A visualisation and interaction tool for short-term electricity load forecasting

A dissertation submitted in partial fulfilment of

the requirements for the degree of

BACHELOR OF SCIENCE in Computer Science

in

The Queen's University of Belfast

by

James Robert Bailey

21 April 2019

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC3002 – COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name:	James Bailey	Student Number:	40156063
Project Title:	A visualisation and interaction tool for short-term electricity load forecasting		
Supervisor:	Seán McLoone		

Declaration of Academic Integrity

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

By submitting your dissertation you declare that you have completed the tutorial on plagiarism at http://www.qub.ac.uk/cite2write/introduction5.html and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.

Student's signature James Bailey *21/04/2019*

Acknowledgements

Firstly, I am grateful for the support my parents have provided for me throughout my entire educational journey. I would like to thank my project supervisor Sean McLoone for his guidance throughout the project, and Judith Foster for helping me with handling the dataset. Finally, I would like to thank SONI for allowing access to their load dataset.

Abstract

The goal of the project is the development of a web-based interactive data analysis and visualization suite to enable comparison and evaluation of different short-term electricity load forecasting techniques. Such forecasts are vital to plan for and schedule electricity generation to meet system demand in an efficient and secure manner. The capabilities and accompanying analysis embodied in the suite are informed by data from the Northern Ireland power system provided by SONI. The manual investigative processes undertaken in analysis are the functionality that the software automates for a user. The solution enables users to explore the increasingly complex patterns in electricity load data through dynamic visualisations. The solution enables the user to construct, and systematically evaluate and compare the performance of different displacement and linear regression forecasting models visually and statistically. The linear regression models identified to have the best forecasting performance in accompanying analysis are included within the software.

Contents

1 Introduction

1.1 System Operator Challenges

The core goal of managing electrical supply load is that power generated should be equal to the load demanded. It is expensive and inefficient to store large amounts of electricity; it should be generated as it is demanded. System operators require electricity load forecasting to predict what the load will be in the future to determine a balance between customer demand and generated supply. If the predicted load is greater than the actual load (over-forecasting), the operator will buy more electricity than they need. This will increase their operating costs with their ordered generation units' output exceeding the required generation to match demand.

System operators ensure they deliver a safe and stable feed of electricity by maintaining their grid frequency within an acceptable range using high inertia power generation sources, for example fossil or nuclear fuels [2]. If the forecast is lower than the actual load (under-forecasting), the operator will have to buy considerably more expensive (peaking power) electricity as a redundancy measure to eliminate the power deficit. This makes control of the grid frequency more difficult for the operator and decreases their power generation efficiency by increasing operational costs.

There is an environmental incentive for system operators in predicting the next day's system load demand accurately to accommodate the presence of non-dispatchable renewable energy sources causing a negative load effect. This can reduce CO2 emissions and reliance on fossil fuels by reducing dispatchable power generation [3]. The efficient use of electricity is a major talking point regarding the environmental impact and reducing availability of non-renewable sources [4], and therefore it is important operators reduce waste electricity generation.

1.2 Northern Ireland Context

The dataset used in analysis and development of the solution is from SONI (System Operator for Northern Ireland). SONI operate the electricity transmission system in Northern Ireland. Electricity in Northern Ireland is generated by three main power stations which use coal and gas. SONI use short term next day forecasts of 48 half hourly values early in the afternoon of the preceding day to bid for electricity in I-SEM (Integrated Single Electricity Market). I-SEM is a wholesale market for the whole of the island of Ireland. SONI buy the bulk of their electricity for the following day from the day-ahead I-SEM market which consists of one auction with a deadline of 11:00 GMT [5]. SONI's aim when bidding is to buy the least expensive electricity to match forecasted system demand. Hence, they require accurate forecasts to buy electricity a day ahead without having to bid for more expensive on demand electricity in intraday markets if they under forecasted the system demand load. Inversely, over forecasting the system demand increases their operating costs.

A growing challenge for SONI in creating accurate load forecasts for Northern Ireland is the growth in small-scale renewable energy sources funded through government incentives. Small-scale renewable energy sources can offset a home owner's electricity bill with them using it in preference to grid power, reducing the system demand. Installed small scale generation is now approaching a considerable 20% of NI's average demand [3]. The main two renewable sources are photovoltaic and wind energy with both contributing over 100MW of electricity to the grid. SONI cannot accurately measure the influence of renewable energy generation in different weather conditions as they do not possess the data to pinpoint the exact locations, orientation and the amount of electricity generation. Hence, SONI require load forecasting models that take account of the impact of embedded unmetered renewable generation when producing day ahead forecasts.

1.3 Approaches to Short-Term Load Forecasting

A short-term load forecasting approach is using a displacement model. A displacement model uses an historical system demand value that is displaced from the day to forecast as the load forecast. Displacement models can take advantage of the observed daily cyclic pattern of load for forecasting each system demand time interval. A limitation of using a displacement model to produce load forecasts is that it may not consider load deviations caused by weather-related and/or social factors occurring on the forecasted day.

Another approach is using a linear regression model. A linear regression model can provide accurate forecasts by including explanatory variables that have a strong linear relationship (correlation) with load. Therefore, variable selection for a linear regression model is an important step in building a model that produces accurate load forecasts. Variables are typically chosen through prior heuristic knowledge and understanding of variables that correlate to load deviations. A linear regression model considers load deviations caused by weather-related and/or social factors through its construction with weather and date-time explanatory variables that strongly correlate with load deviations being included.

More advanced linear regression models include an accurate displacement model's load forecast and a correction of it that adds or subtracts a load value to produce a more accurate forecast. The correction's dependent variables can be the deviations in explanatory variables identified to be related to load deviations between the forecasted day and the day used for the displacement model. A challenge in choosing a load forecasting model is balancing the accuracy of the load forecasts with its complexity i.e. number of explanatory variables included.

1.4 Influence of Weather Factors on Load

There are weather-related factors that influence the pattern of electricity load demand. The power output of renewable energy is dependent on the sun and wind. For measuring the influence renewable energy has on system demand load there is typically not a one single base variable, rather the product of several variables. There is a notable delay in time before people react to changes in the weather. Hence, using weather-related temperature variables instead of base temperature variables in load forecasting models can account for this delay and produce more accurate load forecasts.

1.5 Influence of Social Factors on Load

There are social factors that influence load with people using more heating and lighting in the winter months than the summer months that increases electricity demand. People's activities throughout the day effect the load pattern with night time, when most people are sleeping having a less variable load pattern than the period in which people wake up and participate in their daily activities. Unexpected changes in the daily load pattern not considered in the short-term day ahead forecasting model can increase the error of the load forecast. Intraday forecasts are a topic of research not considered in this dissertation which can react to unexpected changes in the daily load pattern.

The pattern for electricity demand is more complex because of public holidays that do not conform to general day load patterns. Days have been observed to be affected by their recency to public holiday days with people taking extended leave, especially in the Christmas period in Northern Ireland. Further increasing the complexity of abnormal day's load pattern, there are one-time events e.g sports games or local events that have an influence on load. Therefore, load forecasting models should include the numerous unique characteristics of a day as input variables to produce an accurate load forecast.

2 Analysis

2.1 Introduction

An analysis of the dataset was undertaken concurrently with the development of the system to appreciate the load forecasting challenges identified in **chapter 1.** The *Jupyter Notebook* analysis environment enabled the creation of load forecasting models and evaluation of their performance statistically in a sandbox environment. Complex data manipulation functionality was written in *Python* in the analysis and was integrated into the solution. Models found to have the best forecasting performance were included in the system. The analysis of model performance was used as a source of proof to cross verify the system's outputted performance results. The dataset was also assessed on its viability to facilitate proposed solution features that leverage specific characteristics of the data.

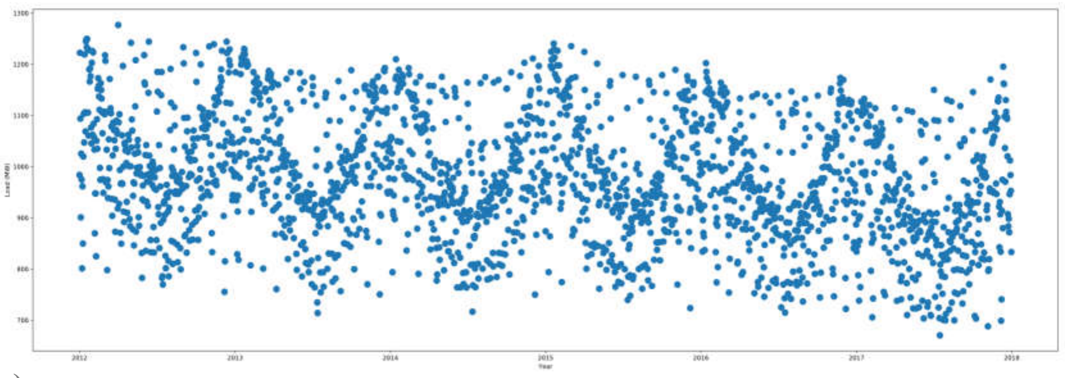
2.2 SONI Dataset

The dataset used for analysis was provided by SONI (System Operator for Northern Ireland). The data was provided in MATLAB data format (.mat). To use the dataset in the analysis environment, its data was converted into Excel workbook format (.xlsx) using MATLAB and then saved as a comma-separated values file (.csv) using Microsoft Excel. The dataset was indexed by date ascending ranging from 1st February 2010 to 1st July 2018, with an interruption in the load value entries between 7th February 2011 and 10th November 2011. The time series interval was 30 minutes.

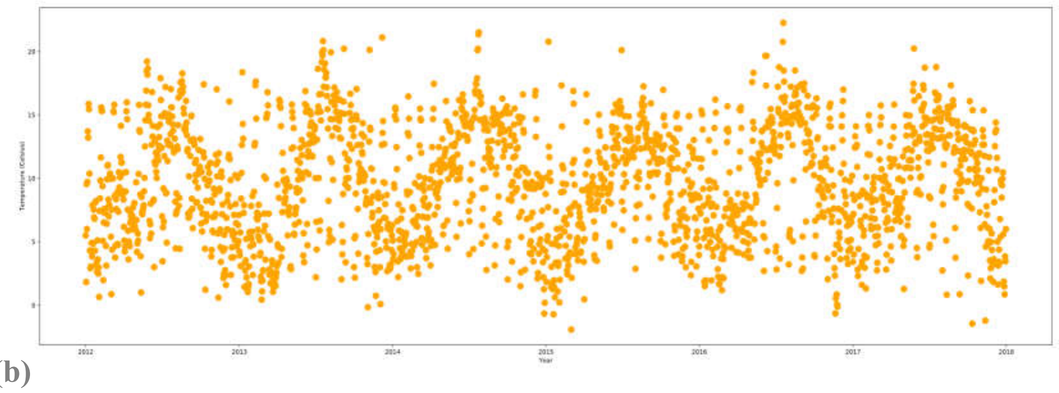
The main column of interest for model performance analysis was the 'Load' column. This is the megawatt (MW) load system demand of the Northern Ireland's power grid for a date and time. The column was used in analysis to assess a model's forecasting performance by comparing the column's value with a model's predicted load. The dataset was also augmented with columns that can be used as candidate explanatory variables in load forecasting regression models. Meteorological columns sourced from the MET office from one location in Northern Ireland (Aldergrove) are included in the dataset. Furthermore, the dataset includes columns derived from the dataset's 'Date' column. Further date derived variables not present in the dataset were added as part of data cleaning process. **See Appendix A** for a table containing all the columns of the dataset.

2.3 Load-Temperature Relationship

Studies on short term electricity load forecasting identify weather as being influential in deviations of load demand [7]. It has been observed that people's activity patterns change between abnormally warm or cold periods. Hence, temperature was chosen for analysis to assess whether it was an influential weather variable that has a strong linear relationship with load.



(a)



(b)

Figure 2.1 (a) *24-hour system load average 2012-2018* ***(b)*** *Temperature 24-hour average 2012-2018*

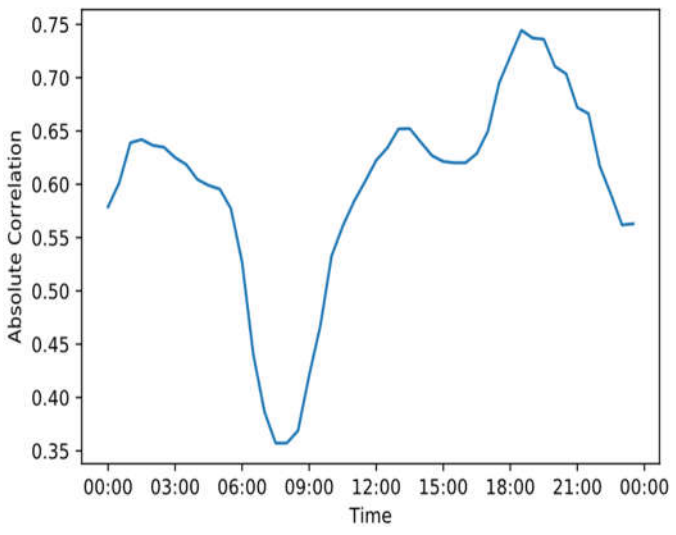


Figure 2.2 *Absolute correlation between load and temperature for each half hour interval.*

There is a sine wave like cyclic pattern in both the load and temperature graphs, **see Figure 2.1.** Load follows an inverse cycle to Temperature, with the troughs and rises in **(a)** visually contrasting **(b).** From the graphs, it can be determined the different seasons of the year influence the temperature each year, causing a cyclic pattern across the year. Therefore, an observation is that in warmer periods, less electricity load is used while in colder periods, more electricity load is used.

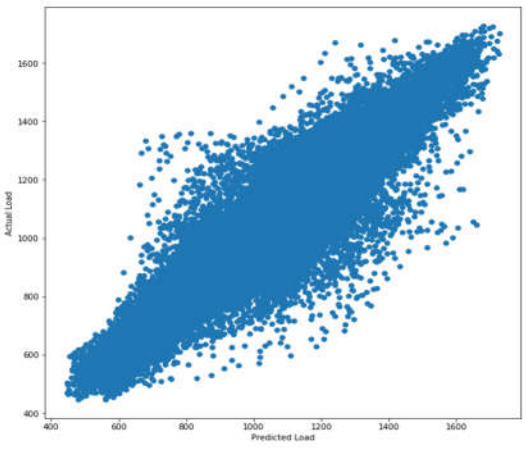
An investigation into whether temperature and load had a linear relationship in a more specific split of the data than yearly trends was undertaken. This was important to determine the value in building forecasting models which include a data entry's datetime characteristics for model input variables. The data was split by the time of the day, **see figure 2.2**. Pearson's correlation coefficient is a metric used to measure the strength of a positive or negative linear relationship between load and temperature. The highest correlation is in the evening and the lowest in the morning. This can be attributed to people being inactive at night and hence less reactive to changes in temperature. During day hours there is sunshine, and this this could be influencing the weaker temperature correlation with load in the day with photovoltaic electricity generation reducing the system demand.

2.4 Displacement Models

Different displacement models were considered for analysis: Last Day, Last Week and Last Year. The models were built on the dataset 2013-2018 to ensure a full range of displaced load entries were present in the data to use as the prediction load value. These approaches to creating a displacement model have their advantages and unique limitations. However, the displacement models all have the advantage of using the same time entry of the day, meaning the different pattern observed in the previous chapter specific to the time of day was considered in the load forecast.

Model	Correlation
SDLD	0.929
SDLW	0.962
SDLY	0.959

(a)



(b)

*Figure 2.3 (a) Correlation between actual load and predicted load for each displacement model **(b)** predicted load-actual load for SDLW Model using half hour load entries between 2013-2018*

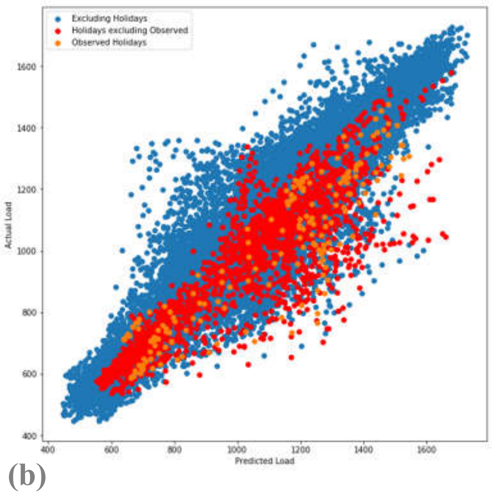
The model with the strongest linear relationship of its forecasted load output to the actual load entry is Same Day Last Week, and weakest is Same Day Last Day, **see Figure 2.3 (a)**. For building linear regression models in further analysis SDLW model forecasts will be included as a variable in the dataset. This will be for selection as an explanatory variable, and as a variable that can be corrected by other explanatory variables to produce more accurate load forecasts.

2.5 Holiday Days

Holiday days that have lower system demand load entries than normal days impact the load forecasting performance of displacement models. Taking advantage of Python, the '*holidays'* library was used to generate a list of all the holiday days in Northern Ireland. The list includes observed holiday days - days which follow a weekend the holiday occurred on.

Day type	**Correlation**
Excluding Holidays	0.968
Holidays excluding Observed	0.875
Observed Holidays	0.898

(a)



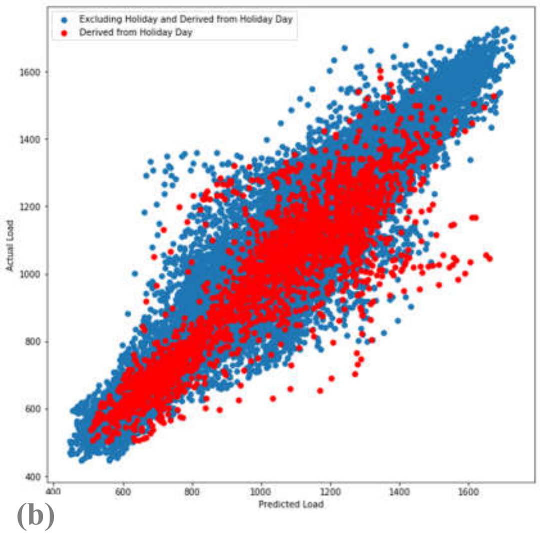
(b)

Figure 2.4 (a) Correlation between predicted load using the SDLW model and actual load for each day type (b) predicted load-actual load for SDLW Model using half hour load entries between 2013-2018

By mapping the 'date' column of the dataset with the generated list of Northern Ireland holiday dates, the dataset entries were labelled if they were a holiday and/or an observed holiday. The SDLW displacement model was used for performance analysis with holiday days. 'Excluding Holidays' days have the strongest positive correlation with actual load using the SDLW model forecasts, and 'Holiday excluding observed' holidays entries have the weakest, see **Figure 2.4 (a).** Visually in **Figure 2.4 (b)**, there is load over forecasting for holiday day entries with the model forecasted load having a larger MW value than the actual load. An explanation for this is that holiday days are not using holiday days to predict the load. This supports the assumption that people in Northern Ireland do not follow the same social pattern on holidays. There is a marginally stronger model performance for observed holiday days than holidays days, but they still underperform compared to excluding holiday day entries.

Day type	Correlation
Excluding Holiday and Derived from Holiday Day	0.970
Derived from Holiday Day	0.885

(a)



(b)

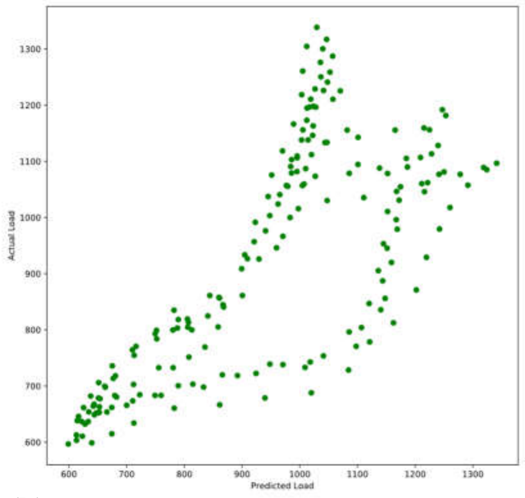
Figure 2.5 (a) *Correlation between predicted load using the SDLW model and actual load for each day type* ***(b)*** *predicted load-actual load for SDLW Model using half hour load entries between 2013-2018*

There are non-holidays entries that use holiday day load entries as their predicted load entry in the SDLW model. The performance of these entries was considered and are referred to as 'Derived from Holiday Day'. The 'Excluding Holiday and Derived from Holiday' days have a stronger linear relationship with actual load than excluding only holiday days, **see Figure 2.5 (a).** The derived from holiday day entries do not perform well in the SDLW model, with a weak linear relationship with actual load. Visually, there is a concentration of entries both under forecasting load and over forecasting load, **see (b).** The under forecasting of the actual load can be attributed to, as identified in the previous analysis, holiday days having abnormally low load entries for the period of the year [6].

The holiday library enables identification of which holiday the data entry is and which holiday entry the load entry is derived from. Therefore, the performance of the SDLW model for different individual holidays was compared.

Holiday	Correlation
Holiday New Year's Day	0.739, **0.833**
Derived Christmas Day	0.748, **0.788**
Holiday Easter Monday	0.755
Holiday Christmas Day	0.820, **0.941**
Holiday Boxing Day	0.841, **0.985**

(a)



(b)

Figure 2.6 (a) *Top 5 weakest actual load-predicted SDLW load correlated holidays. Correlation using SDLY model for Christmas period days is in bold – see* ***Appendix B*** *for full list.* ***(b)*** *Predicted load-actual load for SDLW Model for data entries on New Year's Day.*

The worst performing holiday using the SDLW model is New Year's Day, **see Figure 2.6 (a)**. Visually, the SDLW under forecasts the load on New Year's Day, **see (b)**. The Christmas period holiday days all have a weaker linear relationship with Load. The SDLW model for these days and days derived from these days does not produce an accurate load forecast. A correction strategy was created to create an accurate load forecast for these days.

The correction strategy chosen was using the SDLY (365 days) model for days derived from the Christmas holidays and the actual holiday days. A displacement of 365 days ensures it is the same holiday day in non-leap years and hence the holiday load pattern is considered. This improved the performance of all the Christmas period days **see (a).**

2.6 Linear Regression Models

Linear regression models are a popular choice for load forecasting [6]. Analysis in other research on the SONI dataset has observed accurate forecasts using linear regression models [6]. For analysis the Python machine learning library **scikit-learn** was used which uses the least squares fitting technique to fit the model. A full year of data was used as training data to train the model and the ascending years were used as the test data to evaluate the forecasting performance of the trained model. Mean absolute percentage error (MAPE) was the metric chosen to measure the prediction accuracy of the linear regression model forecasts. It is the most widely used metric in load forecasting [7].

Temperature Only Linear Regression Model

Temperature has been identified as having a linear relationship with load. A linear regression model was created that included 'Base Temperature' T_k as an explanatory variable to predict load $\hat{y}$:

$$\hat{y} = aT_k + b$$

Test	2012	2013	2014	2015	2016	2017	**Average**
Training							
2012	23.334	24.265	24.410	24.096	25.201	27.894	24.867
2013	23.402	24.077	24.145	23.906	24.744	27.000	24.546
2014	23.172	23.808	23.825	23.543	24.307	26.401	24.176
2015	23.082	23.705	23.700	23.402	24.137	26.165	24.032
2016	23.081	23.564	23.504	23.170	23.757	25.463	23.757
2017	23.239	23.584	23.464	23.046	23.435	24.653	23.570

TABLE 2.1 *The MAPE of the temperature only linear regression model fitted with a training year predicting the load of the test year. White cells: Year ahead load forecast performance. Blue cell (right): The average MAPE of the training year predicting test years.*

The average MAPE of the model forecasting a year ahead: 24.311%.

SDLW Linear Regression Model

In prior analysis the SDLW model forecasts have been identified as having the strongest linear relationship with Load. Hence, a linear regression model was created that included 'Load Last Week' $y_k - 7d$ as an explanatory variable to predict load $\hat{y}$:

$$\hat{y} = a(y_{k-7d}) + b$$

The average MAPE of the model forecasting a year ahead: 24.311%. - **see Appendix C** for a table of results. This is a considerable improvement in forecasting performance over the Temperature Only Linear Regression Model.

SDLW Temperature Corrected Linear Regression Model

There are forecasting errors in using only SDLW as a dependent variable because of the difference in temperature between the forecasted day and the day's load used for the prediction which influences load. A linear regression model was created that corrected the Load Last Week prediction $y_k - 7d$ with the displaced temperature difference $T_k - T_{k-7d}$:

$$\hat{y} = y_{k-7d} + a(T_k - T_{k-7d}) + b$$

The average MAPE of the model forecasting a year ahead: 4.183% - **see Appendix C** for a table of results.

SDLW Weather Corrected Linear Regression Model

Strong performing linear regression models in other research include a wider range of weather variables as explanatory variables than only temperature [7]. A linear regression model was constructed with variables: the displaced Temperature-48 hours difference $T_k - T_{k-9d}$ because of the prior identification of temperature influencing load; the product of sun duration and potential solar irradiance difference $s_k i_k c_k - s_{k-7d} i_{k-7d} c_{k-7d}$ and wind speed cubed difference $v_k^3 - v_{k-7d}^3$ because of sun and wind renewable energy power generation; humidity difference $h_k - h_{k-7d}$ because it has been identified in other research to have a relationship with load [8].

$$\begin{aligned}\hat{y} = y_{k-7d} + a(T_k - T_{k-9d}) + b(s_k i_k c_k - s_{k-7d} i_{k-7d} c_{k-7d}) + c(v_k^3 - v_{k-7d}^3) \\ + d(h_k - h_{k-7d}) + e\end{aligned}$$

The average MAPE of the model forecasting a year ahead: 4.005% - **see Appendix C** for results table.

As this was the strongest performing model of the linear regression models analysed, it will be incorporated into the system. The variables chosen for this model were arbitrary through prior analysis and the assumption that they provide a quantitative measure of an influencers to load demand. A less arbitrary method of choosing explanatory variables was investigated in the next chapter.

2.7 LASSO Regression Models

LASSO (least absolute shrinkage and selection operator) regression takes a large feature dataset and penalises the magnitude of the coefficients of the features, converging them to 0 and eliminating them from the model. Forward selection regression approach to variable selection has already been investigated in other research on the Northern Ireland load dataset [6]. Ridge regression was considered but it does not reduce model complexity. A ranked list of the most important variables to the load prediction can be produced using LASSO, enabling the creation of models using a specified number of variables that produces accurate load forecasts.

LASSO regression requires tuning model parameters. Alpha λ determines the weight given to the magnitude of coefficients when reducing the error by adding the residual sum of squares (linear regression error). To cross validate the alpha that produces a model with minimum MAPE, the models are trained on a random subset of the training year selected and tested on a random validation subset of training year. The alpha that produces forecasts with the smallest MAPE was then used as the alpha for feature selection.

All Explanatory Variables LASSO Regression Model

All explanatory variables in the dataset in **Appendix A** were included as candidate variables for LASSO regression feature selection.

The average MAPE of the model forecasting a year ahead: 9.349% - **see Appendix C** for results table.

SDLW Weather Correction Model LASSO Regression Model

SDLW Weather Corrected Linear Regression Model was the best performing linear regression model in prior analysis. Hence, building a LASSO model that chooses a subset of explanatory variables to correct the SDLW load forecasting prediction should produce more accurate forecasts. The variables used to construct the SDLW Weather Corrected Linear Regression Model were added to the dataset as candidate variable for LASSO feature selection.

Test	2012	2013	2014	2015	2016	2017	**Average**	**Variables**	**λ**
Training									
2012	3.598	3.601	3.559	3.748	4.122	5.601	4.038	23	1.41
2013	3.712	3.325	3.225	3.584	3.948	5.486	3.880	35	0.11
2014	3.711	3.369	3.121	3.459	3.698	5.065	3.737	39	0.06
2015	3.939	3.688	3.440	3.435	3.881	5.456	3.973	27	0.71
2016	4.202	3.995	3.689	3.932	3.584	4.842	4.041	42	0.06
2017	4.819	4.905	4.471	4.916	4.217	4.213	4.590	46	0.01

***TABLE 2.4** The MAPE of the LASSO regression model with variables selected and fitted with a training year predicting the load of a test year. White cells: Next year load forecast. Blue cells (RIGHT): The average MAPE of the training year predicting test years, the number of explanatory variables in the model and the alpha used for feature selection.*

The average MAPE of the model forecasting a next year's load using the previous year to train the model was 3.805%.

Variable	**Occurrence (%)**
Temperature	100
Temperature-12hrs	100
Temperature-96hrs	100
Temperature over 24hrs	33.33
Temperature over 36hrs	33.33
Sun duration*potential solar irradiance	33.33

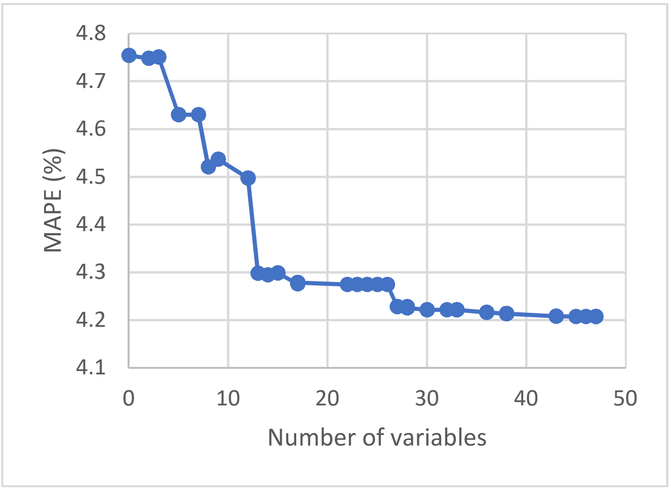
Table 2.2** The top and bottom 3 occurring explanatory variables selected by LASSO regression built on 2012-2017 training years – **see Appendix D for full list

Building a uniform linear regression model for the whole dataset based on the variables selected using LASSO regression for all models constructed, **see Table 2.2,** is not optimal for producing accurate load forecasts. It may exclude important variables that correlate to other variables which are eliminated early by LASSO feature selection Therefore, an approach was needed to determine the importance of the explanatory variables for producing accurate load forecasts for the whole dataset, not individual splits of the dataset.

Ranking Explanatory Variables using LASSO

A list of explanatory variables ranked by the order of elimination from LASSO regression by increasing the alpha incrementally was used to evaluate their 'importance' to creating a linear regression model that produces accurate load forecasts with n-features. The whole dataset was used to fit the model. Several iterations of the process were manually observed and the increments for increasing the alpha were decreased at lower alpha values (<1) to record atomic ranks for variables of least importance

Variable	Alpha
Humidity Difference	6000
Cloud height	500
Wind direction	200
Potential solar irradiance	0.05
Temperature over 18hrs	0.045
Temperature over 96hrs	0.03



(a) **(b)**

Figure 2.7 (a) The top 5 most important (elimination alpha is highest) and least important (elimination alpha is smallest) explanatory variables - **See appendix D** for the full list **(b)** The MAPE performance of the number of explanatory variables selected for the lasso regression model

Visually in **Figure 2.7 (a)** the MAPE decreases unsteadily as the number of variables increases. The MAPE improvements plateau after 12 variables are selected. **In (b),** the variables eliminated first by LASSO feature selection are temperature related which is to be expected as they are highly correlated to other temperature variables. 2nd placed ranked variable 'Cloud height', unlike 1st 'Humidity Difference', was not recognised in prior analysis as being an important explanatory variable. This indicates that there are limitations to using LASSO to minimize MAPE errors, as LASSO minimizes the mean squared error (MSE).

<u>SDLW Lasso Variable Selection Correction Model</u>

A linear regression model was created with the 12 top ranked explanatory variables. The following model was created with the 12 most important variables: Humidity Difference $(h_k - h_{k-7d})$, Cloud height (ch_k), Wind speed^2 (v_k^2), Wind direction (vd_k), Dayofyear (doy_k), Wind speed^3 (v_k^3), Temperature Difference $(T_k - T_{k-7d})$, Day (day_k), Temp-48 Difference $(T_k - T_{k-9d})$, Holiday_Alternate (hol_k), Yearly cycle (sine wave) (yc_k), Wind speed Difference $(v_k^3 - v_{k-7d}^3)$:

$$\begin{aligned}\hat{y} = y_{k-7d} + a(h_k - h_{k-7d}) + b(ch_k) + c(v_k^2) + d(vd_k) + e(doy_k) + f(v_k^3) \\ + g(T_k - T_{k-7d}) + h(day_k) + i(T_k - T_{k-9d}) + j(hol_k) + k(yc_k) \\ + l(v_k^3 - v_{k-7d}^3) + m\end{aligned}$$

The average MAPE of the model forecasting a year ahead: 3.859% - **see Appendix C** for results table. This is higher than the SDLW Weather Correction LASSO Regression Model by 0.054%. However, the SDLW Weather Correction Model uses a maximum of 46 input variables, while the LASSO model only uses 12 variables. The LASSO Variable Selection SDLW Correction Model provides a good balance between model complexity and forecasting performance.

2.8 System Decisions Made

The investigative processes undertaken in analysis is the functionality that the software solution should automate. The system should provide the flexibility to change the forecasting models' parameters and the split of the dataset on which they are built and tested on. The visualisations of the model's performance present in analysis should be generated on the fly by the system and not require an understanding of Python visualisation libraries. Models in the system should be able to be compared statistically with other models using the MAPE metric. The date characteristics of day entries should be provided to the user as an options to highlight them to notice load deviation trends types of day have compared to other days. The regression models correcting a displacement model's predictions should be included in the solution as these models have the strongest forecasting performance in analysis. The analysis benefited both the understanding of the author in realising the challenges of thoroughly analysing the data and characteristics of the data.

3 Solution Description and System Requirements

3.1 Solution Description

The aim of the project is to create a software solution that provides to a user an interactive data visualisation suite to support research in the area of short-term electricity load forecasting. Fundamentally, the manual processes of analysis detailed in **chapter 2** will be automated in the solution for the user to perform investigate analysis on the load dataset. Users will be able to discover trends in an historical load dataset through dynamic visualisations of system load and related variables. Load forecasting models that have been proven in analysis to produce accurate load forecasts for the dataset will be incorporated into the project. Users will be able to systematically compare the performance of different models statistically and visually.

This investigative analysis that will be facilitated by the solution would only otherwise be available by using a specialist computing environment. These environments commonly require commercial licences and an assured programming background to be able to achieve basic data analysis tasks. The solution will be a free open source alternative that allows for complex tasks to be performed through a functional user interface. The solution will be implementation agnostic and will only require a basic understanding of load forecasting.

3.2 Academic User

Following a meeting with the project's supervisor, referred to as the 'stakeholder', an academic user was identified as the target user for the system.

Motivation for using the solution:

Academic users will use the solution as an introductory tool to develop a foundational understanding of the challenges of forecasting load and evaluate and compare different load forecasting models' performances.

What would they do if this solution didn't exist:

Academic users can use data analysis and visualisation environments such as MATLAB and Jupyter Notebook to:

- arrange and clean their load dataset
- create load forecasting model functions
- evaluate model performance visually and statistically.

Value gained by using the solution:

The solution enables the academic user to automate the complex mathematical functionality required for load forecasting that they would have otherwise had to create manually in a specialist data analysis environments.

3.3 User Flows

The expected flow of an academic user using the system to perform functionality:

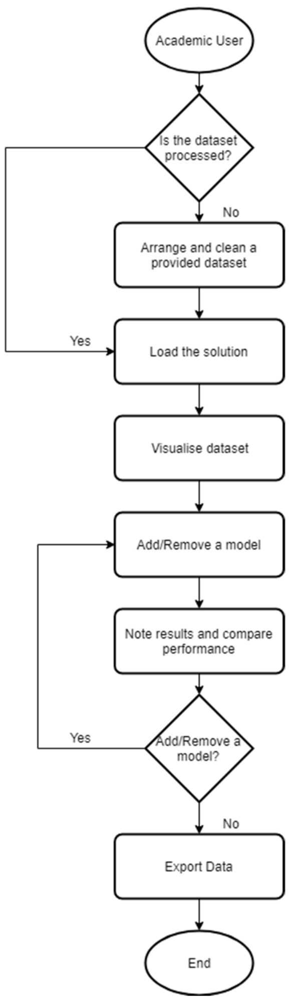


***Figure 3.1** Flow chart visualising the flow of an academic user using the system.*

3.4 Use Cases

A specification was derived from the use cases of the academic user. The specification is written from the viewpoint of the user.

Academic User Use Cases:

- Arrange and clean a provided dataset through an automated step-by-step process.
- Dynamically visualise a dataset with load values to gain an appreciation of the dataset's historical trends.

- Use known good performing load forecasting models to compare their performance over historical date ranges.
- Add load forecasting models with by constructing experimental models identified to be perform optimally in their own independent analysis.
- Export visualisations and statistical values to include in an analysis paper

A use case grid was created to augment attributes to the use cases identified. This approach is derivative of the approach in [9]:

Use Case ID	Use Case Name	Complexity	Priority
1	Arrange and clean a provided dataset	Low	High
2	Visualise dataset	Medium	High
3	Add models	High	High
4	Compare model performance	High	High
5	Export data	Medium	Medium

Table 3.1 *Use case grid. Complexity: the perception of the difficulty in implementing the use case functionality to the solution. Priority: priority of delivering the feature to the primary actor.*

3.5 Non-Functional Requirements

From the use cases the following non-functional requirement that describe the system behaviour and characteristics were identified as being important to the academic user:

Usability

- Functionality is prioritised over accessibility. Users must be able achieve their desired functionality with a minimal number of interactions.
- Each user interface control must have a descriptive label or tool tip describing its functional purpose.
- Constructing a forecasting model must be transparent to the user with details of the model structure visible.

Responsiveness

- User must be able to interact with the system without encountering an extensive processing phase or a page refresh.

Quality

- Users must consider the results produced by the solution comparable to commercial offerings, being able to use them in their academic works.

Extensibility

- Users must be able to use any load dataset with the system to perform the same operations as with the SONI dataset.
- Users must be able to engage with other forecasting models implemented in the future without any noticeable change in method of interaction in constructing them.

Robustness

- The system must not crash unexpectedly.

3.6 Functional Requirements

The functional requirements of the system are written as minimum viable features (MVFs) derived from the use cases in **Table 3.1.** MVFs are 'minimally viable' as they provide value to a stage in the user flow. The analysis work performed on the SONI dataset determined which proposed features can leverage the dataset and which cannot be implemented because of the limitations of the dataset. MVFs include user stories that fulfil an atomic part of the feature and relate directly to implementation work. User stories are prioritised to ensure development time is focused on delivering high priority functionality to the target user. A single feature is developed independently and the progress of it and its associated user stories is visualised using a virtual Kanban board. The IDs of both are included in the virtual Kanban board to relate to this documentation.

The format the features are listed in is as follows:

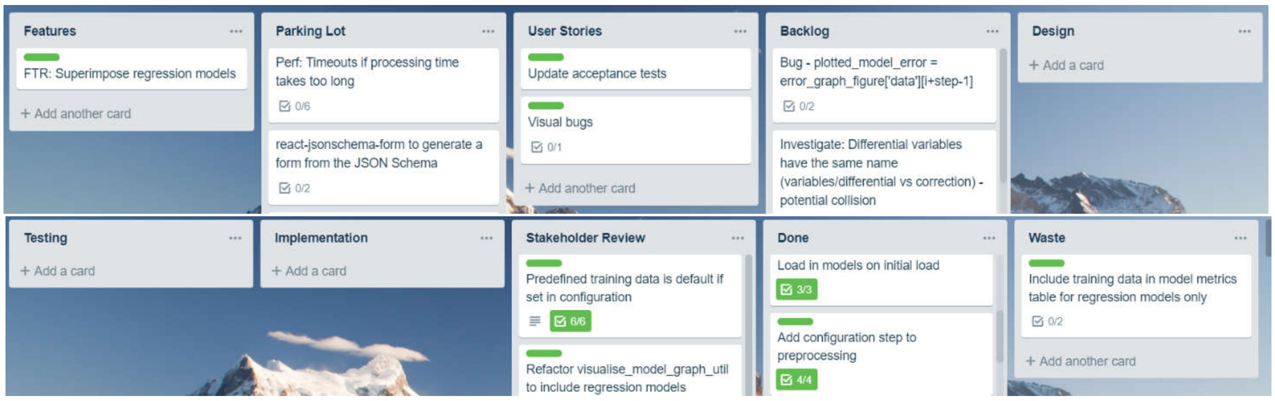
Feature ID	Feature Name			
User Story ID	User Story Complexity	User Story Priority	A high-level user action	Status – done, backlog or parked

Feature List

A more complete list of user stories with defined acceptance criteria is in **Appendix E.**

FTR00	**Visualise and interact with the historical load data**		
US00	High	Open the application in a web browser	Done
US01	High	View a graph visualising historical load data	Done
US02	High	Interact with the date range visualised.	Done
FTR01	**Evaluate the forecasting performance of a displacement model**		
US03	High	Visualise displacement model's load forecasting performance	Done
US04	High	View forecasting performance metrics of a visualised displaced model	Done
FTR02	**Compare multiple forecasting models' performance**		
US05	High	Compare two displaced model's forecasting performance	Done
US06	High	View two graphs with different y axis over the same time range to compare characteristics contributing to differences in model performance	Done
FTR03	**Advanced Visualisation and Interaction**		
US07	Medium	Highlight different types of days	Done
US08	High	Visualise the error distribution of a forecasting model	Done
US09	Low	Choose how the graph is presented	Parked
FTR04	**Assess the forecasting performance of linear regression models**		
US10	High	Visualise a linear regression forecasting model to evaluate its performance	Done
FTR05	**Export the visualisations and statistical metrics**		
US11	High	Export the visualised graphs	Backlog
US12	High	Export the metrics table statistical data	Backlog

3.7 AGILE Development Process



Features
FTR: Superimpose regression models
+ Add another card

Parking Lot
Perf: Timeouts if processing time takes too long
0/6
react-jsonschema-form to generate a form from the JSON Schema
0/2

User Stories
Update acceptance tests
Visual bugs
0/1
+ Add another card

Backlog
Bug - plotted_model_error = error_graph_figure['data'][i+step-1]
0/2
Investigate: Differential variables have the same name (variables/differential vs correction) - potential collision

Design
+ Add a card

Testing
+ Add a card

Implementation
+ Add a card

Stakeholder Review
Predefined training data is default if set in configuration
6/6
Refactor visualise_model_graph_util to include regression models

Done
Load in models on initial load
3/3
Add configuration step to preprocessing
4/4

Waste
Include training data in model metrics table for regression models only
0/2
+ Add another card

Figure 3.2 The virtual Kanban board on Trello used as the process tool for system development

Definition of each stage:

- *Features: List of MVFs currently being developed with a colour code (green) that links stories with the feature.*
- *Parking Lot: User stories that are blocked by planned user stories – unable to be picked up for development as they*
- *User Stories: Stories that are planned to be developed on to complete the feature*
- *Backlog: Stories/Tasks not associated with a feature that are improvements and minor bug fixes.*
- *Design: Stage in which the user story is, if possible, broken into smaller tasks and the acceptance criteria is defined for acceptance testing. The implementation details are added to the tasks after an investigation of the solution.*
- *Testing: The testing defined in the acceptance criteria occurs and new test cases are added to the manual acceptance tests documentation.*
- *Implementation: The stories and tasks currently being worked on*
- *Stakeholder Review: The stories and associated tasks that have yet to be demonstrated to the stakeholder for verification they meet their expectations.*
- *Done: Stories and associated tasks that are fully tested and passes the stakeholder's evaluation.*

The system was developed using an agile approach [10]. Trello [11] was used as the main process tool as a virtual Kanban board, **see Figure 3.2.** This tracked the progression of MVFs and associated tasks. They were added to the board and their progression from initial analysis and completion (done) was tracked. The limit (WIP – work in progress) of tasks/stories allowed to be in the design, testing and implementation stages was one, which ensured that work was delivered as fast as possible with stories and tasks' taking longer to complete being flagged. There were no time boxed sprints, instead rolling development of MVFs was on a development (feature) code branch. Once a feature was completed it was promoted to the deployable master code branch after stakeholder verification and testing.

3.8 Work Plan

A general work plan was maintained to to keep track of the deadlines for the system and base the cadence of development on completing features towards them.

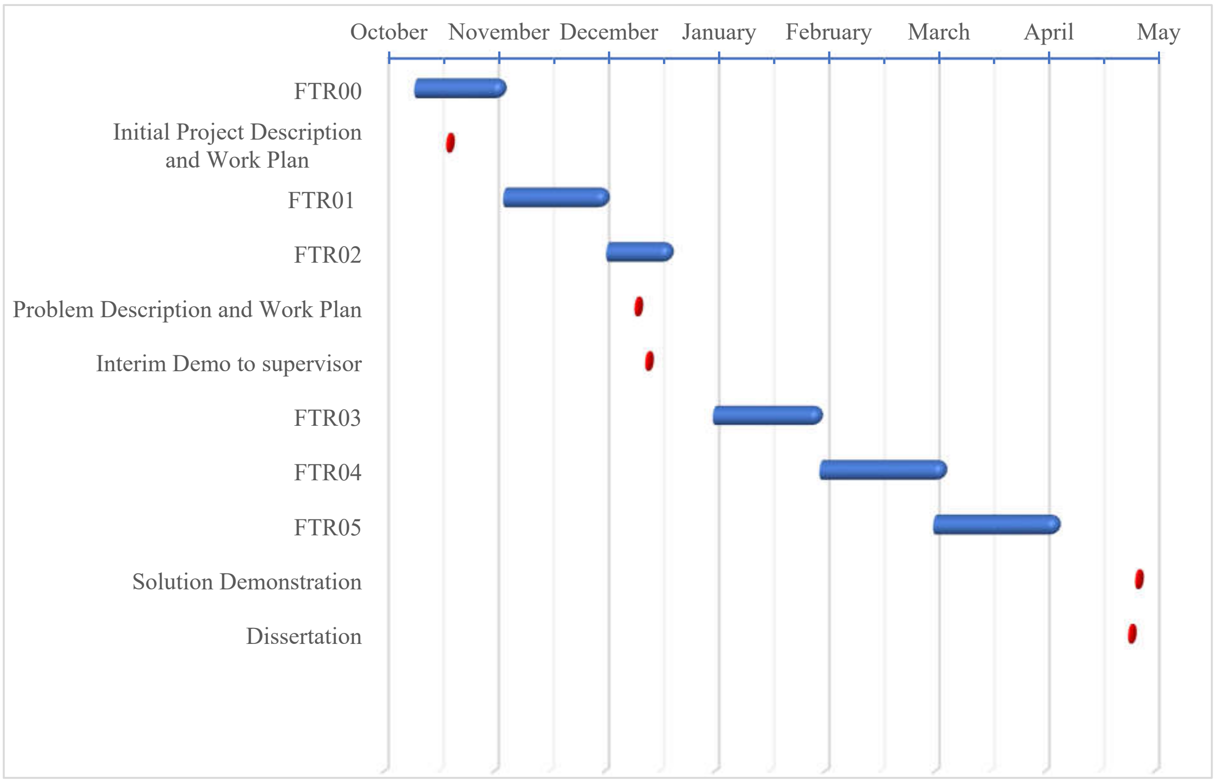


***Figure 3.3** A chart of functional features (blue) and project deadlines (red):*

3.9 System Constraints

There are constraints identified that influence the design of the solution:

- There is a requirement by the providers of the dataset SONI to handle the load dataset in a confidential manner as it not publicly available and has commercial value in being used to generate load forecasts. The design must incorporate a preventative control to mitigate the scraping of data.
- Data manipulation functions that can be done before processing should be pre-processed. With multiple potential clients of the solution this would cause unnecessary processing on the server end.

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

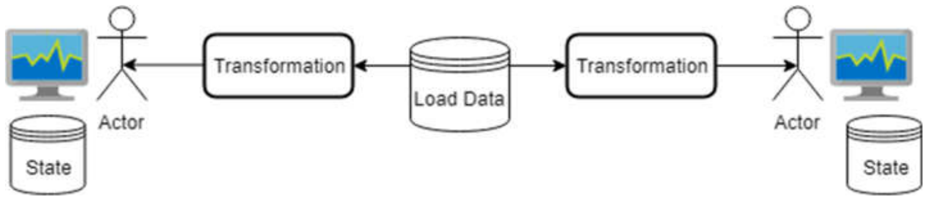


Figure 4.1 Data centred architecture [12]. The process of the system returning different transformations of data to multiple actors concurrently.

The fundamental process the system performs is transforming the load dataset, performing logic on that transformation, and producing a result 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 [13]. The dataset must also be immutable with different transformations returned determined by the users' state as input, **see Figure 4.1**.

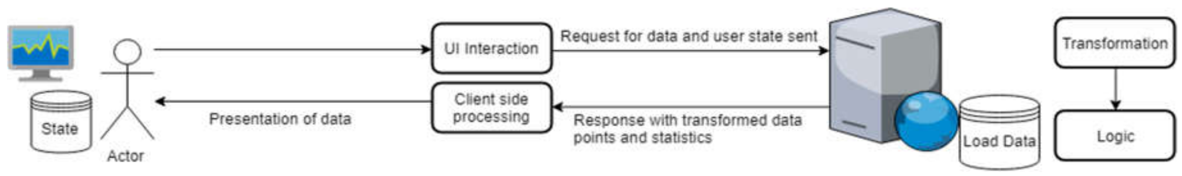


Figure 4.2 *The process of a user interacting with a web server and being returned a transformation of the data and statistics that is presented onto their client*

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 [14], **see Figure 4.2**.

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 in the README of the system [1].

Preprocessing

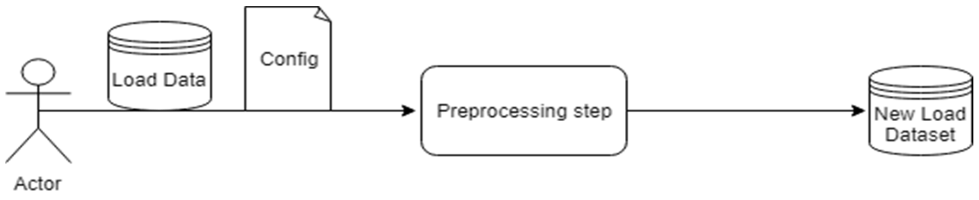


Figure 4.3 *The process of preprocessing of a load dataset, augmenting it with required model variables for highlighting and model construction functionality.*

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 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 Considerations

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 F**.

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 presents a challenge as they have less screen real estate and are less computationally capable than a desktop computer [15]. They also can rely on poor network conditions e.g. mobile internet which can be considerably slower than broadband internet [16]. 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:

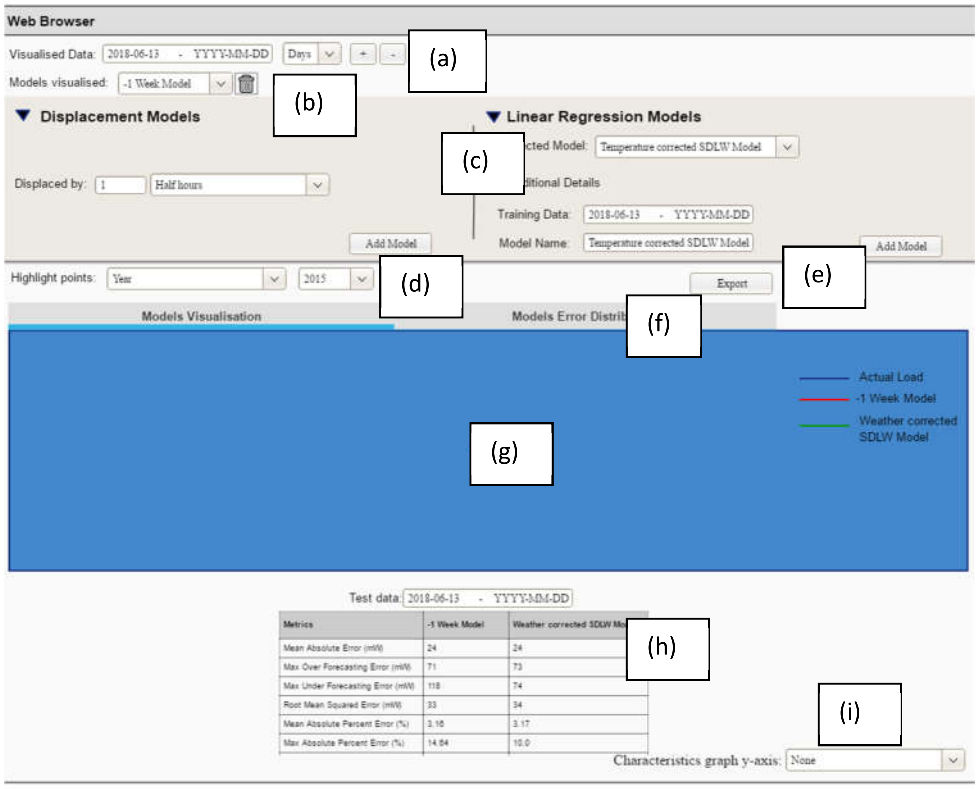
- The styling will be minimalist. This approach is used by the likes of Google to ensure their pages load reasonably quickly on slow connections [16].
- User interface components, visualisations and text will scale to the user's device screen size.
- Landscape will be chosen as the default screen orientation for user interface designs as greater horizontal space is more effective in visualising electricity load trends over time.

4.4 Academic User Interface Prototype

A prototype of the proposed user interface was created using *Pencil* [17], **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.

<u>User Interface</u>



Visualised Data: 2018-06-13 - YYYY-MM-DD Days

Models visualised: -1 Week Model

Displaced by: 1 Half hours

cted Model: Temperature corrected SDLW Model

ditional Details

Training Data: 2018-06-13 - YYYY-MM-DD

Model Name: Temperature corrected SDLW Model

Highlight points: Year 2015

Test data: 2018-06-13 - YYYY-MM-DD

Metrics	-1 Week Model	Weather corrected SDLW Mo
Mean Absolute Error (mW)	24	24
Max Over Forecasting Error (mW)	71	73
Max Under Forecasting Error (mW)	118	74
Root Mean Squared Error (mW)	33	34
Mean Absolute Percent Error (%)	3.16	3.17
Max Absolute Percent Error (%)	14.64	10.0

Characteristics graph y-axis: None

Figure 4.4 *Prototype of the academic user's user interface.*

Annotated descriptions of components in **Figure 4.4** are listed below:

<u>*(a) Graph visualisation controls*</u>

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

<u>*(b) List of models the user has added and can remove*</u>

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

<u>(C) LOAD FORECASTING MODEL CONSTRUCTION</u>

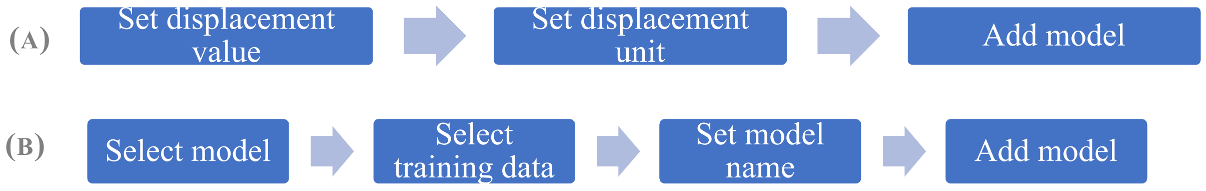


Figure 4.5 *The steps a user construct a load forecasting model using the system (a) Displacement model (b) Linear Regression model*

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

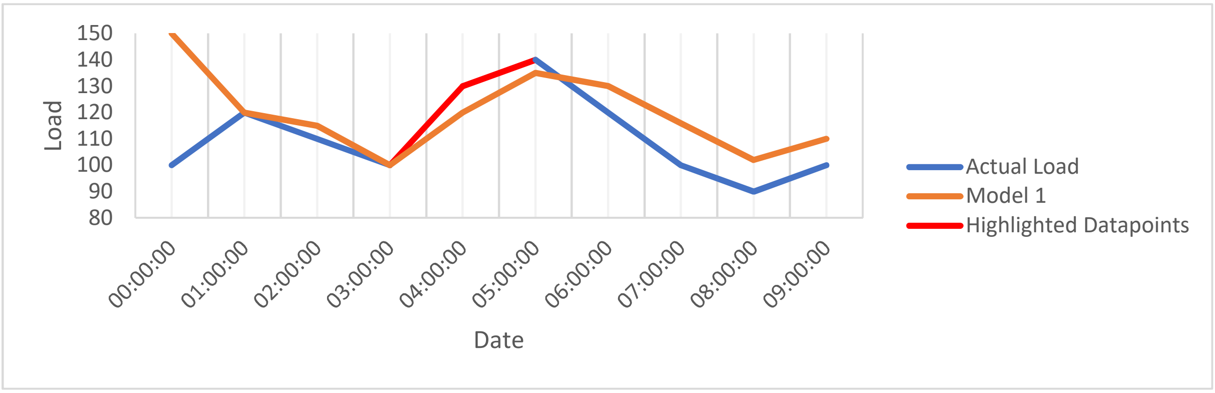


***Figure 4.6** Load visualisation graph mock-up*

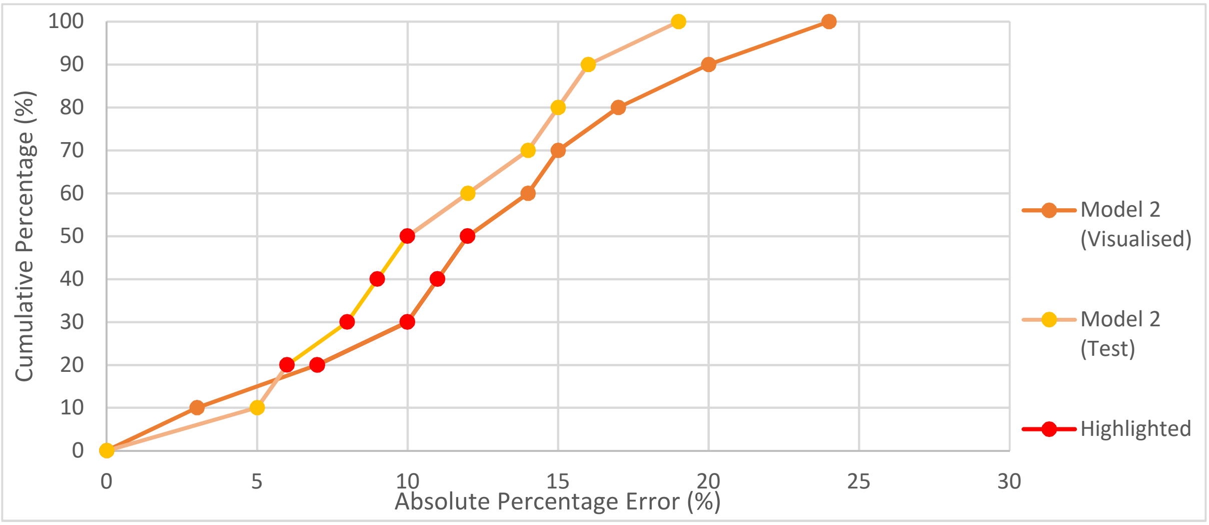


***Figure 4.7** Error distribution graph mock-up.*

The choice of colours for different plots in the visualisations is to distinguish different information [18]. 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 [18]. 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 [18]. The highlighted data points can be used to provide an explanation for high errors.

(h) Model Performance Evaluation Table

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 (%)		

Table 4.1 *Model metrics table mock-up*

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.

<u>(i) Characteristics comparison</u>

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

5 Implementation

5.1 Software package choices

Implementation Language

Python was the chosen implementation language. Python is open source and has a focus on readability, with a low learning curve for those new to the language [19]. This enables developers who want to extend the system functionality to have access to, and efficiently understand the system's code base. Prior analysis work performed in **chapter 2** used Python libraries to develop functionality that the system is to automate. The associated libraries and the functionality they provide were integrated into the solution. The main libraries used were:

- **Pandas** for dataset manipulation
- **Scikit-learn** for linear regression forecasting model construction
- **Holidays** for holiday processing
- **Numpy** for advanced mathematical functions e.g. calculating model performance metrics

System Dependency Installation

Anaconda was chosen to organise the required Python library dependencies for the system. Anaconda is especially suitable for users performing data science. This is because if working with different analysis tools, users may have to install requirements that conflict with each other [20]. A user can create a virtual environment isolated from the Python versions and dependencies installed in the user's system from a list of requirements (environment.yml) specific to the system. This is then be used to run the solution with no external conflicts

Data Visualisation Library

Matplotlib was used in **chapter 2** to produce static SVG visualisations. However, the system is required to produce dynamic visualisations that update with the user input in **FTR00**. Python data visualisation library **Plotly** was chosen as it produces dynamic graphical visualisations.

5.2 Dash Framework

Overview

The Python framework **Dash** was chosen to create an open source interactive dashboard that hosts Plotly graphs and enables user interaction with the graph. Dash is supported by all modern browsers and is mobile ready which meets the requirement of the system to be cross platform

[21]. The implementation of the solution was based heavily on using the Dash framework to meet the requirements of the target users. Dash is built upon the Python web framework **Flask**. Flask is widely supported by web deployment tools, meeting the requirement of the system having the ability to be hosted on the internet. Dash provides components libraries that are useful for creating controls to enable user interaction with visualisations. The dash-html-components library provides web html components. Hence, there was no need to have knowledge of additional web technologies and protocols as these are abstracted by Dash. Dash met the non-functional requirement of all functionality being presented to the user by rendering the dashboard as a single-page application.

5.3 Dash Lifecycle

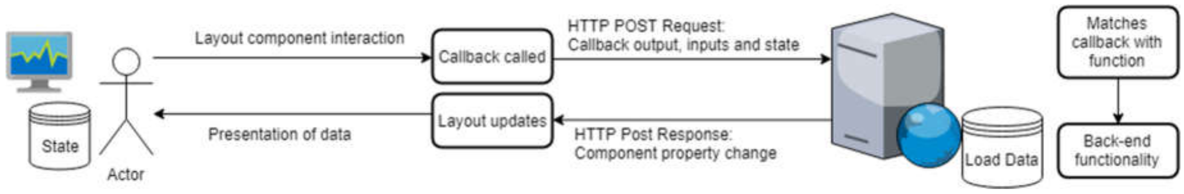


Figure 5.2 The Dash components working together to make the system responsive to the user's interaction.

The system processes described in **Chapter 4** were facilitated using the Dash framework, **see Figure 5.2.** Callbacks are called when the properties of components as inputs to the callback are changed through user interaction. The return value of the callback is then a new property of the component that is the output of the callback. The entry point of the system is the Dash start-up process, which brings together all the components of the dash framework to produce an interactive data visualisation dashboard, **see Figure 5.3.**

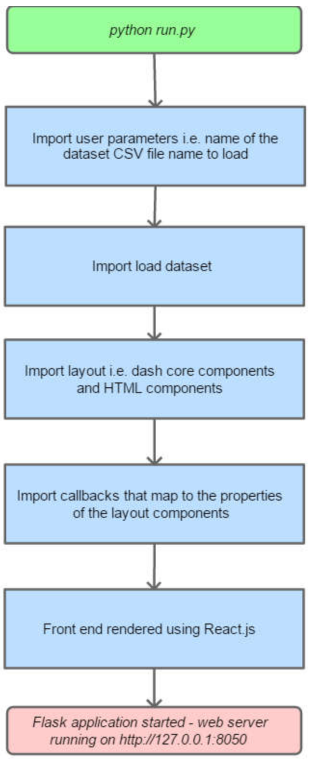

python run.py
Import user parameters i.e. name of the dataset CSV file name to load
Import load dataset
Import layout i.e. dash core components and HTML components
Import callbacks that map to the properties of the layout components
Front end rendered using React.js
Flask application started - web server running on http://127.0.0.1:8050

Figure 5.3 The start-up processes of the system.

5.4 Layout Implementation Decisions

The layout of the system was defined in the *layout.py* file located in the root of the system's code base.

Naming Convention

There is a naming convention for the elements in the layout. It is important to have a consistent naming convention for a project [22]. There is an additional importance to the system as the ids in the layout relate directly to callbacks. Any inconsistency in naming could cause errors with interaction not producing expected results. The ids of the components and stylesheet classes names are named using kebab case, which replaces spaces with hyphens and is all lower case e.g. variable-id-1. All components are appended with a general description of their purpose in the layout e.g. all dropdown components had '_selection' appended at the end.

<u>Component Styling</u>

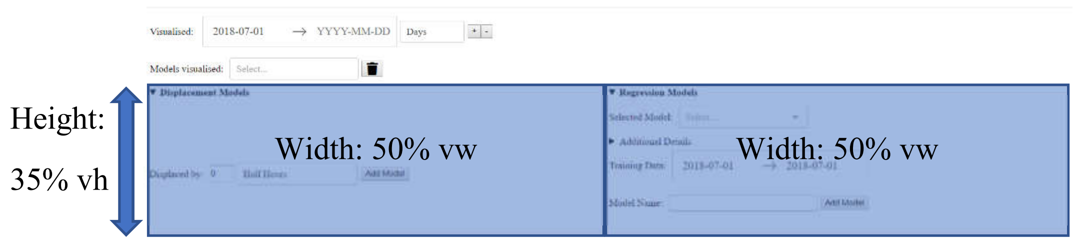


Figure 5.4 *Example of two flex items contained in a single flex box container. The model construction user interface elements for displacement and linear regression models can shrink and grow to match the constraints stated. vw – viewport width. vh – viewport height*

The system was required to be scalable to the user defined window size in **chapter 3.** To meet this requirement components are styled to be contained within flexbox containers that shrink and grow the components to the available window space. Furthermore, instead of using hard coded pixel sizes for the flex box container the height and width of the viewport (browser view) are used to make the layout scalable to the user's defined window size. For an example of a flexbox container implemented in the system's layout, **see Figure 5.4.**

<u>Dash HTML Components</u>

The following html components are implemented in the layout of the system:

- **divs** are used to separate different interactive components and contain the plotly graphical visualisations and model error statistics table.
- **button** is used for user input confirmation e.g. add a model
- **Details** and **Summary** is used for hiding additional details and functionality not required to fulfilling the user's user flow upon initial load, a non-functional requirement of the system. They can be opened with a mouse click of the bold label with a carat.
- **H4** are used for labelling distinct sections of the system to make the system more accessible to the user.

<u>Dash Core Components</u>

The following core components were implemented in the layout of the system:

- **DatePickerRange** are used for setting the start date and end date of data which is constrained by the date range of the load dataset.
- **Dropdown** are used to list options for the user to toggle and use as parameters for visualisations and model construction.
- **Tabs** are used to separate the different visualisations, conforming to the tabbed design user interface requirement for the system.
- **Input** is used for user customisable parameters.

5.5 Back-end Implementation Decisions

The functionality that produced the output results to callbacks for presentation in the user interface components is in the *component* folder. The preprocessing functionality is in the preprocessing folder. Functionality for handling model state in the *state* folder. The naming convention for variables in these files was snake case, which follows a best practice outlined in PE 8 [22]. There were several key implementation decisions made for the system:

Graph visualisation implementation

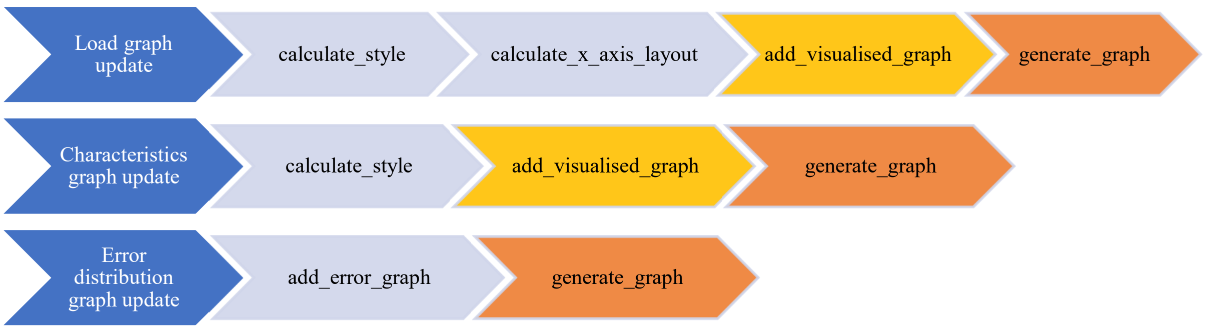


***Figure 5.5** The function calls made for each different type of graph in the system.*

All graph visualisations in the solution have an associated callback in *graph_callbacks.py*. The callback is called when one of the input components that change the visualisation of graph changes. The input values are passed onto the update function. Each graph has different functionality in the update function, **see Figure 5.5**. There is shared functionality between all the graphs that is contained within *graph_util.py*, which contains functions that all graph visualisations need to call e.g. generating data point traces, generating highlighted data points and creating the Plotly graph object. There is different functionality with add_error_graph being called to update the error distribution graph, which generates a graph with error data point traces. The load graph and characteristics graph both share the same date x-axis but with different layout styles and different y-axis values.

Object-Oriented Programming

An object-oriented programming (OOP) design to wrap the functionality and data associated with creating a graph inside graph object was considered [23]. A graph class would have behaviour which can be shared by all inherited subclasses (load, characteristics and error distribution graphs). Graph objects (instances of a graph subclass) in the system would be constructed with the input data to define their instance variables, and methods would be called to perform the data manipulation necessary to generate the visualised data points. This would enforce a contract of using reusable functions rather than the calling the utility functions in the current system.

A pure graph update function that when given the same input will always return the same result was determined to be the best solution. This is a procedure-oriented way of programming [23]. Using OOP, the data contained in the graph object cannot be stored indefinitely as the system is stateless, thus there is limited benefit to using a graph object. Furthermore, the result of the graph update function for the layout is a python dictionary encapsulating a Plotly ScatterGL object and Graph Layout object. To adapt this to follow an OOP pattern would require accessing the data in the graph object to create these structures. Therefore, OOP creates unneeded extra level of abstraction that requires additional code.

Visualisation Performance Considerations

There were performance considerations made when deciding which implementation of the Plotly graph objects to visualise the graphs. Plotly **Scatter** graph component was initially used but took an extensive time to load the visualisations, **see Table 5.1.** It was also sluggish when interacting with a high density of visualised load data points. The **Plotly ScatterGL** graph component was chosen instead. The component is built upon WebGL which is supported by all major web browsers. It is more performant in rendering graphs with larger density of plots which is beneficial for the system as the SONI electronic load dataset contains over 140,000 rows.

Browser	SVG	ScatterGL
Chrome	28.6	2.33
Internet Explorer	38.96	4.46
Firefox	65.34	2.28

Table 5.1 *List of times taken from initial system load to visualisation of load from 2017 to 2018 using different web browsers.*

Error Distribution Graph Implementation

```
List of error traces to plot = empty
x = the load visualisation graph
iterate through all the model plots in x:
        plotted model = the x,y model data points
        calculate the APE between the actual load and the plotted model forecasted load of the model for
        the same data entry and record this
        sort all the data entries by the APE ascending
        give every data entry an increasing index
        calculate the cumulative percentage by dividing the entry index by the number of datapoints * 100
        and record this
        set name visible on the legend as the name of plotted model with (Visualised) appended
        set colour of model as the colour of the plotted model
        generate x,y error traces using the APE and cumulative percentage
        add model error traces to the list of error traces
```

Figure 5.6 *Pseudocode representation of the process transforming the data points of models forecasts into error data points for the error visualisation graph.*

The error distribution graph in the design had the APE of the visualised data points on the x-axis and the cumulative percentage on the y-axis which requires specific processing to generate the required values. In **Figure 5.6** the processing is optimized by using the load visualisation graph plotted forecasting model data as the load forecast to calculate the Absolute Percentage Error (APE) statistic with actual load. This optimization means that the error visualisation graph does not need to recalculate load forecasts used to generate the load visualisation graph data points. The data of the load error visualisation graph is an input to the callback function that updates the error visualisation graph, thus the models visualised in the error visualisation graph is updated synchronously. A consequence of this is that rendering the load visualisation graph is a prerequisite of rendering the load visualisation graph. This needlessly increases processing time for the user if they only want to visualise one of the graphs.

Forecasting Models Implementation

The processes to create and evaluate forecasting models were adapted from functions written in the analysis. The model functionality is contained in the *models* folder with subfolders *displacement* and *regression*. The entry point for both models to generate load forecasts is the *predict* function. For the displacement models an approach was considered to match each entry in the dataset indexed by date time with the load value at the displaced date index. This approach operating sequentially on scalars would have had at worst case O(N) time complexity. Instead, a more performant approach that took advantage of Pandas data structures being built on arrays that can take advantage of vectorized functions. The load values of data entries to be forecasted were set as the entire array of the displaced load entries.

The approach for constructing the linear regression forecasting models used the Scikit-learn library for training the model coefficients and intercept using the *fit_to_training* function. The trained coefficients and intercept are used with the input x variables to create forecast predictions using the *predict* function. The process for creating linear regression models that correct the displacement forecast is different from linear regression models with only explanatory variables and associated coefficients. Instead of training the model with the actual load forecast as the dependent variable, the dependent variable is the sum of a displacement model forecast subtracted by the actual load forecast. The load forecast is the displacement model forecasts subtracted by the model output.

<u>Forecasting Model State Implementation</u>

The stateless design of the system was proposed in **chapter 4**, with the users list of models added being stored client-side. The implementation approach to this was to use a hidden DIV in the layout that stores a JSON representation of the list of models added. This div contains fields required to create load forecasts using the model.

In the code the displacement and linear regression models in back-end functionality are loaded in the JSON structure and are then parsed to create a Python dictionary. Separate DIVs were not used because it would require merging two data structures to display the list of models and remove a model. The different functions required to be called to create forecasts for a model for are determined through the model type of the entry in the dictionary. The coefficients and intercept trained on the training data for a linear regression are included instead of the training data range. This was to reduce processing time by not retraining the model every time the linear regression model forecasts are plotted.

Preprocessing Dataset Implementation

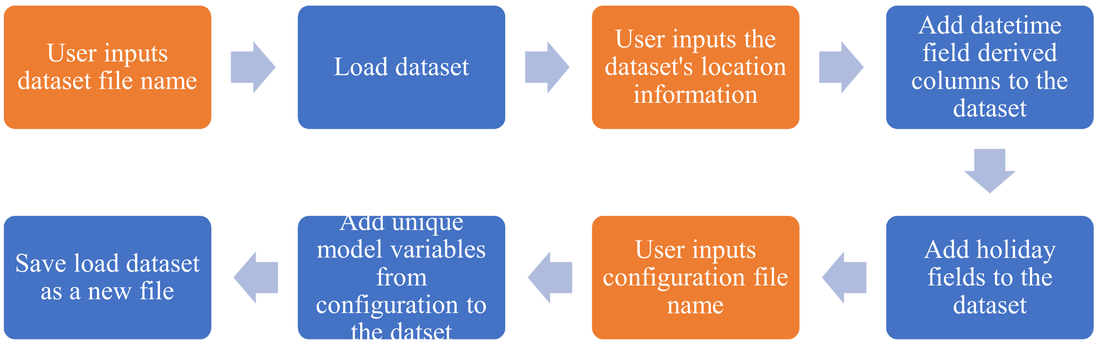

User inputs dataset file name
Load dataset
User inputs the dataset's location information
Add datetime field derived columns to the dataset
Add holiday fields to the dataset
User inputs configuration file name
Add unique model variables from configuration to the datset
Save load dataset as a new file

Figure 5.7 The process for preprocessing a dataset in the process_csv.py file.

The entry point to the preprocessing functionality is by executing the *process_csv.py* file, which accepts console input, **see Figure 5.7.** There is no error handling for erroneous user input in the preprocessing program because the original dataset is not modified. The model configuration file was written in JSON. JSON was chosen as it was designed to represent data structures (models) and associative lists of objects (model variables) in JavaScript [24]. Furthermore, the JSONSchema Python library enables the parsing of a JSON file to a Python dictionary structure.

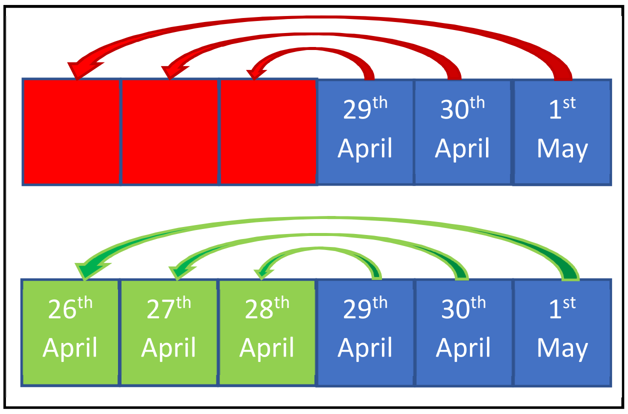


Figure 5.8 The correction function to resolve the absence of displaced data.

There was an unexpected error encountered when performing calculations on the entire dataset of displaced model variables. This occurred as the start date of the displaced values used for load forecasts is before the start date of the master dataset. To resolve this, a correction function was created that concatenated additional data to the start of the master dataset with the load and other numeric values being set to 0, **see Figure 5.8.** The calculation of displaced variables for linear regression models could then be performed over the entire dataset. However, a consequence of this is that linear regression models cannot accurately produce forecasts with the data that has zero variable values set as their displacement forecast. However, as the target user is an academic user it is expected they would understand the displaced input variables for linear regression models exceed the limits of the dataset.

6 Testing

6.1 System Testing Approach

Testing was required for the system to ensure changes to the code base maintained the correct functioning of the system. There are three levels of testing for the system, see **Figure 6.1.** Unit test execution are automated, while the higher-level tests require manual human interaction.

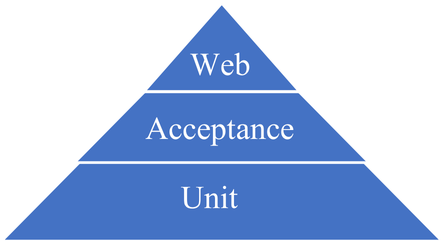


Figure 6.1 *Testing pyramid with the layers of testing for the system. The higher-level of testing the fewer tests written.*

The testing levels test different granular parts of the system:

- Unit tests: ensure individual back-end functions return expected results.
- Acceptance tests: verify the acceptance criteria of functional and non-functional requirements identified for the system are met.
- Web environment tests: ensure the correct functioning of the system in a web environment.

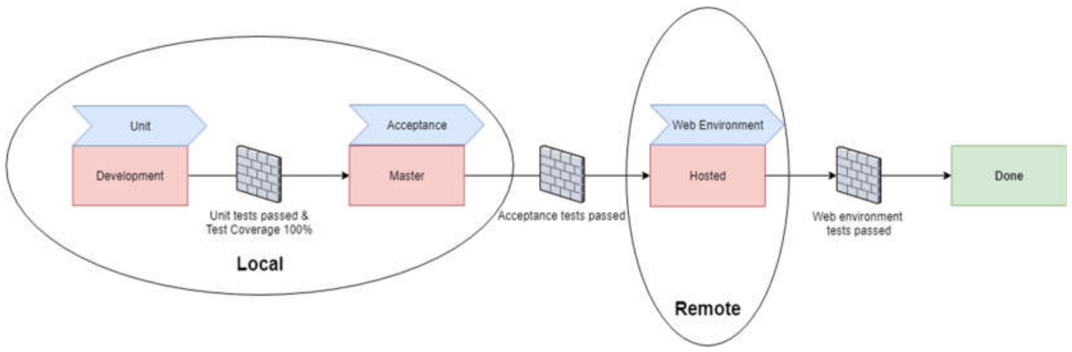


Figure 6.2 *The stages of testing for a feature that is implementation complete. The current source control stage of development is in red, with the testing type performed at that stage.*

At each stages of source control integration, a higher level of testing is performed. To progress to the next stage there is testing quality gate requirements to ensure the feature is fully tested **see Figure 6.2.** Only once all levels of testing have been passed the functional feature is ‘done’.

6.2 Unit Testing

Scope

There were decisions made on what system functions to write unit tests for in the system. Unit test class should generally test callable public methods [25]. However, all functions in the solution are publicly callable by importing their module. Unit tests were written for functions that contained several internal calls to test they were functioning correctly together. Callback functions were not tested as their correct functioning was verified through acceptance testing.

Test-Driven Development

Unit tests were written using a test-driven development (TDD) approach. Tests were written before the functionality was implemented that fail. Functionality was then implemented to pass the tests. It ensured functionality meets the defined requirements of the system, requiring only to have one single responsibility for the system.

Implementation Libraries

The **unittest** Python library was used to create a test suite with a collection of test cases for a single function. The tests are then executed using a command-line test runner, which produced the outcome of the tests. The **mock** library was used to patch functions with 'faked' return values, reducing the need to create 'proper' arguments for functions called in the tested function. To ensure all conditional paths within the functions tested were followed by the unit test cases the **coverage** library was used to generate a unit test code coverage report. See **Appendix G** for the system's report.

6.3 Acceptance Testing

See **Appendix H** for the system's acceptance test script.

Scope

Acceptance tests replicate the academic user's flow using the system. The tests are visual, validating how graphical and statistical results are presented after system components are interacted with in a defined order and using defined input values.

Manual vs. Automated Testing

Automation is the preferred approach to writing acceptance tests. Manual testing requires a person to manually go through a test script and verifying the system output for each case, which takes longer and is more prone to errors than a computer [26]. Early in the system's development the testing framework **Robot** and **Selenium** was used to write an automated test to verify the system was running. The script for this is contained in the *tests* folder. However, there was no open source library written for Robot to gather data from the system's Plotly visualisations to verify their presentation. It was determined that the effort to write these libraries was not as valuable to the stakeholder as the features being developed. In addition, automated test scripts are not effective at verifying the non-functional usability requirements of the user. [25]. Hence, acceptance testing is performed manually using a test script.

External Dependencies

Test data is used as the source of data for the acceptance tests to enable the tester to test the preprocessing of a dataset and the generation of results using the system. It provides a consistent range of data to modify using the user interface components. This ensures the same results are produced each time the acceptance test script is followed for validation. Each part of the test script was written to be independent. This was to accommodate a limited testing period. To provide confidence the system is functioning correctly, tests required to run are only those validating the area of implementation.

Cross-Browser Testing

The acceptance test script was followed using the three most popular Windows web browsers by market share [27]: Google Chrome, Mozilla Firefox and Internet Explorer. This cross-browser testing ensures that the system functionality works, is performant and is presented with no visual irregularities. Any errors identified are fixed to ensure a consistent user experience.

6.4 Web Environment Testing

Scope

Acceptance tests are limited in that they do not test the system up and running in a web environment, only deployed to a local environment. Testing how the system performs in a web environment is important as this is a user's perspective when the system is deployed to a remote web server. Exploratory testing is performed at this level to uncover errors not identified in lower levels because of the different host environment.

Proof of Concept

Hosting the system on the internet provides a proof of concept to the stakeholder that the application is deployable to web infrastructure and can serve multiple remote users. The stakeholder and parties with an interest in the system could also access the system remotely rather than having to setup the solution locally, which requires setup time and technical expertise. The presentation of components was tested on devices with variable screen size, modes of interaction and hardware specifications to ensure the system presents and provides core functionality correctly. There is a focus at this level on assessing whether the system met responsiveness requirements as the remote web server response time to requests is a variable not present in the lower-level testing.

<u>Web Hosting Platform Choices</u>

The cloud platform **Heroku** was chosen to host the system on the internet. It was chosen as it accommodates the testing usage of the hosted system, with a free plan that provides adequate web hosting hours (1000). The deployment process of Heroku is through pushing the code base using Git to a remote Heroku branch. This is the same process as pushing changes to the development and master code branch, and hence seamlessly integrated with the source control tools used during development [28].

<u>Web Deployment</u>

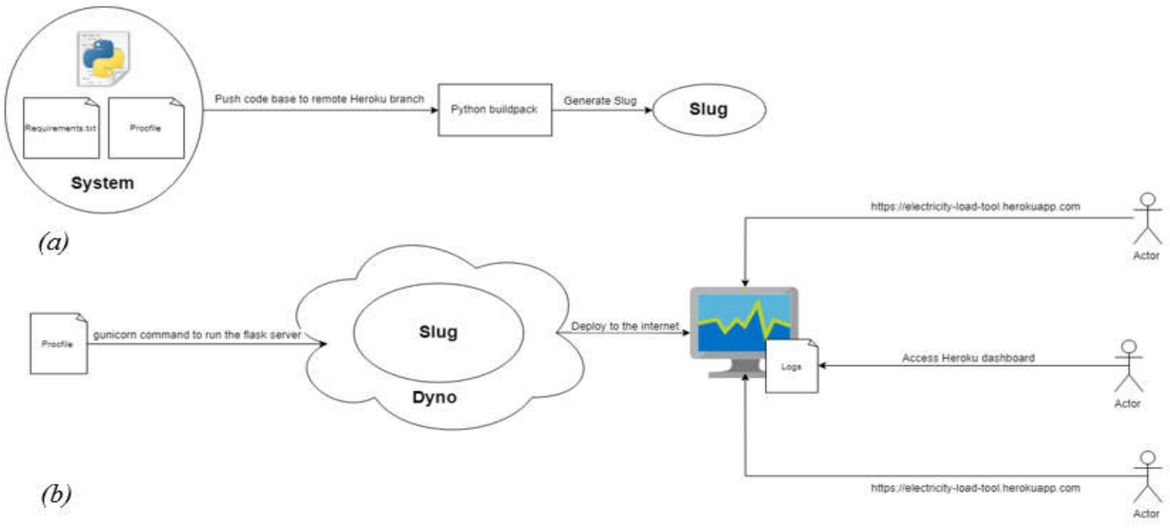

Requirements.txt
Procfile
System
Push code base to remote Heroku branch
Python buildpack
Generate Slug
Slug
(a)
Procfile
gunicorn command to run the flask server
Slug
Dyno
Deploy to the internet
Logs
https://electricity-load-tool.herokuapp.com
Actor
Access Heroku dashboard
Actor
https://electricity-load-tool.herokuapp.com
Actor
(b)

Figure 6.3 *The processes to deploy the system online using Heroku. (a) The creation of the slug. (b) The deployment of the system by executing the slug on the host dyno using the gunicorn web server command in the Procfile. The user's methods to access the web hosted system for testing and logs for debugging are visualised.*

The Python build pack is used to generate the system 'slug', see **Figure 6.3(a)**. The system slug packages the system dependencies to deploy the system. The required library dependencies are listed in a *requirements.txt* that are installed using the Python package manager **pip**. A dyno is an isolated virtualised container that provides the environment for the slug to execute is preloaded with the slug, **see Figure 6.4(b).** Commands that Heroku uses to execute the dyno are contained within the *Procfile* in the root folder of the code base. **Gunicorn** server is called to extract the Flask server in *run.py* - the entry point of the system and execute the program. When the system is deployed time-stamped logs are generated from the Flask web server, which can be used to debug errors.

7 System Evaluation and Experimental Results

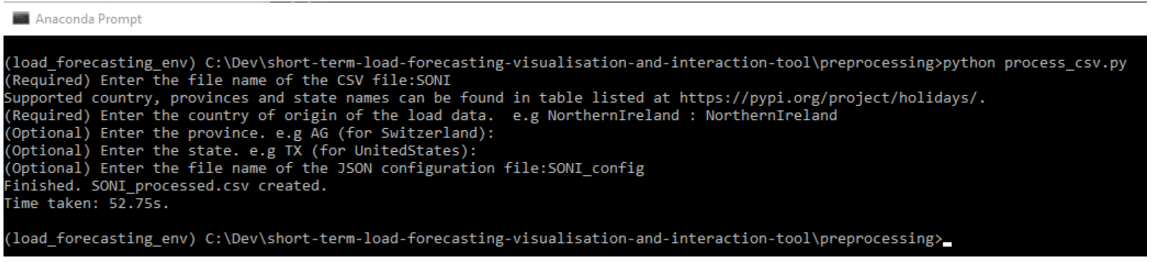
7.1 Academic Benefit

Use Case Assessment

The system has met the use cases identified for the academic user. Each use case is listed below with a short description and evidence of it being fulfilled:

1 Arrange and clean a provided dataset

The user can use a dataset that is organised in accordance with the defined data contract and load it into the preprocessing program. The output is a new dataset with fields required by the system to facilitate functionality.



```
Anaconda Prompt

(load_forecasting_env) C:\Dev\short-term-load-forecasting-visualisation-and-interaction-tool\preprocessing>python process_csv.py
(Required) Enter the file name of the CSV file:SONI
Supported country, provinces and state names can be found in table listed at https://pypi.org/project/holidays/.
(Required) Enter the country of origin of the load data.  e.g NorthernIreland : NorthernIreland
(Optional) Enter the province. e.g AG (for Switzerland):
(Optional) Enter the state. e.g TX (for UnitedStates):
(Optional) Enter the file name of the JSON configuration file:SONI_config
Finished. SONI_processed.csv created.
Time taken: 52.75s.

(load_forecasting_env) C:\Dev\short-term-load-forecasting-visualisation-and-interaction-tool\preprocessing>
```

Figure 7.1 An example of running the preprocessing program.

2 Visualise Dataset

The user can visualise a subset of the dataset's load values and compare the load with other characteristics of the dataset by selecting a column in the characteristic's dropdown. The highlighting functionality enables the user to identify trends specific to type of days.

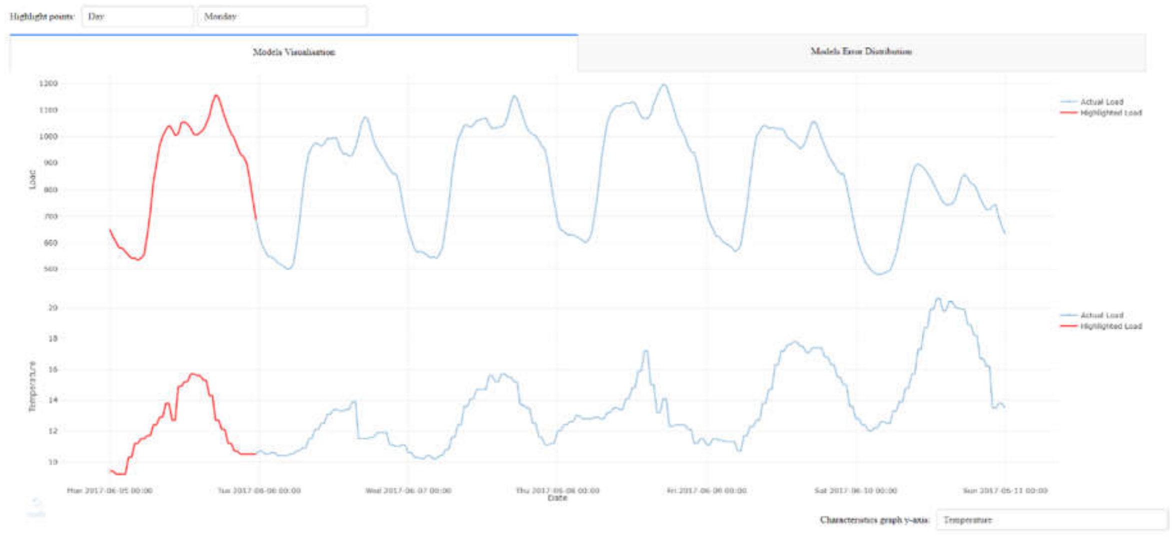


Figure 7.2 An example of visualising a subset of the load dataset and comparing it with a characteristic of the dataset (Temperature). The highlighting functionality is also visualised, with data points of day Monday being highlighted in red.

3 Add Models

The system enables the academic user to add displacement and linear regression load forecasting models to evaluate their performance.

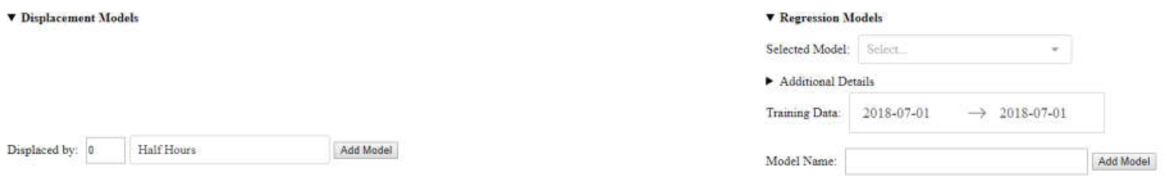
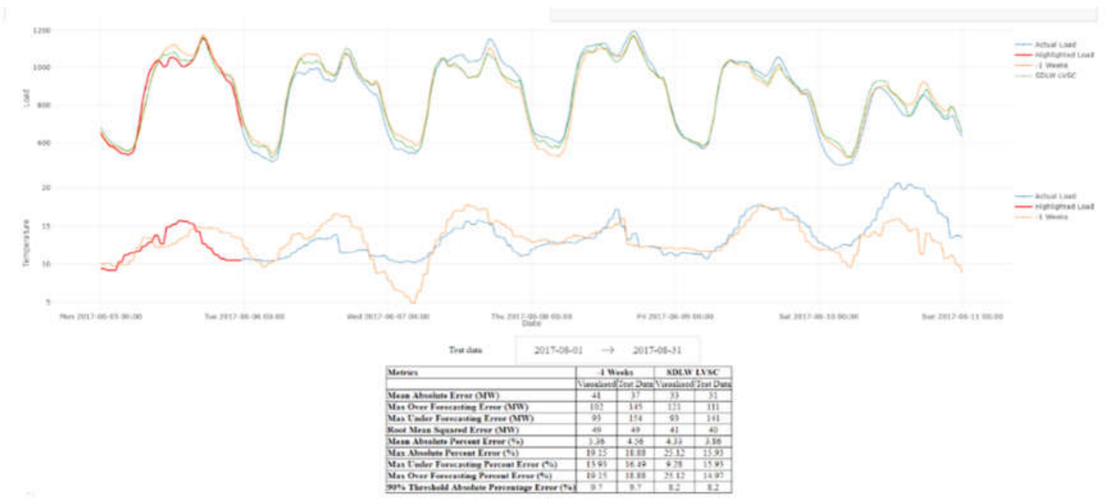


Figure 7.3 The two flows to construct a forecasting model in the solution.

4 Compare Model Performance

The system enables the user to compare the load forecasting performance of models they have added visually by assessing the accuracy of the load forecast with the actual load.



Metrics	-1 Weeks		SDLW LVSC	
	Visualised	Test Data	Visualised	Test Data
Mean Absolute Error (MW)	41	37	33	31
Max Over Forecasting Error (MW)	102	145	121	111
Max Under Forecasting Error (MW)	93	154	93	141
Root Mean Squared Error (MW)	49	49	41	40
Mean Absolute Percent Error (%)	3.36	4.56	4.33	3.86
Max Absolute Percent Error (%)	19.15	18.88	25.12	15.93
Max Under Forecasting Percent Error (%)	13.93	16.49	9.28	15.93
Max Over Forecasting Percent Error (%)	19.15	18.88	25.12	14.97
90% Threshold Absolute Percentage Error (%)	9.7	9.7	8.2	8.2

Figure 7.4 An example of adding a SDLW displacement and a linear regression model to visualise the model load forecasts. The accompanying error statistics below are used to compare the performance of the model on the specified visualised and test data subset.

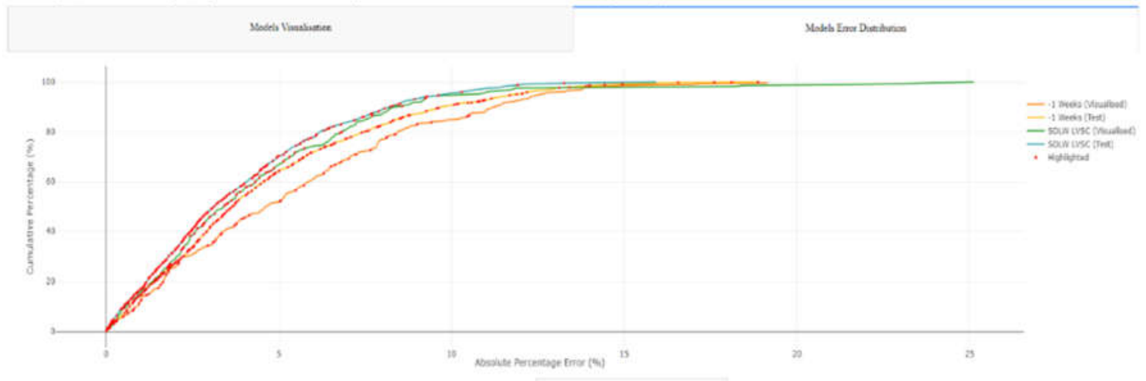


Figure 7.5 An example of comparing the forecasting performance of test and visualised data through an error graph visualising the distribution of errors within the dataset

5 Export Data

The proposed process of exporting both visual data as vector graphs and statistical data results in excel format from the system was not implemented as require in **FTR05**. The academic user can however export PNG visualisations which can be used in their academic papers.

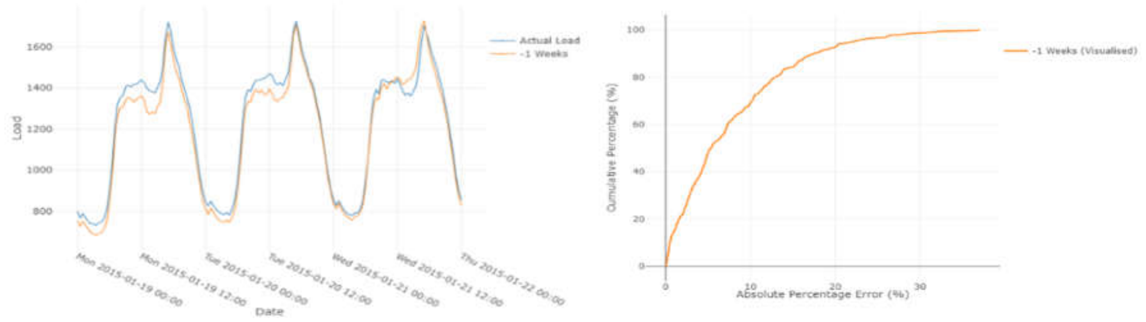


Figure 7.6 *Examples of static PNG visualisations produced by in-built Plotly functionality.*
(a) Load and model forecast visualisation
(b) Model forecast error visualisation

7.2 Further Work

Implementation Libraries

The dissertation provides an appreciation of the challenges encountered in implementation to guide developers in extending the code base. For assistance in implementing new functionality to the system the Dash framework and the Plotly visualisation library both have detailed documentation online [21][29]. They are both being updated with regular updates, therefore their change notes in their repositories should be viewed on a regular basis to keep up to date with new features that could be used to refactor or add new functionality [30].

Commercial Benefit of the System

The system is of commercial benefit to a system operator as they can use the system to have a greater visibility of the performance of a load forecasting model on historical data. They can then use the produced results in their decisions making regarding their day ahead load forecasting to make their operations more efficient. The system could be made more commercially valuable to system operators by adapting the visualisations and interaction to forecast system load for a given day using an automated model selection process. A proposal of this extended system functionality for an industrial user is in **Appendix I.**

New Forecasting Models

The inclusion of new types of models e.g. non-linear models, would require code changes to the system. A new interface in the layout to construct the model analogous to the current interface for model construction in **Figure 7.3**, a new model file with a predict method and changes to the configuration structure would be required. The functionality to adding the model forecast traces to a graph is reusable through unified utility files with shared functionality.

Internationalisation

The strings used in the user interface to label functionality are contained within the system layout. The layout could instead be reworked to reference a configuration file with a list of strings that contain language translations. This would make the solution capable of being made accessible to an international non-English speaking audience by adding new foreign language translations to the configuration file.

Additional Functionality

Other functionality was proposed for the system, but was not developed because of time constraints:

- A tool with a graphical user interface to create linear regression models for use in the solution. This would be more intuitive to the user than the current requirement of having manually modify the JSON configuration in a text editor to change the list of linear regression models used in the system.
- The load dataset decoupled from the solution and stored remotely. Subsets of the dataset can then be accessed through call to an external endpoint. This is the current system process of loading all the data into the host system's memory.
- Minor bugs and tasks identified during implementation in the 'Backlog' and 'Parking Lot' section of the system's Trello board [31].

References

[1] J. Bailey, "Code base of the system developed", *GitLab*, 2019. [Online]. Available: https://gitlab.eeecs.qub.ac.uk/40156063/short-term-load-forecasting-visualisation-and-interaction-tool [Accessed: 17- Apr- 2019].

[2] A. Ulbig, T. Borsche and G. Andersson, "Impact of Low Rotational Inertia on Power System Stability and Operation", IFAC Proceedings Volumes, vol. 47, no. 3, pp. 7290-7297, 2014. Available: 10.3182/20140824-6-za-1003.02615.

[3] J. Foster, X. Liu and S. McLoone, "Load forecasting techniques for power systems with high levels of unmetered renewable generation: A comparative study", IFAC-PapersOnLine, vol. 51, no. 10, pp. 109-114, 2018

[4] R. Azizipanah-Abarghooee, V. Terzija, F. Golestaneh, and A. Roosta, “Multiobjective Dynamic Optimal Power Flow Considering FuzzyBased Smart Utilization of Mobile Electric Vehicles,” IEEE Transactions on Industrial Informatics, vol. 12, no. 2, pp. 503–514, apr 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7384482/

[5] "Intraday Market Auctions", Lg.sem-o.com, 2019. [Online]. Available: http://lg.sem-o.com/ISEM/Pages/IntradayMarketAuctions.aspx. [Accessed: 21- Mar- 2019].

[6] J. Foster, X. Liu, and S. Mcloone, “Adaptive sliding window load forecasting,” 2017 28th Irish Signals and Systems Conference (ISSC), 2017.

[7] E. Almeshaiei and H. Soltan, "A methodology for electric power load forecasting," Alexandria Engineering Journal, vol. 50(2), pp. 137-144, 2011.

[8] Wemcouncil.org, 2019. [Online]. Available: http://www.wemcouncil.org/wp/wp-content/uploads/2015/07/1230_YingChen.pdf. [Accessed: 18- Apr- 2019].

[9] "Use Case Examples", Gatherspace.com, 2019. [Online]. Available: https://www.gatherspace.com/use-case-examples/. [Accessed: 19- Apr- 2019].

[10] "Principles behind the Agile Manifesto", Agilemanifesto.org, 2019. [Online]. Available: https://agilemanifesto.org/principles.html. [Accessed: 19- Apr- 2019].

[11] "Trello", Trello.com, 2019. [Online]. Available: https://trello.com/en. [Accessed: 19- Apr- 2019].

[12] "Software Engineering | Architectural Design", GeeksforGeeks, 2019. [Online]. Available: https://www.geeksforgeeks.org/software-engineering-architectural-design/. [Accessed: 19- Apr- 2019].

[13] "On stateless software design", www.leonmergen.com, 2019. [Online]. Available: https://leonmergen.com/on-stateless-software-design-what-is-state-72b45b023ba2. [Accessed: 19- Apr- 2019].

[14] A. Bartels and A. Bartels, "Coding in the Cloud - Rule 3 - Use a "Stateless" design whenever possible", The Official Rackspace Blog, 2019. [Online]. Available: https://blog.rackspace.com/coding-in-the-cloud-rule-3-use-a-stateless-design-whenever-possible. [Accessed: 19- Apr- 2019].

[15] "Cross browser testing", *MDN Web Docs*, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing. [Accessed: 19- Apr- 2019].

[16] K. Profile, "Web Designs for Slow Internet Connections", Smashstack.com, 2019. [Online]. Available: https://www.smashstack.com/articles/life-in-the-slow-lane-web-designs-for-slow-internet-connections/. [Accessed: 19- Apr- 2019].

[17] "Pencil Project", Pencil.evolus.vn, 2019. [Online]. Available: https://pencil.evolus.vn/. [Accessed: 19- Apr- 2019].

[18] M. Stone, "Expert color choices for presenting data", StoneSoup Consulting, 2006. [Accessed 19 April 2019].

[19] M. Theuwissen, "R vs Python for Data Science", Kdnuggets.com, 2019. [Online]. Available: https://www.kdnuggets.com/2015/05/r-vs-python-data-science.html. [Accessed: 20- Apr- 2019].

[20] "Understanding Conda and Pip", Anaconda, 2019. [Online]. Available: https://www.anaconda.com/understanding-conda-and-pip/. [Accessed: 20- Apr- 2019].

[21] "Dash User Guide and Documentation", Dash.plot.ly, 2019. [Online]. Available: https://dash.plot.ly/introduction. [Accessed: 20- Apr- 2019].

[22] "Most Common Programming Case Types", Chaseonsoftware.com, 2019. [Online]. Available: https://chaseonsoftware.com/most-common-programming-case-types/. [Accessed: 20- Apr- 2019].

[23] S. H, "Object Oriented Programming", *Python.swaroopch.com*, 2019. [Online]. Available: https://python.swaroopch.com/oop.html. [Accessed: 20- Apr- 2019].

[24] “Why use JSON over XML?", *SitePoint*, 2019. [Online]. Available: https://www.sitepoint.com/json-vs-xml/. [Accessed: 20- Apr- 2019].

[25] H. Vocke, "The Practical Test Pyramid", martinfowler.com, 2019. [Online]. Available: https://martinfowler.com/articles/practical-test-pyramid.html. [Accessed: 20- Apr- 2019].

[26] "The different types of testing in Software", Atlassian, 2019. [Online]. Available: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing. [Accessed: 20- Apr- 2019].

[27] "W3Counter: Global Web Stats", W3counter.com, 2019. [Online]. Available: https://www.w3counter.com/globalstats.php. [Accessed: 20- Apr- 2019].

[28] "How Heroku Works", Devcenter.heroku.com, 2019. [Online]. Available: https://devcenter.heroku.com/articles/how-heroku-works#defining-an-application. [Accessed: 20- Apr- 2019].

[29] "Plotly User Guide", Plot.ly, 2019. [Online]. Available: https://plot.ly/python/user-guide/. [Accessed: 20- Apr- 2019].

[30] "Plotly", GitHub, 2019. [Online]. Available: https://github.com/plotly. [Accessed: 20-Apr- 2019].

[31] "Trello", Trello.com, 2019. [Online]. Available: https://trello.com/b/RrMXYpy1/short-term-electricity-load-forecasting-visualisation-and-interaction-tool. [Accessed: 20- Apr- 2019]**.**

Appendices

Appendix A

Date	Temperature over 18hrs	Humidity	**Week**
Load	Temperature over 24hrs	Sun duration*potential solar irradiance	**Quarter**
Temperature	Temperature over 36hrs	Binary indicator sunny day	**Hour**
Temperature-6hrs	Temperature over 48hrs	Binary indicator windy day	**Minute**
Temperature-12hrs	Temperature over 72hrs	Potential solar irradiance	**Day**
Temperature-18hrs	Temperature over 96hrs	Weekday	**Dayofweek**
Temperature-24hrs	Wind speed	Yearly cycle (sine wave)	**Dayofyear**
Temperature-36hrs	Wind speed^2	Yearly cycle (cosine wave)	
Temperature-48hrs	Wind speed^3	Daily cycle (sine wave)	
Temperature-72hrs	Wind direction	Daily cycle (cosine wave)	
Temperature-96hrs	Cloud height	Holiday_Alternate	
Temperature over 6hrs	Sun duration	**Year**	
Temperature over 12hrs	Visibility	**Month**	

Figure 1 *A list of the columns included in the SONI dataset. The variables added during the data cleaning process are in bold.*

<u>Appendix B</u>

Holidays	**SDLW Predicted Load - Actual Load Correlation**
Holiday New Year's Day	0.739
Derived Christmas Day	0.748
Holiday Easter Monday	0.755
Holiday Christmas Day	0.82
Holiday Boxing Day	0.841
Derived Easter Monday	0.846
Derived Battle of the Boyne	0.901
Derived Good Friday	0.911
Holiday Christmas Day (Observed)	0.923
Holiday May Day	0.947
Holiday Boxing Day (Observed)	0.95
Derived Late Summer Bank Holiday	0.961
Derived St. Patrick's Day	0.963
Holiday Good Friday	0.966
Derived Boxing Day (Observed)	0.966
Derived Spring Bank Holiday	0.967
Derived May Day	0.969
Holiday St. Patrick's Day	0.971
Holiday Late Summer Bank Holiday	0.979
Derived St. Patrick's Day (Observed)	0.98
Holiday Spring Bank Holiday	0.981
Holiday St. Patrick's Day (Observed)	0.981
Holiday Battle of the Boyne	0.986
Derived New Year's Day	0.988
Derived Boxing Day	0.991
Derived Christmas Day (Observed)	0.997
Holiday New Year's Day (Observed)	N/A
Derived New Year's Day (Observed)	0.994

Table 1 *All the Northern Ireland holiday days and the correlation of each holiday.*

Appendix C

SDLW Linear Regression Model

Test	2012	2013	2014	2015	2016	2017	**Average**
Training							
2012	4.002	3.834	3.639	4.107	4.566	5.838	4.331
2013	3.998	3.822	3.628	4.099	4.559	5.835	4.323
2014	3.979	3.802	3.595	4.074	4.531	5.822	4.301
2015	4.003	3.839	3.642	4.109	4.568	5.838	4.333
2016	4.005	3.850	3.644	4.110	4.567	5.838	4.336
2017	4.018	3.868	3.666	4.127	4.585	5.847	4.352

SDLW Temperature Corrected Linear Regression Model Results

Test	2012	2013	2014	2015	2016	2017	**Average**
Training							
2012	3.950	3.702	3.487	3.940	4.351	5.627	4.176
2013	4.011	3.706	3.516	3.918	4.302	5.500	4.159
2014	4.013	3.709	3.518	3.919	4.301	5.498	4.160
2015	4.131	3.795	3.612	3.976	4.331	5.430	4.213
2016	4.088	3.765	3.578	3.953	4.312	5.448	4.191
2017	4.222	3.871	3.689	4.037	4.385	5.415	4.270

SDLW Weather Corrected Linear Regression Model Results

Test	2012	2013	2014	2015	2016	2017	**Average**
Training							
2012	3.815	3.579	3.393	3.814	4.215	5.771	4.098
2013	3.905	3.570	3.479	3.876	4.439	6.150	4.237
2014	3.858	3.603	3.343	3.763	4.050	5.617	4.039
2015	4.019	3.678	3.459	3.803	4.125	5.893	4.163
2016	4.411	4.305	3.915	4.243	3.832	5.080	4.298
2017	4.765	4.945	4.476	4.973	4.167	4.695	4.670

SDLW Weather Correction Model LASSO Regression Model

Test	2012	2013	2014	2015	2016	2017	Average	Variables	λ
Training									
2012	8.385	9.093	9.732	9.962	11.022	14.341	10.422	42	0.02
2013	9.309	8.433	8.855	9.342	10.135	12.899	9.829	42	0.04
2014	9.545	8.551	8.481	8.905	9.433	11.984	9.483	43	0.02
2015	9.351	8.810	8.690	8.558	9.279	11.644	9.389	43	0.02
2016	10.022	9.207	8.714	8.667	8.620	10.615	9.308	41	0.12
2017	11.488	10.423	9.705	9.670	9.104	9.303	9.949	43	0.02

Appendix D

Table 2.5 Full

Variable	Occurrence (%)
Temperature	100
Temperature-12hrs	100
Temperature-96hrs	100
Wind speed^3	100
Wind direction	100
Cloud height	100
Visibility	100
Humidity	100
Yearly cycle (sine wave)	100
Holiday_Alternate	100
Hour	100
Day	100
Dayofyear	100
Temperature Difference	100
Temp-48 Difference	100
Wind speed Difference	100
Humidity Difference	100
Temperature-24hrs	83.33333
Temperature-48hrs	83.33333
Temperature over 6hrs	83.33333
Wind speed^2	83.33333
Weekday	83.33333
Week	83.33333
Temperature-18hrs	83.33333
Temperature-72hrs	83.33333
Wind speed	83.33333
Dayofweek	83.33333
Sun duration*potential solar irradiance Difference	83.33333
Temperature-36hrs	66.66667
Sun duration	66.66667
Binary indicator windy day	66.66667
Yearly cycle (cosine wave)	66.66667
Daily cycle (sine wave)	66.66667
Daily cycle (cosine wave)	66.66667
Temperature-6hrs	66.66667
Quarter	66.66667
Temperature over 48hrs	50
Minute	50
Binary indicator sunny day	50
Temperature over 12hrs	33.33333

Temperature over 96hrs	33.33333
Temperature over 18hrs	33.33333
Temperature over 24hrs	33.33333
Temperature over 36hrs	33.33333
Sun duration*potential solar irradiance	33.33333
Potential solar irradiance	16.66667
Month	16.66667

Figure 2.7 (a) Full

Variable	**Alpha**	**MAPE**	**MAPE Difference**
Humidity Difference	6000	4.754	0.000
Cloud height	500	4.748	-0.006
Wind direction	200	4.751	0.003
Wind speed^2	200	4.751	0.000
Dayofyear	100	4.630	-0.121
Wind speed^3	100	4.630	0.000
Temperature Difference	60	4.630	0.000
Day	50	4.520	-0.110
Temp-48 Difference	40	4.537	0.016
Holiday_Alternate	6.5	4.497	-0.039
Yearly cycle (sine wave)	6.5	4.497	0.000
Wind speed Difference	5	4.298	-0.199
Dayofweek	4.5	4.294	-0.004
Temperature-24hrs	4	4.298	0.004
Temperature-96hrs	4	4.298	0.000
Hour	3.5	4.276	-0.023
Humidity	2	4.278	0.003
Temperature	2	4.278	0.000
Weekday	2	4.278	0.000
Year	2	4.278	0.000
Temperature-36hrs	1.5	4.274	-0.004
Temperature-18hrs	1	4.274	0.000
Temperature-48hrs	0.9	4.274	0.000
Temperature-72hrs	0.85	4.274	0.000
Sun duration*potential solar irradiance Difference	0.8	4.274	0.000
Week	0.7	4.228	-0.046
Visibility	0.65	4.228	0.000
Quarter	0.6	4.228	0.000
Binary indicator windy day	0.45	4.225	-0.003
Temperature over 6hrs	0.45	4.225	0.000
Binary indicator sunny day	0.4	4.221	-0.004
Temperature-6hrs	0.4	4.221	0.000
Wind speed	0.35	4.221	0.000

Daily cycle (sine wave)	0.3	4.221	0.000
Sun duration	0.3	4.221	0.000
Temperature over 48hrs	0.3	4.221	0.000
Daily cycle (cosine wave)	0.2	4.216	-0.005
Yearly cycle (cosine wave)	0.2	4.216	0.000
Month	0.1	4.213	-0.003
Sun duration*potential solar irradiance	0.1	4.213	0.000
Temperature over 12hrs	0.1	4.213	0.000
Temperature over 24hrs	0.1	4.213	0.000
Temperature-12hrs	0.1	4.213	0.000
Minute	0.05	4.208	-0.005
Potential solar irradiance	0.05	4.208	0.000
Temperature over 18hrs	0.045	4.207	-0.001
Temperature over 96hrs	0.03	4.207	0.000
Temperature over 36hrs	0.025	4.207	0.000
Temperature over 72hrs	0.025	4.207	0.000

Appendix E

FTR00	**Visualise and interact with the historical load data**		
US00	High	Open the application in a web browser	Done
• View the application in a web browser by navigating to an URL.			
US01	High	View a graph visualising historical load data	Done
• The full date range of historical load data must be plotted. • Date values on the x axis labelled. • Load values on the y axis labelled.			
US02	High	Interact with the date range visualised.	Done
• The range of load data visualised must change when changing the start and/or end date. • The range of load data visualised must change when increasing or decreasing the start or end date by a date-time unit. • Midnight of the end date must be the final point plotted.			
FTR01	**Evaluate the forecasting performance of a displacement model**		
US03	High	Visualise displacement model's load forecasting performance	Done
• The user must be able to choose what unit of time and magnitude for the displacement model construction. • When a model has been added the displaced plots must superimpose the visualised load data. • The new model's name should appear along with the 'Actual Load' legend with matching colours to points on the graph. The name of the model should be in the format: *displacement* | *unit* e.g. -7 Days • The date the load forecast is derived from must be visible hovering over a model forecasted point. • The forecasting model forecast must change when changing the visualised date range.			
US04	High	View forecasting performance metrics of the visualised displaced model	Done

• When a model is added to the load visualisation graph a table containing statistical data relevant to judging the model's predictive performance must be visible with the identifying column named the model name in the visualised graph. • When the visualised load date range changes the statistical data in the table updates dynamically. • The table is not visible when no models are added.			
FTR02	**Compare multiple forecasting models' performance**		
US05	High	Compare two displaced model's forecasting performance	Done
• When two models have been added two forecast model plots with different styles should be superimposed onto the visualised load data graph. • Two displaced model's forecasting performance statistically presented through two distinct columns in the metrics table with the model's name as the identifying column name. • If test data is selected a sub column labelled 'Test Data' must be added to the table beside the model's visualised data metrics. • When a model is removed the columns should be removed from the metrics table and not superimposed on the graph.			
US06	High	View two graphs with different y axis over the same time range to compare characteristics contributing to differences in model performance	Done
• The user should be able to add or remove a characteristics graph. • Y axis options are the dataset's columns. • Changing the visualised date range of the Date-Load graph changes the characteristics graph's visualised date range. • If a model is plotted, the displacement models' plots of the chosen y-axis are superimposed onto the characteristics graph. • The date x axis of the visualised load graph must not be visible. • The date x axis of the characteristics graph must be visible. • The grid lines of the characteristics graph should be seamlessly aligned with the gridlines of the visualised load graph.			

FTR03	**Advanced Visualisation and Interaction**		
US07	Medium	Highlight different types of days	Done
• When choosing to highlight a specific holiday day, the full list of holidays in the dataset must be listed. • When choosing to highlight a specific year, the full list of years in the dataset must be listed. • The selected chosen type of day must be plotted as red points superimposing the load points if matching the day characteristic chosen to highlight. • When a highlighted day type is chosen, the legend must include 'Highlighted Load'.			
US08	High	Visualise the error distribution of a forecasting model	Done
• An error distribution graph must be visible in a separate tab if model(s) are added. • APE values on the x axis labelled. • Cumulative percentage values on the y axis labelled. • The forecasting model points should be the same colour as in the load visualisation graph. • When a model is added/removed it should be added/removed from the error distribution visualisation graph. • If there is test data range selected the test data model forecasting error points must be visible on the graph. • Visualised data forecasting error points must have the model name and (Visualised) appended to it in the legend. • Test data forecasting error points must have the model name and (Test) appended to it in the legend.			
US09	Low	Choose how the graph is presented	Parked
• When the marker dots properties (e.g. colour, width, line type) are modified the forecast model and load points visualised must have the properties specified.			
FTR04	**Assess the forecasting performance of linear regression models**		
US10	High	Visualise a linear regression forecasting model to evaluate its performance	Done
• The list of models the user can choose must be the models included in the dataset configuration file.			

- The user must be able to choose a training data start date and end date.
- The user must be able to view the variables used in the linear regression model and a description of the linear regression model.
- The user must be able to change the model name from default.
- The linear regression model plots must be superimposed onto the visualised load data.
- The name of linear regression model must be visible in the legend of the load and error visualisation graphs.
- If the regression model has a default training range, it must by default change the start date and end date to it when changing the model chosen.

FTR05	**Export the visualisations and statistical metrics**		
US11	High	Export the visualised graphs	Backlog

- When the user clicks export a zip file should be downloaded to their local machine that includes:
  - If a range of data is selected, the load visualisation graph with, if model(s) are added, superimposed model forecasts named 'load_graph'
  - If a characteristic is selected, the characteristics graph named 'characteristics_graph'
  - If model(s) are added, the model error visualisation graph 'model_error_visualisation_graph'
- The format of the visualisations graph images must be SVG.
- The zip file should be named 'lftool_export'

US12	High	Export the metrics table statistical data	Backlog

- When the user clicks export a zip file should be downloaded to their local machine that includes the statistical data in the format:

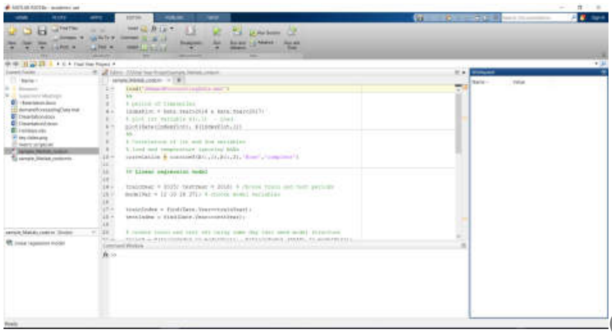
Model Name	**Data_Type**	**Metrics…**
Name of the model in the visualised graph	*Visualised or Test Data*	*The metric value*

- The format of the data must be CSV.
- The name of the data must be 'metrics'

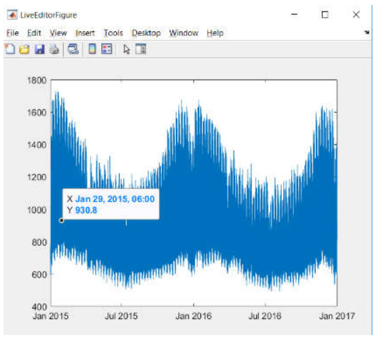
- The metric values must be added to their calculated raw value, not the rounded values in the metrics table.
- If no models are added the file will not be included.

Appendix F

MATLAB User Interface Analysis



(a)



(b)

***Figure 1** Examples of MATLAB's user interface **(a)** Loading a MATLAB script. **(b)** Visualisation of load data.*

In **Figure 1(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.

Appendix G

Coverage report: 99%

filter...

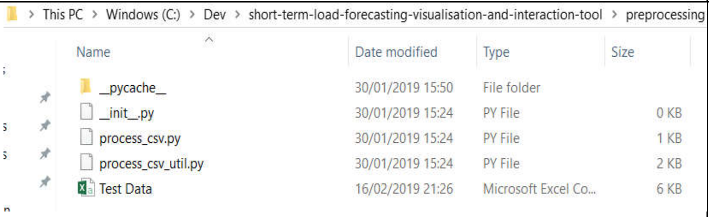
Module ↓	*statements*	*missing*	*excluded*	*coverage*
app.py	5	0	0	100%
callback\__init__.py	6	0	0	100%
callback\date_highlight_callbacks.py	5	1	0	80%
callback\date_picker_callbacks.py	7	2	0	71%
callback\graph_callbacks.py	12	3	0	75%
callback\model_callbacks.py	10	2	0	80%
callback\model_metrics_table_callbacks.py	9	2	0	78%
callback\regression_model_callbacks.py	5	1	0	80%
component\__init__.py	0	0	0	100%
component\graph\__init__.py	0	0	0	100%
component\graph\error_model\__init__.py	0	0	0	100%
component\graph\error_model\error_distribution_graph.py	9	0	0	100%
component\graph\error_model\util\error_model_graph_util.py	88	2	0	98%
component\graph\util\__init__.py	0	0	0	100%
component\graph\util\graph_colors.py	3	0	0	100%
component\graph\util\graph_util.py	26	0	0	100%
component\graph\visualise_model\__init__.py	0	0	0	100%
component\graph\visualise_model\characteristics_graph.py	16	0	0	100%
component\graph\visualise_model\load_graph.py	24	0	0	100%
component\graph\visualise_model\util\__init__.py	0	0	0	100%
component\graph\visualise_model\util\visualise_model_graph_util.py	55	0	0	100%
component\graph_control\__init__.py	0	0	0	100%
component\graph_control\characteristics_selection.py	7	0	0	100%
component\graph_control\date_highlight_value_selection.py	50	0	0	100%
component\graph_control\date_picker_day_range.py	15	0	0	100%
component\model\__init__.py	0	0	0	100%
component\model\displacement\__init__.py	0	0	0	100%
component\model\displacement\displacement_model.py	15	0	0	100%
component\model\metric\__init__.py	0	0	0	100%
component\model\metric\model_metrics_table.py	94	0	0	100%
component\model\metric\util\__init__.py	0	0	0	100%
component\model\metric\util\model_metrics.py	57	0	0	100%
component\model\model_selection.py	8	0	0	100%
component\model\regression\__init__.py	0	0	0	100%
component\model\regression\regression_model.py	36	0	0	100%
component\model\regression\regression_model_selection.py	6	0	0	100%
component\model\regression\regression_model_variables_list.py	14	0	0	100%
data\__init__.py	0	0	0	100%
data\data.py	9	2	0	78%
data\util\__init__.py	0	0	0	100%
data\util\data_characteristics.py	5	0	0	100%
data\util\data_regression_models.py	18	0	0	100%
layout.py	10	0	0	100%
preprocessing\__init__.py	0	0	0	100%
preprocessing\process_config_util.py	64	0	0	100%
Total	**1959**	**15**	**0**	**99%**

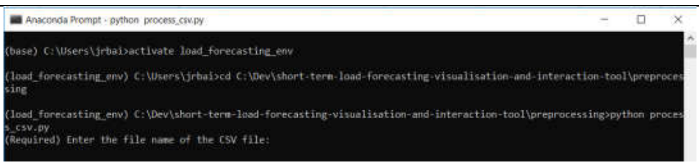
Module ↓	*statements*	*missing*	*excluded*	*coverage*
preprocessing\process_csv_util.py	38	0	0	100%
state\__init__.py	0	0	0	100%
state\model\__init__.py	0	0	0	100%
state\model\model_state.py	56	0	0	100%
unit\__init__.py	0	0	0	100%
unit\component\__init__.py	0	0	0	100%
unit\component\graph\__init__.py	0	0	0	100%
unit\component\graph\error_model\__init__.py	0	0	0	100%
unit\component\graph\error_model\error_distribution_graph_test.py	106	0	0	100%
unit\component\graph\error_model\util\__init__.py	0	0	0	100%
unit\component\graph\error_model\util\error_model_graph_util_test.py	19	0	0	100%
unit\component\graph\util\__init__.py	0	0	0	100%
unit\component\graph\util\graph_util_test.py	0	0	0	100%
unit\component\graph\visualise_model\__init__.py	0	0	0	100%
unit\component\graph\visualise_model\characteristics_graph_test.py	66	0	0	100%
unit\component\graph\visualise_model\load_graph_test.py	63	0	0	100%
unit\component\graph\visualise_model\util\__init__.py	0	0	0	100%
unit\component\graph\visualise_model\util\visualise_model_graph_util_test.py	141	0	0	100%
unit\component\graph_control\__init__.py	0	0	0	100%
unit\component\graph_control\characteristics_selection_test.py	16	0	0	100%
unit\component\graph_control\date_highlight_value_selection_test.py	33	0	0	100%
unit\component\graph_control\date_picker_day_range_test.py	36	0	0	100%
unit\component\model\__init__.py	0	0	0	100%
unit\component\model\metric\__init__.py	0	0	0	100%
unit\component\model\metric\model_metrics_table_test.py	108	0	0	100%
unit\component\model\metric\util\__init__.py	0	0	0	100%
unit\component\model\metric\util\model_metrics_test.py	16	0	0	100%
unit\component\model\model_selection_test.py	14	0	0	100%
unit\component\model\regression\__init__.py	0	0	0	100%
unit\component\model\regression\regression_model_selection_test.py	13	0	0	100%
unit\component\model\regression\regression_model_test.py	72	0	0	100%
unit\component\model\regression\regression_model_variables_list_test.py	25	0	0	100%
unit\data\__init__.py	0	0	0	100%
unit\data\util\__init__.py	0	0	0	100%
unit\data\util\data_characteristics_test.py	0	0	0	100%
unit\data\util\data_regression_models_test.py	21	0	0	100%
unit\preprocessing\__init__.py	2	0	0	100%
unit\preprocessing\process_config_util_test.py	116	0	0	100%
unit\preprocessing\process_csv_util_test.py	137	0	0	100%
unit\state\__init__.py	0	0	0	100%
unit\state\model\__init__.py	0	0	0	100%
unit\state\model\model_state_test.py	172	0	0	100%
user_parameters.py	1	0	0	100%
Total	**1959**	**15**	**0**	**99%**

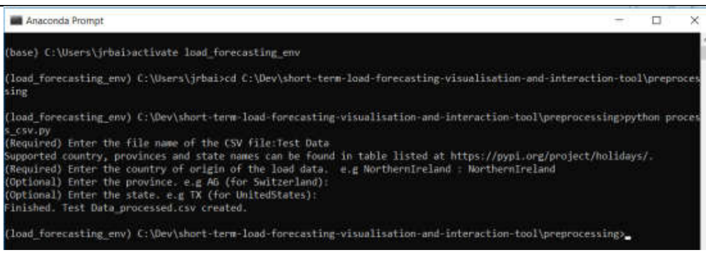
coverage.py v4.5.1, created at 2019-03-18 22:23

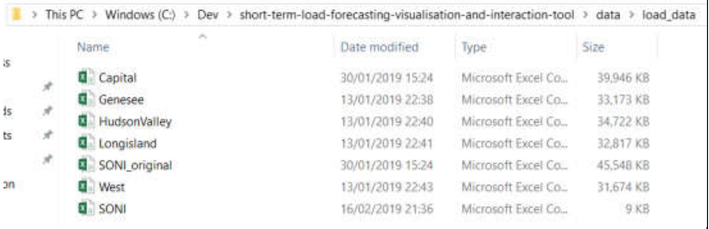
Appendix H

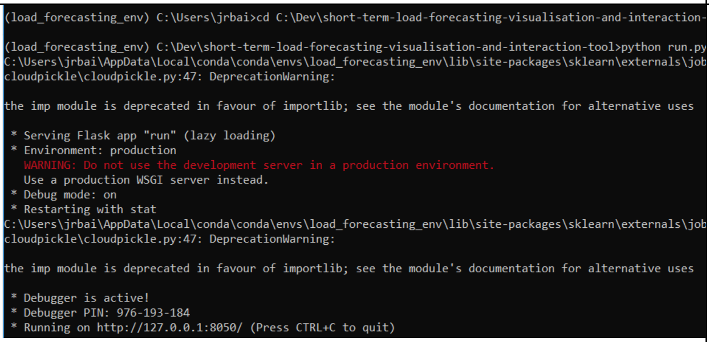
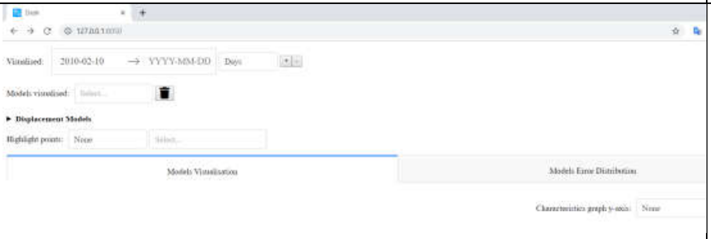
Part 1: Test environment setup

	Actions	Expected Result	Done
1	Copy file 'Test Data.csv' in the test directory and paste to: *<project-install-directory>\short-term-load-forecasting-visualisation-and-interActions-tool\preprocessing*	 This PC > Windows (C:) > Dev > short-term-load-forecasting-visualisation-and-interaction-tool > preprocessing Name / Date modified / Type / Size __pycache__ / 30/01/2019 15:50 / File folder / __init__.py / 30/01/2019 15:24 / PY File / 0 KB process_csv.py / 30/01/2019 15:24 / PY File / 1 KB process_csv_util.py / 30/01/2019 15:24 / PY File / 2 KB Test Data / 16/02/2019 21:26 / Microsoft Excel Co... / 6 KB	

2	Execute *python process_csv.py.*	 Anaconda Prompt - python process_csv.py
		`(base) C:\Users\jrbai>activate load_forecasting_env`
		`(load_forecasting_env) C:\Users\jrbai>cd C:\Dev\short-term-load-forecasting-visualisation-and-interaction-tool\preprocessing`
		`(load_forecasting_env) C:\Dev\short-term-load-forecasting-visualisation-and-interaction-tool\preprocessing>python process_csv.py`
		`(Required) Enter the file name of the CSV file:`	

3	Type in:
	• *Test Data*
	• *NorthernIreland*
	• *Press enter twice to skip state and province*
		 Anaconda Prompt
		`(base) C:\Users\jrbai>activate load_forecasting_env`
		`(load_forecasting_env) C:\Users\jrbai>cd C:\Dev\short-term-load-forecasting-visualisation-and-interaction-tool\preprocessing`
		`(load_forecasting_env) C:\Dev\short-term-load-forecasting-visualisation-and-interaction-tool\preprocessing>python process_csv.py`
		`(Required) Enter the file name of the CSV file:Test Data`
		`Supported country, provinces and state names can be found in table listed at https://pypi.org/project/holidays/.`
		`(Required) Enter the country of origin of the load data.  e.g NorthernIreland : NorthernIreland`
		`(Optional) Enter the province. e.g AG (for Switzerland):`
		`(Optional) Enter the state. e.g TX (for UnitedStates):`
		`Finished. Test Data_processed.csv created.`
		`(load_forecasting_env) C:\Dev\short-term-load-forecasting-visualisation-and-interaction-tool\preprocessing>_`	

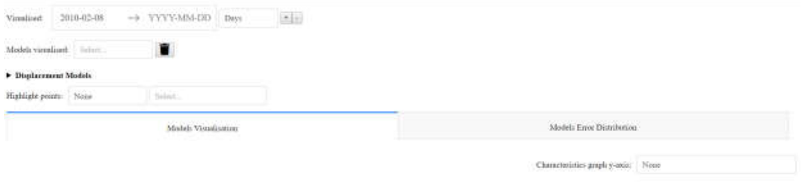
4	Delete 'Test Data.CSV' in *preprocessing.* Rename 'Test Data_processed.csv' to 'SONI.csv' Cut 'SONI.csv' and paste to: *<project-install-directory>\short-term-load-forecasting-visualisation-and-interActions-tool\data\load_data*	 This PC > Windows (C:) > Dev > short-term-load-forecasting-visualisation-and-interaction-tool > data > load_data Name / Date modified / Type / Size Capital / 30/01/2019 15:24 / Microsoft Excel Co... / 39,946 KB Genesee / 13/01/2019 22:38 / Microsoft Excel Co... / 33,173 KB HudsonValley / 13/01/2019 22:40 / Microsoft Excel Co... / 34,722 KB LongIsland / 13/01/2019 22:41 / Microsoft Excel Co... / 32,817 KB SONI_original / 30/01/2019 15:24 / Microsoft Excel Co... / 45,548 KB West / 13/01/2019 22:43 / Microsoft Excel Co... / 31,674 KB SONI / 16/02/2019 21:36 / Microsoft Excel Co... / 9 KB	

	If SONI.csv 'already exists rename the existing SONI.csv in the *load_data* folder to 'SONI_original.CSV' and then paste		
5	In the directory *<project-install-directory>\short-term-load-forecasting-visualisation-and-interActions-tool* Execute *python run.py*		
6	Using a web browser, navigate to web address *http://127.0.0.1:8050/*	 • No graph visible	

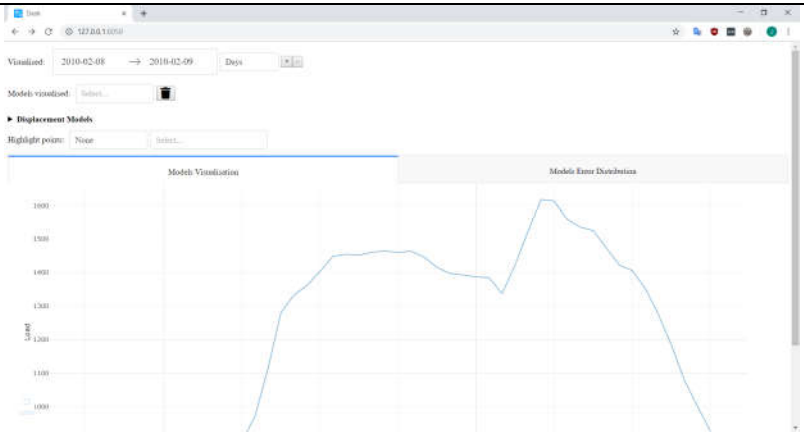
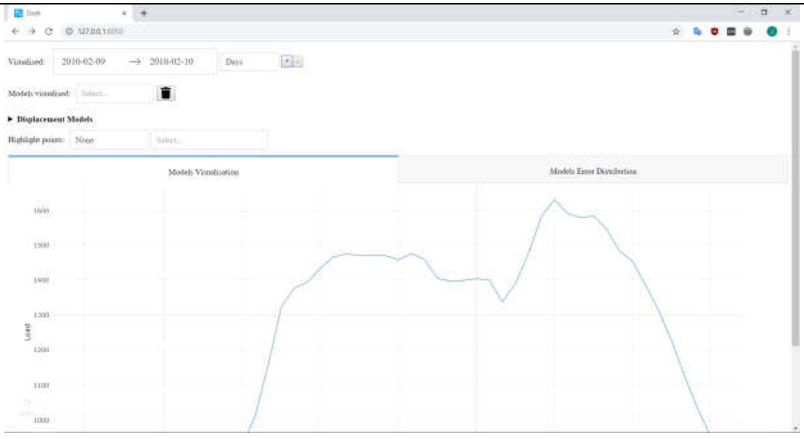
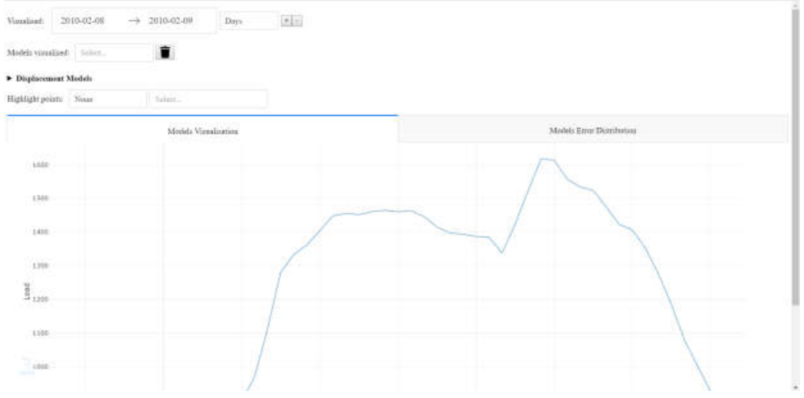
```
(load_forecasting_env) C:\Users\jrbai>cd C:\Dev\short-term-load-forecasting-visualisation-and-interaction-

(load_forecasting_env) C:\Dev\short-term-load-forecasting-visualisation-and-interaction-tool>python run.py
C:\Users\jrbai\AppData\Local\conda\conda\envs\load_forecasting_env\lib\site-packages\sklearn\externals\job
cloudpickle\cloudpickle.py:47: DeprecationWarning:

the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses

 * Serving Flask app "run" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
C:\Users\jrbai\AppData\Local\conda\conda\envs\load_forecasting_env\lib\site-packages\sklearn\externals\job
cloudpickle\cloudpickle.py:47: DeprecationWarning:

the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses

 * Debugger is active!
 * Debugger PIN: 976-193-184
 * Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```

Visualised: 2010-02-10 → YYYY-MM-DD, Days; Models visualised: Select...; Highlight points: None, Select...; Characteristics graph y-axis: None

Part 2: Load Data Visualisation

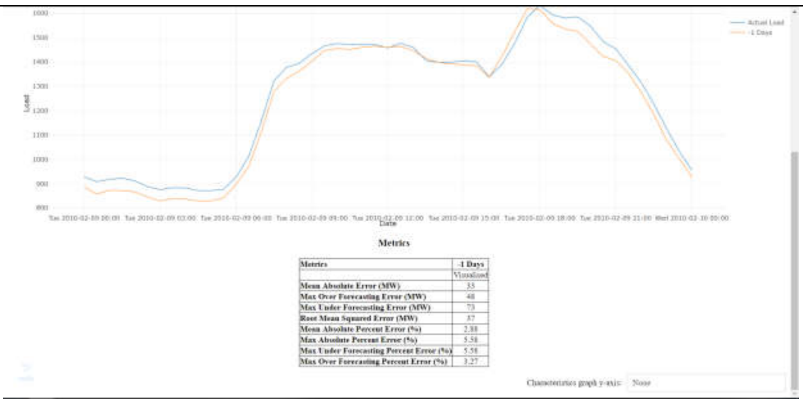
	Actions	**Expected Result**	**Done**
1	Change start date to Monday 8th February 2010	 • No graph visible	

Visualised: 2010-02-08 → YYYY-MM-DD, Days; Models visualised: Select...; Highlight points: None, Select...; Characteristics graph y-axis: None

2	Change start date to Monday 8th February 2010. Change end date to Tuesday 9th February 2010 using the visualised calendar widget	 Visualised: 2010-02-08 → 2010-02-09 Days Models visualised: Select... ▸ Displacement Models Highlight points: None Select... Models Visualisation Models Error Distribution Load • X axis labelled 'Date' • Y axis labelled 'Load' • No legend visible • Graph plotted with start date midnight point plotted (2010-02-08) and end date (2010-02-09) midnight point plotted • X axis shows the single day range (3hr increments) • No model metrics table visible below graph	
3	Click the '+' labelled button beside the calendar date selection widget	 Visualised: 2010-02-09 → 2010-02-10 Days Models visualised: Select... ▸ Displacement Models Highlight points: None Select... Models Visualisation Models Error Distribution Load • Start date in the calendar widget is 2010-02-09 and end date is 2010-02-10. • Graph plotted with start date midnight point (2010-02-09) plotted and end date midnight point (2010-02-10) plotted	
4	Click the '-' labelled button beside the calendar date selection widget	 Visualised: 2010-02-08 → 2010-02-09 Days Models visualised: Select... ▸ Displacement Models Highlight points: None Select... Models Visualisation Models Error Distribution Load	

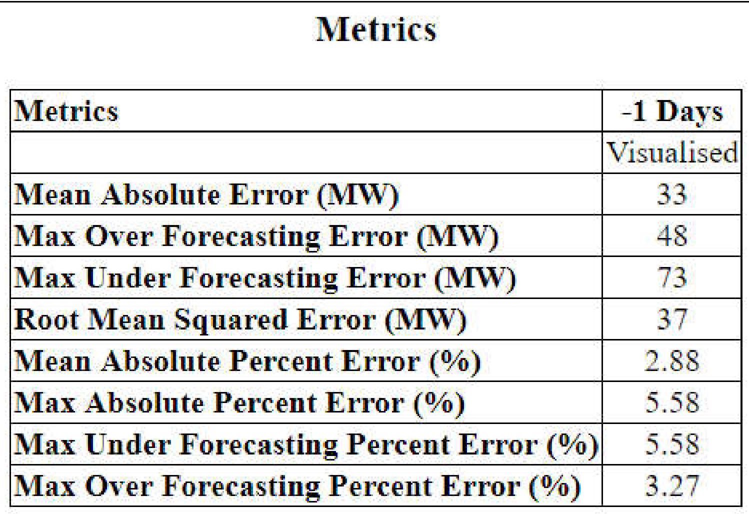
		• Start date in the calendar widget is 2010-02-08 and end date is 2010-02-09. • Graph plotted with start date midnight point (2010-02-08) plotted and end date midnight point (2010-02-09) plotted	

Part 3: Forecasting Visualisations and Statistical Metrics

	Actions	Expected Result	Done
1	Change start date to Tuesday 9th February 2010. Change end date to Wednesday 10th February 2010. Open the displacement model carat. Type in 1 to displaced by text field and select 'Days' in dropdown. Click 'Add Model' button.	 • Directly below 2010-02-09 18:00' actual load point (blue). there is a '2010-02-08 18:00' -1 days point (orange). • 'Actual Load' and '-1 Days' legends visible. • Metrics table has a 'Visualised' column. • No 'Test Data' sub column below -1 Days column	

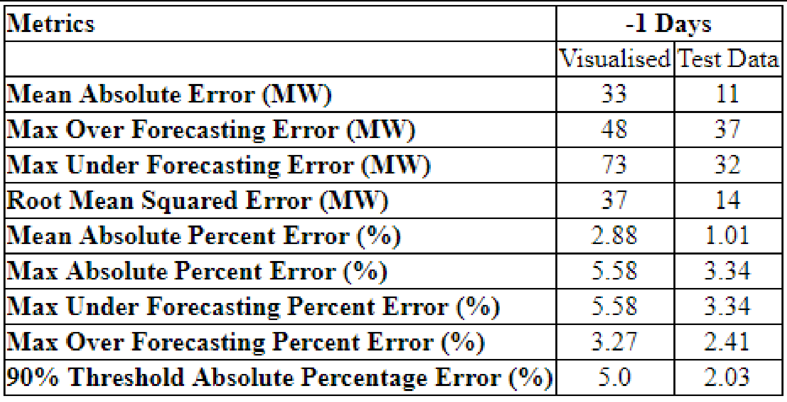
Metrics

Metrics	-1 Days
	Visualised
Mean Absolute Error (MW)	33
Max Over Forecasting Error (MW)	48
Max Under Forecasting Error (MW)	73
Root Mean Squared Error (MW)	37
Mean Absolute Percent Error (%)	2.38
Max Absolute Percent Error (%)	5.58
Max Under Forecasting Percent Error (%)	5.58
Max Over Forecasting Percent Error (%)	3.27

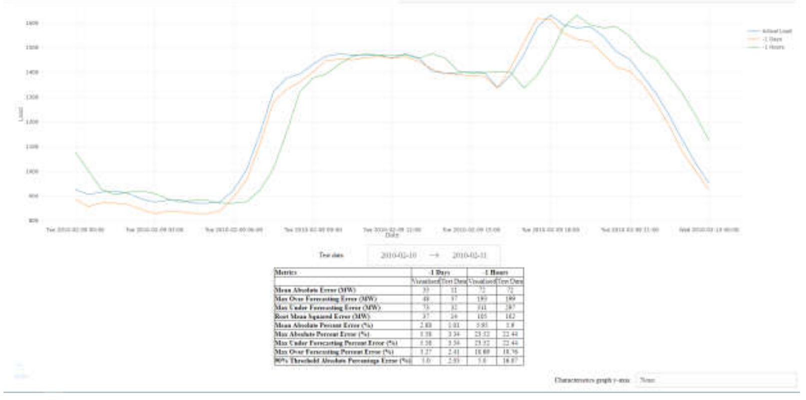
2	Change test data start date to 8th February 2010.	 • No 'Test Data' column	

Metrics

Metrics	-1 Days
	Visualised
Mean Absolute Error (MW)	33
Max Over Forecasting Error (MW)	48
Max Under Forecasting Error (MW)	73
Root Mean Squared Error (MW)	37
Mean Absolute Percent Error (%)	2.88
Max Absolute Percent Error (%)	5.58
Max Under Forecasting Percent Error (%)	5.58
Max Over Forecasting Percent Error (%)	3.27

3	Change test data start date to 10th February 2010. Change test data end date to 11h February 2010.	 • 'Test Data' column	

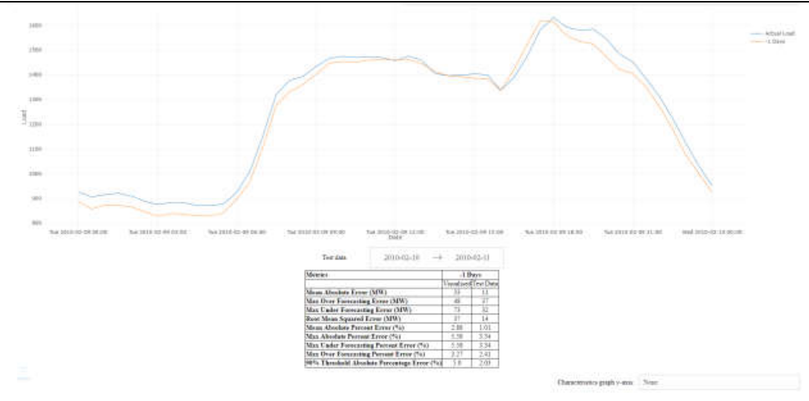
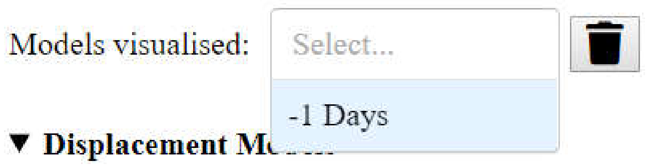
Metrics	-1 Days	
	Visualised	Test Data
Mean Absolute Error (MW)	33	11
Max Over Forecasting Error (MW)	48	37
Max Under Forecasting Error (MW)	73	32
Root Mean Squared Error (MW)	37	14
Mean Absolute Percent Error (%)	2.88	1.01
Max Absolute Percent Error (%)	5.58	3.34
Max Under Forecasting Percent Error (%)	5.58	3.34
Max Over Forecasting Percent Error (%)	3.27	2.41
90% Threshold Absolute Percentage Error (%)	5.0	2.03

4	Type in 1 to displaced by text field and select 'Hours' in dropdown. Click 'Add Model' button.	 • Directly below 2010-02-09 18:00' actual load point (blue). there is a '2010-02-08 18:00' -1 days point (orange) and '2010-02-09 17:00' -1 hours point (green) • 'Actual Load', '-1 Days' and '-1 Hours' legends visible. • Metrics table has a '-1 Hours' column with 'Visualised' and 'Test Data' sub columns	

Test data 2010-02-10 → 2010-02-11

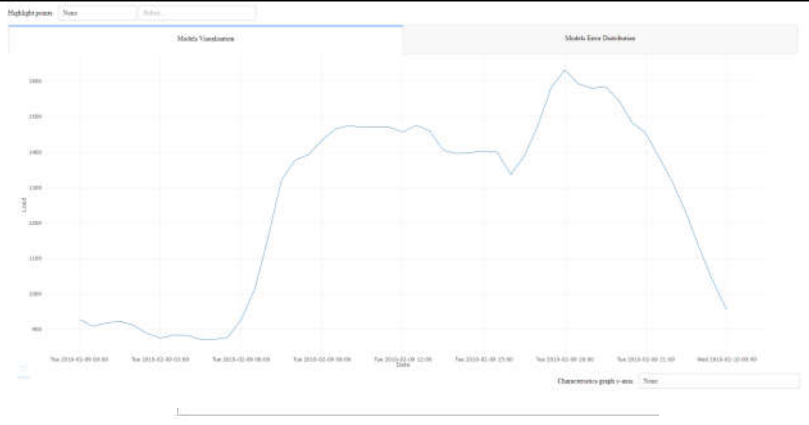
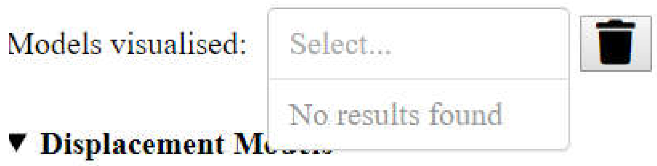
Metrics	-1 Days		-1 Hours	
	Visualised	Test Data	Visualised	Test Data
Mean Absolute Error (MW)	33	11	72	72
Max Over Forecasting Error (MW)	48	37	199	199
Max Under Forecasting Error (MW)	73	32	311	287
Root Mean Squared Error (MW)	37	14	105	142
Mean Absolute Percent Error (%)	2.88	1.01	5.95	5.8
Max Absolute Percent Error (%)	5.58	3.34	23.32	22.44
Max Under Forecasting Percent Error (%)	5.58	3.34	23.32	22.44
Max Over Forecasting Percent Error (%)	3.27	2.41	16.69	18.76
90% Threshold Absolute Percentage Error (%)	5.0	2.03	5.8	16.87

Characteristics graph y-axis None

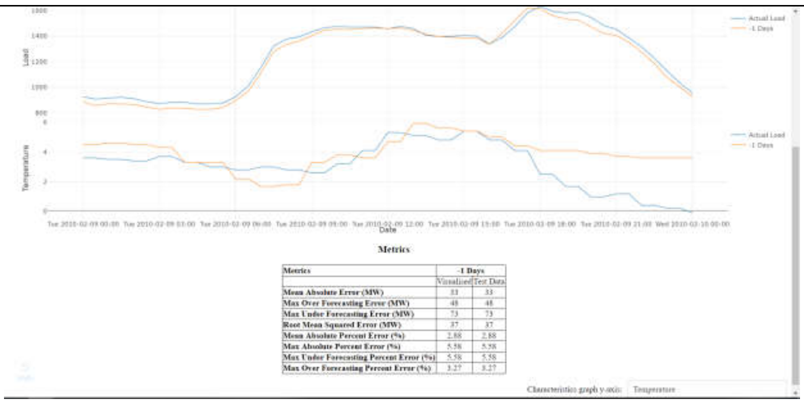
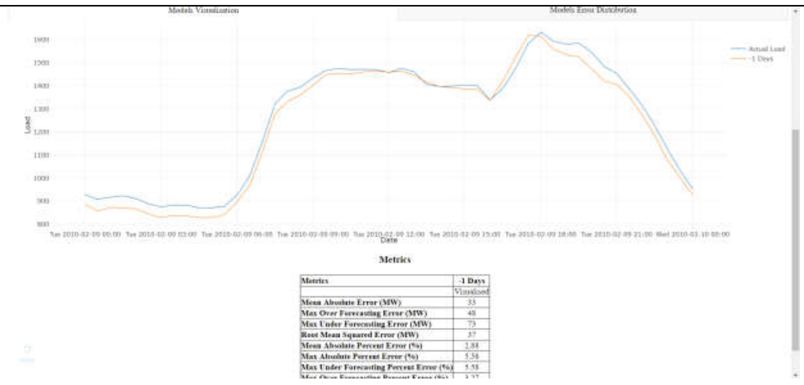
5	From models visualised dropdown select '-1 Hours' Click trash icon.	  Models visualised: Select... -1 Days ▼ Displacement M • • No -1 Hours legend • No -1 Hours green points • No -1 Hours selection in models visualised dropdown • Metrics table -1 Hours column is not visible.	

Test data 2010-02-10 → 2010-02-11

Metrics	-1 Days	
	Visualised	Test Data
Mean Absolute Error (MW)	33	11
Max Over Forecasting Error (MW)	48	27
Max Under Forecasting Error (MW)	73	32
Root Mean Squared Error (MW)	37	14
Mean Absolute Percent Error (%)	2.88	1.01
Max Absolute Percent Error (%)	5.58	3.34
Max Under Forecasting Percent Error (%)	5.58	3.34
Max Over Forecasting Percent Error (%)	3.27	2.41
90% Threshold Absolute Percentage Error (%)	3.8	2.03

6	From models visualised dropdown select '-1 Days' Click trash icon.	  Models visualised: Select... No results found ▼ Displacement M • No -1 Days legend • No -1 Days points • 'No Results Found' in models visualised dropdown • Metrics table is not visible.	

Part 4: Forecasting Model Characteristics

	Actions	Expected Result	Done
1	Change start date to Monday 9th February 2010. Change end date to Tuesday 10th February 2010. Type in 1 to displaced by text field and select 'Days' in dropdown. Click 'Add Model' button. Choose 'Temperature' in the characteristic graph y-axis dropdown	 • -1 Days legend on both graphs. • Colours for each plot (Actual Load and -1 Days) are the same in the two visualised graphs. • Gridlines line up vertically. • Model visualisation (top) graph has a Load labelled y axis and does not have a labelled x axis. • Characteristics graph has a Temperature labelled y axis and a Date labelled x axis.	
2	Choose 'None' in the characteristic graph y-axis dropdown	 • Characteristics graph is not visible	

Metrics

Metrics	-1 Days	
	Visualised	Test Data
Mean Absolute Error (MW)	33	33
Max Over Forecasting Error (MW)	48	48
Max Under Forecasting Error (MW)	73	73
Root Mean Squared Error (MW)	37	37
Mean Absolute Percent Error (%)	2.88	2.88
Max Absolute Percent Error (%)	5.58	5.58
Max Under Forecasting Percent Error (%)	5.58	5.58
Max Over Forecasting Percent Error (%)	3.27	3.27

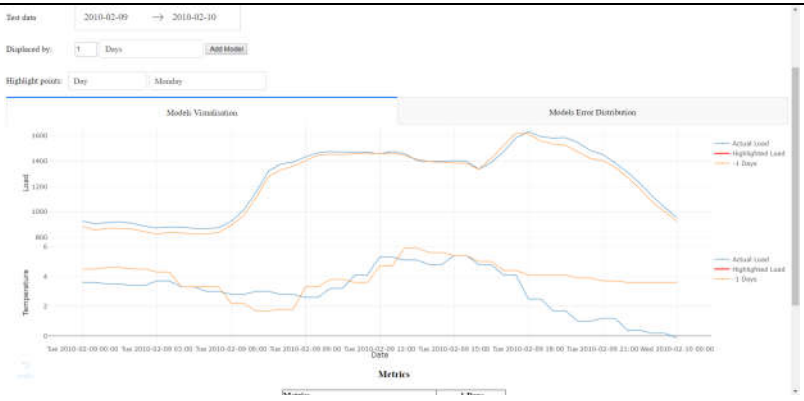
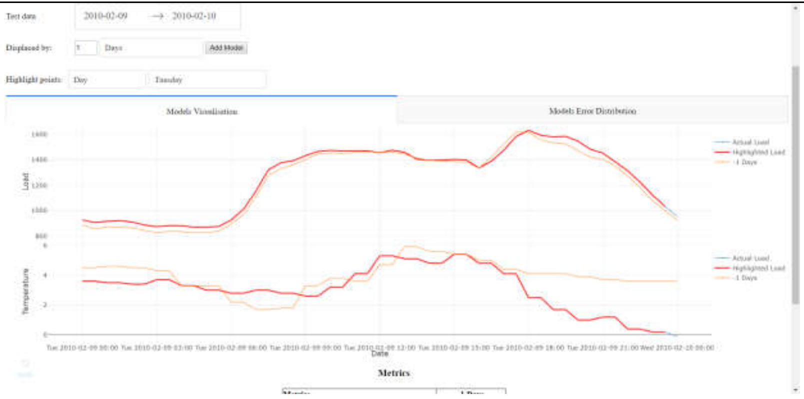
Characteristics graph y-axis: Temperature

Metrics

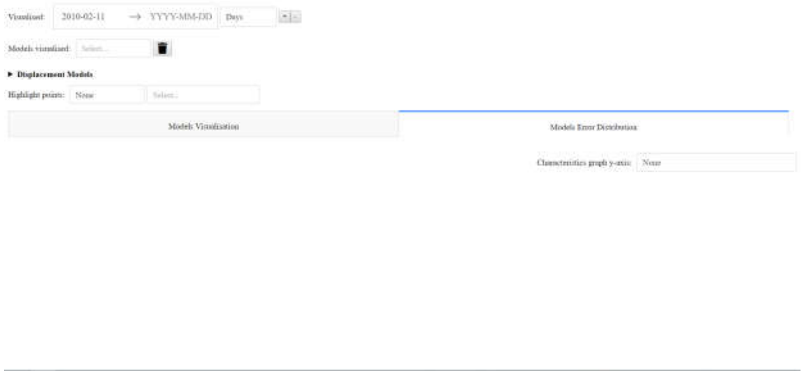
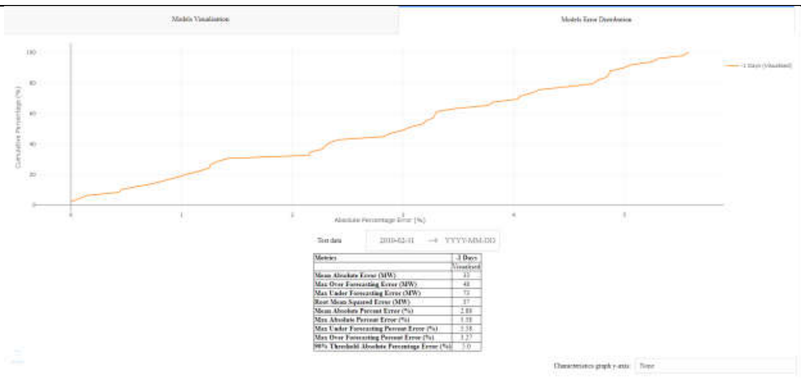
Metrics	-1 Days
	Visualised
Mean Absolute Error (MW)	33
Max Over Forecasting Error (MW)	48
Max Under Forecasting Error (MW)	73
Root Mean Squared Error (MW)	37
Mean Absolute Percent Error (%)	2.88
Max Absolute Percent Error (%)	5.58
Max Under Forecasting Percent Error (%)	5.58
Max Over Forecasting Percent Error (%)	3.27

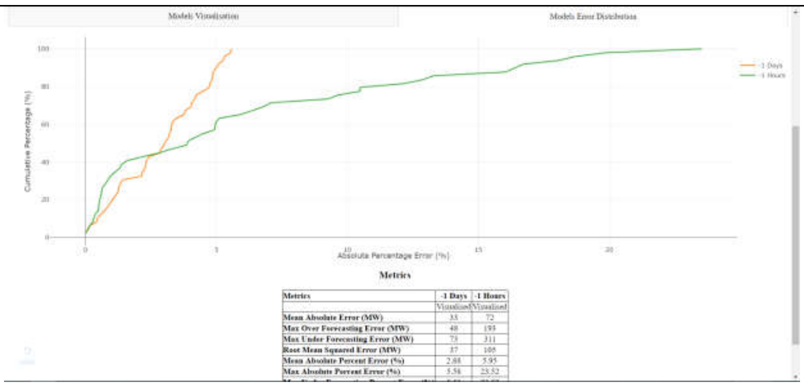
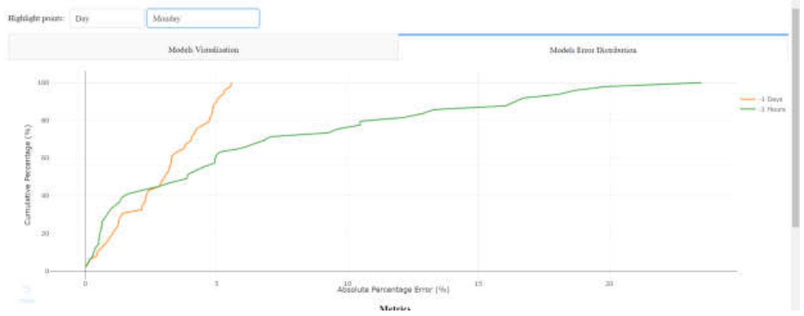
		• Graph has returned to its original height.	

Part 5: Highlighting Data Points

	Actions	Expected Result	Done
1	Choose ‘Temperature’ in the characteristic graph y-axis dropdown Choose ‘Day’ and ‘Monday’ in highlight points dropdowns	 Test data: 2010-02-09 → 2010-02-10 Displaced by: 1 Days Add Model Highlight points: Day Monday Models Visualisation Models Error Distribution Load Temperature Date Actual Load Highlighted Load -1 Days Metrics • No additional plots on the two visualised graphs. • ‘Highlighted Load’ legend added to both graphs	
2	Choose ‘Day’ and ‘Tuesday’ in highlight points dropdowns	 Test data: 2010-02-09 → 2010-02-10 Displaced by: 1 Days Add Model Highlight points: Day Tuesday Models Visualisation Models Error Distribution Load Temperature Date Actual Load Highlighted Load -1 Days Metrics • ‘Highlighted Load’ legend added to both graphs • Red line plot superimposing over all 2010-02-09 day (blue) points. • No red line plot superimposing -1 Day model (orange) points.	

Part 6: Model Error Distribution Graph

	Actions	Expected Result	Done
1	Click on 'Models Error Distribution'	 Visualised: 2010-02-11 → YYYY-MM-DD Days Models visualised: Select... ▸ Displacement Models Highlight points: None Select... Models Visualisation | Models Error Distribution Characteristics graph y-axis: None • No graph visualised	
2	Change start date to Tuesday 9th February 2010. Change end date to Wednesday 10th February 2010. Type in 1 to displaced by text field and select 'Days' in dropdown. Click 'Add Model' button.	 Test data 2010-02-11 → YYYY-MM-DD Metrics | -1 Days Visualised Mean Absolute Error (MW) | 33 Max Over Forecasting Error (MW) | 48 Max Under Forecasting Error (MW) | 73 Root Mean Squared Error (MW) | 37 Mean Absolute Percent Error (%) | 2.88 Max Absolute Percent Error (%) | 5.38 Max Under Forecasting Percent Error (%) | 5.38 Max Over Forecasting Percent Error (%) | 3.27 90% Threshold Absolute Percentage Error (%) | 5.0 Characteristics graph y-axis: None • Model error distribution graph is visible • -1 Day (Visualised) legend is visible • Y axis labelled 'Cumulative Percentage (%)' • X axis labelled 'Absolute Percentage Error (%)' • Range of Y axis 0-100 • '-1 Days' colour (orange) is the same in the model visualisation and characteristics graph	

3	Type in 1 to displaced by text field and select ‘Hours’ in dropdown. Click ‘Add Model’ button.	 Models Visualisation Models Error Distribution Cumulative Percentage (%) Absolute Percentage Error (%) -1 Days -1 Hours • Model error distribution graph is visible • -1 Day and -1 Hours legend is visible • Range of Y axis 0-100 • ‘-1 Days’ colour (orange) and ‘-1 Hours’ colour (green) is the same in the model visualisation and characteristics graph	
4	Choose ‘Day’ and ‘Monday’ in highlight points dropdowns	 Highlight points: Day Monday Models Visualisation Models Error Distribution Cumulative Percentage (%) Absolute Percentage Error (%) -1 Days -1 Hours Metrics • No ‘Highlighted’ legend • No red highlighted point plots on graph	

Metrics

Metrics	-1 Days	-1 Hours
	Visualised	Visualised
Mean Absolute Error (MW)	33	72
Max Over Forecasting Error (MW)	48	193
Max Under Forecasting Error (MW)	73	311
Root Mean Squared Error (MW)	37	105
Mean Absolute Percent Error (%)	2.88	5.95
Max Absolute Percent Error (%)	5.58	23.52

Appendix I

Industrial User

As the dataset for the solution is from SONI, employees of SONI whose role is to produce accurate load forecasts a day ahead are a ‘real’ industrial user that the solution targets.

Motivation for using the solution:

Industrial users can use the solution to generate a day ahead load forecast for a chosen day using a forecast through prior analysis to perform optimally for that type of day. For a system operator they can use this forecast to buy electricity from a day ahead market. The greater the accuracy in the load forecast to the real system demand the more valuable the solution is to them.

What would they do if this solution didn’t exist:

Industrial users can use data analysis and visualisation environments to perform the same operations as an academic user. Crucially, they would have to manually choose a model identified through their own individual analysis to have the best performance for the type of day they are forecasting as the day ahead model forecast.

Value gained by using the solution:

The solution enables the industrial user to generate an accurate day ahead load forecast.

User Flow

The expected flow of the industrial user using the solution is visualised below:

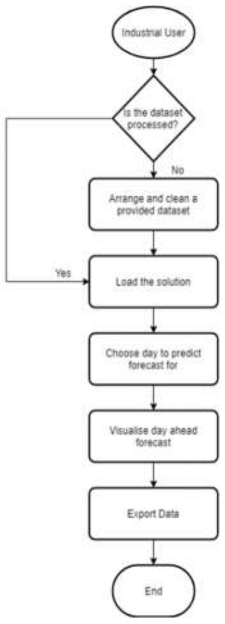

Industrial User
Is the dataset processed?
No
Yes
Arrange and clean a provided dataset
Load the solution
Choose day to predict forecast for
Visualise day ahead forecast
Export Data
End

Figure 1 *The expected flow an industrial user using the solution.*

Use Cases

- Arrange and clean a provided dataset through an automated step-by-step process.
- Visualise the day ahead load forecast for a chosen given day.
- Export the forecasted load values as their guide forecast.

The two identified target users can be considered 'actor's with behaviours associated with them. A 'generic user' that has the shared behaviour of industrial users and academic user is identified to prioritise these for development as implementing them provides value to all targeted users.

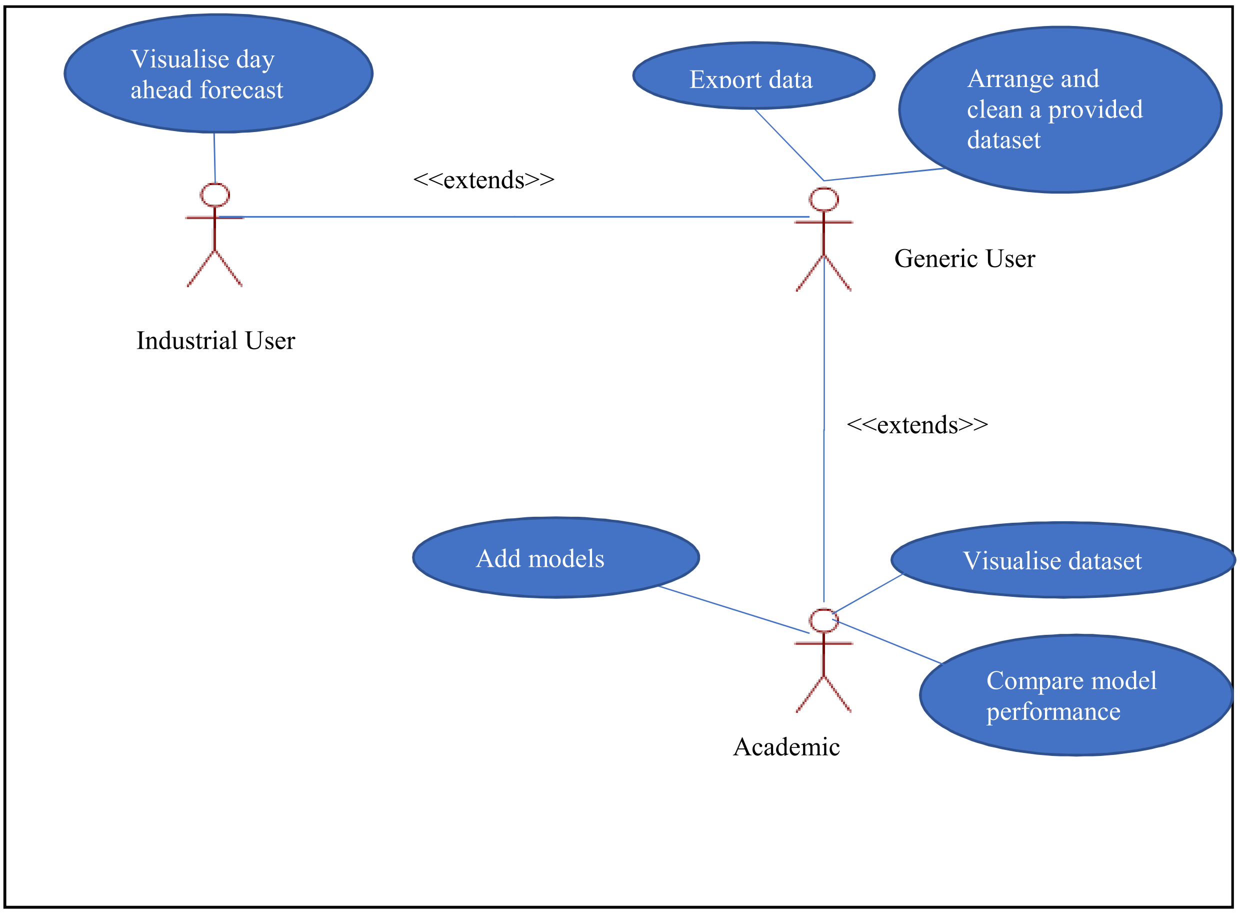


Figure 2 *A UML representation of the actors and the behaviours associated with the actor. The industrial user and academic user inherit the behaviour the generic user has.*

User Interface

The following user interface provides the industrial user a professional interface that produces an accurate day ahead forecast for a given day. The interface has fewer interactive elements than the user interface for an academic user as the expectation is industrial users only need the system's result, and hence do not need to follow the steps to construct a load forecasting model or explore the load dataset. These processes have already been performed in **chapter 2** to create performant models, which are included to be used for the day ahead forecast.

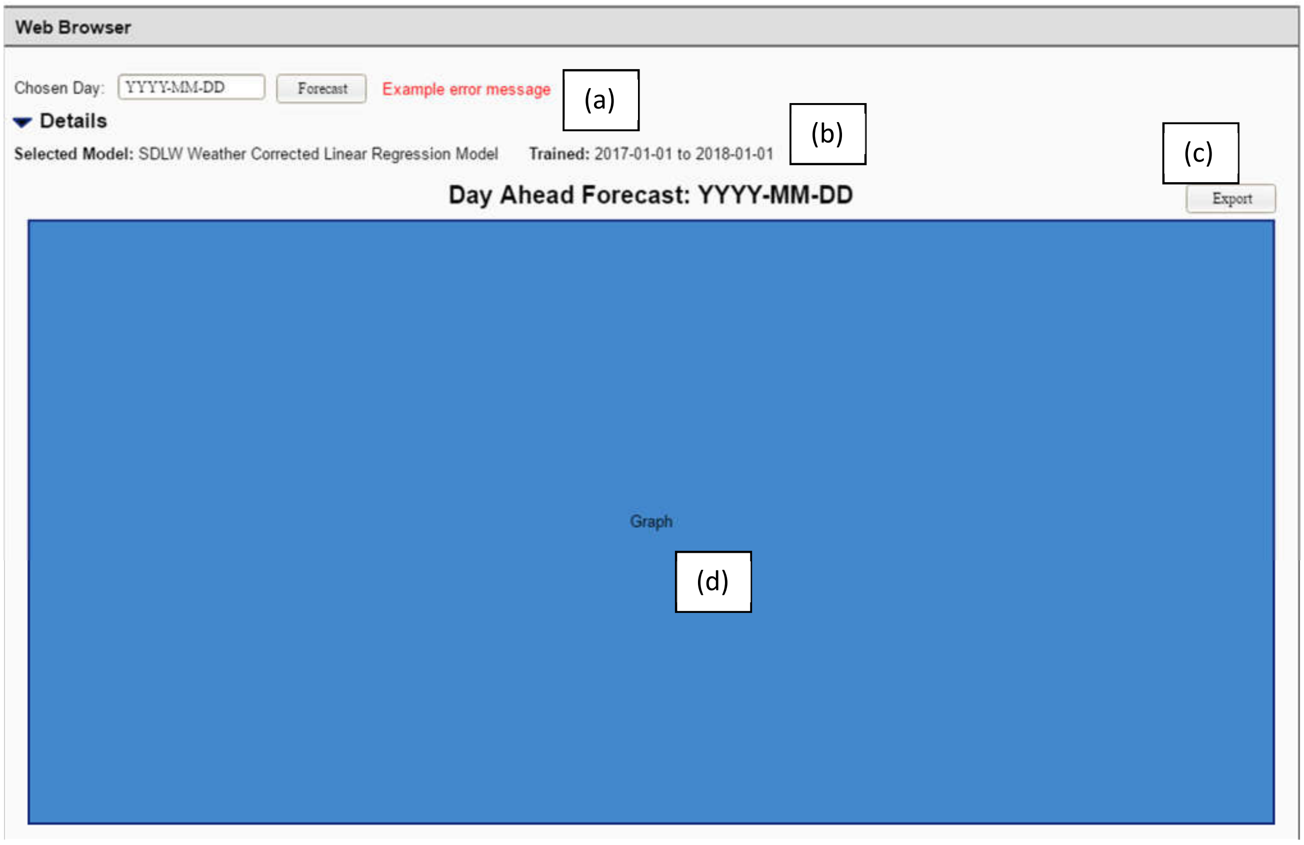

Web Browser
Chosen Day:
YYYY-MM-DD
Forecast
Example error message
(a)
Details
Selected Model: SDLW Weather Corrected Linear Regression Model
Trained: 2017-01-01 to 2018-01-01
(b)
(c)
Day Ahead Forecast: YYYY-MM-DD
Export
Graph
(d)

Figure 3 Prototype of the industrial user's user interface

(a) Controls to choose a given day for a day ahead forecast

The user can choose the specific day they want to forecast a day ahead forecast using the date picker. The date picker date format follows the YYYY-MM-DD standard set in ISO 8601 [10], which provides a standardized way of presenting dates and times which is internationally agreed. This makes the date picking functionality interpretable by industrial users of different nationalities, which is increasingly common within a multinational company. Upon clicking the 'Forecast' button, the visualisation is visible, otherwise a red error message is visible. The red error message will provide the user a short single-line explanation of why a day ahead forecast could not be produced for the day they have chosen. This error message is not technical as the industrial user is not expected to know implementation details of the system.

(b) Chosen forecasting model

The user can optionally view the details of the forecasting model the solution has chosen for the day ahead forecast by expanding the 'Details' carat. This enables the user to record the model and training data it was built upon to assess whether it performed optimally when they have the data to test the load forecast. They can use the performance of the forecasting model to enhance their decision making in using the system as a tool to make future day ahead forecasts.

(c) Export button

By clicking this button, the user can download the datetime and raw load values from the generated day ahead forecast, along with the system's graphical visualisation, to their local machine as required in **FTR05.**

(d) Graph visualisation

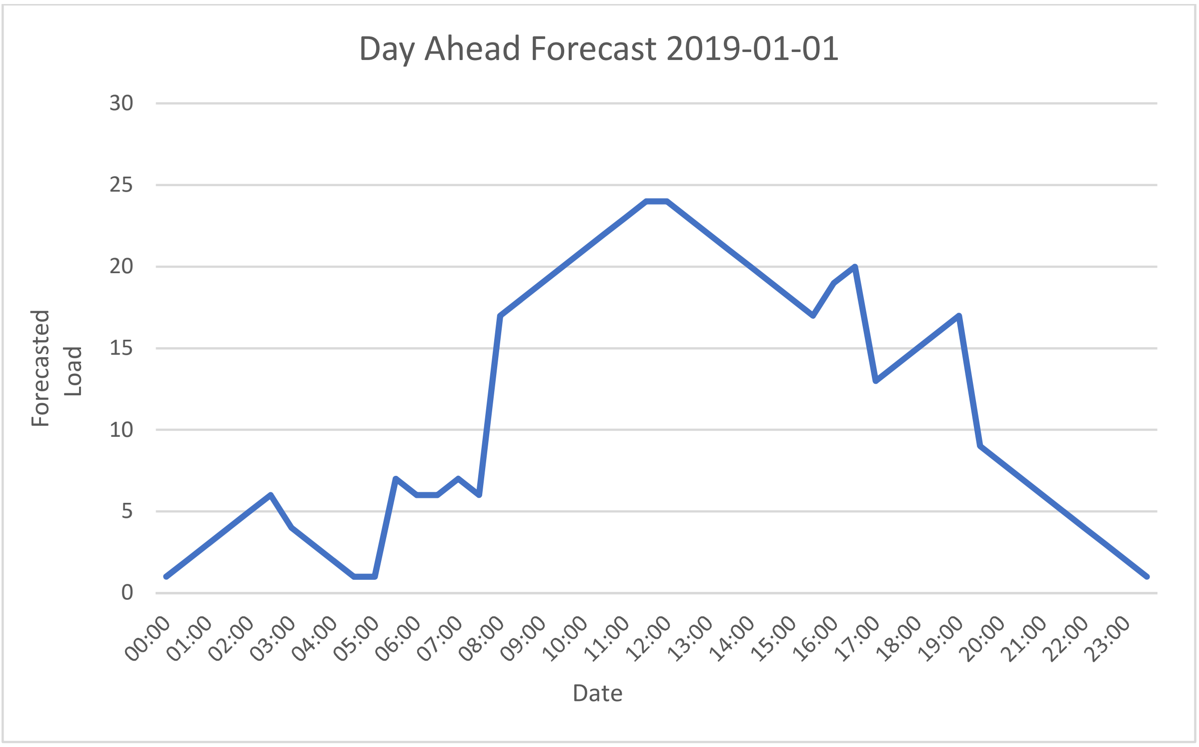


Figure 4 Day ahead forecast mock-up.

The day ahead forecast graph in **Figure 4** only contains the information the industrial needs to know: what the system demand is forecasted to be at each time of the day. They can use this visualised forecast to enhance their decision making when, for example a system operator such as SONI, bidding in a market for electricity. The visualisation is titled to clearly identify the day it is forecasting for. This is important to the industrial user for as an expectation is that they will want to compare and evaluate the performance of the saved system's forecasted load values when the actual load values for the day are finalised.