1.Montiamo Google drive cosi da poter caricare i file e usiamo un comando per visualizzare i file presenti nella cartella

In [37]:
```
import pandas as pd
from google.colab import drive
import os
import glob
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
from sklearn.cluster import KMeans
import matplotlib.dates as mdates
import folium
from matplotlib.colors import LinearSegmentedColormap
import nbformat
from nbconvert import HTMLExporter
from google.colab import drive, files
```

In [60]:
```
import warnings
from pandas.errors import SettingWithCopyWarning

# Disabilita i warning specifici
warnings.filterwarnings("ignore", category=SettingWithCopyWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Se vuoi disabilitare TUTTI i warning (non raccomandato)
# warnings.filterwarnings("ignore")
```

In [61]:
```
drive.mount('/content/drive')

# Let's see a list of files on the directory
!ls '/content/drive/MyDrive/Etna2018'
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", forc
e_remount=True).
'Etna2018_V2.xlsx - terremoti.csv'   'Etna2018.xlsx - descrizione dati.csv'
'Etna2018.xlsx - 3He-4He ratio.csv'  'Etna2018.xlsx - HCl_flux.csv'
'Etna2018.xlsx - clinometria.csv'    'Etna2018.xlsx - heat flux.csv'
'Etna2018.xlsx - CO2_flux.csv'       'Etna2018.xlsx - SO2 flux.csv'
'Etna2018.xlsx - CO2-SO2 ratio.csv'  'Etna2018.xlsx - tremore vulcanico.csv'
```

1.1Pulizia e correzioni degli errori in ciascun file uno dopo l'altro

Per ogni file faremo:

- Pulizia dati:correzione formate data, aggiunta nomi colonna mancanti, gestione valori nulli e outlier, formattazione spazi e virgolettati
- Visualizzazione grafica distribuzione dati dopo aver sistemato tutto
- Visualizzazione prime righe file dopo aver corretto

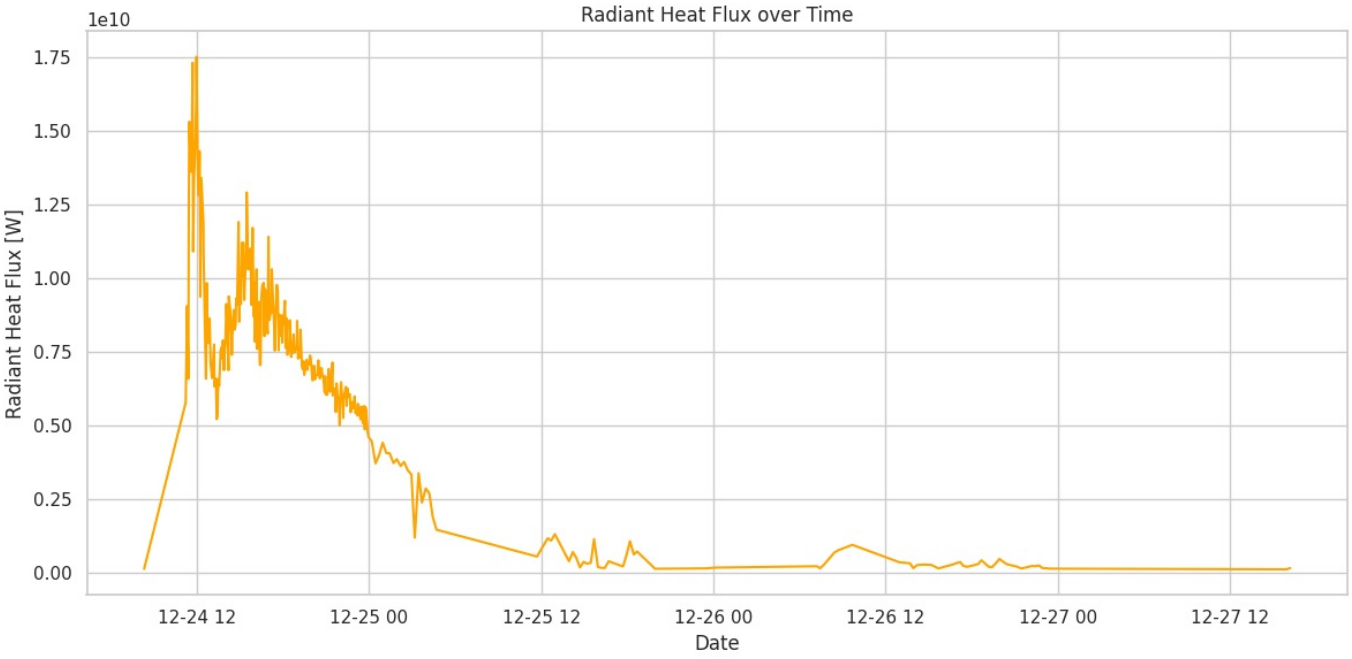
1.2 **Heat-Flux : **

- Rinominiamo colonna contenente le date in Datetime, visto che era senza nome
- Controlliamo il tipo di dati e lo convertiamo nel formato tempo corretto
- Verifichiamo se ci sono valori nulli
- Eliminiamo duplicati
- Filtriamo gli outlier, i valori diversi dal resto dei dati
- Rappresentazione grafica del flusso di calore, dopo aver pulito i dati

In [62]:
```
# First of all read the file on which analysis is based
df = pd.read_csv('/content/drive/MyDrive/Etna2018/Etna2018.xlsx - heat flux.csv')

# Rename time column in "Date"
df = df.rename(columns={df.columns[0]: 'date'})

# Save result in a new result
df.to_csv('heat_flux_with_dates.csv', index=False)
```

In [63]:
```
# 1. Read file and rename column
df = pd.read_csv('/content/drive/MyDrive/Etna2018/Etna2018.xlsx - heat flux.csv')
df = df.rename(columns={df.columns[0]: 'date'})

# 2. Clean data
df['date'] = pd.to_datetime(df['date'], format='%d/%m/%y %H.%M')
df['Radiant Heat Flux [W]'] = (
    df['Radiant Heat Flux [W]']
    .str.replace(',', '.', regex=False)
    .astype(float)
)

# 3. Manage datetime
df = df.sort_values('date').drop_duplicates('date')
df = df.set_index('date').asfreq('1min').reset_index()


# 4. Manage outliers
Q1 = df['Radiant Heat Flux [W]'].quantile(0.25)
Q3 = df['Radiant Heat Flux [W]'].quantile(0.75)
IQR = Q3 - Q1
df = df[(df['Radiant Heat Flux [W]'] >= (Q1 - 3*IQR)) & (df['Radiant Heat Flux [W]'] <= (Q3 + 3*IQR))]



# 5. Saved clean file
df.to_csv('heat_flux_cleaned.csv', index=False)
```

In [64]:
```
# Loading clean file
df = pd.read_csv('heat_flux_cleaned.csv')

# Let's be sure that datetime column is date
df['date'] = pd.to_datetime(df['date'])

# Set up datetime column as index for the graph
df = df.set_index('date')

# Creation of the graph
plt.figure(figsize=(12,6))
plt.plot(df.index, df['Radiant Heat Flux [W]'], color='orange')
plt.title('Radiant Heat Flux over Time')
plt.xlabel('Date')
plt.ylabel('Radiant Heat Flux [W]')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Visualizziamo il heat_flux pulito, le prime righe con le correzioni apportate

In [65]:
```
# Loading cleaned file
df = pd.read_csv('heat_flux_cleaned.csv')
```

```
# Let's be sure that 'date' column is in datetime format
df['date'] = pd.to_datetime(df['date'])

# Mostra le prime 10 righe per un rapido controllo
print("First 10 rows of cleaned Dataframe:")
print(df.head(10))

# Show some summary informations
print("\ngenerale information on Dataframe:")
print(df.info())

print("\ndescriptive statistics:")
print(df.describe())
```

```
First 10 rows of cleaned Dataframe:
          date         Radiant Heat Flux [W]
0 2018-12-24 08:19:00      1.410000e+08
1 2018-12-24 11:13:00      5.780000e+09
2 2018-12-24 11:19:00      9.050000e+09
3 2018-12-24 11:24:00      6.590000e+09
4 2018-12-24 11:26:00      1.250000e+10
5 2018-12-24 11:28:00      1.530000e+10
6 2018-12-24 11:34:00      1.360000e+10
7 2018-12-24 11:38:00      1.410000e+10
8 2018-12-24 11:42:00      1.730000e+10
9 2018-12-24 11:44:00      1.090000e+10

generale information on Dataframe:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 285 entries, 0 to 284
Data columns (total 2 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   date                   285 non-null    datetime64[ns]
 1   Radiant Heat Flux [W]  285 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 4.6 KB
None

descriptive statistics:
                   date              Radiant Heat Flux [W]
count                           285      2.850000e+02
mean   2018-12-25 02:18:09.473684224      6.089067e+09
min              2018-12-24 08:19:00      1.280000e+08
25%              2018-12-24 15:38:00      3.630000e+09
50%              2018-12-24 20:03:00      6.690000e+09
75%              2018-12-25 02:12:00      8.400000e+09
max              2018-12-27 16:12:00      1.750000e+10
std                              NaN      3.822569e+09
```

1.2 #**SO2flux:** Visualizziamo le ultime 5 righe e vediamo che ci sono gli stessi problemi. Sistemiamo il format, trasformando da format letterali a numerici (da feb a 02), aggiustiamo i separatori e la colonna date e rappresentiamo graficamente di nuovo

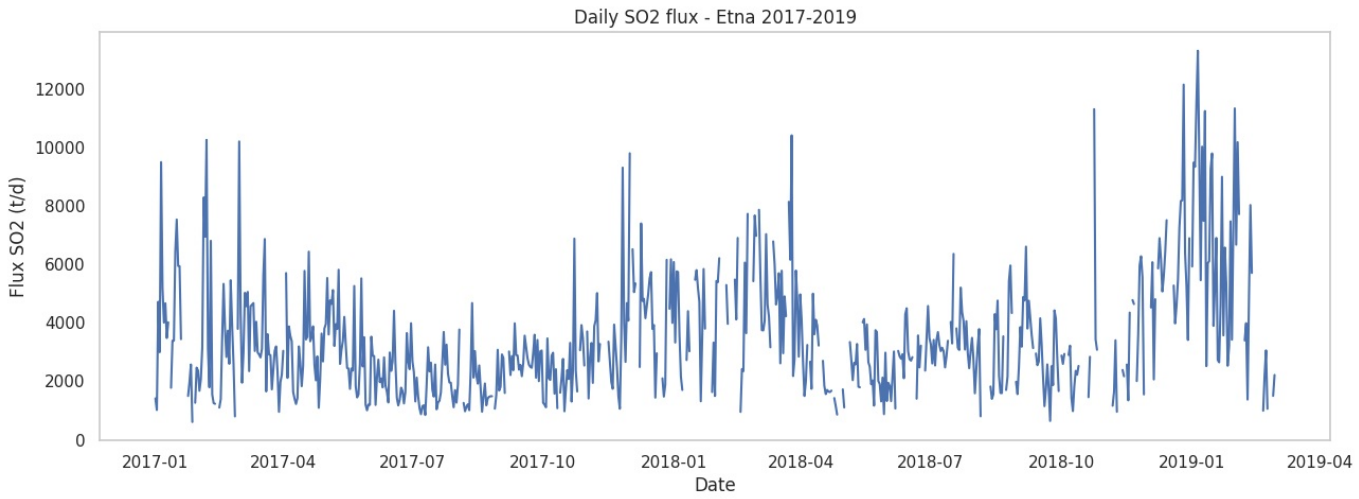
In [66]:
```
# Loading SO2 file
file_path = '/content/drive/MyDrive/Etna2018/Etna2018.xlsx - SO2 flux.csv'

# Read file with optimized parameters
so2flux = pd.read_csv(
    file_path,
    skiprows=1,
    decimal=',',
    thousands='.',
    parse_dates=['Date'],
    dayfirst=True
)

# Cleaning numeric column
so2flux['daily SO2 flux (t/d)'] = pd.to_numeric(
    so2flux['daily SO2 flux (t/d)'].astype(str)
    .str.replace('[^\d,]', '', regex=True)
    .str.replace(',', '.', regex=False),
    errors='coerce'
)

# Set time index
so2flux.set_index('Date', inplace=True)

# Print some rows of cleaned dataset
print(f"Number of SO2 flux observations: {len(so2flux)}")
print("\nlast 5 records")
print(so2flux.tail())
```

```
Number of SO2 flux observations: 671

last 5 records
          daily SO2 flux (t/d)
Date
22-feb-19         305576
23-feb-19         105591
25-feb-19         235755
27-feb-19         149822
28-feb-19         221421
```

Sulla stessa variabile il flusso/emissione di anidride solforosa, rappresento due grafici per visualizzare i dati e la distribuzione

In [67]:
```python
# 1. Reading file
df = pd.read_csv('/content/drive/MyDrive/Etna2018/Etna2018.xlsx - SO2 flux.csv', skiprows=1)

# 2. Cleaning datetime
month_map = {
    'gen': 'Jan', 'feb': 'Feb', 'mar': 'Mar', 'apr': 'Apr',
    'mag': 'May', 'giu': 'Jun', 'lug': 'Jul', 'ago': 'Aug',
    'set': 'Sep', 'ott': 'Oct', 'nov': 'Nov', 'dic': 'Dec'
}

df['Date'] = df['Date'].replace(month_map, regex=True)
df['Date'] = pd.to_datetime(df['Date'], format='%d-%b-%y')

# 3. Converting numeric values
df['daily SO2 flux (t/d)'] = (
    df['daily SO2 flux (t/d)']
    .str.replace(',', '.')
    .str.replace('"', '')
    .astype(float)
)

# 4. Managing missing date
full_range = pd.date_range(
    start=df['Date'].min(),
    end=df['Date'].max(),
    freq='D'
)



df = df.set_index('Date').reindex(full_range).reset_index()
df = df.rename(columns={
    'index': 'date',
    'daily SO2 flux (t/d)': 'SO2_flux_t_d'
})


# 7. Saving cleaned file
df.to_csv('SO2_flux_cleaned.csv', index=False)
```

In [68]:
```python
#Visualize graph
plt.figure(figsize=(15,5))
plt.plot(df['date'], df['SO2_flux_t_d'])
plt.title('Daily SO2 flux - Etna 2017-2019')
plt.xlabel('Date')
plt.ylabel('Flux SO2 (t/d)')
plt.grid()
plt.show()
```



In [69]:
```
# Loading cleaned file
df = pd.read_csv('SO2_flux_cleaned.csv')

# Showing first 10 column
print("First 10 rows of cleaned dataset")
print(df.head(10))
```

```
First 10 rows of cleaned dataset
      date     SO2_flux_t_d
0  2017-01-01     1411.79
1  2017-01-02     1016.61
2  2017-01-03     4720.25
3  2017-01-04     2995.22
4  2017-01-05     9504.67
5  2017-01-06     5194.17
6  2017-01-07     4009.46
7  2017-01-08     4670.08
8  2017-01-09     3480.77
9  2017-01-10     4016.83
```

1.3 CO2 flux

Dalla pulizia dati iniziale possiamo notare ci sono meno outlier, infatti possiamo vedere molti FALSE in corrispondenza della voce outlier.

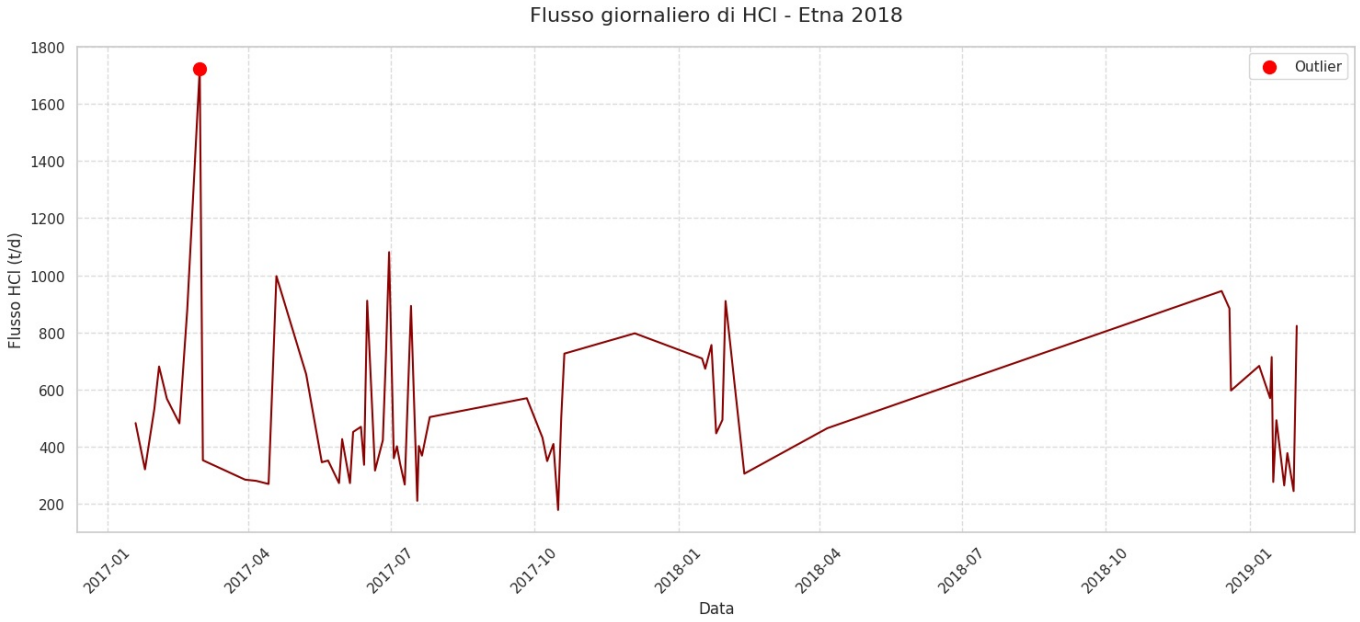
In [70]:
```
def clean_hcl_flux(input_file, output_file):
    """

    Args:
        input_file (str): Input file path
        output_file (str): Output file path
    """

    # 1. Loading data skipping firs useless rows
    df = pd.read_csv(input_file, skiprows=1)

    # 2. Verifying starting structure
    print("\nStrarting structure of the dataset:")
    print(df.head())
    print(f"\nNumber of original rows: {len(df)}")

    # 3. Cleaning columns
    df.columns = df.columns.str.strip()

    # 4. Managing missing values
    print("\nMissing values before the cleaning")
    print(df.isnull().sum())

    # If necessary removing rows with missing values
    df = df.dropna()

    # 5. Converting and uniforming date
    try:
        df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
    except:
        # Trying another format
        df['Date'] = pd.to_datetime(df['Date'], format='%m/%d/%Y')

    # 6. Sort by datetime
    df = df.sort_values('Date')

    # 7. Check and manage outlier
    Q1 = df['daily HCl flux (t/d)'].quantile(0.25)
    Q3 = df['daily HCl flux (t/d)'].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identifying outlier
    df['outlier'] = ~df['daily HCl flux (t/d)'].between(lower_bound, upper_bound)

    print(f"\nNumber of potential outlier identified: {df['outlier'].sum()}")
    print("\nOutlier identified:")
    print(df[df['outlier'] == True])

    # 8. CHeck duplicate values
    duplicates = df.duplicated(subset=['Date'], keep=False)
    if duplicates.any():
        print(f"\nFound {duplicates.sum()} duplicated. Verify:")
        print(df[duplicates])
```

```
        df = df.drop_duplicates(subset=['Date'], keep='first')

    # 9. Verifying non numeric values
    non_numeric = pd.to_numeric(df['daily HCl flux (t/d)'], errors='coerce').isna()
    if non_numeric.any():
        print(f"\nfound {non_numeric.sum()} non numeric values. Verify:")
        print(df[non_numeric])
        # Converting in numeric and settling as "Nan" non convertibles ones
        df['daily HCl flux (t/d)'] = pd.to_numeric(df['daily HCl flux (t/d)'], errors='coerce')

    # 10. Verifying negative values
    negatives = df['daily HCl flux (t/d)'] < 0
    if negatives.any():
        print(f"\nTrovati {negatives.sum()} valori negativi. Verifica:")
        print(df[negatives])

        df.loc[negatives, 'daily HCl flux (t/d)'] = np.nan

    # 11. Removing rows with missing values eventually
    df = df.dropna()

    # 12. Final format
    df['Date'] = df['Date'].dt.strftime('%Y-%m-%d')

    # 13. Saving cleaned file
    df.to_csv(output_file, index=False)

    # 14. Final report
    print("\ncleaning successfully completed!")
    print(f"Number of rows in cleaned file: {len(df)}")
    print(f"File saved as: {output_file}")

    return df

# Using script
if __name__ == "__main__":
    input_file = "/content/drive/MyDrive/Etna2018/Etna2018.xlsx - HCl_flux.csv"
    output_file = "HCl_flux_clean.csv"

    cleaned_data = clean_hcl_flux(input_file, output_file)

    # View first 5 rows
    print("\nfirst 5 rows of cleaned dataset:")
    print(cleaned_data.head())
```

```
Strarting structure of the dataset:
     Date     daily HCl flux (t/d)
0  19/1/2017           483
1  25/1/2017           321
2  31/1/2017           533
3   3/2/2017           681
4   8/2/2017           568

Number of original rows: 64

Missing values before the cleaning
Date                    0
daily HCl flux (t/d)    0
dtype: int64

Number of potential outlier identified: 1

Outlier identified:
     Date     daily HCl flux (t/d)  outlier
7 2017-03-01          1723           True

cleaning successfully completed!
Number of rows in cleaned file: 64
File saved as: HCl_flux_clean.csv

first 5 rows of cleaned dataset:
      Date     daily HCl flux (t/d)  outlier
0  2017-01-19           483           False
1  2017-01-25           321           False
2  2017-01-31           533           False
3  2017-02-03           681           False
4  2017-02-08           568           False
```

In [71]:
```
def plot_hcl_flux(df):
    """
    Function to show HCI flux during time

    Args:
```

```
        df (DataFrame): DataFrame containg cleaned data
    """
    # Converting the column in Datetime for plotting
    df['Date'] = pd.to_datetime(df['Date'])

    # Creating the figure
    plt.figure(figsize=(15, 7))

    # Line graph of flux during the time
    sns.lineplot(data=df, x='Date', y='daily HCl flux (t/d)',
                 color='darkred', linewidth=1.5)

    # Showing outlier if present in columns
    if 'outlier' in df.columns:
        outliers = df[df['outlier'] == True]
        plt.scatter(outliers['Date'], outliers['daily HCl flux (t/d)'],
                    color='red', s=100, label='Outlier', zorder=5)

    # Formatting graph
    plt.title('Flusso giornaliero di HCl - Etna 2018', fontsize=16, pad=20)
    plt.xlabel('Data', fontsize=12)
    plt.ylabel('Flusso HCl (t/d)', fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.7)

    # Improve readibality of labels
    plt.xticks(rotation=45)


    if 'outlier' in df.columns:
        plt.legend()

    plt.tight_layout()
    plt.show()

# Usage after clean the data
if __name__ == "__main__":
    # Loading cleaned data
    cleaned_data = pd.read_csv("HCl_flux_clean.csv")

    # Generating graph
    plot_hcl_flux(cleaned_data)
```



In [72]:

```
def display_clean_data_head(cleaned_data):
    """
    Function to visualize first 5 rows of cleaned DataFrame

    Args:
        cleaned_data (DataFrame): DataFrame clened
    """
    # Creating a copy to avoid modification of the original one
    display_df = cleaned_data.head().copy()

    # Format improved for the visualization
    pd.set_option('display.max_columns', None)
    pd.set_option('display.width', 1000)
    pd.set_option('display.colheader_justify', 'center')

    print("\n" + "="*70)
    print("First 5 rows of cleaned dataframe".center(70))
```

```
    print("="*70 + "\n")

    # View DataFrame
    display(display_df.style
            .set_properties(**{'background-color': '#f7f7f7',
                               'color': '#333333',
                               'border': '1px solid #cccccc'})
            .format({'daily HCl flux (t/d)': '{:.2f}'})
            .set_table_styles([{'selector': 'th',
                                'props': [('background-color', '#4a6baf'),
                                          ('color', 'white'),
                                          ('font-weight', 'bold')]}]))


    print("\n" + "-"*70)
    print(f"Total number of rows of cleaned dataframe: {len(cleaned_data)}")
    print(f"Number of columns: {len(cleaned_data.columns)}")
    print("Available columns:", list(cleaned_data.columns))

    if 'outlier' in cleaned_data.columns:
        total_outliers = cleaned_data['outlier'].sum()
        print(f"\nTotal number of identified outliers: {total_outliers}")



if __name__ == "__main__":
    input_file = "/content/drive/MyDrive/Etna2018/Etna2018.xlsx - HCl_flux.csv"
    output_file = "HCl_flux_clean.csv"

    # Let's execute cleaning of data
    cleaned_data = clean_hcl_flux(input_file, output_file)

    # Visualization of first 5 rows with improved formatting
    display_clean_data_head(cleaned_data)
```

```
Strarting structure of the dataset:
     Date     daily HCl flux (t/d)
0  19/1/2017           483
1  25/1/2017           321
2  31/1/2017           533
3   3/2/2017           681
4   8/2/2017           568

Number of original rows: 64

Missing values before the cleaning
Date                   0
daily HCl flux (t/d)    0
dtype: int64

Number of potential outlier identified: 1

Outlier identified:
     Date     daily HCl flux (t/d)  outlier
7 2017-03-01          1723           True

cleaning successfully completed!
Number of rows in cleaned file: 64
File saved as: HCl_flux_clean.csv

======================================================================
                  First 5 rows of cleaned dataframe
======================================================================
```

	Date	daily HCl flux (t/d)	outlier
0	2017-01-19	483.00	False
1	2017-01-25	321.00	False
2	2017-01-31	533.00	False
3	2017-02-03	681.00	False
4	2017-02-08	568.00	False

```
----------------------------------------------------------------------
Total number of rows of cleaned dataframe: 64
Number of columns: 3
Available columns: ['Date', 'daily HCl flux (t/d)', 'outlier']

Total number of identified outliers: 1
```

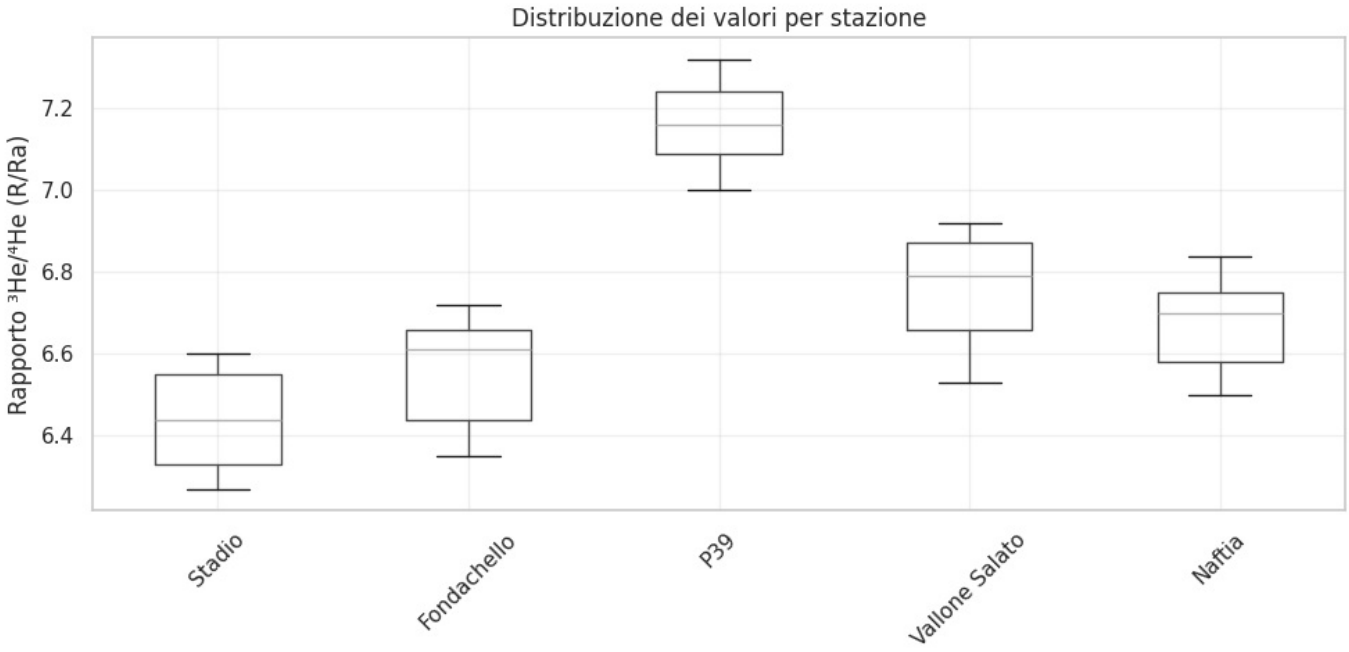
1.4 3HE-4HE_ratio:

direttamente puliamo il file, e rappresentiamo i box plot di ogni variabile, boxplot comparativi facendo il confronto tra loro

In [73]:
```python
def clean_he_ratio(input_file, output_file):

    # Loading with managing problematics rows
    df = pd.read_csv(input_file, skiprows=1)

    # Remove fully empty rows
    df = df[~df['Date'].str.contains('n.a.:', na=False)]
    df = df.dropna(how='all')

    # Managing missing values
    df.replace('n.a.', np.nan, inplace=True)

    # Converting datetime
    df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y', errors='coerce')


    df = df.dropna(subset=['Date'])

    # Converting numbers
    for col in df.columns[1:]:
        if df[col].dtype == object:
            df[col] = df[col].astype(str).str.replace(',', '.').astype(float)

    # Sorting
    df = df.sort_values('Date')

    # Verification of duplicates
    if df.duplicated(subset=['Date']).any():
        df = df.drop_duplicates(subset=['Date'], keep='first')

    # Saving
    df.to_csv(output_file, index=False)
    return df

if __name__ == "__main__":

    cleaned_data = clean_he_ratio('/content/drive/MyDrive/Etna2018/Etna2018.xlsx - 3He-4He ratio.csv', 'cleaned_
    print("Dati puliti salvati in 'cleaned_he_ratio.csv'")
    print(cleaned_data.head())
```

```
Dati puliti salvati in 'cleaned_he_ratio.csv'
     Date     Stadio  Fondachello   P39  Vallone Salato  Naftia
0 2017-01-13   6.32      6.44      7.10       6.58        6.58
1 2017-02-06   6.31      6.42      7.09       6.57        6.57
2 2017-03-02   6.30      6.38      7.04       6.56        6.56
3 2017-03-24   6.27      6.35      7.00       6.53        6.50
4 2017-04-12   6.30      6.40      7.03       6.55        6.54
```

In [74]:
```python
# Loading cleaned data
df = pd.read_csv('cleaned_he_ratio.csv', parse_dates=['Date'])

# Extraction of station names
stations = df.columns[1:].tolist()

# Boxplot
plt.figure(figsize=(10, 5))
df.boxplot(column=stations)
plt.title('Distribuzione dei valori per stazione')
plt.ylabel('Rapporto ³He/He (R/Ra)')
plt.xticks(rotation=45)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



1.5 CO2-SO2 flux:

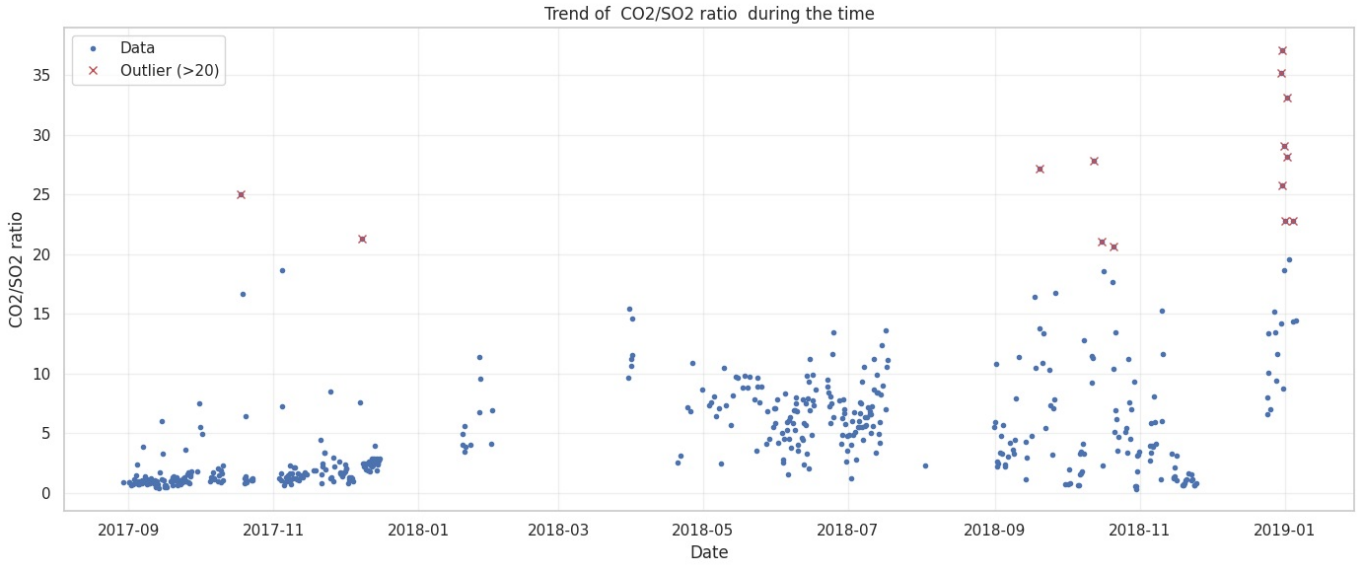
facciamo la stessa cosa qui concentrandoci su una visualizzazione grafica che evidenzi gli outlier

In [75]:
```python
def clean_co2_so2_data(input_file, output_file):

    # 1. Loading data
    try:
        df = pd.read_csv(input_file, skiprows=1)
    except Exception as e:
        print(f"Error on loading file: {e}")
        return None

    # 2. Starting cleaning
    df.columns = df.columns.str.strip()

    # 3. Converting numeric values
    df['CO2/SO2'] = (df['CO2/SO2']
                     .astype(str)
                     .str.replace(',', '.')
                     .replace('nan', np.nan)
                     .astype(float))

    # 4. Converting datetime
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
    df = df.dropna(subset=['Date'])


    df = df.drop_duplicates(subset=['Date', 'CO2/SO2'])


    df['Is_Outlier'] = df['CO2/SO2'] > 20


    df['Year'] = df['Date'].dt.year
    df['Month'] = df['Date'].dt.month
    df['Day'] = df['Date'].dt.day
    df['Date_only'] = df['Date'].dt.date


    df = df.sort_values('Date')

    # 9. Saving
    df.to_csv(output_file, index=False)


    print(f"Cleaning completed. Data saved in {output_file}")
    print(f"\nStatistics:")
    print(f"- Total rows: {len(df)}")
    print(f"- Outliers (>20): {df['Is_Outlier'].sum()}")
    print(f"- Cover period: {df['Date'].min().date()} - {df['Date'].max().date()}")
    print(f"- Mean value: {df['CO2/SO2'].mean():.2f} ± {df['CO2/SO2'].std():.2f}")

    return df
```

```
# Usage of script
if __name__ == "__main__":
    input_path = "/content/drive/MyDrive/Etna2018/Etna2018.xlsx - CO2-SO2 ratio.csv"
    output_path = "cleaned_CO2_SO2_ratio.csv"

    cleaned_data = clean_co2_so2_data(input_path, output_path)

    # Fast visualization of results
    if cleaned_data is not None:
        print("\nPreview of cleaned data:")
        print(cleaned_data.head())
```

```
Cleaning completed. Data saved in cleaned_CO2_SO2_ratio.csv

Statistics:
- Total rows: 498
- Outliers (>20): 14
- Cover period: 2017-08-29 - 2019-01-05
- Mean value: 5.19 ± 5.43

Preview of cleaned data:
          Date         CO2/SO2  Is_Outlier  Year  Month  Day  Date_only
0 2017-08-29 13:05:00   0.89       False    2017    8    29   2017-08-29
1 2017-09-01 01:05:00   0.91       False    2017    9     1   2017-09-01
2 2017-09-01 07:05:00   0.85       False    2017    9     1   2017-09-01
3 2017-09-02 01:05:00   0.60       False    2017    9     2   2017-09-02
4 2017-09-02 19:05:00   0.74       False    2017    9     2   2017-09-02
```

In [76]:
```
def plot_results(cleaned_file):
    df = pd.read_csv(cleaned_file, parse_dates=['Date'])

    plt.figure(figsize=(14, 6))
    plt.plot(df['Date'], df['CO2/SO2'], 'o', markersize=3, label='Data')
    plt.plot(df[df['Is_Outlier']]['Date'],
             df[df['Is_Outlier']]['CO2/SO2'],
             'rx', label='Outlier (>20)')
    plt.title('Trend of  CO2/SO2 ratio  during the time')
    plt.xlabel('Date')
    plt.ylabel('CO2/SO2 ratio')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig('co2_so2_trend.png', dpi=300)
    plt.show()

plot_results('cleaned_CO2_SO2_ratio.csv')
```



1.6 HCI_flux:

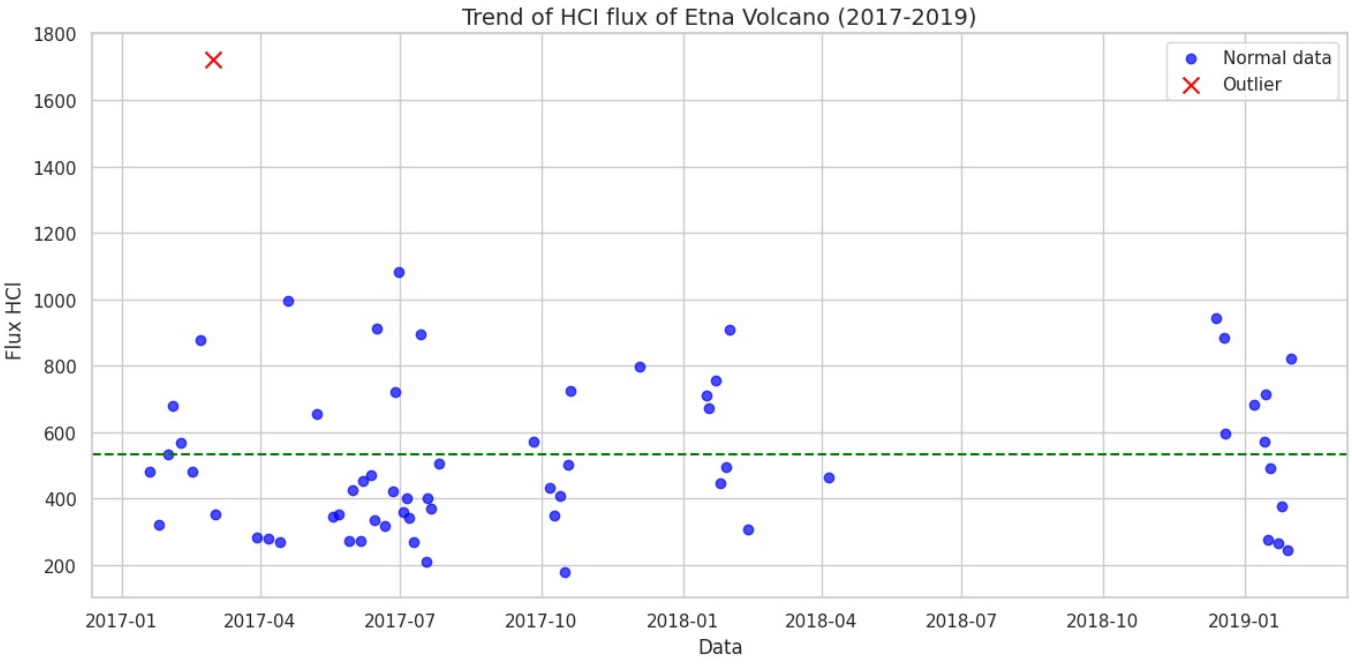
puliamo il file, rinominiamo le colonne, aggiustiamo il formato e poi nello stesso script presentiamo i risultati grafici

In [77]:
```
def clean_hcl_flux(input_file, output_file):


    # 1. Loading data
    try:
```

```
        df = pd.read_csv(input_file, skiprows=1)
    except Exception as e:
        print(f"Error during loading of the file: {e}")
        return None

    # 2. Verifying structure of data
    if len(df.columns) != 2:
        print("Warning, structure not correct.")
        return None

    # 3. Renaming columns
    df.columns = ['Date', 'HCl_flux_td']


    df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y', errors='coerce')


    initial_count = len(df)
    df = df.dropna(subset=['Date', 'HCl_flux_td'])
    if len(df) < initial_count:
        print(f"Rimosse {initial_count - len(df)} rows with missing values")


    df['HCl_flux_td'] = pd.to_numeric(df['HCl_flux_td'], errors='coerce')
    df = df.dropna(subset=['HCl_flux_td'])


    Q1 = df['HCl_flux_td'].quantile(0.25)
    Q3 = df['HCl_flux_td'].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    df['Is_Outlier'] = (df['HCl_flux_td'] < lower_bound) | (df['HCl_flux_td'] > upper_bound)


    df['Year'] = df['Date'].dt.year
    df['Month'] = df['Date'].dt.month
    df['Day'] = df['Date'].dt.day
    df['DayOfYear'] = df['Date'].dt.dayofyear


    df = df.sort_values('Date')


    df = df.reset_index(drop=True)

    # 11. Saving results
    df.to_csv(output_file, index=False)

    # 12. Final report summary
    print("\n" + "="*50)
    print("HCI FLUX data cleaning completed")
    print("="*50)
    print(f"\nOutput filest: {output_file}")
    print(f"Covered period: {df['Date'].min().date()} - {df['Date'].max().date()}")
    print(f"Total rows: {len(df)}")
    print(f"outlier: {df['Is_Outlier'].sum()}")
    print(f"\nStatistics flux HCl:")
    print(f"- Mean: {df['HCl_flux_td'].mean():.1f} t/d")
    print(f"- Median: {df['HCl_flux_td'].median():.1f} t/d")
    print(f"- Min: {df['HCl_flux_td'].min():.1f} t/d")
    print(f"- Max: {df['HCl_flux_td'].max():.1f} t/d")
    print(f"- Standard deviation: {df['HCl_flux_td'].std():.1f} t/d")

    print("\n5 first rows:")
    print(df.head())

    return df

def plot_hcl_trend(cleaned_file):


    df = pd.read_csv(cleaned_file, parse_dates=['Date'])

    plt.figure(figsize=(12, 6))
    sns.set_style("whitegrid")

    # Plot normal data
    plt.scatter(df[~df['Is_Outlier']]['Date'],
                df[~df['Is_Outlier']]['HCl_flux_td'],
                color='blue', label='Normal data', alpha=0.7)
```

```
    # Plot outlier
    plt.scatter(df[df['Is_Outlier']]['Date'],
                df[df['Is_Outlier']]['HCl_flux_td'],
                color='red', label='Outlier', marker='x', s=100)

    plt.title('Trend of HCI flux of Etna Volcano (2017-2019)', fontsize=14)
    plt.xlabel('Data', fontsize=12)
    plt.ylabel('Flux HCl ', fontsize=12)
    plt.legend()

    # Linea della media
    mean_flux = df['HCl_flux_td'].mean()
    plt.axhline(y=mean_flux, color='green', linestyle='--',
                label=f'Mean ({mean_flux:.1f} t/d)')

    plt.tight_layout()
    plt.savefig('hcl_flux_trend.png', dpi=300)
    plt.show()

# Esecuzione
if __name__ == "__main__":
    input_path = "/content/drive/MyDrive/Etna2018/Etna2018.xlsx - HCl_flux.csv"
    output_path = "cleaned_HCl_flux.csv"

    cleaned_data = clean_hcl_flux(input_path, output_path)

    if cleaned_data is not None:
        plot_hcl_trend(output_path)
```

```
==================================================
HCI FLUX data cleaning completed
==================================================

Output filest: cleaned_HCl_flux.csv
Covered period: 2017-01-19 - 2019-01-31
Total rows: 64
outlier: 1

Statistics flux HCl:
- Mean: 535.1 t/d
- Median: 467.5 t/d
- Min: 179.0 t/d
- Max: 1723.0 t/d
- Standard deviation: 269.7 t/d

5 first rows:
     Date     HCl_flux_td  Is_Outlier  Year  Month  Day  DayOfYear
0 2017-01-19      483         False    2017    1    19      19
1 2017-01-25      321         False    2017    1    25      25
2 2017-01-31      533         False    2017    1    31      31
3 2017-02-03      681         False    2017    2     3      34
4 2017-02-08      568         False    2017    2     8      39
```



1.7 Clinometria:

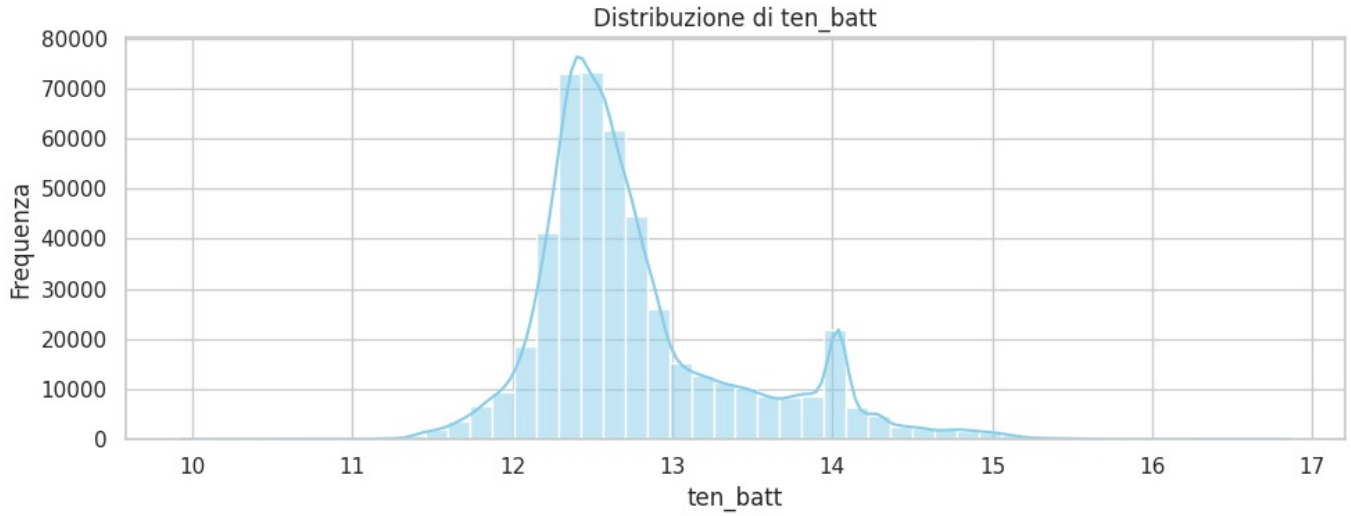
in clinometria si evidenziano leggermente problemi in più, in quanto le colonne non sono separate, i

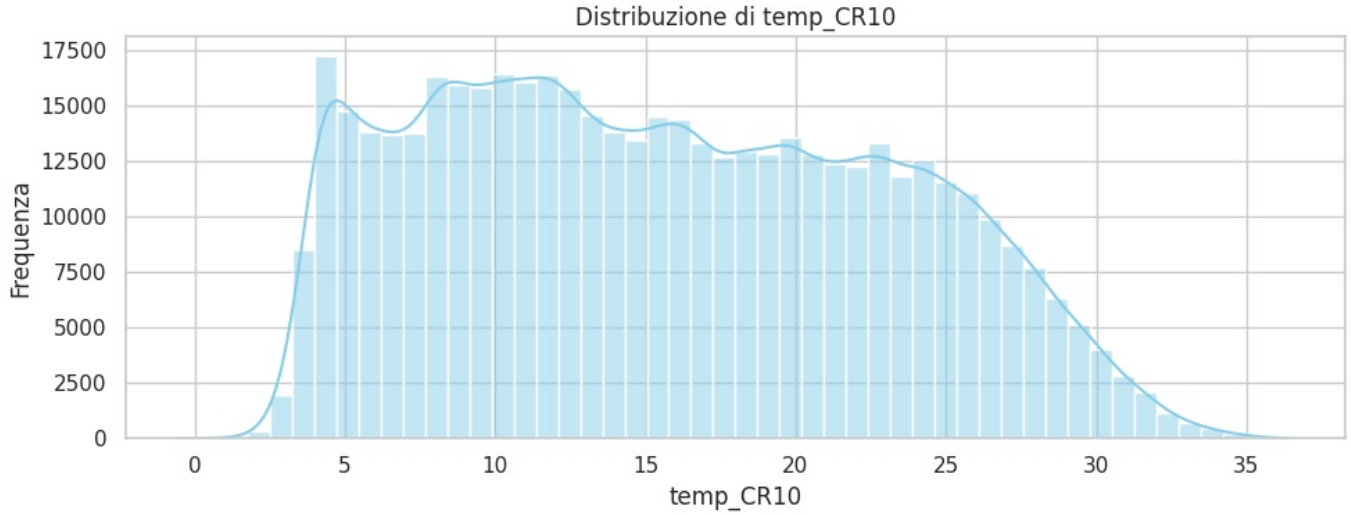
titoli delle colonne non non sono etichettati in caselle diverse, ma sono separati da virgole e in più bisogna adattare le altre operazioni di pulizia, la colonna time al solito, eliminare colonne vuote e salvare il risultato; graficamente rappresento le distribuzioni cosi da capire la differenze tra una colonna e l'altra

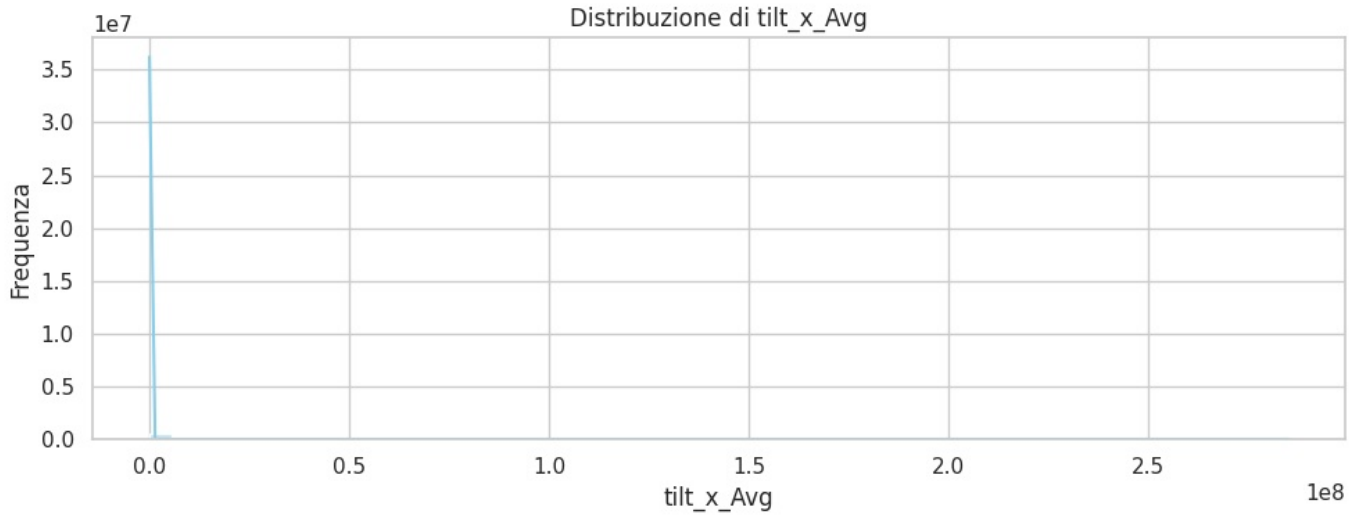
In [78]:
```
# Loading file
raw_df = pd.read_csv("/content/drive/MyDrive/Etna2018/Etna2018.xlsx - clinometria.csv", header=None)


split_df = raw_df[0].str.split(",", expand=True)


split_df.columns = split_df.iloc[0]
split_df = split_df.drop(index=0)


split_df["time"] = pd.to_datetime(split_df["time"], errors="coerce")


for col in split_df.columns:
    if col != "time":
        split_df[col] = pd.to_numeric(split_df[col], errors="coerce")


clean_df = split_df.dropna(subset=["time"]).dropna(how='all')


clean_df.reset_index(drop=True, inplace=True)

# Save result
clean_df.to_csv("clinometria_clean.csv", index=False)

print("Pulizia completata. File salvato come 'clinometria_clean.csv'")
```

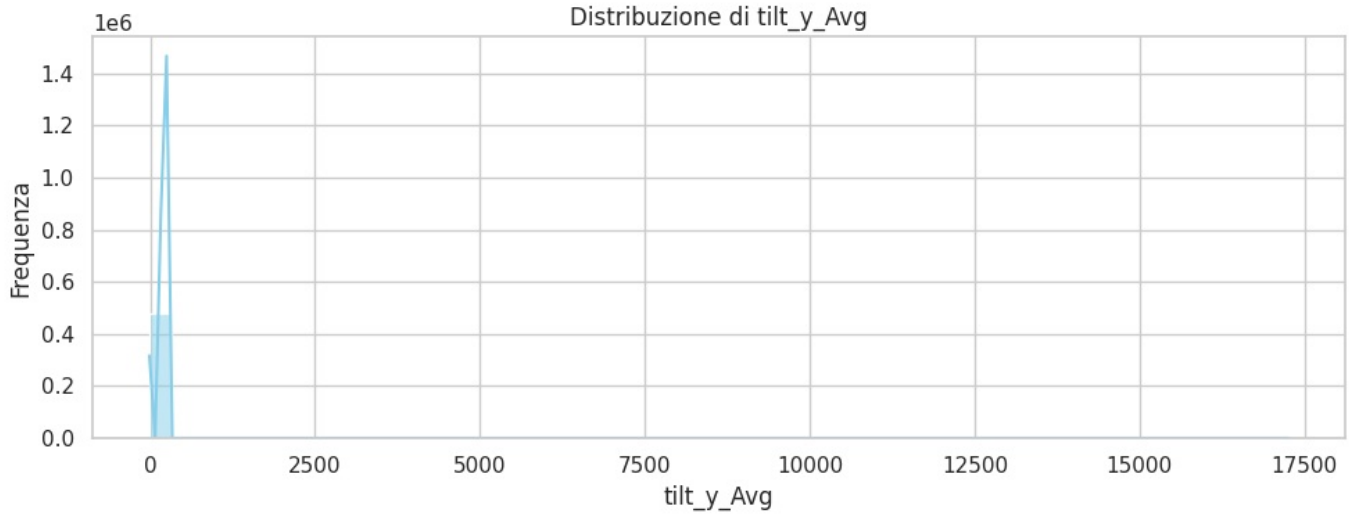
```
Pulizia completata. File salvato come 'clinometria_clean.csv'
```

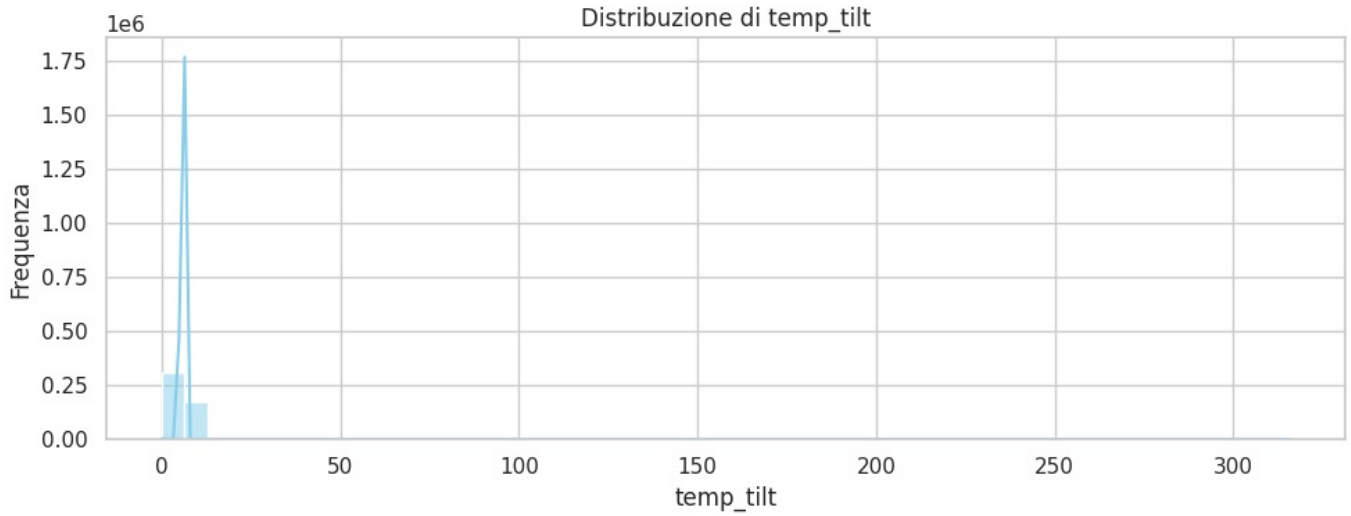
In [79]:
```
# Loading cleaned file
df = pd.read_csv("clinometria_clean.csv")


df["time"] = pd.to_datetime(df["time"], errors="coerce")

# Setting graphs style
sns.set(style="whitegrid")


numeric_cols = df.select_dtypes(include=["float64", "int64"]).columns

for col in numeric_cols:
    plt.figure(figsize=(10, 4))
    sns.histplot(df[col].dropna(), kde=True, bins=50, color="skyblue")
    plt.title(f"Distribuzione di {col}")
    plt.xlabel(col)
    plt.ylabel("Frequenza")
    plt.tight_layout()
    plt.show()

#time series
for col in numeric_cols:
    plt.figure(figsize=(14, 4))
    plt.plot(df["time"], df[col], label=col, color="dodgerblue", linewidth=0.5)
    plt.title(f"Andamento temporale di {col}")
    plt.xlabel("Tempo")
    plt.ylabel(col)
    plt.tight_layout()
    plt.legend()
    plt.show()
```

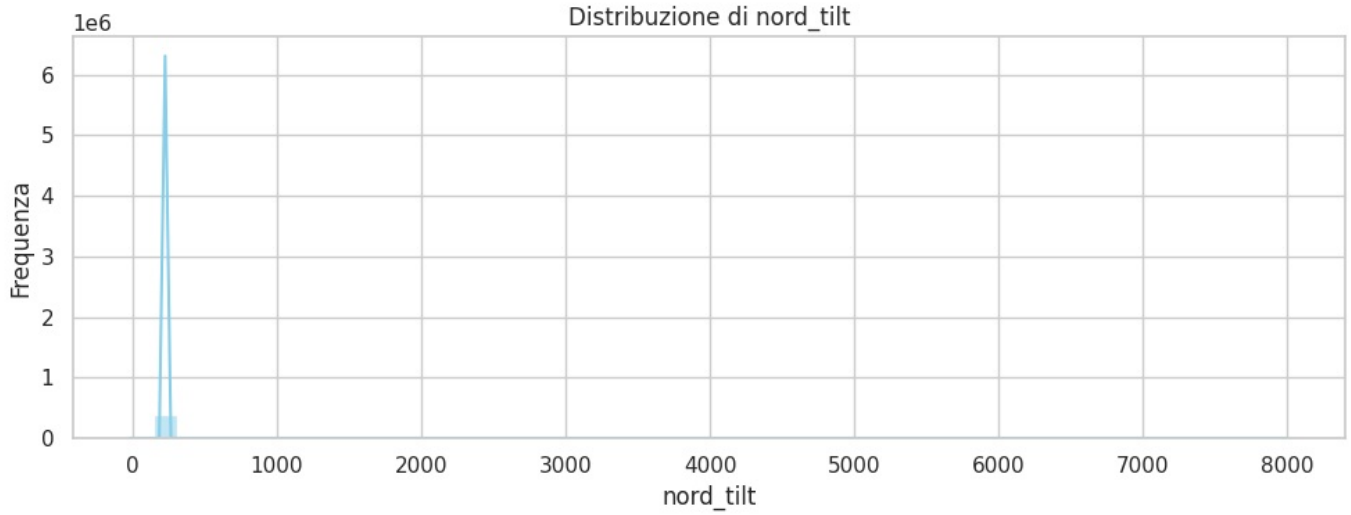

Distribuzione di ten_batt
Frequenza
ten_batt

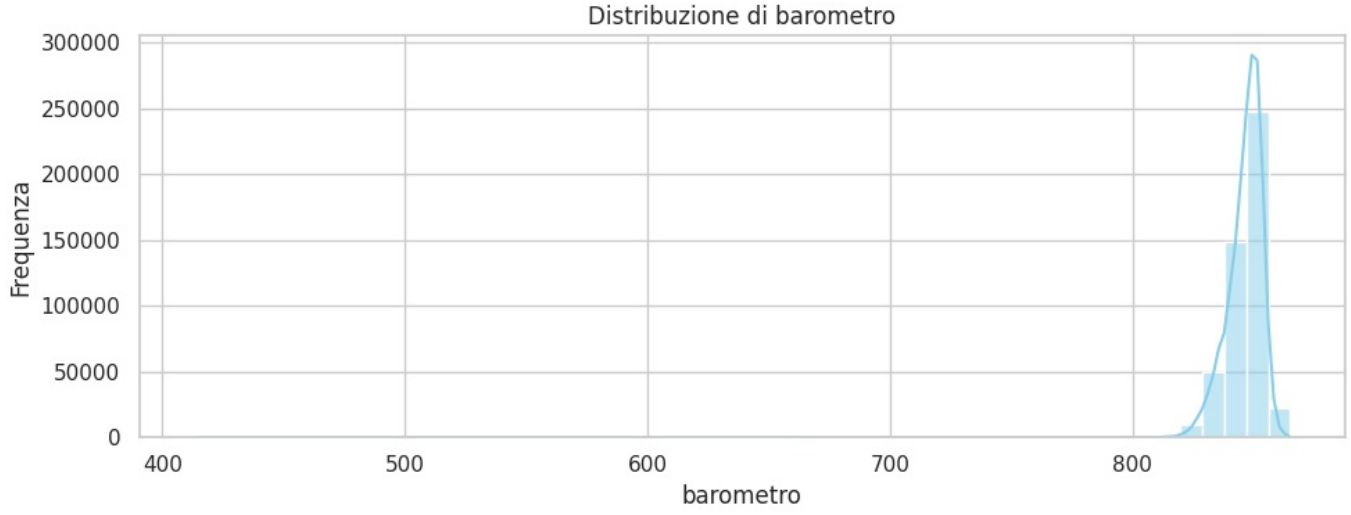

Distribuzione di temp_CR10
Frequenza
temp_CR10

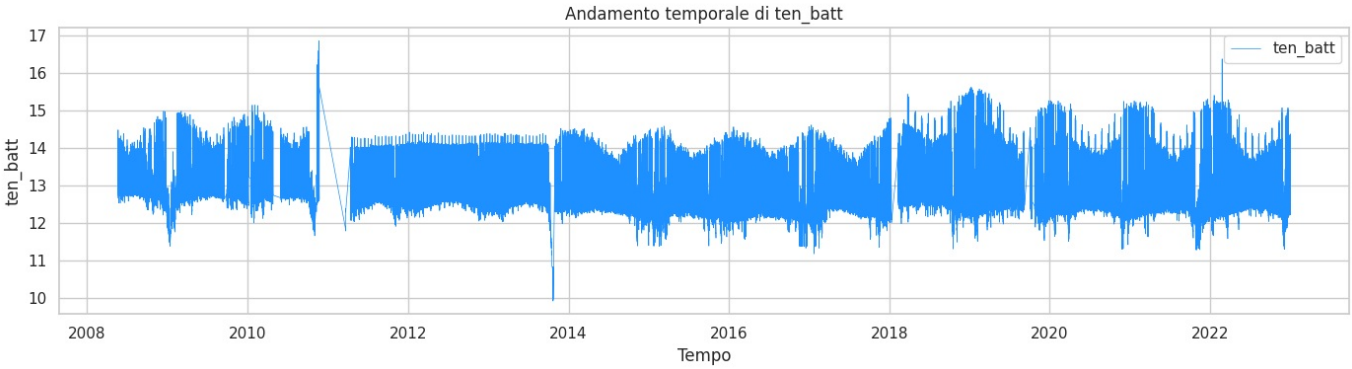

Distribuzione di tilt_x_Avg
1e7
Frequenza
tilt_x_Avg
1e8

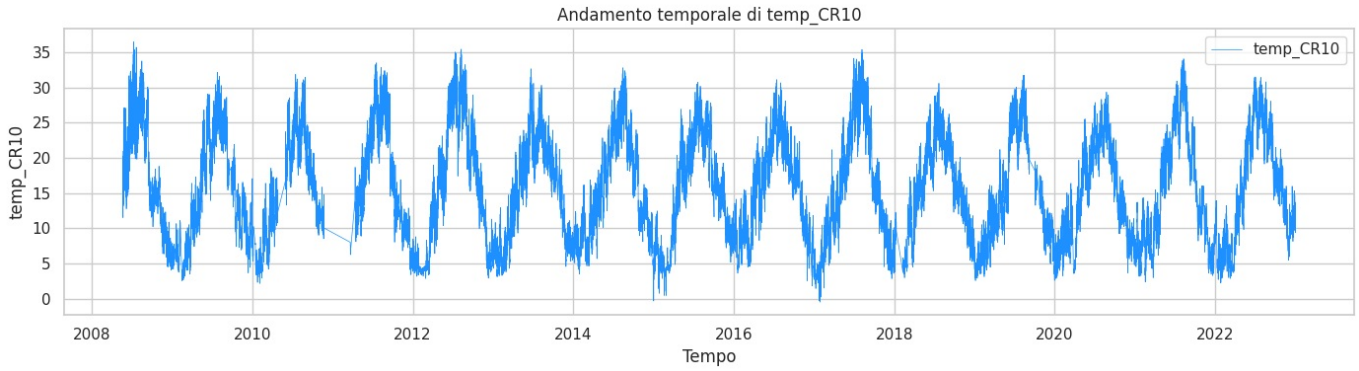

Distribuzione di tilt_y_Avg
1e6
Frequenza
tilt_y_Avg

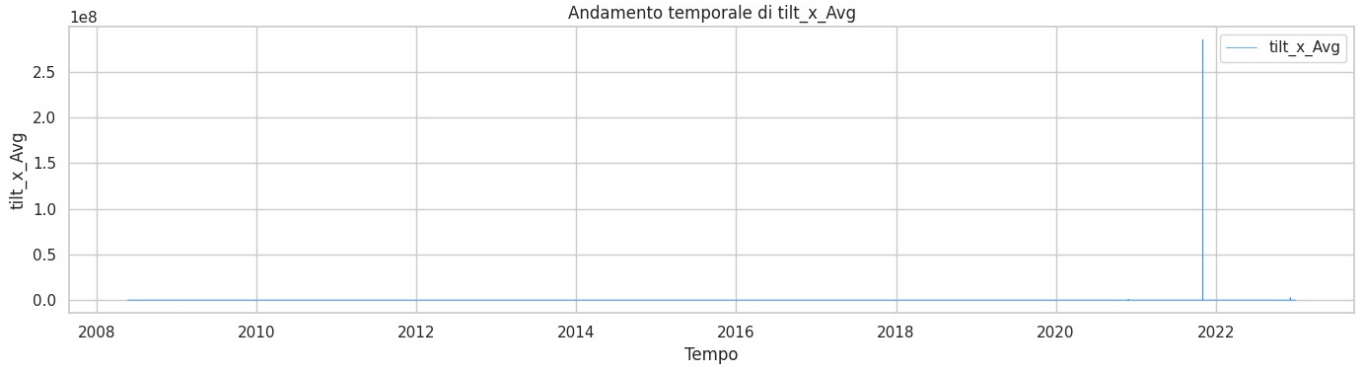

Distribuzione di temp_tilt
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Frequenza
0
50
100
150
200
250
300
temp_tilt

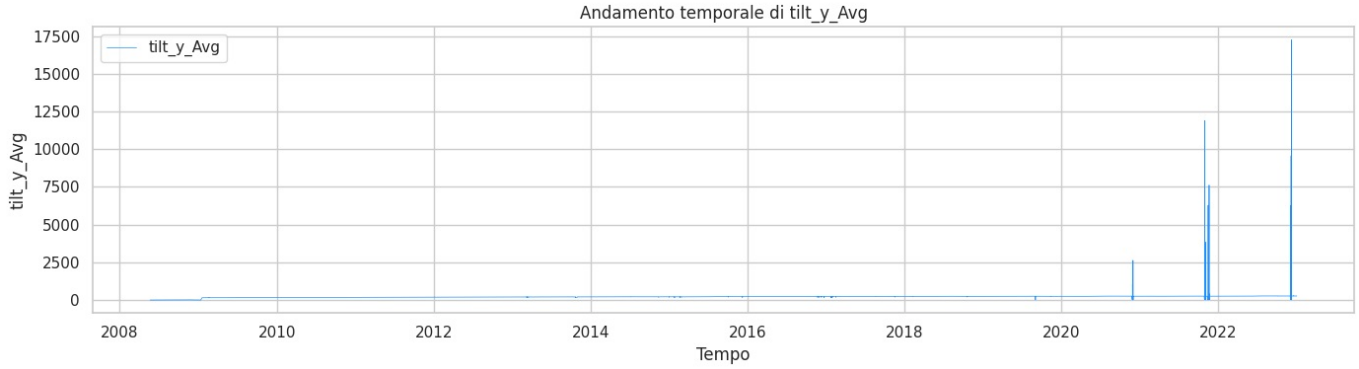

Distribuzione di nord_tilt
1e6
6
5
4
3
2
1
0
Frequenza
0
1000
2000
3000
4000
5000
6000
7000
8000
nord_tilt

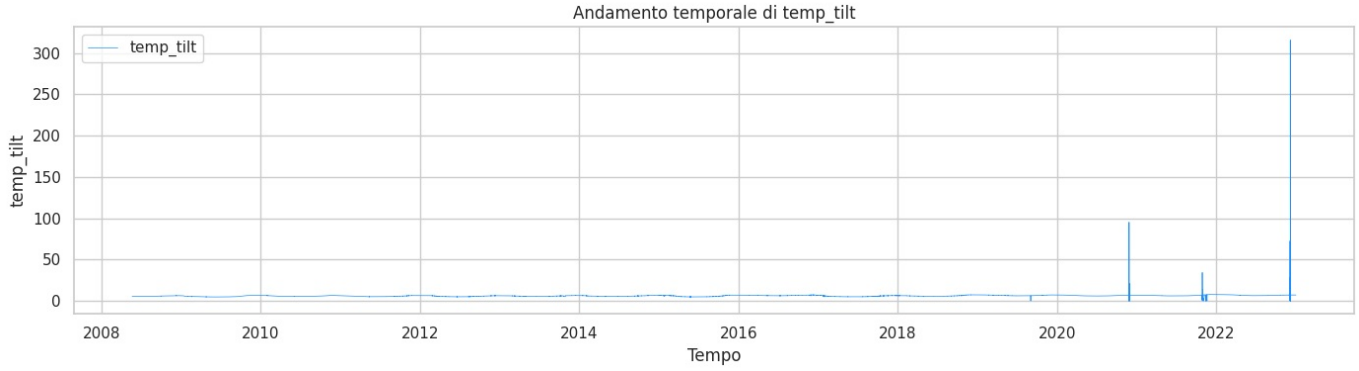

Distribuzione di barometro
300000
250000
200000
150000
100000
50000
0
Frequenza
400
500
600
700
800
barometro

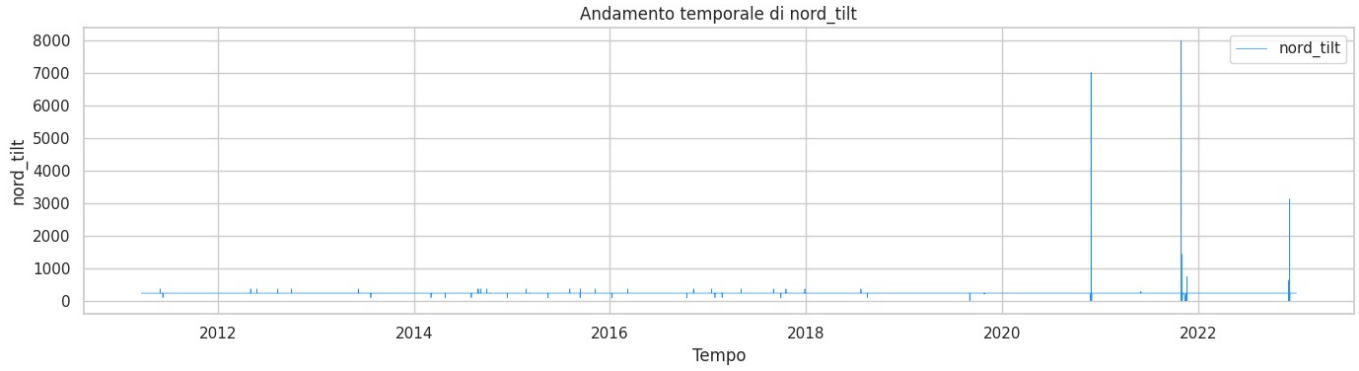

Andamento temporale di ten_batt
ten_batt
17
16
15
14
13
12
11
10
ten_batt
2008
2010
2012
2014
2016
2018
2020
2022
Tempo

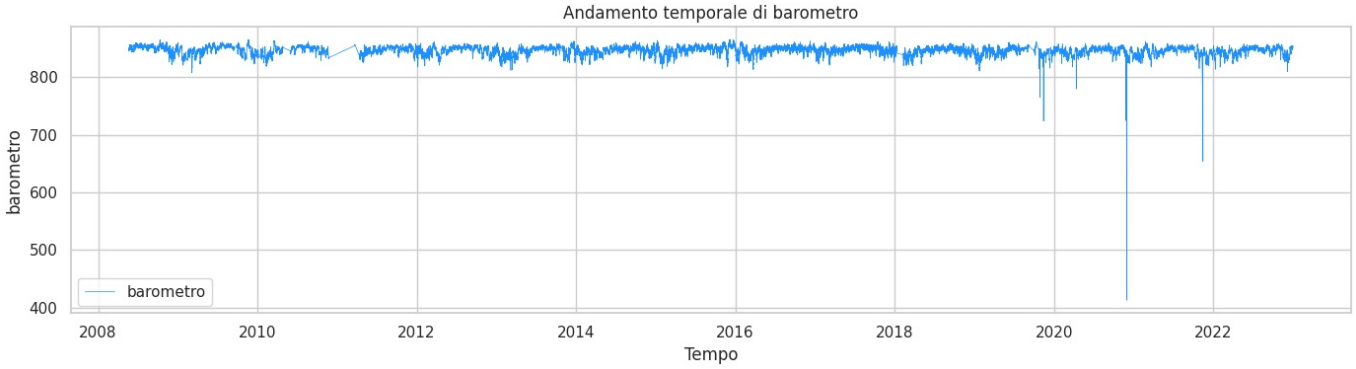

Andamento temporale di temp_CR10
temp_CR10
temp_CR10
Tempo


Andamento temporale di tilt_x_Avg
1e8
tilt_x_Avg
tilt_x_Avg
Tempo


Andamento temporale di tilt_y_Avg
tilt_y_Avg
tilt_y_Avg
Tempo


Andamento temporale di temp_tilt
temp_tilt
temp_tilt
Tempo


Andamento temporale di nord_tilt
nord_tilt
nord_tilt
Tempo



In [80]:
```
#Loading cleaned files
clean_df = pd.read_csv("clinometria_clean.csv")

#Visualization of first 5 rows
print(clean_df.head())
```

```
          time          ten_batt  temp_CR10  tilt_x_Avg  tilt_y_Avg  temp_tilt  nord_tilt  barometro
0  2008-05-21 08:30:00     12.83      14.78     2.94896    -12.3582      5.380        NaN        846
1  2008-05-21 08:45:00     12.85      14.78     2.96006    -12.4142      5.404        NaN        846
2  2008-05-21 09:00:00     12.82      14.73     2.96532    -12.4123      5.410        NaN        848
3  2008-05-21 09:15:00     13.03      14.68     2.96947    -12.4148      5.409        NaN        847
4  2008-05-21 09:30:00     13.13      14.66     2.97167    -12.4201      5.406        NaN        847
```

1.8 tremore vulcanico:

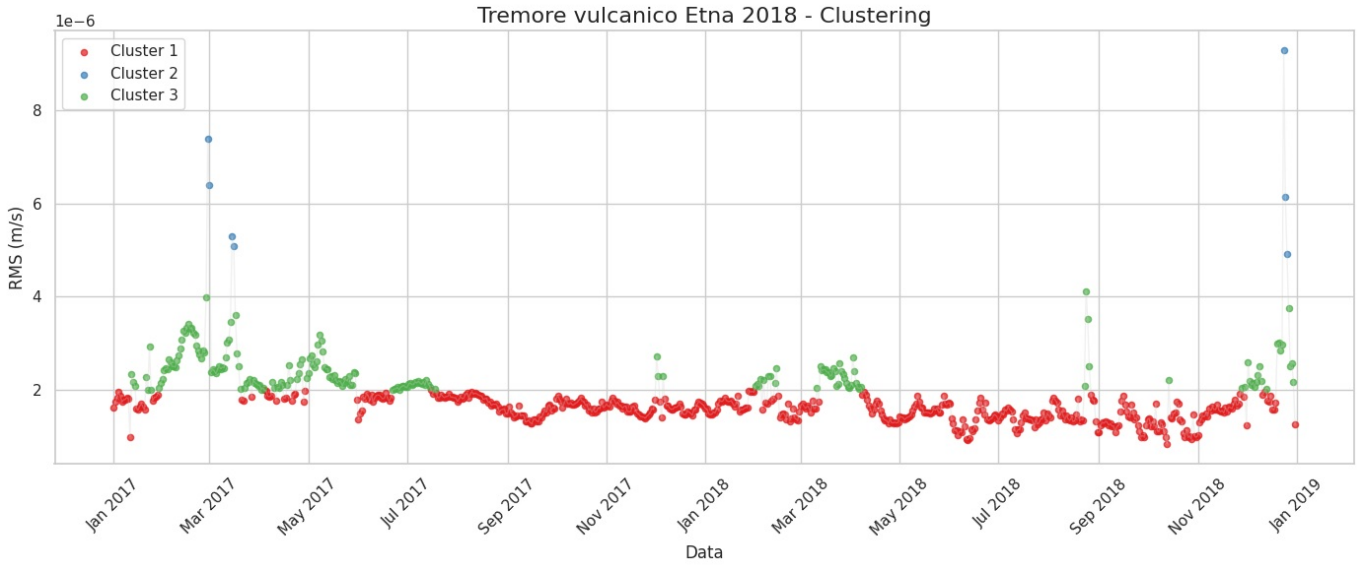
utilizzo RMS come misura per la visualizzazione grafica e mostro una suddivisione in cluster per distinguere bene i periodi di quiete con quelli più movimentati

In [81]:
```
# Loading and starting cleaning
file_path = "/content/drive/MyDrive/Etna2018/Etna2018.xlsx - tremore vulcanico.csv"
df = pd.read_csv(file_path)
df.columns = ['Data', 'RMS']
df['RMS'] = df['RMS'].str.replace(',', '.', regex=False).astype(float)
df['Data'] = pd.to_datetime(df['Data'], dayfirst=True)
df = df.sort_values(by='Data').reset_index(drop=True)


print("Missing values first:\n", df.isnull().sum())
df.dropna(inplace=True)


Q1 = df['RMS'].quantile(0.25)
Q3 = df['RMS'].quantile(0.75)
IQR = Q3 - Q1
df = df[(df['RMS'] >= Q1 - 1.5*IQR) & (df['RMS'] <= Q3 + 1.5*IQR)]


print("Duplicated rows:", df.duplicated(subset=['Data']).sum())
df.drop_duplicates(subset=['Data'], keep='first', inplace=True)

# 4. Verifying RMS range
print("Missing values in RMS:\n", df[df['RMS'] < 0])


df = df.set_index('Data').asfreq('D').reset_index()

# Final result
print("\nCleaned Data:")
print(df.head())
print("\nStatistics RMS:\n", df['RMS'].describe())

# Salva il risultato
df.to_csv("tremore vulcanico_cleaned.csv", index=False)
print("Clean completed. File saved as tremore vulcanico_cleaned.csv'")
```

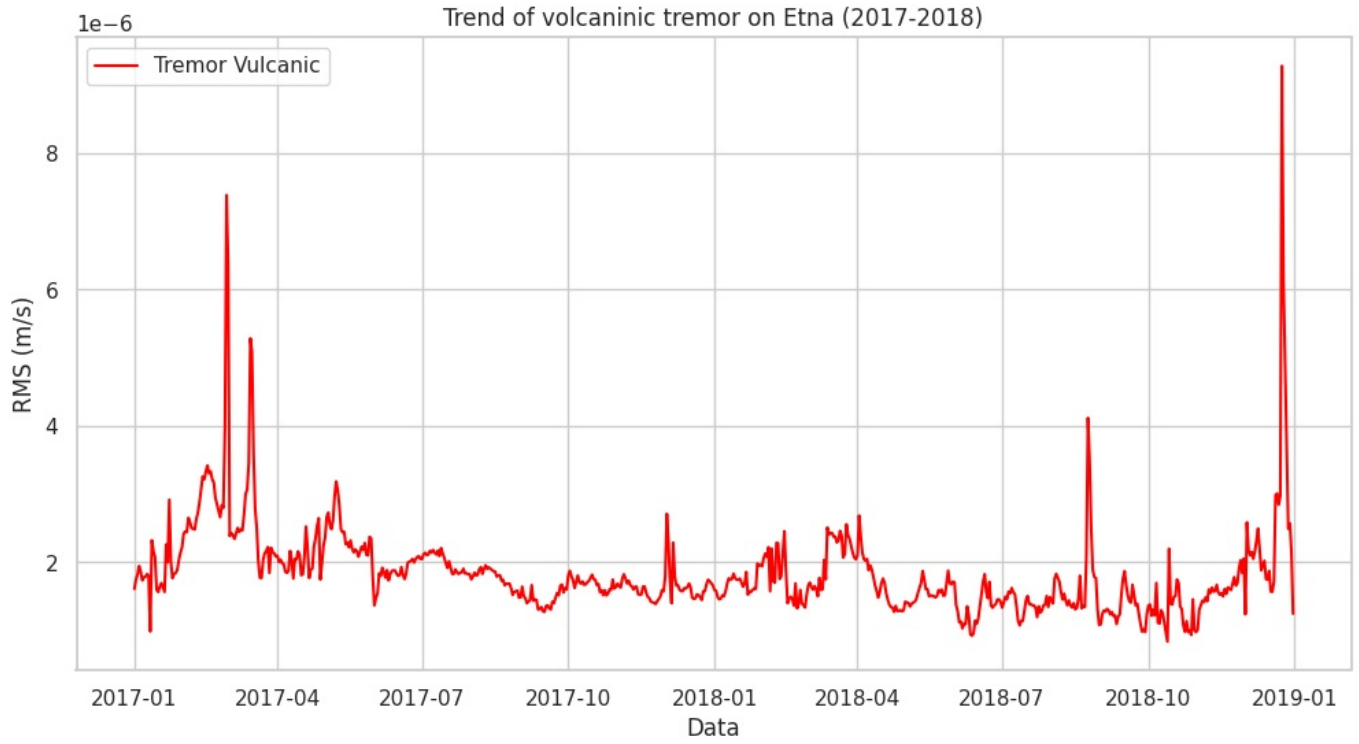
```
Missing values first:
 Data    0
RMS     0
dtype: int64
Duplicated rows: 0
Missing values in RMS:
 Empty DataFrame
Columns: [Data, RMS]
Index: []

Cleaned Data:
     Data        RMS
0 2017-01-01  0.000002
1 2017-01-02  0.000002
2 2017-01-03  0.000002
3 2017-01-04  0.000002
4 2017-01-05  0.000002

Statistics RMS:
 count    6.980000e+02
mean     1.741576e-06
std      3.898435e-07
min      8.386676e-07
25%      1.468191e-06
50%      1.690605e-06
75%      1.984823e-06
max      2.882597e-06
Name: RMS, dtype: float64
Clean completed. File saved as tremore vulcanico_cleaned.csv'
```

In [82]:

```python
# Loading and cleaning data
file_path = "/content/drive/MyDrive/Etna2018/Etna2018.xlsx - tremore vulcanico.csv"
df = pd.read_csv(file_path)
df.columns = ['Data', 'RMS']
df['RMS'] = df['RMS'].str.replace(',', '.', regex=False).astype(float)
df['Data'] = pd.to_datetime(df['Data'], dayfirst=True)
df = df.sort_values(by='Data').reset_index(drop=True)

# K-Means clustering su RMS
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
df['Cluster'] = kmeans.fit_predict(df[['RMS']])


sns.set(style="whitegrid")
palette = sns.color_palette("Set1", n_colors=n_clusters)

# Graph
plt.figure(figsize=(14, 6))
for cluster in range(n_clusters):
    cluster_data = df[df['Cluster'] == cluster]
    plt.scatter(cluster_data['Data'], cluster_data['RMS'],
                label=f"Cluster {cluster+1}", s=20, alpha=0.7,
                color=palette[cluster])

# Formatting
plt.plot(df['Data'], df['RMS'], color='lightgray', alpha=0.3, linewidth=1)
plt.title('Tremore vulcanico Etna 2018 - Clustering', fontsize=16)
plt.xlabel('Data')
plt.ylabel('RMS (m/s)')
plt.legend()
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=2))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [83]:
```
# Loading data
data = pd.read_csv('/content/drive/MyDrive/Etna2018/Etna2018.xlsx - tremore vulcanico.csv', delimiter=',', decim
data['Data'] = pd.to_datetime(data['ECPNZ'], dayfirst=True)
data = data.sort_values('Data')


plt.figure(figsize=(12, 6))
plt.plot(data['Data'], data['RMS(m/s)'], label='Tremor Vulcanic', color='red')
plt.xlabel('Data')
plt.ylabel('RMS (m/s)')
plt.title('Trend of volcaninic tremor on Etna (2017-2018)')
plt.grid(True)
plt.legend()
plt.show()
```



1.9 terremoti:

- Lancio una funzione che permette di mettere le coordinate di longitudine e latitudine in modo da visualizzare una mappa che mostri il segnale dove è stato registrato
- Considerata l'importanza di questo file e dei dati per eventuali modelli da applicare durante l'analisi, ho deciso di mosrare più grafici in modo da catturare più aspetti

1. Una distribuzione temporale per vedere quanti terremoti si verificano per ogni mese
2. Istogramma con distribuzione prima della magnitudo e poi della profondità dei terremoti
3. Mappa per vedere la densità dei terremoti, in quale zona, a quali coordinate geografiche corrispondono più rilevazioni

4. Scatter-plot che mostri come varia la magnitudo al variare della profondità

In [84]:
```
def clean_earthquake_data(file_path):
    # Loading data file
    df = pd.read_csv(file_path)


    numeric_cols = ['Lat (°N)', 'Long (°E)', 'Depth (km)', 'ML']
    for col in numeric_cols:

        df[col] = (df[col].astype(str)
                   .str.replace(',', '.', regex=False)
                   .astype(float))


    try:
        df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['Origin Time'], dayfirst=True)
    except:
        df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['Origin Time'])


    df.drop(['Date', 'Origin Time'], axis=1, inplace=True)


    df['Depth (km)'] = np.where((df['Depth (km)'] >= 0) & (df['Depth (km)'] < 1000),
                                df['Depth (km)'],
                                np.nan)


    cols = ['Datetime', 'Lat (°N)', 'Long (°E)', 'Depth (km)', 'ML']
    df = df[cols]


    df = df.drop_duplicates()


    df = df.sort_values('Datetime')


    df = df.reset_index(drop=True)

    return df


cleaned_data = clean_earthquake_data('/content/drive/MyDrive/Etna2018/Etna2018_V2.xlsx - terremoti.csv')

# Sving cleaned result
cleaned_data.to_csv('terremoti_puliti.csv', index=False)

print("Pulizia completata. Dati salvati in 'terremoti_puliti.csv'")
```

```
Pulizia completata. Dati salvati in 'terremoti_puliti.csv'
```

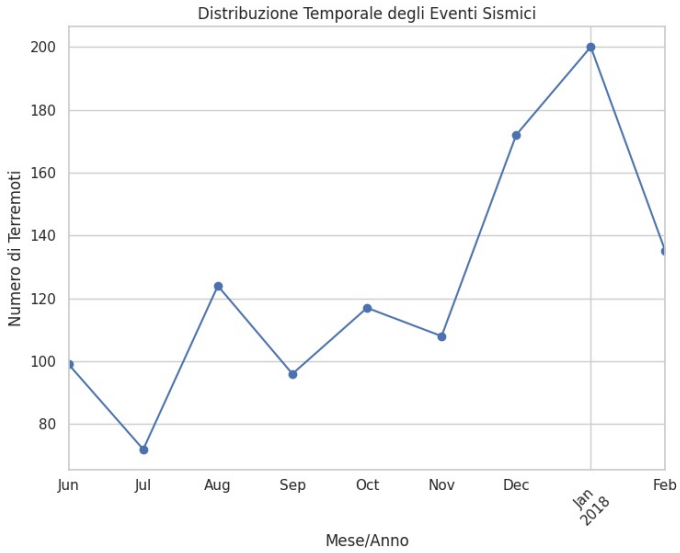
In [85]:
```
def mostra_posizione(latitudine, longitudine):
    # Creating the map centered on coordinates
    mappa = folium.Map(
        location=[latitudine, longitudine],
        zoom_start=15,
        tiles="OpenStreetMap"
    )

    # Adding a marker on the position
    folium.Marker(
        [latitudine, longitudine],
        tooltip="position selected",
        popup=f"Lat: {latitudine}<br>Lon: {longitudine}"
    ).add_to(mappa)

    # Adding a circle to better visualize
    folium.Circle(
        radius=200,
        location=[latitudine, longitudine],
        color="red",
        fill=True,
        fill_opacity=0.2
    ).add_to(mappa)

    # Saving map
    nome_file = f"mappa_{latitudine}_{longitudine}.html"
    mappa.save(nome_file)

    print(f"Map generated with success! Open file '{nome_file}' here on the left")
```

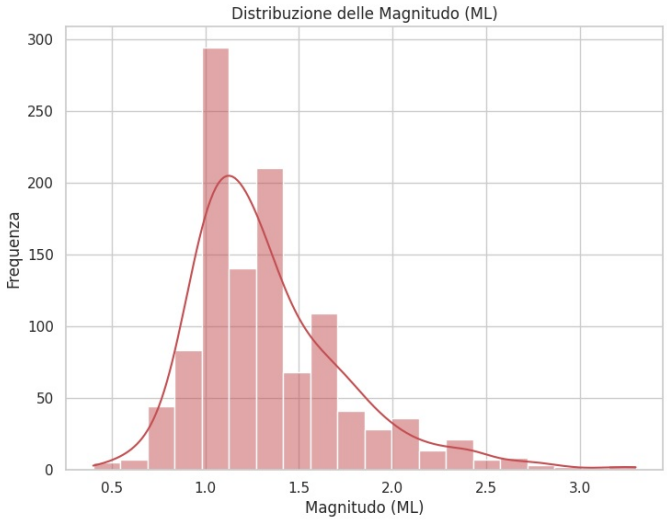
```
# Example of usage
if __name__ == "__main__":
    print("Insert geographic coordinates")

    try:
        lat = float(input("Latitude (es. 37.7749): ").strip())
        lon = float(input("Longitude (es. -122.4194): ").strip())

        # Verifying that coordinates are correct
        if not (-90 <= lat <= 90) or not (-180 <= lon <= 180):
            raise ValueError("Non valid coordinate")

        mostra_posizione(lat, lon)

    except ValueError as e:
        print(f"Error: {e}. Insert valid numbers.")
```

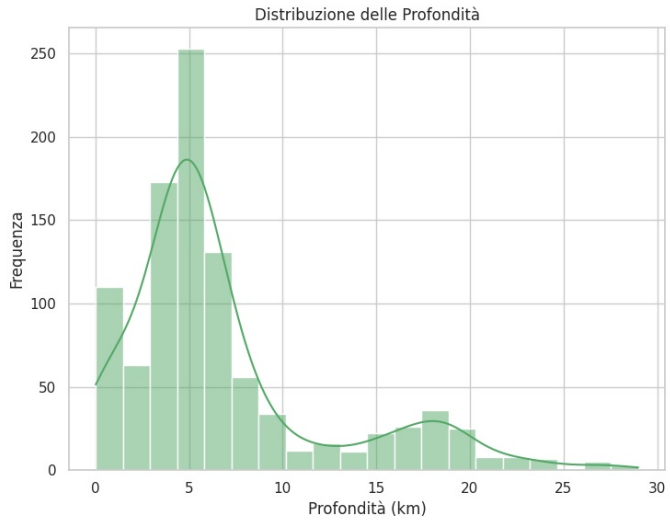
```
Insert geographic coordinates
Latitude (es. 37.7749): 37.877193
Longitude (es. -122.4194): 15.062835
Map generated with success! Open file 'mappa_37.877193_15.062835.html' here on the left
```

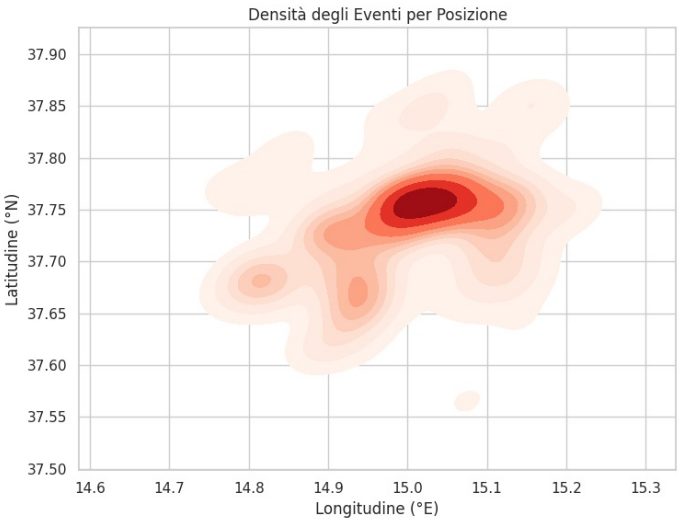
In [86]:

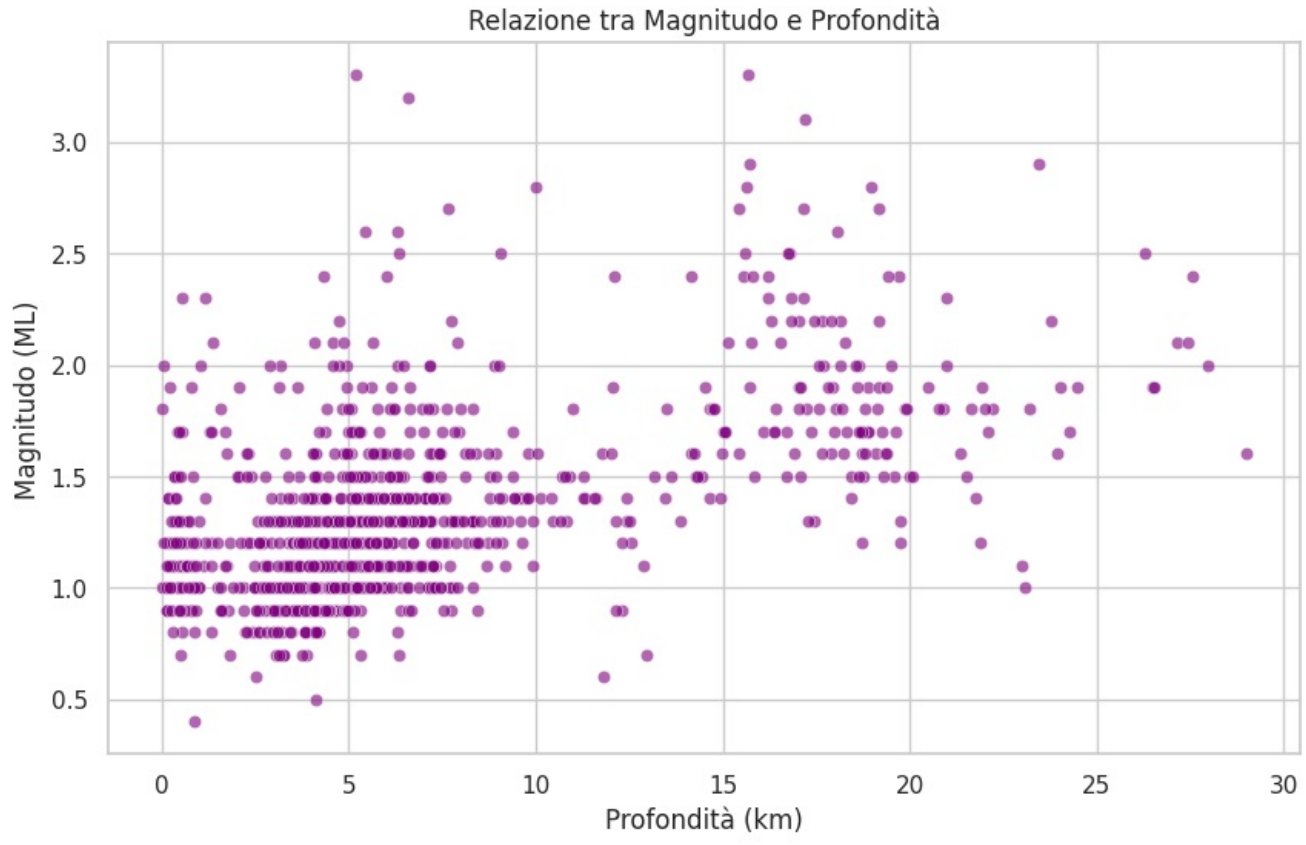
```
def plot_earthquake_distribution(cleaned_data_path):
    # Loading data
    df = pd.read_csv(cleaned_data_path, parse_dates=['Datetime'])


    sns.set(style="whitegrid")
    plt.figure(figsize=(15, 12))


    plt.subplot(2, 2, 1)
    df['YearMonth'] = df['Datetime'].dt.to_period('M')
    monthly_counts = df.groupby('YearMonth').size()
    monthly_counts.plot(kind='line', marker='o', color='b')
    plt.title('Distribuzione Temporale degli Eventi Sismici')
    plt.xlabel('Mese/Anno')
    plt.ylabel('Numero di Terremoti')
    plt.xticks(rotation=45)

    # 2. Magnitudo distribution
    plt.subplot(2, 2, 2)
    sns.histplot(df['ML'], bins=20, kde=True, color='r')
    plt.title('Distribuzione delle Magnitudo (ML)')
    plt.xlabel('Magnitudo (ML)')
    plt.ylabel('Frequenza')

    # 3. Depth distribution
    plt.subplot(2, 2, 3)
    sns.histplot(df['Depth (km)'], bins=20, kde=True, color='g')
    plt.title('Distribuzione delle Profondità')
    plt.xlabel('Profondità (km)')
    plt.ylabel('Frequenza')

    # 4. Density map of latitutde and longitude
    plt.subplot(2, 2, 4)
    sns.kdeplot(data=df, x='Long (°E)', y='Lat (°N)', cmap='Reds', shade=True)
    plt.title('Densità degli Eventi per Posizione')
    plt.xlabel('Longitudine (°E)')
    plt.ylabel('Latitudine (°N)')

    plt.tight_layout()
    plt.show()

    # Adding graph
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=df, x='Depth (km)', y='ML', alpha=0.6, color='purple')
    plt.title('Relazione tra Magnitudo e Profondità')
    plt.xlabel('Profondità (km)')
    plt.ylabel('Magnitudo (ML)')
    plt.show()

# Usage of the function
plot_earthquake_distribution('terremoti_puliti.csv')
```











2 Uniamo i file excel in uno solo cosi da trattarli come unico file

- Rappresentiamo la correlation matrix per vedere se le variabili sono correlate
- Diverse rappresentazioni della correlation matrix, una con un colore ma diverse gradazioni e

l'altra colori diverse per distinguerle meglio

In [87]:
```
# List of cleaned files
clean_files = {
    'heat_flux': 'heat_flux_cleaned.csv',
    'SO2_flux': 'SO2_flux_cleaned.csv',
    'HCl_flux': 'HCl_flux_clean.csv',
    'He_ratio': 'cleaned_he_ratio.csv',
    'CO2_SO2_ratio': 'cleaned_CO2_SO2_ratio.csv',
    'clinometria': 'clinometria_clean.csv',
    'tremore': 'tremore vulcanico_cleaned.csv',
    'terremoti': 'terremoti_puliti.csv'
}

# Loading all of the cleaned files
dfs = {}
for name, file in clean_files.items():
    try:

        dfs[name] = pd.read_csv(file, parse_dates=True, engine='python')
        print(f"File {name} caricato con successo.")
    except Exception as e:
        print(f"Errore nel caricare {file}: {e}")
        continue


for name, df in dfs.items():

    time_col = None
    for col in df.columns:
        if 'date' in col.lower() or 'time' in col.lower() or 'data' in col.lower():
            time_col = col
            break

    if time_col:
        try:
            # Converting in datetime
            dfs[name][time_col] = pd.to_datetime(dfs[name][time_col])
            dfs[name].set_index(time_col, inplace=True)
            # Removing eventual duplicated values
            dfs[name] = dfs[name][~dfs[name].index.duplicated(keep='first')]
            print(f"{name}: temporal columns settled on '{time_col}'")
        except Exception as e:
            print(f"Error on processing of datetime columns {name}: {e}")
    else:
        print(f"{name}: none datetime column found")


merged_df = pd.DataFrame()

for name, df in dfs.items():
    if df.empty:
        continue

    if merged_df.empty:
        merged_df = df.copy()
    else:
        try:
            # Using join to avoid any problems
            merged_df = merged_df.join(df, how='outer', rsuffix=f'_{name}')
            print(f"Adding {name} to merge")
        except Exception as e:
            print(f"Error on merging og {name}: {e}")


merged_df.sort_index(inplace=True)


merged_df = merged_df.loc[:,~merged_df.columns.duplicated()]

# Saving results
try:
    merged_df.to_csv('merged_etna_data.csv')
    print("\nMerge completato! File salvato come 'merged_etna_data.csv'")

    # Final results
    print("\nStatistic of cleaned Dataframe:")
    print(f"- Total rows: {len(merged_df)}")
    print(f"- Total Columns: {len(merged_df.columns)}")
    print(f"- Covered period: {merged_df.index.min()} - {merged_df.index.max()}")
    print("\nfirst 5 rows:")
    print(merged_df.head())
except Exception as e:
```

```
    print(f"Error of saving of the file: {e}")
```

```
File heat_flux caricato con successo.
File SO2_flux caricato con successo.
File HCl_flux caricato con successo.
File He_ratio caricato con successo.
File CO2_SO2_ratio caricato con successo.
File clinometria caricato con successo.
File tremore caricato con successo.
File terremoti caricato con successo.
heat_flux: temporal columns settled on 'date'
SO2_flux: temporal columns settled on 'date'
HCl_flux: temporal columns settled on 'Date'
He_ratio: temporal columns settled on 'Date'
CO2_SO2_ratio: temporal columns settled on 'Date'
clinometria: temporal columns settled on 'time'
tremore: temporal columns settled on 'Data'
terremoti: temporal columns settled on 'Datetime'
Adding SO2_flux to merge
Adding HCl_flux to merge
Adding He_ratio to merge
Adding CO2_SO2_ratio to merge
Adding clinometria to merge
Adding tremore to merge
Adding terremoti to merge

Merge completato! File salvato come 'merged_etna_data.csv'

Statistic of cleaned Dataframe:
- Total rows: 481108
- Total Columns: 27
- Covered period: 2008-05-21 08:30:00 - 2022-12-31 23:45:00

first 5 rows:
                     Radiant Heat Flux [W]  SO2_flux_t_d  daily HCl flux (t/d) outlier  Stadio  Fondachello  P39
Vallone Salato  Naftia  CO2/SO2 Is_Outlier  Year  Month  Day Date_only  ten_batt  temp_CR10  tilt_x_Avg  tilt_y_
Avg  temp_tilt  nord_tilt  barometro  RMS  Lat (°N)  Long (°E)  Depth (km)  ML
2008-05-21 08:30:00          NaN               NaN               NaN             NaN    NaN       NaN      NaN
NaN         NaN     NaN        NaN     NaN   NaN   NaN      NaN      12.83     14.78      2.94896    -12.3582
5.380      NaN        846.0   NaN     NaN       NaN         NaN     NaN
2008-05-21 08:45:00          NaN               NaN               NaN             NaN    NaN       NaN      NaN
NaN         NaN     NaN        NaN     NaN   NaN   NaN      NaN      12.85     14.78      2.96006    -12.4142
5.404      NaN        846.0   NaN     NaN       NaN         NaN     NaN
2008-05-21 09:00:00          NaN               NaN               NaN             NaN    NaN       NaN      NaN
NaN         NaN     NaN        NaN     NaN   NaN   NaN      NaN      12.82     14.73      2.96532    -12.4123
5.410      NaN        848.0   NaN     NaN       NaN         NaN     NaN
2008-05-21 09:15:00          NaN               NaN               NaN             NaN    NaN       NaN      NaN
NaN         NaN     NaN        NaN     NaN   NaN   NaN      NaN      13.03     14.68      2.96947    -12.4148
5.409      NaN        847.0   NaN     NaN       NaN         NaN     NaN
2008-05-21 09:30:00          NaN               NaN               NaN             NaN    NaN       NaN      NaN
NaN         NaN     NaN        NaN     NaN   NaN   NaN      NaN      13.13     14.66      2.97167    -12.4201
5.406      NaN        847.0   NaN     NaN       NaN         NaN     NaN
```

In [88]:

```python
import pandas as pd

# --- 1. CARICA IL DATASET UNIFICATO ---
merged_df = pd.read_csv('merged_etna_data.csv', index_col=0, parse_dates=True)
merged_df.index = pd.to_datetime(merged_df.index, errors='coerce')
merged_df = merged_df[merged_df.index.notna()]  # Rimuove date non valide

print(f" Periodo originale: {merged_df.index.min()} → {merged_df.index.max()}")

# --- 2. FILTRO TEMPORALE ---
# Inserisci qui il tuo intervallo di tempo (modifica se vuoi!)
start_date = '2017-01-01'
end_date   = '2019-12-31'

filtered_df = merged_df.loc[start_date:end_date]
print(f" Periodo filtrato: {filtered_df.index.min()} → {filtered_df.index.max()}")
print(f" Dimensione: {filtered_df.shape}")

# --- 3. IDENTIFICA TIME SERIES NUMERICHE ---
# Scegliamo solo le colonne numeriche
numeric_cols = filtered_df.select_dtypes(include='number')

# Opzionale: rimuovi colonne con pochi dati (es. meno del 30%)
valid_cols = numeric_cols.dropna(axis=1, thresh=int(0.3 * len(numeric_cols)))

print(f" Colonne numeriche con copertura sufficiente:")
print(list(valid_cols.columns))

# --- 4. (OPZIONALE) Salva subset per analisi avanzata ---
valid_cols.to_csv("numeric_timeseries_filtered.csv")
print(" File salvato: numeric_timeseries_filtered.csv")
```

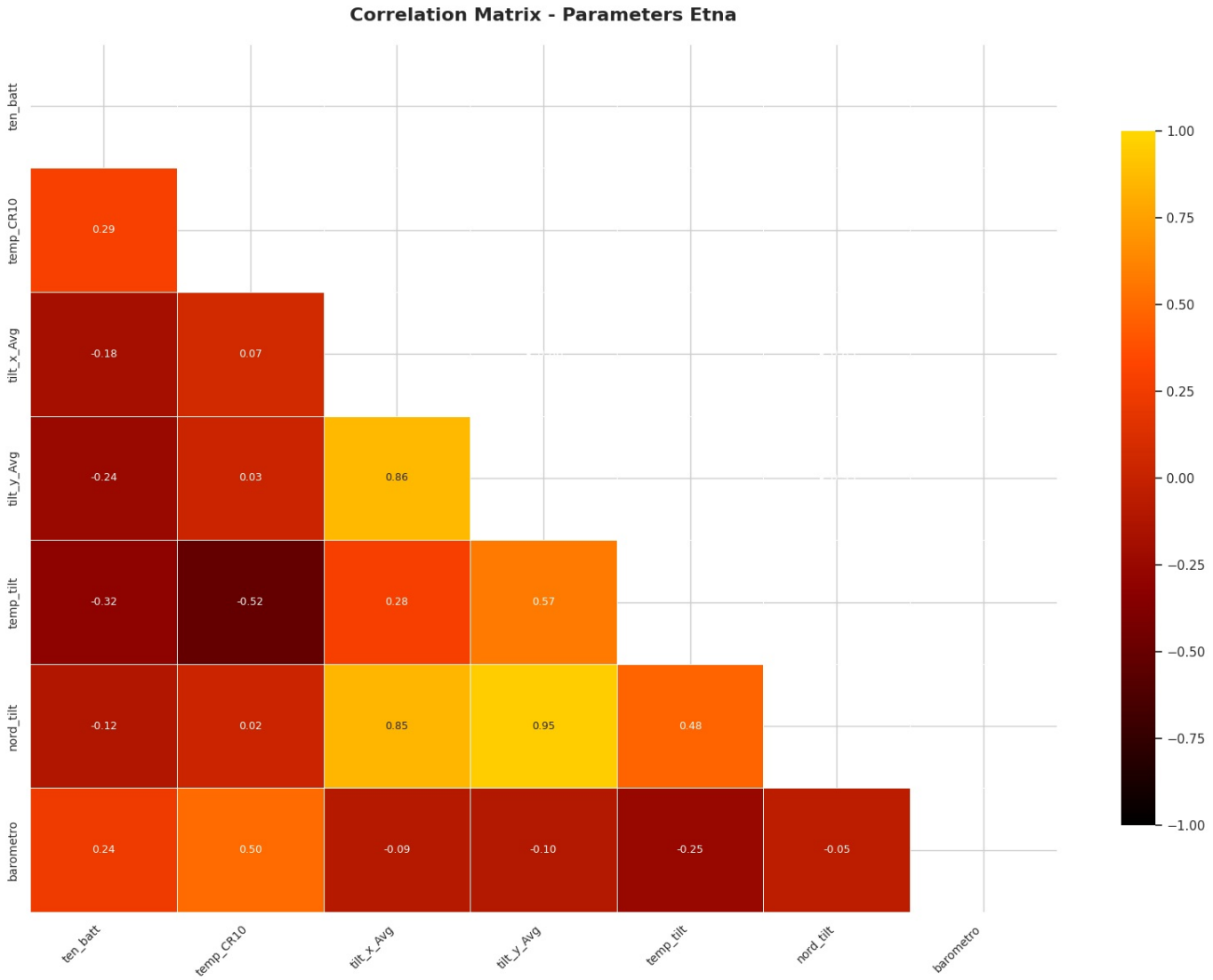
```
Periodo originale: 2008-05-21 08:30:00 → 2022-12-31 23:45:00
Periodo filtrato: 2017-01-01 00:00:00 → 2019-12-31 23:45:00
Dimensione: (70300, 27)
Colonne numeriche con copertura sufficiente:
['ten_batt', 'temp_CR10', 'tilt_x_Avg', 'tilt_y_Avg', 'temp_tilt', 'nord_tilt', 'barometro']
File salvato: numeric_timeseries_filtered.csv
```

2.1 Correlation matrix sul dataframe unito

In [89]:

```python
# 1. Loading cleaned file
def load_data():
    # Loading resulting CSV by the merge
    df = pd.read_csv(
        'merged_etna_data.csv',
        index_col=0,
        parse_dates=True,
        low_memory=False
    )


    df.index = pd.to_datetime(df.index, errors='coerce')


    df = df[df.index.notna()]

    return df

df = load_data()

# 2. Cleaning Data
def clean_data(df):

    numeric_df = df.select_dtypes(include=[np.number])


    cleaned_df = numeric_df.dropna(axis=1, thresh=len(numeric_df)*0.3)


    for col in cleaned_df.columns:
        if cleaned_df[col].isnull().any():
            cleaned_df[col] = cleaned_df[col].interpolate(method='time', limit_area='inside')

    return cleaned_df

cleaned_df = clean_data(df)


def plot_correlation_matrix(df):

    corr_matrix = df.corr(method='spearman')

    # Creating heatmap
    plt.figure(figsize=(16, 12))


    vulcan_cmap = LinearSegmentedColormap.from_list('vulcan', ['#000000', '#8B0000', '#FF4500', '#FFD700'])


    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

    # Plot
    heatmap = sns.heatmap(
        corr_matrix,
        mask=mask,
        cmap=vulcan_cmap,
        vmin=-1,
        vmax=1,
        center=0,
        annot=True,
        fmt=".2f",
        linewidths=0.5,
        annot_kws={"size": 9},
        cbar_kws={"shrink": 0.8}
    )

    # Title and formatting
    plt.title('Correlation Matrix - Parameters Etna', pad=20, fontsize=16, fontweight='bold')
    plt.xticks(rotation=45, ha='right', fontsize=10)
    plt.yticks(fontsize=10)


    threshold = 0.7
    strong_corrs = np.where(np.abs(corr_matrix) > threshold)
```

```
    for i, j in zip(*strong_corrs):
        if i != j and i < j:
            heatmap.text(j+0.5, i+0.5, f" {corr_matrix.iloc[i,j]:.2f}",
                         ha='center', va='center', color='white', fontsize=10)

    plt.tight_layout()
    plt.savefig('correlation_matrix_etna.png', dpi=300, bbox_inches='tight')
    plt.show()

    return corr_matrix

corr_matrix = plot_correlation_matrix(cleaned_df)


strong_correlations = corr_matrix.unstack().sort_values(ascending=False)
strong_correlations = strong_correlations[abs(strong_correlations) > 0.6]
strong_correlations.to_csv('strong_correlations.csv')

print("Completed analysis. Saved files:")
print("- correlation_matrix_etna.png")
print("- strong_correlations.csv")
```



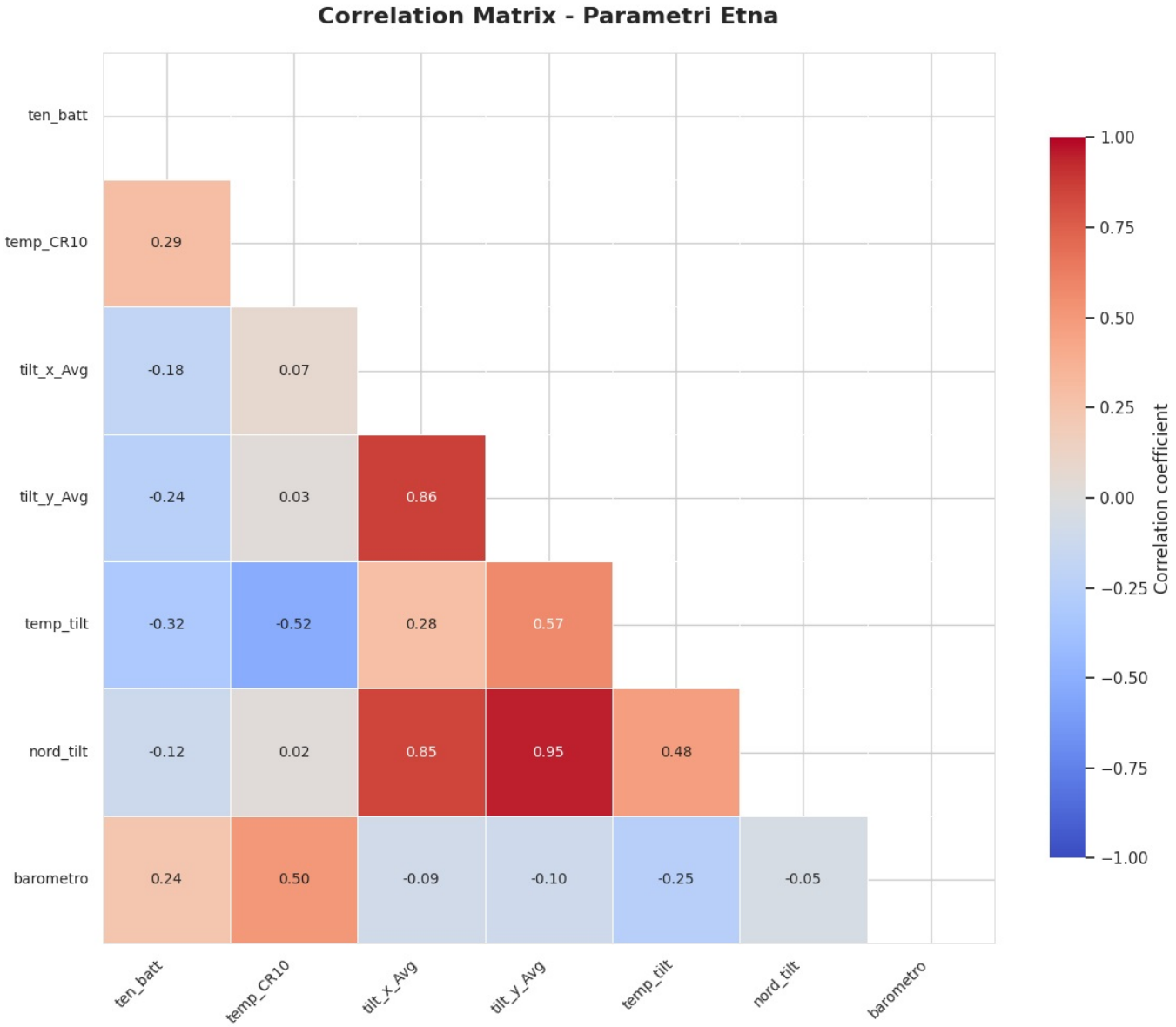

```
Completed analysis. Saved files:
- correlation_matrix_etna.png
- strong_correlations.csv
```

In [90]:
```
# 1. Loading merged Dataset
def load_data():

    try:
        df = pd.read_csv(
            'merged_etna_data.csv',
            index_col=0,
            parse_dates=True,
            low_memory=False
        )
    except FileNotFoundError:
        print("File not found, generating example data")
        date_rng = pd.date_range(start='2020-01-01', end='2020-12-31', freq='D')
        data = {
            'temp_CR10': np.random.normal(15, 5, len(date_rng)),
```

```
            'tilt_x_Avg': np.random.normal(0, 2, len(date_rng)),
            'tilt_y_Avg': np.random.normal(0, 2, len(date_rng)),
            'barometro': np.random.normal(850, 10, len(date_rng)),
            'RHF(W)': np.random.gamma(2, 1e9, len(date_rng)),
            'espc(m/s)': np.random.lognormal(-14, 0.5, len(date_rng))
        }
        df = pd.DataFrame(data, index=date_rng)
        df.to_csv('merged_etna_data.csv')


    df.index = pd.to_datetime(df.index, errors='coerce')
    df = df[df.index.notna()]
    return df

# 2. Cleaning data
def clean_data(df):
    numeric_df = df.select_dtypes(include=[np.number])
    cleaned_df = numeric_df.dropna(axis=1, thresh=len(numeric_df)*0.3)

    for col in cleaned_df.columns:
        if cleaned_df[col].isnull().any():
            cleaned_df[col] = cleaned_df[col].interpolate(method='time', limit_area='inside')

    return cleaned_df

# 3. Correlation matrix
def plot_correlation_matrix(df):

    corr_matrix = df.corr(method='spearman')


    plt.figure(figsize=(12, 10))


    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))


    sns.heatmap(
        corr_matrix,
        mask=mask,
        annot=True,
        cmap='coolwarm',
        fmt=".2f",
        linewidths=0.5,
        cbar_kws={"shrink": 0.8, "label": "Correlation coefficient"},
        annot_kws={"size": 10},
        square=True,
        vmin=-1,
        vmax=1,
        center=0
    )


    plt.title('Correlation Matrix - Parametri Etna',
              fontsize=16, pad=20, fontweight='bold')
    plt.xticks(rotation=45, ha='right', fontsize=10)
    plt.yticks(rotation=0, fontsize=10)


    for _, spine in plt.gca().spines.items():
        spine.set_visible(True)
        spine.set_linewidth(0.5)

    plt.tight_layout()
    plt.savefig('correlation_matrix_etna.png', dpi=300, bbox_inches='tight')
    plt.show()

    return corr_matrix


def main():
    print("Loading data...")
    df = load_data()

    print("\nCleaning data...")
    cleaned_df = clean_data(df)
    print(f"Columns remained after cleaning: {list(cleaned_df.columns)}")

    print("\nGeneration of correlation matrix...")
    corr_matrix = plot_correlation_matrix(cleaned_df)


    strong_corrs = corr_matrix.unstack().sort_values(ascending=False)
```

```
    strong_corrs = strong_corrs[(abs(strong_corrs) > 0.5) & (strong_corrs < 1)]
    strong_corrs.to_csv('strong_correlations.csv')

    print("\nOperations completed with success!")
    print("Files created:")
    print("- correlation_matrix_etna.png")
    print("- strong_correlations.csv")

if __name__ == "__main__":
    main()
```

```
Loading data...

Cleaning data...
Columns remained after cleaning: ['ten_batt', 'temp_CR10', 'tilt_x_Avg', 'tilt_y_Avg', 'temp_tilt', 'nord_tilt',
'barometro']

Generation of correlation matrix...
```




```
Operations completed with success!
Files created:
- correlation_matrix_etna.png
- strong_correlations.csv
```

In [ ]:
```
# Loading merged dataframe
merged_df = pd.read_csv('merged_etna_data.csv', parse_dates=True, index_col=0)


numeric_cols = merged_df.select_dtypes(include=[np.number]).columns.tolist()


corr_matrix = merged_df[numeric_cols].corr()


plt.figure(figsize=(15, 12))
sns.heatmap(corr_matrix,
            annot=True,
            fmt=".2f",
            cmap='coolwarm',
```

```
            center=0,
            linewidths=0.5,
            cbar_kws={"shrink": 0.8})

plt.title('Correlation matrix - Data Etna 2018', fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()


plt.savefig('correlation_matrix_etna.png', dpi=300, bbox_inches='tight')
plt.show()


print("\nStrongest correlations (|correlation| > 0.7):")
corr_pairs = corr_matrix.unstack().sort_values(ascending=False)
high_corr = corr_pairs[(abs(corr_pairs) > 0.7) & (corr_pairs < 1)]
print(high_corr)

# Export the correlation matrix as CSV
corr_matrix.to_csv('correlation_matrix_etna.csv')
print("\nCorrelation matrix saved as 'correlation_matrix_etna.csv'")
```

CONVERSIONE IN PDF

Usiamo questo script per convertire il notebook in pdf e poterlo scaricare

In [ ]:
```
!pip install pdfkit
!apt-get install -y wkhtmltopdf
import pdfkit
config = pdfkit.configuration(wkhtmltopdf="/usr/bin/wkhtmltopdf")
```

In [ ]:
```
# Mounting Google Drive
drive.mount('/content/drive')

# Notebook's path
notebook_path = "/content/drive/MyDrive/ETNA2018-UFFICIAL"
output_html = "/content/tirocinio.html"
output_pdf = "/content/Etna2018.pdf"

# Checking that file exists
if os.path.exists(notebook_path):
    print("Notebook found, converting in HTML...")

    # Loading notebook
    with open(notebook_path, encoding="utf-8") as f:
        notebook = nbformat.read(f, as_version=4)

    # Exporting in HTML
    html_exporter = HTMLExporter()
    (body, resources) = html_exporter.from_notebook_node(notebook)

    # Write HTML on disk
    with open(output_html, "w", encoding="utf-8") as f:
        f.write(body)

    # Converting HTML to PDF
    print("Converting HTML in PDF...")
    pdfkit.from_file(output_html, output_pdf)

    # Downloading PDF
    if os.path.exists(output_pdf):
        print("PDF correctly generated")
        files.download(output_pdf)
    else:
        print("Errore in the generation of PDF")
else:
    print("Notebook not found. Checking the path")
```