```
import pandas as pd
import numpy as np
import math
import os
import sys
import datetime

from sklearn.preprocessing import StandardScaler, FunctionTransformer

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.dates as mdates
plt.style.use('seaborn-whitegrid')
%matplotlib inline
plt.rcParams['figure.figsize'] = (6.0, 4.0) # set default size of plots
from sklearn.decomposition import PCA
```

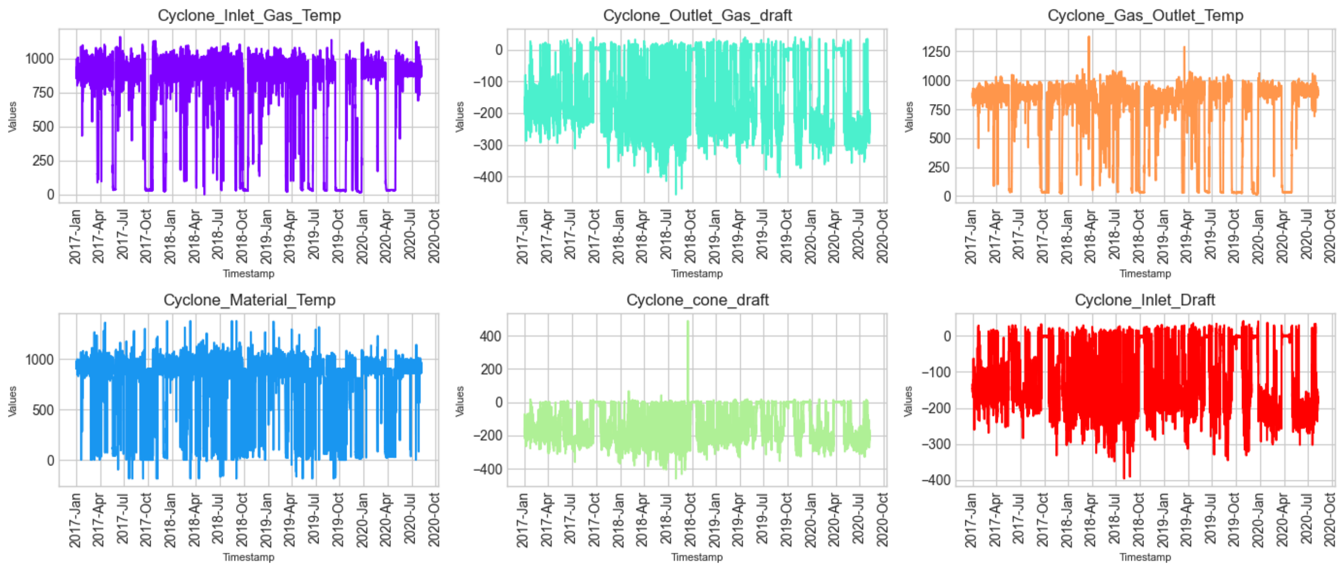
```
C:\Users\ATISHAY SG\AppData\Local\Temp\ipykernel_10104\3051317935.py:13: MatplotlibDeprecationWarning: The seaborn styles shipped by
  plt.style.use('seaborn-whitegrid')
```

```
## Mount Google drive folder if running in Colab
if('google.colab' in sys.modules):
    from google.colab import drive
    drive.mount('/content/drive', force_remount = True)
    DIR = '/content/drive/MyDrive/Colab Notebooks/MAHE/Research/MSIS_RandD/IndustrialTimeSeries/ForStudents'
    DATA_DIR = '/content/drive/MyDrive/Colab Notebooks/MAHE/Research/MSIS_RandD/IndustrialTimeSeries/Data/'
else:
    DATA_DIR = 'Data/'
```

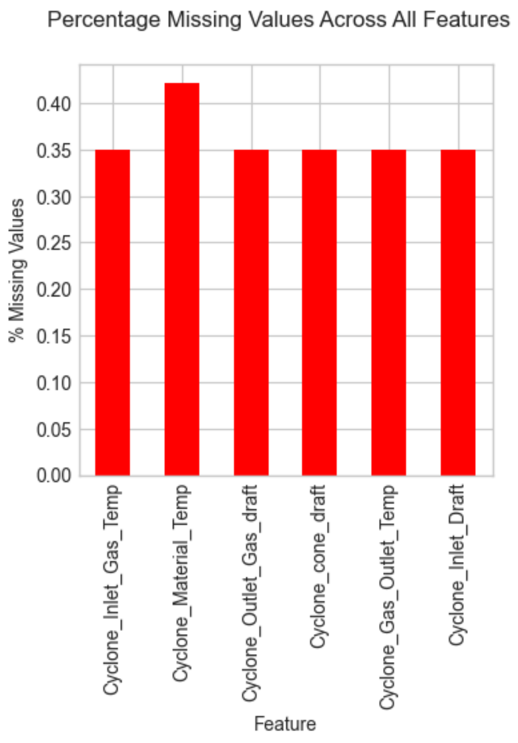
```
## Read data
FILE = 'D:/AAPS/Codes/Data/multivariate_timeseries_data.csv'
df = pd.read_csv(FILE, sep = ",", header = 0)
df['time'] = pd.to_datetime(df['time'], format='%m-%d-%Y %H.%M')
df = df.set_index('time')
df.loc[:, df.columns != 'time'] = df.loc[:, df.columns != 'time'].apply(pd.to_numeric, errors = 'coerce')
s = df.select_dtypes(include = 'object').columns
df[s] = df[s].astype('float')
df.dtypes
```

```
Cyclone_Inlet_Gas_Temp      float64
Cyclone_Material_Temp       float64
Cyclone_Outlet_Gas_draft    float64
Cyclone_cone_draft          float64
Cyclone_Gas_Outlet_Temp     float64
Cyclone_Inlet_Draft         float64
dtype: object
```

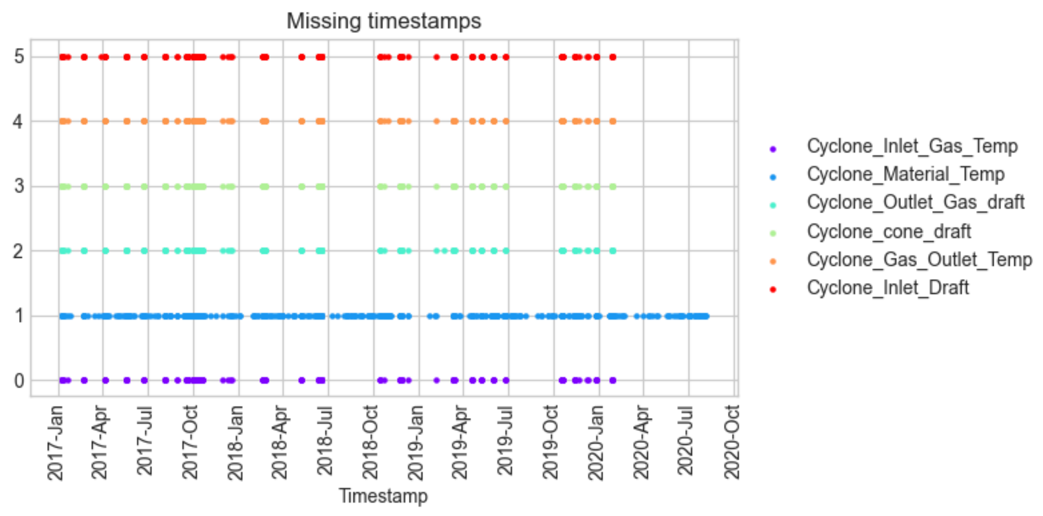
```
## Plot timeseries data
colors = cm.rainbow(np.linspace(0, 1, 6))
fig, ax = plt.subplots(2, 3, figsize = (14, 6), tight_layout = True)
for i in range(2):
  for j in range(3):
    col = df.columns[i+2*j]
    ax[i,j].plot(df.index[df[col].notna()], df[col][df[col].notna()], color = colors[i+2*j])
    ax[i, j].set_xlabel('Timestamp', fontsize = 8)
    ax[i, j].set_ylabel('Values', fontsize = 8)
    ax[i, j].set_title(col, fontsize = 12)
    ax[i, j].xaxis.set_major_locator(mdates.MonthLocator(bymonth = range(1, 12, 3)))
    ax[i, j].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%b'))
    ax[i, j].tick_params(axis = 'x', rotation = 90);
```



```
## Plot percentage of missing values (NaNs) for each feature
cutoff  = 1
fig = plt.figure(figsize=(4, 4))
fig.tight_layout()
percent_missing = (df.isna().sum() / df.shape[0]) * 100
percent_missing.plot(kind = 'bar', color = cm.rainbow(np.linspace(0, 1, 2))[(percent_missing <= cutoff).values.astype(int)])
fig.suptitle('Percentage Missing Values Across All Features', fontsize = 12)
plt.xlabel('Feature', fontsize = 10)
plt.ylabel('% Missing Values', fontsize = 10);
```



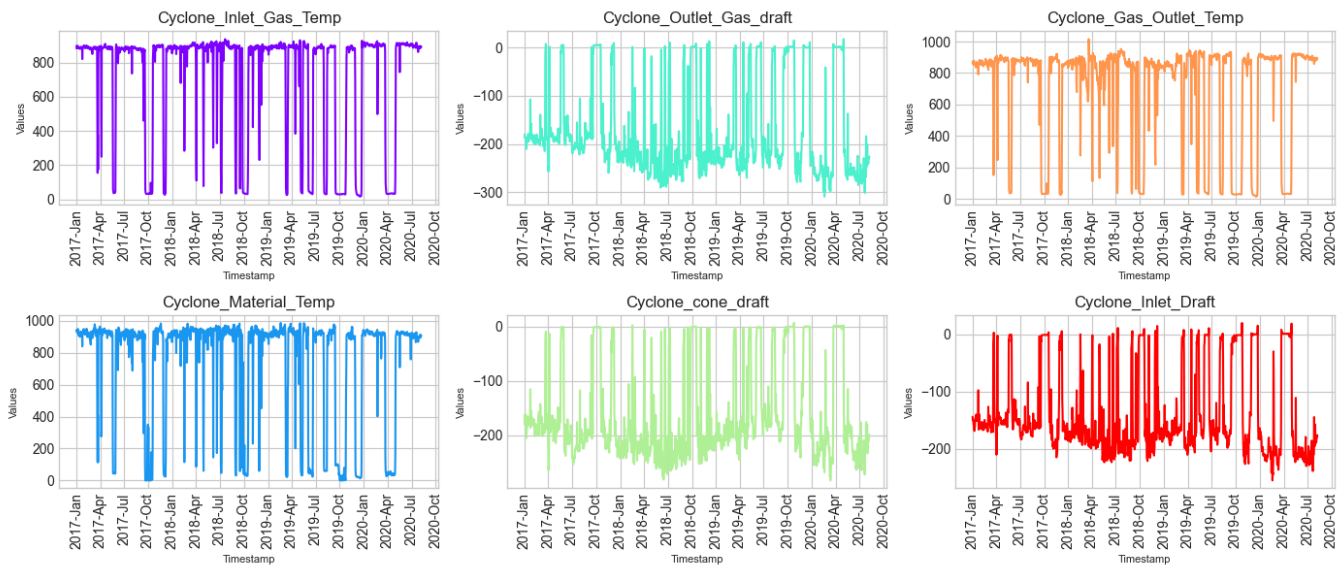
```
## Get timestamps of missing values for each column
## and plot them
fig, ax = plt.subplots(1, 1, figsize = (8, 4), tight_layout = True)
for j, col in enumerate(df.columns):
  # Get missing timestamps for this column
  missing_value_timestamps = df.index[df[col].isna()]
  # Plot missing timestamps for this column
  ax.scatter(missing_value_timestamps, [0+j] * len(missing_value_timestamps),
             color = colors[j],
             s = 5,
             label = col)

ax.set_title('Missing timestamps', fontsize = 12)
ax.set_xlabel('Timestamp', fontsize = 10)
ax.legend(loc = 'center left', bbox_to_anchor = (1, 0.5))
ax.xaxis.set_major_locator(mdates.MonthLocator(bymonth = range(1, 12, 3)))
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%b'))
ax.tick_params(axis = 'x', rotation = 90);
```



```
## Linear interpolation for missing values
#df['Cyclone_Inlet_Gas_Temp'] = df['Cyclone_Inlet_Gas_Temp'].interpolate(method = 'linear')
df.loc[:, (df.columns != 'time')] = df.loc[:, df.columns != 'time'].interpolate(method = 'linear')
(df.isna().sum() / df.shape[0]) * 100
```

```
    Cyclone_Inlet_Gas_Temp      0.0
    Cyclone_Material_Temp       0.0
    Cyclone_Outlet_Gas_draft    0.0
    Cyclone_cone_draft          0.0
    Cyclone_Gas_Outlet_Temp     0.0
    Cyclone_Inlet_Draft         0.0
    dtype: float64
```

```
## Downsample to daily frequency and plot timeseries data
df_daily = df.resample('D').mean()
fig, ax = plt.subplots(2, 3, figsize = (14, 6), tight_layout = True)
for i in range(2):
  for j in range(3):
    col = df_daily.columns[i+2*j]
    ax[i,j].plot(df_daily.index[df_daily[col].notna()], df_daily[col][df[col].notna()], color = colors[i+2*j])
    ax[i, j].set_xlabel('Timestamp', fontsize = 8)
    ax[i, j].set_ylabel('Values', fontsize = 8)
    ax[i, j].set_title(col, fontsize = 12)
    ax[i, j].xaxis.set_major_locator(mdates.MonthLocator(bymonth = range(1, 12, 3)))
    ax[i, j].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%b'))
    ax[i, j].tick_params(axis = 'x', rotation = 90);
```



```
## Data preparation for anomaly detection using numpy
# Note that 5min is the sampling period in the dataset which
# we specify and convert to seconds
sampling_period = int(pd.Timedelta('5min').total_seconds())
# We are interested in 30min data for each sample which
# we specify and convert to seconds
time_period = int(pd.Timedelta('30min').total_seconds())
# The following is a dictionary that we will use for transforming the columns
# 'identity' corresponds to no transformation, 'standard' means standardizing
scaler = {'identity': FunctionTransformer(lambda x: x), 'standard': StandardScaler()}
df_transformed = pd.DataFrame(scaler['standard'].fit_transform(df))
df_transformed.columns = df.columns.copy()
df_transformed.index = df.index.copy()
ncols_reshape = int(pd.Timedelta(str(time_period/sampling_period)+'S').total_seconds())
nrows_reshape = df_transformed.shape[0]//ncols_reshape
df_samples = pd.DataFrame(np.concatenate([np.array(df_transformed[feature])[0:nrows_reshape*ncols_reshape].reshape(nrows_reshape, ncols_
df_samples.index = pd.date_range(df_transformed.index.min(),
                                 df_transformed.index.max() + pd.DateOffset(days = 1),
                                 normalize = True,
                                 freq = str(time_period)+'S')[0:df_samples.shape[0]]
df_samples.head()
```

	0	1	2	3	4	5	6	7	8	9	...	26	27	
2017-01-01 00:00:00	0.429433	0.464612	0.453816	0.452633	0.502307	0.462519	0.457338	0.479265	0.496420	0.493495	...	0.464489	0.463846	0.49
2017-01-01 00:30:00	0.545613	0.460366	0.439684	0.544157	0.452117	0.531238	0.573506	0.540303	0.537747	0.538684	...	0.480697	0.480697	0.50
2017-01-01 01:00:00	0.450298	0.474316	0.436924	0.522959	0.408417	0.499820	0.506731	0.536156	0.499687	0.553567	...	0.459005	0.510539	0.44
2017-01-01 01:30:00	0.412936	0.440836	0.502550	0.430555	0.428827	0.433861	0.491450	0.482077	0.520790	0.491052	...	0.496966	0.471505	0.46
2017-01-01 02:00:00	0.468827	0.495363	0.438895	0.441139	0.447265	0.455574	0.518149	0.541553	0.536810	0.514258	...	0.498559	0.488908	0.49

5 rows × 36 columns

Exploratory Data Analysis

```
from statsmodels.tsa.stattools import adfuller
adf, pvalue, usedlag, nobs, critical_values, icbest = adfuller(df_samples[0])
```

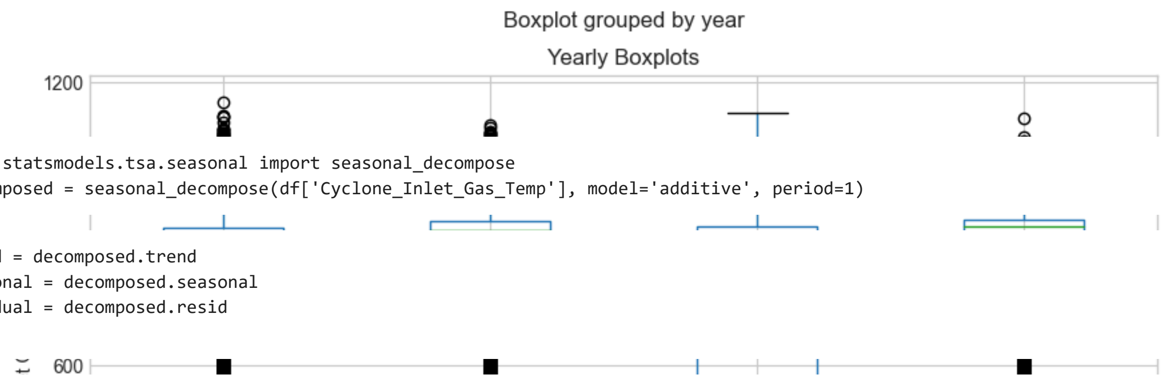
```
df['year'] = [d.year for d in df.index]
df['month'] = [d.strftime('%b') for d in df.index]
year = df['year'].unique()
```

```
df['year'].unique()
```

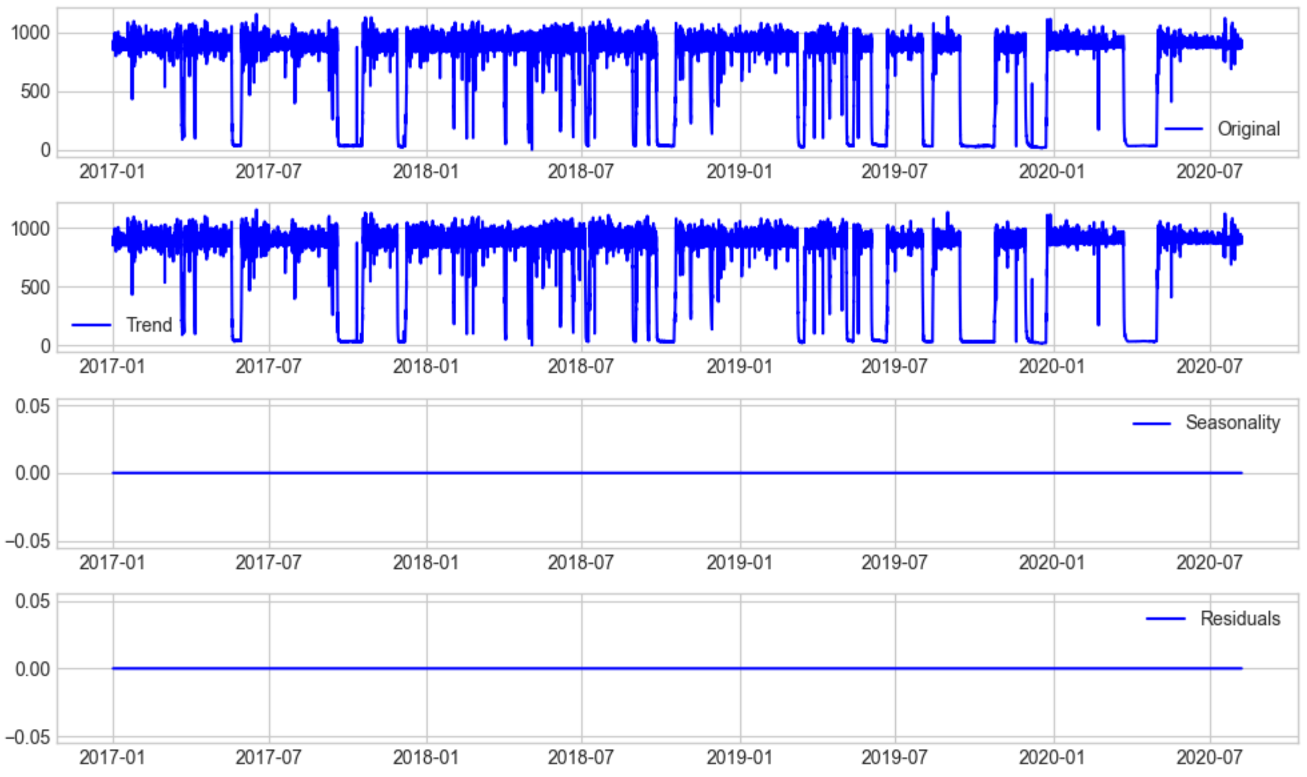
```
     array([2017, 2018, 2019, 2020], dtype=int64)
```

```
# Plot yearly boxplots
fig, ax = plt.subplots(figsize=(10, 6))
df.boxplot(column='Cyclone_Inlet_Gas_Temp', by='year', ax=ax)
ax.set_xlabel('Year')
ax.set_ylabel('Cyclone Inlet Gas Temp')
ax.set_title('Yearly Boxplots')

# Plot monthly boxplots
fig, ax = plt.subplots(figsize=(10, 6))
df.boxplot(column='Cyclone_Inlet_Gas_Temp', by='month', ax=ax)
ax.set_xlabel('Month')
ax.set_ylabel('Cyclone Inlet Gas Temp')
ax.set_title('Monthly Boxplots')
```

```
    Text(0.5, 1.0, 'Monthly Boxplots')
```



```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposed = seasonal_decompose(df['Cyclone_Inlet_Gas_Temp'], model='additive', period=1)

trend = decomposed.trend
seasonal = decomposed.seasonal
residual = decomposed.resid

plt.figure(figsize=(10, 6))
plt.subplot(411)
plt.plot(df['Cyclone_Inlet_Gas_Temp'], label='Original', color='blue')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend', color='blue')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal, label='Seasonality', color='blue')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals', color='blue')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```



Autocorrelation

```
#Autocorrelation is simply the correlation of a series with its own lags.
#Plot lag on x axis and correlation on y axis
#Any correlation above the confidence interval is significant

from statsmodels.tsa.stattools import acf
acf_40 = acf(df['Cyclone_Inlet_Gas_Temp'], nlags=40)
plt.plot(acf_40, marker='o')
```

```
    [<matplotlib.lines.Line2D at 0x2548cb0ca60>]
```