Big Data Energy Services Analytics Portal

Documentation

Author: Jonathan Bennett, Kono Analytics

Version: 1.0

Date: 10/5/2017

This Document applies to version 1.0 of the BDES Application portal located at http://52.55.228.166/

Contents

[User Administration 3](#_Toc494972420)

[Adding a User 3](#_Toc494972421)

[Changing a User’s Password or Making the User an Administrator 3](#_Toc494972422)

[Application Navigation 5](#_Toc494972423)

[Hourly Generation 5](#_Toc494972424)

[Hourly Cash Flow 7](#_Toc494972425)

[Monthly Cash Flow 8](#_Toc494972426)

[Cash Waterfall 9](#_Toc494972427)

[Performance Ratio 11](#_Toc494972428)

[Revenue Ranking 12](#_Toc494972429)

[Solar Ranking 13](#_Toc494972430)

[Bokeh Demo Graphs 13](#_Toc494972431)

[Data Flow 13](#_Toc494972432)

[Notes 14](#_Toc494972433)

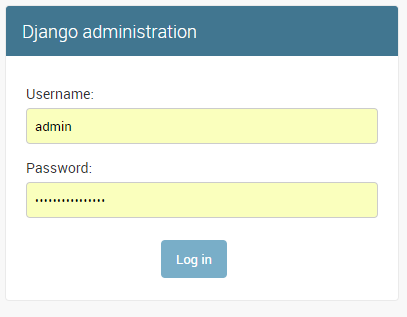
[To Do 14](#_Toc494972434)

[Possible Add-Ons 14](#_Toc494972435)

# User Administration

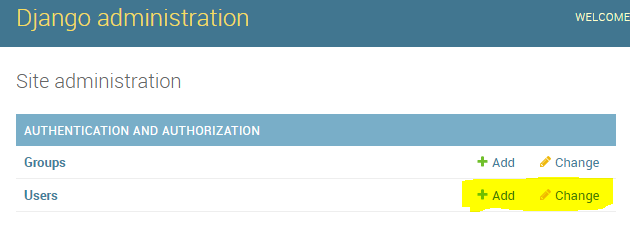
Only users with Administrator access can create and edit users. The administrative panel is used to make these changes:

<http://52.55.228.166/admin>



## Adding a User

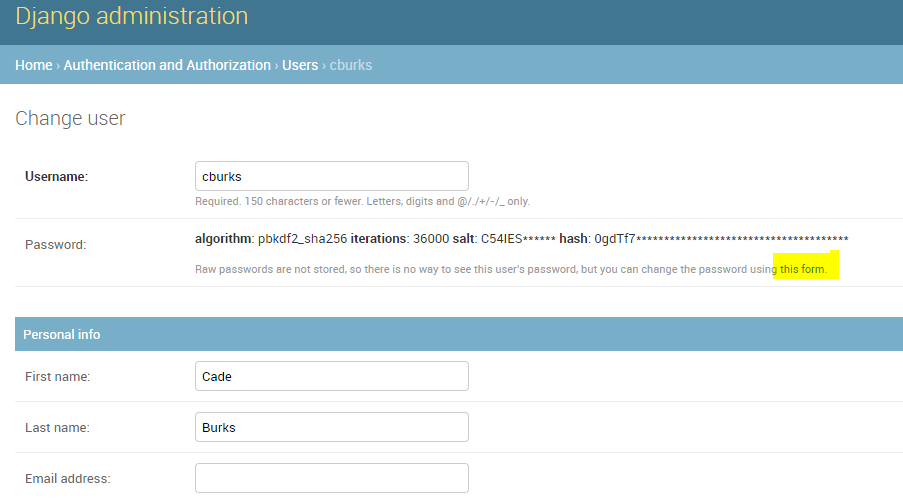
After logging in you can add here:



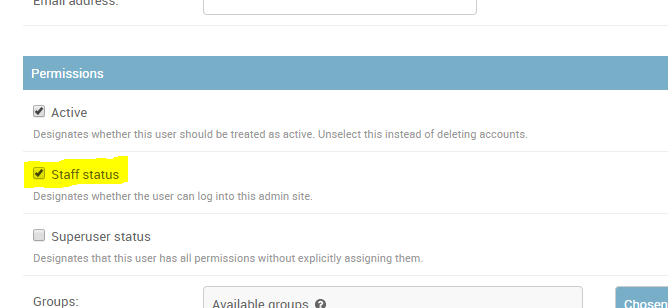
## Changing a User’s Password or Making the User an Administrator

From the main panel, click “Change” under “Users” and then click the account to modify.

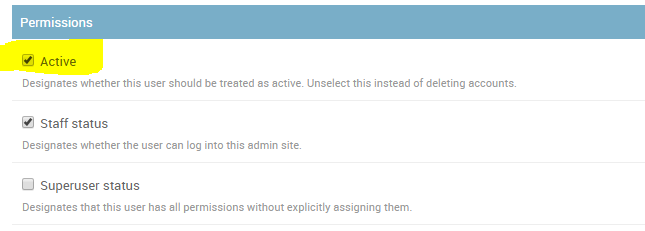
To change the password, click this link:



If you want to toggle the administrator status, check or uncheck the “Staff status” box while editing the user:

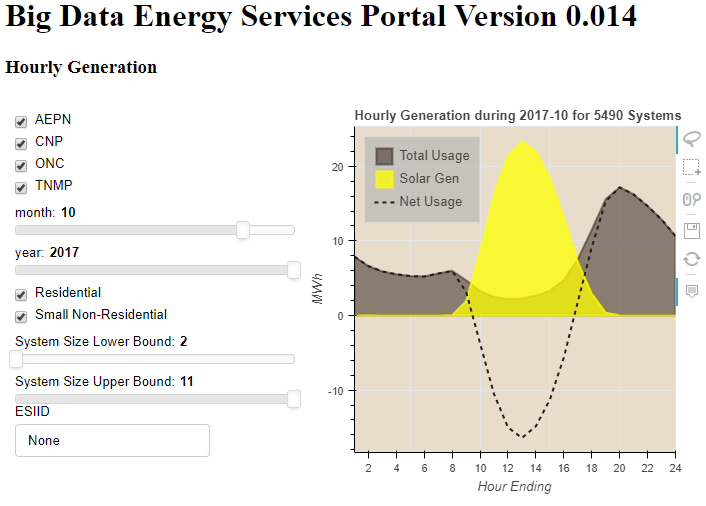


You can delete the user by clicking the “Delete” button on the bottom of the screen, but it’s advisable to uncheck the “Active” box and not actually delete the user’s records.



# Application Navigation

## Hourly Generation



This chart displays an entire month worth of solar generation and consumption along with a line showing the net consumption.

The user has the option to filter based on:

1. TDSP
2. Flow Month and Flow Year
3. Premise Type
4. System Size
5. ESIID

The MP2Energy database has a table called CustomerUsage15 that contains most of the data we need. You can see the actual query in the queries.py file.

GenerationCode = 1 is the Solar Excess. This is essentially the amount of electricity pushed back onto the grid with the solar panels are producing more than the instantaneous need of the site.

GenerationCode = 4 is the draw from the grid. This is the amount of electricity drawn from the grid when there is no solar production, or when solar production is not sufficient to supply all of the instantaneous demand.

The missing component is the actual solar production since Generation Code 1 only shows the excess. With this data source we have no visibility to the actual amount of solar produced nor the actual demand because those happen beyond the meter. The meter only records the net effect.

Definitions:

*# net = feed4 - feed1  
# solar = 1.25 \* feed1  
# total = feed4 + solar - feed1*

In order to approximate the solar generation this application makes a simple assumption that solar is produced at a rate of 1.25x of feed1 (where feed1 = Generation Code 1). Ideally we get this information from the solar inverters. As it is, this is a known faulty assumption because it hides solar production at sunrise and sunset because feed1 is almost certainly 0 during these times since consumption probably typically outweighs the modest solar production during these hours.

So, we’re making the base solar assumption in order to plot solar, net and total.

The TDSP is calculated using this formula:

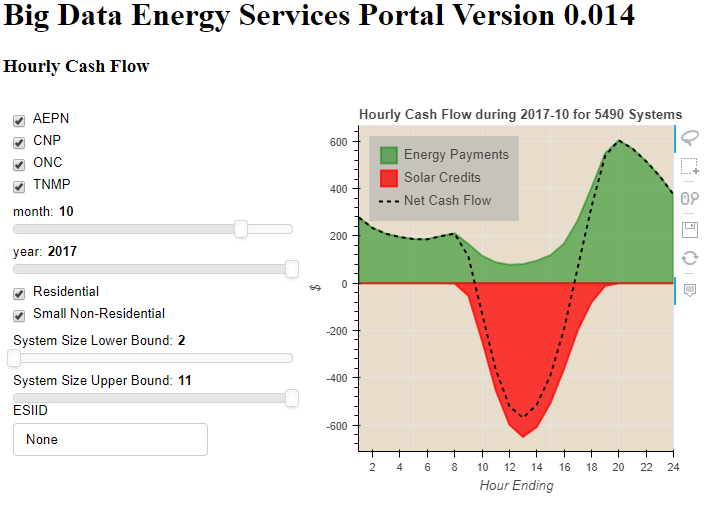
*# use lookup table for TDSP*df[**'TDSP'**] = **"unknown"**df.loc[df.Account\_No.str.startswith(**"1020404"**), **"TDSP"**] = **"AEPN"**df.loc[df.Account\_No.str.startswith(**"1003278"**), **"TDSP"**] = **"AEPC"**df.loc[df.Account\_No.str.startswith(**"1008901"**), **"TDSP"**] = **"CNP"**df.loc[df.Account\_No.str.startswith(**"1017699"**), **"TDSP"**] = **"ONCS"**df.loc[df.Account\_No.str.startswith(**"1003109"**), **"TDSP"**] = **"SHD"**df.loc[df.Account\_No.str.startswith(**"1017008"**), **"TDSP"**] = **"SHDM"**df.loc[df.Account\_No.str.startswith(**"1013830"**), **"TDSP"**] = **"NUE"**df.loc[df.Account\_No.str.startswith(**"1044372"**), **"TDSP"**] = **"ONC"**df.loc[df.Account\_No.str.startswith(**"1040051"**), **"TDSP"**] = **"TNMP"**df.loc[df.Account\_No.str.startswith(**"1040051"**), **"TDSP"**] = **"TNMP"**

System Size is randomly assigned an integer between 2 and 11 inclusive, but should come from the solar provider.

Actual consumption is the aggregate of the appropriate 15-minute fields in the table.

Premise type is a field in the table as well.

## Hourly Cash Flow



This chart displays an entire month worth of cash flows related to solar credits and normal consumption, along with a line that nets the two figures. Solar credits are monies owed to the end customer because the solar production exceeds the premise consumption. Energy payments relate to the rest of the time when the customer is pulling electricity from the grid.

The user has the option to filter based on:

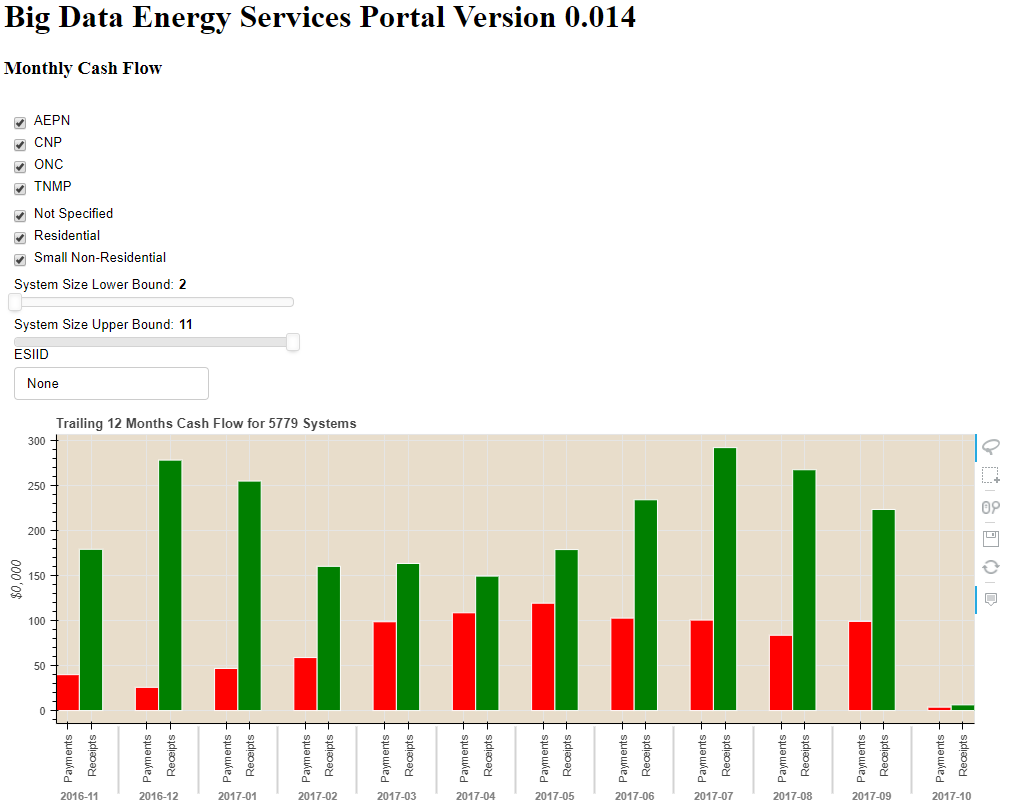
1. TDSP
2. Flow Month and Flow Year
3. Premise Type
4. System Size
5. ESIID

GenerationCode is used again for this visualization. The missing information is the contract information stating what the solar credit rate and the energy rate is. This application assigns random values for those rates according to this formula:

df.loc[df[**'GenerationCode'**] == **'4'**, **'EnergyChargeRate'**] = np.random.randint(200, 500, numnonsolarrecords) / 100  
df.loc[df[**'GenerationCode'**] == **'1'**, **'SolarCreditRate'**] = np.random.randint(200, 500, numsolarrecords) / 100

The Energy rate and Solar Credit rate are random numbers between 2 and 5 cents.

## Monthly Cash Flow

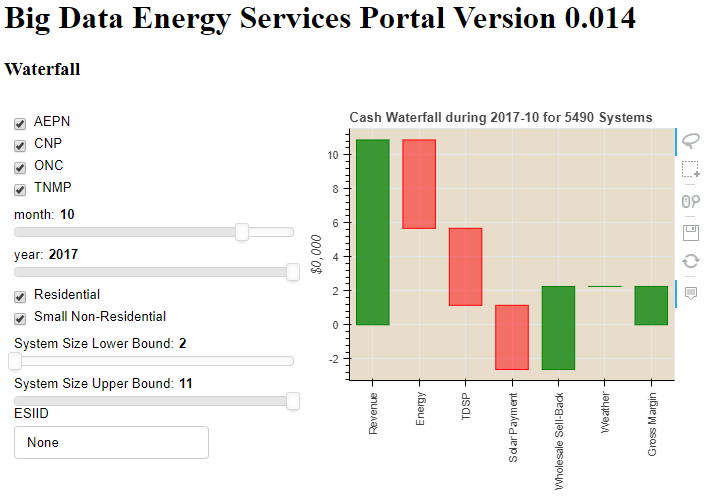


The monthly cash flow is closely related to the hourly cash flow. However, rather than displaying a single month, the Monthly view looks at the trailing twelve months and groups payments and receipts by month.

The user has the option to filter based on:

1. TDSP
2. Premise Type
3. System Size
4. ESIID

## Cash Waterfall



The waterfall view starts from revenue and goes down to gross margin for a given month.

The user has the option to filter based on:

1. TDSP
2. Flow Month and Flow Year
3. Premise Type
4. System Size
5. ESIID

In addition to the prior assumptions, this chart required assumptions on energy costs, tdsp costs, wholesale sellback rates and weather. The energy company and solar provider are ideal sources for the real information. This application makes the following assumptions:

df.loc[df[**'GenerationCode'**] == **'4'**, **'TDSPCharges'**] = df.TotalUsage \* np.random.randint(100, 400, numrecords) / 10000  
df.loc[df[**'GenerationCode'**] == **'1'**, **'ExpectedIrradiance'**] = .02 \* np.random.randn(1, numsolarrecords).flatten() + 1 *# sigma = .02, mean = 1*df.loc[df[**'GenerationCode'**] == **'1'**, **'ActualIrradiance'**] = .05 \* np.random.randn(1, numsolarrecords).flatten() + 1 *# sigma = .02, mean = 1*

energycost = -df.loc[  
 filter].TotalUsage.sum() \* 18 / 1000

wholesalesellback = df.loc[filter & (  
df.GenerationCode == 1)].TotalUsage.sum() / 1000 \* 45

TDSP charges are randomly assigned between 1 and 4 cents per kwh

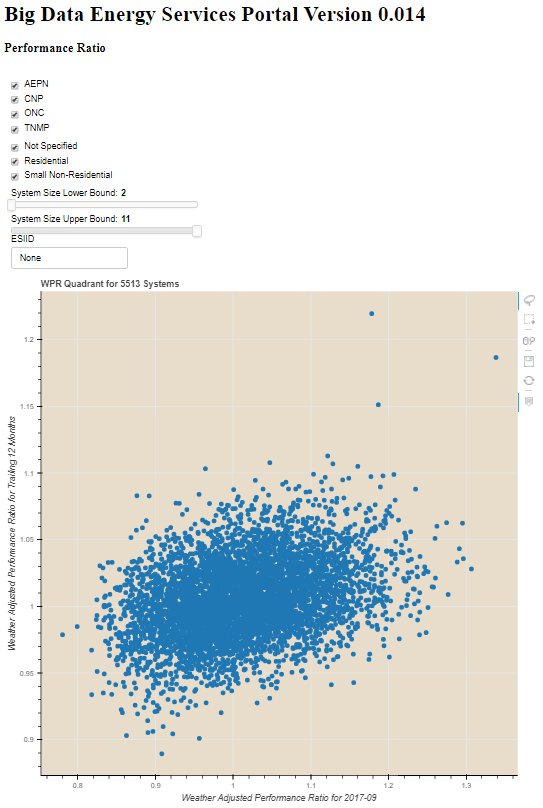
Expected Irradiance is randomly assigned on a Gaussian curve with a center of 1

Actual Irradiance is randomly assigned on a Gaussian curve with a center of 1

Energy Cost is assumed to be $18/MWh

Wholesale Sellback is assumed to be $45/MWh

## Performance Ratio



The Weatherized Performance ratios can be helpful in identifying solar system performance outliers. Normalizing the production by factoring in expected and actual irradiance places all systems on equal footing. Simultaneously looking at the prior month’s performance ratio and the trailing twelve month ratio gives insight into whether a system is behaving according to its historical performance.

The user has the option to filter based on:

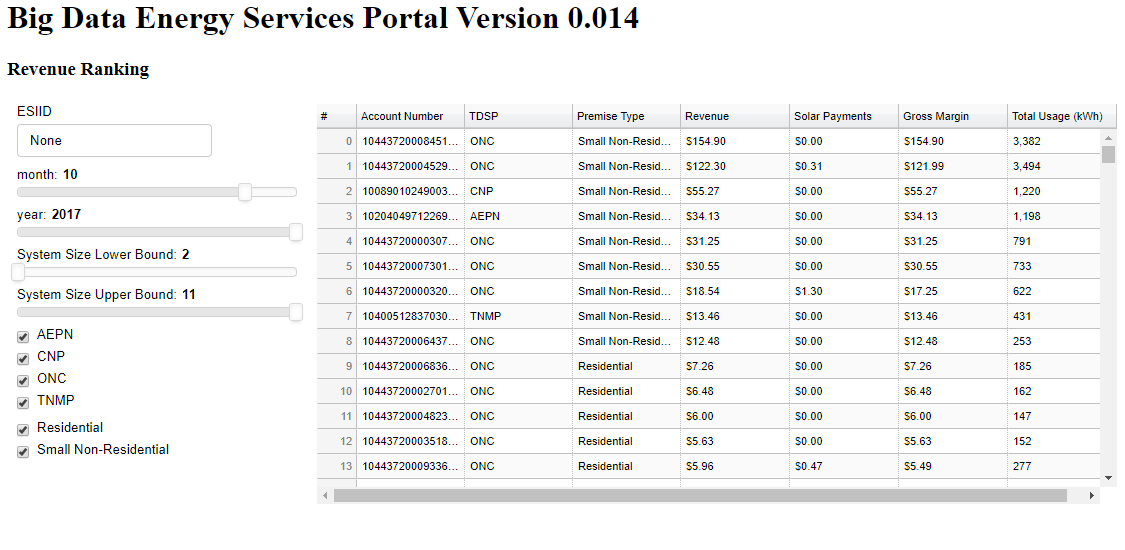
1. TDSP
2. Premise Type
3. System Size
4. ESIID

Actual Irradiance and Expected Irradiance as well as total solar production are needed for this visualization. The method to estimate total solar performance is discussed in the section on Hourly Generation. Expected Irradiance is publicly available for free but you must buy Actual Irradiance measurements. This application makes the following assumptions for irradiance:

df.loc[df[**'GenerationCode'**] == **'1'**, **'ExpectedIrradiance'**] = .02 \* np.random.randn(1, numsolarrecords).flatten() + 1 *# sigma = .02, mean = 1*df.loc[df[**'GenerationCode'**] == **'1'**, **'ActualIrradiance'**] = .05 \* np.random.randn(1, numsolarrecords).flatten() + 1 *# sigma = .05, mean = 1*

Both have a gaussian distribution centered around 1, with the standard deviation for actual being higher than that for expected.

## Revenue Ranking

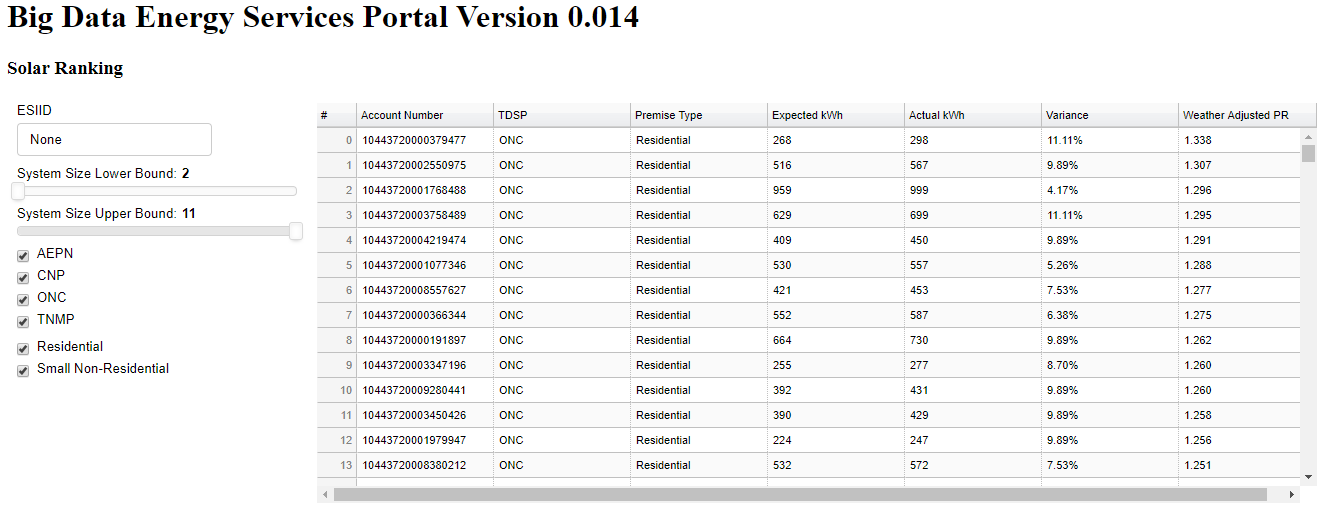


The revenue ranking chart allows users to sort all solar customers based on Gross Margin, Revenue, Solar Payments or Total Usage. This allows users to isolate high value and low value customers.

1. TDSP
2. Flow Month and Flow Year
3. Premise Type
4. System Size
5. ESIID

The data used in this table follows all assumptions listed in prior sections of this document.

## Solar Ranking



The solar ranking chart allows users to sort all solar customer based upon their solar output. The user can rank based on expected production, actual production, variance and weather-adjusted performance rations. This allows users to isolate high value and low performing systems.

1. TDSP
2. Flow Month and Flow Year
3. Premise Type
4. System Size
5. ESIID

The data used in this table follows all assumptions listed in prior sections of this document.

## Bokeh Demo Graphs

There are several examples of Bokeh demo visualizations that broadly fall into two categories: static and dynamic. The static graphs are not interactive, and thus generate each time upon execution. The dynamic graphs depend on the same bokeh server running all of the other solar graphs.

# Data Flow

Data ultimately flows from the MP2Energy database to the screen. Given that there is a large volume of underlying data, we have to be strategic about how we move and transform the data. The current implementation is as such:

1. We manually run a nightly task on the webserver to query and transform the data, and write it to two separate CSV files
   1. customerusage\_hourlybymonth.csv
   2. pr.csv
2. We manually run a nightly task to restart the bokeh server application running on the webserver to make use of the new csv files.
3. Upon loading up the graph, the webserver automatically launches code to manipulate and visualize the data dynamically

At a minimum we should automate the first two steps. We should also investigate eliminating those steps altogether, and having dynamic queries run dynamically in lieu of reading the csv files in step 3.

# Notes

All code is saved on GitHub, including setup files for nginx, bokeh server, gunicorn, SSH, and sudoers.

There is a file named Server\_Instruction.txt that shows how to instantiate this application to a new server.

User login sessions expire after five minutes of inactivity with an expiration warning happening at four minutes. These are adjustable via the settings.py file.

## To Do

## Possible Add-Ons

1. Branding / CSS Styling along with Favicon
2. Change Journey / Separate or hide Bokeh Demo Code
3. Integrate credentials into existing databases or migrate to another permanent database
4. Re-read data file during each updatedata call so we don't have to re-start bokeh server every night, and/or rethink the data flow process altogether
5. Better Hover options
6. Timestamp on data files displayed so the user knows the timestamp of the last available data
7. Allow users to update their own passwords
8. Change to a reserved domain and add SSL Authentication
9. Keep track of user statistics -- logins, session times, tracking pages