This dashboard incorporates several elements divided into distinct files for organizations.

Main.py — this file contains the dashboard code and serves as the front-end organization file.

Support_functions.py — this file is the workhorse of the system. Everything from report retrieval to actual chart creation lives here.

Business_logic.py — this file serves as the bridge to collecting the data and setting up the necessary dataframes.

Layout_configs.py — this file contains helper elements for styling charts the way I want to see them and defining the tools I want to use.

If I were to refactor this further, I could eliminate some of these files just by reorganizing. However, as systems grow, it provides a nice balance for incorporating other functions appropriately. For example, in my larger applications, my business_logic file handles database calls and a lot more data wrangling before things get passed on.

Support Functions

Let's begin by talking about some of the background functions we need to get the data. We can't really do much without some data.This dashboard incorporates several elements divided into distinct files for organizations.

Main.py — this file contains the dashboard code and serves as the front-end organization file.

Support_functions.py — this file is the workhorse of the system. Everything from report retrieval to actual chart creation lives here.

Business_logic.py — this file serves as the bridge to collecting the data and setting up the necessary dataframes.

Layout_configs.py — this file contains helper elements for styling charts the way I want to see them and defining the tools I want to use.

If I were to refactor this further, I could eliminate some of these files just by reorganizing. However, as systems grow, it provides a nice balance for incorporating other functions appropriately. For example, in my larger applications, my business_logic file handles database calls and a lot more data wrangling before things get passed on.

Support Functions

Let's begin by talking about some of the background functions we need to get the data. We can't really do much without some dataThe two functions above (formatted horribly in Medium's code box) do several things. First, the function get_reports() evaluates to see if the text report files exist. If they do, they are checked to see if they are "fresh" as in they are less than or equal to seven days old. If they are good, the existing files are used. If not, new ones are retrieved. Likewise, if the files do not exist, they are retrieved from the CFTC website.

Retrieval of files uses the first function which is get_COT and takes the url and filename as arguments. These functions use python modules zip file, url lib, shutil, os, and datetime. Two variables are set to define path.

If using this code, remember to set those variables appropriately for your situation.

Once we have the files in place, we have data to work with. But, it's still raw. The next two functions in the file "deacot_process" and "DA_process" goes through some dataframe modifications.

Generally, these functions first, rename a number of columns to make things easier to work with. Next, they sort the dataframe by date to ensure things are in order in an expected way. After that, some new, calculated columns are created for easy reference later. Finally, the "exchange" column is split and two new columns for commodity and market are created. The last step is necessary to provide for some labels in charts and other locations later.

Since this was a chart-first process, the charts were defined prior to incorporation into the dashboard framework. It made sense to simply create them as callable functions rather than take other approaches. But, as with anything, there is more than one way to tackle the challenge.

The key to remember is that the chart is returned by the function as a ready-to-go object. That means you can call it as the output of a function, as a figure variable, or inline as part of the layout object. Your choice really comes down to organization and use.

Business Logic

The business_logic.py file is very simple in this application. At 41 lines total, including a lot of comments, it really just provides an entry into running certain functions when the application starts or reloads.

The general layout is as follows:

Set up the dataframe backend for plotting to be plotly.

Configure the default template to be a dark theme.

Get the reports using the function

Read the CSVs into a dataframe and process appropriately

Create a new array of just the unique values in the exchange column

Item 5 is used to create the master list of commodities that we will use to drive the dashboard views and update the charts appropriately.