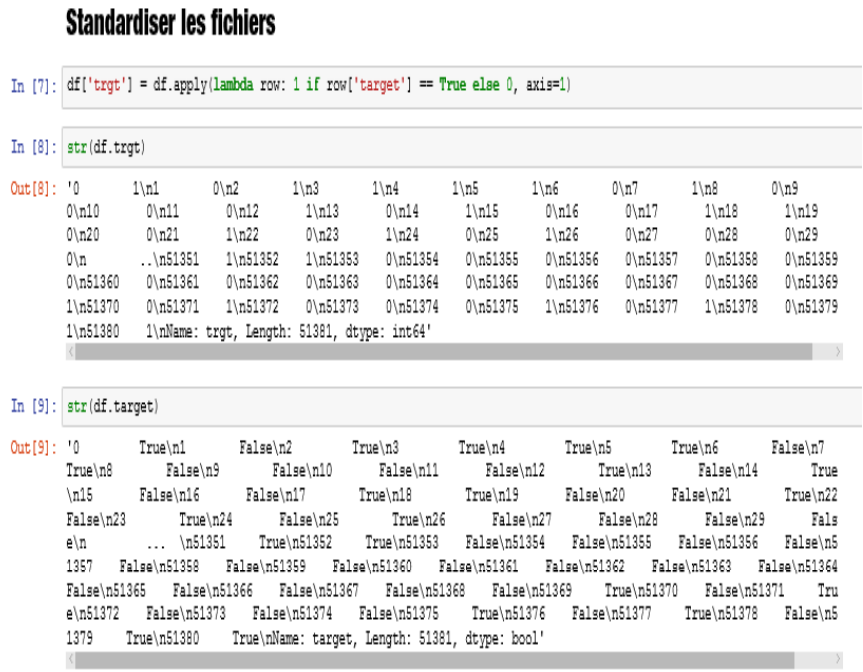
Importation des fonctions

- Machine utilisée: Lenovo P92, CPU Intel Core i5-3470 3.2Ghz, RAM 8 Go - Windows 10 Pro – Puissance raisonnable mais insuffisante pour les calculs lourds.
- Importation des fonctions et du fichier de données
- On vérifie la bonne importation

Standardisation des fichiers

- La variable target étant booléenne, on la transforme en numérique.
  - Création de la variable numérique trgt équivalente en 0 et 1
  - Vérification que les deux variables correspondent bien



Standardiser les fichiers

```
In [7]: df['trgt'] = df.apply(lambda row: 1 if row['target'] == True else 0, axis=1)

In [8]: str(df.trgt)

Out[8]: '0        1\n1        0\n2        1\n3        1\n4        1\n5        1\n6        0\n7        1\n8        0\n9
        0\n10       0\n11       0\n12       1\n13       0\n14       1\n15       0\n16       0\n17       1\n18       1\n19
        0\n20       0\n21       1\n22       0\n23       1\n24       0\n25       1\n26       0\n27       0\n28       0\n29
        0\n        ..\n51351    1\n51352    1\n51353    0\n51354    0\n51355    0\n51356    0\n51357    0\n51358    0\n51359
        0\n51360    0\n51361    0\n51362    0\n51363    0\n51364    0\n51365    0\n51366    0\n51367    0\n51368    0\n51369
        1\n51370    0\n51371    1\n51372    0\n51373    0\n51374    0\n51375    1\n51376    0\n51377    1\n51378    0\n51379
        1\n51380    1\nName: trgt, Length: 51381, dtype: int64'

In [9]: str(df.target)

Out[9]: '0         True\n1        False\n2         True\n3         True\n4        True\n5         True\n6        False\n7
        True\n8        False\n9        False\n10       False\n11       False\n12       True\n13       False\n14        True
        \n15       False\n16       False\n17        True\n18        True\n19       False\n20       False\n21        True\n22
        False\n23        True\n24       False\n25        True\n26       False\n27       False\n28       False\n29       Fals
        e\n         ... \n51351     True\n51352     True\n51353    False\n51354    False\n51355    False\n51356    False\n5
        1357    False\n51358    False\n51359    False\n51360    False\n51361    False\n51362    False\n51363    False\n51364
        False\n51365    False\n51366    False\n51367    False\n51368    False\n51369     True\n51370    False\n51371     Tru
        e\n51372    False\n51373    False\n51374    False\n51375     True\n51376    False\n51377     True\n51378    False\n5
        1379     True\n51380     True\nName: target, Length: 51381, dtype: bool'
```

Standardisation des fichiers

- Extraction de la variable y = tgrt à expliquer, du fichier général renommé X
- Suppression de la variable Product_Info_2 qui comporte des données non numérique
- Vérification de la forme définitive de X et y

```
In [10]: X=df

In [12]: y=X.pop("trgt")

In [13]: X= X.drop('Product_Info_2',axis=1)

In [14]: X=X.drop('target',axis=1)

In [15]: X.head()
```

Out[15]:

	Unnamed: 0	Id	Product_Info_1	Product_Info_3	Product_Info_4	Product_Info_5	Product_Info_6	Product_Info_7	Ins_Age	Ht	...	Medical_Keyword_39	Medical_Keyword_
0	1	2	1	10	0.076923	2	1	1	0.641791	0.581818	...	0	
1	2	5	1	26	0.076923	2	3	1	0.059701	0.600000	...	0	
2	3	6	1	26	0.076923	2	3	1	0.029851	0.745455	...	0	
3	4	7	1	10	0.487179	2	3	1	0.164179	0.672727	...	0	
4	5	8	1	26	0.230769	2	3	1	0.417910	0.654545	...	0	

5 rows × 127 columns

```
In [16]: y.head()
```

Out[16]:

```
0    1
1    0
2    1
3    1
4    1
Name: trgt, dtype: int64
```

Random Forest

- On applique Random Forest et on mesure le ROC = 89,6%
- On étudie le ROC en fonction du nombre d'arbres. On limite à 100 arbres (hardware)
- On passe de 89.6% à près de 90,97%

Random Forest

In [17]:
```
model= RandomForestRegressor(n_estimators=30,oob_score=True,random_state=42)
model.fit(X,y)
```

Out[17]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=30, n_jobs=1,
           oob_score=True, random_state=42, verbose=0, warm_start=False)
```

In [19]:
```
y_oob=model.oob_prediction_
s=roc_auc_score(y,y_oob)
print ("c-stat :",s)
```

```
c-stat : 0.896506492098
```

In [21]:
```
results = []
n_estimator_options = [30,50,100]
for trees in n_estimator_options:
    model=RandomForestRegressor(trees,oob_score=True,random_state=42)
    model.fit(X,y)
    print(trees,"trees")
    roc=roc_auc_score(y,model.oob_prediction_)
    print("C-stat :",roc)
    results.append(roc)
    print("")
```

```
30 trees
C-stat : 0.896506492098

50 trees
C-stat : 0.903979076481

100 trees
C-stat : 0.90976384577
```

Random Forest

- On fait varier la taille des feuilles de 1 à 7, mais avec un nombre d'arbres limité à 30 (hardware)
- On améliore le ROC de 89.6% à 90,7%

```
In [22]: #min samples leaf
         rocs = {}
         min_samples_leaf_options = [1,2,3,4,5,6,7]
         for min_samples in min_samples_leaf_options:
             model=RandomForestRegressor(n_estimators=30,oob_score=True,
                                         random_state=42,
                                         max_features="auto",
                                         min_samples_leaf=min_samples)
             model.fit(X,y)
             print(min_samples,"min_samples")
             roc=roc_auc_score(y,model.oob_prediction_)
             rocs[min_samples] =[roc]
             print("C-stat: ",roc)

         1 min_samples
         C-stat:  0.896506492098
         2 min_samples
         C-stat:  0.90007083883
         3 min_samples
         C-stat:  0.902170571064
         4 min_samples
         C-stat:  0.90315544054
         5 min_samples
         C-stat:  0.905253127582
         6 min_samples
         C-stat:  0.906000140976
         7 min_samples
         C-stat:  0.907108910231
```

Random Forest

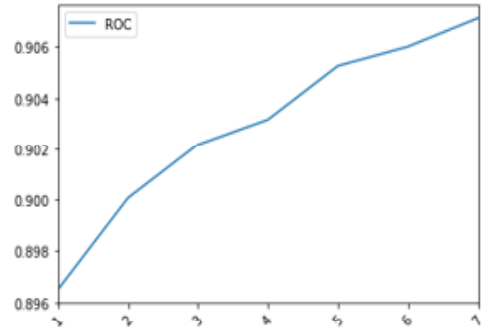
- On visualise la courbe de progression du ROC en fonction du nombre de feuilles
- ROC serait donc meilleur avec à la fois un nombre d'arbres et de taille de feuilles plus important.

Out[25]:

	ROC
1	0.896506
2	0.900071
3	0.902171
4	0.903155
5	0.905253
6	0.906000
7	0.907109

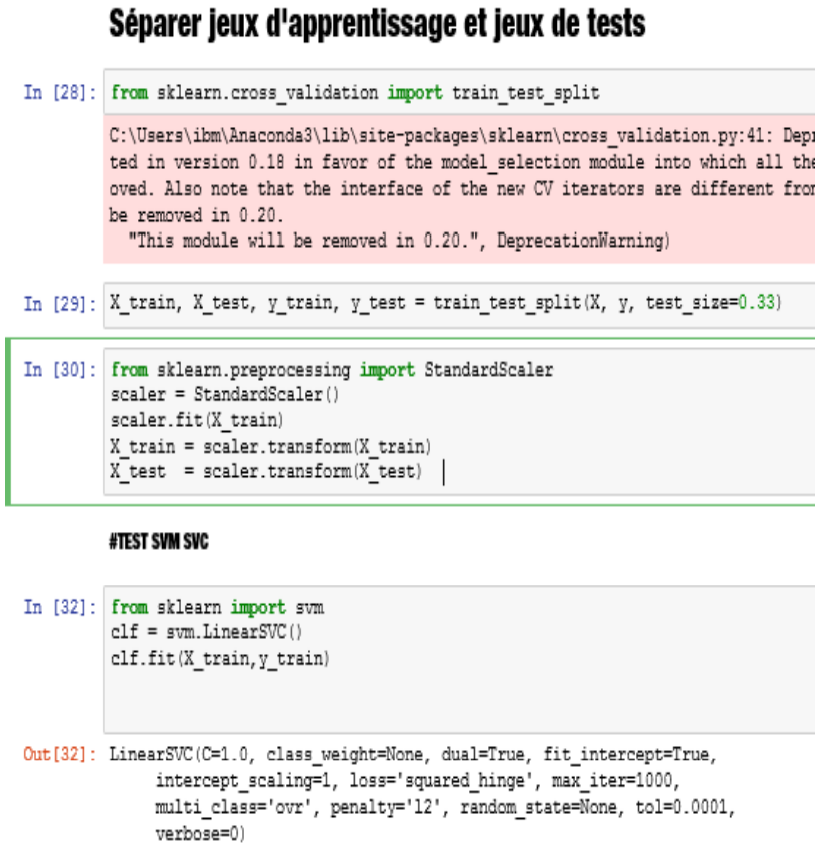
In [26]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```

In [27]:
```
df = df.sort_values(col_name)
ax = df.plot(rot=45, x_compat=True)
```



SVM

- Préparation des données d'apprentissage et de test à partir du fichier X
- Application de l'algo SVM



Séparer jeux d'apprentissage et jeux de tests

In [28]:
```
from sklearn.cross_validation import train_test_split
```

```
C:\Users\ibm\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: Depr
ted in version 0.18 in favor of the model_selection module into which all the
oved. Also note that the interface of the new CV iterators are different fro
be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [29]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

In [30]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test  = scaler.transform(X_test)
```

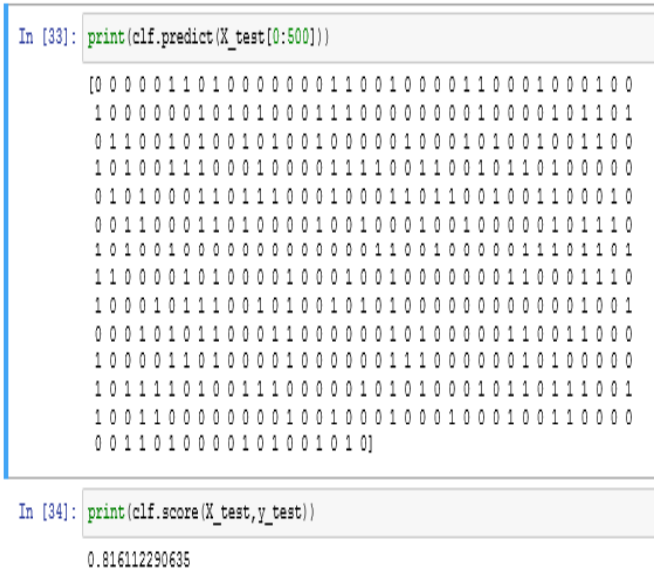
#TEST SVM SVC

In [32]:
```
from sklearn import svm
clf = svm.LinearSVC()
clf.fit(X_train,y_train)
```

Out[32]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
```

SVM

- On applique la prédiction qui donne un score de 81,6%
- On renon ce à faire une analyse des paramètres sous GridSearch car le paramètre n_jobs provoque un bug sous Windows répertorié comme non encore résolu



Série d'algorithmes

- Nous importons tous les algorithmes de Scikit learn

Grande série d'algorithmes

In [35]:
```
def get_sklearn_algorithms(verbose = False):
    """
    Explore all submodule of sklearn and fetch functions having a 'fit' attribute.

    Be careful : some functions are not models (ex : crossvalidators)
    Parameters :
        debug = print or not stuff on console
    Return :
        dict : { module : [ fit_functions] }
    """
    from collections import defaultdict
    import importlib
    import sklearn
    algos = defaultdict(list)
    if verbose : print (dir(sklearn))
    for nom_module in dir(sklearn):
        if verbose : print (nom_module)
        try:
            to_import = "sklearn.%s"%nom_module
            module    = importlib.import_module(to_import)
            for nom_fonction in dir(module):
                fonction = getattr(module, nom_fonction)
                if hasattr(fonction, "fit"):
                    if verbose : print (" nom algorithme  = ", nom_fonction)
                    algos[nom_module].append(fonction)
        except Exception as e:
            if verbose : print (e)
        if verbose: print ("="*30)
    return algos
```

In [36]:
```
algos = get_sklearn_algorithms()
for key in algos.keys():
    print ("\n===>",key)
    algos_ = []
    for algo in algos[key]:
        classe_algo = str(algo)
        nom_algo    = classe_algo[str(classe_algo).rfind(".")+1:str(classe_algo).rfind("'")]
        algos_.append(nom_algo)
    print (",".join(algos_))
```

Série d'algorithmes

- On voit qu'il y a un grand nombre d'algos

```
        classe_algo = str(algo)
        nom_algo    = classe_algo[str(classe_algo).rfind("."}+1:str(classe_algo).rfind("'")]
        algos_.append(nom_algo)
    print (",".join(algos_))
```

```
===> ensemble
AdaBoostClassifier,AdaBoostRegressor,BaggingClassifier,BaggingRegressor,ExtraTreesClassifier,ExtraTreesRegressor,Gra
dientBoostingClassifier,GradientBoostingRegressor,IsolationForest,RandomForestClassifier,RandomForestRegressor,Rando
mTreesEmbedding,VotingClassifier

===> feature_selection
GenericUnivariateSelect,RFE,RFECV,SelectFdr,SelectFpr,SelectFromModel,SelectFwe,SelectKBest,SelectPercentile,Varianc
eThreshold

===> gaussian_process
GaussianProcess,GaussianProcessClassifier,GaussianProcessRegressor

===> linear_model
ARDRegression,BayesianRidge,ElasticNet,ElasticNetCV,HuberRegressor,Lars,LarsCV,Lasso,LassoCV,LassoLars,LassoLarsCV,L
assoLarsIC,LinearRegression,LogisticRegression,LogisticRegressionCV,MultiTaskElasticNet,MultiTaskElasticNetCV,MultiT
askLasso,MultiTaskLassoCV,OrthogonalMatchingPursuit,OrthogonalMatchingPursuitCV,PassiveAggressiveClassifier,PassiveA
ggressiveRegressor,Perceptron,RANSACRegressor,RandomizedLasso,RandomizedLogisticRegression,Ridge,RidgeCV,RidgeClassi
fier,RidgeClassifierCV,SGDClassifier,SGDRegressor,TheilSenRegressor

===> model_selection
GridSearchCV,RandomizedSearchCV

===> multiclass
LabelBinarizer,OneVsOneClassifier,OneVsRestClassifier,OutputCodeClassifier,_ConstantPredictor

===> neighbors
KNeighborsClassifier,KNeighborsRegressor,KernelDensity,LSHForest,LocalOutlierFactor,NearestCentroid,NearestNeighbor
s,RadiusNeighborsClassifier,RadiusNeighborsRegressor

===> preprocessing
Binarizer,FunctionTransformer,Imputer,KernelCenterer,LabelBinarizer,LabelEncoder,MaxAbsScaler,MinMaxScaler,MultiLabe
lBinarizer,Normalizer,OneHotEncoder,PolynomialFeatures,QuantileTransformer,RobustScaler,StandardScaler

===> random_projection
BaseRandomProjection,GaussianRandomProjection,SparseRandomProjection

===> svm
LinearSVC,LinearSVR,NuSVC,NuSVR,OneClassSVM,SVC,SVR,pyd
```

Série d'algorithmes

- On limite le test au 5 premiers (hardware)
- Les résultats de prévision s'échelonnent entre 33% et 83%

```
        modeles_a_tester.extend(algos[classe_de_modeles])


        for pointeur_vers_algo in modeles_a_tester:
            try:
                algorithme = pointeur_vers_algo()
                doc        = algorithme.__doc__
                name       = doc[:min(doc.find(":"), 25)].strip()
                print (name)
                algorithme.fit(X_train, y_train)
                performance = algorithme.score(X_test, y_test)
                print (performance)
                if performance >  best_perf:
                    best_algorithm = algorithme
                    best_perf = performance

                if 0<performance and performance<1:
                    performances[name] = [performance]

            except Exception as e:
                if "label" in str(e): print ("Algo de classification")
                else                : print (str(e)[:50])
            print ("="*30)
compteur=compteur-1
```

```
An AdaBoost classifier.
0.837697570182
==============================
An AdaBoost regressor.
0.333397616622
==============================
A Bagging classifier.
0.818235432885
==============================
A Bagging regressor.
0.448659766454
==============================
An extra-trees classifier
0.808150507195
==============================
```

Série d’algorithmes - Conclusion

- On visualise les résultats de prévision suivant les différents algos

- Conclusion: Random Forest a le plus de potentiel sur ce jeu de données.

In [97]:
```
df
```

Out[97]:

	ROC
An AdaBoost regressor.	0.333398
A Bagging regressor.	0.448660
An extra-trees classifier	0.808151
A Bagging classifier.	0.818235
An AdaBoost classifier.	0.837698

In [63]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```

In [98]:
```
df = df.sort_values(col_name)
ax = df.plot(rot=45, x_compat=True)
```