Section 2:

# Data/operation abstraction design

The primary data used for this Shiny application was is publicly available data provided by the US Energy Information Administration (EIA) in the form of year EIA-923 reports, or equivalent (see Reference XYZ). Before 2006 the relevant data was reported in reports EIA-906/920 or EIA-906, which are consistent with EIA-923. Report EIA-923 reports provide detailed electric power data on electricity generation, fuel consumption, fossil fuel stocks, and receipts at the power plant and prime mover level. This app was only concerned with electricity generation on a yearly basis and therefore was restricted to the Schedule 1 survey data, or equivalent. The reports are available in spreadsheet form from 2001 through 2018.

Additionally, the most current issue of report EIA-860M was also used (December 2018). This report provided generator-specific information for all generators and associated environmental equipment at electric power plants with 1 megawatt or greater of combined nameplate capacity. This report provided context for the EIA-923 information, including energy type, location, and plant name, and other essential information. Location data is comprised of longitude and latitude coordinates, as well as city and state information.

The raw data from reports EIA-923, or equivalent, and EIA-860 were compiled, merged, and joined initially through SAS Enterprise Guide process. The data was then processed in R, until all of the necessary information was contained within a single, tidy dataframe. Each observation within the dataframe was an energy type (see descriptions below) at an operating plant for a given year, with all relevant yearly data such as net electricity generation or net greenhouse gas emissions. If multiple energy types were present at the same location then that location would have as many operations as energy types, with each energy type reporting output data. For example, Plant.ID\_Energy (see Section 2.2) values of 10\_COL and 10\_NG indicate coal generation and natural gas generation at Plant ID 10. Both of these observations have the same location data, but different fuel type, generation, and emission data.

Further processing of the dataframe was then performed within the Shiny app file, although outside the Ui and Server, to create several dataframes specific to Shiny application components. Creating these specific dataframes was done to limit unnecessary processing with the server during operation.

See Appendix, Section XYZ, for the specific data processing steps.

Section 2.2

The complete dataset, before further app specific processing, contained the following twelve variables:

|  |  |
| --- | --- |
| Plant.ID\_Energy | Unique identifier created by combining the Plant ID provided by EIA-923 and the project\_energy\_code |
| Plant.Name | Provided by EIA-860M |
| project\_energy\_code | Two or three letter code representing the project\_energy\_type |
| project\_energy\_type | Energy source as defined by this project. EIA-923 uses 40 different reported fuel type codes. These codes are consolidated into 9 energy types: Biomass, Coal, Hydroelectric, Natural Gas, Nuclear, Oil, Other, Solar, and Wind. A conversion table listing the EIA-923 reported fuel type code and the project defined energy code and energy type, also with a renewable status, is found in Appendix, Section 2. |
| renewable | Binary condition. An observation is renewable if the fuel type is Solar, Wind, or Biomass. |
| net\_gen\_mw | Net generated electricity, in megawatts (MW) provided by EIA-923 |
| emission\_output | Greenhouse emission output for each observation, in gCO2eq. Emissions were calculated by applying a emission per MW factor against the yearly net generation. Emission factors were supplied by IPCCC AR5 (Reference 7.3) and WNA Report (Reference 7.4) |
| Effective\_Rate | The observed average generation rate for the year, MW. See in Appendix, Section 2. |
| State\_SF | State power plant is located in. Due to missing values in EIA-923 and EIA-860M, this variable was calculated via lookup using the latitude and longitude data. See in Appendix, Section 2 |
| Latitude | Plant latitude coordinate. Provided by EIA-923 |
| Longitude | Plant longitude coordinate. Provided by EIA-923 |
| Year | Year corresponding to net electricity generation. Provided by EIA-923. |

# References

EIA-923, 2001 - 2017

EIA-860. December 2018

International Panel of Climate Change Fifth Assessment Report (AR5), Annex III, 2014

Comparison of Lifecycle Greenhouse Gas Emissions of Various Electricity Generation Sources, World Nuclear Association, July 2011.

Section 4

Algorithmic Design

There are two major components to this app, the map visualization utilizing Leaflet, and five charts spread over four separate tab panels. Each component required special consideration for algorithmic design.

1. Leaflet Map

The leaflet map, visualizing all of the power generating location in the Southeast United States, is the only component that is permanently on the screen while the app is in operation. Additionally, this component is processing the largest amount of data in comparison to the charts. The dataframe used to generate the map requires every power generating location for every year as an input. The dataframes used as input for the charts are generally highly summarized and reduced in size.

Therefore, proper design is essential for the map to run smoothly. The primary method for this is to break up the leaflet visual is to several different components. The first component is called outside of the server and the ui. This component performs the initial render the leaflet map. It defines the space, but it does not call the points to represent each power plant.

###Insert Code###

The points are called by a separate component, inside the server. Instead of using the function leaflet() to call the map, the function leafletproxy() is used, nested in an observe() call. By adding the points in the leafletproxy() call, full leaflet map from is not rendered after every change. Only the dynamic aspects of the map are affected, and only when an action is observed.

##Insert Code###

Additionally, options were set in order to limit how often and when the leaflet map would refresh. These options further reduced the computational load of the map, without noticeably degrading the operation of the map for the user.

1. Charts

The primary method for maximizing algorithmic capability for the charts is to perform as much data processing as possible outside of the server and ui before passing the required dataframe into the server. As discussed in Section 2, the raw data was processed into one tidy dataset before import into the shiny app. From this master dataset, eight unique datasets were developed. Each of these datasets was specifically processed for use one or two charts. These datasets were then imported into the server as reactive values, and then further used in the designate chart with the only processing occurring in the server being filtering via year and/or state.

A benefit of having the charts broken out onto separate tabs is that only the chart visible to the user active. See Appendix, Section 3, for a screenshot of the reactlog record of the operation of the app, with Total tab selected. There are two reactive expression expressions active, one grabbing the dataframe to input into the map (observe({ EIA\_923\_SE…}), and the other grabbing the dataframe input into the visible chart on the Total tab (output$net\_gen\_stacked\_bar). The reactive expressions to grab input into the other charts are not active.

Additionally, the stacked bar chart was the most computationally intensive output in the app. A screenshot of the profvis test is also found in Appendix, Section 3.

Alorithmic Optimization / Scalability

Nest Model: Abstract Threats 3.6

Reactivity Log

1. Import master tidy dataset
2. Process master dataset
   1. Process into app component specific dataframes
   2. Minimizing computation within the server. Limited to filtering by state/year
3. Leaflet
   1. UpdatewhenZooming = FALSE
   2. updateWhenIdle = TRUE
   3. Leaflet Proxy
      1. Observe / ObserveEvent

Leaflet

* UpdatewhenZooming = FALSE
* updateWhenIdle = TRUE
* Leaflet Proxy
* ObserveEvent

1. User Eval
   1. Usability: Capable, not optimal
   2. Efficiency: