1

```python
#Nr,Kod stacji,Kod międzynarodowy,Nazwa stacji,"Stary Kod stacji
#(o ile inny od aktualnego)",Data uruchomienia,Data zamknięcia,Typ stacji,Typ obszaru,Rodzaj stacji,Województwo,Miejscowość,Adres,WGS84 φ N,WGS84 λ E
import lib
class Station:
    def __init__(
            self,
            nr,
            kod_stacji,
            kod_miedz,
            nazwa,
            stary_kod,
            data_begin,
            data_end,
            typ,
            obszar,
            rodzaj,
            woj,
            miejsc,
            adres,
            lon,
            lat
    ):
        for name, value in locals().items():
            if name != 'self':
                setattr(self, name, value)

        return

    def __str__(self):
        res = ''
        for k, v in self.__dict__.items():
            res += f'{k}: {v};'
        return res

    def __repr__(self):
        cls_name = self.__class__.__name__
        attrs = ', '.join(f"{k}={repr(v)}" for k, v in self.__dict__.items())
        return f"{cls_name}({attrs})"

    def __eq__(self, other):
        return self.kod_stacji == other.kod_stacji

if __name__ == '__main__':
    filedata = lib.read_stations()
    print(filedata[3])

    print(filedata[4])
    args1 = lib.split_line(filedata[3])
    args2 = lib.split_line(filedata[4])

    st1 = Station(*args1)
    st2 = Station(*args2)
    print(st1 == st2)
    st2.kod_stacji = st1.kod_stacji
    print(st1 == st2)
    print('\n\n')
    print(str(st1))
    print('\n\n')
    print(str(st2))
    print('\n\n')
    print(repr(st1))
    print('\n\n')
    print(repr(st2))
```

```python
import re

def read_stations() -> list:
    split_pattern = r'\n(?=(?:[^"]*"[^"]*")*[^"]*$)'

    file = open('data/stacje.csv', 'r')
    filedata = file.read().strip()
    filedata = re.split(split_pattern, filedata)
    file.close()
    return filedata

def split_line(line):
    comma_pattern = r',(?=(?:[^"]*"[^"]*")*[^"]*$)'
    values = re.split(comma_pattern, line.strip())
    return values

def get_measurements(quan, freq) -> list:
    from os import listdir, path

    folder = 'data/measurements'
    files = []
    for i in listdir(folder):
        temp = re.split(r'_|\.', i)
        if len(temp) != 4: continue
        if quan in temp[1] and freq == temp[2]:
            files.append(path.join(folder, i))

    return files
```

```
3,DsBogatFrancMOB,PL0602A,Bogatynia Mobil,DsBogatMob,2015-01-01,2015-12-31,tło,miejski,mobilna,DOLNOŚLĄSKIE,Bogatynia,ul. Francuska/Kręta,50.940998,14.916790
4,DsBogChop,PL0315A,Bogatynia - Chopina,,1996-01-01,2013-12-31,przemysłowa,miejski,kontenerowa stacjonarna,DOLNOŚLĄSKIE,Bogatynia,ul. Chopina 35,50.905856,14.967175
False
True



nr: 3;kod_stacji: DsBogatFrancMOB;kod_miedz: PL0602A;nazwa: Bogatynia Mobil;stary_kod: DsBogatMob;data_begin: 2015-01-01;data_end: 2015-12-31;typ: tło;obszar: miejski;rodzaj: mobilna;woj: DOLNOŚLĄSKIE;mie
jsc: Bogatynia;adres: ul. Francuska/Kręta;lon: 50.940998;lat: 14.916790;



nr: 4;kod_stacji: DsBogatFrancMOB;kod_miedz: PL0315A;nazwa: Bogatynia - Chopina;stary_kod: ;data_begin: 1996-01-01;data_end: 2013-12-31;typ: przemysłowa;obszar: miejski;rodzaj: kontenerowa stacjonarna;woj
: DOLNOŚLĄSKIE;miejsc: Bogatynia;adres: ul. Chopina 35;lon: 50.905856;lat: 14.967175;



Station(nr='3', kod_stacji='DsBogatFrancMOB', kod_miedz='PL0602A', nazwa='Bogatynia Mobil', stary_kod='DsBogatMob', data_begin='2015-01-01', data_end='2015-12-31', typ='tło', obszar='miejski', rodzaj='mob
ilna', woj='DOLNOŚLĄSKIE', miejsc='Bogatynia', adres='ul. Francuska/Kręta', lon='50.940998', lat='14.916790')



Station(nr='4', kod_stacji='DsBogatFrancMOB', kod_miedz='PL0315A', nazwa='Bogatynia - Chopina', stary_kod='', data_begin='1996-01-01', data_end='2013-12-31', typ='przemysłowa', obszar='miejski', rodzaj='k
ontenerowa stacjonarna', woj='DOLNOŚLĄSKIE', miejsc='Bogatynia', adres='ul. Chopina 35', lon='50.905856', lat='14.967175')
```

2, 3

```python
import numpy as np
from datetime import datetime, timedelta

class TimeSeries:
    def __init__(self, indicator_name, station_code, averaging_time, dates, values, unit):
        self.indicator_name = indicator_name
        self.station_code = station_code
        self.averaging_time = averaging_time
        self.dates = dates
        self.values = values
        self.unit = unit

    def __getitem__(self, key):
        if isinstance(key, slice):
            return [(self.dates[i], self.values[i]) for i in range(key.start, key.stop, key.step or 1)]
        elif isinstance(key, (int, float)):
            return self.dates[key], self.values[key]
        elif isinstance(key, (datetime, datetime.date)):
            if key in self.dates:
                idx = self.dates.index(key)
                return self.dates[idx], self.values[idx]
            else:
                raise KeyError(f"Data {key} nie istnieje w danych pomiarowych.")
        else:
            raise TypeError(f"Nieobsługiwany typ klucza: {type(key)}")

    @property
    def mean(self):
        # Obliczanie średniej arytmetycznej
        values = [v for v in self.values if v is not None]
        if values:
            return np.mean(values)
        return None

    @property
    def stddev(self):
        # Obliczanie odchylenia standardowego
        values = [v for v in self.values if v is not None]
        if values:
            return np.std(values)
        return None

if __name__ == '__main__':
    indicator_name = "PM10"
    station_code = "ST01"
    averaging_time = "1h"
    unit = "µg/m3"
    start = datetime(2024, 1, 1, 0, 0)
    dates = [start + timedelta(hours=i) for i in range(5)]
    values = [15.0, 17.3, None, 19.8, 20.1]
    ts = TimeSeries(indicator_name, station_code, averaging_time, dates, values, unit)
    print("Nazwa wskaźnika:", ts.indicator_name)
    print("Średnia:", ts.mean)
    print("Odchylenie standardowe:", ts.stddev)
    print("Wartość z trzeciego pomiaru:", ts[2])
    print("Wartości od 1 do 4:", ts[1:4])
    print("Wartość z daty:", ts[dates[1]])
    print("Wartość z indexu 1:", ts[1])
    print("Wartość z indexu 2:", ts[2])
    print("Slice object", ts[1:3])
```

```
Nazwa wskaźnika: PM10
Średnia: 18.049999999999997
Odchylenie standardowe: 2.069420208657488
Wartość z trzeciego pomiaru: (datetime.datetime(2024, 1, 1, 2, 0), None)
Wartości od 1 do 4: [(datetime.datetime(2024, 1, 1, 1, 0), 17.3), (datetime.datetime(2024, 1, 1, 2, 0), None), (datetime.datetime(2024, 1, 1, 3, 0), 19.8)]
Wartość z daty: (datetime.datetime(2024, 1, 1, 1, 0), 17.3)
Wartość z indexu 1: (datetime.datetime(2024, 1, 1, 1, 0), 17.3)
Wartość z indexu 2: (datetime.datetime(2024, 1, 1, 2, 0), None)
Slice object [(datetime.datetime(2024, 1, 1, 1, 0), 17.3), (datetime.datetime(2024, 1, 1, 2, 0), None)]



Nazwa wskaźnika: PM10
Średnia: 17.466666666666665
Odchylenie standardowe: 8.377085146729472
Wartość z trzeciego pomiaru: (datetime.datetime(2025, 1, 1, 2, 0), None)
Wartości od 1 do 4: [(datetime.datetime(2025, 1, 1, 1, 0), 17.3), (datetime.datetime(2025, 1, 1, 2, 0), None), (datetime.datetime(2025, 1, 1, 3, 0), 19.8)]
Wartość z daty: (datetime.datetime(2025, 1, 1, 1, 0), 17.3)
Wartość z indexu 1: (datetime.datetime(2025, 1, 1, 1, 0), 17.3)
Wartość z indexu 2: (datetime.datetime(2025, 1, 1, 2, 0), None)
Slice object [(datetime.datetime(2025, 1, 1, 1, 0), 17.3), (datetime.datetime(2025, 1, 1, 3, 0), 19.8), (datetime.datetime(2025, 1, 1, 5, 0), 10.0), (datetime.datetime(2025, 1, 1, 7, 0), 34.0)]



Nazwa wskaźnika: PM10
Średnia: 15.533333333333331
Odchylenie standardowe: 6.7531885473127105
Wartość z trzeciego pomiaru: (datetime.datetime(2025, 1, 1, 2, 0), None)
Wartości od 1 do 4: [(datetime.datetime(2025, 1, 1, 1, 0), 17.3), (datetime.datetime(2025, 1, 1, 2, 0), None), (datetime.datetime(2025, 1, 1, 3, 0), 19.8)]
Wartość z daty: (datetime.datetime(2025, 1, 1, 1, 0), 17.3)
Wartość z indexu 1: (datetime.datetime(2025, 1, 1, 1, 0), 17.3)
Wartość z indexu 2: (datetime.datetime(2025, 1, 1, 2, 0), None)
Slice object [(datetime.datetime(2025, 1, 1, 1, 0), 17.3), (datetime.datetime(2025, 1, 1, 3, 0), 19.8)]
```

4

```python
from z2_3 import TimeSeries
from abc import ABC, abstractmethod
import numpy as np
from typing import List
from datetime import datetime, timedelta

class SeriesValidator(ABC):
    @abstractmethod
    def analyze(self, series: TimeSeries) -> List[str]:
        pass


class OutlierDetector(SeriesValidator):
    def __init__(self, k: float):
        self.k = k

    def analyze(self, series: TimeSeries) -> List[str]:
        mean = series.mean
        stddev = series.stddev
        if mean is None or stddev is None:
            return []

        anomalies = []
        for date, value in zip(series.dates, series.values):
            if value is not None and abs(value - mean) > self.k * stddev:
                anomalies.append(f"Outlier detected: {value} at {date} (more than {self.k} standard deviations from mean)")
        return anomalies


class ZeroSpikeDetector(SeriesValidator):
    def analyze(self, series: TimeSeries) -> List[str]:
        anomalies = []
        count = 0
        for date, value in zip(series.dates, series.values):
            if value == 0 or value is None:
                count += 1
            else:
                count = 0

            if count >= 3:
                anomalies.append(f"Zero spike detected at {date} (3 or more consecutive zeros or missing values)")
        return anomalies



class ThresholdDetector(SeriesValidator):
    def __init__(self, threshold: float):
        self.threshold = threshold

    def analyze(self, series: TimeSeries) -> List[str]:
        anomalies = []
        for date, value in zip(series.dates, series.values):
            if value is not None and value > self.threshold:
                anomalies.append(f"Threshold exceeded: {value} at {date} (greater than {self.threshold})")
        return anomalies


class CompositeValidator(SeriesValidator):
    def __init__(self, validators: List[SeriesValidator], mode: str = "OR"):
        self.validators = validators
        self.mode = mode.upper()

    def analyze(self, series: TimeSeries) -> List[str]:
        anomalies =[]

        # AND returns only if every validator had result if not empty list
        if self.mode == "AND":
            for validator in self.validators:
                result = validator.analyze(series)
                if result:
                    anomalies.extend(result)
                else:
                    return []
        # OR returns everything that validators returns
        elif self.mode == "OR":
            for validator in self.validators:
                result = validator.analyze(series)
                anomalies.extend(result)
            return anomalies

        else:
            raise ValueError("Mode must be 'AND' or 'OR'")

if __name__ == '__main__':
    zero = ZeroSpikeDetector()
    threshold = ThresholdDetector(10)
    outlier = OutlierDetector(1)
    composite = CompositeValidator([zero, threshold])
```

```python
if __name__ == '__main__':
    zero = ZeroSpikeDetector()
    threshold = ThresholdDetector(10)
    outlier = OutlierDetector(1)
    composite = CompositeValidator([zero,threshold])
    indicator_name = "PM10"
    station_code = "ST01"
    averaging_time = "1h"
    unit = "µg/m3"
    start = datetime(2024, 1, 1, 0, 0)
    dates = [start + timedelta(hours=i) for i in range(5)]
    values = [15.0, 17.3, None, None, None]
    ts = TimeSeries(indicator_name, station_code, averaging_time, dates, values, unit)
    print('Zero: ', zero.analyze(ts))
    print('Threshold: ', threshold.analyze(ts))
    print('outlier', outlier.analyze(ts))
    print('composite', composite.analyze(ts))
    print('\n\n')
    start = datetime(2024, 1, 1, 0, 0)
    dates = [start + timedelta(hours=i) for i in range(5)]
    values = [15.0, 17.3, 1.0, 1000.0, None]
    ts = TimeSeries(indicator_name, station_code, averaging_time, dates, values, unit)
    print('Zero: ', zero.analyze(ts))
    print('Threshold: ', threshold.analyze(ts))
    print('outlier', outlier.analyze(ts))
    print('composite', composite.analyze(ts))
    print('\n\n')
    start = datetime(2025, 1, 1, 0, 0)
    dates = [start + timedelta(hours=i) for i in range(5)]
    values = [1.0, None, 0, 0, None]
    ts = TimeSeries(indicator_name, station_code, averaging_time, dates, values, unit)
    print('Zero: ', zero.analyze(ts))
    print('Threshold: ', threshold.analyze(ts))
    print('outlier', outlier.analyze(ts))
    composite.mode = "AND"
    print('composite', composite.analyze(ts))
```

```
Zero:  ['Zero spike detected at 2024-01-01 04:00:00 (3 or more consecutive zeros or missing values)']
Threshold:  ['Threshold exceeded: 15.0 at 2024-01-01 00:00:00 (greater than 10)', 'Threshold exceeded: 17.3 at 2024-01-01 01:00:00 (greater than 10)']
outlier ['Outlier detected: 17.3 at 2024-01-01 01:00:00 (more than 1 standard deviations from mean)']
composite ['Zero spike detected at 2024-01-01 04:00:00 (3 or more consecutive zeros or missing values)', 'Threshold exceeded: 15.0 at 2024-01-01 00:00:00 (greater than 10)', 'Threshold exceeded: 17.3 at 2024-01-01 01:00:00 (greater than 10)']



Zero:  []
Threshold:  ['Threshold exceeded: 15.0 at 2024-01-01 00:00:00 (greater than 10)', 'Threshold exceeded: 17.3 at 2024-01-01 01:00:00 (greater than 10)', 'Threshold exceeded: 1000.0 at 2024-01-01 03:00:00 (greater than 10)']
outlier ['Outlier detected: 1000.0 at 2024-01-01 03:00:00 (more than 1 standard deviations from mean)']
composite ['Threshold exceeded: 15.0 at 2024-01-01 00:00:00 (greater than 10)', 'Threshold exceeded: 17.3 at 2024-01-01 01:00:00 (greater than 10)', 'Threshold exceeded: 1000.0 at 2024-01-01 03:00:00 (greater than 10)']



Zero:  ['Zero spike detected at 2025-01-01 03:00:00 (3 or more consecutive zeros or missing values)', 'Zero spike detected at 2025-01-01 04:00:00 (3 or more consecutive zeros or missing values)']
Threshold:  []
outlier ['Outlier detected: 1.0 at 2025-01-01 00:00:00 (more than 1 standard deviations from mean)']
composite []
```

5, 6

```python
from pathlib import Path
import re
import os
from typing import List
from datetime import datetime
import csv
from z2_3 import TimeSeries
from z4 import ZeroSpikeDetector

class Measurements:
    def __init__(self, directory : str):
        self.directory = directory
        self.files = {}
        self.possibleTimeSeries = None
        for file in Path(directory).iterdir():
            pattern = re.compile(r'(\d{4})_(\w+)_(\d\w*)\.csv$')
            match = pattern.match(file.name)
            if match:
                year, measurement, frequency = match.groups()
                key = (year, measurement, frequency)
                self.files[key] = {"filename": file,
                                   "is_loaded": False,
                                   "TimeSeries": []}

    def __len__(self):
        if self.possibleTimeSeries is None:
            self.possibleTimeSeries = 0
            for file_key in self.files:
                with open(self.files[file_key]["filename"], newline='', encoding='utf-8') as csvfile:
                    self.possibleTimeSeries += len(csvfile.readline().split(',')) - 1
        return self.possibleTimeSeries

    def get_by_parameter(self, param_name: str) -> List:
        matching_series = []
        for file_key in self.files:
            _, measurement, _ = file_key
            if measurement == param_name:
                matching_series.extend(self.load_data(file_key)["TimeSeries"])
        return matching_series

    def get_by_station(self, station_code: str) -> List:
        matching_series = []
        for file_key in self.files:
            year, measurement, frequency = file_key
            file_path = self.files[file_key]["filename"]
            with open(file_path, newline='', encoding='utf-8') as csvfile:
                reader = csv.reader(csvfile)
                next(reader)
                station_codes = next(reader)[1:]
                if station_code in station_codes:
                    time_series_data = self.load_data(file_key)["TimeSeries"]
                    for ts in time_series_data:
                        if ts.station_code == station_code:
                            matching_series.append(ts)
        return matching_series

    def load_data(self, file_key) -> List[tuple]:
        if not self.files[file_key]["is_loaded"]:
            self.files[file_key]["is_loaded"] = True
            with open(self.files[file_key]["filename"], newline='', encoding='utf-8') as f:
                reader = csv.reader(f)
                rows = list(reader)

                indicators = rows[2][1:]
                station_code = rows[1][1:]
                avereging_time = rows[3][1:]
                unit = rows[4][1:]

                for i in range(len(indicators)):
                    self.files[file_key]["TimeSeries"].append( TimeSeries(indicators[i], station_code[i], avereging_time[i],[],[],unit[i]))

                for row in rows[6:]:
                    try:
                        date = datetime.strptime(row[0], "%m/%d/%y %H:%M")
                    except ValueError:
                        continue
                    for i in range(1, len(row)):
                        try:
                            value = float(row[i]) if row[i] != '' else None
                            self.files[file_key]["TimeSeries"][i-1].dates.append(date)
                            self.files[file_key]["TimeSeries"][i-1].values.append(value)
                        except ValueError:
                            self.files[file_key]["TimeSeries"][i-1].dates.append(date)
                            self.files[file_key]["TimeSeries"][i-1].values.append(None)

        return self.files[file_key]

    def detect_all_anomalies(self, validators: list, preload: bool = False) -> dict:
        result = {}
```

```python
    def detect_all_anomalies(self, validators: list, preload: bool = False) -> dict:
        result = {}

        for file_key in self.files:
            if preload or self.files[file_key]["is_loaded"]:
                anomalies_by_series = []

                time_series_list = self.load_data(file_key)["TimeSeries"]

                for ts in time_series_list:
                    anomalies = []
                    for validator in validators:
                        anomalies.extend(validator.analyze(ts))

                    series_info = {
                        'station_code': ts.station_code,
                        'indicator_name': ts.indicator_name,
                        'averaging_time': ts.averaging_time,
                        'unit': ts.unit,
                        'anomalies': anomalies
                    }

                    anomalies_by_series.append(series_info)

                result[file_key] = anomalies_by_series

        return result

f __name__ == '__main__':
    ms = Measurements("data/measurements")
    print(len(ms))
    print(ms.get_by_parameter('NO')[1].indicator_name
    print(ms.get_by_parameter('PM25')[1].indicator_name)
    print(ms.get_by_parameter('PM10')[1].indicator_name)


    print('\n Station test')
    print(ms.get_by_station("DsLegAlRzecz")[1].station_code)
    print(ms.get_by_station("KpNaklWawrzy")[0].station_code)
    print(ms.get_by_station("DsWrocWybCon")[0].station_code)
```

(function) indicator_name: Any

```
kac@kla:~/studia/python_syf/jezyki-skryptowe/l6$ python3 z5_6.py
1324
NO
PM2.5
PM10

 Station test
DsLegAlRzecz
KpNaklWawrzy
DsWrocWybCon
```

```
    return result

if __name__ == '__main__':
    ms = Measurements("data/measurements")
    print(len(ms))
    print(ms.get_by_parameter('NO')[1].indicator_name)

    print('\n Walidation test')
    zero = ZeroSpikeDetector()
    print(ms.detect_all_anomalies([zero], False))

    # print(ms.get_by_parameter('PM25')[1].indicator_name)
```

7

```python
    def detect_all_anomalies(self, validators: list, preload: bool = False) -> dict:
        result = {}

        for file_key in self.files:
            if preload or self.files[file_key]["is_loaded"]:
                anomalies_by_series = []

                time_series_list = self.load_data(file_key)["TimeSeries"]

                for ts in time_series_list:
                    anomalies = []
                    for validator in validators:
                        anomalies.extend(validator.analyze(ts))

                    series_info = {
                        'station_code': ts.station_code,
                        'indicator_name': ts.indicator_name,
                        'averaging_time': ts.averaging_time,
                        'unit': ts.unit,
                        'anomalies': anomalies
                    }

                    anomalies_by_series.append(series_info)

                result[file_key] = anomalies_by_series

        return result

if __name__ == '__main__':
    ms = Measurements("data/measurements")
    print(len(ms))
    print(ms.get_by_parameter('NO')[1].indicator_name)

    print('\n Walidation test')
    zero = ZeroSpikeDetector()
    threshold = ThresholdDetector(10)
    outlier = OutlierDetector(1)
    composite = CompositeValidator([zero,threshold])
    print(ms.detect_all_anomalies([zero, threshold, outlier, composite], False))
```

8

z8.py > ...

```python
from z2_3 import TimeSeries
from typing import List
from z2_3 import TimeSeries
from z4 import ZeroSpikeDetector
from z4 import OutlierDetector
from z4 import ThresholdDetector
from z4 import CompositeValidator
from datetime import datetime, timedelta

class SimpleReporter:
    def analyze(self, series: TimeSeries) -> List[str]:
        return [f"Info: {series.indicator_name} at {series.station_code} has mean = {series.mean:.2f}"]

if __name__ == '__main__':
    analyzer = []
    analyzer.append(ZeroSpikeDetector())
    analyzer.append(ZeroSpikeDetector())
    analyzer.append(ThresholdDetector(10))
    analyzer.append(SimpleReporter())
    analyzer.append(OutlierDetector(1))
    indicator_name = "PM10"
    station_code = "STACJA01"
    averaging_time = "1h"
    unit = "µg/m3"
    start = datetime(2024, 1, 1, 0, 0)
    dates = [start + timedelta(hours=i) for i in range(5)]
    values = [15.0, 17.3, None, None, None]
    ts = TimeSeries(indicator_name, station_code, averaging_time, dates, values, unit)
    for analiz in analyzer:
        print(analiz.analyze(ts))
    analyzer.append(SimpleReporter())
    analyzer.append(SimpleReporter())
    print('\n')
    for analiz in analyzer:
        print(analiz.analyze(ts))
    analyzer.append(SimpleReporter())
    analyzer.append(SimpleReporter())
    print('\n')
    for analiz in analyzer:
        print(analiz.analyze(ts))
```

```
kac@kla:~/studia/python_syf/jezyki-skryptowe/l6$ python3 z8.py
['Zero spike detected at 2024-01-01 04:00:00 (3 or more consecutive zeros or missing values)']
['Zero spike detected at 2024-01-01 04:00:00 (3 or more consecutive zeros or missing values)']
['Threshold exceeded: 15.0 at 2024-01-01 00:00:00 (greater than 10)', 'Threshold exceeded: 17.3 at 2024-01-01 01:00:00 (greater than 10)']
['Info: PM10 at STACJA01 has mean = 16.15']
['Outlier detected: 17.3 at 2024-01-01 01:00:00 (more than 1 standard deviations from mean)']
kac@kla:~/studia/python_syf/jezyki-skryptowe/l6$ python3 z8.py
['Zero spike detected at 2024-01-01 04:00:00 (3 or more consecutive zeros or missing values)']
['Zero spike detected at 2024-01-01 04:00:00 (3 or more consecutive zeros or missing values)']
['Threshold exceeded: 15.0 at 2024-01-01 00:00:00 (greater than 10)', 'Threshold exceeded: 17.3 at 2024-01-01 01:00:00 (greater than 10)']
['Info: PM10 at STACJA01 has mean = 16.15']
['Outlier detected: 17.3 at 2024-01-01 01:00:00 (more than 1 standard deviations from mean)']


['Zero spike detected at 2024-01-01 04:00:00 (3 or more consecutive zeros or missing values)']
['Zero spike detected at 2024-01-01 04:00:00 (3 or more consecutive zeros or missing values)']
['Threshold exceeded: 15.0 at 2024-01-01 00:00:00 (greater than 10)', 'Threshold exceeded: 17.3 at 2024-01-01 01:00:00 (greater than 10)']
['Info: PM10 at STACJA01 has mean = 16.15']
['Outlier detected: 17.3 at 2024-01-01 01:00:00 (more than 1 standard deviations from mean)']
['Info: PM10 at STACJA01 has mean = 16.15']
['Info: PM10 at STACJA01 has mean = 16.15']


['Zero spike detected at 2024-01-01 04:00:00 (3 or more consecutive zeros or missing values)']
['Zero spike detected at 2024-01-01 04:00:00 (3 or more consecutive zeros or missing values)']
['Threshold exceeded: 15.0 at 2024-01-01 00:00:00 (greater than 10)', 'Threshold exceeded: 17.3 at 2024-01-01 01:00:00 (greater than 10)']
['Info: PM10 at STACJA01 has mean = 16.15']
['Outlier detected: 17.3 at 2024-01-01 01:00:00 (more than 1 standard deviations from mean)']
['Info: PM10 at STACJA01 has mean = 16.15']
['Info: PM10 at STACJA01 has mean = 16.15']
['Info: PM10 at STACJA01 has mean = 16.15']
['Info: PM10 at STACJA01 has mean = 16.15']
```