Exercise 1

Task Description

The goal of this exercise is to model the relationship between weather observations and the prevalence of new weekly influenza infections.

To investigate a potential relationship, we will use two datasets:

- tri-daily weather reports from 2009 - 2021 of an unnamed city in Europe.
- weekly reports on new influenza infections in the same city for the same time.

In this exercise, you will

- use `pandas` to read, prepare and transform data,
- use `matplotlib` to visually analyse data,

The data to be used can be found in: `~/shared/data/`.

To complete this exercise, you will have to:

- prepare the data, which (at minimum) involves the following:
  - load and prepare the data
  - handling missing values
  - handling outliers
  - temporal alignment of the two datasets
- analyse the data:
  - compare descriptive statistics
  - visually investigate the raw data to gain an understanding of the data identify patterns, outliers etc.,
  - look at the relationship between the variables of interest
- model the relationship:
  - fit a model that predicts new infections from weather observation data

This notebook, totals 55 points (not in relation with the 65 points for the exam, but previously assigned like this for DOPP):

- Task 1: 20 points
- Task 2: 15 points
- Task 3: 10 points
- Task 4: 5 points
- Task 5: 5 points

```
# DO NOT MODIFY OR COPY THIS CELL!!
# Note: The only imports allowed are Python's standard library,
pandas, numpy, scipy, matplotlib, seaborn and scikit-learn
import numpy as np
```

```
import pandas as pd
import glob
import os
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import pickle
import typing
```

Task 1: Load Data (20 Points)

1.1: Weather observations

As a first step, implement the method `load_weather_data()`, which should read all individual (yearly) datasets from the csv files in `data\weather\` into a single `pd.DataFrame` and return it.

- make sure that you load all the data (2009-2021, 13 years)
- split the tri-daily and daily data (tri-daily data has _7h, _14h, and _19h suffixes for column headers), and convert the tri-daily data from a wide to a long format (use pandas' wide_to_long or melt functions). Introduce a new `hours` column which's values should be taken from the column suffixes.
- make sure all columns are appropriately typed (numeric values -> float, countables, i.e. days -> int, etc.)! Especially the `date` column! See datetime and `pandas.Timestamp`!
- from the `date` column create `year`, `month`, `week`, `day` columns, where `week` contains the week number of the date. Use Pandas built-in datetime handling features.
- from the wide to long transform, you should have an `hour` column with the 7, 14, or 19 hours values.
- create a `MultiIndex` from the date columns with the following hierarchy: `year` - `month` - `week` - `day` - `hour` (make sure to label them accordingly)

Hints:

- LOOK at the data in the original files
- It is advisable not to append each data set individually, but to read each data frame, store it into a list and combine them once at the end.
- Note that for the `precip` data column you will get an unexpected (object) datatype. For this task it is ok to leave it like that, it is due to special values, see next chapters!
- Your resulting data frame should look as follows:

Weather data frame example

```
# DO NOT MODIFY OR COPY THIS CELL!!
data_path = os.path.join(os.environ["HOME"], "shared", "194.192-
2025W", "data")
weather_data_path = os.path.join(data_path, 'weather')
influenza_data_path = os.path.join(data_path, 'influenza')
```

```
def load_weather_data(weather_data_path:str) ->
typing.Tuple[pd.DataFrame, pd.DataFrame]:
    """
    Load all weather data files and combine them into a single Pandas
DataFrame.
    Split the tri-daily data from the daily data.
    For the tri-daily data create a new hour column using the
indicated hour in the column names.
    Add a week column and a hierarchical index (year, month, week,
day, hour).
    For the daily-only data also add a hierarchical index (year,
month, week, day).

    Parameters
    --------
    weather_data_path: path to directory containing weather data CSV
files

    Returns
    --------
    weather_data: data frame containing the tri-daily (hours) weather
data
    weather_data_daily: data frame containing the daily weather data
(e.g. precip, precipType, etc.)
    """

    # YOUR CODE HERE
    df = pd.read_csv(weather_data_path+"/weather_daily_2009.csv",
sep=";")
    datalist = [
        "/weather_daily_2010.csv",
        "/weather_daily_2011.csv",
        "/weather_daily_2012.csv",
        "/weather_daily_2013.csv",
        "/weather_daily_2014.csv",
        "/weather_daily_2015.csv",
        "/weather_daily_2016.csv",
        "/weather_daily_2017.csv",
        "/weather_daily_2018.csv",
        "/weather_daily_2019.csv",
        "/weather_daily_2020.csv",
        "/weather_daily_2021.csv",
    ]
    for data in datalist:
        df2 = pd.read_csv(weather_data_path + data, sep=";")
        df = pd.concat([df,df2],ignore_index=True)
    df = df.reset_index()

    df["date"] = pd.to_datetime(df["date"],format="%d.%m.%Y")
```

```
    df["year"] = df["date"].dt.year.astype(int)
    df["month"] = df["date"].dt.month.astype(int)
    df["week"] = df["date"].dt.isocalendar().week.astype(int)
    df["day"] = df["date"].dt.day.astype(int)


    stubs=
["airPressure","skyCover","temp","hum","windDir","windBeauf"]
    df = pd.wide_to_long(df, stubnames=stubs, i="index",
j='hour',sep='_', suffix=r'\d+h')
    df = df.reset_index()
    df["hour"] = df["hour"].str.removesuffix("h").astype(int)
    df["airPressure"] = df["airPressure"].astype(float)
    df["skyCover"] = df["skyCover"].astype(int)
    df["temp"] = df["temp"].astype(float)
    df["hum"] = df["hum"].astype(int)
    df["windDir"] = df["windDir"].astype(str)
    df["windBeauf"] = df["windBeauf"].astype(int)
    df["precipType"] = df["precipType"].astype(str)


    df = df.set_index(["year","month","week","day","hour"])
    df = df.drop(labels = "index",axis=1)

    weather_data = df
    df = df.reset_index()
    df = df.set_index(["year","month","week","day"])
    df = df.drop(labels = "hour",axis=1)
    weather_data_daily = df.drop(labels = stubs,axis=1)




    return weather_data, weather_data_daily

# DO NOT MODIFY OR COPY THIS CELL!!
weather_data, daily_weather_data =
load_weather_data(weather_data_path)
# print first couple of rows:
print('hourly weather data:')
display(weather_data.head())
print('\ndaily weather data:')
display(daily_weather_data.head())
```

```
hourly weather data:

                                date  precip precipType  airPressure
skyCover  \
year month week day hour
```

```
2009 1     1    1   7    2009-01-01       0         nan         999.7
10
                2   7    2009-01-02  traces       snow         999.6
10
                3   7    2009-01-03       0         nan        1002.7
3
                4   7    2009-01-04  traces       snow         993.4
5
           2    5   7    2009-01-05  traces       snow         988.4
9
```

```
                           temp  hum windDir  windBeauf
year month week day hour
2009 1     1    1   7     -4.5   79       W          2
                2   7     -3.0   67     nan          0
                3   7     -4.8   83       N          2
                4   7     -3.8   75       W          4
           2    5   7     -0.7   64       W          4
```

```
daily weather data:
```

```
                           date  precip precipType
year month week day
2009 1     1    1   2009-01-01       0        nan
                2   2009-01-02  traces       snow
                3   2009-01-03       0        nan
                4   2009-01-04  traces       snow
           2    5   2009-01-05  traces       snow
```

```
# use this cell to inspect the data.

print(f"Data dimensions are: {weather_data.shape[0]} rows and
{weather_data.shape[1]} columns")
print(f"\nindex types are: \n-----------------\
n{weather_data.index.dtypes}")
print(f"\ncolumn types are: \n-----------------\
n{weather_data.dtypes}")

print(f"\nFor daily data: \nData dimensions are:
{daily_weather_data.shape[0]} rows and {daily_weather_data.shape[1]}
columns")
print(f"\nindex types are: \n-----------------\
n{daily_weather_data.index.dtypes}")
print(f"\ncolumn types are: \n-----------------\
n{daily_weather_data.dtypes}")
```

```
Data dimensions are: 14244 rows and 9 columns

index types are:
-----------------
```

```
year     int64
month    int64
week     int64
day      int64
hour     int64
dtype: object

column types are:
-----------------
date           datetime64[ns]
precip                 object
precipType             object
airPressure           float64
skyCover                int64
temp                  float64
hum                     int64
windDir                object
windBeauf               int64
dtype: object

For daily data:
Data dimensions are: 14244 rows and 3 columns

index types are:
-----------------
year     int64
month    int64
week     int64
day      int64
dtype: object

column types are:
-----------------
date          datetime64[ns]
precip                 object
precipType             object
dtype: object
```

Tests

Optional but recommended! Check if the loading of the data was successful using some assertions. The points will automatically assigned by the hidden test, try to make sure that you covered all required points from above!

```
# use this cell to create your own tests. best case: create a test for each requirement above!

# Note: you can add new cells also for the other tasks to add your own tests.
```

```
# But NEVER COPY an existing cell, since this can break the
autograding!

# DO NOT MODIFY OR COPY THIS CELL!!
# TESTS: dimensions should be like this:
assert weather_data.shape[0] == 14244 # 4748
assert weather_data.shape[1] >= 7 # 24

# hidden tests for grading DO NOT MODIFY OR COPY THIS CELL!!

# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!

# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!
```

Question

Which combination of year+month has the highest number of entries?

- Implement the function below to find the answer!
- Find the respective entry/entries using pandas!!

```
def get_year_month_highest_entries(data_frame:pd.DataFrame):
    year = 0
    month = 0

    # YOUR CODE HERE
    df = data_frame
    #df = df["temp"]
    value = df.dropna(subset=["temp"]).groupby(["year","month"])
["temp"].sum().reset_index()
    i = value["temp"].idxmax()
    year=value.loc[i,"year"]
    month=value.loc[i,"month"]

    return year, month

# DO NOT MODIFY OR COPY THIS CELL!!
high_num_year, high_num_month =
get_year_month_highest_entries(weather_data)
print(f"Month {high_num_month}, of year {high_num_year} has the
highest number of entries!")
```

```
Month 7, of year 2015 has the highest number of entries!
```

```
# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!
```

1.2: Influenza infections

Load and prepare the second dataset (`data/influenza/influenza.csv`), which contains the number of new influenza infections on a weekly basis.

- Stack all seasonal data files into one dataframe

- Extract a correct `year` and `week` column from the `Season Week` column, see `data/influenza/descriptions.txt` for details.
- For each entry, extract the year based on the season and month values
- Create a `MultiIndex` from the `year` and `week` columns
- Rename column containing influenza cases to `weekly_infections`
- Make sure that all columns have appropriate types
- Remove rows with missing infection counts
- Your resulting data frame should look as follows:

Example data frame

```
def load_influenza_data() -> pd.DataFrame:
    """
    Load the influenza data from the files into a pndas dataframe

    Returns
    --------
    influenza_data: data frame containing the influenza data
    """

    # YOUR CODE HERE
    df = pd.read_csv(influenza_data_path+"/influenza_09-
19.csv",sep=",")
    df2 = pd.read_csv(influenza_data_path+"/influenza_19-
21.csv",sep=",")
    df = pd.concat([df,df2],ignore_index=True)
    df = df.reset_index()

    df["week"] = df["Season Week"].str.split("_").str[1].astype(int)
    df["year"] = df["Season Week"].str.split("_").str[0]
    years = df["year"].str.split("/", expand=True).astype(int)
    start = years[0].astype(int)
    end = years[1].astype(int)
    df["year"] = np.where(df["week"] < 25, end, start)
    df["weekly infections"] = df["New Cases"]
    df = df.drop(labels=["index","Season Week", "Margin", "New
Cases"],axis=1)
    df["week"] = df["week"].astype(int)
    df = df.set_index(["year","week"])
    df = df.drop(index=(2010, 10))
    pd.set_option("display.max_rows", 10)

    influenza_data = df
    return influenza_data

# DO NOT MODIFY OR COPY THIS CELL!!
data_influenza = load_influenza_data()
display(data_influenza)
```

```
          weekly infections
year week
2009 40                6600
     41                7100
     42                7700
     43                8300
     44                8600
...                     ...
2021 5                 1400
     4                 1800
     3                 1800
     2                 1500
     1                 1300

[310 rows x 1 columns]
```

```
# use this cell to inspect the data.

print(f"Data dimensions are: {data_influenza.shape[0]} rows and
{data_influenza.shape[1]} columns")
print(f"\nindex types are: \n-----------------\
n{data_influenza.index.dtypes}")
print(f"\ncolumn types are: \n-----------------\
n{data_influenza.dtypes}")
```

```
Data dimensions are: 310 rows and 1 columns

index types are:
-----------------
year    int64
week    int64
dtype: object

column types are:
-----------------
weekly infections    object
dtype: object
```

Tests

Optional but recommended! Check if the loading of the data was successful using some assertions. The points will automatically be assigned by the hidden test, try to make sure that you covered all required points from above!

```
# use this cell to create your own tests

# tests DO NOT MODIFY OR COPY THIS CELL!!
# final dimensions should be like this:
assert data_influenza.shape == (310, 1)
```

```
# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!

# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!
```

Question

Which combination of year+week shows the highest number of infections?

- Implement the function below to return the result!
- You should find the respective entry/entries using pandas!
- Return the first answer if there is more than one combination fulfilling these criteria.

```
def get_year_week_most_infections(data_frame:pd.DataFrame
                                  ) -> typing.Tuple[int, int]:
    year = 0
    week = 0

    # YOUR CODE HERE
    df = data_frame
    df
    value = df.dropna(subset=["weekly
infections"]).groupby(["year","week"]).sum()
    i = value["weekly infections"].astype(float).idxmax()

    year = i[0]
    week = i[1]
    return year, week

# DO NOT MODIFY OR COPY THIS CELL!!
high_num_year, high_num_week =
get_year_week_most_infections(data_influenza)
print(f"Week {high_num_week}, of year {high_num_year} has the highest
number of infections!")
```

```
Week 1, of year 2017 has the highest number of infections!
```

```
# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!
```

Task 2: Handling Missing Values (15 Points)

If you take a closer look at the data, you will notice that a few of the observations are missing.

There is a wide range of standard strategies to deal with such missing values, including:

- row deletion
- substitution methods (e.g., replace with mean or median)
- hot-/cold-deck methods (impute from a randomly selected similar record)
- regression methods

To decide which strategy is appropriate, it is essential to investigate the mechanism that led to the missing values to find out whether the missing data is missing completely at random, missing at random, or missing not at random.

- **MCAR** (Missing Completely At Random) means that there is no relationship between the missingness of the data and any of the values.
- **MAR** (Missing At Random) means that there is a systematic relationship between the propensity of missing values and the observed data, but not the missing data.
- **MNAR** (Missing Not At Random) means that there is a systematic relationship between the propensity of a value to be missing and its values.

You talked to the meteorologists who compiled the data to find out more about what mechanisms may have caused the missing values:

1. They told you that they do not know why some of the temperature (`temp`) and pressure (`airPressure`) readings are missing. They suspect a problem with the IT infrastructure. In any case, the propensity of temperature and pressure values to be missing does not have anything to do with the weather itself.

2. For wind intensity values of 0, the wind direction is not provided (for obvious reasons).

Check the plausibility of these hypotheses in the data, consider the implications, and devise appropriate strategies to deal with the various missing values.

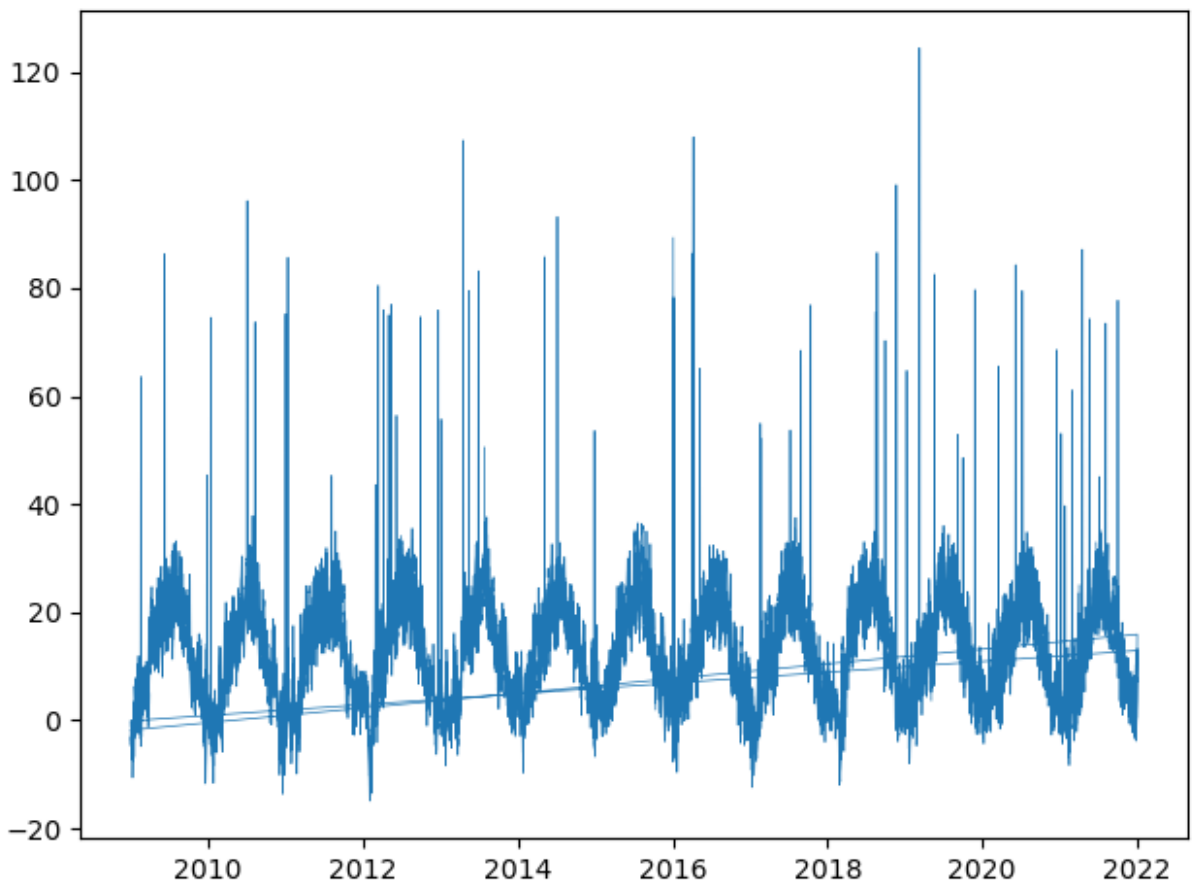
- Handle missing values for the following columns: `temp`, `airPressure`, `windDir`

To implement your strategy, you can use a range of standard mechanisms provided by Pandas.

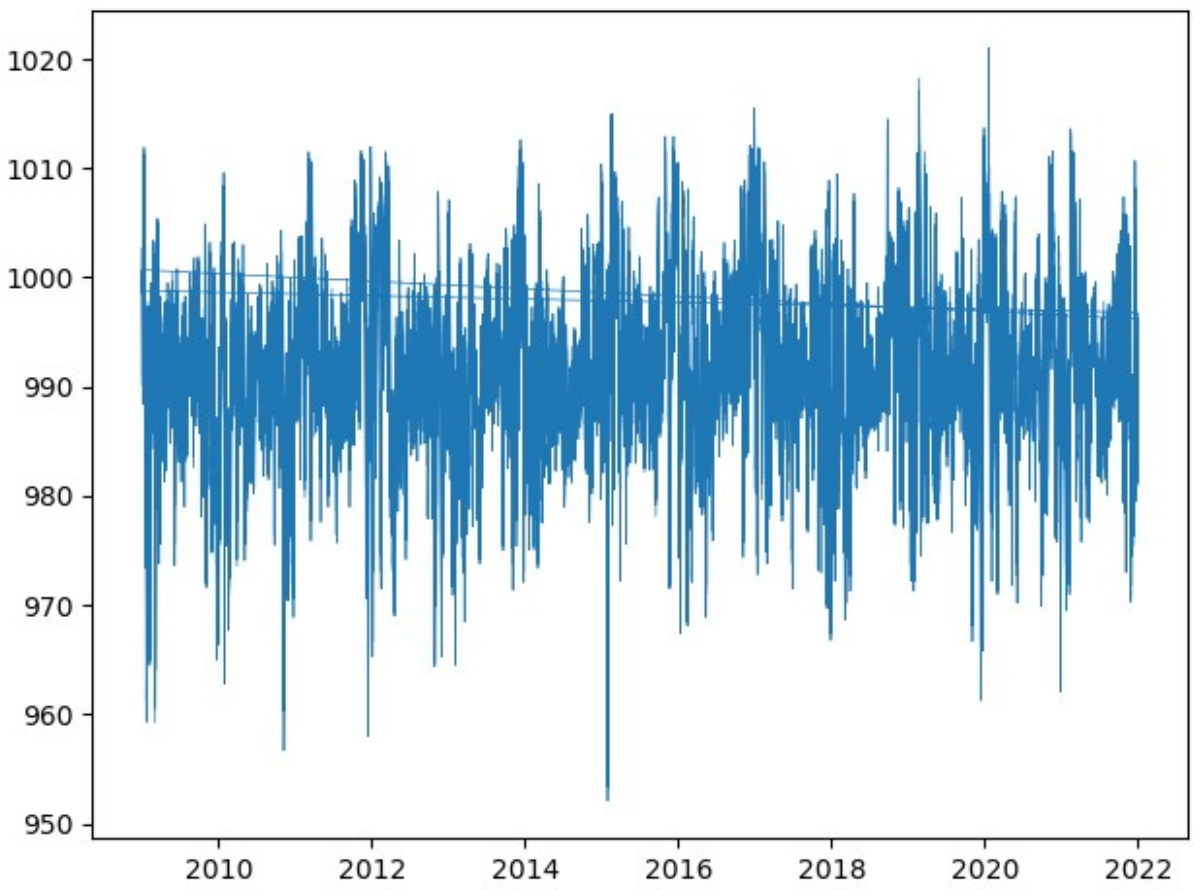
Visualize some data

Plot temperature (temp) and air pressure (airPressure) as a function of time for the weather data. Additionally plot the average weekly sky coverage over all years.

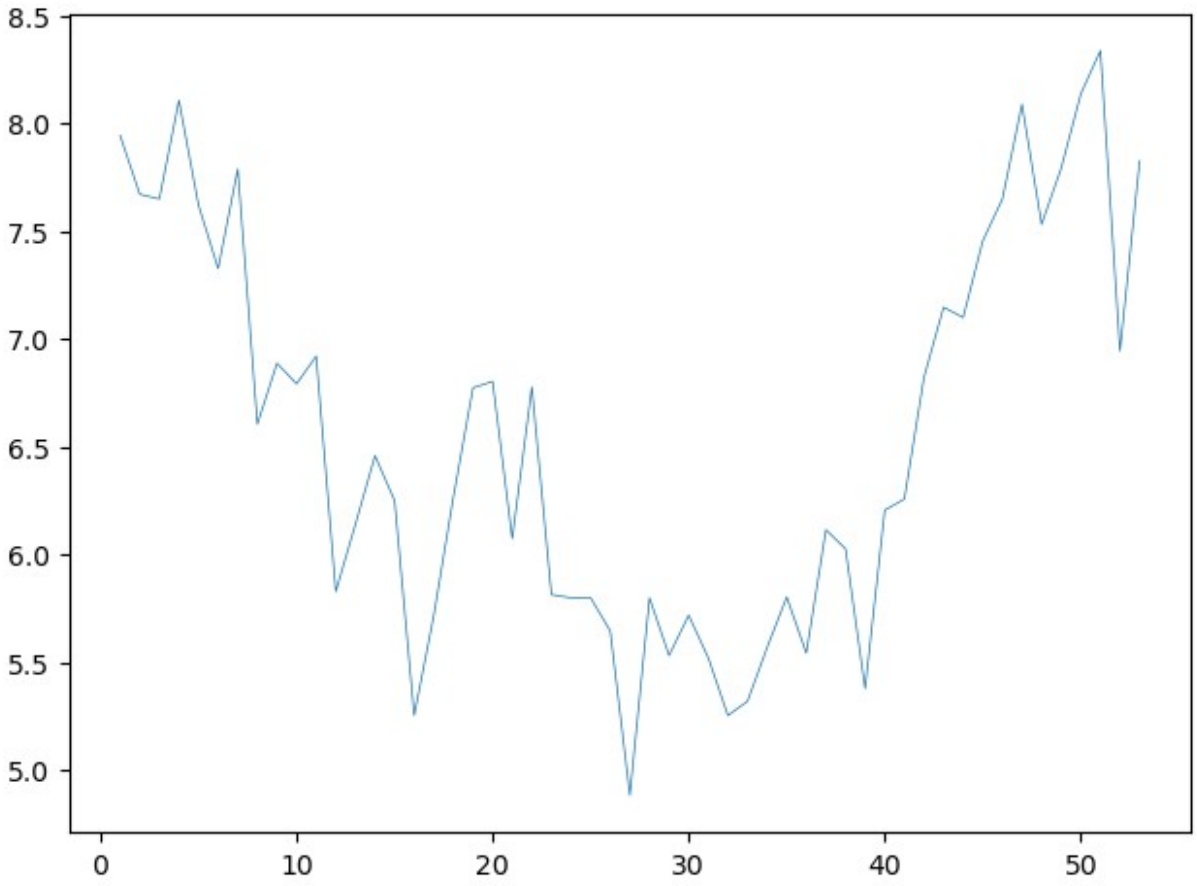
```
# plot the temperature

# YOUR CODE HERE
data,daily = load_weather_data(weather_data_path)
plt.plot(data["date"],data["temp"],linewidth=0.5)
plt.tight_layout()
```




```
# plot the temperature

# YOUR CODE HERE
data,daily = load_weather_data(weather_data_path)
plt.plot(data["date"],data["airPressure"],linewidth=0.5)
plt.tight_layout()
```




```
# plot the sky coverage

# YOUR CODE HERE
data,daily = load_weather_data(weather_data_path)
test = data.groupby(["week"])["skyCover"].mean().reset_index()
plt.plot(test["week"],test["skyCover"],linewidth=0.5)

plt.tight_layout()
```



2.1: Missing values for temperature and air pressure

Find and visualize missing values for `temp` and `airPressure` columns in the `weather_data` data frame.

Data inspection

First, let's visualize the missing data to get a better feel for what is happening.

- Implement the `get_data_around_missing` function below to extract and return a dataframe that only contains rows around missing (`isna`, `isnull`) values for the column indicated by `column`.
- First find missing values in the specified column
- For each missing value, create a dataframe that contain only rows with a date +- `delta_days` from the date of the missing value.
- Put the dataframes into `df_list` and return them.

```
def get_data_around_missing(df:pd.DataFrame, column:str,
                            delta_days:int=2) ->
typing.List[pd.DataFrame]:
    """
    Build a list of dataframes containing missing values indicated by
```

```
column.
    Each dataframe contains rows around a missing (isna)
    value in column, within a date of +- delta_days.

    Parameters
    --------
    df: dataframe containing the missing values
    column: the column to look for missing values
    delta_days: the number of days +-around the date of the missing
values to keep in the returned data frames

    Returns
    --------
    df_list: list of dataframes with some missing data
    """
    # TODO better description
    # check out datetime.timedelta!
    df_list = []

    # YOUR CODE HERE
    dfc = df.copy()
    dfc = dfc.reset_index()
    dfc = dfc.set_index(df["date"])
    dfc = dfc.sort_index()
    delta = pd.Timedelta(days=delta_days)
    for t in dfc.index[dfc[column].isna()]:
        window = dfc.loc[t - delta : t + delta]
        df_list.append(window)
    return df_list

# DO NOT MODIFY OR COPY THIS CELL!!
missing_temp_df_list = get_data_around_missing(weather_data, 'temp')
missing_airPressure_df_list = get_data_around_missing(weather_data,
'airPressure')

# tests, DO NOT MODIFY OR COPY THIS CELL!!
print(len(missing_temp_df_list))
print(len(missing_airPressure_df_list))

assert 150 < len(missing_temp_df_list) < 250, "There should be between
150 and 150 missing values in temp!"
assert 150 < len(missing_airPressure_df_list) < 250, "There should be
between 150 and 250 missing values in airPressure!"

assert all([isinstance(cur_el, pd.DataFrame) for cur_el in
missing_temp_df_list])
assert all([isinstance(cur_el, pd.DataFrame) for cur_el in
missing_airPressure_df_list])
```

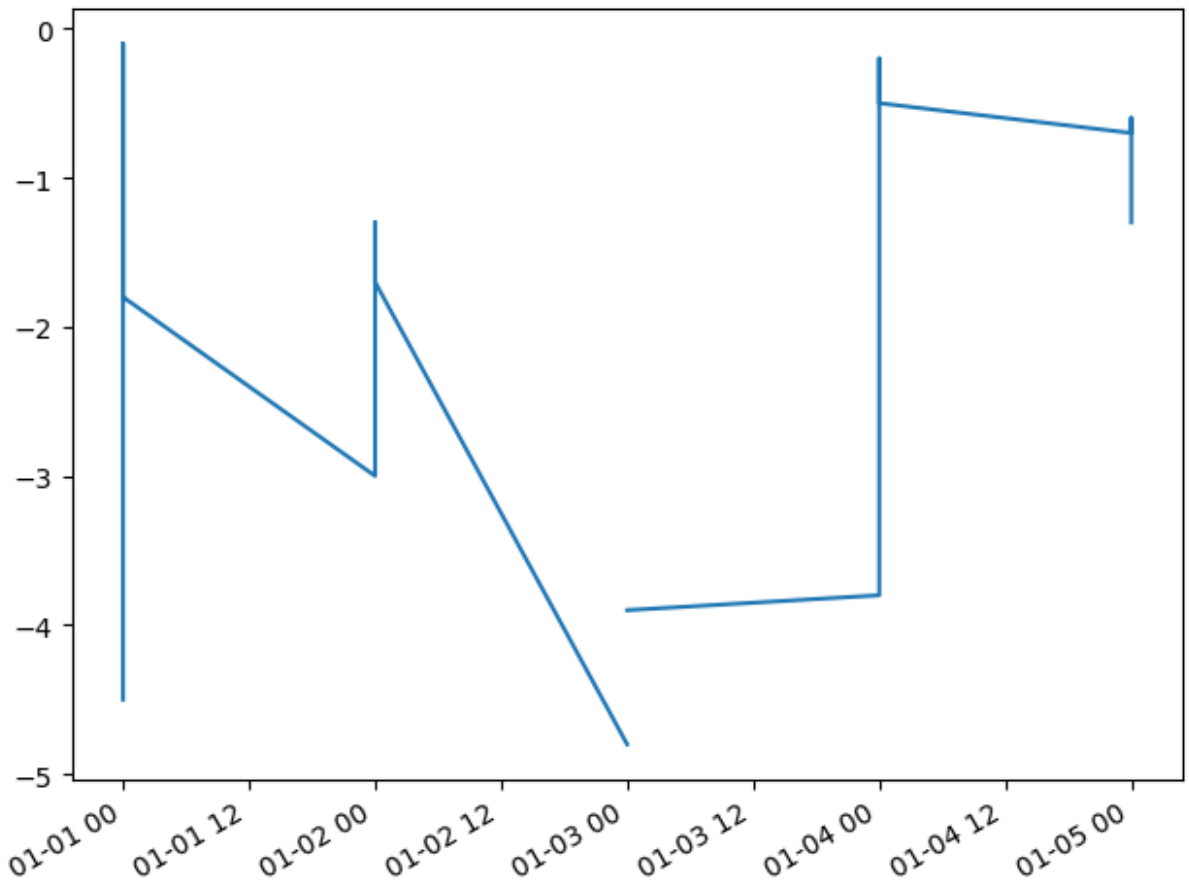
```
231
181

# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!
```

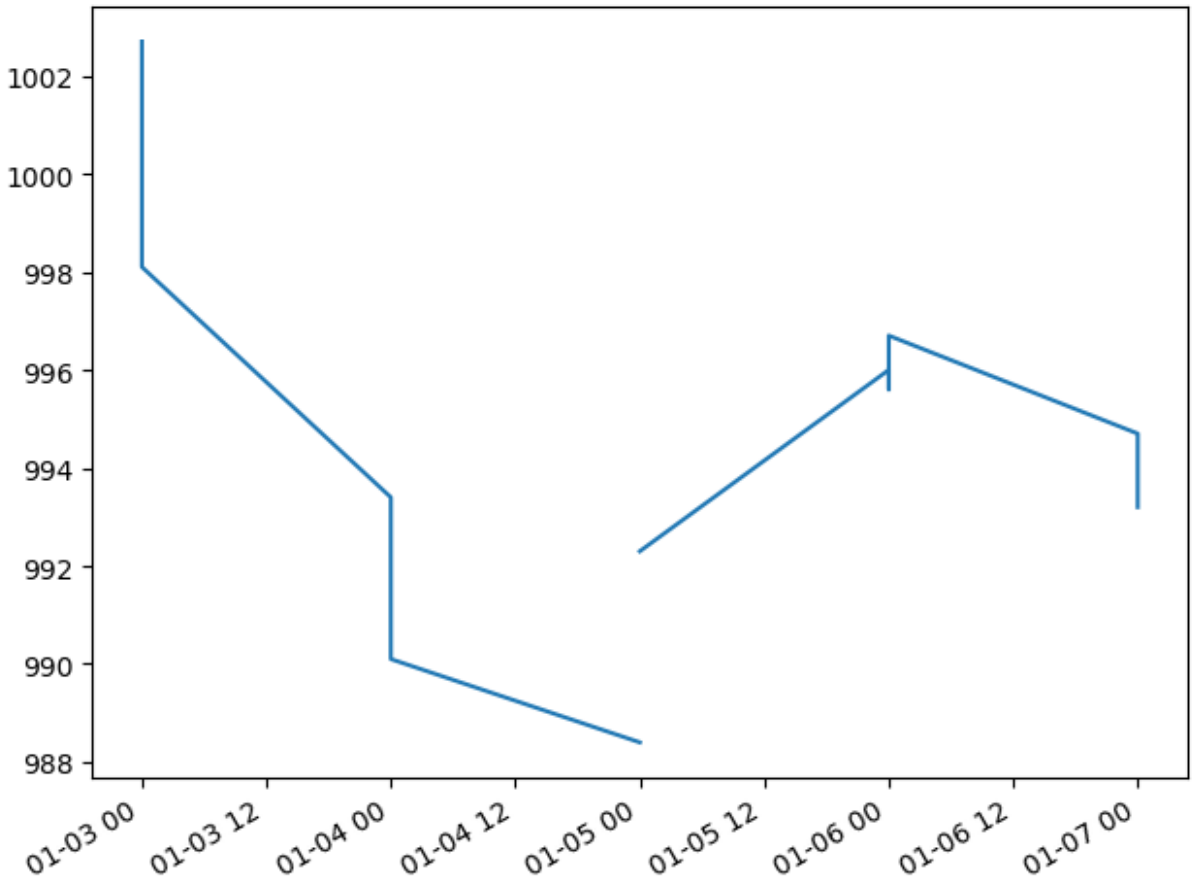
In order to be able to see the data, complete the function `plot_value_series` to plot a timeseries of a dataframe identified by `column`:

```
def plot_value_series(df:pd.DataFrame, column:str) -> None:
    """
    Plot the values in column in data frame df
    """
    # YOUR CODE HERE
    plt.plot(df["date"],df[column])
    plt.gcf().autofmt_xdate()
    plt.tight_layout()
    plt.show()
```

Now we use the function to plot a missing value for `temp` and `airPressure`:

```
# tests,  DO NOT MODIFY OR COPY THIS CELL!!
plot_value_series(missing_temp_df_list[0], 'temp')
plot_value_series(missing_airPressure_df_list[0], 'airPressure')
```


0
−1
−2
−3
−4
−5
01-01 00
01-01 12
01-02 00
01-02 12
01-03 00
01-03 12
01-04 00
01-04 12
01-05 00




```
# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!
```

Handle missing temperature values

Use the plots above and the information that was given to us by the meterologist to decide on a suitable strategy to fix the missing values.

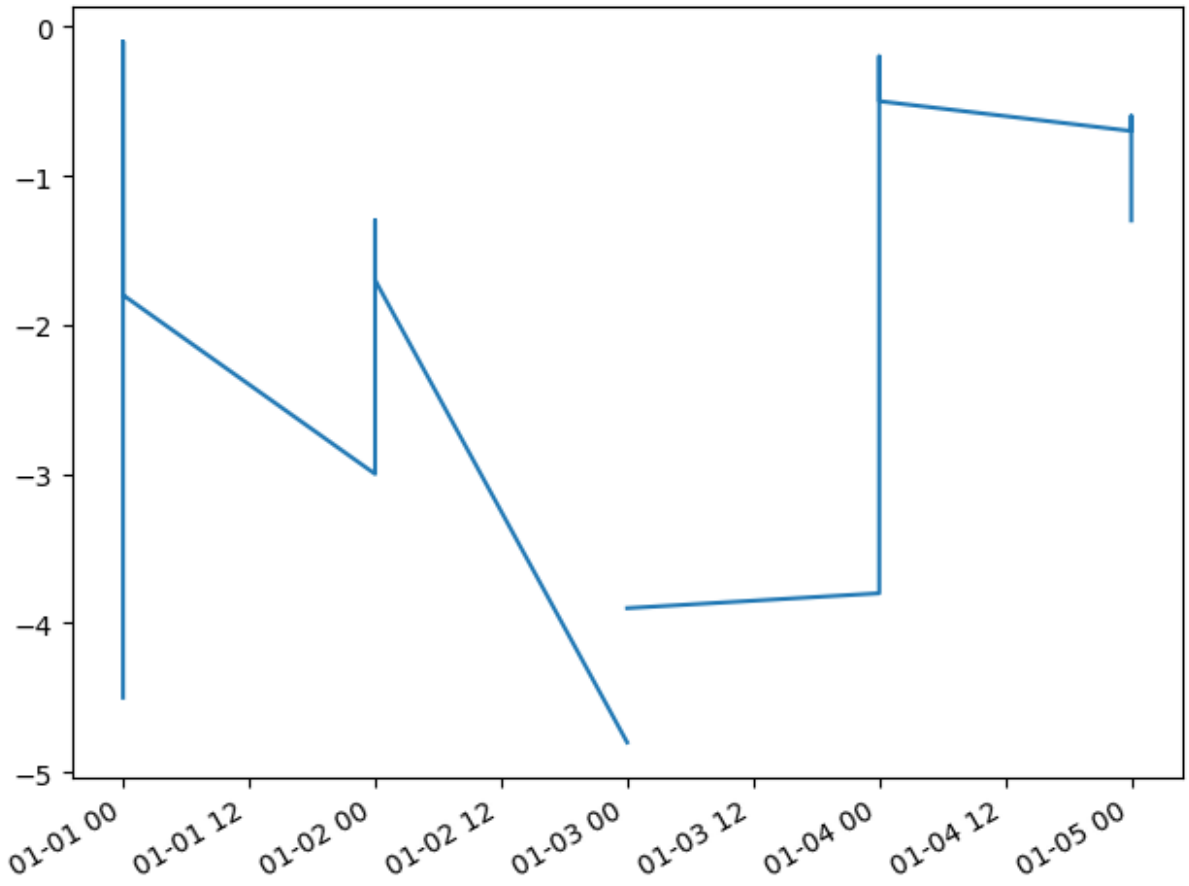
- Implement the function below to get rid of the missing values for temperature (`temp`).
- Choose an appropriate strategy to fill in the misssing values.

```
def handle_missing_temp_values(df:pd.DataFrame) -> pd.DataFrame:
    """
    Handle missing temperature values appropriately!

    Parameters
    --------
    df: dataframe containing the missing values

    Returns
    --------
    df_ret: dataframe with fixed values
    """
```

```
    df_ret = df.copy()

    # YOUR CODE HERE
    df_ret = df_ret.interpolate(method="linear",columns="temp")
    df_ret["temp"] = df_ret["temp"].ffill().bfill()

    return df_ret
```

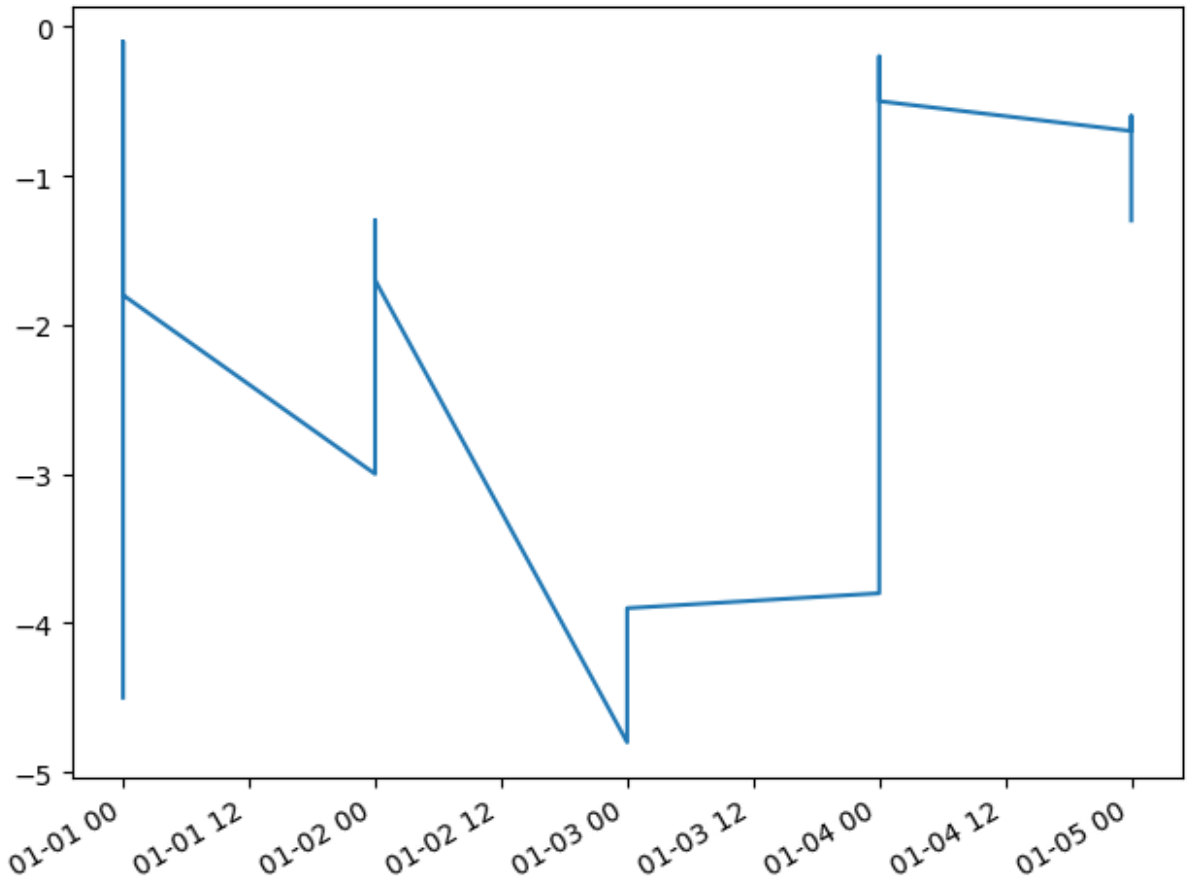
Plot the fixed stretch of temp values from above, and compare to the unmodified version:

```
missing_temp_idx = 0

# DO NOT MODIFY OR COPY THIS CELL!!
plot_value_series(missing_temp_df_list[missing_temp_idx], 'temp')
plot_value_series(handle_missing_temp_values(missing_temp_df_list[miss
ing_temp_idx]), 'temp')
```




```
/tmp/ipykernel_9276/1019226550.py:16: FutureWarning:
DataFrame.interpolate with object dtype is deprecated and will raise
in a future version. Call obj.infer_objects(copy=False) before
```

```
interpolating instead.
  df_ret = df_ret.interpolate(method="linear",columns="temp")
```



Handle missing air pressure values

Do the same for the air pressure values:

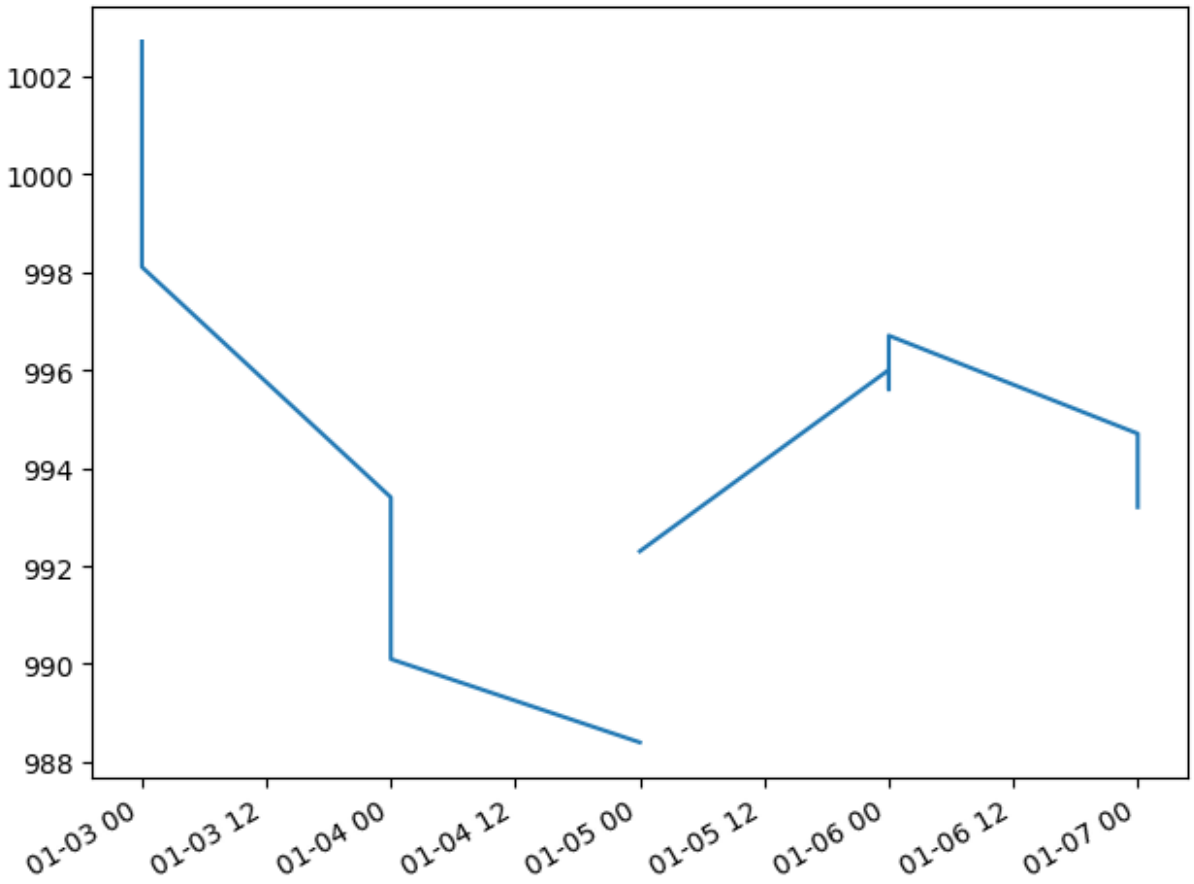
- Implement the function below to get rid of the missing values for air pressure (`airPressure`).
- Choose an appropriate strategy to fill in the misssing values.

```
def handle_missing_airPressure_values(df:pd.DataFrame) ->
pd.DataFrame:
    """
    Handle missing air pressure values appropriately!

    Parameters
    --------
    df: dataframe containing the missing values

    Returns
    --------
```

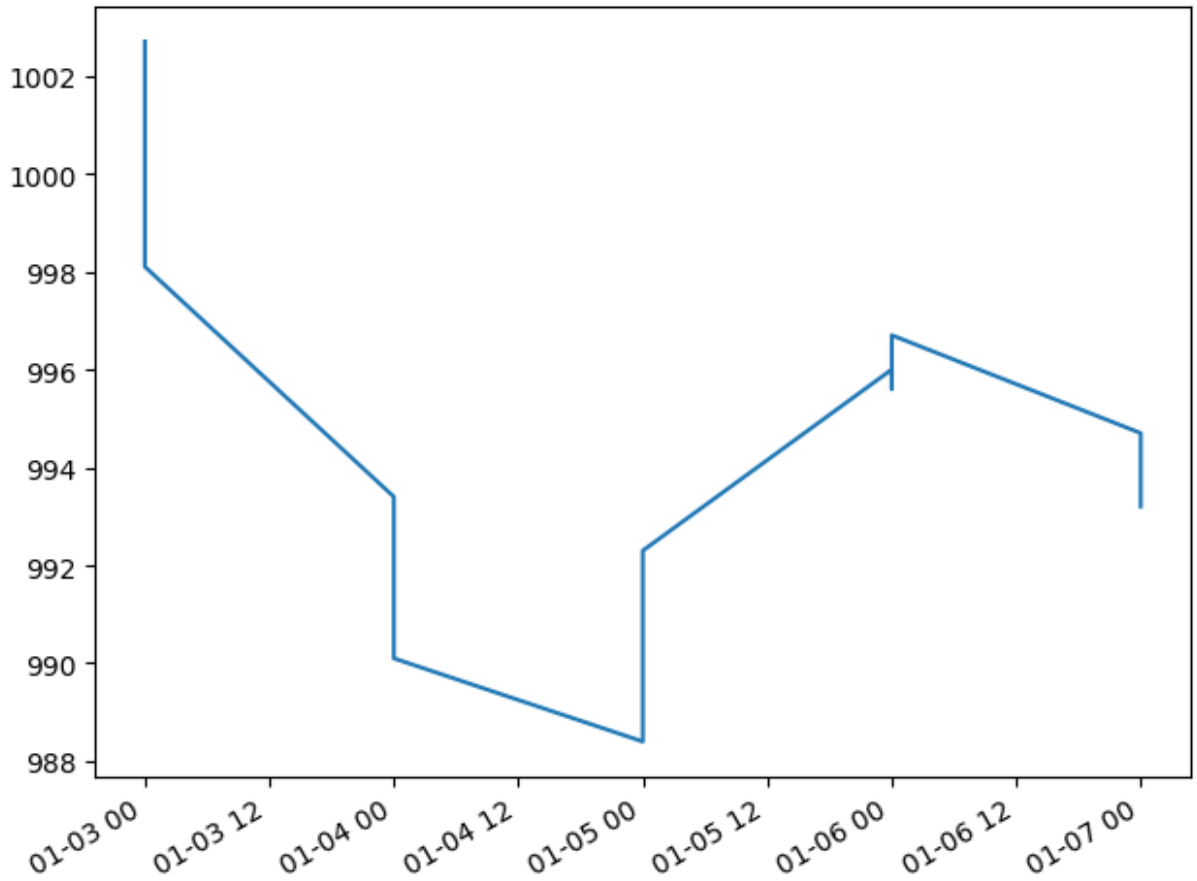
```
    df_ret: dataframe with fixed values
    """
    df_ret = df.copy()

    # YOUR CODE HERE
    df_ret = df_ret.interpolate(method="linear",columns="airPressure")
    df_ret["airPressure"] = df_ret["airPressure"].ffill().bfill()

    return df_ret
```

Plot the fixed stretch of airPressure values from above:

```
missing_apr_idx = 0

# DO NOT MODIFY OR COPY THIS CELL!!
plot_value_series(missing_airPressure_df_list[missing_apr_idx],
'airPressure')
plot_value_series(handle_missing_airPressure_values(missing_airPressur
e_df_list[missing_apr_idx]), 'airPressure')
```



```
/tmp/ipykernel_9276/2298240427.py:16: FutureWarning:
DataFrame.interpolate with object dtype is deprecated and will raise
in a future version. Call obj.infer_objects(copy=False) before
interpolating instead.
  df_ret = df_ret.interpolate(method="linear",columns="airPressure")
```




```
# tests, DO NOT MODIFY OR COPY THIS CELL!!
# the function to get the short dataframes around missing values
should now return an empty list if we fix all temp values!
empty_temp_list =
get_data_around_missing(handle_missing_temp_values(weather_data),
'temp')
empty_airPressure_list =
get_data_around_missing(handle_missing_airPressure_values(weather_data
), 'airPressure')
assert len(empty_temp_list) == 0
assert len(empty_airPressure_list) == 0

/tmp/ipykernel_9276/1019226550.py:16: FutureWarning:
DataFrame.interpolate with object dtype is deprecated and will raise
in a future version. Call obj.infer_objects(copy=False) before
interpolating instead.
```

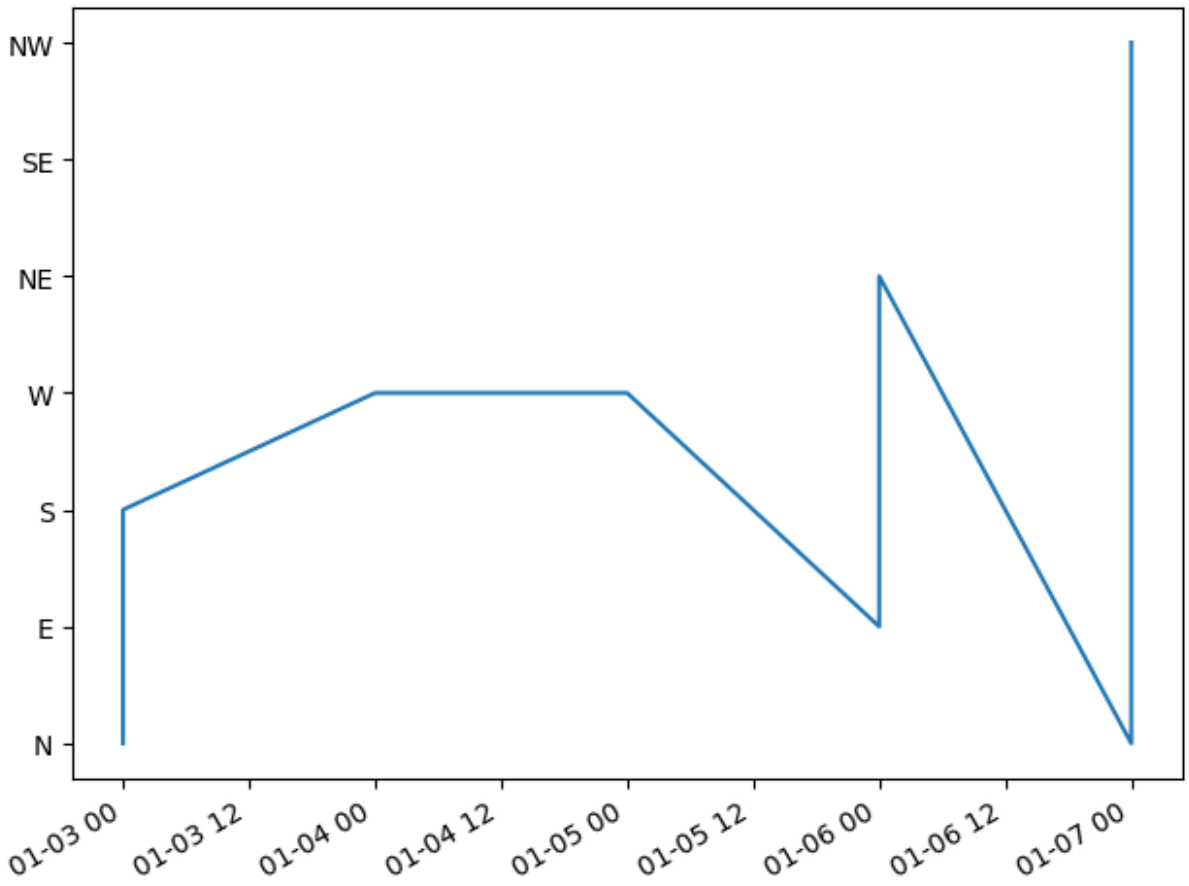
```
  df_ret = df_ret.interpolate(method="linear",columns="temp")
/tmp/ipykernel_9276/2298240427.py:16: FutureWarning:
DataFrame.interpolate with object dtype is deprecated and will raise
in a future version. Call obj.infer_objects(copy=False) before
interpolating instead.
  df_ret = df_ret.interpolate(method="linear",columns="airPressure")

# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!
```

2.2 Missing wind direction values

Data inspection

Check the assumption for missing wind direction values and handle the missing wind direction values in an appropriate way.

```
# write some code to check the assumtion for missing values of windDir

# YOUR CODE HERE
plot_value_series(missing_airPressure_df_list[missing_apr_idx],
'windDir')
```



Handle missing wind direction values

Implement a function that fixes the missing wind direction values.

```
def handle_missing_windDir_values(df:pd.DataFrame) -> pd.DataFrame:
    df_ret = df.copy()

    # YOUR CODE HERE
    df_ret["windDir"] = df_ret["windDir"].ffill().bfill()

    return df_ret

# tests, DO NOT MODIFY OR COPY THIS CELL!!
# Apply the windDir fix and check if any missing values remain
fix_wind_dir = handle_missing_windDir_values(weather_data)
assert fix_wind_dir[fix_wind_dir['windDir'].isnull()].empty

# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!
```

2.3 Combine all fixes to get clean data

```
def handle_missing_values_weather(data:pd.DataFrame) -> pd.DataFrame:
    """
    Parameters
    --------
    data: data frame containing missing values

    Returns
    --------
    data: data frame not containing any missing values
    """

    # YOUR CODE HERE
    data = handle_missing_temp_values(data)
    data = handle_missing_airPressure_values(data)

    return data

# DO NOT MODIFY OR COPY THIS CELL!!
weather_data_complete = handle_missing_values_weather(weather_data)
```

```
/tmp/ipykernel_9276/1019226550.py:16: FutureWarning:
DataFrame.interpolate with object dtype is deprecated and will raise
in a future version. Call obj.infer_objects(copy=False) before
interpolating instead.
  df_ret = df_ret.interpolate(method="linear",columns="temp")
/tmp/ipykernel_9276/2298240427.py:16: FutureWarning:
DataFrame.interpolate with object dtype is deprecated and will raise
in a future version. Call obj.infer_objects(copy=False) before
interpolating instead.
  df_ret = df_ret.interpolate(method="linear",columns="airPressure")
```

```
print(f"Before: \n---------\n{weather_data.isna().sum()}")
print(f"\nAfter: \n---------\n{weather_data_complete.isna().sum()}")
```

```
Before:
---------
date              0
precip            0
precipType        0
airPressure     181
skyCover          0
temp            231
hum               0
windDir           0
windBeauf         0
dtype: int64

After:
---------
date            0
precip          0
precipType      0
airPressure     0
skyCover        0
temp            0
hum             0
windDir         0
windBeauf       0
dtype: int64
```

```
# tests, DO NOT MODIFY OR COPY THIS CELL!!
# check if missing values are no longer present
assert not weather_data_complete.isna().any().any()
assert weather_data_complete.shape[0] == weather_data.shape[0]
assert weather_data_complete.shape[1] == weather_data.shape[1]
```

Task 3: Handling Outliers (10 Points)

Additionally to the missing values, the dataset also seems to have some strange values, that are probably outliers. When confronted with the data, the meterologist gave you a bit more information:

1. Sometimes the temperture readings seem to be off, without any good reason.
2. In the timespan from early October 2015 until mid March 2016, the wind sensor was defective: it might have displayed wrong values for winds from SE direction.
3. In the `daily_weather_data`, the precipitation column contains some non-numeric values.
4. The precipitation sensor usually produces wrong values when hail is involved.

3.1 Temperature outliers

First we want ot take a closer look at the temperature values. Check if we can identify some obvious outliers and come up with a strategy to handle/fix them.

In order to do so you will have to:

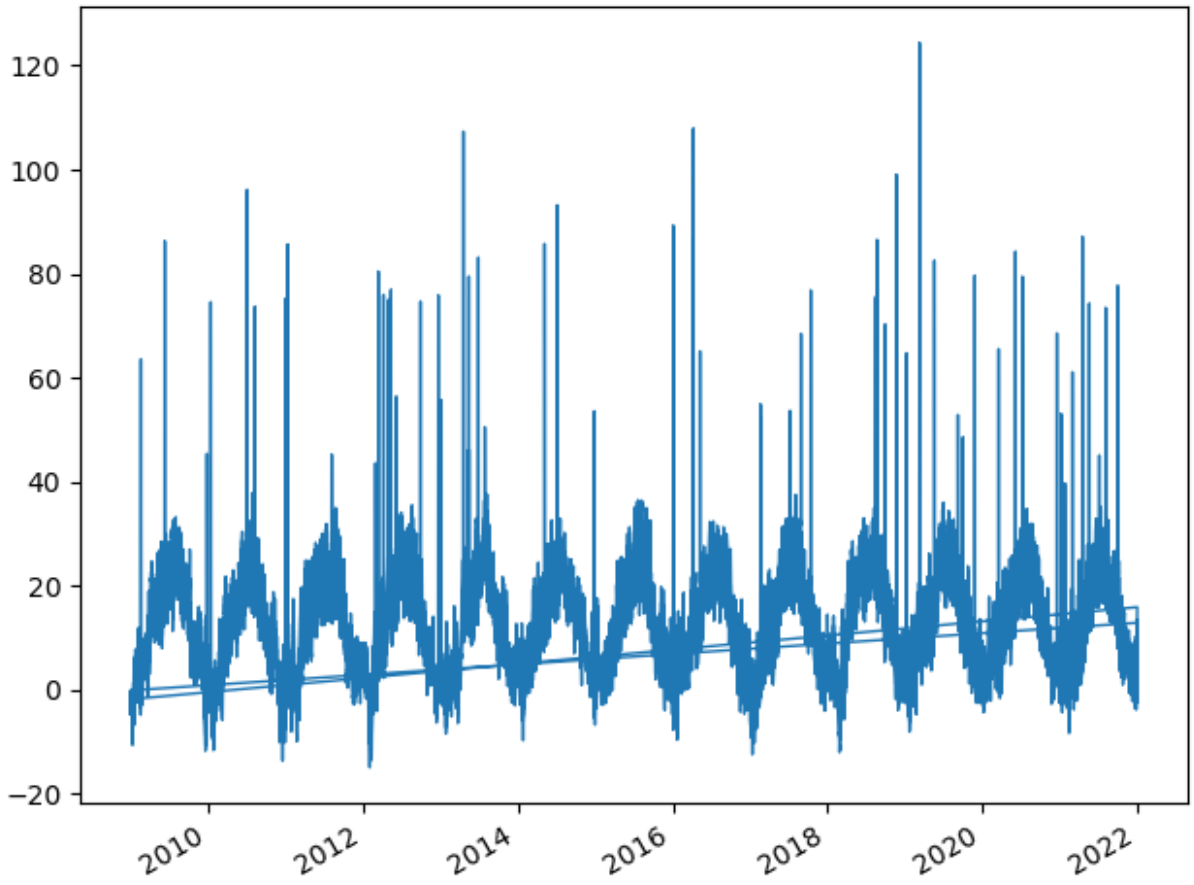
- Plot the temperature curve over time and a histogram of temperature values to identify possible outliers
- Plot a zoomed in version of individual outliers to get a better understanding what's happening
- Devise a strategy to get rid of outliers

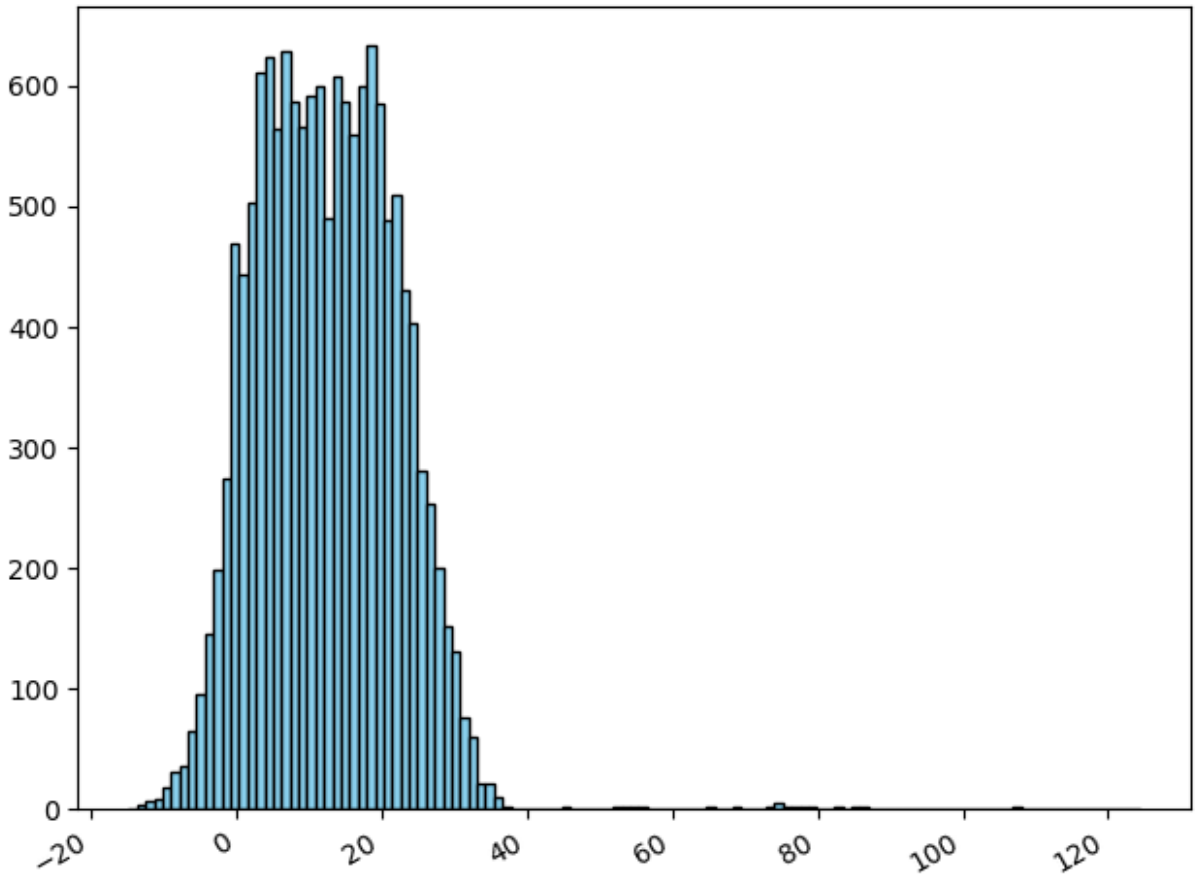
Investigation

Implement the function below to create a plot of the temperature values (`temp`) over time. Additionally create a histogram with reasonable bins to identify the outliers:

```
def plot_temp_analysis(df: pd.DataFrame) -> None:
    """
    Create two plots:
    1) Temperature values over time for the whole dataframe
    2) A histogram for temperature values.
       Choose appropriate bins enabling you to identify outliers!

    Parameters
    --------
    df: data frame containint the temperature values (temp) with
potential outlier

    """


    # YOUR CODE HERE
    plt.plot(df["date"], df["temp"], linewidth=1)
    plt.gcf().autofmt_xdate()
    plt.tight_layout()
    plt.show()

    #histo
    bins = int(np.sqrt(len(df["temp"])))  # simple bin rule: sqrt of
sample size
    plt.hist(df["temp"], bins=bins, color="skyblue",
edgecolor="black")
    plt.gcf().autofmt_xdate()
    plt.tight_layout()
    plt.show()
```

```
# DO NOT MODIFY OR COPY THIS CELL!!
plot_temp_analysis(weather_data_complete)
```





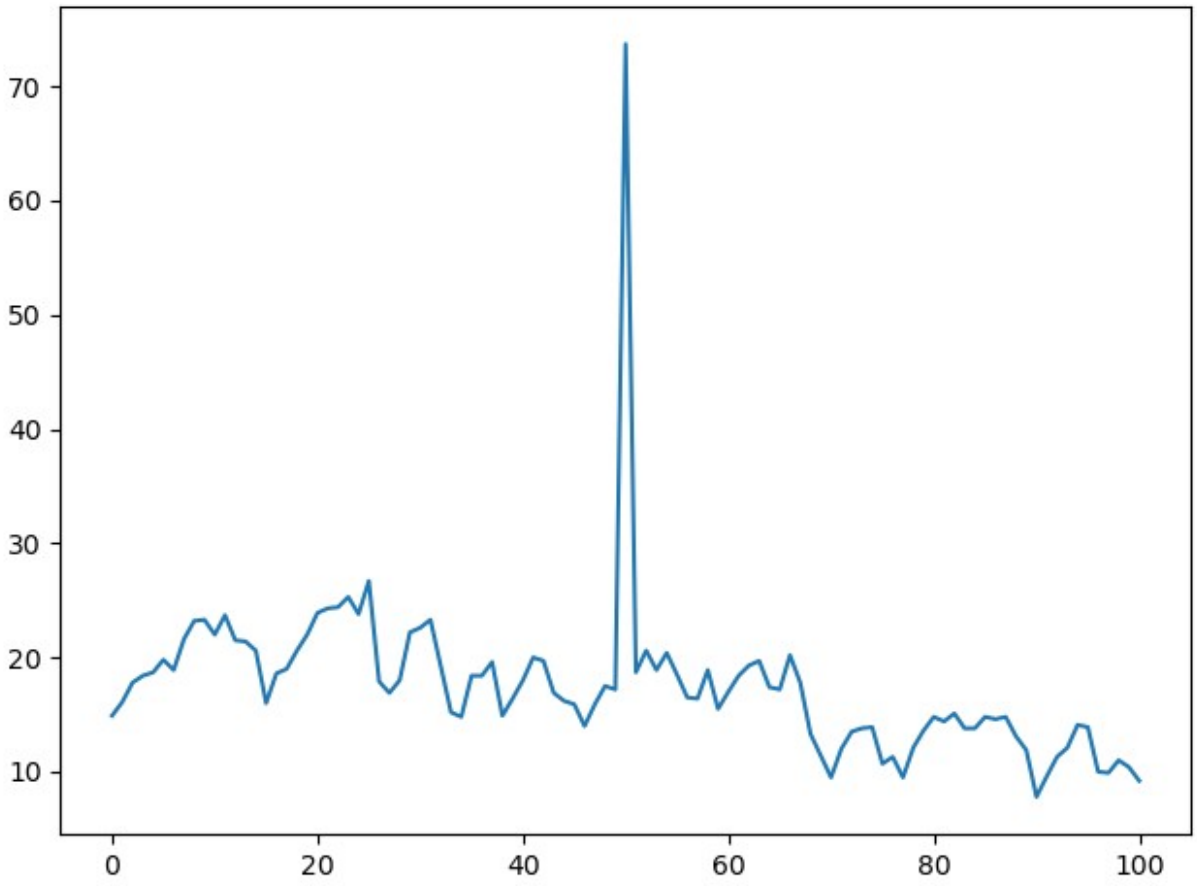

```
# hidden test, DO NOT MODIFY OR COPY THIS CELL!!
```

In the next cell, select a random outlier (e.g. the first) and plot the temperature curve around the outlier.

```
# YOUR CODE HERE
temp = weather_data_complete["temp"]
z = (temp - temp.mean()) / temp.std()
outliers = temp[abs(z) > 3]

first = outliers.index[0]
i = temp.index.get_loc(first)

w = 50
s = max(0, i - w)
e = min(len(temp), i + w + 1)
seg = temp.iloc[s:e]

print(f"First outlier: {first}, value = {temp.loc[first]:.2f}")

x = np.arange(len(seg))          # numeric x avoids tuple index issues
plt.plot(x, seg.to_numpy(), label="Temperature")
```

```
plt.tight_layout()
plt.show()
```

```
First outlier: (2010, 8, 32, 11, 7), value = 73.69
```



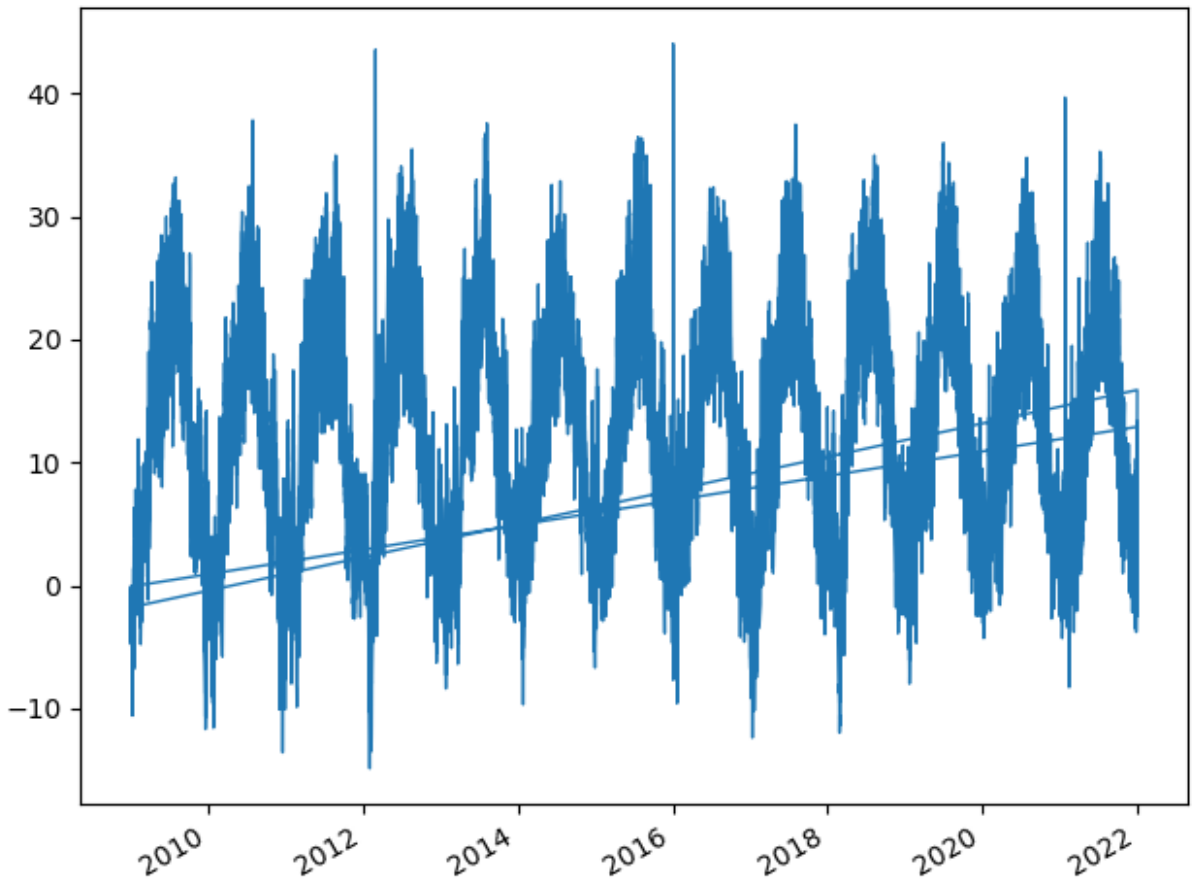

```
# hidden test, DO NOT MODIFY OR COPY THIS CELL!!
```

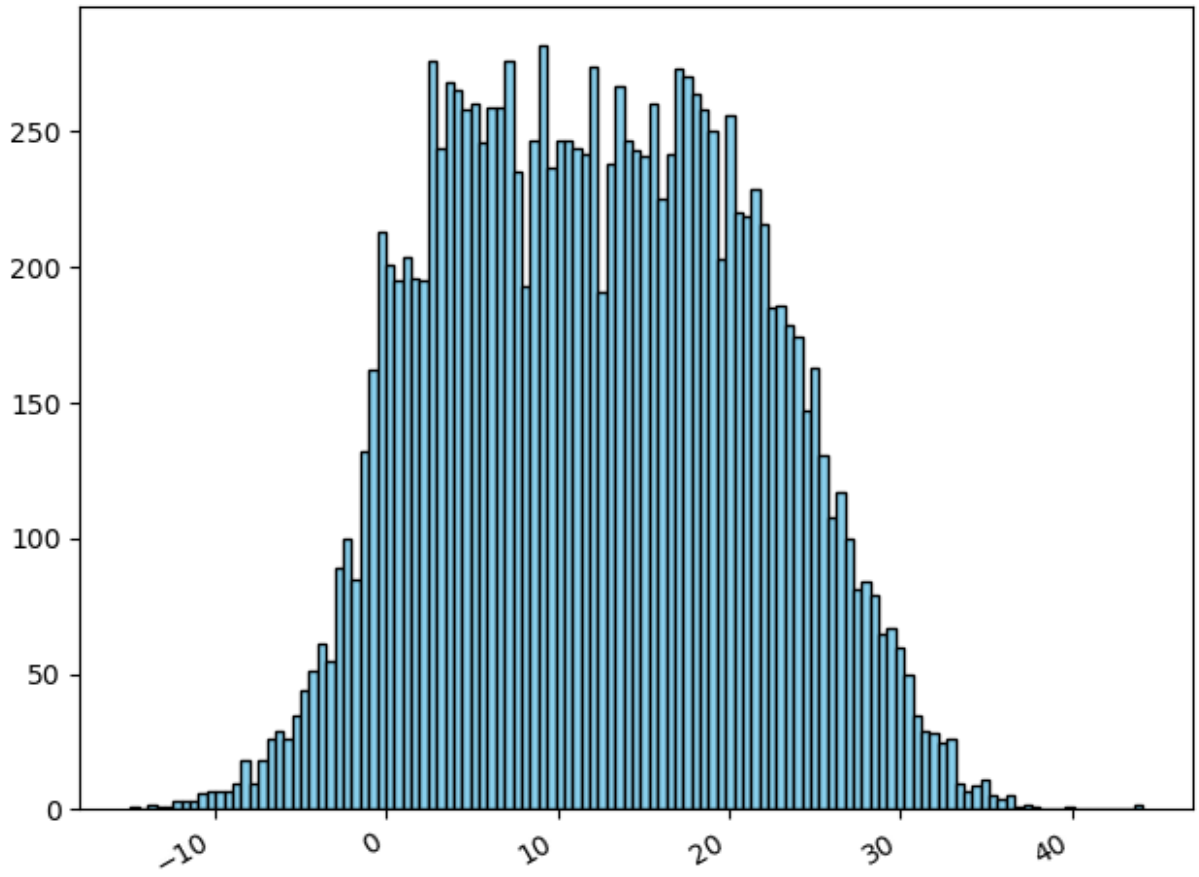
Remove temperature outliers

Implement the below function using the strategy you defined above to get rid of the temperature outliers

```
def handle_temp_outliers(noisy_data) -> pd.DataFrame:
    """
    Parameters
    --------
    noisy_data: data frame that contains temperature outliers ('temp'
column)

    Returns
    --------
```

```
    cleaned_data: data frame with temperature outliers removed/handled
    """
    cleaned_data = noisy_data.copy()

    # YOUR CODE HERE
    df = cleaned_data
    df["temp"] = pd.to_numeric(df["temp"], errors="coerce")
    df["temp"] = np.where((df["temp"] > 45) | (df["temp"] < -30),
np.nan, df["temp"])
    df["temp"] =
df["temp"].interpolate(method="linear").ffill().bfill()

    cleaned_data = df
    return cleaned_data

# DO NOT MODIFY OR COPY THIS CELL!!
weather_data_cleaned = handle_temp_outliers(weather_data_complete)
plot_temp_analysis(weather_data_cleaned)
print(weather_data_cleaned['temp'].describe())
```






```
count    14244.000000
mean        12.302089
std          9.060507
min        -14.900000
25%          5.100000
50%         12.100000
75%         19.300000
max         44.083761
Name: temp, dtype: float64
```

```
# test, DO NOT MODIFY OR COPY THIS CELL!!

# check if new missing values were introduced
assert not weather_data_cleaned.isna().any().any()
# check if outliers were simply dropped
assert weather_data_cleaned.shape == weather_data_complete.shape

# hidden test, DO NOT MODIFY OR COPY THIS CELL!!

# hidden test, DO NOT MODIFY OR COPY THIS CELL!!
```

3.2 Wind speed outliers

The second outlier problem was, that in the timespan from early October 2015 until mid March 2016, the wind sensor was defective: it might have displayed wrong values for winds from SE direction.

Double check if this is true, and if it is, fix the values appropriately.

Data inspection

Implement the function below to visualize the problematic wind sensor data. Complete the function below to check the assumtions for wrong values of windBeauf.

- Plot the wind speed data over the questionable time duration (early October 2015 until mid March 2016)
- Plot a histogram of the windspeed values to identify outliers
- Check if the assumtion regarding the wind direction is true (errors only from SE direction)

Bonus:

- Find out when exactly the sensor started to produce wrong values and when the sensor was fixed!

```
def analyze_wind_range(df:pd.DataFrame) -> typing.Tuple[int,
typing.List[str]]:
    """
    Plot analysis plots for the wind data. Print outputs about
relevant data
    and return the relevant values as indicated.

    Parameters
    --------
    noisy_data: data frame that contains outliers

    Returns
    --------
    wind_dir_outliers: list of wind direction string code, for which
outliers where found
    num_outliers: number of found outliers

    """

    num_outliers = 0  # return a value that reflects how many outliers
you identified
    wind_dir_outliers = [] # a list of the string codes for wind
directions where you identified outliers from


    # YOUR CODE HERE
    vals = pd.to_numeric(df["windBeauf"], errors="coerce")
```

```
    # treat values outside the Beaufort range as outliers
    mask = (vals < 0) | (vals > 12)

    num_outliers = int(mask.sum())
    wind_dir_outliers = sorted(
        df.loc[mask, "windDir"]
        .dropna()
        .astype(str)
        .str[:2]          # codes are two character strings
        .unique()
        .tolist()
    )

    return wind_dir_outliers, num_outliers

# DO NOT MODIFY OR COPY THIS CELL!!
wind_dir_outliers, num_outliers =
analyze_wind_range(weather_data_complete)

assert 0 < num_outliers < 500, "there should be more than zero but
less than 500 outliers!"
assert len(wind_dir_outliers) > 0, "there should be at least one wind
direction!"
assert isinstance(wind_dir_outliers[0], str), "the wind direction
codes are strings with two characters!"
assert len(wind_dir_outliers[0]) == 2, "the wind direction codes are
strings with two characters!"

# hidden test, DO NOT MODIFY OR COPY THIS CELL!!
```

```
---------------------------------------------------------------------
-----
AssertionError                            Traceback (most recent call
last)
Cell In[63], line 1
----> 1 assert 0 < num_outliers < 500, "there should be more than zero
but less than 500 outliers!"
      2 assert len(wind_dir_outliers) > 0, "there should be at least
one wind direction!"
      3 assert isinstance(wind_dir_outliers[0], str), "the wind
direction codes are strings with two characters!"

AssertionError: there should be more than zero but less than 500
outliers!
```

Fix wind sensor outliers

Fix the values appropriately. Implement a function that compensates for the problem you found.

```
def fix_windBeauf_values(df:pd.DataFrame) -> pd.DataFrame:
    """
    Parameters
    --------
    df: data frame that contains potential faulty wind values

    Returns
    -------
    df_ret: data frame with fixed wind values
    """

    df_ret = df.copy()

    # YOUR CODE HERE
    vals = pd.to_numeric(df_ret["windBeauf"], errors="coerce")

    # same outlier rule as in analyze_wind_range
    mask = (vals < 0) | (vals > 12)

    # mark outliers as missing
    df_ret.loc[mask, "windBeauf"] = np.nan
    df_ret["windBeauf"] = pd.to_numeric(df_ret["windBeauf"],
errors="coerce")

    # interpolate and fill remaining ends, then round back to integer
    df_ret["windBeauf"] =
df_ret["windBeauf"].interpolate(method="linear")
    df_ret["windBeauf"] =
df_ret["windBeauf"].ffill().bfill().round().astype(int)

    return df_ret

# test, DO NOT MODIFY OR COPY THIS CELL!!
weather_data_fix_wind = fix_windBeauf_values(weather_data_complete)
wind_dir_outliers_fixed, num_outliers_fixed =
analyze_wind_range(weather_data_fix_wind)

# test, DO NOT MODIFY OR COPY THIS CELL!!
assert num_outliers_fixed == 0, "now no outliers should be found"
assert len(wind_dir_outliers_fixed) == 0, "now no outliers should be
found, so no directions!"
assert weather_data_fix_wind.shape == weather_data_complete.shape

# hidden test, DO NOT MODIFY OR COPY THIS CELL!!
```

3.3 Daily weather data: precipitation

When loading the data, we separated the precipitation data into the `daily_weather_data` dataframe. This dataframe also has an issues:

- The `precip` column contains some non-numeric values

Data inspection

Check the occurance of the non-numeric values in the precipitation data. You can check the file `data/weather/description.txt`, which might have additional clues what is going on.

Implement the function below and return a list of the non-numeric values that occur in the `precip` column of `daily_weather_data`. Make sure to only return every unique value once!

```
def get_non_numeric_precip_values(df:pd.DataFrame) -> typing.Set[str]:
    """
    Parameters
    --------
    df: data frame that contains non-numeric values in precip column

    Returns
    -------
    non_numeric_values: list of unique non-numeric values.
    Do not return duplicate values in the list!
    """
    non_numeric_values = set()

    # YOUR CODE HERE
    s = df["precip"]
    numeric = pd.to_numeric(s, errors="coerce")
    mask = numeric.isna() & s.notna()

    non_numeric_values = set(s[mask].astype(str).unique().tolist())

    return non_numeric_values

# DO NOT MODIFY OR COPY THIS CELL!!
non_numeric_values = get_non_numeric_precip_values(daily_weather_data)
print(f"\nnon-numeric values values: {non_numeric_values}")
```

```
non-numeric values values: {'traces'}
```

```
# tests, DO NOT MODIFY OR COPY THIS CELL!!
assert isinstance(non_numeric_values, set) , "make sure to return a
set, so no duplicate values can be returned!!"
assert len(non_numeric_values) > 0, "there should be some non-numeric
values in daily_weather_data!"
assert isinstance(list(non_numeric_values)[0], str), "only return the
non-numeric values as strings!"
```

Fix non-numeric values

Replace non-numeric values with some appropriate numerical values and convert the column to a more suitable data type. To get an idea, what appropriate values might be, check the file `data/weather/description.txt` and the other numeric values in the `precip` column.

```
def fix_precip_values(df:pd.DataFrame) -> pd.DataFrame:
    """
    Parameters
    --------
    df: data frame that contains non-numeric values in precip column

    Returns
    -------
    ret_df: data frame with fixed precip values
    """
    ret_df = df.copy()

    # YOUR CODE HERE
    df = ret_df
    df["precip"] = df["precip"].astype(str)
    df["precip"] =
df["precip"].str.replace("traces","0.05").astype(float)

    ret_df = df
    return ret_df

# DO NOT MODIFY OR COPY THIS CELL!!
daily_weather_data_fixed_precip =
fix_precip_values(daily_weather_data)

# test, DO NOT MODIFY OR COPY THIS CELL!!
assert
pd.api.types.is_float_dtype(daily_weather_data_fixed_precip['precip'].
dtype), "precip should now be a float column!!"
assert daily_weather_data_fixed_precip.shape ==
daily_weather_data.shape, "do not remove or add rows!"
```

Combining the fixes

```
# DO NOT MODIFY OR COPY THIS CELL!!
def fix_values_daily(data):
    """
    Parameters
    --------
    data: data frame containing missing values

    Returns
    --------
    complete_data: data frame not containing any missing values
    """
    complete_data = data.copy()
    complete_data = fix_precip_values(complete_data)

    return complete_data
```

```
def handle_outliers(data):
    """
    Parameters
    --------
    data: data frame containing outlier values

    Returns
    --------
    complete_data: data frame not containing any outlier values
    """
    complete_data = data.copy()
    complete_data = handle_temp_outliers(complete_data)
    complete_data = fix_windBeauf_values(complete_data)

    return complete_data

# DO NOT MODIFY OR COPY THIS CELL!!
daily_weather_data_finished = fix_values_daily(daily_weather_data)
weather_data_finished = handle_outliers(weather_data_complete)
```

Task 4: Aggregate values (5 Points)

Aggregate the observations on a weekly basis. Return a data frame with a hierarchical index (levels `year` and `week`) and the following weekly aggregations as columns:

- `temp_weeklyMin`: minimum of `temp`
- `temp_weeklyMax`: max of `temp`
- `temp_weeklyMean`: mean of `temp`
- `temp_weeklyMedian`: median of `temp`
- `hum_weeklyMin`: min of `hum`
- `hum_weeklyMax`: max of `hum`
- `hum_weeklyMean`: mean of `hum`
- `wind_weeklyMean`: mean of `windBeauf`
- `wind_weeklyMax`: max of `windBeauf`
- `wind_weeklyMin`: min of `windBeauf`

Additionally merge the precipitation values from the `daily_weather_data` dataframe also into the newly created dataframe and aggregate them into the folling columns:

- `precip_weeklyMin`: min of `precip`
- `precip_weeklyMax`: max of `precip`

- `precip_weeklyMean`: mean of `precip`

Note: Attentive data scientists might have noticed a problem with isocalendars when aggregating on `Year` and `Week`. You can ignore this for the purpose of this lecture. In real-world settings you might consider addressing this issue, depending on your task and data

```
def aggregate_weekly(hourly_data, data_daily):
    """
    Parameters
    --------
    hourly_data: hourly weather data frame, containing temp, hum, and
wind values.
    data_daily: daily weather data frame with precip values

    Returns
    --------
    weekly_stats: data frame that contains statistics aggregated on a
weekly basis
    """
    weekly_weather_data = pd.DataFrame()

    # YOUR CODE HERE
    hourly = hourly_data.copy()
    daily = data_daily.copy()

    # ensure numeric types
    for col in ["temp", "hum", "windBeauf"]:
        if col in hourly.columns:
            hourly[col] = pd.to_numeric(hourly[col], errors="coerce")

    group_cols = ["year", "week"]

    # aggregate hourly variables to weekly level
    hourly_weekly = hourly.groupby(group_cols).agg(
        temp_weeklyMin   = ("temp", "min"),
        temp_weeklyMax   = ("temp", "max"),
        temp_weeklyMean  = ("temp", "mean"),
        temp_weeklyMedian= ("temp", "median"),
        hum_weeklyMin    = ("hum", "min"),
        hum_weeklyMax    = ("hum", "max"),
        hum_weeklyMean   = ("hum", "mean"),
        wind_weeklyMin   = ("windBeauf", "min"),
        wind_weeklyMax   = ("windBeauf", "max"),
        wind_weeklyMean  = ("windBeauf", "mean"),
    )
    hourly_weekly.index = hourly_weekly.index.set_names(group_cols)

    # aggregate daily precip
    if "precip" in daily.columns:
        daily["precip"] = pd.to_numeric(daily["precip"],
```

```
errors="coerce")
        daily_weekly = daily.groupby(group_cols).agg(
            precip_weeklyMin  = ("precip", "min"),
            precip_weeklyMax  = ("precip", "max"),
            precip_weeklyMean = ("precip", "mean"),
        )
        daily_weekly.index = daily_weekly.index.set_names(group_cols)

        weekly = hourly_weekly.join(daily_weekly, how="inner")
    else:
        weekly = hourly_weekly

    weekly = weekly.sort_index()
    weekly_weather_data = weekly
    return weekly_weather_data

# DO NOT MODIFY OR COPY THIS CELL!!
weekly_weather_data = aggregate_weekly(weather_data_finished,
daily_weather_data_finished)
display(weekly_weather_data)
```

```
           temp_weeklyMin  temp_weeklyMax  temp_weeklyMean
temp_weeklyMedian  \
year week

2009 1            40.4004    4.040940e+01     4.040640e+01
40.4084
     2            40.4004    4.040840e+01     4.040383e+01
40.4044
     3            40.4044    4.034040e+04     2.298326e+04
40040.4094
     4         40040.4034    4.074041e+04     4.033564e+04
40340.4024
     5         40040.4024    4.044041e+04     4.014040e+04
40140.4004
...                   ...             ...              ...
...
2021 49        40040.4004    4.044041e+04     4.019830e+04
40140.4064
     50        40240.4054    4.084040e+04     4.056898e+04
40540.4094
     51           40.4024    4.014004e+07     2.893148e+06
40240.4084
     52        40040.4004    4.014054e+07     8.633226e+06
40540.4049
     53        40040.4014    4.054040e+04     4.035707e+04
40390.4039

           hum_weeklyMin  hum_weeklyMax  hum_weeklyMean
wind_weeklyMin  \
```

```
year week

2009 1                62              83       70.500000
0
     2                61              91       78.523810
0
     3                69              95       86.428571
1
     4                40              94       76.666667
1
     5                66              94       84.238095
1
...                  ...             ...             ...               ..
.
2021 49               66              97       82.809524
1
     50               65              91       78.000000
1
     51               50              98       76.619048
1
     52               62              98       84.466667
1
     53               84              99       91.888889
1

           wind_weeklyMax  wind_weeklyMean  precip_weeklyMin  \
year week
2009 1                  4         2.333333              0.00
     2                  4         2.095238              0.00
     3                  4         2.095238              0.00
     4                  4         2.238095              0.00
     5                  4         2.714286              0.00
...                   ...              ...               ...
2021 49                 4         2.333333              0.00
     50                 4         3.142857              0.00
     51                 4         2.380952              0.00
     52                 4         2.200000              0.00
     53                 3         1.666667              0.05

           precip_weeklyMax  precip_weeklyMean
year week
2009 1                 0.05           0.025000
     2                 0.05           0.035714
     3                 4.30           0.692857
     4                 9.10           2.500000
     5                 6.10           1.942857
...                     ...                ...
2021 49                8.10           2.078571
     50                1.60           0.335714
     51                2.70           0.950000
```

```
     52                 7.30            2.030000
     53                 0.70            0.283333

[682 rows x 13 columns]
```

```
# tests, DO NOT MODIFY OR COPY THIS CELL!!
assert len(weekly_weather_data.columns) >= 13, "according to the
instructions, the dataframe should have >= 13 columns"
assert len(weekly_weather_data.index.levels) == 2, "according to the
instructions, the dataframe should have a multi-index with 2 levels"

# hidden tests, DO NOT MODIFY OR COPY THIS CELL!!
```

Task 5: Merge influenza and weather datasets (5 Points)

Merge the `data_weather_weekly` and `data_influenza` datasets. Both dataframes should now be on a weekly index. Beware that both datasets contain rows that do not appear in the other dataset.

```
# tests, DO NOT MODIFY OR COPY THIS CELL!!
# Neither of the tables contain missing data

print(f"influenza data, missing data (should be 0): \
n{data_influenza.isna().sum()}")
print(f"weather data, missing data (should be 0): \
n{weekly_weather_data.isna().sum()}")

assert not data_influenza.isna().any().any(), "we should have
eliminated all missing values!!"
assert not weekly_weather_data.isna().any().any(), "we should have
eliminated all missing values!!"
```

```
influenza data, missing data (should be 0):
weekly infections    0
dtype: int64
weather data, missing data (should be 0):
temp_weeklyMin       1
temp_weeklyMax       1
temp_weeklyMean      1
temp_weeklyMedian    1
hum_weeklyMin        0
                    ..
wind_weeklyMax       0
wind_weeklyMean      0
precip_weeklyMin     0
precip_weeklyMax     0
precip_weeklyMean    0
Length: 13, dtype: int64
```

```
---------------------------------------------------------------------
-----
AssertionError                            Traceback (most recent call
last)
Cell In[81], line 8
      5 print(f"weather data, missing data (should be 0): \
n{weekly_weather_data.isna().sum()}")
      7 assert not data_influenza.isna().any().any(), "we should have
eliminated all missing values!!"
----> 8 assert not weekly_weather_data.isna().any().any(), "we should
have eliminated all missing values!!"

AssertionError: we should have eliminated all missing values!!
```

```
# use this cell for experimentation / analysis for merging data

# YOUR CODE HERE
raise NotImplementedError()
```

```
---------------------------------------------------------------------
-----
NotImplementedError                       Traceback (most recent call
last)
Cell In[82], line 4
      1 # use this cell for experimentation / analysis for merging
data
      2
      3 # YOUR CODE HERE
----> 4 raise NotImplementedError()

NotImplementedError:
```

```
def merge_data(weather_df, influenza_df):
    """
    Parameters
    --------
    weather_df: weekly weather data frame
    influenza_df: influenza data frame

    Returns
    --------
    merged_data: merged data frame that contains both weekly weather
observations and prevalence of influence infections
    """
    # YOUR CODE HERE
    merged = weather_df.join(influenza_df, how="inner")
    merged_data = merged
    return merged_data
```

```
# DO NOT MODIFY OR COPY THIS CELL!!
data_merged = merge_data(weekly_weather_data, data_influenza)
data_merged.head()
```

```
           temp_weeklyMin  temp_weeklyMax  temp_weeklyMean
temp_weeklyMedian  \
year week

2009 40      4.094040e+04    4.024034e+07     3.634046e+07
4.014054e+07
     41      4.014014e+07    4.024074e+07     4.015951e+07
4.014044e+07
     42      4.024040e+04    4.014014e+07     3.859521e+06
4.044041e+04
     43      4.034041e+04    4.014034e+07     1.340715e+07
4.084041e+04
     44      4.014040e+04    4.014034e+07     1.913551e+07
4.084041e+04

           hum_weeklyMin  hum_weeklyMax  hum_weeklyMean
wind_weeklyMin  \
year week

2009 40               46             95       71.000000
0
     41               48             90       73.571429
1
     42               54             91       77.333333
1
     43               63             96       81.238095
1
     44               66             97       81.809524
0

           wind_weeklyMax  wind_weeklyMean  precip_weeklyMin  \
year week
2009 40                 4         2.285714               0.0
     41                 4         2.142857               0.0
     42                 5         3.428571               0.1
     43                 3         1.904762               0.0
     44                 4         2.047619               0.0

           precip_weeklyMax  precip_weeklyMean weekly infections
year week
2009 40                 0.2           0.057143              6600
     41                 4.2           1.700000              7100
     42                 9.3           3.157143              7700
     43                 1.9           0.314286              8300
     44                 4.7           0.707143              8600
```

```
# tests, DO NOT MODIFY OR COPY THIS CELL!!
print(data_merged.shape)
assert data_merged.shape[0] > 300, "there should be more than 300 rows
in the merged dataset"
assert data_merged.shape[1] >= 14, "1 column for infections, 13
feature columns from weather data"
```

```
(310, 14)
```