EE/CprE/SE 492 FINAL REPORT

Group number: 15
Project title: Emissions Tracking Dashboard
Client: MISO (Midcontinent Independent System Operator)
Advisor: Dr. James McCalley
Team Members: Manbir Guron, Jack Riley, Daman Riat, Sean Fleming, Dylan Christensen, Tyler Maglaya, John DiBasilio

Executive Summary

Development Standards & Practices Used

- IEEE Std 1063[1]. Standard for Software User Documentation. Minimum requirements for the structure, information content, and format of user documentation.
- IEEE Std 1220:2005[2]. Standard for Application and Management of the Systems Engineering Process. Describes the systems engineering activities and processes required throughout a system's life cycle to develop systems meeting customer needs, requirements, and constraints.
- IEEE 16326:2019[3]. This standard gives content specifications for project management plans that cover software projects and describes applying sets of common project processes to software and system life cycle.
- IEEE 23026-2015[4]. This standard describes the system engineering and management requirements for the life cycle of websites, including the strategy, design, engineering, testing and validation, and management for the Intranet and Extranet environments.

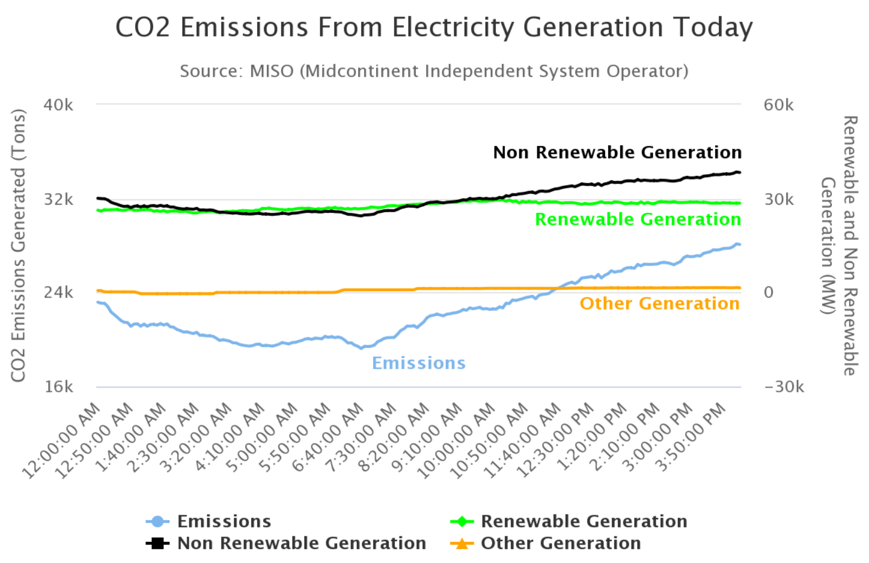
Requirement Summary

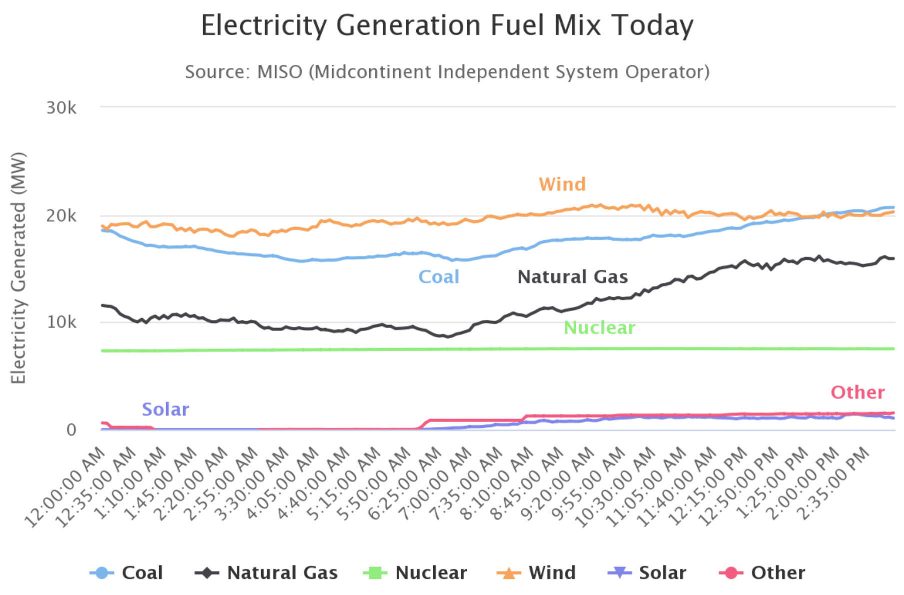
- Graphical User Interface
- Software program design
- Assumption development
- Information access and availability

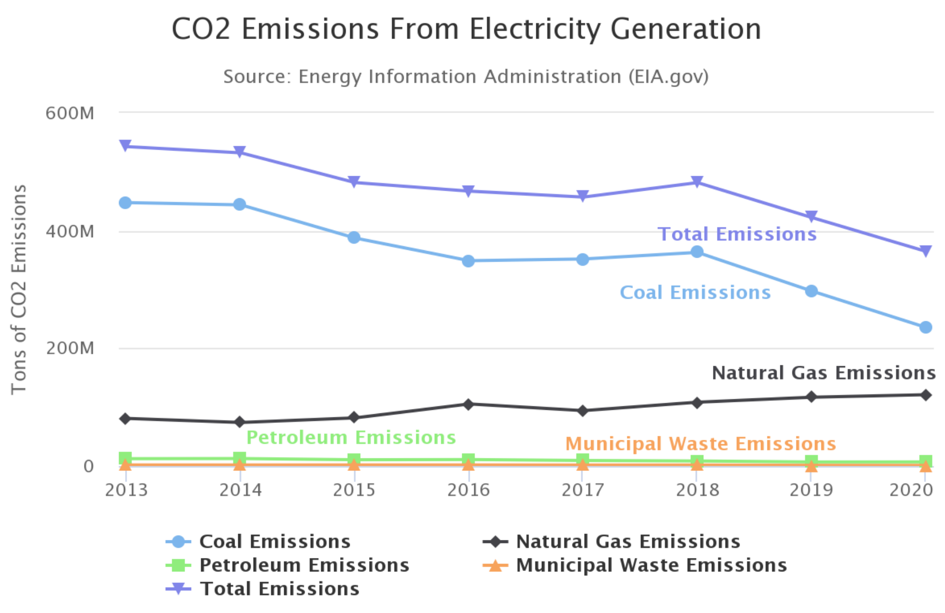
- Program documentation

Final Dashboard

Within this project, our team worked to develop an emissions tracking dashboard for Midcontinent Independent Systems Operator (MISO). The main deliverables for the project were to develop a web page that hosted several real time and historical graphs which highlighted emissions data for MISO’s stakeholders. To gather the data for the graphs, the team developed web scraping tools in order to supply both historical and real time data. All data used in the project was publicly available information. Some of the graphs may be seen in the figures below.


CO2 Emissions From Electricity Generation Today
Source: MISO (Midcontinent Independent System Operator)
CO2 Emissions Generated (Tons)
40k
32k
24k
16k
60k
30k
0
−30k
Renewable and Non Renewable Generation (MW)
Non Renewable Generation
Renewable Generation
Other Generation
Emissions
12:00:00 AM
12:50:00 AM
1:40:00 AM
2:30:00 AM
3:20:00 AM
4:10:00 AM
5:00:00 AM
5:50:00 AM
6:40:00 AM
7:30:00 AM
8:20:00 AM
9:10:00 AM
10:00:00 AM
10:50:00 AM
11:40:00 AM
12:30:00 PM
1:20:00 PM
2:10:00 PM
3:00:00 PM
3:50:00 PM
Emissions
Renewable Generation
Non Renewable Generation
Other Generation


Electricity Generation Fuel Mix Today
Source: MISO (Midcontinent Independent System Operator)
Electricity Generated (MW)
30k
20k
10k
0
Wind
Coal
Natural Gas
Nuclear
Other
Solar
12:00:00 AM
12:35:00 AM
1:10:00 AM
1:45:00 AM
2:20:00 AM
2:55:00 AM
3:30:00 AM
4:05:00 AM
4:40:00 AM
5:15:00 AM
5:50:00 AM
6:25:00 AM
7:00:00 AM
7:35:00 AM
8:10:00 AM
8:45:00 AM
9:20:00 AM
9:55:00 AM
10:30:00 AM
11:05:00 AM
11:40:00 AM
12:15:00 PM
12:50:00 PM
1:25:00 PM
2:00:00 PM
2:35:00 PM
Coal
Natural Gas
Nuclear
Wind
Solar
Other



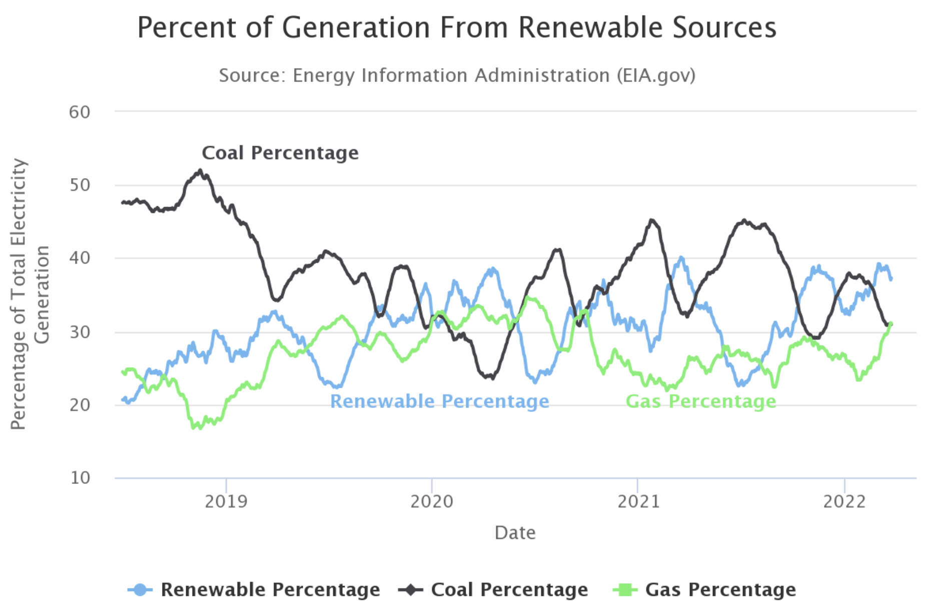


Figure 1. Dashboard Excerpts

Applicable Courses From Iowa State University Curriculum

SE 319: Construction of user interfaces, SE 329: Software project management, SE 309: Software development practices, EE 303: Power systems, CprE 339: Software architecture, CprE 416: Software evolution

Key Skill Development Areas

Javascript and Python programming, Highcharts design, data analysis, user experience analysis, and development.

Table of Contents

Problem Statement

Team 15 is tasked with developing an emissions dashboard for MISO (Midcontinent Independent System Operator). MISO stakeholders lack a fast and efficient way to analyze public emissions data generated by MISO. Currently, the data can be found in numerous different places, which can be cryptic to many consumers. The emissions dashboard is intended to solve this problem and provide stakeholders and MISO an opportunity to interact with public emissions data through a series of user-friendly visualizations. This dashboard should allow users to understand the emissions impact of their energy generation or consumption and create a rich context behind this data to make complex trends in electricity generation emissions simple to view and understand.

Requirements and Constraints

The main goal of this project is to produce a dashboard where users can view a variety of emissions data for either the entire MISO territory or individual MISO regions. Below are the detailed requirements and constraints.

1) Graphic interface

- a) Daily Emission Chart
  - i) Line Chart showing Emissions of MISO / MISO regions up to current hour starting from 00:00
- b) Interactable by public website user
  - i) Filters by region on emission output

ii) Allow user to download recent data (Data up to that hour of the day starting from 00:00)

iii) Allow graph to be “popped out” to a larger standalone display

iv) Include description of chart and data being used to make it

(1) Example: current MISO real time charts have a (i) which when clicked on, give info about the chart

v) Includes area, generation type, possibly fuel type filters

c) Similarity to Current MISO website graphs

i) Hours X axis, Emission Y axis

ii) Use MISO colors

2) Software program design

a) Python is widely used in MISO for automation and data analysis.

b) Highcharts is the graphing software that will be used for the project, as MISO is switching to their platform for future graphics

3) Assumption development

a) Maintain documentation of assumptions made and any sources/discussion notes leading to those assumptions

i) Any formulas or derivations made to reach emissions calculations need to be fully described and maintained in a list of formulas used

ii) Maintain cited source list of all sources of information for any calculations which assist in the emission derivation effort

4) Information access and availability

    a) All information used must be publicly accessible, which could create a degree of approximation in deriving the exact real time emissions data from any source (Constraint).

5) Program documentation

    a) At the conclusion of the project, a write up of the overall project will be created with the intent to inform others of the work done and share knowledge to allow expansion upon the foundation created by the project

Intended Users and Uses

The intended consumers of the emissions dashboard will be MISO, its stakeholders, and the public at large. All parties benefit from the creation of the emission dashboard, whether they are generation facilities, utility companies, or electrical consumers. Users can easily access emissions data that is historical and in real time, that can be used for projects and other business decisions for electric utilities and their customers. They will be able to take the data provided in the dashboard and use it however it applies to their situation.

Researchers employed by MISO area utilities and outside institutions can see how various trends impact emissions of the MISO region, allowing them to study new generation projects and technologies to effectively meet emissions targets set by MISO customers and all levels of government.

Stakeholders of MISO utilities, including utility companies and individual end users, can view this dashboard to see the emissions impacts of their regular activities and look to see how they can decrease the impact of their energy usage by shifting the time of day, season, or physical location of their energy usage.

With the implementation of interactable charts, maps, and graphs that represent the emissions data from the various MISO regions, large amounts of data can be expressed in a user-friendly way so that the parties above can easily access it in a quick and efficient manner.

Project Plan

Project Management

We used an agile project management style, because our sponsor wanted a dashboard that included a variety of data sources, views, and elements of user experience. This required an interactive design process with small tasks designed to build layers of functionality on top of each other.

Our team used Git to manage any code written for our dashboard, to track issues, and major changes. We used our Microsoft Teams chat and associated integrations to manage smaller discussions with less time sensitivity.

Task Decomposition and Key Metrics

Construct a model to find all plants in the MISO system, categorize them into regions to create the *Regional Generation Source Data (RGSD)*

A. Determine required input sources to achieve desired model output.

   a. Determine and select available data sources relevant to plant location, state, and MISO region. *(Metric:MISO/team discretion)*

   b. Determine the methods to extract and store data from relevant sources into the project's dataset. *(Metric: Data can be extracted and usable during subsequent steps)*

   c. Write code to extract and organize relevant data from selected sources into the project dataset. *(Metric(s): Extracted data matches source data, code is well organized, code runs in an efficient manner (efficiency metric TBD))*

2. Construct a module to import historical emissions data from all plants/ MISO region and states Plant Historical Emissions Data (PHED)

A. Determine relevant data sources and what calculations or estimations need to be performed. *(Metric: MISO acceptance)*
B. Write code to extract source data and link it to RGSD from (1) *(Metric: Verify data by "fact checking" samples with an external, existing data source.)*

C. Write code to perform required calculations or estimations on source data and preserve source data for potential user output. *(Metric: Code output matches expected calculations)*

D. Write code that prepares data for output to dashboard: Perform any variable type conversions, organizational changes, etc., to make the dataset easily incorporated by dashboard code. *(Metric: Data is organized such that it is easy to incorporate and manipulate using the dashboard (team/subteam discretion)*

3. Construct a module to import real time data from all plants: Plant Instantaneous Emission System (PIES)

The task/subtask goals for (3) are the same as for (2)

4. Construct the dashboard to visualize regional emissions data.

A. Develop and code the initial dashboard visualization template. *(Metric: MISO acceptance)*

   a. (Possible goal): Create a static, team-defined dataset (and test server?) for testing purposes which will give a known output.

   b. Support functionality for various filters and user interaction tools. *(Metric(s): User can view data as specified by MISO proposal, filters function properly without error, output is as expected)*

   c. Support functionality for user data download. *(Metric: Data can be downloaded in a format similar to other MISO dashboards (CSV or other formats as required)*

B. Write code to incorporate RGSD (1) and PHED (2) data into the dashboard. *(Metric: Dashboard shows accurate visualizations from source data)*

C. Test dashboard interaction. *(Metric: Dashboard operates (proper filtering, updating, allows repeated filters)*

   a. (Possible goal): Test dashboard on the "test dataset" to verify the information is displayed properly for all user filters and outputs correctly.

   b. Test dashboard using the real dataset.

D. Optimize user experience *(Metric: Dashboard response is up to par with MISO standards)*

   a. Create a smooth user experience that can be replicated on a wide variety of devices and viewing environments

E. Submit dashboard model to MISO for approval; revise dashboard as requested. *(Metric: MISO approval)*

5. Construct the dashboard view to visualize MISO total emissions graphs

The task/subtask goals for (5) are essentially the same as for (4).

7. Analyze emissions trends

A. Rough outline of overall emissions trends. (Metric: Meets general report standards)

B. Incorporation of dashboard figures, tables, and other useful information into the report relating to textual references.

Risks

1. **Construct a model to find all plants in the MISO system, categorize them into *Regional Generation Source Data (RGSD)***

a. Select data sources from MISO and EIA databases
    i. *These data sources may not include all the information we need.*
    Mitigation: We have to estimate our data based on the information we are allowed to use. (HIGH RISK)
    ii. *The data source location could change and become unavailable to our project.*
    Mitigation: We will need to (i) find other sources for the data or (ii) handle these errors appropriately by making the data sources used in our code easy to understand and update as required. (HIGH RISK)

b. Determine the methods to use to extract and store data
    i. The methods we use to extract and store data could become deprecated or removed from public use. (LOW RISK)
    Mitigation: Use established, open source applications throughout the project.

c. Extract and store relevant plant metadata
    i. Relevant data is provided by internal and external sources with respect to MISO.
        1. Internal source problems. (LOW RISK)
        Mitigation: Communicate concerns with MISO to build redundancy.
        2. External source problem. (MEDIUM RISK)
        Mitigation: Incorporate multiple paths to source data?

2. **Construct module to import historical emissions data from all plants, Plant Historical Emissions Data (PHED)**

   Low risk - If data is properly sourced, the data collection should be straightforward.

   Mitigation: Build multiple paths to access data/easily configure code for future changes to source data.

3. **Construct module to import real time data from all plants, Plant Instantaneous Emission System (PIES)**

   Same as task #2

4. **Construct the dashboard to visualize regional emissions data**
   a. Determine initial user requirements for dashboard views, interactivity, UX/UI
      Mitigation: Communicate with MISO to ensure user requirements and update as required.
   b. Incorporate RGSD, PHED, and PIES datasets into base dashboard
      i. (Medium Risk): different data formats and other incompatibilities may create difficulty here
      ii. **Mitigation:** Care will need to be taken to make sure data formats are synchronized during tasks 2 and 3
   c. Optimize dashboard interaction and performance

        i. Data analysis required excess computation time. (MEDIUM RISK)

           Mitigation: Optimize code as much as possible to remove latency. Verify load time is an acceptable range for MISO.

    d. Test dashboard interaction

        i. Dashboard does not fall in line with the current MISO dashboard latency.

           Mitigation: Verify MISO requirements on latency and attempt to optimize code for speed.

5. **Construct dashboard view to visualize MISO total emissions data**
    a. Add together the various data sources and graphs into a single dashboard

        i. Dashboard information does not correlate with the combined dashboard information.

           Mitigation: Review and refine data fed to the combined dashboard.

6. **Analyze emissions trends**

        i. (Low Risk): If the dashboard works and data is robust, constructing an analysis and report should be straightforward

        ii. **Mitigation**: If the dashboard is not operable, the team may need to manually sort through data to construct our analysis

Project Design

Contextual Considerations

Relevant considerations related to the project in each of the following areas:

Area	Description	Examples
Public health, safety, and welfare	How does your project affect the general well-being of various stakeholder groups? These groups may be direct users or indirectly affected (e.g., a solution is implemented in their communities)	This project will give stakeholders an understanding of the existing and historical health effects of electricity generation. This will allow for more educated decision-making regarding how to improve the electricity grid to reduce the harmful effects of pollution.
Global, cultural, and social	How well does your project reflect the values, practices, and aims of the cultural groups it affects? Groups may include but are not limited to specific communities, nations, professions, workplaces, and ethnic cultures.	This project advances the social responsibility and transparency of MISO and its planning engineers. By using open data and displaying it in an accessible format for the general public, electricity consumers are better educated on the effects of their behavior. This project promotes the values of transparency and environmental consciousness.

Environmental	What environmental impact might your project have? This can include indirect effects, such as deforestation or unsustainable practices, related to materials manufacture or procurement.	Our project will allow MISO customers to better understand the environmental impacts of their electricity usage. This allows for more educated decision making on environmental grounds.
Economic	What economic impact might your project have? This can include the financial viability of your product within your team or company, cost to consumers, or broader economic effects on communities, markets, nations, and other groups.	Product has minimal costs in implementation, but awareness of the data presented could distort electricity demand depending on the time of day, etc. This potential distortion of the market while seeking lower emissions could adversely impact consumers in states with more liberalized electricity pricing models via demand fluctuations. Additionally, the clear data presented in this project could drive decision-making in future network planning which could

		financially benefit the producers of clean energy.

Table 1. Design Context

User Needs

MISO customers (Utilities) need a way to see the emissions profile of the entire MISO network to better understand where and how to introduce more clean energy into their networks.

Electricity ratepayers and concerned citizens need to understand the emissions profile of their electricity usage to find ways to alter their behavior to reduce their environmental impacts.

MISO wants to collate a large amount of data about generation emissions as customers, regulators, and similar groups will demand more detailed information about generation emissions in the future.

Prior Solutions

California ISO has produced a dashboard[5] showing the historical, real time, and source of CO2 emissions in the electricity generation occurring in their footprint. This dashboard contains a variety of time scales, sources, and contexts that help explain the trends in CO2 emissions clearly. Another advantage of the dashboard is the hyperlinking of official government sources that give more context to the data and the statutory regime that necessitates and produces the data. Additionally, the dashboard is consistent with the rest of the ISO's website regarding visual standards and visual appeal. One shortcoming of the database is the lack of information on the exact sources of the CO2 emissions beyond the fuel used in generation. Ideally, a customer would have some information about the region where the electricity is being produced and some more insight into pollutants besides CO2.

Other ISO's such as New York ISO[6], ISO-New England[7], and Southwest Power Pool[8] have produced dashboards that visualize the mix of fuels used in generation of electricity in their regions. These dashboards succeed in communicating the fuel mix but lack detailed context on the emissions impacts of this mix. The dashboards are all clear and simple to understand, and many contain the option to interact with the data or download the source data. This interactivity is very important and allows stakeholders to engage with our project immediately and should be integrated as a feature in our design.

Proposed Design

The proposed design consists of an emissions dashboard similar to what California ISO has on their website, with how they have their CO2 resources charted in trends and charts, with included historical data. Research is still being conducted on implementing this data into a website interface user using Highcharts, a JavaScript library. Data will be polled from external sources such as Energy Information Administration and MISO's websites.

The design will consist of three main technical components consisting of a Python based historical data scraper, a Javascript based real time data scraper, and a Javascript based graphing engine implementing the Highcharts library. The Component level design diagram is below. Note that this diagram does not include EIA API data, which would be represented as a fourth independent column in this figure.

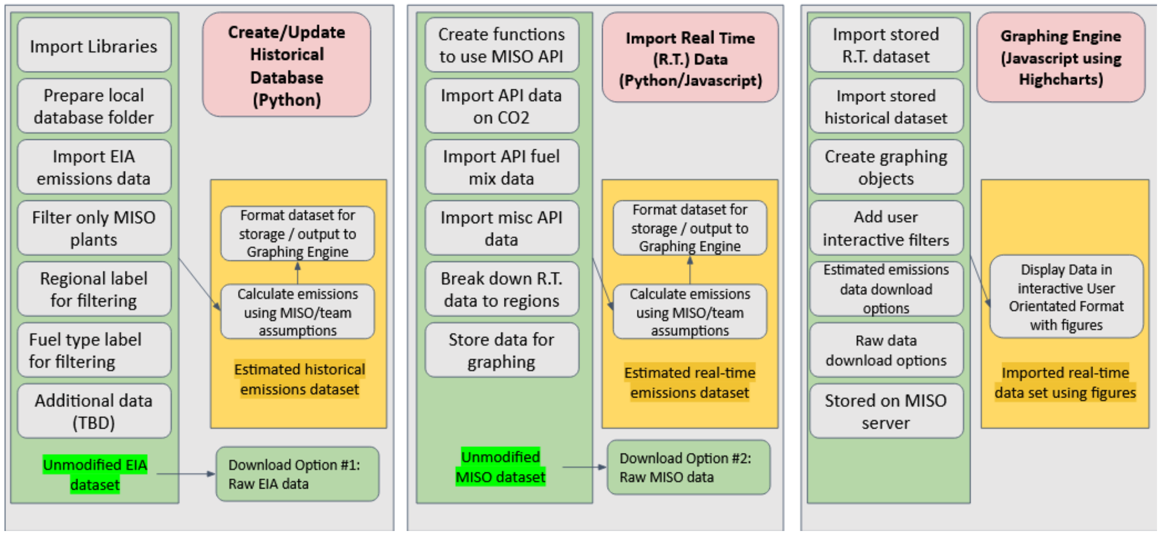

Import Libraries
Prepare local database folder
Import EIA emissions data
Filter only MISO plants
Regional label for filtering
Fuel type label for filtering
Additional data (TBD)
Unmodified EIA dataset
Create/Update Historical Database (Python)
Format dataset for storage / output to Graphing Engine
Calculate emissions using MISO/team assumptions
Estimated historical emissions dataset
Download Option #1: Raw EIA data
Create functions to use MISO API
Import API data on CO2
Import API fuel mix data
Import misc API data
Break down R.T. data to regions
Store data for graphing
Unmodified MISO dataset
Import Real Time (R.T.) Data (Python/Javascript)
Format dataset for storage / output to Graphing Engine
Calculate emissions using MISO/team assumptions
Estimated real-time emissions dataset
Download Option #2: Raw MISO data
Import stored R.T. dataset
Import stored historical dataset
Create graphing objects
Add user interactive filters
Estimated emissions data download options
Raw data download options
Stored on MISO server
Graphing Engine (Javascript using Highcharts)
Display Data in interactive User Orientated Format with figures
Imported real-time data set using figures

Figure 2. Component Design Diagram

The historical data importing module will interact with EIA data by importing data from a given URL, filtering the resulting API output into plants within the MISO region and MISO subregions, outputting a final set of CSV files to be used in the graphing engine. This module will only run when new EIA data is released in the autumn of each year. The long term location of this module is not determined, as this script runs on an infrequent schedule.

The real time data importing module will interact with MISO API data by calling this API when new data is released (every 5-15 minutes) and store this data into .json files for storage for the concurrent day. These .json files will be stored for a user activated data download option and used in the graphing engine. This module will run on MISO servers and be maintained by MISO engineers on a long term basis.

The graphing engine will import the CSV files from the historical data scraper and .json files from the real time data scraper and manipulate these files to populate the given graphing objects. This graphing engine will be stored on MISO servers and conform to MISO UX/UI standards to create a seamless and professional user experience reflecting MISO’s brand values and stakeholder commitments.

Design Visualizations

The design on the dashboard will cover the historical and partial real time emissions data from the North, Central, and South regions of MISO’s area of operations. The user will be able to access data from each region or the MISO network, meaning historical emissions data from other MISO regions will be filtered out. This allows the information that the user was seeking to be easily visible. There will be various charts and graphs that will display which regions are hitting their emissions targets, trends of emission data, and show a historical outlook of how a region's emissions are affected by different variables. The visual of the screen will be shown below:

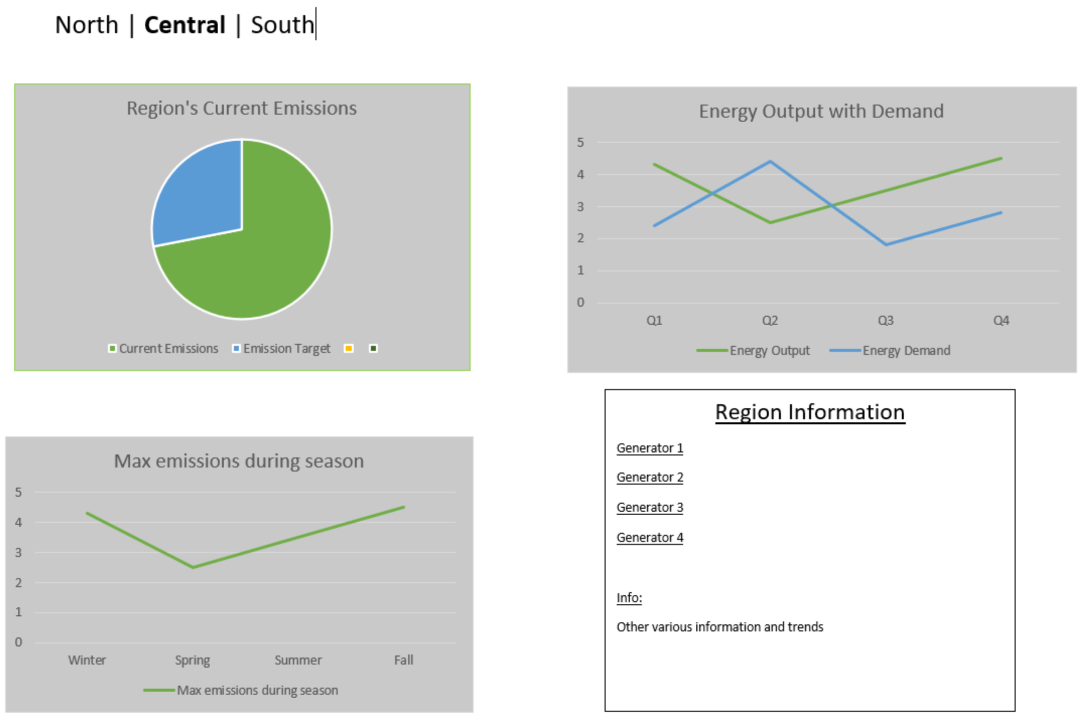


Figure 2. Sample Dashboard layouts

Functionality

Users will be able to publicly access this dashboard on MISO's website. We will implement interactable maps and data for easy UI traversal for the user and quick data gathering. The dashboard aims to allow MISO to see how various trends impact emissions of each MISO region, and customers of MISO can see how their activities impact their emissions data so they can more easily regulate their activities to hit set emission targets.

Currently, all functional and non-functional requirements are satisfied due to the many conversations about the design and implementation of features with MISO. Javascript, along with its Highchart library, will be used to cover these requirements.

Areas of Concern and Development

Primary concerns:

1. Data sources moving or links breaking.
2. Dashboard layout or functionality do not meet customer's (MISO) vision of the product.
3. Estimated emission information is not accurate or detailed enough for customer requirements.
4. User experience interacting with the dashboard is less than desired.
5. Project is successful but cannot be maintained or updated by the customer.
6. Data sources moving or links breaking.
    a. Properly document code and design for the possibility of data sources moving. For example, create a SOP for updating any broken data sources, have the code look and notify that "DATASOURCE_X" is missing, etc.
7. Dashboard layout and/or functionality does not meet customer's (MISO) vision of product.
    a. Ensure that the customer specifications (listed in MISO provided document "Project Requirements" (PR) Section 2) are implemented early in the development.
    b. Early prototyping of the dashboard to share with the customer.
    c. Maintain open communication with the customer throughout the development process.

8. Estimated emission information is not accurate or detailed enough for customer requirements.
    a. Develop a clear plan for data collection, manipulation, and calculations required, and share this plan with the faculty adviser and MISO before major programming tasks begin.
    b. Record clear documentation of all data sources accessed, assumptions made, data calculations or manipulations performed, etc., to ease any modifications requested by the customer during the development process.
9. User experience interacting with the dashboard is less than desired.
    a. Create a test server to host the dashboard during development.
    b. Perform various tests and performance evaluations as the dashboard evolves.
    c. With each successive iteration of the dashboard, obtain input from the customer or ask fellow students to test out UI to locate potential problems or improvements.
10. Project is successful but is not able to be maintained or updated by the customer.
    a. Create proper documentation on how to use the dashboard and perform critical maintenance such as changing data sources, adding data views, etc.
    b. Properly format and document all code so that it is easy to understand and edit.

Technology Considerations

Our design produces an expandable dashboard that allows for a great degree of flexibility in implementation, but this flexibility in the graphing framework can create a large degree of uncertainty in implementation. If our group used a higher level graphing framework like Tableau, we might be able to produce a dashboard in less time with a higher degree of user experience sophistication. The main caveat to using a tool other than Highcharts is that further expansions of our dashboard will be difficult as MISO is looking to exclusively use Highcharts for similar activities.

Additionally, the decision to use both Python and Javascript for different portions of the project creates a situation where packaging the project for further storage may require a bespoke or inelegant solution. If the group used one programming language natively packaged into one application, overall interaction with the whole program would be easier, but the implementation of basic functionality could be more difficult than the baseline solution.

Design Analysis

The design plan was effective as it effectively isolated major functional delivery requirements and created discrete functional components for project implementation. After further work on the project and the initial creation of the design plan, greater emphasis on systems integration and software reusability was critical in creating a more robust final project. The design plan is largely based on initial functional requirements and pays less attention to the final user experience requirements. By adding more focus on the target product experience, the design was more cohesive and professional. Overall, the design plan iterated in section 3.3 of this document was effectively implemented in a multifaceted approach towards a complex project.

Design Plan

Describe a design plan concerning use-cases within the context of requirements, modules in your design (dependency/concurrency of modules through a module diagram, interfaces, architectural overview), and module constraints tied to requirements.

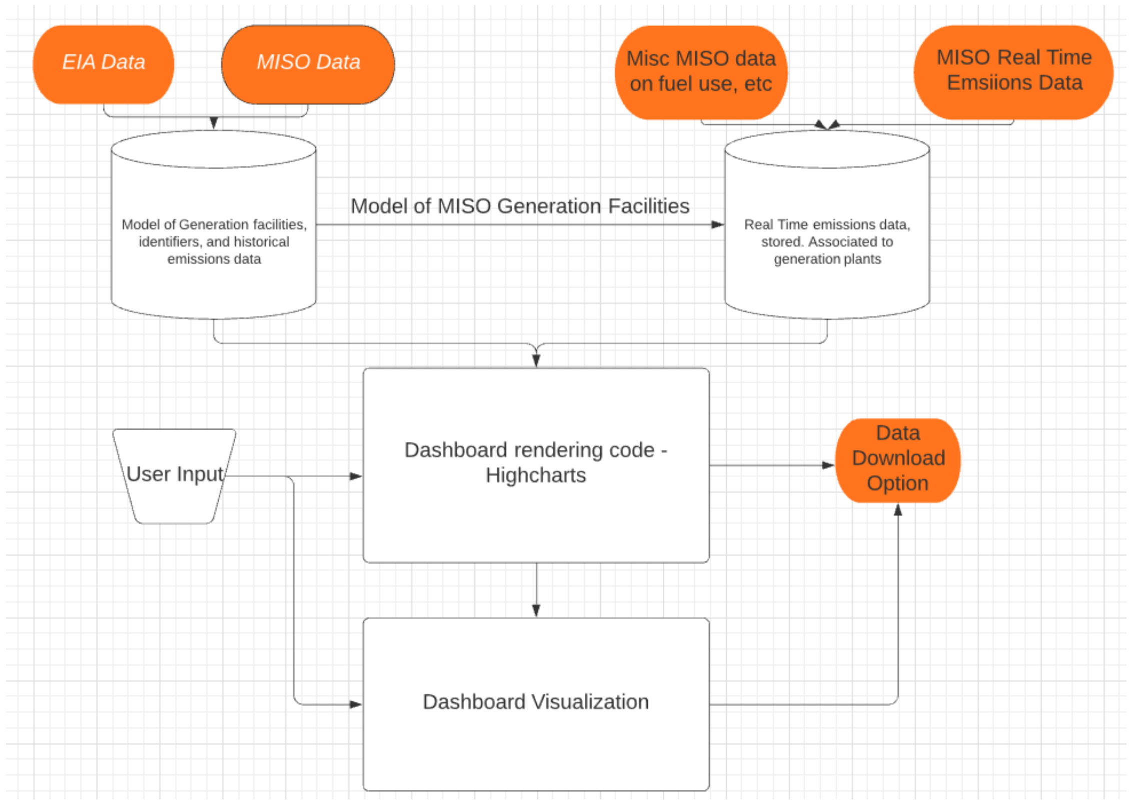

EIA Data
MISO Data
Misc MISO data on fuel use, etc
MISO Real Time Emsiions Data
Model of Generation facilities, identifiers, and historical emissions data
Model of MISO Generation Facilities
Real Time emissions data, stored. Associated to generation plants
User Input
Dashboard rendering code - Highcharts
Data Download Option
Dashboard Visualization

Figure 3. Mid-High Level Design Plan

1. **Construct a model to find all plants in MISO system, categorize them into regions to create the *Regional Generation Source Data (RGSD)***

   This design stage will require interfacing with emissions data provided by government agencies like EIA (Energy Information Administration) and internal MISO data. With data links established, a model of all generation plants in the MISO footprint will be created.

2. Construct a module to import historical emissions data from all plants Plant Historical Emissions Data (PHED)

Similar to task one, this stage of our design will rely on data importing from MISO, EIA, and state regulatory agencies. Expanding on task one, data associated with each generation plant will be imported into CSV files for long term data storage.

3. Construct a module to import real time data from all plants Plant Instantaneous Emission System (PIES)

The task/subtask goals for (3) are the same as (2).

4. Construct the dashboard to visualize regional emissions data.

Using the collated emissions data, the Highcharts graphing library will be used to display the emissions for the MISO network and the three regions within.

5. Construct the dashboard view to visualize MISO total emissions data

The task/subtask goals for (5) are essentially the same as for (4).

6. Host dashboard and code on MISO server. Format all code to MISO standards.

This task will primarily focus on systems integration with existing MISO sites and standards for data visualization to present a cohesive user experience and allow for MISO data analysts and programmers to modify our dashboard.

7. Write a report analyzing emissions trends

Using the dashboard and the associated data model, a report will be created that analyzes emissions trends across the MISO network and within the regions of the MISO network. This report will highlight the utility of the dashboard and create a discussion for future data analysis activities.

Testing

Unit Testing

Each data importing module will need to be tested separately. This can be done by comparing the imported data to the data source. Because the imported data is simply extracted from larger datasets with more information, our imported data should exactly match a subsection of the source data. To verify our operations were performed correctly, unit testing is performed by comparing a section of imported data to the corresponding section of the data source.

All data visualization subsystems need to be tested to verify their correct operation characteristics. To accomplish this, test data will be fed into our visualizations to verify correct visual behavior and interactivity. Using test data allows the team to construct test cases reflective of the diversity of data contained within the existing MISO and EIA datasets without spending time running all of the imported data through our visualizations.

Interface Testing

In our data importing modules, the interfaces between modules will be the EIA plant identifier and applicable MISO unique identifiers for each generation plant. To maintain interface integrity, we will need to test for timely handoffs of plant identifiers. The preservation of plant identifiers between interfaces will also need to be verified. To test for this, the team will compare the plant identifiers passed on at each step to the original plant identifiers associated with a plant from the source data, verifying that our code is not distorting plant identifiers.

Additionally, our dashboard library, Highcharts, will need to be verified. This is an important interface between our web environment and the input data. To verify that this interface is performing correctly, test data can be input into our Highcharts code to verify the use of the correct parts of the library vis-a-vis our specifications and the input data.

Integration Testing

What are the critical integration paths in your design? Justification for criticality may come from your requirements. How will they be tested? Tools?

1) Design Requirements

a. Graphic interface

i. Daily Emission Chart

1. Line Chart showing Emissions of MISO / MISO regions up to current

hour starting from 00:00

ii. Interactable by public website user

1. Filters by region on emission output

2. Allow users to download recent data (Data up to that hour of the day

starting from 00:00)

3. Allow graph to be “popped out” to a larger standalone display

4. Include a description of chart and data being used to make it

a. Example: current MISO real time charts have a (i) which, when clicked on give information.

5. Includes area, generation type, possibly fuel type filters

iii. Similarity to Current MISO website graphs

1. Hours X axis, Emission Y axis

2. Use MISO colors

b. Software program design

i. Python is widely used in MISO for automation and data analysis. Students can utilize python for any data manipulation or automation needed

ii. Utilize JavaScript (https://www.javascript.com/) as the code base for the project graphical implementation.

1. Scripts must be able to be utilized to create widgets to be plugged into a website

2. Highcharts (https://www.highcharts.com/) is the preferred javascript library to utilize for chart creation

a. Library is able to be readily downloaded here for free: https://www.highcharts.com/download/

c. Assumption development

i. Maintain documentation of assumptions made and any sources/discussion notes leading to those assumptions

ii. Any formulas or derivations made to reach emissions calculations need to be fully described and maintained in a list of formulas used

iii. Maintain cited source list of all sources of information for any calculations which assist in the emission derivation effort

d. Information access and availability

i. All information used must be publicly accessible information

e. Program documentation

i. At the conclusion of the project, a write up of the overall project will be created with

the intent to inform others of the work done and share knowledge to allow expansion upon the foundation created by the project

Critical Integration Path in our design

5 Assumption Development - While discussing with MISO employees, we decided on some assumptions that will go into our data sources. One example is if a generator

in the MISO region also feeds power into a different ISO, we will assume it is only feeding the MISO region.

6 Gathering information - Next, we are going to pull together public information using MISO public data and EIA (Energy Information Administration) data based on our assumptions.

7 Next, we will start developing the interface using Python, JavaScript, and HighCharts. Some requirements of the interface include having a visible line chart, filtering by region, and allowing the user to download recent data. HighCharts will be used mainly for the graphics.

8 The final step will be checking to see if all visual aspects are implemented and write a report documenting any conclusions we came to.

Acceptance Testing

From our testing, we want to ensure data identifiers are preserved between operations, the correct data is being graphed, and the dashboard output matches our initial specifications. This process will ensure compliance with our project requirements by ensuring that our underlying data is maintained with a high degree of fidelity and that the functional requirements of our project are met. Our overall design needs to maintain the desired interactivity and user experience defined by our project sponsor, and our testing plan for the dashboard components needs to be focused on specific functional requirements to ensure that our sponsor's needs are being met in an expeditious fashion.

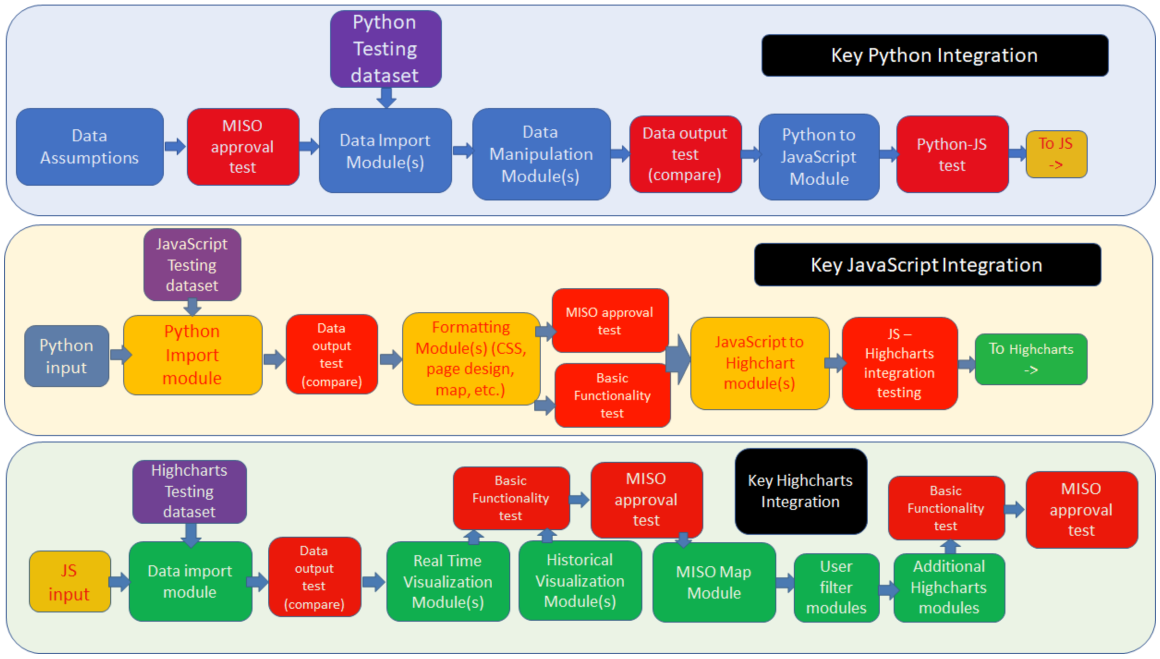

Key Python Integration
Python Testing dataset
Data Assumptions
MISO approval test
Data Import Module(s)
Data Manipulation Module(s)
Data output test (compare)
Python to JavaScript Module
Python-JS test
To JS ->
Key JavaScript Integration
JavaScript Testing dataset
Python input
Python Import module
Data output test (compare)
Formatting Module(s) (CSS, page design, map, etc.)
MISO approval test
Basic Functionality test
JavaScript to Highchart module(s)
JS – Highcharts integration testing
To Highcharts ->
Highcharts Testing dataset
Key Highcharts Integration
JS input
Data import module
Data output test (compare)
Real Time Visualization Module(s)
Basic Functionality test
Historical Visualization Module(s)
MISO approval test
MISO Map Module
User filter modules
Additional Highcharts modules
Basic Functionality test
MISO approval test

Figure 4. Testing Plan

Implementation

Engineering Constraints

The largest constraint that we had to work under in the project was the fact that we had to only use public data for the entire project. This meant that we would have large limitations on where we could receive our data, which would primarily be government public data from the NIA and data that was approved for public release by MISO. Some more constraints were that we only had two sources of data from which we could draw, the EIA and MISO's real time data stream. These two streams defined the way we could approach our project through our two halves of historical and real time data analysis in that the historical data would only be run once a year, while the real time was updated in real time. This necessitated that we handle the real time data code in JS as it would be run whenever the page was opened. However, for the historical data, it would be run through python as it was a much larger data set only needing to be processed once a year. Another constraint that we had to deal with was the interfacing behind our backend python code and our frontend javascript and HTML. The nature of this constraint is that there is a fundamental mismatch between these two types of coding languages as the python code processed the data that we would scrape and then output a CSV with the data we needed. However, this python code could not generate a webpage and as such, necessitated the use of HTML and JavaScript in order to tweak the outputs of the previous step into a form that could be processed by our Highcharts graphing library.

Project Evolution During Spring Semester

Our project has evolved during the current semester as we have focused more on the availability of public data and the realities of creating graphs. Last semester, our team planned to utilize Tableau for our frontend visualization tool. After speaking further with our stakeholders this semester, we shifted our focus to tools that were more readily used by MISO and more up to date with what is used in industry, such as the library called Highcharts.

During this semester, we have been able to rely on our calculated and obtained historical data and have shifted our focus to the realities of creating graphs. This led to the research and visualization of Highcharts using JavaScript with HTML code.

Development of the graphs from historical data started with reviewing the Highcharts documentation and viewing examples of graphs that shared a resemblance to what we wanted our final emissions dashboard to have. This included graphs such as line charts, pie charts, and other graphs that were considered, but ultimately weren't implemented due to time constraints and going back to relating to the core concepts of the project goal from MISO.

After researching and understanding visualizations on Highcharts and what we had at our disposal, the next step was to understand JavaScript and research how to display some of the example graphs in our own HTML local server (as we don't have a readily available web server). This involved much needed collaboration with the team to get a working example with real data, but once visualizations were created, parsing of our own generated data could start.

The parsing of our own data was done through Python with the already grabbed data from the EIA 860 data from eia.gov. This data was parsed into a readable format with rows and columns that could be easily read from HighCharts into either a comma-separated variable format or a JSON format, but ultimately the JSON approach was scrapped due to unnecessary work. Once the data was formatted on the backend correctly, JavaScript could then grab the CSV files that would be used to implement the historical data from functionality given through Highcharts. After we could see the data in Highcharts with our own recovered data, we completed the line graphs with this data and did backend math to create pie charts with the percentage of data.

Additionally, our real time data importing procedure changed during this semester. The initial plan was to scrape data from the publicly available real time generation data on MISO’s website. This created some difficulties as the data available in this source only contained data from the nearest 5 minute interval and did not have any knowledge of the generated data from earlier in the day. Adding additional complexity, designing a web scraping tool to access this data could prove to be difficult. After consulting with MISO web developers, they were able to create a web resource that would store all of the real time generation data for the current day in an accessible JSON. Using this resource, we were able to create code to simply download the JSON file at a specified URL to access the real time generation data from the current day.

While the majority of the historical data scraper was completed early in the spring semester, during the development of the final output code used to sum emissions data from the various regions, a serious data inaccuracy was found based on the overlooked fact that generation facilities can change ISOs over time. Getting to the source of this problem took significant debugging efforts and ended up requiring a major rewrite of the backend code to correct. Once this was fixed, the data scraper was able to output the correct sums for tons of emissions, total MW generation, and tons of emissions per MW filtered by MISO region or fuel type.

The other main improvements to the data scraper were the outputting of SO2 and NOX emissions data, rather than only CO2, cleaning up the local temp storage files and folders into a more structured format, and cleaning up comments to make the code more readable.

Overall, our project vision has remained similar and the core requirements have not changed from the end of 491. We have just scaled back in a few technical endeavors as the reality of their implementation became apparent. This has led to decisions like choosing to not create graphs using latitude and longitude data, using EIA 860 data, and accessing additional information from the EIA API, as the skills required to implement geographical charts and diversity of data sources are more highly specialized and distinct from the main skills needed for our core project deliverables.

Security Concerns

Physical Security

Due to the nature of the design being a webpage, the team does not expect to see any physical security concerns. Upon transition of the codebase to the internal MISO team, the only possible threat may include a breach of MISO servers. This is considered a non-critical threat because all data used in our visualizations are public data that can be readily found on various websites.

Cyber Security

For the cyber security of our project, all of our code is being hosted upon MISO internal servers and will be protected by their own internal suite of security methods. All of the data that we use within our design is public information from MISO or from the US Government's Energy Information Administration's Website should be a fairly secure data stream.

Technical Details

Real Time Data Importing

For importing and displaying real time data, a special URL has been prepared by MISO engineers to allow our team to access an aggregated version of the public real time generation data. The data at this URL is then obtained on the client side using a JavaScript function detailed below.

```javascript
export function getGenerationData(url) { //This function takes in a
URL and returns the real time generation data JSON

  let req = new XMLHttpRequest(); //create new request
  req.open('GET', url, false); //set request to synchronous
  req.send();//Send null data to complete GET request
  if (req.status == 200) {
    return JSON.parse(req.responseText) //Return parsed json
  } else {
    console.log('Error: ',req.status)
  }
}
```

Figure 5. getGenerationData function

The function above creates a synchronous XMLHTTPRequest to obtain the generation data and then returns the entirety of the contents of the given URL as a JSON.

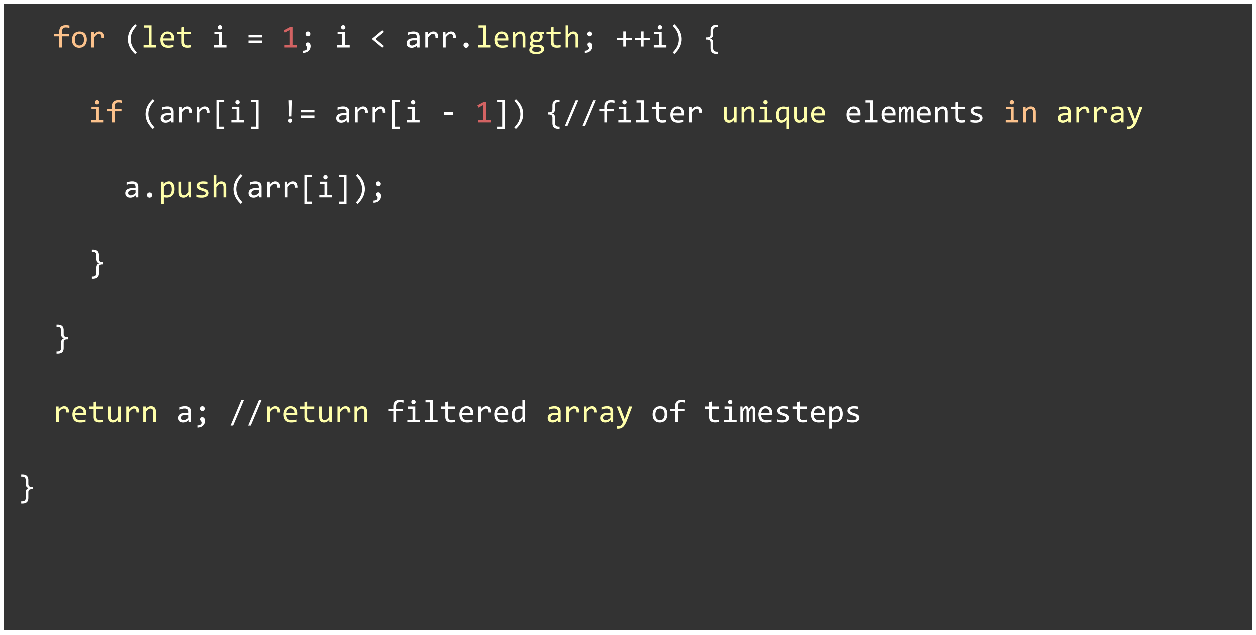
Next, in order to create graphs, the JSON returned by the getGenerationData() function is parsed to obtain the time and generation data. Below is the function that takes in a generation category as a string and the full real time data JSON and returns an array of values corresponding to the amount of electricity generated by the given category at 5 minute intervals until the current time.

```
export function filterSeriesData(obj, cat) {//Filter full json to extract generation data for each fuel type
  let arr = [];
  for (let i in obj.Fuel.Type) {
    if (obj.Fuel.Type[i].CATEGORY == cat) {
      arr.push(parseInt(obj.Fuel.Type[i].ACT));//Adds electricity generated at each timestep to an array, converts string to int'
    }
  }
  return arr; //return filtered array
}
```

Figure 6. filterSeriesData function

Next, the set of unique timesteps is extracted from the real time generation data. This data is directly extracted from the real time data to allow for a reliable matching of timesteps to generation values at the cost of a more computationally expensive data management operation.

```
export function filterTimeData(obj) {//Return unique timesteps from
real time data
  let arr = [];//create blank arrays
  let a = [];
  for (let i in obj.Fuel.Type) {

    if (obj.Fuel.Type[i].INTERVALEST != arr[arr.length - 1] || i ==
1) {//These conditions are maybe not helpful, but it works
      var time = obj.Fuel.Type[i].INTERVALEST;
      time = time.substring(time.indexOf(' ') + 1);//Strip out date
information
      arr.push(time)//add to arr
    }
  }
  a.push(arr[0]) //add first element of arr to a, needed to capture
first timestep
```



```
  for (let i = 1; i < arr.length; ++i) {
    if (arr[i] != arr[i - 1]) {//filter unique elements in array
      a.push(arr[i]);
    }
  }
  return a; //return filtered array of timesteps
}
```

Figure 7. filterTimeData function

The three functions detailed so far constitute the bulk of the data processing operations used to create all of the real time data charts.

Historical Data Importing

The historical data scraper (also referred to as backend code) is intended to be used once per year to generate the files needed for the historical data viewing. The intention is that the script is to be run manually inside some sort of development environment. The code is split up into <Main> and <Functions> files, with the Main file used to set user desired variables such as folder and filenames, create the temporary folders used by the data scraper, and call the individual routines defined in the Functions file, which contains the bulk of the code.

After creating the list of years, Main calls a function which downloads the EIA 860 plant location data from the EIA website. This function is summarized below.

```python
def download_eia_860_data():
    pickle_file = (output_filename_head + str(year) + ".pkl")#temp file for dev
    pickleExists = os.path.isfile(dir_output01 / pickle_file)
    if not pickleExists:
        urlData = base_url + specific_url+ str(year) + '.zip'#link to EIA data
        getfolder = requests.head(urlData) #check our initial link
        if getfolder.status_code==200:  #if good link we download
            download = True #set this bool for later use
        else:
            urlData = base_url + 'archive/' + specific_url + str(year) + '.zip'
            getfolder = requests.head(urlData)
            if getfolder.status_code == 200:#if good link we download archived file
                getfolder = requests.head(urlData)
                download = True #bool for use later
            else:   #this handles the yearly data file not existing
                print('Data does not exist for year ' + str(year))
                download = False
        if download:#if data exists for year we download and store as pickle file
            getfolder = requests.get(urlData)
            archive = zipfile.ZipFile(io.BytesIO(getfolder.content))
            xlfile = archive.open('2___Plant_Y' + str(year) + '.xlsx')
            df2 = pd.read_excel(xlfile, skiprows=1)
            df2.to_pickle(dir_output01 / pickle_file)
        if not download:    #if no data exists for the year we exit
            print("No data exists for " + str(year))
            return
    dropCols = [0, 1, *range(14, len(df2.columns), 1)]
    df2 = df2.drop(df2.columns[dropCols], axis=1)#trim data
    df2.insert(0, 'Year', year) #adding year marker
    df2 = df2.sort_values(by='Plant Code', ascending=True)  #sorting by plant code
    csvfilename = output_filename_head + str(year) + "CSV"
    df3 = df2[df2['Balancing Authority Code'] == 'MISO']
    csvfilename_miso = output_filename_head + str(year) + misotag
    df2.to_csv(dir_output01 / csvfilename, encoding='utf-8', index=False)
    df3.to_csv(new_dir_out/ csvfilename_miso, encoding='utf-8', index=False)
    return
```

Figure 8. EIA 860 Download Code

The function shown above looks for the EIA860 data for each year in “yearlist” and builds the download link to the “Plant Data” Excel sheet inside the EIA860 yearly zip archive and then downloads only that file to avoid unnecessary file downloads. The function has to “build” the download link, as the EIA archives data and changes the download link format after archiving. Once the file is downloaded, the data is loaded into a data frame, stored as a “pickle file” (for development reasons), unwanted columns are removed, the data year is appended to the data frame, filtered for MISO controlled plants, and output as a local CSV file for later use. These files are then combined into a single CSV file and data frame with the “loadCSV” function, which simply combines and sorts the yearly EIA80 files.

The next function to be called is the download_eia_emissions function:

```
def download_eia_emissions():
    excel_file = (eia_emissions_raw_head + str(year) + ".xlsx")   #build filename
    excelExists = os.path.isfile(dir_output02 / excel_file)
    if excelExists:
        return #file already downloaded, no need to re-download
    Else:
    #build emissions url
    urlEmissions = baseurl + specificurl + "xls/emissions"+ str(year) + '.xlsx'
    test_url = requests.head(urlEmissions)  #test if our url is good
    if test_url.status_code == 200:     #we have a good link
        download = True
    else:   #we try to 'fix' the url so we can download
        urlEmissions = baseurl + specificurl + 'archive/xls/emissions' + str(year)
+ '.xlsx'
        #test to see if we have a good response with modified url
        test_url = requests.head(urlEmissions)
        if test_url.status_code == 200:     #if we find the url
            download = True
        else:       #if we don't find a file we set download to no
            download = False
    if download:    #if file found we download
        urllib.request.urlretrieve(urlEmissions, dir_output02 / excel_file)
        last_good_year = year #used to mark the last year of data
    if not download:    #if no file (yearly data not released, we return)
        last_good_year = year - 1
        print("EIA emissions data for " + str(year)+ " has not released.")
        return last_good_year
    return last_good_year
```

Figure 9. EIA Emissions Downloader

The function shown above works similar to the download EIA860 data, we build a download link and modify it as needed to download the yearly data. The files are stored locally and the function returns the last year of emissions data. The files are then processed by the following functions

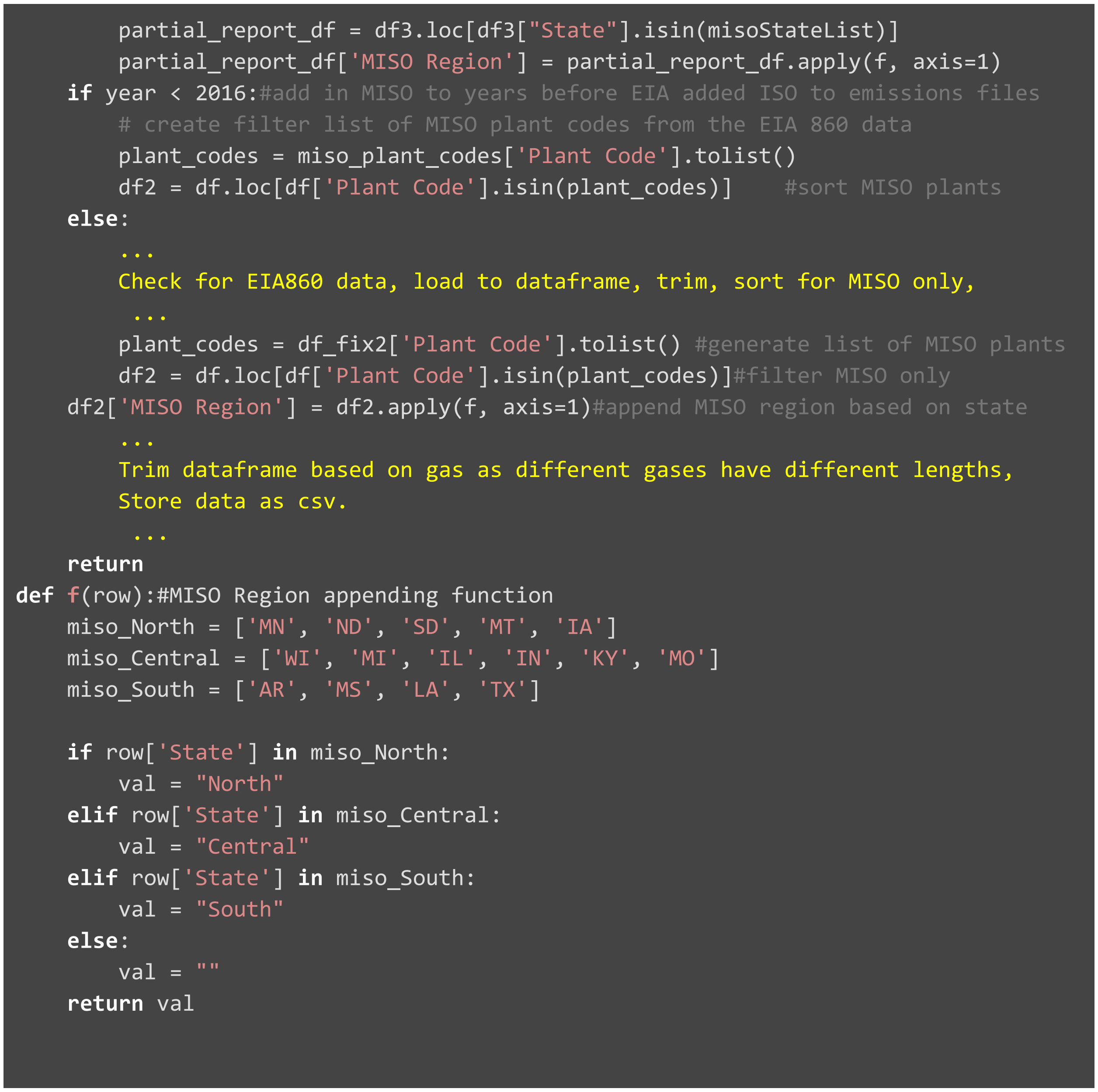
```
def extract_EIA_emissions_data():
    excel_file = (input_head + str(year) + ".xlsx")   #create excel filename
    outfilename = output_head + str(ingas) + "_" + str(year) + "CSV"#output file
    outputExists = os.path.isfile(dir_OUT_03 / outfilename)
    if outputExists:
        print("Files already split for year" +str(year))
        return
    inputExists = os.path.isfile(dir_01_input / excel_file)
    if inputExists:#check for inputfile
        load = True
    else:
        load = False
    if load:    #if we have a file we pull out the data
        sheet = pd.read_excel(dir_01_input / excel_file, sheet_name=[0, 1, 2],
skiprows=1)
        #here we loop through sheets and set the gas value based on sheet index
        for i in sheet:
            if i == 0:
                gas = "co2"
            elif i == 1:
                gas = "so2"
            elif i == 2:
                gas = "nox"
            else:
                raise Exception("Source files changed. Look at EIA emissions
website")
            df = sheet[i]   #create dataframe from dict
            df = df.iloc[1:]    #skip first header row
            df = df.dropna(subset=['Plant Name'])   #trim the trailing rows with no
data
```

```
            #here we create a new column for unique id
            df['UniqueID'] = df['Plant Code'].map(str) + "_" + df['Prime
Mover'].map(str) + "_" +  df['Fuel Code'].map(str) + "_" + df['Aggregated Fuel
Group'].map(str)
            first_column = df.pop('UniqueID')   # shift 'ID' to first position
            df.insert(0, 'UniqueID', first_column)  #insert ID to first location
            df.insert(0, 'Year', year)  # adding year marker
            test_col = df.columns.values.tolist()  # create list of column names
            for i in test_col:
                i = i.replace("\n", "")#remove newline char
            csvfilename = output_head + str(gas) + "_" + str(year) + "CSV"
            df.to_csv(dir_OUT_03 / csvfilename, encoding='utf-8', index=False)
            csvfilename = output_head + "original_format_" + str(gas) + "_" +
str(year) + "CSV"
            df.to_csv(dir_OUT_07 / csvfilename, encoding='utf-8', index=False)
        return
```

Figure 10. Extract EIA Emissions

The extract_EIA_emissions function loads the yearly emissions Excel workbooks and extracts the individual sheets for CO2, SO2, and NOX gasses, and stores them as csv files. This function also appends a year column and creates a “UniqueID” column used to account for generation facilities that use more than one type of fuel for the data frames. The files are then processed by the following function:

```
def extract_MISO_emissions_data():
...
#define filenames and MISO states
#check for input file existence..
...

    misoStateList = ['MN', 'ND', 'SD', 'MT', 'IA', 'WI', 'MI', 'IL', 'IN', 'KY',
'MO', 'AR', 'MS', 'LA', 'TX']
    #load EIA860 dataframe
    plant_codes = miso_plant_codes['Plant Code'].tolist()
    df = pandas.read_csv(dir_01_input_emis_files/sourcefile)#readcsv for gas/year
    if year < 2018:#remove partial reporting facilities
        df3 = df[df['Plant Code'] == 99999]
        df3["State"] = df3["State"].apply(str)
```



```python
        partial_report_df = df3.loc[df3["State"].isin(misoStateList)]
        partial_report_df['MISO Region'] = partial_report_df.apply(f, axis=1)
    if year < 2016:#add in MISO to years before EIA added ISO to emissions files
        # create filter list of MISO plant codes from the EIA 860 data
        plant_codes = miso_plant_codes['Plant Code'].tolist()
        df2 = df.loc[df['Plant Code'].isin(plant_codes)]    #sort MISO plants
    else:
        ...
        Check for EIA860 data, load to dataframe, trim, sort for MISO only,
         ...
        plant_codes = df_fix2['Plant Code'].tolist() #generate list of MISO plants
        df2 = df.loc[df['Plant Code'].isin(plant_codes)]#filter MISO only
    df2['MISO Region'] = df2.apply(f, axis=1)#append MISO region based on state
        ...
        Trim dataframe based on gas as different gases have different lengths,
        Store data as csv.
         ...
    return
def f(row):#MISO Region appending function
    miso_North = ['MN', 'ND', 'SD', 'MT', 'IA']
    miso_Central = ['WI', 'MI', 'IL', 'IN', 'KY', 'MO']
    miso_South = ['AR', 'MS', 'LA', 'TX']

    if row['State'] in miso_North:
        val = "North"
    elif row['State'] in miso_Central:
        val = "Central"
    elif row['State'] in miso_South:
        val = "South"
    else:
        val = ""
    return val
```

Figure 11. Further EIA Emissions Code

The function shown above loads the yearly, gas-specific CSV files and trims them and adds data as required. EIA emissions files prior to 2016 do not include balancing authority information for the plants, so the code uses the EIA860 data, which does contain this information, to create a lookup table to filter MISO only plants for these years. This function also assigns each plant a MISO region value based on the state the plant is located in and also removes the “partial reporting facilities.”

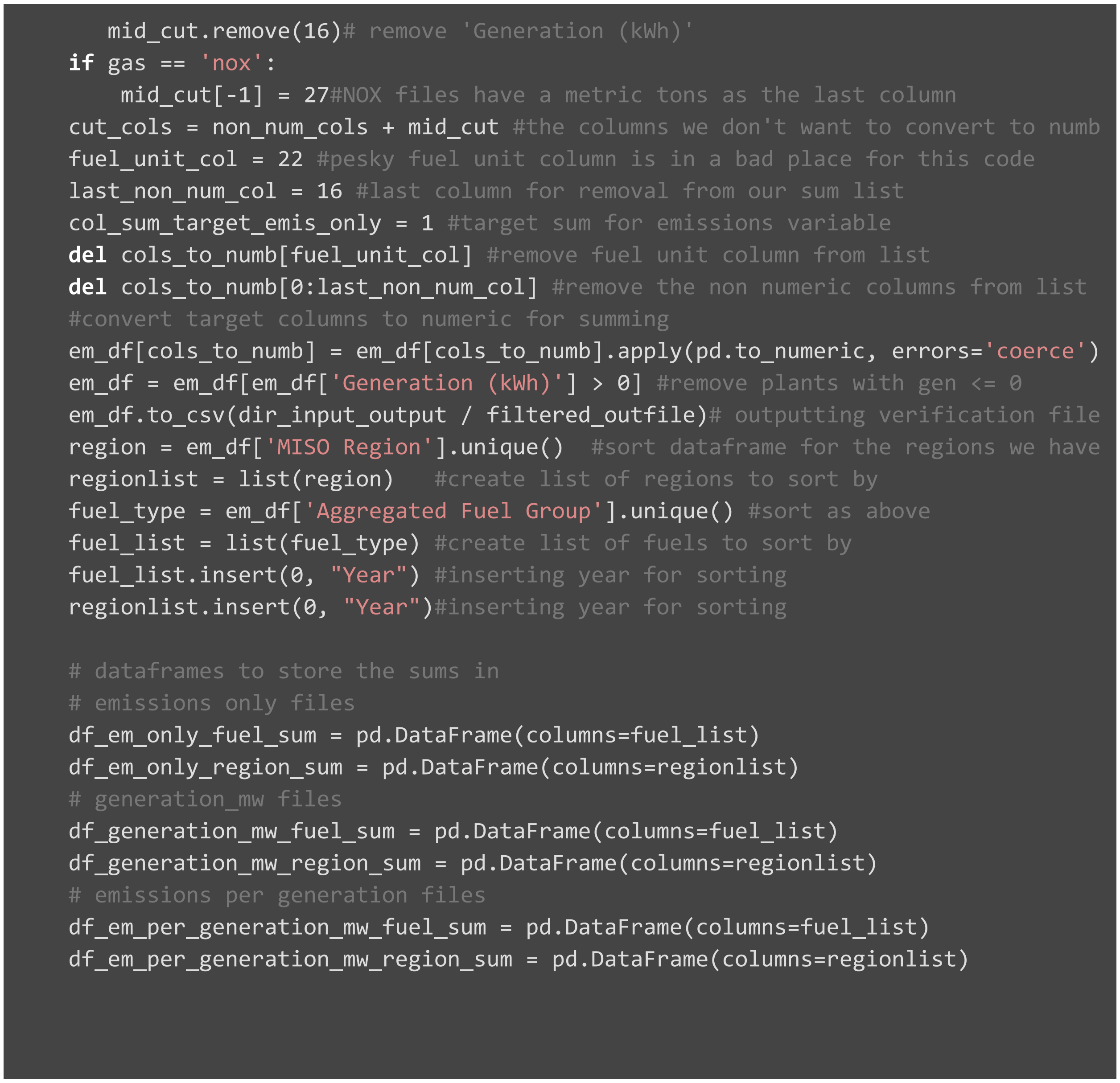
Main then calls the “ready location data” function (not shown), which simply gets the yearly EIA860 CSV files ready for insertion into the yearly emissions files by trimming and rearranging the data. Following this, the “load MISO emissions to database” function is called:

```
def load_MISO_emissions_to_db():
...
#define input filenames
#check for input file existence..
...
    em_df = pd.read_csv(source_dir / misoFile)# load in emissions
    db_df = pd.read_csv(source2_dir / databaseFile)# load in EIA860 database
# define our last column of emissions data for each gas
    if gas == 'co2':
        last_good_col = 19
    elif gas == 'so2':
        last_good_col = 23
    elif gas == 'nox':
        last_good_col = 23
    else:
        raise Exception("Gas not found in data.")  # exit if
    em_df = em_df.iloc[:, :last_good_col]#here we trim the df
    # prepare to merge emissions data into database
    test_col = db_df.columns.values.tolist()  # create list of columns
    test_col.pop(0)
    del test_col[1:11]  # remove ID from our list

    # merge emissions data into database
    col = 'UniqueID'
    cols_to_replace = test_col
    em_df.loc[em_df[col].isin(db_df[col]), cols_to_replace] = db_df.loc[
        db_df[col].isin(em_df[col]), cols_to_replace].values
    #reorder dataframe
    test_col = em_df.columns.values.tolist()  # create list of columns
    static_cols = list(range(0, 11))
    last_element = len(test_col)
    first_new = last_element - 5
    new_cols = list(range(first_new, last_element))
    em_cols = list(range(11, first_new))
    new_order = static_cols + new_cols + em_cols
    em_df = em_df.iloc[:, new_order]
    # store file as csv for verification
    # TODO Change the filename output here
    em_df.to_csv(target_dir / outputFile, encoding='utf-8', index=False)
    return
```

Figure 12. Load EIA Emissions Data

The function above combines the yearly emissions files with the yearly EIA 860 files to create a more comprehensive emissions file containing precise location information (Zip Code, County, latitude and longitude data, etc.). This was done with the intention of using this location data to create a geographical visualization tool. Our team ran out of time to implement the geographical view, but the data still exists in the output files. These files are then combined into a single master emissions database file in the “create master” function (not shown), which is essentially a copy of the “combine csv files” used earlier, and the output files from this function are used for certain graphs. Finally, we create a list of years that data has been found for and call “sum EIA emissions”:

```
def sum_EIA_emissions(dir_input_output, year, gas, infilehead):
    hours_in_year = 8760 #hours in a year for kWHr to MW
...
#define input filenames
#check for input file existence..
...
    em_df = pd.read_csv(dir_input_output / misoFile)
...
#code to handle the differences in target column locations for various gas files
...
    cols_to_numb = em_df.columns.values.tolist()#initial column list
    non_num_cols = list(range(1, 9))#columns we know don't need conversion
    last_element = len(cols_to_numb)#the last element of the dataframe should be the tons of "gas" emissions
    mid_cut = list(range(11, last_element - 1)) #additional columns we don't want to sum
```



```
       mid_cut.remove(16)# remove 'Generation (kWh)'
    if gas == 'nox':
        mid_cut[-1] = 27#NOX files have a metric tons as the last column
    cut_cols = non_num_cols + mid_cut #the columns we don't want to convert to numb
    fuel_unit_col = 22 #pesky fuel unit column is in a bad place for this code
    last_non_num_col = 16 #last column for removal from our sum list
    col_sum_target_emis_only = 1 #target sum for emissions variable
    del cols_to_numb[fuel_unit_col] #remove fuel unit column from list
    del cols_to_numb[0:last_non_num_col] #remove the non numeric columns from list
    #convert target columns to numeric for summing
    em_df[cols_to_numb] = em_df[cols_to_numb].apply(pd.to_numeric, errors='coerce')
    em_df = em_df[em_df['Generation (kWh)'] > 0] #remove plants with gen <= 0
    em_df.to_csv(dir_input_output / filtered_outfile)# outputting verification file
    region = em_df['MISO Region'].unique()  #sort dataframe for the regions we have
    regionlist = list(region)   #create list of regions to sort by
    fuel_type = em_df['Aggregated Fuel Group'].unique() #sort as above
    fuel_list = list(fuel_type) #create list of fuels to sort by
    fuel_list.insert(0, "Year") #inserting year for sorting
    regionlist.insert(0, "Year")#inserting year for sorting

    # dataframes to store the sums in
    # emissions only files
    df_em_only_fuel_sum = pd.DataFrame(columns=fuel_list)
    df_em_only_region_sum = pd.DataFrame(columns=regionlist)
    # generation_mw files
    df_generation_mw_fuel_sum = pd.DataFrame(columns=fuel_list)
    df_generation_mw_region_sum = pd.DataFrame(columns=regionlist)
    # emissions per generation files
    df_em_per_generation_mw_fuel_sum = pd.DataFrame(columns=fuel_list)
    df_em_per_generation_mw_region_sum = pd.DataFrame(columns=regionlist)
```

Figure 13. EIA Emissions Summation

This first section of the “Sum EIA Emissions” function sets the values we want to convert to numeric and selectively trims the data frame to the correct length for summing the gas emissions based on the gas we are summing for. This portion of the code also removes plants with generation less than or equal to zero and prepares the dataframes and lists we will use to filter files and store the summed data.

```
    #beginning loop for sums
    for y in year:
        # copy dataframe for manipulation
        df_filtered_em = em_df.copy()
        # remove unwanted columns from the dataframe
        df_filtered_em_gen = df_filtered_em.drop(df_filtered_em.columns[cut_cols],
axis=1)
        # df_filtered_em_only holds year, region, fuel, and tons of gas emissions
        df_filtered_em_only =
df_filtered_em_gen.drop(df_filtered_em_gen.columns[3], axis=1)

        # filter dataframes for current year info only
        df_filtered_em_only = df_filtered_em_only[df_filtered_em_only['Year'] == y]
        df_filtered_em_gen = df_filtered_em_gen[df_filtered_em_gen['Year'] == y]
        #changing the column name for ease of incorporation of different gasses
        fix_gas = df_filtered_em_gen.columns.values.tolist()
        column_to_replace = str(fix_gas[4])
        df_filtered_em_gen = df_filtered_em_gen.rename(columns={column_to_replace:
'Tons of gas emissions'})
        df_filtered_em_gen['MW Generation'] = (df_filtered_em_gen['Generation
(kWh)'] / hours_in_year) / 1000
        df_filtered_em_gen['Tons of Emisssions per MegaWatt'] = (
                    df_filtered_em_gen['Tons of gas emissions'] /
df_filtered_em_gen['MW Generation'])
        # emissions only sums
        list_sums_em_region = [y]
        list_sums_em_fuel = [y]
        # generation only sums
        list_sums_gen_region = [y]
        list_sums_gen_fuel = [y]
        # em/generation sums
        list_sums_em_per_Kw_gen_region = [y]
        list_sums_em_per_Kw_gen_fuel = [y]
```

Figure 14. EIA Emissions Per Generation Code

The above portion of the “Sum EIA Emissions” function begins the summing process by looping through each year (we will later loop through each type of fuel and region) and preparing the emissions dataframes for summing. The “Tons of <gas> emissions” column is changed to a generic name, the dataframes are filter for year, generation in kWHr is converted to MegaWatts, tons of emissions per MW are calculated, and the lists which will hold the yearly summed values are created. The function then loops through the values for Miso Region and Fuel Type and creates the sums. This is shown for “Fuel Type” below, but “MISO Region" works similarly:

```
        for i in fuel_type:
            # filter emissions only df
            df_filtered_em_only_fuel =
df_filtered_em_only[df_filtered_em_only['Aggregated Fuel Group'] == str(i)]
            # filter emissions and generation df
            df_filtered_em_gen_fuel =
df_filtered_em_gen[df_filtered_em_gen['Aggregated Fuel Group'] == str(i)]
            # get emissions only totals
            t_em_tons_fuel = df_filtered_em_only_fuel.sum(numeric_only=True)
            t_em_tons_fuel.name = 'Total_em_tons'
            t_em_tons_fuel = t_em_tons_fuel.to_numpy()
            list_sums_em_fuel.append(t_em_tons_fuel.item(col_sum_target_emis_only))
            # MODIFY FOR GENERATION
            # get emissions only totals
            t_gen_fuel = df_filtered_em_gen_fuel.sum(numeric_only=True)
            t_gen_fuel.name = 'Total_generation'
            t_gen_fuel = t_gen_fuel.to_numpy()
            list_sums_gen_fuel.append(t_gen_fuel.item(3))
            # MODIFY FOR EM/GEN
            # get emissions only totals
            t_em_per_genfuel = df_filtered_em_gen_fuel.sum(numeric_only=True)
            t_em_per_genfuel.name = 'Total_generation'
            t_em_per_genfuel = t_em_per_genfuel.to_numpy()
            list_sums_em_per_Kw_gen_fuel.append(t_em_per_genfuel.item(4))
```

Figure 15. Regional EIA Data Summation

In the above portion of code, we find the sum for tons of emissions, generation in MW, and tons of emissions per MW. The combined loops filter the master emissions database by {[gas], [year], [Fuel Type/MISO Region]} and appends {(sum for tons of emissions), (sum for generation), (sum for emissions/generation)} to the respective lists.

```
        #for y in year loop
            ...
            #regional and fuel loops
            ...
 #sums for each year
        df_em_only_fuel_sum.loc[len(df_em_only_fuel_sum)] = list_sums_em_fuel
        df_generation_mw_fuel_sum.loc[len(df_generation_mw_fuel_sum)] =
list_sums_gen_fuel
        df_em_per_generation_mw_fuel_sum.loc[len(df_em_per_generation_mw_fuel_sum)]
= list_sums_em_per_Kw_gen_fuel
    #final output filenames
    outfile1 = "03_" + str(gas) + "_regional_emissions_only" + str(csv_test)
    outfile2 = "04_" + str(gas) + "_fuel_emissions_only" + str(csv_test)
    outfile3 = "03_" + str(gas) + "_regional_generation_only" + str(csv_test)
    outfile4 = "04_" + str(gas) + "_fuel_generation_only" + str(csv_test)
    outfile5 = "03_" + str(gas) + "_regional_em_per_gen" + str(csv_test)
    outfile6 = "04_" + str(gas) + "_fuel_em_per_gen" + str(csv_test)
    #writing output file
    df_em_only_region_sum.to_csv(dir_input_output / outfile1, encoding='utf-8',
index=False)
    df_em_only_fuel_sum.to_csv(dir_input_output / outfile2, encoding='utf-8',
index=False)
    df_generation_mw_region_sum.to_csv(dir_input_output / outfile3,
encoding='utf-8', index=False)
    df_generation_mw_fuel_sum.to_csv(dir_input_output / outfile4, encoding='utf-8',
index=False)
    df_em_per_generation_mw_region_sum.to_csv(dir_input_output / outfile5,
encoding='utf-8', index=False)
    df_em_per_generation_mw_fuel_sum.to_csv(dir_input_output / outfile6,
encoding='utf-8', index=False)
    return
```

Figure 16. EIA Emissions CSV Export

The final portion of the “SUM EIA Emissions” function is shown above. It appends the sums for the target values to three data frames for each year. Once all years have been found, the data frames are stored as CSV files for loading by the frontend code.

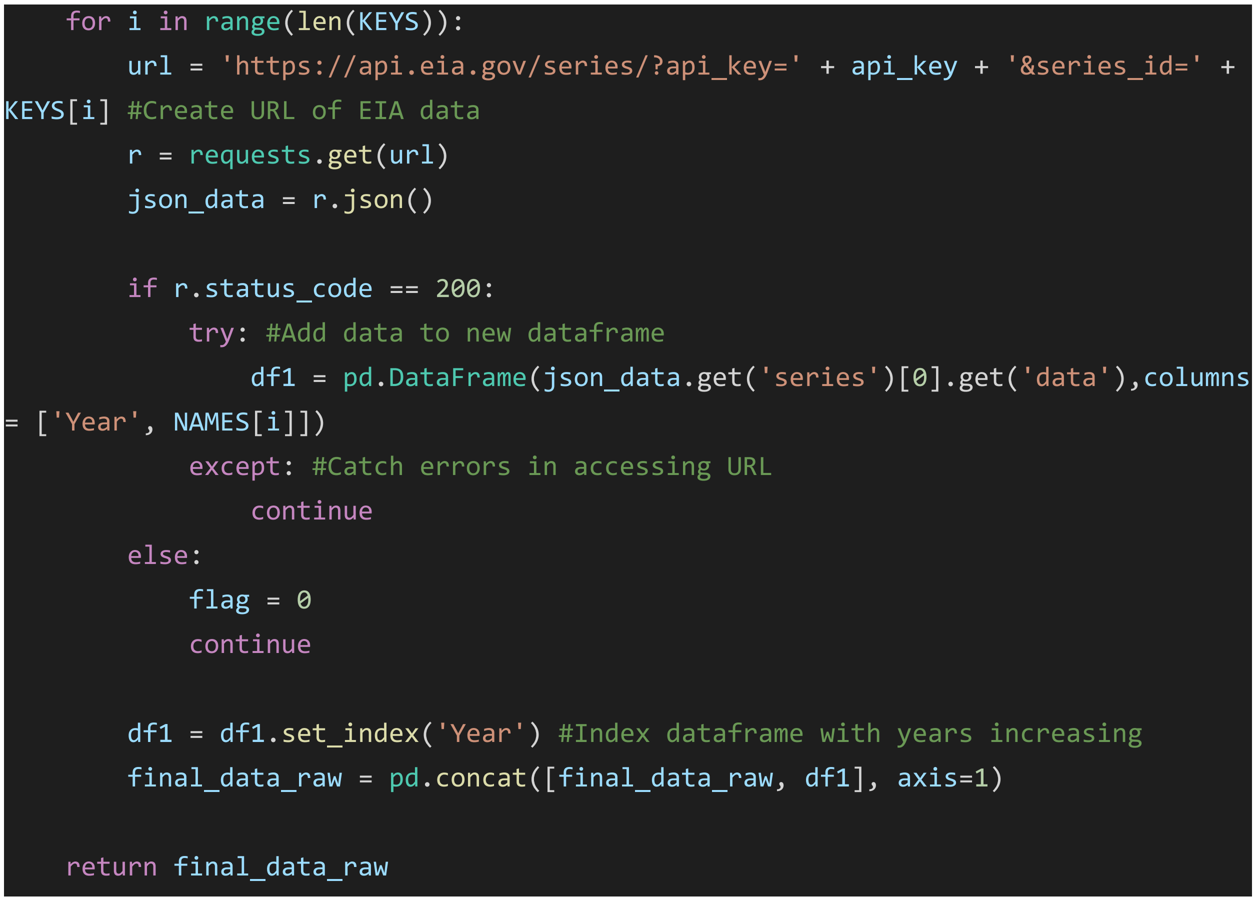
The historical data scraper (HDS) was written with certain assumptions and utilized the EIA webpages for acquiring data; if the team had more time, we would utilize the EIA API data instead of attempting to access the data via downloaded Excel files. Downloading the individual Excel file, instead of calling the API data, assumes that the EIA data links will not change. Assuming that the data we wanted to acquire was not available via the API, the code could be improved to use HTML redirects directly, rather than trying to rebuild the download link. However, it is likely that our customer will be able to use a better data source to input the historical data, which will end up being more accurate than the data we were able to use.

Another potential weakness with the HDS is running the code without a cleared local temporary file repository. There are multiple points where the backend code looks for locally stored data to reduce the run time during development; this is easily avoided by manually clearing out the temporary file directory but could be a problem for the client.

EIA API Data Import

The final approach used to create the graphs visible in the dashboard was using the EIA API[9]. Using this API, we were able to create Python functions to download and parse this data to create a dataset corresponding to hourly generation data for the entire MISO network. To import EIA API data, a Python function has been created to take in a URL formatted to EIA specifications, as well as an API key. This function is below.

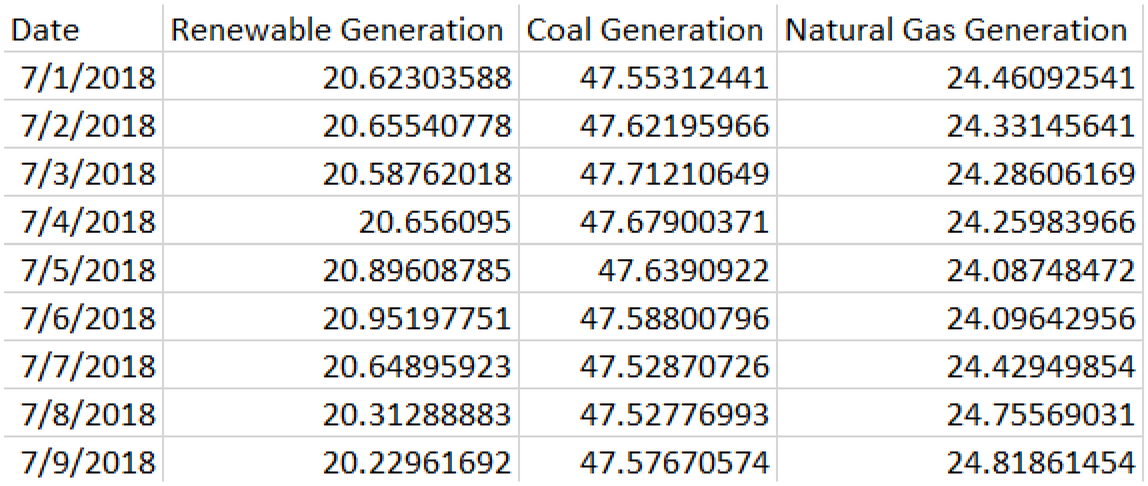
```
#Function to download data from EIA website
def grabData (api_key,KEYS,NAMES):
    final_data_raw = pd.DataFrame() #Declaring empty dataframe to be populated
```



```
    for i in range(len(KEYS)):
        url = 'https://api.eia.gov/series/?api_key=' + api_key + '&series_id=' +
KEYS[i] #Create URL of EIA data
        r = requests.get(url)
        json_data = r.json()

        if r.status_code == 200:
            try: #Add data to new dataframe
                df1 = pd.DataFrame(json_data.get('series')[0].get('data'),columns
= ['Year', NAMES[i]])
            except: #Catch errors in accessing URL
                continue
        else:
            flag = 0
            continue

        df1 = df1.set_index('Year') #Index dataframe with years increasing
        final_data_raw = pd.concat([final_data_raw, df1], axis=1)

    return final_data_raw
```

Figure 17. EIA API Data Import

The above function returns a Pandas data frame containing the entire available EIA dataset for a given URL. In practice, this function is used with the EIA API data for utility-scale generation in the MISO region to access data about electric generation from each generation source. The raw generation numbers are then converted into percentages of total generation at each timestep of data. Additionally, a 24 point moving average filter is applied to the dataset to convert the noisy hourly data into a daily dataset that can quickly load in a web environment. To finish, this data is then output to a CSV file for reading into Highcharts. An excerpt of this file is below.



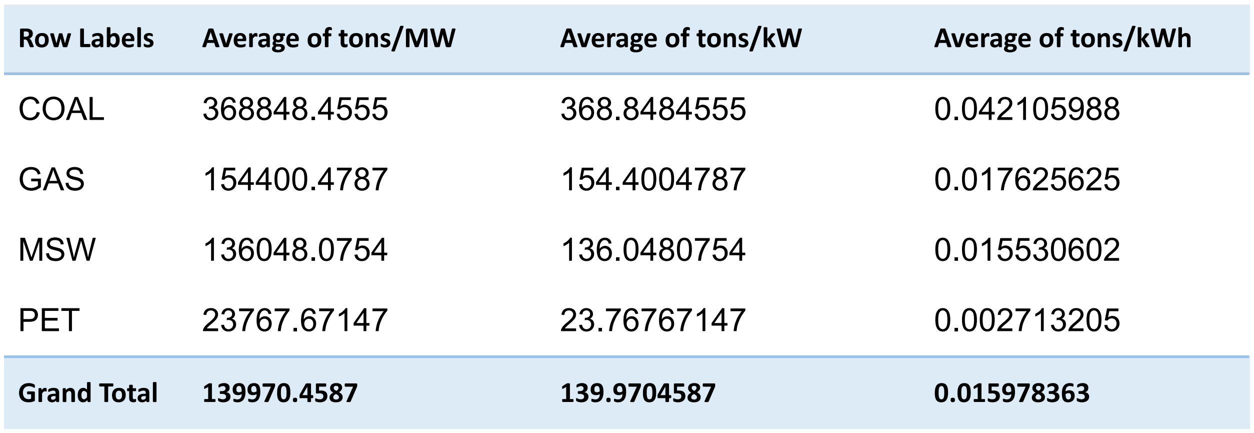
Date	Renewable Generation	Coal Generation	Natural Gas Generation
7/1/2018	20.62303588	47.55312441	24.46092541
7/2/2018	20.65540778	47.62195966	24.33145641
7/3/2018	20.58762018	47.71210649	24.28606169
7/4/2018	20.656095	47.67900371	24.25983966
7/5/2018	20.89608785	47.6390922	24.08748472
7/6/2018	20.95197751	47.58800796	24.09642956
7/7/2018	20.64895923	47.52870726	24.42949854
7/8/2018	20.31288883	47.52776993	24.75569031
7/9/2018	20.22961692	47.57670574	24.81861454

Figure 18. EIA API Data Excerpt

Generation to Emissions Calculations

One of our customer's main goals was to have a visual displaying the real time estimate of emissions based on the current generation in megawatts for the main fuel types. After some discussion on how best to create these values, it was decided that the best way to calculate this, for our project at least, was to look at the emissions per generation values for the last few years in order to keep the values based on the most current data.

The process of finding this information involved a few additional assumptions. First, since the EIA emissions data includes plants with negative and zero values for generation, including these in the calculation would cause some problems. These values, along with any "Partially Reporting Facilities," were removed from the data to be summed. The values for the selected years were calculated using Excel pivot tables and are shown below.



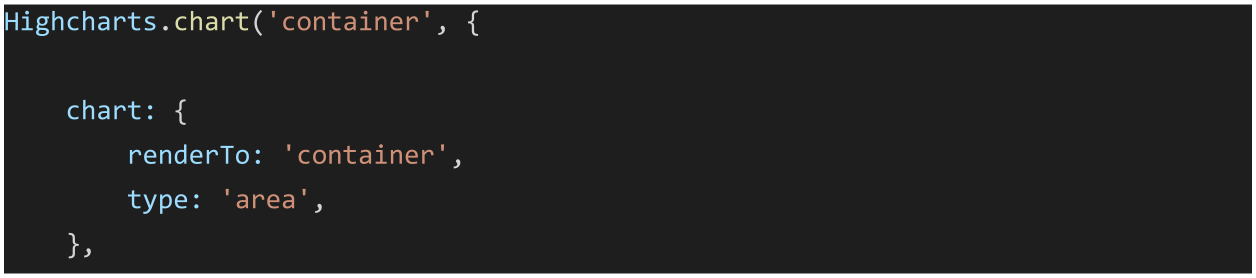
Row Labels	Average of tons/MW	Average of tons/kW	Average of tons/kWh
COAL	368848.4555	368.8484555	0.042105988
GAS	154400.4787	154.4004787	0.017625625
MSW	136048.0754	136.0480754	0.015530602
PET	23767.67147	23.76767147	0.002713205
Grand Total	**139970.4587**	**139.9704587**	**0.015978363**

Figure 19. Emissions per Generation Sums

Graph Creation

In order to create the graphing dashboard, the Highcharts library was used. This library was used due to MISO's desire to explore new industry leading graphing frameworks in order to aid enterprise modernization efforts. To construct the Highcharts dashboard, charts are first created in unique javascript files, as detailed below.

First, the chart object is initialized to use a specific HTML container object as defined in the full HTML dashboard.



```
Highcharts.chart('container', {

    chart: {
        renderTo: 'container',
        type: 'area',
    },
```

Figure 20. Highcharts Container Initialization

Next, the major graph series and x-axis data are added using the real time data importing functions detailed in an earlier segment of this report.

```
series: [{
        type: 'line',
        name: 'Coal',
        data: filterSeriesData(todayData, 'Coal')
    },
    {
        type: 'line',
        name: 'Natural Gas',
        data: filterSeriesData(todayData, 'Natural Gas')
    },
    {
        type: 'line',
        name: 'Nuclear',
        data: filterSeriesData(todayData, 'Nuclear')
    },
```

Figure 21. Highcharts Series Options

To complete the individual charts, different customization options are applied to refine the final visual product. In the code excerpt below, the y axis of the graph is customized to change display text and colors, as well as the position of the graph legend.

```
yAxis: {
        min: 0,
        title: {
            text: 'Electricity Generated (MW)'
        },
        plotLines: [{
            value: 0,
            width: 1,
            color: '#808080'
        }],
    },
    tooltip: {
        valueSuffix: 'MW'
    },
    legend: {
        layout: 'horizontal',
        align: 'center',
        verticalAlign: 'bottom',
        borderWidth: 0
    },
```

Figure 22. Highcharts Y-Axis Properties

A similar process was used to create all of the charts shown in the final dashboard, where different data sources were used within a common graphing framework and strategy. For graphs using historical data, the graph series were defined in the output CSV files from the historical data importing process. Using the Highcharts functionality designed for CSV files, the historical graphs were created using the approach below.

```
data: {
      csvURL: './co2_fuel_emissions_only_summed_outputCSV',
    },
```

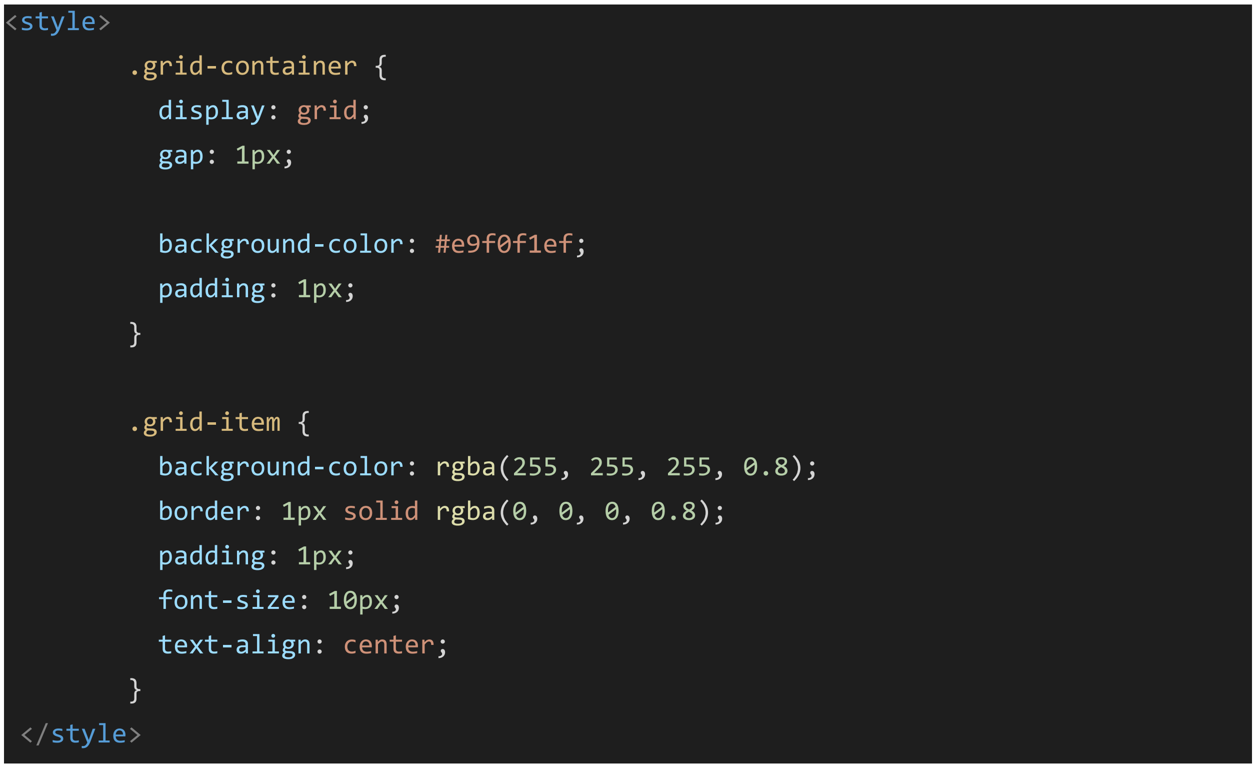
Figure 23. Highcharts CSV to URL Function

The CSV file specified lies locally in the directory of the JavaScript file used to create the respective graph.

Dashboard Creation

Using the populated JavaScript files detailed in the previous section, a final user visible dashboard can be created. This dashboard is created in a single HTML file that creates graphs upon the opening of the web page and details the visual style used in the dashboard.

To create the visual style and layout of the dashboard, a CSS grid object was used to maintain graph readability in a variety of resized conditions. The CSS used to create the grid object is below:



```
<style>
        .grid-container {
          display: grid;
          gap: 1px;

          background-color: #e9f0f1ef;
          padding: 1px;
        }

        .grid-item {
          background-color: rgba(255, 255, 255, 0.8);
          border: 1px solid rgba(0, 0, 0, 0.8);
          padding: 1px;
          font-size: 10px;
          text-align: center;
        }
 </style>
```

Figure 24. Dashboard CSS

The object above specifies the color, row size, and resizing behavior of the graphing dashboard to maintain a seamless and professional looking dashboard. Next, the grid holding all of the Highcharts graphs is created in the body tag of the HTML file.

```
<div class="grid-container">

        <script type="module" src="graphSkeleton.js"></script>
        </script>
        <div class="grid-item" id="container">1</div>
            <figure   class="highcharts-figure"></figure>
            <p class="highcharts-description">
        </figure>
```

Figure 25. HTML/CSS Grid

Using the created grid container, all of the Highcharts graphs are inserted into a figure inside an individual grid object as specified with a <div> tag. Each JavaScript file that contains a graph is assigned to a specific container object as created in the beginning argument inside each Highcharts file. The code block containing the <script>, <div>, and <figure> tags is repeated for each graph.

Using our grid layout, the graphs are put into a single column layout for maximum readability, as seen below.

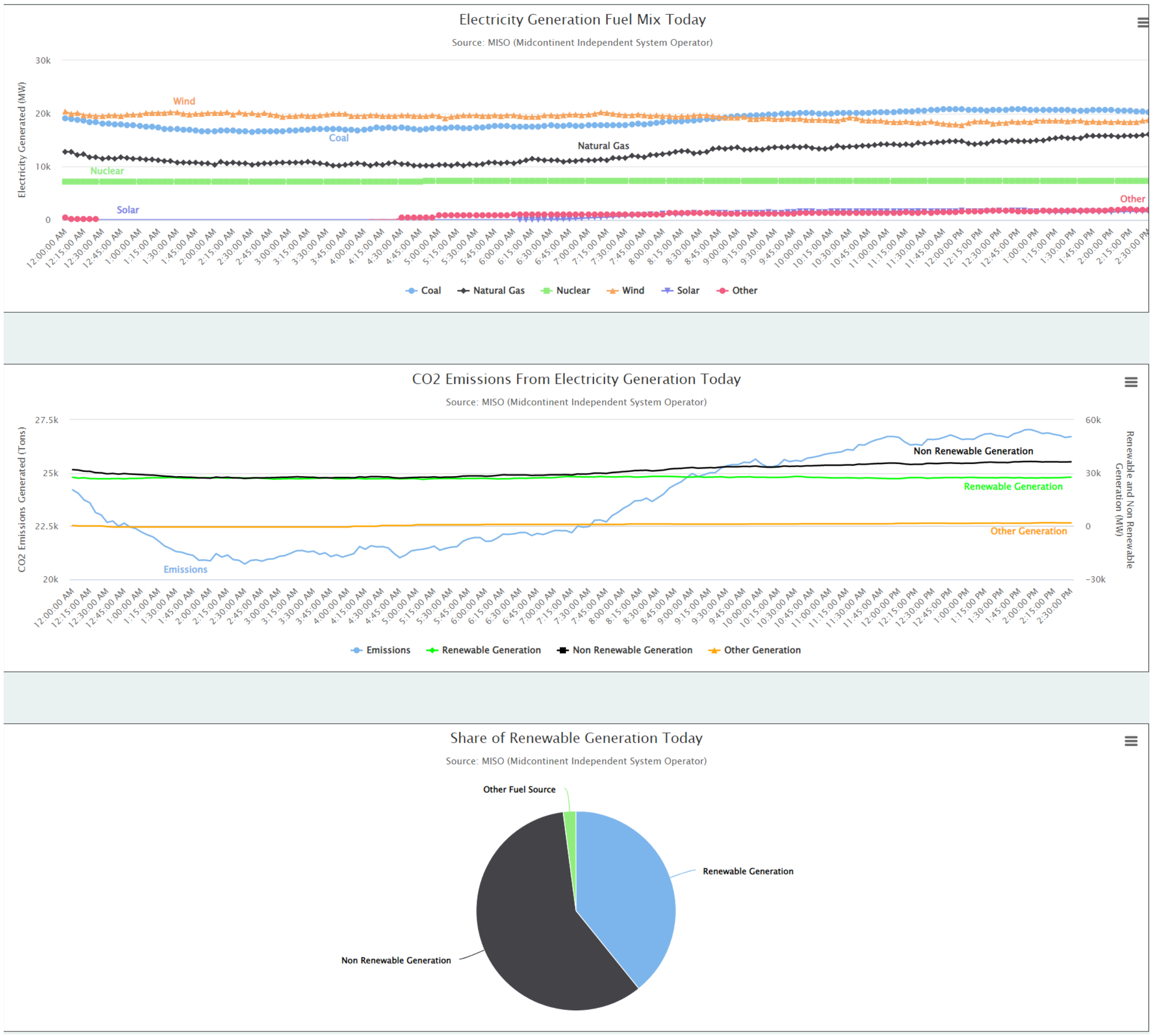


Figure 26. Final Dashboard Excerpt

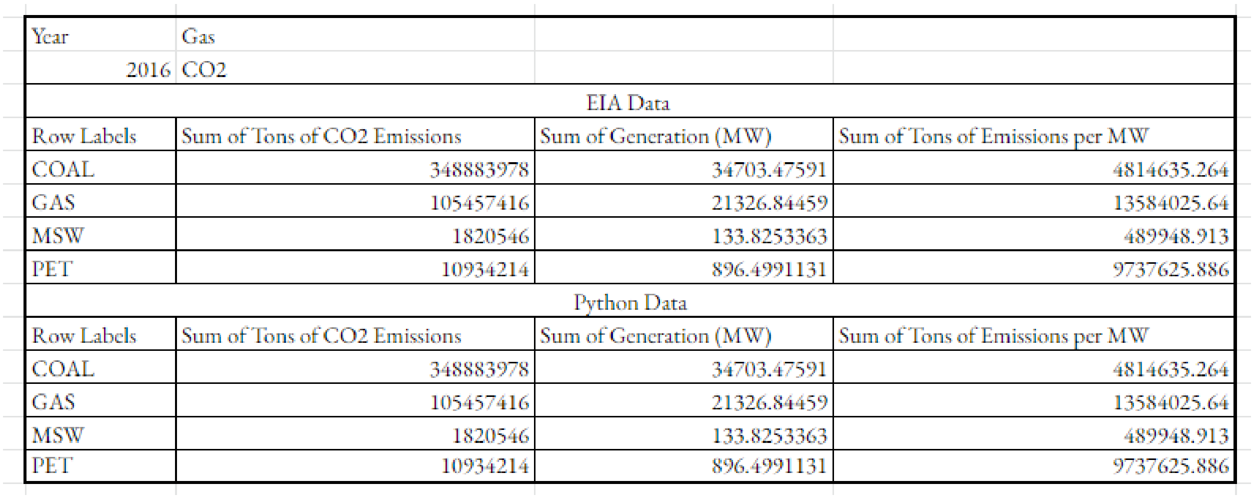
Results

Testing Process - Historical Data Verification

The data from our historical data scraper (HDS) was exhaustively verified in Excel for the year 2016; this year was selected both because the EIA began adding the

"Balancing Authority" (also referred to as "ISO") to the emissions files this year, allowing easy filtration for MISO operated plants, and because this was the year that first showed the error in the original backend database creation.

The data verification began by manually downloading the emissions data from the EIA website and cleaning the file up to remove those plants not operated by MISO, those labeled as "partially reporting facilities," and those with net generation less than or equal to zero. Pivot tables were made for all three gasses so the data could be easily filtered and manipulated. Since the EIA does not include the ISO's regional information for the plants, the verification sums used were for each type of fuel in the "Aggregated Fuel Type" column of the EIA data. These values were compared with the output from the HDS, which matched the EIA data. The results for the CO2 emissions are shown below.



Year	Gas		
2016	CO2		
	EIA Data		
Row Labels	Sum of Tons of CO2 Emissions	Sum of Generation (MW)	Sum of Tons of Emissions per MW
COAL	348883978	34703.47591	4814635.264
GAS	105457416	21326.84459	13584025.64
MSW	1820546	133.8253363	489948.913
PET	10934214	896.4991131	9737625.886
	Python Data		
Row Labels	Sum of Tons of CO2 Emissions	Sum of Generation (MW)	Sum of Tons of Emissions per MW
COAL	348883978	34703.47591	4814635.264
GAS	105457416	21326.84459	13584025.64
MSW	1820546	133.8253363	489948.913
PET	10934214	896.4991131	9737625.886

Figure 27. EIA 860 Data Verification

Testing Process - Dashboard Interface

Using our final dashboard, our testing process consisted of major efforts on both testing the frontend interactivity and the backend data verification. In order to test the frontend interactivity, each of the Highcharts user interactivity functions was explored.

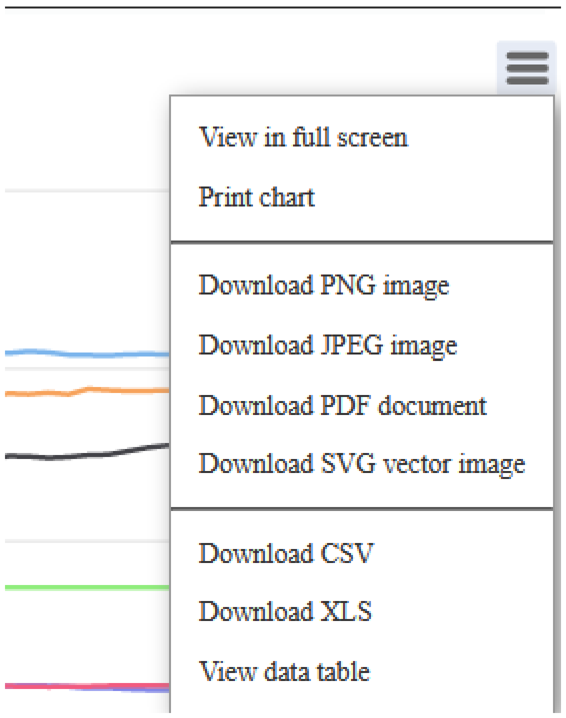


Figure 28. Highcharts Interactivity Pane

The main testing done at this end was to compare the downloaded CSV file to the file populating the series data in the graph. If the data in the CSV file downloaded from the graph is equivalent to the file used to create the graph, the graph creation code could be validated. This process of comparing the two CSV files was repeated for all of the graphs in our dashboard and was found to be completed successfully for all graphs.

Related Products (CAISO)

Our dashboard is similar to the dashboard created by California ISO to showcase their emissions data but differs in a few key areas. The largest difference between our dashboard and California ISO's is that our dashboard has no information on energy demand, as this is not directly captured in public data sources. Additionally, California ISO has access to monthly data about CO2 emissions, which we do not have access to. However, the bulk of the content and its delivery is common between our dashboard and California ISO's. Both have information about historical and real time emissions, electricity generation, and an attempt to convey this information in a manner that is appropriate for the general public as well as interested parties in the industry.

Results in Context of Related Literature

As we designed the webpage to be based around the California ISO's emission page, we were able to come to a similar style of layout and data for our own page. In this regard, we were able to achieve our goal of creating a product on par with other related pages that already exist in the industry while also receiving praise from our client company, MISO.

Project Value

The value of the project comes from the ability to translate data into information. In today's world, there is a push for energy companies to operate as green as possible. Researchers employed by MISO area utilities and outside institutions will find it very simple to analyze various trends and impact emissions of the MISO region, allowing them to study new generation projects and technologies to effectively meet emissions targets set by MISO customers and all levels of government. Customers of MISO utilities, including utility companies and individual end-users, can view this dashboard to see the emissions impacts of their regular activities and look to see how they can decrease the impact of their energy usage by shifting the time of day, season, or physical location of their energy usage.

From the existing dashboard, users can see that CO2 emissions from generation in the MISO region have declined precipitously from 2014, with the emissions decrease led by a decrease in coal emissions, as shown below.

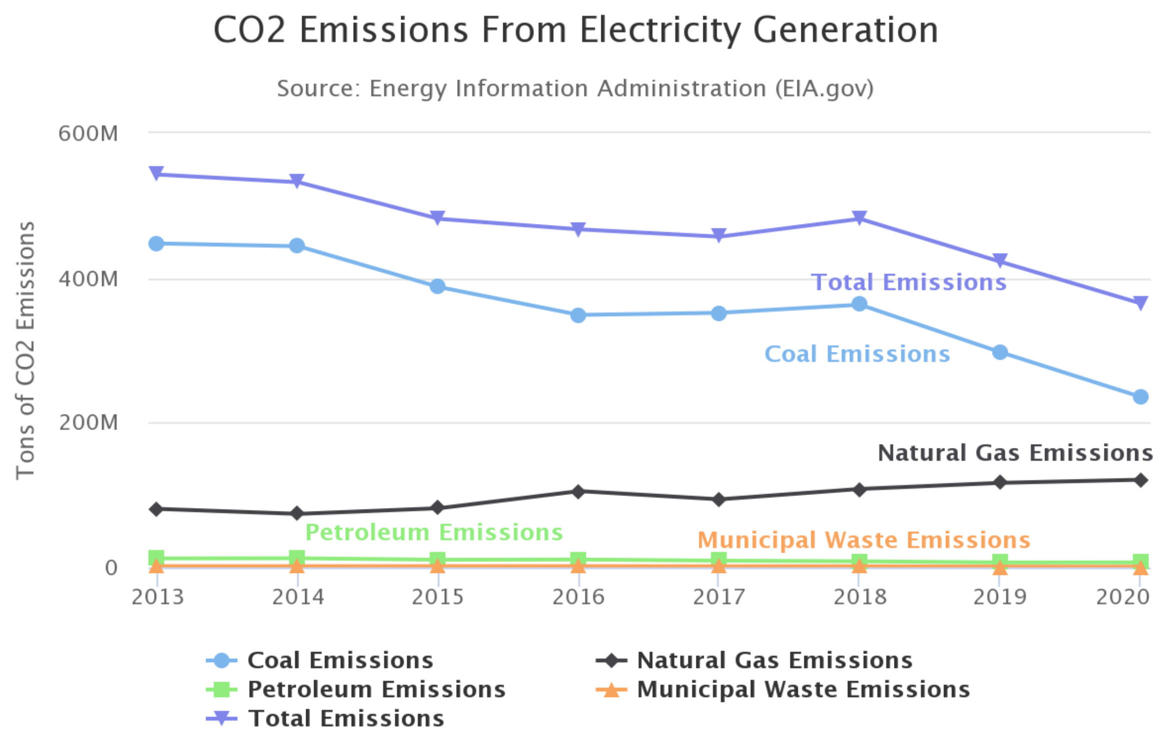


Figure 29. MISO Network Emissions Since 2013

Breaking down the above graph by region, the trend in declining emissions is seen throughout all regions, but the central region has seen the greatest decline, over 30%, in emissions over this period.

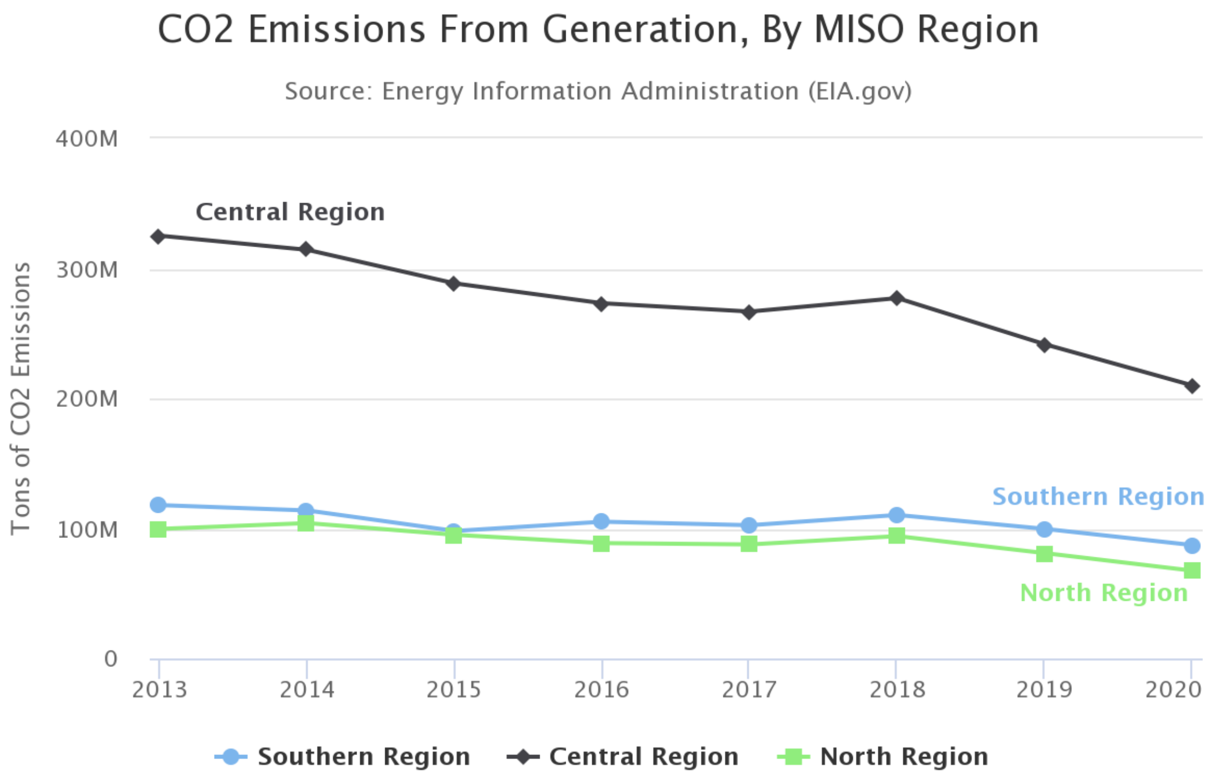


Figure 30. Regional Emissions By Year

Digging deeper into the declines, we can see that the summer months and hard winters rely on coal to generate the most electricity, while renewable and natural gas generation contributes a higher percentage of generation in other parts of the year. The recent trend in the amount of coal used in peak summer months is a declining one, especially seen in the transition from 2019 to 2020. To maximize grid decarbonization, developing renewable energy technologies to better withstand summer temperatures and storms may be prudent.

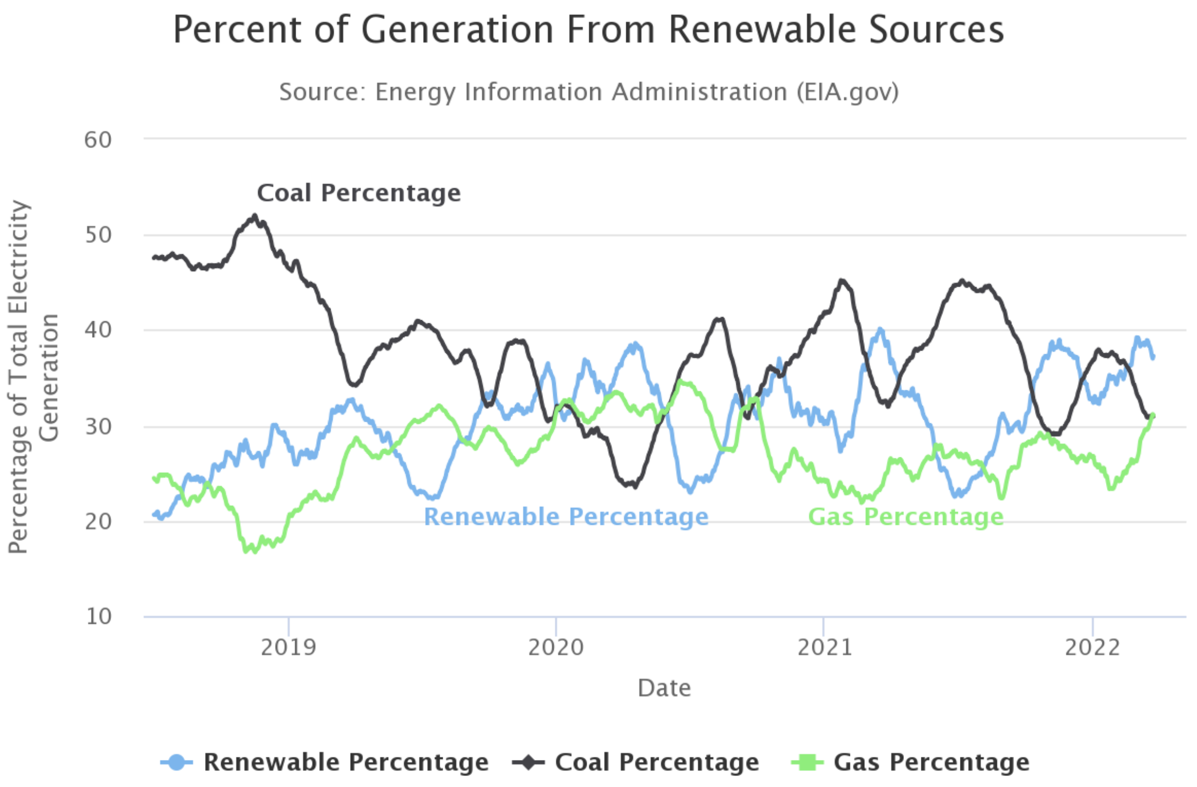


Figure 31. Historical Generation Source Percentages

Additionally, the high demand for electricity in the winter months should cause planners to better understand how to plan for broader electrification in the residential and commercial heating sector. Further advances in the prevalence of electric heat will drive new peaks in electricity demand that are not met by the current profile of renewable generation.

Taking the graph above and the MISO network emissions graph, the clear correlation between declining emissions and elevated generation of renewable electricity can be shown, especially when comparing the ever elevated yearly troughs in the amount of renewable electricity generated.

Appendix I - Operation Manual

Historical Data Scraper

Running

The historical data scraper is written in Python and is intended to be used annually (or possibly more often, depending on when the EIA releases the information to be scraped) to gather the yearly EIA 860 and EIA Emissions files from the EIA website. The source contains two files: <main> and <functions>. Once the files are downloaded, the code is currently initiated by running the <main.py> script. The user may also require elevated privileges to install required dependencies. The data scraper code is intended to require no modification unless the EIA moves the data files, drastically changes the output format, or the user requires the data storage paths to change. *Note: The software could potentially be ported to run on Mac or Linux operating systems by changing the local storage paths, but this has not been verified.*

Editing Storage Locations

To edit the storage locations and output file names generated by the backend code, the <main.py> can be opened in a text editor or development environment. The base directories should not be modified; however, the development and final output folders can be renamed as the user prefers. Shown below are the current storage locations, which were purposely set to be inside the C:\Users\Public\Documents folder for easing portability.

```
#base directories
dir_00_base_directory = Path("C:/Users/Public/Documents/MISO_TEMP")# base directory to store local data
dir_01_base_temp = Path(str(dir_00_base_directory) + "/temp")#base temporary file directory
dir_02_base_output = Path(str(dir_00_base_directory) + "/OUTPUT_FILES")# base directory for output files
#individual dev directories
dir_temp01 = Path(str(dir_01_base_temp) + "/01_downloaded_eia_860_data_created_by_F02")#downloaded EIA860 data directory
dir_temp02 = Path(str(dir_01_base_temp) + "/02_downloaded_eia_em_data_created_by_F04")#downloaded EIA emissions data directory created by F04 "download_eia_emissions"
dir_temp03 = Path(str(dir_01_base_temp) + "/03_extracted_eia_em_data_created_by_F05")#gas specific extracted EIA emissions data director
dir_temp04= Path(str(dir_01_base_temp) + "/04_extracted_MISO_ONLY_emissions_data")#gas specific extracted MISO ONLY EIA emissions data directory
dir_temp05= Path(str(dir_01_base_temp) + "/05_extracted_MISO_ONLY_emissions_data_original_format")#gas specific extracted MISO ONLY EIA emissions data directory original format
dir_temp06= Path(str(dir_01_base_temp) + "/06_final_output_original")#ORIGINAL OUTPUT fILES Directory
dir_temp07 = Path(str(dir_01_base_temp) + "/07_yearly_location_data")# location data file directory
dir_temp08 = Path(str(dir_01_base_temp) + "/08_yearly_emissions_with_location")
#final output directory
dir_03_final_output = Path(str(dir_02_base_output) + "/FINAL_OUTPUT_01")
```

Figure 32. EIA 860 Export Directories

Output Files

The historical data scraper outputs files of two main types: final output CSV files (used by the Dashboard) and temp files which are used by the script and are left in place in case the user wants to use them to verify output data or to debug any problems that arise. The temp files are located in the C:\Users\Public\Documents\temp directory and are separated into folders based on the function that created them (with some minor exceptions). The user can choose to delete these files after running the script.

Dashboard Operation

Operation of the dashboard will require a basic understanding of web page navigation along with a greater understanding of how to read graphs. Since this dashboard will be used as a tool for MISO stakeholders, an understanding of what these graphs represent will already be assumed. As for how the graphs can be interacted with, we have implemented ways for the specific graphs to be filtered to only show what the user wishes to see. Options to download the raw data the graphs represent will also be an option for users to be able to grab that data and use it for their own purposes. The data on these graphs will be used to make important business decisions as to what areas need more attention, what areas to make new generators, or run various projects to lower emissions from various sources.

In order to create a new dashboard instance, the HTML file named "containerCreator.HTMl," must be launched. This file will run all of the javascript files containing graphs, creating a full dashboard with all graphs visible to the end-user. At this point, the user can interact with the dashboard using the functionalities available in Highcharts, including graph highlighting and the use of various tooltips, as demonstrated below.

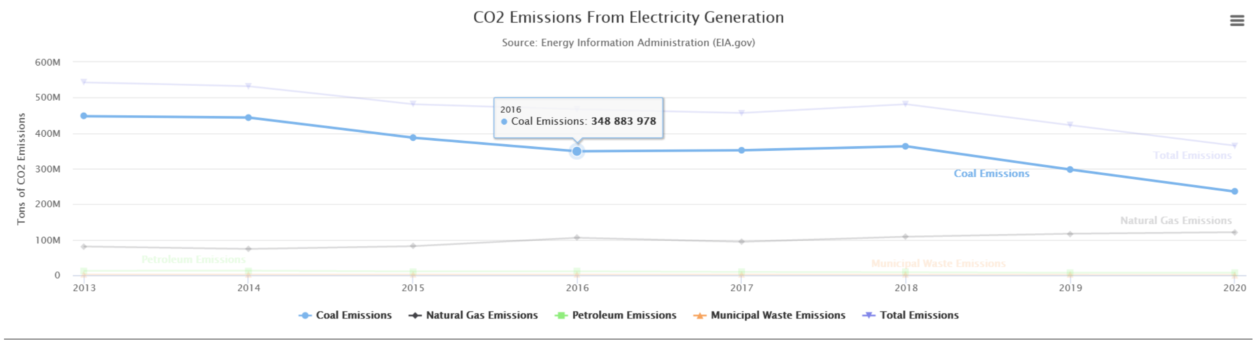


Figure 33. Highlighted Graph

Appendix II - Alternative/other initial versions of the Design

Alternative Design: Implementation of Highmaps

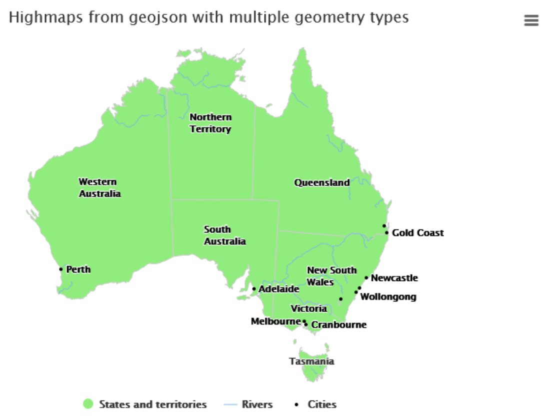


Figure 34: Highmaps Example

Highmaps, an extension of Highcharts Javascript API, was considered for some time to show the MISO region like Figure 7. Although complications with MISO shapefiles and time constraints have made us stop the development of this interactive map in order to create a stronger final deliverable for MISO. Further development into this area would continue with researching more on the implementation of GeoJSON files and how they interact with highmaps or the possibility of using GeoSet data from the EIA website per series that we can capture. More research would have to go into the implementation of the GeoSet data.

Initial version of our Design

Earliest implementation of graphs:

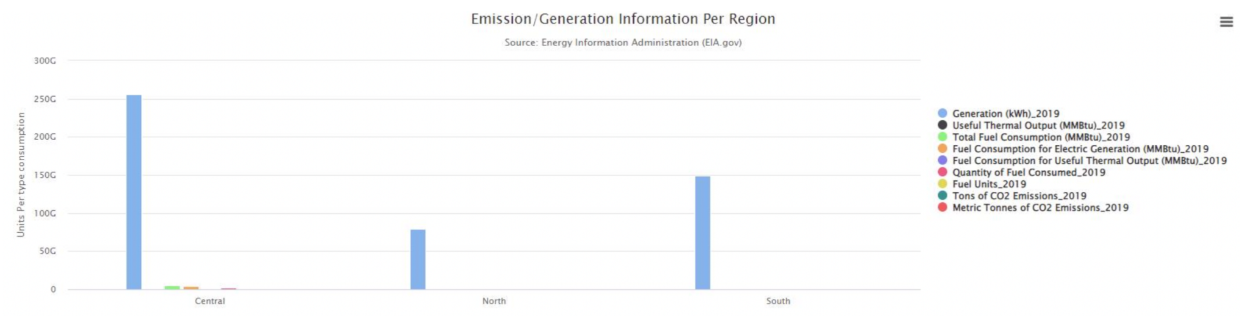
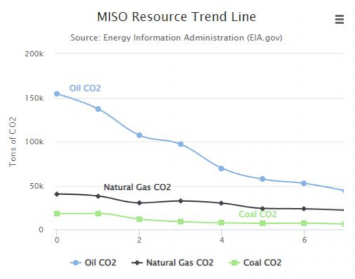
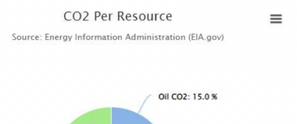
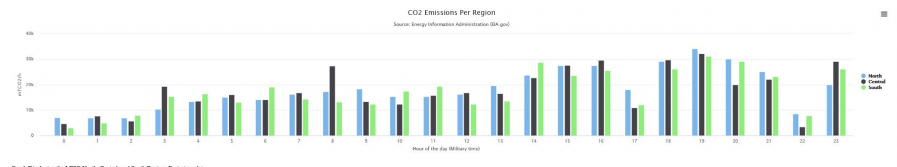


Figure 35: Data graph showing North, Central, South Regions

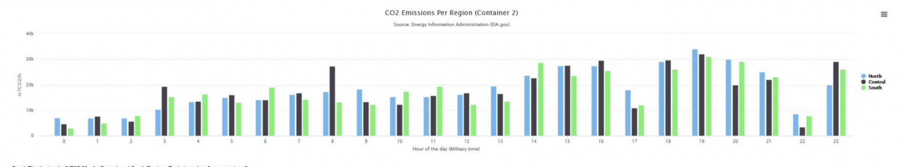
Figure 8 is one of our earliest designs of a graph in Highcharts. We can see here that we are just getting started on how to grab data and put it into the chart for visual representation and getting used to what data to show on what graph as generation is so high that other graph data cannot be seen.







Graph Displaying the MISO North, Central, and South Regions Emissions data



Graph Displaying the MISO North, Central, and South Regions Emissions data from container 2

Figure 36: Better graphs with raw data

Figure 9 shows our progress a little over the halfway semester point. We can see that our implementation with Highcharts is significantly better with very readable graphs and data, and the charts are starting to look more like a dashboard. This implementation is done with placeholder data, and plans to implement real time data were already underway.

Next, we worked on creating a dashboard layout for our final product. The initial design for the dashboard layout included an attempt at a two column layout, but this was found to be impractical and difficult to reliably implement due to the dynamic dimensions of grid elements in our dashboards and the dynamic dimensions of our Highcharts graph.

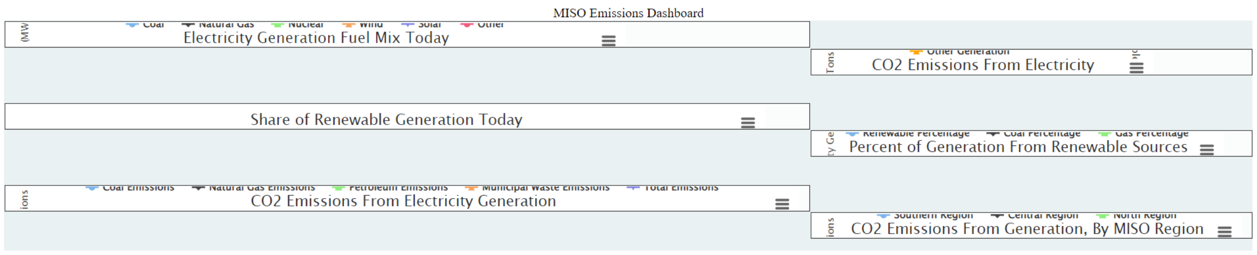

MISO Emissions Dashboard
Electricity Generation Fuel Mix Today
CO2 Emissions From Electricity
Share of Renewable Generation Today
Percent of Generation From Renewable Sources
CO2 Emissions From Electricity Generation
CO2 Emissions From Generation, By MISO Region

Figure 37. Attempted Two Column Graph

After trying to create a two column graph using our CSS grid, the team decided to use a single column layout to minimize the dashboard complexity while preserving a quality user experience with dynamically sized graphs.

Appendix III - Other Considerations

Assumptions Made for Dashboard

- For parsing EIA form 860 data, facilities that reported less than zero generation were assumed to have zero generation. This will exclude facilities like coal mines attached to coal power plants, as seen in a few instances in Illinois. Additionally, this may exclude pumped storage hydroelectric power plants depending on electricity demand and rainfall patterns.
- MISO Approved - States that had multiple ISOs in their respective state were assumed to be completely MISO controlled. (Missouri, for example, is a state that has multiple ISOs operating within their state). This assumption was applied as EIA does not release data at the scale of a MISO region, but state-level data is available.
- Emissions and Generation data being reported by EIA is accurate. Some possible data errors have been observed with the different scales of input data from EIA, as utility scale data is sometimes combined with industrial or microgeneration data in certain form 860 or API entries.

Appendix IV - Code

Git repository of final project code

Appendix VI - Figures

Appendix VII - Bibliography

"Basic line," *Basic line | Highcharts.com*. [Online]. Available: https://www.highcharts.com/demo/line-basic. [Accessed: 24-Apr-2022].

"IEEE SA - IEEE Standard for Software User Documentation," *SA Main Site*. [Online]. Available: https://standards.ieee.org/ieee/1063/1554/. [Accessed: 24-Apr-2022].

"IEEE SA - IEEE Standard for application and management of the Systems Engineering Process," *SA Main Site*. [Online]. Available: https://standards.ieee.org/ieee/1220/3372/. [Accessed: 24-Apr-2022].

“IEEE SA - ISO/IEC/IEEE International Standard - Systems and software engineering - life cycle processes - project management,” *SA Main Site*. [Online]. Available: https://standards.ieee.org/ieee/16326/6769/. [Accessed: 24-Apr-2022].

“IEEE SA - ISO/IEC/IEEE International Standard - Systems and software engineering - engineering and management of websites for systems, software, and services information,” *SA Main Site*. [Online]. Available: https://standards.ieee.org/ieee/23026/5537/. [Accessed: 24-Apr-2022].

“MISO Real Time Displays,” *Operations displays*. [Online]. Available: https://www.misoenergy.org/markets-and-operations/real-time--market-data/operations-displays/. [Accessed: 24-Apr-2022].

“Todays Emissions,” *California iso - emissions, today's outlook*. [Online]. Available: https://www.caiso.com/TodaysOutlook/Pages/emissions.html. [Accessed: 14-Oct-2021].

“Real Time Dashboard,” *NYISO*. [Online]. Available: https://www.nyiso.com/real-time-dashboard. [Accessed: 14-Oct-2021].

“Real-time maps and charts,” *ISO New England - Real-Time Maps and Charts*. [Online]. Available: https://www.iso-ne.com/isoexpress/web/charts. [Accessed: 14-Oct-2021].

Southwest Power Pool, “Generation Mix,” *Generation mix*. [Online]. Available: https://marketplace.spp.org/pages/generation-mix. [Accessed: 14-Oct-2021].

“U.S. Energy Information Administration - EIA - independent statistics and analysis,” EIA. [Online]. Available: https://www.eia.gov/opendata/commands.php. [Accessed: 23-Apr-2022].