Taller MVP usando modelos b?sicos con datos de inicio de diabetes en los indios Pima

August 5, 2017

In [134]:
```
import pandas as pd
from pandas import read_csv
import numpy as np
import scipy as sp
import matplotlib as plt
get_ipython().magic(u'matplotlib inline')
get_ipython().magic(u"config InlineBackend.figure_format='retina'")
import plotly
import plotly.plotly as py
import plotly.graph_objs as go
from plotly.tools import FigureFactory as FF
import seaborn as sbs
#Librerias de sklearn
from sklearn.preprocessing import StandardScaler,scale, binarize
from sklearn.model_selection import train_test_split,KFold,cross_val_score
from sklearn.ensemble import GradientBoostingClassifier,RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
from sklearn.grid_search import GridSearchCV
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']
df = read_csv(url, names=names)
df.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]:
```
#Nos entrega un resumen estadístico rápido de cada una de las variables (deben ser num
df.describe()
```

Out[3]:

	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 [4]:
```
#Attribute Information:

#1. Number of times pregnant
#2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
#3. Diastolic blood pressure (mm Hg)
#4. Triceps skin fold thickness (mm)
#5. 2-Hour serum insulin (mu U/ml)
#6. Body mass index (weight in kg/(height in m)^2)
#7. Diabetes pedigree function
#8. Age (years)
#9. Class variable (0 or 1)
```

In [5]:
```
df.shape
```

Out[5]: (768, 9)

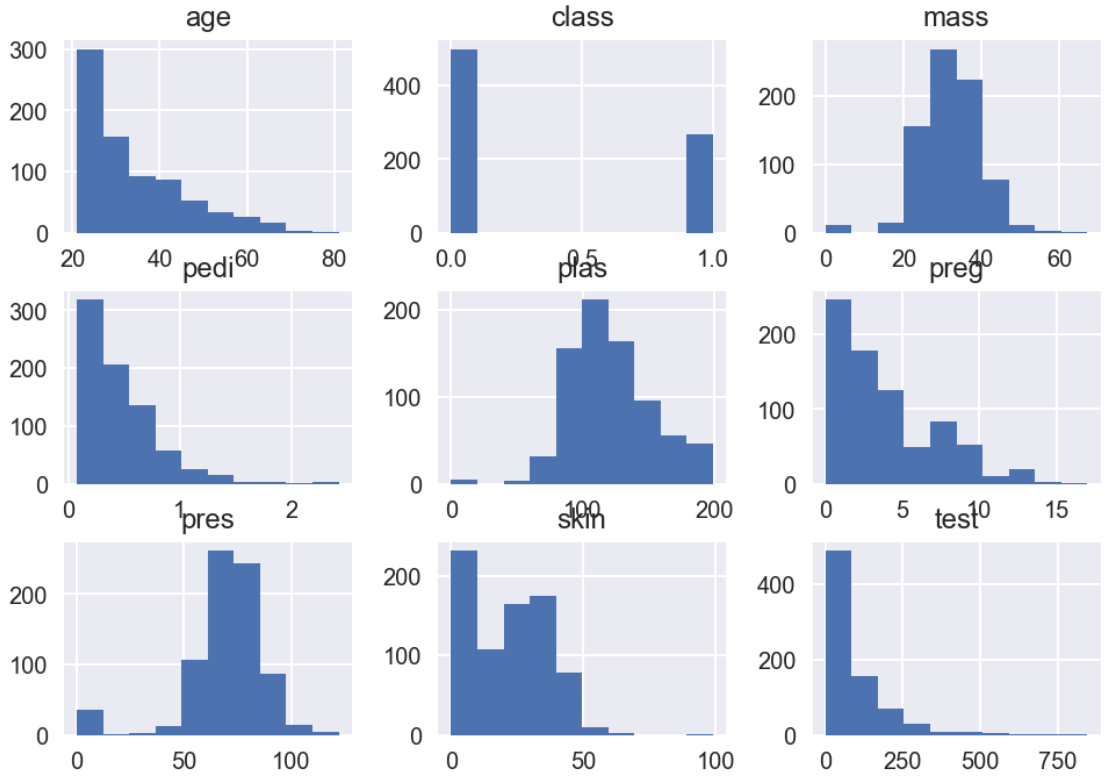
In [6]:
```
#forma rapida de contar los valores nuloes o "missing values" en las variables como %
round(100*(df.isnull().sum()/1000000))
```

Out[6]:
```
preg     0.0
plas     0.0
pres     0.0
skin     0.0
test     0.0
mass     0.0
pedi     0.0
age      0.0
class    0.0
dtype: float64
```

In [7]:
```
#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 [8]:
```
#Pareciera ser que estamos en presencia de distribuciones normales, normales con sesgo
#y exponencial. Veremos si esto efectivamente es así más adelante
df.hist()
```

Out[8]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x11cb5a0b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1207716d8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1207ee9b0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x12084c048>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1208bb1d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1208bb208>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x12095cbe0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1209db898>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x120a14dd8>]], dtype=object)
```

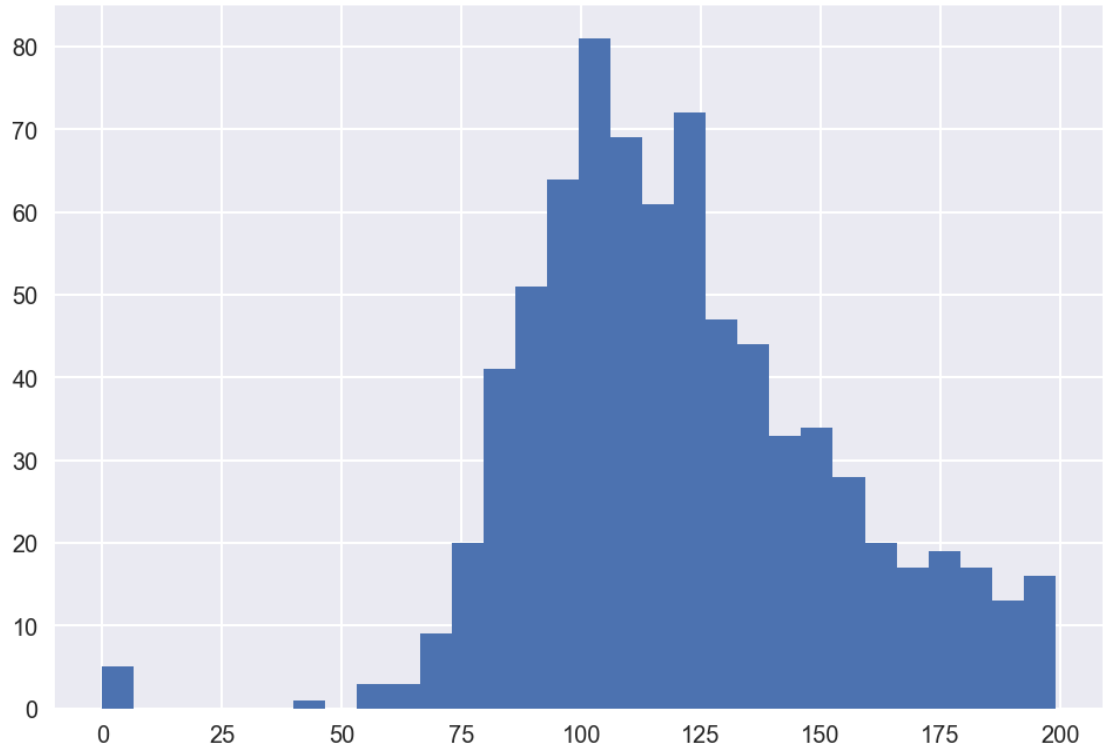


In [9]:
```
#COmenzamos con el análisis univariante y multivariante con respectoa la variable targ
#análisis multivariante entre las disitinstas variables explicativas.
```

In [10]:
```
#1.-Análsis de la variable plas.
#Variable número ica con valores entre 0 y 199 que representa la concentración de plasm
#tolerancia a la glucosa oral. Debemos cuestionar si es que los valores 0 corresponde
```

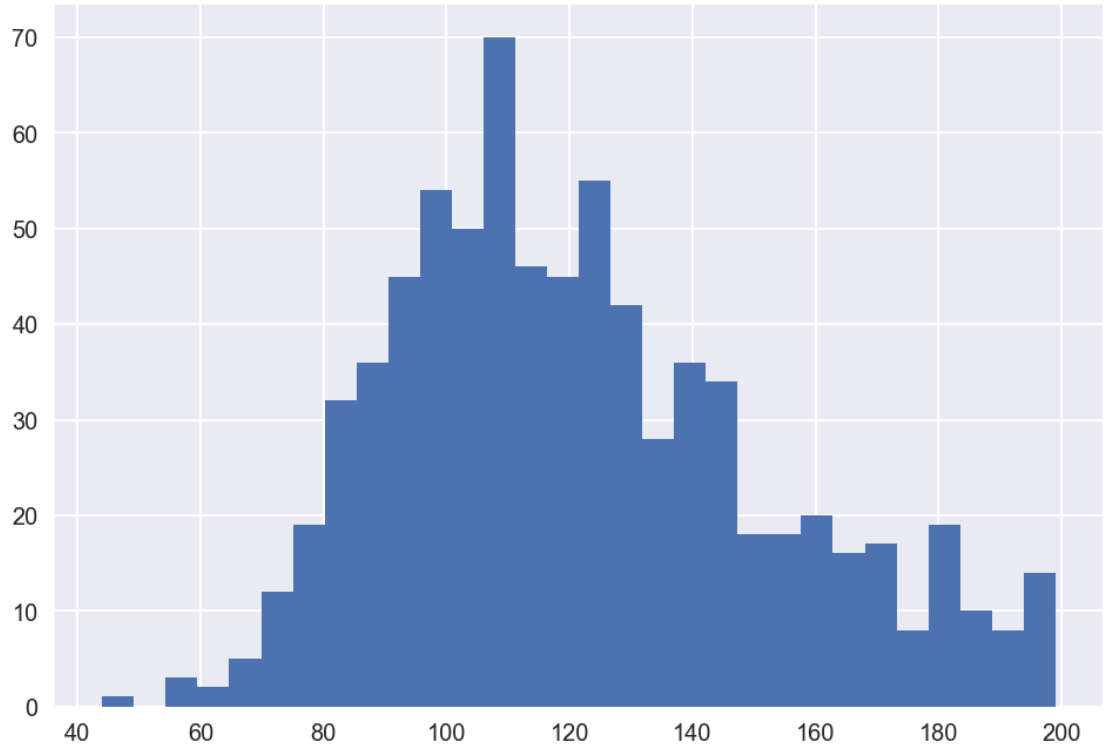
```
#Con respectoa a la distribución de la variable, podemos decir que se ve como una nor
#De hecho, su media es casi igual a la mediana y el la media se encuentra 20% más cer
#de 199.Adicionalmente,podemos ver muy pocos valores entre 0 y 99 (solo un 25%), mien
#de los valores se encuentra en el intervalo cercano al máximo del rango.
df.plas.hist(bins=30)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x120af7898>



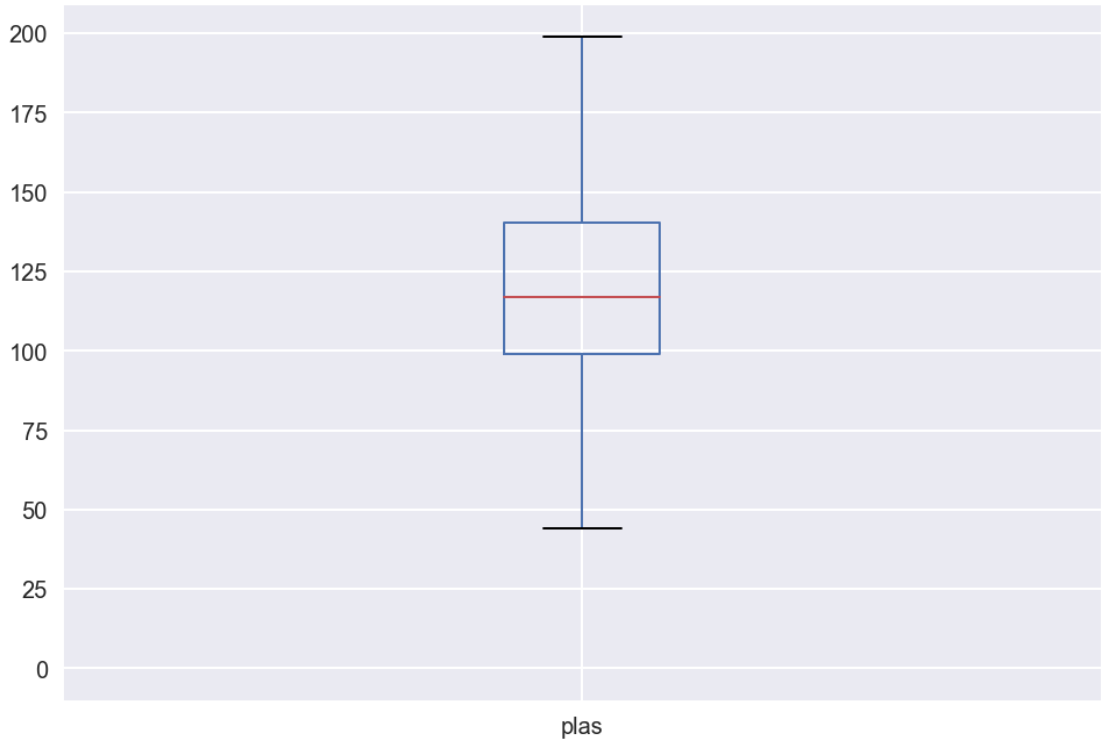
In [11]:
```
#eliminando los valores 0 vemos que el histograma se parece más a una normal con sesg
df[df['plas']>0].plas.hist(bins=30)
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x12147e748>



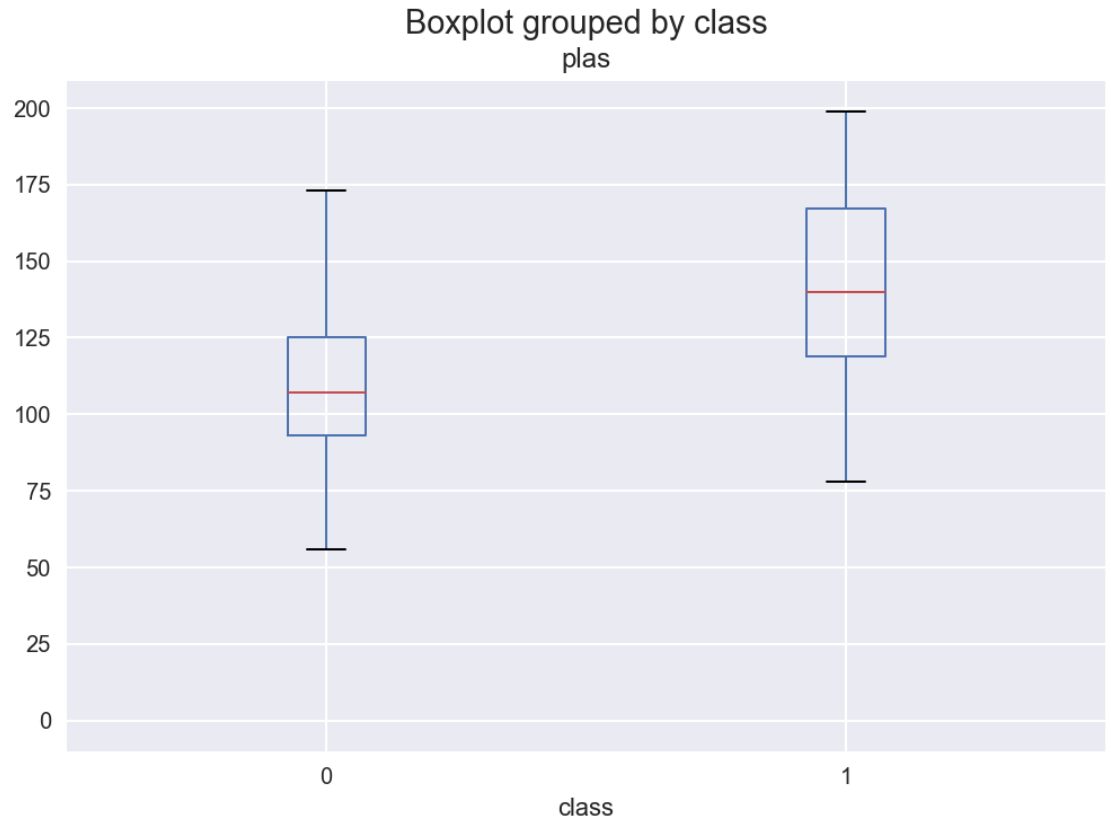
In [12]:
```
#El boxplot nos confirma lo que enunciamos anteriormente, que el 50% de la data se en
df.boxplot(column='plas')
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1218218d0>
```



In [13]:
```
#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
df.boxplot(column='plas',by='class')
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x121b48128>
```



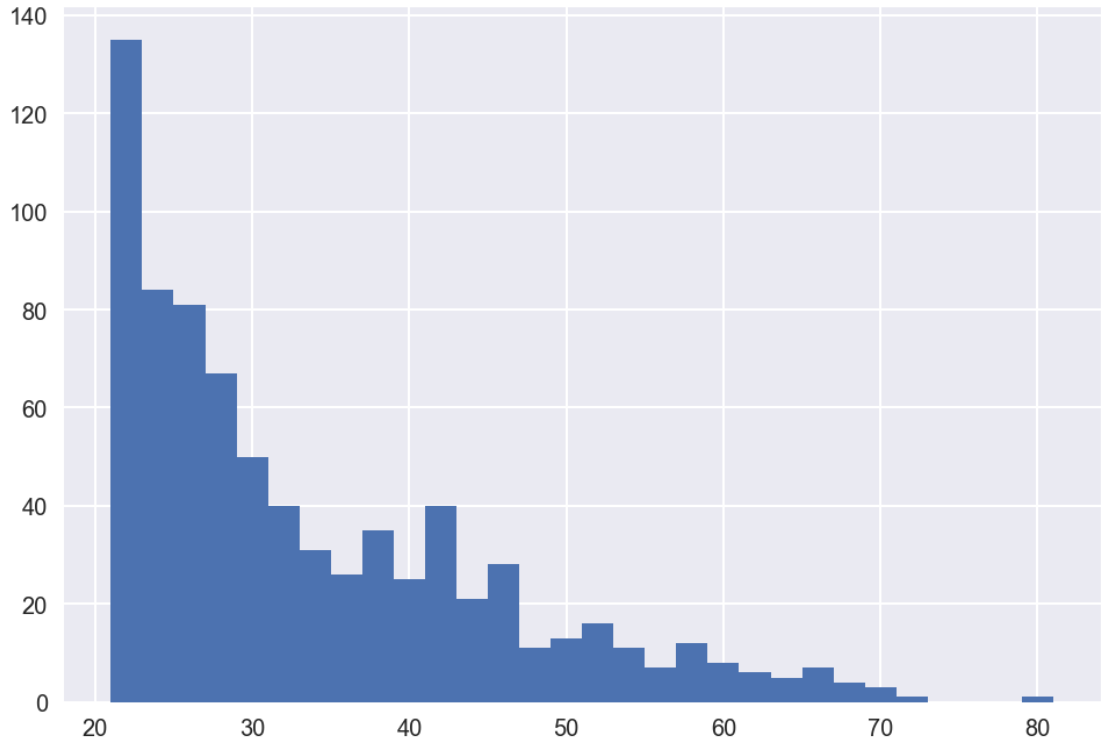
In [14]:
```
df.groupby('class')['plas'].describe()
```

Out[14]:

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 [15]:
```
#Como podemos ver,la edad tiene una distribución exponencial,la cual parte desde los
#es claro
df['age'].hist(bins=30)
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x121840f28>
```



In [16]:
```
plotly.tools.set_credentials_file(username='ilinker', api_key='fXcCnCdQtQ8OMF4r19Va')
```

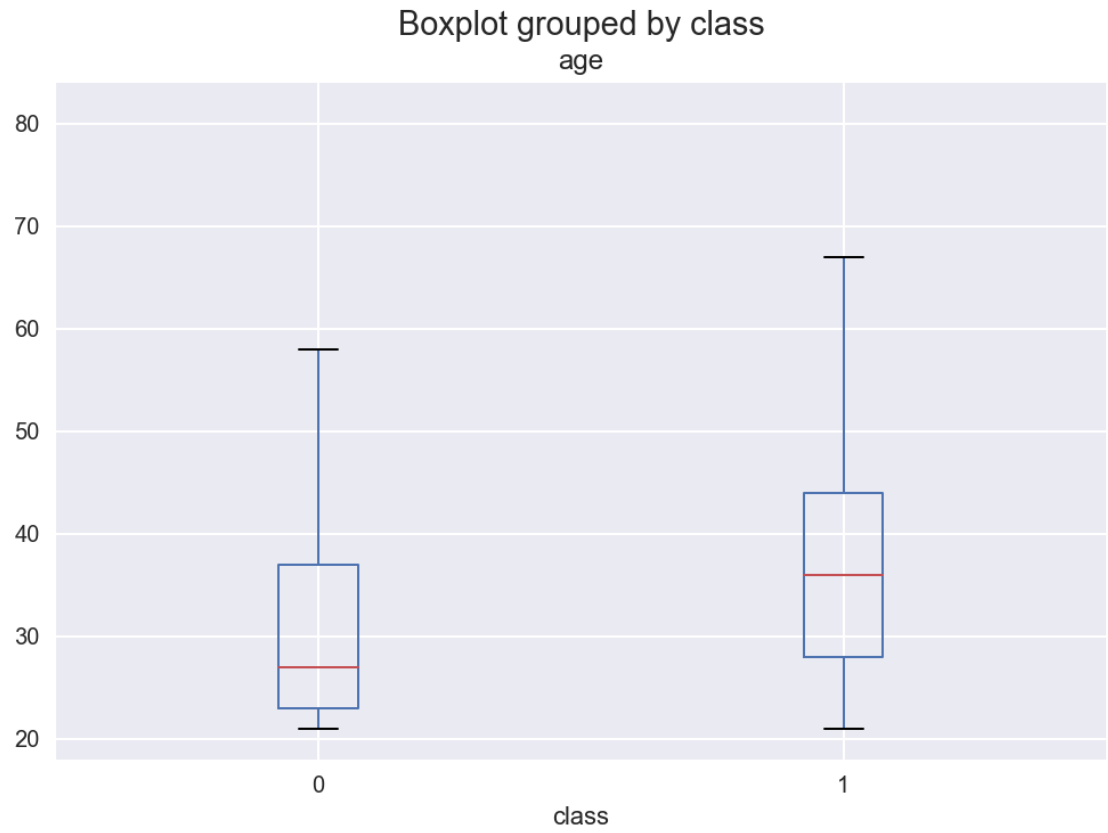
In [17]:
```
#Vamos. a usar plotly para dejar en evidencia la concentración de la edad en las pers
trace = go.Histogram(x=df['age'], xbins=dict(start=np.min(df['age']), size=5, end=np.m
                     marker=dict(color='rgb(0, 0, 100)'))

layout = go.Layout(
    title="Histogram Frequency Counts Age of Indios-Pima"
)

fig = go.Figure(data=go.Data([trace]), layout=layout)
py.iplot(fig, filename='histogram-freq-counts-age-Indios-Pima')
```

Out[17]:
```
<plotly.tools.PlotlyDisplay object>
```

In [18]:
```
#cumfreqs, lowlim, binsize, extrapoints = sp.stats.cumfreq(df['age'], numbins=6)
#counts = df['age'].value_counts()
```

In [19]:
```
#Podemos obersvar que los datos de edad distribuyen distinto según clase.Vemos que
df.boxplot(column='age',by='class')
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12225e550>
```



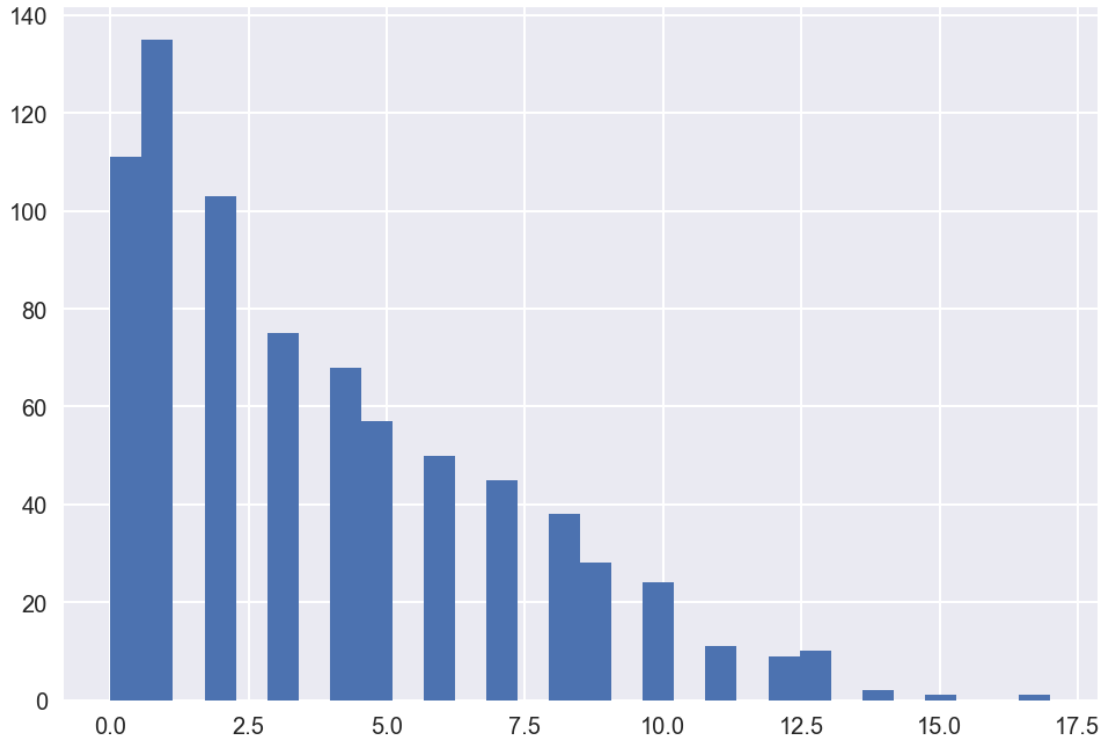

```
In [20]: df.groupby('class')['age'].describe()
```

Out[20]:

class	count	mean	std	min	25%	50%	75%	max
0	500.0	31.190000	11.667655	21.0	23.0	27.0	37.0	81.0
1	268.0	37.067164	10.968254	21.0	28.0	36.0	44.0	70.0

```
In [21]: df['preg'].hist(bins=30)
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1225fcbe0>
```



In [22]:
```
#Como podemos ver, más del 50% de los valores está concentrado entre 0 y 3 veces, y es
#personas sin diabetes hasta los 7 embarazos, en donde se invierte la proporción.
contingencyTable = pd.crosstab(index=df['class'],columns=df['preg'],margins=True)
contingencyTable
```

Out[22]:

preg	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	17	All
class																		
0	73	106	84	48	45	36	34	20	16	10	14	4	5	5	0	0	0	500
1	38	29	19	27	23	21	16	25	22	18	10	7	4	5	2	1	1	268
All	111	135	103	75	68	57	50	45	38	28	24	11	9	10	2	1	1	768

In [23]:
```
#Acá podemos ver de forma más clara el punto de inflexión en el número de embarazos.
#transformación de variable en la cual agrupamos las categorías en función de su rela
#usaremos el test chi-2 para testear independencia, pero es evidente que podemos agru
contingencyTable = pd.crosstab(index=df['class'],columns=df['preg'],margins=True, norm
contingencyTable
```

Out[23]:
preg	0	1	2	3	4	5	6	7
class								
0	0.657658	0.785185	0.815534	0.64	0.661765	0.631579	0.68	0.444444
1	0.342342	0.214815	0.184466	0.36	0.338235	0.368421	0.32	0.555556

preg	8	9	10	11	12	13	14	15	17
class									
0	0.421053	0.357143	0.583333	0.363636	0.555556	0.5	0.0	0.0	0.0
1	0.578947	0.642857	0.416667	0.636364	0.444444	0.5	1.0	1.0	1.0

preg	All
class	
0	0.651042
1	0.348958

In [ ]:

In [24]:
```
df[['preg']]=df.preg.map({0:0,1:0,2:0,3:0,4:0,5:0,6:1,7:1,8:1,9:1,10:1,11:1,12:1,13:1
list(df['preg'].unique())
```

Out[24]: [1, 0]

In [25]:
```
contingencyTable = pd.crosstab(index=df['class'],columns=df['preg'],margins=True)
contingencyTable
```

Out[25]:
preg	0	1	All
class			
0	392	108	500
1	157	111	268
All	549	219	768

In [26]:
```
contingencyTable = pd.crosstab(index=df['class'],columns=df['preg'],margins=True, norr
contingencyTable
```

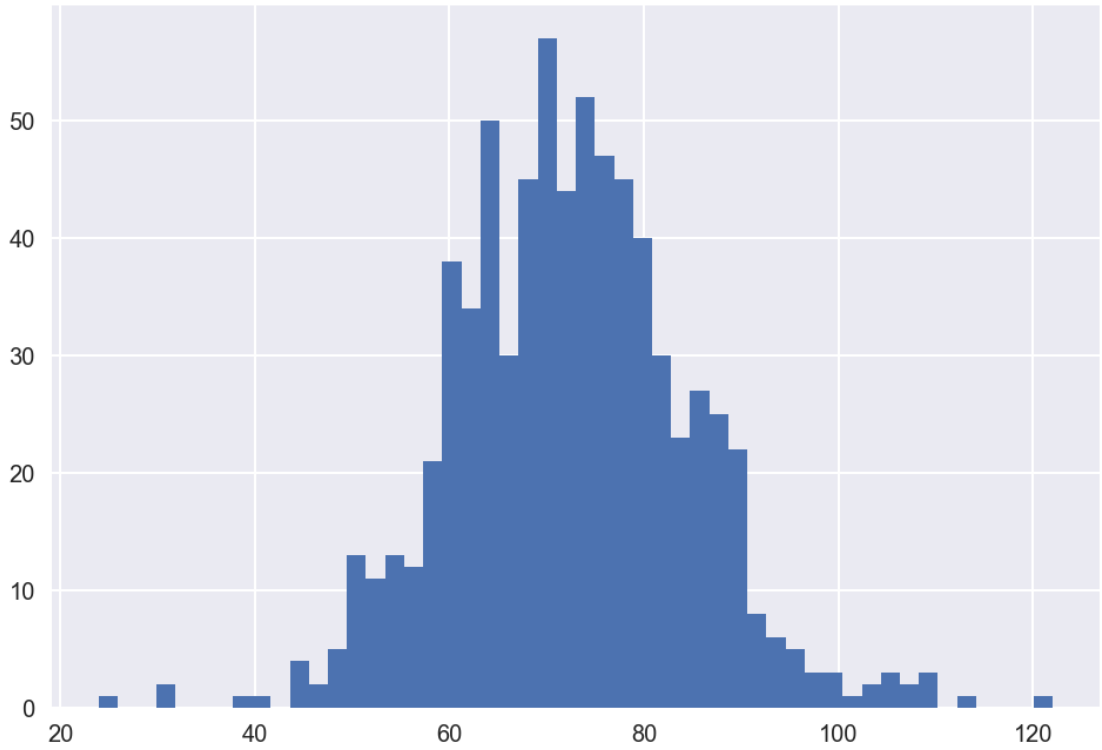
Out[26]:
preg	0	1	All
class			
0	0.714026	0.493151	0.651042
1	0.285974	0.506849	0.348958

In [27]:
```
#Vamos. a usar plotly para dejar en evidencia la concentración de la edad en las pers
trace = go.Histogram(x=df['pres'], xbins=dict(start=np.min(df['pres']), size=5, end=np
                     marker=dict(color='rgb(0, 0, 100)'))

layout = go.Layout(
    title="Histogram Frequency Counts pres of Indios-Pima"
)

fig = go.Figure(data=go.Data([trace]), layout=layout)
py.iplot(fig, filename='histogram-freq-counts-pres-Indios-Pima')
```

Out[27]: <plotly.tools.PlotlyDisplay object>

In [28]:
```
df[df['pres']>0]['pres'].hist(bins=50)
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x12227c7f0>



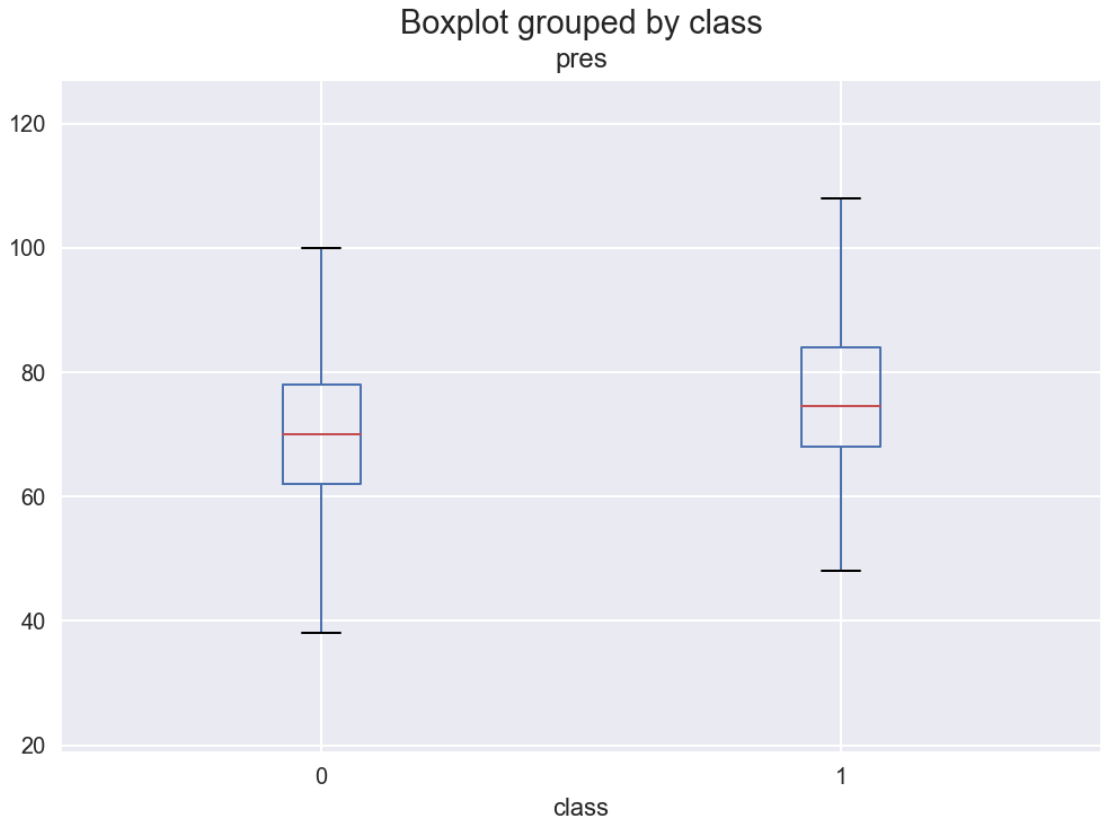
In [29]:
```
#La media está centrada y la distribución se parece mucho a una normal
df[df['pres']>0].groupby('class')['pres'].describe()
```

Out[29]:

class	count	mean	std	min	25%	50%	75%	max
0	481.0	70.877339	12.161223	24.0	62.0	70.0	78.0	122.0
1	252.0	75.321429	12.299866	30.0	68.0	74.5	84.0	114.0

In [30]:
```
#Podemos obersvar que los datos de edad distribuyen distinto según clase.Vemos que
df[df['pres']>0].boxplot(column='pres',by='class')
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x122d2d358>

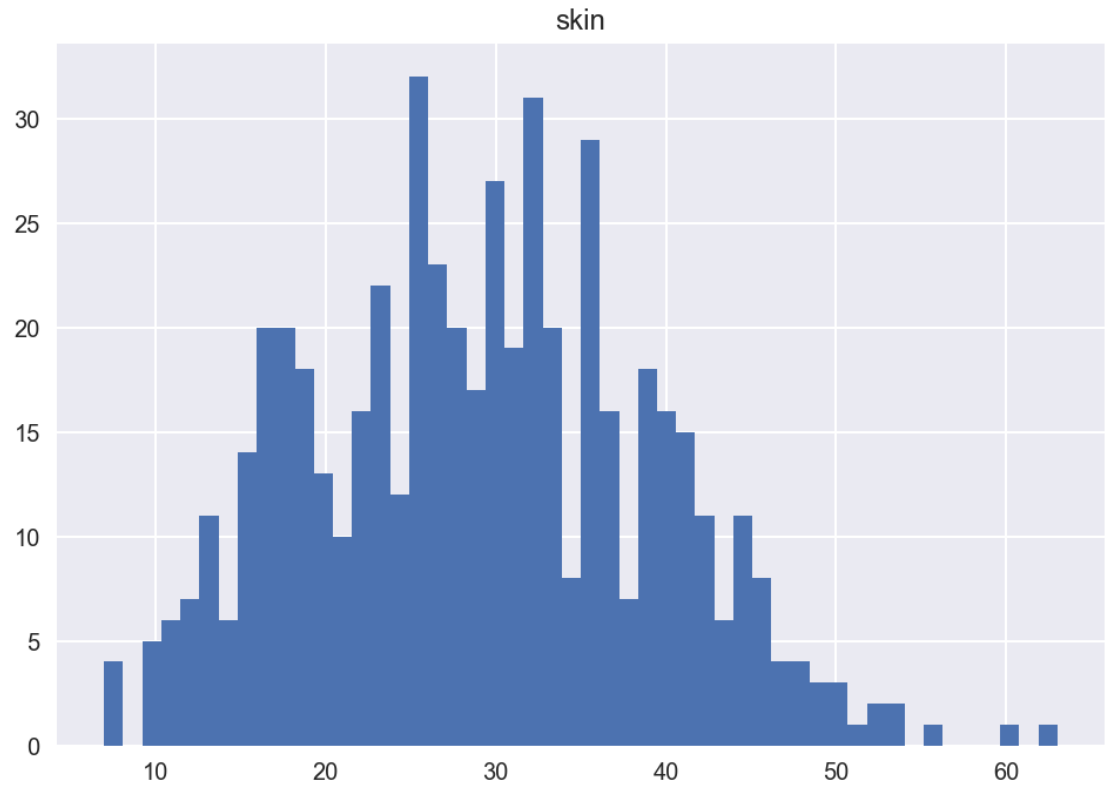


In [31]:
```
#Vamos. a usar plotly para dejar en evidencia la concentración de la edad en las pers
trace = go.Histogram(x=df['skin'], xbins=dict(start=np.min(df['skin']), size=5, end=n
                     marker=dict(color='rgb(0, 0, 100)'))

layout = go.Layout(
    title="Histogram Frequency Counts skin of Indios-Pima"
)

fig = go.Figure(data=go.Data([trace]), layout=layout)
py.iplot(fig, filename='histogram-freq-counts-skin-Indios-Pima')
```

Out[31]:
```
<plotly.tools.PlotlyDisplay object>
```

In [32]:
```
df.loc[(df.skin<70) & (df.skin >0), ['skin']].hist(bins=50)
```

Out[32]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x122623cc0>]], dtype=object
```



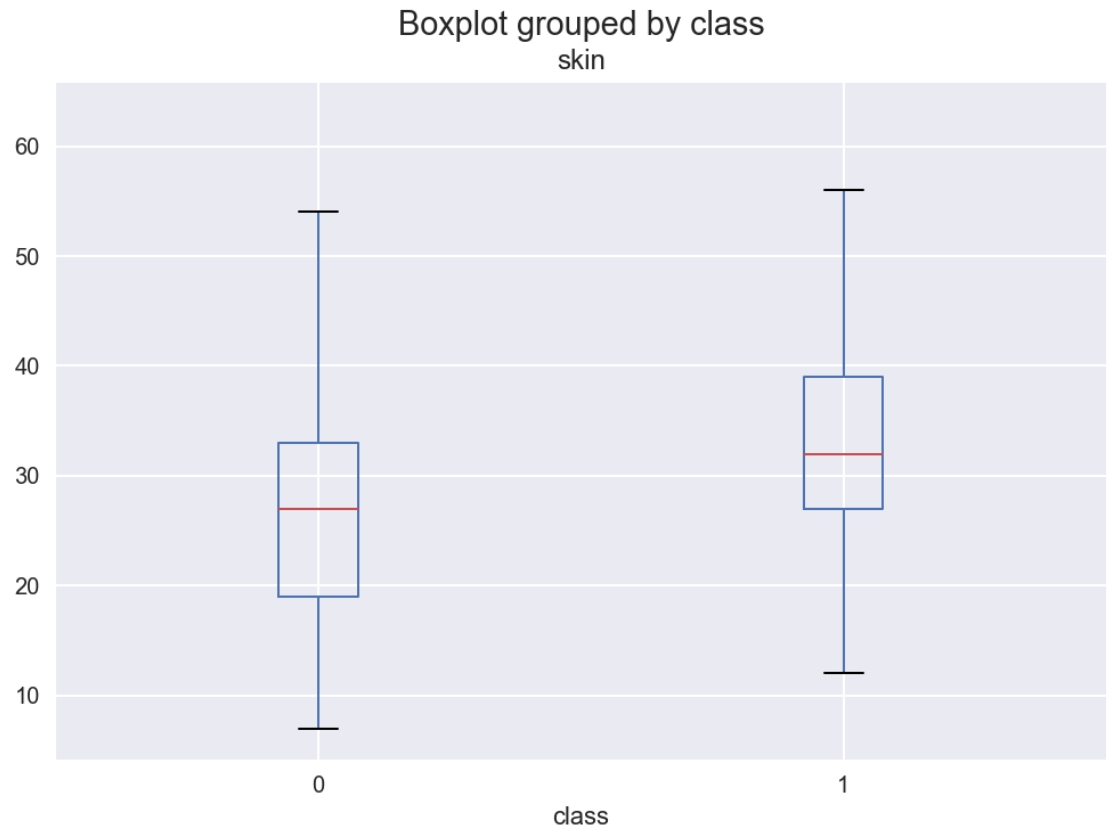
In [33]:
```
#La media está centrada, pero no se parece mucho a una normal
df.loc[(df.skin<70) & (df.skin >0), ['skin','class']].groupby('class')['skin'].descril
```

Out[33]:

class	count	mean	std	min	25%	50%	75%	max
0	361.0	27.235457	10.026491	7.0	19.0	27.0	33.0	60.0
1	179.0	32.631285	9.091194	7.0	27.0	32.0	39.0	63.0

In [34]:
```
#Podemos obersvar que los datos de edad distribuyen distinto según clase.Vemos que
df.loc[(df.skin<70) & (df.skin >0), ['skin','class']].boxplot(column='skin',by='class
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12356c2b0>
```



In [35]:
```
#Test chi-2: Es un test que busca probar la independencia entre variables.
#Referencias:
#http://hamelg.blogspot.cl/2015/11/python-for-data-analysis-part-25-chi.html
#https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chisquare.html
#https://www.slideshare.net/armando310388/prueba-chicuadrado
```

In [36]:
```
def expected_values(var1,var2):
    contingencyTable = pd.crosstab(index=var1,columns=var2,margins=True)
    expected =  np.outer(contingencyTable.iloc[0:(len(contingencyTable.index)-1)]['Al
                         contingencyTable.loc["All"][0:(len(contingencyTable.columns)-
    expected = pd.DataFrame(expected)
    expected.columns = contingencyTable.columns[0:(len(contingencyTable.columns)-1)]
    expected.index = contingencyTable.index[0:(len(contingencyTable.index)-1)]
    return expected

def chi2_test(var1,var2,alpha):
    #Creamos la tabla de contingencia con los valores totales
    contingencyTable = pd.crosstab(index=var1,columns=var2,margins=True)
    #Creamos la tabla de valores esperados
    expected =  np.outer(contingencyTable.iloc[0:(len(contingencyTable.index)-1)]['Al
                         contingencyTable.loc["All"][0:(len(contingencyTable.columns)-
```

```
            expected = pd.DataFrame(expected)
            expected.columns = contingencyTable.columns[0:(len(contingencyTable.columns)-1)]
            expected.index = contingencyTable.index[0:(len(contingencyTable.index)-1)]
            #Creamos la tabla de contingencia sin los valores totales
            contingencyTable = pd.crosstab(index=var1,columns=var2)
            #calculamos el valor calculado de chi2
            chi_squared_stat = (((contingencyTable-expected)**2)/expected).sum().sum()
            #calculamos los grados de libertad
            dof=(len(contingencyTable.columns)-1)*(len(contingencyTable.index)-1)
            #calculamos el valor crítico de chi2
            crit = sp.stats.chi2.ppf(q = alpha,df = dof)
            #calulamos el p-valor
            p_value = 1 - sp.stats.chi2.cdf(x=chi_squared_stat,df=dof)
            #testeamos la hipótesis nula de independencia entre las variables
            if(chi_squared_stat<=crit):
                print("se acepta H0, los valores son independientes y el p-valor es: "+str(p_v
                print("los valores esperados son: "+ str(expected))
            else:
                print("Se rechaza H0, no hay evidencia para decir que los valores son indepen
                print("los valores esperados son:"+ str(expected))
```

In [37]:
```
chi2_test(df['class'],df['preg'],0.99)
```

```
Se rechaza H0, no hay evidencia para decir que los valores son independientes y el p-valor es:
los valores esperados son:preg            0           1
class
0      357.421875  142.578125
1      191.578125   76.421875
```

In [38]:
```
#Recordemos que el test X2 tiene como hipótesis nula-->H0:La variable class es indepe
#El output entrega el valor del estadístico chi-2, el valor p, los grados de libertad
#esperados.Como podemos ver, en este caso no se acepta H0,ya que el p-valor es menor
#x2 calculado es mayor que el estadístico x2. En consecuencia existe una relación ent
contingencyTable = pd.crosstab(index=df['class'],columns=df['preg'])
sp.stats.chi2_contingency(observed=contingencyTable,correction=False)
#Al agregar el parámetro correction como falso le estamos pidiendo que no aplique la
```

Out[38]:
```
(33.617489842500945,
 6.7086793772820514e-09,
 1,
 array([[ 357.421875,  142.578125],
        [ 191.578125,   76.421875]]))
```

In [39]:
```
#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 númerico.
df.corr()
```

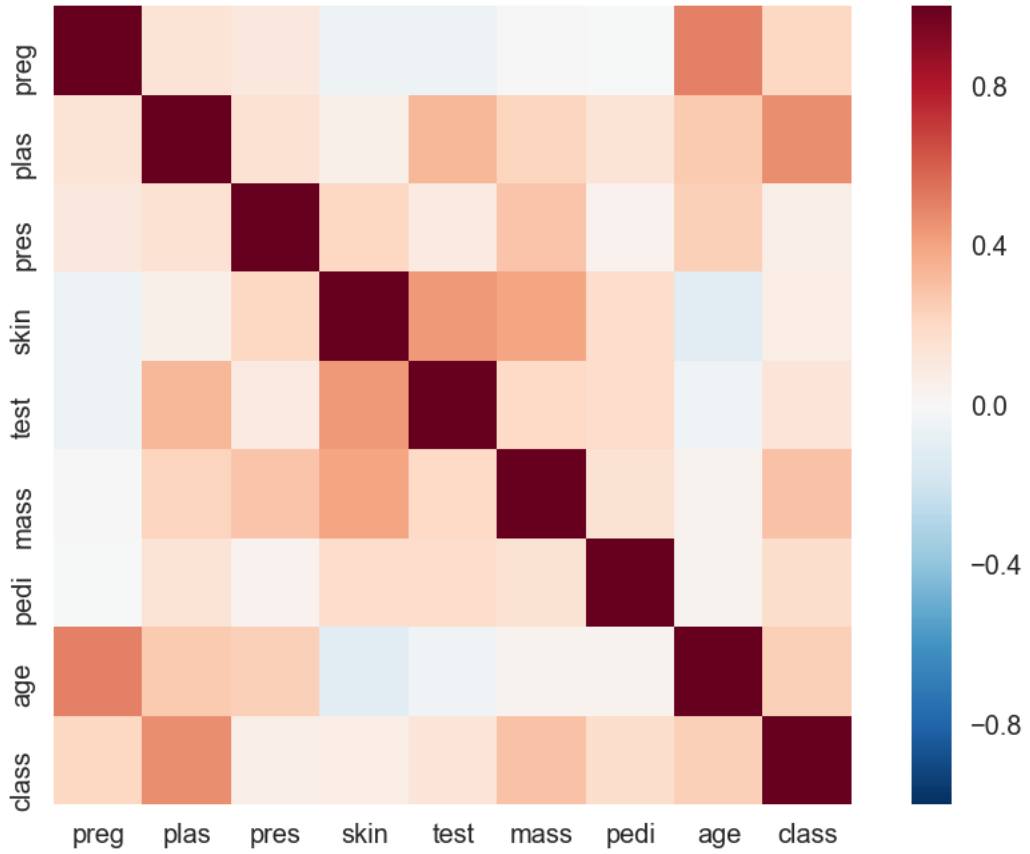
Out[39]:

	preg	plas	pres	skin	test	mass	pedi	\
preg	1.000000	0.138378	0.107036	-0.053998	-0.049233	0.004695	-0.003744	
plas	0.138378	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	
pres	0.107036	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	
skin	-0.053998	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	
test	-0.049233	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	
mass	0.004695	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	
pedi	-0.003744	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	
age	0.505075	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	
class	0.209219	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	

	age	class
preg	0.505075	0.209219
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

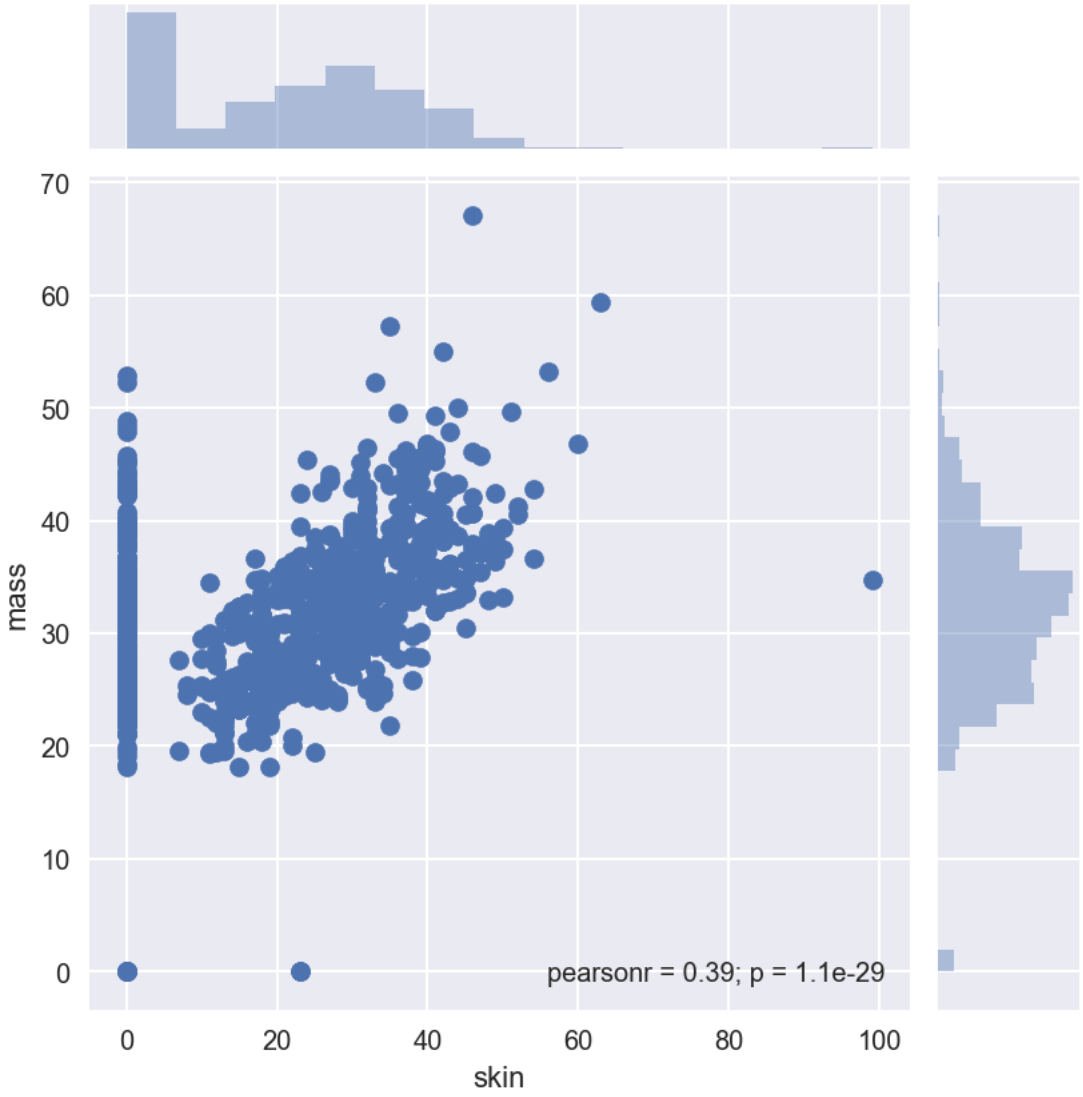
In [40]:
```
#Vemos una correlación fuerte en la medida que los colores se van haciendo más oscuro
#Centraremos los análisis en estudiar aquellas correlaciones más oscurecidas, en part
#Entre Skin y mass,pres y mass,age y pedi,plas y test,test y skin.
cm=df.corr()
sbs.heatmap(cm,square=True)
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x123143278>
```



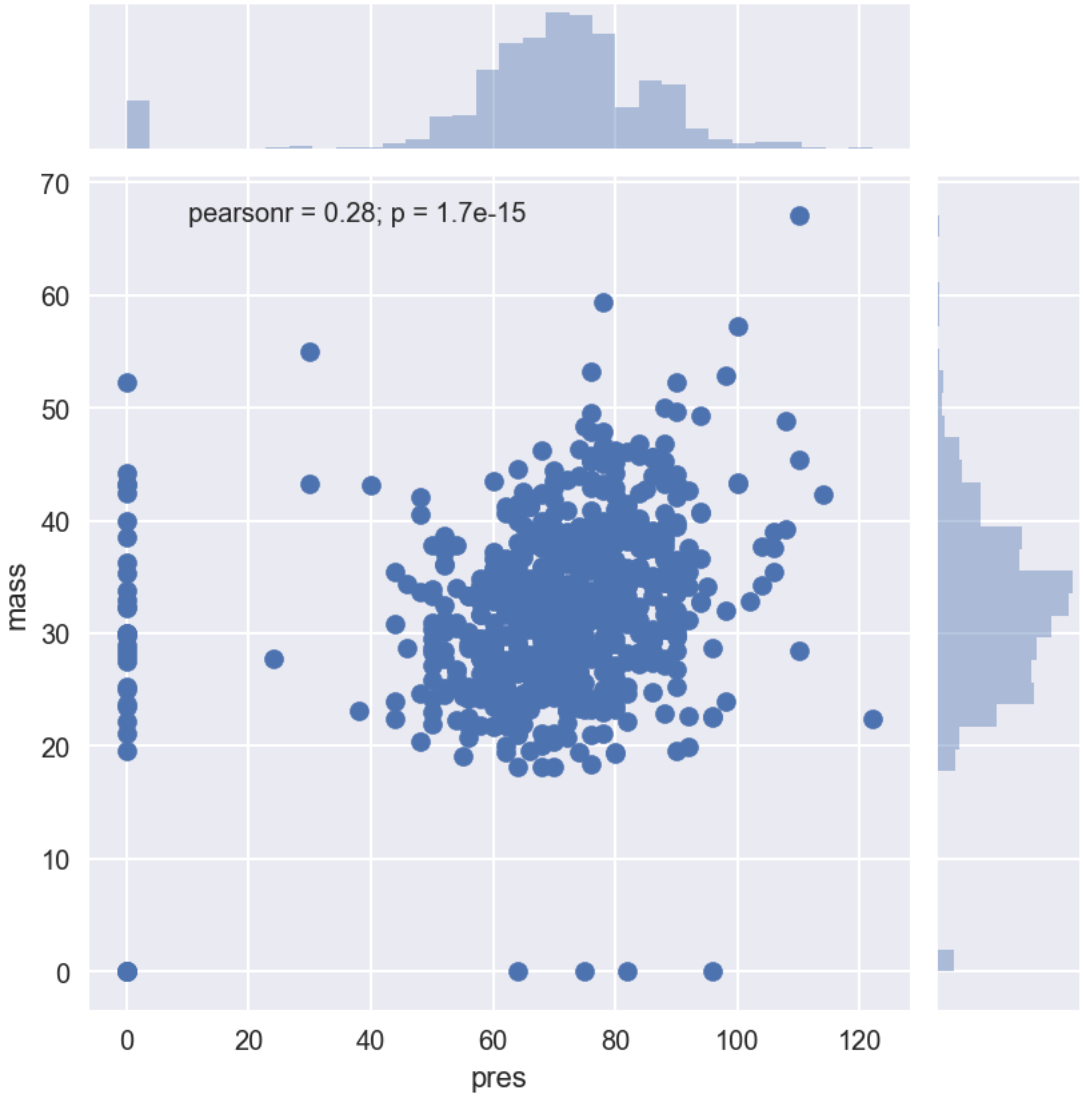
In [41]:
```
#Es evidente que las variables skin y mass tienen una alta correlación y que está sub
#de skin y mass que son valores fuera de rango.Veremos que pasa lo mismo con pres y m
sbs.jointplot(df["skin"], df["mass"])
```

Out[41]: <seaborn.axisgrid.JointGrid at 0x123c89f98>



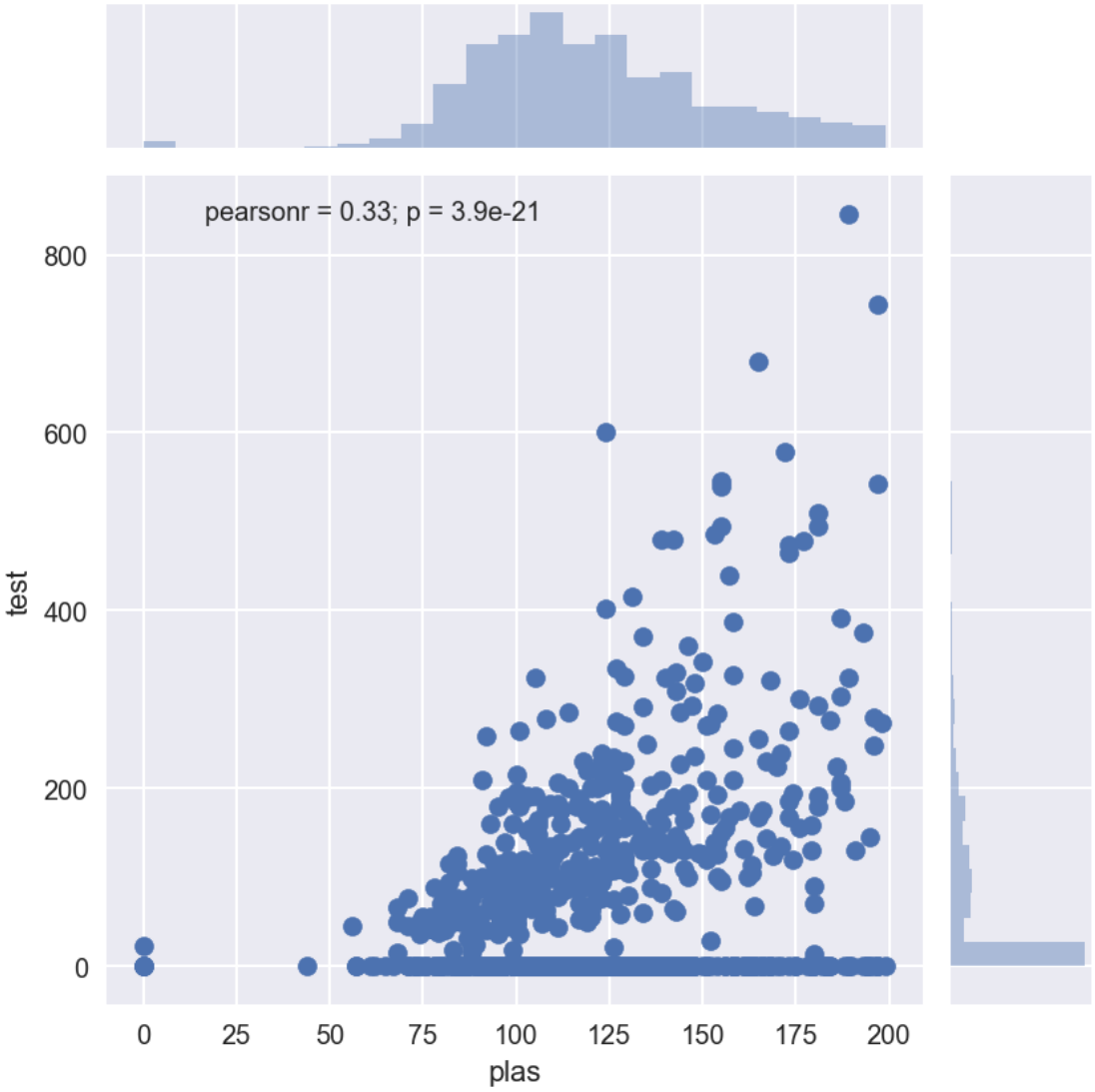

```
In [43]: sbs.jointplot(df["pres"], df["mass"])
```

```
Out[43]: <seaborn.axisgrid.JointGrid at 0x123cd6518>
```



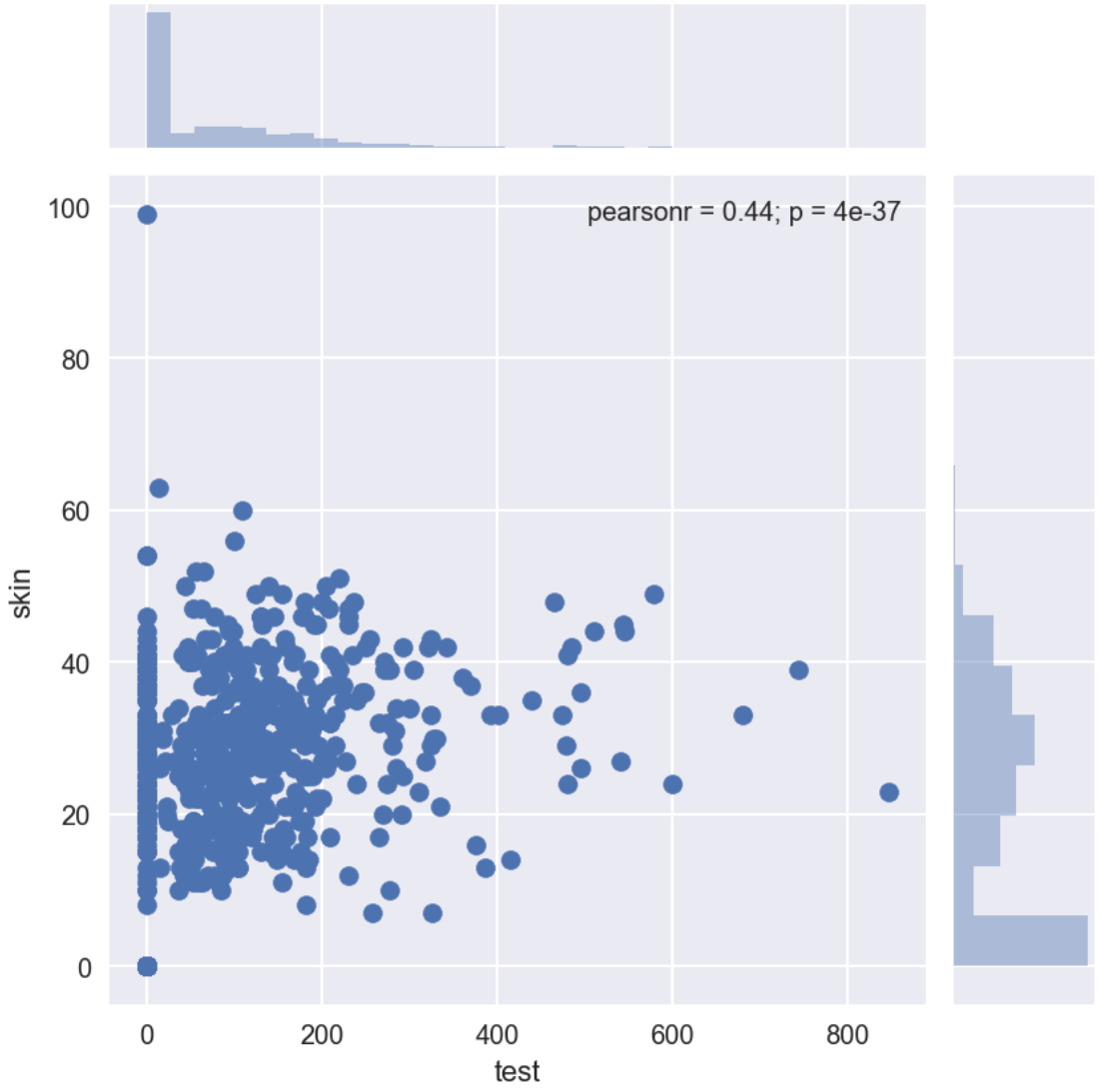

```
In [44]: sbs.jointplot(df["plas"], df["test"])
```

```
Out[44]: <seaborn.axisgrid.JointGrid at 0x124566e80>
```



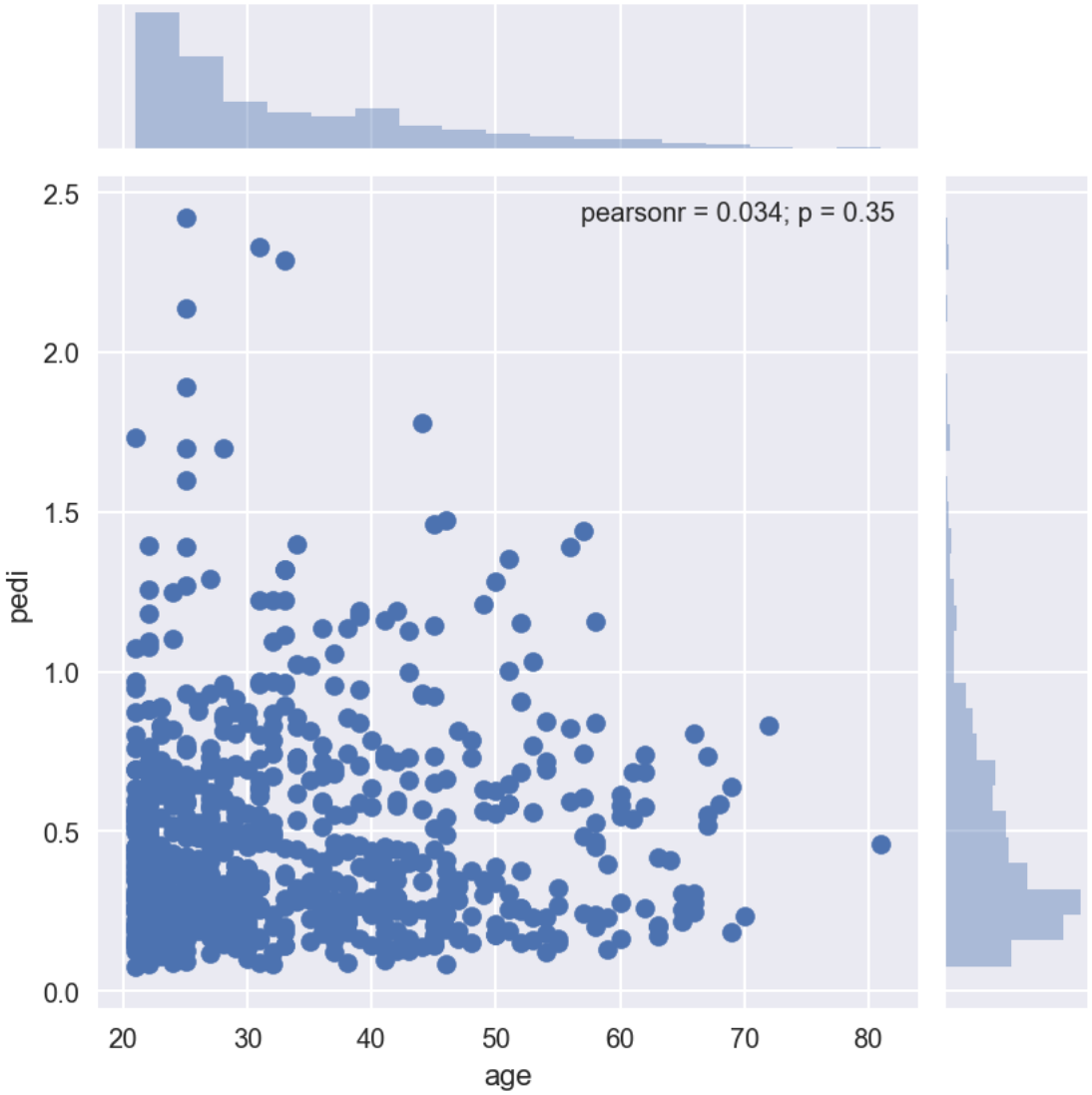

```
In [45]: sbs.jointplot(df["test"], df["skin"])
```

```
Out[45]: <seaborn.axisgrid.JointGrid at 0x124b83eb8>
```




```
In [46]: sbs.jointplot(df["age"], df["pedi"])
```

```
Out[46]: <seaborn.axisgrid.JointGrid at 0x124c78f28>
```




```
In [ ]: #Acá comienza el "hands on"
        #Debemos agrupar el número de embarazos mayores que 6 en una sola categoría
        #Debemos definir qué hacer con los valores 0 de las variables
        #Debemos definir cuáles son las variables relevante con respecto a nuestro target
        #Debemos escoger modelos,tomando en cuenta la correlación entre las variables explicat
```

```
In [ ]:
```

```
In [50]:
```

Out[50]:

	preg	plas	pres	skin	test	mass	pedi	age
0	1	148	72	35	0	33.6	0.627	50
1	0	85	66	29	0	26.6	0.351	31

```
2     1   183    64     0     0  23.3  0.672   32
3     0    89    66    23    94  28.1  0.167   21
4     0   137    40    35   168  43.1  2.288   33
```

In [ ]:
```
#Modelos sin procesamiento
```

In [132]:
```
#Vamos a dejar la data como numpy array que es lo que requiere la libreria SKLEARN
#El fragmento siguiente carga el conjunto de datos de inicio de diabetes de los indi
#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']
df = read_csv(url, names=names)
label = df['class']
df.drop('class', axis=1, inplace=True)
X, y = df, label
y.value_counts()
```

Out[132]:
```
0    500
1    268
Name: class, dtype: int64
```

In [120]:
```
# distributing our Dataset into a training and testing distribution
# we use the default SKL split (0.75 (75%) for training)
# random_state=0 Setting the random seed (for reproductibility purpose)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0,test_size=0
```

In [121]:
```
# counting the split of Positive (1) and Negative (0) labels in our testing distribu
#Es importante comprender cuándo las clases están desbalanceadas, así como también c
#una variable en particular
y_test.value_counts()
```

Out[121]:
```
0    157
1     74
Name: class, dtype: int64
```

In [122]:
```
# El promedio de nuestra clase nos entrega cuál es el mínimo de precisión que se esp
# '1-' will allow us to return the max value
1- y_test.mean()
```

Out[122]:
```
0.6796536796536796
```

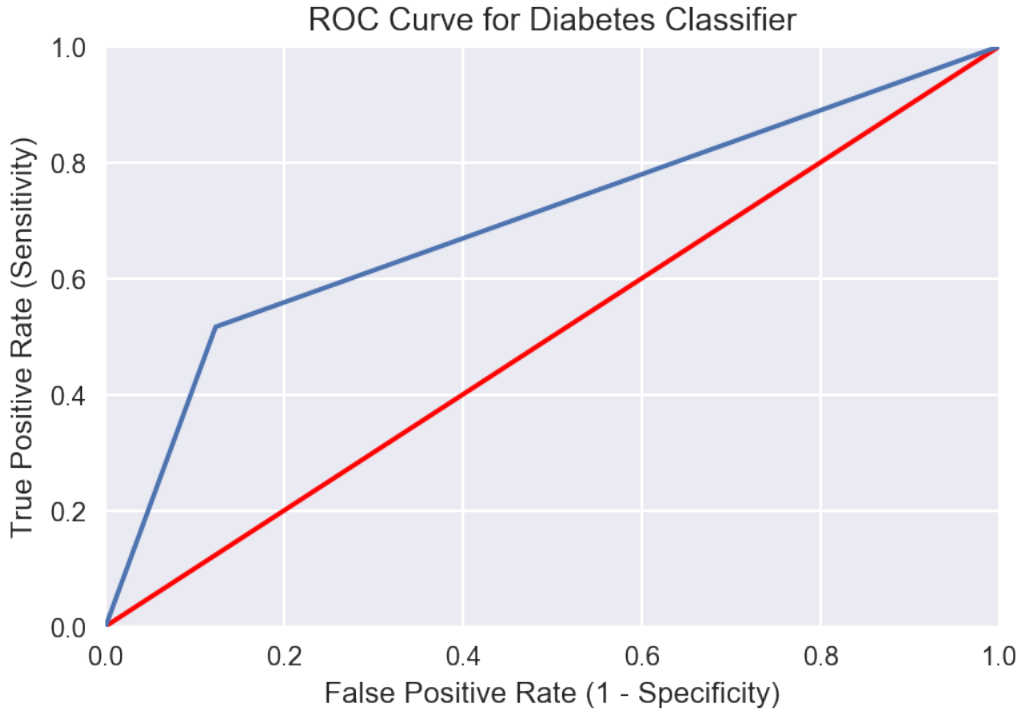
In [149]:
```
#Partimos haciendo el clásico Random Forest
# Create a random forest classifier. By convention, clf means 'classifier'
clf = RandomForestClassifier(n_jobs=2)
# Train the classifier to take the training features and learn how they relate
# to the training y
clf.fit(X_train, y_train)
```

```
Out[149]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_split=1e-07, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=10, n_jobs=2, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)

In [150]: # Apply the classifier we trained to the test data (which, remember, it has never se
          clf.predict(X_test)
          # View the predicted probabilities of the first 10 observations
          clf.predict_proba(X_test)[0:10]

Out[150]: array([[ 0.2,  0.8],
                 [ 0.9,  0.1],
                 [ 1. ,  0. ],
                 [ 0.2,  0.8],
                 [ 0.8,  0.2],
                 [ 0.9,  0.1],
                 [ 0.1,  0.9],
                 [ 0. ,  1. ],
                 [ 0.6,  0.4],
                 [ 0.8,  0.2]])

In [151]: preds = clf.predict(X_test)
          # View the PREDICTED value for the first 10 observations
          preds[0:10]

Out[151]: array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0])

In [156]: # Create confusion matrix
          pd.crosstab(y_test, preds, rownames=['Actual class'], colnames=['Predicted class'],ma

Out[156]: Predicted class    0   1  All
          Actual class
          0                114  16  130
          1                 30  32   62
          All              144  48  192

In [153]: print(metrics.classification_report(y_test, preds))

             precision    recall  f1-score   support

          0       0.79      0.88      0.83       130
          1       0.67      0.52      0.58        62

avg / total       0.75      0.76      0.75       192
```

In [154]:
```
# View a list of the features and their importance scores
list(zip(X_test.columns, clf.feature_importances_))
```

Out[154]:
```
[('preg', 0.093404356335734393),
 ('plas', 0.22927800690026462),
 ('pres', 0.094144447062222109),
 ('skin', 0.070247146441522329),
 ('test', 0.074356490255363741),
 ('mass', 0.16141924867553406),
 ('pedi', 0.11886449050741425),
 ('age', 0.15828581382194448)]
```

In [155]:
```
#Data Visualisation of ROC curve :
#equilibrium between True (tpr) and False (fpr) Positive Rate

import numpy as np
line = np.linspace(0, 1, 2)
plt.pyplot.plot(line, 'r')

fpr, tpr, thresholds = roc_curve(y_test, preds)
plt.pyplot.plot(fpr, tpr)
plt.pyplot.xlim([0.0, 1.0])
plt.pyplot.ylim([0.0, 1.0])
plt.pyplot.title('ROC Curve for Diabetes Classifier')
plt.pyplot.xlabel('False Positive Rate (1 - Specificity)')
plt.pyplot.ylabel('True Positive Rate (Sensitivity)')
plt.pyplot.grid(True)
```



```
In [158]: print(metrics.accuracy_score(y_test, preds))
          print(metrics.roc_auc_score(y_test, preds))
```

```
0.760416666667
0.696526054591
```

```
In [159]: %%time
          # Magic keyword time who display the cpu usage
          # GridSearch procedure = cpu demanding
          gradient_boost_eval = GradientBoostingClassifier(random_state=0)

          params = {
              'learning_rate': [0.05, 0.1, 0.5],
              'max_features': [0.5, 1],
              'max_depth': [3, 4, 5]
          }

          grid_search = GridSearchCV(gradient_boost_eval, params, cv=10, scoring='roc_auc')
          grid_search.fit(X, y)

          print(grid_search.best_params_)
          print(grid_search.best_score_)
```

```
{'learning_rate': 0.05, 'max_depth': 3, 'max_features': 0.5}
0.8338228573124407
CPU times: user 20.6 s, sys: 155 ms, total: 20.7 s
Wall time: 20.9 s
```

```
In [160]: # GBC belong to the Random Tree Classifier family
          # a complex model who perform well on small Dataset
          # We set our hyper-params with the best params providing by GridSearch
          gradient_boost = GradientBoostingClassifier(
              learning_rate=0.05, max_depth=3, max_features=0.5, random_state=0)

          #Cross Validation handle the .fit and .predict method
          cross_val_score(gradient_boost, X, y, cv=10, scoring='roc_auc').mean()
```

```
Out[160]: 0.83383190883190872
```

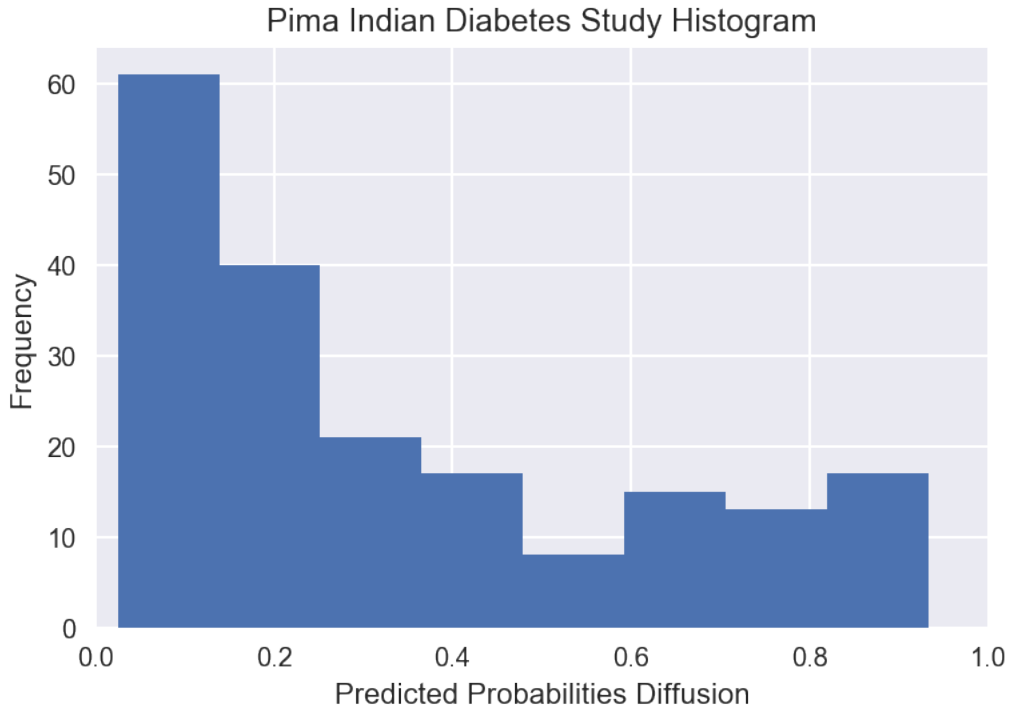
```
In [161]: #Adjusting the Threshold for improvement y Usando la partición de 70-30
          gradient_boost_imp = GradientBoostingClassifier(
              learning_rate=0.05, max_depth=3, max_features=0.5, random_state=0)

          X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

          gradient_boost_imp.fit(X_train, y_train)
```

```
Out[161]: GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.05, loss='deviance', max_depth=3,
                           max_features=0.5, max_leaf_nodes=None,
                           min_impurity_split=1e-07, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=100, presort='auto', random_state=0,
                           subsample=1.0, verbose=0, warm_start=False)
```

```
In [162]: # Storing the predicted probabilities of classification
          y_pred_prob = gradient_boost_imp.predict_proba(X_test)[:, 1]
```

```
In [163]: plt.pyplot.hist(y_pred_prob, bins=8)
          plt.pyplot.xlim(0, 1)
          plt.pyplot.title('Pima Indian Diabetes Study Histogram')
          plt.pyplot.xlabel('Predicted Probabilities Diffusion')
          plt.pyplot.ylabel('Frequency')
```

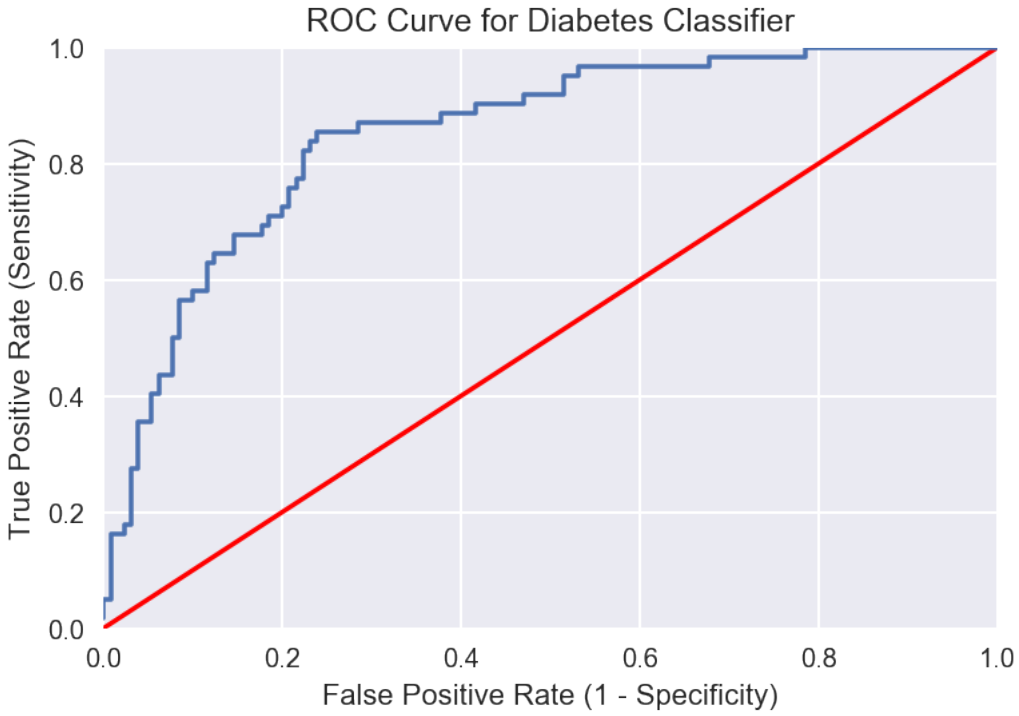
```
Out[163]: <matplotlib.text.Text at 0x126cbe860>
```




```
In [164]: #Adjusting the threshold to 0.1
          y_pred_class = binarize([y_pred_prob], 0.1)[0]
          print(metrics.classification_report(y_test, y_pred_class))

             precision    recall  f1-score   support
```

```
          0       0.96      0.36      0.53       130
          1       0.42      0.97      0.59        62

avg / total       0.78      0.56      0.54       192
```

In [165]:
```
#Data Visualisation of ROC curve :
#equilibrium between True (tpr) and False (fpr) Positive Rate

import numpy as np
line = np.linspace(0, 1, 2)
plt.pyplot.plot(line, 'r')

fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.pyplot.plot(fpr, tpr)
plt.pyplot.xlim([0.0, 1.0])
plt.pyplot.ylim([0.0, 1.0])
plt.pyplot.title('ROC Curve for Diabetes Classifier')
plt.pyplot.xlabel('False Positive Rate (1 - Specificity)')
plt.pyplot.ylabel('True Positive Rate (Sensitivity)')
plt.pyplot.grid(True)
```



In [169]:
```
print(metrics.accuracy_score(y_test, y_pred_class))
print(metrics.roc_auc_score(y_test, y_pred_prob))
```

```
0.557291666667
0.852481389578
```

```
In [ ]: #Pre-procesamiento de la data usando la libreria scikit-learn, el cual provee dos leng
        #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 (range
        #Explorar un feature engineering más avanzado como Binarizing, que transforma a binari
        #For example, the snippet below loads the Pima Indians onset of diabetes dataset, calc
        #to standardize the data, then creates a standardized copy of the input data.
```

```
In [101]: url = "https://goo.gl/vhm1eU"
          names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
          df = read_csv(url, names=names)
```

```
In [102]: #Separamos el arreglo en inputs y outputs, o bien variables explicativas y target
          #array = dataframe.values
          #X = array[:,0:8]
          #Y = array[:,8]
          X, Y = df, label
```

```
In [103]: #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 usand
          #https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.set_printoptions.
          #https://stackoverflow.com/questions/2891790/how-to-pretty-printing-a-numpy-array-wi
          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  1.366]
 [-0.845 -1.123 -0.161  0.531 -0.693 -0.684 -0.365 -0.191 -0.732]
 [ 1.234  1.944 -0.264 -1.288 -0.693 -1.103  0.604 -0.106  1.366]
 [-0.845 -0.998 -0.161  0.155  0.123 -0.494 -0.921 -1.042 -0.732]
 [-1.142  0.504 -1.505  0.907  0.766  1.41   5.485 -0.02   1.366]]
```

```
In [104]: #Tarea optativa: Estandarizar la data númerica usando center o normalizer
```

```
In [105]: #Modelos de predicción
          #Usaremos modelos de regresión logística primero, haciendo énfasis en la dificultad
          #modelos dado la gran cantidad de supuestos que se deben cumplir. Posteriormente usa
          #y generaremos una comparación con los modelos de black box.
```

```
In [106]: #Hacemos una regresión logística usando los datos sin el pre-procesamiento y la técn
          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()*
```

```
Accuracy: %.3f%% (%.3f%%)100.00.0
```

```
In [107]: #Hacemos la regresión logística usando los datos con el pre-procesamiento y la técni
          #Vemos un incremento en la métrica de precisión de un 1.04 puntos porcentuales con u
          #0.16 puntos porcentuales.Este incremento es marginal y nos indica que debemos traba
          #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()*
```

```
Accuracy: %.3f%% (%.3f%%)100.00.0
```

```
In [108]: #Lo que contiene results son las métricas de precisión para cada uno de los 10 data
          #lo que se muestra como resultado en ambos modelos anteriores es el promedio y desvi
          #tante entender el rango en el que se mueven, pues podemos ver que la mínima precisi
          #Si no estamos dispuestos a tolerar la posibilidad de un 68% de precisión, entonces
          results
```

```
Out[108]: array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

```
In [109]: df.columns=['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'clas']
```

```
In [110]: #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 [111]: Y,X= dmatrices(formula_like=f,data=df, 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 [112]: #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: DataCo

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_s
```

```
Out[112]: 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 [113]: #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 [114]: #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 r
          print(metrics.accuracy_score(y_test, predicted))
          print(metrics.roc_auc_score(y_test, probs[:, 1]))
```

```
0.779220779221
0.814167670856
```

```
In [115]: #Mostramos la matriz de confusión donde las filas son, de arriba hacia abajo,persona
          #y personas que si,y las columnas son mi prección que va de izquierda a derecha pers
          #diabetes y personas que si
          print(metrics.confusion_matrix(y_test, predicted))
```

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

```
In [116]: #Finalmente desplegamos un resumen de las métricas donde podemos obersvar que el gra
          #predicción de 1,es decir las personas que si tendrán inicio de diabetes pero yo pre
          #En el contexto de este análisis esto es un resultado malo ya que dejaré de darles e
          #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
```

In [117]:
```
#Data Visualisation of ROC curve :
#equilibrium between True (tpr) and False (fpr) Positive Rate

import numpy as np
line = np.linspace(0, 1, 2)
plt.pyplot.plot(line, 'r')

fpr, tpr, thresholds = roc_curve(y_test,predicted)
plt.pyplot.plot(fpr, tpr)
plt.pyplot.xlim([0.0, 1.0])
plt.pyplot.ylim([0.0, 1.0])
plt.pyplot.title('ROC Curve for Diabetes Classifier')
plt.pyplot.xlabel('False Positive Rate (1 - Specificity)')
plt.pyplot.ylabel('True Positive Rate (Sensitivity)')
plt.pyplot.grid(True)
```