

## Background

The Electric Reliability Council of Texas (ERCOT) services 90% of the state's electrical load.

ERCOT publishes data that market participants (e.g. wind farms that produce power, bitcoin mines that consume power, power traders that arbitrage forward and spot markets) can use to make decisions on how to operate their business. Some of this data is updated at regular intervals to reflect changing system (the power grid) conditions.

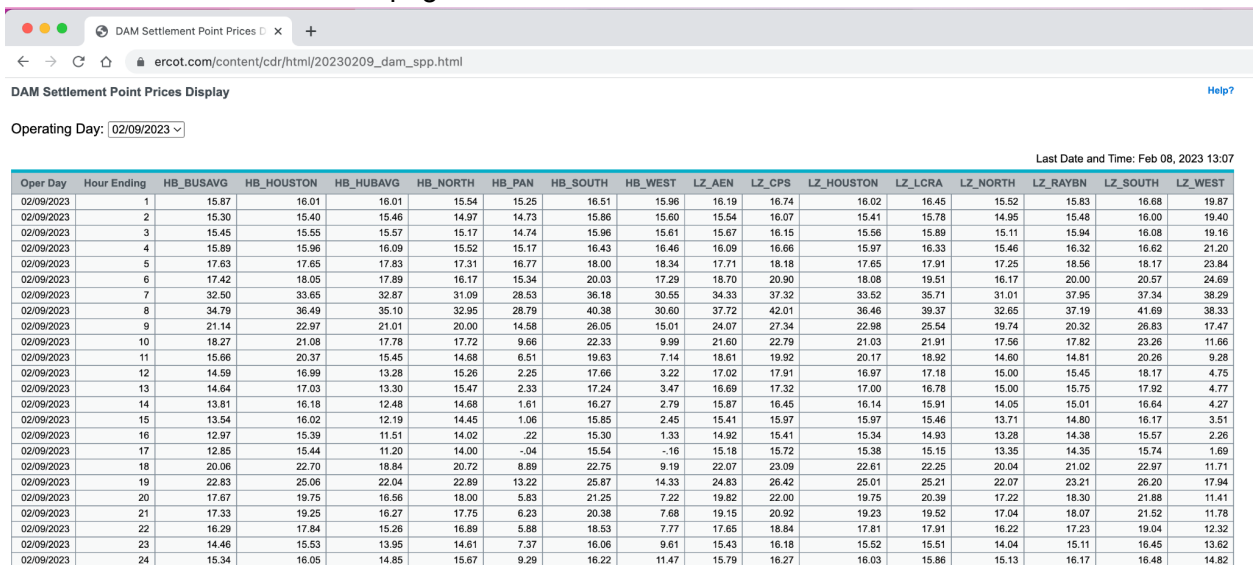
One data set that ERCOT publishes is the Day Ahead Market (DAM) clearing price for Hub and Load Zone settlement points. The ERCOT DAM is a forward market for energy and ancillary services. DAM clearing prices offer insight into how market participants believe the spot price of power will behave in real-time. ERCOT publishes DAM clearing prices for the next day after 2 PM CT. To read a full definition of DAM see [here](#).

If this jargon is confusing, that is ok! The ultimate goal is described below.

## Goal

Create a Rust [binary application](#) (% cargo new binary-name) that:

1. Retrieves data from ERCOT's DAM Settlement Point Prices Display for the most recent available date.
  - a. The url to access ERCOT's DAM Settlement Point Prices Display page takes the following format:  
`https://www.ercot.com/content/cdr/html/YYYYMMDD_dam_spp.html`, where the date in **red** is the date you want to access the page for. For example, to access the page on March 8, 2023, the URL is:  
`https://www.ercot.com/content/cdr/html/20230308_dam_spp.html`
  - b. Note that these prices update daily at 14:00 UTC, and only 7 days of data are retained. The full web page with all data should look like this:



Oper Day	Hour Ending	HB_BUSAVG	HB_HOUSTON	HB_HUBAVG	HB_NORTH	HB_PAN	HB_SOUTH	HB_WEST	LZ_AEN	LZ_CPS	LZ_HOUSTON	LZ_LCRA	LZ_NORTH	LZ_RAYBN	LZ_SOUTH	LZ_WEST
02/09/2023	1	15.87	16.01	16.01	15.54	15.25	16.51	15.96	16.19	16.74	16.02	16.45	15.52	15.83	16.68	19.87
02/09/2023	2	15.30	15.40	15.46	14.97	14.73	15.86	15.60	15.54	16.07	15.41	15.78	14.95	15.48	16.00	19.40
02/09/2023	3	15.45	15.55	15.57	15.17	14.74	15.96	15.61	15.67	16.15	15.56	15.89	15.11	15.94	16.08	19.16
02/09/2023	4	15.89	15.96	16.09	15.52	15.17	16.43	16.46	16.09	16.66	15.97	16.33	15.46	16.32	16.62	21.20
02/09/2023	5	17.63	17.65	17.83	17.31	16.77	18.00	18.34	17.71	18.18	17.65	17.91	17.25	18.56	18.17	23.84
02/09/2023	6	17.42	18.05	17.89	16.17	15.34	20.03	17.29	18.70	20.90	18.08	19.51	16.17	20.00	20.57	24.69
02/09/2023	7	32.50	33.65	32.87	31.09	28.53	36.18	30.55	34.33	37.32	33.52	35.71	31.01	37.95	37.34	38.29
02/09/2023	8	34.79	36.49	35.10	32.95	28.79	40.38	30.60	37.72	42.01	36.46	39.37	32.65	37.19	41.69	38.33
02/09/2023	9	21.14	22.97	21.01	20.00	14.58	26.05	15.01	24.07	27.34	22.98	25.54	19.74	20.32	26.83	17.47
02/09/2023	10	18.27	21.08	17.78	17.72	9.66	22.33	9.99	21.60	22.79	21.03	21.91	17.56	17.82	23.26	11.66
02/09/2023	11	15.66	20.37	15.45	14.68	6.51	19.63	7.14	18.61	19.92	20.17	18.92	14.60	14.81	20.26	9.28
02/09/2023	12	14.59	16.99	13.28	15.26	2.25	17.66	3.22	17.02	17.91	16.97	17.18	15.00	15.45	18.17	4.75
02/09/2023	13	14.64	17.03	13.30	15.47	2.33	17.24	3.47	16.69	17.32	17.00	16.78	15.00	15.75	17.92	4.77
02/09/2023	14	13.81	16.18	12.48	14.68	1.61	16.27	2.79	15.87	16.45	16.14	15.91	14.05	15.01	16.84	4.27
02/09/2023	15	13.54	16.02	12.19	14.45	1.06	15.85	2.45	15.41	15.97	15.97	15.46	13.71	14.80	16.17	3.51
02/09/2023	16	12.97	15.39	11.51	14.02	.22	15.30	1.33	14.92	15.41	15.34	14.93	13.28	14.38	15.57	2.26
02/09/2023	17	12.85	15.44	11.20	14.00	-.04	15.54	-.16	15.18	15.72	15.38	15.15	13.35	14.35	15.74	1.69
02/09/2023	18	20.06	22.70	18.84	20.72	8.89	22.75	9.19	22.07	23.09	22.61	22.25	20.04	21.02	22.97	11.71
02/09/2023	19	22.83	25.06	22.04	22.89	13.22	25.87	14.33	24.83	26.42	25.01	25.21	22.07	23.21	26.20	17.94
02/09/2023	20	17.67	19.75	16.56	18.00	5.83	21.25	7.22	19.82	22.00	19.75	20.39	17.22	18.30	21.88	11.41
02/09/2023	21	17.33	19.25	16.27	17.75	6.23	20.38	7.68	19.15	20.92	19.23	19.52	17.04	18.07	21.52	11.78
02/09/2023	22	16.29	17.84	15.26	16.89	5.88	18.53	7.77	17.65	18.84	17.81	17.91	16.22	17.23	19.04	12.32
02/09/2023	23	14.46	15.53	13.95	14.61	7.37	16.06	9.61	15.43	16.18	15.52	15.51	14.04	15.11	16.45	13.62
02/09/2023	24	15.34	16.05	14.85	15.67	9.29	16.22	11.47	15.79	16.27	16.03	15.86	15.13	16.17	16.48	14.82

- Extracts and stores the following table columns: LZ\_HOUSTON, LZ\_NORTH, LZ\_SOUTH, and LZ\_WEST. This data should be stored in a Rust data structure called a struct ([definition](#), [example](#)). You may also find that Rust's [Vec type](#) will come in handy. For clarity, these columns are boxed in red:

Oper Day	Hour Ending	HB_BUSAVG	HB_HOUSTON	HB_HUBAVG	HB_NORTH	HB_PAN	HB_SOUTH	HB_WEST	LZ_AEN	LZ_CPS	LZ_HOUSTON	LZ_LCRA	LZ_NORTH	LZ_RAYBN	LZ_SOUTH	LZ_WEST
02/09/2023	1	15.87	16.01	16.01	15.54	15.25	16.51	15.96	16.19	16.74	16.02	16.45	15.52	15.83	16.68	19.87
02/09/2023	2	15.30	15.40	15.46	14.97	14.73	15.86	15.60	15.54	16.07	15.41	15.78	14.95	15.48	16.00	19.40
02/09/2023	3	15.45	15.55	15.57	15.17	14.74	15.96	15.61	15.67	16.15	15.56	15.89	15.11	15.94	16.08	19.16
02/09/2023	4	15.89	15.96	16.09	15.52	15.17	16.43	16.46	16.09	16.66	15.97	16.33	15.46	16.32	16.62	21.20
02/09/2023	5	17.63	17.65	17.83	17.31	16.77	18.00	18.34	17.71	18.18	17.65	17.91	17.25	18.56	18.17	23.84
02/09/2023	6	17.42	18.05	17.89	16.17	15.34	20.03	17.29	18.70	20.90	18.08	19.51	16.17	20.00	20.57	24.69
02/09/2023	7	32.50	33.65	32.87	31.09	28.53	36.18	30.55	34.33	37.32	33.52	35.71	31.01	37.95	37.34	38.29
02/09/2023	8	34.79	36.49	35.10	32.95	28.79	40.38	30.60	37.72	42.01	36.46	39.37	32.65	37.19	41.69	38.33
02/09/2023	9	21.14	22.97	21.01	20.00	14.58	26.05	15.01	24.07	27.34	22.98	25.54	19.74	20.32	26.83	17.47
02/09/2023	10	18.27	21.08	17.78	17.72	9.66	22.33	9.99	21.60	22.79	21.03	21.91	17.56	17.82	23.26	11.66
02/09/2023	11	15.66	20.37	15.45	14.68	6.51	19.63	7.14	18.61	19.92	20.17	18.92	14.60	14.81	20.26	9.28
02/09/2023	12	14.59	16.99	13.28	15.26	2.25	17.66	3.22	17.02	17.91	16.97	17.18	15.00	15.45	18.17	4.75
02/09/2023	13	14.64	17.03	13.30	15.47	2.33	17.24	3.47	16.69	17.32	17.00	16.78	15.00	15.75	17.92	4.77
02/09/2023	14	13.81	16.18	12.48	14.68	1.61	16.27	2.79	15.87	16.45	16.14	15.91	14.05	15.01	16.64	4.27
02/09/2023	15	13.54	16.02	12.19	14.45	1.06	15.85	2.45	15.41	15.97	15.97	15.46	13.71	14.80	16.17	3.51
02/09/2023	16	12.97	15.39	11.51	14.02	.22	15.30	1.33	14.92	15.41	15.34	14.93	13.28	14.38	15.57	2.26
02/09/2023	17	12.85	15.44	11.20	14.00	-.04	15.54	-.16	15.18	15.72	15.38	15.15	13.35	14.35	15.74	1.69
02/09/2023	18	20.06	22.70	18.84	20.72	8.89	22.75	9.19	22.07	23.09	22.61	22.25	20.04	21.02	22.97	11.71
02/09/2023	19	22.83	25.06	22.04	22.89	13.22	25.87	14.33	24.83	26.42	25.01	25.21	22.07	23.21	26.20	17.94
02/09/2023	20	17.67	19.75	16.56	18.00	5.83	21.25	7.22	19.82	22.00	19.75	20.39	17.22	18.30	21.88	11.41
02/09/2023	21	17.33	19.25	16.27	17.75	6.23	20.38	7.68	19.15	20.92	19.23	19.52	17.04	18.07	21.52	11.78
02/09/2023	22	16.29	17.84	15.26	16.89	5.88	18.53	7.77	17.65	18.84	17.81	17.91	16.22	17.23	19.04	12.32
02/09/2023	23	14.46	15.53	13.95	14.61	7.37	16.06	9.61	15.43	16.18	15.52	15.51	14.04	15.11	16.45	13.62
02/09/2023	24	15.34	16.05	14.85	15.67	9.29	16.22	11.47	15.79	16.27	16.03	15.86	15.13	16.17	16.48	14.82

- Exports the data structure as a CSV file.

## Deliverables

- Please use git to track changes in your code, and push to a private repo on your GitHub account. Please share the GitHub repo with GitHub user [rybarczyk](#).
- When we meet to review, you will share your screen on a zoom meet and run the program in your own dev environment.

## What we are looking for

- Above all, we are looking for someone who is able to put together a basic Rust project. We understand Rust is a new language for many people, and are not testing who is the most proficient Rust dev, but rather can you adapt your programming skills to pick up this new language by referencing online documentation (like The Rust Book, the Rust standard library, and Rust package docs) and resources (like StackOverflow or ChatGPT - be warned, ChatGPT is a great resource but it won't write this code for you. If you rely solely on ChatGPT you may run into more headaches). We want to mimic a real world environment, not trip candidates up on obscure algorithms.
- If your final project does not compile, that is ok too. The Rust compiler is very strict and can take some getting used to. Just submit the most complete version of your work.
- Everything to run your code (even if it does not compile) needs to be committed to the GitHub repo. Really all the files you need is what `cargo new` provides you out of the box.

- Bonus things that we like:
  - Write readable code, adding comments where necessary (try to avoid obvious comments, like `let x = 1; // Set x equal to 1`)
  - Write code using proper Rust syntax (like using lower snake\_case for variable names and upper camel case for struct names). You are encouraged to use [cargo clippy](#) to help!
  - A couple unit tests that can be run using [cargo test](#).
  - If you completed the task and want to do more, you can make the code automatically detect the date and construct the URL dynamically with the latest date.
  - Error handling. Unwraps are totally fine, but if you finish early give error handling a try.

### Helpful Hints/Starting Points

- [reqwest crate](#)
  - Use the blocking feature flag in your Cargo.toml file as: `reqwest = { version = "0.11.14", features = [ "blocking" ] }`
  - And refer to the [reqwest::blocking docs](#)
- [scraper crate](#)
  - You technically do not need this as the reqwest crate can pull down the information you need, but scraper makes it easier to extract this data.
- [csv crate](#)
  - Plays nicely with the serde crate :)
- [serde crate](#)
  - Use the derive feature flag in your Cargo.toml file as: `serde = { version = "1.0.152", features = [ "derive" ] }`
  - The serde crate has some [great docs](#). There is no need for any custom serialization or deserialization implementations (but don't let that limit you if you want to go there). The most helpful content for this project is [how to use the Deserialize custom derive on your struct](#), and the [rename field attribute](#) for your struct fields.
- [chrono crate](#)
  - Totally optional crate, but throwing it in there in case you find a use case for it. Depending on how you use it, you may want to look into that [custom deserialization implementation](#) I told you you didn't need. Don't get too hung up here though.
- You are encouraged to use the documentation and take from any examples in the documentation, crate git repo README files and/or example directories.
- If you run into trouble and are beating your head against the wall on a bug or compiler issue, text me and we can work through it together! My email is [rachel.rybarczyk@galaxy.com](mailto:rachel.rybarczyk@galaxy.com).

## Helpful Links General to Rust

- [Rust Installation Guide](#)
- [A very brief guide of Rust's syntax](#)
- [The Rust Book](#)
- [Rust by Example](#)
- [The Rust Standard Library Documentation](#)
- [Rust external packages \(called crates\) registry](#)
- [Specifying Rust dependencies in your Cargo.toml file](#)
- [Error handling in Rust](#)