DataStream

A repository containing all the code for data builders alongside a user-facing python library other projects may use to interact with the datasets.

What this project should be called is still to be decided.

```
datastream/
+ builders/
  + equities/
  + ml-features/
  + etc.
+ public/
  + library code
```

Data builders

The steps for registering a new dataset are as follows:

1. Write a data builder file somewhere under `builders/`:
   - I've chosen a decorator pattern to specify the datasets, but an alternative approach could be through inheritance. I favoured this because it makes for less boilerplate code, especially if we wish to create a light dataset without much functionality needed.

   ```py
   # builders/example_builder.py
   from datastream.builders import data_source, dataset, field, dependency
   from datastream.calendars import NyseTradingDays
   import pandas as pd

   trading_days = NyseTradingDays()

   # these decorators shouldn't be limited to functions,
   # they should be able to wrap any Callable
   @data_source(
     name="closes",
     version="1",
     calendar=trading_days,
     start_date=date(2020, 1, 1),
     fields=[
       field("symbol", type=str),
       field("close_price", type=float),
     ]
   )
   def download_close_prices(now: datetime):
     close: pd.DataFrame = something.download_close(now)[["symbol", "close"]]
     return close

   @dataset(
     name="mav5",
     version="1",
     calendar=trading_days,
     dependencies=[dependency("closes", version="1", lookback=5)],
     fields=[
       field("symbol", type=str),
       field("moving_average", type=float)
     ]
   )
   def calc_mav5(now: datetime, closes: pd.DataFrame):
     return closes.groupby("symbol").apply(
       lambda df: df["close"].sum() / 5
     ).rename(columns={"close", "moving_average"})
   ```

2. Some functionality to register this data builder file. This would automatically create a (empty) table in the databases for this dataset. We will mostly be working with time-series data so SQL is the clear approach. There are some things that need to be kept in mind when designing this:
   a. *Dataset versioning:* If the data builder for a certain data set ever gets updated, we should keep both the old version and new version available for access. Two datasets with the same name but different versions should be treated as entirely different datasets. That is, datasets are indexed by `(name, version)`.
   b. *New datasets:* Only register new datasets, we do not want multiple tables in the database dedicated to a single dataset.

   Ideas for registering the new files:
   a. Similar to what django does, and create a `manage.py` that scans through all of `builders/` for new data sources and datasets, then register those.
      - The issue here is how do we detect "new" datasets, and how do we detect datasets with new versions.
      - There is also a minor concern with performance if we have to walk through the entire `builders/` directory every time.
   b. Make registering manual, create a separate `register.py` that would be used like:

   ```
   python3 register.py example_builder.closes example_builder.mav5
   ```

3. Build the dataset. There are also several potential approaches to this:
   a. *Lazy building:* Only build specific data when a user requests it. This laziness would require recursively build the dependency chain. A concern with this is potentially long delays for when a user wants to use un-built data.
   b. *Daily build jobs:* Set up scheduled jobs to build all datasets every day. The concern with this is that we might not have enough compute resources to pull this off.
   c. *Dedicated building script:* A file `build.py` with usage:

   ```
   python3 build.py <dataset-name> <dataset-version> <start-date> [end-date]
   ```

   This gives the user control over when datasets are built. Of course this can be layered on top of the lazy building or daily build jobs.
   d. *Light datasets are not stored in database:* This is just an idea, but datasets that require very little computation to build can optionally not be stored in the databases, to save storage space.

Exposed Python library

Users of the datasets would simply import the library and use the data.

- Should there be a separate library form

```py
# example_usage.py
import datastream.reader as dsr

mav5 = dsr.get_dataset("mav5", version="1")  # dataset object
today_mav5 = mav5.get_data(today)  # pd.DataFrame
aapl_mav5 = today_mav5[today_mav5["symbol"] == "AAPL"]
print(aapl_mav5)
```