Assessment Task: Crypto Historical Data Retrieval

Submission by: Jenil Padshala | jenil_padshala@srmap.edu.in

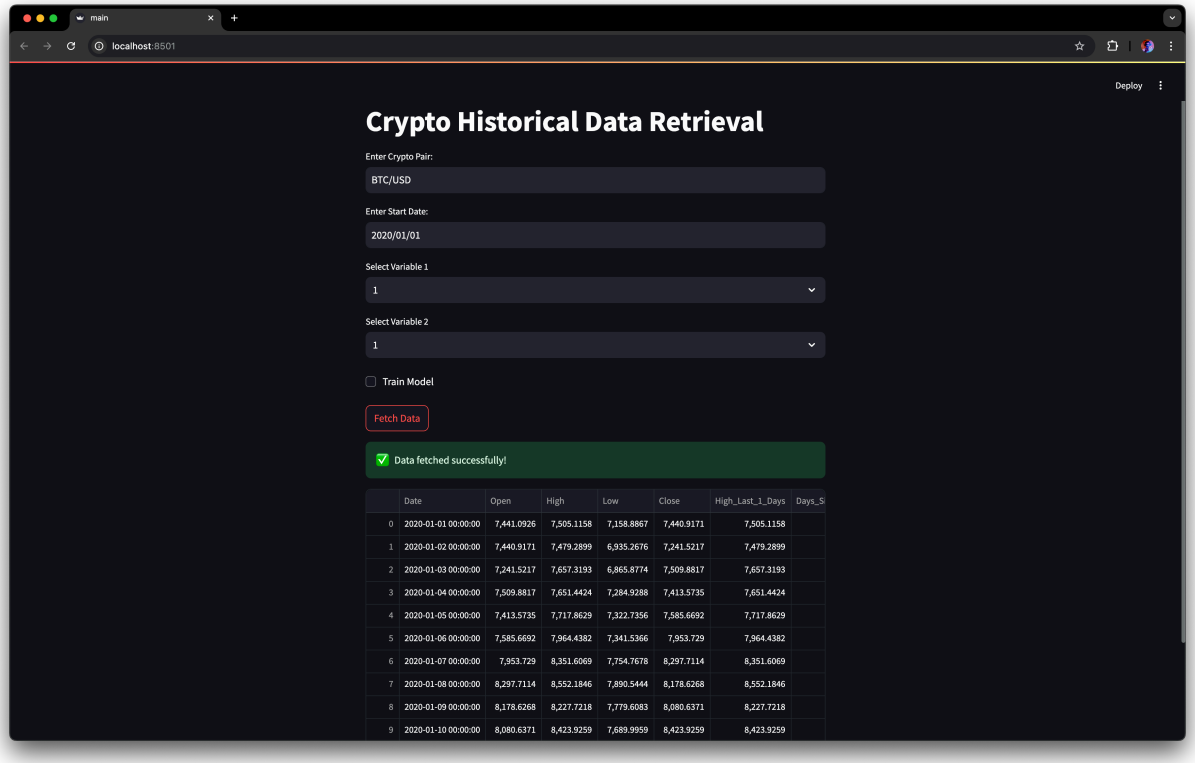

Fig: A Streamlit Dashboard to demonstrate the project

Task 1: Identify a Reliable API for Crypto Data Retrieval

API selected: Cryptocompare.com Historical OHLCV+ (link)

Number of crypto pairs supported: 260,000+ crypto-pairs and pricing data of 5,700+ coins (link)

Available timeframes: Days, hours, minutes (For this assessment, I have utilized daily timeframe for data retrieval)

Data availability range: Varies based on crypto pair. For example, earliest possible record for BTC-USD is from 2010-07-15 18:00:00.

Task 2: Retrieve Historical Data

Created a function **fetch_crypto_data**.

Approach:

1. **Environment configuration:**
   Sensitive information such as API Key and API URL have been stored in an *.env* file loaded via *dotenv*, ensuring secure handling of credentials.
2. **API Data Retrieval:**
   Each request retrieves up to 5000 data points (maximum limit as mentioned in API documentation). The data retrieval loop continues until the desired date range, from the current date back to the specified *start_date*, is covered. This iterative approach minimizes the number of API calls while efficiently retrieving the full dataset.
3. **Data Parsing and Concatenation:**
   After each API call, the response JSON is parsed and converted to a DataFrame *(batch_data)* using *utils.parse_response_data,* which is then concatenated with *all_data*, building a comprehensive dataset. The timestamp variable *(to_ts)* is updated to the earliest date in *batch_data* to continue fetching older data in the subsequent iterations.
4. **Throttling and Rate Management:**
   A one-second delay *(time.sleep(1))* is added between requests to reduce the risk of hitting rate limits.
5. **Final Date Filtering:**
   After data collection, the entire dataset is filtered to include only records starting from the specified *start_date*.

Notable Challenge:

To avoid infinite loop during retrieval, *to_ts* timestamp needed to be updated after every API request to the minimum date in the retrieved data batch, till *to_ts* was greater than the *start_timestamp*.

Task 3: Calculate Highest, Lowest, and Percentage Difference Metrics

Calculated and added Columns mentioned in the assessment.

Notable Challenge:
Calculating 'Days Since High' and 'Days Since Low'.

To tackle 'Days Since High', I devised the following approach:

1. Define a look-back period, starting from a specified number of days back from the current date up to the current date.
2. Identify the highest price in the window.
3. Determine the most recent date of that high price.
4. Subtract this date from the current date to determine the number of days since the last high.
5. If no high price exists within the window, set the value as 0.

A similar approach was devised for calculating 'Days Since Low'.

Task 4: Machine Learning Model

Defined features and target variable columns and extracted them from the retrieved dataset.

Split the data into train and test dataset using *train_test_split* form *sklearn*.

Utilised an *XGBoost* regressor to predict the target variables.

Evalution metrics calculated: MAE (Mean Absolute Error), MSE (Mean Squared Error) and R^2 error.

Notable Challenge:
The assessment required a multi-output target value. Therefore, I wrapped the *XGBoost* regressor in *MultiOutputRegressor* from *sklearn* to handle the multi-output target.

Note:

1. For easy demonstration of the assessment, I have created a steamlit app in main.py.
2. Each function has been properly documented with docstrings and comments.
3. The helper functions have been moved into utils.py. This will ensure ease in writing tests for the project in the future.

*****