Imperial College London

Dyson School of Design Engineering

Sensing & Internet of Things

Coursework Report

Benedict Greenberg

01071639

10th January 2019

Code Repository	https://github.com/nebbles/SIOT
Presentation Video	https://goo.gl/xmT9Xe
Data Storage	https://goo.gl/sTxePj

Coursework 1: Sensing

Introduction and objectives

This section aims to focus on the automated collection, pre-processing and basic analysis of two open APIs (application programmable interfaces) available from independent sources. The data sources themselves are completely different and independent, and are time-series based. The data sources are sampled live over a period of at least one week.

The key objectives for this section are:

1. Establish an independent and remotely accessible host for data collection.
2. Establish a robust system for collection of data autonomously without human direction required.
3. Establish a graceful failing mechanism and provide information to administrator without active monitoring needed.
4. Present collected data in real-time through an easily accessible portal.
5. Identify any basic characteristics of the collected data.

Data Sources and sensing set-up

Set-up of host server

A critical foundational requirement of the data collection process was to have an remotely accessible host computer that would do the collection on behalf of the administrator. Due to the scope of the project, a paid service was undesirable, and so a Raspberry Pi computer suited this requirement very well. The Pi was set up with NOOBS (New Out Of Box Software) with a number of additional software packages removed to lighten the installation and free disk space. These were non-essential to the host, such as Scratch and Wolfram Alpha. As a result, multiple gigabytes of space was freed.

For remote connection, the IP address of the Pi was needed regularly. Since the Pi was set up in a domestic dwelling, the ISP-assigned IP address was dynamic. To solve this, the author used dynamic DNS records with their personal domain registrar. This allowed the DNS records to be updated remotely with a custom URL and the HTTP GET method.

A custom script was written for the DDNS (dynamic domain name system) update. This script can be found in the following directory[1]:

```
$ cd ddns/
$ nano ddns.py
```

This script was carefully written to ensure that the detected public IP had changed since the last check, and only if it had would an update be sent to the domain registrar. This would ensure the registrar is not spammed with needless updates.

This script was then scheduled to run every 10th minute and every time the Pi is rebooted. The scheduling was achieved with **crontab**, a built-in command scheduler for Unix based operating systems.

> ***Note:*** *For remote connections to the host via SSH, port 22 (standard for the SSH protocol) needs to be forwarded on the domestic router.*

[1] *All code references are made relative top-level directory of the project repository (unless otherwise specified).*

Once complete, the Pi was ready to host data collection scripts. A new non-admin (for safety) account was created to run the data collection scripts and to remotely log in to the Pi. Since the set up was robust and accessible remotely, user accounts were created for others who needed to host data collection scripts on an always-on always-connected system. Log in could be achieved from any location:

```
$ ssh siot@greenberg.io
```

Selection of Data Sources

Choosing the data source was at first particularly hard. The aim was to find two completely different web data APIs that could both have easy-to-parse data in JSON (JavaScript Object Notation) format.

The first source selected was weather data. With many sources available, Dark Sky[2] was chosen due to their renown hyper-local data provision. Dark Sky sources from many other independent sources and knit together using predictive models to provide local weather data. The API was free, with a generous upper limit on call rates at 1,000 per day.

Finding a suitable second data API was difficult, especially given the desire to have a well-documented, free, RESTful API. Eventually, a few stock and forex APIs were found. Alpha Vantage[3] was chosen due to its generous call limits (500 per day) and simple concise documentation. API keys were also very easy to produce, allowing for two keys to be made, one for stock quotes for the FTSE100 to be called, and another for quotes on GBP/USD. An alternative data source identified was 1Forge[4].

Data collection and storage process

Sampling Rate

Each data source was sampled at the same rate, in order to make data cross correlation more suitable. They were sampled at an allowed and suitable call rate of every 3 minutes. This comes to a total of 480 calls per day which is under the rate limit set by Alpha Vantage. This high sample rate was useful for the weather data which provides minute by minute updates. Historical data however is reduced to hourly summaries. This level of granularity is unsuitable for a weeks-worth of data (168 data points), and thus real-time collection was used to collect at least 3,360 data points.

The scripts are run automatically by the crontab scheduler. The setup for this can be found in a tutorial the author wrote [link].

Dark Sky

The Dark Sky API is very well documented, and allows for plenty of types of data to be collected in a single call. It returns JSON.

```
https://api.darksky.net/forecast/{apikey}/{lo
c[0]:},{loc[1]:}?exclude=minutely,hourly,dail
y&units=si

# `apikey` refers to Dark Sky Dev account key
# `loc` refers to tuple containing longitude,
latitude
```

[2] https://darksky.net/dev

[3] https://www.alphavantage.co/support/#api-key

[4] https://1forge.com/forex-data-api

The API URL above fetches the current weather data for a given location when called. The location given in the script is that of the Dyson School of Design Engineering, Imperial College London, located on Exhibition Road.

The attributes of the current weather that are stored by the script are a selection of the available fields. This is both for the sake of relevance and storage space. These fields are:

```
# apparentTemperature
# precipProbability
# precipIntensity
# humidity
# pressure
# nearestStormDistance
```

The storage process for both Dark Sky and Alpha Vantage are the same and are detailed in a later section.

Alpha Vantage

The Alpha Vantage API is similar to that of Dark Sky.

```
https://www.alphavantage.co/query?function=GL
OBAL_QUOTE&symbol={symbol}&apikey={apikey}

# `symbol` refers to market symbol (^FTSE)
# `apikey` refers to developer authentication

https://www.alphavantage.co/query?function=CU
RRENCY_EXCHANGE_RATE&from_currency={fc}&to_cu
rrency={tc}&apikey={apikey}

# `fc` refers to currency to convert from
# `tc` refers to currency to convert to
# `apikey` refers to developer authentication
```

The results of these are JSON data from which the FTSE quote and exchange rate can easily be extracted.

Data Storage Process

Both data collection scripts follow the same basic structure. These scripts can be found in the code repository under **data_collection**.

```
$ cd data_collection/
$ nano weather.py
$ nano stocks.py
```

The steps these scripts follow is:

1. Timestamp the time the script was called. This is important as it can be used to compare across multiple databases. This is also required for the time-series analysis.
2. The call is made to API URLs. This call returns JSON data that is parsed into native python structures (lists and dictionaries). From this, only the desired information is formatted into a new table row.
3. The data is then stored in a local CSV file as a backup to the following step.
4. The scripts then attempt to store their fetched data point in a Google Sheet. This is a useful step for administration to check the script is still performing as expected.
5. At any point in the process, if there is an exception raised due to a problem, the error is caught and a complete stack trace is emailed to the administrator. This ensures that the administrator does not need to actively monitor the scripts to ensure they have not encountered errors.

The Google Sheets API has less restrictive limits than the two data APIs used in this project at 100 requests per 100 seconds per user permitted (no

daily limits). At 3 calls every 3 minutes, this usage is well under the developer limit.

The data for this project was loaded into a Google Sheet for the sake of the administrator and easy remote access to the data from any device. This Sheet can be viewed at the following link: https://goo.gl/sTxePj

The sheet contains a data dashboard for visualisation of the data and an indicator to show whether a data point was received within the recent expected window.

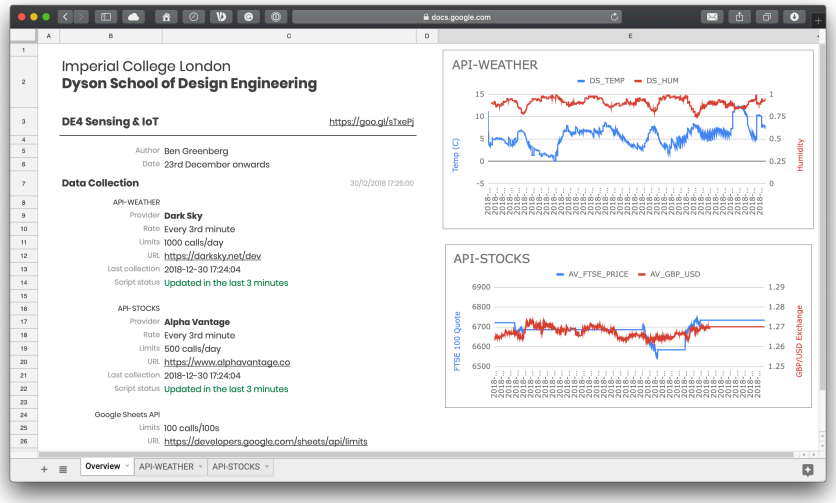


Figure 1: Dashboard view of data collection (may not be latest version)

Google Sheets stores up to 5 million cells per document. This limitation is *well* above the estimated number of required cells for this projects.

Basic characteristics of the end-to-end system

Achieved Characteristics

The fundamental objectives of this coursework were achieved in a timely manner with many features coming in use for the autonomy of the system.

1. *Establish an independent and remotely accessible host for data collection.*
   The host was accessible remotely via author's domain from anywhere in the world at no extra cost. The host is always-on and internet connected. Dynamic IP updates automatically.

2. *Establish a robust system for collection of data autonomously without human direction required.*
   Scripts are run automatically with the crontab scheduler. Calls are made to well documented RESTful APIs for both weather and stock data. Relevant data is extracted and stored in the target databases.

3. *Establish a graceful failing mechanism and provide information to administrator without active monitoring needed.*
   Any issues with the call to any APIs are caught by the script and sent automatically to author's email address. See Figure 2 for a real example of how this assisted the author.

4. *Present collected data in real-time through an easily accessible portal.*
   See Figure 1 for screenshot of Google Sheets based dashboard for viewing status of incoming data and live visualisation.

5. *Identify any basic characteristics of the collected data.*
   See the following section.

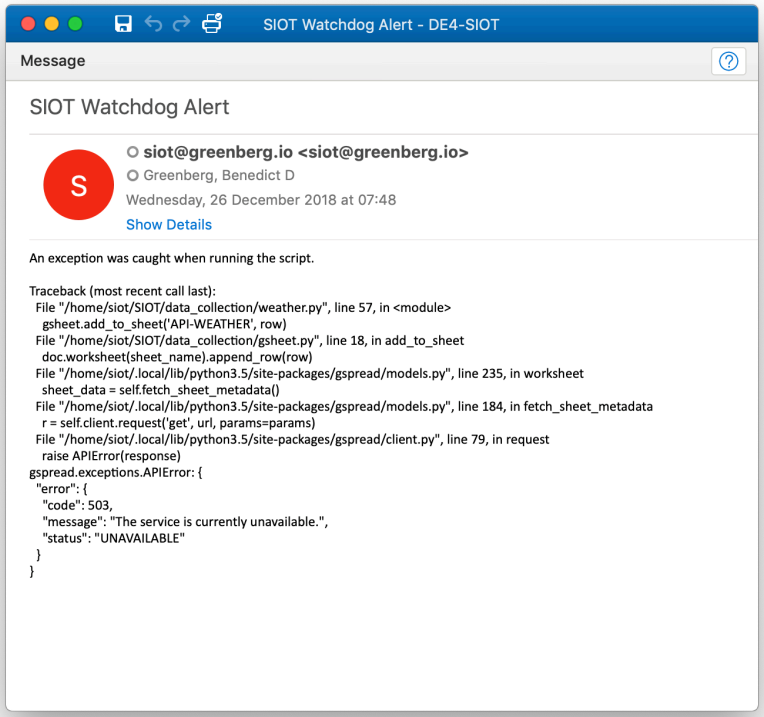


Figure 2: Real example of automatic error catch and email to administrator

Streamlined Basic Time-Series Data Analysis

Basic data analysis was conducted on the collected data. This was processed and analysed in the [data_collection/Data_Analysis.ipynb] file which can be found on the project repository.

The normalise data sources can be seen in Figure 3. A basic seasonal decomposition of these signals can be seen in Figure 4. It roughly indicates there is little to no correlation in the data.

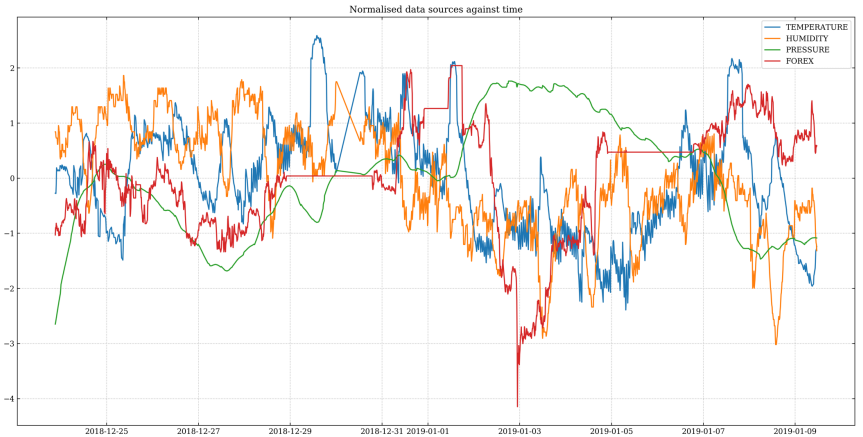


Figure 3: Normalised data sources after they had been resampled to 10 minute periods and linearly interpolated

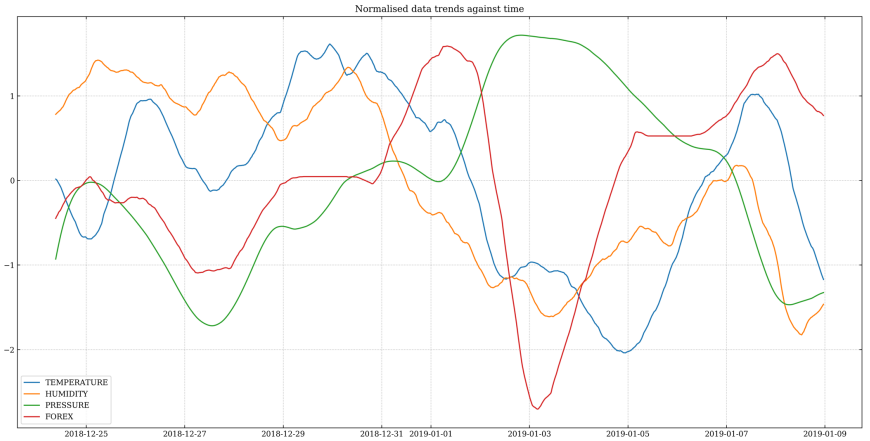


Figure 4: Normalised trend signals of data sources after undergoing a seasonal decomposition

Coursework 2: Internet of Things

Data platform

The data platform built was designed to be entirely front-end and dynamically load any necessary content in without the requirement for a backend service. This dramatically reduces the effort required to deploy the site which was an important outcome of the project. The author will deploy this site statically to his personal portfolio. Further possible functionality will be discussed later in the section: Avenues for future work and potential impact.

Interaction

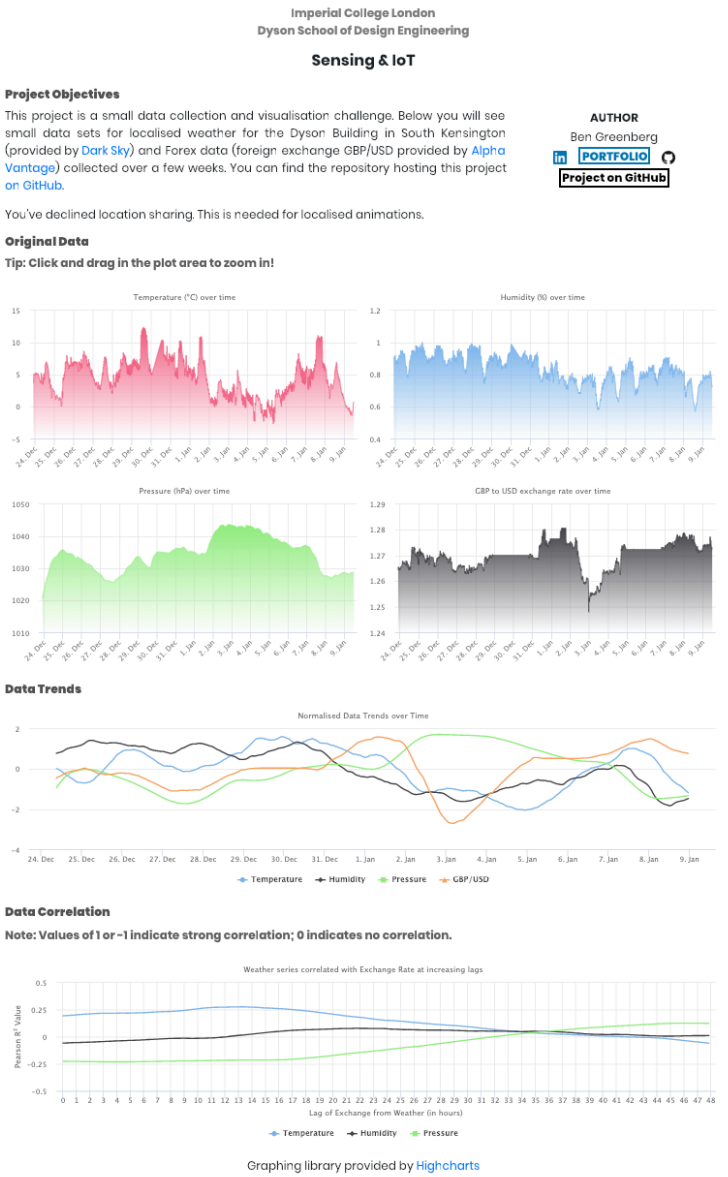
The site was designed mobile first and has an fluid layout that adapts to both narrow and large screen sizes. A charting library was used to display the data once it had been processed in and read from a CSV. This CSV was an output of the previous stage which saw the analysis of the cleaned data in a Jupyter Notebook.

The charting library used was Highcharts which allows for highly granular specification of the appearance and interaction. The user is able to zoom in on the data by clicking and dragging a section of the chart (on desktop) and this is also supported on mobile.

Visualisation

The visualisations chosen help to give the user an overview of the underlying data. This is in contrast with publishing all of the data analysis charts etc. since this would more likely confuse the user.

The normalised trend data allows the user to absorb the general relationship between the multiple sources very quickly and see how the data correlates. For slightly in-depth metrics, there is a coefficient vs. lag



plot displayed to prove there is no correlation even if the Foreign exchange had a delayed response.

Actuation

Actuation is performed in two places within the project. During the data collection phase, any form of error in the remote scripts will automatically be caught and then a full traceback will be sent to the administrator so they are aware of the problem and know whether it needs to be addressed.

Furthermore, the site features an engaging animation for the end user that will inform them if it is raining where they are. This is dynamic, but can also be disabled manually by the user if necessary.

Data analytics, inferences and insights

Once all the raw data collected from Dark Sky and Alpha Vantage was imported into the notebook, it was fully displayed. Immediately it was clear that certain variables were useless to the analysis. Precipitation, precipitation intensity, storm distance were all irrelevant to the time period the data was collected over. These were all eliminated. From the stock data, the FTSE 100 quote was largely unusable due to the lengthy periods of market close over the festive period. Thus this was eliminated too and the Foreign Exchange price (GBP to USD) was used instead.

The four selected variables were then resampled to 10 minute periods. Despite originally collecting the data at a sample rate of every three minutes, in order to have a significantly sized dataset, the data collection process had not been flawless due to a faulty Raspberry Pi. This meant there were a few small gaps in the data. As a compromise, the data was resampled to 10 minutes to smooth over any odd gaps there may have been, and then interpolated linearly to fill one or two larger gaps in the data.

The data was then decomposed seasonally (using the additive model) in order to extract trend and seasonality data. This mostly indicated how little seasonality was present in the data due to the short collection period of 16 days. Most seasonality (at daily frequency) showed that it had a marginal impact on the actual time-series.

The full and partial autocorrelation analysis run (seen in Figure 5) did indicate that there was a roughly daily seasonal correlation for the temperature and humidity signals however unsurprisingly the pressure and exchange data showed no repetitive correlation at any lag point (which explains the insignificant seasonality during decomposition).

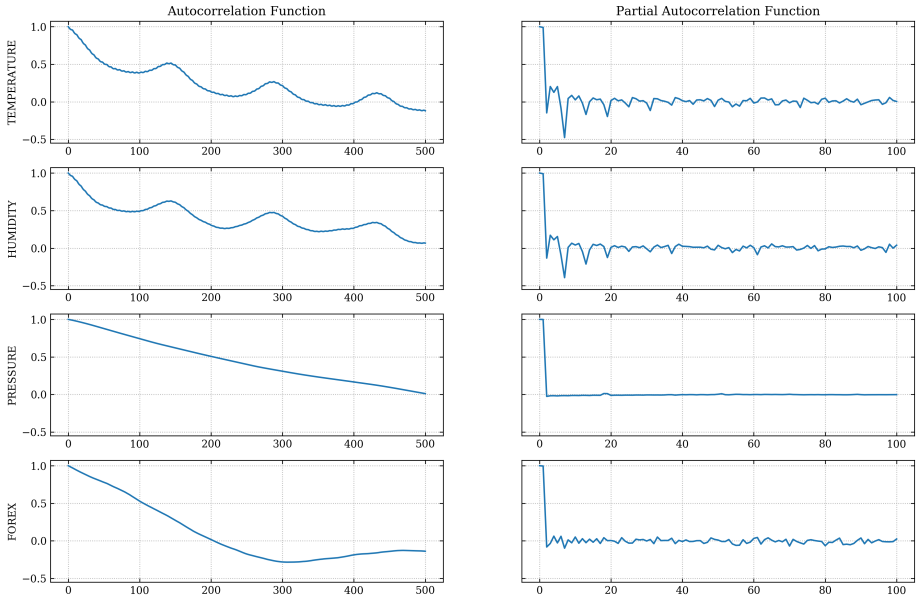


Figure 5: Autocorrelation (left column) and partial autocorrelation (right column) of each time-series

Cross Correlation

Each time-series was then plotted against every other time-series in a matrix scatter plot (data was normalised). It soon become apparent that there was little to know correlation of the data, especially when comparing the Forex (exchange prices) against the weather metrics (see bottom row of Figure 6).

This was followed up with an analysis into whether there could be a lag on the Forex prices to react to one of the other time-series. For this correlation, a Pearson R^2 value was calculated for each lag step (periods of 10 minutes). This was done for a lag of up to 48 hours. As seen in Figure 7, there is a minimal rise of correlation around the 12 hour mark, however this is not significant enough to draw any alternative conclusions from the data.

Ultimately it is clear within the constraints of this small time period, that there is no indication of any series correlation between the trading price of GBP/USD and key weather metrics.

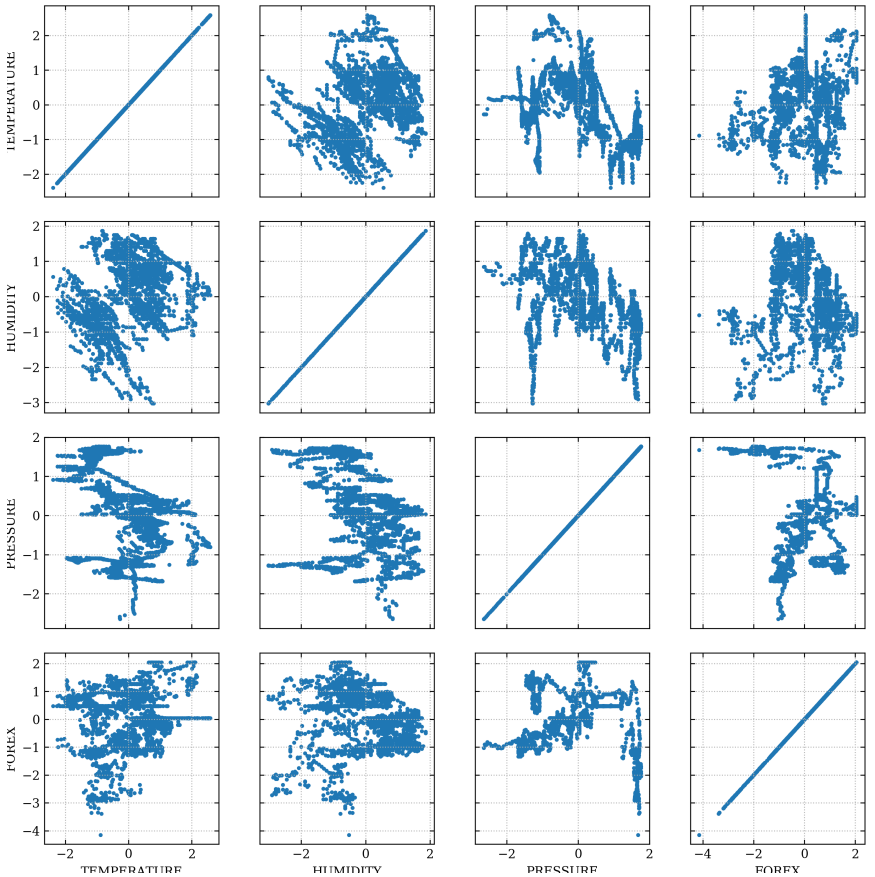


Figure 6: Matrix scatter plot of each time-series against each time-series

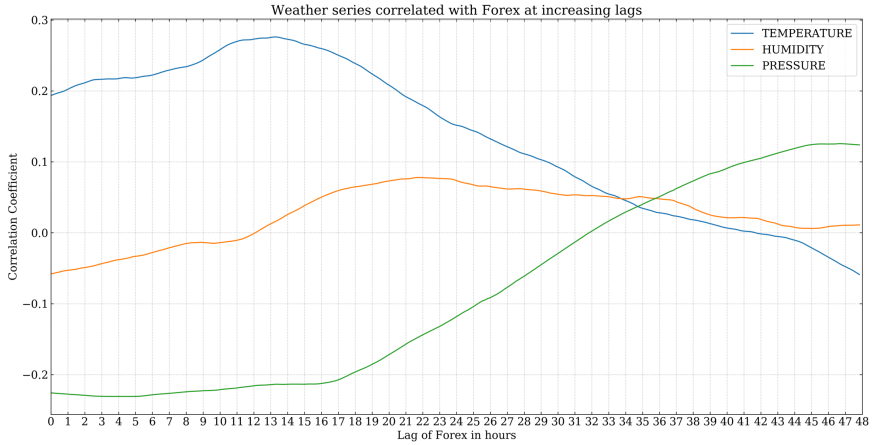


Figure 7: Correlation of weather metrics and Forex at increasingly lagged intervals

Discussions on the important aspects of the project

There were a number of key goals from the project. Some were mentioned previously in relation to the data collection however overall, achieved objectives were:

1. Established an independent and remotely accessible host for autonomous data collection.

2. Established a recovery system of backing up data to two locations (Google Sheets and CSV) in case there was an issue with the Google API. Autonomously inform the administrator of any issues with the script runtime.

3. Visualise collected data in real-time through an easily accessible dashboard with identifiable status alerts.

4. Evaluated basic and cross analysis of time-series data sources to conclude there is no indication of any dependency or correlation.

5. Build a simple and easily-understood dashboard interface to inform a layperson of the relationship of these data sources.

6. Bundle all code for dashboard interface into front-end JavaScript for easy deployment to static web hosting services.

7. Build dashboard with highly functional graphs that allows for user interaction. Design and build layout to be fluid and mobile friendly.

8. Integrate a novelty interaction/actuation to the webpage to further engage users with the relevant data.

Avenues for future work and potential impact

During the project, there were many phases that were streamlined due to its the small scale nature. Data storage for example was via a Google spreadsheet with a backup to CSV. For very large datasets, in a more dynamic scenario, this is not the most ideal storage.

There were a number of areas that were identified for improvement, however were not addressed due to time constraints and project scale. These were:

- Data storage methods
- Data processing
- Data recall

Within an corporate setting, these steps would make sense due to increased budgets (allowing for increased API access and more computational resources), and demand to serve a larger client base.

Data Storage Methods

On a larger scale system, data storage would be better suited to a cloud based database. This could be serviced from AWS, Google, MS Azure, etc. or even from private servers, however it would be an important stage to ensuring all endpoints had ready access to the same data.

This would replace the current Google Sheet and CSV solution, however CSV storage may be used locally for short periods if data collection continued but access to the centralised database was blocked for some reason.

Data Processing

In a larger network, each aspect would be managed by different workers nodes. The processing node would be a backend service to act on the data in one of two manners, either when a user is requesting an up to date analysis, or periodically through scheduled updates.

This could be done on a large scale, and more advanced analysis could be performed using other online services for machine learning etc.; importantly this stage would not be reliant on a manual administrator analysis but instead would be end-to-end automated since the desired statistical tests would be predetermined.

Data Recall

The final major area to develop would be the method of data recall. This could either be a more developed front-end based in JavaScript, or could be down to a dedicated back-end sat behind the web-based user interface. Modern web development has come a long way, and with Electron a valuable framework for creating cross-platform applications, it would be easy to port from a web dashboard to a desktop application.

This would mean that a suitable API would need to be established for the front-end code to be able to fetch the latest data from the database. The API would also manage any heavy pre-processing required for the displaying of the data so as to reduce the load on the client hardware. As such, a Golang-based custom built API is recommended. With strong concurrency features (useful for large data processing), it lends itself well to great scalability with a natural ability to adhere to microservice architecture.

For the sake of rapid DevOps, Docker would be employed to maintain development and deployment speeds whilst ensuring the service-system is able to cope with a scalable network of data collection nodes depositing to the database, and potentially large number of consumer at interface touchpoints.

Conclusion

This project as provided an simple insight into semi-automation of a complete data collection and consumption workflow, with a developed solution that goes end-to-end.