```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
```

Data Preprocessing

```
# Read Dataset to pandas dataframe
heartdata = pd.read_csv('heart.csv')

heartdata.head()

# Assign data from first ten columns to x variable
x = heartdata.iloc[:, 0:11]

# Assign data from the eleventh column to y variable
y = heartdata.iloc[:, 11:12]


# Convert x from categorical to numerical
from sklearn import preprocessing
le = preprocessing.LabelEncoder()

y = y.apply(le.fit_transform)
x = x.apply(le.fit_transform)

# Recheck column values
x.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Ex
0	12	1	1	41	147	0	1	98	
1	21	0	2	55	40	0	1	82	
2	9	1	1	31	141	0	2	25	
3	20	0	0	39	72	0	1	34	
4	26	1	2	49	53	0	1	48	

Train, Test, Split and Feature Scaling

```
# Train, Test, Split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20)

# Feature Scaling (Standardization)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_train)

x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
print(x_test)
```

```
[[-1.23159588 -1.90933748  1.28038485 ... -0.83846064 -0.88056038
  -0.57218741]
 [ 2.53010881  0.52374188 -0.81973242 ...  1.19266183 -0.88056038
   1.06838119]
 [ 1.56281332  0.52374188 -0.81973242 ... -0.83846064  2.27564322
  -0.57218741]
 ...
 [-0.15682311  0.52374188  0.23032621 ... -0.83846064 -0.88056038
   1.06838119]
 [-1.98393682  0.52374188  0.23032621 ... -0.83846064 -0.88056038
   1.06838119]
 [-0.26430039 -1.90933748  1.28038485 ... -0.83846064  0.59891006
   1.06838119]]
```

```
# Train, Test, Split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20)

# Feature Scaling (Standardization)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(x)

X_standardized = scaler.transform(x)

data = pd.DataFrame(X_standardized)
data.describe()
```

	0	1	2	3	4	
count	9.180000e+02	9.180000e+02	9.180000e+02	9.180000e+02	9.180000e+02	9.1800(
mean	-7.304735e-17	-9.447780e-16	-7.178958e-16	1.079988e-16	4.573925e-16	-1.00
std	1.000545e+00	1.000545e+00	1.000545e+00	1.000545e+00	1.000545e+00	1.0005
min	-2.706015e+00	-1.938163e+00	-8.169950e-01	-2.334820e+00	-1.390554e+00	-5.51
25%	-6.906294e-01	5.159524e-01	-8.169950e-01	-7.735319e-01	-8.242184e-01	-5.51
50%	5.188098e-02	5.159524e-01	-8.169950e-01	-1.348231e-01	-1.090980e-02	-5.51
75%	6.883185e-01	5.159524e-01	1.275059e+00	5.748534e-01	7.385264e-01	-5.51
max	2.491558e+00	5.159524e-01	2.321086e+00	2.349045e+00	2.373660e+00	1.8137

Artificial Neural Network (ANN)

```
# Training and Predictions
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes = (10, 10), max_iter = 2000)
mlp.fit(x_train, y_train.values.ravel())
MLPClassifier(hidden_layer_sizes = (10, 10), max_iter = 2000)

# Making Predictions
predictions = mlp.predict(x_test)

# Evaluating algorithm
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
[[64 14]
 [19 87]]
              precision    recall  f1-score   support

           0       0.77      0.82      0.80        78
           1       0.86      0.82      0.84       106

    accuracy                           0.82       184
   macro avg       0.82      0.82      0.82       184
weighted avg       0.82      0.82      0.82       184
```

Deep Neural Network (DNN)

```
from sklearn.model_selection import GridSearchCV, KFold
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.optimizers import Adam
```

Do a grid search for the optimal batch size and number of epochs

```
# Define a random seed
seed = 6
np.random.seed(seed)

# Start defining the model
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(30, input_dim = 11, kernel_initializer='normal', activation='tanh'))
```

```
    model.add(Dense(20, activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))

    # compile the model
    adam = Adam(learning_rate = 0.01)
    model.compile(loss = 'binary_crossentropy', optimizer = adam, metrics = ['accuracy'])
    return model

# create the model
model = KerasClassifier(build_fn = create_model, verbose = 1)

# define the grid search parameters
batch_size = [10, 20, 40]
epochs = [10, 50, 100]

# make a dictionary of the grid search parameters
param_grid = dict(batch_size=batch_size, epochs=epochs)

# build and fit the GridSearchCV
grid = GridSearchCV(estimator = model, param_grid = param_grid, cv = KFold(random_state=Nc
grid_results = grid.fit(X_standardized, y)

# summarize the results
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))
means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

```
 19/19 [==============================] - 0s 3ms/step - loss: 0.0109 - accuracy: 1
 Epoch 89/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0104 - accuracy: 1
 Epoch 90/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0098 - accuracy: 1
 Epoch 91/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0099 - accuracy: 0
 Epoch 92/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0114 - accuracy: 0
 Epoch 93/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0088 - accuracy: 1
 Epoch 94/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0084 - accuracy: 1
 Epoch 95/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0084 - accuracy: 1
 Epoch 96/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0079 - accuracy: 1
 Epoch 97/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0075 - accuracy: 1
 Epoch 98/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0076 - accuracy: 1
 Epoch 99/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0089 - accuracy: 0
 Epoch 100/100
 19/19 [==============================] - 0s 2ms/step - loss: 0.0136 - accuracy: 0
 5/5 [==============================] - 0s 2ms/step - loss: 0.9383 - accuracy: 0.7
 [CV 5/5; 9/9] END ....batch_size=40, epochs=100;, score=0.781 total time=   5.9s
 Epoch 1/10
 92/92 [==============================] - 1s 2ms/step - loss: 0.4149 - accuracy: 0
```

```
92/92 [==============================] - 1s 2ms/step - loss: 0.4149 - accuracy: 0
Epoch 2/10
92/92 [==============================] - 0s 2ms/step - loss: 0.3837 - accuracy: 0
Epoch 3/10
92/92 [==============================] - 0s 2ms/step - loss: 0.3620 - accuracy: 0
Epoch 4/10
92/92 [==============================] - 0s 2ms/step - loss: 0.3503 - accuracy: 0
Epoch 5/10
92/92 [==============================] - 0s 2ms/step - loss: 0.3417 - accuracy: 0
Epoch 6/10
92/92 [==============================] - 0s 2ms/step - loss: 0.3428 - accuracy: 0
Epoch 7/10
92/92 [==============================] - 0s 2ms/step - loss: 0.3269 - accuracy: 0
Epoch 8/10
92/92 [==============================] - 0s 2ms/step - loss: 0.3180 - accuracy: 0
Epoch 9/10
92/92 [==============================] - 0s 2ms/step - loss: 0.3054 - accuracy: 0
Epoch 10/10
92/92 [==============================] - 0s 2ms/step - loss: 0.3012 - accuracy: 0
Best: 0.8408410668373107, using {'batch_size': 10, 'epochs': 10}
0.8408410668373107 (0.05035498960911226) with: {'batch_size': 10, 'epochs': 10}
0.8081967234611511 (0.05231542721626151) with: {'batch_size': 10, 'epochs': 50}
0.8277737975120545 (0.04468666054623347) with: {'batch_size': 10, 'epochs': 100}
0.8375504732131958 (0.05699627005299195) with: {'batch_size': 20, 'epochs': 10}
0.8136255621910096 (0.0455254209913486) with: {'batch_size': 20, 'epochs': 50}
0.8049239754676819 (0.0426845026420387) with: {'batch_size': 20, 'epochs': 100}
0.8332086086273194 (0.05138278139361874) with: {'batch_size': 40, 'epochs': 10}
0.8278094530105591 (0.03212091608706881) with: {'batch_size': 40, 'epochs': 50}
0.8255998849868774 (0.04125767045110486) with: {'batch_size': 40, 'epochs': 100}
```

Do a grid search to find the optimal number of neurons in each hidden layer

```
# import necessary packages

# Define a random seed
seed = 6
np.random.seed(seed)

# Start defining the model
def create_model(neuron1, neuron2):
    # create model
    model = Sequential()
    model.add(Dense(neuron1, input_dim = 11, kernel_initializer= 'uniform', activation= 'l
    model.add(Dense(neuron2, input_dim = neuron1, kernel_initializer= 'uniform', activatic
    model.add(Dense(1, activation='sigmoid'))

    # compile the model
    adam = Adam(learning_rate = 0.001)
    model.compile(loss = 'binary_crossentropy', optimizer = adam, metrics = ['accuracy'])
    return model

# create the model
model = KerasClassifier(build_fn = create_model, epochs = 100, batch_size = 20, verbose =

# define the grid search parameters
```

```
neuron1 = [4, 8, 16]
neuron2 = [2, 4, 8]

# make a dictionary of the grid search parameters
param_grid = dict(neuron1 = neuron1, neuron2 = neuron2)

# build and fit the GridSearchCV
grid = GridSearchCV(estimator = model, param_grid = param_grid, cv = KFold(random_state=Nc
grid_results = grid.fit(X_standardized, y)

# summarize the results
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))
means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

```
[CV 2/5; 5/9] END .........neuron1=8, neuron2=4;, score=0.848 total time=   5.7s
[CV 3/5; 5/9] START neuron1=8, neuron2=4.....................................
[CV 3/5; 5/9] END .........neuron1=8, neuron2=4;, score=0.870 total time=   4.6s
[CV 4/5; 5/9] START neuron1=8, neuron2=4.....................................
[CV 4/5; 5/9] END .........neuron1=8, neuron2=4;, score=0.803 total time=   4.6s
[CV 5/5; 5/9] START neuron1=8, neuron2=4.....................................
[CV 5/5; 5/9] END .........neuron1=8, neuron2=4;, score=0.770 total time=   4.5s
[CV 1/5; 6/9] START neuron1=8, neuron2=8.....................................
[CV 1/5; 6/9] END .........neuron1=8, neuron2=8;, score=0.837 total time=   4.7s
[CV 2/5; 6/9] START neuron1=8, neuron2=8.....................................
[CV 2/5; 6/9] END .........neuron1=8, neuron2=8;, score=0.848 total time=   6.0s
[CV 3/5; 6/9] START neuron1=8, neuron2=8.....................................
[CV 3/5; 6/9] END .........neuron1=8, neuron2=8;, score=0.870 total time=   5.7s
[CV 4/5; 6/9] START neuron1=8, neuron2=8.....................................
[CV 4/5; 6/9] END .........neuron1=8, neuron2=8;, score=0.809 total time=   5.7s
[CV 5/5; 6/9] START neuron1=8, neuron2=8.....................................
[CV 5/5; 6/9] END .........neuron1=8, neuron2=8;, score=0.770 total time=   4.7s
[CV 1/5; 7/9] START neuron1=16, neuron2=2....................................
[CV 1/5; 7/9] END ........neuron1=16, neuron2=2;, score=0.837 total time=   5.7s
[CV 2/5; 7/9] START neuron1=16, neuron2=2....................................
[CV 2/5; 7/9] END ........neuron1=16, neuron2=2;, score=0.848 total time=   5.7s
[CV 3/5; 7/9] START neuron1=16, neuron2=2....................................
[CV 3/5; 7/9] END ........neuron1=16, neuron2=2;, score=0.870 total time=   5.7s
[CV 4/5; 7/9] START neuron1=16, neuron2=2....................................
[CV 4/5; 7/9] END ........neuron1=16, neuron2=2;, score=0.809 total time=   4.5s
[CV 5/5; 7/9] START neuron1=16, neuron2=2....................................
[CV 5/5; 7/9] END ........neuron1=16, neuron2=2;, score=0.770 total time=   4.6s
[CV 1/5; 8/9] START neuron1=16, neuron2=4....................................
[CV 1/5; 8/9] END ........neuron1=16, neuron2=4;, score=0.842 total time=   5.7s
[CV 2/5; 8/9] START neuron1=16, neuron2=4....................................
[CV 2/5; 8/9] END ........neuron1=16, neuron2=4;, score=0.853 total time=   5.7s
[CV 3/5; 8/9] START neuron1=16, neuron2=4....................................
[CV 3/5; 8/9] END ........neuron1=16, neuron2=4;, score=0.870 total time=   4.8s
[CV 4/5; 8/9] START neuron1=16, neuron2=4....................................
[CV 4/5; 8/9] END ........neuron1=16, neuron2=4;, score=0.809 total time=   5.7s
[CV 5/5; 8/9] START neuron1=16, neuron2=4....................................
[CV 5/5; 8/9] END ........neuron1=16, neuron2=4;, score=0.770 total time=   4.6s
[CV 1/5; 9/9] START neuron1=16, neuron2=8....................................
[CV 1/5; 9/9] END ........neuron1=16, neuron2=8;, score=0.837 total time=   5.7s
[CV 2/5; 9/9] START neuron1=16, neuron2=8....................................
[CV 2/5; 9/9] END ........neuron1=16, neuron2=8;, score=0.853 total time=   4.6s
[CV 3/5; 9/9] START neuron1=16, neuron2=8....................................
```

```
[CV 3/5; 9/9] START neuron1=16, neuron2=8.....................................
[CV 3/5; 9/9] END .......neuron1=16, neuron2=8;, score=0.870 total time=   4.5s
[CV 4/5; 9/9] START neuron1=16, neuron2=8.....................................
[CV 4/5; 9/9] END .......neuron1=16, neuron2=8;, score=0.809 total time=   4.5s
[CV 5/5; 9/9] START neuron1=16, neuron2=8.....................................
[CV 5/5; 9/9] END .......neuron1=16, neuron2=8;, score=0.770 total time=   4.5s
Best: 0.8288904666900635, using {'neuron1': 16, 'neuron2': 4}
0.8267165422439575 (0.03426584590587069) with: {'neuron1': 4, 'neuron2': 2}
0.8267165422439575 (0.03426584590587069) with: {'neuron1': 4, 'neuron2': 4}
0.8267165422439575 (0.03426584590587069) with: {'neuron1': 4, 'neuron2': 8}
0.8267106056213379 (0.03532102443466339) with: {'neuron1': 8, 'neuron2': 2}
0.8256236433982849 (0.034902894484794715) with: {'neuron1': 8, 'neuron2': 4}
0.8267165422439575 (0.03426584590587069) with: {'neuron1': 8, 'neuron2': 8}
0.8267165422439575 (0.03426584590587069) with: {'neuron1': 16, 'neuron2': 2}
0.8288904666900635 (0.0353466903424687) with: {'neuron1': 16, 'neuron2': 4}
0.8278035044670105 (0.0349966376632872) with: {'neuron1': 16, 'neuron2': 8}
```

Generate predictions with optimal hyperparameters

```
y_pred = grid.predict(X_standardized)
```

```
print(y_pred.shape)
```

```
(918, 1)
```

```
print(y_pred[:5])
```

```
[[0]
 [0]
 [0]
 [1]
 [0]]
```

Generate a classification report

```
from sklearn.metrics import classification_report, accuracy_score

print(accuracy_score(y, y_pred))
print(classification_report(y, y_pred))
```

```
0.8496732026143791
              precision    recall  f1-score   support

           0       0.84      0.82      0.83       410
           1       0.86      0.87      0.87       508

    accuracy                           0.85       918
   macro avg       0.85      0.85      0.85       918
weighted avg       0.85      0.85      0.85       918
```