This task is to build a classification model to detect diabetes. This dataset contains 768 observations and 9 features or variables as described by the following figures.

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

The variable outcome contains the presence of diabetes with 1 depicting its presence while 0 depicting its absence.

There are two library that we are going to use, namely scikit-learn and keras. Before running the experiment, we will first normalize our dataset, as can be seen from the following figure.

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	0.226180	0.198210	0.000000	0.058824	0.176471	0.352941	1.0
Glucose	768.0	0.607510	0.160666	0.000000	0.497487	0.587940	0.704774	1.0
BloodPressure	768.0	0.566438	0.158654	0.000000	0.508197	0.590164	0.655738	1.0
SkinThickness	768.0	0.207439	0.161134	0.000000	0.000000	0.232323	0.323232	1.0
Insulin	768.0	0.094326	0.136222	0.000000	0.000000	0.036052	0.150414	1.0
BMI	768.0	0.476790	0.117499	0.000000	0.406855	0.476900	0.545455	1.0
DiabetesPedigreeFunction	768.0	0.194990	0.136913	0.032231	0.100723	0.153926	0.258781	1.0
Age	768.0	0.410381	0.145188	0.259259	0.296296	0.358025	0.506173	1.0
Outcome	768.0	0.348958	0.476951	0.000000	0.000000	0.000000	1.000000	1.0

Now, our values range between 0 and 1. After normalizing our dataset, we split our data into train data and test data with 70% being train data and the rest being test data. This can be seen in the following figure.

```
X = df[predictors].values
y = df[target_column].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=40)
print(X_train.shape); print(X_test.shape)
```

```
(537, 8)
(231, 8)
```

In order to build neural network model with scikit-learn for this problem, the following line of code is used.

```
mlp = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu', solver='adam', max_iter=500)
```

While for keras, we can use the following code.

```
model = Sequential()
model.add(Dense(8, activation='relu', input_shape=(8,)))
model.add(Dense(8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, name='output'))
```

```
model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['accuracy'])
```

As can be seen from the two figures, I used relu as the activation function. I also used 3 hidden layers with 8 neurons each. For the optimizer, adam was used. For both scikit-learn and keras, after building the model, we can call 'fit' function to fit the model to the training data. The figure bellow shows the syntax for scikit learn,

```
mlp.fit(X_train,y_train)
```

While the following figure show the syntax for keras.

```
history = model.fit(X_train, y_train,
                    epochs=500,
                    verbose=0,
                    validation_data=(X_test, y_test))
```

After training our model, in scikit-learn, we use the function 'predict' to generate predictions on the train and test data.

```
predict_train = mlp.predict(X_train)
predict_test = mlp.predict(X_test)
```

After having our prediction result, we can call the following function to evaluate it.

```
print(confusion_matrix(y_train,predict_train))
print(classification_report(y_train,predict_train))
```

```
[[314  44]
 [ 72 107]]
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       358
           1       0.71      0.60      0.65       179

    accuracy                           0.78       537
   macro avg       0.76      0.74      0.75       537
weighted avg       0.78      0.78      0.78       537
```

```
print(confusion_matrix(y_test,predict_test))
print(classification_report(y_test,predict_test))
```

```
[[120  22]
 [ 37  52]]
              precision    recall  f1-score   support

           0       0.76      0.85      0.80       142
           1       0.70      0.58      0.64        89

    accuracy                           0.74       231
   macro avg       0.73      0.71      0.72       231
weighted avg       0.74      0.74      0.74       231
```

As can be seen from the two images above, we have 78% and 74% accuracy on train and test data respectively.

For keras, we can evaluate our model by calling ‘evaluate’ function.

```
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.20717827356480933
Test accuracy: 0.7142857148017718
```

As can be seen from the image above, we get the accuracy of around 71.4% on the test data.