Execute a multiple linear regression (MLR) for BME680 gas readings of a single sensor

Read the CSV file created by the CCU Historian, ensure that field separator is ';' and decimal separator is '.', if necessary edit CSV file before reading it by the provided script 'csv_convert_historian.bsh'
The provided script 'get_new_history.bsh' is search 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'

In [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],c

df0.head(9)
#df0['Datum']
#type(df0['sensor 1'][0])
```

Out[1]:

	Datum	Mode	raw_gas_resistance	relative_humidity	temperature
0	2020-12-31 16:26:56.439	2	147800	36.9	23.278
1	2020-12-31 16:30:32.862	2	147800	36.9	22.800
2	2020-12-31 16:30:32.866	2	147800	35.6	22.800
3	2020-12-31 16:30:32.873	2	157780	35.6	22.800
4	2020-12-31 16:35:04.477	2	157780	35.6	22.600
5	2020-12-31 16:35:04.486	2	157780	36.5	22.600
6	2020-12-31 16:35:04.492	2	161480	36.5	22.600
7	2020-12-31 16:39:36.091	2	161480	36.5	22.700
8	2020-12-31 16:39:36.099	2	161480	36.8	22.700

In [2]:

```
# keep every 3rd row (CCU historian is tracking every change of a datapoint separately)
# each three consecutive entries in history.csv are identical; therefore we take every thir
df = df0[(df0.index % 3 == 0)]

df.head(9)
```

Out[2]:

	Datum	Mode	raw_gas_resistance	relative_humidity	temperature
0	2020-12-31 16:26:56.439	2	147800	36.9	23.278
3	2020-12-31 16:30:32.873	2	157780	35.6	22.800
6	2020-12-31 16:35:04.492	2	161480	36.5	22.600
9	2020-12-31 16:39:36.106	2	162460	36.8	22.700
12	2020-12-31 16:44:07.740	2	160520	37.2	22.800
15	2020-12-31 16:48:39.353	2	158000	37.5	22.900
18	2020-12-31 16:53:10.973	2	155280	37.8	22.900
21	2020-12-31 16:57:42.592	2	153440	38.1	23.000
24	2020-12-31 17:02:14.215	2	152540	38.1	23.000

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

Out[3]:

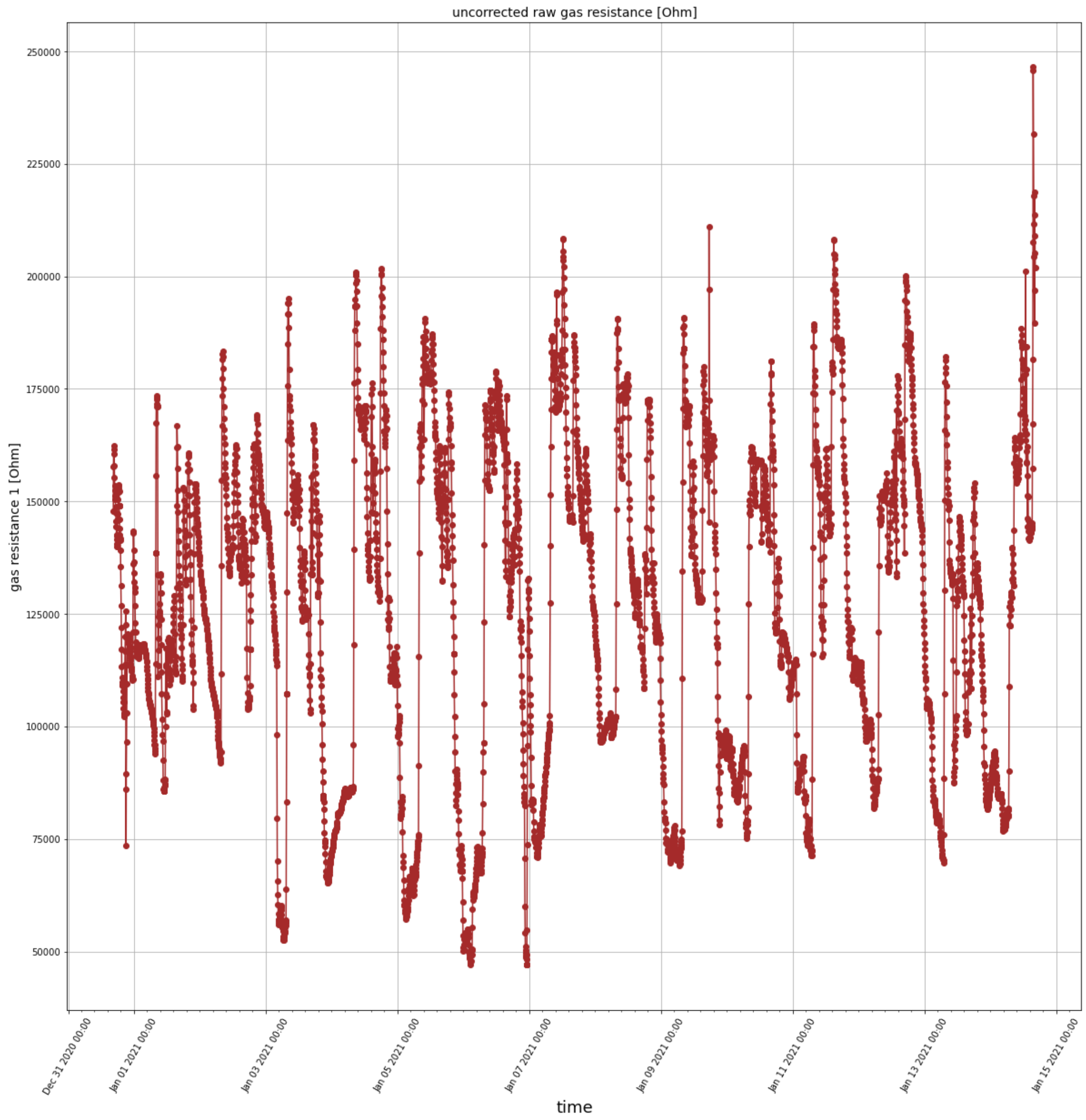
	Datum	Mode	raw_gas_resistance	relative_humidity	temperature
0	2020-12-31 16:26:56.439	2	147800	36.9	23.278
3	2020-12-31 16:30:32.873	2	157780	35.6	22.800
6	2020-12-31 16:35:04.492	2	161480	36.5	22.600
9	2020-12-31 16:39:36.106	2	162460	36.8	22.700
12	2020-12-31 16:44:07.740	2	160520	37.2	22.800
...	...	...	...	...	...
14007	2021-01-14 16:08:06.854	2	189640	27.8	26.100
14010	2021-01-14 16:09:02.681	2	213600	28.0	25.000
14013	2021-01-14 16:11:22.754	2	218780	26.9	25.500
14016	2021-01-14 16:15:55.504	2	209040	26.8	26.400
14019	2021-01-14 16:20:29.140	2	205100	27.1	26.100

4674 rows × 5 columns

Time series diagram of the measured raw gas resistance

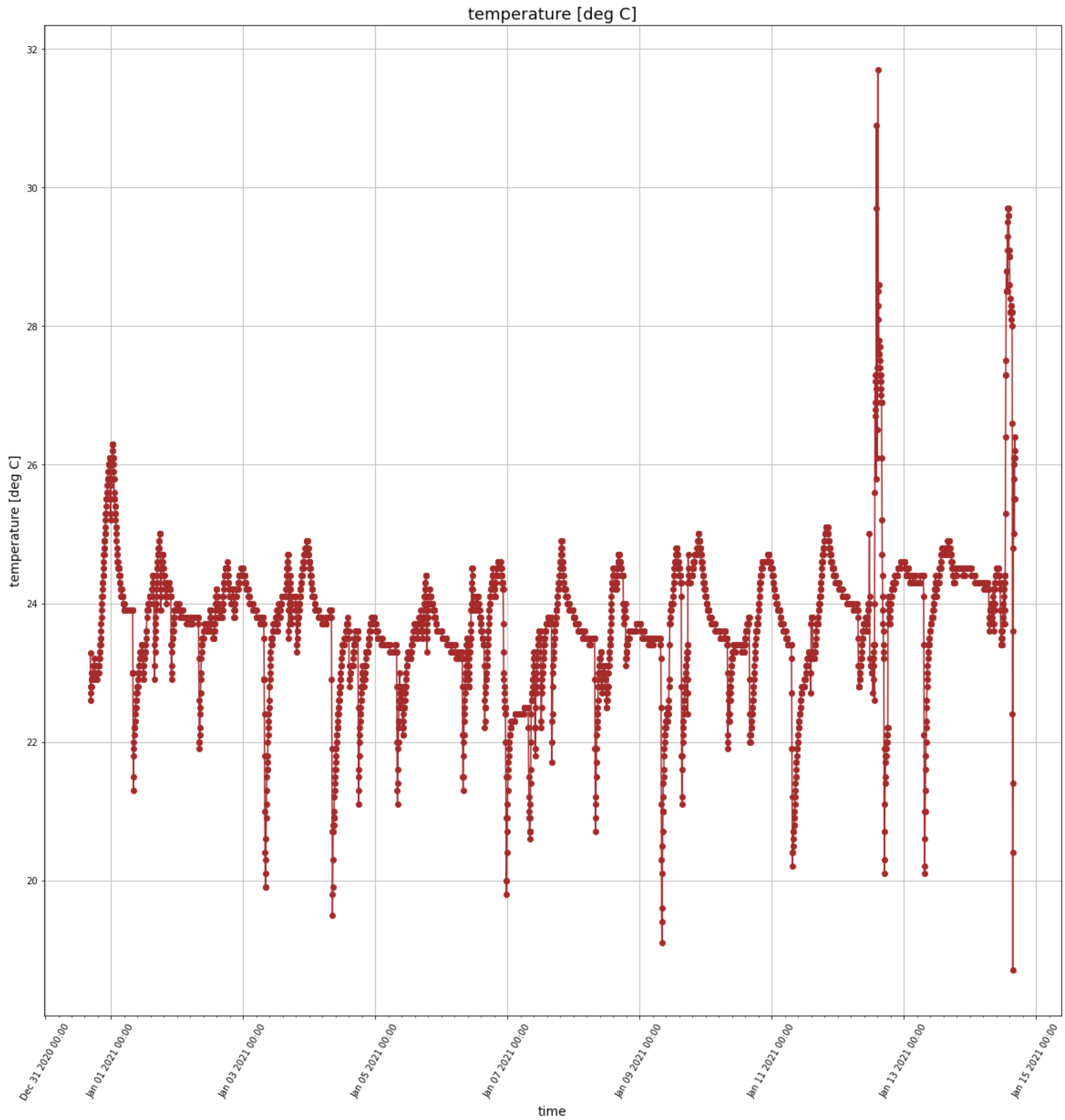
In [4]:

```
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', 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()
```



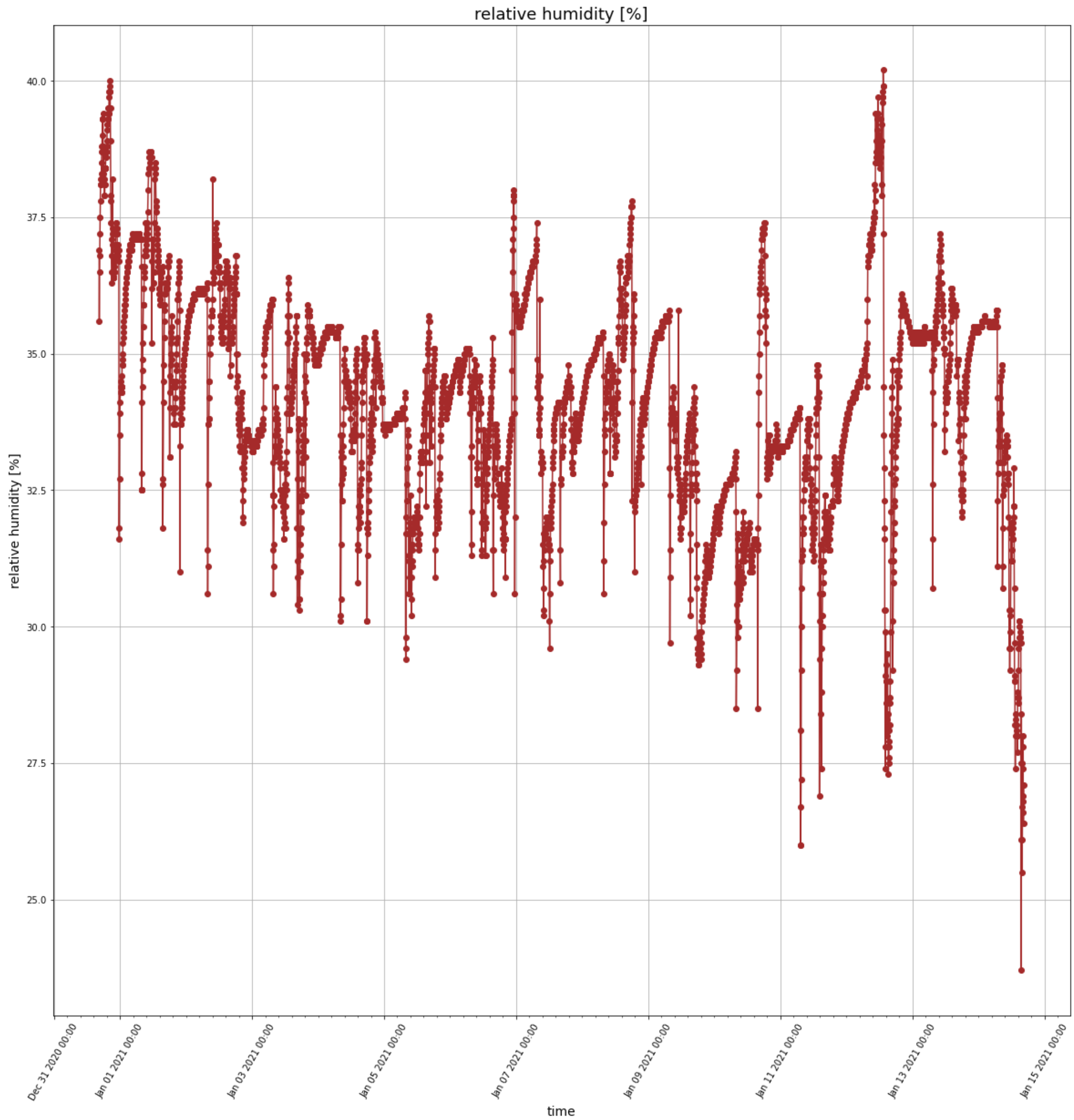
Time series diagram of the measured temperature

In [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', 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

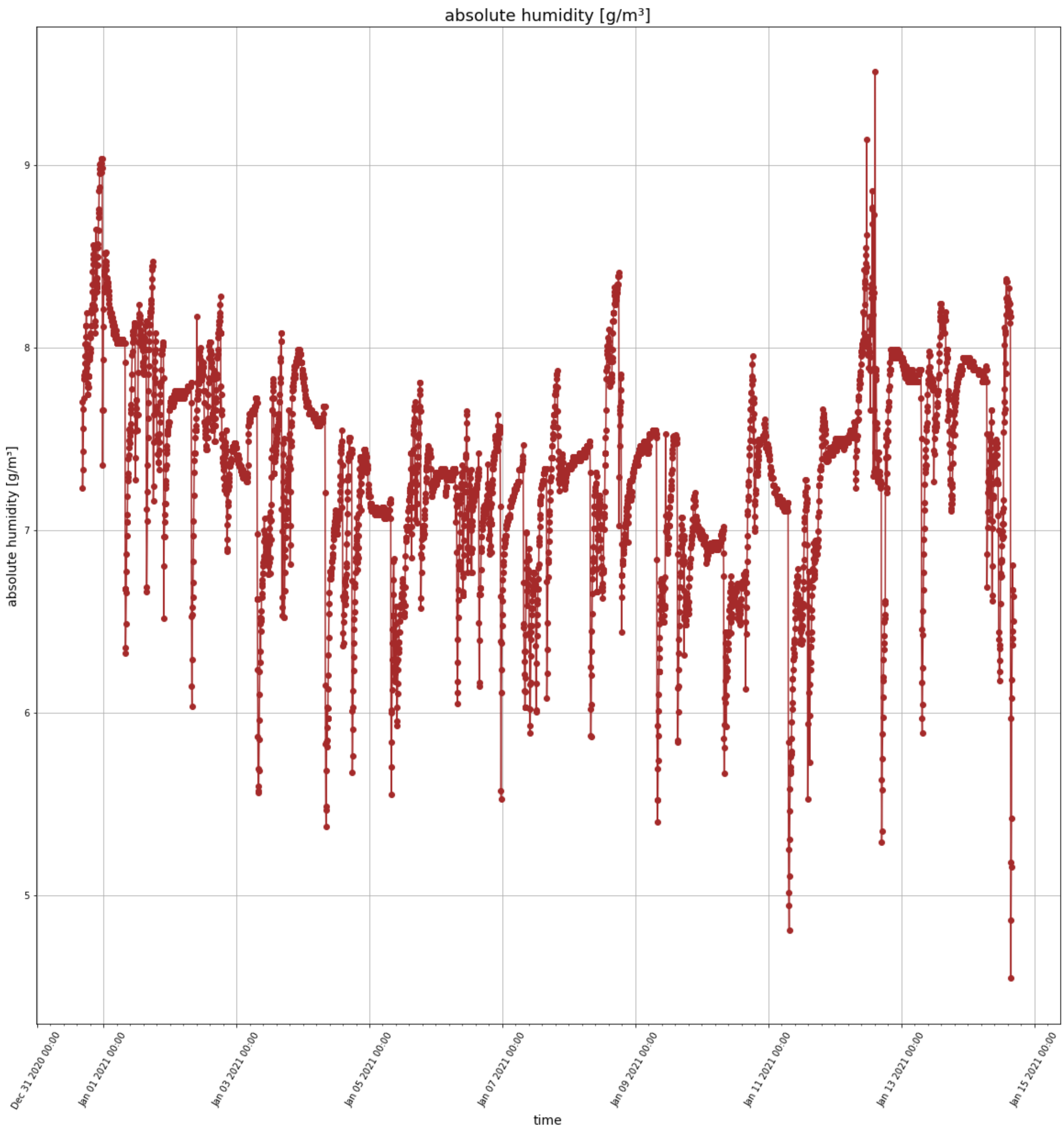
In [6]:
```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)

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', 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 approximate formuala (same formula as used inside the sensor)

In [7]:

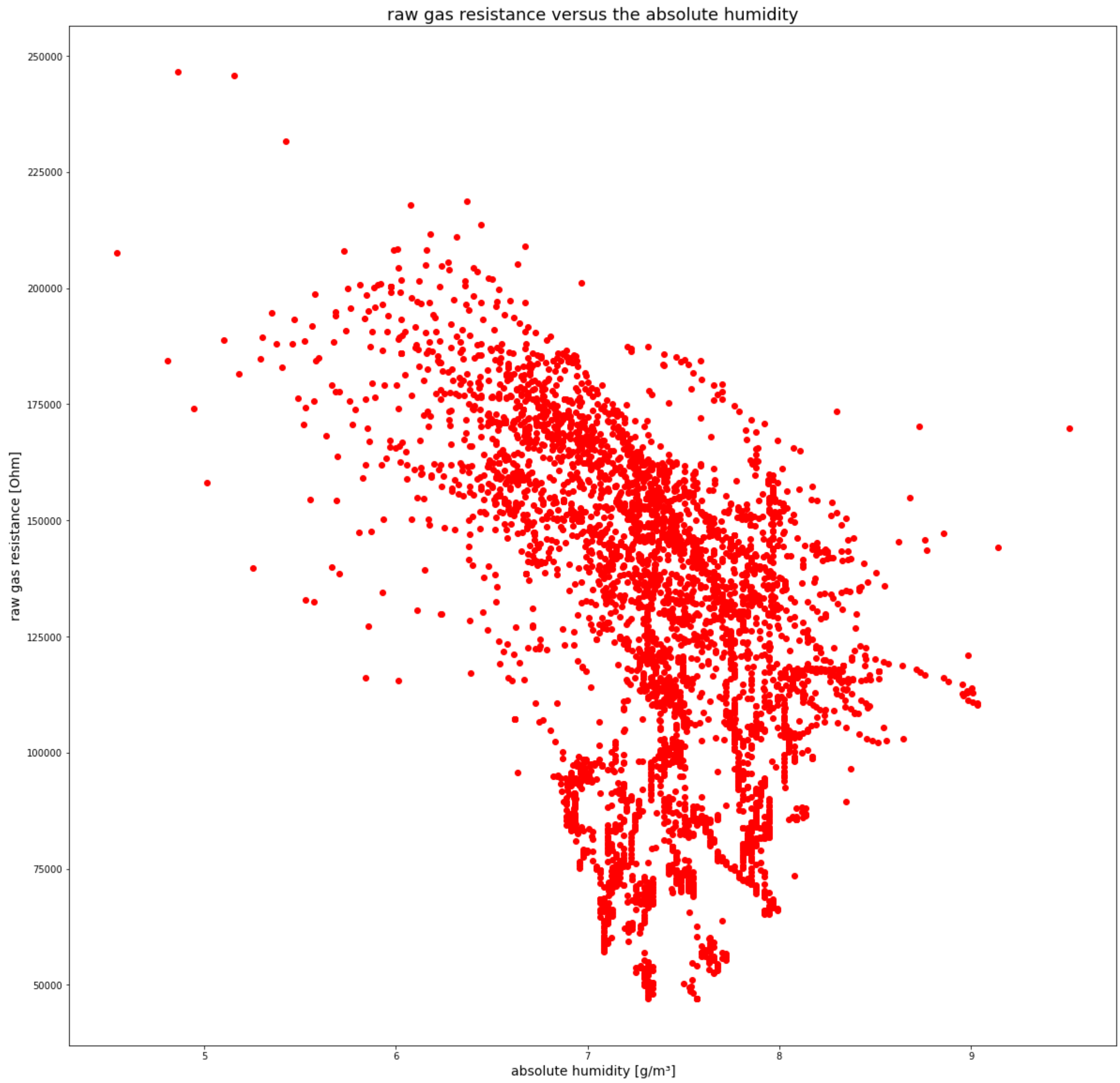
```
import numpy as np
# Create a function that  calculates the absolute humidity from the two arguments 'temperatu
# see for details https://www.kompf.de/weather/vent.html or  https://rechneronline.de/barome

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

pd.set_option('mode.chained_assignment', None)
# now apply the above defined formulas to get the pandas dataframe column 'absolute_humidity
df['absolute_humidity'] = calculate_absolute_humidity(df['temperature'], df['relative_humidi

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', 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()


```



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

In [8]:

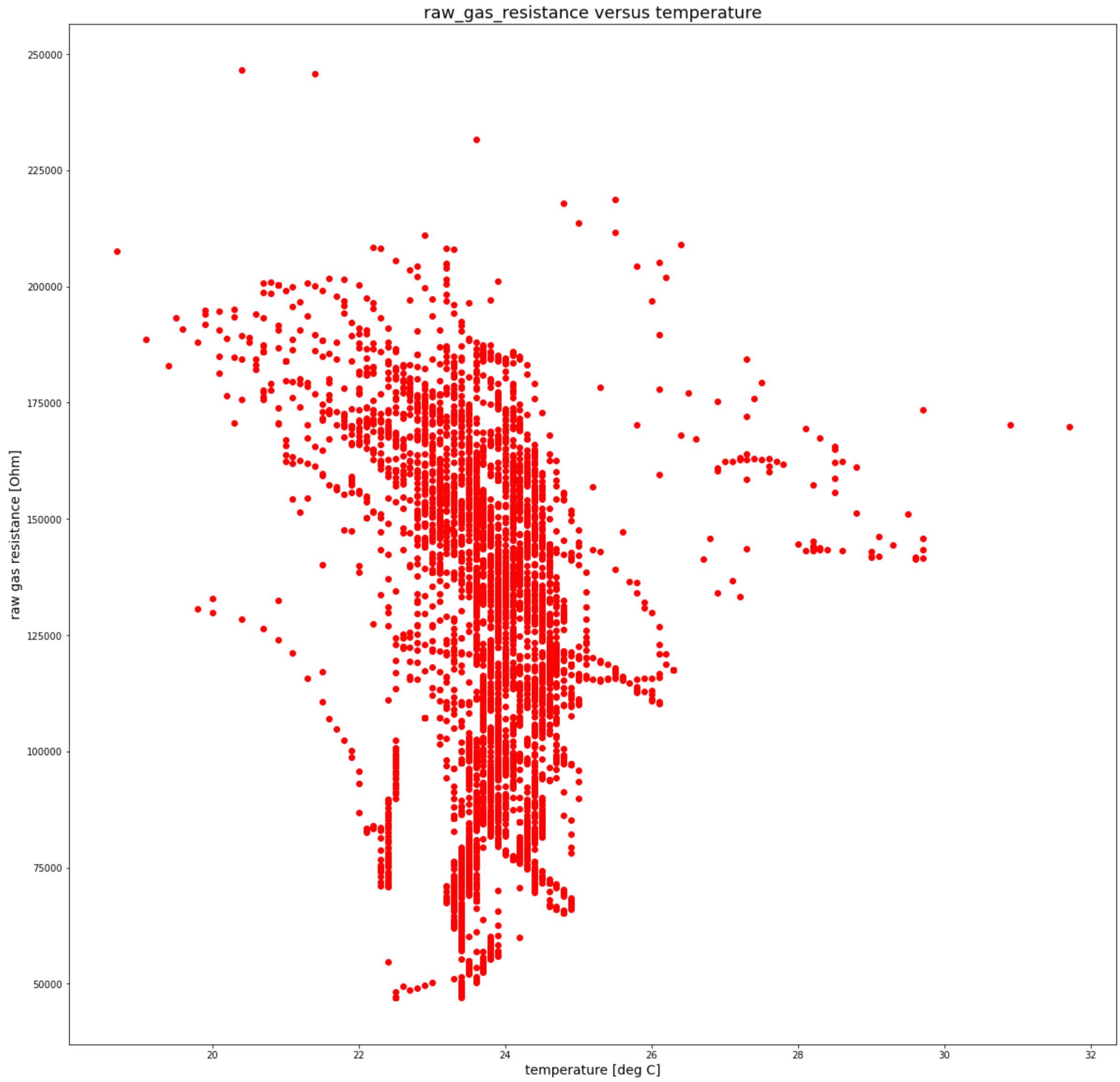
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


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)?

In [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()
```



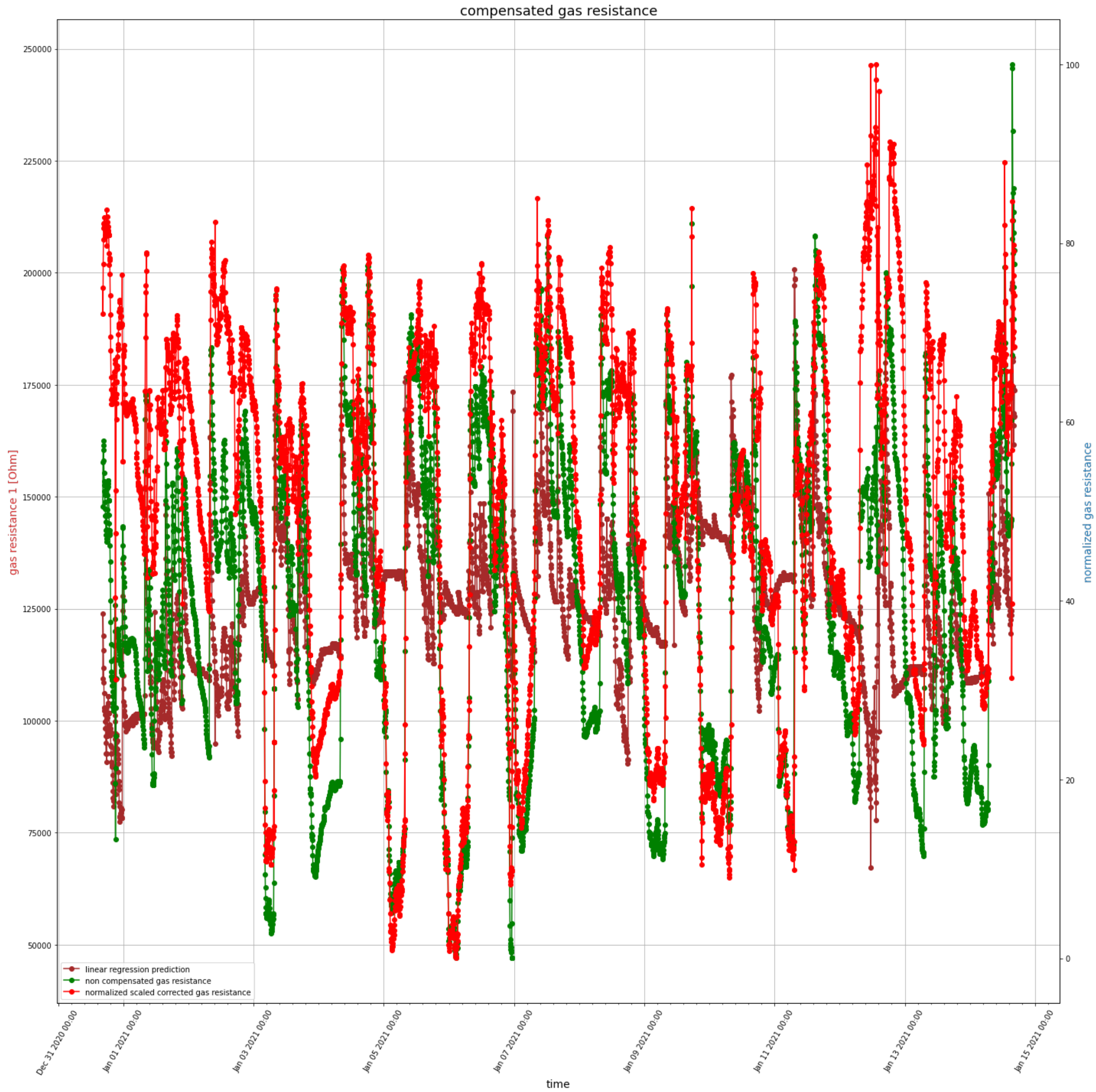
Execute a multiple linear regression of raw gas resistance oin dependency of the absolute humidity and the temperature
use the prediction 'predictions1' of the mutiple linear regression to create a corrected gas resistance 'residuals' with eliminated influence of the absolute humidity and the temperature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

In [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 regressi
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',color='brown', label='linear
lns2=ax1.plot_date(df['Datum'], df['raw_gas_resistance'], linestyle='solid',color='green', l
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
lns3=ax2.plot_date(df['Datum'], normalized_residuals, linestyle='solid', color='red', label=

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: 
 243521.11446688976
Coefficients: 
 [  6603.48933636 -37355.38940408]
                            OLS Regression Results                            
==============================================================================
Dep. Variable:     raw_gas_resistance   R-squared:                       0.223
Model:                            OLS   Adj. R-squared:                  0.223
Method:                 Least Squares   F-statistic:                     671.6
Date:                Sun, 04 Jun 2023   Prob (F-statistic):          4.33e-257
```

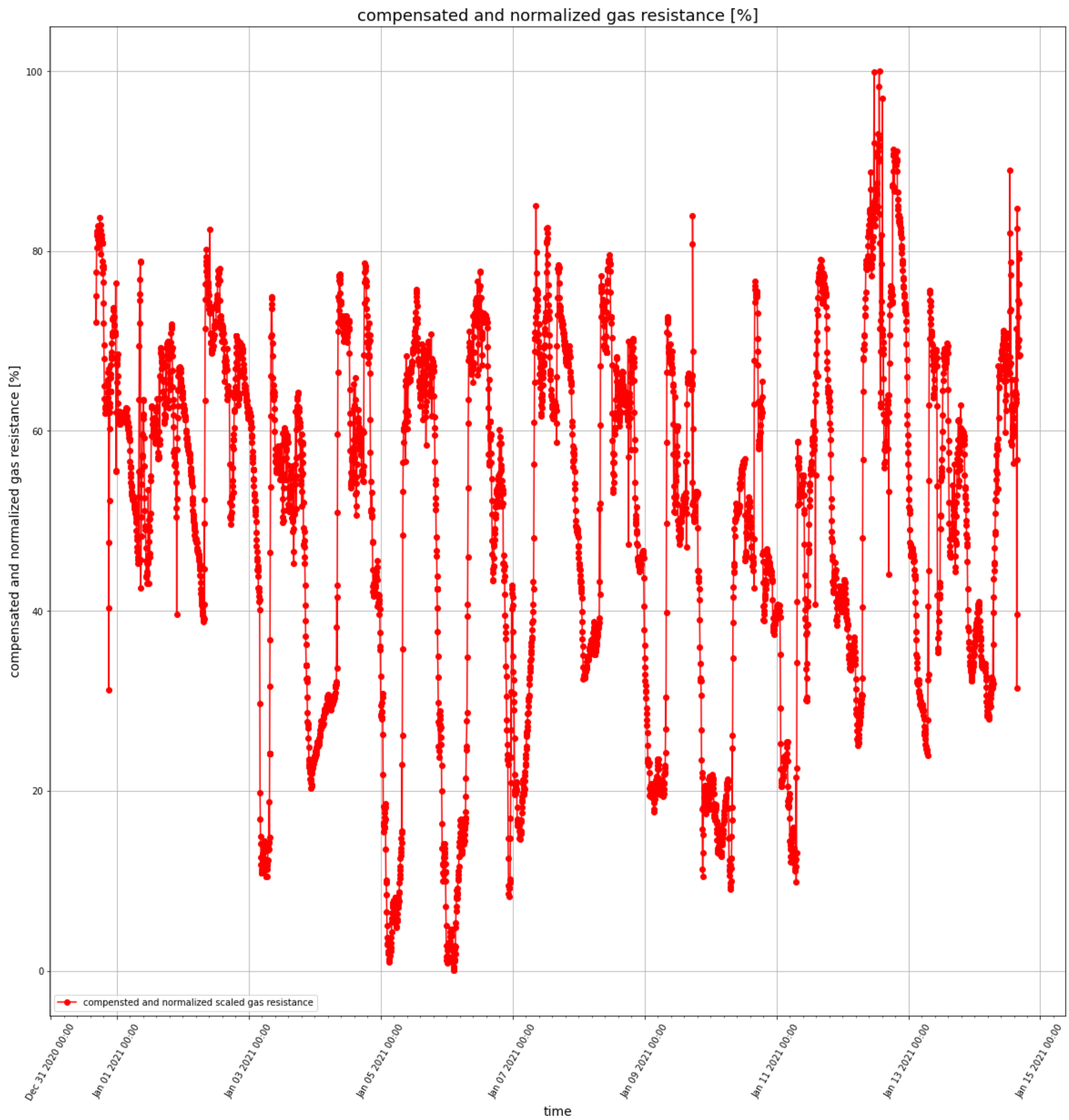
```
Time:                        13:30:56   Log-Likelihood:                -55087.
No. Observations:                4675   AIC:                         1.102e+05
Df Residuals:                    4672   BIC:                         1.102e+05
Df Model:                           2
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
const              2.435e+05   1.11e+04     21.999      0.000    2.22e+05    2.65e+05
temperature        6603.4893    610.281     10.820      0.000    5407.051    7799.928
absolute_humidity -3.736e+04   1111.429    -33.610      0.000   -3.95e+04   -3.52e+04
==============================================================================
Omnibus:                      395.538   Durbin-Watson:                   0.015
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              277.025
Skew:                          -0.488   Prob(JB):                     6.99e-61
Kurtosis:                       2.315   Cond. No.                         594.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
0.2232967344642237
```



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

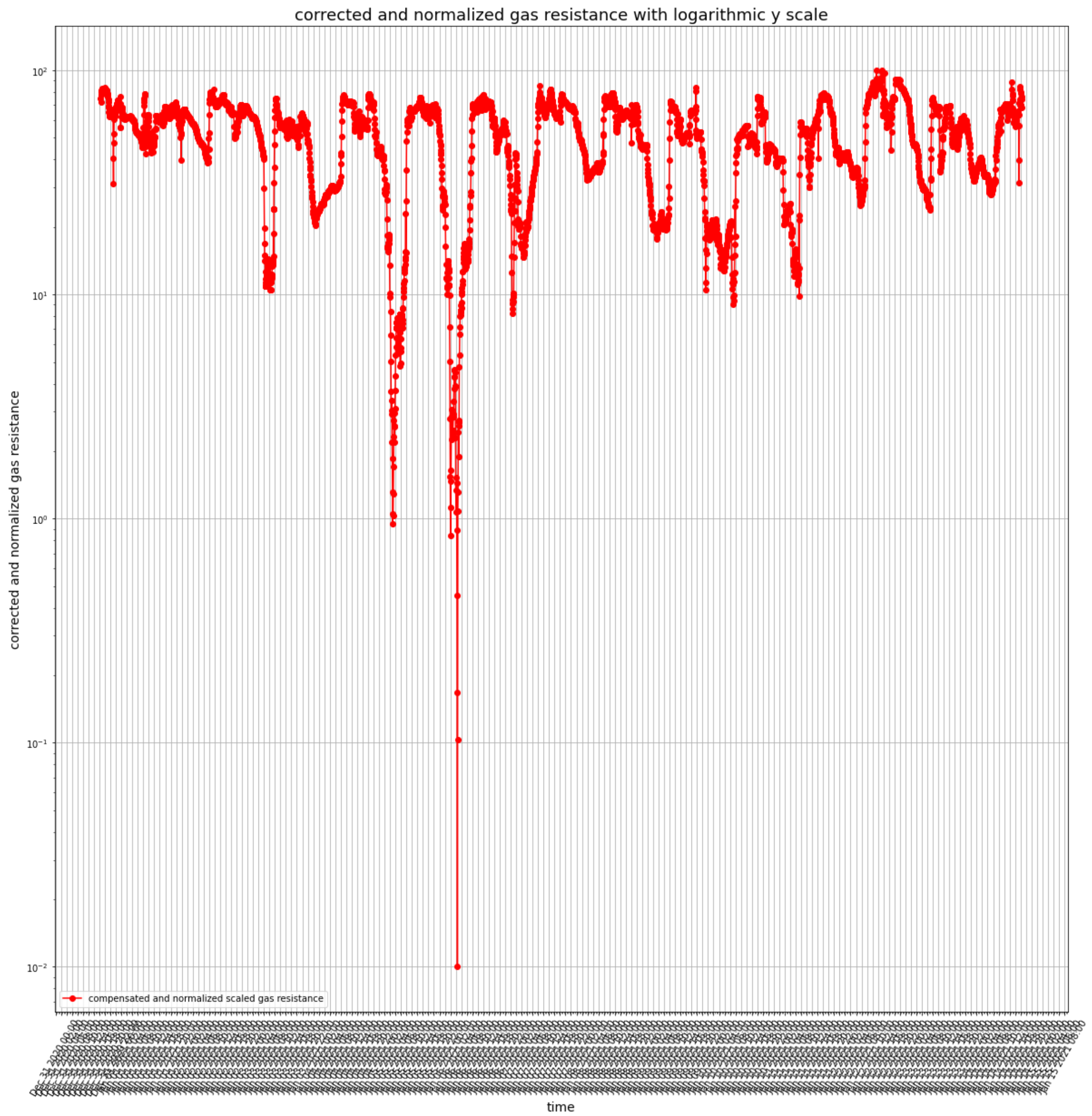
In [11]:

```python
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', color='red', label='comp

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()
```



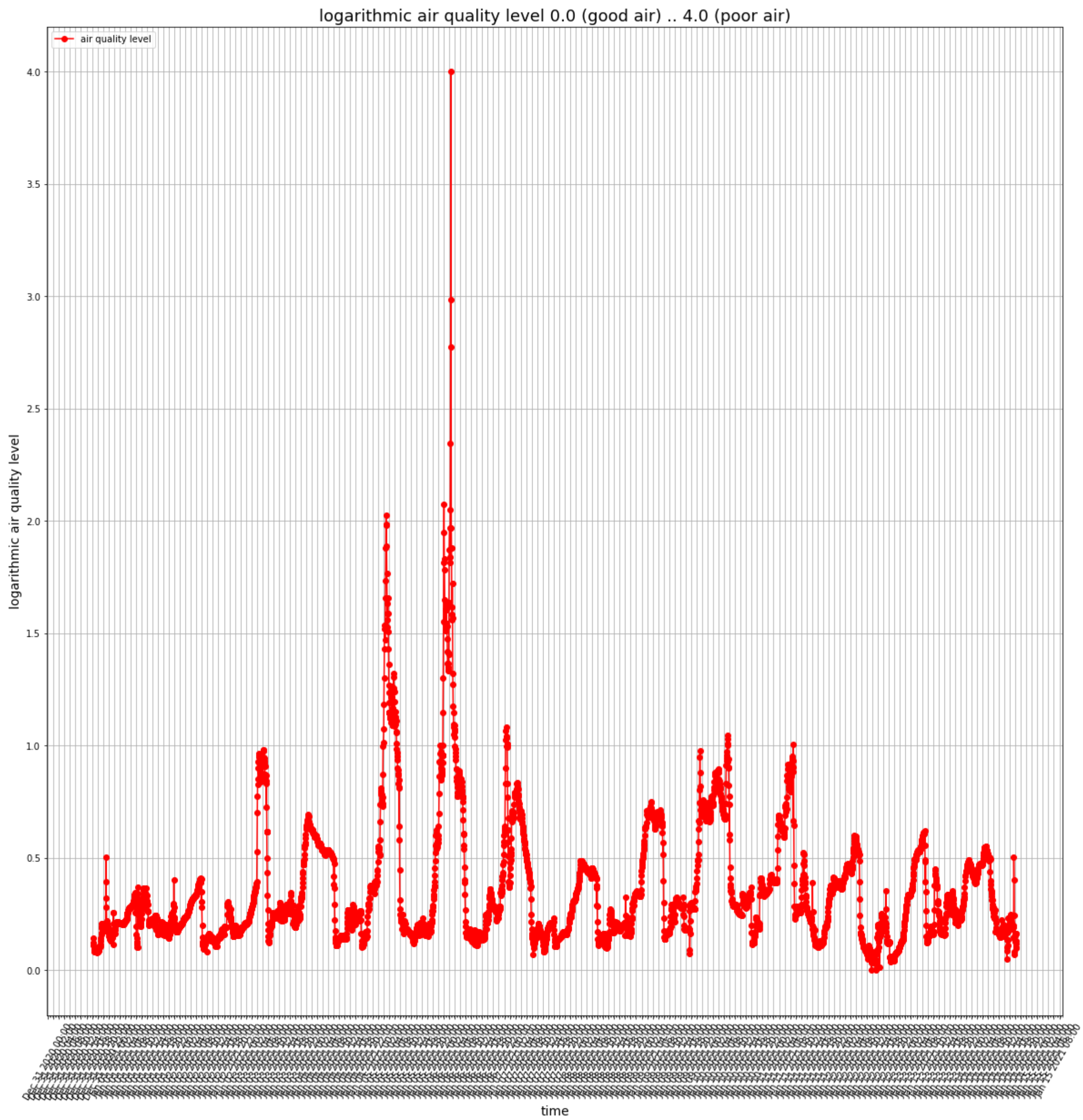
time series diagrams of compensated and normalized scaled gas resistance with logarithmic y scale 0.01 .. 100

In [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', color='red', label='comp

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)

In [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', color='red', label='

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!

In [14]:

```
print("\n\nR-squared (uncentered) of the multiple linear regression      = %11.2lf\n\n" % m
```

```
R-squared (uncentered) of the multiple linear regression      =       0.22
```

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

In [15]:

```
print("\nNumber of captured data points used for the MLR                    = %11d" % len(X

print("\nPlease enter the WebUI device parameter 'WEATHER|mlr_alpha'         = %11.3lf" % re
print("Please enter the WebUI device parameter 'WEATHER|mlr_beta'          = %11.3lf" % regr
print("Please enter the WebUI device parameter 'WEATHER|mlr_delta'         = %11.3lf" % regr

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


```

```
Number of captured data points used for the MLR                    =        4675

Please enter the WebUI device parameter 'WEATHER|mlr_alpha'        =    6603.489
Please enter the WebUI device parameter 'WEATHER|mlr_beta'         =  -37355.389
Please enter the WebUI device parameter 'WEATHER|mlr_delta'        =  243521.114


Please check whether current date and time are correct: 
2023-06-04 13:31:02.235573
```

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 ..