Time Series Analysis

Yair Mau

Table of contents

about

Welcome to **Time Series Analysis for Environmental Sciences** (71606) at the Hebrew University of Jerusalem. This is Yair Mau, your host for today. I am a senior lecturer at the Institute of Environmental Sciences, at the Faculty of Agriculture, Food and Environment, in Rehovot, Israel.

This website contains (almost) all the material you'll need for the course. If you find any mistakes, or have any comments, please email me.

disclaimer

The material here is not comprehensive and `does not` constitute a stand alone course in Time Series Analysis. This is only the support material for the actual presential course I give.

what, who, when and where?

Course number 71606, 3 academic points
Yair Mau (lecturer), Erez Feuer (TA)
Tuesdays, from 14:15 to 17:00
Computer classroom #18
Office hours: upon request

syllabus

course description

Data analysis of time series, with practical examples from environmental sciences.

course aims

This course aims at giving the students a broad overview of the main steps involved in the analysis of time series: data management, data wrangling, visualization, analysis, and forecast. The course will provide a hands-on approach, where students will actively engage with real-life datasets from the field of environmental science.

learning outcomes

On successful completion of this module,students should be able to:

- Explore a time-series dataset, while formulating interesting questions.
- Choose the appropriate tools to attack the problem and answer the questions.
- Communicate their findings and the methods they used to achieve them, using graphs, statistics, text, and a well-documented code.

course content

- **Data wrangling:** organization, cleaning, merging, filling gaps, excluding outliers, smoothing, resampling.
- **Visualization:** best practices for graph making using leading python libraries.
- **Analysis:** stationarity, seasonality, (auto)correlations, lags, derivatives, spectral analysis.
- **Forecast:** ARIMA
- **Data management:** how to plan ahead and best organize large quantities of data. If there is enough time, we will build a simple time-series database.

books and other sources

Click here.

course evaluation

There will be assignments during the semester (totaling 50% of the final grade), and one final project (50%).

Evaluation policy

- **Individual Work:** While we support helping your peers, it's important to remember that all assignments must be completed individually. This means that your submissions should be your own unique work and not contain code or text that is identical to someone else's.
- **Zero Plagiarism:** Do not copy text verbatim from any source. Always express ideas in your own words.
- **On-Time Submission:** Assignments must be turned in by the specified deadline. Late submissions will receive a grade of 0. If you require an extension, requests will only be considered if made at least 24 hours before the due date.
- **Non-Compliance Consequence:** Assignments that do not adhere to these guidelines will automatically receive a grade of 0.

weekly program

This year's course will be a bit different that planned due to the shortening of the academic semester. The information below is NOT up to date. Ask Yair what is relevant this year.

week 1

- **Lecture:** Course overview, setting of expectations. Introduction, basic concepts, continuous vs discrete time series, sampling, aliasing
- **Exercise:** Loading csv file into python, basic time series manipulation with pandas and plotting

week 2

- **Lecture:** Filling gaps, removing outliers
- **Exercise:** Practice the same topics learned during the lecture. Data: air temperature and relative humidity

week 3

- **Lecture:** Interpolation, resampling, binning statistics
- **Exercise:** Practice the same topics learned during the lecture. Data: air temperature and relative humidity, precipitation

week 4

- **Lecture:** Time series plotting: best practices. Dos and don'ts and maybes
- **Exercise:** Practice with Seaborn, Plotly, Pandas, Matplotlib

Project 1
Basic data wrangling, using real data (temperature, relative humidity, precipitation) downloaded from USGS. 25% of the final grade

week 5

- **Lecture:** Smoothing, running averages, convolution
- **Exercise:** Practice the same topics learned during the lecture. Data: sap flow, evapotranspiration

week 6

- **Lecture:** Strong and weak stationarity, stochastic processes, auto-correlation
- **Exercise:** Practice the same topics learned during the lecture. Data: temperature and wind speed

week 7

- **Lecture:** Correlation between signals. Pearson correlation, time-lagged cross-correlations, dynamic time warping
- **Exercise:** Practice the same topics learned during the lecture. Data: temperature, solar radiation, relative humidity, soil moisture, evapotranspiration

week 8

Same as lecture 7 above

week 9

- **Lecture:** Download data from repositories, using API, merging, documentation
- **Exercise:** Download data from USGS, NOAA, Fluxnet, Israel Meteorological Service

Project 2
Students will study a Fluxnet site of their choosing. How do gas fluxes (CO_2, H_2O) depend on environmental conditions? 25% of the final grade

week 10

- **Lecture:** Fourier decomposition, filtering, Nyquist–Shannon sampling theorem
- **Exercise:** Practice the same topics learned during the lecture. Data: dendrometer data

week 11

- **Lecture:** Seasonality, seasonal decomposition (trend, seasonal, residue), Hilbert transform
- **Exercise:** Practice the same topics learned during the lecture. Data: monthly atmospheric CO_2 concentration, hourly air temperature

week 12

- **Lecture:** Derivatives, differencing
- **Exercise:** Practice the same topics learned during the lecture. Data: dendrometer data

week 13

- **Lecture:** Forecasting. ARIMA
- **Exercise:** Practice the same topics learned during the lecture. Data: vegetation variables (sap flow, ET, DBH, etc)

Final Project
In consultation with the lecturer, students will ask a specific scientific question about a site of their choosing (from NOAA, USGS, Fluxnet), and answer it using the tools learned during the semester. The report will be written in Jupyter Notebook, combining in one document all the calculations, documentation, figures, analysis, and discussion. 50% of the final grade.

who cares?

why "Time Series Analysis?"

Time has two aspects. There is the arrow, the running river, without which there is no change, no progress, or direction, or creation. And there is the circle or the cycle, without which there is chaos, meaningless succession of instants, a world without clocks or seasons or promises.
URSULA K. LE GUIN

You are here because you are interested in how things change, evolve. In this course I want to discuss with you how to make sense of data whose temporal nature is in its very essence. We will talk about randomness, cycles, frequencies, correlations, and more.

why "Environmental Sciences"

This same time series analysis (TSA) course could be called instead "TSA for finance", "TSA for Biology", or any other application. The emphasis in this course is **not** Environmental Sciences, but the concepts and tools of TSA. Because my research is in Environmental Science, and many of the graduate students at HUJI-Rehovot research this, I chose to use examples "close to home". The same toolset should be useful for students of other disciplines.

what is it good for?

In many fields of science we are flooded by data, and it's hard to see the forest for the trees. I hope that the topics we'll discuss in this course can help you find meaningful patterns in your data, formulate interesting hypotheses, and design better experiments.

do I need it?

Maybe. If you are a grad student and you have temporal data to analyze, then probably yes. However, I have very fond memories of courses that I took as a grad student that were completely unrelated to my research. Sometimes "because it's fun" is a perfectly good answer.

what will I actually gain from it?

By the end of this course you will have gained:

- a **hands-on** experience of fundamental time-series analysis tools
- an **intuition** regarding the basic concepts
- **technical** abilities
- a **springboard** for learning more about the subject by yourself

Part I

start here

1 the boring stuff you absolutely need to do

I assume everyone registered has taken a basic Python course. On your computer, do the following:

1.1 Anaconda

Install Anaconda's Python distribution. The Anaconda installation brings with it all the main python packages we will need to use. In order to install extra packages, refer to these two tutorials: tutorial 1, tutorial 2.

1.2 VSCode

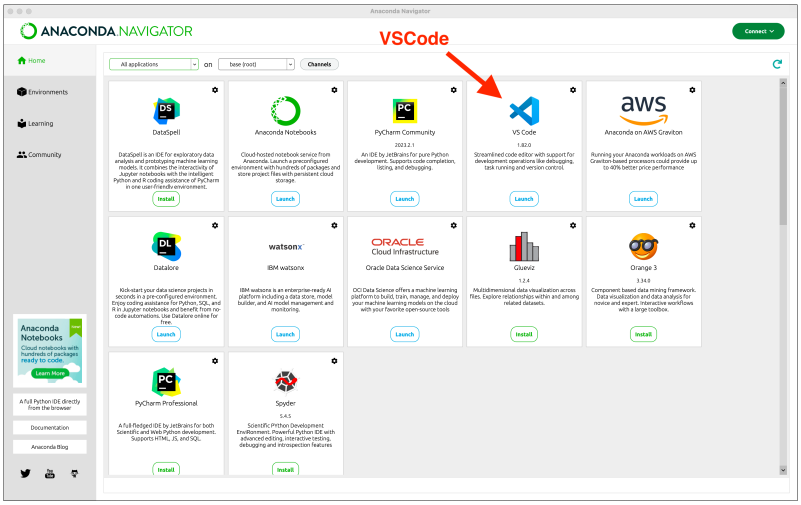
Install VSCode. Visual Studio Code is a very nice IDE (Integrated Development Environment) made by Microsoft, available to all operating systems. Contrary to the title of this page, it is not absolutely necessary to use it, but I like VSCode, and as my student, so do you .

1.3 jupyter notebooks

We will code exclusively in Jupyter Notebooks. Get acquainted with them. Make sure you can point VSCode to the Anaconda environment of your choice ("base" by default). Don't worry, this is easier than it sounds.

One failproof way of making sure VSCode uses the Anaconda installation is the following:

- Open Anaconda Navigator
- If you are using HUJI's computers, in "Environments", choose "asgard". If you are using your own computer, ignore this step.
- open VSCode from inside Anaconda Navigator (see image below).



Sometimes you will need to manualy install the Jupyter extension on VSCode. In this case follow this tutorial.

1.4 folder structure

You **NEED** to be confortable with you computer's folder (or directory) structure. Where are files located? How to navigate through different folders? How is my stuff organized? If you don't feel **absolutely** comfortable with this, then read this, Windows, MacOS. If you use Linux then you surely know this stuff. **Make yourself a "time-series" folder** wherever you want, and have it backed up regularly (use Google Drive, Dropbox, do it manually, etc). "My dog deleted my files" is not an excuse.

2 numpy, pandas, matplotlib

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

The three lines above are the most common way you will start every project in this course.

- **numpy** = numerical python. This library has a ton useful mathematical functions, and most importantly, it has an object called `numpy array`, which is one of the most useful data structures we have for time series analysis.
- **pandas** is built upon numpy, and allows us to easily manipulate data stored in `dataframes`, a fancy name for a table.
- **pyplot** is a submodule of `matplotlib`, and allows us to beautifully plot data.

The best resource I know to get acquainted with all three packages is Python Data Science Handbook, by Jake VanderPlas. This is a free online book, with excellent step by step examples.

2.1 pandas

We will primarily use the Pandas package to deal with data. Pandas has become the standard Python tool to manipulate time series, and you should get acquainted with its basic usage. This course will provide you the opportunity to learn by example, but I'm sure we will only scratch the surface, and you'll be left with lots of questions.

I provide below a (non-comprehensive) list of useful tutorials, they are a good reference for the beginner and for the experienced user.

- Python Data Science Handbook, by Jake VanderPlas
- Data Wrangling with pandas Cheat Sheet
- Working with Dates and Times in Python
- Cheat Sheet: The pandas DataFrame Object
- YouTube tutorials by Corey Schafer

2.2 pyplot

Matplotlib, and its submodule pyplot, are probably the most common Python plotting tool. Pyplot is both great and horrible:

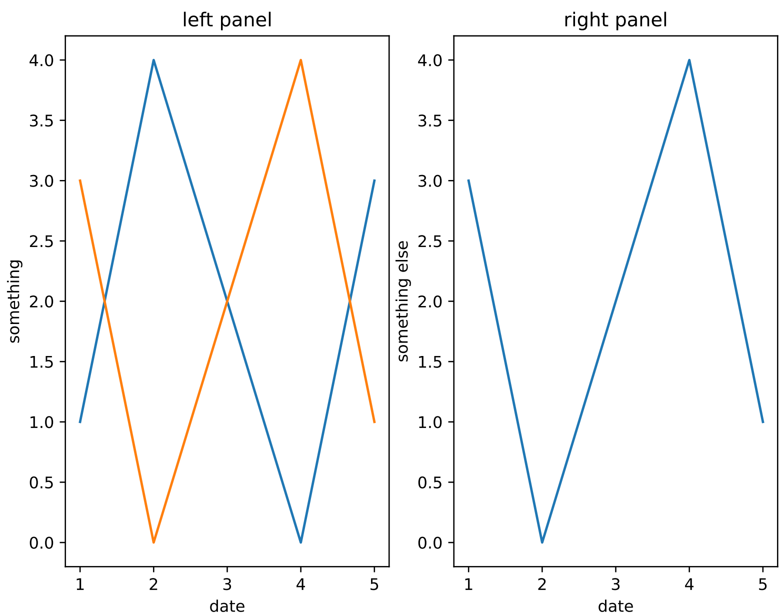
- Great: you'll have absolutely full control of everything you want to plot. The sky is the limit.
- Horrible: you'll cry as you do it, because there is so much to know, and it is not the most friendly plotting package.

Pyplot is *object oriented*, so you will usually manipulate the **axes** object like this.

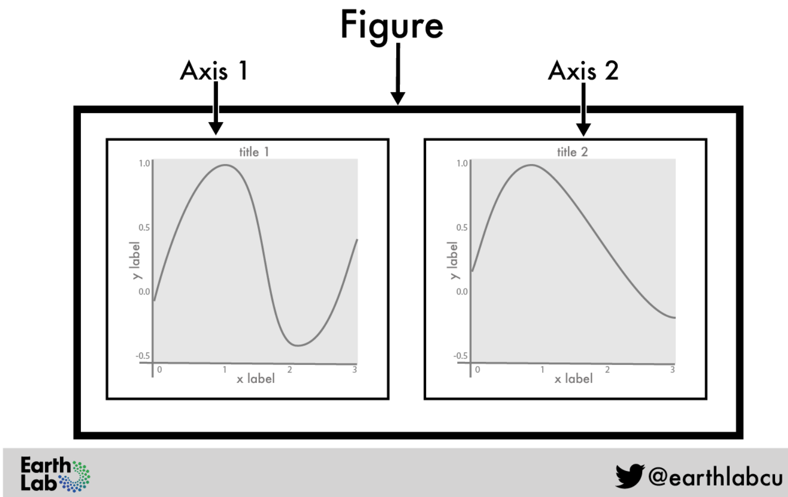
```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [1, 4, 2, 0, 3]

# Figure with two plots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (8, 6))
# plot on the left
ax1.plot(x, y, color="tab:blue")
ax1.plot(x, y[::-1], color="tab:orange")
ax1.set(xlabel="date",
        ylabel="something",
        title="left panel")
# plot on the right
ax2.plot(x, y[::-1])
ax2.set(xlabel="date",
```

```
            ylabel="something else",
            title="right panel")
```

```
[Text(0.5, 0, 'date'),
 Text(0, 0.5, 'something else'),
 Text(0.5, 1.0, 'right panel')]
```

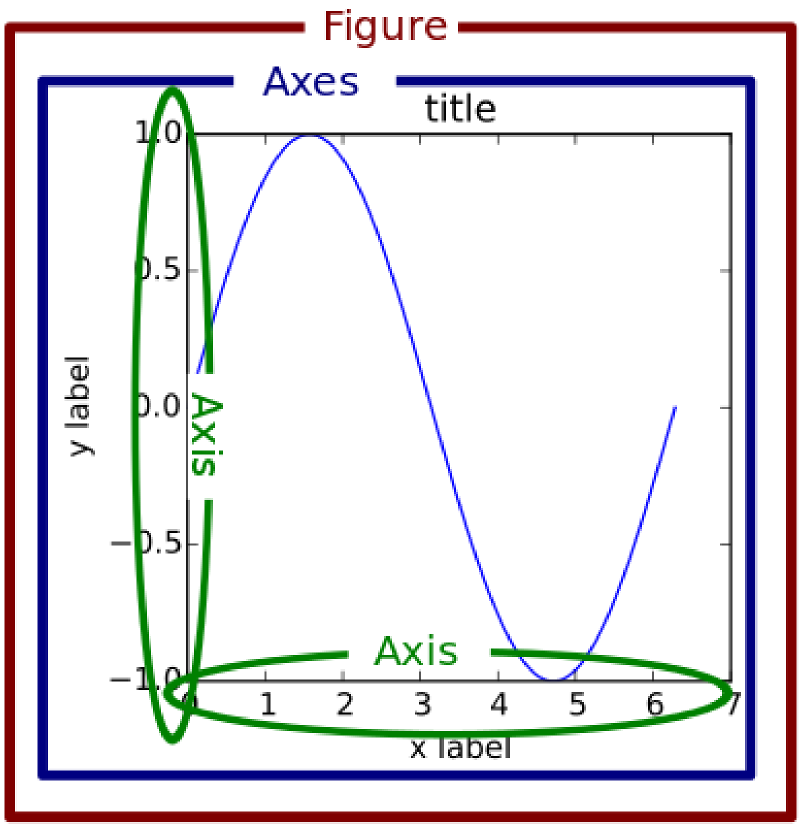


For the very beginners, you need to know that `figure` refers to the whole white canvas, and `axes` means the rectangle inside which something will be plotted:



The image above is good because it has 2 panels, and it's easy to understand what going on. Sadly, they mixed the two terms, axis and axes.

- **axes** is where the whole plot will be drawn. In the figure above it is the same as each panel.
- **axis** is each of the vertical and horizontal lines, where you have ticks and numbers.



If you are new to all this, I recommend that you go to:

- Earth Lab's Introduction to Plotting in Python Using Matplotlib
- Jake VanderPlas's Python Data Science Handbook

3 learn by example

Now that everything is installed, try to run the code below *before* the first lecture. Don't worry if you don't understand everything.

- If you manage to run everything without errors, this means that your computer is good to go!
- You might encounter a few problems. That's ok. Make a note and we will solve everything in the first lecture.

Let's make a first plot of real data. We will use NOAA's Global Monitoring Laboratory data on Trends in Atmospheric Carbon Dioxide.

3.1 open a new Jupyter Notebook

1. On your computer, open the program `Anaconda Navigator` (it may take a while to load).
2. Find the white box called `VS Code` and click `Launch`.
3. Now go to `File` > `Open Folder`, and open the folder you created for this course. VS Code may ask you if you trust the authors, and the answer is "yes" (it's your computer).
4. `File` > `New File`, and call it `example.ipynb`
5. You can start copying and pasting code from this website to your Jupyter Notebook. To run a cell, press Shift+Enter.
6. You may be asked to choose to Select Kernel. This is VS Code wanting to know which python installation to use. Click on "Python Environments", and then choose the option with the word `anaconda` in it.
7. That's all! Congratulations!

3.2 import packages

First, import packages to be used. They should all be already included in the Anaconda distribution you installed.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
```

3.3 load data

Load CO2 data into a Pandas dataframe. You can load it directly from the URL (option 1), or first download the CSV to your computer and then load it (option 2). The link to download the data directly form NOAA is this. If for some reason this doesn't work, download here.

```
# option 1: load data directly from URL
# url = "https://gml.noaa.gov/webdata/ccgg/trends/co2/co2_weekly_mlo.csv"
# df = pd.read_csv(url,
#                   header=34,
#                   na_values=[-999.99]
#                   )

# option 2: download first (use the URL above and save it to your computer), then load csv
filename = "co2_weekly_mlo.csv"
df = pd.read_csv(filename,
                comment='#',  # will ignore rows starting with #
                 na_values=[-999.99]  # substitute -999.99 for NaN (Not a Number), data not
                 )
# check how the dataframe (table) looks like
df
```

	year	month	day	decimal	average	ndays	1 year ago	10 years ago	increase since 1800
0	1974	5	19	1974.3795	333.37	5	NaN	NaN	50.39
1	1974	5	26	1974.3986	332.95	6	NaN	NaN	50.05

	year	month	day	decimal	average	ndays	1 year ago	10 years ago	increase since 1800
2	1974	6	2	1974.4178	332.35	5	NaN	NaN	49.59
3	1974	6	9	1974.4370	332.20	7	NaN	NaN	49.64
4	1974	6	16	1974.4562	332.37	7	NaN	NaN	50.06
...	...	...	...	...	...	...	...	...	...
2566	2023	7	23	2023.5575	421.28	4	418.03	397.30	141.60
2567	2023	7	30	2023.5767	420.83	6	418.10	396.80	141.69
2568	2023	8	6	2023.5959	420.02	6	417.36	395.65	141.41
2569	2023	8	13	2023.6151	418.98	4	417.25	395.24	140.89
2570	2023	8	20	2023.6342	419.31	2	416.64	395.22	141.71

3.4 dealing with dates

Create a new column called `date`, that combines the information from three separate columns: `year`, `month`, `day`.

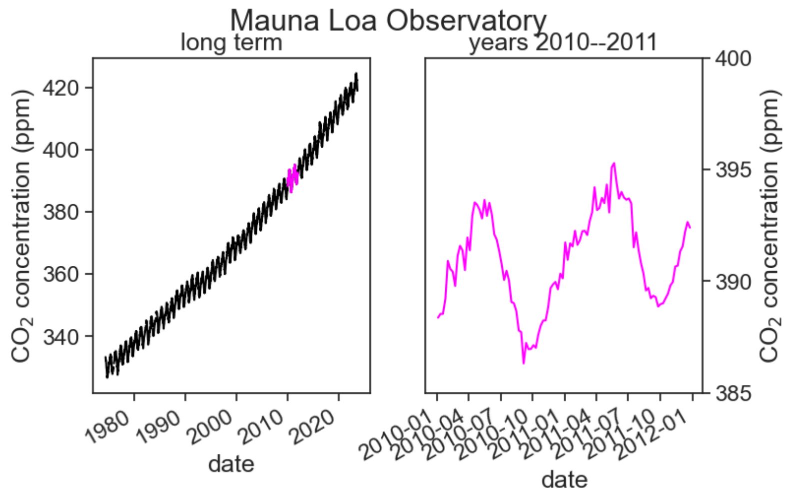
```
# function to_datetime translates the full date into a pandas datetime object,
# that is, pandas knows this is a date, it's not just a string
df['date'] = pd.to_datetime(df[['year', 'month', 'day']])
# make 'date' column the dataframe index
df = df.set_index('date')
# now see if everything is ok
df
```

	year	month	day	decimal	average	ndays	1 year ago	10 years ago	increase since 1800
date									
1974-05-19	1974	5	19	1974.3795	333.37	5	NaN	NaN	50.39
1974-05-26	1974	5	26	1974.3986	332.95	6	NaN	NaN	50.05
1974-06-02	1974	6	2	1974.4178	332.35	5	NaN	NaN	49.59
1974-06-09	1974	6	9	1974.4370	332.20	7	NaN	NaN	49.64
1974-06-16	1974	6	16	1974.4562	332.37	7	NaN	NaN	50.06
...	...	...	...	...	...	...	...	...	...
2023-07-23	2023	7	23	2023.5575	421.28	4	418.03	397.30	141.60
2023-07-30	2023	7	30	2023.5767	420.83	6	418.10	396.80	141.69
2023-08-06	2023	8	6	2023.5959	420.02	6	417.36	395.65	141.41
2023-08-13	2023	8	13	2023.6151	418.98	4	417.25	395.24	140.89
2023-08-20	2023	8	20	2023.6342	419.31	2	416.64	395.22	141.71

3.5 first plot

We are now ready for our first plot! Let's see the weekly CO2 average.

```
# %matplotlib widget
# uncomment the above line if you want dynamic control of the figure when using VSCode
fig, (ax1, ax2) = plt.subplots(1, 2,  # 1 row, 2 columns
                               figsize=(8,5)  # width, height, in inches
                               )
# left panel
ax1.plot(df['average'], color="black")
ax1.plot(df.loc['2010-01-01':'2011-12-31','average'], color="magenta")
ax1.set(xlabel="date",
       ylabel=r"CO$_2$ concentration (ppm)",
       title="long term");
# right panel
ax2.plot(df.loc['2010-01-01':'2011-12-31','average'], color="magenta")
ax2.set(xlabel="date",
        ylabel=r"CO$_2$ concentration (ppm)",
        ylim=[385, 400],  # choose y limits
        yticks=np.arange(385, 401, 5),  # choose ticks
        title="years 2010--2011");
# put ticks and label on the right for ax2
ax2.yaxis.tick_right()
ax2.yaxis.set_label_position("right")
# title above both panels
fig.suptitle("Mauna Loa Observatory")
# makes slanted dates
plt.gcf().autofmt_xdate()
```



3.6 first plot, v2.0

The dates in the x-label are not great. Let's try to make them prettier.

We need to import a few more packages first.

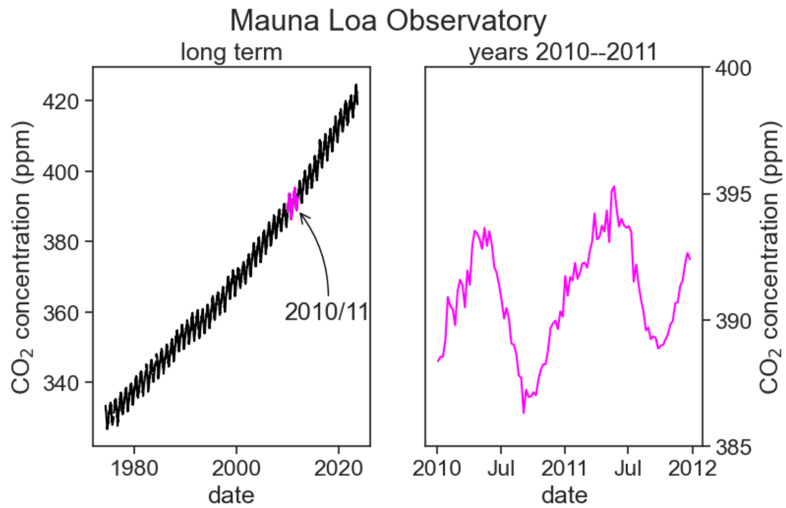
```
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()  # datetime converter for a matplotlib
```

Now let's replot.

```
# %matplotlib widget
# uncomment the above line if you want dynamic control of the figure when using VSCode
fig, (ax1, ax2) = plt.subplots(1, 2,  # 1 row, 2 columns
                               figsize=(8,5)  # width, height, in inches
                               )
# left panel
ax1.plot(df['average'], color="black")
ax1.plot(df.loc['2010-01-01':'2011-12-31','average'], color="magenta")
ax1.set(xlabel="date",
       ylabel=r"CO$_2$ concentration (ppm)",
```

```python
       title="long term");
# right panel
ax2.plot(df.loc['2010-01-01':'2011-12-31','average'], color="magenta")
ax2.set(xlabel="date",
        ylabel=r"CO$_2$ concentration (ppm)",
        ylim=[385, 400],  # choose y limits
        yticks=np.arange(385, 401, 5),  # choose ticks
        title="years 2010--2011");
# put ticks and label on the right for ax2
ax2.yaxis.tick_right()
ax2.yaxis.set_label_position("right")
# title above both panels
fig.suptitle("Mauna Loa Observatory", y=1.00)

locator = mdates.AutoDateLocator(minticks=3, maxticks=5)
formatter = mdates.ConciseDateFormatter(locator)
ax1.xaxis.set_major_locator(locator)
ax1.xaxis.set_major_formatter(formatter)

locator = mdates.AutoDateLocator(minticks=4, maxticks=5)
formatter = mdates.ConciseDateFormatter(locator)
ax2.xaxis.set_major_locator(locator)
ax2.xaxis.set_major_formatter(formatter)

ax1.annotate(
    "2010/11",
    xy=('2011-12-25', 389),  xycoords='data',
    xytext=(-10, -80), textcoords='offset points',
    arrowprops=dict(arrowstyle="->",
                    color="black",
                    connectionstyle="arc3,rad=0.2"))
fig.savefig("CO2-graph.png", dpi=300)
```

```
/var/folders/hc/jhnmlst937d27zzq9fhfks780000gn/T/ipykernel_10652/850389963.py:42: UserWarning:
  fig.savefig("CO2-graph.png", dpi=300)
/opt/anaconda3/lib/python3.9/site-packages/IPython/core/pylabtools.py:151: UserWarning: AutoDa
  fig.canvas.print_figure(bytes_io, **kw)
```



The dates on the horizontal axis are determined thus:

1. `locator = mdates.AutoDateLocator(minticks=3, maxticks=5)`
   This deremines the location of the ticks (between 3 and 5 ticks, whatever "works best")
2. `ax1.xaxis.set_major_locator(locator)`
   This actually puts the ticks in the positions determined above
3. `formatter = mdates.ConciseDateFormatter(locator)`
   This says that the labels will be placed at the locations determined in 1.
4. `ax1.xaxis.set_major_formatter(formatter)`
   Finally, labels are written down

The arrow is placed in the graph using `annotate`. It has a tricky syntax and a million options. Read Jake VanderPlas's excellent examples to learn more.

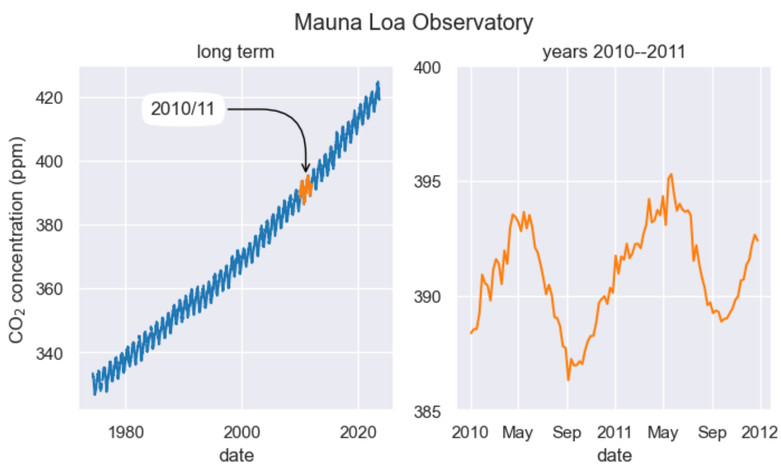
3.7 modifications

Let's change a lot of plotting options to see how things could be different.

```
sns.set(style="darkgrid")
sns.set_context("notebook")

# %matplotlib widget
# uncomment the above line if you want dynamic control of the figure when using VSCode
fig, (ax1, ax2) = plt.subplots(1, 2,  # 1 row, 2 columns
                               figsize=(8,4)  # width, height, in inches
                               )
# left panel
ax1.plot(df['average'], color="tab:blue")
ax1.plot(df.loc['2010-01-01':'2011-12-31','average'], color="tab:orange")
ax1.set(xlabel="date",
       ylabel=r"CO$_2$ concentration (ppm)",
       title="long term");
# right panel
ax2.plot(df.loc['2010-01-01':'2011-12-31','average'], color="tab:orange")
ax2.set(xlabel="date",
        ylim=[385, 400],  # choose y limits
        yticks=np.arange(385, 401, 5),  # choose ticks
        title="years 2010--2011");
# title above both panels
fig.suptitle("Mauna Loa Observatory", y=1.00)

locator = mdates.AutoDateLocator(minticks=3, maxticks=5)
formatter = mdates.ConciseDateFormatter(locator)
ax1.xaxis.set_major_locator(locator)
ax1.xaxis.set_major_formatter(formatter)

locator = mdates.AutoDateLocator(minticks=5, maxticks=8)
formatter = mdates.ConciseDateFormatter(locator)
ax2.xaxis.set_major_locator(locator)
ax2.xaxis.set_major_formatter(formatter)

ax1.annotate(
    "2010/11",
    xy=('2010-12-25', 395),  xycoords='data',
    xytext=(-100, 40), textcoords='offset points',
    bbox=dict(boxstyle="round4,pad=.5", fc="white"),
    arrowprops=dict(arrowstyle="->",
```

```
                         color="black",
                         connectionstyle="angle,angleA=0,angleB=-90,rad=40"))
```

```
Text(-100, 40, '2010/11')
```



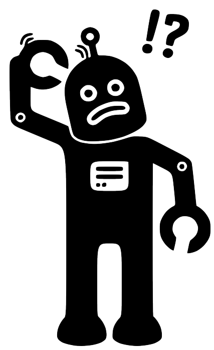
The main changes were:

1. Using the Seaborn package, we changed the fontsize and the overall plot style. Read more.
   `sns.set(style="darkgrid")`
   `sns.set_context("notebook")`
2. We changed the colors of the lineplots. To know what colors exist, click here.
3. The arrow annotation has a different style. Read more.

3.8 playing with the code

I encourage you to play with the code you just ran. An easy way of learning what each line does is to comment something out and see what changes in the output you see. If you feel brave, try to modify the code a little bit.

4 AI policy



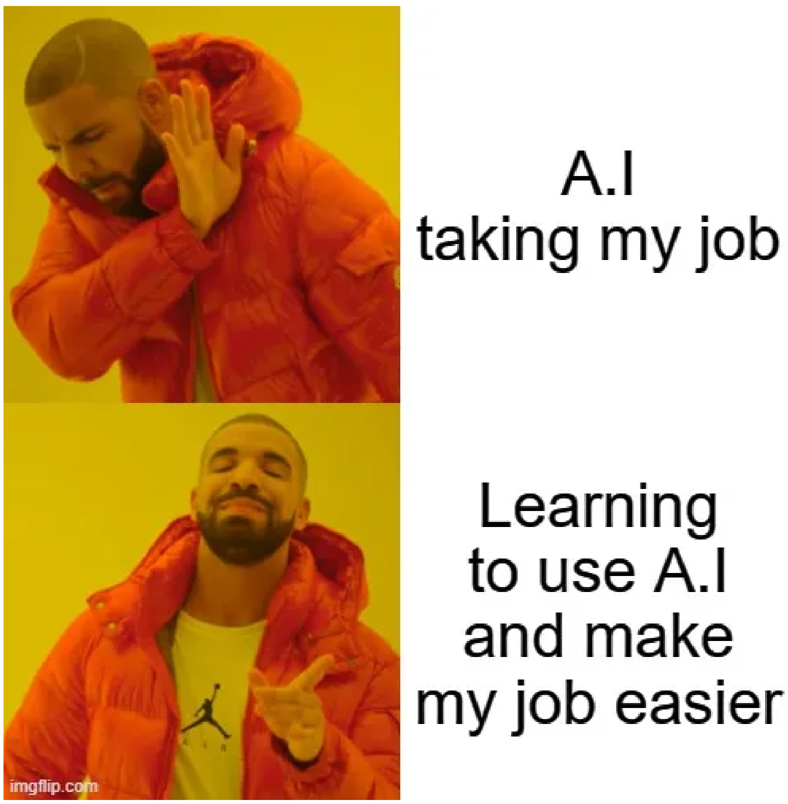
The guidelines below are an adaptation of Ethan Mollick's extremely useful ideas on AI as an assistant tool for teaching.

I EXPECT YOU to use LLMs (large language models) such as ChatGPT, Bing AI, Google Bard, or whatever else springs up since the time of this writing. You should familiarize yourself with the AI's capabilities and limitations.

Use LLMs to help you learn, chat with them about what you want to accomplish and learn from them how to do it. **Ask** your LLM what each part of the code means, copy and pasting blindly is unacceptable. You are here to learn.

Consider the following important points:

- Ultimately, you, the student, are responsible for the assignment.
- Acknowledge the use of AI in your assignment. Be transparent about your use of the tool and the extent of assistance it provided.


A.I
taking my job
Learning
to use A.I
and make
my job easier
imgflip.com

Part II

resampling

5 motivation

5.1 Jerusalem, 2019

Data from the Israel Meteorological Service, IMS.

See the temperature at a weather station in Jerusalem, for the whole 2019 year. This is an interactive graph: to zoom in, play with the bottom panel.

```
alt.VConcatChart(...)
```

5.1.0.0.1 * discussion

The temperature fluctuates on various time scales, from daily to yearly. Let's think together a few questions we'd like to ask about the data above.

Now let's see precipitation data:

```
alt.VConcatChart(...)
```

5.1.0.0.2 * discussion

What would be interesting to know about precipitation?

We have not talked about what kind of data we have in our hands here. The csv file provided by the IMS looks like this:

	Station	Date & Time (Winter)	Diffused radiation (W/m^2)	Global radiation (W/m^
0	Jerusalem Givat Ram	01/01/2019 00:00	0.0	0.0
1	Jerusalem Givat Ram	01/01/2019 00:10	0.0	0.0
2	Jerusalem Givat Ram	01/01/2019 00:20	0.0	0.0
3	Jerusalem Givat Ram	01/01/2019 00:30	0.0	0.0

	Station	Date & Time (Winter)	Diffused radiation (W/m^2)	Global radiation (W/m^
4	Jerusalem Givat Ram	01/01/2019 00:40	0.0	0.0
...	...	...	...	...
52549	Jerusalem Givat Ram	31/12/2019 22:20	0.0	0.0
52550	Jerusalem Givat Ram	31/12/2019 22:30	0.0	0.0
52551	Jerusalem Givat Ram	31/12/2019 22:40	0.0	0.0
52552	Jerusalem Givat Ram	31/12/2019 22:50	0.0	0.0
52553	Jerusalem Givat Ram	31/12/2019 23:00	0.0	0.0

We see that we have data points spaced out evenly every 10 minutes.

5.2 Challenges

Let's try to answer the following questions:

What is the mean temperature for each month?

First we have to divide temperature data by month, and then take the average for each month.
a possible solution

```
df_month = df['temperature'].resample('M').mean()
```

For each month, what is the mean of the daily maximum temperature? What about the minimun?

This is a bit trickier.

1. We need to find the maximum/minimum temperature for each day.
2. Only then we split the daily data by month and take the average.

a possible solution

```
df_day['max temp'] = df['temperature'].resample('D').max()
df_month['max temp'] = df_day['max temp'].resample('MS').mean()
```

What is the average night temperature for every season? What about the day temperature?

1. We need to filter our data to contain only night times.
2. We need to divide rain data by seasons (3 months), and then take the mean for each season.

a possible solution

```
# filter only night data
df_night = df.loc[((df.index.hour < 6) | (df.index.hour >= 18))]
season_average_night_temp = df_night['temperature'].resample('Q').mean()
```

another possible solution

```
# filter using between_time
df_night = df.between_time('18:00', '06:00', inclusive='left')
season_average_night_temp = df_night['temperature'].resample('Q').mean()
```

What is the daily precipitation?

First we have to divide rain data by day, and then take the sum for each day.
a possible solution

```
daily_precipitation = df['rain'].resample('D').sum()
```

How much rain was there every month?

We have to divide rain data by month, and then sum the totals of each month.
a possible solution

```
monthly_precipitation = df['rain'].resample('M').sum()
```

How many rainy days were there each month?

1. We need to sum rain by day.
2. We need to count how many days are there each month where `rain > 0`.

a possible solution

```
daily_precipitation = df['rain'].resample('D').sum()
only_rainy_days = daily_precipitation.loc[daily_precipitation > 0]
rain_days_per_month = only_rainy_days.resample('M').count()
```

How many days, hours, and minutes were between the last rain of the season (Malkosh) to the first (Yoreh)?

1. We need to divide our data into two: `rainy_season_1` and `rainy_season_2`.
2. We need to find the time of the last rain in `rainy_season_1`.
3. We need to find the time of the first rain in `rainy_season_2`.
4. We need to compute the time difference between the two dates.

a possible solution

```
split_date = '2019-08-01'
rainy_season_1 = df[:split_date]  # everything before split date
rainy_season_2 = df[split_date:]  # everything after split date
malkosh = rainy_season_1['rain'].loc[rainy_season_1['rain'] > 0].last_valid_index()
yoreh = rainy_season_2['rain'].loc[rainy_season_2['rain'] > 0].first_valid_index()
dry_period = yoreh - malkosh
# extracting days, hours, and minutes
days = dry_period.days
hours = dry_period.components.hours
minutes = dry_period.components.minutes
print(f'The dry period of 2019 was {days} days, {hours} hours and {minutes} minutes.')
```

What was the rainiest morning (6am-12pm) of the year? Bonus, what about the rainiest night (6pm-6am)?

1. We need to filter our data to contain only morning times.
2. We need to sum rain by day.
3. We need to find the day with the maximum value.

a possible solution

```
# filter to only day data
morning_df = df.loc[((df.index.hour >= 6) & (df.index.hour < 18))]
morning_rain = morning_df['rain'].resample('D').sum()
rainiest_morning = morning_rain.idxmax()
# plot
morning_rain.plot()
plt.axvline(rainiest_morning, c='r', alpha=0.5, linestyle='--')
```

bonus solution

```
# filter to only night data
df_night = df.loc[((df.index.hour < 6) | (df.index.hour >= 18))]
# resampling night for each day is tricky because the date changes at 12:00. We can do thi
# we shift the time back by 6 hours so all the data for the same night will have the same
df_shifted = df_night.tshift(-6, freq='H')
night_rain = df_shifted['rain'].resample('D').sum()
rainiest_night = night_rain.idxmax()
# plot
night_rain.plot()
plt.axvline(rainiest_night, c='r', alpha=0.5, linestyle='--')
```

Note: this whole webpage is actually a Jupyter Notebook rendered as html. If you want to know how to make interactive graphs, go to the top of the page and click on “ Code”

Useful functions compatible with `pandas.resample()` can be found here. The full list of resampling frequencies can be found here.

6 resampling

We can only really understand how to calculate monthly means if we do it ourselves.

First, let's import a bunch of packages we need to use.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import warnings
# Suppress FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
```

Now we load the csv file for Jerusalem (2019), provided by the IMS.

6.0.0.0.1 * discussion

We will go to the IMS website together and see what are the options available and how to download. If you just need the csv right away, download it here.

- We substitute every occurence of `-` for NaN (not a number, that is, the data is missing).
- We call the columns `Temperature (°C)` and `Rainfall (mm)` with more convenient names, since we will be using them a lot.
- We interpret the column `Date & Time (Winter)` as a date, saying to python that day comes first.

- We make `date` the index of the dataframe.

```
filename = "../archive/data/jerusalem2019.csv"
df = pd.read_csv(filename, na_values=['-'])
df.rename(columns={'Temperature (°C)': 'temperature',
                   'Rainfall (mm)': 'rain'}, inplace=True)
df['date'] = pd.to_datetime(df['Date & Time (Winter)'], dayfirst=True)
df = df.set_index('date')
df
```

	Station	Date & Time (Winter)	Diffused radiation (W/m^2)	Global rad
date				
2019-01-01 00:00:00	Jerusalem Givat Ram	01/01/2019 00:00	0.0	0.0
2019-01-01 00:10:00	Jerusalem Givat Ram	01/01/2019 00:10	0.0	0.0
2019-01-01 00:20:00	Jerusalem Givat Ram	01/01/2019 00:20	0.0	0.0
2019-01-01 00:30:00	Jerusalem Givat Ram	01/01/2019 00:30	0.0	0.0
2019-01-01 00:40:00	Jerusalem Givat Ram	01/01/2019 00:40	0.0	0.0
…	…	…	…	…
2019-12-31 22:20:00	Jerusalem Givat Ram	31/12/2019 22:20	0.0	0.0
2019-12-31 22:30:00	Jerusalem Givat Ram	31/12/2019 22:30	0.0	0.0
2019-12-31 22:40:00	Jerusalem Givat Ram	31/12/2019 22:40	0.0	0.0
2019-12-31 22:50:00	Jerusalem Givat Ram	31/12/2019 22:50	0.0	0.0
2019-12-31 23:00:00	Jerusalem Givat Ram	31/12/2019 23:00	0.0	0.0

With `resample` it's easy to compute monthly averages. Resample by itself only divides the data into buckets (in this case monthly buckets), and waits for a further instruction. Here, the next instruction is `mean`.

```
df_month = df['temperature'].resample('M').mean()
df_month
```

```
date
2019-01-31     9.119937
2019-02-28     9.629812
2019-03-31    10.731571
2019-04-30    14.514329
2019-05-31    22.916894
```

```
2019-06-30    23.587361
2019-07-31    24.019403
2019-08-31    24.050822
2019-09-30    22.313287
2019-10-31    20.641868
2019-11-30    17.257153
2019-12-31    11.224131
Freq: M, Name: temperature, dtype: float64
```

Instead of `M` for month, which other options do I have? The full list can be found here, but the most commonly used are:

```
M         month end frequency
MS        month start frequency
A         year end frequency
AS, YS    year start frequency
D         calendar day frequency
H         hourly frequency
T, min    minutely frequency
S         secondly frequency
```

The results we got for the monthly means were given as a pandas series, not dataframe. Let's correct this:

```
df_month = (df['temperature'].resample('M')          # resample by month
                             .mean()                 # take the mean
                             .to_frame('mean temp') # make output a dafaframe
           )
df_month
```

	mean temp
date	
2019-01-31	9.119937
2019-02-28	9.629812
2019-03-31	10.731571
2019-04-30	14.514329
2019-05-31	22.916894
2019-06-30	23.587361

date	mean temp
2019-07-31	24.019403
2019-08-31	24.050822
2019-09-30	22.313287
2019-10-31	20.641868
2019-11-30	17.257153
2019-12-31	11.224131

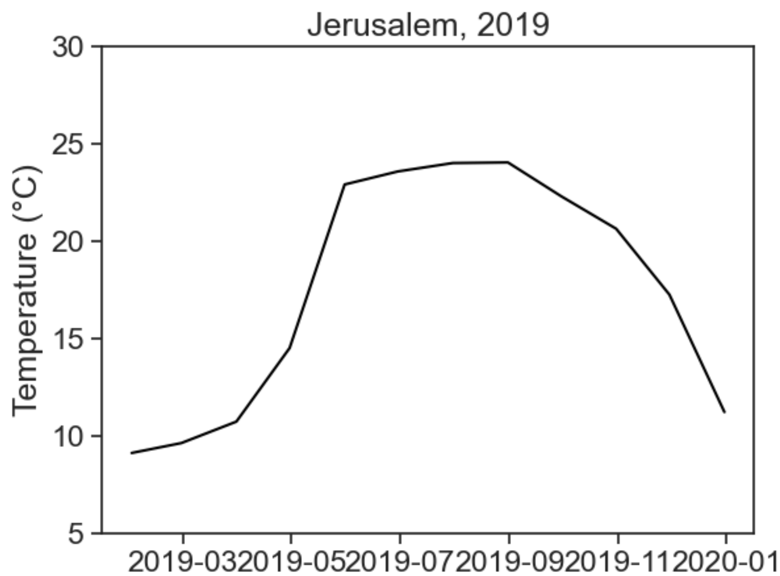
6.0.0.0.2 * hot tip

Sometimes, a line of code can get too long and messy. In the code above, we broke line for every step, which makes the process so much cleaner. We **highly** advise you to do the same. **Attention:** This trick works as long as all the elements are inside the same parenthesis.

Now it's time to plot!

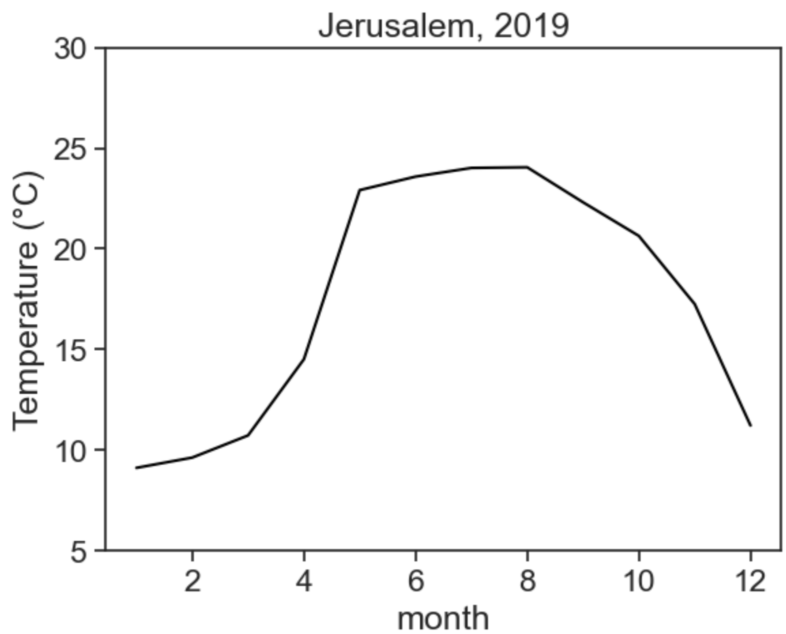
```
fig, ax = plt.subplots()
ax.plot(df_month['mean temp'], color='black')
ax.set(ylabel='Temperature (°C)',
       yticks=np.arange(5,35,5),
       title="Jerusalem, 2019")
```

```
[Text(0, 0.5, 'Temperature (°C)'),
 [<matplotlib.axis.YTick at 0x7faf784c6d60>,
  <matplotlib.axis.YTick at 0x7faf7843a220>,
  <matplotlib.axis.YTick at 0x7faf784c62b0>,
  <matplotlib.axis.YTick at 0x7faf784f3400>,
  <matplotlib.axis.YTick at 0x7faf784f3760>,
  <matplotlib.axis.YTick at 0x7faf784fa5b0>],
 Text(0.5, 1.0, 'Jerusalem, 2019')]
```



The dates in the horizontal axis are not great. An easy fix is to use the month numbers instead of dates.

```
fig, ax = plt.subplots()
ax.plot(df_month.index.month, df_month['mean temp'], color='black')
ax.set(xlabel="month",
       ylabel='Temperature (°C)',
       yticks=np.arange(5,35,5),
       title="Jerusalem, 2019",);
```



6.0.0.0.3 * discussion

When you have datetime as the dataframe index, you don't need to give the function `plot` two arguments, date and values. You can just tell `plot` to use the column you want, the function will take the dates by itself.

What does this line mean?
`df_month['mean temp'].index.month`

Print on the screen the following, and see yourself what each thing is:

- `df_month`
- `df_month.index`
- `df_month.index.month`
- `df_month.index.day`

We're done! Congratulations :)

Now we need to calculate the average minimum/maximum daily temperatures. We start by creating an empty dataframe.

```
df_day = pd.DataFrame()
```

Now resample data by day (`D`), and take the min/max of each day.

```
df_day['min temp'] = df['temperature'].resample('D').min()
df_day['max temp'] = df['temperature'].resample('D').max()
df_day
```

	min temp	max temp
date		
2019-01-01	7.5	14.1
2019-01-02	6.6	11.5
2019-01-03	6.3	10.7
2019-01-04	6.6	14.6
2019-01-05	7.0	11.4
...	...	...
2019-12-27	4.4	7.4
2019-12-28	6.6	10.3
2019-12-29	8.1	12.5
2019-12-30	6.9	13.0
2019-12-31	5.2	13.3

The next step is to calculate the average minimum/maximum for each month. This is similar to what we did above.

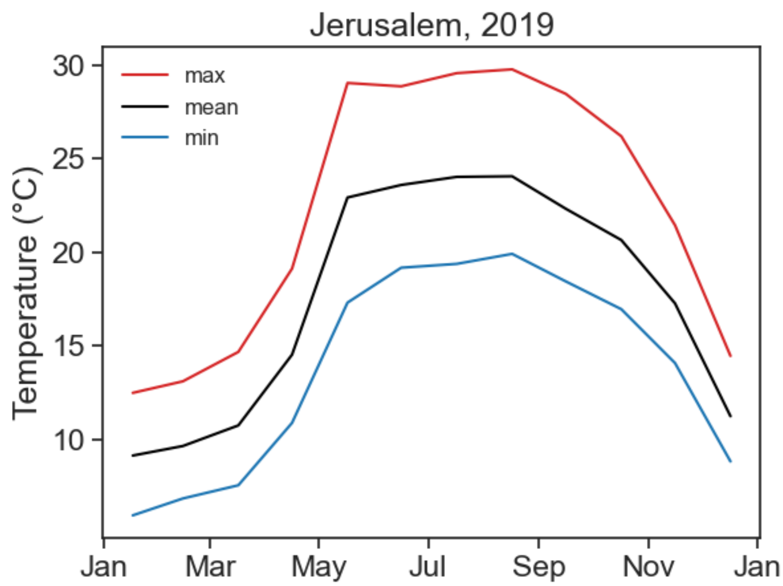
```
df_month['min temp'] = df_day['min temp'].resample('M').mean()
df_month['max temp'] = df_day['max temp'].resample('M').mean()
df_month
```

	mean temp	min temp	max temp
date			
2019-01-31	9.119937	5.922581	12.470968
2019-02-28	9.629812	6.825000	13.089286
2019-03-31	10.731571	7.532258	14.661290
2019-04-30	14.514329	10.866667	19.113333
2019-05-31	22.916894	17.296774	29.038710

date	mean temp	min temp	max temp
2019-06-30	23.587361	19.163333	28.860000
2019-07-31	24.019403	19.367742	29.564516
2019-08-31	24.050822	19.903226	29.767742
2019-09-30	22.313287	18.430000	28.456667
2019-10-31	20.641868	16.945161	26.190323
2019-11-30	17.257153	14.066667	21.436667
2019-12-31	11.224131	8.806452	14.448387

Let's plot…

```
fig, ax = plt.subplots()
ax.plot(df_month['max temp'], color='tab:red', label='max')
ax.plot(df_month['mean temp'], color='black', label='mean')
ax.plot(df_month['min temp'], color='tab:blue', label='min')
ax.set(ylabel='Temperature (°C)',
       yticks=np.arange(10,35,5),
       title="Jerusalem, 2019")
ax.xaxis.set_major_locator(mdates.MonthLocator(range(1, 13, 2), bymonthday=15))
date_form = DateFormatter("%b")
ax.xaxis.set_major_formatter(date_form)
ax.legend(fontsize=12, frameon=False);
```



Voilà! You made a beautiful graph!

6.0.0.0.4 * discussion

This time we did not put month numbers in the horizontal axis, we now have month names. How did we do this black magic, you ask? See lines 8–10 above. Matplotlib gives you absolute power over what to put in the axis, if you can only know how to tell it to… Wanna know more? Click here.

7 upsampling

In the previous chapter, we resampled from fine temporal resolution to a coarser one. This is also called **downsampling**. We will learn the **upsampling** now: how to go from coarse data to a finer scale.

Sadly, there is no free lunch, and we just can't get data that was not measured. What to do then?

It's best to consider a practical example.

7.1 Potential Evapotranspiration using Penman's equation

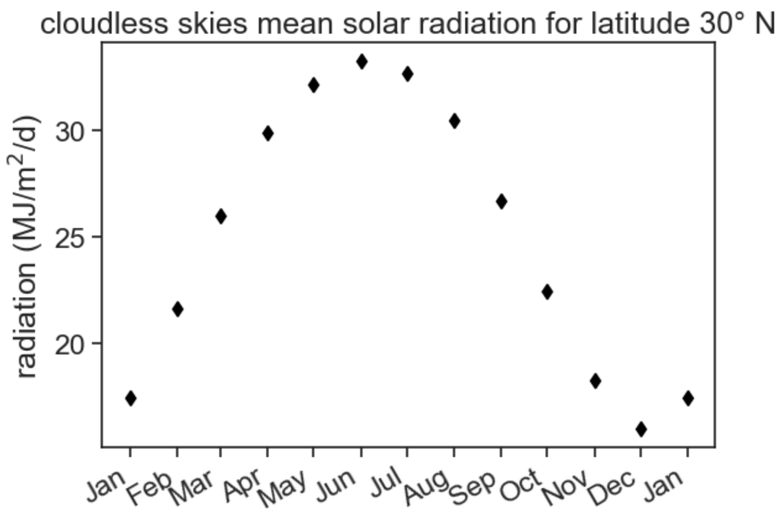
We want to calculate the daily potential evapotranspiration using Penman's equation. Part of the calculation involves characterizing the energy budget on soil surface. When direct solar radiation measurements are not available, we can estimate the energy balance by knowing the "cloudless skies mean solar radiation", R_{so}. This is the amount of energy (MJ/m^2/d) that hits the surface, assuming no clouds. This radiation depends on the season and on the latitude you are. For Israel, located at latitude 32° N, we can use the following data for 30°:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
```

```python
dates = pd.date_range(start='2021-01-01', periods=13, freq='MS')
values = [17.46, 21.65, 25.96, 29.85, 32.11, 33.20, 32.66, 30.44, 26.67, 22.48, 18.30, 16.04
df = pd.DataFrame({'date': dates, 'radiation': values})
df = df.set_index('date')
df
```

date	radiation
2021-01-01	17.46
2021-02-01	21.65
2021-03-01	25.96
2021-04-01	29.85
2021-05-01	32.11
2021-06-01	33.20
2021-07-01	32.66
2021-08-01	30.44
2021-09-01	26.67
2021-10-01	22.48
2021-11-01	18.30
2021-12-01	16.04
2022-01-01	17.46

```python
fig, ax = plt.subplots()
ax.plot(df['radiation'], color='black', marker='d', linestyle='None')
ax.set(ylabel=r'radiation (MJ/m$^2$/d)',
       title="cloudless skies mean solar radiation for latitude 30° N")
ax.xaxis.set_major_locator(mdates.MonthLocator())
date_form = DateFormatter("%b")
ax.xaxis.set_major_formatter(date_form)
plt.gcf().autofmt_xdate()  # makes slanted dates
```



We only have 12 values for the whole year, and we can't use this dataframe to compute daily ET. We need to upsample!

In the example below, we resample the monthly data into daily data, and do nothing else. Pandas doesn't know what to do with the new points, so it fills them with NaN.

```
df_nan = df['radiation'].resample('D').asfreq().to_frame()
df_nan.head(33)
```

	radiation
date	
2021-01-01	17.46
2021-01-02	NaN
2021-01-03	NaN
2021-01-04	NaN
2021-01-05	NaN
2021-01-06	NaN
2021-01-07	NaN
2021-01-08	NaN
2021-01-09	NaN
2021-01-10	NaN
2021-01-11	NaN
2021-01-12	NaN

	radiation
date	
2021-01-13	NaN
2021-01-14	NaN
2021-01-15	NaN
2021-01-16	NaN
2021-01-17	NaN
2021-01-18	NaN
2021-01-19	NaN
2021-01-20	NaN
2021-01-21	NaN
2021-01-22	NaN
2021-01-23	NaN
2021-01-24	NaN
2021-01-25	NaN
2021-01-26	NaN
2021-01-27	NaN
2021-01-28	NaN
2021-01-29	NaN
2021-01-30	NaN
2021-01-31	NaN
2021-02-01	21.65
2021-02-02	NaN

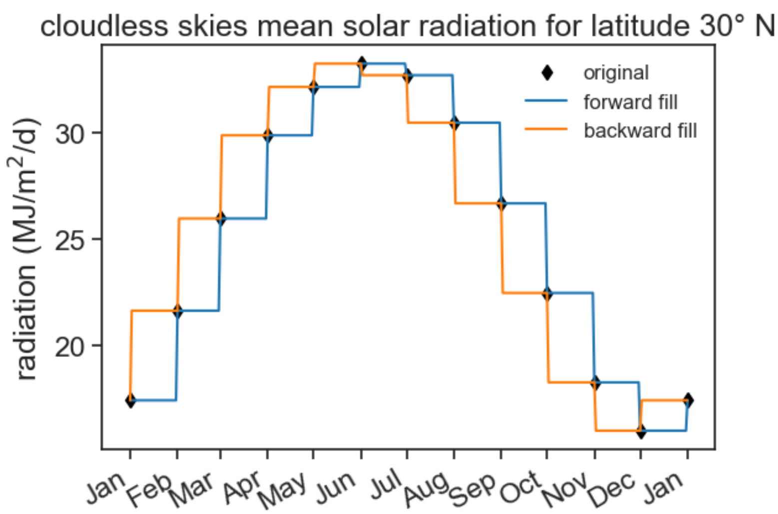
7.2 Forward/Backward fill

We can forward/backward fill these NaNs:

```
df_forw = df['radiation'].resample('D').ffill().to_frame()
df_back = df['radiation'].resample('D').bfill().to_frame()
```

```
fig, ax = plt.subplots()
ax.plot(df['radiation'], color='black', marker='d', linestyle='None', label="original")
ax.plot(df_forw['radiation'], color='tab:blue', label="forward fill")
ax.plot(df_back['radiation'], color='tab:orange', label="backward fill")
ax.set(ylabel=r'radiation (MJ/m$^2$/d)',
       title="cloudless skies mean solar radiation for latitude 30° N")
```

```
ax.legend(frameon=False, fontsize=12)
ax.xaxis.set_major_locator(mdates.MonthLocator())
date_form = DateFormatter("%b")
ax.xaxis.set_major_formatter(date_form)
plt.gcf().autofmt_xdate()  # makes slanted dates
```



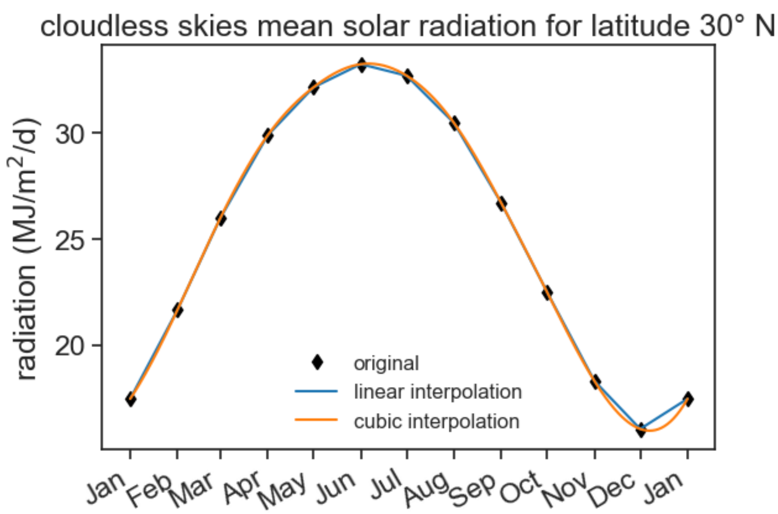
This does the job, but I want something better, not step functions. The radiation should vary smoothly from day to day. Let's use interpolation.

7.3 Interpolation

```
df_linear = df['radiation'].resample('D').interpolate(method='time').to_frame()
df_cubic = df['radiation'].resample('D').interpolate(method='cubic').to_frame()
```

```
fig, ax = plt.subplots()
ax.plot(df['radiation'], color='black', marker='d', linestyle='None', label="original")
ax.plot(df_linear['radiation'], color='tab:blue', label="linear interpolation")
ax.plot(df_cubic['radiation'], color='tab:orange', label="cubic interpolation")
ax.set(ylabel=r'radiation (MJ/m$^2$/d)',
       title="cloudless skies mean solar radiation for latitude 30° N")
```

```
ax.legend(frameon=False, fontsize=12)
ax.xaxis.set_major_locator(mdates.MonthLocator())
date_form = DateFormatter("%b")
ax.xaxis.set_major_formatter(date_form)
plt.gcf().autofmt_xdate()  # makes slanted dates
```



There are many ways to fill NaNs and to interpolate. A nice detailed guide can be found here.

8 interpolation

Interpolation is the act of getting data you don't have from data you alreay have. We used some interpolation when upsampling, and now it is time to talk about it a little bit more in depth.

There is no one correct way of interpolating, the method you use depends in the end on what you want to accomplish, what are your (hidden or explicit) assumptions, etc. Let's see a few examples.

9 FAQ

9.1 How to resample by year, but have it end in September?

This is called anchored offset. One possible use to it is to calculate statistics according to the hydrological year that, for example, ends in September.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
```

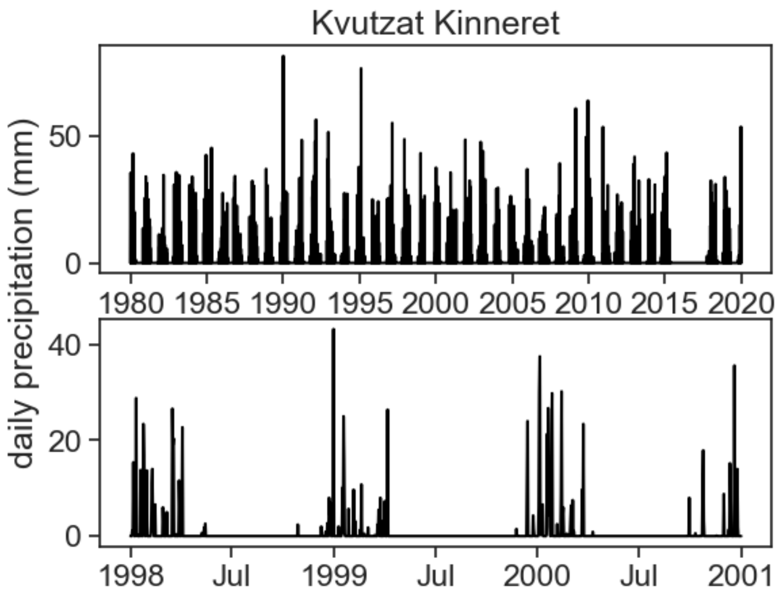
```
filename = "../archive/data/Kinneret_Kvuza_daily_rainfall.csv"
df = pd.read_csv(filename, na_values=['-'])
df.rename(columns={'Date': 'date',
                   'Daily Rainfall (mm)': 'rain'}, inplace=True)
df['date'] = pd.to_datetime(df['date'], dayfirst=True)
df = df.set_index('date')
df = df.resample('D').asfreq().fillna(0)  # asfreq = replace
df
```

	Station	rain
date		
1980-01-02	Kinneret Kvuza 09/1977-08/2023	0.0
1980-01-03	0	0.0
1980-01-04	0	0.0

date	Station	rain
1980-01-05	Kinneret Kvuza 09/1977-08/2023	35.5
1980-01-06	Kinneret Kvuza 09/1977-08/2023	2.2
...	...	...
2019-12-26	Kinneret Kvuza 09/1977-08/2023	39.4
2019-12-27	Kinneret Kvuza 09/1977-08/2023	5.2
2019-12-28	Kinneret Kvuza 09/1977-08/2023	1.6
2019-12-29	0	0.0
2019-12-30	Kinneret Kvuza 09/1977-08/2023	0.1

```python
fig, ax = plt.subplots(2,1)
ax[0].plot(df['rain'], color='black')
ax[1].plot(df.loc['1998':'2000', 'rain'], color='black')
locator = mdates.AutoDateLocator(minticks=4, maxticks=8)
formatter = mdates.ConciseDateFormatter(locator)
ax[1].xaxis.set_major_locator(locator)
ax[1].xaxis.set_major_formatter(formatter)
fig.text(0.02, 0.5, 'daily precipitation (mm)', va='center', rotation='vertical')
ax[0].set_title("Kvutzat Kinneret")
```

```
Text(0.5, 1.0, 'Kvutzat Kinneret')
```



We see a marked dry season during the summer, so let's assume the Hydrological Year ends in September.

```
df_year = df.resample('A-SEP').sum()
df_year = df_year.loc['1980':'2003']
df_year
```

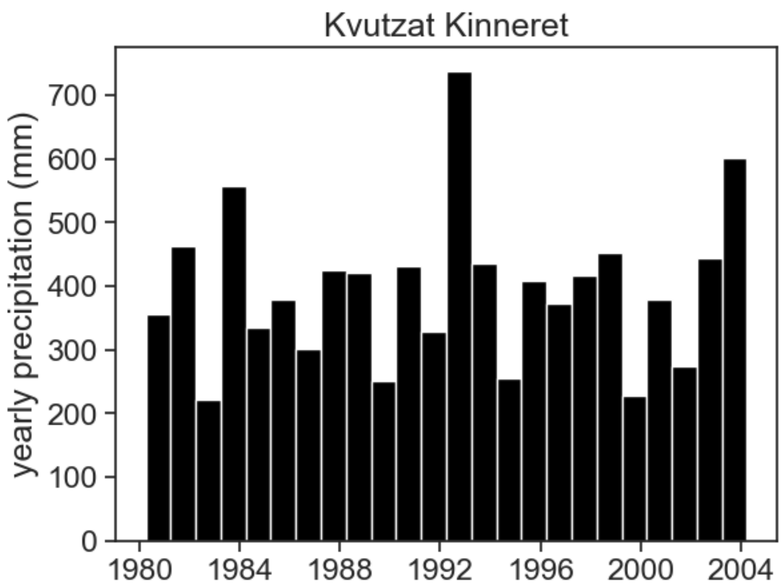
```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_94063/2047090134.py:1: FutureWarning
  df_year = df.resample('A-SEP').sum()
```

	rain
date	
1980-09-30	355.5
1981-09-30	463.1
1982-09-30	221.7
1983-09-30	557.1
1984-09-30	335.3
1985-09-30	379.8
1986-09-30	300.7
1987-09-30	424.7

date	rain
1988-09-30	421.6
1989-09-30	251.6
1990-09-30	432.5
1991-09-30	328.3
1992-09-30	738.4
1993-09-30	434.9
1994-09-30	255.4
1995-09-30	408.6
1996-09-30	373.0
1997-09-30	416.2
1998-09-30	451.9
1999-09-30	227.8
2000-09-30	378.9
2001-09-30	273.9
2002-09-30	445.2
2003-09-30	602.4

```python
fig, ax = plt.subplots()
ax.bar(df_year.index, df_year['rain'], color='black',
       width=365)
ax.set_ylabel("yearly precipitation (mm)")
ax.set_title("Kvutzat Kinneret")
```

```
Text(0.5, 1.0, 'Kvutzat Kinneret')
```



9.2 When upsampling, how to fill missing values with zero?

We did that in the example above, like this:

```
df = df.resample('D').asfreq().fillna(0)  # asfreq = replace
```

Part III

smoothing

10 motivation

This is the temperature for the Yatir Forest (Shani station, see map), between 2 and 5 of January 2022. Data is in intervals of 10 minutes, and was downloaded from the Israel Meteorological Service.

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import datetime as dt
import matplotlib.ticker as ticker
import warnings
# Suppress FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
import requests
import json
import os
# %matplotlib widget
```

```python
# read token from file
with open('../archive/IMS-token.txt', 'r') as file:
    TOKEN = file.readline()
# 28 = SHANI station
STATION_NUM = 28
start = "2022/01/01"
end = "2022/01/07"
filename = 'shani_2022_january.json'

# check if the JSON file already exists
```

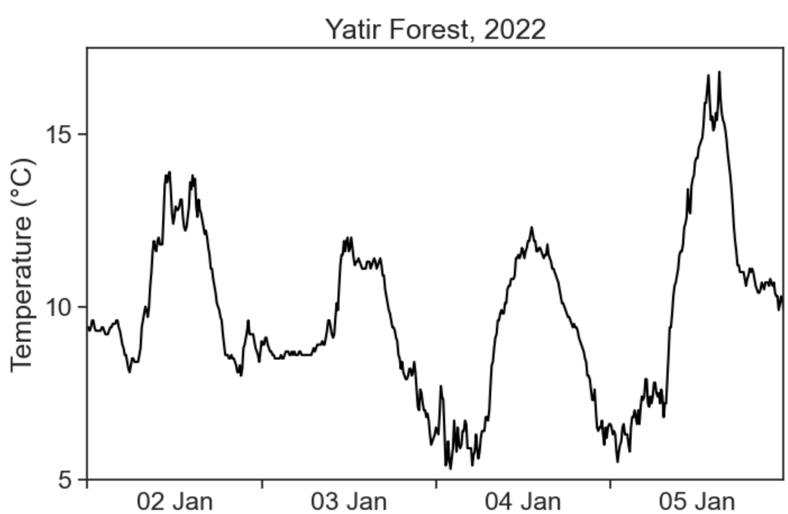
```python
# if so, then load file
if os.path.exists(filename):
    with open(filename, 'r') as json_file:
        data = json.load(json_file)
else:
    # make the API request if the file doesn't exist
    url = f"https://api.ims.gov.il/v1/envista/stations/{STATION_NUM}/data/?from={start}&to={
    headers = {'Authorization': f'ApiToken {TOKEN}'}
    response = requests.get(url, headers=headers)
    data = json.loads(response.text.encode('utf8'))

    # save the JSON data to a file
    with open(filename, 'w') as json_file:
        json.dump(data, json_file)
# show data to see if it's alright
# data
```

```python
df = pd.json_normalize(data['data'],record_path=['channels'], meta=['datetime'])
df['date'] = (pd.to_datetime(df['datetime'])
                .dt.tz_localize(None)  # ignores time zone information
             )
df = df.pivot(index='date', columns='name', values='value')
# df
```

```python
# dirty trick to have dates in the middle of the 24-hour period
# make minor ticks in the middle, put the labels there!
# from https://matplotlib.org/stable/gallery/ticks/centered_ticklabels.html

def centered_dates(ax):
    date_form = DateFormatter("%d %b")  # %d 3-letter-Month
    # major ticks at midnight, every day
    ax.xaxis.set_major_locator(mdates.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(date_form)
    # minor ticks at noon, every day
    ax.xaxis.set_minor_locator(mdates.HourLocator(byhour=[12]))
    # erase major tick labels
    ax.xaxis.set_major_formatter(ticker.NullFormatter())
    # set minor tick labels as define above
    ax.xaxis.set_minor_formatter(date_form)
```

```
    # completely erase minor ticks, center tick labels
    for tick in ax.xaxis.get_minor_ticks():
        tick.tick1line.set_markersize(0)
        tick.tick2line.set_markersize(0)
        tick.label1.set_horizontalalignment('center')
```

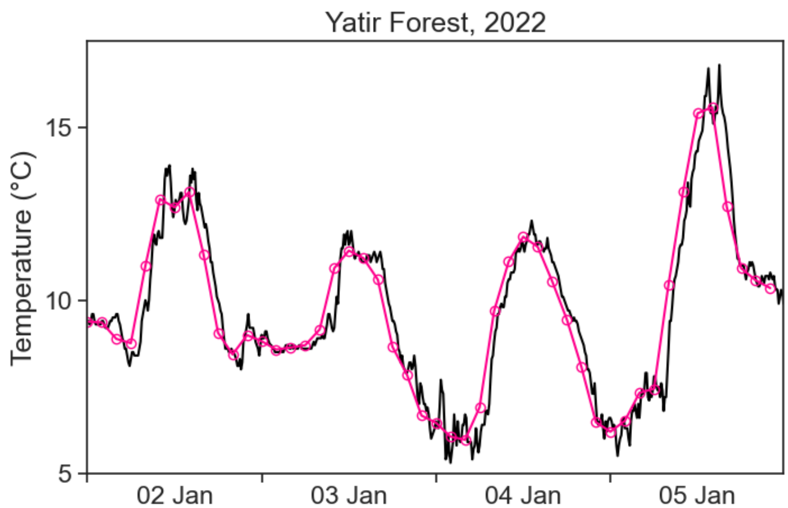
```
fig, ax = plt.subplots(figsize=(8,5))
start = "2022-01-02"
end = "2022-01-05"
df = df.loc[start:end]
ax.plot(df['TD'], color='black')
ax.set(ylim=[5, 17.5],
       xlim=[df.index[0], df.index[-1]],
       ylabel="Temperature (°C)",
       title="Yatir Forest, 2022",
       yticks=[5,10,15])
centered_dates(ax)
fig.savefig("YF-temperature_2022_jan.png", dpi=300)
```



We see that the temperature curve has a rough profile. Can we find ways of getting smoother curves?

We learned how to average over a window with `resample`. Let's try that for a 2-hour window:

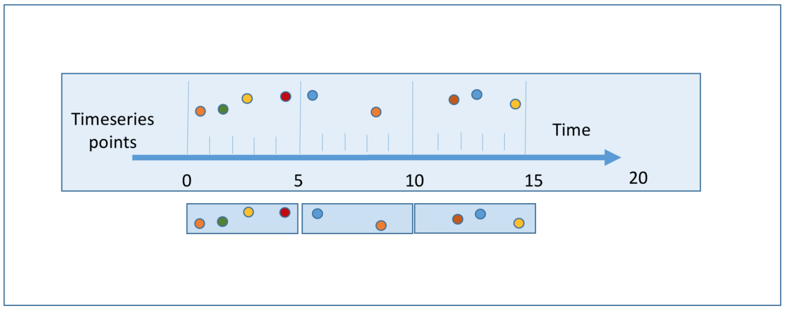
```
fig, ax = plt.subplots(figsize=(8,5))
ax.plot(df['TD'], color='black')
ax.plot(df['TD'].resample('2H').mean(),
        color='xkcd:hot pink', ls='-',
        marker="o", mfc="None")
ax.set(ylim=[5, 17.5],
       xlim=[df.index[0], df.index[-1]],
       ylabel="Temperature (°C)",
       title="Yatir Forest, 2022",
       yticks=[5,10,15])
centered_dates(ax)
```

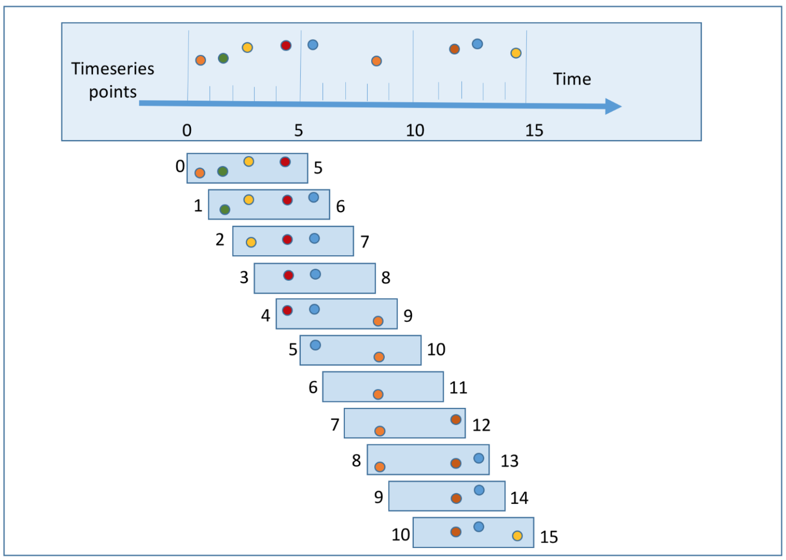


The temperature profile now is much smoother, but when using `resample`, we lost temporal resolution. Our original data had 10-minute frequency, and now we have a 2-hour frequency.

How can we get a smoother curve without losing resolution?

10.1 Tumbling vs Sliding





11 sliding window

This is the temperature for the Yatir Forest, between 2 and 5 of January 2022. Data (download .csv here) is in intervals of 10 minutes, and was downloaded from the Israel Meteorological Service.

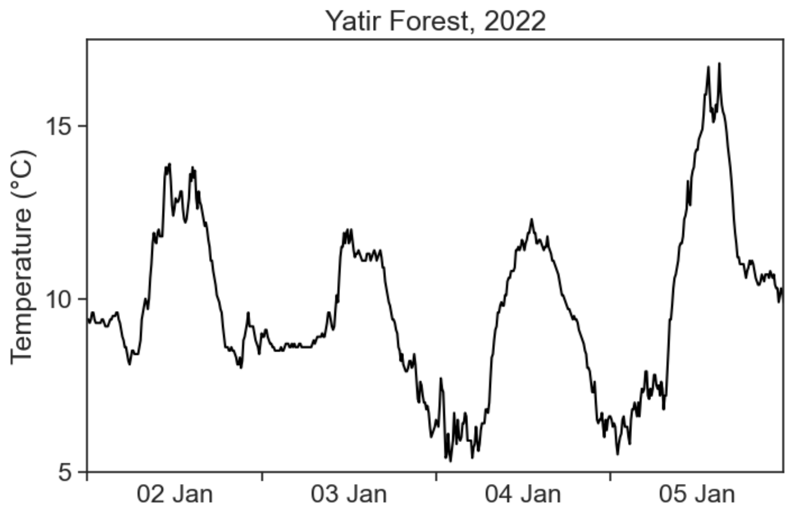
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import datetime as dt
import matplotlib.ticker as ticker
import os
import warnings
import scipy
# Suppress FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
# %matplotlib widget
```

```
# dirty trick to have dates in the middle of the 24-hour period
# make minor ticks in the middle, put the labels there!
# from https://matplotlib.org/stable/gallery/ticks/centered_ticklabels.html

def centered_dates(ax):
    date_form = DateFormatter("%d %b")  # %d 3-letter-Month
    # major ticks at midnight, every day
    ax.xaxis.set_major_locator(mdates.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(date_form)
    # minor ticks at noon, every day
    ax.xaxis.set_minor_locator(mdates.HourLocator(byhour=[12]))
```

```python
    # erase major tick labels
    ax.xaxis.set_major_formatter(ticker.NullFormatter())
    # set minor tick labels as define above
    ax.xaxis.set_minor_formatter(date_form)
    # completely erase minor ticks, center tick labels
    for tick in ax.xaxis.get_minor_ticks():
        tick.tick1line.set_markersize(0)
        tick.tick2line.set_markersize(0)
        tick.label1.set_horizontalalignment('center')
```

```python
df = pd.read_csv('shani_2022_january.csv', parse_dates=['date'], index_col='date')
```

```python
fig, ax = plt.subplots(figsize=(8,5))
start = "2022-01-02"
end = "2022-01-05"
df = df.loc[start:end]
ax.plot(df['TD'], color='black')

plot_settings = {
    'ylim': [5, 17.5],
    'xlim': [df.index[0], df.index[-1]],
    'ylabel': "Temperature (°C)",
    'title': "Yatir Forest, 2022",
    'yticks': [5, 10, 15]
}

ax.set(**plot_settings)
centered_dates(ax)
```



We see that the temperature curve has a rough profile. Can we find ways of getting smoother curves?

11.1 convolution

Convolution is a fancy word for averaging a time series using a sliding window. We will use the terms **convolution, running average, and rolling average** interchangeably. See the animation below. We take all temperature values inside a window of width 500 minutes (51 points), and average them with equal weights. The weights profile is called `kernel`.

The pink curve is much smoother than the original! However, the running average cannot describe sharp temperature changes. If we decrease the window width to 200 minutes (21 points), we get the following result.

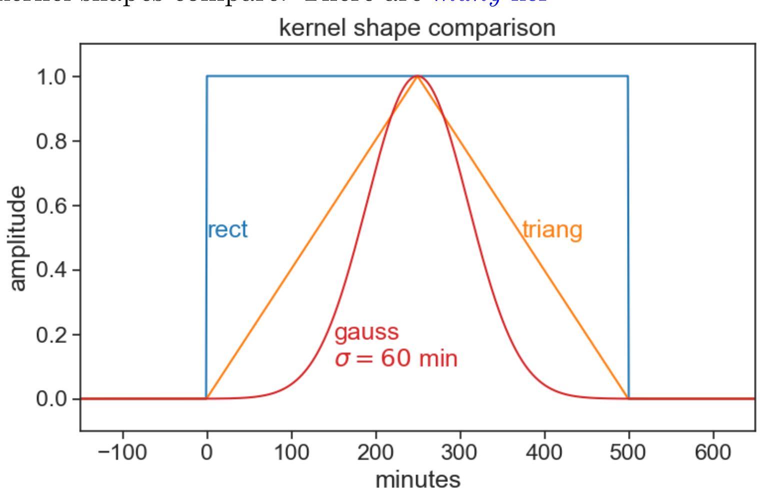
There is a tradeoff between the smoothness of a curve, and its ability to describe sharp temporal changes.

11.2 kernels

We can modify our running average, so that values closer to the center of the window have higher weights, and those further away count less. This is achieved by changing the weight profile, or the shape of the kernel. We see below the result of a running average using a triangular window of base 500 minutes (51 points).

Things can get as fancy as we want. Instead of a triangular kernel, which has sharp edges, we can choose a smoother gaussian kernel, see the difference below. We used a gaussian kernel with 60-minute standard deviation.

See how the three kernel shapes compare. There are *many* kernels to chose from.



11.3 math

The definition of a convolution between signal $f(t)$ and kernel $k(t)$ is

$$(f * k)(t) = \int f(\tau)k(t-\tau)d\tau.$$

The expression $f * k$ denotes the convolution of these two functions. The argument of k is $t-\tau$, meaning that the kernel runs from left to right (as t does), and at every point the two signals (f and k) are multiplied together. It is the product of the signal with the weight function k that gives us an average. Because of $-\tau$, the kernel is flipped backwards, but this has no effect to symmetric kernels, like to ones in the examples above. Finally, the actual running average is not the convolution, but

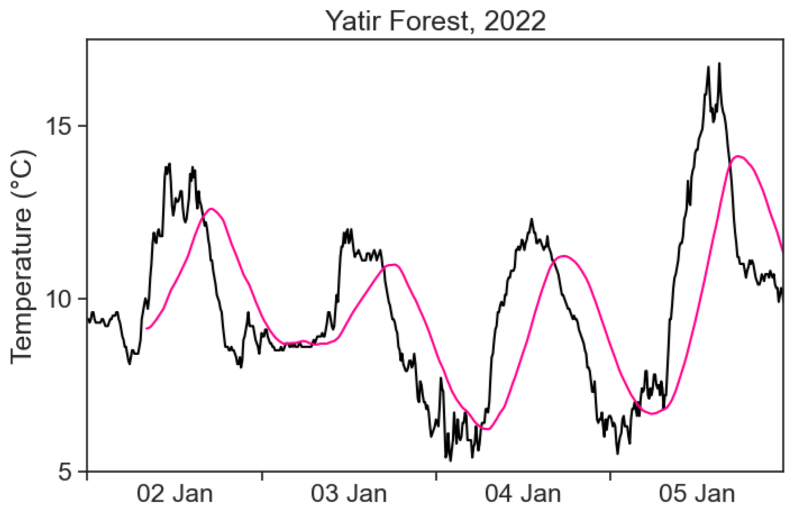
$$\frac{(f*k)(t)}{\displaystyle\int k(t)dt}.$$

Whenever the integral of the kernel is 1, then the convolution will be identical with the running average.

11.4 numerics

Running averages are very common tools in time-series analysis. The `pandas` package makes life quite simple. For example, in order to calculate the running average of temperature using a rectangular kernel, one writes:

```
df['temp_smoothed'] = (
                       df['TD'].rolling(window='500min',
                                        min_periods=50   # comment this to see what happens
                                        )
                              .mean()
                      )

fig, ax = plt.subplots(figsize=(8,5))
ax.plot(df['TD'], color='black')
ax.plot(df['temp_smoothed'], color='xkcd:hot pink')
ax.set(**plot_settings)
centered_dates(ax)
```



The pink curve looks smooth, but why does it lag behind the data?! What's going on?

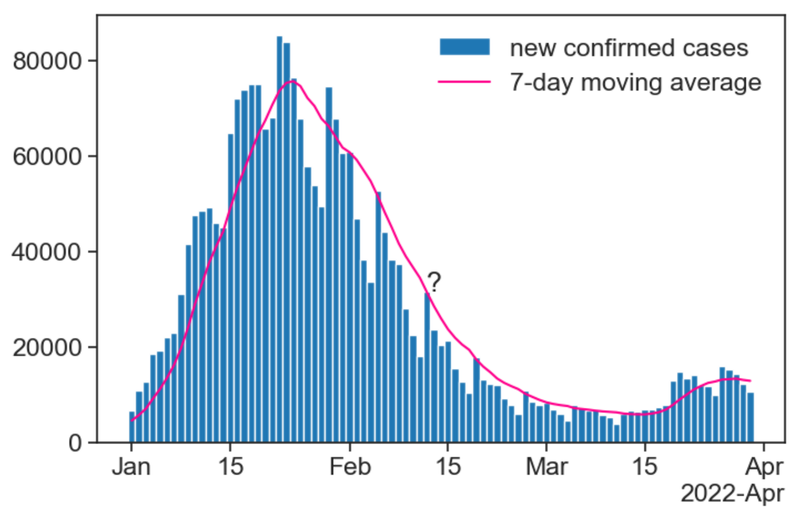
11.4.1 7-day average of COVID-19 infections

During the COVID-19 pandemic, we would see graphs like this all the time in the news:

```
# data from https://health.google.com/covid-19/open-data/raw-data?loc=IL
# define the local file path
local_file_path = 'COVID_19_israel.csv'
# check if the local file exists
if os.path.exists(local_file_path):
    # if the local file exists, load it
    covid_IL = pd.read_csv(local_file_path, parse_dates=['date'], index_col='date')
else:
    # if the local file doesn't exist, download from the URL
    url = "https://storage.googleapis.com/covid19-open-data/v3/location/IL.csv"
    covid_IL = pd.read_csv(url, parse_dates=['date'], index_col='date')
    # save the downloaded data to the local file for future use
    covid_IL.to_csv(local_file_path)

df_covid = covid_IL['new_confirmed'].to_frame()
```

```
df_covid['7d_avg'] = df_covid['new_confirmed'].rolling(window='7D').mean()
```

```
fig, ax = plt.subplots(figsize=(8,5))
st = '2022-01-01'
en = '2022-03-30'
new_cases = ax.bar(df_covid[st:en].index, df_covid.loc[st:en,'new_confirmed'],
       color="tab:blue", width=1)
mov_avg, = ax.plot(df_covid.loc[st:en,'7d_avg'],
        color='xkcd:hot pink')
ax.legend(handles=[new_cases, mov_avg],
          labels=['new confirmed cases', '7-day moving average'],
          frameon=False)
weird_day = "2022-02-12"
weird_day_x = mdates.date2num(dt.datetime.strptime(weird_day, "%Y-%m-%d"))
ax.text(weird_day_x, df_covid.loc[weird_day,'new_confirmed'], "?")
# formating dates on x axis
locator = mdates.AutoDateLocator(minticks=7, maxticks=11)
formatter = mdates.ConciseDateFormatter(locator)
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)
```



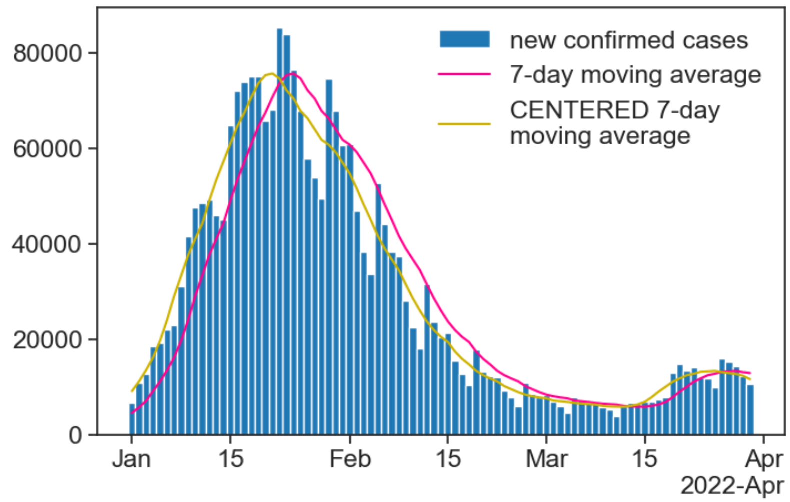
Take a look at the moving average next to the question mark. How can it be that high, when all the bars around that date are lower? Is the calculation right?

The answer is that the result of the moving average is assigned to the right-most date in the running window. This is reasonable for COVID-19 cases: for a given day, I can only calculate a 7-day average based on **past** values, I don't know what the future will be.

There is a simple way of assigning the result to the center of the window:

```
df_covid['7d_avg_center'] = (
                             df_covid['new_confirmed']
                                 .rolling(window='7D',
                                          center=True)  # THIS
                                 .mean()
                            )
```

```
fig, ax = plt.subplots(figsize=(8,5))
st = '2022-01-01'
en = '2022-03-30'
new_cases = ax.bar(df_covid[st:en].index, df_covid.loc[st:en,'new_confirmed'],
       color="tab:blue", width=1)
mov_avg, = ax.plot(df_covid.loc[st:en,'7d_avg'],
        color='xkcd:hot pink')
mov_avg_center, = ax.plot(df_covid.loc[st:en,'7d_avg_center'],
                          color='xkcd:mustard')
ax.legend(handles=[new_cases, mov_avg, mov_avg_center],
          labels=['new confirmed cases',
                  '7-day moving average',
                  'CENTERED 7-day\nmoving average'],
          frameon=False)
# formating dates on x axis
locator = mdates.AutoDateLocator(minticks=7, maxticks=11)
formatter = mdates.ConciseDateFormatter(locator)
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)
```



As a rule, we will used a **centered** moving average (`center=True`), unless stated otherwise. Also, only use `min_periods` if you know what you are doing.

11.4.2 gaussian

You can easily change the kernel shape by using the `win_type` argument. See how to perform a rolling mean with a gaussian kernel:

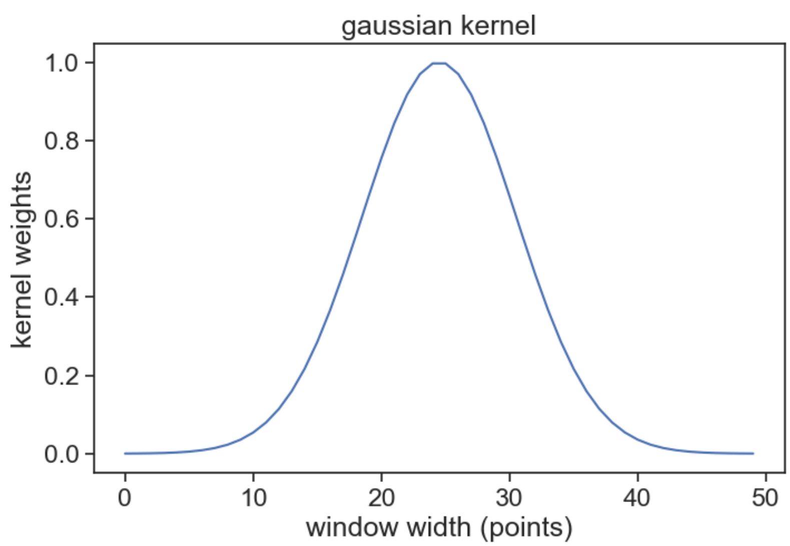
```
(
df['temperature'].rolling(window=window_width,
                          center=True,
                          win_type="gaussian")
                 .mean(std=std_gaussian)
)
```

where

- `window_width` is an integer, number of points in your window
- `std_gaussian` is the standard deviation of your gaussian, **measured in sample points, not time!**

For instance, if we have measurements every 10 minutes, and our window width is 500 minutes, then `window_width = 500/10 + 1` (first and last included). If we want a standard deviation of 60 minutes, then `std_gaussian = 6`. The gaussian kernel will look like this:

```
window_width = 50  # in points = 500 min
std = 6  # in points = 60 min
fig, ax = plt.subplots(figsize=(8,5))
g = scipy.signal.gaussian(window_width, std)
ax.plot(g)
ax.set(xlabel="window width (points)",
       ylabel="kernel weights",
       title="gaussian kernel");
```



You can take a look at various options for kernel shapes here, provided by the `scipy` package.

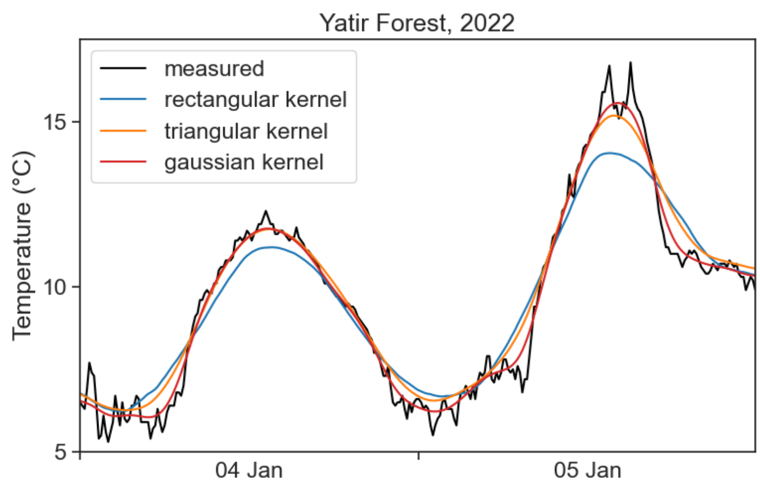
11.4.3 triangular

Same idea as gaussian, but simpler, because we don't need to think about standard deviation.

```
(
df['temperature'].rolling(window=window_width,
                          center=True,
                          win_type="triang")
                  .mean()
)
```

11.5 which window shape and width to choose?

Sorry, there is not definite answer here… It really depends on your data and what you need to do with it. See below a comparison of all examples in the videos above.



One important question you need to ask is: what are the time scales associated with the processes I'm interested in? For example, if I'm interested in the daily temperature pattern, getting rid of 1-minute-long fluctuations would probably be ok. On the other hand, if we were to smooth the signal so much that all that can be seen are the temperature changes between summer and winter, then my smoothing got out of hand, and I threw away the very process I wanted to study.

All this is to say that you need to know in advance a few things about the system you are studying, otherwise you can't know what is "noise" that can be smoothed away.

12 not only averages

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import datetime as dt
import matplotlib.ticker as ticker
import warnings
# Suppress FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
import requests
import json
import os
# %matplotlib widget
```

```python
# read token from file
with open('../archive/IMS-token.txt', 'r') as file:
    TOKEN = file.readline()
# 28 = SHANI station
STATION_NUM = 28
start = "2022/01/01"
end = "2022/01/07"
filename = 'shani_2022_january.json'

# check if the JSON file already exists
# if so, then load file
if os.path.exists(filename):
    with open(filename, 'r') as json_file:
        data = json.load(json_file)
```

```
else:
    # make the API request if the file doesn't exist
    url = f"https://api.ims.gov.il/v1/envista/stations/{STATION_NUM}/data/?from={start}&to={
    headers = {'Authorization': f'ApiToken {TOKEN}'}
    response = requests.get(url, headers=headers)
    data = json.loads(response.text.encode('utf8'))

    # save the JSON data to a file
    with open(filename, 'w') as json_file:
        json.dump(data, json_file)
# show data to see if it's alright
# data
```

```
df = pd.json_normalize(data['data'],record_path=['channels'], meta=['datetime'])
df['date'] = (pd.to_datetime(df['datetime'])
                .dt.tz_localize(None)  # ignores time zone information
             )
df = df.pivot(index='date', columns='name', values='value')
# let's work only with a few days, and only temperature
start = "2022-01-02"
end = "2022-01-05"
df = df.loc[start:end, 'TD'].to_frame()
df.rename(columns={"TD": "temp"}, inplace=True)
# df
```

```
# dirty trick to have dates in the middle of the 24-hour period
# make minor ticks in the middle, put the labels there!
# from https://matplotlib.org/stable/gallery/ticks/centered_ticklabels.html

def centered_dates(ax):
    date_form = DateFormatter("%d %b")  # %d 3-letter-Month
    # major ticks at midnight, every day
    ax.xaxis.set_major_locator(mdates.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(date_form)
    # minor ticks at noon, every day
    ax.xaxis.set_minor_locator(mdates.HourLocator(byhour=[12]))
    # erase major tick labels
    ax.xaxis.set_major_formatter(ticker.NullFormatter())
    # set minor tick labels as define above
```

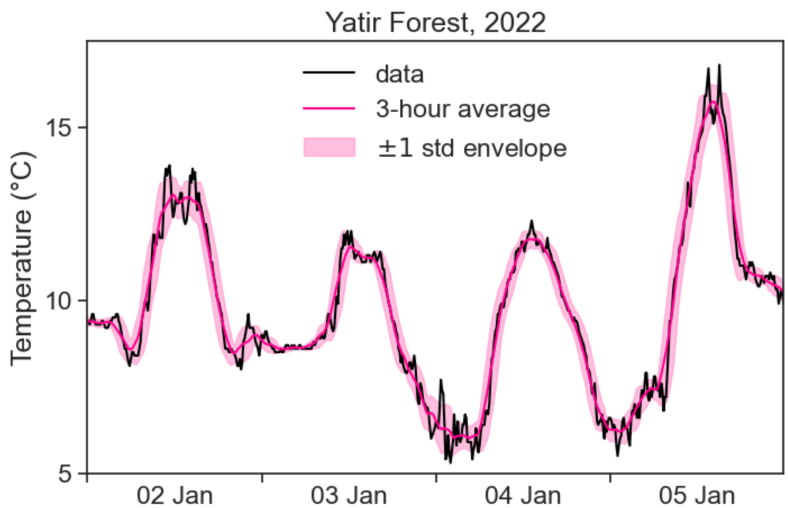
```
    ax.xaxis.set_minor_formatter(date_form)
    # completely erase minor ticks, center tick labels
    for tick in ax.xaxis.get_minor_ticks():
        tick.tick1line.set_markersize(0)
        tick.tick2line.set_markersize(0)
        tick.label1.set_horizontalalignment('center')

# creating the dictionary with the desired settings
plot_settings = {
    'ylim': [5, 17.5],
    'xlim': [df.index[0], df.index[-1]],
    'ylabel': 'Temperature (°C)',
    'title': 'Yatir Forest, 2022',
    'yticks': [5, 10, 15]
}
```

Let's see on a graph the average temperature, with an envelope of 1 standard deviation around it:

```
df['mean'] = df['temp'].rolling('3H', center=True).mean()
df['std'] = df['temp'].rolling('3H', center=True).std()
```

```
fig, ax = plt.subplots(figsize=(8,5))


plot_std = ax.fill_between(df.index,
                           df['mean'] + df['std'],
                           df['mean'] - df['std'],
                           color="xkcd:pink", alpha=0.5)
plot_data, = ax.plot(df['temp'], color='black')
plot_mean, =ax.plot(df['mean'], color='xkcd:hot pink')

ax.legend([plot_data, plot_mean, plot_std],
          ['data', '3-hour average', r"$\pm1$ std envelope"],
          frameon=False)

# applying the settings to the ax object
ax.set(**plot_settings)
centered_dates(ax)
```

```
# fig.savefig("YF-temperature_2022_jan.png", dpi=300)
```



12.1 Confidence Interval

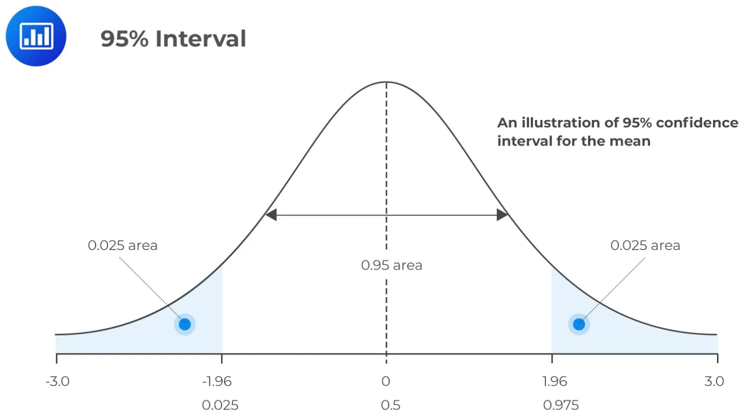
We can calculate anything we want inside the sliding window. One good example is the **Confidence Interval of the Mean**, given by:

$$CI(\alpha) = Z(\alpha) \cdot SE.$$

$Z(\alpha)$ is the Z-score corresponding to the chosen confidence level α. The most commonly used confidence level is 95%, which corresponds to a Z-score of 1.96. What does this mean? This means that we expect to find 95% of the points within ± 1.96 standard deviations away from the mean.

This is called “ ” - in hebrew.

- $Z(\alpha)$ = Z-score.
- SE = standard error.



Source: Dhaval Raval's Medium article

You can find the Z-score using the following python code:

```
from scipy.stats import norm

confidence_level = 0.95
# 5% outside
out = 1 - confidence_level
# 0.975 of points to the left of right boundary
p = 1 - out/2
# inverse of cdf: 0.975 of the points will be smaller than what distance (in sigma units)?
z_score = norm.ppf(p)
print(f"z-score = {z_score}")
```

```
z-score = 1.959963984540054
```

If you are still not convinced why we need 0.975 instead of 0.95, read this excellent response on stackoverflow.

SE is the standard error:

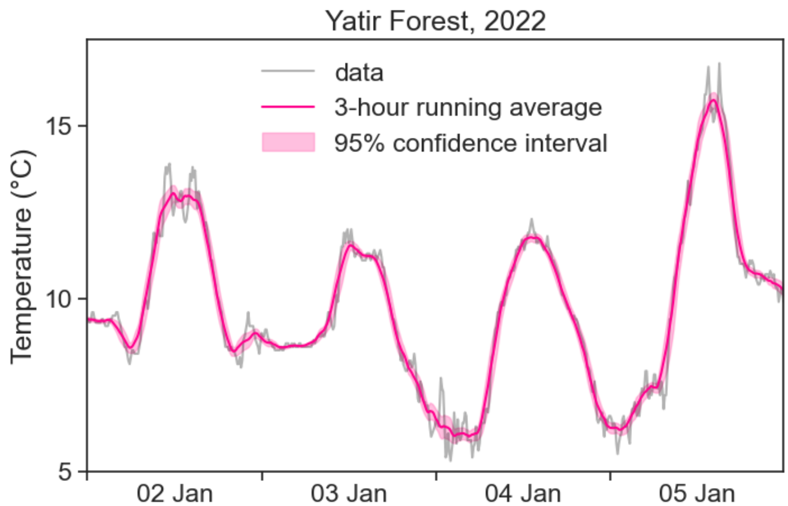
$$SE = \frac{\sigma}{\sqrt{N}}.$$

- σ = standard deviation.
- N = number of points.

We can write a function to calculate the confidence interval of the mean, and use it with the sliding window:

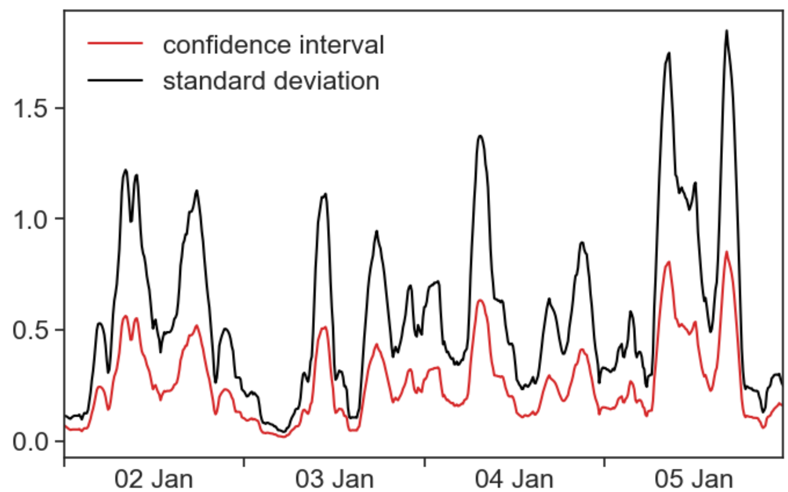
```python
def std_error_of_the_mean(window):
    return window.std() / np.sqrt(window.count())

def confidence_interval(window):
    return z_score * std_error_of_the_mean(window)

df['std_error'] = (
                   df['temp'].rolling('3H',
                                      center=True)
                             .apply(std_error_of_the_mean)
                  )
df['confidence_int'] = (
                        df['temp'].rolling('3H',
                                           center=True)
                                  .apply(confidence_interval)
                       )
```

```python
fig, ax = plt.subplots(figsize=(8,5))

plot_std = ax.fill_between(df.index,
                           df['mean'] + df['confidence_int'],
                           df['mean'] - df['confidence_int'],
                           color="xkcd:pink", alpha=0.5)
plot_data, = ax.plot(df['temp'], color='black', alpha=0.3)
plot_mean, =ax.plot(df['mean'], color='xkcd:hot pink')

ax.legend([plot_data, plot_mean, plot_std],
          ['data', '3-hour running average', r"95% confidence interval"],
          frameon=False)

# applying the settings to the ax object
ax.set(**plot_settings)
centered_dates(ax)
# fig.savefig("YF-temperature_2022_jan.png", dpi=300)
```



When the time series has a regular sampling frequency, all positions of the running window will have the same number of data points in them. Because the Confidence Interval is proportional to the Standard Error, and the SE is proportional to the Standard Deviation ($\sqrt{N}$ is constant), then the envelope created by the CI is identical to the envelope created by the standard deviation, up to a multiplying constant. Nice.

```
fig, ax = plt.subplots(figsize=(8,5))
plot_ci, = ax.plot(df['confidence_int'], color='tab:red')
plot_std, = ax.plot(df['std'], color="black")
ax.legend([plot_ci, plot_std],
          ['confidence interval', 'standard deviation'],
          frameon=False)

# applying the settings to the ax object
# ax.set(**plot_settings)
ax.set(xlim=[df.index[0], df.index[-1]])
centered_dates(ax)
```


confidence interval
standard deviation
1.5
1.0
0.5
0.0
02 Jan
03 Jan
04 Jan
05 Jan

13 fit

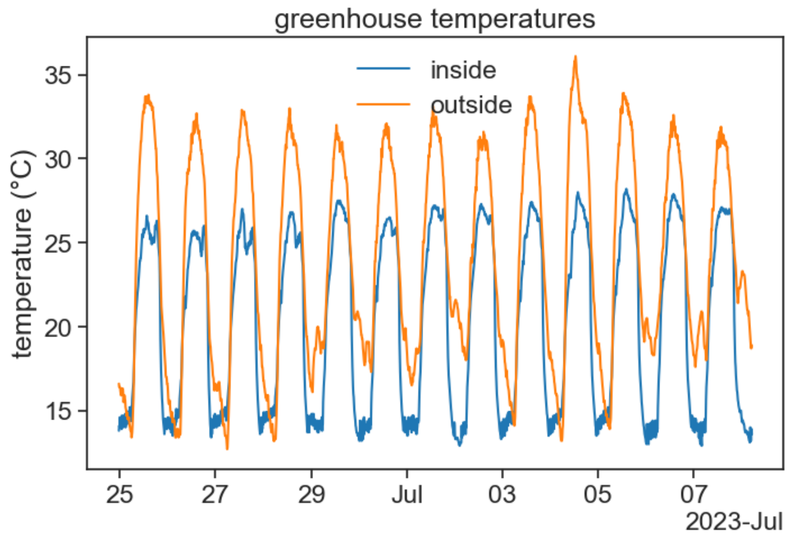
We will make a little parenthesis to talk about a very important topic: **fitting**.

See below temperature data inside and outside a greenhouse, for a period of about 2 weeks.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import altair as alt
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
from scipy.optimize import curve_fit
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
# avoid "SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a Dat
pd.options.mode.chained_assignment = None  # default='warn'
# %matplotlib widget
```

```
df = pd.read_csv('greenhouse_cooling.csv', index_col='time', parse_dates=True)
# df
```

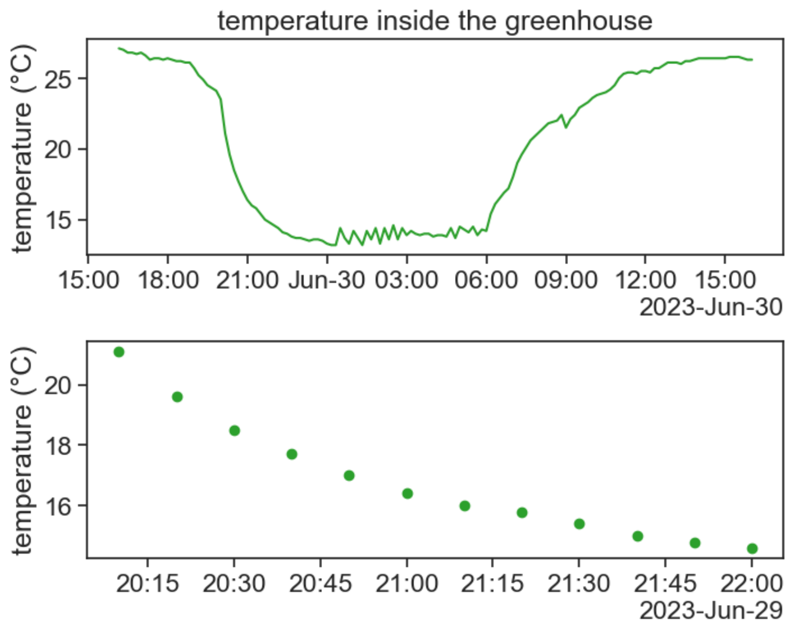
```
fig, ax = plt.subplots(figsize=(8,5))

ax.plot(df['T_in'], c='tab:blue', label='inside')
ax.plot(df['T_out'], c='tab:orange', label='outside')
ax.set(ylabel='temperature (°C)',
       title="greenhouse temperatures")

# formating dates on x axis
locator = mdates.AutoDateLocator(minticks=7, maxticks=11)
formatter = mdates.ConciseDateFormatter(locator)
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)

ax.legend(frameon=False);
```



Every evening, at 20:00, the air conditioning turns on, and we see a fast decrease in temperature:

```
df_fit = df['2023-06-29 20:10:00':'2023-06-29 22:00:00']

fig, ax = plt.subplots(2, 1, figsize=(8,6))
fig.subplots_adjust(hspace=0.4)  # Adjust the vertical space between subplots

ax[0].plot(df.loc['2023-06-29 16:10:00':'2023-06-30 16:00:00', 'T_in'], color='tab:green')
ax[0].set(ylabel='temperature (°C)',
       title="temperature inside the greenhouse")

ax[1].scatter(df_fit['T_in'].index, df_fit['T_in'], color='tab:green')
ax[1].set(ylabel='temperature (°C)',)

# formating dates on x axis
locator = mdates.AutoDateLocator(minticks=7, maxticks=11)
formatter = mdates.ConciseDateFormatter(locator)
ax[0].xaxis.set_major_locator(locator)
ax[0].xaxis.set_major_formatter(formatter)

locator = mdates.AutoDateLocator(minticks=7, maxticks=11)
formatter = mdates.ConciseDateFormatter(locator)
```

```
ax[1].xaxis.set_major_locator(locator)
ax[1].xaxis.set_major_formatter(formatter)
```



The AC is able to bring the temperature down, but up to a limit. The AC can work at a maximum given power, and the cooler it is outside, the more effectively the AC will be able to bring down the temperature inside the greenhouse. We can imagine that the AC behaves as an effective external environment to the greenhouse, and the greenhouse cools down according to Newton's law of cooling:

$$\frac{dT}{dt} = r \cdot (T_{\text{env}} - T)$$

The cooling rate is proportional to the difference in temperature between the inside and outside. Assuming T_{env} and r to be constant, the solution of this differential equation is:

$$T(t) = T_{\text{env}} + (T_0 - T_{\text{env}})\, e^{-rt}.$$

- T = the greenhouse temperature
- T_{env} = the outside environment temperature
- r = coefficient of heat transfer.

- T_0 = the initial greenhouse temperature

We want to check if the temperature measured inside the greenhouse behaves like Newton's law of cooling, and if so, what can we say about the cooling coefficient r and about T_{env}.

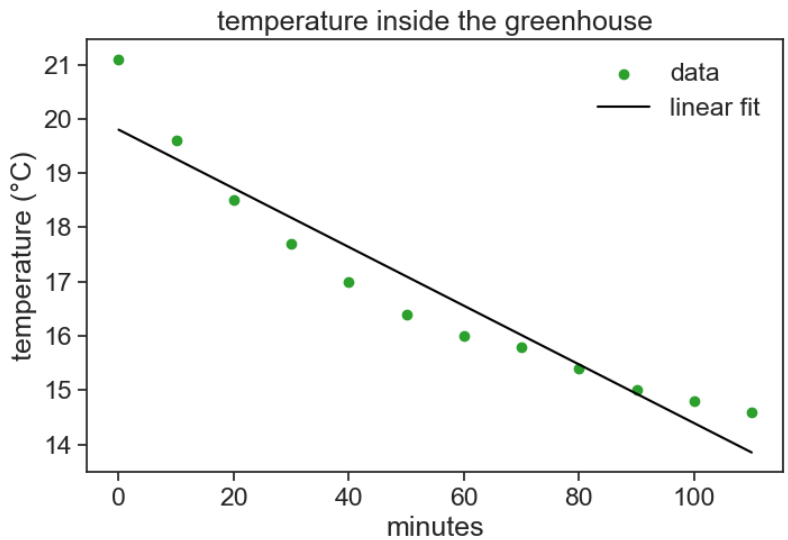
13.1 linear fit

The following is a **very short** introduction to curve fitting. The natural place to start is with a linear fit.

```
# the "fit" process can't deal with datetimes
# we therefore make a new column 'minutes', that will be used here
df_fit['minutes'] = (df_fit.index - df_fit.index[0]).total_seconds() / 60
# linear Fit (degree 1)
degree = 1
coeffs = np.polyfit(df_fit['minutes'], df_fit['T_in'], degree)
# linear Function
linear_function = np.poly1d(coeffs)
```

```
fig, ax = plt.subplots(figsize=(8,5))

ax.scatter(df_fit['minutes'], df_fit['T_in'],
           color='tab:green', label='data')
ax.plot(df_fit['minutes'], linear_function(df_fit['minutes']),
        color='black', label='linear fit')

ax.set(xlabel='minutes',
       ylabel='temperature (°C)',
       title="temperature inside the greenhouse")

ax.legend(frameon=False)
print(f"starting at {coeffs[1]:.2f} degrees,\nthe temperature decreases by {-coeffs[0]:.2f}
```

```
starting at 19.80 degrees,
the temperature decreases by 0.05 degrees every minute.
```



The line above is the "best" straight line that describes our data. Defining the residual as the difference between our data and our model (straight line),

$$e = T_{\text{data}} - T_{\text{model}},$$

the straight line above is the one that **minimizes** the sum of the squares of residuals. For this reason, the method used above to fit a curve to the data is called "least-squares method".

Can we do better than a straight line?

it minimizes the sum

$$S = \sum_i e_i^2$$

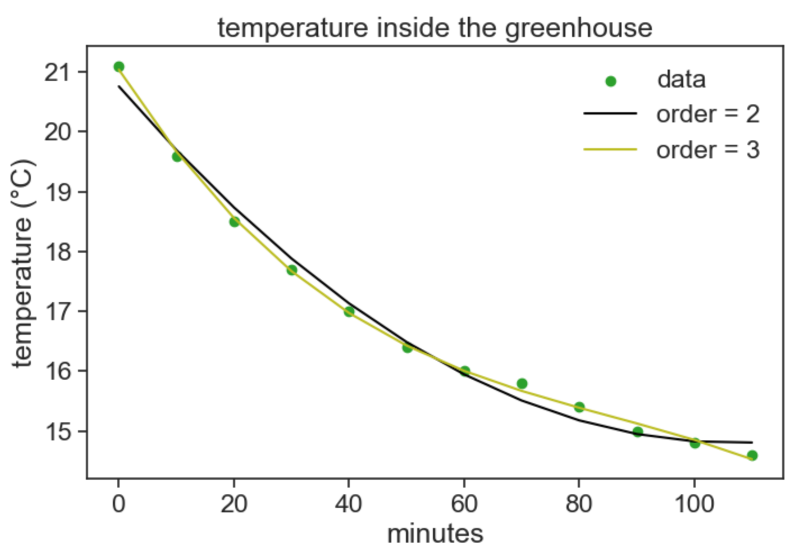
13.2 polynomial fit

```
# polynomial fit (degree 2)
degree = 2
coeffs2 = np.polyfit(df_fit['minutes'], df_fit['T_in'], degree)
quad_function = np.poly1d(coeffs2)

# polynomial fit (degree 2)
degree = 3
coeffs3 = np.polyfit(df_fit['minutes'], df_fit['T_in'], degree)
```

```
cubic_function = np.poly1d(coeffs3)
```

```
fig, ax = plt.subplots(figsize=(8,5))

ax.scatter(df_fit['minutes'], df_fit['T_in'],
           color='tab:green', label='data')
ax.plot(df_fit['minutes'], quad_function(df_fit['minutes']),
        color='black', label='order = 2')
ax.plot(df_fit['minutes'], cubic_function(df_fit['minutes']),
        color='tab:olive', label='order = 3')

ax.set(xlabel='minutes',
       ylabel='temperature (°C)',
       title="temperature inside the greenhouse")
ax.legend(frameon=False)
```

```
<matplotlib.legend.Legend at 0x7fd0a0833eb0>
```



13.3 any function you want

Now let's get back to our original assumption, that the greenhouse cools according to Newton's cooling law. We can still use the least-squares method for any function we want!

```
def cooling(t, T_env, T0, r):
    """
    t = time
    other stuff = parameters to be fitted
    """
    return T_env + (T0 - T_env)*np.exp(-r*t)
```

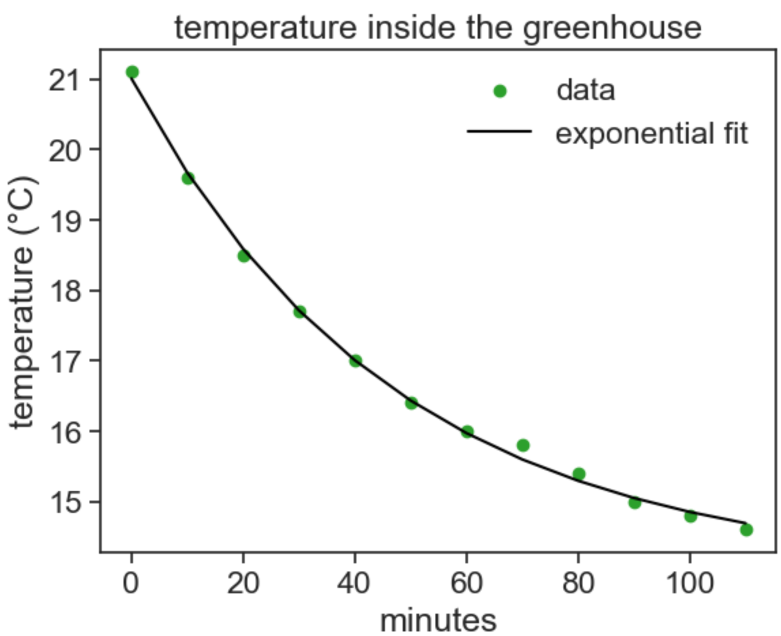
```
t = df_fit['minutes'].values
y = df_fit['T_in'].values

T_init = df_fit['T_in'][0]

popt, pcov = curve_fit(f=cooling,                # model function
                       xdata=t,                  # x data
                       ydata=y,                  # y data
                       p0=(2, T_init, 0.5),      # initial guess of the parameters
                       )
print(f"the optimal parameters are {popt}")
```

```
the optimal parameters are [14.01663586 21.0074623   0.02121802]
```

```
fig, ax = plt.subplots(sharex=True)

ax.scatter(df_fit['minutes'], df_fit['T_in'],
           color='tab:green', label='data')
ax.plot(t, cooling(t, *popt),
        color='black', label='exponential fit')

ax.set(xlabel='minutes',
       ylabel='temperature (°C)',
       title="temperature inside the greenhouse")
```

```
ax.legend(frameon=False)
```

```
<matplotlib.legend.Legend at 0x7fd0b0140850>
```



That looks really good :)

We can use curve fitting to retrieve important parameters from our data. Let's write a function that executes the fit and returns two of the fitted parameters: `T_env` and `r`.

```
def run_fit(data):
    data['minutes'] = (data.index - data.index[0]).total_seconds() / 60
    t = data['minutes'].values
    y = data['T_in'].values
    T_init = data['T_in'][0]
    popt, pcov = curve_fit(f=cooling,              # model function
                        xdata=t,               # x data
                        ydata=y,               # y data
                        p0=(2, T_init, 0.5),   # initial guess of the parameters
                        )
```

```
    return popt[0],popt[2]
```

We now apply this function to several consecutive evenings, and we keep the results in a new dataframe.

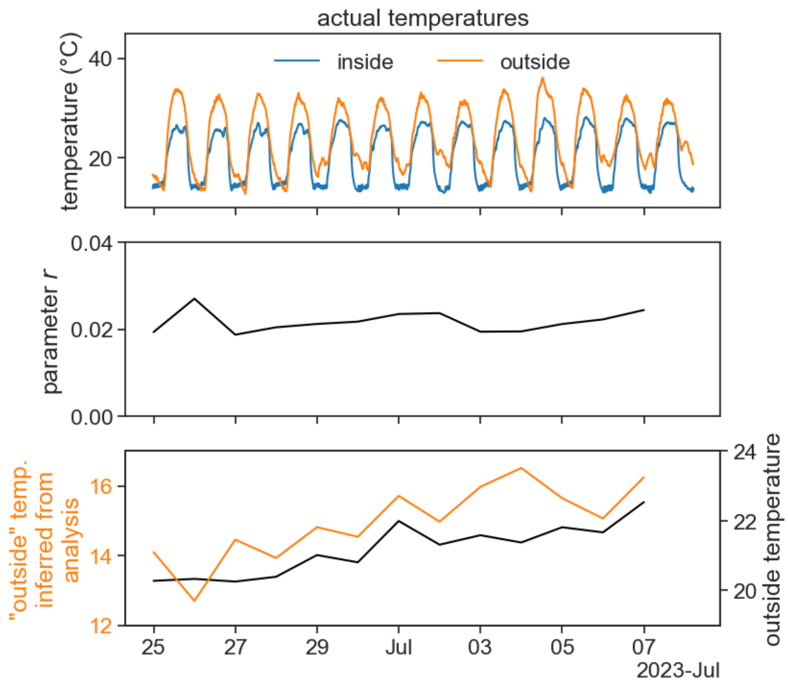
```
df_night = df.between_time('20:01', '22:01', inclusive='left')

# group by day and apply the function
# this is where the magic happens.
# if you are not familiar with "groupby", this will be hard to understand
result_series = df_night.groupby(df_night.index.date).apply(run_fit)

# convert the series to a dataframe
result_df = pd.DataFrame(result_series.tolist(), index=result_series.index, columns=['T_env'
result_df.index = pd.to_datetime(result_df.index)
result_df
```

	T_env	r
2023-06-25	13.275540	0.019354
2023-06-26	13.331949	0.027034
2023-06-27	13.254827	0.018753
2023-06-28	13.392919	0.020449
2023-06-29	14.016636	0.021218
2023-06-30	13.807517	0.021749
2023-07-01	14.994207	0.023504
2023-07-02	14.314220	0.023705
2023-07-03	14.585848	0.019438
2023-07-04	14.377220	0.019504
2023-07-05	14.814939	0.021202
2023-07-06	14.667792	0.022264
2023-07-07	15.535115	0.024421

```
fig, ax = plt.subplots(3,1,sharex=True, figsize=(8,8))

ax[0].plot(df['T_in'], c='tab:blue', label='inside')
ax[0].plot(df['T_out'], c='tab:orange', label='outside')
ax[0].set(ylabel='temperature (°C)',
          title="actual temperatures",
```

```python
          ylim=[10,45])

# formating dates on x axis
locator = mdates.AutoDateLocator(minticks=7, maxticks=11)
formatter = mdates.ConciseDateFormatter(locator)
ax[0].xaxis.set_major_locator(locator)
ax[0].xaxis.set_major_formatter(formatter)

ax[0].legend(ncol=2, loc='upper center', frameon=False)

ax[1].plot(result_df['r'], color='black')
ax[1].set(ylabel=r"parameter $r$",
          ylim=[0, 0.04])

ax[2].plot(result_df['T_env'], color='black')
ax2b = ax[2].twinx()
ax2b.plot(df_night['T_out'].resample('D').mean(), color='tab:orange')
ax[2].set(ylim=[12, 17])
ax2b.set(ylim=[19, 24],
        ylabel='outside temperature')
# color the xticks
for tick in ax[2].get_yticklabels():
    tick.set_color('tab:orange')
# color the xlabel
ax[2].set_ylabel(r'"outside" temp.'+'\ninferred from\nanalysis', color='tab:orange')
```

```
Text(0, 0.5, '"outside" temp.\ninferred from\nanalysis')
```



Conclusions:

1. The cooling coefficient r seems quite stable throughout the two weeks of measurements. This probably says that the greenhouse and AC properties did not change much. For instance, the greenhouse thermal insulation stayed constant, and the AC power output stayed constant.
2. The AC tracks very well the outside temperature! This is to say: the AC works better (more easily) when temperatures outsides are low, and vice-versa.

14 Savitzky–Golay

The Savitzky-Golay filter, also known as LOESS, smoothes a noisy signal by performing a polynomial fit over a sliding window.

Polynomial fit of order 3, window size = 51 pts

Polynomial fit of order 2, window size = 51 pts

The simulations look different because the order of the polynomial makes a very different impression on us, but in reality the outcome of the two filtering is almost identical:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import datetime as dt
import matplotlib.ticker as ticker
from scipy.signal import savgol_filter
import os
import warnings
import scipy
# Suppress FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
# %matplotlib widget
```

```
# dirty trick to have dates in the middle of the 24-hour period
# make minor ticks in the middle, put the labels there!
# from https://matplotlib.org/stable/gallery/ticks/centered_ticklabels.html
```

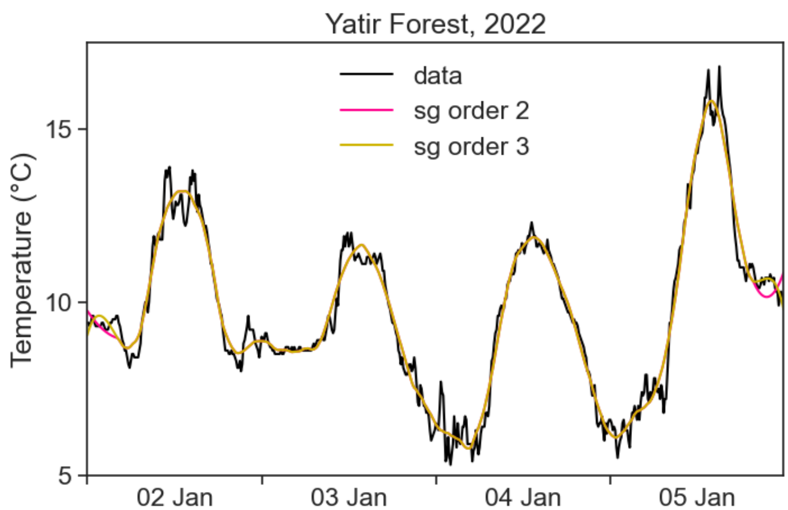
```python
def centered_dates(ax):
    date_form = DateFormatter("%d %b")  # %d 3-letter-Month
    # major ticks at midnight, every day
    ax.xaxis.set_major_locator(mdates.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(date_form)
    # minor ticks at noon, every day
    ax.xaxis.set_minor_locator(mdates.HourLocator(byhour=[12]))
    # erase major tick labels
    ax.xaxis.set_major_formatter(ticker.NullFormatter())
    # set minor tick labels as define above
    ax.xaxis.set_minor_formatter(date_form)
    # completely erase minor ticks, center tick labels
    for tick in ax.xaxis.get_minor_ticks():
        tick.tick1line.set_markersize(0)
        tick.tick2line.set_markersize(0)
        tick.label1.set_horizontalalignment('center')
```

```python
df = pd.read_csv('shani_2022_january.csv', parse_dates=['date'], index_col='date')
start = "2022-01-02"
end = "2022-01-05"
df = df.loc[start:end]
```

```python
df['sg_3_51'] = savgol_filter(df['TD'], window_length=51, polyorder=3)
df['sg_2_51'] = savgol_filter(df['TD'], window_length=51, polyorder=2)
```

```python
fig, ax = plt.subplots(figsize=(8,5))

plot_data, = ax.plot(df['TD'], color='black')
plot_sg2, = ax.plot(df['sg_2_51'], color='xkcd:hot pink')
plot_sg3, = ax.plot(df['sg_3_51'], color='xkcd:mustard')

ax.legend(handles=[plot_data, plot_sg2, plot_sg3],
          labels=['data', 'sg order 2', 'sg order 3'],
          frameon=False)

plot_settings = {
    'ylim': [5, 17.5],
    'xlim': [df.index[0], df.index[-1]],
```

```
    'ylabel': "Temperature (°C)",
    'title': "Yatir Forest, 2022",
    'yticks': [5, 10, 15]
}

ax.set(**plot_settings)
centered_dates(ax)
```



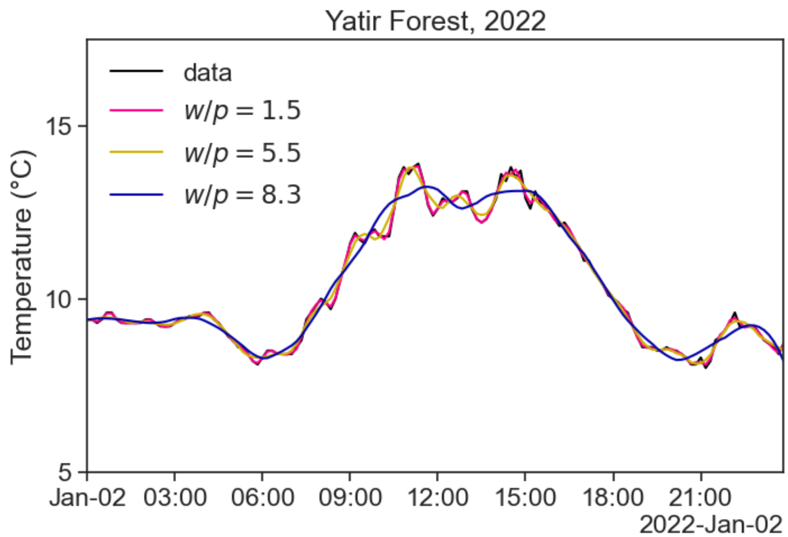
To really see the difference between window width and polynomial order, we need to play with their ratio,

$$\text{ratio} = \frac{w}{p} = \frac{\text{window width}}{\text{polynomial order}}$$

```
start = "2022-01-02 00:00:00"
end = "2022-01-02 23:50:00"
df = df.loc[start:end]
```

```
# window_length, polyorder
df['sg_1'] = savgol_filter(df['TD'], 5, 3)
df['sg_2'] = savgol_filter(df['TD'], 11, 2)
df['sg_3'] = savgol_filter(df['TD'], 25, 3)
```

```python
fig, ax = plt.subplots(figsize=(8,5))

plot_data, = ax.plot(df['TD'], color='black')
plot_sg1, = ax.plot(df['sg_1'], color='xkcd:hot pink')
plot_sg2, = ax.plot(df['sg_2'], color='xkcd:mustard')
plot_sg3, = ax.plot(df['sg_3'], color='xkcd:royal blue')

ax.legend(handles=[plot_data, plot_sg1, plot_sg2, plot_sg3],
          labels=['data', r'$w/p=1.5$', r'$w/p=5.5$', r'$w/p=8.3$'],
          frameon=False)

plot_settings = {
    'ylim': [5, 17.5],
    'xlim': [df.index[0], df.index[-1]],
    'ylabel': "Temperature (°C)",
    'title': "Yatir Forest, 2022",
    'yticks': [5, 10, 15]
}

ax.set(**plot_settings)

locator = mdates.AutoDateLocator(minticks=7, maxticks=11)
formatter = mdates.ConciseDateFormatter(locator)

ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)
```



The higher the ratio, the more aggressive the smoothing.

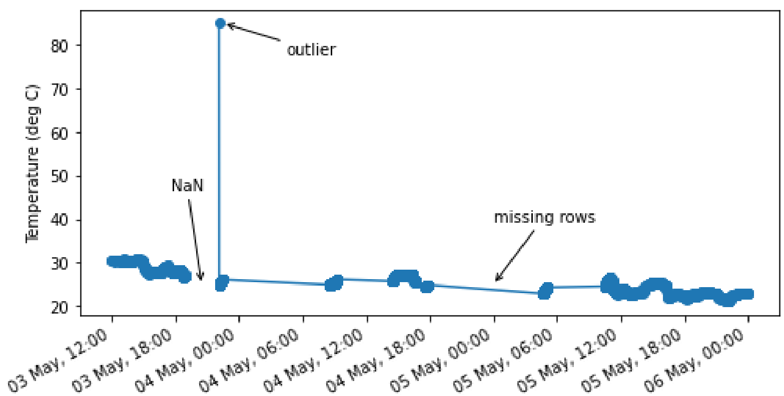
There is **a lot** more about the Savitzky-Golay filter, but for our purposes this is enough. If you want some more discussion about how to choose the parameters of the filter, read this.

Part IV

outliers and gaps

15 motivation

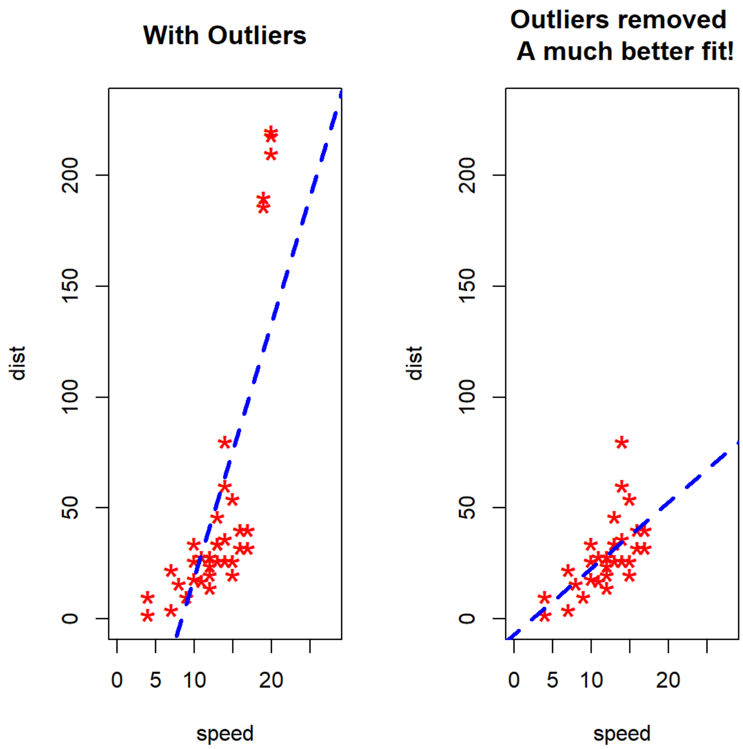
Outliers are observations significantly different from all other observations. Consider, for example, this temperature graph:



While most measured points are between 20 and 30 °C, there is obviously something very wrong with the one data point above 80 °C.

How could such a thing come about? This could be the result of **non-natural causes**, such as measurement errors, wrong data collection, or wrong data entry. On the other hand, this point could have **natural** sources, such as a very hot spark flying next to the temperature sensor.

Identifying outliers is important, because they might greatly impact measures like mean and standard deviation. When left untouched, outliers might make us reach wrong conclusions about our data. See what happens to the slope of this linear regression with and without the outliers.

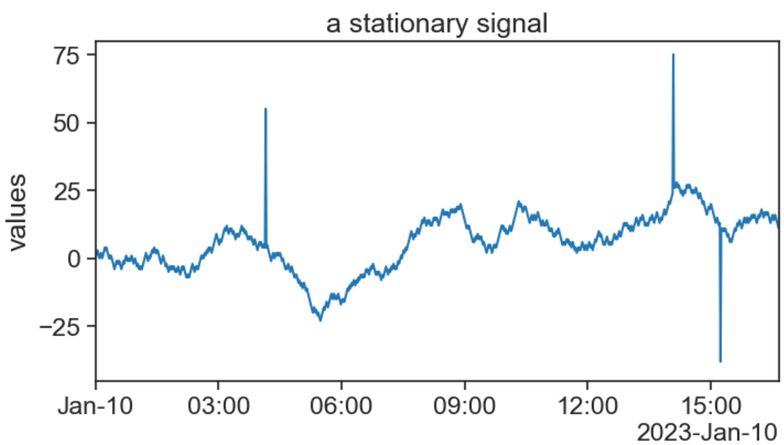


Source: Zhang (2020)

16 outlier identification

16.1 visual inspection

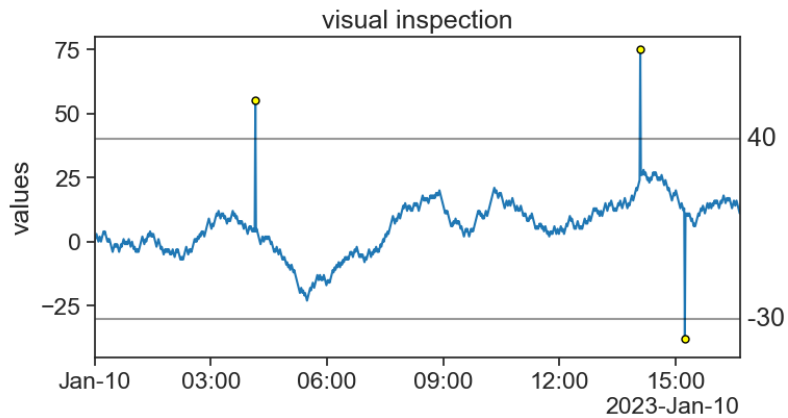
I produced a stationary signal and added to it a few ouliers. Can you tell where just by looking at the graph?



The easiest way of identifying the outliers is:

- First plot the time series.
- Choose upper and lower boundaries. Whatever falls outside these boundaries is an outlier.

Easy.



If all you have is this one time series, you're done, congratulations. However, it is often the case that one has very long time series, or a great number of time series to analyze. In this case it is impractical to use the visual inspection method. We would like to devise an algorithm to automate this task.

16.2 Z-score

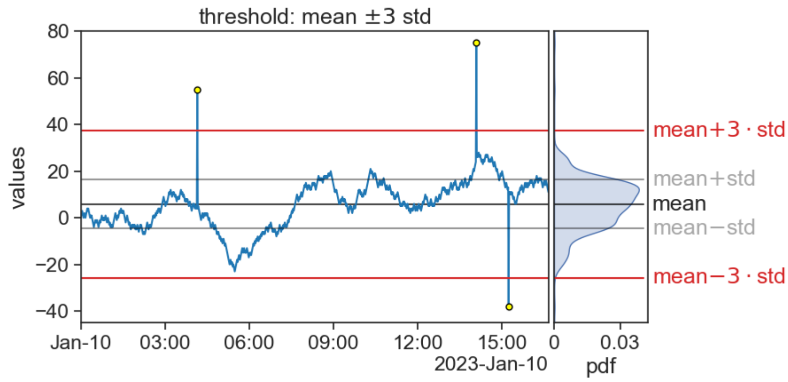
The Z-score is the distance, in units of 1 standard deviation, of a point in the series with respect to the mean:

$$z = \frac{x - \mu}{\sigma},$$

where

- x = data point,
- μ = time series mean
- σ = time series standard deviation.

A common choice is to consider an outlier a point whose Z-score is greater that 3, in absolute value. In other words: If a point is more than 3 standard deviations away form the mean, then we call it an outlier.

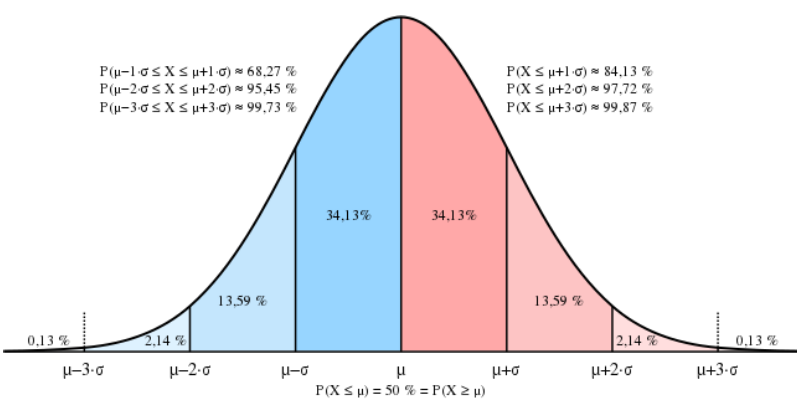


You can now use this algorithm to any number of time series, let the computer do the hard work.

Of course, there is nothing sacred about the number 3. You can choose any Z-score you want to perform an analysis on your own data, depending on your needs.

16.2.1 ATTENTION!

For data that is gaussianly distributed, we expect that 99.73% of data to fall within 3 standard deviations from the mean. In other words, 0.27% of points would be considered as *outliers* according to the Z-score method.

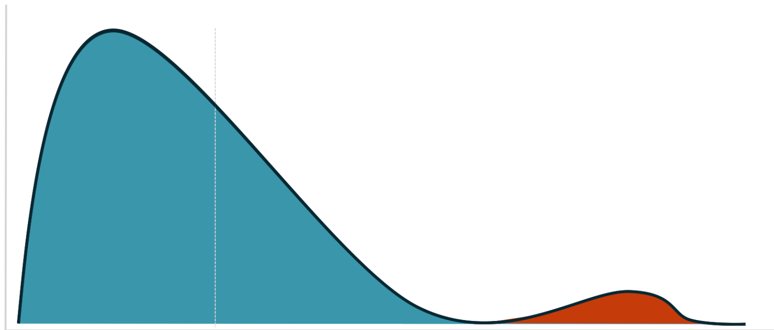


Source: Wikimedia Commons

Assume you have a time series gaussianly distributed, with 10k measurements. We would expect to find about 27 outliers in this time series.

So what is the problem?!

The thing is, outliers are not supposed to be only data points far from the other points. That's not enough. A better way of understanding outliers is to imagine that our expected measurements are sampled from a given distribution, and every now an then we have measurements that are sampled from **another** distribution.



We should have this in mind always. We wouldn't want to single out good data as something weird. Our true task is to identify which points in our time series were sampled from a different distribution. This can be a very challenging task.

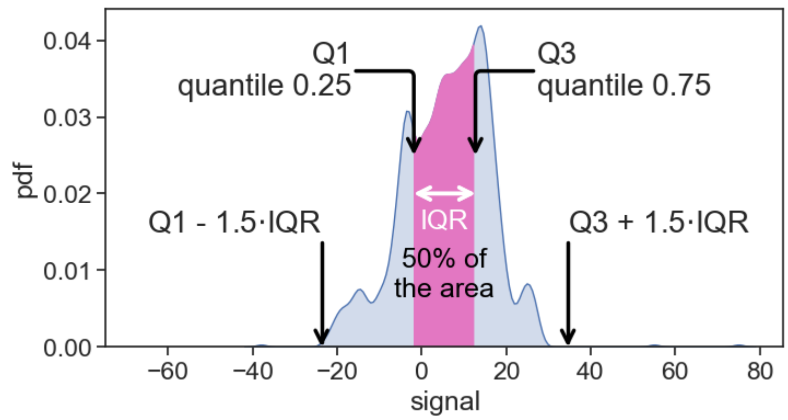
Source: Taylor Wilson's "Dealing with Outliers (Part 1): Ignore Them at Your Peril"

16.3 IQR

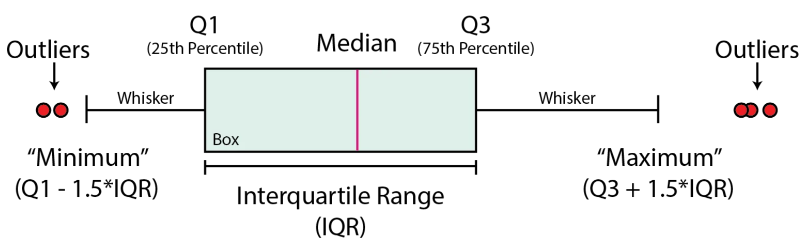
Another super common criterion for identifying outliers is the IQR, or InterQuartile Range.

Take a look at the statistics below of the time series we have been working with so far. The IQR is the distance between the first quartile (Q1) and the third quartile (Q3), where exactly 50% of the data is.

The algorithm here is to determine two thresholds, whose distance is 1.5 times the IQR from Q1 and Q3. Whatever falls outside these two thresholds is an outlier.

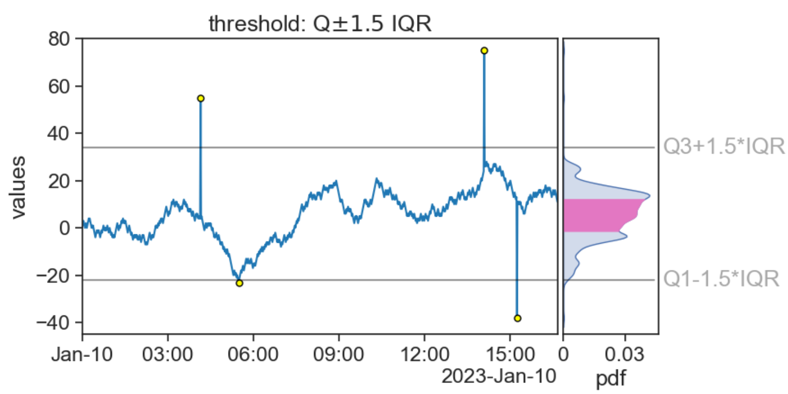


We are used to see this in box plots:



Source: McDonald (2022)

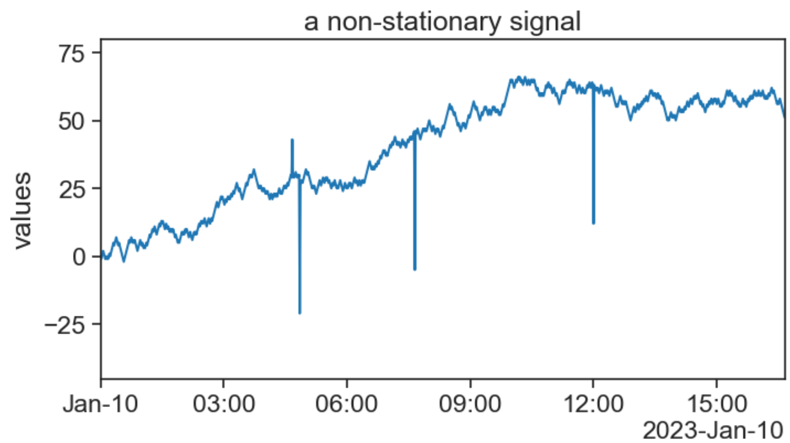
Again, the distance 1.5 is not sacred, it's only the most common. You might want to choose other values depending on your needs. Let's now apply the IQR method to our time series.



It works pretty well! Notice that now we have an additional outlier (a bit before 06:00). What do we do with that?

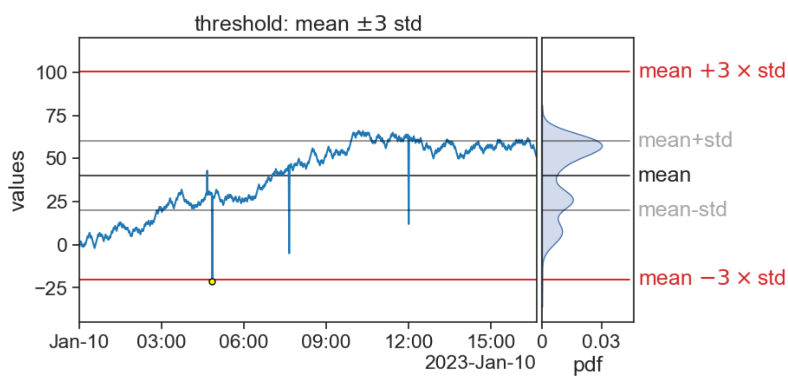
16.4 non-stationary time series

I have produced a new time series, one that on average goes up with time. Can you point in the graph where are the outliers?

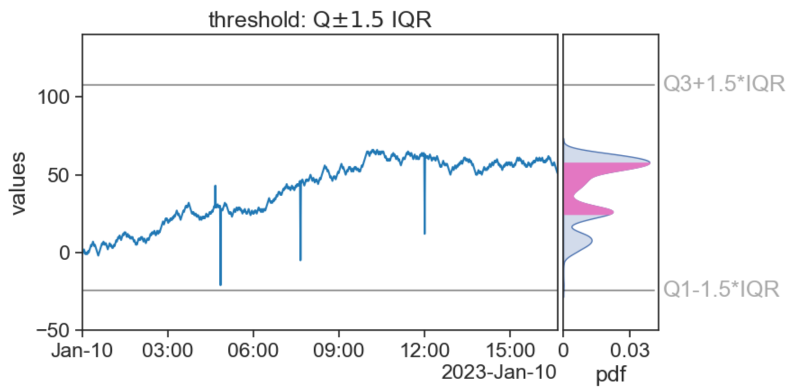


Now, see what happens when we apply the previous two methods to this time series.

Z-score



IQR



What happened? Do you have ideas how to solve this?

16.5 Sources

17 robust analysis

A tool is said to be **robust** if outliers don't influence (much) its results.

The average and standard deviation are **not** robust.

```python
import numpy as np
series1 = np.array([0, 1, 2, 3, 4, 5, 6])
series2 = np.array([0, 1, 2, 3, 4, 5, 60])
print(f"series 1: mean={series1.mean():.2f}, std={series1.std():.2f}")
print(f"series 2: mean={series2.mean():.2f}, std={series2.std():.2f}")
```

```
series 1: mean=3.00, std=2.00
series 2: mean=10.71, std=20.18
```

On the other hand, the median and IQR are robust:

```python
from scipy.stats import iqr
print(f"series 1: median={np.median(series1):.2f}, IQR={iqr(series1):.2f}")
print(f"series 2: median={np.median(series2):.2f}, IQR={iqr(series2):.2f}")
```

```
series 1: median=3.00, IQR=3.00
series 2: median=3.00, IQR=3.00
```

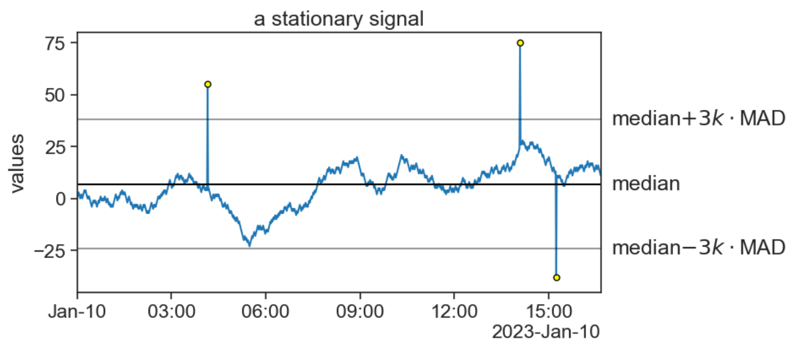
17.1 MAD

Another rubust method is MAD, the Median Absolute Deviation, given by

$$\text{MAD} = \text{median}(|x_i - \text{median}(x)|),$$

where $|\cdot|$ is the absolute value.

Applying MAD to the stationary time series from before, yields

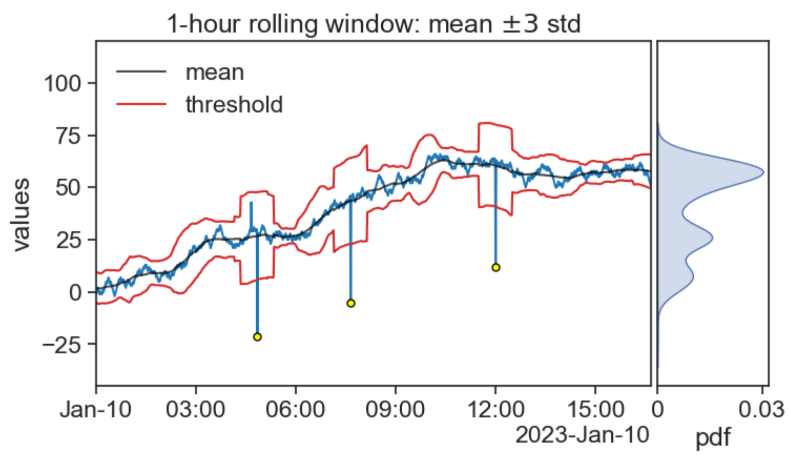


Here, the threshold is the median $\pm 3k\cdot$ MAD, where the value $k = 1.4826$ scales MAD so that when the data is gaussianly distributed, $3k$ equals 1 standard deviation.

18 sliding algorithms

None of the methods learned before seem appropriate to deal with non-stationary data. A simple solution is to apply those methods for sliding windows of "appropriate" widths.

18.1 Sliding Z-score

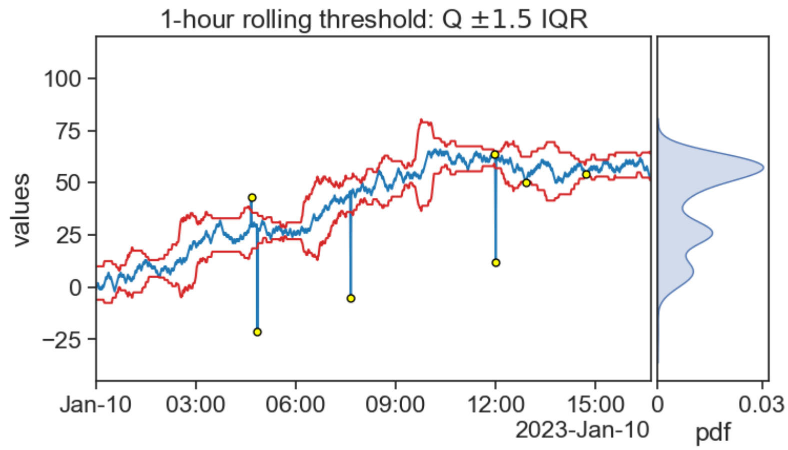


Now the Z-score seems to give really nice results (but not perfect). Maybe playing with the window width and Z-score threshold would give better results?

In any case, we clearly see why the Z-score is not a robust algorithm. See how the standard deviation is sensitive to outliers?

18.2 Sliding IQR

Let's see how well the sliding IQR method fares.



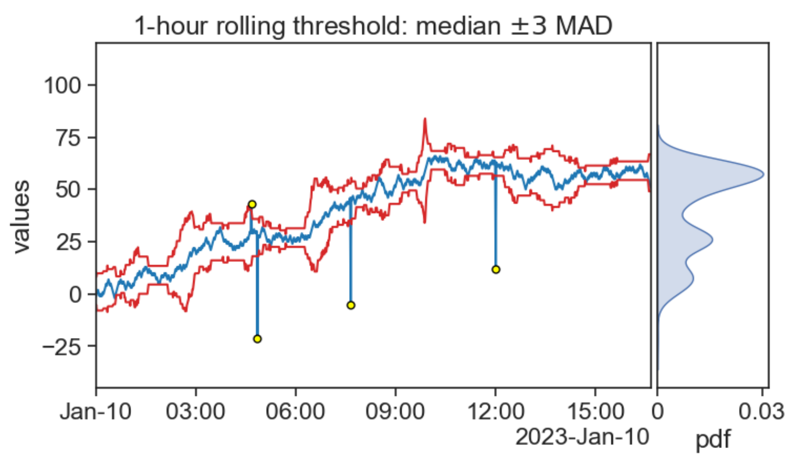
It identified all the outliers, but also found that a few *other* points should be considered outliers? What do you think of that?

See that the threshold does not jump abruptly when the sliding window includes an outlier. In fact, the threshold doesn't even care! This is what it means to be robust.

However, we do see large fluctuations in the threshold. When does this happen? Why?

18.3 Sliding MAD

Now it's MAD's time to shine.



Compare this result to the previous two. Which yields best results?

MAD is robust to outliers, and again we see that the threshold envelope widens when there is a rising or falling trend in the data.

18.4 Challenges

Now it's your turn to work, I'm tired! Write algorithms for the following outlier identification methods:

1. visual inspection
2. Z-score
3. IQR
4. MAD

Excluding the visual inspection method, write first an algorithm that operates on a full time series, and then write a new version that can work with sliding windows.

19 substituting outliers

Ok. We found the outliers. Now what?!

As usual, it depends.

19.1 Do nothing

Assuming the outlier indeed happened in real life, and is not the result of faulty data transmission or bad data recording, then excluding an outlier might be the last thing you want to do. Sometimes extreme events do happen, such as a one-in-a-hundred-year storm, and they have a disproportionate weight on the system you are studying. The outliers might actually be the most interesting points in your data for all you know!

In case the outliers are not of interest to you, if you are using **robust** methods to analyze your data, you don't necessarily need to do anything either. For instance, let's say that you want to smooth your time series. If instead of taking the `mean` inside a sliding window you choose to calculate the `median`, then outliers shouldn't be a problem. Test it and see if it's true. Go on.

For many things you need to do (not only smoothing), you might be able to find robust methods. What do you do if you **have** to use a non-robust method? Well, then you can substitute the outlier for two things: NaN or imputated values.

19.2 NaN

Substitute outliers for NaN.

NaN means “Not a Number”, and is what you get when you try to perform a mathematical operation like 0/0. It is common to see NaN in dataset rows when data was not collected for some reason.

This might seem like a neutral solution, but it actually can generate problems down the line. See this example:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
from scipy.signal import savgol_filter
```

```
# example using numpy
series = np.array([2, 4, 5, np.nan, 8, 15])
mean = np.mean(series)
print(f"the series average is {mean}")
```

```
the series average is nan
```

A single NaN in your time series ruins the whole calculation! There is a workaround though:

```
mean = np.nanmean(series)
print(f"the series average is {mean}")
```

```
the series average is 6.8
```

You have to make sure what is the behavior of each function you use with respect to NaNs, and if possible, use a suitable substitute.

The same example in `pandas` would not fail:

```
date_range = pd.date_range(start='2024-01-01', periods=len(series), freq='1D')
df = pd.DataFrame({'series': series}, index=date_range)
mean = df['series'].mean()
print(f"the series average is {mean}")
```

```
the series average is 6.8
```

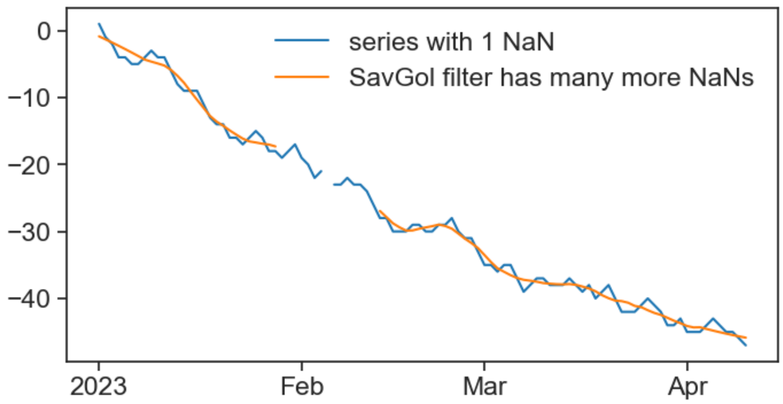
19.3 imputate values

To "imputate values" means to fill in the missing value with a guess, an estimation of what this data point "should have been" if it were measured in the first place. Why should we bother to do so? Because many tools that we know and love don't do well with missing values.

We learned about the Savitzky-Golay filter for smoothing data. See what happens when there is a single NaN in the series:

```
steps = np.random.randint(low=-2, high=2, size=100)
data = steps.cumsum()
date_range = pd.date_range(start='2023-01-01', periods=len(data), freq='1D')
df = pd.DataFrame({'series': data}, index=date_range)
df.loc['2023-02-05', 'series'] = np.nan
```

```
df['sg'] = savgol_filter(df['series'], window_length=15, polyorder=2)

def concise(ax):
    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)
    ax.xaxis.set_major_locator(locator)
    ax.xaxis.set_major_formatter(formatter)

fig, ax = plt.subplots(figsize=(8,4))
ax.plot(df['series'], color="tab:blue", label="series with 1 NaN")
ax.plot(df['sg'], color="tab:orange", label="SavGol filter has many more NaNs")
concise(ax)
ax.legend(frameon=False);
```



We will deal with this topic in the next chapter, “interpolation”. There, we will learn a few methods to fill in missing data, and basic NaN operations you should be acquainted with.

20 challenge

20.1 importing bad .csv files

Here we will get a taste of what it feels like to work with bad `.csv` files and how to fix them.

TO DO:

1. create a folder for this challenge, name it whatever you want.
2. downlad this jupyter notebook and move to that folder.
3. download these 3 `.csv` files and part 2 notebook:

    1. cleaning1
    2. cleaning2-
    3. cleaning3
    4. part_2

4. move them files into your folder

20.1.1 import

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import warnings
# Suppress FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
```

```
# %matplotlib widget
```

20.2 dataset 1

```
df1 = pd.read_csv('cleaning1.csv')
df1
```

	date	A	B	C	D	E
0	2023-01-01 00:00:00	0	0.000000	0.000000	0.000000	0.000000
1	2023-01-01 00:05:00	1	2.245251	-1.757193	1.899602	-0.999300
2	2023-01-01 00:10:00	2	2.909648	0.854732	2.050146	-0.559504
3	2023-01-01 00:15:00	3	3.483155	0.946937	1.921080	-0.402084
4	2023-01-01 00:20:00	2	4.909169	0.462239	1.368470	-0.698579
...	...	...	...	...	...	...
105115	2023-12-31 23:35:00	-37	1040.909898	-14808.285199	1505.020266	423.595984
105116	2023-12-31 23:40:00	-36	1040.586547	-14808.874072	1503.915566	423.117110
105117	2023-12-31 23:45:00	-37	1042.937417	-14808.690745	1505.479671	423.862810
105118	2023-12-31 23:50:00	-36	1042.411572	-14809.212002	1506.174334	423.862432
105119	2023-12-31 23:55:00	-35	1043.053520	-14809.990338	1505.767197	423.647007

Now let's put the column 'date' in the index with datetime format

```
# we can change the format of the column to datetime and then set it as the index.
df1['date'] = pd.to_datetime(df1['date'])
df1.set_index('date', inplace=True)
df1
```

	A	B	C	D	E
date					
2023-01-01 00:00:00	0	0.000000	0.000000	0.000000	0.000000
2023-01-01 00:05:00	1	2.245251	-1.757193	1.899602	-0.999300
2023-01-01 00:10:00	2	2.909648	0.854732	2.050146	-0.559504
2023-01-01 00:15:00	3	3.483155	0.946937	1.921080	-0.402084
2023-01-01 00:20:00	2	4.909169	0.462239	1.368470	-0.698579

	A	B	C	D	E
date					
...	...	...	...	...	...
2023-12-31 23:35:00	-37	1040.909898	-14808.285199	1505.020266	423.595984
2023-12-31 23:40:00	-36	1040.586547	-14808.874072	1503.915566	423.117110
2023-12-31 23:45:00	-37	1042.937417	-14808.690745	1505.479671	423.862810
2023-12-31 23:50:00	-36	1042.411572	-14809.212002	1506.174334	423.862432
2023-12-31 23:55:00	-35	1043.053520	-14809.990338	1505.767197	423.647007

If we know that in advance, we can write everything in one command when we import the csv.

```
df1 = pd.read_csv('cleaning1.csv',
                  index_col='date',      # set the column date as index
                  parse_dates=True)      # turn to datetime format
df1
```

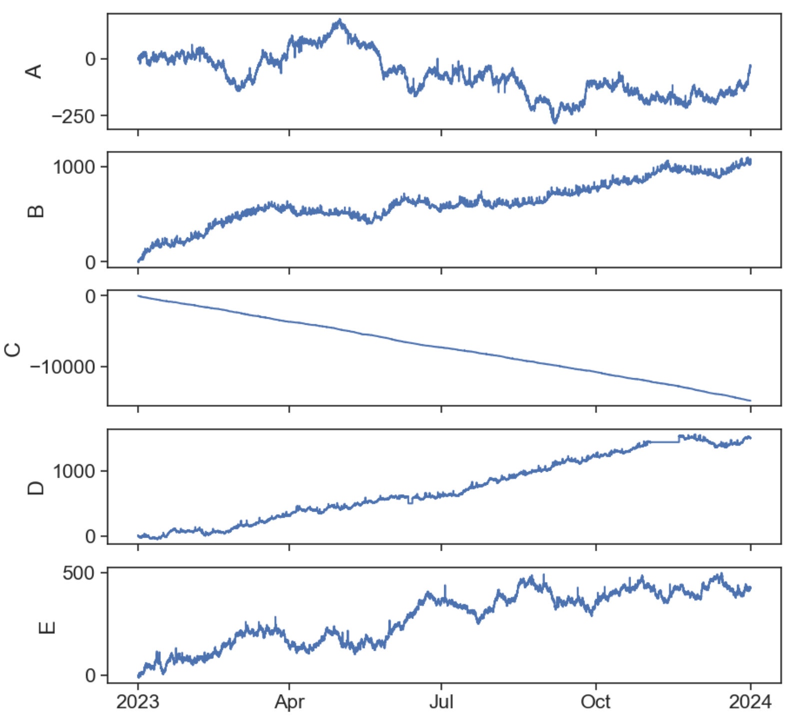
	A	B	C	D	E
date					
2023-01-01 00:00:00	0	0.000000	0.000000	0.000000	0.000000
2023-01-01 00:05:00	1	2.245251	-1.757193	1.899602	-0.999300
2023-01-01 00:10:00	2	2.909648	0.854732	2.050146	-0.559504
2023-01-01 00:15:00	3	3.483155	0.946937	1.921080	-0.402084
2023-01-01 00:20:00	2	4.909169	0.462239	1.368470	-0.698579
...	...	...	...	...	...
2023-12-31 23:35:00	-37	1040.909898	-14808.285199	1505.020266	423.595984
2023-12-31 23:40:00	-36	1040.586547	-14808.874072	1503.915566	423.117110
2023-12-31 23:45:00	-37	1042.937417	-14808.690745	1505.479671	423.862810
2023-12-31 23:50:00	-36	1042.411572	-14809.212002	1506.174334	423.862432
2023-12-31 23:55:00	-35	1043.053520	-14809.990338	1505.767197	423.647007

Now let's plot all the columns and see what we have.

```
def plot_all_columns(data):
    column_list = data.columns

    fig, ax = plt.subplots(len(column_list),1, sharex=True, figsize=(10,len(column_list)*2))
```

```
    if len(column_list) == 1:
        ax.plot(data[column_list[0]])
        return
    for i, column in enumerate(column_list):
        ax[i].plot(data[column])
        ax[i].set(ylabel=column)

    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)
    ax[i].xaxis.set_major_locator(locator)
    ax[i].xaxis.set_major_formatter(formatter)

    return
```

```
plot_all_columns(df1)
```



Looks like some of this data needs cleaning…

20.3 dataset 2

```
df2 = pd.read_csv('cleaning2-.csv')
df2
```

	A B date time
0	0.0 0.0 01012023 00:00:00
1	-2.0275363548598184 0.011984922825112581 01012...
2	-2.690616715983192 -0.29792822981957684 010120...
3	-1.9859899758267612 -0.30940867922490206 01012...
4	-2.290897621584889 -2.8666633353521624 0101202...
...	...
8755	-74.51464473079395 293.8680858227996 31122023 ...
8756	-74.73805809332175 294.7593463919649 31122023 ...
8757	-75.84842465358788 294.07634907736116 31122023...
8758	-77.27218272637339 293.526461290973 31122023 2...
8759	-76.55739976945038 293.35336890454107 31122023...

Something is wrong…
Let's open the actual `.csv` file and take a quick look.
It seems that the values are seperated by spaces and not by commas `,`.

```
df2 = pd.read_csv('cleaning2-.csv', delimiter=' ')
df2
```

	A	B	date	time
0	0.000000	0.0	1012023	00:00:00
1	-2.027536	0.011984922825112581	1012023	01:00:00
2	-2.690617	-0.29792822981957684	1012023	02:00:00
3	-1.985990	-0.30940867922490206	1012023	03:00:00
4	-2.290898	-2.8666633353521624	1012023	04:00:00
...	...	...	...	...
8755	-74.514645	293.8680858227996	31122023	19:00:00
8756	-74.738058	294.7593463919649	31122023	20:00:00
8757	-75.848425	294.07634907736116	31122023	21:00:00
8758	-77.272183	293.526461290973	31122023	22:00:00

	A	B	date	time
8759	-76.557400	293.35336890454107	31122023	23:00:00

```
# convert the date column to datetime
df2['date_corrected'] = pd.to_datetime(df2['date'])
# df2['date_corrected'] = pd.to_datetime(df2['date'], format='%d%m%Y')

df2
```

	A	B	date	time	date_corrected
0	0.000000	0.0	1012023	00:00:00	1970-01-01 00:00:00.001012023
1	-2.027536	0.011984922825112581	1012023	01:00:00	1970-01-01 00:00:00.001012023
2	-2.690617	-0.29792822981957684	1012023	02:00:00	1970-01-01 00:00:00.001012023
3	-1.985990	-0.30940867922490206	1012023	03:00:00	1970-01-01 00:00:00.001012023
4	-2.290898	-2.8666633353521624	1012023	04:00:00	1970-01-01 00:00:00.001012023
...	...	...	...	...	...
8755	-74.514645	293.8680858227996	31122023	19:00:00	1970-01-01 00:00:00.031122023
8756	-74.738058	294.7593463919649	31122023	20:00:00	1970-01-01 00:00:00.031122023
8757	-75.848425	294.07634907736116	31122023	21:00:00	1970-01-01 00:00:00.031122023
8758	-77.272183	293.526461290973	31122023	22:00:00	1970-01-01 00:00:00.031122023
8759	-76.557400	293.35336890454107	31122023	23:00:00	1970-01-01 00:00:00.031122023

```
# df2['date_corrected'] = pd.to_datetime(df2['date'][:780], format='%d%m%Y')
# df2['date_corrected'] = pd.to_datetime(df2['date'][780:800], format='%d%m%Y')
# df2['date'][780:800]
```

```
data_types = {'date': str , 'time':str}

# Read the CSV file with specified data types
df2 = pd.read_csv('cleaning2-.csv', delimiter=' ', dtype=data_types)
df2
```

	A	B	date	time
0	0.000000	0.0	01012023	00:00:00
1	-2.027536	0.011984922825112581	01012023	01:00:00
2	-2.690617	-0.29792822981957684	01012023	02:00:00

	A	B	date	time
3	-1.985990	-0.30940867922490206	01012023	03:00:00
4	-2.290898	-2.8666633353521624	01012023	04:00:00
...	...	...	...	...
8755	-74.514645	293.8680858227996	31122023	19:00:00
8756	-74.738058	294.7593463919649	31122023	20:00:00
8757	-75.848425	294.07634907736116	31122023	21:00:00
8758	-77.272183	293.526461290973	31122023	22:00:00
8759	-76.557400	293.35336890454107	31122023	23:00:00

```
df2['date_corrected'] = pd.to_datetime(df2['date'], format='%d%m%Y')
df2
```

	A	B	date	time	date_corrected
0	0.000000	0.0	01012023	00:00:00	2023-01-01
1	-2.027536	0.011984922825112581	01012023	01:00:00	2023-01-01
2	-2.690617	-0.29792822981957684	01012023	02:00:00	2023-01-01
3	-1.985990	-0.30940867922490206	01012023	03:00:00	2023-01-01
4	-2.290898	-2.8666633353521624	01012023	04:00:00	2023-01-01
...	...	...	...	...	...
8755	-74.514645	293.8680858227996	31122023	19:00:00	2023-12-31
8756	-74.738058	294.7593463919649	31122023	20:00:00	2023-12-31
8757	-75.848425	294.07634907736116	31122023	21:00:00	2023-12-31
8758	-77.272183	293.526461290973	31122023	22:00:00	2023-12-31
8759	-76.557400	293.35336890454107	31122023	23:00:00	2023-12-31

```
df2['datetime'] = pd.to_datetime(df2['date'] + ' ' + df2['time'], format='%d%m%Y %H:%M:%S')
df2
```

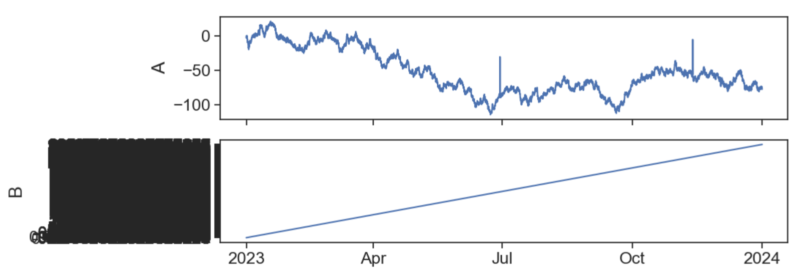
	A	B	date	time	date_corrected	datetime
0	0.000000	0.0	01012023	00:00:00	2023-01-01	2023-01-01 00:00:00
1	-2.027536	0.011984922825112581	01012023	01:00:00	2023-01-01	2023-01-01 01:00:00
2	-2.690617	-0.29792822981957684	01012023	02:00:00	2023-01-01	2023-01-01 02:00:00
3	-1.985990	-0.30940867922490206	01012023	03:00:00	2023-01-01	2023-01-01 03:00:00
4	-2.290898	-2.8666633353521624	01012023	04:00:00	2023-01-01	2023-01-01 04:00:00
...	...	...	...	...	...	...

	A	B	date	time	date_corrected	datetime
8755	-74.514645	293.8680858227996	31122023	19:00:00	2023-12-31	2023-12-31 19:00:00
8756	-74.738058	294.7593463919649	31122023	20:00:00	2023-12-31	2023-12-31 20:00:00
8757	-75.848425	294.07634907736116	31122023	21:00:00	2023-12-31	2023-12-31 21:00:00
8758	-77.272183	293.526461290973	31122023	22:00:00	2023-12-31	2023-12-31 22:00:00
8759	-76.557400	293.35336890454107	31122023	23:00:00	2023-12-31	2023-12-31 23:00:00

```
df2.drop(columns=['date', 'time', 'date_corrected'], inplace=True)
df2.set_index('datetime', inplace=True)
df2
```

	A	B
datetime		
2023-01-01 00:00:00	0.000000	0.0
2023-01-01 01:00:00	-2.027536	0.011984922825112581
2023-01-01 02:00:00	-2.690617	-0.29792822981957684
2023-01-01 03:00:00	-1.985990	-0.30940867922490206
2023-01-01 04:00:00	-2.290898	-2.8666633353521624
...	...	...
2023-12-31 19:00:00	-74.514645	293.8680858227996
2023-12-31 20:00:00	-74.738058	294.7593463919649
2023-12-31 21:00:00	-75.848425	294.07634907736116
2023-12-31 22:00:00	-77.272183	293.526461290973
2023-12-31 23:00:00	-76.557400	293.35336890454107

```
plot_all_columns(df2)
```



What happened in the second ax?

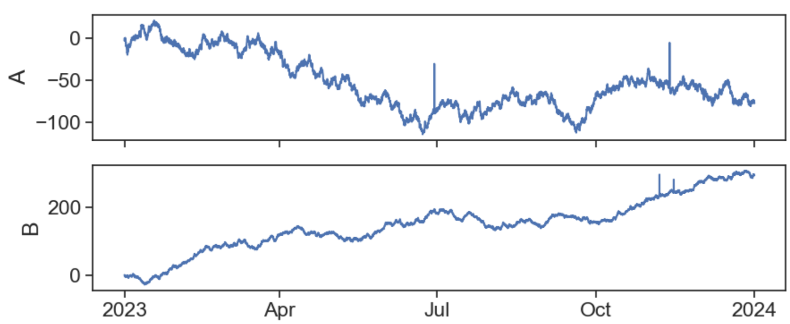
```
df2.dtypes
```

```
A    float64
B     object
dtype: object
```

```
# use pd.to_numeric to convert column 'B' to float
df2['B'] = pd.to_numeric(df2['B'],
                          errors='coerce'  # 'coerce' will convert non-numeric values to NaN
                          )
df2.dtypes
```

```
A    float64
B    float64
dtype: object
```

```
plot_all_columns(df2)
```




```
data_types = {'date': str , 'time':str}

# Read the CSV file with specified data types
df2 = pd.read_csv('cleaning2-.csv', delimiter=' ', dtype=data_types, na_values='-')
df2['datetime'] = pd.to_datetime(df2['date'] + ' ' + df2['time'], format='%d%m%Y %H:%M:%S')
df2.drop(columns=['date', 'time'], inplace=True)
df2.set_index('datetime', inplace=True)
df2
```

	A	B
datetime		
2023-01-01 00:00:00	0.000000	0.000000
2023-01-01 01:00:00	-2.027536	0.011985
2023-01-01 02:00:00	-2.690617	-0.297928
2023-01-01 03:00:00	-1.985990	-0.309409
2023-01-01 04:00:00	-2.290898	-2.866663
...	...	...
2023-12-31 19:00:00	-74.514645	293.868086
2023-12-31 20:00:00	-74.738058	294.759346
2023-12-31 21:00:00	-75.848425	294.076349
2023-12-31 22:00:00	-77.272183	293.526461
2023-12-31 23:00:00	-76.557400	293.353369

```
df2.to_csv('cleaning2_formated.csv')
```

20.4 dataset 3

```
df3 = pd.read_csv('cleaning3.csv')
df3
```

	#
0	# data created by
1	# Yair Mau
2	# for time series data analysis
3	#
4	# time format: unix (s)
...	...
370	6.651300774019661 1703635200.0
371	6.41517483494087151703721600.0
372	7.603140054178304 1703808000.0
373	8.668182044560869 1703894400.0
374	8.472767724946076 1703980800.0

Again, let's look at the actual `.csv` file.

```python
df3 = pd.read_csv('cleaning3.csv', comment='#')
df3
```

	A time
0	0.0 1672531200.0
1	-0.03202661701444382 1672617600.0
2	-0.5863508675173621 1672704000.0
3	-1.5759721567247762 1672790400.0
4	-2.7267995149281266 1672876800.0
…	…
360	6.651300774019661 1703635200.0
361	6.4151748349408715 1703721600.0
362	7.603140054178304 1703808000.0
363	8.668182044560869 1703894400.0
364	8.472767724946076 1703980800.0

```python
df3 = pd.read_csv('cleaning3.csv', comment='#', delimiter=' ')
df3
```

	A	time
0	0.000000	1.672531e+09
1	-0.032027	1.672618e+09
2	-0.586351	1.672704e+09
3	-1.575972	1.672790e+09
4	-2.726800	1.672877e+09
…	…	…
360	6.651301	1.703635e+09
361	6.415175	1.703722e+09
362	7.603140	1.703808e+09
363	8.668182	1.703894e+09
364	8.472768	1.703981e+09

```python
df3.dtypes
```

```
A       float64
time    float64
dtype: object
```

Time is in unix.
unix converter

```
print(df3['time'][0])
```

```
1672531200.0
```

```
df3['time'] = pd.to_datetime(df3['time'], unit='s')
df3.set_index('time', inplace=True)
df3
```

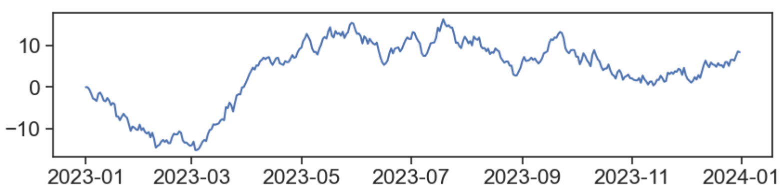
	A
time	
2023-01-01	0.000000
2023-01-02	-0.032027
2023-01-03	-0.586351
2023-01-04	-1.575972
2023-01-05	-2.726800
...	...
2023-12-27	6.651301
2023-12-28	6.415175
2023-12-29	7.603140
2023-12-30	8.668182
2023-12-31	8.472768

```
df3 = pd.read_csv('cleaning3.csv',
                  index_col='time',      # set the column date as index
                  parse_dates=True,      # turn to datetime format
                  comment='#',
                  delimiter=' ')
df3.index = pd.to_datetime(df3.index, unit='s')
df3
```

	A
time	
2023-01-01	0.000000

	A
time	
2023-01-02	-0.032027
2023-01-03	-0.586351
2023-01-04	-1.575972
2023-01-05	-2.726800
...	...
2023-12-27	6.651301
2023-12-28	6.415175
2023-12-29	7.603140
2023-12-30	8.668182
2023-12-31	8.472768

```
plot_all_columns(df3)
```




```
df3.to_csv('cleaning3_formated.csv')
```

21 challenge part 2

21.1 outliers and missing values

Here you have 3 dataframes that need cleaning. Use the methods learned in class to process the data.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import warnings
# Suppress FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters

# %matplotlib widget
```

```
df1 = pd.read_csv('cleaning1.csv',
                  index_col='date',     # set the column date as index
                  parse_dates=True)     # turn to datetime format
df1
```

	A	B	C	D	E
date					
2023-01-01 00:00:00	0	0.000000	0.000000	0.000000	0.000000
2023-01-01 00:05:00	1	2.245251	-1.757193	1.899602	-0.999300
2023-01-01 00:10:00	2	2.909648	0.854732	2.050146	-0.559504
2023-01-01 00:15:00	3	3.483155	0.946937	1.921080	-0.402084

	A	B	C	D	E
date					
2023-01-01 00:20:00	2	4.909169	0.462239	1.368470	-0.698579
...	...	...	...	...	...
2023-12-31 23:35:00	-37	1040.909898	-14808.285199	1505.020266	423.595984
2023-12-31 23:40:00	-36	1040.586547	-14808.874072	1503.915566	423.117110
2023-12-31 23:45:00	-37	1042.937417	-14808.690745	1505.479671	423.862810
2023-12-31 23:50:00	-36	1042.411572	-14809.212002	1506.174334	423.862432
2023-12-31 23:55:00	-35	1043.053520	-14809.990338	1505.767197	423.647007

```
df2 = pd.read_csv('cleaning2_formated.csv',
                  index_col='datetime',     # set the column date as index
                  parse_dates=True)     # turn to datetime format
df2
```

	A	B
datetime		
2023-01-01 00:00:00	0.000000	0.000000
2023-01-01 01:00:00	-2.027536	0.011985
2023-01-01 02:00:00	-2.690617	-0.297928
2023-01-01 03:00:00	-1.985990	-0.309409
2023-01-01 04:00:00	-2.290898	-2.866663
...	...	...
2023-12-31 19:00:00	-74.514645	293.868086
2023-12-31 20:00:00	-74.738058	294.759346
2023-12-31 21:00:00	-75.848425	294.076349
2023-12-31 22:00:00	-77.272183	293.526461
2023-12-31 23:00:00	-76.557400	293.353369

```
df3 = pd.read_csv('cleaning3_formated.csv',
                  index_col='time',     # set the column date as index
                  parse_dates=True,     # turn to datetime format
                 )
df3
```

	A
time	
2023-01-01	0.000000
2023-01-02	-0.032027
2023-01-03	-0.586351
2023-01-04	-1.575972
2023-01-05	-2.726800
...	...
2023-12-27	6.651301
2023-12-28	6.415175
2023-12-29	7.603140
2023-12-30	8.668182
2023-12-31	8.472768

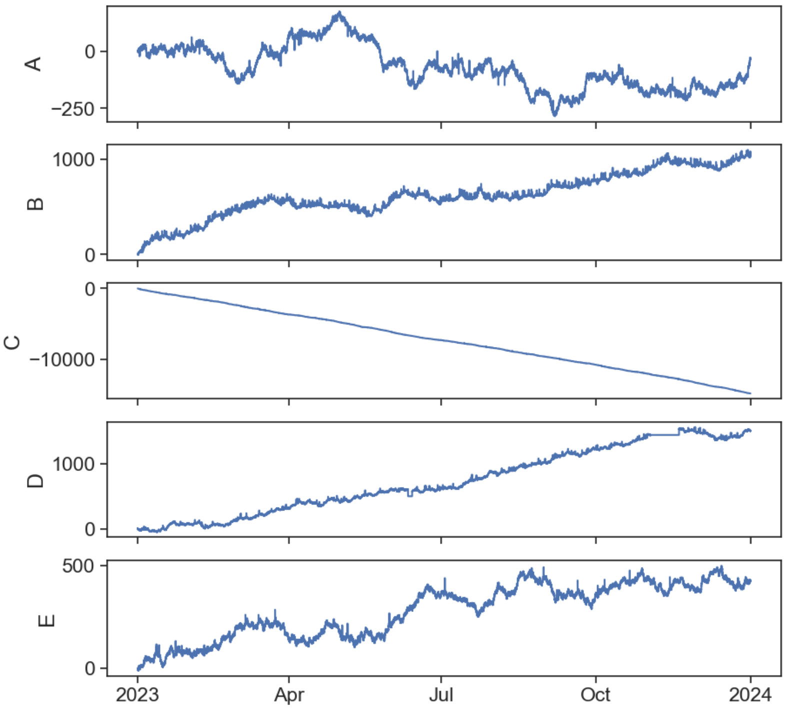
21.2 cleaning df1 from outliers

21.2.1 method 1: rolling standard deviation envelope

Visual inspection of all the data:

```
def plot_all_columns(data):
    column_list = data.columns

    fig, ax = plt.subplots(len(column_list),1, sharex=True, figsize=(10,len(column_list)*2))

    if len(column_list) == 1:
        ax.plot(data[column_list[0]])
        return
    for i, column in enumerate(column_list):
        ax[i].plot(data[column])
        ax[i].set(ylabel=column)

    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)
    ax[i].xaxis.set_major_locator(locator)
    ax[i].xaxis.set_major_formatter(formatter)

    return
```

```
plot_all_columns(df1)
```



Applying the rolling `std` method on column A:

```
# find the rolling std
df1['A_std'] = df1['A'].rolling(50, center=True, min_periods=1).std()
# find the rolling mean
df1['A_mean'] = df1['A'].rolling(50, center=True, min_periods=1).mean()
# define the k parameter -> the number of standard deviations from the mean which above them
k = 2
# finding the top and bottom threshold for each datapoint
df1['A_top'] = df1['A_mean'] + k*df1['A_std']
df1['A_bot'] = df1['A_mean'] - k*df1['A_std']
# creating a mask of booleans that places true if the row is an outliar and false if its not
df1['A_out'] = ((df1['A'] > df1['A_top']) | (df1['A'] < df1['A_bot']))
# applying the mask and replacing all outliers with nans.
df1['A_filtered'] = np.where(df1['A_out'],
                             np.nan,  # use this if A_out is True
```

```
                                          df1['A']) # otherwise
df1
```

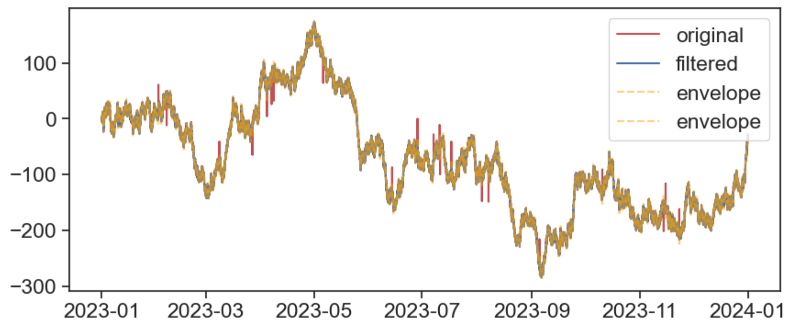
	A	B	C	D	E	A_std	A_mean	A_
date								
2023-01-01 00:00:00	0	0.000000	0.000000	0.000000	0.000000	1.262273	1.520000	4.0
2023-01-01 00:05:00	1	2.245251	-1.757193	1.899602	-0.999300	1.331858	1.423077	4.0
2023-01-01 00:10:00	2	2.909648	0.854732	2.050146	-0.559504	1.462738	1.296296	4.2
2023-01-01 00:15:00	3	3.483155	0.946937	1.921080	-0.402084	1.649114	1.142857	4.4
2023-01-01 00:20:00	2	4.909169	0.462239	1.368470	-0.698579	1.880022	0.965517	4.7
...	...	...	...	...	...	...	...	...
2023-12-31 23:35:00	-37	1040.909898	-14808.285199	1505.020266	423.595984	2.988291	-32.633333	-26
2023-12-31 23:40:00	-36	1040.586547	-14808.874072	1503.915566	423.117110	3.038748	-32.655172	-26
2023-12-31 23:45:00	-37	1042.937417	-14808.690745	1505.479671	423.862810	3.077483	-32.714286	-26
2023-12-31 23:50:00	-36	1042.411572	-14809.212002	1506.174334	423.862432	3.088901	-32.814815	-26
2023-12-31 23:55:00	-35	1043.053520	-14809.990338	1505.767197	423.647007	3.128283	-32.884615	-26

21.2.2 ploting the results:

Use `%matplotlib widget` to visualy inspect the results.

```
fig, ax = plt.subplots(figsize=(10,4))

ax.plot(df1['A'], c='r', label='original')
ax.plot(df1['A_filtered'], c='b', label='filtered')
ax.plot(df1['A_bot'], c='orange',linestyle='--', label='envelope', alpha=0.5)
ax.plot(df1['A_top'], c='orange',linestyle='--', label='envelope', alpha=0.5)

ax.legend()
```

```
<matplotlib.legend.Legend at 0x7f9028c8b5e0>
```



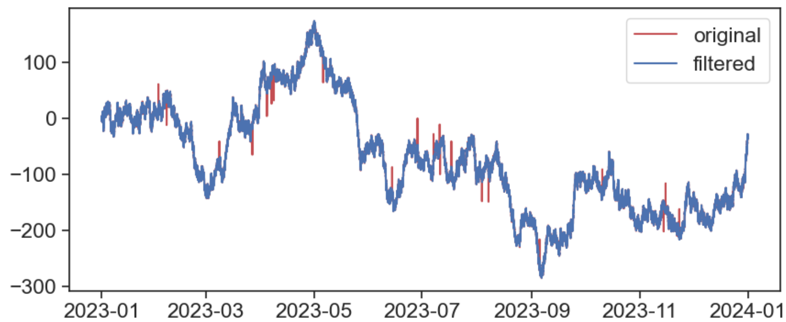
Now let's write a function so we can easly apply it to all our data:

```
def rolling_std_envelop(series, window_size=50, k=2):
    series.name = 'original'
    data = series.to_frame()
    data['std'] = data['original'].rolling(window_size, center=True, min_periods=1).std()
    # find the rolling mean
    data['mean'] = data['original'].rolling(window_size, center=True, min_periods=1).mean()
    # finding the top and bottom threshold for each datapoint
    data['top'] = data['mean'] + k*data['std']
    data['bottom'] = data['mean'] - k*data['std']
    # creating a mask of booleans that places true if the row is an outliar and false if its
    data['outliers'] = ((data['original'] > data['top']) | (data['original'] < data['bottom'
    # applying the mask and replacing all outliers with nans.
    data['filtered'] = np.where(data['outliers'],
                                np.nan,  # use this if outliers is True
                                data['original']) # otherwise
    return data['filtered']
```

Let's test the new function:

```
fig, ax = plt.subplots(figsize=(10,4))

ax.plot(df1['A'], c='r', label='original')
ax.plot(rolling_std_envelop(df1['A']), c='b', label='filtered')
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7f9029b58070>
```



Now let's reload `df1` so it will be clean (without all the added columns from before) and apply the function on all columns:

```
df1 = pd.read_csv('cleaning1.csv',
                  index_col='date',     # set the column date as index
                  parse_dates=True)     # turn to datetime format
```

```
df1_filtered = df1.copy()
df1_filtered
```

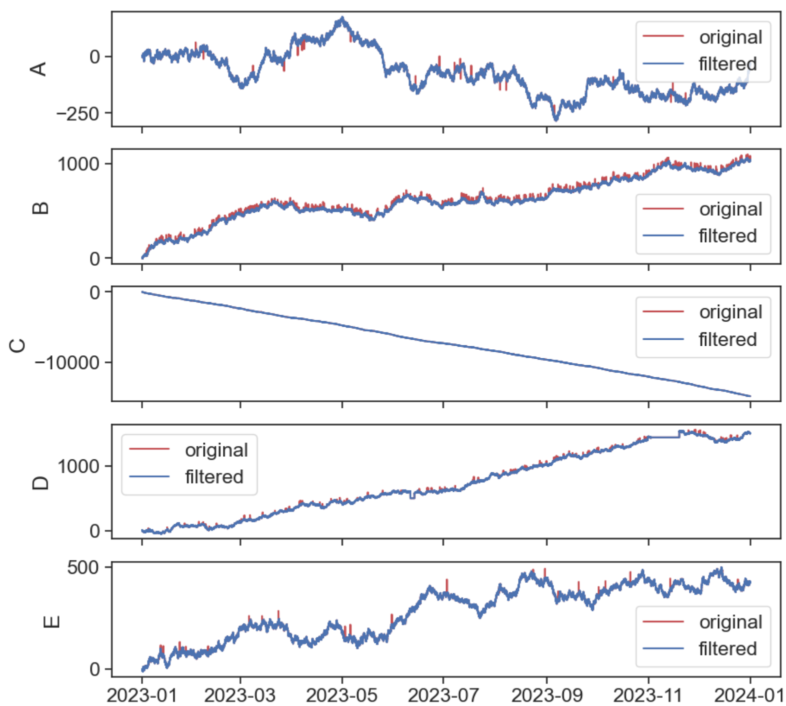
	A	B	C	D	E
date					
2023-01-01 00:00:00	0	0.000000	0.000000	0.000000	0.000000
2023-01-01 00:05:00	1	2.245251	-1.757193	1.899602	-0.999300
2023-01-01 00:10:00	2	2.909648	0.854732	2.050146	-0.559504
2023-01-01 00:15:00	3	3.483155	0.946937	1.921080	-0.402084
2023-01-01 00:20:00	2	4.909169	0.462239	1.368470	-0.698579
…	…	…	…	…	…
2023-12-31 23:35:00	-37	1040.909898	-14808.285199	1505.020266	423.595984
2023-12-31 23:40:00	-36	1040.586547	-14808.874072	1503.915566	423.117110
2023-12-31 23:45:00	-37	1042.937417	-14808.690745	1505.479671	423.862810
2023-12-31 23:50:00	-36	1042.411572	-14809.212002	1506.174334	423.862432
2023-12-31 23:55:00	-35	1043.053520	-14809.990338	1505.767197	423.647007

```
columns = df1_filtered.columns

for column in columns:
```

```
    filtered_column = rolling_std_envelop(df1_filtered[column], window_size=50, k=2)
    df1_filtered[column] = filtered_column
```

Now let's plot the results:

```
fig, ax = plt.subplots(len(columns),1, sharex=True, figsize=(10,len(columns)*2))

for i, column in enumerate(columns):
    ax[i].plot(df1[column], c='r', label='original')
    ax[i].plot(df1_filtered[column], c='b', label='filtered')
    ax[i].legend()
    ax[i].set(ylabel=column)
```



21.3 plateaus

Now what about the plateaus in the series D?
That's another form of outliers.

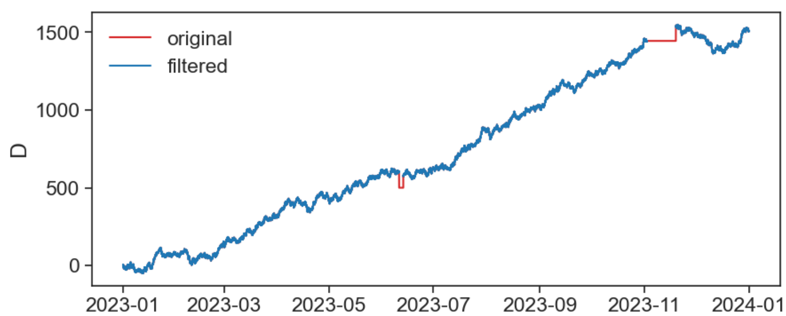
The following function will find rows wich have more than `n` equal following values and replace them with NaNs.

```
# function to copy paste:
def conseq_series(series, N):
    """
    part A:
    1. assume a string of 5 equal values. that's what we want to identify
    2. diff produces a string of only 4 consecutive zeros
    3. no problem, because when applying cumsum, the 4 zeros turn into a plateau of 5, that'
    so far, so good
    part B:
    1. groupby value_grp splits data into groups according to cumsum.
    2. because cumsum is monotonically increasing, necessarily all groups will be composed o
    3. what are those groups made of? of rows of column 'series'. this specific column is no
    4. count 'counts' the number of elements inside each group.
    5. the real magic here is that 'transform' assigns each row of the original group with t
    6. finally, we can ask the question: which rows belong to a string of identical values g
    zehu, you now have a mask (True-False) with the same shape as the original series.

    """
    # part A:
    sumsum_series = (
                (series.diff() != 0)         # diff zero becomes false, otherwise true
                   .astype('int')           # true -> 1  , false -> 0
                   .cumsum()                # cumulative sum, monotonically increasing
               )
    # part B:
    mask_outliers = (
                series.groupby(sumsum_series)           # take original series and group
                            .transform('count')        # now count how many are in each
                            .ge(N)                     # if row count >= than user-define
                )

    # apply mask:
    result = pd.Series(np.where(mask_outliers,
                             np.nan,  # use this if mask_outliers is True
                             series), # otherwise
                         index=series.index)
    return result
```

Let's apply it to the df1_filtered df so we will end with a cleaner signal

```
fig, ax = plt.subplots(figsize=(10,4))
ax.plot(df1_filtered['D'], color="tab:red", label='original')
ax.plot(conseq_series(df1_filtered['D'], 10), c='tab:blue', label='filtered')
ax.set_ylabel('D')
ax.legend(frameon=False)
```

```
<matplotlib.legend.Legend at 0x7f9029b0e9a0>
```



21.3.1 TO DO:

it's not homework but you should definitely do it

- Try other filtering methods
- Tweak the parameters
- Use other dataframes (1,2,3 and if you have your own so better)
- Write custom filtering functions that you can save and use in your future work.

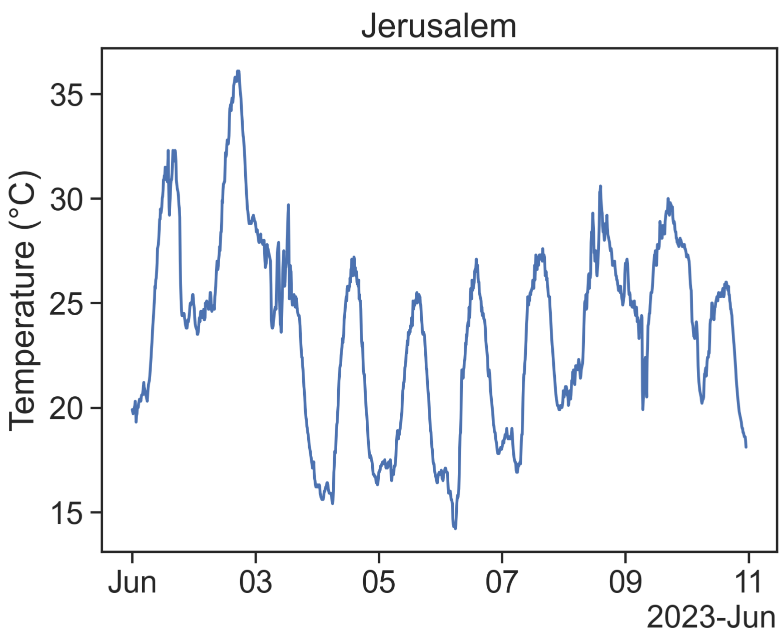
Yalla have fun

Part V

stationarity

22 motivation

See the temperatures for Jerusalem in a 4-day interval:



22.1 questions

- If I know the temperature right now, what does that tell me about the temperature 10 minutes from now? How about 100 minutes? 1000 minutes?
- The same applies to the past: how heavily does past information inform today's measurements?
- **information degradation**: how fast does information from a signal degrade, and gets swamped by noise?

22.2 goals

- discuss what is noise, signal, trend, cycles, etc.
- learn a useful framework to make sense of all the above.
- acquire statistical and time-series tools to analyze my data
- eventually, all the above will be crucial to forcast future states.

Let's go!

23 random variables

This lecture is partially based on Brockwell and Davis (2016, chaps. 1, 2). Also this.

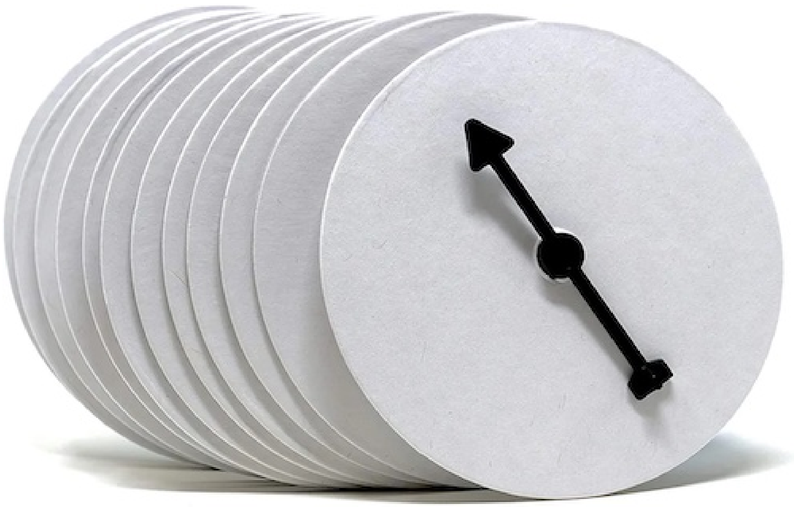
A random variable is a mathematical concept used in probability theory and statistics to represent and quantify uncertainty. It is a variable whose possible values are outcomes of a random phenomenon. In other words, it's a variable that can take on different values with certain probabilities associated with each value.

discrete random variable

There is a countable number of distinct outcomes. The obvious examples are coins and dice, which have 2 and 6 possible outcomes.

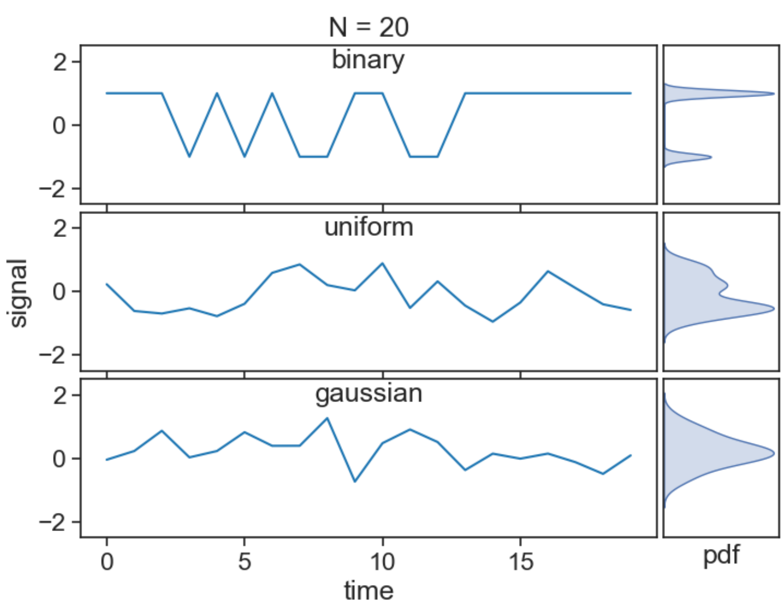
continuous random variable

Any value within a range is possible. The position of a horizontal game spinner is a good example.

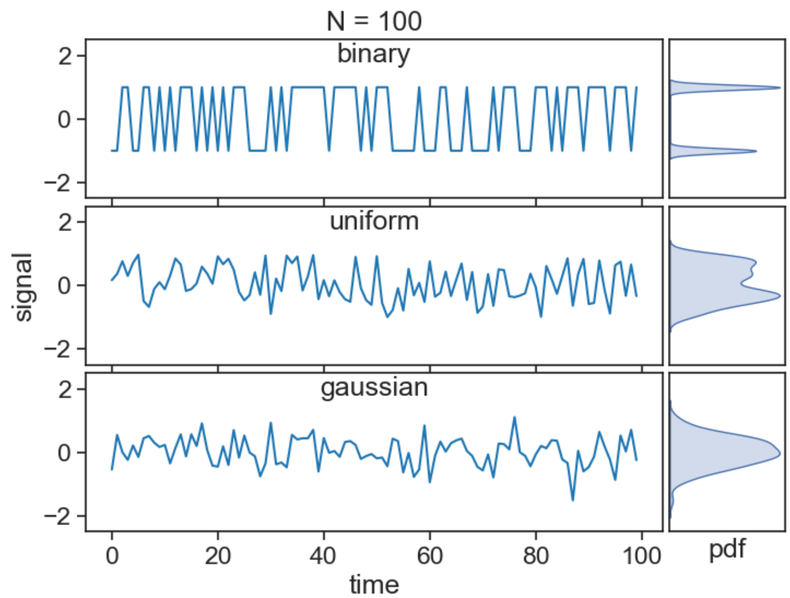


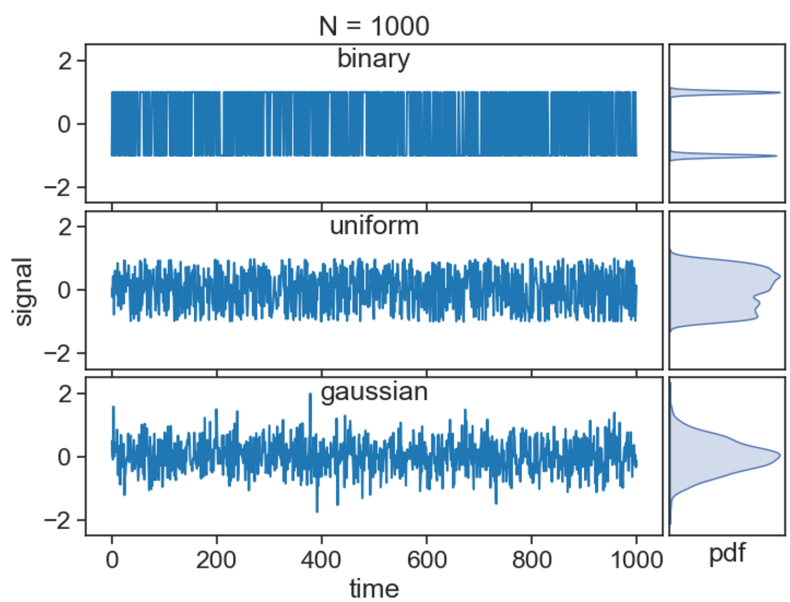
23.1 white noise

See below three time series made up by three different stochastic (random) processes. All terms in **each** of the series are **independent and identically distributed (iid)**, meaning that they are uncorrelated and taken from the same underlying distribution.



As we increase the length of the series, the statistics of each series reveal hints of the distributions they were sampled from:





The mathematical way of describing these series is thus: $\{X\}$ represents the stochastic process (binary, uniform, etc), from which a specific series is randomly drawn:

$$\{x_0, x_1, x_2, \cdots\}$$

All of these processes above have zero mean ($\mu = 0$) and a finite variance (σ^2), which qualify them as **white noise**.

23.2 random walk

A random walk S_t (for $t = 0, 1, 2, \cdots$) is obtained by cumulatively summing iid random variables:

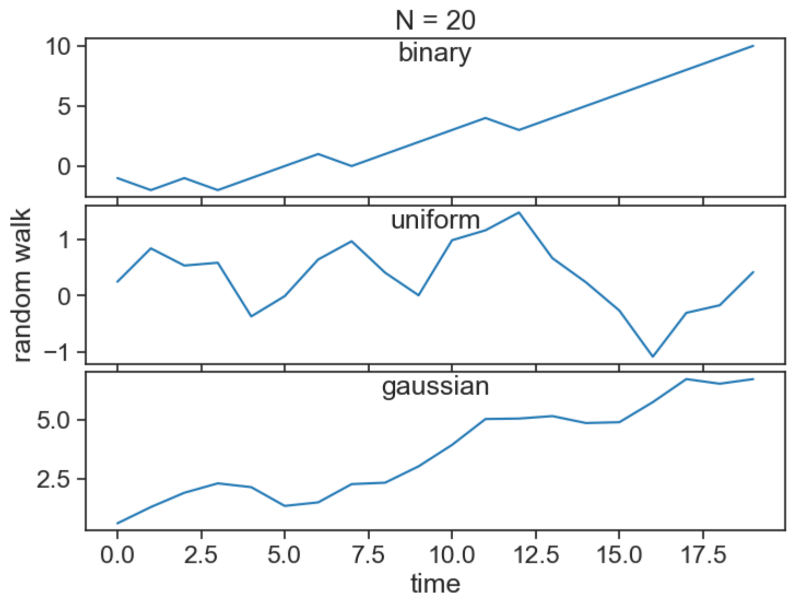
$$S_t = X_1 + X_2 + X_3 + \cdots + X_{t-1} + X_t$$

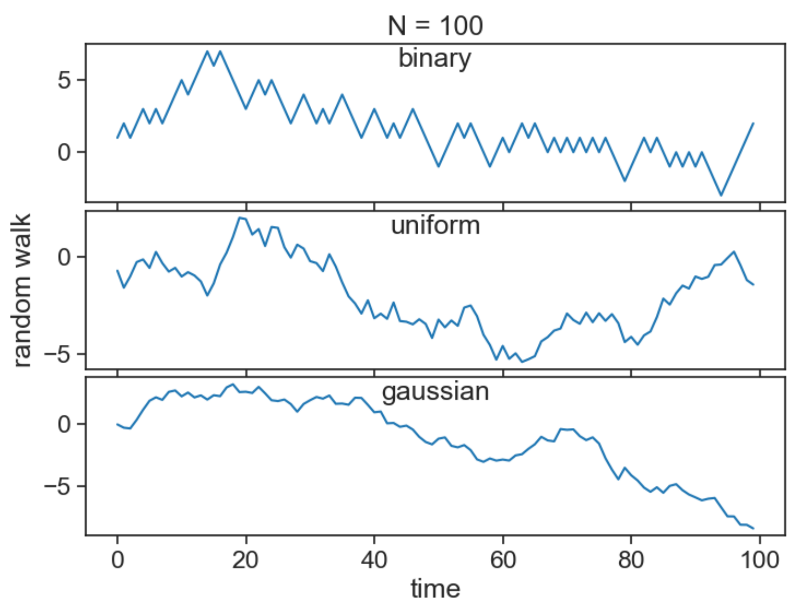
where $S_0 = 0$.

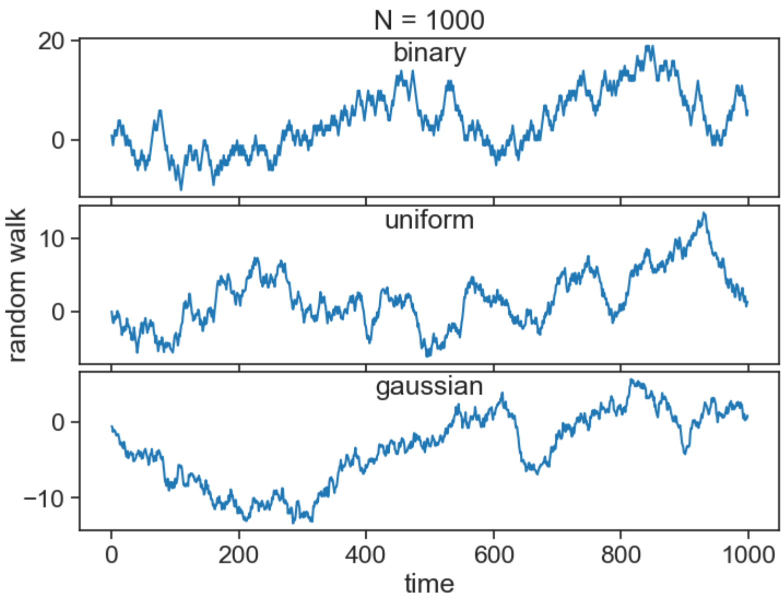
In the case of a binary process, you can think of the random walk as the position of a person who takes a step forward every time a coin toss yields heads, and a step backward for tails. Of course, by differencing the random walk, we can recover the original random sequence:

$$X_i = S_i - S_{i-1}.$$

See below the random walks associated with the three white noise processes from before:


N = 20
binary
uniform
gaussian
random walk
time
10
5
0
1
0
−1
5.0
2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


N = 100
binary
uniform
gaussian
random walk
time
5
0
0
−5
0
−5
0
20
40
60
80
100

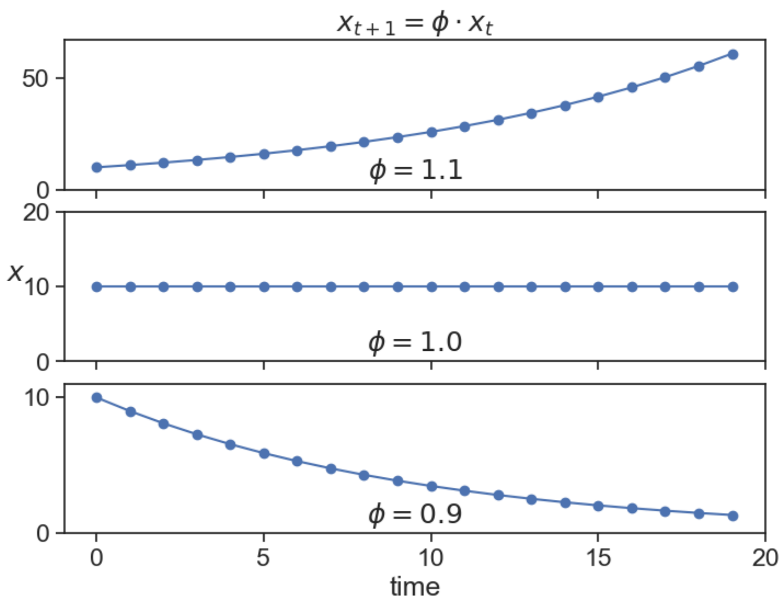

N = 1000
binary
uniform
gaussian
random walk
time
20
0
10
0
0
−10
0
200
400
600
800
1000

24 autoregressive processes

This lecture is partially based on Shumway and Stoffer (2017).

Consider the process

$$x_t = \phi \, x_{t-1}$$



These solutions look suspiciously similar to those obtained with the simplest 1st-order-homogenous differential equation:

$$\frac{dx}{dt} = \varphi \, x$$

whose solution is

$$x(t) = x_0 e^{\varphi\, t}$$

For the exponential solution, it is clear that $x(t)$ grows to infinity if $\varphi > 0$, and it goes to zero if $\varphi < 0$. This connection is not casual, they are the discrete and continuous versions of the same process. Starting from the continuous process

$$\frac{dx}{dt} = \varphi\, x$$

and using an approximation of the derivative as

$$\frac{dx}{dt} \simeq \frac{x(t+\Delta t) - x(t)}{\Delta t}$$

we have that

$$\frac{x(t+\Delta t) - x(t)}{\Delta t} = \varphi\, x(t).$$

Solving for $x(t + \Delta t)$ yields

$$x(t+\Delta t) = (1 + \Delta t\, \varphi) x(t).$$

Calling $\phi = 1 + \Delta t\, \varphi$ and modifying the notation a little bit gives

$$X_{t+1} = \phi\, X_t.$$

Using separation of variables:

$$\frac{dx}{x} = \varphi\, dt$$

We now integrate

$$\int \frac{1}{x}\, dy = \int \varphi\, dt$$

...yielding

$$\ln(x) = \varphi\, t + C$$

where C is an integration constant. Exponentiating both sides:

$$x(t) = e^{\varphi\, t} \cdot e^{C} = C e^{\varphi\, t}$$

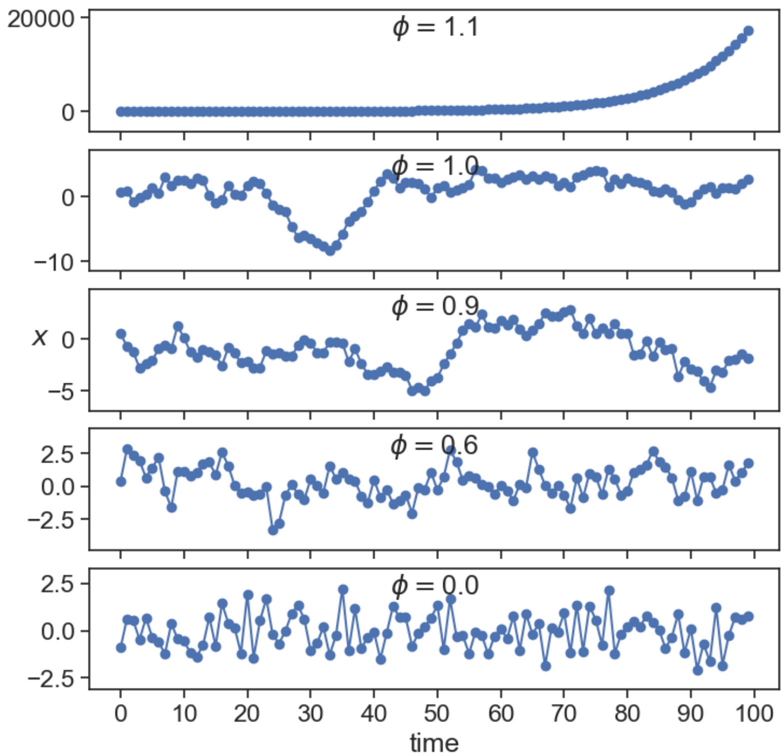
If we call the initial condition $x(0) = x_0$, we find that $C = x_0$, and we finally arrive at the solution.

24.1 AR(1)

Let's add some white noise (ε) to this process.

$$X_t = \phi\, X_{t-1} + \varepsilon.$$

This is called an Autoregressive Process of order 1, or AR(1). Here, the current value x_t is dependent on the immediately preceding value x_{t-1}.



What can we call these special cases?

- [1] $\phi = 1.0$
- [2] $\phi = 0.0$

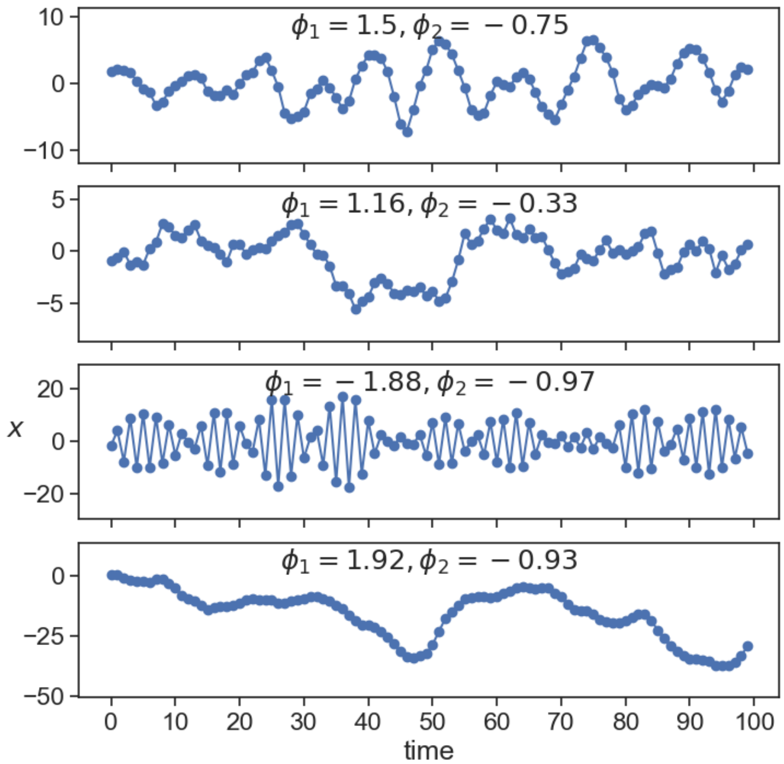
The time series clearly explodes to infinity if $\phi > 1$, and seems to stay bounded for values equal or smaller than 1. We will come back to this observation in a little while, when we discuss stationarity.

24.2 AR(2)

We can define a process that the current state is dependent on the two previous states, each with a different weight.

[1]random walk
[2]white noise

$$X_t = \phi_1 \, X_{t-1} + \phi_2 \, X_{t-2} + \varepsilon$$



24.3 AR(p)

The next thing to do is to generalize, and define an autoregressive process that depends on p previous states:

$$x_t = \phi_1 \, x_{t-1} + \phi_2 \, x_{t-2} + \cdots + \phi_p \, x_{t-p} + \varepsilon$$

25 autocorrelation

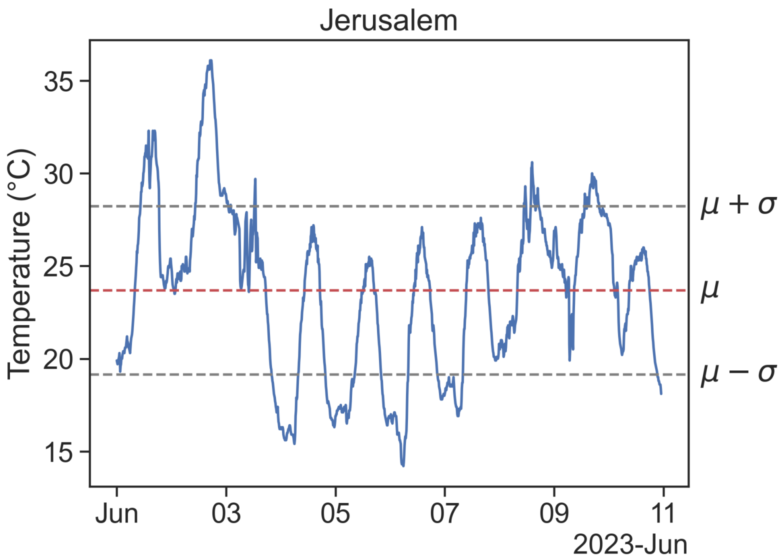
In this section, we will make use of a few fundamental concepts from statistics. Knowing these concepts well is fundamental to make sense of stationarity.

25.1 mean and standard deviation

Let's call our time series X, and its length N. Then:

$$\begin{aligned} \text{mean} \quad & \mu = \frac{\sum_{i=1}^{N} X_i}{N} \\ \text{standard deviation} \quad & \sigma = \sqrt{\frac{\sum_{i=1}^{N} (X_i - \mu)^2}{N}} \end{aligned}$$

The mean and standard deviation can be visualized thus:



25.2 expected value

The expected value (or expectation) of a variable X is given by

$$E[X] = \sum_{i=1}^{N} X_i p_i.$$

p_i is the weight or probability that X_i occurs. For a time series, the probability p_i that a given point X_i is in the dataset is simply $1/N$, therefore we can write the following measures in terms of expected values:

- mean, also called 1st moment:

$$\mu = E[X].$$

- variance, also called 2nd moment:

$$\sigma^2 = E[(X - E[X])^2] = E[(X - \mu)^2].$$

  Of course, σ is called the standard deviation.

25.3 covariance

The covariance between two time series X and Y is given by

$$\begin{aligned}\mathrm{cov}(X,Y) &= E[(X-E[X])(Y-E[Y])] \\ &= E[(X-\mu_X)(Y-\mu_Y)]\end{aligned}$$

Compare this to the definition of the variance, and it is obvious that the covariance cov(X,X) of a time series with itself is its variance.

25.4 correlation

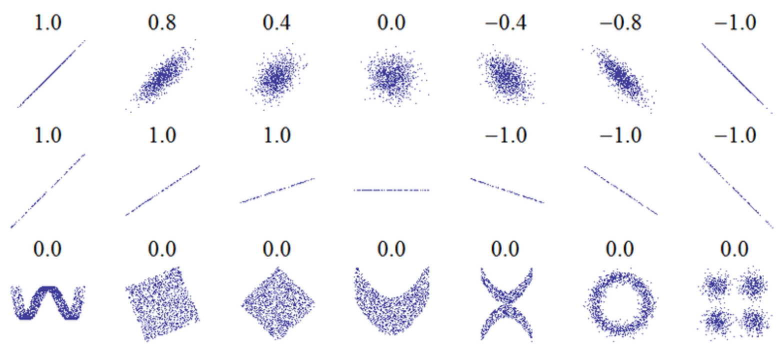
We are almost there. I promise.

The fact that cov(X,X) $= \sigma_X^2$ begs us to define a new measure, the correlation:

$$\mathrm{corr}(X,Y) = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}.$$

This is convenient, because now we can say that the correlation of a time series with itself is $\mathrm{corr}(X,X)=1$.

This is also called the Pearson correlation coefficient, and the result has a value between 1 and -1.



Source: Wikimedia

25.5 autocorrelation

The autocorrelation of a time series X is the answer to the following question:

> if we shift X by τ units, how similar will this be to the original signal?

In other words:

> how correlated are $X(t)$ and $X(t+\tau)$?

The autocorrelation is expressed as

$$\rho_{XX}(\tau) = \frac{E\left[(X_t - \mu)(X_{t+\tau} - \mu)\right]}{\sigma^2}$$

The autocorrelation function $\rho_{XX}(\tau)$ provides a useful measure of the degree of dependence among the values of a time series at different times.

In other disciplines, the autocorrelation is simply the autocovariance, i.e., it is not normalized by dividing by σ^2. In time series it is assumed that the autocorrelation is always normalized, therefore between -1 and 1.

A video is worth a billion words, so let's see the autocorrelation in action:

https://youtu.be/tpf-tuYHR5w

A few comments:

- The autocorrelation for $\tau = 0$ (zero shift) is always 1. [Can you prove this? All the necessary equations are above!]

26 stationarity

26.1 weak stationarity

A time series is called weakly stationary if the following conditions are met:

1. its mean μ does not vary in time:

   $$\mu_X(t) = \mu_X(t+\tau)$$

   for all values of t and τ.
2. its variance is finite for any time t:

   $$\sigma_X^2(t) < \infty.$$

3. The autocorrelation function between two lagged versions of the same time series, $X(t_1)$ and $X(t_2)$, depends only on the difference $\tau = t_2 - t_1$.

26.2 strict/strong stationarity

As the name suggests, strict stationarity requires much more than the above. It requires that the joint CDF of two lagged versions of X is the same. It is rare to require such strong terms, we will assume weak stationarity always from now on.

26.3 stationarity of AR(p)

Let's write once more the definition of AR(p):

$$x_t = \phi_1\, x_{t-1} + \phi_2\, x_{t-2} + \cdots + \phi_p\, x_{t-p} + \varepsilon$$

We can define the **backward shift operator** B as

$$BX_t = X_{t-1}$$

Of course, if B is applied p times, it shifts X thus:

$$B^p X_t = X_{t-p}.$$

With that in hand, we can rewrite the definition of AR(p) as follows:

$$x_t = \phi_1 \, B \, x_t + \phi_2 \, B^2 \, x_t + \cdots + \phi_p \, B^p \, x_t + \varepsilon$$

We have many terms with x_t, so let's group them on the left-hand side:

$$x_t \, \phi(B) = \varepsilon,$$

where the **characteristic polynomial** $\phi(B)$ is

$$\phi(B) = 1 - \phi_1 \, B - \phi_2 \, B^2 - \cdots - \phi_p \, B^p$$

In order to determine the stationarity of AR(p), we need to find the roots of

$$\phi(B) = 0,$$

called characteristic roots. These roots are often complex numbers.

AR(p) is stationary if ALL the characteristic roots lie OUTSIDE the unit circle.

The reason for this is not obvious. A nice explanation can be found in this StackExchange response. Another good text is Tsay (2010, chap. 2).

Brockwell and Davis (2016, chap. 2 p. 49 and chapter 3 p. 75) explain that, strictly speaking, an AR(p) process is stationary as long as the roots do not lie on the unit circle. However, in the case that the roots lie **inside** the unit circle, the process is **noncausal**, meaning that the present state depends on future states. In reality, everyone just ignores this point, and simply say that we require the roots to lie outside the unit circle to guarantee stationarity.

26.4 stationarity of AR(1)

We need to solve the roots of the characteristic equation

$$1 - \phi_1 B = 0.$$

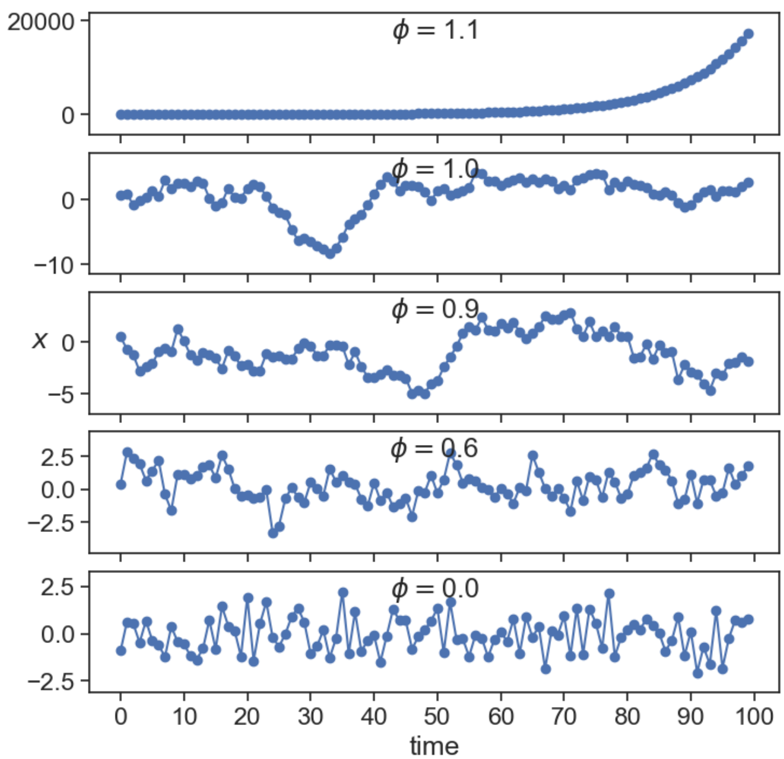
The only root is

$$B = \frac{1}{\phi_1}$$

Because we require that this root is greater than 1 (in absolute value), we have that:

$$\left|\frac{1}{\phi_1}\right| > 1 \quad \longrightarrow \quad |\phi| < 1.$$

This result corroborates our conclusion from before:

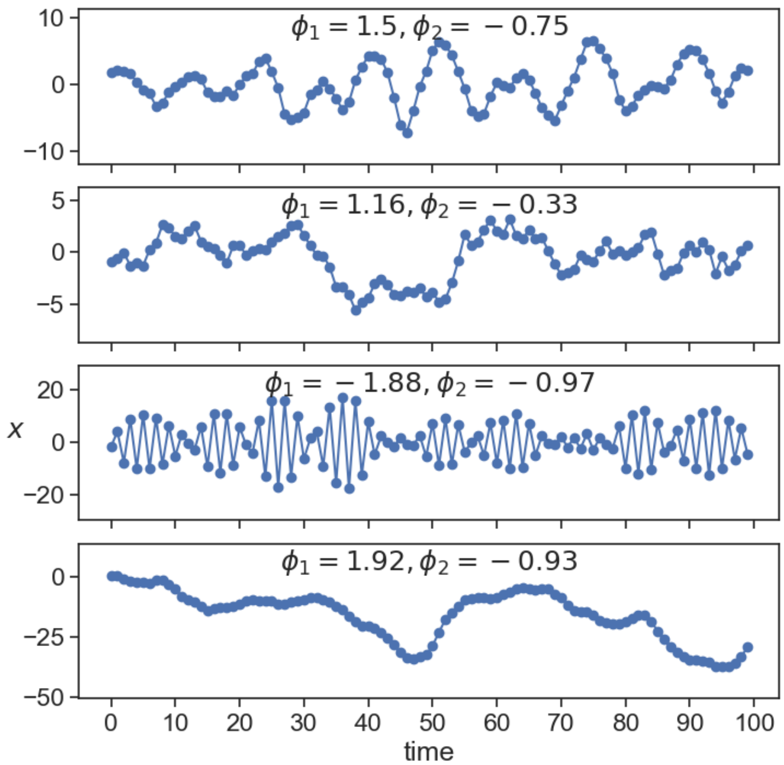


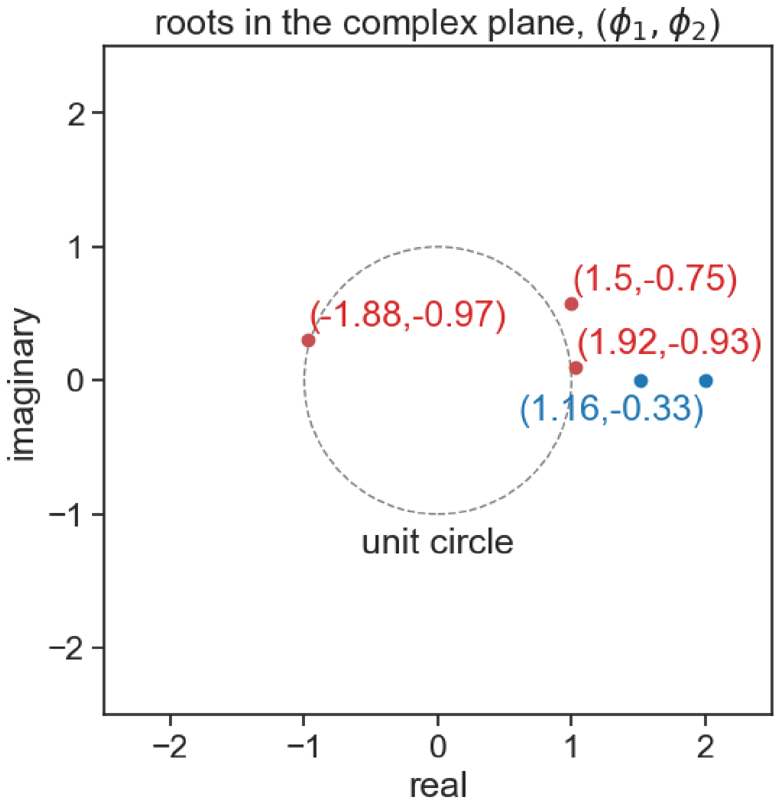
26.5 stationarity of AR(2)

We need to solve the roots of the characteristic equation

$$1 - \phi_1 B - \phi_2 B^2 = 0.$$

For the AR(2) time series from before, here are the roots of the characteristic polynomial plotten on the complex plane:





Because complex roots come in pairs, I plotted above only one of the roots, the one with positive imaginary component. The second panel ($\phi_1 = 1.16, \phi_2 = -0.33$) has two real roots, so both are plotted in blue.

The AR(2) time series will have a periodic component if the roots are complex, and the frequency of oscillation is

$$f_0 = \frac{1}{2\pi} \cos^{-1}\left(\frac{\phi_1}{2\sqrt{(-\phi_2)}}\right)$$

Play with the widget below to get a feel of how the complex roots depend on the values of ϕ_1 and ϕ_2. In the left panel you can move the point A to choose different ϕ values, and on the right you see the complex roots of the polynomial instantly updated. As long as the point A is inside the blue triangle, the roots will be outside the unit circle, and therefore the process

See an interesting discussion in David Josephs' excellent time series webpage will be stationary. For ϕ_2 values above (below) the red line, the complex roots will be real (complex conjugates).

Conversely, play with the position of one of the complex roots, and see how this influences the value of ϕ_1, ϕ_2.

Geogebra app made by Yair Mau (2024)

The two roots are easy to find from ϕ_1, ϕ_2, you just need to solve

$$1 - \phi_1 B - \phi_2 B^2 = 0$$

for B. What if you choose two roots z_1, z_2 and want to derive from them the value of ϕ_1, ϕ_2? Just use this:

$$\phi_1 = \frac{z_1 + z_2}{z_1 \cdot z_2}$$

$$\phi_2 = -\frac{1}{z_1 \cdot z_2}$$

26.6 stationarity of AR(4)

Conceptually, this is exactly like AR(2). In case you ever need to choose AR(4) ϕ values, play with the widget below and see if you can put all the roots outside the unit circle.

27 ACF and PACF graphs

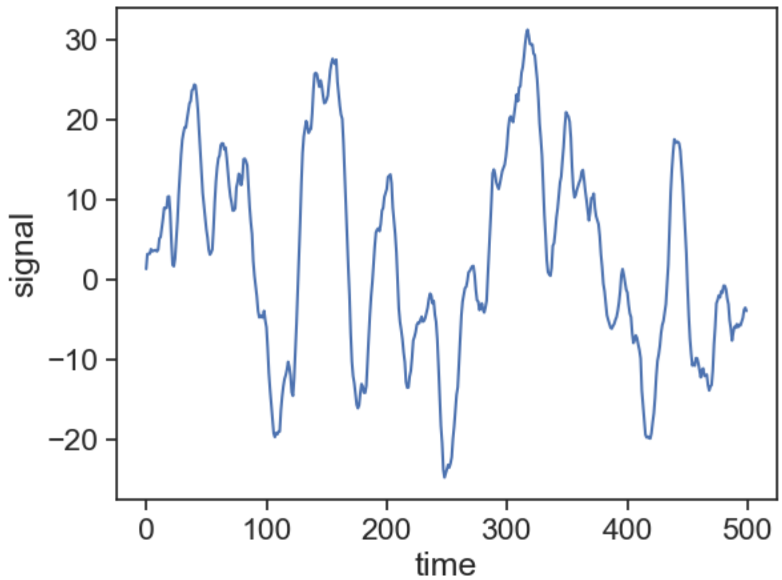
We will now see what is the connection between stationarity and autocorrelation.

Using Python's `statsmodels` package, let's create an AR time series and plot it.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import pandas as pd
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
import statsmodels.api as sm
from statsmodels.tsa.arima_process import ArmaProcess
```

```
phi1 = 1.86
phi2 = -0.87
N = 500
np.random.seed(10)  # For reproducibility
ar2_process = ArmaProcess(ar=[1, -phi1, -phi2], ma=[1])
ar2_values = ar2_process.generate_sample(nsample=N)
```

```
fig, ax = plt.subplots()
ax.plot(ar2_values)
ax.set(xlabel="time",
       ylabel="signal");
```



27.1 ACF

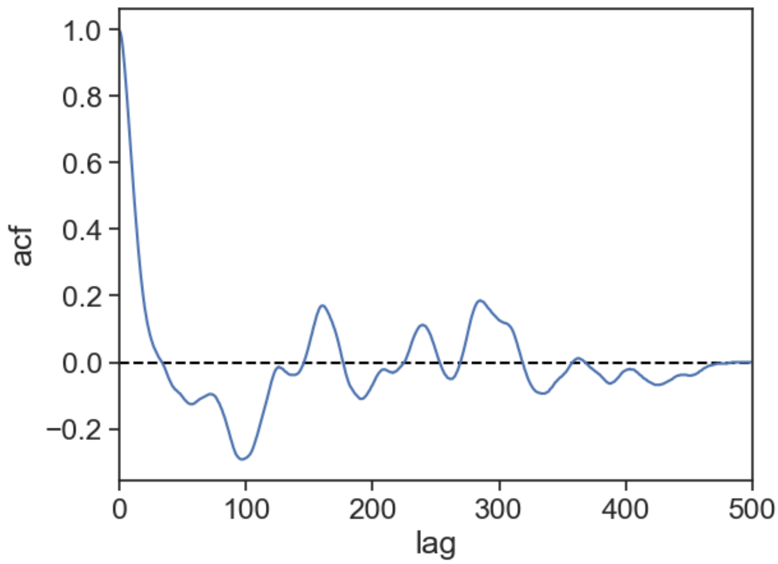
We need to calculate now the autocorrelation function of our series:

$$\rho_{XX}(\tau) = \frac{E\left[(X_t - \mu)(X_{t+\tau} - \mu)\right]}{\sigma^2}$$

```
def compute_acf(series):
    N = len(series)
    lags = np.arange(N)
    acf = np.zeros_like(lags)
    series = (series - series.mean()) / series.std()
    for i in lags:
        acf[i] = np.sum(series[i:] * series[:N-i])
    acf = acf / N
    return lags, acf
```

```
fig, ax = plt.subplots()
lags, acf = compute_acf(ar2_values)
ax.plot([0, N], [0]*2, color="black", ls="--")
```

```
ax.plot(lags, acf)
ax.set(xlabel="lag",
       ylabel="acf",
       xlim=[0, N]);
```

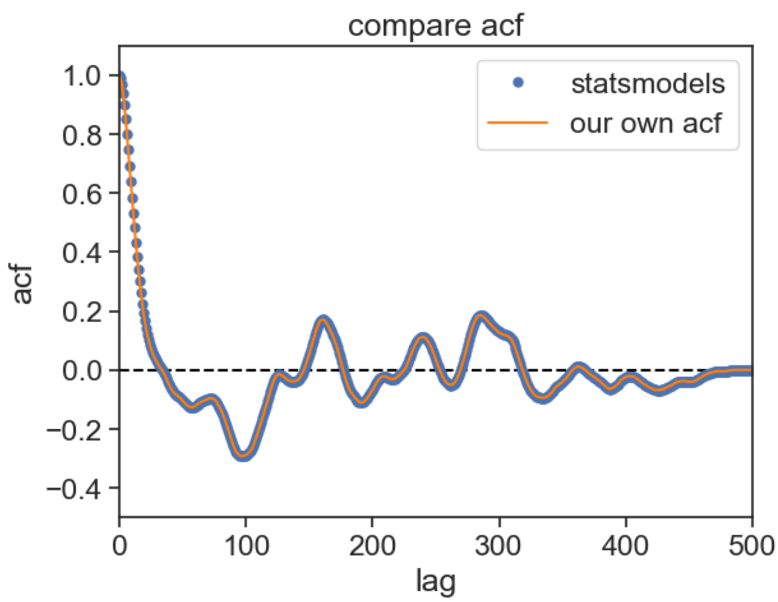


Notice that the ACF always starts at 1 for zero lag, and it gets closer to zero as the lag increases.

- **Intuitive interpretation**: Two measurements taken within a short time interval (lag) should be similar, therefore their correlation is expected to be high. As we compare measurements from increasing time intervals, they are less and less similar to one another, therefore their correlation goes down.
- **Mathematical interpretation**: Take a look at the code we wrote above. As the lags grows, the length of both arrays keeps shrinking, but we still divide the result by N. The logical conclusion is that when $\tau = N$ the ACF will be exactly zero.

The `statsmodels` package also offers an easy way to plot the ACF, let's compare our calculation with the built-in function:

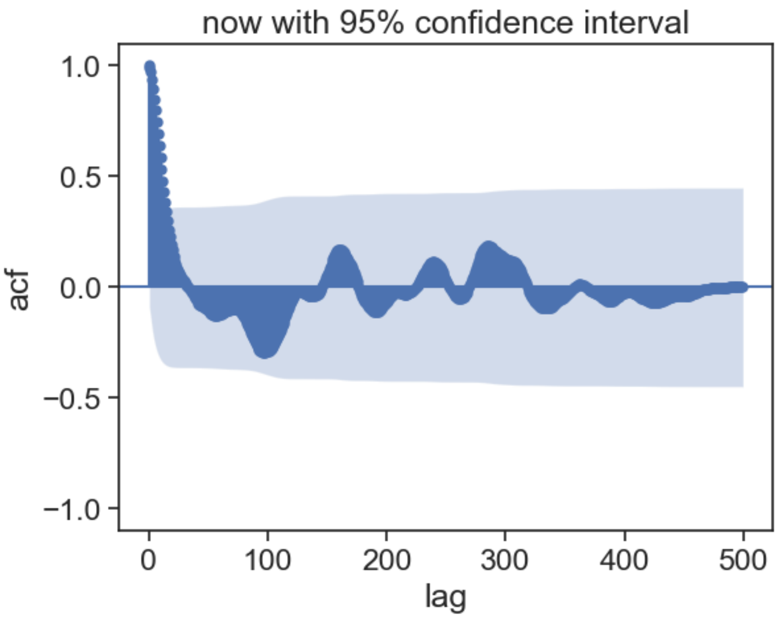
```
fig, ax = plt.subplots()
ax.plot([0, N], [0]*2, color="black", ls="--")
sm.graphics.tsa.plot_acf(ar2_values, lags= N-1, ax=ax, label="statsmodels", alpha=None, use_
ax.plot(lags, acf, color="tab:orange", label="our own acf")
ax.legend()
ax.set(ylim=[-0.5, 1.1],
       xlim=[0, N],
       title="compare acf",
       xlabel="lag",
       ylabel="acf");
```



Excellent! From now on we will continue using `statsmodels` functions. We can spice up the ACF graph, by showing an envelope of 95% confidence interval.

```
fig, ax = plt.subplots()
sm.graphics.tsa.plot_acf(ar2_values, lags= N-1, ax=ax, alpha=.05)
ax.set(ylim=[-1.1, 1.1],
       title="now confidence interval")
ax.set(ylim=[-1.1, 1.1],
       title="now with 95% confidence interval",
```

```
        xlabel="lag",
        ylabel="acf");
```



If an autocorrelation value at a specific lag falls outside the confidence interval, it suggests that the autocorrelation at that lag is statistically significant. In other words, there is evidence of correlation at that lag. If an autocorrelation value is within the confidence interval, it suggests that the autocorrelation at that lag is not statistically significant, and any observed correlation might be due to random noise. The width of the confidence interval is influenced by the significance level. For a 95% confidence interval, it means that you are 95% confident that the true autocorrelation lies within the interval. If you choose a higher confidence level, the interval will become wider, making it harder to reject the null hypothesis of no correlation.

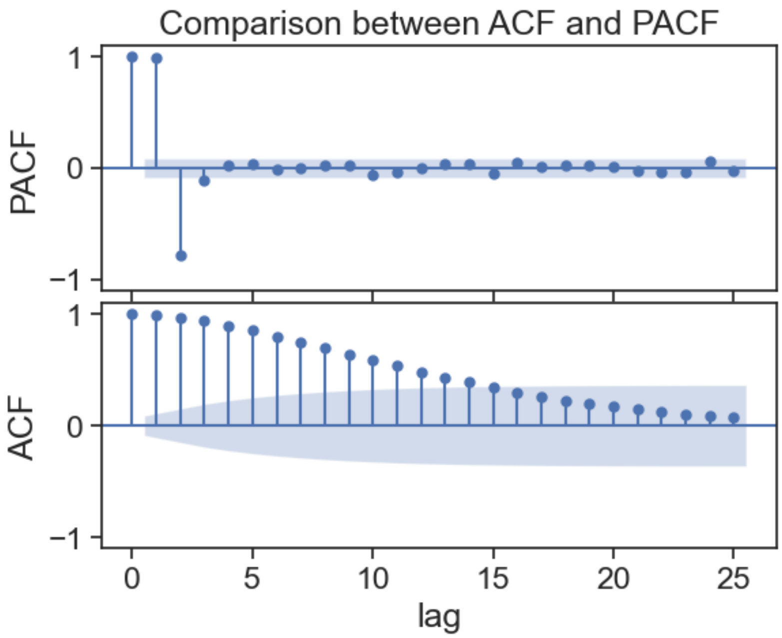
27.1.1 problem?

There is something a bit troubling about the ACF graph. We can learn from it how fast the correlation between two points in time falls, but this analysis is not too clean. Assume that the present state x_t is only dependent on one time step back, x_{t-1}. Because x_{t-1} is dependent on x_{t-2}, the result is that we will find that x_t is weakly dependent on x_{t-2}, although the direct dependence is zero.

The Partial ACF (PACF) solves this problem. It removes the intermediate effects between two points, and returns only the direct influence of one time instant on another one lagged by τ. Let's see how it looks like for the proceess above.

27.2 PACF

```
fig, (ax1, ax2) = plt.subplots(2,1)
fig.subplots_adjust(hspace=0.05)
sm.graphics.tsa.plot_pacf(ar2_values, lags=25, ax=ax1, alpha=.05)
ax1.set(ylim=[-1.1, 1.1],
       title="now confidence interval")
ax1.set(ylim=[-1.1, 1.1],
       title="Comparison between ACF and PACF",
       xlabel="lag",
       ylabel="PACF");

sm.graphics.tsa.plot_acf(ar2_values, lags= 25, ax=ax2, alpha=.05, title=None)
ax2.set(ylim=[-1.1, 1.1],
       xlabel="lag",
       ylabel="ACF");
```



We see three bars significantly far from the confidence interval. The leftmost shows PACF($\tau = 0$) = 1, which is expected, so let's not discuss it. The two next bars are the really important ones, they show the greatest correlation. From then on, the correlation for lags greater than 2 is not significant. With PACF's help, we can infer that the original AR processes must have been of order 2.

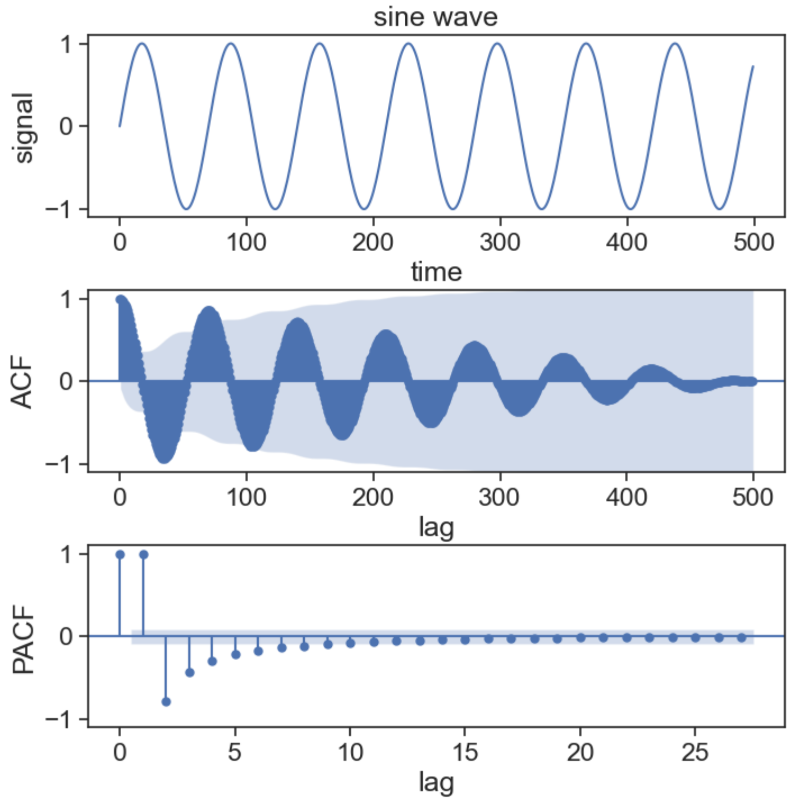
27.3 discussion

What can we say about the following series?

27.3.1 sine wave

```
N = 500
time = np.arange(N)
period = 70
omega = 2.0 * np.pi / period
```

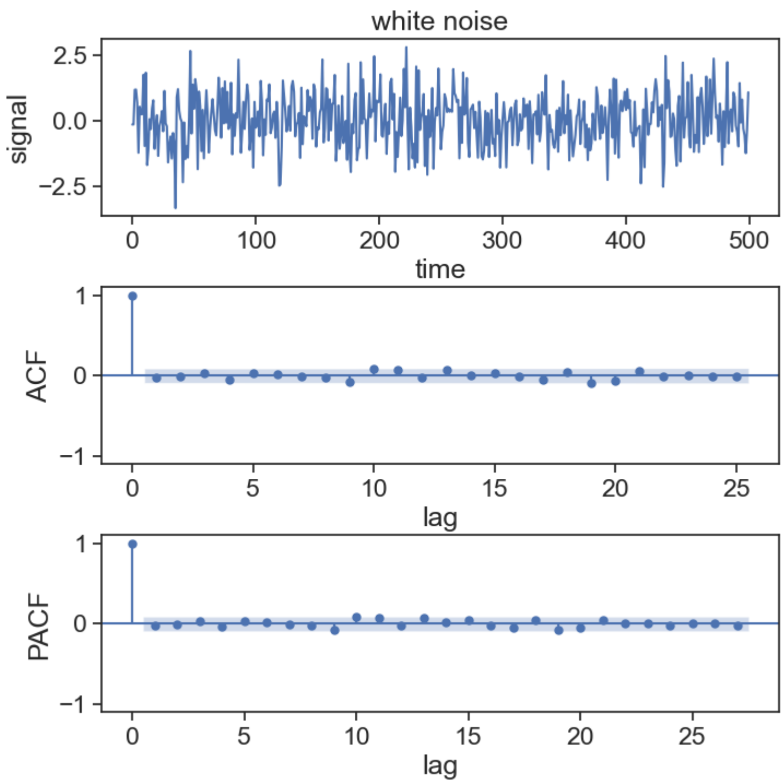
```
signal = np.sin(omega * time)

fig, (ax1, ax2, ax3) = plt.subplots(3,1, figsize=(8,8))
fig.subplots_adjust(hspace=0.4)

ax1.plot(time, signal)
ax1.set(ylabel="signal",
        title="sine wave",
        xlabel="time")

sm.graphics.tsa.plot_acf(signal, ax=ax2, alpha=.05, title=None, lags=N-1)
ax2.set(ylim=[-1.1, 1.1],
       xlabel="lag",
       ylabel="ACF");

sm.graphics.tsa.plot_pacf(signal, ax=ax3, alpha=.05, title=None)
ax3.set(ylim=[-1.1, 1.1],
       ylabel="PACF",
       xlabel="lag");
```



27.3.2 white noise

```
N = 500
time = np.arange(N)
signal = np.random.normal(size=N)

fig, (ax1, ax2, ax3) = plt.subplots(3,1, figsize=(8,8))
fig.subplots_adjust(hspace=0.4)

ax1.plot(time, signal)
ax1.set(ylabel="signal",
        title="white noise",
        xlabel="time")

sm.graphics.tsa.plot_acf(signal, ax=ax2, alpha=.05, title=None, lags=25)
```

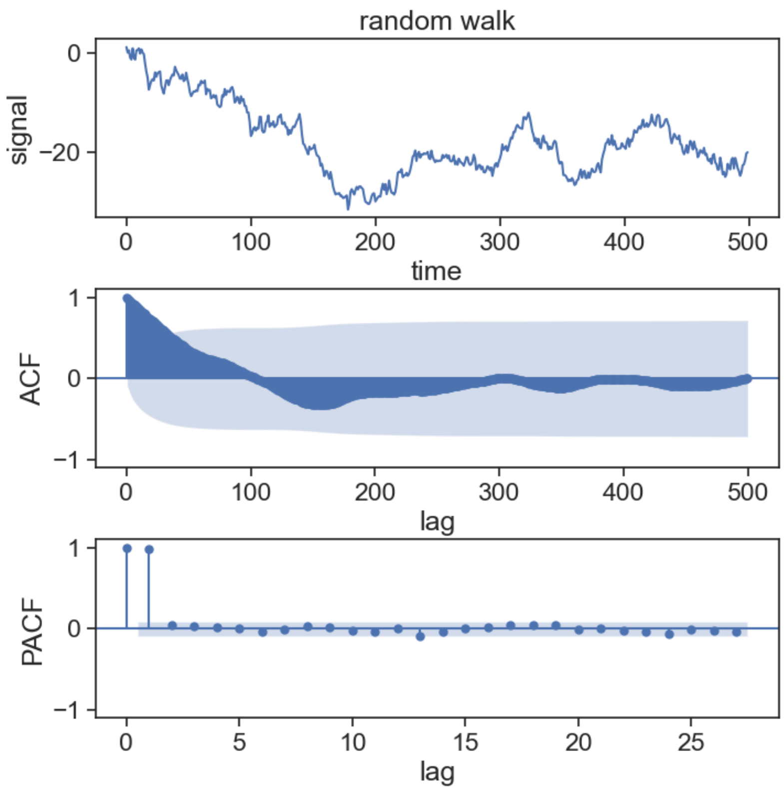
```
ax2.set(ylim=[-1.1, 1.1],
        xlabel="lag",
        ylabel="ACF");

sm.graphics.tsa.plot_pacf(signal, ax=ax3, alpha=.05, title=None)
ax3.set(ylim=[-1.1, 1.1],
        ylabel="PACF",
        xlabel="lag");
```



27.3.3 random walk

```
N = 500
time = np.arange(N)
signal = np.cumsum(np.random.normal(size=N))
```

```
fig, (ax1, ax2, ax3) = plt.subplots(3,1, figsize=(8,8))
fig.subplots_adjust(hspace=0.4)

ax1.plot(time, signal)
ax1.set(ylabel="signal",
        title="random walk",
        xlabel="time")

sm.graphics.tsa.plot_acf(signal, ax=ax2, alpha=.05, title=None, lags=N-1)
ax2.set(ylim=[-1.1, 1.1],
       xlabel="lag",
       ylabel="ACF");

sm.graphics.tsa.plot_pacf(signal, ax=ax3, alpha=.05, title=None)
ax3.set(ylim=[-1.1, 1.1],
       ylabel="PACF",
       xlabel="lag");
```



28 from AR to ARIMA

Up to this point, we learned what an AR process is, and how it relates to the concept of stationarity.

Our long term goal is to use these concepts to make forecasts (predictions) about the future. Before we do that, it is useful to talk about a generalization of the AR process, that better resembles real-life data.

28.1 AR(p)

An **autoregressive** process X is one that depend on p past states:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + \varepsilon$$

From what we already learned, if the complex roots of the polynomial

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p$$

lie **outside** the unit circle, then the AR process is **causal and stationary**.

28.2 MA(q)

Similarly, a **moving average** process X is one that depend on q past noise steps:

$$X_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_p \varepsilon_{t-q}$$

This has nothing to do with the sliding averages used for smoothing we studied before, it's just the same name for a different concept.

Note that this equation is identical in structure to that of AR(p), but with weights θ standing for ϕ, and past noise ε_{t-i} standing in for past states X_{t-i}.

This process also has its characteristic polynomial:

$$\theta(B) = 1 - \theta_1\, B - \theta_2\, B^2 - \cdots - \theta_p\, B^p$$

The complex roots of this polynomial are also important. As long as the roots are **outside** the unit circle, the MA(q) process will be considered **invertible**, which is to say that it can be transformed into an AR(∞) process.

The story is of course more complex than that. Using intelligent mathematical tricks (substitutions), one can change the noise term to make roots move from inside the unit circle to the outside, so effectively there shouldn't be any problems as long as there aren't any roots exacly on the unit circle.

28.3 ARMA(p,q)

An ARMA(p,q) process is simply the combination of an AR(p) and an MA(q) process:

$$\begin{aligned} X_t = {} & \phi_1\, X_{t-1} + \phi_2\, X_{t-2} + \cdots + \phi_p\, X_{t-p} \\ & + \varepsilon_t + \theta_1\, \varepsilon_{t-1} + \theta_2\, \varepsilon_{t-2} + \cdots + \theta_p\, \varepsilon_{t-p} \end{aligned}$$

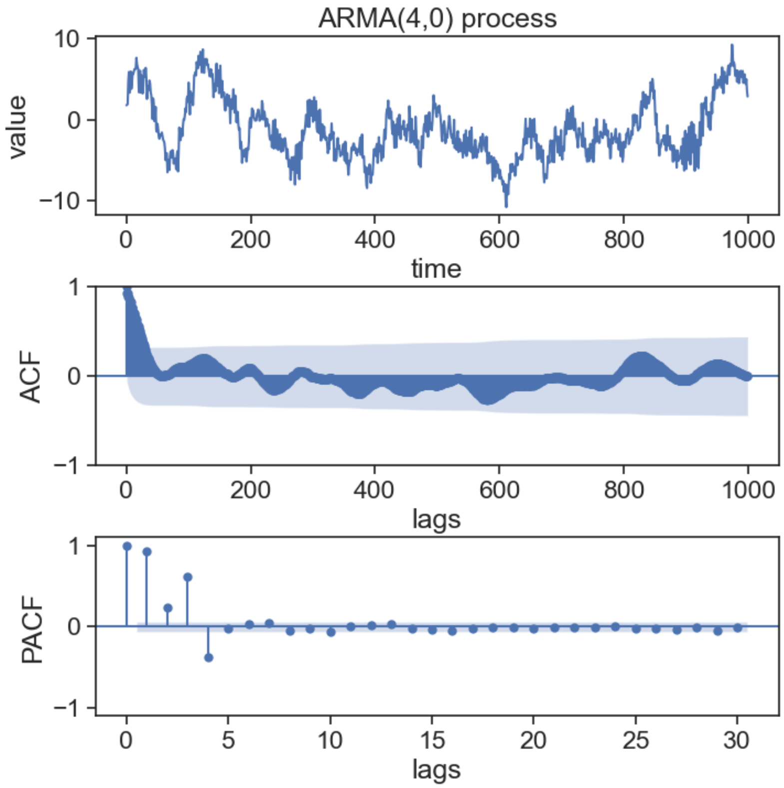
28.4 ACF and PACF

The graphs for the autocorrelation and partial autocorrrelation functions can be very useful to identify the order p and q of an ARMA(p,q) process.

```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
```

```python
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
import statsmodels.api as sm
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.tsa.stattools import adfuller
```

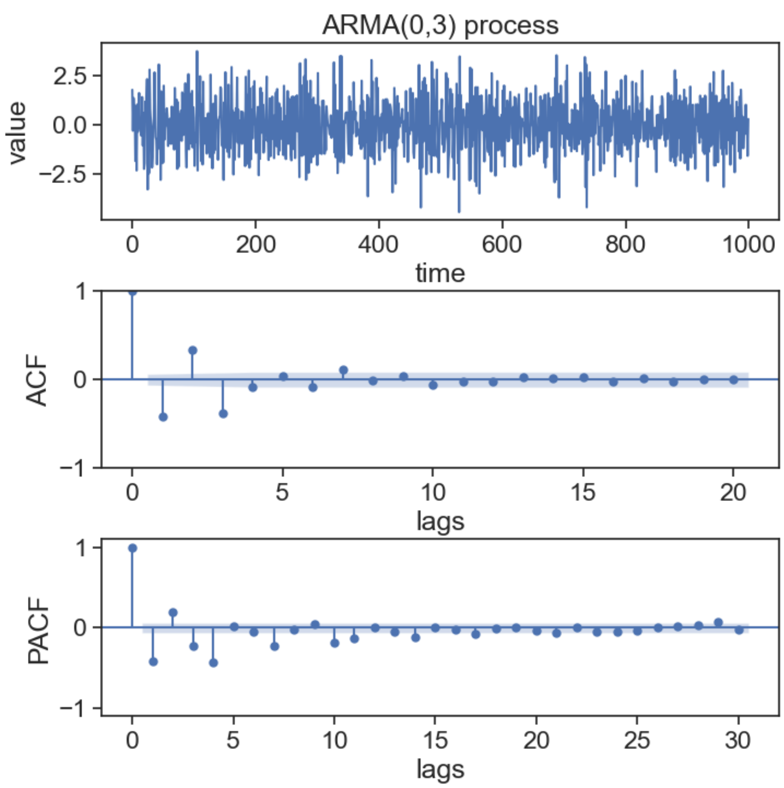
```python
np.random.seed(0)
n = 1000  # number of data points
phi_list = np.array([0.8, -0.28, 0.8, -0.36])
ar_coefs = np.insert(-phi_list, 0, 1)  # AR coefficients. append 1 at the beginning
ma_coefs = [1] # MA coefficients

arma_process = ArmaProcess(ar_coefs, ma_coefs)
data = arma_process.generate_sample(nsample=n)

fig, axes = plt.subplots(3, 1, figsize=(8, 8))
fig.subplots_adjust(hspace=0.4)  # increase vertical space between panels

ax1 = axes[0]
ax1.plot(data)
ax1.set(xlabel='time',
        ylabel='value',
        title='ARMA(4,0) process',
       )

# plot ACF and PACF graphs
ax2 = axes[1]
sm.graphics.tsa.plot_acf(data, lags=n-1, ax=ax2, title=None)
ax2.set(ylabel="ACF",
        xlabel="lags")

ax3 = axes[2]
sm.graphics.tsa.plot_pacf(data, lags=30, ax=ax3, title=None)
ax3.set(ylim=[-1.1, 1.1],
        ylabel='PACF',
        xlabel="lags");
```



Note that for the ARMA(4,0) process, the last significant PACF value is at lag $\tau = 4$.

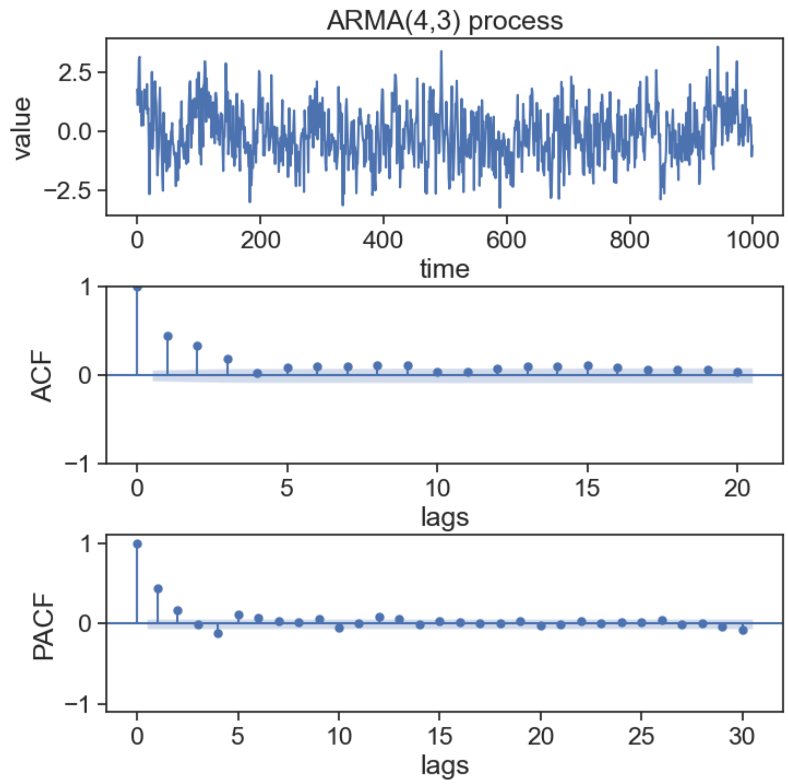
```
np.random.seed(0)
n = 1000  # number of data points
theta_list = np.array([0.4, -0.3, 0.8])
ma_coefs = np.insert(-theta_list, 0, 1)  # MA coefficients. append 1 at the beginning
ar_coefs = [1]

arma_process = ArmaProcess(ar_coefs, ma_coefs)
data = arma_process.generate_sample(nsample=n)

fig, axes = plt.subplots(3, 1, figsize=(8, 8))
fig.subplots_adjust(hspace=0.4)

ax1 = axes[0]
```

```
ax1.plot(data)
ax1.set(xlabel='time',
        ylabel='value',
        title='ARMA(0,3) process',
       )

# plot ACF and PACF graphs
ax2 = axes[1]
sm.graphics.tsa.plot_acf(data, lags=20, ax=ax2, title=None)
ax2.set(ylabel="ACF",
        xlabel="lags")

ax3 = axes[2]
sm.graphics.tsa.plot_pacf(data, lags=30, ax=ax3, title=None)
ax3.set(ylim=[-1.1, 1.1],
        ylabel='PACF',
        xlabel="lags");
```



For the ARMA(0,3) process, the last significant ACF value is at lag $\tau = 3$.

```
np.random.seed(0)
n = 1000  # number of data points
theta_list = np.array([0.4, -0.3, 0.8])
phi_list = np.array([0.8, -0.28, 0.8, -0.36])
ar_coefs = np.insert(-phi_list, 0, 1)  # AR coefficients
ma_coefs = np.insert(-theta_list, 0, 1)  # MA coefficients

arma_process = ArmaProcess(ar_coefs, ma_coefs)
data = arma_process.generate_sample(nsample=n)

# Create a single figure with panels
fig, axes = plt.subplots(3, 1, figsize=(8, 8))
fig.subplots_adjust(hspace=0.4)


# Plot the ARMA process
ax1 = axes[0]
ax1.plot(data)
ax1.set(xlabel='time',
        ylabel='value',
        title='ARMA(4,3) process',
       )

# Plot ACF and PACF graphs
ax2 = axes[1]
sm.graphics.tsa.plot_acf(data, lags=20, ax=ax2, title=None)
ax2.set(ylabel="ACF",
        xlabel="lags")

ax3 = axes[2]
sm.graphics.tsa.plot_pacf(data, lags=30, ax=ax3, title=None)
ax3.set(ylim=[-1.1, 1.1],
        ylabel='PACF',
        xlabel="lags");
```



This table from Shumway and Stoffer (2017, 108) is useful to sum up what we've learned so far.

	AR(p)	MA(q)	ARMA(p,q)
ACF	gradually goes down	cuts off after lag q	gradually goes down
PACF	cuts off after lag p	gradually goes down	gradually goes down

28.5 Non-stationary data and ADF test

The following is partially based on Chatfield (2016, chap. 3, page 63).

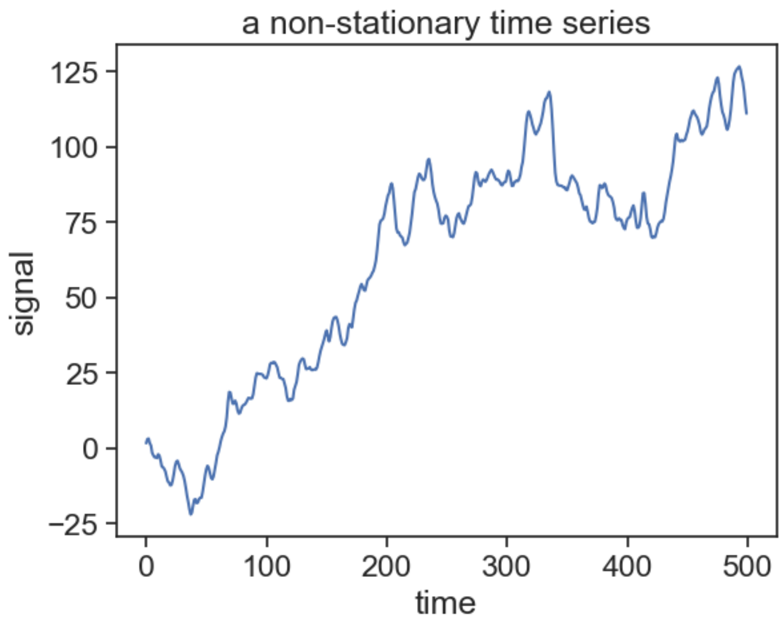
What do we do if it turns out that our data is not stationary? Heck, how can we even tell if our data is stationary or not? The most common stationarity test is the Augmented Dickey–Fuller (ADF) test. This is not a trivial subject that can be completely understood in a few words, so I'll give the very basic intuition here.

A stationary time series has a constant mean μ. If at a given instant t our state X_t is way above the mean, we would expect that, with a high probability, the next step brings it closer to the mean. This is to say that the the difference between two consecutive states $X_t - X_{t-1}$ depends on the value of X_t! Nonstationary time series do not show this behavior: the differences between two time steps do not depend on the state value. The idea described here is for the Dickey-Fuller test. The Augmented Dickey-Fuller test is basically the same, but for time lags p between states, not only 1.

The ADF test has a null hypothesis that the time series is **not stationary**. By applying the test to a given time series, we get a p-value as one of the results. The smaller the p-value, the more evidence we have to reject the null hypothesis, and therefore conclude that our time series is indeed stationary.

Let's see an example:

```
# Generate ARIMA(1,1,2) process with differencing
N = 500
np.random.seed(1)
arima_112_diff = sm.tsa.arma_generate_sample(ar=[1, -0.5], ma=[1, 0.7, 0.3], nsample=N)
arima_112 = np.cumsum(arima_112_diff)

fig, ax = plt.subplots()
ax.plot(arima_112)
ax.set(xlabel="time",
       ylabel="signal",
       title="a non-stationary time series")
plt.show()

result = adfuller(arima_112)
print('p-value: ', result[1])
```




```
p-value:  0.591478751185507
```

So what do we do if we have a non-stationary time series? One common solution is to apply successive **differencing** operations, until the outcome becomes stationary.

Let's define the difference operator ∇ as

$$\nabla X_t = X_t - X_{t-1}.$$

Now recalling that the backward shift operator B is defined as

$$BX_t = X_{t-1},$$

we can rewrite the difference operator as

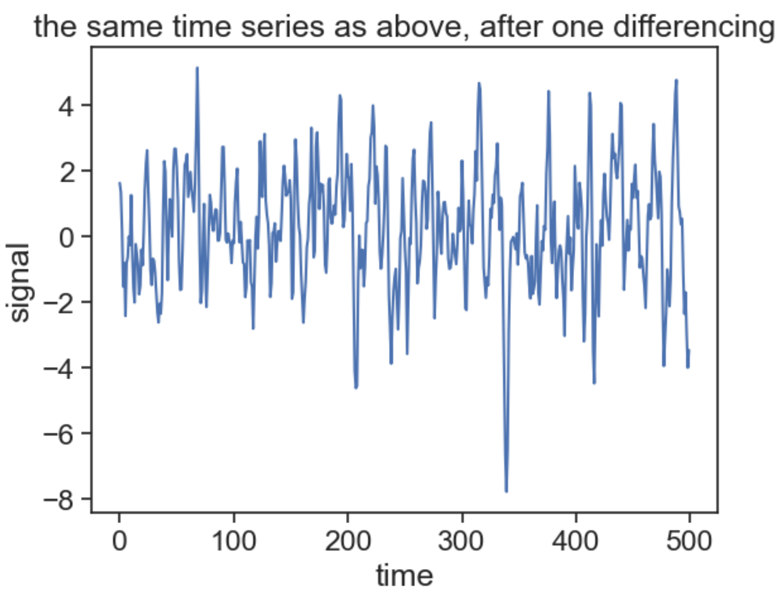
$$\begin{aligned}\nabla X_t &= X_t - BX_t \\ &= (1 - B)X_t.\end{aligned}$$

If we apply the difference operator d times, then we denote this as

$$W_t = \nabla^d X_t = (1 - B)^d X_t.$$

Let's apply the difference operator once to the time series plotted above, and then apply the ADF test.

```
fig, ax = plt.subplots()
ax.plot(arima_112_diff)
ax.set(xlabel="time",
       ylabel="signal",
       title="the same time series as above, after one differencing")
plt.show()

result = adfuller(arima_112_diff)
print('p-value: ', result[1])
```




```
p-value:  4.7341140554650393e-14
```

28.6 ARIMA(p,d,q)

We are ready to describe an Autoregressive (AR) Integrated (I) Moving Average (MA) process:

$$
\begin{aligned}
W_t = {} & \phi_1 W_{t-1} + \phi_2 W_{t-2} + \cdots \phi_q W_{t-q} \\
& + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots \theta_q \varepsilon_{t-q}
\end{aligned}
$$

Rearranging the terms in this equation, we can also express an ARIMA(p,d,q) process as

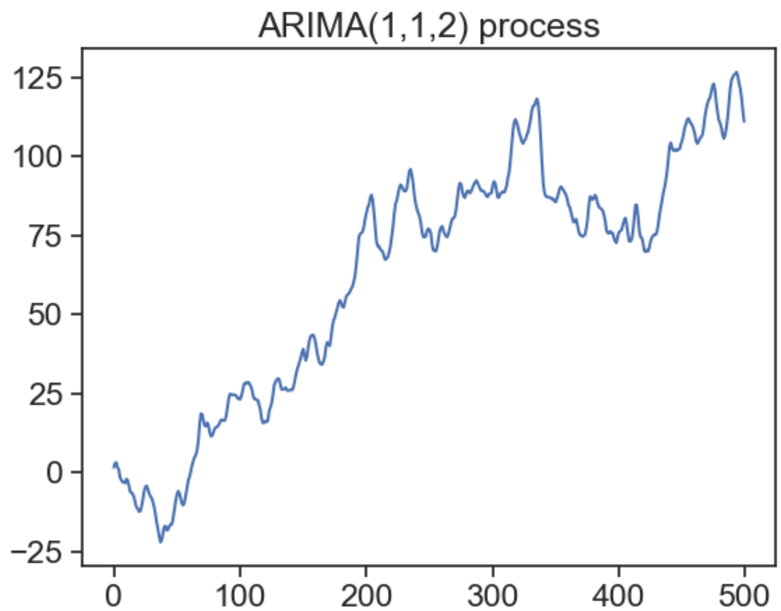
$$\phi(B)(1-B)^d X_t = \theta(B)\epsilon_t.$$

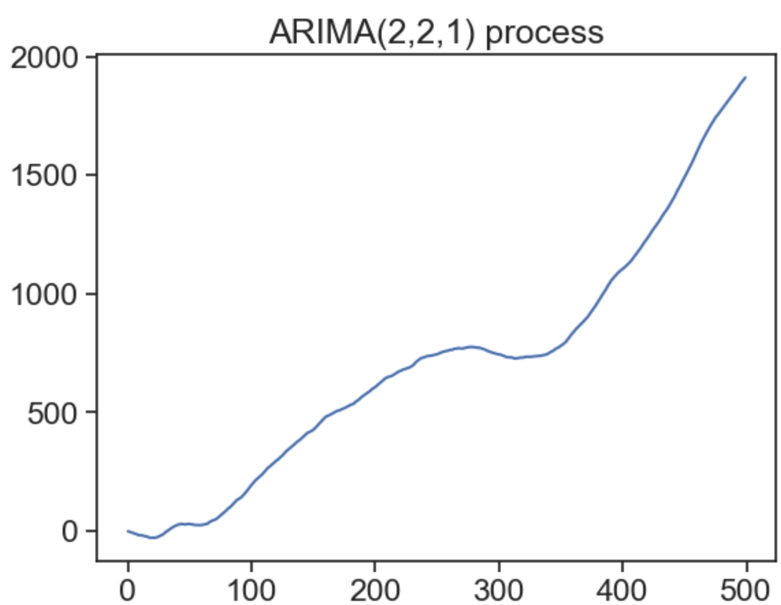
Let's try to put this in a context we already know. We saw that a random walk is the integrated version of a white noise. The random walk can be interpreted as a special case of an AR(1) process for $\phi = 1$. However, an AR process is usually called as such when it is stationary. Because a white noise can be understood as an ARMA(0,0) process, and because differencing the random walk yields a white noise, we can say that the white noise is an ARIMA(0,1,0) process.

Let's see two examples of ARIMA processes.

```
# Generate ARIMA(2,2,1) process with differencing
arima_221_diff = sm.tsa.arma_generate_sample(ar=[1, -0.18, 0.06], ma=[1, -0.5], nsample=N)
# arima_221 = np.cumsum(arima_221_diff)
arima_221 = np.cumsum(np.cumsum(arima_221_diff)) #

# Plot the ARIMA(1,1,2) process
fig, ax = plt.subplots()
ax.plot(arima_112)
ax.set_title('ARIMA(1,1,2) process')
plt.show()

# Plot the ARIMA(2,2,1) process
fig, ax = plt.subplots()
```

```
ax.plot(arima_221)
ax.set_title('ARIMA(2,2,1) process')
plt.show()
```





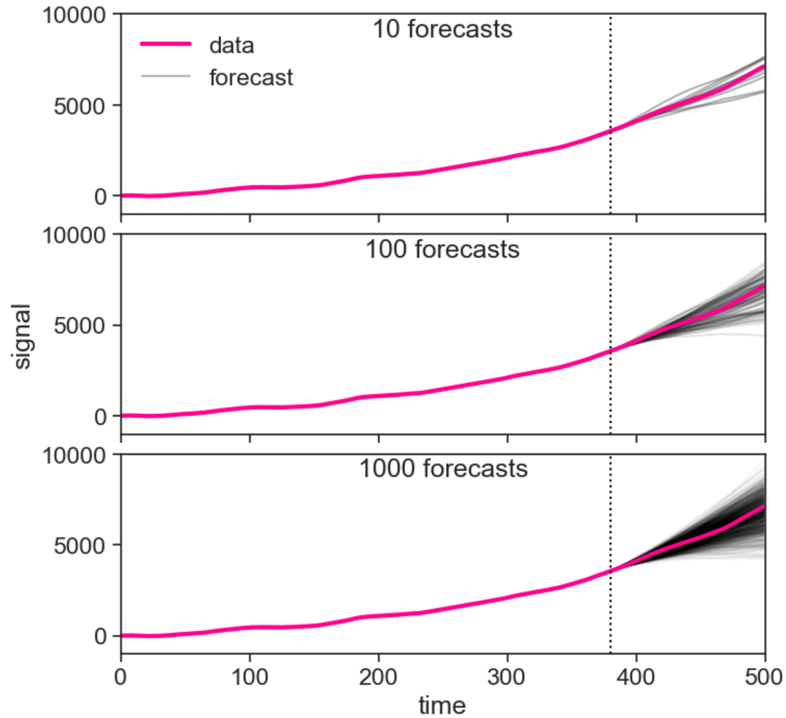
We will not fully delve into forecasting right now, but it would be nice to see a real application of ARIMA. If we can reasonably well estimate the parameters associated with a given ARIMA(p,d,q) process, we can use this knowledge to predict future states within a confidence interval. In the simulations below, we see forecasts an ARIMA(2,2,0) process.

```
def arima_forecast(series, ar_coeff):
    s = series.copy()
    phi1 = -ar_coeff[1]
    phi2 = -ar_coeff[2]
    start_index = np.argmax(np.isnan(s))
    for i in np.arange(start_index,len(series)):
        s[i] = phi1 * s[i-1] + phi2 * s[i-2] + np.random.normal()
    return s

np.random.seed(1998)
arima_220_diff = sm.tsa.arma_generate_sample(ar=[1, -0.18, 0.06], ma=[1], nsample=N)
arima_220 = np.cumsum(np.cumsum(arima_220_diff)) #


l = 380
missing = arima_220_diff.copy()
missing[l:] = np.nan
```

```
fig, axes = plt.subplots(3, 1, figsize=(8, 8))
fig.subplots_adjust(hspace=0.1)

xlim = [0, len(missing)]
ylim = [-1000,10000]

ax0 = axes[0]
ntries = 10
for i in range(ntries):
    np.random.seed(i)
    try_diff = arima_forecast(missing, [1, -0.18, 0.06])
    t = np.cumsum(np.cumsum(try_diff))
    ax0.plot(t, color="black", alpha=0.3)
ax0.plot(arima_220, color="xkcd:hot pink", lw=3, label="data")
```

```python
ax0.plot([len(missing),len(missing)+1], [0]*2, color="black", alpha=0.3, label="forecast")
ax0.set(xticklabels=[],
        xlim=xlim,
        ylim=ylim)
ax0.text(0.5, 0.98, f"{ntries} forecasts",
         transform=ax0.transAxes, ha="center", va="top")
ax0.legend(frameon=False)
ax0.plot([380]*2, ylim, color="black", ls=":")

ax1 = axes[1]
ntries = 100
for i in range(ntries):
    np.random.seed(i)
    try_diff = arima_forecast(missing, [1, -0.18, 0.06])
    t = np.cumsum(np.cumsum(try_diff))
    ax1.plot(t, color="black", alpha=0.1)
ax1.plot(arima_220, color="xkcd:hot pink", lw=3)
ax1.set(xticklabels=[],
        ylabel="signal",
        xlim=xlim,
        ylim=ylim)
ax1.text(0.5, 0.98, f"{ntries} forecasts",
         transform=ax1.transAxes, ha="center", va="top")
ax1.plot([380]*2, ylim, color="black", ls=":")


ax2 = axes[2]
ntries = 1000
for i in range(ntries):
    np.random.seed(i)
    try_diff = arima_forecast(missing, [1, -0.18, 0.06])
    t = np.cumsum(np.cumsum(try_diff))
    ax2.plot(t, color="black", alpha=0.03)
ax2.plot(arima_220, color="xkcd:hot pink", lw=3)
ax2.set(xlabel='time',
        xlim=xlim,
        ylim=ylim)
ax2.text(0.5, 0.98, f"{ntries} forecasts",
         transform=ax2.transAxes, ha="center", va="top")
```

```
ax2.plot([380]*2, ylim, color="black", ls=":")
```



We will discuss this later in the course, but estimating the parameters is quite easy:

```
from statsmodels.tsa.arima.model import ARIMA

arima_model = ARIMA(arima_220, order=(2,2,0))
model = arima_model.fit()
print(model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  500
Model:                 ARIMA(2, 2, 0)   Log Likelihood                -703.606
Date:                Tue, 06 Feb 2024   AIC                           1413.212
Time:                        13:12:24   BIC                           1425.844
```

```
Sample:                             0   HQIC                          1418.170
                                - 500
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2171      0.045      4.782      0.000       0.128       0.306
ar.L2          0.0636      0.045      1.401      0.161      -0.025       0.153
sigma2         0.9878      0.072     13.739      0.000       0.847       1.129
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 5.16
Prob(Q):                              0.99   Prob(JB):                         0.08
Heteroskedasticity (H):               1.08   Skew:                             0.05
Prob(H) (two-sided):                  0.62   Kurtosis:                         2.51
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

29 practice

If you don't have it yet, please download here the meteorological data fro 2019 in Jerusalem.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.arima_process import ArmaProcess
np.random.seed(49)

# %matplotlib widget
```

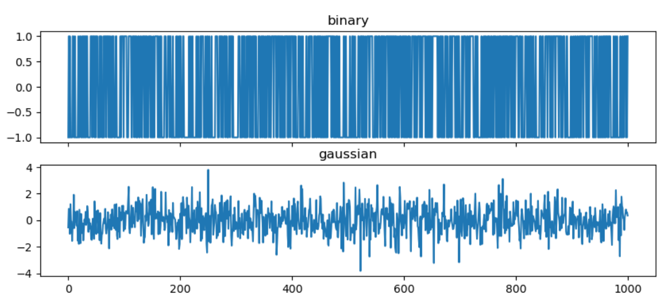
30 White noise

If we randomly draw values from the same distribution we will get white noise.

```
# generate binary noise random ones and zeroes with equal distribution
n = 1000
binary_noise = np.random.choice([-1, 1], size=n)

# generate gaussian noise with mean 0 and standard deviation 1
gaussian_noise = np.random.normal(0, 1, n)
```

```
# plot:
fig, ax = plt.subplots(2,1, figsize=(10,4), sharex=True)
ax[0].plot(binary_noise)
ax[0].set_title('binary')

ax[1].plot(gaussian_noise)
ax[1].set_title('gaussian')
```

```
Text(0.5, 1.0, 'gaussian')
```

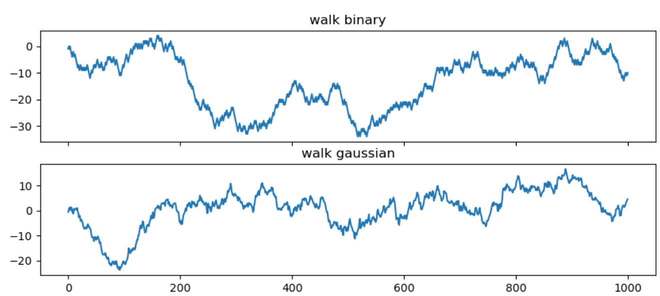


31 Random walk

If we cumulatively sum the white noise, we then get a random walk

```
walk_binary = binary_noise.cumsum()
walk_gaussian = gaussian_noise.cumsum()

fig, ax = plt.subplots(2,1, figsize=(10,4), sharex=True)
ax[0].plot(walk_binary)
ax[0].set_title('walk binary')

ax[1].plot(walk_gaussian)
ax[1].set_title('walk gaussian')
```

```
Text(0.5, 1.0, 'walk gaussian')
```



31.1 Differencing

Given an array

$$a = [a_0, a_1, a_2, ..., a_{n-1}]$$

the operation performed by `np.diff(a)` can be represented as:

$$\Delta a = [\Delta a_1, \Delta a_2, ..., \Delta a_{n-1}]$$

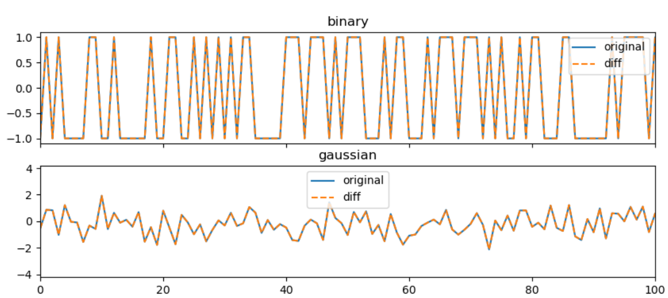
where

$$\Delta a_i = a_i - a_{i-1} \quad \text{for} \quad i = 1, 2, ..., n-1$$

If we difference the random walk we will get the white noise.

```
fig, ax = plt.subplots(2,1, figsize=(10,4), sharex=True)
ax[0].plot(binary_noise, label='original')
ax[0].plot(np.diff(walk_binary, prepend=0), label='diff', linestyle='--')
ax[0].set_title('binary')
ax[0].legend()
ax[0].set_xlim(0,100)

ax[1].plot(gaussian_noise, label='original')
ax[1].plot(np.diff(walk_gaussian, prepend=0), label='diff', linestyle='--')
ax[1].set_title('gaussian')
ax[1].legend()
ax[1].set_xlim(0,100)
```

```
(0.0, 100.0)
```



Another way of understanding this: the python operations `cumsum` and `diff` are each other's inverse.

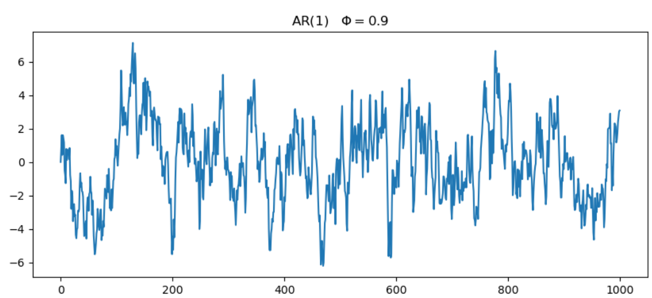
32 AR(1)

$$X_t = \phi X_{t-1} + \varepsilon.$$

This is called an Autoregressive Process of order 1, or AR(1). Here, the current value X_t is dependent on the immediately preceding value X_{t-1}.

```
#  initialize time series array
ar1_series = np.zeros(n)

# set a phi value, in addition to this value you should try phi>1 or phi=0
phi = 0.9

for i in range(1, n):
    ar1_series[i] = phi*ar1_series[i-1] + gaussian_noise[i]

# plot:
fig, ax = plt.subplots(figsize=(10,4))
ax.plot(ar1_series)
ax.set_title(f'AR(1)\t$\Phi={phi}$')
```

```
Text(0.5, 1.0, 'AR(1)\t$\\Phi=0.9$')
```



32.1 AR(p)

The next thing to do is to generalize, and define an autoregressive process that depends on p previous states:

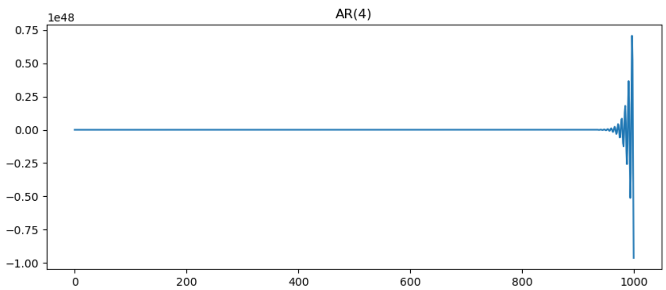
$$x_t = \phi_1 \, x_{t-1} + \phi_2 \, x_{t-2} + \cdots + \phi_p \, x_{t-p} + \varepsilon$$

```
# Function to generate AR(p) time series
# this function can recive p as an integer and then is will draw random phi values
# or, you can pass p as a np array of the specific phi values you want.
def generate_ar(n, p):
    # Check if p is an integer or an array
    if isinstance(p, int):
        # Generate random coefficients between -1 and 1
        phi = np.random.uniform(-1, 1, size=p)
    elif isinstance(p, np.ndarray):
        phi = p  # Use the provided array as coefficients
    else:
        raise ValueError("p should be either an integer or a NumPy array")

    print(phi)
    # Generate white noise
    noise = np.random.normal(0, 1, n)

    # Initialize time series array
    ar_series = np.zeros(n)

    for i in range(phi.size, n):
        ar_series[i] = np.dot(phi, ar_series[i-phi.size:i]) + noise[i]

    return ar_series
```

```
# plot using p as an int
p = 4
ar = generate_ar(n, p)
fig, ax = plt.subplots(figsize=(10,4))
ax.plot(ar)
ax.set_title(f'AR({p})')
```

```
[-0.82679831 -0.50310415 -0.68089179  0.1555622 ]
```

```
Text(0.5, 1.0, 'AR(4)')
```



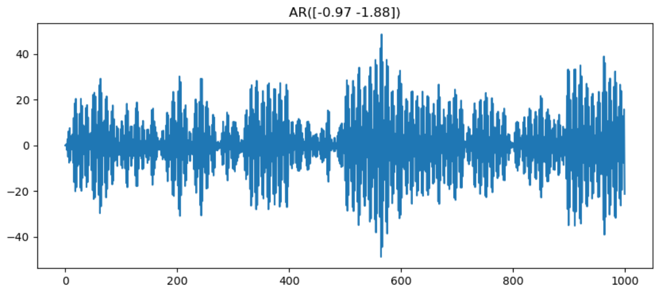
32.1.1 using specific ϕ values

In the cell below we can specify specific ϕ values.
Use the interactive tool from our website to chose the right values.
Remember, if one of the roots is inside the unit circle, the series will be **not** stationary.

```
# plot using p as an array of phi values
# the order should be [phi2, phi1]
p = np.array([-0.97,-1.88])
# p = np.array([-1.88,-0.97])

ar2 = generate_ar(n, p)
fig, ax = plt.subplots(figsize=(10,4))

ax.plot(ar2)
ax.set_title(f'AR({p})')
```

```
[-0.97 -1.88]
```

```
Text(0.5, 1.0, 'AR([-0.97 -1.88])')
```



32.1.2 Weak stationarity

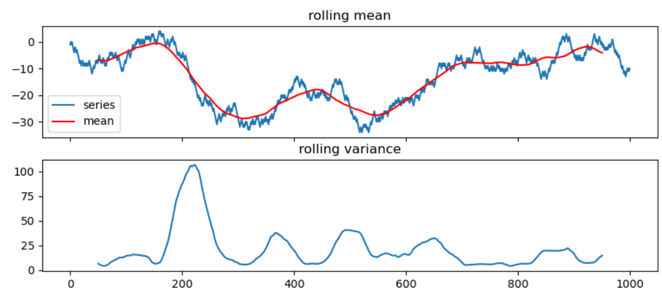
1. its mean μ does not vary in time:
$$\mu_X(t) = \mu_X(t+\tau)$$
for all values of t and τ.
2. its variance is finite for any time t:
$$\sigma^2_X(t) < \infty.$$
3. The autocorrelation function between two lagged versions of the same time series, $X(t_1)$ and $X(t_2)$, depends only on the difference $\tau = t_2 - t_1$.

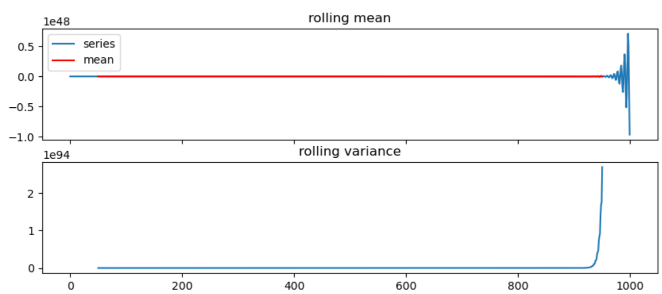
Let's get a feeling by plotting

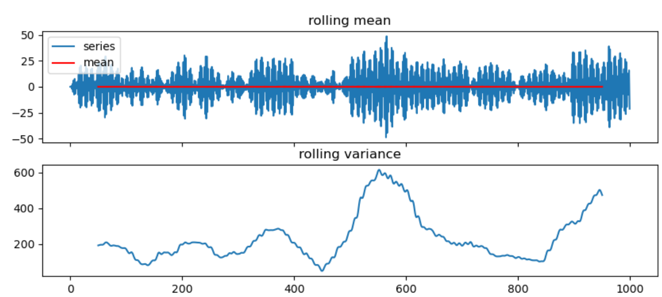
```
def test_stationarity(time_series, window=100):
    series = pd.Series(time_series)
    rolling_var = series.rolling(window=window, center=True).std()**2
    rolling_mean = series.rolling(window=window, center=True).mean()

    fig, ax = plt.subplots(2,1, figsize=(10,4), sharex=True)
    ax[0].plot(series, label='series')
    ax[0].plot(rolling_mean, c='r', label='mean')
    ax[0].legend()
    ax[0].set_title('rolling mean')

    ax[1].plot(rolling_var)
    ax[1].set_title('rolling variance')
```

```
    return
```

```
test_stationarity(walk_binary)
# test_stationarity(walk_gaussian)
# test_stationarity(ar1_series)
test_stationarity(ar)
test_stationarity(ar2)
```



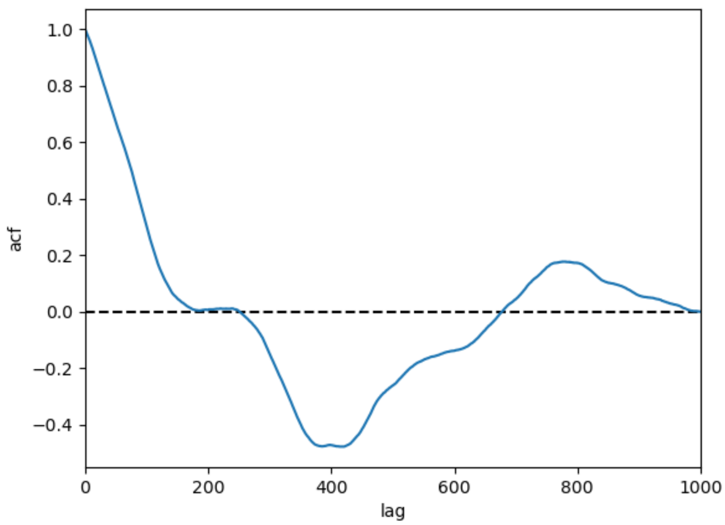




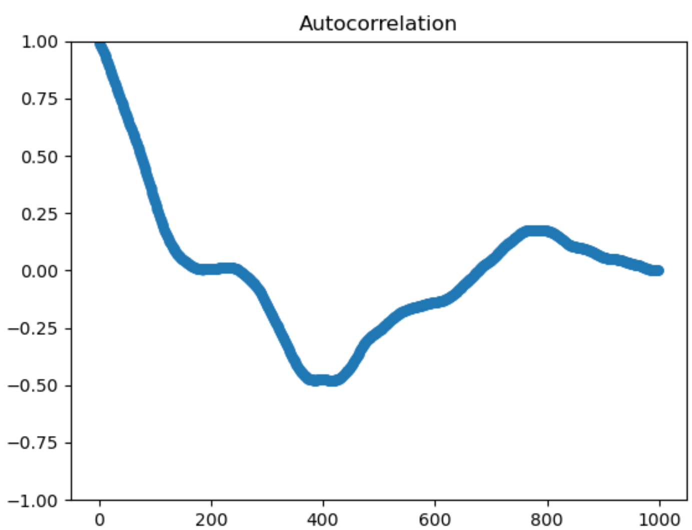
33 ACF

```
def compute_acf(series):
    N = len(series)
    lags = np.arange(N)
    acf = np.zeros_like(lags)
    series = (series - series.mean()) / series.std()
    for i in lags:
        acf[i] = np.sum(series[i:] * series[:N-i])
    acf = acf / N
    return lags, acf
```

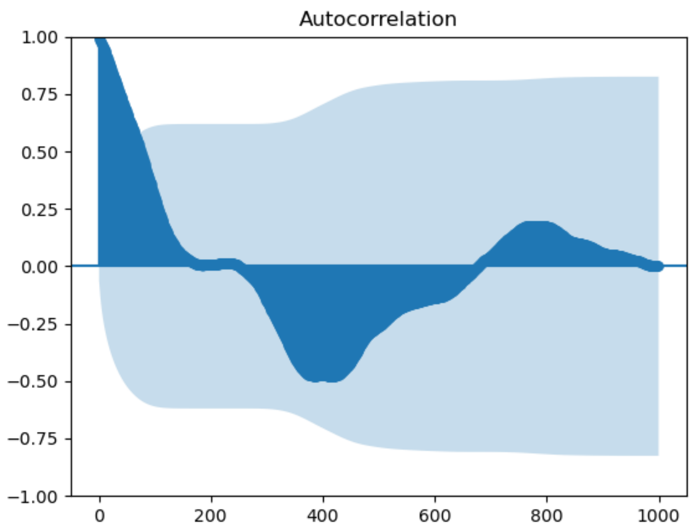
```
# walk_binary, walk_gaussian, ar1_series, ar, ar2
series_to_plot = walk_binary
fig, ax = plt.subplots()
lags, acf = compute_acf(series_to_plot)
ax.plot([0, n], [0]*2, color="black", ls="--")
ax.plot(lags, acf)
ax.set(xlabel="lag",
       ylabel="acf",
       xlim=[0, n]);
```




```
fig, ax = plt.subplots()
sm.graphics.tsa.plot_acf(series_to_plot, lags= n-1, ax=ax, label="statsmodels", alpha=None,
```

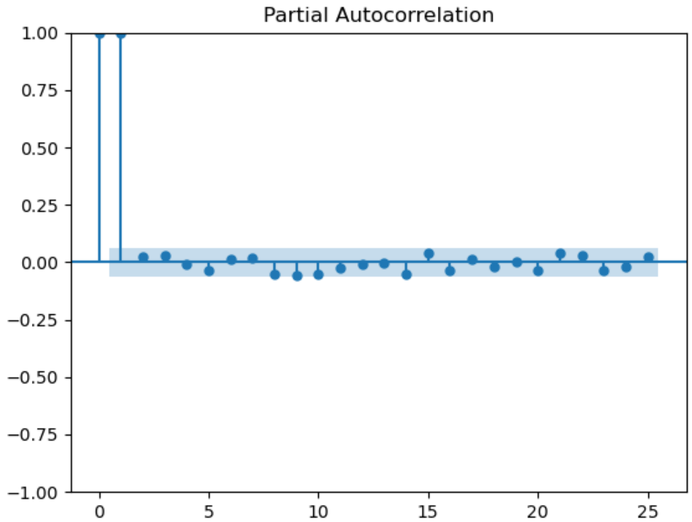


```python
fig, ax = plt.subplots()
sm.graphics.tsa.plot_acf(series_to_plot, lags= n-1, ax=ax, alpha=.05);
```




```python
fig, ax = plt.subplots()
sm.graphics.tsa.plot_pacf(series_to_plot, lags=25, ax=ax, alpha=.05);
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348: FutureWarning
  warnings.warn(
```



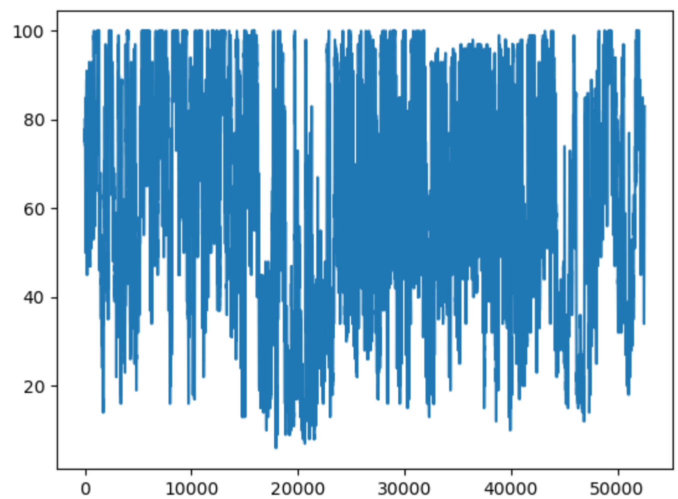
33.1 Now let's work with actual data

```
filename = "jerusalem2019.csv"
df = pd.read_csv(filename, na_values=['-'])
df.rename(columns={'Temperature (°C)': 'temperature',
                   'Rainfall (mm)': 'rain'}, inplace=True)
df['date'] = pd.to_datetime(df['Date & Time (Winter)'], dayfirst=True)
df = df.set_index('date')
df = df.fillna(method='ffill')
df
```

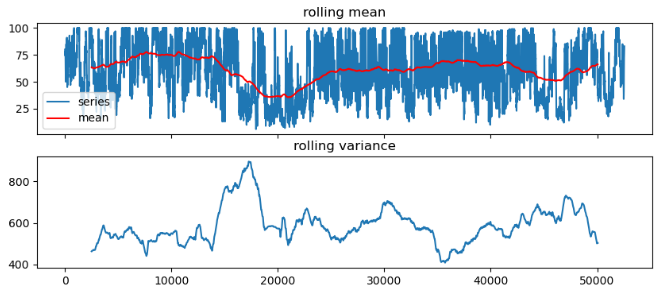
	Station	Date & Time (Winter)	Diffused radiation (W/m^2)	Global rad
date				
2019-01-01 00:00:00	Jerusalem Givat Ram	01/01/2019 00:00	0.0	0.0
2019-01-01 00:10:00	Jerusalem Givat Ram	01/01/2019 00:10	0.0	0.0
2019-01-01 00:20:00	Jerusalem Givat Ram	01/01/2019 00:20	0.0	0.0
2019-01-01 00:30:00	Jerusalem Givat Ram	01/01/2019 00:30	0.0	0.0
2019-01-01 00:40:00	Jerusalem Givat Ram	01/01/2019 00:40	0.0	0.0
...	...	...	...	...
2019-12-31 22:20:00	Jerusalem Givat Ram	31/12/2019 22:20	0.0	0.0

	Station	Date & Time (Winter)	Diffused radiation (W/m^2)	Global rad
date				
2019-12-31 22:30:00	Jerusalem Givat Ram	31/12/2019 22:30	0.0	0.0
2019-12-31 22:40:00	Jerusalem Givat Ram	31/12/2019 22:40	0.0	0.0
2019-12-31 22:50:00	Jerusalem Givat Ram	31/12/2019 22:50	0.0	0.0
2019-12-31 23:00:00	Jerusalem Givat Ram	31/12/2019 23:00	0.0	0.0

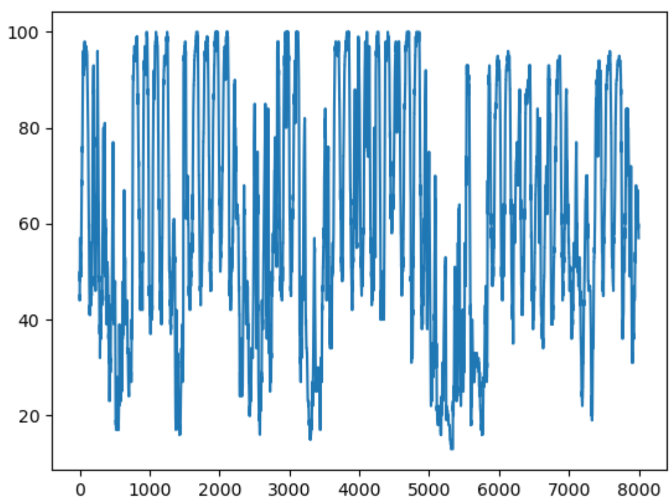
```
# t = df['temperature'].values
t = df['Relative humidity (%)'].values
# t = df['Wind speed (m/s)'].values
# t = df['Wind direction (°)'].values
fig, ax = plt.subplots()
ax.plot(t)
```

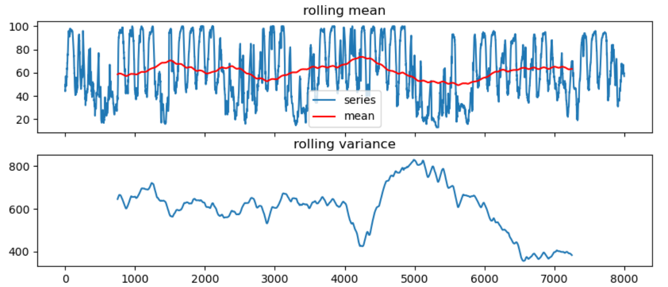



```
test_stationarity(t, window=5000)
```

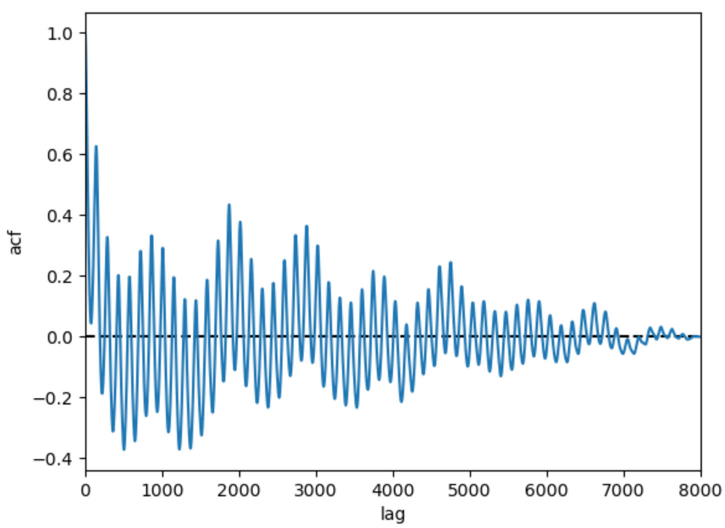



```
t_stationary = t[27000:35000]
fig, ax = plt.subplots()
ax.plot(t_stationary)
test_stationarity(t_stationary, window=1500)
```



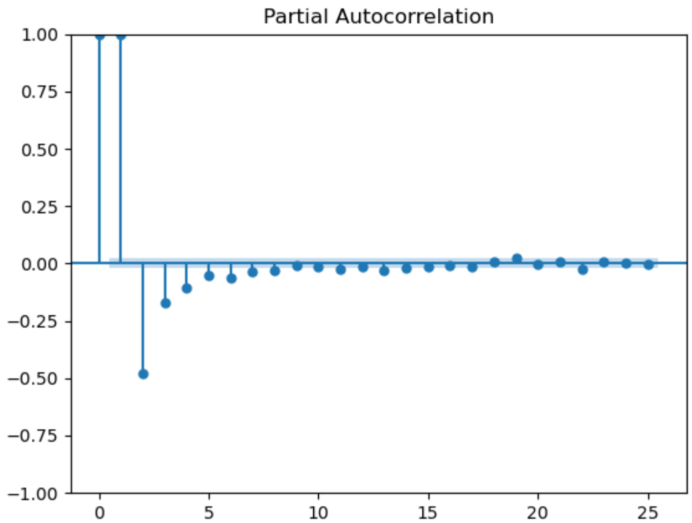



```
series_to_plot = t_stationary
fig, ax = plt.subplots()
lags, acf = compute_acf(series_to_plot)
ax.plot([0, len(series_to_plot)], [0]*2, color="black", ls="--")
ax.plot(lags, acf)
ax.set(xlabel="lag",
       ylabel="acf",
       xlim=[0, len(series_to_plot)]);
```




```
fig, ax = plt.subplots()
sm.graphics.tsa.plot_pacf(series_to_plot, lags=25, ax=ax, alpha=.05);
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348: FutureWarning
  warnings.warn(
```



34 filling missing values

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
import statsmodels.api as sm
from statsmodels.tsa.arima_process import ArmaProcess
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import warnings
# Suppress FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
import random

# %matplotlib widget
```

```
def plot_all_columns(data):
    column_list = data.columns

    fig, ax = plt.subplots(len(column_list),1, sharex=True, figsize=(10,len(column_list)*2))

    if len(column_list) == 1:
        ax.plot(data[column_list[0]])
        return
    for i, column in enumerate(column_list):
        ax[i].plot(data[column])
```

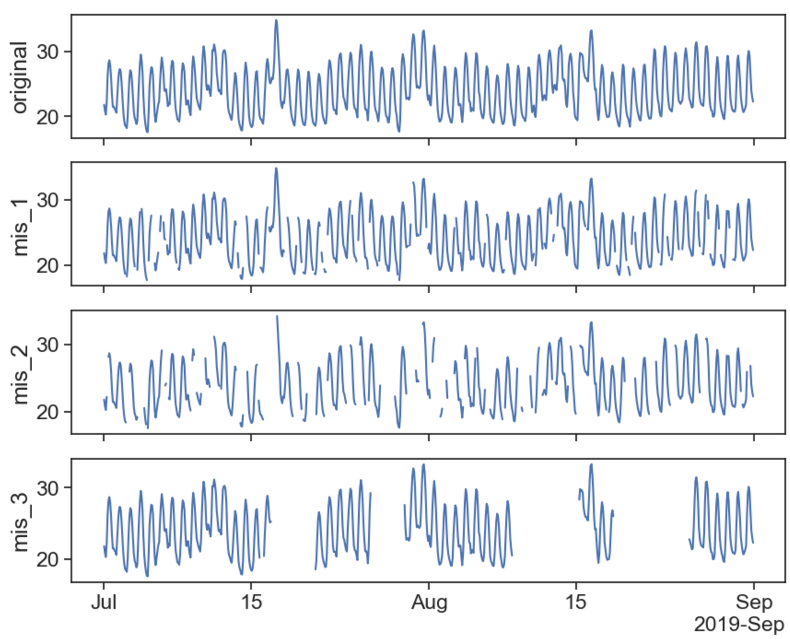
```
        ax[i].set(ylabel=column)

    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)
    ax[i].xaxis.set_major_locator(locator)
    ax[i].xaxis.set_major_formatter(formatter)

    return
```

```
def plot_missing_vals(original, new, title=''):
    column_list = original.columns

    fig, ax = plt.subplots(len(column_list),1, sharex=True, figsize=(10,len(column_list)*2))

    if len(column_list) == 1:
        ax.set_title(title)
        ax.plot(new[column_list[0]], c='r')
        ax.plot(original[column_list[0]])
        return
    for i, column in enumerate(column_list):
        ax[i].plot(original[column_list[0]], c='tab:blue', alpha=0.2)
        ax[i].plot(new[column], c='r')
        ax[i].plot(original[column])
        if i != 0:
            ax[i].scatter(new[column].index,new[column].values, marker='o', s=4, c='r')
            ax[i].scatter(original[column].index,original[column].values, marker='o', s=4)
        else:
            ax[i].set_title(title)
        ax[i].set(ylabel=column)
        # calculate and display MSE
        mse = mean_squared_error(original[column_list[0]], new[column])
        ax[i].text(0.01, 0.95, f'MSE: {mse:.4f}', transform=ax[i].transAxes, fontsize=12, ve

    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)
    ax[i].xaxis.set_major_locator(locator)
    ax[i].xaxis.set_major_formatter(formatter)

    return
```

```python
df = pd.read_csv('data_missing.csv', index_col='date', parse_dates=True)
df
```

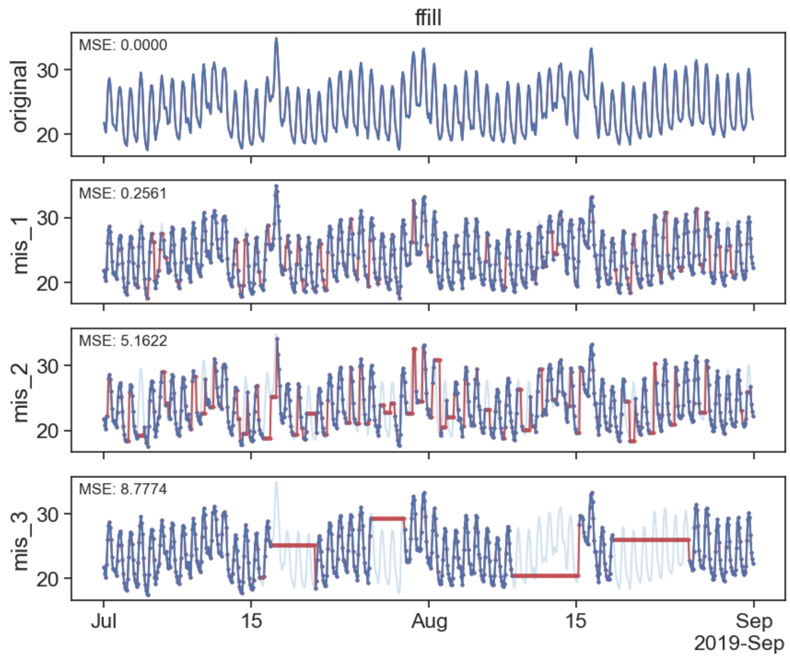
date	original	mis_1	mis_2	mis_3
2019-07-01 00:00:00	21.808333	21.808333	21.808333	21.808333
2019-07-01 02:00:00	20.808333	20.808333	20.808333	20.808333
2019-07-01 04:00:00	20.283333	20.283333	20.283333	20.283333
2019-07-01 06:00:00	22.216667	22.216667	22.216667	22.216667
2019-07-01 08:00:00	26.091667	26.091667	NaN	26.091667
...	...	...	...	...
2019-08-31 14:00:00	29.400000	29.400000	NaN	29.400000
2019-08-31 16:00:00	26.808333	26.808333	26.808333	26.808333
2019-08-31 18:00:00	24.050000	24.050000	24.050000	24.050000
2019-08-31 20:00:00	23.008333	23.008333	23.008333	23.008333
2019-08-31 22:00:00	22.275000	22.275000	22.275000	22.275000

```python
plot_all_columns(df)
```



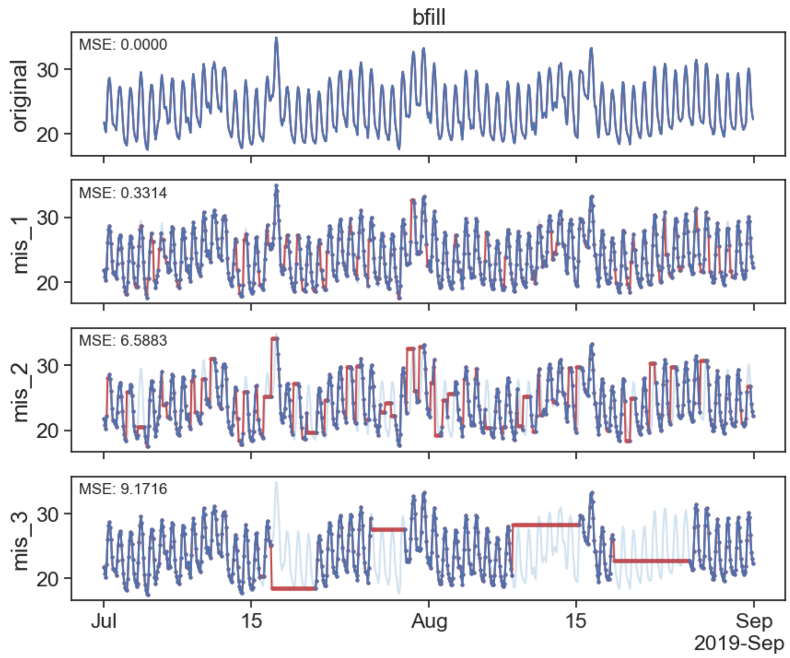
34.1 Forward fill

```
df_ffil = df.fillna(method='ffill')
plot_missing_vals(df, df_ffil, title='ffill')
```



34.2 Backwrds fill

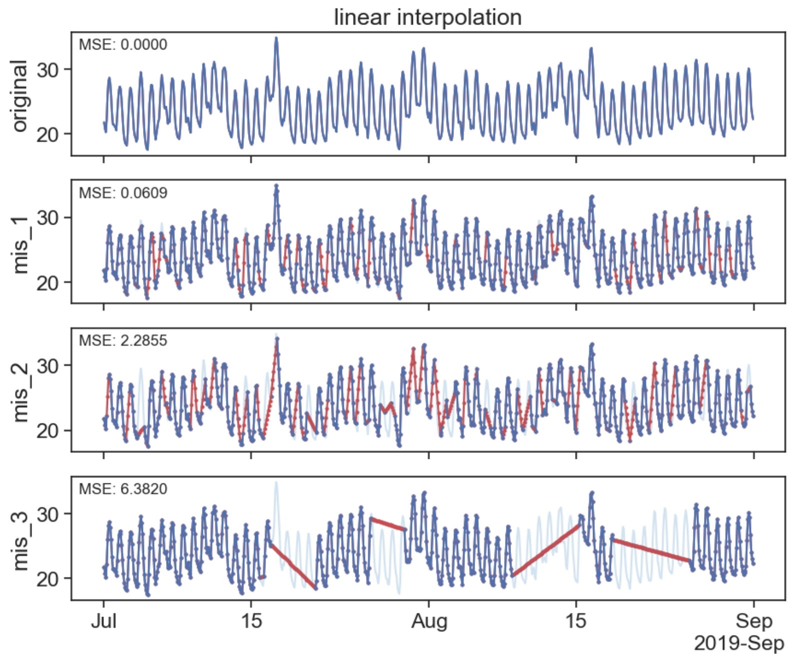
```
df_bfil = df.fillna(method='bfill')
plot_missing_vals(df, df_bfil, title='bfill')
```



34.3 Interpolation

34.3.1 linear

```
interpolated_df = df.interpolate(method='linear')
plot_missing_vals(df, interpolated_df, title='linear interpolation')
```

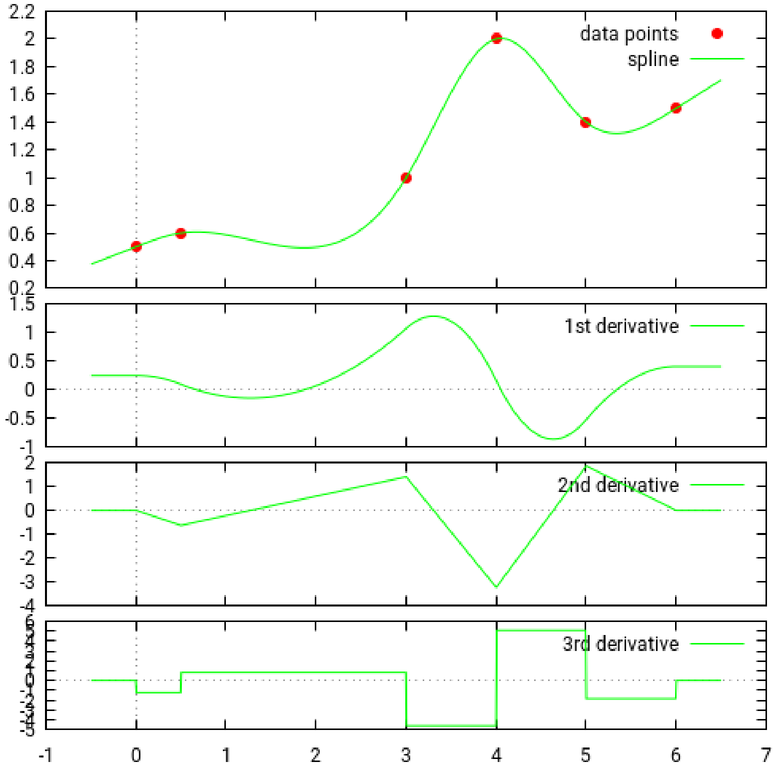


https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.interpolate.html

34.3.2 cubic splines

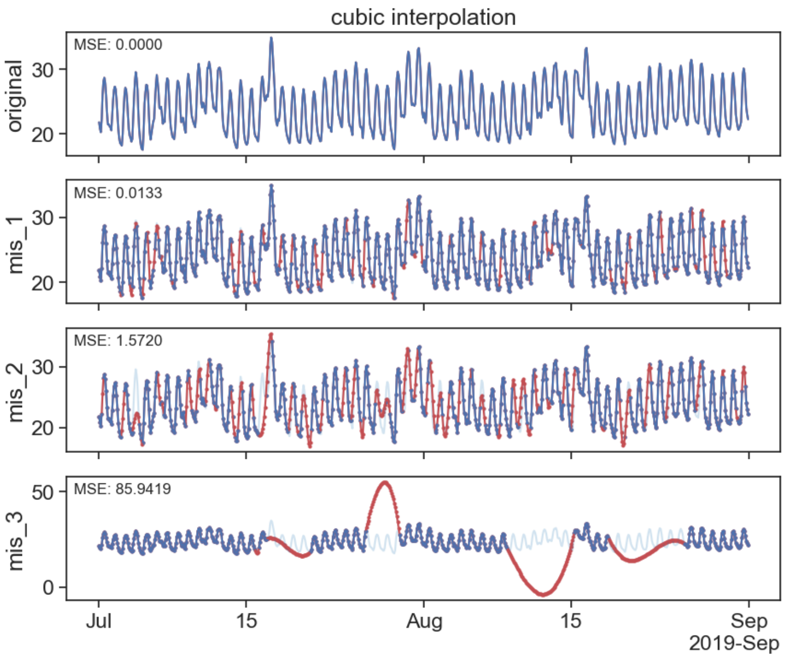
Source: https://docs.scipy.org/doc/scipy/tutorial/interpolate/1D.html#cubic-splines

Piecewise linear interpolation produces corners at data points, where linear pieces join. To produce a smoother curve, you can use cubic splines, where the interpolating curve is made of cubic pieces with **matching first and second derivatives**.

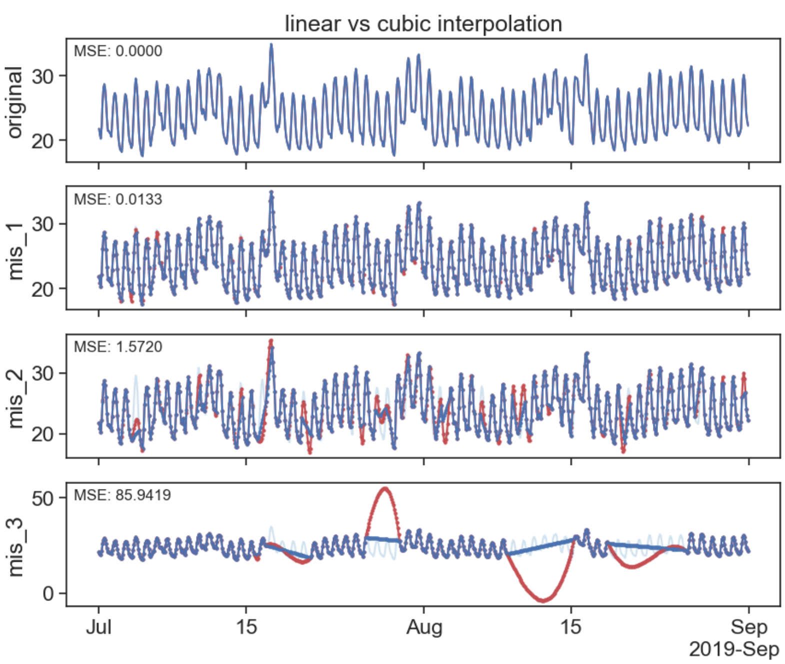


Source: https://kluge.in-chemnitz.de/opensource/spline/

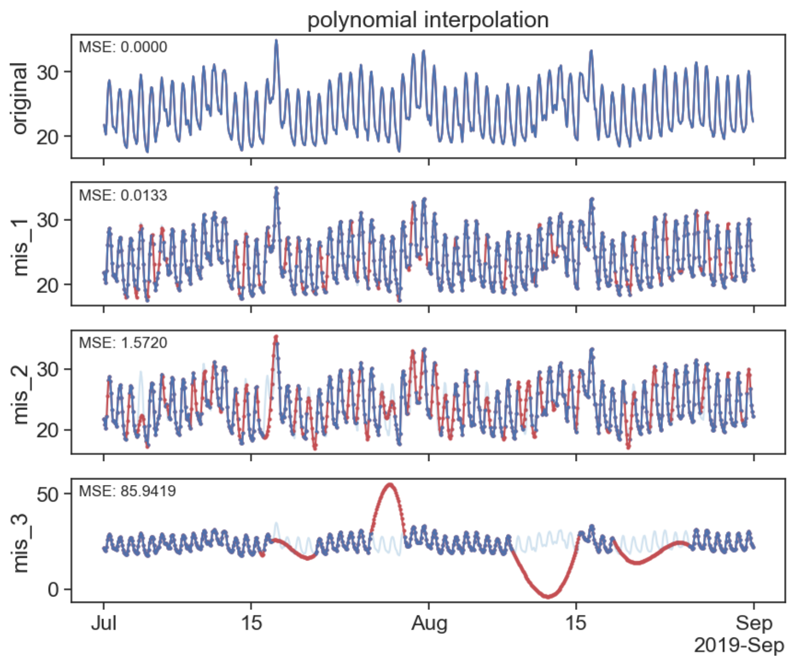
```
interpolated_cubic_df = df.interpolate(method='cubic')
plot_missing_vals(df, interpolated_cubic_df, title='cubic interpolation')
```



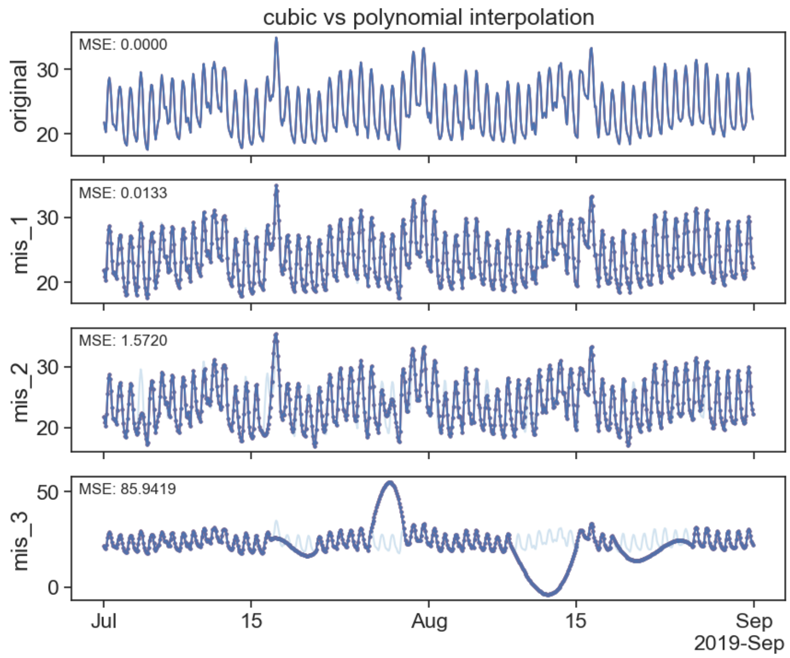

```
plot_missing_vals(interpolated_df, interpolated_cubic_df, title='linear vs cubic interpolati
```



```
interpolated_poly_df = df.interpolate(method='polynomial', order=3)
plot_missing_vals(df, interpolated_poly_df, title='polynomial interpolation')
```




```
plot_missing_vals(interpolated_cubic_df, interpolated_poly_df, title='cubic vs polynomial in
```



All available interpolation types can be found at the pandas documentation

34.4 random forest

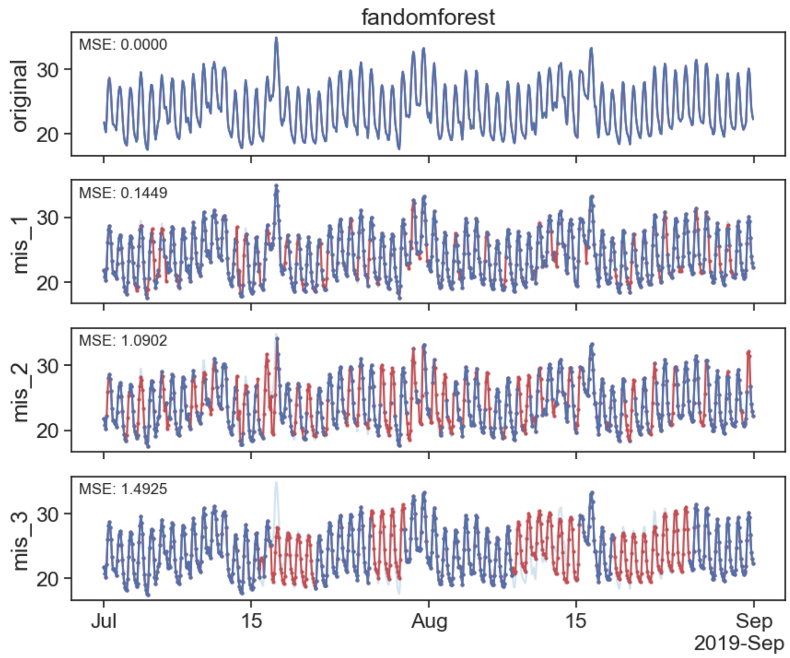
Here we explore the use of a commnly used Machine learning model - RandomForest

```
def fillna_randomforest(series):
    # Ensure the series index is a datetime object
    if not isinstance(series.index, pd.DatetimeIndex):
        raise ValueError("Index must be a DatetimeIndex")

    # Split series into observed and missing values
    observed = series.dropna()
    missing = series[series.isna()]

    # Extracting time-based features
```

```
    def create_features(index):
        return np.array([index.hour, index.day, index.month, index.year]).T

    # Create features for training data
    X_train = create_features(observed.index)
    y_train = observed.values

    # Train the Random Forest regression model
    model = RandomForestRegressor()
    model.fit(X_train, y_train)

    # Create features for missing data and predict
    X_missing = create_features(missing.index)
    predicted_values = model.predict(X_missing)

    # Assign the predicted values to the missing positions
    series_filled = series.copy()
    series_filled[series_filled.isna()] = predicted_values

    return series_filled
```

```
columns = df.columns
df_rf = df.copy()
for i, column in enumerate(columns):
    if i == 0:
        continue
    df_rf[column] = fillna_randomforest(df_rf[column])
```

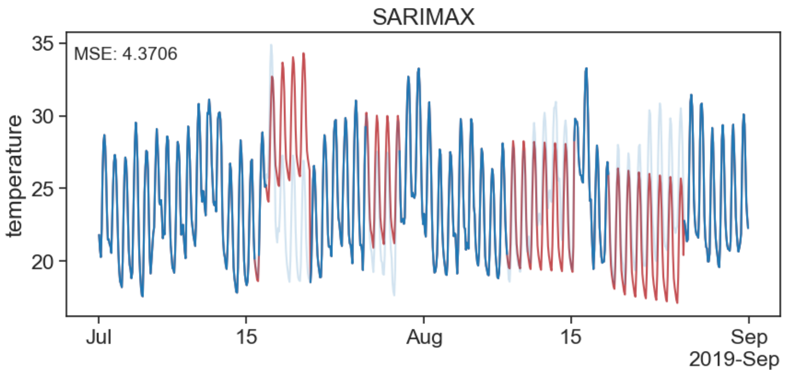
```
plot_missing_vals(df, df_rf, title='fandomforest')
```


fandomforest
MSE: 0.0000
original
MSE: 0.1449
mis_1
MSE: 1.0902
mis_2
MSE: 1.4925
mis_3
30
20
Jul
15
Aug
15
Sep
2019-Sep

35 SARIMAX

```
# Model Building and Imputation
# Build a SARIMA model on the data with gaps
seasonal_period = 24/2
model = SARIMAX(df['mis_3'], order=(2, 1, 0), seasonal_order=(1, 1, 1, seasonal_period), enf
results = model.fit(disp=False)

# Impute missing values one by one
ts_data_filled = df['mis_3'].copy()
missing_indices = df['mis_3'].loc[df['mis_3'].isna()]
for missing_index in missing_indices.index:
    ts_data_filled[missing_index] = results.predict(start=missing_index, end=missing_index)
```

```
fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(ts_data_filled, label='Imputed Data', color='r')
ax.plot(df['original'], label='Original Data', color='tab:blue', alpha=0.2)
ax.plot(df['mis_3'], label='Data with Gaps', color='tab:blue', alpha=1)
ax.set_ylabel('temperature')
ax.set_title('SARIMAX')
# calculate and display MSE
mse = mean_squared_error(df['original'], ts_data_filled)
ax.text(0.01, 0.95, f'MSE: {mse:.4f}', transform=ax.transAxes, fontsize=14, verticalalignmen
# ax.legend()

locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
formatter = mdates.ConciseDateFormatter(locator)
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)
```



The above looks good at the begining of the gap but at the end of the gap it looks bad. That is because we are forcasting and not filling in between the lines... # forward - reverse combination

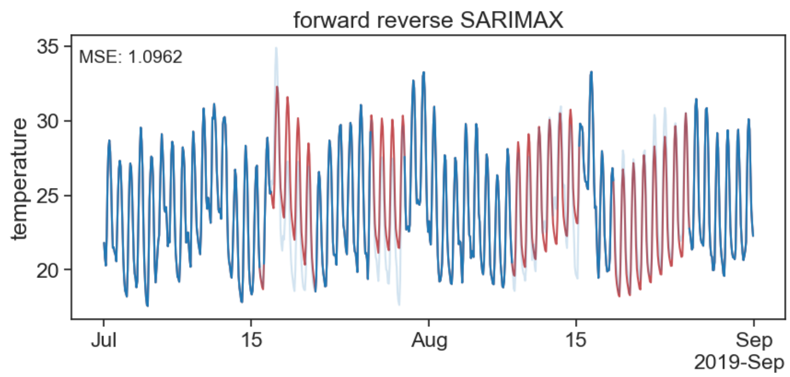
```
def find_nan_gaps_indexes_datetime(series):
    """
    Find and pair the start and end datetime indexes of gaps in a pandas Series with a datet

    Parameters:
    series (pandas Series): The input pandas Series with a datetime index containing NaN gap

    Returns:
    list of tuples: A list of tuples where each tuple contains the start datetime index and
    """
    is_nan = series.isna().values
    start_indexes = np.where(is_nan & ~np.roll(is_nan, 1))[0]
    end_indexes = np.where(is_nan & ~np.roll(is_nan, -1))[0]

    # If the last gap extends to the end of the Series, add its end index
    if is_nan[-1]:
        end_indexes = np.append(end_indexes, len(series) - 1)

    # Pair start and end datetime indexes together
    gap_pairs = [(series.index[start], series.index[end]) for start, end in zip(start_indexe

    return gap_pairs
```

```
def forward_reverse_SARIMAX(ts_data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12), seasonal
    # The function forward_reverse_SARIMAX aims to impute missing values in a time series da
    # SARIMAX model approach, both in the original and reversed order of the data. Initially
    # SARIMAX model on the provided dataset to predict and fill in the missing values. Then,
    # the time series data, applies another SARIMAX model on this reversed data, and imputes
    # values again. After obtaining the imputed datasets from both forward and reverse direc
    # function iteratively blends the imputed values from both models for each missing point
    # the weight given to the forward and reverse imputations based on the position within t
    # This blended approach aims to leverage the predictive insights from both the preceding
    # data points, potentially providing a more accurate and balanced imputation for the mis

    # Model Building and Imputation
    # Build a SARIMA model on the data with gaps
    model = SARIMAX(ts_data, order=order, seasonal_order=seasonal_order, enforce_stationarit
    results = model.fit(disp=False)

    # Impute missing values one by one
    ts_data_filled = ts_data.copy()
    missing_indices = ts_data.loc[ts_data.isna()]
    for missing_index in missing_indices.index:
        ts_data_filled[missing_index] = results.predict(start=missing_index, end=missing_ind


    # Reverse the time series
    ts_data_reversed = ts_data.iloc[::-1]

    # Build SARIMAX model on reversed data
    model_reversed = SARIMAX(ts_data_reversed, order=order, seasonal_order=seasonal_order, e
    results_reversed = model_reversed.fit(disp=False)

    # Impute missing values in reversed data
    ts_data_filled_reversed = ts_data_reversed.copy()
    missing_indices_reversed = ts_data_reversed.loc[ts_data_reversed.isna()]
    for missing_index in missing_indices_reversed.index:
        ts_data_filled_reversed[missing_index] = results_reversed.predict(start=missing_inde

    # Reverse the imputed data back to original order
    ts_data_filled_reversed = ts_data_filled_reversed.iloc[::-1]
```

```
    # Initialize a series to hold the combined predictions
    ts_data_combined = ts_data.copy()

    # Iterate over each gap
    for gap_start, gap_end in find_nan_gaps_indexes_datetime(ts_data):
        # print(gap_start)
        # Calculate the number of periods in the gap
        gap = ts_data[gap_start:gap_end]
        gap_length = len(gap)

        # Iterate over each index in the gap
        # for i, index in enumerate(pd.date_range(start=gap_start, end=gap_end, freq=ts_data
        for i, index in enumerate(gap.index):
            forward_weight = (gap_length - i) / gap_length
            backward_weight = i / gap_length
            combined_prediction = (ts_data_filled.at[index] * forward_weight +
                                   ts_data_filled_reversed.at[index] * backward_weight)
            ts_data_combined.at[index] = combined_prediction

    return ts_data_combined
```

```
f_r_SARIMAX = forward_reverse_SARIMAX(df['mis_3'])
```

```
fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(f_r_SARIMAX, label='Imputed Data', color='r')
ax.plot(df['original'], label='Original Data', color='tab:blue', alpha=0.2)
ax.plot(df['mis_3'], label='Data with Gaps', color='tab:blue', alpha=1)
ax.set_ylabel('temperature')
ax.set_title('forward reverse SARIMAX')
# calculate and display MSE
mse = mean_squared_error(df['original'], f_r_SARIMAX)
ax.text(0.01, 0.95, f'MSE: {mse:.4f}', transform=ax.transAxes, fontsize=14, verticalalignmen
# ax.legend()

locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
formatter = mdates.ConciseDateFormatter(locator)
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)
```




```
# Calculate MSE
mse_fr = mean_squared_error(df['original'], f_r_SARIMAX)
mse_rf = mean_squared_error(df['original'], df_rf['mis_3'])

print("Mean Squared Error sarimax:", mse_fr)
print("Mean Squared Error rand:", mse_rf)
```

```
Mean Squared Error sarimax: 1.0961749546737887
Mean Squared Error rand: 1.4924638305799802
```

Part VI

time lags

36 motivation

37 cross-correlation

```
import numpy as np
```

```
print('dfvdfv')
```

```
dfvdfv
```

38 dynamic time warp

39 LDTW

according to this paper

Part VII

frequency

40 motivation

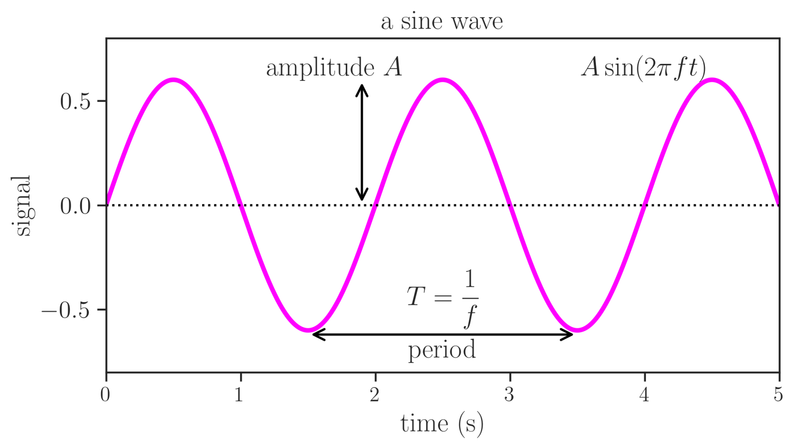
41 Fourier transform

41.1 basic wave concepts

The function

$$f(t) = B\sin(2\pi ft) \tag{41.1}$$

has two basic characteristics, its amplitude B and frequency f.

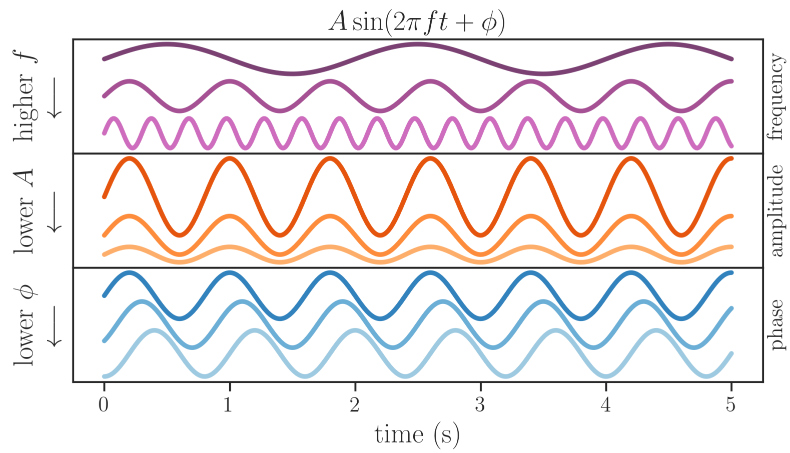


In the figure above, the amplitude $B = 0.6$ and we see that the distance between two peaks is called period, $T = 2$ s. The frequency is defined as the inverse of the period:

$$f = \frac{1}{T}. \tag{41.2}$$

When time is in seconds, then the frequency is measured in Hertz (Hz). For the graph above, therefore, we see a wave whose frequency is $f = 1/(2\text{ s}) = 0.5$ Hz.

In the figure below, we see what happens when we vary the values of the frequency and amplitude.



The graph above introduces two new characteristics of a wave, its phase ϕ, and its offset B. A more general description of a sine wave is

$$f(t) = B\sin(2\pi ft + \phi) + B_0. \tag{41.3}$$

The offset B_0 moves the wave up and down, while changing the value of ϕ makes the sine wave move left and right. When the phase $\phi = 2\pi$, the sine wave will have shifted a full period, and the resulting wave is identical to the original:

$$B\sin(2\pi ft) = B\sin(2\pi ft + 2\pi). \tag{41.4}$$

All the above can also be said about a cosine, whose general for can be given as

$$A\cos(2\pi ft + \phi) + A_0 \tag{41.5}$$

One final point before we jump into the deep waters is that the sine and cosine functions are related through a simple phase shift:

$$\cos\left(2\pi ft + \frac{\pi}{2}\right) = \sin\left(2\pi ft\right)$$

41.2 Fourier's theorem

Fourier's theorem states that

> Any periodic signal is composed of a superposition of pure sine waves, with suitably chosen amplitudes and phases, whose frequencies are harmonics of the fundamental frequency of the signal.

See the following animations to visualize the theorem in action.

Source: https://en.wikipedia.org/wiki/File:Fourier_series_and_transform.gif

Source: https://commons.wikimedia.org/wiki/File:Fourier_synthesis_square_wave_animated.gif

Source: https://commons.wikimedia.org/wiki/File:Sawtooth_Fourier_Animation.gif

Source: https://commons.wikimedia.org/wiki/File:Continuous_Fourier_transform_of_rect_and_sinc_functions.gif

41.3 Fourier series

> a periodic function can be described as a sum of sines and cosines.

The classic examples are usually the square function and the sawtooth function:

Not any function, but certainly most functions we will deal with in this course. The function has to fullful the Dirichlet conditions

[Source: https://www.geogebra.org/m/tkajbzmg]

https://www.geogebra.org/m/k4eq4fkr

$$F[x(t)] = F(f) = \int_{-\infty}^{\infty} x(t)e^{-2\pi ift}dt$$

$$f(t) = \int_{-\infty}^{\infty} F(f) e^{2\pi i f t} df$$

https://dibsmethodsmeetings.github.io/fourier-transforms/

https://www.jezzamon.com/fourier/index.html

42 filtering

43 Nyquist-Shannon sampling theorem

Part VIII

seasonality

44 motivation

45 seasonal decomposition

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()  # datetime converter for a matplotlib
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.dates as mdates
from matplotlib.dates import DateFormatter
```

45.1 trends in atmospheric carbon dioxide

Mauna Loa CO2 concentration.
data from NOAA

```
url = "https://gml.noaa.gov/webdata/ccgg/trends/co2/co2_weekly_mlo.csv"
# df = pd.read_csv(url, header=47, na_values=[-999.99])

# you can first download, and then read the csv
filename = "co2_weekly_mlo.csv"
df = pd.read_csv(filename, header=35, na_values=[-999.99])

df
```

	1974	5	19	1974.3795	333.37	5.1	-999.99	-999.99.1	50.39
0	1974	5	26	1974.3986	332.95	6	NaN	NaN	50.05
1	1974	6	2	1974.4178	332.35	5	NaN	NaN	49.59
2	1974	6	9	1974.4370	332.20	7	NaN	NaN	49.64

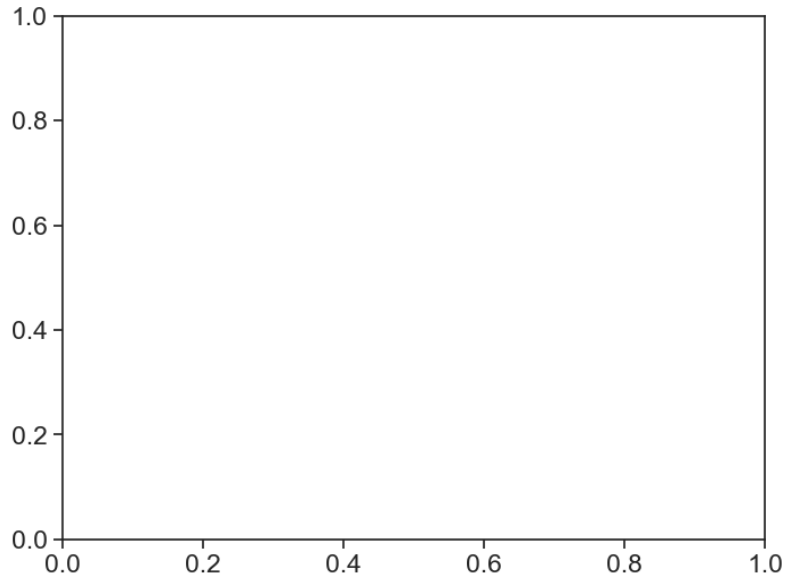
	1974	5	19	1974.3795	333.37	5.1	-999.99	-999.99.1	50.39
3	1974	6	16	1974.4562	332.37	7	NaN	NaN	50.06
4	1974	6	23	1974.4753	331.73	5	NaN	NaN	49.72
...	...	...	...	...	...	...	...	...	...
2565	2023	7	23	2023.5575	421.28	4	418.03	397.30	141.60
2566	2023	7	30	2023.5767	420.83	6	418.10	396.80	141.69
2567	2023	8	6	2023.5959	420.02	6	417.36	395.65	141.41
2568	2023	8	13	2023.6151	418.98	4	417.25	395.24	140.89
2569	2023	8	20	2023.6342	419.31	2	416.64	395.22	141.71

```
df['date'] = pd.to_datetime(df[['year', 'month', 'day']])
df = df.set_index('date')
df
```

	year	month	day	decimal	average	ndays	1 year ago	10 years ago	increase since 1800
date									
1974-05-19	1974	5	19	1974.3795	333.37	5	NaN	NaN	50.40
1974-05-26	1974	5	26	1974.3986	332.95	6	NaN	NaN	50.06
1974-06-02	1974	6	2	1974.4178	332.35	5	NaN	NaN	49.60
1974-06-09	1974	6	9	1974.4370	332.20	7	NaN	NaN	49.65
1974-06-16	1974	6	16	1974.4562	332.37	7	NaN	NaN	50.06
...	...	...	...	...	...	...	...	...	...
2022-06-26	2022	6	26	2022.4836	420.31	7	418.14	395.36	138.71
2022-07-03	2022	7	3	2022.5027	419.73	6	417.49	395.15	138.64
2022-07-10	2022	7	10	2022.5219	419.08	6	417.25	394.59	138.52
2022-07-17	2022	7	17	2022.5411	418.43	6	417.14	394.64	138.41
2022-07-24	2022	7	24	2022.5603	417.84	6	415.68	394.11	138.36

```
# %matplotlib widget

fig, ax = plt.subplots(1, figsize=(8,6))
ax.plot(df['average'])
ax.set(xlabel="date",
       ylabel="CO2 concentration (ppm)",
       # ylim=[0, 430],
       title="Mauna Loa CO2 concentration");
```

```
KeyError: 'average'
```



fill missing data. interpolate method: 'time'
interpolation methods visualized

```
df['co2'] = (df['average'].resample("D") #resample daily
                            .interpolate(method='time') #interpolate by time
             )
df
```

	year	month	day	decimal	average	ndays	1 year ago	10 years ago	increase since 1800
date									
1974-05-19	1974	5	19	1974.3795	333.37	5	NaN	NaN	50.40
1974-05-26	1974	5	26	1974.3986	332.95	6	NaN	NaN	50.06
1974-06-02	1974	6	2	1974.4178	332.35	5	NaN	NaN	49.60
1974-06-09	1974	6	9	1974.4370	332.20	7	NaN	NaN	49.65
1974-06-16	1974	6	16	1974.4562	332.37	7	NaN	NaN	50.06
...	...	...	...	...	...	...	...	...	...
2022-06-26	2022	6	26	2022.4836	420.31	7	418.14	395.36	138.71
2022-07-03	2022	7	3	2022.5027	419.73	6	417.49	395.15	138.64
2022-07-10	2022	7	10	2022.5219	419.08	6	417.25	394.59	138.52

date	year	month	day	decimal	average	ndays	1 year ago	10 years ago	increase since 1800
2022-07-17	2022	7	17	2022.5411	418.43	6	417.14	394.64	138.41
2022-07-24	2022	7	24	2022.5603	417.84	6	415.68	394.11	138.36

45.2 decompose data

`seasonal_decompose` returns an object with four components:

- observed: $Y(t)$
- trend: $T(t)$
- seasonal: $S(t)$
- resid: $e(t)$

Additive model:

$$Y(t) = T(t) + S(t) + e(t)$$

Multiplicative model:

$$Y(t) = T(t) \times S(t) \times e(t)$$

45.2.0.1 Interlude

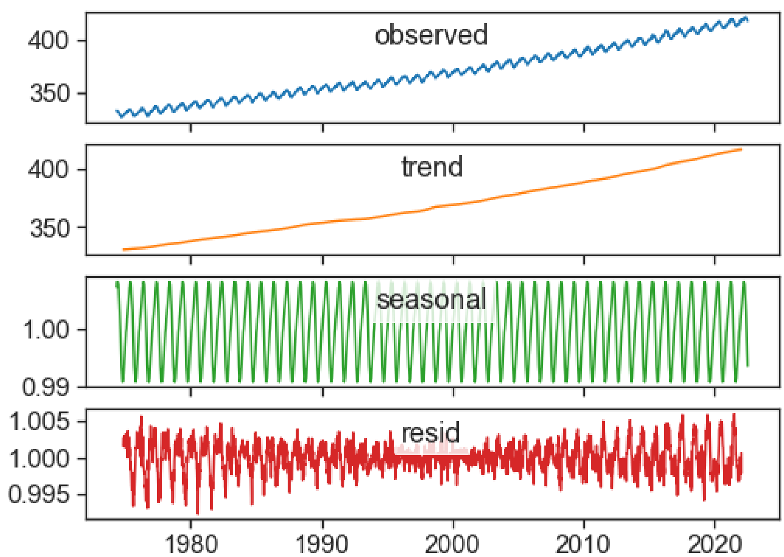
learn how to use `zip` in a loop

```python
letters = ['a', 'b', 'c', 'd', 'e']
numbers = [1, 2, 3, 4, 5]
# zip let's us iterate over to lists at the same time
for l, n in zip(letters, numbers):
    print(f"{l} = {n}")
```

```
a = 1
b = 2
c = 3
d = 4
e = 5
```

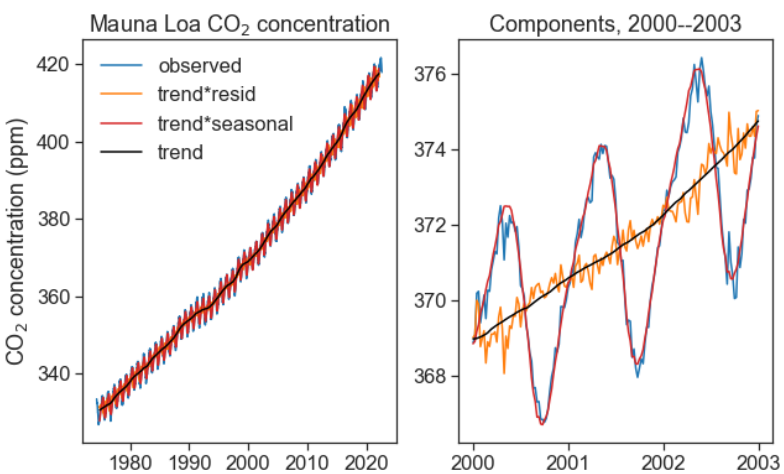
Plot each component separately.

```
# %matplotlib widget

fig, ax = plt.subplots(4, 1, figsize=(8,6), sharex=True)
decomposed_m = seasonal_decompose(df['co2'], model='multiplicative')
decomposed_a = seasonal_decompose(df['co2'], model='additive')
decomposed = decomposed_m
pos = (0.5, 0.9)
components =["observed", "trend", "seasonal", "resid"]
colors = ["tab:blue", "tab:orange", "tab:green", "tab:red"]
for axx, component, color in zip(ax, components, colors):
    data = getattr(decomposed, component)
    axx.plot(data, color=color)
    axx.text(*pos, component, bbox=dict(facecolor='white', alpha=0.8),
           transform=axx.transAxes, ha='center', va='top')
```




```
# %matplotlib widget

decomposed = decomposed_m

fig, ax = plt.subplots(1, 2, figsize=(10,6))
```

```
ax[0].plot(df['co2'], color="tab:blue", label="observed")
ax[0].plot(decomposed.trend * decomposed.resid, color="tab:orange", label="trend*resid")
ax[0].plot(decomposed.trend * decomposed.seasonal, color="tab:red", label="trend*seasonal")
ax[0].plot(decomposed.trend, color="black", label="trend")
ax[0].set(ylabel="CO$_2$ concentration (ppm)",
          title="Mauna Loa CO$_2$ concentration")
ax[0].legend(frameon=False)

start = "2000-01-01"
end = "2003-01-01"
zoom = slice(start, end)
ax[1].plot(df.loc[zoom, 'co2'], color="tab:blue", label="observed")
ax[1].plot((decomposed.trend * decomposed.resid)[zoom], color="tab:orange", label="trend*res
ax[1].plot((decomposed.trend * decomposed.seasonal)[zoom], color="tab:red", label="trend*sea
ax[1].plot(decomposed.trend[zoom], color="black", label="trend")
date_form = DateFormatter("%Y")
ax[1].xaxis.set_major_formatter(date_form)
ax[1].xaxis.set_major_locator(mdates.YearLocator(1))
ax[1].set_title("Components, 2000--2003");
```



46 Hilbert transform

Part IX

rates of change

47 motivation

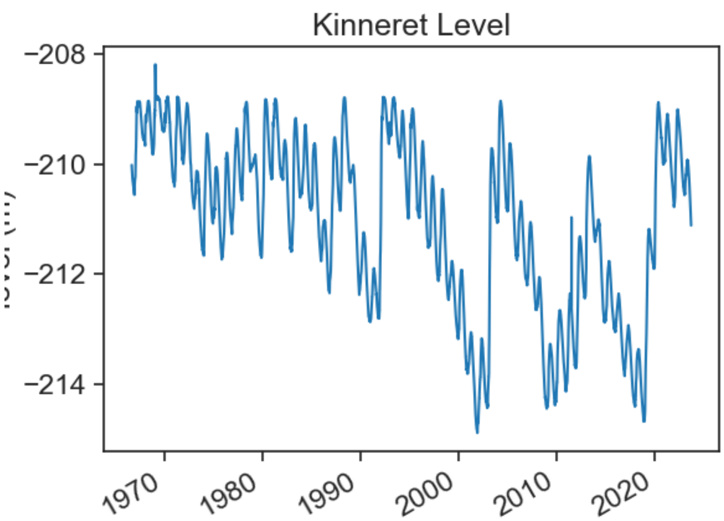
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
%matplotlib widget
```

```
filename = "../archive/data/kinneret_cleaned.csv"
df = pd.read_csv(filename)
df['date'] = pd.to_datetime(df['date'], dayfirst=True)
df = df.set_index('date')
df
```

	level
date	
2023-09-12	-211.115
2023-09-11	-211.105
2023-09-10	-211.095
2023-09-09	-211.085
2023-09-08	-211.070
...	...
1966-11-01	-210.390
1966-10-15	-210.320
1966-10-01	-210.270
1966-09-15	-210.130
1966-09-01	-210.020

```
fig, ax = plt.subplots()
ax.plot(df['level'], color="tab:blue")
ax.set(title="Kinneret Level",
```

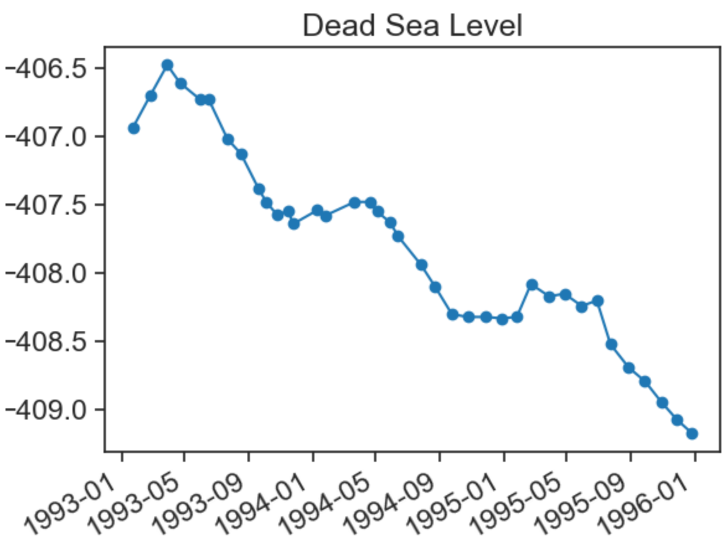
```
       ylabel="level (m)")
plt.gcf().autofmt_xdate()  # makes slanted dates
```



The data seems ok, until we take a closer look. Data points are not evenly spaced in time.

```
fig, ax = plt.subplots()
ax.plot(df.loc["1993":"1995", 'level'], color="tab:blue", marker="o")
ax.set(title="Dead Sea Level",
       ylabel="level (m)")
plt.gcf().autofmt_xdate()  # makes slanted dates
```

```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_3777/934261896.py:2: FutureWarning:
  ax.plot(df.loc["1993":"1995", 'level'], color="tab:blue", marker="o")
```



We can resample by day (a much higher rate than the original), and linearly interpolate:

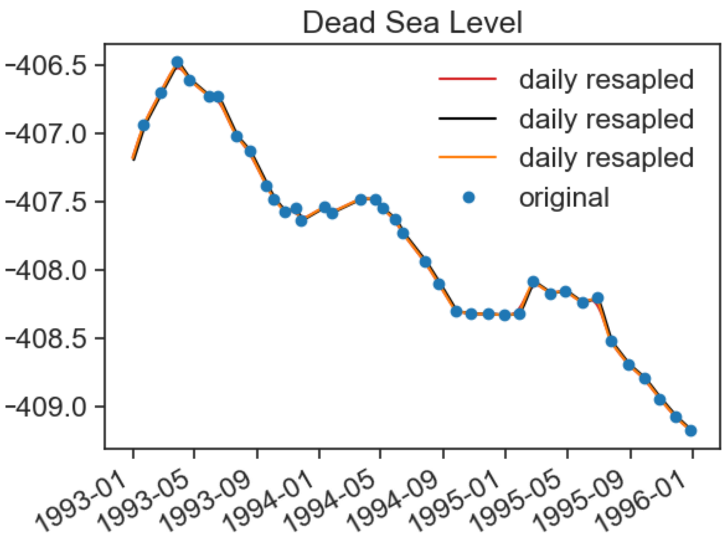
```
df2 = df['level'].resample('D').interpolate('time').to_frame()
df2['level_sm'] = df2['level'].rolling('30D', center=True).mean()
df3 = df2['level'].resample('W').mean().to_frame()
```

```
fig, ax = plt.subplots()
ax.plot(df2.loc["1993":"1995", 'level_sm'],
        color="tab:red",
        label="daily resapled")
ax.plot(df3.loc["1993":"1995", 'level'],
        color="black",
        label="daily resapled")
ax.plot(df2.loc["1993":"1995", 'level'],
        color="tab:orange",
        label="daily resapled")
ax.plot(df.loc["1993":"1995", 'level'],
        color="tab:blue",
        marker="o",
        linestyle="None",
```

```
         label="original")
ax.set(title="Dead Sea Level",
       ylabel="level (m)")
plt.gcf().autofmt_xdate()  # makes slanted dates
ax.legend(frameon=False)
```

```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_3777/2583247388.py:11: FutureWarning
  ax.plot(df.loc["1993":"1995", 'level'],
```

```
<matplotlib.legend.Legend at 0x7fa71e8b0bb0>
```

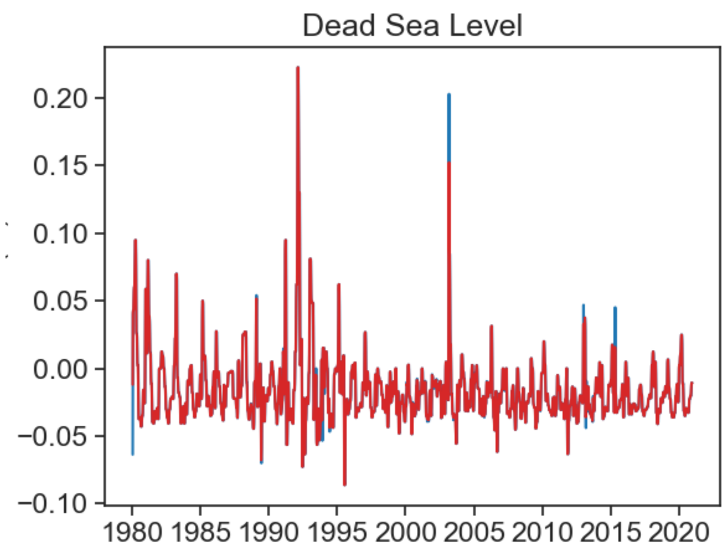



```
df2['naive'] = df2['level'].diff()
df2['gradient'] = np.gradient(df2['level'])

df3['naive'] = df3['level'].diff()
df3['gradient'] = np.gradient(df3['level'])
```

```
fig, ax = plt.subplots()
ax.plot(df3.loc["1980":"2020", 'naive'], color="tab:blue")
ax.plot(df3.loc["1980":"2020", 'gradient'], color="tab:red")
```

```
ax.set(title="Dead Sea Level",
       ylabel="level (m)")
```

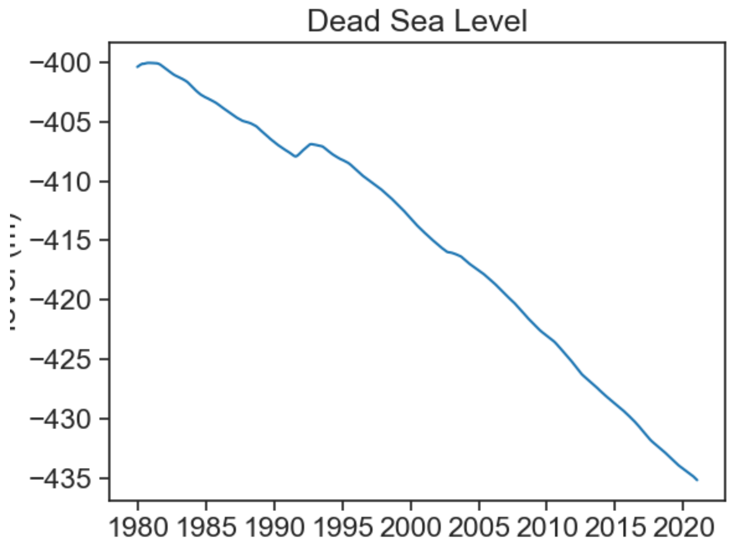
```
[Text(0.5, 1.0, 'Dead Sea Level'), Text(0, 0.5, 'level (m)')]
```




```
df3 = df2["level"].rolling('365.24D', center=True).mean().to_frame()
```

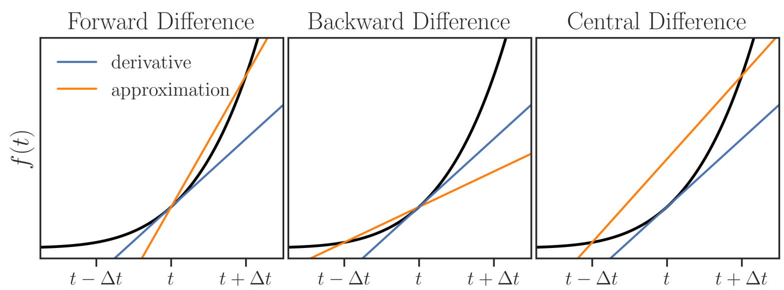
```
fig, ax = plt.subplots()
ax.plot(df3.loc["1980":"2020", 'level'], color="tab:blue")
ax.set(title="Dead Sea Level",
       ylabel="level (m)")
```

```
[Text(0.5, 1.0, 'Dead Sea Level'), Text(0, 0.5, 'level (m)')]
```


Dead Sea Level
−400
−405
−410
−415
−420
−425
−430
−435
1980 1985 1990 1995 2000 2005 2010 2015 2020

48 derivatives

49 finite differences



Definition of a **derivative**:

$$\underbrace{\dot{f} = f'(t) = \frac{df(t)}{dt}}_{\text{same thing}} = \lim_{\Delta t \to 0} \frac{f(t+\Delta t) - f(t)}{\Delta t}.$$

Numerically, we can approximate the derivative $f'(t)$ of a time series $f(t)$ as

$$\frac{df(t)}{dt} = \frac{f(t+\Delta t) - f(t)}{\Delta t} + \mathcal{O}(\Delta t). \tag{49.1}$$

The expression above is called the *two-point forward difference formula*. Likewise, we can define the *two-point backward difference formula*:

The expression $\mathcal{O}(\Delta t)$ means that the error associated with the approximation is proportional to Δt. This is called "Big O notation".

$$\frac{df(t)}{dt} = \frac{f(t) - f(t-\Delta t)}{\Delta t} + \mathcal{O}(\Delta t). \tag{49.2}$$

If we sum together Equation 49.1 and Equation 49.2 we get:

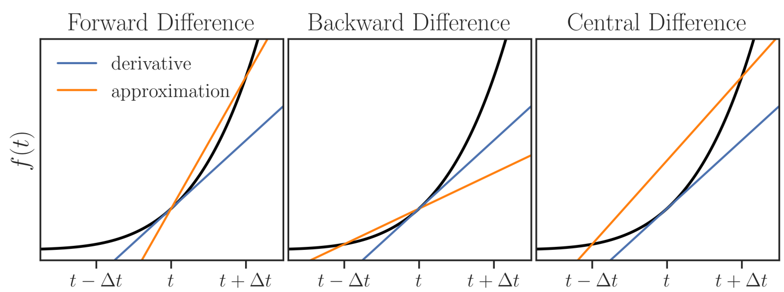
$$\begin{aligned} 2\frac{df(t)}{dt} &= \frac{f(t+\Delta t) - \cancel{f(t)}}{\Delta t} + \frac{\cancel{f(t)} - f(t-\Delta t)}{\Delta t} \\ &= \frac{f(t+\Delta t) - f(t-\Delta t)}{\Delta t}. \end{aligned} \tag{49.3}$$

Dividing both sides by 2 gives the *two-point central difference formula*:

$$\frac{df(t)}{dt} = \frac{f(t + \Delta t) - f(t - \Delta t)}{2\Delta t} + \mathcal{O}(\Delta t^2). \qquad (49.4)$$

Two things are worth mentioning about the approximation above:

1. it is balanced, that is, there is no preference of the future over the past.
2. its error is proportional to Δt^2, it is a lot more precise than the unbalanced approximations :)



To understand why the error is proportional to Δt^2, one can subtract the Taylor expansion of $f(t - \Delta t)$ from the Taylor expansion of $f(t + \Delta t)$. See this, pages 3 and 4.

The function `np.gradient` calculates the derivative using the central difference for points in the interior of the array, and uses the forward (backward) difference for the derivative at the beginning (end) of the array.

Check out this nice example.

The “gradient” usually refers to a first derivative with respect to space, and it is denoted as $\nabla f(x) = \frac{df(x)}{dx}$. However, it doesn’t really matter if we call the independent variable x or t, the derivative operator is exactly the same.

50 Fourier-based derivatives

This tutorial is based on Pelliccia (2019).

nice trick: https://math.stackexchange.com/questions/430858/fourier-transform-of-derivative

51 LOESS-based derivatives

Part X

forecasting

52 motivation

53 ARIMA

Part XI

assignments

54 assignment 1

This assignment comes right after the first session, where we discussed resampling. Read the whole instructions.

54.1 task

Go to the IMS website, and choose another weather station we have not worked with yet. Download 10-minute data for a full year, any year.

Make 3 graphs:

1. Daily maximum humidity. Bonus: add another line to the graph, the daily minimum humidity.
2. The number of rainy days for each month.
3. For each day of the year, show the number of hours when global solar radiation was above, on average, the threshold 10 W/m^2. Now add another line, for the threshold 500 W/m^2.

Make 5 more graphs (total of 8 graphs) of whatever you find interesting. You have the liberty to explore various facets of your dataset that capture your interest. It's essential, however, to maintain a focus on resampling. Each of your plots should effectively showcase and emphasize different aspects or techniques of resampling in your data analysis. To ensure diversity in your visualizations, avoid repetitive themes; for instance, if your first plot illustrates daily wind speed, then your second plot should not simply be a monthly resampling of wind speed. Aim for variety and innovation in each plot to fully explore the potential of resampling in data visualization.

You must download this Jupyter Notebook template. Create a zip file with your Jupyter notebook and with the `.csv` you used. Upload this zip file to the moodle task we created.

54.2 guidelines

1. Always name the axes and add units when relevant.
2. Always give a title to the plot.
3. Make sure that all axis tick labels (the numbers/dates on the axes) are readable.
4. Include a legend if you have multiple lines, colors, or groups.
5. Use appropriate scales for the axes (linear, logarithmic, etc.) depending on the data's nature.
6. Ensure that the plot is adequately sized for all elements to be clear and visible.
7. Choose colors and markers that are distinguishable, especially for plots with multiple elements.
8. If applicable, include error bars to indicate the variability or uncertainty in the data.
9. Use grid lines sparingly; they should not overshadow the data.

54.3 evaluation

All your assignments will be evaluated according to the following criteria:

- Presentation. How the graphs look, labels, general organization, markdown, clean code.
- Discussion. This is where you explain what you did, what you found out, etc.
- Depth of analysis. You can analyze/explore the data with different levels of complexity, this is where we take that into consideration.
- Replicability: Your code runs flawlessly.
- Code commenting. Explain in your code what you are doing, this is good for everyone, especially for yourself!

- Bonus: for originality, creative problem solving, or notable analysis.

55 assignment 2

55.1 Smoothing

In this assignment, you will delve into the application of different smoothing techniques on time series data. Utilizing meteorological data, your task is to create a series of plots that demonstrate the effects of various smoothing methods.

55.1.1 1. Comparative Smoothing Methods Analysis

- **Goal**: Showcase three smoothing techniques – Rolling Average, Savitzky-Golay, and Resampling – on the same time series data.
- **Task**: Overlay these methods over the actual data in a single plot. Ensure each method uses the same window size for consistency. **Describe in a few lines the differences you see.**

```
# code goes here
```

55.1.2 2. Rolling Average Window Size Impact

- **Goal**: Analyze the effect of varying window sizes on the Rolling Average method.
- **Task**: Produce a plot with three lines, each representing the Rolling Average with a different window size. **Describe in a few lines the differences you see.**

```
# code goes here
```

55.1.3 3. Savitzky-Golay Polynomial Order Variation

- **Goal**: Investigate how changing the polynomial order affects the Savitzky-Golay smoothing method.
- **Task**: Create a plot with three lines, where each represents the Savitzky-Golay method with a different polynomial order. **Describe in a few lines the differences you see.**

```
# code goes here
```

55.1.4 4. Kernel Shape Influence in Rolling Mean

- **Goal**: Explore the impact of different kernel shapes on the Rolling Mean.
- **Task**: Generate a plot displaying three lines, each using a different kernel shape in the Rolling Mean. We encorage to use unique kernel shapes that we did not showcase in class. See this list of kernels. **Describe in a few lines the differences you see.**

```
# code goes here
```

55.1.5 5. Moving Average with Confidence Interval

- **Goal**: Plot a Moving Average along with a 75% confidence interval.
- **Task**: Design a plot illustrating both the Moving Average and its 75% confidence interval.

```
# code goes here
```

Part XII

technical stuff

56 technical stuff

56.1 operating systems

I recommend working with UNIX-based operating systems (MacOS or Linux). Everything is easier.

If you use Windows, consider installing Linux on Windows with WSL.

56.2 software

Anaconda's Python distribution

VSCode

56.3 python packages

Kats — a one-stop shop for time series analysis
Developed by Meta

statsmodels statsmodels is a Python package that provides a complement to scipy for statistical computations including descriptive statistics and estimation and inference for statistical models.

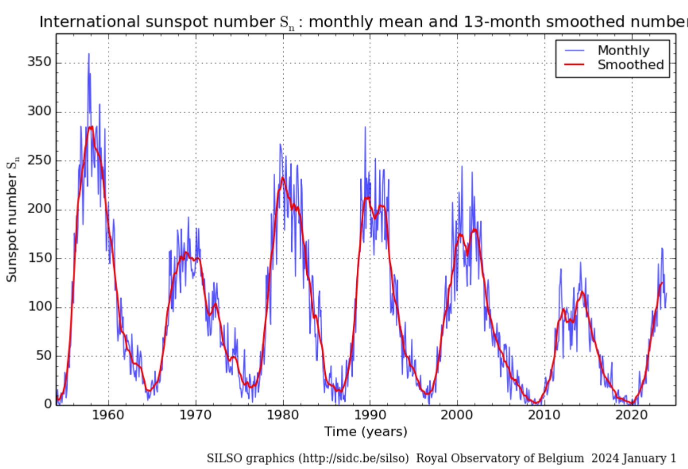
ydata-profiling
Quick Exploratory Data Analysis on time-series data. Read also this.

57 datasets

where to find data?

57.1 Sunspots

The solar cycle produces varying amounts of sunspots throughout the years.

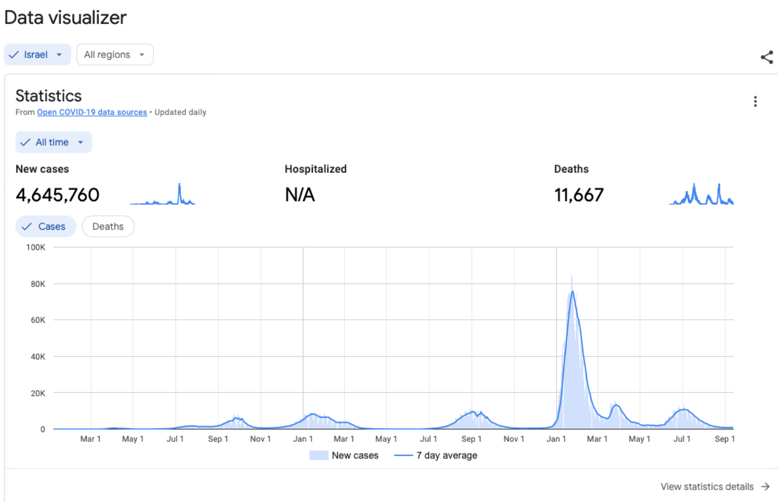


Download data from the Royal Observatory of Belgium.

Source: https://www.sidc.be/SILSO/monthlyssnplot

57.2 Covid-19 Open Data

Download the data into your own tools and systems to analyze the virus's spread or decline, investigate COVID-related deaths, study the effects of different vaccines, and more in 20,000-plus locations worldwide.



Click here to go to the download page. Choose desired region under section “Understanding the data”.

Source: https://health.google.com/covid-19/open-data/explorer

58 date formatting

Here you will find several examples of how to format dates in your plots. Not many explanations are provided.

How to use this page? Find first an example of a plot you like, only then go to the code and see how it's done.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import datetime
from datetime import timedelta
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
import matplotlib.gridspec as gridspec
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
```

```
import pandas as pd

start_date = '2018-01-01'
end_date = '2018-04-30'

# create date range with 1-hour intervals
dates = pd.date_range(start_date, end_date, freq='1H')
# create a random variable to plot
var = np.random.rand(len(dates)) - 0.51
var = var.cumsum()
var = var - var.min()
# create dataframe, make "date" the index
df = pd.DataFrame({'date': dates, 'variable': var})
df.set_index(df['date'], inplace=True)
```

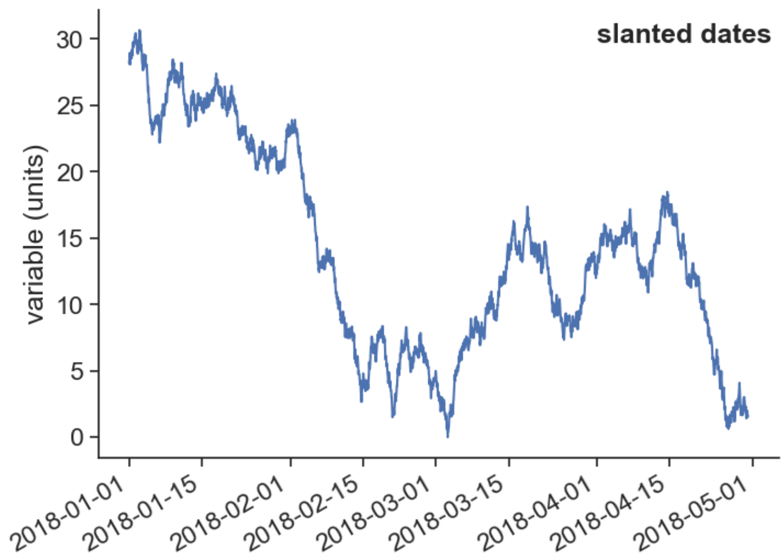
```
df
```

	date	variable
date		
2018-01-01 00:00:00	2018-01-01 00:00:00	28.317035
2018-01-01 01:00:00	2018-01-01 01:00:00	28.120523
2018-01-01 02:00:00	2018-01-01 02:00:00	28.596894
2018-01-01 03:00:00	2018-01-01 03:00:00	28.931941
2018-01-01 04:00:00	2018-01-01 04:00:00	28.561778
...	...	...
2018-04-29 20:00:00	2018-04-29 20:00:00	1.914343
2018-04-29 21:00:00	2018-04-29 21:00:00	1.648757
2018-04-29 22:00:00	2018-04-29 22:00:00	1.992956
2018-04-29 23:00:00	2018-04-29 23:00:00	1.500860
2018-04-30 00:00:00	2018-04-30 00:00:00	1.650439

define a useful function to plot the graphs below

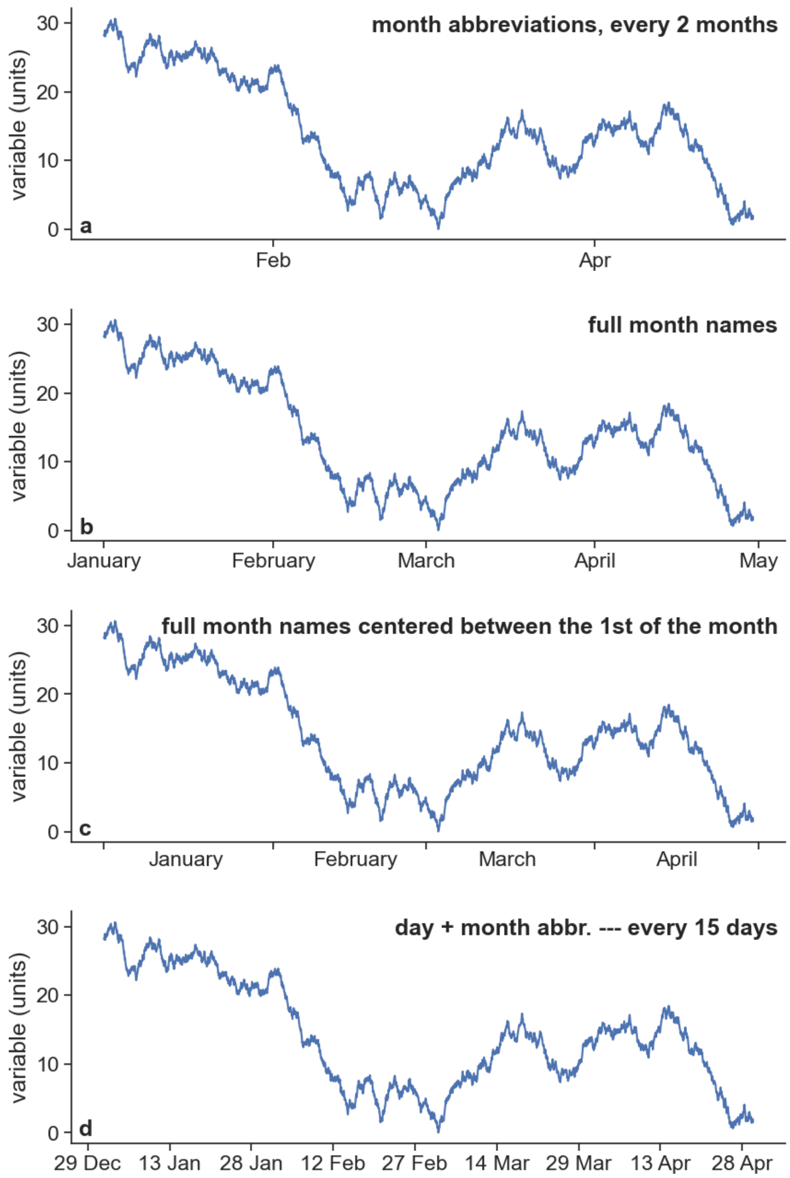
```
def explanation(ax, text, letter):
    ax.text(0.99, 0.97, text,
            transform=ax.transAxes,
            horizontalalignment='right', verticalalignment='top',
            fontweight="bold")
    ax.text(0.01, 0.01, letter,
            transform=ax.transAxes,
            horizontalalignment='left', verticalalignment='bottom',
            fontweight="bold")
    ax.set(ylabel="variable (units)")
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
ax.plot(df['variable'])
plt.gcf().autofmt_xdate()  # makes slated dates
explanation(ax, "slanted dates", "")
fig.savefig("dates1.png")
```



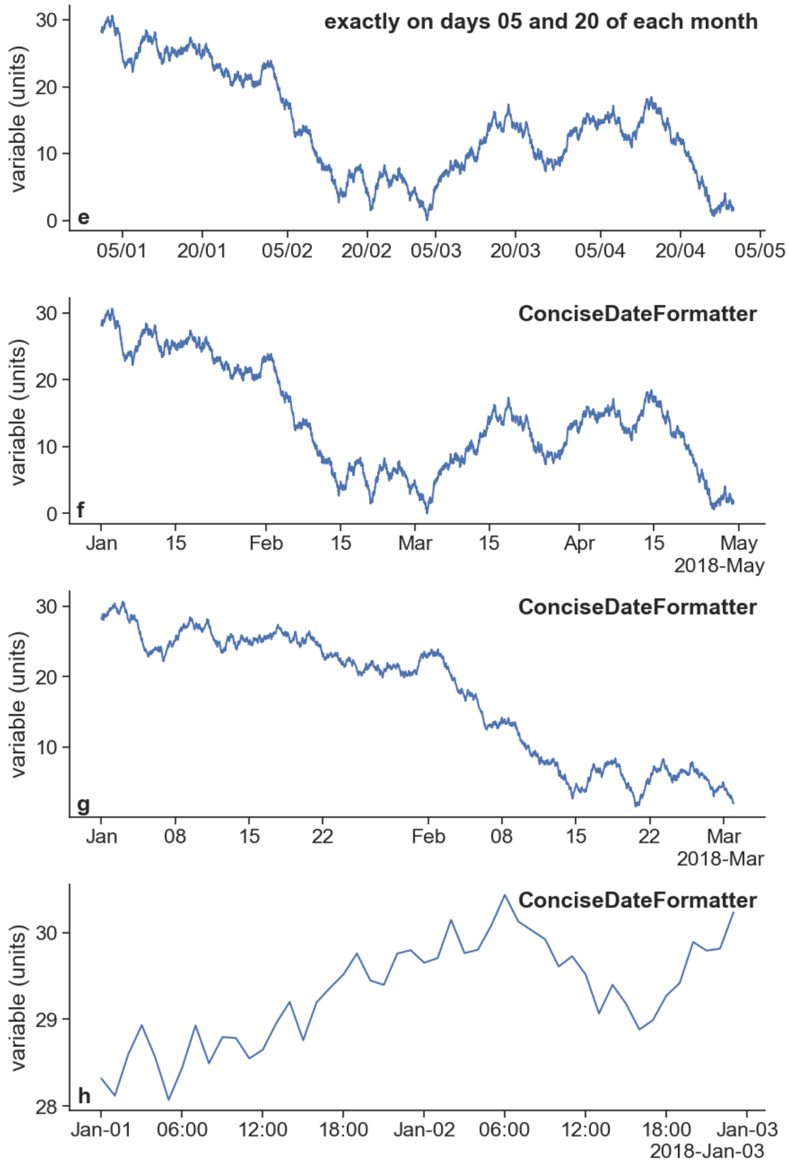

```
fig, ax = plt.subplots(4, 1, figsize=(10, 16),
                       gridspec_kw={'hspace': 0.3})

### plot a ###
ax[0].plot(df['variable'])
date_form = DateFormatter("%b")
ax[0].xaxis.set_major_locator(mdates.MonthLocator(interval=2))
ax[0].xaxis.set_major_formatter(date_form)

### plot b ###
ax[1].plot(df['variable'])
date_form = DateFormatter("%B")
ax[1].xaxis.set_major_locator(mdates.MonthLocator(interval=1))
ax[1].xaxis.set_major_formatter(date_form)

### plot c ###
ax[2].plot(df['variable'])
ax[2].xaxis.set_major_locator(mdates.MonthLocator())
# 16 is a slight approximation for the center, since months differ in number of days.
ax[2].xaxis.set_minor_locator(mdates.MonthLocator(bymonthday=16))
ax[2].xaxis.set_major_formatter(ticker.NullFormatter())
ax[2].xaxis.set_minor_formatter(DateFormatter('%B'))
```

```
for tick in ax[2].xaxis.get_minor_ticks():
    tick.tick1line.set_markersize(0)
    tick.tick2line.set_markersize(0)
    tick.label1.set_horizontalalignment('center')

### plot d ###
ax[3].plot(df['variable'])
date_form = DateFormatter("%d %b")
ax[3].xaxis.set_major_locator(mdates.DayLocator(interval=15))
ax[3].xaxis.set_major_formatter(date_form)

explanation(ax[0], "month abbreviations, every 2 months", "a")
explanation(ax[1], "full month names", "b")
explanation(ax[2], "full month names centered between the 1st of the month", "c")
explanation(ax[3], "day + month abbr. --- every 15 days", "d")

fig.savefig("dates2.png")
```



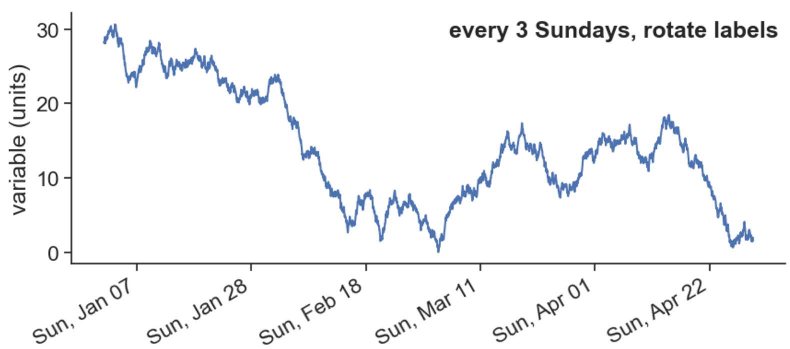

```
fig, ax = plt.subplots(4, 1, figsize=(10, 16),
                       gridspec_kw={'hspace': 0.3})

### plot e ###
ax[0].plot(df['variable'])
```

```
date_form = DateFormatter("%d/%m")
ax[0].xaxis.set_major_locator(mdates.DayLocator(bymonthday=[5, 20]))
ax[0].xaxis.set_major_formatter(date_form)

### plot f ###
ax[1].plot(df['variable'])
locator = mdates.AutoDateLocator(minticks=11, maxticks=17)
formatter = mdates.ConciseDateFormatter(locator)
ax[1].xaxis.set_major_locator(locator)
ax[1].xaxis.set_major_formatter(formatter)

### plot g ###
ax[2].plot(df.loc['2018-01-01':'2018-03-01', 'variable'])
locator = mdates.AutoDateLocator(minticks=6, maxticks=14)
formatter = mdates.ConciseDateFormatter(locator)
ax[2].xaxis.set_major_locator(locator)
ax[2].xaxis.set_major_formatter(formatter)

### plot h ###
ax[3].plot(df.loc['2018-01-01':'2018-01-02', 'variable'])
locator = mdates.AutoDateLocator(minticks=6, maxticks=10)
formatter = mdates.ConciseDateFormatter(locator)
ax[3].xaxis.set_major_locator(locator)
ax[3].xaxis.set_major_formatter(formatter)

explanation(ax[0], "exactly on days 05 and 20 of each month", "e")
explanation(ax[1], "ConciseDateFormatter", "f")
explanation(ax[2], "ConciseDateFormatter", "g")
explanation(ax[3], "ConciseDateFormatter", "h")

fig.savefig("dates3.png")
```




```
fig, ax = plt.subplots(1, 1, figsize=(10, 4),
                       gridspec_kw={'hspace': 0.3})

# import constants for the days of the week
from matplotlib.dates import MO, TU, WE, TH, FR, SA, SU
ax.plot(df['variable'])
```

```
# tick on sundays every third week
loc = mdates.WeekdayLocator(byweekday=SU, interval=3)
ax.xaxis.set_major_locator(loc)
date_form = DateFormatter("%a, %b %d")
ax.xaxis.set_major_formatter(date_form)
fig.autofmt_xdate(bottom=0.2, rotation=30, ha='right')
explanation(ax, "every 3 Sundays, rotate labels", "")
```



Code	Explanation
%Y	4-digit year (e.g., 2022)
%y	2-digit year (e.g., 22)
%m	2-digit month (e.g., 12)
%B	Full month name (e.g., December)
%b	Abbreviated month name (e.g., Dec)
%d	2-digit day of the month (e.g., 09)
%A	Full weekday name (e.g., Tuesday)
%a	Abbreviated weekday name (e.g., Tue)
%H	24-hour clock hour (e.g., 23)
%I	12-hour clock hour (e.g., 11)
%M	2-digit minute (e.g., 59)
%S	2-digit second (e.g., 59)
%p	"AM" or "PM"
%Z	Time zone name
%z	Time zone offset from UTC (e.g., -0500)

59 sources

59.1 books

from Data to Viz

Fundamentals of Data Visualization, by Claus O. Wilke

PyNotes in Agriscience

Forecasting: Principles and Practice (3rd ed), by Rob J Hyndman and George Athanasopoulos

Python for Finance Cookbook 2nd Edition - Code Repository

Practical time series analysis,: prediction with statistics and machine learning, by Aileen Nielsen
The online edition of this book is available for Hebrew University staff and students.

Time series analysis with Python cookbook : practical recipes for exploratory data analysis, data preparation, forecasting, and model evaluation, by Tarek A. Atwan
The online edition of this book is available for Hebrew University staff and students.

Hands-on Time Series Analysis with Python: From Basics to Bleeding Edge Techniques, by B V Vishwas, Ashish Patel
The online edition of this book is available for Hebrew University staff and students.

59.2 videos

Times Series Analysis for Everyone, by Bruno Goncalves
This series is available for Hebrew University staff and students.

Time Series Analysis with Pandas, by Joshua Malina This video is available for Hebrew University staff and students.

59.3 references

Brockwell, Peter J., and Richard A. Davis. 2016. *Introduction to Time Series and Forecasting*. 3rd ed. Springer.

Chatfield, C. 2016. *The Analysis of Time Series: An Introduction, Sixth Edition*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press.

McDonald, Andy. 2022. "Creating Boxplots with the Seaborn Python Library." *Medium*. Towards Data Science. https://towardsdatascience.com/creating-boxplots-with-the-seaborn-python-library-f0c20f09bd57.

Pelliccia, Daniel. 2019. "Fourier Spectral Smoothing Method." 2019. https://nirpyresearch.com/fourier-spectral-smoothing-method/.

Shumway, Robert H., and David S. Stoffer. 2017. *Time Series Analysis and Its Applications With R Examples*. 4th ed. Springer. http://www.stat.ucla.edu/~frederic/415/S23/tsa4.pdf.

Tsay, R. S. 2010. *Analysis of Financial Time Series*. Wiley.

Zhang, Ou. 2020. "Outliers-Part 3:outliers in Regression." ouzhang.me. https://ouzhang.me/blog/outlier-series/outliers-part3/.

Part XIII

behind-the-scenes

sliding window video

Import packages and stuff.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()  # datetime converter for a matplotlib
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import scipy as sp
import json
import requests
import os
import subprocess
from tqdm import tqdm
from scipy import signal

# avoid "SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a Dat
pd.options.mode.chained_assignment = None  # default='warn'
```

Download data from the IMS using an API.

```
# read token from file
with open('../archive/IMS-token.txt', 'r') as file:
    TOKEN = file.readline()
# 28 = SHANI station
STATION_NUM = 28
start = "2022/01/01"
end = "2022/01/07"
```

```python
filename = 'shani_2022_january.json'

# check if the JSON file already exists
# if so, then load file
if os.path.exists(filename):
    with open(filename, 'r') as json_file:
        data = json.load(json_file)
else:
    # make the API request if the file doesn't exist
    url = f"https://api.ims.gov.il/v1/envista/stations/{STATION_NUM}/data/?from={start}&to={
    headers = {'Authorization': f'ApiToken {TOKEN}'}
    response = requests.get(url, headers=headers)
    data = json.loads(response.text.encode('utf8'))

    # save the JSON data to a file
    with open(filename, 'w') as json_file:
        json.dump(data, json_file)
# show data to see if it's alright
# data
```

Load and process data.

```python
df = pd.json_normalize(data['data'],record_path=['channels'], meta=['datetime'])
df['date'] = (pd.to_datetime(df['datetime'])
                .dt.tz_localize(None)  # ignores time zone information
             )
df = df.pivot(index='date', columns='name', values='value')
df
```

name date	Grad	RH	Rain	STDwd	TD	TDmax	TDmin	TG	TW	Time	WD
2022-01-01 00:00:00	0.0	77.0	0.0	10.3	11.2	11.2	11.1	10.7	-9999.0	2354.0	75.0
2022-01-01 00:10:00	0.0	77.0	0.0	11.2	11.2	11.2	11.1	10.8	-9999.0	1.0	77.0
2022-01-01 00:20:00	0.0	75.0	0.0	10.0	11.4	11.5	11.2	10.9	-9999.0	20.0	80.0
2022-01-01 00:30:00	0.0	74.0	0.0	9.6	11.5	11.5	11.4	11.0	-9999.0	22.0	76.0
2022-01-01 00:40:00	0.0	73.0	0.0	9.1	11.6	11.7	11.5	11.1	-9999.0	34.0	74.0
...	...	...	...	...	...	...	...	...	...	...	...
2022-01-06 23:10:00	0.0	36.0	0.0	16.1	11.6	12.0	11.1	6.8	-9999.0	2310.0	144.0
2022-01-06 23:20:00	0.0	35.0	0.0	10.1	12.1	12.3	11.9	6.3	-9999.0	2320.0	118.0
2022-01-06 23:30:00	0.0	36.0	0.0	7.1	12.4	12.6	11.9	7.3	-9999.0	2330.0	113.0

name	Grad	RH	Rain	STDwd	TD	TDmax	TDmin	TG	TW	Time	WD
date											
2022-01-06 23:40:00	0.0	37.0	0.0	5.6	12.6	12.7	12.5	7.8	-9999.0	2339.0	119.0
2022-01-06 23:50:00	0.0	39.0	0.0	11.5	11.9	12.6	11.5	7.1	-9999.0	2341.0	102.0

Define useful functions.

```
def concise(ax):
    """
    Let python choose the best xtick labels for you
    """
    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)
    ax.xaxis.set_major_locator(locator)
    ax.xaxis.set_major_formatter(formatter)

# dirty trick to have dates in the middle of the 24-hour period
# make minor ticks in the middle, put the labels there!
# from https://matplotlib.org/stable/gallery/ticks/centered_ticklabels.html

def center_dates(ax):
    # show day of the month + month abbreviation. see full option list here:
    # https://strftime.org
    date_form = DateFormatter("%d %b")
    # major ticks at midnight, every day
    ax.xaxis.set_major_locator(mdates.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(date_form)
    # minor ticks at noon, every day
    ax.xaxis.set_minor_locator(mdates.HourLocator(byhour=[12]))
    # erase major tick labels
    ax.xaxis.set_major_formatter(ticker.NullFormatter())
    # set minor tick labels as define above
    ax.xaxis.set_minor_formatter(date_form)
    # completely erase minor ticks, center tick labels
    for tick in ax.xaxis.get_minor_ticks():
        tick.tick1line.set_markersize(0)
        tick.tick2line.set_markersize(0)
        tick.label1.set_horizontalalignment('center')
```

```
def center_dates_two_panels(ax0, ax1):
    # show day of the month + month abbreviation. see full option list here:
    date_form = DateFormatter("%d %b")
    # major ticks at midnight, every day
    ax0.xaxis.set_major_locator(mdates.DayLocator(interval=1))
    ax1.xaxis.set_major_locator(mdates.DayLocator(interval=1))
    ax1.xaxis.set_major_formatter(date_form)
    # minor ticks at noon, every day
    ax1.xaxis.set_minor_locator(mdates.HourLocator(byhour=[12]))
    # erase major tick labels
    ax1.xaxis.set_major_formatter(ticker.NullFormatter())
    # set minor tick labels as define above
    ax1.xaxis.set_minor_formatter(date_form)
    # completely erase minor ticks, center tick labels
    for tick in ax0.xaxis.get_minor_ticks():
        tick.tick1line.set_markersize(0)
        tick.tick2line.set_markersize(0)
    for tick in ax1.xaxis.get_minor_ticks():
        tick.tick1line.set_markersize(0)
        tick.tick2line.set_markersize(0)
        tick.label1.set_horizontalalignment('center')
```

We don't need the full month, let's cut the dataframe to fewer days.

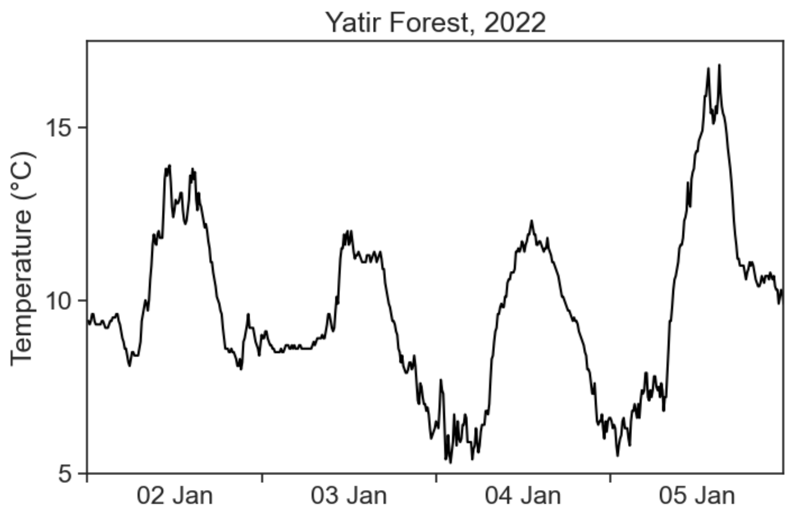
```
start = "2022-01-01 00:00:00"
end = "2022-01-06 23:50:00"
df = df.loc[start:end]
```

We now redefine a narrower window, this will be the graph's xlimits. We leave the dataframe as is, because we will need some data outside the graph's limits.

```
start = "2022-01-02 00:00:00"
end = "2022-01-05 23:50:00"
```

```
fig, ax = plt.subplots(figsize=(8,5))
ax.plot(df.loc[start:end, 'TD'], color='black')
ax.set(ylim=[5, 17.5],
```

```
        xlim=[start, end],
        ylabel="Temperature (°C)",
        title="Yatir Forest, 2022",
        yticks=[5,10,15])
center_dates(ax)
fig.savefig("sliding_YF_temperature_2022.png")
```



Looks good. Let's move on.

59.4 Rectangular kernel

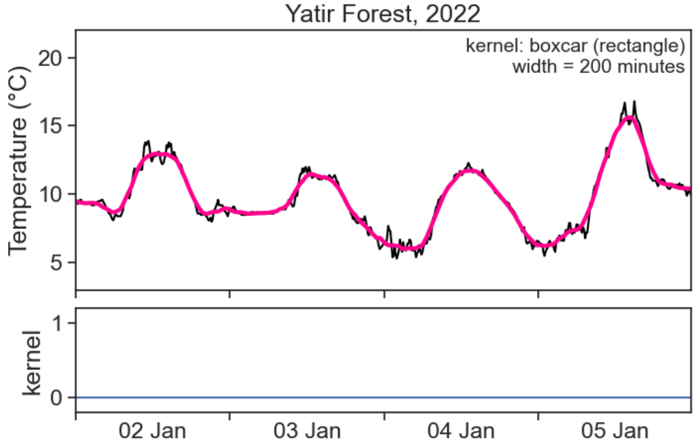
```
%matplotlib widget
fig, ax = plt.subplots(2, 1, figsize=(8,5), sharex=True,
                       gridspec_kw={'height_ratios':[1,0.4], 'hspace':0.1})

class Lines:
    """
    empty class, later will be populated with graph objects.
    this is useful to draw and erase lines on demand.
    """
    pass
lines = Lines()
```

```
# rename axes for convenience
ax0 = ax[0]
ax1 = ax[1]
# sm = df['TD'].rolling(10, center=True).mean()
# ga = df['TD'].rolling(10, center=True, win_type="gaussian").mean(std=100.0)

# set graph y limits
ylim = [3, 22]
# choose here windown width in minutes
window_width_min = 200.0
window_width_min_integer = int(window_width_min)  # same but integer
window_width_int = int(window_width_min // 10 + 1)  # window width in points
N = len(df)  # df length
# time range over which the kernel will slide
# starts at "start", minus the width of the window,
# minus half an hour, so that the window doesn't start sliding right away at the beginning o
# ends an hour after the window has finished sliding
t_swipe = pd.date_range(start=pd.to_datetime(start) - pd.Timedelta(minutes=window_width_min)
                        end=pd.to_datetime(end) + pd.Timedelta(minutes=60),
                        freq="10min")
# starting time
t0 = t_swipe[0]
# show sliding window on the top panel as a light blue shade
lines.fill_bet = ax0.fill_between([t0, t0 + pd.Timedelta(minutes=window_width_min)],
                                  y1=ylim[0], y2=ylim[1], alpha=0.1, zorder=-1)
# this is our "boxcart" kernel (a rectangle)
kernel_rect = np.ones(window_width_int)
# calculate the moving average with "kernel_rect" as weights
# this is the same as a convolution, which is just faster to compute
df.loc[:, 'con'] = np.convolve(df['TD'].values, kernel_rect, mode='same') / len(kernel_rect)
# create a new column for the kernel, fill it with zeros
df['kernel_plus'] = 0.0
# populate the kernel column with the window at the very beginning
df.loc[t0: t0 + pd.Timedelta(minutes=window_width_min), 'kernel_plus'] = kernel_rect
# plot kernel on the bottom panel
lines.kernel_line, = ax1.plot(df['kernel_plus'])
# plot temperature on the top panel
ax0.plot(df.loc[start:end, 'TD'], color="black")
# make temperature look gray when inside the sliding window
```

```
lines.gray_line, = ax0.plot(df.loc[df['kernel_plus']==1.0, 'TD'],
                     color=[0.6]*3, lw=3)
# calculate the middle of the sliding window
window_middle = t0 + pd.Timedelta(minutes=window_width_min/2)
# plot a pink line showing the result of the moving average
# from the beginning to the middle of the sliding window
lines.pink_line, = ax0.plot(df.loc[start:window_middle, 'con'], color="xkcd:hot pink", lw=3)
# emphasize the location of the middle on the window with a circle
lines.pink_circle, = ax0.plot([window_middle], [df.loc[window_middle, 'con']],
         marker='o', markerfacecolor="None", markeredgecolor="xkcd:dark pink", markeredgewid
         markersize=8)
# some explanation
ax0.text(0.99, 0.97, f"kernel: boxcar (rectangle)\nwidth = {window_width_min:.0f} minutes",
         horizontalalignment='right', verticalalignment='top',
         fontsize=14)
# axis tweaking
ax0.set(ylim=ylim,
        xlim=[start, end],
        ylabel="Temperature (°C)",
        yticks=[5,10,15,20],
        title="Yatir Forest, 2022")
ax1.set(ylim=[-0.2, 1.2],
        xlim=[start, end],
        ylabel="kernel"
       )
# adjust dates on both panels as defined before
center_dates_two_panels(ax0, ax1)

def update_swipe(k, lines):
    """
    updates both panels, given the index k along which the window is sliding
    """
    # left side of the sliding window
    t0 = t_swipe[k]
    # middle position
    window_middle = t0 + pd.Timedelta(minutes=window_width_min/2)
    # erase previous blue shade on the top graph
    lines.fill_bet.remove()
    # fill again the blue shade in the updated window position
```

```
    lines.fill_bet = ax0.fill_between([t0, t0 + pd.Timedelta(minutes=window_width_min)],
                                         y1=ylim[0], y2=ylim[1], alpha=0.1, zorder=-1,
    # update pink curve
    lines.pink_line.set_data(df[start:window_middle].index,
                             df.loc[start:window_middle, 'con'].values)
    # update pink circle
    lines.pink_circle.set_data([window_middle], [df.loc[window_middle, 'con']])
    # update the kernel in its current position
    lines.kernel_rect = np.ones(window_width_int)
    df.loc[:, 'kernel_plus'] = 0.0
    df.loc[t0: t0 + pd.Timedelta(minutes=window_width_min), 'kernel_plus'] = kernel_rect
    # update gray line
    lines.gray_line.set_data(df.loc[df['kernel_plus']==1.0, 'TD'].index,
                             df.loc[df['kernel_plus']==1.0, 'TD'].values)
    # update kernel line
    lines.kernel_line.set_data(df['kernel_plus'].index, df['kernel_plus'].values)

# create a tqdm progress bar
progress_bar = tqdm(total=len(t_swipe), unit="iteration")
# loop over all sliding indices, update graph and then save it
for fignum, i in enumerate(np.arange(0, len(t_swipe)-1, 1)):
    update_swipe(i, lines)
    fig.savefig(f"pngs/boxcar{window_width_min_integer}/boxcar_{window_width_min_integer}min
    # update the progress bar
    progress_bar.update(1)
# close the progress bar
progress_bar.close()
```

```
100%|     | 604/605 [05:27<00:00,  1.85iteration/s]
```



Combine all saved images into one mp4 video.

```
# Define the path to your PNG images
pngs_path = f"pngs/boxcar{window_width_min_integer}"
pngs_name = f"boxcar_{window_width_min_integer}min_%03d.png"

# Define the output video file path
video_output = f"output{window_width_min_integer}.mp4"

# Use ffmpeg to create a video from PNG images
# desired framerate. choose 24 if you don't know what to do
fr = 12
# run command
ffmpeg_cmd = f"ffmpeg -framerate {fr} -i {pngs_path}/{pngs_name} -c:v libx264 -vf fps={fr} {
subprocess.run(ffmpeg_cmd, shell=True)
```

```
ffmpeg version 6.0 Copyright (c) 2000-2023 the FFmpeg developers
  built with Apple clang version 14.0.3 (clang-1403.0.22.14.1)
  configuration: --prefix=/usr/local/Cellar/ffmpeg/6.0 --enable-shared --enable-pthreads --enal
  libavutil      58.  2.100 / 58.  2.100
  libavcodec     60.  3.100 / 60.  3.100
  libavformat    60.  3.100 / 60.  3.100
  libavdevice    60.  1.100 / 60.  1.100
  libavfilter     9.  3.100 /  9.  3.100
  libswscale      7.  1.100 /  7.  1.100
```

```
  libswresample   4. 10.100 /  4. 10.100
  libpostproc    57.  1.100 / 57.  1.100
Input #0, image2, from 'pngs/boxcar200/boxcar_200min_%03d.png':
  Duration: 00:00:50.33, start: 0.000000, bitrate: N/A
  Stream #0:0: Video: png, rgba(pc), 4800x3000 [SAR 23622:23622 DAR 8:5], 12 fps, 12 tbr, 12 t
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> h264 (libx264))
Press [q] to stop, [?] for help
[libx264 @ 0x7fa027f2e300] using SAR=1/1
[libx264 @ 0x7fa027f2e300] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AV
[libx264 @ 0x7fa027f2e300] profile High 4:4:4 Predictive, level 6.0, 4:4:4, 8-bit
[libx264 @ 0x7fa027f2e300] 264 - core 164 r3095 baee400 - H.264/MPEG-4 AVC codec - Copyleft 20
Output #0, mp4, to 'output200.mp4':
  Metadata:
    encoder         : Lavf60.3.100
  Stream #0:0: Video: h264 (avc1 / 0x31637661), yuv444p(tv, progressive), 4800x3000 [SAR 1:1 D
    Metadata:
      encoder         : Lavc60.3.100 libx264
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: N/A
frame=  604 fps= 23 q=-1.0 Lsize=    1412kB time=00:00:50.08 bitrate= 231.0kbits/s speed=1.91x
video:1404kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 0.56
[libx264 @ 0x7fa027f2e300] frame I:3     Avg QP: 9.98  size:135751
[libx264 @ 0x7fa027f2e300] frame P:154   Avg QP:14.75  size:  2507
[libx264 @ 0x7fa027f2e300] frame B:447   Avg QP:22.66  size:  1440
[libx264 @ 0x7fa027f2e300] consecutive B-frames:  1.0%  0.7%  1.0% 97.4%
[libx264 @ 0x7fa027f2e300] mb I  I16..4: 55.5% 38.8%  5.7%
[libx264 @ 0x7fa027f2e300] mb P  I16..4:  0.4%  0.3%  0.0%  P16..4:  0.2%  0.1%  0.0%  0.0%  0
[libx264 @ 0x7fa027f2e300] mb B  I16..4:  0.1%  0.0%  0.0%  B16..8:  1.0%  0.2%  0.0%  direct:
[libx264 @ 0x7fa027f2e300] 8x8 transform intra:37.1% inter:49.0%
[libx264 @ 0x7fa027f2e300] coded y,u,v intra: 3.6% 0.4% 0.6% inter: 0.1% 0.0% 0.0%
[libx264 @ 0x7fa027f2e300] i16 v,h,dc,p: 90% 10%  0%  0%
[libx264 @ 0x7fa027f2e300] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 41%  3% 56%  0%  0%  0%  0%  0%  0%
[libx264 @ 0x7fa027f2e300] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 51% 14% 20%  3%  2%  3%  2%  3%  2%
[libx264 @ 0x7fa027f2e300] Weighted P-Frames: Y:0.0% UV:0.0%
[libx264 @ 0x7fa027f2e300] ref P L0: 56.6%  3.8% 28.4% 11.3%
[libx264 @ 0x7fa027f2e300] ref B L0: 85.6% 13.4%  1.0%
[libx264 @ 0x7fa027f2e300] ref B L1: 95.9%  4.1%
[libx264 @ 0x7fa027f2e300] kb/s:228.44

CompletedProcess(args='ffmpeg -framerate 12 -i pngs/boxcar200/boxcar_200min_%03d.png -c:v libx
```

The following code does exactly as you see above, but it is not well commented. You are an intelligent person, you'll figure this out.

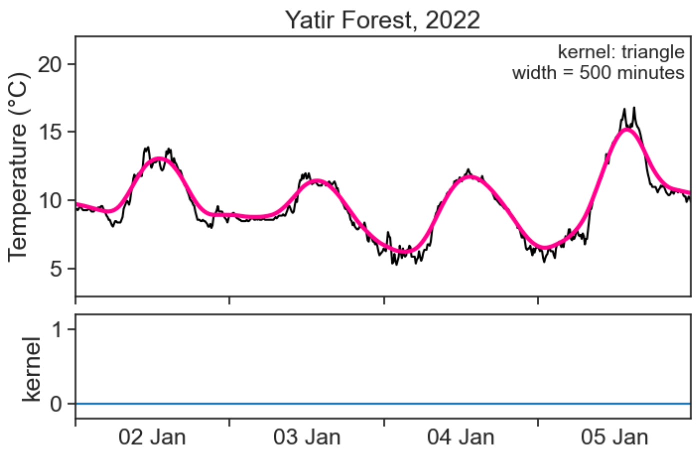
59.5 Triangular kernel

```
%matplotlib widget
fig, ax = plt.subplots(2, 1, figsize=(8,5), sharex=True,
                       gridspec_kw={'height_ratios':[1,0.4], 'hspace':0.1})

class Lines:
    pass
lines = Lines()

ax0 = ax[0]
ax1 = ax[1]
ylim = [3, 22]
window_width_min = 500.0
window_width_int = int(window_width_min / 10) + 1
N = len(df)
t_swipe = pd.date_range(start=pd.to_datetime(start) - pd.Timedelta(minutes=window_width_min)
                        end=pd.to_datetime(end) + pd.Timedelta(minutes=60),
                        freq="10min")
t0 = t_swipe[200]
window_middle = t0 + pd.Timedelta(minutes=window_width_min/2)
# fill between blue shade, plot kernel
lines.fill_bet = ax0.fill_between([t0, t0 + pd.Timedelta(minutes=window_width_min)],
                                  y1=ylim[0], y2=ylim[1], alpha=0.1, zorder=-1)
half_triang = np.arange(1, window_width_int/2+1, 1)
kernel_triang = np.hstack([half_triang, half_triang[-2::-1]])
kernel_triang = kernel_triang / kernel_triang.max()
df.loc[:, 'con'] = np.convolve(df['TD'].values, kernel_triang, mode='same') / len(kernel_tri
df['kernel_plus'] = 0.0
df.loc[t0: t0 + pd.Timedelta(minutes=window_width_min), 'kernel_plus'] = kernel_triang
lines.kernel_line, = ax1.plot(df['kernel_plus'], color="tab:blue")
ax0.plot(df.loc[start:end, 'TD'], color="black")
lines.gray_line, = ax0.plot(df.loc[df['kernel_plus']!=0.0, 'TD'],
                     color=[0.6]*3, lw=3)
```

```
lines.pink_line, = ax0.plot(df.loc[start:window_middle, 'con'], color="xkcd:hot pink", lw=3)
lines.pink_circle, = ax0.plot([window_middle], [df.loc[window_middle, 'con']],
         marker='o', markerfacecolor="None", markeredgecolor="xkcd:dark pink", markeredgewid
         markersize=8)
ax0.text(0.99, 0.97, f"kernel: triangle\nwidth = {window_width_min:.0f} minutes", transform=
         horizontalalignment='right', verticalalignment='top',
         fontsize=14)
ax0.set(ylim=ylim,
        xlim=[start, end],
        ylabel="Temperature (°C)",
        yticks=[5,10,15,20],
        title="Yatir Forest, 2022")
ax1.set(ylim=[-0.2, 1.2],
        xlim=[start, end],
        ylabel="kernel"
       )
center_dates_two_panels(ax0, ax1)

def update_swipe(k, lines):
    t0 = t_swipe[k]
    window_middle = t0 + pd.Timedelta(minutes=window_width_min/2)
    lines.fill_bet.remove()
    lines.fill_bet = ax0.fill_between([t0, t0 + pd.Timedelta(minutes=window_width_min)],
                                              y1=ylim[0], y2=ylim[1], alpha=0.1, zorder=-1,
    lines.pink_line.set_data(df[start:window_middle].index,
                             df.loc[start:window_middle, 'con'].values)
    lines.pink_circle.set_data([window_middle], [df.loc[window_middle, 'con']])
    lines.kernel_rect = np.ones(window_width_int)
    df['kernel_plus'] = 0.0
    df.loc[t0: t0 + pd.Timedelta(minutes=window_width_min), 'kernel_plus'] = kernel_triang
    lines.gray_line.set_data(df.loc[df['kernel_plus']!=0.0,'TD'].index,
                             df.loc[df['kernel_plus']!=0.0,'TD'].values)
    lines.kernel_line.set_data(df['kernel_plus'].index, df['kernel_plus'].values)

progress_bar = tqdm(total=len(t_swipe), unit="iteration")
for fignum, i in enumerate(np.arange(0, len(t_swipe)-1, 1)):
    update_swipe(i, lines)
    fig.savefig(f"pngs/triangle/triangle_{fignum:03}.png", dpi=600)
    # update the progress bar
```

```
    progress_bar.update(1)
# close the progress bar
progress_bar.close()
```

```
100%|      | 634/635 [05:35<00:00,  1.89iteration/s]
```




```
# Define the path to your PNG images
pngs_path = "pngs/triangle"
pngs_name = "triangle_%03d.png"

# Define the output video file path
video_output = "output_triangle.mp4"

fr = 12
# run command
ffmpeg_cmd = f"ffmpeg -framerate {fr} -i {pngs_path}/{pngs_name} -c:v libx264 -vf fps={fr} {
subprocess.run(ffmpeg_cmd, shell=True)
```

```
ffmpeg version 6.0 Copyright (c) 2000-2023 the FFmpeg developers
  built with Apple clang version 14.0.3 (clang-1403.0.22.14.1)
  configuration: --prefix=/usr/local/Cellar/ffmpeg/6.0 --enable-shared --enable-pthreads --enal
  libavutil      58.  2.100 / 58.  2.100
  libavcodec     60.  3.100 / 60.  3.100
```

```
  libavformat    60.  3.100 / 60.  3.100
  libavdevice    60.  1.100 / 60.  1.100
  libavfilter     9.  3.100 /  9.  3.100
  libswscale      7.  1.100 /  7.  1.100
  libswresample   4. 10.100 /  4. 10.100
  libpostproc    57.  1.100 / 57.  1.100
Input #0, image2, from 'pngs/triangle/triangle_%03d.png':
  Duration: 00:00:52.83, start: 0.000000, bitrate: N/A
  Stream #0:0: Video: png, rgba(pc), 4800x3000 [SAR 23622:23622 DAR 8:5], 12 fps, 12 tbr, 12 tl
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> h264 (libx264))
Press [q] to stop, [?] for help
[libx264 @ 0x7fa9b0807f80] using SAR=1/1
[libx264 @ 0x7fa9b0807f80] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AV
[libx264 @ 0x7fa9b0807f80] profile High 4:4:4 Predictive, level 6.0, 4:4:4, 8-bit
[libx264 @ 0x7fa9b0807f80] 264 - core 164 r3095 baee400 - H.264/MPEG-4 AVC codec - Copyleft 20
Output #0, mp4, to 'output_triangle.mp4':
  Metadata:
    encoder         : Lavf60.3.100
  Stream #0:0: Video: h264 (avc1 / 0x31637661), yuv444p(tv, progressive), 4800x3000 [SAR 1:1 D
    Metadata:
      encoder         : Lavc60.3.100 libx264
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: N/A
frame=  634 fps= 23 q=-1.0 Lsize=    1324kB time=00:00:52.58 bitrate= 206.2kbits/s speed=1.94x
video:1316kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 0.62
[libx264 @ 0x7fa9b0807f80] frame I:3     Avg QP:10.55  size:133880
[libx264 @ 0x7fa9b0807f80] frame P:162   Avg QP:12.92  size:  2541
[libx264 @ 0x7fa9b0807f80] frame B:469   Avg QP:21.63  size:  1137
[libx264 @ 0x7fa9b0807f80] consecutive B-frames:  0.6%  0.3%  5.7% 93.4%
[libx264 @ 0x7fa9b0807f80] mb I  I16..4: 51.0% 43.4%  5.6%
[libx264 @ 0x7fa9b0807f80] mb P  I16..4:  0.4%  0.2%  0.0%  P16..4:  0.2%  0.1%  0.0%  0.0%  0
[libx264 @ 0x7fa9b0807f80] mb B  I16..4:  0.0%  0.0%  0.0%  B16..8:  1.0%  0.2%  0.0%  direct:
[libx264 @ 0x7fa9b0807f80] 8x8 transform intra:39.7% inter:41.2%
[libx264 @ 0x7fa9b0807f80] coded y,u,v intra: 2.9% 0.5% 0.6% inter: 0.0% 0.0% 0.0%
[libx264 @ 0x7fa9b0807f80] i16 v,h,dc,p: 91%  9%  0%  0%
[libx264 @ 0x7fa9b0807f80] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 41%  3% 55%  0%  0%  0%  0%  0%  0%
[libx264 @ 0x7fa9b0807f80] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 44% 17% 20%  4%  3%  4%  2%  4%  2%
[libx264 @ 0x7fa9b0807f80] Weighted P-Frames: Y:0.0% UV:0.0%
[libx264 @ 0x7fa9b0807f80] ref P L0: 53.5%  2.7% 25.8% 18.0%
[libx264 @ 0x7fa9b0807f80] ref B L0: 85.3% 13.2%  1.5%
```

```
[libx264 @ 0x7fa9b0807f80] ref B L1: 96.6%  3.4%
[libx264 @ 0x7fa9b0807f80] kb/s:203.87
```

```
CompletedProcess(args='ffmpeg -framerate 12 -i pngs/triangle/triangle_%03d.png -c:v libx264 -v
```

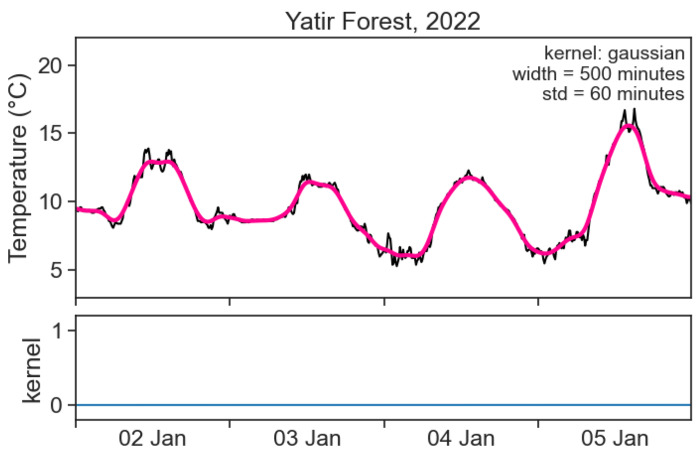
59.6 Gaussian kernel

```
%matplotlib widget
fig, ax = plt.subplots(2, 1, figsize=(8,5), sharex=True,
                       gridspec_kw={'height_ratios':[1,0.4], 'hspace':0.1})

class Lines:
    pass
lines = Lines()

ax0 = ax[0]
ax1 = ax[1]
ylim = [3, 22]
window_width_min = 500.0
window_width_int = int(window_width_min / 10) + 1
N = len(df)
t_swipe = pd.date_range(start=pd.to_datetime(start) - pd.Timedelta(minutes=window_width_min)
                        end=pd.to_datetime(end) + pd.Timedelta(minutes=60),
                        freq="10min")
t0 = t_swipe[0]
window_middle = t0 + pd.Timedelta(minutes=window_width_min/2)
# fill between blue shade, plot kernel
half_triang = np.arange(1, window_width_int/2+1, 1)
kernel_triang = np.hstack([half_triang, half_triang[-2::-1]])
kernel_triang = kernel_triang / kernel_triang.max()
df['con'] = np.convolve(df['TD'].values, kernel_triang, mode='same') / len(kernel_triang) *
df['kernel_plus'] = 0.0
df.loc[t0: t0 + pd.Timedelta(minutes=window_width_min), 'kernel_plus'] = kernel_triang

# array of minutes. multiply by 10 because data is every 10 minutes

std_in_minutes = 60
g = sp.signal.gaussian(window_width_int, std_in_minutes/10)#, sym=True)
```

```python
df.loc[t0: t0 + pd.Timedelta(minutes=window_width_min), 'kernel_plus'] = g
gaussian_threshold = np.exp(-2**2)  # two sigmas
lines.kernel_line, = ax1.plot(df['kernel_plus'], color="tab:blue")
window_above_threshold = df.loc[df['kernel_plus'] > gaussian_threshold, 'kernel_plus'].index
lines.fill_bet = ax0.fill_between([window_above_threshold[0], window_above_threshold[-1]],
                                  y1=ylim[0], y2=ylim[1], alpha=0.1, zorder=-1, col

# gaussian convolution from here: https://stackoverflow.com/questions/27205402/pandas-rollin
df.loc[:, 'con'] = np.convolve(df['TD'].values, g/g.sum(), mode='same')
ax0.plot(df.loc[start:end, 'TD'], color="black")
lines.gray_line, = ax0.plot(df.loc[window_above_threshold[0]:window_above_threshold[-1], 'TD
                     color=[0.6]*3, lw=3)
lines.pink_line, = ax0.plot(df.loc[start:window_middle, 'con'], color="xkcd:hot pink", lw=3)
lines.pink_circle, = ax0.plot([window_middle], [df.loc[window_middle, 'con']],
         marker='o', markerfacecolor="None", markeredgecolor="xkcd:dark pink", markeredgewid
         markersize=8)
ax0.text(0.99, 0.97, f"kernel: gaussian\nwidth = {window_width_min:.0f} minutes\nstd = {std_
         horizontalalignment='right', verticalalignment='top',
         fontsize=14)
ax0.set(ylim=ylim,
        xlim=[start, end],
        ylabel="Temperature (°C)",
        yticks=[5,10,15,20],
        title="Yatir Forest, 2022")
ax1.set(ylim=[-0.2, 1.2],
        xlim=[start, end],
        ylabel="kernel"
       )
gauss = df['TD'].rolling(window=window_width_int, center=True, win_type="gaussian").mean(std
center_dates_two_panels(ax0, ax1)

def update_swipe(k, lines):
    t0 = t_swipe[k]
    window_middle = t0 + pd.Timedelta(minutes=window_width_min/2)
    lines.fill_bet.remove()
    lines.pink_line.set_data(df[start:window_middle].index,
                             df.loc[start:window_middle, 'con'].values)
    lines.pink_circle.set_data([window_middle], [df.loc[window_middle, 'con']])
    lines.kernel_rect = np.ones(window_width_int)
```

```
    df['kernel_plus'] = 0.0
    df.loc[t0: t0 + pd.Timedelta(minutes=window_width_min), 'kernel_plus'] = g
    window_above_threshold = df.loc[df['kernel_plus'] > gaussian_threshold, 'kernel_plus'].i
    lines.gray_line.set_data(df.loc[window_above_threshold[0]:window_above_threshold[-1], 'T
                             df.loc[window_above_threshold[0]:window_above_threshold[-1], 'T
    lines.kernel_line.set_data(df['kernel_plus'].index, df['kernel_plus'].values)
    window_above_threshold = df.loc[df['kernel_plus'] > gaussian_threshold, 'kernel_plus'].i
    lines.fill_bet = ax0.fill_between([window_above_threshold[0], window_above_threshold[-1]
                                      y1=ylim[0], y2=ylim[1], alpha=0.1, zorder=-1, color="

progress_bar = tqdm(total=len(t_swipe), unit="iteration")
for fignum, i in enumerate(np.arange(0, len(t_swipe)-1, 1)):
    update_swipe(i, lines)
    fig.savefig(f"pngs/gaussian/gaussian_{fignum:03}.png", dpi=600)
    progress_bar.update(1)
# close the progress bar
progress_bar.close()
```

```
100%|      | 634/635 [05:47<00:00,  1.83iteration/s]
```




```
# Define the path to your PNG images
pngs_path = "pngs/gaussian"
pngs_name = "gaussian_%03d.png"
```

```python
# Define the output video file path
video_output = "output_gaussian.mp4"

fr = 12
# run command
ffmpeg_cmd = f"ffmpeg -framerate {fr} -i {pngs_path}/{pngs_name} -c:v libx264 -vf fps={fr} {
subprocess.run(ffmpeg_cmd, shell=True)
```

```
ffmpeg version 6.0 Copyright (c) 2000-2023 the FFmpeg developers
  built with Apple clang version 14.0.3 (clang-1403.0.22.14.1)
  configuration: --prefix=/usr/local/Cellar/ffmpeg/6.0 --enable-shared --enable-pthreads --enal
  libavutil      58.  2.100 / 58.  2.100
  libavcodec     60.  3.100 / 60.  3.100
  libavformat    60.  3.100 / 60.  3.100
  libavdevice    60.  1.100 / 60.  1.100
  libavfilter     9.  3.100 /  9.  3.100
  libswscale      7.  1.100 /  7.  1.100
  libswresample   4. 10.100 /  4. 10.100
  libpostproc    57.  1.100 / 57.  1.100
Input #0, image2, from 'pngs/gaussian/gaussian_%03d.png':
  Duration: 00:00:52.83, start: 0.000000, bitrate: N/A
  Stream #0:0: Video: png, rgba(pc), 4800x3000 [SAR 23622:23622 DAR 8:5], 12 fps, 12 tbr, 12 t
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> h264 (libx264))
Press [q] to stop, [?] for help
[libx264 @ 0x7ff6d8907580] using SAR=1/1
[libx264 @ 0x7ff6d8907580] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AV
[libx264 @ 0x7ff6d8907580] profile High 4:4:4 Predictive, level 6.0, 4:4:4, 8-bit
[libx264 @ 0x7ff6d8907580] 264 - core 164 r3095 baee400 - H.264/MPEG-4 AVC codec - Copyleft 20
Output #0, mp4, to 'output_gaussian.mp4':
  Metadata:
    encoder         : Lavf60.3.100
  Stream #0:0: Video: h264 (avc1 / 0x31637661), yuv444p(tv, progressive), 4800x3000 [SAR 1:1 D
    Metadata:
      encoder         : Lavc60.3.100 libx264
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: N/A
frame=  634 fps= 21 q=-1.0 Lsize=    1386kB time=00:00:52.58 bitrate= 215.9kbits/s speed=1.77x
video:1378kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 0.60
[libx264 @ 0x7ff6d8907580] frame I:3     Avg QP:10.13  size:140267
```

```
[libx264 @ 0x7ff6d8907580] frame P:161   Avg QP:14.05  size:  2700
[libx264 @ 0x7ff6d8907580] frame B:470   Avg QP:21.81  size:  1180
[libx264 @ 0x7ff6d8907580] consecutive B-frames:  0.9%  0.6%  0.0% 98.4%
[libx264 @ 0x7ff6d8907580] mb I  I16..4: 53.9% 40.2%  5.9%
[libx264 @ 0x7ff6d8907580] mb P  I16..4:  0.4%  0.3%  0.0%  P16..4:  0.2%  0.1%  0.0%  0.0%  0
[libx264 @ 0x7ff6d8907580] mb B  I16..4:  0.1%  0.0%  0.0%  B16..8:  1.0%  0.1%  0.0%  direct:
[libx264 @ 0x7ff6d8907580] 8x8 transform intra:39.0% inter:41.4%
[libx264 @ 0x7ff6d8907580] coded y,u,v intra: 3.0% 0.5% 0.6% inter: 0.0% 0.0% 0.0%
[libx264 @ 0x7ff6d8907580] i16 v,h,dc,p: 91%  9%  0%  0%
[libx264 @ 0x7ff6d8907580] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 40%  5% 55%  0%  0%  0%  0%  0%  0%
[libx264 @ 0x7ff6d8907580] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 45% 17% 20%  4%  3%  4%  2%  3%  2%
[libx264 @ 0x7ff6d8907580] Weighted P-Frames: Y:0.0% UV:0.0%
[libx264 @ 0x7ff6d8907580] ref P L0: 60.0%  3.7% 25.1% 11.2%
[libx264 @ 0x7ff6d8907580] ref B L0: 87.0% 11.7%  1.3%
[libx264 @ 0x7ff6d8907580] ref B L1: 96.7%  3.3%
[libx264 @ 0x7ff6d8907580] kb/s:213.50
```

```
CompletedProcess(args='ffmpeg -framerate 12 -i pngs/gaussian/gaussian_%03d.png -c:v libx264 -v
```

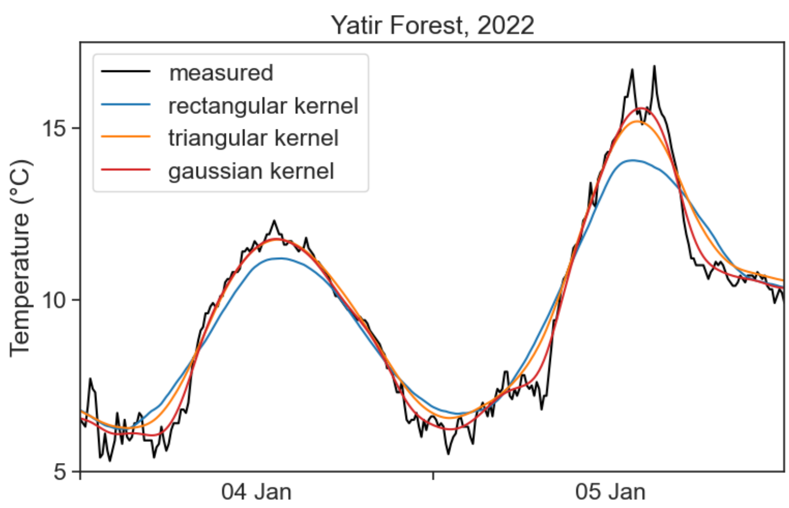
59.7 Comparison

Let's plot in one graph the smoothed temperature for each kernel shape we calculated above (rectangular, triangular, gaussian), all of which with a 500-minute-wide window.

```python
window_width_min = 500.0
window_width_int = int(window_width_min // 10 + 1)

# rectangular, 500 min
kernel_rect = np.ones(window_width_int)
rect = np.convolve(df['TD'].values, kernel_rect, mode='same') / len(kernel_rect)

# triangular
half_triang = np.arange(1, window_width_int/2+1, 1)
kernel_triang = np.hstack([half_triang, half_triang[-2::-1]])
kernel_triang = kernel_triang / kernel_triang.max()
triang = np.convolve(df['TD'].values, kernel_triang, mode='same') / len(kernel_triang) * 2

# gaussian
```

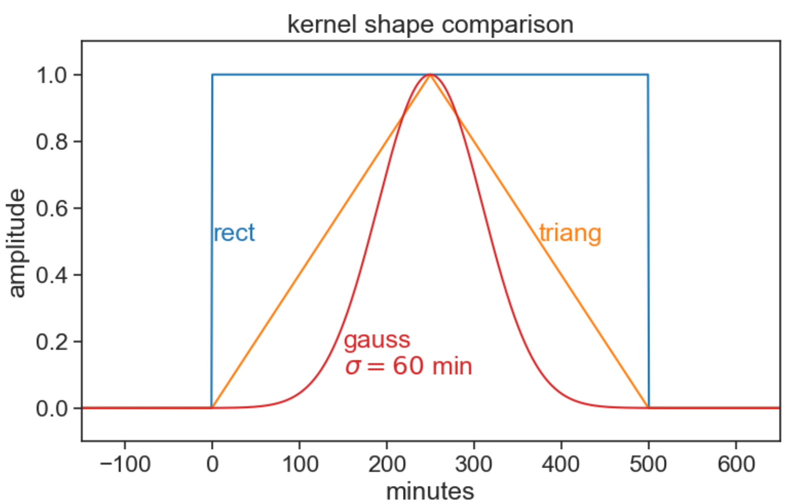
```python
gauss = df['TD'].rolling(window=window_width_int, center=True, win_type="gaussian").mean(std
```

```python
fig, ax = plt.subplots(figsize=(8,5))
ax.figure.subplots_adjust(top=0.93, bottom=0.10, left=0.1, right=0.95)

ax.plot(df.loc[start:end, 'TD'], color='black', label="measured")
ax.plot(df.index, rect, color="tab:blue", label="rectangular kernel")
ax.plot(df.index, triang, color="tab:orange", label="triangular kernel")
ax.plot(df.index, gauss, color="tab:red", label="gaussian kernel")
ax.legend()

ax.set(ylim=[5, 17.5],
       xlim=['2022-01-04 00:00:00', '2022-01-05 23:50:00'],
       ylabel="Temperature (°C)",
       title="Yatir Forest, 2022",
       yticks=[5,10,15])
center_dates(ax)
fig.savefig("kernel_comparison.png")
```




```python
fig, ax = plt.subplots(figsize=(8,5))
ax.figure.subplots_adjust(top=0.93, bottom=0.15, left=0.1, right=0.95)

N=500
```

```python
rec_window = np.zeros(800)
rec_window[150:150+N] = signal.windows.boxcar(N)
tri_window = np.zeros(800)
tri_window[150:150+N] = signal.windows.triang(N)
gau_window = np.zeros(800)
gau_window[150:150+N] = signal.windows.gaussian(N, std=60)
t = np.arange(-150, 650)
ax.plot(t, rec_window, color="tab:blue")
ax.plot(t, tri_window, color="tab:orange")
ax.plot(t, gau_window, color="tab:red")
ax.text(0, 0.5, "rect", color="tab:blue")
ax.text(373, 0.5, "triang", color="tab:orange")
ax.text(150, 0.1, "gauss\n"+r"$\sigma=60$ min", color="tab:red")
ax.set(ylim=[-0.1, 1.1],
       xlim=[-150, 650],
       ylabel="amplitude",
       xlabel="minutes",
       title="kernel shape comparison",)
fig.savefig("kernel_shapes.png")
```



savgol video

Import packages and stuff.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()  # datetime converter for a matplotlib
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import scipy as sp
import json
import requests
import os
import subprocess
from tqdm import tqdm
from scipy import signal
from scipy.signal import savgol_filter


# avoid "SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a Dat
pd.options.mode.chained_assignment = None  # default='warn'
```

Download data from the IMS using an API.

```
# read token from file
with open('../archive/IMS-token.txt', 'r') as file:
    TOKEN = file.readline()
# 28 = SHANI station
STATION_NUM = 28
```

```python
start = "2022/01/01"
end = "2022/01/07"
filename = 'shani_2022_january.json'

# check if the JSON file already exists
# if so, then load file
if os.path.exists(filename):
    with open(filename, 'r') as json_file:
        data = json.load(json_file)
else:
    # make the API request if the file doesn't exist
    url = f"https://api.ims.gov.il/v1/envista/stations/{STATION_NUM}/data/?from={start}&to={
    headers = {'Authorization': f'ApiToken {TOKEN}'}
    response = requests.get(url, headers=headers)
    data = json.loads(response.text.encode('utf8'))

    # save the JSON data to a file
    with open(filename, 'w') as json_file:
        json.dump(data, json_file)
# show data to see if it's alright
# data
```

Load and process data.

```python
df = pd.json_normalize(data['data'],record_path=['channels'], meta=['datetime'])
df['date'] = (pd.to_datetime(df['datetime'])
                .dt.tz_localize(None)  # ignores time zone information
             )
df = df.pivot(index='date', columns='name', values='value')
df
```

name date	Grad	RH	Rain	STDwd	TD	TDmax	TDmin	TG	TW	Time	WD
2022-01-01 00:00:00	0.0	77.0	0.0	10.3	11.2	11.2	11.1	10.7	-9999.0	2354.0	75.0
2022-01-01 00:10:00	0.0	77.0	0.0	11.2	11.2	11.2	11.1	10.8	-9999.0	1.0	77.0
2022-01-01 00:20:00	0.0	75.0	0.0	10.0	11.4	11.5	11.2	10.9	-9999.0	20.0	80.0
2022-01-01 00:30:00	0.0	74.0	0.0	9.6	11.5	11.5	11.4	11.0	-9999.0	22.0	76.0
2022-01-01 00:40:00	0.0	73.0	0.0	9.1	11.6	11.7	11.5	11.1	-9999.0	34.0	74.0
...	...	...	...	...	...	...	...	...	...	...	...
2022-01-06 23:10:00	0.0	36.0	0.0	16.1	11.6	12.0	11.1	6.8	-9999.0	2310.0	144.0

name	Grad	RH	Rain	STDwd	TD	TDmax	TDmin	TG	TW	Time	WD
date											
2022-01-06 23:20:00	0.0	35.0	0.0	10.1	12.1	12.3	11.9	6.3	-9999.0	2320.0	118.0
2022-01-06 23:30:00	0.0	36.0	0.0	7.1	12.4	12.6	11.9	7.3	-9999.0	2330.0	113.0
2022-01-06 23:40:00	0.0	37.0	0.0	5.6	12.6	12.7	12.5	7.8	-9999.0	2339.0	119.0
2022-01-06 23:50:00	0.0	39.0	0.0	11.5	11.9	12.6	11.5	7.1	-9999.0	2341.0	102.0

Define useful functions.

```
def concise(ax):
    """
    Let python choose the best xtick labels for you
    """
    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)
    ax.xaxis.set_major_locator(locator)
    ax.xaxis.set_major_formatter(formatter)

# dirty trick to have dates in the middle of the 24-hour period
# make minor ticks in the middle, put the labels there!
# from https://matplotlib.org/stable/gallery/ticks/centered_ticklabels.html

def center_dates(ax):
    # show day of the month + month abbreviation. see full option list here:
    # https://strftime.org
    date_form = DateFormatter("%d %b")
    # major ticks at midnight, every day
    ax.xaxis.set_major_locator(mdates.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(date_form)
    # minor ticks at noon, every day
    ax.xaxis.set_minor_locator(mdates.HourLocator(byhour=[12]))
    # erase major tick labels
    ax.xaxis.set_major_formatter(ticker.NullFormatter())
    # set minor tick labels as define above
    ax.xaxis.set_minor_formatter(date_form)
    # completely erase minor ticks, center tick labels
    for tick in ax.xaxis.get_minor_ticks():
        tick.tick1line.set_markersize(0)
        tick.tick2line.set_markersize(0)
```

```
        tick.label1.set_horizontalalignment('center')

def center_dates_two_panels(ax0, ax1):
    # show day of the month + month abbreviation. see full option list here:
    date_form = DateFormatter("%d %b")
    # major ticks at midnight, every day
    ax0.xaxis.set_major_locator(mdates.DayLocator(interval=1))
    ax1.xaxis.set_major_locator(mdates.DayLocator(interval=1))
    ax1.xaxis.set_major_formatter(date_form)
    # minor ticks at noon, every day
    ax1.xaxis.set_minor_locator(mdates.HourLocator(byhour=[12]))
    # erase major tick labels
    ax1.xaxis.set_major_formatter(ticker.NullFormatter())
    # set minor tick labels as define above
    ax1.xaxis.set_minor_formatter(date_form)
    # completely erase minor ticks, center tick labels
    for tick in ax0.xaxis.get_minor_ticks():
        tick.tick1line.set_markersize(0)
        tick.tick2line.set_markersize(0)
    for tick in ax1.xaxis.get_minor_ticks():
        tick.tick1line.set_markersize(0)
        tick.tick2line.set_markersize(0)
        tick.label1.set_horizontalalignment('center')
```

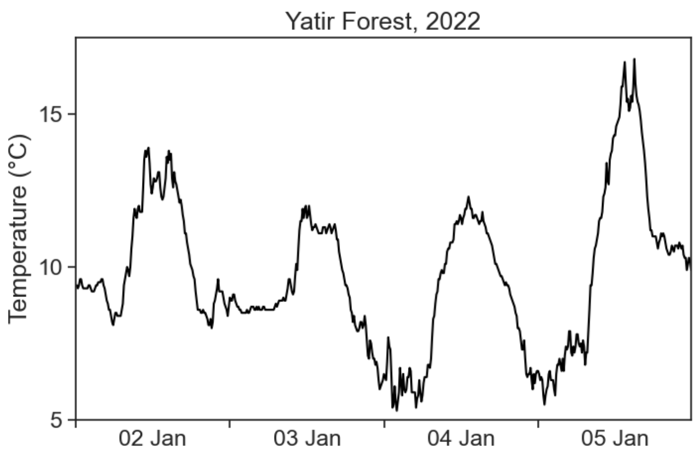
We don't need the full month, let's cut the dataframe to fewer days.

```
start = "2022-01-01 00:00:00"
end = "2022-01-06 23:50:00"
df = df.loc[start:end]
```

We now redefine a narrower window, this will be the graph's xlimits. We leave the dataframe as is, because we will need some data outside the graph's limits.

```
start = "2022-01-02 00:00:00"
end = "2022-01-05 23:50:00"
```

```
fig, ax = plt.subplots(figsize=(8,5))
ax.plot(df.loc[start:end, 'TD'], color='black')
ax.set(ylim=[5, 17.5],
       xlim=[start, end],
       ylabel="Temperature (°C)",
       title="Yatir Forest, 2022",
       yticks=[5,10,15])
center_dates(ax)
# fig.savefig("sliding_YF_temperature_2022.png")
```



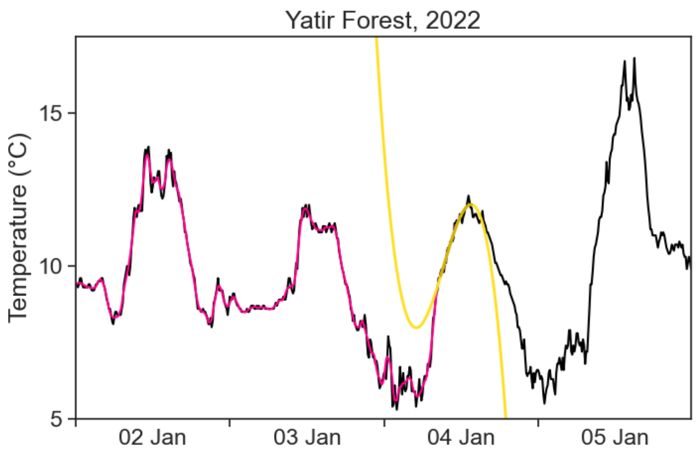
Looks good. Let's move on.

59.8 Savgol filter

```
# Function to fit and get polynomial values
def fit_polynomial(x, y, degree):
    coeffs = np.polyfit(x, y, degree)
    poly = np.poly1d(coeffs)
    return poly(x), coeffs

# Function to fit and get polynomial values
def poly_coeffs(x, y, degree):
```

```
    coeffs = np.polyfit(x, y, degree)
    return coeffs
```

```
%matplotlib widget
fig, ax = plt.subplots(figsize=(8,5))
ax.plot(df.loc[start:end, 'TD'], color='black')

sg = savgol_filter(df['TD'], 13, 2)

i = 500
ax.plot(df.index[:i], sg[:i], color='xkcd:hot pink')

window_pts = 31
p_order = 3

window_x = np.arange(i - window_pts // 2, i + window_pts // 2)
window_y = df['TD'][i - window_pts // 2:i + window_pts // 2]

# Fit and plot polynomial inside the window
fitted_y, coeffs = fit_polynomial(window_x, window_y, p_order)

whole_x = np.arange(len(df))
whole_y = df['TD'].values
poly = np.poly1d(coeffs)
whole_poly = poly(whole_x)

ax.plot(df.index, whole_poly, color='xkcd:sun yellow', lw=2)
# ax.plot(df.index[window_x], fitted_y, color='0.8', lw=3)
ax.plot(df.index[window_x], fitted_y, color='xkcd:mustard', lw=2)



ax.set(ylim=[5, 17.5],
       xlim=[start, end],
       ylabel="Temperature (°C)",
       title="Yatir Forest, 2022",
       yticks=[5,10,15])
center_dates(ax)
# fig.savefig("sliding_YF_temperature_2022.png")
```




```
p_order = 3
```

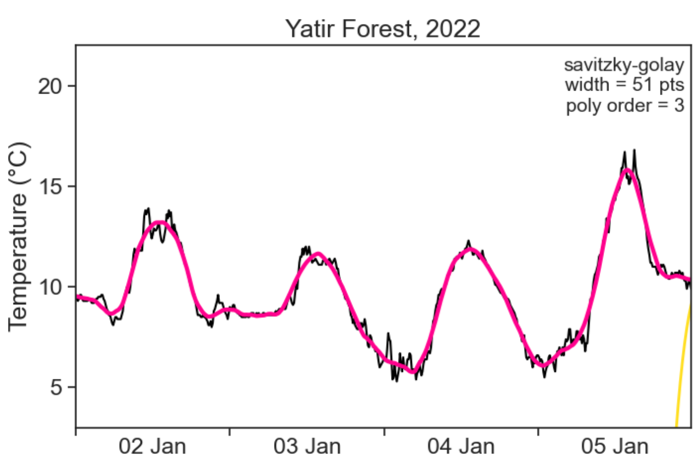
```
%matplotlib widget
fig, ax = plt.subplots(figsize=(8,5))

class Lines:
    """
    empty class, later will be populated with graph objects.
    this is useful to draw and erase lines on demand.
    """
    pass
lines = Lines()

# set graph y limits
ylim = [3, 22]
# choose here windown width in minutes
window_width_min = 500.0
window_width_min_integer = int(window_width_min)  # same but integer
window_width_int = int(window_width_min // 10 + 1)  # window width in points
```

```
N = len(df)  # df length
t_swipe = pd.date_range(start=pd.to_datetime(start) - pd.Timedelta(minutes=window_width_min)
                        end=pd.to_datetime(end) + pd.Timedelta(minutes=60),
                        freq="10min")
# starting time
t0 = t_swipe[0]
ind0 = df.index.get_loc(t0) + window_width_int//2 + 1
# show sliding window on the top panel as a light blue shade
lines.fill_bet = ax.fill_between([t0, t0 + pd.Timedelta(minutes=window_width_min)],
                                           y1=ylim[0], y2=ylim[1], alpha=0.1, zorder=-1)

sg = savgol_filter(df['TD'], window_width_int, p_order)
df.loc[:, 'sg'] = sg
# plot temperature
ax.plot(df.loc[start:end, 'TD'], color="black")

# define x,y data inside window to execute polyfit on
window_x = np.arange(ind0 - window_width_int // 2, ind0 + window_width_int // 2)
window_y = df['TD'][ind0 - window_width_int // 2:ind0 + window_width_int // 2].values
# fit and plot polynomial inside the window
fitted_y, coeffs = fit_polynomial(window_x, window_y, p_order)
# get x,y data for the whole array
whole_x = np.arange(len(df))
whole_y = df['TD'].values
poly = np.poly1d(coeffs)
whole_poly = poly(whole_x)

# calculate the middle of the sliding window
window_middle = t0 + pd.Timedelta(minutes=window_width_min/2)
# plot a pink line showing the result of the moving average
# from the beginning to the middle of the sliding window
lines.pink_line, = ax.plot(df.loc[start:window_middle, 'sg'], color="xkcd:hot pink", lw=3)

lines.poly_all, = ax.plot(df.index, whole_poly, color='xkcd:sun yellow', lw=2)
lines.poly_window, = ax.plot(df.index[window_x], fitted_y, color='xkcd:mustard', lw=2)

# emphasize the location of the middle on the window with a circle
lines.pink_circle, = ax.plot([window_middle], [df.loc[window_middle, 'sg']],
         marker='o', markerfacecolor="None", markeredgecolor="xkcd:dark pink", markeredgewid
         markersize=8)
```

```
# some explanation
ax.text(0.99, 0.97, f"savitzky-golay\nwidth = {window_width_int:.0f} pts\npoly order = {p_or
         horizontalalignment='right', verticalalignment='top',
         fontsize=14)
# axis tweaking
ax.set(ylim=ylim,
        xlim=[start, end],
        ylabel="Temperature (°C)",
        yticks=[5,10,15,20],
        title="Yatir Forest, 2022")
# adjust dates on both panels as defined before
center_dates(ax)

def update_swipe(k, lines):
    """
    updates both panels, given the index k along which the window is sliding
    """
    # left side of the sliding window
    t0 = t_swipe[k]
    # middle position
    window_middle = t0 + pd.Timedelta(minutes=window_width_min/2)
    ind0 = df.index.get_loc(window_middle)
    # erase previous blue shade on the top graph
    lines.fill_bet.remove()
    # fill again the blue shade in the updated window position
    lines.fill_bet = ax.fill_between([t0, t0 + pd.Timedelta(minutes=window_width_min)],
                                     y1=ylim[0], y2=ylim[1], alpha=0.1, zorder=-1,
    # update pink curve
    lines.pink_line.set_data(df[start:window_middle].index,
                             df.loc[start:window_middle, 'sg'].values)
    # update pink circle
    lines.pink_circle.set_data([window_middle], [df.loc[window_middle, 'sg']])
    # define x,y data inside window to execute polyfit on

    window_x = np.arange(ind0 - window_width_int // 2, ind0 + window_width_int // 2)
    window_y = df['TD'][ind0 - window_width_int // 2:ind0 + window_width_int // 2]
    # fit and plot polynomial inside the window
    fitted_y, coeffs = fit_polynomial(window_x, window_y, p_order)
    poly = np.poly1d(coeffs)
```

```
    whole_poly = poly(whole_x)
    lines.poly_all.set_data(df.index, whole_poly)
    lines.poly_window.set_data(df.index[window_x], fitted_y)

fig.savefig(f"pngs/savgol{window_width_int}/savgol_zero.png", dpi=600)

# create a tqdm progress bar
progress_bar = tqdm(total=len(t_swipe), unit="iteration")
# loop over all sliding indices, update graph and then save it
for fignum, i in enumerate(np.arange(0, len(t_swipe)-1, 1)):
    update_swipe(i, lines)
    fig.savefig(f"pngs/savgol{window_width_int}/savgol_{window_width_int}_{fignum:03}.png",
    # update the progress bar
    progress_bar.update(1)
# close the progress bar
progress_bar.close()
```

```
100%|      | 634/635 [13:07<00:01,  1.24s/iteration]
```



Combine all saved images into one mp4 video.

```
# Define the path to your PNG images
pngs_path = f"pngs/savgol51"
pngs_name = f"savgol_51_%03d.png"
```

```python
# Define the output video file path
video_output = f"output_savgol51.mp4"

# Use ffmpeg to create a video from PNG images
# desired framerate. choose 24 if you don't know what to do
fr = 12
# run command
ffmpeg_cmd = f"ffmpeg -framerate {fr} -i {pngs_path}/{pngs_name} -c:v libx264 -vf fps={fr} {
subprocess.run(ffmpeg_cmd, shell=True)
```

```
ffmpeg version 6.1.1 Copyright (c) 2000-2023 the FFmpeg developers
  built with Apple clang version 15.0.0 (clang-1500.1.0.2.5)
  configuration: --prefix=/usr/local/Cellar/ffmpeg/6.1.1_2 --enable-shared --enable-pthreads --
  libavutil      58. 29.100 / 58. 29.100
  libavcodec     60. 31.102 / 60. 31.102
  libavformat    60. 16.100 / 60. 16.100
  libavdevice    60.  3.100 / 60.  3.100
  libavfilter     9. 12.100 /  9. 12.100
  libswscale      7.  5.100 /  7.  5.100
  libswresample   4. 12.100 /  4. 12.100
  libpostproc    57.  3.100 / 57.  3.100
Input #0, image2, from 'pngs/savgol51/savgol_51_%03d.png':
  Duration: 00:00:52.83, start: 0.000000, bitrate: N/A
  Stream #0:0: Video: png, rgba(pc, gbr/unknown/unknown), 4800x3000 [SAR 23622:23622 DAR 8:5],
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> h264 (libx264))
Press [q] to stop, [?] for help
[libx264 @ 0x7f9cb7906dc0] using SAR=1/1
[libx264 @ 0x7f9cb7906dc0] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AV
[libx264 @ 0x7f9cb7906dc0] profile High 4:4:4 Predictive, level 6.0, 4:4:4, 8-bit
[libx264 @ 0x7f9cb7906dc0] 264 - core 164 r3108 31e19f9 - H.264/MPEG-4 AVC codec - Copyleft 20
Output #0, mp4, to 'output_savgol51.mp4':
  Metadata:
    encoder         : Lavf60.16.100
  Stream #0:0: Video: h264 (avc1 / 0x31637661), yuv444p(tv, progressive), 4800x3000 [SAR 1:1 D
    Metadata:
      encoder         : Lavc60.31.102 libx264
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: N/A
```

```
[out#0/mp4 @ 0x7f9cb7806000] video:3513kB audio:0kB subtitle:0kB other streams:0kB global heade
frame=  634 fps=3.2 q=-1.0 Lsize=    3521kB time=00:00:52.58 bitrate= 548.5kbits/s speed=0.27x
[libx264 @ 0x7f9cb7906dc0] frame I:3     Avg QP:13.70  size:146882
[libx264 @ 0x7f9cb7906dc0] frame P:232   Avg QP:19.02  size:  7856
[libx264 @ 0x7f9cb7906dc0] frame B:399   Avg QP:24.04  size:  3341
[libx264 @ 0x7f9cb7906dc0] consecutive B-frames: 11.4% 10.7% 10.4% 67.5%
[libx264 @ 0x7f9cb7906dc0] mb I  I16..4: 32.8% 61.0%  6.2%
[libx264 @ 0x7f9cb7906dc0] mb P  I16..4:  0.5%  0.7%  0.3%  P16..4:  0.4%  0.2%  0.1%  0.0%  0
[libx264 @ 0x7f9cb7906dc0] mb B  I16..4:  0.1%  0.0%  0.0%  B16..8:  1.9%  0.3%  0.0%  direct:
[libx264 @ 0x7f9cb7906dc0] 8x8 transform intra:51.3% inter:35.8%
[libx264 @ 0x7f9cb7906dc0] coded y,u,v intra: 7.2% 6.5% 4.4% inter: 0.1% 0.1% 0.0%
[libx264 @ 0x7f9cb7906dc0] i16 v,h,dc,p: 89% 10%  1%  0%
[libx264 @ 0x7f9cb7906dc0] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 31%  3% 65%  0%  0%  0%  0%  0%  0%
[libx264 @ 0x7f9cb7906dc0] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 54%  7% 23%  3%  1%  4%  1%  5%  1%
[libx264 @ 0x7f9cb7906dc0] Weighted P-Frames: Y:0.0% UV:0.0%
[libx264 @ 0x7f9cb7906dc0] ref P L0: 58.7%  5.7% 25.2% 10.4%
[libx264 @ 0x7f9cb7906dc0] ref B L0: 85.9% 11.8%  2.4%
[libx264 @ 0x7f9cb7906dc0] ref B L1: 96.9%  3.1%
[libx264 @ 0x7f9cb7906dc0] kb/s:544.58

CompletedProcess(args='ffmpeg -framerate 12 -i pngs/savgol51/savgol_51_%03d.png -c:v libx264 -
```

API to download data from IMS

```
# TOKEN = "f058958a-d8bd-47cc-95d7-7ecf98610e47"
# STATION_NUM = 28  # 28 = "SHANI"
# DATA = 10  # 10 = TDmax (max temperature)
# start = "2022/01/01"
# end = "2022/02/01"
# url = f"https://api.ims.gov.il/v1/envista/stations/{STATION_NUM}"
# url = f"https://api.ims.gov.il/v1/envista/stations/{STATION_NUM}/data/?from={start}&to={en
# # url = f"https://api.ims.gov.il/v1/envista/stations/{STATION_NUM}/data/{DATA}/data/11/?fr
# headers = {'Authorization': 'ApiToken f058958a-d8bd-47cc-95d7-7ecf98610e47'}
# response = requests.request("GET", url, headers=headers)
# data= json.loads(response.text.encode('utf8'))

# # Save the JSON data to a file
# with open('shani_2022_january.json', 'w') as json_file:
#     json.dump(data, json_file)

# data
```

```
# # https://ims.gov.il/he/ObservationDataAPI
# # https://ims.gov.il/sites/default/files/2021-09/API%20explanation.pdf
# # https://ims.gov.il/sites/default/files/2022-04/Python%20API%20example.pdf
# TOKEN = "f058958a-d8bd-47cc-95d7-7ecf98610e47"
# STATION_NUM = 23  # 23 = "JERUSALEM CENTRE"
# DATA = 9  # 9 = TDmax (max temperature)
# start = "2022/01/01"
# end = "2022/02/01"
# url = f"https://api.ims.gov.il/v1/envista/stations/{STATION_NUM}/data/{DATA}/?from={start}
# headers = {'Authorization': 'ApiToken f058958a-d8bd-47cc-95d7-7ecf98610e47'}
# response = requests.request("GET", url, headers=headers)
# data= json.loads(response.text.encode('utf8'))
```

```
# print(url)
```

```
# url = "https://api.ims.gov.il/v1/envista/stations/28/data/10/data/11/?from=2022/01/01&to=2
# response = requests.request("GET", url, headers=headers)
# data = json.loads(response.text.encode('utf8'))
```

```
# # RH = 8
# # TDmax = 10, max temperature
# # TDmin = 11, min temperature
# url = "https://api.ims.gov.il/v1/envista/stations/28/data/10/?from=2022/01/01&to=2022/01/3
# response = requests.request("GET", url, headers=headers)
# data = json.loads(response.text.encode('utf8'))

# df = pd.json_normalize(data['data'],record_path=['channels'], meta=['datetime'])
# df['date'] = pd.to_datetime(df['datetime']).dt.tz_localize(None)  # ignore time zone infor
# df = df.set_index('date')

# df
# data['data']
```

remove consecutive values

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import warnings
# Suppress FutureWarnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters

# %matplotlib widget
```

create data, put some defective windows here and there…

```
steps = np.random.randint(low=-2, high=2, size=500)
data = steps.cumsum()
date_range = pd.date_range(start='2023-01-01', periods=len(data), freq='1D')
df = pd.DataFrame({'series': data}, index=date_range)

# make sequence of consecutive values
df.loc['2023-06-05':'2023-07-20', 'series'] = 2
df.loc['2023-10-05':'2023-10-25', 'series'] = -150
```

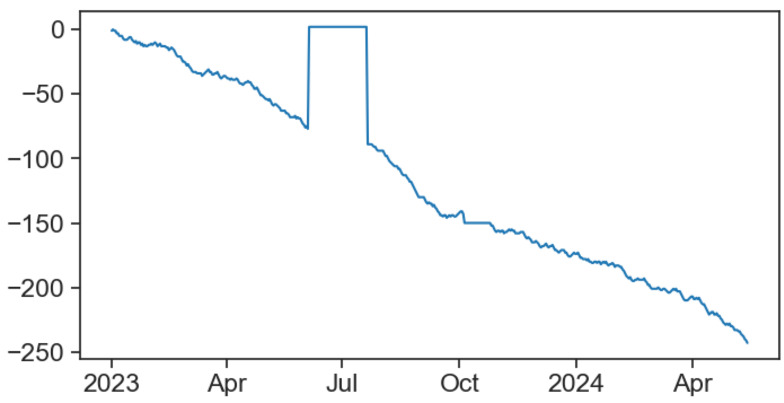
plot

```
def concise(ax):
    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)
    ax.xaxis.set_major_locator(locator)
```

```
    ax.xaxis.set_major_formatter(formatter)

fig, ax = plt.subplots(figsize=(8,4))
ax.plot(df['series'], color="tab:blue")
concise(ax)
ax.legend(frameon=False)
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an und
```

```
<matplotlib.legend.Legend at 0x7fe480a66230>
```



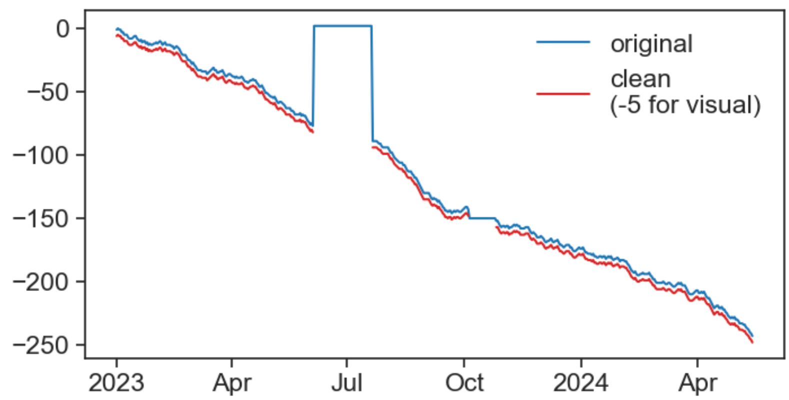
nice function, keep that for future reference

```
# function to copy paste:
def conseq_series(series, N):
    """
    part A:
    1. assume a string of 5 equal values. that's what we want to identify
    2. diff produces a string of only 4 consecutive zeros
    3. no problem, because when applying cumsum, the 4 zeros turn into a plateau of 5, that'
    so far, so good
    part B:
    1. groupby value_grp splits data into groups according to cumsum.
    2. because cumsum is monotonically increasing, necessarily all groups will be composed o
    3. what are those groups made of? of rows of column 'series'. this specific column is no
    4. count 'counts' the number of elements inside each group.
```

```
    5. the real magic here is that 'transform' assigns each row of the original group with t
    6. finally, we can ask the question: which rows belong to a string of identical values g
    zehu, you now have a mask (True-False) with the same shape as the original series.

    """
    # part A:
    sumsum_series = (
                   (series.diff() != 0)         # diff zero becomes false, otherwise true
                      .astype('int')           # true -> 1  , false -> 0
                      .cumsum()                # cumulative sum, monotonically increasing
                  )
    # part B:
    mask_outliers = (
                    series.groupby(sumsum_series)           # take original series and group
                                .transform('count')        # now count how many are in each
                                .ge(N)                    # if row count >= than user-define
                    )

    # apply mask:
    result = pd.Series(np.where(mask_outliers,
                                np.nan,  # use this if mask_outliers is True
                                series), # otherwise
                            index=series.index)
    return result
```

plot results. it works :)

```
fig, ax = plt.subplots(figsize=(8,4))
ax.plot(df['series'], color="tab:blue", label='original')
ax.plot(conseq_series(df['series'], 10)-5, c='tab:red', label='clean\n(-5 for visual)')
concise(ax)
ax.legend(frameon=False);
```



0
−50
−100
−150
−200
−250
2023
Apr
Jul
Oct
2024
Apr
original
clean
(-5 for visual)

outliers graphs

```
import matplotlib.pyplot as plt
import warnings
import pandas as pd
import numpy as np
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)  # white graphs, with large and legible letters
warnings.simplefilter(action='ignore', category=FutureWarning)
import matplotlib.gridspec as gridspec
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
from scipy.stats import median_abs_deviation
```

https://www.google.com/imgres?imgurl=https%3A%2F%2Fcxl.com%2Fwp-content%2Fuploads%2F2017%2F01%2Fchart-1.png&tbnid=RClfEFYNWm0WWM&vet=12ahUKEwjsx9KiqtKD

source: https://github.com/erykml/medium_articles/blob/master/Machine%20Learning/outlier_detection_ham

59.9 define functions

```
def random_walk_with_outliers(origin, n_steps, perc_outliers=0.0, outlier_mult=10, seed=42):
    '''
    Function for generating a random time series based on random walk.
    It adds a specified percentage of outliers by multiplying the random walk step by a scal

    Parameters
    ------------
    origin : int
        The starting point of the series
    n_steps : int
        Lenght of the series
```

```
    perc_outliers : float
        Percentage of outliers to introduce to the series [0.0-1.0]
    outlier_mult : float
        Scalar by which to multiply the RW increment to create an outlier
    seed : int
        Random seed

    Returns
    -----------
    rw : np.ndarray
        The generated random walk series with outliers
    indices : np.ndarray
        The indices of the introduced outliers
    '''
    assert (perc_outliers >= 0.0) & (perc_outliers <= 1.0)

    #set seed for reproducibility
    np.random.seed(seed)

    # possible steps
    steps = [-1, 1]

    # simulate steps
    steps = np.random.choice(a=steps, size=n_steps-1)
    rw = np.append(origin, steps).cumsum(0)

    # add outliers
    n_outliers = int(np.round(perc_outliers * n_steps, 0))
    indices = np.random.randint(0, len(rw), n_outliers)
    rw[indices] = rw[indices] + steps[indices + 1] * outlier_mult

    return rw, indices

def concise(ax):
    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)
    ax.xaxis.set_major_locator(locator)
    ax.xaxis.set_major_formatter(formatter)
```

59.10 load and process data

```
start = '2023-01-10 00:00:00'
n_steps = 1000
rw39, outlier_ind39 = random_walk_with_outliers(origin=0,
                                                n_steps=n_steps,
                                                perc_outliers=0.0031,
                                                outlier_mult=50,
                                                seed=39)
date_range = pd.date_range(start, periods=n_steps, freq='1min')
df = pd.DataFrame({'date': date_range, 'signal': rw39}).set_index('date')
start = df.index[0]
end = df.index[-1]
```

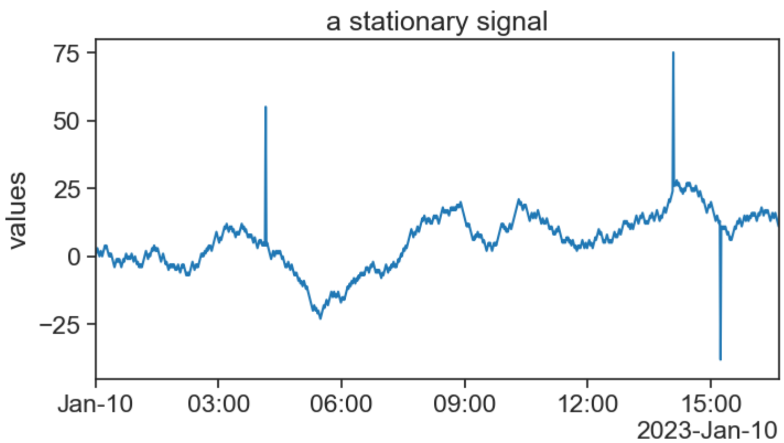
```
rw40, outlier_ind40 = random_walk_with_outliers(origin=0,
                                                n_steps=n_steps,
                                                perc_outliers=0.0031,
                                                outlier_mult=50,
                                                seed=40)
df['signal40'] = rw40

df.loc['2023-01-10 04:40:00', 'signal40'] = 43.0
```

```
# rw41, outlier_ind41 = random_walk_with_outliers(origin=0,
#                                                 n_steps=n_steps,
#                                                 perc_outliers=0.0031,
#                                                 outlier_mult=50,
#                                                 seed=41)
# df['signal41'] = rw41
```

59.11 stationary signal

```
fig, ax = plt.subplots(figsize=(8,4))
# plot signal
ax.plot(df['signal'], color="tab:blue")
# make graph look nice
```

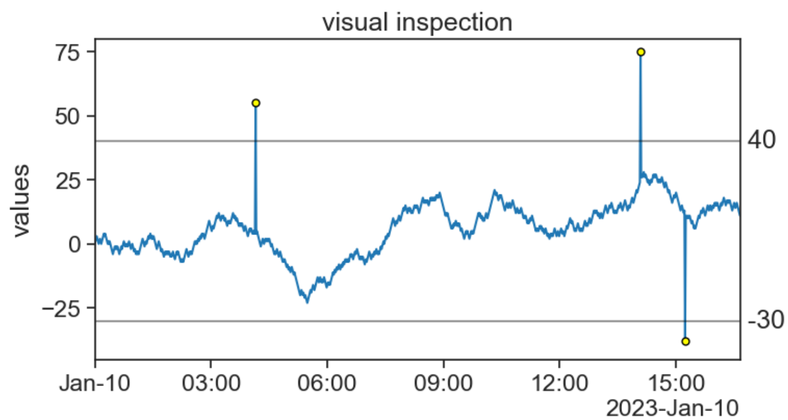
```
ax.set(ylabel='values',
       xlim=[start,end],
       title="a stationary signal",
       ylim=[-45, 80])
concise(ax)
fig.savefig("signal_39_stationary.png", bbox_inches='tight')
```



59.12 visual inspection

```
fig, ax = plt.subplots(figsize=(8,4))
# plot signal
ax.plot(df['signal'], color="tab:blue")
# plot horizontal lines
ax.plot([start, end], [40]*2, color="black", alpha=0.4)
ax.text(end, 40, " 40", va="center")
ax.plot([start, end], [-30]*2, color="black", alpha=0.4)
ax.text(end, -30, " -30", va="center")
# find and plot outliers
outliers_index = df.index[(df['signal'] > 40) | (df['signal'] < -30)]
ax.plot(df.loc[outliers_index, 'signal'], ls='None',
        marker='o', markerfacecolor='yellow', markersize=5,
        markeredgecolor="black")
# make graph look nice
ax.set(ylabel='values',
```

```
        xlim=[start,end],
        title="visual inspection",
        ylim=[-45, 80])
concise(ax)
fig.savefig("outliers_visual_inspection.png", bbox_inches='tight')
```



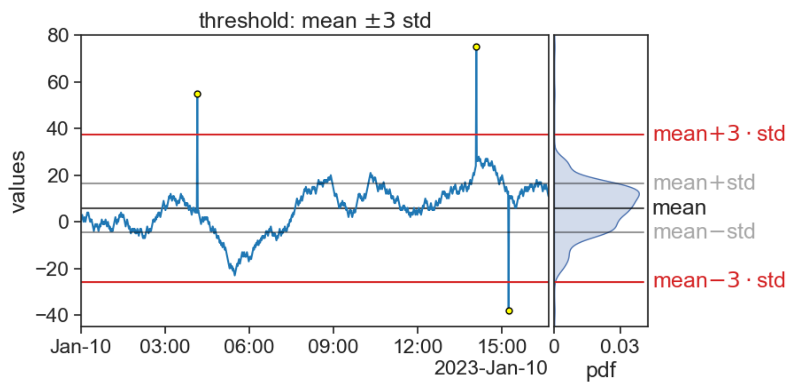
59.13 mean +- 3 std

```
fig, ax = plt.subplots(figsize=(8,4))
gs = gridspec.GridSpec(1, 2, width_ratios=[1, 0.2], height_ratios=[1])
gs.update(left=0.10, right=0.90, top=0.95, bottom=0.13,
          hspace=0.02, wspace=0.02)

ax0 = plt.subplot(gs[0, 0])
ax1 = plt.subplot(gs[0, 1])

avg = df['signal'].mean()
std = df['signal'].std()

# plot signal
ax0.plot(df['signal'], color="tab:blue")

sns.kdeplot(data=df, y='signal', shade=True, ax=ax1)
```

```python
pdf_xlim = ax1.get_xlim()
# plot horizontal lines
# mean
ax0.plot([start, end], [avg]*2, color="black", zorder=-10, alpha=0.7)
ax1.plot(pdf_xlim, [avg]*2, color="black", alpha=0.7)
ax1.text(1.1*pdf_xlim[1], avg, "mean", va="center")
# mean + std
ax0.plot([start, end], [avg+std]*2, color="black", alpha=0.4)
ax1.plot(pdf_xlim, [avg+std]*2, color="black", alpha=0.4)
ax1.text(1.1*pdf_xlim[1], avg+std, r"mean$+$std", va="center", alpha=0.4)
# mean - std
ax0.plot([start, end], [avg-std]*2, color="black", alpha=0.4)
ax1.plot(pdf_xlim, [avg-std]*2, color="black", alpha=0.4)
ax1.text(1.1*pdf_xlim[1], avg-std, r"mean$-$std", va="center", alpha=0.4)

n_sigma = 3
# mean + 3std
ax0.plot([start, end], [avg+n_sigma*std]*2, color="tab:red")
ax1.plot(pdf_xlim, [avg+n_sigma*std]*2, color="tab:red")
ax1.text(1.1*pdf_xlim[1], avg+n_sigma*std, r"mean$+3\cdot$std", va="center", color="tab:red"
# mean - 3std
ax0.plot([start, end], [avg-n_sigma*std]*2, color="tab:red")
ax1.plot(pdf_xlim, [avg-n_sigma*std]*2, color="tab:red")
ax1.text(1.1*pdf_xlim[1], avg-n_sigma*std, r"mean$-3\cdot$std", va="center", color="tab:red"

# find and plot outliers
outliers_index = df.index[(df['signal'] > avg + n_sigma*std) |
                          (df['signal'] < avg - n_sigma*std)
                         ]
ax0.plot(df.loc[outliers_index, 'signal'], ls='None',
        marker='o', markerfacecolor='yellow', markersize=5,
        markeredgecolor="black")
# make graph look nice
ax0.set(ylabel='values',
       xlim=[start,end],
       ylim=[-45, 80],
       title=r"threshold: mean $\pm3$ std",
       )
concise(ax0)
```

```
ax1.set(xlabel='pdf',
        ylabel='',
        ylim=[-45, 80],
        yticks=[],
        xticks=[0, 0.03],
        xticklabels=['0', '0.03']
        )
fig.savefig("outliers_3sigma.png", bbox_inches='tight')
```

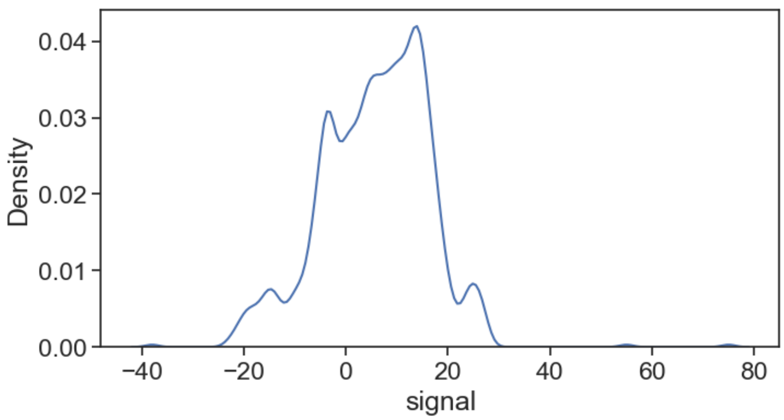
```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_73326/653833997.py:6: MatplotlibDep
  ax0 = plt.subplot(gs[0, 0])
```



59.14 IQR

```
# get kdeplot data
fig, ax = plt.subplots(figsize=(8,4))
my_kde = sns.kdeplot(df['signal'], bw_adjust=0.5)
line = my_kde.lines[0]
kde_vals, kde_pdf = line.get_data()
kde_cdf = np.cumsum(kde_pdf) / np.sum(kde_pdf)

def find_nearest(array, value):
    return (np.abs(array - value)).argmin()

# Find the boundaries where the KDE is 25% and 75% of the area
```

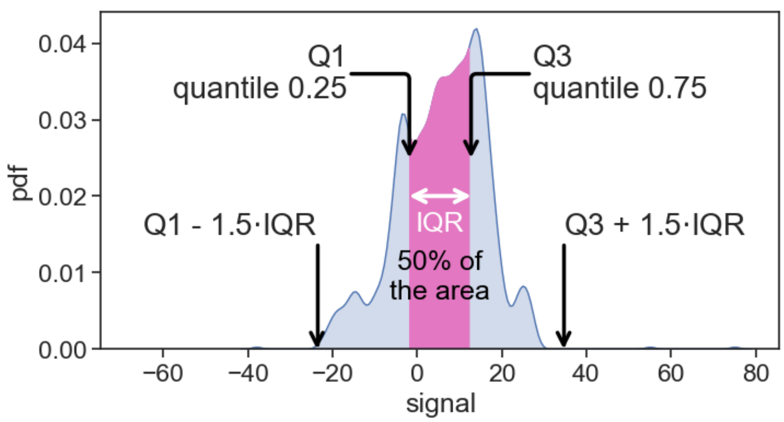
```python
Q1_index = find_nearest(kde_cdf, 0.25)
Q1_boundary = kde_vals[Q1_index]
Q3_index = find_nearest(kde_cdf, 0.75)
Q3_boundary = kde_vals[Q3_index]
IQR = Q3_boundary - Q1_boundary
```




```python
fig, ax = plt.subplots(figsize=(8,4))

sns.kdeplot(df['signal'], ax=ax, shade=True, bw_adjust=0.5)

ax.fill_between(x=kde_vals[Q1_index:Q3_index],
                y1=kde_pdf[Q1_index:Q3_index],
                color="tab:pink"
                )

h = 0.02
ax.annotate("",
            xy=(Q1_boundary, h), xycoords='data',
            xytext=(Q3_boundary, h), textcoords='data',
            size=20,
            arrowprops=dict(arrowstyle="<->",
                            connectionstyle="arc3,rad=0.0",
                            shrinkA=0, shrinkB=0,
                            linewidth=2.5
                            ),
            )
```

```
ax.annotate("",
            xy=(Q1_boundary, h), xycoords='data',
            xytext=(Q3_boundary, h), textcoords='data',
            size=20,
            arrowprops=dict(arrowstyle="<->",
                            connectionstyle="arc3,rad=0.0",
                            shrinkA=0, shrinkB=0,
                            linewidth=2.5
                            ),
            )

ax.annotate("Q1\nquantile 0.25",
            xy=(Q1_boundary, 0.025), xycoords='data',
            xytext=(Q1_boundary-IQR, 0.040), textcoords='data',
            size=20,
            ha="right",
            va="top",
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="angle,angleA=0,angleB=90,rad=5",
                            shrinkA=0, shrinkB=0,
                            linewidth=2.5,
                            color="black"
                            ),
            )

ax.annotate("Q3\nquantile 0.75",
            xy=(Q3_boundary, 0.025), xycoords='data',
            xytext=(Q3_boundary+IQR, 0.040), textcoords='data',
            size=20,
            ha="left",
            va="top",
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="angle,angleA=0,angleB=90,rad=5",
                            shrinkA=0, shrinkB=0,
                            linewidth=2.5,
                            color="black"
                            ),
            )
```

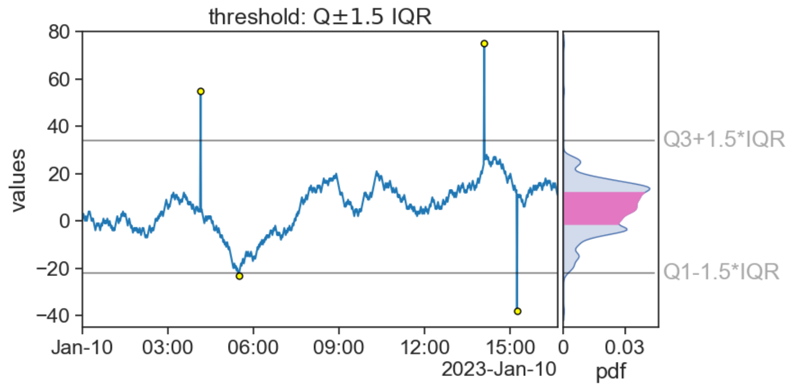
```
ax.annotate(r"Q3 + 1.5$\cdot$IQR",
            xy=(Q3_boundary+1.5*IQR, 0.00), xycoords='data',
            xytext=(Q3_boundary+1.5*IQR, 0.015), textcoords='data',
            size=20,
            ha="left",
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="angle,angleA=0,angleB=90,rad=5",
                            shrinkA=0, shrinkB=0,
                            linewidth=2.5,
                            color="black"
                            ),
            )

ax.annotate(r"Q1 - 1.5$\cdot$IQR",
            xy=(Q1_boundary-1.5*IQR, 0.00), xycoords='data',
            xytext=(Q1_boundary-1.5*IQR, 0.015), textcoords='data',
            size=20,
            ha="right",
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="angle,angleA=0,angleB=90,rad=5",
                            shrinkA=0, shrinkB=0,
                            linewidth=2.5,
                            color="black"
                            ),
            )

ax.text(Q1_boundary+IQR/2, 0.018, "IQR",
        ha="center", va="top", color="white")
ax.text(Q1_boundary+IQR/2, 0.013, r"50% of"+"\nthe area",
        ha="center", va="top", color="black")

ax.set(xlim=[Q1_boundary-5*IQR, Q3_boundary+5*IQR],
       ylabel="pdf",)
fig.savefig("IQR_pdf.png", bbox_inches='tight')
```




```
fig, ax = plt.subplots(figsize=(8,4))
gs = gridspec.GridSpec(1, 2, width_ratios=[1, 0.2], height_ratios=[1])
gs.update(left=0.10, right=0.90, top=0.95, bottom=0.13,
          hspace=0.02, wspace=0.02)

ax0 = plt.subplot(gs[0, 0])
ax1 = plt.subplot(gs[0, 1])

median = df['signal'].quantile(0.50)
Q1 = df['signal'].quantile(0.25)
Q3 = df['signal'].quantile(0.75)
IQR = Q3 - Q1

# plot signal
ax0.plot(df['signal'], color="tab:blue")

my_kde = sns.kdeplot(data=df, y='signal', shade=True, ax=ax1, bw_adjust=0.5)
pdf_xlim =ax1.get_xlim()

kde_q1_idx = find_nearest(kde_vals, Q1)
kde_q3_idx = find_nearest(kde_vals, Q3)
ax1.fill_betweenx(y=kde_vals[kde_q1_idx:kde_q3_idx],
                  x1=kde_pdf[kde_q1_idx:kde_q3_idx],
                  color="tab:pink")
# plot horizontal lines
# Q3 + 1.5 IQR
```

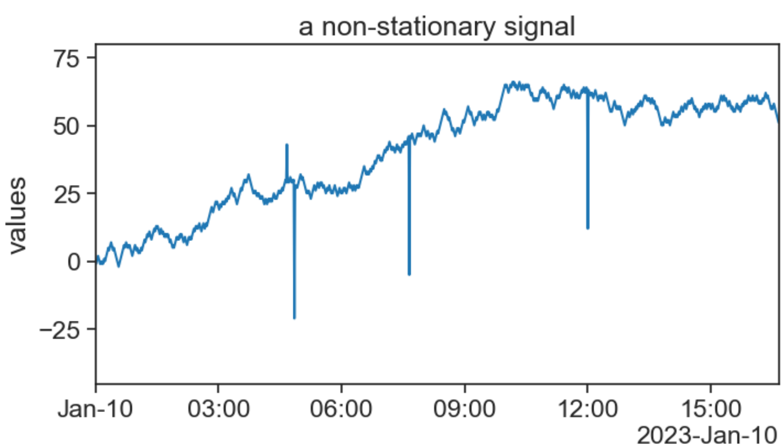
```python
ax0.plot([start, end], [Q3+1.5*IQR]*2, color="black", alpha=0.4)
ax1.plot(pdf_xlim, [Q3+1.5*IQR]*2, color="black", alpha=0.4)
ax1.text(1.1*pdf_xlim[1], Q3+1.5*IQR, "Q3+1.5*IQR", va="center", alpha=0.4)
# Q1 - 1.5 IQR
ax0.plot([start, end], [Q1-1.5*IQR]*2, color="black", alpha=0.4)
ax1.plot(pdf_xlim, [Q1-1.5*IQR]*2, color="black", alpha=0.4)
ax1.text(1.1*pdf_xlim[1], Q1-1.5*IQR, "Q1-1.5*IQR", va="center", alpha=0.4)

# find and plot outliers
outliers_index = df.index[(df['signal'] > Q3+1.5*IQR) |
                          (df['signal'] < Q1-1.5*IQR)
                         ]
ax0.plot(df.loc[outliers_index, 'signal'], ls='None',
        marker='o', markerfacecolor='yellow', markersize=5,
        markeredgecolor="black")
# make graph look nice
ax0.set(ylabel='values',
       xlim=[start,end],
       ylim=[-45, 80],
       title=r"threshold: Q$\pm1.5$ IQR",
       )
concise(ax0)
ax1.set(xlabel='pdf',
        ylabel='',
        ylim=[-45, 80],
        yticks=[],
        xticks=[0, 0.03],
        xticklabels=['0', '0.03']
        )
fig.savefig("outliers_1.5IQR.png", bbox_inches='tight')
```

```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_73326/3077230861.py:6: MatplotlibDep
  ax0 = plt.subplot(gs[0, 0])
```



59.15 non stationary signal

```
fig, ax = plt.subplots(figsize=(8,4))
# plot signal
ax.plot(df['signal40'], color="tab:blue")
# make graph look nice
ax.set(ylabel='values',
       xlim=[start,end],
       title="a non-stationary signal",
       ylim=[-45, 80])
concise(ax)
fig.savefig("signal_40_non_stationary.png", bbox_inches='tight')
```

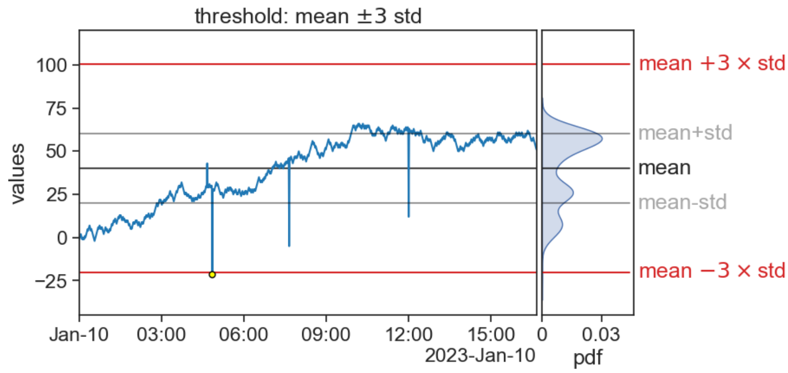


59.16 running +- 3 std

```
fig, ax = plt.subplots(figsize=(8,4))
gs = gridspec.GridSpec(1, 2, width_ratios=[1, 0.2], height_ratios=[1])
gs.update(left=0.10, right=0.90, top=0.95, bottom=0.13,
          hspace=0.02, wspace=0.02)

ax0 = plt.subplot(gs[0, 0])
ax1 = plt.subplot(gs[0, 1])

avg = df['signal40'].mean()
std = df['signal40'].std()

# plot signal
ax0.plot(df['signal40'], color="tab:blue")

sns.kdeplot(data=df, y='signal40', shade=True, ax=ax1)


# plot horizontal lines
# mean
ax0.plot([start, end], [avg]*2, color="black", zorder=-10, alpha=0.7)
ax1.plot(pdf_xlim, [avg]*2, color="black", alpha=0.7)
ax1.text(1.1*pdf_xlim[1], avg, "mean", va="center")
# mean + std
ax0.plot([start, end], [avg+std]*2, color="black", alpha=0.4)
ax1.plot(pdf_xlim, [avg+std]*2, color="black", alpha=0.4)
ax1.text(1.1*pdf_xlim[1], avg+std, "mean+std", va="center", alpha=0.4)
# mean - std
ax0.plot([start, end], [avg-std]*2, color="black", alpha=0.4)
ax1.plot(pdf_xlim, [avg-std]*2, color="black", alpha=0.4)
ax1.text(1.1*pdf_xlim[1], avg-std, "mean-std", va="center", alpha=0.4)

n_sigma = 3
# mean + 3std
ax0.plot([start, end], [avg+n_sigma*std]*2, color="tab:red")
ax1.plot(pdf_xlim, [avg+n_sigma*std]*2, color="tab:red")
ax1.text(1.1*pdf_xlim[1], avg+n_sigma*std, r"mean $+3\times$std", va="center", color="tab:re
# mean - 3std
```

```
ax0.plot([start, end], [avg-n_sigma*std]*2, color="tab:red")
ax1.plot(pdf_xlim, [avg-n_sigma*std]*2, color="tab:red")
ax1.text(1.1*pdf_xlim[1], avg-n_sigma*std, r"mean $-3\times$std", va="center", color="tab:re

# find and plot outliers
outliers_index = df.index[(df['signal40'] > avg + n_sigma*std) |
                          (df['signal40'] < avg - n_sigma*std)
                         ]
ax0.plot(df.loc[outliers_index, 'signal40'], ls='None',
        marker='o', markerfacecolor='yellow', markersize=5,
        markeredgecolor="black")
# make graph look nice
ax0.set(ylabel='values',
       xlim=[start,end],
       ylim=[-45, 120],
       title=r"threshold: mean $\pm3$ std",
       )
concise(ax0)
ax1.set(xlabel='pdf',
        ylabel='',
        ylim=[-45, 120],
        yticks=[],
        xticks=[0, 0.03],
        xticklabels=['0', '0.03']
        )
fig.savefig("outliers_3sigma_seed40.png", bbox_inches='tight')
```

```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_73326/372016966.py:6: MatplotlibDep
  ax0 = plt.subplot(gs[0, 0])
```

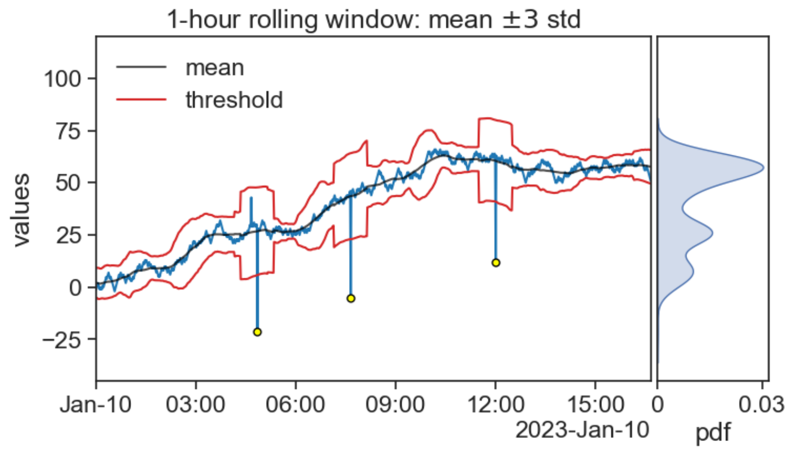



```
df['signal40_rol_mean'] = df['signal40'].rolling('60min', center=True).mean()
df['signal40_rol_std'] = df['signal40'].rolling('60min', center=True).std()
```

```
fig, ax = plt.subplots(figsize=(8,4))
gs = gridspec.GridSpec(1, 2, width_ratios=[1, 0.2], height_ratios=[1])
gs.update(left=0.10, right=0.90, top=0.95, bottom=0.13,
          hspace=0.02, wspace=0.02)

ax0 = plt.subplot(gs[0, 0])
ax1 = plt.subplot(gs[0, 1])

avg = df['signal40'].mean()
std = df['signal40'].std()

# plot signal
ax0.plot(df['signal40'], color="tab:blue")

sns.kdeplot(data=df, y='signal40', shade=True, ax=ax1)

# mean
ax0.plot(df['signal40_rol_mean'], color="black", alpha=0.7, label="mean")
# mean +- 3 std
n_sigma = 3
ax0.plot(df['signal40_rol_mean']+ n_sigma*df['signal40_rol_std'], color="tab:red")
plot_threshold, = ax0.plot(df['signal40_rol_mean']- n_sigma*df['signal40_rol_std'],
                           color="tab:red", label="threshold")
```

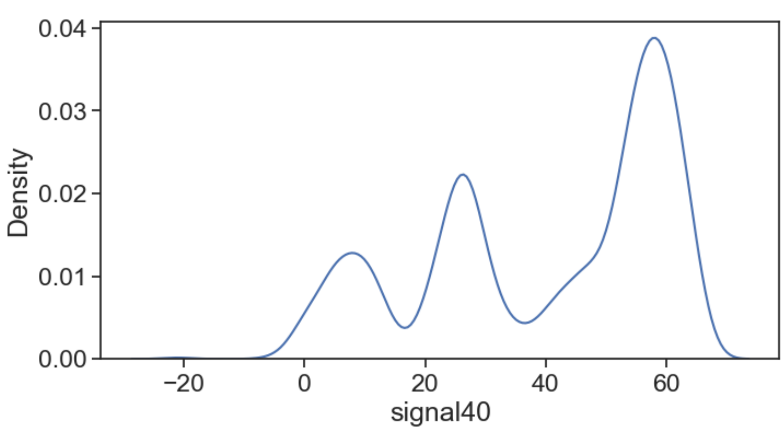
```
# find and plot outliers
outliers_index = df.index[(df['signal40'] > df['signal40_rol_mean']+ n_sigma*df['signal40_ro
                          (df['signal40'] < df['signal40_rol_mean']- n_sigma*df['signal40_ro
                         ]
ax0.plot(df.loc[outliers_index, 'signal40'], ls='None',
        marker='o', markerfacecolor='yellow', markersize=5,
        markeredgecolor="black")
# make graph look nice
ax0.set(ylabel='values',
       xlim=[start,end],
       ylim=[-45, 120],
       title=r"1-hour rolling window: mean $\pm3$ std",
       )
concise(ax0)
ax1.set(xlabel='pdf',
        ylabel='',
        ylim=[-45, 120],
        yticks=[],
        xticks=[0, 0.03],
        xticklabels=['0', '0.03']
        )
ax0.legend(frameon=False, loc="upper left")

fig.savefig("outliers_rolling_3std.png", bbox_inches='tight')
```

```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_73326/142534422.py:6: MatplotlibDep
  ax0 = plt.subplot(gs[0, 0])
```



59.17 running: Q +- IQR

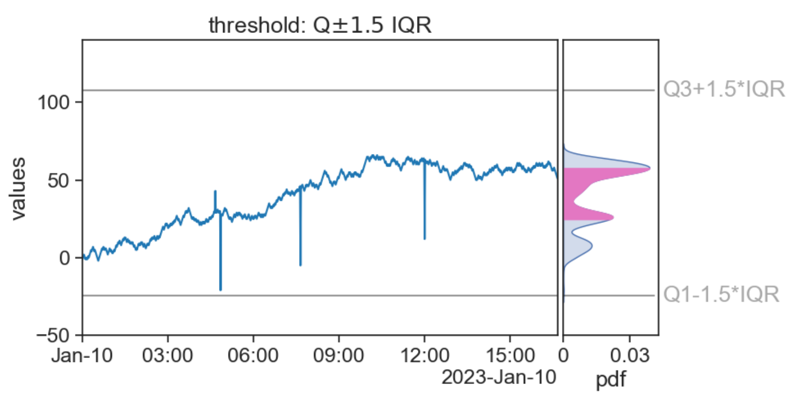
```
fig, ax = plt.subplots(figsize=(8,4))

my_kde = sns.kdeplot(df['signal40'], bw_adjust=0.5)
line = my_kde.lines[0]
kde_vals, kde_pdf = line.get_data()
kde_cdf = np.cumsum(kde_pdf) / np.sum(kde_pdf)
```



```
fig, ax = plt.subplots(figsize=(8,4))
gs = gridspec.GridSpec(1, 2, width_ratios=[1, 0.2], height_ratios=[1])
gs.update(left=0.10, right=0.90, top=0.95, bottom=0.13,
          hspace=0.02, wspace=0.02)

ax0 = plt.subplot(gs[0, 0])
ax1 = plt.subplot(gs[0, 1])

median = df['signal40'].quantile(0.50)
Q1 = df['signal40'].quantile(0.25)
Q3 = df['signal40'].quantile(0.75)
IQR = Q3 - Q1

# plot signal
ax0.plot(df['signal40'], color="tab:blue")

my_kde = sns.kdeplot(data=df, y='signal40', shade=True, ax=ax1, bw_adjust=0.5)
pdf_xlim =ax1.get_xlim()

kde_q1_idx = find_nearest(kde_vals, Q1)
kde_q3_idx = find_nearest(kde_vals, Q3)
ax1.fill_betweenx(y=kde_vals[kde_q1_idx:kde_q3_idx],
                  x1=kde_pdf[kde_q1_idx:kde_q3_idx],
                  color="tab:pink")
# plot horizontal lines
# Q3 + 1.5 IQR
ax0.plot([start, end], [Q3+1.5*IQR]*2, color="black", alpha=0.4)
ax1.plot(pdf_xlim, [Q3+1.5*IQR]*2, color="black", alpha=0.4)
ax1.text(1.1*pdf_xlim[1], Q3+1.5*IQR, "Q3+1.5*IQR", va="center", alpha=0.4)
# Q1 - 1.5 IQR
ax0.plot([start, end], [Q1-1.5*IQR]*2, color="black", alpha=0.4)
ax1.plot(pdf_xlim, [Q1-1.5*IQR]*2, color="black", alpha=0.4)
ax1.text(1.1*pdf_xlim[1], Q1-1.5*IQR, "Q1-1.5*IQR", va="center", alpha=0.4)

# find and plot outliers
outliers_index = df.index[(df['signal40'] > Q3+1.5*IQR) |
                          (df['signal40'] < Q1-1.5*IQR)
                         ]
ax0.plot(df.loc[outliers_index, 'signal40'], ls='None',
```

```
        marker='o', markerfacecolor='yellow', markersize=5,
        markeredgecolor="black")
# make graph look nice
ax0.set(ylabel='values',
       xlim=[start,end],
       ylim=[-50, 140],
       title=r"threshold: Q$\pm1.5$ IQR",
       )
concise(ax0)
ax1.set(xlabel='pdf',
        ylabel='',
        ylim=[-50, 140],
        yticks=[],
        xticks=[0, 0.03],
        xticklabels=['0', '0.03']
        )
fig.savefig("outliers_1.5IQR_seed40.png", bbox_inches='tight')
```

```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_73326/678430211.py:6: MatplotlibDep
  ax0 = plt.subplot(gs[0, 0])
```



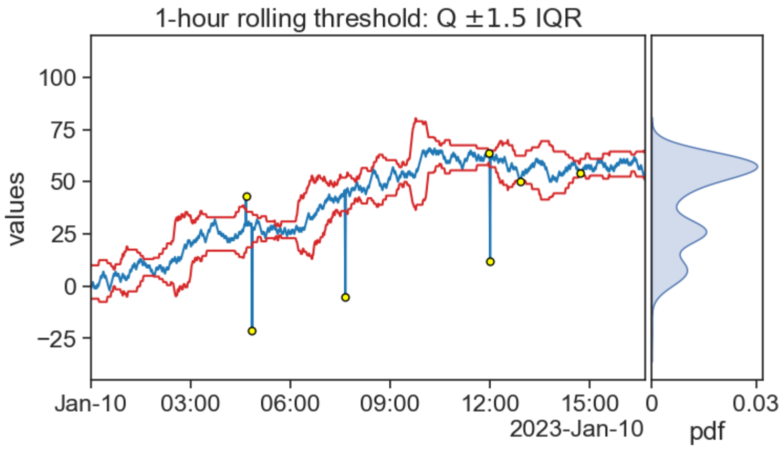

```
def Q1(window):
    return window.quantile(0.25)
def Q3(window):
    return window.quantile(0.75)
```

```
df['signal40_rol_Q1'] = df['signal40'].rolling('60min', center=True).apply(Q1)
df['signal40_rol_Q3'] = df['signal40'].rolling('60min', center=True).apply(Q3)
df['signal40_rol_IQR'] = df['signal40_rol_Q3'] - df['signal40_rol_Q1']
```

```
fig, ax = plt.subplots(figsize=(8,4))
gs = gridspec.GridSpec(1, 2, width_ratios=[1, 0.2], height_ratios=[1])
gs.update(left=0.10, right=0.90, top=0.95, bottom=0.13,
          hspace=0.02, wspace=0.02)

ax0 = plt.subplot(gs[0, 0])
ax1 = plt.subplot(gs[0, 1])

avg = df['signal40'].mean()
std = df['signal40'].std()

# plot signal
ax0.plot(df['signal40'], color="tab:blue")

sns.kdeplot(data=df, y='signal40', shade=True, ax=ax1)

# median
# ax0.plot(df['signal40_rol_median'], color="black", alpha=0.7, label="median")
# Q1 - 1.5 IQR
threshold_bottom = df['signal40_rol_Q1'] - 1.5*df['signal40_rol_IQR']
ax0.plot(threshold_bottom, color="tab:red")
# Q3 + 1.5 IQR
threshold_top = df['signal40_rol_Q3'] + 1.5*df['signal40_rol_IQR']
ax0.plot(threshold_top, color="tab:red")

# find and plot outliers
outliers_index = df.index[(df['signal40'] > threshold_top) |
                          (df['signal40'] < threshold_bottom)
                         ]
ax0.plot(df.loc[outliers_index, 'signal40'], ls='None',
        marker='o', markerfacecolor='yellow', markersize=5,
        markeredgecolor="black")
# make graph look nice
ax0.set(ylabel='values',
       xlim=[start,end],
```

```
        ylim=[-45, 120],
        title=r"1-hour rolling threshold: Q $\pm1.5$ IQR",
        )
concise(ax0)
ax1.set(xlabel='pdf',
        ylabel='',
        ylim=[-45, 120],
        yticks=[],
        xticks=[0, 0.03],
        xticklabels=['0', '0.03']
        )

ax0.legend(frameon=False, loc="upper left")


fig.savefig("outliers_rolling_IQR.png", bbox_inches='tight')
```

```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_73326/3108172465.py:6: MatplotlibDep
  ax0 = plt.subplot(gs[0, 0])
No artists with labels found to put in legend.  Note that artists whose label start with an und
```



59.18 Hampel, running MAD

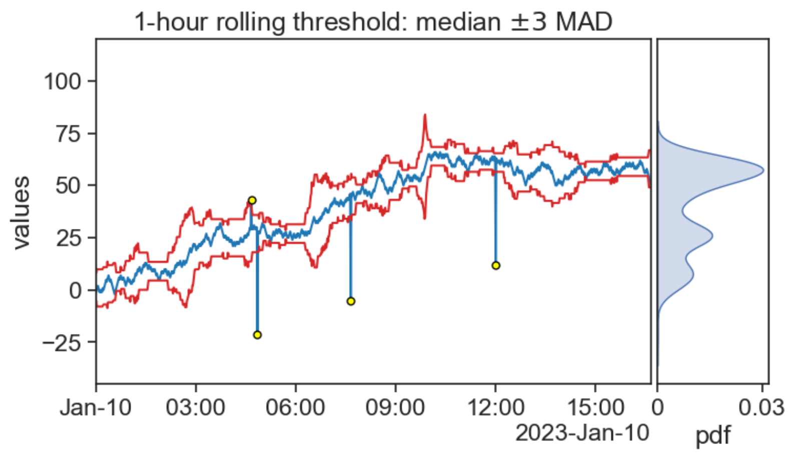
```
# def MAD(window):
#     return np.median(                              # 3. compute median
#                      np.abs(                       # 2. take absolute values
#                          window - np.median(window)  # 1. calculate residuals
#                          )
#                     )

k = 1.4826 # scale factor for Gaussian distribution
def MAD(window):
    return (window - np.median(window)).abs().median()
```

```
df['signal40_rol_mad'] = k * df['signal40'].rolling('60min', center=True).apply(MAD)
df['signal40_rol_median'] = df['signal40'].rolling('60min', center=True).median()
```

```
fig, ax = plt.subplots(figsize=(8,4))
gs = gridspec.GridSpec(1, 2, width_ratios=[1, 0.2], height_ratios=[1])
gs.update(left=0.10, right=0.90, top=0.95, bottom=0.13,
          hspace=0.02, wspace=0.02)

ax0 = plt.subplot(gs[0, 0])
ax1 = plt.subplot(gs[0, 1])

# plot signal
ax0.plot(df['signal40'], color="tab:blue")

sns.kdeplot(data=df, y='signal40', shade=True, ax=ax1)

# median
# ax0.plot(df['signal40_rol_median'], color="black", alpha=0.7, label="median")
# Q1 - 1.5 IQR
threshold_bottom = df['signal40_rol_median'] - 3 * df['signal40_rol_mad']
ax0.plot(threshold_bottom, color="tab:red")
# Q3 + 1.5 IQR
threshold_top = df['signal40_rol_median'] + 3 * df['signal40_rol_mad']
ax0.plot(threshold_top, color="tab:red")
```

```python
# find and plot outliers
outliers_index = df.index[(df['signal40'] > threshold_top) |
                          (df['signal40'] < threshold_bottom)
                         ]
ax0.plot(df.loc[outliers_index, 'signal40'], ls='None',
        marker='o', markerfacecolor='yellow', markersize=5,
        markeredgecolor="black")
# make graph look nice
ax0.set(ylabel='values',
       xlim=[start,end],
       ylim=[-45, 120],
       title=r"1-hour rolling threshold: median $\pm3$ MAD",
       )
concise(ax0)
ax1.set(xlabel='pdf',
        ylabel='',
        ylim=[-45, 120],
        yticks=[],
        xticks=[0, 0.03],
        xticklabels=['0', '0.03']
        )

ax0.legend(frameon=False, loc="upper left")


fig.savefig("outliers_rolling_MAD.png", bbox_inches='tight')
```

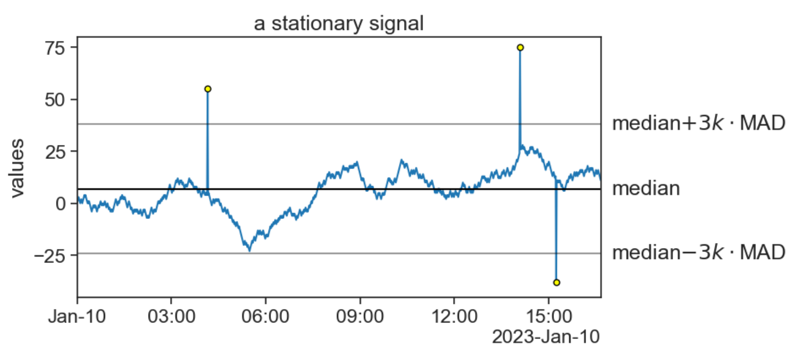
```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_73326/3446982039.py:6: MatplotlibDep
  ax0 = plt.subplot(gs[0, 0])
No artists with labels found to put in legend.  Note that artists whose label start with an und
```



59.19 stationary MAD

```
fig, ax = plt.subplots(figsize=(8,4))
# plot signal
ax.plot(df['signal'], color="tab:blue")

k = 1.4826 # scale factor for Gaussian distribution
mad = median_abs_deviation(df['signal'])
median = df['signal'].median()

xlim = ax.get_xlim()

# median +- 3*k*mad
ax.plot([start, end], [median]*2, color="black")
# ax.text(1.1, median, "median", va="center", transform=ax.transAxes,)

threshold_top = median+3*k*mad
threshold_bottom = median-3*k*mad

ax.plot([start, end], [threshold_bottom]*2, color="black", alpha=0.4)
ax.plot([start, end], [threshold_top]*2, color="black", alpha=0.4)

ax.annotate("median", xy=(1.02, median), xycoords=('axes fraction', 'data'),
            va='center')
```

```
ax.annotate(r"median$+3k\cdot$MAD", xy=(1.02, threshold_top), xycoords=('axes fraction', 'da
            va='center')
ax.annotate(r"median$-3k\cdot$MAD", xy=(1.02, threshold_bottom), xycoords=('axes fraction',
            va='center')

# find and plot outliers
outliers_index = df.index[(df['signal'] > threshold_top) |
                          (df['signal'] < threshold_bottom)
                         ]
ax.plot(df.loc[outliers_index, 'signal'], ls='None',
        marker='o', markerfacecolor='yellow', markersize=5,
        markeredgecolor="black")

# make graph look nice
ax.set(ylabel='values',
       xlim=[start,end],
       title="a stationary signal",
       ylim=[-45, 80])
concise(ax)
fig.savefig("outliers_MAD_stationary.png", bbox_inches='tight')
```



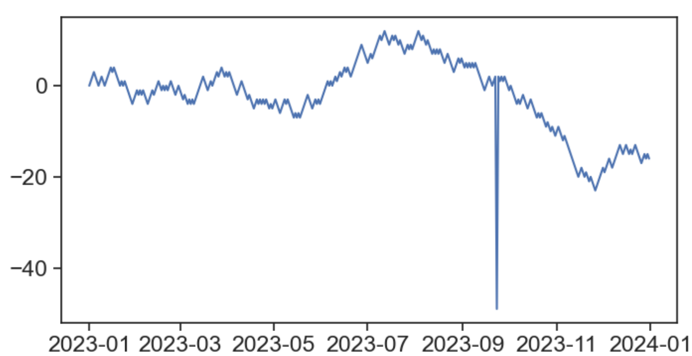

```
outliers_index
```

```
DatetimeIndex(['2023-01-10 04:09:00', '2023-01-10 14:05:00',
               '2023-01-10 15:14:00'],
              dtype='datetime64[ns]', name='date', freq=None)
```

59.20 save data as csv for later usage

```
start = '2023-01-01 00:00:00'
end = '2023-12-31 23:55:00'
# date_range = pd.date_range(start, end, freq='5min')
date_range = pd.date_range(start, end, freq='1D')
n_steps = len(date_range)
rw39, outlier_ind39 = random_walk_with_outliers(origin=0,
                                                n_steps=n_steps,
                                                perc_outliers=0.0031,
                                                outlier_mult=50,
                                                seed=39)
# date_range = pd.date_range(start, periods=n_steps, freq='1min')
df = pd.DataFrame({'date': date_range, 'A': rw39}).set_index('date')
```

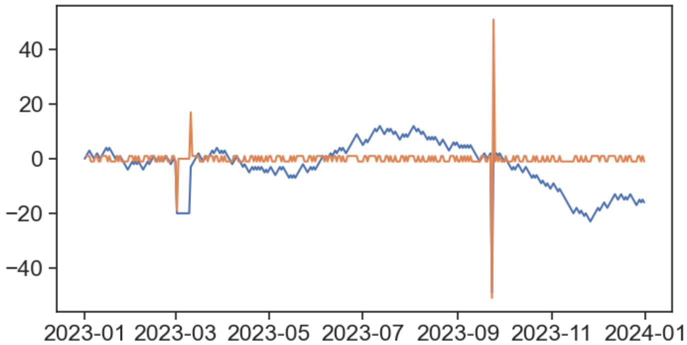
```
%matplotlib widget
fig, ax = plt.subplots(figsize=(8,4))

ax.plot(df['A'])
```




```
df.loc['2023-03-02':'2023-03-10', 'A'] = -20.0
```

```
%matplotlib widget
fig, ax = plt.subplots(figsize=(8,4))
df['Adiff'] = df['A'].diff()
```

```
ax.plot(df['A'])
ax.plot(df['Adiff'])
```




```
(df.loc['2023-03-03':'2023-03-06', 'Adiff'] == np.zeros(4)).all()
```

```
True
```

```
df.loc['2023-03-03':'2023-03-06', 'Adiff']
```

```
date
2023-03-03    0.0
2023-03-04    0.0
2023-03-05    0.0
2023-03-06    0.0
Name: Adiff, dtype: float64
```

```
np.zeros(4)
```

```
array([0., 0., 0., 0.])
```

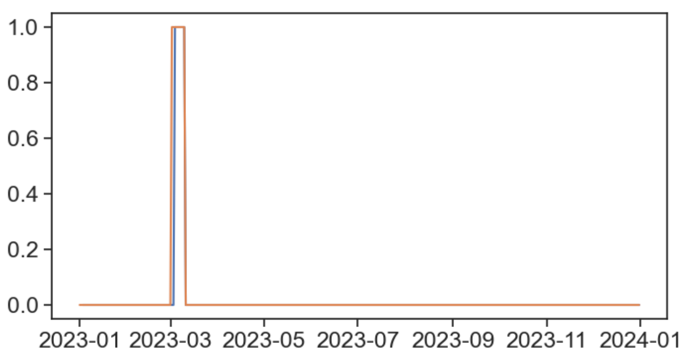
```
n_consecutive = 2

def n_zeros(series, N):
```

```
    """
    True if all series equals np.zeros(N)
    False otherwise
    """
    return (series == np.zeros(N)).all()
```

```
df['mask1'] = df['Adiff'].rolling(n_consecutive).apply(n_zeros, args=(n_consecutive,))
df['mask'] = 0.0
for i in range(len(df)):
    if df['mask1'][i] == 1.0:
        df['mask'][i-n_consecutive:i+1] = 1.0
```

```
/var/folders/kv/9cqw3y_s6c75xmgqm9n0t5d40000gn/T/ipykernel_6180/2921310703.py:5: SettingWithCop
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
  df['mask'][i-n_consecutive:i+1] = 1.0
```

```
%matplotlib widget
fig, ax = plt.subplots(figsize=(8,4))
ax.plot(df['mask1'])
ax.plot(df['mask'])
```

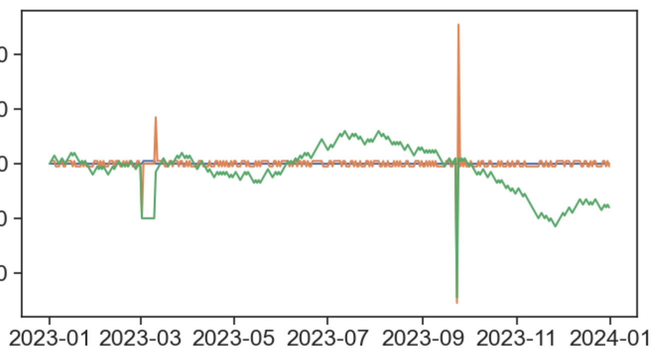



```
df['mask'] = 0.0
for j in range(len(df)-n_consecutive):
```

```
    if (df['Adiff'][j:j+n_consecutive] == np.zeros(n_consecutive)).all():
        df['mask'][j:j+n_consecutive] = 1.0
```

```
/var/folders/kv/9cqw3y_s6c75xmgqm9n0t5d40000gn/T/ipykernel_6180/2364603426.py:4: SettingWithCop
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
  df['mask'][j:j+n_consecutive] = 1.0
```

```
%matplotlib widget
fig, ax = plt.subplots(figsize=(8,4))
# ax.plot(df['mask1'])
ax.plot(df['mask'])
ax.plot(df['Adiff'])
ax.plot(df['A'])
```



59.21 generate datasets

```
def random_walk_with_outliers2(origin, steps_dist, outlier_dist, n_steps, perc_outliers=0.0,
    '''
    Function for generating a random time series based on random walk.
    It adds a specified percentage of outliers by multiplying the random walk step by a scal

    Parameters
```

```
------------
origin : int
    The starting point of the series
steps_dist : list of int or float
    step distribution
outlier_dist : list of int or float
    outlier distribution
n_steps : int
    Lenght of the series
perc_outliers : float
    Percentage of outliers to introduce to the series [0.0-1.0]
outlier_mult : float
    Scalar by which to multiply the RW increment to create an outlier
seed : int
    Random seed

Returns
-----------
rw : np.ndarray
    The generated random walk series with outliers
indices : np.ndarray
    The indices of the introduced outliers
'''
assert (perc_outliers >= 0.0) & (perc_outliers <= 1.0)

#set seed for reproducibility
np.random.seed(seed)

# possible steps
# steps = [-1, 1]

# simulate steps
steps = np.random.choice(a=steps_dist, size=n_steps-1)
rw = np.append(origin, steps).cumsum(0)

# add outliers
n_outliers = int(np.round(perc_outliers * n_steps, 0))
indices = np.random.randint(0, len(rw), n_outliers)
outlier_jumps = np.random.choice(a=outlier_dist, size=n_outliers)
# rw[indices] = rw[indices] + steps[indices + 1] * outlier_mult
```

```
    rw[indices] = rw[indices] + outlier_jumps

    return rw, indices
```

```
rw39test, _ = random_walk_with_outliers2(origin=0,
                                         steps_dist=np.random.normal(size=1000),
                                         outlier_dist=10*np.random.normal(loc=5.0, size=100),
                                         n_steps=n_steps,
                                         perc_outliers=0.0002,
                                         outlier_mult=50,
                                         seed=206)
```

```
df['B'] = rw39test
```

```
start = '2023-01-01 00:00:00'
end = '2023-12-31 23:00:00'
# date_range = pd.date_range(start, end, freq='5min')
date_range = pd.date_range(start, end, freq='1D')
n_steps = len(date_range)
```

```
# date_range = pd.date_range(start, periods=n_steps, freq='1min')
df = pd.DataFrame({'date': date_range, 'A': rw39test}).set_index('date')
```

```
df
```

	A
date	
2023-01-01	0.000000
2023-01-02	-0.032027
2023-01-03	-0.586351
2023-01-04	-1.575972
2023-01-05	-2.726800
...	...
2023-12-27	6.651301
2023-12-28	6.415175
2023-12-29	7.603140
2023-12-30	8.668182

	A
date	
2023-12-31	8.472768

```
df['unix_time'] = df.index.timestamp()
```

```
AttributeError: 'DatetimeIndex' object has no attribute 'timestamp'
```

```
df
```

	A
date	
2023-01-01	0.000000
2023-01-02	-0.032027
2023-01-03	-0.586351
2023-01-04	-1.575972
2023-01-05	-2.726800
...	...
2023-12-27	6.651301
2023-12-28	6.415175
2023-12-29	7.603140
2023-12-30	8.668182
2023-12-31	8.472768

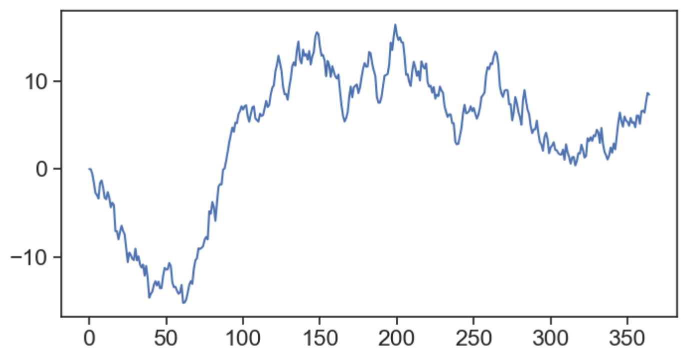
```
rw02, outlier_ind02 = random_walk_with_outliers(origin=0,
                                                n_steps=n_steps,
                                                perc_outliers=0.0001,
                                                outlier_mult=500,
                                                seed=2)
```

```
df.loc['2023-01-02 23:00:00':'2023-01-03 03:00:00', 'E'] = np.nan
```

```
df.iloc[np.random.randint(0, high=len(df), size=600)]['E'] = np.nan
```

```
/var/folders/kv/9cqw3y_s6c75xmgqm9n0t5d40000gn/T/ipykernel_9222/2255542716.py:1: SettingWithCop
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
  df.iloc[np.random.randint(0, high=len(df), size=600)]['E'] = np.nan
```

```
%matplotlib widget
fig, ax = plt.subplots(figsize=(8,4))
# ax.plot(df['B'])
ax.plot(rw39test)
```




```
df['date'] = df.index.strftime('%d%m%Y')
df['time'] = df.index.strftime('%H:%M:%S')
```

```
df['unix time (s)'] = df.index.view('int64') / 1e9
df
```

	A	date	unix time (s)
date			
2023-01-01	0.000000	2023-01-01	1.672531e+09
2023-01-02	-0.032027	2023-01-02	1.672618e+09
2023-01-03	-0.586351	2023-01-03	1.672704e+09
2023-01-04	-1.575972	2023-01-04	1.672790e+09
2023-01-05	-2.726800	2023-01-05	1.672877e+09

	A	date	unix time (s)
date			
...	...	...	...
2023-12-27	6.651301	2023-12-27	1.703635e+09
2023-12-28	6.415175	2023-12-28	1.703722e+09
2023-12-29	7.603140	2023-12-29	1.703808e+09
2023-12-30	8.668182	2023-12-30	1.703894e+09
2023-12-31	8.472768	2023-12-31	1.703981e+09

```
df.drop(columns=['date'], inplace=True)
df
```

	A	unix time (s)
date		
2023-01-01	0.000000	1.672531e+09
2023-01-02	-0.032027	1.672618e+09
2023-01-03	-0.586351	1.672704e+09
2023-01-04	-1.575972	1.672790e+09
2023-01-05	-2.726800	1.672877e+09
...	...	...
2023-12-27	6.651301	1.703635e+09
2023-12-28	6.415175	1.703722e+09
2023-12-29	7.603140	1.703808e+09
2023-12-30	8.668182	1.703894e+09
2023-12-31	8.472768	1.703981e+09

```
df.loc['2023-10-11 03:00:00', 'B'] = '-'
```

```
df.to_csv('cleaning3.csv', index=False, sep=' ')
```

```
df['A']
```

```
date
2023-01-01 00:00:00     0.000000
2023-01-01 01:00:00    -2.027536
2023-01-01 02:00:00    -2.690617
```

```
2023-01-01 03:00:00    -1.985990
2023-01-01 04:00:00    -2.290898
                          ...
2023-12-31 19:00:00   -74.514645
2023-12-31 20:00:00   -74.738058
2023-12-31 21:00:00   -75.848425
2023-12-31 22:00:00   -77.272183
2023-12-31 23:00:00   -76.557400
Name: A, Length: 8760, dtype: float64
```

```python
import datetime
import time

dt = datetime.datetime.now()
timestamp = time.mktime(dt.timetuple())
print(timestamp)
```

```
1705330442.0
```

```python
timestamp
```

```
1705330442.0
```

```python
dt
```

```
datetime.datetime(2024, 1, 15, 16, 54, 2, 420872)
```

```python
dt.timetuple()
```

```
time.struct_time(tm_year=2024, tm_mon=1, tm_mday=15, tm_hour=16, tm_min=54, tm_sec=2, tm_wday=(
```

```python
pd.to_datetime(1705330442.0, unit='s')
```

```
Timestamp('2024-01-15 14:54:02')
```

```python
(df.index.values).apply(lambda x: x.timetuple())
```

```
AttributeError: 'numpy.ndarray' object has no attribute 'apply'
```

```python
df['date'] = df.index.strftime('%Y-%m-%d')
```

```python
df['date'].apply(lambda x: x.timetuple())
```

```
AttributeError: 'str' object has no attribute 'timetuple'
```

```python
df.index.to_pydatetime()
```

```
array([datetime.datetime(2023, 1, 1, 0, 0),
       datetime.datetime(2023, 1, 2, 0, 0),
       datetime.datetime(2023, 1, 3, 0, 0),
       datetime.datetime(2023, 1, 4, 0, 0),
       datetime.datetime(2023, 1, 5, 0, 0),
       datetime.datetime(2023, 1, 6, 0, 0),
       datetime.datetime(2023, 1, 7, 0, 0),
       datetime.datetime(2023, 1, 8, 0, 0),
       datetime.datetime(2023, 1, 9, 0, 0),
       datetime.datetime(2023, 1, 10, 0, 0),
       datetime.datetime(2023, 1, 11, 0, 0),
       datetime.datetime(2023, 1, 12, 0, 0),
       datetime.datetime(2023, 1, 13, 0, 0),
       datetime.datetime(2023, 1, 14, 0, 0),
       datetime.datetime(2023, 1, 15, 0, 0),
       datetime.datetime(2023, 1, 16, 0, 0),
       datetime.datetime(2023, 1, 17, 0, 0),
       datetime.datetime(2023, 1, 18, 0, 0),
       datetime.datetime(2023, 1, 19, 0, 0),
       datetime.datetime(2023, 1, 20, 0, 0),
       datetime.datetime(2023, 1, 21, 0, 0),
       datetime.datetime(2023, 1, 22, 0, 0),
       datetime.datetime(2023, 1, 23, 0, 0),
       datetime.datetime(2023, 1, 24, 0, 0),
```

```
datetime.datetime(2023, 1, 25, 0, 0),
datetime.datetime(2023, 1, 26, 0, 0),
datetime.datetime(2023, 1, 27, 0, 0),
datetime.datetime(2023, 1, 28, 0, 0),
datetime.datetime(2023, 1, 29, 0, 0),
datetime.datetime(2023, 1, 30, 0, 0),
datetime.datetime(2023, 1, 31, 0, 0),
datetime.datetime(2023, 2, 1, 0, 0),
datetime.datetime(2023, 2, 2, 0, 0),
datetime.datetime(2023, 2, 3, 0, 0),
datetime.datetime(2023, 2, 4, 0, 0),
datetime.datetime(2023, 2, 5, 0, 0),
datetime.datetime(2023, 2, 6, 0, 0),
datetime.datetime(2023, 2, 7, 0, 0),
datetime.datetime(2023, 2, 8, 0, 0),
datetime.datetime(2023, 2, 9, 0, 0),
datetime.datetime(2023, 2, 10, 0, 0),
datetime.datetime(2023, 2, 11, 0, 0),
datetime.datetime(2023, 2, 12, 0, 0),
datetime.datetime(2023, 2, 13, 0, 0),
datetime.datetime(2023, 2, 14, 0, 0),
datetime.datetime(2023, 2, 15, 0, 0),
datetime.datetime(2023, 2, 16, 0, 0),
datetime.datetime(2023, 2, 17, 0, 0),
datetime.datetime(2023, 2, 18, 0, 0),
datetime.datetime(2023, 2, 19, 0, 0),
datetime.datetime(2023, 2, 20, 0, 0),
datetime.datetime(2023, 2, 21, 0, 0),
datetime.datetime(2023, 2, 22, 0, 0),
datetime.datetime(2023, 2, 23, 0, 0),
datetime.datetime(2023, 2, 24, 0, 0),
datetime.datetime(2023, 2, 25, 0, 0),
datetime.datetime(2023, 2, 26, 0, 0),
datetime.datetime(2023, 2, 27, 0, 0),
datetime.datetime(2023, 2, 28, 0, 0),
datetime.datetime(2023, 3, 1, 0, 0),
datetime.datetime(2023, 3, 2, 0, 0),
datetime.datetime(2023, 3, 3, 0, 0),
datetime.datetime(2023, 3, 4, 0, 0),
datetime.datetime(2023, 3, 5, 0, 0),
datetime.datetime(2023, 3, 6, 0, 0),
```

```
datetime.datetime(2023, 3, 7, 0, 0),
datetime.datetime(2023, 3, 8, 0, 0),
datetime.datetime(2023, 3, 9, 0, 0),
datetime.datetime(2023, 3, 10, 0, 0),
datetime.datetime(2023, 3, 11, 0, 0),
datetime.datetime(2023, 3, 12, 0, 0),
datetime.datetime(2023, 3, 13, 0, 0),
datetime.datetime(2023, 3, 14, 0, 0),
datetime.datetime(2023, 3, 15, 0, 0),
datetime.datetime(2023, 3, 16, 0, 0),
datetime.datetime(2023, 3, 17, 0, 0),
datetime.datetime(2023, 3, 18, 0, 0),
datetime.datetime(2023, 3, 19, 0, 0),
datetime.datetime(2023, 3, 20, 0, 0),
datetime.datetime(2023, 3, 21, 0, 0),
datetime.datetime(2023, 3, 22, 0, 0),
datetime.datetime(2023, 3, 23, 0, 0),
datetime.datetime(2023, 3, 24, 0, 0),
datetime.datetime(2023, 3, 25, 0, 0),
datetime.datetime(2023, 3, 26, 0, 0),
datetime.datetime(2023, 3, 27, 0, 0),
datetime.datetime(2023, 3, 28, 0, 0),
datetime.datetime(2023, 3, 29, 0, 0),
datetime.datetime(2023, 3, 30, 0, 0),
datetime.datetime(2023, 3, 31, 0, 0),
datetime.datetime(2023, 4, 1, 0, 0),
datetime.datetime(2023, 4, 2, 0, 0),
datetime.datetime(2023, 4, 3, 0, 0),
datetime.datetime(2023, 4, 4, 0, 0),
datetime.datetime(2023, 4, 5, 0, 0),
datetime.datetime(2023, 4, 6, 0, 0),
datetime.datetime(2023, 4, 7, 0, 0),
datetime.datetime(2023, 4, 8, 0, 0),
datetime.datetime(2023, 4, 9, 0, 0),
datetime.datetime(2023, 4, 10, 0, 0),
datetime.datetime(2023, 4, 11, 0, 0),
datetime.datetime(2023, 4, 12, 0, 0),
datetime.datetime(2023, 4, 13, 0, 0),
datetime.datetime(2023, 4, 14, 0, 0),
datetime.datetime(2023, 4, 15, 0, 0),
datetime.datetime(2023, 4, 16, 0, 0),
```

```
datetime.datetime(2023, 4, 17, 0, 0),
datetime.datetime(2023, 4, 18, 0, 0),
datetime.datetime(2023, 4, 19, 0, 0),
datetime.datetime(2023, 4, 20, 0, 0),
datetime.datetime(2023, 4, 21, 0, 0),
datetime.datetime(2023, 4, 22, 0, 0),
datetime.datetime(2023, 4, 23, 0, 0),
datetime.datetime(2023, 4, 24, 0, 0),
datetime.datetime(2023, 4, 25, 0, 0),
datetime.datetime(2023, 4, 26, 0, 0),
datetime.datetime(2023, 4, 27, 0, 0),
datetime.datetime(2023, 4, 28, 0, 0),
datetime.datetime(2023, 4, 29, 0, 0),
datetime.datetime(2023, 4, 30, 0, 0),
datetime.datetime(2023, 5, 1, 0, 0),
datetime.datetime(2023, 5, 2, 0, 0),
datetime.datetime(2023, 5, 3, 0, 0),
datetime.datetime(2023, 5, 4, 0, 0),
datetime.datetime(2023, 5, 5, 0, 0),
datetime.datetime(2023, 5, 6, 0, 0),
datetime.datetime(2023, 5, 7, 0, 0),
datetime.datetime(2023, 5, 8, 0, 0),
datetime.datetime(2023, 5, 9, 0, 0),
datetime.datetime(2023, 5, 10, 0, 0),
datetime.datetime(2023, 5, 11, 0, 0),
datetime.datetime(2023, 5, 12, 0, 0),
datetime.datetime(2023, 5, 13, 0, 0),
datetime.datetime(2023, 5, 14, 0, 0),
datetime.datetime(2023, 5, 15, 0, 0),
datetime.datetime(2023, 5, 16, 0, 0),
datetime.datetime(2023, 5, 17, 0, 0),
datetime.datetime(2023, 5, 18, 0, 0),
datetime.datetime(2023, 5, 19, 0, 0),
datetime.datetime(2023, 5, 20, 0, 0),
datetime.datetime(2023, 5, 21, 0, 0),
datetime.datetime(2023, 5, 22, 0, 0),
datetime.datetime(2023, 5, 23, 0, 0),
datetime.datetime(2023, 5, 24, 0, 0),
datetime.datetime(2023, 5, 25, 0, 0),
datetime.datetime(2023, 5, 26, 0, 0),
datetime.datetime(2023, 5, 27, 0, 0),
```

```
datetime.datetime(2023, 5, 28, 0, 0),
datetime.datetime(2023, 5, 29, 0, 0),
datetime.datetime(2023, 5, 30, 0, 0),
datetime.datetime(2023, 5, 31, 0, 0),
datetime.datetime(2023, 6, 1, 0, 0),
datetime.datetime(2023, 6, 2, 0, 0),
datetime.datetime(2023, 6, 3, 0, 0),
datetime.datetime(2023, 6, 4, 0, 0),
datetime.datetime(2023, 6, 5, 0, 0),
datetime.datetime(2023, 6, 6, 0, 0),
datetime.datetime(2023, 6, 7, 0, 0),
datetime.datetime(2023, 6, 8, 0, 0),
datetime.datetime(2023, 6, 9, 0, 0),
datetime.datetime(2023, 6, 10, 0, 0),
datetime.datetime(2023, 6, 11, 0, 0),
datetime.datetime(2023, 6, 12, 0, 0),
datetime.datetime(2023, 6, 13, 0, 0),
datetime.datetime(2023, 6, 14, 0, 0),
datetime.datetime(2023, 6, 15, 0, 0),
datetime.datetime(2023, 6, 16, 0, 0),
datetime.datetime(2023, 6, 17, 0, 0),
datetime.datetime(2023, 6, 18, 0, 0),
datetime.datetime(2023, 6, 19, 0, 0),
datetime.datetime(2023, 6, 20, 0, 0),
datetime.datetime(2023, 6, 21, 0, 0),
datetime.datetime(2023, 6, 22, 0, 0),
datetime.datetime(2023, 6, 23, 0, 0),
datetime.datetime(2023, 6, 24, 0, 0),
datetime.datetime(2023, 6, 25, 0, 0),
datetime.datetime(2023, 6, 26, 0, 0),
datetime.datetime(2023, 6, 27, 0, 0),
datetime.datetime(2023, 6, 28, 0, 0),
datetime.datetime(2023, 6, 29, 0, 0),
datetime.datetime(2023, 6, 30, 0, 0),
datetime.datetime(2023, 7, 1, 0, 0),
datetime.datetime(2023, 7, 2, 0, 0),
datetime.datetime(2023, 7, 3, 0, 0),
datetime.datetime(2023, 7, 4, 0, 0),
datetime.datetime(2023, 7, 5, 0, 0),
datetime.datetime(2023, 7, 6, 0, 0),
datetime.datetime(2023, 7, 7, 0, 0),
```

```
datetime.datetime(2023, 7, 8, 0, 0),
datetime.datetime(2023, 7, 9, 0, 0),
datetime.datetime(2023, 7, 10, 0, 0),
datetime.datetime(2023, 7, 11, 0, 0),
datetime.datetime(2023, 7, 12, 0, 0),
datetime.datetime(2023, 7, 13, 0, 0),
datetime.datetime(2023, 7, 14, 0, 0),
datetime.datetime(2023, 7, 15, 0, 0),
datetime.datetime(2023, 7, 16, 0, 0),
datetime.datetime(2023, 7, 17, 0, 0),
datetime.datetime(2023, 7, 18, 0, 0),
datetime.datetime(2023, 7, 19, 0, 0),
datetime.datetime(2023, 7, 20, 0, 0),
datetime.datetime(2023, 7, 21, 0, 0),
datetime.datetime(2023, 7, 22, 0, 0),
datetime.datetime(2023, 7, 23, 0, 0),
datetime.datetime(2023, 7, 24, 0, 0),
datetime.datetime(2023, 7, 25, 0, 0),
datetime.datetime(2023, 7, 26, 0, 0),
datetime.datetime(2023, 7, 27, 0, 0),
datetime.datetime(2023, 7, 28, 0, 0),
datetime.datetime(2023, 7, 29, 0, 0),
datetime.datetime(2023, 7, 30, 0, 0),
datetime.datetime(2023, 7, 31, 0, 0),
datetime.datetime(2023, 8, 1, 0, 0),
datetime.datetime(2023, 8, 2, 0, 0),
datetime.datetime(2023, 8, 3, 0, 0),
datetime.datetime(2023, 8, 4, 0, 0),
datetime.datetime(2023, 8, 5, 0, 0),
datetime.datetime(2023, 8, 6, 0, 0),
datetime.datetime(2023, 8, 7, 0, 0),
datetime.datetime(2023, 8, 8, 0, 0),
datetime.datetime(2023, 8, 9, 0, 0),
datetime.datetime(2023, 8, 10, 0, 0),
datetime.datetime(2023, 8, 11, 0, 0),
datetime.datetime(2023, 8, 12, 0, 0),
datetime.datetime(2023, 8, 13, 0, 0),
datetime.datetime(2023, 8, 14, 0, 0),
datetime.datetime(2023, 8, 15, 0, 0),
datetime.datetime(2023, 8, 16, 0, 0),
datetime.datetime(2023, 8, 17, 0, 0),
```

```
datetime.datetime(2023, 8, 18, 0, 0),
datetime.datetime(2023, 8, 19, 0, 0),
datetime.datetime(2023, 8, 20, 0, 0),
datetime.datetime(2023, 8, 21, 0, 0),
datetime.datetime(2023, 8, 22, 0, 0),
datetime.datetime(2023, 8, 23, 0, 0),
datetime.datetime(2023, 8, 24, 0, 0),
datetime.datetime(2023, 8, 25, 0, 0),
datetime.datetime(2023, 8, 26, 0, 0),
datetime.datetime(2023, 8, 27, 0, 0),
datetime.datetime(2023, 8, 28, 0, 0),
datetime.datetime(2023, 8, 29, 0, 0),
datetime.datetime(2023, 8, 30, 0, 0),
datetime.datetime(2023, 8, 31, 0, 0),
datetime.datetime(2023, 9, 1, 0, 0),
datetime.datetime(2023, 9, 2, 0, 0),
datetime.datetime(2023, 9, 3, 0, 0),
datetime.datetime(2023, 9, 4, 0, 0),
datetime.datetime(2023, 9, 5, 0, 0),
datetime.datetime(2023, 9, 6, 0, 0),
datetime.datetime(2023, 9, 7, 0, 0),
datetime.datetime(2023, 9, 8, 0, 0),
datetime.datetime(2023, 9, 9, 0, 0),
datetime.datetime(2023, 9, 10, 0, 0),
datetime.datetime(2023, 9, 11, 0, 0),
datetime.datetime(2023, 9, 12, 0, 0),
datetime.datetime(2023, 9, 13, 0, 0),
datetime.datetime(2023, 9, 14, 0, 0),
datetime.datetime(2023, 9, 15, 0, 0),
datetime.datetime(2023, 9, 16, 0, 0),
datetime.datetime(2023, 9, 17, 0, 0),
datetime.datetime(2023, 9, 18, 0, 0),
datetime.datetime(2023, 9, 19, 0, 0),
datetime.datetime(2023, 9, 20, 0, 0),
datetime.datetime(2023, 9, 21, 0, 0),
datetime.datetime(2023, 9, 22, 0, 0),
datetime.datetime(2023, 9, 23, 0, 0),
datetime.datetime(2023, 9, 24, 0, 0),
datetime.datetime(2023, 9, 25, 0, 0),
datetime.datetime(2023, 9, 26, 0, 0),
datetime.datetime(2023, 9, 27, 0, 0),
```

```
datetime.datetime(2023, 9, 28, 0, 0),
datetime.datetime(2023, 9, 29, 0, 0),
datetime.datetime(2023, 9, 30, 0, 0),
datetime.datetime(2023, 10, 1, 0, 0),
datetime.datetime(2023, 10, 2, 0, 0),
datetime.datetime(2023, 10, 3, 0, 0),
datetime.datetime(2023, 10, 4, 0, 0),
datetime.datetime(2023, 10, 5, 0, 0),
datetime.datetime(2023, 10, 6, 0, 0),
datetime.datetime(2023, 10, 7, 0, 0),
datetime.datetime(2023, 10, 8, 0, 0),
datetime.datetime(2023, 10, 9, 0, 0),
datetime.datetime(2023, 10, 10, 0, 0),
datetime.datetime(2023, 10, 11, 0, 0),
datetime.datetime(2023, 10, 12, 0, 0),
datetime.datetime(2023, 10, 13, 0, 0),
datetime.datetime(2023, 10, 14, 0, 0),
datetime.datetime(2023, 10, 15, 0, 0),
datetime.datetime(2023, 10, 16, 0, 0),
datetime.datetime(2023, 10, 17, 0, 0),
datetime.datetime(2023, 10, 18, 0, 0),
datetime.datetime(2023, 10, 19, 0, 0),
datetime.datetime(2023, 10, 20, 0, 0),
datetime.datetime(2023, 10, 21, 0, 0),
datetime.datetime(2023, 10, 22, 0, 0),
datetime.datetime(2023, 10, 23, 0, 0),
datetime.datetime(2023, 10, 24, 0, 0),
datetime.datetime(2023, 10, 25, 0, 0),
datetime.datetime(2023, 10, 26, 0, 0),
datetime.datetime(2023, 10, 27, 0, 0),
datetime.datetime(2023, 10, 28, 0, 0),
datetime.datetime(2023, 10, 29, 0, 0),
datetime.datetime(2023, 10, 30, 0, 0),
datetime.datetime(2023, 10, 31, 0, 0),
datetime.datetime(2023, 11, 1, 0, 0),
datetime.datetime(2023, 11, 2, 0, 0),
datetime.datetime(2023, 11, 3, 0, 0),
datetime.datetime(2023, 11, 4, 0, 0),
datetime.datetime(2023, 11, 5, 0, 0),
datetime.datetime(2023, 11, 6, 0, 0),
datetime.datetime(2023, 11, 7, 0, 0),
```

```
datetime.datetime(2023, 11, 8, 0, 0),
datetime.datetime(2023, 11, 9, 0, 0),
datetime.datetime(2023, 11, 10, 0, 0),
datetime.datetime(2023, 11, 11, 0, 0),
datetime.datetime(2023, 11, 12, 0, 0),
datetime.datetime(2023, 11, 13, 0, 0),
datetime.datetime(2023, 11, 14, 0, 0),
datetime.datetime(2023, 11, 15, 0, 0),
datetime.datetime(2023, 11, 16, 0, 0),
datetime.datetime(2023, 11, 17, 0, 0),
datetime.datetime(2023, 11, 18, 0, 0),
datetime.datetime(2023, 11, 19, 0, 0),
datetime.datetime(2023, 11, 20, 0, 0),
datetime.datetime(2023, 11, 21, 0, 0),
datetime.datetime(2023, 11, 22, 0, 0),
datetime.datetime(2023, 11, 23, 0, 0),
datetime.datetime(2023, 11, 24, 0, 0),
datetime.datetime(2023, 11, 25, 0, 0),
datetime.datetime(2023, 11, 26, 0, 0),
datetime.datetime(2023, 11, 27, 0, 0),
datetime.datetime(2023, 11, 28, 0, 0),
datetime.datetime(2023, 11, 29, 0, 0),
datetime.datetime(2023, 11, 30, 0, 0),
datetime.datetime(2023, 12, 1, 0, 0),
datetime.datetime(2023, 12, 2, 0, 0),
datetime.datetime(2023, 12, 3, 0, 0),
datetime.datetime(2023, 12, 4, 0, 0),
datetime.datetime(2023, 12, 5, 0, 0),
datetime.datetime(2023, 12, 6, 0, 0),
datetime.datetime(2023, 12, 7, 0, 0),
datetime.datetime(2023, 12, 8, 0, 0),
datetime.datetime(2023, 12, 9, 0, 0),
datetime.datetime(2023, 12, 10, 0, 0),
datetime.datetime(2023, 12, 11, 0, 0),
datetime.datetime(2023, 12, 12, 0, 0),
datetime.datetime(2023, 12, 13, 0, 0),
datetime.datetime(2023, 12, 14, 0, 0),
datetime.datetime(2023, 12, 15, 0, 0),
datetime.datetime(2023, 12, 16, 0, 0),
datetime.datetime(2023, 12, 17, 0, 0),
datetime.datetime(2023, 12, 18, 0, 0),
```

```
       datetime.datetime(2023, 12, 19, 0, 0),
       datetime.datetime(2023, 12, 20, 0, 0),
       datetime.datetime(2023, 12, 21, 0, 0),
       datetime.datetime(2023, 12, 22, 0, 0),
       datetime.datetime(2023, 12, 23, 0, 0),
       datetime.datetime(2023, 12, 24, 0, 0),
       datetime.datetime(2023, 12, 25, 0, 0),
       datetime.datetime(2023, 12, 26, 0, 0),
       datetime.datetime(2023, 12, 27, 0, 0),
       datetime.datetime(2023, 12, 28, 0, 0),
       datetime.datetime(2023, 12, 29, 0, 0),
       datetime.datetime(2023, 12, 30, 0, 0),
       datetime.datetime(2023, 12, 31, 0, 0)], dtype=object)
```

```
unix = df.index.view('int64') / 1e9
```

```
unix[0]
```

```
1672531200.0
```

```
pd.to_datetime(unix, unit='s')
```

```
DatetimeIndex(['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04',
               '2023-01-05', '2023-01-06', '2023-01-07', '2023-01-08',
               '2023-01-09', '2023-01-10',
               ...
               '2023-12-22', '2023-12-23', '2023-12-24', '2023-12-25',
               '2023-12-26', '2023-12-27', '2023-12-28', '2023-12-29',
               '2023-12-30', '2023-12-31'],
              dtype='datetime64[ns]', length=365, freq=None)
```

make your own website

1. Install Quarto on your machine.
2. Install VS Code.
3. Open a GitHub account. Choose a good username for you, this will be also the name of your website.
4. Get acquainted with basic markdown syntax. You will be writing in markdown, this is a good investment of your time.

The more you know about `git` and basic `command line` instructions, the easier all this will be. You don't need to be a jedi master in computers to make this work, this is not hard, I promisse. Quarto has a great webpage with detailed explanations. Also, ChatGPT can help you if you're lucky.

random tips

extensions

Use icons on your website with iconify and fontawesome. You can actually use a wide variety of extensions, check them out.

quarto.yml

You can configure whatever you need in the `_quarto.yml` file. I'll paste here my html formatting for reference.

```
format:
html:
  theme:
  # see all available themes https://bootswatch.com
  - flatly            # chose whatever theme you find suitable
```

```yaml
    - custom.scss       # customize how your website looks
  fontsize: 1.2em     # self explanatory
  # choose a nice highlight style for the code
  highlight-style: breezedark # monokai # breezedark # espresso
  include-in-header:
    - includes.html          # you might need to use css configurations in all your pages
  code-line-numbers: true    # turn on line numbering
  code-tools:
    # if you defined repo-url, this will link your website to it.
    # repo-url: https://github.com/github_username/repository_name/
    source: repo  # https://quarto.org/docs/output-formats/html-code.html#code-tools
  callout-icon: false
  fig-align: center         # center images as default
  # the default MathJax rendering option yields ugly results, use katex
  html-math-method: katex
```

configure your notebook with a suitable header

You could start your jupyter notebook with a markdown cell with this header

```
# this jupyter notebook title
```

but in case you need a lot of control over the details, consider using:

```yaml
---
title: "this jupyter notebook title"
execute:
  # echo: false  # chose this if you don't want to see the code at all, just the output
  freeze: auto  # re-render only when source changes, VERY useful
format:
  html:
    code-fold: true                  # hide code blocks, show them upon click
    code-summary: "Show the code"    # rename button to show code block
---
```

obvious statement

This very website is a "Quarto website" project hosted on github. Click on " Code" on the top of the page to go to the github repository. Then copy whatever you want, it's all open for everyone to see.