

```
In [1]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
In [2]: df = pd.read_csv(r"C:\Users\Zhi\Desktop\diabetes.csv")
```

```
In [3]: df.describe()
```

Out[3]:

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure	Triceps skin fold thickness	2-Hour serum insulin	Body mass index	Diabetes pedigree function	Age	Class variable
<b>count</b>	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
<b>mean</b>	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
<b>std</b>	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
<b>min</b>	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
<b>25%</b>	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
<b>50%</b>	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
<b>75%</b>	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
<b>max</b>	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
In [4]: df.head()
#some features with 0 input are consider as NA. For example, BMI, 2-hour serum insulin, skin fold thickness can't be
```

Out[4]:

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure	Triceps skin fold thickness	2-Hour serum insulin	Body mass index	Diabetes pedigree function	Age	Class variable
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

```
In [5]: Need_fix_cols = ["Plasma glucose concentration a 2 hours in an oral glucose tolerance test", "Diastolic blood pressure",  
                        "Diastolic blood pressure", "Triceps skin fold thickness", "2-Hour serum insulin", "Body mass index",  
                        "Diabetes pedigree function", "Age"]  
#Grouping these features, because their output can't be 0
```

```
In [6]: Need_fix_cols= list(dict.fromkeys(Need_fix_cols))
```

```
In [7]: df[Need_fix_cols] = df[Need_fix_cols].replace(0, np.nan)
```

```
In [8]: df.fillna(df.median(),inplace=True) #replace with median values
```

```
In [9]: df.head()
```

Out[9]:

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure	Triceps skin fold thickness	2-Hour serum insulin	Body mass index	Diabetes pedigree function	Age	Class variable
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

```
In [10]: df.isna().sum()
```

```
Out[10]: Number of times pregnant          0
Plasma glucose concentration a 2 hours in an oral glucose tolerance test  0
Diastolic blood pressure              0
Triceps skin fold thickness           0
2-Hour serum insulin                  0
Body mass index                       0
Diabetes pedigree function            0
Age                                   0
Class variable                        0
dtype: int64
```

```
In [11]: df.duplicated().sum()
```

```
Out[11]: np.int64(0)
```

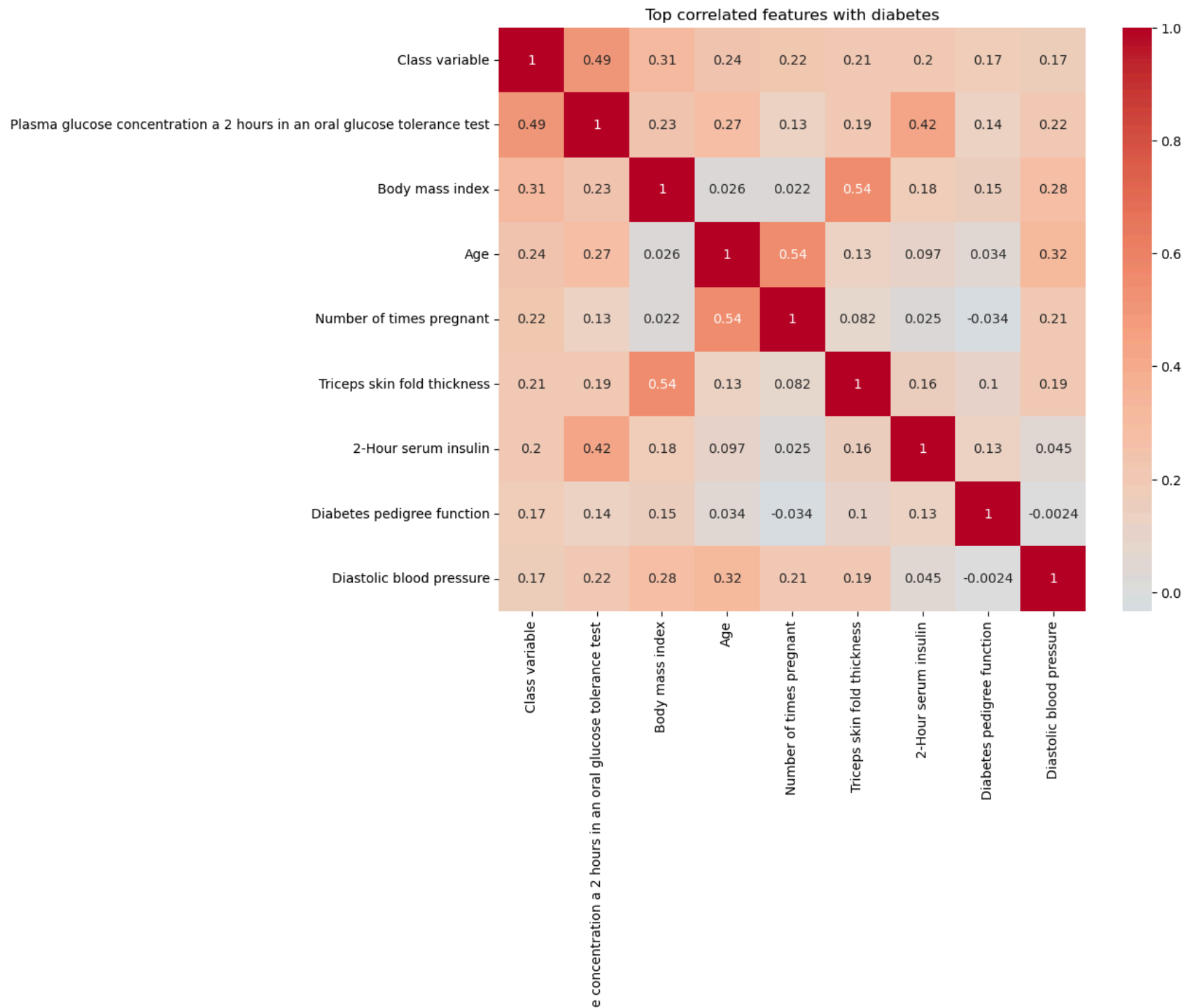
```
In [12]: df.corr()
```

Out[12]:

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure	Triceps skin fold thickness	2-Hour serum insulin	Body mass index	Diabetes pedigree function	Age	Class variable
Number of times pregnant	1.000000	0.128213	0.208615	0.081770	0.025047	0.021559	-0.033523	0.544341	0.221898
Plasma glucose concentration a 2 hours in an oral glucose tolerance test	0.128213	1.000000	0.218937	0.192615	0.419451	0.231049	0.137327	0.266909	0.492782
Diastolic blood pressure	0.208615	0.218937	1.000000	0.191892	0.045363	0.281257	-0.002378	0.324915	0.165723
Triceps skin fold thickness	0.081770	0.192615	0.191892	1.000000	0.155610	0.543205	0.102188	0.126107	0.214873
2-Hour serum insulin	0.025047	0.419451	0.045363	0.155610	1.000000	0.180241	0.126503	0.097101	0.203790
Body mass index	0.021559	0.231049	0.281257	0.543205	0.180241	1.000000	0.153438	0.025597	0.312038
Diabetes pedigree function	-0.033523	0.137327	-0.002378	0.102188	0.126503	0.153438	1.000000	0.033561	0.173844
Age	0.544341	0.266909	0.324915	0.126107	0.097101	0.025597	0.033561	1.000000	0.238356
Class variable	0.221898	0.492782	0.165723	0.214873	0.203790	0.312038	0.173844	0.238356	1.000000

In [13]:

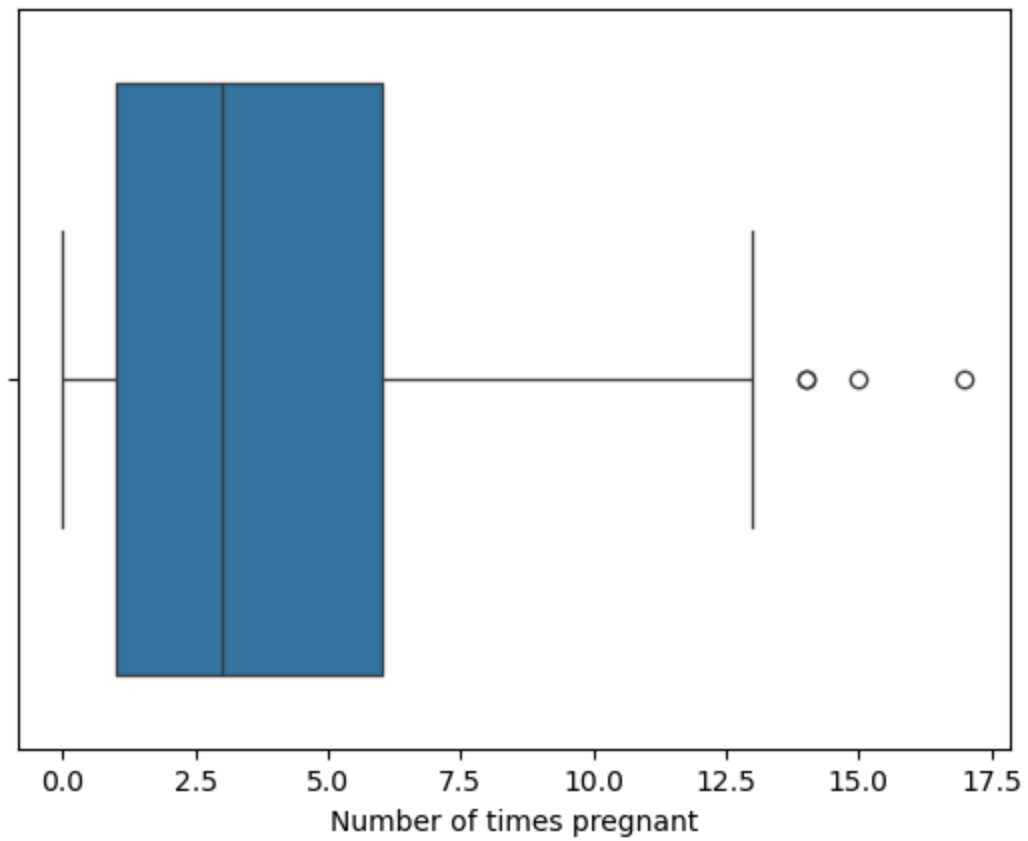
```
corr= df.corr()
top =corr["Class variable"].abs().sort_values(ascending=False).head(10).index
plt.figure(figsize=(10,8))
sns.heatmap(df[top].corr(),cmap="coolwarm",center= 0,annot=True)
plt.title("Top correlated features with diabetes")
plt.show()
#As we can see, glucose has highest positivie correlation with diabetes(0.49)
#BMI and age are the secondary and third factors of getting diabetes.
```



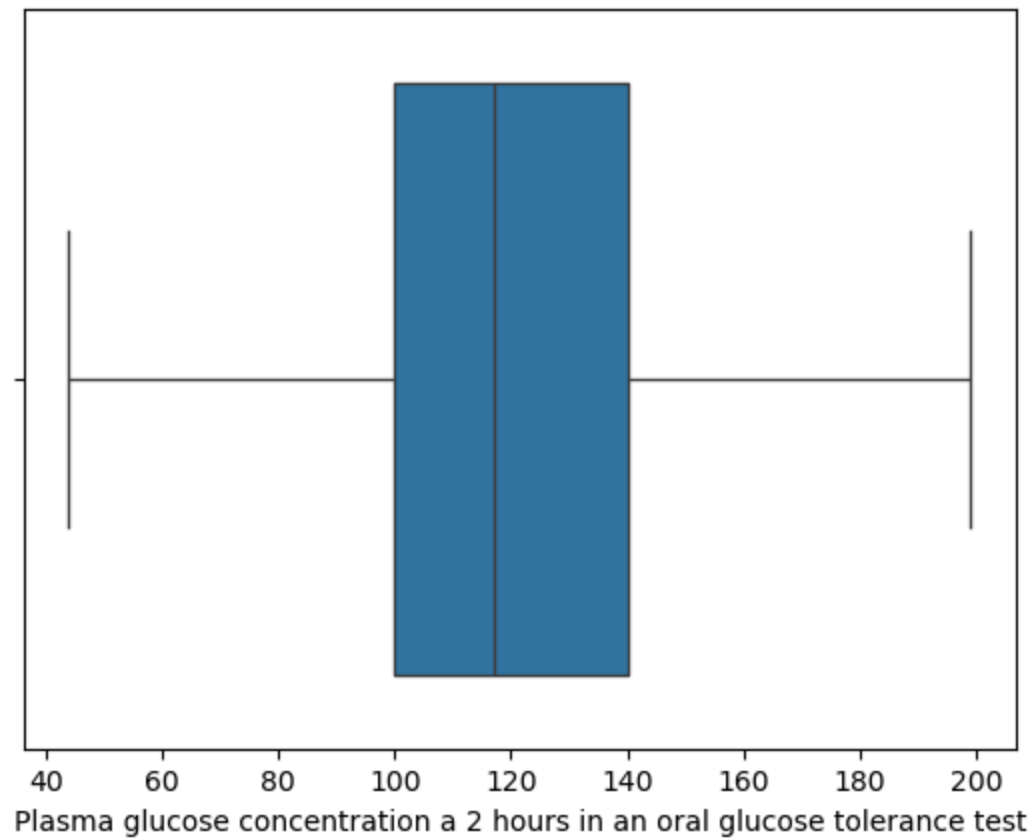
```
In [14]: # Im curious about does higer pregnancy lead to higher chance of diabetes?
df.groupby("Class variable")["Number of times pregnant"].mean()
#The answer is yes, higher pregnancy has higher chance o
```

```
Out[14]: Class variable
0      3.298000
1      4.865672
Name: Number of times pregnant, dtype: float64
```

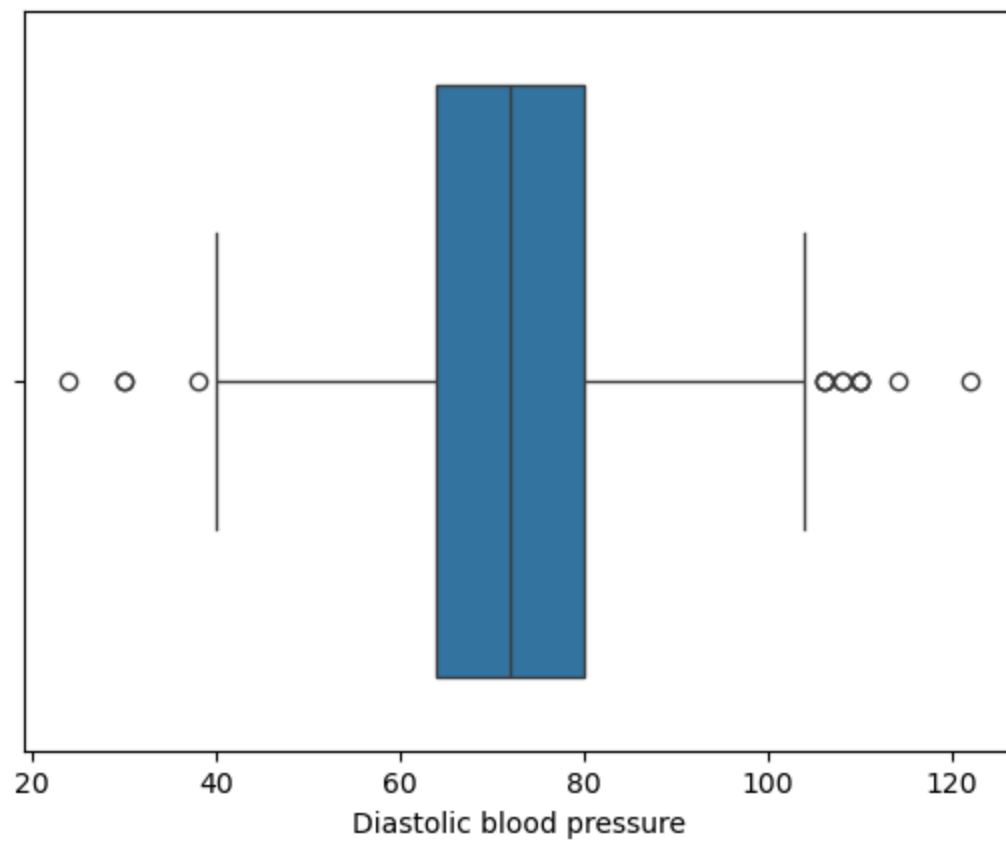
```
In [15]: sns.boxplot(x=df['Number of times pregnant'])
plt.show()
#Not sure, but biologically could happen.
```



```
In [16]: sns.boxplot(x=df['Plasma glucose concentration a 2 hours in an oral glucose tolerance test'])  
plt.show()
```



```
In [17]: sns.boxplot(x=df['Diastolic blood pressure'])  
plt.show()  
# whatever less than 40 blood pressure might consider as error.
```

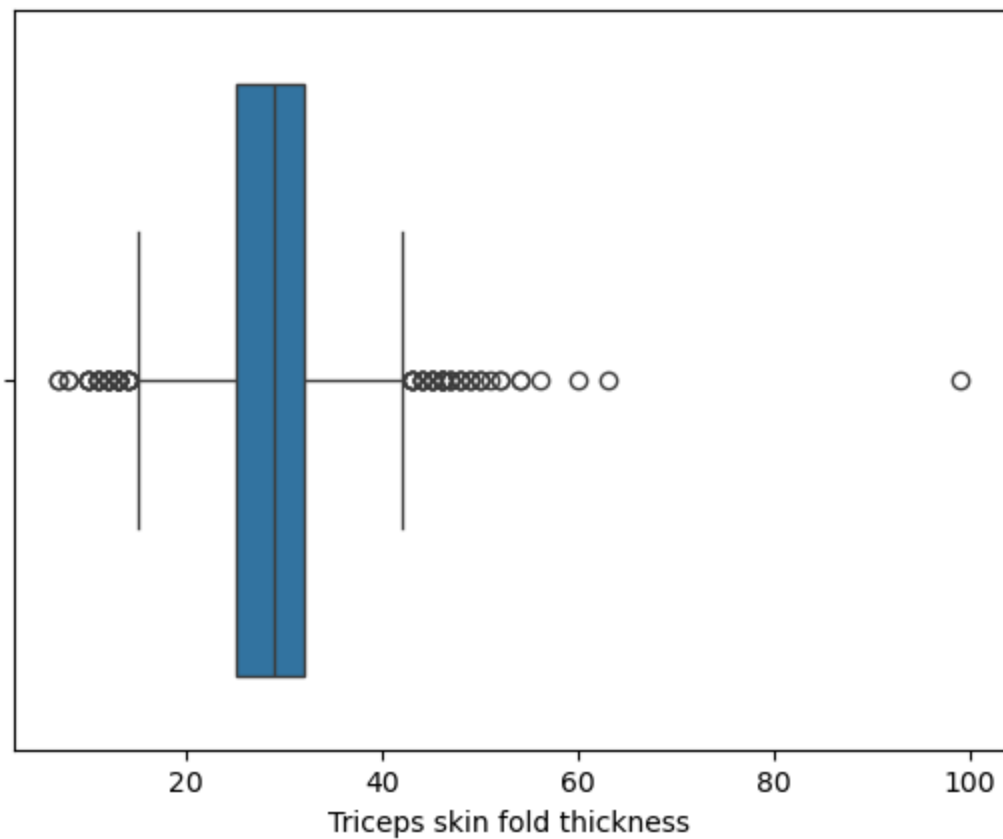


```
In [18]: df['Diastolic blood pressure'] = df['Diastolic blood pressure'].replace(0, np.nan)
```

```
In [19]: df['Diastolic blood pressure'].fillna(df['Diastolic blood pressure'].median())
```

```
Out[19]: 0      72.0
          1      66.0
          2      64.0
          3      66.0
          4      40.0
          ...
          763    76.0
          764    70.0
          765    72.0
          766    60.0
          767    70.0
          Name: Diastolic blood pressure, Length: 768, dtype: float64
```

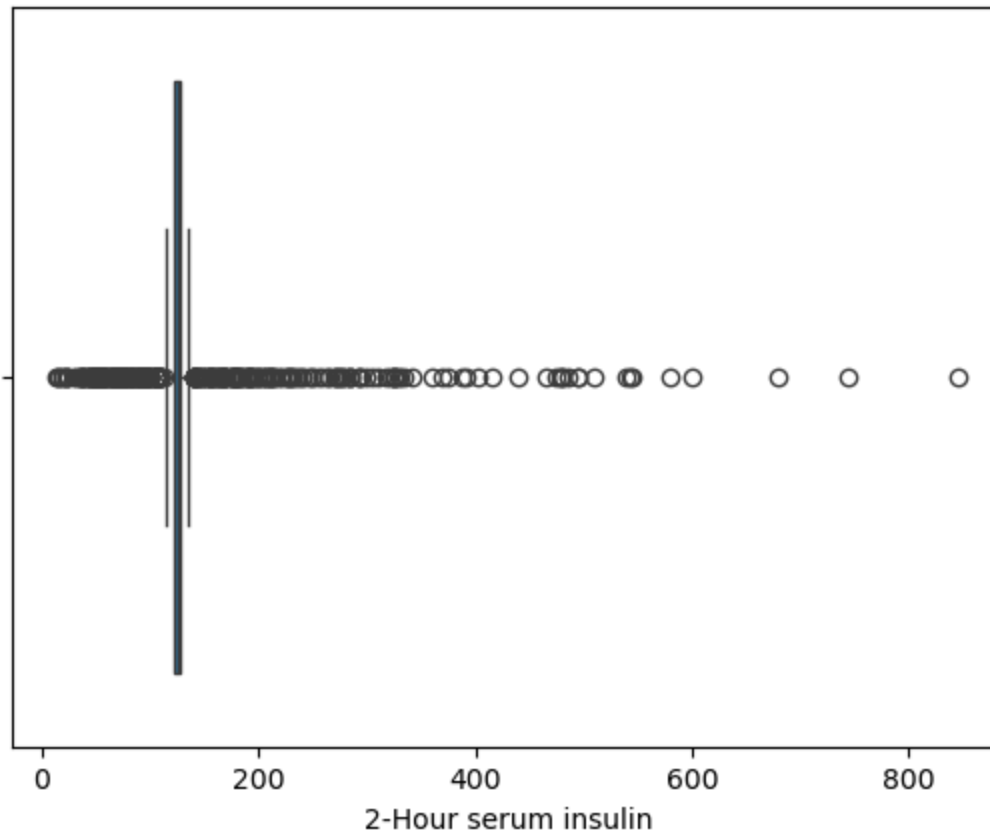
```
In [20]: sns.boxplot(x=df['Triceps skin fold thickness'])
          plt.show()
```



```
In [21]: df['Triceps skin fold thickness'] = df['Triceps skin fold thickness'].replace(0, np.nan)
df['Triceps skin fold thickness'].fillna(df['Triceps skin fold thickness'].median())
```

```
Out[21]: 0      35.0
1      29.0
2      29.0
3      23.0
4      35.0
...
763    48.0
764    27.0
765    23.0
766    29.0
767    31.0
Name: Triceps skin fold thickness, Length: 768, dtype: float64
```

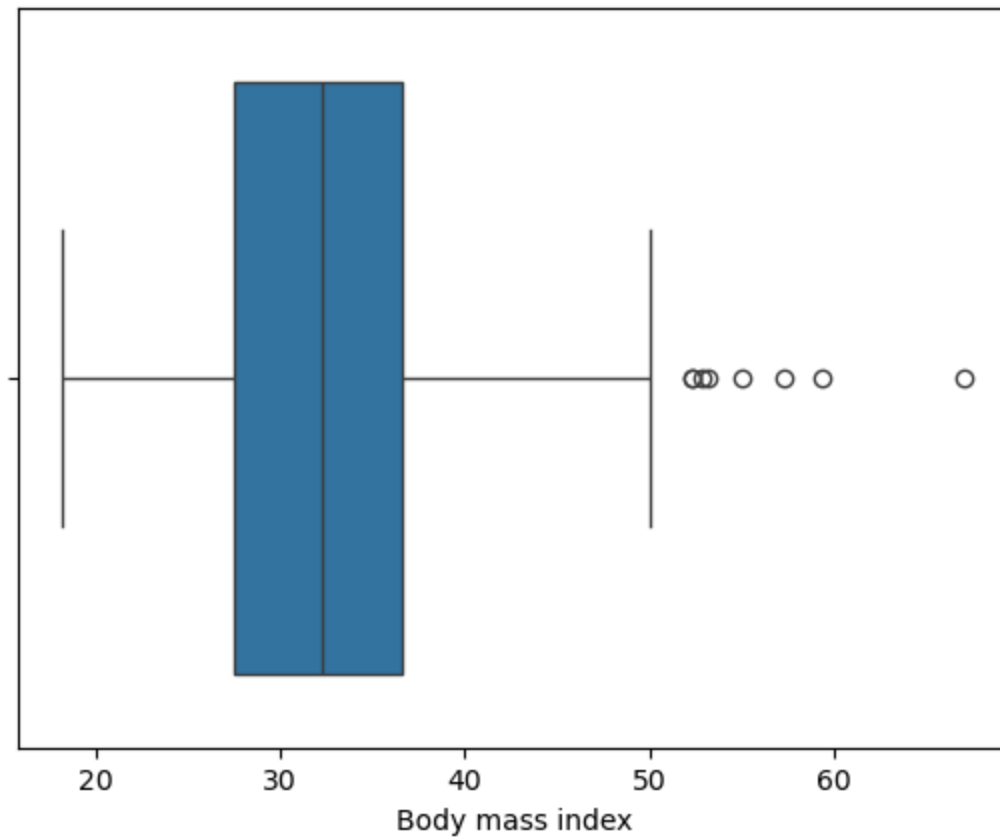
```
In [22]: sns.boxplot(x=df['2-Hour serum insulin'])
plt.show()
```



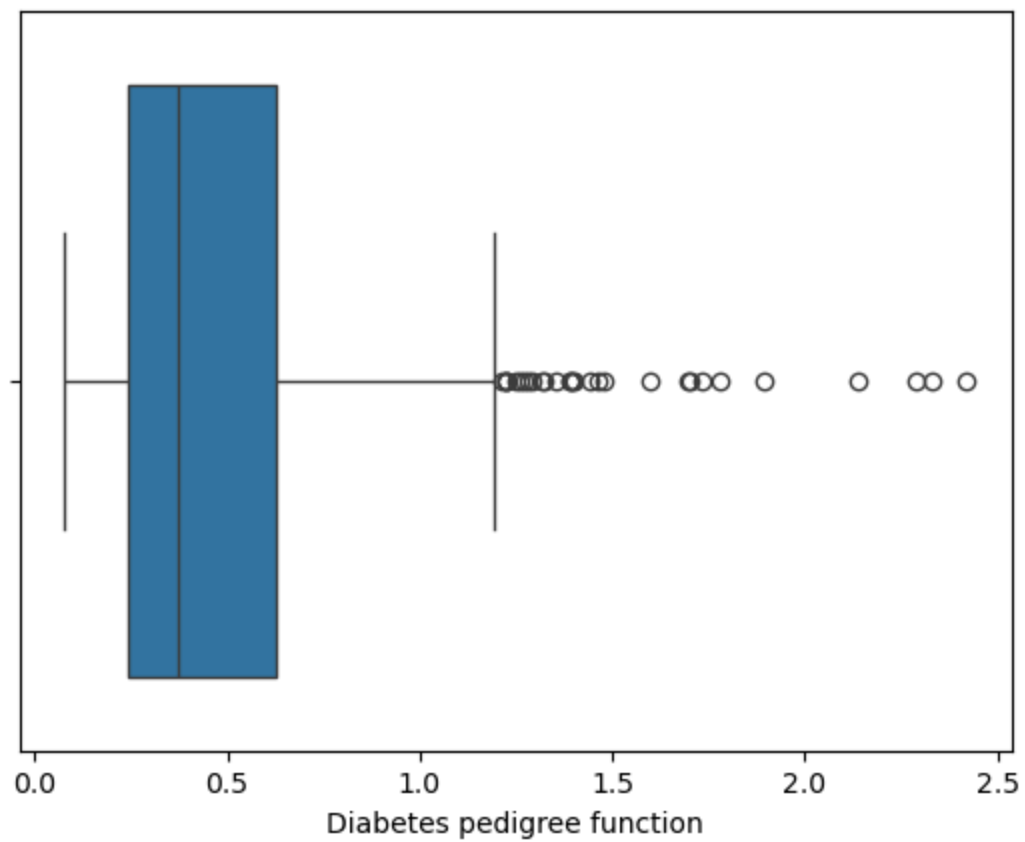
```
In [23]: df['2-Hour serum insulin'].replace(0, np.nan)
df['2-Hour serum insulin'].fillna(df['2-Hour serum insulin'].median())
```

```
Out[23]: 0      125.0
1      125.0
2      125.0
3       94.0
4      168.0
...
763    180.0
764    125.0
765    112.0
766    125.0
767    125.0
Name: 2-Hour serum insulin, Length: 768, dtype: float64
```

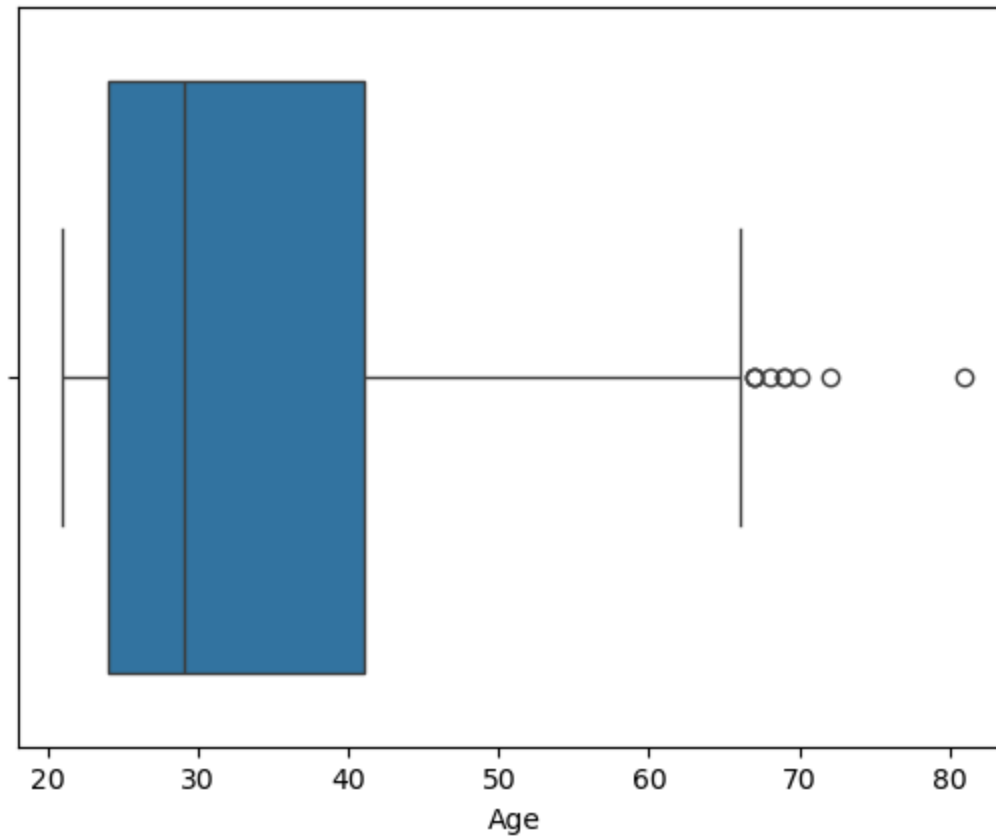
```
In [24]: sns.boxplot(x=df['Body mass index'])  
plt.show()
```



```
In [25]: sns.boxplot(x=df['Diabetes pedigree function'])  
plt.show()
```



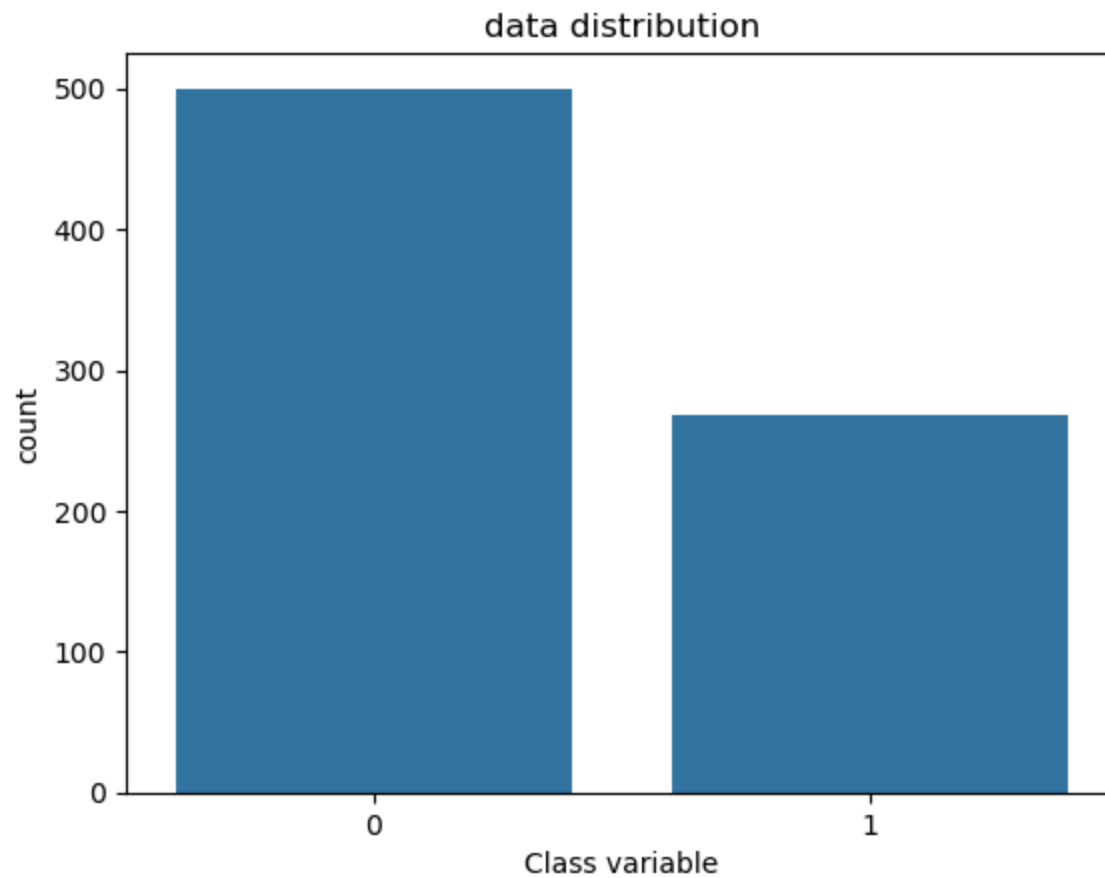
```
In [26]: sns.boxplot(x=df['Age'])  
plt.show()
```



```
In [27]: df['Class variable'].value_counts()
```

```
Out[27]: Class variable  
0      500  
1      268  
Name: count, dtype: int64
```

```
In [28]: sns.countplot(x='Class variable', data=df)  
plt.title('data distribution')  
plt.show()
```



```
In [29]: from sklearn.model_selection import train_test_split
```

```
In [30]: X= df.drop('Class variable', axis=1)
```

```
In [31]: y=df['Class variable']
```

```
In [32]: X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.25, random_state=42, stratify=y)
```

```
In [33]: from sklearn.preprocessing import StandardScaler
```

```
In [34]: scaler = StandardScaler()
```

```
In [35]: X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
In [36]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import RMSprop
```

```
In [37]: model = Sequential()
```

```
In [38]: model.add(Dense(12, activation = 'relu', input_dim = X_train.shape[1]))
```

R:\Python\Lib\site-packages\keras\src\layers\core\dense.py:106: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
In [39]: model.add(Dense(1, activation = 'sigmoid'))
```

```
In [40]: model_optimizer = RMSprop(learning_rate = 0.001)
model.compile(optimizer = model_optimizer, loss = 'binary_crossentropy', metrics=['accuracy'])
```

```
In [41]: model.summary()
```

Model: "sequential"





















Layer (type)	Output Shape	Param #
dense (Dense)	(None, 12)	108
dense_1 (Dense)	(None, 1)	13

Total params: 121 (484.00 B)

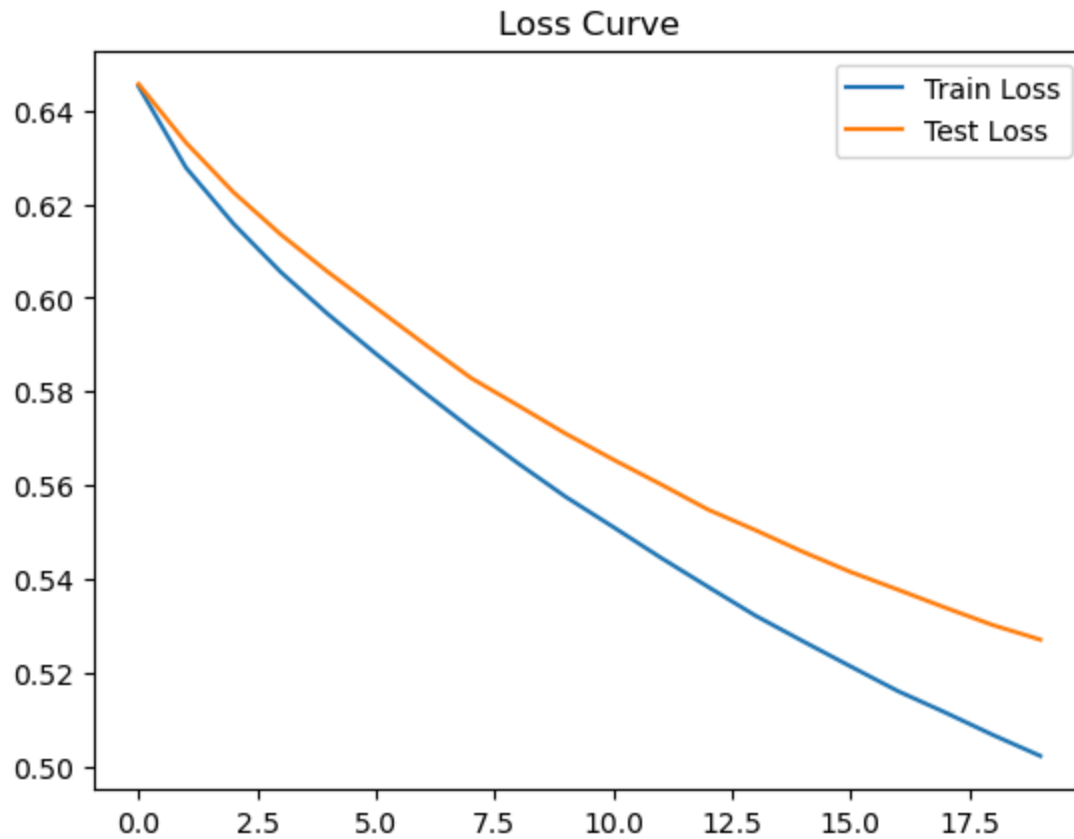
Trainable params: 121 (484.00 B)

Non-trainable params: 0 (0.00 B)

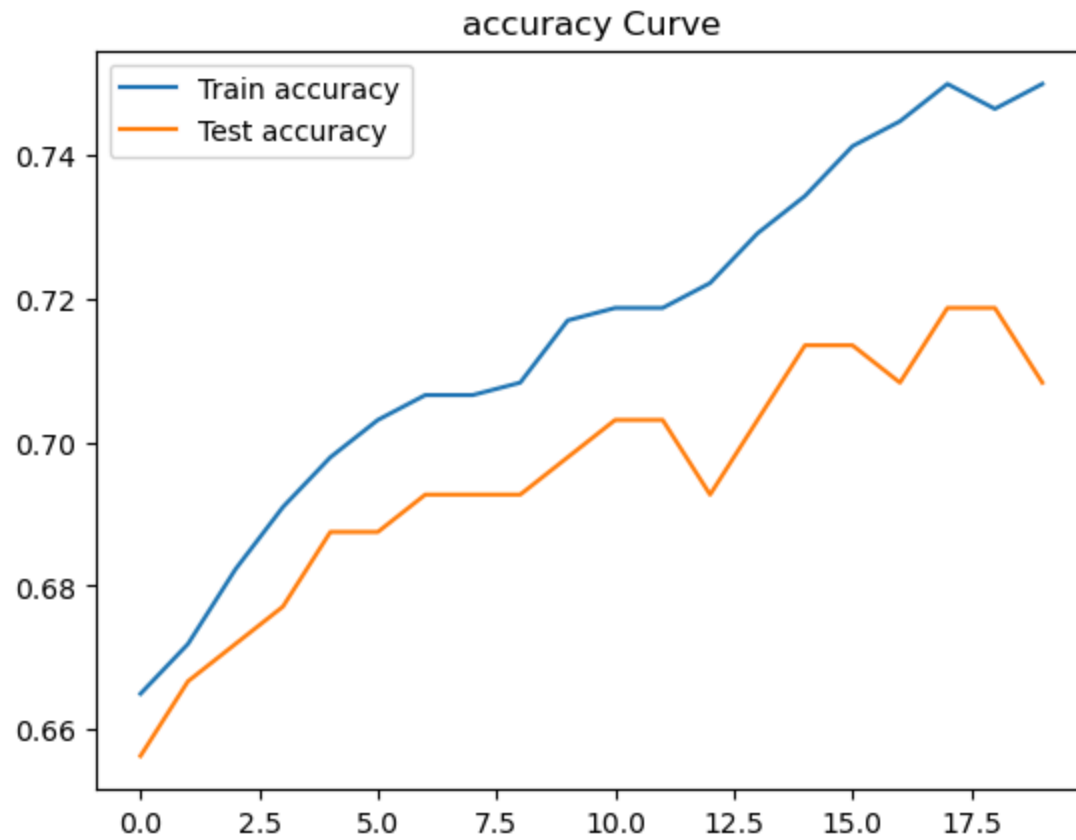
```
In [42]: train_model = model.fit( X_train, y_train, epochs=20, batch_size =128, validation_data =(X_test,y_test))
```

Epoch 1/20  
5/5  1s 42ms/step - accuracy: 0.6649 - loss: 0.6451 - val\_accuracy: 0.6562 - val\_loss: 0.6456  
Epoch 2/20  
5/5  0s 12ms/step - accuracy: 0.6719 - loss: 0.6278 - val\_accuracy: 0.6667 - val\_loss: 0.6331  
Epoch 3/20  
5/5  0s 12ms/step - accuracy: 0.6823 - loss: 0.6158 - val\_accuracy: 0.6719 - val\_loss: 0.6225  
Epoch 4/20  
5/5  0s 12ms/step - accuracy: 0.6910 - loss: 0.6055 - val\_accuracy: 0.6771 - val\_loss: 0.6135  
Epoch 5/20  
5/5  0s 12ms/step - accuracy: 0.6979 - loss: 0.5964 - val\_accuracy: 0.6875 - val\_loss: 0.6054  
Epoch 6/20  
5/5  0s 12ms/step - accuracy: 0.7031 - loss: 0.5881 - val\_accuracy: 0.6875 - val\_loss: 0.5979  
Epoch 7/20  
5/5  0s 12ms/step - accuracy: 0.7066 - loss: 0.5800 - val\_accuracy: 0.6927 - val\_loss: 0.5903  
Epoch 8/20  
5/5  0s 12ms/step - accuracy: 0.7066 - loss: 0.5721 - val\_accuracy: 0.6927 - val\_loss: 0.5829  
Epoch 9/20  
5/5  0s 12ms/step - accuracy: 0.7083 - loss: 0.5647 - val\_accuracy: 0.6927 - val\_loss: 0.5770  
Epoch 10/20  
5/5  0s 12ms/step - accuracy: 0.7170 - loss: 0.5576 - val\_accuracy: 0.6979 - val\_loss: 0.5710  
Epoch 11/20  
5/5  0s 12ms/step - accuracy: 0.7188 - loss: 0.5511 - val\_accuracy: 0.7031 - val\_loss: 0.5655  
Epoch 12/20  
5/5  0s 12ms/step - accuracy: 0.7188 - loss: 0.5446 - val\_accuracy: 0.7031 - val\_loss: 0.5602  
Epoch 13/20  
5/5  0s 12ms/step - accuracy: 0.7222 - loss: 0.5384 - val\_accuracy: 0.6927 - val\_loss: 0.5548  
Epoch 14/20  
5/5  0s 12ms/step - accuracy: 0.7292 - loss: 0.5322 - val\_accuracy: 0.7031 - val\_loss: 0.5504  
Epoch 15/20  
5/5  0s 12ms/step - accuracy: 0.7344 - loss: 0.5267 - val\_accuracy: 0.7135 - val\_loss: 0.5458  
Epoch 16/20  
5/5  0s 12ms/step - accuracy: 0.7413 - loss: 0.5214 - val\_accuracy: 0.7135 - val\_loss: 0.5415  
Epoch 17/20  
5/5  0s 12ms/step - accuracy: 0.7448 - loss: 0.5161 - val\_accuracy: 0.7083 - val\_loss: 0.5377  
Epoch 18/20  
5/5  0s 12ms/step - accuracy: 0.7500 - loss: 0.5115 - val\_accuracy: 0.7188 - val\_loss: 0.5339  
Epoch 19/20  
5/5  0s 13ms/step - accuracy: 0.7465 - loss: 0.5068 - val\_accuracy: 0.7188 - val\_loss: 0.5302  
Epoch 20/20  
5/5  0s 13ms/step - accuracy: 0.7500 - loss: 0.5023 - val\_accuracy: 0.7083 - val\_loss: 0.5271

```
In [43]: plt.plot(train_model.history['loss'], label='Train Loss')
plt.plot(train_model.history['val_loss'], label='Test Loss')
plt.legend(); plt.title('Loss Curve');
plt.show()
```



```
In [44]: plt.plot(train_model.history['accuracy'], label='Train accuracy')
plt.plot(train_model.history['val_accuracy'], label='Test accuracy')
plt.legend(); plt.title('accuracy Curve');
plt.show()
```



```
In [45]: train_loss, train_accuracy= model.evaluate(X_train, y_train, verbose= 0)
```

```
In [46]: test_loss, test_accuracy= model.evaluate(X_test, y_test, verbose= 0)
```

```
In [47]: train_loss, train_accuracy
```

```
Out[47]: (0.4991549253463745, 0.7534722089767456)
```

```
In [48]: test_loss, test_accuracy
```

```
Out[48]: (0.5270856022834778, 0.7083333134651184)
```






















```
In [49]: #Adams model  
from tensorflow.keras.optimizers import Adam
```

```
model_1 = Sequential([Input(shape =(X_train.shape[1],)), Dense(12, activation ='relu'),Dense(1, activation='sigmoid')])
```




















```
In [50]: adam_optimizer = Adam(learning_rate = 0.001)
```

```
In [51]: model_1.compile(optimizer = adam_optimizer , loss ='binary_crossentropy', metrics=['accuracy'])
```

```
In [52]: model_1_result = model_1.fit(X_train, y_train , epochs =40, batch_size = 128, validation_data=(X_test,y_test), verbose=0)
```

Epoch 1/40  
5/5  1s 42ms/step - accuracy: 0.7413 - loss: 0.5633 - val\_accuracy: 0.7396 - val\_loss: 0.5449  
Epoch 2/40  
5/5  0s 12ms/step - accuracy: 0.7448 - loss: 0.5566 - val\_accuracy: 0.7448 - val\_loss: 0.5392  
Epoch 3/40  
5/5  0s 12ms/step - accuracy: 0.7448 - loss: 0.5508 - val\_accuracy: 0.7448 - val\_loss: 0.5338  
Epoch 4/40  
5/5  0s 12ms/step - accuracy: 0.7465 - loss: 0.5455 - val\_accuracy: 0.7448 - val\_loss: 0.5290  
Epoch 5/40  
5/5  0s 12ms/step - accuracy: 0.7483 - loss: 0.5401 - val\_accuracy: 0.7500 - val\_loss: 0.5246  
Epoch 6/40  
5/5  0s 12ms/step - accuracy: 0.7465 - loss: 0.5353 - val\_accuracy: 0.7500 - val\_loss: 0.5206  
Epoch 7/40  
5/5  0s 12ms/step - accuracy: 0.7465 - loss: 0.5306 - val\_accuracy: 0.7500 - val\_loss: 0.5170  
Epoch 8/40  
5/5  0s 12ms/step - accuracy: 0.7500 - loss: 0.5266 - val\_accuracy: 0.7448 - val\_loss: 0.5135  
Epoch 9/40  
5/5  0s 12ms/step - accuracy: 0.7465 - loss: 0.5227 - val\_accuracy: 0.7448 - val\_loss: 0.5102  
Epoch 10/40  
5/5  0s 12ms/step - accuracy: 0.7500 - loss: 0.5188 - val\_accuracy: 0.7448 - val\_loss: 0.5075  
Epoch 11/40  
5/5  0s 12ms/step - accuracy: 0.7517 - loss: 0.5153 - val\_accuracy: 0.7500 - val\_loss: 0.5048  
Epoch 12/40  
5/5  0s 12ms/step - accuracy: 0.7552 - loss: 0.5119 - val\_accuracy: 0.7500 - val\_loss: 0.5026  
Epoch 13/40  
5/5  0s 12ms/step - accuracy: 0.7552 - loss: 0.5087 - val\_accuracy: 0.7500 - val\_loss: 0.5004  
Epoch 14/40  
5/5  0s 12ms/step - accuracy: 0.7569 - loss: 0.5057 - val\_accuracy: 0.7500 - val\_loss: 0.4983  
Epoch 15/40  
5/5  0s 12ms/step - accuracy: 0.7569 - loss: 0.5029 - val\_accuracy: 0.7500 - val\_loss: 0.4963  
Epoch 16/40  
5/5  0s 12ms/step - accuracy: 0.7569 - loss: 0.5001 - val\_accuracy: 0.7448 - val\_loss: 0.4945  
Epoch 17/40  
5/5  0s 13ms/step - accuracy: 0.7569 - loss: 0.4975 - val\_accuracy: 0.7396 - val\_loss: 0.4928  
Epoch 18/40  
5/5  0s 12ms/step - accuracy: 0.7569 - loss: 0.4952 - val\_accuracy: 0.7448 - val\_loss: 0.4912  
Epoch 19/40  
5/5  0s 13ms/step - accuracy: 0.7569 - loss: 0.4930 - val\_accuracy: 0.7448 - val\_loss: 0.4897  
Epoch 20/40  
5/5  0s 12ms/step - accuracy: 0.7604 - loss: 0.4907 - val\_accuracy: 0.7500 - val\_loss: 0.4883  
Epoch 21/40  
5/5  0s 12ms/step - accuracy: 0.7622 - loss: 0.4887 - val\_accuracy: 0.7552 - val\_loss: 0.4872

```

Epoch 22/40
5/5  0s 12ms/step - accuracy: 0.7622 - loss: 0.4867 - val_accuracy: 0.7552 - val_loss: 0.4862
Epoch 23/40
5/5  0s 12ms/step - accuracy: 0.7622 - loss: 0.4849 - val_accuracy: 0.7604 - val_loss: 0.4851
Epoch 24/40
5/5  0s 12ms/step - accuracy: 0.7604 - loss: 0.4832 - val_accuracy: 0.7604 - val_loss: 0.4842
Epoch 25/40
5/5  0s 12ms/step - accuracy: 0.7604 - loss: 0.4814 - val_accuracy: 0.7604 - val_loss: 0.4834
Epoch 26/40
5/5  0s 12ms/step - accuracy: 0.7622 - loss: 0.4799 - val_accuracy: 0.7604 - val_loss: 0.4828
Epoch 27/40
5/5  0s 12ms/step - accuracy: 0.7622 - loss: 0.4782 - val_accuracy: 0.7552 - val_loss: 0.4821
Epoch 28/40
5/5  0s 12ms/step - accuracy: 0.7622 - loss: 0.4768 - val_accuracy: 0.7552 - val_loss: 0.4815
Epoch 29/40
5/5  0s 12ms/step - accuracy: 0.7622 - loss: 0.4753 - val_accuracy: 0.7552 - val_loss: 0.4810
Epoch 30/40
5/5  0s 12ms/step - accuracy: 0.7656 - loss: 0.4739 - val_accuracy: 0.7552 - val_loss: 0.4804
Epoch 31/40
5/5  0s 12ms/step - accuracy: 0.7691 - loss: 0.4725 - val_accuracy: 0.7500 - val_loss: 0.4799
Epoch 32/40
5/5  0s 12ms/step - accuracy: 0.7674 - loss: 0.4713 - val_accuracy: 0.7500 - val_loss: 0.4793
Epoch 33/40
5/5  0s 12ms/step - accuracy: 0.7674 - loss: 0.4701 - val_accuracy: 0.7552 - val_loss: 0.4789
Epoch 34/40
5/5  0s 12ms/step - accuracy: 0.7708 - loss: 0.4688 - val_accuracy: 0.7500 - val_loss: 0.4786
Epoch 35/40
5/5  0s 13ms/step - accuracy: 0.7708 - loss: 0.4677 - val_accuracy: 0.7500 - val_loss: 0.4782
Epoch 36/40
5/5  0s 12ms/step - accuracy: 0.7743 - loss: 0.4665 - val_accuracy: 0.7500 - val_loss: 0.4777
Epoch 37/40
5/5  0s 14ms/step - accuracy: 0.7760 - loss: 0.4653 - val_accuracy: 0.7500 - val_loss: 0.4773
Epoch 38/40
5/5  0s 13ms/step - accuracy: 0.7778 - loss: 0.4643 - val_accuracy: 0.7500 - val_loss: 0.4769
Epoch 39/40
5/5  0s 12ms/step - accuracy: 0.7778 - loss: 0.4632 - val_accuracy: 0.7500 - val_loss: 0.4765
Epoch 40/40
5/5  0s 12ms/step - accuracy: 0.7795 - loss: 0.4621 - val_accuracy: 0.7500 - val_loss: 0.4762

```

```
In [53]: from sklearn.metrics import roc_curve, roc_auc_score
```

```
In [54]: adam_roc = model_1.predict(X_test).ravel() #In keras we use ravel()
```

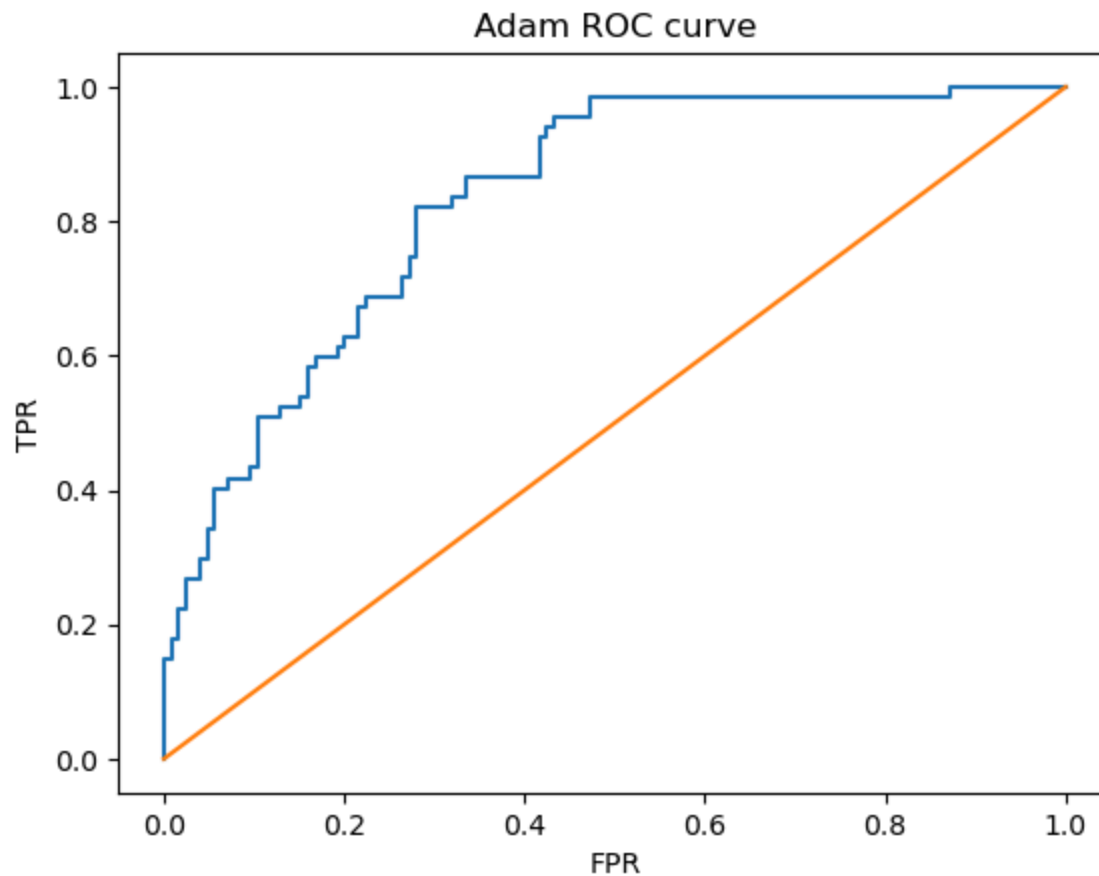
6/6 ————— 0s 3ms/step

```
In [55]: fpr, tpr, _ = roc_curve(y_test, adam_roc)
```

```
In [56]: adam_roc_auc = roc_auc_score(y_test, adam_roc)
```

```
In [57]: plt.figure()  
plt.plot(fpr, tpr)  
plt.plot([0,1],[0,1])  
plt.xlabel('FPR')  
plt.ylabel('TPR')  
plt.title('Adam ROC curve')
```

```
Out[57]: Text(0.5, 1.0, 'Adam ROC curve')
```



```
In [72]: adam_auc = roc_auc_score(y_test, adam_roc)
         print(adam_auc)
```

0.8318805970149253

```
In [58]: # Model 2 SDG(stochastic gradient descent)
         from tensorflow.keras.optimizers import SGD
         model_2 = Sequential([Input(shape=(X_train.shape[1],)), Dense(12, activation='relu'), Dense(1, activation='sigmoid')])
         model_2.compile(optimizer = SGD(learning_rate = 0.01), loss='binary_crossentropy', metrics=['accuracy'])
```

```
In [59]: sgd_model_2 = model_2.fit(X_train, y_train, epochs = 20, batch_size = 10, validation_data=(X_test, y_test), verbose = 1)
         # I tried batch_size with 128, the accuracy only hits 69%. After tuning the size to 10, about 10 % increase.
```

```

Epoch 1/20
58/58 ————— 1s 4ms/step - accuracy: 0.5521 - loss: 0.7226 - val_accuracy: 0.5781 - val_loss: 0.7020
Epoch 2/20
58/58 ————— 0s 2ms/step - accuracy: 0.6458 - loss: 0.6360 - val_accuracy: 0.6198 - val_loss: 0.6379
Epoch 3/20
58/58 ————— 0s 2ms/step - accuracy: 0.6944 - loss: 0.5842 - val_accuracy: 0.6510 - val_loss: 0.5979
Epoch 4/20
58/58 ————— 0s 2ms/step - accuracy: 0.7188 - loss: 0.5508 - val_accuracy: 0.6510 - val_loss: 0.5721
Epoch 5/20
58/58 ————— 0s 2ms/step - accuracy: 0.7344 - loss: 0.5283 - val_accuracy: 0.6406 - val_loss: 0.5548
Epoch 6/20
58/58 ————— 0s 2ms/step - accuracy: 0.7465 - loss: 0.5124 - val_accuracy: 0.6667 - val_loss: 0.5427
Epoch 7/20
58/58 ————— 0s 2ms/step - accuracy: 0.7535 - loss: 0.5002 - val_accuracy: 0.6771 - val_loss: 0.5340
Epoch 8/20
58/58 ————— 0s 2ms/step - accuracy: 0.7604 - loss: 0.4910 - val_accuracy: 0.6823 - val_loss: 0.5274
Epoch 9/20
58/58 ————— 0s 2ms/step - accuracy: 0.7674 - loss: 0.4838 - val_accuracy: 0.6927 - val_loss: 0.5225
Epoch 10/20
58/58 ————— 0s 2ms/step - accuracy: 0.7708 - loss: 0.4781 - val_accuracy: 0.6875 - val_loss: 0.5187
Epoch 11/20
58/58 ————— 0s 2ms/step - accuracy: 0.7708 - loss: 0.4734 - val_accuracy: 0.6875 - val_loss: 0.5156
Epoch 12/20
58/58 ————— 0s 2ms/step - accuracy: 0.7726 - loss: 0.4694 - val_accuracy: 0.6875 - val_loss: 0.5134
Epoch 13/20
58/58 ————— 0s 2ms/step - accuracy: 0.7691 - loss: 0.4661 - val_accuracy: 0.6927 - val_loss: 0.5113
Epoch 14/20
58/58 ————— 0s 2ms/step - accuracy: 0.7778 - loss: 0.4630 - val_accuracy: 0.7031 - val_loss: 0.5099
Epoch 15/20
58/58 ————— 0s 2ms/step - accuracy: 0.7882 - loss: 0.4603 - val_accuracy: 0.7031 - val_loss: 0.5085
Epoch 16/20
58/58 ————— 0s 2ms/step - accuracy: 0.7882 - loss: 0.4580 - val_accuracy: 0.7031 - val_loss: 0.5075
Epoch 17/20
58/58 ————— 0s 2ms/step - accuracy: 0.7917 - loss: 0.4559 - val_accuracy: 0.7083 - val_loss: 0.5067
Epoch 18/20
58/58 ————— 0s 2ms/step - accuracy: 0.7951 - loss: 0.4540 - val_accuracy: 0.7135 - val_loss: 0.5059
Epoch 19/20
58/58 ————— 0s 2ms/step - accuracy: 0.7934 - loss: 0.4523 - val_accuracy: 0.7188 - val_loss: 0.5052
Epoch 20/20
58/58 ————— 0s 2ms/step - accuracy: 0.7917 - loss: 0.4507 - val_accuracy: 0.7188 - val_loss: 0.5048

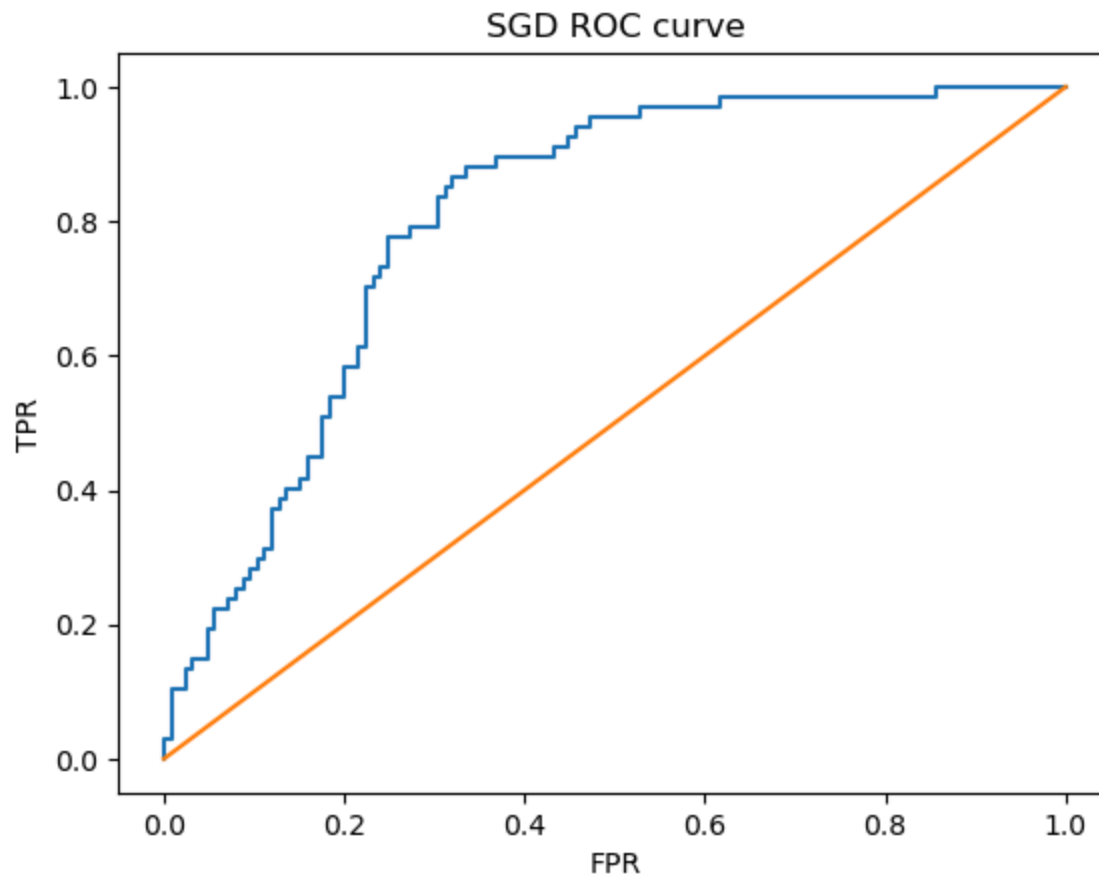
```

```
In [60]: SGD_roc = model_2.predict(X_test).ravel()
```

```
In [61]: sgd_fpr, sgd_tpr, _ = roc_curve(y_test, SGD_roc)
        SGD_roc_auc = roc_auc_score(y_test, SGD_roc)
```

```
In [62]: plt.figure()
        plt.plot(sgd_fpr, sgd_tpr)
        plt.plot([0,1],[0,1])
        plt.xlabel('FPR')
        plt.ylabel('TPR')
        plt.title('SGD ROC curve')
```

```
Out[62]: Text(0.5, 1.0, 'SGD ROC curve')
```
























```
In [73]: SGD_auc = roc_auc_score(y_test,SGD_roc)
print(SGD_auc)
```




















0.8039402985074627

```
In [63]: from tensorflow.keras.optimizers import Nadam
model_3 = Sequential([Input(shape=(X_train.shape[1],)), Dense(12, activation='relu'),Dense(1, activation='sigmoid')
model_3.compile(optimizer = Nadam(learning_rate = 0.01),loss='binary_crossentropy', metrics=['accuracy'])
```

```
In [64]: Nadam_model_3 = model_3.fit(X_train,y_train, epochs=40, batch_size = 200, validation_data=(X_test,y_test), verbose=
#epo = 20, batch_size=100 with 80% accuracy and 74% val_accuracy
#epo = 40, batch_size=128 with 82% accuracy and 72% val_accuracy
```

Epoch 1/40  
3/3  1s 79ms/step - accuracy: 0.4253 - loss: 0.7486 - val\_accuracy: 0.5208 - val\_loss: 0.7014  
Epoch 2/40  
3/3  0s 23ms/step - accuracy: 0.5781 - loss: 0.6820 - val\_accuracy: 0.6042 - val\_loss: 0.6552  
Epoch 3/40  
3/3  0s 23ms/step - accuracy: 0.6302 - loss: 0.6341 - val\_accuracy: 0.6406 - val\_loss: 0.6173  
Epoch 4/40  
3/3  0s 23ms/step - accuracy: 0.6667 - loss: 0.5959 - val\_accuracy: 0.6719 - val\_loss: 0.5883  
Epoch 5/40  
3/3  0s 23ms/step - accuracy: 0.6927 - loss: 0.5658 - val\_accuracy: 0.6771 - val\_loss: 0.5675  
Epoch 6/40  
3/3  0s 23ms/step - accuracy: 0.7170 - loss: 0.5436 - val\_accuracy: 0.6771 - val\_loss: 0.5537  
Epoch 7/40  
3/3  0s 23ms/step - accuracy: 0.7240 - loss: 0.5291 - val\_accuracy: 0.6823 - val\_loss: 0.5447  
Epoch 8/40  
3/3  0s 22ms/step - accuracy: 0.7344 - loss: 0.5164 - val\_accuracy: 0.6979 - val\_loss: 0.5390  
Epoch 9/40  
3/3  0s 23ms/step - accuracy: 0.7361 - loss: 0.5056 - val\_accuracy: 0.6875 - val\_loss: 0.5355  
Epoch 10/40  
3/3  0s 22ms/step - accuracy: 0.7465 - loss: 0.4969 - val\_accuracy: 0.7031 - val\_loss: 0.5334  
Epoch 11/40  
3/3  0s 23ms/step - accuracy: 0.7535 - loss: 0.4891 - val\_accuracy: 0.7083 - val\_loss: 0.5316  
Epoch 12/40  
3/3  0s 23ms/step - accuracy: 0.7639 - loss: 0.4815 - val\_accuracy: 0.7135 - val\_loss: 0.5297  
Epoch 13/40  
3/3  0s 23ms/step - accuracy: 0.7708 - loss: 0.4753 - val\_accuracy: 0.7083 - val\_loss: 0.5279  
Epoch 14/40  
3/3  0s 25ms/step - accuracy: 0.7708 - loss: 0.4687 - val\_accuracy: 0.7135 - val\_loss: 0.5260  
Epoch 15/40  
3/3  0s 23ms/step - accuracy: 0.7812 - loss: 0.4624 - val\_accuracy: 0.7188 - val\_loss: 0.5238  
Epoch 16/40  
3/3  0s 22ms/step - accuracy: 0.7830 - loss: 0.4572 - val\_accuracy: 0.7188 - val\_loss: 0.5215  
Epoch 17/40  
3/3  0s 23ms/step - accuracy: 0.7865 - loss: 0.4527 - val\_accuracy: 0.7188 - val\_loss: 0.5196  
Epoch 18/40  
3/3  0s 23ms/step - accuracy: 0.7830 - loss: 0.4483 - val\_accuracy: 0.7135 - val\_loss: 0.5178  
Epoch 19/40  
3/3  0s 23ms/step - accuracy: 0.7812 - loss: 0.4442 - val\_accuracy: 0.7135 - val\_loss: 0.5160  
Epoch 20/40  
3/3  0s 22ms/step - accuracy: 0.7865 - loss: 0.4408 - val\_accuracy: 0.7188 - val\_loss: 0.5143  
Epoch 21/40  
3/3  0s 23ms/step - accuracy: 0.7865 - loss: 0.4376 - val\_accuracy: 0.7083 - val\_loss: 0.5131

```

Epoch 22/40
3/3  0s 23ms/step - accuracy: 0.7899 - loss: 0.4345 - val_accuracy: 0.7135 - val_loss: 0.5119
Epoch 23/40
3/3  0s 23ms/step - accuracy: 0.7899 - loss: 0.4317 - val_accuracy: 0.7135 - val_loss: 0.5114
Epoch 24/40
3/3  0s 23ms/step - accuracy: 0.7899 - loss: 0.4292 - val_accuracy: 0.7188 - val_loss: 0.5112
Epoch 25/40
3/3  0s 23ms/step - accuracy: 0.7899 - loss: 0.4272 - val_accuracy: 0.7188 - val_loss: 0.5111
Epoch 26/40
3/3  0s 23ms/step - accuracy: 0.7899 - loss: 0.4253 - val_accuracy: 0.7240 - val_loss: 0.5114
Epoch 27/40
3/3  0s 23ms/step - accuracy: 0.7934 - loss: 0.4237 - val_accuracy: 0.7240 - val_loss: 0.5119
Epoch 28/40
3/3  0s 22ms/step - accuracy: 0.7951 - loss: 0.4222 - val_accuracy: 0.7292 - val_loss: 0.5121
Epoch 29/40
3/3  0s 23ms/step - accuracy: 0.7951 - loss: 0.4203 - val_accuracy: 0.7240 - val_loss: 0.5119
Epoch 30/40
3/3  0s 28ms/step - accuracy: 0.7934 - loss: 0.4195 - val_accuracy: 0.7292 - val_loss: 0.5115
Epoch 31/40
3/3  0s 23ms/step - accuracy: 0.7986 - loss: 0.4181 - val_accuracy: 0.7396 - val_loss: 0.5115
Epoch 32/40
3/3  0s 23ms/step - accuracy: 0.8003 - loss: 0.4167 - val_accuracy: 0.7396 - val_loss: 0.5113
Epoch 33/40
3/3  0s 23ms/step - accuracy: 0.8021 - loss: 0.4155 - val_accuracy: 0.7344 - val_loss: 0.5109
Epoch 34/40
3/3  0s 24ms/step - accuracy: 0.8021 - loss: 0.4144 - val_accuracy: 0.7344 - val_loss: 0.5108
Epoch 35/40
3/3  0s 23ms/step - accuracy: 0.7986 - loss: 0.4136 - val_accuracy: 0.7344 - val_loss: 0.5104
Epoch 36/40
3/3  0s 23ms/step - accuracy: 0.8021 - loss: 0.4124 - val_accuracy: 0.7396 - val_loss: 0.5111
Epoch 37/40
3/3  0s 22ms/step - accuracy: 0.8038 - loss: 0.4114 - val_accuracy: 0.7188 - val_loss: 0.5114
Epoch 38/40
3/3  0s 23ms/step - accuracy: 0.8038 - loss: 0.4106 - val_accuracy: 0.7188 - val_loss: 0.5122
Epoch 39/40
3/3  0s 23ms/step - accuracy: 0.8038 - loss: 0.4096 - val_accuracy: 0.7135 - val_loss: 0.5129
Epoch 40/40
3/3  0s 23ms/step - accuracy: 0.8038 - loss: 0.4091 - val_accuracy: 0.7135 - val_loss: 0.5135

```

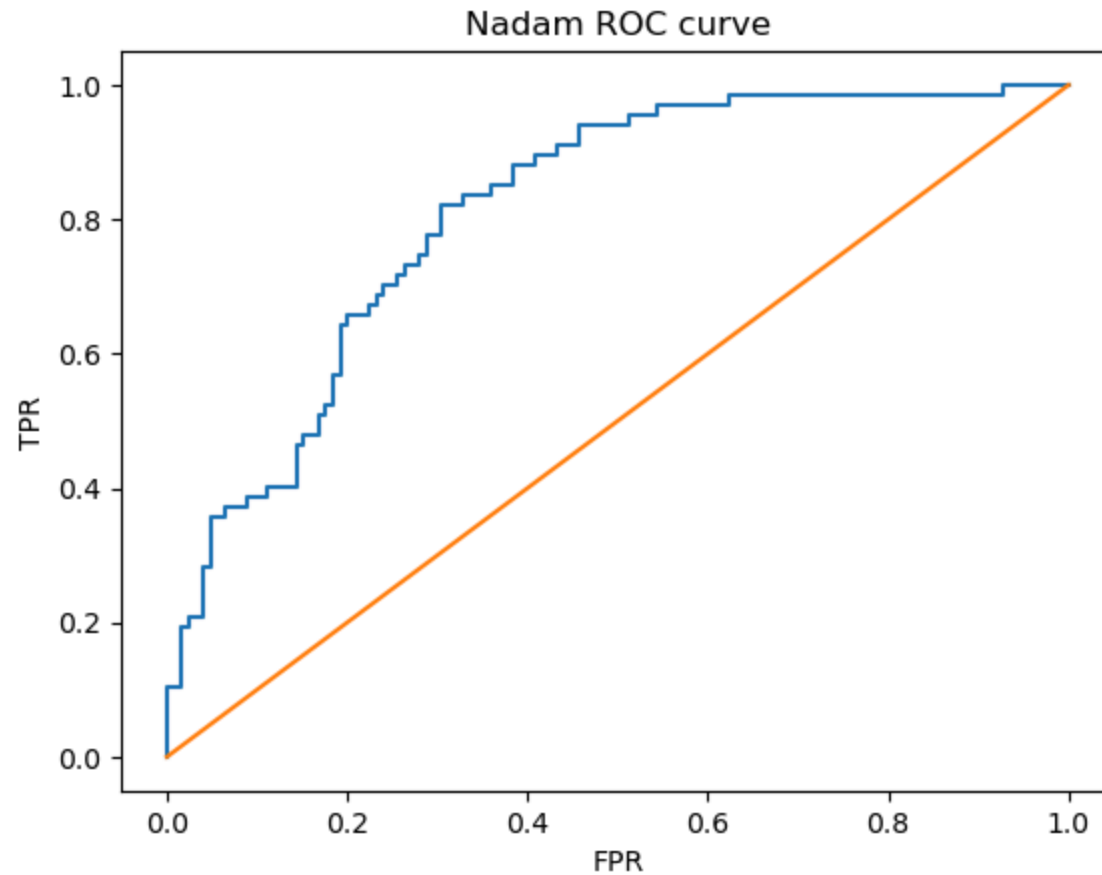
```

In [65]: Nadam_roc = model_3.predict(X_test).ravel()
         Nadam_fpr, Nadam_tpr, _ = roc_curve(y_test, Nadam_roc)
         Nadam_roc_auc = roc_auc_score(y_test, Nadam_roc)

```

```
In [66]: plt.figure()  
plt.plot(Nadam_fpr,Nadam_tpr)  
plt.plot([0,1],[0,1])  
plt.xlabel('FPR')  
plt.ylabel('TPR')  
plt.title('Nadam ROC curve')
```

Out[66]: Text(0.5, 1.0, 'Nadam ROC curve')
























```
In [74]: Nadam_auc = roc_auc_score(y_test,Nadam_roc)  
print(Nadam_auc)
```

0.8122985074626866

```
In [67]: from tensorflow.keras.optimizers import Ftrl
model_4 = Sequential([Input(shape=(X_train.shape[1],)), Dense(12, activation='relu'), Dense(1, activation='sigmoid')])
model_4.compile(optimizer = Ftrl(learning_rate = 0.01), loss='binary_crossentropy', metrics=['accuracy'])
```

```
In [68]: Ftrl_model_4 = model_4.fit(X_train, y_train, epochs=40, batch_size=5, validation_data=(X_test, y_test), verbose=1)
#epo = 10, batch_size=128 with 65% accuracy and 65% val_accuracy
#epo = 40, batch_size=5 with 79% accuracy and 73% val_accuracy
```

Epoch 1/40  
116/116  1s 3ms/step - accuracy: 0.6493 - loss: 0.6787 - val\_accuracy: 0.6510 - val\_loss: 0.6655  
Epoch 2/40  
116/116  0s 2ms/step - accuracy: 0.6510 - loss: 0.6496 - val\_accuracy: 0.6510 - val\_loss: 0.6341  
Epoch 3/40  
116/116  0s 2ms/step - accuracy: 0.6510 - loss: 0.6138 - val\_accuracy: 0.6510 - val\_loss: 0.6003  
Epoch 4/40  
116/116  0s 2ms/step - accuracy: 0.6632 - loss: 0.5797 - val\_accuracy: 0.6875 - val\_loss: 0.5711  
Epoch 5/40  
116/116  0s 2ms/step - accuracy: 0.7344 - loss: 0.5503 - val\_accuracy: 0.7344 - val\_loss: 0.5492  
Epoch 6/40  
116/116  0s 2ms/step - accuracy: 0.7812 - loss: 0.5272 - val\_accuracy: 0.7188 - val\_loss: 0.5337  
Epoch 7/40  
116/116  0s 2ms/step - accuracy: 0.7812 - loss: 0.5095 - val\_accuracy: 0.7188 - val\_loss: 0.5233  
Epoch 8/40  
116/116  0s 2ms/step - accuracy: 0.7795 - loss: 0.4961 - val\_accuracy: 0.7240 - val\_loss: 0.5169  
Epoch 9/40  
116/116  0s 2ms/step - accuracy: 0.7743 - loss: 0.4862 - val\_accuracy: 0.7292 - val\_loss: 0.5128  
Epoch 10/40  
116/116  0s 2ms/step - accuracy: 0.7726 - loss: 0.4790 - val\_accuracy: 0.7240 - val\_loss: 0.5102  
Epoch 11/40  
116/116  0s 2ms/step - accuracy: 0.7778 - loss: 0.4732 - val\_accuracy: 0.7292 - val\_loss: 0.5087  
Epoch 12/40  
116/116  0s 2ms/step - accuracy: 0.7743 - loss: 0.4687 - val\_accuracy: 0.7292 - val\_loss: 0.5079  
Epoch 13/40  
116/116  0s 2ms/step - accuracy: 0.7795 - loss: 0.4653 - val\_accuracy: 0.7344 - val\_loss: 0.5075  
Epoch 14/40  
116/116  0s 2ms/step - accuracy: 0.7795 - loss: 0.4625 - val\_accuracy: 0.7344 - val\_loss: 0.5072  
Epoch 15/40  
116/116  0s 2ms/step - accuracy: 0.7812 - loss: 0.4601 - val\_accuracy: 0.7344 - val\_loss: 0.5072  
Epoch 16/40  
116/116  0s 2ms/step - accuracy: 0.7865 - loss: 0.4582 - val\_accuracy: 0.7344 - val\_loss: 0.5072  
Epoch 17/40  
116/116  0s 2ms/step - accuracy: 0.7830 - loss: 0.4565 - val\_accuracy: 0.7344 - val\_loss: 0.5073  
Epoch 18/40  
116/116  0s 2ms/step - accuracy: 0.7847 - loss: 0.4551 - val\_accuracy: 0.7344 - val\_loss: 0.5073  
Epoch 19/40  
116/116  0s 2ms/step - accuracy: 0.7812 - loss: 0.4540 - val\_accuracy: 0.7344 - val\_loss: 0.5074  
Epoch 20/40  
116/116  0s 2ms/step - accuracy: 0.7795 - loss: 0.4528 - val\_accuracy: 0.7344 - val\_loss: 0.5075  
Epoch 21/40  
116/116  0s 2ms/step - accuracy: 0.7830 - loss: 0.4518 - val\_accuracy: 0.7344 - val\_loss: 0.5076

```

Epoch 22/40
116/116 ————— 0s 2ms/step - accuracy: 0.7882 - loss: 0.4509 - val_accuracy: 0.7396 - val_loss: 0.5077
Epoch 23/40
116/116 ————— 0s 2ms/step - accuracy: 0.7865 - loss: 0.4501 - val_accuracy: 0.7292 - val_loss: 0.5078
Epoch 24/40
116/116 ————— 0s 2ms/step - accuracy: 0.7847 - loss: 0.4494 - val_accuracy: 0.7292 - val_loss: 0.5075
Epoch 25/40
116/116 ————— 0s 2ms/step - accuracy: 0.7899 - loss: 0.4488 - val_accuracy: 0.7292 - val_loss: 0.5075
Epoch 26/40
116/116 ————— 0s 2ms/step - accuracy: 0.7882 - loss: 0.4482 - val_accuracy: 0.7292 - val_loss: 0.5076
Epoch 27/40
116/116 ————— 0s 2ms/step - accuracy: 0.7830 - loss: 0.4476 - val_accuracy: 0.7292 - val_loss: 0.5077
Epoch 28/40
116/116 ————— 0s 2ms/step - accuracy: 0.7847 - loss: 0.4470 - val_accuracy: 0.7292 - val_loss: 0.5078
Epoch 29/40
116/116 ————— 0s 2ms/step - accuracy: 0.7865 - loss: 0.4465 - val_accuracy: 0.7292 - val_loss: 0.5078
Epoch 30/40
116/116 ————— 0s 2ms/step - accuracy: 0.7882 - loss: 0.4460 - val_accuracy: 0.7344 - val_loss: 0.5080
Epoch 31/40
116/116 ————— 0s 2ms/step - accuracy: 0.7882 - loss: 0.4455 - val_accuracy: 0.7344 - val_loss: 0.5081
Epoch 32/40
116/116 ————— 0s 2ms/step - accuracy: 0.7899 - loss: 0.4450 - val_accuracy: 0.7292 - val_loss: 0.5079
Epoch 33/40
116/116 ————— 0s 2ms/step - accuracy: 0.7865 - loss: 0.4446 - val_accuracy: 0.7292 - val_loss: 0.5080
Epoch 34/40
116/116 ————— 0s 2ms/step - accuracy: 0.7882 - loss: 0.4442 - val_accuracy: 0.7292 - val_loss: 0.5082
Epoch 35/40
116/116 ————— 0s 2ms/step - accuracy: 0.7899 - loss: 0.4438 - val_accuracy: 0.7292 - val_loss: 0.5082
Epoch 36/40
116/116 ————— 0s 2ms/step - accuracy: 0.7917 - loss: 0.4434 - val_accuracy: 0.7292 - val_loss: 0.5083
Epoch 37/40
116/116 ————— 0s 2ms/step - accuracy: 0.7917 - loss: 0.4431 - val_accuracy: 0.7344 - val_loss: 0.5082
Epoch 38/40
116/116 ————— 0s 2ms/step - accuracy: 0.7899 - loss: 0.4428 - val_accuracy: 0.7292 - val_loss: 0.5083
Epoch 39/40
116/116 ————— 0s 2ms/step - accuracy: 0.7882 - loss: 0.4425 - val_accuracy: 0.7292 - val_loss: 0.5084
Epoch 40/40
116/116 ————— 0s 2ms/step - accuracy: 0.7917 - loss: 0.4422 - val_accuracy: 0.7292 - val_loss: 0.5085

```

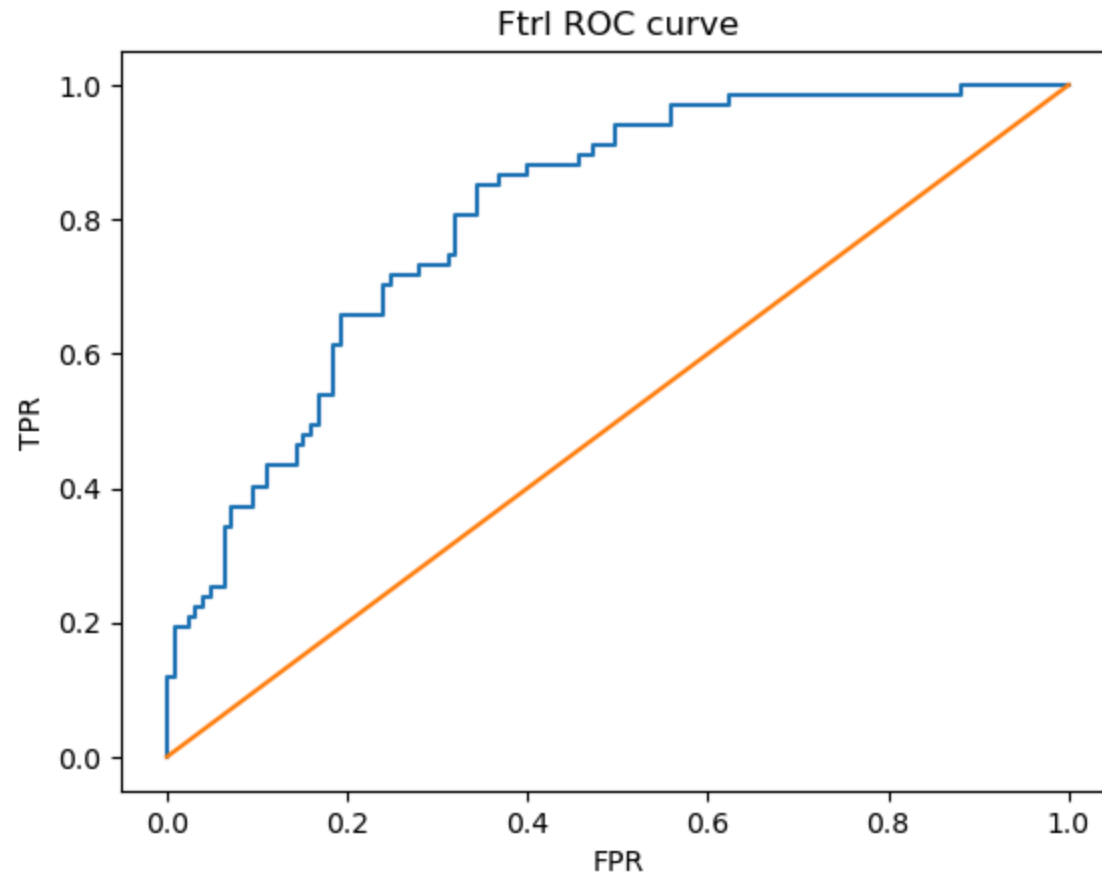
```

In [69]: Ftrl_roc = model_4.predict(X_test).ravel()
          Ftrl_fpr, Ftrl_tpr, _ = roc_curve(y_test, Ftrl_roc)
          Ftrl_roc_auc = roc_auc_score(y_test, Ftrl_roc)

```

```
In [70]: plt.figure()  
plt.plot(Ftrl_fpr, Ftrl_tpr)  
plt.plot([0,1],[0,1])  
plt.xlabel('FPR')  
plt.ylabel('TPR')  
plt.title('Ftrl ROC curve')
```

Out[70]: Text(0.5, 1.0, 'Ftrl ROC curve')



```
In [75]: Ftrl_auc = roc_auc_score(y_test, Ftrl_roc)  
print(Ftrl_auc)
```

0.8072835820895522

```
In [ ]: #Which has the best performance and why?  
#Among all the models, Adam are the best with highest AUC(0.832).  
#Which means Adam optimized can learn from the unseen data and distinguish between diabetic and non diabetic patterns
```

```
In [81]: winner_model = model_2  
winner_model.save_weights("best_model.weights.h5")
```