Quant Database System

1. Scope the Project and Gather Data

We are going to be obtaining fundamental
and market data on equities, indices, mutual funds, ETFS, commodity futures, cryptocurrencies,
fixed income, FX, macro and other miscellaneous data.

1.1. Data Sources:

1. Eodhistoricaldata
2. Oanda
3. SEC
4. FRED
5. OECD
6. DBnomics

1.2. End Use Cases:

We design and implement a useful system for
quantitative research of scale for trading.

2. Explore and Assess the Data

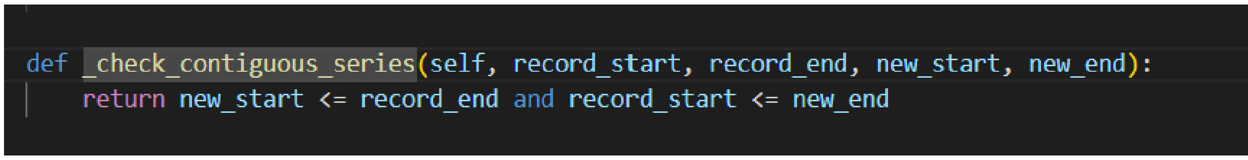
2.1. Data Quality Issues:

1. Receiving timestamps in different units and timezones
2. Missing Data for select time intervals
3. Duplicate Data for select time intervals
4. Misalignment in date and time stamps for different assets
5. Different code/ticker, same underlying (across assets/time)
6. Same code/ticker, different underlying (across assets/time)

2.2. Data Quality Issues Avoidance Procedure:

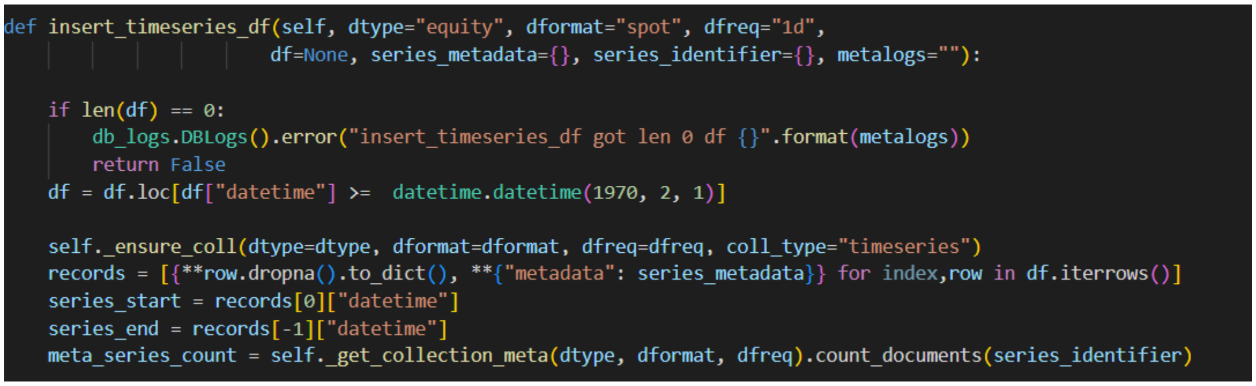
- Combining ISIN, ticker,exchange and source should have a unique representation in the database.
- Picking a right data model to avoid data integrity issues
- Selecting a time series collection(MongoDB offers it) to avoid heavy reads and writes
- Introducing metadata for faster read and write

- Introducing _check_contiguous_series to check whether time series data is contiguous or not, if not proper action is taken depending on different scenarios.



```
def _check_contiguous_series(self, record_start, record_end, new_start, new_end):
    return new_start <= record_end and record_start <= new_end
```

- Pymongo module uses Pyhon dattime object to meet MongoDB specification and convert between the two. There is a Unix epoch(Jan 1,1970) involved, and date before 1970 have negative epoch.Their driver has a few bugs when it comes to data filtering and does not give us the desired result when period_start is before 1970.To avoid this we only allow our database take in date times starting from Feb,1970.



```
def insert_timeseries_df(self, dtype="equity", dformat="spot", dfreq="1d",
                        df=None, series_metadata={}, series_identifier={}, metalogs=""):

    if len(df) == 0:
        db_logs.DBLogs().error("insert_timeseries_df got len 0 df {}".format(metalogs))
        return False
    df = df.loc[df["datetime"] >=  datetime.datetime(1970, 2, 1)]

    self._ensure_coll(dtype=dtype, dformat=dformat, dfreq=dfreq, coll_type="timeseries")
    records = [{**row.dropna().to_dict(), **{"metadata": series_metadata}} for index,row in df.iterrows()]
    series_start = records[0]["datetime"]
    series_end = records[-1]["datetime"]
    meta_series_count = self._get_collection_meta(dtype, dformat, dfreq).count_documents(series_identifier)
```

3.Define the Data Model

3.1. Conceptual Data Model:

- Time Series Data Model

```
{
    "datetime": ,
    "open": ,
    "high": ,
    "low":  ,
    "close": ,
    "adj_close": ,
    "volume": ,
    "metadata": {
        "isin": , "ticker": , "source": , "exchange"
    }
}
```

- Metadata Model

```
{
    "type": "ticker_series",
    "isin": ,
    "ticker": ,
    "source": ,
    "exchange": ,
    "time_start": ,
    "time_end": ,
    "last_updated":
```

- Data Dictionary
  -General:

code	Code of the company issues by SEC
name	Name of the company

exchange	Exchange the company is traded at
currencyCode	Denominated currency the company’s shares is traded in
CurrencyName	Name of the currency
CurrencySymbol	Symbol of the currency
sector	The sector in which the company is categorized into
Industry	The industry in which the company is categorized into
Description	A short description of what the company does
fullTimeEmployees	Number of full time employees

4.Run ETL to Model the Data

4.1. ETL Steps For Time Series Data:

1. Check if the data frame is not empty
2. Check if there is existing time series collection and sister meta collection
3. Check if there is related data in the collection by querying the sister meta collection
4. If there is no existing data, insert the records together with metadata appended to each record.Additionally, create an entry for it in the sister meta collection
5. If there is existing data for it, ensure that there is only a unique representation of it. If not,log a critical error
6. If there is unique representation for it, check the record log for the range of timestamps already existing in the database.
7. If the new records and existing records create a contiguous time series, then add the new, non-existent data to the head and tail of the existing

records. Update the time series span in the sister meta collection accordingly

8. If the new records and existing records do not create a contiguous time series, do not conduct
the insertion.

5. Complete Project Write Up

1. What's the goal? What queries will you want to run? How would Spark or Airflow be incorporated? Why did you choose the model you chose?

Data is expensive. Not everyone can afford a Bloomberg terminal. Virtually all of retail and a great proportion of professionals would not be able to do so. Even if you are a small prop shop with AUM in the millions, getting a BB terminal is likely a ballsy and stupid move. It will definitely eat into your performance. And unless you are a FI trader, or chat regularly with a PM, you are using a fraction of its capabilities but all of the 20000 thousand dollars. Goodbye money.
A cheaper alternative is Refinitiv Eikon. Great vendor, but perhaps still expensive for most.
A hundred thousand dollar portfolio cannot be chucking 2k at data. Our focus is on cheaper (and therefore almost inevitably worse) data but still complete enough for us to perform quantitative research.

We are going to demonstrate how to create a data library to pull necessary data
on markets. We are going to be integrating with multiple different APIs, some which are paid services and others which are developed in-house. We are going to be obtaining fundamental and market data on equities, indices, mutual funds, ETFS, commodity futures, cryptocurrencies, fixed income, FX, macro and other miscellaneous data.
Here is a sublist of queries we might want to perform:

1. Add a list of 500 stock tickers and their OHLCV to the database.
2. Append and/or update for a list of 500 stock tickers the OHLC for the last 300 records.
3. Get all stocks listed on the NASDAQ from 2000/01/01~2001/01/01.
4. Get all option chains related to TESLA in the past month.
5. Get all securities in the database with underlying assets as AAPL.
6. Get all stocks listed in country ISO code CHN with a market capitalization of less than 1B.
7. Update the last 50 GDP releases with a new data series.
8. Create a dataset of option chains where underlying is a firm with high D/E ratio and high market cap.
9. Delete datasets that have not been read or updated in the last 100 days

Airflow could be incorporated in our pipeline to take care of the automation of parts in which data need to be updated regularly (e.g earnings, p/e ratio, OHLC time series data in quarterly, yearly and daily basis)

2. Clearly state the rationale for the choice of tools and technologies for the project. Document the steps of the process.

There are many different approaches to incorporating data retrieval in our system. Instead of having monolithic code, our objective is to create a data library that can be used in arbitrary systems. It is a data service. We know we want data. We don't care how, where or what it is used for. We want to create a service library that is suitable regardless of whether we are obtaining the data for academic research, live trading or quant research.
What formats of data are we likely to encounter? The simplest and likely most frequent example is the time-series data, such as the prices of a stock listed on an exchange. Time series data can be organized in the form of panels (dataframes), which are multiple time series data sampled at the same frequencies with respect to a sampled subject. Other values include singular values (represented by integers, doubles, floats, strings and so on), dictionaries, tables and textual corpus.

We don't know 1- what data we expect to obtain in the future, and 2- in what format they will arrive in. Perhaps it is futile to design a database focusing on storing fundamental artifacts, because even the most prescient of foresight will not be able to plan this reasonably well. We have the advantage of using a flexible document-oriented database that does not enforce schema - and this plays well in our favor. It would be nice if we knew what data we will receive and plan our schema accordingly, but in this case we don't and that is okay because we can just do it on demand. We can perhaps then focus on a more tractable problem, which is to design schema for pricing information and incorporate fundamental data as unstructured documents.

We can create meaningful data schema by considering three levels of classification:

1. Data/Asset Types:
   equity, FX, commodity, rates, crypto, credit, volatility, economic, baskets and others.
2. Data Formats:
   options, futures, forwards, swaps, spot, macro, fundamentals and miscellaneous
3. Sampling Frequency:
   tick,1s,5s,30s,1m,5m,15m,1h,4h,1d,1w,1m,1q,1y, irregular

In general, we have two types of collections timeseries collections and regular collections. The timeseries collections have the optimizations (MongoDb) but we need to specify the timestamp. Additionally, it is good to specify metadata about what we intend to query on.
Let's consider the 1-day equity spot OHLC time series schema as an example, but the principle applies to other time series data.
In general, the metadata uniquely identifies a time series. What do we want to be able to identify our time series by? Some useful properties of equity listings are ticker code, CUSIP, ISIN, exchange, data source, country etc. Combining ISIN, ticker, exchange and source should have a unique representation in the database. For each time-series collection created in our database, we can maintain a meta collection containing details we need about the time series for faster access.

For unstructured documents, we do not know what format or schema we want the data to be stored in. We can hence define a more general function intended to store general schema that sits nicely beside our time series collections. We do not need a meta document for this, and accordingly the upsertion and reading of the documents are made easier.
In addition to the usual variables, we take in a variable expire db which determines whether the data that we read is too old - if the data read has not been updated for more than expire db hours, we tell the caller that the data has expired. We can increase the expiration hours based on how expensive getting a new update is - for instance different API calls to a third-party can cost the application differently in terms of time, server load and financial costs etc. Additionally, data that is only sampled annually do not need to be replaced every few days. Data corresponding to slow samplers can be updated infrequently.

3. Include a description of how you would approach the problem differently under the following scenarios:

- If the data was increased by 100x.
- If the pipelines were run on a daily basis by 7am.
- If the database needed to be accessed by 100+ people.

The following is the general framework and techniques we use to overcome the bottlenecks which exist within our system due to database,network and programming language(python) in order to scale the design to meet a potential surge in demand in terms of data or users.(codes provided)

We can also use a data management workflow system such as Airflow to update our database on a daily basis such as daily OHLC prices which exist in our asset universe.

AsyncIO	Sessioning	Batching	Multi-Core	Timing(min)
N	N	N	N	14.9
Y	N	N	N	9.5
Y	Y	Y	N	9
Y	Y	Y	Y	3.

Read & write of 300 tickers using different techniques.

Multi-Threading	Server-Side Optimization	Timing(min)

N	N	5
Y	N	1.9
Y	Y	0.10

Read & write of 300 tickers using different techniques.