoch [94/150], Loss: 0.1424
Epoch [95/150], Loss: 0.1357
Epoch [96/150], Loss: 0.1378
Epoch [97/150], Loss: 0.1424
Epoch [98/150], Loss: 0.1363
Epoch [99/150], Loss: 0.1360
Epoch [100/150], Loss: 0.1367
Epoch [101/150], Loss: 0.1409
Epoch [102/150], Loss: 0.1403
Epoch [103/150], Loss: 0.1374
Epoch [104/150], Loss: 0.1308
Epoch [105/150], Loss: 0.1308
Epoch [106/150], Loss: 0.1387
Epoch [107/150], Loss: 0.1351
Epoch [108/150], Loss: 0.1383
Epoch [109/150], Loss: 0.1396
Epoch [110/150], Loss: 0.1402
Epoch [111/150], Loss: 0.1359
Epoch [112/150], Loss: 0.1393
Epoch [113/150], Loss: 0.1298
Epoch [114/150], Loss: 0.1328
Epoch [115/150], Loss: 0.1302
Epoch [116/150], Loss: 0.1371
Epoch [117/150], Loss: 0.1325
Epoch [118/150], Loss: 0.1379
Epoch [119/150], Loss: 0.1357
Epoch [120/150], Loss: 0.1342
Epoch [121/150], Loss: 0.1400
Epoch [122/150], Loss: 0.1363
Epoch [123/150], Loss: 0.1359
```

```
Epoch [124/150], Loss: 0.1322
Epoch [125/150], Loss: 0.1306
Epoch [126/150], Loss: 0.1338
Epoch [127/150], Loss: 0.1299
Epoch [128/150], Loss: 0.1254
Epoch [129/150], Loss: 0.1297
Epoch [130/150], Loss: 0.1311
Epoch [131/150], Loss: 0.1315
Epoch [132/150], Loss: 0.1311
Epoch [133/150], Loss: 0.1289
Epoch [134/150], Loss: 0.1358
Epoch [135/150], Loss: 0.1307
Epoch [136/150], Loss: 0.1301
Epoch [137/150], Loss: 0.1312
Epoch [138/150], Loss: 0.1297
Epoch [139/150], Loss: 0.1352
Epoch [140/150], Loss: 0.1322
Epoch [141/150], Loss: 0.1337
Epoch [142/150], Loss: 0.1260
Epoch [143/150], Loss: 0.1281
Epoch [144/150], Loss: 0.1324
Epoch [145/150], Loss: 0.1312
Epoch [146/150], Loss: 0.1259
Epoch [147/150], Loss: 0.1283
Epoch [148/150], Loss: 0.1314
Epoch [149/150], Loss: 0.1234
Epoch [150/150], Loss: 0.1310
Training time: 2.52 seconds
```

In [20]:
```
thr = 0.6 #threshold to tune

y_pred=model(Xte).cpu()
y_true = yte.detach().numpy()
y_pred_bin = (y_pred.detach().numpy() > thr).astype(int)  # binary predictions

# Accuracy
acc = accuracy_score(y_true, y_pred_bin)
print(f'ACC:{accuracy_score(y_true,y_pred_bin)}') #classification

cm = confusion_matrix(y_true, y_pred_bin)
print("Confusion Matrix:")
```

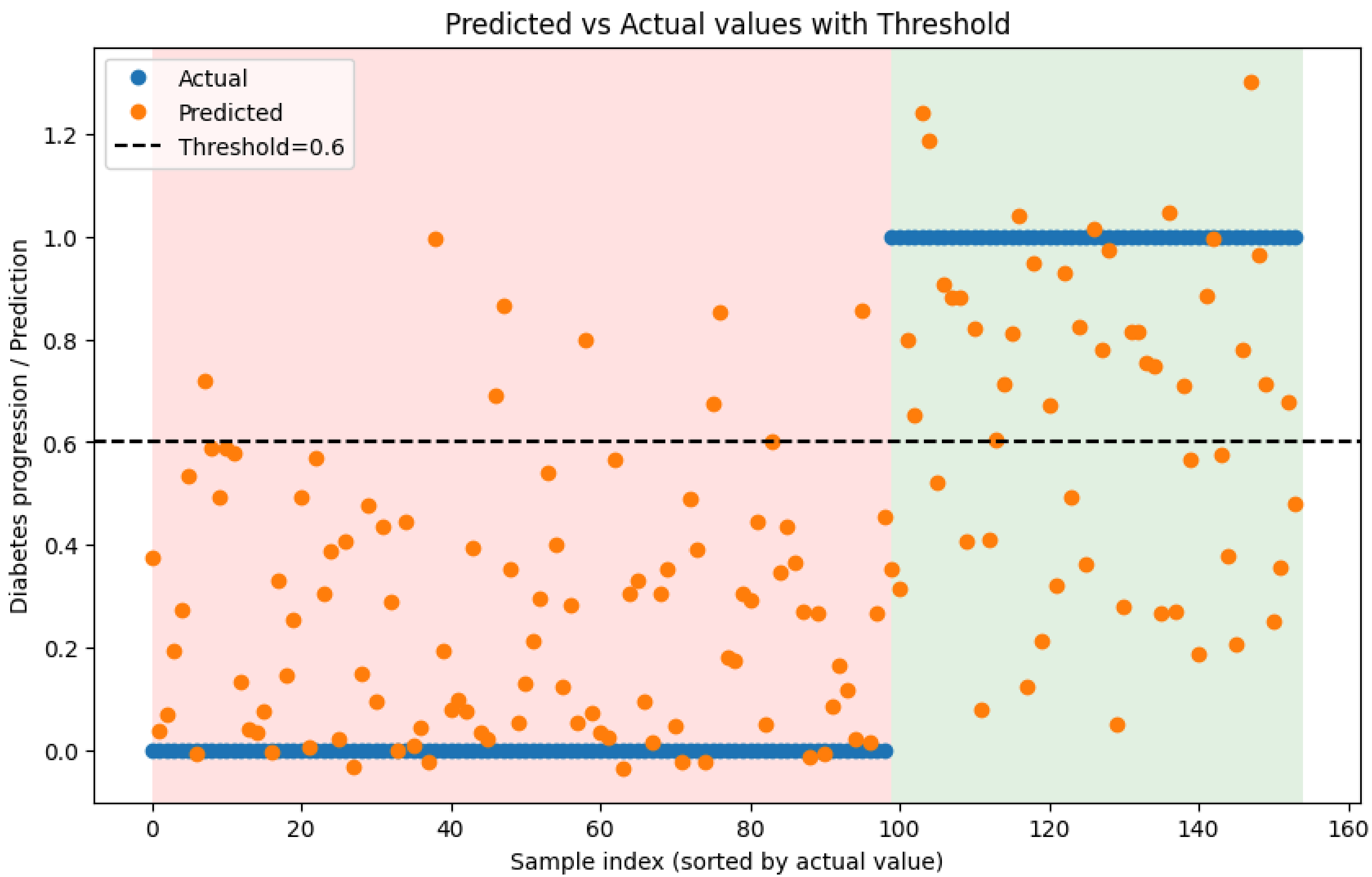
```
print(cm)

#print(f'ACC:{mean_squared_error(yte.detach().numpy(),y_pred.detach().numpy())}') #regression
```

```
ACC:0.7922077922077922
Confusion Matrix:
[[90  9]
 [23 32]]
```

After manually tuning the hyperparameters, the best configuration obtained was: num_epochs = 150, lr = 0.0001, dropout = 0.1, and batch_size = 128. Dropout with a probability of 0.1 was applied after each layer to mitigate overfitting. To validate the model, both accuracy and the confusion matrix were computed. An accuracy of 79.22% was achieved, which is comparable to other models. It was observed that the model performs better when predicting negative cases compared to positive cases. This behavior is likely due to the dataset being unbalanced, highlighting the importance of data quality for achieving reliable model performance. One possible approach to address this issue is to oversample the positive class so that both classes have similar representation during training.

In [21]:
```
# Convert y_pred to numpy and flatten
y_pred_np = y_pred.detach().numpy().flatten()
y_pred_bin_np = y_pred_bin
yte_np = yte.detach().numpy()

# Get indices that sort yte
sort_idx = np.argsort(yte_np)

# Sort yte and y_pred
yte_sorted = yte_np[sort_idx]
y_pred_sorted = y_pred_np[sort_idx]

# Determine the index where actual target switches from 0 to 1
frontier_idx = np.argmax(yte_sorted == 1)  # first occurrence of 1

# Plot
plt.figure(figsize=(10,6))

# Shade negative region (yte = 0)
plt.axvspan(0, frontier_idx, facecolor='red', alpha=0.1)

# Shade positive region (yte = 1)
plt.axvspan(frontier_idx, len(yte_sorted), facecolor='green', alpha=0.1)
```

```python
# Actual and predicted points
plt.plot(range(len(yte_sorted)), yte_sorted, label="Actual", marker="o", linestyle='')
plt.plot(range(len(y_pred_sorted)), y_pred_sorted, label="Predicted", marker="o", linestyle='')

# Threshold line
plt.axhline(y=thr, color='k', linestyle='--', label=f'Threshold={thr}')

plt.xlabel("Sample index (sorted by actual value)")
plt.ylabel("Diabetes progression / Prediction")
plt.title("Predicted vs Actual values with Threshold")
plt.legend()
plt.show()
```


Predicted vs Actual values with Threshold
Actual
Predicted
Threshold=0.6
Diabetes progression / Prediction
Sample index (sorted by actual value)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
20
40
60
80
100
120
140
160

```
In [ ]:
```