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

Disattiviamo i warnings; potrebbe dare nel file di output messaggi di avviso, non rilevanti, li disattiviamo per tenere pulito il file pdf finale

In [1]:
```
import os
# Disattiva i messaggi C++ INFO e WARNING (valore 2)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import warnings
warnings.simplefilter('ignore', category=FutureWarning)
warnings.simplefilter('ignore', category=Warning)

import tensorflow as tf
tf.get_logger().setLevel('ERROR')
tf.autograph.set_verbosity(0)
```

In [2]:
```
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
from scipy import signal
from sklearn.decomposition import PCA
from matplotlib.patches import Ellipse
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
import warnings
from sklearn.metrics import (classification_report, confusion_matrix,
                             accuracy_score, precision_score, recall_score, f1_score)
from sklearn.model_selection import TimeSeriesSplit, train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, classification_report, confusion_
from sklearn.utils.class_weight import compute_class_weight
```

In [3]:
```
for w in [UserWarning, DeprecationWarning, FutureWarning]:
    warnings.filterwarnings("ignore", category=w)

try:
    from pandas.errors import SettingWithCopyWarning
    warnings.filterwarnings("ignore", category=SettingWithCopyWarning)
except ImportError:
    pass
```

In [4]:
```
drive.mount('/content/drive')

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

```
Mounted at /content/drive
'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

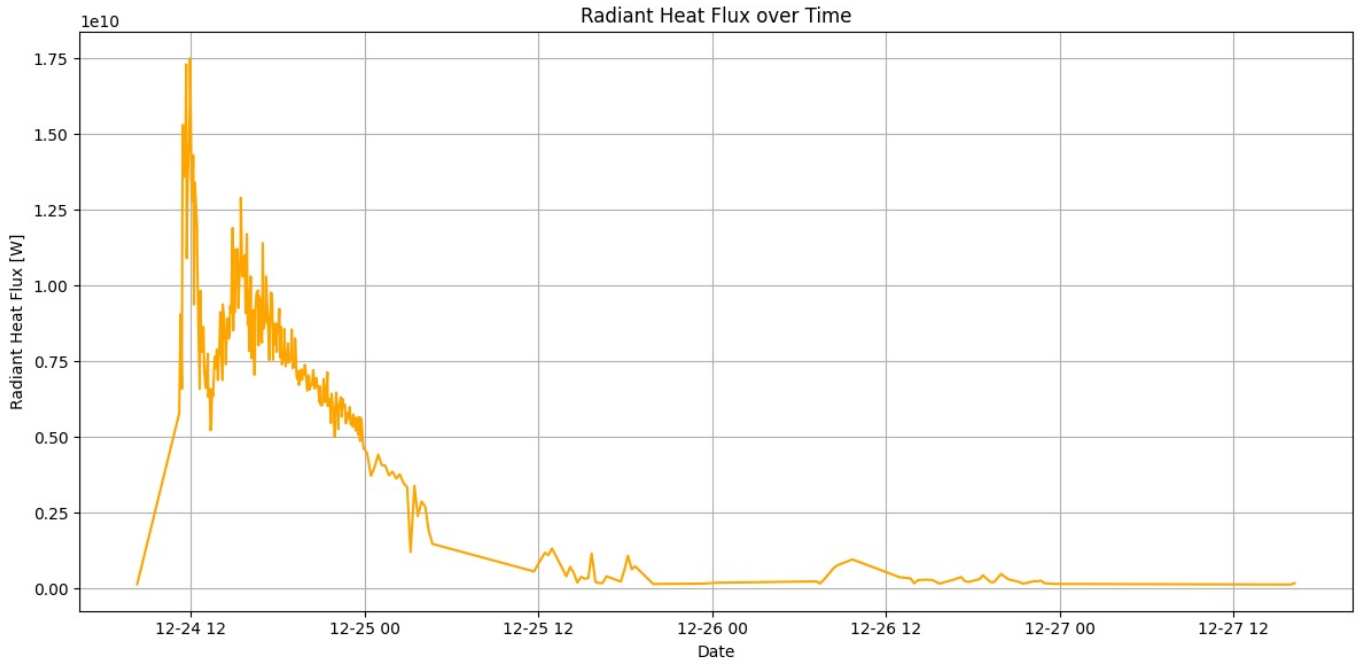
1.2 **Heat-Flux : **

- 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 [5]:
```
# 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 [6]:
```
# 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 [7]:
```
# 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 [8]:
```
# 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

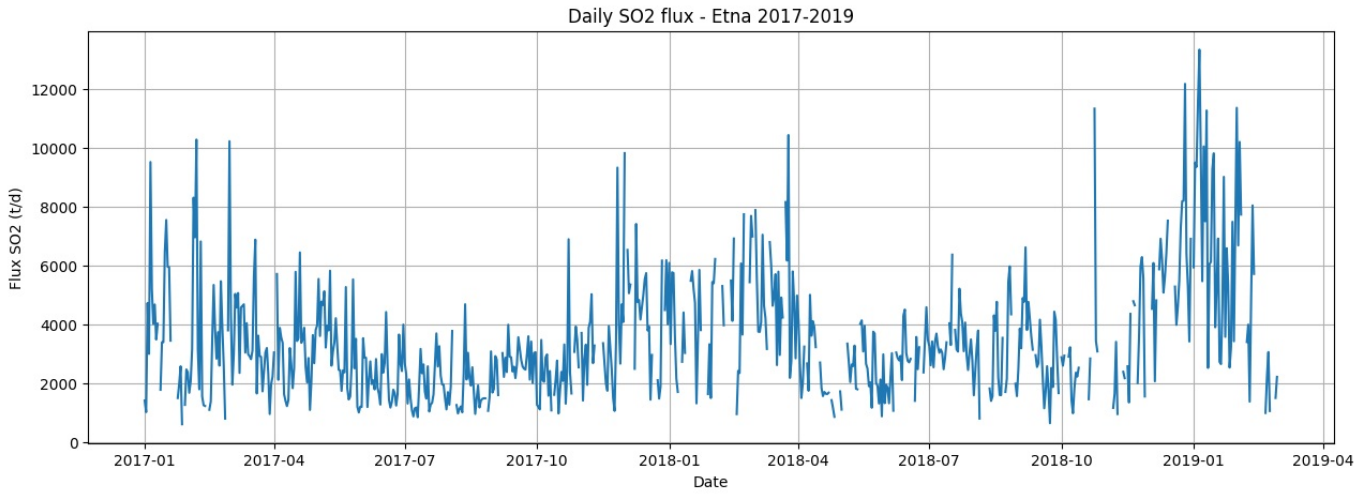
In [9]:
```python
# Loading SO2 file
file_path = '/content/drive/MyDrive/Etna2018/Etna2018.xlsx - SO2 flux.csv'

# 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 [10]:
```
# 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 [11]:
```
#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 [12]:
```
# 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.

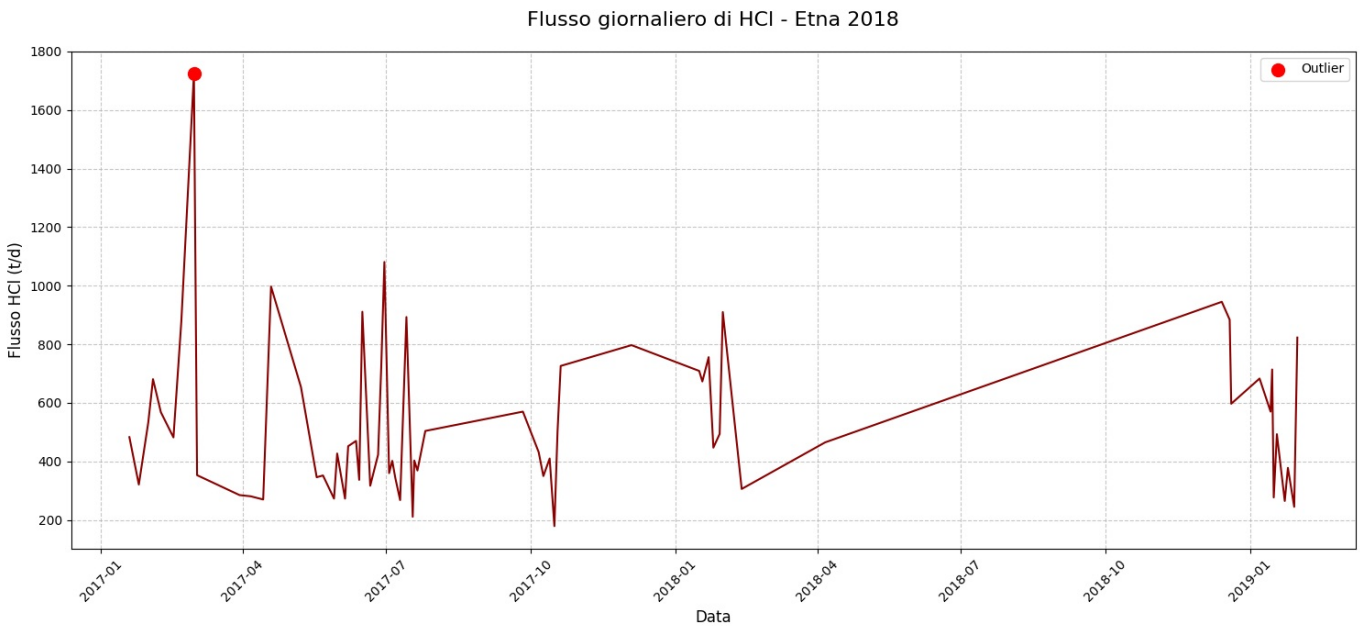
In [13]:
```python
def clean_hcl_flux(input_file, output_file):
    """

    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 [14]:
```
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 [15]:

```
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
```

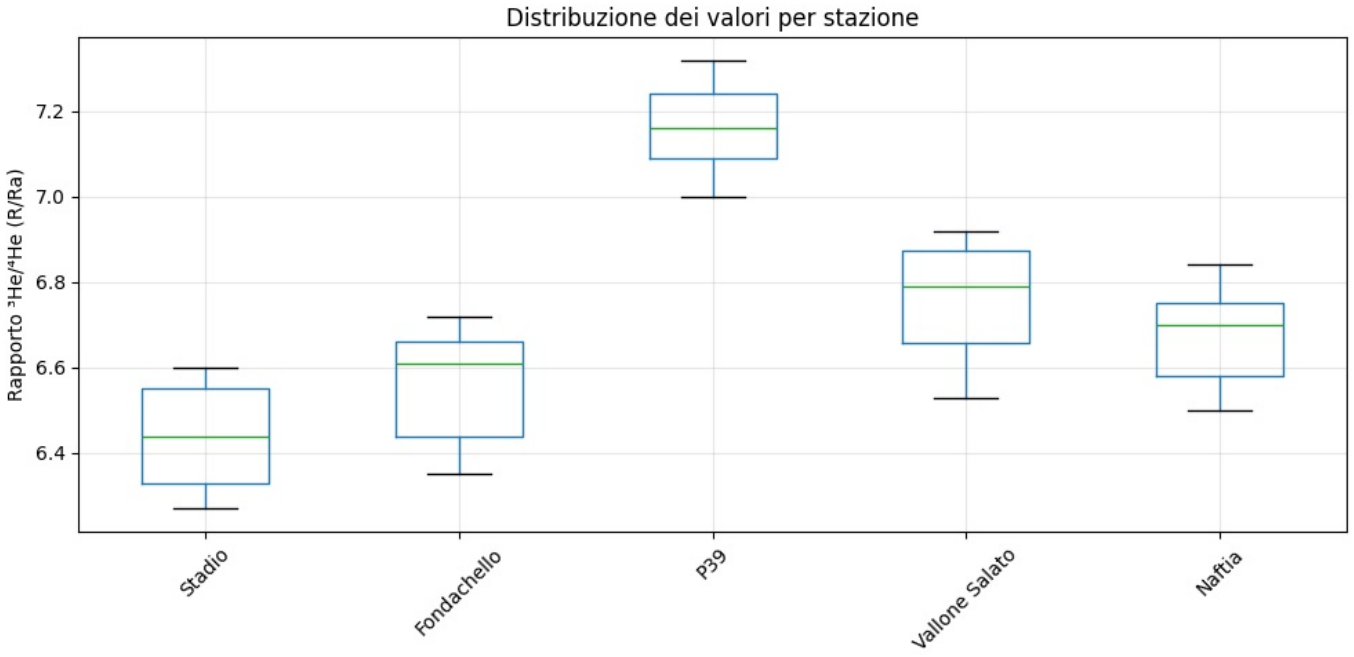
1.4 3HE-4HE_ratio:

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

In [16]:
```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 [17]:
```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:

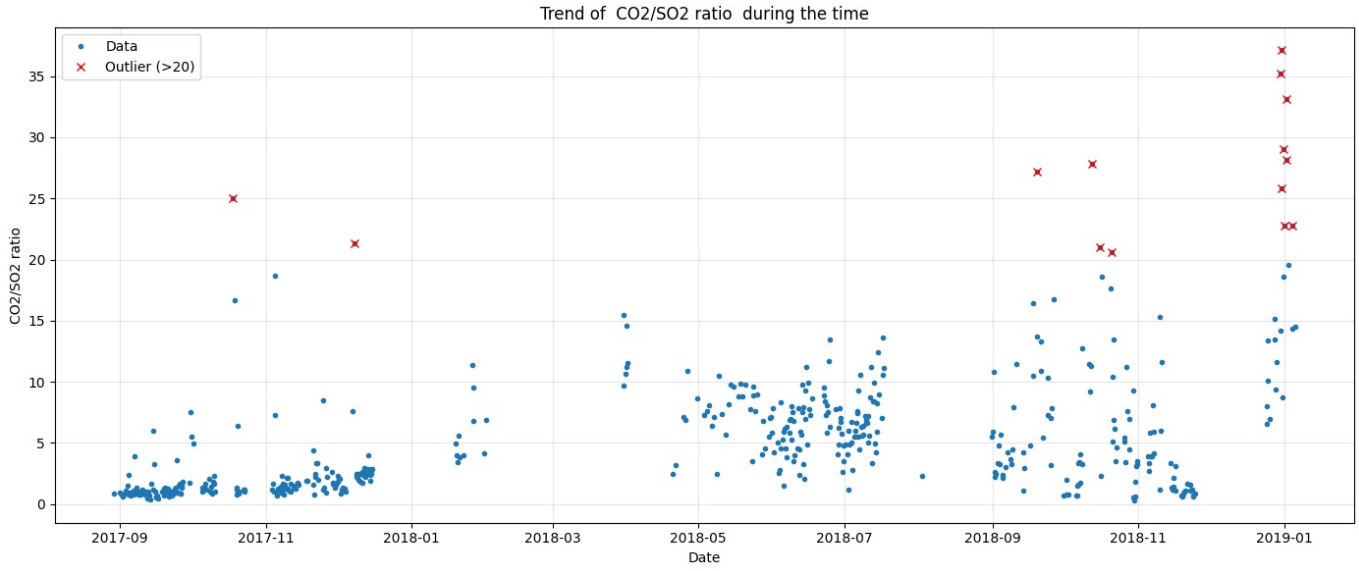
facciamo la stessa cosa qui concentrandoci su una visualizzazione grafica che evidenzi gli outlier

In [18]:
```
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 [19]:
```
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:

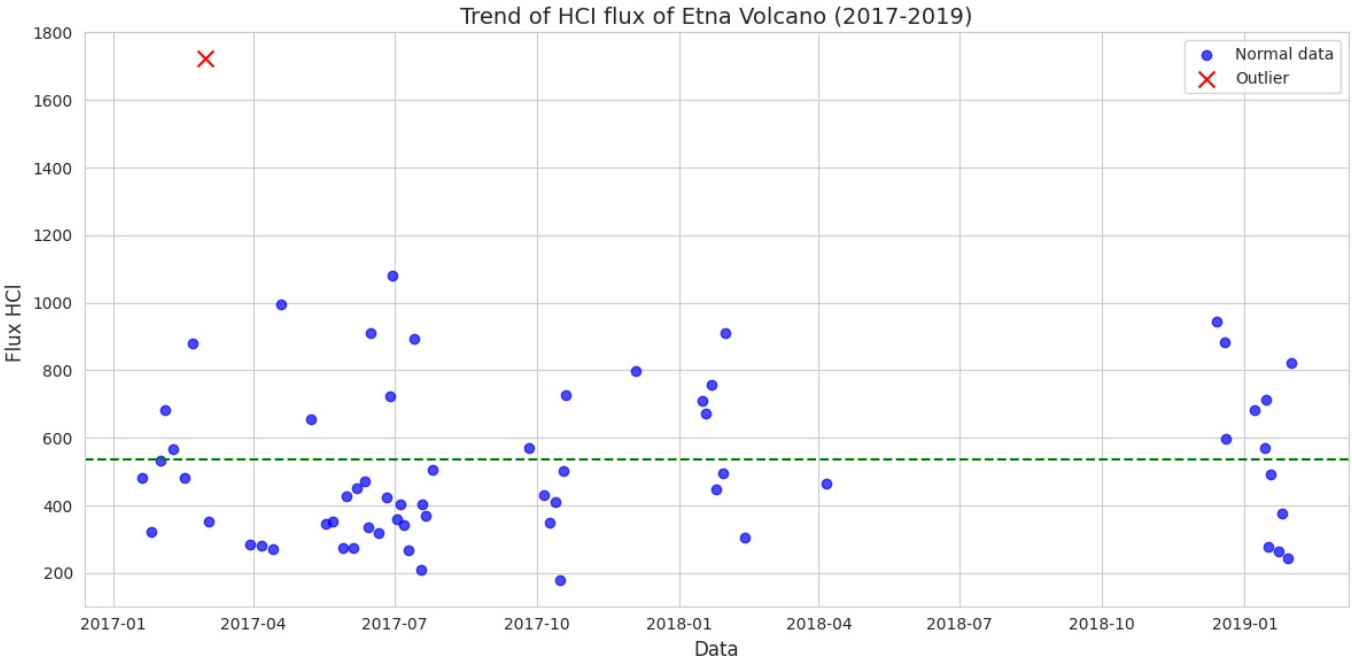
puliamo il file, rinominiamo le colonne, aggiustiamo il formato e poi nello stesso script presentiamo i risultati grafici

In [20]:
```
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:

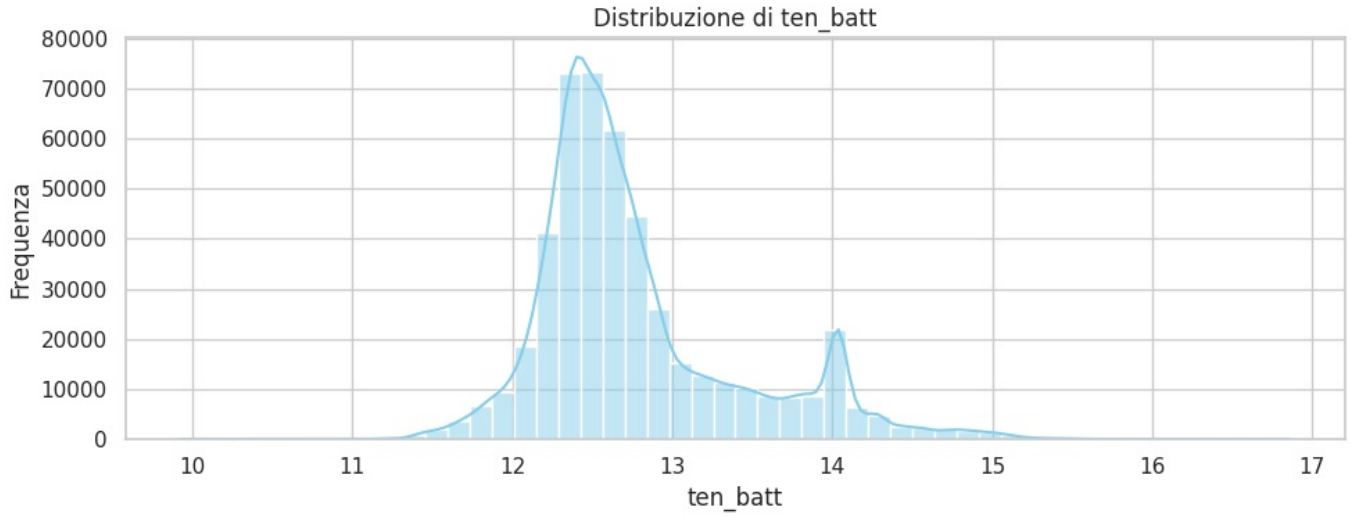
in clinometria si evidenziano leggermente problemi in più, in quanto le colonne non sono separate, i

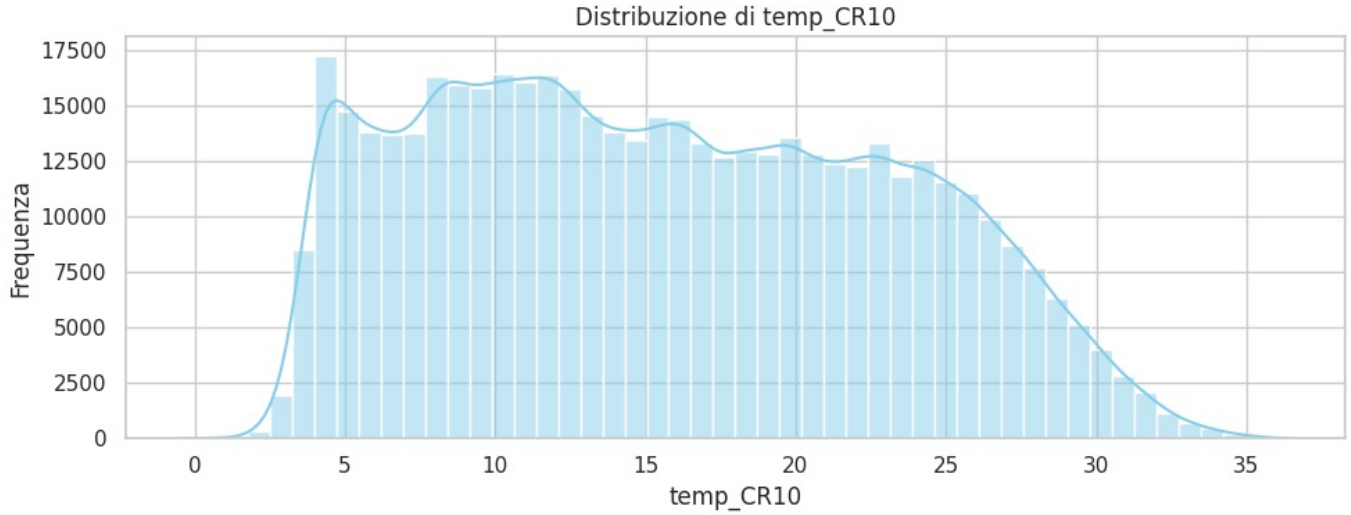
titoli delle colonne non non sono etichettati in caselle diverse, ma sono separati da virgole e in più bisogna adattare le altre operazioni di pulizia, la colonna time al solito, eliminare colonne vuote e salvare il risultato; graficamente rappresento le distribuzioni cosi da capire la differenze tra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

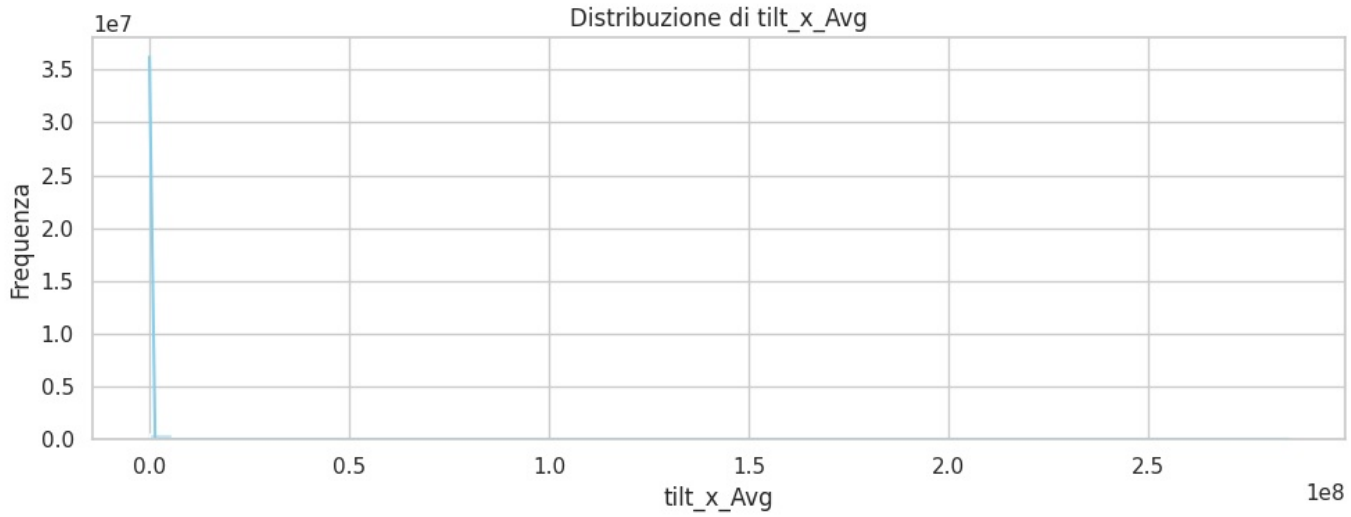
In [21]:
```
# Loading file
raw_df = pd.read_csv("/content/drive/MyDrive/Etna2018/Etna2018.xlsx - clinometria.csv", header=None)


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'")
```

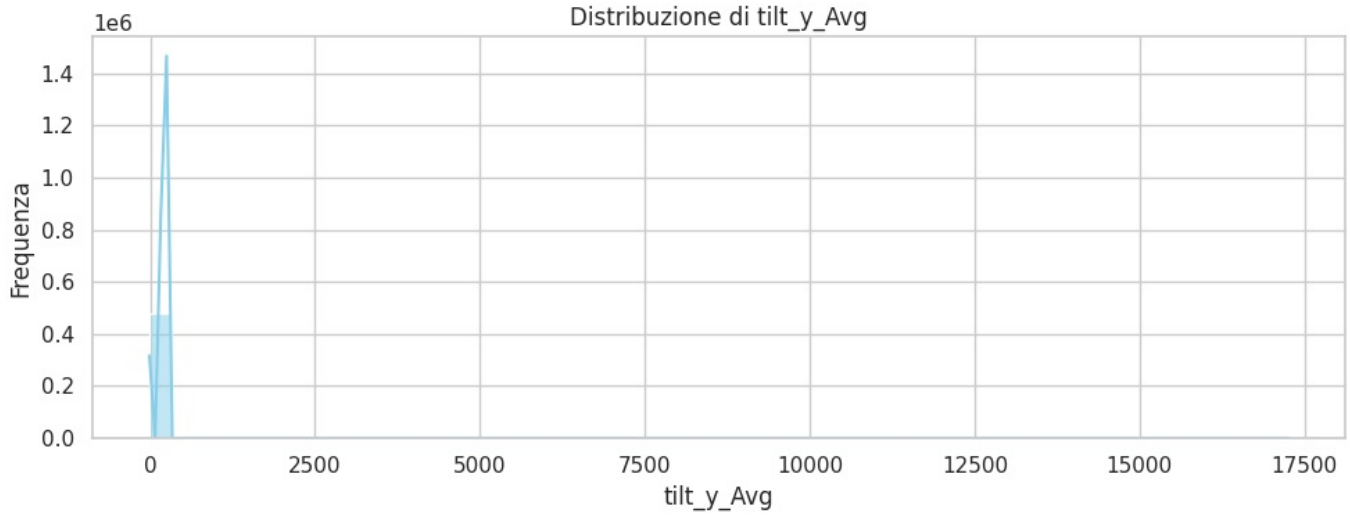
```
Pulizia completata. File salvato come 'clinometria_clean.csv'
```

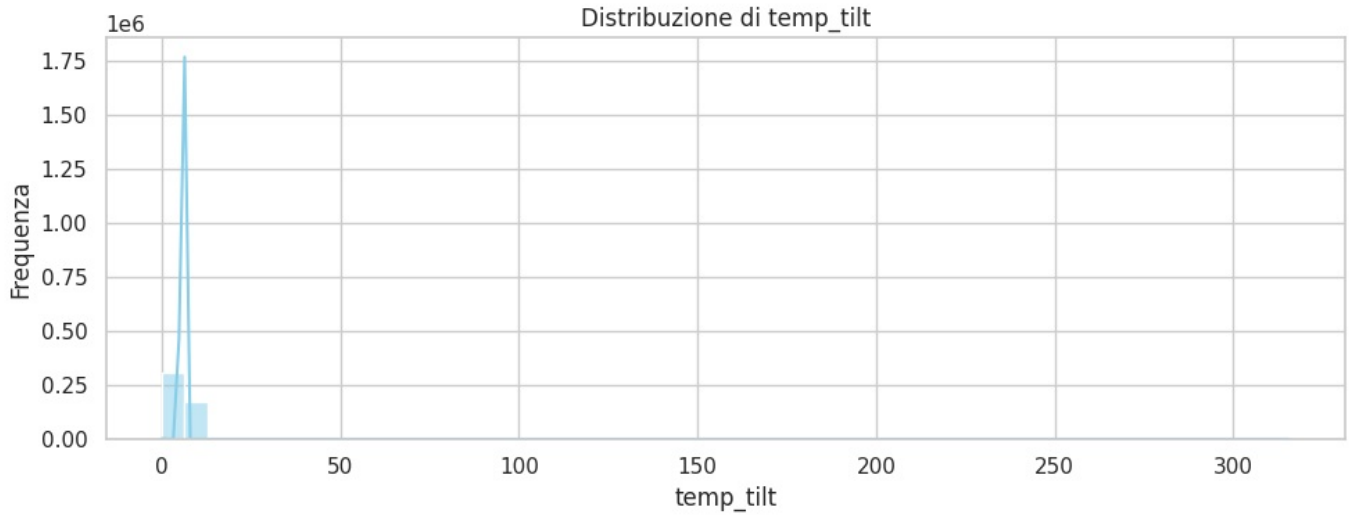
In [22]:
```
# Loading cleaned file
df = pd.read_csv("clinometria_clean.csv")


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()
```

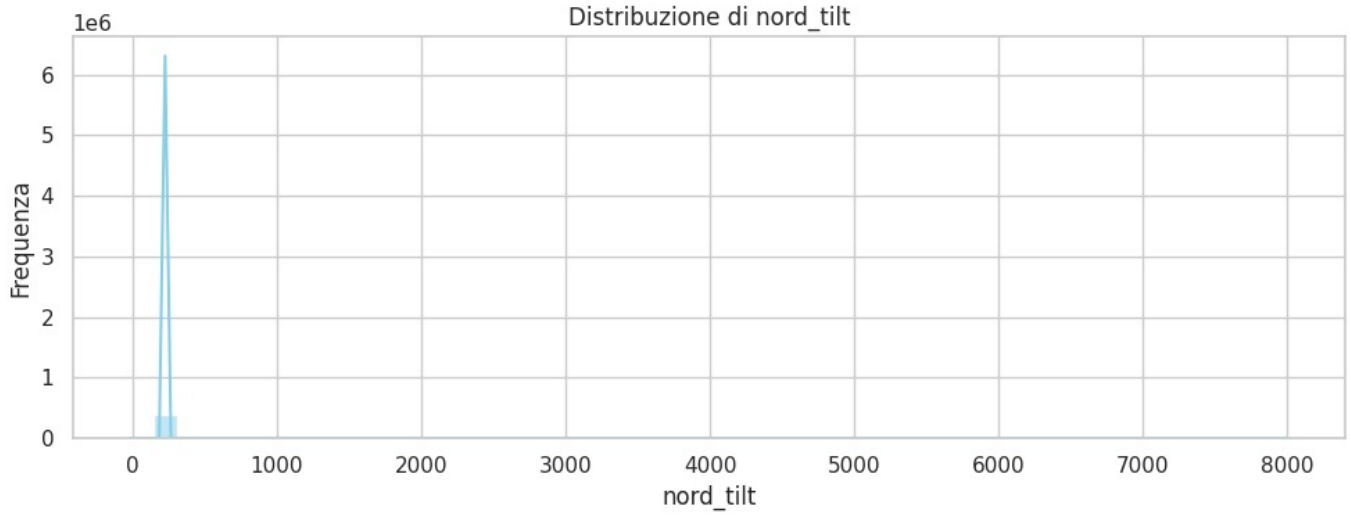

Distribuzione di ten_batt
Frequenza
ten_batt

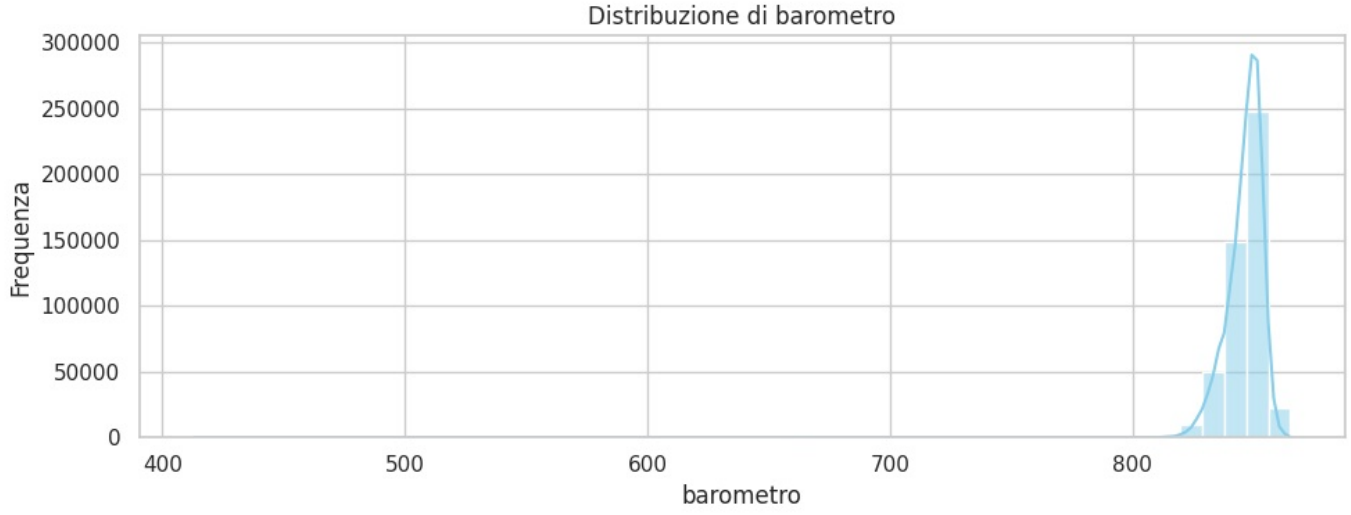

Distribuzione di temp_CR10
Frequenza
temp_CR10

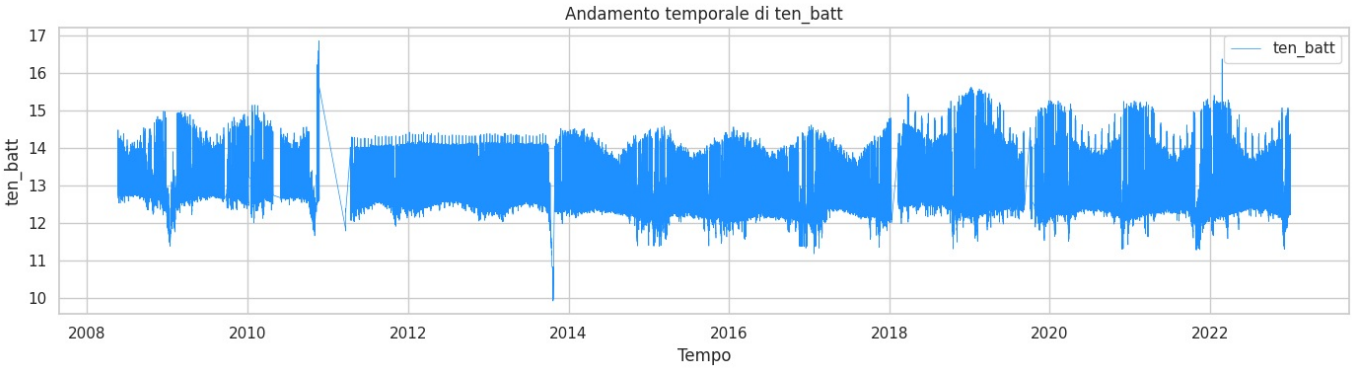

Distribuzione di tilt_x_Avg
1e7
Frequenza
tilt_x_Avg
1e8

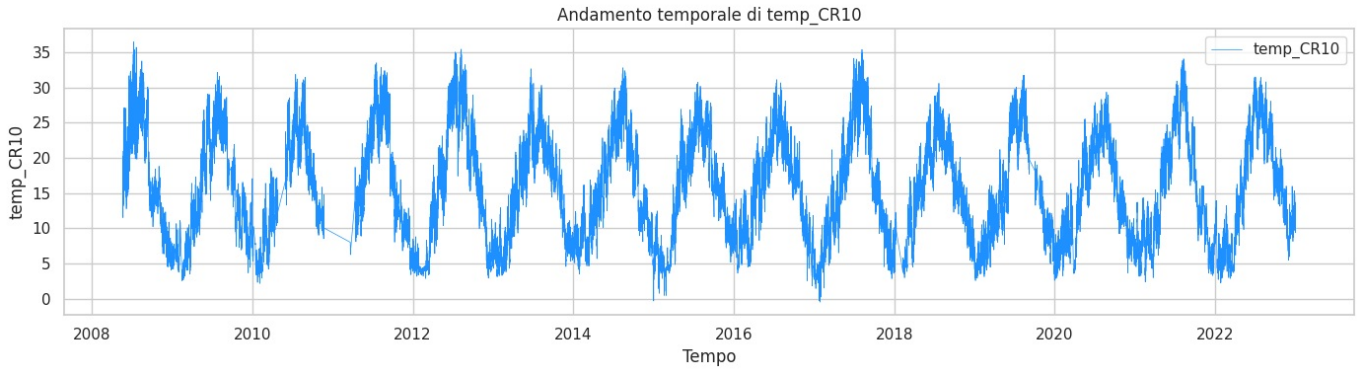

Distribuzione di tilt_y_Avg
1e6
Frequenza
tilt_y_Avg

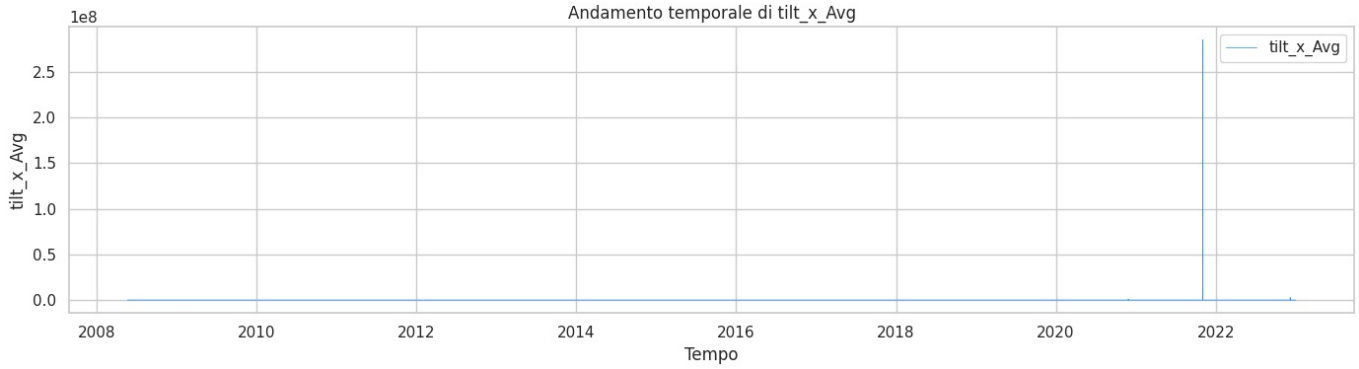

Distribuzione di temp_tilt
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Frequenza
0
50
100
150
200
250
300
temp_tilt

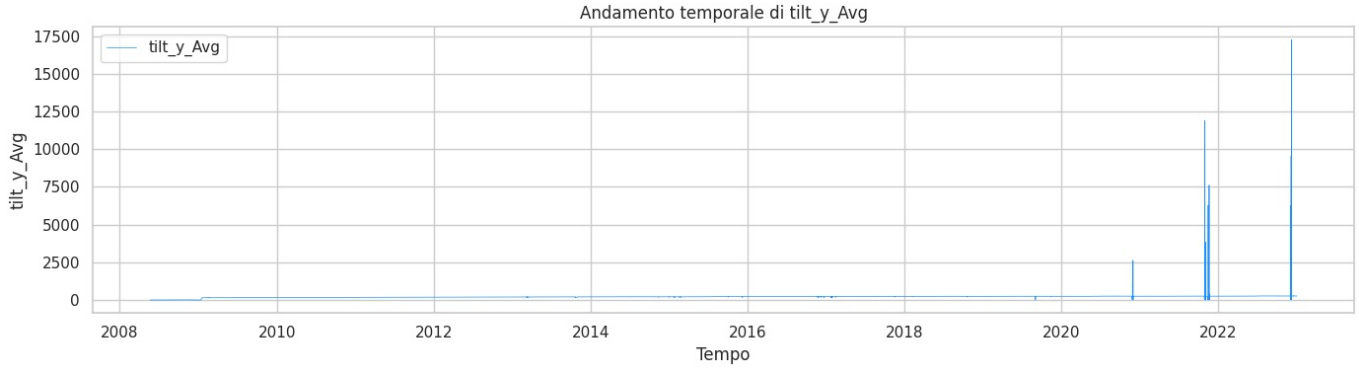

Distribuzione di nord_tilt
1e6
6
5
4
3
2
1
0
Frequenza
0
1000
2000
3000
4000
5000
6000
7000
8000
nord_tilt

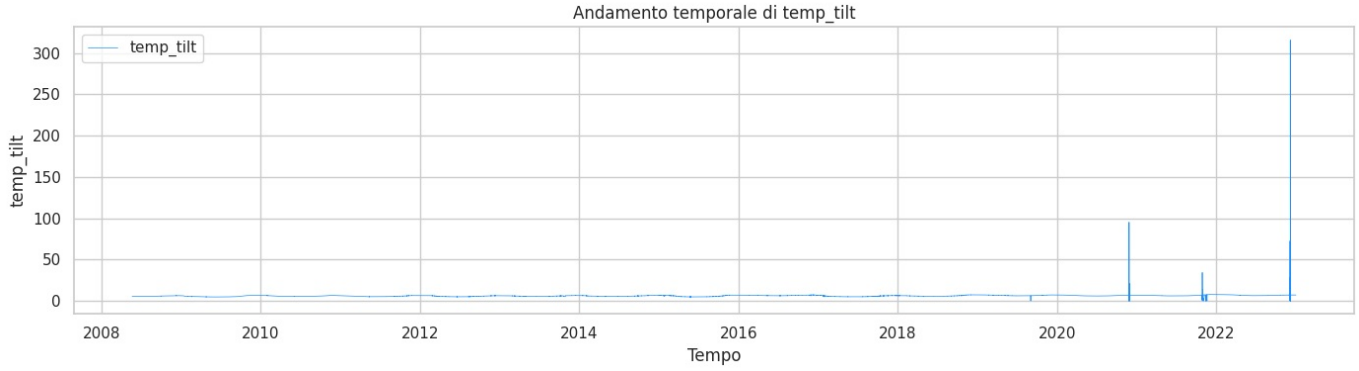

Distribuzione di barometro
300000
250000
200000
150000
100000
50000
0
Frequenza
400
500
600
700
800
barometro

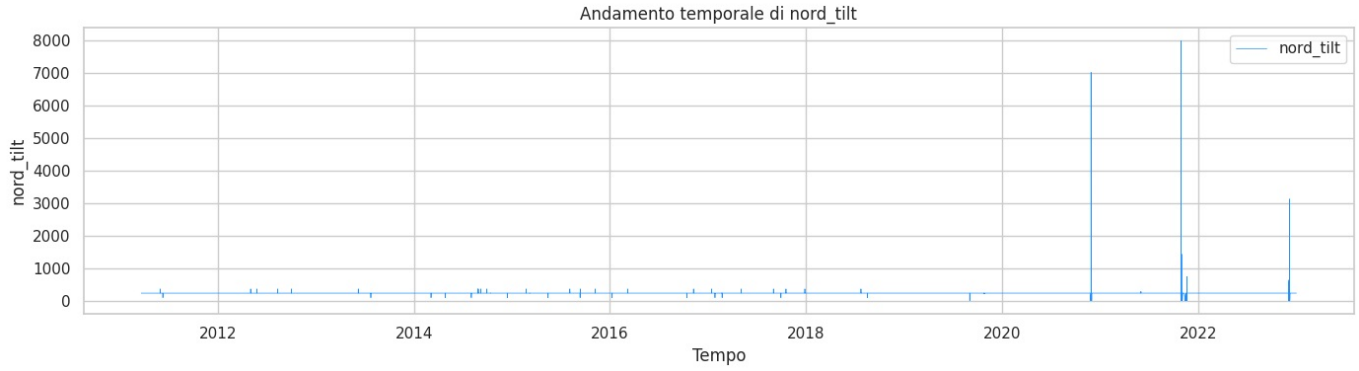

Andamento temporale di ten_batt
ten_batt
17
16
15
14
13
12
11
10
ten_batt
2008
2010
2012
2014
2016
2018
2020
2022
Tempo

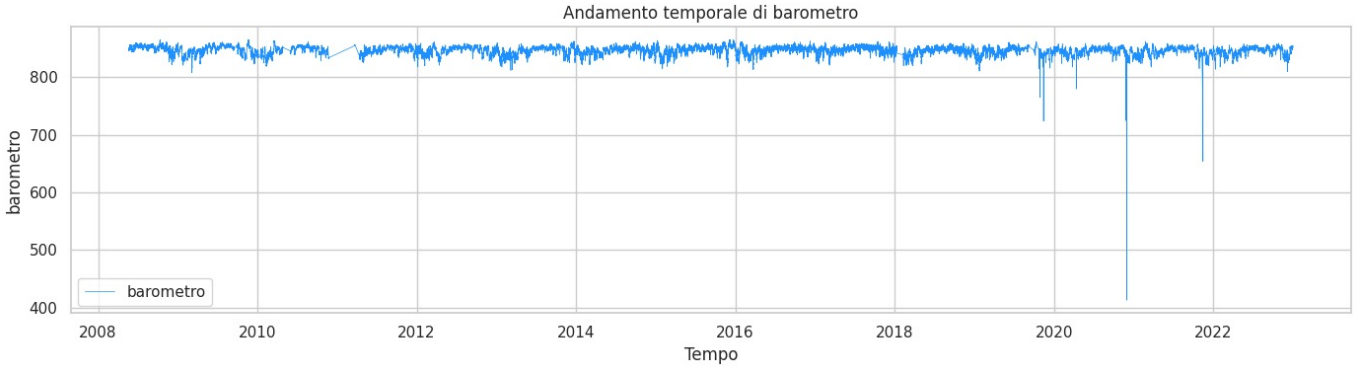

Andamento temporale di temp_CR10
temp_CR10
temp_CR10
Tempo


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 [23]:
```
#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:

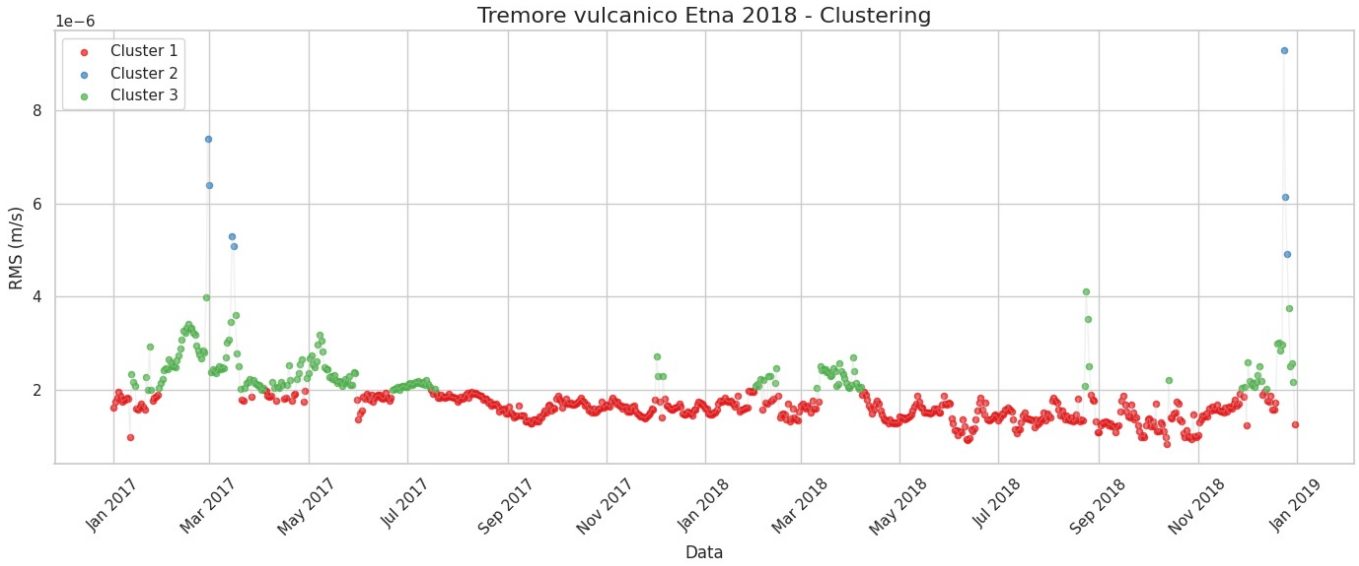
utilizzo RMS come misura per la visualizzazione grafica e mostro una suddivisione in cluster per distinguere bene i periodi di quiete con quelli più movimentati

In [24]:
```
# 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'")
```

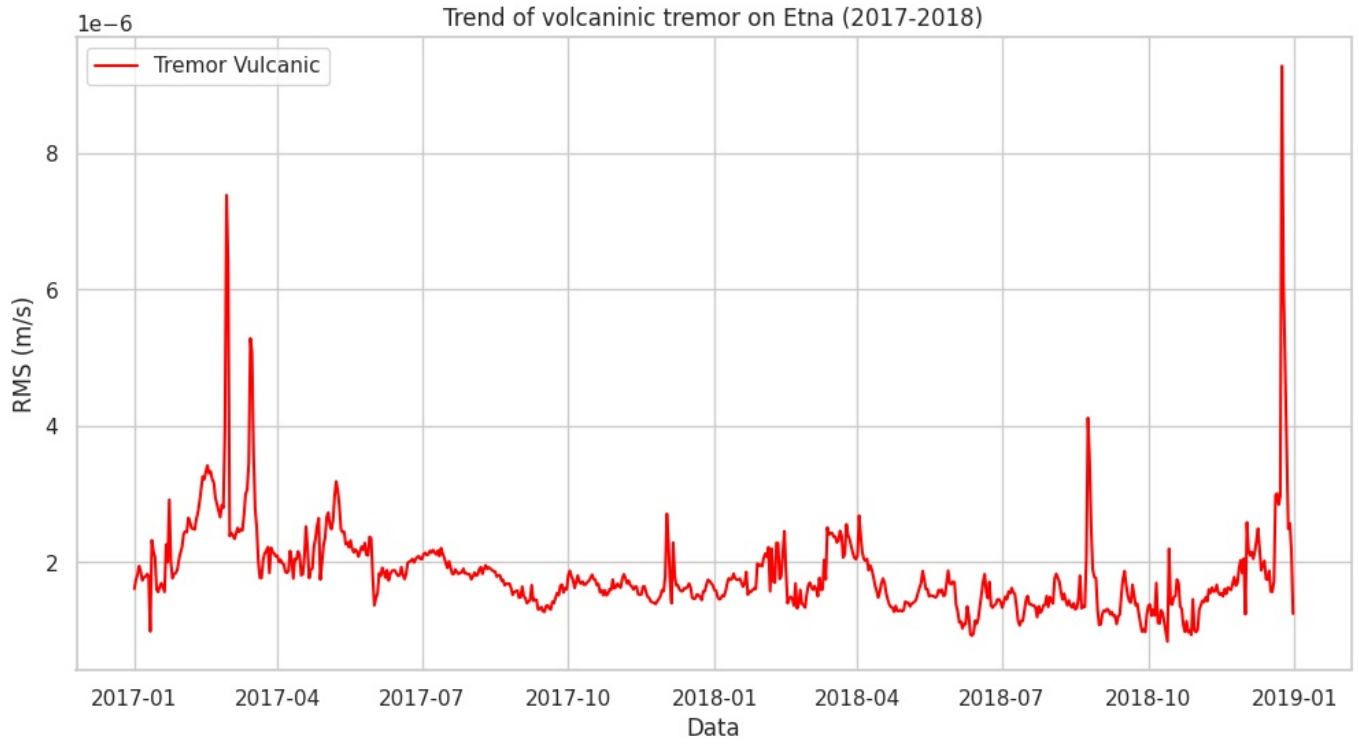
```
Missing values first:
 Data    0
RMS     0
dtype: int64
Duplicated rows: 0
Missing values in RMS:
 Empty DataFrame
Columns: [Data, RMS]
Index: []

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 [25]:

```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 [26]:
```
# 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 [27]:
```
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'
```

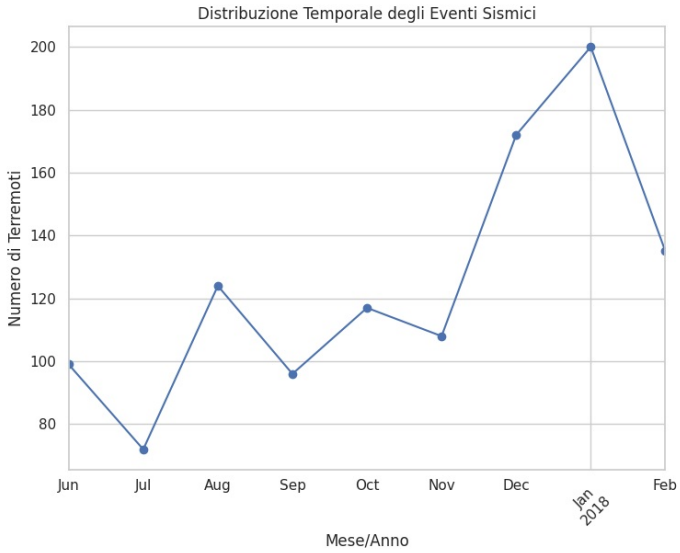
In [28]:
```
def mostra_posizione(latitudine, longitudine):
    # Creating the map centered on coordinates
    mappa = folium.Map(
        location=[latitudine, longitudine],
        zoom_start=15,
        tiles="OpenStreetMap"
    )

    # 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")
```

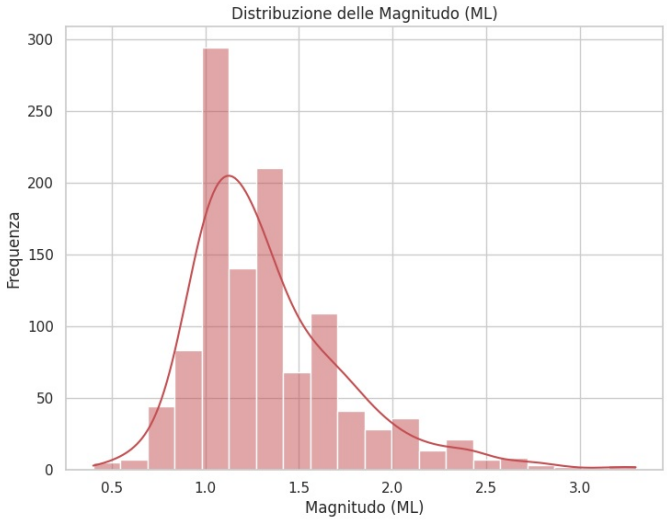
```python
# Example of usage
if __name__ == "__main__":
    print("Insert geographic coordinates")

    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.")
```

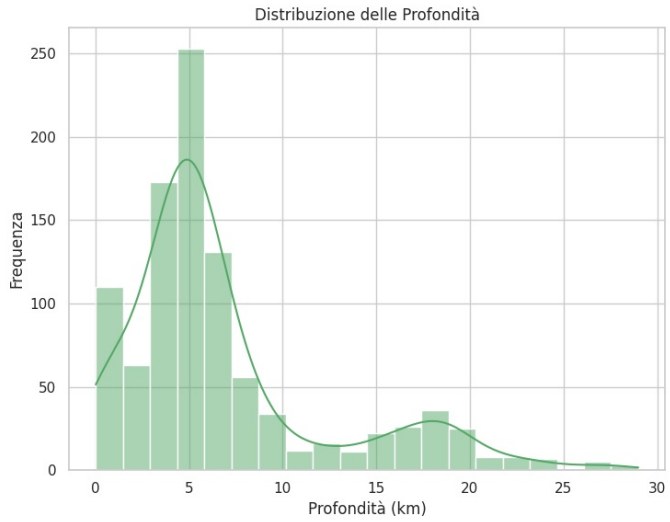
```
Insert geographic coordinates
Latitude (es. 37.7749): 37.592867
Longitude (es. -122.4194): 15.133191
Map generated with success! Open file 'mappa_37.592867_15.133191.html' here on the left
```

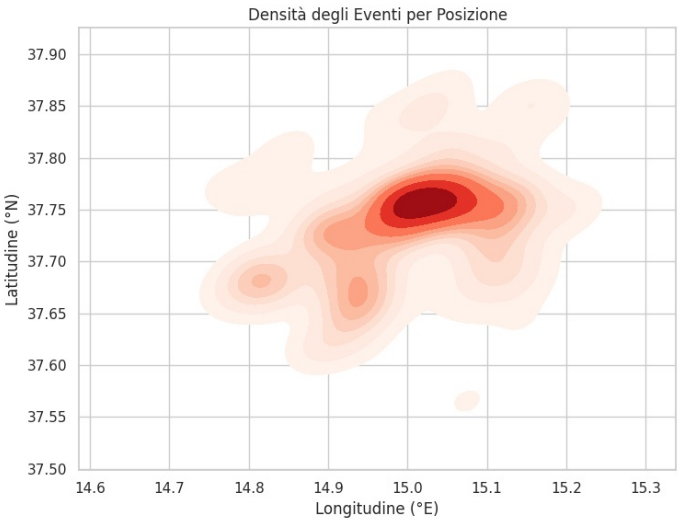
In [29]:

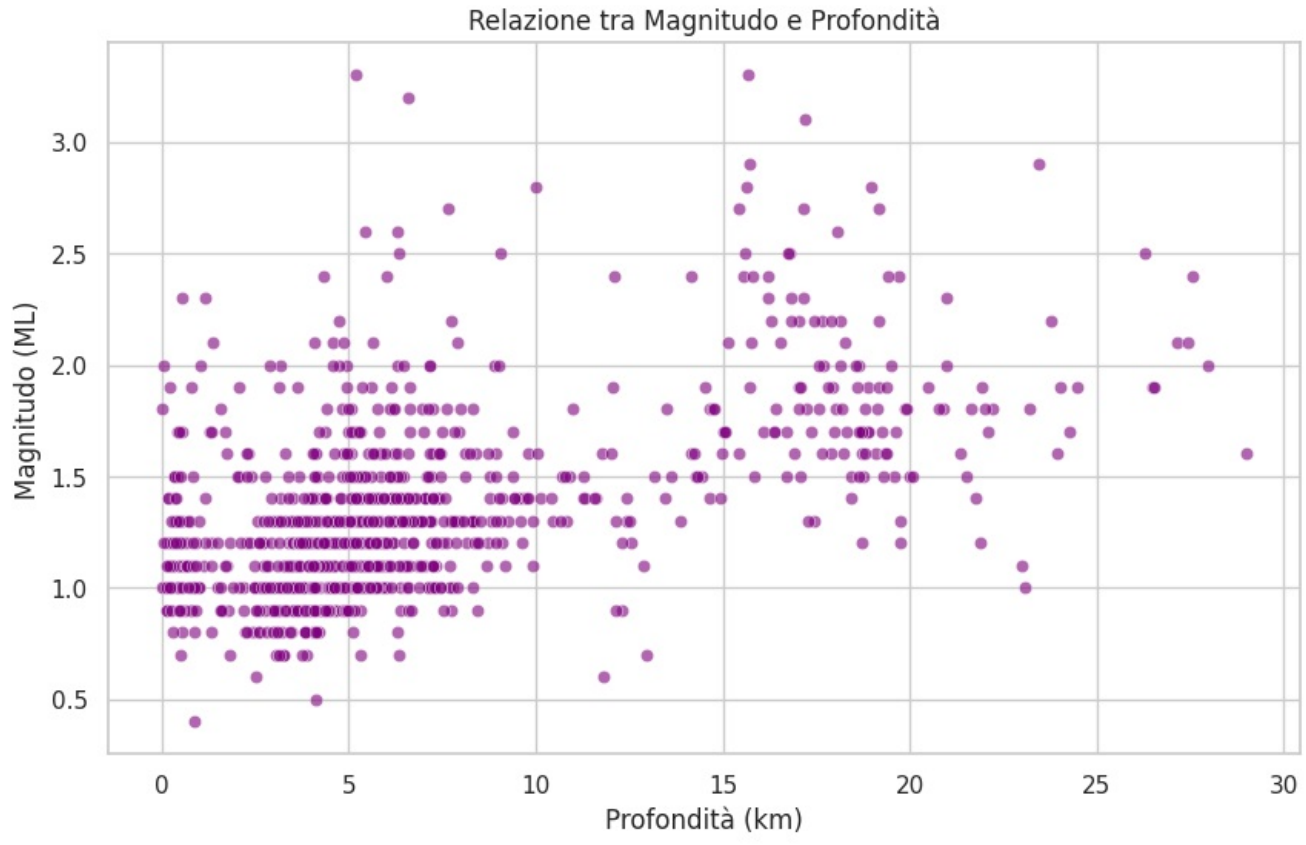
```python
def plot_earthquake_distribution(cleaned_data_path):
    # Loading data
    df = pd.read_csv(cleaned_data_path, parse_dates=['Datetime'])


    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 [30]:
```
# 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 [31]:

```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")
```

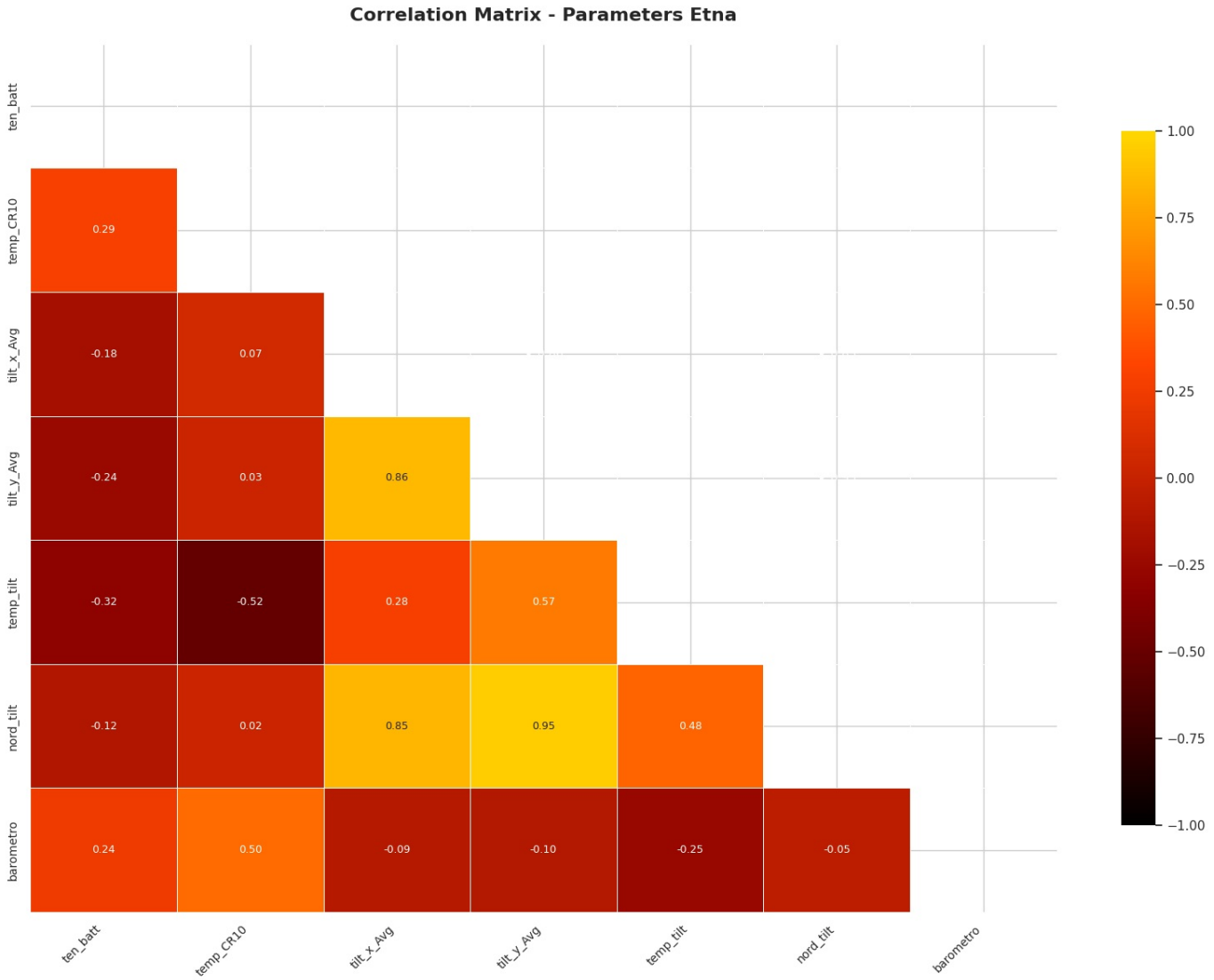
```
Periodo originale: 2008-05-21 08:30:00 → 2022-12-31 23:45:00
Periodo filtrato: 2017-01-01 00:00:00 → 2019-12-31 23:45:00
Dimensione: (70300, 27)
Colonne numeriche con copertura sufficiente:
['ten_batt', 'temp_CR10', 'tilt_x_Avg', 'tilt_y_Avg', 'temp_tilt', 'nord_tilt', 'barometro']
File salvato: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

2.1 Correlation matrix sul dataframe unito

In [32]:

```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")
```



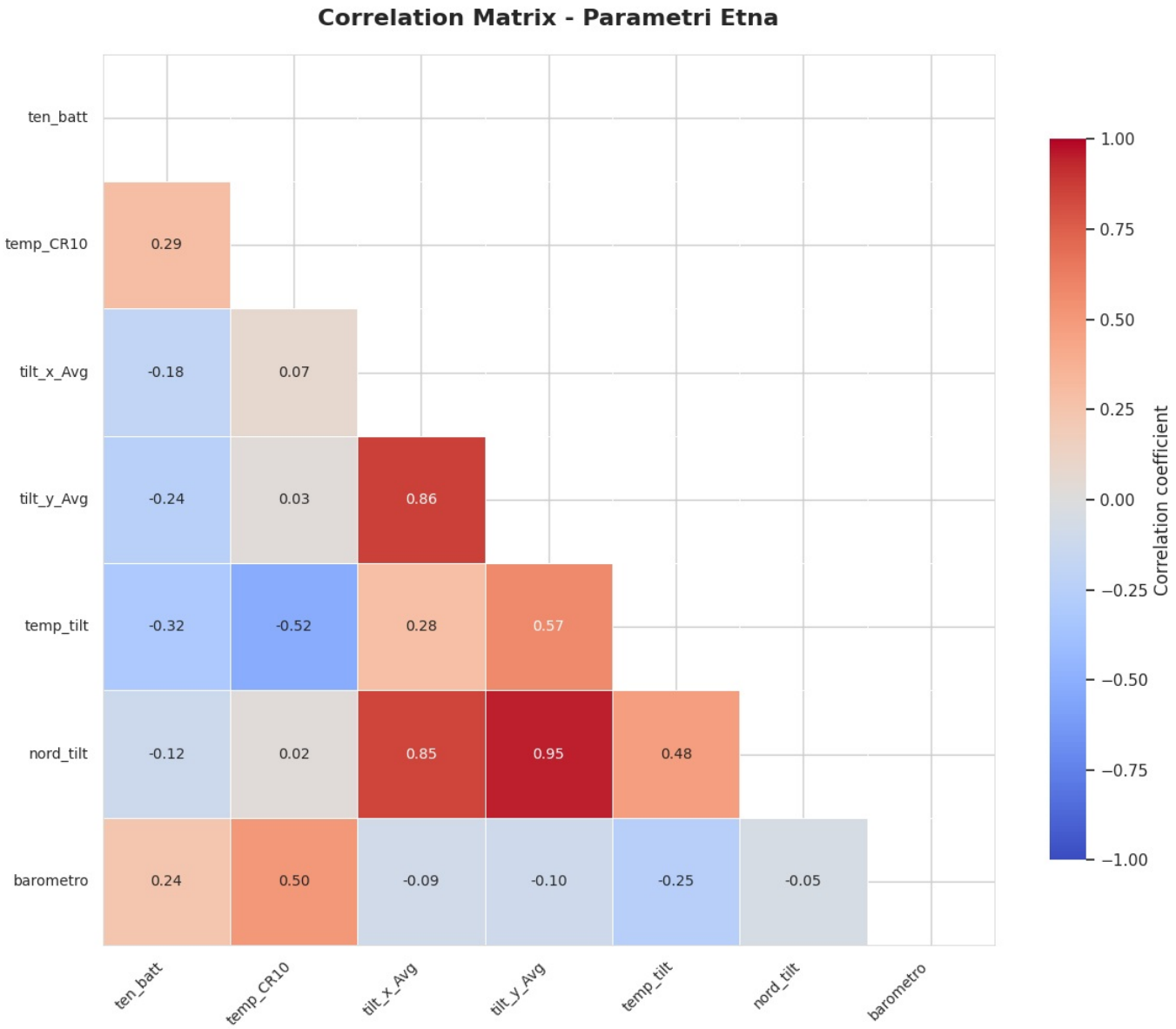

```
Completed analysis. Saved files:
- correlation_matrix_etna.png
- strong_correlations.csv
```

In [33]:
```
# 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()
```

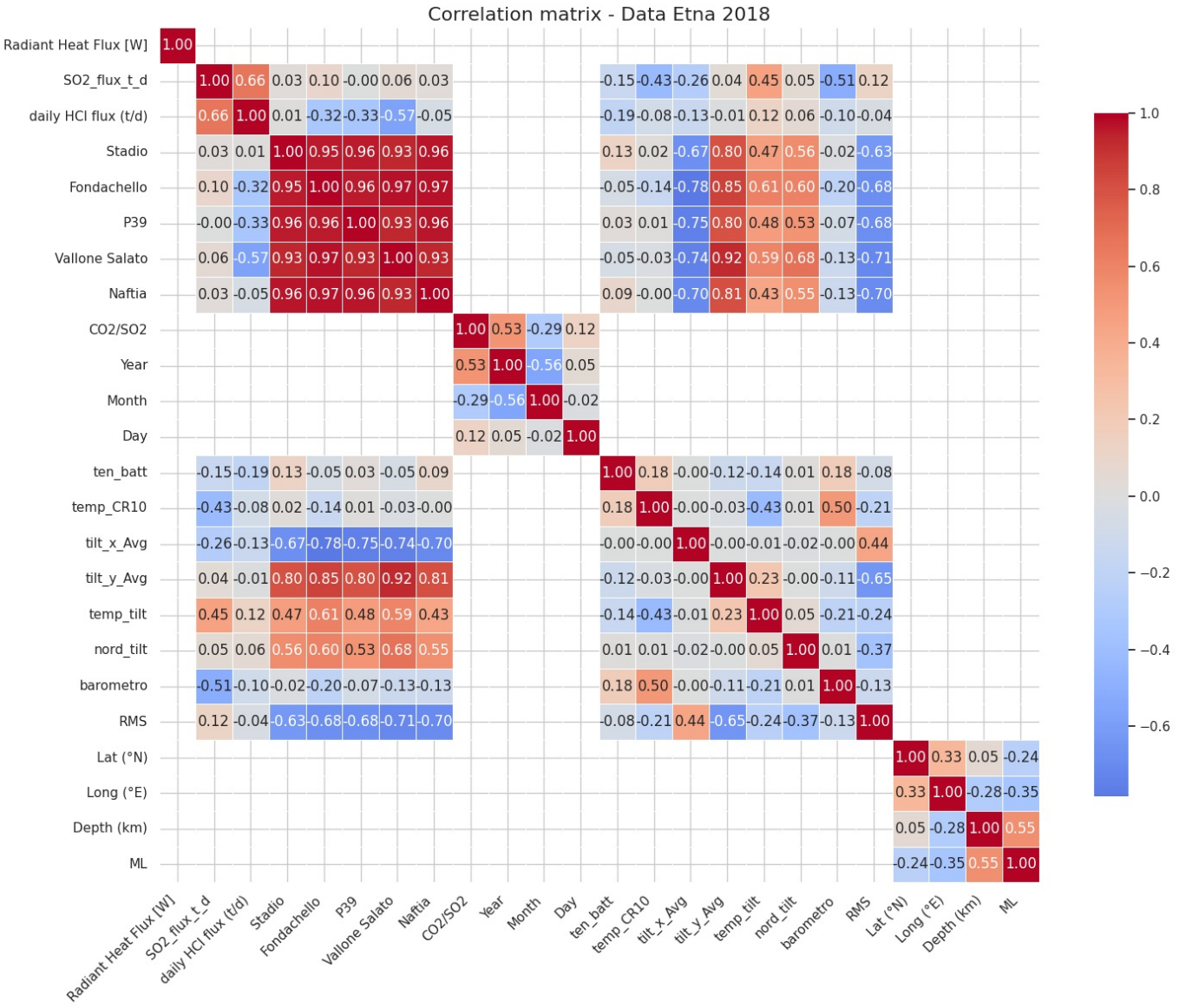
```
Loading data...

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 [34]:
```
# 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',
```

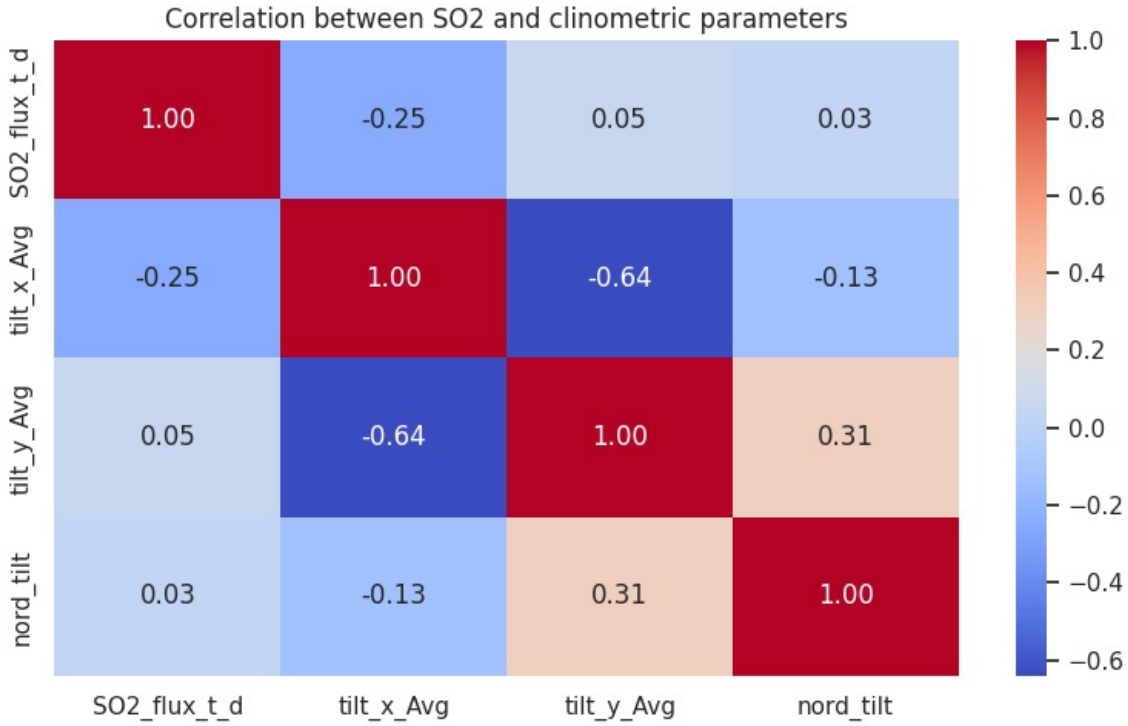
```
            center=0,
            linewidths=0.5,
            cbar_kws={"shrink": 0.8})

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'")
```



```
Strongest correlations (|correlation| > 0.7):
Vallone Salato  Fondachello       0.970715
Fondachello     Vallone Salato    0.970715
Naftia          Fondachello       0.965663
Fondachello     Naftia            0.965663
Stadio          Naftia            0.961481
Naftia          Stadio            0.961481
                P39               0.960085
P39             Naftia            0.960085
                Stadio            0.957299
Stadio          P39               0.957299
Fondachello     P39               0.956407
P39             Fondachello       0.956407
Stadio          Fondachello       0.953457
Fondachello     Stadio            0.953457
Stadio          Vallone Salato    0.927436
Vallone Salato  Stadio            0.927436
                P39               0.926112
P39             Vallone Salato    0.926112
Naftia          Vallone Salato    0.925775
Vallone Salato  Naftia            0.925775
                tilt_y_Avg        0.918534
tilt_y_Avg      Vallone Salato    0.918534
                Fondachello       0.851123
Fondachello     tilt_y_Avg        0.851123
Naftia          tilt_y_Avg        0.810645
tilt_y_Avg      Naftia            0.810645
                P39               0.800561
P39             tilt_y_Avg        0.800561
Stadio          tilt_y_Avg        0.795898
tilt_y_Avg      Stadio            0.795898
RMS             Naftia           -0.700137
Naftia          RMS              -0.700137
                tilt_x_Avg       -0.704042
tilt_x_Avg      Naftia           -0.704042
RMS             Vallone Salato   -0.708611
Vallone Salato  RMS              -0.708611
                tilt_x_Avg       -0.740772
tilt_x_Avg      Vallone Salato   -0.740772
                P39              -0.754351
P39             tilt_x_Avg       -0.754351
Fondachello     tilt_x_Avg       -0.784875
tilt_x_Avg      Fondachello      -0.784875
dtype: float64

Correlation matrix saved as 'correlation_matrix_etna.csv'
```

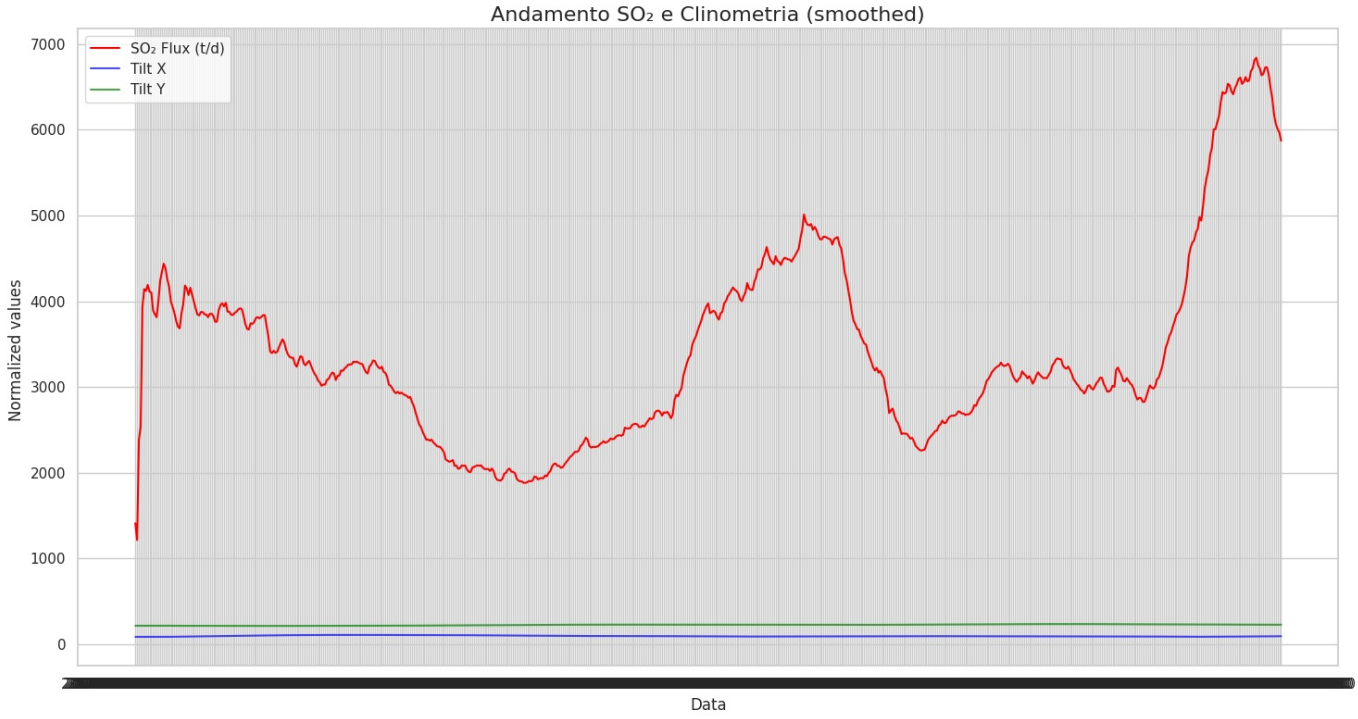
In [35]:

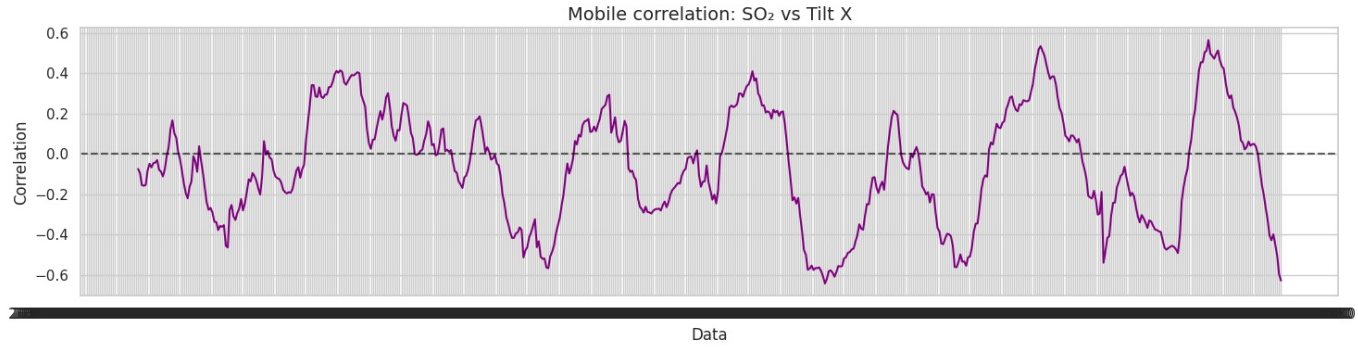
```python
# --- Loading SO2 data ---
so2 = pd.read_csv('SO2_flux_cleaned.csv')
so2['date'] = pd.to_datetime(so2['date'])
so2.set_index('date', inplace=True)

# --- 2. Loading Clinometria data ---
clin = pd.read_csv('clinometria_clean.csv')
clin['time'] = pd.to_datetime(clin['time'], errors='coerce')
clin.set_index('time', inplace=True)

# --- 3. Selecting only numeric columns interesting ---
clinometric_cols = ['tilt_x_Avg', 'tilt_y_Avg', 'nord_tilt']
clin = clin[clinometric_cols]

# --- 4. Daily resampling ---
clin_daily = clin.resample('D').mean()

# --- 5. Merging both dataset on datetime ---
merged = so2.join(clin_daily, how='inner')

# --- 6. Computing correlation ---
corr_matrix = merged.corr()

# --- 7. Visualization of Heatmap ---
plt.figure(figsize=(8, 5))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation between SO2 and clinometric parameters")
plt.tight_layout()
plt.show()
```



In [36]:

```python
# 1. Loading unified file
df = pd.read_csv('merged_etna_data.csv', index_col=0, parse_dates=True)

# 2. Base cleaning
df = df[['SO2_flux_t_d', 'tilt_x_Avg', 'tilt_y_Avg']]
df = df.dropna()
df = df[df['SO2_flux_t_d'] > 0]

# 3. Smooth with mobile mean
window = 24 * 2
df_smooth = df.rolling(window=window, min_periods=1).mean()

# 4. Wave syncronized graph
plt.figure(figsize=(15, 8))
plt.plot(df_smooth.index, df_smooth['SO2_flux_t_d'], label='SO Flux (t/d)', color='red')
plt.plot(df_smooth.index, df_smooth['tilt_x_Avg'], label='Tilt X', color='blue', alpha=0.7)
plt.plot(df_smooth.index, df_smooth['tilt_y_Avg'], label='Tilt Y', color='green', alpha=0.7)
plt.title("Andamento SO e Clinometria (smoothed)", fontsize=16)
plt.ylabel("Normalized values ")
plt.xlabel("Data")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# 5. Analysis of temporal correlation
rolling_corr = df['SO2_flux_t_d'].rolling(window=window).corr(df['tilt_x_Avg'])

plt.figure(figsize=(15, 4))
plt.plot(rolling_corr.index, rolling_corr, color='purple')
plt.title("Mobile correlation: SO vs Tilt X", fontsize=14)
plt.xlabel("Data")
plt.ylabel("Correlation")
plt.grid(True)
plt.axhline(0, color='black', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

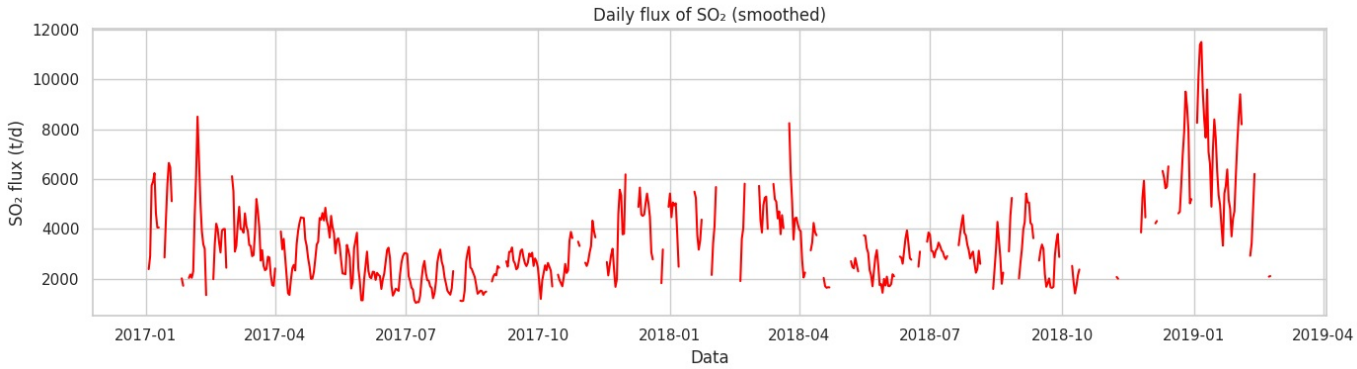


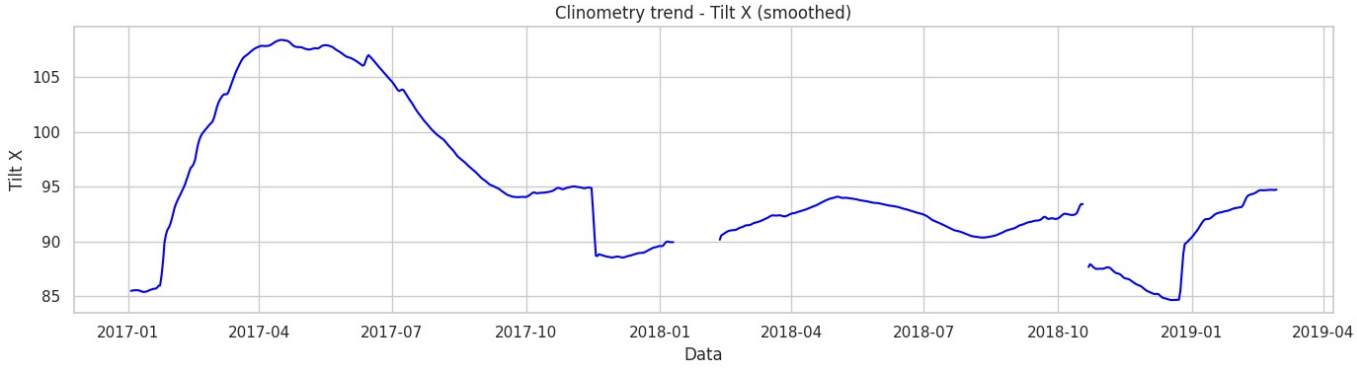


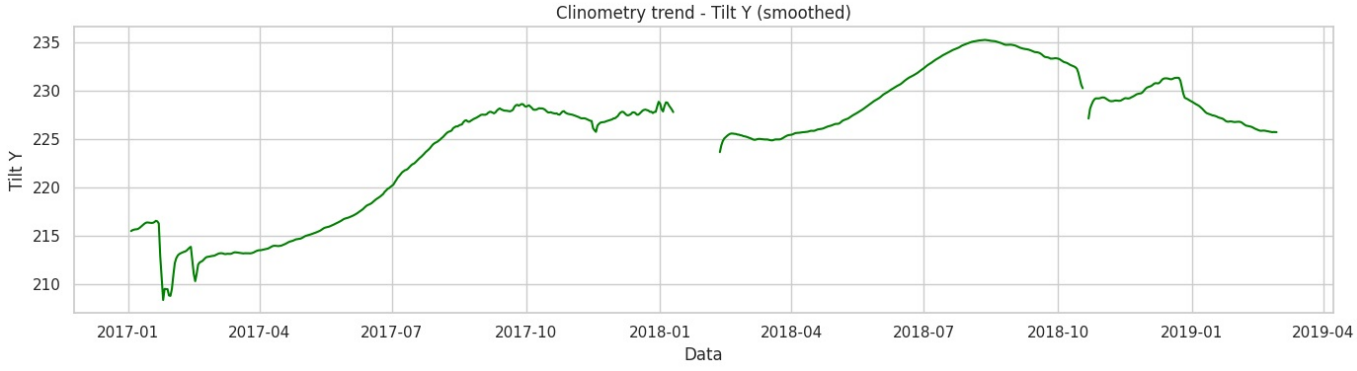
In [37]:

```python
# 1. Load the two files
so2_df = pd.read_csv("SO2_flux_cleaned.csv", parse_dates=["date"])
clino_df = pd.read_csv("clinometria_clean.csv", parse_dates=["time"])

# 2. Rename column for uniformity
so2_df = so2_df.rename(columns={"date": "Date", "SO2_flux_t_d": "SO2_flux"})
clino_df = clino_df.rename(columns={"time": "Date"})

# 3. Resample clinometry
clino_df = clino_df.set_index("Date").resample("D").mean().reset_index()

# 4. Temporal merge
merged = pd.merge(so2_df, clino_df, on="Date", how="inner")

# 5. Data smooth
window = 3
merged['SO2_flux_smooth'] = merged['SO2_flux'].rolling(window).mean()
merged['tilt_x_smooth'] = merged['tilt_x_Avg'].rolling(window).mean()
merged['tilt_y_smooth'] = merged['tilt_y_Avg'].rolling(window).mean()

# 6. Separated graph:S02
plt.figure(figsize=(14, 4))
plt.plot(merged['Date'], merged['SO2_flux_smooth'], color='red')
plt.title("Daily flux of SO (smoothed)")
plt.xlabel("Data")
plt.ylabel("SO flux (t/d)")
plt.grid(True)
plt.tight_layout()
plt.show()

# 7. Separated graph: Tilt X
plt.figure(figsize=(14, 4))
plt.plot(merged['Date'], merged['tilt_x_smooth'], color='blue')
plt.title("Clinometry trend - Tilt X (smoothed)")
plt.xlabel("Data")
plt.ylabel("Tilt X")
plt.grid(True)
plt.tight_layout()
plt.show()

# 8. Separated graph: Tilt Y
```

```
plt.figure(figsize=(14, 4))
plt.plot(merged['Date'], merged['tilt_y_smooth'], color='green')
plt.title("Clinometry trend - Tilt Y (smoothed)")
plt.xlabel("Data")
plt.ylabel("Tilt Y")
plt.grid(True)
plt.tight_layout()
plt.show()

# 9. Correlation between SO2 e tilt X/Y
corr_x = merged['SO2_flux'].corr(merged['tilt_x_Avg'])
corr_y = merged['SO2_flux'].corr(merged['tilt_y_Avg'])

print(f"Correlation between SO - Tilt X: {corr_x:.2f}")
print(f"Correlation between - Tilt Y: {corr_y:.2f}")
```







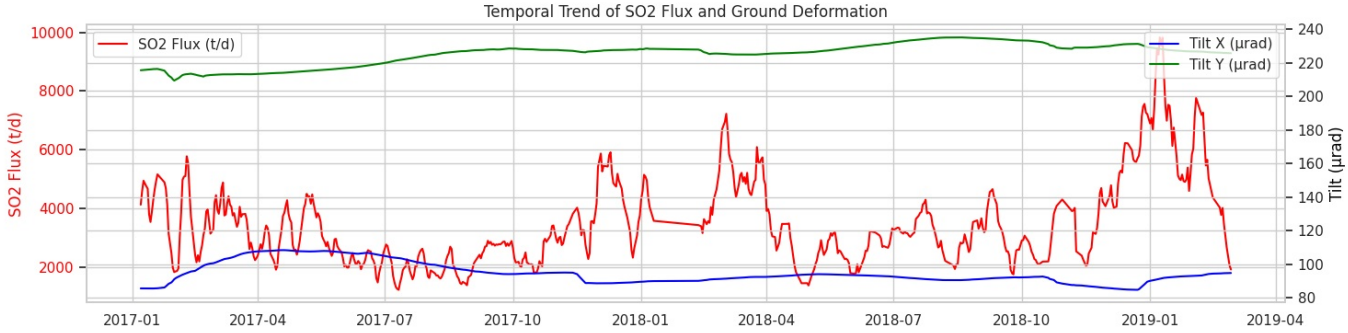

```
Correlation between SO - Tilt X: -0.25
Correlation between - Tilt Y: 0.05
```

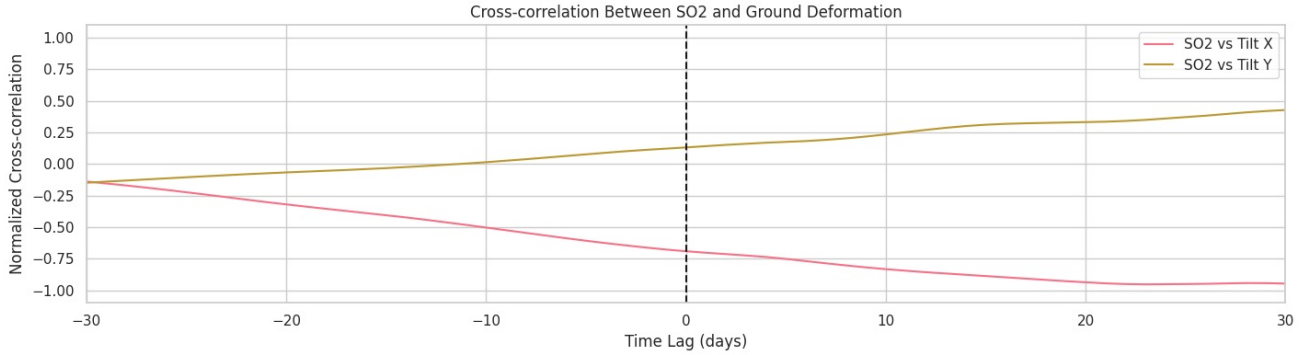
In [38]:
```
# Set plotting style and parameters
sns.set_style("whitegrid")  # Changed from plt.style.use('seaborn') to avoid error
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (15, 8)

def load_and_prepare_data():
    """
    Load and prepare SO2 flux and clinometer (tilt) data for analysis.

    Returns:
        pd.DataFrame: Merged dataset with smoothed time series
    """
    # Load SO2 flux data
    so2 = pd.read_csv('SO2_flux_cleaned.csv', parse_dates=['date'])
    so2.set_index('date', inplace=True)
    so2 = so2.resample('D').mean()  # Daily resampling

    # Load clinometer data (ground deformation measurements)
    clin = pd.read_csv('clinometria_clean.csv', parse_dates=['time'])
    clin.set_index('time', inplace=True)
    clin = clin[['tilt_x_Avg', 'tilt_y_Avg']].resample('D').mean()  # Keep only tilt columns
```

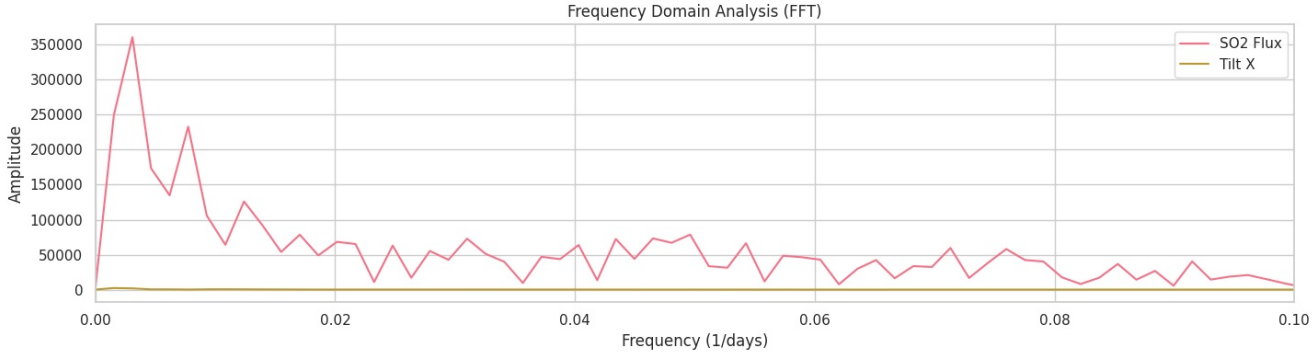
```
    # Merge datasets on date index
    merged = pd.concat([so2, clin], axis=1).dropna()
    merged.columns = ['SO2_flux', 'tilt_x', 'tilt_y']

    # Apply rolling mean to smooth the data (7-day window)
    window_size = 7
    merged['SO2_smoothed'] = merged['SO2_flux'].rolling(window=window_size).mean()
    merged['tilt_x_smoothed'] = merged['tilt_x'].rolling(window=window_size).mean()
    merged['tilt_y_smoothed'] = merged['tilt_y'].rolling(window=window_size).mean()

    return merged.dropna(), window_size

def analyze_waveforms(df,window_size):
    """
    Calculate cross-correlation between SO2 emissions and ground deformation.

    Args:
        df (pd.DataFrame): Prepared dataset with smoothed time series

    Returns:
        tuple: Normalized cross-correlation arrays for x and y tilt components
    """
    # Calculate cross-correlation between SO2 and tilt x
    xcorr_x = signal.correlate(df['SO2_smoothed'] - df['SO2_smoothed'].mean(),
                               df['tilt_x_smoothed'] - df['tilt_x_smoothed'].mean(),
                               mode='same', method='auto') / (window_size * df['SO2_smoothed'].std() * df['tilt_x

    # Calculate cross-correlation between SO2 and tilt y
    xcorr_y = signal.correlate(df['SO2_smoothed'] - df['SO2_smoothed'].mean(),
                               df['tilt_y_smoothed'] - df['tilt_y_smoothed'].mean(),
                               mode='same', method='auto') / (window_size * df['SO2_smoothed'].std() * df['tilt_y

    # Normalize cross-correlation values
    xcorr_x /= np.max(np.abs(xcorr_x))
    xcorr_y /= np.max(np.abs(xcorr_y))

    return xcorr_x, xcorr_y

def plot_results(df, xcorr_x, xcorr_y):
    """
    Visualize the relationship between SO2 emissions and ground deformation.

    Args:
        df (pd.DataFrame): Prepared dataset
        xcorr_x (np.array): Cross-correlation values for x-tilt
        xcorr_y (np.array): Cross-correlation values for y-tilt
    """
    fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(15, 12))

    # Plot 1: Time series comparison
    ax1.plot(df.index, df['SO2_smoothed'], label='SO2 Flux (t/d)', color='red')
    ax1.set_ylabel('SO2 Flux (t/d)', color='red')
    ax1.tick_params(axis='y', labelcolor='red')
    ax1.legend(loc='upper left')

    # Add twin axis for tilt data
    ax1b = ax1.twinx()
    ax1b.plot(df.index, df['tilt_x_smoothed'], label='Tilt X (μrad)', color='blue')
    ax1b.plot(df.index, df['tilt_y_smoothed'], label='Tilt Y (μrad)', color='green')
    ax1b.set_ylabel('Tilt (μrad)', color='black')
    ax1b.legend(loc='upper right')
    ax1.set_title('Temporal Trend of SO2 Flux and Ground Deformation')

    # Plot 2: Cross-correlation analysis
    lags = np.arange(-len(xcorr_x)//2, len(xcorr_x)//2)
    ax2.plot(lags, xcorr_x, label='SO2 vs Tilt X')
    ax2.plot(lags, xcorr_y, label='SO2 vs Tilt Y')
    ax2.axvline(0, color='k', linestyle='--')
    ax2.set_xlim(-30, 30)
    ax2.set_xlabel('Time Lag (days)')
    ax2.set_ylabel('Normalized Cross-correlation')
    ax2.set_title('Cross-correlation Between SO2 and Ground Deformation')
    ax2.legend()
    ax2.grid(True)

    # Plot 3: Frequency domain analysis
    n = len(df)
    yf_so2 = np.fft.fft(df['SO2_smoothed'].values - df['SO2_smoothed'].mean())
    yf_tiltx = np.fft.fft(df['tilt_x_smoothed'].values - df['tilt_x_smoothed'].mean())
    xf = np.fft.fftfreq(n, d=1)  # Frequencies in days^-1

    ax3.plot(xf[:n//2], np.abs(yf_so2[:n//2]), label='SO2 Flux')
```

```
    ax3.plot(xf[:n//2], np.abs(yf_tiltx[:n//2]), label='Tilt X')
    ax3.set_xlabel('Frequency (1/days)')
    ax3.set_ylabel('Amplitude')
    ax3.set_title('Frequency Domain Analysis (FFT)')
    ax3.legend()
    ax3.grid(True)
    ax3.set_xlim(0, 0.1)  # Focus on low frequencies

    plt.tight_layout()
    plt.savefig('so2_tilt_analysis.png', dpi=300)
    plt.show()

if __name__ == "__main__":
    print("Loading and preparing data...")
    data,window_size = load_and_prepare_data()

    print("Analyzing relationship between SO2 emissions and ground deformation...")
    xcorr_x, xcorr_y = analyze_waveforms(data,window_size)

    print("Generating visualizations...")
    plot_results(data, xcorr_x, xcorr_y)

    print("Analysis complete. Plot saved as 'so2_tilt_analysis.png'")
```

```
Loading and preparing data...
Analyzing relationship between SO2 emissions and ground deformation...
Generating visualizations...
```








```
Analysis complete. Plot saved as 'so2_tilt_analysis.png'
```

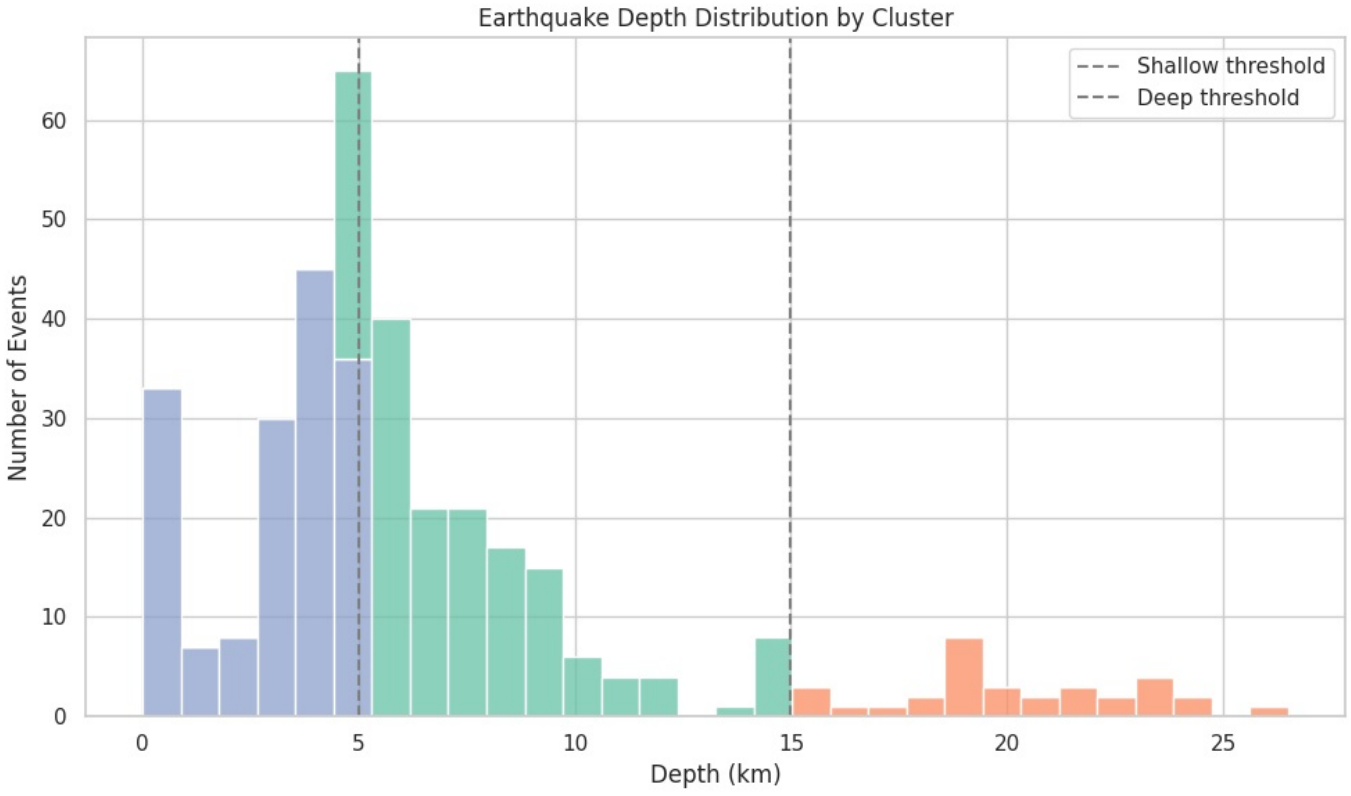
Continuazione analisi

In [39]:

```
# 1. Load the file with flexible separator detection
df = pd.read_csv('/content/drive/MyDrive/Etna2018/Etna2018_V2.xlsx - terremoti.csv', sep=None, engine='python')

# 2. Print actual column names for debugging
print("Detected columns:")
print(df.columns.tolist())

# 3. Clean column names (remove tabs, spaces)
df.columns = df.columns.str.strip().str.replace('\t', '').str.replace(' ', '')
```

```python
# 4. Check and convert depth
if 'Depth(km)' in df.columns:
    df['Depth(km)'] = df['Depth(km)'].astype(str).str.replace(',', '.').astype(float)
else:
    raise KeyError("Column 'Depth(km)' not found after cleaning. Please check the file.")

# 5. Build datetime if needed
if 'Datetime' not in df.columns:
    df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['OriginTime'], errors='coerce')

# 6. Drop invalid values
df = df[df['Depth(km)'].notna() & (df['Depth(km)'] >= 0) & df['Datetime'].notna()]

# 7. Manual clustering
def classify_depth(depth):
    if depth < 5:
        return 'Shallow (Cluster 1)'
    elif 5 <= depth < 15:
        return 'Intermediate (Cluster 2)'
    else:
        return 'Deep (Cluster 3)'

df['Depth_cluster'] = df['Depth(km)'].apply(classify_depth)

# 8. Plot histogram
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Depth(km)', hue='Depth_cluster', multiple='stack', palette='Set2', bins=30)
plt.axvline(5, color='gray', linestyle='--', label='Shallow threshold')
plt.axvline(15, color='gray', linestyle='--', label='Deep threshold')
plt.title('Earthquake Depth Distribution by Cluster')
plt.xlabel('Depth (km)')
plt.ylabel('Number of Events')
plt.legend()
plt.tight_layout()
plt.show()

# 9. Export
df.to_csv('clustered_earthquakes_depth.csv', index=False)
print("Clustering completed. File saved as 'clustered_earthquakes_depth.csv'.")

# 10. Example lookup
specific_date = pd.to_datetime('2017-07-07')
match = df[df['Datetime'].dt.date == specific_date.date()]
print("\nExample from July 7, 2017:")
print(match[['Datetime', 'Depth(km)', 'Depth_cluster']])
```

```
Detected columns:
['Date', 'Origin Time', 'Lat (°N)', 'Long (°E)', 'Depth (km)', 'ML']
```

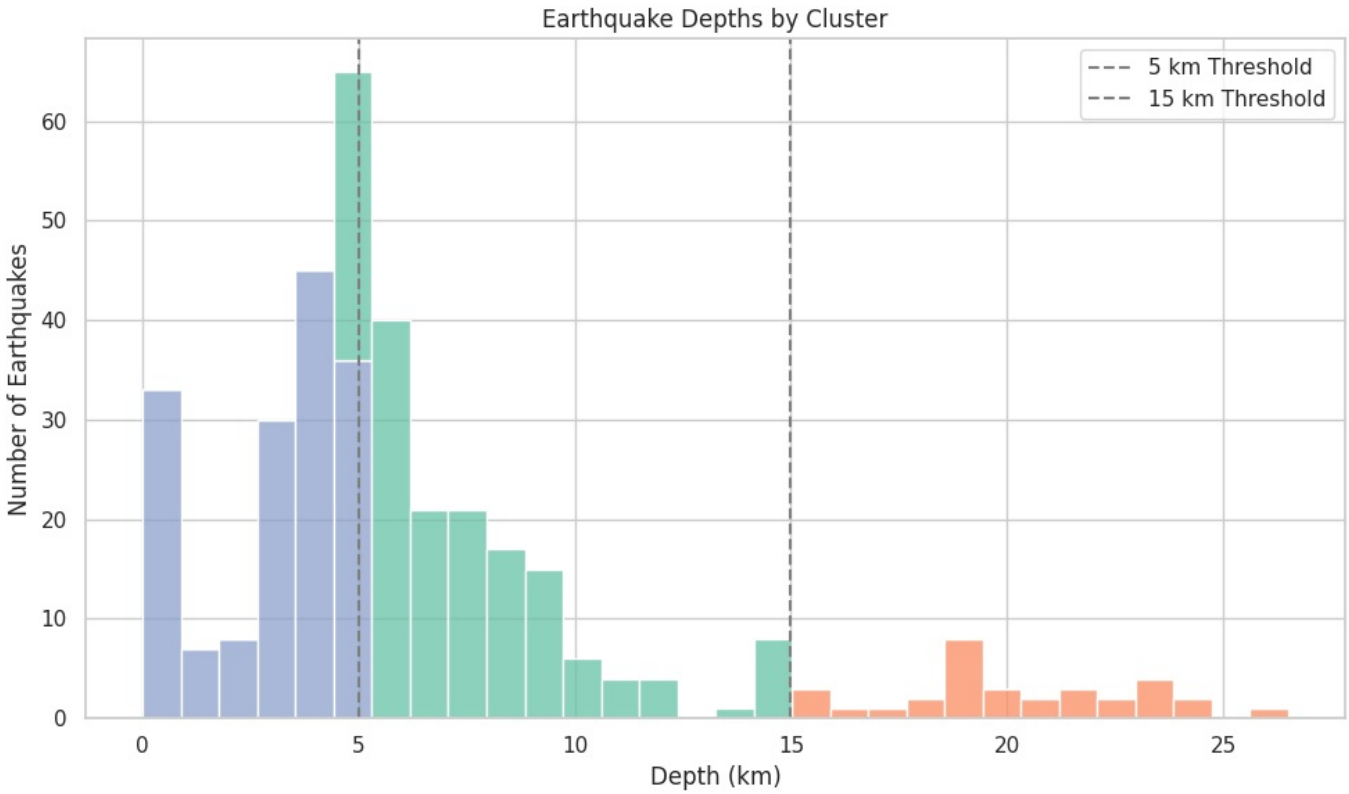


```
Clustering completed. File saved as 'clustered_earthquakes_depth.csv'.

Example from July 7, 2017:
            Datetime         Depth(km)       Depth_cluster
123 2017-07-07 02:02:43.550    3.527         Shallow (Cluster 1)
124 2017-07-07 02:14:44.020    2.973         Shallow (Cluster 1)
125 2017-07-07 08:54:31.850    2.683         Shallow (Cluster 1)
126 2017-07-07 12:52:17.280    9.253    Intermediate (Cluster 2)
127 2017-07-07 12:55:27.640    6.533    Intermediate (Cluster 2)
128 2017-07-07 13:37:39.300    8.297    Intermediate (Cluster 2)
129 2017-07-07 13:38:25.070    8.362    Intermediate (Cluster 2)
130 2017-07-07 14:07:22.990   10.140    Intermediate (Cluster 2)
131 2017-07-07 14:55:36.660    7.041    Intermediate (Cluster 2)
132 2017-07-07 14:55:45.290    8.003    Intermediate (Cluster 2)
133 2017-07-07 14:57:07.290    9.805    Intermediate (Cluster 2)
134 2017-07-07 15:02:28.520    8.758    Intermediate (Cluster 2)
135 2017-07-07 17:25:57.080    8.927    Intermediate (Cluster 2)
```

In [40]:

```python
# 1. Load the dataset, auto-detect separator
df = pd.read_csv('/content/drive/MyDrive/Etna2018/Etna2018_V2.xlsx - terremoti.csv', sep=None, engine='python')

# 2. Clean column names: remove tabs, extra spaces
df.columns = df.columns.str.strip().str.replace('\t', '').str.replace(' ', '')

# 3. Convert Depth column to float (handling commas as decimal separator)
df['Depth(km)'] = df['Depth(km)'].astype(str).str.replace(',', '.').astype(float)

# 4. Construct Datetime if not already present
if 'Datetime' not in df.columns:
    df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['OriginTime'], errors='coerce')

# 5. Filter out invalid rows
df = df[df['Depth(km)'].notna()]
df = df[df['Depth(km)'] >= 0]
df = df[df['Datetime'].notna()]

# 6. Manual clustering by depth
def classify_depth(depth):
    if depth < 5:
        return 'Cluster 1 - Shallow'
    elif 5 <= depth < 15:
        return 'Cluster 2 - Normal'
    else:
        return 'Cluster 3 - Deep'

df['DepthCluster'] = df['Depth(km)'].apply(classify_depth)

# 7. Output table: one row per earthquake with date, depth, cluster
result_df = df[['Datetime', 'Depth(km)', 'DepthCluster']].copy()

# 8. Save result
result_df.to_csv('earthquakes_depth_clusters.csv', index=False)
print(" Earthquakes clustered by depth. Saved to 'earthquakes_depth_clusters.csv'.")

# 9. Optional: histogram for visual check
plt.figure(figsize=(10, 6))
sns.histplot(data=result_df, x='Depth(km)', hue='DepthCluster', multiple='stack', palette='Set2', bins=30)
plt.title('Earthquake Depths by Cluster')
plt.xlabel('Depth (km)')
plt.ylabel('Number of Earthquakes')
plt.axvline(5, color='gray', linestyle='--', label='5 km Threshold')
plt.axvline(15, color='gray', linestyle='--', label='15 km Threshold')
plt.legend()
plt.tight_layout()
plt.show()

# 10. Check example: July 7, 2017
example_date = pd.to_datetime('2017-07-07')
match = result_df[result_df['Datetime'].dt.date == example_date.date()]
print("\nExample on July 7, 2017:")
print(match)
```

```
Earthquakes clustered by depth. Saved to 'earthquakes_depth_clusters.csv'.
```



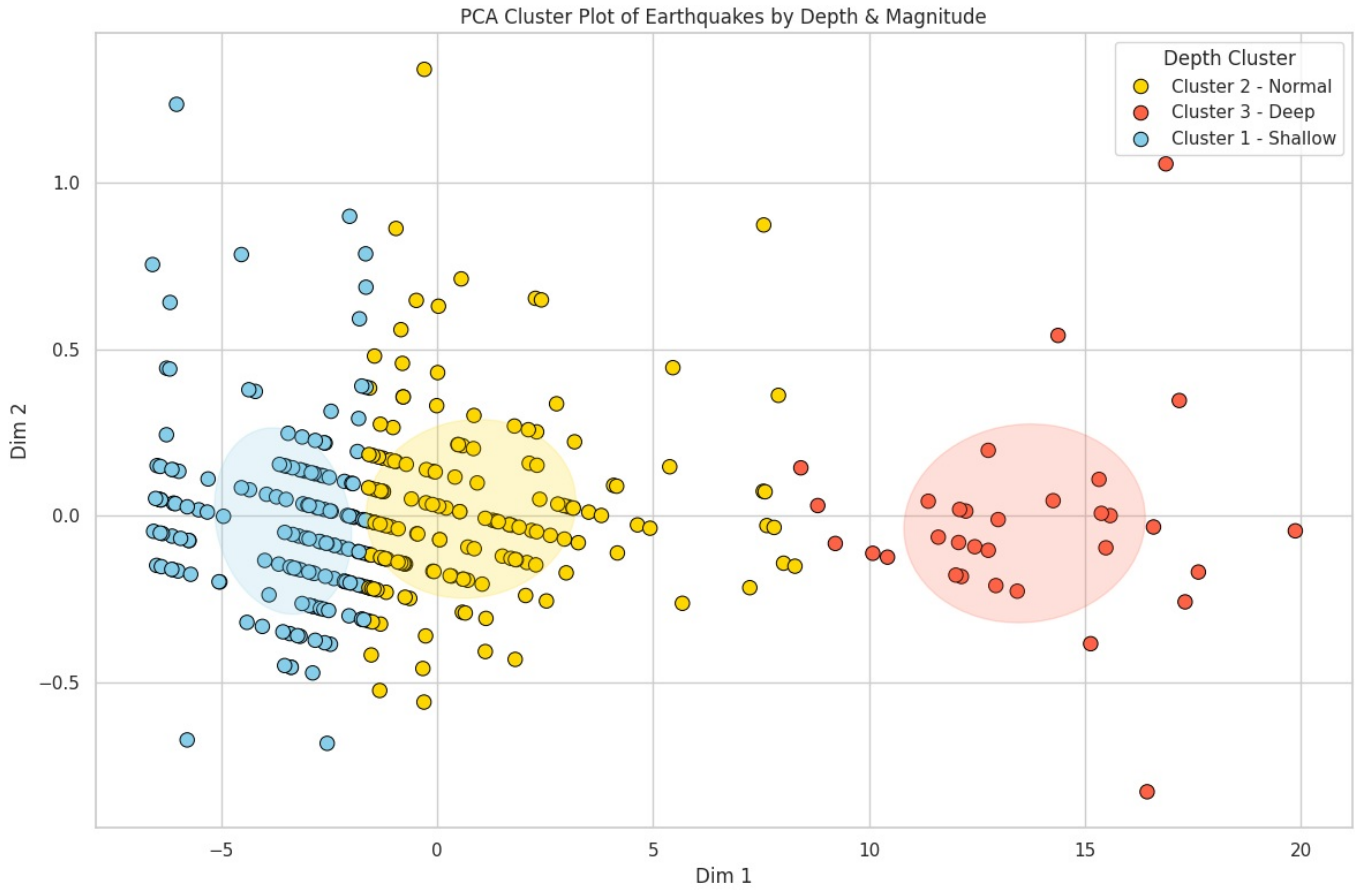
Example on July 7, 2017:

	Datetime	Depth(km)	DepthCluster
123	2017-07-07 02:02:43.550	3.527	Cluster 1 - Shallow
124	2017-07-07 02:14:44.020	2.973	Cluster 1 - Shallow
125	2017-07-07 08:54:31.850	2.683	Cluster 1 - Shallow
126	2017-07-07 12:52:17.280	9.253	Cluster 2 - Normal
127	2017-07-07 12:55:27.640	6.533	Cluster 2 - Normal
128	2017-07-07 13:37:39.300	8.297	Cluster 2 - Normal
129	2017-07-07 13:38:25.070	8.362	Cluster 2 - Normal
130	2017-07-07 14:07:22.990	10.140	Cluster 2 - Normal
131	2017-07-07 14:55:36.660	7.041	Cluster 2 - Normal
132	2017-07-07 14:55:45.290	8.003	Cluster 2 - Normal
133	2017-07-07 14:57:07.290	9.805	Cluster 2 - Normal
134	2017-07-07 15:02:28.520	8.758	Cluster 2 - Normal
135	2017-07-07 17:25:57.080	8.927	Cluster 2 - Normal

In [41]:

```
# Load and clean data
df = pd.read_csv('/content/drive/MyDrive/Etna2018/Etna2018_V2.xlsx - terremoti.csv', sep=None, engine='python')
df.columns = df.columns.str.strip().str.replace('\t', '').str.replace(' ', '')

# Convert values
df['Depth(km)'] = df['Depth(km)'].astype(str).str.replace(',', '.').astype(float)
df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['OriginTime'], errors='coerce')
df = df[df['Depth(km)'].notna() & (df['Depth(km)'] >= 0) & df['Datetime'].notna()]

# Add Magnitude check (try multiple common column names)
for mag_col in ['Magnitude', 'Mag', 'ML']:
    if mag_col in df.columns:
        df['Magnitude'] = df[mag_col].astype(str).str.replace(',', '.').astype(float)
        break
else:
    raise ValueError("No usable magnitude column found. Please check column names like 'Magnitude', 'Mag', or 'M

# Drop missing magnitude
df = df[df['Magnitude'].notna()]

# Clustering by depth
def classify_depth(depth):
    if depth < 5:
        return 'Cluster 1 - Shallow'
    elif 5 <= depth < 15:
```

```
        return 'Cluster 2 - Normal'
    else:
        return 'Cluster 3 - Deep'

df['DepthCluster'] = df['Depth(km)'].apply(classify_depth)

# PCA on Depth and Magnitude
pca = PCA(n_components=2)
df[['Dim1', 'Dim2']] = pca.fit_transform(df[['Depth(km)', 'Magnitude']])

# Plotting with ellipses
def draw_ellipse(position, covariance, ax, **kwargs):
    if covariance.shape == ():
        angle = 0
        width = height = 2 * np.sqrt(covariance)
    else:
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    ax.add_patch(Ellipse(position, width, height, angle=angle, **kwargs))

plt.figure(figsize=(12, 8))
ax = plt.gca()
palette = {'Cluster 1 - Shallow': 'skyblue', 'Cluster 2 - Normal': 'gold', 'Cluster 3 - Deep': 'tomato'}

sns.scatterplot(data=df, x='Dim1', y='Dim2', hue='DepthCluster', palette=palette, s=80, edgecolor='black')

# Draw ellipses for each cluster
for cluster in df['DepthCluster'].unique():
    subset = df[df['DepthCluster'] == cluster]
    if len(subset) > 1:
        cov = np.cov(subset[['Dim1', 'Dim2']].values.T)
        mean = subset[['Dim1', 'Dim2']].mean().values
        draw_ellipse(mean, cov, ax, alpha=0.2, color=palette[cluster])

plt.title('PCA Cluster Plot of Earthquakes by Depth & Magnitude')
plt.xlabel('Dim 1')
plt.ylabel('Dim 2')
plt.grid(True)
plt.legend(title='Depth Cluster')
plt.tight_layout()
plt.savefig('earthquake_clusters_pca.png', dpi=300)
plt.show()
```



Modello predittivo

In [42]:

```
# Suppress specific warnings if needed (adjust categories as necessary)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)


# 1. Carica il dataset con gestione tipi
df = pd.read_csv('/content/merged_etna_data.csv', index_col=0, low_memory=False)
df.index = pd.to_datetime(df.index, errors='coerce')
df = df[~df.index.isna()]  # Rimuove righe con indice non valido

# Print initial info after loading and dropping invalid index
print("Initial DataFrame info after loading and index cleanup:")
df.info()
print(f"Shape after index cleanup: {df.shape}")


# 2. Interpola valori mancanti nel tempo
df = df.interpolate(method='time')
print("\nDataFrame info after first time interpolation:")
df.info()

# 3. Seleziona solo colonne numeriche e rimuovi righe con troppi NaN
numerical_cols = df.select_dtypes(include=[np.number]).columns
df_numeric = df[numerical_cols].copy()

print(f"\nNumeric DataFrame shape: {df_numeric.shape}")

# 4. Rimuovi righe con troppi NaN (es. +50% mancanti)
# Calculate the threshold based on the number of numeric columns
threshold = int(len(numerical_cols) * 0.5)
initial_rows_before_thresh_drop = len(df_numeric)
df_numeric = df_numeric.dropna(thresh=threshold)

print(f"\nShape after dropping rows with more than {threshold} NaNs: {df_numeric.shape}")
print(f"Dropped {initial_rows_before_thresh_drop - len(df_numeric)} rows at this step.")

# 5. Interpola di nuovo se rimangono ancora NaN
# Only attempt interpolation if there are still rows left
if not df_numeric.empty:
    df_numeric = df_numeric.interpolate(method='time')
    print("\nDataFrame info after second time interpolation:")
    df_numeric.info()
else:
    print("\ndf_numeric is empty after threshold drop. Skipping second interpolation.")


# >>> DIAGNOSTIC STEP <<<
print("\n--- Diagnostic before final dropna ---")
print(f"Shape of df_numeric before final dropna: {df_numeric.shape}")
print("Number of remaining NaNs per column before final dropna:")
print(df_numeric.isnull().sum())
print("First 10 rows of df_numeric before final dropna:")
print(df_numeric.head(10))
print("--- End Diagnostic ---")
# >>> END DIAGNOSTIC STEP <<<


# 6. Rimuovi ancora righe con NaN (eventuali residui)
# Check if df_numeric is already empty before attempting dropna
initial_rows_before_final_drop = len(df_numeric)
if not df_numeric.empty:
    df_numeric = df_numeric.dropna()
    print(f"\nShape after final dropna: {df_numeric.shape}")
    print(f"Dropped {initial_rows_before_final_drop - len(df_numeric)} rows at final dropna step.")
else:
     print("\ndf_numeric was already empty before final dropna. Cannot proceed.")


# 7. Standardizza e applica KMeans
# Check if df_numeric is empty before scaling
if not df_numeric.empty:
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(df_numeric)

    kmeans = KMeans(n_clusters=3, random_state=42, n_init=10) # Added n_init to KMeans
    cluster_labels = kmeans.fit_predict(X_scaled)

    # 8. Aggiungi le etichette
    df_clustered = df_numeric.copy()
    df_clustered['vulcano_state'] = cluster_labels

    # 9. Ordina gli stati in base alla media RMS (se RMS è presente)
    if 'RMS' in df_clustered.columns:
```

```
        cluster_order = df_clustered.groupby('vulcano_state')['RMS'].mean().sort_values().index.tolist()
        state_mapping = {old: new for new, old in enumerate(cluster_order)}
        df_clustered['vulcano_state'] = df_clustered['vulcano_state'].map(state_mapping)

    # 10. Prepara i dati per la classificazione supervisata
    def create_supervised_sequences(df, window=3):
        feature_cols = df.columns.drop('vulcano_state')
        X, y = [], []

        # Iterate up to the second to last row, as we predict the state of i+1
        for i in range(window, len(df) - 1):
            # Ensure the window is within bounds and has no NaNs (should be covered by previous steps, but good
            window_slice = df.iloc[i - window : i]
            if not window_slice.isnull().values.any():
                past_window = window_slice[feature_cols].values.flatten()
                label = df.iloc[i + 1]['vulcano_state']
                 # Ensure the target label is not NaN (shouldn't happen if df_clustered.dropna() worked)
                if not np.isnan(label):
                    X.append(past_window)
                    y.append(int(label)) # Convert to int for classification

        return np.array(X), np.array(y)


    print("\nCreating supervised sequences...")
    # Ensure window size is less than the number of samples
    if len(df_clustered) > 4: # Minimum 4 rows needed for window=3 (3 history + 1 target)
        X, y = create_supervised_sequences(df_clustered, window=3)

        if len(X) > 0:
            print(f"Generated {len(X)} sequences.")

            # 11. Split temporale train/test
            split_index = int(len(X) * 0.8)
            X_train, X_test = X[:split_index], X[split_index:]
            y_train, y_test = y[:split_index], y[split_index:]

            print(f"Train set size: {len(X_train)}")
            print(f"Test set size: {len(X_test)}")

            # Check if train/test sets are not empty
            if len(X_train) > 0 and len(X_test) > 0:
                # 12. Modello Random Forest
                print("\nTraining Random Forest classifier...")
                clf = RandomForestClassifier(n_estimators=100, random_state=42)
                clf.fit(X_train, y_train)

                # 13. Valutazione
                print("\nClassification Report:")
                # Adjust target names if needed based on your cluster interpretation
                print(classification_report(y_test, y_pred, target_names=['Stato 0', 'Stato 1', 'Stato 2']))
            else:
                print("\nNot enough data to create valid train/test sets for classification.")
        else:
            print("\nNo valid sequences generated for classification.")
    else:
        print(f"\nNot enough data points ({len(df_clustered)}) for the specified window size (3) for supervised

else:
    print("\ndf_numeric is empty. Cannot perform scaling, clustering, or classification.")
```

```
Initial DataFrame info after loading and index cleanup:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 451484 entries, 2008-05-21 08:30:00 to 2022-12-31 23:45:00
Data columns (total 27 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Radiant Heat Flux [W]  285 non-null     float64
 1   SO2_flux_t_d           385 non-null     float64
 2   daily HCl flux (t/d)   26 non-null      float64
 3   outlier                26 non-null      object
 4   Stadio                 23 non-null      float64
 5   Fondachello            22 non-null      float64
 6   P39                    23 non-null      float64
 7   Vallone Salato         22 non-null      float64
 8   Naftia                 23 non-null      float64
 9   CO2/SO2                287 non-null     float64
 10  Is_Outlier             287 non-null     object
 11  Year                   287 non-null     float64
 12  Month                  287 non-null     float64
 13  Day                    287 non-null     float64
 14  Date_only              287 non-null     object
 15  ten_batt               450907 non-null  float64
 16  temp_CR10              450907 non-null  float64
```

```
 17  tilt_x_Avg             450907 non-null  float64
 18  tilt_y_Avg             450907 non-null  float64
 19  temp_tilt              450907 non-null  float64
 20  nord_tilt              371159 non-null  float64
 21  barometro              450907 non-null  float64
 22  RMS                    377 non-null     float64
 23  Lat (°N)               0 non-null       float64
 24  Long (°E)              0 non-null       float64
 25  Depth (km)             0 non-null       float64
 26  ML                     0 non-null       float64
dtypes: float64(24), object(3)
memory usage: 96.4+ MB
Shape after index cleanup: (451484, 27)

DataFrame info after first time interpolation:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 451484 entries, 2008-05-21 08:30:00 to 2022-12-31 23:45:00
Data columns (total 27 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Radiant Heat Flux [W]  136407 non-null  float64
 1   SO2_flux_t_d           174265 non-null  float64
 2   daily HCl flux (t/d)   172539 non-null  float64
 3   outlier                26 non-null      object
 4   Stadio                 173115 non-null  float64
 5   Fondachello            173115 non-null  float64
 6   P39                    173115 non-null  float64
 7   Vallone Salato         173115 non-null  float64
 8   Naftia                 173115 non-null  float64
 9   CO2/SO2                162303 non-null  float64
 10  Is_Outlier             287 non-null     object
 11  Year                   162303 non-null  float64
 12  Month                  162303 non-null  float64
 13  Day                    162303 non-null  float64
 14  Date_only              287 non-null     object
 15  ten_batt               451484 non-null  float64
 16  temp_CR10              451484 non-null  float64
 17  tilt_x_Avg             451484 non-null  float64
 18  tilt_y_Avg             451484 non-null  float64
 19  temp_tilt              451484 non-null  float64
 20  nord_tilt              371736 non-null  float64
 21  barometro              451484 non-null  float64
 22  RMS                    174265 non-null  float64
 23  Lat (°N)               0 non-null       float64
 24  Long (°E)              0 non-null       float64
 25  Depth (km)             0 non-null       float64
 26  ML                     0 non-null       float64
dtypes: float64(24), object(3)
memory usage: 96.4+ MB

Numeric DataFrame shape: (451484, 24)

Shape after dropping rows with more than 12 NaNs: (173115, 24)
Dropped 278369 rows at this step.

DataFrame info after second time interpolation:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 173115 entries, 2017-01-13 00:00:00 to 2022-12-31 23:45:00
Data columns (total 24 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Radiant Heat Flux [W]  136407 non-null  float64
 1   SO2_flux_t_d           173115 non-null  float64
 2   daily HCl flux (t/d)   172539 non-null  float64
 3   Stadio                 173115 non-null  float64
 4   Fondachello            173115 non-null  float64
 5   P39                    173115 non-null  float64
 6   Vallone Salato         173115 non-null  float64
 7   Naftia                 173115 non-null  float64
 8   CO2/SO2                162303 non-null  float64
 9   Year                   162303 non-null  float64
 10  Month                  162303 non-null  float64
 11  Day                    162303 non-null  float64
 12  ten_batt               173115 non-null  float64
 13  temp_CR10              173115 non-null  float64
 14  tilt_x_Avg             173115 non-null  float64
 15  tilt_y_Avg             173115 non-null  float64
 16  temp_tilt              173115 non-null  float64
 17  nord_tilt              173115 non-null  float64
 18  barometro              173115 non-null  float64
 19  RMS                    173115 non-null  float64
 20  Lat (°N)               0 non-null       float64
 21  Long (°E)              0 non-null       float64
```

```
 22  Depth (km)             0 non-null       float64
 23  ML                     0 non-null       float64
dtypes: float64(24)
memory usage: 33.0 MB

--- Diagnostic before final dropna ---
Shape of df_numeric before final dropna: (173115, 24)
Number of remaining NaNs per column before final dropna:
Radiant Heat Flux [W]     36708
SO2_flux_t_d                  0
daily HCl flux (t/d)        576
Stadio                        0
Fondachello                   0
P39                           0
Vallone Salato                0
Naftia                        0
CO2/SO2                   10812
Year                      10812
Month                     10812
Day                       10812
ten_batt                      0
temp_CR10                     0
tilt_x_Avg                    0
tilt_y_Avg                    0
temp_tilt                     0
nord_tilt                     0
barometro                     0
RMS                           0
Lat (°N)                 173115
Long (°E)                173115
Depth (km)               173115
ML                       173115
dtype: int64
First 10 rows of df_numeric before final dropna:
                     Radiant Heat Flux [W]  SO2_flux_t_d  daily HCl flux (t/d)   Stadio   Fondachello     P39
Vallone Salato   Naftia   CO2/SO2  Year  Month  Day  ten_batt  temp_CR10  tilt_x_Avg  tilt_y_Avg  temp_tilt  nor
d_tilt  barometro     RMS    Lat (°N)  Long (°E)  Depth (km)  ML
2017-01-13 00:00:00          NaN             3388.360000          NaN           6.320000   6.440000    7.100000
6.580000     6.580000   NaN     NaN   NaN   NaN     12.17     3.165      85.498      216.386     6.54       230.
0      848.0    0.000002    NaN       NaN         NaN     NaN
2017-01-13 00:15:00          NaN             3388.383854          NaN           6.319996   6.439991    7.099996
6.579996     6.579996   NaN     NaN   NaN   NaN     12.17     3.142      85.523      216.373     6.54       230.
1      845.0    0.000002    NaN       NaN         NaN     NaN
2017-01-13 00:30:00          NaN             3388.407708          NaN           6.319991   6.439983    7.099991
6.579991     6.579991   NaN     NaN   NaN   NaN     12.16     3.142      85.535      216.399     6.54       230.
1      845.0    0.000002    NaN       NaN         NaN     NaN
2017-01-13 00:45:00          NaN             3388.431562          NaN           6.319987   6.439974    7.099987
6.579987     6.579987   NaN     NaN   NaN   NaN     12.15     3.122      85.516      216.380     6.52       230.
1      845.0    0.000002    NaN       NaN         NaN     NaN
2017-01-13 01:00:00          NaN             3388.455417          NaN           6.319983   6.439965    7.099983
6.579983     6.579983   NaN     NaN   NaN   NaN     12.15     3.122      85.529      216.393     6.52       230.
0      848.0    0.000002    NaN       NaN         NaN     NaN
2017-01-13 01:15:00          NaN             3388.479271          NaN           6.319978   6.439957    7.099978
6.579978     6.579978   NaN     NaN   NaN   NaN     12.14     3.122      85.535      216.386     6.54       230.
1      845.0    0.000002    NaN       NaN         NaN     NaN
2017-01-13 01:30:00          NaN             3388.503125          NaN           6.319974   6.439948    7.099974
6.579974     6.579974   NaN     NaN   NaN   NaN     12.14     3.099      85.541      216.354     6.54       230.
1      845.0    0.000002    NaN       NaN         NaN     NaN
2017-01-13 01:45:00          NaN             3388.526979          NaN           6.319970   6.439939    7.099970
6.579970     6.579970   NaN     NaN   NaN   NaN     12.13     3.099      85.553      216.361     6.52       230.
1      845.0    0.000002    NaN       NaN         NaN     NaN
2017-01-13 02:00:00          NaN             3388.550833          NaN           6.319965   6.439931    7.099965
6.579965     6.579965   NaN     NaN   NaN   NaN     12.13     3.099      85.535      216.348     6.52       230.
0      848.0    0.000002    NaN       NaN         NaN     NaN
2017-01-13 02:15:00          NaN             3388.574687          NaN           6.319961   6.439922    7.099961
6.579961     6.579961   NaN     NaN   NaN   NaN     12.13     3.099      85.535      216.342     6.53       230.
0      845.0    0.000002    NaN       NaN         NaN     NaN
--- End Diagnostic ---

Shape after final dropna: (0, 24)
Dropped 173115 rows at final dropna step.

df_numeric is empty. Cannot perform scaling, clustering, or classification.
```

In [43]:

```python
def load_and_preprocess_data(filepath):
    """Carica e preelabora i dati"""
    try:
        # 1. Carica il dataset
        df = pd.read_csv(filepath, index_col=0, low_memory=False)
        print(f"Dati caricati. Dimensioni iniziali: {df.shape}")

        # 2. Converti indice a datetime
        df.index = pd.to_datetime(df.index, errors='coerce')
```

```
        df = df[~df.index.isna()]
        print(f"Dopo pulizia indice: {df.shape}")

        if df.empty:
            raise ValueError("DataFrame vuoto dopo pulizia indice")

        return df
    except Exception as e:
        print(f"Errore nel caricamento dati: {str(e)}")
        raise

def prepare_numeric_data(df):
    """Prepara i dati numerici"""
    # 3. Seleziona solo colonne numeriche
    numerical_cols = df.select_dtypes(include=[np.number]).columns
    print(f"Trovate {len(numerical_cols)} colonne numeriche")

    if len(numerical_cols) == 0:
        raise ValueError("Nessuna colonna numerica trovata")

    df_numeric = df[numerical_cols].copy()

    # 4. Interpola e pulisci i valori mancanti
    df_numeric = df_numeric.interpolate(method='time')

    # 5. Soglia più conservativa per valori mancanti (max 80% mancanti)
    threshold = int(len(numerical_cols) * 0.2)
    df_numeric = df_numeric.dropna(thresh=threshold)

    if df_numeric.empty:
        raise ValueError("Nessun dato valido dopo pulizia valori mancanti")

    # 6. Riempimento finale valori mancanti con la mediana
    df_numeric = df_numeric.fillna(df_numeric.median())

    print(f"Dati numerici finali: {df_numeric.shape}")
    return df_numeric

def apply_clustering(df_numeric):
    """Applica il clustering K-means"""
    # 7. Standardizzazione
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(df_numeric)

    # 8. Clustering
    kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
    cluster_labels = kmeans.fit_predict(X_scaled)

    # 9. Aggiungi etichette al DataFrame
    df_clustered = df_numeric.copy()
    df_clustered['vulcano_state'] = cluster_labels

    # 10. Ordina stati se RMS è presente
    if 'RMS' in df_clustered.columns:
        cluster_order = df_clustered.groupby('vulcano_state')['RMS'].mean().sort_values().index.tolist()
        state_mapping = {old: new for new, old in enumerate(cluster_order)}
        df_clustered['vulcano_state'] = df_clustered['vulcano_state'].map(state_mapping)

    return df_clustered

def create_sequences(df, window=3):
    """Crea sequenze temporali per la classificazione"""
    feature_cols = df.columns.drop('vulcano_state')
    X, y = [], []

    for i in range(window, len(df) - 1):
        past_window = df.iloc[i - window:i][feature_cols].values.flatten()
        label = df.iloc[i + 1]['vulcano_state']
        if not np.isnan(past_window).any():
            X.append(past_window)
            y.append(label)

    return np.array(X), np.array(y)

def main():
    try:
        # Percorso del file - modifica se necessario
        file_path = '/content/numeric_timeseries_filtered.csv'

        # 1. Caricamento e preprocessamento
        df = load_and_preprocess_data(file_path)

        # 2. Preparazione dati numerici
```

```
        df_numeric = prepare_numeric_data(df)

        # 3. Clustering
        df_clustered = apply_clustering(df_numeric)
        print("Cluster creati con successo")
        print(df_clustered['vulcano_state'].value_counts())

        # 4. Preparazione dati supervisionati
        X, y = create_sequences(df_clustered, window=3)
        print(f"Dimensioni dataset finale: X={X.shape}, y={y.shape}")

        if len(X) == 0:
            raise ValueError("Nessuna sequenza valida creata")

        # 5. Split temporale
        split_index = int(len(X) * 0.8)
        X_train, X_test = X[:split_index], X[split_index:]
        y_train, y_test = y[:split_index], y[split_index:]

        # 6. Addestramento modello
        clf = RandomForestClassifier(n_estimators=100, random_state=42)
        clf.fit(X_train, y_train)

        # 7. Valutazione
        y_pred = clf.predict(X_test)
        print(classification_report(y_test, y_pred,
              target_names=['Stato 0 - quiete', 'Stato 1 - agitato', 'Stato 2 - instabile']))

    except Exception as e:
        print(f"Errore durante l'esecuzione: {str(e)}")
        # Stampa informazioni utili per il debug
        if 'df' in locals():
            print("\nInformazioni sul DataFrame:")
            print(df.info())
            print("\nPrime righe:")
            print(df.head())

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

```
Dati caricati. Dimensioni iniziali: (70300, 7)
Dopo pulizia indice: (70300, 7)
Trovate 7 colonne numeriche
Dati numerici finali: (70300, 7)
Cluster creati con successo
vulcano_state
0    30705
1    30219
2     9376
Name: count, dtype: int64
Dimensioni dataset finale: X=(70296, 21), y=(70296,)
                    precision    recall  f1-score   support

   Stato 0 - quiete       0.78      0.99      0.88      7633
  Stato 1 - agitato       0.98      0.68      0.80      6426
Stato 2 - instabile       0.00      0.00      0.00         1

          accuracy                           0.85     14060
         macro avg       0.59      0.56      0.56     14060
      weighted avg       0.87      0.85      0.84     14060
```

In [44]:

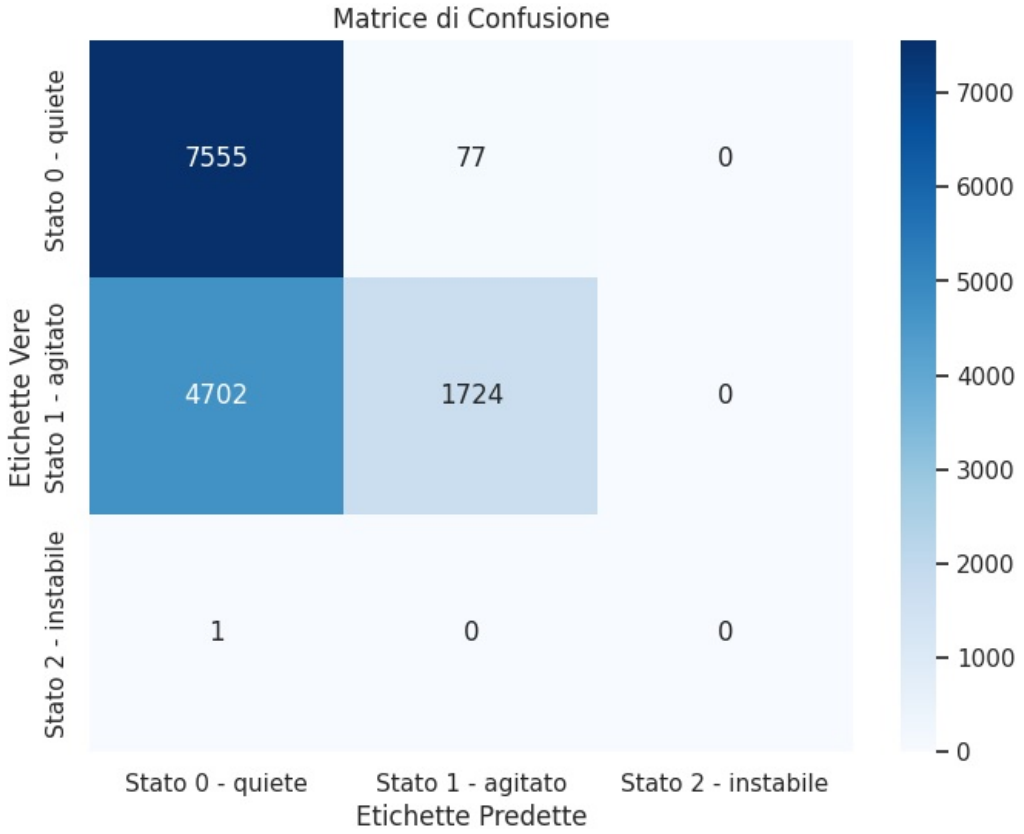
```
def evaluate_model(y_true, y_pred, class_names):
    """Calcola e visualizza le metriche di valutazione"""
    # Metriche principali
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')

    print("\n" + "="*50)
    print("METRICHE PRINCIPALI DEL MODELLO")
    print("="*50)
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision (media pesata): {precision:.4f}")
    print(f"Recall (media pesata): {recall:.4f}")
    print(f"F1-Score (media pesata): {f1:.4f}")

    # Report completo
    print("\n" + "="*50)
    print("CLASSIFICATION REPORT DETTAGLIATO")
    print("="*50)
    print(classification_report(y_true, y_pred, target_names=class_names))
```

```
    # Matrice di confusione
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names,
                yticklabels=class_names)
    plt.title('Matrice di Confusione')
    plt.ylabel('Etichette Vere')
    plt.xlabel('Etichette Predette')
    plt.show()

def train_and_evaluate(X_train, X_test, y_train, y_test, class_names):
    """Addestra e valuta il modello RandomForest"""
    print("\nAddestramento del modello RandomForest...")
    # Aggiunto class_weight='balanced' e aumentato n_estimators
    clf = RandomForestClassifier(n_estimators=200,
                                 random_state=42,
                                 class_weight='balanced',
                                 max_depth=10)
    clf.fit(X_train, y_train)

    # Predizioni sul test set
    y_pred = clf.predict(X_test)

    # Valutazione
    evaluate_model(y_test, y_pred, class_names)

    # Feature importance (se disponibili)
    if hasattr(clf, 'feature_importances_'):
        print("\nTop 10 feature più importanti:")
        feature_importance = pd.Series(clf.feature_importances_,
                                       index=[f"feature_{i}" for i in range(X_train.shape[1])])
        print(feature_importance.sort_values(ascending=False).head(10))

    return clf

def analyze_folds(fold_details):
    """Analisi dettagliata dei risultati dei fold"""
    df_folds = pd.DataFrame(fold_details)

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    df_folds.plot(x='fold', y='accuracy', kind='bar', title='Accuracy per Fold')
    plt.axhline(y=df_folds['accuracy'].mean(), color='r', linestyle='--')

    plt.subplot(1, 2, 2)
    pd.DataFrame(df_folds['class_distribution'].tolist()).plot(kind='bar', stacked=True)
    plt.title('Distribuzione Classi per Fold')

    plt.tight_layout()
    plt.show()

    return df_folds

def main():
    try:
        # Configurazione
        FILE_PATH = '/content/numeric_timeseries_filtered.csv'
        CLASS_NAMES = ['Stato 0 - quiete', 'Stato 1 - agitato', 'Stato 2 - instabile']
        WINDOW_SIZE = 5  # Aumentata la finestra temporale

        # 1. Caricamento dati
        print("Caricamento dati...")
        df = pd.read_csv(FILE_PATH, index_col=0, low_memory=False)
        df.index = pd.to_datetime(df.index, errors='coerce')
        df = df[~df.index.isna()]

        # 2. Preparazione dati numerici
        print("\nPreparazione dati numerici...")
        numerical_cols = df.select_dtypes(include=[np.number]).columns
        df_numeric = df[numerical_cols].interpolate(method='time')
        df_numeric = df_numeric.fillna(df_numeric.median())

        # 3. Clustering - Aggiunto controllo per classi vuote
        print("\nApplicazione clustering...")
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(df_numeric)
        kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
        df_numeric['vulcano_state'] = kmeans.fit_predict(X_scaled)

        # Verifica distribuzione classi
        print("\nDistribuzione classi originale:")
        print(df_numeric['vulcano_state'].value_counts())
```

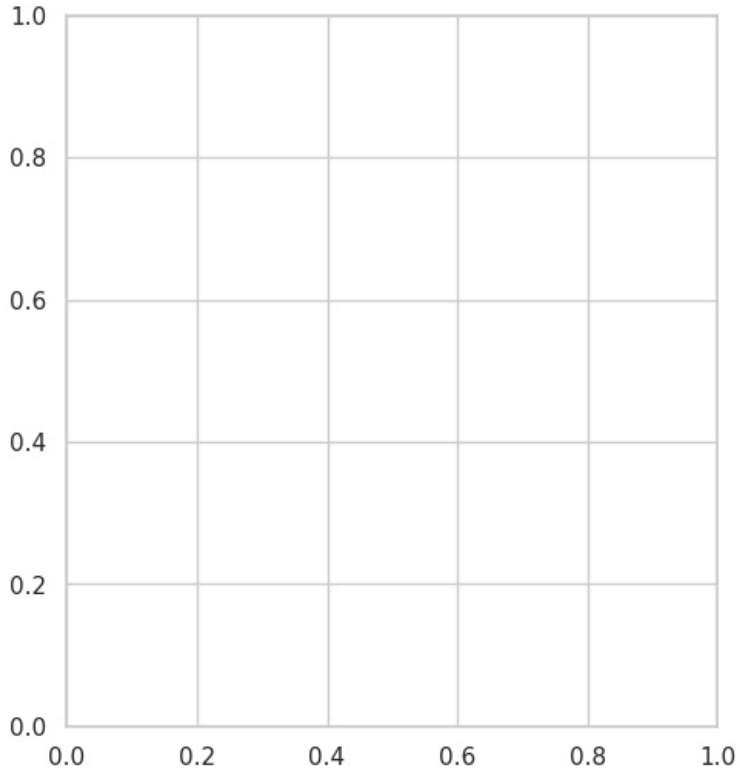
```python
# 4. Creazione sequenze temporali
print("\nCreazione sequenze temporali...")
X, y = [], []
feature_cols = df_numeric.columns.drop('vulcano_state')

for i in range(WINDOW_SIZE, len(df_numeric) - 1):
    past_window = df_numeric.iloc[i-WINDOW_SIZE:i][feature_cols].values.flatten()
    label = df_numeric.iloc[i+1]['vulcano_state']
    if not np.isnan(past_window).any():
        X.append(past_window)
        y.append(label)

X, y = np.array(X), np.array(y)

# Filtra le classi con pochi campioni (es. classe 2)
class_counts = pd.Series(y).value_counts()
min_samples = 100  # Soglia minima
valid_classes = class_counts[class_counts >= min_samples].index
mask = np.isin(y, valid_classes)
X, y = X[mask], y[mask]

print("\nDistribuzione classi dopo filtraggio:")
print(pd.Series(y).value_counts())

# 5. Split temporale (80% train, 20% test)
split_idx = int(len(X) * 0.8)
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

# 6. Addestramento e valutazione
print(f"\nDati di addestramento: {X_train.shape[0]} samples")
print(f"Dati di test: {X_test.shape[0]} samples")

model = train_and_evaluate(X_train, X_test, y_train, y_test, CLASS_NAMES)

# 7. Validazione aggiuntiva con TimeSeriesSplit
print("\nValidazione con TimeSeriesSplit (5 fold)...")
tscv = TimeSeriesSplit(n_splits=5)
fold_accuracies = []
fold_details = []  # Memorizza i dettagli di ogni fold

for fold, (train_idx, val_idx) in enumerate(tscv.split(X)):
    X_train_fold, X_val_fold = X[train_idx], X[val_idx]
    y_train_fold, y_val_fold = y[train_idx], y[val_idx]

    # Usa gli stessi parametri del modello principale
    clf = RandomForestClassifier(n_estimators=200,
                                 random_state=42,
                                 class_weight='balanced',
                                 max_depth=10)
    clf.fit(X_train_fold, y_train_fold)
    y_pred_fold = clf.predict(X_val_fold)

    acc = accuracy_score(y_val_fold, y_pred_fold)
    fold_accuracies.append(acc)

    # Memorizza i dettagli del fold
    fold_dates = df_numeric.index[val_idx + WINDOW_SIZE + 1]  # Aggiustato l'indice
    fold_details.append({
        'fold': fold+1,
        'accuracy': acc,
        'start_date': fold_dates.min(),
        'end_date': fold_dates.max(),
        'class_distribution': pd.Series(y_val_fold).value_counts().to_dict()
    })
    print(f"Fold {fold+1}: Accuracy = {acc:.4f} | Periodo: {fold_dates.min()} to {fold_dates.max()}")

print(f"\nAccuracy media su 5 fold: {np.mean(fold_accuracies):.4f}")
print(f"Deviazione standard: {np.std(fold_accuracies):.4f}")

# Analisi dettagliata dei fold
fold_analysis_df = analyze_folds(fold_details)

# Analisi del fold problematico
worst_fold = min(fold_details, key=lambda x: x['accuracy'])
print("\n" + "="*50)
print(f"FOLD PROBLEMATICO (Accuracy: {worst_fold['accuracy']:.4f})")
print(f"Periodo: {worst_fold['start_date']} to {worst_fold['end_date']}")
print("Distribuzione classi:", worst_fold['class_distribution'])

# Analisi delle feature importanti nel fold problematico
if worst_fold['accuracy'] < 0.5:  # Se l'accuracy è molto bassa
```

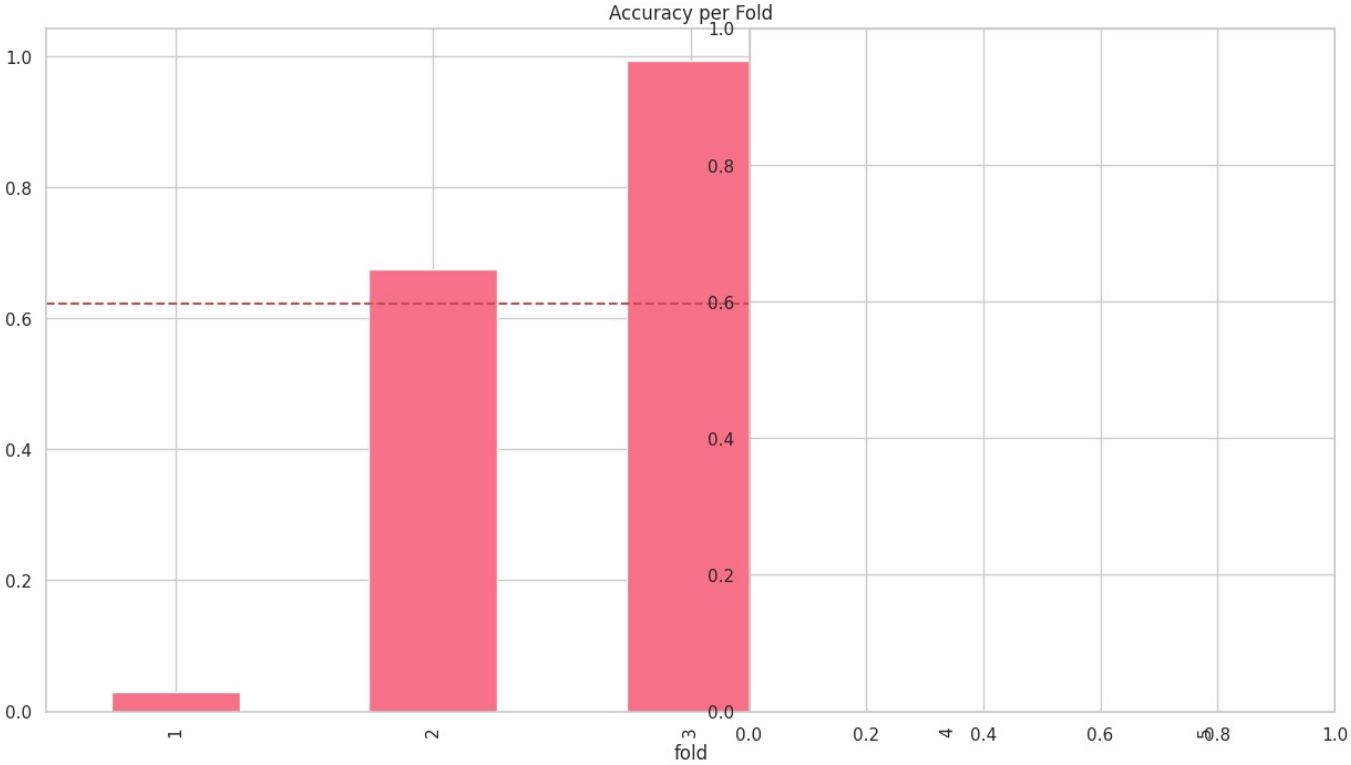
```
            print("\nAnalisi feature importanti nel fold problematico:")
            problem_mask = (df_numeric.index >= worst_fold['start_date']) & \
                           (df_numeric.index <= worst_fold['end_date'])
            problem_data = df_numeric[problem_mask]

            plt.figure(figsize=(12, 6))
            problem_data[feature_cols].mean().plot(kind='bar')
            plt.title('Media delle feature nel periodo problematico')
            plt.xticks(rotation=90)
            plt.show()

    except Exception as e:
        print(f"\nErrore durante l'esecuzione: {str(e)}")

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

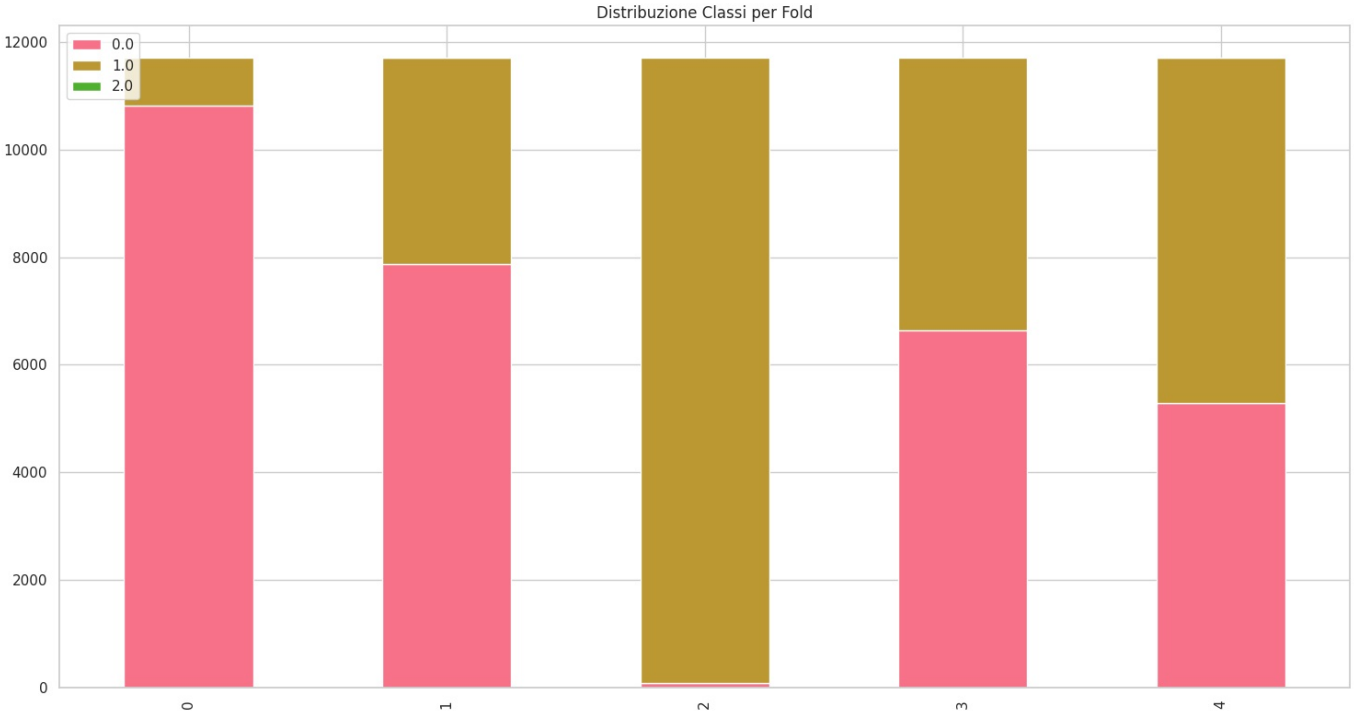
```
Caricamento dati...

Preparazione dati numerici...

Applicazione clustering...

Distribuzione classi originale:
vulcano_state
0    30705
1    30219
2     9376
Name: count, dtype: int64

Creazione sequenze temporali...

Distribuzione classi dopo filtraggio:
0.0    30705
1.0    30213
2.0     9376
Name: count, dtype: int64

Dati di addestramento: 56235 samples
Dati di test: 14059 samples

Addestramento del modello RandomForest...

==================================================
METRICHE PRINCIPALI DEL MODELLO
==================================================
Accuracy: 0.6600
Precision (media pesata): 0.7721
Recall (media pesata): 0.6600
F1-Score (media pesata): 0.6040

==================================================
CLASSIFICATION REPORT DETTAGLIATO
==================================================
                    precision    recall  f1-score   support

   Stato 0 - quiete       0.62      0.99      0.76      7632
  Stato 1 - agitato       0.96      0.27      0.42      6426
Stato 2 - instabile       0.00      0.00      0.00         1

          accuracy                           0.66     14059
         macro avg       0.52      0.42      0.39     14059
      weighted avg       0.77      0.66      0.60     14059
```



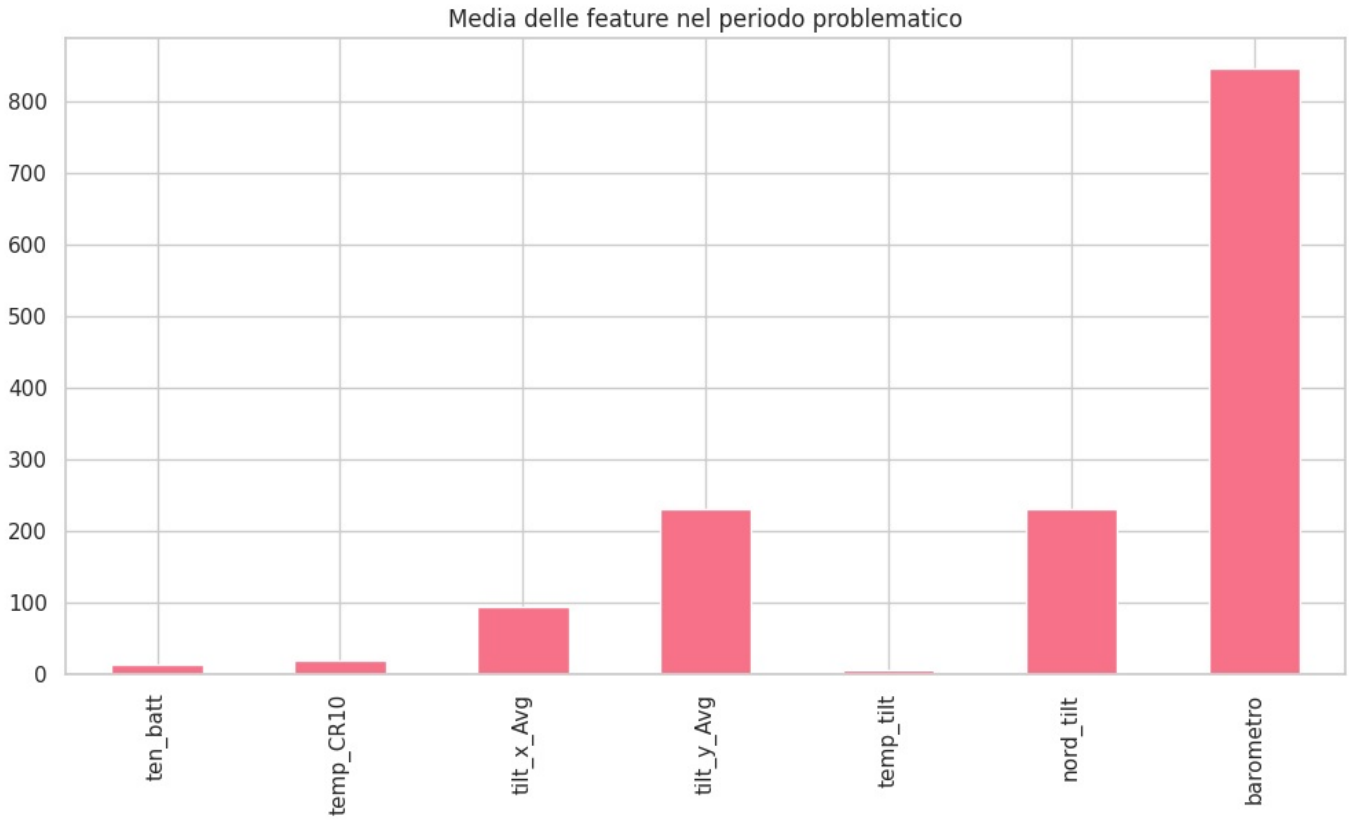

```
Top 10 feature più importanti:
feature_31    0.094997
feature_24    0.091995
feature_17    0.082897
feature_25    0.071603
feature_4     0.067641
feature_18    0.062370
feature_10    0.054039
feature_11    0.051309
feature_32    0.051010
feature_3     0.049575
dtype: float64

Validazione con TimeSeriesSplit (5 fold)...
Fold 1: Accuracy = 0.0293 | Periodo: 2018-03-28 02:30:00 to 2018-07-26 14:00:00
Fold 2: Accuracy = 0.6738 | Periodo: 2018-07-26 14:15:00 to 2018-11-26 04:30:00
Fold 3: Accuracy = 0.9931 | Periodo: 2018-11-26 04:45:00 to 2019-03-25 00:15:00
Fold 4: Accuracy = 0.9055 | Periodo: 2019-03-25 00:30:00 to 2019-07-25 01:00:00
Fold 5: Accuracy = 0.5119 | Periodo: 2019-07-25 01:15:00 to 2019-12-31 23:45:00

Accuracy media su 5 fold: 0.6227
Deviazione standard: 0.3418
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Accuracy per Fold
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
fold
0.0
0.2
0.4
0.6
0.8
1.0




```
==================================================
FOLD PROBLEMATICO (Accuracy: 0.0293)
Periodo: 2018-03-28 02:30:00 to 2018-07-26 14:00:00
Distribuzione classi: {0.0: 10822, 1.0: 893}

Analisi feature importanti nel fold problematico:
```



In [45]:
```
#top features
# Per vedere quali colonne corrispondono alle feature importanti
```

```
print(df_numeric.columns[4], df_numeric.columns[18], df_numeric.columns[11])
```

```
Fondachello barometro Day
```

Standardizzazione feautures

In [46]:
```
# Carica il dataset
df = pd.read_csv('numeric_timeseries_filtered.csv', parse_dates=True)

# Rimuove eventuali colonne non numeriche (es. 'date')
df = df.select_dtypes(include='number')

# Rimuove le righe con valori mancanti
df_clean = df.dropna()

# Applica la standardizzazione (Z-score: media 0, std 1)
scaler = StandardScaler()
scaled_array = scaler.fit_transform(df_clean)

# Ricrea il DataFrame con le stesse colonne
df_scaled = pd.DataFrame(scaled_array, columns=df_clean.columns)

# Salva il file trasformato
df_scaled.to_csv('features_standardized.csv', index=False)

print("Standardizzazione completata. File salvato come 'features_standardized.csv'")
```

```
Standardizzazione completata. File salvato come 'features_standardized.csv'
```

In [47]:
```
df = pd.read_csv('numeric_timeseries_filtered.csv', parse_dates=True)
df = df.select_dtypes(include='number')
df_clean = df.dropna()

scaler = MinMaxScaler()
scaled_array = scaler.fit_transform(df_clean)

df_scaled = pd.DataFrame(scaled_array, columns=df_clean.columns)
df_scaled.to_csv('features_normalized.csv', index=False)

print("Normalizzazione completata. File salvato come 'features_normalized.csv'")
```

```
Normalizzazione completata. File salvato come 'features_normalized.csv'
```

Genriamo sliding window

In [48]:
```
# PARAMETRI
input_file = 'numeric_timeseries_filtered.csv'
window_size = 30         # numero di timesteps (es. 30 giorni)
horizon = 1              # previsione a 1 passo nel futuro (target)
stride = 1               # passo dello sliding window
target_col = 'barometro'  # scegli una colonna da predire (puoi cambiarla)

df = pd.read_csv(input_file)

# Rimuove colonne non numeriche (es. 'date') e righe con NaN
df = df.select_dtypes(include='number').dropna().reset_index(drop=True)


features = df.columns.tolist()
if target_col not in features:
    raise ValueError(f"Colonna target '{target_col}' non trovata nel dataset")

# 2. Normalizzazione MinMax (puoi sostituire con StandardScaler)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)

# 3. Crea sliding window (X, y)
X = []
y = []

for i in range(0, len(scaled_data) - window_size - horizon + 1, stride):
    window = scaled_data[i : i + window_size]
    target = scaled_data[i + window_size + horizon - 1, df.columns.get_loc(target_col)]

    X.append(window)
    y.append(target)

X = np.array(X)  # shape: (num_samples, window_size, num_features)
y = np.array(y)  # shape: (num_samples,)

print(f"Sliding window completato!")
```

```
print(f"X shape: {X.shape}")
print(f"y shape: {y.shape}")

# 4. (Facoltativo) salva su file per addestramento
np.save('X_window.npy', X)
np.save('y_window.npy', y)
print("Salvati file: X_window.npy e y_window.npy")
```

```
Sliding window completato!
X shape: (69693, 30, 7)
y shape: (69693,)
Salvati file: X_window.npy e y_window.npy
```

LSTM

stima del valore al passo successivo, predizione

i dati (X, Y) sono presi dallo sliding window creato nello script precedente iL modello agisce sulla variabile "barometro"

In [49]:
```
# 1. Carica i dati
X = np.load('X_window.npy')  # shape: (samples, timesteps, features)
y = np.load('y_window.npy')  # shape: (samples,)

# 2. Dividi in train/test
split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

print(f"Train shape: {X_train.shape}, Test shape: {X_test.shape}")

# 3. Costruisci il modello LSTM
model = Sequential([
    LSTM(64, input_shape=(X.shape[1], X.shape[2]), return_sequences=False),
    Dense(32, activation='relu'),
    Dense(1)  # regressione
])

model.compile(optimizer='adam', loss='mse')
model.summary()

# 4. Addestra il modello
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)

# 5. Valutazione
y_pred = model.predict(X_test).flatten()

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)

print(f"\n RMSE: {rmse:.4f}")
print(f" MAE:  {mae:.4f}")

# 6. Grafico predizioni vs target
plt.figure(figsize=(10, 5))
plt.plot(y_test[:200], label='True')
plt.plot(y_pred[:200], label='Predicted')
plt.title('True vs Predicted (prime 200 predizioni)')
plt.xlabel('Time Step')
plt.ylabel('Valore target')
plt.legend()
plt.tight_layout()
plt.show()

# 7. Salva il modello
model.save('lstm_etna_model.h5')
print(" Modello salvato come 'lstm_etna_model.h5'")
```

```
Train shape: (55754, 30, 7), Test shape: (13939, 30, 7)
Model: "sequential"
```

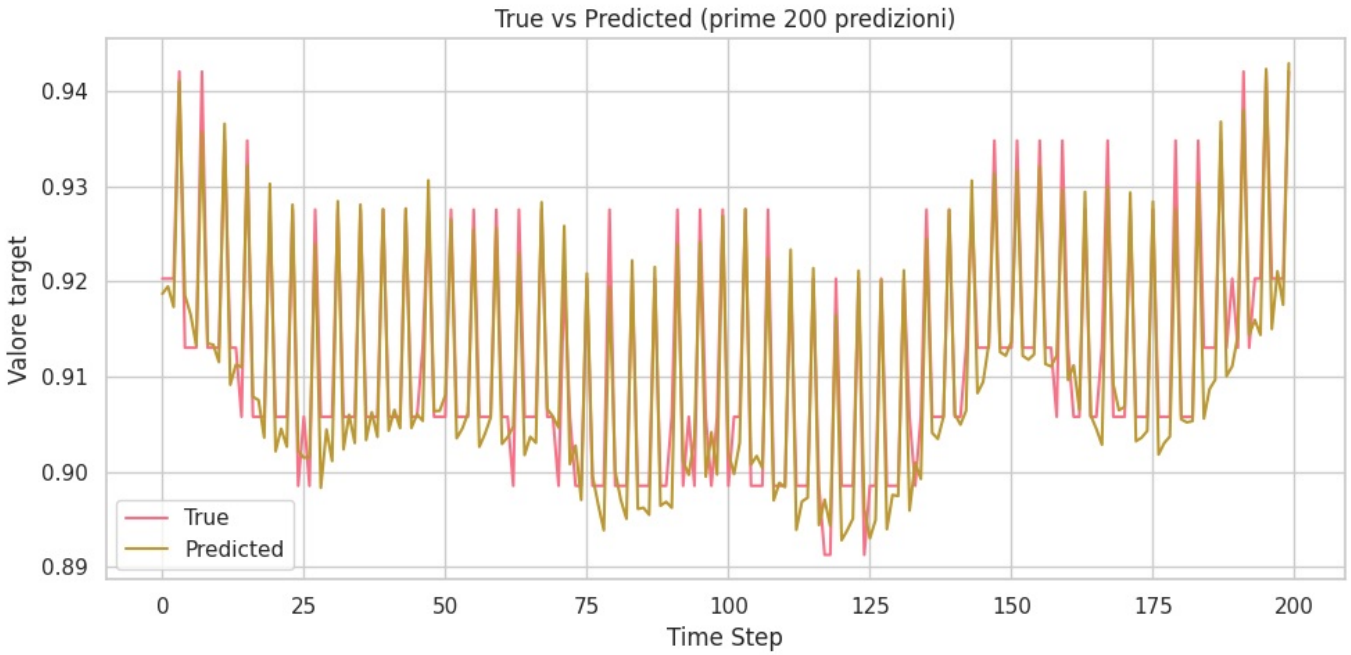
Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	18,432
dense (Dense)	(None, 32)	2,080
dense_1 (Dense)	(None, 1)	33

Total params: 20,545 (80.25 KB)

Trainable params: 20,545 (80.25 KB)

Non-trainable params: 0 (0.00 B)

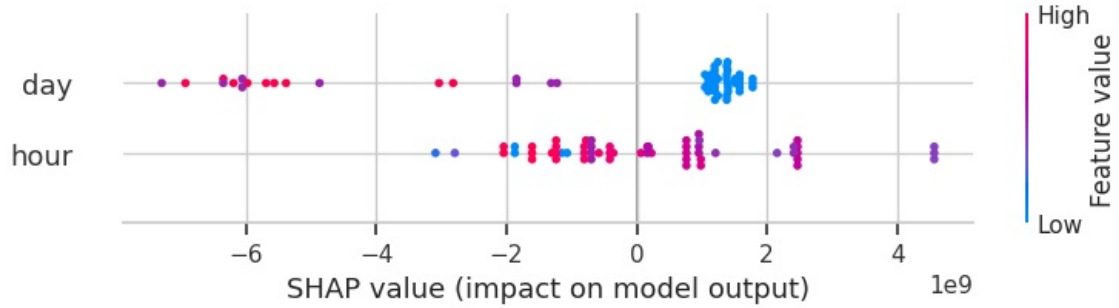
```
Epoch 1/100
1394/1394  9s 5ms/step - loss: 0.0033 - val_loss: 2.2339e-04
Epoch 2/100
1394/1394  6s 5ms/step - loss: 1.9588e-04 - val_loss: 2.4846e-04
Epoch 3/100
1394/1394  6s 5ms/step - loss: 1.7789e-04 - val_loss: 2.2440e-04
Epoch 4/100
1394/1394  7s 5ms/step - loss: 1.6502e-04 - val_loss: 1.6671e-04
Epoch 5/100
1394/1394  6s 5ms/step - loss: 1.5197e-04 - val_loss: 1.9482e-04
Epoch 6/100
1394/1394  6s 5ms/step - loss: 1.5370e-04 - val_loss: 1.2474e-04
Epoch 7/100
1394/1394  6s 4ms/step - loss: 1.4993e-04 - val_loss: 1.2497e-04
Epoch 8/100
1394/1394  6s 5ms/step - loss: 1.2581e-04 - val_loss: 5.8112e-05
Epoch 9/100
1394/1394  6s 5ms/step - loss: 6.8535e-05 - val_loss: 5.6445e-05
Epoch 10/100
1394/1394  6s 5ms/step - loss: 5.1232e-05 - val_loss: 1.8937e-05
Epoch 11/100
1394/1394  6s 5ms/step - loss: 2.8627e-05 - val_loss: 3.3513e-05
Epoch 12/100
1394/1394  6s 5ms/step - loss: 2.4008e-05 - val_loss: 2.1047e-05
Epoch 13/100
1394/1394  6s 5ms/step - loss: 2.3085e-05 - val_loss: 1.5488e-05
Epoch 14/100
1394/1394  6s 5ms/step - loss: 2.2869e-05 - val_loss: 3.3978e-05
Epoch 15/100
1394/1394  6s 5ms/step - loss: 2.2785e-05 - val_loss: 1.8425e-05
Epoch 16/100
1394/1394  6s 5ms/step - loss: 2.1806e-05 - val_loss: 1.4328e-05
Epoch 17/100
1394/1394  6s 5ms/step - loss: 2.3910e-05 - val_loss: 1.2018e-05
Epoch 18/100
1394/1394  6s 5ms/step - loss: 2.3132e-05 - val_loss: 1.8434e-05
Epoch 19/100
1394/1394  6s 5ms/step - loss: 2.0513e-05 - val_loss: 1.7929e-05
Epoch 20/100
1394/1394  6s 4ms/step - loss: 2.0678e-05 - val_loss: 1.6465e-05
Epoch 21/100
1394/1394  6s 5ms/step - loss: 2.0269e-05 - val_loss: 2.0271e-05
Epoch 22/100
1394/1394  6s 5ms/step - loss: 2.3068e-05 - val_loss: 1.1581e-05
Epoch 23/100
1394/1394  6s 5ms/step - loss: 1.9323e-05 - val_loss: 1.1640e-05
Epoch 24/100
1394/1394  6s 5ms/step - loss: 1.8465e-05 - val_loss: 3.1719e-05
Epoch 25/100
1394/1394  6s 5ms/step - loss: 1.8803e-05 - val_loss: 2.3510e-05
Epoch 26/100
1394/1394  6s 5ms/step - loss: 1.9608e-05 - val_loss: 1.3819e-05
Epoch 27/100
1394/1394  6s 5ms/step - loss: 1.8963e-05 - val_loss: 1.5579e-05
Epoch 28/100
1394/1394  6s 5ms/step - loss: 1.8581e-05 - val_loss: 1.1791e-05
Epoch 29/100
1394/1394  6s 5ms/step - loss: 1.7957e-05 - val_loss: 9.7766e-05
Epoch 30/100
1394/1394  7s 5ms/step - loss: 2.0773e-05 - val_loss: 1.8681e-05
Epoch 31/100
1394/1394  6s 5ms/step - loss: 1.7389e-05 - val_loss: 3.0851e-05
Epoch 32/100
1394/1394  6s 5ms/step - loss: 1.9139e-05 - val_loss: 1.4182e-05
436/436  1s 2ms/step

 RMSE: 0.0127
 MAE:  0.0029
```

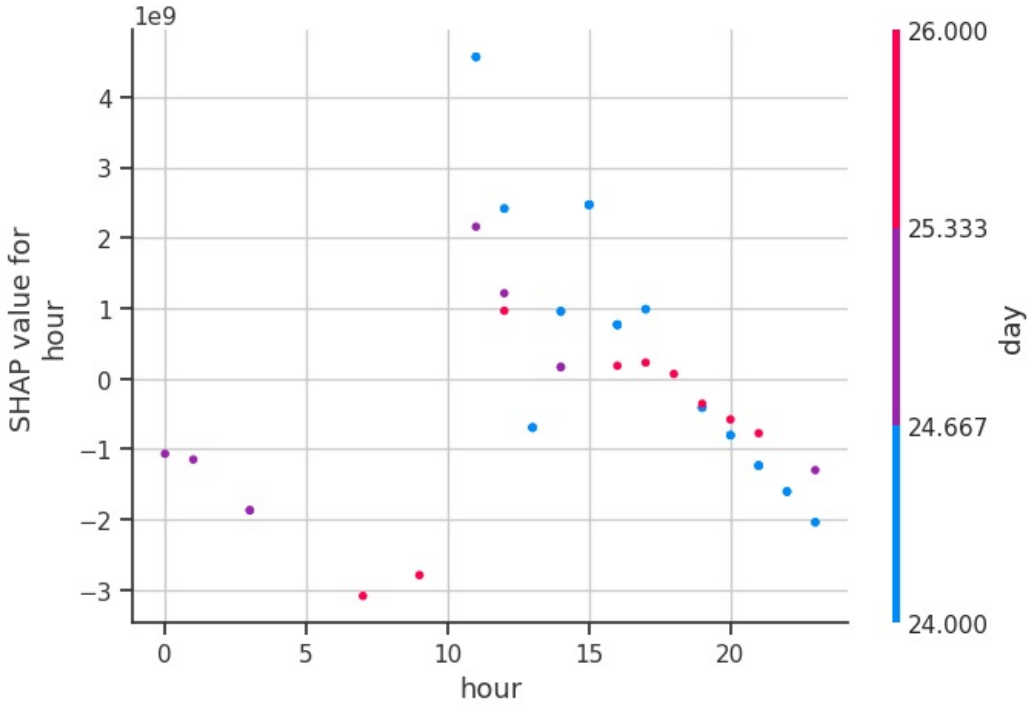



```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Modello salvato come 'lstm_etna_model.h5'
```

In [50]:

```python
# Install shap if needed: pip install shap

import pandas as pd
import shap
import xgboost
from sklearn.model_selection import train_test_split

# 1. Load and prepare the data
df = pd.read_csv('heat_flux_cleaned.csv')
df['date'] = pd.to_datetime(df['date'])
df['hour'] = df['date'].dt.hour
df['day'] = df['date'].dt.day

# 2. Define features and target
X = df.drop(['Radiant Heat Flux [W]', 'date'], axis=1)
y = df['Radiant Heat Flux [W]']

# 3. Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 4. Train XGBoost model
model = xgboost.XGBRegressor()
model.fit(X_train, y_train)

# 5. SHAP analysis
shap.initjs()  # Enables JS visualizations

explainer = shap.Explainer(model, X_train)  # Oppure shap.TreeExplainer(model)
shap_values = explainer(X_test)

# Summary plot (feature importance)
shap.summary_plot(shap_values.values, X_test)  # Usa .values per estrarre l'array

# Dependence plot for a specific feature
shap.dependence_plot("hour", shap_values.values, X_test)  # Usa .values qui

# Force plot for a single prediction
shap.plots.force(shap_values[0])
```





Out[50]:

Visualization omitted, Javascript library not loaded!

Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

Early warning

In [51]:
```
# Esegui in una NUOVA sessione (riavvia il runtime/kernel prima!)
import tensorflow as tf
print("Versione TensorFlow:", tf.__version__)
```

```
Versione TensorFlow: 2.19.0
```

In [52]:
```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential, save_model
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler

# 1. Caricamento dati
def safe_load(filepath):
    df = pd.read_csv(filepath, parse_dates=['Data'], index_col='Data')
    rms = df['RMS'].ffill().bfill().values.reshape(-1, 1)
    scaler = MinMaxScaler()
    return scaler.fit_transform(rms), scaler

# 2. Classe modello
class SafeLSTM:
```

```python
    def __init__(self):
        self.model = Sequential([
            LSTM(64, input_shape=(30, 1)),  # <-- Aggiunto parentesi chiusa mancante!
            Dense(5)
        ])
        self.model.compile(optimizer='adam', loss='mse')

    def train(self, X, y):
        history = self.model.fit(X, y, epochs=50, batch_size=32, verbose=1)
        return history

# 3. Esecuzione corretta
if __name__ == "__main__":
    # Caricamento dati
    data, scaler = safe_load("tremore vulcanico_cleaned.csv")

    # Creazione sequenze ALLINEATE
    seq_length = 30
    pred_steps = 5

    X = np.array([data[i:i+seq_length] for i in range(len(data)-seq_length-pred_steps+1)])
    y = np.array([data[i+seq_length:i+seq_length+pred_steps] for i in range(len(data)-seq_length-pred_steps+1)])

    print(f" Dimensioni allineate - X: {X.shape}, y: {y.shape}")

    # Addestramento
    lstm = SafeLSTM()
    print(" Training in ambiente protetto...")
    history = lstm.train(X[:2000], y[:2000])  # Usa subset per test

    save_model(lstm.model, "safe_model.h5")
    print(" Modello salvato correttamente")
```

```
 Dimensioni allineate - X: (696, 30, 1), y: (696, 5, 1)
 Training in ambiente protetto...
Epoch 1/50
22/22  1s 5ms/step - loss: 0.1815
Epoch 2/50
22/22  0s 5ms/step - loss: 0.0287
Epoch 3/50
22/22  0s 5ms/step - loss: 0.0223
Epoch 4/50
22/22  0s 5ms/step - loss: 0.0206
Epoch 5/50
22/22  0s 5ms/step - loss: 0.0219
Epoch 6/50
22/22  0s 5ms/step - loss: 0.0192
Epoch 7/50
22/22  0s 5ms/step - loss: 0.0209
Epoch 8/50
22/22  0s 5ms/step - loss: 0.0188
Epoch 9/50
22/22  0s 5ms/step - loss: 0.0194
Epoch 10/50
22/22  0s 5ms/step - loss: 0.0216
Epoch 11/50
22/22  0s 5ms/step - loss: 0.0180
Epoch 12/50
22/22  0s 5ms/step - loss: 0.0168
Epoch 13/50
22/22  0s 5ms/step - loss: 0.0170
Epoch 14/50
22/22  0s 5ms/step - loss: 0.0183
Epoch 15/50
22/22  0s 5ms/step - loss: 0.0172
Epoch 16/50
22/22  0s 5ms/step - loss: 0.0173
Epoch 17/50
22/22  0s 5ms/step - loss: 0.0174
Epoch 18/50
22/22  0s 5ms/step - loss: 0.0164
Epoch 19/50
22/22  0s 5ms/step - loss: 0.0172
Epoch 20/50
22/22  0s 5ms/step - loss: 0.0165
Epoch 21/50
22/22  0s 5ms/step - loss: 0.0155
Epoch 22/50
22/22  0s 5ms/step - loss: 0.0161
Epoch 23/50
22/22  0s 5ms/step - loss: 0.0167
Epoch 24/50
22/22  0s 5ms/step - loss: 0.0155
Epoch 25/50
```

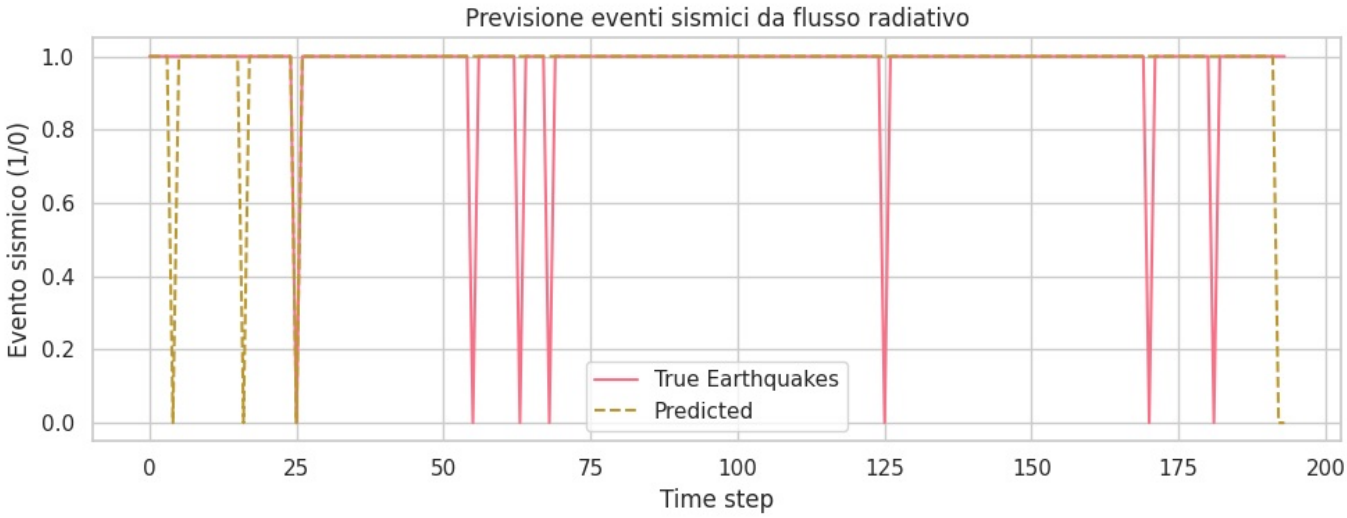
```
22/22  0s 5ms/step - loss: 0.0154
Epoch 26/50
22/22  0s 5ms/step - loss: 0.0157
Epoch 27/50
22/22  0s 5ms/step - loss: 0.0153
Epoch 28/50
22/22  0s 5ms/step - loss: 0.0157
Epoch 29/50
22/22  0s 5ms/step - loss: 0.0152
Epoch 30/50
22/22  0s 5ms/step - loss: 0.0157
Epoch 31/50
22/22  0s 5ms/step - loss: 0.0163
Epoch 32/50
22/22  0s 5ms/step - loss: 0.0145
Epoch 33/50
22/22  0s 5ms/step - loss: 0.0148
Epoch 34/50
22/22  0s 5ms/step - loss: 0.0152
Epoch 35/50
22/22  0s 5ms/step - loss: 0.0157
Epoch 36/50
22/22  0s 5ms/step - loss: 0.0152
Epoch 37/50
22/22  0s 5ms/step - loss: 0.0140
Epoch 38/50
22/22  0s 5ms/step - loss: 0.0150
Epoch 39/50
22/22  0s 5ms/step - loss: 0.0162
Epoch 40/50
22/22  0s 5ms/step - loss: 0.0145
Epoch 41/50
22/22  0s 5ms/step - loss: 0.0154
Epoch 42/50
22/22  0s 5ms/step - loss: 0.0156
Epoch 43/50
22/22  0s 5ms/step - loss: 0.0135
Epoch 44/50
22/22  0s 5ms/step - loss: 0.0145
Epoch 45/50
22/22  0s 5ms/step - loss: 0.0162
Epoch 46/50
22/22  0s 5ms/step - loss: 0.0131
Epoch 47/50
22/22  0s 5ms/step - loss: 0.0138
Epoch 48/50
22/22  0s 5ms/step - loss: 0.0153
Epoch 49/50
22/22  0s 4ms/step - loss: 0.0132
Epoch 50/50
22/22  0s 5ms/step - loss: 0.0144
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Modello salvato correttamente

Simulazione
```

In [53]:

```python
# 1. CARICAMENTO DEI DATI
# Simulazione: generiamo un flusso oscillante + sismicità
time_steps = 1000
t = np.arange(time_steps)

# Flusso radiativo simulato (oscillante + trend)
flux = 0.5 * np.sin(2 * np.pi * t / 50) + 0.01 * t + np.random.normal(0, 0.05, size=time_steps)

# Label terremoti (eventi = 1) se il flusso supera una soglia, con rumore
earthquakes = (flux + np.random.normal(0, 0.05, size=time_steps)) > 0.8
earthquakes = earthquakes.astype(int)

# Salviamo tutto in un DataFrame
df = pd.DataFrame({
    'flux': flux,
    'quake': earthquakes
})

# 2. PREPROCESSING DATI
scaler = MinMaxScaler()
df['flux'] = scaler.fit_transform(df[['flux']])

# Funzione per creare sequenze LSTM
def create_sequences(data, labels, seq_length):
    X, y = [], []
```

```
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(labels[i + seq_length])
    return np.array(X), np.array(y)

seq_length = 30
X, y = create_sequences(df['flux'].values, df['quake'].values, seq_length)

X = X.reshape((X.shape[0], X.shape[1], 1))  # LSTM expects 3D

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. MODELLO LSTM
model = Sequential()
model.add(LSTM(64, input_shape=(seq_length, 1), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # output binario (probabilità di terremoto)

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=15, batch_size=32, validation_split=0.2)

# 4. VALUTAZIONE
loss, acc = model.evaluate(X_test, y_test)
print(f"\n Accuracy: {acc:.2f}")

# 5. GRAFICO RISULTATI
y_pred = (model.predict(X_test) > 0.5).astype(int)

plt.figure(figsize=(10,4))
plt.plot(y_test[:200], label='True Earthquakes')
plt.plot(y_pred[:200], label='Predicted', linestyle='--')
plt.title("Previsione eventi sismici da flusso radiativo")
plt.xlabel("Time step")
plt.ylabel("Evento sismico (1/0)")
plt.legend()
plt.tight_layout()
plt.show()
```

```
Epoch 1/15
20/20  3s 19ms/step - accuracy: 0.6382 - loss: 0.6463 - val_accuracy: 0.9359 - val_loss: 0.2098
Epoch 2/15
20/20  0s 8ms/step - accuracy: 0.9436 - loss: 0.1841 - val_accuracy: 0.9359 - val_loss: 0.1494
Epoch 3/15
20/20  0s 9ms/step - accuracy: 0.9498 - loss: 0.1221 - val_accuracy: 0.9359 - val_loss: 0.1168
Epoch 4/15
20/20  0s 9ms/step - accuracy: 0.9445 - loss: 0.0904 - val_accuracy: 0.9359 - val_loss: 0.0984
Epoch 5/15
20/20  0s 9ms/step - accuracy: 0.9477 - loss: 0.0794 - val_accuracy: 0.9551 - val_loss: 0.0905
Epoch 6/15
20/20  0s 8ms/step - accuracy: 0.9622 - loss: 0.0817 - val_accuracy: 0.9487 - val_loss: 0.0893
Epoch 7/15
20/20  0s 8ms/step - accuracy: 0.9677 - loss: 0.0691 - val_accuracy: 0.9487 - val_loss: 0.0876
Epoch 8/15
20/20  0s 8ms/step - accuracy: 0.9625 - loss: 0.0645 - val_accuracy: 0.9423 - val_loss: 0.0853
Epoch 9/15
20/20  0s 8ms/step - accuracy: 0.9515 - loss: 0.0806 - val_accuracy: 0.9487 - val_loss: 0.1037
Epoch 10/15
20/20  0s 8ms/step - accuracy: 0.9749 - loss: 0.0765 - val_accuracy: 0.9487 - val_loss: 0.0877
Epoch 11/15
20/20  0s 8ms/step - accuracy: 0.9751 - loss: 0.0686 - val_accuracy: 0.9487 - val_loss: 0.0818
Epoch 12/15
20/20  0s 8ms/step - accuracy: 0.9677 - loss: 0.0635 - val_accuracy: 0.9615 - val_loss: 0.0888
Epoch 13/15
20/20  0s 8ms/step - accuracy: 0.9685 - loss: 0.0722 - val_accuracy: 0.9551 - val_loss: 0.0805
Epoch 14/15
20/20  0s 8ms/step - accuracy: 0.9694 - loss: 0.0638 - val_accuracy: 0.9679 - val_loss: 0.0859
Epoch 15/15
20/20  0s 8ms/step - accuracy: 0.9706 - loss: 0.0882 - val_accuracy: 0.9487 - val_loss: 0.0774
7/7  0s 6ms/step - accuracy: 0.9492 - loss: 0.0753

 Accuracy: 0.95
7/7  0s 17ms/step
```



Terremoti previsione

In [54]:
```
# COSTRUISCE la colonna 'quake_next24h' nel dataframe principale
# 1. Carica i dati terremoto
quakes = pd.read_csv('terremoti_puliti.csv', parse_dates=['Datetime'])

# 2. Carica il file unificato
df = pd.read_csv('merged_etna_data.csv', index_col=0, parse_dates=True)
df.index = pd.to_datetime(df.index, errors='coerce', format='mixed')


# 3. Crea la colonna target: 1 se un terremoto avviene entro le successive 24 ore
df['quake_next24h'] = 0

for quake_time in quakes['Datetime']:
    window_start = quake_time - pd.Timedelta(hours=24)
    df.loc[(df.index >= window_start) & (df.index < quake_time), 'quake_next24h'] = 1

# 4. Salva il nuovo file
df.to_csv('merged_etna_with_quake_target.csv')
print(" Colonna 'quake_next24h' creata e file salvato.")
```

```
Colonna 'quake_next24h' creata e file salvato.
```

In [55]:
```
df = pd.read_csv("merged_etna_with_quake_target.csv", parse_dates=True, index_col=0)
```

In [56]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import (
    classification_report, confusion_matrix,
    precision_score, recall_score, f1_score, roc_auc_score
)
from sklearn.utils import class_weight
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# 1. Carica e prepara il dataset
data = pd.read_csv("terremoti_puliti.csv", parse_dates=['Datetime'], index_col='Datetime')
data = data.sort_index()

# 2. Crea la variabile target (1 se nelle prossime 24 ore ci sarà un terremoto con ML >= 2.0)
# Definiamo una soglia per considerare un terremoto "significativo"
ML_THRESHOLD = 2.0
WINDOW_HOURS = 24

# Crea la colonna target
data['quake_next24h'] = 0
for i in range(len(data)):
    current_time = data.index[i]
```

```
    future_window = current_time + pd.Timedelta(hours=WINDOW_HOURS)
    # Verifica se c'è almeno un terremoto significativo nelle prossime 24 ore
    if any((data.index > current_time) &
           (data.index <= future_window) &
           (data['ML'] >= ML_THRESHOLD)):
        data.iloc[i, -1] = 1  # Imposta a 1 se c'è un terremoto significativo

# Seleziona solo le colonne numeriche ed elimina NA
features = data.drop(columns=['quake_next24h']).select_dtypes(include='number').interpolate().dropna()
labels = data['quake_next24h'].loc[features.index]

# 3. Funzione per creare finestre temporali
def create_sequences(df, target, window_size):
    X, y = [], []
    for i in range(len(df) - window_size):
        seq_x = df.iloc[i:i+window_size].values
        seq_y = target.iloc[i+window_size]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

# 4. Crea le sequenze
WINDOW_SIZE = 24  # Usiamo 24 ore di dati storici
X, y = create_sequences(features, labels, WINDOW_SIZE)

# 5. Split train/test
split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# 6. Calcolo class weights per sbilanciamento
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train),
    y=y_train
)
class_weights = dict(zip(np.unique(y_train), class_weights))
print("Class weights:", class_weights)

# 7. Costruisci il modello LSTM
model = Sequential([
    LSTM(64, input_shape=(X.shape[1], X.shape[2])),  # Solo l'ultimo output
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

# 8. Addestramento del modello
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    class_weight=class_weights,
    verbose=1
)

# 9. Valutazione sul test set
y_pred_proba = model.predict(X_test).flatten()
y_pred = (y_pred_proba > 0.5).astype(int)

print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=4))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred))
print("ROC AUC:", roc_auc_score(y_test, y_pred_proba))

# 10. Visualizza curva delle probabilità
plt.figure(figsize=(10,4))
plt.hist(y_pred_proba, bins=100, color='orange')
plt.title("Distribuzione delle probabilità predette")
plt.xlabel("Probabilità predetta di terremoto")
```

```python
plt.ylabel("Frequenza")
plt.grid(True)
plt.show()

# 11. Salva il modello
model.save("lstm_earthquake_predictor.keras")
```

```
Class weights: {np.int64(0): np.float64(0.6751152073732719), np.int64(1): np.float64(1.9276315789473684)}
```

Model: "sequential_3"

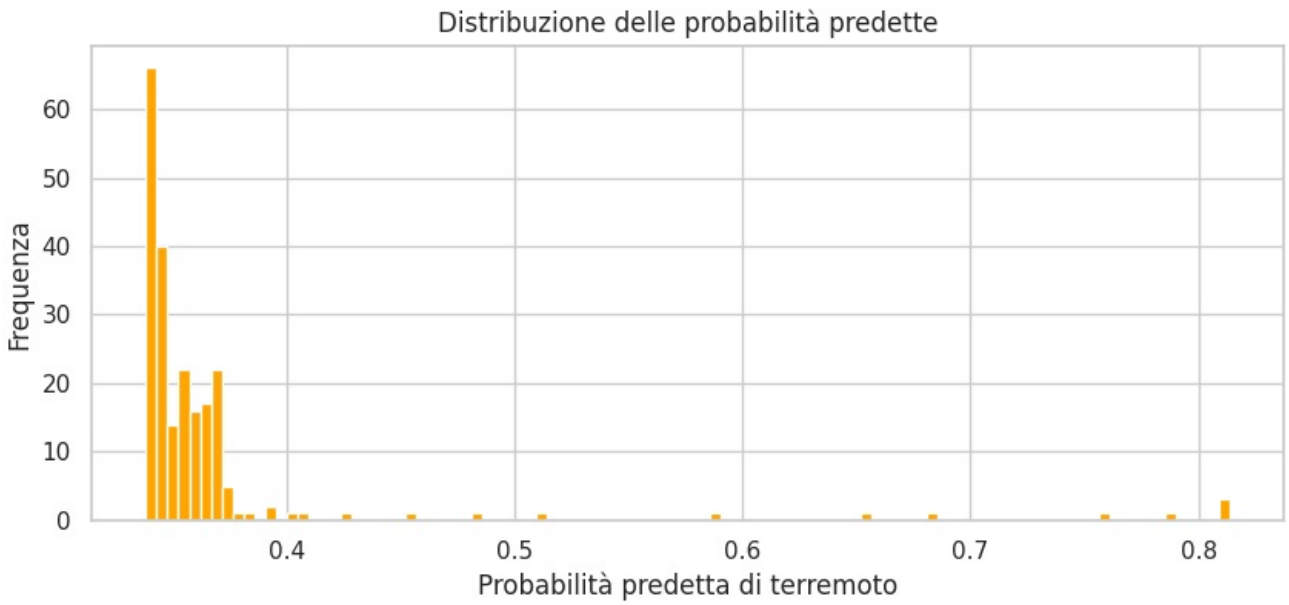
Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(None, 64)	17,664
dropout_1 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 32)	2,080
dense_6 (Dense)	(None, 1)	33

Total params: 19,777 (77.25 KB)

Trainable params: 19,777 (77.25 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
22/22  2s 18ms/step - accuracy: 0.5864 - loss: 0.6822 - val_accuracy: 0.7614 - val_loss: 0.6166
Epoch 2/50
22/22  0s 8ms/step - accuracy: 0.6159 - loss: 0.6782 - val_accuracy: 0.7500 - val_loss: 0.6244
Epoch 3/50
22/22  0s 8ms/step - accuracy: 0.6403 - loss: 0.6430 - val_accuracy: 0.7557 - val_loss: 0.5868
Epoch 4/50
22/22  0s 8ms/step - accuracy: 0.7249 - loss: 0.6288 - val_accuracy: 0.7557 - val_loss: 0.6175
Epoch 5/50
22/22  0s 8ms/step - accuracy: 0.6551 - loss: 0.6104 - val_accuracy: 0.7614 - val_loss: 0.6146
Epoch 6/50
22/22  0s 8ms/step - accuracy: 0.6336 - loss: 0.6342 - val_accuracy: 0.7841 - val_loss: 0.5873
Epoch 7/50
22/22  0s 8ms/step - accuracy: 0.7021 - loss: 0.6388 - val_accuracy: 0.7784 - val_loss: 0.5802
Epoch 8/50
22/22  0s 8ms/step - accuracy: 0.7008 - loss: 0.6243 - val_accuracy: 0.7784 - val_loss: 0.5688
Epoch 9/50
22/22  0s 8ms/step - accuracy: 0.7433 - loss: 0.6217 - val_accuracy: 0.7898 - val_loss: 0.5729
Epoch 10/50
22/22  0s 8ms/step - accuracy: 0.7006 - loss: 0.6226 - val_accuracy: 0.7898 - val_loss: 0.5633
Epoch 11/50
22/22  0s 8ms/step - accuracy: 0.7355 - loss: 0.6398 - val_accuracy: 0.7330 - val_loss: 0.6431
Epoch 12/50
22/22  0s 8ms/step - accuracy: 0.6606 - loss: 0.6382 - val_accuracy: 0.7557 - val_loss: 0.6222
Epoch 13/50
22/22  0s 8ms/step - accuracy: 0.6944 - loss: 0.6266 - val_accuracy: 0.7727 - val_loss: 0.6082
Epoch 14/50
22/22  0s 8ms/step - accuracy: 0.7402 - loss: 0.5978 - val_accuracy: 0.7557 - val_loss: 0.5979
Epoch 15/50
22/22  0s 8ms/step - accuracy: 0.6680 - loss: 0.6391 - val_accuracy: 0.7386 - val_loss: 0.6214
7/7  0s 17ms/step

Classification Report:
              precision    recall  f1-score   support

           0     0.7725    0.9760    0.8624       167
           1     0.5556    0.0943    0.1613        53

    accuracy                         0.7636       220
   macro avg     0.6640    0.5352    0.5119       220
weighted avg     0.7202    0.7636    0.6935       220

Confusion Matrix:
[[163   4]
 [ 48   5]]
Precision: 0.5555555555555556
Recall: 0.09433962264150944
F1-score: 0.16129032258064516
ROC AUC: 0.5049146989040786
```



Ottimizziamo per F1-score e recall perchè sono variabili più informativi in questo caso

-accuracy segnala eventi falsi -F1-score e recall evitano di perdere eventi reali e in questo caso è più urgente ed importante quest'ultima opzione

In [57]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import (
    classification_report, confusion_matrix,
    precision_score, recall_score, f1_score,
    roc_auc_score, precision_recall_curve, roc_curve, auc
)
from sklearn.utils import class_weight
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# 1. Carica e prepara il dataset
data = pd.read_csv("terremoti_puliti.csv", parse_dates=['Datetime'], index_col='Datetime')
data = data.sort_index()

ML_THRESHOLD = 2.0
WINDOW_HOURS = 24

data['quake_next24h'] = 0
for i in range(len(data)):
    current_time = data.index[i]
    future_window = current_time + pd.Timedelta(hours=WINDOW_HOURS)
    if any((data.index > current_time) & (data.index <= future_window) & (data['ML'] >= ML_THRESHOLD)):
        data.iloc[i, -1] = 1

features = data.drop(columns=['quake_next24h']).select_dtypes(include='number').interpolate().dropna()
labels = data['quake_next24h'].loc[features.index]

# 2. Creazione delle sequenze
def create_sequences(df, target, window_size):
    X, y = [], []
    for i in range(len(df) - window_size):
        seq_x = df.iloc[i:i+window_size].values
        seq_y = target.iloc[i+window_size]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)
```

```
WINDOW_SIZE = 24
X, y = create_sequences(features, labels, WINDOW_SIZE)

split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# 3. Class weights
class_weights = class_weight.compute_class_weight(
    class_weight='balanced', classes=np.unique(y_train), y=y_train
)
class_weights = dict(zip(np.unique(y_train), class_weights))
print("Class weights:", class_weights)

# 4. Modello LSTM
model = Sequential([
    LSTM(64, input_shape=(X.shape[1], X.shape[2])),
    Dropout(0.4),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

# 5. Addestramento
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
checkpoint = ModelCheckpoint("best_model.keras", monitor='val_loss', save_best_only=True)

history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop, checkpoint],
    class_weight=class_weights,
    verbose=1
)

# 6. Valutazione
model.load_weights("best_model.keras")
y_pred_proba = model.predict(X_test).flatten()

# Ottimizzazione soglia (F1-score)
thresholds = np.linspace(0.1, 0.9, 100)
f1_scores = [f1_score(y_test, y_pred_proba > t) for t in thresholds]
best_threshold = thresholds[np.argmax(f1_scores)]
print(f"\nBest threshold (massimo F1): {best_threshold:.2f}")

y_pred = (y_pred_proba > best_threshold).astype(int)

print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=4))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred))
print("ROC AUC:", roc_auc_score(y_test, y_pred_proba))

# 7. Curve PR e ROC
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)

plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plt.plot(recall, precision, label=f'F1 max @ {best_threshold:.2f}')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.grid(True)
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(fpr, tpr, label=f'AUC = {roc_auc_score(y_test, y_pred_proba):.2f}')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid(True)
plt.legend()
```

```python
plt.tight_layout()
plt.show()

# 8. Istogramma delle probabilità
plt.figure(figsize=(10, 4))
plt.hist(y_pred_proba, bins=100, color='orange')
plt.axvline(best_threshold, color='red', linestyle='--', label='Threshold ottimale')
plt.title("Distribuzione delle probabilità predette")
plt.xlabel("Probabilità predetta di terremoto")
plt.ylabel("Frequenza")
plt.grid(True)
plt.legend()
plt.show()

# 9. Salva il modello
model.save("lstm_earthquake_final.keras")
```

```
Class weights: {np.int64(0): np.float64(0.6751152073732719), np.int64(1): np.float64(1.9276315789473684)}
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 64)	17,664
dropout_2 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 32)	2,080
dropout_3 (Dropout)	(None, 32)	0
dense_8 (Dense)	(None, 1)	33

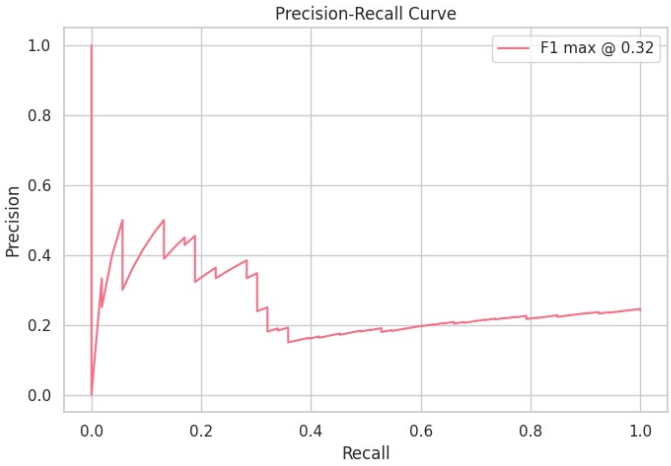
Total params: 19,777 (77.25 KB)

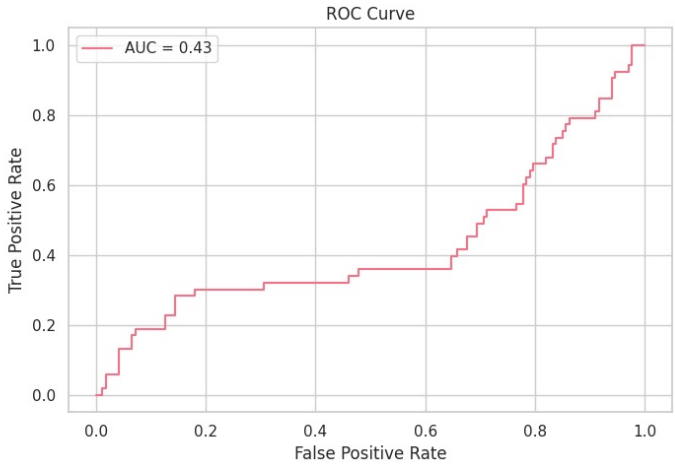
Trainable params: 19,777 (77.25 KB)

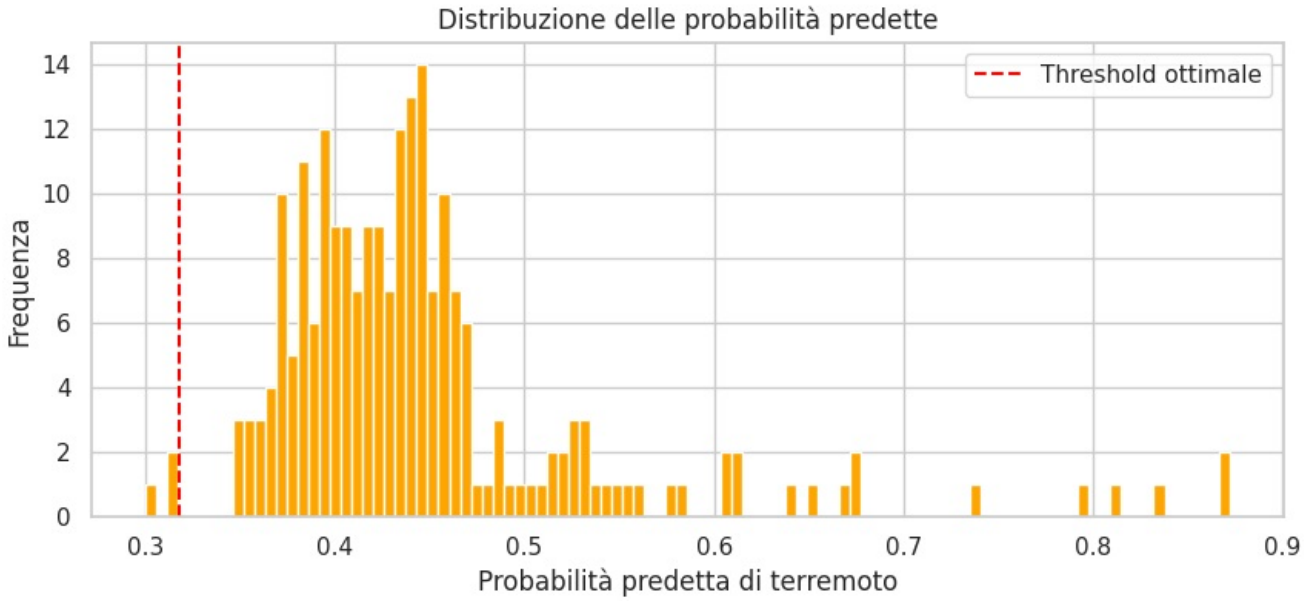
Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
22/22  2s 19ms/step - accuracy: 0.5662 - loss: 0.7492 - val_accuracy: 0.2045 - val_loss: 0.7658
Epoch 2/50
22/22  0s 9ms/step - accuracy: 0.5348 - loss: 0.6968 - val_accuracy: 0.2045 - val_loss: 0.7644
Epoch 3/50
22/22  0s 10ms/step - accuracy: 0.5075 - loss: 0.7118 - val_accuracy: 0.5966 - val_loss: 0.6880
Epoch 4/50
22/22  0s 8ms/step - accuracy: 0.4917 - loss: 0.6994 - val_accuracy: 0.4716 - val_loss: 0.7380
Epoch 5/50
22/22  0s 9ms/step - accuracy: 0.5481 - loss: 0.7097 - val_accuracy: 0.6477 - val_loss: 0.6772
Epoch 6/50
22/22  0s 10ms/step - accuracy: 0.6217 - loss: 0.6689 - val_accuracy: 0.6477 - val_loss: 0.6729
Epoch 7/50
22/22  0s 10ms/step - accuracy: 0.5691 - loss: 0.6876 - val_accuracy: 0.7273 - val_loss: 0.6687
Epoch 8/50
22/22  0s 8ms/step - accuracy: 0.6471 - loss: 0.6618 - val_accuracy: 0.6648 - val_loss: 0.6814
Epoch 9/50
22/22  0s 8ms/step - accuracy: 0.6426 - loss: 0.6454 - val_accuracy: 0.6193 - val_loss: 0.7216
Epoch 10/50
22/22  0s 8ms/step - accuracy: 0.5734 - loss: 0.6547 - val_accuracy: 0.6818 - val_loss: 0.6728
Epoch 11/50
22/22  0s 10ms/step - accuracy: 0.6671 - loss: 0.6520 - val_accuracy: 0.7273 - val_loss: 0.6577
Epoch 12/50
22/22  0s 10ms/step - accuracy: 0.7057 - loss: 0.6467 - val_accuracy: 0.7614 - val_loss: 0.6379
Epoch 13/50
22/22  0s 8ms/step - accuracy: 0.6270 - loss: 0.6826 - val_accuracy: 0.7216 - val_loss: 0.6821
Epoch 14/50
22/22  0s 9ms/step - accuracy: 0.6049 - loss: 0.6735 - val_accuracy: 0.7841 - val_loss: 0.6369
Epoch 15/50
22/22  0s 8ms/step - accuracy: 0.7640 - loss: 0.6220 - val_accuracy: 0.7784 - val_loss: 0.6458
Epoch 16/50
22/22  0s 8ms/step - accuracy: 0.7118 - loss: 0.6467 - val_accuracy: 0.7614 - val_loss: 0.6591
Epoch 17/50
22/22  0s 8ms/step - accuracy: 0.6825 - loss: 0.6715 - val_accuracy: 0.7955 - val_loss: 0.6424
Epoch 18/50
22/22  0s 9ms/step - accuracy: 0.7276 - loss: 0.6213 - val_accuracy: 0.7955 - val_loss: 0.6324
Epoch 19/50
22/22  0s 8ms/step - accuracy: 0.6984 - loss: 0.6518 - val_accuracy: 0.7443 - val_loss: 0.6579
Epoch 20/50
22/22  0s 9ms/step - accuracy: 0.6641 - loss: 0.6540 - val_accuracy: 0.7955 - val_loss: 0.6141
Epoch 21/50
22/22  0s 8ms/step - accuracy: 0.7408 - loss: 0.6386 - val_accuracy: 0.7386 - val_loss: 0.6533
Epoch 22/50
22/22  0s 8ms/step - accuracy: 0.6684 - loss: 0.6363 - val_accuracy: 0.7898 - val_loss: 0.6260
Epoch 23/50
22/22  0s 8ms/step - accuracy: 0.6837 - loss: 0.6157 - val_accuracy: 0.7614 - val_loss: 0.6390
Epoch 24/50
```

```
22/22  0s 8ms/step - accuracy: 0.6800 - loss: 0.6577 - val_accuracy: 0.7955 - val_loss: 0.6176
Epoch 25/50
22/22  0s 9ms/step - accuracy: 0.7468 - loss: 0.6349 - val_accuracy: 0.7898 - val_loss: 0.5981
Epoch 26/50
22/22  0s 10ms/step - accuracy: 0.7284 - loss: 0.6024 - val_accuracy: 0.7955 - val_loss: 0.5935
Epoch 27/50
22/22  0s 9ms/step - accuracy: 0.7404 - loss: 0.6489 - val_accuracy: 0.7727 - val_loss: 0.5829
Epoch 28/50
22/22  0s 8ms/step - accuracy: 0.7141 - loss: 0.6203 - val_accuracy: 0.7443 - val_loss: 0.6101
Epoch 29/50
22/22  0s 8ms/step - accuracy: 0.6613 - loss: 0.6535 - val_accuracy: 0.7614 - val_loss: 0.6204
Epoch 30/50
22/22  0s 8ms/step - accuracy: 0.7025 - loss: 0.6181 - val_accuracy: 0.7443 - val_loss: 0.6344
Epoch 31/50
22/22  0s 8ms/step - accuracy: 0.7056 - loss: 0.6126 - val_accuracy: 0.7841 - val_loss: 0.5966
Epoch 32/50
22/22  0s 9ms/step - accuracy: 0.7036 - loss: 0.6085 - val_accuracy: 0.7841 - val_loss: 0.5766
Epoch 33/50
22/22  0s 8ms/step - accuracy: 0.7095 - loss: 0.6271 - val_accuracy: 0.7614 - val_loss: 0.5884
Epoch 34/50
22/22  0s 8ms/step - accuracy: 0.7350 - loss: 0.6162 - val_accuracy: 0.7500 - val_loss: 0.6021
Epoch 35/50
22/22  0s 8ms/step - accuracy: 0.7601 - loss: 0.5470 - val_accuracy: 0.6250 - val_loss: 0.7099
Epoch 36/50
22/22  0s 8ms/step - accuracy: 0.7397 - loss: 0.5485 - val_accuracy: 0.7045 - val_loss: 0.6291
Epoch 37/50
22/22  0s 8ms/step - accuracy: 0.7829 - loss: 0.5529 - val_accuracy: 0.7841 - val_loss: 0.5901
7/7  0s 17ms/step

Best threshold (massimo F1): 0.32

Classification Report:
              precision    recall  f1-score   support

           0     1.0000    0.0180    0.0353       167
           1     0.2442    1.0000    0.3926        53

    accuracy                         0.2545       220
   macro avg     0.6221    0.5090    0.2139       220
weighted avg     0.8179    0.2545    0.1214       220

Confusion Matrix:
[[  3 164]
 [  0  53]]
Precision: 0.24423963133640553
Recall: 1.0
F1-score: 0.3925925925925926
ROC AUC: 0.4261665348548187
```







Riduciamo i falsi positivi e assicuriamoci di non perdere eventi(eruzioni) reali

In [58]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import (
    classification_report, confusion_matrix,
    precision_score, recall_score, f1_score,
    roc_auc_score, precision_recall_curve, roc_curve
)
from sklearn.utils import class_weight
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# ========================================
# 1. Caricamento e preparazione dati
# ========================================
data = pd.read_csv("terremoti_puliti.csv", parse_dates=['Datetime'], index_col='Datetime')
data = data.sort_index()

ML_THRESHOLD = 2.0
WINDOW_HOURS = 24

# Target: evento significativo nelle prossime 24h
data['quake_next24h'] = 0
for i in range(len(data)):
    current_time = data.index[i]
    future_window = current_time + pd.Timedelta(hours=WINDOW_HOURS)
    if any((data.index > current_time) & (data.index <= future_window) & (data['ML'] >= ML_THRESHOLD)):
        data.iloc[i, -1] = 1

# Feature numeriche
features = data.drop(columns=['quake_next24h']).select_dtypes(include='number').interpolate().dropna()
labels = data['quake_next24h'].loc[features.index]

# ========================================
```

```python
# 2. Creazione sequenze temporali
# ========================================
def create_sequences(df, target, window_size):
    X, y = [], []
    for i in range(len(df) - window_size):
        seq_x = df.iloc[i:i+window_size].values
        seq_y = target.iloc[i+window_size]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

WINDOW_SIZE = 24
X, y = create_sequences(features, labels, WINDOW_SIZE)

# Train-test split
split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# ========================================
# 3. Class weights per sbilanciamento
# ========================================
class_weights = class_weight.compute_class_weight(
    class_weight='balanced', classes=np.unique(y_train), y=y_train
)
class_weights = dict(zip(np.unique(y_train), class_weights))
print("Class weights:", class_weights)

# ========================================
# 4. Modello LSTM
# ========================================
model = Sequential([
    LSTM(64, input_shape=(X.shape[1], X.shape[2])),
    Dropout(0.4),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

# ========================================
# 5. Addestramento
# ========================================
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
checkpoint = ModelCheckpoint("best_model.keras", monitor='val_loss', save_best_only=True)

history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop, checkpoint],
    class_weight=class_weights,
    verbose=1
)

# ========================================
# 6. Valutazione
# ========================================
model.load_weights("best_model.keras")
y_pred_proba = model.predict(X_test).flatten()

# === Nuovo criterio: soglia per recall >= 0.90
target_recall = 0.90
thresholds = np.linspace(0, 1, 1001)
best_t = 0.5
best_prec = 0
for t in thresholds:
    preds = (y_pred_proba > t).astype(int)
    r = recall_score(y_test, preds)
    p = precision_score(y_test, preds, zero_division=0)
    if r >= target_recall and p > best_prec:
        best_prec = p
        best_t = t

print(f"\nSoglia scelta (recall >= {target_recall}): {best_t:.3f} | Precision: {best_prec:.3f}")

# ========================================
# 7. Post-processing: smoothing temporale
# ========================================
def consecutive_alerts(preds, k=2):
    out = np.zeros_like(preds)
```

```
    c = 0
    for i, v in enumerate(preds):
        if v == 1:
            c += 1
            if c >= k:
                out[i] = 1
        else:
            c = 0
    return out

y_pred_bin = (y_pred_proba > best_t).astype(int)
y_pred_smooth = consecutive_alerts(y_pred_bin, k=2)

# ========================================
# 8. Metriche finali
# ========================================
print("\n=== Risultati con soglia ottimizzata e smoothing ===")
print(classification_report(y_test, y_pred_smooth, digits=4))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_smooth))
print("Precision:", precision_score(y_test, y_pred_smooth))
print("Recall:", recall_score(y_test, y_pred_smooth))
print("F1-score:", f1_score(y_test, y_pred_smooth))
print("ROC AUC:", roc_auc_score(y_test, y_pred_proba))

# False alarm rate per giorno
fa_rate = ((y_test == 0) & (y_pred_smooth == 1)).sum() / (len(y_test)/24)
print(f"False Alarm Rate (al giorno): {fa_rate:.2f}")

# ========================================
# 9. Curve PR e ROC
# ========================================
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)

plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plt.plot(recall, precision, label=f'Threshold {best_t:.2f}')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.grid(True)
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(fpr, tpr, label=f'AUC = {roc_auc_score(y_test, y_pred_proba):.2f}')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# ========================================
# 10. Istogramma delle probabilità
# ========================================
plt.figure(figsize=(10, 4))
plt.hist(y_pred_proba, bins=100, color='orange')
plt.axvline(best_t, color='red', linestyle='--', label='Threshold operativa')
plt.title("Distribuzione delle probabilità predette")
plt.xlabel("Probabilità predetta di terremoto")
plt.ylabel("Frequenza")
plt.grid(True)
plt.legend()
plt.show()

# ========================================
# 11. Salva modello
# ========================================
model.save("lstm_earthquake_final.keras")
```

```
Class weights: {np.int64(0): np.float64(0.6751152073732719), np.int64(1): np.float64(1.9276315789473684)}
```

Model: "sequential_5"

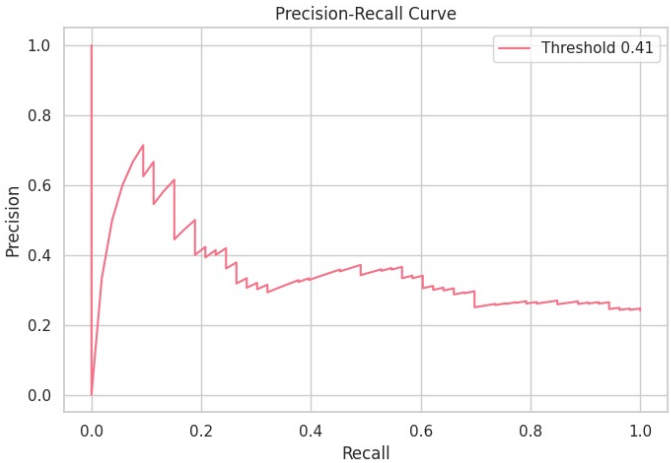
Layer (type)	Output Shape	Param #
lstm_5 (LSTM)	(None, 64)	17,664
dropout_4 (Dropout)	(None, 64)	0
dense_9 (Dense)	(None, 32)	2,080
dropout_5 (Dropout)	(None, 32)	0
dense_10 (Dense)	(None, 1)	33

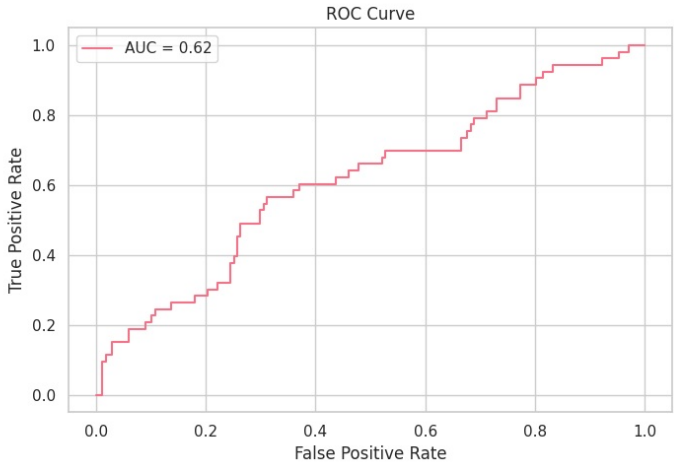
Total params: 19,777 (77.25 KB)

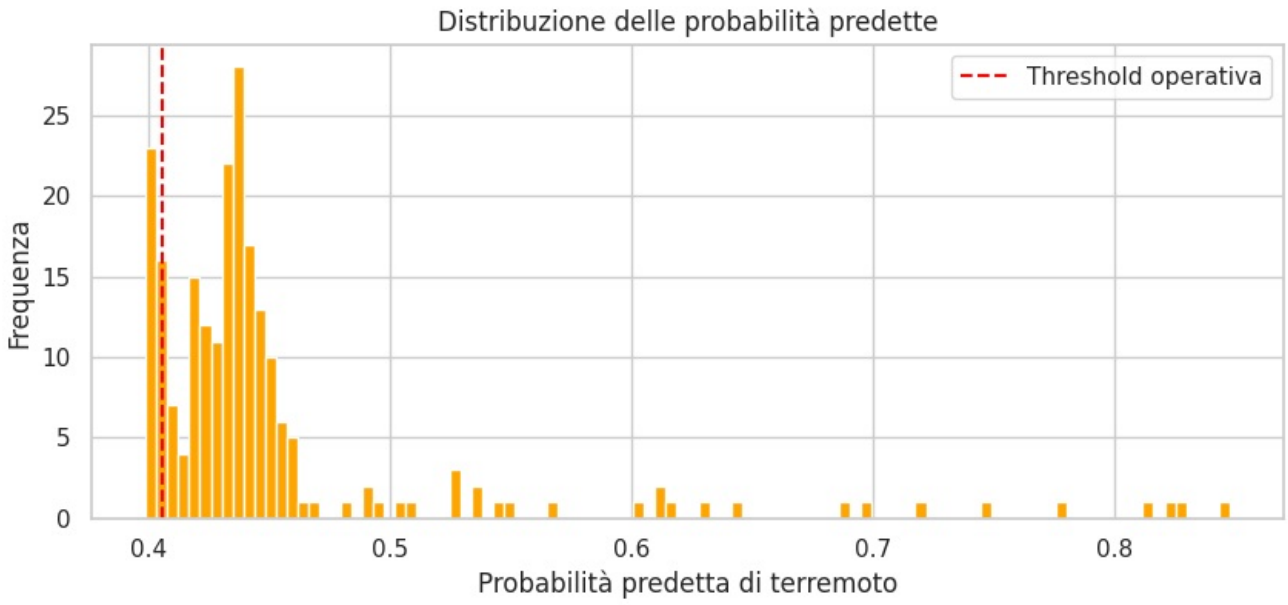
Trainable params: 19,777 (77.25 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
22/22  2s 20ms/step - accuracy: 0.4436 - loss: 0.7429 - val_accuracy: 0.6875 - val_loss: 0.6597
Epoch 2/50
22/22  0s 8ms/step - accuracy: 0.5286 - loss: 0.7458 - val_accuracy: 0.6250 - val_loss: 0.6758
Epoch 3/50
22/22  0s 8ms/step - accuracy: 0.5775 - loss: 0.6850 - val_accuracy: 0.6420 - val_loss: 0.6745
Epoch 4/50
22/22  0s 8ms/step - accuracy: 0.5725 - loss: 0.6684 - val_accuracy: 0.5170 - val_loss: 0.6936
Epoch 5/50
22/22  0s 9ms/step - accuracy: 0.5366 - loss: 0.6730 - val_accuracy: 0.7614 - val_loss: 0.6396
Epoch 6/50
22/22  0s 8ms/step - accuracy: 0.6544 - loss: 0.6482 - val_accuracy: 0.6364 - val_loss: 0.6820
Epoch 7/50
22/22  0s 8ms/step - accuracy: 0.5939 - loss: 0.6998 - val_accuracy: 0.7386 - val_loss: 0.6402
Epoch 8/50
22/22  0s 8ms/step - accuracy: 0.6512 - loss: 0.6584 - val_accuracy: 0.6989 - val_loss: 0.6541
Epoch 9/50
22/22  0s 8ms/step - accuracy: 0.6274 - loss: 0.6717 - val_accuracy: 0.7557 - val_loss: 0.6481
Epoch 10/50
22/22  0s 8ms/step - accuracy: 0.7059 - loss: 0.6237 - val_accuracy: 0.7386 - val_loss: 0.6607
7/7  0s 17ms/step

Soglia scelta (recall >= 0.9): 0.406 | Precision: 0.263

=== Risultati con soglia ottimizzata e smoothing ===
              precision    recall  f1-score   support

           0     0.8571    0.2515    0.3889       167
           1     0.2690    0.8679    0.4107        53

    accuracy                         0.4000       220
   macro avg     0.5631    0.5597    0.3998       220
weighted avg     0.7155    0.4000    0.3941       220

Confusion Matrix:
 [[ 42 125]
 [  7  46]]
Precision: 0.26900584795321636
Recall: 0.8679245283018868
F1-score: 0.4107142857142857
ROC AUC: 0.6160885775618574
False Alarm Rate (al giorno): 13.64
```







Più falsi positivi ma meno errori globali, che riesca a riconoscere meglio un evento da un non evento

In [59]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import (
    classification_report, confusion_matrix,
    precision_score, recall_score, f1_score,
    roc_auc_score, precision_recall_curve, roc_curve
)
from sklearn.utils import class_weight
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# === 1. Caricamento e preparazione dati ===
data = pd.read_csv("terremoti_puliti.csv", parse_dates=['Datetime'], index_col='Datetime')
data = data.sort_index()

ML_THRESHOLD = 2.0
WINDOW_HOURS = 24

# Creazione target binaria
data['quake_next24h'] = 0
for i in range(len(data)):
    current_time = data.index[i]
    future_window = current_time + pd.Timedelta(hours=WINDOW_HOURS)
    if any((data.index > current_time) & (data.index <= future_window) & (data['ML'] >= ML_THRESHOLD)):
        data.iloc[i, -1] = 1

# Seleziona feature numeriche
features = data.drop(columns=['quake_next24h']).select_dtypes(include='number').interpolate().dropna()
labels = data['quake_next24h'].loc[features.index]

# === 2. Creazione sequenze temporali ===
def create_sequences(df, target, window_size):
    X, y = [], []
    for i in range(len(df) - window_size):
        seq_x = df.iloc[i:i+window_size].values
        seq_y = target.iloc[i+window_size]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

WINDOW_SIZE = 24
X, y = create_sequences(features, labels, WINDOW_SIZE)

split = int(len(X) * 0.8)
```

```python
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# === 3. Class weights ===
class_weights = class_weight.compute_class_weight(
    class_weight='balanced', classes=np.unique(y_train), y=y_train
)
class_weights = dict(zip(np.unique(y_train), class_weights))
print("Class weights:", class_weights)

# === 4. Modello LSTM ===
model = Sequential([
    LSTM(64, input_shape=(X.shape[1], X.shape[2])),
    Dropout(0.4),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

# === 5. Addestramento ===
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
checkpoint = ModelCheckpoint("best_model.keras", monitor='val_loss', save_best_only=True)

history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop, checkpoint],
    class_weight=class_weights,
    verbose=1
)

# === 6. Valutazione ===
model.load_weights("best_model.keras")
y_pred_proba = model.predict(X_test).flatten()

# Soglia per recall >= 0.9
thresholds = np.linspace(0.1, 0.9, 100)
best_threshold = 0.5
for t in thresholds:
    if recall_score(y_test, (y_pred_proba > t).astype(int)) >= 0.9:
        best_threshold = t
        break
print(f"\nSoglia scelta (recall >= 0.9): {best_threshold:.3f}")

y_pred_bin = (y_pred_proba > best_threshold).astype(int)

# === 7. Smoothing k=3 ===
def consecutive_alerts(predictions, k=3):
    smoothed = predictions.copy()
    count = 0
    for i in range(len(predictions)):
        if predictions[i] == 1:
            count += 1
            if count < k:
                smoothed[i] = 0
        else:
            count = 0
    return smoothed

y_pred_smooth = consecutive_alerts(y_pred_bin, k=3)

# === 8. Vincoli fisici ===
# Esempio: tremore RMS > 0.5 e gas SO2 > 500
if 'RMS' in features.columns:
    rms_test = features['RMS'].iloc[-len(y_pred_smooth):].values
else:
    rms_test = np.ones(len(y_pred_smooth)) * 1  # Se manca, non filtra

if 'SO2' in features.columns:
    so2_test = features['SO2'].iloc[-len(y_pred_smooth):].values
else:
    so2_test = np.ones(len(y_pred_smooth)) * 1000

rms_threshold = 0.5
so2_threshold = 500

y_pred_final = [
    1 if (pred == 1 and rms > rms_threshold and so2 > so2_threshold) else 0
    for pred, rms, so2 in zip(y_pred_smooth, rms_test, so2_test)
```

```
]
y_pred_final = np.array(y_pred_final)

# === 9. Metriche finali ===
print("\n=== Risultati con smoothing k=3 + vincoli fisici ===")
print(classification_report(y_test, y_pred_final, digits=4))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_final))
print("Precision:", precision_score(y_test, y_pred_final))
print("Recall:", recall_score(y_test, y_pred_final))
print("F1-score:", f1_score(y_test, y_pred_final))
print("ROC AUC:", roc_auc_score(y_test, y_pred_proba))

# Calcolo False Alarm Rate (al giorno)
days_test = (features.index[-1] - features.index[-len(y_pred_final)]).days
false_alarms = ((y_pred_final == 1) & (y_test == 0)).sum()
far_day = false_alarms / days_test
print(f"False Alarm Rate (al giorno): {far_day:.2f}")

# === 10. Salva il modello ===
model.save("lstm_earthquake_final_filtered.keras")
```

```
Class weights: {np.int64(0): np.float64(0.6751152073732719), np.int64(1): np.float64(1.9276315789473684)}
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
lstm_6 (LSTM)	(None, 64)	17,664
dropout_6 (Dropout)	(None, 64)	0
dense_11 (Dense)	(None, 32)	2,080
dropout_7 (Dropout)	(None, 32)	0
dense_12 (Dense)	(None, 1)	33

Total params: 19,777 (77.25 KB)

Trainable params: 19,777 (77.25 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
22/22  2s 19ms/step - accuracy: 0.5715 - loss: 0.7150 - val_accuracy: 0.5682 - val_loss: 0.7084
Epoch 2/50
22/22  0s 9ms/step - accuracy: 0.4492 - loss: 0.7129 - val_accuracy: 0.6932 - val_loss: 0.6452
Epoch 3/50
22/22  0s 9ms/step - accuracy: 0.5563 - loss: 0.7046 - val_accuracy: 0.7670 - val_loss: 0.5998
Epoch 4/50
22/22  0s 8ms/step - accuracy: 0.6882 - loss: 0.6333 - val_accuracy: 0.5739 - val_loss: 0.7226
Epoch 5/50
22/22  0s 8ms/step - accuracy: 0.5384 - loss: 0.6622 - val_accuracy: 0.7670 - val_loss: 0.6494
Epoch 6/50
22/22  0s 8ms/step - accuracy: 0.6236 - loss: 0.6673 - val_accuracy: 0.7386 - val_loss: 0.6548
Epoch 7/50
22/22  0s 8ms/step - accuracy: 0.5910 - loss: 0.6883 - val_accuracy: 0.7614 - val_loss: 0.6497
Epoch 8/50
22/22  0s 8ms/step - accuracy: 0.6487 - loss: 0.6476 - val_accuracy: 0.7670 - val_loss: 0.6421
7/7  0s 17ms/step

Soglia scelta (recall >= 0.9): 0.100

=== Risultati con smoothing k=3 + vincoli fisici ===
              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000       167
           1     0.2339    0.9623    0.3764        53

    accuracy                         0.2318       220
   macro avg     0.1170    0.4811    0.1882       220
weighted avg     0.0564    0.2318    0.0907       220

Confusion Matrix:
[[  0 167]
 [  2  51]]
Precision: 0.23394495412844038
Recall: 0.9622641509433962
F1-score: 0.3763837638376384
ROC AUC: 0.6621850638345949
False Alarm Rate (al giorno): 4.64
```

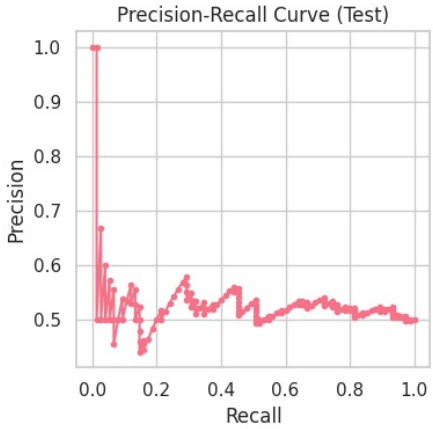
Riamuntano i falsi positivi, sono tanti, ma

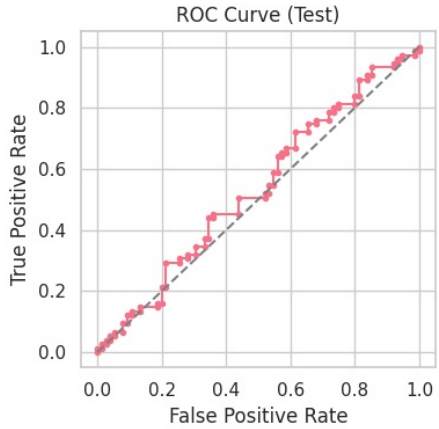
proviamo nuovamente a migliorare le performance e predire benissimo gli eventi veir senza mai perderli

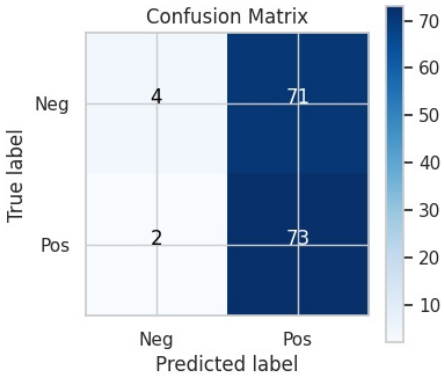
In [60]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import precision_recall_curve, f1_score, confusion_matrix, roc_auc_score, roc_curve, precis
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# ========================================
# 1) CARICAMENTO / PREPROCESSING DEI DATI
# ========================================
# X e y devono essere già preprocessati (scalati e con shape corretta)
# X.shape -> (n_samples, timesteps, n_features)
# y.shape -> (n_samples,)

# ESEMPIO: qui metto dati fittizi
# Sostituisci con i tuoi dati reali
n_samples = 1000
timesteps = 10
n_features = 5
X = np.random.rand(n_samples, timesteps, n_features)
y = np.random.randint(0, 2, size=n_samples)

# Suddivisione train (70%), val (15%), test (15%)
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp
)

# ========================================
# 2) DEFINIZIONE MODELLO
# ========================================
def build_model(lr=0.001):
    model = Sequential([
        LSTM(64, input_shape=(X_train.shape[1], X_train.shape[2])),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    optimizer = Adam(learning_rate=lr)
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

# ========================================
# 3) TUNING SEMPLICE
# ========================================
learning_rates = [0.001, 0.0005]
batch_sizes = [16, 32]

best_val_f1 = 0
best_model = None
best_lr = None
best_bs = None
best_threshold_final = 0.5

for lr in learning_rates:
    for bs in batch_sizes:
        print(f"Training with lr={lr}, batch_size={bs}")

        # Calcolo class weights per bilanciare dataset sbilanciato
        class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y_train), y=y_train)
        class_weights_dict = {i: class_weights[i] for i in range(len(class_weights))}

        model = build_model(lr)
        model.fit(X_train, y_train, epochs=20, batch_size=bs, validation_data=(X_val, y_val),
                  class_weight=class_weights_dict, verbose=0)

        # Predizioni probabilistiche su val set
        y_val_probs = model.predict(X_val).ravel()

        # Scegli soglia ottimale con curva Precision-Recall
```

```
        precisions, recalls, thresholds = precision_recall_curve(y_val, y_val_probs)
        f1_scores = 2 * (precisions * recalls) / (precisions + recalls + 1e-8)
        best_idx = np.argmax(f1_scores)
        best_threshold = thresholds[best_idx] if best_idx < len(thresholds) else 0.5
        best_f1_val = f1_scores[best_idx]

        print(f"Best threshold: {best_threshold:.3f}, Best F1-val: {best_f1_val:.3f}")

        if best_f1_val > best_val_f1:
            best_val_f1 = best_f1_val
            best_model = model
            best_lr = lr
            best_bs = bs
            best_threshold_final = best_threshold

print(f"\nBest tuning parameters: learning_rate={best_lr}, batch_size={best_bs}, threshold={best_threshold_final

# ========================================
# 4) VALUTAZIONE SU TEST SET
# ========================================
y_test_probs = best_model.predict(X_test).ravel()
y_test_pred = (y_test_probs >= best_threshold_final).astype(int)

print("\nConfusion matrix:")
print(confusion_matrix(y_test, y_test_pred))
print(f"Precision: {precision_score(y_test, y_test_pred):.3f}")
print(f"Recall: {recall_score(y_test, y_test_pred):.3f}")
print(f"F1-score: {f1_score(y_test, y_test_pred):.3f}")
print(f"ROC AUC: {roc_auc_score(y_test, y_test_probs):.3f}")

# ========================================
# 5) VISUALIZZAZIONE GRAFICI
# ========================================
precisions_test, recalls_test, thresholds_test = precision_recall_curve(y_test, y_test_probs)

plt.figure(figsize=(12, 4))

# Precision-Recall
plt.subplot(1, 3, 1)
plt.plot(recalls_test, precisions_test, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (Test)')

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_test_probs)
plt.subplot(1, 3, 2)
plt.plot(fpr, tpr, marker='.')
plt.plot([0, 1], [0, 1], '--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Test)')

# Confusion Matrix
cm = confusion_matrix(y_test, y_test_pred)
plt.subplot(1, 3, 3)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
ticks = np.arange(2)
plt.xticks(ticks, ['Neg', 'Pos'])
plt.yticks(ticks, ['Neg', 'Pos'])
thresh = cm.max() / 2.
for i, j in np.ndindex(cm.shape):
    plt.text(j, i, format(cm[i, j], 'd'),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")
plt.xlabel('Predicted label')
plt.ylabel('True label')

plt.tight_layout()
plt.show()
```

```
Training with lr=0.001, batch_size=16
5/5  0s 24ms/step
Best threshold: 0.396, Best F1-val: 0.673
Training with lr=0.001, batch_size=32
5/5  0s 24ms/step
Best threshold: 0.476, Best F1-val: 0.685
Training with lr=0.0005, batch_size=16
5/5  0s 24ms/step
Best threshold: 0.482, Best F1-val: 0.688
Training with lr=0.0005, batch_size=32
5/5  0s 24ms/step
Best threshold: 0.468, Best F1-val: 0.679

Best tuning parameters: learning_rate=0.0005, batch_size=16, threshold=0.48188358545303345
5/5  0s 4ms/step

Confusion matrix:
[[ 4 71]
 [ 2 73]]
Precision: 0.507
Recall: 0.973
F1-score: 0.667
ROC AUC: 0.536
```







Aggiustmaneti valori NAN e miglioramento con metriche RSE etc

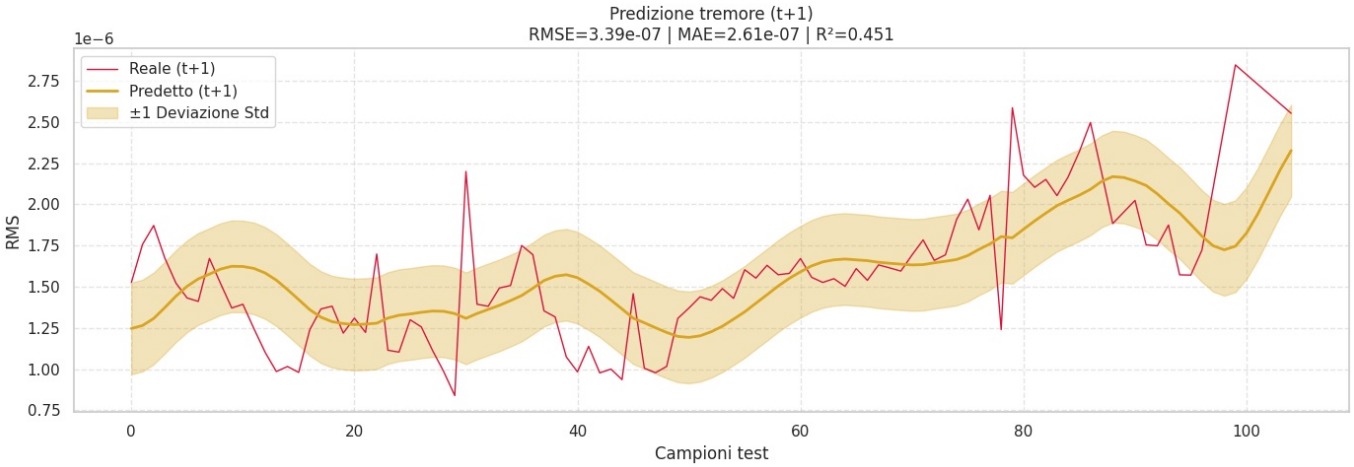
In [61]:

```python
time_step = 30
n_future = 5
filename = 'tremore vulcanico_cleaned.csv'
column = 'RMS'

# 1. CARICAMENTO E INTERPOLAZIONE
filename = 'tremore vulcanico_cleaned.csv'
column = 'RMS'

df = pd.read_csv(filename)

# Converte eventuali errori in NaN e interpola
df[column] = pd.to_numeric(df[column], errors='coerce')
df[column] = df[column].interpolate(method='linear')

# Elimina eventuali NaN residui ai bordi
df = df.dropna(subset=[column])


# 2. NORMALIZZA
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(df[[column]])

# 3. CREA SEQUENZE
def create_dataset(data, time_step=1, n_future=1):
    X, y = [], []
    for i in range(len(data) - time_step - n_future + 1):
        X.append(data[i:(i + time_step), 0])
        y.append(data[(i + time_step):(i + time_step + n_future), 0])
    return np.array(X), np.array(y)

X, y = create_dataset(data_scaled, time_step, n_future)

X = X.reshape(X.shape[0], X.shape[1], 1)  # formato richiesto da LSTM

# 4. SPLIT TRAIN / VAL / TEST
```

```python
train_size = int(len(X) * 0.7)
val_size = int(len(X) * 0.15)

X_train, y_train = X[:train_size], y[:train_size]
X_val, y_val = X[train_size:train_size+val_size], y[train_size:train_size+val_size]
X_test, y_test = X[train_size+val_size:], y[train_size+val_size:]

# 5. MODELLO LSTM
model = Sequential()
model.add(LSTM(64, return_sequences=True, input_shape=(time_step, 1)))
model.add(LSTM(32))
model.add(Dense(n_future))
model.compile(optimizer='adam', loss='mse')

callbacks = [
    EarlyStopping(patience=10, restore_best_weights=True),
    ModelCheckpoint("best_model.h5", save_best_only=True)
]

# 6. ALLENA
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=100,
                    batch_size=32,
                    callbacks=callbacks,
                    verbose=1)

# 7. PREVISIONE
y_pred = model.predict(X_test)

# 8. INVERSIONE SCALING ROBUSTA
y_test_inv = np.zeros_like(y_test)
y_pred_inv = np.zeros_like(y_pred)

for i in range(n_future):
    y_test_inv[:, i] = scaler.inverse_transform(y_test[:, i].reshape(-1, 1)).ravel()
    y_pred_inv[:, i] = scaler.inverse_transform(y_pred[:, i].reshape(-1, 1)).ravel()

# 9. METRICHE con filtro anti-NaN e salvataggio per plotting
step = 0  # t+1
y_true = y_test_inv[:, step]
y_hat = y_pred_inv[:, step]

# Filtro valori NaN
mask = ~np.isnan(y_true) & ~np.isnan(y_hat)
y_true_clean = y_true[mask]
y_hat_clean = y_hat[mask]

# Calcolo metriche
rmse = np.sqrt(mean_squared_error(y_true_clean, y_hat_clean))
mae = mean_absolute_error(y_true_clean, y_hat_clean)
r2 = r2_score(y_true_clean, y_hat_clean)

print(f"\n Step {step+1}:")
print("  RMSE:", rmse)
print("  MAE:", mae)
print("  R²:", r2)

# 10. GRAFICO (t+1) migliorato
plt.figure(figsize=(14, 5))
plt.plot(y_true_clean, label='Reale (t+1)', color='crimson', linewidth=1)
plt.plot(y_hat_clean, label='Predetto (t+1)', color='goldenrod', linewidth=2)

# Intervallo di confidenza (±1 std)
std_dev = np.std(y_hat_clean)
plt.fill_between(range(len(y_hat_clean)),
                 y_hat_clean - std_dev,
                 y_hat_clean + std_dev,
                 color='goldenrod',
                 alpha=0.3,
                 label='±1 Deviazione Std')

# Formattazione
plt.title(f"Predizione tremore (t+1)\nRMSE={rmse:.2e} | MAE={mae:.2e} | R²={r2:.3f}")
plt.xlabel("Campioni test")
plt.ylabel("RMS")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.5)
plt.ticklabel_format(style='sci', axis='y', scilimits=(0, 0))
plt.tight_layout()
plt.show()
```

```
Epoch 1/100
 9/16  0s 6ms/step - loss: 0.2503
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

16/16  2s 30ms/step - loss: 0.2110 - val_loss: 0.0534
Epoch 2/100
 9/16  0s 6ms/step - loss: 0.0363
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

16/16  0s 12ms/step - loss: 0.0337 - val_loss: 0.0277
Epoch 3/100
16/16  0s 10ms/step - loss: 0.0219 - val_loss: 0.0313
Epoch 4/100
10/16  0s 6ms/step - loss: 0.0212
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

16/16  0s 11ms/step - loss: 0.0205 - val_loss: 0.0263
Epoch 5/100
10/16  0s 6ms/step - loss: 0.0181
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

16/16  0s 11ms/step - loss: 0.0184 - val_loss: 0.0262
Epoch 6/100
 9/16  0s 6ms/step - loss: 0.0192
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

16/16  0s 12ms/step - loss: 0.0185 - val_loss: 0.0257
Epoch 7/100
16/16  0s 10ms/step - loss: 0.0176 - val_loss: 0.0280
Epoch 8/100
16/16  0s 10ms/step - loss: 0.0152 - val_loss: 0.0283
Epoch 9/100
16/16  0s 10ms/step - loss: 0.0164 - val_loss: 0.0257
Epoch 10/100
16/16  0s 10ms/step - loss: 0.0151 - val_loss: 0.0323
Epoch 11/100
16/16  0s 10ms/step - loss: 0.0150 - val_loss: 0.0279
Epoch 12/100
16/16  0s 10ms/step - loss: 0.0151 - val_loss: 0.0305
Epoch 13/100
16/16  0s 10ms/step - loss: 0.0133 - val_loss: 0.0293
Epoch 14/100
10/16  0s 6ms/step - loss: 0.0172
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. T
his file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_m
odel.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

16/16  0s 11ms/step - loss: 0.0164 - val_loss: 0.0254
Epoch 15/100
16/16  0s 10ms/step - loss: 0.0167 - val_loss: 0.0266
Epoch 16/100
16/16  0s 9ms/step - loss: 0.0132 - val_loss: 0.0265
Epoch 17/100
16/16  0s 10ms/step - loss: 0.0140 - val_loss: 0.0277
Epoch 18/100
16/16  0s 10ms/step - loss: 0.0132 - val_loss: 0.0284
Epoch 19/100
16/16  0s 10ms/step - loss: 0.0130 - val_loss: 0.0287
Epoch 20/100
16/16  0s 10ms/step - loss: 0.0132 - val_loss: 0.0280
Epoch 21/100
16/16  0s 10ms/step - loss: 0.0138 - val_loss: 0.0259
Epoch 22/100
16/16  0s 9ms/step - loss: 0.0143 - val_loss: 0.0263
Epoch 23/100
16/16  0s 9ms/step - loss: 0.0135 - val_loss: 0.0273
Epoch 24/100
16/16  0s 9ms/step - loss: 0.0117 - val_loss: 0.0257
4/4  0s 50ms/step

 Step 1:
  RMSE: 3.3908169632444305e-07
  MAE: 2.608847352648921e-07
  R²: 0.45065155737579665
```



Predizione terremoti futuri

Sulla base dei dati registrati c'è una % di possibilità che avvenga un terremoto

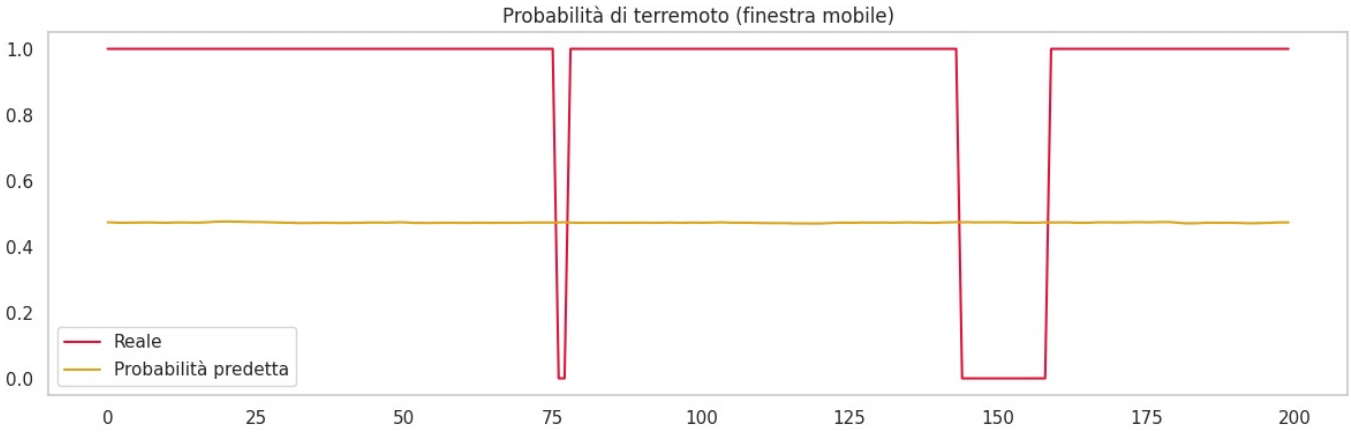
In [62]:
```
# Caricamento del file originale con solo eventi sismici
df_raw = pd.read_csv("/content/terremoti_puliti.csv")

# Parsing della colonna temporale
df_raw['Datetime'] = pd.to_datetime(df_raw['Datetime'])

# Creazione timeline regolare ogni 10 minuti
timeline = pd.date_range(start=df_raw['Datetime'].min(),
                         end=df_raw['Datetime'].max(),
                         freq='10min')

df = pd.DataFrame({'Datetime': timeline})

# Crea colonna 'TERREMOTO' = 1 se un terremoto è avvenuto nei successivi 6h
window = pd.Timedelta('6H')

df['TERREMOTO'] = df['Datetime'].apply(
    lambda t: int(((df_raw['Datetime'] > t) & (df_raw['Datetime'] <= t + window)).any())
)

# Genera feature simulate
np.random.seed(42)
df['RMS'] = np.random.normal(loc=1e-6, scale=3e-7, size=len(df))
df['FREQ_LOW'] = np.random.normal(loc=3.0, scale=0.5, size=len(df))
df['FREQ_HIGH'] = np.random.normal(loc=6.0, scale=1.0, size=len(df))
df['SO2'] = np.random.normal(loc=500, scale=100, size=len(df))
df['GPS'] = np.random.normal(loc=0.0, scale=1.0, size=len(df))

# Salva il file pronto per il codice
output_path = "dati_terremoti.csv"
df.to_csv(output_path, index=False)

output_path  # Per usarlo nel codice fornito dall'utente
```

Out[62]:
```
'dati_terremoti.csv'
```

In [63]:
```
# === PARAMETRI ===
time_step = 30
target_col = 'TERREMOTO'
features = ['RMS', 'FREQ_LOW', 'FREQ_HIGH', 'SO2', 'GPS']

# === CARICAMENTO DATI ===
df = pd.read_csv("dati_terremoti.csv")
df[features] = df[features].apply(pd.to_numeric, errors='coerce')
df = df.interpolate().dropna()

# === NORMALIZZAZIONE ===
scaler = MinMaxScaler()
df[features] = scaler.fit_transform(df[features])

# === CREAZIONE SEQUENZE ===
def create_sequences(df, features, target, time_step):
    X, y = [], []
    data = df[features].values
    labels = df[target].values
    for i in range(len(df) - time_step):
        X.append(data[i:i+time_step])
```

```
        y.append(labels[i+time_step])
    return np.array(X), np.array(y)

X, y = create_sequences(df, features, target_col, time_step)

# === SPLIT TRAIN/TEST ===
split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# === CLASS WEIGHTS ===
weights = compute_class_weight(class_weight='balanced', classes=np.unique(y_train), y=y_train)
class_weights = dict(zip(np.unique(y_train), weights))

# === MODELLO ===
model = Sequential([
    LSTM(64, input_shape=(X.shape[1], X.shape[2]), return_sequences=True),
    Dropout(0.4),
    LSTM(32),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# === ALLENAMENTO ===
callbacks = [EarlyStopping(patience=5, restore_best_weights=True)]
model.fit(
    X_train, y_train,
    epochs=30,
    batch_size=32,
    validation_split=0.2,
    callbacks=callbacks,
    class_weight=class_weights
)

# === VALUTAZIONE ===
y_pred = model.predict(X_test).flatten()
y_pred_class = (y_pred > 0.5).astype(int)

print("\n Classification Report:")
print(classification_report(y_test, y_pred_class))

print("\n Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_class))

print("\n AUC Score:")
print(roc_auc_score(y_test, y_pred))

# === GRAFICO ===
plt.figure(figsize=(12, 4))
plt.plot(y_test[:200], label='Reale', color='crimson')
plt.plot(y_pred[:200], label='Probabilità predetta', color='goldenrod')
plt.title("Probabilità di terremoto (finestra mobile)")
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```

```
Epoch 1/30
785/785  7s 7ms/step - accuracy: 0.6212 - loss: 0.6788 - val_accuracy: 0.4258 - val_loss: 0.7104
Epoch 2/30
785/785  5s 7ms/step - accuracy: 0.6379 - loss: 0.6781 - val_accuracy: 0.4258 - val_loss: 0.7029
Epoch 3/30
785/785  5s 7ms/step - accuracy: 0.6380 - loss: 0.6780 - val_accuracy: 0.4258 - val_loss: 0.7077
Epoch 4/30
785/785  5s 7ms/step - accuracy: 0.6417 - loss: 0.6762 - val_accuracy: 0.4258 - val_loss: 0.7078
Epoch 5/30
785/785  5s 7ms/step - accuracy: 0.6342 - loss: 0.6794 - val_accuracy: 0.4258 - val_loss: 0.7074
Epoch 6/30
785/785  5s 7ms/step - accuracy: 0.6361 - loss: 0.6786 - val_accuracy: 0.4258 - val_loss: 0.7079
Epoch 7/30
785/785  5s 7ms/step - accuracy: 0.6385 - loss: 0.6777 - val_accuracy: 0.4258 - val_loss: 0.7143
246/246  1s 2ms/step

 Classification Report:
              precision    recall  f1-score   support

           0       0.36      1.00      0.53      2833
           1       0.00      0.00      0.00      5014

    accuracy                           0.36      7847
   macro avg       0.18      0.50      0.27      7847
weighted avg       0.13      0.36      0.19      7847


 Confusion Matrix:
[[2833    0]
 [5014    0]]

 AUC Score:
0.4720584340549603
```



LSTM multivariata obiettivo predizione terremoti

In [64]:

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# === CONFIG ===
SEQUENCE_LENGTH = 96
FEATURES = ['ten_batt', 'temp_CR10', 'tilt_x_Avg', 'tilt_y_Avg',
            'temp_tilt', 'nord_tilt', 'barometro']
TARGET = 'quake_next24h'

# === 1. Caricamento ===
df = pd.read_csv("merged_etna_with_quake_target.csv")
print("Colonne trovate:", df.columns)

# === 2. Pulizia ===
df = df[FEATURES + [TARGET]].dropna()
print("Dopo dropna:", df.shape)

# === 3. Normalizzazione ===
scaler = MinMaxScaler()
df[FEATURES] = scaler.fit_transform(df[FEATURES])

# === 4. Trova una finestra con almeno 20 eventi positivi ===
```

```python
window_size = 10000
step = 5000
found = False

for start in range(0, len(df) - window_size, step):
    df_slice = df.iloc[start:start + window_size]
    num_quakes = df_slice[TARGET].sum()
    if num_quakes >= 20:
        print(f"Trovata finestra: {start}-{start+window_size}, eventi positivi: {num_quakes}")
        df_small = df_slice.copy()
        found = True
        break

if not found:
    raise ValueError("Nessuna finestra trovata con abbastanza eventi.")

# === 5. Funzione per creare sequenze ===
def create_sequences(df, features, target, seq_length):
    X, y = [], []
    for i in range(len(df) - seq_length):
        seq = df[features].iloc[i:i+seq_length].values
        label = df[target].iloc[i + seq_length]
        X.append(seq)
        y.append(label)
    return np.array(X), np.array(y)

# === 6. Creazione sequenze ===
X, y = create_sequences(df_small, FEATURES, TARGET, SEQUENCE_LENGTH)
print("X shape:", X.shape)
print("y shape:", y.shape)
print("Distribuzione y:", np.unique(y, return_counts=True))

# === 7. Train/Test split ===
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, shuffle=False)
y_train = y_train.astype(int)
y_val = y_val.astype(int)
print("Distribuzione y_train:", np.unique(y_train, return_counts=True))

# === 8. Class weights ===
if len(np.unique(y_train)) < 2:
    raise ValueError("I dati di training contengono solo una classe. Impossibile addestrare.")

neg, pos = np.bincount(y_train)
total = neg + pos
class_weight = {
    0: (1 / neg) * (total / 2.0),
    1: (1 / pos) * (total / 2.0)
}
print("Class weights:", class_weight)

# === 9. Modello LSTM ===
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(SEQUENCE_LENGTH, len(FEATURES))),
    Dropout(0.3),
    LSTM(32),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# === 10. Training ===
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=30,
    batch_size=64,
    class_weight=class_weight,
    callbacks=[early_stop],
    verbose=2
)

# === 11. Valutazione ===
y_pred = (model.predict(X_val) > 0.5).astype(int)
print("\n=== CLASSIFICATION REPORT ===")
print(classification_report(y_val, y_pred, digits=4))
```

```
Colonne trovate: Index(['Unnamed: 0', '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', 'quake_next24h'], dtype='object')
Dopo dropna: (399420, 8)
Trovata finestra: 205000-215000, eventi positivi: 2679
X shape: (9904, 96, 7)
y shape: (9904,)
Distribuzione y: (array([0, 1]), array([7225, 2679]))
Distribuzione y_train: (array([0, 1]), array([6781, 1142]))
Class weights: {0: np.float64(0.5842058693408052), 1: np.float64(3.4689141856392296)}
Epoch 1/30
124/124 - 3s - 27ms/step - accuracy: 0.7245 - loss: 0.4509 - val_accuracy: 0.7759 - val_loss: 0.5821
Epoch 2/30
124/124 - 1s - 10ms/step - accuracy: 0.7757 - loss: 0.3885 - val_accuracy: 0.7759 - val_loss: 0.5501
Epoch 3/30
124/124 - 1s - 10ms/step - accuracy: 0.7675 - loss: 0.3566 - val_accuracy: 0.7759 - val_loss: 0.5500
Epoch 4/30
124/124 - 1s - 10ms/step - accuracy: 0.7548 - loss: 0.3649 - val_accuracy: 0.7759 - val_loss: 0.5524
Epoch 5/30
124/124 - 1s - 10ms/step - accuracy: 0.5926 - loss: 0.5224 - val_accuracy: 0.7759 - val_loss: 0.6051
Epoch 6/30
124/124 - 1s - 10ms/step - accuracy: 0.5343 - loss: 0.6346 - val_accuracy: 0.7759 - val_loss: 0.5716
Epoch 7/30
124/124 - 1s - 10ms/step - accuracy: 0.4222 - loss: 0.7089 - val_accuracy: 0.7759 - val_loss: 0.6841
Epoch 8/30
124/124 - 1s - 10ms/step - accuracy: 0.5633 - loss: 0.6644 - val_accuracy: 0.7759 - val_loss: 0.6239
62/62  0s 6ms/step

=== CLASSIFICATION REPORT ===
              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000       444
           1     0.7759    1.0000    0.8738      1537

    accuracy                         0.7759      1981
   macro avg     0.3879    0.5000    0.4369      1981
weighted avg     0.6020    0.7759    0.6779      1981
```

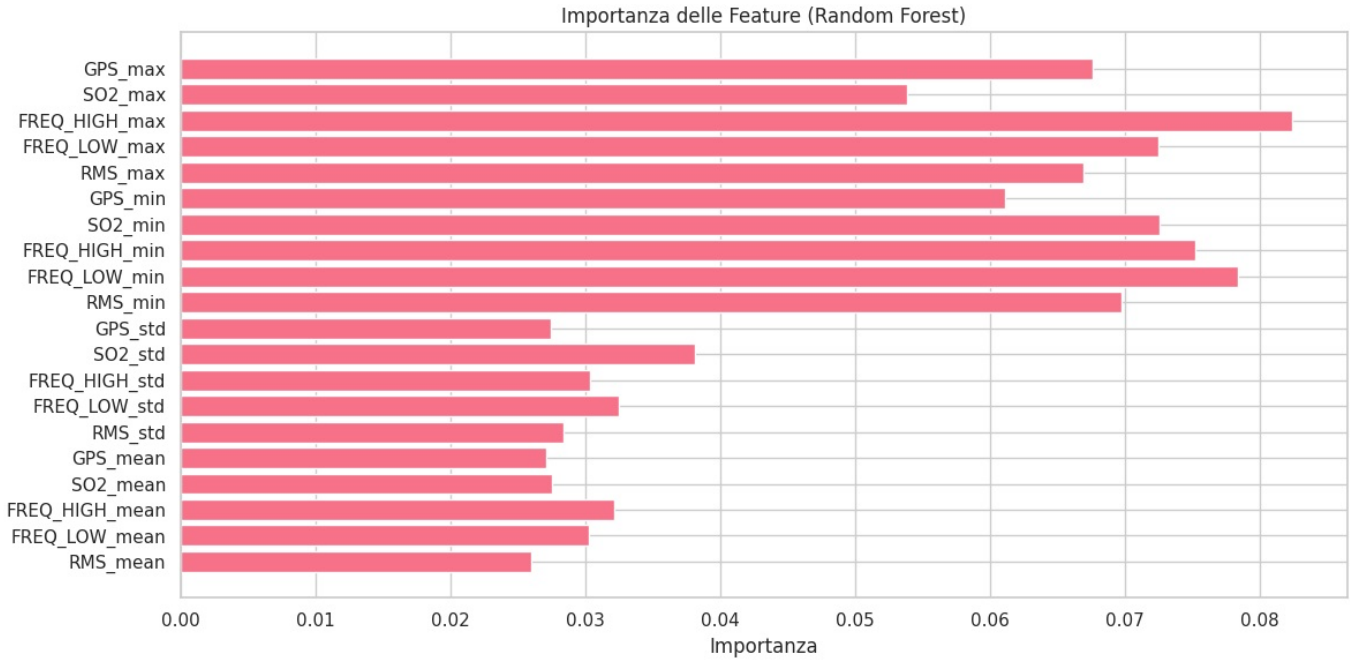
Random forrest

In [65]:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

# === PARAMETRI ===
time_step = 30
target_col = 'TERREMOTO'
features = ['RMS', 'FREQ_LOW', 'FREQ_HIGH', 'SO2', 'GPS']

# === CARICAMENTO ===
df = pd.read_csv("dati_terremoti.csv")

# Check colonne disponibili
print("Colonne trovate nel dataset:")
print(df.columns)

# Rimuove eventuali errori e interpolazioni
df[features] = df[features].apply(pd.to_numeric, errors='coerce')
df = df.interpolate().dropna()

if target_col not in df.columns:
    raise ValueError(f"La colonna target '{target_col}' non è presente nel DataFrame!")

# Normalizzazione
scaler = MinMaxScaler()
df[features] = scaler.fit_transform(df[features])

# === CREAZIONE SEQUENZE PER RANDOM FOREST (statistiche su finestre) ===
def create_feature_table(df, features, target, time_step):
    X, y = [], []
    for i in range(len(df) - time_step):
        if i % 1000 == 0:
            print(f"Processando finestra {i}/{len(df) - time_step}")
        window = df[features].iloc[i:i+time_step]
        stats = window.agg(['mean', 'std', 'min', 'max']).values.flatten()
```

```
        X.append(stats)
        y.append(df[target].iloc[i + time_step])
    return np.array(X), np.array(y)

X, y = create_feature_table(df, features, target_col, time_step)

print("Shape X:", X.shape)
print("Shape y:", y.shape)

# === SPLIT ===
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# === RANDOM FOREST ===
model = RandomForestClassifier(n_estimators=100, max_depth=10, class_weight='balanced', random_state=42)
model.fit(X_train, y_train)

# === VALUTAZIONE ===
y_pred = model.predict(X_test)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# === FEATURE IMPORTANCE ===
importances = model.feature_importances_
stat_names = ['mean', 'std', 'min', 'max']
columns = [f"{f}_{s}" for s in stat_names for f in features]

plt.figure(figsize=(12, 6))
plt.barh(columns, importances)
plt.title("Importanza delle Feature (Random Forest)")
plt.xlabel("Importanza")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Colonne trovate nel dataset:
Index(['Datetime', 'TERREMOTO', 'RMS', 'FREQ_LOW', 'FREQ_HIGH', 'SO2', 'GPS'], dtype='object')
Processando finestra 0/39234
Processando finestra 1000/39234
Processando finestra 2000/39234
Processando finestra 3000/39234
Processando finestra 4000/39234
Processando finestra 5000/39234
Processando finestra 6000/39234
Processando finestra 7000/39234
Processando finestra 8000/39234
Processando finestra 9000/39234
Processando finestra 10000/39234
Processando finestra 11000/39234
Processando finestra 12000/39234
Processando finestra 13000/39234
Processando finestra 14000/39234
Processando finestra 15000/39234
Processando finestra 16000/39234
Processando finestra 17000/39234
Processando finestra 18000/39234
Processando finestra 19000/39234
Processando finestra 20000/39234
Processando finestra 21000/39234
Processando finestra 22000/39234
Processando finestra 23000/39234
Processando finestra 24000/39234
Processando finestra 25000/39234
Processando finestra 26000/39234
Processando finestra 27000/39234
Processando finestra 28000/39234
Processando finestra 29000/39234
Processando finestra 30000/39234
Processando finestra 31000/39234
Processando finestra 32000/39234
Processando finestra 33000/39234
Processando finestra 34000/39234
Processando finestra 35000/39234
Processando finestra 36000/39234
Processando finestra 37000/39234
Processando finestra 38000/39234
Processando finestra 39000/39234
Shape X: (39234, 20)
Shape y: (39234,)

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.83      0.87      4306
           1       0.81      0.92      0.86      3541

    accuracy                           0.87      7847
   macro avg       0.87      0.87      0.87      7847
weighted avg       0.87      0.87      0.87      7847


Confusion Matrix:
[[3560  746]
 [ 287 3254]]
```



In [66]:

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Caricamento dei dati
data = pd.read_csv('dati_terremoti.csv')

# Conversione della colonna Datetime in formato datetime
data['Datetime'] = pd.to_datetime(data['Datetime'])

# Esplorazione dei dati
print("Anteprima dei dati:")
print(data.head())
print("\nInformazioni sul dataset:")
print(data.info())
print("\nStatistiche descrittive:")
print(data.describe())

# Analisi delle correlazioni
correlation_matrix = data.corr()
print("\nMatrice di correlazione:")
print(correlation_matrix['TERREMOTO'].sort_values(ascending=False))

# Preparazione dei dati per il modello
X = data[['RMS', 'FREQ_LOW', 'FREQ_HIGH', 'SO2', 'GPS']]
y = data['TERREMOTO']

# Divisione in training e test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Creazione e addestramento del modello
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Valutazione del modello
y_pred = model.predict(X_test)
print("\nRapporto di classificazione:")
print(classification_report(y_test, y_pred))
print("\nMatrice di confusione:")
print(confusion_matrix(y_test, y_pred))
print("\nAccuratezza del modello:", accuracy_score(y_test, y_pred))

# Feature importance
feature_importances = pd.Series(model.feature_importances_, index=X.columns).sort_values(ascending=False)
print("\nImportanza delle feature:")
print(feature_importances)

# Creazione di grafici
plt.figure(figsize=(15, 10))

# Grafico 1: Distribuzione dei terremoti nel tempo
plt.subplot(2, 2, 1)
data['Date'] = data['Datetime'].dt.date
earthquakes_by_date = data.groupby('Date')['TERREMOTO'].sum()
```

```
earthquakes_by_date.plot(kind='line', title='Terremoti per giorno', color='red')
plt.xlabel('Data')
plt.ylabel('Numero di terremoti')

# Grafico 2: Distribuzione delle feature per terremoti vs non terremoti
plt.subplot(2, 2, 2)
for feature in ['RMS', 'FREQ_LOW', 'FREQ_HIGH', 'SO2']:
    data.groupby('TERREMOTO')[feature].mean().plot(kind='bar', alpha=0.6, title=f'Media {feature} per classe')
    plt.xticks([0, 1], ['No Terremoto', 'Terremoto'], rotation=0)
    plt.ylabel(feature)
    break  # Mostriamo solo una feature per brevità

# Grafico 3: Feature importance
plt.subplot(2, 2, 3)
feature_importances.plot(kind='bar', title='Importanza delle feature')
plt.ylabel('Importanza')

# Grafico 4: Confronto tra RMS e SO2 per terremoti
plt.subplot(2, 2, 4)
plt.scatter(data[data['TERREMOTO']==0]['RMS'], data[data['TERREMOTO']==0]['SO2'],
            alpha=0.5, label='No Terremoto', color='blue')
plt.scatter(data[data['TERREMOTO']==1]['RMS'], data[data['TERREMOTO']==1]['SO2'],
            alpha=0.5, label='Terremoto', color='red')
plt.xlabel('RMS')
plt.ylabel('SO2')
plt.title('Confronto RMS vs SO2')
plt.legend()

plt.tight_layout()
plt.show()

# Analisi temporale dei terremoti
earthquake_data = data[data['TERREMOTO'] == 1]
print("\nOre con più terremoti:")
print(earthquake_data['Datetime'].dt.hour.value_counts().sort_index())

# Statistiche per i terremoti
print("\nStatistiche durante i terremoti:")
print(earthquake_data.describe())

# Statistiche per i non terremoti
non_earthquake_data = data[data['TERREMOTO'] == 0]
print("\nStatistiche durante i non terremoti:")
print(non_earthquake_data.describe())
```

Anteprima dei dati:

	Datetime	TERREMOTO	RMS	FREQ_LOW	FREQ_HIGH	SO2	GPS
0	2017-06-01 02:33:51.360	0	1.149014e-06	2.344481	6.999679	503.366290	-1.457163
1	2017-06-01 02:43:51.360	0	9.585207e-07	4.007211	7.580188	352.033959	0.444277
2	2017-06-01 02:53:51.360	0	1.194307e-06	2.901839	6.751767	450.899934	-0.325639
3	2017-06-01 03:03:51.360	0	1.456909e-06	3.123790	6.814825	702.040852	-1.262116
4	2017-06-01 03:13:51.360	0	9.297540e-07	3.455967	5.551815	353.411010	-0.175654

Informazioni sul dataset:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39264 entries, 0 to 39263
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Datetime   39264 non-null  datetime64[ns]
 1   TERREMOTO  39264 non-null  int64
 2   RMS        39264 non-null  float64
 3   FREQ_LOW   39264 non-null  float64
 4   FREQ_HIGH  39264 non-null  float64
 5   SO2        39264 non-null  float64
 6   GPS        39264 non-null  float64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 2.1 MB
None
```

Statistiche descrittive:

	Datetime	TERREMOTO	RMS	FREQ_LOW	FREQ_HIGH	SO2	GPS
count	39264	39264.000000	3.926400e+04	39264.000000	39264.000000	39264.000000	3.926400e+04
mean	2017-10-15 10:28:51.360000	0.451431	9.989548e-07	3.002343	6.006479	499.919982	-1.553443e-03
min	2017-06-01 02:33:51.360000	0.000000	-3.396812e-07	0.768516	2.217384	100.066775	-4.374043e+00
25%	2017-08-08 06:31:21.360000	0.000000	7.947420e-07	2.664608	5.338790	432.126026	-6.739732e-01
50%	2017-10-15 10:28:51.360000	0.000000	9.995148e-07	3.004698	6.006931	499.390228	-6.816180e-07
75%	2017-12-22 14:26:21.360000	1.000000	1.202409e-06	3.341545	6.676874	567.338966	6.718711e-01
max	2018-02-28 18:23:51.360000	1.000000	2.343725e-06	4.880078	10.202026	956.211472	3.782188e+00
std	NaN	0.497642	3.000384e-07	0.500895	0.999444	99.752031	9.999531e-01

Matrice di correlazione:

```
TERREMOTO    1.000000
Datetime     0.251254
SO2          0.001718
FREQ_HIGH    0.001509
RMS          0.000529
GPS         -0.000801
FREQ_LOW    -0.005513
Name: TERREMOTO, dtype: float64
```

Rapporto di classificazione:

	precision	recall	f1-score	support
0	0.55	0.69	0.61	6536
1	0.44	0.31	0.36	5244
accuracy			0.52	11780
macro avg	0.50	0.50	0.49	11780
weighted avg	0.50	0.52	0.50	11780

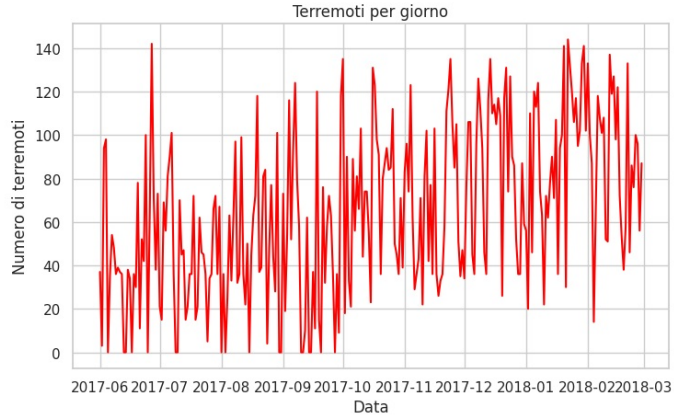
Matrice di confusione:

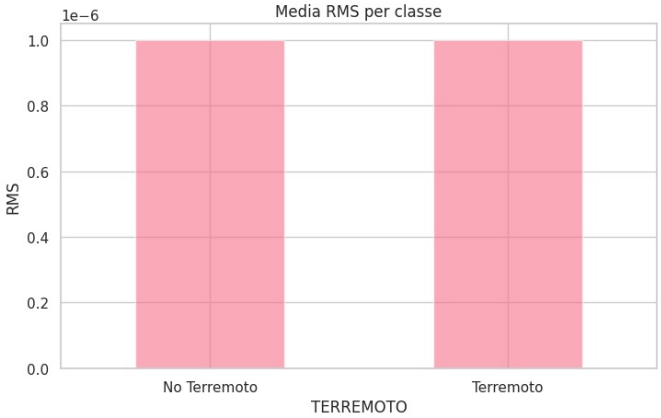
```
[[4513 2023]
 [3636 1608]]
```

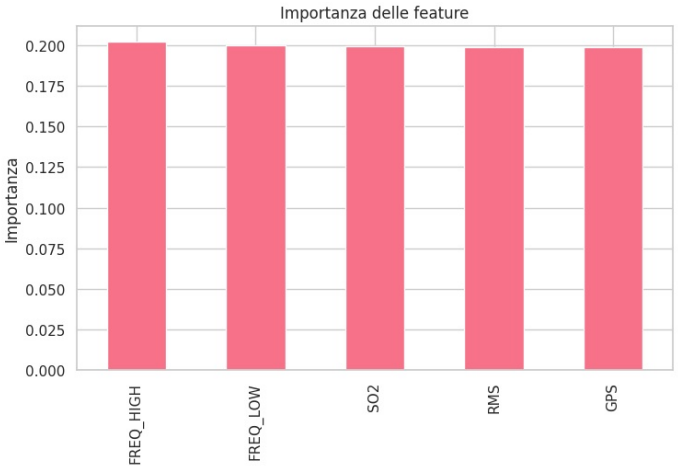
Accuratezza del modello: 0.5196095076400679

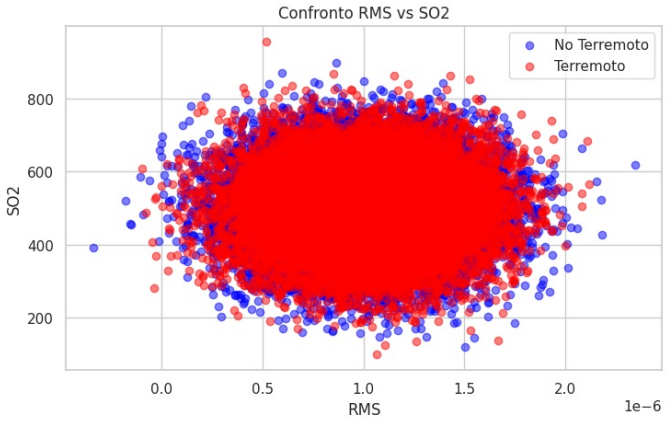
Importanza delle feature:

```
FREQ_HIGH    0.202194
FREQ_LOW     0.200138
SO2          0.199369
RMS          0.199160
GPS          0.199138
dtype: float64
```


Terremoti per giorno
Numero di terremoti
Data
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11
2017-12
2018-01
2018-02
2018-03


Media RMS per classe
1e−6
RMS
No Terremoto
Terremoto
TERREMOTO


Importanza delle feature
Importanza
FREQ_HIGH
FREQ_LOW
SO2
RMS
GPS


Confronto RMS vs SO2
No Terremoto
Terremoto
SO2
RMS
1e−6

```
Ore con più terremoti:
Datetime
0     807
1     792
2     766
3     710
4     710
5     722
6     666
7     652
8     675
9     703
10    674
11    668
12    691
13    694
14    709
15    729
16    724
17    732
18    777
19    806
20    844
21    835
22    824
23    815
Name: count, dtype: int64

Statistiche durante i terremoti:
                 Datetime             TERREMOTO       RMS        FREQ_LOW      FREQ_HIGH        SO2           GP
S
count                           17725   17725.0   1.772500e+04  17725.000000  17725.000000  17725.000000  17725.0
00000
mean   2017-11-06 05:42:11.027137024       1.0   9.991299e-07      2.999299      6.008141    500.108896     -0.0
02436
min       2017-06-01 04:13:51.360000       1.0  -9.652595e-08      0.768516      2.372792    100.066775     -4.3
74043
25%       2017-09-06 01:13:51.360000       1.0   7.942158e-07      2.656484      5.343193    432.245375     -0.6
75666
50%       2017-11-17 15:03:51.360000       1.0   9.988754e-07      3.001264      6.011569    500.150007      0.0
02350
75%       2018-01-12 11:03:51.360000       1.0   1.202446e-06      3.334407      6.670028    567.294733      0.6
72085
max       2018-02-28 18:23:51.360000       1.0   2.118350e-06      4.768024     10.202026    956.211472      3.4
64319
std                               NaN       0.0   3.003937e-07      0.501809      0.995055     99.855095      1.0
02060

Statistiche durante i non terremoti:
                 Datetime             TERREMOTO       RMS        FREQ_LOW      FREQ_HIGH        SO2           GP
S
count                           21539   21539.0   2.153900e+04  21539.000000  21539.000000  21539.000000  21539.0
00000
mean   2017-09-27 11:54:28.492643328       0.0   9.988107e-07      3.004848      6.005111    499.764519     -0.0
00827
min       2017-06-01 02:33:51.360000       0.0  -3.396812e-07      0.852305      2.217384    120.536256     -3.5
99061
25%       2017-07-24 21:38:51.360000       0.0   7.952081e-07      2.671241      5.334885    432.076094     -0.6
72667
50%       2017-09-17 19:53:51.360000       0.0   1.000227e-06      3.007201      6.002892    498.910876     -0.0
01569
75%       2017-11-26 16:18:51.360000       0.0   1.202336e-06      3.346799      6.681874    567.359560      0.6
71299
max       2018-02-28 09:43:51.360000       0.0   2.343725e-06      4.880078      9.766234    897.686819      3.7
82188
std                               NaN       0.0   2.997526e-07      0.500139      1.003063     99.669188      0.9
98239
```

In [67]:
```
print(df.columns.tolist())
```

```
['Datetime', 'TERREMOTO', 'RMS', 'FREQ_LOW', 'FREQ_HIGH', 'SO2', 'GPS']
```

CONVERSIONE IN PDF

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

In [68]:
```
%cd "/content/drive/MyDrive"
!pip install "nbconvert[webpdf]"
!jupyter nbconvert --to webpdf "ETNA2018_UFFICIAL (3).ipynb" --allow-chromium-download
from google.colab import files
files.download("ETNA2018_UFFICIAL (3).pdf")
```

```
/content/drive/MyDrive
Requirement already satisfied: nbconvert[webpdf] in /usr/local/lib/python3.12/dist-packages (7.16.6)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dist-packages (from nbconvert[webpdf]
) (4.13.4)
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.12/dist-packages (from bleach[css]!=5.0.0
->nbconvert[webpdf]) (6.2.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.12/dist-packages (from nbconvert[webpdf]) (0
.7.1)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.12/dist-packages (from nbconvert[webpdf]) (
3.1.6)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.12/dist-packages (from nbconvert[webp
df]) (5.8.1)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.12/dist-packages (from nbconvert[we
bpdf]) (0.3.0)
Requirement already satisfied: markupsafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from nbconvert[webpdf
]) (3.0.2)
Requirement already satisfied: mistune<4,>=2.0.3 in /usr/local/lib/python3.12/dist-packages (from nbconvert[webp
df]) (3.1.3)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.12/dist-packages (from nbconvert[webpdf
]) (0.10.2)
Requirement already satisfied: nbformat>=5.7 in /usr/local/lib/python3.12/dist-packages (from nbconvert[webpdf])
(5.10.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from nbconvert[webpdf]) (25
.0)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.12/dist-packages (from nbconvert[w
ebpdf]) (1.5.1)
Requirement already satisfied: pygments>=2.4.1 in /usr/local/lib/python3.12/dist-packages (from nbconvert[webpdf
]) (2.19.2)
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.12/dist-packages (from nbconvert[webpdf]
) (5.7.1)
Collecting playwright (from nbconvert[webpdf])
  Downloading playwright-1.54.0-py3-none-manylinux1_x86_64.whl.metadata (3.5 kB)
Requirement already satisfied: webencodings in /usr/local/lib/python3.12/dist-packages (from bleach!=5.0.0->blea
ch[css]!=5.0.0->nbconvert[webpdf]) (0.5.1)
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from bleach[css]
!=5.0.0->nbconvert[webpdf]) (1.4.0)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.12/dist-packages (from jupyter-core>=
4.7->nbconvert[webpdf]) (4.3.8)
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.12/dist-packages (from nbclient>
=0.5.0->nbconvert[webpdf]) (6.1.12)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.12/dist-packages (from nbformat>=5
.7->nbconvert[webpdf]) (2.21.2)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.12/dist-packages (from nbformat>=5.7->n
bconvert[webpdf]) (4.25.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4->nb
convert[webpdf]) (2.7)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.12/dist-packages (from beautif
ulsoup4->nbconvert[webpdf]) (4.14.1)
Collecting pyee<14,>=13 (from playwright->nbconvert[webpdf])
  Downloading pyee-13.0.0-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: greenlet<4.0.0,>=3.1.1 in /usr/local/lib/python3.12/dist-packages (from playwrigh
t->nbconvert[webpdf]) (3.2.4)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.12/dist-packages (from jsonschema>=2.6->n
bformat>=5.7->nbconvert[webpdf]) (25.3.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.12/dist-packages (
from jsonschema>=2.6->nbformat>=5.7->nbconvert[webpdf]) (2025.4.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.12/dist-packages (from jsonschema>=
2.6->nbformat>=5.7->nbconvert[webpdf]) (0.36.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.12/dist-packages (from jsonschema>=2.6->
nbformat>=5.7->nbconvert[webpdf]) (0.27.0)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.12/dist-packages (from jupyter-client>=6.1.12
->nbclient>=0.5.0->nbconvert[webpdf]) (26.2.1)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.12/dist-packages (from jupyter-cli
ent>=6.1.12->nbclient>=0.5.0->nbconvert[webpdf]) (2.9.0.post0)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.12/dist-packages (from jupyter-client>=6.1
.12->nbclient>=0.5.0->nbconvert[webpdf]) (6.4.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.1->j
upyter-client>=6.1.12->nbclient>=0.5.0->nbconvert[webpdf]) (1.17.0)
Downloading playwright-1.54.0-py3-none-manylinux1_x86_64.whl (45.9 MB)
    45.9/45.9 MB 43.7 MB/s eta 0:00:00
Downloading pyee-13.0.0-py3-none-any.whl (15 kB)
Installing collected packages: pyee, playwright
Successfully installed playwright-1.54.0 pyee-13.0.0
[NbConvertApp] Converting notebook ETNA2018_UFFICIAL (3).ipynb to webpdf
Traceback (most recent call last):
  File "/usr/local/bin/jupyter-nbconvert", line 10, in <module>
    sys.exit(main())
             ^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/jupyter_core/application.py", line 284, in launch_instance
    super().launch_instance(argv=argv, **kwargs)
  File "/usr/local/lib/python3.12/dist-packages/traitlets/config/application.py", line 992, in launch_instance
    app.start()
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/nbconvertapp.py", line 420, in start
```

```
    self.convert_notebooks()
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/nbconvertapp.py", line 597, in convert_notebooks
    self.convert_single_notebook(notebook_filename)
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/nbconvertapp.py", line 563, in convert_single_notebook
    output, resources = self.export_single_notebook(
                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/nbconvertapp.py", line 487, in export_single_notebook
    output, resources = self.exporter.from_filename(
                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/exporters/templateexporter.py", line 390, in from_file
name
    return super().from_filename(filename, resources, **kw)  # type:ignore[return-value]
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/exporters/exporter.py", line 201, in from_filename
    return self.from_file(f, resources=resources, **kw)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/exporters/templateexporter.py", line 396, in from_file
    return super().from_file(file_stream, resources, **kw)  # type:ignore[return-value]
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/exporters/exporter.py", line 220, in from_file
    return self.from_notebook_node(
           ^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/exporters/webpdf.py", line 171, in from_notebook_node
    html, resources = super().from_notebook_node(nb, resources=resources, **kw)
                      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/exporters/html.py", line 278, in from_notebook_node
    html, resources = super().from_notebook_node(nb, resources, **kw)
                      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/exporters/templateexporter.py", line 429, in from_note
book_node
    output = self.template.render(nb=nb_copy, resources=resources)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/jinja2/environment.py", line 1295, in render
    self.environment.handle_exception()
  File "/usr/local/lib/python3.12/dist-packages/jinja2/environment.py", line 942, in handle_exception
    raise rewrite_traceback_stack(source=source)
  File "/usr/local/share/jupyter/nbconvert/templates/lab/index.html.j2", line 4, in top-level template code
    {% from 'jupyter_widgets.html.j2' import jupyter_widgets %}
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/lab/base.html.j2", line 3, in top-level template code
    {% from 'cell_id_anchor.j2' import cell_id_anchor %}
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/base/display_priority.j2", line 1, in top-level template co
de
    {%- extends 'base/null.j2' -%}
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/base/null.j2", line 26, in top-level template code
    {%- block body -%}
    ^^^^^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/base/null.j2", line 29, in block 'body'
    {%- block body_loop -%}
^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/base/null.j2", line 31, in block 'body_loop'
    {%- block any_cell scoped -%}
^^^^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/base/null.j2", line 34, in block 'any_cell'
    {%- block codecell scoped -%}
^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/lab/base.html.j2", line 13, in block 'codecell'
    {{ super() }}
    ^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/base/null.j2", line 44, in block 'codecell'
    {%- block output_group -%}
^
  File "/usr/local/share/jupyter/nbconvert/templates/lab/base.html.j2", line 39, in block 'output_group'
    {{ super() }}
    ^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/base/null.j2", line 48, in block 'output_group'
    {%- block outputs scoped -%}

  File "/usr/local/share/jupyter/nbconvert/templates/lab/base.html.j2", line 45, in block 'outputs'
    {{ super() }}
    ^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/base/null.j2", line 50, in block 'outputs'
    {%- block output scoped -%}

  File "/usr/local/share/jupyter/nbconvert/templates/lab/base.html.j2", line 92, in block 'output'
    {{ super() }}
    ^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/base/null.j2", line 67, in block 'output'
    {%- block display_data scoped -%}

  File "/usr/local/share/jupyter/nbconvert/templates/base/null.j2", line 68, in block 'display_data'
```

```
    {%- block data_priority scoped -%}

  File "/usr/local/share/jupyter/nbconvert/templates/lab/base.html.j2", line 131, in block 'data_priority'
    {{ super() }}
    ^^^^^^^^^^^^^^
  File "/usr/local/share/jupyter/nbconvert/templates/base/display_priority.j2", line 7, in block 'data_priority'
    {%- for type in output.data | filter_data_type -%}
^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/nbconvert/filters/widgetsdatatypefilter.py", line 58, in __call_
_
    metadata["widgets"][WIDGET_STATE_MIMETYPE]["state"]
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^
KeyError: 'state'
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipython-input-556377299.py in <cell line: 0>()
      3 get_ipython().system('jupyter nbconvert --to webpdf "ETNA2018_UFFICIAL (3).ipynb" --allow-chromium-downl
oad')
      4 from google.colab import files
----> 5 files.download("ETNA2018_UFFICIAL (3).pdf")
      6

/usr/local/lib/python3.12/dist-packages/google/colab/files.py in download(filename)
    231   if not _os.path.exists(filename):
    232     msg = 'Cannot find file: {}'.format(filename)
--> 233     raise FileNotFoundError(msg)  # pylint: disable=undefined-variable
    234
    235   comm_manager = _IPython.get_ipython().kernel.comm_manager

FileNotFoundError: Cannot find file: ETNA2018_UFFICIAL (3).pdf
```

In [69]:
```
!pip install pdfkit
!apt-get install -y wkhtmltopdf
import pdfkit
config = pdfkit.configuration(wkhtmltopdf="/usr/bin/wkhtmltopdf")
```

```
Collecting pdfkit
  Downloading pdfkit-1.0.0-py3-none-any.whl.metadata (9.3 kB)
Downloading pdfkit-1.0.0-py3-none-any.whl (12 kB)
Installing collected packages: pdfkit
Successfully installed pdfkit-1.0.0
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  avahi-daemon geoclue-2.0 glib-networking glib-networking-common
  glib-networking-services gsettings-desktop-schemas iio-sensor-proxy
  libavahi-core7 libavahi-glib1 libdaemon0 libevdev2 libgudev-1.0-0 libhyphen0
  libinput-bin libinput10 libjson-glib-1.0-0 libjson-glib-1.0-common
  libmbim-glib4 libmbim-proxy libmd4c0 libmm-glib0 libmtdev1 libnl-genl-3-200
  libnotify4 libnss-mdns libproxy1v5 libqmi-glib5 libqmi-proxy libqt5core5a
  libqt5dbus5 libqt5gui5 libqt5network5 libqt5positioning5 libqt5printsupport5
  libqt5qml5 libqt5qmlmodels5 libqt5quick5 libqt5sensors5 libqt5svg5
  libqt5webchannel5 libqt5webkit5 libqt5widgets5 libsoup2.4-1
  libsoup2.4-common libudev1 libwacom-bin libwacom-common libwacom9 libwoff1
  libxcb-icccm4 libxcb-image0 libxcb-keysyms1 libxcb-render-util0 libxcb-util1
  libxcb-xinerama0 libxcb-xinput0 libxcb-xkb1 libxkbcommon-x11-0 modemmanager
  qt5-gtk-platformtheme qttranslations5-l10n session-migration
  systemd-hwe-hwdb udev usb-modeswitch usb-modeswitch-data wpasupplicant
Suggested packages:
  avahi-autoipd gnome-shell | notification-daemon avahi-autoipd | zeroconf
  qt5-image-formats-plugins qtwayland5 qt5-qmltooling-plugins comgt wvdial
  wpagui libengine-pkcs11-openssl
The following NEW packages will be installed:
  avahi-daemon geoclue-2.0 glib-networking glib-networking-common
  glib-networking-services gsettings-desktop-schemas iio-sensor-proxy
  libavahi-core7 libavahi-glib1 libdaemon0 libevdev2 libgudev-1.0-0 libhyphen0
  libinput-bin libinput10 libjson-glib-1.0-0 libjson-glib-1.0-common
  libmbim-glib4 libmbim-proxy libmd4c0 libmm-glib0 libmtdev1 libnl-genl-3-200
  libnotify4 libnss-mdns libproxy1v5 libqmi-glib5 libqmi-proxy libqt5core5a
  libqt5dbus5 libqt5gui5 libqt5network5 libqt5positioning5 libqt5printsupport5
  libqt5qml5 libqt5qmlmodels5 libqt5quick5 libqt5sensors5 libqt5svg5
  libqt5webchannel5 libqt5webkit5 libqt5widgets5 libsoup2.4-1
  libsoup2.4-common libwacom-bin libwacom-common libwacom9 libwoff1
  libxcb-icccm4 libxcb-image0 libxcb-keysyms1 libxcb-render-util0 libxcb-util1
  libxcb-xinerama0 libxcb-xinput0 libxcb-xkb1 libxkbcommon-x11-0 modemmanager
  qt5-gtk-platformtheme qttranslations5-l10n session-migration
  systemd-hwe-hwdb udev usb-modeswitch usb-modeswitch-data wkhtmltopdf
  wpasupplicant
The following packages will be upgraded:
  libudev1
1 upgraded, 67 newly installed, 0 to remove and 34 not upgraded.
Need to get 35.5 MB of archives.
After this operation, 141 MB of additional disk space will be used.
```

```
Get:1 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libavahi-core7 amd64 0.8-5ubuntu5.2 [90.8 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/main amd64 libdaemon0 amd64 0.14-7.1ubuntu3 [14.1 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 avahi-daemon amd64 0.8-5ubuntu5.2 [69.7 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 libqt5core5a amd64 5.15.3+dfsg-2ubuntu0.2 [2
,006 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy/main amd64 libevdev2 amd64 1.12.1+dfsg-1 [39.5 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/main amd64 libmtdev1 amd64 1.1.6-1build4 [14.5 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libudev1 amd64 249.11-0ubuntu3.16 [76.7 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy/main amd64 libgudev-1.0-0 amd64 1:237-2build1 [16.3 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy/main amd64 libwacom-common all 2.2.0-1 [54.3 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy/main amd64 libwacom9 amd64 2.2.0-1 [22.0 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libinput-bin amd64 1.20.0-1ubuntu0.3 [19.9 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libinput10 amd64 1.20.0-1ubuntu0.3 [131 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libmd4c0 amd64 0.4.8-1 [42.0 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 libqt5dbus5 amd64 5.15.3+dfsg-2ubuntu0.2 [2
22 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 libqt5network5 amd64 5.15.3+dfsg-2ubuntu0.2
[731 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxcb-icccm4 amd64 0.4.1-1.1build2 [11.5 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxcb-util1 amd64 0.4.0-1build2 [11.4 kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxcb-image0 amd64 0.4.0-2 [11.5 kB]
Get:19 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxcb-keysyms1 amd64 0.4.0-1build3 [8,746 B]
Get:20 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxcb-render-util0 amd64 0.3.9-1build3 [10.3 kB]
Get:21 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxcb-xinerama0 amd64 1.14-3ubuntu3 [5,414 B]
Get:22 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxcb-xinput0 amd64 1.14-3ubuntu3 [34.3 kB]
Get:23 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxcb-xkb1 amd64 1.14-3ubuntu3 [32.8 kB]
Get:24 http://archive.ubuntu.com/ubuntu jammy/main amd64 libxkbcommon-x11-0 amd64 1.4.0-1 [14.4 kB]
Get:25 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 libqt5gui5 amd64 5.15.3+dfsg-2ubuntu0.2 [3,
722 kB]
Get:26 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 libqt5widgets5 amd64 5.15.3+dfsg-2ubuntu0.2
[2,561 kB]
Get:27 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libqt5svg5 amd64 5.15.3-1 [149 kB]
Get:28 http://archive.ubuntu.com/ubuntu jammy/main amd64 libhyphen0 amd64 2.8.8-7build2 [28.2 kB]
Get:29 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libqt5positioning5 amd64 5.15.3+dfsg-3 [223 kB]
Get:30 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 libqt5printsupport5 amd64 5.15.3+dfsg-2ubun
tu0.2 [214 kB]
Get:31 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libqt5qml5 amd64 5.15.3+dfsg-1 [1,472 kB]
Get:32 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libqt5qmlmodels5 amd64 5.15.3+dfsg-1 [205 kB]
Get:33 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libqt5quick5 amd64 5.15.3+dfsg-1 [1,748 kB]
Get:34 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libqt5sensors5 amd64 5.15.3-1 [123 kB]
Get:35 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libqt5webchannel5 amd64 5.15.3-1 [62.9 kB]
Get:36 http://archive.ubuntu.com/ubuntu jammy/main amd64 libwoff1 amd64 1.0.2-1build4 [45.2 kB]
Get:37 http://archive.ubuntu.com/ubuntu jammy/universe amd64 libqt5webkit5 amd64 5.212.0~alpha4-15ubuntu1 [12.8
MB]
Get:38 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 udev amd64 249.11-0ubuntu3.16 [1,557 kB]
Get:39 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libavahi-glib1 amd64 0.8-5ubuntu5.2 [8,296 B]
Get:40 http://archive.ubuntu.com/ubuntu jammy/main amd64 libjson-glib-1.0-common all 1.6.6-1build1 [4,432 B]
Get:41 http://archive.ubuntu.com/ubuntu jammy/main amd64 libjson-glib-1.0-0 amd64 1.6.6-1build1 [69.9 kB]
Get:42 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libmm-glib0 amd64 1.20.0-1~ubuntu22.04.4 [262 k
B]
Get:43 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libnotify4 amd64 0.7.9-3ubuntu5.22.04.1 [20.3 k
B]
Get:44 http://archive.ubuntu.com/ubuntu jammy/main amd64 libproxy1v5 amd64 0.4.17-2 [51.9 kB]
Get:45 http://archive.ubuntu.com/ubuntu jammy/main amd64 glib-networking-common all 2.72.0-1 [3,718 B]
Get:46 http://archive.ubuntu.com/ubuntu jammy/main amd64 glib-networking-services amd64 2.72.0-1 [9,982 B]
Get:47 http://archive.ubuntu.com/ubuntu jammy/main amd64 session-migration amd64 0.3.6 [9,774 B]
Get:48 http://archive.ubuntu.com/ubuntu jammy/main amd64 gsettings-desktop-schemas all 42.0-1ubuntu1 [31.1 kB]
Get:49 http://archive.ubuntu.com/ubuntu jammy/main amd64 glib-networking amd64 2.72.0-1 [69.8 kB]
Get:50 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libsoup2.4-common all 2.74.2-3ubuntu0.6 [4,778
B]
Get:51 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libsoup2.4-1 amd64 2.74.2-3ubuntu0.6 [288 kB]
Get:52 http://archive.ubuntu.com/ubuntu jammy/main amd64 geoclue-2.0 amd64 2.5.7-3ubuntu3 [111 kB]
Get:53 http://archive.ubuntu.com/ubuntu jammy/main amd64 iio-sensor-proxy amd64 3.3-0ubuntu6 [34.4 kB]
Get:54 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libmbim-glib4 amd64 1.28.0-1~ubuntu20.04.2 [192
kB]
Get:55 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libmbim-proxy amd64 1.28.0-1~ubuntu20.04.2 [6,1
60 B]
Get:56 http://archive.ubuntu.com/ubuntu jammy/main amd64 libnl-genl-3-200 amd64 3.5.0-0.1 [12.4 kB]
Get:57 http://archive.ubuntu.com/ubuntu jammy/main amd64 libnss-mdns amd64 0.15.1-1ubuntu1 [27.0 kB]
Get:58 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libqmi-glib5 amd64 1.32.0-1ubuntu0.22.04.1 [772
kB]
Get:59 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 libqmi-proxy amd64 1.32.0-1ubuntu0.22.04.1 [6,0
72 B]
Get:60 http://archive.ubuntu.com/ubuntu jammy/main amd64 libwacom-bin amd64 2.2.0-1 [13.6 kB]
Get:61 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 modemmanager amd64 1.20.0-1~ubuntu22.04.4 [1,09
4 kB]
Get:62 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 qt5-gtk-platformtheme amd64 5.15.3+dfsg-2ub
untu0.2 [130 kB]
Get:63 http://archive.ubuntu.com/ubuntu jammy/universe amd64 qttranslations5-l10n all 5.15.3-1 [1,983 kB]
Get:64 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 systemd-hwe-hwdb all 249.11.5 [3,228 B]
Get:65 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 wpasupplicant amd64 2:2.10-6ubuntu2.2 [1,482 kB
]
Get:66 http://archive.ubuntu.com/ubuntu jammy/main amd64 usb-modeswitch-data all 20191128-4 [33.2 kB]
```

```
Get:67 http://archive.ubuntu.com/ubuntu jammy/main amd64 usb-modeswitch amd64 2.6.1-3ubuntu2 [46.0 kB]
Get:68 http://archive.ubuntu.com/ubuntu jammy/universe amd64 wkhtmltopdf amd64 0.12.6-2 [173 kB]
Fetched 35.5 MB in 5s (6,699 kB/s)
Extracting templates from packages: 100%
Selecting previously unselected package libavahi-core7:amd64.
(Reading database ... 126371 files and directories currently installed.)
Preparing to unpack .../0-libavahi-core7_0.8-5ubuntu5.2_amd64.deb ...
Unpacking libavahi-core7:amd64 (0.8-5ubuntu5.2) ...
Selecting previously unselected package libdaemon0:amd64.
Preparing to unpack .../1-libdaemon0_0.14-7.1ubuntu3_amd64.deb ...
Unpacking libdaemon0:amd64 (0.14-7.1ubuntu3) ...
Selecting previously unselected package avahi-daemon.
Preparing to unpack .../2-avahi-daemon_0.8-5ubuntu5.2_amd64.deb ...
Unpacking avahi-daemon (0.8-5ubuntu5.2) ...
Selecting previously unselected package libqt5core5a:amd64.
Preparing to unpack .../3-libqt5core5a_5.15.3+dfsg-2ubuntu0.2_amd64.deb ...
Unpacking libqt5core5a:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Selecting previously unselected package libevdev2:amd64.
Preparing to unpack .../4-libevdev2_1.12.1+dfsg-1_amd64.deb ...
Unpacking libevdev2:amd64 (1.12.1+dfsg-1) ...
Selecting previously unselected package libmtdev1:amd64.
Preparing to unpack .../5-libmtdev1_1.1.6-1build4_amd64.deb ...
Unpacking libmtdev1:amd64 (1.1.6-1build4) ...
Preparing to unpack .../6-libudev1_249.11-0ubuntu3.16_amd64.deb ...
Unpacking libudev1:amd64 (249.11-0ubuntu3.16) over (249.11-0ubuntu3.12) ...
Setting up libudev1:amd64 (249.11-0ubuntu3.16) ...
Selecting previously unselected package libgudev-1.0-0:amd64.
(Reading database ... 126440 files and directories currently installed.)
Preparing to unpack .../00-libgudev-1.0-0_1%3a237-2build1_amd64.deb ...
Unpacking libgudev-1.0-0:amd64 (1:237-2build1) ...
Selecting previously unselected package libwacom-common.
Preparing to unpack .../01-libwacom-common_2.2.0-1_all.deb ...
Unpacking libwacom-common (2.2.0-1) ...
Selecting previously unselected package libwacom9:amd64.
Preparing to unpack .../02-libwacom9_2.2.0-1_amd64.deb ...
Unpacking libwacom9:amd64 (2.2.0-1) ...
Selecting previously unselected package libinput-bin.
Preparing to unpack .../03-libinput-bin_1.20.0-1ubuntu0.3_amd64.deb ...
Unpacking libinput-bin (1.20.0-1ubuntu0.3) ...
Selecting previously unselected package libinput10:amd64.
Preparing to unpack .../04-libinput10_1.20.0-1ubuntu0.3_amd64.deb ...
Unpacking libinput10:amd64 (1.20.0-1ubuntu0.3) ...
Selecting previously unselected package libmd4c0:amd64.
Preparing to unpack .../05-libmd4c0_0.4.8-1_amd64.deb ...
Unpacking libmd4c0:amd64 (0.4.8-1) ...
Selecting previously unselected package libqt5dbus5:amd64.
Preparing to unpack .../06-libqt5dbus5_5.15.3+dfsg-2ubuntu0.2_amd64.deb ...
Unpacking libqt5dbus5:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Selecting previously unselected package libqt5network5:amd64.
Preparing to unpack .../07-libqt5network5_5.15.3+dfsg-2ubuntu0.2_amd64.deb ...
Unpacking libqt5network5:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Selecting previously unselected package libxcb-icccm4:amd64.
Preparing to unpack .../08-libxcb-icccm4_0.4.1-1.1build2_amd64.deb ...
Unpacking libxcb-icccm4:amd64 (0.4.1-1.1build2) ...
Selecting previously unselected package libxcb-util1:amd64.
Preparing to unpack .../09-libxcb-util1_0.4.0-1build2_amd64.deb ...
Unpacking libxcb-util1:amd64 (0.4.0-1build2) ...
Selecting previously unselected package libxcb-image0:amd64.
Preparing to unpack .../10-libxcb-image0_0.4.0-2_amd64.deb ...
Unpacking libxcb-image0:amd64 (0.4.0-2) ...
Selecting previously unselected package libxcb-keysyms1:amd64.
Preparing to unpack .../11-libxcb-keysyms1_0.4.0-1build3_amd64.deb ...
Unpacking libxcb-keysyms1:amd64 (0.4.0-1build3) ...
Selecting previously unselected package libxcb-render-util0:amd64.
Preparing to unpack .../12-libxcb-render-util0_0.3.9-1build3_amd64.deb ...
Unpacking libxcb-render-util0:amd64 (0.3.9-1build3) ...
Selecting previously unselected package libxcb-xinerama0:amd64.
Preparing to unpack .../13-libxcb-xinerama0_1.14-3ubuntu3_amd64.deb ...
Unpacking libxcb-xinerama0:amd64 (1.14-3ubuntu3) ...
Selecting previously unselected package libxcb-xinput0:amd64.
Preparing to unpack .../14-libxcb-xinput0_1.14-3ubuntu3_amd64.deb ...
Unpacking libxcb-xinput0:amd64 (1.14-3ubuntu3) ...
Selecting previously unselected package libxcb-xkb1:amd64.
Preparing to unpack .../15-libxcb-xkb1_1.14-3ubuntu3_amd64.deb ...
Unpacking libxcb-xkb1:amd64 (1.14-3ubuntu3) ...
Selecting previously unselected package libxkbcommon-x11-0:amd64.
Preparing to unpack .../16-libxkbcommon-x11-0_1.4.0-1_amd64.deb ...
Unpacking libxkbcommon-x11-0:amd64 (1.4.0-1) ...
Selecting previously unselected package libqt5gui5:amd64.
Preparing to unpack .../17-libqt5gui5_5.15.3+dfsg-2ubuntu0.2_amd64.deb ...
Unpacking libqt5gui5:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Selecting previously unselected package libqt5widgets5:amd64.
Preparing to unpack .../18-libqt5widgets5_5.15.3+dfsg-2ubuntu0.2_amd64.deb ...
```

```
Unpacking libqt5widgets5:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Selecting previously unselected package libqt5svg5:amd64.
Preparing to unpack .../19-libqt5svg5_5.15.3-1_amd64.deb ...
Unpacking libqt5svg5:amd64 (5.15.3-1) ...
Selecting previously unselected package libhyphen0:amd64.
Preparing to unpack .../20-libhyphen0_2.8.8-7build2_amd64.deb ...
Unpacking libhyphen0:amd64 (2.8.8-7build2) ...
Selecting previously unselected package libqt5positioning5:amd64.
Preparing to unpack .../21-libqt5positioning5_5.15.3+dfsg-3_amd64.deb ...
Unpacking libqt5positioning5:amd64 (5.15.3+dfsg-3) ...
Selecting previously unselected package libqt5printsupport5:amd64.
Preparing to unpack .../22-libqt5printsupport5_5.15.3+dfsg-2ubuntu0.2_amd64.deb ...
Unpacking libqt5printsupport5:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Selecting previously unselected package libqt5qml5:amd64.
Preparing to unpack .../23-libqt5qml5_5.15.3+dfsg-1_amd64.deb ...
Unpacking libqt5qml5:amd64 (5.15.3+dfsg-1) ...
Selecting previously unselected package libqt5qmlmodels5:amd64.
Preparing to unpack .../24-libqt5qmlmodels5_5.15.3+dfsg-1_amd64.deb ...
Unpacking libqt5qmlmodels5:amd64 (5.15.3+dfsg-1) ...
Selecting previously unselected package libqt5quick5:amd64.
Preparing to unpack .../25-libqt5quick5_5.15.3+dfsg-1_amd64.deb ...
Unpacking libqt5quick5:amd64 (5.15.3+dfsg-1) ...
Selecting previously unselected package libqt5sensors5:amd64.
Preparing to unpack .../26-libqt5sensors5_5.15.3-1_amd64.deb ...
Unpacking libqt5sensors5:amd64 (5.15.3-1) ...
Selecting previously unselected package libqt5webchannel5:amd64.
Preparing to unpack .../27-libqt5webchannel5_5.15.3-1_amd64.deb ...
Unpacking libqt5webchannel5:amd64 (5.15.3-1) ...
Selecting previously unselected package libwoff1:amd64.
Preparing to unpack .../28-libwoff1_1.0.2-1build4_amd64.deb ...
Unpacking libwoff1:amd64 (1.0.2-1build4) ...
Selecting previously unselected package libqt5webkit5:amd64.
Preparing to unpack .../29-libqt5webkit5_5.212.0~alpha4-15ubuntu1_amd64.deb ...
Unpacking libqt5webkit5:amd64 (5.212.0~alpha4-15ubuntu1) ...
Selecting previously unselected package udev.
Preparing to unpack .../30-udev_249.11-0ubuntu3.16_amd64.deb ...
Unpacking udev (249.11-0ubuntu3.16) ...
Selecting previously unselected package libavahi-glib1:amd64.
Preparing to unpack .../31-libavahi-glib1_0.8-5ubuntu5.2_amd64.deb ...
Unpacking libavahi-glib1:amd64 (0.8-5ubuntu5.2) ...
Selecting previously unselected package libjson-glib-1.0-common.
Preparing to unpack .../32-libjson-glib-1.0-common_1.6.6-1build1_all.deb ...
Unpacking libjson-glib-1.0-common (1.6.6-1build1) ...
Selecting previously unselected package libjson-glib-1.0-0:amd64.
Preparing to unpack .../33-libjson-glib-1.0-0_1.6.6-1build1_amd64.deb ...
Unpacking libjson-glib-1.0-0:amd64 (1.6.6-1build1) ...
Selecting previously unselected package libmm-glib0:amd64.
Preparing to unpack .../34-libmm-glib0_1.20.0-1~ubuntu22.04.4_amd64.deb ...
Unpacking libmm-glib0:amd64 (1.20.0-1~ubuntu22.04.4) ...
Selecting previously unselected package libnotify4:amd64.
Preparing to unpack .../35-libnotify4_0.7.9-3ubuntu5.22.04.1_amd64.deb ...
Unpacking libnotify4:amd64 (0.7.9-3ubuntu5.22.04.1) ...
Selecting previously unselected package libproxy1v5:amd64.
Preparing to unpack .../36-libproxy1v5_0.4.17-2_amd64.deb ...
Unpacking libproxy1v5:amd64 (0.4.17-2) ...
Selecting previously unselected package glib-networking-common.
Preparing to unpack .../37-glib-networking-common_2.72.0-1_all.deb ...
Unpacking glib-networking-common (2.72.0-1) ...
Selecting previously unselected package glib-networking-services.
Preparing to unpack .../38-glib-networking-services_2.72.0-1_amd64.deb ...
Unpacking glib-networking-services (2.72.0-1) ...
Selecting previously unselected package session-migration.
Preparing to unpack .../39-session-migration_0.3.6_amd64.deb ...
Unpacking session-migration (0.3.6) ...
Selecting previously unselected package gsettings-desktop-schemas.
Preparing to unpack .../40-gsettings-desktop-schemas_42.0-1ubuntu1_all.deb ...
Unpacking gsettings-desktop-schemas (42.0-1ubuntu1) ...
Selecting previously unselected package glib-networking:amd64.
Preparing to unpack .../41-glib-networking_2.72.0-1_amd64.deb ...
Unpacking glib-networking:amd64 (2.72.0-1) ...
Selecting previously unselected package libsoup2.4-common.
Preparing to unpack .../42-libsoup2.4-common_2.74.2-3ubuntu0.6_all.deb ...
Unpacking libsoup2.4-common (2.74.2-3ubuntu0.6) ...
Selecting previously unselected package libsoup2.4-1:amd64.
Preparing to unpack .../43-libsoup2.4-1_2.74.2-3ubuntu0.6_amd64.deb ...
Unpacking libsoup2.4-1:amd64 (2.74.2-3ubuntu0.6) ...
Selecting previously unselected package geoclue-2.0.
Preparing to unpack .../44-geoclue-2.0_2.5.7-3ubuntu3_amd64.deb ...
Unpacking geoclue-2.0 (2.5.7-3ubuntu3) ...
Selecting previously unselected package iio-sensor-proxy.
Preparing to unpack .../45-iio-sensor-proxy_3.3-0ubuntu6_amd64.deb ...
Unpacking iio-sensor-proxy (3.3-0ubuntu6) ...
Selecting previously unselected package libmbim-glib4:amd64.
```

```
Preparing to unpack .../46-libmbim-glib4_1.28.0-1~ubuntu20.04.2_amd64.deb ...
Unpacking libmbim-glib4:amd64 (1.28.0-1~ubuntu20.04.2) ...
Selecting previously unselected package libmbim-proxy.
Preparing to unpack .../47-libmbim-proxy_1.28.0-1~ubuntu20.04.2_amd64.deb ...
Unpacking libmbim-proxy (1.28.0-1~ubuntu20.04.2) ...
Selecting previously unselected package libnl-genl-3-200:amd64.
Preparing to unpack .../48-libnl-genl-3-200_3.5.0-0.1_amd64.deb ...
Unpacking libnl-genl-3-200:amd64 (3.5.0-0.1) ...
Selecting previously unselected package libnss-mdns:amd64.
Preparing to unpack .../49-libnss-mdns_0.15.1-1ubuntu1_amd64.deb ...
Unpacking libnss-mdns:amd64 (0.15.1-1ubuntu1) ...
Selecting previously unselected package libqmi-glib5:amd64.
Preparing to unpack .../50-libqmi-glib5_1.32.0-1ubuntu0.22.04.1_amd64.deb ...
Unpacking libqmi-glib5:amd64 (1.32.0-1ubuntu0.22.04.1) ...
Selecting previously unselected package libqmi-proxy.
Preparing to unpack .../51-libqmi-proxy_1.32.0-1ubuntu0.22.04.1_amd64.deb ...
Unpacking libqmi-proxy (1.32.0-1ubuntu0.22.04.1) ...
Selecting previously unselected package libwacom-bin.
Preparing to unpack .../52-libwacom-bin_2.2.0-1_amd64.deb ...
Unpacking libwacom-bin (2.2.0-1) ...
Selecting previously unselected package modemmanager.
Preparing to unpack .../53-modemmanager_1.20.0-1~ubuntu22.04.4_amd64.deb ...
Unpacking modemmanager (1.20.0-1~ubuntu22.04.4) ...
Selecting previously unselected package qt5-gtk-platformtheme:amd64.
Preparing to unpack .../54-qt5-gtk-platformtheme_5.15.3+dfsg-2ubuntu0.2_amd64.deb ...
Unpacking qt5-gtk-platformtheme:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Selecting previously unselected package qttranslations5-l10n.
Preparing to unpack .../55-qttranslations5-l10n_5.15.3-1_all.deb ...
Unpacking qttranslations5-l10n (5.15.3-1) ...
Selecting previously unselected package systemd-hwe-hwdb.
Preparing to unpack .../56-systemd-hwe-hwdb_249.11.5_all.deb ...
Unpacking systemd-hwe-hwdb (249.11.5) ...
Selecting previously unselected package wpasupplicant.
Preparing to unpack .../57-wpasupplicant_2%3a2.10-6ubuntu2.2_amd64.deb ...
Unpacking wpasupplicant (2:2.10-6ubuntu2.2) ...
Selecting previously unselected package usb-modeswitch-data.
Preparing to unpack .../58-usb-modeswitch-data_20191128-4_all.deb ...
Unpacking usb-modeswitch-data (20191128-4) ...
Selecting previously unselected package usb-modeswitch.
Preparing to unpack .../59-usb-modeswitch_2.6.1-3ubuntu2_amd64.deb ...
Unpacking usb-modeswitch (2.6.1-3ubuntu2) ...
Selecting previously unselected package wkhtmltopdf.
Preparing to unpack .../60-wkhtmltopdf_0.12.6-2_amd64.deb ...
Unpacking wkhtmltopdf (0.12.6-2) ...
Setting up session-migration (0.3.6) ...
Created symlink /etc/systemd/user/graphical-session-pre.target.wants/session-migration.service → /usr/lib/system
d/user/session-migration.service.
Setting up libproxy1v5:amd64 (0.4.17-2) ...
Setting up libxcb-xinput0:amd64 (1.14-3ubuntu3) ...
Setting up libwoff1:amd64 (1.0.2-1build4) ...
Setting up libhyphen0:amd64 (2.8.8-7build2) ...
Setting up libxcb-keysyms1:amd64 (0.4.0-1build3) ...
Setting up libxcb-render-util0:amd64 (0.3.9-1build3) ...
Setting up libxcb-icccm4:amd64 (0.4.1-1.1build2) ...
Setting up libxcb-util1:amd64 (0.4.0-1build2) ...
Setting up libxcb-xkb1:amd64 (1.14-3ubuntu3) ...
Setting up libxcb-image0:amd64 (0.4.0-2) ...
Setting up libxcb-xinerama0:amd64 (1.14-3ubuntu3) ...
Setting up qttranslations5-l10n (5.15.3-1) ...
Setting up libnotify4:amd64 (0.7.9-3ubuntu5.22.04.1) ...
Setting up libxkbcommon-x11-0:amd64 (1.4.0-1) ...
Setting up usb-modeswitch-data (20191128-4) ...
Setting up udev (249.11-0ubuntu3.16) ...
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.
Setting up libqt5core5a:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Setting up libmtdev1:amd64 (1.1.6-1build4) ...
Setting up libsoup2.4-common (2.74.2-3ubuntu0.6) ...
Setting up systemd-hwe-hwdb (249.11.5) ...
Setting up libmm-glib0:amd64 (1.20.0-1~ubuntu22.04.4) ...
Setting up libqt5dbus5:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Setting up libnl-genl-3-200:amd64 (3.5.0-0.1) ...
Setting up libmd4c0:amd64 (0.4.8-1) ...
Setting up libavahi-glib1:amd64 (0.8-5ubuntu5.2) ...
Setting up libjson-glib-1.0-common (1.6.6-1build1) ...
Setting up usb-modeswitch (2.6.1-3ubuntu2) ...
Setting up glib-networking-common (2.72.0-1) ...
Setting up libqt5sensors5:amd64 (5.15.3-1) ...
Setting up libdaemon0:amd64 (0.14-7.1ubuntu3) ...
Setting up libavahi-core7:amd64 (0.8-5ubuntu5.2) ...
Setting up libnss-mdns:amd64 (0.15.1-1ubuntu1) ...
First installation detected...
Checking NSS setup...
```

```
Setting up libevdev2:amd64 (1.12.1+dfsg-1) ...
Setting up libgudev-1.0-0:amd64 (1:237-2build1) ...
Setting up libmbim-glib4:amd64 (1.28.0-1~ubuntu20.04.2) ...
Setting up libwacom-common (2.2.0-1) ...
Setting up gsettings-desktop-schemas (42.0-1ubuntu1) ...
Setting up glib-networking-services (2.72.0-1) ...
Setting up iio-sensor-proxy (3.3-0ubuntu6) ...
Setting up libwacom9:amd64 (2.2.0-1) ...
Setting up libqt5positioning5:amd64 (5.15.3+dfsg-3) ...
Setting up libmbim-proxy (1.28.0-1~ubuntu20.04.2) ...
Setting up libqt5network5:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Setting up libjson-glib-1.0-0:amd64 (1.6.6-1build1) ...
Setting up libinput-bin (1.20.0-1ubuntu0.3) ...
Setting up wpasupplicant (2:2.10-6ubuntu2.2) ...
Created symlink /etc/systemd/system/dbus-fi.w1.wpa_supplicant1.service → /lib/systemd/system/wpa_supplicant.serv
ice.
Created symlink /etc/systemd/system/multi-user.target.wants/wpa_supplicant.service → /lib/systemd/system/wpa_sup
plicant.service.
Setting up libqt5qml5:amd64 (5.15.3+dfsg-1) ...
Setting up libqt5webchannel5:amd64 (5.15.3-1) ...
Setting up libwacom-bin (2.2.0-1) ...
Setting up avahi-daemon (0.8-5ubuntu5.2) ...
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of force-reload.
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.
Created symlink /etc/systemd/system/dbus-org.freedesktop.Avahi.service → /lib/systemd/system/avahi-daemon.servic
e.
Created symlink /etc/systemd/system/multi-user.target.wants/avahi-daemon.service → /lib/systemd/system/avahi-dae
mon.service.
Created symlink /etc/systemd/system/sockets.target.wants/avahi-daemon.socket → /lib/systemd/system/avahi-daemon.
socket.
Setting up libinput10:amd64 (1.20.0-1ubuntu0.3) ...
Setting up libqt5qmlmodels5:amd64 (5.15.3+dfsg-1) ...
Setting up libqt5gui5:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Setting up libqmi-glib5:amd64 (1.32.0-1ubuntu0.22.04.1) ...
Setting up libqt5widgets5:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Setting up qt5-gtk-platformtheme:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Setting up libqt5printsupport5:amd64 (5.15.3+dfsg-2ubuntu0.2) ...
Setting up libqt5quick5:amd64 (5.15.3+dfsg-1) ...
Setting up libqt5svg5:amd64 (5.15.3-1) ...
Setting up libqmi-proxy (1.32.0-1ubuntu0.22.04.1) ...
Setting up libqt5webkit5:amd64 (5.212.0~alpha4-15ubuntu1) ...
Setting up modemmanager (1.20.0-1~ubuntu22.04.4) ...
Created symlink /etc/systemd/system/dbus-org.freedesktop.ModemManager1.service → /lib/systemd/system/ModemManage
r.service.
Created symlink /etc/systemd/system/multi-user.target.wants/ModemManager.service → /lib/systemd/system/ModemMana
ger.service.
Setting up wkhtmltopdf (0.12.6-2) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for dbus (1.12.20-2ubuntu4.1) ...
Processing triggers for mailcap (3.70+nmu1ubuntu1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for libglib2.0-0:amd64 (2.72.4-0ubuntu2.6) ...
Processing triggers for libc-bin (2.35-0ubuntu3.8) ...
/sbin/ldconfig.real: /usr/local/lib/libumf.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libhwloc.so.15 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_loader.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_adapter_level_zero.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc.so.2 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_0.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtcm_debug.so.1 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_adapter_opencl.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_5.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc_proxy.so.2 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_adapter_level_zero_v2.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtcm.so.1 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbb.so.12 is not a symbolic link
```

```
Setting up glib-networking:amd64 (2.72.0-1) ...
Setting up libsoup2.4-1:amd64 (2.74.2-3ubuntu0.6) ...
Setting up geoclue-2.0 (2.5.7-3ubuntu3) ...
Processing triggers for libc-bin (2.35-0ubuntu3.8) ...
/sbin/ldconfig.real: /usr/local/lib/libumf.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libhwloc.so.15 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_loader.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_adapter_level_zero.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc.so.2 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_0.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtcm_debug.so.1 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_adapter_opencl.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_5.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc_proxy.so.2 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_adapter_level_zero_v2.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtcm.so.1 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbb.so.12 is not a symbolic link

Processing triggers for dbus (1.12.20-2ubuntu4.1) ...
```

In [ ]:

```
# Installa pdfkit e wkhtmltopdf
!pip install pdfkit
!sudo apt-get update
!sudo apt-get install -y wkhtmltopdf

# Verifica il percorso di wkhtmltopdf (di solito è /usr/bin/wkhtmltopdf)
!which wkhtmltopdf

# Configura pdfkit con il percorso corretto
import pdfkit

# Specifica il percorso di wkhtmltopdf (verifica con `which wkhtmltopdf`)
config = pdfkit.configuration(wkhtmltopdf="/usr/bin/wkhtmltopdf")

# Esempio di conversione da HTML a PDF
html_content = """
<html>
  <body>
    <h1>Test PDF</h1>
    <p>Questo è un PDF generato con pdfkit e wkhtmltopdf!</p>
  </body>
</html>
"""

# Genera il PDF
pdfkit.from_string(html_content, "Etna2018.pdf", configuration=config)

print("PDF generato con successo!")
```

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")
```