# Visitor Prediction for the Library of buisness administration (BWI)

**Lecture:** Selected Topics in AI
**Lecturer:** Prof. Dr. Johannes Maucher

**Group:**


- Firaz Ilhan (fi007)
- Patryk Gadziomski (pg058)


**Project description:**

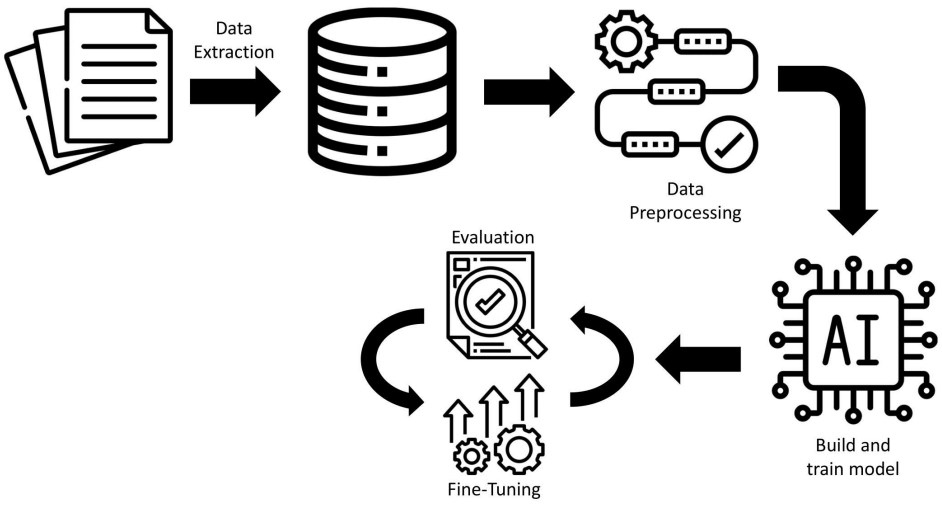
For the project in "Selected Topics of AI" we decided to make a visitor forecast for the BWI library (library of the Institute of Business Administration) at the University of Stuttgart.

In the BWI library, the number of visitors is recorded every day (Monday - Friday). This is done in the form of "visitor statistics". In the visitor statistics, visitors are recorded in eight different time periods: `09:30`, `10:45`, `13:45`, `16:15`, `17:45`, `18:00`, `18:30` and `18:45`. Due to the library's opening hours (Mon. - Thurs. 09:00-19:00; Fri. 09:00 - 17:00), the last three time periods are not recorded on Fridays.

**Data:** Visitor statistics from the BWI library, which are available in analogue form. Since the data is recorded by hand on paper, the data must be digitized for the first time. This is done by manually filling in a pre-generated Excel table (see `data/dataset_creator.ipynb`).

**Goal:** The goal of this project, and the AI model, is to predict the number of visitors for the coming day.

**Method:** To achieve the goal, an LSTM model is trained and fine-tuned.



Source: own creation (Patryk Gadziomski)

# 0. Import Requirements and set the device

In the following step, all necessary libraries for the project are imported.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import datetime
from meteostat import Point, Daily
import holidays
import torch
import torch.nn as nn
from torchinfo import summary
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from copy import deepcopy as dc
```

The GPU is used to execute the model and for training. If this is not available, the CPU is used instead.

In [2]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
device
```

Out[2]:
```
'cpu'
```

# 1. Data Exploration

In this step, the data is loaded and examined in more detail. The data is self-collected data from the BWI library of the University of Stuttgart. These represent the number of visitors to the BWI library.

At first glance, it is clear that the data contains a large number of "missing values".

These must be filled in in the next steps.

In [3]:
```
# Read excel file as dataframe: visitor_data
visitor_data = pd.read_excel("data/bwi_library_visitor_data.xlsx")
visitor_data.head()
```

Out[3]:

| | timestemp | value |
|---|---|---|
| **0** | 2022-03-28 09:30:00 | NaN |
| **1** | 2022-03-28 10:45:00 | NaN |
| **2** | 2022-03-28 13:45:00 | NaN |
| **3** | 2022-03-28 16:15:00 | NaN |
| **4** | 2022-03-28 17:45:00 | NaN |

Using the `info()` function, the missing values are emphasized again by the difference in the count values.

You can also see that the feature `timestemp` is of the type `datetime64`, which cannot be transferred to a model input in this way.

In [4]:
```
visitor_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4896 entries, 0 to 4895
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype         
---  ------     --------------  -----         
 0   timestemp  4896 non-null   datetime64[ns]
 1   value      3782 non-null   float64       
dtypes: datetime64[ns](1), float64(1)
memory usage: 76.6 KB
```

In order to have a fixed number that represents the number of missing values, the `isnull()` function is used and added up. The result is the number of missing values in `visitor_data`.

In [5]:
```
visitor_data.isnull().sum()
```

Out[5]:
```
timestemp       0
value        1114
dtype: int64
```

With the `describe()`-function, you get an overview of the earliest and the last recorded date (`min` and `max`). The other values of the date are not important. The `value` values are of no use due to the many missing values. This will change after preprocessing.

In [6]:
```
visitor_data.describe()
```

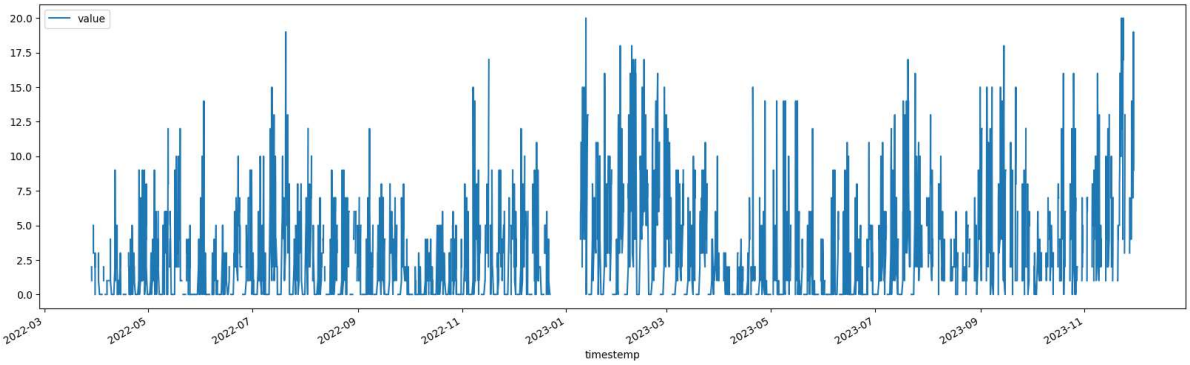
Out[6]:

| | timestemp | value |
|---|---|---|
| **count** | 4896 | 3782.000000 |
| **mean** | 2023-01-28 03:24:22.500000 | 2.870968 |
| **min** | 2022-03-28 09:30:00 | 0.000000 |
| **25%** | 2022-08-28 05:48:45 | 0.000000 |
| **50%** | 2023-01-28 02:07:30 | 2.000000 |
| **75%** | 2023-06-29 22:26:15 | 5.000000 |
| **max** | 2023-11-29 18:45:00 | 20.000000 |
| **std** | NaN | 3.526116 |

The following figure shows the progression of visitor numbers over time. The missing values already mentioned can also be seen. A larger gap can also be seen in the December-January period. This indicates the winter vacations.

In [7]:
```
visitor_data.plot(x="timestemp", y="value", figsize=(20, 6))
```

Out[7]:
```
<Axes: xlabel='timestemp'>
```




# 2. Data Preprocessing

The preprocessing step is about filling in the missing values. To fill in the missing values, the `backfill` method is used, which fills in the missing values based on the past values.

In addition to the `backfill` method, the `interpolate()`, `backfill(limit=7)` and `forward()` methods were also tried.

The best results were obtained using the `backfill` method without `limit`.

In the old Notebokk the command:

```
visitor_data = visitor_data['value'].fillna(value=None,
method='backfill', axis=None, limit=None, downcast=None)
```

is used. However, this is outdated and would cause errors in future pandas versions. For this reason, it has been replaced by the updated command:

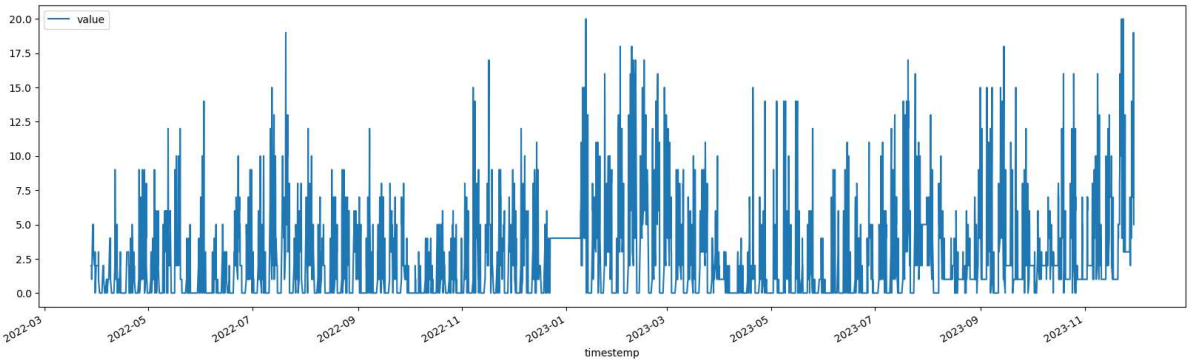
```
visitor_data['value'] = visitor_data['value'].bfill(axis=None)
```

In [8]:
```
# Missing Values Handling
visitor_data['value'] = visitor_data['value'].bfill(axis=None)
```

The following figure shows the progression of visitor numbers over time, after preprocessing and therefore without missing values.

In [9]:
```
visitor_data.plot(x="timestemp", y="value", figsize=(20, 6))
```

Out[9]:
```
<Axes: xlabel='timestemp'>
```




# 3. Feature Selection and further Preprocessing

Further features are extracted using the existing features ("timestemp"):

- Date: The calendar date
- Weekday: Regular visitor patterns can be identified by the day of the week
- Weekend: The library is closed at weekends, so there will be no record of visitors
- Exam periods: During examination periods, there can be an increase in library use as students study and prepare for their exams
- Holiday: During the holidays, the library is closed
- t-1 Sequences: Historical data can provide context. The sequences containing the values for the last two weeks are created for input into the LSTM model
- Weather data: Weather conditions can have an impact on a student's decision to visit the library, with adverse weather conditions potentially leading to a reduction in footfall

In [10]:
```
# Extract the date
date_list = []

for i in visitor_data["timestemp"]:
    new_date = (str(i)[0:10])
    new_date = datetime.datetime.strptime(new_date,'%Y-%m-%d')
    date_list.append(new_date)

visitor_data["date"] = date_list
visitor_data.head()
```

Out[10]:

| | timestemp | value | date |
|---|---|---|---|
| **0** | 2022-03-28 09:30:00 | 2.0 | 2022-03-28 |
| **1** | 2022-03-28 10:45:00 | 2.0 | 2022-03-28 |
| **2** | 2022-03-28 13:45:00 | 2.0 | 2022-03-28 |
| **3** | 2022-03-28 16:15:00 | 2.0 | 2022-03-28 |
| **4** | 2022-03-28 17:45:00 | 2.0 | 2022-03-28 |

In [11]:
```
# add column "weekday"
visitor_data["weekday"] = [weekday for weekday in visitor_data["timestemp"].dt.dayo
visitor_data.head()
```

Out[11]:

| | timestemp | value | date | weekday |
|---|---|---|---|---|
| **0** | 2022-03-28 09:30:00 | 2.0 | 2022-03-28 | 0 |
| **1** | 2022-03-28 10:45:00 | 2.0 | 2022-03-28 | 0 |
| **2** | 2022-03-28 13:45:00 | 2.0 | 2022-03-28 | 0 |
| **3** | 2022-03-28 16:15:00 | 2.0 | 2022-03-28 | 0 |
| **4** | 2022-03-28 17:45:00 | 2.0 | 2022-03-28 | 0 |

In [12]:
```
# Check if weekday or weekend
weekend_list = []

for timestemp in visitor_data['weekday']:
    if timestemp == 5 or timestemp == 6:
        weekend_list.append(1)
    else:
        weekend_list.append(0)
```

```
visitor_data['weekend'] = weekend_list
visitor_data.head()
```

Out[12]:

| | timestemp | value | date | weekday | weekend |
|---|---|---|---|---|---|
| **0** | 2022-03-28 09:30:00 | 2.0 | 2022-03-28 | 0 | 0 |
| **1** | 2022-03-28 10:45:00 | 2.0 | 2022-03-28 | 0 | 0 |
| **2** | 2022-03-28 13:45:00 | 2.0 | 2022-03-28 | 0 | 0 |
| **3** | 2022-03-28 16:15:00 | 2.0 | 2022-03-28 | 0 | 0 |
| **4** | 2022-03-28 17:45:00 | 2.0 | 2022-03-28 | 0 | 0 |

In [13]:
```
# Set all values, where weekend day is 1 to 0
visitor_data.loc[visitor_data['weekend'].eq(1), 'value'] = 0
```

In [14]:
```
# Lecture phase = 0, exam period = 1
exam_phase_ws_2021_begin = datetime.datetime(2022, 2, 10)
exam_phase_ws_2021_end = datetime.datetime(2022, 3, 31)

exam_phase_ss_2022_begin = datetime.datetime(2022, 7, 20)
exam_phase_ss_2022_end = datetime.datetime(2022, 9, 30)

exam_phase_ws_2022_begin = datetime.datetime(2023, 2, 10)
exam_phase_ws_2022_end = datetime.datetime(2023, 3, 31)

exam_phase_ss_2023_begin = datetime.datetime(2023, 7, 20)
exam_phase_ss_2023_end = datetime.datetime(2023, 9, 30)

exam_phase_ws_2023_begin = datetime.datetime(2024, 2, 10)
exam_phase_ws_2023_end = datetime.datetime(2024, 3, 31)
```

In [15]:
```
# Exam period
exam_phase_list = []

# check if the timestamp falls within any of the specified exam phases
for timestemp in visitor_data['timestemp']:
    if timestemp >= exam_phase_ws_2021_begin and timestemp <= exam_phase_ws_2021_er
        exam_phase_list.append(1)
    elif timestemp >= exam_phase_ss_2022_begin and timestemp <= exam_phase_ss_2022_
        exam_phase_list.append(1)
    elif timestemp >= exam_phase_ws_2022_begin and timestemp <= exam_phase_ws_2022_
        exam_phase_list.append(1)
    elif timestemp >= exam_phase_ss_2023_begin and timestemp <= exam_phase_ss_2023_
        exam_phase_list.append(1)
    elif timestemp >= exam_phase_ws_2023_begin and timestemp <= exam_phase_ws_2023_
        exam_phase_list.append(1)
    else:
        exam_phase_list.append(0)

exam_phase_list

# attach the exam_phase_list to the visitor_data DataFrame
visitor_data['exam_phase'] = exam_phase_list
visitor_data.head()
```

Out[15]:

| | timestemp | value | date | weekday | weekend | exam_phase |
|---|---|---|---|---|---|---|
| **0** | 2022-03-28 09:30:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 |
| **1** | 2022-03-28 10:45:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 |
| **2** | 2022-03-28 13:45:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 |
| **3** | 2022-03-28 16:15:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 |
| **4** | 2022-03-28 17:45:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 |

In [16]:

```
de_holidays = holidays.country_holidays('DE', subdiv='BW')

holiday_array = []

# Check if the date is a public holiday in BaWü
# holiday = 1, else = 0
for date in visitor_data['date']:
    if date in de_holidays:
        holiday_array.append(1)
    else:
        holiday_array.append(0)

# add a new column holiday to the visitor_data DataFrame
visitor_data['holiday'] = holiday_array

# Update the 'value' column in visitor_data
visitor_data.loc[visitor_data['holiday'].eq(1), 'value'] = 0
visitor_data.head()
```

Out[16]:

| | timestemp | value | date | weekday | weekend | exam_phase | holiday |
|---|---|---|---|---|---|---|---|
| **0** | 2022-03-28 09:30:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 | 0 |
| **1** | 2022-03-28 10:45:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 | 0 |
| **2** | 2022-03-28 13:45:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 | 0 |
| **3** | 2022-03-28 16:15:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 | 0 |
| **4** | 2022-03-28 17:45:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 | 0 |

In [17]:

```
# Set the values of the winter vacations to 0, and holiday to 1

wv_begin_2022 = datetime.datetime(2022, 12, 23)
wv_end_2022 = datetime.datetime(2023, 1, 8)

wv_begin_2023 = datetime.datetime(2023, 12, 23)
wv_end_2023 = datetime.datetime(2024, 1, 7)

wv_list = []

for timestemp in visitor_data['timestemp']:
    if timestemp >= wv_begin_2022 and timestemp <= wv_end_2022:
        wv_list.append(1)
    elif timestemp >= wv_begin_2023 and timestemp <= wv_end_2023:
        wv_list.append(1)
    else:
        wv_list.append(0)

visitor_data['winter_vacation'] = wv_list

visitor_data.loc[visitor_data['winter_vacation'].eq(1), 'value'] = 0
```

```
visitor_data.loc[visitor_data['winter_vacation'].eq(1), 'holiday'] = 1

visitor_data.head()
```

Out[17]:

| | timestemp | value | date | weekday | weekend | exam_phase | holiday | winter_vacation |
|---|---|---|---|---|---|---|---|---|
| **0** | 2022-03-28 09:30:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 | 0 | 0 |
| **1** | 2022-03-28 10:45:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 | 0 | 0 |
| **2** | 2022-03-28 13:45:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 | 0 | 0 |
| **3** | 2022-03-28 16:15:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 | 0 | 0 |
| **4** | 2022-03-28 17:45:00 | 2.0 | 2022-03-28 | 0 | 0 | 1 | 0 | 0 |

This prepares the data frame for the LSTM model. The model looks at 112 time records, which is 2 weeks, to produce a forecast.

In [18]:

```
# Get the historical values of t-1 ... t-112
def prepare_dateframe_for_lstm(df, n_steps):
    """
    Parameters:
    df (DataFrame): The original DataFrame with time series data
    n_steps (int): The number of lag steps to create

    Returns:
    DataFrame: A modified DataFrame with lagged features for LSTM.
    """
    # use a copy instead of the original data frame object
    df = df.copy()

    cols = [df['value'].shift(i).rename(f'value (t-{i})') for i in range(1, n_steps

    df = pd.concat([df] + cols, axis=1)

    df.dropna(inplace=True)

    return df

# we have to go back 8*14=112 values if we want to go back two weeks.
lookback = 112
visitor_data = prepare_dateframe_for_lstm(visitor_data, lookback)

visitor_data.head()
```

Out[18]:

| | timestemp | value | date | weekday | weekend | exam_phase | holiday | winter_vacation | value (t-1) |
|---|---|---|---|---|---|---|---|---|---|
| **112** | 2022-04-11 09:30:00 | 2.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 0.0 |
| **113** | 2022-04-11 10:45:00 | 9.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 2.0 |
| **114** | 2022-04-11 13:45:00 | 5.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 9.0 |
| **115** | 2022-04-11 16:15:00 | 5.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 5.0 |
| **116** | 2022-04-11 17:45:00 | 1.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 5.0 |

5 rows × 120 columns

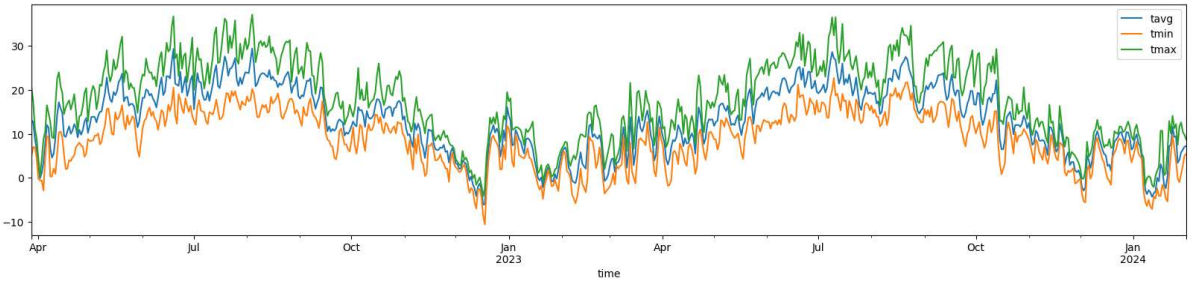
In [19]:

```
# Get the weather data until today
today_year = int(str(datetime.datetime.today())[0:4])
today_month = int(str(datetime.datetime.today())[5:7])
today_day = int(str(datetime.datetime.today())[8:10])

# get weather data thorught the date
start = datetime.datetime(2022, 3, 28)
end = datetime.datetime(today_year, today_month, today_day)

# get the location for Stuttgart
location = Point(48.7823200, 9.1770200, 252)

# fetch the weather data
weather_data = Daily(location, start, end)
weather_data = weather_data.fetch()

"""
time: The date string (format: YYYY-MM-DD)      : String
tavg: The average air temperature in °C         : Float
tmin: The minimum air temperature in °C         : Float
tmax: The maximum air temperature in °C         : Float
prcp: The daily precipitation total in mm       : Float
snow: The maximum snow depth in mm                 : Integer
wdir: The average wind direction in degrees (°) : Integer
wspd: The average wind speed in km/h            : Float
wpgt: The peak wind gust in km/h                   : Float
pres: The average sea-level air pressure in hPa : Float
tsun: The daily sunshine total in minutes (m)   : Integer
"""
weather_data.head()
```

Out[19]:

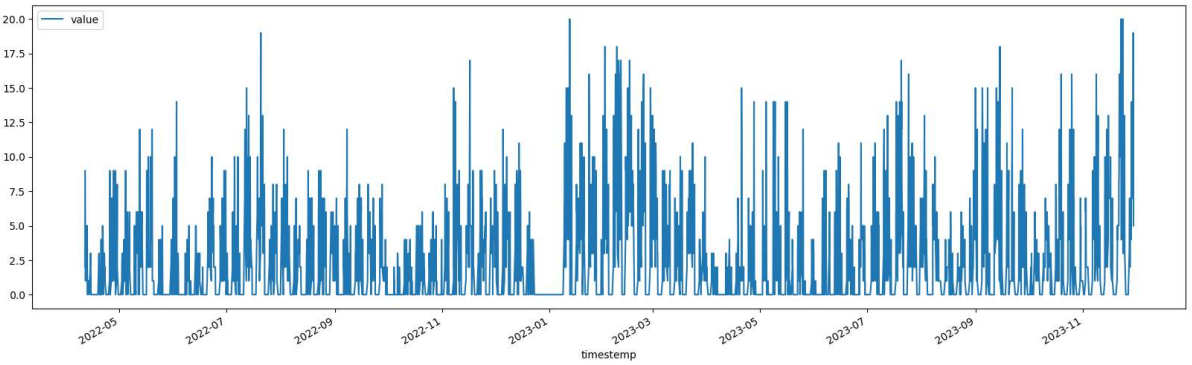
| time | tavg | tmin | tmax | prcp | snow | wdir | wspd | wpgt | pres | tsun |
|---|---|---|---|---|---|---|---|---|---|---|
| **2022-03-28** | 13.2 | 4.9 | 20.7 | 0.0 | 0.0 | 160.0 | 10.8 | 30.6 | 1021.4 | 630.0 |
| **2022-03-29** | 12.6 | 7.0 | 17.4 | 1.8 | 0.0 | 133.0 | 8.6 | 22.7 | 1009.8 | 210.0 |
| **2022-03-30** | 9.0 | 6.9 | 11.2 | 0.8 | 0.0 | 300.0 | 7.2 | 30.6 | 1001.8 | 0.0 |
| **2022-03-31** | 6.4 | 4.5 | 9.0 | 14.9 | 0.0 | 2.0 | 9.7 | 31.7 | 996.7 | 3.0 |
| **2022-04-01** | 1.9 | -0.2 | 4.5 | 8.7 | 0.0 | 337.0 | 12.2 | 28.1 | 1001.3 | 0.0 |

In [20]:
```
weather_data.plot(y=['tavg', 'tmin', 'tmax'], figsize=(20, 4))
plt.show()
```




In [21]:
```
visitor_data.plot(x="timestemp", y="value", figsize=(20, 6))
```

Out[21]:
```
<Axes: xlabel='timestemp'>
```




The data runs from March 2022 to December 2023. The graph shows the fluctuations in visitor numbers over time, with some periods having higher visitor numbers and others having lower or no visitor numbers, such as during the winter holidays.

In [22]:
```
visitor_data.describe()
```

Out[22]:

| | timestemp | value | date | weekday | weekend | exam_phase |
|---|---|---|---|---|---|---|
| **count** | 4783 | 4783.000000 | 4783 | 4783.000000 | 4783.000000 | 4783.00000 |
| **mean** | 2023-02-04 01:54:27.886263808 | 2.421911 | 2023-02-03 10:30:07.902989824 | 2.990174 | 0.284340 | 0.32281 |
| **min** | 2022-04-11 09:30:00 | 0.000000 | 2022-04-11 00:00:00 | 0.000000 | 0.000000 | 0.00000 |
| **25%** | 2022-09-07 17:00:00 | 0.000000 | 2022-09-07 00:00:00 | 1.000000 | 0.000000 | 0.00000 |
| **50%** | 2023-02-03 18:45:00 | 1.000000 | 2023-02-03 00:00:00 | 3.000000 | 0.000000 | 0.00000 |
| **75%** | 2023-07-03 15:00:00 | 4.000000 | 2023-07-03 00:00:00 | 5.000000 | 1.000000 | 1.00000 |
| **max** | 2023-11-29 18:30:00 | 20.000000 | 2023-11-29 00:00:00 | 6.000000 | 1.000000 | 1.00000 |
| **std** | NaN | 3.340391 | NaN | 2.001178 | 0.451147 | 0.46760 |

8 rows × 120 columns

# 4. JOIN both dataframes (visitor_data + weather_data) and calculate correlations

In [23]:
```
# merge data frames
visitor_data_with_weather = visitor_data.merge(
    weather_data,
    left_on="date",
    right_on="time"
)

visitor_data_with_weather.head()
```

Out[23]:

| | timestemp | value | date | weekday | weekend | exam_phase | holiday | winter_vacation | value (t-1) | val (t |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2022-04-11 09:30:00 | 2.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 0.0 | |
| **1** | 2022-04-11 10:45:00 | 9.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 2.0 | |
| **2** | 2022-04-11 13:45:00 | 5.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 9.0 | |
| **3** | 2022-04-11 16:15:00 | 5.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 5.0 | |
| **4** | 2022-04-11 17:45:00 | 1.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 5.0 | |

5 rows × 130 columns

With `set_index('timestemp', inplace=True)` we set the feature `timestemp` as the index of the rows and thus remove it from the features that are later passed as input.

Since we realized later in the notebook that the weather data is not relevant, we also set `timestemp` of `visitor_data` as index so that we can use this DataFrame again later.

In [24]:
```
visitor_data.set_index('timestemp', inplace=True)
visitor_data_with_weather.set_index('timestemp', inplace=True)
visitor_data_with_weather.head()
```

Out[24]:

| timestemp | value | date | weekday | weekend | exam_phase | holiday | winter_vacation | value (t-1) | value (t-2) |
|---|---|---|---|---|---|---|---|---|---|
| **2022-04-11 09:30:00** | 2.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 0.0 | 0.0 |
| **2022-04-11 10:45:00** | 9.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 2.0 | 0.0 |
| **2022-04-11 13:45:00** | 5.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 9.0 | 2.0 |
| **2022-04-11 16:15:00** | 5.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 5.0 | 9.0 |
| **2022-04-11 17:45:00** | 1.0 | 2022-04-11 | 0 | 0 | 0 | 0 | 0 | 5.0 | 5.0 |

5 rows × 129 columns

In [25]:
```
#calculate correlations between all features of data frame
data_corr = visitor_data_with_weather.corr()
data_corr
```

Out[25]:

| | value | date | weekday | weekend | exam_phase | holiday | winter_vacation | |
|---|---|---|---|---|---|---|---|---|
| **value** | 1.000000 | 0.106489 | -0.413847 | -0.457059 | 0.123916 | -0.172433 | -0.120241 | |
| **date** | 0.106489 | 1.000000 | 0.003070 | 0.003880 | 0.089429 | -0.055974 | -0.033567 | |
| **weekday** | -0.413847 | 0.003070 | 1.000000 | 0.790623 | 0.003391 | -0.054540 | 0.016353 | -0 |
| **weekend** | -0.457059 | 0.003880 | 0.790623 | 1.000000 | -0.006960 | -0.051050 | 0.010351 | -0 |
| **exam_phase** | 0.123916 | 0.089429 | 0.003391 | -0.006960 | 1.000000 | -0.164185 | -0.114489 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **wdir** | -0.021816 | 0.052963 | 0.028722 | 0.024208 | 0.001665 | 0.072697 | 0.013864 | -0 |
| **wspd** | 0.026588 | 0.120266 | -0.022200 | -0.003549 | 0.003103 | 0.069806 | 0.112650 | |
| **wpgt** | -0.013010 | 0.094966 | -0.017266 | -0.003433 | 0.057805 | 0.097528 | 0.118859 | -0 |
| **pres** | 0.023017 | -0.159301 | 0.011614 | 0.054377 | -0.060715 | 0.088855 | 0.073145 | |
| **tsun** | -0.066452 | -0.180748 | -0.013643 | 0.004177 | 0.083467 | -0.055036 | -0.128761 | -0 |

129 rows × 129 columns

In [26]:
```
data_corr_label = visitor_data_with_weather.iloc[:, :].corr()["value"]
data_corr_label
```

Out[26]:
```
value         1.000000
date          0.106489
weekday      -0.413847
weekend      -0.457059
exam_phase    0.123916
                ...
wdir         -0.021816
wspd          0.026588
wpgt         -0.013010
pres          0.023017
tsun         -0.066452
Name: value, Length: 129, dtype: float64
```

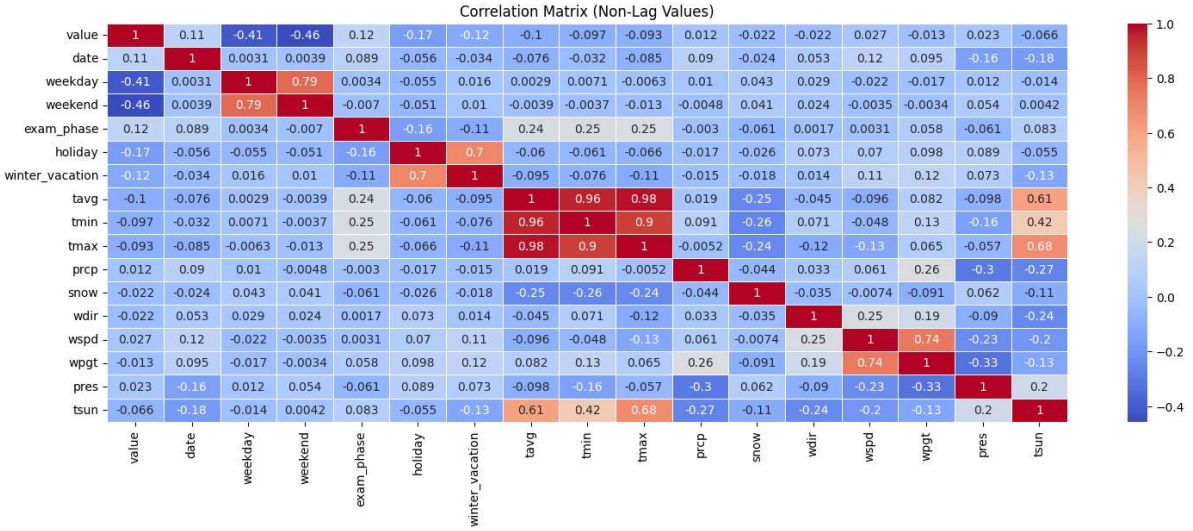
The correlation matrix shows the correlation between the features.

Key findings:

- tavg, tmin and tmax are correlated with each other, which means that they provide the same information and can be reduced to a single feature.
- As the library is closed on weekends, the number of visitors on these days is zero. This results in a negative correlation, as the number of visitors on weekends is always zero.
- Given that the library is open on weekdays, the correlation coefficient of -0.41 is interesting. As the weekday is defined as a numerical value, it's unlikely that the number of visitors will change linearly. This could mean that visitors prefer certain days of the week or different opening times during the week.

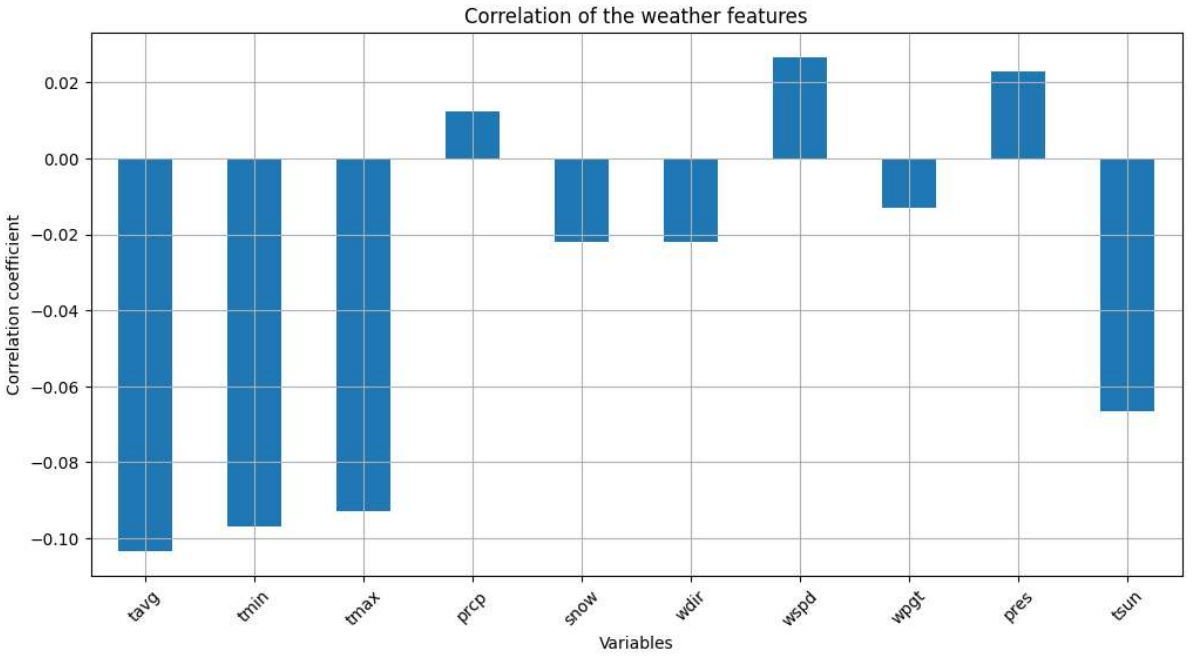
In [27]:
```
# filtering out non-lagged columns from the correlation matrix
# by selecting columns whose names do not contain "(t-"
non_lag_columns = [col for col in data_corr.columns if "(t-" not in col]

data_corr_non_lag = data_corr[non_lag_columns].loc[non_lag_columns]
```

```
plt.figure(figsize=(18, 6))
sns.heatmap(data_corr_non_lag, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title('Correlation Matrix (Non-Lag Values)')
plt.show()
```




The correlation between the visitor numbers and the weather data is is relatively low. The highest correlation of the weather data is the average temperature with a value of -0.1.

In [28]:
```
plt.figure(figsize=(12, 6))
data_corr_label["tavg":].plot(kind='bar')
plt.title('Correlation of the weather features')
plt.xlabel('Variables')
plt.ylabel('Correlation coefficient')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```




Weather data is removed from the data frame as it has a very low correlation.

In the following code, three features are removed:

- `date` : This feature is a feature of type `datetime64` , which cannot be transferred to the network in this form
- `winter_vacation` : This feature was only used to integrate the winter vacations into the `holiday` -future
- `weekend` : As this feature already exists in a different form in the `weekday` -feature, there is a co-correlation

In [29]:
```
# drop unnecessary features
droplist = ['date', 'winter_vacation', 'weekend']
visitor_data = visitor_data.drop(droplist, axis=1)

visitor_data.head()
```

Out[29]:

| timestemp | value | weekday | exam_phase | holiday | value (t-1) | value (t-2) | value (t-3) | value (t-4) | value (t-5) | value (t-6) | ... | v |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **2022-04-11 09:30:00** | 2.0 | 0 | 0 | 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | |
| **2022-04-11 10:45:00** | 9.0 | 0 | 0 | 0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | |
| **2022-04-11 13:45:00** | 5.0 | 0 | 0 | 0 | 9.0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | |
| **2022-04-11 16:15:00** | 5.0 | 0 | 0 | 0 | 5.0 | 9.0 | 2.0 | 0.0 | 0.0 | 0.0 | ... | |
| **2022-04-11 17:45:00** | 1.0 | 0 | 0 | 0 | 5.0 | 5.0 | 9.0 | 2.0 | 0.0 | 0.0 | ... | |

5 rows × 116 columns

# 5. Prepare Data for Model

In [30]:
```
visitor_data.head()
```

Out[30]:

| timestemp | value | weekday | exam_phase | holiday | value (t-1) | value (t-2) | value (t-3) | value (t-4) | value (t-5) | value (t-6) | ... | v |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **2022-04-11 09:30:00** | 2.0 | 0 | 0 | 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | |
| **2022-04-11 10:45:00** | 9.0 | 0 | 0 | 0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | |
| **2022-04-11 13:45:00** | 5.0 | 0 | 0 | 0 | 9.0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | |
| **2022-04-11 16:15:00** | 5.0 | 0 | 0 | 0 | 5.0 | 9.0 | 2.0 | 0.0 | 0.0 | 0.0 | ... | |
| **2022-04-11 17:45:00** | 1.0 | 0 | 0 | 0 | 5.0 | 5.0 | 9.0 | 2.0 | 0.0 | 0.0 | ... | |

5 rows × 116 columns

In [31]:
```
visitor_data.shape
```

Out[31]:
```
(4783, 116)
```

We need to scale the data so that they have the same range. We have decided to use a `MinMaxScaler()`. This scales the data to a range from `0` to `1`. By scaling, we bring all features to the same scale.

In [32]:
```
# Scaling the data
scaler = MinMaxScaler(feature_range=(0, 1))
new_visitor_data = scaler.fit_transform(visitor_data)

new_visitor_data
```

Out[32]:
```
array([[0.1       , 0.        , 0.        , ..., 0.1       , 0.1       ,
        0.1       ],
       [0.45      , 0.        , 0.        , ..., 0.1       , 0.1       ,
        0.1       ],
       [0.25      , 0.        , 0.        , ..., 0.1       , 0.1       ,
        0.1       ],
       ...,
       [0.45      , 0.33333333, 0.        , ..., 0.35      , 0.35      ,
        0.35      ],
       [0.25      , 0.33333333, 0.        , ..., 0.35      , 0.35      ,
        0.35      ],
       [0.25      , 0.33333333, 0.        , ..., 0.35      , 0.35      ,
        0.35      ]])
```

In [33]:
```
# Split into features and labels

data_features = new_visitor_data[:, 1:]
data_labels = new_visitor_data[:, 0]
```

```
data_features.shape, data_labels.shape
```

Out[33]:
```
((4783, 115), (4783,))
```

In [34]:
```
data_labels
```

Out[34]:
```
array([0.1 , 0.45, 0.25, ..., 0.45, 0.25, 0.25])
```

The order of the `t-1` values is currently `t-1` - `t-112`. Therefore, we should flip the order of the `t-1` features. To analyze future data, we need to reorder the `t-1` features to start from the past and progress towards the future.

In [35]:
```
# Flip the data because of the order of the sequence
data_features = dc(np.flip(data_features, axis=1))
data_features
```

Out[35]:
```
array([[0.1       , 0.1       , 0.1       , ..., 0.        , 0.        ,
        0.        ],
       [0.1       , 0.1       , 0.1       , ..., 0.        , 0.        ,
        0.        ],
       [0.1       , 0.1       , 0.1       , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.35      , 0.35      , 0.35      , ..., 0.        , 0.        ,
        0.33333333],
       [0.35      , 0.35      , 0.35      , ..., 0.        , 0.        ,
        0.33333333],
       [0.35      , 0.35      , 0.35      , ..., 0.        , 0.        ,
        0.33333333]])
```

In [36]:
```
# Train test Split
split_index = int(len(data_features) * 0.8)

train_features = data_features[:split_index]
test_features = data_features[split_index:]

train_labels = data_labels[:split_index]
test_labels = data_labels[split_index:]
```

In [37]:
```
train_features.shape, train_labels.shape, test_features.shape, test_labels.shape
```

Out[37]:
```
((3826, 115), (3826,), (957, 115), (957,))
```

In [38]:
```
# Reshape the arrays into 3-dimeniosnal arrays.
# LSTM-Models requires a 3-dimensional input.
goback = len(visitor_data.columns) -1

train_features = train_features.reshape((-1, goback, 1))
test_features = test_features.reshape((-1, goback, 1))

train_labels = train_labels.reshape((-1, 1))
test_labels = test_labels.reshape((-1, 1))

train_features.shape, test_features.shape, train_labels.shape, test_labels.shape
```

Out[38]:
```
((3826, 115, 1), (957, 115, 1), (3826, 1), (957, 1))
```

In [39]:
```
# Turn the data into tensors
train_features = torch.tensor(train_features).float()
train_labels = torch.tensor(train_labels).float()
```

```
test_features = torch.tensor(test_features).float()
test_labels = torch.tensor(test_labels).float()

train_features.shape, test_features.shape, train_labels.shape, test_labels.shape
```

Out[39]:
```
(torch.Size([3826, 115, 1]),
 torch.Size([957, 115, 1]),
 torch.Size([3826, 1]),
 torch.Size([957, 1]))
```

# 6. Dataset and DataLoader

In [40]:
```
class TimeSeriesDataset(Dataset):
    """
    This class is for a PyTorch DataLoader to create batches of data.
    It stores features and labels of the time series data and
    provides methods to access them.

    X: the tensor containing the features of the dataset
    y: the tensor containing the labels of the dataset
    """
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        """
        Returns the number of samples in the dataset.
        """
        return len(self.X)

    def __getitem__(self, i):
        """
        Retrieves the i-th sample from the dataset.
        """
        return self.X[i], self.y[i]

train_dataset = TimeSeriesDataset(train_features, train_labels)
test_dataset = TimeSeriesDataset(test_features, test_labels)
```

In [41]:
```
train_dataset, test_dataset
```

Out[41]:
```
(<__main__.TimeSeriesDataset at 0x19482589910>,
 <__main__.TimeSeriesDataset at 0x194824eee10>)
```

With a lookback of 112 steps, we choose a relatively small batch size of 8 to avoid memory problems. With more memory available, the batch size could be increased to 16. The data set is divided into 80% training and 20% test data.

In [42]:
```
BS = 8

train_loader = DataLoader(train_dataset, batch_size=BS, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BS, shuffle=False)

for x, y in train_loader:
    print(f"Input Batch Shape: {x.shape} - (batch size, Features, scalar label)")
    print(f"Label Batch Shape: {y.shape} - (batch size, label dimension)")
    break
```

```
Input Batch Shape: torch.Size([8, 115, 1]) - (batch size, Features, scalar label)
Label Batch Shape: torch.Size([8, 1]) - (batch size, label dimension)
```

# 7. Build Model

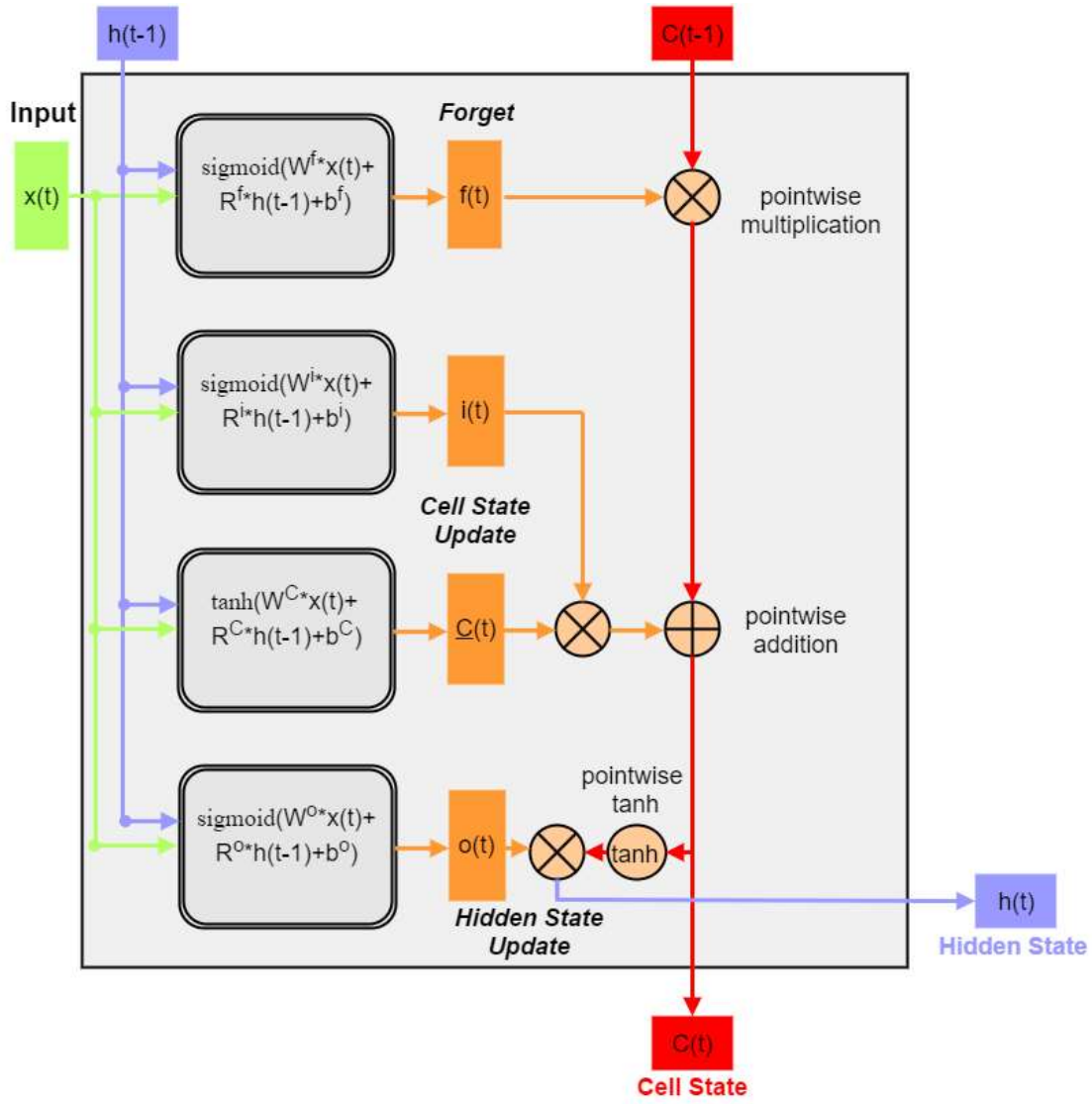
## 7.1. LSTM Model Architecture

An LSTM is specifically designed to recognise and learn from long-term dependencies. The problem with simple recurrent neural networks is that they do not retain information over a long period of time, the gradient of the loss function tends to become smaller or larger as it propagates back through many layers. The problem is that the network can hardly learn due to the low weights in the earlier layers, or the network becomes unstable.

The most important thing about the LSTM is the vertical red line in the diagram. This is the cell state that memorises the patterns of the students.

Steps:

- The first layer decides which information from the cell state should be discarded. The sigmoid function outputs values between 0 and 1, where 0 means no information is passed and 1 means all information is passed. Initially the LSTM doesn't have any previous cell state so it passes a vector of zeros.
- The next layer decides what new information to add to the cell state. The next layer, which is also a sigmoid function, determines which values in the cell state are updated. At the same time, the tanh layer generates a vector of new candidate values that could potentially be added to the state. These two aspects work together to prepare an update of the cell state.
- In the last layer, the LSTM decides what to output, which is based on the filtered cell state. A sigmoid layer determines which parts of the cell state are output. The cell state is then passed through a tanh function, which scales the values between -1 and 1, and multiplied by the output.




Quelle: https://griesshaber.pages.mi.hdm-stuttgart.de/nlp/07neuralnetworks/02RecurrentNeuralNetworks.html

In [43]:
```
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_layer_size, output_size, num_layers):
        super(LSTMModel, self).__init__()

        self.input_size = input_size
        self.num_layers = num_layers
        self.hidden_layer_size = hidden_layer_size
        self.output_size = output_size

        self.lstm = nn.LSTM(input_size=self.input_size, # input size
                            hidden_size=self.hidden_layer_size, # hidden layer
                            num_layers=self.num_layers #stacked layers
                            )
        self.linear = nn.Linear(self.hidden_layer_size, self.output_size)

        self.hidden_cell = (
            torch.zeros(self.num_layers, 1, self.hidden_layer_size),
            torch.zeros(self.num_layers, 1, self.hidden_layer_size),
        )

    def forward(self, input_seq):
        """
        Defines the forward pass of the LSTM model and returns
```

```
        the output predictions of the model.
        """

        # reshape the input sequence for LSTM
        input_seq_reshaped = input_seq.view(len(input_seq), 1, -1)

        # get the output from the LSTM layers
        lstm_out, self.hidden_cell = self.lstm(input_seq_reshaped, self.hidden_cell

        # reshape the LSTM output for linear layer
        lstm_out_reshaped = lstm_out.view(len(input_seq), -1)

        # pass through the linear layer for a prediction
        predictions = self.linear(lstm_out_reshaped)

        return predictions
```

## 7.2. Training

- The output_size is set to 1, as the model should predict a regression
- By trial and error, we determined the dimension of the hidden state and the learning_rate
- We use a stacked LSTM with 2 layers to allow for greater model complexity
- We use the Adam optimiser because it dynamically adjusts learning rates and often converges more quickly

In [44]:
```
loss_list = []
input_size = goback
hidden_layer_size = 100
output_size = 1
num_layers = 2
learning_rate = 0.001

model = LSTMModel(input_size, hidden_layer_size, output_size, num_layers)

loss_function = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

In [45]:
```
summary(model, input_size=(8, 115))
```

Out[45]:
```
==========================================================================================
========
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
========
LSTMModel                                [8, 1]                    --
├─LSTM: 1-1                              [8, 1, 100]               167,600
├─Linear: 1-2                            [8, 1]                    101
==========================================================================================
========
Total params: 167,701
Trainable params: 167,701
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 1.34
==========================================================================================
========
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.67
Estimated Total Size (MB): 0.68
==========================================================================================
========
```

For the training loop we used `epochs = 3`, because the general min-loss after the third epoch doesn't fall lower than the loss in the third epoch. The general loss over all epochs becomes lower, but not the minimum value.

In [46]:
```python
epochs = 3

print(f"Using: {device} for training")

for epoch in range(epochs):
    model.train()

    # reset the hidden state at each loop
    model.hidden_cell = (
        torch.zeros(num_layers, 1, model.hidden_layer_size),
        torch.zeros(num_layers, 1, model.hidden_layer_size),
    )

    for i, (x_batch, y_batch) in enumerate(train_loader):

        x_batch, y_batch = x_batch.to(device), y_batch.to(device)

        # reset the gradients
        optimizer.zero_grad()

        # prevent backpropagating of the dataset
        model.hidden_cell = (
            model.hidden_cell[0].detach(),
            model.hidden_cell[1].detach(),
        )

        # forward pass
        y_pred = model(x_batch.float())

        # compute loss
        loss = loss_function(y_pred, y_batch.float())
        loss_list.append(loss.detach().numpy())

        # backward pass
        loss.backward()
        optimizer.step()
```

```
    print(f"Epoch {epoch+1}/{epochs} Loss: {loss.item()}")

print("Training complete")
```

```
Using: cpu for training
Epoch 1/3 Loss: 0.002281684661284089
Epoch 2/3 Loss: 0.00023948917805682868
Epoch 3/3 Loss: 5.104986121295951e-05
Training complete
```

## 7.3. Validation

To validate the results, we tested the model together with test data using three metrics:

- **Mean Squared Error**

  The mean squared error calculates the average squared difference between the actual and the predicted value.

- **Root mean squared error**

  The Root Mean Squared Error calculates the square root of the average squared difference between the current and the predicted value.

- **Mean Absolute Error**

  The Mean Absolute Error calculates the average absolute difference between the current and the predicted value.

### MSE, Root MSE, MAE

In [47]:
```
total_loss = 0
count = 0
actuals = []
predictions = []
test_loss = []

with torch.no_grad():
    model.train(False)
    for x_batch, y_batch in test_loader:

        y_pred = model(x_batch.float())

        predictions.extend(y_pred.view(-1).tolist())
        actuals.extend(y_batch.view(-1).tolist())

        loss = loss_function(y_pred, y_batch.float())
        total_loss += loss.item()
        test_loss.append(loss.item())
        count += 1

mse = total_loss / count
rmse = mse ** 0.5
actuals_np = np.array(actuals)
predictions_np = np.array(predictions)

mae = mean_absolute_error(actuals_np, predictions_np)
```

```
print(f"Mean Squared Error on test data: {mse}")
print(f"Root Mean Squared Error on test data: {rmse}")
print(f"Mean Absolute Error on test data: {mae}")
```

```
Mean Squared Error on test data: 0.017305774612759707
Root Mean Squared Error on test data: 0.13155141433203865
Mean Absolute Error on test data: 0.10335973930295046
```

Compared to the average number of visitors (including weekends and public holidays), the error metrics appear to be relatively low. In other words, the model's predictions are relatively accurate compared to the average number of visitors.

In [48]:
```
average_visitors = visitor_data['value'].mean()
print(f'Average Visitors: {round(average_visitors, 2)}')
```

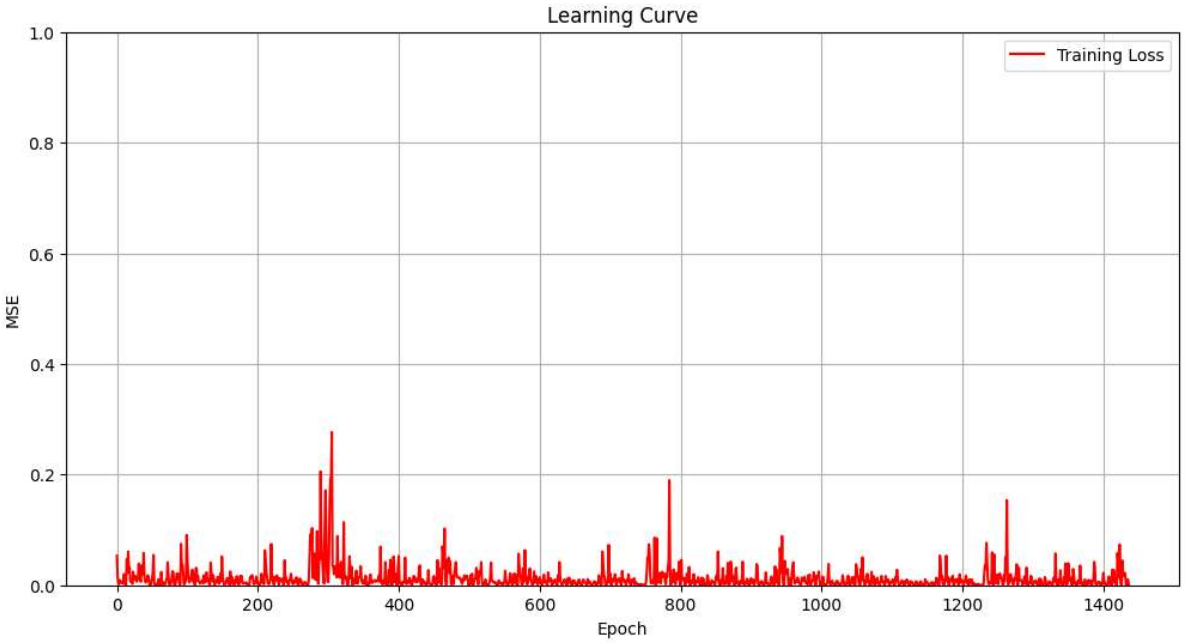
```
Average Visitors: 2.42
```

## 7.4. Evaluation

In the evaluation we will have a look at the performance of the model. We will look at the learning curve for the training and test data and graphically compare the predicted values with the actual values.

Here we can see the Learning Curve for the Training date, with the Training Loss.

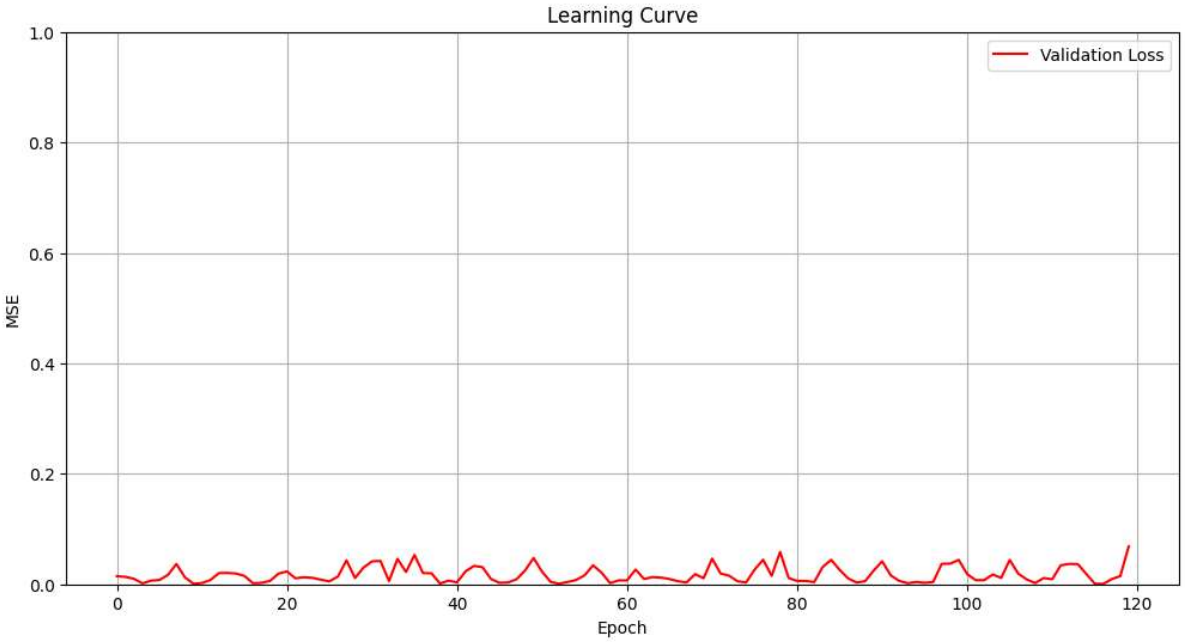
In [49]:
```
plt.figure(figsize=(12,6))
plt.plot(range(len(loss_list)),loss_list,"r",label="Training Loss")
plt.axis([None, None, 0, 1])
plt.xlabel("Epoch")
plt.ylabel("MSE")
plt.grid(True)
plt.legend()
plt.title("Learning Curve")
plt.show()
```




Here we can see the Learning Curve for the Test date, with the Validation Loss.

In [50]:
```
plt.figure(figsize=(12,6))
plt.plot(range(len(test_loss)),test_loss,"r",label="Validation Loss")
```

```
plt.axis([None, None, 0, 1])
plt.xlabel("Epoch")
plt.ylabel("MSE")
plt.grid(True)
plt.legend()
plt.title("Learning Curve")
plt.show()
```




For the following graphic to work and show us the values on the same scale, we first have to bring all values ( `test_predictions` and `test_features` ) to the same scale. the reason is that we have scaled the values with the `MinMaxScaler()` to a range from `0` to `1` . In order to display the values properly, we must therefore scale the values back to their original form using `.inverse_transform()` .

Inverse scaling of the `test_predictions` :

In [51]:
```
test_predictions = model(test_features).detach().numpy().flatten()

pred_dummies = np.zeros((test_features.shape[0], goback+1))
pred_dummies[:, 0] = test_predictions
pred_dummies = scaler.inverse_transform(pred_dummies)

test_predictions = dc(pred_dummies[:, 0])
test_predictions
```

Out[51]:
```
array([ 7.15670764e+00,  7.15341568e+00,  7.25207031e+00,  6.58038139e+00,
        3.77212942e+00,  3.57428610e+00,  3.45477164e+00,  6.25654459e+00,
        6.82031095e+00,  7.56991923e+00,  5.28577089e+00,  4.64342117e+00,
        3.51957798e+00,  3.46360326e+00,  3.35654259e+00,  3.83893132e+00,
        3.02533805e+00,  2.44656488e+00,  1.85349837e+00,  2.40852103e+00,
        2.07919165e+00,  1.48850501e+00,  7.06712008e-01,  6.80394471e-01,
        4.72126603e-01,  4.01607901e-01,  4.92338613e-01,  1.15415625e+00,
        1.11905932e+00,  8.61113518e-01,  4.27679271e-01,  1.08489051e+00,
        1.14673279e+00,  1.47966981e+00,  9.74783078e-01,  1.02669150e+00,
        8.33495930e-01,  4.45971414e-01,  2.42575511e+00,  5.02872586e+00,
        7.92969882e+00,  7.46743679e+00,  5.13836622e+00,  5.62853217e+00,
        3.96894157e+00,  3.74901086e+00,  4.00819391e+00,  6.99556172e+00,
        7.54904985e+00,  8.20370197e+00,  7.41241455e+00,  6.34260297e+00,
        5.50606310e+00,  5.20392299e+00,  5.12850106e+00,  7.36151576e+00,
        9.11687374e+00,  8.69503796e+00,  7.89611459e+00,  6.51283383e+00,
        5.33033669e+00,  5.52593350e+00,  4.58556056e+00,  5.15325069e+00,
        5.72905242e+00,  5.91013610e+00,  5.04106462e+00,  4.87695545e+00,
        3.46982241e+00,  2.55982369e+00,  1.76947474e+00,  1.88235402e+00,
        1.99678853e+00,  1.73099279e+00,  1.30692691e+00,  1.48536339e+00,
        1.76068082e+00,  1.34037942e+00,  4.60279286e-01,  4.77427728e-01,
        6.34475872e-01,  7.97000676e-01,  8.72814581e-01,  1.05682872e+00,
        1.27565533e+00,  1.31173745e+00,  6.11525923e-01,  9.00129154e-01,
        1.24999635e+00,  1.15230985e+00,  9.01772305e-01,  1.09719917e+00,
        1.08828962e+00,  1.02509156e+00,  2.82299578e+00,  4.81397748e+00,
        5.79329312e+00,  6.43328309e+00,  5.97931027e+00,  5.05344808e+00,
        3.95618558e+00,  3.27616036e+00,  2.67582864e+00,  4.04238105e+00,
        6.07529998e+00,  6.78850114e+00,  5.52215278e+00,  4.72515792e+00,
        3.93140078e+00,  3.24592888e+00,  2.11950287e+00,  3.04976135e+00,
        4.49695200e+00,  6.02378130e+00,  4.66530502e+00,  4.14707422e+00,
        3.39612722e+00,  2.64148772e+00,  1.97783455e+00,  2.52138495e+00,
        2.98538089e+00,  4.51765746e+00,  4.64099407e+00,  3.79815996e+00,
        3.00192982e+00,  2.25747332e+00,  1.36330083e+00,  1.45046577e+00,
        1.93936422e+00,  2.26637155e+00,  2.19826296e+00,  1.82019100e+00,
        1.76249370e+00,  1.72202155e+00,  1.16389357e+00,  1.00714855e+00,
        1.01889484e+00,  1.02470897e+00,  7.50693008e-01,  8.95578787e-01,
        1.18319586e+00,  1.27658635e+00,  9.33519974e-01,  1.00809015e+00,
        1.47747591e+00,  1.57865584e+00,  1.30519897e+00,  9.44965035e-01,
        8.80794227e-01,  7.88271502e-01,  2.43047401e+00,  3.94322693e+00,
        5.26648521e+00,  6.16788149e+00,  5.55722475e+00,  3.74952197e+00,
        3.36427033e+00,  2.88882256e+00,  2.06063807e+00,  2.55301923e+00,
        4.10101593e+00,  4.92803514e+00,  4.58845735e+00,  4.30094957e+00,
        3.61792147e+00,  2.93061674e+00,  2.02487782e+00,  2.32637659e+00,
        3.87538522e+00,  6.21027231e+00,  6.57629192e+00,  4.99785244e+00,
        3.76494586e+00,  2.75386482e+00,  2.08551243e+00,  3.47252905e+00,
        6.19896173e+00,  7.87004232e+00,  6.54661119e+00,  3.96812141e+00,
        3.45383406e+00,  3.27614546e+00,  2.89280385e+00,  3.33462238e+00,
        4.64847684e+00,  5.77783287e+00,  4.91459012e+00,  3.12943757e+00,
        2.63941139e+00,  2.55397558e+00,  2.16996714e+00,  2.02419534e+00,
        2.48196051e+00,  2.47368500e+00,  1.87211901e+00,  1.51242912e+00,
        1.89797416e+00,  1.78507537e+00,  1.00277252e+00,  7.60459602e-01,
        1.24913707e+00,  1.57637894e+00,  1.36475101e+00,  1.02061749e+00,
        1.23860434e+00,  1.21092029e+00,  2.71068305e+00,  3.99558544e+00,
        7.35672116e+00,  9.11438584e+00,  7.38349378e+00,  5.73382139e+00,
        4.69599515e+00,  4.43607330e+00,  3.81234646e+00,  4.63142276e+00,
        7.85757542e+00,  7.44626045e+00,  5.92587113e+00,  3.98216605e+00,
        3.44888896e+00,  2.95190811e+00,  2.94291407e+00,  4.68516588e+00,
        7.29387581e+00,  8.63086820e+00,  7.60628879e+00,  7.32405901e+00,
        7.14417815e+00,  6.56615436e+00,  5.57016492e+00,  6.54689193e+00,
        1.06609571e+01,  1.16073811e+01,  9.98662949e+00,  1.09253430e+01,
        9.25611317e+00,  8.47813368e+00,  7.83161223e+00,  6.52805448e+00,
        8.19020867e+00,  1.00447977e+01,  1.17977381e+01,  1.13666785e+01,
        1.06569302e+01,  9.77887392e+00,  5.24616420e+00,  4.55070436e+00,
        4.94175255e+00,  5.93256950e+00,  5.74968934e+00,  4.56395268e+00,
        3.27765167e+00,  2.34921619e+00,  1.65052921e+00,  1.63511977e+00,
```

```
2.30051413e+00,  2.05820546e+00,  1.58541933e+00,  1.30292892e+00,
8.17689002e-01,  9.22011510e-01,  2.77927071e+00,  4.49340880e+00,
7.11576700e+00,  1.01160467e+01,  1.06483495e+01,  9.32543337e+00,
8.55496585e+00,  7.33342826e+00,  7.30909705e+00,  6.38989985e+00,
8.49432230e+00,  1.11044037e+01,  1.07656229e+01,  8.66865098e+00,
7.04862237e+00,  5.65843999e+00,  4.93888140e+00,  6.35878086e+00,
1.05706167e+01,  1.22403312e+01,  1.22544551e+01,  1.01188254e+01,
7.38445401e+00,  5.00877917e+00,  4.40911353e+00,  6.57838821e+00,
1.03512526e+01,  1.14534986e+01,  1.14982343e+01,  8.75642955e+00,
7.24542499e+00,  4.95654881e+00,  2.97547817e+00,  3.31441462e+00,
4.68114853e+00,  4.99895751e+00,  4.38782632e+00,  3.80350709e+00,
2.98868954e+00,  1.66454881e+00,  5.89601994e-01,  4.03557830e-01,
3.03510986e-01, -1.54959597e-02,  1.20454598e-01,  6.17728792e-01,
1.20022409e+00,  1.12104200e+00,  6.21439219e-01,  9.45546106e-01,
1.04621060e+00,  9.50168148e-01,  7.68364444e-01,  8.88660625e-01,
9.22462940e-01,  9.51922759e-01,  2.18697771e+00,  3.63585770e+00,
6.70735359e+00,  7.99164653e+00,  6.03241086e+00,  5.49046218e+00,
4.45312053e+00,  4.03635740e+00,  3.08381200e+00,  4.77127314e+00,
6.65019155e+00,  7.70581603e+00,  8.64996731e+00,  6.04190230e+00,
5.16622126e+00,  4.32100832e+00,  3.40631127e+00,  5.01549959e+00,
8.33179355e+00,  9.10992384e+00,  1.03049815e+01,  8.71587396e+00,
7.48706400e+00,  6.49800897e+00,  5.25091946e+00,  5.06137192e+00,
9.73176479e+00,  1.28803003e+01,  1.17755258e+01,  8.74335766e+00,
7.44410634e+00,  5.20643353e+00,  2.17174456e+00,  3.07032645e+00,
5.35590351e+00,  6.75499201e+00,  5.12807250e+00,  2.89342105e+00,
2.04693362e+00,  1.13081150e+00,  4.10372242e-01,  8.22907686e-01,
1.21050172e+00,  9.17423964e-01,  4.22745571e-01,  9.56227928e-01,
1.43917814e+00,  1.64668515e+00,  3.48570794e-01,  4.87690195e-01,
1.31403983e+00,  1.46718830e+00,  4.92030233e-01,  1.08309709e+00,
1.42000154e+00,  1.17653824e+00,  1.78759173e+00,  4.44134474e+00,
6.24950051e+00,  7.26753831e+00,  6.04424834e+00,  5.79618931e+00,
5.36322474e+00,  3.91273737e+00,  1.97775155e+00,  5.01364708e+00,
9.48788583e+00,  1.12529540e+01,  8.32370341e+00,  6.31321430e+00,
4.13476557e+00,  3.11036646e+00,  1.72471166e+00,  4.38139796e+00,
6.71106339e+00,  8.65365624e+00,  4.94213849e+00,  4.09725457e+00,
3.18817317e+00,  2.67392874e+00,  1.75041020e+00,  3.75536501e+00,
7.43241847e+00,  9.31260586e+00,  8.33210707e+00,  6.40097260e+00,
5.51114798e+00,  5.00773907e+00,  2.54294693e+00,  3.57778192e+00,
4.19740379e+00,  4.78005439e+00,  3.54574353e+00,  2.94170141e+00,
2.39423111e+00,  1.51512399e+00,  3.97309922e-01,  4.39329520e-01,
6.34089932e-01,  5.69335148e-01, -6.50385953e-02,  5.11729941e-01,
9.41440538e-01,  9.90521163e-01,  2.91001648e-01,  6.61755651e-01,
6.27308562e-01,  7.07021803e-01,  5.12997881e-01,  9.09345299e-01,
1.04701050e+00,  7.20207989e-01,  1.50021300e+00,  3.91838253e+00,
6.71816468e+00,  8.53195012e+00,  5.64923763e+00,  3.92915577e+00,
3.34876657e+00,  2.69256532e+00,  1.59963861e+00,  4.48938012e+00,
6.21605992e+00,  9.34470713e+00,  5.57580948e+00,  4.30939913e+00,
2.81537503e+00,  2.08225355e+00,  1.22071601e+00,  3.43227327e+00,
6.45915449e+00,  9.65453327e+00,  4.53447729e+00,  3.92367542e+00,
2.94743448e+00,  2.48910248e+00,  2.22347647e+00,  6.05330825e+00,
8.59807968e+00,  9.11679268e+00,  5.22593021e+00,  4.50041324e+00,
3.69331360e+00,  2.14906484e+00,  1.29346475e+00,  2.59055972e+00,
4.28854287e+00,  4.33543444e+00,  2.69170672e+00,  2.15603039e+00,
2.20831573e+00,  1.60559699e+00,  1.25207439e+00,  1.26540437e+00,
1.98489293e+00,  1.98825911e+00,  1.32870868e+00,  1.81035340e+00,
2.04181567e+00,  1.52544752e+00,  1.05192065e+00,  1.40555426e+00,
2.13041633e+00,  1.95287853e+00,  1.03952169e+00,  1.29143476e+00,
1.49825543e+00,  8.74901190e-01,  2.15113282e+00,  4.92750466e+00,
7.21992075e+00,  5.81384420e+00,  4.27771568e+00,  3.23494494e+00,
2.84514189e+00,  2.25851208e+00,  1.10296078e+00,  1.90214396e+00,
3.49031061e+00,  3.23575646e+00,  1.51575223e+00,  1.19680971e+00,
1.30934268e+00,  6.11067414e-01,  1.69346109e+00,  4.78262246e+00,
8.82331550e+00,  8.67981434e+00,  6.24853373e+00,  4.87023532e+00,
4.35854346e+00,  2.77870864e+00,  2.40686968e+00,  4.51447219e+00,
```

```
6.50187850e+00,  5.91461062e+00,  4.66755301e+00,  3.93790066e+00,
3.51398647e+00,  2.35365793e+00,  1.63615495e+00,  3.36803734e+00,
5.18391728e+00,  4.37279522e+00,  3.26533437e+00,  2.96729892e+00,
3.20205897e+00,  2.54469246e+00,  1.91887364e+00,  1.70295015e+00,
1.93287194e+00,  1.63714215e+00,  1.52986526e+00,  1.65704027e+00,
1.72486767e+00,  1.16997071e+00,  6.65407479e-01,  1.06563732e+00,
1.59081548e+00,  1.24710955e+00,  1.00994758e+00,  1.08316883e+00,
9.75373313e-01,  6.37111813e-01,  2.24606290e+00,  4.84853834e+00,
6.60218894e+00,  5.39738595e+00,  4.20134246e+00,  3.81062567e+00,
3.56266975e+00,  3.38310003e+00,  3.65739167e+00,  4.90874410e+00,
7.03201413e+00,  7.14764297e+00,  5.59193730e+00,  5.20723164e+00,
4.88302171e+00,  4.64937568e+00,  6.12139463e+00,  8.56458008e+00,
9.85762298e+00,  9.54456627e+00,  7.60503292e+00,  6.04954600e+00,
5.05813837e+00,  4.34088230e+00,  4.23418701e+00,  6.35959983e+00,
8.01906228e+00,  7.08405197e+00,  5.41196704e+00,  4.37876344e+00,
3.44092906e+00,  3.59781027e+00,  4.00751650e+00,  4.81505454e+00,
5.13172090e+00,  3.99440527e+00,  3.75131726e+00,  3.67362529e+00,
3.65890712e+00,  3.36219311e+00,  2.81828225e+00,  2.03099266e+00,
1.45345867e+00,  1.51958868e+00,  1.55081928e+00,  1.65984228e+00,
1.71451420e+00,  1.55397281e+00,  1.01495408e+00,  1.03109635e+00,
9.94260237e-01,  9.13898572e-01,  9.68462005e-01,  1.08136944e+00,
1.07103772e+00,  1.31507948e+00,  3.40727389e+00,  5.46427250e+00,
7.96533823e+00,  9.15417314e+00,  7.95919716e+00,  7.43888974e+00,
5.19806802e+00,  4.86612439e+00,  5.09919345e+00,  6.79080188e+00,
8.60030651e+00,  7.75440335e+00,  6.82084382e+00,  5.11758924e+00,
4.26297367e+00,  3.88399661e+00,  4.41848397e+00,  5.79427004e+00,
7.19988644e+00,  9.78328347e+00,  7.58354902e+00,  6.28184497e+00,
5.09357870e+00,  5.23347020e+00,  6.02164686e+00,  8.91021252e+00,
1.01612890e+01,  1.22043574e+01,  8.12303126e+00,  5.99618614e+00,
4.57926482e+00,  4.47263300e+00,  4.10675943e+00,  6.54835939e+00,
6.31008565e+00,  5.84916174e+00,  4.87164319e+00,  3.47170591e+00,
2.94233680e+00,  2.45637551e+00,  2.10613176e+00,  2.19184175e+00,
2.26322755e+00,  1.77135468e+00,  1.29187375e+00,  1.52813345e+00,
1.61525086e+00,  1.35229662e+00,  1.15150593e+00,  1.19919188e+00,
1.56165227e+00,  1.49075419e+00,  1.08922638e+00,  1.10952668e+00,
1.28327534e+00,  9.32842270e-01,  3.27325225e+00,  6.14305258e+00,
8.53898823e+00,  8.78847241e+00,  6.90257967e+00,  5.39588690e+00,
4.91215765e+00,  4.07215387e+00,  4.29517686e+00,  6.66901708e+00,
9.83579993e+00,  1.04848301e+01,  7.55878925e+00,  5.20024419e+00,
4.18978989e+00,  3.62809688e+00,  4.62682247e+00,  9.69947040e+00,
1.27275717e+01,  1.42556643e+01,  1.10145819e+01,  8.65797162e+00,
7.21668243e+00,  5.65099895e+00,  6.85597062e+00,  8.55357111e+00,
1.18344927e+01,  1.11739838e+01,  7.43922830e+00,  6.29522800e+00,
5.79553604e+00,  3.37769955e+00,  2.84613788e+00,  4.23473656e+00,
5.66033065e+00,  5.11262476e+00,  3.04063678e+00,  2.28801146e+00,
2.44079798e+00,  1.60803184e+00,  8.96704644e-01,  1.23146117e+00,
1.49341360e+00,  1.04561634e+00,  9.94261205e-01,  1.30651072e+00,
1.65394828e+00,  9.70860049e-01,  5.32862470e-01,  6.75066635e-01,
1.96199670e+00,  1.34981751e+00,  9.26890299e-01,  1.07989110e+00,
1.05512835e+00,  5.33869490e-01,  2.41320476e+00,  4.64599848e+00,
7.81907439e+00,  8.56885314e+00,  5.35268307e+00,  4.97367203e+00,
4.88596529e+00,  3.62362593e+00,  3.68517071e+00,  5.95150113e+00,
9.08597112e+00,  6.93058133e+00,  4.87707019e+00,  3.73845339e+00,
3.08754325e+00,  1.78991944e+00,  1.13790967e+00,  2.04581663e+00,
4.43089902e+00,  3.57242107e+00,  1.89139962e+00,  1.36052027e+00,
1.17275946e+00,  1.14009622e-01,  1.22032188e+00,  3.43007565e+00,
7.45401144e+00,  6.48021460e+00,  5.60755253e+00,  3.74614686e+00,
2.98779428e+00,  1.93172738e+00,  1.81227282e+00,  2.56328106e+00,
3.64820480e+00,  3.08066070e+00,  2.47904614e+00,  2.51157552e+00,
2.55543590e+00,  1.85312480e+00,  1.13782428e+00,  7.90302306e-01,
1.18388101e+00,  8.39961842e-01,  1.07576646e+00,  1.62258655e+00,
1.95723787e+00,  1.48337126e+00,  8.57707933e-01,  8.81437212e-01,
1.27179444e+00,  9.41082314e-01,  9.42178145e-01,  1.11081488e+00,
1.09290592e+00,  7.01591149e-01,  2.54008442e+00,  4.78311896e+00,
```

```
8.43604147e+00,  8.92077029e+00,  7.73567915e+00,  6.65828824e+00,
5.10291457e+00,  3.56007636e+00,  4.46166933e+00,  7.53190637e+00,
1.25763190e+01,  1.08902323e+01,  9.75680351e+00,  6.21306956e+00,
4.61674064e+00,  3.15400392e+00,  3.63395214e+00,  7.10024118e+00,
1.22399986e+01,  1.45573819e+01,  1.11149013e+01,  8.73623133e+00,
6.03482008e+00,  5.08157372e+00,  6.35707319e+00,  9.63629544e+00,
1.32504904e+01,  1.33674157e+01,  9.71055984e+00,  6.59069598e+00,
4.88574475e+00,  4.30809647e+00,  4.36883628e+00,  6.00667894e+00,
7.82342553e+00,  5.46120822e+00,  3.60502928e+00,  2.51683831e+00,
2.43587062e+00,  2.47248068e+00,  2.05645517e+00,  2.26058945e+00,
2.42372319e+00,  1.98225424e+00,  1.35752797e+00,  1.75866231e+00,
1.84368297e+00,  1.51274249e+00,  1.17567286e+00,  1.45410642e+00,
2.29399443e+00,  2.29313955e+00,  1.59160137e+00,  1.61408529e+00,
1.51410460e+00,  1.01424538e+00,  2.61452764e+00,  5.51369429e+00,
9.60704684e+00,  1.05565608e+01,  7.92132556e+00,  5.86981416e+00,
4.40937132e+00,  4.28232074e+00,  4.43405569e+00,  7.71380067e+00,
1.08365512e+01,  1.10500181e+01,  8.79439950e+00,  6.89669371e+00,
5.11633396e+00,  4.44518089e+00,  4.56822664e+00,  9.28567648e+00,
1.60518992e+01,  1.29392052e+01,  8.95434201e+00,  7.02850580e+00,
6.56358898e+00,  6.74341440e+00,  7.79074967e+00,  1.13455641e+01,
1.20251155e+01,  9.80335236e+00,  7.52718091e+00,  7.16533482e+00,
6.52604103e+00,  5.68321407e+00,  3.74312967e+00,  4.58208859e+00,
4.85364407e+00,  4.83095586e+00,  4.40369666e+00,  3.22557569e+00,
2.60706991e+00,  1.65550396e+00,  2.74847858e-01,  5.44720814e-01,
1.06744774e+00,  7.45696723e-01,  7.63765424e-01,  1.25996962e+00,
1.37482256e+00,  9.07369480e-01,  3.05536631e-01,  8.60131755e-01,
1.49728164e+00,  1.41951904e+00,  7.77954683e-01,  9.44196582e-01,
1.01211458e+00,  8.30968395e-01,  2.54852623e+00,  6.13514125e+00,
7.45754302e+00,  7.10337996e+00,  5.14321566e+00,  4.10258025e+00,
3.63340259e+00,  3.68442893e+00,  3.84031296e+00,  7.45909750e+00,
1.09002531e+01,  1.03925622e+01,  9.30096984e+00,  9.48259830e+00,
7.38614678e+00,  6.82136297e+00,  7.06309676e+00,  1.12005496e+01,
1.47180676e+01,  1.60380876e+01,  1.47218132e+01,  1.18496454e+01,
9.52584863e+00,  8.54523838e+00,  8.15003693e+00,  1.19206214e+01,
1.64871085e+01,  1.69315922e+01,  1.53950000e+01,  1.34151828e+01,
1.05522668e+01,  1.01745701e+01,  7.16490984e+00,  8.70290160e+00,
9.51856077e+00,  1.04550457e+01,  9.03142333e+00,  5.64043880e+00,
4.18142438e+00,  3.24391216e+00,  2.11783990e+00,  1.45175233e+00,
1.05544709e+00,  7.97815621e-01,  4.09882665e-01,  5.67508712e-01,
8.59843716e-01,  8.62718150e-01,  8.46279599e-02,  1.88631620e-01,
3.48076299e-01,  5.74895553e-02,  5.82488254e-03,  2.90917270e-01,
7.11420625e-01,  9.98860970e-01,  2.31964841e+00,  5.32012045e+00,
7.30211914e+00,  8.29196334e+00,  5.56143463e+00,  4.36772943e+00,
3.51226568e+00,  3.37632269e+00,  3.80782783e+00,  6.89783990e+00,
8.96158814e+00,  1.09520054e+01,  9.64148045e+00,  7.62914062e+00,
6.80516124e+00,  6.94396496e+00,  5.72439790e+00,  7.91882753e+00,
9.57438231e+00,  1.36124468e+01,  1.14422464e+01,  8.33230138e+00,
6.53977394e+00])
```

Inverse scaling of the `test_features`:

In [52]:
```
test_dummies = np.zeros((test_features.shape[0], goback+1))
test_dummies[:, 0] = test_labels.flatten()
test_dummies = scaler.inverse_transform(test_dummies)

new_test_labels = dc(test_dummies[:, 0])
new_test_labels
```

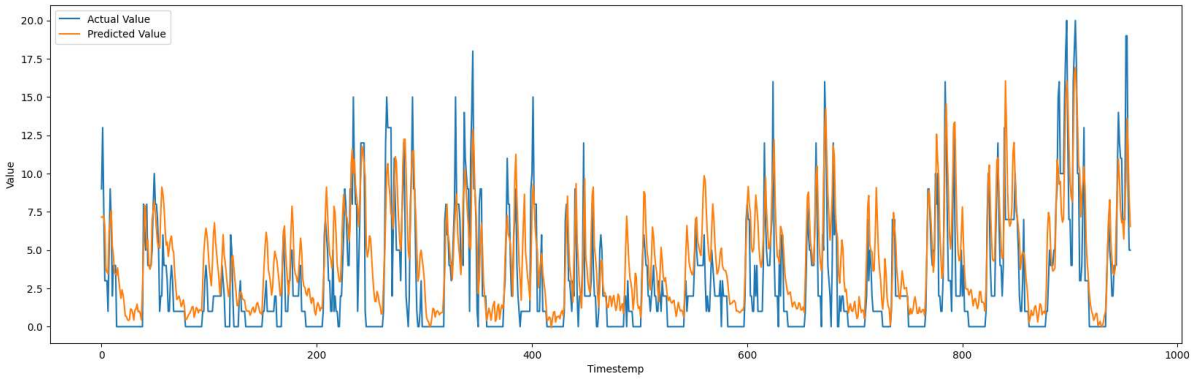
Out[52]:
```
array([ 8.99999976, 12.99999952,  8.00000012,  3.00000012,  3.00000012,
        3.00000012,  1.00000001,  5.        ,  8.99999976,  6.99999988,
        2.00000003,  4.00000006,  4.00000006,  4.00000006,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  8.00000012,
        6.00000024,  5.        ,  8.00000012,  4.00000006,  4.00000006,
        4.00000006,  4.00000006,  5.        ,  8.00000012, 10.        ,
        8.00000012,  8.00000012,  6.99999988,  5.        ,  1.00000001,
        2.00000003,  2.00000003,  6.00000024,  4.00000006,  4.00000006,
        4.00000006,  3.00000012,  1.00000001,  1.00000001,  3.00000012,
        4.00000006,  3.00000012,  1.00000001,  1.00000001,  1.00000001,
        1.00000001,  1.00000001,  1.00000001,  1.00000001,  1.00000001,
        1.00000001,  1.00000001,  1.00000001,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  1.00000001,
        1.00000001,  3.00000012,  4.00000006,  3.00000012,  1.00000001,
        1.00000001,  1.00000001,  1.00000001,  1.00000001,  2.00000003,
        2.00000003,  2.00000003,  2.00000003,  2.00000003,  2.00000003,
        2.00000003,  2.00000003,  4.00000006,  3.00000012,  2.00000003,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        6.00000024,  5.        ,  2.00000003,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  2.00000003,  3.00000012,
        1.00000001,  1.00000001,  1.00000001,  1.00000001,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        1.00000001,  1.00000001,  2.00000003,  3.00000012,  1.00000001,
        1.00000001,  1.00000001,  1.00000001,  1.00000001,  1.00000001,
        1.00000001,  2.00000003,  2.00000003,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  4.00000006,  6.00000024,
        4.00000006,  1.00000001,  1.00000001,  1.00000001,  3.00000012,
        3.00000012,  5.        ,  5.        ,  2.00000003,  2.00000003,
        2.00000003,  2.00000003,  2.00000003,  2.00000003,  4.00000006,
        4.00000006,  1.00000001,  1.00000001,  1.00000001,  1.00000001,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  1.00000001,  6.00000024,  6.99999988,  6.00000024,
        5.        ,  3.00000012,  3.00000012,  1.00000001,  1.00000001,
        5.        ,  1.00000001,  2.00000003,  1.00000001,  1.00000001,
        0.        ,  0.        ,  2.00000003,  2.00000003,  4.00000006,
        6.00000024,  8.99999976,  8.00000012,  6.00000024,  4.00000006,
        4.00000006,  8.99999976,  8.99999976,  8.00000012, 15.        ,
       10.        ,  8.00000012,  8.00000012,  1.00000001,  4.00000006,
        6.99999988, 12.00000048, 12.00000048, 12.00000048, 12.00000048,
        1.00000001,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  1.00000001,  5.        , 12.00000048,
       15.        , 12.99999952, 12.99999952, 12.99999952, 12.99999952,
        2.00000003,  6.00000024, 11.00000024,  8.00000012,  5.        ,
        5.        ,  5.        ,  5.        ,  3.00000012,  6.99999988,
        8.99999976, 12.00000048,  6.99999988,  2.00000003,  1.00000001,
        0.        ,  2.00000003,  8.00000012, 10.        , 15.        ,
        8.99999976,  8.99999976,  4.00000006,  1.00000001,  0.        ,
        0.        ,  1.00000001,  3.00000012,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  6.99999988,
```

```
 8.00000012,  6.00000024,  6.00000024,  4.00000006,  4.00000006,
 3.00000012,  4.00000006,  6.00000024,  8.00000012, 15.        ,
 8.00000012,  8.00000012,  8.00000012,  6.99999988,  4.00000006,
 5.        ,  4.00000006, 13.99999976, 11.00000024, 10.        ,
 8.99999976,  8.99999976,  1.00000001, 11.00000024, 13.99999976,
17.99999952,  8.99999976,  8.99999976,  6.99999988,  1.00000001,
 0.        ,  8.00000012,  8.99999976,  8.99999976,  2.00000003,
 2.00000003,  2.00000003,  2.00000003,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  2.00000003,
 3.00000012,  6.00000024, 11.00000024,  8.00000012,  8.00000012,
 4.00000006,  2.00000003,  2.00000003,  6.00000024,  6.99999988,
 8.99999976,  4.00000006,  2.00000003,  1.00000001,  0.        ,
 1.00000001,  1.00000001,  1.00000001,  1.00000001,  1.00000001,
 1.00000001,  1.00000001,  1.00000001,  1.00000001,  8.99999976,
10.        , 15.        ,  8.00000012,  8.00000012,  8.00000012,
 1.00000001,  1.00000001,  1.00000001,  4.00000006,  6.00000024,
 1.00000001,  1.00000001,  1.00000001,  1.00000001,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 1.00000001,  6.99999988,  8.00000012,  6.99999988,  3.00000012,
 3.00000012,  2.00000003,  1.00000001,  3.00000012,  2.00000003,
 8.99999976,  6.00000024,  2.00000003,  0.        ,  2.00000003,
 2.00000003,  2.00000003,  4.00000006, 12.00000048,  2.00000003,
 2.00000003,  2.00000003,  2.00000003,  2.00000003,  2.00000003,
 3.00000012,  8.00000012,  4.00000006,  2.00000003,  2.00000003,
 0.        ,  0.        ,  1.00000001,  5.        ,  6.00000024,
 5.        ,  2.00000003,  2.00000003,  2.00000003,  2.00000003,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  2.00000003,  2.00000003,  0.        ,  3.00000012,
 1.00000001,  1.00000001,  1.00000001,  1.00000001,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  3.00000012,  5.        ,  6.00000024,
 5.        ,  4.00000006,  4.00000006,  2.00000003,  2.00000003,
 1.00000001,  2.00000003,  3.00000012,  4.00000006,  2.00000003,
 2.00000003,  1.00000001,  1.00000001,  3.00000012,  2.00000003,
 1.00000001,  3.00000012,  2.00000003,  2.00000003,  2.00000003,
 2.00000003,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  2.00000003,  3.00000012,  1.00000001,
 2.00000003,  2.00000003,  2.00000003,  2.00000003,  2.00000003,
 2.00000003,  4.00000006,  6.99999988,  5.        ,  4.00000006,
 4.00000006,  4.00000006,  4.00000006,  4.00000006,  4.00000006,
 6.00000024,  3.00000012,  1.00000001,  2.00000003,  1.00000001,
 1.00000001,  3.00000012,  5.        ,  4.00000006,  3.00000012,
 2.00000003,  2.00000003,  2.00000003,  2.00000003,  2.00000003,
 3.00000012,  0.        ,  3.00000012,  2.00000003,  2.00000003,
 2.00000003,  2.00000003,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  2.00000003,  6.99999988,
 8.00000012,  6.99999988,  6.99999988,  2.00000003,  2.00000003,
 1.00000001,  1.00000001,  4.00000006,  2.00000003,  4.00000006,
 1.00000001,  1.00000001,  1.00000001,  1.00000001,  1.00000001,
 4.00000006, 12.00000048,  6.99999988,  5.        ,  4.00000006,
 4.00000006,  4.00000006,  6.00000024,  6.99999988, 16.00000024,
 5.        ,  2.00000003,  2.00000003,  2.00000003,  0.        ,
 5.        ,  2.00000003,  6.00000024,  6.00000024,  1.00000001,
 1.00000001,  1.00000001,  1.00000001,  0.        ,  0.        ,
```

```
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  1.00000001,
 4.00000006,  8.00000012,  6.00000024,  5.        ,  5.        ,
 4.00000006,  4.00000006,  4.00000006,  6.99999988, 12.00000048,
 6.99999988,  2.00000003,  2.00000003,  2.00000003,  0.        ,
 6.00000024,  5.        , 16.00000024, 12.99999952,  8.00000012,
 4.00000006,  3.00000012,  2.00000003,  0.        ,  4.00000006,
12.00000048,  6.00000024,  6.99999988,  6.00000024,  0.        ,
 0.        ,  2.00000003,  1.00000001,  2.00000003,  2.00000003,
 1.00000001,  1.00000001,  1.00000001,  1.00000001,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 1.00000001,  2.00000003,  6.99999988,  3.00000012,  5.        ,
 4.00000006,  2.00000003,  1.00000001,  1.00000001,  1.00000001,
 1.00000001,  1.00000001,  1.00000001,  1.00000001,  1.00000001,
 1.00000001,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  1.00000001,
 6.99999988,  6.00000024,  6.99999988,  2.00000003,  2.00000003,
 2.00000003,  2.00000003,  2.00000003,  2.00000003,  2.00000003,
 2.00000003,  2.00000003,  2.00000003,  2.00000003,  2.00000003,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  1.00000001,  5.        ,  8.99999976,  8.99999976,
 6.99999988,  5.        ,  4.00000006,  5.        ,  5.        ,
10.        ,  8.00000012, 10.        ,  2.00000003,  2.00000003,
 1.00000001,  1.00000001,  3.00000012,  8.00000012, 16.00000024,
12.00000048,  6.99999988,  3.00000012,  3.00000012,  3.00000012,
 1.00000001,  6.99999988, 12.99999952,  6.99999988,  2.00000003,
 2.00000003,  2.00000003,  2.00000003,  2.00000003,  5.        ,
 2.00000003,  4.00000006,  1.00000001,  1.00000001,  1.00000001,
 1.00000001,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  1.00000001,  6.99999988, 10.        ,
 8.00000012,  4.00000006,  2.00000003,  2.00000003,  2.00000003,
 2.00000003,  6.00000024,  8.99999976, 12.00000048,  6.99999988,
 4.00000006,  4.00000006,  2.00000003,  4.00000006, 12.99999952,
 6.99999988,  6.99999988,  6.99999988,  6.99999988,  6.99999988,
 6.99999988,  6.99999988,  6.99999988,  6.99999988, 10.        ,
 8.00000012,  6.99999988,  5.        ,  2.00000003,  1.00000001,
 1.00000001,  3.00000012,  6.99999988,  1.00000001,  1.00000001,
 1.00000001,  1.00000001,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  1.00000001,  1.00000001,
 3.00000012,  5.        ,  4.00000006,  4.00000006,  4.00000006,
 5.        ,  5.        ,  8.00000012,  8.99999976, 15.        ,
16.00000024, 10.        , 10.        , 10.        , 10.        ,
15.        , 17.99999952, 20.        , 12.00000048,  6.99999988,
 6.99999988,  4.00000006,  4.00000006, 16.00000024, 17.99999952,
20.        , 17.00000048, 10.        , 10.        ,  3.00000012,
 3.00000012,  4.00000006,  8.00000012, 12.99999952,  3.00000012,
 3.00000012,  3.00000012,  3.00000012,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
 0.        ,  0.        ,  0.        ,  0.        ,  3.00000012,
 5.        ,  6.99999988,  5.        ,  4.00000006,  2.00000003,
 2.00000003,  4.00000006,  4.00000006,  4.00000006, 10.        ,
13.99999976, 12.00000048, 11.00000024, 11.00000024,  6.99999988,
 6.99999988,  6.99999988, 18.99999976, 18.99999976,  8.99999976,
 5.        ,  5.        ])
```

Plotting the graphic for comparison in detail:

In [53]:
```
plt.figure(figsize=(20, 6))
plt.plot(new_test_labels, label='Actual Value')
plt.plot(test_predictions, label='Predicted Value')
plt.xlabel('Timestemp')
plt.ylabel('Value')
plt.legend()
plt.show()
```




To see all the data `train_labels`, `test_labels` and the `predictions` we have to inverse sclae the `training_labels` as well.

In [54]:
```
train_dummies = np.zeros((train_features.shape[0], goback+1))
train_dummies[:, 0] = train_labels.flatten()
train_dummies = scaler.inverse_transform(train_dummies)

new_train_labels = dc(train_dummies[:, 0])
new_train_labels
```

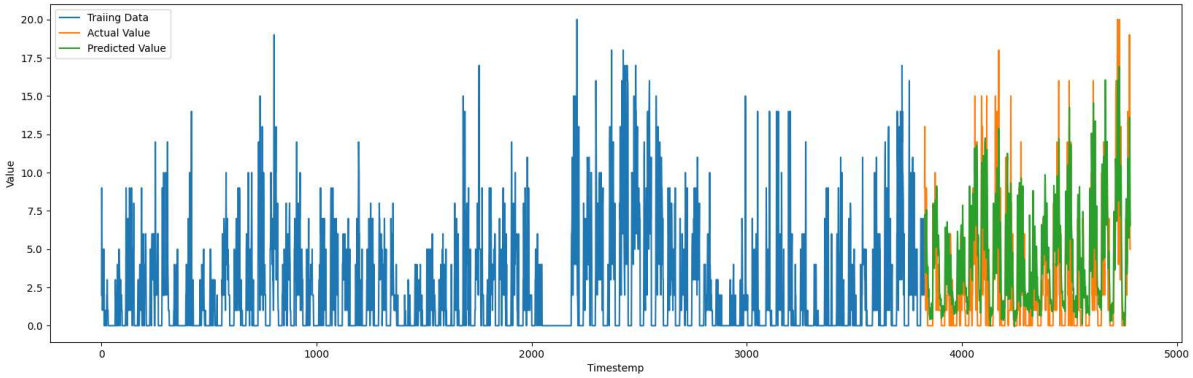
Out[54]:
```
array([2.00000003, 8.99999976, 5.        , ..., 3.00000012, 3.00000012,
       6.99999988])
```

To show off the data in one plot, we have to create an new array with `None` values in the beginning of the array, so the values are in the right order.

In [55]:
```
a = [None for i in new_train_labels]
new_test_labels = np.append(a, new_test_labels)
test_predictions = np.append(a, test_predictions)
```

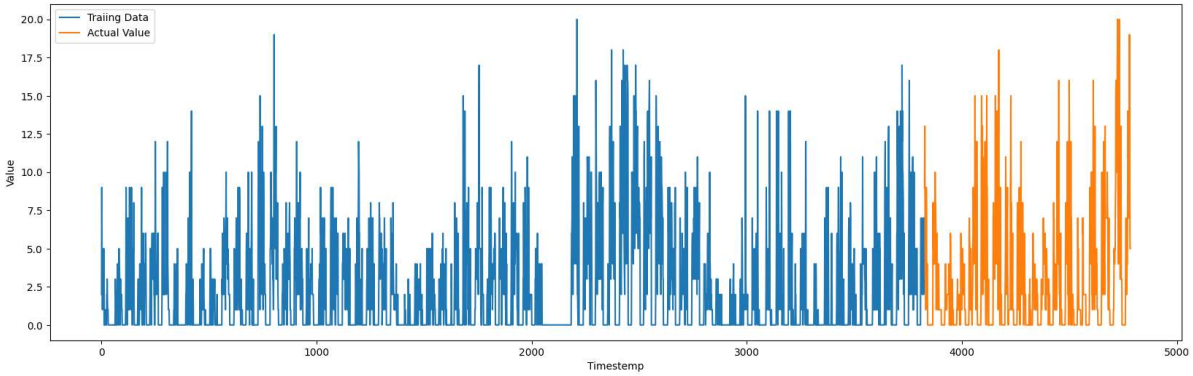
All the value in on plot:

In [56]:
```
plt.figure(figsize=(20, 6))
plt.plot(new_train_labels, label='Traiing Data')
plt.plot(new_test_labels, label='Actual Value')
plt.plot(test_predictions, label='Predicted Value')
plt.xlabel('Timestemp')
plt.ylabel('Value')
plt.legend()
plt.show()
```



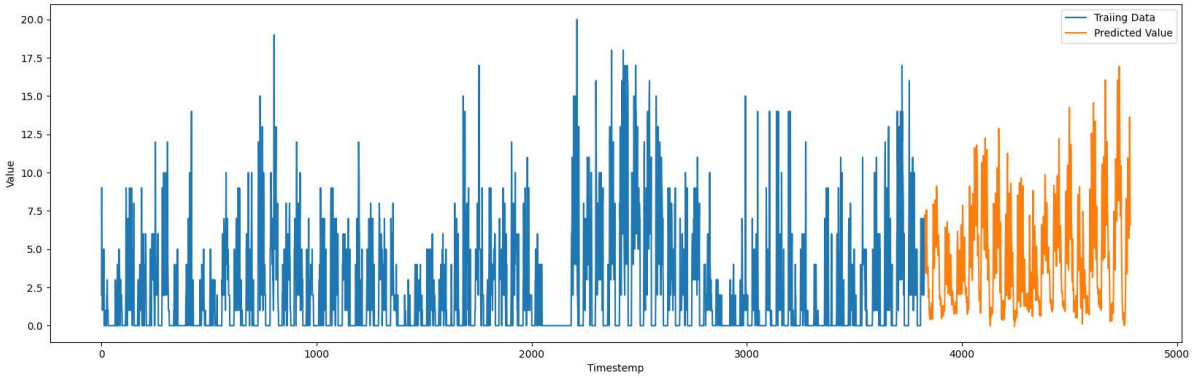

The values of `train_labels` and `test_labels` in one plot (out original data):

In [57]:
```
plt.figure(figsize=(20, 6))
plt.plot(new_train_labels, label='Traiing Data')
plt.plot(new_test_labels, label='Actual Value')
plt.xlabel('Timestemp')
plt.ylabel('Value')
plt.legend()
plt.show()
```




Te values of `train_labels` and `predictions` in one plot:

In [58]:
```
plt.figure(figsize=(20, 6))
plt.plot(new_train_labels, label='Traiing Data')
plt.plot(test_predictions, label='Predicted Value')
plt.xlabel('Timestemp')
plt.ylabel('Value')
plt.legend()
plt.show()
```




# 8. Suggestions for improvement:

After training, validation and evaluation, we came up with further ideas and suggestions for improvements that could improve the model's prediction values.

- **New feature: Opening hours: open = 1; closed = 0**

  In the visitor statistics (our data), the time periods are divided into eight sections: `09:30`, `10:45`, `13:45`, `16:15`, `17:45`, `18:00`, `18:30` and `18:45`. However, the opening hours of the library are not the same every day (Mon. - Thurs. 09:00-19:00; Fri. 09:00 - 17:00). This results in predictions > 0 on Friday, even though the library is closed. By adding the new feature, the model could learn that there can be no visitors at this time on Fridays. This would perhaps also improve the overall metric somewhat.

- **Conduct a survey in the library to find out the reasons for coming/not coming (empirical)**

  The second idea is to conduct a survey in the library to find out when and why visitors come to the library. Based on the answers, new features could be added and current features could be removed. However, the idea would go in the direction of an "empirical study" and would be an idea for a bachelor thesis, but would go beyond the scope of this project.

# 9. Learnings

- **Dealing with PyTorch**:

  The lecture and project deepened our understanding of using PyTorch as a library for programming DeepLearning networks.

- **Building an LSTM**:

  Through the hands-on project, we got a deep understanding of how an LSTM works and understood the architecture.

- **Data science process (ETL)**:

  Due to the fact that the data is not finished data, but self-collected data, we were able to experience the whole ETL process. Etract, Transform, Load. We extracted the data from the analogue statistics, processed this data and finally saved it to create an LSTM model.

- **Preparing data for a model**:

  Through the preprocessing and preparation steps, we realised how to prepare data for an AI model, or for an LSTM model.

- **Understanding AI sources**:

  By using an LSTM model, without prior knowledge in this area, we had to deal with a lot of literature and tutorials in the field of DeepLeanring and LSTM and got a good understanding of hot to evaluate these sources and extract the content for our own ideas/applications.

- **Practical experience**:

  Through this practical project we have gained practical experience in the field of DeepLearning, which was previously only based on theoretical knowledge.

- **Finding an idea for a model/implementing an idea**:

  Through the practical project, we are now able to implement ideas using a DeepLeanring, or ML model.

# Quellen

**Recherche Quelen:**


- https://griesshaber.pages.mi.hdm-stuttgart.de/nlp/07neuralnetworks/02RecurrentNeuralNetworks.html
- https://machinelearningmastery.com/lstm-for-time-series-prediction-in-pytorch/
- https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://databasecamp.de/en/ml/lstms