In [5]:
```
# This Python 3 environment comes with many helpful analytics libraries
 installed
# It is defined by the kaggle/python Docker image: https://github.com/k
aggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) w
ill list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input/signal-processing'
):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) t
hat gets preserved as output when you create a version using "Save & Ru
n All"
# You can also write temporary files to /kaggle/temp/, but they won't b
e saved outside of the current session
```

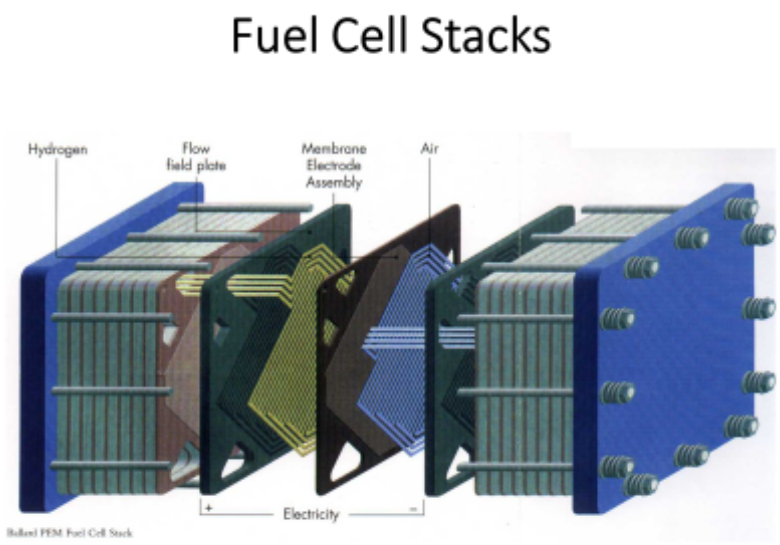
```
/kaggle/input/signal-processing/fmserp1903.txt
/kaggle/input/signal-processing/fmserp1902.txt
/kaggle/input/signal-processing/fmserp1901.txt
```

This notebook is all about analysing stochiastic signals , in order to obtain some information hidden in them. This data is a sample data from fuel cell device. More information about fuel cell can be found here. (https://www.hydrogenics.com/technology-resources/hydrogen-technology/fuel-cells/), which works on hydrogen. Data science is slowly making a progress in this area in order to predict the parameters and analyse signals.

Why data science is helpful ?

This is helpful because a fuel cell is highly complex device, it is not important to understand all parameters and to get a link between them. So we develop a black box model and then try to find relations using other algorithms such as neural network/ logistics/ SVM etc.

Here we will be exploring the signals. In order to process the signal,lets explore the data first. Signal processing is important to understand the reality/hidden information of the signals in specific domains. Not every signal are subjected to remove noise. In this tutorial, I will explore the signal in time domain and frequency domain and then to analyse the signals.



Lets explore the data. The three files added are text file contains the voltage, pressure at inlet, pressure at outlet and current in the signal. The duration of file is 10 minute each so total 30 minute signal. Start by exploring the first file ....

In [6]:
```
#Lets insert the necessary libraries to work with
```

```
from scipy import signal
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib as mpl

from scipy.stats import norm
import seaborn as sns
from scipy import stats
```

In [7]:
```
file_1 = pd.read_table('../input/signal-processing/fmserp1901.txt', sep
="\t", header=None)
file_1.head()
```

Out[7]:

	0	1	2	3
0	-0,106095	0,064650	0,019172	-0,000478
1	-0,105342	0,064215	0,015870	-0,000455
2	-0,107630	0,061248	0,014717	-0,000380
3	-0,108123	0,061536	0,019126	-0,000384
4	-0,108258	0,062158	0,019610	-0,000360

The data has no column names so columns names need to be added , The first column is **voltage**, second - **pressure at inlet** , third -**pressure at outlet** and finally current.As we can check, Voltage values are negative , values are seperated by commas , current value is negative. Voltage value needs to be and same for current. Probably the probe is connected in opposite direction.

Let's merge the files in order to analyse the 30 minute signal. We will use concat to merge the files in the signal.

In [8]:
```
file_1 = pd.read_table('../input/signal-processing/fmserp1901.txt', sep
="\t", header=None)
file_2 = pd.read_table('../input/signal-processing/fmserp1902.txt', sep
```

```
="\t", header=None)
file_3 = pd.read_table('../input/signal-processing/fmserp1903.txt', sep
="\t", header=None)
result =pd.concat([file_1, file_2, file_3], ignore_index=True)
# Replacing the comma values

vol = result.iloc[:,0] =result.iloc[:,0].str.replace(',', '.').astype(f
loat)
vol=abs(vol)
pa = result.iloc[:,1] = result.iloc[:,1].str.replace(',', '.').astype(f
loat)
pc = result.iloc[:,2] = result.iloc[:,2].str.replace(',', '.').astype(f
loat)
i  = result.iloc[:,3] = result.iloc[:,3].str.replace(',', '.').astype(f
loat)
```

In [9]:
```
result.shape
```

Out[9]:
```
(3686400, 4)
```

In [10]:
```
result.head()
```

Out[10]:

	0	1	2	3
0	-0.106095	0.064650	0.019172	-0.000478
1	-0.105342	0.064215	0.015870	-0.000455
2	-0.107630	0.061248	0.014717	-0.000380
3	-0.108123	0.061536	0.019126	-0.000384
4	-0.108258	0.062158	0.019610	-0.000360

In [11]:
```
result.describe()
```

Out[11]:

	0	1	2	3
count	3.686400e+06	3.686400e+06	3.686400e+06	3.686400e+06

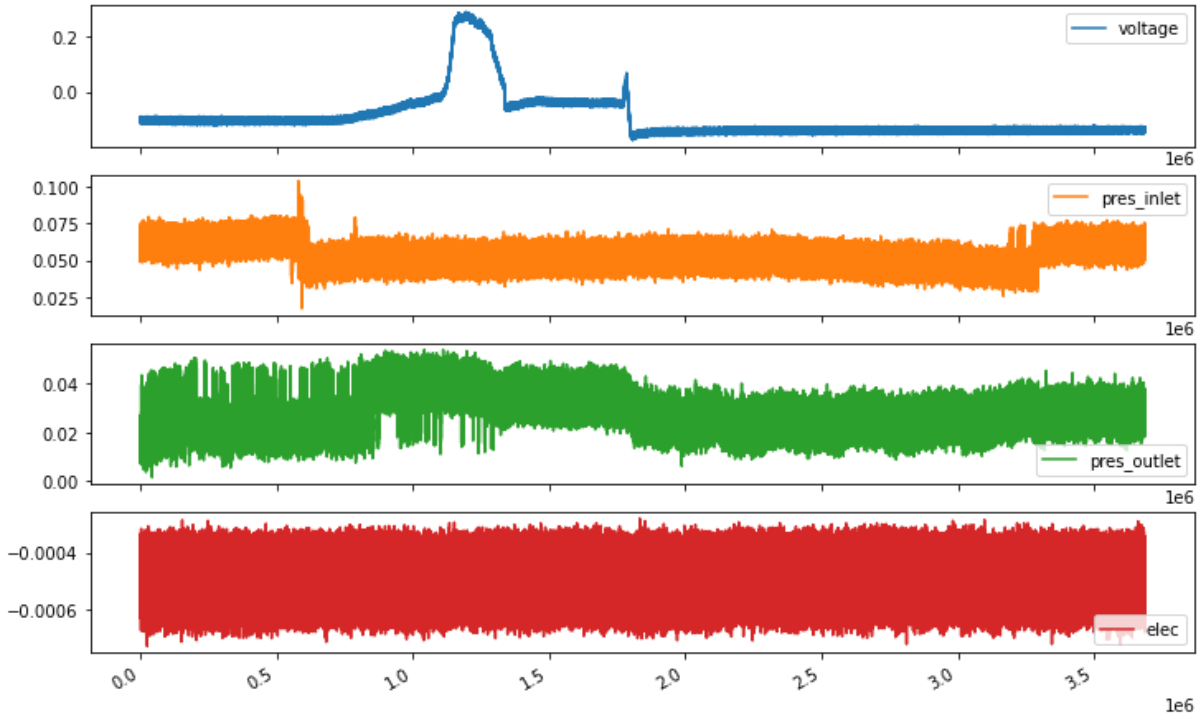
	0	1	2	3
mean	-9.500333e-02	5.386326e-02	2.838900e-02	-4.973958e-04
std	8.395515e-02	6.834604e-03	6.869441e-03	5.617059e-05
min	-1.752920e-01	1.721500e-02	1.452000e-03	-7.270000e-04
25%	-1.429140e-01	4.898700e-02	2.319000e-02	-5.390000e-04
50%	-1.347620e-01	5.277900e-02	2.713700e-02	-5.010000e-04
75%	-7.802700e-02	5.857100e-02	3.362125e-02	-4.570000e-04
max	2.888500e-01	1.041360e-01	5.407200e-02	-2.810000e-04

Considering data above, current value can be excluded from the calculations as it is way too less and it implies a reading / recording error.

In [12]:
```
result.columns = ['voltage', 'pres_inlet', 'pres_outlet', 'elec']
```

Lets add column names and plot the data !!!!

In [13]:
```
# result.plot(subplots = True )
result.plot(subplots = True , figsize=(12,8));
```



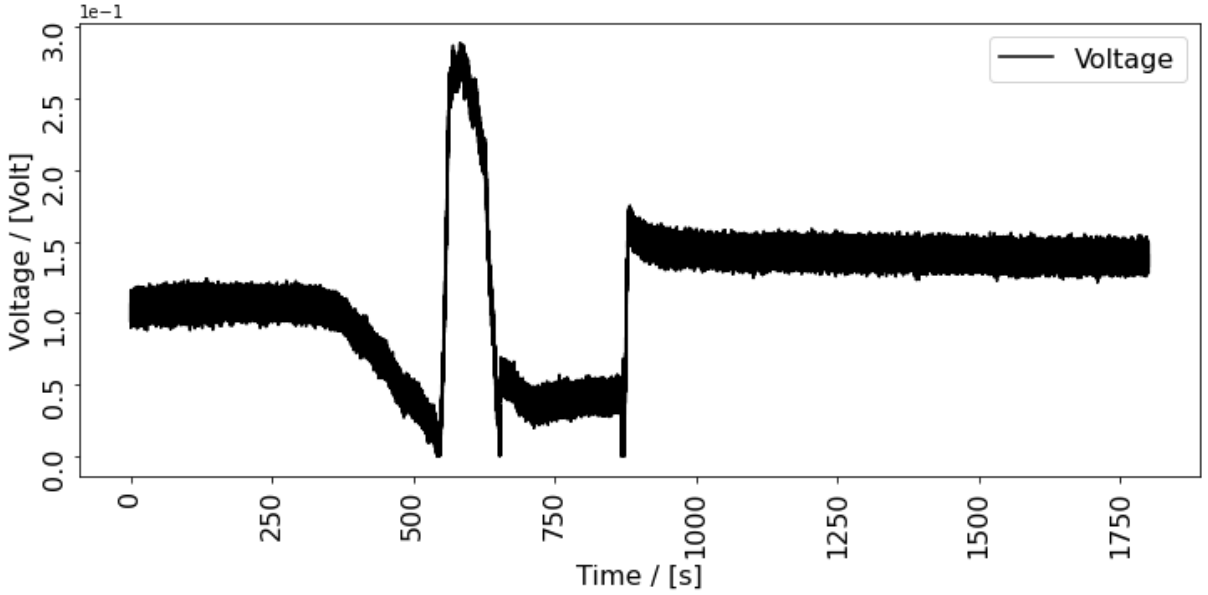
Colourful !! Signal processing is actually colourful !!! Decoding the information. Based on the plots , voltage is highly unevenly distributed which shows some issues with respect to the fluctuations. , pressure values again have fluctuations and current is dead .... Its good to keep it out. A very symmetrical distribution of current shows it has no effect by the system.

The data collection is in frequency (1s = 2048 Hz) of 2048 points. It means lets convert this to seconds and analyse further.

In [14]:
```
t= len(result)
test_list = [0 + (x * 0.00048828125) for x in range(0, t)]
x = test_list[0:t]
```

Ok let's plot the signal of voltage on time domain , with tick labels and other information .

In [15]:
```
fig = plt.figure(figsize =  (12,5))
plt.plot(x,vol, c='black', label = 'Voltage' )
plt.xticks(rotation=90, fontsize = 16, fontname = 'Arial')
plt.yticks(rotation=90,fontsize = 16, fontname = 'Arial')
plt.xlabel('Time / [s]', fontsize = 16)
plt.ylabel('Voltage / [Volt]', fontsize = 16)
plt.legend(loc='upper right', fontsize = 16)
plt.ticklabel_format(axis='y', style = 'sci', scilimits=(0,0))
```



In [16]:
```
vol.skew()
```

Out[16]:
```
0.08924600832637279
```

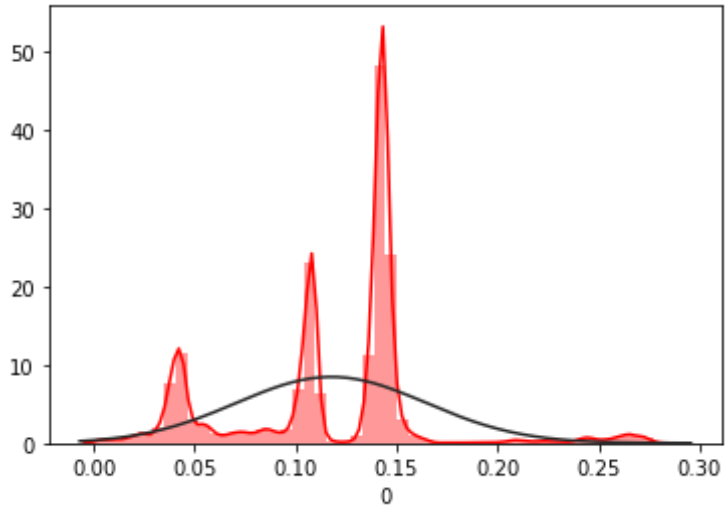
In [17]:
```
vol.kurt()
```

Out[17]:
```
1.1778835815782935
```

Let's check the voltage distribution plot !!

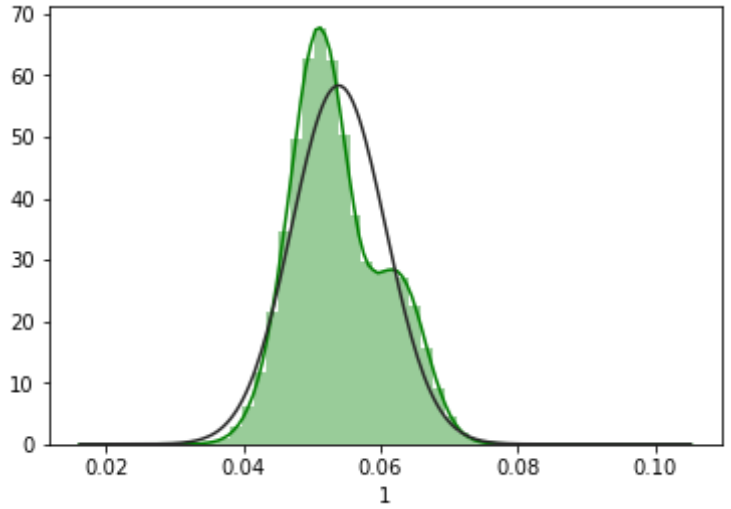
In [18]:
```
# Normal distribution for the voltage

sns.distplot(vol, fit=norm, color="r")
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8553b426d0>
```



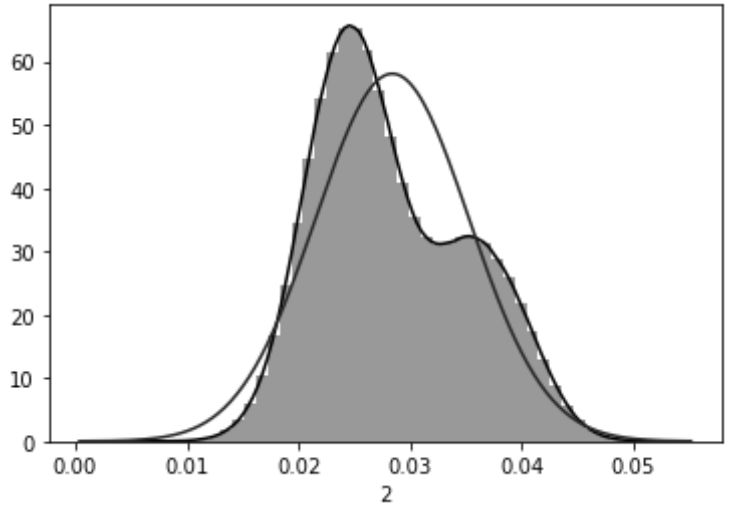
In [19]:
```
# Normal distribution for the voltage

sns.distplot(pa, fit=norm, color="g")
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8576c426d0>
```



In [20]:
```
# Normal distribution for the voltage

sns.distplot(pc, fit=norm, color="k")
```

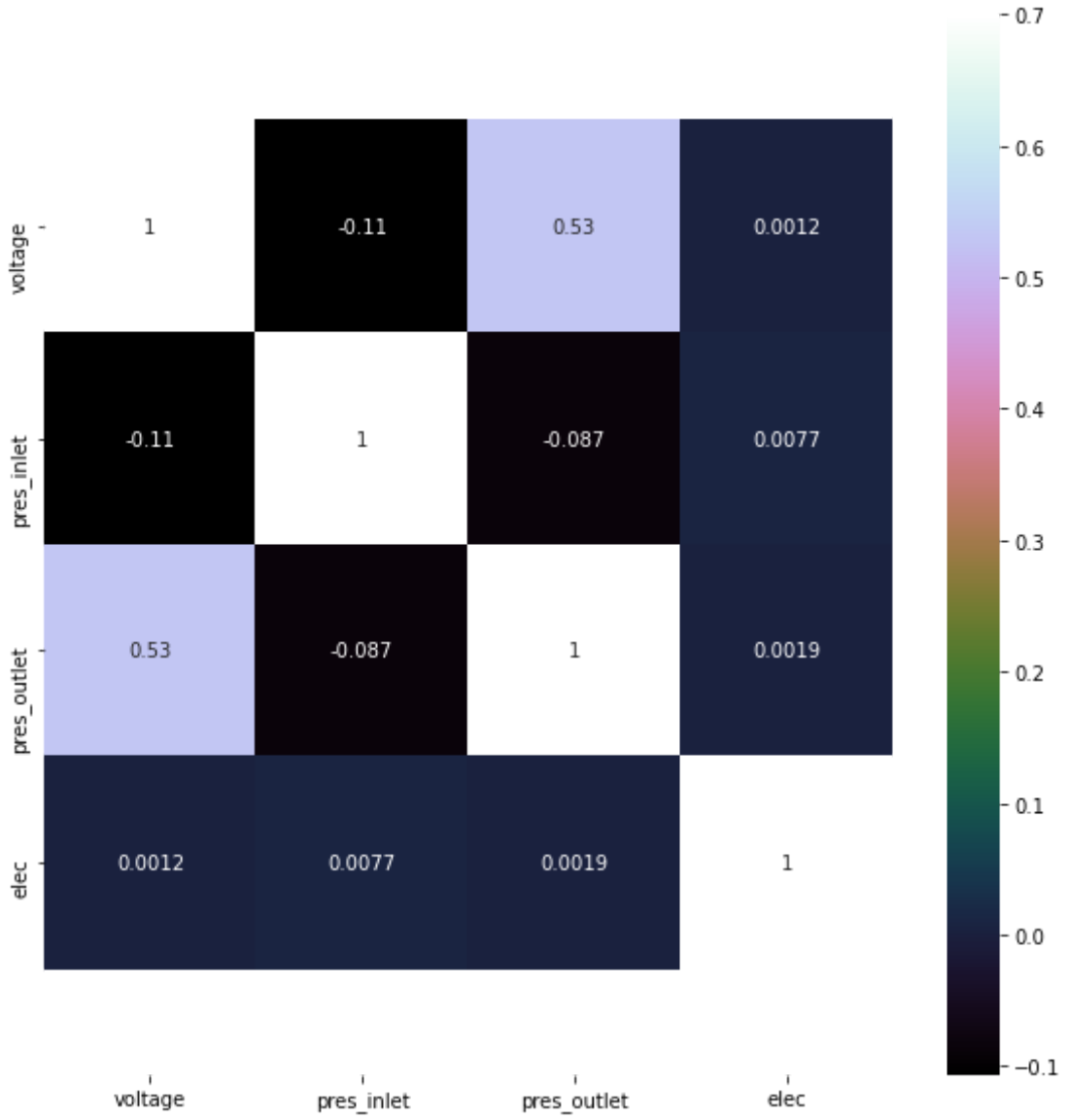
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f855a0f0dd0>



Let's check the relation among the variables , if the pressure and voltage are related to each other !!!

In [21]:
```
import seaborn as sns
corrmat = result.corr()
f, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(corrmat, vmax=0.7, square=True, cmap="cubehelix", annot=Tru
e);
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

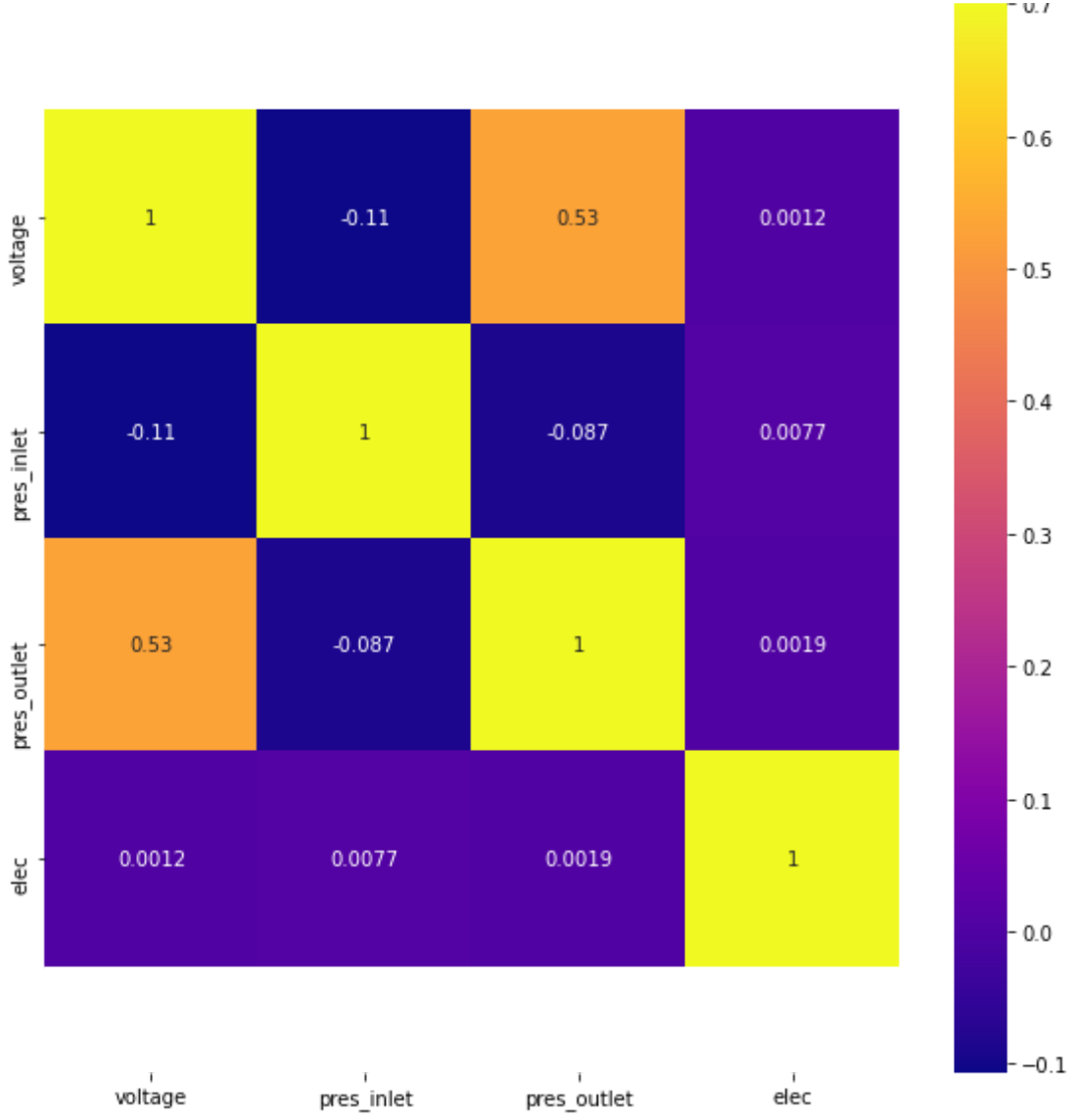
Out[21]:
```
(4.5, -0.5)
```



Let's change the cmap , always good to play and represent data in an interesting way !!!
Remember the more interesting presentation , the better the results will be....

In [22]:
```
import seaborn as sns
corrmat = result.corr()
f, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(corrmat, vmax=0.7, square=True, cmap="plasma", annot=True);
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[22]:
```
(4.5, -0.5)
```



Based on the correlation plot, we don't see a strong correlation among different parameters !!! Moreover current shows the weakest. Hence we omit analysing relation between these parameters !!! How ever in future analysis , we will introduce data which has relations among each other.

Based on the normal plot distribution, the result of voltage are not good which shows a huge shift. We will not directly apply tools to make data stationary but apply further signal treatment technique and chek if the data is normalized or not.

Now let's apply statistical models on the signals. These signals are stochastic signals which can only be represented by probabilistic distribution. Nor they have a preferred outcome , nor they can be determined. In order to analyse the signal statistical tools can be used. They can be mean , standard deviation ,skewness or kurtosis, but here's a catch in order to apply this technique your signal needs to be stationary.

Hahaha , well the signal is nowhere near to stationarity ....There are many options to check this property.
What next ?

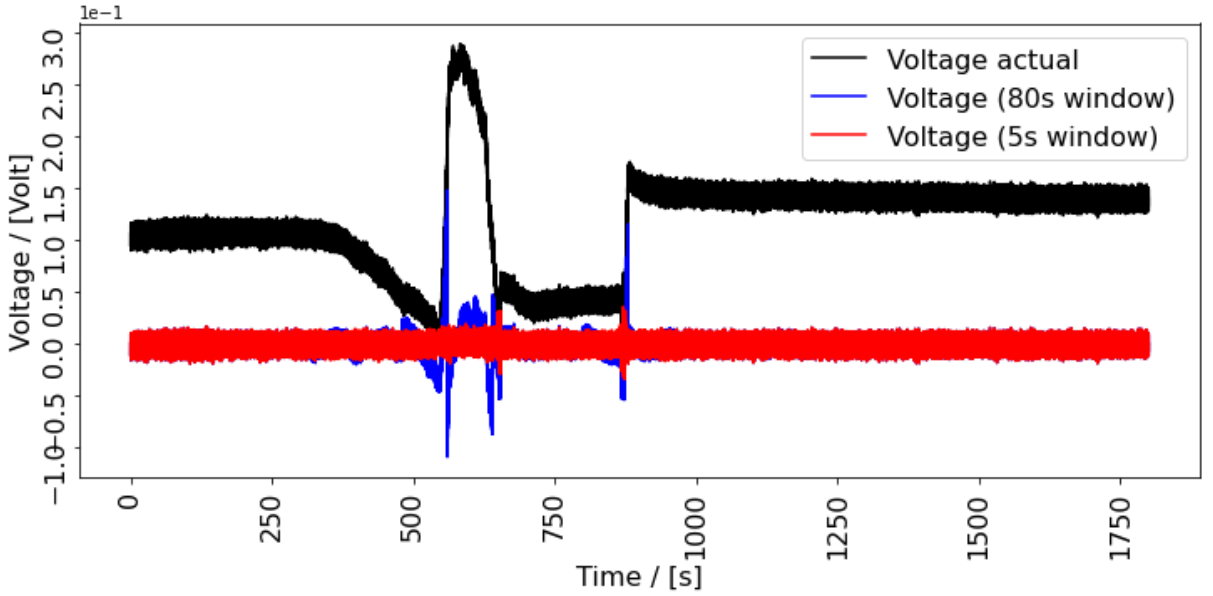
Detrend signal. Detrend signal means to remove the trends in the signal but simply removing trends will not solve the case. So we need to divide this signal into small parts , break them into small interval of time and then apply detrend procedure.

For the moment I will use time window of 5s but in the later tutorials will explain the way to calculate efficient time window.

Let's do it

In [23]:
```
vol_80 = signal.detrend(vol, axis = -1,type = 'linear', bp =  np.arange
(0,len(vol),80*2048))
vol_5 = signal.detrend(vol, axis = -1,type = 'linear', bp =  np.arange(
0,len(vol),5*2048))

fig = plt.figure(figsize =  (12,5))

plt.plot(x,vol, c='black', label = 'Voltage actual' )
plt.plot(x,vol_80, c='blue', label = 'Voltage (80s window)' )
plt.plot(x,vol_5, c='red', label = 'Voltage (5s window)' )


plt.xticks(rotation=90, fontsize = 16, fontname = 'Arial')
```

```
plt.yticks(rotation=90,fontsize = 16, fontname = 'Arial')
plt.xlabel('Time / [s]', fontsize = 16)
plt.ylabel('Voltage / [Volt]', fontsize = 16)
plt.legend(loc='upper right', fontsize = 16)
plt.ticklabel_format(axis='y', style = 'sci', scilimits=(0,0))
```



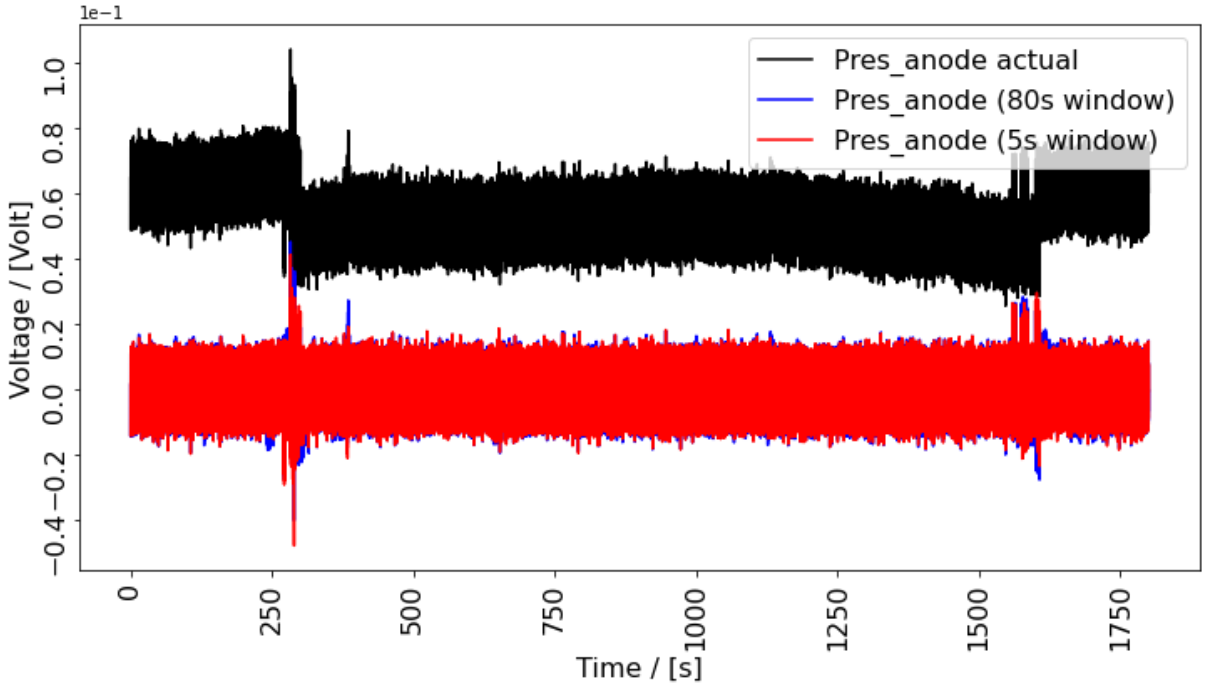
Consider the above figure, the blue signal is detrended linear at a 80s window, and it shows sharp peaks at (500-750)s interval, however red signal shows a more or less uniform stationary signal with very less peaks because it is detrended at a 5s window. Now it is entirely choice of the user to use the time window.

How ?

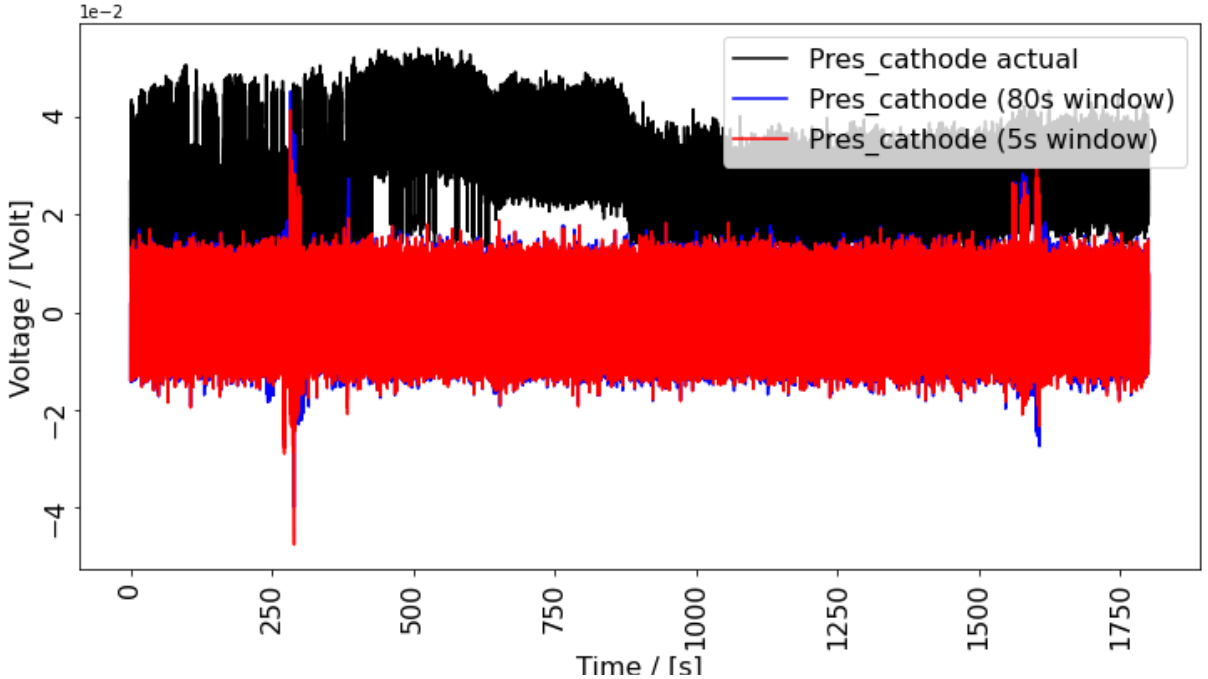
If the details needed are necessary to identify then do not use a very small window otherwise all the details will be lost. We keep up with small time windows.. Let's do it for other data too...

Let's not check the normal distribution !!!!

In [24]:
```
pa_80 = signal.detrend(pa, axis = -1,type = 'linear', bp =  np.arange(0
,len(pa),80*2048))
pa_5 = signal.detrend(pa, axis = -1,type = 'linear', bp =  np.arange(0,
len(pa),5*2048))

fig = plt.figure(figsize =  (12,6))

plt.plot(x,pa, c='black', label = 'Pres_anode actual' )
plt.plot(x,pa_80, c='blue', label = 'Pres_anode (80s window)' )
plt.plot(x,pa_5, c='red', label = 'Pres_anode (5s window)' )


plt.xticks(rotation=90, fontsize = 16, fontname = 'Arial')
plt.yticks(rotation=90,fontsize = 16, fontname = 'Arial')
plt.xlabel('Time / [s]', fontsize = 16)
plt.ylabel('Voltage / [Volt]', fontsize = 16)
plt.legend(loc='upper right', fontsize = 16)
plt.ticklabel_format(axis='y', style = 'sci', scilimits=(0,0))
```

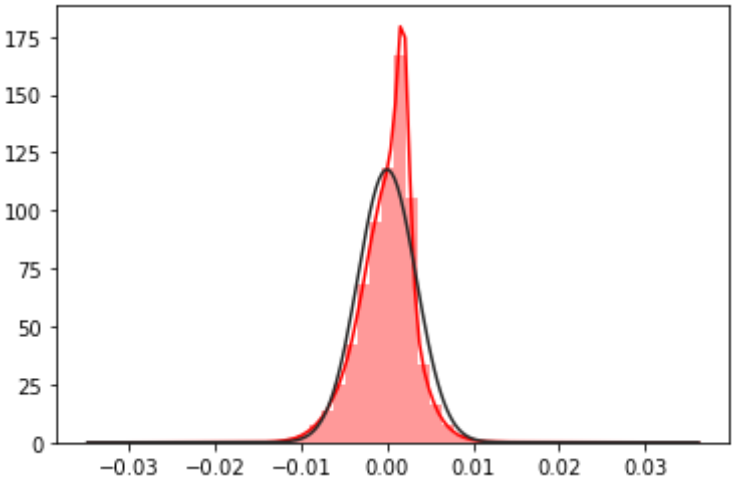


In [25]:
```
pc_80 = signal.detrend(pa, axis = -1,type = 'linear', bp =  np.arange(0
,len(pc),80*2048))
pc_5  = signal.detrend(pa, axis = -1,type = 'linear', bp =  np.arange(0
,len(pc),5*2048))

fig = plt.figure(figsize =  (12,6))

plt.plot(x,pc, c='black', label = 'Pres_cathode actual' )
plt.plot(x,pc_80, c='blue', label = 'Pres_cathode (80s window)' )
plt.plot(x,pc_5, c='red', label = 'Pres_cathode (5s window)' )


plt.xticks(rotation=90, fontsize = 16, fontname = 'Arial')
plt.yticks(rotation=90,fontsize = 16, fontname = 'Arial')
plt.xlabel('Time / [s]', fontsize = 16)
plt.ylabel('Voltage / [Volt]', fontsize = 16)
plt.legend(loc='upper right', fontsize = 16)
plt.ticklabel_format(axis='y', style = 'sci', scilimits=(0,0))
```



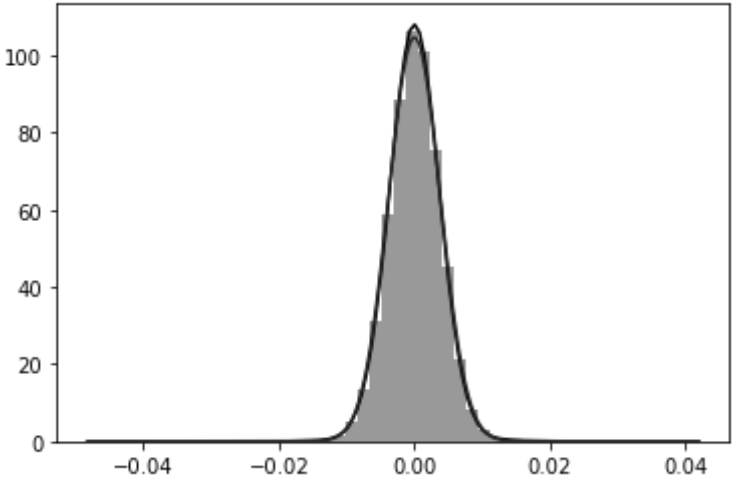
In [26]:
```
sns.distplot(vol_5, fit=norm, color="r")
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8559d4f7d0>



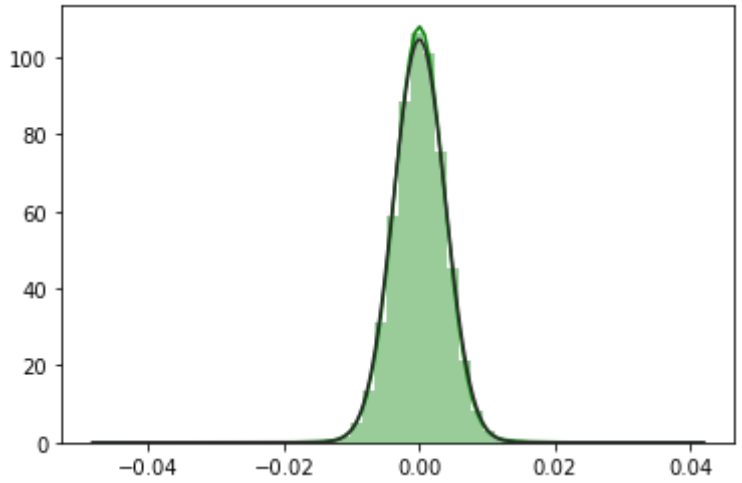
In [27]:
```
sns.distplot(pa_5, fit=norm, color="k")
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8559bbf9d0>



In [28]:
```
sns.distplot(pc_5, fit=norm, color="g")
```

Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8559ac3a90>
```



After detrending the signals we can see that the the data is normalize , however again I will mention the aim is not to normalize data completely ?

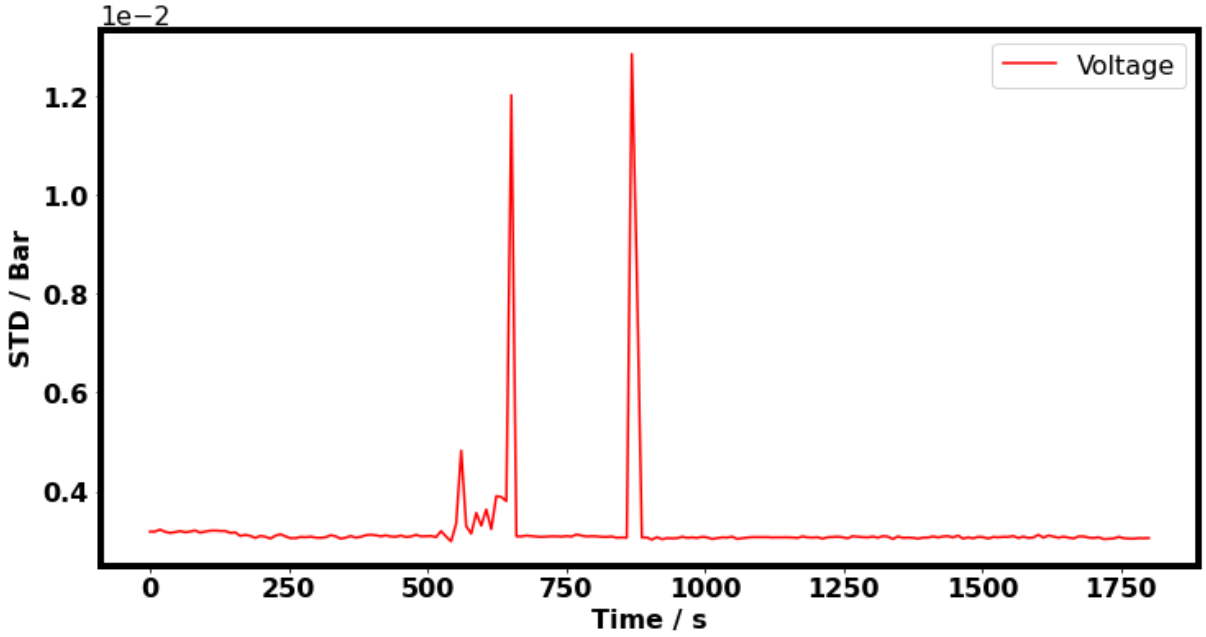
Why?

Because if a very small data window is chosen , normally we will loose high frequency components or other additional information. Moreover the more smaller the window , higher will be the computational times and more difficult it will be to anlyse signals.

We are approaching the end , signal is more or less static , we have chosen a window !!! Now its time to calcuate the std of the signals. SInce we have divided the signals into many small intervals of 5s window hence the standard deviation will be of these break points. Let's calculate the standard deviation of the of all the signals....

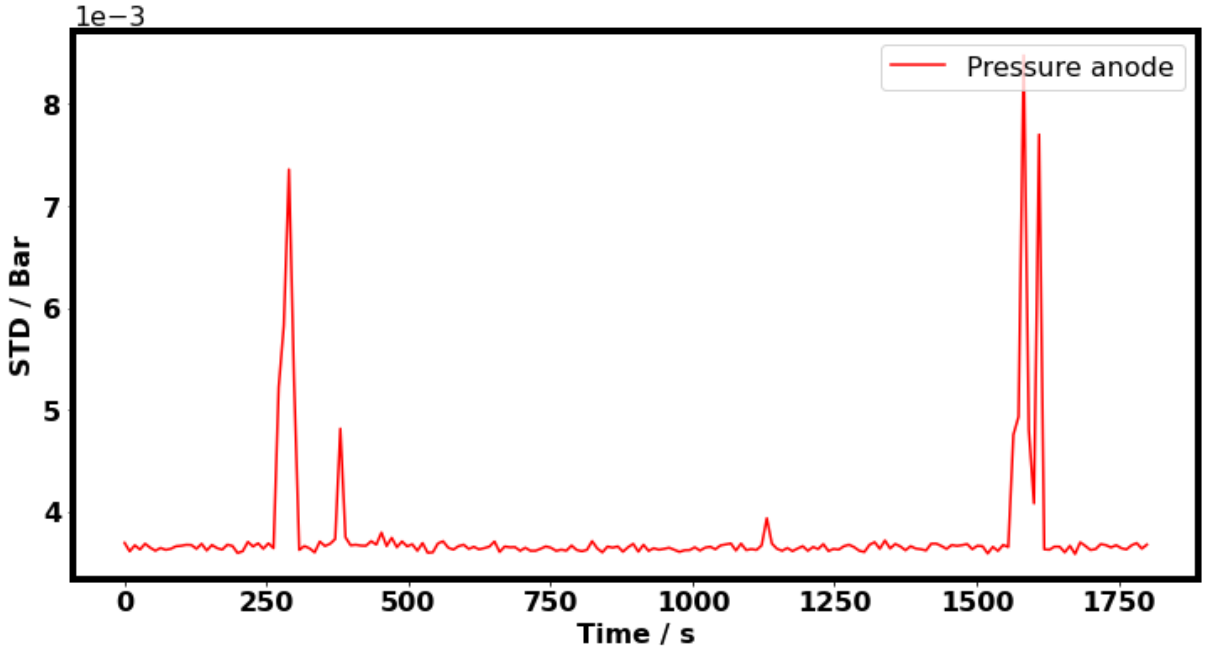
In [29]:
```
parts = 200
two_split = np.array_split(vol_5, parts)
```

```
b3 = []
for array in two_split:
    a =np.std(array)
    b3.append(a)
x1 = np.linspace(0, 1800, parts)


mpl.rcParams['font.size']=16
plt.rc('font', size=16)
plt.rcParams["axes.edgecolor"] = "black"
plt.rcParams["axes.linewidth"] = 4


fig = plt.figure(figsize =  (12, 6))


plt.plot(x1,b3, c='red', label = 'Voltage')
plt.xlabel('Time / s',  fontname = 'Arial', weight = 'bold')
plt.ylabel('STD / Bar',  fontname = 'Arial', weight = 'bold')
plt.legend(loc='upper right', fontsize = 16)

plt.xticks( fontname = 'Arial', weight = 'bold')


plt.yticks( fontname = 'Arial', weight = 'bold')
plt.ticklabel_format(axis='y', scilimits=(0,0))
```



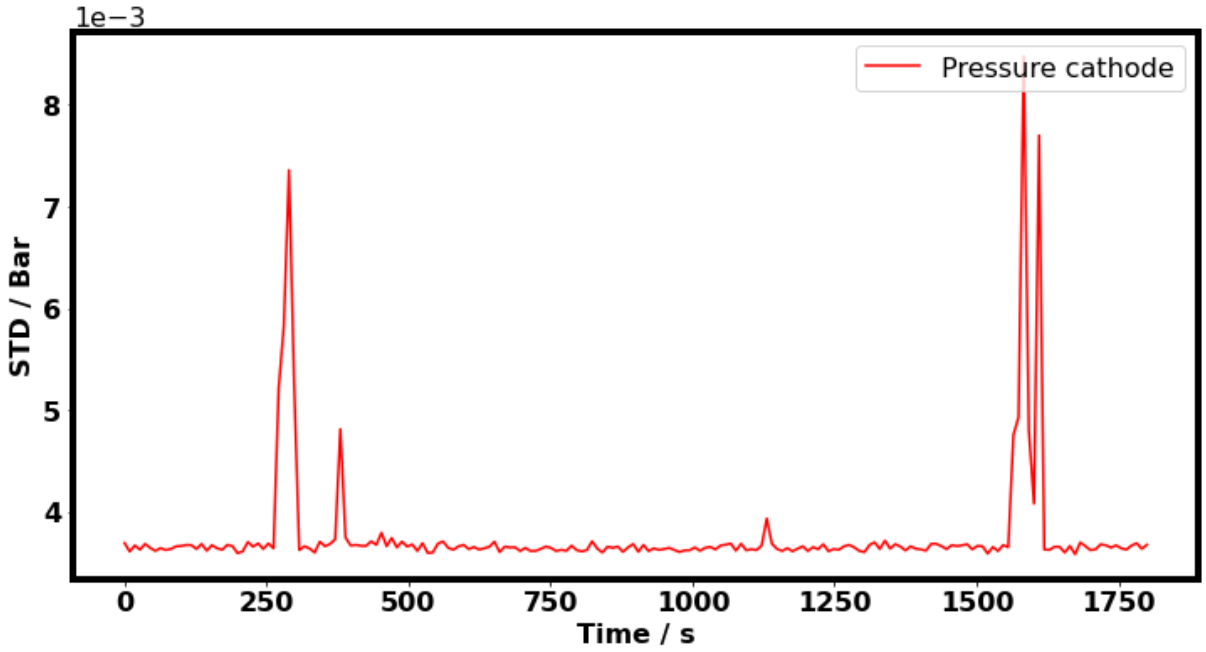
In [30]:
```
parts = 200
two_split = np.array_split(pa_5, parts)

b3 = []
for array in two_split:
    a =np.std(array)
    b3.append(a)
x1 = np.linspace(0, 1800, parts)


mpl.rcParams['font.size']=16
plt.rc('font', size=16)
plt.rcParams["axes.edgecolor"] = "black"
plt.rcParams["axes.linewidth"] = 4


fig = plt.figure(figsize =  (12, 6))
```

```
plt.plot(x1,b3, c='red', label = 'Pressure anode')
plt.xlabel('Time / s',  fontname = 'Arial', weight = 'bold')
plt.ylabel('STD / Bar',  fontname = 'Arial', weight = 'bold')
plt.legend(loc='upper right', fontsize = 16)

plt.xticks( fontname = 'Arial', weight = 'bold')


plt.yticks( fontname = 'Arial', weight = 'bold')
plt.ticklabel_format(axis='y', scilimits=(0,0))
```



In [31]:
```
parts = 200
two_split = np.array_split(pc_5, parts)

b3 = []
for array in two_split:
    a =np.std(array)
    b3.append(a)
x1 = np.linspace(0, 1800, parts)
```

```
mpl.rcParams['font.size']=16
plt.rc('font', size=16)
plt.rcParams["axes.edgecolor"] = "black"
plt.rcParams["axes.linewidth"] = 4


fig = plt.figure(figsize =  (12, 6))


plt.plot(x1,b3, c='red', label = 'Pressure cathode')
plt.xlabel('Time / s',  fontname = 'Arial', weight = 'bold')
plt.ylabel('STD / Bar',  fontname = 'Arial', weight = 'bold')
plt.legend(loc='upper right', fontsize = 16)

plt.xticks( fontname = 'Arial', weight = 'bold')


plt.yticks( fontname = 'Arial', weight = 'bold')
plt.ticklabel_format(axis='y', scilimits=(0,0))
```



Here we come to end of this notebook of signal processing. Many things can be added , more filters can be used. More tools to anlyse stationary of the signals can be added and it entirely depends on the choice of the user.

Based on the result there is no strong relation between parameters.

We see how breaking signals in to smaller parts can acheive staionarity, moreover how std values can be useful to anlyse signals.

For example comparing different voltage, what we can confirm ? 1) If the std value is higher then the disturbances , there is mismanagement of electricity , losses in the membrane or other issues related to improper water management... Same theory can be extended to fuel cell approach....