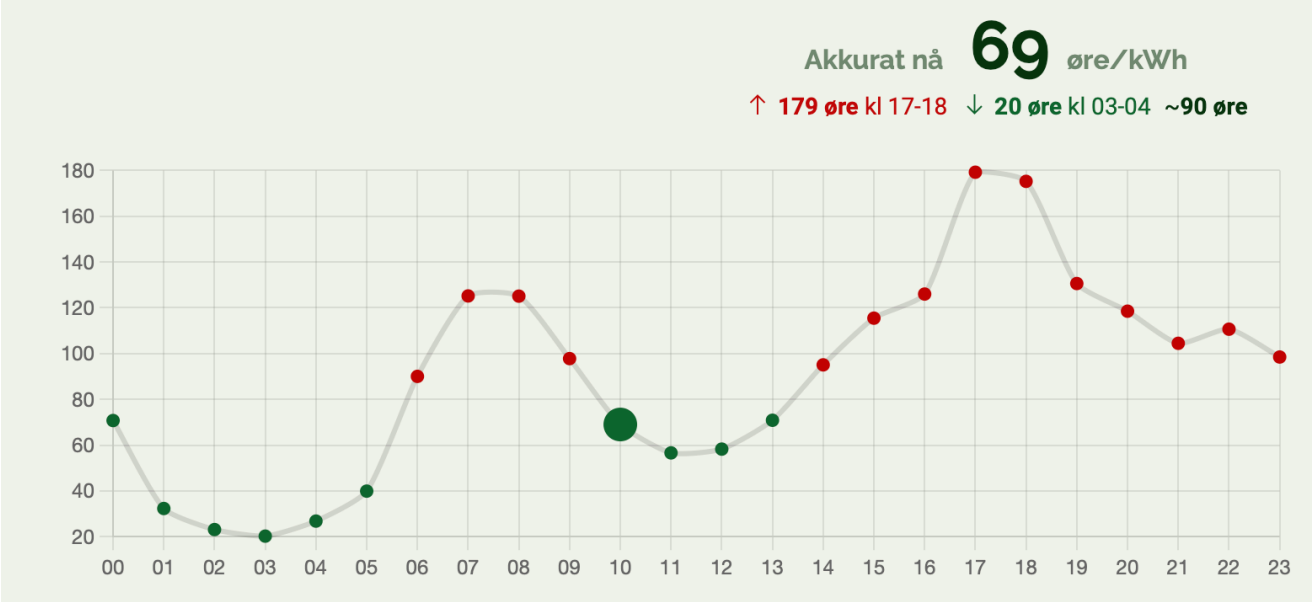
Assignment 5: Strømpris

Contents

updates

- `test_plot_prices_json` included tests with dates in 2022 when the assignment suggested that you could assert that requests wouldn't be before October, 2023. You can safely update the tests to 2023 if you had checked out the assignment already. If the tests pass with the dates as they are, you don't have to do anything. The code skeleton has been updated to use 2023. We will not grade your code with any requests before October, 2023.

Skip to main content



In this assignment we will delve into the new national past-time: Obsessively checking electricity prices. To be more precise, you will be building a web-based visualization of energy prices in Norway, using the Hva Koster Strømmen API: https://www.hvakosterstrommen.no/strompris-api. Your implementation will be based on on the following libraries:

- altair
- fastapi
- pandas
- requests

General information

Total points (IN3110): 30 (+ 16 bonus points)

Total points (IN4110): 40 (+ 16 bonus points)

As usual, you can download the code skeleton, which you should extract to the `assignment5` folder in your in3110 repo.

The packages required for the assignment are in `requirements.txt`, which you can install with `pip install -e .`.

Your `assignment5` directory must contain a `README.md` file with information on the required dependencies and packages (versions the code ran with) as well as how to install them, as well as an explanation on how to run your code and display the webpage. It is also useful to

Skip to main content

Furthermore, your code needs to be well commented and documented. **All** functions need to have docstrings explaining what the function does, how it should be used, an explanation of the parameters and return value(s) (including types). We recommend you use a well-established docstring style such as the Google-style docstrings.

We expect your code to be well-formatted and readable. Black is useful tool for automatically formatting Python code. Coding style and documentation will be part of the point evaluation for **all tasks** in this assignment.

A note on your implementation

In this assignment we have provided a smaller code skeleton than in previous assignments. As a result, you are more free in this assignment to implement the functionality we ask for in the way you find most convenient. You can freely reuse your code from previous tasks in this assignment.

You may use additional packages, but if you do, make sure to add them to `requirements.txt`.

We have provided some tests which your code should be able to pass, which should help clarify some specific requirements of the assignment.

You can run the tests with:

```
pytest -v tests/
```

Files to Deliver for this assignment

All files will be delivered in the same `assignment5` folder.

Required files:

- `README.md` including dependencies and their installation and commands you used for running your examples and creating your output files
- `requirements.txt` - specify package requirements. A starting point is provided, but if you use additional packages, make sure to add them here.
- `pyproject.toml`
- `strompris.py`

Skip to main content

- `templates/...`
- `docs/...`

If you choose to do bonus tasks (5.6 and 5.7) you will need to submit additional files, which are not present in the skeleton.

A last note on grading

Full points are awarded for each task if the python function is well documented and works as expected, and the web site behaves as specified. If you want to make life easy for the grader (which is always a good idea), we recommend making 110% sure that your website works as expected after you have pushed your solution. We have written some tips on that in the last section of the assignment.

5.1 Price Plotter

The strømpris API is very simple, and has only one endpoint with four inputs:

```
GET https://www.hvakosterstrommen.no/api/v1/prices/[year]/[month]-[day]_[location].
```

The input variables are:

Variable	Description	Example
Year	four digit year	2023
Month	two digits, with leading 0	02, 10
Day	two digits, with leading 0	02, 31
Location	location code, starting with “NO”, then a number 1-5	NO1

The following location codes are used:

Skip to main content

Code	Name
NO1	Oslo / Øst-Norge
NO2	Kristiansand / Sør-Norge
NO3	Trondheim / Midt-Norge
NO4	Tromsø / Nord-Norge
NO5	Bergen / Vest-Norge

(Note that this is "NO" as in NOrway, not a zero as in N0).

The following gives an example request for Trondheim for Nov 6., 2023:

```
url = "https://www.hvakosterstrommen.no/api/v1/prices/2023/11-06_NO3.json"
r = requests.get(url)
```

As you can see, one HTTP request gets data for one day and one location.

> **(3 points)** Write a Python function `fetch_day_prices` that takes a `datetime.date` object and location string as input. You can assume that the date is on or after the first of October, 2023. Your function should return a DataFrame containing the electricity prices for that day:

```
def fetch_day_prices(date: datetime.date=..., location: str=...) -> pd.DataFrame:
```

Both arguments should be **optional**, with defaults:

- `date`: the current date when the function is called
- `location`: "NO1" (Oslo)

The return value should be a pandas DataFrame with columns:

- NOK_per_kWh (float)
- time_start (datetime)

Skip to main content

> **Note**
>
> As you will find out, the input data requires some handling. In particular, there's an issue with crossing Daylight Savings Time. We leave it up to you to handle the details. Feel free to wrangle the data as you see fit, but one tip is to use
>
> ```
> df[...] = pd.to_datetime(df[...], utc=True).dt.tz_convert("Europe/Oslo")
> ```

Useful functions may include:

- datetime.date.today
- pd.DataFrame.from_dict
- pd.to_datetime
- pd.DataFrame.astype

Now that you have a convenient function to fetch one day of data for one location, it's time to collect *more* data so we can explore longer trends and comparisons.

> **(4 points)** Write a function `fetch_prices` that returns a DataFrame with the prices for a given time period:

```
def fetch_prices(end_date: datetime.date =..., days: int=..., locations: list[str]=
```

`fetch_prices` should have default inputs:

- end_date: today
- days: 7
- locations: all locations

where it fetches `days` days of data, up to and including `end_date` (i.e. with no arguments, it should fetch the latest 7 days of data). In addition to the columns returned by `fetch_day_prices`, the new DataFrame should add the columns:

- `location_code`: The location *code* (`NO2` or similar)
- `location`: The location *name* (`Oslo`, `Trondheim`, etc.)

Useful functions:

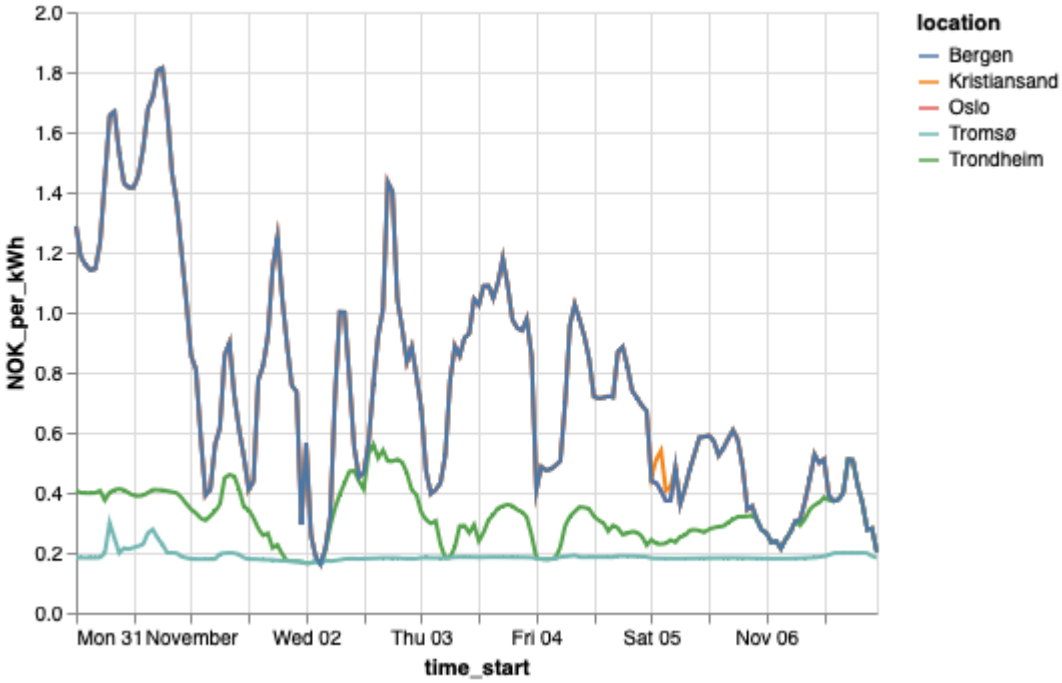
Skip to main content

- pd.concat
- adding a `datetime.timedelta` to a `datetime.date` (or subtracting)
- `dataframe["column"] = "scalar"` to add a column

> **(3 points)** Write a function `plot_prices` using altair to *plot* the resulting data.

```
def plot_prices(df: pd.DataFrame) -> alt.Chart:
```

The output produced by the script should be a line plot of the price over time, with one line for each region. It should look something like:



Once you've created the chart with your function, you can view it by either displaying it to the user or writing it to a .html file to view it with a browser. In the outline provided, we provide an example of displaying the chart using the built-in `chart.show()` method when you run `python3 strompris.py`.

Note that viewing the chart created by altair outside jupyter notebook or vs code for example might require installing the altair viewer or saving to a file. You can see the altair docs on viewing charts. The altair viewer package can be installed with:

```
pip install altair_viewer
```

Skip to main content

> **Note**
>
> If you use altair_viewer, it does not yet support altair 5, so you will need to specify `altair==4.*`:
>
> ```
> pip install altair==4.*
> ```
>
> (or add this to requirements.txt).

Files Required in this Subtask

- `strompris.py`

5.2: Becoming a Web Developer with FastAPI

> **(4 points)** Build a FastAPI app which uses your module in `strompris.py` from 5.1 to generate a plot of energy prices by date and display it on a web page.

The default time frame should be 7 days.

An outline of your app can be found in `app.py`. We also added an html template you can use in the directory `templates` named `strompris.html`.

For information on `FastAPI` we recommend taking a look at the lecture notes.

In particular, it should handle the following requests:

- `GET /` - renders the webpage with the chart (template in templates/strompris.html)
- `GET /plot_prices.json` returns the altair Chart JSON

Files Required in this Subtask

- `app.py`
- `templates/strompris.html`

Note You should allow FastAPI some seconds to load the webpage for you.

> **(1 point)** Extend the html template so that it it displays more than just the plot: Include

Skip to main content

5.3: Interactive Visualization: Upgrading to Pro-Level

screenshot of an example implementation

> **(5 points)** Modify your solution so far, so that the user can select a time range for the plot.

In order for this to integrate easily with the functions you have already defined, you can e.g. let the user select (i) an end date and (ii) the number of (preceding) days they want to plot for.

Hint: Add html inputs, like:

```
<input type=... id=... name=... value=... min=... max=... />
```

Inputs have a *type* which presents the right picker, e.g. for numbers, dates, text, etc.

and make sure the `name` attributes correspond to what is used in `async function refreshPlot()`, specifically that you populate the input names:

- `end`
- `days`

> **(5 points)** Modify your solution further so that the visitor of the web page can select the location(s). Add a checkbox input to select location(s) to plot, which should map to the `locations` argument of the `fetch_prices` function.

This should populated by checkbox inputs with `name="locations"`.

We further encourage you to have a look at the altair_web_demo example in the course repo, which shows an example of passing similar inputs to a chart function, specifically this part where we create checkbox inputs to be used in exactly the same way as you need for this assignment.

Skip to main content

> **Note**
>
> There is some javascript in the template. You are not required to write *any* javascript for this assignment, but it may be helpful to read and understand what's going on, and you are free to modify it if you wish. Mostly it gets inputs from your form by `name` and then builds a URL to send to your `/plot_prices.json`

5.4: Advanced visualization (IN4110 only)

When looking at a given point in time, often what's interesting is not so much the exact value, as the *trend*. That often means picking a comparable point in time, and showing the difference. To make this useful, we can add new columns with some derivative values in our DataFrame.

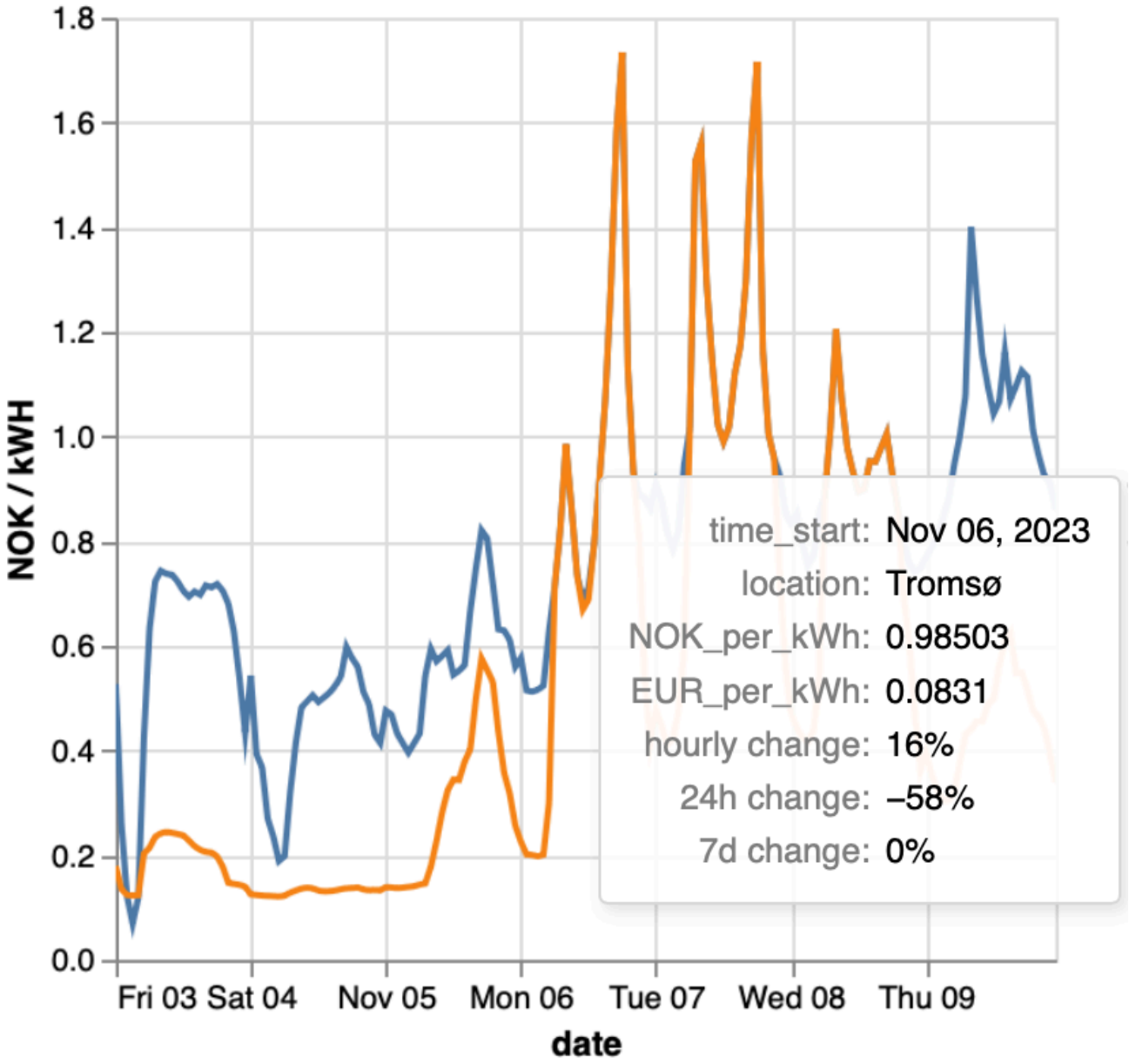
> **(5 points, IN4110 only)** Modify `fetch_prices` to add these 3 columns to the DataFrame it returns:
>
> - `1h change`: the price difference from the previous hour
> - `24h change`: the difference from the same hour on the previous day
> - `7d change`: the difference from the same hour on the same day of the previous week
>
> and make sure all 3 values are displayed in your chart's tooltip in `plot_prices`.

These should all be *relative* changes, for example given a few sample values for consecutive hours:

NOK_per_kWh	1h change	explanation
2	NaN	no data from previous hour to compare to
2.5	0.2	0.5 / 2.5 = 0.2
2	-0.25	-.5 / 2 = -0.25

Fantastic! You can check check that this works as expected by hovering your mouse over the lines in the plot to see the tooltip:

Skip to main content



> **(5 points, IN4110 only)** Finally, make a *compound* chart, showing the daily average for each day.

You should write an additional plot function (`plot_daily_prices`), which should present the daily average (mean or median) price. You can use whatever mark you feel renders this best: lines, points, box plot, etc.

This chart should be *added* to your existing chart, to make a *compound chart*. It can be next to, below, or even layered on top if you can make that look good :)

Some useful links:

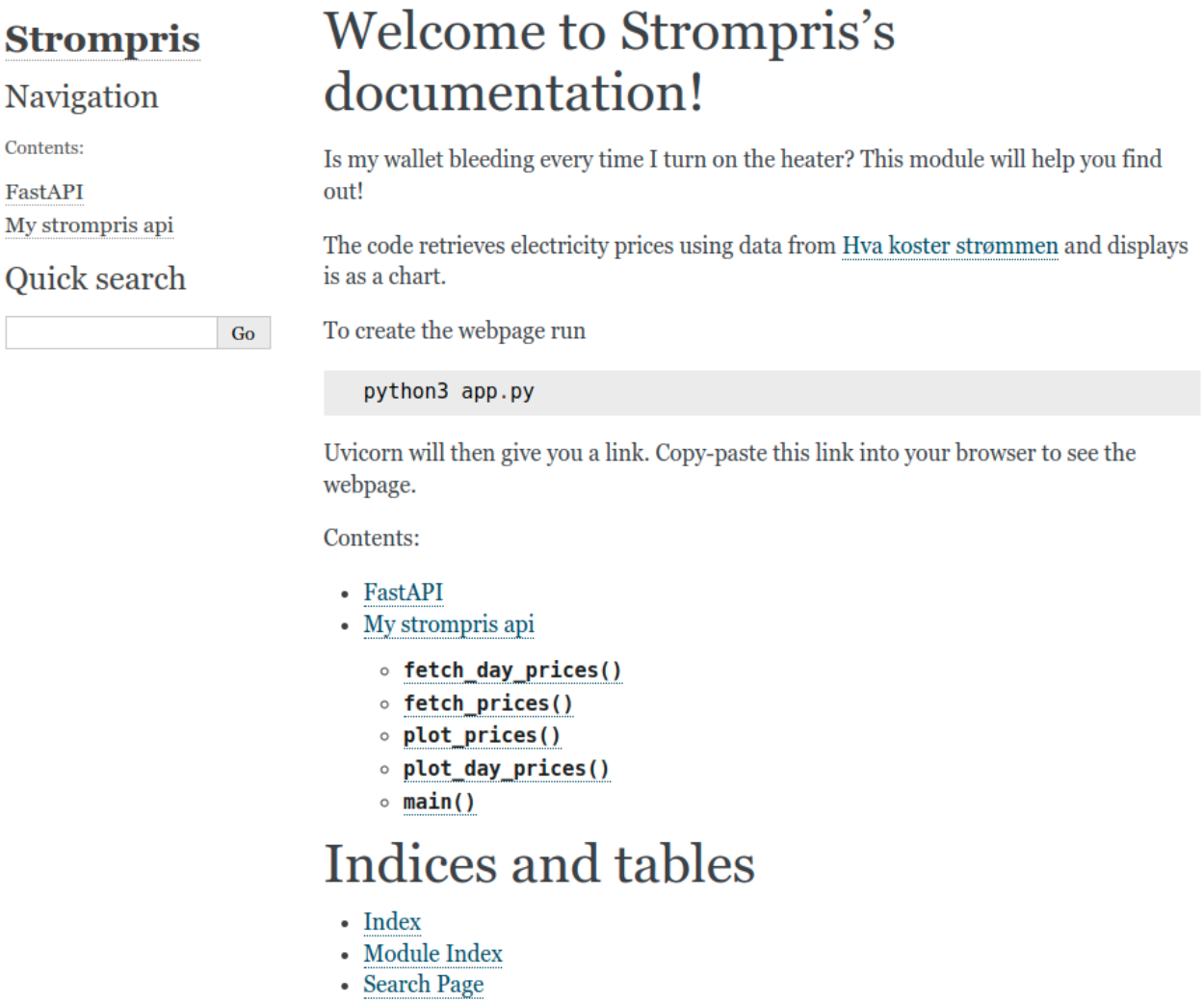
- pandas.Series.diff
- altair timeunit transforms

Skip to main content

- altair.hconcat
- altair box plot

5.5: Documentation and Help Page

In the next tasks we want you to create and/or link the documentation for your module. We start by creating Sphinx documentation for the overall project. `Sphinx` provides a way to quickly generate a documentation page for the project, using information you've already provided in your docstrings. After finishing the next task you should have a documentation home page that looks something like this:

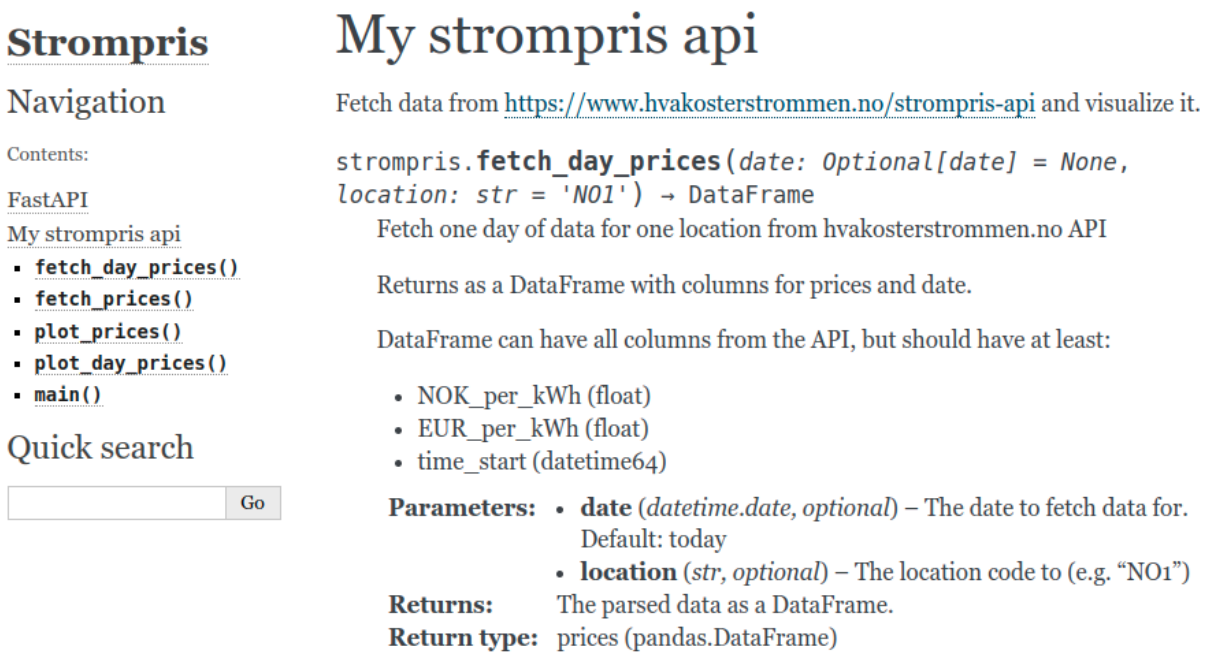


> **(3 points)** Use sphinx to generate documentation for your module. In particular, use `sphinx` to automatically generate documentation for `strompris.py`.

We recommend watching this short lecture video, where we walk you through creating the automated documentation

Skip to main content

The documentation for `strompris.py` should live in `strompris.md` or `strompris.rst` in your documentation directory, and should contain information about the functions you have implemented in `strompris.py`:



.

`Sphinx` generates so-called "static websites" (which are created when you run `make html`). FastAPI already has its own functionality for generating documentation. In the next task we want you to link to your `Sphinx` documentation and the FastAPI documentation.

> **(2 points)** Add links (on the plot webpage) to the documentation for the implementation. Link to both the sphinx documentation and the FastAPI documentation. One option to solve this could be by adding a navigation bar like the one shown below.

In the html file the code snippet for the navigation bar could look like this:

```
<!-- Add Navigation Bar. -->
<div class="topnav">
  <a class="active" href="/">Home</a>
  <a href="/docs">FastAPI Docs</a>
  <a href="/help">Help</a>
</div>
```

Skip to main content

The sphinx documentation should be available at `/help`, and the fastapi docs should be at `/docs`.

Files Required in this Subtask

- `app.py`
- `.html` file(s) for your help page. Naming and location depends on the tool of choice you used for creation. For sphinx, this is often `docs/_build/html`.

5.6: Upgrading your app to the Next Level (bonus for all)

Now that we've collected and visualized energy prices, let's try to gather some useful information from it.

> **(6 bonus points)** Add a second chart page to answer the question: **When should I take a shower/cook/heat my apartment?**

This means:

- implement `plot_activity_prices`, which takes:
  1. a DataFrame of energy prices (just like our other plot functions)
  2. an activity name, and
  3. a number of minutes

  and returns an altair Chart plotting the *actual price* for that activity
- create a new handler for `/plot_activity.json`, which should take inputs:
  - activity
  - minutes
  - location and call `plot_activity_prices` using prices for the current day.
- duplicate `strompris.html` template to `activity.html`, and update it to implement the new form inputs, and
- create a new handler for `/activity` that renders the new `activity.html` with the necessary inputs.

The inputs for the form should be:

Skip to main content

- dropdown for the location where the activity will take place
- number of minutes for the duration of the activity

The date need not be an input, it can always return data for today.

Use these activity energy costs:

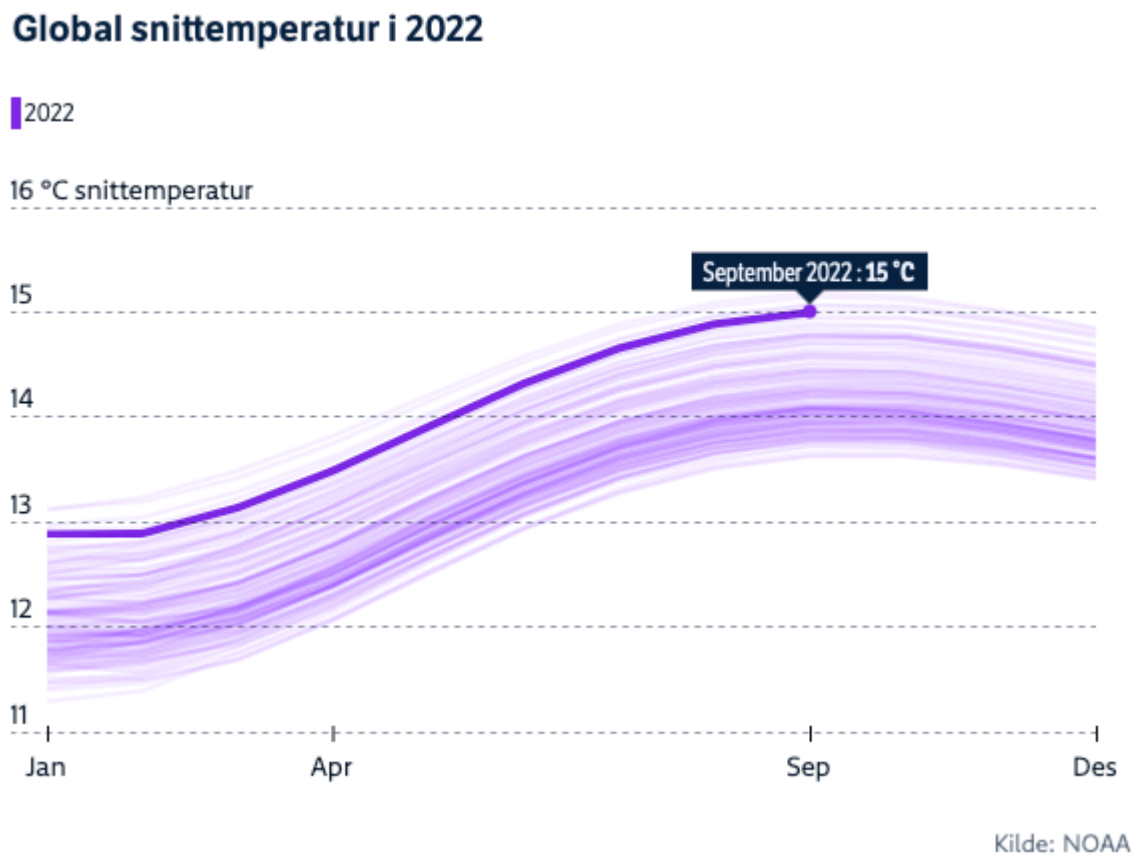
activity	energy usage
shower	30 kW
baking	2.5 kW
heat	1 kW

Note: do not consider activities that last more than an hour. We only need you to consider a single energy price for this calculation.

5.7: Did Someone Say More Bonus Points?! - Creating an Interactive Plot of the Climate Status (10 bonus points for all)

> **(10 bonus points)** Make another interactive visualization using Pandas and FastAPI to reproduce the interactive graphic about the global average temperature in 2022 as shown here in “Global snittemperatur i 2023”.

Skip to main content



The task is similar to the previous strømpris task (gather data, make chart, show on webpage), but we provide no code skeleton.

The data needed to create the plot can be found on the following websites. You will work with the combined mean surface temperature. The first link will provide you with the data for the anomalies recorded, which can be downloaded in form of a `.csv` file:

https://www.ncei.noaa.gov/access/monitoring/climate-at-a-glance/global/time-series/globe/land_ocean/all/1/1880-2023

The second link takes you to a website providing you with the mean monthly temperature for surface, water, and surface and water combined:

https://www.ncdc.noaa.gov/monitoring-references/faq/anomalies.php#mean

By taking the sum of the anomalies and the monthly mean temperature, you can calculate the mean temperature for each month for the different years.

For each year, plot the average temperature you calculated over the months of the year as a line plot. The graphs for different years should be made in a single plot, with this years data highlighted bold. Feel free to re-use or adapt functions created in earlier tasks. The end-product should be an interactive visualization using Pandas and FastAPI

Skip to main content

Allow for the user to "hover" over the line plot displaying a "tooltip" box containing the month they are hovering over, the coldest year (1904) and the corresponding temperature for that month. The "tooltip" should also contain the temperature measured for that month this year, as well as the warmest prior year measured (2016).

Files Required in this Subtask

- `klima/app.py`
- `klima/klima.py`
- `klima/templates/klima.html`

Checklist before submitting

- Remember to push to `https://github.uio.no/IN3110/IN3110-yourname` before the deadline
- All files should be in the folder `assignment5/` (we will not grade files delivered elsewhere)

Full points are awarded for each task if the python function is well documented and works as expected, and the web app behaves as specified.

To make life easier for yourself and our graders, we recommend you double check your submission, following this workflow

1. Commit and push your final solution
2. Make a fresh clone:

```
git clone git@github.uio.no/IN3110/IN3110-yourname
```

3. Enter the repo and run the application:

```
cd assignment5
python3 -m pip install -e .
python3 app.py
```

4. Then go the web address `http://127.0.0.1:5000/` and check that the website you made behaves as expected. This includes visiting the documentation page, plotting for different dates etc.

Skip to main content

The reason we recommend you to make a fresh clone of the repo for this check is to ensure you haven't forgotten to commit something, which might break the website.

Pat yourself on the shoulder - you finished the last big assignment! Gratulerer!

Previous
Assignment 4: Web scraping

Next
Week 1: git