Multiple linear regression for BME680 gas readings of a single sensor

January 15, 2021

Read the CSV file created by the CCU Historian, ensure that field separator is ';' and decimal separator is ',' (German notation), if necessary edit CSV file before reading it by the provided script 'csv_convert_historian.bsh' The provided script 'get_new_history.bsh' is searching for the CCU Historian's CSV in the directory '${HOME}/Downloads'. The conversion script 'csv_convert_historian.bsh' is invoked inside 'csv_convert_historian.bsh'.

[1]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import pandas as pd
from datetime import datetime

import numpy as np


dateparse = lambda x: pd.datetime.strptime(x, '%d.%m.%Y %H:%M:%S,%f')

df0 = pd.read_csv("historian.csv", sep=';', thousands=".", decimal=",",␣
↪skiprows = [0,1,2],dtype={'High': np.float64, 'Low': np.float64}, header =␣
↪None, encoding= 'unicode_escape',  parse_dates=[0], date_parser=dateparse,␣
↪names = [ 'Datum', 'Mode', 'raw_gas_resistance', 'relative_humidity',␣
↪'temperature'])

df0.head(19)
```

[1]:

	Datum	Mode	raw_gas_resistance	relative_humidity	\
0	2020-12-29 22:13:34.000	2	99100	37.3	
1	2020-12-29 22:15:43.830	2	99100	37.3	
2	2020-12-29 22:15:43.840	2	99100	37.3	
3	2020-12-29 22:15:43.845	2	99920	37.3	
4	2020-12-29 22:20:15.627	2	99920	37.3	
5	2020-12-29 22:20:15.637	2	99920	37.2	
6	2020-12-29 22:20:15.643	2	100260	37.2	
7	2020-12-29 22:24:47.322	2	100260	37.2	
8	2020-12-29 22:24:47.326	2	100260	37.2	
9	2020-12-29 22:24:47.332	2	101220	37.2	

```
10 2020-12-29 22:29:19.016     2              101220               37.2
11 2020-12-29 22:29:19.019     2              101220               37.2
12 2020-12-29 22:29:19.025     2              101460               37.2
13 2020-12-29 22:33:50.707     2              101460               37.2
14 2020-12-29 22:33:50.716     2              101460               37.3
15 2020-12-29 22:33:50.722     2               98800               37.3
16 2020-12-29 22:38:22.398     2               98800               37.3
17 2020-12-29 22:38:22.408     2               98800               37.9
18 2020-12-29 22:38:22.414     2               94600               37.9

    temperature
0          23.4
1          23.4
2          23.4
3          23.4
4          23.4
5          23.4
6          23.4
7          23.4
8          23.4
9          23.4
10         23.4
11         23.4
12         23.4
13         23.4
14         23.4
15         23.4
16         23.4
17         23.4
18         23.4
```

[2]:
```
# keep every 3rd row (CCU historian is tacking every change of a datapoint␣
 ↪separately)
df = df0[(df0.index % 3 == 0)]

df.head(7)
```

[2]:
```
                    Datum  Mode  raw_gas_resistance  relative_humidity  \
0  2020-12-29 22:13:34.000     2               99100               37.3
3  2020-12-29 22:15:43.845     2               99920               37.3
6  2020-12-29 22:20:15.643     2              100260               37.2
9  2020-12-29 22:24:47.332     2              101220               37.2
12 2020-12-29 22:29:19.025     2              101460               37.2
15 2020-12-29 22:33:50.722     2               98800               37.3
18 2020-12-29 22:38:22.414     2               94600               37.9

    temperature
```

```
0          23.4
3          23.4
6          23.4
9          23.4
12         23.4
15         23.4
18         23.4
```

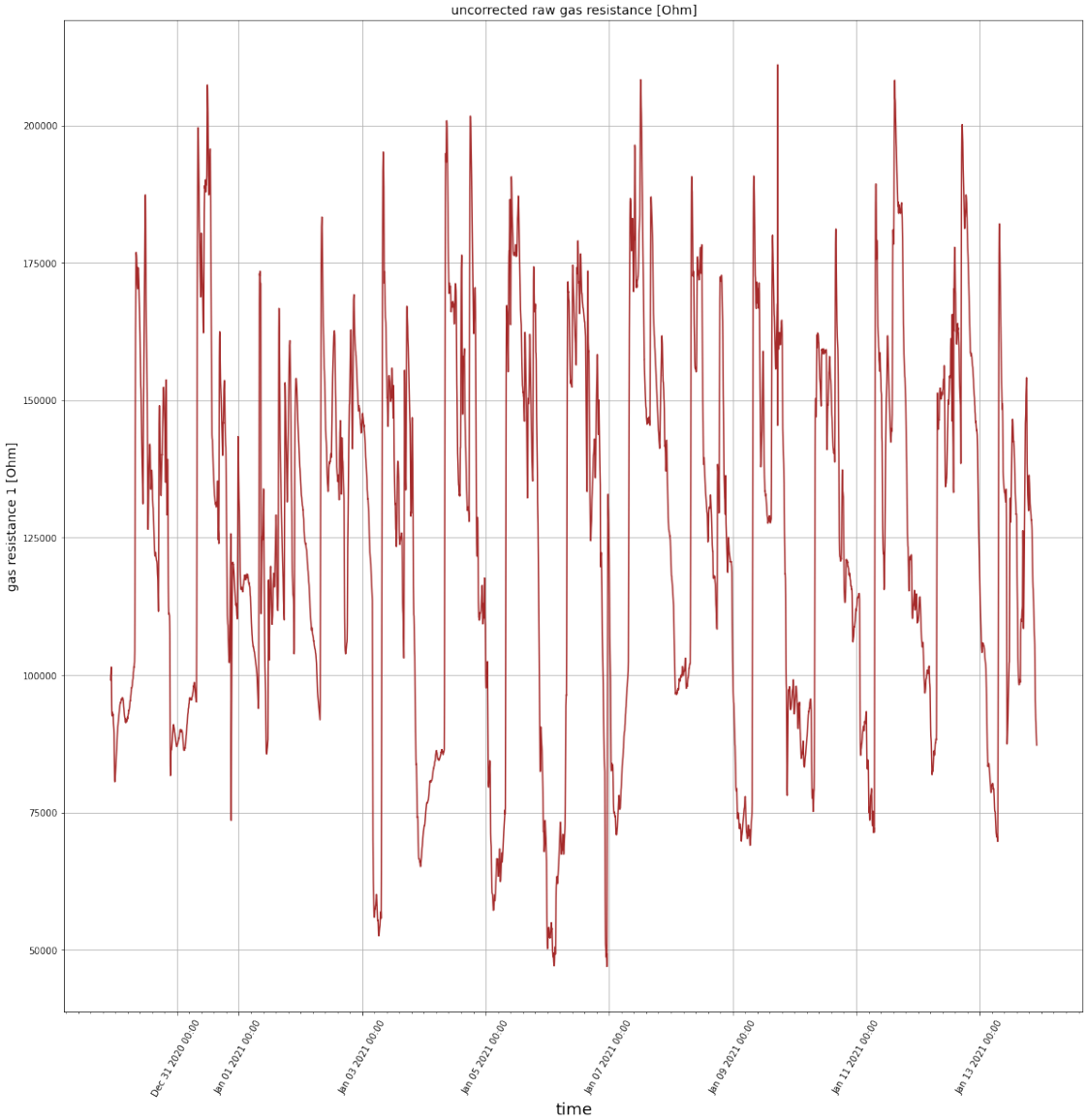
[3]:
```
df.head(-1)
```

[3]:
```
                         Datum  Mode  raw_gas_resistance  relative_humidity  \
0     2020-12-29 22:13:34.000     2               99100               37.3
3     2020-12-29 22:15:43.845     2               99920               37.3
6     2020-12-29 22:20:15.643     2              100260               37.2
9     2020-12-29 22:24:47.332     2              101220               37.2
12    2020-12-29 22:29:19.025     2              101460               37.2
…                         …   …                   …                  …
15129 2021-01-13 21:37:32.971     2               95580               35.3
15132 2021-01-13 21:46:38.513     2               91900               35.4
15135 2021-01-13 21:51:11.255     2               91280               35.4
15138 2021-01-13 22:00:16.743     2               89340               35.4
15141 2021-01-13 22:04:49.485     2               88300               35.4

       temperature
0             23.4
3             23.4
6             23.4
9             23.4
12            23.4
…              …
15129         24.4
15132         24.4
15135         24.4
15138         24.4
15141         24.4

[5048 rows x 5 columns]
```

Time series diagram of the measured raw gas resistance

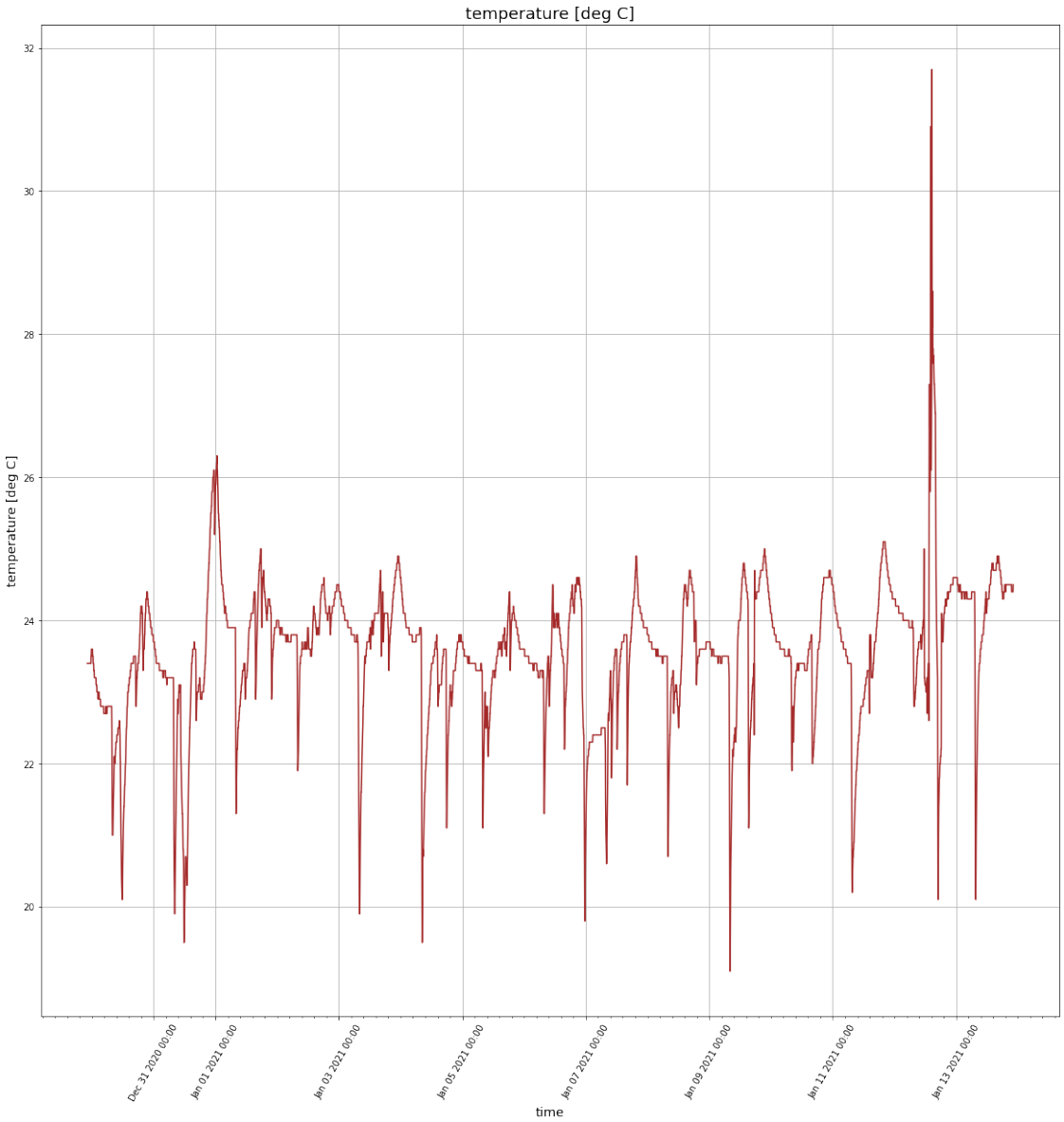
[4]:
```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)

fig, ax = plt.subplots(figsize=(20, 20))
```

```
plt.xticks(rotation=60)
ax.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))

ax.xaxis.set_minor_locator(AutoMinorLocator())

ax.plot_date(df['Datum'], df['raw_gas_resistance'], linestyle='solid', marker="␣
 ↪", color='brown')
plt.title('uncorrected raw gas resistance [Ohm]', fontsize=14)
plt.xlabel('time', fontsize=18)
plt.ylabel('gas resistance 1 [Ohm]', fontsize=14)
plt.grid(True)
plt.show()
```



Time series diagram of the measured temperature

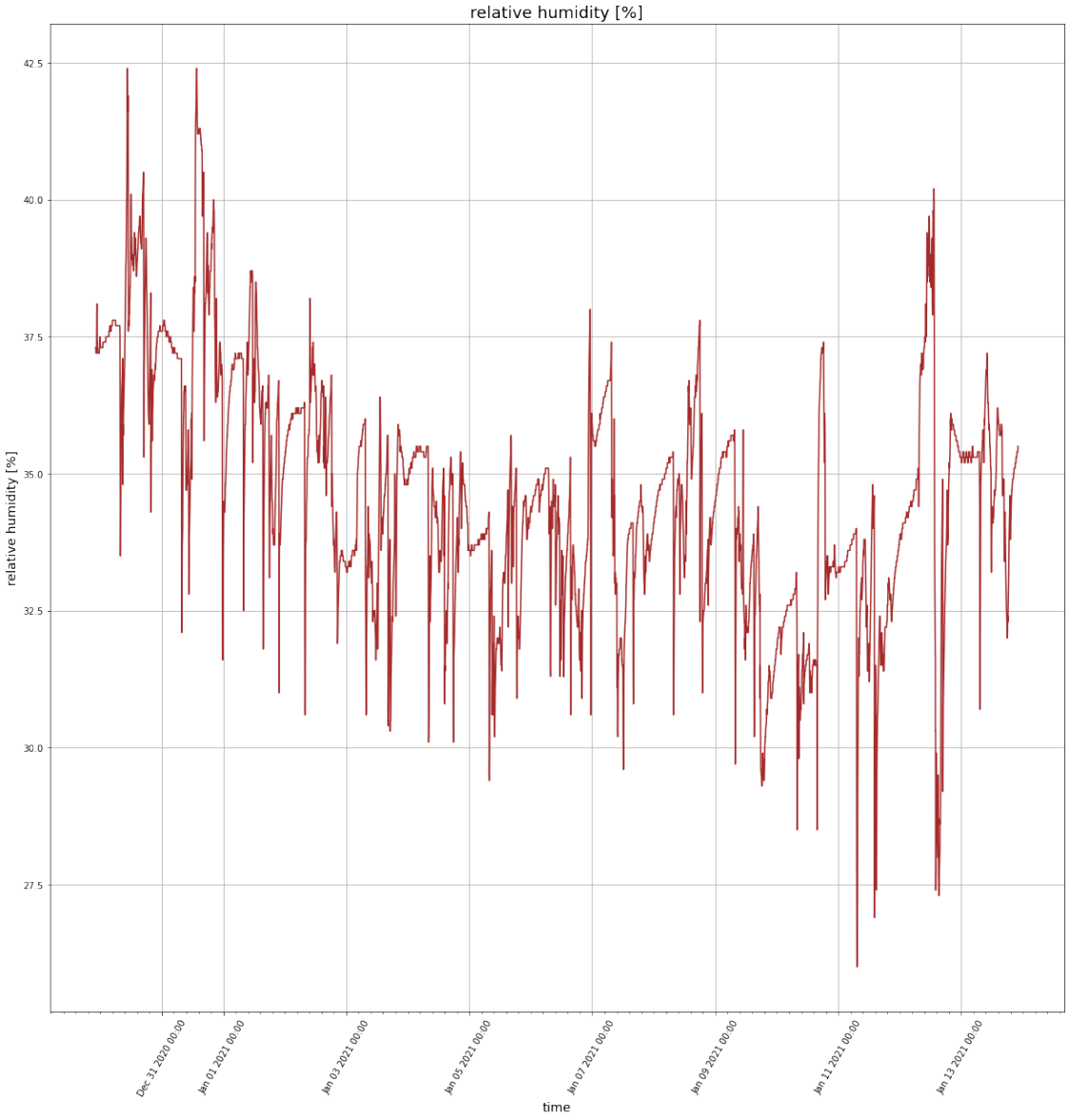
[5]:
```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)

fig, ax = plt.subplots(figsize=(20, 20))
plt.xticks(rotation=60)
ax.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))

ax.xaxis.set_minor_locator(AutoMinorLocator())

ax.plot_date(df['Datum'], df['temperature'], linestyle='solid', marker=" ",␣
 ↪color='brown')
plt.title('temperature [deg C]', fontsize=18)
plt.xlabel('time', fontsize=14)
plt.ylabel('temperature [deg C]', fontsize=14)
plt.grid(True)
plt.show()
```



Time series diagram of the measured relative humidity

[6]:
```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)

fig, ax = plt.subplots(figsize=(20, 20))
plt.xticks(rotation=60)
ax.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))
```

```
ax.xaxis.set_minor_locator(AutoMinorLocator())

ax.plot_date(df['Datum'], df['relative_humidity'], linestyle='solid', marker="␣
 ↪", color='brown')
plt.title('relative humidity [%]', fontsize=18)
plt.xlabel('time', fontsize=14)
plt.ylabel('relative humidity [%]', fontsize=14)
plt.grid(True)
plt.show()
```



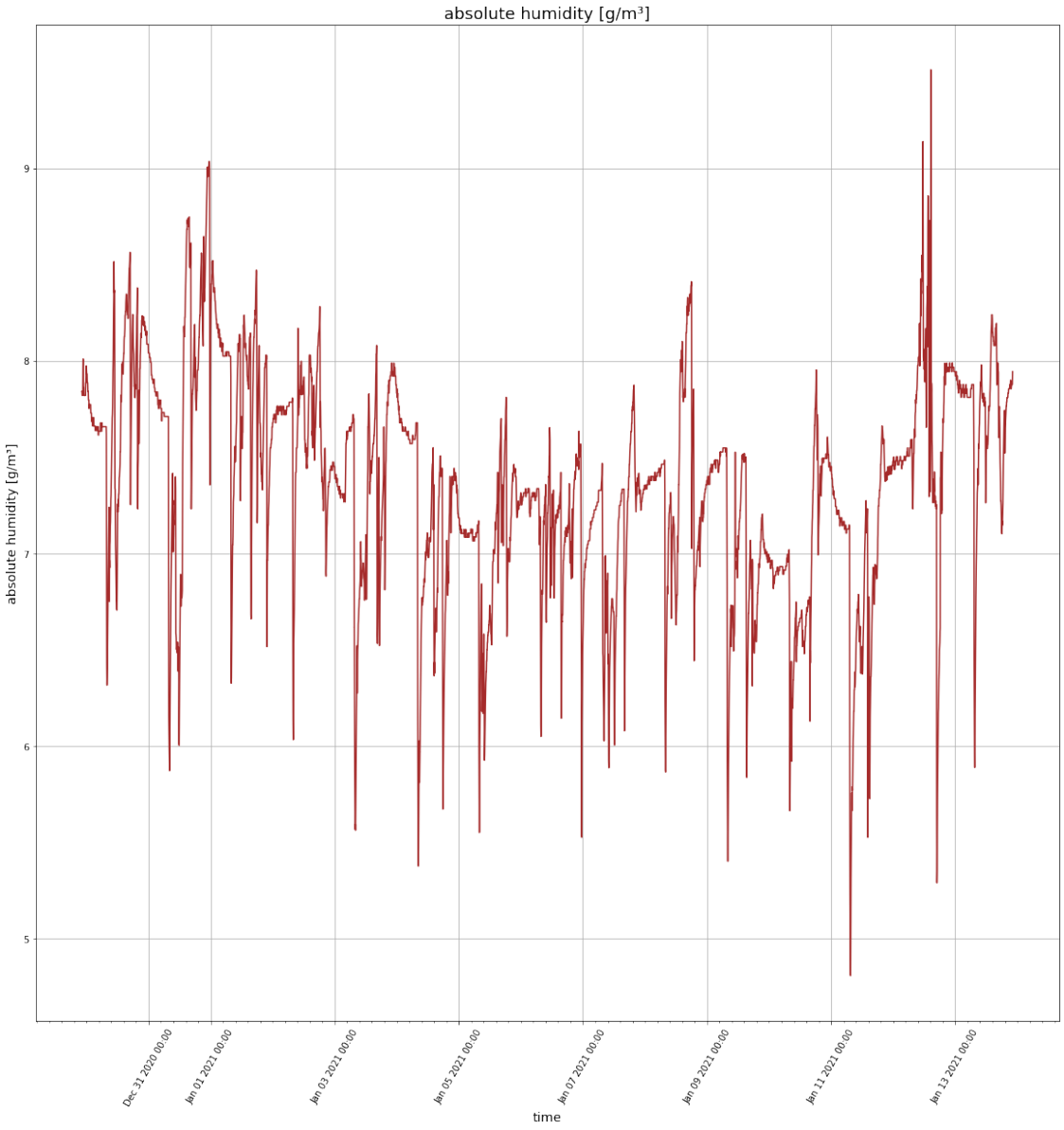
Calculate and plot the absolute humidity from temperature and relative humidity by an approxi-

mate formuala (same formula as used inside the sensor)

[7]:
```
import numpy as np
# Create a function that  calculates the absolute humidity from the two␣
 ↪arguments 'temperature' and 'relative humidity'
# see for details https://www.kompf.de/weather/vent.html or  https://
 ↪rechneronline.de/barometer/luftfeuchtigkeit.php for x-checking the␣
 ↪calculated result

a = 6.112
b = 17.67
c = 243.5

# Compute saturated water vapor pressure in hPa
# Param t - temperature in °C
def svp(t):
  svp = a * np.exp((b*t)/(c+t))
  return svp

# Compute actual water vapor pressure in hPa
# Param rh - relative humidity in %
# Param t - temperature in °C
def vp(rh, t):
  vp = rh/100. * svp(t)
  return vp

# Compute the absolute humidity in g/m³
# Param rh - relative humidity in %
# Param t - temperature in °C
def calculate_absolute_humidity(t, rh):
  mw = 18.016 # kg/kmol (Molekulargewicht des Wasserdampfes)
  rs = 8314.3 # J/(kmol*K) (universelle Gaskonstante)
  ah = 10**5 * mw/rs * vp(rh, t)/(t + 273.15)
  #return the absolute humidity in [g/m³]
  return ah

# now apply the above defined formulas to get the pandas dataframe column␣
 ↪'absolute_humidity'
df['absolute_humidity'] = calculate_absolute_humidity(df['temperature'],␣
 ↪df['relative_humidity'])

import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)
```

```
fig, ax = plt.subplots(figsize=(20, 20))
plt.xticks(rotation=60)
ax.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))

ax.xaxis.set_minor_locator(AutoMinorLocator())

ax.plot_date(df['Datum'], df['absolute_humidity'], linestyle='solid', marker="␣
 ↪", color='brown')
plt.title('absolute humidity [g/m³]', fontsize=18)
plt.xlabel('time', fontsize=14)
plt.ylabel('absolute humidity [g/m³]', fontsize=14)
plt.grid(True)
plt.show()
```

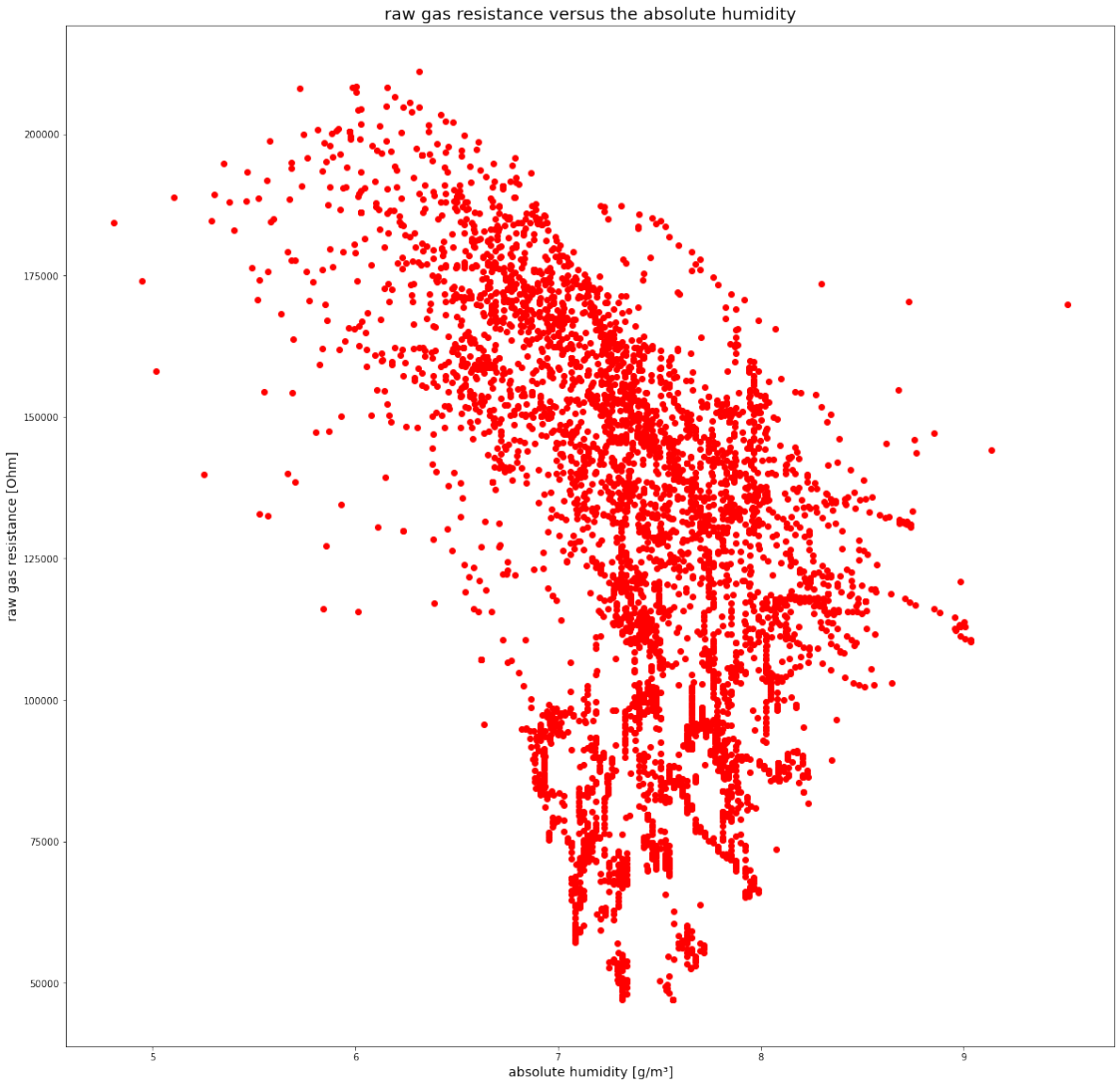
```
<ipython-input-7-6b16bc84f520>:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['absolute_humidity'] = calculate_absolute_humidity(df['temperature'],
df['relative_humidity'])
```



Scatter plot of raw gas resistance versus the absolute humidityy, is the dependency somehow linear (should be for a multilinear regression)?

[8]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


plt.figure(figsize=(20,20))

plt.scatter(df['absolute_humidity'], df['raw_gas_resistance'], color='red')
```

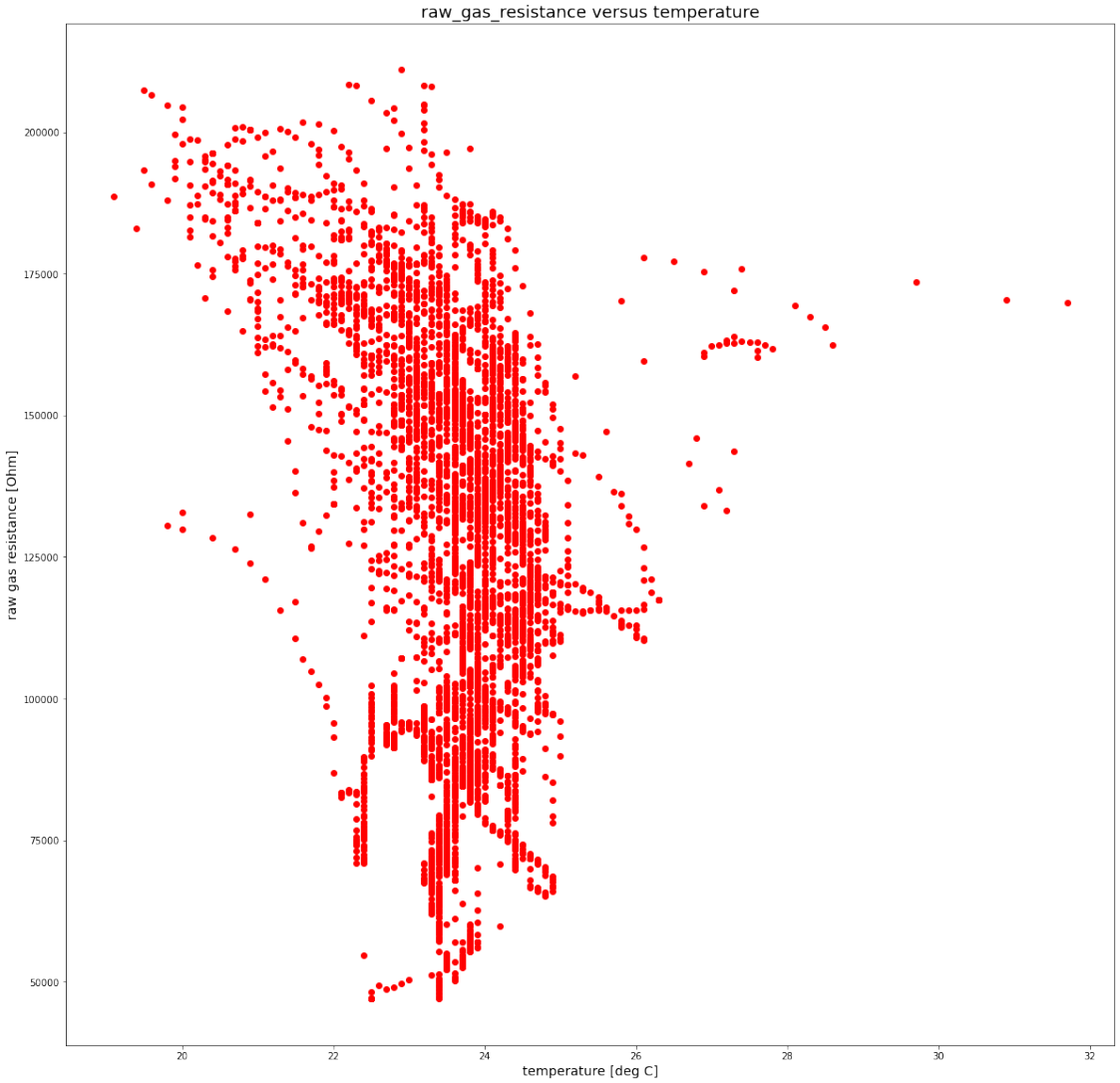
```
plt.title('raw gas resistance versus the absolute humidity', fontsize=18)
plt.xlabel('absolute humidity [g/m³]', fontsize=14)
plt.ylabel('raw gas resistance [Ohm]', fontsize=14)
plt.show()
```



Scatter plot of raw gas resistance versus the temperature, is the dependency somehow linear (should be for a multilinear regression)?

[9]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


plt.figure(figsize=(20,20))
```

```
plt.scatter(df['temperature'], df['raw_gas_resistance'], color='red')
plt.title('raw_gas_resistance versus temperature', fontsize=18)
plt.xlabel('temperature [deg C]', fontsize=14)
plt.ylabel('raw gas resistance [Ohm]', fontsize=14)
plt.grid(False)
plt.show()
```



Execute a multiple linear regression of raw gas resistance oin dependency of the absolute humidity and the temperature use the prediction 'predictions1' of the mutiple linear regression to create a corrected gas resistance 'residuals' with eliminated influence of the absolute humidity and the temperature create a normalized scaled corrected gas resistance 'normalized_residuals'

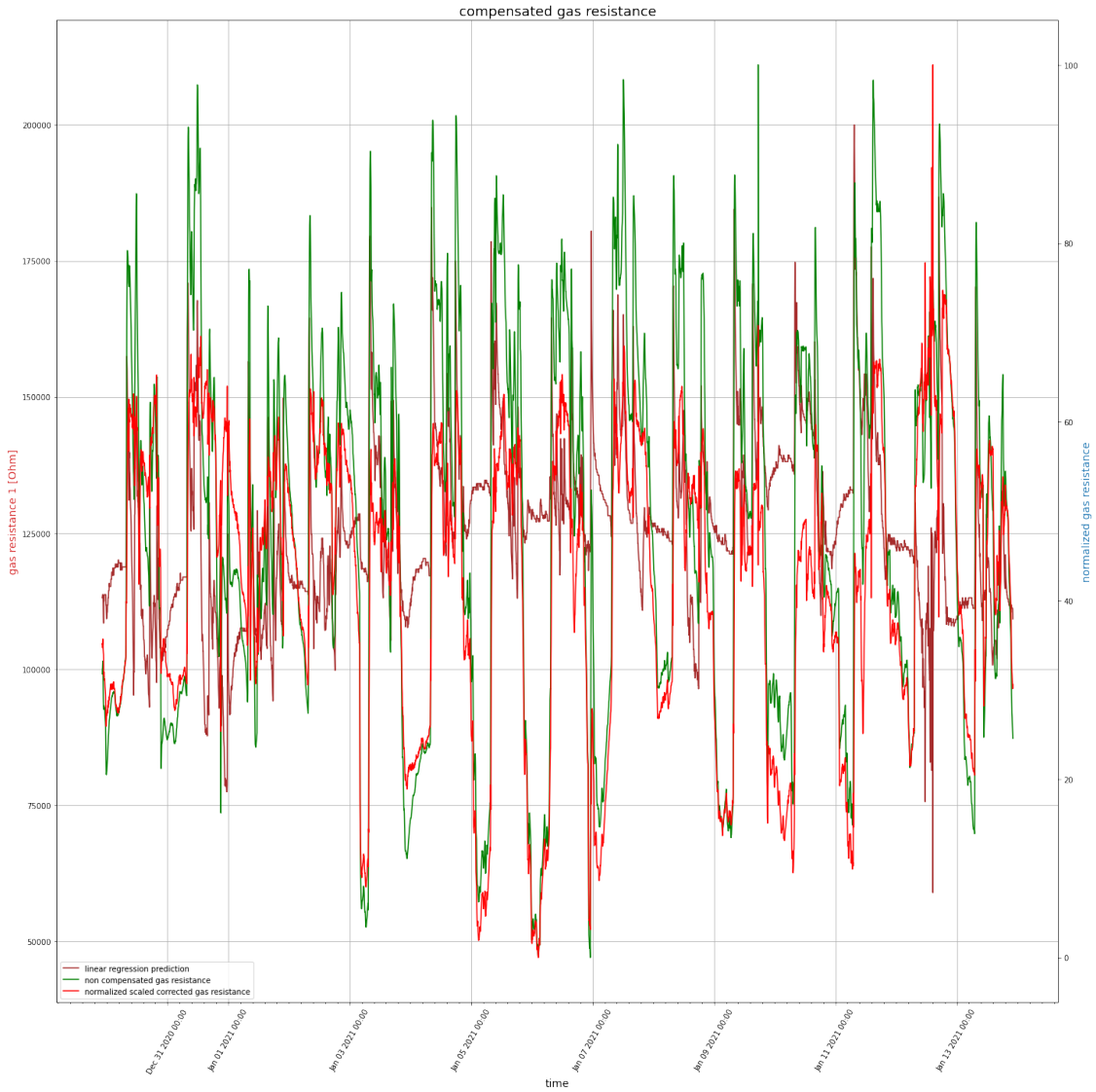
[10]:
```
import pandas as pd
from sklearn import linear_model
```

```
import statsmodels.api as sm
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)


X = df[['temperature','absolute_humidity']] # here we have 2 variables for␣
 ↪multiple regression
Y = df['raw_gas_resistance']

# with sklearn
regr = linear_model.LinearRegression()
regr.fit(X, Y)

print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)

X = sm.add_constant(X)
model = sm.OLS(Y, X).fit()
predictions = model.predict(X)

print_model = model.summary()
print(print_model)
print(model.rsquared)

residuals=df['raw_gas_resistance']-predictions
min_res=min(residuals)
max_res=max(residuals)

#clip min of residual to epsilon in order to avoid a log(0) trap
epsilon=0.0001

normalized_residuals=((residuals-min_res)/(max_res-min_res)).
 ↪clip(epsilon,None)*100


fig, ax1 = plt.subplots(figsize=(20, 20))
plt.xticks(rotation=60)
ax1.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))

ax1.xaxis.set_minor_locator(AutoMinorLocator())

lns1=ax1.plot_date(df['Datum'], predictions, linestyle='solid', marker=" ",␣
 ↪color='brown', label='linear regression prediction')
```

```
lns2=ax1.plot_date(df['Datum'], df['raw_gas_resistance'], linestyle='solid',␣
 ↪marker=" ", color='green', label='non compensated gas resistance')
color = 'tab:red'
ax1.set_xlabel('time', fontsize=14)
ax1.set_ylabel('gas resistance 1 [Ohm]', color=color, fontsize=14)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
ax2.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))

ax2.xaxis.set_minor_locator(AutoMinorLocator())

color = 'tab:blue'
ax2.set_ylabel('normalized gas resistance', color=color, fontsize=14)  # we␣
 ↪already handled the x-label with ax1
lns3=ax2.plot_date(df['Datum'], normalized_residuals, linestyle='solid',␣
 ↪marker=" ", color='red', label='normalized scaled corrected gas resistance')

plt.title('compensated gas resistance', fontsize=18)

ax1.grid(True)
lns = lns1+lns2+lns3
labs = [l.get_label() for l in lns]
ax1.legend(lns, labs, loc="lower left")

fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.show()
```

```
Intercept:
 352749.6148985638
Coefficients:
 [  -951.04961362 -27722.38305622]
                            OLS Regression Results
==============================================================================
Dep. Variable:     raw_gas_resistance   R-squared:                       0.203
Model:                            OLS   Adj. R-squared:                  0.202
Method:                 Least Squares   F-statistic:                     640.9
Date:                Fri, 15 Jan 2021   Prob (F-statistic):          9.37e-249
Time:                        14:18:34   Log-Likelihood:                -59499.
No. Observations:                5049   AIC:                         1.190e+05
Df Residuals:                    5046   BIC:                         1.190e+05
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
=====
                         coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
```

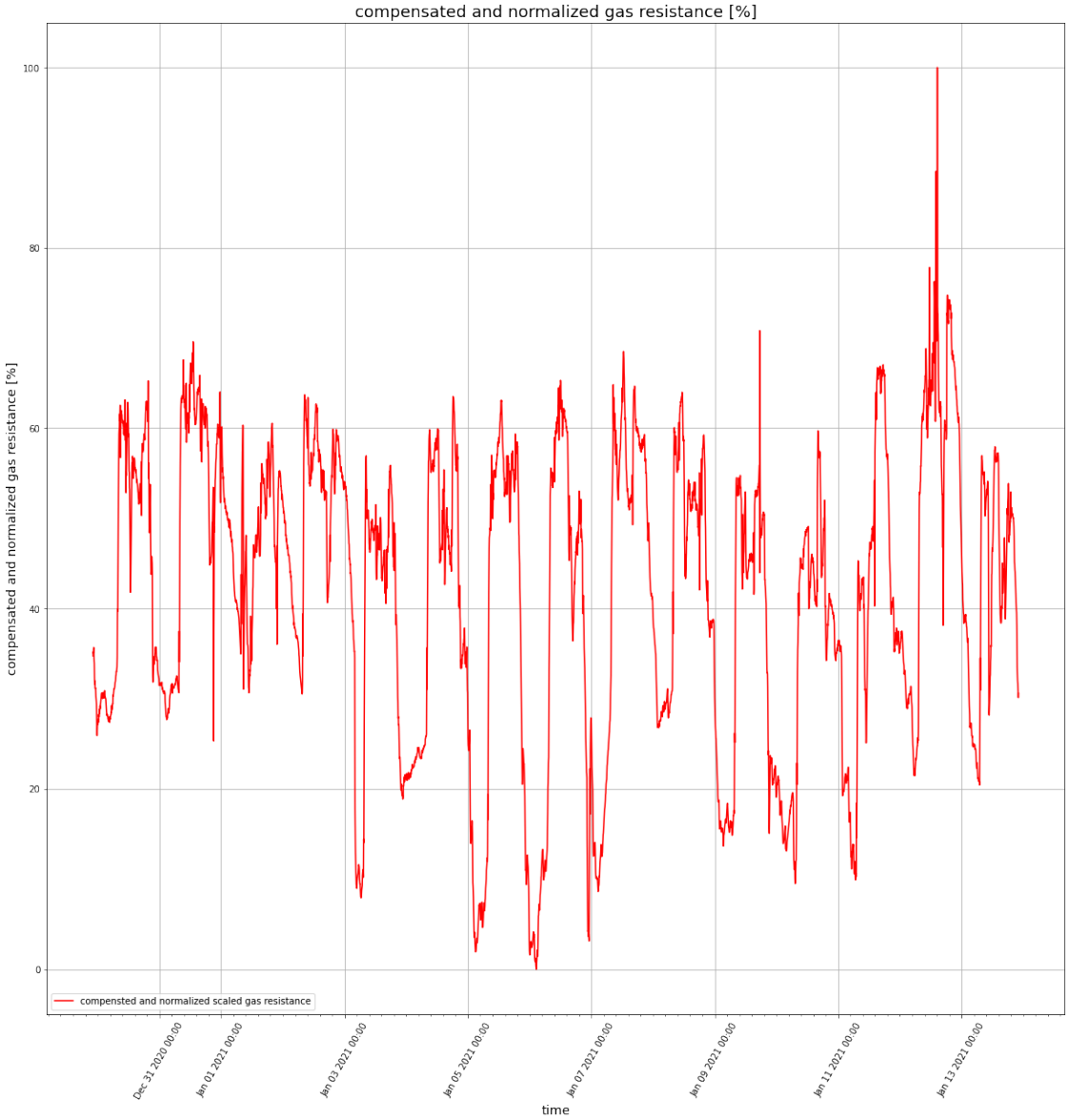
```
-----
const               3.527e+05    1.1e+04     32.182      0.000    3.31e+05
3.74e+05
temperature         -951.0496    574.196     -1.656      0.098   -2076.722
174.623
absolute_humidity -2.772e+04    993.063    -27.916      0.000   -2.97e+04
-2.58e+04
==============================================================================
Omnibus:                      380.331   Durbin-Watson:                   0.011
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              309.929
Skew:                          -0.523   Prob(JB):                     5.01e-68
Kurtosis:                       2.384   Cond. No.                         607.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
0.20256910480233203
```



time series diagrams of normalized scaled gas resistance; y range is 0.0..100.0 [%]

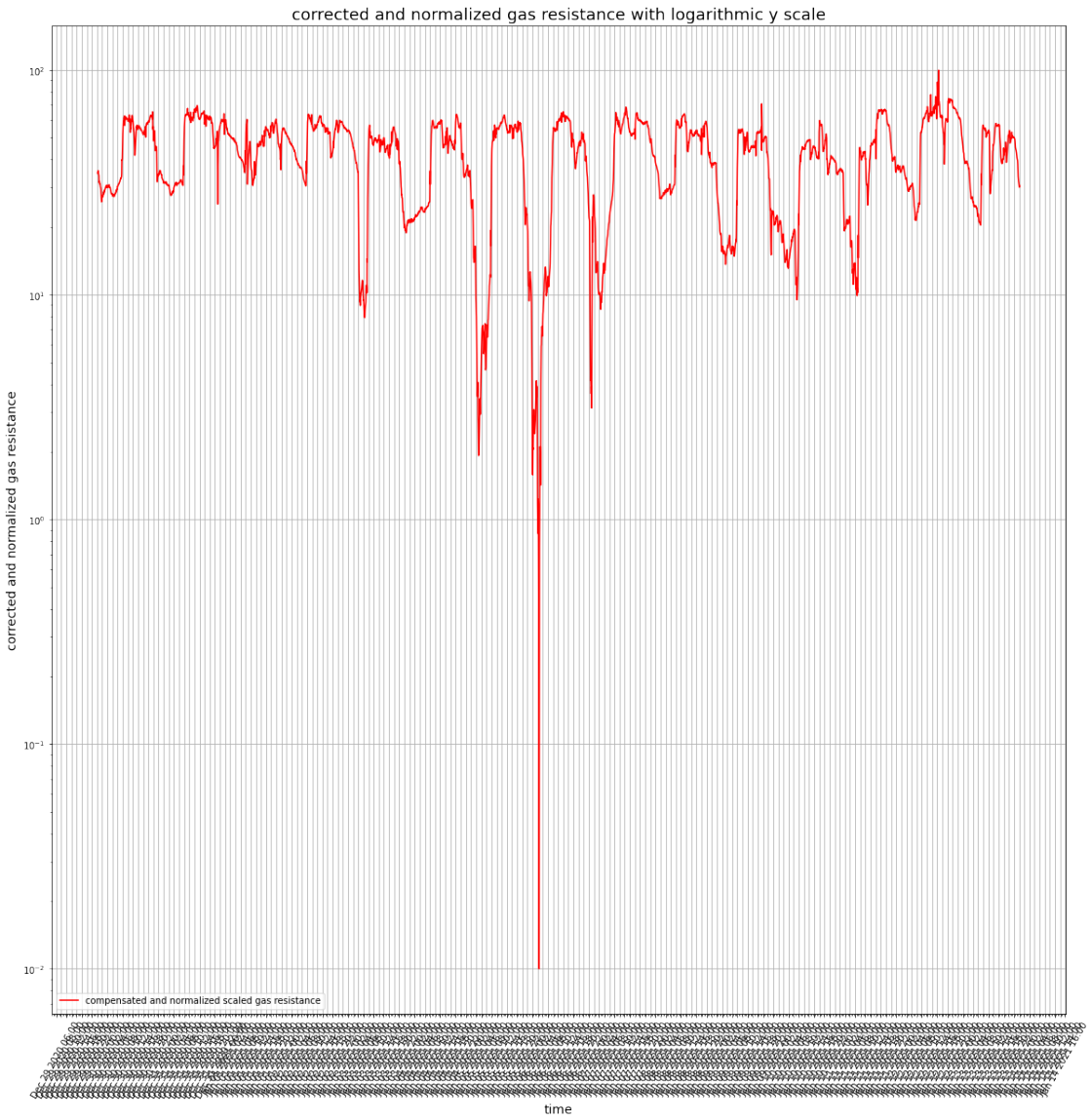
[11]:
```
import pandas as pd
from sklearn import linear_model
import statsmodels.api as sm
import matplotlib.pyplot as plt
from datetime import datetime
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)


fig, ax = plt.subplots(figsize=(20, 20))
```

```
plt.xticks(rotation=60)
ax.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))

ax.xaxis.set_minor_locator(AutoMinorLocator())

plt.plot_date(df['Datum'], normalized_residuals, linestyle='solid', marker=" ",␣
 ↪color='red', label='compensted and normalized scaled gas resistance')

plt.title('compensated and normalized gas resistance [%]', fontsize=18)
plt.xlabel('time', fontsize=14)
plt.ylabel('compensated and normalized gas resistance [%]', fontsize=14)
plt.grid(True)
plt.legend(loc ="lower left")
plt.show()
```



time series diagrams of compensated and normalized scaled gas resistance with logarithmic y scale 0.01 .. 100

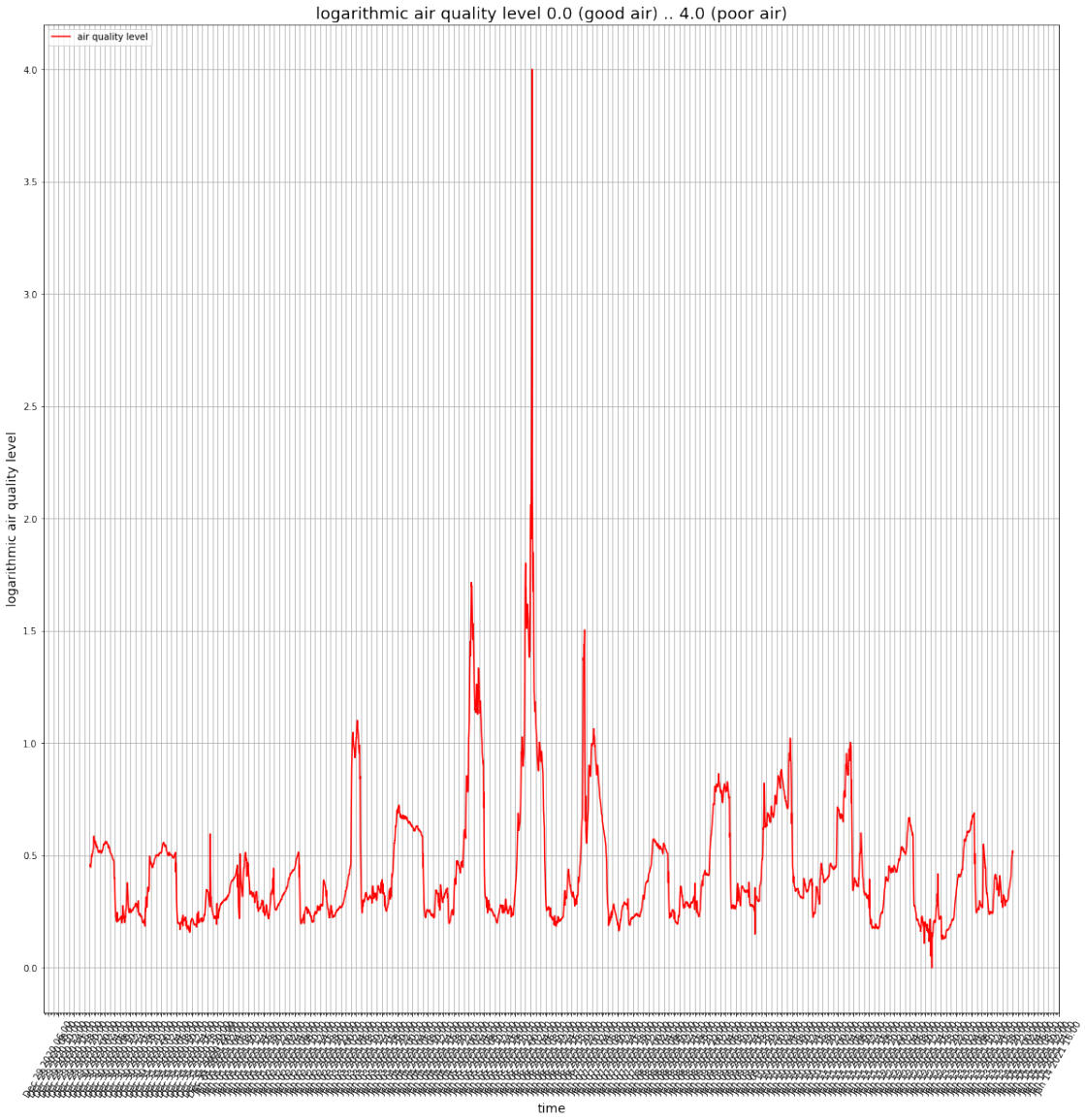
[12]:
```
import pandas as pd
from sklearn import linear_model
import statsmodels.api as sm
import matplotlib.pyplot as plt
from datetime import datetime
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)


fig, ax = plt.subplots(figsize=(20, 20))
plt.yscale('log')
plt.xticks(rotation=60)
ax.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))
hours = mdates.HourLocator(interval = 2)
ax.xaxis.set_major_locator(hours)
ax.xaxis.set_minor_locator(AutoMinorLocator())


plt.xticks(rotation=60)
plt.plot_date(df['Datum'], normalized_residuals, linestyle='solid', marker=" ",␣
 ↪color='red', label='compensated and normalized scaled gas resistance')

plt.title('corrected and normalized gas resistance with logarithmic y scale',␣
 ↪fontsize=18)
plt.xlabel('time', fontsize=14)
plt.ylabel('corrected and normalized gas resistance', fontsize=14)
plt.grid(True)
plt.legend(loc ="lower left")
plt.show()
```



time series diagrams of the logarithmic air quality level the air quality level can vary between 0.0 (fresh air) and 4.0 (very poor air quality)

[13]:
```
import pandas as pd
from sklearn import linear_model
import statsmodels.api as sm
import matplotlib.pyplot as plt
from datetime import datetime
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)
```

```
log_normalized_residuals = -(np.log10(normalized_residuals)-2)


fig, ax = plt.subplots(figsize=(20, 20))
plt.xticks(rotation=60)
ax.xaxis.set_major_formatter(DateFormatter('%b %d %Y %H:%M'))
hours = mdates.HourLocator(interval = 2)
ax.xaxis.set_major_locator(hours)
ax.xaxis.set_minor_locator(AutoMinorLocator())


plt.xticks(rotation=60)
plt.plot_date(df['Datum'], log_normalized_residuals, linestyle='solid',␣
 ↪marker=" ", color='red', label='air quality level')

plt.title('logarithmic air quality level 0.0 (good air) .. 4.0 (poor air)',␣
 ↪fontsize=18)
plt.xlabel('time', fontsize=14)
plt.ylabel('logarithmic air quality level', fontsize=14)
plt.grid(True)
plt.legend(loc ="upper left")
plt.show()
```



Please check whether the R-squared (uncentered) of the multiple linear regression above is sufficiently good (should be > 0.7): R-squared (also called coefficient of determination) is the portion of variance in the dependent variables that can be explained by the independent variables. Hence, as a rule of thumb for interpreting the strength of a relationship based on its R-squared value is: - if R-squared value < 0.3 this value is generally considered as None or very weak effect size - if R-squared value $0.3 < r < 0.5$ this value is generally considered as weak or low effect size - if R-squared value $0.5 < r < 0.7$ this value is generally considered as moderate effect size - if R-squared value $0.7 < r < 1.0$ this value is generally considered as strong effect size

If R-squared value is < 0.3, the collected history may be too short. Please try to collect datapoints for a longer timeframe!

[14]:
```
print(model.rsquared)
```

```
0.20256910480233203
```

Please enter the following parameters of the multilinear regression into the Homematic/RaspberryMatic WebUI page 'Startseite > Einstellungen > Geräte > Geräte-/ Kanalparameter einstellen' of your concerning BME680 AQ sensor device which was the source of the history.csv file above

[15]:
```
print("\nPlease enter the WebUI device parameter 'WEATHER|mlr_alpha'         =␣
 ↪%11.3lf" % regr.coef_[0])
print("Please enter the WebUI device parameter 'WEATHER|mlr_beta'          =␣
 ↪%11.3lf" % regr.coef_[1])
print("Please enter the WebUI device parameter 'WEATHER|mlr_delta'         =␣
 ↪%11.3lf" % regr.intercept_)

import datetime
now = datetime.datetime.now()
print("\n\nPlease check whether current date and time are correct: ")
print(str(now))
```

```
Please enter the WebUI device parameter 'WEATHER|mlr_alpha'         =
-951.050
Please enter the WebUI device parameter 'WEATHER|mlr_beta'          =
-27722.383
Please enter the WebUI device parameter 'WEATHER|mlr_delta'         =
352749.615


Please check whether current date and time are correct:
2021-01-15 14:18:43.988145
```

Congratulations, you are done!

Please repeat the multilinear regression update of the WebUI device parameters on a regular basis every month or similar as appropriate ..