Autor: Prof. Mateus Grellert, Universidade Católica de Pelotas

In [1]:
```
# esse trecho de código vai aparecer em todos os notebooks para sup
rimir warnings chatos de versão numpy
import matplotlib.pyplot as plt
%matplotlib inline

import warnings

warnings.filterwarnings("ignore", message="numpy.dtype size changed
")
warnings.filterwarnings("ignore", message="numpy.ufunc size changed
")
```

1 - Pré-processamento de Dados

Limpeza

Essa etapa consiste em tratar valores inválidos/ruidosos no nosso banco para que eles não influenciem na etapa posterior de análise e modelagem.

Dados inválidos incluem valores nulos, outliers ou mesmo valores que não fazem sentido para uma dada variável. Por exemplo, uma altura negativa num banco é certamente um valor inválido e deve ser tratado como tal.

Vamos carregar um data set conhecido chamado "Pima Indians Diabetes". (https://www.kaggle.com/uciml/pima-indians-diabetes-database (https://www.kaggle.com/uciml/pima-indians-diabetes-database))

Esse data set contém medições clínicas de vários biomarcadores e a variável "Outcome", que representa se a paciente tem ou não diabetes. 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)

É sabido que ele contém valores faltantes, mas nesse caso eles não estão claramente identificados!

Vamos tentar identificar os missing values através de uma análise sumarizada do banco usando o método **describe** do pandas.

In [2]:
```
import pandas as pd

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

# podemos ver o número de linhas e colunas no banco
print 'Banco com %d amostras e %d colunas' % dataset_original.shape

# o vetor com os nomes das colunas ficam no parâmetro chamado "colu
mns"
print "Colunas: ", ' '.join(dataset_original.columns)

# vamos iterar sobre cada coluna do nosso banco e imprimir
# algumas linhas
for col in dataset_original.columns:
    print "Primeiras 3 linhas da coluna %s:" % col
    # indexação de DataFrames é usando o médoto .loc é feita da mes
ma forma que matrizes numpy
    # [0:5, col] imprime as linhas 0 a 4 da coluna col
    # [:, col] imprime todas as linhas da coluna col
    # [:-5,col] imprime as 5 últimas linhas da coluna col
    # [0,:] imprime todas as colunas da linha zero
    print dataset_original.loc[0:3,col]

# o comando describe mostra um resumo estatistico do banco
print "\nTabela sumarizando as colunas do banco"
dataset_original.describe()
```

```
Banco com 768 amostras e 9 colunas
Colunas:  Pregnancies Glucose BloodPressure SkinThickness Insulin
BMI DiabetesPedigreeFunction Age Outcome
Primeiras 3 linhas da coluna Pregnancies:
0    6
1    1
2    8
3    1
Name: Pregnancies, dtype: int64
Primeiras 3 linhas da coluna Glucose:
0    148
1     85
2    183
3     89
Name: Glucose, dtype: int64
Primeiras 3 linhas da coluna BloodPressure:
0    72
1    66
2    64
3    66
Name: BloodPressure, dtype: int64
Primeiras 3 linhas da coluna SkinThickness:
0    35
1    29
2     0
```

```
3    23
Name: SkinThickness, dtype: int64
Primeiras 3 linhas da coluna Insulin:
0     0
1     0
2     0
3    94
Name: Insulin, dtype: int64
Primeiras 3 linhas da coluna BMI:
0    33.6
1    26.6
2    23.3
3    28.1
Name: BMI, dtype: float64
Primeiras 3 linhas da coluna DiabetesPedigreeFunction:
0    0.627
1    0.351
2    0.672
3    0.167
Name: DiabetesPedigreeFunction, dtype: float64
Primeiras 3 linhas da coluna Age:
0    50
1    31
2    32
3    21
Name: Age, dtype: int64
Primeiras 3 linhas da coluna Outcome:
0    1
1    0
2    1
3    0
Name: Outcome, dtype: int64

Tabela sumarizando as colunas do banco
```

Out[2]:

	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 o data set, podemos perceber que **algumas colunas claramente contêm valores inválidos!**

Glucose, BloodPressure, SkinThickness, Insulin e *BMI* não podem ser zero, então esses valores claramente devem ser considerados inválidos.

Normalmente usamos o valor "NaN" em Python para representar esses casos, então vamos substituir todas as ocorrências do valor zero nas colunas que mencionamos por NaN.

O método **replace** do pandas permite que façamos isso de forma rápida.

In [3]:
```
import numpy as np

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

In [4]:
```
dataset_original.isnull().any()
```

Out[4]:
```
Pregnancies                 False
Glucose                      True
BloodPressure                True
SkinThickness                True
Insulin                      True
BMI                          True
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

O método acima permite identificar se alguma das colunas possui valores nulos. Como podemos ver, agora nossos dados faltantes estão devidamente identificados!

Agora podemos fazer duas coisas:

1. remover as linhas com valores inválidos
2. imputar valores faltantes pela média de cada coluna.

Para imputar valores, primeiro calculamos a média pela função **mean**. Depois, substituímos os valores inválidos da coluna pela média usando a função **fillna**.

Vamos criar dois datasets distintos para comparar os resultados posteriormente.

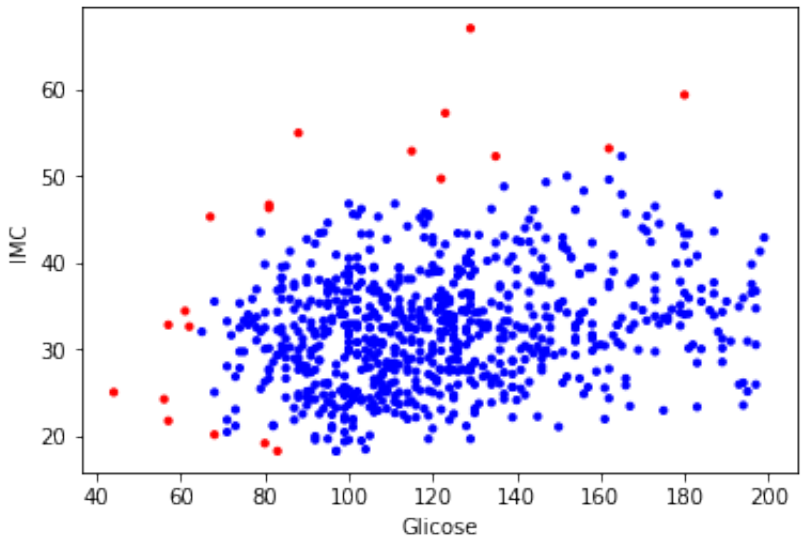
In [5]:
```
dataset_removido = dataset_original.dropna() # nesse dataset as linhas com NaN vão ser removidas


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

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

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 121.686763
substituindo valores faltando na coluna BloodPressure por 72.405184
substituindo valores faltando na coluna SkinThickness por 29.153420
substituindo valores faltando na coluna Insulin por 155.548223
substituindo valores faltando na coluna BMI por 32.457464

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

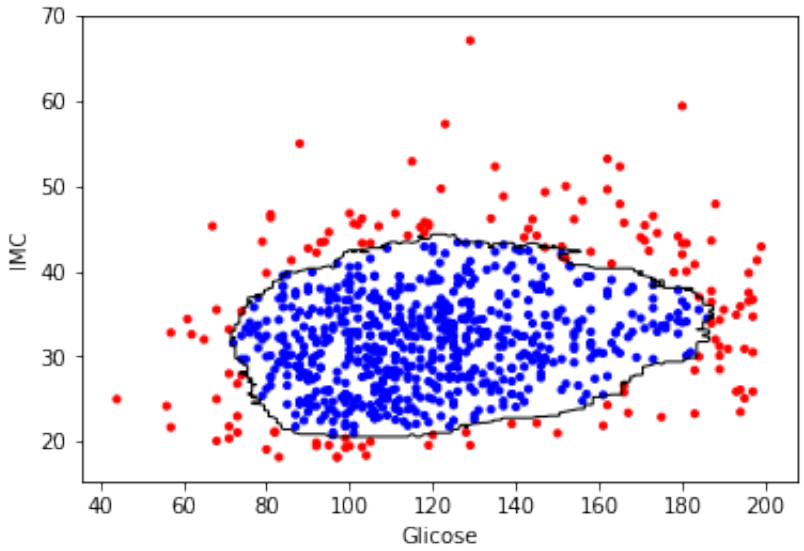
Perdemos **quase 50% da nossa amostra** devido à remoção de valores inválidos, portanto talvez a melhor opção seja mesmo imputar os valores inválidos. Para responder essa dúvida, a melhor forma é treinar modelos preditivos com os dois data sets e comparar os resultados. Isso fica a cargo do leitor.

Remoção de Outliers

In [15]:
```
from sklearn.neighbors import LocalOutlierFactor
import matplotlib.pyplot as plt
import numpy as np

lof = LocalOutlierFactor( n_neighbors=30, contamination = "auto")

X = dataset_imputado.loc[:,['Glucose','BMI']]
cluster_predito = lof.fit_predict(X)

colors = np.asarray(['red', 'blue'])

plt.scatter(X.Glucose, X.BMI, s=10, color=colors[(cluster_predito +
1) // 2])
plt.xlabel('Glicose')
plt.ylabel('IMC')
plt.show()
```



In [7]:
```
def plot_decision_countour(x1, x2, algo):
    xx, yy = np.meshgrid(sorted(x1),sorted(x2))
    Z = algo.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contour(xx, yy, Z, levels=[0], linewidths=1, colors='black'
)

from sklearn.ensemble import IsolationForest

colors = np.asarray(['red', 'blue'])


iso = IsolationForest(behaviour='new',  contamination = "auto")
cluster_predito = iso.fit(X).predict(X)


plot_decision_countour(X.Glucose, X.BMI, iso)

plt.scatter(X.Glucose, X.BMI, s=10, color=colors[(cluster_predito +
1) // 2])
plt.xlabel('Glicose')
plt.ylabel('IMC')
plt.show()
```



In [16]:
```
cluster_predito = lof.fit_predict(X)

dataset_imputado_sem_outliers = pd.DataFrame(dataset_imputado.loc[c
luster_predito == 1,:])
```

In [17]:
```
print 'Dataset com remoção:   %d amostras e %d colunas' % dataset_removido.shape
print 'Dataset com imputação: %d amostras e %d colunas' % dataset_imputado.shape
print 'Dataset com imputação + remoção de outliers: %d amostras e %d colunas' % dataset_imputado_sem_outliers.shape
```

```
Dataset com remoção:   392 amostras e 9 colunas
Dataset com imputação: 768 amostras e 9 colunas
Dataset com imputação + remoção de outliers: 748 amostras e 9 colunas
```

Balanceamento

Balancear as amostras de acordo com a distribuição de desfecho pode alterar completamente a análise estatística e a geração de modelos!

Existem diferentes formas de fazer esse balanceamento:

- **over sampling**: consiste em gerar novas amostras para que a classe com menos ocorrência tenha a mesma contagem que a classe com mais ocorrência
- **under sampling**: reduz as amostras da classe com mais ocorrência para que ela tenha a mesma contagem da classe com menos ocorrência

Vamos usar a biblioteca imblearn para aplicar under sampling no nosso conjunto de dados Pima.

In [91]:
```
count = np.unique(dataset_imputado_sem_outliers.Outcome, return_counts = True)
print 'Distribuição da classe sem under sampling %d %d' % (count[1][0], count[1][1])


from imblearn.under_sampling import RandomUnderSampler
X = dataset_imputado_sem_outliers.loc[:, dataset_imputado_sem_outliers.columns != 'Outcome']
y = dataset_imputado_sem_outliers.Outcome

rus = RandomUnderSampler()
Xr, yr = rus.fit_sample(X,y)
count_under = np.unique(yr, return_counts = True)
print 'Distribuição da classe sem under sampling %d %d' % (count_under[1][0], count_under[1][1])
```

```
Distribuição da classe sem under sampling 487 261
Distribuição da classe sem under sampling 261 261
```

Denoising

Consiste em remover ruído das amostras.

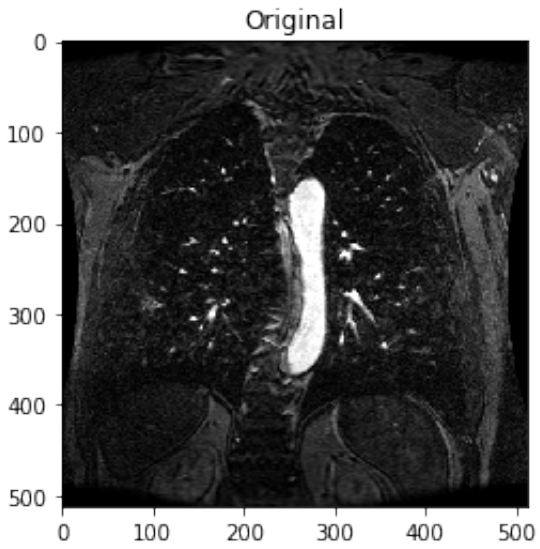
Dependendo da fonte do ruído, diferentes abordagens devem ser consideradas.

Vamos ver um caso clássico de ruído com imagens médicas. Um dos artefatos mais comuns são os ruídos salt & pepper (parecido com o ruído da TV analógica). Uma abordagem comum para atacar esse ruído é utilizando o filtro mediana.

Vamos usar um MRA (Magnetic Resonance Angiography) do tórax de um paciente como exemplo (exemplo retirado do banco PhysioNet [https://www.physionet.org/physiobank/database/images (https://www.physionet.org/physiobank/database/images)]

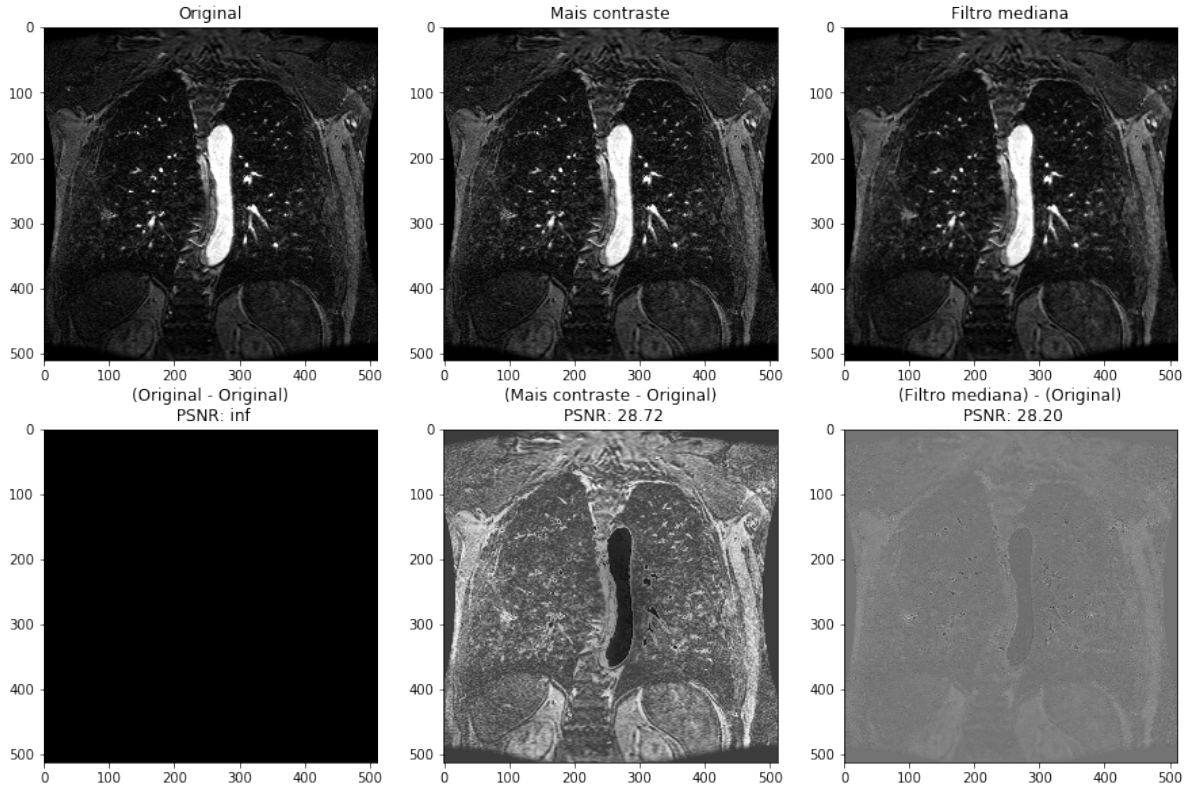
In [76]:
```
import matplotlib.pyplot as plt

im = plt.imread('./MRA_example.png').astype(float)

plt.figure()
plt.imshow(im, plt.cm.gray)
plt.title('Original')
```

Out[76]:
```
Text(0.5,1,'Original')
```



Agora vamos aplicar o filtro mediana (median_filter), mas antes vamos aumentar o contraste da nossa ressonância usando uma técnica chamada CLAHE (Contrast Limited Adaptive Histogram Equalization).

In [81]:
```
from scipy import ndimage
from skimage import exposure as exp
from skimage.measure import compare_psnr

fig = plt.figure(figsize= (15,10))
ax0 = fig.add_subplot(2,3,1)
ax0.imshow(im, plt.cm.gray)
ax0.set_title('Original')

im_contrast = exp.equalize_adapthist(im)
ax1 = fig.add_subplot(2,3,2)
ax1.imshow(im_contrast, plt.cm.gray)
ax1.set_title('Mais contraste')

im_median = ndimage.median_filter(im_contrast, size = [3,3])

ax2 = fig.add_subplot(2,3,3)
ax2.imshow(im_median, plt.cm.gray)
ax2.set_title('Filtro mediana')



ax0 = fig.add_subplot(2,3,4)
ax0.imshow(im-im, plt.cm.gray)
ax0.set_title('(Original - Original)\n PSNR: %.2f' % (compare_psnr(
im, im)) )

ax1 = fig.add_subplot(2,3,5)
ax1.imshow(im_contrast-im, plt.cm.gray)
ax1.set_title('(Mais contraste - Original)\n PSNR: %.2f' % (compare
_psnr(im, im_contrast)) )

ax2 = fig.add_subplot(2,3,6)
ax2.imshow(im_median-im, plt.cm.gray)
ax2.set_title('(Filtro mediana) - (Original)\n PSNR: %.2f' % (compa
re_psnr(im, im_median)) )



plt.show()
```


Original
Mais contraste
Filtro mediana
(Original - Original)
PSNR: inf
(Mais contraste - Original)
PSNR: 28.72
(Filtro mediana) - (Original)
PSNR: 28.20

In [ ]:

In [ ]: