**4 Proposed System Design**

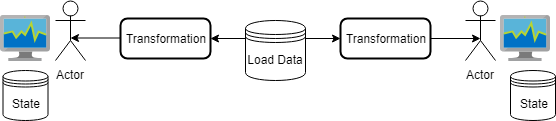
**4.1 Architectural Design**

Security Considerations

Maintaining the confidentiality of the SONI dataset used in the system is required. SONI have proprietary rights to the raw load dataset and model forecasts built using the historical load data have commercial value. The endpoint that returns the data used to produce visualised and statistical results must not allow a user to have access to the load dataset in its raw form. A proposed solution is implementing a transformation process at the endpoint to return only the data required to produce results to the user. This prevents a user retrieving the raw load dataset. However, this transformation does not prevent a malicious actor scraping the load dataset with multiple calls and reassembling the data into its raw form. Effective access control of the system preventing malicious users from accessing the system mitigates the threat. This is outside the scope of the system development and should be implemented by those deploying the system to their environment.

System Processes

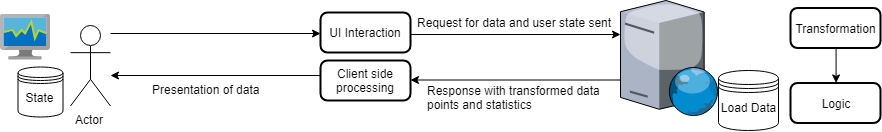
***Figure 3.1*** *Data centred architecture [1]. The process of the system returning different transformations of data to multiple actors concurrently.*



The fundamental process the system has performs is transforming the load dataset, performing logic on that transformation, and producing a results to the user. The stakeholder required that multiple users could use the system at the same time, and hence the system must be capable of propagating different transformations of the dataset concurrently. An additional complexity is that different users have their own unique state interacting with the system e.g. added models. A stateless design pattern was chosen for the system to manage user state. The state of the user is stored locally in their client instead of a server managing their session and data. The user’s state is immutable with a change of state through interaction with the system creating a new state [2]. The dataset must also be immutable with different transformations returned determined by the users’ state as input **see Figure 4.1**.

The stakeholder’s requirement that the system should be deployable to the internet contributed to the choice of a stateless design pattern. Client-side processing managing the state reduces the dependency on server-side resources to manage state, which with greater usage increases operating costs. Furthermore, the logic of using the transformed data and plotting a visualisation of the graph delegated to client-side saves on server-side resources. Client-side processing ensures the system is more scalable horizontally and runs much better under periods of high load, **see Figure 4.2** [3].

***Figure 4.2*** *The process of a user interacting with a web server and being returned a transformation of the data and statistics that is presented onto their client*



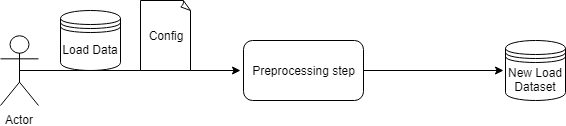
**4.2 Dataset Oriented Design**

Data Contract

The proposed features of the system were based on the capabilities of the SONI dataset. However, the stakeholder required that the system be agnostic to the dataset. The same capabilities as provided for the SONI dataset in the system had to be available for use with other load datasets. Hence, a data contract was required for a dataset to be compatible for use with the system. This is listed in the README of the system [1].

Preprocessing

***Figure 4.3*** *The process of preprocessing of a load dataset, augmenting it with required model variables for highlighting and model construction functionality.*



There are additional challenges by designing the system to be used with multiple datasets. The feature of highlighting days (**FTR03)** requires the dataset to denote whether the load entries are holiday days and contain date derived variables. Default linear regression forecasting models included in the system (**FTR04)** that perform optimally on one dataset may not be performant with other datasets. The dataset may also not have the required input variables to construct the model. A proposed solution to this is a preprocessing step performed on the dataset before it is loaded into the system, **see Figure 4.3**. This adds the columns required by linear regression models and highlighting functionality. The fields added after the preprocessing step are listed in the README [1].

Configuration

The configuration file specific to the dataset contains a list of linear regression models with information required for construction, and descriptive information to present to the user in the system. The organisation of the configuration is in the README [1]. The configuration contains the variables the model requires for construction, which after the preprocessing step are augmented to the dataset. This enables the direct use of the augmented columns in the dataset in model construction with the system. This is preferable to processing the variables on demand for the user using the system, which will increase the required server-side processing by having to augment the dataset prior to construction. A limitation of this design choice is that it makes model construction restricted to a template and hence more complex models e.g. non-linear models, will require both implementation and configuration changes. However, the performance optimization in not recalculating variables for potentially multiple users of the system constructing the same model is important, especially with high density datasets.

**4.3 User Interface Design**

The user interface design was mainly based upon the functional and non-functional requirements in **chapter 3**. An analysis of MATLAB’s user interface was also undertaken to assess how existing software available to academic users facilitate user interaction to facilitate load forecasting analysis, see **Appendix E**.

Decisions Made from MATLAB Analysis

From analysing MATLAB’s user interface, the following user interface decisions were made:

* The system interaction and information initially visible to the user will only be that facilitates a step in the defined user flow.
* The system will have a tabbed design to split different types of functionality e.g. different graph visualisations
* The system content will be contained within one page and have no unconditional popup windows.
* The system will be resizable to fit the user’s desired window size.
* Upon resizing the window, the visualised graphs will contain the entirety of the range of data requested by scaling their x-y axis.
* The system will require the use of a mouse for navigation, and for text input a keyboard.

Solution Usage

When the system is deployed onto the internet a wide range of devices and browsers will be able to access it. The system must provide an acceptable user experience across the diverse hardware users will use. The rising use of mobile devices to enhance productivity proves a challenge as they have less screen real estate and are less computationally capable than a desktop computer [5]. They also can rely on poor network conditions e.g. mobile internet which can be considerably slower than broadband internet [6]. An approach to allow mobile devices to be supported by the solution was to develop two versions of the system for each type of device. An issue with this approach is that it requires developing features for both synchronously. The approach chosen instead was to make the user interface styling of the individual system responsive to both desktop and mobile devices.

To ensure the system had a responsive design the following user interface decisions were made:

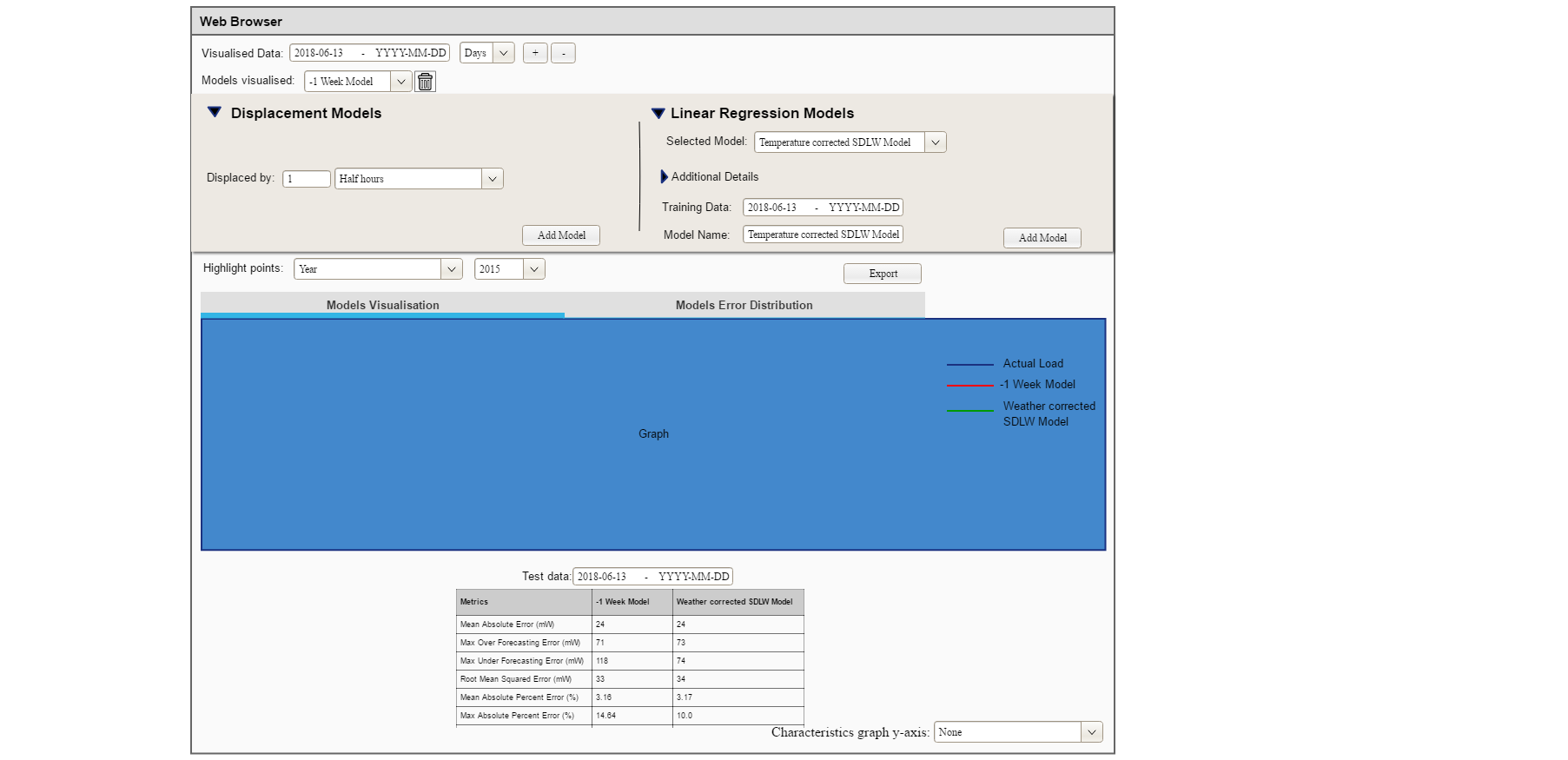
* The styling will be minimalist. This approach is used by the likes of Google to ensure their pages load reasonably quickly on slow connections [6].
* User interface components, visualisations and text will scale to the user’s device screen size.
* Landscape will be chosen as the default screen orientation for user interface designs as greater horizontal space is more effective in visualising electricity load trends over time.

**Academic User Interface**

A prototype of the proposed user interface was created using *Pencil* [7], **see Figure 4.4**. The prototypes were used to communicate the user interface decisions made to the stakeholder in meetings, and feedback received was used to iteratively change the design. When feature development began, the prototype was reviewed to determine whether it required additions to add components to enable the user to interface with the feature’s added functionality.

The user interface provides the academic user a fully customisable suite of options to enable an experimental experience of visualising load and evaluating the performance of different forecasting models. The prototype does not include error message reporting as the expectation is the user will choose options that they know are within the parameters of the dataset they loaded e.g. the dataset has the data required for a displacement model constructed.

User Interface



(a)

(b)

(c)

(d)

(e)

(f)

(g))

(h))

(i))

**Figure 4.4** Prototype of the academic user’s user interface

*(a) Graph visualisation controls*

Enables user data interaction required in **FTR00**.

*(b) List of models the user has added and can remove*

Enables comparing the performance multiple models required in **FTR02.**

(c) Load Forecasting Model Construction

**(b)**

**(a)**



**Figure 4.5** The steps a user construct a load forecasting model using the system   
**(a)** Displacement model **(b)** Linear Regression model

Facilitates the creation of displacement load forecasting models required in **FTR01** and linear regression models required in **FTR04**. The two flows are different with the only step shared between these two processes being the end process ‘Add model’, **see Figure 4.5**. Therefore, the prototype has two separate flows for the academic user to construct models**.**

*(d) Highlighting data points*

Enables selection of the type of day to highlight in the visualisation required in **FTR03.**

*(e) Export button*

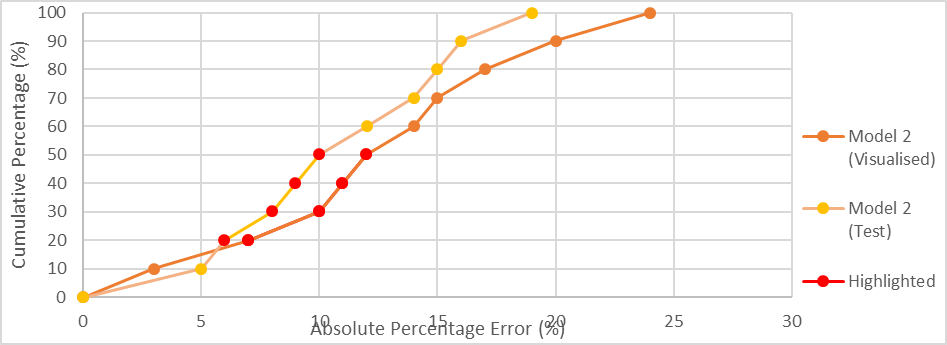
Enables exporting graphical visualisations and statistical data required in **FTR05.**

*(f) Tabs for different visualisation selection*

Enables changing between a load visualisation and error distribution graph, graphs required in **FTR00** and **FTR03.**

*(g) Graphical Visualisations*

**Figure 4.7** Error distribution graph mock-up.



**Figure 4.6** Load visualisation graph mock-up

The choice of colours for different plots in the visualisations is to distinguish different information [9]. The contextual information of the graph (grid, axis, labels, borders) are black, attracting the attention of the user to the brighter coloured visualised data. The background colour of the visualisation was chosen as research has discovered the human visual system perceives different coloured information relative to their definition of white [9].The user specified highlighted days are plotted in a high contrast red to capture the user’s attention and make them distinctive from the other plots. Forecasting model predictions are plotted with different distinct colours. Functionality described in **FTR03** enables the user to customise the colour choice for models.

The error distribution graph in **Figure 4.7** was, through discussions with the stakeholder determined to be the optimal visual presentation of the error distribution of different forecasting models for performance comparison. A histogram was considered initially for presenting the error distribution. However, the presentation chosen is better at conveying the maximum error values at specific points of the dataset e.g. maximum APE at 90% of the data, and for determining the 100% data convergence error value.

A forecasting model’s performance with a range of test data specified is visualised with an analogous colour to the visualised data performance. Making related elements in a graph a similar colour visually conveys they are grouped to the user i.e. the same model with different ranges of data [9]. The highlighted data points can be used to provide an explanation for high errors.

(h) Model Performance Evaluation Table

**Table 4.1** Model metrics table mock-up

|  |  |  |
| --- | --- | --- |
| **Metrics** | Model Name | |
|  | Visualised | Test Data |
| Mean Absolute Error (MW) |  |  |
| Max Over Forecasting Error (MW) |  |  |
| Max Under Forecasting Error (MW) |  |  |
| Root Mean Squared Error (MW) |  |  |
| Mean Absolute Percent Error (%) |  |  |
| Max Absolute Percent Error (%) |  |  |
| Max Under Forecasting Percent Error (%) |  |  |
| Max Over Forecasting Percent Error (%) |  |  |
| 90% Threshold Absolute Percentage Error (%) |  |  |

The metrics table in **Table 4.1** is visible when a forecasting model is added by the user to evaluate the performance of model, as required in **FTR01.** Multiple models can be added to enable the user to compare their forecasting performance, as required in **FTR02**. The MAPE error statistic used in prior analysis was included as it proved to be an effective performance comparison statistic for different forecasting models. Error statistics producing mW error values were chosen to provide an actual MW load value difference between the forecasted load to actual load of the data entry. The root mean square error was chosen for inclusion rather than the mean squared error, which measures the average of the squares of errors. Percentage metrics produce an indication of the model’s forecast error performance relative to the magnitude of the system demand.

There were other considerations made when choosing which error statistics to include:

* The mean statistics were included as they provide a general indicator of the forecasting performance over the whole dataset to the user.
* Maximum error statistics were included as they provide the forecasting error outliers to the user.
* The inclusion of under forecasting and over forecasting statistics enable the user to determine whether the maximum error outliers were under forecasting or under forecasting errors.
* The ‘90% Threshold Absolute Percentage Error’ statistic was included as a statistical representation of the error distribution graph for the user to determine how the forecasting model performs excluding error outliers.

(i) Characteristics comparison

Enables selecting a variable to visually compare with the load visualisation graph, as required in **FTR02.**

**Citations**

[1] https://www.geeksforgeeks.org/software-engineering-architectural-design/

[2] <https://leonmergen.com/on-stateless-software-design-what-is-state-72b45b023ba2>

[3] <https://blog.rackspace.com/coding-in-the-cloud-rule-3-use-a-stateless-design-whenever-possible>

[4] <https://flaviocopes.com/single-page-application/>

[5] https://developer.mozilla.org/en-US/docs/Learn/Tools\_and\_testing/Cross\_browser\_testing

[6] <https://www.smashstack.com/articles/life-in-the-slow-lane-web-designs-for-slow-internet-connections/>)

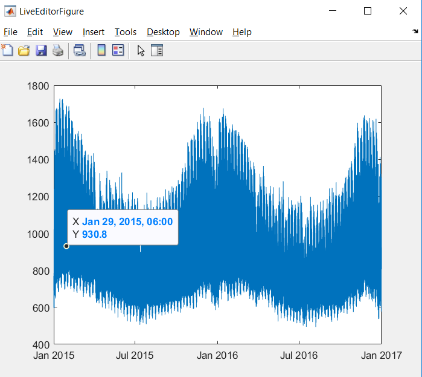
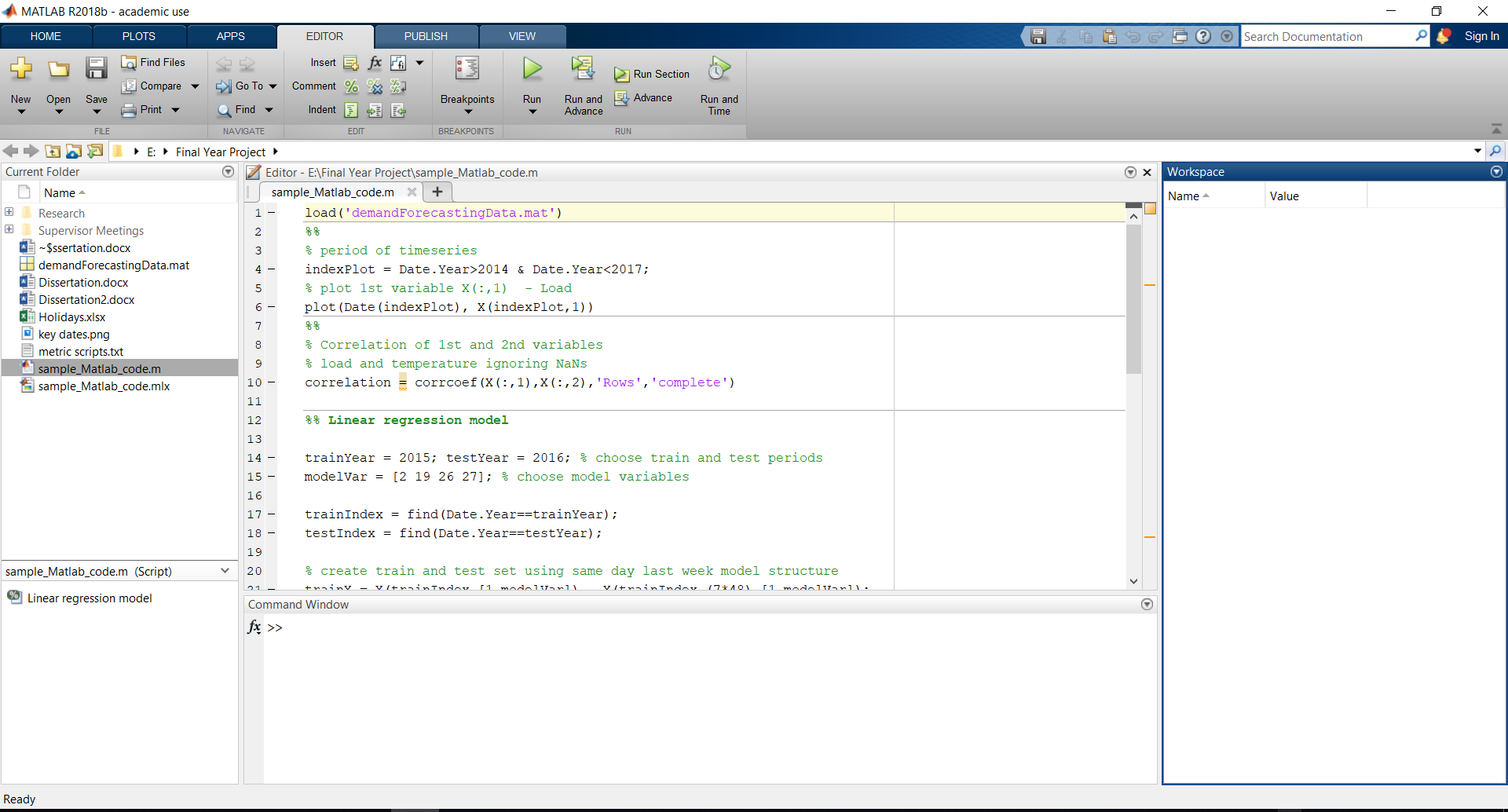
[7] <https://pencil.evolus.vn/>

[8] <http://agilemodeling.com/artifacts/uiPrototype.htm>

[9] Expert Color Choices for Presenting Data

[10] https://www.iso.org/iso-8601-date-and-time-format.html

**Appendix F**

**MATLAB User Interface Analysis*****(a) (b)***

***Figure 3.4*** *Examples of MATLAB’s user interface* ***(a)*** *Loading a MATLAB script****. (b)*** *Visualisation of load data.*

In **Figure 3.4 (a)** the following observations were made:

* The menu operations are visible to the user to interact with the system.
* Tabs categorise interaction specific to the current user flow (light blue) and not specific (dark blue)
* Application functionality is facilitated by buttons with an accompanying informative text label.
* Additional functionality is available on some buttons with an expanding carat.
* The non-modified presentation of the application is a single page with no popups.
* The application window is resizable.
* A mouse and keyboard are used for interaction with the application.

In **(b)**:

* The visualised graph is resizable, with the accuracy of the x and y axis increasing as the window size increases.
* The window’s name and icon change with the context of the user’s functionality.