chapter-3-2

May 2, 2022

1 NN Heston Model - Market Generations

```
In [12]: import sys
         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import time
         import math

         from smt.sampling_methods import LHS
         from sklearn.model_selection import train_test_split
```

```
In [13]: import Cte

         Seed    = Cte.mktSeed
         TAG     = Cte.mktTAG
```

```
In [14]: SMALL_SIZE  = 8
         MEDIUM_SIZE = 10
         BIG_SIZE    = 12
         BIGGER_SIZE = 14

         plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
         plt.rc('axes', titlesize=BIG_SIZE)       # fontsize of the axes title
         plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
         plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
         plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
         plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
         plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
```

1.1 Python Auxiliary Functions

1.1.1 Latin Hypercube Sampling Function

```
In [15]: def lhs_sampling(rand, NUM, bounds=None):

             mInt = (1 << 15)
             MInt = (1 << 16)
```

```
    kw = bounds.keys()

    # builds the array of bounds
    limits = np.empty( shape=(0,2) )
    for k in kw: limits = np.concatenate((limits, [bounds[k]]), axis=0)

    sampling = LHS(xlimits=limits, random_state=rand.randint(mInt,MInt))
    x    = sampling(NUM)

    y = np.where( 2*x[:,0]*x[:,1] < x[:,2]*x[:,2], 1, 0)
    p = (100.*np.sum(y))/NUM
    print("@ %-34s: %s = %6d out of %6d ( %.7f %s)" %("Info", "Feller violations", np.s

    return kw, x
```

1.1.2 Pricing Functions

```
In [16]: from Lib.Heston import Heston
         from Lib.FT_opt import ft_opt

         def HestonPut(St, Strike, T, kappa, theta, sigma, v0, rho, r, Xc = 30):

             kT    = (Strike/St)*math.exp(-r*T)

             hestn = Heston(lmbda=kappa, eta=sigma, nubar=theta, nu_o=v0, rho=rho)
             res   = ft_opt(hestn, kT, T, Xc)

             return res['put'];
```

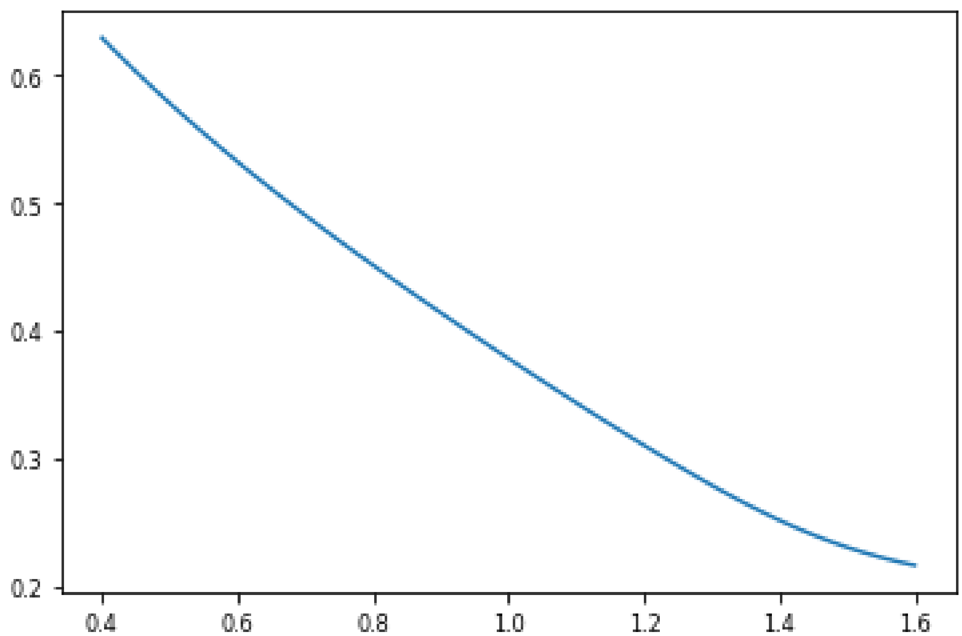
```
In [17]: from Lib.euro_opt import impVolFromFwPut

         def build_smile(strikes=None, Fw=1.0, T= 1.0, Kappa=1., Theta=1., sgma=1.0, Ro=0.0, Rho
             vol = {}
             for k in strikes:
                 tag = "k=%5.3f" %k
                 fwPut = HestonPut(Fw, k, T, Kappa, Theta, sgma, Ro, Rho, 0.0, Xc)
                 if fwPut < max( k-Fw, 0.0): return None
                 vol[tag] = impVolFromFwPut(price = fwPut, T = T, kT = k)

             return vol

         def build_smile_np(strikes=None, Fw=1.0, T= 1.0, Kappa=1., Theta=1., sgma=1.0, Ro=0.0,
             vol = []
             for k in strikes:
                 fwPut = HestonPut(Fw, k, T, Kappa, Theta, sgma, Ro, Rho, 0.0, Xc)
                 if fwPut < max( k-Fw, 0.0): return None
                 vol.append(impVolFromFwPut(price = fwPut, T = T, kT = k))
```

```
            return np.array(vol)
```

```
In [18]: strikes = np.arange(.4, 1.6,.025)
         T     = 1.0
         theta = 0.4
         kappa = 1.5
         sigma = 1.5
         rho   = -0.9
         ro    = 0.05

         feller = sigma*sigma/(2*kappa*theta)
         print(feller)

         smile = build_smile_np(strikes, 1.0, T, kappa, theta, sigma, ro, rho, 30)

         plt.plot(strikes, smile)
```

```
1.8749999999999998
```

```
Out[18]: [<matplotlib.lines.Line2D at 0x1c809aa5f48>]
```



1.1.3 Display Functions

```
In [19]: def histo_array(keys, x, title="None"):
             LEN = len(keys)
```

```
    fig, ax = plt.subplots(1,LEN, figsize=(12,4))
    if not title == None: fig.suptitle(title)
    for n in range(LEN):
        k     = keys[n]
        lo    = np.min(x[:,n])
        hi    = np.max(x[:,n])
        bins = np.arange(lo, hi, (hi-lo)/100.)
        ax[n].hist(x[:,n], density=True, facecolor='g', bins=bins)
        ax[n].set_title(k)
        n += 1
    plt.subplots_adjust(left=.05, right=.95, bottom=.10, top=.80, wspace=.50)
    plt.show()
```

```
In [20]: def show_smiles(smiles=None):

             fig, ax = plt.subplots(1,1, figsize=(12,12))
             fig.suptitle("Sample smiles")
             ll = list(smiles.keys())
             ll.sort()
             for t in ll:
                 smile = smiles[t]
                 ax.plot( smile[:,0], smile[:,1], label = "T=%5.3f" %(t))

             ax.set_title("Sample smiles")
             ax.set_xlabel("Strike")
             ax.set_ylabel("Imp Vol")

             plt.subplots_adjust(left=.10, right=0.7)
             plt.legend(title='Smiles', bbox_to_anchor=(1.05, 1), loc='upper left')
             plt.show()
```

```
In [21]: def histo_dict(df, TAG="000000"):
             keys = list(df.keys())
             LEN = len(keys)
             fig, ax = plt.subplots(1,LEN, figsize=(12,6))
             for n in range(LEN):
                 k     = keys[n]
                 x     = df[k]
                 lo    = np.min(x)
                 hi    = np.max(x)
                 bins = np.arange(lo, hi, (hi-lo)/100.)
                 ax[n].hist(x, density=True, facecolor='g', bins=bins)
                 ax[n].set_title("%s len=%d" %(k,len(x)))
                 n += 1

             plt.savefig("pdf_%s.png" %TAG, format="png")
             plt.show()
```

1.2 Generating DataSet Functions

```
In [22]: def mkt_gen( lhs = None, kw = None, Xc=10, strikes=None):


             if lhs is None: raise Exception("No data to process")
             if kw  is None: raise Exception("No list of tags")
             x = lhs

             NUM = len(x[:,0])

             X = {}
             for k in strikes:
                 tag = "k=%5.3f" %k
                 X[tag] = np.full(NUM,0.0, dtype = np.double)
             X["T"]      = np.full(NUM,0.0, dtype = np.double)
             X["Price"]  = np.full(NUM,0.0, dtype = np.double)
             X["Strike"] = np.full(NUM,0.0, dtype = np.double)

             __tStart=time.perf_counter()
             pCount = 0
             cCount = 0
             n      = 0

             for m in range(NUM):
                 Fw    = 1.0
                 Kappa = x[m,0]
                 Theta = x[m,1]
                 sgma  = x[m,2]
                 Ro    = x[m,3]
                 Rho   = x[m,4]
                 # --
                 T     = x[m,5]
                 K     = x[m,6]

                 fwPut = HestonPut(Fw, K, T, Kappa, Theta, sgma, Ro, Rho, 0.0, Xc = Xc)
                 if fwPut < max(K-Fw,0.):
                     pCount += 1
                     continue

                 vol = build_smile(strikes=strikes, Fw=Fw, T= T, Kappa=Kappa, Theta=Theta, sgma=
                 if vol == None:
                     cCount += 1
                     continue

                 for k in strikes:
                     tag = "k=%5.3f" %k
                     X[tag][n] = vol[tag]
```

```
            X["Price"][n]  = fwPut
            X["Strike"][n] = K
            X["T"][n]      = T
            n += 1
            # -------------------------------------

        __tEnd = time.perf_counter()
        print("@ %-34s: elapsed %.4f sec" %("Seq. pricing", __tEnd - __tStart) )

        nSamples = n

        df = {}
        for s in X.keys():
            df[s] = np.copy(X[s][0:nSamples])
        print("@ %-34s: Violations Put=%d, Call=%d DB=%d out of %d" %("Info", pCount, cCoun
        return pd.DataFrame(df)
```

```
In [23]: def smiles_select(strikes, X, NUMSMILES=1):
             smiles  = {}
             ns = NUMSMILES
             N  = len(X["T"])
             for n in range(N):
                 if X["T"][n] > .5: continue

                 smile = np.ndarray(shape=(len(strikes),2), dtype=np.double)
                 for j in range(len(strikes)):
                     k = strikes[j]
                     tag = "k=%5.3f" %k
                     smile[j,0] = k
                     smile[j,1] = X[tag][n]

                 T = X["T"][n]
                 smiles[T] = smile
                 ns -= 1
                 if ns == 0: break

             return smiles
```

Constants Definition

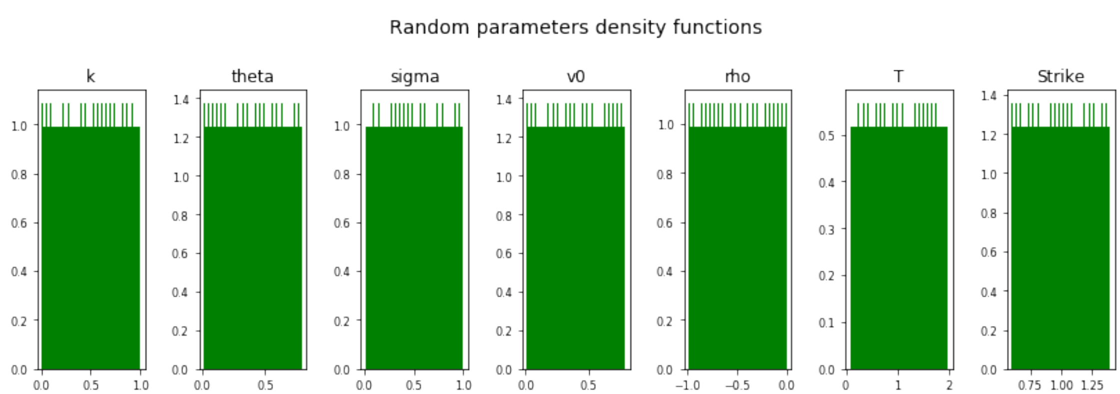
```
In [24]: verbose = False

         outputPrfx    = "full"
         challengePrfx = "test"
         targetPrfx    = "trgt"

         EPS           = 0.01
         XC            = 10.0
```

```
# bounds for the random generation of model parameters
# and contract parameters
bounds = { "k":        [ .01    , 1.00]
         , "theta":    [ .01    ,  .80]
         , "sigma":    [ .01    , 1.00]
         , "v0":       [ .01    ,  .80]
         , "rho":      [-.99    , 0.00]
         , "T":        [ 1./12., 2.00]
         , "Strike":   [ .6     , 1.40]
         }


mInt     = (1 << 15)
MInt     = (1 << 16)
NUM      = 1024
rand     = np.random.RandomState(Seed)
```

Generats and displays the random parameters

```
In [25]: __tStart= time.perf_counter()
         kw, x = lhs_sampling(rand, NUM, bounds = bounds)
         __tEnd = time.perf_counter()
         print("@ %-34s: elapsed %.4f sec" %("LHS", __tEnd - __tStart) )

         # Let's check the distribution of the parameters we have generated
         histo_array(list(kw), x, title = "Random parameters density functions")
```

```
@ Info                              : Feller violations =     438 out of    1024 ( 42.7734375 %)
@ LHS                               : elapsed 0.0141 sec
```



Generate training/test set

```
In [26]: # strikes used to build the smile used as a feature (regressor)
         strikes = np.arange(.8, 1.2, .025)
```

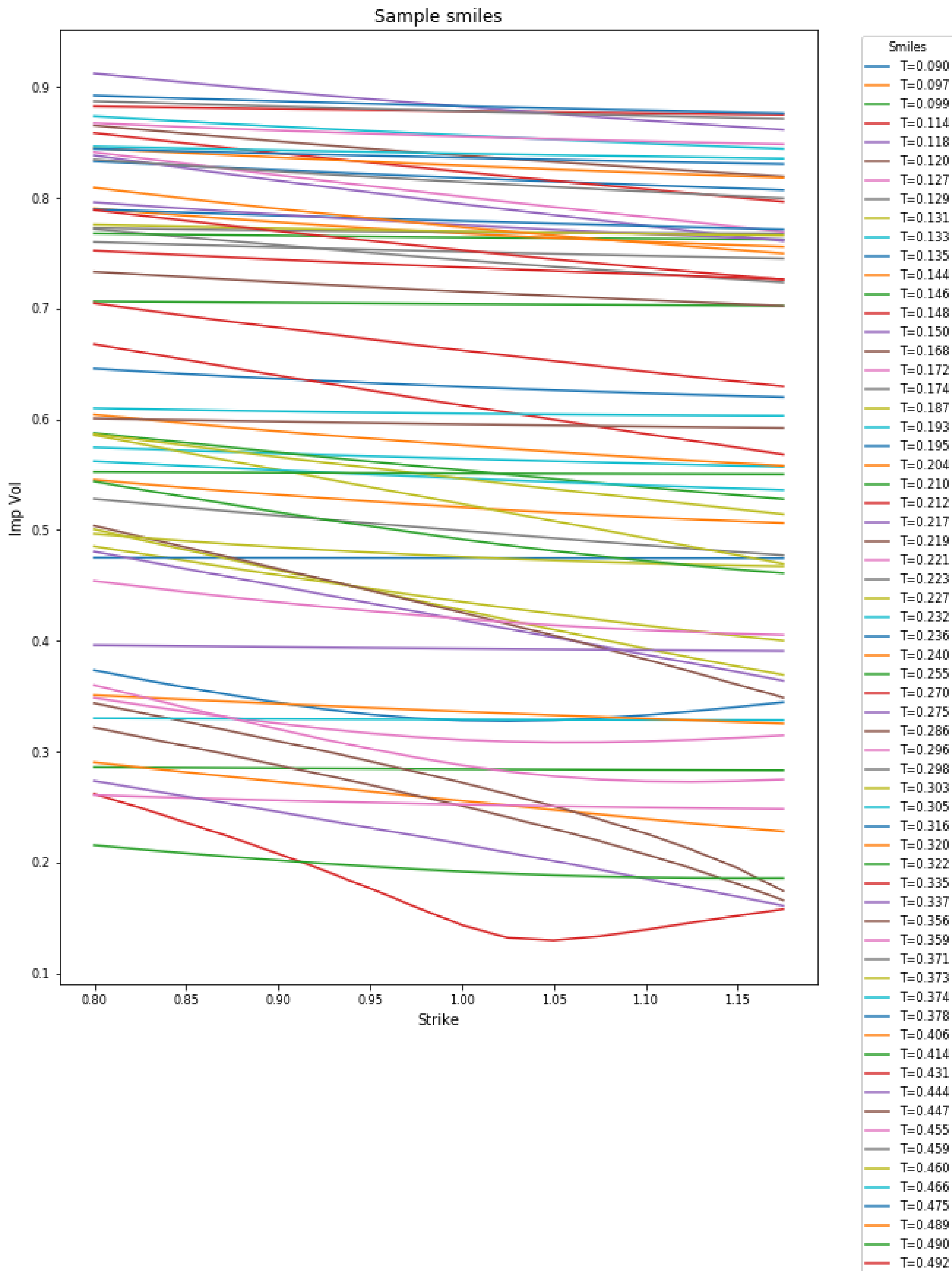
```
# Generate training/test set
__tStart=time.perf_counter()
df =  mkt_gen( lhs = x, kw = kw, Xc=XC, strikes = strikes)
__tEnd = time.perf_counter()
print("@ %-34s: elapsed %.4f sec" %("GEN", __tEnd-__tStart) )
```

```
@ Seq. pricing                        : elapsed 218.1362 sec
@ Info                                : Violations Put=1, Call=0 DB=1023 out of 1024
@ GEN                                 : elapsed 218.1404 sec
```

Select and display a subset of the generated smiles...

```
In [27]: # displays a subset of the generated smiles ....
         smiles = smiles_select(strikes, df, NUMSMILES=64)
         show_smiles( smiles=smiles)
```

Sample smiles



Selects a random subset as test set

```
In [28]: Xv, Y = train_test_split(df, test_size=0.33, random_state=rand.randint(mInt,MInt))
```

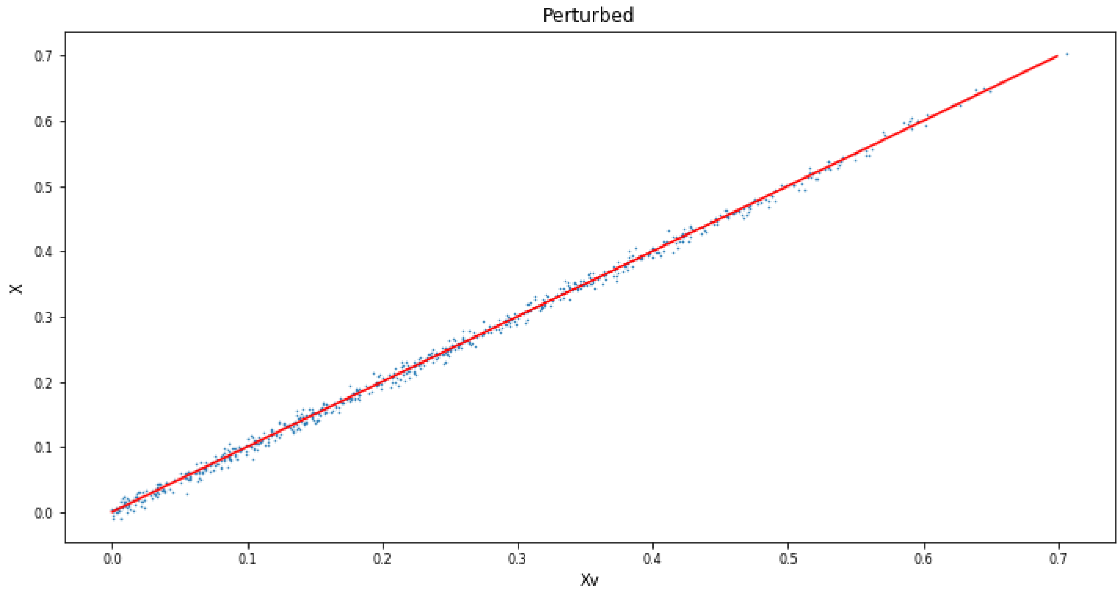
Generate some noise for the training set

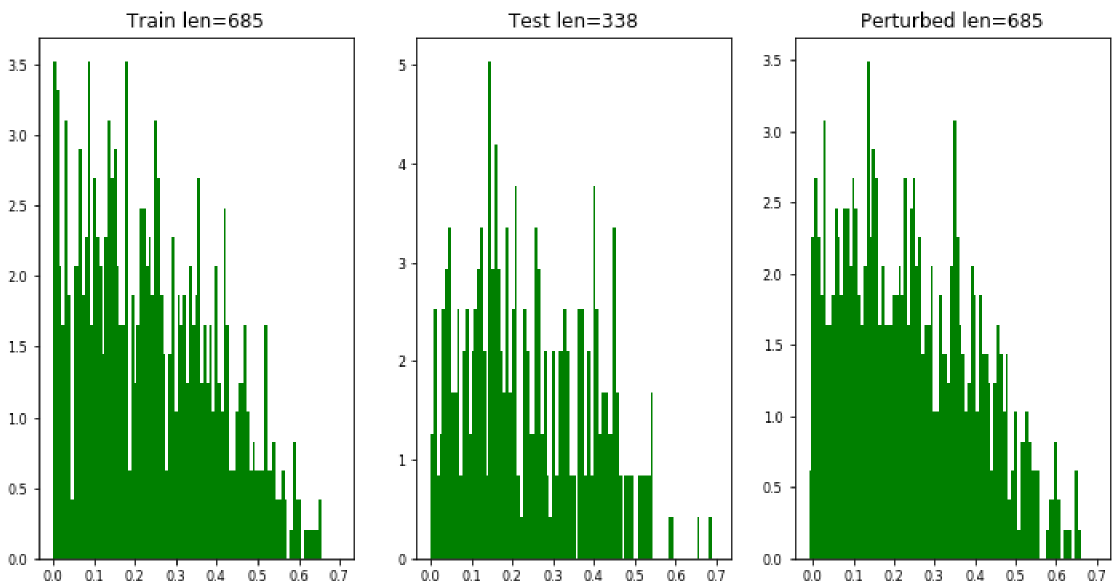
```
In [29]: if EPS > 0.0:
             X = Xv.copy()
             xl = np.min(X["Price"])
             xh = np.max(X["Price"])

             xi = rand.normal( loc = 0.0, scale = EPS*(xh-xl), size=X.shape[0])
             X["Price"] += xi
         else: X = Xv
```

Display the amount of noise

```
In [30]: import warnings
         warnings.simplefilter('ignore')
```

```
In [31]: if EPS > 0.0:
             xMin = 0.0
             xMax = max(Xv["Price"])
             v = np.arange(xMin, xMax, (xMax - xMin)/100.)

             fig, ax = plt.subplots(1,1, figsize=(12,6))

             ax.plot( Xv["Price"], X["Price"], ".", markersize=1)
             ax.plot( v, v, color="red")
             ax.set_title("Perturbed")
             ax.set_xlabel("Xv")
             ax.set_ylabel("X")

             figName = "scatter_%s.png" %(TAG)
             plt.savefig(figName, format="png")

             plt.show()

             histo_dict( {"Train": np.array(Xv["Price"]),
                          "Test": np.array(Y["Price"]),
                          "Perturbed": np.array(X["Price"]) } , TAG=TAG )
         else:
             histo_dict( {"Train": np.array(Xv["Price"]),
                          "Test": np.array(Y["Price"]) }, TAG=TAG )
```





Remove target from challenge set

```
In [32]: t = pd.DataFrame({"Price": Y["Price"]})
         y = Y.drop(columns="Price")
```

1.3 Saving Result to Disk

Write training set to disk

```
In [33]: TAG = '0001'
```

```
In [34]: outputFile = "%s_%s.csv" %(outputPrfx, TAG)
         X.to_csv(outputFile, sep=',', float_format="%.6f", index=False)
         print("@ %-34s: training data frame written to '%s'" %("Info", outputFile))
         if verbose: print(outputFile); print(X)
```

```
@ Info                              : training data frame written to 'full_0001.csv'
```

Write challenge set to disk

```
In [35]: challengeFile = "%s_%s.csv" %(challengePrfx, TAG)
         y.to_csv(challengeFile, sep=',', float_format="%.6f", index=False)
         print("@ %-34s: challenge data frame written to '%s'" %("Info", challengeFile))
         if verbose: print(challengeFile); print(y)
```

```
@ Info                              : challenge data frame written to 'test_0001.csv'
```

Write target array do disk

```
In [36]:     targetFile = "%s_%s.csv" %(targetPrfx, TAG)
             t.to_csv(targetFile, sep=',', float_format="%.6f", index=False)
             print("@ %-34s: target data frame written to '%s'" %("Info", targetFile))
             if verbose: print(targetFile); print(t)
```

```
@ Info                              : target data frame written to 'trgt_0001.csv'
```