Autor: Prof. Mateus Grellert, Universidade Católica de Pelotas

2 - Análise e Visualização de Dados

Essa etapa consiste em analisar os atributos do nosso data set em busca de informações ruidosas, inconsistências, outliers, etc. Também ajuda a entender quão importante são nossos dados pra predizer nosso desfecho.

Vamos carregar novamente o dataset "Pima Indians Diabetes". (https://www.kaggle.com/uciml/pima-indians-diabetes-database (https://www.kaggle.com/uciml/pima-indians-diabetes-database))

Lembrando que os atributos são:

1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. BloodPressure: Diastolic blood pressure (mm Hg)
4. SkinThickness: Triceps skin fold thickness (mm)
5. Insulin: 2-Hour serum insulin (mu U/ml)
6. BMI: Body mass index (weight in kg/(height in m)^2)
7. DiabetesPedigreeFunction: Diabetes pedigree function
8. Age: Age (years)

Vamos começar com o 5-number summary, que pode ser visualizado pelo método **describe()** do pandas.

In [1]:
```
import pandas as pd

dataset_original = pd.read_csv('../BANCOS/pima_diabetes.csv', sep =
',')

# o comando describe mostra um resumo estatistico do banco
print "\n5-number summary"

dataset_original.describe()
```

```
5-number summary
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Di
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	

Analisando esses valores, já podemos ter uma ideia sobre a forma como cada variável se comporta. Por exemplo, sabemos que nenhuma distribuição é 100% simétrica, pois a média e a mediana (quartil 50%) não são as mesmas em nenhum caso.

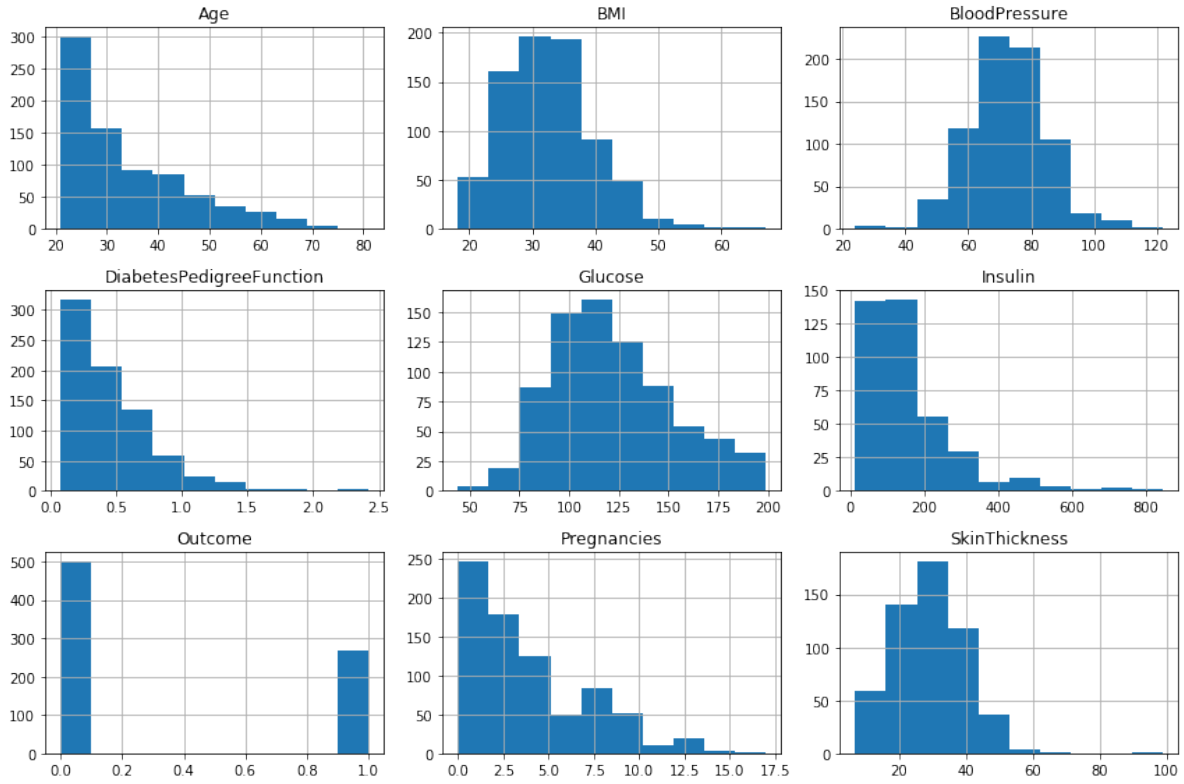
Também podemos ver que 75% da nossa amostra teve entre 1 e 6 gravidezes, atingindo um máximo de 17 (possível outlier).

Também conseguimos percerber algo que já tínhamos visto na aula anterior: temos valores inválidos. Vamos substituí-los por NaN.

In [2]:
```
import numpy as np

for col in ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
'BMI']:
    dataset_original[col] = dataset_original[col].replace(0,np.nan)
```

Antes de imputar, no entanto, vamos observar quais são as distruições de cada dado. Para isso, vamos usar o método **hist()** do pandas.

In [5]:
```
# melhor amigo da plotagem: matplotlib.pyplot
import matplotlib.pyplot as plt

#criando uma área de 12 de largura e 8 de altura para plotagem
fig,ax = plt.subplots(figsize = (12,8))

#passando o ax para o hist, forçamos que o pandas plote na área 12x
8 (área maior que o padrão)
# se não passarmos a figura vai ficar pequena
dataset_original.hist(ax = ax)


#médoto show() mostra a figura na tela
plt.tight_layout()
plt.show()
```

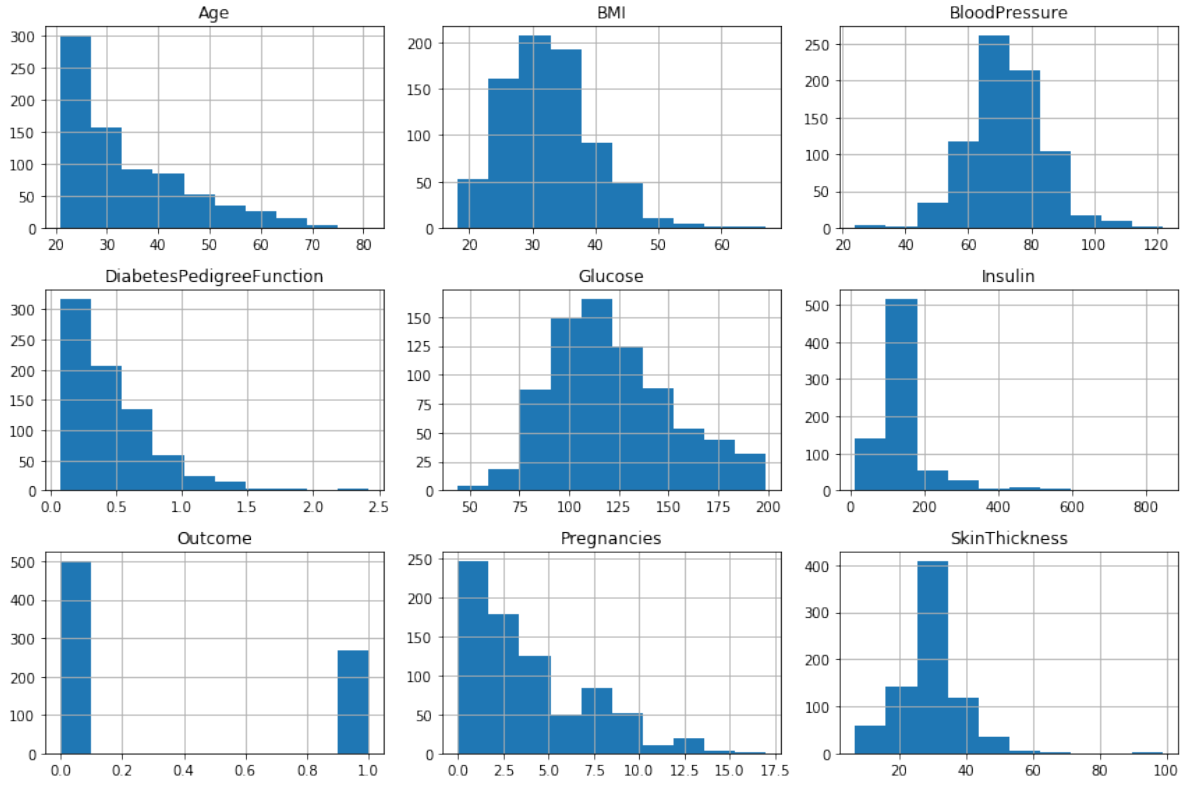


Podemos ver que todas as variáveis (com exceção de Outcome) seguem uma distribuição próxima da normal ou deslocada para a esquerda (skewed left), então vamos usar a **mediana** para imputar os valores faltantes, porque a mediana funciona bem nos dois casos. O método para isso é o **median()**.

In [7]:
```
dataset_removido = dataset_original.dropna() # nesse dataset as linhas com NaN vão ser removidas«

dataset_imputado = dataset_original.copy() # copiando original para imputar depois

for col in ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']:
    mediana = dataset_imputado[col].median()
    dataset_imputado[col] = dataset_imputado[col].fillna(mediana)
    print 'substituindo valores faltando na coluna %s por %f' % (col, mediana)

print '\nDataset com imputação: %d amostras e %d colunas' % dataset_imputado.shape
print 'Dataset com remoção:   %d amostras e %d colunas' % dataset_removido.shape
```

```
substituindo valores faltando na coluna Glucose por 117.000000
substituindo valores faltando na coluna BloodPressure por 72.000000
substituindo valores faltando na coluna SkinThickness por 29.000000
substituindo valores faltando na coluna Insulin por 125.000000
substituindo valores faltando na coluna BMI por 32.300000

Dataset com imputação: 768 amostras e 9 colunas
Dataset com remoção:   392 amostras e 9 colunas
```

In [8]:
```
import matplotlib.pyplot as plt
fig,ax = plt.subplots(figsize = (12,8))
dataset_imputado.hist(ax = ax)


plt.tight_layout()
plt.show();
```



Também podemos ver que nossa variável desfecho (Outcome) é desbalanceada, ou seja, existem mais exemplos negativos do que positivos. Dependendo do caso, pode ser interessante balancear os nossos exemplos para eliminar o bias do desfecho. Pense que se tivermos muitos exemplos de casos negativos, nossos modelos preditivos podem acabar dando preferência para acertar nesses casos. No entanto, acertar um diagnóstico positivo pode ser muito mais importante. Para eliminar esse bias, balanceamos os exemplos usando o método **RandomUnderSampler** da biiblioteca **imblearn**.

In [14]:
```
from imblearn.under_sampling import RandomUnderSampler

# primeiro vamos separar nosso pandas Dataframe em atributos e desf
echo
X = dataset_imputado.loc[:,dataset_imputado.columns != 'Outcome']
y = dataset_imputado['Outcome']

Xb, yb = RandomUnderSampler().fit_sample(X,y)


# criamos agora o dataset balanceado agrupando novamento as colunas
dataset_balanceado = pd.DataFrame(Xb, columns = dataset_imputado.co
lumns[:-1])
dataset_balanceado["Outcome"] = yb

print 'Dataset com imputação:            %d amostras e %d colunas'
% dataset_imputado.shape
print 'Dataset com imputação balanceado: %d amostras e %d colunas'
% dataset_balanceado.shape
```

```
Dataset com imputação:            768 amostras e 9 colunas
Dataset com imputação balanceado: 536 amostras e 9 colunas
```

Vamos usar o teste **D'Agostino K**^{2} para detecção de normalidade de uma distribuição, implementado pelo método **normaltest(dist)** da biblioteca **scipy.stats**.

Esse teste retorna dois valores: um valor que representa o deslocamento e o kurtosis (não nos interessa agora); e um valor que representa o p-value. O p-value indica a probabilidade da variável em questão seguir uma distribuição normal.

Usamos normalmente o valor 5% de threshold α (Intervalo de Confiança de 95%). Em testes estatísticos, esse valor representa a chance de ocorrer um Erro Tipo I (falso positivo).

A normalidade serve para sabermos que tipo de método usar. Basicamente, usa-se a regra:

Se (distribuição normal):

- Métodos paramétricos

Se não:

- Métodos não paramétricos

Para media a correlação, por exemplo, temos o método paramétrico (Pearson) e o não paramétrico (Spearman).

No entanto, alguns testes paramétricos podem ser aplicados a distribuições não paramétricas desde que o número de amostras seja grande (> algumas dezenas), ou ainda desde que seja uma distribuição simétrica. No fim das contas sempre vale a pena estudar o método e saber suas limitações.

In [15]:
```
from scipy.stats import shapiro
from scipy.stats import normaltest

for col in dataset_balanceado.columns[:-1]:
    stat, p = normaltest(dataset_balanceado[col])
    #print('Statistics=%.3f, p=%.3f' % (stat, p))
    alpha = 0.05
    if p > alpha: # H0: a distribuição é normal
        print 'Distribuição %s parece Gaussiana (falha ao rejeitar
H0)' % (col)
    else:
        print 'Distribuição %s não parece Gaussiana (rejeita H0)' %
(col)
```

```
Distribuição Pregnancies não parece Gaussiana (rejeita H0)
Distribuição Glucose não parece Gaussiana (rejeita H0)
Distribuição BloodPressure não parece Gaussiana (rejeita H0)
Distribuição SkinThickness não parece Gaussiana (rejeita H0)
Distribuição Insulin não parece Gaussiana (rejeita H0)
Distribuição BMI não parece Gaussiana (rejeita H0)
Distribuição DiabetesPedigreeFunction não parece Gaussiana (rejeit
a H0)
Distribuição Age não parece Gaussiana (rejeita H0)
```

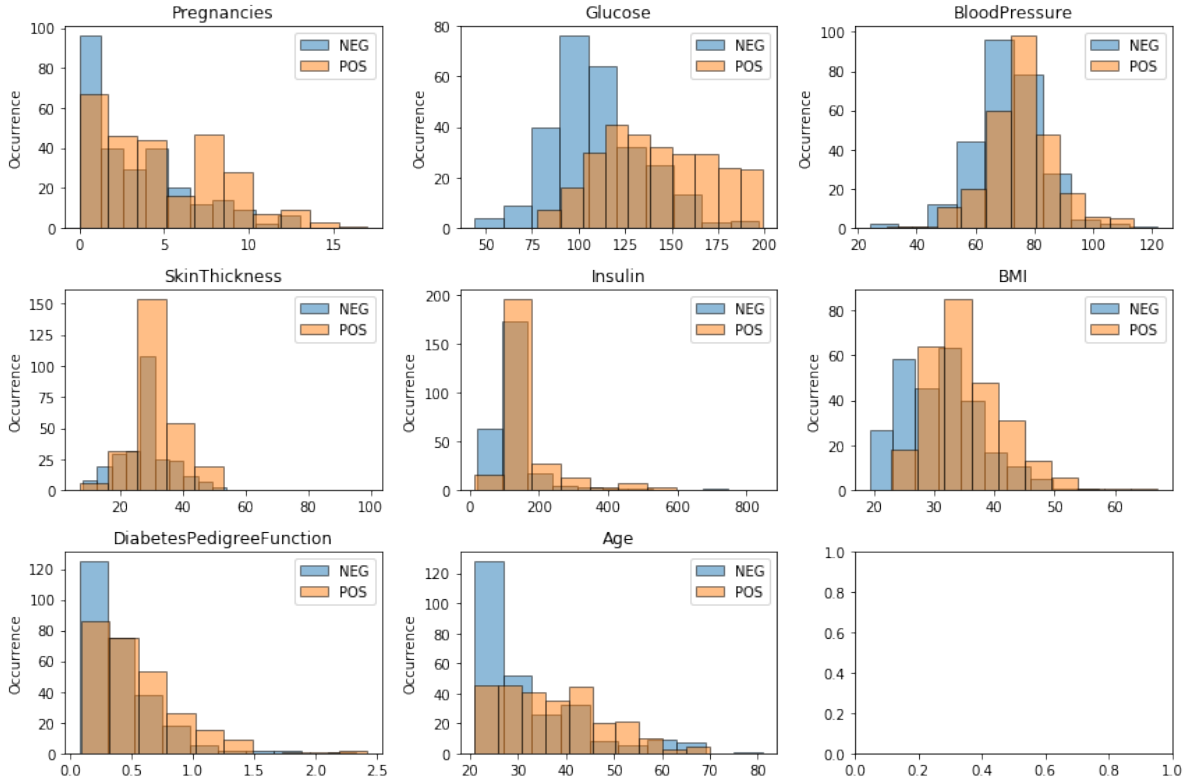
Nenhuma distribuição é paramétrica! Isso não necessariamente indica que esteja tudo perdido. Só temos que tomar cuidado com os testes estatísticos e as conclusões que vamos tomar com base neles. Alguns testes que dependem de normalidade podem ainda ser aplicados a distribuições não-paramétricas desde que o número de amostras seja maior que um certo valor (normalmente mais que algumas dezenas).

Agora vamos fazer uma análise separando nossa amostra em dois grupos: Casos Positivos (POS), nos quais a coluna Outcome vale 1; e casos Negativos (NEG), com Outcome = 0.

Isso vai ajudar a enxergar como fica o comportamento das nossas variáveis em cada grupo, o que nos ajuda a detectar mudanças em cada caso. Isso vai ficar mais claro depois do exemplo.

In [16]:
```
import matplotlib.pyplot as plt
from scipy.stats import anderson

fig,axes = plt.subplots(3,3,figsize = (12,8))

i = j = 0
for idx, col in enumerate(dataset_balanceado.columns[:-1]):
    # Grupo 0: Casos NEGATIVOS (NEG)
    tmp0 = dataset_balanceado.loc[dataset_balanceado.Outcome == 0,
col]
    # Grupo 1: Casos POSITIVOS (POS)
    tmp1 = dataset_balanceado.loc[dataset_balanceado.Outcome == 1,
col]

    stat, p = normaltest(tmp0)
    alpha = 0.05
    if p > alpha:
        print 'Distribuição %s parece Gaussiana (falha ao rejeitar
H0), group NEG' % (col)
    else:
        print 'Distribuição %s não parece Gaussiana (rejeita H0), g
roup NEG' % (col)

    stat, p = normaltest(tmp1)
    alpha = 0.05
    if p > alpha:
        print 'Distribuição %s parece Gaussiana (falha ao rejeitar
H0), group POS' % (col)
    else:
        print 'Distribuição %s não parece Gaussiana (rejeita H0), g
roup POS' % (col)



    i = idx / 3
    j = idx % 3

    axes[i][j].hist(tmp0, label ='NEG', alpha = 0.5, lw = 1,edgecol
or='black')
    axes[i][j].hist(tmp1, label = 'POS', alpha = 0.5, lw = 1,edgeco
lor='black')
    axes[i][j].set_title(col)
    axes[i][j].set_ylabel('Occurrence')
    axes[i][j].legend()



plt.tight_layout()
plt.show();
```

```
Distribuição Pregnancies não parece Gaussiana (rejeita H0), group
NEG
Distribuição Pregnancies não parece Gaussiana (rejeita H0), group
POS
Distribuição Glucose não parece Gaussiana (rejeita H0), group NEG
Distribuição Glucose não parece Gaussiana (rejeita H0), group POS
Distribuição BloodPressure não parece Gaussiana (rejeita H0), grou
p NEG
Distribuição BloodPressure não parece Gaussiana (rejeita H0), grou
p POS
Distribuição SkinThickness parece Gaussiana (falha ao rejeitar H0)
, group NEG
Distribuição SkinThickness não parece Gaussiana (rejeita H0), grou
p POS
Distribuição Insulin não parece Gaussiana (rejeita H0), group NEG
Distribuição Insulin não parece Gaussiana (rejeita H0), group POS
Distribuição BMI não parece Gaussiana (rejeita H0), group NEG
Distribuição BMI não parece Gaussiana (rejeita H0), group POS
Distribuição DiabetesPedigreeFunction não parece Gaussiana (rejeit
a H0), group NEG
Distribuição DiabetesPedigreeFunction não parece Gaussiana (rejeit
a H0), group POS
Distribuição Age não parece Gaussiana (rejeita H0), group NEG
Distribuição Age não parece Gaussiana (rejeita H0), group POS
```



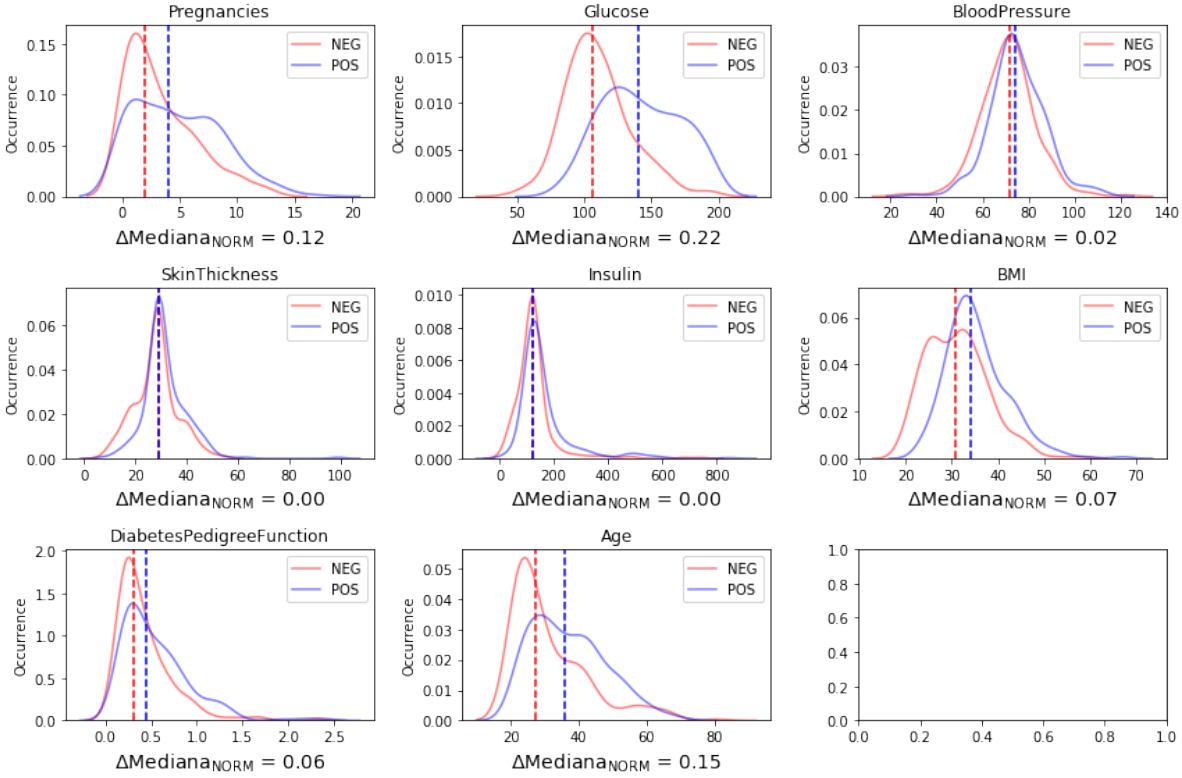
Podemos concluir algumas coisas com esses gráficos:

- Existe uma ocorrência maior de diabetes em mulheres com:
  - mais de 5 gravidezes
  - nível de glicose superior a 150
  - mais de 40 anos
- Também observamos que nenhuma distribuição é normal mesmo quando separamos os dados por grupos (POS e NEG). No entanto, o nível de glicose segue uma distribuição simétrica para o grupo NEG, enquanto que no grupo POS ela é próxima de uma distribuição bem mais espalhada no intervalo ~110-200

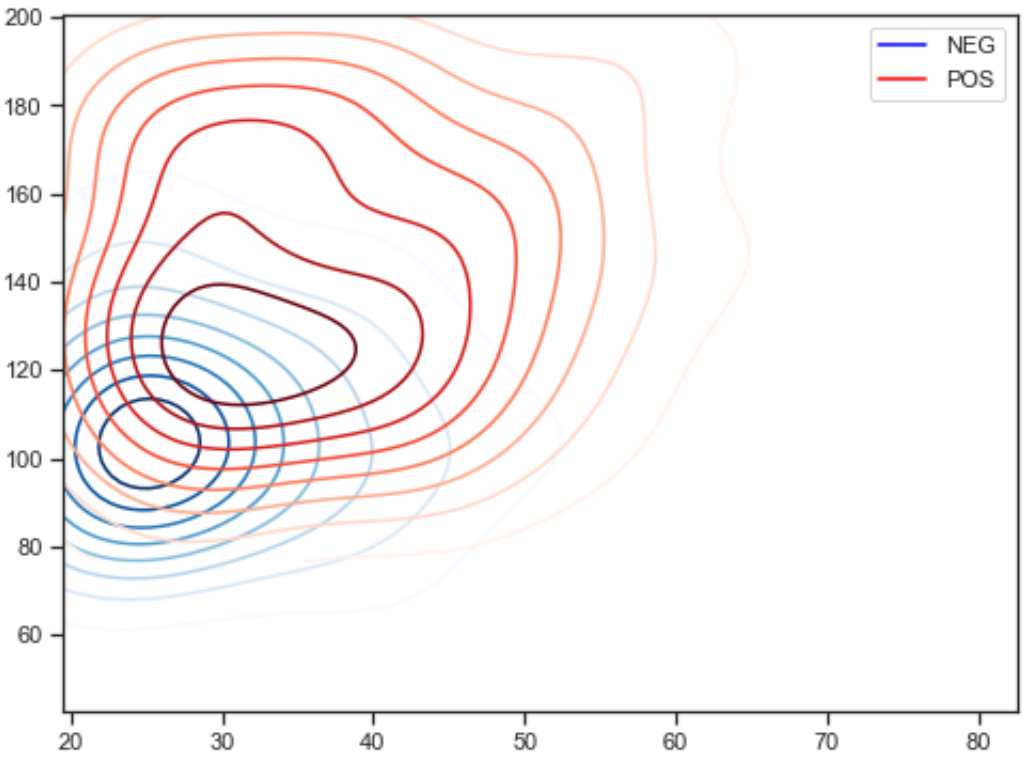
Vamos agora plotar as PDFs dos grupos ao invés do histograma. Vamos aproveitar e plotar também as linhas que indicam a mediana (quartil 50%, ou seja, o ponto que divide a ocorrência das amostras igualmente em cada lado).

Fazendo as diferenças dessas medianas, poderemos concluir que variáveis são se relacionam com a diabete. Para computar a PDF de uma série S, usamos o método **kdeplot(S)** da biblioteca **seaborn**.

In [37]:
```
import seaborn as sns

fig,axes = plt.subplots(3,3,figsize = (12,8))

i = j = 0
for idx, col in enumerate(dataset_balanceado.columns[:-1]):
    tmp0 = dataset_balanceado.loc[dataset_balanceado.Outcome == 0,
col]
    tmp1 = dataset_balanceado.loc[dataset_balanceado.Outcome == 1,
col]

    i = idx / 3
    j = idx % 3

    sns.kdeplot(tmp0, ax = axes[i][j], label ='NEG', alpha = 0.5,c=
'r')
    sns.kdeplot(tmp1,ax = axes[i][j], label ='POS', alpha = 0.5,c='
b')

    # cálculo das medianas e plotagem com a função axvline(x = medi
ana)
    median0 = tmp0.median()
    median1 = tmp1.median()

    axes[i][j].axvline(x=median0,c='r', linestyle='--')
    axes[i][j].axvline(x=median1,c='b', linestyle='--')

    axes[i][j].set_title(col)
    axes[i][j].set_ylabel('Occurrence')
    axes[i][j].legend()

    # vamos adicionar uma caixinha com a distância entre as mediana
s
    # vamor normalizar para interpretar essa distância independente
da escala

    diff_med = abs(median0 - median1)
    min_val = dataset_balanceado[col].min()
    max_val = dataset_balanceado[col].max()
    diff_med_norm = (diff_med)/(max_val - min_val)

    axes[i][j].set_xlabel( r'$\Delta$Mediana$_{\mathrm{NORM}}$ = %.
2f' % (diff_med_norm), fontsize=14)



plt.tight_layout()
plt.show();
```


Pregnancies
NEG
POS
Occurrence
ΔMediana$_{NORM}$ = 0.12
Glucose
ΔMediana$_{NORM}$ = 0.22
BloodPressure
ΔMediana$_{NORM}$ = 0.02
SkinThickness
ΔMediana$_{NORM}$ = 0.00
Insulin
ΔMediana$_{NORM}$ = 0.00
BMI
ΔMediana$_{NORM}$ = 0.07
DiabetesPedigreeFunction
ΔMediana$_{NORM}$ = 0.06
Age
ΔMediana$_{NORM}$ = 0.15

In [120]:
```
import scipy

fig,axes = plt.subplots(figsize = (8,6))

tmp0 = dataset_balanceado.loc[dataset_balanceado.Outcome == 0, ['Age', 'Glucose']]
tmp1 = dataset_balanceado.loc[dataset_balanceado.Outcome == 1, ['Age', 'Glucose']]


p0 = sns.kdeplot(tmp0, ax = axes, label ='NEG', alpha = 1,cmap="Blues", legend= True,  bw=.5)
p1 = sns.kdeplot(tmp1, ax = axes, label ='POS', alpha = 1,cmap="Reds" , legend=True,  bw=.5)

# gambiarras para deixar a legnda na cor certa
axes.legend()
leg = axes.get_legend()
leg.legendHandles[0].set_color('blue')
leg.legendHandles[1].set_color('red')

plt.show()
```



O comando **pairplot()** do seaborn faz duas coisas importantes em uma única vez: para cada par de atributos, ele computa o plot de dispersão separado por grupos (usando o parâmetro **hue**).

Nas diagonais (mesmo atributo), ele plota a PDF para cada grupo (afinal, nas diagonais só temos um atributo).

Muito útil para já de cara enxergar atributos importantes.

In [101]:
```
sns.set(style="ticks")
attributes = dataset_balanceado.columns.drop("Outcome")
sns.pairplot(dataset_balanceado, hue="Outcome", vars=attributes)
```

Out[101]: <seaborn.axisgrid.PairGrid at 0x12329d590>

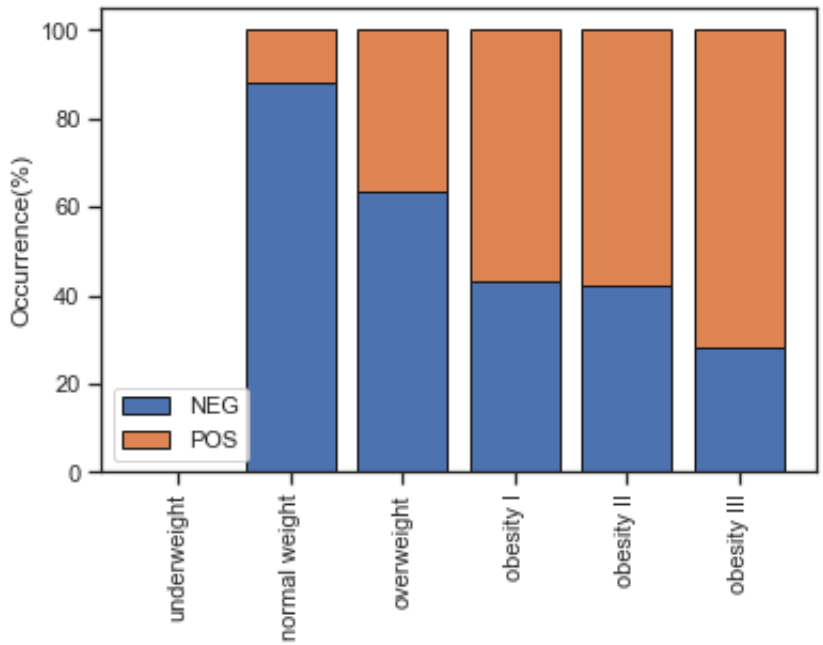
Sabe-se que o BMI está relacionado com o disgnóstico de diabetes, mas isso não fica visualmente claro pelo histograma. Para evidenciar essa relação, podemos criar uma nova feature que torna o valor número BMI para um valor categórico. Para isso, podemos usar a classificação de obesidade da OMS:

- < 18.5 underweight
- 18.5–24.9 normal weight
- 25.0–29.9 overweight
- 30.0–34.9 class I obesity
- 35.0–39.9 class II obesity
- ≥ 40.0 class III obesity

In [121]:
```
def bmi_cat(x):
    if x < 18.5:
        return 0 #'underweight'
    if 18.5 <= x and  x < 25:
        return 1 #'normal weight'
    if 25 <= x and x < 30:
        return 2 #'overweight'
    if 30 <= x and x < 35:
        return 3 #'obesity I'
    if 35 <= x and x <= 40:
        return 4 #'obesity II'
    else:
        return 5 #'obesity III'



dataset_balanceado['BMI_cat'] = dataset_balanceado['BMI'].apply(bmi
_cat)
```

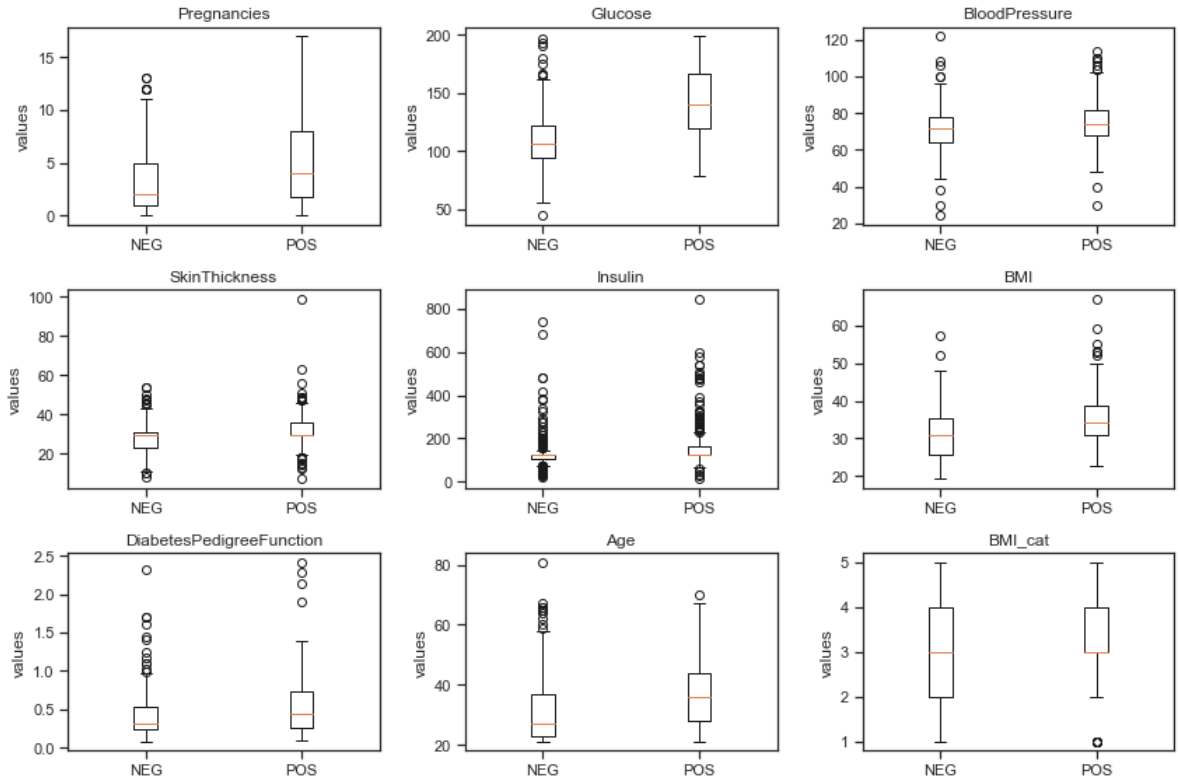
Uma forma interessante de visualizar uma variável categórica é usando o gráfico de barras, no qual cada barra representa o valor V da variável, e é dividida para cada grupo. Normalizando isso pelo número total de ocorrências, temos uma porcentagem. Fica mais claro vendo o exemplo abaixo.

In [130]:
```
tmp0 = dataset_balanceado.loc[dataset_balanceado.Outcome == 0, 'BMI
_cat']
tmp1 = dataset_balanceado.loc[dataset_balanceado.Outcome == 1, 'BMI
_cat']

vals = ['underweight', 'normal weight','overweight', 'obesity I', '
obesity II', 'obesity III']
counts0 = []
counts1 = []

# para cada valor de BMI_cat
for val in range(0,6):
    # total de valores V para o grupo NEG
    count0 = np.sum(tmp0 == val)
    # total de valores V para o grupo POS
    count1 = np.sum(tmp1 == val)
    # total de valores V para os dois grupos + um valorm mto pequen
o para não dar erro na divisão
    total = float(np.sum(dataset_balanceado['BMI_cat'] == val))+0.0
0001

    counts0.append(count0/total*100)
    counts1.append(count1/total*100)


plt.bar(range(len(vals)), counts0, label = 'NEG', lw = 1, edgecolor
= 'k')
plt.bar(range(len(vals)), counts1, bottom = counts0, label = 'POS',
lw = 1, edgecolor= 'k')
plt.xticks(range(len(vals)), vals, rotation = 90)
plt.legend()
plt.ylabel('Occurrence(%)')
plt.show()
```



Notem no nosso dataset nenhuma mulher foi categorizada como abaixo do peso (underweight), por isso a barra acabou não aparecendo. Se não tivéssemos somado um valor muito pequeno no denominador (+0.00001), geraríamos um erro nesse caso!

In [131]:
```
import matplotlib.pyplot as plt
fig,axes = plt.subplots(3,3,figsize = (12,8))

i = j = 0
idx = 0
for col in dataset_balanceado.columns:
    if col == 'Outcome': continue

    tmp0 = dataset_balanceado.loc[dataset_balanceado.Outcome == 0,
col]
    tmp1 = dataset_balanceado.loc[dataset_balanceado.Outcome == 1,
col]

    i = idx / 3
    j = idx % 3
    idx += 1

    axes[i][j].boxplot([tmp0, tmp1])

    axes[i][j].set_title(col)
    axes[i][j].set_xticklabels(['NEG','POS'])
    axes[i][j].set_ylabel('values')



plt.tight_layout()
plt.show();
```



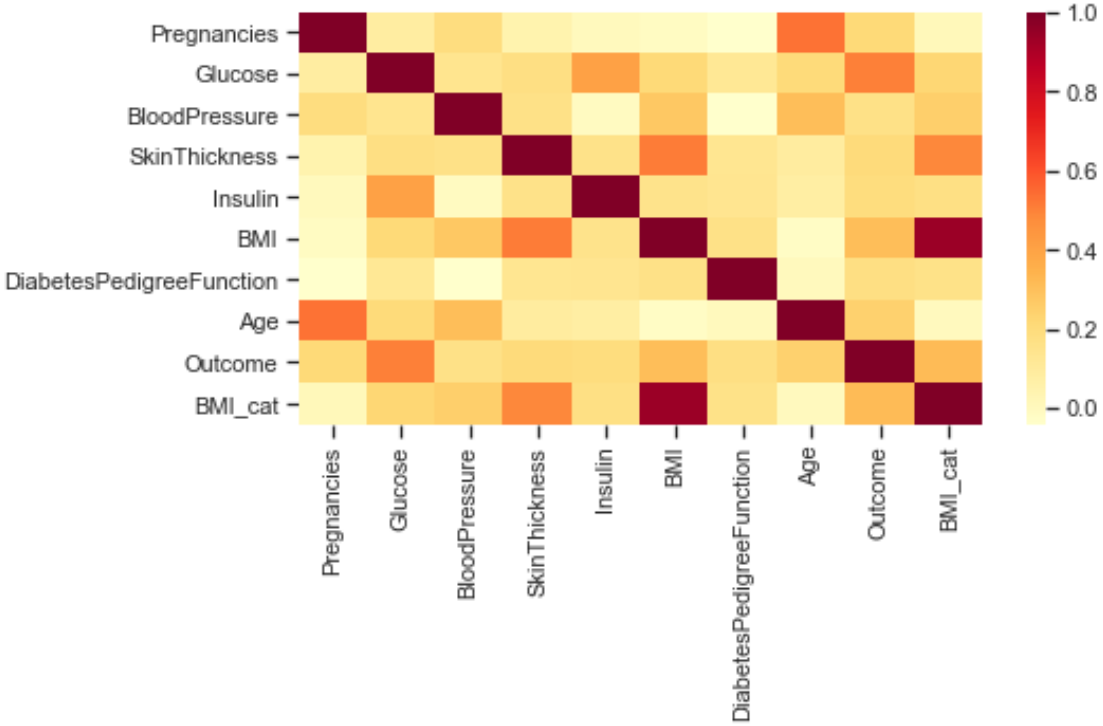
Vamos agora computar a correlação entre os atributos e entre a variável desfecho (Outcome). Como vamos calcular para cada par, vamos acabar gerando uma matriz de correlação.

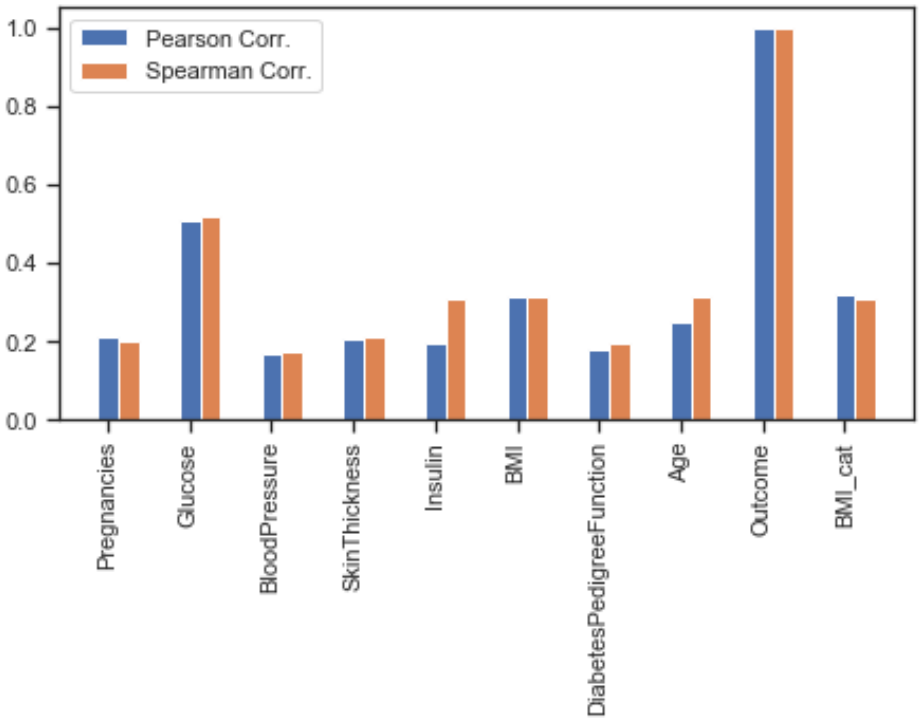
Esse método já existe no pandas e se chama **corr()**. Se chamarmos esse método para um DataFrame, ele vai computar todas as correlações entre as colunas.

Para plotar uma matriz 2D de correlação, um método interessante é o heatmap do seaborn, que gera códigos de calor dependendo do valor de cada posição da matriz. Correlações maiores têm uma cor mais forte.

Nota: esse método usa a correlação de Pearson para o cálculo das correlações. A correlação de Spearman pode ser computada usando o método **df[coluna].corr(method="spearman")**.

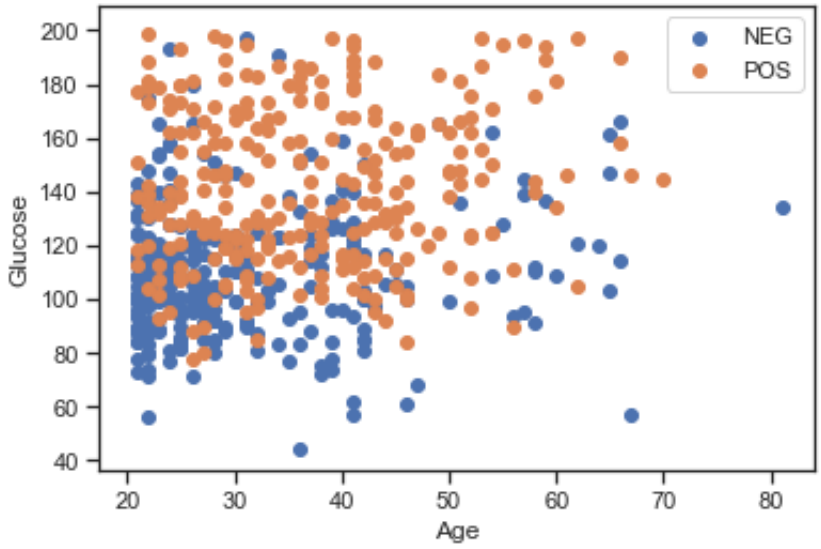
In [148]:
```
import seaborn as sns

fig, axes = plt.subplots(2,1,figsize=(8,10))

# plotando as correlações entre atributos usando Pearson
sns.heatmap(dataset_balanceado.corr(), cmap = 'YlOrRd', ax = axes[0
])


corr_pearson_vet = []
corr_spearman_vet = []

# para calcular Spearman, vamos ter que percorrer todas as colunas
e fazer caso a caso
for col in dataset_balanceado.columns:
    corr_pearson = dataset_balanceado[col].corr(dataset_balanceado[
'Outcome'], method = "pearson")
    corr_spearman = dataset_balanceado[col].corr(dataset_balanceado
['Outcome'], method = "spearman")

    corr_pearson_vet.append(corr_pearson)
    corr_spearman_vet.append(corr_spearman)


# aqui vamos plotar as barras com as correlações de Pearson e Spear
man
axes[1].bar(np.arange(corr_outcome.shape[0]),corr_pearson_vet, widt
h=0.25, label = "Pearson Corr.")
axes[1].bar(np.arange(corr_outcome.shape[0])+0.25,corr_spearman_vet
, width=0.25, label = "Spearman Corr.")

# aqui vamos renomear os rótulos do eixo X para colocar os nomes do
s nossos atributos
axes[1].set_xticks(range(dataset_balanceado.columns.shape[0]))
axes[1].set_xticklabels(dataset_balanceado.columns, rotation=90)

axes[1].legend()

plt.tight_layout()
plt.show()
```



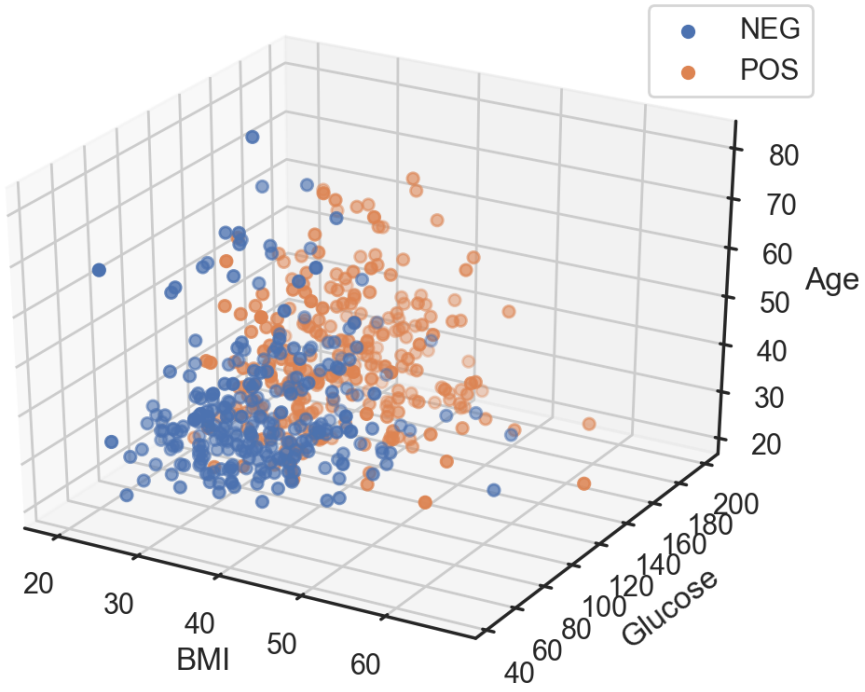


Os exemplos abaixo apresentam outras formas de visualizar os atributos que podem ser úteis.

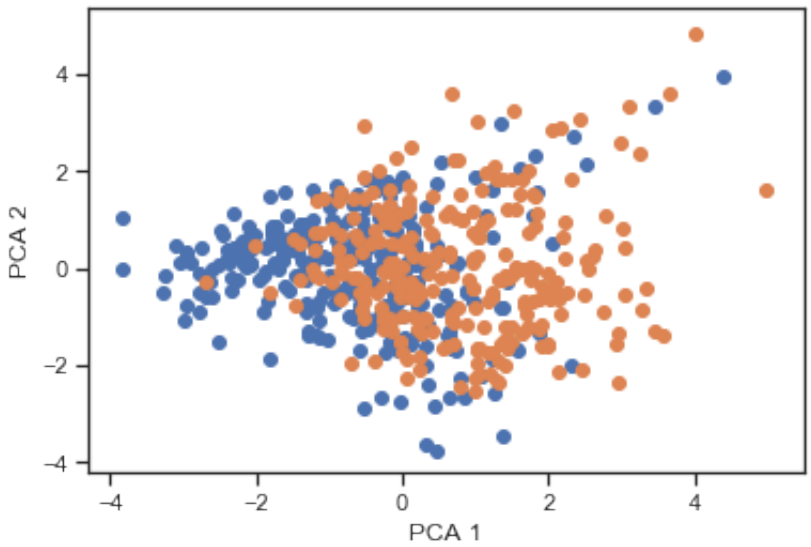
In [150]:
```
tmp0 = dataset_balanceado.loc[dataset_balanceado.Outcome == 0 ]
tmp1 = dataset_balanceado.loc[dataset_balanceado.Outcome == 1 ]

plt.scatter(tmp0['Age'],tmp0['Glucose'], label = 'NEG' )
plt.scatter(tmp1['Age'],tmp1['Glucose'], label = 'POS' )

plt.xlabel('Age')
plt.ylabel('Glucose')
plt.legend()
plt.show()
```



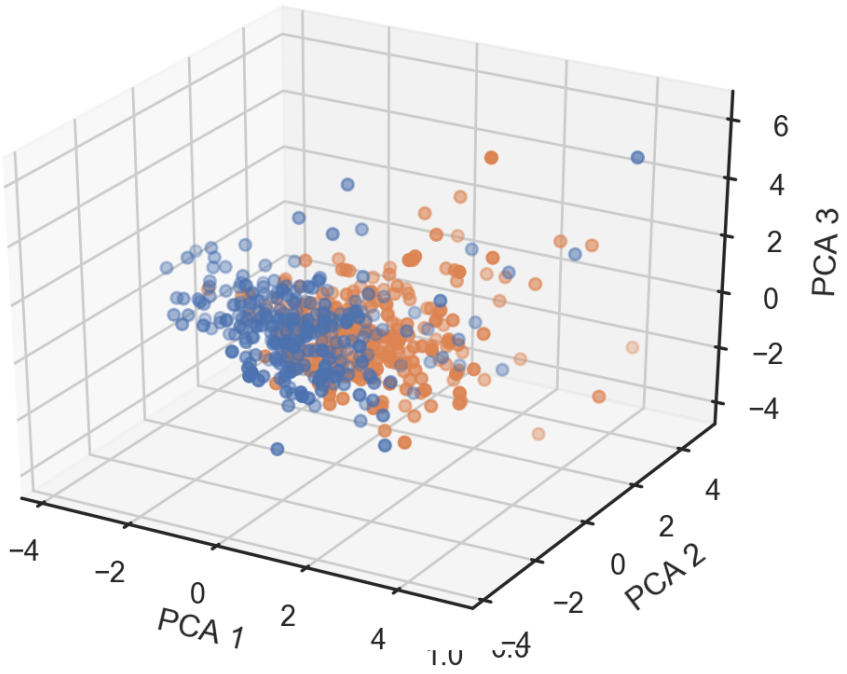
In [151]:
```
%matplotlib notebook


from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

tmp0 = dataset_balanceado.loc[dataset_balanceado.Outcome == 0 ]
tmp1 = dataset_balanceado.loc[dataset_balanceado.Outcome == 1 ]

ax.scatter(tmp0['BMI'],tmp0['Glucose'], tmp0['Age'], label = 'NEG'
)
ax.scatter(tmp1['BMI'],tmp1['Glucose'], tmp1['Age'], label = 'POS'
)

ax.set_xlabel('BMI')
ax.set_ylabel('Glucose')
ax.set_zlabel('Age')

plt.legend()
plt.show()
```



In [152]:
```
from sklearn.decomposition import PCA
%matplotlib inline

fig = plt.figure()
ax = fig.add_subplot(111)

X = dataset_balanceado.loc[:,dataset_imputado.columns != 'Outcome']
y = dataset_balanceado['Outcome']

from sklearn.preprocessing import StandardScaler
Xscale = StandardScaler().fit_transform(X)

pca = PCA(n_components=2)
pca = pca.fit(Xscale)
Xtr = pca.transform(Xscale)

Xtr0 = Xtr[y==0,:]
Xtr1 = Xtr[y==1,:]


ax.scatter(Xtr0[:,0],Xtr0[:,1], label = 'NEG' )
ax.scatter(Xtr1[:,0],Xtr1[:,1], label = 'POS' )


ax.set_xlabel('PCA 1')
ax.set_ylabel('PCA 2')

plt.show()
```



In [153]:
```
%matplotlib notebook
from sklearn.decomposition import PCA

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

pca = PCA(n_components=3)
pca = pca.fit(Xscale)
Xtr = pca.transform(Xscale)

ax = fig.add_subplot(111, projection='3d')

Xtr0 = Xtr[y==0,:]
Xtr1 = Xtr[y==1,:]


ax.scatter(Xtr0[:,0],Xtr0[:,1], Xtr0[:,2], label = 'NEG' )
ax.scatter(Xtr1[:,0],Xtr1[:,1], Xtr1[:,2], label = 'POS' )


ax.set_xlabel('PCA 1')
ax.set_ylabel('PCA 2')
ax.set_zlabel('PCA 3')

plt.show()
```



In [ ]: