Data Engineering Capstone Project Plan

Name: Christopher de Wardt

Intended Data Source: **https://eodhd.com/financial-apis/macroeconomic-data-api**

Define the Objectives

1. Extract UK Government daily gilt prices for some or all of the 1/3/5/10/30 yr maturities, dating as far back as reasonably possible. (Ideally as far back as 2019)
2. Make any transformations, where required, to sanitise; then load the data into a Postgres DB using a schema that specifies separate tables for each maturity class.
3. Create and periodically run an ETL process that captures only the new prices of these data sets as they are updated on a daily basis. Add this new data to the existing Postgres DB.
    a. Ensure the ingested data is clean and consistent – i.e., no duplicate entries, no missing or anomalous values.
4. Create and connect a Streamlit app to the Postgres DB holding the ingested data - using my personal key credentials and make an appropriate chart to visualise some selection of the ingested data.
    a. Incorporate controls that allow the user to select:
        i. Time-frame
        ii. Bond maturity class(es) – using toggle switches to allow multiple class selections to be displayed simultaneously
    b. Add a control to view/hide a dataframe of the current selection

Timeline & Goals

By End of Saturday 29th of June → Data collection & database updates completed

By End of Sunday 30th of June → Data Visualisation Completed

By End of Monday 1st of July → Diagram and presentation completed

By End of Tuesday 2nd of July → Presentation rehearsed twice, and any basic tweaks/improvements implemented to my solution

By End of Wednesday 3rd of July → Final presentation rehearsal and beginning to explore stretch goals

Technical Plan

1. **eodhd.com/financial-apis/macroeconomic-data-api** is a JSON data feed, that can be requested openly. Using a Colab notebook for these initial steps:
    a. Call the API to extract daily prices of each different maturity, going as far back as 2019
    b. Load each into separate Pandas dataframes, using `pd.read_json()`
    c. Check for basic data cleanliness and consistency using these dataframes, making any transformations where necessary
        i. NB there will be many days – typically weekends and bank holidays - where this data will definitely be missing
        ii. Assess whether any removal/replacement is necessary
    d. Write the code that allows this to be date-sorted and ingested into Pagila
2. From this Colab notebook code, create and load into a new Github repo three .py files and one .env file:
    a. *.env* – ".env" file containing my personal Pagila credentials and the API key
    b. *etl_functions.py* – containing the function definitions that facilitate the required ETL functionality
    c. *full_reload.py* – script that connects to database, creates new tables (replace existing tables if they're present), and performs the whole ETL process up to latest available prices
        i. I won't be able to directly run this script but it might be useful during development
        ii. I could hypothetically run this as part of any code that intermittently checks for data integrity/consistency in my Pagila DB tables
    d. *update.py* – script that performs updates on the tables, only running if the latest available API data is newer than what has already been processed
3. *update.py* script will run on a CRON job, hosted on the Digital Futures server, collecting the data every 120 minutes, performing updates if necessary:
    a. This is running much more often than strictly necessary in order to provide some level of fault tolerance in the event that some aspect of the ETL process experiences temporary failure
4. Create a Streamlit app that provides a dashboard for visualising the ingested price data on Pagila:
    a. Find a specialised Streamlit component (e.g. Plotly, Bokeh) to create the chart and interface – or, use one of the usual libraries that facilitate this
    b. Provide controls for controlling the particular data of interest shown in the visualisation
    c. Make an "info card" to display/explain what instrument(s) are being inspected
    d. Make a key explaining the various attributes (e.g. daily high/low, open/close)
5. Once this is functioning, upload the various components to Digital Futures AWS infrastructure and make any tests where possible.

Potential Risks

1. Potential duplication-of-data due to multiple extracts/ingests
2. Being blindsided by limits of the API wrt to call limits – potential free vs paid account issues
3. I need to make sure I understand what data I am actually handling from a domain knowledge perspective – particularly with respect to date of maturity vs tenure
4. Making sure the extracted maturities/tenures are not mixed-up after extraction (and are clearly distinguishable in the visualisation if viewed simultaneously)
5. Is whatever is displayed on the Streamlit app going to be meaningful to someone not particularly familiar with bonds – ie carefully considering/researching my approach to visualisation so I don't lose time doing something that isn't going to work
6. The freedom to choose a long timeframe may lead to congestion on the chart if I use a candlestick visualisation or similar
7. This suggested step from tech plan section has me slightly confused → *"A Streamlit database, hosted on Streamlit community cloud, will read the SQL database and display the trend information"*
    a. *Not sure if this might be necessary for some reason, ie why Pagila on AWS and pandas within streamlit app might not suffice*
8. The low frequency with which my API updates its data (ie daily) may present difficulties during testing, particularly with respect to testing update.py

Stretch goals and next steps

1. Collect the stream of *specifically* UK political news stories from available news article APIs. Aim to collect across the media sources available (as opposed to just a few media firms that might have a sizable political bias)
    a. Run a sentiment analysis to assign a rating of favourability to each article from those collected
    b. Would need to make some kind of single score to aggregate every article's sentiment rating from each given day – can start with simple average, but ideally might want to identify political leaning of article providers and use a metric that balances the volume of articles from across the typical L/R spectrum
2. Make timeline on Streamlit showing the daily prices (/price movements) annotate with a selection of news articles that showed strong positive/negative sentiment towards/against Labour/Tories; allowing the user a good deal of granularity and flexibility in the timeframe they would select on the visualisation – ideally with intuitive tactile controls such as sliders.
    a. Can provide an interface feature to view the full list of article metadata.
3. A further stretch goal would be to run a time series analysis on the sequence of prices and sentiment analysis ratings to try and identify correlation in the two time-series.
    a. A final much more demanding stretch goal of interest might be an attempt to develop a model that runs a natural language analysis of the articles but attempts to train against the bond price fluctuation as its loss function.