Reproduction Information

Cost uncertainties and ecological impacts drive tradeoffs between electrical system decarbonization pathways in New England, U.S.A.

Amir M. Gazar[1,2], Chloe Jackson[3], Georgia Mavrommati[3], Rich B. Howarth[4], and Ryan S.D. Calder[1,2,5,*]

[1]Department of Population Health Sciences, Virginia Tech, Blacksburg, VA, 24061, USA
[2]Global Change Center, Virginia Tech, Blacksburg, VA, 24061, USA
[3]School for the Environment, University of Massachusetts Boston, Boston, MA 02125, USA
[4]Environmental Program, Dartmouth College, Hanover, NH, 03755, USA
[5]Department of Civil and Environmental Engineering, Duke University, Durham, NC, 27708, USA
[*]Corresponding author: rsdc@vt.edu

Contents

1 Configuration and Setup

This document presents a comprehensive guide outlining the sequential process required to execute and replicate the capacity expansion model and total cost calculation presented in our article. Additionally, we provide a conceptual overview of code functionality in Section 2.2.

1.1 Hardware and Operating System

We utilized a MacBook Pro with an Apple M1 Pro chip, featuring an 8-core CPU and 16 GB of memory. The startup disk is the Macintosh HD. The system operates on macOS 14.5 `Sonoma`. To run the generation expansion model script, we utilized Virginia Tech's Advanced Research Computing (ARC) resources. We used the following specifications for this source for each simulation.

```
#SBATCH --partition=normal_q
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=50
#SBATCH --cpus-per-task=1
#SBATCH --time=24:00:00
```

1.2 Software Versions and Dependencies

We used R version 4.4.2 `(2024-10-31)` for the x86_64-apple-darwin20 platform. Furthermore, we used the RStudio environment, version 2024.09.1 Build 394 to execute the code. Table 1 shows theackages and versions utilized in this code.

Table 1: Package Version

Package	Version
data.table	1.14.8
lubridate	1.9.3
httr	1.4.6
jsonlite	1.8.7
htmltools	0.5.5
ggplot2	3.4.2
scales	1.2.1
zoo	1.8.12
gridExtra	2.3
dplyr	1.1.2

1.3 Install Time

Installation of RStudio and R depends on the specific hardware and operating system in use. This also holds true for the packages employed within the code. However, the packages used in this script typically take a few minutes to install.

1.4 Run Time

The run time to execute the generation expansion model script in ARC at Virginia Tech is as follows:

- Full demand curve (9,495 days): 15 minutes for 1 simulation for each decarbonization pathway.

Depending on the size of the data set and the hardware configuration, this duration may vary. For all other scripts, the run time was a few minutes on our computer.

2 Conceptual Overview and Code Availability

2.1 Code availability

The codes for this study are available on the GitHub repository via this link.

2.2 Conceptual Overview

The framework integrates key components across three stages: (1) Input Variables, which include decarbonization pathways (e.g., new offshore wind, transmission lines to/from Quebec, and new small modular reactors) and external data sources (e.g., ISO-NE demand, EIA and EPA datasets, and IEA resources); (2) the Generation Expansion Model, which computes demand curves, clean and fossil fuel generation, imports, energy storage, and unmet demand at the Regional Transmission Organization (RTO) level; and (3) Total Costs, which are determined through cost modules covering CAPEX, FOM, VOM, emissions, fuel, imports, and unmet demand penalties to estimate total social costs.

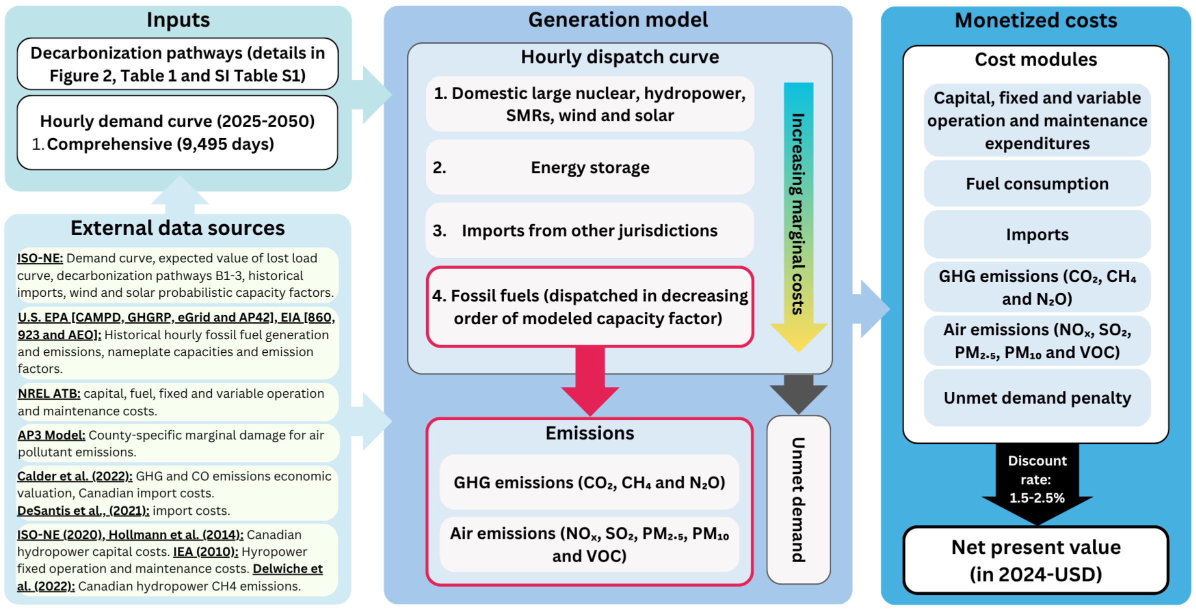
Inputs
Decarbonization pathways (details in Figure 2, Table 1 and SI Table S1)
Hourly demand curve (2025-2050)
1. Comprehensive (9,495 days)
External data sources
ISO-NE: Demand curve, expected value of lost load curve, decarbonization pathways B1-3, historical imports, wind and solar probabilistic capacity factors.
U.S. EPA [CAMPD, GHGRP, eGrid and AP42], EIA [860, 923 and AEO]: Historical hourly fossil fuel generation and emissions, nameplate capacities and emission factors.
NREL ATB: capital, fuel, fixed and variable operation and maintenance costs.
AP3 Model: County-specific marginal damage for air pollutant emissions.
Calder et al. (2022): GHG and CO emissions economic valuation, Canadian import costs.
DeSantis et al., (2021): import costs.
ISO-NE (2020), Hollmann et al. (2014): Canadian hydropower capital costs. IEA (2010): Hyropower fixed operation and maintenance costs. Delwiche et al. (2022): Canadian hydropower CH4 emissions.
Generation model
Hourly dispatch curve
1. Domestic large nuclear, hydropower, SMRs, wind and solar
2. Energy storage
3. Imports from other jurisdictions
4. Fossil fuels (dispatched in decreasing order of modeled capacity factor)
Increasing marginal costs
Emissions
GHG emissions (CO_2, CH_4 and N_2O)
Air emissions (NO_x, SO_2, $PM_{2.5}$, PM_{10} and VOC)
Unmet demand
Monetized costs
Cost modules
Capital, fixed and variable operation and maintenance expenditures
Fuel consumption
Imports
GHG emissions (CO_2, CH_4 and N_2O)
Air emissions (NO_x, SO_2, $PM_{2.5}$, PM_{10} and VOC)
Unmet demand penalty
Discount rate: 1.5-2.5%
Net present value (in 2024-USD)

Figure 1: Conceptual overview of the modeling framework.

3 Decarbonization Pathways

3.1 Decarbonization Pathways Data Processing

The decarbonization pathways are analyzed using a custom R script, `Hourly_Installed_Capacity.R`, which processes data from an Excel file. The key steps are as follows:

1. **Loading Libraries**: The script utilizes the `readxl`, `data.table`, and `lubridate` libraries for reading Excel files, managing data efficiently, and handling date-time information, respectively.

2. **Reading Decarbonization Data**:

   - The Excel file, `Decarbonization_Pathways.xlsx`, contains multiple sheets, each representing a decarbonization pathway.
   - All sheets are read and combined into a unified dataset.

3. **Adding Metadata**: A new column `Pathway` is added to the data from each sheet, identifying the respective pathway.

4. **Combining Data**: The data from all sheets are combined into a single `data.table` for further analysis.

5. **Extracting Year Range**: The script identifies the range of years in the dataset (`start_year` and `end_year`) for contextualizing analysis.

Important Functions and Steps:

- Loading Excel sheet names:

Listing 1: Loading Excel Sheet Names

```
    sheet_names <- excel_sheets(file_path)
```

- Adding pathway metadata to each sheet:

Listing 2: Adding Pathway Metadata

```
    sheet_data[, Pathway := sheet]
```

- Combining all data:

Listing 3: Combining All Sheets

```
    combined_data <- rbindlist(data_tables, fill = TRUE)
```

- Extracting year range:

Listing 4: Extracting Year Range

```
    years <- unique(as.numeric(combined_data$Year))
    start_year <- min(years, na.rm = TRUE)
    end_year <- max(years, na.rm = TRUE)
```

This script prepares the decarbonization pathways data for subsequent analysis, ensuring all relevant information is integrated and organized.

4 Generation Expansion Model

4.1 Wind and Solar Hourly Capacity Factors

4.1.1 Overview

The `Wind_Solar_Hourly_CF.R` script processes hourly wind and solar generation data to calculate capacity factors. The script performs several preprocessing steps to standardize the data and align it with the time zones and calendar conventions used in the model.

4.1.2 Detailed Explanation

1. **Loading Data**: Hourly generation data is read from CSV files, and the required columns are selected for analysis.

2. **Time Zone Adjustment**: Timestamps are converted from UTC to Eastern Standard Time (EST), aligning with local conditions.

3. **Handling Leap Years**: Leap year adjustments ensure the data is consistent, avoiding errors in models that depend on the length of the year.

4.1.3 Key Code

Listing 5: Adjusting Timestamps and Handling Leap Years

```
data_processing_func <- function(file, columns_to_keep) {
  data <- readr::read_csv(file, show_col_types = FALSE)
  data <- data %>% dplyr::select(all_of(columns_to_keep))
```

```r
  data$Date <- as.POSIXct(data$Date, format = "%m/%d/%Y %I:%M:%S %p", tz = "UTC")

  # Convert UTC to EST
  data$Date <- data$Date - hours(5)

  # Add DayLabel and Hour columns
  data <- data %>%
    mutate(DayLabel = yday(Date), Hour = hour(Date) + 1)

  # Adjust DayLabel for non-leap years
  if (!is_leap_year(year(data$Date[365]))) {
    data <- data %>% mutate(DayLabel = ifelse(DayLabel == 366, 365, DayLabel))
  }
}
```

—

4.2 Small Modular Reactors (SMR)

4.2.1 Overview

The `SMR_CF.R` script models the operational characteristics of small modular reactors. These include:

- Capacity factor (90%)
- Ramp rate and minimum power output
- Core fuel characteristics, including enrichment and burnup

4.2.2 Key Code

Listing 6: Defining SMR Specifications

```r
data <- data.frame(
  Label = "SMR300",
  Category = "SMR",
  Nameplate_Capacity_MW = 300,
  CF = 0.9,
  Ramp = 1,
  Minimum_Power_Output_MW = 60,
  Enrichment = "4.95%",
  Fuel = "UO2",
  Reactor_Lifetime = 60
)
```

—

4.3 Fossil Fuels: Facility Data and Emissions

4.3.1 Fossil Fuel Facility Data

The `Fossil_Fuels_Facilities_Data.R` script processes historical fossil fuel facility data for New England. This data is critical for understanding the existing capabilities and limitations of the regional fossil fuel infrastructure.

Detailed Explanation:

1. **Loading Data**: Facility data for all U.S. states is loaded. The dataset includes information on facility location, capacity, operating status, and ramp rates.
2. **Filtering for Active Facilities**: Facilities are filtered to include only those located in New England and those that are operational (i.e., not retired).
3. **Ramp Rate Conversion**: Ramping times (e.g., ”10M”, ”1H”) are converted into numeric values representing hours. This standardization ensures compatibility with the model.

Key Code Excerpts:

Listing 7: Ramp Rate Conversion Function

```
convert_ramp_to_numeric <- function(ramp) {
  if (is.na(ramp)) {
    return(24) # Default to 24 hours for missing values
  } else if (ramp == "10M") {
    return(0.1666667)  # 10 minutes
  } else if (ramp == "1H") {
    return(1)   # 1 hour
  } else if (ramp == "12H") {
    return(12)  # 12 hours
  } else if (ramp == "OVER") {
    return(24)  # Default for 'OVER'
  } else {
    return(24)
  }
}
```

Listing 8: Filtering Facilities for New England

```
# Filter data for active facilities in New England
new_england_states <- c("CT", "ME", "MA", "NH", "RI", "VT")
facilities_data_NE <- facilities_data[State %in% new_england_states]
facilities_data_NE <- facilities_data_NE[!grepl("retired", Operating_Status, ignore.case = ...
    TRUE)]
```

—

4.3.2 Fossil Fuel Generation Emissions

The `Fossil_Fuels_Gen_Emissions.R` script compiles hourly emissions data for fossil fuel facilities in New England. Emissions data is crucial for estimating environmental costs and compliance with emission regulations.

Detailed Explanation:

1. **Reading Emissions Data**: Hourly emissions data is read for each New England state.
2. **Combining Emissions Data**: Data from individual states is combined into a single dataset.
3. **Cross-Referencing with Facility Data**: Emissions data is filtered to include only active facilities, ensuring consistency with the facility dataset.

Key Code Excerpts:

Listing 9: Combining Emissions Data

```
combined_emissions_data <- rbindlist(all_emissions_data, fill = TRUE)

# Filter emissions data to match active facilities
active_facilities <- facilities_data_NE$Facility_Unit.ID
combined_emissions_data <- combined_emissions_data[Facility_Unit.ID %in% active_facilities]
```

—

4.4 New Fossil Fuel Facilities

4.4.1 Overview

The `Fossil_Fuels_New.R` script models the addition of new fossil fuel facilities to meet future energy demands. These facilities are phased in starting from 2033, with three units per facility and one facility added annually.

Detailed Explanation:

1. **Defining Facility Specifications**: Each facility is given a unique ID, and the year it comes online is specified.
2. **Expanding Facility Data**: Each facility is expanded into three units to represent its operational characteristics accurately.

Key Code Excerpts:

Listing 10: Defining New Fossil Fuel Facilities

```
# Define 18 facilities starting in 2033
new_facilities <- data.table(
  Facility_ID = paste0("NGCC", 3000:(3000 + num_facilities - 1)),
  Year_Online = start_year:(start_year + num_facilities - 1)
)

# Expand each facility into 3 units
new_facility_units <- new_facilities[, .(
  Unit_ID = 1:3,
  Facility_ID,
  Year_Online
)]
```

4.5 Imports

4.5.1 Overview

The `imports.R` script processes electricity imports data to calculate statistics and capacity factors for modeling electricity imports into the region. Key components of this script include:

- Loading daily imports data for the years 2011 to 2023 from an Excel file.
- Extracting the maximum capacity factors for transmission lines based on NREL ATB standards (set to 95%).
- Computing percentiles for electricity imports over the given time period.

4.5.2 Detailed Explanation

1. **Loading Libraries**: The script uses a range of libraries, including `dplyr`, `readxl`, `lubridate`, `data.table`, and `tidyverse` for data manipulation, reading Excel files, and date handling.
2. **Defining Time Range**: The imports data is analyzed for the period between January 1, 2011, and December 31, 2023.
3. **Loading Data**:
   - The script identifies and reads sheets from the Excel file corresponding to years between 2011 and 2023.
   - Data from each sheet is combined into a single dataset, appending a `Year` column for clarity.
4. **Percentile Calculations**: The `calculate_percentiles` function calculates the percentiles (from 1% to 99%) of a specified column in the dataset. This is useful for understanding the distribution of electricity imports.

4.5.3 Key Code

Defining Time Range and Max Capacity Factor The time range for the analysis and the maximum capacity factor for transmission lines (set to 95%) are defined as follows:

Listing 11: Defining Time Range and Max Capacity Factor

```
start_date <- as.Date("2011-01-01")
end_date <- as.Date("2023-12-31")
max_CF_transmission_lines <- 0.95 # According to NREL ATB
```

Loading Imports Data The imports data is loaded from an Excel file, extracting only the sheets corresponding to the years of interest (2011–2023):

Listing 12: Loading Electricity Imports Data

```r
file <- "daily_capacity_status.xlsx"
imports_data <- data.frame()
years <- as.character(2011:2023)
sheet_names <- excel_sheets(file)
sheet_names_of_interest <- sheet_names[sapply(sheet_names, function(name) name %in% years)]

for (sheet_name in sheet_names_of_interest) {
  sheet_data <- read_excel(file, sheet = sheet_name)
  sheet_data$Year <- sheet_name
  imports_data <- rbind(imports_data, sheet_data)
}
```

Calculating Percentiles The following function calculates percentiles for a specified column of the imports dataset:

Listing 13: Percentile Calculation Function

```r
calculate_percentiles <- function(data, column) {
  quantiles <- quantile(data[[column]], probs = seq(0.01, 0.99, by = 0.01), na.rm = TRUE)
  tibble(Percentile = seq(1, 99), Value = quantiles)
}
```

This function is applied to the imports data to understand the variability and trends in electricity imports. —

4.5.4 Insights and Results

The `imports.R` script prepares the data for modeling electricity imports, providing:

- A unified dataset of daily electricity imports from 2011 to 2023.
- Insights into the statistical distribution of imports through percentile calculations.
- Capacity factor assumptions for transmission infrastructure.

4.6 Demand Analysis

4.6.1 Overview

The script `demand.R` processes hourly demand data to ensure consistency and accuracy, especially for leap years. Its main functionalities include:

- Estimating missing February 29th demand values in leap years.
- Cleaning and standardizing hourly demand data for the energy model.

4.6.2 Leap Year Adjustment

The script checks for leap years and fills missing February 29th demand data by averaging February 28th and March 1st values for each hour. The following code performs this operation:

Listing 14: Filling Missing Leap Year Data

```r
fill_leap_year_data <- function(data) {
  for (year in unique(data$Year)) {
    if (year %% 4 == 0 && (year %% 100 != 0 || year %% 400 == 0)) {  # Check if leap year
      for (hour in 0:23) {
        feb_28_data <- data[Year == year & Month == 2 & Day == 28 & Hour == hour, Demand]
        mar_1_data <- data[Year == year & Month == 3 & Day == 1 & Hour == hour, Demand]

    if (length(feb_28_data) == 1 && length(mar_1_data) == 1) {
     leap_day_demand <- mean(c(feb_28_data, mar_1_data), na.rm = TRUE)
     data <- rbind(data, data.table(
      Year = year,
      Month = 2,
      Day = 29,
      Hour = hour,
      Demand = leap_day_demand
     ))
    }
   }
  }
 }
 return(data)
}
```

—

4.7 Randomization

4.7.1 Overview

The `randomization.R` script generates pseudo-random sequences for use in probabilistic modeling. These sequences are designed to represent variability in key parameters or inputs to the model. The script performs the following tasks:

- Defines parameters for the randomization process, including the number of sequences and the length of each sequence.
- Uses a custom function to generate pseudo-random sequences.
- Saves the generated sequences to a CSV file for integration into the modeling framework.

4.7.2 Detailed Explanation

1. **Loading Libraries**: The `randtoolbox` library is loaded, which provides tools for generating random and quasi-random numbers.
2. **Defining Parameters**:
   - `num_sequences`: The total number of sequences to generate (1,000,000).
   - `num_numbers`: The length of each sequence (250 numbers per sequence).

   These parameters define the scope and granularity of the randomization process.
3. **Generating Pseudo-Random Sequences**: A custom function, `generate_pseudo_random_sequences`, is defined to generate sequences of pseudo-random numbers. This function uses the `sample` function to create a sequence of integers between 1 and 99.
4. **Saving Random Sequences**: The generated sequences are saved to a CSV file for later use in the generation expansion model.

4.7.3 Key Code

Defining the Randomization Function The following function generates pseudo-random sequences:

Listing 15: Pseudo-Random Sequence Generator

```
generate_pseudo_random_sequences <- function(num_sequences, num_numbers) {
 random_sequences <- replicate(num_sequences, sample(1:99, num_numbers, replace = TRUE), ...
     simplify = FALSE)
 return(random_sequences)
}
```

Generating and Transposing Random Sequences The generated sequences are combined into a matrix and transposed for storage:

Listing 16: Generating and Transposing Random Sequences

```r
# Parameters
num_sequences <- 1e6
num_numbers <- 250

# Generate Pseudo-random sequences
Percentile_sequences_random <- generate_pseudo_random_sequences(num_sequences, num_numbers)

# Convert the list to a matrix and transpose it
Percentile_sequences_matrix <- t(do.call(rbind, Percentile_sequences_random))
```

Saving to a CSV File The pseudo-random sequences are saved to a CSV file for subsequent use:

Listing 17: Saving Random Sequences to CSV

```r
file_path_random_csv <- "Random_Sequence.csv"
write.csv(Percentile_sequences_matrix, file_path_random_csv, row.names = FALSE)
```

—

4.7.4 Insights and Applications

The randomization process enables:

- Creation of large-scale probabilistic datasets for use in energy modeling.
- Representation of uncertainty and variability in model parameters.
- Efficient storage of sequences for repeated use across simulations.

The generated pseudo-random sequences are integrated into the generation expansion model, adding flexibility and robustness to the analysis.

4.8 Dispatch Curve Generation

4.8.1 Overview

The `dispatch_curve_base.R` and `dispatch_curve_rep_days.R` scripts generate dispatch curves for electricity generation. A dispatch curve offsets electricity demand with available generation capacities, ensuring an optimal balance between supply and demand. The scripts focus on:

- Integrating data on installed capacities and facilities for various energy sources.
- Modeling generation patterns for both hourly and representative days.
- Accounting for constraints such as ramp rates and capacity factors.

—

4.9 Dispatch Curve for Full Data

4.9.1 Installed Capacity and Facility Data

The `dispatch_curve_base.R` script begins by loading data on installed capacities for renewable and non-renewable energy sources. This includes data for wind, solar, nuclear, hydro, biomass, batteries, SMRs, imports, and natural gas facilities. Fossil fuel data is also integrated.

Listing 18: Loading Installed Capacity Data

```r
# Load installed capacity data
file_path <- "Hourly_Installed_Capacity.csv"
```

```r
Hourly_Installed_Capacity <- fread(file_path)

# Load SMR facility data
file_path <- "SMR_Facility_Data.csv"
SMR_Facility_Data <- fread(file_path)

# Load fossil fuel facilities data
file_path <- "Fossil_Fuel_Facilities_Data.csv"
Fossil_Fuels_NPC <- fread(file_path)
```

4.9.2 Balancing Demand and Generation

The script calculates the dispatch curve by offsetting demand with generation across all sources, prioritizing renewable energy. Excess or unmet demand is identified and logged.

Listing 19: Calculating Dispatch Curve

```r
# Calculate the total generation for each source
Hourly_Generation <- Hourly_Installed_Capacity[, .(
  Total_Generation = sum(Wind + Solar + Nuclear + Hydro + Bio + Battery + SMR + NG, na.rm ...
      = TRUE)
)]

# Offset demand with generation
Hourly_Dispatch <- Hourly_Generation[, .(
  Net_Demand = Demand - Total_Generation,
  Excess_Generation = pmax(0, Total_Generation - Demand),
  Unmet_Demand = pmax(0, Demand - Total_Generation)
)]
```

—

5 Dispatch Curve Results Processing

Write this up!!!

6 Total Costs

6.1 CAPEX, FOM, and VOM Costs

6.1.1 CAPEX and FOM for Fossil Energy

The `1_CAPEX_FOM_ATB_Fossil.R` script calculates capital expenditures (CAPEX) and fixed operation and maintenance costs (FOM) for fossil fuel-based facilities. It performs the following tasks:

- Loads ATB (Annual Technology Baseline) cost data for fossil fuels.
- Calculates net present value (NPV) for CAPEX and FOM using a discount rate of 2%.
- Extracts data for various fossil fuel types, such as coal, gas, oil, and wood.

Key Code:

Listing 20: Calculating NPV for Fossil CAPEX and FOM

```r
calculate_npv <- function(dt, rate, base_year) {
  npv <- sum(dt[,2] / (1 + rate)^(dt[,1] - base_year))
  return(npv)
}

discount_rate <- 0.02
base_year <- 2023

Coal_NPC <- Fossil_Fuels_NPC[grepl("Coal", Primary.Fuel.Type, ignore.case = TRUE)]
Gas_CC_NPC <- Fossil_Fuels_NPC[grepl("Gas", Primary.Fuel.Type, ignore.case = TRUE) &
                                grepl("Combined", Unit.Type, ignore.case = TRUE)]
```

6.1.2 CAPEX and FOM for Non-Fossil Energy

The `2_CAPEX_FOM_ATB_Non_Fossil.R` script focuses on renewable and nuclear energy sources. It:

- Loads ATB data for non-fossil technologies, including solar, onshore wind, offshore wind, nuclear, and hydro.
- Calculates CAPEX and FOM for each technology across different scenarios (Advanced, Moderate, Conservative).

Key Code:

Listing 21: Loading and Setting Non-Fossil Data

```r
ATBe <- fread("ATBe.csv")
path <- "Installed Capacity and CF Non Fossil"
files <- list.files(path = path, pattern = "\\.csv$", full.names = TRUE)

datasets <- list()
for (file in files) {
  dataset_name <- gsub(".*/|\\.csv$", "", file)
  datasets[[dataset_name]] <- fread(file)
}

list2env(datasets, envir = .GlobalEnv)  # Release datasets into the environment
setDT(Solar_NPC)
setDT(Nuclear_hydro_bio_NPC)
```

6.1.3 CAPEX and FOM for Imports

The `3_CAPEX_FOM_Imports.R` script calculates CAPEX and FOM costs for import energy capacity. Key operations include:

- Loading capacity data for imports across three scenarios (S1, S2, S3).
- Calculating new capacity built each year and the associated costs.

Key Code:

Listing 22: Loading and Calculating Import Capacity

```r
Imports_S1 <- read_excel("Import Capacity.xlsx", sheet = "S1")
Imports_S2 <- read_excel("Import Capacity.xlsx", sheet = "S2")
Imports_S3 <- read_excel("Import Capacity.xlsx", sheet = "S3")

Imports_Capacity <- rbindlist(list(Imports_S1, Imports_S2, Imports_S3), use.names = TRUE, ...
    fill = TRUE)
Imports_Capacity[, new_capacity := QC - shift(QC, 1, type = "lag"), by = Scenario]
```

—

6.1.4 VOM for Fossil Energy

The `4_VOM_ATB_Fossil.R` script calculates variable operation and maintenance (VOM) costs for fossil fuels:

- Merges fossil fuel generation data with facility-specific data.
- Calculates yearly VOM costs for different fossil fuel technologies, accounting for scenarios in the ATB data.

Key Code:

Listing 23: Calculating VOM Costs for Fossil Fuels

```r
Yearly_Facility_Level_Results <- selected_cols[Yearly_Facility_Level_Results, on = ...
    "Facility_Unit.ID"]
Yearly_Facility_Level_Results[, VOM_Cost := Generation * VOM_Rate, by = Unit.Type]
```

6.1.5 VOM for Non-Fossil Energy

The 5_VOM_ATB_Non_Fossil.R script calculates VOM costs for renewable and nuclear energy. It:

- Processes VOM costs for solar, wind, hydro, and nuclear technologies.
- Filters data based on ATB scenarios and technology-specific parameters.

Key Code:

Listing 24: Processing VOM Costs for Non-Fossil Technologies

```r
process_non_fossil <- function(technology, techdetail, dataset, column_name, simulation, ...
    scenario) {
  dataset <- dataset[Technology == technology & TechDetail == techdetail]
  dataset[, VOM_Cost := Generation * VOM_Rate]
  return(dataset)
}
```

—

6.1.6 Insights and Applications

These scripts collectively provide comprehensive cost assessments for CAPEX, FOM, and VOM across fossil, non-fossil, and imported energy sources. The outputs are critical for evaluating the economic feasibility of different decarbonization pathways under various scenarios.

6.2 Fuel Costs

6.2.1 Fuel Costs for Fossil Energy

The 7_Fuel_ATB_Fossil.R script calculates fuel costs for fossil fuel energy sources using the NREL ATB (Annual Technology Baseline) dataset. Key operations include:

- Loading CPI (Consumer Price Index) data via the fredr API to adjust costs to 2024 dollars.
- Loading ATB fuel cost data for fossil fuels, including coal, natural gas, and oil.
- Calculating Net Present Value (NPV) of fuel costs using a 2% discount rate.

Key Code:

Listing 25: Loading and Adjusting Fuel Costs for Fossil Fuels

```r
# Set discount rate and base year
discount_rate <- 0.02
base_year <- 2023

# Load CPI data to calculate conversion rate
cpi_data <- fredr(series_id = "CPIAUCSL", observation_start = as.Date("2000-01-01"), ...
    observation_end = as.Date("2024-01-01"))
cpi_2021 <- filter(cpi_data, year(date) == 2021) %>% summarise(YearlyAvg = mean(value))
cpi_2024 <- filter(cpi_data, year(date) == 2024) %>% summarise(YearlyAvg = mean(value))
conversion_rate <- cpi_2024$YearlyAvg / cpi_2021$YearlyAvg

# Load ATB fuel cost data
file_path <- "ATB_2021_Fuel_Costs.csv"
ATB_Fuel_Costs <- fread(file_path)
ATB_Fuel_Costs[, Adjusted_Cost := Original_Cost * conversion_rate]
```

This script adjusts fossil fuel costs to 2024 dollars and calculates NPVs to evaluate long-term fuel expenditure.
—

6.2.2 Fuel Costs for Non-Fossil Energy

The `8_Fuel_ATB_Non_Fossil.R` script processes fuel costs for non-fossil energy sources such as nuclear, hydro, and biomass. Its primary functions include:

- Loading ATB cost data for non-fossil fuel types.
- Filtering data based on technology, scenarios, and simulation parameters.
- Calculating VOM (Variable Operation and Maintenance) costs where applicable.

Key Code:

Listing 26: Processing Non-Fossil Fuel Costs

```
process_non_fossil <- function(technology, techdetail, dataset, column_name, simulation, ...
    scenario) {
  dataset <- dataset[Technology == technology & TechDetail == techdetail]
  dataset[, Adjusted_Cost := Fuel_Cost * VOM_Rate]
  return(dataset)
}

# Example: Processing nuclear fuel costs
process_non_fossil("Nuclear", "Advanced", ATBe, "Fuel_Cost", "Simulation1", "Scenario1")
```

This script enables scenario-based analysis of non-fossil fuel costs under varying economic and technological assumptions.

—

6.2.3 Insights and Applications

These scripts provide a detailed analysis of fuel costs for fossil and non-fossil energy sources:

- Fossil fuel costs are adjusted to 2024 dollars using CPI data and evaluated for long-term feasibility.
- Non-fossil fuel costs are integrated into scenario-based models to assess their economic competitiveness.

The outputs support informed decision-making in decarbonization pathway modeling.

6.3 Imports Costs

6.3.1 Overview

The `6_Imports.R` script calculates costs related to electricity imports. Key functionalities include:

- Adjusting costs to 2024 dollars using Consumer Price Index (CPI) data retrieved from the `fredr` API.
- Calculating Net Present Value (NPV) of imports costs using a discount rate of 2%.
- Processing capacity data across multiple scenarios to estimate annual costs.

6.3.2 Detailed Explanation

1. **Loading CPI Data**:
   - The script retrieves CPI data for 2021, 2022, and 2024 to adjust historical costs to 2024 values.
   - Conversion rates are calculated based on the ratio of CPI values.
2. **Loading Capacity Data**:
   - Import capacity data is loaded from an Excel file, with separate sheets representing different scenarios (e.g., S1, S2, S3).
   - New capacity additions are calculated for each year within each scenario.
3. **NPV Calculation**: The script calculates the NPV of costs for each scenario, providing a long-term cost assessment.

6.3.3 Key Code

CPI Adjustments:

Listing 27: Loading and Adjusting CPI Data

```r
# Set API key and retrieve CPI data
fredr_set_key("63522eae4ec927d6f1d9d86bf7826cc8")
cpi_data <- fredr(series_id = "CPIAUCSL", observation_start = as.Date("2000-01-01"), ...
    observation_end = as.Date("2024-01-01"))

# Calculate conversion rates
cpi_2021 <- filter(cpi_data, year(date) == 2021) %>% summarise(YearlyAvg = mean(value))
cpi_2024 <- filter(cpi_data, year(date) == 2024) %>% summarise(YearlyAvg = mean(value))
conversion_rate_2021 <- cpi_2024$YearlyAvg / cpi_2021$YearlyAvg
```

Loading Capacity Data:

Listing 28: Loading Import Capacity Data

```r
path <- "Imports_Capacity.xlsx"

# Load data for different scenarios
Imports_S1 <- read_excel(path, sheet = "S1")
Imports_S2 <- read_excel(path, sheet = "S2")
Imports_S3 <- read_excel(path, sheet = "S3")

# Combine and calculate new capacity
Imports_Capacity <- rbindlist(list(Imports_S1, Imports_S2, Imports_S3), use.names = TRUE, ...
    fill = TRUE)
Imports_Capacity[, new_capacity := QC - shift(QC, 1, type = "lag"), by = Scenario]
```

NPV Calculation:

Listing 29: Calculating NPV for Import Costs

```r
calculate_npv <- function(dt, rate, base_year, col) {
  npv <- sum(dt[[col]] / (1 + rate)^(dt[['Year']] - base_year))
  return(npv)
}

discount_rate <- 0.02
base_year <- 2023

# Example NPV calculation
npv_scenario1 <- calculate_npv(Imports_Capacity[Scenario == "S1"], discount_rate, ...
    base_year, "new_capacity")
```

6.3.4 Insights and Applications

This script provides a detailed assessment of import costs:

- Adjusting for inflation ensures that all costs are consistent with 2024 dollars.
- Scenario-based analysis enables a comparison of costs across different import strategies.
- NPV calculations provide a long-term perspective on import-related expenditures.

6.4 GHG Emissions Costs

6.4.1 Overview

The `9_GHG_Emissions.R` script calculates the costs associated with greenhouse gas (GHG) emissions. Key functionalities include:

- Loading GHG emissions data and converting all values to metric tons of CO_2-equivalent using global warming potentials (GWP) from IPCC's AR4.

- Interpolating GHG costs over time from a 2021 starting value to a 2050 target value.
- Adjusting costs to 2024 dollars using CPI (Consumer Price Index) data from the `fredr` API.
- Calculating the Net Present Value (NPV) of GHG costs using a 2% discount rate.

6.4.2 Detailed Explanation

1. **Loading Emissions Data**:
   - GHG emissions data is loaded in metric tons of CO_2-equivalent.
   - Data includes emissions from fossil fuels, imports, and other energy sources.
2. **CPI Adjustments**: Costs are adjusted to 2024 dollars using CPI data retrieved from the `fredr` API.
3. **Cost Interpolation**: An interpolation function is used to estimate GHG costs between the 2021 base year and the 2050 target year.
4. **NPV Calculation**: The script calculates the NPV of GHG emissions costs for long-term cost assessment.

6.4.3 Key Code

CPI Adjustments:

Listing 30: Adjusting Costs Using CPI Data

```r
fredr_set_key("63522eae4ec927d6f1d9d86bf7826cc8")
cpi_data <- fredr(series_id = "CPIAUCSL", observation_start = as.Date("2000-01-01"), ...
    observation_end = as.Date("2024-01-01"))

# Calculate conversion rate
cpi_2024 <- filter(cpi_data, year(date) == 2024) %>% summarise(YearlyAvg = mean(value))
conversion_rate <- cpi_2024$YearlyAvg / 1.0  # Assuming base rate is 1.0
```

Interpolating GHG Costs:

Listing 31: Interpolating GHG Costs Over Time

```r
interpolate_cost <- function(year, start_year, end_year, start_cost, end_cost) {
  return(start_cost + (end_cost - start_cost) * (year - start_year) / (end_year - start_year))
}

# Example interpolation for 2030
ghg_cost_2030 <- interpolate_cost(2030, 2021, 2050, start_cost = 50, end_cost = 200)
```

Calculating NPV for GHG Costs:

Listing 32: Calculating NPV for GHG Costs

```r
calculate_npv <- function(dt, rate, base_year, col) {
  npv <- sum(dt[[col]] / (1 + rate)^(dt[["Year"]] - base_year))
  return(npv)
}

discount_rate <- 0.02
base_year <- 2024

# Example NPV calculation
npv_ghg_costs <- calculate_npv(ghg_emissions_data, discount_rate, base_year, "Cost")
```

6.4.4 Insights and Applications

This script enables a detailed assessment of GHG emissions costs:

- Interpolation of costs provides flexibility in modeling emissions cost trajectories over time.
- CPI adjustments ensure all costs are aligned with a 2024 dollar value, enhancing comparability.
- NPV calculations support long-term planning and economic feasibility studies in decarbonization pathways.

6.5 Air Pollutant Emissions Costs

6.5.1 Overview

The `10_Air_emissions.R` script calculates costs associated with air pollutant emissions. Key functionalities include

- Converting emission quantities to metric tons for consistency.
- Adjusting costs to 2024 dollars using Consumer Price Index (CPI) data from the `fredr` API.
- Estimation of the net present value (NPV) of the air pollutant costs using a discount rate of 2%.

6.5.2 Detailed Explanation

1. **Emission unit conversions**:
   - Emission data are converted from pounds (lbs) to US tons, and subsequently to metric tons for compatibility with international standards.
2. **CPI Adjustments**: Costs are adjusted to 2024 dollars by calculating a conversion rate between CPI values from 2000 and 2024.
3. **NPV Calculation**: The script calculates the NPV of air pollutant costs, enabling long-term economic analysis.

6.5.3 Key Code

Unit Conversions:

Listing 33: Converting Emission Units to Metric Tons

```r
lbs_tons_conversion <- 1 / 2204.62   # lbs to tons
ton_conversion <- 0.907185  # US tons to metric tons

# Example conversion
emission_data[, Metric_Tons := Emissions_Lbs * lbs_tons_conversion * ton_conversion]
```

CPI Adjustments:

Listing 34: Adjusting Costs Using CPI Data

```r
fredr_set_key("63522eae4ec927d6f1d9d86bf7826cc8")
cpi_data <- fredr(series_id = "CPIAUCSL", observation_start = as.Date("2000-01-01"), ...
    observation_end = as.Date("2024-01-01"))

# Calculate conversion rate
cpi_2000 <- filter(cpi_data, year(date) == 2000) %>% summarise(YearlyAvg = mean(value))
cpi_2024 <- filter(cpi_data, year(date) == 2024) %>% summarise(YearlyAvg = mean(value))
conversion_rate <- cpi_2024$YearlyAvg / cpi_2000$YearlyAvg

# Adjust costs
emission_data[, Adjusted_Cost := Original_Cost * conversion_rate]
```

NPV Calculation:

Listing 35: Calculating NPV for Air Pollutant Costs

```r
calculate_npv <- function(dt, rate, base_year, col) {
  npv <- sum(dt[[col]] / (1 + rate)^(dt[["Year"]] - base_year))
  return(npv)
}

discount_rate <- 0.02
base_year <- 2024

# Example NPV calculation
npv_air_emissions <- calculate_npv(emission_data, discount_rate, base_year, "Adjusted_Cost")
```

6.5.4 Insights and Applications

This script supports air pollutant cost modeling by:

- Ensuring consistency in emission quantities via unit conversions.
- Adjusting historical costs to 2024 values for accurate comparisons.
- Providing a long-term economic perspective through NPV calculations.

These analyses are essential for assessing the economic impact of air pollutant emissions and supporting policy-making in decarbonization strategies.

6.6 Unmet Demand Penalty Costs

6.6.1 Overview

The `11_Unmet_Demand.R` script estimates the penalties associated with unmet electricity demand. Key functionalities include:

- Loading yearly electricity shortage data from the results summary file.
- Calculating the range of shortage ratios (upper and lower bounds) based on the mean and standard deviation of shortages.
- Applying cost estimates for unmet demand using the ISO-NE 2025 Estimated Value of Lost Load (EVOLL).
- Calculating Net Present Value (NPV) of unmet demand penalties using a 2% discount rate.

6.6.2 Detailed Explanation

1. **Loading Shortage Data**: The script reads yearly results of electricity shortages, including mean and standard deviation values for shortage ratios.
2. **Shortage Ratio Bounds**: Upper and lower bounds for shortage ratios are calculated using a step size parameter.
3. **Applying Cost Estimates**: Unmet demand costs are computed using the EVOLL value from ISO-NE 2025 reports, adjusted to represent penalties in $ per TWh.
4. **NPV Calculation**: The NPV of unmet demand costs is calculated to assess long-term economic impact.

6.6.3 Key Code

Loading Shortage Data:

Listing 36: Loading Yearly Shortage Data

```r
# Load Results
rds_path <- "Yearly_Results_Shortages.rds"
Yearly_Results <- readRDS(rds_path)
setDT(Yearly_Results)
```

Calculating Shortage Ratio Bounds:

Listing 37: Calculating Upper and Lower Shortage Bounds

```r
sd_mean_step <- 1

Yearly_Results[, Shortage_ratio_upper := (Shortage_ratio_mean + sd_mean_step * ...
    Shortage_ratio_sd) * 100]  # Percentage
Yearly_Results[, Shortage_ratio_lower := (Shortage_ratio_mean - sd_mean_step * ...
    Shortage_ratio_sd) * 100]

# Ensure lower bounds are non-negative
Yearly_Results[Shortage_ratio_lower < 0, Shortage_ratio_lower := 0]
```

Applying Cost Estimates:

Listing 38: Calculating Unmet Demand Costs

```r
# Estimated Value of Lost Load (EVOLL) in $/TWh
EVOLL_2025 <- 9337 * 1e6  # Converted to $/TWh

# Calculate unmet demand cost
Yearly_Results[, Unmet_Demand_Cost := (Shortage_ratio_mean / 100) * Demand.mean_yr_MW * ...
    EVOLL_2025]
```

NPV Calculation:

Listing 39: Calculating NPV for Unmet Demand Penalties

```r
calculate_npv <- function(dt, rate, base_year, col) {
  npv <- sum(dt[[col]] / (1 + rate)^(dt[["Year"]] - base_year))
  return(npv)
}

discount_rate <- 0.02
base_year <- 2023

# Example NPV calculation
npv_unmet_demand <- calculate_npv(Yearly_Results, discount_rate, base_year, ...
    "Unmet_Demand_Cost")
```

6.6.4 Insights and Applications

This script provides a detailed assessment of unmet demand penalties:

- Shortage ratios account for variability and uncertainty using upper and lower bounds.
- EVOLL values ensure cost calculations are aligned with industry standards.
- NPV calculations provide a long-term perspective on unmet demand penalties, critical for ensuring system reliability in decarbonization scenarios.

6.7 Beyond the Fence Line Costs

6.7.1 Overview

The scripts `12_Other_S3_CAN_Hydro_CostAssumptions.R` and `12_Other_S3_CAN_Hydro.R` focus on hydropower cost modeling and capacity calculations for Canada under Scenario 3 (S3). Key functionalities include:

- Estimating cost distributions for different hydropower classes using log-normal distribution parameters.
- Modeling the incremental installation of hydropower capacity between 2024 and 2050.
- Calculating Net Present Value (NPV) of hydropower-related costs.
- Adjusting all costs to 2024 dollars using Consumer Price Index (CPI) data.

—

6.7.2 Hydropower Cost Assumptions

The script `12_Other_S3_CAN_Hydro_CostAssumptions.R` defines cost assumptions for hydropower classes (Class 3, 4, and 5) using log-normal distributions based on provided percentiles.
Key Code for Cost Assumptions:

Listing 40: Cost Distribution Calculations for Hydropower Classes

```r
# Define mean and standard deviation for each class
mean_class_3 <- 1.24
mean_class_4 <- 1.40
mean_class_5 <- 1.79

sd_class_3 <- (log(1.63) - log(0.99)) / (qnorm(0.9) - qnorm(0.1))
sd_class_4 <- (log(2.09) - log(1.06)) / (qnorm(0.9) - qnorm(0.1))
sd_class_5 <- (log(3.01) - log(1.27)) / (qnorm(0.9) - qnorm(0.1))

# Function to calculate density for a given class
calculate_density <- function(x, mean, sd) {
  dlnorm(x, meanlog = log(mean), sdlog = sd)
}

# Define range and calculate densities
x_values <- seq(0.6, 3.4, length.out = 100)
density_class_3 <- calculate_density(x_values, mean_class_3, sd_class_3)
density_class_4 <- calculate_density(x_values, mean_class_4, sd_class_4)
density_class_5 <- calculate_density(x_values, mean_class_5, sd_class_5)
```

This script calculates the cost probability distributions for hydropower classes, allowing stochastic modeling of cost uncertainties.

—

6.7.3 Hydropower Capacity and Costs

The script `12_Other_S3_CAN_Hydro.R` calculates the incremental installation of hydropower capacity in Canada from 2024 to 2050 and adjusts all costs to 2024 dollars using CPI data.
Key Code for Capacity Modeling:

Listing 41: Incremental Installation of Hydropower Capacity

```r
years <- 2024:2050
increment <- (9000 - 0) / (2045 - 2024)  # Linear increment until 2045
values <- ifelse(years ≤ 2045, 0 + (years - 2024) * increment, 9000)

Hydropower <- data.table(Year = years, Capacity_MW = values)
```

CPI Adjustments for Costs:

Listing 42: Adjusting Costs Using CPI Data

```r
fredr_set_key("63522eae4ec927d6f1d9d86bf7826cc8")
cpi_data <- fredr(series_id = "CPIAUCSL", observation_start = as.Date("2000-01-01"), ...
    observation_end = as.Date("2024-01-01"))

# Calculate conversion rate
```

```r
cpi_2024 <- filter(cpi_data, year(date) == 2024) %>% summarise(YearlyAvg = mean(value))
conversion_rate <- cpi_2024$YearlyAvg / 1.0  # Assuming base cost is normalized

# Adjust hydropower costs
Hydropower[, Adjusted_Cost := Original_Cost * conversion_rate]
```

NPV Calculation for Hydropower Costs:

Listing 43: Calculating NPV for Hydropower Costs

```r
calculate_npv <- function(dt, rate, base_year, col) {
  npv <- sum(dt[[col]] / (1 + rate)^(dt[["Year"]] - base_year))
  return(npv)
}

discount_rate <- 0.02
base_year <- 2024

# Example NPV calculation
npv_hydro_costs <- calculate_npv(Hydropower, discount_rate, base_year, "Adjusted_Cost")
```

—

6.7.4 Insights and Applications

These scripts provide a comprehensive analysis of hydropower costs and capacity for Scenario 3 in Canada:

- Cost assumptions use log-normal distributions, reflecting uncertainties in hydropower class costs.
- Incremental capacity modeling estimates realistic growth in hydropower installations up to 2050.
- CPI adjustments ensure that all costs are aligned to 2024 dollars for consistency.
- NPV calculations provide a long-term economic perspective, aiding in scenario analysis and planning.

6.8 Total Costs

6.8.1 Overview

The `All_Costs_Tabulated.R` script consolidates all cost data into a single tabular format for comprehensive analysis. This includes:

- Loading and merging cost data for CAPEX, FOM, VOM, fuels, GHG emissions, air pollutant emissions, and unmet demand.
- Expanding scenarios (e.g., ”S1-3”) into individual scenario components for detailed scenario-specific analyses.
- Formatting and saving the consolidated dataset for visualization and modeling.

6.8.2 Detailed Explanation

1. **Scenario Expansion**: Scenarios labeled as ”S1-3” are expanded into their individual components (”S1”, ”S2”, and ”S3”) to ensure data accuracy during aggregation.
2. **File Loading**: The script recursively loads all data files related to cost components (e.g., CAPEX, GHG emissions) and renames columns to ensure consistency across datasets.
3. **Data Consolidation**: All cost data is combined into a single dataset, enabling streamlined analysis and visualization.

6.8.3 Key Code

Scenario Expansion:

Listing 44: Expanding Scenarios for Cost Analysis

```r
expand_scenarios <- function(scenario) {
  if (scenario == "S1-3") {
    return(c("S1", "S2", "S3"))
  } else {
    return(scenario)
  }
}
```

File Loading and Renaming:

Listing 45: Loading and Renaming Cost Data

```r
file_paths <- list.files(path = "NPV Results", recursive = TRUE, full.names = TRUE)
data_tables <- lapply(file_paths, function(file) {
  data <- fread(file)
  # Extract the base name without extension to use as the name
  name <- tools::file_path_sans_ext(basename(file))
  setnames(data, old = names(data), new = paste0(name, "_", names(data)))
  return(data)
})
```

Data Consolidation:

Listing 46: Consolidating All Cost Data

```r
# Combine all tables into a single dataset
all_costs <- rbindlist(data_tables, fill = TRUE, use.names = TRUE)

# Save consolidated data for further analysis
fwrite(all_costs, file = "All_Costs_Consolidated.csv")
```

6.8.4 Insights and Applications

This script ensures efficient consolidation of all cost components, supporting:

- Comprehensive analysis of cost breakdowns across scenarios and energy types.
- Integration of cost data into visualization tools such as `ggplot2` for detailed insights.
- Streamlined modeling by providing a unified dataset for subsequent analyses.

The resulting consolidated dataset is pivotal for scenario-based decision-making and decarbonization pathway evaluation.

6.9 CPI and Other Cost Variables

6.9.1 Overview

The `CPI_Electricity.R` script retrieves Consumer Price Index (CPI) data from the Bureau of Labor Statistics (BLS) API. It focuses on CPI series related to electricity costs across multiple regions. Key functionalities include:

- Retrieving regional CPI data for electricity using the BLS API.
- Structuring the retrieved data into a tabular format.
- Storing the processed data for integration into cost modeling workflows.

6.9.2 Detailed Explanation

1. **API Integration**: The script uses the `blsAPI` and `rjson` libraries to query the BLS API for CPI series data.
2. **Regions and Series**: The script specifies electricity-related series IDs and their corresponding regions, such as New England, Boston, and Middle Atlantic.
3. **Data Processing**: Retrieved data is organized into a data frame with fields for year, period, and CPI values.

6.9.3 Key Code

API Query and Data Retrieval:

Listing 47: Retrieving CPI Data Using BLS API

```
library(blsAPI)
library(rjson)

api_key <- "a2c1e702f81947118e83cb8d029d6623"
series_ids <- c("APU011072610", "APUS11A72610", "APU012072610", "APUS12A72610", ...
    "APUS12B72610")
regions <- c("New England", "Boston", "Middle Atlantic", "New York", "Philadelphia")

payload <- list(
  'seriesid' = series_ids,
  'regions' = regions,
  'startyear' = '2022',
  'endyear' = '2024',
  'registrationkey' = api_key
)

response <- blsAPI(payload)
json_data <- fromJSON(response)
```

Data Organization:

Listing 48: Organizing CPI Data into a Data Frame

```
cpi_data <- do.call(rbind, lapply(json_data$Results$series, function(series) {
  data.frame(
    seriesID = series$seriesID,
    Region = series$regions,
    year = sapply(series$data, function(x) x$year),
    period = sapply(series$data, function(x) x$period),
    value = sapply(series$data, function(x) as.numeric(x$value))
  )
}))
```

—

6.10 Supplemental Costs Calculation

6.10.1 Overview

The `Supplemental_Costs_calc.R` script calculates supplemental costs using GDP, population, and household data. Key functionalities include:

- Loading and filtering population and GDP growth data for the years 2025–2050.
- Converting GDP and household data to 2023 dollars using CPI adjustments.
- Preparing the processed data for integration into demand and cost models.

6.10.2 Detailed Explanation

1. **Data Loading**: Population and GDP data are loaded from an Excel file and filtered for the target years (2025–2050).
2. **CPI Adjustments**: CPI data is retrieved from the FRED API and used to adjust GDP values from 2012 to 2023 dollars.
3. **Data Preparation**: The processed data is stored for further analysis in supplemental cost modeling.

6.10.3 Key Code

Loading and Filtering Data:

Listing 49: Loading Population and GDP Data

```
path <- "/path/to/population_GDP_growth.xlsx"

household_data <- read_excel(path)
household_data <- household_data %>%
  filter(Year ≥ 2025 & Year ≤ 2050)
colnames(household_data) <- c("Year", "n_households_millions", "GDP_bUSD_2012")
```

CPI Adjustments:

Listing 50: Adjusting GDP to 2023 Dollars

```
library(fredr)

fredr_set_key("63522eae4ec927d6f1d9d86bf7826cc8")
cpi_data <- fredr(series_id = "CPIAUCSL", observation_start = as.Date("2000-01-01"), ...
    observation_end = as.Date("2024-01-01"))

cpi_2012 <- filter(cpi_data, year(date) == 2012) %>% summarise(YearlyAvg = mean(value))
cpi_2023 <- filter(cpi_data, year(date) == 2023) %>% summarise(YearlyAvg = mean(value))

conversion_rate <- cpi_2023$YearlyAvg / cpi_2012$YearlyAvg
household_data$GDP_bUSD_2023 <- household_data$GDP_bUSD_2012 * conversion_rate
```

6.10.4 Insights and Applications

These scripts provide critical data processing capabilities:

- **CPI_Electricity.R**: Supports cost modeling by providing up-to-date CPI data for electricity across key regions.
- **Supplemental_Costs_calc.R**: Prepares supplemental cost data, including GDP and household data, adjusted for inflation to 2023 values.

The outputs are essential for integrating economic and demographic variables into energy modeling frameworks.

7 Ecological Impacts

7.0.1 Overview

The `All_Costs_Tabulated.R` script consolidates all cost data into a single tabular format for comprehensive analysis. This includes:

7.1 Land use

7.1.1 Overview

The `All_Costs_Tabulated.R` script consolidates all cost data into a single tabular format for comprehensive analysis. This includes:

7.2 Water withdrawals

7.2.1 Overview

The `All_Costs_Tabulated.R` script consolidates all cost data into a single tabular format for comprehensive analysis. This includes:

7.3 Avian mortality

7.3.1 Overview

The `All_Costs_Tabulated.R` script consolidates all cost data into a single tabular format for comprehensive analysis. This includes:

7.4 View shed

7.4.1 Overview

The `All_Costs_Tabulated.R` script consolidates all cost data into a single tabular format for comprehensive analysis. This includes:

8 Figures

8.0.1 Overview

The `All_Costs_Tabulated.R` script consolidates all cost data into a single tabular format for comprehensive analysis. This includes:
This is some text.