Les Pima sont un groupe d'Amérindiens vivant en Arizona. Une prédisposition génétique a permis à ce groupe de survivre normalement à un régime pauvre en glucides pendant des années. Au cours des dernières années, en raison d'un passage soudain des cultures agricoles traditionnelles aux aliments transformés, ainsi que d'un déclin de l'activité physique, ils ont développé la prévalence la plus élevée de diabète de type 2 et pour cette raison ils ont fait l'objet de nombreuses études.

L'ensemble de données comprend des données provenant de 768 femmes présentant 8 caractéristiques, en particulier :

- Nombre de grossesses
- Concentration de glucose plasmatique a 2 heures dans un test de tolérance au glucose par voie orale
- Tension artérielle diastolique (mm Hg)
- Epaisseur du pli cutané du triceps (mm)
- 2 heures d'insuline sérique (mu U/ml)
- Indice de masse corporelle (poids en kg/(taille en m)^2)
- Le diabète pedigree fonction
- Âge (années) La dernière colonne de l'ensemble de données indique si la personne a reçu un diagnostic de diabète (1) ou non (0).

L'objectif est de déterminer quelles sont les caractéristiques (features) pour identifier les personnes qui ont un diabète de type 2

Récupérer le fichier pima-indians-diabetes.csv et le mettre dans un dataframe. Attention les premières lignes correspondent à la description des données. Il est possible de ne pas les lire en mettant skiprows=9 dans la fonction read_csv.

In [56]:

```
import pandas as pd
import numpy as np
names=[
    "NumTimesPrg", "PlGlcConc", "BloodP",
    "SkinThick", "TwoHourSerIns", "BMI",
    "DiPedFunc", "Age", "HasDiabetes"]
#il faut sauter les 9 premières lignes qui sont le descriptif des variables
df = pd.read_csv('pima-indians-diabetes.csv',names=names,skiprows=9)

```

Afficher le nombre de ligne et de colonnes du dataframe ainsi que les 5 premières lignes

In [8]:

```
print (df.shape)
display(df.head())
```

```
(768, 9)
```

	NumTimesPrg	PlGlcConc	BloodP	SkinThick	TwoHourSerIns	BMI	DiPedFunc	Age	Hasl
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

Afficher la matrice de corrélation. Rappel il faut utiliser la fonction corr().

In [10]:

```
corr = df.corr()
display(corr)
```

	NumTimesPrg	PlGlcConc	BloodP	SkinThick	TwoHourSerIns	BMI	DiF
NumTimesPrg	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0
PlGlcConc	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0
BloodP	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0
SkinThick	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0
TwoHourSerIns	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0
DiPedFunc	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0
HasDiabetes	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0

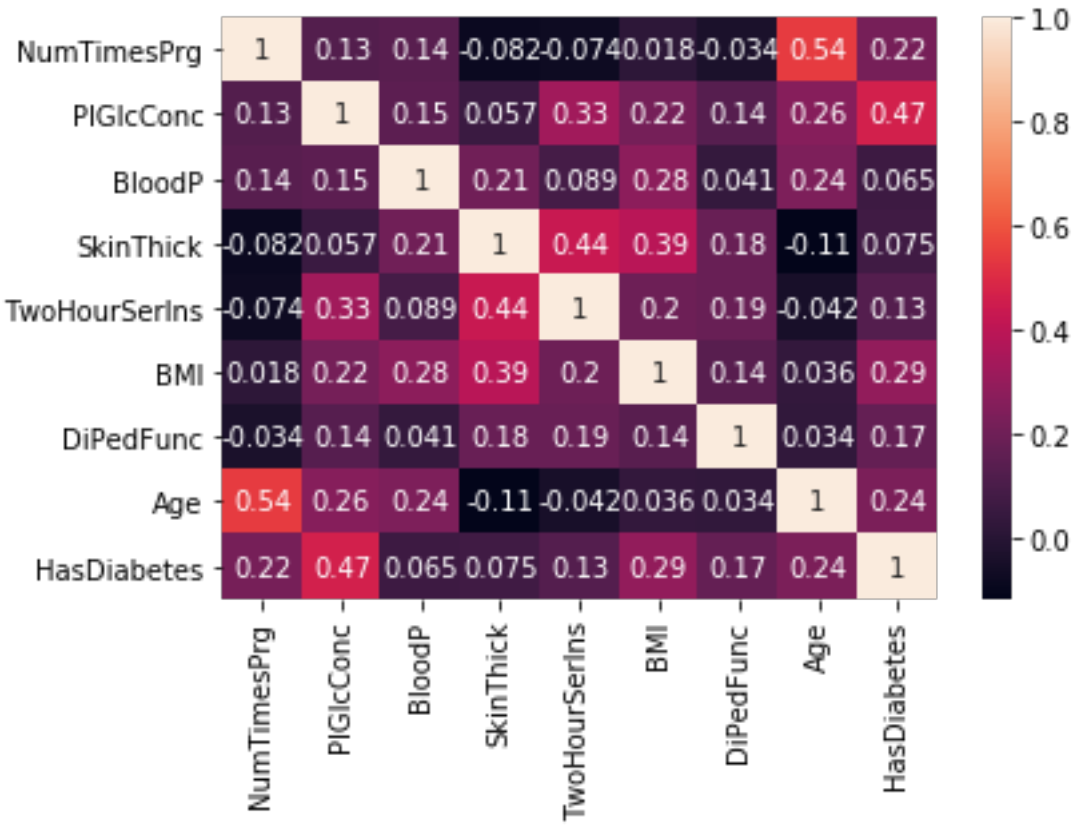
Afficher, à l'aide de seaborn, la matrice de correlation

In [12]:

```
%matplotlib inline
import seaborn as sns
sns.heatmap(corr, annot = True)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x11428a518>
```

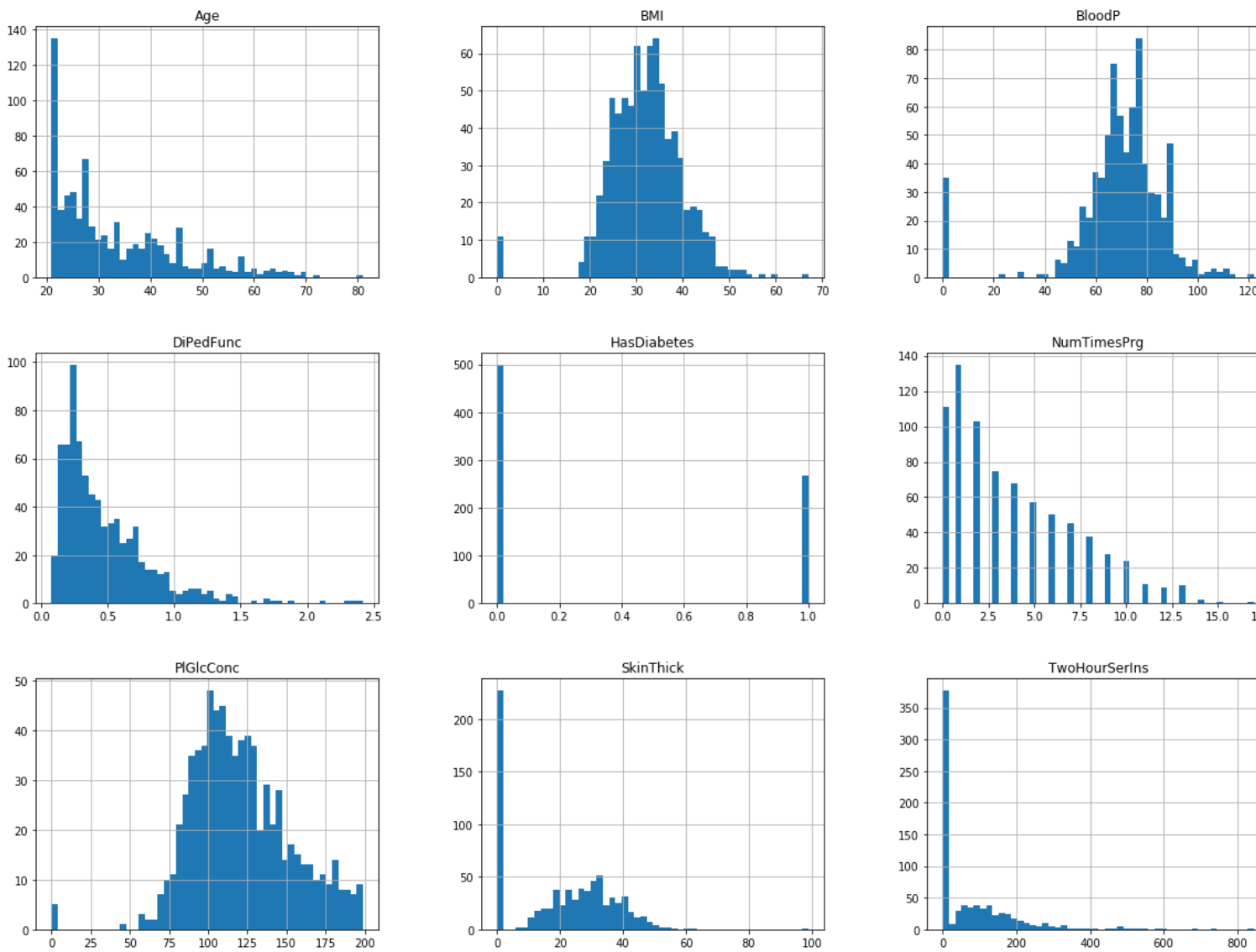


Il est important d'analyser les histogrammes de chaque variable pour mieux comprendre comment les données sont réparties.
A l'aide du code suivant, afficher les différents histogrammes.

```
import matplotlib.pyplot as plt
df.hist(bins=50, figsize=(20, 15))
plt.show()
```

In [15]:

```
import matplotlib.pyplot as plt
df.hist(bins=50, figsize=(20, 15))
plt.show()
```



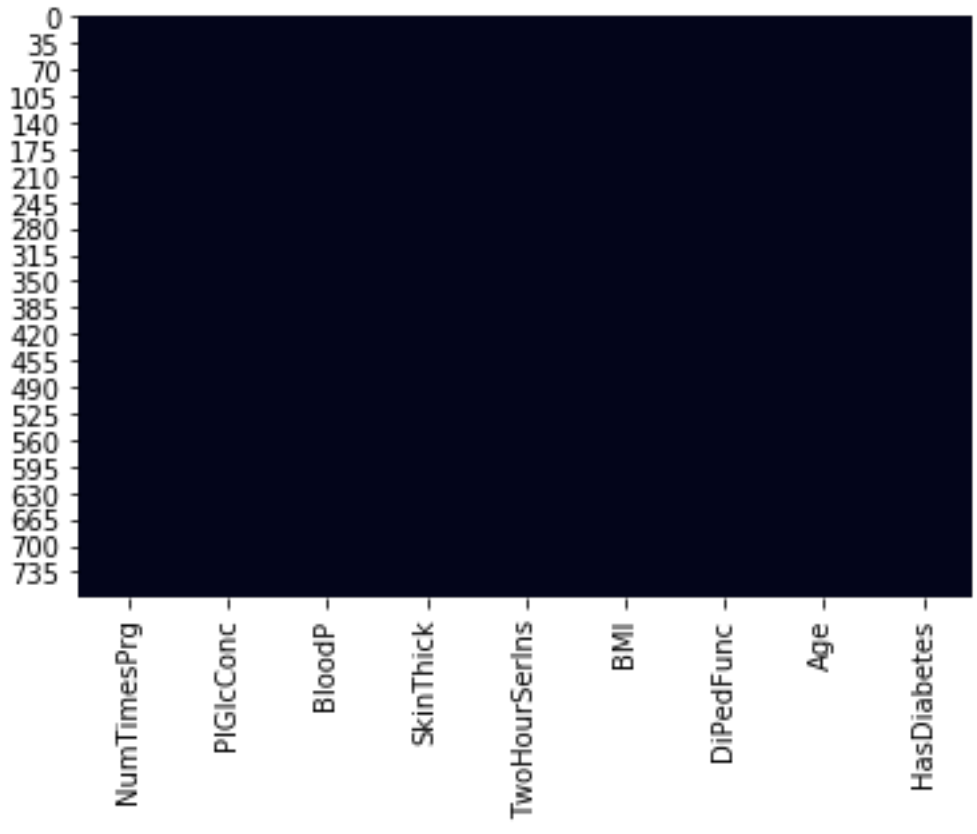
Existe-t'il des valeurs nulles ? Existe-til des valeurs manquantes ? Rappel vous pouvez le voir avec des histogrammes mais aussi avec une heatmap.

In [57]:

```
import seaborn as sns
sns.heatmap(df.isnull(), cbar=False)
```

Out[57]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x117ad12e8>
```



En fait on peut constater qu'il n'y a pas de valeurs manquantes avec le heatmap mais par contre il y a des valeurs nulles. Il faut toujours faire attention à la manière dont sont codées les valeurs manquantes. Ici nous voyons dans les histogrammes que pour BMI, BloodP, PlGlcConc, SkinThick, TwoHourSerIns il existe des valeurs manquantes. Le nombre de grossesses n'est pas considéré comme une valeur manquante bien sûr.

Transformer les valeurs nulles par la médiane de la série.

In [18]:

```
# valeur médiane
median_bmi = df['BMI'].median()
# remplacement par la médiane
df['BMI'] = df['BMI'].replace(
    to_replace=0, value=median_bmi)

# valeur médiane
median_bloodP = df['BloodP'].median()
# remplacement par la médiane
df['BloodP'] = df['BloodP'].replace(
    to_replace=0, value=median_bloodP)

# valeur médiane
median_PlGlcConc = df['PlGlcConc'].median()
# remplacement par la médiane
df['PlGlcConc'] = df['PlGlcConc'].replace(
    to_replace=0, value=median_PlGlcConc)

# valeur médiane
median_SkinThick = df['SkinThick'].median()
# remplacement par la médiane
df['SkinThick'] = df['SkinThick'].replace(
    to_replace=0, value=median_SkinThick)

# valeur médiane
median_TwoHourSerIns = df['TwoHourSerIns'].median()
# remplacement par la médiane
df['TwoHourSerIns'] = df['TwoHourSerIns'].replace(
    to_replace=0, value=median_TwoHourSerIns)

```

Les données sont, à présent, transformées et nous allons pouvoir créer un jeu de données de test et d'apprentissage. Faire une copie du dataframe en df2. Sur df appliquer un scaling pour normaliser les valeurs par rapport à la moyenne et l'écart type (utilisation de StandardScaler (). Nous conservons la copie df2 sans transformation.
L'objectif à présent est d'appliquer différents classifieurs pour voir celui qui est le plus performant. Pour le ou les meilleurs rechercher les hyperparamètres et créer un pipeline à sauvegarder. Il faut ensuite pouvoir traiter de nouvelles données pour prédire si il y a diabète ou pas.

Tester les résultats sur df et sur df2.

In [69]:

```
# pour essayer sans scaling
df2=df.copy()

#traitement de df
# séparation données à prédire
array = df.values
X = array[:,0:8]
y = array[:,8]


#utilisation de StandardScaler


standardscaler = StandardScaler()
X = standardscaler.fit_transform(X)
print ("Pour vérifier que les données ont bien été transformées")
print (pd.DataFrame(X).head())


#création d'un jeu de test et d'apprentissage (70,30)
from sklearn.model_selection import train_test_split

validation_size=0.3 #30% du jeu de données pour le test

testsize= 1-validation_size
seed=30
X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)
```

```
Pour vérifier que les données ont bien été transformées
          0         1         2         3         4         5
6  \
0  0.639947  0.848324  0.149641  0.907270 -0.692891  0.204013  0.468
492
1 -0.844885 -1.123396 -0.160546  0.530902 -0.692891 -0.684422 -0.365
061
2  1.233880  1.943724 -0.263941 -1.288212 -0.692891 -1.103255  0.604
397
3 -0.844885 -0.998208 -0.160546  0.154533  0.123302 -0.494043 -0.920
763
4 -1.141852  0.504055 -1.504687  0.907270  0.765836  1.409746  5.484
909

          7
0  1.425995
1 -0.190672
2 -0.105584
3 -1.041549
4 -0.020496
```

In [70]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#traitement de df
# séparation données à prédire
array = df.values
X = array[:,0:8]
y = array[:,8]


#utilisation de StandardScaler
standardscaler = StandardScaler()
X = standardscaler.fit_transform(X)
print ("Pour vérifier que les données ont bien été transformées")
print (pd.DataFrame(X).head())


#création d'un jeu de test et d'apprentissage (70,30)


validation_size=0.2 #30% du jeu de données pour le test

testsize= 1-validation_size
seed=20
X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)
```

```
Pour vérifier que les données ont bien été transformées
          0         1         2         3         4         5
6  \
0  0.639947  0.848324  0.149641  0.907270 -0.692891  0.204013  0.468
492
1 -0.844885 -1.123396 -0.160546  0.530902 -0.692891 -0.684422 -0.365
061
2  1.233880  1.943724 -0.263941 -1.288212 -0.692891 -1.103255  0.604
397
3 -0.844885 -0.998208 -0.160546  0.154533  0.123302 -0.494043 -0.920
763
4 -1.141852  0.504055 -1.504687  0.907270  0.765836  1.409746  5.484
909

          7
0  1.425995
1 -0.190672
2 -0.105584
3 -1.041549
4 -0.020496
```

In [71]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn import model_selection
```

In [72]:

```
models = []
models.append(('LR', LogisticRegression()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVC', SVC(gamma='auto')))
models.append(('LSVC', LinearSVC(max_iter=3000)))
models.append(('RFC', RandomForestClassifier()))
models.append(('DTR', DecisionTreeRegressor()))
```

In [73]:

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import time
seed = 7
results = []
names = []
scoring='accuracy'
for name,model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    start_time = time.time()
    cv_results = cross_val_score(model, X, y, cv=kfold, scoring=scoring)
    print ("Time pour",name," ",time.time() - start_time)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

```

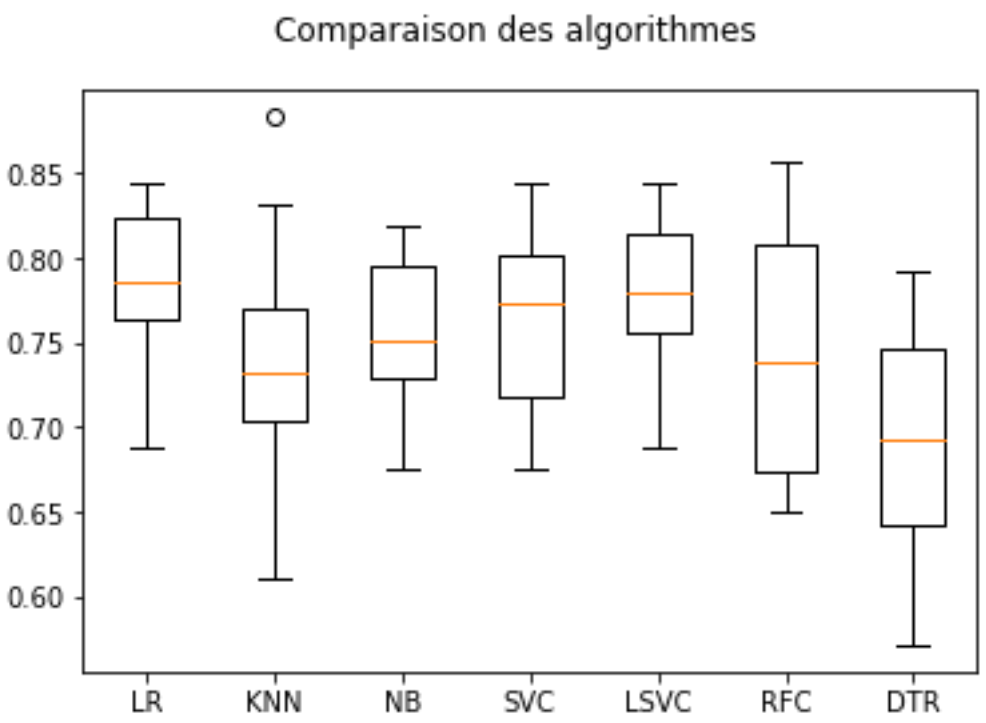
```
Time pour LR   0.048712968826293945
LR: 0.779956 (0.050088)
Time pour KNN   0.030850887298583984
KNN: 0.742139 (0.071500)
Time pour NB   0.015557050704956055
NB: 0.755178 (0.042766)
Time pour SVC   0.12708377838134766
SVC: 0.764286 (0.056962)
Time pour LSVC   0.44120788574218 75
LSVC: 0.776059 (0.049143)
Time pour RFC   0.1575641632080078
RFC: 0.743438 (0.072740)
Time pour DTR   0.028022050857543945
DTR: 0.692618 (0.066232)
```

In [74]:

```
fig = plt.figure()
fig.suptitle('Comparaison des algorithmes')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
```

Out[74]:

```
[Text(0, 0, 'LR'),
 Text(0, 0, 'KNN'),
 Text(0, 0, 'NB'),
 Text(0, 0, 'SVC'),
 Text(0, 0, 'LSVC'),
 Text(0, 0, 'RFC'),
 Text(0, 0, 'DTR')]
```



Les meilleurs algorithmes sont LR et SVC. Nous pouvons rechercher les hyperparamètres pour ces deux algorithmes.

In [75]:

```
from sklearn.model_selection import GridSearchCV

grid_param = {
    'C' : [0.001,0.01,0.1,1,10,100]
}


gd_sr = GridSearchCV(estimator=LogisticRegression(),
                     param_grid=grid_param,
                     scoring='accuracy',
                     cv=5,
                     n_jobs=-1,
                    iid=True,
                    return_train_score=True)

gd_sr.fit(X_train, y_train)

print ('meilleur score ',gd_sr.best_score_,'\n')
print ('meilleurs paramètres', gd_sr.best_params_,'\n')
print ('meilleur estimateur',gd_sr.best_estimator_,'\n')
```

```
meilleur score  0.7516339869281046

meilleurs paramètres {'C': 1}

meilleur estimateur LogisticRegression(C=1, class_weight=None, dual=
False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn
',
          tol=0.0001, verbose=0, warm_start=False)
```

In [52]:

```
grid_param = {
    'C': [0.001, 0.01, 0.1, 1, 10],
    'gamma' : [0.001, 0.01, 0.1, 1],
    'kernel': ['linear','rbf']}

gd_sr = GridSearchCV(estimator=SVC(),
                     param_grid=grid_param,
                     scoring='accuracy',
                     cv=5,
                     n_jobs=1,
                     iid=True,
                    return_train_score=True)

gd_sr.fit(X_train, y_train)

print ('meilleur score ',gd_sr.best_score_,'\n')
print ('meilleurs paramètres', gd_sr.best_params_,'\n')
print ('meilleur estimateur',gd_sr.best_estimator_,'\n')
```

```
meilleur score  0.7608695652173914

meilleurs paramètres {'C': 1, 'gamma': 0.001, 'kernel': 'linear'}

meilleur estimateur SVC(C=1, cache_size=200, class_weight=None, coef
0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.001, kernel='line
ar',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

SVC s'avère obtenir de meilleurs résultats. Nous l'utilisons pour faire de la prédiction sur les données test.

In [76]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# Creation d'une instance de l'algorithme en utilisant les meilleurs parame
svc = gd_sr.best_estimator_

validation_size=0.2 #20% du jeu de données pour le test

testsize= 1-validation_size
seed=20
X_train,X_test,y_train,y_test=train_test_split(X, y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)



svc.fit(X_train, y_train)
result = svc.predict(X_test)
print('\n accuracy: ', accuracy_score(result, y_test),'\n')

conf = confusion_matrix(y_test, result)
print ('\n matrice de confusion \n',conf)
print ('\n',classification_report(y_test, result))
```

```
accuracy:  0.7430894308943089
```

```
 matrice de confusion
 [[363  26]
 [132  94]]

              precision    recall  f1-score   support

         0.0       0.73      0.93      0.82       389
         1.0       0.78      0.42      0.54       226

   micro avg       0.74      0.74      0.74       615
   macro avg       0.76      0.67      0.68       615
weighted avg       0.75      0.74      0.72       615
```

Création d'un pipeline complet pour sauvegarder le modèle et le tester sur de nouvelles données.

In [77]:

```
from sklearn.pipeline import Pipeline

names=[
    "NumTimesPrg", "PlGlcConc", "BloodP",
    "SkinThick", "TwoHourSerIns", "BMI",
    "DiPedFunc", "Age", "HasDiabetes"]
#il faut sauter les 9 premières lignes qui sont le descriptif des variables
df = pd.read_csv('pima-indians-diabetes.csv',names=names,skiprows=9)

array = df.values
X = array[:,0:8]
y = array[:,8]


print ('Création du pipeline \n')
pipeline = Pipeline([('scl', StandardScaler()),
                     ('clf', svc)])

validation_size=0.3 #30% du jeu de données pour le test

testsize= 1-validation_size
seed=30
X_train,X_test,y_train,y_test=train_test_split(X, y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)


pipeline.fit(X_train, y_train)
result = pipeline.predict(X_test)

print('\n accuracy:',accuracy_score(result, y_test),'\n')

import pickle
filename = 'modelsvcpima.pkl'
pickle.dump(pipeline, open(filename, 'wb'))
```

```
Création du pipeline


 accuracy: 0.7881040892193308
```

In [78]:

```
print ("Chargement du modèle \n")
filename = 'modelsvcpima.pkl'
clf_loaded = pickle.load(open(filename, 'rb'))

#Considérons deux nouvelles données qui ne sont pas standardisées
new_df = pd.DataFrame([[6, 168, 72, 35, 0, 43.6, 0.627, 65],
                       [2,85,67,30,2,27.6,0.351,22]], dtype='float')
print ("La première ligne correspond à une personne qui a tendance à avoir
       "la prédiction devrait être de 1. Pour la seconde elle devrait être
      "il faut remarquer que les données ne sont pas standardisées à l'ent
      "de la prédiction une étape de standardisation à lieu grâce au pipel
prediction = clf_loaded.predict(new_df)
print(prediction)

```

```
Chargement du modèle

La première ligne correspond à une personne qui a tendance à avoir d
u diabète de type 2 la prédiction devrait être de 1. Pour la seconde
elle devrait être de 0 il faut remarquer que les données ne sont pas
standardisées à l'entrée et que lors de la prédiction une étape de s
tandardisation à lieu grâce au pipeline
[1. 0.]
```

Essai de df2 sans standardisation.

Nous avons vu que SVC avait de meilleurs résultats aussi nous l'utilisons ici pour voir l'intérêt de la standardisation.

In [67]:

```
array = df2.values
X = array[:,0:8]
y = array[:,8]

svc = gd_sr.best_estimator_

validation_size=0.2 #20% du jeu de données pour le test

testsize= 1-validation_size
seed=20
X_train,X_test,y_train,y_test=train_test_split(X, y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)



svc.fit(X_train, y_train)
result = svc.predict(X_test)
print('\n accuracy: ', accuracy_score(result, y_test),'\n')

conf = confusion_matrix(y_test, result)
print ('\n matrice de confusion \n',conf)
print ('\n',classification_report(y_test, result))
```

```
accuracy:  0.7382113821138211
```

```
 matrice de confusion
 [[366  23]
 [138  88]]

              precision    recall  f1-score   support

         0.0       0.73      0.94      0.82       389
         1.0       0.79      0.39      0.52       226

   micro avg       0.74      0.74      0.74       615
   macro avg       0.76      0.67      0.67       615
weighted avg       0.75      0.74      0.71       615
```

Comme attendu, étant donné qu'il y a des données à des échelles très différentes l'accuracy diminue lorsque les données ne sont pas transformées.