Cyclone Data Analysis

About the data

There are 6 variables and 370k records. Data is recorded once every 5 minutes over a duration of 3 years.

1. Cyclone_Inlet_Gas_Temp – Temperature of Hot gas entering the cyclone.
2. Cyclone_Gas_Outlet_Temp – Temperature of Hot gas leaving the cyclone.
3. Cyclone_Outlet_Gas_draft – Draft (pressure) of gas at outlet of cyclone.
4. Cyclone_cone_draft – Draft (pressure) of gas at cone section of cyclone.
5. Cyclone_Inlet_Draft – Draft (pressure) of gas at inlet of cyclone.
6. Cyclone_Material_Temp – Temperature of the material at the outlet of the cyclone.

Imports

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline

```

In [2]:

```
df=pd.read_csv('data.csv')
display(df.head(2))
display(df.info())
```

	time	Cyclone_Inlet_Gas_Temp	Cyclone_Material_Temp	Cyclone_Outlet_Gas_draft	Cyclone
0	1/1/2017 0:00	867.63	910.42	-189.54	
1	1/1/2017 0:05	879.23	918.14	-184.33	

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 377719 entries, 0 to 377718
Data columns (total 7 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   time                      377719 non-null  object
 1   Cyclone_Inlet_Gas_Temp    377719 non-null  object
 2   Cyclone_Material_Temp     377719 non-null  object
 3   Cyclone_Outlet_Gas_draft  377719 non-null  object
 4   Cyclone_cone_draft        377719 non-null  object
 5   Cyclone_Gas_Outlet_Temp   377719 non-null  object
 6   Cyclone_Inlet_Draft       377719 non-null  object
dtypes: object(7)
memory usage: 20.2+ MB
```

None

Data Preprocessing

In [3]:

```
df["Cyclone_Inlet_Gas_Temp"].value_counts().head(10)
```

Out[3]:

```
Not Connect    723
I/O Timeout    470
23.53          309
900.12         161
879.55         159
852.95         146
820            116
Configure      108
29.97           92
32.65           90
Name: Cyclone_Inlet_Gas_Temp, dtype: int64
```

Observation: There is some unwanted data in the dataset like "Not Connect" so we have impute those data with some other values and the data points are object type so that we also have to convert the data into float type.

In [4]:

```
df['time'] = pd.to_datetime(df['time'])
```

In [5]:

```
df.set_index("time", inplace = True)
df.index
```

Out[5]:

```
DatetimeIndex(['2017-01-01 00:00:00', '2017-01-01 00:05:00',
               '2017-01-01 00:10:00', '2017-01-01 00:15:00',
               '2017-01-01 00:20:00', '2017-01-01 00:25:00',
               '2017-01-01 00:30:00', '2017-01-01 00:35:00',
               '2017-01-01 00:40:00', '2017-01-01 00:45:00',
               ...
               '2020-08-07 11:30:00', '2020-08-07 11:35:00',
               '2020-08-07 11:40:00', '2020-08-07 11:45:00',
               '2020-08-07 11:50:00', '2020-08-07 11:55:00',
               '2020-08-07 12:00:00', '2020-08-07 12:05:00',
               '2020-08-07 12:10:00', '2020-08-07 12:15:00'],
              dtype='datetime64[ns]', name='time', length=377719, freq=None)
```

So, The data seems to be a Timeseries thats why the index of the data set is changed with the 'time' column

In [6]:

```
for feature in df.columns:
    df[feature]=pd.to_numeric(df[feature],errors='coerce')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 377719 entries, 2017-01-01 00:00:00 to 2020-08-07 12:15:00
Data columns (total 6 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   Cyclone_Inlet_Gas_Temp     376399 non-null  float64
 1   Cyclone_Material_Temp      376128 non-null  float64
 2   Cyclone_Outlet_Gas_draft   376398 non-null  float64
 3   Cyclone_cone_draft         376399 non-null  float64
 4   Cyclone_Gas_Outlet_Temp    376398 non-null  float64
 5   Cyclone_Inlet_Draft        376397 non-null  float64
dtypes: float64(6)
memory usage: 20.2 MB
```

In [7]:

```
df_sample=df.copy()
```

In [8]:

```
df_sample.head()
```

Out[8]:

time	Cyclone_Inlet_Gas_Temp	Cyclone_Material_Temp	Cyclone_Outlet_Gas_draft	Cyclone_c
2017-01-01 00:00:00	867.63	910.42	-189.54	
2017-01-01 00:05:00	879.23	918.14	-184.33	
2017-01-01 00:10:00	875.67	924.18	-181.26	
2017-01-01 00:15:00	875.28	923.15	-179.15	
2017-01-01 00:20:00	891.66	934.26	-178.32	

In [9]:

```
df_sample.isnull().sum()
```

Out[9]:

```
Cyclone_Inlet_Gas_Temp       1320
Cyclone_Material_Temp        1591
Cyclone_Outlet_Gas_draft     1321
Cyclone_cone_draft           1320
Cyclone_Gas_Outlet_Temp      1321
Cyclone_Inlet_Draft          1322
dtype: int64
```

Observation: It seems that there are some null values in the dataset so here 'bfill' method is used to fill up those missing data points

In [10]:

```
df_sample=df_sample.fillna(method="bfill")
```

In [11]:

```
1  df_sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 377719 entries, 2017-01-01 00:00:00 to 2020-08-07 12:15:00
Data columns (total 6 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   Cyclone_Inlet_Gas_Temp    377719 non-null  float64
 1   Cyclone_Material_Temp     377719 non-null  float64
 2   Cyclone_Outlet_Gas_draft  377719 non-null  float64
 3   Cyclone_cone_draft        377719 non-null  float64
 4   Cyclone_Gas_Outlet_Temp   377719 non-null  float64
 5   Cyclone_Inlet_Draft       377719 non-null  float64
dtypes: float64(6)
memory usage: 20.2 MB
```

There is no null values in the dataset so the data is now ready.

In [12]:

```
1  df_sample.describe()
```

Out[12]:

	Cyclone_Inlet_Gas_Temp	Cyclone_Material_Temp	Cyclone_Outlet_Gas_draft	Cyclone_con
count	377719.000000	377719.000000	377719.000000	377719.(
mean	726.001091	749.380080	-177.444841	-164.:
std	329.761473	352.103577	99.393401	90.:
min	0.000000	-185.000000	-456.660000	-459.:
25%	855.880000	867.060000	-247.150000	-226.'
50%	882.290000	913.200000	-215.080000	-198..
75%	901.070000	943.570000	-169.290000	-142.:
max	1157.630000	1375.000000	40.270000	488.:

Ploting the last month data

In [13]:

```
# ploting the last month data

df_sample.plot(xlim=('2020-07-07','2020-08-07'),figsize=(20,6))
```

Out[13]:

```
<AxesSubplot:xlabel='time'>
```

Resampling the data for different observations

Resampling data based on Avarage values per year

In [14]:

```
df_sample.resample(rule='A').min()
```

Out[14]:

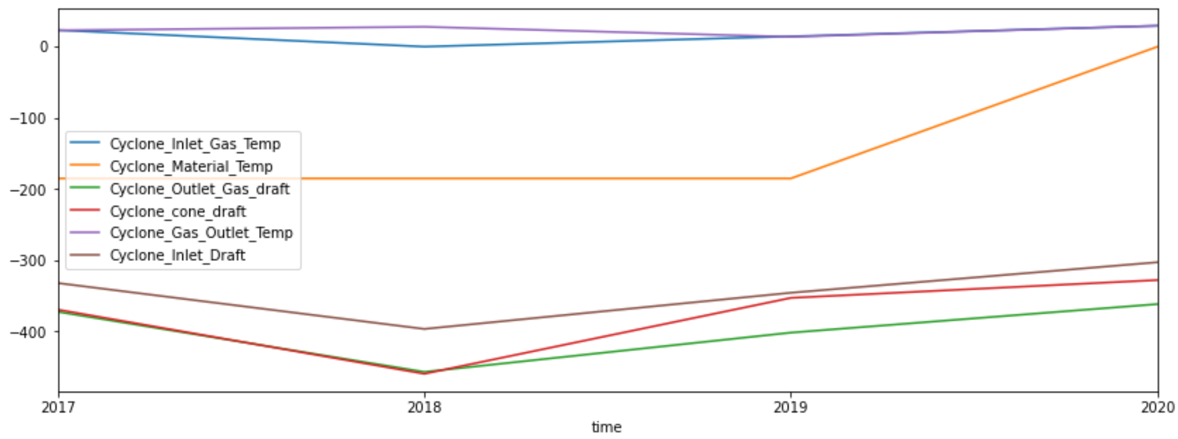
time	Cyclone_Inlet_Gas_Temp	Cyclone_Material_Temp	Cyclone_Outlet_Gas_draft	Cyclone_cone
2017-12-31	23.16	-185.0	-372.23	-
2018-12-31	0.00	-185.0	-456.66	-
2019-12-31	14.38	-185.0	-401.52	-
2020-12-31	29.32	0.0	-361.48	-

In [15]:

```
df_sample.resample(rule='A').min().plot(figsize=(14,5))
```

Out[15]:

```
<AxesSubplot:xlabel='time'>
```



Quarterly observation of the data

In [16]:

```
#Quterly data
df_sample.resample(rule='QS').max()
```

Out[16]:

time	Cyclone_Inlet_Gas_Temp	Cyclone_Material_Temp	Cyclone_Outlet_Gas_draft	Cyclone_cone
2017-01-01	1097.00	1263.91	26.45	
2017-04-01	1157.63	1358.76	31.35	
2017-07-01	1102.02	1297.69	38.11	
2017-10-01	1128.94	1375.00	24.47	
2018-01-01	1095.63	1375.00	31.87	
2018-04-01	1082.22	1252.58	27.14	
2018-07-01	1110.69	1375.00	29.19	
2018-10-01	1081.03	1265.86	33.68	
2019-01-01	1081.69	1368.98	28.38	
2019-04-01	1078.94	1291.52	38.66	
2019-07-01	1135.63	1314.44	34.14	
2019-10-01	1116.06	1089.03	40.27	
2020-01-01	1066.29	1227.06	34.27	
2020-04-01	1059.31	1085.57	30.92	
2020-07-01	1123.07	1139.22	34.13	

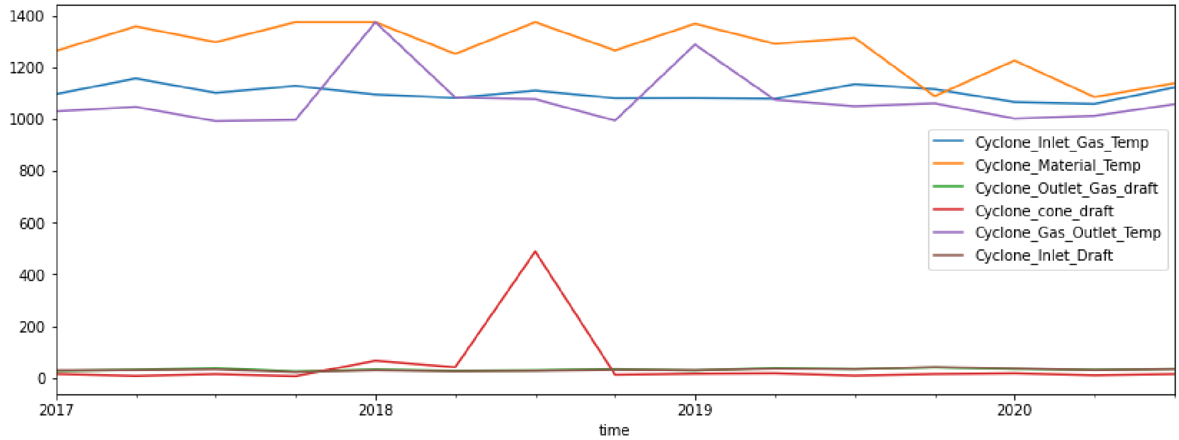
Plotting the maximum quarterly values of the data points.

In [17]:

```
1 df_sample.resample(rule='QS').max().plot(figsize=(14,5))
```

Out[17]:

<AxesSubplot:xlabel='time'>



Monthly avarage data

In [18]:

```
df_sample.resample(rule='M').mean()
```

Out[18]:

time	Cyclone_Inlet_Gas_Temp	Cyclone_Material_Temp	Cyclone_Outlet_Gas_draft	Cyclone_cone
2017-01-31	886.048201	922.470292	-183.781008	-171.4
2017-02-28	885.739420	911.682018	-193.667455	-184.0
2017-03-31	806.085026	843.912717	-164.045861	-157.5
2017-04-30	852.450715	889.300667	-181.646800	-185.1
2017-05-31	580.704478	602.554288	-110.857515	-110.1
2017-06-30	877.398731	906.937297	-180.562260	-203.8
2017-07-31	877.574540	903.563404	-202.322410	-225.8
2017-08-31	881.609334	902.123772	-195.772772	-204.9
2017-09-30	533.168457	538.598795	-97.426693	-109.1
2017-10-31	396.315532	414.013174	-70.829617	-72.1
2017-11-30	817.006612	849.990214	-197.156979	-186.3
2017-12-31	675.519918	685.078653	-149.155572	-154.8
2018-01-31	885.747397	913.128757	-222.292848	-222.1
2018-02-28	838.761214	863.714621	-200.698839	-187.1
2018-03-31	898.865691	938.677730	-232.441020	-220.7
2018-04-30	797.442910	815.458822	-214.021387	-193.7
2018-05-31	881.721221	908.762399	-236.042674	-215.1
2018-06-30	840.970834	866.831113	-232.788508	-220.6
2018-07-31	781.563918	803.896718	-203.752121	-200.9
2018-08-31	818.945095	852.909004	-223.213373	-209.4
2018-09-30	641.210595	711.132155	-153.605407	-151.2

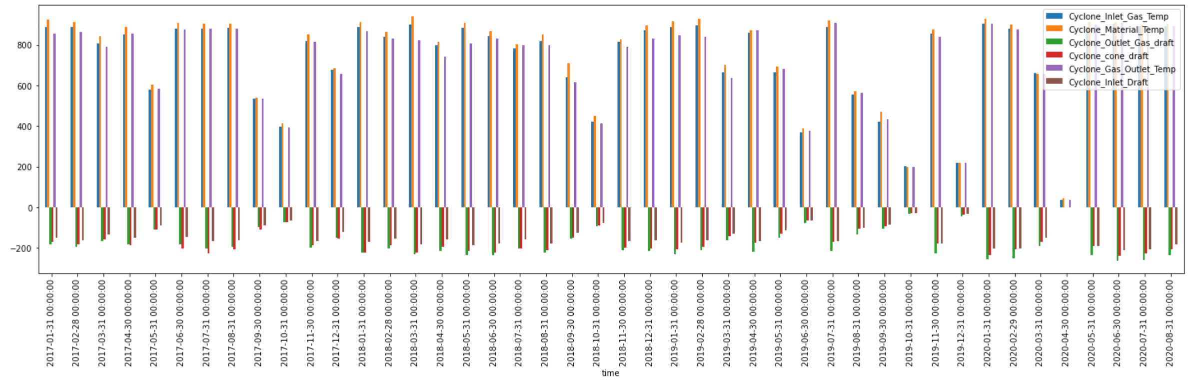
time	Cyclone_Inlet_Gas_Temp	Cyclone_Material_Temp	Cyclone_Outlet_Gas_draft	Cyclone_cone
2018-10-31	421.681108	450.617516	-94.012836	-89.4
2018-11-30	812.600793	828.024087	-210.284034	-198.1
2018-12-31	869.284226	893.831732	-213.828371	-202.4
2019-01-31	888.583247	915.227903	-231.463572	-207.5
2019-02-28	893.876138	928.290856	-212.432026	-192.4
2019-03-31	665.877214	699.783301	-160.964574	-143.1
2019-04-30	858.241594	870.551072	-217.103397	-174.0
2019-05-31	666.275366	691.193905	-151.089311	-129.6
2019-06-30	369.334275	388.935422	-75.310036	-62.5
2019-07-31	886.153292	921.525250	-215.786336	-169.7
2019-08-31	554.465114	570.508461	-132.682137	-105.0
2019-09-30	421.916277	468.923071	-106.450605	-90.9
2019-10-31	200.800585	197.544944	-32.042142	-29.2
2019-11-30	853.648869	875.228802	-225.228355	-178.5
2019-12-31	218.463456	219.970095	-44.577606	-36.1
2020-01-31	905.268843	928.639494	-256.167953	-233.4
2020-02-29	881.121920	897.768344	-249.791505	-204.8
2020-03-31	660.767640	655.922084	-190.704961	-168.5
2020-04-30	35.534472	44.449779	0.077049	0.4
2020-05-31	889.257560	910.219983	-236.304097	-190.4
2020-06-30	904.589568	909.896869	-262.467972	-236.8
2020-07-31	892.572208	901.082843	-257.193309	-226.5
2020-08-31	890.104067	904.933310	-234.077617	-206.6

In [19]:

```
# Monthly infomation about 'Cyclone_Outlet_Gas_draft'
df_sample.resample(rule='M').mean().plot(kind='bar', figsize=(25,6))
```

Out[19]:

```
<AxesSubplot:xlabel='time'>
```



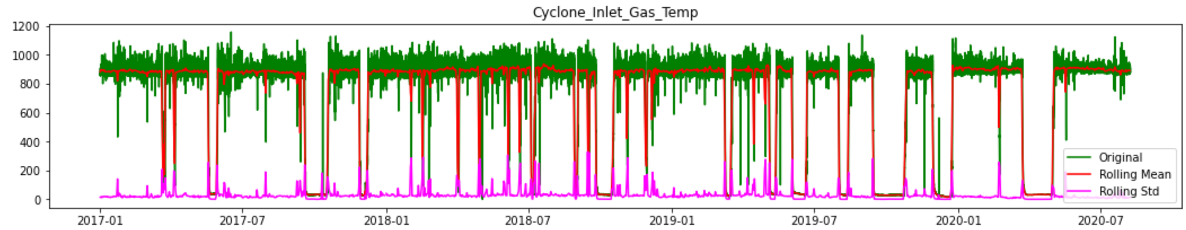
In [21]:

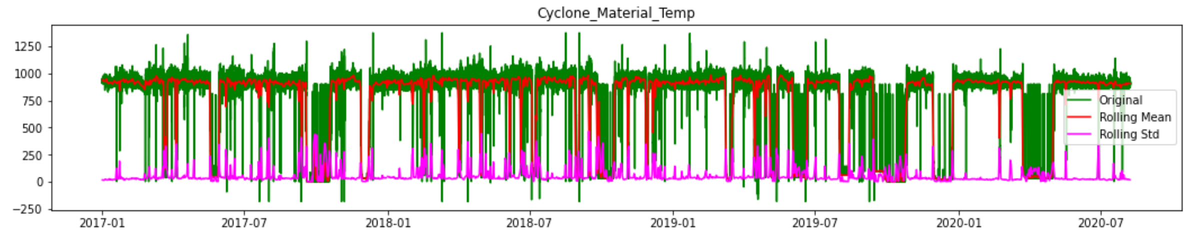
```
features=df_sample.columns
```

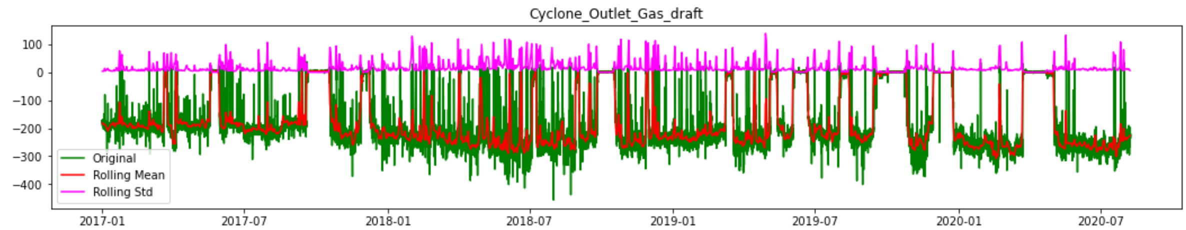
Ploting the data based on their day basis mean and Standard deviation

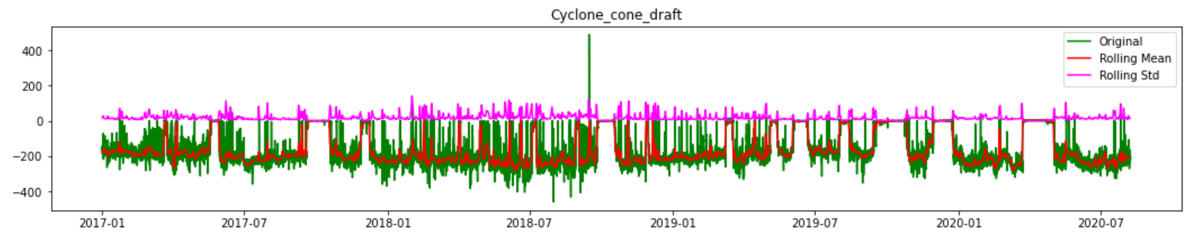
In [25]:

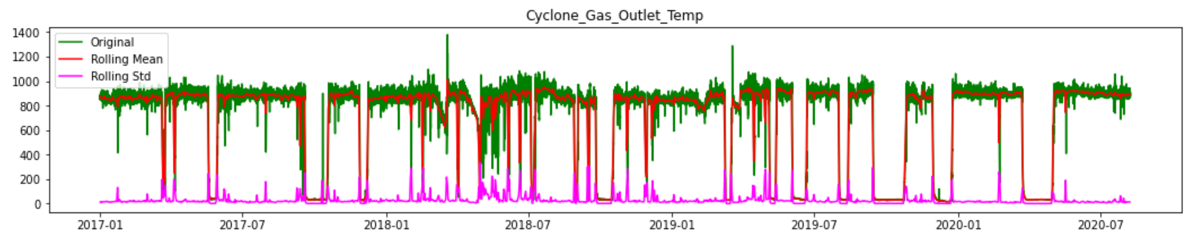
```
# Resample the entire dataset by daily average
rollmean = df_sample.resample(rule='D').mean()
rollstd = df_sample.resample(rule='D').std()
# Plot time series for each sensor with its mean and standard deviation

for name in features:
    plt.figure(figsize=(18,3))
    plt.plot(df_sample[name], color='green', label='Original')
    plt.plot(rollmean[name], color='red', label='Rolling Mean')
    plt.plot(rollstd[name], color='magenta', label='Rolling Std' )
    plt.legend(loc='best')
    plt.title(name)
    plt.show()
```

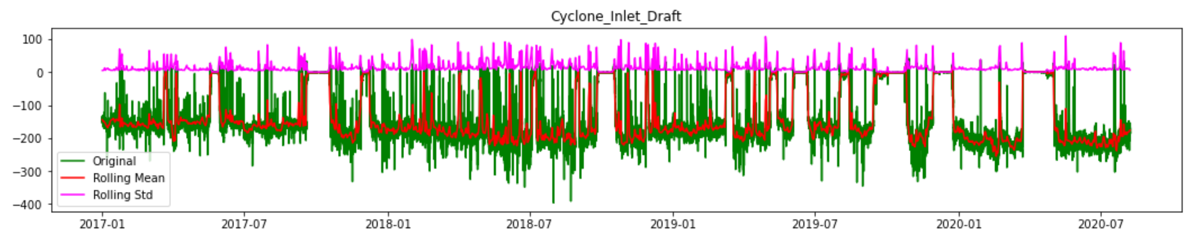












Simple moving avarage

This basically helps used to smoothing our data

In [26]:

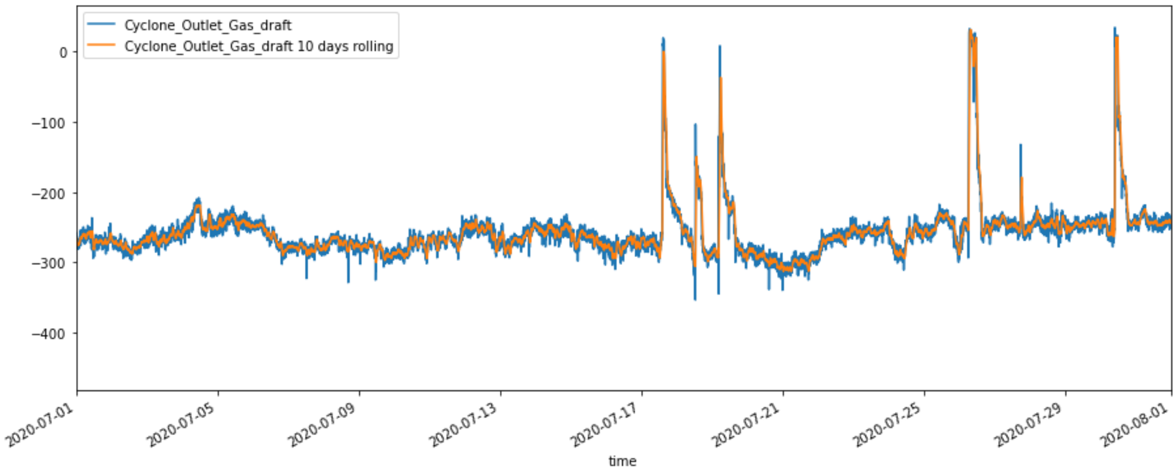
```
# Simple moving avarage
df_sample['Cyclone_Outlet_Gas_draft 10 days rolling']=df_sample['Cyclone_Outlet_Gas_dra
```

In [27]:

```
df_sample[['Cyclone_Outlet_Gas_draft','Cyclone_Outlet_Gas_draft 10 days rolling']].plot
```

Out[27]:

```
<AxesSubplot:xlabel='time'>
```

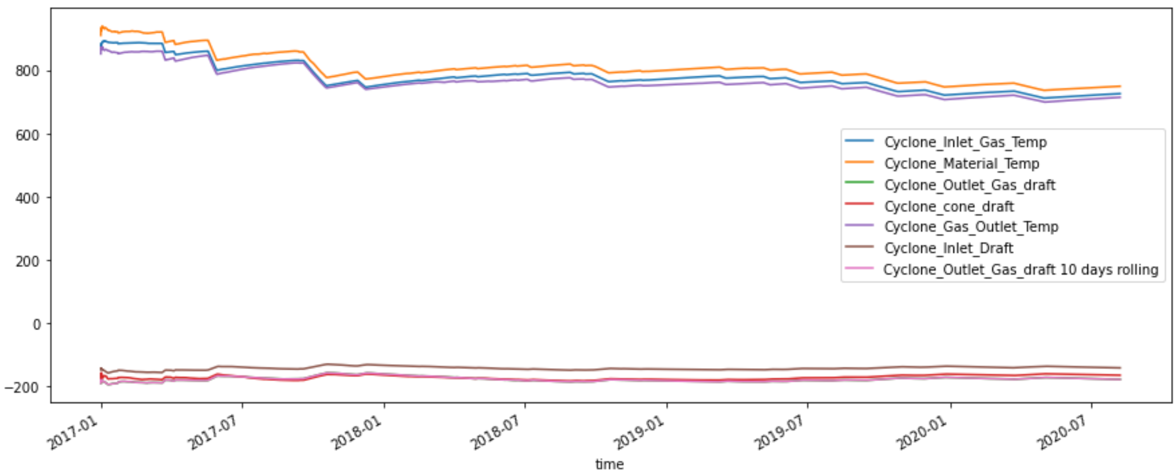


In [28]:

```
# Cumelative Moving avarage
df_sample.expanding().mean().plot(figsize=(15,6))
```

Out[28]:

```
<AxesSubplot:xlabel='time'>
```



PCA

Principle Component Analysis is used basically for feature selection and dimensionality reduction

In [29]:

```
1  df_sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 377719 entries, 2017-01-01 00:00:00 to 2020-08-07 12:15:00
Data columns (total 7 columns):
 #   Column                                 Non-Null Count   Dtype  
---  ------                                 --------------   -----  
 0   Cyclone_Inlet_Gas_Temp                 377719 non-null  float64
 1   Cyclone_Material_Temp                  377719 non-null  float64
 2   Cyclone_Outlet_Gas_draft               377719 non-null  float64
 3   Cyclone_cone_draft                     377719 non-null  float64
 4   Cyclone_Gas_Outlet_Temp                377719 non-null  float64
 5   Cyclone_Inlet_Draft                    377719 non-null  float64
 6   Cyclone_Outlet_Gas_draft 10 days rolling  377719 non-null  float64
dtypes: float64(7)
memory usage: 23.1 MB
```

Though the data set contains data for every 5 minutes thats why to reduce the computation here the data is converted into hourly manner

In [26]:

```
1  df_sample=df_sample.resample(rule='H').mean()
2
```

In [27]:

```
1  df_sample.head()
```

Out[27]:

	Cyclone_Inlet_Gas_Temp	Cyclone_Material_Temp	Cyclone_Outlet_Gas_draft	Cyclone_c
time				
2017-01-01 00:00:00	883.713333	933.111667	-182.855000	-17
2017-01-01 01:00:00	875.574167	926.837500	-179.320000	-17
2017-01-01 02:00:00	876.804167	933.455000	-177.724167	-16
2017-01-01 03:00:00	877.250000	933.257500	-178.219167	-16
2017-01-01 04:00:00	874.260833	936.023333	-178.020000	-15

In [28]:

```
df_sample.isnull().sum()
```

Out[28]:

```
Cyclone_Inlet_Gas_Temp      69
Cyclone_Material_Temp       69
Cyclone_Outlet_Gas_draft    69
Cyclone_cone_draft          69
Cyclone_Gas_Outlet_Temp     69
Cyclone_Inlet_Draft         69
dtype: int64
```

In [29]:

```
df_sample=df_sample.fillna(method="bfill")
```

In [30]:

```
df_sample.isnull().sum()
```

Out[30]:

```
Cyclone_Inlet_Gas_Temp      0
Cyclone_Material_Temp       0
Cyclone_Outlet_Gas_draft    0
Cyclone_cone_draft          0
Cyclone_Gas_Outlet_Temp     0
Cyclone_Inlet_Draft         0
dtype: int64
```

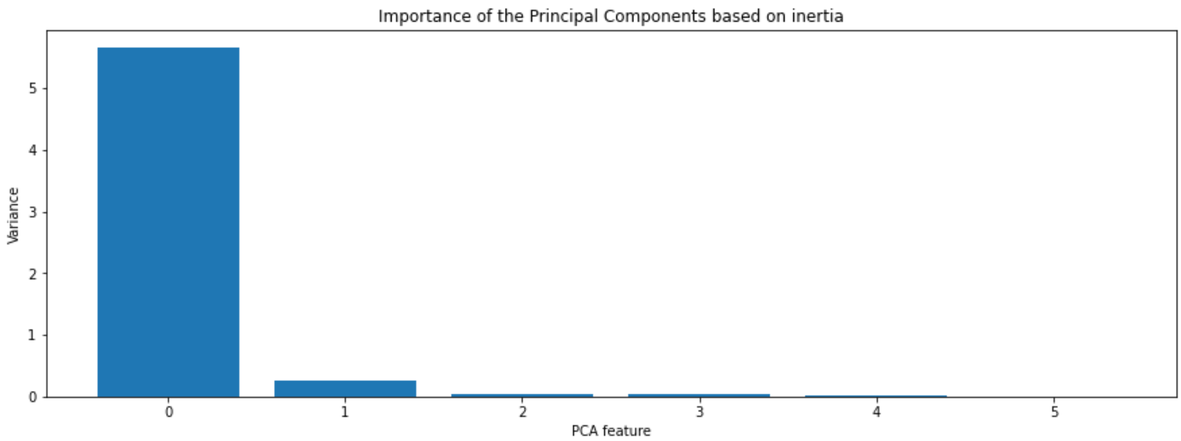
In [31]:

```
df_sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 31549 entries, 2017-01-01 00:00:00 to 2020-08-07 12:00:00
Freq: H
Data columns (total 6 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Cyclone_Inlet_Gas_Temp    31549 non-null  float64
 1   Cyclone_Material_Temp     31549 non-null  float64
 2   Cyclone_Outlet_Gas_draft  31549 non-null  float64
 3   Cyclone_cone_draft        31549 non-null  float64
 4   Cyclone_Gas_Outlet_Temp   31549 non-null  float64
 5   Cyclone_Inlet_Draft       31549 non-null  float64
dtypes: float64(6)
memory usage: 1.7 MB
```

In [32]:

```
# Standardize/scale the dataset and apply PCA
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline

x = df_sample

scaler = StandardScaler()
pca = PCA()
pipeline = make_pipeline(scaler, pca)
pipeline.fit(x)

features = range(pca.n_components_)
plt.figure(figsize=(15, 5))
plt.bar(features, pca.explained_variance_)
plt.xlabel('PCA feature')
plt.ylabel('Variance')
plt.xticks(features)
plt.title("Importance of the Principal Components based on inertia")
plt.show()
```



it seems that first two features have some effects

In [33]:

```
# Calculate PCA with 2 components
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents, columns = ['pc1', 'pc2'])
```

In [34]:

```
pca.explained_variance_ratio_
```

Out[34]:

```
array([0.97411138, 0.01287452])
```

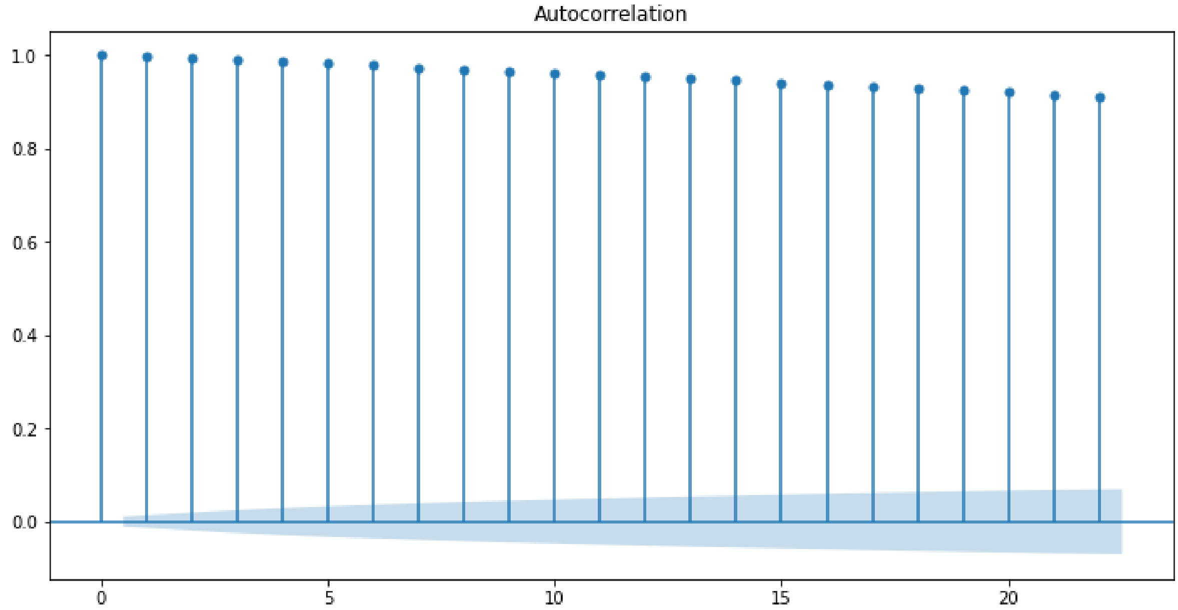
In [35]:

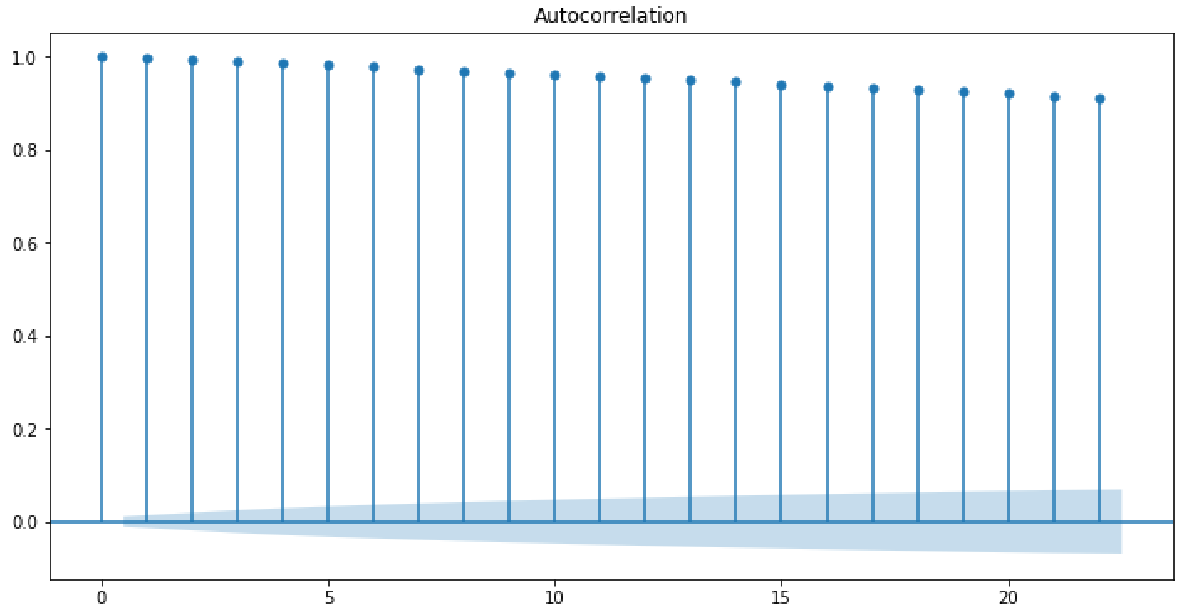
```
# principalDf.variace()
```

In [36]:

```
# Plot ACF
from statsmodels.graphics.tsaplots import plot_acf
N, M = 12, 6
fig, ax = plt.subplots(figsize=(N, M))
plot_acf(principalDf["pc1"].dropna(), lags=22, alpha=0.05,ax=ax)
```

Out[36]:





Observation:

Here we can observe that the spike has a value that is significantly different from zero. If a spike is significantly different from zero, that is evidence of autocorrelation.

Dickey Fuller Test

In [37]:

```
from statsmodels.tsa.stattools import adfuller
# Run Augmented Dickey Fuller Test
result = adfuller(principalDf['pc1'])
# Print p-value
print(result[1])
```

```
1.7454803621358068e-11
```

In [38]:

```
from statsmodels.tsa.stattools import adfuller
result = adfuller(principalDf['pc2'])
# Print p-value
print(result[1])
```

```
4.336166959889649e-30
```

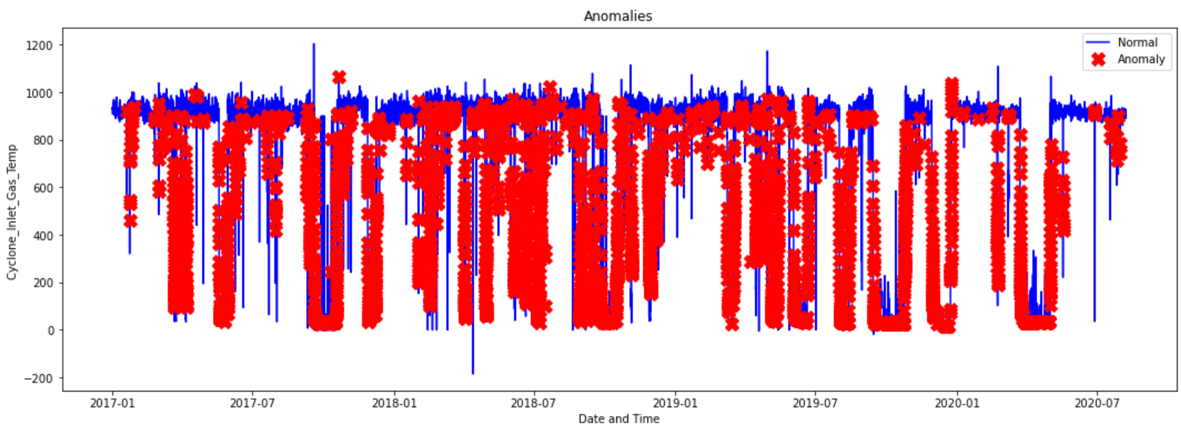
Observation:

Here we can observe tha running the Dickey Fuller test on the 1st principal component, I got a p-value of 1.754711962303802e-11 which is very small number (much smaller than 0.05). Thus, It will reject the Null Hypothesis and say the data is stationary. here we can say same thing in case of 2nd feature.

IsolationForest

In [39]:

```
# Import IsolationForest
from sklearn.ensemble import IsolationForest
# Assume that 13% of the entire data set are anomalies

outliers_fraction = 0.13
model =  IsolationForest(contamination=outliers_fraction)
model.fit(principalDf.values)
principalDf['anomaly'] = pd.Series(model.predict(principalDf.values))
# plotting
df_sample['anomaly'] = pd.Series(principalDf['anomaly'].values, index=df_sample.index)
a = df_sample.loc[df_sample['anomaly'] == -1] #anomaly
plt.figure(figsize=(18,6))
plt.plot(df_sample.iloc[:,1], color='blue', label='Normal')
plt.plot(a[df_sample.columns[0]], linestyle='none', marker='X', color='red', markersiz
plt.xlabel('Date and Time')
plt.ylabel('Cyclone_Inlet_Gas_Temp ')
plt.title('Anomalies')
plt.legend(loc='best')
plt.show();
```



In [40]:

```
df_sample['anomaly'].value_counts()
```

Out[40]:

```
 1    27447
-1     4102
Name: anomaly, dtype: int64
```

Observation: From the final result we can conclude that there are 4102 anomaly data point based on IsolationForest

In [ ]:

```

```