Ablation Medical Device Sensor Measurement Anomaly Detection

1. Objective

Ablation medical devices sensors record many measurements during the ablation of nerves for pain relief.

Anomalous measurements, particularly with impedance, may indicate potential failure of the therapy to ablate the nerve tissue for successful pain relief. The ability to forecast ahead of time during the ablation process if such anomalous events may happen can lead to interventions to prevent such treatment failures.

In [36]:
```
import os, glob
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.tsa.stattools as ts
from statsmodels.tsa.seasonal import seasonal_decompose
import fbprophet
import numpy as np
import math
```

2. Data

Data is from the logs from a commercially available ablation device.

Data consists of time, temperature, voltage, current, impedance and other measurements. Parameter of interest is impedance of the tissue which could increase to high levels indicating potential treatment failure.

The log contains 3 different types of impedance profile behaviors.

1. Normal
2. Sudden high impedance
3. Gradual high impedance

Approach

Evaluate the time series data for the 3 types for stationarity. Use Facebook Prohpet to make forecast and detect anomaly.

1. No smoothing
2. Moving Average smoothing
3. Exponential smoothing

3. Data Exploration

Read the sensor logs and extract the impedance data. Visualize the impedance time series for the 3 types,

1. Normal
2. Sudden high impedance
3. Gradual high impedance

In [2]:
```
# read sensor log
def read_log(filepath):
    is_cooled = False
    headerlines = None
    sw_version = None

    column_names = None
    list_lines = list()
    with open(filepath, 'rt') as f:
        for headerlines, line in enumerate(f):
            line = line.strip()
            character = line.split(' ')
            character = [i for i in character if i != '']

            if len(character) > 7:
                if character[0] == 'chan':
                    column_names = character
                    if 'measT' not in column_names:
                        return None, None, None
                else:
                    list_lines.append(character)
            elif 7 > len(character) > 1:
                if character[0] == 'Cooled':
                    is_cooled = True
                if character[0] == 'SW':
                    sw_version = character[2].split('-')
                    if len(sw_version) > 0:
                        sw_version = float(sw_version[0])
                    else:
                        sw_version = None

    df = pd.DataFrame(list_lines, columns=column_names)
    for col in df.columns:
        if col != 'chan' and col != 'error':
            df[col] = df[col].apply(pd.to_numeric, errors='coerce')

    return df, is_cooled, sw_version
```

In [3]:
```
def readCRGlog(filepath):
    df, is_cooled, sw_version = read_log(filepath)
    if df is None and is_cooled is None:
        return None, None, None

    df_channels_list = split_channels(df)

    return df_channels_list, is_cooled, sw_version
```

In [4]:
```
def split_channels(df):
    channels_list = ['A', 'B', 'C', 'D']
```

```python
    # df['dur(ms)'] /= 1000.0
    # df.rename(columns={'dur(ms)': 'dur(s)'}, inplace=True)

    list_df_channels = list()
    for channel in channels_list:
        df_channel = df[df['chan'] == channel].reset_index()

        if len(df_channel) > 0 and df_channel['dur(ms)'].iloc[-1] > 0.0:
            index_multiple = df_channel.index[(df_channel['dur(ms)'].diff()) < 0].toli

            if len(index_multiple) > 0:
                start = 0
                index_multiple.append(len(df_channel) - 1)

                for i, idx in enumerate(index_multiple):
                    df_multiple = df_channel[start: idx]
                    start = idx + 1
                    if len(df_multiple) > 0:
                        list_df_channels.append([df_multiple, i, channel])
            else:
                list_df_channels.append([df_channel, 0, channel])

    return list_df_channels
```

Reading of sensor log

In [5]:

```python
# read logs
log_data = [redacted]\high_impedance\\test\\20160521160530.log"

df_list, iscooled, sw_version = readCRGlog(log_data)
```

Extract impedance and time from sensor log for Normal

In [6]:

```python
# extract impedance data
def extract_data(df):
    df = df[['dur(ms)', 'z']]
    df = df.drop_duplicates(subset=['dur(ms)'], keep='last').reset_index(drop=True)

    # change timestep, assume each is 1 hour to speed up forecasting
    time_step = int(df['dur(ms)'].diff().median())
    actual_index = pd.date_range(df.index[0], periods=len(df), freq='240L')
    df.index = actual_index
    df = df.drop(['dur(ms)'], axis=1)
    return df
```

In [7]:

```python
df_normal = extract_data(df_list[0][0])
df_normal
```

Out[7]:

	z
1970-01-01 00:00:00.000	252.8
1970-01-01 00:00:00.240	251.8
1970-01-01 00:00:00.480	250.4
1970-01-01 00:00:00.720	249.6
1970-01-01 00:00:00.960	248.3
...	...
1970-01-01 00:02:29.040	244.3
1970-01-01 00:02:29.280	244.4
1970-01-01 00:02:29.520	245.3
1970-01-01 00:02:29.760	244.5
1970-01-01 00:02:30.000	218.7

626 rows × 1 columns

Evaluate for stationarity

p value is too high, non-stationary.

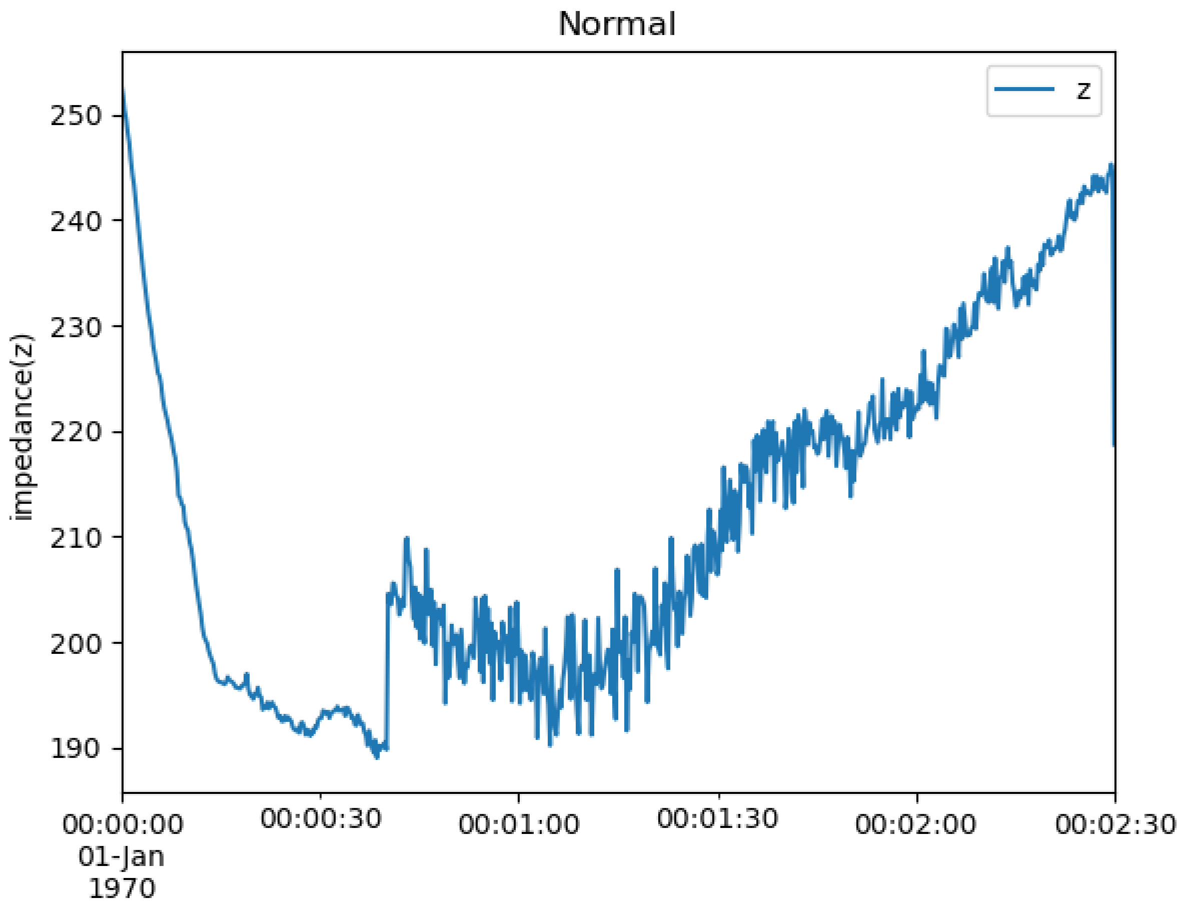
In [8]:
```
# evaluate for stationarity
def adfuller_test(df):
    dftest = ts.adfuller(df,)
    dfoutput = pd.Series(dftest[0:4],
                         index=['Test Statistic', 'p-value', 'Lags Used', 'Observatior
    for k, v in dftest[4].items():
        dfoutput['Critical Vlaue (%s)'%k] = v
    print(dfoutput)
```

In [9]:
```
adfuller_test(df_normal)
```

```
Test Statistic           -1.438641
p-value                   0.563617
Lags Used                15.000000
Observations Used       610.000000
Critical Vlaue (1%)      -3.441116
Critical Vlaue (5%)      -2.866290
Critical Vlaue (10%)     -2.569300
dtype: float64
```

Plot to visualize time series.

In [10]:
```
# plotting
df_normal.plot.line();
plt.ylabel('impedance(z)');
plt.title('Normal');
```



Extract impedance and time from sensor log for Sudden High Impedance

In [11]:
```
# extract impedance data
df_sudden = extract_data(df_list[1][0])
df_sudden
```

Out[11]:

	z
1970-01-01 00:00:00.000	165.3
1970-01-01 00:00:00.240	165.6
1970-01-01 00:00:00.480	165.6
1970-01-01 00:00:00.720	166.3
1970-01-01 00:00:00.960	166.9
...	...
1970-01-01 00:00:30.720	395.9
1970-01-01 00:00:30.960	513.6
1970-01-01 00:00:31.200	809.0
1970-01-01 00:00:31.440	1040.1
1970-01-01 00:00:31.680	221.1

133 rows × 1 columns

Evaluate for stationarity

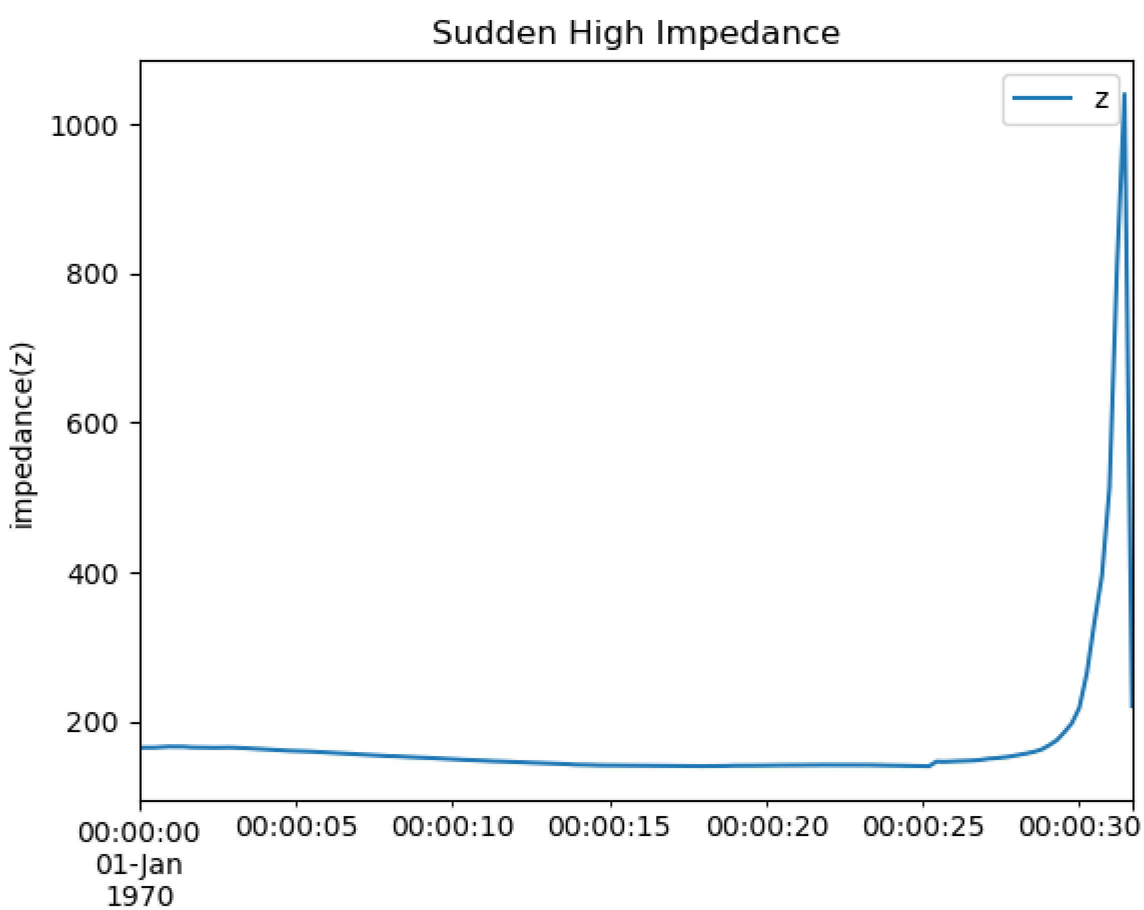
p value is too high, non-stationary.

In [12]:
```
# evaluate for stationarity
adfuller_test(df_sudden)
```

```
Test Statistic              0.111635
p-value                     0.966847
Lags Used                  12.000000
Observations Used         120.000000
Critical Vlaue (1%)        -3.486056
Critical Vlaue (5%)        -2.885943
Critical Vlaue (10%)       -2.579785
dtype: float64
```

Plot to visualize time series.

In [13]:
```
# plotting
df_sudden.plot.line();
plt.ylabel('impedance(z)');
plt.title('Sudden High Impedance');
```



Extract impedance and time from sensor log for Gradual High Impedance

In [14]:
```
# extract impedance data
df_gradual = extract_data(df_list[2][0])
df_gradual
```

Out[14]:

	z
1970-01-01 00:00:00.000	224.8
1970-01-01 00:00:00.240	223.8
1970-01-01 00:00:00.480	223.5
1970-01-01 00:00:00.720	224.1
1970-01-01 00:00:00.960	224.2
...	...
1970-01-01 00:02:02.160	881.2
1970-01-01 00:02:02.400	940.4
1970-01-01 00:02:02.640	957.6
1970-01-01 00:02:02.880	1004.3
1970-01-01 00:02:03.120	1071.9

514 rows × 1 columns

Evaluate for stationarity

p value is too high, non-stationary.

In [15]:

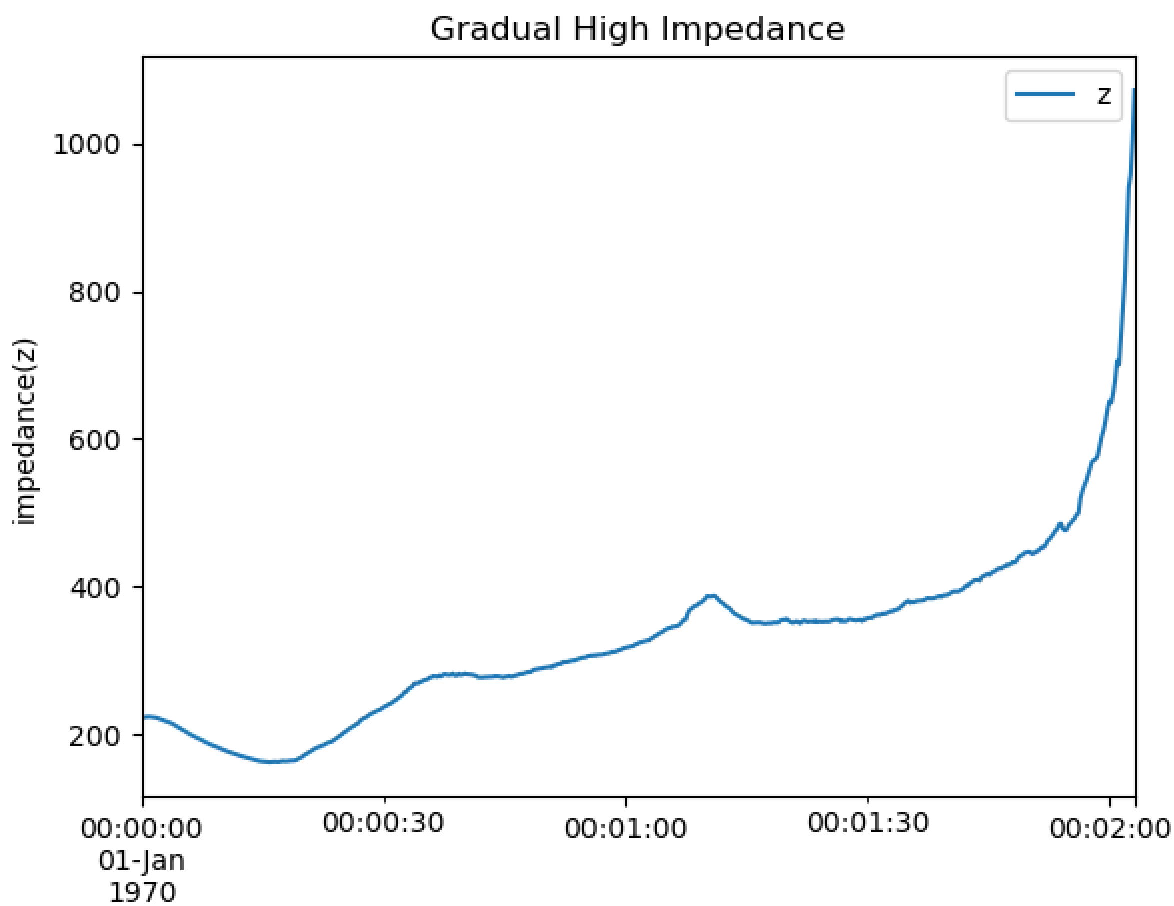
```
# evaluate for stationarity
adfuller_test(df_gradual)
```

```
Test Statistic              0.592361
p-value                     0.987426
Lags Used                  19.000000
Observations Used         494.000000
Critical Vlaue (1%)        -3.443657
Critical Vlaue (5%)        -2.867408
Critical Vlaue (10%)       -2.569896
dtype: float64
```

Plot to visualize time series.

In [16]:

```
# plotting
df_gradual.plot.line();
plt.ylabel('impedance(z)');
plt.title('Gradual High Impedance');
```



4. Time Series Modeling

Using Facebook Prophet to forecast if anomaly will occur.

1. No Smoothing
2. Moving Average Smoothing
3. Exponential Smoothing

In [17]:
```
# change data for Prophet
def change_format(df):
    # new_index = pd.date_range(df.index[0], periods=len(df), freq='1D')
    df['ds'] = df.index
    df = df.rename(columns={"z": "y"})
    df['y'] = np.log10(df['y'])

    return df.reset_index(drop=True)
```

4.1. Normal Sensor Measurement

4.1.1 No Smoothing

Green dots indicate no anomaly detected as measurements are less than the forecasted upper range enveloped in blue.

In [18]:
```
df_normal_nosmooth = change_format(df_normal)
```

```
df_normal_nosmooth
```

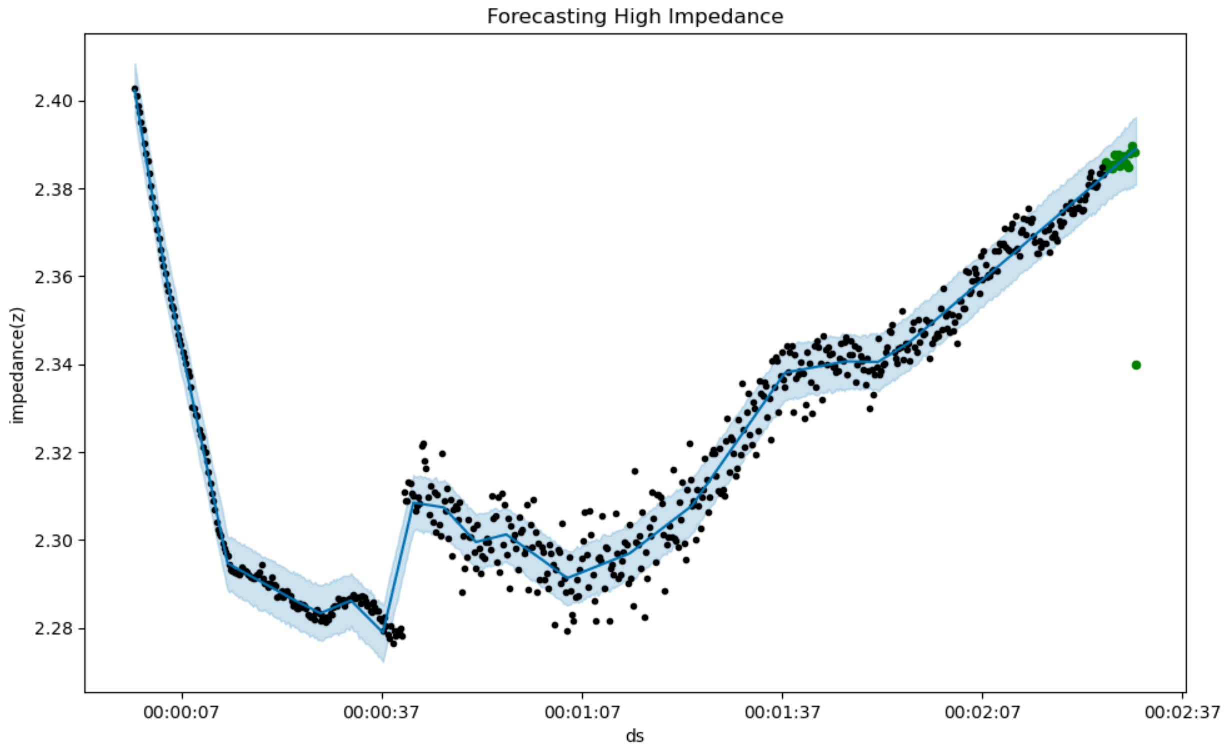
Out[18]:

	y	ds
0	2.402777	1970-01-01 00:00:00.000
1	2.401056	1970-01-01 00:00:00.240
2	2.398634	1970-01-01 00:00:00.480
3	2.397245	1970-01-01 00:00:00.720
4	2.394977	1970-01-01 00:00:00.960
...	...	...
621	2.387923	1970-01-01 00:02:29.040
622	2.388101	1970-01-01 00:02:29.280
623	2.389698	1970-01-01 00:02:29.520
624	2.388279	1970-01-01 00:02:29.760
625	2.339849	1970-01-01 00:02:30.000

626 rows × 2 columns

In [19]:

```
m = fbprophet.Prophet(growth='linear', yearly_seasonality=False, weekly_seasonality=Fa
m.fit(df_normal_nosmooth.iloc[:-20])
future = m.make_future_dataframe(periods=20, freq='240L')
forecast = m.predict(future)
m.plot(forecast);

ax = plt.gca();
df_normal_nosmooth['anomaly'] = 0
df_normal_nosmooth.loc[df_normal_nosmooth['y'] > forecast['yhat_upper'], 'anomaly'] =
# df_normal.iloc[-20:].plot.scatter(x='ds', y='anomaly', ax=ax, color='red');
df_normal_nosmooth_predict = df_normal_nosmooth.iloc[-20:]
df_normal_nosmooth_predict[df_normal_nosmooth_predict['anomaly'] == 0].plot.scatter(x=
df_normal_nosmooth_predict[df_normal_nosmooth_predict['anomaly'] == 1].plot.scatter(x=

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Forecasting High Impedance');
```



4.1.2 Moving Average Smoothing

Green dots indicate no anomaly detected as measurements are less than the forecasted upper range enveloped in blue.

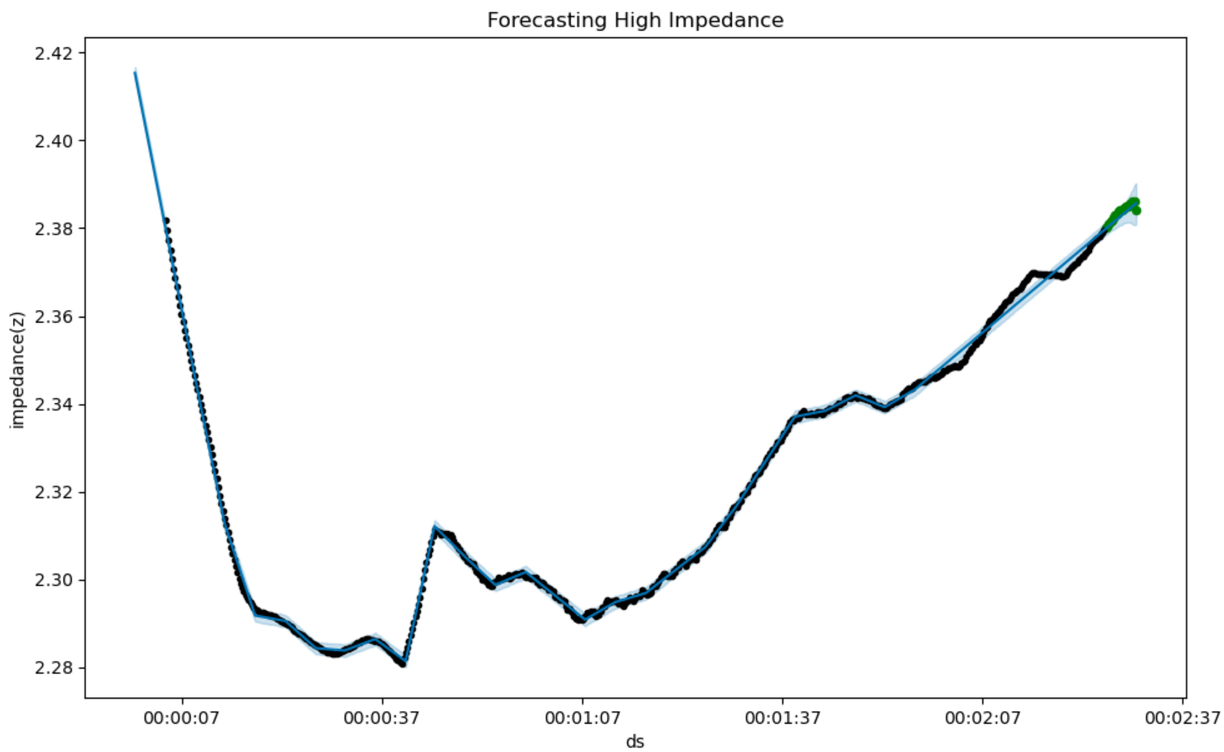
In [20]:
```
df_normal_ma = change_format(df_normal)
df_normal_ma['y'] = df_normal_ma['y'].rolling(20).mean()
# df_normal_ma = df_normal_ma.dropna()
df_normal_ma
```

Out[20]:

	y	ds
0	NaN	1970-01-01 00:00:00.000
1	NaN	1970-01-01 00:00:00.240
2	NaN	1970-01-01 00:00:00.480
3	NaN	1970-01-01 00:00:00.720
4	NaN	1970-01-01 00:00:00.960
...	...	...
621	2.385443	1970-01-01 00:02:29.040
622	2.385675	1970-01-01 00:02:29.280
623	2.385996	1970-01-01 00:02:29.520
624	2.386175	1970-01-01 00:02:29.760
625	2.384012	1970-01-01 00:02:30.000

626 rows × 2 columns

In [21]:
```
m = fbprophet.Prophet(growth='linear', yearly_seasonality=False, weekly_seasonality=Fa
m.fit(df_normal_ma.iloc[:-20])
future = m.make_future_dataframe(periods=20, freq='240L')
forecast = m.predict(future)
m.plot(forecast);

ax = plt.gca();
df_normal_ma['anomaly'] = 0
df_normal_ma.loc[df_normal_ma['y'] > forecast['yhat_upper'], 'anomaly'] = 1
# df_normal.iloc[-20:].plot.scatter(x='ds', y='anomaly', ax=ax, color='red');
df_normal_ma_predict = df_normal_ma.iloc[-20:]
df_normal_ma_predict[df_normal_ma_predict['anomaly'] == 0].plot.scatter(x='ds', y='y',
df_normal_ma_predict[df_normal_ma_predict['anomaly'] == 1].plot.scatter(x='ds', y='y',

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Forecasting High Impedance');
```



4.1.3 Exponential Smoothing

Green dots indicate no anomaly detected as measurements are less than the forecasted upper range enveloped in blue.

In [22]:
```
df_normal_ewm = change_format(df_normal)
df_normal_ewm['y'] = df_normal_ewm['y'].ewm(span=20).mean()
# df_normal_ma = df_normal_ma.dropna()
df_normal_ewm
```

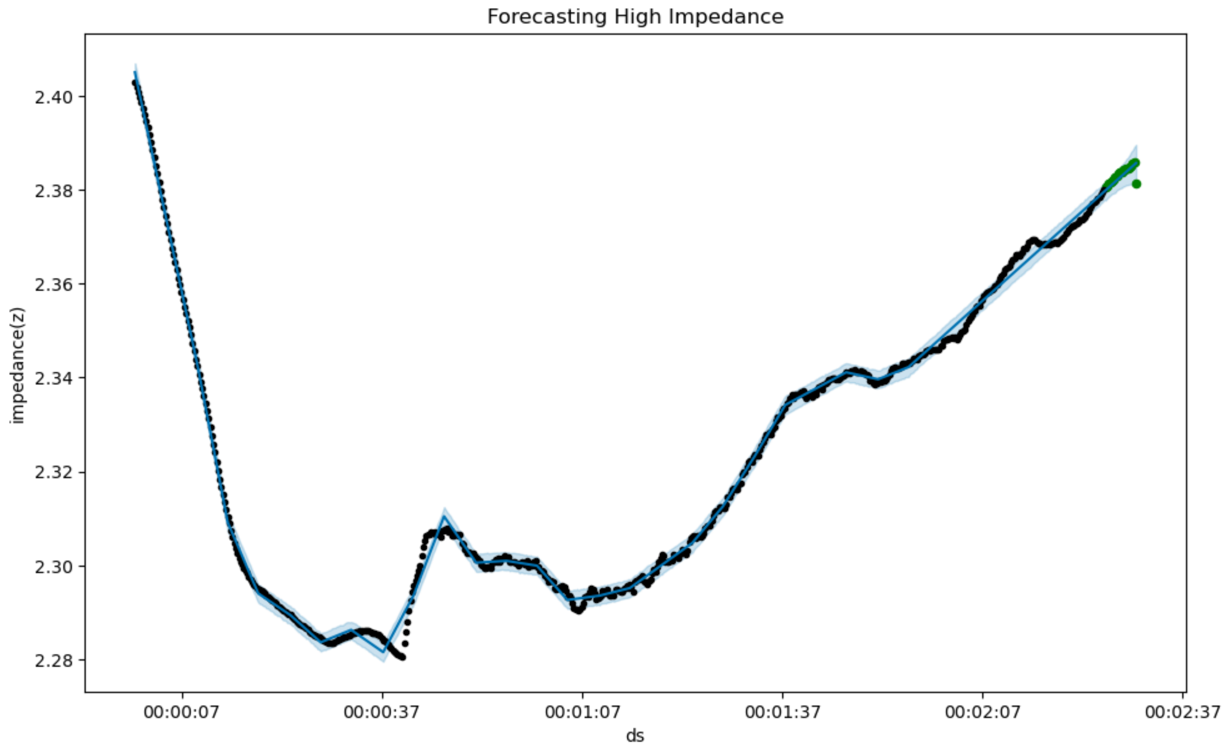
Out[22]:

	y	ds
0	2.402777	1970-01-01 00:00:00.000
1	2.401873	1970-01-01 00:00:00.240
2	2.400684	1970-01-01 00:00:00.480
3	2.399691	1970-01-01 00:00:00.720
4	2.398551	1970-01-01 00:00:00.960
...	...	...
621	2.384829	1970-01-01 00:02:29.040
622	2.385140	1970-01-01 00:02:29.280
623	2.385575	1970-01-01 00:02:29.520
624	2.385832	1970-01-01 00:02:29.760
625	2.381453	1970-01-01 00:02:30.000

626 rows × 2 columns

In [23]:

```
m = fbprophet.Prophet(growth='linear', yearly_seasonality=False, weekly_seasonality=Fa
m.fit(df_normal_ewm.iloc[:-20])
future = m.make_future_dataframe(periods=20, freq='240L')
forecast = m.predict(future)
m.plot(forecast);

ax = plt.gca();
df_normal_ewm['anomaly'] = 0
df_normal_ewm.loc[df_normal_ewm['y'] > forecast['yhat_upper'], 'anomaly'] = 1
# df_normal.iloc[-20:].plot.scatter(x='ds', y='anomaly', ax=ax, color='red');
df_normal_ewm_predict = df_normal_ewm.iloc[-20:]
df_normal_ewm_predict[df_normal_ewm_predict['anomaly'] == 0].plot.scatter(x='ds', y='y
df_normal_ewm_predict[df_normal_ewm_predict['anomaly'] == 1].plot.scatter(x='ds', y='y

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Forecasting High Impedance');
```



4.2. Sudden High Impedance Sensor Measurement

4.2.1 No Smoothing

Green dots indicate no anomaly detected as measurements are less than the forecasted upper range enveloped in blue. Red dots indicate anomaly detects as measurements are not greater than forcasted upper range.

In [24]:
```
df_sudden_nosmooth = change_format(df_sudden)
df_sudden_nosmooth
```

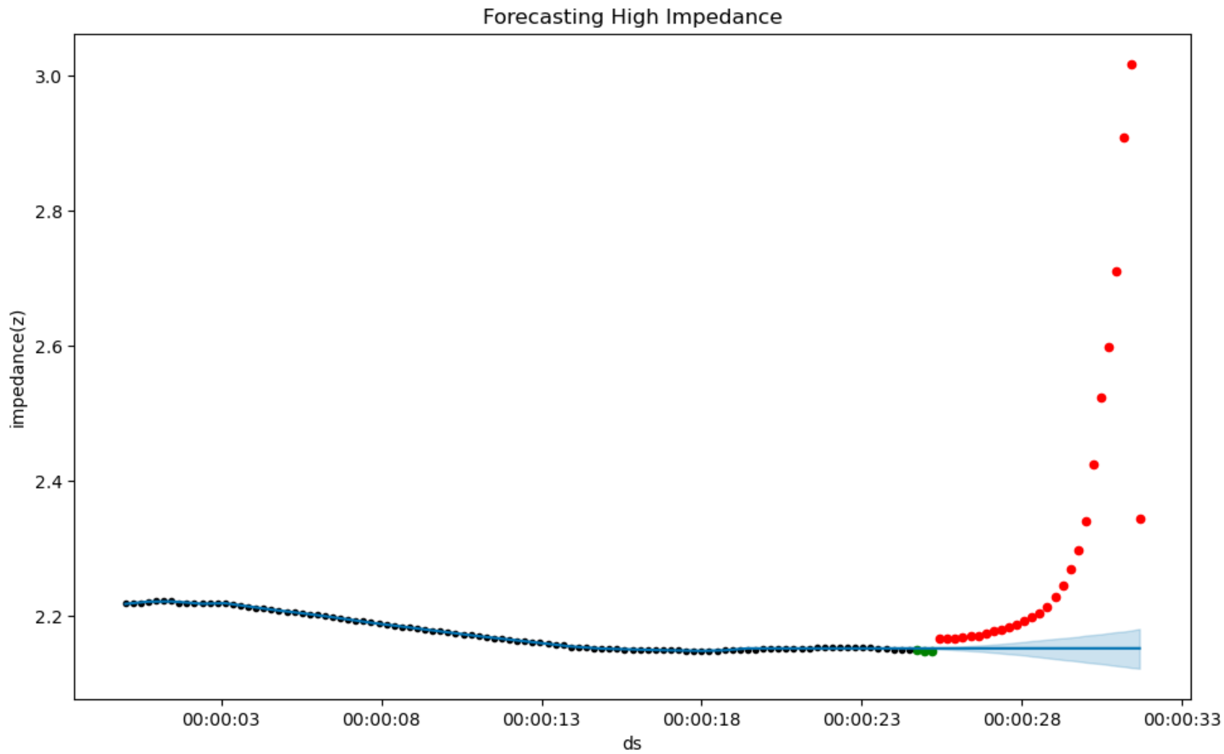
Out[24]:

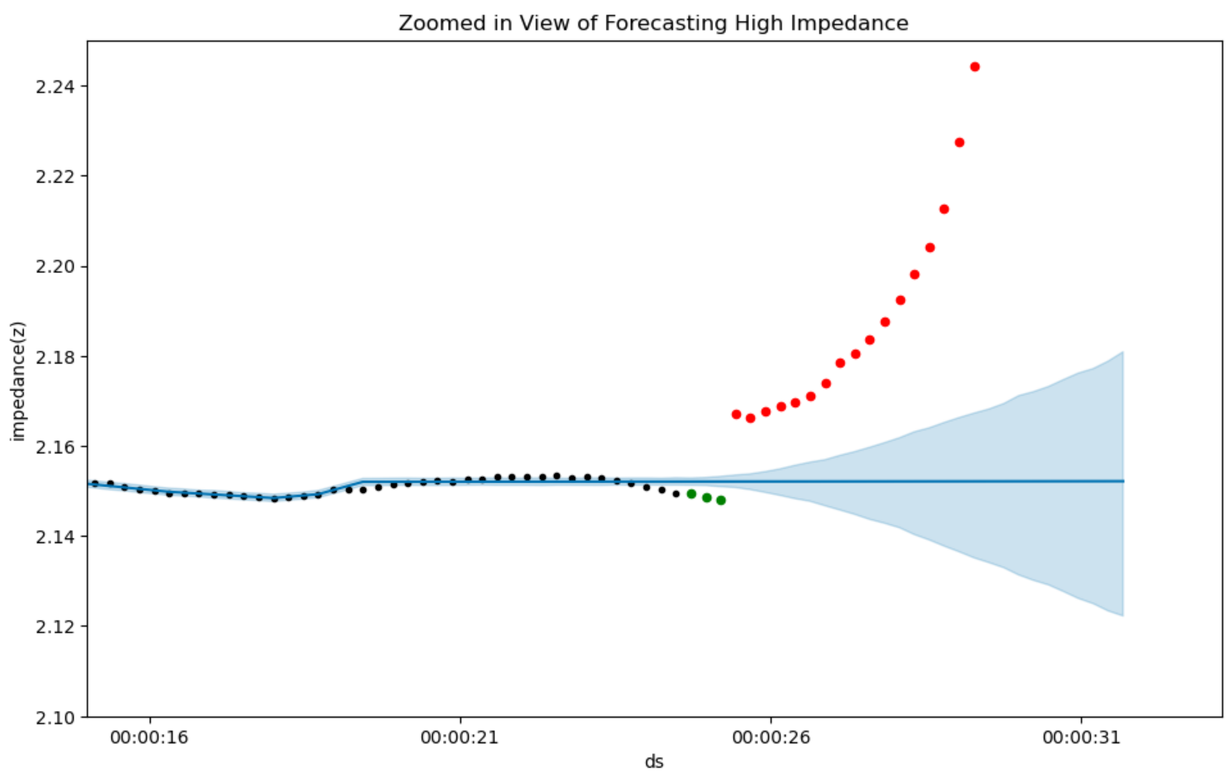
	y	ds
0	2.218273	1970-01-01 00:00:00.000
1	2.219060	1970-01-01 00:00:00.240
2	2.219060	1970-01-01 00:00:00.480
3	2.220892	1970-01-01 00:00:00.720
4	2.222456	1970-01-01 00:00:00.960
...	...	...
128	2.597586	1970-01-01 00:00:30.720
129	2.710625	1970-01-01 00:00:30.960
130	2.907949	1970-01-01 00:00:31.200
131	3.017075	1970-01-01 00:00:31.440
132	2.344589	1970-01-01 00:00:31.680

133 rows × 2 columns

In [25]:

```
m = fbprophet.Prophet(growth='linear', yearly_seasonality=False, weekly_seasonality=Fa
m.fit(df_sudden_nosmooth.iloc[:-30])
future = m.make_future_dataframe(periods=30, freq='240L')
forecast = m.predict(future)
m.plot(forecast);

ax = plt.gca();
df_sudden_nosmooth['anomaly'] = 0
df_sudden_nosmooth.loc[df_sudden_nosmooth['y'] > forecast['yhat_upper'], 'anomaly'] =
# df_normal.iloc[-20:].plot.scatter(x='ds', y='anomaly', ax=ax, color='red');
df_sudden_nosmooth_predict = df_sudden_nosmooth.iloc[-30:]
df_sudden_nosmooth_predict[df_sudden_nosmooth_predict['anomaly'] == 0].plot.scatter(x=
df_sudden_nosmooth_predict[df_sudden_nosmooth_predict['anomaly'] == 1].plot.scatter(x=

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Forecasting High Impedance');

m.plot(forecast);
ax = plt.gca();
df_sudden_nosmooth_predict[df_sudden_nosmooth_predict['anomaly'] == 0].plot.scatter(x=
df_sudden_nosmooth_predict[df_sudden_nosmooth_predict['anomaly'] == 1].plot.scatter(x=
ax.set_xlim(left=pd.Timestamp("1970-01-01 00:00:15"))

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Zoomed in View of Forecasting High Impedance');
```






```
In [ ]:
```

4.2.2 Moving Average Smoothing

Green dots indicate no anomaly detected as measurements are less than the forecasted upper range enveloped in blue. Red dots indicate anomaly detects as measurements are not greater than forcasted upper range.

```
df_sudden_ma = change_format(df_sudden)
```

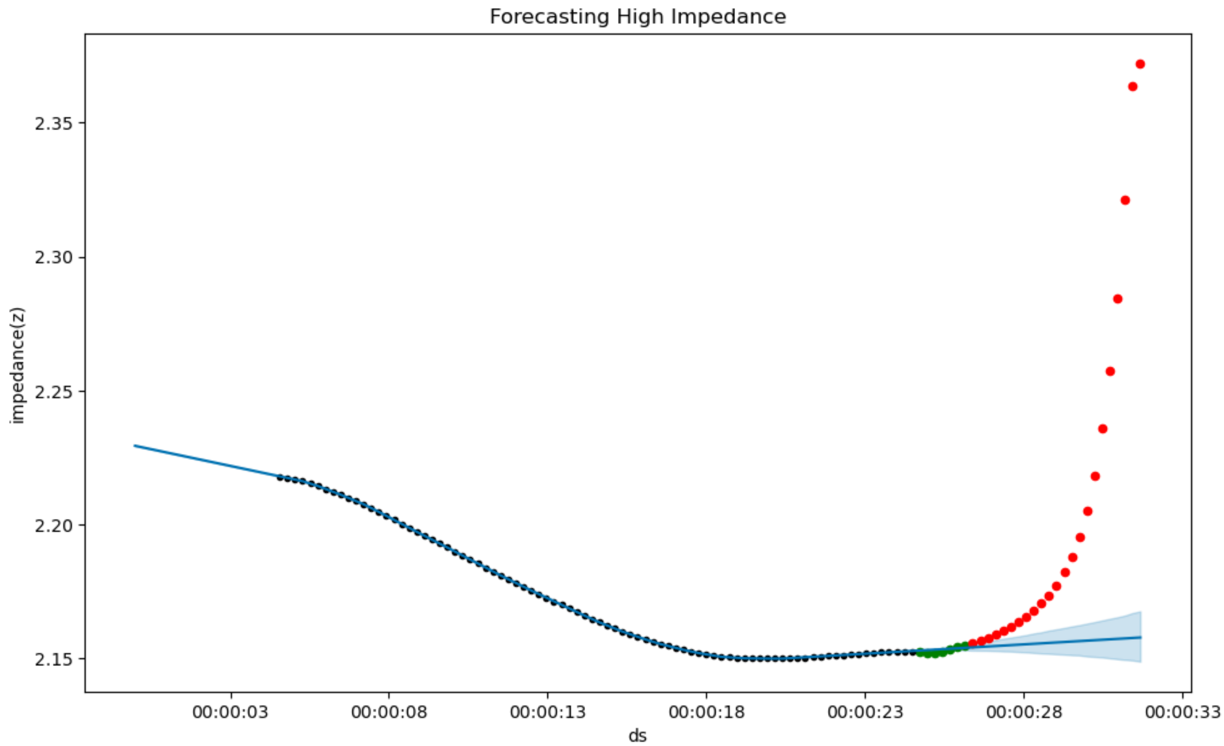
In [26]:
```
df_sudden_ma['y'] = df_sudden_ma['y'].rolling(20).mean()
# df_normal_ma = df_normal_ma.dropna()
df_sudden_ma
```

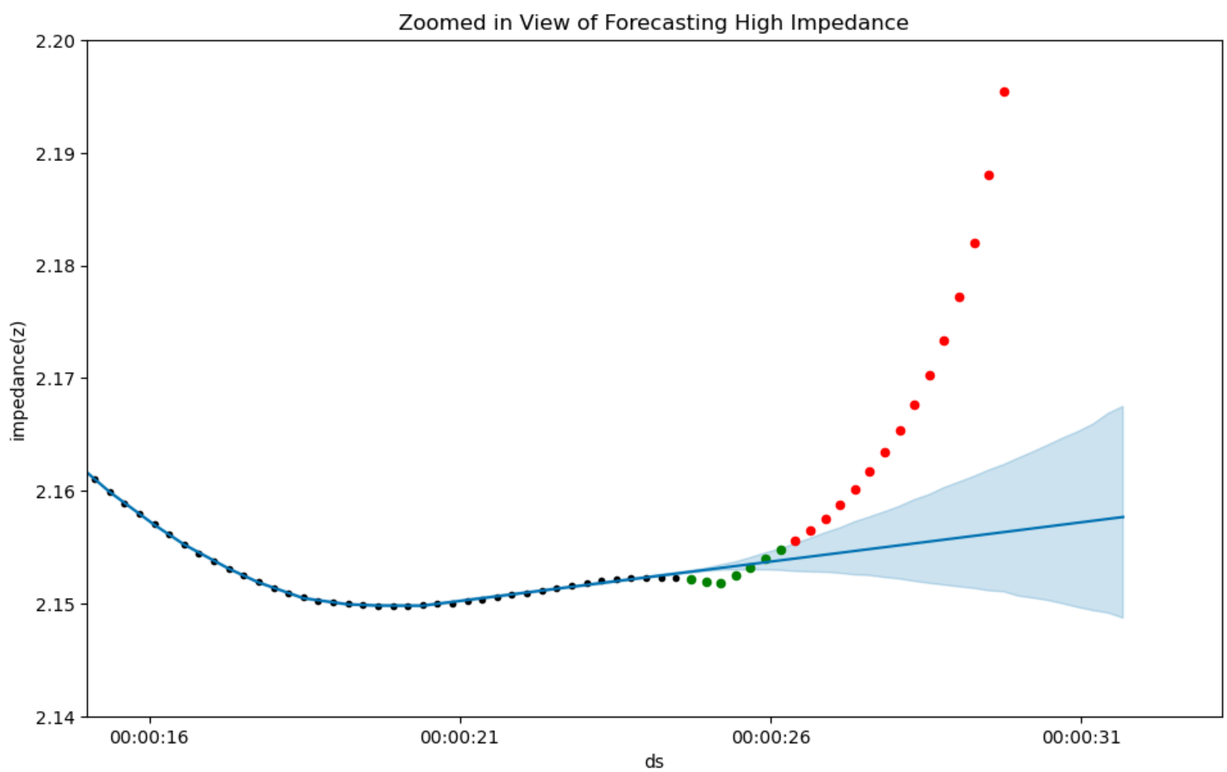
Out[26]:

	y	ds
0	NaN	1970-01-01 00:00:00.000
1	NaN	1970-01-01 00:00:00.240
2	NaN	1970-01-01 00:00:00.480
3	NaN	1970-01-01 00:00:00.720
4	NaN	1970-01-01 00:00:00.960
...	...	...
128	2.257307	1970-01-01 00:00:30.720
129	2.284399	1970-01-01 00:00:30.960
130	2.321312	1970-01-01 00:00:31.200
131	2.363609	1970-01-01 00:00:31.440
132	2.372135	1970-01-01 00:00:31.680

133 rows × 2 columns

In [27]:
```
m = fbprophet.Prophet(growth='linear', yearly_seasonality=False, weekly_seasonality=Fa
m.fit(df_sudden_ma.iloc[:-30])
future = m.make_future_dataframe(periods=30, freq='240L')
forecast = m.predict(future)
m.plot(forecast);

ax = plt.gca();
df_sudden_ma['anomaly'] = 0
df_sudden_ma.loc[df_sudden_ma['y'] > forecast['yhat_upper'], 'anomaly'] = 1
# df_normal.iloc[-20:].plot.scatter(x='ds', y='anomaly', ax=ax, color='red');
df_sudden_ma_predict = df_sudden_ma.iloc[-30:]
df_sudden_ma_predict[df_sudden_ma_predict['anomaly'] == 0].plot.scatter(x='ds', y='y',
df_sudden_ma_predict[df_sudden_ma_predict['anomaly'] == 1].plot.scatter(x='ds', y='y',

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Forecasting High Impedance');

m.plot(forecast);
ax = plt.gca();
df_sudden_ma_predict[df_sudden_ma_predict['anomaly'] == 0].plot.scatter(x='ds', y='y',
df_sudden_ma_predict[df_sudden_ma_predict['anomaly'] == 1].plot.scatter(x='ds', y='y',
ax.set_xlim(left=pd.Timestamp("1970-01-01 00:00:15"))

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Zoomed in View of Forecasting High Impedance');
```





4.2.3 Exponential Smoothing

Green dots indicate no anomaly detected as measurements are less than the forecasted upper range enveloped in blue. Red dots indicate anomaly detects as measurements are not greater than forcasted upper range.

In [28]:
```
df_sudden_ewm = change_format(df_sudden)
df_sudden_ewm['y'] = df_sudden_ewm['y'].ewm(span=30).mean()
```

```
# df_normal_ma = df_normal_ma.dropna()
df_sudden_ewm
```

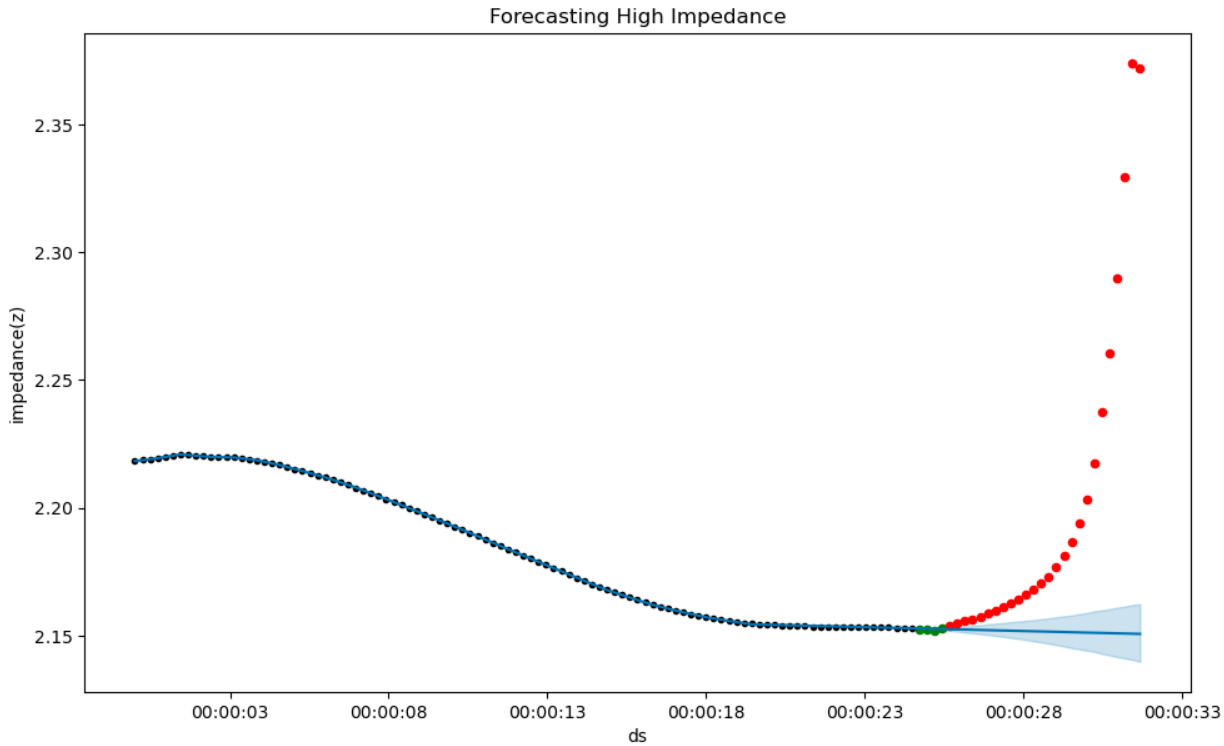
Out[28]:

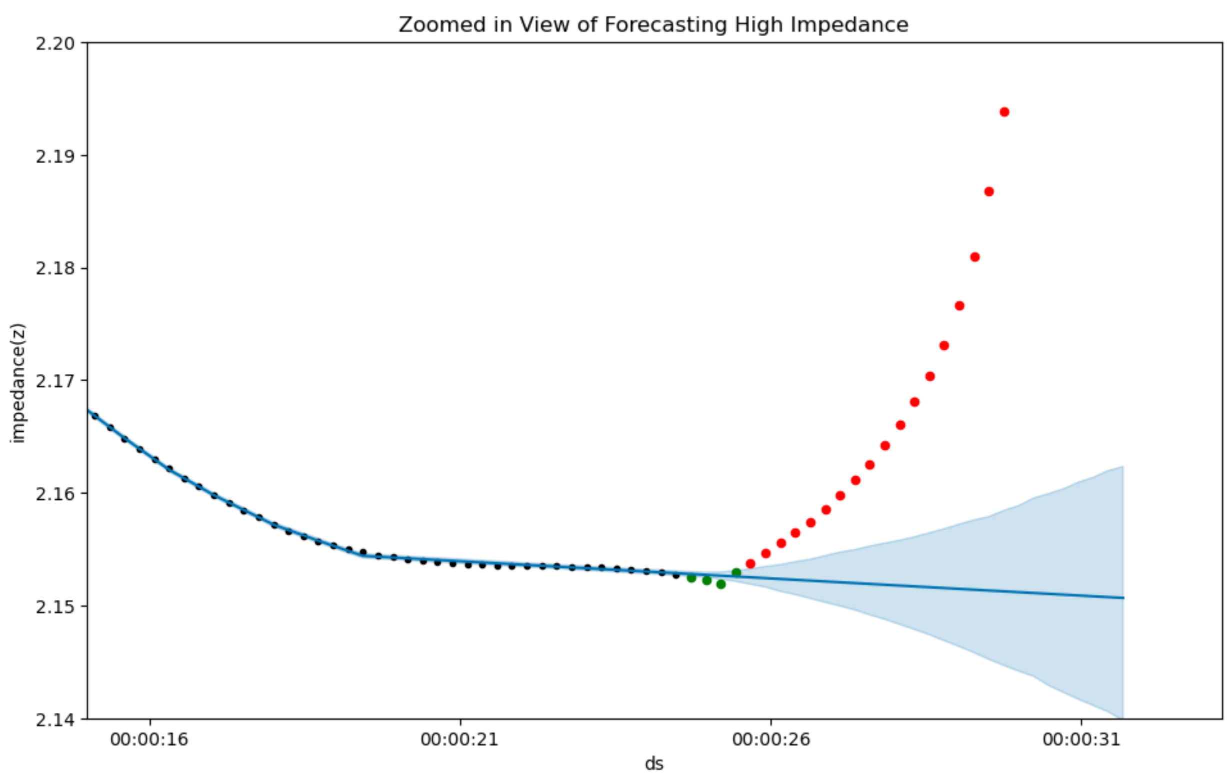
	y	ds
0	2.218273	1970-01-01 00:00:00.000
1	2.218680	1970-01-01 00:00:00.240
2	2.218815	1970-01-01 00:00:00.480
3	2.219387	1970-01-01 00:00:00.720
4	2.220086	1970-01-01 00:00:00.960
...	...	...
128	2.260597	1970-01-01 00:00:30.720
129	2.289636	1970-01-01 00:00:30.960
130	2.329534	1970-01-01 00:00:31.200
131	2.373898	1970-01-01 00:00:31.440
132	2.372007	1970-01-01 00:00:31.680

133 rows × 2 columns

In [29]:

```
m = fbprophet.Prophet(growth='linear', yearly_seasonality=False, weekly_seasonality=Fa
m.fit(df_sudden_ewm.iloc[:-30])
future = m.make_future_dataframe(periods=30, freq='240L')
forecast = m.predict(future)
m.plot(forecast);

ax = plt.gca();
df_sudden_ewm['anomaly'] = 0
df_sudden_ewm.loc[df_sudden_ewm['y'] > forecast['yhat_upper'], 'anomaly'] = 1
# df_normal.iloc[-20:].plot.scatter(x='ds', y='anomaly', ax=ax, color='red');
df_sudden_ewm_predict = df_sudden_ewm.iloc[-30:]
df_sudden_ewm_predict[df_sudden_ewm_predict['anomaly'] == 0].plot.scatter(x='ds', y='y
df_sudden_ewm_predict[df_sudden_ewm_predict['anomaly'] == 1].plot.scatter(x='ds', y='y

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Forecasting High Impedance');

m.plot(forecast);
ax = plt.gca();
df_sudden_ewm_predict[df_sudden_ewm_predict['anomaly'] == 0].plot.scatter(x='ds', y='y
df_sudden_ewm_predict[df_sudden_ewm_predict['anomaly'] == 1].plot.scatter(x='ds', y='y
ax.set_xlim(left=pd.Timestamp("1970-01-01 00:00:15"))

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Zoomed in View of Forecasting High Impedance');
```





4.3. Gradual High Impedance Sensor Measurement

4.3.1 No Smoothing

Green dots indicate no anomaly detected as measurements are less than the forecasted upper range enveloped in blue. Red dots indicate anomaly detects as measurements are not greater than forcasted upper range.

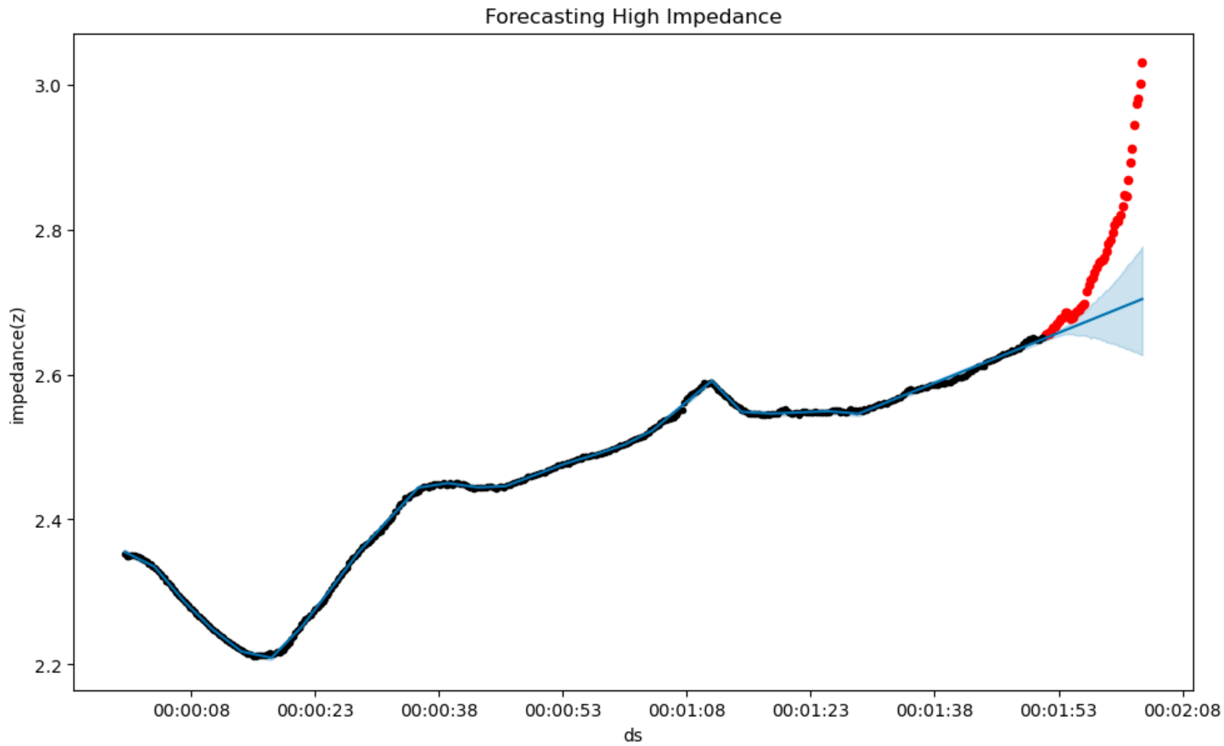
In [30]:
```
df_gradual_nosmooth = change_format(df_gradual)
df_gradual_nosmooth
```

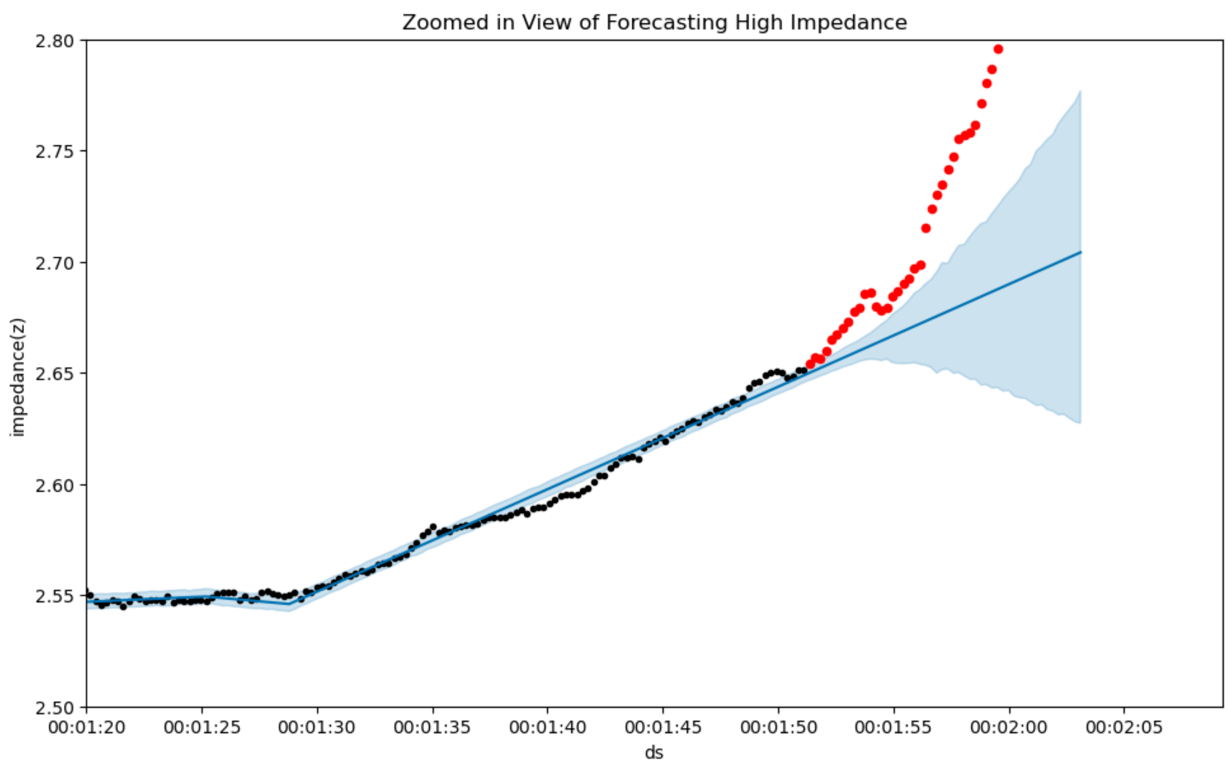
Out[30]:

	y	ds
0	2.351796	1970-01-01 00:00:00.000
1	2.349860	1970-01-01 00:00:00.240
2	2.349278	1970-01-01 00:00:00.480
3	2.350442	1970-01-01 00:00:00.720
4	2.350636	1970-01-01 00:00:00.960
...	...	...
509	2.945074	1970-01-01 00:02:02.160
510	2.973313	1970-01-01 00:02:02.400
511	2.981184	1970-01-01 00:02:02.640
512	3.001863	1970-01-01 00:02:02.880
513	3.030154	1970-01-01 00:02:03.120

514 rows × 2 columns

In [31]:
```
m = fbprophet.Prophet(growth='linear', yearly_seasonality=False, weekly_seasonality=Fa
m.fit(df_gradual_nosmooth.iloc[:-50])
future = m.make_future_dataframe(periods=50, freq='240L')
forecast = m.predict(future)
m.plot(forecast);

ax = plt.gca();
df_gradual_nosmooth['anomaly'] = 0
df_gradual_nosmooth.loc[df_gradual_nosmooth['y'] > forecast['yhat_upper'], 'anomaly']
# df_normal.iloc[-20:].plot.scatter(x='ds', y='anomaly', ax=ax, color='red');
df_gradual_nosmooth_predict = df_gradual_nosmooth.iloc[-50:]
df_gradual_nosmooth_predict[df_gradual_nosmooth_predict['anomaly'] == 0].plot.scatter(
df_gradual_nosmooth_predict[df_gradual_nosmooth_predict['anomaly'] == 1].plot.scatter(

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Forecasting High Impedance');

m.plot(forecast);
ax = plt.gca();
df_gradual_nosmooth_predict[df_gradual_nosmooth_predict['anomaly'] == 0].plot.scatter(
df_gradual_nosmooth_predict[df_gradual_nosmooth_predict['anomaly'] == 1].plot.scatter(
ax.set_xlim(left=pd.Timestamp("1970-01-01 00:01:20"))

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Zoomed in View of Forecasting High Impedance');
```





4.3.2 Moving Average Smoothing

Green dots indicate no anomaly detected as measurements are less than the forecasted upper range enveloped in blue. Red dots indicate anomaly detects as measurements are not greater than forcasted upper range.

In [32]:

```
df_gradual_ma = change_format(df_gradual)
df_gradual_ma['y'] = df_gradual_ma['y'].rolling(30).mean()
# df_normal_ma = df_normal_ma.dropna()
df_gradual_ma
```

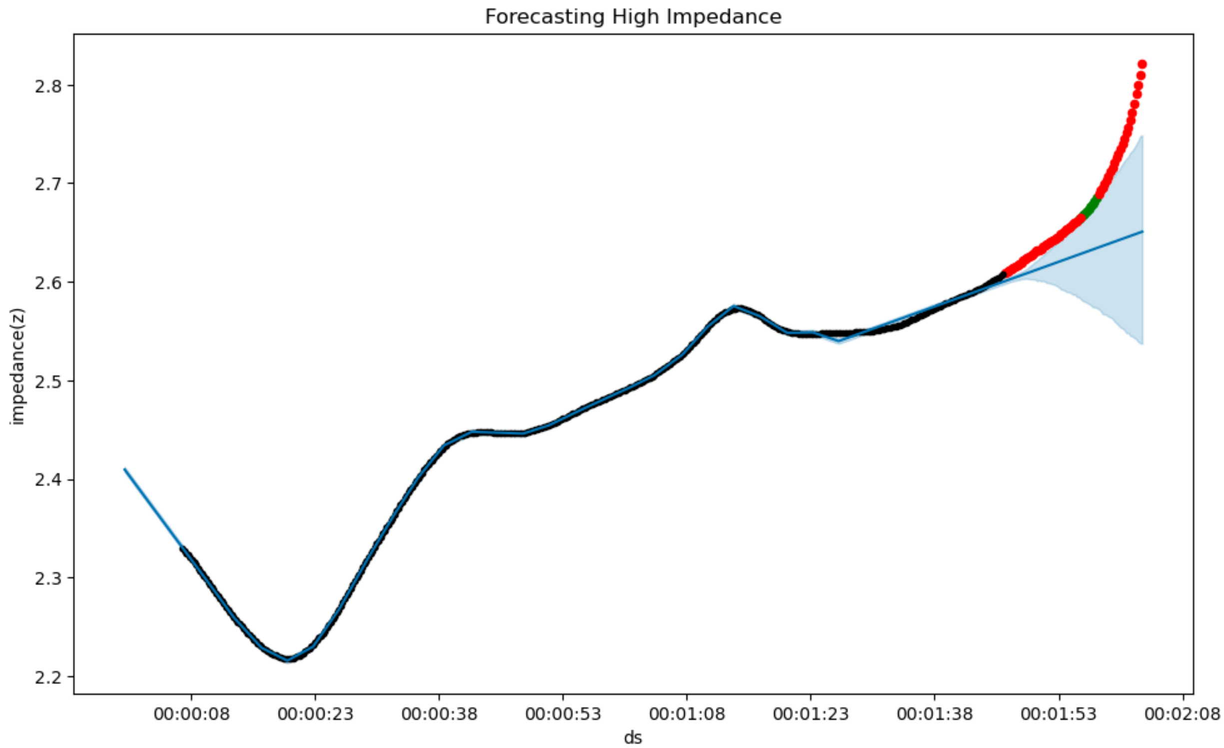
Out[32]:

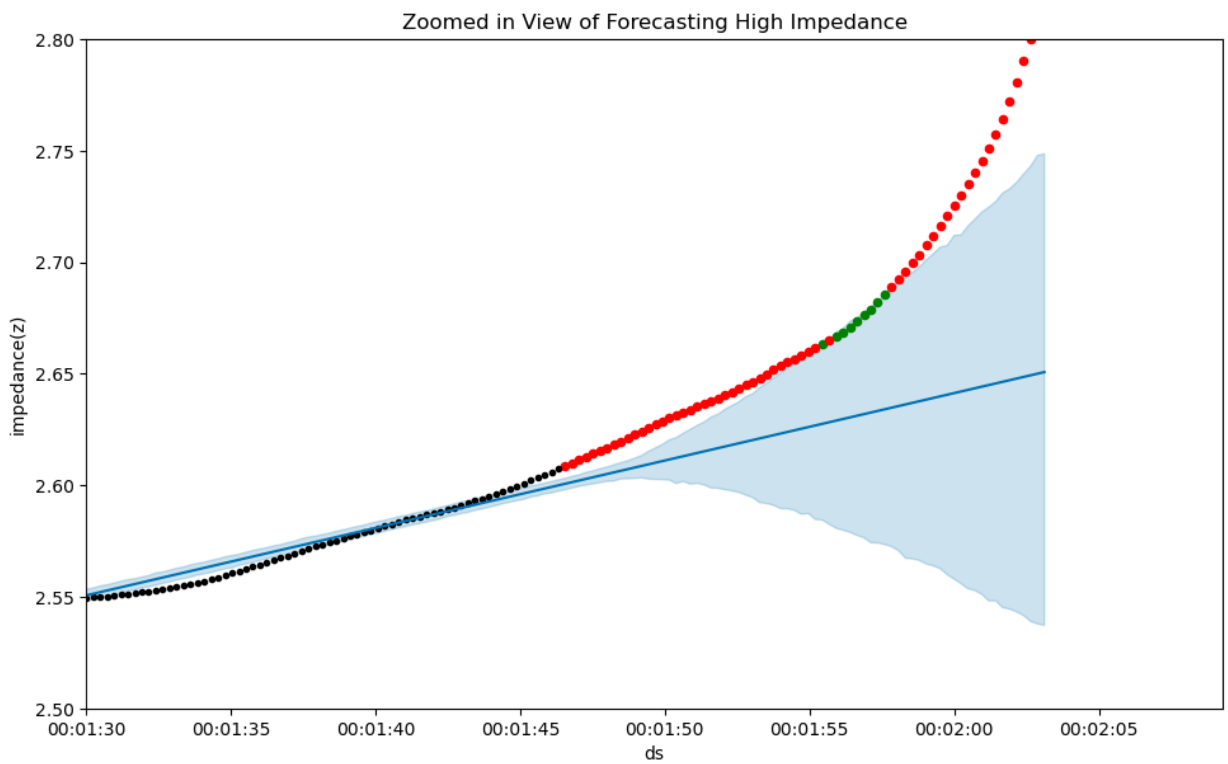
	y	ds
0	NaN	1970-01-01 00:00:00.000
1	NaN	1970-01-01 00:00:00.240
2	NaN	1970-01-01 00:00:00.480
3	NaN	1970-01-01 00:00:00.720
4	NaN	1970-01-01 00:00:00.960
...	...	...
509	2.780652	1970-01-01 00:02:02.160
510	2.790199	1970-01-01 00:02:02.400
511	2.799910	1970-01-01 00:02:02.640
512	2.810232	1970-01-01 00:02:02.880
513	2.821346	1970-01-01 00:02:03.120

514 rows × 2 columns

In [33]:

```
m = fbprophet.Prophet(growth='linear', yearly_seasonality=False, weekly_seasonality=Fa
m.fit(df_gradual_ma.iloc[:-70])
future = m.make_future_dataframe(periods=70, freq='240L')
forecast = m.predict(future)
m.plot(forecast);

ax = plt.gca();
df_gradual_ma['anomaly'] = 0
df_gradual_ma.loc[df_gradual_ma['y'] > forecast['yhat_upper'], 'anomaly'] = 1
# df_normal.iloc[-20:].plot.scatter(x='ds', y='anomaly', ax=ax, color='red');
df_gradual_ma_predict = df_gradual_ma.iloc[-70:]
df_gradual_ma_predict[df_gradual_ma_predict['anomaly'] == 0].plot.scatter(x='ds', y='y
df_gradual_ma_predict[df_gradual_ma_predict['anomaly'] == 1].plot.scatter(x='ds', y='y

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Forecasting High Impedance');

m.plot(forecast);
ax = plt.gca();
df_gradual_ma_predict[df_gradual_ma_predict['anomaly'] == 0].plot.scatter(x='ds', y='y
df_gradual_ma_predict[df_gradual_ma_predict['anomaly'] == 1].plot.scatter(x='ds', y='y
ax.set_xlim(left=pd.Timestamp("1970-01-01 00:01:30"))

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Zoomed in View of Forecasting High Impedance');
```





4.3.3 Exponential Smoothing

Green dots indicate no anomaly detected as measurements are less than the forecasted upper range enveloped in blue. Red dots indicate anomaly detects as measurements are not greater than forcasted upper range.

In [34]:
```
df_gradual_ewm = change_format(df_gradual)
df_gradual_ewm['y'] = df_gradual_ewm['y'].ewm(span=30).mean()
```

```
# df_normal_ma = df_normal_ma.dropna()
df_gradual_ewm
```

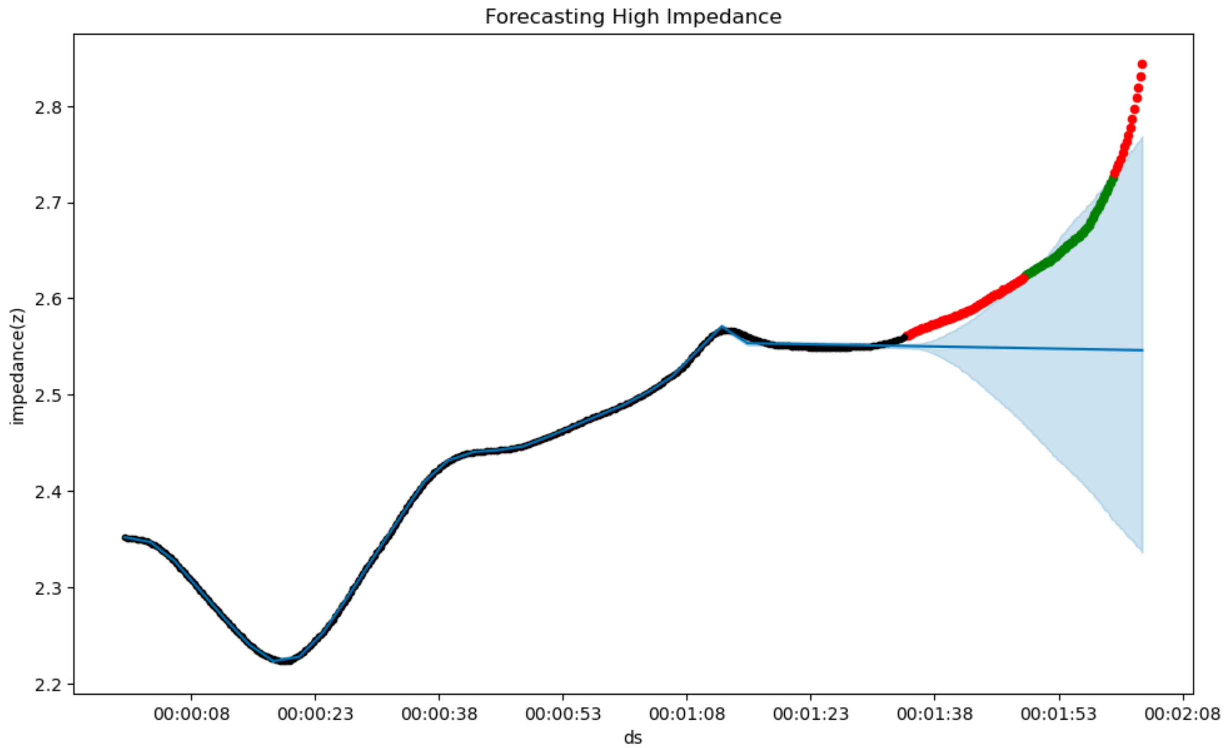
Out[34]:

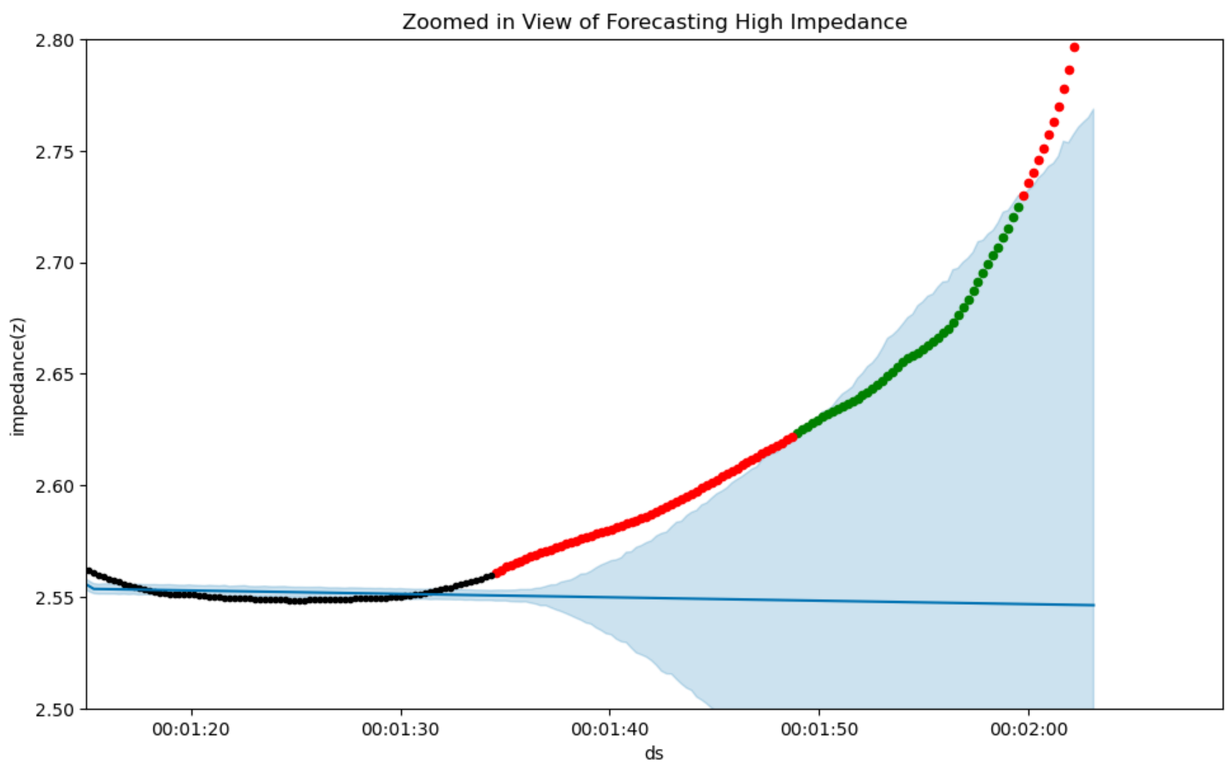
	y	ds
0	2.351796	1970-01-01 00:00:00.000
1	2.350796	1970-01-01 00:00:00.240
2	2.350256	1970-01-01 00:00:00.480
3	2.350307	1970-01-01 00:00:00.720
4	2.350382	1970-01-01 00:00:00.960
...	...	...
509	2.796749	1970-01-01 00:02:02.160
510	2.808140	1970-01-01 00:02:02.400
511	2.819304	1970-01-01 00:02:02.640
512	2.831082	1970-01-01 00:02:02.880
513	2.843926	1970-01-01 00:02:03.120

514 rows × 2 columns

In [35]:

```
m = fbprophet.Prophet(growth='linear', yearly_seasonality=False, weekly_seasonality=Fa
m.fit(df_gradual_ewm.iloc[:-120])
future = m.make_future_dataframe(periods=120, freq='240L')
forecast = m.predict(future)
m.plot(forecast);

ax = plt.gca();
df_gradual_ewm['anomaly'] = 0
df_gradual_ewm.loc[df_gradual_ewm['y'] > forecast['yhat_upper'], 'anomaly'] = 1
# df_normal.iloc[-20:].plot.scatter(x='ds', y='anomaly', ax=ax, color='red');
df_gradual_ewm_predict = df_gradual_ewm.iloc[-120:]
df_gradual_ewm_predict[df_gradual_ewm_predict['anomaly'] == 0].plot.scatter(x='ds', y=
df_gradual_ewm_predict[df_gradual_ewm_predict['anomaly'] == 1].plot.scatter(x='ds', y=

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Forecasting High Impedance');

m.plot(forecast);
ax = plt.gca();
df_gradual_ewm_predict[df_gradual_ewm_predict['anomaly'] == 0].plot.scatter(x='ds', y=
df_gradual_ewm_predict[df_gradual_ewm_predict['anomaly'] == 1].plot.scatter(x='ds', y=
ax.set_xlim(left=pd.Timestamp("1970-01-01 00:01:15"))

plt.ylabel('time');
plt.ylabel('impedance(z)');
plt.title('Zoomed in View of Forecasting High Impedance');
```





5. Recommended Model

The addition of smoothing helps to eliminate potential jumps in the raw impedence data. This is prevalent in biological tissue and when measurements are acquired at a fast pace 240 ms. The smoothing of the time series with either moving average or exponential smoothing also perform equally well. In conclusion, moving average is recommended as its a faster computation than exponential and Prophet performed well in forecasting.

6. Key Findings

Analysis of the 3 time series showed non-stationarity for all 3. From the plots, multiplicative trends are present for all 3 with no seasonality.

For the 2 extremes in high impedances behaviors, gradual and sudden high impedance, anomalies are successfully predicted using Facebook Prophet with either moving average or exponential smoothing. In the normal scenario where no high impedance occurs, no anomalies are predicted. These results indicate good performance in forecasting whether high impedance anomalies may occur and when interventions can be implement to avoid them. Sensitivity to the anomaly criteria can be modified by adding scaling to the yhat_upper threshold limit.

7. Next Steps

Tuning of parameters used including

- size of moving window or span.
- how far advance to forecast.
- much data is needed to make good forecast i.e. more dependent on more recent data than older ones.
- adding scaling to yhat_upper to add more buffer in determining anomaly.

Also more logs are needed to understand the variability in the data, particularly since its a measurement of human tissue behavior.

```
In [ ]:
```