Predicting Diabetes

Import Libraries

In [1]:
```
import pandas as pd                    # pandas is a dataframe library
import matplotlib.pyplot as plt        # matplotlib.pyplot plot data
import numpy as np                     # numpy provides N-dim object support

#  do ploting line instead of in a separate window
%matplotlib inline
```

Load and review data

In [2]:
```
df = pd.read_csv("./data/pima-data.csv")         # Load pima data , adjust path as necessary
```

In [3]:
```
df.shape        # it returns number or rows and columns , 768 rows and 10 columns
```

Out[3]:
```
(768, 10)
```

In [4]:
```
df.head(5)      # it returns 1st 5 data sets
```

Out[4]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	diabetes
0	6	148	72	35	0	33.6	0.627	50	1.3790	True
1	1	85	66	29	0	26.6	0.351	31	1.1426	False
2	8	183	64	0	0	23.3	0.672	32	0.0000	True
3	1	89	66	23	94	28.1	0.167	21	0.9062	False
4	0	137	40	35	168	43.1	2.288	33	1.3790	True

In [5]:
```
df.tail(5)     # it returns last 5 data set
```

Out[5]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	diabetes
763	10	101	76	48	180	32.9	0.171	63	1.8912	False
764	2	122	70	27	0	36.8	0.340	27	1.0638	False
765	5	121	72	23	112	26.2	0.245	30	0.9062	False
766	1	126	60	0	0	30.1	0.349	47	0.0000	True
767	1	93	70	31	0	30.4	0.315	23	1.2214	False

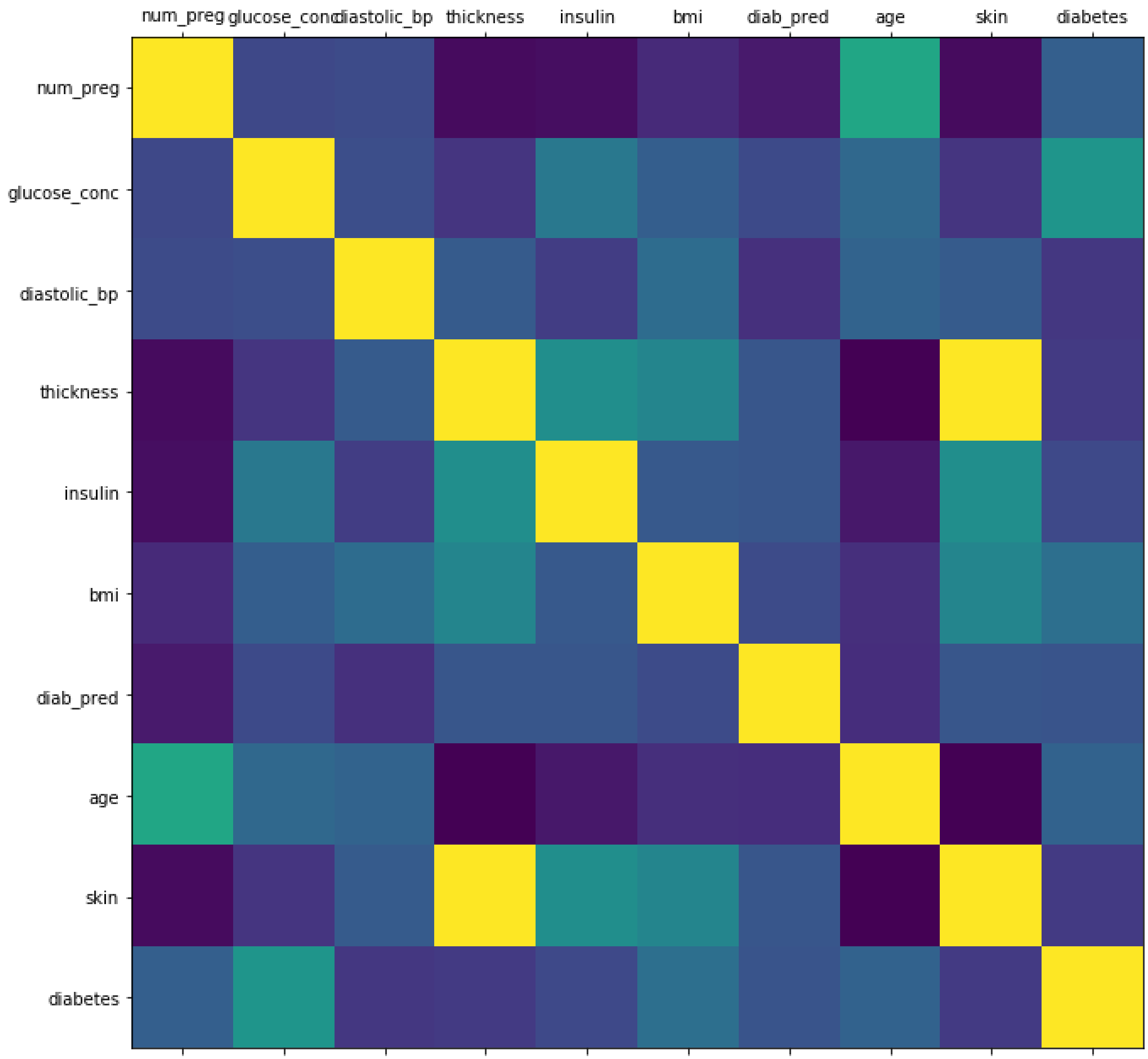
Cleaning the Data : Check for null values in data frame

In [6]:
```
df.isnull().values.any()
```

Out[6]:
```
False
```

In [7]:
```
def plot_corr(df, size=11):
    """
    function plots a graphical coreleastion matrix for each pair of column in different dataframe .

    Input :
       df : pandas dataframe
       size : vertical and horizontal size of the plot

    Display :
       matrix of corelation between columns.
         Blue-cyan-yellow  => less to more coreated
         0 -------------------> 1
         Expect a darked light running from top left to bottom right
    """

    corr = df.corr()  # data frame corelation function
    fig , ax = plt.subplots(figsize =(size,size))
    ax.matshow(corr)  # color code the rectanges by corelation value
    plt.xticks (range(len(corr.columns)), corr.columns)    # draw x ticks mark
    plt.yticks (range(len(corr.columns)), corr.columns)    # draw y ticks mark
```

In [8]:
```
plot_corr(df)
```



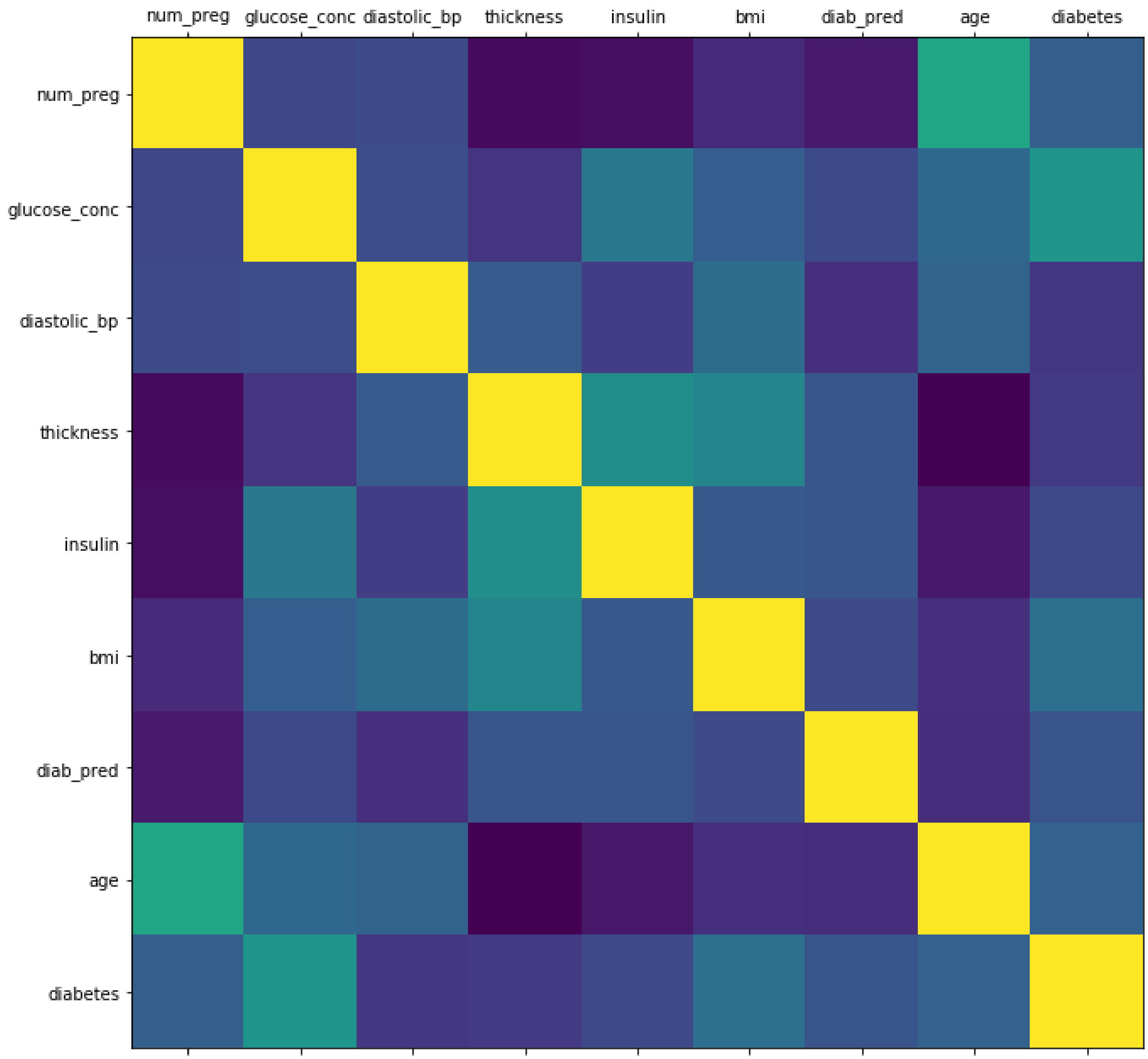
In [9]:
```
df.corr()
```

Out[9]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	diabe
num_preg	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	-0.081672	0.221
glucose_conc	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.057328	0.466
diastolic_bp	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.207371	0.065
thickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	1.000000	0.074
insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.436783	0.130
bmi	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.392573	0.292
diab_pred	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.183928	0.173
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	-0.113970	0.238
skin	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	1.000000	0.074
diabetes	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	0.074752	1.000

In [10]:
```
del df['skin'] # remove the skin row corelated row
```

In [11]:
```
plot_corr(df)
```



In [12]:
```
df.head()
```

Out[12]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	True
1	1	85	66	29	0	26.6	0.351	31	False
2	8	183	64	0	0	23.3	0.672	32	True
3	1	89	66	23	94	28.1	0.167	21	False
4	0	137	40	35	168	43.1	2.288	33	True

Modeling the data : Check data types

Here we have Bool value in diabetes column , Need to change to int True to 1 and False to 0

In [13]:
```
diabetes_map = {True:1 , False:0}
```

In [14]:
```
df['diabetes'] = df['diabetes'].map(diabetes_map)
```

In [15]:
```
df.head(5)
```

Out[15]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
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

Check true false ratio

In [16]:
```
num_true = len(df.loc[df.diabetes == True])
num_false = len(df.loc[df.diabetes == False])
print ("Number of true cases: {0} ({1:2.2f}%)".format(num_true, (num_true / (num_true + num_false))*1
00))
print ("Number of false cases: {0} ({1:2.2f}%)".format(num_false, (num_false / (num_true +
num_false))*100))
```

```
Number of true cases: 268 (34.90%)
Number of false cases: 500 (65.10%)
```

Splitting the data

70% for training , 30% for testing

In [17]:
```
from sklearn.cross_validation import train_test_split

feature_col_names = ['num_preg', 'glucose_conc' , 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'dia
b_pred', 'age']
predicted_class_names = ['diabetes']

X = df[feature_col_names].values # predictor feature coloumns ( 8 X m )
y = df[predicted_class_names].values # predicted class ( 1= true , 0=false ) column ( 1 X m )
split_test_size = 0.30

X_train , X_test , y_train , y_test = train_test_split(X, y, test_size = split_test_size, random_stat
e = 42)
          # test size = 0.30 is 30% , 42 is the answer to everything , any number can be used for ran
dom state
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/sklearn/cross_validati
on.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selec
tion module into which all the refactored classes and functions are moved. Also note that the interf
ace of the new CV iterators are different from that of this module. This module will be removed in
 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [18]:
```
print("{0:0.2f}% in training set".format((len(X_train)/len(df.index))*100))
print("{0:0.2f}% in test set".format((len(X_test)/len(df.index))*100))
```

```
69.92% in training set
30.08% in test set
```

verifying predicted value was split correctly

In [19]:
```
print ("Origional True: {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 1]),(len(df.loc[df['diab
etes'] == 1]) / len(df.index))*100.0))
print ("Origional False: {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 0]),(len(df.loc[df['dia
betes'] == 0]) / len(df.index))*100.0))
print(" ")
print ("Training True: {0} ({1:0.2f}%)".format(len(y_train[y_train[:] == 1]),(len(y_train[y_train[:]
== 1]) / len(y_train))*100.0))
print ("Training False: {0} ({1:0.2f}%)".format(len(y_train[y_train[:] == 0]),(len(y_train[y_train[:]
 == 0]) / len(y_train))*100.0))
print(" ")
print ("Testing True: {0} ({1:0.2f}%)".format(len(y_test[y_test[:] == 1]),(len(y_test[y_test[:] ==
1]) / len(y_test))*100.0))
print ("Testing False: {0} ({1:0.2f}%)".format(len(y_test[y_test[:] == 0]),(len(y_test[y_test[:] ==
0]) / len(y_test))*100.0))
```

```
Origional True: 268 (34.90%)
Origional False: 500 (65.10%)

Training True: 188 (35.01%)
Training False: 349 (64.99%)

Testing True: 80 (34.63%)
Testing False: 151 (65.37%)
```

Post Split data preparation

hidden missing values , we already check the null values , but still their might be some 0 values .. like in thinkness. Are the 0 values possible ? we need to find out these unexpected 0 values.

In [20]:
```
print("# rows in dataframe {0}".format(len(df)))
print("# rows missing glucose_conc: {0}".format(len(df.loc[df['glucose_conc'] == 0 ])))
print("# rows missing diastolic_bp: {0}".format(len(df.loc[df['diastolic_bp'] == 0 ])))
print("# rows missing thickness: {0}".format(len(df.loc[df['thickness'] == 0 ])))
print("# rows missing insulin: {0}".format(len(df.loc[df['insulin'] == 0 ])))
print("# rows missing bmi: {0}".format(len(df.loc[df['bmi'] == 0 ])))
print("# rows missing diab_pred: {0}".format(len(df.loc[df['diab_pred'] == 0 ])))
print("# rows missing age: {0}".format(len(df.loc[df['age'] == 0 ])))
```

```
# rows in dataframe 768
# rows missing glucose_conc: 5
# rows missing diastolic_bp: 35
# rows missing thickness: 227
# rows missing insulin: 374
# rows missing bmi: 11
# rows missing diab_pred: 0
# rows missing age: 0
```

In [21]:
```
from sklearn.preprocessing import Imputer
#impute with mean all 0 readings

fill_0 = Imputer(missing_values = 0 , strategy ="mean", axis=0)

X_train = fill_0.fit_transform(X_train)
X_test = fill_0.fit_transform(X_test)
```

Naive Bayes algorithim

In [22]:
```
from sklearn.naive_bayes import GaussianNB

# Create Gaussian naive bayes model object and train it with the data
nb_model = GaussianNB()
nb_model.fit(X_train, y_train.ravel())
```

Out[22]:
```
GaussianNB(priors=None)
```

Performance on Training Data

In [23]:
```
# predict values using the Training data
nb_predict_train = nb_model.predict(X_train)

# import the performance metrices library
from sklearn import metrics

# Accuracy
print("Accuracy : {0:.4f}".format(metrics.accuracy_score(y_train, nb_predict_train)))
print()
```

```
Accuracy : 0.7542
```

In [24]:
```
# predict values using the Texting data
nb_predict_text = nb_model.predict(X_test)

# import the performance metrices library
from sklearn import metrics

# Accuracy
print("Accuracy : {0:.4f}".format(metrics.accuracy_score(y_test, nb_predict_text)))
print()
```

```
Accuracy : 0.7359
```

In [25]:
```
print("Confusion Matrix")

# the use of labels to set 1=True to upper left and 0=False to lower right

print("{0}".format(metrics.confusion_matrix(y_test, nb_predict_text, labels = [1,0])))
print(" ")

print(" Classification Report ")
print(metrics.classification_report(y_test, nb_predict_text, labels = [1,0]))


# * left column : Predictive true
#   right column : predictive false

#   top row : actual true
#   bottom row : actual false

#   TP FP
#   FN TN

#   Perfect classifier :
#    TP = 80
#    FP = 0
#    FN = 0
#    TN = 151

#    In Classification matrix ::
#    recall = TP / [TP+FN] >= 70%
#    precision = TP / [TP+FP] >= 70%
```

```
Confusion Matrix
[[ 52  28]
 [ 33 118]]

 Classification Report
             precision    recall  f1-score   support

          1       0.61      0.65      0.63        80
          0       0.81      0.78      0.79       151

avg / total       0.74      0.74      0.74       231
```

Random Forest algorithim

In [26]:
```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(random_state = 42) # create random forest object
rf_model.fit(X_train, y_train.ravel())
```

Out[26]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=42, verbose=0, warm_start=False)
```

Predicting Training data

In [27]:
```
rf_predict_train = rf_model.predict(X_train)
# training metrics
print("Accuracy : {0:.4f}".format(metrics.accuracy_score(y_train, rf_predict_train)))
print()
```

```
Accuracy : 0.9870
```

Predicting Test data

In [28]:
```
rf_predict_test = rf_model.predict(X_test)
# training metrics
print("Accuracy : {0:.4f}".format(metrics.accuracy_score(y_test, rf_predict_test)))
print()
```

```
Accuracy : 0.7100
```

In [29]:
```
print("Confusion Matrix")

# the use of labels to set 1=True to upper left and 0=False to lower right

print("{0}".format(metrics.confusion_matrix(y_test, rf_predict_test, labels = [1,0])))
print(" ")

print(" Classification Report ")
print(metrics.classification_report(y_test, rf_predict_test, labels = [1,0]))
```

```
Confusion Matrix
[[ 43  37]
 [ 30 121]]

 Classification Report
             precision    recall  f1-score   support

          1       0.59      0.54      0.56        80
          0       0.77      0.80      0.78       151

avg / total       0.70      0.71      0.71       231
```

Random Forest performs well with Training data , but seems to be low performance with Text data

Logistic Regression

In [30]:
```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(C=0.7, random_state=42) # create random forest object
lr_model.fit(X_train, y_train.ravel())
```

Out[30]:
```
LogisticRegression(C=0.7, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=42, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [31]:
```
lr_predict_test = lr_model.predict(X_test)
# training metrics
print("Accuracy : {0:.4f}".format(metrics.accuracy_score(y_test, lr_predict_test)))
print()
```

```
Accuracy : 0.7446
```

In [32]:
```
print("Confusion Matrix")

# the use of labels to set 1=True to upper left and 0=False to lower right

print("{0}".format(metrics.confusion_matrix(y_test, lr_predict_test, labels = [1,0])))
print(" ")

print(" Classification Report ")
print(metrics.classification_report(y_test, lr_predict_test, labels = [1,0]))
```

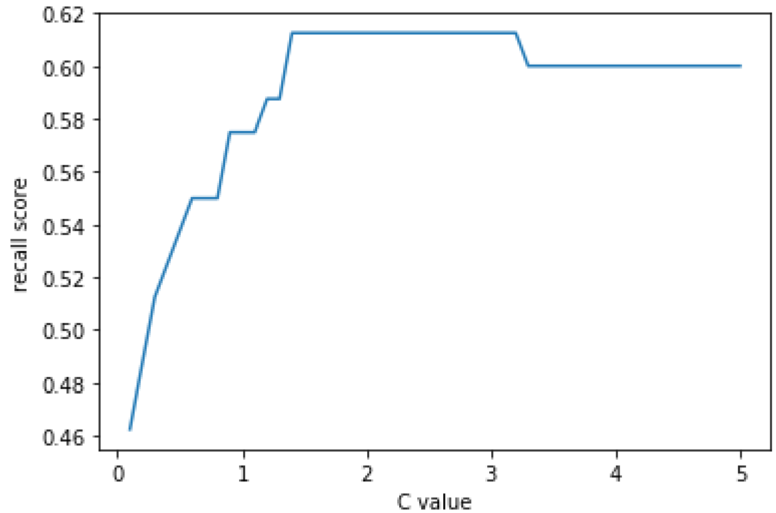
```
Confusion Matrix
[[ 44  36]
 [ 23 128]]

 Classification Report
             precision    recall  f1-score   support

          1       0.66      0.55      0.60        80
          0       0.78      0.85      0.81       151

avg / total       0.74      0.74      0.74       231
```

Setting Regularziation Parameter

In [33]:
```
C_start = 0.1
C_end = 5
C_inc = 0.1


C_values, recall_scores = [] , []
C_val = C_start
best_recall_score = 0

while (C_val < C_end):
    C_values.append(C_val)
    lr_model_loop = LogisticRegression(C=C_val , random_state=42)
    lr_model_loop.fit(X_train , y_train.ravel())
    lr_predict_loop_test = lr_model_loop.predict(X_test)
    recall_score = metrics.recall_score(y_test, lr_predict_loop_test)
    recall_scores.append(recall_score)
    if (recall_score > best_recall_score):
        best_recall_score = recall_score
        best_lr_predict_test = lr_predict_loop_test

    C_val = C_val + C_inc

best_score_C_val = C_values[recall_scores.index(best_recall_score)]
print("1st max value of {0:.3f} occured at c={1:.3f}".format(best_recall_score,best_score_C_val))

%matplotlib inline
plt.plot(C_values , recall_scores, "-")
plt.xlabel("C value")
plt.ylabel("recall score")
```

```
1st max value of 0.613 occured at c=1.400
```

Out[33]: <matplotlib.text.Text at 0x10c53c8d0>



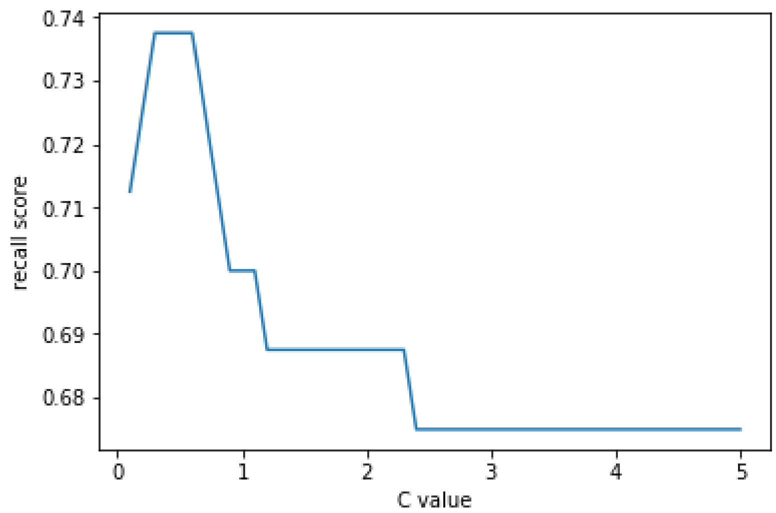
We have inbalace data , having more non diabetics results than diabetics . This is casuing an issue so mac value is not exceeding mote than 0.70

Logisitic Regression with class_weight = 'balanced'

In [34]:
```
C_start = 0.1
C_end = 5
C_inc = 0.1


C_values, recall_scores = [] , []
C_val = C_start
best_recall_score = 0

while (C_val < C_end):
    C_values.append(C_val)
    lr_model_loop = LogisticRegression(C=C_val , class_weight='balanced', random_state=42)
    lr_model_loop.fit(X_train , y_train.ravel())
    lr_predict_loop_test = lr_model_loop.predict(X_test)
    recall_score = metrics.recall_score(y_test, lr_predict_loop_test)
    recall_scores.append(recall_score)
    if (recall_score > best_recall_score):
        best_recall_score = recall_score
        best_lr_predict_test = lr_predict_loop_test

    C_val = C_val + C_inc

best_score_C_val = C_values[recall_scores.index(best_recall_score)]
print("1st max value of {0:.3f} occured at c={1:.3f}".format(best_recall_score,best_score_C_val))

%matplotlib inline
plt.plot(C_values , recall_scores, "-")
plt.xlabel("C value")
plt.ylabel("recall score")
```

```
1st max value of 0.738 occured at c=0.300
```

Out[34]: <matplotlib.text.Text at 0x10c5d2b38>



In [35]:
```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(class_weight='balanced' , C=best_score_C_val, random_state=42)
lr_model.fit(X_train, y_train.ravel())
```

Out[35]:
```
LogisticRegression(C=0.30000000000000004, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=42,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [36]:
```
lr_predict_test = lr_model.predict(X_test)
# training metrics
print("Accuracy : {0:.4f}".format(metrics.accuracy_score(y_test, lr_predict_test)))
print()
```

```
Accuracy : 0.7143
```

In [37]:
```
print("Confusion Matrix")

# the use of labels to set 1=True to upper left and 0=False to lower right

print("{0}".format(metrics.confusion_matrix(y_test, lr_predict_test, labels = [1,0])))
print(" ")

print(" Classification Report ")
print(metrics.classification_report(y_test, lr_predict_test, labels = [1,0]))
```

```
Confusion Matrix
[[ 59  21]
 [ 45 106]]

 Classification Report
             precision    recall  f1-score   support

          1       0.57      0.74      0.64        80
          0       0.83      0.70      0.76       151

avg / total       0.74      0.71      0.72       231
```

LogisticRegression CrossValidation

In [38]:
```
from sklearn.linear_model import LogisticRegressionCV
lr_cv_model = LogisticRegressionCV(n_jobs=-1, random_state=42, Cs=3, cv=10, refit=True,
class_weight='balanced')
lr_cv_model.fit(X_train, y_train.ravel())
```

Out[38]:
```
LogisticRegressionCV(Cs=3, class_weight='balanced', cv=10, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=-1, penalty='l2', random_state=42,
           refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
```

Predict on Test data

In [39]:
```
lr_cv_predict_test = lr_cv_model.predict(X_test)
# training metrics
print("Accuracy : {0:.4f}".format(metrics.accuracy_score(y_test, lr_cv_predict_test)))
print()
```

```
Accuracy : 0.6970
```

In [40]:
```
print("Confusion Matrix")

# the use of labels to set 1=True to upper left and 0=False to lower right

print("{0}".format(metrics.confusion_matrix(y_test, lr_cv_predict_test, labels = [1,0])))
print(" ")

print(" Classification Report ")
print(metrics.classification_report(y_test, lr_cv_predict_test, labels = [1,0]))
```

```
Confusion Matrix
[[ 53  27]
 [ 43 108]]

 Classification Report
             precision    recall  f1-score   support

          1       0.55      0.66      0.60        80
          0       0.80      0.72      0.76       151

avg / total       0.71      0.70      0.70       231
```