Clase EDA y modelamiento b?sico usando datos de inicio de diabetes en los indios Pima

July 11, 2017

In [72]:
```
# Evaluate using Cross Validation
import pandas as pd
from pandas import read_csv
import numpy as np
import matplotlib as plt
get_ipython().magic(u'matplotlib inline')
get_ipython().magic(u"config InlineBackend.figure_format='retina'")
import seaborn as sbs
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
```

In [2]:
```
#El fragmento siguiente carga el conjunto de datos de inicio de diabetes de los indios
#Link a los datos https://archive.ics.uci.edu/ml/datasets/pima+indians+diabetes
url = "https://goo.gl/vhm1eU"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
dataframe.head()
```

Out[2]:

	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

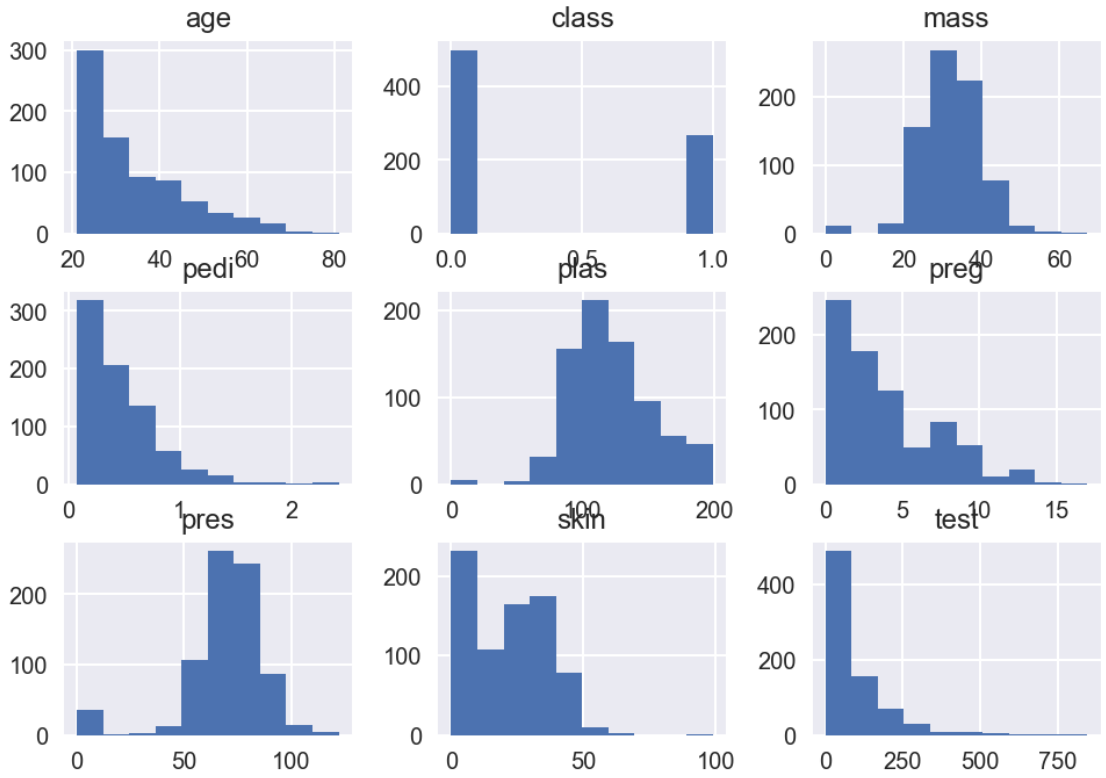
In [3]:
```
dataframe.shape
```

Out[3]: (768, 9)

In [4]:
```
#Comenzamos el análisis univariante gráficando un histograma para entender las distrib
#de cada variable usando visualización. (solo sirve para platear hipótesis por si solo
```

In [5]:
```
#Pareciera ser que estamos en presencia de distribuciones normales, normales con sesgo
#y exponencial. Veremos si esto efectivamente es así más adelante
dataframe.hist()
```

Out[5]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x11c2103c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11ff16978>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11ff935c0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x11ffead30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1200262b0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x120026160>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x1200c44e0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x120182860>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1201df668>]], dtype=object)
```



In [6]:
```
#Nos entrega un resumen estadístico rápido de cada una de las variables (deben ser num
dataframe.describe()
```

Out[6]:

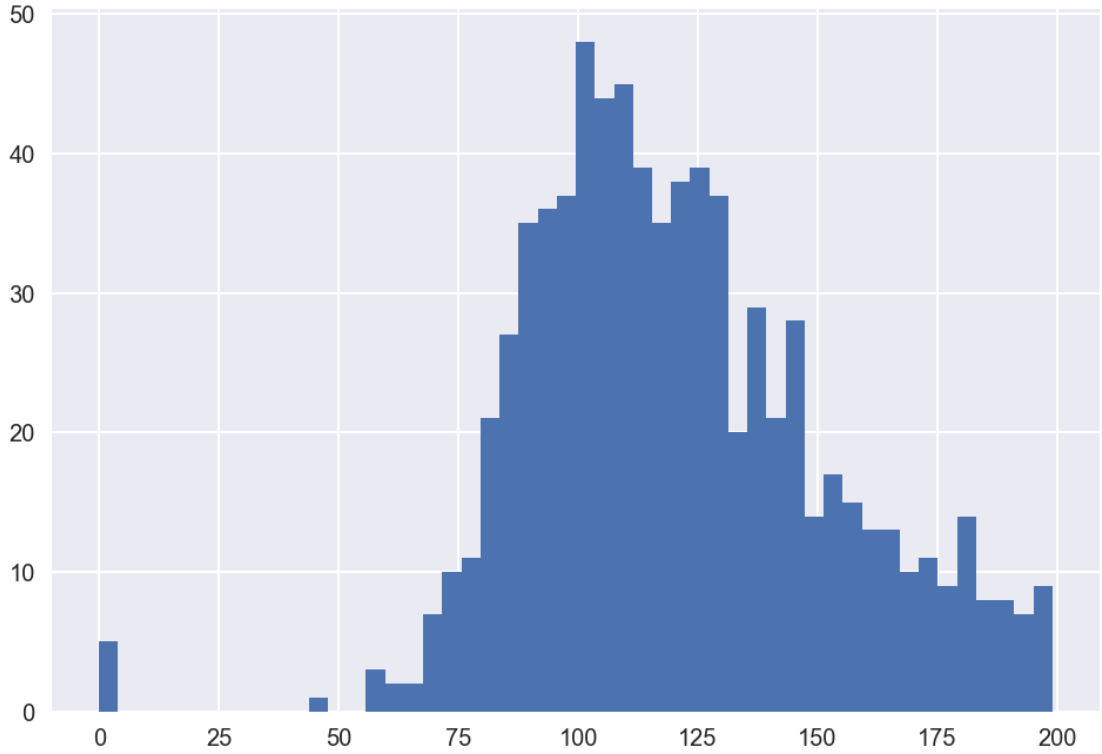
	preg	plas	pres	skin	test	mass	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

```
              pedi         age       class
count   768.000000  768.000000  768.000000
mean      0.471876   33.240885    0.348958
std       0.331329   11.760232    0.476951
min       0.078000   21.000000    0.000000
25%       0.243750   24.000000    0.000000
50%       0.372500   29.000000    0.000000
75%       0.626250   41.000000    1.000000
max       2.420000   81.000000    1.000000
```

In [7]:
```
#COmenzamos con el análisis univariante y multivariante con respectoa la variable targ
#análisis multivariante entre las disitinstas variables explicativas.
```

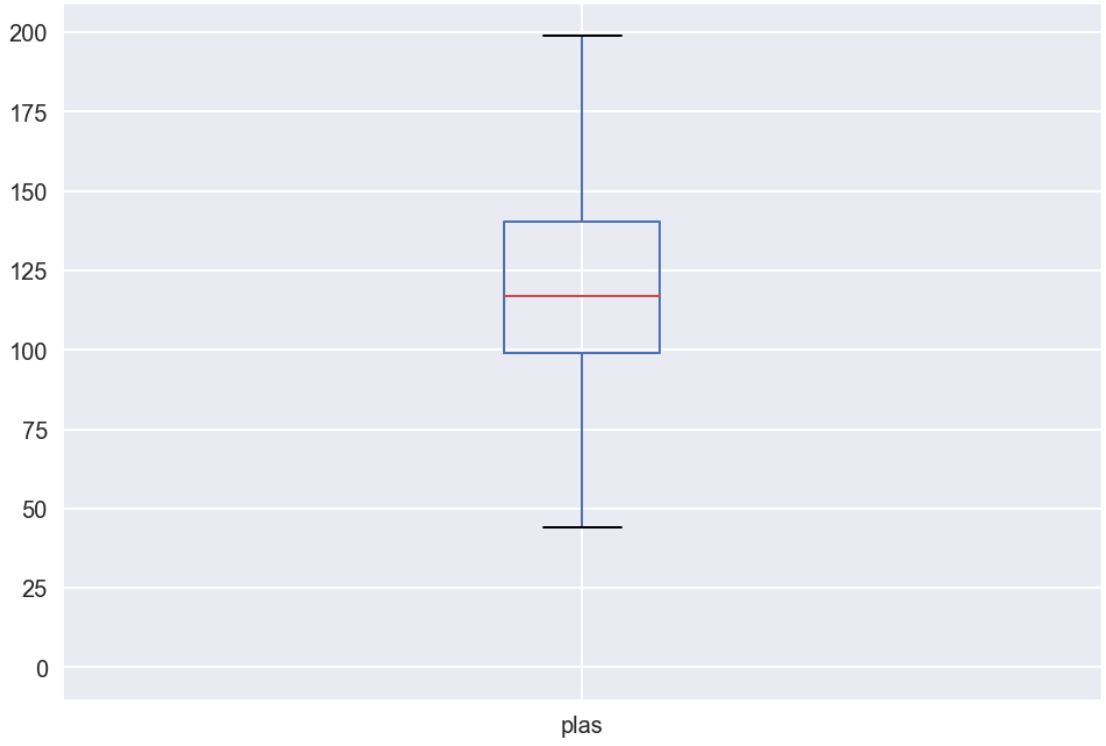
In [8]:
```
#1.-Análsis de la variable plas.
#Variable númerica con valores entre 0 y 199 que representa la concentración de plasma
#tolerancia a la glucosa oral.
#Con respectoa a la distribución de la variable, podemos decir que se ve como una norm
#De hecho, su media es casi igual a la mediana y el la media se encuentra 20% más cerc
#de 199.Adicionalmente,podemos ver muy pocos valores entre 0 y 99 (solo un 25%), mient
#de los valores se encuentra en el intervalo cercano al máximo del rango.
dataframe.plas.hist(bins=50)
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12086a5f8>
```



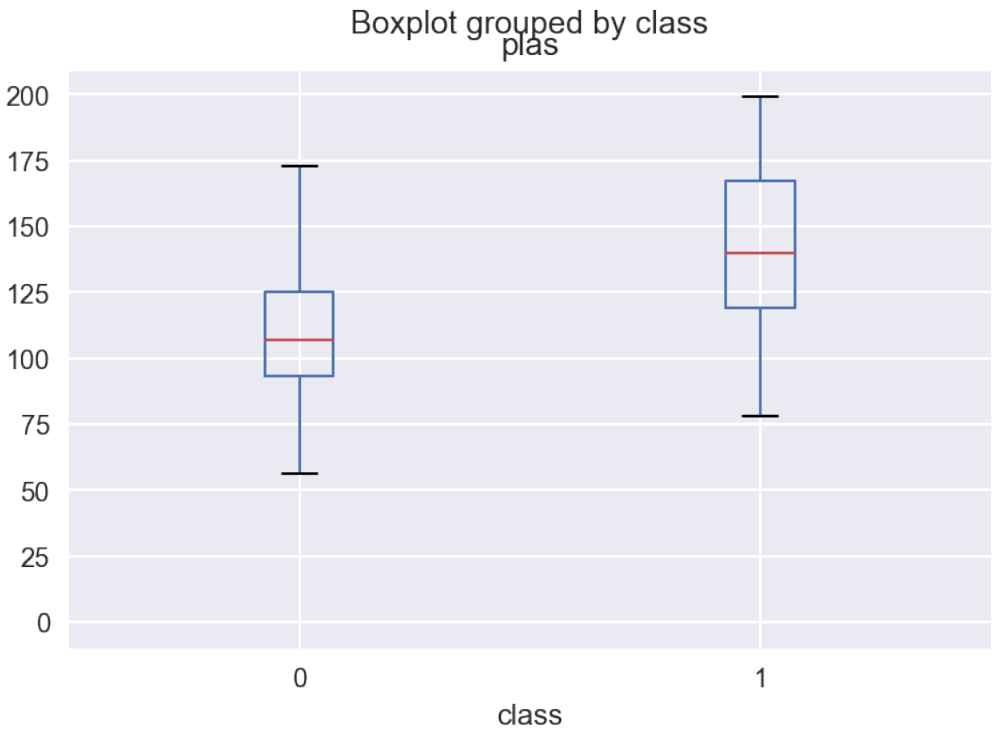
In [9]:
```
#El boxplot nos confirma lo que enunciamos anteriormente, que el 50% de la data se enc
dataframe.boxplot(column='plas')
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x120c5d978>
```



In [84]:
```
#Adicionalmente podemos ver que esta tendencia se mantiene cuando revisamos el boxplo
#ver también que enla clase 0 el 50% de los datos centrales están contenidos en un ra
#que es equivalente a decir que se espera una mayor desviación de la media para los v
#la 1, tal y como podemos observar en el cuadro que mostramos más abajo con las medid
#por clase
dataframe.boxplot(column='plas',by='class')
```

Out[84]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x123413b38>
```



In [11]:
```
dataframe.groupby('class')['plas'].describe()
```

Out[11]:

class	count	mean	std	min	25%	50%	75%	max
0	500.0	109.980000	26.141200	0.0	93.0	107.0	125.0	197.0
1	268.0	141.257463	31.939622	0.0	119.0	140.0	167.0	199.0

In [12]:
```
#Comenzamos el análisis multivariante entre las variables usando las correlaciones de
#Es importante notar que las correlaciones de Pearson son solo válidas entre variable
#cuya naturaleza sea categórica, como preg o class. Si bien sus valores están expresa
#presentan una categoría y no un valor número.
dataframe.corr()
```

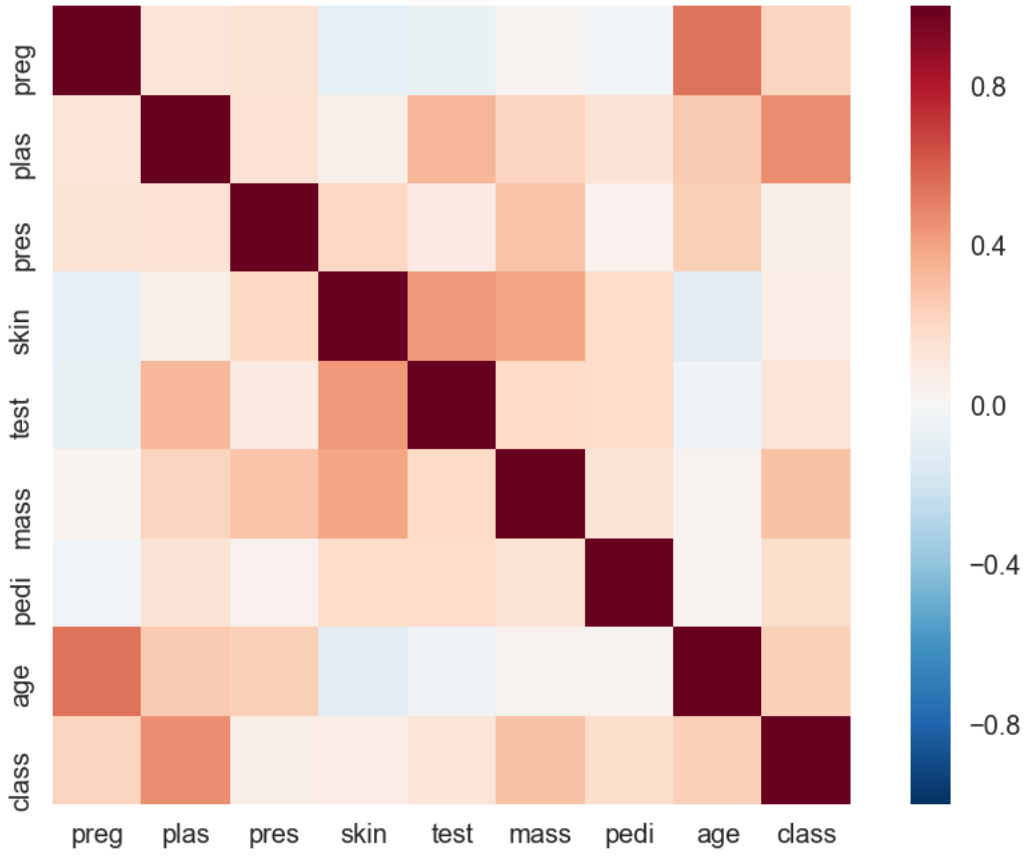
Out[12]:

	preg	plas	pres	skin	test	mass	pedi	\
preg	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	
plas	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	
pres	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	
skin	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	
test	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	
mass	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	
pedi	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	
class	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	

```
              age     class
preg     0.544341  0.221898
plas     0.263514  0.466581
pres     0.239528  0.065068
skin    -0.113970  0.074752
test    -0.042163  0.130548
mass     0.036242  0.292695
pedi     0.033561  0.173844
age      1.000000  0.238356
class    0.238356  1.000000
```

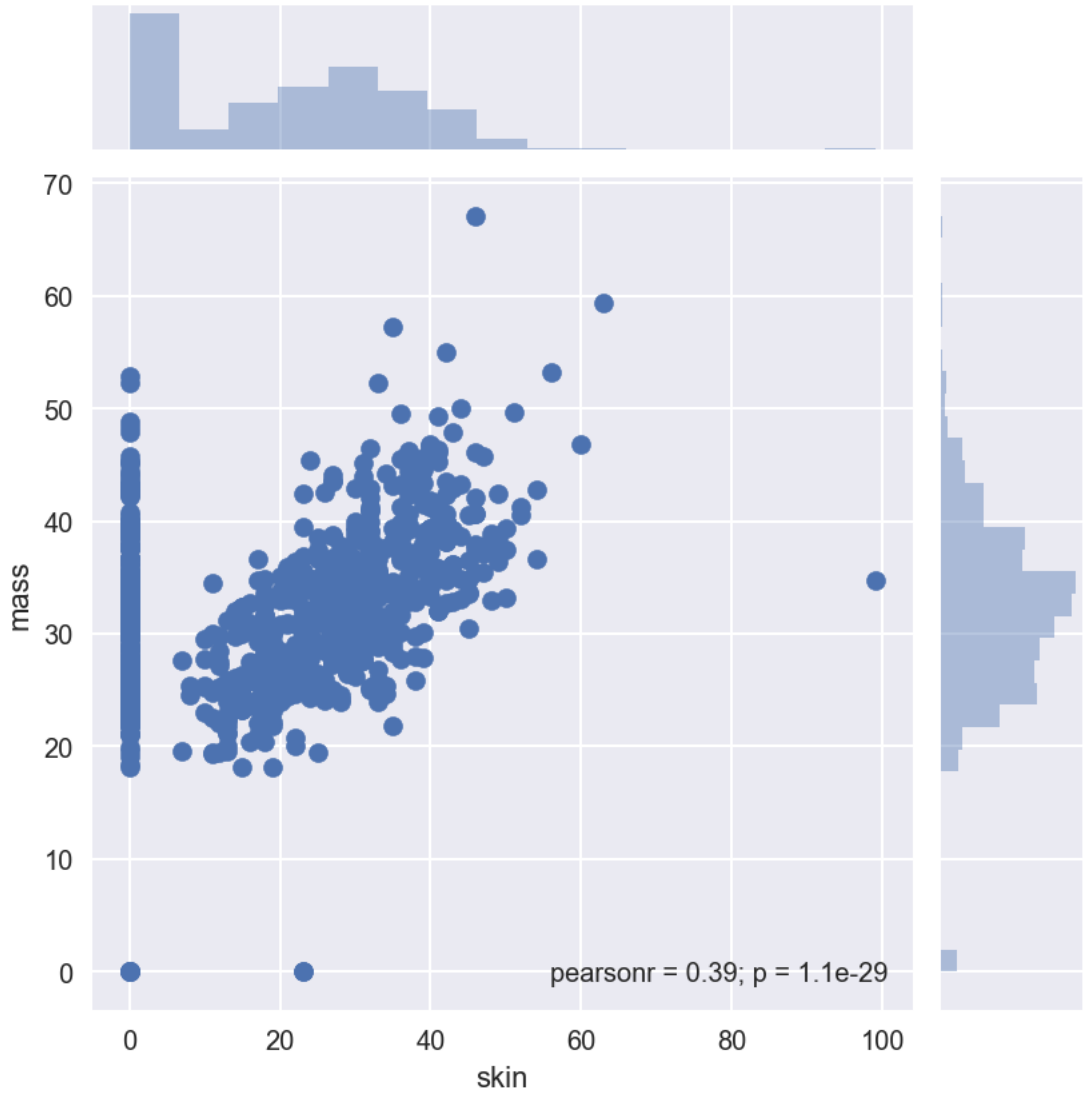
In [13]:
```
#Vemos una correlación fuerte en la medida que los colores se van haciendo más oscuro
#Centraremos los análisis en estudiar aquellas correlaciones más oscurecidas, en part
#Entre Skin y mass,pres y mass,age y pedi,plas y test,test y skin.
cm=dataframe.corr()
sbs.heatmap(cm,square=True)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x120cba940>



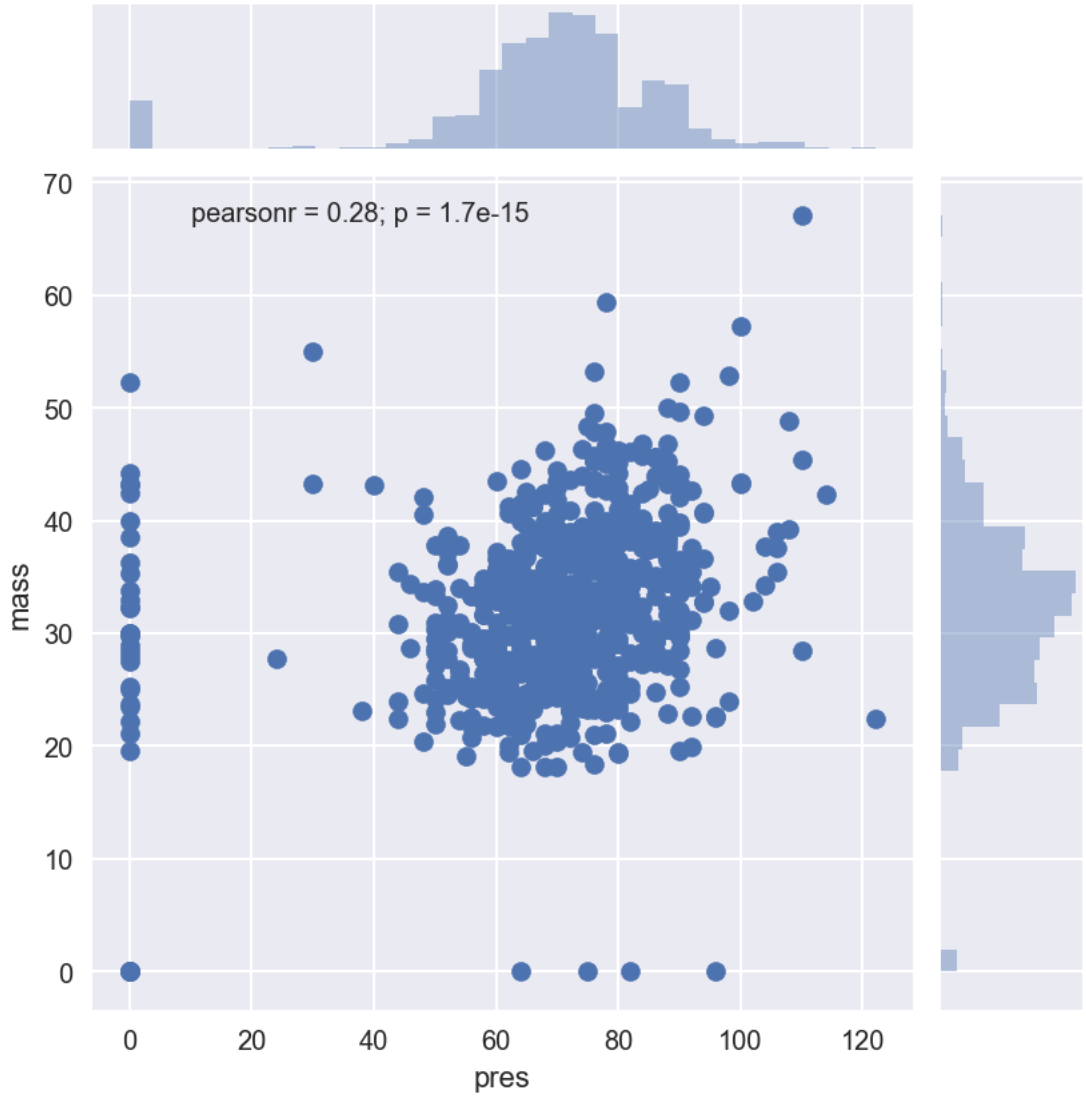
In [14]:
```
sbs.jointplot(dataframe["skin"], dataframe["mass"])
```

Out[14]: <seaborn.axisgrid.JointGrid at 0x12135d390>



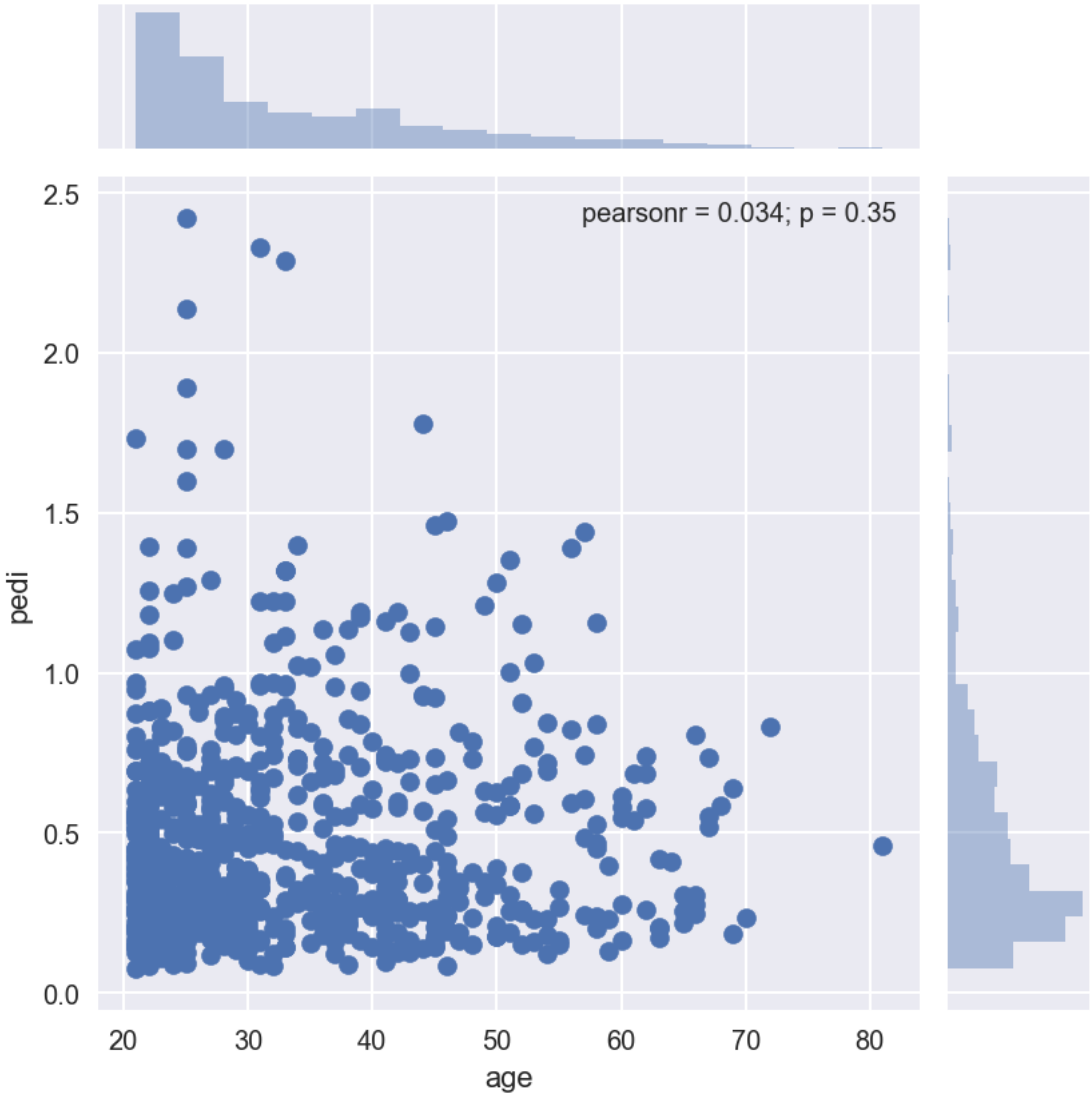
In [15]: sbs.jointplot(dataframe["pres"], dataframe["mass"])

Out[15]: <seaborn.axisgrid.JointGrid at 0x1212c1f60>



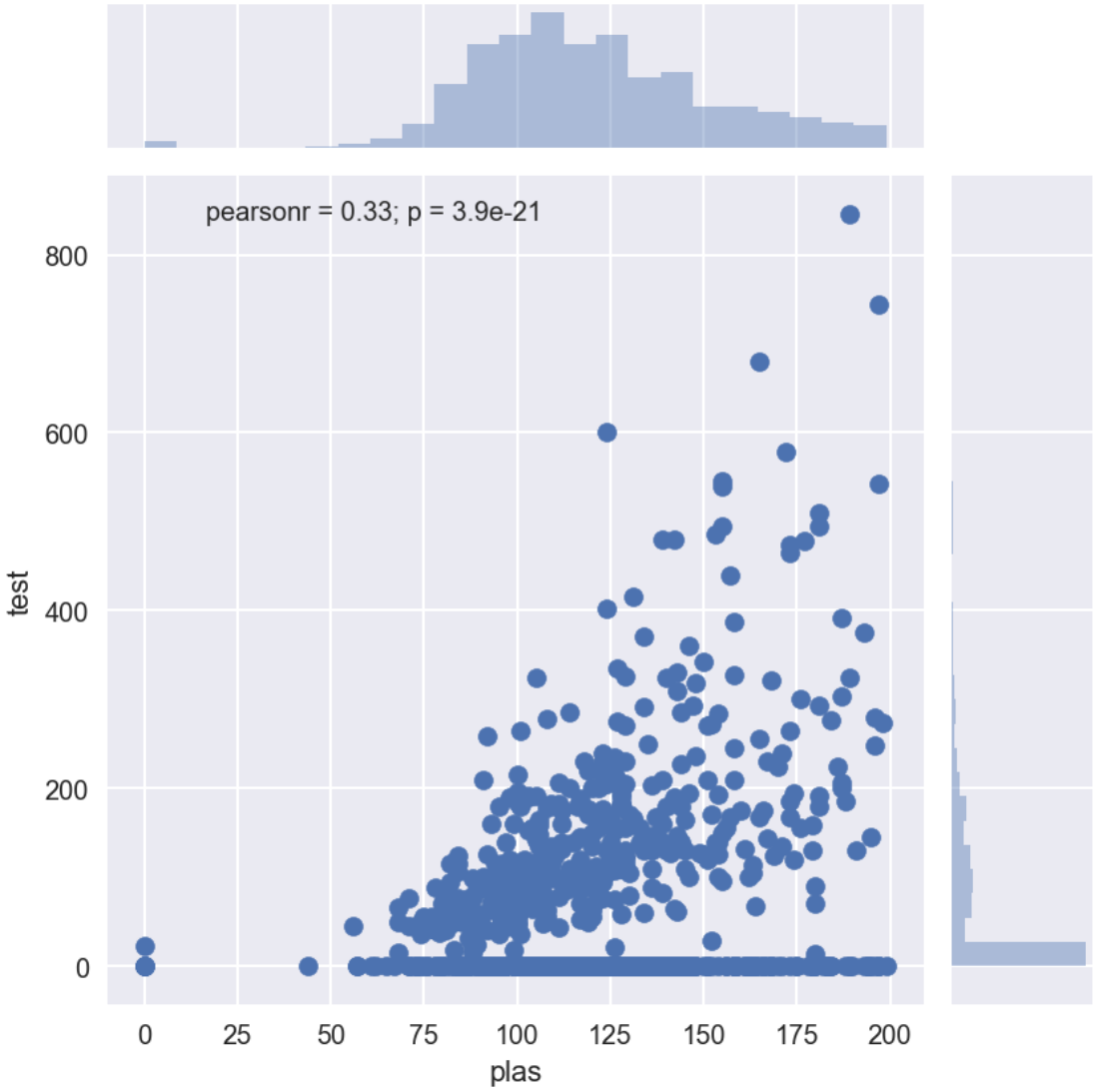

```
In [16]: sbs.jointplot(dataframe["age"], dataframe["pedi"])
```

```
Out[16]: <seaborn.axisgrid.JointGrid at 0x12182ca20>
```



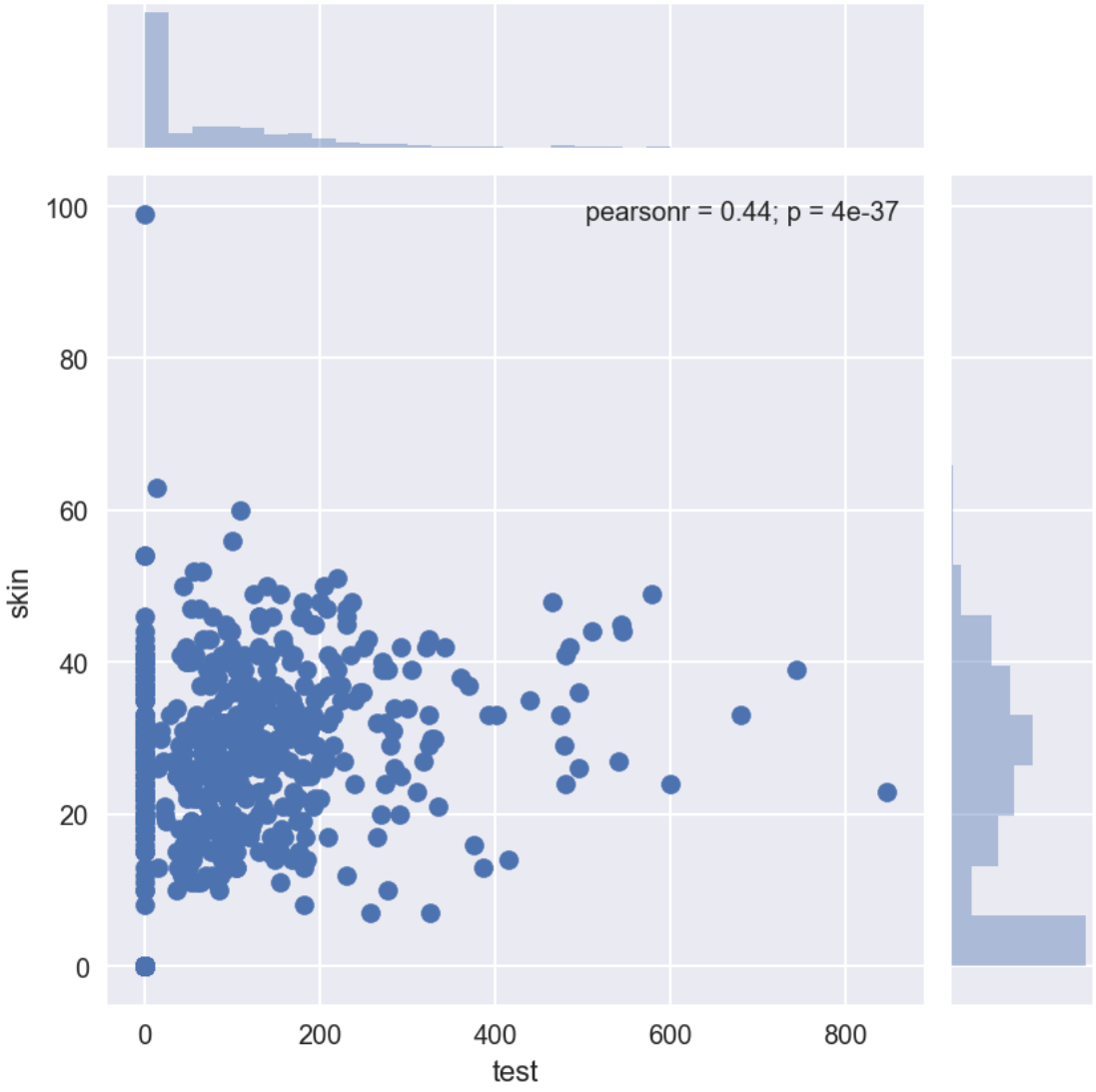

```
In [17]: sbs.jointplot(dataframe["plas"], dataframe["test"])
```

```
Out[17]: <seaborn.axisgrid.JointGrid at 0x12244b278>
```




```
In [18]: sbs.jointplot(dataframe["test"], dataframe["skin"])
```

```
Out[18]: <seaborn.axisgrid.JointGrid at 0x122a15518>
```



In [19]: *#Ahora usaremos con los test chi-2 para aquellas variables que son categóricas*

In [ ]: *#In progress*

In [20]: *#Pre-procesamiento de la data usando la libreria scikit-learn, el cual provee dos len*
#la data, los cuales son útiles en distintos contextos:Fit and Multiple Transform.
#Combinaremos Fit-And-Transform para realizar las siguientes tareas:
#Estandarizar la data númerica (e.g. promedio de 0 y desviación estandar de 1) usando
#Normalizar la data númerica (e.g. a un rango de 0-1) usando la opción de rango (rang
#Explorar un feature engineering más avanzado como Binarizing, que transforma a binar
#For example, the snippet below loads the Pima Indians onset of diabetes dataset, cal
#to standardize the data, then creates a standardized copy of the input data.

In [21]: *#Separamos el arreglo en inputs y outputs, o bien variables explicativas y target*
```
array = dataframe.values
```

```
X = array[:,0:8]
Y = array[:,8]
```

In [22]:
```
#Tarea 1a: Estandarizar los datos usado scale
#Calcula los parámetros necesarios para estandarizar los datos
scaler = StandardScaler().fit(X)
#crea una copia estandarizada de los datos de entrada
rescaledX = scaler.transform(X)
#obtenemos un print entendible de las 5 primeras filas de la data transformada usando
#https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.set_printoptions.h
#https://stackoverflow.com/questions/2891790/how-to-pretty-printing-a-numpy-array-wit
np.set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

```
[[ 0.64   0.848  0.15   0.907 -0.693  0.204  0.468  1.426]
 [-0.845 -1.123 -0.161  0.531 -0.693 -0.684 -0.365 -0.191]
 [ 1.234  1.944 -0.264 -1.288 -0.693 -1.103  0.604 -0.106]
 [-0.845 -0.998 -0.161  0.155  0.123 -0.494 -0.921 -1.042]
 [-1.142  0.504 -1.505  0.907  0.766  1.41   5.485 -0.02 ]]
```

In [ ]:
```
#Tarea 1b: Estandarizar la data númerica usando center (in progress)
```

In [26]:
```
#Hacemos una regresión logística usando los datos sin el pre-procesamiento y la técni
kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression()
results = cross_val_score(model, X, Y, cv=kfold)
print(("Accuracy: %.3f%% (%.3f%%)") + str(results.mean()*100.0) + str(results.std()*1
```

```
Accuracy: %.3f%% (%.3f%%)76.9514695834.84105192457
```

In [27]:
```
#Hacemos la regresión logística usando los datos con el pre-procesamiento y la técnic
#Vemos un incremento en la métrica de precisión de un 1.04 puntos porcentuales con un
#0.16 puntos porcentuales.Este incremento es marginal y nos indica que debemos trabaj
#si lo que queremos es lograr mayor precisión.
kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression()
results = cross_val_score(model, rescaledX, Y, cv=kfold)
print(("Accuracy: %.3f%% (%.3f%%)") + str(results.mean()*100.0) + str(results.std()*1
```

```
Accuracy: %.3f%% (%.3f%%)77.99555707455.0088006076
```

In [28]:
```
#Lo que contiene results son las métricas de precisión para cada uno de los 10 data s
#lo que se muestra como resultado en ambos modelos anteriores es el promedio y desvia
#tante entender el rango en el que se mueven, pues podemos ver que la mínima precisió
#Si no estamos dispuestos a tolerar la posibilidad de un 68% de precisión, entonces n
results
```

```
Out[28]: array([ 0.688,  0.831,  0.766,  0.701,  0.779,  0.792,  0.844,  0.831,
                 0.763,  0.803])
```

```
In [83]: dataframe.columns=['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'cla
```

```
In [68]: #Usaremos la libreria patsy para crear el modelo de regresión logística con matrices
         from patsy import dmatrices
         f='clas ~ C(preg) + plas + pres + skin + test + mass + pedi + age'
```

```
In [70]: Y,X= dmatrices(formula_like=f,data=dataframe, return_type="dataframe")
         print(X.columns)
```

```
Index(['Intercept', 'C(preg)[T.1]', 'C(preg)[T.2]', 'C(preg)[T.3]',
       'C(preg)[T.4]', 'C(preg)[T.5]', 'C(preg)[T.6]', 'C(preg)[T.7]',
       'C(preg)[T.8]', 'C(preg)[T.9]', 'C(preg)[T.10]', 'C(preg)[T.11]',
       'C(preg)[T.12]', 'C(preg)[T.13]', 'C(preg)[T.14]', 'C(preg)[T.15]',
       'C(preg)[T.17]', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age'],
      dtype='object')
```

```
In [73]: #Ahora usaremos un simple split en 70-30 para contrastar
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=
         model2 = LogisticRegression()
         model2.fit(X_train, y_train)
```

```
/Users/iairlinker/anaconda/lib/python3.6/site-packages/sklearn/utils/validation.py:526: DataCon
  y = column_or_1d(y, warn=True)
```

```
Out[73]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
In [78]: #Generamos la predicción sobre el dataset de test utilizando el modelo
         predicted = model2.predict(X_test)
         #generamos las probabilidades para el conjunto de test
         probs = model2.predict_proba(X_test)
```

```
In [79]: #mostramos las métricas de evaluación correspondiente a la precisión y el auc score.
         #resultado que con la data estandarizada y usando 10k-fold y el segundo es un buen re
         print(metrics.accuracy_score(y_test, predicted))
         print(metrics.roc_auc_score(y_test, probs[:, 1]))
```

```
0.779220779221
0.814167670856
```

```
In [80]: #Mostramos la matriz de confusión donde las filas son, de arriba hacia abajo,personas
         #y personas que si,y las columnas son mi prección que va de izquierda a derecha perso
         #diabetes y personas que si
         print(metrics.confusion_matrix(y_test, predicted))
```

```
[[142  15]
 [ 36  38]]
```

In [82]:
```
#Finalmente desplegamos un resumen de las métricas donde podemos obersvar que el gran
#predicción de 1,es decir las personas que si tendrán inicio de diabetes pero yo pred
#En el contexto de este análisis esto es un resultado malo ya que dejaré de darles el
#de las personas.
print(metrics.classification_report(y_test, predicted))
```

```
             precision    recall  f1-score   support

        0.0       0.80      0.90      0.85       157
        1.0       0.72      0.51      0.60        74

avg / total       0.77      0.78      0.77       231
```