Breakout AI assessment Documentation

Task 1: Identifying reliable API for Crypto Data Retrieval

API chosen: **CryptoCompare**

1. Number of crypto pairs supported: **357,139**.
2. Available timeframes: Supports **daily**, **hourly** and **minutely** historical data.
3. Data availability range: Data available for the **most recent day, hour and minute**.

Additional benefits of choosing cryptocompare API:-

- **Extensive documentation** available which eases the integration process.
- Allows upto **1,00,000** free API calls per month.
- Supports wide range of cryptocurrency data including all **top-traded cryptocurrency pairs** and is more crypto focused compared to other available APIs.

Task 2: Retrieve historical data

- Developed a function named get crypto data that accepts two inputs i.e. crypto_pair(eg. BTC/USD) and start_data(in YYYY-MM-DD) format.

- Defined API endpoint: "https://min-api.cryptocompare.com/data/v2/histoday" and its parameters namely fysm, tysm, limit, toTs

1. Fysm: This parameter specifies the **from symbol**, which is the cryptocurrency you want historical data for. It should be the symbol of the cryptocurrency

2. Tysm: This parameter specifies the **to symbol**, which is the currency you want the historical prices converted to. It can be any supported currency symbol

3. Limit:This parameter sets the **maximum number of data points** you want to retrieve. The CryptoCompare API for historical data limits the maximum value to 2000, so I have set it to the maximum allowed.

4. toTs: This parameter specifies the **ending timestamp** for the historical data you want to retrieve. It should be a Unix timestamp in milliseconds (e.g., the number of milliseconds since January 1st, 1970 UTC).

Then fetched the required data using the requests method from the mentioned url and obtained the required parameters namely **[‘Data’, ‘Open’, ‘High’, ‘Low’, ‘Close’]** and this fetched data is stored in the form of pandas DataFrame and also converted to excel.

Task 3: Calculate Highest, Lowest and Percentage Difference Metrics

Created a function calculate_metrics() which accepts 3 inputs namely data, variable1, variable2

- Data: The DataFrame containing historical data
- Variable1: Integer representing a look-back period (e.g., 7 days) for historical high and low metrics.
- Variable2: Integer representing a look-forward period (e.g., 5 days) for future high and low metrics.

1. High and low over past {variable1} days:
   To calculate the historical high price used pandas rolling function with a window size of 'variable1' days. It calculates the max value of the prices in the 'high' column for each day.
   ```
   data[f'High_Last_{variable1}_Days'] = data['High'].rolling(window=variable1, min_periods=1).max()
   ```
   Similarly for calculating the historical low prices for each day utilized the same rolling function but this time to used the min() function to calculate the lowest prices over the last 'variable1' days for each day from the 'Low' column.
   ```
   data[f'Low_Last_{variable1}_Days'] = data['Low'].rolling(window=variable1, min_periods=1).min()
   ```

2. Days since historical high and low:
   To calculate the the Days since historical high and low columns, used a lambda function to calculate the number of days since the last occurrence of historical high/low for each day.
   The lambda function checks if the historical high/low value exists (not NaN) for the current row. If it does, it finds the index of the last occurrence of that value in the "Date" column before the current row. It then calculates the difference in days between the current date and the date of the last occurrence. If the historical high/low is not available (NaN), it sets the value to NaN.

3. Percentage Difference from Historical High and Low:
   This calculates the percentage difference by subtracting the historical high from the "Close" price, dividing by the historical high, and multiplying by 100.
   Similarly we can calculate Percentage Difference from Historical Low
   ```
   data[f'%_Diff_From_High_Last_{variable1}_Days'] = (data['Close'] - data[f'High_Last_{variable1}_Days']) / data[f'High_Last_{variable1}_Days'] * 100
   ```

4. High and Low Over Next {variable2} Days:

```
data[f'High_Next_{variable2}_Days'] =
data['High'].shift(-variable2).rolling(window=variable2,
min_periods=1).max():
```

- This line first uses `shift(-variable2)` to move the "High" price column `variable2` days forward (future). Then, it performs a similar rolling window calculation as before to find the maximum value within the next `variable2` days (excluding the current day). SImilarly we calculate the minimum value of the "Low" price shifted `variable2` days forward.

5. Percentage Difference from Future High and Low:

   These calculations find the percentage difference between the "Close" price and the future high/low for each day. The calculations are similar to those used for historical differences, but they use the future high/low values instead.

Task 4: Machine Learning Model

Imports:

- Imports necessary libraries, including `pandas`, `numpy`, `LinearRegression`, and others for model building, evaluation, and saving.

Set Parameters:

- Defines `variable1` and `variable2` to set look-back and look-forward periods.

`train_model` Function:

- Purpose: Trains a machine learning model (multi-target linear regression) to predict future price differences.
- Steps:
  - Defines feature columns (based on historical metrics) and target columns (future price differences).
  - Splits the dataset into training and testing sets.
  - Initializes and trains a `MultiOutputRegressor` with `LinearRegression`.
  - Evaluates the model using RMSE and R^2 score.
  - Saves the trained model to a file (`trained_model.pkl`).
- Output: Returns the trained model and evaluation metrics.

`predict_outcomes` Function:

- Purpose: Uses the trained model to make predictions on new data.
- Input: Takes in a list of feature values (`input_features`) for prediction.
- Output: Returns predicted values for the two target variables.

`main` Function:

- Steps:
  - Loads the dataset.
  - Calls `train_model` to train the model and prints metrics.
  - Loads the saved model (`trained_model.pkl`).
  - Uses `predict_outcomes` with new example input data to generate and print predictions.

Execution:

- The code runs the `main()` function if executed as a standalone script, training the model, evaluating it, and making predictions with new data.

Key points and challenges:

- Used **Linear Regression** for this task **ease of training** and **explainability** compared to other models compared to other models like neural networks and tree-based algorithms which require a lot of parameter tuning and optimization to achieve the best results and they also are prone to getting **overfit** a lot.

- For the use of this project considering the parameters specified to be used for training the ML model I had to limit the model's capabilities to only one dataset that is BTC/USD as each crypto pair has different price dynamics influenced by different market factors, which means it can predict the future values for this crypto pair, however for future purposes we can incorporate other crypto pairs data and append it to this dataset by adding another column called which mentions the crypto pair name.

- Linear Regression did provide a fairly **decent accuracy** and {'RMSE': 6.283251564166778, 'R2_Score': 0.10750164002639623}, however it is to be noted that it may **not be the best option** considering the **nonlinear nature** of financial data.

- One of the key challenges was **task 3** in which I had to calculate those different metrics for which I took the help of **AI tools like chatgpt-4o and gemini** which helped me discover the **rolling window** function of pandas library which made the task simple.

- Also as we had to predict two target variables in this case, I got to learn about the **MultiOutputRegressor** wrapper which enables the Linear regression model to predict 2 target variables than its standard one.