Tradeoffs:

Time: Weather for any requested time will return weather for that hour. i.e. '2018-03-01 13:40:34' will return weather for 13:00 on that date, as will '2018-03-01 13:01:34'.

Data's time detail: We floor the minutes down to the current hour. This helps reduce API calls and keeps things clear because it's clearer and less expensive to parse the response for minutely data and the prompt said an hour interval is sufficient.

Datetime Parsing: Parsing date time rather than accepting only a standardized format can take longer but is more user friendly and more of a defensive programming technique.

Break and give error message to user: When things go wrong give a useful error message rather than trying to recover. The error messages should be pretty clear, either data is not available for that location or your timestamp couldn't be parsed.

Fixes/Missing Functionality:

Testing - Testing is a bit too sparse because of the time frame I set for myself. I would like to work toward 100% test coverage.

Database - Obviously it is not ideal for every user to have a sqlite db file on their disk so hopefully we would be able to push that to an existing db over the network.

Time - Time returned is not exactly the time requested in that location's timezone because the API takes requests in GMT time and returns the time in the local timezone of the requested location. However, we do store the result as the time that the user requested, so at any point you could adjust for time zone given latitude and longitude. I didn't work on fixing this in the interest of time, especially considering we're starting with Seattle and only expanding to two more locations.

Location - Related to the location quirks, there are locations that don't return weather data. Shouldn't really be a problem in the Seattle area or the two mentioned expansion cites.

Output/DB write - The way it is written now, the user has to wait for the function to push the requested data to our database before they get their output. This shouldn't be *too* long since we're only going to be uploading 23 rows of about 5 reasonable sized columns at any given time. I wasn't sure how to remedy this, I think it could be resolved by getting a thread open to work on that while we return the data for the requested time to the user.

Refinements:

There are some things I would like to do:

Testing could use a bit more work to have robust coverage.

There is an error that occurs if you put in an invalid timestamp like ‘dec 34 2019’ the error message in this case says that your format is bad, if you change it to ‘2019-12-34 12:00:00’ it will still say that, the problem is the date is out of range which the parser library’s error does explain. Minor as the user should be able to figure out what was wrong with that timestamp pretty clearly.

I didn’t pay much attention to the setup or schema of the database, I figured there would be one set up or done outside of this function that would have been created with more thought. For the purposes of having a solution that implements some method of caching I thought this was enough.

Refactoring
There is some refactoring that could be done particularly to better encapsulate some data and functionality, like with the way I work with timestamps here, the code in the main module could be simplified if there was a class surrounding what we might call a WeatherTimestamp.

Related to this is the rebuilding of the string to go from ‘2018-03-02 12:00:00’ to ‘2018-03-02 T12:00:00’. I have an extra function just to build a new string replacing that space with a T that may be able to be avoided if we look deeper into timestamp modifications or building of request urls, but this is just a linear time operation on a short string so I don’t think it is a huge issue outside of code cleanliness.

Module folder structure:
For simplicity and in the interest of time I kept the tests and helper modules in the same directory, given more time and understanding of how the code would be used I would want to reorganize this to have these files in their own directories.