Alternative Data Winter Project Documentation

Introduction

The idea for this project is to create a database + data pipeline for alternative data sources. As we currently don't have much non-traditional data based research, having viable sources of alternative data allows for interesting and new quantitative finance research. One of the main issues with current alternative data sources is that it exists as a large static dataset. While this allows us to research it heavily and build a model out of it, the lack of live data poses a struggle for us to deploy the model in a live setting.

Project Description

This project consists of several steps: Collecting the data (api vs scraping), storing it in a database, and making it accessible to others. Right now, the first two are the most important as that determines if the project is viable or not. Since making the data accessible to others isn't the most important thing right now, we can just work with a local MySql database to speed up data storing for the time being.

Data Scripts

The process of data collection isn't necessarily too complicated, just a lot of work of finding data sources and writing a script or using an API to collect live volumes of data. The idea is to have a central hub script that imports the key methods from different scripts, each being for a different alternative data source. Since the methods all have differing (time.sleep()) values as the data updates at different speeds and necessity, we use multithreading to execute all the methods at the correct times for each.

Sunset/Sunrise Data

This is definitely the least useful data source, although it was the easiest to implement and allowed for a great blueprint for future scripts. Given different latitude and longitude values, the API returns the time of the sunset/sunrise at that location for this day. The API also takes in a field representing the date of the data requested, meaning we can set the datetime as 2010, and keep looping until the present day, to create a database of the sunrise/sunset times for the last decade.

Again, not sure about the value of this data but it's interesting to analyze as perhaps there could be strategies that companies more south than others do better as they spend more time in the sun year round? Or do northern retail companies get more sales during the summer vs southern ones? Some things that can be looked at.

Since the sunrise/sunset information doesn't change until the end of the day, we only need to execute the script to collect the live data once per day.

Flight Data

As a whole, this dataset allows for live tracking of planes through its transponder. During initial tests, I found that it had a consistent amount of about 7000 planes it was tracking. Granted, this was between 1 and 4 am so it definitely would be higher during the day. The point being that

storing high frequency amounts of data of this volume wouldn't be viable at least in the short term.

In the image below, you can see all the possible data points that can be obtained from each plane. There are a lot of random pieces of information that don't seem useful as of this current moment, but could be implemented as we progress further. Answering the data load issue, my initial thought was to create a dictionary and store the origin data for each country, i.e. how many flights in the air have departed from that country. This could be useful for analyzing macro trends in certain countries? For instance, if a sudden drop in flights from a country could mean less tourists or people visiting, meaning their stock market could suffer.

icao24 - ICAO24 address of the transmitter in hex string representation.
callsign - callsign of the vehicle. Can be None if no callsign has been received.
origin_country - inferred through the ICAO24 address
time_position - seconds since epoch of last position report. Can be None if there was no position report received by OpenSky within 15s before.
last_contact - seconds since epoch of last received message from this transponder
longitude - in ellipsoidal coordinates (WGS-84) and degrees. Can be None
latitude - in ellipsoidal coordinates (WGS-84) and degrees. Can be None
geo_altitude - geometric altitude in meters. Can be None
on_ground - true if aircraft is on ground (sends ADS-B surface position reports).
velocity - over ground in m/s. Can be None if information not present
true_track - in decimal degrees (0 is north). Can be None if information not present.
vertical_rate - in m/s, incline is positive, decline negative. Can be None if information not present.
sensors - serial numbers of sensors which received messages from the vehicle within the validity period of this state vector. Can be None if no filtering for sensor has been requested.
baro_altitude - barometric altitude in meters. Can be None
squawk - transponder code aka Squawk. Can be None
spi - special purpose indicator
position_source - origin of this state's position: 0 = ADS-B, 1 = ASTERIX, 2 = MLAT, 3 = FLARM

This data also updates at the fastest every 10 seconds. We don't need information that fast regarding flights, so I set a baseline update speed of every minute, which can be increased or decreased depending on necessity.

YFinance

Here's the collection of data sources that we obtained from yahoo finance

Currency Data

Through the yfinance, we're able to get the current value of dozens of major currencies pairs. This value is currently being stored in 30 second intervals, which can be changed based on need. During the testing stages, we're just storing EUR/USD, JPY/USD, GBP/USD, and AUD/USD, but more can be added efficiently.

Futures Data

Using yfinance, we can collect the live price of dozens of commodities. This data is currently being stored every minute, but is also 10 minutes behind live prices. Using this data, we can analyze trends in different large economic features (companies related to those commodities, or countries that rely heavily on the commodity), and trade based on that (currency for those countries, or stocks for the companies). We can also group the commodities into 3 major groups (agriculture, energy, metals) and analyze trends in that

Treasury Bond Data

Using yfinance, we can collect the price of several US treasury bond rates. This data is being collected every 30 seconds, and is roughly 15 minutes behind live prices.

Weather Data

This is the data source that I'm most interested to work with right now, just due to the size of the data source. The data provides a ton of information regarding all the weather in a city, and could be useful for data analysis. Using the documentation page, we can see all the parameters that are given. Again, like the source above, we can use this to assess performance of companies in different locations, or overall macro trends.

An example of this could be to analyze the weather in different major cities that a large retailer has buildings. Should there be an above average amount of cold weather, rain, snow, etc., we can analyze the impact this would have on consumer shopping behaviors, and perhaps that they choose online options (like amazon). This could reduce the sales of shops in those areas, and thus can be used in a model to assess the odds that a company beats their quarterly revenue targets, and purchase that stock accordingly.

With the initial run-through of this source, I'm saving the weather type, information about the weather, visibility, rain, snow, wind direction, humidity, and a few other parameters. This API updates its information every hour, so the script is running at that frequency.

Note: They provide a list of 20k cities that have specific city id codes, should we want to verify that a certain city is actually accurate and it's not choosing a different one with the same name.

Finnhub

Social **Sentiment**

Using the Finnhub API, we can get the live social sentiment of stocks on reddit and twitter. This can be used to track the social perception of companies, which is important when looking at stock performance. For instance, Tesla only became overvalued when the perception of the company was that it would take over the car industry, which is a very positive social sentiment.

Visa **Applications**

Using the Finnhub API, we can get the amount of visa applications a company puts in for H1-B and permanent employees. Using a sliding window (we can adjust the size of the window), we can have a rolling value of visa applications, and use that to see how much a company is growing

Senate **Lobbying**

Using the Finnhub API, we can get the amount of lobbying a company is participating in. Using a sliding window (we can adjust the size of the window), we can have a rolling value of senate lobbying costs.

Government Spending

Using the Finnhub API, we can get the amount of contracts and the size of the contract that companies are winning. Using this, we can see which companies are earning large amount of money through contracts, and invest accordingly

Future Data Sources that will be looked at (still building)

This is a collection of sources I've written down to look at, but haven't gotten around to do so.

Government Department APIs

The government provides a massive amount of datasets that are useful to analyze. There are some from most if not all of the federal government agencies. Based on the sheer amount of different data sources here, this is an interesting thing to look at.

Note: Since I don't think the data source has a live API, we can continuously extract the database and compare it with the one stored in our database, and the differing values represent the new data points inputted into the government's database, which we can use for live analysis

IMF Data

The IMF provides a lot of data regarding countries' macroeconomic situations, currencies, etc. This could be useful to look at, and then find out of analysis can be made from it (live information, etc)

Instagram Data

Using selenium or instagram's API to analyze like-based trends on different accounts to see if there is a noticeable trend regarding a company's popularity, etc.

Data Collection

All of the data sources that we are analyzing from this project contain live information. However, some also can have an input as a date and procure the information from that time in history. Based on that, we plan on writing a script to store as much historical data from those data sources as possible.

Fundamentally, there's going to be two main scripts ("HistoricalCenter" and "methodCenter"), and these are going to be the essential "hubs" for the scripts. The HistoricalCenter file will link all the historical data scraping scripts together, and run them all concurrently. The methodCenter file will link all the live data collecting scripts together, and run them all concurrently.

But regarding the place where the data is stored, at the moment it's being stored in a local MySql database set up on my computer. Should the project start moving onto the next stage, I'm going to plan on using AWS/Azure's free 1 year trial online database, and start storing data

there. But by using MySql right now, it's easy for me to access and store the data for initial tests and data visualization examples.

API

This is where it gets interesting, as this is useful for the future. Eventually I think the idea is to make an easy API that can distribute all the data from this database I'm building from a bunch of data sources. I'm not sure if this is the best course of action, but it's immediately what comes to mind when I think of ways for people in SIF to be able to easily access the data.

Future Plans

There are a couple things that come to mind when I think of my long term prospects of this project. The first being the development of the API, or an easily accessible way for people in SIF or other authorized users to be able to access the data, without direct permissions to delete or add stuff to the database. This API would connect with the database rather than the data scripts, meaning we wouldn't have to waste calls to the alternative data sources on API calls.

The second is to create a live site that shows the information being stored. For instance, have it automatically collect the data that is being newly stored into the database, and create visuals that can be easily understood by someone looking at the site. Just due to the sheet size of the data that's being stored and analyzed, having quick automatic visuals regarding the data makes it easy to understand what we're looking at.

Website

The website has two parts that need to be developed. The python backend and the react frontend.

Python

The python backend works to create the graphs of the necessary alternative data based on the input. It uses Flask to connect to the react frontend. The response url it gets from flask includes the values from the alternative source it needs, as well as any stock values it wants to include in the chart. It returns a matplotlib image that's being stored in the folder. This image is what is displayed in the frontend, and gets constantly updated every time the backend runs.

React

This is the front end display of the database. This will be the description of one page of it, as it’s roughly the same implementation for each source.

There is a chart, displaying the information present in this page in the database. Below it, there's a clickable list of data points to be added to the chart (for example, in the flight data, this clickable list would be the countries that the planes are flying from). When each point is clicked, it sends a request to the python backend that reconfigures the chart, and saves the image as the new chart. This new image is then displayed to the webpage, meaning that the chart updates live every time a new value is pressed.

To the side of that, there’s a textbox for stock tickers to be typed. As these are typed,