Data Visualization and Analytics for Financial Markets: The Python Financial Data Platform

Wyatt Marciniak

April 10, 2019

Contents

Introduction & Motivation

Data sourcing is a key element for research, modelling and analysis. The best resources with respect to breadth and depth, such as Bloomberg Research Terminals used in the Stevens Institute of Technology Financial Labs, provide the best competitive advantage but come with a large cost of ownership. Retail investors, those who trade outside of banks, firms and funds, view the operating cost as a large barrier to entry for these types of data sources. These individuals' data needs are also, predominantly, no where near as large as the pool of data Bloomberg (and equivalent) services provide.

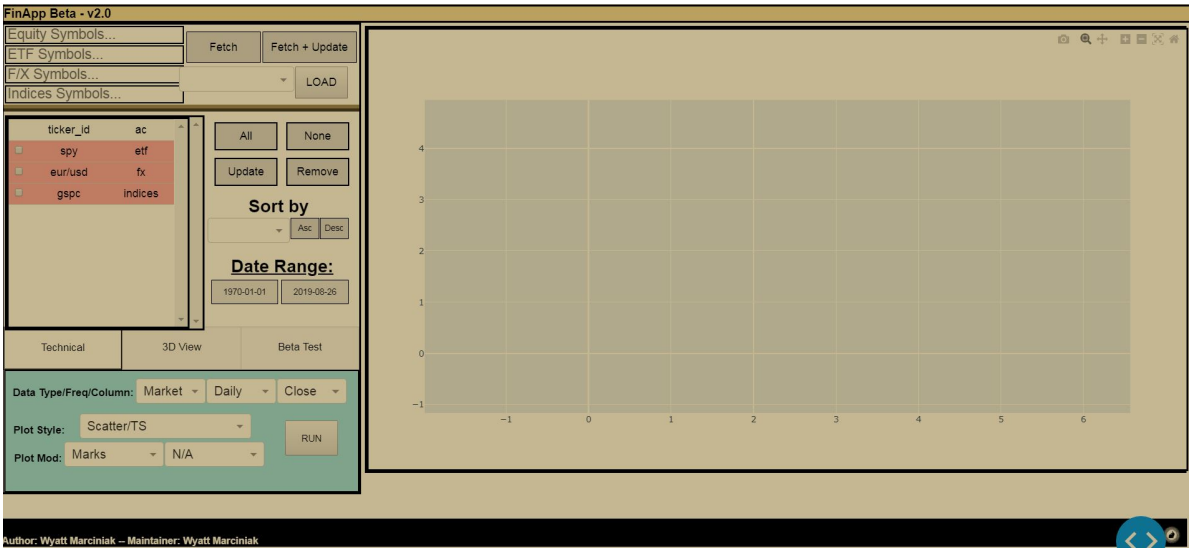
Cheaper options exist but they are still steeply priced, such as Money.net costing around 300 U.S. Dollars (USD) per month. At this point, investors could turn to API services and use programming languages such as R and Python to interface with them and source data from their hosts (owners). Big data APIs are cheaper than applications but their costs are usually still relatively high because they are targeted for heavy use clients. Smaller server services, such as Quandl, are commonly used alternatives. They offer small samples of free data and paid services to access their full server database which is centered around core market asset data. These assets are generally equities, Exchnage Traded Funds (ETFs), options and Foreign Exchnage (FX) activities. While this is convenient, targeted and cost effective, there is yet still another option that improves upon the price. R and Python are open source languages and libraries have been developed, such as Quantmod in R, to extract data for free through programs or algorithms. The functions were written by a 3rd party developer and shared with the public under an open (to use/share/modify) license.

Open source APIs are a huge resource for sourcing data, and wide adoption by users keeps them maintained and operational. For certain developers however, there is still one more issue to address. Open source code is usually convenient but is also maintained by a small group of developers instead of a large firm with funding, manpower and resources to maintain the back-end (servers and data). This means that the probability of error is higher, but more importantly it means an increase in recovery time when patches (fixes) are needed. In addition to that, custom adaptations are harder to implement and for larger designs or designs where data extraction needs to be dynamic and continuous, the user must rely largely on the author of the API for operational stability. This creates a problem with building applications and tools, especially with respect to financial markets and trade execution where the data needs to be accurate, timely and reliable; flaws in design can lead to things like incorrect trades and missed opportunities to trade or execute a strategy at all. In both scenarios, the investor is losing in the markets. Therefore, to keep costs low (or at zero) and be able to create tools for sourcing the necessary data, the investor should turn to creating web scraping APIs themselves. They exist, as shown, but the tools and knowledge to build them are widely available to those who want to learn.

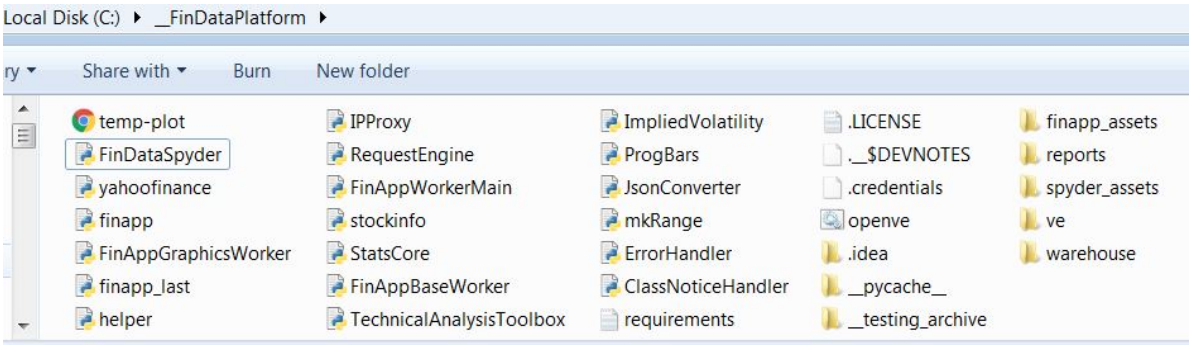
The potential of web scraping, for financial and almost every other type of data, is almost infinite because the algorithms are developed to search the largest accessible data source on Earth: The Internet. Google, for example, uses them daily to "crawl" websites' pages to catalog them for search references as well as search for optimal advertisement locations. For investors, the web scrapers could be used to collect financial market data on many asset types, rates and indicators whose data is displayed on public web pages (such as Yahoo Finance and Nasdaq). Once functional algorithms are tested and stable at retrieving data from their target ending nodes, the data can be sourced into the local language's session (whatever language the web scrapers are written in), cleaned, manipulated, referenced and/or saved in the user's local system memory for later use. This methodology allows for custom implementations, tailored to investment strategies or client needs and allows the user to be able to edit the APIs or fix/update them when needed. This puts the user in control and therefore the user can react and adapt new solutions without relying on any second or third party developers. Designing as much of the system in-house as possible ensures the developing project or application behaves as expected and error testing can be done more accurately. Web scraping financial data is a cost-effective way of getting targeted data when and where it is needed with only the need for an internet connection. The user will not have to pay significant amounts of money just to have the data they need to compete and without being tethered to a service or company, the APIs are flexible and portable, even between systems.

This application is a financial data platform, designed to aggregate market data and metrics across the entire market, but without the ability to use that data in a manageable way, it is just as useless as no data at all. With the integretion of Dash (by Plotly) for Python, the HTML and CSS structures used in building web based applications is wrapped nicely to integrate with the local backend. Plotly itself, is an immense visualization library spanning across multiple lanquages (including Python, R and JS - JavaScript) and provides a tool box of APIs and source code to achieve the goal of seeing and manipulating data as an object in space.

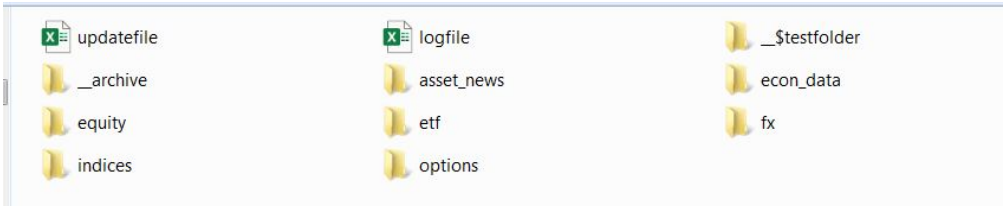
This application, as I will show, allows the user to implement many diverse visualization techniques on market data to obtain a better understanding of its behaviour and relationship to the world of assets around it. This application is designed to build and mainatain a constantly growing data warehouse, where the the backend, at this stage in development, is stable and self-maintaining. This is a key element to have when visualizing data because the user expects reliable and consistent operation, as well as flexability to scale as their investment universe grows. At this time, market data in daily, weekly and montly frequencies for Open, High, Low, Close, Volume and Adjusted prices are integrated, as well as intraday data sets but if you look through the code and warehouse (A copy of the current projet state is being submitted), you will see the full breadth and depth of the assets, data types, backend archiving and all maintenance directories built and used by this platform. This tool is to become the main data anaysis and visualization tool for a Hedge Fund Startup that has been developing for the last few months. The tools integrated here are in their first stage of full development space, however, this version is extremely stable and versatile, and the visualization tools were targetted as a large milestone in the integration process. Feel free to spin up the application and try it out, all files needed to run the scripts are present in the submission, including the virtual environment used to build and test this v2.0.0 beta application.



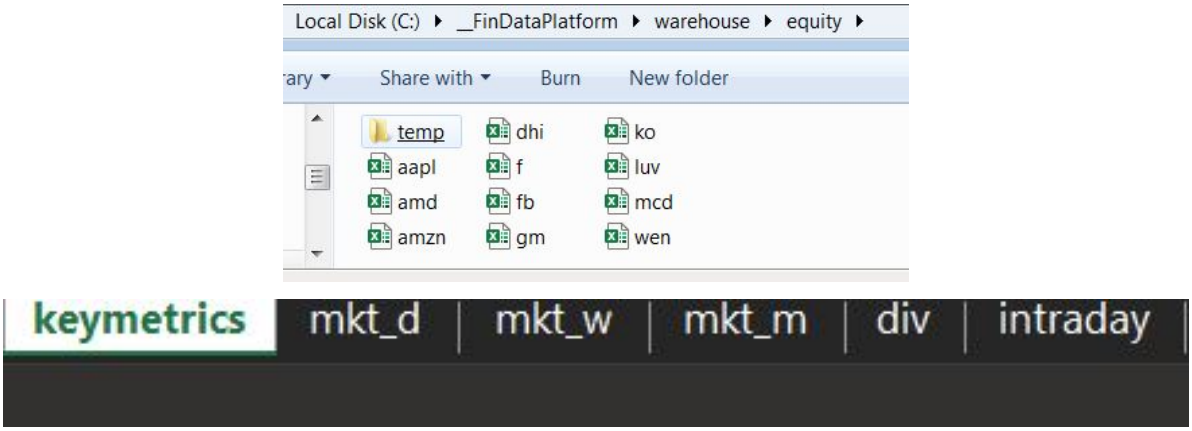
Application Structure



The root directory (shown) holds all files and scripts the application needs to run, All (almost) python scripts are dedicated classes with dedicated assignments, and those assignments form the building blocks of the class inheritance chain. The application sources data by requests for assets and tickers sent through FinDataSpyder. This class is in charge of updating, cleaning, and maintining the backend warehouse data files.



The warehouse stores asset profiles by asset class (Excel workbooks (.xlsx)) where each sheet is a data type, and new data is appended to the assets per these sheet ('table') seperations.



The update file holds the last update time per data type, per asset. This is done to keep the web-scrapers from always trying to fetch new data at every request. The current application is set (chanegable) to try and fetch / load 3 assets at startup (SPY, the S&P500 ETF SPDR, GSPC the S&P500 index itself and the EUR/USD FX Exchange rate). Updates will be denied by time held in update log and will skip all data sets that are invald based on this. New assets are fully fetched from the their sources, meaning if a new asset (not in the logfile or warehouse) is entered, all data sets in its respective 'profile' are requested by the Spyder.

The Application at RunTime

Step 1 - Load and fetch data

When the application loads, any preset data will be fetched first (as shown before), then the UI will load and the assets are seen added to the table (local cache) for selection, sorting, filtering, etc... which all relagtes directly to the visualizations and what data they process and when. The table holds all available assets loaded locally (Not in the entire warehouse, unless all are loaeded) and those that are highlighted green are selected (red is not). Selected assets are applied to plotting calls and data sorting operations as well. The tables are designed in such a way that the current selected assets and table state (including the asset orders) are maiantained at ALL TIMES. This was a complex issue to tackle but the solution, which can been seen in the finapp.py main application file, is fairly straightfoward.

During fetching, the console will report the flow of operations to you, clearly and succintly (for the majority of cases). One thing to notice is the RequestEngine running the multi-threaded requests. This ProgBar class is custom made to work within classes and multi-threaded operations calls to accurately report the progress to the user. The multi-threading operation makes requests by trying from a self-maintained list of free proixes (filtered to extract the most secure - see IPProxy.py) and so every request is 'masking' the users endpoint, so even if an IP address is flagged and blocked, the user is unharmed, the application drops the now 'dead' proxy and uses other ones from the vast cache of possible nodes. This structure is absolutely necessary, else this structure cannot be maintained in any real capacity for long periods of time.

After the fetching is complete (where fetching loads stored data and fetch + update loads and tries to update the data sets in memory), the data is appended, set andOr archived on the backend, while the data is also locally sourced to the application.

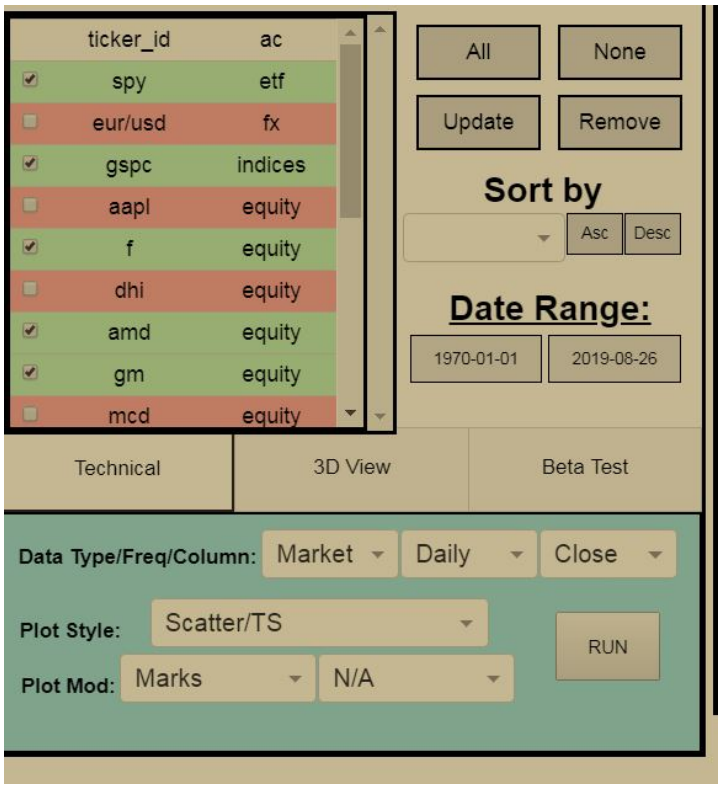

```
> [RUN] FinAppWorkerMain.fetchData_inapp(): ...
   > equity: 7 valid requests found: ['aapl', 'f', 'dhi', 'amd', 'gm', 'mcd', 'wmt']
   > etf: 4 valid requests found: ['xlb', 'xlf', 'xly', 'xli']
   > fx: 3 valid requests found: ['gbp/usd', 'jpy/usd', 'btc/usd']
   > indices: 2 valid requests found: ['dji', 'vix']
> Requested assets are valid - moving on to processing...
```

```
-----
> Feeding Spyder (Standby)...
-----



-----| Progress Bar --> Iter: 86, Scale: 1/43 |-----
[          25          50          75     90    100]
[----------25----------50----------75-----90--- 100] (RT: 151.17)
> Attempting to drop dead proxies from cache...
    > List to drop contains 565 proxies | 529 are unique... | Dropping 34... | OK
```

Step 2 - Technical Analysis Toolsets

After the user is satisfied with the ticker universe, they can then peruse the tools available to them for analysis. Please note, however, that at this time the 'sorting' options and loading from saved files (load option at thge top of the application where the fatch submit buttons are) are under development and will not do anything in runtime. AS stated before, the data controls the data in the plotting tools, so clicking on the boxes per row will eith selected or deselect the asset. The ALL or NONE buttons will do the same, for their respective titles that is. UPDATE and REMOVE will check for upodates at button submit or, if removes is presses, will delete the assets from the table AND the session cache data. In this case, if you remnove and want to retrieve the same data again, you can simply fetch it without updates from the warehouse storage. Please also note: certain actions update drop-down selection option in real time, which is stable but this is still a beta version so there may be some lag. This is normal. [NOTE: Beta testing is done in the 3D testing area – dedicated Beta analsysis will be ready in a few days].

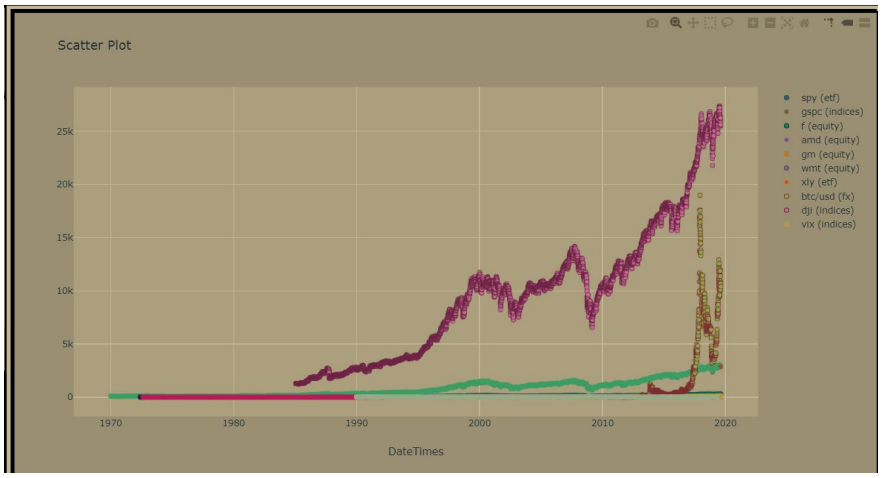
In general, all data operations will include the setting of the data type (Market data as it is, or some change applied such as periodic returns or growth from a common base), the frequency and the selected metric from the data tables (Close, Volume, etc...). Plot style allows the user to change the analysis type and those will be covered in order. The Plot Mod Drop-downs update in real time to the Plot Style selected, so for instance: Marks implies the data is represented as some of dot, dash or symbols (there are almost 100 to choose from in editing) and the alternative is a line. If NA is shown, the drop-down menu is not-needed (at this time). I have selected some assets on the table (now highlighted green) and will show some examples of analysis across the current tool set:



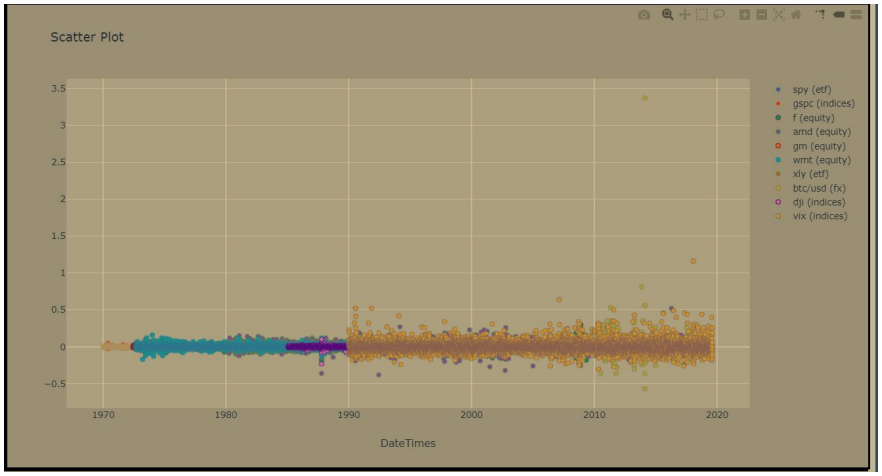
I. Scatter Plots and Time Series:

The scatter plots and time series plots are staple analysis tools for tracking and studying asset behavior for patterns and trends. In the next version, technical indicators will be available for table editing (please see TechnicalAnalysisToolbox.py for current working examples), however, one can still ascertain value from the data held now. If we look at this huge set of price data we see non-normalized data sets [1], which is useless so we can normalize it with returns [2] and then analyze the return structures to identify historical volatility. If we focus on the pre-post 2008 crisis [3], we see a large distortion casued by the Vix, which makes perfect sense so we remove that [4] and can now see a clearer picture of, including granular points within, the returns affetced by 2008. We can actually see an almost bimodal shape forming between Ford (F) and the BTC/USD rate, where Ford was buffetted heavily by the Recession and shortly thereater the markets started to see a larger interest in alt-currencies. This was already known, but this visualization tool made that distinction much clearer, and with the variability of source code, this is a good first step in integration. (ALSO: All plots can be dragged, zoomed, selected and even exported, with more features in the 3D plots)

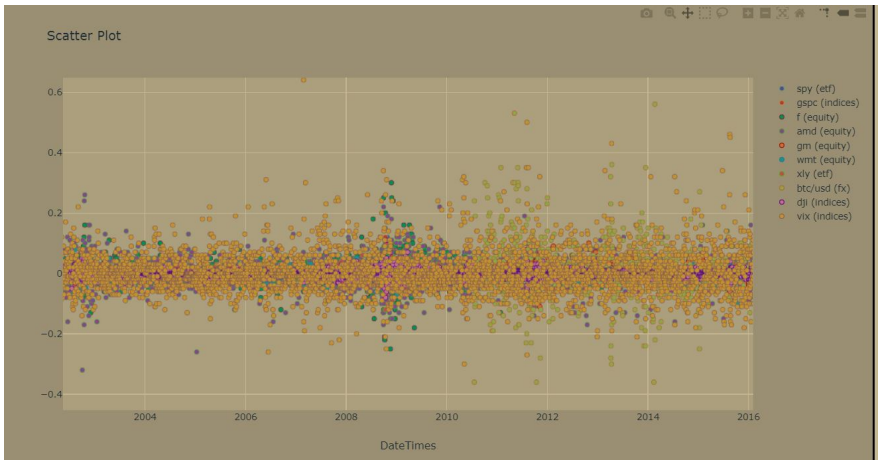
[1]



[2]



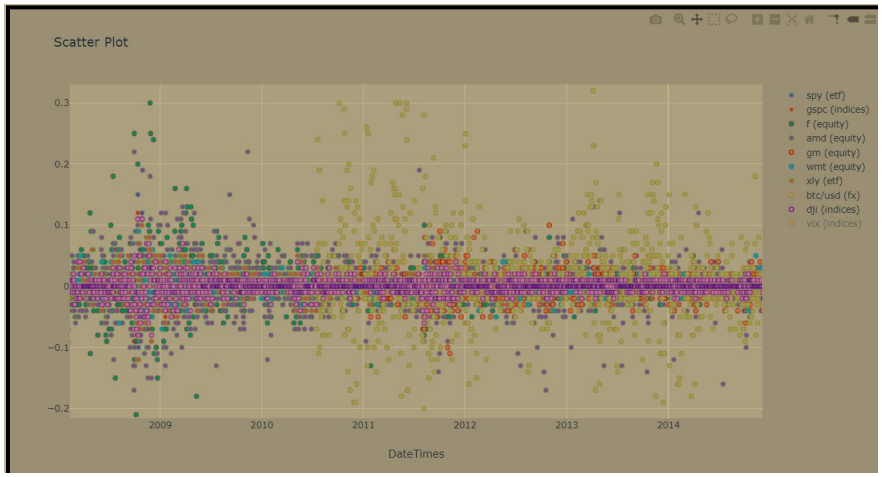
[3]


Scatter Plot
0.6
0.4
0.2
0
−0.2
−0.4
2004
2006
2008
2010
2012
2014
2016
DateTimes
spy (etf)
gspc (indices)
f (equity)
amd (equity)
gm (equity)
wmt (equity)
xly (etf)
btc/usd (fx)
dji (indices)
vix (indices)

[4]


2004
2006
2008
2010
2012
2014
2016
DateTimes
spy (etf)
gspc (indices)
f (equity)
amd (equity)
gm (equity)
wmt (equity)
xly (etf)
btc/usd (fx)
dji (indices)
vix (indices)

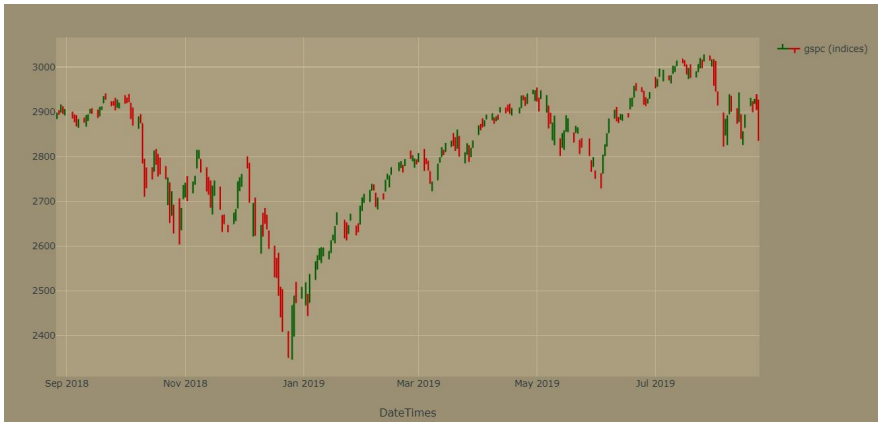
[5]


Scatter Plot
0.3
0.2
0.1
0
−0.1
−0.2
2009
2010
2011
2012
2013
2014
DateTimes
spy (etf)
gspc (indices)
f (equity)
amd (equity)
gm (equity)
wmt (equity)
xly (etf)
btc/usd (fx)
dji (indices)
vix (indices)

II. OHLC and Candle Plots (Day Trading):

We can use the OHLC and candle stock plots to view data changes and behaviors within each day, to track or model trends for day, swing trading, scalping, etc. [1] Shows the standard green or red OHLC and [2] is the same for candle plots for the S&P 500 Index (GSPC). This is a clear picture, and with the ability to build on this application, any indicators or graphing or tracing tools can be created to tailor the analysis and make custom indicators as well. In a basic example for the time, the colors can changed for the increasing and decreasing lines, and varying them can be useful. [3] The black for up and white for down serves a good purpose, isolated a certain trend direction to map for reversals or patterns. This shows the bias for up but can easily be written to be interchangable. In the event that one wants to alter their perception or 'manipulate' the data object, a simple change in shade can reveal cleare signals. One tool to be built will be one that can be assigned ti custom events (price movements, etc. . .) to ID specfific patters. In [3], we can see, much more clearly, the nodes we could use to map down trends leading into Nov 2018 and, if flipped, easily see the drops jan and Dec as well.

[1]



[2]

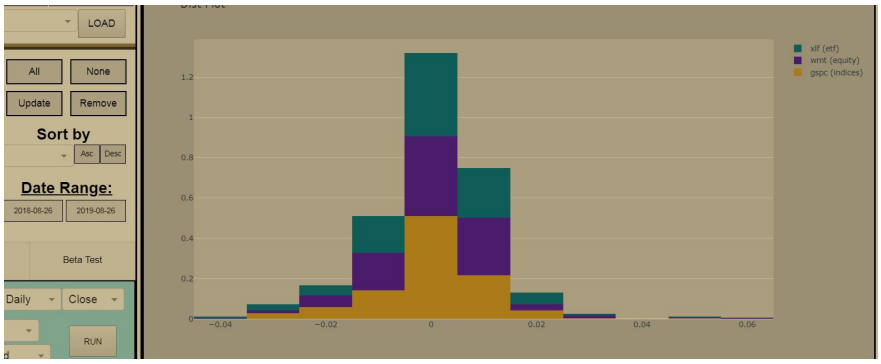
[3]


Candle Plot
gspc (indices
2900
2800
2700
2600
2500
2400
Sep 2018
Oct 2018
Nov 2018
Dec 2018
Jan 2019
DateTimes

III. Distributions

These are straightforward and the current stacking or overlapping features work well to place the distributions with one another. [1] We can see an interesting stacking of the return distributions for the last year. This is covering the S&P500 index, WMT (Walmart) and the Financial Sector SPDR ETF (XLF). We see the financnial sector, whose behvior we can extract to a reasonable degree from the proxy ETF, has the talles distribution of the 3, and WMT is second. It makes sense to the S&P500 index as the lower returning of the 3 but it is also more stable and we can see less action in the tails (less likelyhood of larger scale moves – kurtosis greater than 3).

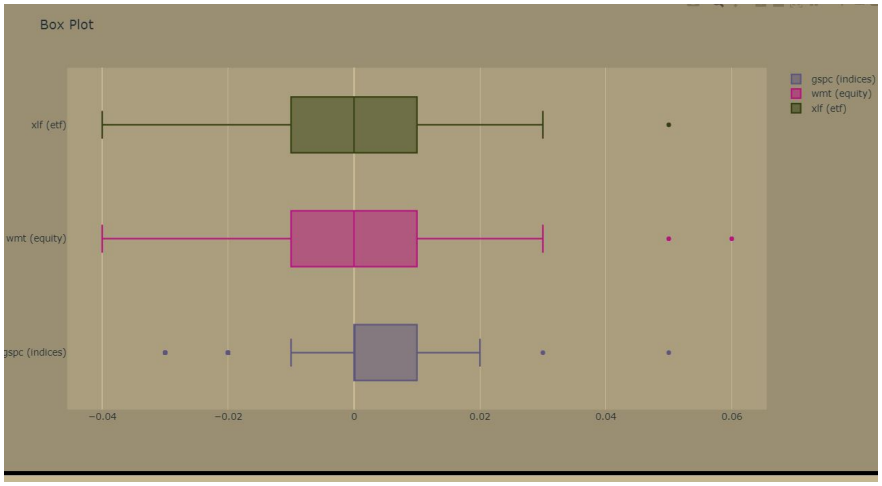
[1]



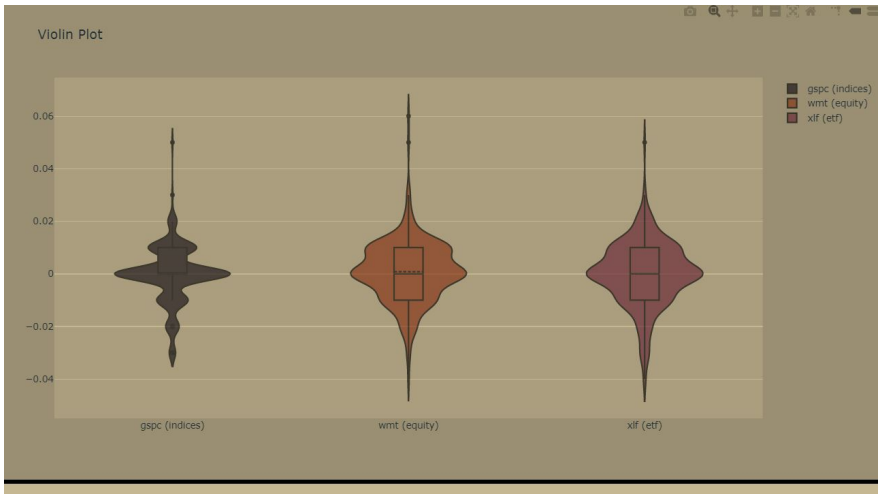
IV. Stat Summaries (Box and Violin Plots)

[1] and [1] Show the same summary statistics (quantile, median, etc) as well as outliers which are points beyond the interquartile range (as a rule of thumb – more levels will be added for outlier analysis in the backend for this). The interesting aspect here is how they are shown. The box plots [1] are cut and dry, as well as reliable byt he violin [2] plots include a distribution component to thge surrouding noise (points in the data set). I cannot get a screenshot of the statistic on-hover but they are listed (as well as can be set for all plots) and these representations tell us alot more information. For instance, the box plots show basic tiers of stats but the violin shows us the variability as well as the variance at any given level (relatively). From the box plots, the GSPC looks like a stable distribution with market-related outliers, but the violin shows us the variability within the density, while not as consitently wide, does show some volatility that was unexpected. The shape is almost tri-modal, with a clearn bell for some semblence of normality but still some interesting behavior to analyze further.

[1]



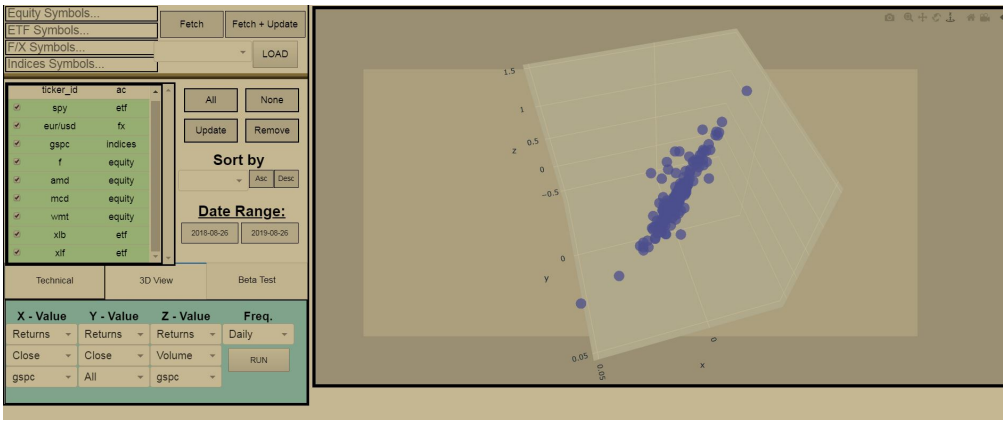
[2]



Step 3 - 3D Model testing

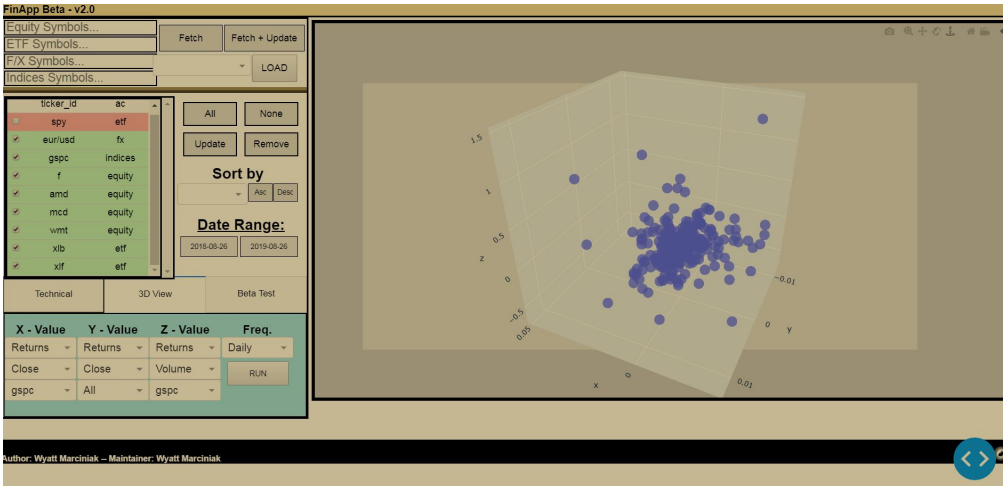
This is being integrated now, and there are issues with very large data sets and rotation speed but it is stable and interesting to use, as well as holds immense promise in later versions. I will use it so show relationships among the data but also to conduct some loose Beta testing before the final (and final tab we will not cover) is finished. I have designed the tool to let the user select seperate data for x,y and z as expected however, the user can choose to select 1 to 3 target asset(s) to sit on one, all or none of the axis and still compare to all of the assets currently selected or sompare metrics of the whole to the whole. The potential is huge and this is just a demo of what the implmentation will eventually be.

[1]



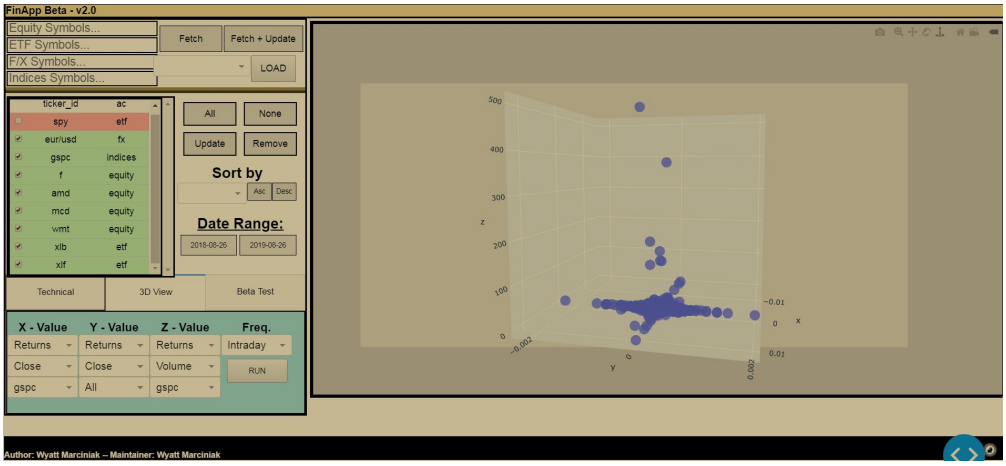
In [1] I make the error of comparing all the data but I leave the S&P500 and the SPDR tracking it in the data set at the same time so the results are erroneous. In [2] I correct this, and we can easily see a more normal relationship forming.

[2]



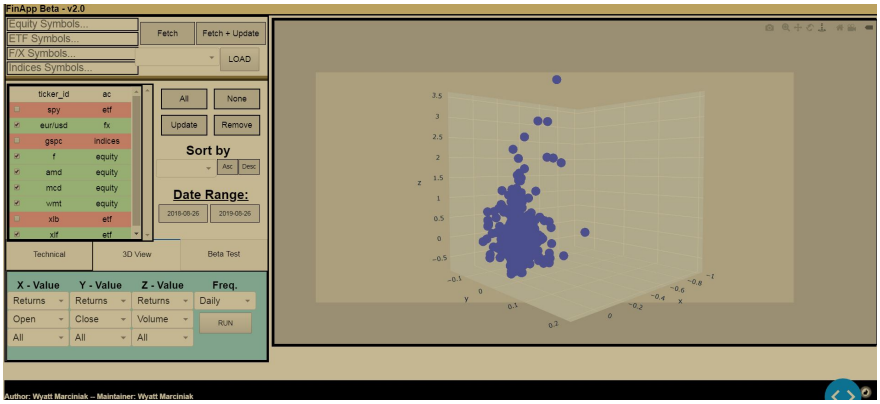
The x axis is GSPC daily returns and the Y is the daily returns aggregated from all other selected assest (not including itself). The z axis is the volume of the GSPC, which leads to some expected outliers and results but begs the question of what can be analyzed when other, non-price data is used to model this 3D representation.

[3]



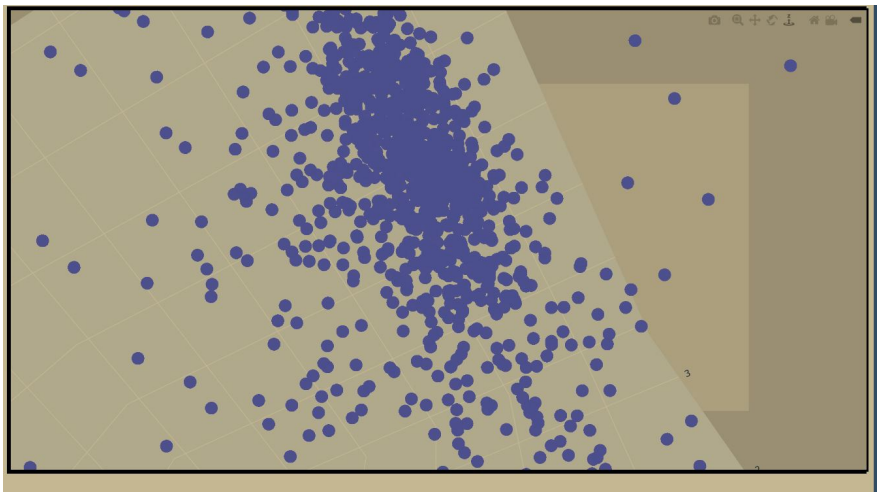
In [3] I use intraday data. Now for this and for all data, duplicates and invlaid rows are already filtered, so we may be seeing effects from data deterioration (too many elems removed from original set so it loses its behaviors) and one or 2 assets may have less or not yet archived data from bad requests. This needs toi be vetted, however the majority oif the assets are whole and the intraday data sets are per min, with no assets having more than 2 months stored at this stage of growth so to a degree, this weird relationship resembles some actual market traits. Intraday data, when modelled by ARMIMA(p,d,q) processes shows heavy amounts of pure randomness, mostly dirven by HFT traders and traders conducting pumpNdump, scalping or other aggresive strategies aimed at statistically trading smaller moves or over smaller time-steps. Therefore, the relationshop between these 2 assets based on (z) the S&P500 trading volume is almost worthless but the principles of larger tail movements still appllies, as seen in the resulting outliers. Below [4] I implement a test of all selected data (seen in column) to relate changes in Open prices, Close prices and volume. What is observed is more easily seen when rotating in real time (color schemes and unique data color tags are in devlopment testing), but the change un closing prices is much more impactful on the data than the changin in opening prices, which is logical, and volume scales the relationship up towards the upper-limit outliers, but closing price changes influences the general drift more often than note.

[4]



Finally, in [5], I took this screenshot after trying to sit in the main cluster. This is difficult and with more data comes heavier lag, which could cause the debugger in Dash to restart so if that happens its ok. This is, however, an exciting challenge to implmement clustering models on market data to identify relationships that may lie beyond the normal scope or may even help find new ones forming. If we can analyze the clusters, with proper tags and colors to actually see the seperations, we could create models to gain better understadings of asset class interactions, especially because was can interact with it as an object and not as lists of raw data.

[5]



Conclusion (Brief)

In conclusion, this application has taken me 2 years to learn everything needed to have it become stable, self-maintaining and realistically applicable to the markets. Now that this beta is archived, there is no limit to the scale and depth that can be achieved. One could even create their own vidualizations and nest them in the application framework, so custom algorithms can be designed and tracked by the UI as well as terminal output. In all I am very excited by the future for the application as well as the progress made to this point. One can use this tool, build onto it and scale it to their needs and above all else, its 100% free to use. The original research goals have been met and I am already working on v2.5.0. Data analytics and visualization go hand in hand, and to be able to build the platform to control them in means having access to a very powerful edge in financial markets.