In [1]:

```
# To enable plotting graphs in Jupyter notebook
%matplotlib inline
```

In [2]:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

# importing ploting libraries
import matplotlib.pyplot as plt

#importing seaborn for statistical plots
import seaborn as sns

#Let us break the X and y dataframes into training set and test set. For this we will use
#Sklearn package's data splitting function which is based on random function

from sklearn.model_selection import train_test_split

import numpy as np


# calculate accuracy measures and confusion matrix
from sklearn import metrics
```

In [3]:

```
# The data lies in the following URL.
#url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/p
ima-indians-diabetes.data"
```

In [4]:

```
# Since it is a data file with no header, we will supply the column names which have been
obtained from the above URL
# Create a python list of column names called "names"

colnames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

#Load the file from local directory using pd.read_csv which is a special form of read_tab
le
#while reading the data, supply the "colnames" list

pima_df = pd.read_csv("pima-indians-diabetes.data", names= colnames)
```

In [5]:

```
pima_df.head(50)
```

Out[5]:

	preg	plas	pres	skin	test	mass	pedi	age	class
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
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0

	preg	plas	pres	skin	test	mass	pedi	age	class
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1
10	4	110	92	0	0	37.6	0.191	30	0
11	10	168	74	0	0	38.0	0.537	34	1
12	10	139	80	0	0	27.1	1.441	57	0
13	1	189	60	23	846	30.1	0.398	59	1
14	5	166	72	19	175	25.8	0.587	51	1
15	7	100	0	0	0	30.0	0.484	32	1
16	0	118	84	47	230	45.8	0.551	31	1
17	7	107	74	0	0	29.6	0.254	31	1
18	1	103	30	38	83	43.3	0.183	33	0
19	1	115	70	30	96	34.6	0.529	32	1
20	3	126	88	41	235	39.3	0.704	27	0
21	8	99	84	0	0	35.4	0.388	50	0
22	7	196	90	0	0	39.8	0.451	41	1
23	9	119	80	35	0	29.0	0.263	29	1
24	11	143	94	33	146	36.6	0.254	51	1
25	10	125	70	26	115	31.1	0.205	41	1
26	7	147	76	0	0	39.4	0.257	43	1
27	1	97	66	15	140	23.2	0.487	22	0
28	13	145	82	19	110	22.2	0.245	57	0
29	5	117	92	0	0	34.1	0.337	38	0
30	5	109	75	26	0	36.0	0.546	60	0
31	3	158	76	36	245	31.6	0.851	28	1
32	3	88	58	11	54	24.8	0.267	22	0
33	6	92	92	0	0	19.9	0.188	28	0
34	10	122	78	31	0	27.6	0.512	45	0
35	4	103	60	33	192	24.0	0.966	33	0
36	11	138	76	0	0	33.2	0.420	35	0
37	9	102	76	37	0	32.9	0.665	46	1
38	2	90	68	42	0	38.2	0.503	27	1
39	4	111	72	47	207	37.1	1.390	56	1
40	3	180	64	25	70	34.0	0.271	26	0
41	7	133	84	0	0	40.2	0.696	37	0
42	7	106	92	18	0	22.7	0.235	48	0
43	9	171	110	24	240	45.4	0.721	54	1
44	7	159	64	0	0	27.4	0.294	40	0
45	0	180	66	39	0	42.0	1.893	25	1
46	1	146	56	0	0	29.7	0.564	29	0
47	2	71	70	27	0	28.0	0.586	22	0
48	7	103	66	32	0	39.1	0.344	31	1
49	7	105	0	0	0	0.0	0.305	24	0

In [6]:

```
# Let us check whether any of the columns has any value other than numeric i.e. data is n
ot corrupted such as a "?" instead of
```

```
# a number.

# we use np.isreal a numpy function which checks each column for each row and returns a b
ool array,
# where True if input element is real.
# applymap is pandas dataframe function that applies the np.isreal function columnwise
# Following line selects those rows which have some non-numeric value in any of the colum
ns hence the  ~ symbol

pima_df[~pima_df.applymap(np.isreal).all(1)]
```

Out[6]:

preg	plas	pres	skin	test	mass	pedi	age	class

In [7]:

```
# replace the missing values in pima_df with median value :Note, we do not need to specif
y the column names
# every column's missing value is replaced with that column's median respectively
#pima_df = pima_df.fillna(pima_df.median())
#pima_df
```

In [8]:

```
#Lets analysze the distribution of the various attributes
pima_df.describe().transpose()
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
preg	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
plas	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
pres	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
skin	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
test	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
mass	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
pedi	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
class	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

In [9]:

```
# Let us look at the target column which is 'class' to understand how the data is distrib
uted amongst the various values
pima_df.groupby(["class"]).count()

# Most are not diabetic. The ratio is almost 1:2 in favor or class 0.  The model's abilit
y to predict class 0 will
# be better than predicting class 1.
```

Out[9]:

class	preg	plas	pres	skin	test	mass	pedi	age
0	500	500	500	500	500	500	500	500
1	268	268	268	268	268	268	268	268

In [10]:

```
# Let us do a correlation analysis among the different dimensions and also each dimension
with the dependent dimension
```

```
# This is done using scatter matrix function which creates a dashboard reflecting useful
information about the dimensions
# The result can be stored as a .png file and opened in say, paint to get a larger view

#pima_df_attr = pima_df.iloc[:,0:9]

#axes = pd.plotting.scatter_matrix(pima_df_attr)
#plt.tight_layout()
#plt.savefig('d:\greatlakes\pima_pairpanel.png')
```

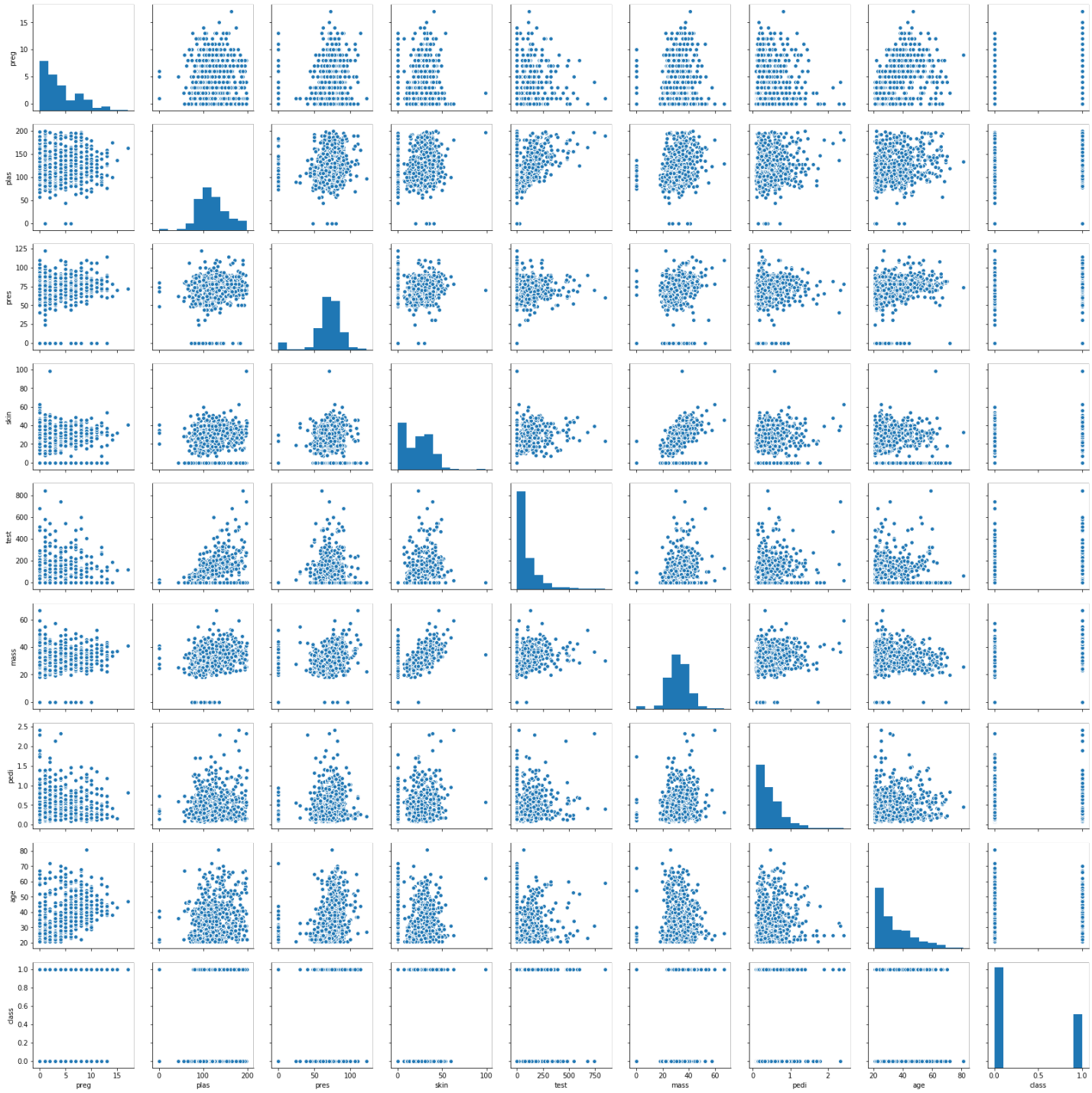
In [11]:

```
# Pairplot using sns

sns.pairplot(pima_df)
```

Out[11]:

```
<seaborn.axisgrid.PairGrid at 0x23e2e0a8898>
```



In [11]:

```
#data for all the attributes are skewed, especially for the variable "test"
```

```
#The mean for test is 80(rounded) while the median is 30.5 which clearly indicates an ext
reme long tail on the right
```

In [12]:

```
# Attributes which look normally distributed (plas, pres, skin, and mass).
# Some of the attributes look like they may have an exponential distribution (preg, test,
pedi, age).
# Age should probably have a normal distribution, the constraints on the data collection
may have skewed the distribution.

# There is no obvious relationship between age and onset of diabetes.
# There is no obvious relationship between pedi function and onset of diabetes.
```

In [13]:

```
array = pima_df.values
X = pima_df.iloc[:,0:8]
y = pima_df.iloc[:,8]
#X = array[:,0:8] # select all rows and first 8 columns which are the attributes
#Y = array[:,8]   # select all rows and the 8th column which is the classification "Yes",
"No" for diabeties
test_size = 0.30 # taking 70:30 training and test set
seed =1 # Random numbmer seeding for reapeatability of the code
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_st
ate=seed)
```

In [19]:

```
# Fit the model on 30%
model = LogisticRegression()
model.fit(X_train, y_train)
y_predict = model.predict(X_test)



coef_df = pd.DataFrame(model.coef_, columns= t)
coef_df['intercept'] = model.intercept_
print(coef_df)
```

```
       preg      plas      pres      skin      test      mass      pedi  \
0  0.094378  0.025543 -0.019857 -0.001549 -0.00007  0.056306  0.389516

        age  intercept
0  0.008663  -5.058842
```

In [20]:

```
model_score = model.score(X_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_predict))
```

```
0.7748917748917749
[[132  14]
 [ 38  47]]
```

In [21]:

```
# Improve the model ------------------------------Iteration 2 -------------------------
----------------------
```

In [22]:

```
# To scale the dimensions we need scale function which is part of sckikit preprocessing l
ibraries

from sklearn import preprocessing

# scale all the columns of the mpg_df. This will produce a numpy array
#pima_df_scaled = preprocessing.scale(pima_df[0:7])
X_train_scaled = preprocessing.scale(X_train)
```

```
X_test_scaled = preprocessing.scale(X_test)
```

In [23]:

```
# Fit the model on 30%
model = LogisticRegression()
model.fit(X_train_scaled, y_train)
y_predict = model.predict(X_test_scaled)
model_score = model.score(X_test_scaled, y_test)
print(model_score)


# IMPORTANT: first argument is true values, second argument is predicted values
# this produces a 2x2 numpy array (matrix)
print(metrics.confusion_matrix(y_test, y_predict))
```

```
0.7792207792207793
[[132  14]
 [ 37  48]]
```

Analyzing the confusion matrix

True Positives (TP): we correctly predicted that they do have diabetes 46

True Negatives (TN): we correctly predicted that they don't have diabetes 134

False Positives (FP): we incorrectly predicted that they do have diabetes (a "Type I error") 13 Falsely predict positive Type I error

False Negatives (FN): we incorrectly predicted that they don't have diabetes (a "Type II error") 38 Falsely predict negative Type II error

In [ ]:

```

```