<table>
<tr><td colspan="3">Politechnika Świętokrzyska w Kielcach</td></tr>
<tr><td colspan="3">Wydział Elektrotechniki, Automatyki i Informatyki</td></tr>
<tr><td colspan="3">Języki Skryptowe - Laboratorium</td></tr>
<tr><td>Prowadzący:
Dr inż. Dariusz Michalski</td><td rowspan="2">Temat:
Projekt – Analizator danych meteorologicznych</td><td rowspan="2">Wykonali:
Wojciech Lis
Tomasz Kundera</td></tr>
<tr><td>Data oddania:
29 czerwca 2025</td></tr>
</table>

Cel projektu

Celem projektu było zaprojektowanie i napisane programu analizującego dane meteorologiczne.

Funkcje aplikacji

- Generowanie danych meteorologicznych do pliku tekstowego CSV
- Importowanie danych meteorologicznych z podanego pliku tekstowego CSV
- Wprowadzanie parametrów programu za pomocą wiersza poleceń
- Modyfikacja parametrów programu w trakcie jego działania
- Przedstawienie interfejsu aplikacji w postaci okien tekstowych
- Kreślenie i zapisywanie wykresów ilustrujących zaimportowane dane meteorologiczne
- Modyfikacja i przeglądanie zaimportowanych danych meteorologicznych w trakcie działania programu

Użyte technologie i biblioteki

- Interpreter języka `Python` 3.10.12 oraz system kontroli wersji `git`
- System operacyjny `GNU/Linux` oparty na dystrybucji `Ubuntu` i emulator terminala
- Biblioteki `matplotlib` wersji 3.5.1, `console-menu` wersji 0.8.0 i `flake8`

Struktura projektu

```
./
├── CHANGELOG.md
├── misc/
├── README.md
├── setup.cfg
└── src/
    ├── meteolizer.py
    ├── tests/
    │   ├── test_analysis.py
    │   ├── test_data_generator.py
    │   └── test_models.py
    └── utils/
        ├── analysis.py
        ├── data_generator.py
        ├── __init__.py
        └── models.py
```

Opis struktury projektu

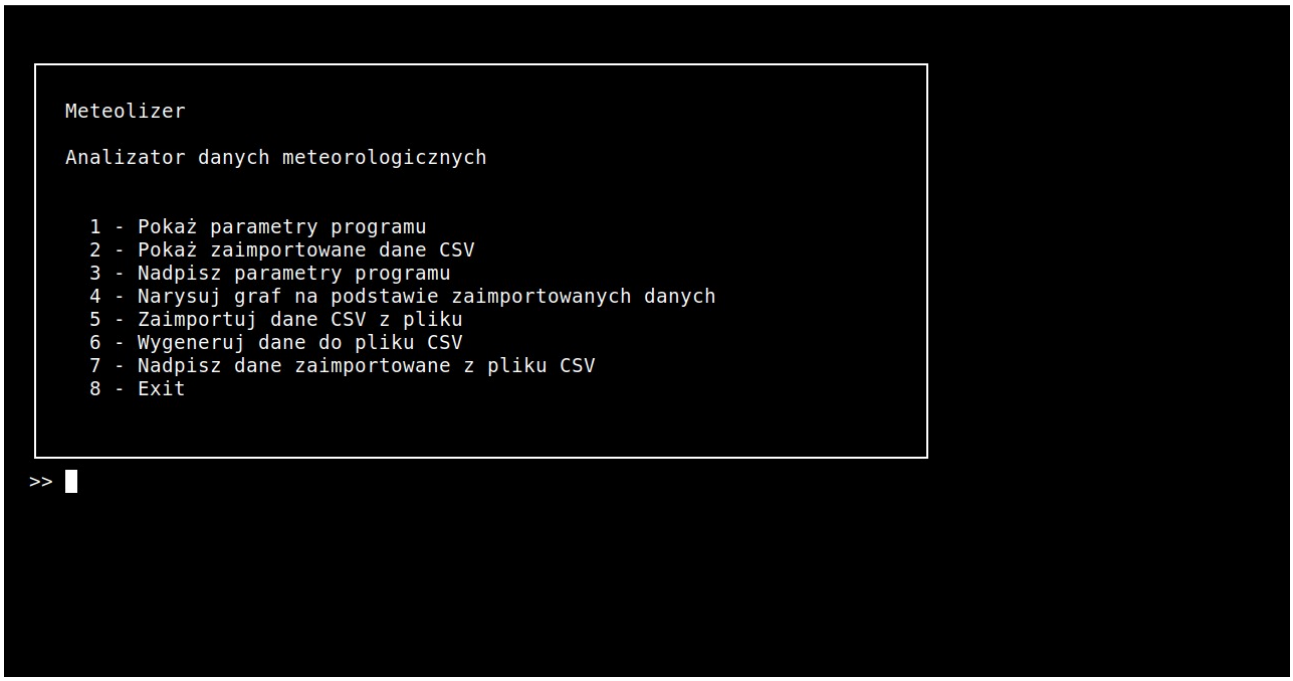
- Katalog `/` - Korzeń drzewa projektu

Plik `CHANGELOG.md` – Plik `git` służący do szerszej dokumentacji zmian w oprogramowaniu

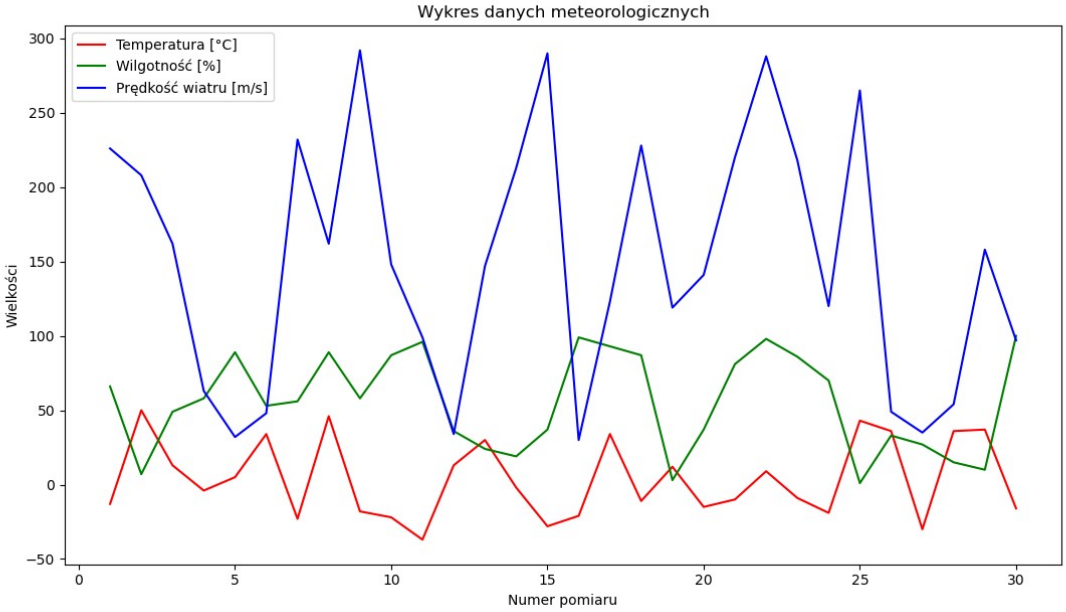
  - Plik `README.md` – Plik `git` służący do krótkiej dokumentacji oprogramowania jako ogółu
  - Plik `setup.cfg` – Plik biblioteki `flake8` zawierający jej konfigurację

- Katalog `/src` – Zawiera pliki kodu źródłowego
  - Plik `meteolizer.py` – Plik kodu źródłowego programu, który łączy funkcjonalności modułów i pakietów aplikacji
- Katalog `/misc` – Zawiera przykładowe dane wygenerowane przez program
- Katalog `/src/tests` – Zawiera moduły umożliwiające przeprowadzenie testów poszczególnych części oprogramowania
  - Plik `test_analysis.py` – Moduł zawierający zestaw testów poprawności działania modułu `/src/utils/analysis.py`
  - *Plik* `test_data_generator.py` *– Moduł zawierający zestaw testów poprawności działania modułu* `/src/utils/data_generator.py`
  - *Plik* `test_models.py` *– Moduł zawierający zestaw testów poprawności działania modułu* `/src/utils/models.py`
- Katalog `/src/utils` – Pakiet zawierający moduły pomocnicze programu
  - Plik `analysis.py` – Zawiera narzędzia służące do obliczania podstawowych statystyk
  - Plik `data_generator.py` – Zawiera narzędzia służące do generowania i importowania danych CSV z pliku
  - Plik `models.py` – Zawiera definicję klasy `Record` będącą kontenerem gromadzącym informacje na temat pomiaru
  - Plik `__init__.py` – Jego utworzenie daje możliwość utworzenia pakietu z modułami

Zrzuty ekranu działającej aplikacji



Menu główne



Wykres powstały w wyniku zaimportowania danych z pliku CSV i użycia biblioteki matplotlib

Krótka instrukcja uruchomienia

```
usage: Meteolizer [options]

Analizator danych meteorologicznych

options:
  -h, --help         show this help message and exit
  -n N               Liczba pomiarów
  --tmin TMIN        Minimalna wartość temperatury (w stopniach Celsjusza)
  --tmax TMAX        Maksymalna wartość temperatury (w stopniach Celsjusza)
  --hmin HMIN        Minimalna wartość wilgotności (w procentach)
  --hmax HMAX        Maksymalna wartość wilgotności (w procentach)
  --wmin WMIN        Minimalna prędkość wiatru (m/s)
  --wmax WMAX        Maksymalna prędkość wiatru (m/s)
  --csvfile CSVFILE  Nazwa pliku do eksportu danych CSV
  --csvdir CSVDIR    Ścieżka pliku do eksportu danych CSV

Wykonany przez: Wojciech Lis, Tomasz Kundera
```

Kod źródłowy pliku `/setup.cfg`

```
[flake8]
exclude = .git,.gitignore,*.pyc,*.png,*.jpg,*.csv
max-line-length = 119
```

Kod źródłowy pliku `/src/utils/analysis.py`

```
import functools
import math
import random
import numpy as np
from functools import reduce
from memory_profiler import profile
import timeit

def get_arithmetic_average(data):
    """
    Oblicz średnią arytmetyczną
    """

    length = len(data)

    dat_sum = reduce(lambda x, y: x + y, [ x for x in data ])

    try:
        res = dat_sum / length
    except ZeroDivisionError:
        print(f'Wykryto dzielenie przez zero')
    else:
        return res
```

```
# Algorytm quicksort do sortowania wyników
def partition(array, low, high):
    pivot = array[high].get('wartosc')

    i = low - 1

    j = low
    while j < high:
        if array[j].get('wartosc') <= pivot:
            i = i + 1
            (array[i], array[j]) = (array[j], array[i])
        j += 1

    (array[i + 1], array[high]) = (array[high], array[i + 1])
    return i + 1


def quick_sort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quick_sort(array, low, pi - 1)
        quick_sort(array, pi + 1, high)
```

```
def get_standard_deviation(data):
    """
    Oblicz odchylenie standardowe
    """

    deviation = 0
    res = 0
    length = len(data)

    avg = get_arithmetic_average(data)

    for i in range(length):
        deviation += ((i - avg) ** 2)

    try:
        deviation /= length-1
        res = math.sqrt(deviation)
    except ZeroDivisionError:
        print(f'Wykryto dzielenie przez zero')
    except:
        print(f'Wykryto wyjątek!')
    finally:
        return res

def get_celsius(fahrenheit):
    assert isinstance(fahrenheit, float), 'Typ podanej wartości się nie zgadza!'
    return fahrenheit * 33.8

def get_fahrenheit(celsius):
    assert isinstance(celsius, float), 'Typ podanej wartości się nie zgadza!'
    return celsius * -17.222222

def main():
  pass

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

Kod źródłowy pliku `/src/utils/data_generator.py`

```
#! /usr/bin/python3

import csv
import random
import datetime

def row_generator(n, temp_range, humidity_range, wind_range):
    """
    Wygeneruj pojedynczy wiersz danych meteorologicznych
    """
    temperature = random.randint(temp_range[0], temp_range[1])
    humidity = generate_humidity(humidity_range[0], humidity_range[1])
    wind_speed = generate_wind_speed(wind_range[0], wind_range[1])

    row = []

    row.append(n+1)  # pomiar
    row.append(temperature)
    row.append(humidity)
    row.append(wind_speed)

    return tuple(row)

def generate_humidity(minimum, maximum):
    """
    Generuj losową wilgotność procentową
    """
    return random.randint(minimum, maximum)

def generate_wind_speed(minimum, maximum):
    """
    Generuj siłę wiatru w m/s
    """
    return random.randint(minimum, maximum)

def file_data_generator(n, temp_range, humidity_range, wind_range):
    """
    Utwórz zawartość pliku z danymi meteorologicznymi
    """
    data = []

    for i in range(n):
        data_set = row_generator(i, temp_range, humidity_range, wind_range)
        data.append(data_set)
    return data

def data_writer(filepath, data):
    """
    Utwórz plik CSV i wpisz do niego dane meteorologiczne
    """
    with open(filepath, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(['pomiar', 'temperatura', 'wilgotnosc',
'predkosc_wiatru'])
        writer.writerows(data)
```

```
def data_reader(filepath):
    """
    Odczytaj plik CSV i zwróć jego zawartość w liście
    """
    file_content = []

    with open(filepath, 'r') as f:
        reader = csv.DictReader(f)

        for line in reader:
            line['pomiar'] = int(line['pomiar'])
            line['temperatura'] = int(line['temperatura'])
            line['wilgotnosc'] = int(line['wilgotnosc'])
            line['predkosc_wiatru'] = int(line['predkosc_wiatru'])
            file_content.append(line)

    return file_content

def main():
    # Definiujemy zakresy dla temperatury, wilgotności i wiatru
    temp_range = (20, 30)
    humidity_range = (30, 90)   # wilgotność od 30% do 90%
    wind_range = (0, 20)        # siła wiatru od 0 do 20 m/s

    data = file_data_generator(10, temp_range, humidity_range, wind_range)
    data_writer('dane.csv', data)

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

Kod źródłowy pliku */src/*`utils/models.py`

```
#!/usr/bin/python3

class Record:
    '''Pomiar'''
    class Station:
        '''Stacja badawcza'''
        # name: nazwa stacji
        # city: nazwa miasta
        # quantity: mierzona wielkość fizyczna (temperatura)
        def __init__(self):
            pass
        def set_name(self, new_name):
            self.name = new_name
        def set_city(self, new_city):
            self.city = new_city
        def set_quantity(self, new_quantity):
            self.quantity = new_quantity
        def get_name(self):
            return self.name
        def get_city(self):
            return self.city
        def get_quantity(self):
            return self.quantity
    def __init__(self, data):
        self.data = data.copy()
        self.station = self.Station()
    def set_station_name(self, new_name):
        self.station.set_name(new_name)
    def set_station_city(self, new_city):
        self.station.set_city(new_city)
    def set_station_quantity(self, new_quantity):
        self.station.set_quantity(new_quantity)
    def get_station_name(self):
        return self.station.get_name() if hasattr(self.station, 'name') else
None
    def get_station_city(self):
        return self.station.get_city() if hasattr(self.station, 'city') else
None
    def get_station_quantity(self):
        return self.station.get_quantity() if hasattr(self.station, 'quantity')
else None
    def set_data(self, new_data):
        self.data = new_data.copy()
    def get_data(self):
        return self.data.copy()

def main():
    data = [ 10, 20, 30 ]

    record = Record(data)

    record.set_station_name('Stacja 1')
    record.set_station_city('Kielce')
    record.set_station_quantity('Temperatura')

    print(record.get_station_name())

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

Kod źródłowy pliku */src/*`tests/test_analysis.py`

```
from memory_profiler import profile
import timeit
import sys
import random
import numpy as np.

sys.path.insert(0, '..')

import utils.analysis as asys


def quick_sort_perf_test():
    start = timeit.default_timer()
    for _ in range(10):
        data = [{ 'pomiar': 0, 'wartosc': -20.0 },{ 'pomiar': 1, 'wartosc': -
10.0 },
                { 'pomiar': 2, 'wartosc': -81.0 },{ 'pomiar': 3, 'wartosc': 24.0
},
                { 'pomiar': 4, 'wartosc': 2.0 },{ 'pomiar': 5, 'wartosc': 0.0 }]
        asys.quick_sort(data, 0, len(data)-1)
    stop = timeit.default_timer()
    print(asys.quick_sort.__name__ + ': Czas wykonania: ', stop - start)

@profile
def get_arthm_average_perf_test():
    start = timeit.default_timer()
    for _ in range(10):
        asys.get_arithmetic_average([ 10, 20, 30, 40, 50, 60, 70, 80, 90 ])
    stop = timeit.default_timer()
    print(asys.get_arithmetic_average.__name__ + ': Czas wykonania: ', stop -
start)

@profile
def get_std_deviation_perf_test():
    start = timeit.default_timer()
    for _ in range(10):
        asys.get_standard_deviation([ 10, 20, 30, 40, 50, 60, 70, 80, 90 ])
    stop = timeit.default_timer()
    print(asys.get_standard_deviation.__name__ + ': Czas wykonania: ', stop -
start)

def test_get_arithmetic_average():
    random_data = []

    def gen_random_data():
        nonlocal random_data

        for i in range(10):
            random_data.append(random.randint(-100, 100))

    gen_random_data()
    tmp_data = random_data.copy()

    avg1 = asys.get_arithmetic_average(random_data)
    avg2 = np.average(np.array(tmp_data))

    assert avg1 == avg2
```

```
def test_quick_sort():
    data = []

    def gen_random_data():
        nonlocal data

        for i in range(10):
            data.append( { 'pomiar': i, 'wartosc': random.randint(-100, 100) } )

    gen_random_data()

    tmp = data.copy()

    asys.quick_sort(data, 0, len(data)-1)

    tmp_sorted = sorted(tmp, key=lambda x: x['wartosc'])

    assert all(x == y for x, y in zip(data, tmp_sorted)) == True

@profile
def test_get_celsius():
    assert asys.get_celsius(1.0) == 33.8

@profile
def test_get_fahrenheit():
    assert int(asys.get_fahrenheit(1.0)) == -17


def main():
    quick_sort_perf_test()
    get_arthm_average_perf_test()
    get_std_deviation_perf_test()

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

Kod źródłowy pliku `/src/tests/test_data_generator.py`

```
#!/usr/bin/python3

from memory_profiler import profile
import timeit
import sys
import random
import numpy as np
sys.path.insert(0, '..')

import utils.data_generator as dgen

def main():
    pass

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

Kod źródłowy pliku */src/*tests/test_models.py

```
#!/usr/bin/python3

import unittest
import sys
from memory_profiler import profile
sys.path.insert(0, '..')

import utils.models as models

class test_model(unittest.TestCase):
    def setUp(self):
        self.record = models.Record(list())
        self.record.set_station_name('Stacja')
        self.record.set_station_city('Kielce')
        self.record.set_station_quantity('Temperatura')
    def test_get_station_name(self):
        self.assertEqual(self.record.get_station_name(), 'Stacja')
    def test_get_station_city(self):
        self.assertEqual(self.record.get_station_city(), 'Kielce')
    def test_get_station_quantity(self):
        self.assertEqual(self.record.get_station_quantity(), 'Temperatura')

if __name__ == '__main__':
    unittest.main()
```

Kod źródłowy pliku */src/*`meteolizer.py`

```
#!/usr/bin/python3

from consolemenu import *
from consolemenu.items import *

import matplotlib.pyplot as plt
import argparse
import numpy as np
import os

import utils.data_generator as datagen
import utils.analysis as asys
import utils.models as models

record = models.Record(list())

def main():
    def argparser():
        parser = argparse.ArgumentParser(
                prog='Meteolizer',
                description='Analizator danych meteorologicznych',
                epilog='Wykonany przez: Wojciech Lis, Tomasz Kundera',
                usage='%(prog)s [options]')

        parser.add_argument('-n', type=int, help='Liczba pomiarów', default=30)
        parser.add_argument('--tmin', type=int, help='Minimalna wartość temperatury (w stopniach Celsjusza)',
default=-40)
        parser.add_argument('--tmax', type=int, help='Maksymalna wartość temperatury (w stopniach Celsjusza)',
default=50)
        parser.add_argument('--hmin', type=int, help='Minimalna wartość wilgotności (w procentach)', default=0)
        parser.add_argument('--hmax', type=int, help='Maksymalna wartość wilgotności (w procentach)', default=100)
        parser.add_argument('--wmin', type=int, help='Minimalna prędkość wiatru (m/s)', default=30)
        parser.add_argument('--wmax', type=int, help='Maksymalna prędkość wiatru (m/s)', default=300)
        parser.add_argument('--csvfile', type=str, help='Nazwa pliku do eksportu danych CSV', default='dane.csv')
        parser.add_argument('--csvdir', type=str, help='Ścieżka pliku do eksportu danych CSV',
default=os.path.dirname(__file__))
        args = parser.parse_args()

        return args

    def console_menu(scr, args):
        def print_params(scr, args):
            scr.println('Parametry programu:')
            scr.println(f'Ilość pomiarów: {args.n}')
            scr.println(f'Minimalna temperatura: {args.tmin}')
            scr.println(f'Maksymalna temperatura: {args.tmax}')
            scr.println(f'Minimalna wilgotność: {args.hmin}')
            scr.println(f'Maksymalna wilgotność: {args.hmax}')
            scr.println(f'Maksymalna prędkość wiatru: {args.wmin}')
            scr.println(f'Maksymalna prędkość wiatru: {args.wmax}')
            scr.println(f'Domyślna ścieżka katalogu eksportu: {args.csvdir}')
            scr.println(f'Domyślna nazwa pliku eksportu: {args.csvfile}')
        def show_params_scr(scr, args):
            print_params(scr, args)
            scr.input('Kliknij Enter, aby wyjść')
        def import_data_scr(scr, args):
            global record

            try:
                data = datagen.data_reader(args.csvfile)
            except FileNotFoundError:
                scr.println(f'Nie znaleziono pliku {args.csvfile} w katalogu {args.csvdir}!')
                scr.input()
            except (TypeError, OSError, ValueError):
                scr.println(f'Błąd odczytu pliku {args.csvfile} w katalogu {args.csvdir}!')
            else:
                record.set_data(data)

                scr.println(f'Udało się zaimportować dane z pliku {args.csvfile}!')
                scr.input()
        def gen_data_scr(scr, args):
            data = datagen.file_data_generator(args.n,
                                               (args.tmin, args.tmax),
                                               (args.hmin, args.hmax),
                                               (args.wmin, args.wmax))

            try:
                datagen.data_writer(args.csvdir + '/' + args.csvfile, data)
            except OSError:
                scr.println(f'Wystąpił problem z zapisaniem danych do pliku {args.csvfile}')
            except:
                scr.println(f'Wystąpił wyjątek')
            else:
                scr.println(f'Udało się zapisać dane do pliku {args.csvfile} w katalogu {args.csvdir}!')
                scr.input()
```

```
        def draw_graph_scr(scr, args):
            global record

            try:
                data = record.get_data()

                if not data:
                    scr.println('Nie zaimportowano danych!')
                    scr.input()

                    raise ValueError('Nie zaimportowano danych!')
            except ValueError as e:
                print(e)
            else:
                num = [ nm['pomiar'] for nm in data ]
                temp = [ tp['temperatura'] for tp in data ]
                hum = [ hm['wilgotnosc'] for hm in data ]
                wind = [ wn['predkosc_wiatru'] for wn in data ]

                plt.plot(num, temp, color='r', label='Temperatura [°C]')
                plt.plot(num, hum, color='g', label='Wilgotność [%]')
                plt.plot(num, wind, color='b', label='Prędkość wiatru [m/s]')
                plt.legend()
                plt.xlabel('Numer pomiaru')
                plt.ylabel('Wielkości')
                plt.title('Wykres danych meteorologicznych')

                plt.show()

        def modify_data_scr(scr, args):
            global record

            def insert_num(scr, args):
                global record

                data = record.get_data()

                vmax = len(data)

                try:
                    user_value = int(scr.input('Podaj numer pomiaru, który chcesz zmodyfikować (od 1 do {vmax+1}):
'))

                    if user_value-1 < 0 or user_value-1 > vmax:
                        scr.println('Niepoprawny numer pomiaru')
                        raise ValueError(f'Niepoprawny number pomiaru! {user_value}')
                except TypeError:
                    print(e)
                else:
                    return user_value
            def insert_param(scr, args):
                global record
                chosen_key = ''

                try:
                    data = record.get_data()

                    if not data:
                        raise ValueError('Brak danych!')
                except:
                    scr.println('Wystąpił wyjątek!')
                else:
                    scr.println('Możliwe klucze:')

                    i = 1

                    available_keys = {}

                    for key in data[0].keys():
                        available_keys[str(i)] = key
                        scr.println(f'{i}. {key}')
                        i += 1

                    try:
                        user_value = int(input('Podaj numer klucza: '))

                        if user_value not in range(1, i):
                            scr.println(f'Podano niepoprawny numer klucza! {user_value}')
                            raise ValueError(f'Podano niepoprawny numer klucza! {user_value}')
                    except (TypeError, ValueError) as e:
                        scr.println(f'Wykryto wyjątek: {e}')
                    else:
                        chosen_key = available_keys[str(user_value)]
                        scr.println(f'Wybrano klucz {user_value} - {chosen_key}')
                        input()
                        return chosen_key

            data = record.get_data()

            try:
                idx = insert_num(scr, args)
```

```
                key = insert_param(scr, args)
            except (ValueError, TypeError) as e:
                scr.println(f'Wystąpił wyjątek! {e}')
            else:
                old_value = data[idx][key]

                old_val_type = type(old_value)

                try:
                    scr.println(f'Wybrana wartość: {old_value}, Typ: {old_val_type.__name__}')
                    new_value = old_val_type(scr.input('Podaj nową wartość: '))

                    if key == 'pomiar' or key == 'predkosc_wiatru':
                        if new_value < 0:
                            raise ValueError(f'Niepoprawna wartość: {new_value})')
                            scr.input()
                    if key == 'wilgotnosc':
                        if new_value < 0 or new_value > 100:
                            scr.println(f'Niepoprawna wartość: {new_value})')
                            scr.input()
                            raise ValueError(f'Niepoprawna wartość: {new_value})')

                except (TypeError, ValueError) as e:
                    scr.println(f'Wykryto wyjątek: {e}')
                    scr.input()
                else:
                    data[idx][key] = new_value
                    scr.println('Nadpisano dane!')
                    scr.input()
        def show_data_scr(scr, args):
            global record

            data = record.get_data()

            scr.println('Dane CSV')
            scr.println()

            for item in data:
                scr.println(f"| {item['pomiar']} | {item['temperatura']} | {item['wilgotnosc']} |
{item['predkosc_wiatru']} |")

            scr.input()


        def modify_params_scr(scr, args):
            def modify_n(scr, args):
                try:
                    new_n = int(scr.input('Podaj nową wartość liczby pomiarów: '))

                    if new_n <= 0:
                        raise ValueError(f'Zbyt mała liczba pomiarów: {new_n}')
                except (TypeError, ValueError) as e:
                    print(e)
                else:
                    args.n = new_n
                    scr.println(f'Nadpisano wartość liczby pomiarów: {args.n}')
                    scr.input('Kliknij Enter, aby wyjść')

            def modify_tmin(scr, args):
                try:
                    new_tmin = int(scr.input('Podaj nową wartość najmniejszej temperatury (w stopniach Celsjusza):
'))

                except (TypeError, ValueError) as e:
                    print(e)
                else:
                    args.tmin = new_tmin
                    scr.println(f'Nadpisano wartość liczby najmniejszej temperatury: {args.tmin}')
                    scr.input('Kliknij Enter, aby wyjść')
            def modify_tmax(scr, args):
                try:
                    new_tmax = int(scr.input('Podaj nową wartość największej temperatury (w stopniach Celsjusza):
'))

                except (TypeError, ValueError) as e:
                    print(e)
                else:
                    args.tmax = new_tmax
                    scr.println(f'Nadpisano wartość liczby największej temperatury: {args.tmax}')
                    scr.input('Kliknij Enter, aby wyjść')
            def modify_hmin(scr, args):
                try:
                    new_hmin = int(scr.input('Podaj nową wartość najmniejszej wilgotności (w procentach): '))

                    if new_hmin < 0 and new_hmin > 100:
                        raise ValueError('Niepoprawa wartość wilgotności: {new_hmin}%')

                except (TypeError, ValueError) as e:
                    print(e)
                else:
                    args.hmin = new_hmin
```

```
                    scr.println(f'Nadpisano wartość najmniejszej wilgotności: {args.hmin}%')
                    scr.input('Kliknij Enter, aby wyjść')
            def modify_hmax(scr, args):
                try:
                    new_hmax = int(scr.input('Podaj nową wartość największej wilgotności (w procentach): '))

                    if new_hmax < 0 and new_hmax > 100:
                        raise ValueError('Niepoprawa wartość wilgotności: {new_hmax}%')

                except (TypeError, ValueError) as e:
                    print(e)
                else:
                    args.hmax = new_hmax
                    scr.println(f'Nadpisano wartość największej wilgotności: {args.hmax}%')
                    scr.input('Kliknij Enter, aby wyjść')




def modify_wmin(scr, args):
                try:
                    new_wmin = int(scr.input('Podaj nową najmniejszą prędkość wiatru (w m/s): '))

                    if new_wmin < 0:
                        scr.println('Niepoprawna prędkość wiatru {new_wmin} m/s')
                        scr.input()
                        raise ValueError('Niepoprawa prędkość wiatru: {new_wmin} m/s')

                except (TypeError, ValueError) as e:
                    print(e)
                else:
                    args.wmin = new_wmin
                    scr.println(f'Nadpisano najmniejszą prędkość wiatru: {args.wmin} m/s')
                    scr.input('Kliknij Enter, aby wyjść')
            def modify_wmax(scr, args):
                try:
                    new_wmax = int(scr.input('Podaj nową największą prędkość wiatru (w m/s): '))

                    if new_wmax < 0:
                        raise ValueError('Niepoprawa prędkość wiatru: {new_wmax} m/s')

                except (TypeError, ValueError) as e:
                    print(e)
                else:
                    args.wmax = new_wmax
                    scr.println(f'Nadpisano największą prędkość wiatru: {args.wmax} m/s')
                    scr.input('Kliknij Enter, aby wyjść')
            def modify_csvdir(scr, args):
                try:
                    new_csvdir = scr.input(f"Podaj nazwę nowej ścieżki do katalogu eksportu (bez
{os.getenv('HOME')}): ")

                    full_csvdir = os.getenv('HOME') + '/' + new_csvdir

                    if not os.path.exists(full_csvdir):
                        raise ValueError(f'Podana ścieżka do pliku nie istnieje: {full_csvdir}')
                except (OSError) as e:
                    print(e)
                else:
                    args.csvdir = full_csvdir
                    scr.println(f'Nadpisano domyślną ścieżkę do katalogu eksportu: {args.csvdir}')
                    scr.input('Kliknij Enter, aby wyjść')
            def modify_csvfile(scr, args):
                try:
                    new_csvfile = scr.input(f'Podaj nową nazwę pliku eksportu: ')

                except (OSError) as e:
                    print(e)
                else:
                    args.csvfile = new_csvfile
                    scr.println(f'Nadpisano nazwę pliku eksportu: {args.csvfile}')
                    scr.input('Kliknij Enter, aby wyjść')

            params_submenu = ConsoleMenu('Zmień ustawienia parametrów')

            params_n = FunctionItem('Zmień liczbę pomiarów', modify_n, [scr, args])
            params_tmin = FunctionItem('Zmień najmniejszą wartość temperatury', modify_tmin, [scr, args])
            params_tmax = FunctionItem('Zmień największą wartość temperatury', modify_tmax, [scr, args])
            params_hmin = FunctionItem('Zmień najmniejszą wartość wilgotności', modify_hmin, [scr, args])
            params_hmax = FunctionItem('Zmień największą wartość wilgotności', modify_hmax, [scr, args])
            params_wmin = FunctionItem('Zmień najmniejszą prędkość wiatru', modify_wmin, [scr, args])
            params_wmax = FunctionItem('Zmień największą prędkość wiatru', modify_wmax, [scr, args])
            params_csvfile = FunctionItem('Zmień domyślną nazwę pliku eksportu', modify_csvfile, [scr, args])
            params_csvdir = FunctionItem('Zmień domyślną ścieżkę katalogu eksportu', modify_csvdir, [scr, args])

            params_submenu.append_item(params_n)
            params_submenu.append_item(params_tmin)
            params_submenu.append_item(params_tmax)
            params_submenu.append_item(params_hmin)
```

```
            params_submenu.append_item(params_hmax)
            params_submenu.append_item(params_wmin)
            params_submenu.append_item(params_wmax)
            params_submenu.append_item(params_csvfile)
            params_submenu.append_item(params_csvdir)

            params_submenu.show()


        menu = ConsoleMenu('Meteolizer', 'Analizator danych meteorologicznych', screen=scr)

        show_params = FunctionItem("Pokaż parametry programu", show_params_scr, [scr, args])
        mod_params = FunctionItem('Nadpisz parametry programu', modify_params_scr, [scr, args])
        draw_graph = FunctionItem('Narysuj graf na podstawie zaimportowanych danych', draw_graph_scr, [scr, args])
        import_data = FunctionItem('Zaimportuj dane CSV z pliku', import_data_scr, [scr, args])
        gen_data = FunctionItem('Wygeneruj dane do pliku CSV', gen_data_scr, [scr, args])
        modify_data = FunctionItem('Nadpisz dane zaimportowane z pliku CSV', modify_data_scr, [scr, args])
        show_data = FunctionItem('Pokaż zaimportowane dane CSV', show_data_scr, [scr, args])

        menu.append_item(show_params)
        menu.append_item(show_data)
        menu.append_item(mod_params)
        menu.append_item(draw_graph)
        menu.append_item(import_data)
        menu.append_item(gen_data)
        menu.append_item(modify_data)

        menu.show()


    scr = Screen()
    args = argparser()
    console_menu(scr, args)


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

Krótkie omówienie działania niektórych funkcji w pliku `meteolizer.py`

Funkcja argparser()

Funkcja `argparser()` tworzy parser argumentów wiersza poleceń, pozwalający użytkownikowi na:

- Ustalenie liczby pomiarów (-n)
- Określenie zakresów temperatury (--tmin, --tmax)
- Zakresów wilgotności (--hmin, --hmax)
- Zakresów prędkości wiatru (--wmin, --wmax)
- Nazwy i ścieżki pliku CSV do eksportu/importu (--csvfile, --csvdir)
- Metoda parse_args() odczytuje i zwraca te parametry w obiekcie args

Interfejs użytkownika (menu)

- Funkcja console_menu() tworzy menu tekstowe za pomocą biblioteki consolemenu
- Menu zawiera kilka opcji, które wywołują funkcje obsługi
  - Wyświetlanie parametrów (show_params_scr)
  - Wczytywanie danych z pliku CSV (import_data_scr)
  - Generowanie danych i zapis do pliku CSV (gen_data_scr)
  - Rysowanie wykresów danych (draw_graph_scr)
  - Edycja danych (modify_data_scr)
  - Wyświetlanie danych (show_data_scr)
  - Modyfikacja parametrów programu (modify_params_scr)

Wnioski

Powyższy projekt pozwolił nam na utrwalenie wiedzy z całego zakresu materiału z języków skryptowych. Szczególnie ważnym elementem tworzenie aplikacji było zastosowanie niestandardowych bibliotek programistycznych takich jak `matplotlib`