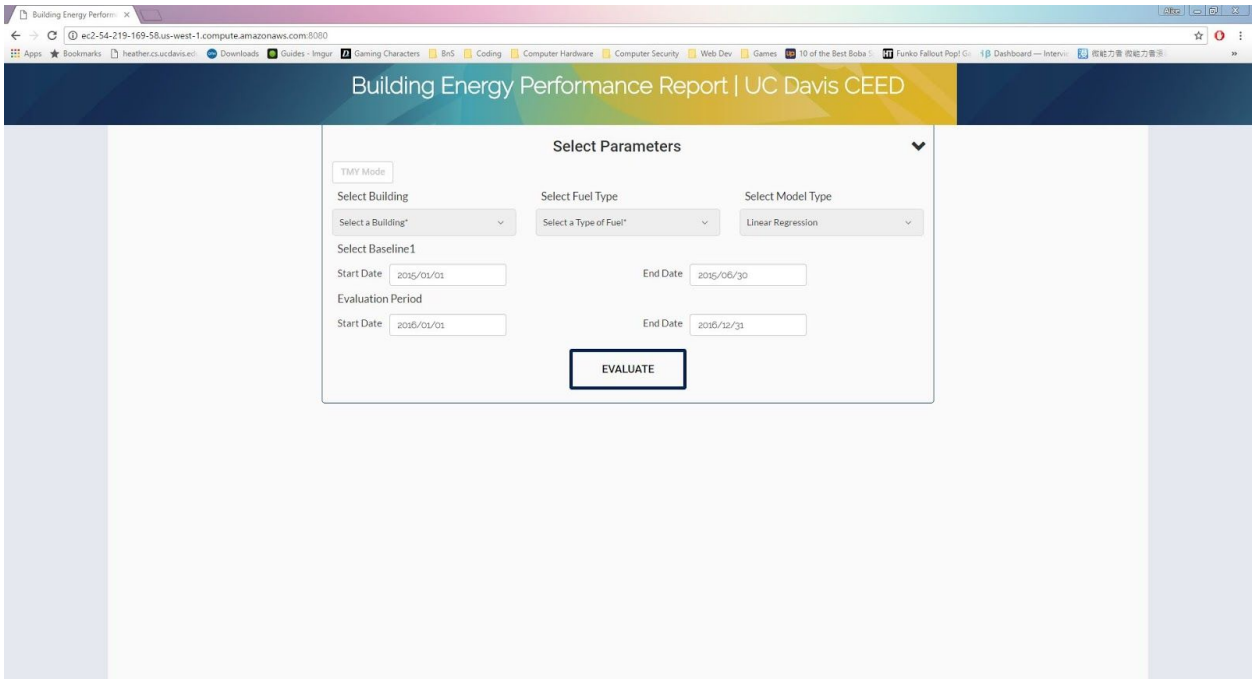
ECS 193:
Building Energy Performance
User Guide

Introduction

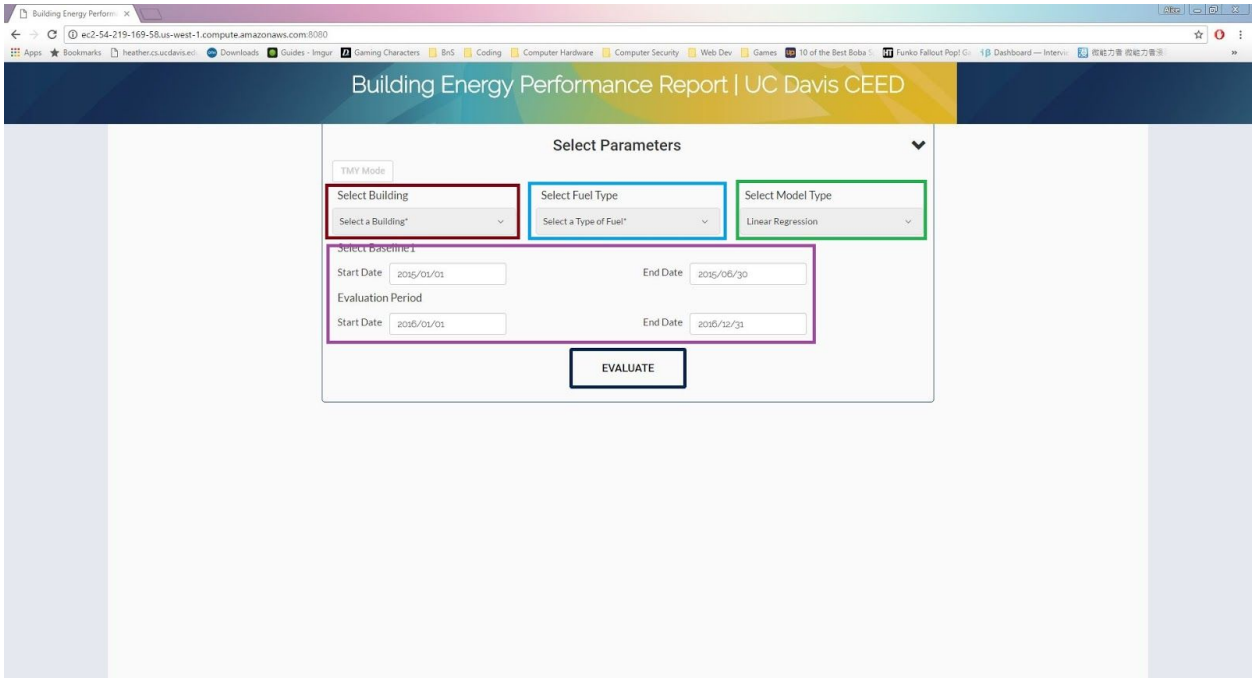
Our Python web application allows mechanical engineers at the UC Davis Energy Conservation Office to analyze the energy performance of UC Davis buildings. Our application includes a simple user interface that allows users to easily select the building, energy type, model type, and dates to perform analyses and predict energy usage. Users will be able to easily generate numbers such as r^2 scores and savings, as well as plots to compare data between specified dates. The simple interface of our application will also allow individuals such as other engineers and managers to analyze energy usage with little to no knowledge of how the specific code is written. Our application aims to increase usability and efficiency of analyzing energy usage in UC Davis buildings.

Using the Application

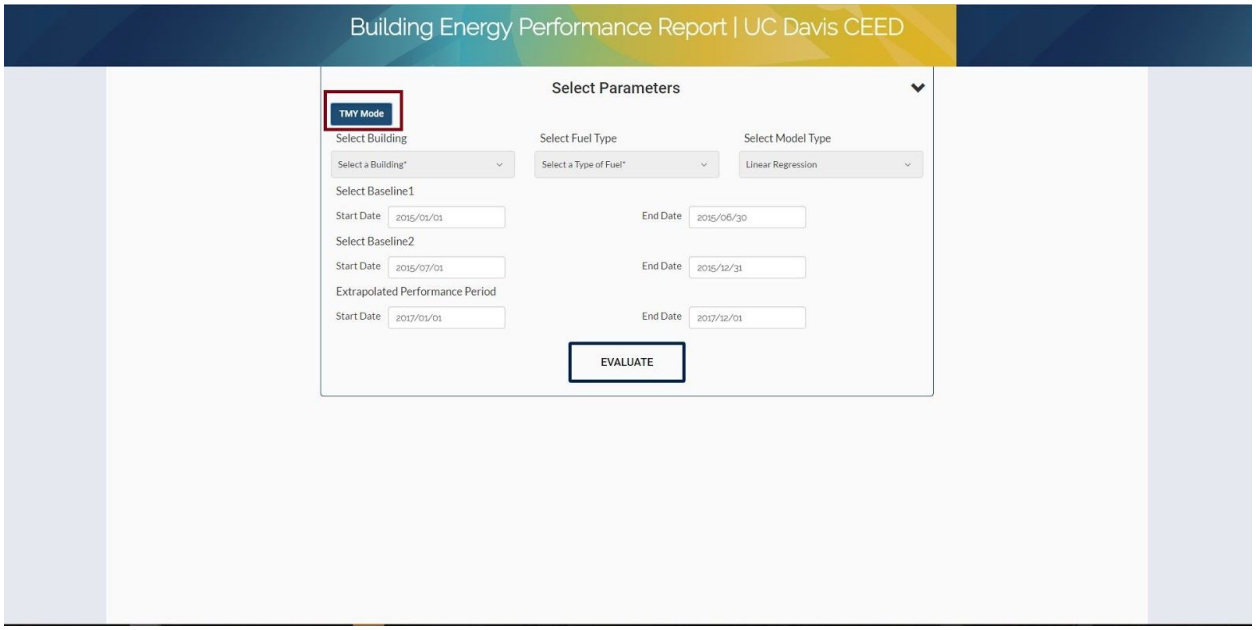
Go to http://ec2-54-219-169-58.us-west-1.compute.amazonaws.com:8080/ to access the application. Below is the homepage of the application.



Select the building, fuel type, dates, and model type from the drop down menu

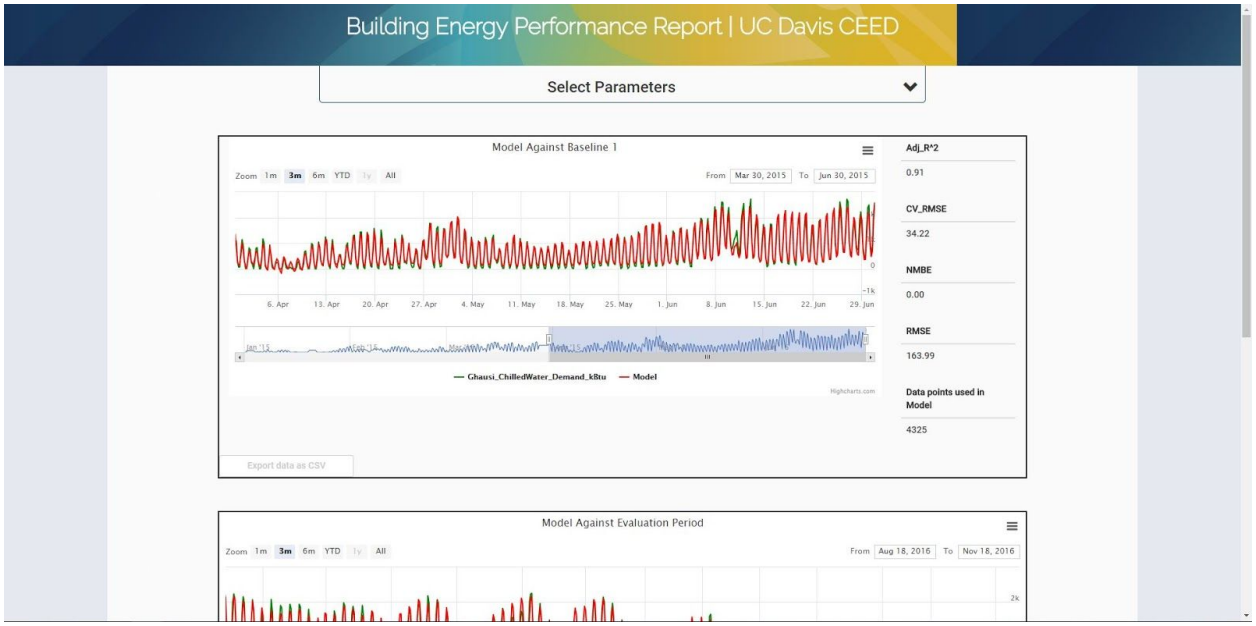


There is a TMY option for the user to use. Selecting this mode will show additional parameters for the user to select.

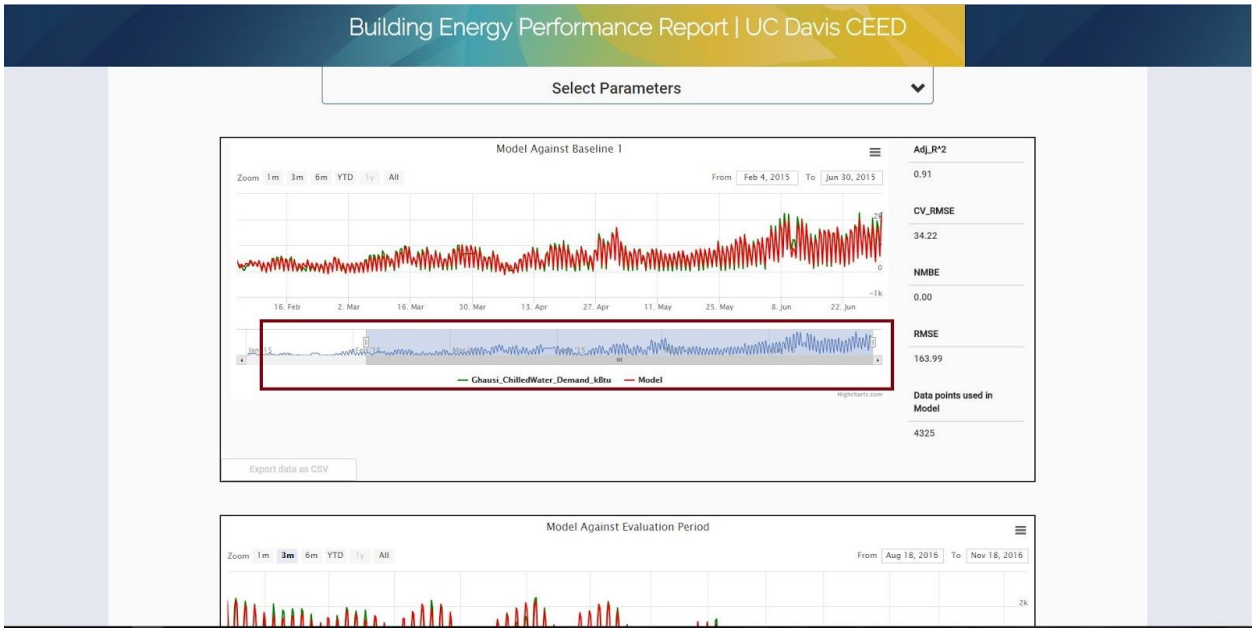


Click evaluate to generate charts.

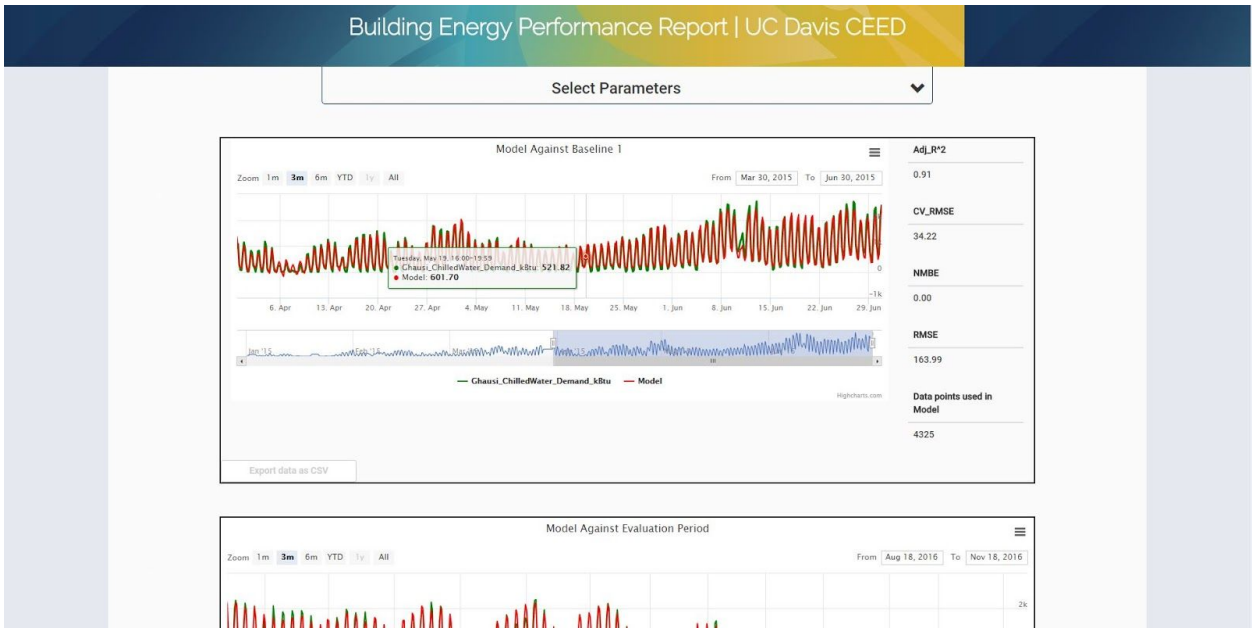
Scrolling down, you will see the charts generated from your desired parameters



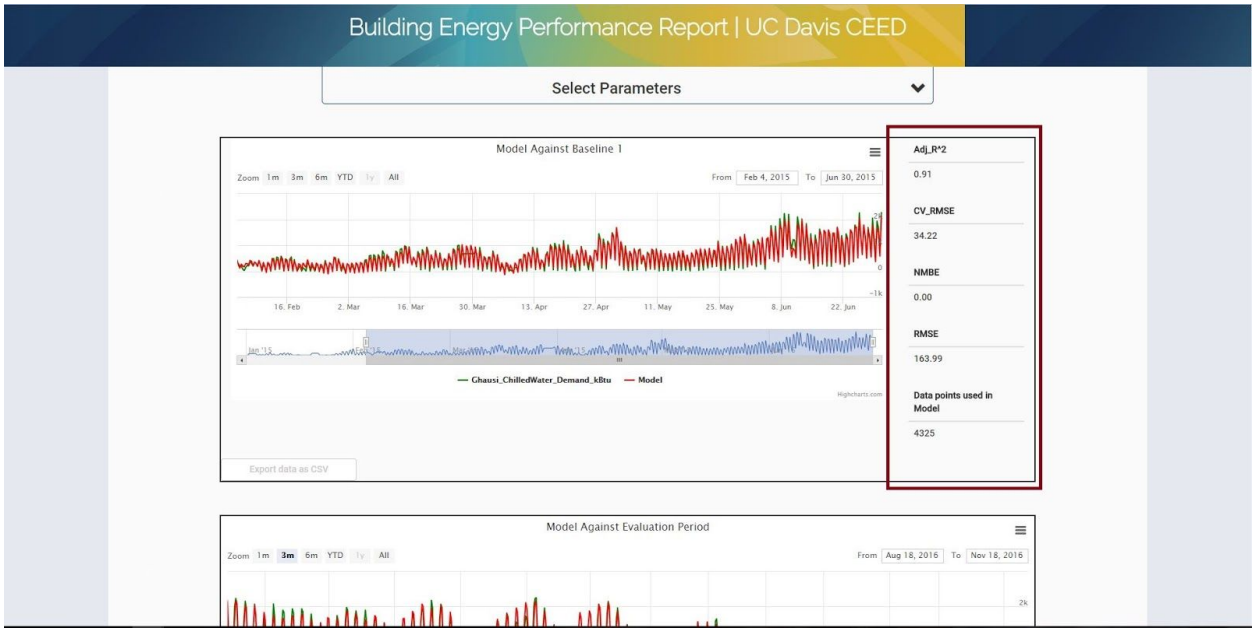
Moving the sliders under the graph will allow the user to change the time periods that the associated graph is displaying.



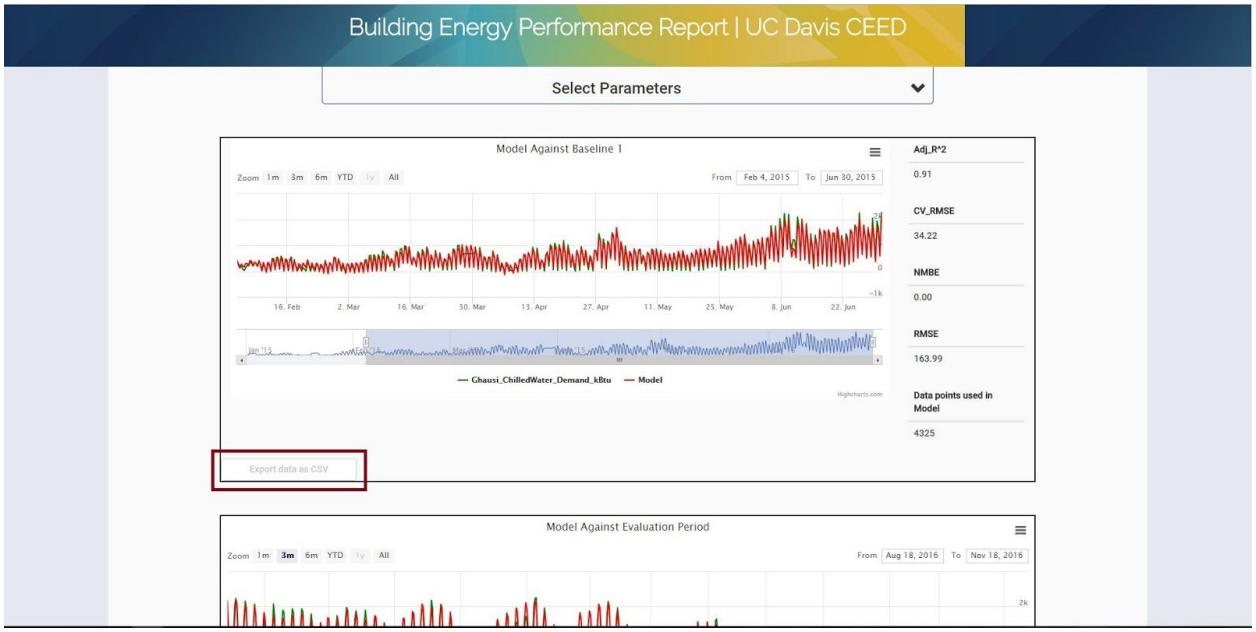
Hovering over the graph will show the data at the specified date and time.



Relevant data values and calculations are displayed to the side of the graph.



There is an “Export data as CSV” option for each graph. Clicking this will download the graph’s data in the form of a CSV file.



Exported data

A1 Timestamps

	A	B	C	D	E	F
1	Timestamps	Ghausi_ChilledWater_Demand_kBtu	Model			
2	1.42007E+12	0.22507052	247.3869464			
3	1.42007E+12	0.22507052	64.11020854			
4	1.42008E+12	0.22507052	29.69861802			
5	1.42008E+12	0.22507052	160.5496345			
6	1.42008E+12	0.22507052	80.55165347			
7	1.42009E+12	0.22507052	22.83966512			
8	1.42009E+12	0.22507052	2.27295786			
9	1.4201E+12	1.8274005	-110.3667256			
10	1.4201E+12	1.8274005	-150.4338822			
11	1.4201E+12	1.8274005	-254.6812771			
12	1.42011E+12	1.8274005	-260.7654781			
13	1.42011E+12	1.8274005	-281.7642705			
14	1.42011E+12	1.8274005	-278.8713206			
15	1.42012E+12	1.8274005	-268.7901173			
16	1.42012E+12	1.8274005	-272.5039634			
17	1.42012E+12	1.8274005	-336.5899985			
18	1.42013E+12	1.8274005	-323.8596542			
19	1.42013E+12	1.8274005	-186.1559735			
20	1.42014E+12	1.8274005	-92.93167317			
21	1.42014E+12	1.8274005	-10.94608905			
22	1.42014E+12	155.1125651	43.26356519			
23	1.42015E+12	155.1125651	72.94871735			
24	1.42015E+12	155.1125651	85.84762874			
25	1.42015E+12	155.1125651	130.6513789			
26	1.42016E+12	155.1125651	107.966587			
27	1.42016E+12	155.1125651	-84.61866686			
28	1.42016E+12	155.1125651	-143.1189979			
29	1.42017E+12	155.1125651	-65.41409093			
30	1.42017E+12	155.1125651	-120.8460089			
31	1.42017E+12	155.1125651	-221.8306916			
32	1.42018E+12	155.1125651	-252.1918555			
33	1.42018E+12	155.1125651	-283.184382			
34	1.42019E+12	155.1125651	-317.950844			
35	1.42019E+12	155.1125651	-341.1146183			
36	1.42019E+12	155.1125651	-342.9618624			
37	1.4202E+12	155.1125651	-368.8408447			

Model Against Baseline 1

Server Setup and Installation

Our app runs using a simple Node.js server with the Express framework, with the underlying Python modeling code done in an Anaconda 2 environment. To setup the app, these dependencies must be installed.

Assuming a Linux environment with command line access, here are how to install these dependencies

Anaconda 2
Download the installer using

wget https://repo.continuum.io/archive/Anaconda2-4.3.1-Linux-x86_64.sh

We used the 4.3.1 version of Anaconda during our development, though if future development requires a different or newer version, simply choose the correct version from here https://repo.continuum.io/archive.
Once downloaded, install it using

bash Anaconda2-4.3.1-Linux-x86_64.sh

After installing, you should have the folder 'anaconda2' in your current directory.

Node.js
Node.js has a guide for installing via a command line here
https://nodejs.org/en/download/package-manager/#debian-and-ubuntu-based-linux-distributions
It should be noted that at the time of development, Node 4.4.5 was the recommended stable version for install, though at the time of writing with the release of Node 8, the stable version has moved to Node 6.10.3. This new version of node should still work, but since we haven't upgraded yet we can't say for certain.

Now would be the time to download the source for our app, which includes the Python code and the code for our webpage. These files can be sent to the server through some file transfer protocol, or you can clone the GitHub project directly.

Node.js Packages
Our app requires the use of two Node packages that need to be installed, Express for the server framework and python-shell for the ability to run our Python scripts programmatically. Navigate to the root directory of the building-energy project and run the following two lines to install these.

npm install express
npm install python-shell

With this completed, you should be able to run the app. Move to the src directory in the building-energy folder and type 'node server.js' to run the server. You should see a message saying 'Server running at port 8000' by default. If you attempt to connect to the server now, you

should see the homepage, but if you hit the evaluate button there should be an error in spawning the Python process because there is a last bit of configuration needed to setup the python-shell.

Open server.js in an editor, and in the ‘options’ object in the module requires change the ‘pythonPath’ attribute to the correct path for the python executable in your newly installed Anaconda distribution. This executable should be in the anaconda2/bin/ folder, so your path should look like ‘../../anaconda2/bin/python’.

Code Overview

File Structure

```
building-energy-master
+---data - *.csv data for python code
+---node_modules - node modules from npm
+---old - older version files from project start
+---src - src folder containing python code
\---web interface demo v1
      +---css
      +---fonts
      +---image
      \---js - contains scripts used by webpage
```

Python

The mv_model.py file is where the code queries the database for data, creates the models using the data, and calls the necessary functions to generate the data for the graphs. Notable libraries that it uses are logging, argparse, numpy, yaml, and sklearn.

The create_models function is where the code processes the data from the database after retrieving it. Arguments/parameters supplied by the user from the command line is also parsed. By calling functions from the DataPreprocessor class, the code will clean, slice, and format the data so models can be created. The finished model's data is printed. Logging is also used to prevent errors from being printed and affecting any data that would also be printed.

The DataSet class that takes the queried data and organizes it.

The Model class creates the model using the organized data. Depending on whether the user wants a Linear Regression model or a Random Forest model, the function for the appropriate model type is called and the model is initialized. In the train function, the data is fitted and the scores are calculated. In the project function, the model's predict function is used on the time period specified by the evaluation data. The savings data is also calculated here. In the predict function, we predict actual data. In the calc_scores function, the r^2 scores of the data values are calculated and returned. In the output function, the data is printed as json.

The _StreamWriter and _InfoFilter classes are used for logging purposes.

Functions for preprocessing and formatting the data as well as a function for parsing command line arguments are located in mv_model. This is to improve the readability of the code.

The pi_datalink.py and preprocessor.py files provide the functions necessary to query data from the database and do the actual cleaning of the data.

The profile.py file is used to test the run time of the code and get_data contains functions for retrieving data locally and from server.

Web

To communicate with the webpage, any standard output of the program is redirected to the Node server, which sends an array of strings as an http response where each entry is an instance of something being printed.

The Javascript for the web page makes use of the HighCharts library to produce its graphs, particularly the HighStock chart. https://www.highcharts.com/docs

Index.js runs on page start, setting up form input defaults, style wrappers for form elements, and button handlers.

Evaluate.js defines the onclick handler for the evaluate button, which gathers the arguments for the Python script and makes the AJAX request to the server, receives and parses the data and statistics for the charts, then displays them.

Other js files are included to do small worker tasks like parsing/encoding CSV files and performing some longer calculations.

The web server has fairly basic configuration settings. All files within the ‘web interface v2’ folder are served statically as no sensitive information is kept in this folder. There are only two unique requests at the time of writing, the default ‘/’ route is routed to the index.html file, and the ‘python.json’ request which is accessed by the evaluate action and calls the Python modeling script.