Report: Transbike project

G10: Alber Roethel, Karolina Bogacka, Paulina Pacyna, Marek Brynda

June 22, 2022

1 Introduction

This project aims at analysing the bike usage in a urban bike rental system in Warsaw. We tried to predict the demand and usage of the bikes on a given station using various methods and analysed which stations needs to be extended due to frequent overflow.

2 Methods

2.1 Original data cleaning

The original data, obtained in the form of multiple responses from the nextbike API in the XML format, was later divided into groups of around a 100 responses, naively concatenated and saved as separate files. This files were then first compressed as a single file and then as a the whole folder, with final size reduction from 42.14 GB of the whole uncompressed dataset to around 67 MB of the compressed file.

In order to extract readable data from the folder, we had to first unpack and then divide all of the concatenated files into singular results.

```
previous_name = ""
previous_index = 0
for u in os.listdir("intermediate"):
    if os.path.isdir(os.path.join("intermediate", u)):
        for i in os.listdir(os.path.join("intermediate", u)):
            with open(os.path.join("intermediate", u,
               i),'r',encoding='utf-8-sig') as data_file:
                if u[0:10] != previous_name:
                    previous_index = 0
                for line in data_file:
                    data = line.split("<?xml")
                    data2 = ["<?xml" + d2 for d2 in data]
                    data_final = data2[1:-1]
                    for i2, d in enumerate(data_final):
                        with open(os.path.join("results3",
                           u[0:10]+"-"+str(previous_index+i2)+".xml"),
                           "w",encoding='utf-8-sig') as text:
                            text.write(d)

                previous_index = previous_index+i2+1
                previous_name = u[0:10]
```

Then, we have loaded all files from the folder into a pandas Dataframe and we performed removal of redundant information and saved them in a CSV format, which resulted in significant reduction of disk space needed for the dataset. We decided to only preserve the changing state of the bicycle, which means that we know the first and last timestamp at which the vehicle was on the station. All the remaining data was dropped. Unfortunately, although the names of concatenated files suggested, that the data scraping spanned from around the 11.10.2021 until 23.10.2021, the labelling of the data did not

date	station_id	bike_number	bike_type	state
22.06.2022 18:24	2585259	24998	174	ok
22.06.2022 18:24	2585259	24521	174	ok
22.06.2022 18:24	2585259	28439	174	ok
22.06.2022 18:24	2585259	27499	174	ok
22.06.2022 18:24	2585259	27342	174	ok
22.06.2022 18:24	2585259	27261	174	ok
22.06.2022 18:24	2585259	27104	174	ok

include accurate timestamps measured for a given measurement (or an equal number of measurements recorded for a given labelled file). The lack of precise timestamps deemed the data unusable for time series forecasting and therefore necessitated the scraping of additional data.

2.2 Scraping new data

As we aimed at predicting bike demand for a given time of a day, we decided that it is necessary for us to obtain more data. We discovered that the data is available all the time on a public website. As we wanted to download the data from the nextbike API continouosly, we decided to build a custom web scraper based on AWS cloud services. The solution consisted of three cloud services:

- S3 bucket - to store the data we scraped
- Lambda function - to download, preprocess and upload the data from the nextbike API
- EventBridge - to run the Lambda function every 10 minutes

As we wanted to fit into the free trial on AWS and we used a normal account with one of our private credit cards linked, we knew that we need to use as less memory as possible. Therefore we strongly preprocessed the data and stored in a csv file. The source code of the Lambda functions is as follows:

```
import requests
import datetime
import pandas as pd
import boto3

def lambda_handler(event, context):
    s3 = boto3.resource('s3')
    bucket = s3.Bucket('rowerki')
    date = datetime.datetime.now().strftime("%Y-%m-%d␣%H:%M")
    json =
       requests.get("https://nextbike.net/maps/nextbike-official.json?\
␣␣␣␣city=372,210,475").json()
    data = []
    for city in json["countries"][0]['cities']:
        for place in city['places']:
            for bike in place['bike_list']:
                data.append((date, place['uid'], bike["number"],
                   bike["bike_type"], bike["state"]))
    csv = pd.DataFrame(data, columns=['date', 'station_id',
       "bike_number", "bike_type", "state"]).to_csv(index=False)
    s3_obj = s3.Object("rowerki",
       str(int(datetime.datetime.now().timestamp())) + '.csv')
    s3_obj.put(Body=csv)
```

By preprocessing the large json document available on the website we decreased the size of a single snapshot of the data from c.a. 800 kb to c.a. 130 kb. Here is a sample of the csv file. The whole data from 2 months takes 1.08 GB, compared to 67 GB of data from one month that was given to us at the beginning of the project. This optimisation led to cutting the cost of storage by a great factor and improving the processing time. The data has also a timestamp assigned to each entry, and it was

obtained regularly, every 10 minutes with no gaps, which makes it a very clean and consistent dataset. Another advantage of keeping the data on a public S3 bucket was that everybody in the team, given a credentials file, could download the latest version of the data by typing just one command to the terminal:

```
aws s3 sync s3://rowerki <destination folder>
```

Right now the command is not working, as purposely disabled downloading the data, in order to avoid further costs. However if you want to download the data and use it for other students projects, please find it here or on Teams.

2.3 Explorative data analysis

The data, in the form stored on the rowerki_df AWS S3, consisted of 6 fields: index, date (containing both the day and the minute of measurement), station_id (which is unique to a given station in Warsaw), bike_number (which is unique to a given nextbike bike), bike_type and state. The state of the bikes was equal to 'ok' for every obtained measurement, therefore it was dropped as an unnecessary column. The bike_type, on the other hand, consisted of a limited number of numeric values. Analyzing the source code of the nextbike website allowed us to understand the meaning of some of them:

- **174** stood for standard bikes,
- **26** stood for tandem bikes,
- **37** stood for electric bikes,
- **35** and **34** stood for various types of children's bikes

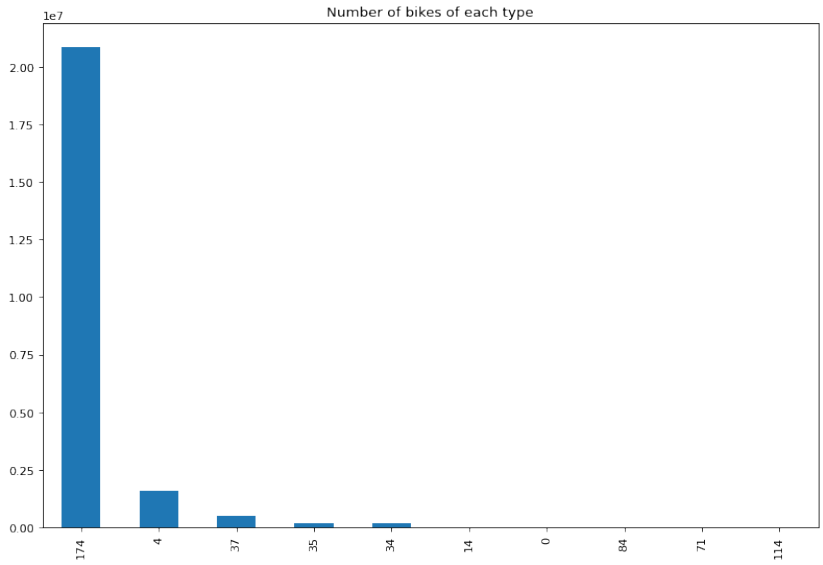


Figure 1: Distribution of bike types

The vast majority of the measurements gathered on the S3 instance belonged to the standard bike type. At the time of final model development, there were 23307413 measurements stored on the S3 instance.

We have also performed the analysis of several stations to investigate whether we can recognize any significant pattern or seasonality in the data. We were able to recognize two neighbouring stations near University of Warsaw (scientific campus) that presented significant seasonality of the station occupation. As it can be seen of Fig ?? at the before 8 AM the number of vehicles available is constant. It is followed, however, by rapid increase in the number of bikes attached to the station till 10 AM. It corresponds to period during which mosts of the classes at the campus starts. Later all the bikes are taken out through the rest of the day and it decreases towards 0. It can be observed that in the night there are some drastic bumps which most likely correspond to bikes delivery. Some of the

days exhibit less dynamic fluctuation of bikes and it turns out that these are weekend during which there si no classes at the university.

We hope that there will be more of such visible patterns in the data so that models can be easily trained, however after some investigation it is not trivial to detect any other station that shows so strong seasonality of the station occupation.

Another analysis that we performed was to check which stations had the largest drop in station occupation through the day. In order to do so, we calculated the average number of bikes available in the station aggregated by hour, and then we subtracted the minimum value from maximal one to observe the jump of the bikes availability in the station. In turned out that two stations with the greatest peak were those that are close to the Warsaw University. The other ones were Metro Centrum Nauki Kopernik, which is often overwhelmed by the huge number of bikes attached to it by people that wants to spend some time by the Wisla river and the Metro Rondo Daszynskiego which is a skyscrapers hub with huge number of offices. Stations with the smallest day peaks were Veturilko stations for youngest users on suburbs.

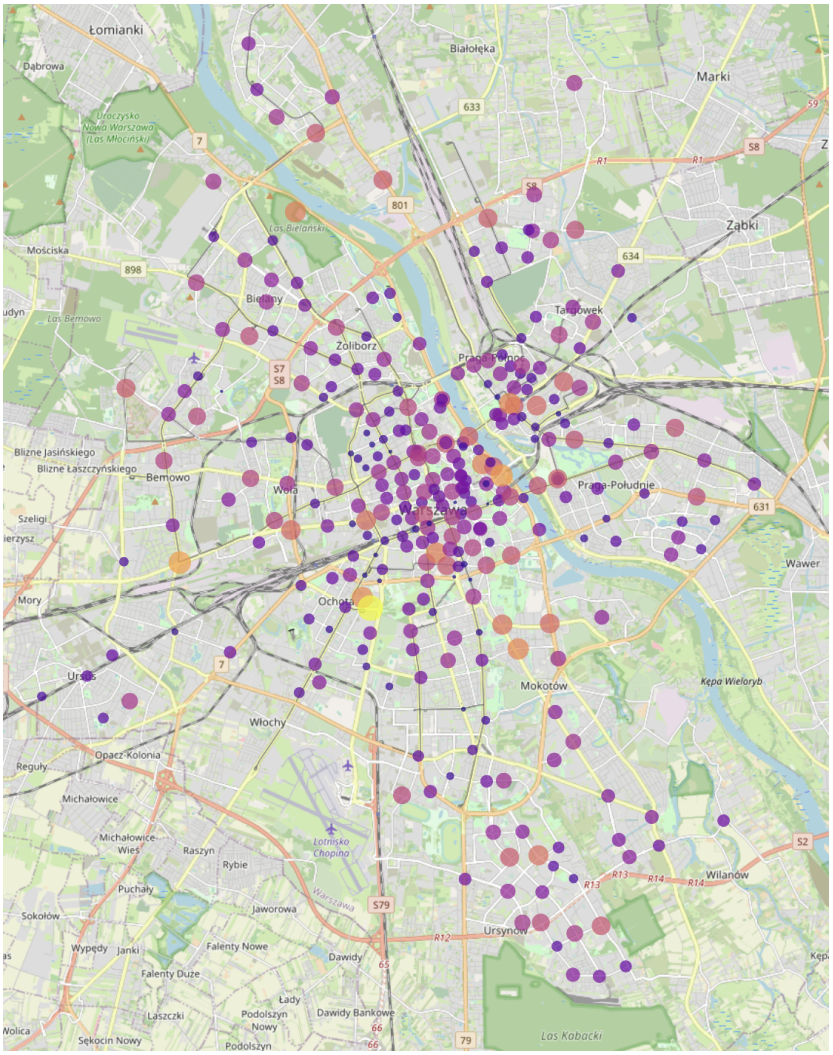


Figure 2: Average occupation of all stations at 10 AM. The bigger and brighter the dot, the more bikes were assigned to it.

2.4 Rented bikes activity visualisation

As a part of our explorative data analysis, we created a simple visualisation of the activity during the day. We plotted the stations on a map, highling each station on red when we recorded that somebody

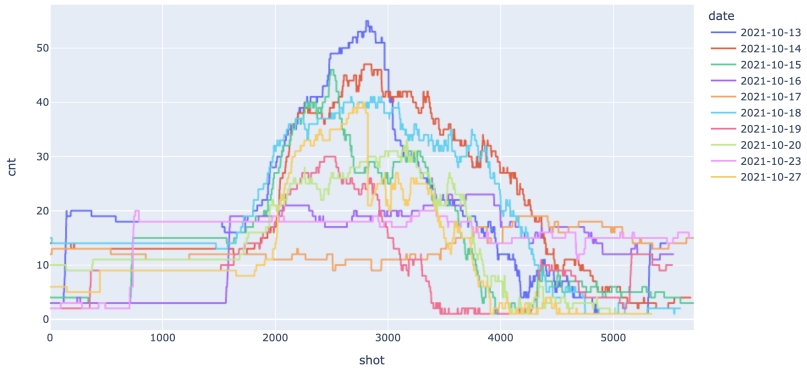


Figure 3: Univeristy of Warsaw station occupation.

rented a bike on this station. Then the color of the station was slowly diminishing, indicating that there was some activity recently. The visualisation can be found under this link.

2.5 Bike demand forecasting

One of the ways in which we decided to provide business value to the company was by forecasting the bike demand of a given station for a given time interval. Our reasoning was, that in order to maximize profits in a situation in which, for example, there is a need to conduct maintenance on a given station, it would be beneficial to estimate the costs of deactivating such a station for a set date and hour. The company could thus schedule the maintenance for the most advantageous hours possible.

2.5.1 Data preprocessing

We have first grouped the measurements per a given date and station_id to obtain the number of bikes which were registered on a given station for a given measurement. Then, we have subtracted the bike_number for a given measurement (for a given station for a given time) with the bike_number for the next timestamp, which gave us the number of bikes rented for a given timestamp. We have later grouped these differences in order to get either daily or hourly demand.

The information about a given weekday, day of the month, month and hour (in the case of the forecasting for a given hour) of a measurement were later separately extracted and added to the DataFrame.

In order to improve the expected results for this goal, additional data was obtained and merged with the current Pandas DataFrame. This data mainly consisted of historical weather data, acquired with the help of Python **meteostat** library. Since there was only one meteostat Station available for all the bike stations listed, the weather data was scraped for a given hour, identical for every station in Warsaw.

```
def get_weather(row):
  # weather for the hour (more or less) of the measurement
  start = datetime(2022, row['Month'], row['Day'], row['Hour'])
  end = datetime(2022, row['Month'], row['Day'], row.Hour)
  location = Point(float(52.183992), float(21.00984), 100)
  data = Hourly(location, start, end)
  data = data.fetch()
  row['temp'] = float(data['temp']) if not math.isnan(data['temp'])
     else 0.0
  row['dwpt'] = float(data['dwpt']) if not math.isnan(data['dwpt'])
     else 0.0
  row['rhum'] = float(data['rhum']) if not math.isnan(data['rhum'])
     else 0.0
  row['prcp'] = float(data['prcp']) if not math.isnan(data['prcp'])
     else 0.0
```

```
    row['snow'] = float(data['snow']) if not math.isnan(data['snow'])
        else 0.0
    row['wdir'] = float(data['wdir']) if not math.isnan(data['wdir'])
        else 0.0
    row['wspd'] = float(data['wspd']) if not math.isnan(data['wspd'])
        else 0.0
    row['wpgt'] = float(data['wpgt']) if not math.isnan(data['wpgt'])
        else 0.0
    row['pres'] = float(data['pres']) if not math.isnan(data['pres'])
        else 0.0
    row['tsun'] = float(data['tsun']) if not math.isnan(data['tsun'])
        else 0.0
    row['coco'] = float(data['coco']) if not math.isnan(data['coco'])
        else 0.0
    return row

weather_df2 = weather_df.apply(get_weather, axis=1)
weather_df2
```

The data contained fields related to the air temperature in *Celcius*, the dewpoint in *Celsius*, the relative humidity in percent (%), the one hour precipitation total in *mm*, the snow depth in *mm*, the average wind direction in degrees, the average wind speed in *km/hs*, the peak wind gust in *km/h*, the average sea-level air pressure in *hPa*, the one hour sunshine total in minutes(m) as well as the weather condition code.

Due to the high present of null values, columns like **snow** (meaning snow depth) and **tsun** meaning one hour sunshine total were dropped.

As a part of the preprocessing, columns unnecessary for later predictions, such as station latitude, longitude, name and number of bike racks get removed. Then, using the Python package **holidays**, the information about whether a given measurement refers to a holiday gets added.

2.5.2 LSTM model development

Before LSTM training, the listed station_id's were encoded using **OrdinalEncoder**, while the rest of the training data was scaled using **StandardScaler** in order to lead to faster model convergence. The data was then divided into 72 hour windows, with labels in the form of predicted demand for a given bike. The validation data and testing data each constituted around 0.1 of the entire dataset.

We have tried both BiLSTMs and LSTMs for prediction. However, until we have decided to conduct the training for a chosen, given station, the results were less than satisfactory, with the estimated predictions resembling a moving mean of the whole dataset.

After the decision to limit the scope of a given model to consider only one station, we have decided to use an LSTM architecture with 3 dense layers, two layers of 220 LSTM units each and one Dropout layer. We have used the Adam optimizer for training.

2.5.3 XGRegressor model development

After the initial lack of success while using the simple BiLSTMs and LSTMs, an alternative method on the XGBoost was developed. In this case, the data was not divided into sliding windows, but presented as single measurements gathered for a given day and station. The training data was scaled beforehand, with station_id once again encoded using **OrdinalEncoder**, but a different method was used to eliminate or condense potentially unnecessary data: Principal Component Analysis, which enabled us to determine the most crucial components and use only them for predictions. Only the 8 most important components were selected.

Data was divided into the training set of 12212 samples and test set of 2442 samples.

We have achieved the best results for an XGBRegressor with the learning rate of 0.1, subsample of 0.9, max_depth of 8 and, interestingly, n_estimators of 10000.

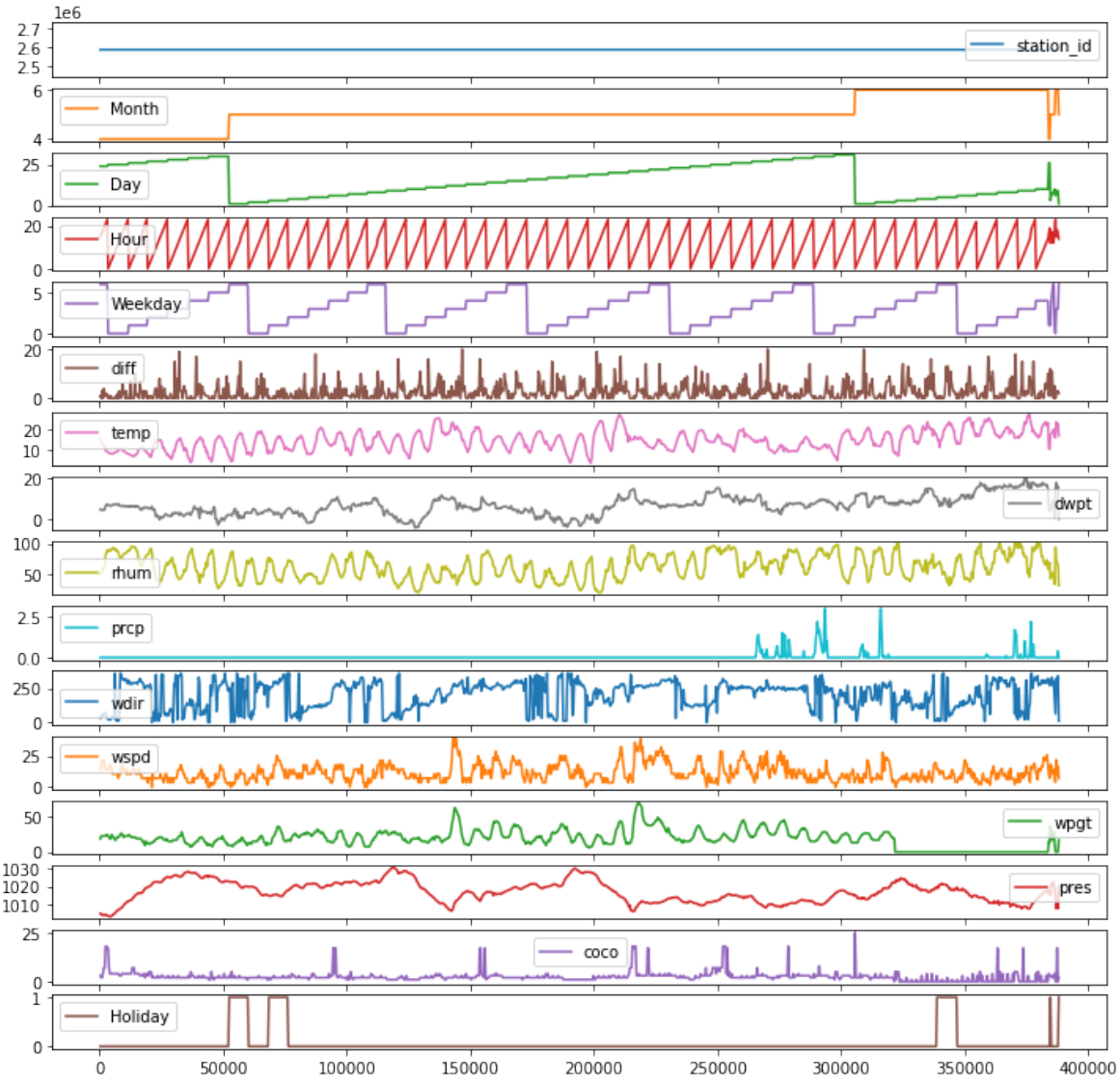


Figure 4: DataFrame values gathered for a sample station

2.6 Seasonality analysis

We suspected that there could be some seasonality present in the data. We wanted to decomposed the timeseries of bike count on each stations into components corresponding to weekday, hour and overall trend. We performed such an analysis for each station separately. Here (Figure 2.6) is an example of this decomposition for the Metro Stokłosy station:

2.7 Station traffic analysis

Another product designed to assist the company management in decision making involved the placement of bike stations in Warsaw. We have aggregated the data in order to display how crowded a given station is, using a couple of various coefficients, and provided tools for visual analysis that could prove crucial in the event in which nextbike would either plan to add or subtract a station in Warsaw.

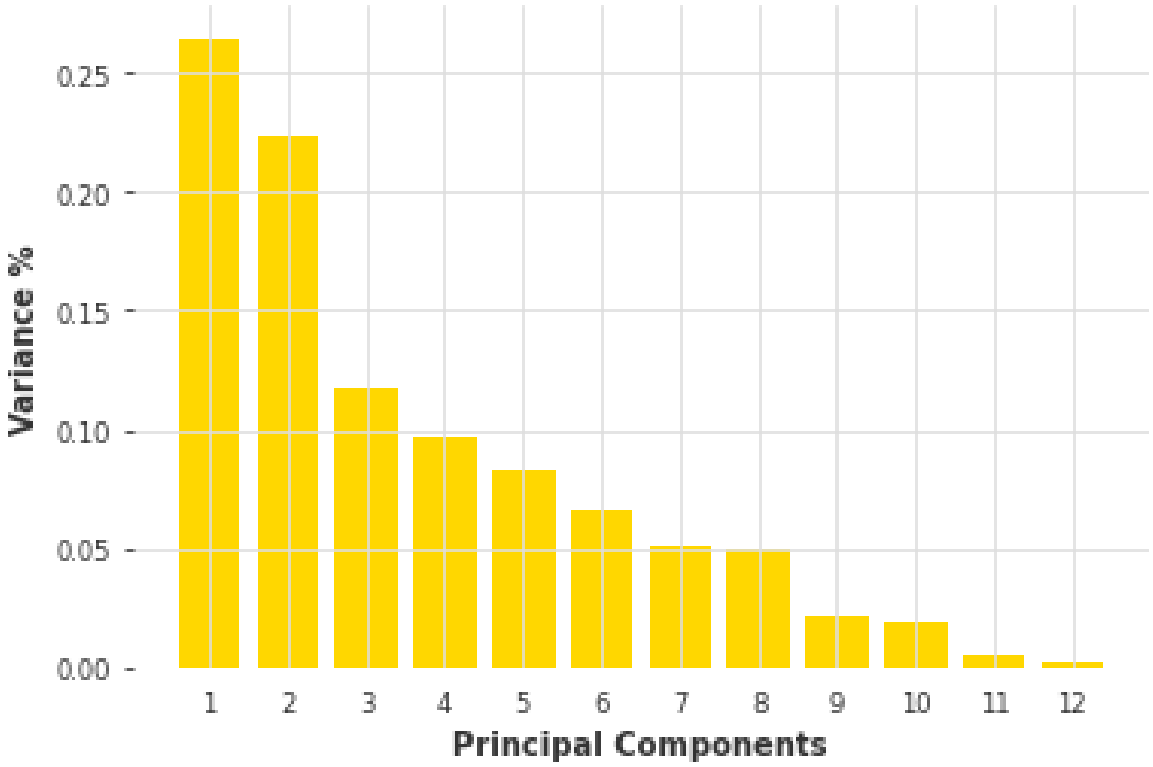


Figure 5: PCA components on the test set

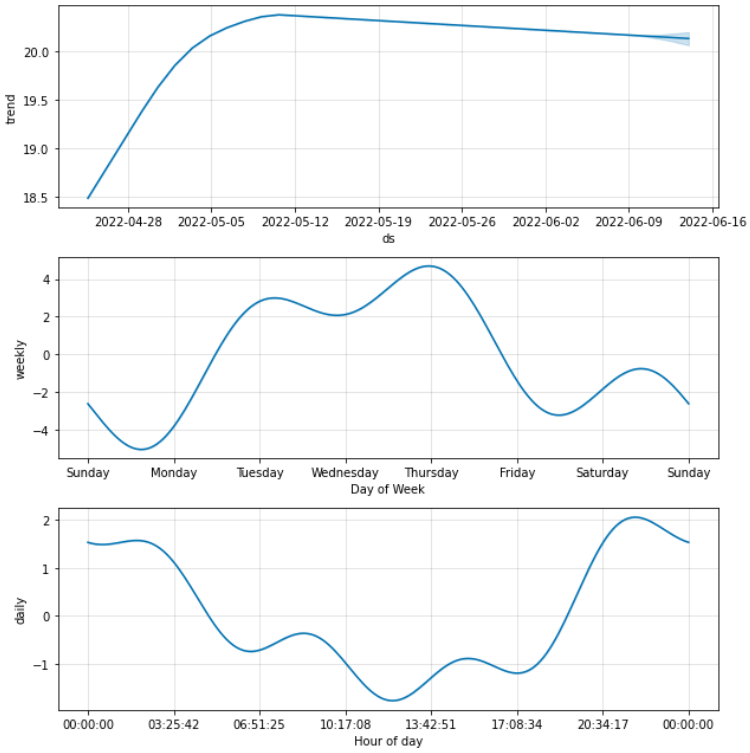


Figure 6: Seasonality decomposition for the timeseries based on data from Metro Stokłosy

2.7.1 Data preprocessing

In order to conduct this analysis, our scrapped data was merged with the data from nextbike informing us about a number of bike racks for a given station. Then, three types of coefficients were obtained:

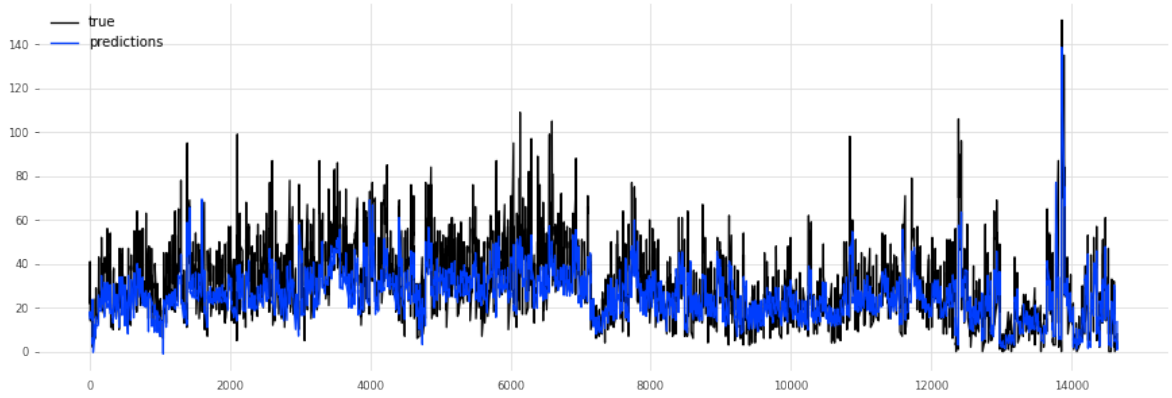
- mean bike number by the mean number of racks per measurement if the bike number is bigger than the number of racks,
- mean bike number by the mean number of racks per measurement,
- mean bike number by the mean number of racks variance per station.

2.7.2 Data visualization

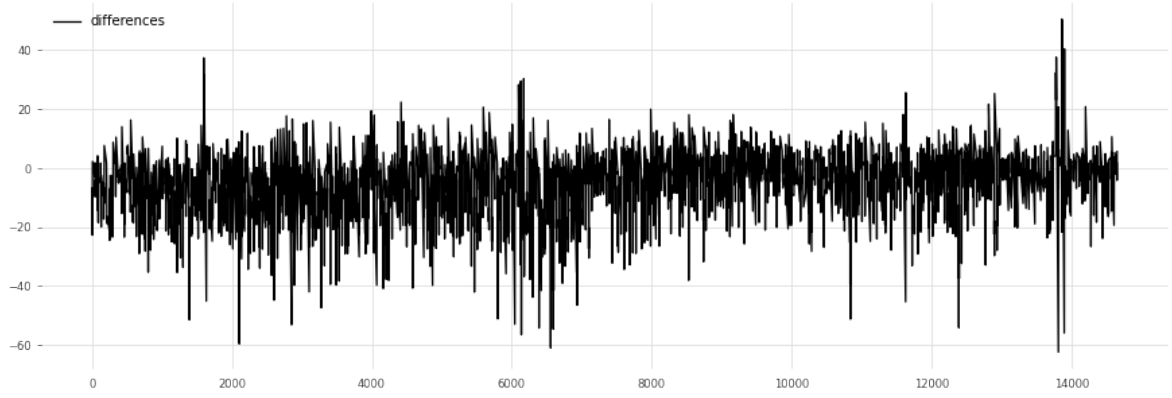
The data was then plotted using the **folium** and **geopandas** Python libraries.

3 Results

The best result for the XGBRegressor was of mean square error of 0.13 on the training set, but 180.95 on the test set, which is unfortunately insufficient for our needs.



(a) Predicted and true labels



(b) Differences between measurements and predictions

Figure 7: Results for the XGBRegressor model

The best result obtained for the final LSTM for a given station was Mean Squared Error of 0.3293 on the training set and 1.6267 on the validation set. Comparable results were obtained for 2 different stations, which suggests that a similar process of data preprocessing and building the model may work for all stations in the city.

The visualization results of station overflow can be accessed in full in the coefficient-maps directory of the repostitory as interactive maps (with the option to see the coefficient values for a given station). As a figure, these were as follows:

The results suggest, that the station most viable for expansion is Metro Centrum Nauki Kopernik, since it achieved the highest possible coefficient value for all coefficient definitions.

4 Failed approaches

4.1 Convolutional LSTM

Part of our research was dedicated to combining spatial relations between the stations and temporal dependencies. One of our ideas was to encode the current state of bike usage in a given moment as an image. For each timestamp, we computed three metrics:

- bike count on a given station
- how many bikes were just rented
- how many bikes were just returned

and encoded those metric as a RGB channed in a image. We treated each station as pixel and we tried to resemble actual location of the bike station on Warsaw map. Here (Figure 12)is an exemplary picture that served as an input to our algorithm.

As we see the pixels resemble the shape of Warsaw. The pixel in the top left corner represents the date. It consists of the month, weekday and hour encoded in RGB channel. This way the model can also learn seasonality from the data. We divided the dataset into test and training dataset, with the data from the last 10 days as the test dataset. We aim at predicting those 3 metrics. For that we used a combination of convolutional neural neural network and LSTM network. The architecture was as follows:

- ConvLstm2D layer- to capture both spaial and temporal dependecies
- Flatten layer - to prepare inputs for the next layer
- Repeat layer - also to prepare inputs for the next layer
- LSTM layer - to once again capture the temporal dependencies
- 2x Dense layer - to draw predictions the from previous layer.

Unfortunately, we model would not learn the dependencies. Usually it predicted a straight line near the average usage, but it would not learn when peaks occur. Figure 13 a prediction for one of the stations:

As we identified that the network is underfitted, we tried to increase the complexity of the model, by increasing or adding new layers and training the model for more epochs than before. As those approaches also failed, we tried to encode the data in a different way. One idea was to encode the date in all pixels, by switching to 6 channels instead of 3. We hoped that this way the network will focus more the seasonality. We also tried to include only the "bike count" metric, as it appeared to be the most stable. Unfortunately none of those approaches lead to improving the results. One of the reasons might be that we had data from 2 months only. Training the network on more data can lead to better results.

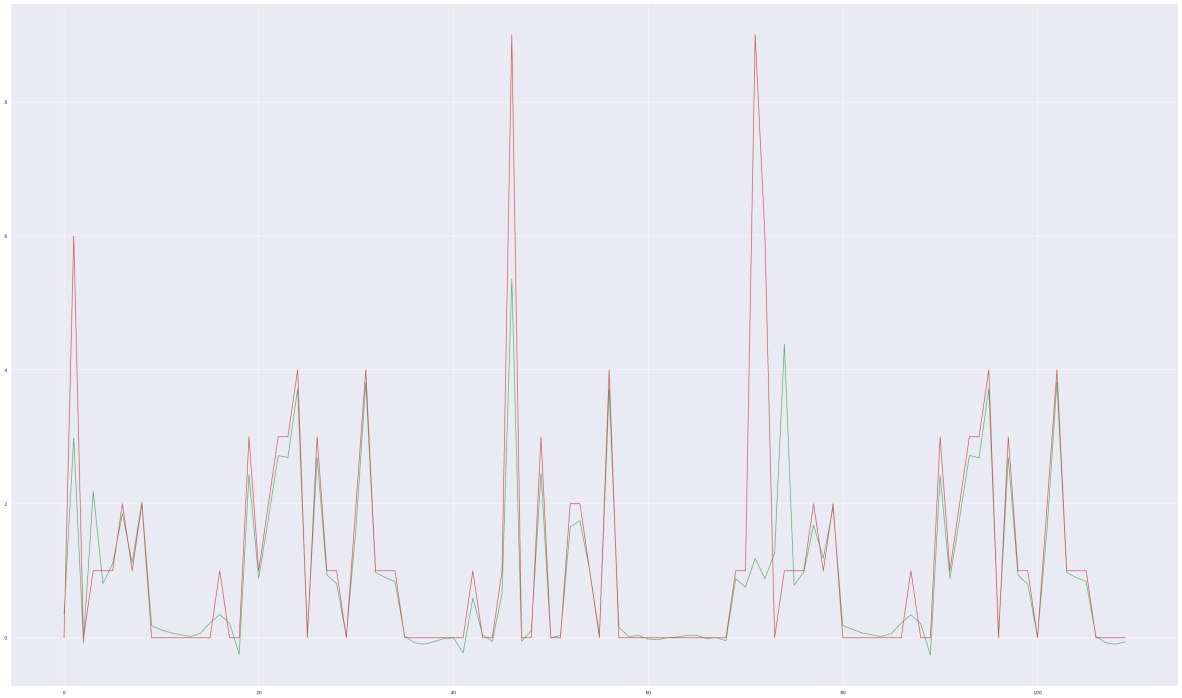
5 Discussion

To sum up the bike demand forecasting, although we haven't managed to obtain satisfactory results for a model that would combine the data from all stations in the city, we have a recipe for model development dedicated to a single station. This result may indicate that the bike demand on a given

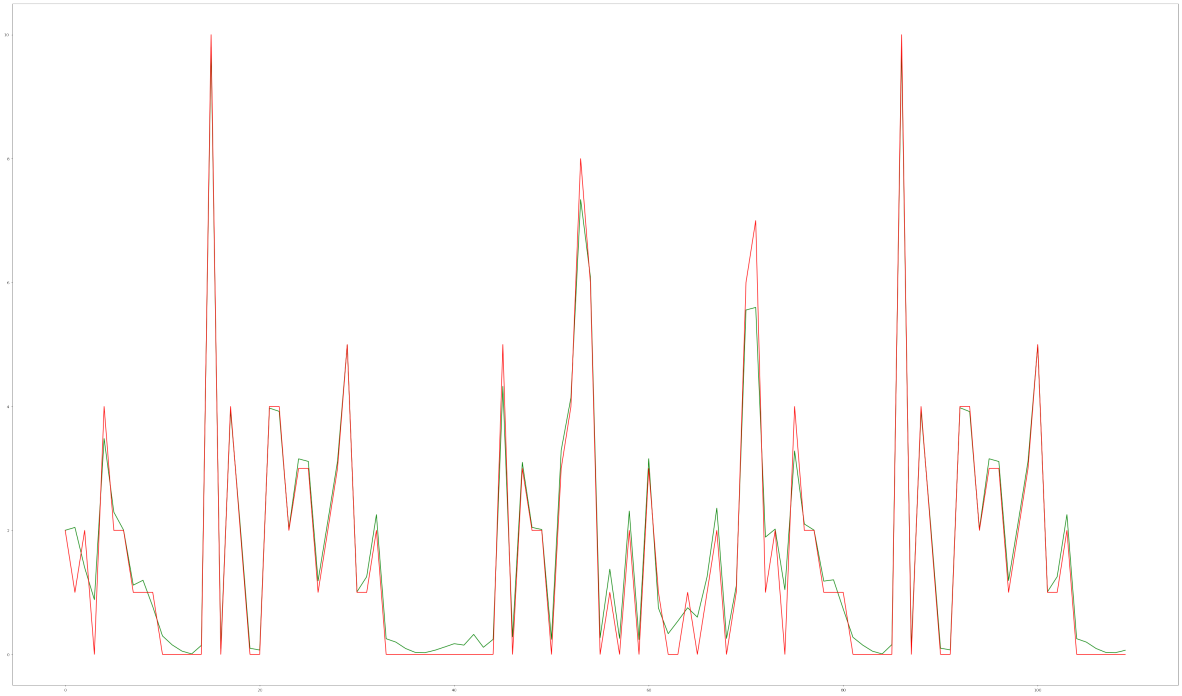
station is not very reliant on the demand on other stations. Additionally, it's important to note as adversarial factors both the interference of Veturilo, which occasionally moves the bikes from one station to another and therefore interferes with our results in ways which are hard to forecast, and the short time of data scraping (3 months), which may cause worse performance in models that have difficulty reflecting trends, like XGBoost.

As for the development of multiple models or a singular model for each station, the additional difficulty may lie in the high complexity that such a model would need. The time window approach in LSTM currently assumes, that the training data is sorted according to time, which may not be designed for datasets with multiple measurements for a given timestamps.

The overflow coefficient results, on the other hand, suggest Metro Centrum Nauki Kopernik to be the most viable station for expansion. Other stations, though they might be frequently overflowing, do not have as high of a variance and therefore may achieve more extreme results as defined by other coefficients due to them, for example, overflowing due to the natural order of people going to work and coming back from work.



(a) Station A



(b) Station B

Figure 8: Predicted results (green) plotted with actual results (red) for the LSTM

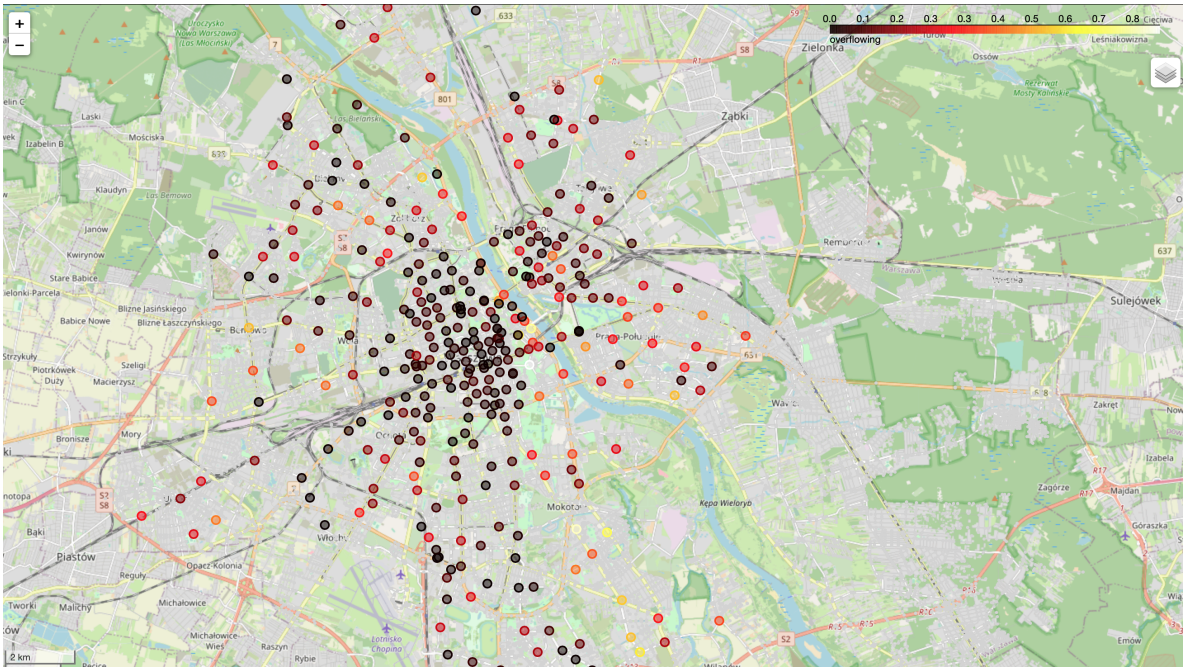


Figure 9: Results of the first coefficient for Warsaw stations

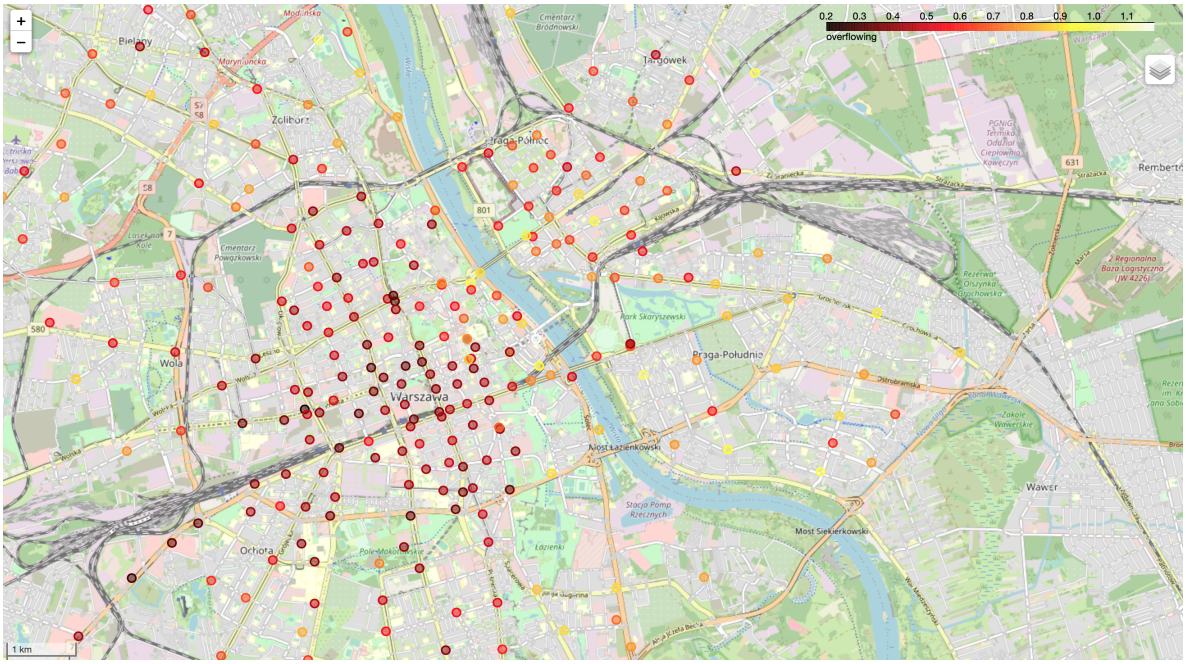


Figure 10: Results of the second coefficient for Warsaw stations

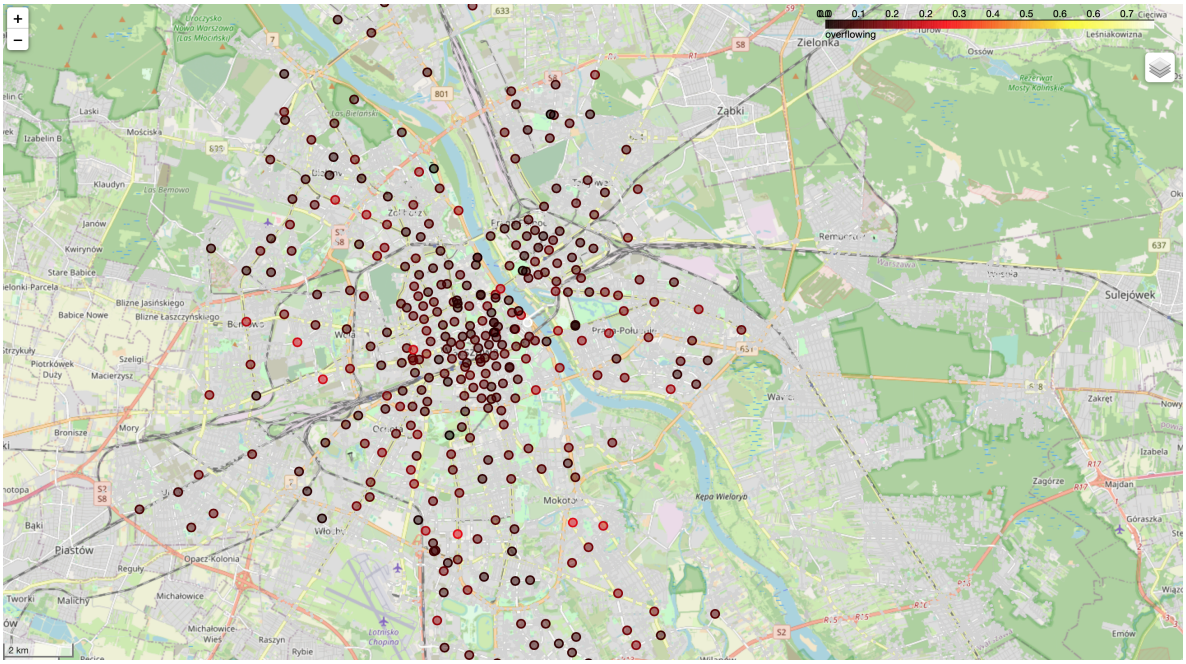


Figure 11: Results of the third coefficient for Warsaw stations

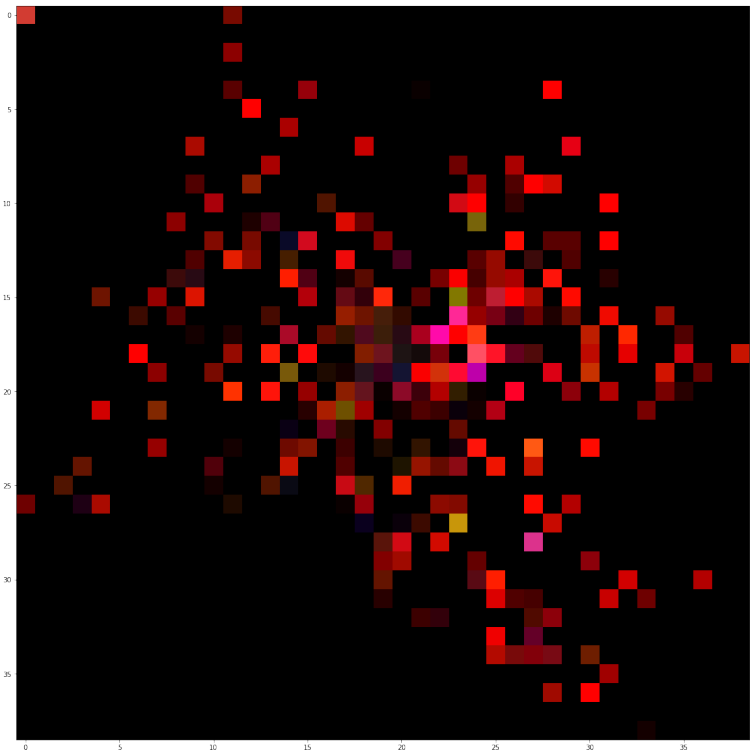


Figure 12: Snapshot of the data for a given moment encoded as an image

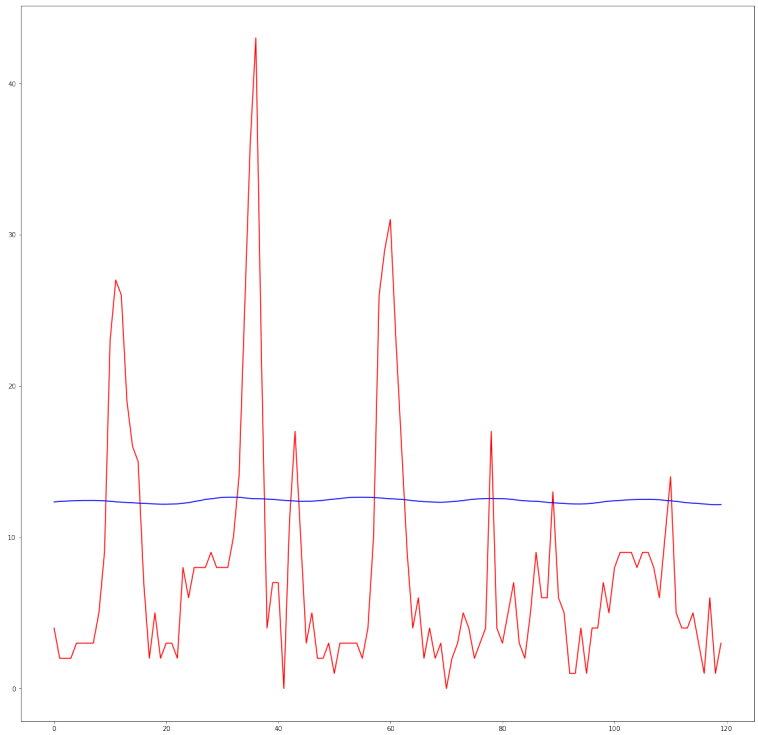


Figure 13: Result of the approach of combining an CNN and LSTM