University of Warsaw

Faculty of Economic Sciences

Grzegorz Krochmal

Student number: 372412

**Pollution Stats in Poland – Plotting Engine**

Final project of Python&SQL: intro/SQL platforms

Warsaw, January 2019

Zawartość

[Introduction - Overview of the Project 2](#_Toc535083498)

[Chapter 1 App’s Software Background 3](#_Toc535083499)

[1.1. Necessary Python Libraries 3](#_Toc535083501)

[1.2. Connection with GIOŚ’s API and establishing the database 4](#_Toc535083502)

[1.2.1. Connection with GIOŚ’s API 4](#_Toc535083503)

[1.2.2. Establishing the database 4](#_Toc535083504)

[1.2.2. Creating the final table with all data and updating it 4](#_Toc535083505)

[1.3. Database’s Entity Relationship Diagram 5](#_Toc535083506)

[1.4. Program’s py files and their tasks 6](#_Toc535083507)

[1.5. Java Script, HTML and CSS 6](#_Toc535083508)

[1.5.1. Java Script 6](#_Toc535083509)

[1.5.2. HTML and CSS 7](#_Toc535083510)

[Chapter II App’s manual with examples 8](#_Toc535083511)

[2. 1. Main page 8](#_Toc535083513)

[2.2. Plot page 9](#_Toc535083514)

[2.3. Possible scenarios 10](#_Toc535083515)

[2.3.1. Day date 11](#_Toc535083516)

[2.3.2. Last 3 days or Last 7 days 12](#_Toc535083517)

[2.3.3. Error page 13](#_Toc535083518)

# Introduction - Overview of the Project

The main idea to this app is to provide a fast and comfortable tool for getting and visualizing basic pollution stats from Poland. The program uses GIOŚ’S API, which is Polish Environmental Protection Agency, to gather pollution data. GIOŚ itself offers a complex tool

to check state of air pollution in Poland (here is the link to it:

<http://powietrze.gios.gov.pl/pjp/current>). Although the tool can provide us with all the statistic we would want to get about air pollution, it is really hard to use it. Choosing a measuring station is a nightmare, because you have to do it with an interactive map, which is not as interactive as it should be and has a problem with calibration.

The app I created is free of mentioned problems. The interface is meant to be simple. All the parameters, which have to be set to provide you with proper pollution stats, you choose with cascade[[1]](#footnote-1) select fields. The program is based on Python and SQL.

It is all it has to be said at the beginning. All specified information about the project are included in the next chapters. This documentation covers following issues.

1. Chapter 1 – App’s software background

* Necessary Python libraries
* Connection with GIOŚ’s API and establishing the database
* Database’s Entity Relationship Diagram
* Program’s py files and their tasks
* Java Script, HTML and CSS

2. Chapter 2 – App’s manual with examples

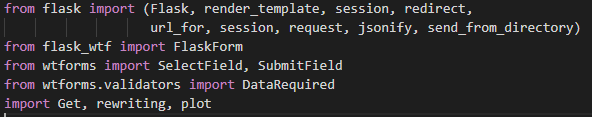
# Chapter 1

# App’s Software Background

## 1.1. Necessary Python Libraries

The whole app bases on few py files responsible for different operations. Here the necessary libraries are shown divided into py files they are in. The deeper description what are the tasks of mentioned files is included in 1.4.

* Main.py



* Get.py



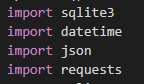
* plot.py



* rewriting.py



* uptade\_new.py



## 1.2. Connection with GIOŚ’s API and establishing the database

### 1.2.1. Connection with GIOŚ’s API

GIOŚ with its API gives an opportunity to gather air pollution data from Poland. The APIs are easy to access on page: <http://powietrze.gios.gov.pl/pjp/content/api>. The data is provided in JSON format. To create database for this App, following APIs were used:

* <http://api.gios.gov.pl/pjp-api/rest/station/findAll>
* [http://api.gios.gov.pl/pjp-api/rest/station/sensors/{stationId}](http://api.gios.gov.pl/pjp-api/rest/station/sensors/%7bstationId%7d)
* http://api.gios.gov.pl/pjp-api/rest/data/getData/{sensorId}

The most important of those is the last one, because it is where the pollution stats are included, but those links work in a cascade way. It is impossible to have a sensorId without using a second link, and there is no option to gather stationId without the first API.

### 1.2.2. Establishing the database

The code which is responsible for storing the API’s data into an App’s database is included in *rewriting.py* file. There are python functions which creates separated tables in database:

* MeasuringStation() uses first API and creates a table measuring\_station
* SensorsID() uses second API and creates a table senors\_data

For the last one API the case is more complicated. It is because in most cases in every city there are few stations and every station has several sensors. So the effect is that to get data from all sensor there are more than 600 separated APIs which the program needs to connect with. This is handled with the update\_new.py file.

### 1.2.2. Creating the final table with all data and updating it

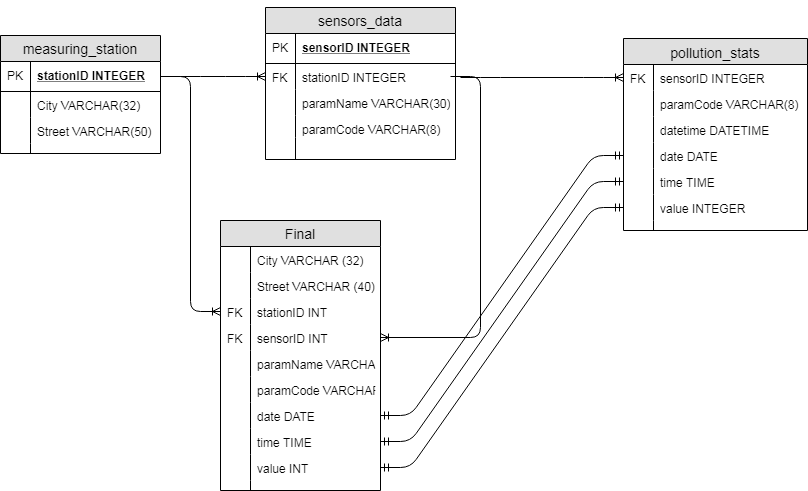
As it was said before, to gather all pollution data for all the measuring stations the program has to run more than 600 APIs. It makes it impossible for the real-time uptade in App. The concession had to be made. The update\_new.py file is this concession. It contains UpdateDB() function which verifies if yesterday’s date is in the final table (containing all pollution stats). If it is not, it runs the connection with all those more than 600 APIs and create a “scratchpad” table pollution\_stats which is cleared as soon the data is added to the final table, for which function FinalStats() from rewriting is responsible.

The precise structure of database is described in chapter 1.3.

## 1.3. Database’s Entity Relationship Diagram

The App’s database contains 4 tables: measuring\_station, sensors\_data, pollution\_stats and final. What is important is that table pollution\_stats does not keep data. It is only used in the process of collecting data after connection with GIOŚ’s API. When the pollution stats are inserted into this table it is joined with measuring\_station and sensors\_data and after that the final data is inserted into table final. Afterwards pollution\_stats is cleared automatically.

The relationship diagram of database is shown below:



## 1.4. Program’s py files and their tasks

Names and functionalities of some of py files might have been described in previous chapters when it was necessary. This chapter contains a short overview of tasks which are handled by all specified files.

* Main.py – That is the main App file. It is based on Flask. Main.py integrates all the tasks done in other py files with Flask and then with Java Script, HTML and CSS.
* rewriting.py - This script is responsible for creating tables in database (measuring\_stations, sensors\_data and final).
* Get.py – It contains different functions which gather specified data from the database.
* update\_new.py – This script verifies if yesterday’s date is in final table. If it is not it launches updating process. As it was said in 1.2.2. such an approach was necessary, because real-time update of pollution stats would make the App unusable in the effect of long time which is necessary to establish connections with all the APIs. After update process the script also verifies the length of database. If number of distinct dates is greater than 30, it drops the oldest one.
* plot.py – It uses matplotlib to provide the user with a plot based on parameters the user selected. It can draw two plots, depending on user’s choice.
  + Average Plot – Which is a barplot
  + Hourly Plot – Which is a lineplot

Both plots include safe limit line, which shows the safe for human limit of selected pollutant’s level. The levels were drawn from:

http://powietrze.gios.gov.pl/pjp/content/annual\_assessment\_air\_acceptable\_level

## 1.5. Java Script, HTML and CSS

### 1.5.1. Java Script

The main page contains cascade select fields. City’s choice indicates available streets, street’s choice indicates available pollutants and date’s choice indicates available data types (plot types). To achieve that goal I had to introduce to my App some Java Script. The Java Script code is placed in index.html.

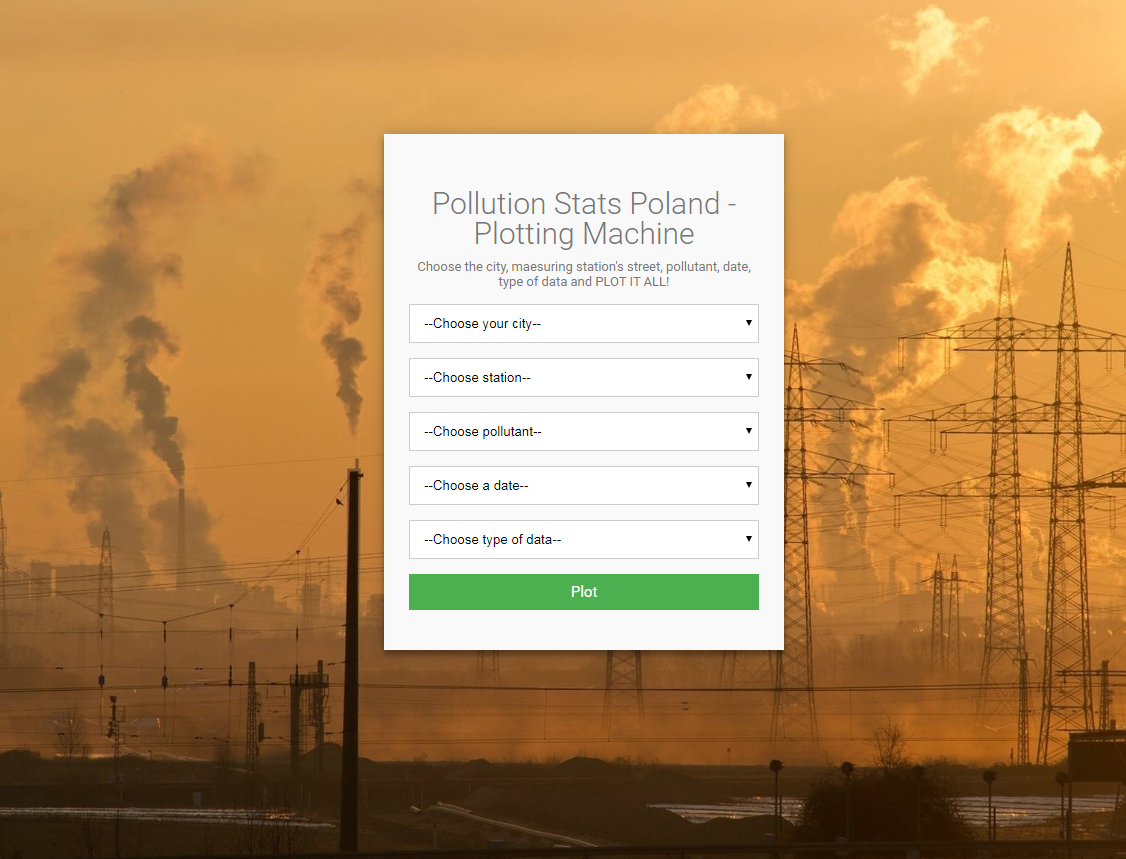
### 1.5.2. HTML and CSS

The visual side of project is supported by HTML and CSS. For HTML there are 3 templates, accordingly: index, plot, error. For CSS the names are: master, plot, error.

# Chapter II

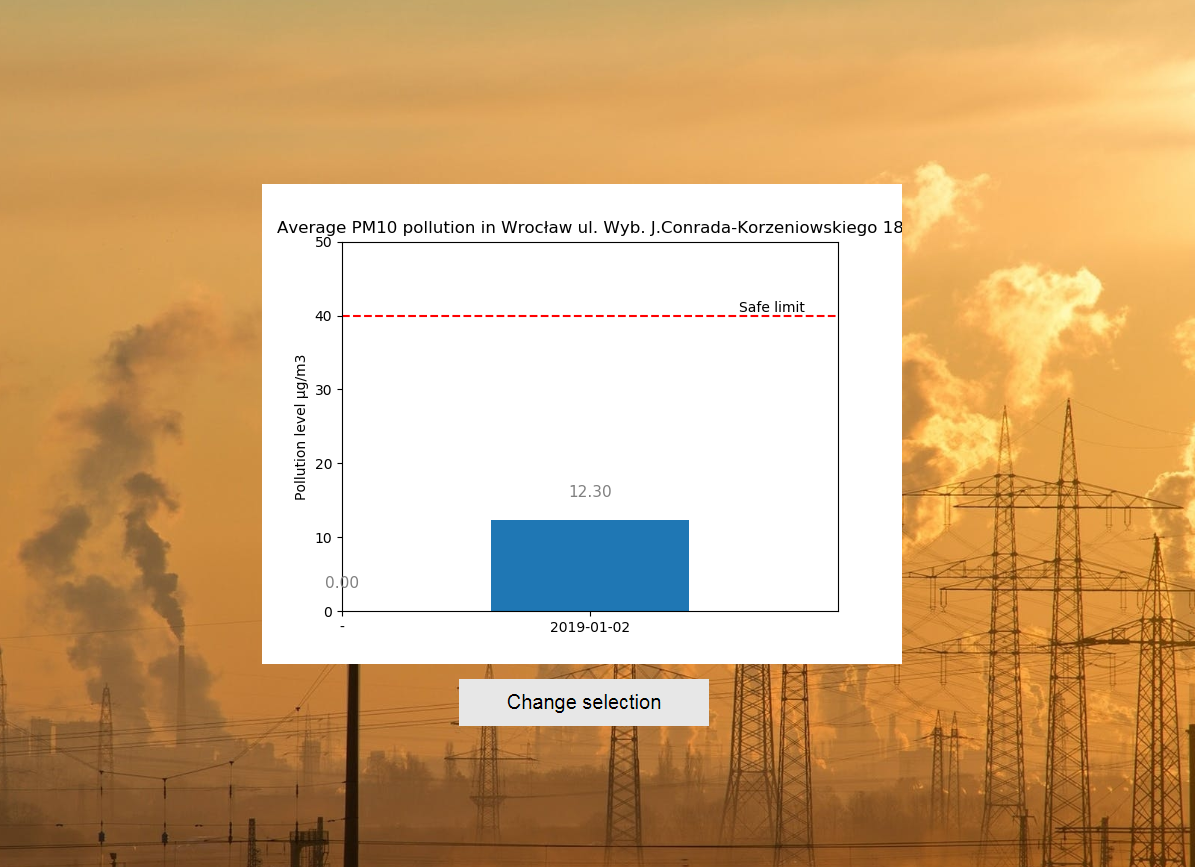
# App’s manual with examples

## 2. 1. Main page

****

After loading an App that is the opening page. Here you can select for which city, station, pollutant, date and type of data you want to create a plot. Those are all select fields, so the only thing you have to expand the list and make a choice. However the choice fields are cascade fields, it means that you have to make selections in order. The station (the street where the station is located) will not occur until you choose a city. The pollutant can’t be chosen without choosing a station and type of data will not be available without choosing a date. When all fields are selected just click Plot button it will return you a proper plot.

## 2.2. Plot page

****

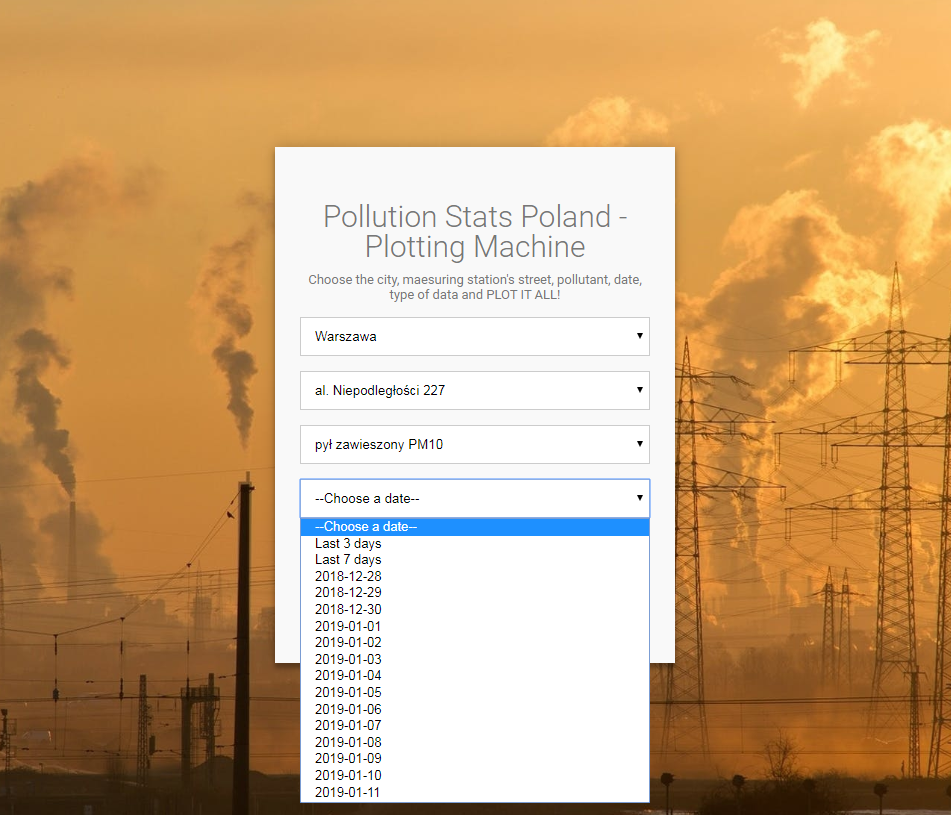
That is an example look of a plot page. It is simple. You only get the plot based on your choices made on the main page. If you want to get another plot just click change selection. Do not click go back to the previous page button of your browser, because the main page will not load properly.

Plot of each pollutant contains a “Safe limit” line which indicates the maximum level of pollution which is supposed not to be harmful for a human being. All the limits were taken from: <http://powietrze.gios.gov.pl/pjp/content/annual_assessment_air_acceptable_level>

## 2.3. Possible scenarios

Possible scenarios depend on your choice of date because it indicates the type of data (type of plot) which is available for specified selection.

The options of date are:

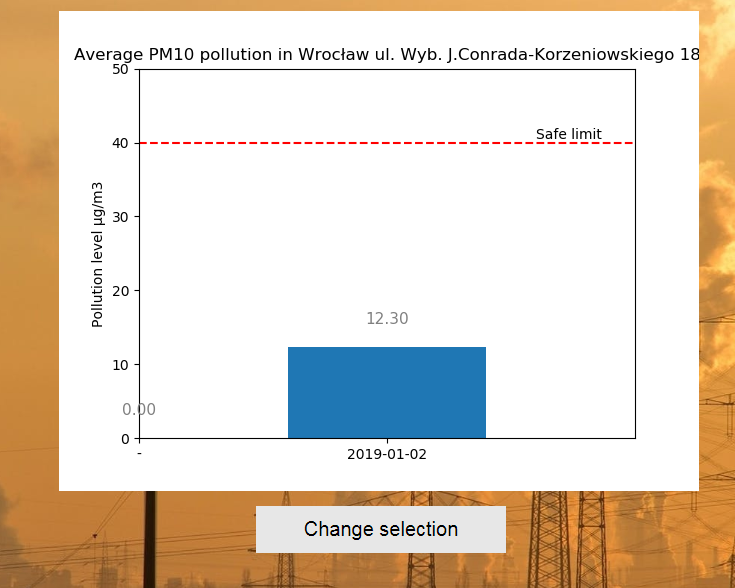


### 2.3.1. Day date

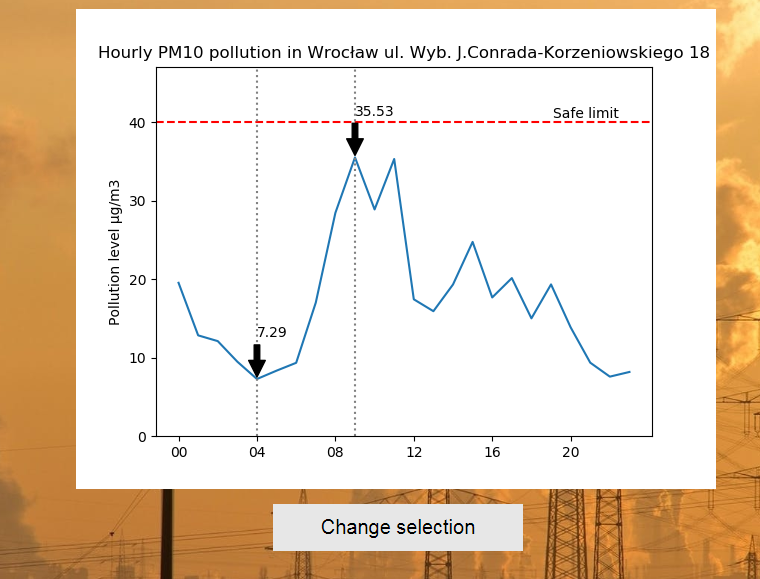
****

If you have chosen a day date (year-month-day) there are 2 types of plot you can get. The average or hourly one.

1. Average



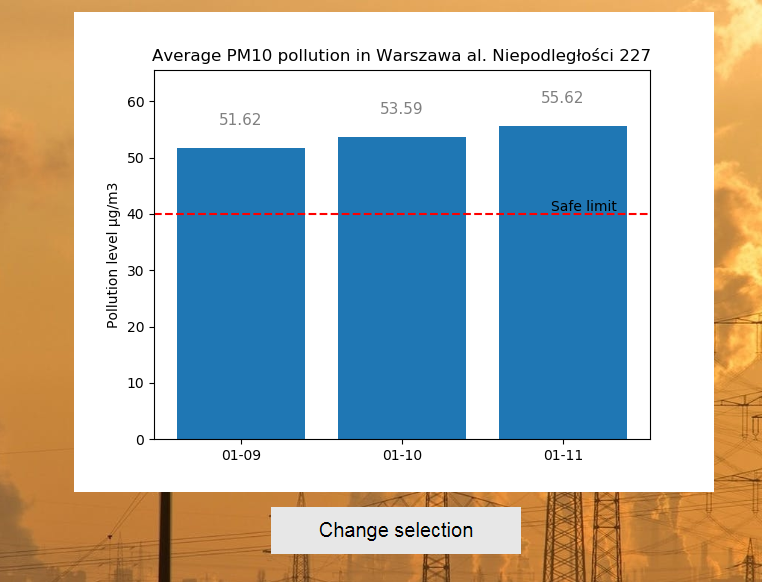
1. Hourly



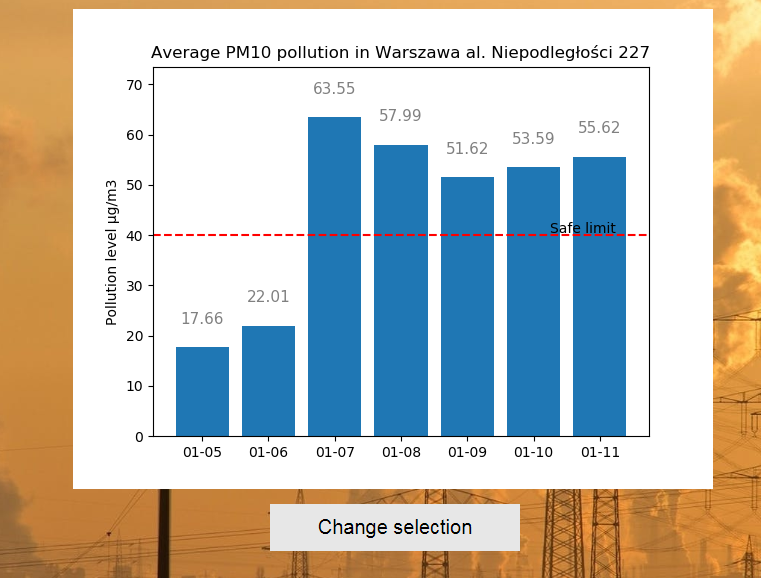
### 2.3.2. Last 3 days or Last 7 days

If you in date select field you have chosen Last 3 days or Last 7 days the only type of data available for you is average one. It is how it looks like

1. Last 3 days

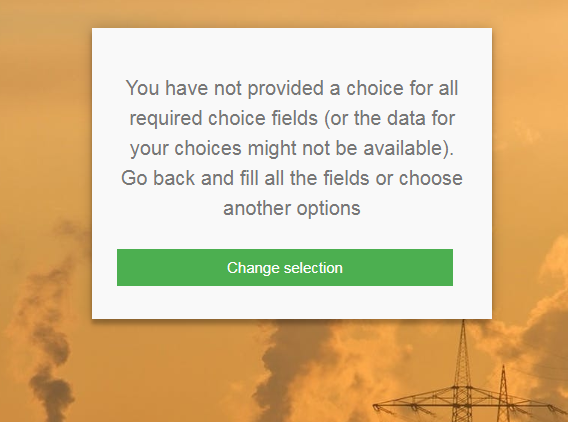


1. Last 7 days



### 2.3.3. Error page

If you have not made all choices in select fields and still clicked plot button you will get an error page. The error page also can appear when there is no available data for selections you made. It does not happen often but it might occur.



With change selection button you can come back to main page and correct your choices.

1. Cascade means that selection from one field determines available options in the next one. [↑](#footnote-ref-1)