

Geo Informatics Final Project : Group B

M-IE_2.02 Geoinformatics, WS2022/23

Under supervision of : Prof. Rolf Becker

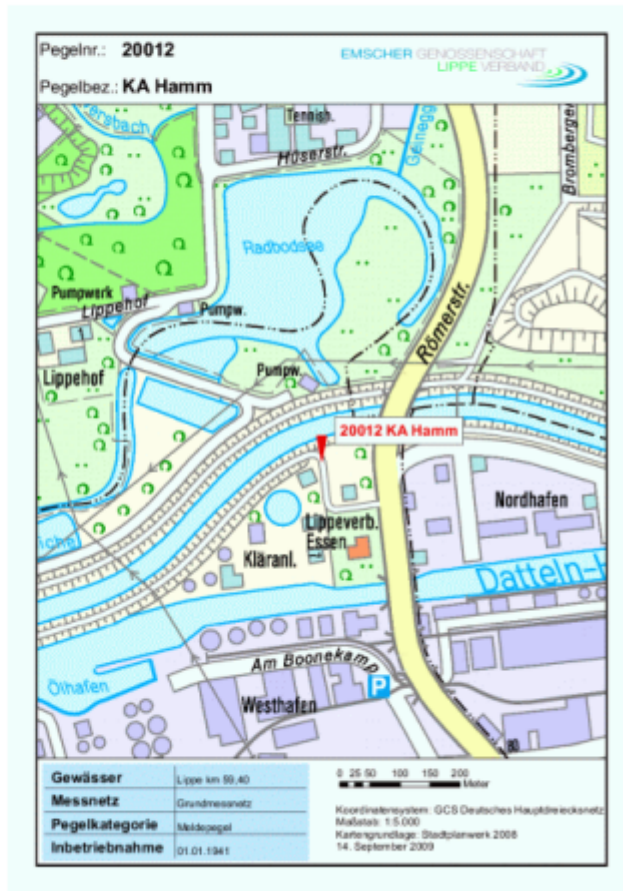
Participants:

Sindhya Babu (29928), Kiara Meço (32358), Sahil Chande (29927) ===== Introduction ===== The Emscher Gennosenschaft Lippe Verband provides open raw data on the water level and discharge with daily updated values of the Emscher and Lippe area. The data is updated approximately every 15 minutes and two versions which is intranet (for registered users) and public versions are published. In our project, we have used open public data. ===== Project aim ===== The project aim is to scrape time-varying data on Water level and discharge from the website of Übersichtskarte Pegelstände Emscher Lippe continuously using the python library beautiful soup and also geo pandas and save them to the PostgreSQL database. Then, we have geo-referenced five Pegel (or Water) stations on the map using QGIS along with plotting the stations precise location on the map. https://howis.eglv.de/pegel/html/uebersicht_internet.php ===== Tools and packages used ===== * Python * For web-scraping: BeautifulSoup, pandas, numpy, and requests. * For the creation of geo data frame: geoPandas, pyproj, shapely.geometry. * For database connection to PostgreSQL: sqlalchemy, psycopg2. * PostgreSQL * Database to store data and geometry * Pg Admin 4 and PostGIS extension for PostgreSQL * UI for easier operations with PostgreSQL * QGIS * Application used for plotting different graphs, maps and georeferencing the stations to their precise locations. ===== One-Time Scraping of Master Data of the gauges ("Pegelstammdaten") ===== The Base data (Stammdaten in German) provided contains information such as Station number (Pegelnummer), Water body if it is either Lippe or Emscher (Gewässer), River length (Flusskilometer) in km, Level zero-point(Pegelnulpunkt), above sea-level in mNN, total Catchment area (Einzugsgebiet) in km², Easting (Rechtswert) and Northing (Hochwert) Gauss Krüger co-ordinates with Mean High Water level (MHW) in cm, Mean Lowest Water level (MNW) in cm and Medium Water level (MW) in cm for the periods from 2001 to 2010. In addition to this, the image of the Pegel Station and the map showing the location of the Pegel station is displayed. Firstly, we scrape the text displayed for the Pegel station and also the corresponding map for each station and store it locally. The below image shows an example of the Master data for Station KA Hamm.

Stammdaten : KA Hamm

zurück

Pegelnummer:	20012
Gewässer:	Lippe
Flusskilometer (km):	59.40
Pegelnulppunkt (m+NN):	50.13
Einzugsgebiet (km ²):	2656.00
Rechtswert (Gauss-Krüger):	2622828.00
Hochwert (Gauss-Krüger):	5728949.00
(2001 - 2010) MHW (cm):	432
(2001 - 2010) MW (cm):	337
(2001 - 2010) MNW (cm):	314

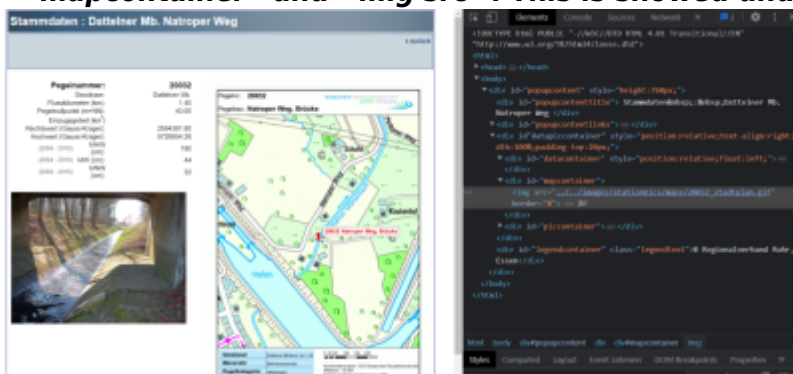


© Regionalverband Ruhr, Essen Figure 1:

Pegel Master data of station KA Hamm

https://howis.eglv.de/pegel/html/stammdaten_html/MO_StammdatenPegel.php?PIDVal=32

To determine the above-mentioned values for all the Stations, we scrape the website using Python, BeautifulSoup package. We loop over 200 PIDVal to get the master data of all the stations possible. To achieve this, the text stored under the HTML tags needs to be identified by inspecting the web page. Consider the example of Station KA Hamm, where it can be seen that the master data text is stored under `<div id = " datacontainer"` and `<tr class = " normtext"` html tags. The name of the station is contained in `<div id = " popupcontenttitle"` and the map image is however stored under the tag `<div id = " mapcontainer"` and `<img src =`. This is showed under Figure 1.



```

<div id="datacontainer" style="position:relative;float:left;">
  <table class="normtext" border="0">
    <tbody>
      <tr class="normtext">
        <td width="180" colspan="2"> </td>
        <td width="160"> </td>
      </tr>
      <tr class="normtext">
        <td colspan="2">Gewässer:</td>
        <td>Datteiner Mb.</td>
      </tr>
      <tr class="normtext">
        <td colspan="2">Flusskilometer (km): </td>
        <td>1.45</td>
      </tr>
      <tr class="normtext">
        <td colspan="2">Pegelnullpunkt (m+NN):</td>
        <td>43.00</td>
      </tr>
      <tr class="normtext"> </tr>
      <tr class="normtext"> </tr>
      <tr class="normtext"> </tr>
      <!--
      <tr class="normtext">
        <td>&nbsp;</td>
        <td>H&nbsp;</td>
      </tr>
    </tbody>
  </table>

```

Figure 2: Inspecting source code to determine html tags to be extracted. **The data extracted for one station is shown below. The data frame contains two values 'Station' and 'Station Values'. The Station Values column is then split to several columns and renamed and stored as a new data frame.**

```

Stammdaten = []
# Print Station Name
Station_Name = soup.find('div', {'id':'popupcontenttitle'}).text.split()[-1]

# Print only the values of stammdaten
Station_Values = soup.find_all('tr', {'class':'normtext'})

Values = []

for i in Station_Values:
    i = i.text.replace('\n', ' ').split()[-1]
    Values.append(i)

Stammdaten.append({'Station': Station_Name, 'Values': Values})
df = pd.DataFrame(Stammdaten).replace('\n', '')

df

  Station  Values
0 Hamm [20012, Lippe, 59.40, 50.13, 2656.00, 2622828...

df
  Station  Values
0 Hamm [20012, Lippe, 59.40, 50.13, 2656.00, 2622828...

split = pd.DataFrame(df['Values'].to_list(), columns = ['Pegelnummer', 'Gewässer', 'Flusskilometer_(km)', 'Pegelnul
df = pd.concat([df, split], axis=1)
df = df.drop('Values', axis=1)
df

  Station  Pegelnummer  Gewässer  Flusskilometer_(km)  Pegelnullpunkt_(m+NN)  Einzugsgebiet_(km²)  Rechtswert_(Gauss-Krüger)  Hochwert_(Gauss-Krüger)  MHW_(m)  MW_(m)
0 Hamm            20012    Lippe            59.40            50.13            2656.00            2622828.00            5728949.00            432

```

Figure 3: Python code showing extracting text of station name and values for KA Hamm **After looping over, we found that several PIDVal contained no data. We drop these rows with no data and now store the new data frame with non-null values. The new data frame contains 131 Stations and only 103 stations had geo-coordinates data available as shown under figure 4.**

```

In [24]: M df.info() # now df contains correct data type except pegel Number

<class 'pandas.core.frame.DataFrame'>
Int64Index: 131 entries, 1 to 168
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ---                ---
 0   Station                131 non-null   object
 1   Pegelnummer            131 non-null   object
 2   Gewässer               131 non-null   object
 3   Flusskilometer_(km)   131 non-null   float64
 4   Pegelnullpunkt_(m+NN) 131 non-null   float64
 5   Einzugsgebiet_(km²)    11 non-null    float64
 6   Rechtswert_(Gauss-Krüger) 103 non-null   float64
 7   Hochwert_(Gauss-Krüger) 103 non-null   float64
 8   MNW_(cm)              45 non-null    float64
 9   MW_(cm)               45 non-null    float64
10  MNW_(cm)              45 non-null    float64
dtypes: float64(8), object(3)
memory usage: 12.3+ KB

M %time
gdf = gpd.GeoDataFrame(df, geometry=gpd.points_from_xy(df['rechtswert_(Gauss-Krüger)'], df['hochwert_(Gauss-Krüger)']), crs=
# creating geo data frame , passing co-ordinates of Gauss Krüger Easting and Northing Values as x,y points with EPSG 31466

CPU times: total: 0 ms
Wall time: 18.9 ms

M gdf.info() # checking once for geometry data type created

<class 'geopandas.geodataframe.GeoDataFrame'>
Int64Index: 131 entries, 1 to 168
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ---                ---
 0   Station                131 non-null   object
 1   Pegelnummer            131 non-null   int64
 2   Gewässer               131 non-null   object
 3   Flusskilometer_(km)   131 non-null   float64
 4   Pegelnullpunkt_(m+NN) 131 non-null   float64
 5   Einzugsgebiet_(km²)    11 non-null    float64
 6   Rechtswert_(Gauss-Krüger) 103 non-null   float64
 7   Hochwert_(Gauss-Krüger) 103 non-null   float64
 8   MNW_(cm)              45 non-null    float64
 9   MW_(cm)               45 non-null    float64
10  MNW_(cm)              45 non-null    float64
11  geometry               131 non-null   geometry
dtypes: float64(8), geometry(1), int64(1), object(2)
memory usage: 13.3+ KB

```

Figure 4: Data frame showing the data types and a number of non-null column values.

The geo-coordinates values of Rechtswert and Hochwert stored in the above data frame are still of data type float63. Since we want our coordinates to be recognized as geographic location data, we use the geoPandas package in python, to convert the pandas data frame to a geo data frame or gdf. Since a geo data frame requires a shapely object, we pass the columns containing Easting and Northing values i.e Rechtswert_(Gauss-Krüger), Hochwert_(Gauss-Krüger) respectively into the function points_from_xy to transform it to shapely points. The below figure shows an example of what geo data frame, gdf looks like.

```

M gdf # prints geo data frame created

In [5]:

```

xr	Flusskilometer_(km)	Pegelnullpunkt_(m+NN)	Einzugsgebiet_(km²)	Rechtswert_(Gauss-Krüger)	Hochwert_(Gauss-Krüger)	MNW_(cm)	MW_(cm)	MNW_(cm)	geometry
10	1.40	31.48	NaN	2580184.00	5707172.00	NaN	NaN	NaN	POINT (2580184.000 5707172.000)
10	2.83	32.91	NaN	2587323.00	5706410.00	NaN	NaN	NaN	POINT (2587323.000 5706410.000)
b.	0.85	34.22	NaN	2587540.00	5705975.00	NaN	NaN	NaN	POINT (2587540.000 5705975.000)
b.	1.79	37.10	NaN	2587629.03	5704851.75	244.0	47.0	37.0	POINT (2587629.030 5704851.750)
b.	2.20	37.18	NaN	2587554.00	5704540.00	NaN	NaN	NaN	POINT (2587554.000 5704540.000)
...
25	1.74	0.00	NaN	2587355.90	5724144.20	NaN	NaN	NaN	POINT (2587355.900 5724144.200)
25	0.22	0.00	NaN	NaN	NaN	NaN	NaN	NaN	POINT EMPTY
b.	0.80	0.00	NaN	NaN	NaN	NaN	NaN	NaN	POINT EMPTY

Figure 5: Geo data frame containing geometry column as shapely points

==== Storing the master data of Water Stations in PostgreSQL database ==== We create a database env_db and a new schema named 'eglv' is created under the database using super user env_master. Under this schema, we create a table 'eglv_stations' and upload the geo data frame to the table 'eglv_stations'. The connection to the PostGIS database from python is enabled by creating a connection engine using sqlalchemy package and we pass this connection engine to_postgis. With chunksize=100, 100 rows will be written at a time to the database. This is shown under figure 6,7. But since the data frame contains only 131 rows, chunksize does not play a significant role when compared to data base with larger values.

```
[43]: X3time
gdf.to_postgis(con=engine, name="eglv_stations", schema="eglv", index=True, chunksize=100, if_exists="replace")
CPU times: total: 62.5 ms
Wall time: 194 ms

[40]: XXsql
CREATE SCHEMA IF NOT EXISTS eglv AUTHORIZATION env_master
* postgresql://env_master:***@localhost/env_db
Done.

[40]: []
```

Figure 6: Creating schema evgl with

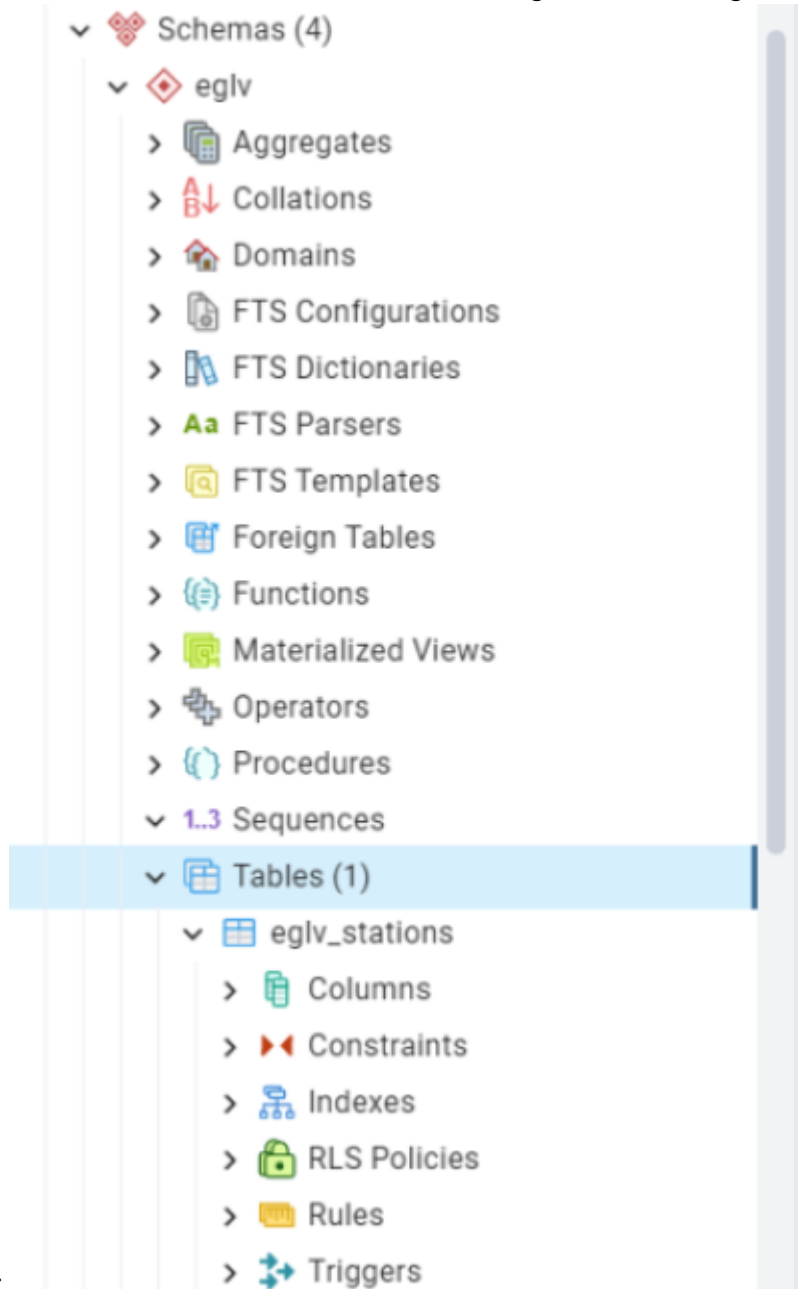


Figure 7:

super user env_master 'eglv_stations' table created under schema eglv shown in PgAdmin 4 **Next, we use a select query to query the table 'eglv_stations' to get all the rows and check if all the data has been**



coordinate system.

Figure 9: The station locations plotted on NRW Topographische Karte Map in EPSG: 31466 CRS **Figure 10 shows the snippet of the location of a few of the stations with a scale of 1:1000000. Dark red dots are used to mark the station on the WMS layer.**

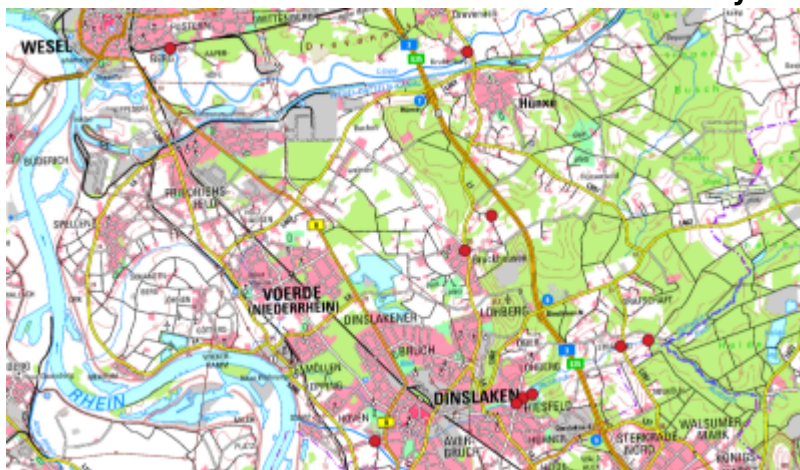


Figure 10: The station locations plotted on NRW Topographische Karte Map in EPSG: 31466 CRS on scale 1:1000000 **While plotting exact points on the map it is also important to take a background map similar to the one we have for referencing. Here in figure 11 below it can be seen that the first image is the selected QGIS map for plotting stations and the second image shows the map which they have on the website. Both maps show the location of the station in KA Hamm.**



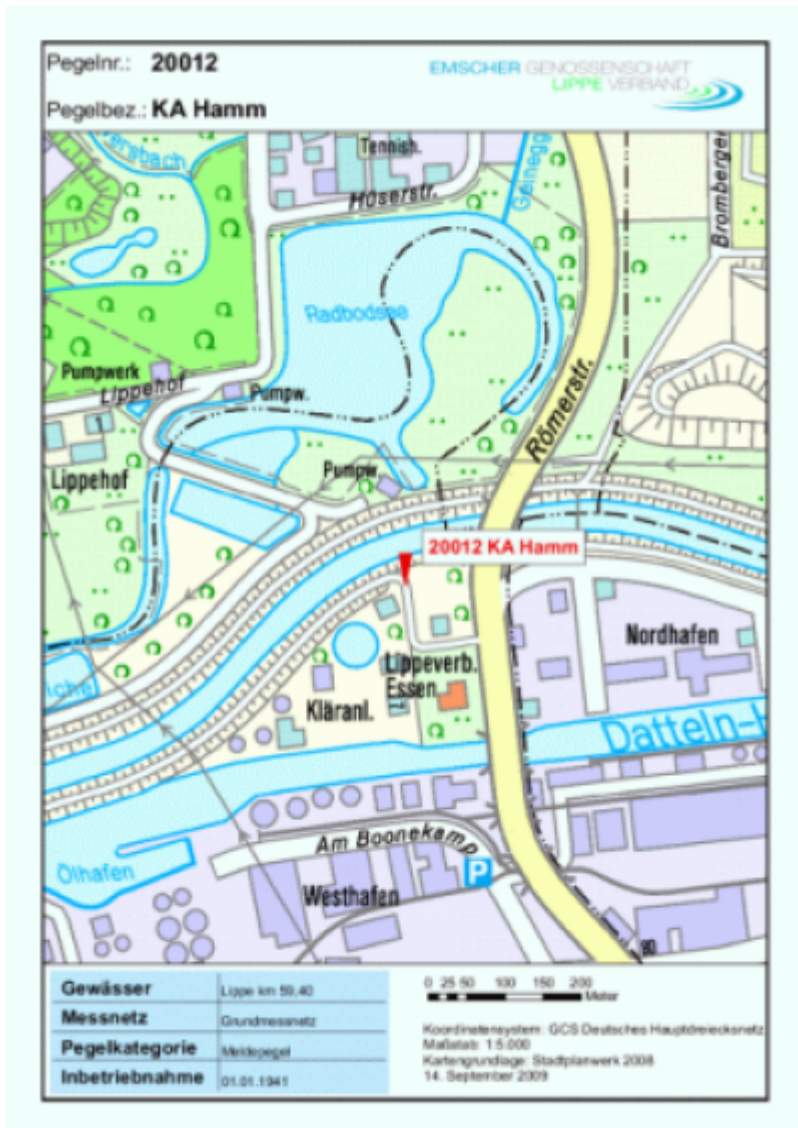
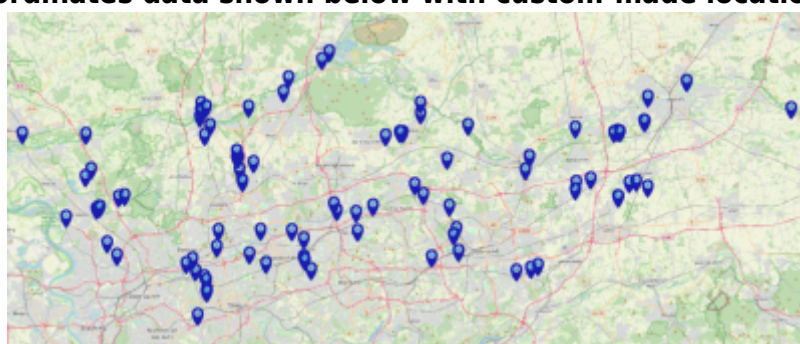
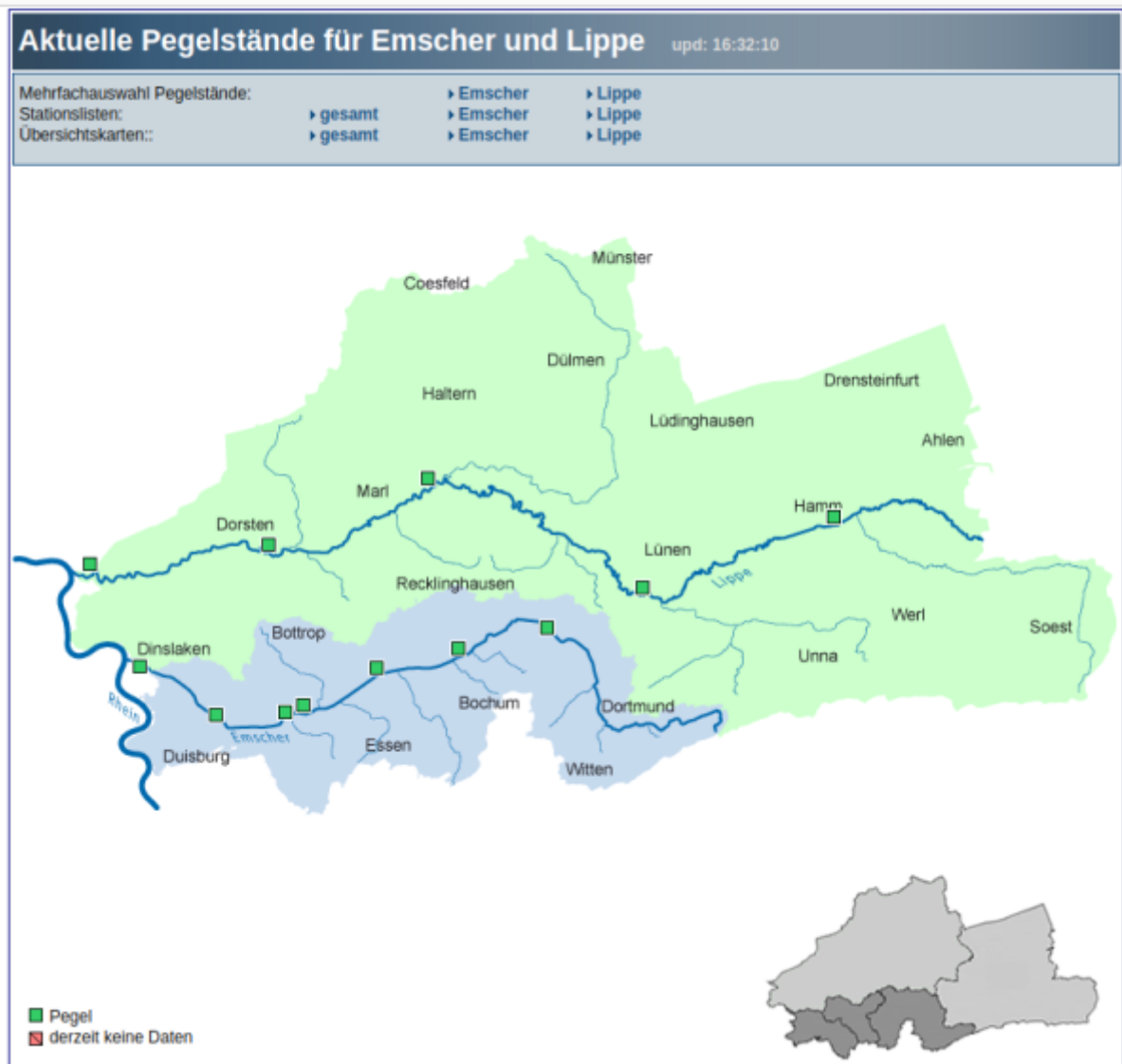


Figure 11: Comparison between KA Hamm Station in QGIS Vs KA Hamm Station in Emscher Genossenschaft Lippe Verband web page. In figure 12 we can see that all the stations are listed on the Emscher Genossenschaft Lippe Verband web page with coordinates data shown below with custom-made location



markers in dark blue color. Figure 12: All stations which are listed on the Emscher Genossenschaft Lippe Verband web page marked with a custom symbol. ===== Periodic Web Scraping of 'Aktuelle Pegelstände für Emscher und Lippe' ===== The objective of this part of the project was to periodically scrape water level and discharge data for the Emscher and Lippe rivers from the website https://howis.eglv.de/pegel/html/uebersicht_internet.php. The scraped data was then stored in a database and visualized using temporal controllers in QGIS. The purpose of this report is to document the methodology used, the findings, and the conclusions of the



project.

==== Web Scraping ==== To scrape the water level and discharge data from the website, we used the Python programming language and the Beautiful Soup library. The data was scraped for a period of 21 days, from March 6, 2023, to March 27, 2023, for different periods of time throughout the day. The data was collected for 12 stations: Fusternberg, Lünen, Hamm, Mengede, Dorsten, Recklinghausen, Am Stadthafen, Bahnstraße, Dinslaken, Konrad-Adenauer-Straße, Bottrop, Essener Straße, Bottrop-Süd, Haltern, Gelsenkirchen, Adenauerallee. The code we have used to do the web scraping is shown in the figure below:

```
import requests
#Imports the 'requests' library, which allows sending HTTP requests and receiving responses
from bs4 import BeautifulSoup
#Imports the 'BeautifulSoup' library, which is a parser for HTML and XML documents
import pandas as pd
import time
#Imports the 'time' module, which provides various time-related functions

while True:
    url = 'https://howis.eglv.de/pegel/html/uebersicht_internet.php'
    #Defines the URL of the website to be scraped
    response = requests.get(url)
    #Sends a GET request to the URL and receives the response

    soup = BeautifulSoup(response.content, 'html.parser')
    #Parses the HTML content of the response using BeautifulSoup and stores it in the variable 'soup'

    tooltips = soup.find_all('div', {'class': 'tooltip'})
    #Extracts all the div tags with the class 'tooltip' from the parsed HTML content and stores them in the variable 'tooltips'

    #A for loop that iterates over each tooltip in 'tooltips' and extracts the relevant information from it.
    #It extracts the location, water level, discharge, date, and time, and appends them to the list 'tooltip_info'
    tooltip_info = []

    for tooltip in tooltips:
        location = tooltip.find('div', {'class': 'tooltip-head'}).text.strip()
        values = tooltip.find('div', {'class': 'tooltip-body'}).find_all('td', {'class': 'tooltip-value'})
        dates = tooltip.find('div', {'class': 'tooltip-body'}).find_all('td', {'class': 'tooltip-date'})

        if len(values) >= 1:
            width = values[0].text.strip()
        else:
            width = None

        if len(values) >= 2:
            flow = values[1].text.strip()
        else:
            flow = None

        date = None
        time_ = None
        if len(dates) >= 1:
            date = dates[0].text.strip()
        if len(dates) >= 2:
            time_ = dates[1].text.strip()

        tooltip_info.append({'location': location, 'Water_level': width, 'Discharge': flow, 'date': date, 'time': time_})

    new_df = pd.DataFrame(tooltip_info)
    #Creates a new Pandas dataframe from the list 'tooltip_info' and stores it in the variable 'new_df'

    df = pd.concat([df, new_df], ignore_index=True)
    #Concatenates the new dataframe 'new_df' with the existing dataframe 'df', and updates 'df' with the concatenated result.

    time.sleep(300)
    #Pauses the loop for 300 seconds (5 minutes) before starting the next iteration
```

Figure 13.

Web Scraping code **The code uses Python to perform web scraping of water level and discharge data from the website https://howis.eglv.de/pegel/html/uebersicht_internet.php. The purpose is to periodically collect data over a range of dates, and store it in a Pandas dataframe for visualization purposes. The code uses the requests library to send HTTP GET requests to the website and receive responses. Then, it uses the BeautifulSoup library to parse the HTML content of the response and extract relevant information. Specifically, it extracts the location, water level, discharge, date, and time for each station using a for loop that iterates over each div tag with the class 'tooltip'. The extracted information is then appended to a list called 'tooltip_info'. Next, the code creates a new Pandas dataframe from the list 'tooltip_info' and stores it in the variable 'new_df'. Finally, the code concatenates the new dataframe 'new_df' with an existing dataframe 'df', and updates 'df' with the concatenated result. This process is repeated every 5 minutes using the time.sleep() function. The resulting dataframe contains the water level and discharge data for the 12 stations over the range of dates from March 6th to March 27th, 2023, which can be used for visualization in QGIS using the temporal controller. After scraping the data from the website and removing duplicates, we needed to create a unique identifier for each station and combine the date and time columns into a single column with datetime format.**



Figure 14.

Adding SID and timestamp **To achieve this, we first made a copy of the cleaned DataFrame df1 using the copy() method, as shown in Figure 13 and assigned it to a new variable df_copy. Then, we created a new column named SID in df_copy which groups the data based on the location column and assigns unique integer values to each group using the groupby() and ngroup() methods. This allows us to uniquely identify each station in the data. Next, we combined the date and time columns in df_copy into a new column named timestamp. This was done by concatenating the two columns with a space in between using the + operator. Finally, we converted the timestamp column into a datetime format using the pd.to_datetime() method and provided the format string '%d.%m.%Y %H:%M' which specifies the format of the date and time strings in the column. The resulting modified DataFrame df_copy contains the original columns as well as the newly created SID and timestamp columns which will be used for further analysis and visualization. After eliminating duplicates and NaN values, we performed data cleaning by creating two new dataframes named 'new_df2' and 'new_df3'. 'new_df2' contains only the 'timestamp', 'SID', and 'Water_level' columns from the previous data frame named 'new_df1'. A new column named 'VAL' is added to 'new_df2' with 'cm' as its value for all rows. The 'Water_level' column is renamed as 'PARAM', which stands for PARAMETER. This is shown in Figure 15. Similarly, 'new_df3' contains only the 'timestamp', 'SID', and 'Discharge' columns from the previous data frame named 'new_df1'.**

```
#create a new DataFrame 'new_df2' that contains only the 'timestamp', 'SID', and 'Water_level'
#columns from the 'new_df1' DataFrame
new_df2 = new_df1[['timestamp', 'SID', 'Water_level']].copy()

#add a new column named 'VAL' and assign 'cm' as its value for all rows
new_df2['VAL'] = 'cm'

#rename the 'Water_level' column to 'PARAM' which stands for PARAMETER
new_df2.rename(columns={'Water_level': 'PARAM'}, inplace=True)

new_df2
```

	timestamp	SID	PARAM	VAL
0	2023-03-06 23:30:00	8	274.0	cm
3	2023-03-06 23:30:00	10	255.0	cm
4	2023-03-06 23:30:00	11	338.0	cm
5	2023-03-06 23:30:00	2	128.0	cm
1	2023-03-06 23:35:00	9	499.0	cm
...
234031	2023-03-28 18:30:00	4	159.0	cm
234032	2023-03-28 18:30:00	6	81.0	cm
234035	2023-03-28 18:30:00	7	92.0	cm
234034	2023-03-28 18:35:00	5	231.0	cm
234033	2023-03-28 18:35:00	3	225.0	cm

14026 rows x 4 columns

```
#do the same thing as above but this time for Discharge with unit m3/s
new_df3 = new_df1[['timestamp', 'SID', 'Discharge']].copy()
new_df3['VAL'] = 'm3/s'
new_df3.rename(columns={'Discharge': 'PARAM'}, inplace=True)
new_df3
```

Figure 15.

Creating specific dataframe for Water Level and Discharge **Then the two cleaned DataFrames, 'new_df2' and 'new_df3', were concatenated into a single DataFrame 'joined_df', which contained the 'timestamp', 'SID', 'PARAM', and 'VAL' columns from both the original DataFrames. This was achieved using the 'pd.concat()' function, with the two DataFrames and the axis set to 0.**

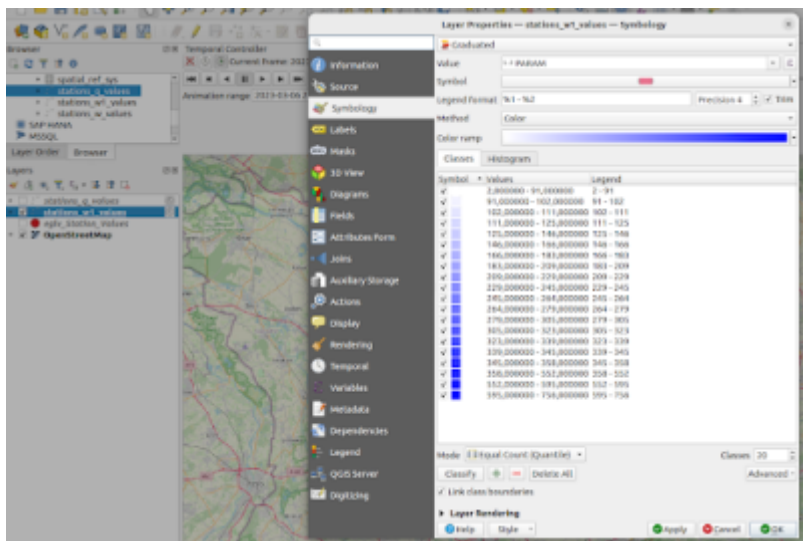
```
#concatenate the 'timestamp', 'SID', 'PARAM', and 'VAL' columns of the 'new_df2' and 'new_df3' DataFrames
#along the rows using pd.concat() function and create a new DataFrame 'joined_df'
joined_df = pd.concat([new_df2[['timestamp', 'SID', 'PARAM', 'VAL']], new_df3[['timestamp', 'SID', 'PARAM', 'VAL']]], axis=0)
joined_df
```

	timestamp	SID	PARAM	VAL
0	2023-03-06 23:30:00	8	274.0	cm
3	2023-03-06 23:30:00	10	255.0	cm
4	2023-03-06 23:30:00	11	338.0	cm
5	2023-03-06 23:30:00	2	128.0	cm
1	2023-03-06 23:35:00	9	499.0	cm
...
234021	2023-03-28 18:25:00	3	15.9	m3/s
234013	2023-03-28 18:25:00	9	88.9	m3/s
234025	2023-03-28 18:30:00	9	88.6	m3/s
234028	2023-03-28 18:30:00	11	39.5	m3/s
234033	2023-03-28 18:35:00	3	15.9	m3/s

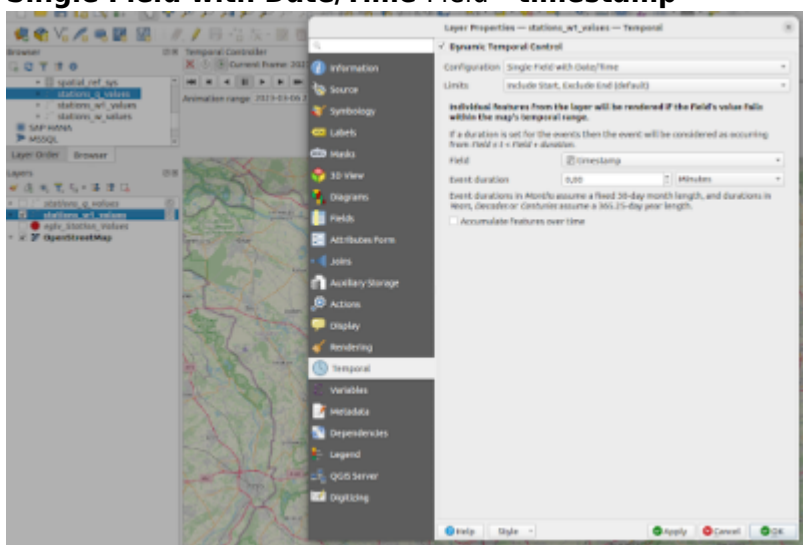
19026 rows x 4 columns

Figure 16.

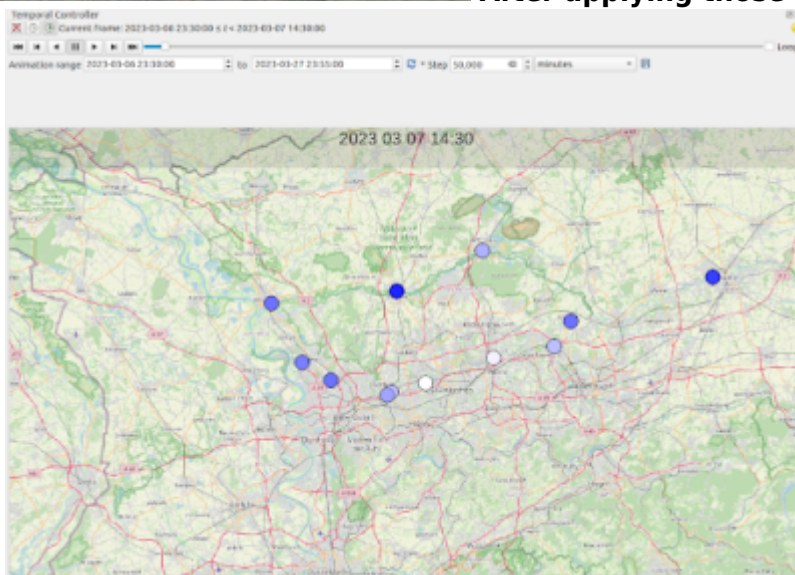
Creating a new DataFrame with both parameters **In order to create a DataFrame containing the station ID, location name, and coordinates for each location, we performed several data**



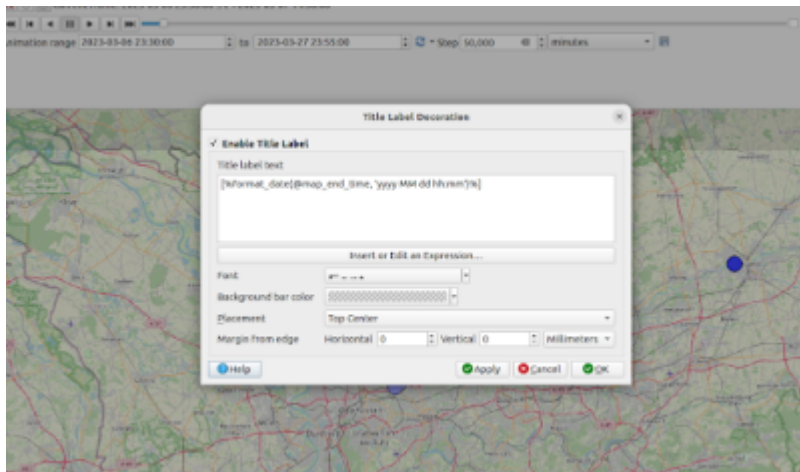
Type - **Graduated Value - PARAM**
 Classes - **20** After that we define Temporal Controller settings as above: Configuration - **Single Field with Date/Time** Field - **timestamp**



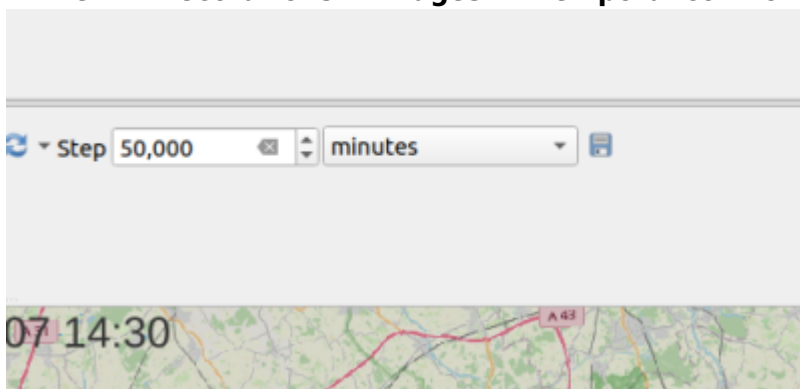
After applying those properties the



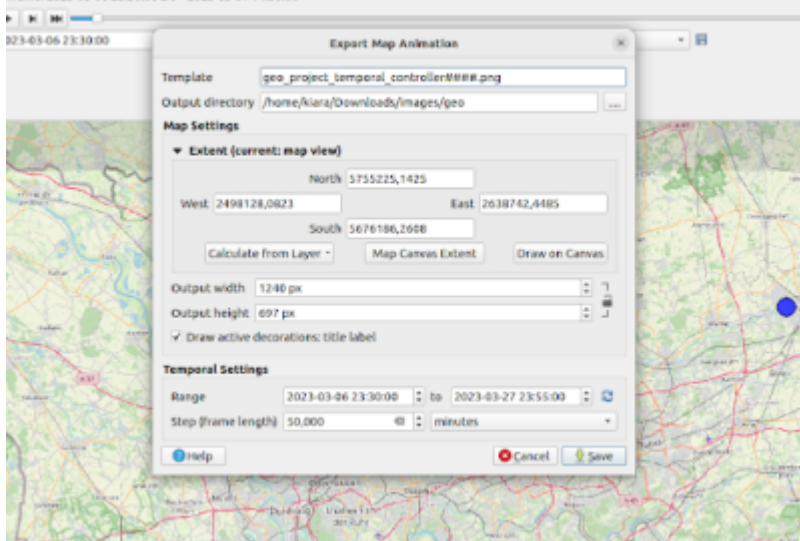
view will be as below: we have also added here a Text Label that shows date and time when the temporal controller moves. This can be found at: View → Decorations → Title label by adding the expression as below:



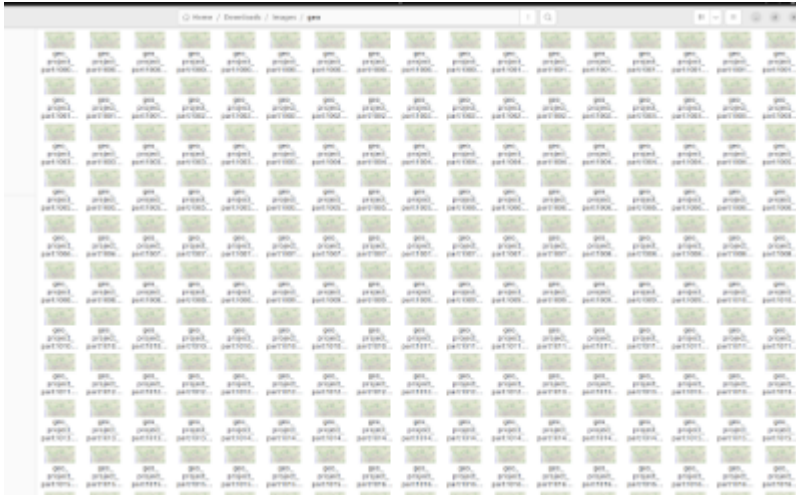
It is possible also to add the legend in View → Decorations → Images. In temporal controller toolbar there is a save option:



Now we are going to save Map animations into folder Geo and then will use these “.png” images to create a video.



Folder geo looks like this:



Then we can use an external tool to

create the video out of these images. We used

<https://clideo.com/image-sequence-to-video> and the result is as above: For water level

[screencast_from_30.03.2023_224438.webm](#)

For Discharge

[screencast_from_30.03.2023_221708_2_.webm](#)

==== Findings: ==== Based on the data we collected, it was found that the Emscher river had more changes in water level in the stations where data was collected compared to the Lippe river. In terms of discharge, we only had data available for a limited number of stations including 20012 KA Hamm, 20004 Dorsten, 10099 Dinslaken Konrad-Adenauer-Straße, and 10103 Bahnstraße. However, it was observed that even these stations had variations in discharge over time. ==== **Save Pegel Station's images in PostgreSQL** ==== **We can store images in PostgreSQL by using the byte data type. The byte data type is used to store binary data, including images, in a PostgreSQL database. To store an image in PostgreSQL, we would need to convert the image to a binary format, such as a byte array, and then insert the binary data into a bytea column in our table. We used the code below to save images in database:**

```

In [1]: import requests
from bs4 import BeautifulSoup
import os
import io
from PIL import Image
import psycopg2

#Set the proj_lib_path variable to the path where the proj folder is located in my local machine.
proj_lib_path = "/home/kisara/anacardalis/emsch/pegtop/sharu/proj"

#Added the proj_lib_path to the PROJ_LIB environment variable using the os.environ dictionary.
os.environ['proj_lib'] = proj_lib_path
proj_lib = os.environ['proj_lib']

#Printed the updated value of the PROJ_LIB environment variable
print(f"New env var value: \nPROJ_LIB={proj_lib}")

# Connect to PostgreSQL database
conn = psycopg2.connect(database='env_db', user='env_master', password='M123xyz', host='localhost', port='5432')
print("Database opened successfully")

# Create a table for images
cur = conn.cursor()
cur.execute("""CREATE TABLE IF NOT EXISTS images
(id SERIAL PRIMARY KEY,
pid_val INTEGER,
image BYTEA);""")
print("Table created successfully")

# Loop through PIDVal values and save images to database
for pid_val in range(1, 101):
    try:
        url = f'https://hwsis.ejlv.de/pegel/html/staendeter.html?PIDVal={pid_val}'
        page = requests.get(url)
        soup = BeautifulSoup(page.content, 'html.parser')
        map_container = soup.find('div', {'id': 'mapcontainer'})
        img_tag = map_container.find('img')
        ext_url = img_tag['src'] if img_tag else None # Check if img_tag exists before accessing its 'src' attr

        if ext_url:
            base_url = "https://hwsis.ejlv.de/pegel/"
            full_url = base_url + ext_url
            response = requests.get(full_url, stream=True)
            img_data = response.content

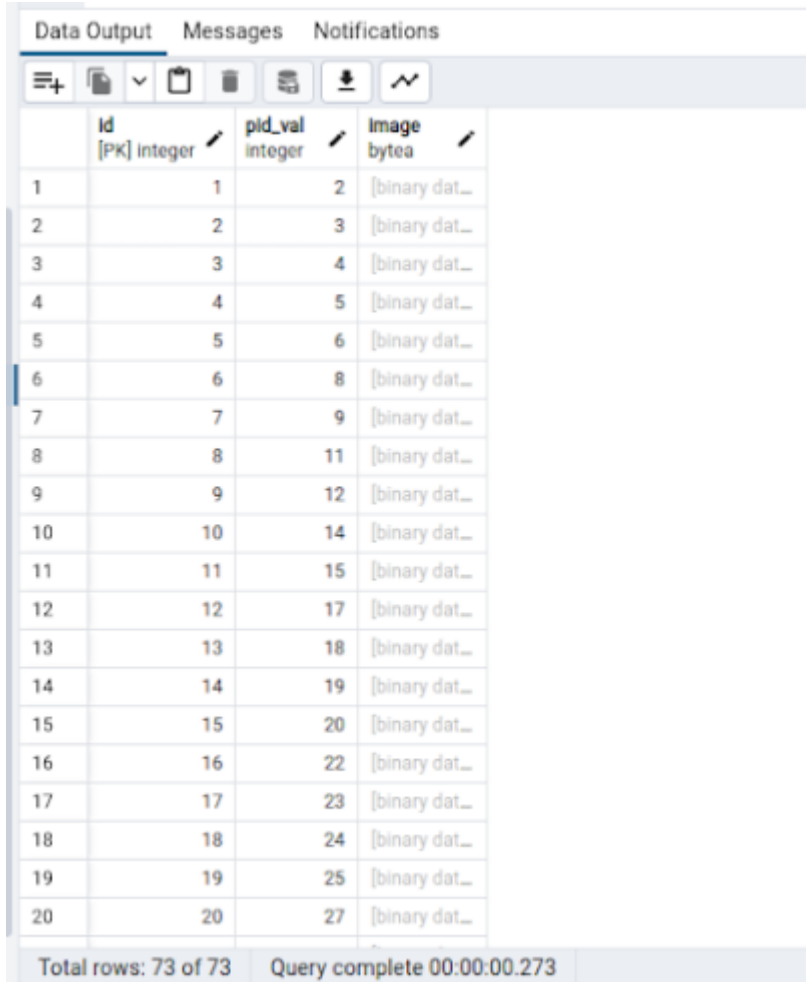
            # Insert image data into the database
            cur.execute(f"INSERT INTO images (pid_val, image) VALUES (%s, %s)", (pid_val, psycopg2.Binary(img_data))
            print(f"Image for PIDVal {pid_val} saved successfully")
        else:
            print(f"No image found for PIDVal {pid_val}")
    except Exception as e:
        print(f"Error while processing PIDVal {pid_val}: {e}")

# Commit the changes and close the connection
conn.commit()
conn.close()

```

We firstly import the necessary libraries: requests, BeautifulSoup, os, io, PIL, and psycopg2 and set the path of the proj folder to the proj_lib_path variable and adds it to the PROJ_LIB environment variable using the os.environ dictionary. This is necessary for properly reading geographic coordinates in

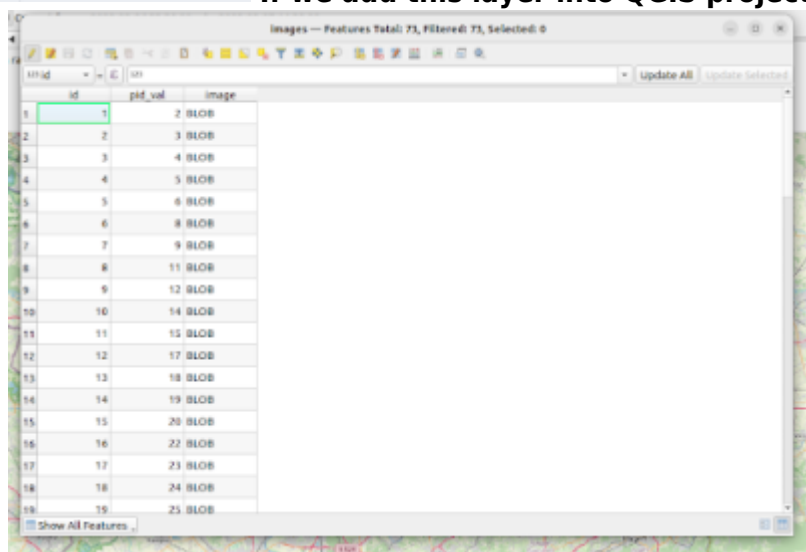
the data. We then connect with the env_db database as the user env_master with the password M123xyz, hosted on localhost at port 5432. Later we create a new table called images in the env_db database with three columns: id (an auto-incrementing serial primary key), pid_val (an integer value from 1 to 100 representing the water level gauge), and image (a binary data type that will store the image data for each water level gauge). It then loops through the values of pid_val from 1 to 100, extracts the image data for each corresponding water level gauge from the website, and saves the image data to the images table in the PostgreSQL database. Finally we commit the changes made to the database and close the connection. In database we will have this table:



	id [PK] integer	pid_val integer	image bytea
1	1	2	[binary dat...
2	2	3	[binary dat...
3	3	4	[binary dat...
4	4	5	[binary dat...
5	5	6	[binary dat...
6	6	8	[binary dat...
7	7	9	[binary dat...
8	8	11	[binary dat...
9	9	12	[binary dat...
10	10	14	[binary dat...
11	11	15	[binary dat...
12	12	17	[binary dat...
13	13	18	[binary dat...
14	14	19	[binary dat...
15	15	20	[binary dat...
16	16	22	[binary dat...
17	17	23	[binary dat...
18	18	24	[binary dat...
19	19	25	[binary dat...
20	20	27	[binary dat...

Total rows: 73 of 73 Query complete 00:00:00.273

If we add this layer into QGIS project



the attribute table will look like this:

==== Georeferencing 5 stations from the scrape data ==== 1. Krudenburg : https://howis.eglv.de/pegel/images/stationpics/maps/20002_stadtplan.gif a. OSM as base



layer

b. DTK farbe as base layer (NRW)



Topographische Map)

https://howis.eglv.de/pegel/images/stationpics/maps/2012_stadtplan.gif

2. KA Hamm:

a. OSM as base



layer

b. DTK farbe as base layer (NRW)



Topographische Map)
Rappahofs Mühlenbach, Zulauf

3. 20018 HRB

https://howis.eglv.de/pegel/images/stationpics/maps/20018_stadtplan.gif a. OSM as base



layer

b. DTK farbe as base layer (NRW)



Topographische Map)

4. OB- Königstraße

https://howis.eglv.de/pegel/html/stammdaten_html/MO_StammdatenPegel.php?PIDVal=18



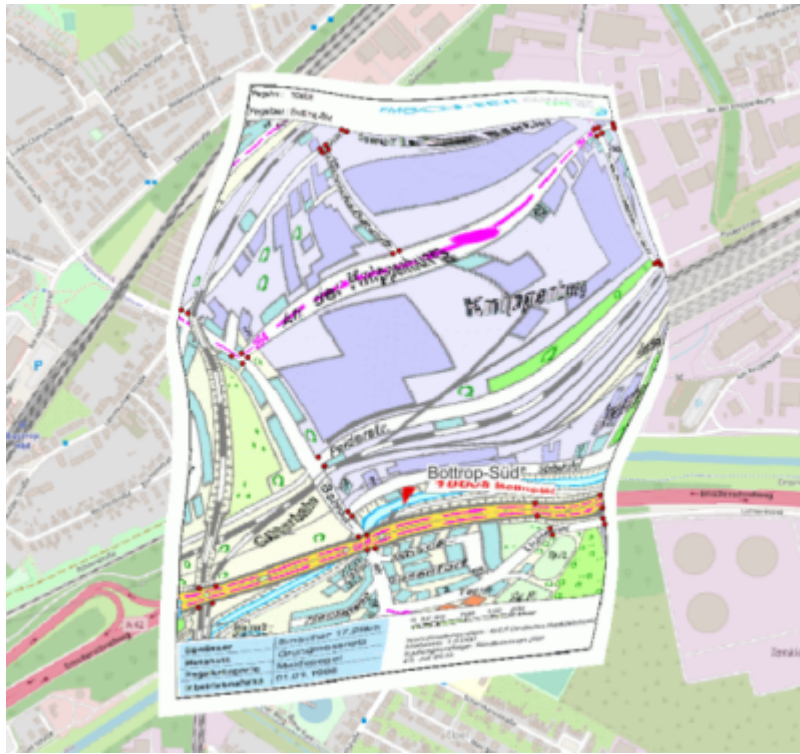
a. OSM as base layer

b. DTK farbe as



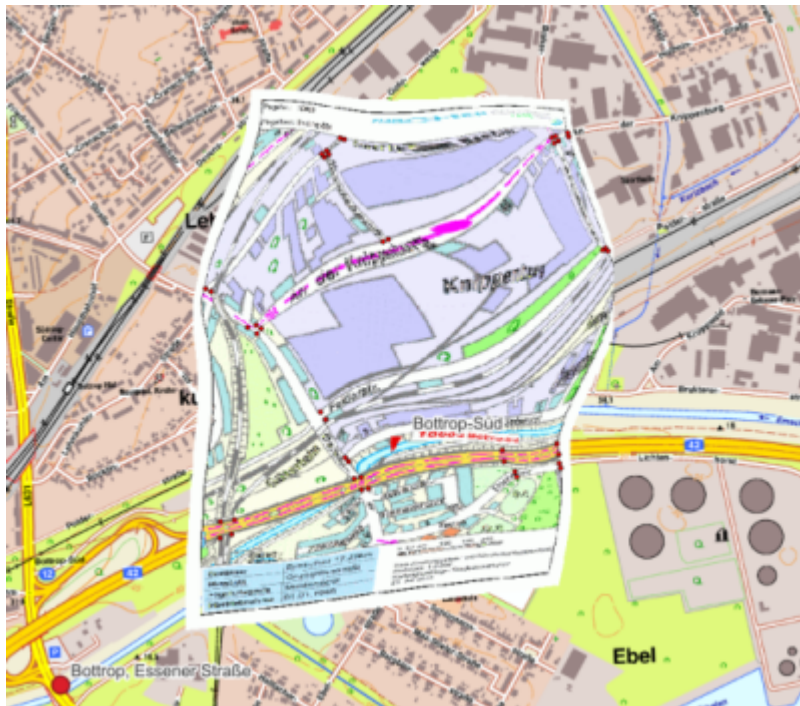
base layer (NRW Topographische Map)
Bottrop Süd https://howis.eglv.de/pegel/images/stationpics/maps/10008_stadtplan.gif

5.



OSM as base layer
layer (NRW Topographische Map) **

b. DTK farbe as base



Conclusion

In conclusion, the web scraping and data visualization project was successful in collecting and visualizing water level and discharge data for the Emscher and Lippe rivers. The project demonstrated the usefulness of web scraping in collecting data for analysis and the effectiveness of using QGIS for visualizing temporal data. Further analysis could be conducted to investigate the factors that contribute to the variation in water levels and discharge over time. Storing the data in a database, as we did in this project, can make it more efficient to access and analyze. Additionally, having historical

data on water level and discharge can be useful in building prediction models that can help mitigate the impact of floods. By understanding the patterns and changes in water level and discharge, we can better prepare and respond to potential floods, which can save lives and reduce damage to property and infrastructure.

Link to Git hub (Jupyter Notebook):

<https://github.com/SindhyaBabu/GeoInformatics-Final>

Additional References

1 - official website for eglv. <https://howis.eglv.de/pegel/intro/index.html>

2 - website used for web scraping https://howis.eglv.de/pegel/html/uebersicht_internet.php

3 - Beautiful Soup documentation

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/#bs4.Tag.name>

4 - GeoPandas library docs

https://geopandas.org/en/stable/gallery/create_geopandas_from_pandas.html

From:

<https://student-wiki.eolab.de/> - HSRW EOLab Students Wiki

Permanent link:

<https://student-wiki.eolab.de/doku.php?id=geoinfo2223:groupb:start&rev=1680297689>

Last update: **2023/03/31 23:21**

