```
####################################################
# Project1 B Key
####################################################

####################################################
# Importing necessary libraries
####################################################
import pandas as pd                                # Import pandas for data
manipulation/handling
import numpy as np                                 # Import numpy for number
processing
from pandas.plotting import scatter_matrix         # Used for plotting scatter
matrix
import matplotlib                                  # Need to import matplotlib
matplotlib.use("TkAgg")                            # Used to prevent crash on my
mac
import matplotlib.pyplot as plt                    # Used to plot

# import the various Machine Learning modules from sklearn:
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler



####################################################
# Perceptron
####################################################
# Reading in heart disease data
heart_df = pd.read_csv("heart1.csv")        # reading the data from the csv file

# Preparing to split out validation dataset
heart_array = heart_df.values               # Array to hold all data
X = heart_array[:,0:13]                     # Array to hold all input data
Y = heart_array[:,13]                       # Array to hold all answers
validation_size = 0.2                       # Use 1/5 of the data for validation

seed = int(10*np.random.rand())             # Seed to feed to model_selection

# Splitting out training set
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X,
Y,test_size=validation_size, random_state=seed)



####################################################
# Scaling all input data with standard scaler
####################################################
sc = StandardScaler()                       # Creating an instance of a scaler
class
sc.fit(X_train)                             # fitting the scaler on X_train
X_train = sc.transform(X_train)             # Fitting the training input data
X_test = sc.transform(X_test)               # Fitting the training input data
scoring = 'accuracy'                        # Test options and evaluation metric


####################################################
# Perceptron
####################################################
```

```
# Calling MLPClassifier with options tuned to the best I could find
model = MLPClassifier(hidden_layer_sizes=(800,), activation='logistic',
max_iter=2000, alpha=0.00001, solver='adam', random_state=seed,tol=0.00001)
model.fit(X_train,Y_train)
# Fitting model on training data
Y_predictions = model.predict(X_test)
# Making predictions on unseen data
correct_or_no = np.array(Y_predictions) - np.array(Y_test)
# Taking the difference between the arrays
number_correct = np.sum(correct_or_no == 0)
# Counting the number of zeros
total_samples = len(correct_or_no)
# Finding total number of zeros
print("Accuracy of Perceptron:\t\t\t%f" % (number_correct/total_samples))
# Printing out the accuracy of the algorithm



###################################################
# Logistic Regression
###################################################
model = LogisticRegression(solver='saga', multi_class='ovr',C=1000)
# C = 200 gave best accuracy, same with 'saga'
model.fit(X_train,Y_train)
# Fitting model on training data
Y_predictions = model.predict(X_test)
# Making predictions on unseen data
correct_or_no = np.array(Y_predictions) - np.array(Y_test)
# Taking the difference between the arrays
number_correct = np.sum(correct_or_no == 0)
# Counting the number of zeros
total_samples = len(correct_or_no)
# Finding total number of zeros
print("Accuracy of Logistic Regression:\t%f" % (number_correct/total_samples))



###################################################
# Support Vector Machine
###################################################
model = SVC(C=5,kernel='sigmoid',gamma='auto',tol=.0000001,max_iter=20000)
# sigmoid gave best accuracy, as did C = 5
model.fit(X_train,Y_train)
# Fitting model on training data
Y_predictions = model.predict(X_test)
# Making predictions on unseen data
correct_or_no = np.array(Y_predictions) - np.array(Y_test)
# Taking the difference between the arrays
number_correct = np.sum(correct_or_no == 0)
# Counting the number of zeros
total_samples = len(correct_or_no)
# Finding total number of zeros
print("Accuracy of Support Vector Machine:\t%f" %
(number_correct/total_samples))



###################################################
# Decision Tree
###################################################
model = DecisionTreeClassifier(criterion='entropy',splitter='random')
# random splits = higher accuracy, criterion = entropy was higher than other
options
model.fit(X_train,Y_train)
```

```
# Fitting model on training data
Y_predictions = model.predict(X_test)
# Making predictions on unseen data
correct_or_no = np.array(Y_predictions) - np.array(Y_test)
# Taking the difference between the arrays
number_correct = np.sum(correct_or_no == 0)
# Counting the number of zeros
total_samples = len(correct_or_no)
# Finding total number of zeros
print("AAccuracy of Decision Tree:\t\t%f" % (number_correct/total_samples))



###################################################
# KNN
###################################################
knn_accuracies = {}
for k in range(27,30):
    model = KNeighborsClassifier(n_neighbors=k,algorithm='auto')
# creating KNN model
    model.fit(X_train,Y_train)
# Fitting model on training data
    Y_predictions = model.predict(X_test)
# Making predictions on unseen data
    correct_or_no = np.array(Y_predictions) - np.array(Y_test)
# Taking the difference between the arrays
    number_correct = np.sum(correct_or_no == 0)
# Counting the number of zeros
    total_samples = len(correct_or_no)
# Finding total number of zeros
    knn_accuracies[str(k)] = number_correct/total_samples



key_max = max(knn_accuracies.keys(), key=(lambda k: knn_accuracies[k]))
# Finding the best value of K
print('Accuracy of best K value (%d)\t\t%f' %
(int(key_max),knn_accuracies[key_max]))   # Printing out the best value of K
```