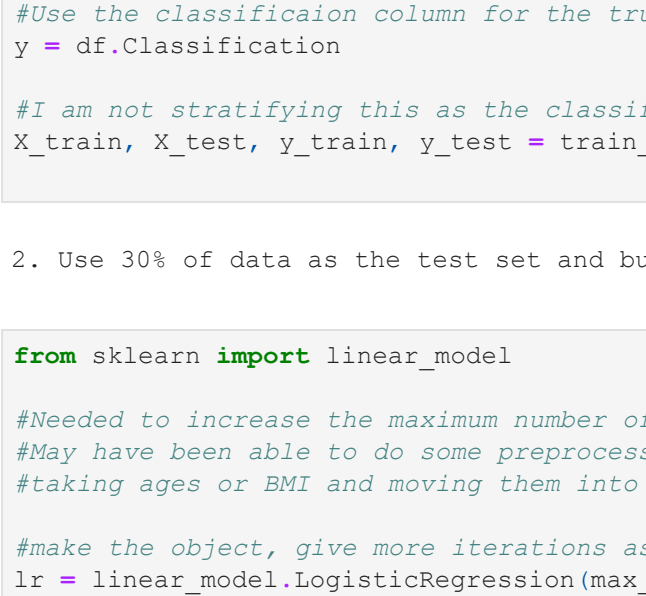
In [1]:
```python
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 28 19:25:46 2022

@author: Brandon Botzer - btb5103



Attribute Information:

Quantitative    Attributes:
Age                 (years)
BMI                 (kg/m2)
Glucose         (mg/dL)
Insulin         (µU/mL)
HOMA
Leptin          (ng/mL)
Adiponectin         (µg/mL)
Resistin            (ng/mL)
MCP-1(pg/dL)    (ng/mL)


Labels:

1 = Healthy Controls

2 = Patients

1. Perform Data exploratory analysis on the data (10 points)
2. Use 30% of data as the test set and build a Logistic regression model to predict Labels variable (20 points)
3. Build the Naïve Bayes model to predict Labels variable (20 points)
4. Build the Decision tree model to predict Labels variable (20 points)
5. Build Neural network model to predict Labels variable (20 points)
6. Which model is the best? Which variable is the most important one? (10 points)




"""

#Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import os

from sklearn.model_selection import train_test_split
from sklearn import metrics

#0. Import the data

#Set the path for the CSV file
readPath = "J:\DSDegree\PennState\DAAN_862\Week 8\Homework"

#Change the directory
os.chdir(readPath)

#Read the CSV file in
df = pd.read_csv("breastcancer.csv")
```

In [2]:
```python
#1. Perform Data exploratory analysis on the data (10 points)

print("\n\n1. Perform Data exploratory analysis on the data (10 points)\n")
#Look at a correlation matrix
df.corr()

#plot a correlation matrix
plt.matshow(df.corr())
plt.title("Corr Matrix on Breast Cancer")
plt.colorbar()
plt.xticks(range(10), list(df.columns))
plt.yticks(range(10), list(df.columns))

#Note to self, it looks like Adiponectin has very low correlation values
#to many things.  When considering the last row of classification,
#it looks as if Glucose may be the determining feature.
#There is also an off diagonal of high correlation between features.
#I wonder if these could be confounding against one another.
#Especially Glucose and HOMA.
```

```
1. Perform Data exploratory analysis on the data (10 points)
```

Out[2]:
```
([<matplotlib.axis.YTick at 0x25739294d90>,
  <matplotlib.axis.YTick at 0x25739294610>,
  <matplotlib.axis.YTick at 0x25739278460>,
  <matplotlib.axis.YTick at 0x257399ea880>,
  <matplotlib.axis.YTick at 0x257399f10a0>,
  <matplotlib.axis.YTick at 0x257399f1760>,
  <matplotlib.axis.YTick at 0x257399f7040>,
  <matplotlib.axis.YTick at 0x257399f7640>,
  <matplotlib.axis.YTick at 0x257399f1850>,
  <matplotlib.axis.YTick at 0x257399df220>],
 [Text(0, 0, 'Age'),
  Text(0, 1, 'BMI'),
  Text(0, 2, 'Glucose'),
  Text(0, 3, 'Insulin'),
  Text(0, 4, 'HOMA'),
  Text(0, 5, 'Leptin'),
  Text(0, 6, 'Adiponectin'),
  Text(0, 7, 'Resistin'),
  Text(0, 8, 'MCP.1'),
  Text(0, 9, 'Classification')])
```



In [3]:
```python
#2. Use 30% of data as the test set and build a Logistic regression model to
#predict Labels variable (20 points)

print("\n\n2. Use 30% of data as the test set and build a Logistic" +
      " regression model to predict Labels variable (20 points)\n")

#Split the data into training and test sets
#I'll be using these repeatedly
#Get rid of the classificaion column in the X direction
X = df.iloc[:, 0:9]
#Use the classificaion column for the true answer
y = df.Classification

#I am not stratifying this as the classification proportions are faily equal
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.3)
```

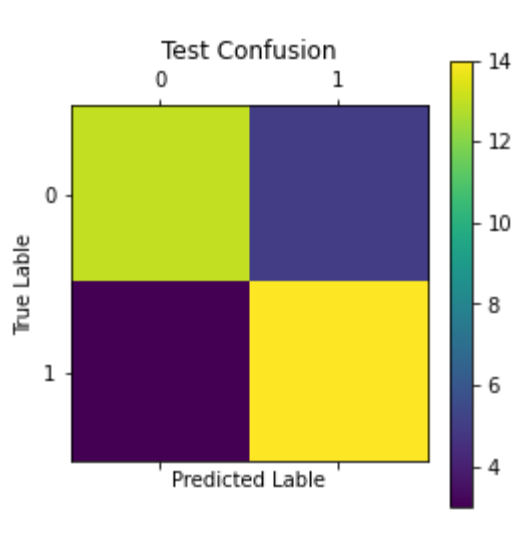
```
2. Use 30% of data as the test set and build a Logistic regression model to predict Labels variable (20 points)
```

In [4]:
```python
from sklearn import linear_model

#Needed to increase the maximum number of iterations
#May have been able to do some preprocessing of
#taking ages or BMI and moving them into grouped bins

#make the object, give more iterations as it was maxing out
lr = linear_model.LogisticRegression(max_iter = 1000)

#Call the fit using the training data
lr.fit(X_train, y_train)

#Look at some fit information
print("Coefficients of the models: \n" + str(lr.coef_))
print("\n")
print("Inercepts of the models: \n" + str(lr.intercept_))


#Run predictions for training and test sets and evaluate model performance

print("\n\nCacluating predictions and evaluating model performance...\n\n")
#predictions for training data
lr_train_pred = lr.predict(X_train)
#predictions for test data
lr_test_pred = lr.predict(X_test)

#look at metrics
print("\nAccuracy scores for the LogReg training data:\n")
print(str(metrics.accuracy_score(y_train, lr_train_pred)))

print("\n\nAccuracy scores for the LogReg test data:\n")
print(str(metrics.accuracy_score(y_test, lr_test_pred)))

#store the test accuracy score for later
lr_test_acc_score = metrics.accuracy_score(y_test, lr_test_pred)
```

```
Coefficients of the models:
[[-0.02331395 -0.08625893  0.10144227  0.136907   -0.33857971 -0.01511282
   0.02043387  0.03631697  0.00100965]]


Inercepts of the models:
[-6.82154149]


Cacluating predictions and evaluating model performance...



Accuracy scores for the LogReg training data:

0.7160493827160493


Accuracy scores for the LogReg test data:

0.7714285714285715
```

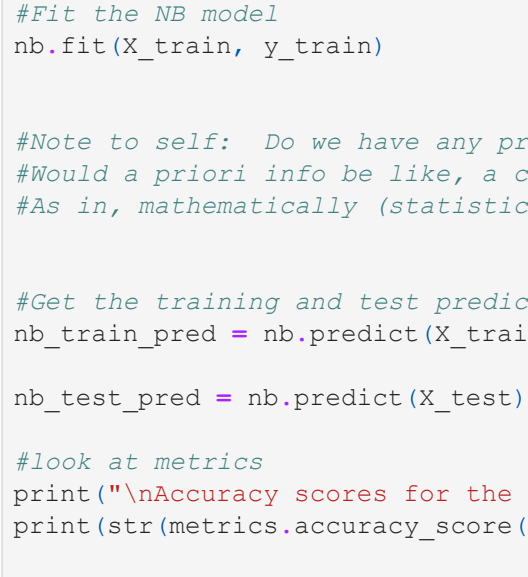
In [5]:
```python
#Make confusion matrix and plot for training data
train_cm = metrics.confusion_matrix(y_train, lr_train_pred)

#plot the confusion matrix
plt.matshow(train_cm)
plt.title("Training Confusion")
plt.colorbar()
plt.ylabel("True Lable")
plt.xlabel("Predicted Lable")
```

Out[5]:
```
Text(0.5, 0, 'Predicted Lable')
```



In [6]:
```python
#Make confusion matrix and plot for test data
test_cm = metrics.confusion_matrix(y_test, lr_test_pred)

#plot the confusion matrix
plt.matshow(test_cm)
plt.title("Test Confusion")
plt.colorbar()
plt.ylabel("True Lable")
plt.xlabel("Predicted Lable")
```

Out[6]:
```
Text(0.5, 0, 'Predicted Lable')
```



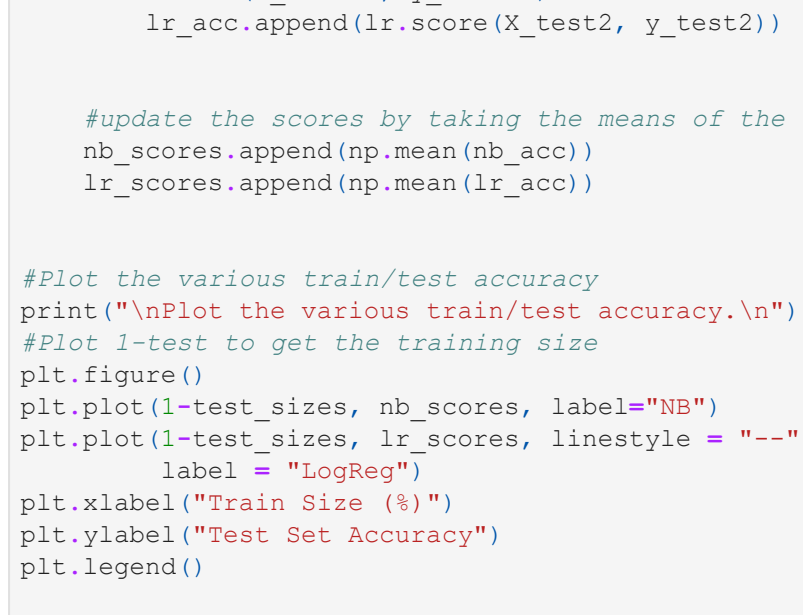
In [7]:
```python
#3. Build the Naïve Bayes model to predict Labels variable (20 points)
print("\n\n3. Build the Naïve Bayes model to predict Labels "+
      "variable (20 points)\n\n")

#Import the Gaussian NB
from sklearn.naive_bayes import GaussianNB

#Make the Naive Bayes object
nb = GaussianNB()

#Fit the NB model
nb.fit(X_train, y_train)


#Note to self:  Do we have any priori probabilites for breast cancer?
#Would a priori info be like, a coin flip will be 50/50?
#As in, mathematically (statistically) we know this to be true?


#Get the training and test predictions based on the built model
nb_train_pred = nb.predict(X_train)

nb_test_pred = nb.predict(X_test)

#look at metrics
print("\nAccuracy scores for the NB training data:\n")
print(str(metrics.accuracy_score(y_train, nb_train_pred)))

print("\n\nAccuracy scores for the NB test data:\n")
print(str(metrics.accuracy_score(y_test, nb_test_pred)))

#store the test nb accuracy score for later
nb_test_acc_score = metrics.accuracy_score(y_test, nb_test_pred)
```

```
3. Build the Naïve Bayes model to predict Labels variable (20 points)



Accuracy scores for the NB training data:

0.654320987654321


Accuracy scores for the NB test data:

0.6285714285714286
```

In [8]:
```python
#Let's look at LogReg and GausNB as training set increases size
print("\n\nLet's look at LogReg and GausNB as training set increases size")


lr_scores = []
nb_scores = []

test_sizes = np.linspace(0.1, 0.6, 11)

#Loop for each of the training sizes
for test_size in test_sizes:

    nb_acc = []
    lr_acc = []

    #Loop to run each training size 30 times to find averages
    for i in range(30):

        #run the train/test split
        #Use '2' so the first t/T split can be used again
        #for the dt and NN tests
        X_train2,X_test2,y_train2,y_test2 = train_test_split(X, y,
                                                             test_size = test_size)

        #Get fit and score data for GausNB
        nb.fit(X_train2, y_train2)
        nb_acc.append(nb.score(X_test2, y_test2))

        #Get fit and score data for LogReg
        #still running into number of iteration errors for some fits
        lr.fit(X_train2, y_train2)
        lr_acc.append(lr.score(X_test2, y_test2))


    #update the scores by taking the means of the 30 train/test splits
    nb_scores.append(np.mean(nb_acc))
    lr_scores.append(np.mean(lr_acc))


#Plot the various train/test accuracy
print("\nPlot the various train/test accuracy.\n")
#Plot 1-test to get the training size
plt.figure()
plt.plot(1-test_sizes, nb_scores, label="NB")
plt.plot(1-test_sizes, lr_scores, linestyle = "--",
         label = "LogReg")
plt.xlabel("Train Size (%)")
plt.ylabel("Test Set Accuracy")
plt.legend()

#Note to self, we'll be doing something similar to this with
#Cross-Validation and grid searches next week
```

```
Let's look at LogReg and GausNB as training set increases size

Plot the various train/test accuracy.
```

Out[8]:
```
<matplotlib.legend.Legend at 0x25739dfdcd0>
```



In [9]:
```python
#4. Build the Decision tree model to predict Labels variable (20 points)
print("\n\n4. Build the Decision tree model to predict " +
      "Labels variable (20 points)\n\n")

from sklearn import tree

#Make the Decision Tree object
#Setting the max depth to 10 so I don't get a overly branched tree
#May want to loop over different values of these hyperparams later
dt = tree.DecisionTreeClassifier(max_depth = 10, min_samples_split = 3)

#train the dt model from train data
#uses the 'gini' criterion
dt.fit(X_train, y_train)

#model evaluation
dt_pred = dt.predict(X_test)

#Accuracy scores
print("\n\nAccuracy scores for the DT test data:\n")
print(str(metrics.accuracy_score(y_test, dt_pred)))

#store the test accuracy score for later
dt_test_acc_score = metrics.accuracy_score(y_test, dt_pred)

#Look at the feature importance
print("\nFeature importance for the decision tree:")
print(str(dt.feature_importances_))

#store the importance in the data frame
dtreeDF = pd.DataFrame({'variable':df.columns[:9],
                        'importance':dt.feature_importances_})
```

```
4. Build the Decision tree model to predict Labels variable (20 points)



Accuracy scores for the DT test data:

0.7714285714285715

Feature importance for the decision tree:
[0.26254851 0.08868725 0.19333751 0.09422622 0.06407383 0.04526721
 0.0604528  0.10866843 0.08273824]
```

In [10]:
```python
#Make the tree plot with graphviz
from graphviz import Source

print("\nBuilding the decision tree... ")

#store the dot data
dot_data = tree.export_graphviz(dt, out_file=None,
                                feature_names=X_train.columns)

#Display the tree (does not work for me in Spyder)
Source(dot_data)
#Set filename
fname = "Breast_Cancer_DTree"
#Save the tree out to a pdf file
Source(dot_data).render(fname)

print("Decision tree has been built.  Check your folder for the "+
      str(fname) + ".pdf file.")
```

```
Building the decision tree...
Decision tree has been built.  Check your folder for the Breast_Cancer_DTree.pdf file.
```

In [11]:
```python
#5. Build Neural network model to predict Labels variable (20 points)
print("\n\n5. Build Neural network model to predict "+
      "Labels variable (20 points)\n")


#We must first rescale the data [0, 1]

#impor the MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

#Create the object
scaler = MinMaxScaler()

#Scale the train and test data
X_train_scaled = scaler.fit_transform(X_train, y=None)
X_test_scaled = scaler.transform(X_test)


#Note to professor: The model generation tab under the NN page states,
#"To import the DECISION TREE model, create... "

#Import the NN MLP Classifier
from sklearn.neural_network import MLPClassifier

#Make NN object, note we have a smaller data set so we'll use the
#solver = 'ldfgs' as it can converge faster and perform better
nn = MLPClassifier(solver='lbfgs',
                   alpha=1e-5,
                   hidden_layer_sizes=(10, 4),
                   verbose = False)



#Fit the model
#Note:  When I had the verbose=True for the nn object, I had a common
#report of bad direction in the line search
#despite this, I still had the greatest accuracy with the 'lbfgs' search
#compared to the 'adam' and 'sgd' searches
nn.fit(X_train_scaled, y_train)




#Run model evaluation
nn_pred = nn.predict(X_test_scaled)

#Accuracy scores
print("\n\nAccuracy scores for the NN test data:\n")
print(str(metrics.accuracy_score(y_test, nn_pred)))

#Store the test accuracy score for later
nn_test_acc_score = metrics.accuracy_score(y_test, nn_pred)

print("\nMetrics classificaiton report:\n")
print(metrics.classification_report(y_test, nn_pred))
```

```
5. Build Neural network model to predict Labels variable (20 points)



Accuracy scores for the NN test data:

0.8

Metrics classificaiton report:

              precision    recall  f1-score   support

           1       0.76      0.89      0.82        18
           2       0.86      0.71      0.77        17

    accuracy                           0.80        35
   macro avg       0.81      0.80      0.80        35
weighted avg       0.81      0.80      0.80        35
```

In [12]:
```python
#6. Which model is the best? Which variable is the most important one? (10 points)
print("6. Which model is the best?" +
      " Which variable is the most important one? (10 points)")

print("\nAccuracy Scores:")
print("Log Reg:        " + str(lr_test_acc_score))
print("Naive Bayes:    " + str(nb_test_acc_score))
print("Decision Tree:  " + str(dt_test_acc_score))
print("Neural Network: " + str(nn_test_acc_score))
```

```
6. Which model is the best? Which variable is the most important one? (10 points)

Accuracy Scores:
Log Reg:        0.7714285714285715
Naive Bayes:    0.6285714285714286
Decision Tree:  0.7714285714285715
Neural Network: 0.8
```

In [13]:
```python
#Based on the current DISPLAYED accuracy scores, I would feel best about the Neural Network classifier.
#The Log Reg beat out the Naive Bayes classifier as seen in the plot above
#and even more so as the training size increased.
#However, I have run trials where the Log Reg has the best accuracy.
#A more comprehensive averaging of the tests would make for a better comparison.

#I also have not tinkered with running various sets of hyperparmaters or using bagging/boosting
#for a random forest decision tree.  These methods could change these reults.
```

In [ ]:
```python
#As for the most important variable, the combination of the first correlation matrix and the numerous decision
#created from various runs of the data have led me to conclude that glucose is the most important variable.

#Glucose consistantly begins my decision trees with gini values above 0.480.  It does a good job at quicky
#splitting the data set and moving into other options.
```