Azkya Said

Machine Learning for Diabetes Prediction

Objective:

This project focuses on supervised binary classification to develop and evaluate machine learning models for diabetes prediction from patient data. We employ Random Forest, Support Vector Machine, and LSTM Neural Network algorithms, utilizing K-folds cross-validation and the area under the ROC curve (AUC) for performance assessment.

Dataset:

Diabetes dataset with 8 predictive variables from: https://github.com/plotly/datasets/blob/master/diabetes.csv

The csv file is also provided in the zip file that this report was from.

Environment Setup:

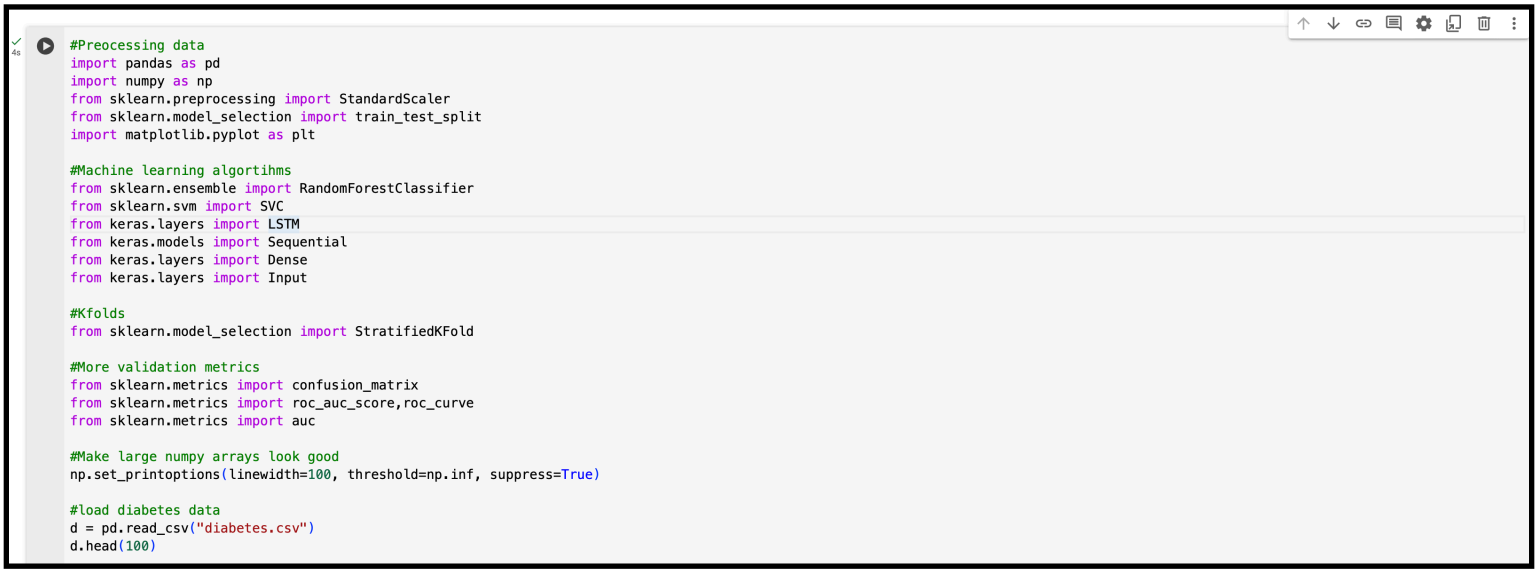
Prerequisites
Operating System: modern version of Windows, macOS, or Linux with at least 8 GB RAM

1. Install Anaconda Distributer
2. Visit the Anaconda website: https://www.anaconda.com/products/distribution
3. Choose the appropriate installer for your operating system and download it
4. Create a Virtual Environment (Highly Recommended)
   1. Why Virtual Environments: They isolate project dependencies, preventing conflicts with other Python projects on your system.
   2. Create Environment:
      - Open a terminal or command prompt
      - Navigate to your desired project directory
      - Create the environment: conda create -n diabetes-project python=3.9
      - Activate the environment: conda/source activate diabetes-project
      - Download dependencies: conda install -c conda-forge scikit-learn matplotlib numpy pandas keras tensorflow (you can use pip install as well)
      - In your activated environment, type: jupyter notebook
      -

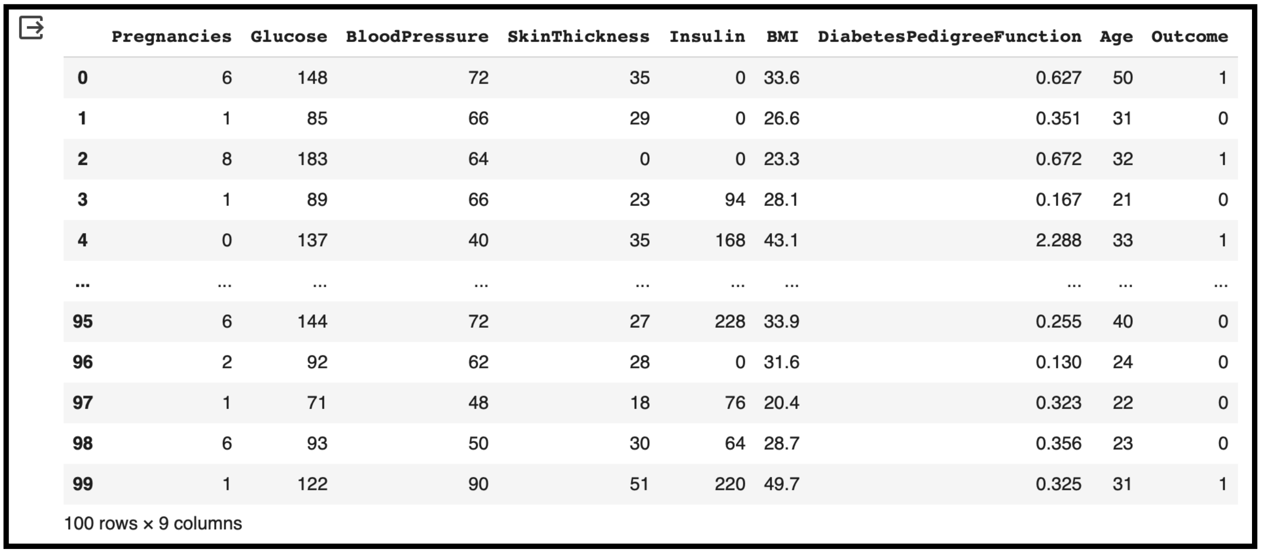
Lets get started!

Azkya Said

Importing dependencies and loading data:



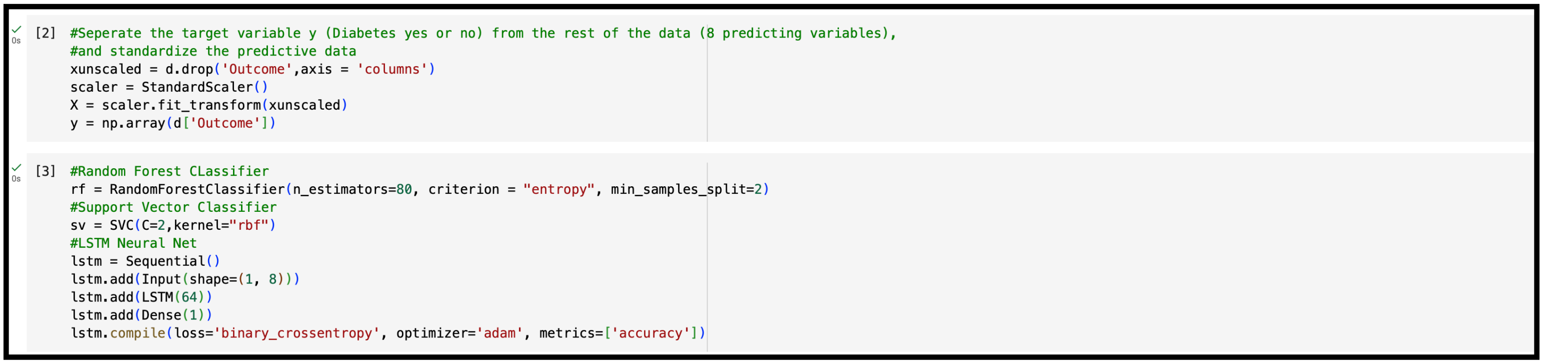
```
#Preocessing data
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

#Machine learning algortihms
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from keras.layers import LSTM
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Input

#Kfolds
from sklearn.model_selection import StratifiedKFold

#More validation metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import auc

#Make large numpy arrays look good
np.set_printoptions(linewidth=100, threshold=np.inf, suppress=True)

#load diabetes data
d = pd.read_csv("diabetes.csv")
d.head(100)
```



	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
...	...	...	...	...	...	...	...	...	...
95	6	144	72	27	228	33.9	0.255	40	0
96	2	92	62	28	0	31.6	0.130	24	0
97	1	71	48	18	76	20.4	0.323	22	0
98	6	93	50	30	64	28.7	0.356	23	0
99	1	122	90	51	220	49.7	0.325	31	1

100 rows × 9 columns

Preparing the data and setting up the models:



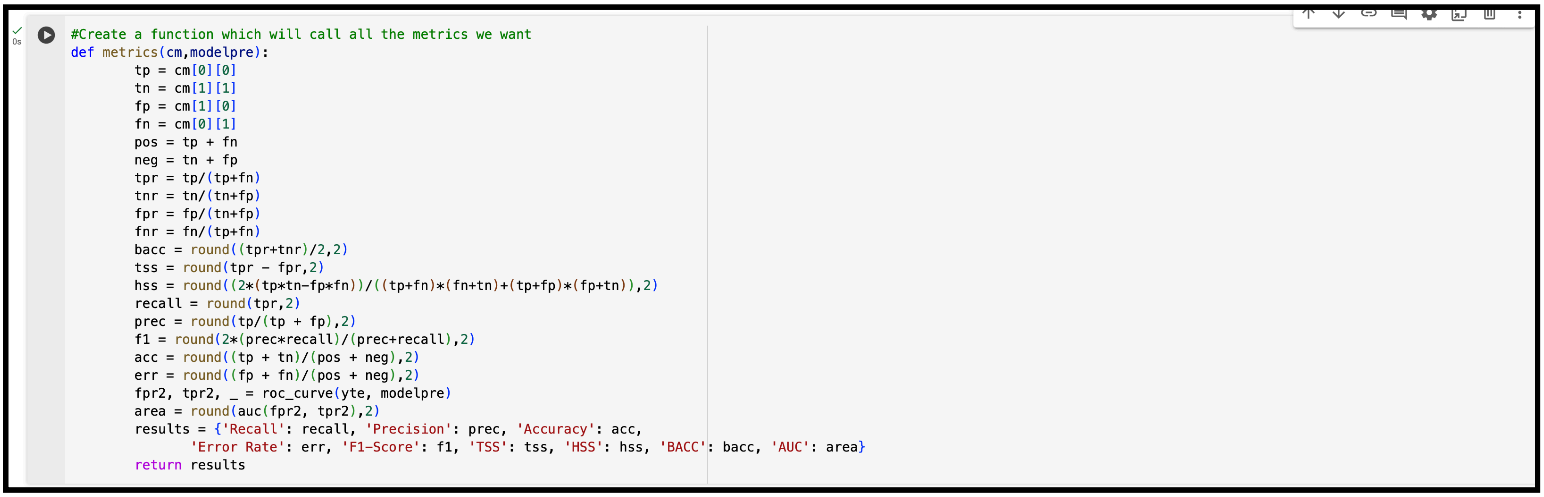
[2]

```
#Seperate the target variable y (Diabetes yes or no) from the rest of the data (8 predicting variables),
#and standardize the predictive data
xunscaled = d.drop('Outcome',axis = 'columns')
scaler = StandardScaler()
X = scaler.fit_transform(xunscaled)
y = np.array(d['Outcome'])
```

[3]

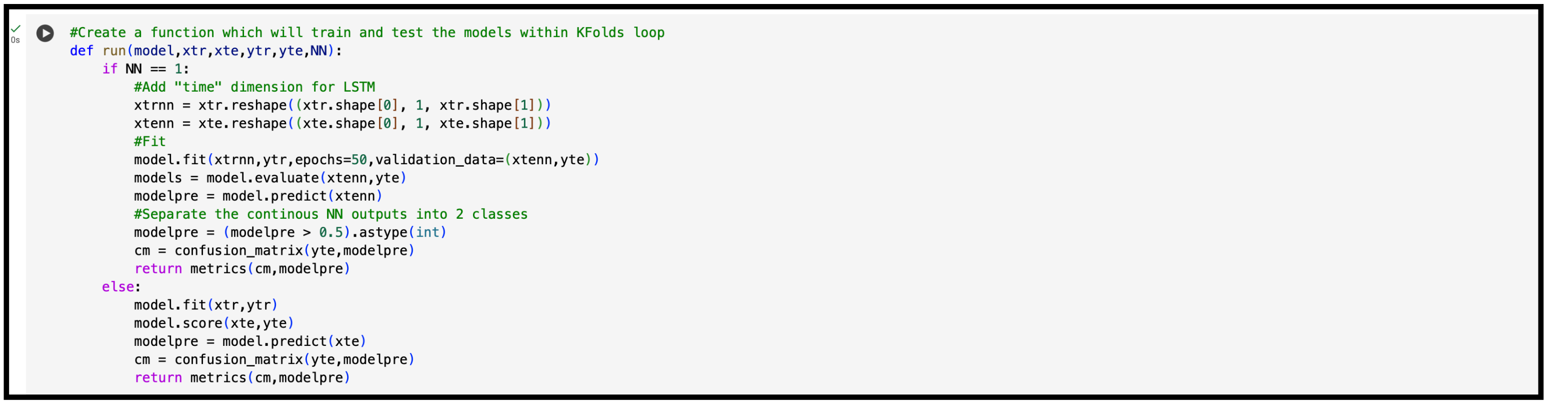
```
#Random Forest CLassifier
rf = RandomForestClassifier(n_estimators=80, criterion = "entropy", min_samples_split=2)
#Support Vector Classifier
sv = SVC(C=2,kernel="rbf")
#LSTM Neural Net
lstm = Sequential()
lstm.add(Input(shape=(1, 8)))
lstm.add(LSTM(64))
lstm.add(Dense(1))
lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Creating a metrics function which will calculate all of our performance metrics:



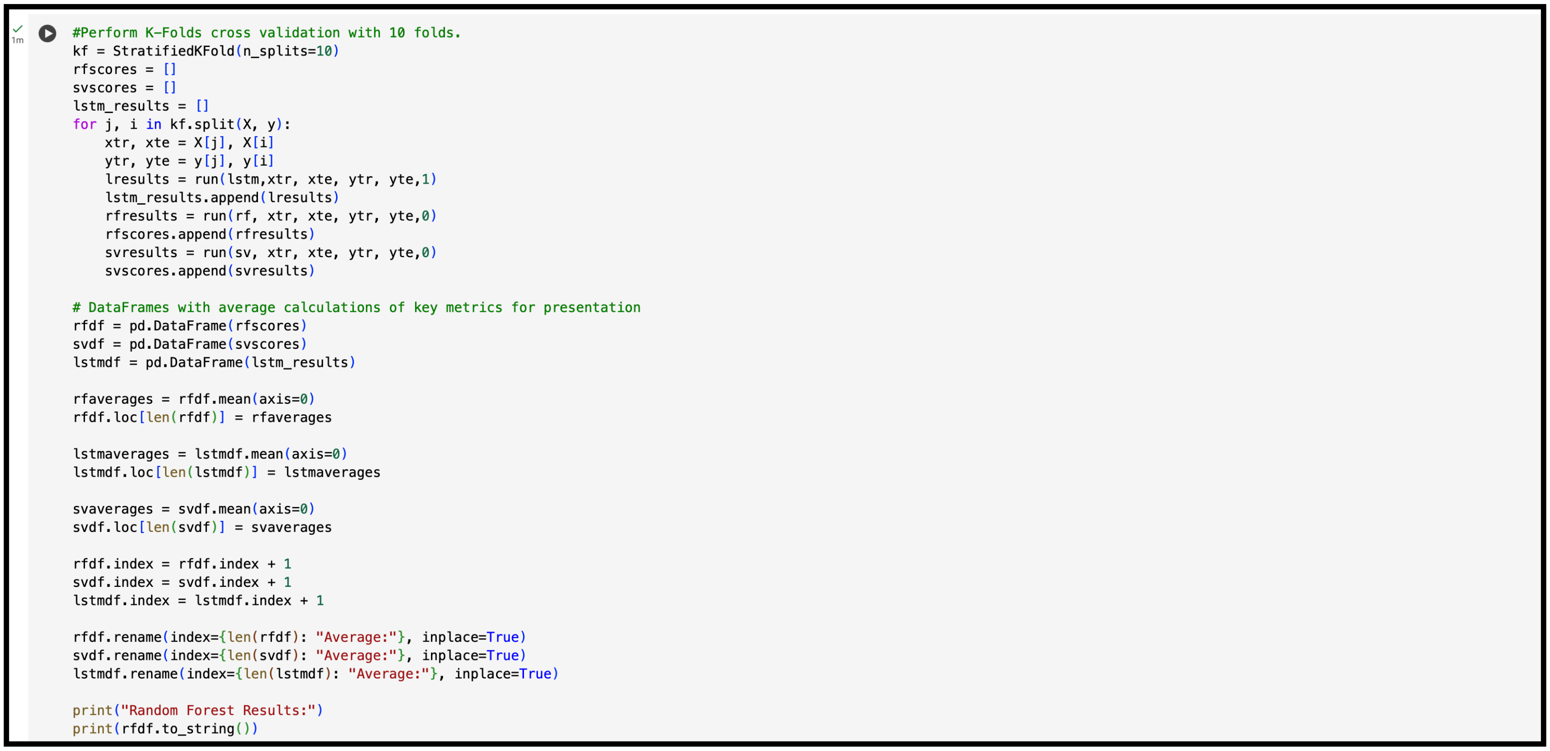
```
#Create a function which will call all the metrics we want
def metrics(cm,modelpre):
        tp = cm[0][0]
        tn = cm[1][1]
        fp = cm[1][0]
        fn = cm[0][1]
        pos = tp + fn
        neg = tn + fp
        tpr = tp/(tp+fn)
        tnr = tn/(tn+fp)
        fpr = fp/(tn+fp)
        fnr = fn/(tp+fn)
        bacc = round((tpr+tnr)/2,2)
        tss = round(tpr - fpr,2)
        hss = round((2*(tp*tn-fp*fn))/((tp+fn)*(fn+tn)+(tp+fp)*(fp+tn)),2)
        recall = round(tpr,2)
        prec = round(tp/(tp + fp),2)
        f1 = round(2*(prec*recall)/(prec+recall),2)
        acc = round((tp + tn)/(pos + neg),2)
        err = round((fp + fn)/(pos + neg),2)
        fpr2, tpr2, _ = roc_curve(yte, modelpre)
        area = round(auc(fpr2, tpr2),2)
        results = {'Recall': recall, 'Precision': prec, 'Accuracy': acc,
               'Error Rate': err, 'F1-Score': f1, 'TSS': tss, 'HSS': hss, 'BACC': bacc, 'AUC': area}
        return results
```

Create a function to train, test the models, and output the metrics:

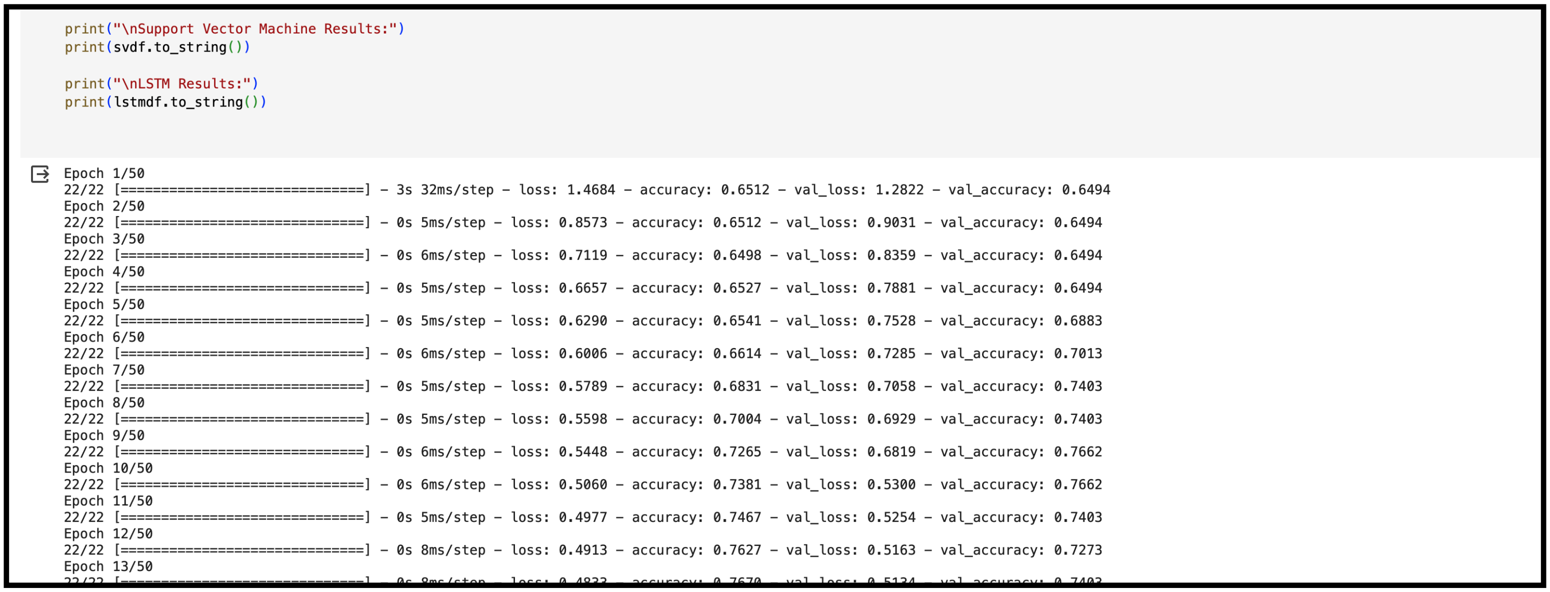


```
#Create a function which will train and test the models within KFolds loop
def run(model,xtr,xte,ytr,yte,NN):
    if NN == 1:
        #Add "time" dimension for LSTM
        xtrnn = xtr.reshape((xtr.shape[0], 1, xtr.shape[1]))
        xtenn = xte.reshape((xte.shape[0], 1, xte.shape[1]))
        #Fit
        model.fit(xtrnn,ytr,epochs=50,validation_data=(xtenn,yte))
        models = model.evaluate(xtenn,yte)
        modelpre = model.predict(xtenn)
        #Separate the continous NN outputs into 2 classes
        modelpre = (modelpre > 0.5).astype(int)
        cm = confusion_matrix(yte,modelpre)
        return metrics(cm,modelpre)
    else:
        model.fit(xtr,ytr)
        model.score(xte,yte)
        modelpre = model.predict(xte)
        cm = confusion_matrix(yte,modelpre)
        return metrics(cm,modelpre)
```

Now run K-Folds cross validation on all 3 models!

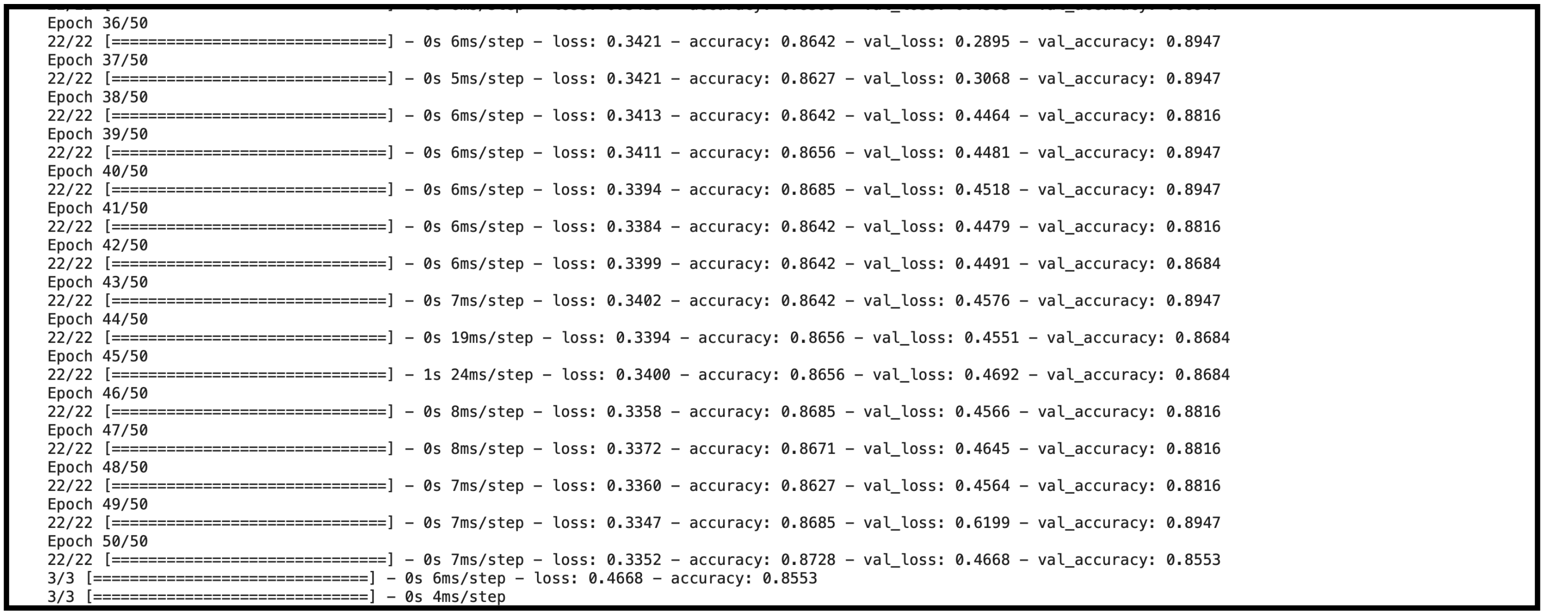


```
#Perform K-Folds cross validation with 10 folds.
kf = StratifiedKFold(n_splits=10)
rfscores = []
svscores = []
lstm_results = []
for j, i in kf.split(X, y):
    xtr, xte = X[j], X[i]
    ytr, yte = y[j], y[i]
    lresults = run(lstm,xtr, xte, ytr, yte,1)
    lstm_results.append(lresults)
    rfresults = run(rf, xtr, xte, ytr, yte,0)
    rfscores.append(rfresults)
    svresults = run(sv, xtr, xte, ytr, yte,0)
    svscores.append(svresults)

# DataFrames with average calculations of key metrics for presentation
rfdf = pd.DataFrame(rfscores)
svdf = pd.DataFrame(svscores)
lstmdf = pd.DataFrame(lstm_results)

rfaverages = rfdf.mean(axis=0)
rfdf.loc[len(rfdf)] = rfaverages

lstmaverages = lstmdf.mean(axis=0)
lstmdf.loc[len(lstmdf)] = lstmaverages

svaverages = svdf.mean(axis=0)
svdf.loc[len(svdf)] = svaverages

rfdf.index = rfdf.index + 1
svdf.index = svdf.index + 1
lstmdf.index = lstmdf.index + 1

rfdf.rename(index={len(rfdf): "Average:"}, inplace=True)
svdf.rename(index={len(svdf): "Average:"}, inplace=True)
lstmdf.rename(index={len(lstmdf): "Average:"}, inplace=True)

print("Random Forest Results:")
print(rfdf.to_string())
```



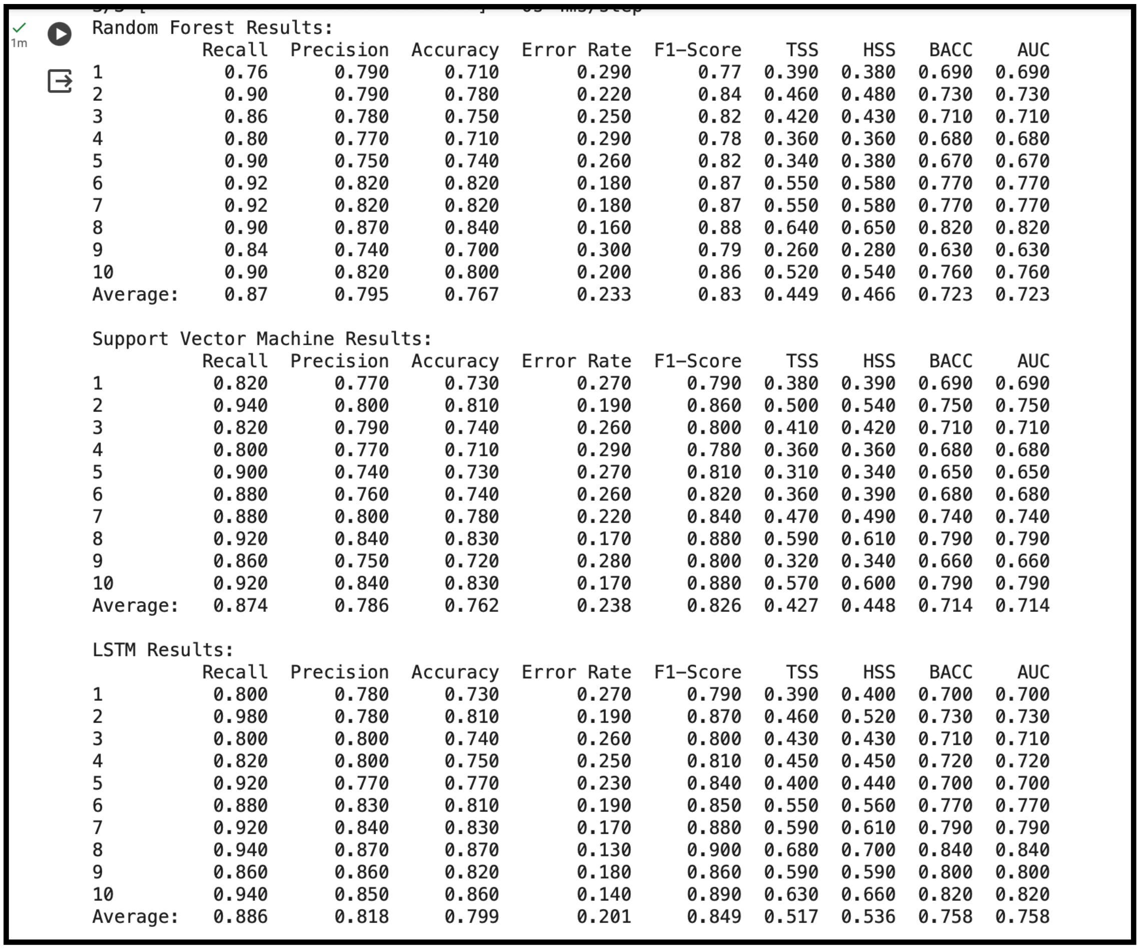
```
print("\nSupport Vector Machine Results:")
print(svdf.to_string())

print("\nLSTM Results:")
print(lstmdf.to_string())
```

```
Epoch 1/50
22/22 [==============================] - 3s 32ms/step - loss: 1.4684 - accuracy: 0.6512 - val_loss: 1.2822 - val_accuracy: 0.6494
Epoch 2/50
22/22 [==============================] - 0s 5ms/step - loss: 0.8573 - accuracy: 0.6512 - val_loss: 0.9031 - val_accuracy: 0.6494
Epoch 3/50
22/22 [==============================] - 0s 6ms/step - loss: 0.7119 - accuracy: 0.6498 - val_loss: 0.8359 - val_accuracy: 0.6494
Epoch 4/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6657 - accuracy: 0.6527 - val_loss: 0.7881 - val_accuracy: 0.6494
Epoch 5/50
22/22 [==============================] - 0s 5ms/step - loss: 0.6290 - accuracy: 0.6541 - val_loss: 0.7528 - val_accuracy: 0.6883
Epoch 6/50
22/22 [==============================] - 0s 6ms/step - loss: 0.6006 - accuracy: 0.6614 - val_loss: 0.7285 - val_accuracy: 0.7013
Epoch 7/50
22/22 [==============================] - 0s 5ms/step - loss: 0.5789 - accuracy: 0.6831 - val_loss: 0.7058 - val_accuracy: 0.7403
Epoch 8/50
22/22 [==============================] - 0s 5ms/step - loss: 0.5598 - accuracy: 0.7004 - val_loss: 0.6929 - val_accuracy: 0.7403
Epoch 9/50
22/22 [==============================] - 0s 6ms/step - loss: 0.5448 - accuracy: 0.7265 - val_loss: 0.6819 - val_accuracy: 0.7662
Epoch 10/50
22/22 [==============================] - 0s 6ms/step - loss: 0.5060 - accuracy: 0.7381 - val_loss: 0.5300 - val_accuracy: 0.7662
Epoch 11/50
22/22 [==============================] - 0s 5ms/step - loss: 0.4977 - accuracy: 0.7467 - val_loss: 0.5254 - val_accuracy: 0.7403
Epoch 12/50
22/22 [==============================] - 0s 8ms/step - loss: 0.4913 - accuracy: 0.7627 - val_loss: 0.5163 - val_accuracy: 0.7273
Epoch 13/50
22/22 [==============================] - 0s 8ms/step - loss: 0.4833 - accuracy: 0.7670 - val_loss: 0.5134 - val_accuracy: 0.7403
```



```
Epoch 36/50
22/22 [==============================] - 0s 6ms/step - loss: 0.3421 - accuracy: 0.8642 - val_loss: 0.2895 - val_accuracy: 0.8947
Epoch 37/50
22/22 [==============================] - 0s 5ms/step - loss: 0.3421 - accuracy: 0.8627 - val_loss: 0.3068 - val_accuracy: 0.8947
Epoch 38/50
22/22 [==============================] - 0s 6ms/step - loss: 0.3413 - accuracy: 0.8642 - val_loss: 0.4464 - val_accuracy: 0.8816
Epoch 39/50
22/22 [==============================] - 0s 6ms/step - loss: 0.3411 - accuracy: 0.8656 - val_loss: 0.4481 - val_accuracy: 0.8947
Epoch 40/50
22/22 [==============================] - 0s 6ms/step - loss: 0.3394 - accuracy: 0.8685 - val_loss: 0.4518 - val_accuracy: 0.8947
Epoch 41/50
22/22 [==============================] - 0s 6ms/step - loss: 0.3384 - accuracy: 0.8642 - val_loss: 0.4479 - val_accuracy: 0.8816
Epoch 42/50
22/22 [==============================] - 0s 6ms/step - loss: 0.3399 - accuracy: 0.8642 - val_loss: 0.4491 - val_accuracy: 0.8684
Epoch 43/50
22/22 [==============================] - 0s 7ms/step - loss: 0.3402 - accuracy: 0.8642 - val_loss: 0.4576 - val_accuracy: 0.8947
Epoch 44/50
22/22 [==============================] - 0s 19ms/step - loss: 0.3394 - accuracy: 0.8656 - val_loss: 0.4551 - val_accuracy: 0.8684
Epoch 45/50
22/22 [==============================] - 1s 24ms/step - loss: 0.3400 - accuracy: 0.8656 - val_loss: 0.4692 - val_accuracy: 0.8684
Epoch 46/50
22/22 [==============================] - 0s 8ms/step - loss: 0.3358 - accuracy: 0.8685 - val_loss: 0.4566 - val_accuracy: 0.8816
Epoch 47/50
22/22 [==============================] - 0s 8ms/step - loss: 0.3372 - accuracy: 0.8671 - val_loss: 0.4645 - val_accuracy: 0.8816
Epoch 48/50
22/22 [==============================] - 0s 7ms/step - loss: 0.3360 - accuracy: 0.8627 - val_loss: 0.4564 - val_accuracy: 0.8816
Epoch 49/50
22/22 [==============================] - 0s 7ms/step - loss: 0.3347 - accuracy: 0.8685 - val_loss: 0.6199 - val_accuracy: 0.8947
Epoch 50/50
22/22 [==============================] - 0s 7ms/step - loss: 0.3352 - accuracy: 0.8728 - val_loss: 0.4668 - val_accuracy: 0.8553
3/3 [==============================] - 0s 6ms/step - loss: 0.4668 - accuracy: 0.8553
3/3 [==============================] - 0s 4ms/step
```

50 epochs (for LSTM) * 10 folds later, the results:

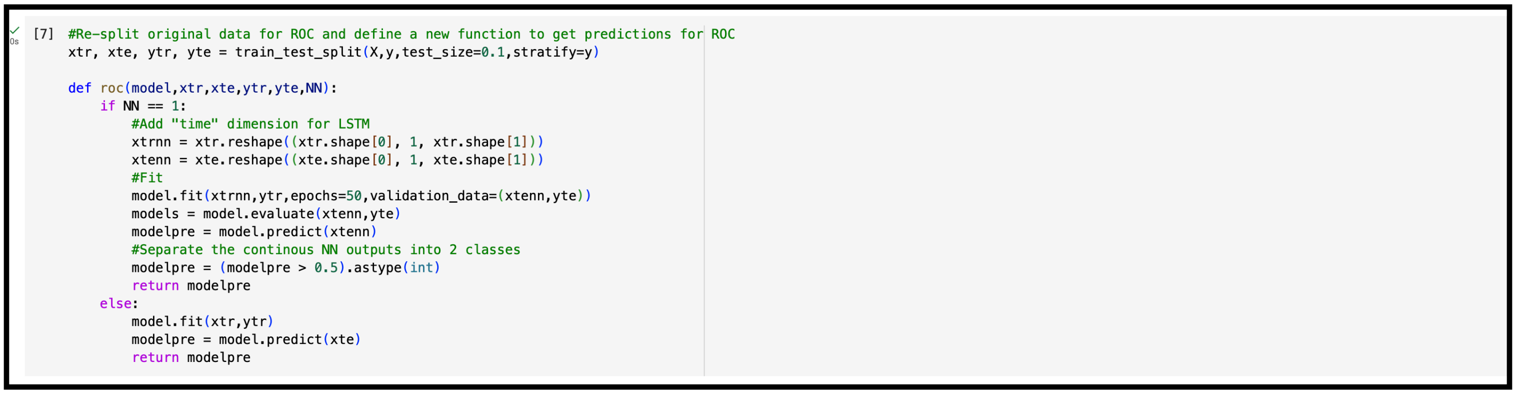


```
Random Forest Results:
          Recall  Precision  Accuracy  Error Rate  F1-Score    TSS    HSS   BACC    AUC
1           0.76      0.790     0.710       0.290      0.77  0.390  0.380  0.690  0.690
2           0.90      0.790     0.780       0.220      0.84  0.460  0.480  0.730  0.730
3           0.86      0.780     0.750       0.250      0.82  0.420  0.430  0.710  0.710
4           0.80      0.770     0.710       0.290      0.78  0.360  0.360  0.680  0.680
5           0.90      0.750     0.740       0.260      0.82  0.340  0.380  0.670  0.670
6           0.92      0.820     0.820       0.180      0.87  0.550  0.580  0.770  0.770
7           0.92      0.820     0.820       0.180      0.87  0.550  0.580  0.770  0.770
8           0.90      0.870     0.840       0.160      0.88  0.640  0.650  0.820  0.820
9           0.84      0.740     0.700       0.300      0.79  0.260  0.280  0.630  0.630
10          0.90      0.820     0.800       0.200      0.86  0.520  0.540  0.760  0.760
Average:    0.87      0.795     0.767       0.233      0.83  0.449  0.466  0.723  0.723

Support Vector Machine Results:
          Recall  Precision  Accuracy  Error Rate  F1-Score    TSS    HSS   BACC    AUC
1          0.820      0.770     0.730       0.270     0.790  0.380  0.390  0.690  0.690
2          0.940      0.800     0.810       0.190     0.860  0.500  0.540  0.750  0.750
3          0.820      0.790     0.740       0.260     0.800  0.410  0.420  0.710  0.710
4          0.800      0.770     0.710       0.290     0.780  0.360  0.360  0.680  0.680
5          0.900      0.740     0.730       0.270     0.810  0.310  0.340  0.650  0.650
6          0.880      0.760     0.740       0.260     0.820  0.360  0.390  0.680  0.680
7          0.880      0.800     0.780       0.220     0.840  0.470  0.490  0.740  0.740
8          0.920      0.840     0.830       0.170     0.880  0.590  0.610  0.790  0.790
9          0.860      0.750     0.720       0.280     0.800  0.320  0.340  0.660  0.660
10         0.920      0.840     0.830       0.170     0.880  0.570  0.600  0.790  0.790
Average:   0.874      0.786     0.762       0.238     0.826  0.427  0.448  0.714  0.714

LSTM Results:
          Recall  Precision  Accuracy  Error Rate  F1-Score    TSS    HSS   BACC    AUC
1          0.800      0.780     0.730       0.270     0.790  0.390  0.400  0.700  0.700
2          0.980      0.780     0.810       0.190     0.870  0.460  0.520  0.730  0.730
3          0.800      0.800     0.740       0.260     0.800  0.430  0.430  0.710  0.710
4          0.820      0.800     0.750       0.250     0.810  0.450  0.450  0.720  0.720
5          0.920      0.770     0.770       0.230     0.840  0.400  0.440  0.700  0.700
6          0.880      0.830     0.810       0.190     0.850  0.550  0.560  0.770  0.770
7          0.920      0.840     0.830       0.170     0.880  0.590  0.610  0.790  0.790
8          0.940      0.870     0.870       0.130     0.900  0.680  0.700  0.840  0.840
9          0.860      0.860     0.820       0.180     0.860  0.590  0.590  0.800  0.800
10         0.940      0.850     0.860       0.140     0.890  0.630  0.660  0.820  0.820
Average:   0.886      0.818     0.799       0.201     0.849  0.517  0.536  0.758  0.758
```

Results:

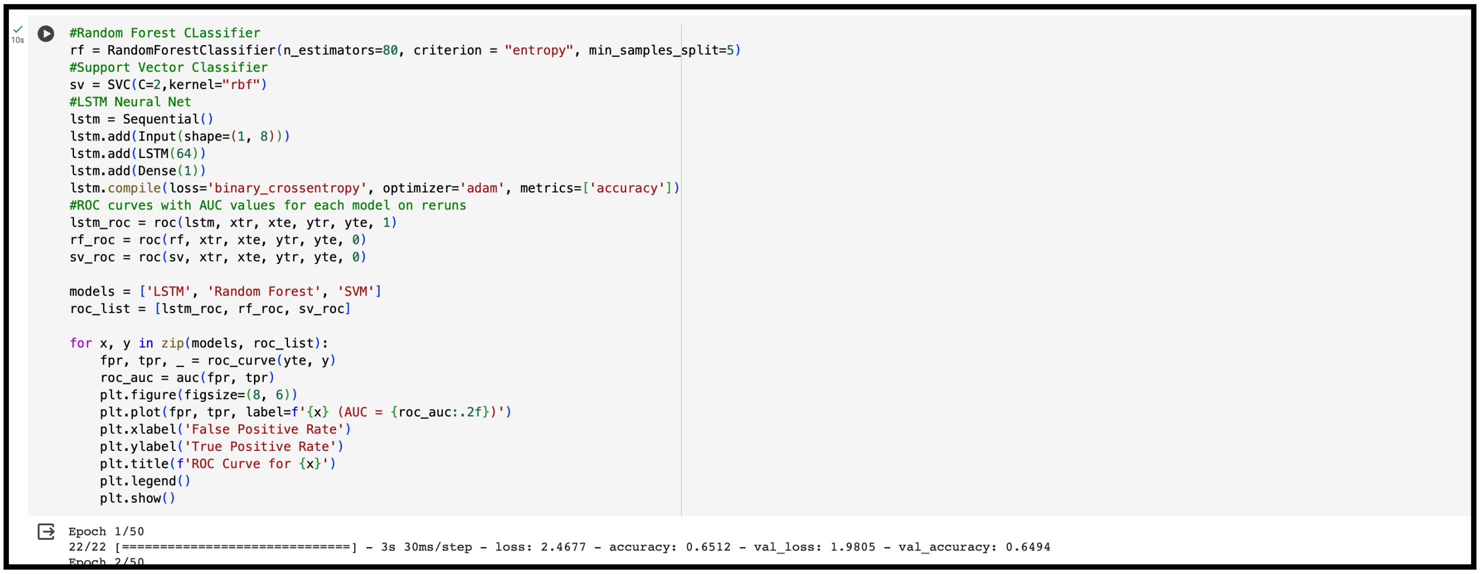
All three models (Random Forest, SVM, and LSTM) perform well in predicting diabetes, demonstrating high accuracy and AUC scores. Random Forest and SVM exhibit remarkably similar performance, indicating they might be equally suitable for this dataset. Even though LSTM is more suited for time-series data, the LSTM model demonstrates the highest average accuracy, suggesting its potential for achieving superior results with additional data or optimization.

Now we need to prepare the data again in order to run the models to generate an ROC curve for one model run vs K-Folds cross validation. A new function is defined to keep it clean and separate from the KFolds analysis.



```
[7] #Re-split original data for ROC and define a new function to get predictions for ROC
    xtr, xte, ytr, yte = train_test_split(X,y,test_size=0.1,stratify=y)

    def roc(model,xtr,xte,ytr,yte,NN):
        if NN == 1:
            #Add "time" dimension for LSTM
            xtrnn = xtr.reshape((xtr.shape[0], 1, xtr.shape[1]))
            xtenn = xte.reshape((xte.shape[0], 1, xte.shape[1]))
            #Fit
            model.fit(xtrnn,ytr,epochs=50,validation_data=(xtenn,yte))
            models = model.evaluate(xtenn,yte)
            modelpre = model.predict(xtenn)
            #Separate the continous NN outputs into 2 classes
            modelpre = (modelpre > 0.5).astype(int)
            return modelpre
        else:
            model.fit(xtr,ytr)
            modelpre = model.predict(xte)
            return modelpre
```

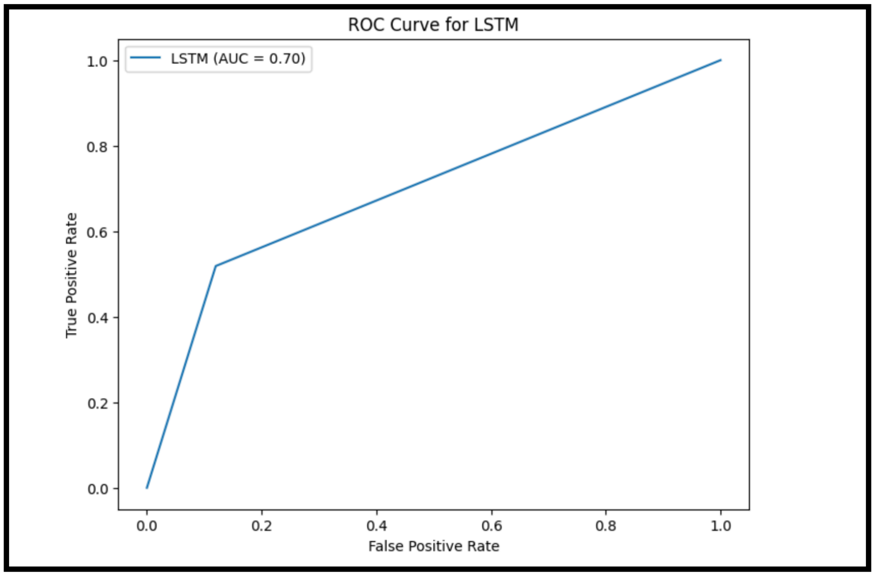
Now we need to re-prepare the models and generate the curves

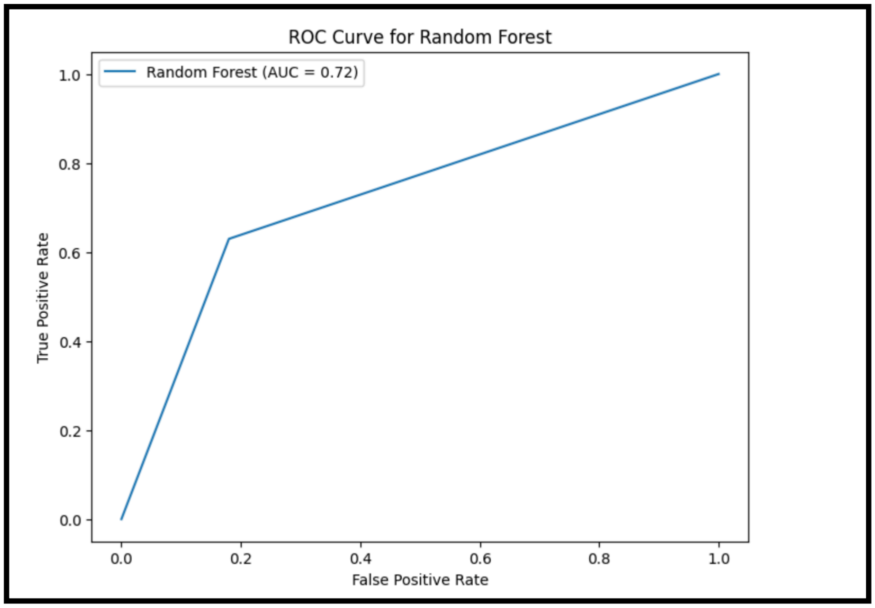


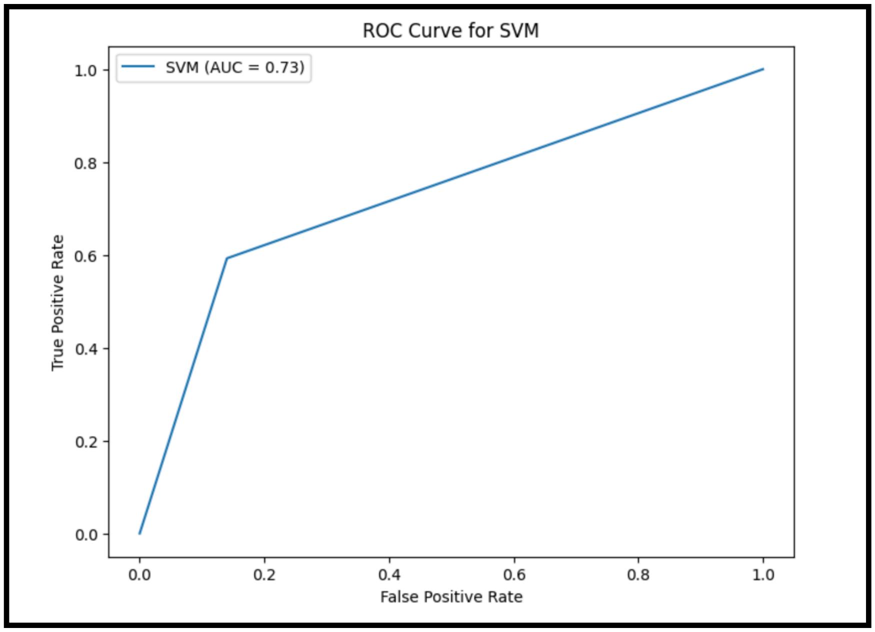
```
#Random Forest CLassifier
rf = RandomForestClassifier(n_estimators=80, criterion = "entropy", min_samples_split=5)
#Support Vector Classifier
sv = SVC(C=2,kernel="rbf")
#LSTM Neural Net
lstm = Sequential()
lstm.add(Input(shape=(1, 8)))
lstm.add(LSTM(64))
lstm.add(Dense(1))
lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
#ROC curves with AUC values for each model on reruns
lstm_roc = roc(lstm, xtr, xte, ytr, yte, 1)
rf_roc = roc(rf, xtr, xte, ytr, yte, 0)
sv_roc = roc(sv, xtr, xte, ytr, yte, 0)

models = ['LSTM', 'Random Forest', 'SVM']
roc_list = [lstm_roc, rf_roc, sv_roc]

for x, y in zip(models, roc_list):
    fpr, tpr, _ = roc_curve(yte, y)
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label=f'{x} (AUC = {roc_auc:.2f})')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {x}')
    plt.legend()
    plt.show()
```

```
Epoch 1/50
22/22 [==============================] - 3s 30ms/step - loss: 2.4677 - accuracy: 0.6512 - val_loss: 1.9805 - val_accuracy: 0.6494
Epoch 2/50
```

And here are the curves:







All 3 performed similarly. A larger dataset and more trials would provide more insight, and perhaps using a neural network model which isn't recurrent would be best for diabetes prediction.