OpenWeather API Time Series Analysis

KDB+/Q Programming

Ricky Camilo
June 2024

Table of Contents

What is KDB+ and Q?

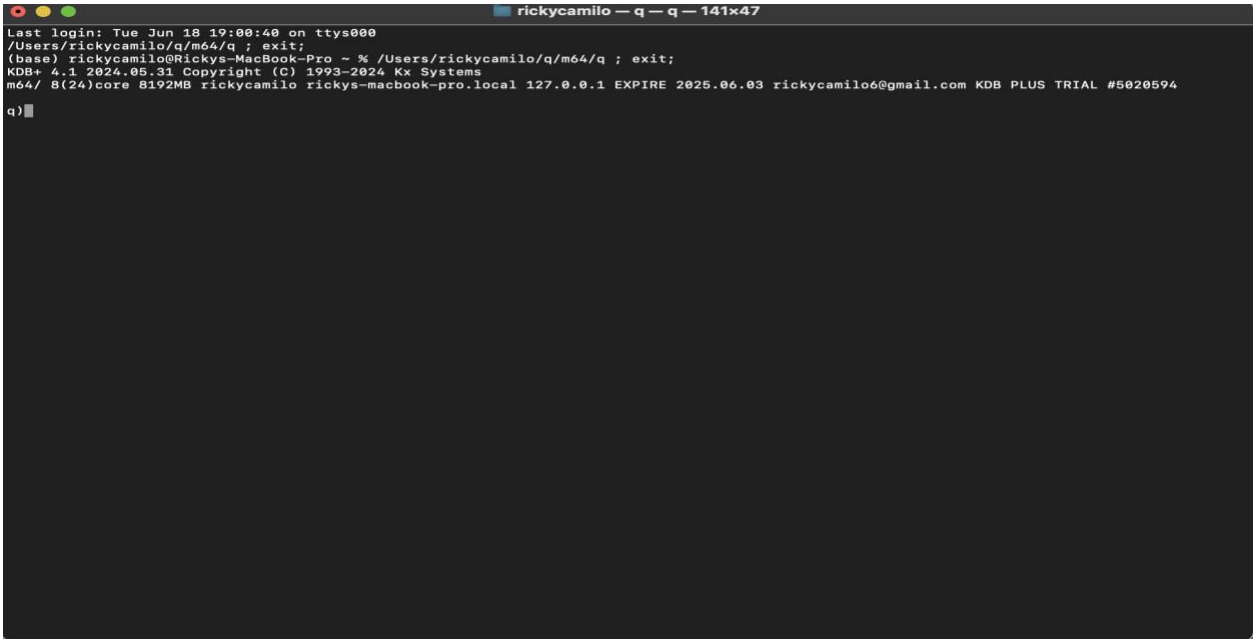
KDB+ is a high-performance column-based database designed for handling and analyzing large datasets in real-time. It's particularly known for its speed and efficiency in processing time-series data, making it popular in financial institutions, where analyzing vast amounts of market data quickly is crucial.

Q is the query language used with KDB+. It's a concise and powerful language designed for interacting with KDB+ databases. Q is optimized for handling time-series data and supports a wide range of operations for querying, filtering, aggregating, and analyzing data efficiently.

Together, KDB+ and Q form a powerful platform for handling and analyzing large datasets, particularly in real-time and high-frequency trading environments. However, they're also used in other industries where rapid data analysis is essential, such as telecommunications and energy trading.

To implement this project, it is essential to note that KDB+ is a proprietary, paid software developed by Kx Systems, and a valid license is required to access its high-performance capabilities for handling and analyzing large volumes of time-series data. Additionally, you will need to have the Q programming language installed on your machine, as it is integral to interacting with KDB+, making HTTP requests, parsing JSON data, and performing real-time analytics.

https://code.kx.com/q4m3/ (Q for Mortals)
https://kx.com/



Key Features of Q:

1. **Array-Oriented:** Q is optimized for handling large arrays of data, making it suitable for processing and analyzing large datasets efficiently.
2. **Time-Series Focused:** It is specifically designed to manage and analyze time-series data.
3. **Concise Syntax:** Q has a very concise and expressive syntax, allowing complex operations to be performed with minimal code.
4. **Interoperability with KDB+:** Q is tightly integrated with KDB+, enabling seamless querying and manipulation of data stored in KDB+ databases.
5. **Functional Programming Paradigm:** Q supports functional programming concepts, making it easier to write and understand complex data transformations.
6. **Performance:** It is highly performant, capable of handling vast amounts of data in real-time, which is crucial for applications in trading and financial analysis.

What's an API?

An API (Application Programming Interface) is a set of protocols, routines, and tools for building software and applications. It specifies how software components should interact and allows different software systems to communicate with each other. APIs can be used for various purposes, such as accessing web services, databases, or hardware devices. Q can be used to call APIs, particularly web APIs, using HTTP(S) requests. The process generally involves the following steps:

1. **Prepare the Request**: Formulate the request URL and headers.
2. **Send the Request**: Use the built-in `.Q.hg` function for HTTP GET requests or `.Q.hp` for HTTP POST requests.
3. **Handle the Response**: Parse the response to extract and use the required data.

The OpenWeather API is a web service provided by OpenWeather, which allows developers to access weather data for various locations around the world. This API offers a range of weather-related information, including current weather conditions, forecasts, historical data, and more. The service is commonly used in applications that require weather data for purposes such as weather forecasting, monitoring, and analysis. It is also completely free to use!

How Will We Collect and Time Series Analyze This Data?

We are going to use the Q programming language to call the OpenWeather API to collect and analyze real-time weather data in London. By leveraging API calling functions of Q, we will send HTTP requests to the OpenWeather API, retrieving current weather conditions such as temperature, humidity, wind speed, and weather descriptions.

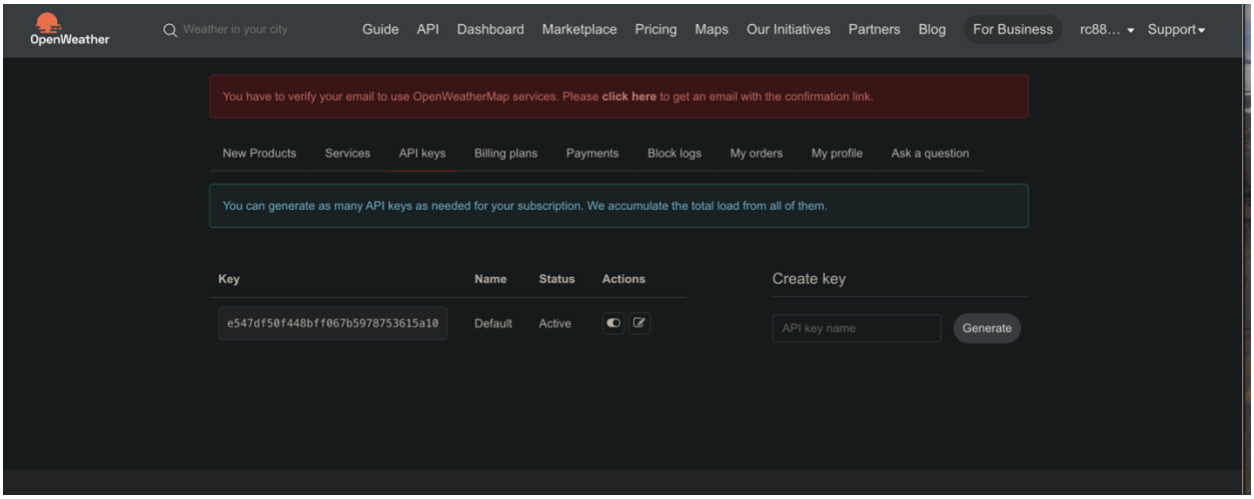
The Q language's efficient data handling and processing capabilities will allow us to parse the JSON responses and store the weather data in a structured format within KDB+. This will enable us to perform real-time analysis, identify patterns, and generate insights on the weather conditions in London. Additionally, Q allows for automated tasks to periodically fetch and update the weather data if such functionality was needed, ensuring our analysis remains current and relevant. Later we will go through a series of queries to further analyze the data.

Code

In this section, we aim to interact with the OpenWeather API to retrieve current weather data for a specific location. We'll utilize the API to make HTTP requests, parse JSON responses, create tables from the parsed data, and perform queries to extract relevant weather information.

Obtaining API Key

Before accessing the OpenWeather API, we need to obtain an API key by registering on the OpenWeather website. The API key is essential for authenticating our requests to the API endpoints. In the case OpenWeather API these are free since it’s public.



Making HTTP GET Request

We use Q to make an HTTP GET request to the OpenWeather API endpoint, passing necessary parameters such as the API key and location coordinates.

```
// Define the base URL of the OpenWeatherMap API endpoint
baseUrl: "http://api.openweathermap.org/data/2.5/weather"

// Define the city for which to get the weather
city: "London"

// Define your API key
apiKey: "e547df50f448bff067b5978753615a10"

// Construct the full URL with query parameters (city and API key)
url: baseUrl, "?q=", city, "&appid=", apiKey

// Make the GET request
response: .Q.hg url
```

Parsing the JSON Response

After receiving the HTTP response from the API, we parse the JSON data to extract relevant weather information. This includes data such as temperature, humidity, wind speed, and weather description.

```
// Parse the JSON response
parsedResponse: .j.k response
```

```
// Parse the JSON response
parsedResponse: .j.k response

// Extract weather information from parsed response
temperature: parsedResponse[`main;`temp]
humidity: parsedResponse[`main;`humidity]
windSpeed: parsedResponse[`wind;`speed]
weatherDescription: parsedResponse[`weather;0;`description]
```

```
q)parsedResponse
coord      | `lon`lat!-0.1257 51.5085
weather    | +`id`main`description`icon!(,803f;,"Clouds";,"broken clouds";,"04..
base       | "stations"
main       | `temp`feels_like`temp_min`temp_max`pressure`humidity!293.09 292.5..
visibility| 10000f
wind       | `speed`deg!4.63 40
clouds     | (,`all)!,75f
dt         | 1.718804e+09
sys        | `type`id`country`sunrise`sunset!(2f;268730f;"GB";1.718769e+09;1.7..
timezone   | 3600f
id         | 2643743f
name       | "London"
cod        | 200f
```

Creating Tables from Parsed Data

We create a table from the "weather" data to organize and manipulate the information more effectively.

```
// Create tables for weather data
weatherTable: ([] temperature: temperature; humidity: humidity; windSpeed:
windSpeed; weatherDescription: weatherDescription)
```

```
weatherTable
temperature humidity windSpeed weatherDescription
---------------------------------------------
293.09     52       4.63      b
293.09     52       4.63      r
293.09     52       4.63      o
293.09     52       4.63      k
```

```
293.09    52    4.63    e
293.09    52    4.63    n
293.09    52    4.63
293.09    52    4.63    c
293.09    52    4.63    l
293.09    52    4.63    o
293.09    52    4.63    u
293.09    52    4.63    d
293.09    52    4.63    s
```

Queries

We can perform various queries on the created tables to extract specific information or perform calculations.

Example Queries

1.Count the number of records in the table:

```
q)count weatherTable
13
```

2.Select specific columns (e.g., temperature and humidity):

```
q) select temperature, humidity from weatherTable
temperature humidity
-----------------
293.09      52
293.09      52
293.09      52
293.09      52
293.09      52
293.09      52
293.09      52
```

```
293.09    52
293.09    52
293.09    52
293.09    52
293.09    52
293.09    52
```

3.Adding columns to the weatherTable

```
weatherTable: ([] timestamp: timestamp; temperature: temperature; feelsLike: feelsLike; tempMin: tempMin; tempMax: tempMax; pressure: pressure; humidity: humidity; visibility: visibility; windSpeed: windSpeed; weatherDescription: weatherDescription)
q)
q)weatherTable
timestamp    temperature feelsLike tempMin tempMax pressure humidity visibili..
--------------------------------------------------------------------------..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
1.718806e+09 293.01      292.44    290.85  294.89  1022     53       10000  ..
```

4.Creating a mainTable, and windData (table) and weatherTable

```
// Extract main weather data

mainData: parsedResponse[`main]
```

```
// Create a table for main weather data

mainTable: flip `temp`feels_like`temp_min`temp_max`pressure`humidity!((enlist mainData`temp); (enlist mainData`feels_like); (enlist mainData`temp_min); (enlist mainData`temp_max); (enlist mainData`pressure); (enlist mainData`humidity))


// Extract weather description (assume there's only one weather entry)

weatherDescription: parsedResponse[`weather] 0

// Create a table for weather description

weatherTable: flip `main`description`icon!((enlist weatherDescription`main); (enlist weatherDescription`description); (enlist weatherDescription`icon))

// Extract wind data

windData: parsedResponse[`wind] // Create a table for wind data windTable: flip `speed`deg!((enlist windData`speed); (enlist windData`deg))


//Print the 3 tables

windData
weatherTable
mainTable

q))windData
weatherTable
mainTable
speed| 3.09
deg  | 10
q))main    description icon
------------------------
"Clear" "clear sky" "01d"
q))temp   feels_like temp_min temp_max pressure humidity
-----------------------------------------------------
285.31 284.77    282.18   287.67   1017     84
```

5.Calculating the difference between temp_max and temp_min

```
// Calculate the difference between temp_max and temp_min

update temp_diff: temp_max - temp_min from mainTable
q))temp   feels_like temp_min temp_max pressure humidity temp_diff
-----------------------------------------------------------------
285.31 284.77    282.18   287.67   1017    84      5.49
```

6.Calculating humidity index

```
// Calculate a simple humidity index (temperature multiplied by humidity)

update humidity_index: temp * humidity from mainTable
q))))temp   feels_like temp_min temp_max pressure humidity humidity_index
-------------------------------------------------------------------
285.3184.77    282.18   287.67   1017    84      23966.04
```

7.Convert temperature from Kelvin to Celsius

```
// Convert temperature fields from Kelvin to Celsius

update temp_C: temp - 273.15, feels_like_C: feels_like - 273.15, temp_min_C:
temp_min - 273.15, temp_max_C: temp_max - 273.15 from mainTable
q))))temp   feels_like temp_min temp_max pressure humidity temp_C feels_like_C
tem..
-------------------------------------------------------------------------..
285.31 284.77    282.18   287.67   1017    84      12.16  11.62      9.0..
```

Conclusion

The integration of the OpenWeather API with KDB+ and the Q programming language enabled real-time weather data retrieval, storage, and analysis. By utilizing Q for API calls and JSON parsing, weather data was efficiently ingested and organized into structured KDB+ tables. This

facilitated dynamic, real-time analytics, demonstrating the strengths of KDB+ in handling high-frequency time-series data and the robust capabilities of Q for real-time data processing. The implementation highlighted the effectiveness of KDB+ and Q for real-time environmental data applications.