Spring 1-1-2018

Reduced Order Models for Rapid Analysis of Ambient Loops for Commercial Buildings

Nicholas Lee Long
University of Colorado at Boulder, nicholas.lee.long@gmail.com

Reduced Order Models for Rapid Analysis of Ambient Loops for Commercial Buildings

by

Nicholas Long

B.S., Colorado School of Mines, 2002

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Civil, Environmental, and Architectural Engineering

2018

This thesis entitled:
Reduced Order Models for Rapid Analysis of Ambient Loops for Commercial Buildings
written by Nicholas Long
has been approved for the Department of Civil, Environmental, and Architectural Engineering

Gregor P. Henze, Ph.D., P.E.

Rajagopalan Balaji, Ph.D.

Wangda Zuo, Ph.D.

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Long, Nicholas (M.S., Architectural Engineering)

Reduced Order Models for Rapid Analysis of Ambient Loops for Commercial Buildings

Thesis directed by Gregor P. Henze, Ph.D., P.E.

Fifth-generation district heating and cooling (5GHDC) systems are the next generation of district systems which rely on water approaching indoor ambient temperatures (≈20-25°C). The lower temperatures allow for additional heat sources to be added to the network, allowing for some buildings to become prosumers (e.g. data centers, supermarkets through refrigeration heat rejection). The energy performance of buildings served by a 5GDHC network is a strong function of the inlet temperature, which corresponds to the network supply temperature. Optimizing the network's efficiency and its grid topology leads to an evaluation of many possible network topologies, which is numerically expensive. To be able to analyze and contrast any given district energy system layout, the impact of connecting individual buildings to a 5GDHC network must be quickly evaluated, while allowing for flexibility in deciding which buildings should be connected to the network.

Conventionally, each building on a district heating and cooling network is represented by a physical building energy model that is run in conjunction with the network simulation. Although this setup allows for running the analysis with flexibility on the building load side, it results in long-running simulations, thus limiting the ability to quickly analyze various network topologies. An alternate approach to determine the building loads is to create a surrogate model, allowing the network simulation to request the building loads from a myriad of predefined building types and building characteristics. Three reduced order modeling techniques were investigated to replace the conventional long running physical building energy models: ordinary least squares regression (linear models), random forests, and support vector machines.

Dedication

Dedicated to my two daughters, Ella Lee and Alexis Rae. May they have a life full of learning and discovery.

Acknowledgements

I would like to thank the many people who have journeyed with me through my academic and research career to date. First and foremost, I want to extend my gratitude to my family and friends for always being supportive and letting me know that I am eternally loved. Next, I would like to thank my thesis advisor, Gregor P. Henze, for his encouragement, guidance, motivation, and continuous inspiration to pursue knowledge and remain curious. Gregor brought together a great collaborative team with Justus von Rhein, Simone Buffa, and Anders Overgaard to investigate ambient loop systems.

I would like to thank my colleagues at the Department of Energy and the National Renewable Energy Laboratory who supported me with their time, patience, knowledge, encouragement, and friendship. Thanks to Larry Brackney, William Livingood, Luigi Gentile Polese, Drury Crawley, Amir Roth, Harry Bergmann, Katherine Fleming, Edwin Lee, Rob Guglielmetti, Daniel Macumber, Kyle Benne, and many others from the commercial building's research group.

Lastly, I would like to extend a warm thank you to Paul Torcellini who helped mentor me during my early career. Paul's passion for energy efficiency and commercial buildings was contagious and I would not be where I am today without his mentoring, guidance, support, and friendship. Thank you everyone for pushing and challenging me!

Contents

Tables

Table

Figures

Chapter 1

Introduction

Urban design and development are at the forefront of many ongoing projects as the buildings industry moves from the ability to build zero energy buildings to building zero energy districts [1]. One of the challenges of designing zero energy districts is the ability to optimize the interconnection of building heating and cooling systems with district energy systems. In a conventional design context for district energy systems, the systems are sized based on the expected loads of the interconnected buildings. The ability to optimize which buildings are connected to the district energy system, the topology, and the operating temperatures become a time-consuming task. The task involves building energy modeling for connected loads and pressure driven simulation for the network loop design.

Designing a district energy system could be completed in a single engineering modeling tool; however, the building models in these tools are typically simplified models with limited configurations and have potentially long running simulation times. Ideally, the analysis would run a fully defined physical models for each network topology, which requires substantial simulation time and resources while allowing for the largest diversity in design considerations in both the building systems and the district energy system. The approach described in this thesis uses physical building energy models to generate a large parameter space of buildings that are used to generate reduced order models using linear models, random forests, and support vector machines.

The ability to use reduced order models comes with several advantages and disadvantages ranging from time to generate models, time to load the models into memory to run an analysis, and overall accuracy of the reduced order models. The goal is to provide building energy loads to the network topology simulation tool faster than running fully defined physical building energy models. The faster running models should provide similar accuracy, ultimately allowing for a larger number of network topologies and building connections to be evaluated.

1.1 Background and Motivation

Energy consumption has steadily increased over the years, in part due to the energy consumption from new and emerging economies, see Figure 1.1 [2]. A good portion of this increase in energy consumption is related to the growth in the building sector and building-related services, such as heating, ventilation, and air conditioning (HVAC). As developed nations experience aging HVAC systems in need of replacement, and as emerging nations add millions of square meters of buildings per year, the time for evaluating and designing centralized district energy systems has come. Nearly half of the new building construction square footage is occurring in China [3] and the majority in urban areas [4]. In China, the urbanization rate is around 55% and is expected to continue growing as they continue to add nearly 3 billion square meters annual of which nearly 2 billion square meters are urban residential or commercial/public buildings. The pattern of increased urbanization rate is expected to continue throughout the world, which increases the overall need for energy efficient and maintainable district energy systems.

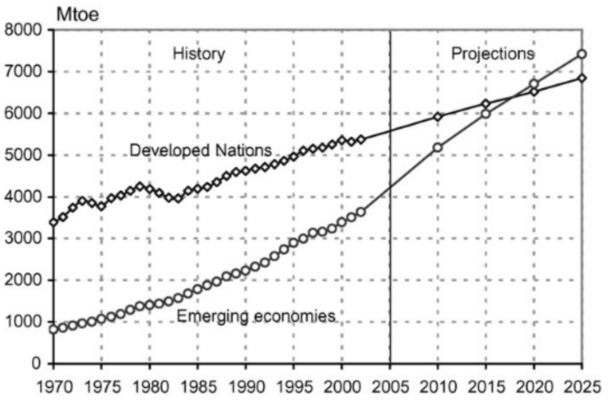


Figure 1.1: Million tons of oil equivalent energy projection

Perez-Lombard, et al., estimate that between 20% to 40% of the world energy consumption is for buildings, see Figure1.2 [2]. The United States uses roughly 40% of its total energy for buildings with 22% in residential and 18% in commercial buildings.

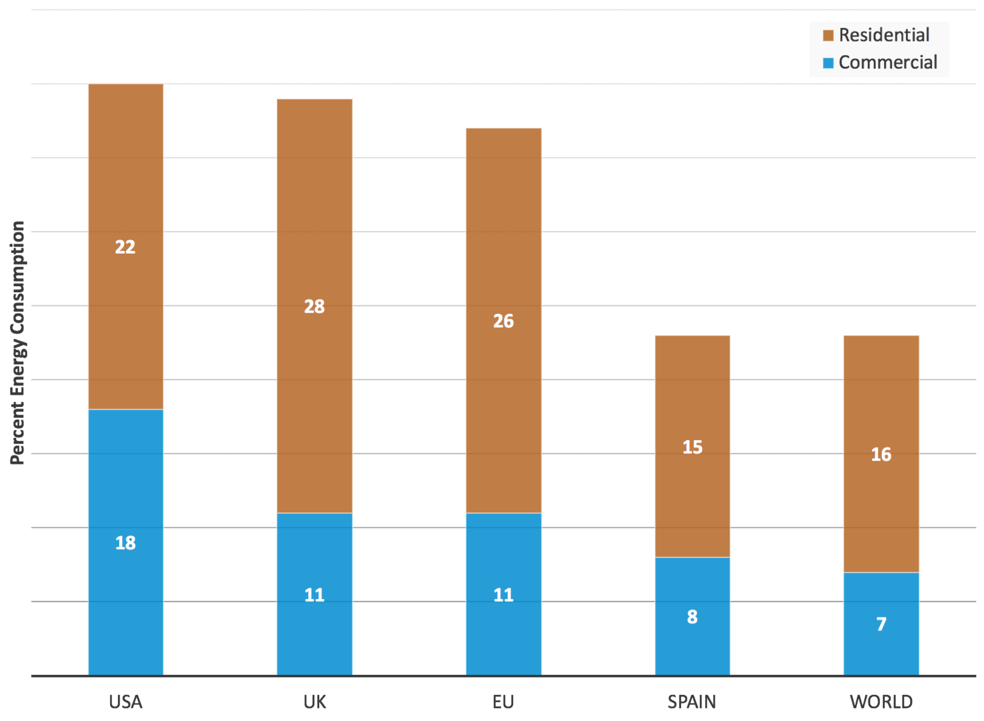


Figure 1.2: Building energy consumption by country

The end-use breakdown for commercial buildings is shown in Figure 1.3. Nearly two-thirds of the energy consumption in commercial buildings is for HVAC and refrigeration (HVAC&R) with 35% used for space heating and service water heating [5]. In the residential sector, nearly half of the energy is used for HVAC&R, with 29% going to space heating and hot water [6], see Figure 1.4. The HVAC&R related end uses are of interest since district heating (and cooling) can be effective at reducing this type of energy consumption.

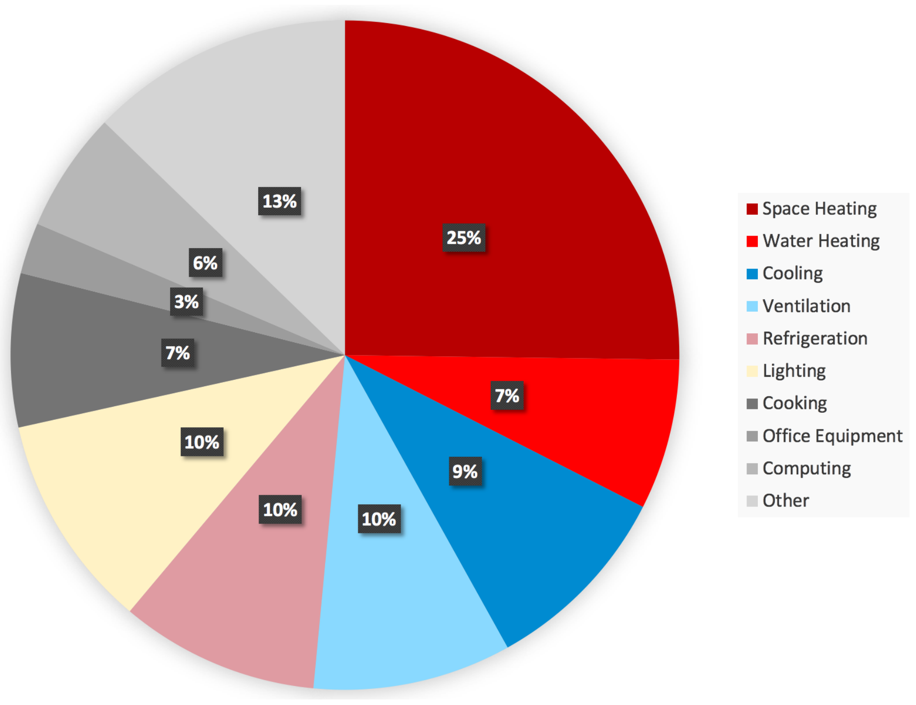


Figure 1.3: Commercial building energy end-use consumption in the United States, by percent

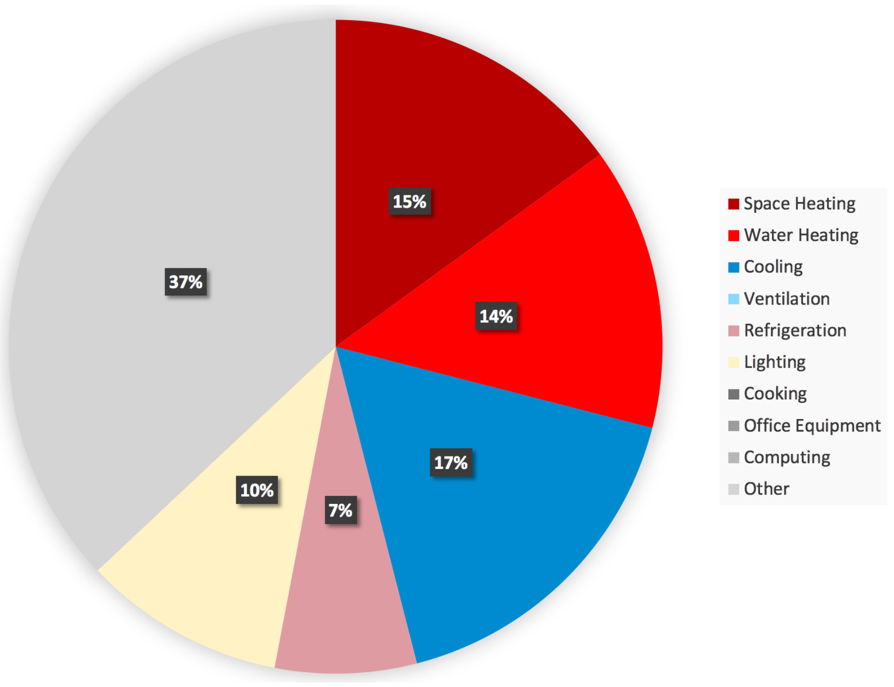


Figure 1.4: Residential building energy end-use consumption in the United States, by percent

In 2012, it was estimated that the United States has 5.6 million commercial buildings totaling nearly 8.4 billion square meters (90 billion square feet) [5]. Similarly, in 2015, the estimated number of residential buildings was 118 million homes (74 million single-family detached homes), totaling 22 billion square meters (237 billion square feet) of which 17.5 billion square meters (189 billion square feet) were for single-family detached homes [6]. It is estimated that only 0.5 billion square meters (5.4 billion square feet) of buildings is heated using district heating and 0.18 billion square meters (1.9 billion square feet) is cooled using district cooling [7]. Figure 1.5 shows the breakdown of district heating and cooling systems by fuel type in the United States as of 2012. As shown, the majority of the systems in the United States are for district heating with the similar fractions of non-CHP heating and CHP heating. The main fuel type for district heating is natural gas for both CHP and non-CHP systems. The energy generated for district cooling is much less, accounting for only 2% of the total energy generated for district energy.

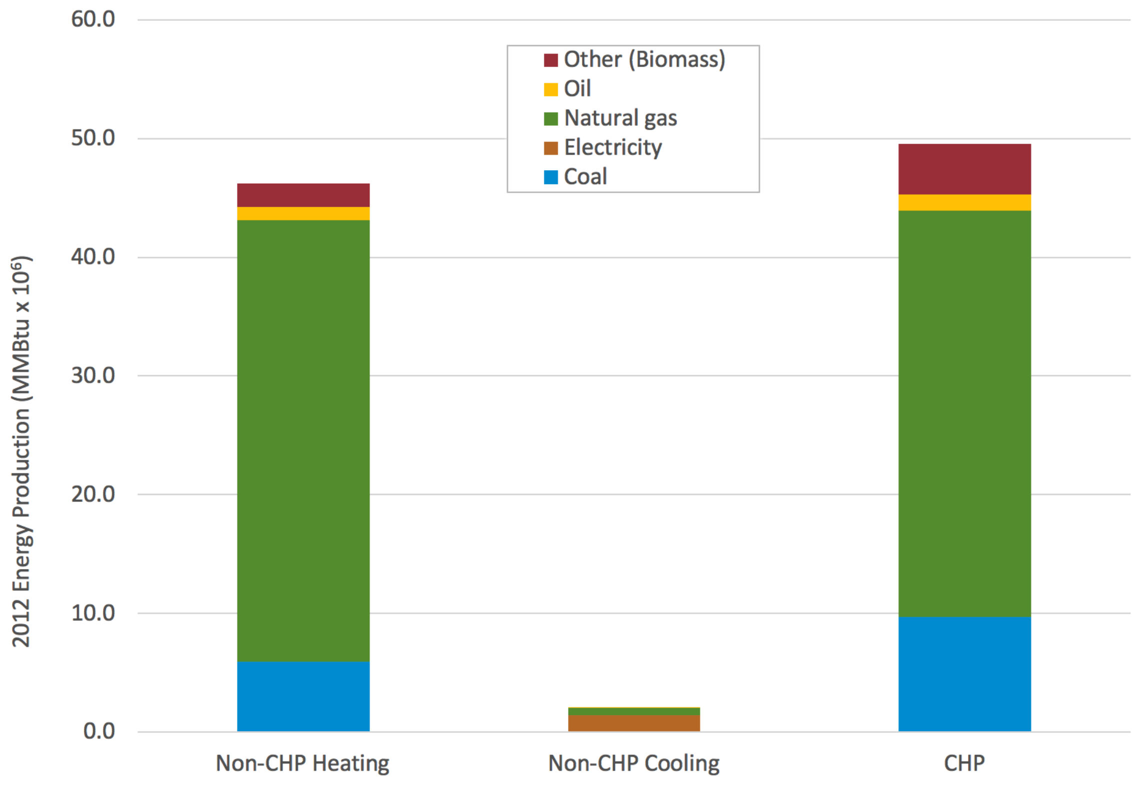


Figure 1.5: District energy systems by type and fuel in the United States

Over the last century, district heating systems have slowly been decreasing loop temperatures in an effort to reduce thermal distribution losses and increase the available types of heat sources. While the district energy industry is currently grappling with adopting low supply water temperatures, on the order of 50-60°C as part of fourth-generation district heating systems [8], the next generation is already on the horizon, termed fifth-generation district heating and cooling systems (5GDHC).

In 5GDHC systems, water at near indoor ambient temperatures (10-25°C) is circulated to accept low-temperature waste heat sources that normally go untapped and to balance coincident heating and cooling loads on the network. In an ideal 5GDHC system, each building is equipped with a water-to-water heat pump (WWHP) to remove energy from the district system and use the heat pump to meet heating and cooling needs. The water-to-water heat pump (WWHP) is installed to boost or reduce the temperature of the ambient loop to the required supply temperature for each individual building. The WWHP can be the only system for the building or could be added as a retrofit system by replacing the building's existing central boiler and chiller plant while keeping unchanged the building's existing air handling units and other secondary delivery systems. Figure 1.6 shows a simple configuration of connecting a retail, office, and residential building to a single source district heating and cooling system [9].

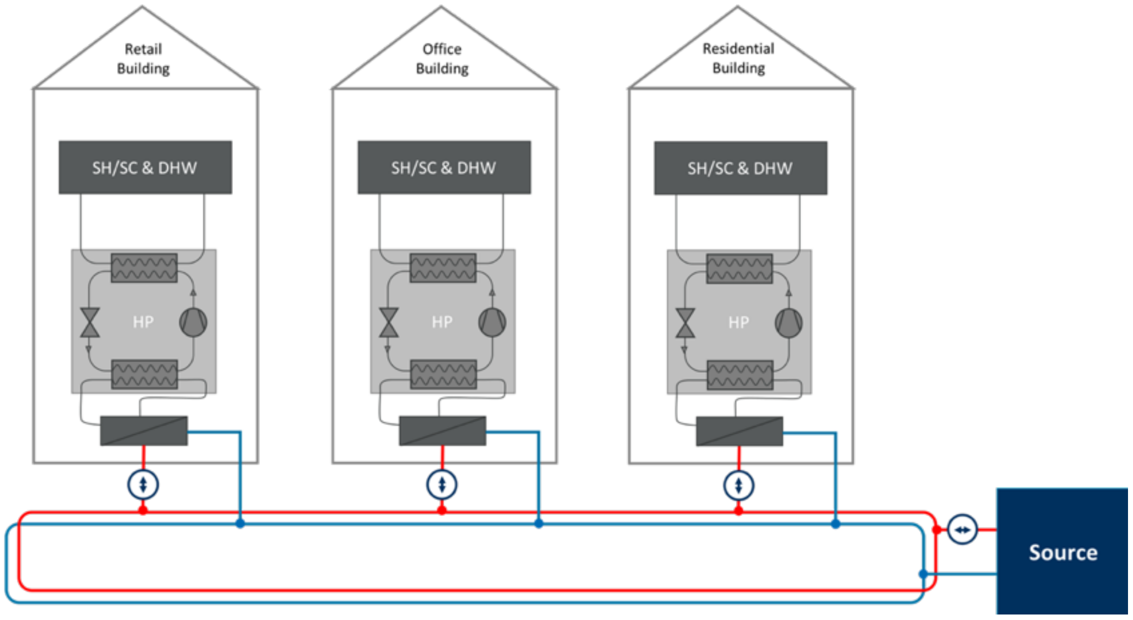


Figure 1.6: Diagram of simple 5GDHC system configuration

1.2 Objectives

The complexity of defining models for building energy modeling and the computational time required for modeling a single building can be significant. Modeling multiple buildings for a parametric analysis or optimization only exacerbates the computational time requirements (10s of minutes to several hours). The overarching objective of the ambient loop project is to determine the most efficient loop design, including the network topology (i.e. which buildings should be connected and how), the heat sources (and sinks) to connect, and the optimal 5GDHC operating temperature. Figure 1.7 shows seven different configurations of how three buildings could be connected to a district energy system [9]. The gray circles are non-connected buildings and the red circles are connected to the district energy system. The full combinatorial of three buildings and one district energy system being connected in all the various configurations yields a total of 53 combinations. As expected, the number of configurations grows quickly when the number of buildings increase (8 buildings and a district heating system yields over 65 trillion combinations).

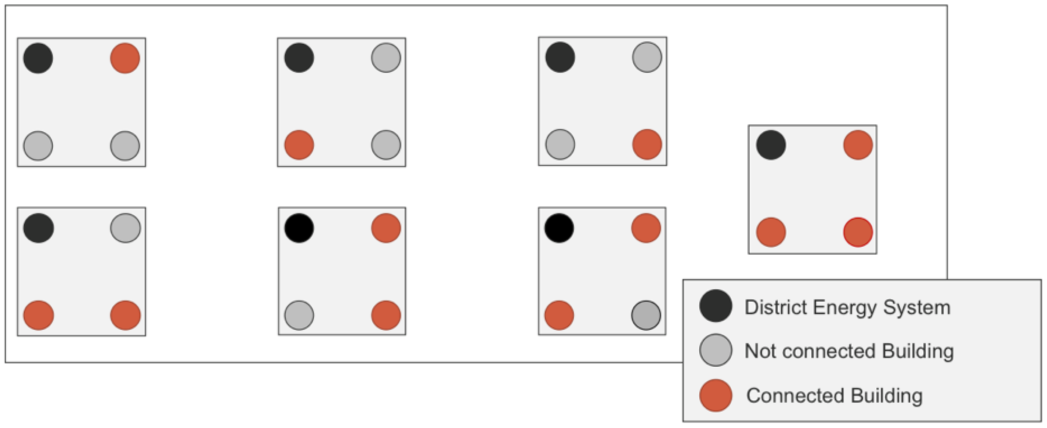


Figure 1.7: Seven examples of three buildings connected to a district energy system

To further complicate the modeling of 5GDHC systems, the temperature of the water does not remain constant between buildings. Some system topologies contain a parallel flow arrangement using a uniform header that supplies the same water temperature for each building, but that is a design consideration. As a result, the inlet water temperature changes for each building during each time step for each layout. If the return temperature changes for each building, then the models would need to be run in series resulting in even longer runtimes.

Based on the fast-increasing complexity of adding more buildings to the system, the use of a quick and reasonably accurate building energy model becomes critical. The computation time for a single scenario with three buildings Using conventional building energy modeling would be roughly five minutes. The runtime is not prohibitive if the layout is fully prescribed and only contained a handful of buildings; however, the layout, number of buildings, and operating conditions may not be known and is typically a varying parameter in the design.

The overall objective of this thesis is to:

(1) provide a framework for generating and evaluating various reduced order models for ambient loop analysis,

(2) estimate building energy loads based on various covariates (such as inlet ambient loop supply temperature, date and time, and various building characteristics),

(3) evaluate the performance of the reduced order models including creation time, load time, execution time, and accuracy.

In the context of 5GDHC systems, the existence of a reduced order model to determine building loads based on various covariates would ultimately allow for several questions to be answered that would have otherwise been simulation time prohibitive such as:

(1) Which buildings should be connected to the 5GDHC system?

(2) What is the best network topology for the 5GDHC?

(3) What should be the optimal network topology if new heating or cooling sources are added to the loop?

(4) If operating schedules change for a building, should the building still be connected?

(5) How should we control the buildings?

(6) What should the current supply temperature of the 5GDHC loop be at a specific date and time?

1.3 Thesis Organization

The thesis is organized into five chapters, this Introduction, a Literature Review discussing district energy systems, building energy modeling, and reduced order modeling research. The next section, Methodology, explains the creation of the building energy models and the reduced order models. The section also presents the results of the building energy models and the results of each individual reduced order model. The Results and Discussion chapter compares the results of the various reduced order models, and the Summary and Conclusions presents the effectiveness of using reduced ordered models in lieu of building energy modeling. Finally, the last chapter contains Future Work discussing the possible expansion of this work.

Chapter 2

Literature Review

There is a need to understand the impact on energy, health, and comfort as district energy system designers and engineers push the loop temperatures to near ambient temperature. Also, as energy efficient urban design is becoming a goal for many growing cities, there needs to exist various analysis tools to evaluate options surrounding best system configurations, buildings to connect, temperature optimization, and system control. This chapter will discuss the state-of-the-current on district energy systems, building energy modeling, and the current use of reduced order models for evaluating building energy use.

2.1 District Heating and Cooling Systems

2.1.1 History

District heating systems have been around for hundreds of years; however, modern systems started to appear in the United States in the late 1870's [10]. Some of the first district energy systems in the United States are still in operation, such as Denver, Colorado's district heating system, which was installed in 1880 [10] and is recognized as the oldest continuous operating district heating system in the United States. The original systems are considered first generation (1G) steam systems with supply temperatures running between 100-200°C. Even in the 1G systems, there was a desire to reduce maintenance costs and to save energy. In the case of Denver, the main distribution lines were installed in hollowed out wood trunks with asbestos lining to reducing line losses.

Second generation heating systems starting appearing in the 1930's with hot water supply temperatures around 100°C. The trend of reducing the hot water supply temperature has continued, resulting in energy savings due to reduced transmission losses and the ability to add new heating sources to the district system. Figure 2.2 and Figure 2.1 show the progression of district heating and cooling systems over the last one and a half centuries [8].

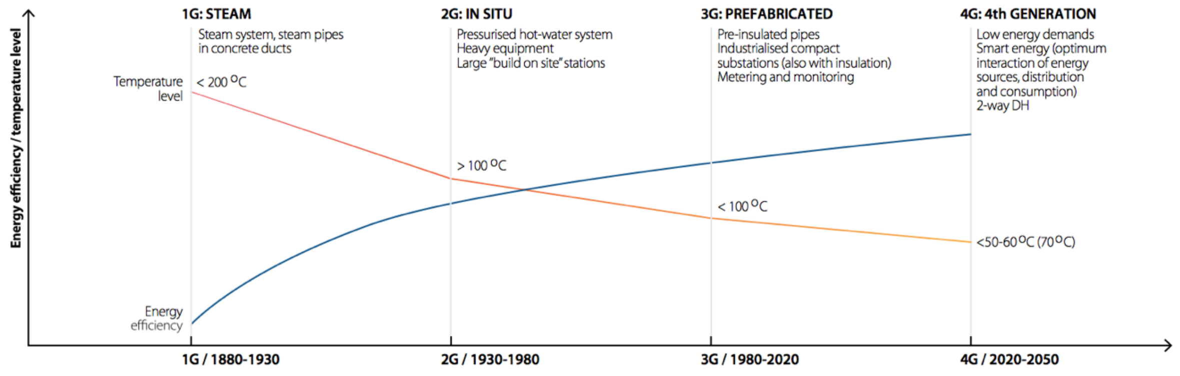


Figure 2.1: Progression of district energy temperatures

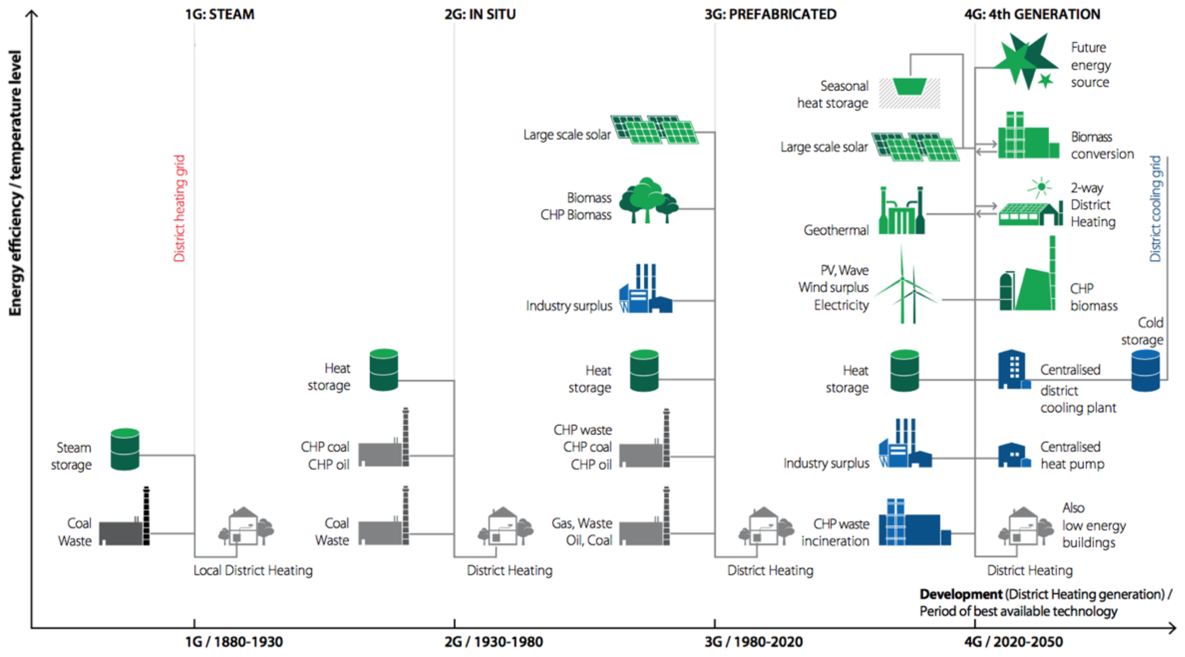


Figure 2.2: Progression of district energy systems

Von Rhein points out that the third generation district heating systems were created during the transition from oil-based heating systems to coal-based systems due to the oil crises [9]. Not only did the supply fuel change, but buildings were becoming more energy efficient allowing for the supply temperatures to drop to between 70°C and 100°C. Finally, fourth generation district heating and cooling systems have recently emerged and are typically considered the state-of-the-art. These systems have lower temperatures, typically less than 70°C.

As the temperature of district heating systems has decreased, the ability to use new lower-grade heat sources has emerged. The original heat sources were steam boilers, which evolved over time to use combined heat and power (CHP), and eventually geothermal, biomass, waste incineration [8], and solar thermal [11]. The approach to further lower the loop temperature enables even more heating sources such as wastewater heat recovery [1] and data center waste heat [12]. Unfortunately, there have been few advances in cooling sources. Absorption chillers can be used if there is high-temperature waste heat, otherwise, heat rejection for district cooling is typically accomplished with mechanical cooling, cooling towers, and/or cooling ponds [13, 14].

2.1.2 District Heating Energy Saving Potential

Gils, et al. [15] conducted a penetration study for district heating focusing on the existing population centers in the United States and forecasting energy consumption, building counts, length of pipes, and various other variables. Based on the research, it is estimated that 43% of the heating demand for residential and commercial buildings could be met through district heating. The effectiveness of district heating is climate specific and population density sensitive. As shown in Figure 2.3, the northeastern states with larger population densities can achieve higher district heating potential. For example, District of Columbia, New Jersey, and New York can all achieve over 70% of their needed heating energy through district heating; whereas warmer southern states (e.g. Louisiana, Florida, etc.) or low population states in the north (e.g. Montana, North Dakota) have low district heating potentials.

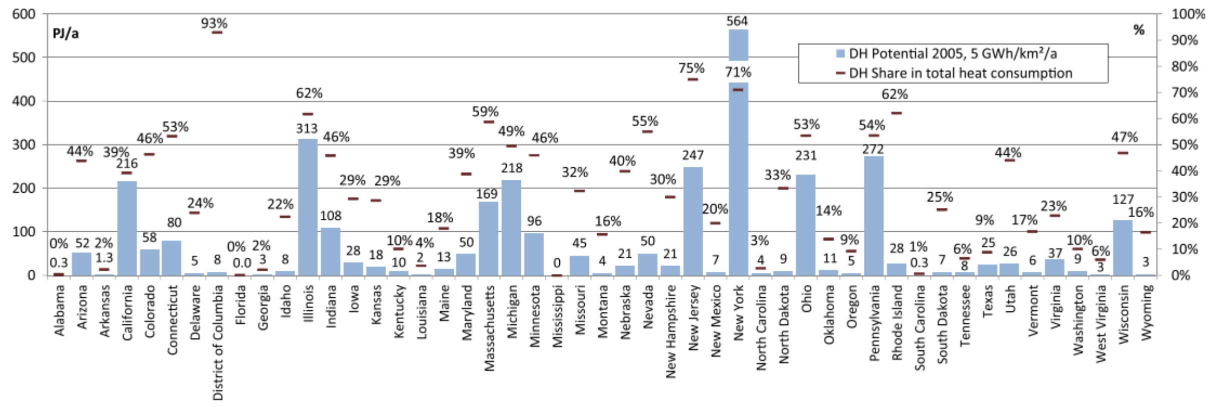


Figure 2.3: District heating potential by state

2.2 Building Energy Modeling

Building energy modeling (BEM) has historically been a time-consuming task requiring thousands of inputs and taking minutes to hours to simulate a single complex scenario to produce annual results [16]. The complexity of BEM is often seen as acceptable when designing complex systems such as advanced heating, ventilation, and air-conditioning (HVAC) systems or integrating multiple systems across a building (e.g. HVAC, photovoltaics, district heating and cooling, ambient loops, etc.). In the last several years, tools have been developed to facilitate BEM and, more importantly, to add parametric analyses to conventional workflows. The ability to easily run parametric analyses has caused a shift in the users' ability from simply learning how to run individual models to now using large parametric results to perform more advanced analyses, such as, sensitivity analyses, optimization, and meta-modeling. These meta-models, or reduced order models, can run faster depending on the selected covariates than a running fully defined building energy models.

Building energy modeling is a large domain with many available tools. The International Building Performance Simulation Association (IBPSA) lists 179 building-related software tools [17]. The list contains tools for various topics including whole building energy modeling, lighting, weather, airflow, life cycle analysis, photovoltaics, and various others. Fifty-seven of the tools are listed as whole building energy modeling tools and are the main focus of building energy modeling as it pertains to 5GDHC and reduced order modeling.

Foucquier defines three types of models: physical (white box), statistical (black box), and hybrid (gray box) models [18]. Physical models (for buildings) are often developed by engineers which create bottom-up component models exposing key user inputs that can be varied by a competent user. There are two main subcategories of physical models: compiled and equation-based. Compiled physical models include DOE-2 [19], TRNSYS [20], ESP-r [21], IES [22], and EnergyPlus [23]. The compiled physical models often have the ability to add in additional components defined in modules or simplified equations. Some examples of equation-based physical models include Engineering Equation Software (EES) [24] and Modelica [25]. Equation-based physical models are flexible but often require a more experienced and domain educated user. These models compile the equations at runtime and the compiled equations are then used for simulation. Once a set of equations has been compiled, then the simulation can be run again with different variable inputs without the need to recompile.

Statistical models (for buildings) use statistics or mathematical equations to link inputs (covariates) to a specific set of outputs. An example of a statistical model is the development of a 4-parameter change point model [26]. The model uses training data to estimate the values of the four parameters, see Eq. 2.1 where E is the expected energy consumption, a is the slope offset, b_c is the cooling slope, b_r is base level slope, t is change point temperature, and T is the average daily temperature. Once the model is fit, then daily cooling energy consumption can be estimated based on the outdoor dry-bulb temperature. In general, statistical models require a clear understanding of the covariates, the sensitivity of the covariates, and the desired outputs. Statistical models can also require a large amount of training data in order to perform the parameter estimation or to feed into machine learning algorithms.

$$E = a + b_c(T - t)^+ - b_r(t - T)^+ \tag{2.1}$$

Based on the type of data required to build the models and how the models run creates another category: bottom-up vs. top-down [27]. A bottom-up model aggregates the results of various components based on the user inputs. A bottom-up model has the ability to report the state of any internal variable in the model. This is a major advantage of the bottom-up model and allows for easy estimation of end uses based on differing building designs. In contrast, a top-down approach functions on the already aggregated dataset, some of which was used to train the model. In reality, bottom-up physical models are an aggregation of other physical models or statistical models, thus are hybrid models. A comparison of the various models is below.

Bottom-Up Physical Model (White Box):

- Advantages
    - Ability to both represent and analyze results of physical systems (e.g. package single zone HVAC systems vs. water-to-water heat pump HVAC systems).
    - Models can provide a more accurate representation of building performance with varying temporal resolution.
    - Building controls can be evaluated.
    - Missing data can be inferred and/or defaulted.
- Disadvantages
    - Requires a large amount of physical data on the building systems including fully defined building geometry.
    - Although it is an advantage that missing data can be inferred and/or defaulted, typically the missing data are defaulted to minimum code and standard value , or defaulted to an unknown/unreasonable value.
    - Simulation runtime can be long depending on model complexity.

Top-Down Statistical Model (Black Box):

- Advantages
  - ∗ Models can run quickly.
  - ∗ Building does not need be fully defined.
  - ∗ Can be part of an aggregate model (i.e. can be referenced by a bottom-up physical model).
- Disadvantages
  - ∗ Can require a large amount of training data to fit the statistical model. Training data needs to cover all the possible conditions that is expected to be encountered.
  - ∗ The results are not physical and must be interpreted.
  - ∗ Various models may be needed to evaluate multiple outputs.

2.3 Reduced Order Models

A reduced order model (ROM) is the general term used in this thesis to describe a black box statistical model. Reduced order models could also be described as data-driven models, metamodels, response surface models, or surrogate models. Although there are specific (and subtle) differences between the models listed, the term reduced order model encompasses various features found in the similarly named approaches above.

Machine learning has become ubiquitous in many fields of study and more recently in the building sciences. As machine learning becomes more approachable, the ability to use reduced order models in optimization, uncertainty analysis, energy conservation measure selection for building retrofits, and overall building energy performance predictions can help provide results significantly faster than running fully defined physics-based building energy models, all the while, maintaining sufficient accuracy.

Modern-day physics-based building simulation engines, such as EnergyPlus are excellent at providing a single, deterministic result based on the input described. However, a single result is rarely the only result that falls within the error bounds of the many different possibilities of inputs. These engines are notorious for having long runtimes, making the exploration of large parametric spaces intractable. In addition, building energy modeling, in its true form, has difficulty abstracting higher level input parameterizations. For example, changing the window-to-wall ratio of a single input file can be challenging due to the fact that a window is described vertex by vertex within the BEM input file. Due to the runtimes, difficulty obtaining empirical data, and the challenge of running high-level parametric analyses, reduced order models for building analyses have been successfully deployed for various reasons including:

- faster iteration time during the building design and engineering phase,
- model reduction to limit the number of inputs (further reducing modeling complexity)
- ability to run analyses real-time on web servers,
- more time efficient navigation of large parameter spaces, and
- running uncertainty analysis on inputs.

One area where reduced order models have been successfully deployed is in building benchmarking and energy asset ratings. Many jurisdictions around the world have started requiring asset ratings, and in some cases performance ratings, to be calculated for individual building [28]. The methods of performing the ratings vary widely; however, most methods require fast and accurate approximations of the desired results (such as the building's total annual energy consumption).

The U.S. Department of Energy developed the Asset Score Tool to perform an asset rating calculation. The main version of the asset rating is performed by a fully described BEM, however, Goel, et al., also used a random forest model to provide a quick approximation of a building's asset score based on a few high-level inputs [29]. Also, Brazil's building energy labeling program investigated the use of artificial neural networks (ANN) for the building shell labels [30]. The neural networks performed well (within 16%) based on the input variables selected; further research was able to reduce the error to 0.6% [31]. The model inputs were high-level geometric values such as the linear distance of interior wall, shadow angles, window-to-wall ratios, thermal transmittance, thermal capacities, and basic operational data such as schedules and equipment power densities.

Load-based reduced order modelings methods have been investigated in several cooling dominated climates. Li et al. investigated the use of various types of neural networks and support vector machines (SVM) to approximate the building's hourly cooling load [32]. The results showed that the SVM model performed better than the neural network models. The inputs of the model were strictly climate data including a 2-time step lag. Kalogirou et al. used ANN to develop a surrogate model to evaluate building thermal behavior and to calculate the building energy consumption over a 24-hour period[33]. For their use case with very specific data, the ANN performed well (R^2 of 0.9991). Models have also been developed for approximating the building's heating loads [34] using various regression techniques such as linear and polynomial regressions.

Aijazi et al. used a surrogate model to evaluate the total building energy consumption comparing random forests, linear regressions, and radial basis functions [35]. The runtime of the surrogate models were roughly five orders of magnitude faster than the detailed EnergyPlus simulations. Other surrogate model development has occurred using various techniques such as equation-based simplification [36, 37], unsupervised learning [38, 36] such as clustering and neural networks [39, 40], and supervised learning such as linear (and nonlinear) regression [41], regression trees, and classification [42]. Lastly, there are several kernel-based supervised learning methods such as kernel ridge regression and support vector machines which have been employed for building energy and comfort analysis [43, 44].

Based on the prior research, reduced order models have been used to accurately represent detailed building simulation results. The type of reduced order model is heavily dependent on the use case for the model. Overall, the models tend to run faster than the detailed simulations while providing similar accuracy. The literature review justifies the hypothesis of using reduced order models to more quickly model ambient loop systems while maintaining model accuracy.

Chapter 3

Methodology

The first application of the proposed reduced order models is to develop an analysis tool to evaluate ambient loop network topologies. The network topology analysis tool [9] was developed in parallel to this thesis. The development of the tool involved investigating the ability of two different simulation engines to model buildings and district systems, EnergyPlus [23] and Modelica [25]. Ultimately, Modelica with the Modelica Buildings Library [45] was chosen to model the network systems due to its ability to handle pressure driven flow, advanced controls, and its integration capabilities of Python-based scripts. EnergyPlus was chosen to model the building physics and the data generation engine for the reduced order models. The reduced order model results were loaded into Modelica to approximate building loads based on specific covariates such as date and time, inlet temperature, and building operating conditions. This thesis focused on the development and validation of the baseline models, parametric simulations, and the development of the reduced order models.

The development of reduced order models require both a bottom-up physical model and a top-down statistical model. The initial step was to run a parametric analysis of physical models using EnergyPlus. The results of the parametric analysis were then used to generate the statistical model, see Figure 3.1. The line in the figure shows the separation between the physical model and the statistical model. The simulation results from the physical model are post-processed and passed to a Python library (termed the ROM Framework) to develop and evaluate various reduced order modeling methods.

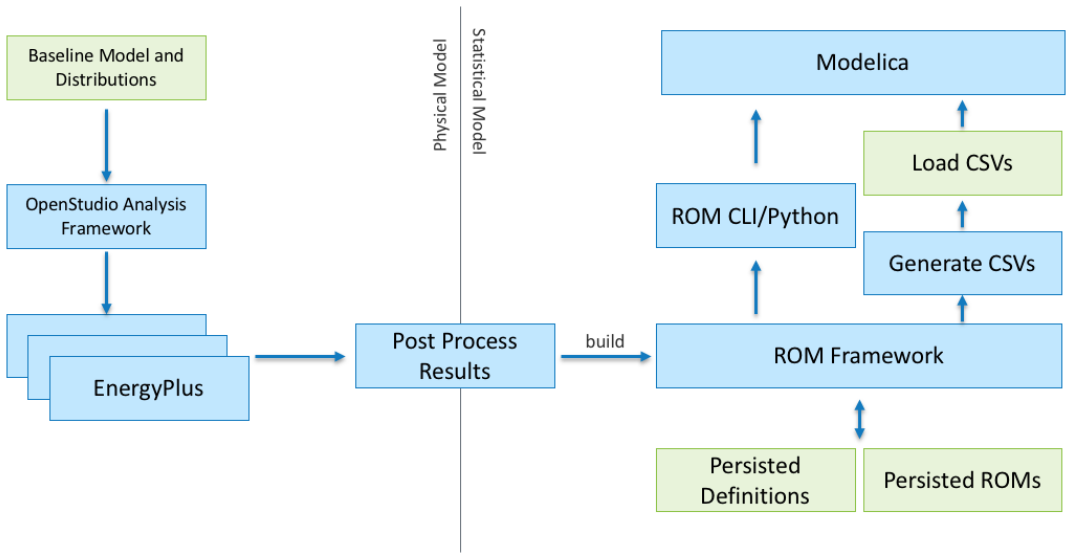


Figure 3.1: Development and integration of ROMs in 5GDHC Modeling

The methodology is divided into two sections, Building Energy Modeling Data Generation and Reduced Order Modeling Development. Each section discusses the background, procedure, and results of generating the respective data sets.

3.1 Building Energy Modeling Data Generation

Empirical data of 5GDHC loop performance and buildings attached to the loop are not readily available. Even if empirical data were available, the data would be for very specific building designs, network system designs, and operating schemes. A comprehensive dataset that includes the expected ranges of all operating conditions is needed in order to generate the statistical models. As a result, building energy models were developed and used to generate the comprehensive data set. This section describes the background, procedure, and discusses the results of generating the building data.

3.1.1 Background

Building energy modeling (BEM) was used to generate the datasets for developing the reduced order models. The ability to use BEM allows for a larger than typical parameter space with varying building types, configurations (e.g. square footage, interior gains), and building operations (e.g. schedules, inlet temperatures, etc).

EnergyPlus was chosen as the physical modeling engine. EnergyPlus is a whole building energy analysis tool that performs detailed sub-hourly simulations [23]. The input to EnergyPlus is a text file in the input description format (IDF). The file fully describes the building in pain-staking detail. EnergyPlus is typically run as a single input/single output simulation engine, meaning that the IDF file is a single building configuration and results in a single set of results, all related to the input. Furthermore, EnergyPlus is a bottom-up physical model of a building, thus, the models require a large amount of data to run simulations. The type of data required includes everything from the location of the building, geometric configuration including its walls, windows, slabs, roofs, and the detailed description of the HVAC system including details on the system performance. The amount of data required to run a simple single small office building can easily be several thousand inputs.

OpenStudio [46] is an analysis tool that makes generating EnergyPlus input files easier. OpenStudio has a custom input format (similar to the IDF format) that extends EnergyPlus and enables more advanced modeling workflows through its user interfaces and application programming interfaces (APIs). For modeling, OpenStudio has several features that make modeling easier, more accurate, and repeatable including:

- *Modeling Building Spaces*: EnergyPlus's canonical modeling unit is a thermal zone, that is, the physical boundary in a building sharing a common heating and/or cooling setpoint. Building energy standards and architectural models typically include the concept of rooms or spaces; therefore, OpenStudio enables the user to define spaces with specific loads and characteristics. This enables codes and standards to be more easily applied due to how standards typically reference space types. Spaces are aggregated into thermal zones for modeling in EnergyPlus upon simulation.

- *Application Programming Interface (API)*: Conventionally, generating the input files for modeling is the most time-consuming task and can easily result in many input errors. The entirety of the OpenStudio input description can be accessed from APIs in various languages including Ruby, Python, and C#. Ruby is the preferred and supported API language. A major advantage to APIs is the ability for validations to run during model assembly (e.g. making sure the right HVAC objects are added to the loop).

- *Advanced Daylighting*: OpenStudio enables the use of Radiance [47] for more advanced (and arguably more accurate) daylighting analysis. Advanced daylighting analysis was not leveraged in this analysis.

In addition to the modeling improvements listed above, there are several other workflow improvements in the OpenStudio ecosystem. Not only does OpenStudio make modeling a single building easier and more consistent, but it also provides an easy way to run parametric analyses, either through user developed software or through OpenStudio provided software. Below is a list of additional OpenStudio components designed to make modeling a single building more reliable and components designed to make modeling many buildings easier.

- *User Interfaces*: There are several user interfaces that are shipped with OpenStudio. The items that were used in this analysis are listed below along with the description.
  - *The Application* [46]: The OpenStudio Application allows users to view the model inputs. This is preferred over using a text editor to manually manipulate the model due to the ease of making mistakes in the text file.
  - *Parametric Analysis Tool (PAT)* [48]: PAT is used to generate and package the data needed to run parametric analyses, optimizations, and calibrations in the OpenStudio Analysis Framework [49]. PAT creates a zip file with seed models, weather data, and measures (described below) along with a JavaScript Object Notation (JSON) file that describes the algorithms to run. This is described more below in the Analysis Framework bullet.

    ∗ *Dview*: Dview is a user interface that was developed in conjunction with other tools (e.g. BEopt, System Advisor Model) to visualize the resulting time series data. Dview can load EnergyPlus SQL files, CSV files, and weather files.

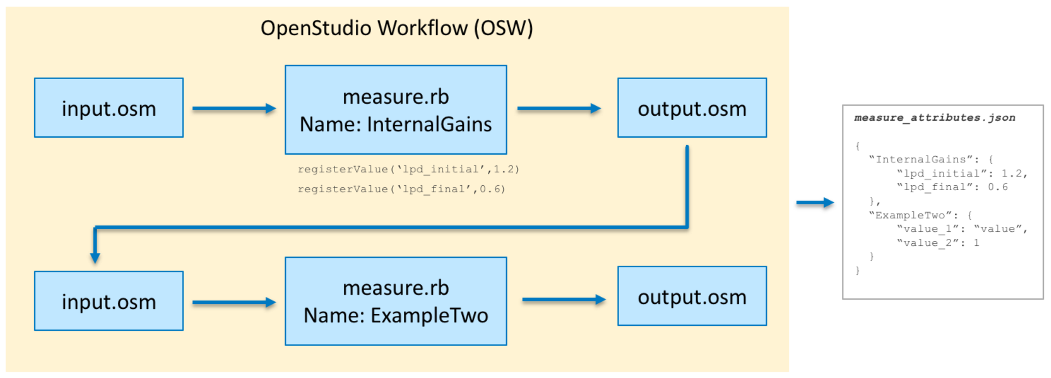
- *Measures*: Measures are Ruby scripts that can programmatically perturb a building energy model [50]. Measures can act on the OpenStudio model, the EnergyPlus model, or be used as a reporting measure to access and post-process results. This is described in more detail in the Measures section.
- *Standards*: Standards [51] is a Ruby library enabling users to a) select a prototype model based on building vintage, type, and climate zone, or b) apply codes and standards to an existing building based on one of the following standards: ASHRAE 90.1, Title 24, National Energy Code of Canada, Energy Conservation Code (India), or the International Construction Code.
- *Prototype Buildings*: The U.S. Department of Energy publishes prototype buildings [52] in EnergyPlus format which represent models suitable for sector analyses and impact studies. OpenStudio has these same models available in the OpenStudio format.
- *Analysis Framework*: The Analysis Framework is a web-based application that allows for parametric analyses, optimizations, and calibrations to be run on local or cloud-based infrastructure [53]. More information is in the Parametric Analysis Tool and Analysis Framework section.

3.1.1.1 Measures

The 5GHDC analysis leveraged the concept of OpenStudio measures; therefore, a more descriptive discussion of measures is warranted. As briefly described above, a measure is a Ruby script that runs within an OpenStudio workflow. The measure has a set of arguments as inputs in the form of simple computational types such as strings, integers, or doubles. These arguments become internal variables within the measure and are used to programmatically perturb the building energy model. Multiple measures can be chained together to generate linear workflows of model perturbations. OpenStudio measures have no concept of dependency, other than data in and data out dependencies, for example, if a downstream measure relies on an upstream measure to set a specific value in the building energy model, then there is nothing to ensure that the upstream measure sets the value. It is important that measures are developed, chained together, and tested with care to prevent such errors.

There are three specific places in which measures are executed in the OpenStudio workflow: 1) post seed model loading, 2) after model translation to EnergyPlus IDF, and 3) after EnergyPlus simulation completion. The OpenStudio Workflow library [54] was designed to chain these events together in order to systematically and deterministically generate the same simulation file every single time. In addition to running measures, the OpenStudio workflow library is responsible for keeping track of log data including the user defined attributes during the running of each measure. These measure attributes can be simple log messages or structured data that can be stored for use in subsequent post-processing.

An example of a structured measure attribute would be to store the average lighting power density of the building before and after a measure runs. All structured measure attributes are accessible after simulation completion in a JSON format. In this provided example, having the value before and after allows for model validation, variable extraction, and simple comparisons. For example, setting the initial and final values of arguments in the measure attributes is very helpful when the inputs of measures are percent increases from the already defined values in the building energy model. Since the value is not defined, the measure is able to calculate the average value (e.g. LPD, square footages) and return the value for later use by the statistical models. Figure 3.2 graphically shows how a measure consumes an input, saves structured measure attributes, and passes the results to another measure.



```
registerValue('lpd_initial',1.2)
registerValue('lpd_final',0.6)
```

```
measure_attributes.json

{
  "InternalGains": {
      "lpd_initial": 1.2,
      "lpd_final": 0.6
  },
  "ExampleTwo": {
      "value_1": "value",
      "value_2": 1
  }
}
```

Figure 3.2: OpenStudio workflow and measure attributes

Both the measure name and the measure attributes are user-defined; therefore, it is important to keep the values consistent throughout the analysis. For example, in order to vary the inlet temperature of a district system between 15 and 25°C required the development of a measure to set the loop temperature. The measure was named Ambient Loop Temperature Setpoint and had two arguments, the setpoint temperature and the design delta. Measure arguments are automatically added to the measure attributes as snake cased variables (e.g. Variable Name becomes variable_name) and in this case the measure attributes contained the "setpoint_temperature" and the "design_delta". The measure name is prepended to the measure attributes for the final variable name. In this example, the full measure variables of interest were named ambient_loop_temperature_setpoint setpoint_temperature and ambient_loop_temperature_setpoint design_delta. It was important to understand the naming convention as the procedures below rely on having the data in very specific formats.

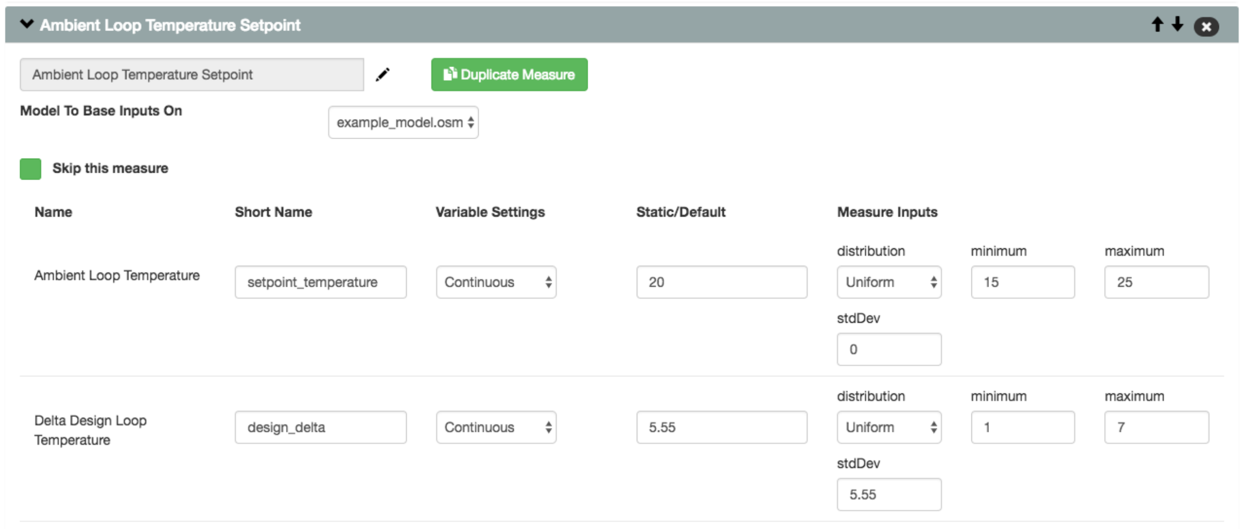
An example of a measure in Ruby is shown in Figure 3.3. The measure is defined as a Ruby class and inherits its functionality from the OpenStudio ModelMeasure library. The measure has a section to define its human-readable name and its arguments. In this case, the measure is expecting a single input (or argument) named setpoint_temperature. Based on the convention above, the measure identifiable name for post-processing is ambient_loop_example_measure, and by default, it will have a variable named ambient_loop_example_measure.setpoint_temperature. The existence of the runner.registerValue will result in another variable named ambient_loop_example_measure lpd_average and will be set to 1.3 W/m^2.

```
class AmbientLoopExampleMeasure < OpenStudio::Measure::
   ModelMeasure
  def name
    return "Ambient Loop Example Measure"
  end

  def arguments()
    args = OpenStudio::Ruleset::OSArgumentVector.new

    setpoint = OpenStudio::Ruleset::OSArgument.makeDoubleArgument(
       "setpoint_temperature", true)
    setpoint.setUnits("Degrees Celsius")
    setpoint.setDefaultValue(20)
    args << setpoint

    return args
  end

 def run(model, runner, user_arguments)
    super(model, runner, user_arguments)

    return false unless runner.validateUserArguments(arguments(
       model), user_arguments)
    setpoint = runner.getDoubleArgumentValue("setpoint_temperature
       ", user_arguments)

    // manipulate building model

    lpd_average = 1.3  # set to some value for demostration
       purposes
    runner.registerValue('lpd_average', lpd_average, 'W/m2')

    return true
  end
end

AmbientLoopExampleMeasure.new.registerWithApplication
```

Figure 3.3: Structure of an OpenStudio measure

3.1.1.2 Parametric Analysis Tool and Analysis Framework

PAT allows for a user to easily define an analysis (e.g. optimization, sampling, etc.) which is passed to the OpenStudio Analysis Framework for running. PAT (Version 2.6.0) was used to specify the seed model, weather files, add measures in a specific order, set the ranges and distributions of variables, and provide the communication to the OpenStudio Analysis Framework. Figure 3.4 shows an example measure input from PAT. The definition includes two continuous variables, setpoint temperature and design delta, for the ambient loop. The distributions were chosen as uniform, the reasoning will be discussed further in the subsequent sections.


Ambient Loop Temperature Setpoint

Ambient Loop Temperature Setpoint

Duplicate Measure

Model To Base Inputs On: example_model.osm

Skip this measure

Name	Short Name	Variable Settings	Static/Default	distribution	minimum	maximum	stdDev
Ambient Loop Temperature	setpoint_temperature	Continuous	20	Uniform	15	25	0
Delta Design Loop Temperature	design_delta	Continuous	5.55	Uniform	1	7	5.55

Figure 3.4: OpenStudio PAT configuration example

The OpenStudio Analysis Framework is a server-based application that can scale horizontally to run a large number of building energy models in parallel. The analysis framework has access to R (the statistical programming language) for running advanced algorithms. In this case, R is used to sample the parameter space using a Latin Hypercube Sampling (LHS) algorithm, treating all the parameter spaces equally [55]. LHS is effective at reducing the number of datapoints needed to represent the parameter space without impacting the overall mean and variance of the entire parameter space.

The OpenStudio Analysis Framework can be deployed locally, remotely on Amazon Web

Services (AWS), or on in-house workstations.

3.1.2 Procedure

The development of the datasets needed for evaluating the 5GDHC required knowledge of the various components of OpenStudio in order to adequately extend the components. Out of the box, OpenStudio does not provide libraries for modeling 5GDHC systems; however, both EnergyPlus and OpenStudio have the ability to model district heating and cooling systems with ease. This section will discuss the procedure used to extend, model, and post-process the data needed to generate a representative dataset of buildings connected to a 5GDHC system.

Overall, the steps needed to generate the building energy models included:

- Generate baseline buildings: one with a conventional HVAC system and one with an energy transfer station.
- Run one off analysis to represent the building not connected to a district system with conventional HVAC system
- Validate the baseline building to ensure its representational accuracy
- Extend OpenStudio standards library to include a water-based HVAC system
- Determine the perturbation variables and ranges
- Develop measures needed for perturbing baseline models to generate parameter space
- Develop reporting measures for evaluating and validating the building models
- Run the analysis
- Post process and validate the results

The process above is highly iterative and requires substantial time to ensure each step is running as expected. The subsections below will discuss the development of the baseline buildings, the modifications to the OpenStudio Standards library, and the development of the measures needed for the analysis.

3.1.2.1 Baseline Model Description

There were two prototypical buildings that were selected for the analysis: a small office and a retail building. The two buildings were selected to provide load diversity in the buildings that were to be modeled in Modelica's network topology simulation. The baseline buildings (also called seed models) were based on the prototype buildings as described in the Extending OpenStudio Standards subsection.

The characteristics of the buildings are shown in Table 3.1. Figure 3.5 and Figure 3.6 show geometry renderings of the small office and retail baseline buildings, respectively.

Table 3.1: Baseline building characteristics

Characteristic	Units	Small Office	Retail
Default Standards	N/A	90.1-2010	90.1-2010
Location		Golden, CO	Golden, CO
Construction			
Roof	m^2K/W	IEAD, R-3.7	IEAD, R-3.7
Exterior Walls	m^2K/W	Mass Walls, R-1.8	Mass Walls, R-1.8
Geometry			
Square Footage	m^2	511	2,294
Number of Stories	N/A	1	1
Window-to-wall Ratio	%	20	7
Skylight-roof Ratio	%	0	1
Internal Loads			
Average Lighting Power Density	W/m^2	9.7	16.1

Average Electric Power Density	W/m^2	6.8	5.23
Number of People	$\#/100m^2$	6	15
HVAC			
System Type	N/A	Air Cooled PTHP	PSZ-AC
Cooling Efficiency	COP	3.65	3.87
Heating Efficiency	COP / Eff	3.74	0.8
Distribution	N/A	Rooftop	Rooftop

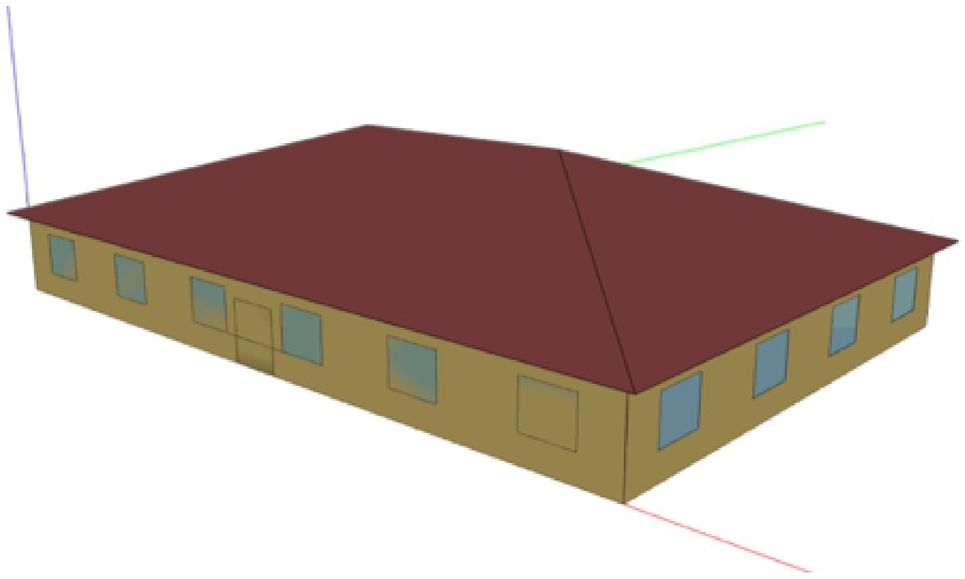


Figure 3.5: Rendering of the small office baseline model

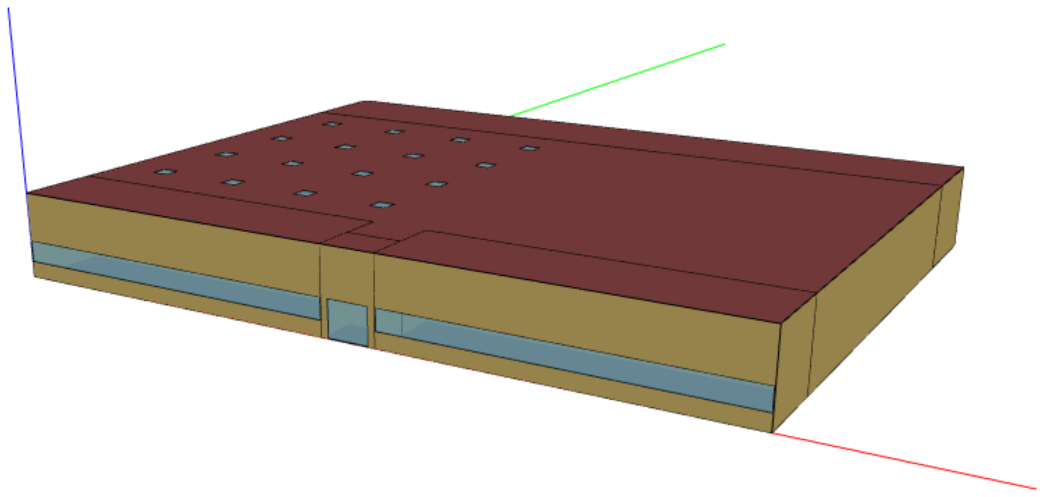


Figure 3.6: Rendering of the retail baseline model

The baseline buildings contained the prototypical HVAC systems as defined by the OpenStudio Standards library. The design of district heating and cooling systems require the connection of the HVAC systems to the district loop; therefore, another set of baseline buildings were developed with updated HVAC systems in order to connect the buildings to the district system.

The water-based building's HVAC systems were chosen based on a typical system that would be installed for a building the same size. Table 3.2 shows the equivalent water-based system based on the total square footage of the building. This lookup was created based on the increased use of water-based systems in commercial buildings due to the larger adoption of geothermal systems (e.g. ground source heat pumps) [56]. The table uses the generally accepted guidance of 350-400 ft^2 (32-37 m^2) per ton of cooling [57] to help provide the selection. In general, it is reasonable to assume that smaller buildings (less than 5,000 square meters) connected to district systems or geothermal systems will have distributed HVAC systems (either water-to-water heat pumps or water-to-air heat pumps). Conversely, larger buildings will typically have chiller/boiler based cooling/heating systems.

Table 3.2: Type of ambient loop primary system by building area

Recommended System Type	Building Area Ranges (m^2)
Distributed water-based heat pump system	70 to 5,250
Single Chiller / Boiler	2,100 to 7,000
Multiple Chillers / Boilers	4,200 to 52,500

The development of the HVAC systems is described in more detail in the Extending OpenStudio Standards section. The results of the baseline buildings, both with conventional systems and water-based systems are discussed in the results portion of this section.

3.1.2.2 Extending OpenStudio Standards

The OpenStudio Standards library is a Ruby gem to either generate a prototypical building or apply various existing codes and standards to a building energy model. The OpenStudio Standards library was extended to add water-based systems specific to ambient loop district energy systems. For the purpose of this thesis, the entire system used to connect a building to the ambient loop was termed the energy transfer station (ETS). Based on the type of building the ETS was modeled as either a water-to-water heat pump (WWHP) or a water-to-air heat pump (commonly named a packaged terminal heat pump - PTHP)

Figure 3.7 is a high-level diagram showing the primary water loop of a conventional HVAC system with a chiller. The heat rejection, in this case, is either air-cooled, cooling towers, or cooling ponds. The extensions to the OpenStudio Standards library replaces the heat rejection side of this model with the ambient loop and replaces the chiller with the water-based system.

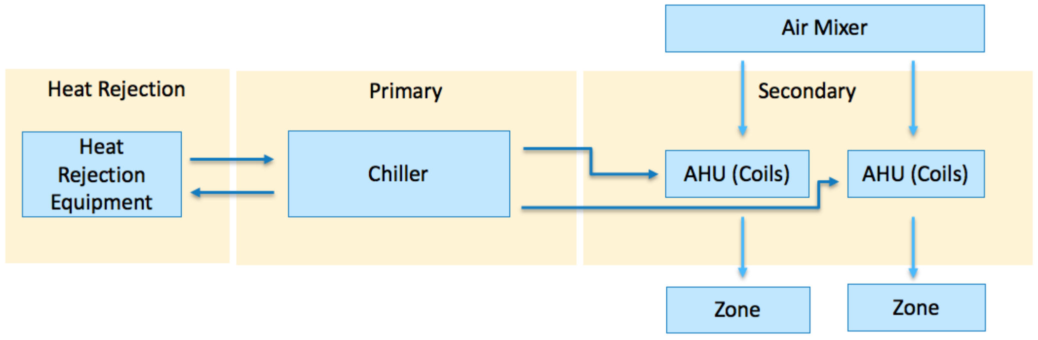


Figure 3.7: Primary water loop in conventional building

Figure 3.8 shows the primary water loop of the building after swapping the system with an ETS. The heat rejection (assuming cooling mode) is connected to the ambient loop through a heat exchanger. The heat exchanger transfers the heating or cooling to the primary input to the WWHP. The WWHP then pumps the chilled water (again, in cooling mode) to the terminal unit coils where the primary air loops can distribute the cooled air.

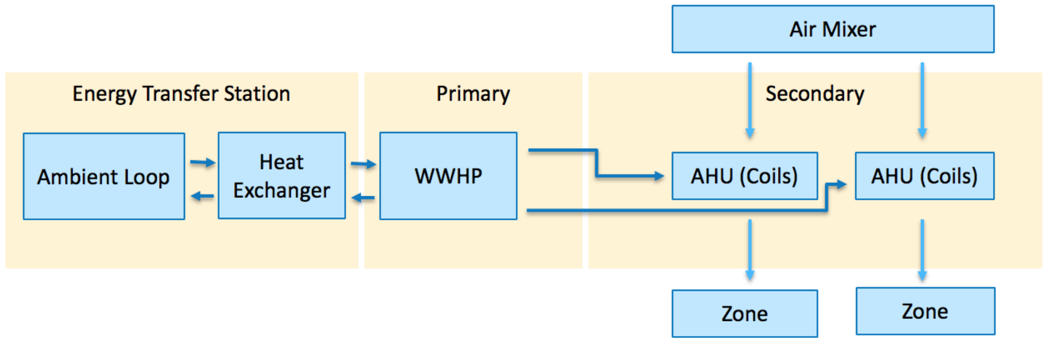


Figure 3.8: Energy transfer station in the ambient loop connected building

EnergyPlus and OpenStudio do not have the concept of an ETS but it could be modeled as a district energy system with lower temperatures and a heat exchanger. Note that the ETS heat exchanger was considered an ideal heat exchanger for the purpose of this project. Adding a physical heat exchanger model using either EnergyPlus or Modelica would increase the accuracy of the model by decreasing the effectiveness of the district heating and cooling system, thus, increasing the heating and cooling electricity needed to meet building setpoint. The ability to regress the data would not be impacted.

The ability to add an ETS to an existing building model was directly added to the OpenStudio Standards library to leverage the pre-existing helper methods such as adding the water loop, chilled water loop, and setting the default values based on the ASHRAE standard being referenced. The code-block in Figure 3.9 shows the method in detail. Note the use of the "get_or_add" helper methods. Since the ETS was added to an existing building model, most of these methods simply return the already existing objects from the model. The developed method only added energy transfer stations for water-to-air and water-to-water heat pumps; however, the method can be expanded in the future to allow for other types of water-based HVAC systems such as chilled water distribution systems (replacing chillers).

```
# Add Energy Transfer Station
# Used for Ambient Loop connections.
def add_energy_transfer_station(primary_system, zones)
    return true if zones.empty?

    if primary_system.downcase == 'water-to-air heat pump'
        condenser_loop = get_or_add_ambient_water_loop

        add_water_source_hp(condenser_loop, zones, ventilation=
           true)
    elsif primary_system.downcase == 'water-to-water heat pump'
        ambient_loop = get_or_add_ambient_water_loop
        hot_water_loop = get_or_add_hot_water_loop('HeatPump',
           ambient_loop)
        chilled_water_loop = get_or_add_chilled_water_loop('
           90.1-2010', 'HeatPump', true, ambient_loop)

        add_vav_reheat('90.1-2010',
                        sys_name=nil,
                        hot_water_loop,
                        chilled_water_loop,
                        zones,
                        hvac_op_sch=nil,
                        oa_damper_sch=nil,
                        vav_fan_efficiency=0.62,
                        vav_fan_motor_efficiency=0.9,
                        vav_fan_pressure_rise=OpenStudio.convert
                           (4.0, 'inH_{2}O', 'Pa').get,
                        return_plenum=nil,
                        reheat_type='Water')
    else
        raise "Unknown primary system '#{primary_system}' for #{
           __method__}"
    end
end
```

Figure 3.9: Energy transfer station in ambient loop connected building

After the ETS methods were added, the next step was to create the measures to chain all the required parts together. The next section discusses the development of these measures.

3.1.2.3 Measure Development

Several new measures were written to perform the model perturbations. There were three succinct energy modeling tasks to be run: 1) baseline simulation, 2) baseline simulation with ETS, and 3) parametric analysis with varying building inputs. These three tasks were repeated for the two building types (small office and retail building). The three tasks built on measures used from the previous tasks and can be seen in Figure 3.10. As shown in the figure, the three tasks in green have the same workflow but with different measures enabled. The baseline simulation only required instantiating the prototype building and changing the building location. The baseline simulation with ETS required adding in the ETS and setting the ambient and hot water loop temperatures. Lastly, the parametric analysis required adding in a couple additional measures to change building loads. Note that the three modeling tasks used the same reporting measures (the orange blocks in the figure). Having a similar workflow and using the same measures for the modeling tasks is advantageous as it reduces errors since the tasks are all dependent on the results of previous tasks.

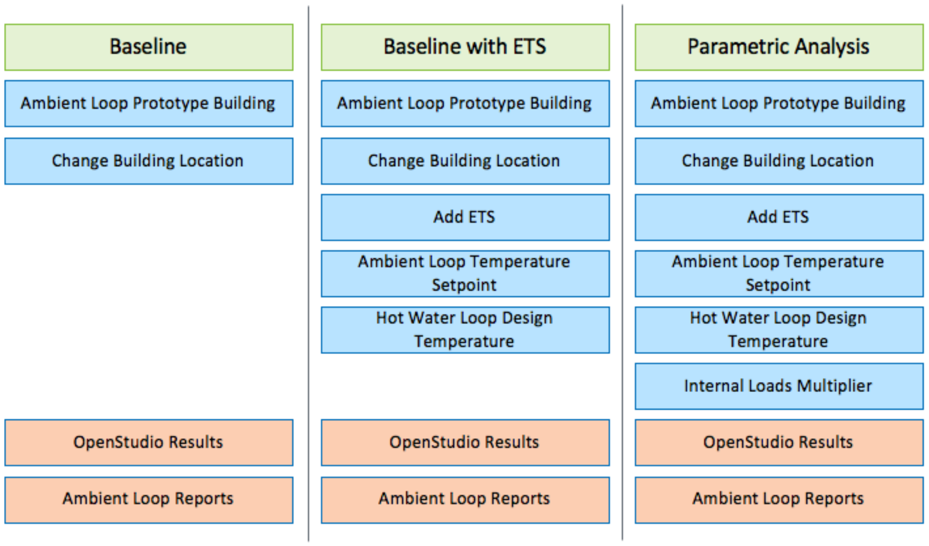

Baseline
Ambient Loop Prototype Building
Change Building Location
OpenStudio Results
Ambient Loop Reports
Baseline with ETS
Ambient Loop Prototype Building
Change Building Location
Add ETS
Ambient Loop Temperature Setpoint
Hot Water Loop Design Temperature
OpenStudio Results
Ambient Loop Reports
Parametric Analysis
Ambient Loop Prototype Building
Change Building Location
Add ETS
Ambient Loop Temperature Setpoint
Hot Water Loop Design Temperature
Internal Loads Multiplier
OpenStudio Results
Ambient Loop Reports

Figure 3.10: Order and list of measures for simulations

The measures above have various arguments. Table 3.3 shows the same list of measures along with the arguments and example argument values.

Table 3.3: List of measures and arguments

Measure Name	Arguments	Example Argument Value
Ambient Loop Prototype Building (C.1)		
	Building Type	SmallOffice
	Template	90.1-2010
	Climate Zone	ASHRAE 169-2006-5B
Change Building Location (C.2)		
	Weather File Name	Golden, CO
	Climate Zone	Lookup from .stat file
Ambient Loop Add ETS (C.3)		
	No Arguments	
Ambient Loop Temperature Setpoint (C.4)		
	Loop Temperature	20
	Delta Design Loop Temperature	5.55
Hot Water Loop Design Temperature (C.5)		
	Hot Water Temperature	60
Internal Loads Multiplier (C.6)		
	LPD Multiplier	1
	EPD Multiplier	1
	People Multiplier	1

The reporting measures typically do not contain arguments (but they can) and are used to post-process the results of a single simulation. The analysis used two reporting measures, the OpenStudio Results and the Ambient Loop Reports (see C.7). The OpenStudio Results measure is packaged with OpenStudio and used to report basic diagnostics on models; however, the Ambient Loop Reports measure was written in order to post-process the results into a format that can be consumed by the reduced order modeling framework. The Ambient Loop Reports measure queries the EnergyPlus simulation results for the needed hourly reporting variables (e.g., district cooling inlet/outlet temperature, district heating inlet/outlet temperature, cooling electricity, heating electricity, etc.) and converts the results into hourly values over the year (i.e., an 8,760). These results are saved to a CSV file and saved for later processing by the ROM Framework.

Table 3.4 lists all output variables of interest from the simulations. These outputs were created for all three modeling tasks (baseline, baseline with ETS, and parametric analysis). The "constant" items in the table are building characteristics (e.g. the building's average lighting power density). These values are held constant for every time step and simply stored for use in the reduced order modeling framework.

Table 3.4: Building energy modeling output variables

Variable	Time step	Units
Date and Time, Day of Week	Hourly	N/A
Outdoor Relative Humidity	Hourly	Percent
Outdoor Drybulb Temperature	Hourly	C
District Heating Energy	Hourly	GJ
District Cooling Energy	Hourly	GJ
Heating Energy	Hourly	GJ
Cooling Energy	Hourly	GJ
ETS Inlet Temperature	Hourly	°C
ETS Outlet Temperature	Hourly	°C
Ambient Loop Setpoint Temperature	Constant	°C
Ambient Loop Delta T	Constant	°C
Average EPD	Constant	W/m2
Average LPD	Constant	W/m2

3.1.2.4 Parametric Simulations

The OpenStudio Analysis Framework and the Parametric Analysis Tool (PAT) were used to generate and run the parametric simulations. Each seed model (i.e., the small office and retail buildings) were run with the same algorithm to populate the parameter space. The building models were parameterized by differing the inlet temperature of the ambient loop, the district system design temperature difference (to emulate varying mass flow rates), and the internal gains of the space. Table 3.5 lists the distributions for the measure arguments that were used to populate the parameter space. The distributions were all set to uniform since the goal of the parametric analysis was to provide the largest set of operational data to maximize the effectiveness of the datasets for the reduced order modeling framework. Also, note that only measures with input distributions are shown in the table below.

Table 3.5: Parameter space for building energy models

Measure	Variable	Value(s)
Ambient Loop Setpoint Temperatures	Setpoint Temperature	Uniform[15, 25] °C
Ambient Loop Setpoint Temperatures	Design Delta T	Uniform[1, 7] °C
Hot Water Design Loop Temperature	Temperature	Static[60] °C
Internal Loads Multiplier	LPD Multiplier	Uniform[0.33, 3]

The parameter space was sampled 100 times using the Latin Hypercube Sampling (LHS) scheme. Historically, most sampling schemes leveraged Monte Carlo-based algorithms; however, due to the "long-running" time of simulations, it was preferred to run as few simulations as possible while maintaining statistical validity, thus the LHS scheme was selected. Figure 3.11 shows the difference between sampling a variable 100 times using LHS vs. a pure random sampling for a normal distribution. As shown, using LHS allows fewer samples to more accurately represent the distribution type.

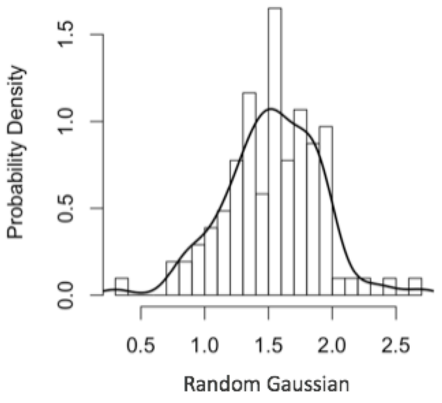
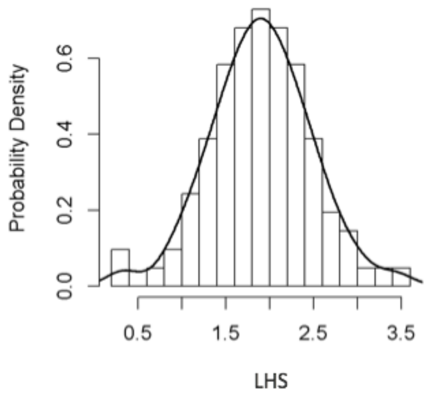



Figure 3.11: LHS vs random sampling (normal distribution with $n = 100$)

The OpenStudio Analysis Framework used the LHS samples to set the arguments of the measures which then generated and ran a fully defined EnergyPlus model, utilizing OpenStudio. The results are stored both as files and in the OpenStudio Analysis Framework's database.

3.1.3 Results and Discussion

The analysis used EnergyPlus, OpenStudio, and the OpenStudio Analysis Framework to generate the datasets needed for the ROM Framework. There were two major sections of this analysis, 1) the generation and simulation of the baseline models, and 2) the generation and simulation of a parametric analysis. The results for each are discussed in the following subsections.

3.1.3.1 Baseline Models

There were two building types with two different configurations that were modeled for the baseline models. The two building types were small office and retail. Each of these was modeled with and without an ETS. Table 3.6 shows the high-level results of the four simulations.

Table 3.6: Baseline model results

	Small Office	Small Office with ETS	Retail	Retail with ETS
EUI (MJ/m^2)	376.9	440.6	689.5	690.4
Total Heating EUI (MJ/m^2)	30.6	35.1	133.9	61.9
Total Cooling EUI (MJ/m^2)	22.0	117.3	37.7	301.1
Total Fans EUI (MJ/m^2)	42.7	5.3	203.6	5.3
Total Pumps EUI (MJ/m^2)	0.0	1.3	0.0	3.4
Heating setpoint not met (occupied hours)	27	14	0	12
Cooling setpoint not met (occupied hours)	0	4	0	0

Based on the results shown in the table, the comparison of the buildings with and without the ETS was similar. The only major difference is related to how ventilation occurs between the baseline building and the baseline with ETS building. The ventilation in the baseline buildings was substantially higher due to the existence of a full air handling unit. For example, the ventilation rate in the small office baseline building was $\approx$0.4 ACH; whereas, the baseline with ETS was around 0.05 ACH. This significantly changed the load due to outdoor air not being introduced into the building.

Figure 3.12 and Figure 3.13 show the difference in the loads of the small office by month. The difference is related to the difference in ventilation strategies, as described above. Overall, the energy use intensities (EUI) were similar between the baseline and baseline with ETS buildings.

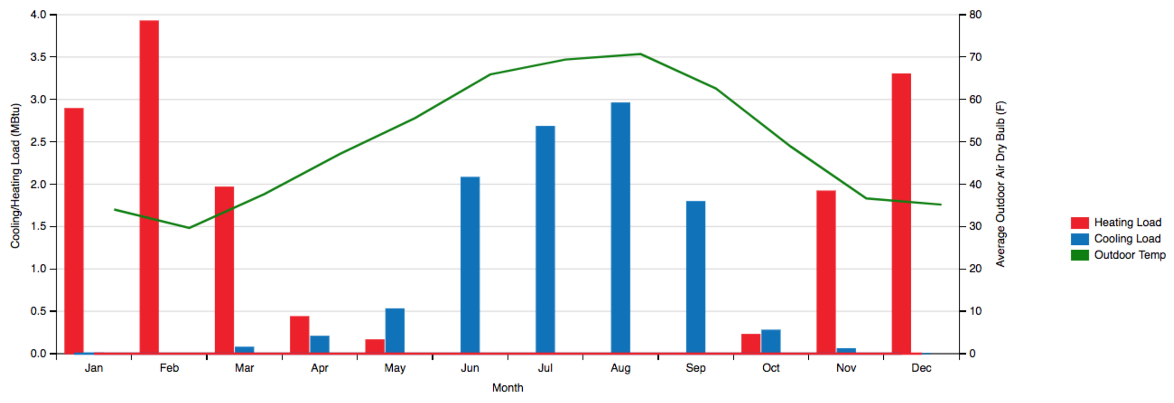


Figure 3.12: Small office baseline loads by month

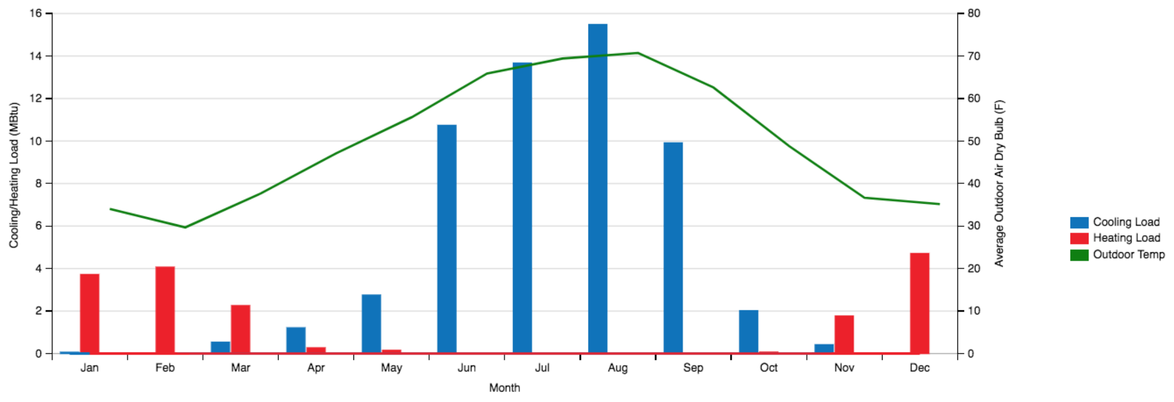


Figure 3.13: Small office baseline with ETS loads by month

Figures in Appendix A: Supporting Analysis Results show images of the baseline and baseline with ETS systems. Figures A.1, A.2 show the supply and demand side of the ambient loop systems which were the same for both baseline buildings. Figure A.3 and Figure B.1 graphically show the baseline system for the small office and retail building, respectively.

The monthly energy end uses for the small office building are shown in Figure 3.14 for electricity and Figure 3.15 for gas. Figure 3.16 shows the monthly energy end uses for the small office building with ETS. As shown, the peak energy is roughly the same between the two buildings. The majority of the variance is in the heating and cooling energy. Note that the baseline building used gas for heating, whereas the ETS baseline used electricity (and district heating). Similar plots for the retail building are shown in Appendix B.1: Baseline Results.

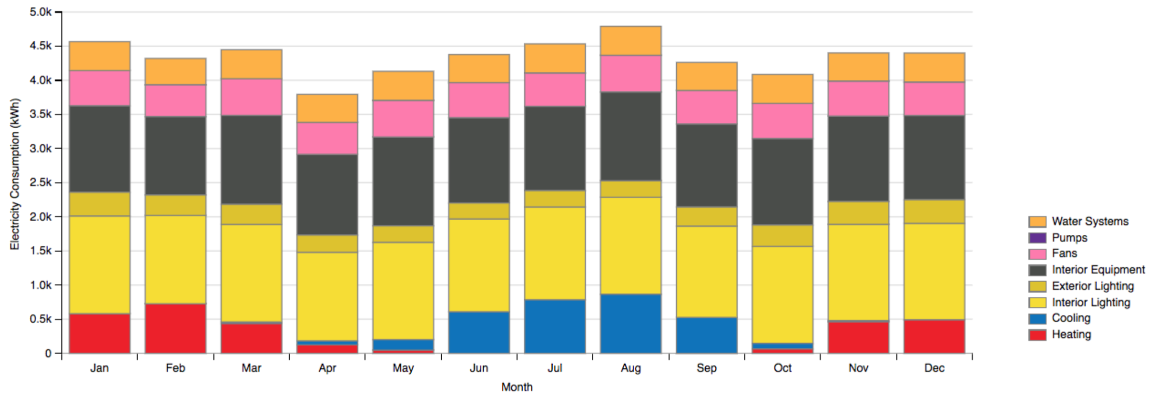


Figure 3.14: Small office baseline monthly electricity energy end use

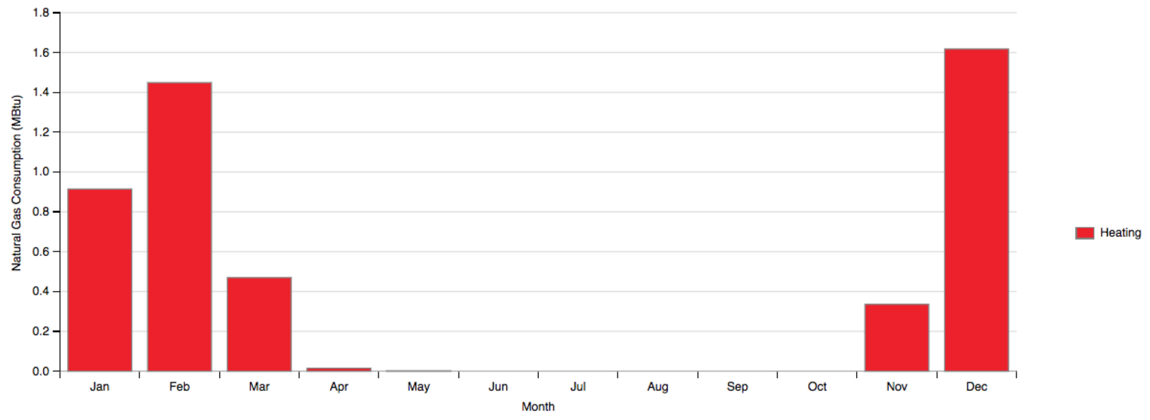


Figure 3.15: Small office baseline monthly gas energy end use

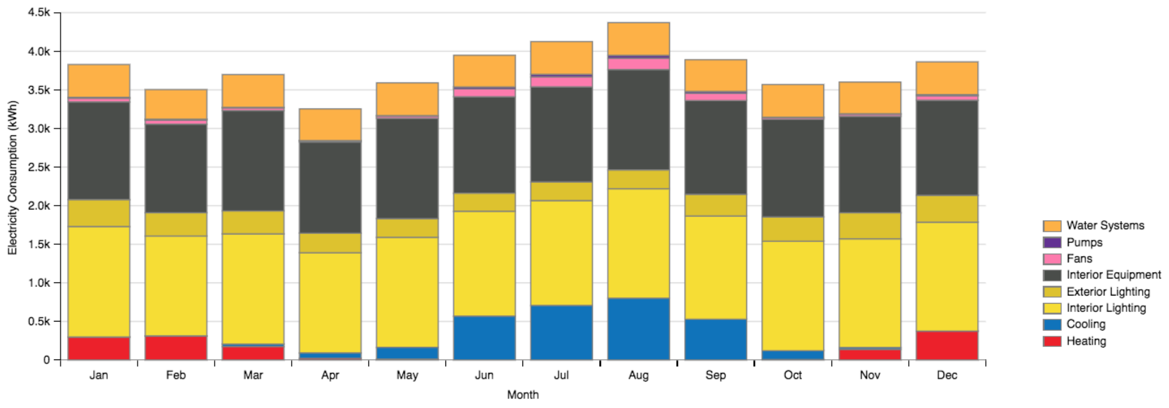


Figure 3.16: Small office baseline with ETS monthly electricity energy end use

The results of the ambient loop were inspected to ensure the results were reasonable. Figure 3.17 shows the ETS outlet temperature compared to the ETS inlet temperature from the ambient loop. As expected, the delta temperature of the inlet compared to the outlet temperature increased as the building load increased. In addition, the supplemental heating power ramped up as the building load increased over the day.

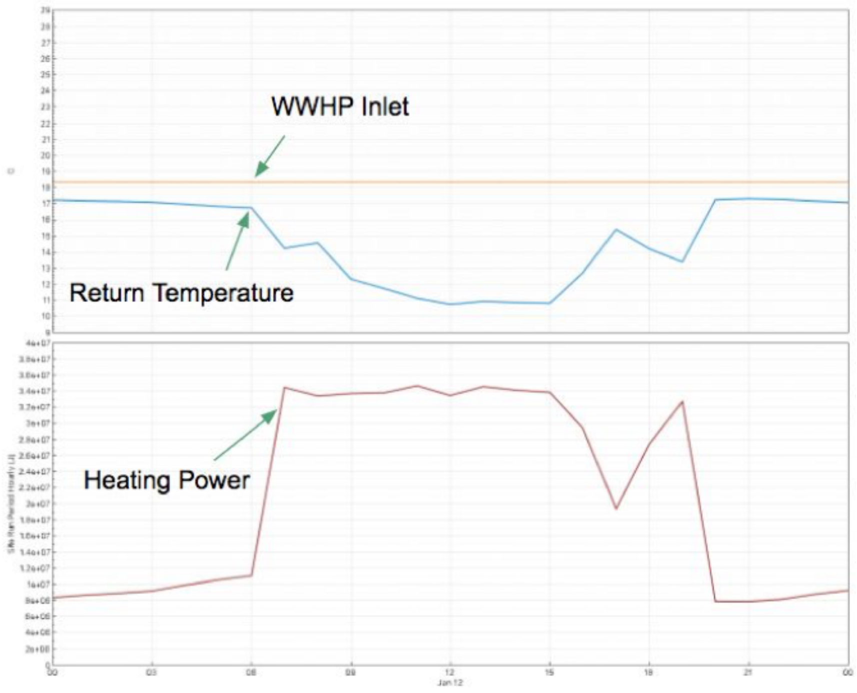


Figure 3.17: BEM simulation results (January 12)

The governing equation for the amount of energy extracted from the district heating and cooling plant is shown in 3.1 where $\dot{Q}$ is the amount of energy added/extracted, $\dot{m}$ is the mass flow rate, C_p is the specific heat (constant), and ΔT is the temperature across the inlet/outlet. The ΔT is a design parameter, thus assumed constant. The only non-constant variables are the mass flow rate and the resulting energy.

$$\dot{Q} = \dot{m} C_p \Delta T \tag{3.1}$$

Figure 3.18 shows the HVAC energy as a function of the outdoor air temperature. The plot shows the classic V-shaped energy signature showing the heating slope (left side) and the cooling slope (right side). The plot shows that for this building configuration in ASHRAE climate zone 5B, more energy was used for cooling than heating.

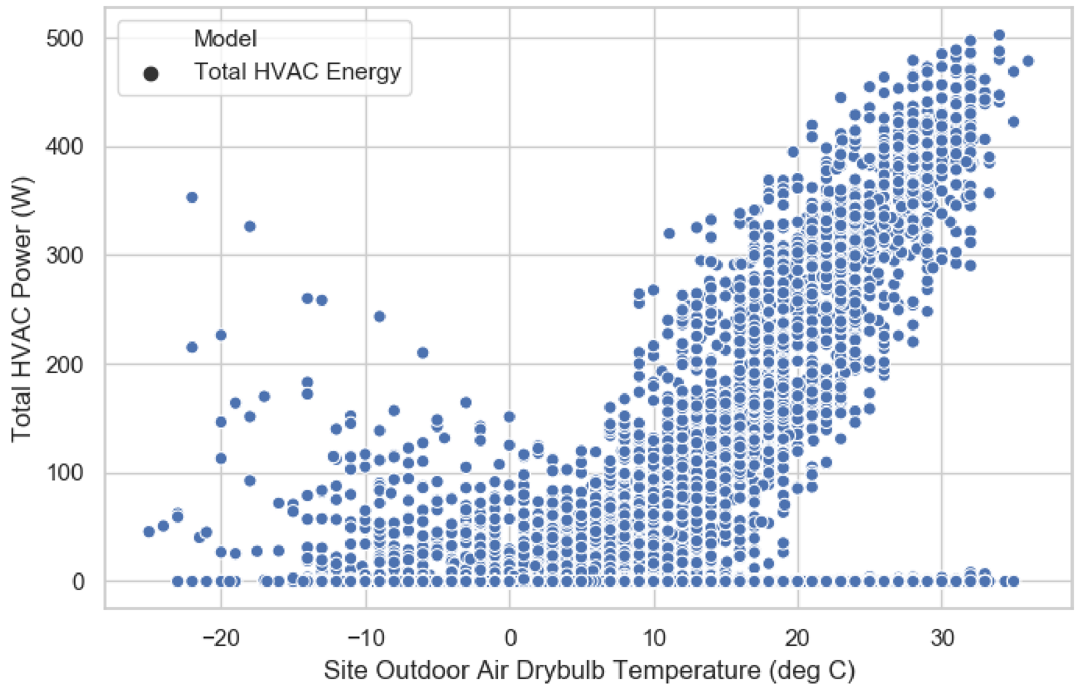


Figure 3.18: Small office heating power vs. the outdoor dry-bulb temperature

3.1.3.2 Parametric Analysis

Parametric analyses were run for both the small office and the retail building. Each analysis used LHS with 100 samples to generate a parameter space. The simulations were run on a local desktop computer with four cores dedicated to simulations using the OpenStudio Analysis Framework. Part of the project is to understand runtimes; therefore, the simulation runtimes were logged. There were three specific runtimes of interest:

- Measure Runtime: The amount of time to run all of the measures. This includes the prototype measures which requires sizing simulations and the reporting measures which can require time-intensive data queries for post-processing.
- EnergyPlus Runtime: The amount of time to run only EnergyPlus.
- Other Runtime: The other runtime is time needed to download, start the simulations, transition between the run states (e.g apply measure 1 to apply measure 2), and reporting the results back to the central run database

Figure 3.19 and Figure 3.20 show the runtimes for the office and retail building, respectively. On average, the small office building took 63 seconds to run; 18 seconds for EnergyPlus, 37 seconds for measures, and 8 seconds for other. The retail building performed similar, on average taking 63 seconds to run; 20 seconds for EnergyPlus, 36 seconds for measures, and 7 seconds for other.

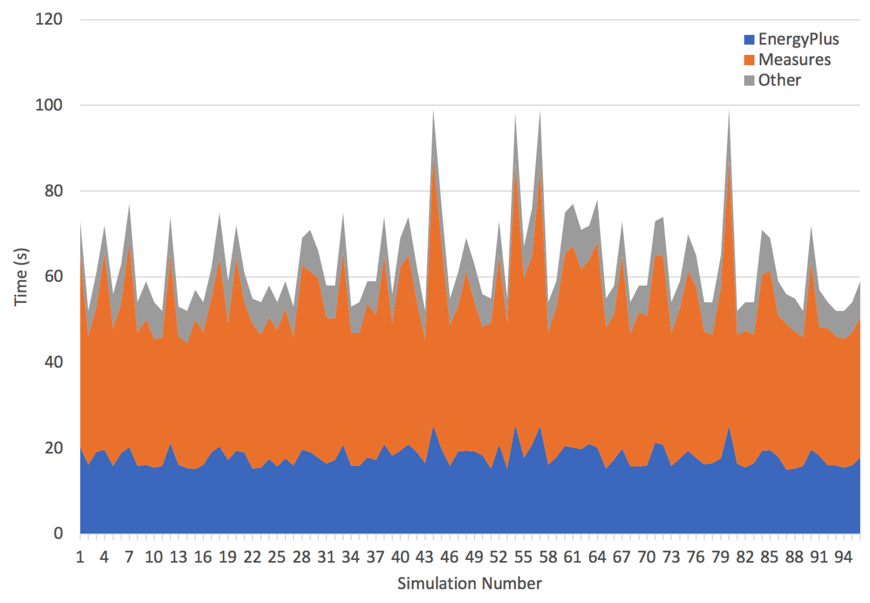


Figure 3.19: Simulation runtime for small office

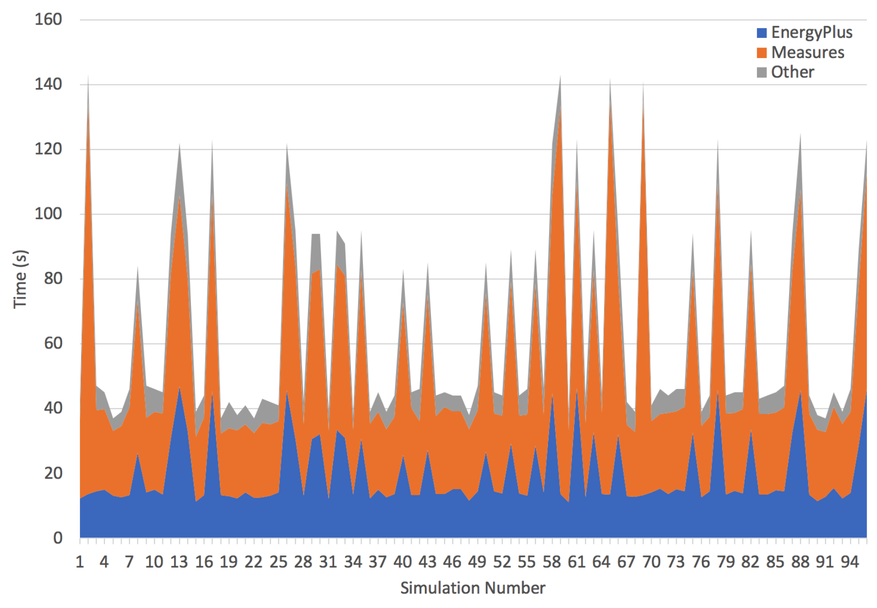


Figure 3.20: Simulation runtime for retail

The variability in runtime was consistent for the small office building and much more variable in the retail building. This was most likely due to thread affinity (how the computer assigns work to the processor) causing simulations to require more processing power than available, thus increasing the runtime. In general, the computational cost to run the measures was higher than running EnergyPlus. This cost is deemed reasonable due to the complexity of generating dynamic models without the aid of OpenStudio.

Parallel coordinate plots were used to visually verify the results of the parametric analysis. Figure 3.21 shows the plot of the small office after the parametric analysis. Each vertical axis is a variable (input or output) and each line (blue or gray) connecting the vertical axes is a single simulation. In the figure, the data are constrained to low EUI buildings (gray box on the total_site_eui axis), showing that low EUI buildings typically have lower LPD, and the delta design temperature and ambient loop temperature all not heavily dependent on the EUI (visible by the large spread in the design_delta and setpoint_temperature axes). A similar plot for the retail building is shown in Appendix B.2: Parametric Analysis Results.

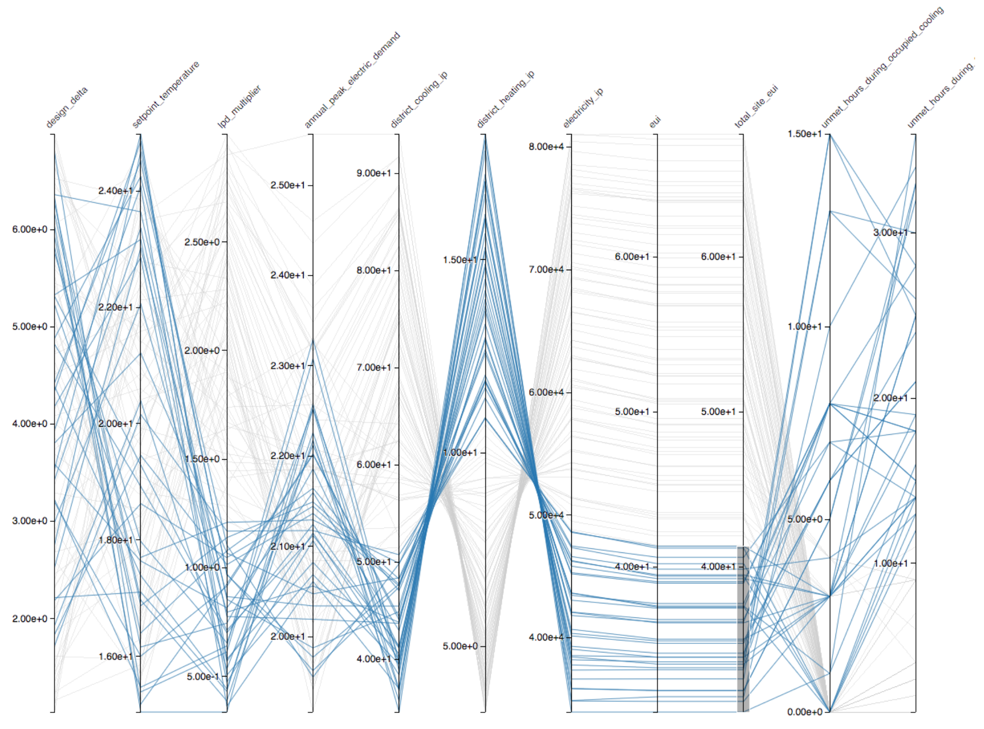


Figure 3.21: Small office parallel coordinate plot

3.2 Reduced Order Modeling Development

The reduced order modeling development required investigating various regression techniques such as ordinary least squares regression and machine learning approaches such as random forests and support vector regressions. A generalized framework was built to run these various regression algorithms on the same dataset to easily compare the performance. This generalized framework was termed the ROM Framework.

3.2.1 Background

There were three specific regression algorithms that were used, 1) ordinary least squares regression, 2) random forest, and 3) support vector regression. Each of these methods was researched to understand how they function and how to best evaluate their performance. This section will go into the details of each of the algorithms.

All of the models below used a validation, test, and training dataset. These datasets were extracted from the initial dataset and set apart for the various purposes which are described in the procedure section. Note that the term validation dataset used here is not the same as the validation data that is used in the k-fold cross-validation technique.

3.2.1.1 Ordinary Least Squares Regression

Ordinary least square regression models (or linear models) are very popular, easy to implement, and can perform very well with simple linearly-related data. In this case, the linear model was a multiple regression model in the form of Equation 3.2, where y is a matrix of results, X is a matrix of input data, β are the parameters as a vector, and ϵ is the error as a matrix.

$$y = X\beta + \epsilon \tag{3.2}$$

The results of the linear model were evaluated by calculating the coefficient of determination (R^2), visualizing the residuals to check for homoskedasticity, and generating Q-Q plots to check for normality. The resulting models were also evaluated with the validation dataset.

3.2.1.2 Random Forest

Random forests [58] are a type of ensemble modeling method where the predictions of various underlying models are combined. In the case of regression, the random forests model uses decision trees as the underlying model. Each tree is built using a bootstrapped sample (i.e. samples are replaced) by splitting the tree into branches with the largest reduction of the mean square error based on a limited selection of features. Splitting continues until the minimum number of samples per leaf node or the maximum depth of the tree is reached. For prediction, the results of each tree are averaged to provide the estimate. The Python package used to generate the random forests model [58] include many parameters that need to be tuned. The parameters of focus were:

- *n_estimators*: the number of trees to generate; initially 10 but used 100 or 200 for the final models
- *max_depth*: the maximum size that the tree can grow, defaults to no maximum
- *min_samples_split*: the minimum number of samples to perform a split; default was 2
- *min_samples_leaf*: the minimum number of samples required to be leaf node; default was 1
- *max_features*: the number of features to evaluate the best split, typically equal to the number of features divided by 3

The model generation included hyper-parameter tuning of the variables above. Hyper-parameter tuning was accomplished by performing a grid search with k-fold cross-validation, in this case the number of folds was selected to be three in order to decrease the build time. Cross-validation is a method in which the training dataset is divided into k folds and one of the folds is withheld as the other folds are used to train the model. The withheld fold (which is termed the validation data) is then used to validate the accuracy of the trained model. This is repeated until all the folds have been tested at which point the model performance results are averaged and persisted. Equations 3.3 and 3.4 show the error calculation for each cross-validation, where E_k is the error predicting the k-th fold, λ is a tuning parameter, y is the test data, x is the test parameters, $\hat{\beta}$ is the model.

$$E_k(\lambda) = \sum_{i \in k-\text{th part}} (y_i - \mathbf{x}_i \hat{\beta}^{-k}(\lambda))^2 \tag{3.3}$$

$$CV(\lambda) = \frac{1}{K} \sum_{k=1}^{K} E_k(\lambda) \tag{3.4}$$

In the case of the hyper-parameters and grid search, the user specifies ranges of model parameters to evaluate. Table 3.7 lists the hyper-parameters that were varied and the different values that were tested. An effort was taken to select the minimum number of reasonable values to test since the total number of cross-validation runs quickly adds up (and is time consuming). In this case, the values here resulted in 243 candidate models, and with 3-folds, 729 fits. This was done for each of the 6 response variables, 4,734 fits in total.

Table 3.7: Random forest hyper-parameter test values

Hyper-parameter	Tested Values
max_depth	auto, 3, 6
max_features	0.5, 0.66, 0.75
min_samples_leaf	1, 2, 4
min_samples_split	2, 3, 5
n_estimators	10, 100, 200

3.2.1.3 Support Vector Regression

The support vector algorithm is a class of nonlinear generalization algorithms developed in the early 1960's [59]. Support vectors are created by minimizing the error between the proposed hyperplane and the training data. Once minimized, the datapoints that are used to define the "tube" are the support vectors. If a support vector is removed, then the shape and performance of the model can change significantly. Figure 3.22 shows a contrived 2-D example of the support vector [59], where ϵ is the range where observed data will not incur the cost penalties. The higher the ϵ the smoother the resulting regression algorithm, and, consequently, the fewer support vectors. The figure also shows ζ which is the distance between the observed datapoint and the upper (or lower) bound of the error "tube". These values incur a cost penalty, which can be scaled using the C parameter, which typically is defaulted to 1.

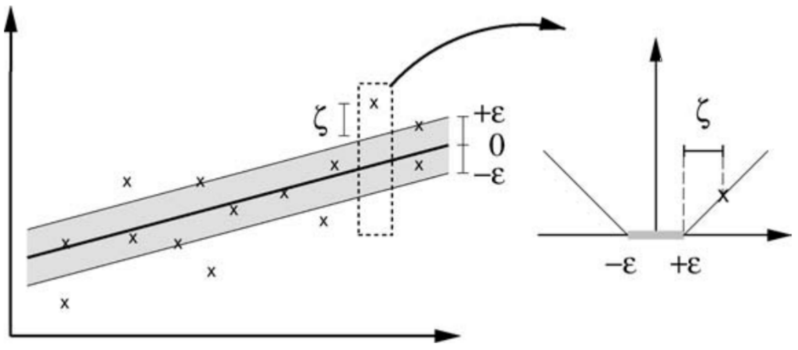


Figure 3.22: Support vector margins

Support vector algorithms use kernels to select the observed data that are included in the minimization equation. The most common kernels are linear, polynomial, sigmoid, and radial basis functions. The radial basis function (3.5) non-linearly maps datapoints to a higher dimension which allows for relationships between the covariates to be non-linear as well. The parameter γ is recommended to be set at $1/n_{\text{features}}$; however, γ should still be included in the hyper-parameter tuning.

$$K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2) \tag{3.5}$$

The build times for support vectors can be quite long due to the computational requirements (memory and CPU) to calculate and store the dot product matrix. In 2004, it was suggested that after 3,000 samples the user must use another approach to compute the support vectors [59]. As of

2018, computational advances and algorithm improvements (such sequential minimal optimization) have allowed more samples to be evaluated. The ROM Framework's down sampling capabilities allow for the samples to be sized within reason before memory or computation issues arise.

The training of support vector regressions require the data to be normalized within a standard range, in this case, -1 to 1. Each covariate was scaled using Python's StandardScaler object. In addition to scaling, several of the variables have cyclical properties. For example, the day of week is represented by an integer between 0 (Sunday) and 6 (Saturday) which are numerically distant but temporally near. The cyclical variables were transformed using Eq. 3.6, where v is the resulting value, i is the categorical value, and n is the total number of categories for the variable.

$$v = sin(2\pi i/n) \tag{3.6}$$

Support vectors have few training hyper-parameters that need to be evaluated and are shown below:

- C: the penalty parameter of the error term, defaults to 1
- γ: the parameter in the RBF equation, defaults to $1/n_{\text{features}}$
- ϵ: range of the penalty-free "tube", defaults to 0.1

In the case of the hyper-parameters and grid search, the user specifies ranges of model parameters to evaluate. Table 3.8 lists the hyper-parameters that were varied and the different values that were tested. Based on research, the values of C, γ, and ϵ should be varied by order of magnitudes.

Table 3.8: SVR hyper-parameter test values

Hyper-parameter	Tested Values
C	0.1, 1, 10
γ	0.25, 0.1, 0.01
ϵ	0.1, 0.25, 0.5

3.2.2 Procedure

The results from the Building Energy Modeling Data Generation section became the input for the reduced order models described in this section. The reduced order models were developed using a software library developed for this research termed the ROM Framework. The library is a collection of Python classes and a command line interface to load datasets, build regression models, run validation, save plots, and store the generated models as files on the computer for later use. The ROM Framework also includes an analysis library which allows for a ROM to be loaded and run through parametric sweeps with various input distributions. The goal of the ROM Framework was to consistently and deterministically evaluate regression techniques without needing to copy and paste large chunks of code in single-use scripts. Information on the ROM Framework can be seen in Appendix D: ROM Framework Information.

There were several metrics of interest based on the use case of the reduced order models:

(1) Amount of time to build and test the models (with k-fold cross validation if applicable)

(2) Accuracy of the models based on various performance metrics such as PCC, NMBE, and CVRMSE.

(3) Amount of time to load the models after they were built and persisted to disk

(4) Size of the persisted models on disk

(5) Amount of time to run a single time step through the reduced order models after models are loaded from disk

(6) Amount of time to run 8,760 hours through the reduced order models after models are loaded from disk

The overarching use case of the reduced order models was to quickly approximate the amount of energy used in an ambient loop system as well as the return temperature from the building back into the ambient loop. In order to accomplish this, various reduced order models were developed using the following high-level procedure:

- Withhold single 8,760 building results for testing model performance
- Optional: Down sample dataset to a fraction of the whole dataset
- Withhold percentage of results for the training dataset
- Use remaining data to fit or train model
- Run training data through model to test model performance
- If applicable, run k-fold cross validation and grid sear4ch on model to tune hyper-parameters
- Evaluate model performance for a single 8,760 model which was excluded from training dataset

The response variables and covariates list were considered the same for each reduced order model type and building type. The response variables are heavily focused on HVAC performance and it is reasonable to assume that the same response variables and covariates would be applicable for most building types. There were five response variables of interest and are shown in Table 3.9. The covariates were chosen by choosing a set of inputs and outputs to the EnergyPlus models that would physically explain the heating and cooling loads (e.g. time of day, inlet temperatures, outdoor environmental conditions). A larger set of covariates was originally chosen and was later reduced to the few most important covariates based on the random forest's relative importance outputs. The covariates were a mix of BEM inputs and outputs and are shown in Table 3.10. In the table, several of the variables were treated as factors (or categories), for example, the day of week is an integer value between 0 and 6. Depending on the reduced order model type, these factor-based variables are transformed to represent cyclical variables.

Table 3.9: ROM response variables

Variable	Time step	Units
District Heating Energy	Hourly	GJ
District Cooling Energy	Hourly	GJ
Heating Energy	Hourly	GJ
Cooling Energy	Hourly	GJ
ETS Outlet Temperature	Hourly	°C

Table 3.10: ROM covariates

Variable	Type	Units
Month	Factor (int)	N/A
Hour	Factor (int)	N/A
Day of Week	Factor (int)	N/A
Outdoor Relative Humidity	Variable	Percent
Outdoor Drybulb Temperature	Variable	°C
ETS Inlet Temperature	Variable	°C
ETS Outlet Temperature	Variable	°C
Average LPD	Variable	W/m2

3.2.2.1 Training Data

There were three types of data that were extracted for the purpose of training, testing, and validating the reduced order models: 1) a validation dataset, 2) training dataset, and 3) testing dataset. The validation dataset consisted of a randomly selected single annual simulation from the dataset. The purpose of the validation dataset was to compare the performance across multiple reduced order models with a dataset that was not used in fitting the model, used for model validation, as well as 30% of the remaining dataset (withheld at random) for the test dataset.

The building energy models resulted in a large number of datapoints for training the reduced order models. In order to reduce the physical size of the presisted models, reduce the likelihood of over-fitting, and to decrease the training time, a down sampling feature was created to down select the total number of datapoints before training. The ROM Framework allows the user to specify the fraction to randomly down sample the dataset after extracting the validation simulation dataset but before splitting the dataset into the training and testing datasets. Down sampling seemed reasonable due to the fact that each simulation resulted in 8,760 datapoints and if there were 100 samples, then down sampling to 15% of the total still resulted in 131,400 total samples.

The ROM Framework sets the random seed before sampling the dataset; therefore, all the ROMs used the exact same set of data for training and testing. This resulted in a training dataset size of 70% of the total 867,240 datapoints (8,760 hours x 99 simulations).

3.2.2.2 Model Evaluation Criteria

The validation criterion for each response variable was the Pearson correlation coefficient (PCC) 3.7, often denoted by r, which determined how linear the test data results correspond to the ROM modeled results. Where x_i,y_i are the actual and modeled value for each datapoint, n is the total number of data points, $\bar{x}, \bar{y}$ are the means of the actual and modeled values.

$$PCC = r = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{\sqrt{\sum x_i^2 - n\bar{x}^2}\sqrt{\sum y_i^2 - n\bar{y}^2}} \tag{3.7}$$

In addition to the PCC, the validation dataset was run with the reduced order models and compared using the normalized mean bias error (NMBE) and the coefficient of variation of the root mean square Error (CVRMSE), Equations 3.8 and 3.9, where y_i is the measured value for datapoint i, $\hat{y}_i$ is the modeled value, n is the total number of datapoints, and $\bar{y}$ is mean of the measured data. Conventionally, NMBE and CVRMSE are used to evaluate the error in measured data vs. modeled data; however, in this case, these metrics were used to describe the deviation between an idealized model (assumed to be correct) and the ROM results. ASHRAE Guideline 14 [24] was developed for standardizing the energy (and demand) saving calculations for buildings. The NMBE and CVRMSE metrics are defined in ASHRAE Guideline 14 as a metric for measuring the applicability of a model compared to actual measured energy consumption. In this case, the actual energy consumption was defined as the EnergyPlus results and the reduced order models represent the approximations. ASHRAE Guideline 14 recommends NMBE less than 5% and CVRMSE less than 15% should be considered as a reasonable model.

$$NMBE = \frac{\sum (y_i - \hat{y}_i)}{(n-1) \cdot \bar{y}} \tag{3.8}$$

$$CVRMSE = \frac{1}{\bar{y}} \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{n-1}} \tag{3.9}$$

3.2.3 Results and Discussion

This section discusses the results of each of the individual reduced order models. The results presented in this section are for the small office building (the retail building is in Appendix B.3: Reduced Order Model Results). Also, the section Results and Discussion contain results of comparing and contrasting the various ROM performances.

The results of each ROM focused on three categories: 1) build performance, 2) training data performance, and 3) validation data performance. The build performance includes discussion on the time to build the ROMs and any algorithm specific build performance metrics (e.g. analysis of variance for linear models). The training data discusses how well the models performed based on the withheld training data. Lastly, the validation data performance discusses model performance based on the withheld validation data. The validation data results are discussed in the Results and Discussion section.

3.2.3.1 Linear Model

Build Performance

The models built in 0.13 seconds on average for the full dataset with no down sampling. Figure 3.23 shows the build time for each of the response. As shown, the heating electricity response built the fastest and the cooling electricity took the longest; however, the differences between the two is minimal. The time for cross-validation was zero for the linear models.

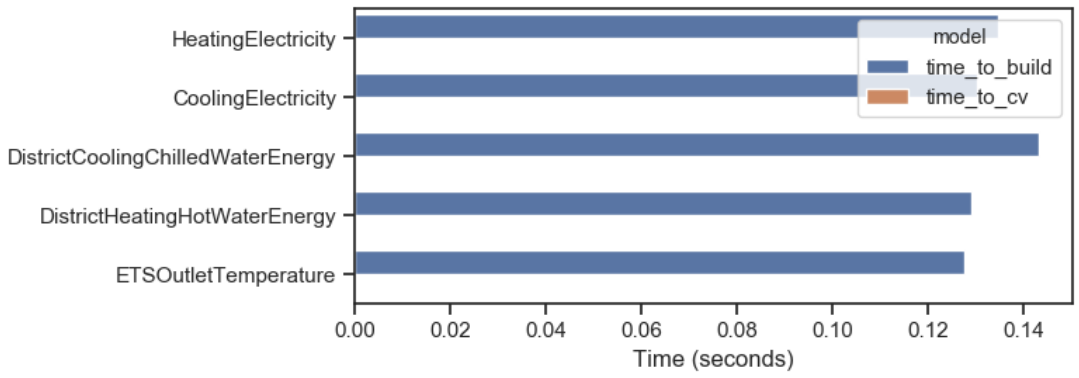


Figure 3.23: Linear model build time based on response variable, no down sampling

The ROM Framework creates diagnostic plots such as a residual and Q-Q plot for inspection for each response variable. Figures 3.24 and 3.25 show the performance of the district heating energy response variable and the ETS outlet temperature, respectively. The results show that the district heating energy is non-homoskedastic; whereas the ETS outlet temperature is slightly more homoskedastic. The Q-Q normal plots are not linear in either case, meaning that the models are non-normal.

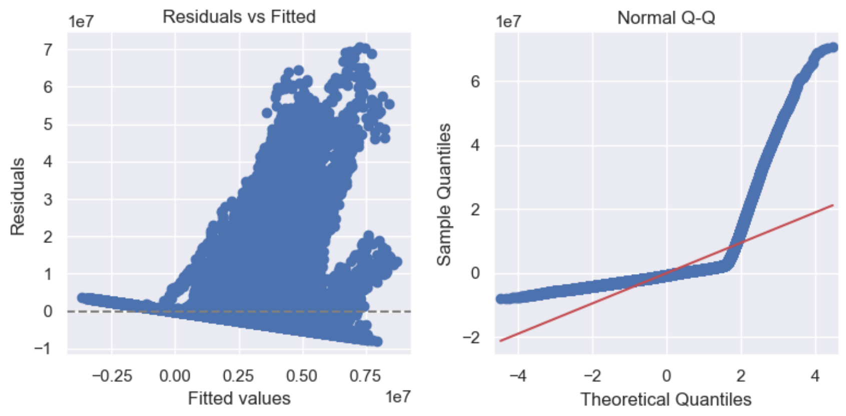


Figure 3.24: Diagnostic plot for district heating energy

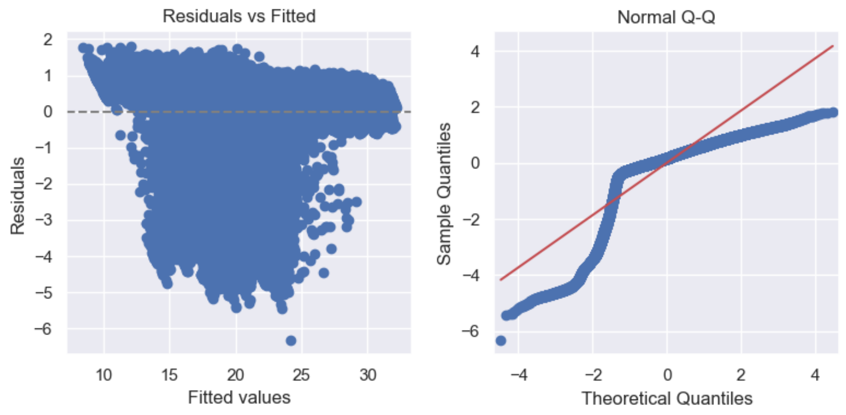


Figure 3.25: Diagnostic plot for ETS outlet temperature

The use of the down sampled datasets was unnecessary for the linear models due to the fast build time of the models and the lack of needing to run hyper-parameter tuning.

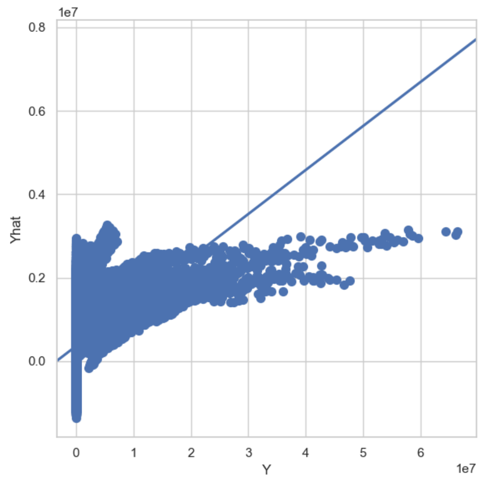
Training Data Performance

The performance of the linear models is presented in Table 3.11. As shown, the ETS outlet temperature response variables performed well based on the Pearson correlation coefficient. Also, down sampling the data set to 15% of the entire dataset did not significantly affect the performance.

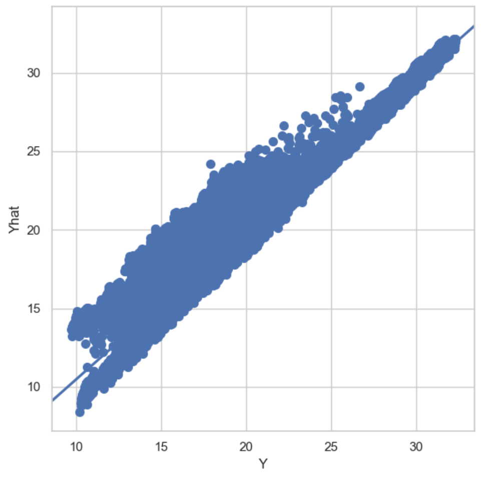
Table 3.11: Linear model performance

Response	PCC (15% Down sample)	PCC (100% Data)
Heating Electricity	0.322	0.322
Cooling Electricity	0.531	0.531
District Cooling Chilled Water Energy	0.533	0.532
District Heating Hot Water Energy	0.343	0.342
ETS Outlet Temperature	0.980	0.981

The training data was passed into the linear model and the results of each response variable was plotted against the actual training results. Figure 3.26 shows the Y-Y plot for the heating electricity and ETS outlet temperature. As shown, the non-linear nature of heating and cooling systems prevent the linear model from fitting well. However, the ETS outlet temperature model showed reasonable results due to the fact that the delta temperature across the ETS is an operational parameter and relatively constant during full load operation of the HVAC systems.



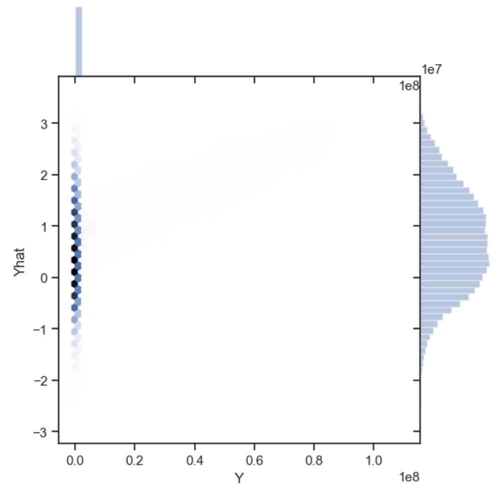
(a) Heating Electricity



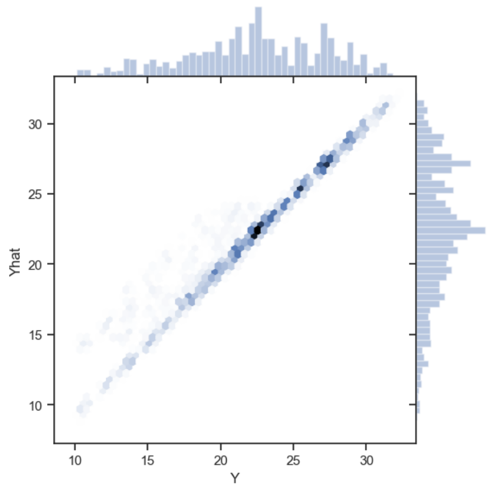
(b) ETS Outlet Temperature

Figure 3.26: Y-Y plots for the linear model

Hex plots were also generated for the Y-Y plots allowing inspection into the density of the plotted data. Figure 3.27 shows that the linear model for the district cooling had the majority of the training data near zero but predicted a wide range of values. The ETS outlet temperature had data much more reasonably spread out and evenly distributed.



(a) District Cooling



(b) ETS Outlet Temperature

Figure 3.27: Y-Y plots with densities

3.2.3.2 Random Forest

The random forest models took longer to build than linear regression models. In addition, there were more parameters that affect the performance of the models that needed to be accounted for. For these reasons, and to save build time, the use of the down sampled dataset was used when performing the cross-validation.

Build Performance

Using the full dataset (no down sampling), a single random forest model took 50 seconds on average to build. To perform k-fold cross-validation using the full dataset would take over 10 hours using the grid search parameters defined above. In addition, it is highly likely that fitting the model on the full dataset would not drastically improve the model performance. Therefore, several down sampled datasets were evaluated (5%, 15%, and 25%) to determine if the model performance was impacted based on the down sampling fraction.

Using the 15% down sampled dataset, the random forest model took, on average, 3.7 seconds to build and ≈15 minutes for hyper-parameter tuning/cross-validation, see Figure 3.28. Note that the cross-validation time was significantly higher than the build time.

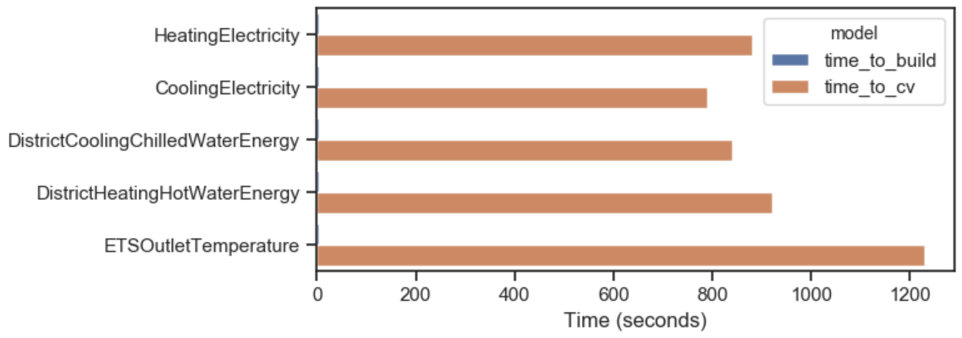
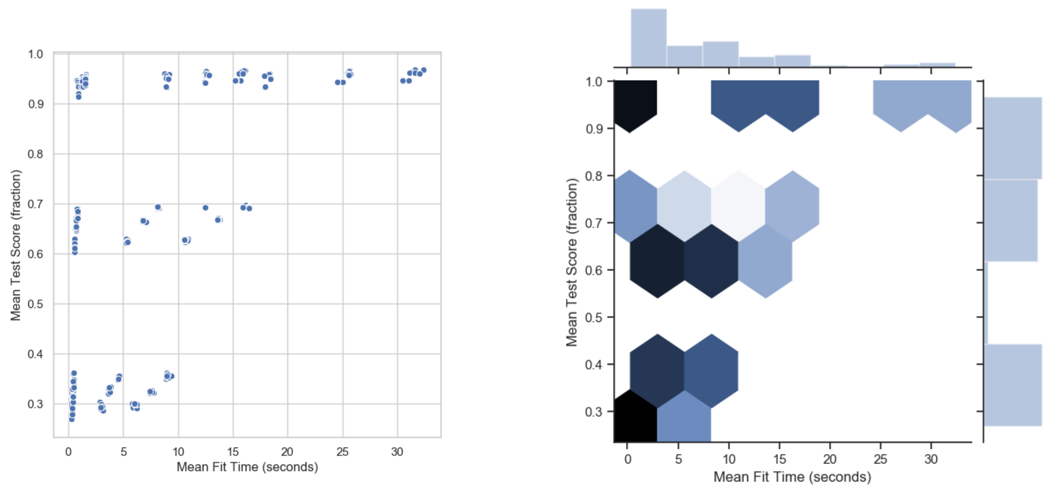


Figure 3.28: Random forest build and cross-validation time

Data were stored during the grid search and cross-validation to understand how the hyper-parameters impacted the build time and performance of the models. Figure 3.29 shows the mean fit time (for a complete cross-validation) compared to the mean test score for the district heating energy. The raw data show a positive correlation between longer fit times with higher mean test scores; however, the hex plot version shows that there is a high density of data with a low mean fit time and high test score. This pattern was consistent for the cases with the other response variables.



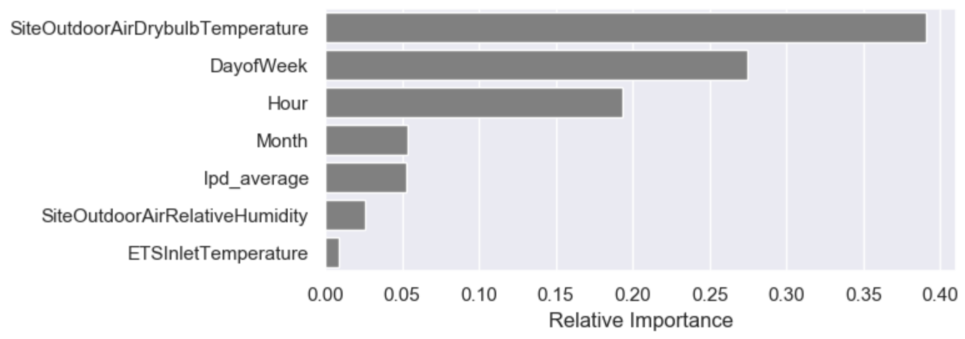
(a) Raw Data

(b) Hex Plot

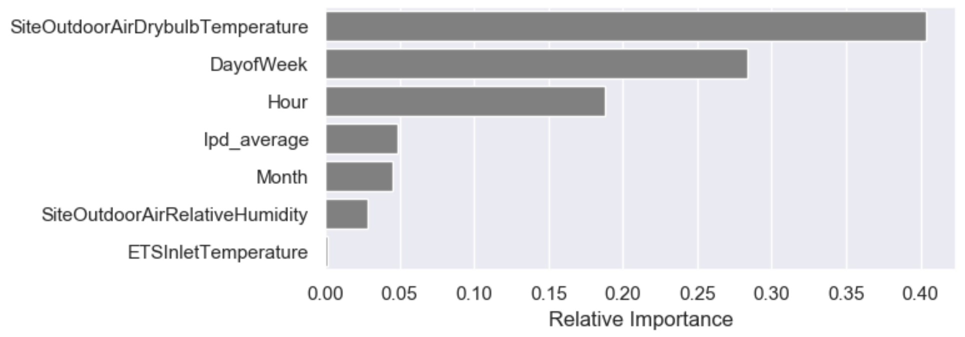
Figure 3.29: District heating energy build time compared to test score

Training Data Performance

The random forests models performed very well but require a long time to generate. The random forests algorithm also has the built-in ability to determine the most important variable as a byproduct of using random forests. The relative importance is a measure of how much the variable reduces the mean square error. Figure 3.30 shows the cooling variable importance factors. The greater the relative importance, the more sensitive the model was to that covariate. The most important factor for cooling related energy use was the outdoor dry-bulb temperature, day of week, and hour of day.



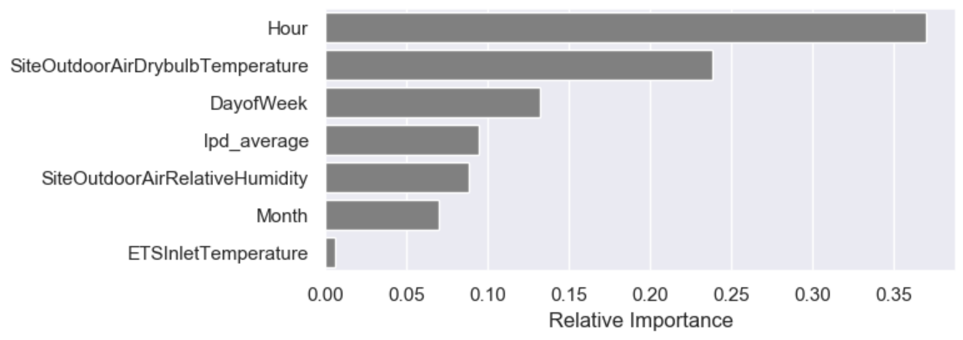
(a) Cooling Electricity



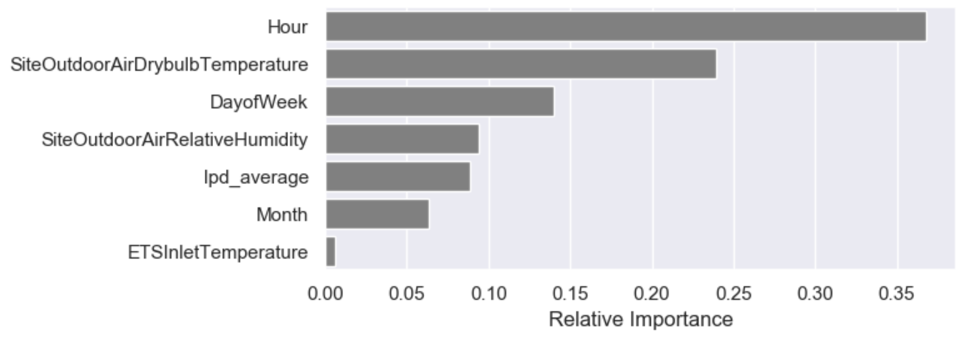
(b) District Cooling Energy

Figure 3.30: Cooling response relative importances

Similarly, Figure 3.31 shows the important covariates for the heating electricity and district heating hot water energy. The heating loads were more a function of the hour and day of week, compared to the cooling energy loads.



(a) Heating Electricity



(b) District Heating Energy

Figure 3.31: Heating response relative importances

The ETS outlet temperature was modeled reasonably as a linear model. Figure 3.32 shows that the random forests model that the most important covariate for predicting the ETS outlet temperature was the ETS inlet temperature.

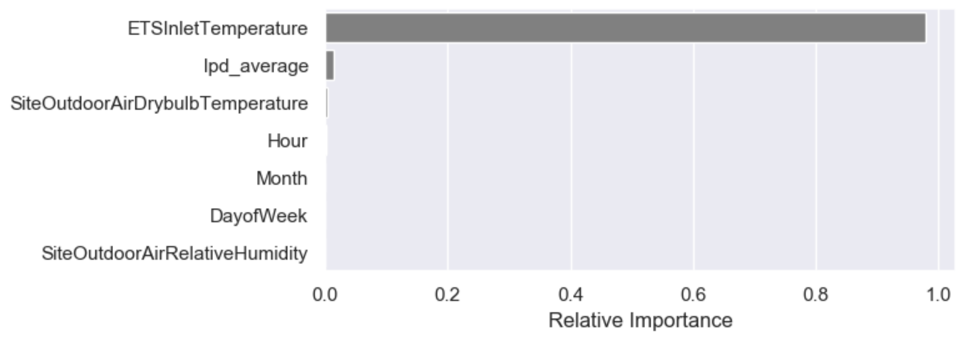


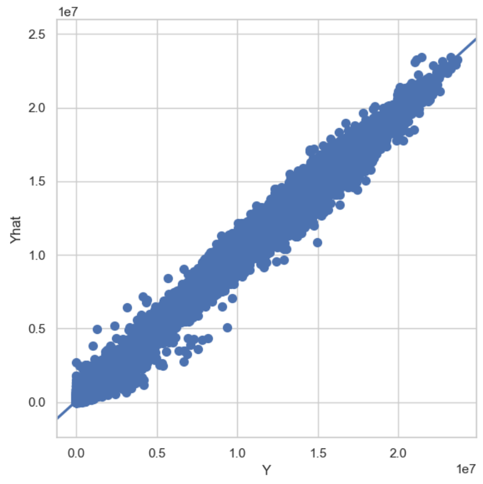
Figure 3.32: ETS outlet temperature importance

Table 3.12 lists the performance of the random forest models based on the Pearson correlation coefficient and the percent of down sampled data. As shown, using 15% down sampling of data shows comparable performance to a dataset that includes 100% of the data.

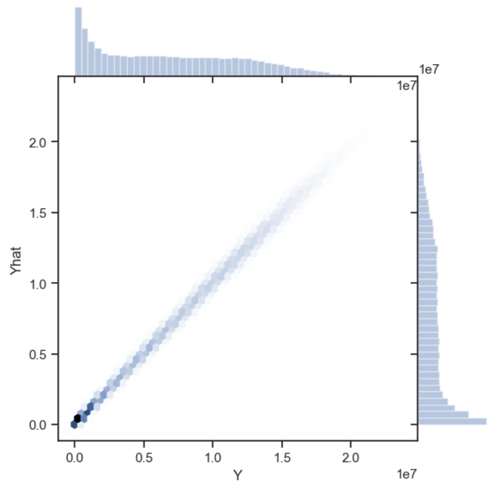
Table 3.12: Random forest performance

Response	PCC(15% Down sample)	PCC (100% Down sample)
Heating Electricity	0.989	0.999
Cooling Electricity	0.995	0.999
District Cooling Chilled Water Energy	0.997	0.999
District Heating Hot Water Energy	0.999	0.999
ETS Outlet Temperature	0.999	0.999

Figure 3.33 shows the Y-Y plots of the cooling electricity response, with and without hex-binning. As shown, the correlations are reasonable, with a large amount of data around zero (based on the hex plot). Figure 3.34 shows similar plots for the outlet temperature.

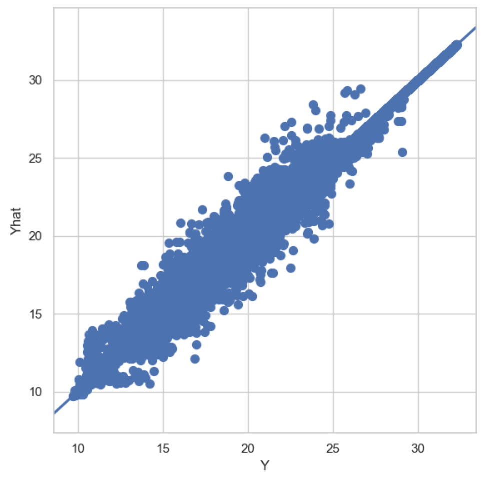


(a) Raw data

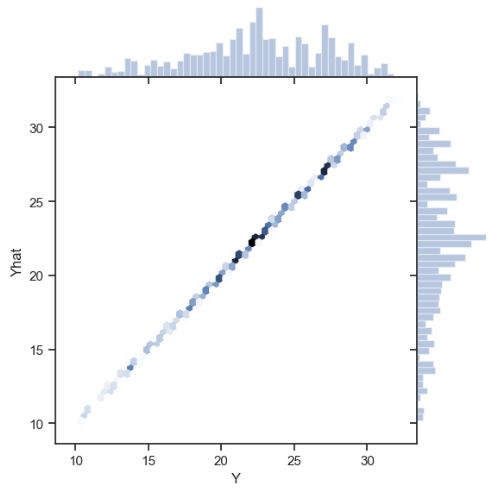


(b) Hex plot

Figure 3.33: Y-Y Plots for random forest cooling electricity



(a) Raw data



(b) Hex plot

Figure 3.34: Y-Y Plots for random forest ETS outlet temperature

3.2.3.3 Support Vector Regression

The support vector regressions with a radial basis function performed well depending on the response variable. An additional covariate was added to improve the algorithm. The added covariate was a state variable signifying if the HVAC system was off, in cooling mode, in heating mode, or was heating and cooling during the sample hour. The SVR models took a prohibitive amount of time to build on the entire dataset; therefore, the models were only built on the down-sampled datasets.

Build Performance

The SVR models were built with scaled covariates. Also, if a covariate's value had a cyclical relationship, the results were transformed using the sine wave definition in the procedures section above. The full dataset of 876,000 samples before splitting the training data was initially used; however, the SVR model was unable to be trained due to the large amount of data causing memory crashes. The 15% down sampled dataset was therefore used for analyzing the SVM ROM performance. In addition, the hyper-parameter tuning using 3-fold cross-validation took a significant amount of time. The grid search for the hyper-parameter tuning only consisted of 3 variables (ϵ, C, and γ) and the predetermined values for each resulted in 36 fits. The 15% down sampled dataset resulted in ≈90,000 training samples and the cross-validation for a single fit, in the worst case, took ≈50 minutes.

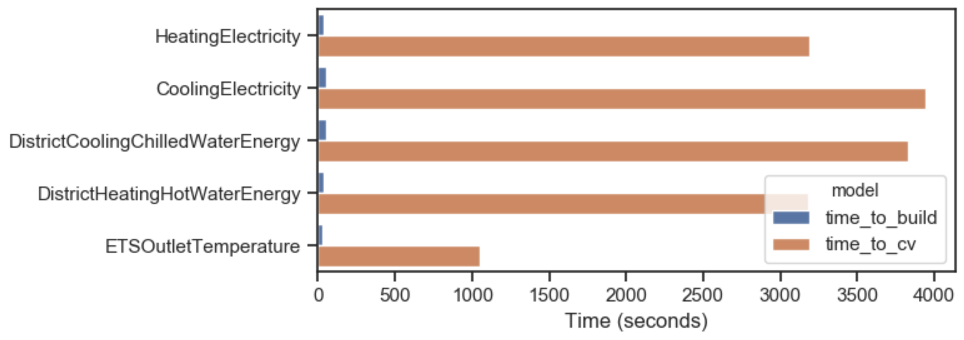


Figure 3.35: SVR build and cross-validation time, for 15% down sampled data

Figure 3.36 shows the mean test score of the cross-validation grid search as a function of the mean fit time for the ETS outlet temperature. It shows that, for the ETS outlet temperature, the test score is not highly dependent on the fit time, that is, there are low fit times that perform better than the higher fit times. This trend was consistent with the other response variables.

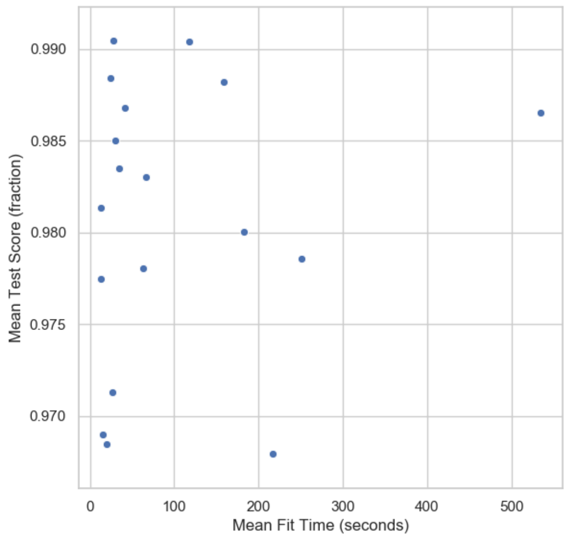


Figure 3.36: SVR build and cross-validation time for ETS outlet temperature, 15% down sampled data

The grid search was set up for the SVR using the values specified in the background section. All of the response variables ended up selecting the following parameters: $\gamma = 0.5$, $\epsilon = 0.25$, and $C = 1$. The gamma variable was higher than the default meaning that the radial basis function had increased the 'radius' and included more data in the kernel. The ϵ value was also higher than the default resulting in a large tube area where the penalty value was not incurred.

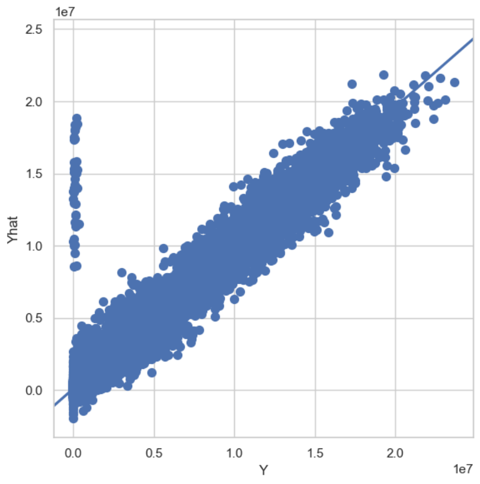
Training Data Performance

Table 3.13 lists the performance of the support vector regression based on the Pearson correlation coefficient. Values are shown for the 15% down sampled case with and without hyper-parameter tuning. Other down sampled values were not investigated due to the inability to build large datasets with support vector algorithms. In most cases, the hyper-parameter tuned parameters performed better than the untuned case. In general, the cooling response variables performed better than the heating response variables.

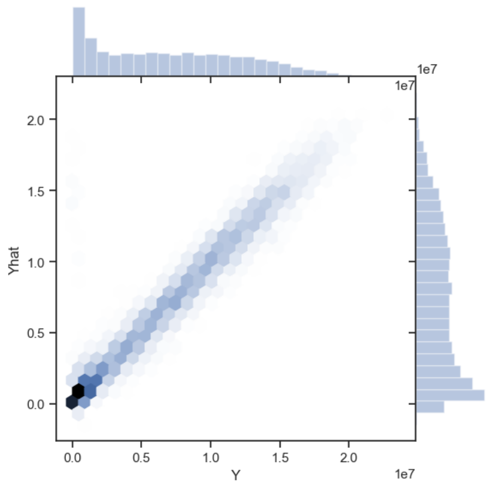
Table 3.13: Support vector regression

Response	PCC (no tuning)	PCC (with tuning)
Heating Electricity	0.827	0.881
Cooling Electricity	0.974	0.984
District Cooling Chilled Water Energy	0.974	0.984
District Heating Hot Water Energy	0.850	0.919
ETS Outlet Temperature	0.995	0.995

Figures 3.37, Figure 3.38, and Figure 3.39 show the Y-Y plots for cooling electricity, heating electricity, and ETS outlet temperature, respectively. Visually the models perform well. In the hex plots below, values that were exactly (0,0) were removed in order to increase the visibility of the hex bins. Due to the nature of the data sets, a large amount of data will be at 0,0 for heating and cooling (if the model performs well) due to the nature of the system being off because it is either the other season or night time. As discussed above, the addition of the HVAC state variable was added to increase the detectability of the HVAC state and increase the performance. The cooling electricity (and district cooling) both had values on the top left portion of the Y-Y plot (low actual energy but higher modeled energy); however, these data do not appear in the hex plot, meaning that the frequency of these points is low.

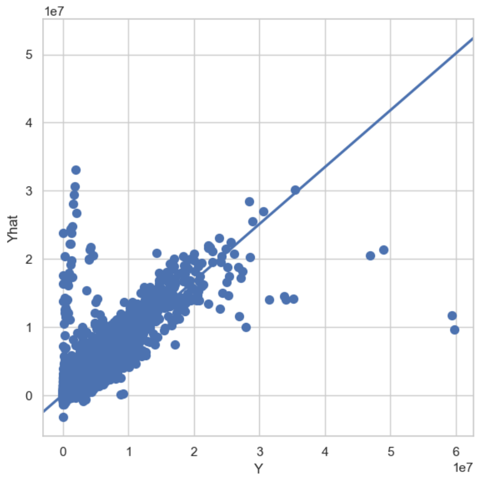


(a) Cooling Electricity

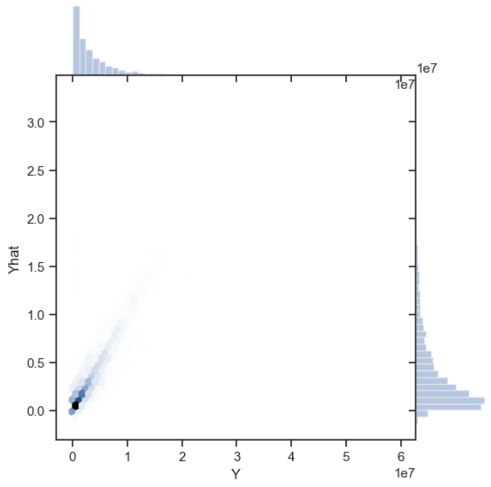


(b) Hex plot

Figure 3.37: Y-Y Plots of the cooling electricity for support vector regression

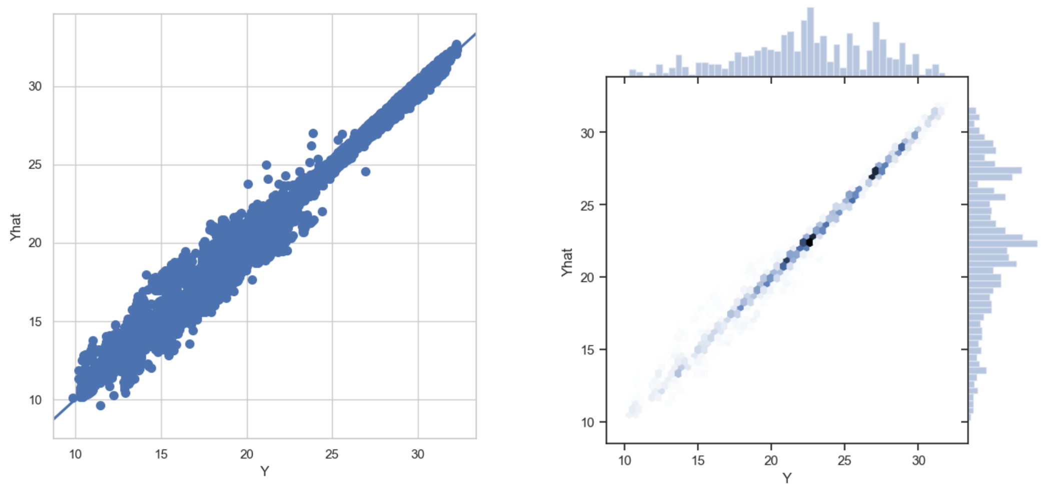


(a) Raw data



(b) Hex plot

Figure 3.38: Y-Y Plots of the heating electricity for support vector regression



(a) Raw data

(b) Hex plot

Figure 3.39: Y-Y Plots for support vector regression ETS outlet temperature

Chapter 4

Results and Discussion

A small office and a retail building were modeled with and without an ETS in order to generate the dataset used to build reduced order models. The results and discussion presented here focus on the results of the reduced order models for only the small office building. The first section discusses how the models performed relative to each other based on the withheld validation data, the runtime performance, and the down sampling performance. The results for the retail building are presented in Appendix B.4: Validation Results.

4.1 Model Performance

The model performance used the PCC metric to determine the performance of the various models. The results are presented in Table 4.1 using the 15% down sampled data. As shown the random forest and the SVR models performed well with PCC values greater than 0.9. The linear model performed well for the ETS outlet temperature. Based on these results, the preferred model would be the random forest; however, there are several other metrics of interest and will be discussed in the following subsections.

Table 4.1: PCC performance of reduced order models

Response	ROM Type	PCC
HeatingElectricity	LinearModel	0.322
HeatingElectricity	RandomForest	0.989
HeatingElectricity	SVR	0.882
CoolingElectricity	LinearModel	0.532
CoolingElectricity	RandomForest	0.995
CoolingElectricity	SVR	0.983
DistrictCoolingChilledWaterEnergy	LinearModel	0.533
DistrictCoolingChilledWaterEnergy	RandomForest	0.997
DistrictCoolingChilledWaterEnergy	SVR	0.984
DistrictHeatingHotWaterEnergy	LinearModel	0.342
DistrictHeatingHotWaterEnergy	RandomForest	0.992
DistrictHeatingHotWaterEnergy	SVR	0.919
ETSOutletTemperature	LinearModel	0.980
ETSOutletTemperature	RandomForest	0.999
ETSOutletTemperature	SVR	0.995

4.1.1 Validation Data

The validation data was a random temporally contiguous 8,760 sample from the original EnergyPlus models which was withheld before the data were split for training. The validation data was loaded into memory and passed into each of the reduced order models. The results were stored in a larger data frame. Each of the reduced order model results was compared to the EnergyPlus model and two performance metrics were calculated, the CVRMSE and NMBE. As described in Section Model Evaluation Criteria, the use of ASHRAE Guideline 14 was used to assess the goodness of the reduced order models compared to the actual EnergyPlus model. Table 4.2 shows the results of the metrics. As shown, there were few models that fell within ASHRAE Guideline 14 requirements, however, overall, the SVR and the random forest models performed better on predicting the energy and the linear model performed well on the ETS outlet temperature. The cooling models overall fit better than the heating models.

Table 4.2: Performance of validation data

Response	ROM Type	NMBE	CVRMSE
HeatingElectricity	LinearModel	18.0	700
HeatingElectricity	RandomForest	24.7	135
HeatingElectricity	SVR	7.44	275
CoolingElectricity	LinearModel	14.4	181.3
CoolingElectricity	RandomForest	14.0	40.5
CoolingElectricity	SVR	6.93	47.4
DistrictCoolingChilledWaterEnergy	LinearModel	9.84	185
DistrictCoolingChilledWaterEnergy	RandomForest	6.47	24.2
DistrictCoolingChilledWaterEnergy	SVR	1.92	45.4
DistrictHeatingHotWaterEnergy	LinearModel	15.9	733
DistrictHeatingHotWaterEnergy	RandomForest	28.1	139
DistrictHeatingHotWaterEnergy	SVR	7.05	289
ETSOutletTemperature	LinearModel	0.29	1.39
ETSOutletTemperature	RandomForest	2.45	2.93
ETSOutletTemperature	SVR	0.612	2.16

In addition to the core performance metrics, three time periods were visually inspected to investigate how well the reduced order models accounted for the variations based on the time of year. The time periods are described below. The swing season was chosen to include a time period where both heating and cooling were most likely to occur, even within the same day.

(1) : Summer: July 01 to July 10

(2) : Winter: January 15 to January 25

(3) : Swing Season: March 01 to March 10

A few choice images are presented below to demonstrate the performance of the models. The remaining images are in Appendix A.2.4: Validation Results. Figure 4.1 shows the performance of the three models during the summer period. As shown, the random forest model follows the actual data the best, although, slightly under predicting. The support vector model followed the weekday modeled data well; however, the support vector regression was unable to predict the zero weekend energy consumption for cooling. The data for district cooling energy performed similar to the cooling electricity.

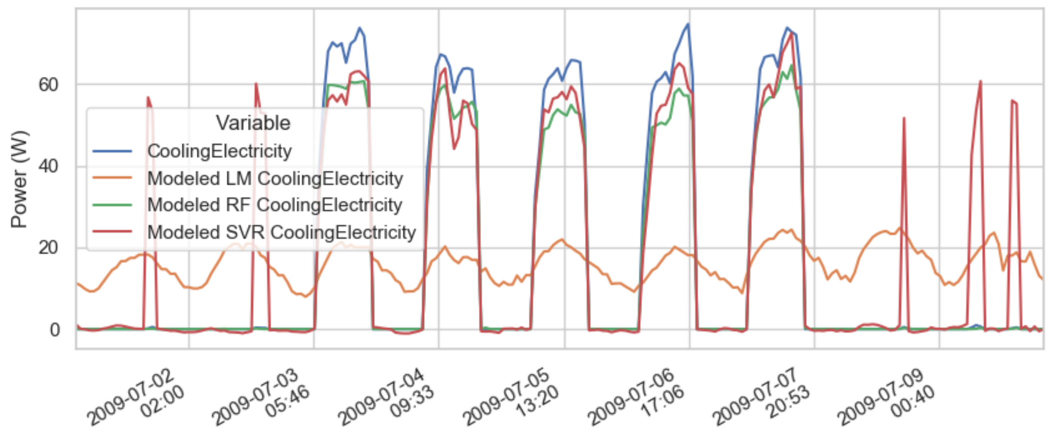


Figure 4.1: Summer validation period cooling electricity performance

Figure 4.2 shows the heating electricity in the summer period. As shown, the random forest model was the only model that reasonably predicted no heating electricity during the summer. The linear model predicted negative energy and the support vector regression erratically predicted values around zero. The district heating energy models predicted similarly to the heating electricity.

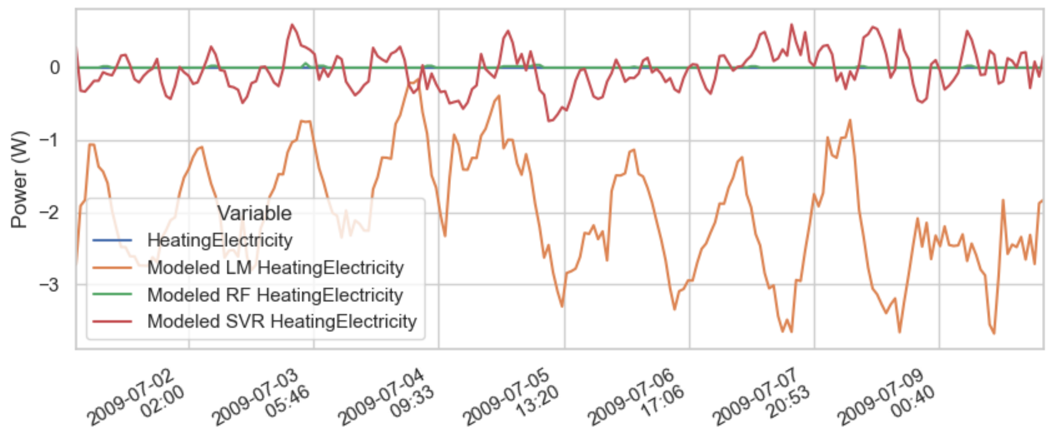


Figure 4.2: Summer validation period heating electricity performance

Figure 4.3 presents the ETS outlet temperature during the summer validation period. The actual data has little variation due to the summer run period with a stable 24.8°C. All three models predicted between 24.4 and 25.2°C.

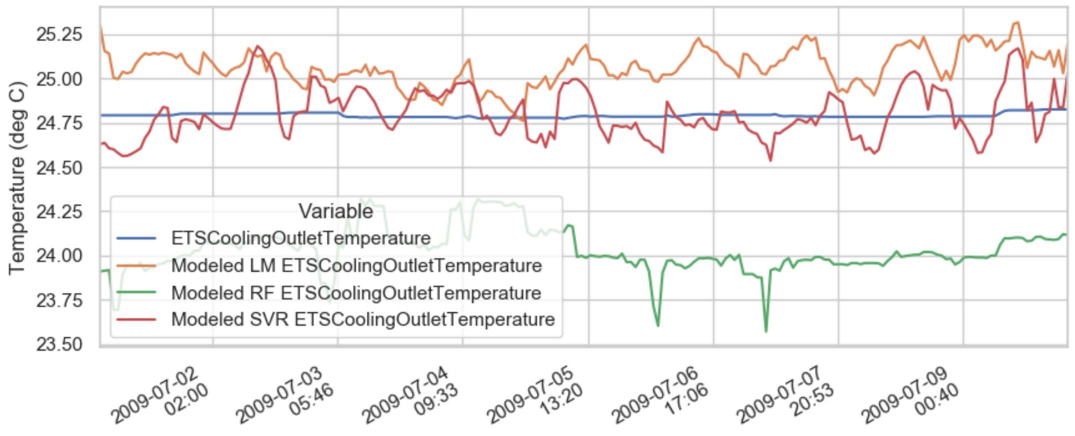


Figure 4.3: Summer validation period ETS outlet temperature performance

The winter validation period visually performed similar to the summer validation. Figure 4.4 presents the modeled performance and shows that the random forest, again, performed the best; however, the support vector regression was reasonable as well. Both the support vector regression and linear model had values below zero which are physically impossible.

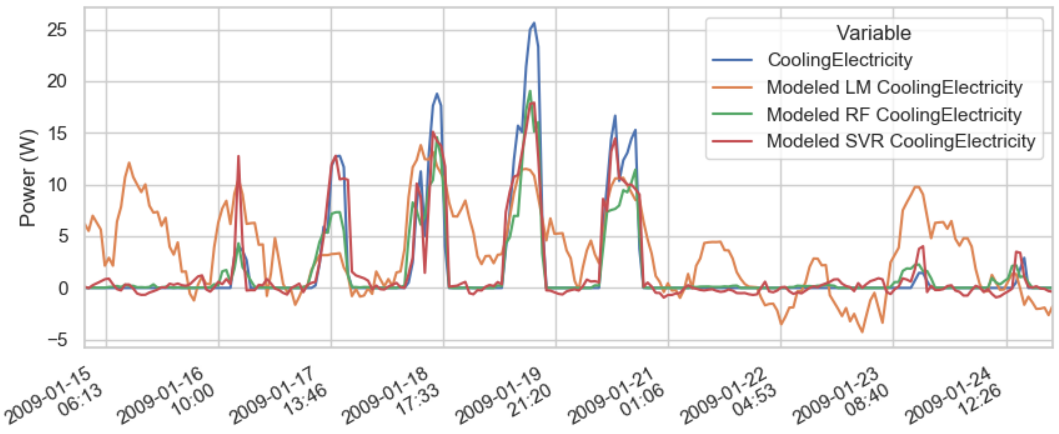


Figure 4.4: Winter validation period cooling electricity performance

The winter validation period district heating performance is shown in Figure 4.5. Again, the random forest model performed the best as it was able to track the peaks much better than the support vector regression or the linear model.

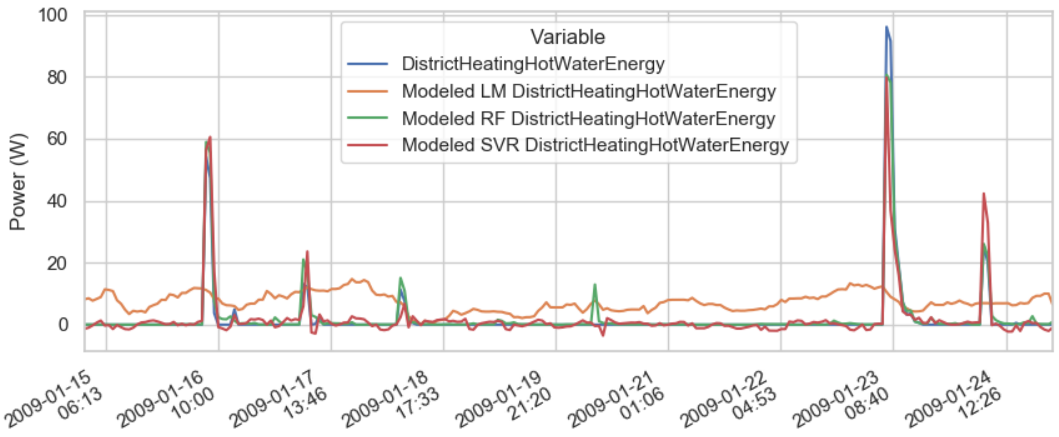


Figure 4.5: Winter validation period district heating energy performance

The ETS outlet temperature for the winter validation period is shown in 4.6. For the random model selected out of the dataset, the random forest model underpredicted the ETS outlet temperature by around 0.5°C. In this case, the support vector regression predicted the higher ETS outlet temperature the best, but poorly predicted the lower temperatures.

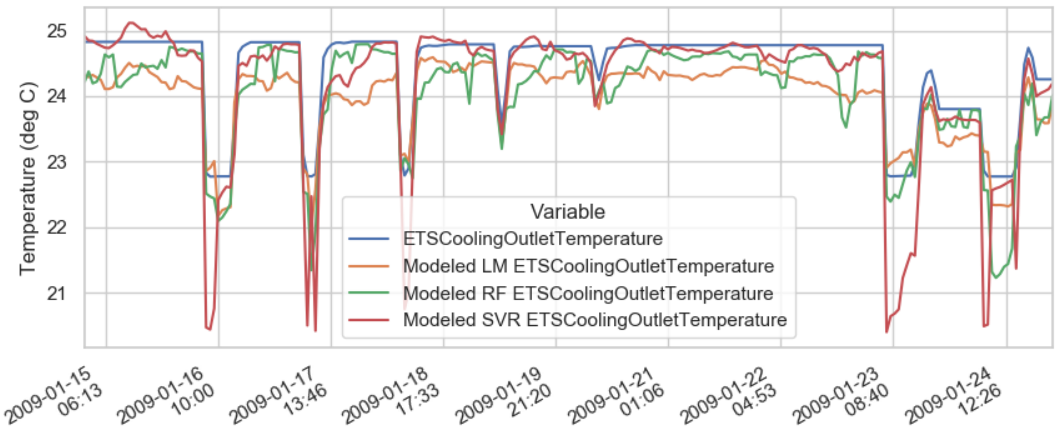


Figure 4.6: Winter validation period ETS outlet temperature performance

Finally, the swing season validation period results are presented in Figures 4.7, 4.8, and 4.9. The random forest model tended to perform the best. As shown in the ETS outlet temperature plot, the random forest model is best at handling the day-to-day swing due to the random forest model's training based on categorical variables and splitting trees.

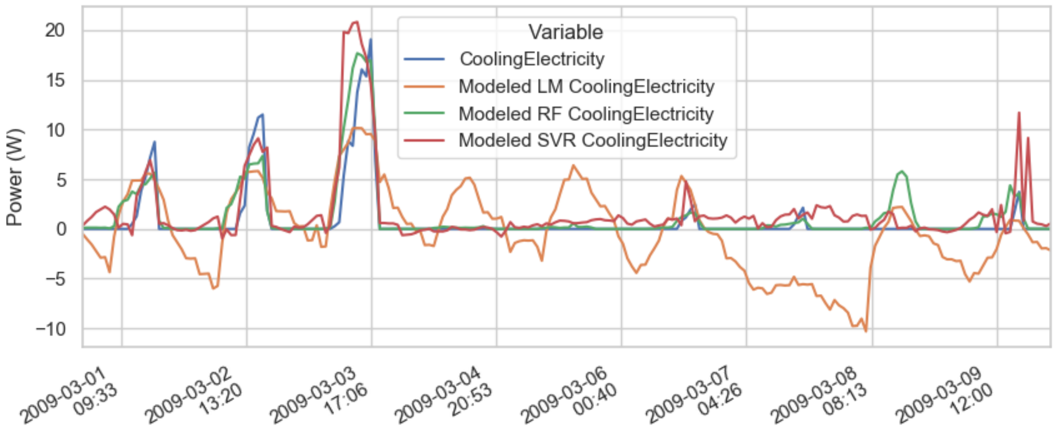


Figure 4.7: Swing season validation period cooling electricity performance

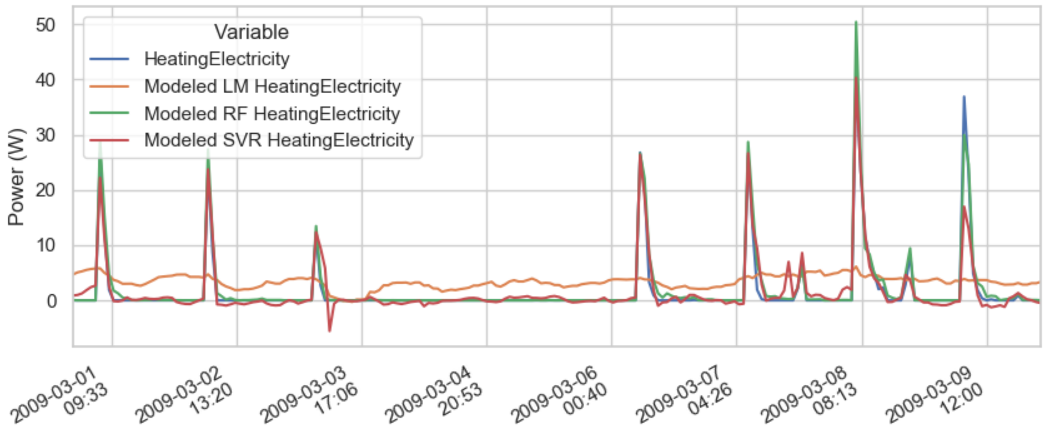


Figure 4.8: Swing season validation period heating electricity performance

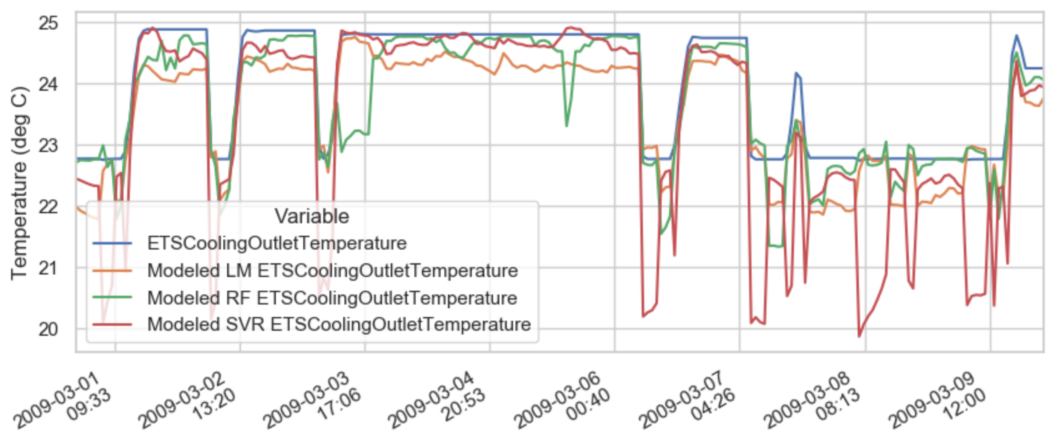


Figure 4.9: Swing season validation period ETS outlet temperature performance

4.1.2 Runtime Performance

There were four metrics of interest in the runtime performance, 1) the size of the persisted model, 2) the time to load the model from disk, 3) the time to run a single datapoint after loading the model, and 4) the time to run 8,760 datapoints after loading the model.

The persisted size of the random forest models were large relative to other models, see Figure 4.10. The figure is presented on a log-scale and shows that the size difference between the support vector regressions is several orders of magnitude larger than the linear models ($\approx$ 3 MB compared to $\approx$ 1 KB). The random forest models are also several orders of magnitude larger than the support vector regression models ($\approx$ 800 MB compared to $\approx$ 3 MB).

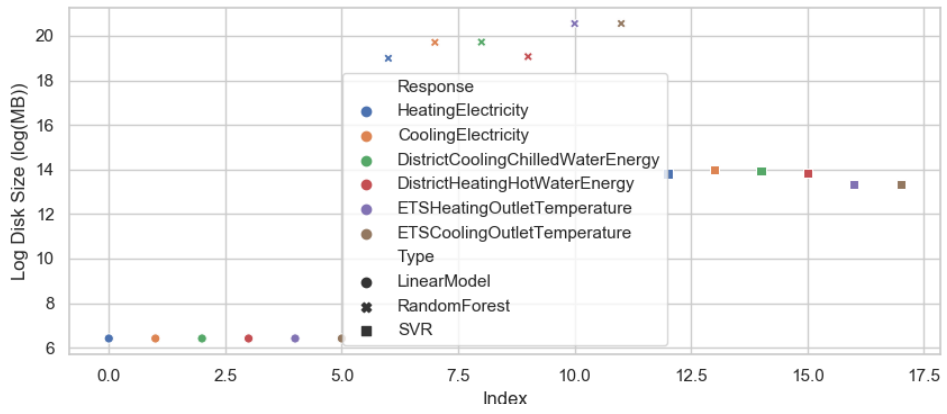


Figure 4.10: Storage size of persisted models

The reason for the large random forest persisted disk size is due to the need to store all of the trees. The support vector regression and the linear model are able to store mathematical representations such as equations and support vectors. Figure 4.11 shows only a portion of the first tree of the random forest which demonstrates the amount of data needed to represent the model. In the case of the hyper-parameter tuned models, there were 200 trees that were chosen with the max depth set to none, this greatly increases the size of the models that need to be persisted.

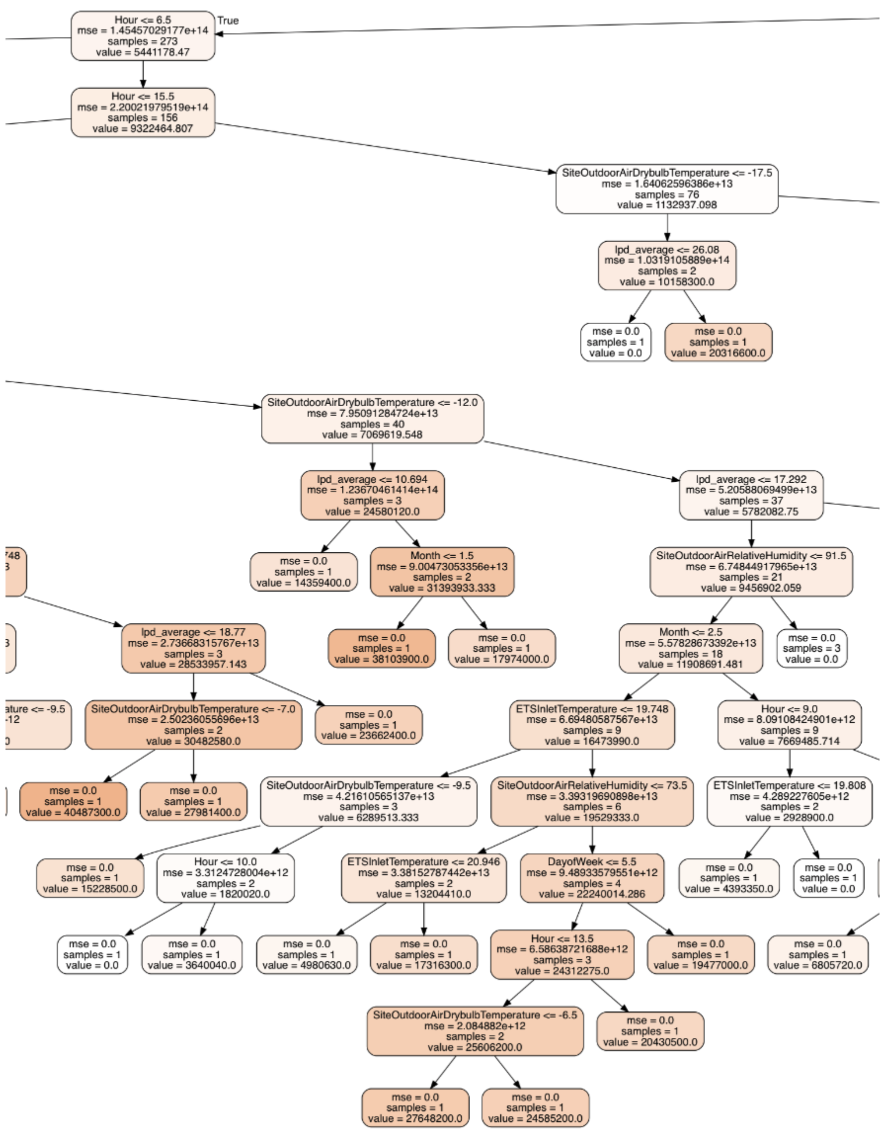


Figure 4.11: Portion of the first tree of the random forest

The remaining three metrics are presented as averages in Table 4.3. As shown the time to load the random forest model is the longest compared to the other models. The support vector regression models took the longest to run an 8,760.

Table 4.3: Load time and runtime of reduced order models

Type	Avg Load Time (Sec)	Avg Run Time - Single (Sec)	Avg Run Time - 8760 (Sec)
Linear Model	0.00034	0.00291	0.00361
Random Forest	0.898	0.0109	0.163
SVR	0.066	0.036	2.56

Figure 4.12 and Figure 4.13 show the load times and runtimes, respectively. The data are shown with log-scale on the time. The support vector regression runtime for the 8,760 datapoints takes substantially longer. This is due in part in the need to pre-process the input data by scaling the data and transforming the cyclical data. There are clear advantage to loading the model once and processing as many datapoints as possible in chunks, instead of loading and processing single datapoints one-by-one.

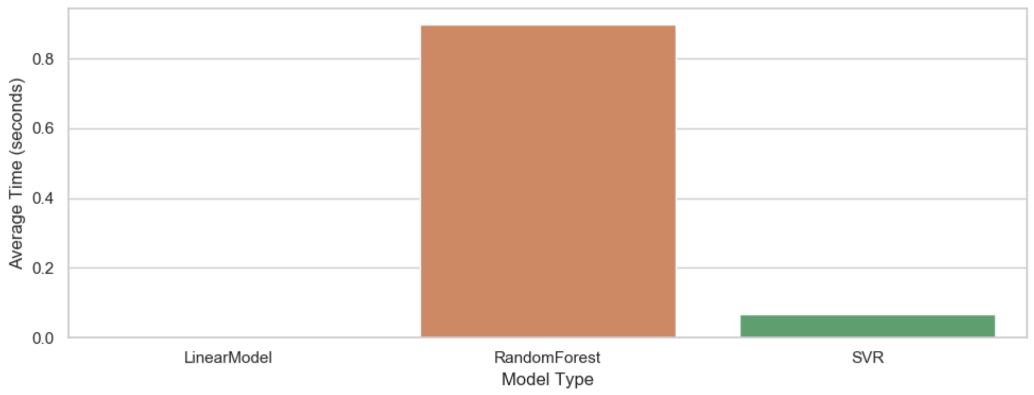


Figure 4.12: Load time as a function of the model type

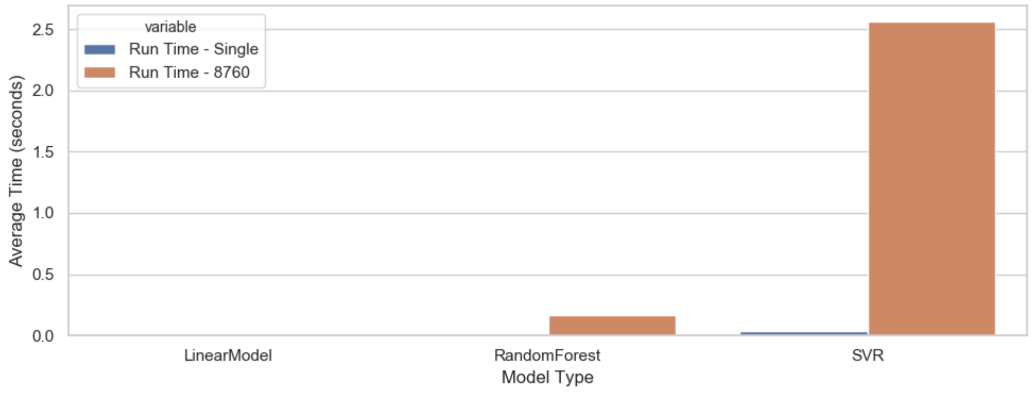


Figure 4.13: Run times as a function of the model type

4.1.3 Down Sampling Performance

Down sampling of the data was performed in order to decrease the build time and to reduce the possibility of over fitting the data. All models were evaluated with 5% and 15% down sampling; whereas only the linear and random forest models were evaluated with 25% and 100% down sampling. The support vector algorithm was unable to handle more than 150,000 samples based on the library chosen to do the analysis.

As the down sample was increased, the PCC did not appreciably increase; however, the build times and persisted disk size increased. Figure 4.14 shows the increases in PCC based on all of the models. The type of model is not important in the shown figure, rather, the lack of any significant increase as the down sampling rate increased shows that the models are not necessarily better based on a larger down sample size.

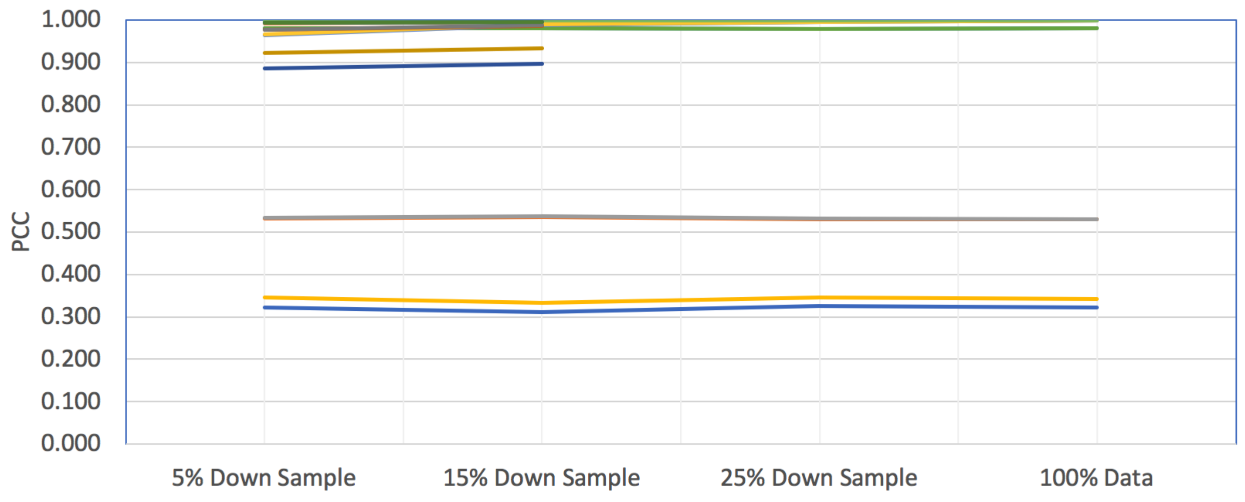


Figure 4.14: PCC change based on down sample (Note: SVR only down sampled for 5% and 15%)

The use of down sampling decreased the size of the persisted models for the random forest by more than one-quarter. The average size of the random forest persisted model for 5% down sample was 300 MB whereas for the 25% down sample it was around 2 GB. Having a smaller persisted disk size also reduced the load time of the model. Since the performance of the models did not vary significantly when using the lower down sampling percentage, it is recommended to use the lowest value possible when running analyses with reduced order models.

4.2 ROM Framework

A ROM Framework was developed in order to build, evaluate, validate, and run the reduced order models. The framework is written in Python. The interface is currently a command line utility that can be installed via source code [60].

The first stage of the ROM Framework is to build and evaluate the models. The ROM Framework consumes a generic CSV file where each row is a sample or datapoint. An accompanying JSON file is used to describe the data and options for the reduced order modeling algorithms. The suggested workflow for using the ROM Framework is shown in Figure 4.15.

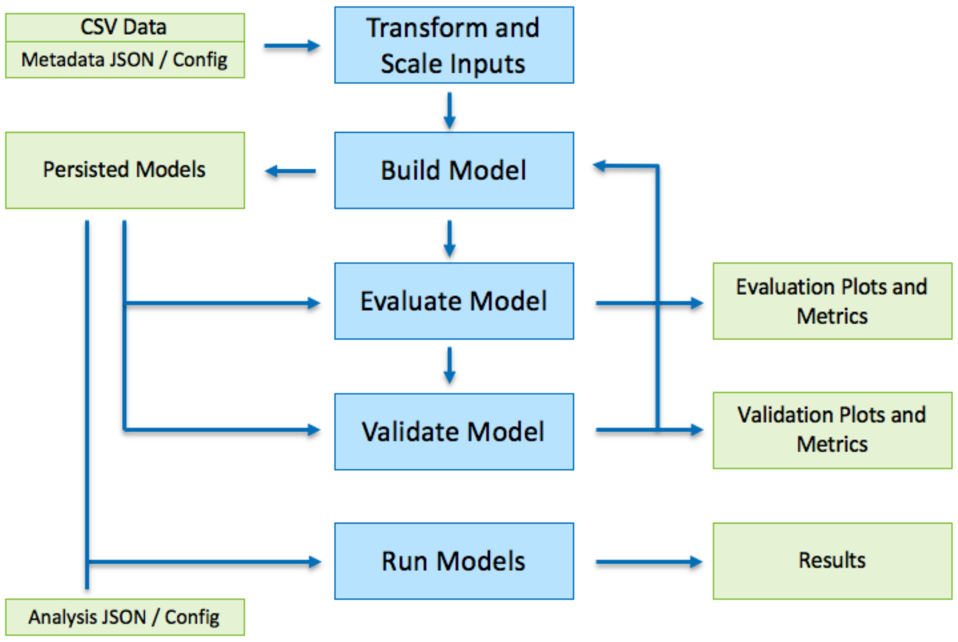


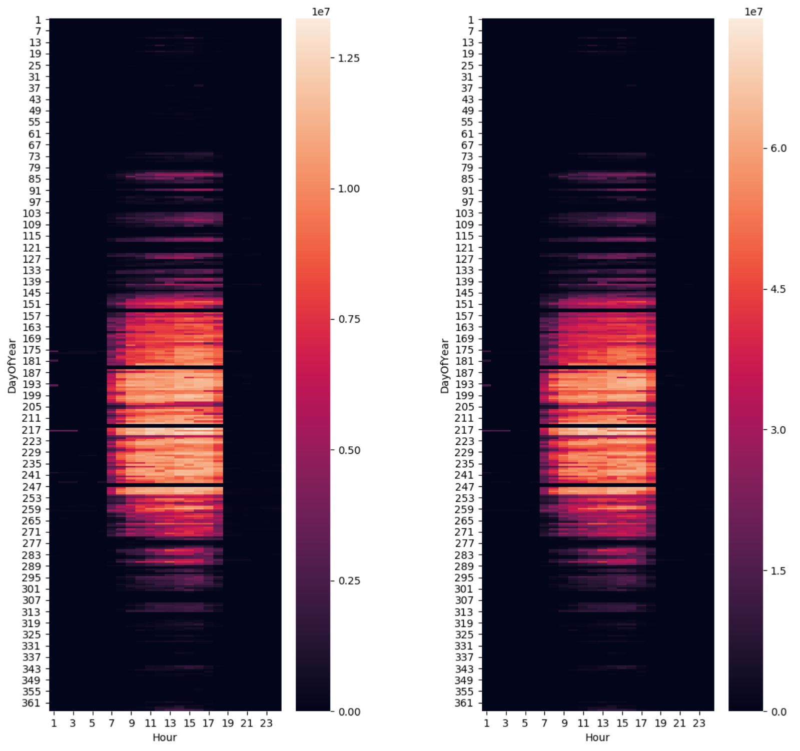
Figure 4.15: Flowchart for building and running reduced order models

The first step when building and evaluating the models is to define the variables in the CSV file. The definition is saved in a specific format as defined by the metadata JSON file. The file describes where the results are stored, the name of the analysis, which reduced order models to build/evaluate, any configuration option on the reduced order modeling algorithms, the down sampling values (if needed), a list of covariates and their transformations, and, lastly, the response variables. An example of the metadata.json file can be found in the ROM Framework documentation. Information on accessing the documentation is found in Appendix D: ROM Framework Information. The CSV and metadata JSON file are passed into the ROM Framework and, depending on the algorithm, undergo transformation and scaling. The reduced order modeling algorithm then builds the model and persists the models to disk, as well as some high-level diagnostic plots and metric CSV files. The user is then able to run an evaluation and a validation task on the persisted models to further investigate model performance. The result of these tasks are diagnostic plots and CSV files. Lastly, the user has the ability to run any of the persisted models using an analysis definition JSON file. An example of the file is shown in Figure 4.16.

```json
{
  "variables": [
    {
      "name": "Month",
      "data_source": "epw",
      "data_source_field": "month"
    },
    {
      "name": "Hour",
      "data_source": "epw",
      "data_source_field": "hour"
    },
    {
      "name": "DayofWeek",
      "data_source": "epw",
      "data_source_field": "day"
    },
    {
      "name": "SiteOutdoorAirDrybulbTemperature",
      "data_source": "epw",
      "data_source_field": "dry_bulb"
    },
    {
      "name": "SiteOutdoorAirRelativeHumidity",
      "data_source": "epw",
      "data_source_field": "rh"
    },
    {
      "name": "ETSInletTemperature",
      "data_source": "value",
      "value": 20
    },
    {
      "name": "lpd_average",
      "data_source": "value",
      "value": 9.4
    }
  ]
}
```

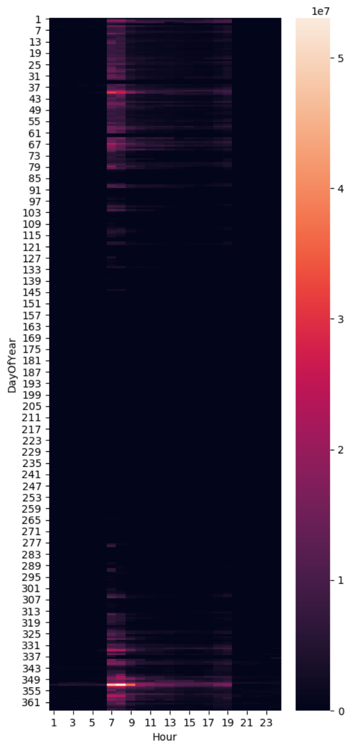
Figure 4.16: Analysis definition example input file

The example analysis definition file allows the user to specify either fixed values (e.g. ETSInletTemperature) or values from an EnergyPlus weather file (e.g. Month, Hour, SiteOutdoorAirDrybulbTemperature). The user is also able to specify a distribution or specific array of values to sweep over. In the example above, the analysis generated results in order to visualize the cooling and heating energy use over 8,760 hours with a fixed ETS inlet temperature of 20°C and LPD of 9.4 W/m^2. The random forest reduced order model was used in the example. Figure 4.17 and Figure 4.18 show the heat maps of the cooling energy and heating energy, respectively.

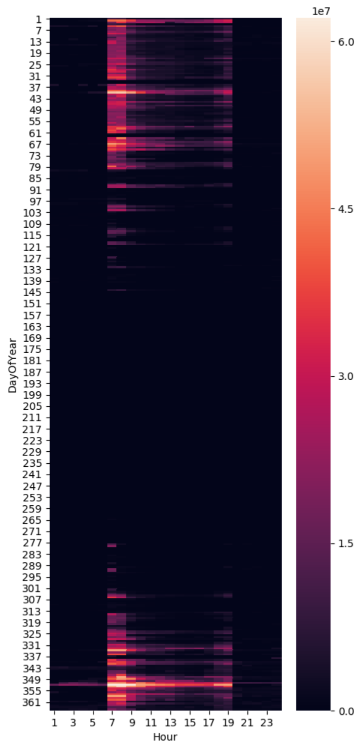


(a) Cooling Electricity (b) District Chilled Water Energy

Figure 4.17: Flood plot of random forest model with fixed inlet temperature



(a) Heating Electricity



(b) District Hot Water Energy

Figure 4.18: Flood plot of random forest model with fixed inlet temperature

As shown in the images, with a fixed ETS inlet temperature, the cooling energy consumption was higher than the heating overall. During the cooling season, the energy extracted from the ambient loop was around five times higher than the electrical energy provided by the water-to-water heat pump's compression cycle. There are several periods in the heating heat map where there was no heating needed, and in many cases, the district hot water was covering the majority of the heating demand (note that the scales between the two plots are slightly different). The ROM Framework allows the user to quickly run these type of analyses without needing to fully articulate an EnergyPlus model.

4.2.1 Integrating Models with Other Tools

One of the main outcomes of this thesis was to create a method for connecting the reduced order models with other tools (namely Modelica). There were three methods that were investigated:

(1) calling from a command line interface (CLI) with passed arguments

(2) called directly from Python by loading a custom developed library

(3) storing results in an n-dimension comma-separated values (CSV) files and loaded into the analysis tool

4.2.1.1 Command Line Interface

The ROM Framework has a CLI which consumes an analysis definition file; however, a simpler CLI was created to connect third-party applications without needing the analysis definition file. An example is shown in Figure 4.19. The CLI takes several arguments to define 1) the building type (analysis_id), 2) the ROM model (model), 3) the response variable, and 4) all of the covariates needed to perform the predict procedure. This is a very customized approach to predict any response variables because any changes to the covariates (i.e. adding/removing) will require an update to the CLI. The other limitation to this method is that each CLI call requires fully loading the persisted reduced order models from disk which can be prohibitive as in the case of the random forest models. Lastly, the result of this CLI is simply printed to the screen; therefore, any third-party application calling the CLI will need to capture STDOUT and parse the result.

```
parser = argparse.ArgumentParser()
parser.add_argument('-f', '--file', help='Description file to use'
    , default='metamodels.json')
parser.add_argument('-a', '--analysis_id', default='
    smoff_parametric_sweep', help='ID of the Analysis Models')
parser.add_argument('--model_type', default='RandomForest',
    choices=['LinearModel', 'RandomForest', 'SVR'])
parser.add_argument('-r', '--responses', nargs='*', default=['
    HeatingElectricity'], help='List of responses')
parser.add_argument('-d', '--day_of_week', type=int, default=0,
    help='Day of Week: 0-Sun to 6-Sat')
parser.add_argument('-m', '--month', type=int, default=1, help='
    Month: 1-Jan to 12-Dec')
parser.add_argument('-H', '--hour', type=int, default=9, help='
    Hour of Day: 0 to 23')
parser.add_argument('-T', '--outdoor_drybulb', type=float, default
    =-5, help='Outdoor Drybulb Temperature')
parser.add_argument('-RH', '--outdoor_rh', type=float, default=50,
     help='Percent Outdoor Relative Humidity')
parser.add_argument('-i', '--inlet_temp', type=float, default=20,
    help='Inlet Temperature')
parser.add_argument('--lpd', type=float, default=9.5, help='
    Lighting Power Density (W/m2)')
args = parser.parse_args()

rom = Metamodels(args.file)
rom.set_analysis(args.analysis_id)
rom.load_models(args.model_type, models_to_load=args.responses,
    root_path='smoff')

data = {
    'Month': args.month,
    'Hour': args.hour,
    'DayofWeek': args.day_of_week,
    'SiteOutdoorAirDrybulbTemperature': args.outdoor_drybulb,
    'SiteOutdoorAirRelativeHumidity': args.outdoor_rh,
    'lpd_average': args.lpd,
    'ETSInletTemperature': args.inlet_temp,
}
df = pd.DataFrame([data])
for response in args.responses:
    v = rom.yhat(response, df)
    print(v[0])
```

Figure 4.19: Example CLI implementation

This CLI method was originally proposed for Modelica, however, Modelica is unable to pass categorical variables with ease; therefore, the second approach was pursued as discussed in the next subsection.

4.2.1.2 Python Method

The second approach of connecting third-party tools was exposing a Python method. In the case of Modelica, a Python method can be accessed at each time step in the Modelica simulation. Figure 4.20 shows an example of how the method *run_model* was created for use in Modelica. As shown, the argument to the method is an array of numbers (either floats or integers). Modelica's library does not easily allow strings or categorical variables, therefore, the enumerations were mapped to integers in Modelica and the Python method unmapped the values as needed. This method was non-ideal and hopefully, future versions of Modelica can expand the Python-method implementation to be more flexible.

The advantage of the Python-based method is the ability for a third-party application to directly call and receive the result of the method without needing to parse STDOUT. This method though suffers from the same need as the CLI to load the persisted reduced order model every time step. The ability to preload the reduced order models in Modelica was investigated but did not work as expected, thus, further research is needed.

```python
def run_model(values):
    model = int(values[0])
    response = int(values[1])
    data = {
        'Month': int(values[2]),
        'Hour': int(values[3]),
        'DayofWeek': int(values[4]),
        'SiteOutdoorAirDrybulbTemperature': float(values[5]),
        'SiteOutdoorAirRelativeHumidity': float(values[6]),
        'lpd_average': float(values[7]),
        'ETSInletTemperature': float(values[8]),
    }

    # Convert the model integer to correct ROM type
    if model == 1:
        model = 'LinearModel'
    elif model == 2:
        model = 'RandomForest'
    elif model == 3:
        model = 'SVR'

    # Convert the response integer to the correct type
    if response == 1:
        response = 'CoolingElectricity'
    elif response == 2:
        response = 'HeatingElectricity'
    elif response == 3:
        response = 'DistrictCoolingChilledWaterEnergy'
    elif response == 4:
        response = 'DistrictHeatingHotWaterEnergy'
    elif response == 5:
        response = 'ETSOutletTemperature'

    rom = load_models('smoff/metamodels.json', model, [response])

    print('Predicting...')
    df = pd.DataFrame([data])
    print(rom.yhat(response, df)[0])
    v = rom.yhat(response, df)[0]
    print(f'Predicted value is {v}')

    return v
```

Figure 4.20: Example Python method implementation

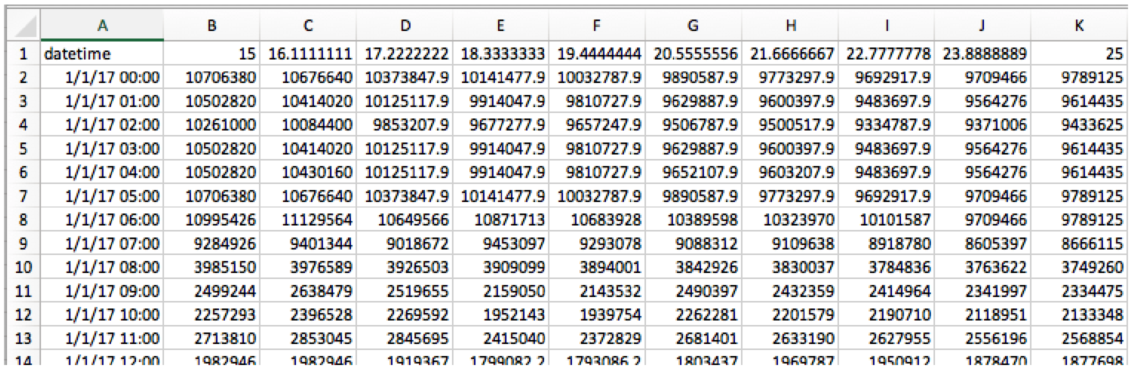
4.2.1.3 *n*-Dimensional CSV Export

The last method to interconnect third-party applications was simply exporting multidimensional CSV files. The ROM Frameworks analysis definition interface was designed for this purpose. The analysis definition file can specify distributions of data and the ROM Framework can export multiple CSV files based on a pivot variable. Figure 4.21 shows an example of how the ETS inlet temperature will be sampled 10-times inclusively between 15 and 25°C.

```
{
  "name": "ETSInletTemperature",
  "data_source": "distribution",
  "distribution": {
    "minimum": 15,
    "maximum": 25,
    "number_of_samples": 10
  }
}
```

Figure 4.21: Analysis definition with distribution

The analysis definition can handle more than one variable distribution which results in the full combinatorial of all the variables. The ROM Framework will then return CSV files based on each of the swept variables. In the case of the ETS inlet temperature, the result will be a table with inlet temperatures across the top, date times on the y-axis, and the matrix will be the value of the response variable. There is one file for each response variable. Figure 4.22 shows a view of one of the exported CSV files. This file was used to approximate the heating electricity of the small office building.



	A	B	C	D	E	F	G	H	I	J	K
1	datetime	15	16.1111111	17.2222222	18.3333333	19.4444444	20.5555556	21.6666667	22.7777778	23.8888889	25
2	1/1/17 00:00	10706380	10676640	10373847.9	10141477.9	10032787.9	9890587.9	9773297.9	9692917.9	9709466	9789125
3	1/1/17 01:00	10502820	10414020	10125117.9	9914047.9	9810727.9	9629887.9	9600397.9	9483697.9	9564276	9614435
4	1/1/17 02:00	10261000	10084400	9853207.9	9677277.9	9657247.9	9506787.9	9500517.9	9334787.9	9371006	9433625
5	1/1/17 03:00	10502820	10414020	10125117.9	9914047.9	9810727.9	9629887.9	9600397.9	9483697.9	9564276	9614435
6	1/1/17 04:00	10502820	10430160	10125117.9	9914047.9	9810727.9	9652107.9	9603207.9	9483697.9	9564276	9614435
7	1/1/17 05:00	10706380	10676640	10373847.9	10141477.9	10032787.9	9890587.9	9773297.9	9692917.9	9709466	9789125
8	1/1/17 06:00	10995426	11129564	10649566	10871713	10683928	10389598	10323970	10101587	9709466	9789125
9	1/1/17 07:00	9284926	9401344	9018672	9453097	9293078	9088312	9109638	8918780	8605397	8666115
10	1/1/17 08:00	3985150	3976589	3926503	3909099	3894001	3842926	3830037	3784836	3763622	3749260
11	1/1/17 09:00	2499244	2638479	2519655	2159050	2143532	2490397	2432359	2414964	2341997	2334475
12	1/1/17 10:00	2257293	2396528	2269592	1952143	1939754	2262281	2201579	2190710	2118951	2133348
13	1/1/17 11:00	2713810	2853045	2845695	2415040	2372829	2681401	2633190	2627955	2556196	2568854
14	1/1/17 12:00	1982946	1982946	1919367	1799082.2	1793086.2	1803437	1969787	1950912	1878470	1877698

Figure 4.22: CSV example export

Ultimately, this method was chosen for the Modelica integration due to how fast Modelica could load and interpolate values from CSV files compared to the overhead in loading the reduced order models every time step.

Chapter 5

Summary and Conclusions

In the United States, district heating and cooling systems are widely underutilized. Based on the literature review, around 43% of the heating demands in the United States could be met using district heating systems. Conventional district heating systems use high-temperature water or steam to provide heat to the buildings. These systems have high system losses as well as few available types of heat source. More recently, there has been an effort to further reduce the district heating system water temperatures to near ambient temperatures. The lower temperatures reduce the system losses and enable additional heating sources such as the heat rejected from commercial refrigeration system and data centers.

This thesis discussed the background of district energy systems, explored modeling techniques for buildings, and provided a new modeling strategy for quickly evaluating district energy systems. A reduced order modeling framework (ROM Framework) was developed to build, evaluate, and validate various reduced order modeling algorithms. The ROM Framework is able to consume any building energy modeling result file (as a CSV) and generate reusable reduce order models based on the supplied covariates and response variables. The overarching goals were to:

(1) provide a framework for generating and evaluating various reduced order models for ambient loop analysis,

(2) estimate building energy loads based on various covariates (such as inlet ambient loop supply temperature, date and time, and various building characteristics),

(3) evaluate the performance of the reduced order models including creation time, load time, execution time, and accuracy.

5.1 Building Energy Models

Two building types were evaluated, a retail and small office. The majority of the results presented are for the small office. The results for the retail building are provided in Appendix B: Retail Building Analysis Results. The building energy models were generated using OpenStudio and EnergyPlus. A parametric analysis was run to create a diverse set of buildings based on the district energy inlet temperatures, lighting power density, and various date and time metrics.

5.2 Reduced Order Models

Three reduced order modeling algorithms were evaluated: linear model, random forest, and support vector regression. The originating building energy modeling results were a large dataset with nearly a million samples. The reduced order modeling algorithms were able to fit reasonable models but in the case of support vector regression, was unable to mathematically compute the vectors on the full sample. Table 5.1 summarizes the small office average Pearson's correlation coefficient (PCC) based on the reduced order model type. As shown the PCC of both the random forest and SVR models performed very well (greater than 0.9) on average; however, the build and cross-validation time for the random forest was significantly lower than that of the SVR.

Table 5.1: Average model performance

Model Type	Avg PCC	Avg Build Time	Avg CV Time	Avg Load Time	Avg Run Time
Linear Model	0.62	0.02	0.00	0.0002	0.003
Random Forest	0.99	3.77	981.1	0.739	0.011
SVR	0.96	39.2	2708	0.061	0.033

Depending on the use case of the reduced order models, it may be reasonable to use the SVR model due to quicker load times compared to the random forest. Another option would be to down sample the random forest model even more in order to reduce the size of the persisted model since the disk size of the model is directly proportional to the size of the forest.

Overall, the performance of the linear model was poor for the energy response variables; however performed well for the temperature variable. Figure 5.1 shows the HVAC energy as a function of outdoor dry-bulb temperature for the three reduced order models. The difference is indistinguishable between the random forest and SVR; however, the linear model is obviously problematic.

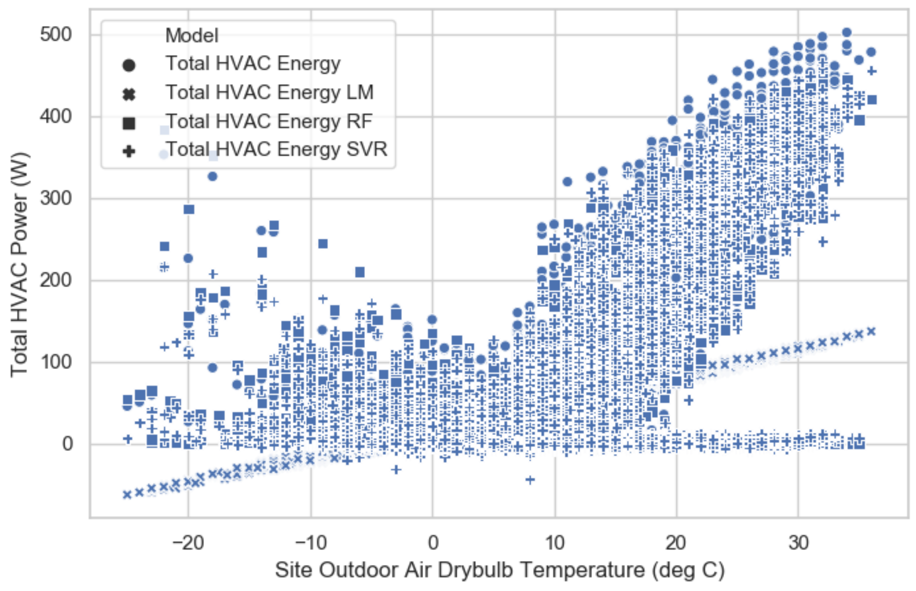


Figure 5.1: Energy consumption vs outdoor dry-bulb temperature

The ability to evaluate different reduced order models, such as linear models, random forests, and support vector machines was easily accomplished using the developed ROM Framework.

The ROM Framework also enabled the integration of the reduced order models with system modeling tools such as Modelica. Three different methods of tool integration were investigated, CLI interface, direct python method, and n-dimensional CSV files. The three options were explored with Modelica to determine the best integration path. Due to Modelica's need to load the models at every time step, the CSV files were initially chosen. Extensions to Modelica and the Modelica building's library are being investigated to load the reduced order models upon model initialization; therefore, the only time incurred would be the model runtime for a single datapoint which is small (0.05 seconds). Additionally, since the linear model performed well for the ETS outlet temperature, the model could easily be programmed as a simple C function in Modelica.

5.3 Future Work

The ability to use quickly executed reduced order models for evaluating district energy power and supplemental power needed for heating and cooling allows for analyses that would be otherwise time prohibitive. A district energy system network topology analysis is possible using the reduced order modeled since the ETS inlet temperature is one of the covariates. The questions that can be answered in the near future using these models include:

(1) Which buildings should be connected to the 5GDHC system?

(2) What is the best network topology for the 5GDHC?

(3) What should be the optimal network topology if new heating or cooling sources are added to the loop?

(4) If operating schedules change for a building, should the building still be connected?

(5) How should we control the buildings?

(6) What should the current supply temperature of the 5GDHC loop be at a specific date and time?

There are three areas for future work including 1) extending building energy models and covariates, 2) extending the ROM Framework functionality, and 3) integrating reduced order models with third-party applications.

5.3.1 Extending Building Energy Models and Covariate

The ability to run different buildings would allow for more comprehensive network topology analyses due to a larger building selection. There are several building energy model extensions and improvements that could benefit from future work including:

(1) *Generic building types*: Currently, the building energy models used the DOE Prototype buildings. These buildings have fixed geometry and dimensions. There are some generic

OpenStudio measures that exist that are able to create buildings based on requested floor areas and aspect ratios. Using these generic building types would allow for more flexibility.

(2) *Expose additional covariates*: Covariates are directly linked to OpenStudio measures. Additional covariates allow for flexibility with the reduced order models and ultimately would allow for more load diversity in the network topology analyses. An example of an additional covariate would be a "schedule shifter" which would take a building's operating schedule to dynamically create the many needed modeling schedules.

(3) *Update OpenStudio Server and OpenStudio Standards*: OpenStudio Server and OpenStudio Standards are actively developed projects and inevitably the ambient loop measures that were developed as part of this thesis will need to be updated.

5.3.2 Extending ROM Framework

Presently, the ROM Framework only evaluated linear models, random forests, and support vector regressions. There are hundreds of other methods that could be used to create reasonably performing reduced order models. The ROM Framework is designed with a base class for "easy" extension. Future work for the ROM Framework includes:

(1) *Additional Models*: Add additional models to be evaluated such as artificial neural networks, classification and regression trees, autoregressive moving average (ARMA), wavelets, etc.

(2) *Additional Parameters*: Expose various model parameters based on the type of reduced order model. For example, allow for different basis functions or log-based normalizations.

(3) *More diagnostic plots and tabular data*: Hundreds of images and CSV files are created during the building the reduced order models. There is plenty of research needed to find better ways to present the performance results.

(4) *Build Time Performance*: Decrease the time to build and evaluate the reduced order models by using optimization for cross-validation instead of grid search.

(5) *Load Time Performance*: The best performing reduced order model takes too long to load and more research is needed to reduce the persisted size of the models thereby decreasing the load time.

5.3.3 Integration of Reduced Order Models with Third-party Applications

The ROM Framework was designed to easily load an already built reduced order model; however, the ability for the ROM Framework to integrate with generic third-party applications is still lacking. Three example scripts were created to demonstrate integration but future work could include:

(1) *Generic CLI*: Add a method to the ROM Framework that translates the variables in the metamodels.json file to dynamically create a CLI.

(2) *Generic Python Method*: Currently the Python method requires an integer-based mapping to categorical variables. A reasonable extension would be to dynamically convert the integers to the categorical values based on a definition in the metadata.json file.

Bibliography

[1] B. Polly, C. Kutscher, D. Macumber, and M. Schott, "From Zero Energy Buildings to Zero Energy Districts," in ACEEE Summer Study on Energy Efficiency in Buildings, (Asilomar, CA), pp. 1–16, 2016.

[2] L. Pérez-Lombard, J. Ortiz, and C. Pout, "A review on buildings energy consumption information," Energy and Buildings, vol. 40, pp. 394–398, jan 2008.

[3] L. Hong, N. Zhou, D. Fridley, W. Feng, and N. Khanna, "Modeling China's Building Floor-Area Growth and the Implications for Building Materials and Energy Demand," in 2014 ACEEE Summer Study on Energy Efficiency in Buildings, (Pacific Grove, CA), 2014.

[4] Tsinghua University Building Energy Research Center, "China Building Energy Use 2016," Tech. Rep. July, Building Energy Research Center Tsinghua University, 2016.

[5] EIA (Energy Information Administration), "Commercial Buildings Energy Consumption Survey (CBECS)," tech. rep., Department of Energy (DOE), 2012.

[6] EIA (Energy Information Administration), "Residential Energy Consumption Survey (RECS)," tech. rep., Department of Energy (DOE), 2015.

[7] EIA (Energy Information Administration), "U.S. District Energy Services Market Characterization," tech. rep., ICF L.L.C. and the International District Energy Association, Washington, DC, 2018.

[8] H. Lund, S. Werner, R. Wiltshire, S. Svendsen, J. E. Thorsen, F. Hvelplund, and B. V. Mathiesen, "4th Generation District Heating (4GDH). Integrating smart thermal grids into future sustainable energy systems.," Energy, vol. 68, pp. 1–11, apr 2014.

[9] J. von Rhein, "Modeling, Simulation, and Life-Cost Analysis of Fifth-Generation District Heating and Cooling Networks," 2018.

[10] J. Wagner and S. P. Kutska, "District Energy Denver's Historic System: Still Full Steam Ahead," International District Energy Association, vol. 94, no. 4, pp. 16–20, 2008.

[11] C. Winterscheid, J.-O. Dalenbäck, and S. Holler, "Integration of solar thermal systems in existing district heating systems," Energy, vol. 137, pp. 579–585, oct 2017.

[12] M. Wahlroos, M. Pärssinen, J. Manner, and S. Syri, "Utilizing data center waste heat in district heating Impacts on energy efficiency and prospects for low-temperature district heating networks," Energy, vol. 140, pp. 1228–1238, dec 2017.

[13] V. Mikler, “District Energy 101,” tech. rep., Integral Group, Vancouver, 2017.

[14] A. Lake, B. Rezaie, and S. Beyerlein, “Review of district heating and cooling systems for a sustainable future,” Renewable and Sustainable Energy Reviews, vol. 67, pp. 417–425, 2017.

[15] H. C. Gils, J. Cofala, F. Wagner, and W. Schöpp, “GIS-based assessment of the district heating potential in the USA,” Energy, vol. 58, pp. 318–329, 2013.

[16] T. Hong, F. Buhl, and P. Haves, “EnergyPlus Run Time Analysis,” tech. rep., Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA, apr 2009.

[17] IBPSA-USA, “Building Energy Software Tools.”

[18] A. Foucquier, S. Robert, F. Suard, L. Stéphan, and A. Jay, “State of the art in building modelling and energy performances prediction: A review,” Renewable and Sustainable Energy Reviews, vol. 23, pp. 272–288, jul 2013.

[19] F. Winkelmann, B. Birdsall, W. Buhl, K. Ellington, and A. Erdem, “DOE-2 Supplement: Version 2.1E,” 1993.

[20] S. A. Klein, W. A. Beckman, and J. A. Duffie, “TRNSYS: A Transient Simulation Program,” ASHRAE Transactions, vol. 82, 1976.

[21] ESRU, “ESP-r.”

[22] IES, “VE for Architects and Engineers.”

[23] DOE, “EnergyPlus Energy Simulation Software.”

[24] F-Chart Software, “EES: Engineering Equation Solver.”

[25] Modelica Association, “Modelica,” 2000.

[26] D. Ruch and D. E. Claridge, “A Four-Parameter Change-Point Model for Predicting Energy Consumption in Commercial Buildings,” Journal of Solar Energy Engineering, vol. 114, no. 2, p. 77, 1992.

[27] M. Kavgic, A. Mavrogianni, D. Mumovic, A. Summerfield, Z. Stevanovic, and M. Djurovic-Petrovic, “A review of bottom-up building stock models for energy consumption in the residential sector,” Building and Environment, vol. 45, pp. 1683–1697, jul 2010.

[28] A. Burr, “Energy Rating and Disclosure in the United States and Europe,” tech. rep., Institute for Market Transformation, 2012.

[29] S. Goel, N. Wang, H. Horsey, and N. Long, “Building Efficiency Evaluation and Uncertainty Analysis with DOE’s Asset Score Preview,” in ASHRAE and IBPSA-USA SimBuild 2016 Building, (Salt Lake City, UT), pp. 173–180, 2016.

[30] A. P. Melo, D. Cóstola, R. Lamberts, and J. L. M. Hensen, “Development of surrogate models using artificial neural network for building shell energy labelling,” Energy Policy, vol. 69, pp. 457–466, 2014.

[31] A. Melo, R. Versage, G. Sawaya, and R. Lamberts, "A novel surrogate model to support building energy labelling system: A new approach to assess cooling energy demand in commercial buildings," Energy and Buildings, vol. 131, pp. 233–247, nov 2016.

[32] Q. Li, Q. Meng, J. Cai, H. Yoshino, and A. Mochida, "Predicting hourly cooling load in the building: A comparison of support vector machine and different artificial neural networks," Energy Conversion and Management, vol. 50, pp. 90–96, 2009.

[33] S. A. Kalogirou and M. Bojic, "Artificial neural networks for the prediction of the energy consumption of a passive solar building," Energy, vol. 25, pp. 479–491, may 2000.

[34] I. Jaffal, C. Inard, and C. Ghiaus, "Fast method to predict building heating demand based on the design of experiments," Energy and Buildings, vol. 41, pp. 669–677, jun 2009.

[35] A. N. Aijazi and L. R. Glicksman, "Comparison of regression techniques for surrogate models of building energy performance," in ASHRAE and IBPSA-USA SimBuild 2016, pp. 327–334, 2016.

[36] T. Ostergard, R. L. Jensen, and S. E. Maagaard, "A comparison of six metamodeling techniques applied to building performance simulations," Applied Energy, vol. 211, pp. 89–103, feb 2018.

[37] A. Caucheteux, A. Gautier, and R. Lahrech, "A meta model-based methodology for an energy savings uncertainty assessment of building retrofitting," International Journal of Metrology and Quality Engineering, vol. 7, p. 402, oct 2016.

[38] L. Van Gelder, P. Das, H. Janssen, and S. Roels, "Comparative study of metamodelling techniques in building energy simulation: Guidelines for practitioners," Simulation Modelling Practice and Theory, vol. 49, pp. 245–257, dec 2014.

[39] M. Krarti, D. Cohen, A. Dhar, T. A. Reddy, and D. E. Claridge, "Estimation of energy savings for building retrofits using neural networks," Journal of Solar Energy Engineering, Transactions of the ASME, vol. 120, no. 3, pp. 217–223, 1998.

[40] S. A. Kalogirou, "Applications of artificial neural-networks for energy systems," Applied Energy, vol. 67, pp. 17–35, sep 2000.

[41] J. S. Hygh, J. F. DeCarolis, D. B. Hill, and S. Ranji Ranjithan, "Multivariate regression as an energy assessment tool in early building design," Building and Environment, vol. 57, pp. 165–175, nov 2012.

[42] Z. Shi and W. O'Brien, "Building energy model reduction using model-cluster-reduce pipeline," Journal of Building Performance Simulation, pp. 1–15, dec 2017.

[43] B. Eisenhower, Z. O'Neill, S. Narayanan, V. A. Fonoberov, and I. Mezić, "A methodology for meta-model based optimization in building energy models," Energy and Buildings, vol. 47, pp. 292–301, apr 2012.

[44] B. Dong, C. Cao, and S. E. Lee, "Applying support vector machines to predict building energy consumption in tropical region," Energy and Buildings, vol. 37, pp. 545–553, may 2005.

[45] M. Wetter, W. Zuo, T. S. Nouidui, and X. Pang, "Modelica Buildings Library," Journal of Building Performance Simulation, vol. 7, pp. 253–270, jul 2014.

[46] R. Guglielmetti, D. Macumber, and N. L. Long, "Openstudio: An open source integrated analysis platform," in Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association, 2011.

[47] R. Guglielmetti and B. L. Ball, "A Framework for Daylighting Optimization in Whole Buildings With OpenStudio," in ASHRAE and IBPSA-USA SimBuild 2016, (Salt Lake City, UT), pp. 33–40, 2016.

[48] NREL, "Parametric Analysis Tool Version 2 (PAT) Interface Guide - OpenStudio® User Docs."

[49] N. Long, B. Ball, K. Fleming, and D. Macumber, "Scaling Building Energy Modeling Horizontally in the Cloud with OpenStudio," in ASHRAE/IBPSA-USA Building Simulation Conference, pp. 80–86, 2014.

[50] A. Roth, D. Goldwasser, and A. Parker, "There's a measure for that!," Energy and Buildings, vol. 117, pp. 321–331, apr 2016.

[51] A. Roth, "New OpenStudio-Standards Gem Delivers One Two Punch," 2016.

[52] DOE (Department of Energy), "Commercial Prototype Building Models."

[53] NREL, "OpenStudio Analysis Framework," 2018.

[54] N. L. Long, H. Horsey, and D. Macumber, "OpenStudio Workflow Gem," 2014.

[55] T. Simpson, J. Poplinski, P. N. Koch, and J. Allen, "Metamodels for Computer-based Engineering Design: Survey and recommendations," Engineering with Computers, vol. 17, pp. 129–150, jul 2001.

[56] L. Cline and J. Harshaw, "Commercial Geothermal Is Heating Up!," Trane Engineers Newsletter, vol. 40, no. 1, p. 5, 2011.

[57] J. Markley, M. Pritoni, and P. Fortunato, "HVAC Equipment Demographics and Capacity Analysis Tools Applicable to Multi-tenant Light Commercial Buildings," tech. rep., Western Cooling Efficiency Center - UC Davis, Davis, CA, 2012.

[58] L. Breiman, "Random Forests," Machine Learning, vol. 45, pp. 5–32, oct 2001.

[59] A. J. Smola and B. Scholkopf, "A tutorial on support vector regression," tech. rep., 2004.

[60] N. Long, "ROM Framework," 2018.

Appendix A

Supporting Analysis Results

A.1 Ambient Loop HVAC System

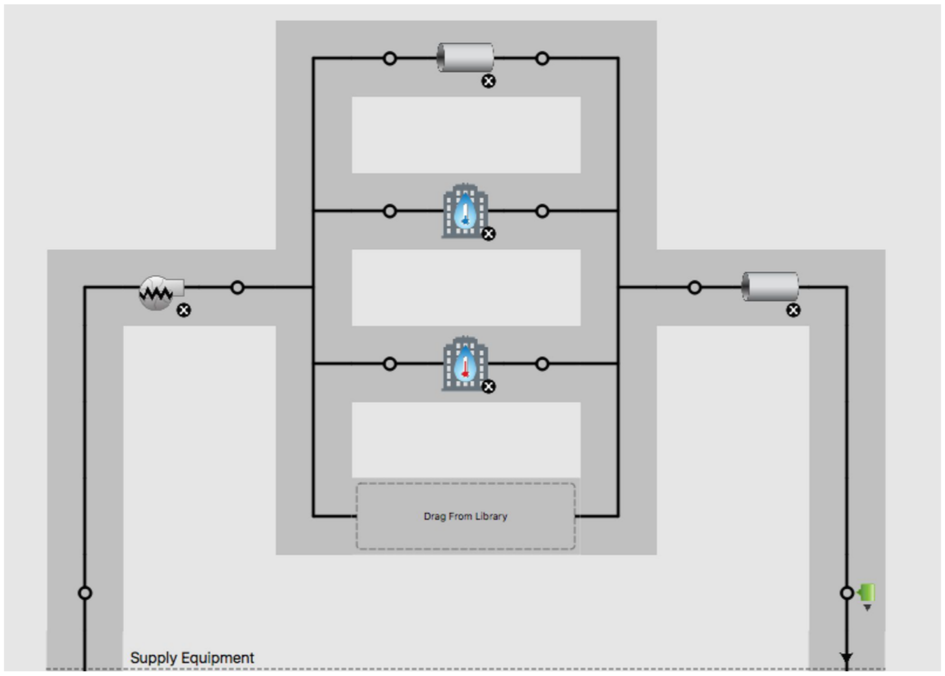


Figure A.1: Ambient loop supply loop

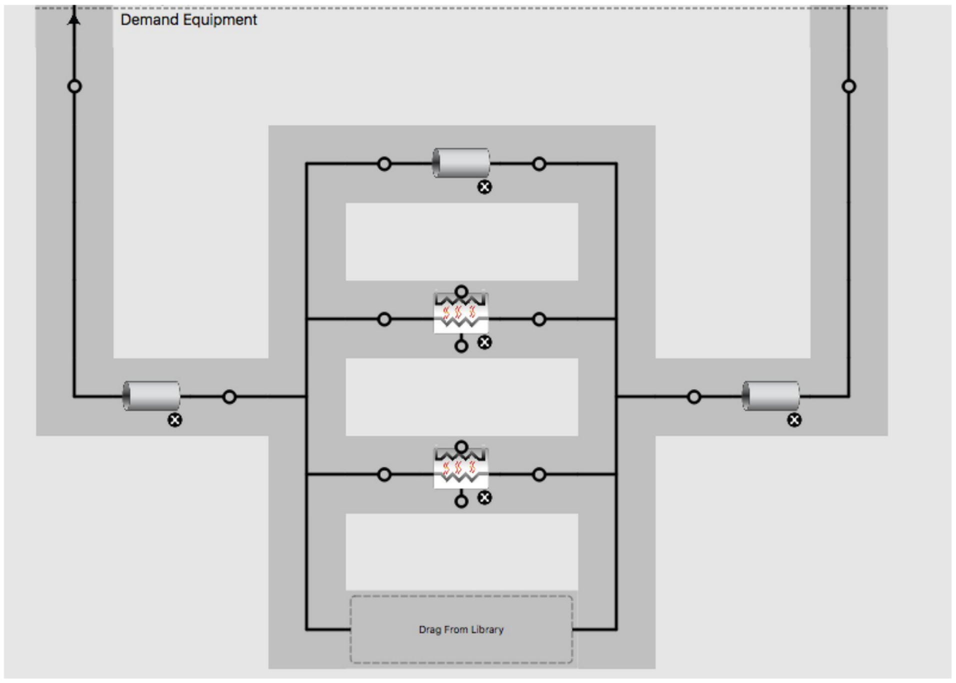


Figure A.2: Ambient loop demand loop with water-to-air heat pumps

A.2 Small Office Building Analysis Results

A.2.1 Baseline Results

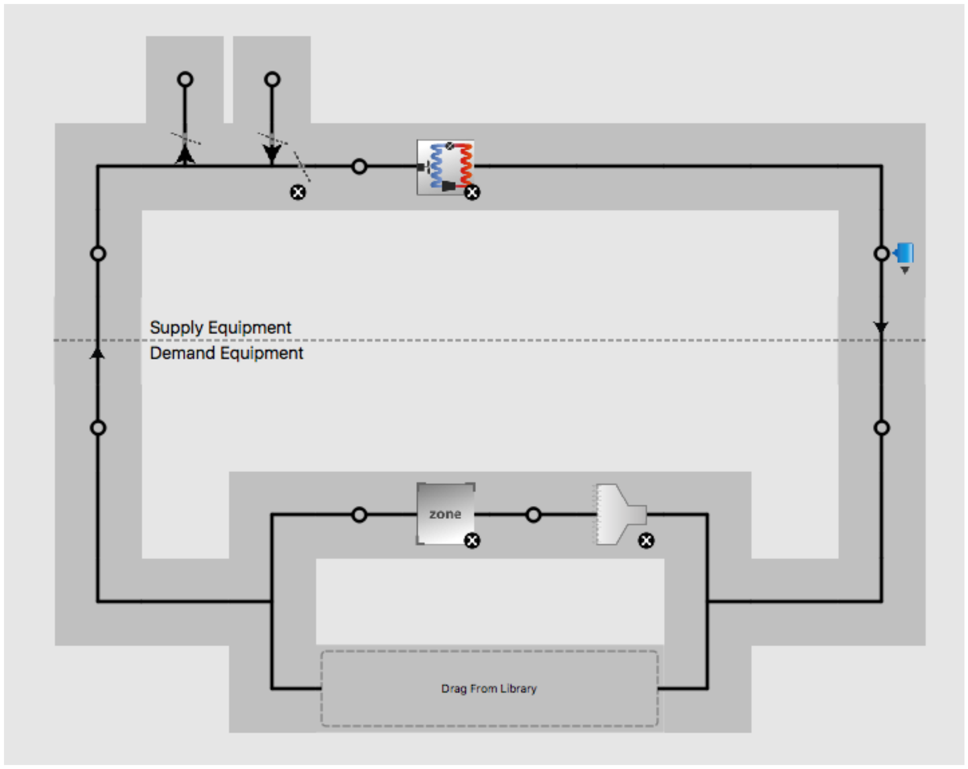


Figure A.3: Small office baseline HVAC system

A.2.2 Parametric Analyses Results

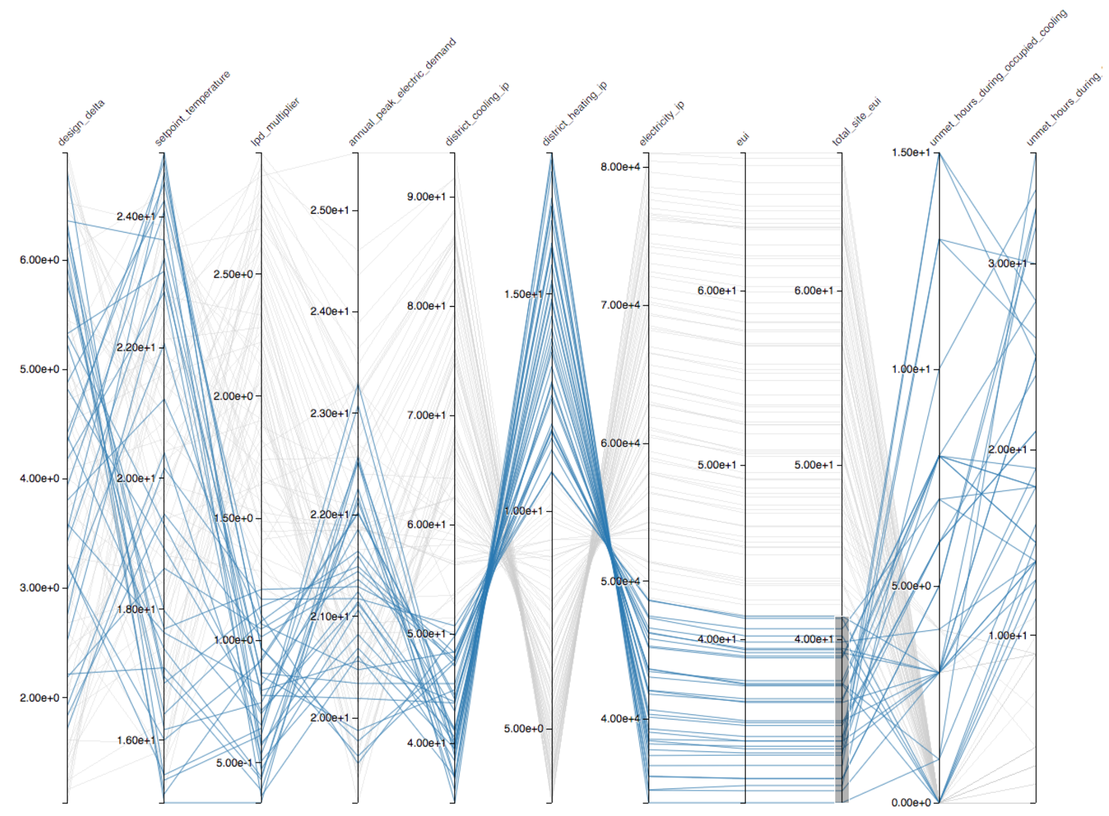


Figure A.4: Small office building parallel coordinate plot

A.2.3 Reduced Order Model Results

A.2.3.1 Linear Models

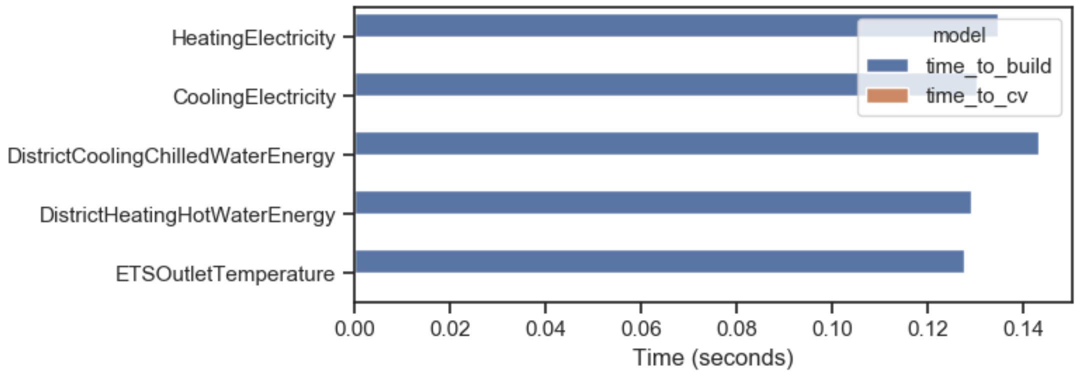


Figure A.5: Small Office: Linear model build time based on response variable, no down sampling

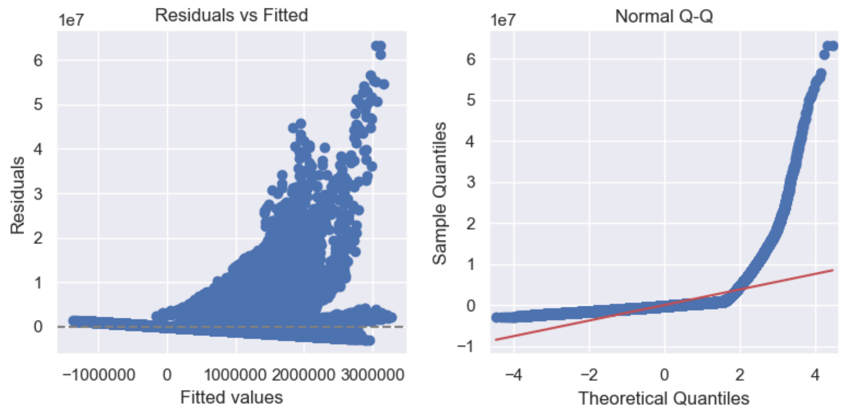


Figure A.6: Small Office: Diagnostic plot for heating electricity

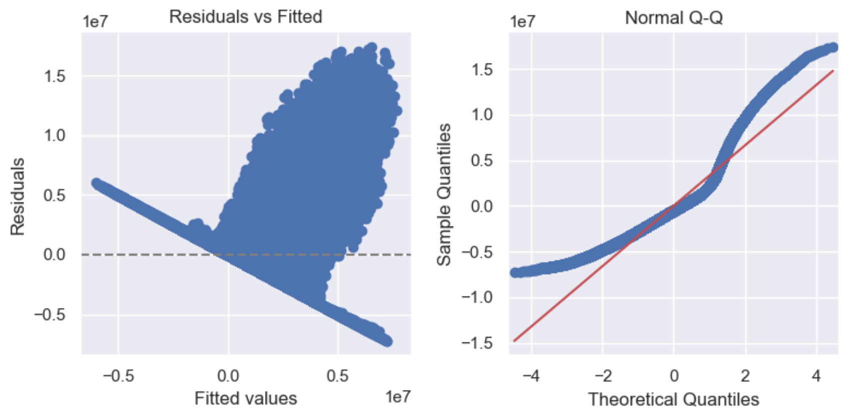


Figure A.7: Small Office: Diagnostic plot for cooling electricity

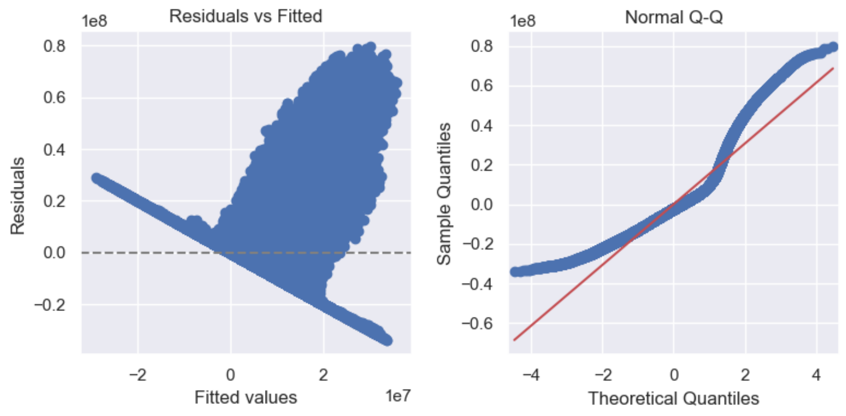


Figure A.8: Small Office: Diagnostic plot for district cooling energy

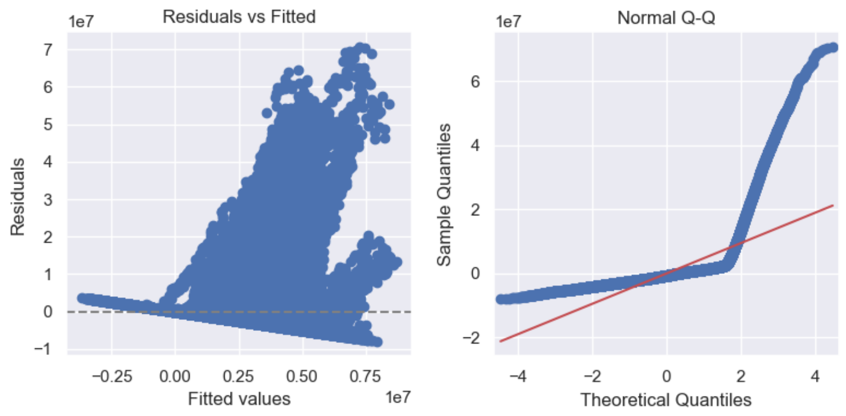


Figure A.9: Small Office: Diagnostic plot for district heating energy

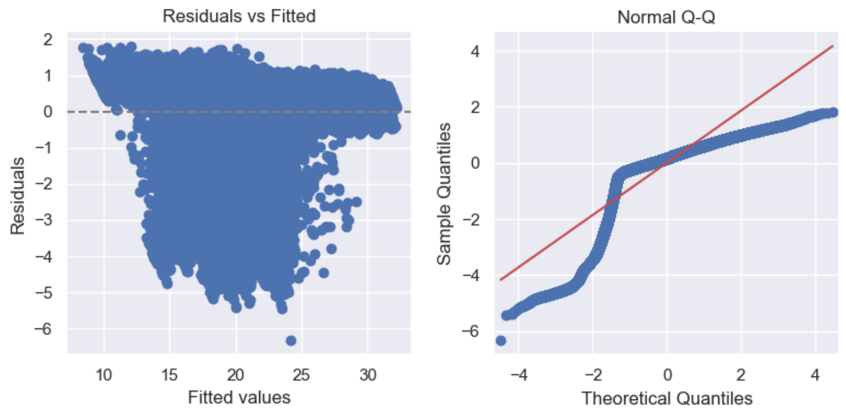
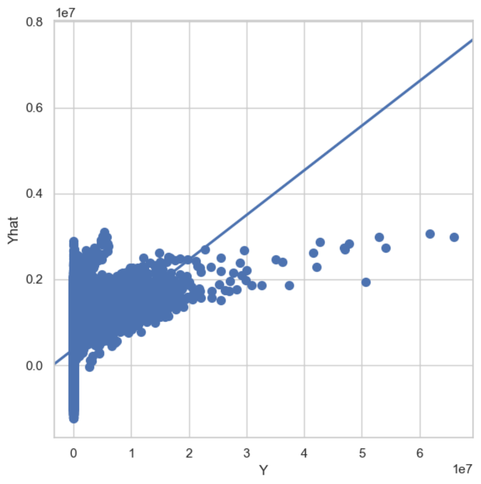
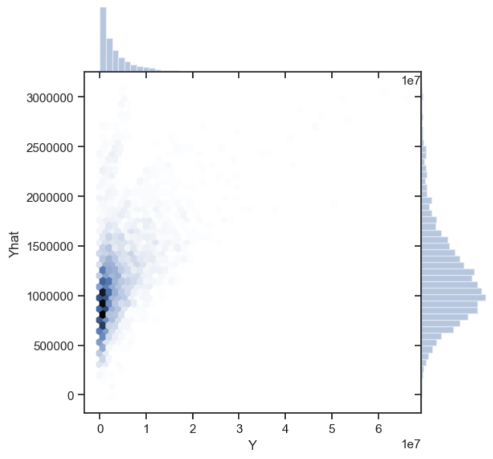


Figure A.10: Small Office: Diagnostic plot for ETS outlet temperature

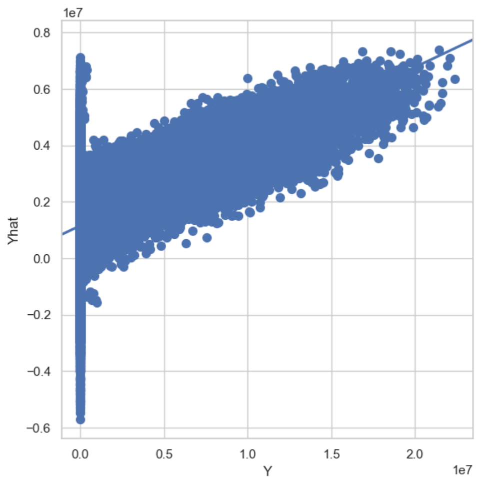


(a) Raw Data

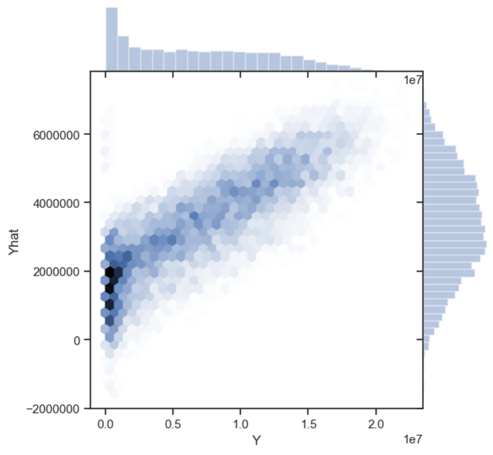


(b) Hex Plot

Figure A.11: Small Office: Heating electricity

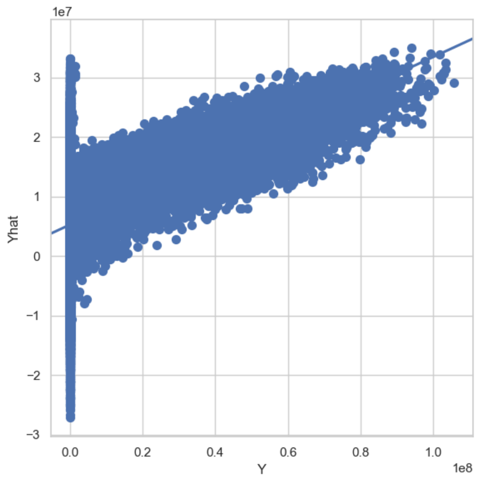


(a) Raw Data

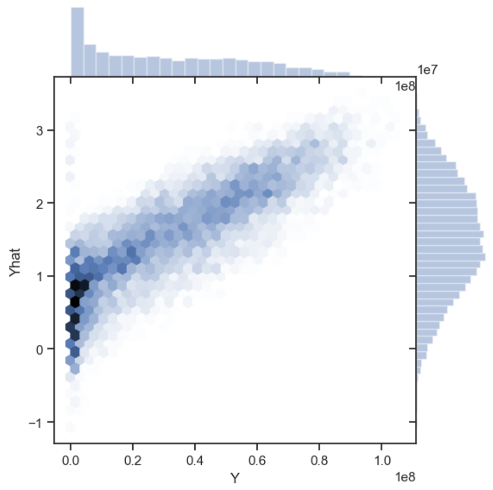


(b) Hex Plot

Figure A.12: Small Office: Cooling electricity

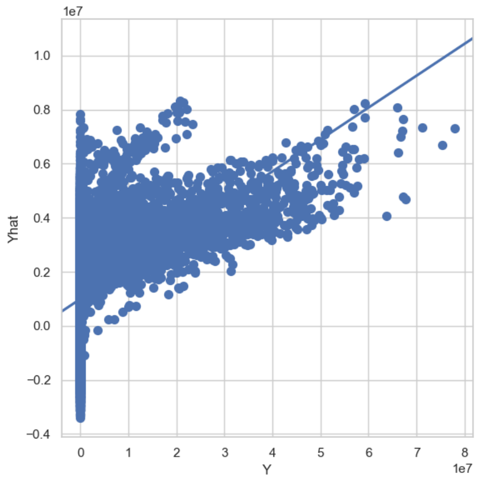


(a) Raw Data

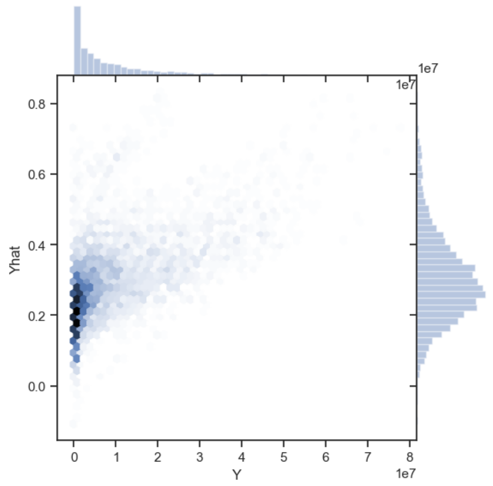


(b) Hex Plot

Figure A.13: Small Office: District cooling energy

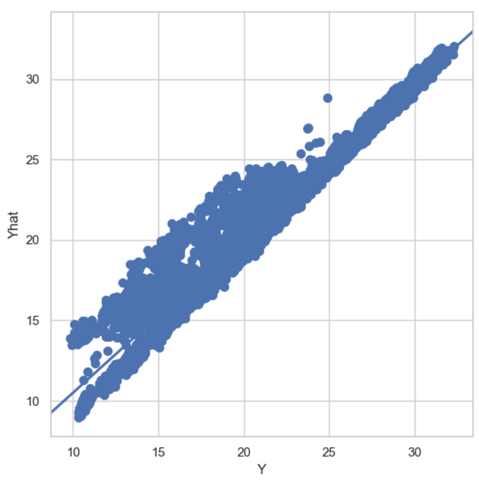


(a) Raw Data

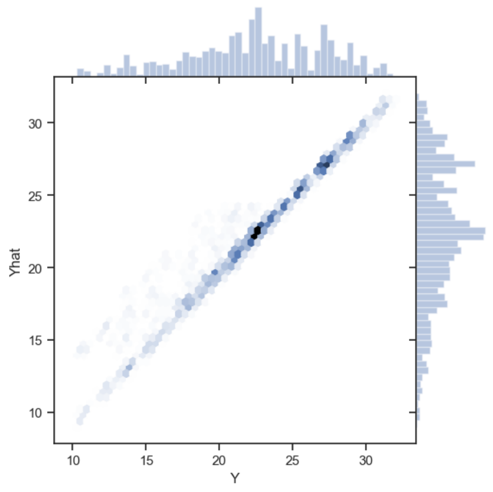


(b) Hex Plot

Figure A.14: Small Office: District heating energy



(a) Raw Data



(b) Hex Plot

Figure A.15: Small Office: ETS outlet temperature

A.2.3.2 Random Forest

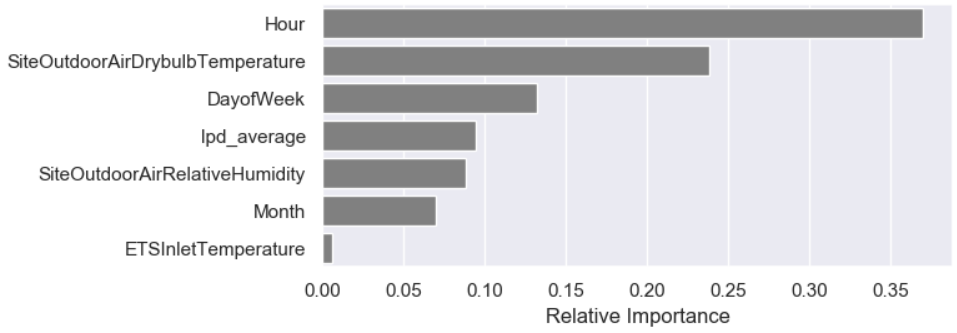


Figure A.16: Small Office: Heating electricity relative importance

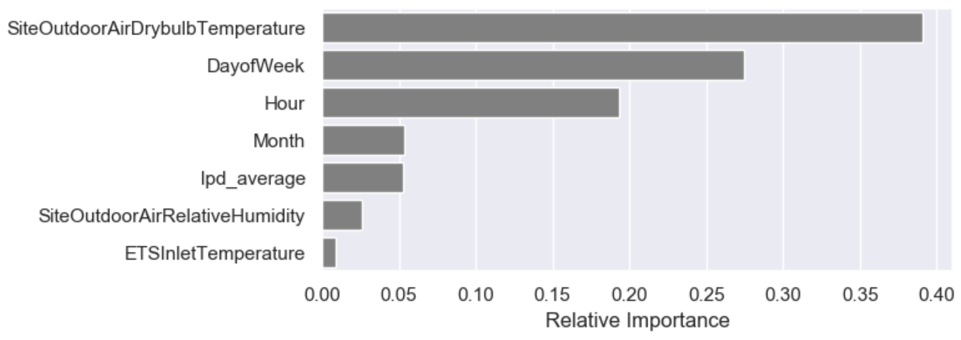


Figure A.17: Small Office: Cooling electricity relative importance

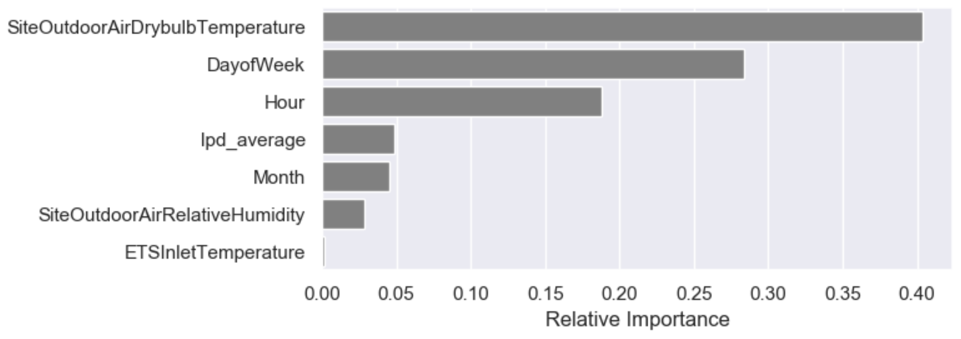


Figure A.18: Small Office: District cooling energy relative importance

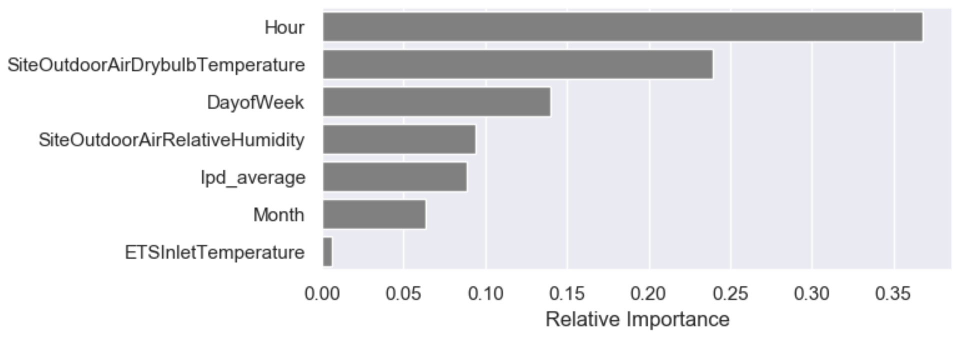


Figure A.19: Small Office: District heating energy relative importance

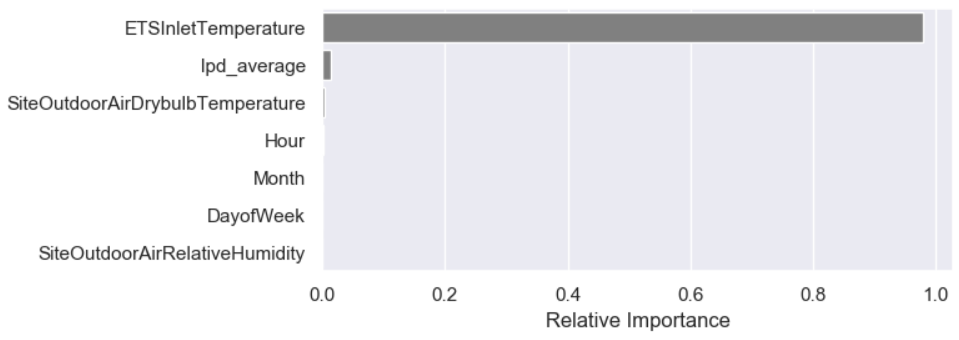
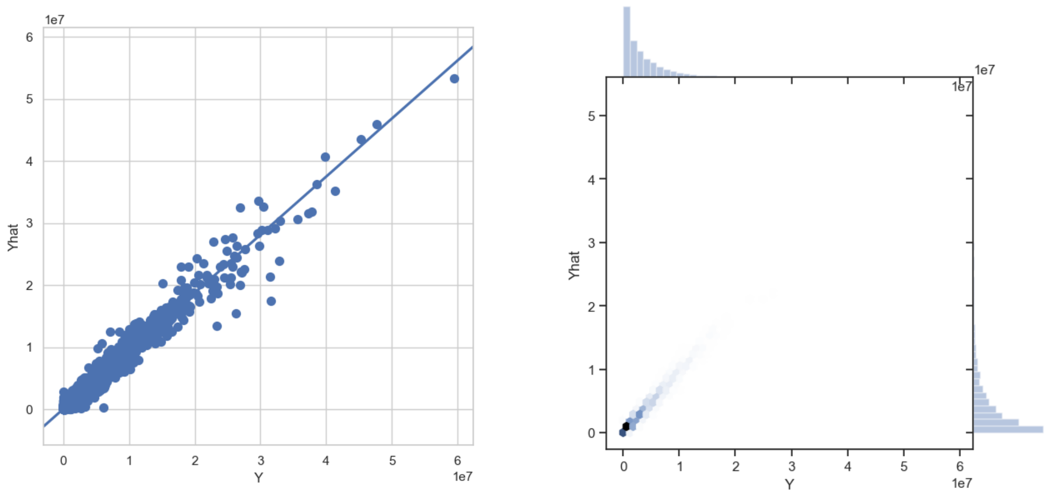


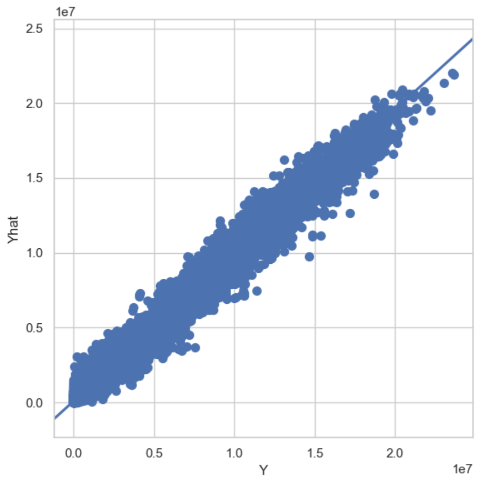
Figure A.20: Small Office: ETS outlet temperature importance



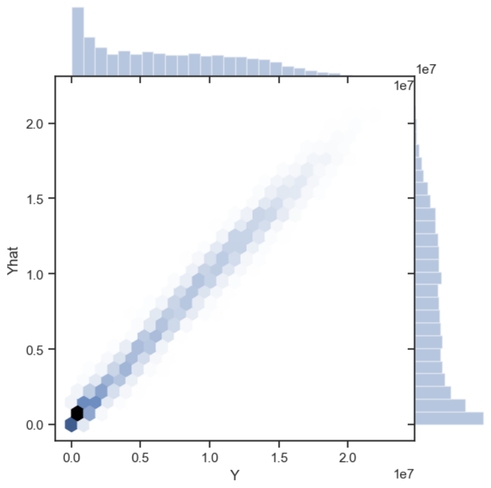
(a) Raw Data

(b) Hex Plot

Figure A.21: Small Office: Heating electricity

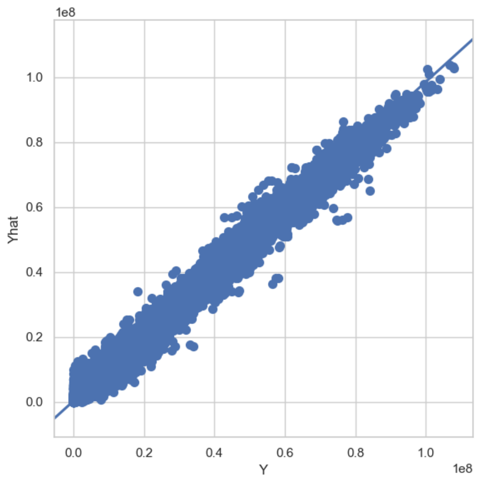


(a) Raw Data

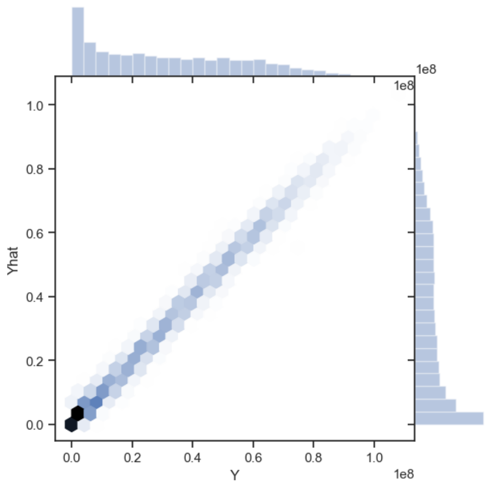


(b) Hex Plot

Figure A.22: Small Office: Cooling electricity

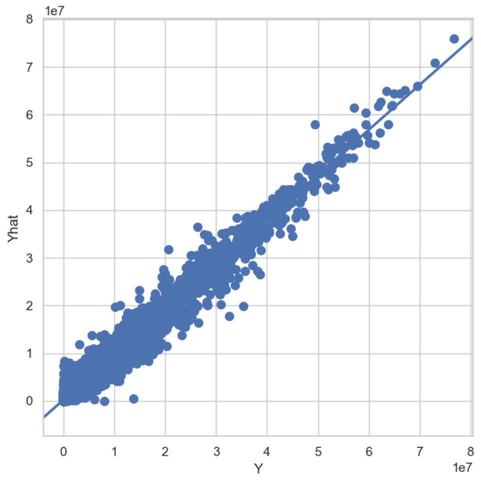


(a) Raw Data

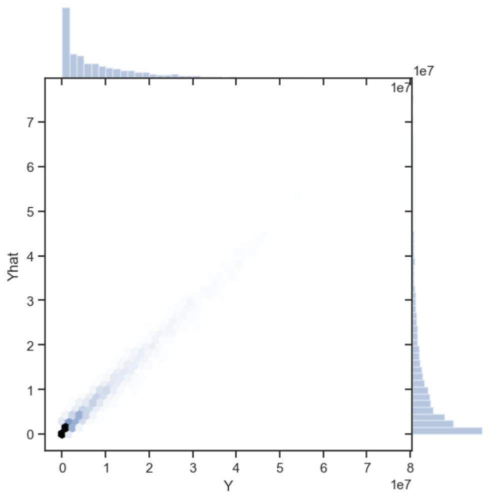


(b) Hex Plot

Figure A.23: Small Office: District cooling energy

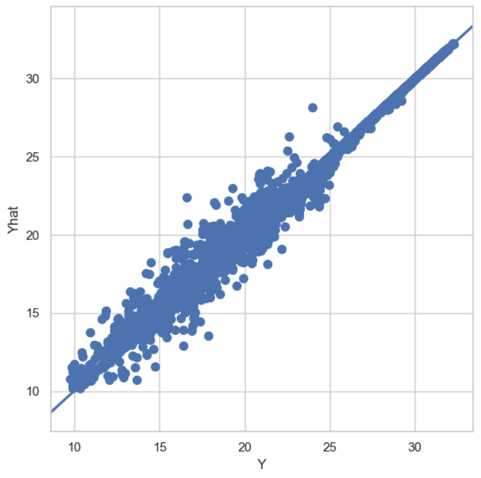


(a) Raw Data

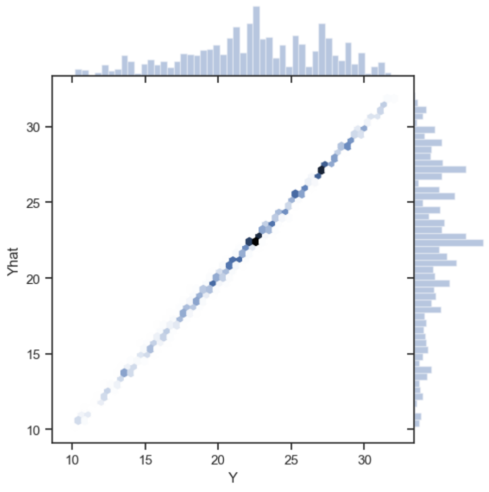


(b) Hex Plot

Figure A.24: Small Office: District heating energy



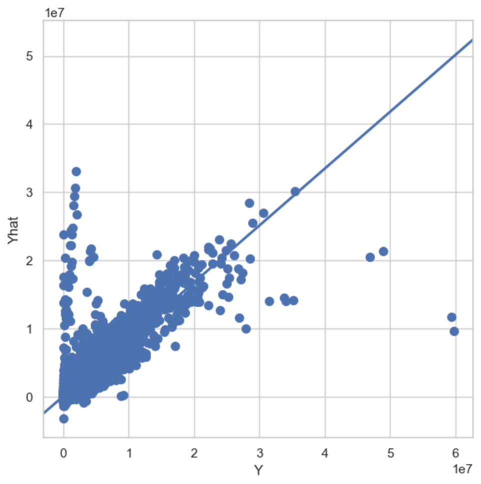
(a) Raw Data



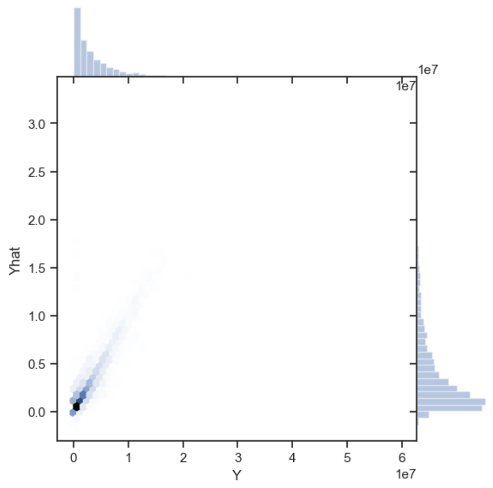
(b) Hex Plot

Figure A.25: Small Office: ETS outlet temperature

A.2.3.3 Support Vector Regression

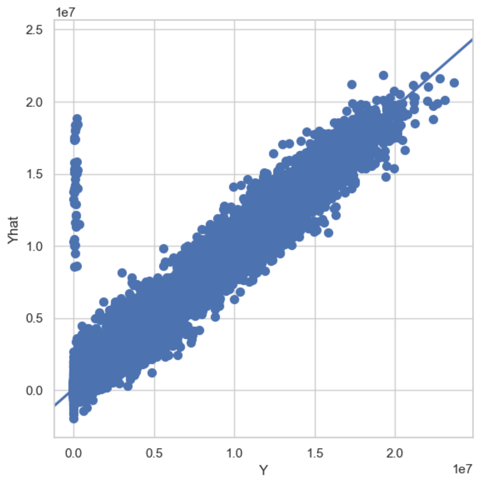


(a) Raw Data

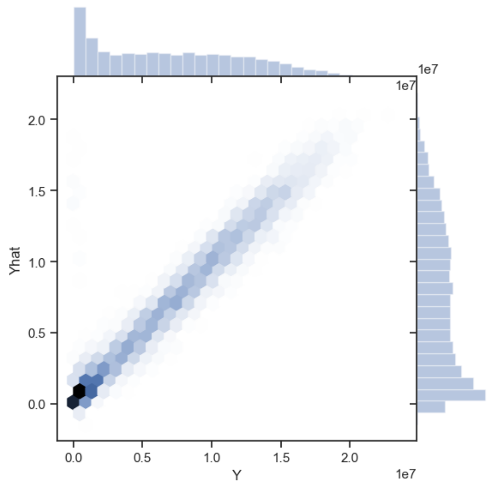


(b) Hex Plot

Figure A.26: Small Office: Heating electricity

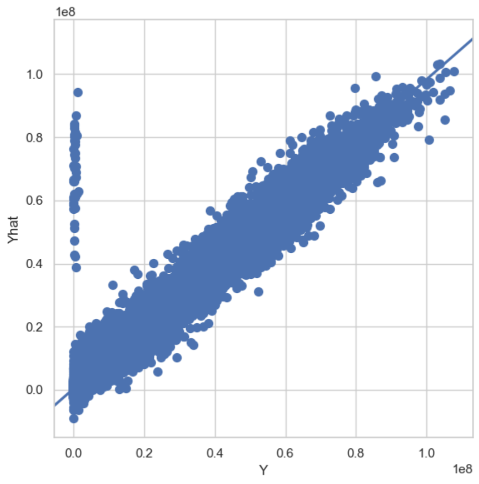


(a) Raw Data

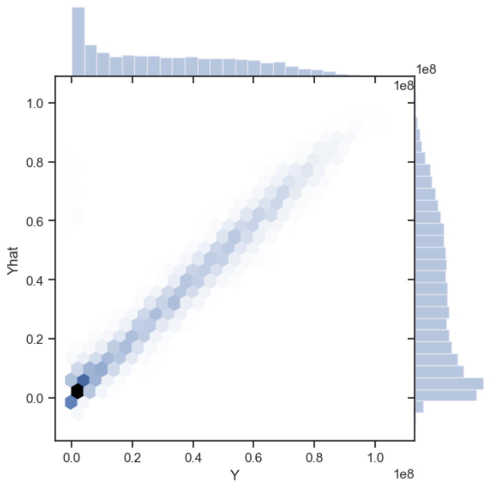


(b) Hex Plot

Figure A.27: Small Office: Cooling electricity

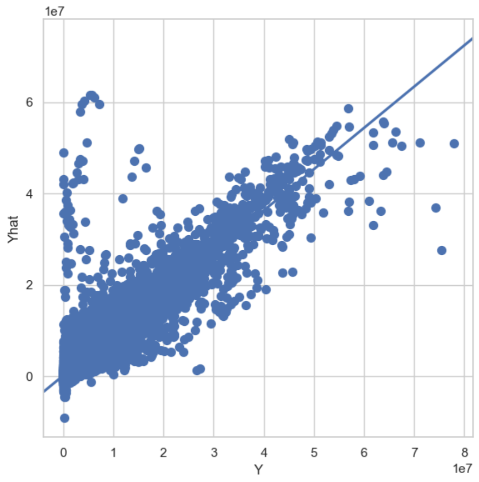


(a) Raw Data

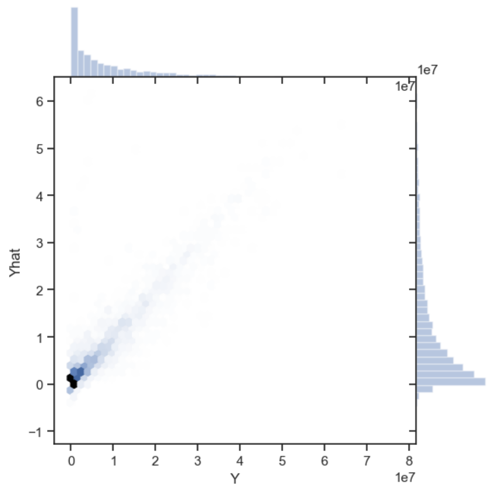


(b) Hex Plot

Figure A.28: Small Office: District cooling energy

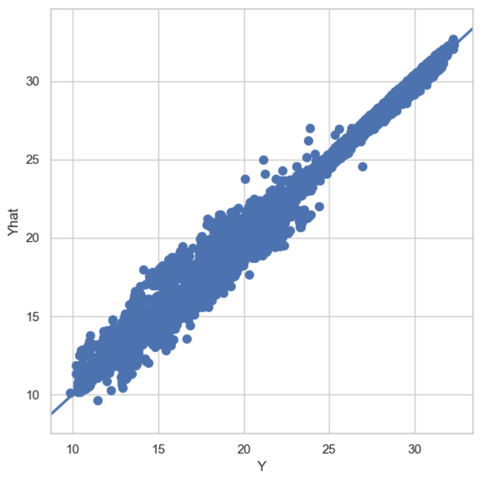


(a) Raw Data

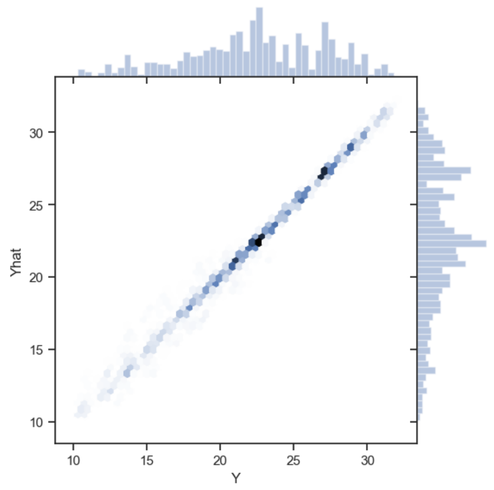


(b) Hex Plot

Figure A.29: Small Office: District heating energy



(a) Raw Data



(b) Hex Plot

Figure A.30: Small Office: ETS outlet temperature

A.2.4 Validation Results

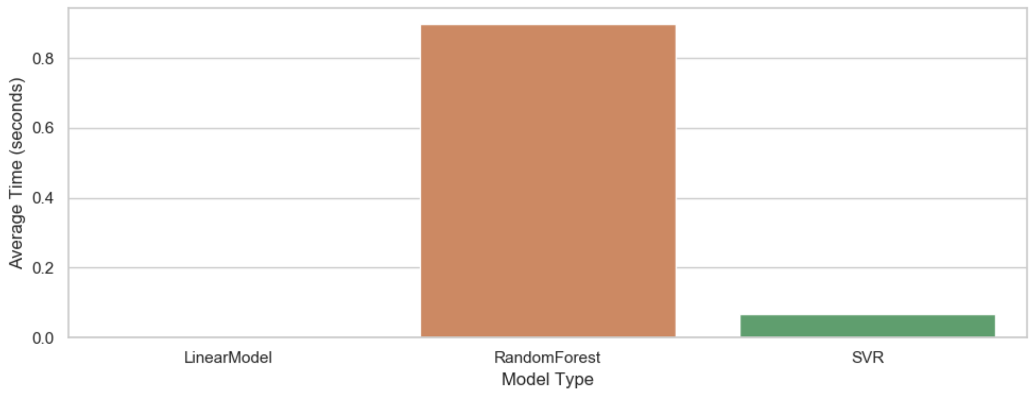


Figure A.31: Small Office: Load time as a function of the model type

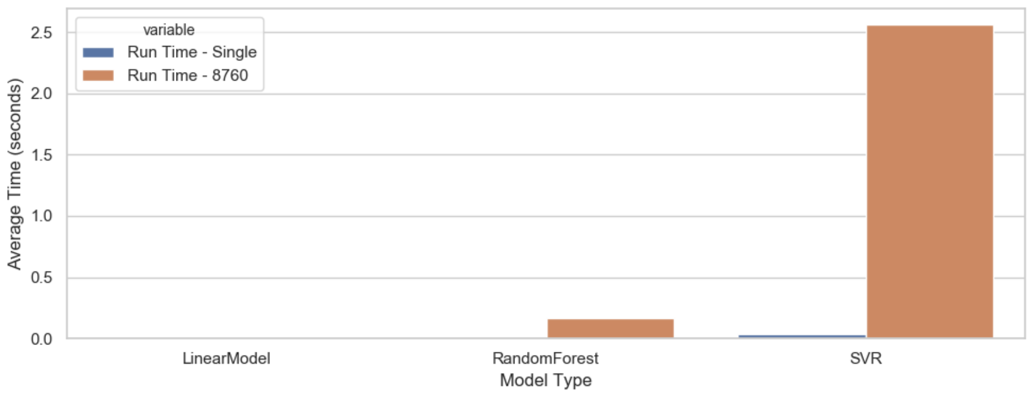


Figure A.32: Small Office: Run times as a function of the model type

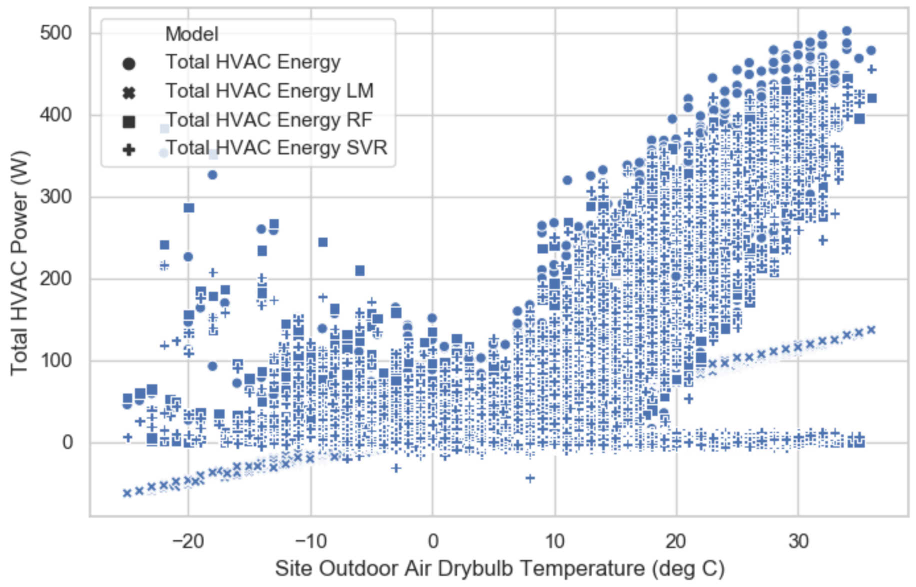


Figure A.33: Small Office: Energy Consumption vs Outdoor Dry-Bulb Temperature

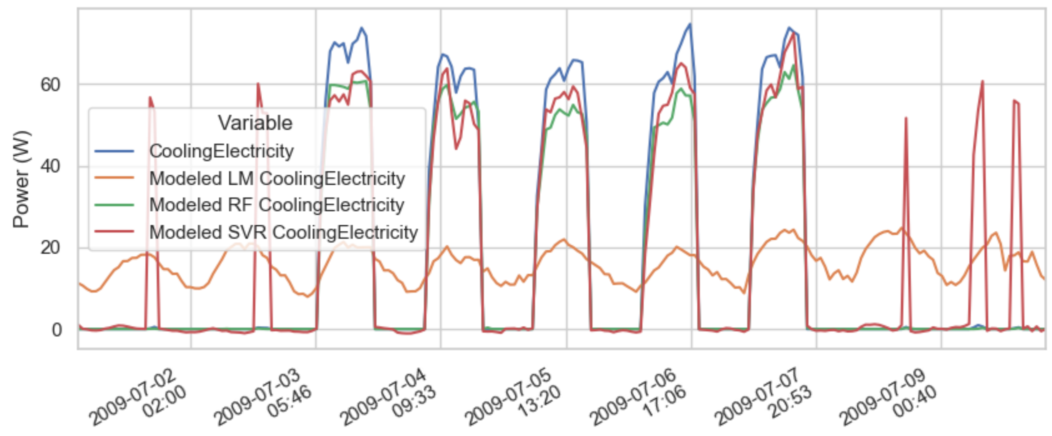


Figure A.34: Small Office: Summer validation period cooling electricity performance

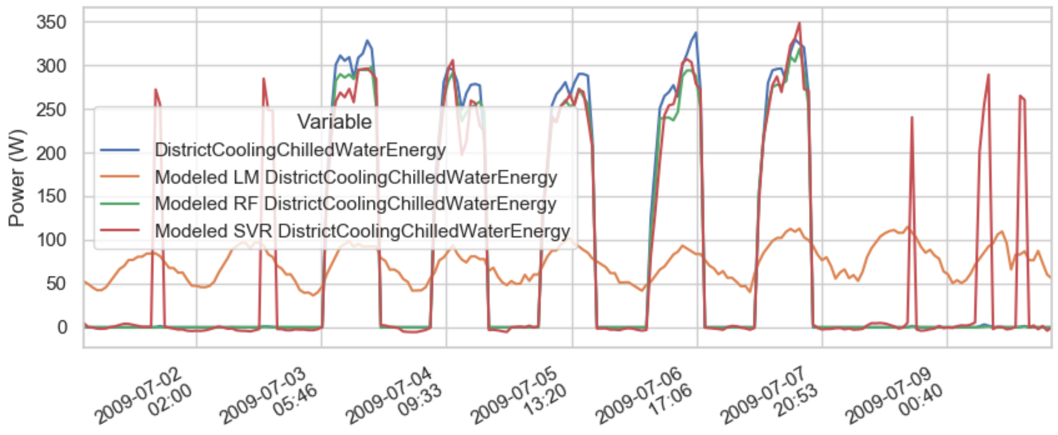


Figure A.35: Small Office: Summer validation period district cooling energy

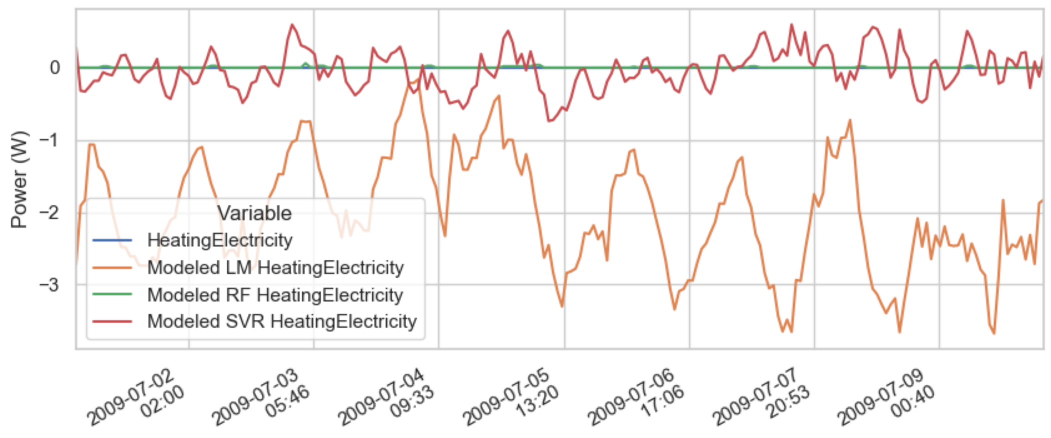


Figure A.36: Small Office: Summer validation period heating electricity performance

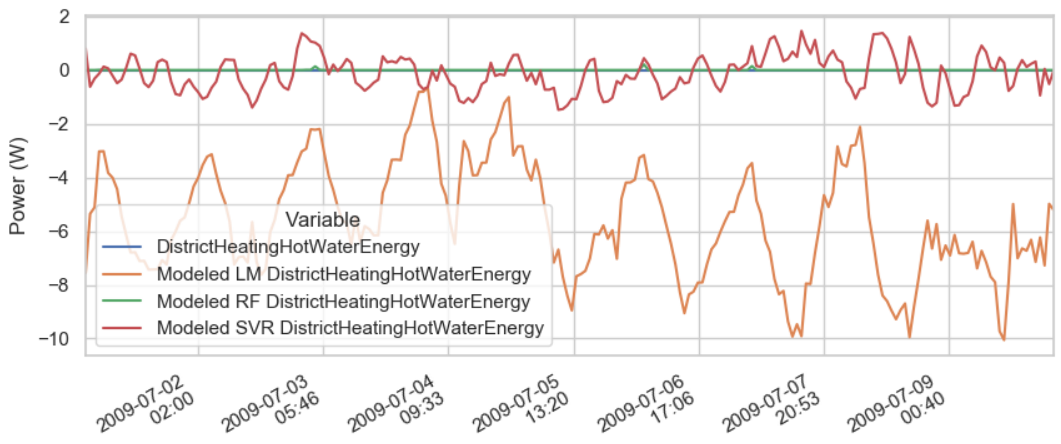


Figure A.37: Small Office: Summer validation period district heating energy

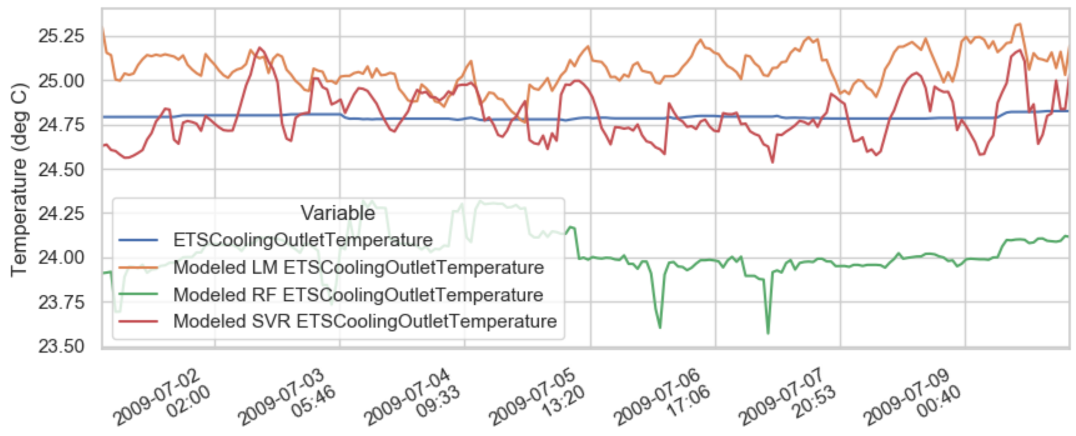


Figure A.38: Small Office: Summer validation period ETS outlet temperature performance

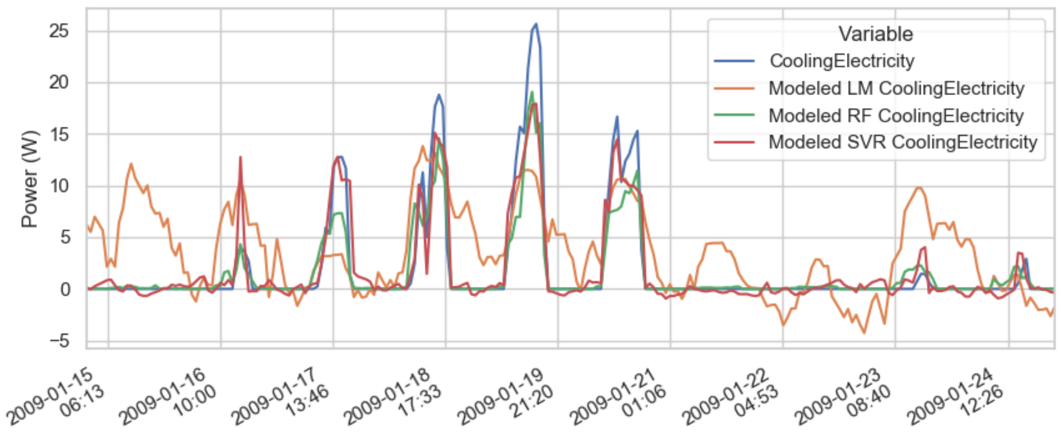


Figure A.39: Small Office: Winter validation period cooling electricity performance

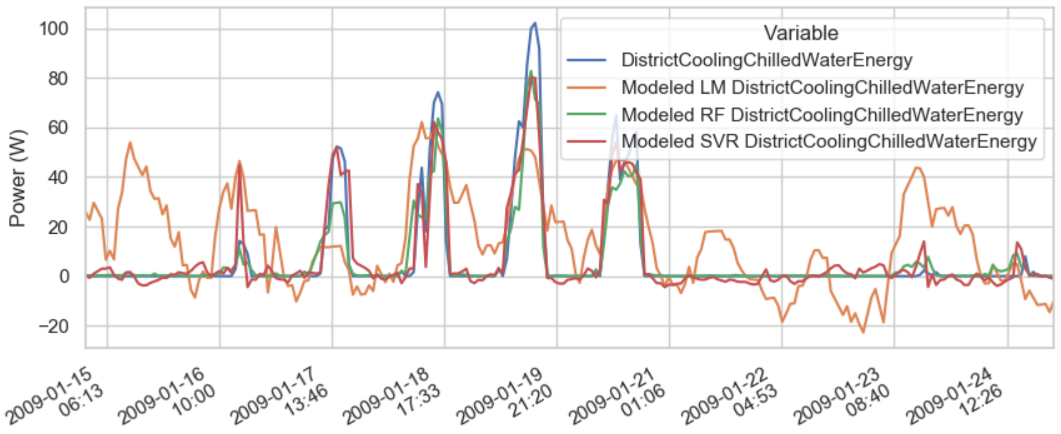


Figure A.40: Small Office: Winter validation period district cooling energy

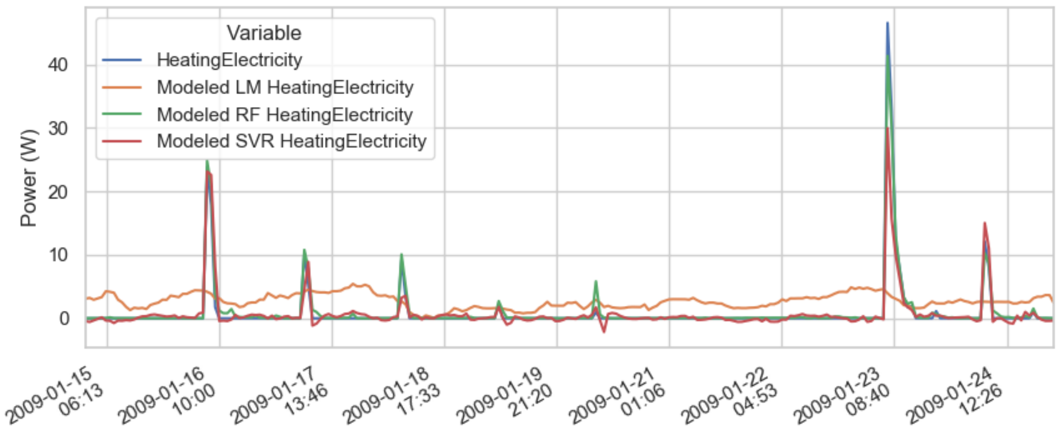


Figure A.41: Small Office: Winter validation period heating electricity performance

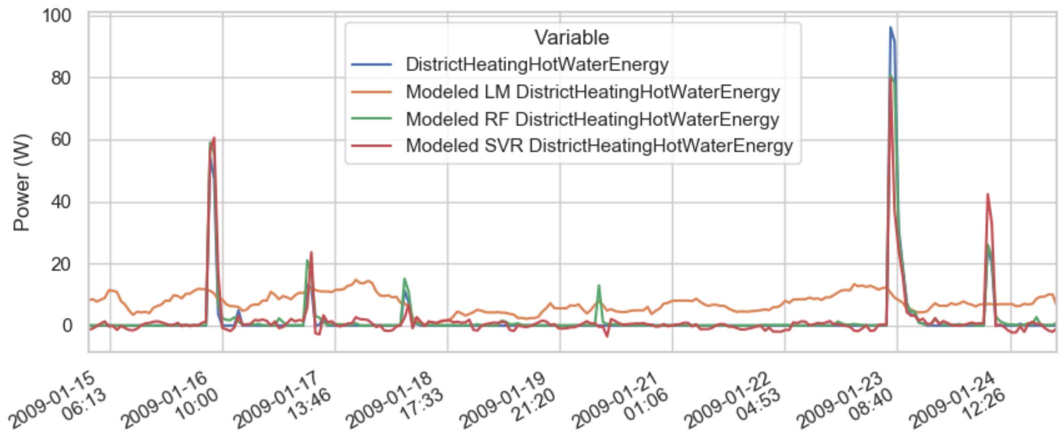


Figure A.42: Small Office: Winter validation period district heating energy

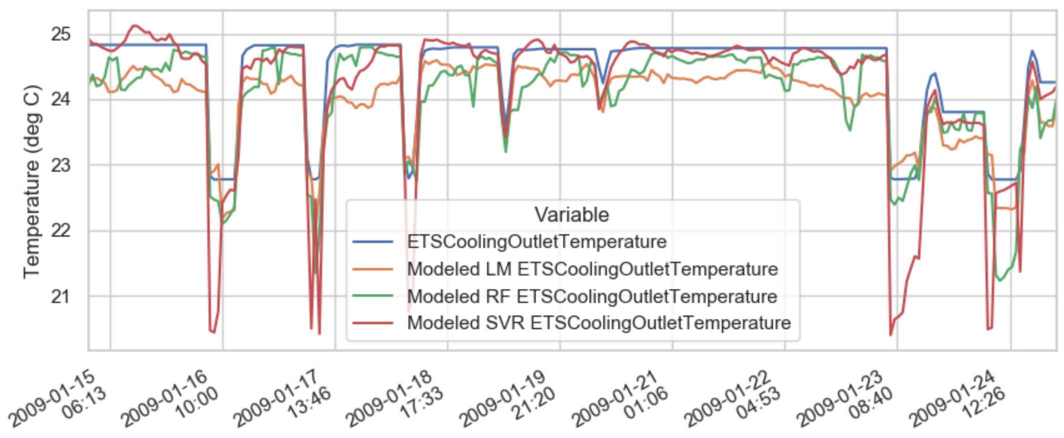


Figure A.43: Small Office: Winter validation period ETS outlet temperature performance

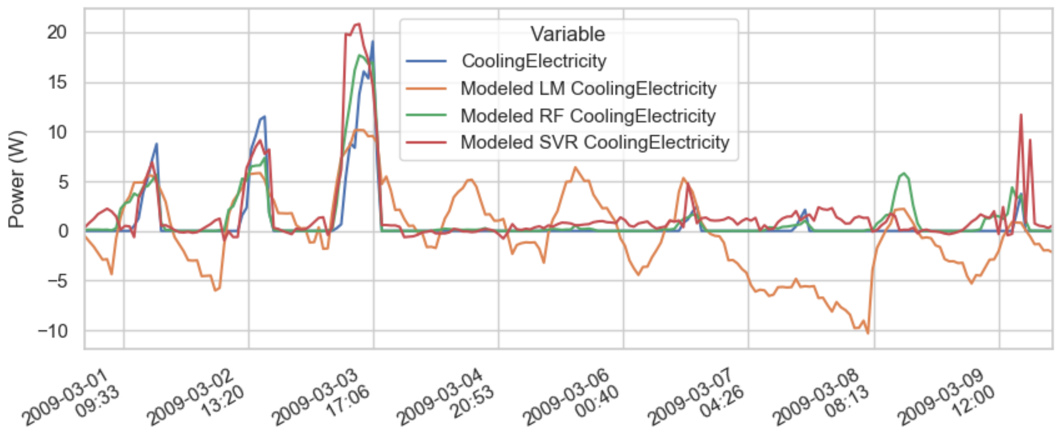


Figure A.44: Small Office: Swing validation period cooling electricity performance

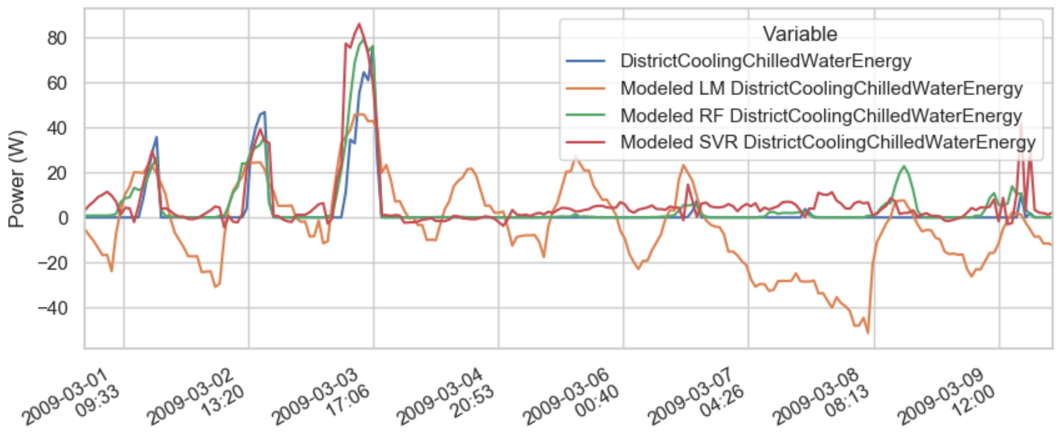


Figure A.45: Small Office: Swing validation period district cooling energy

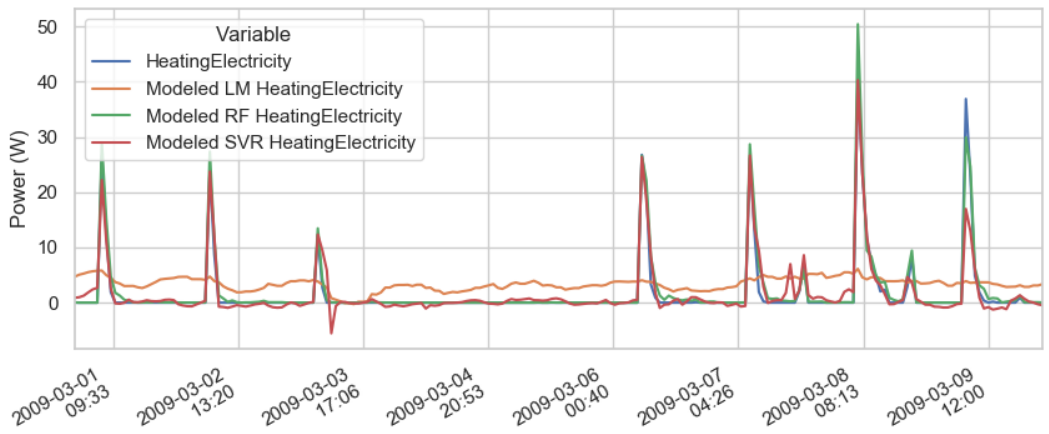


Figure A.46: Small Office: Swing validation period heating electricity performance

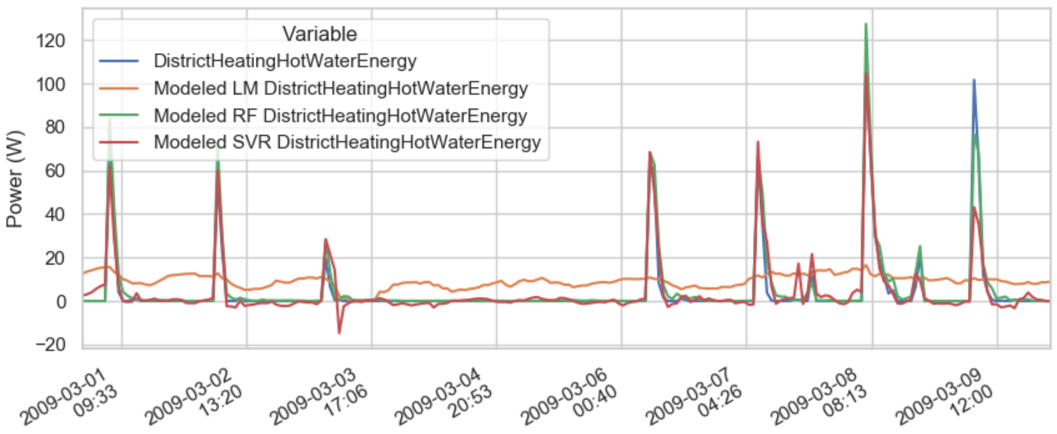


Figure A.47: Small Office: Swing validation period district heating energy

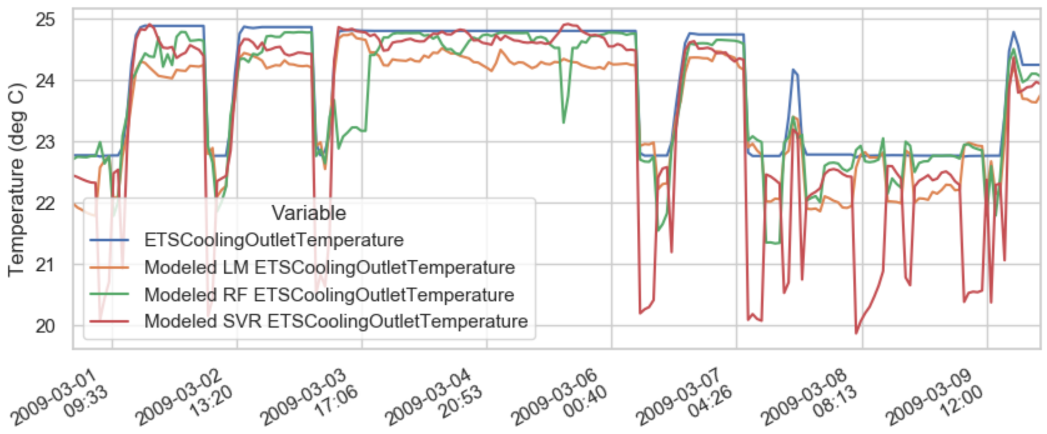


Figure A.48: Small Office: Swing validation period ETS outlet temperature performance

Appendix B

Retail Building Analysis Results

B.1 Baseline Results

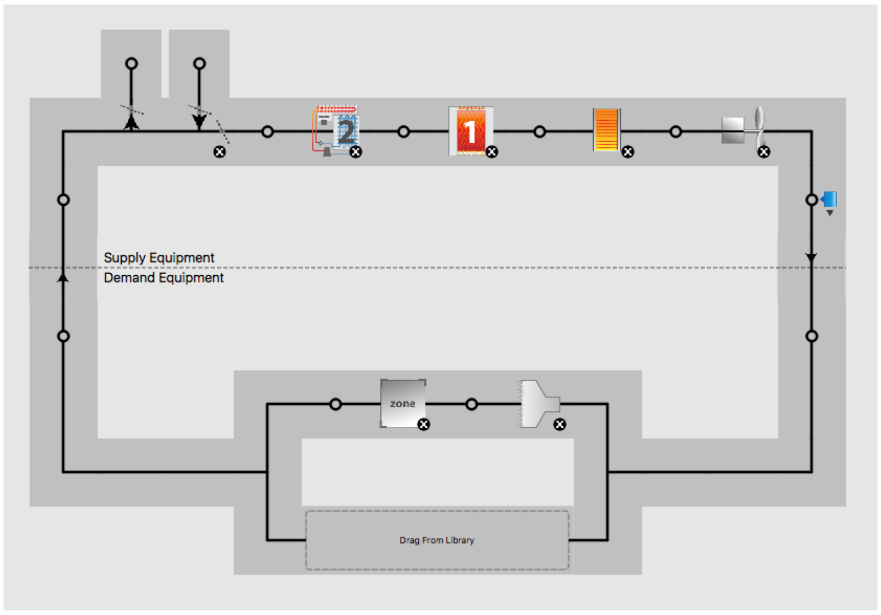


Figure B.1: Retail baseline HVAC system

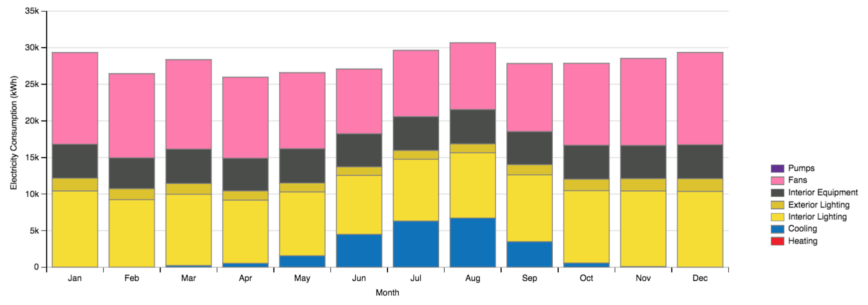


Figure B.2: Retail baseline monthly electricity energy end use

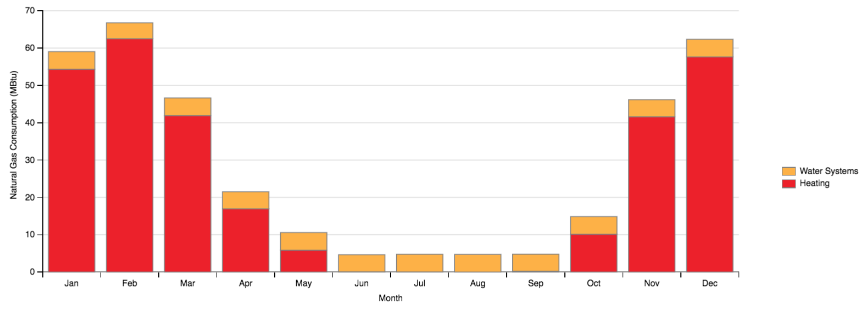


Figure B.3: Retail baseline monthly gas energy end use

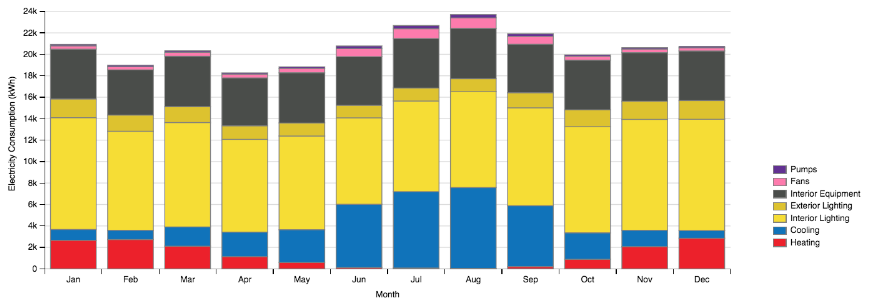


Figure B.4: Retail baseline with ETS monthly electricity energy end use

B.2 Parametric Analysis Results

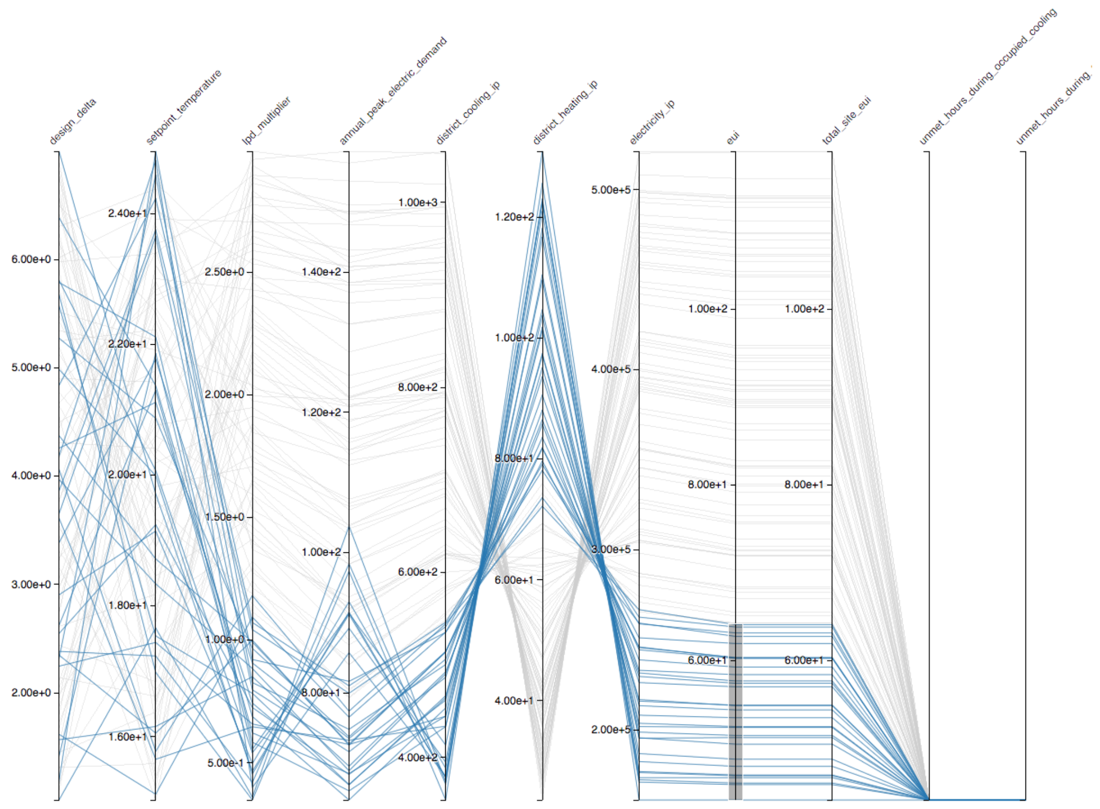


Figure B.5: Retail building parallel coordinate plot

B.3 Reduced Order Model Results

B.3.1 Linear Models

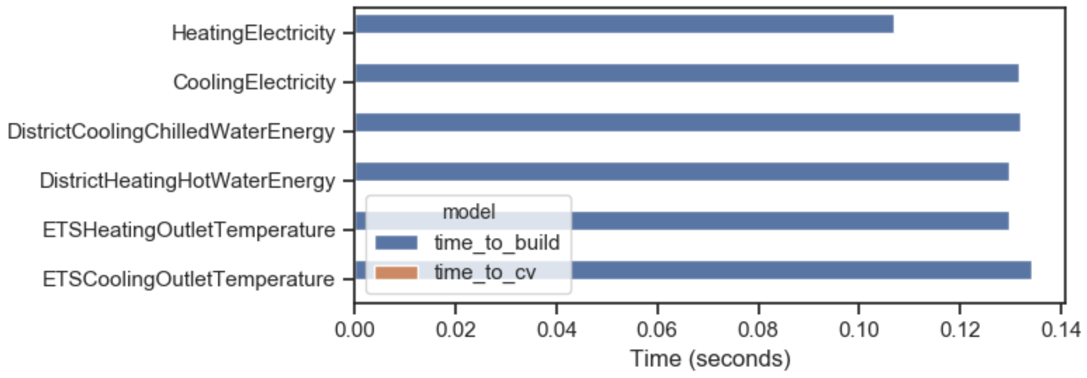


Figure B.6: Linear model build time based on response variable, no down sampling

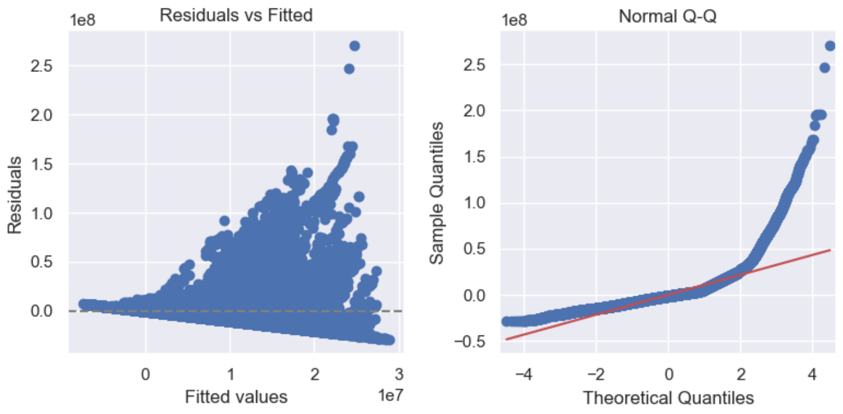


Figure B.7: Retail: Diagnostic plot for heating electricity

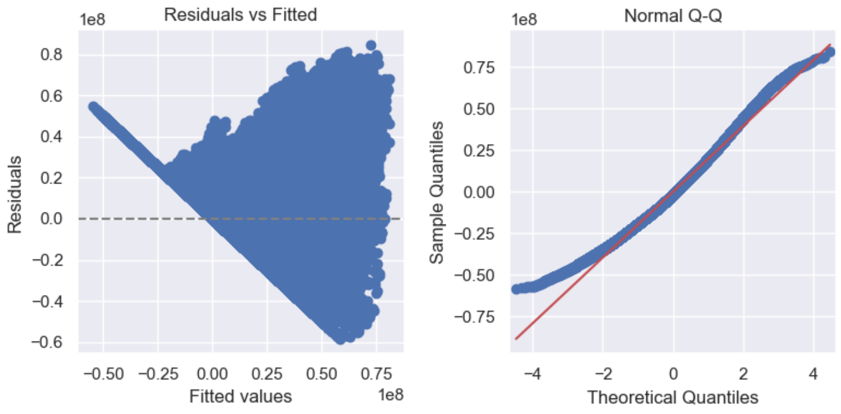


Figure B.8: Retail: Diagnostic plot for cooling electricity

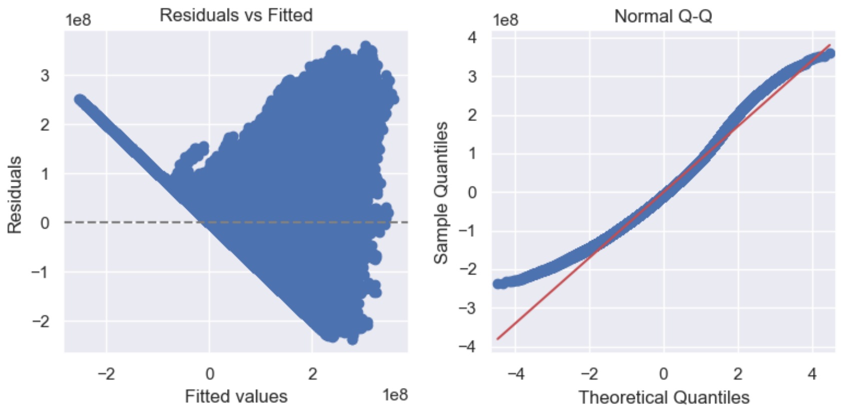


Figure B.9: Retail: Diagnostic plot for district cooling energy

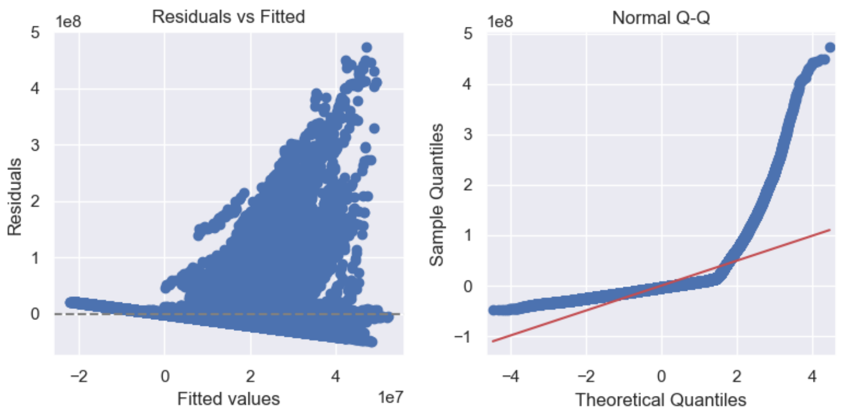


Figure B.10: Retail: Diagnostic plot for district heating energy

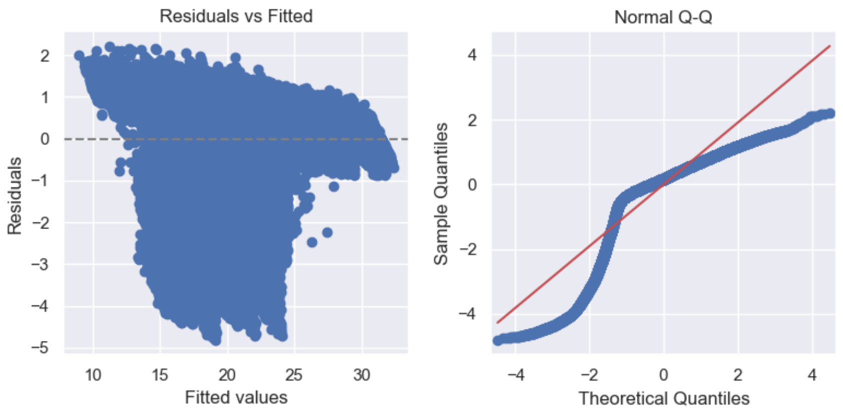
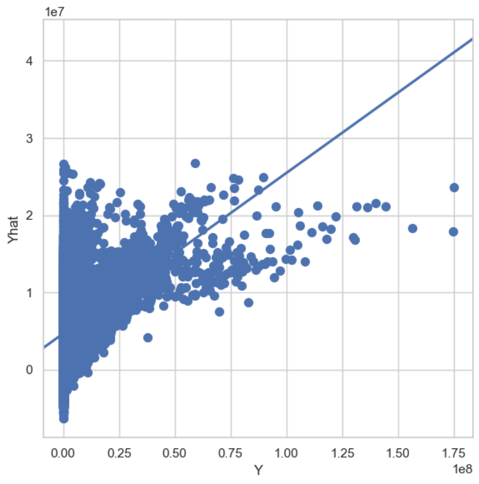
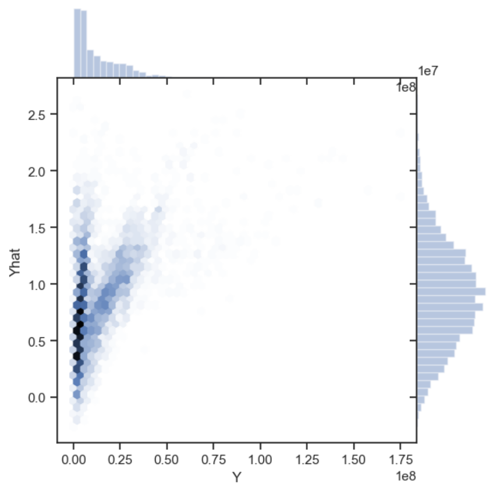


Figure B.11: Retail: Diagnostic plot for ETS outlet temperature

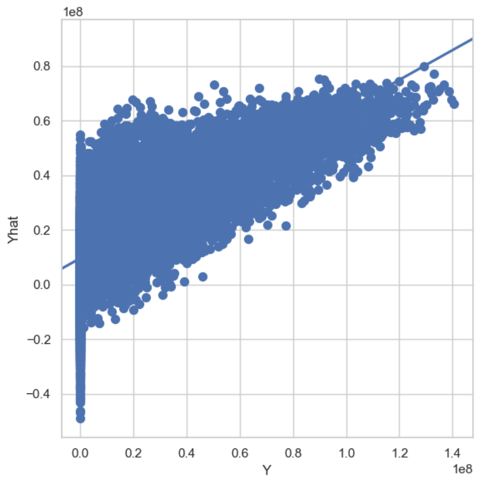


(a) Raw Data

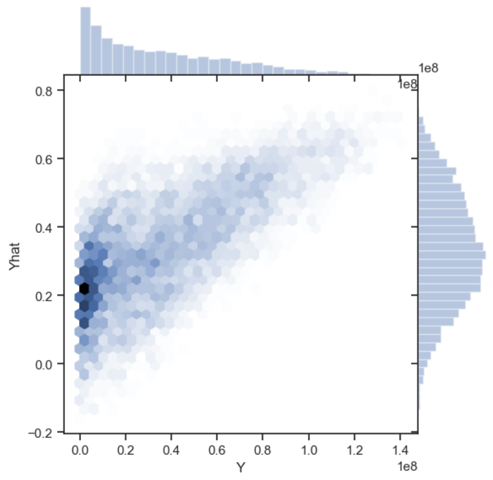


(b) Hex Plot

Figure B.12: Retail: Heating electricity

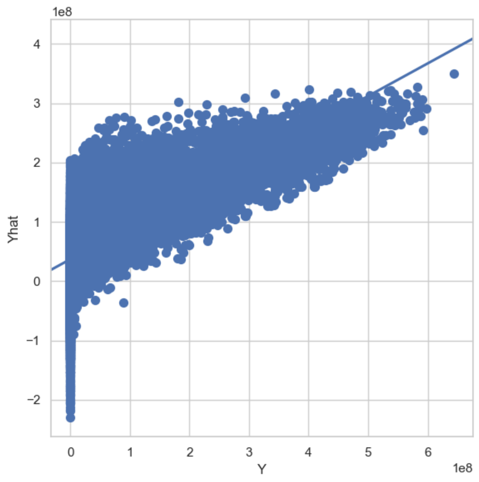


(a) Raw Data

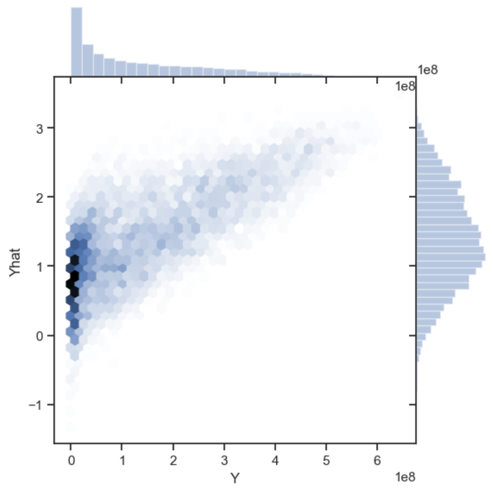


(b) Hex Plot

Figure B.13: Retail: Cooling electricity

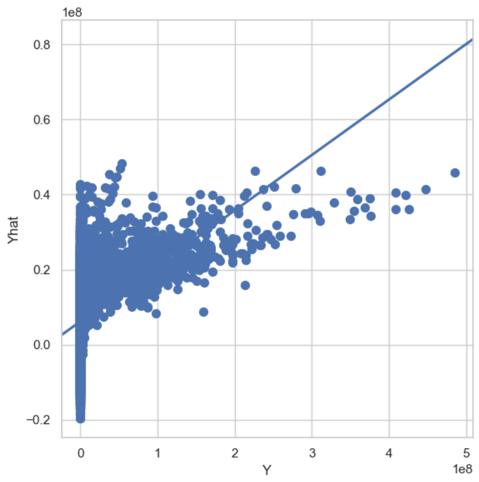


(a) Raw Data

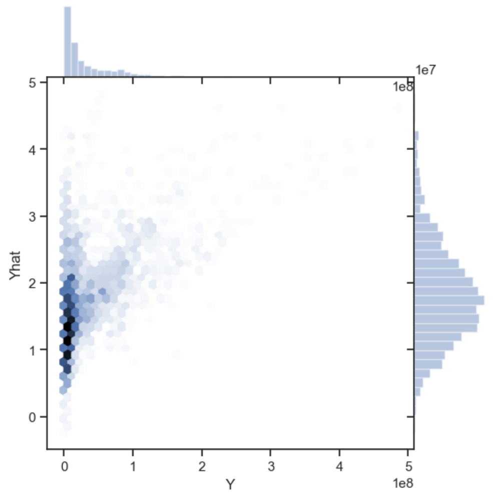


(b) Hex Plot

Figure B.14: Retail: District cooling energy

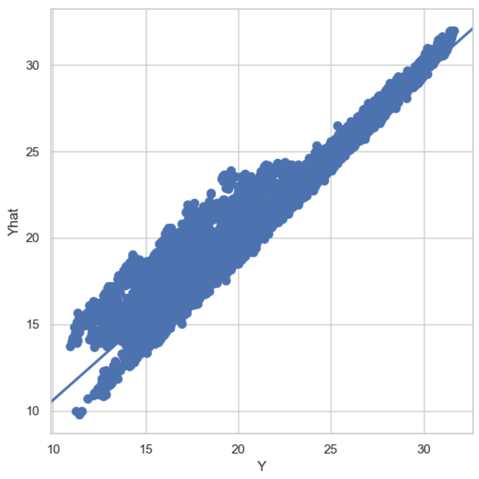


(a) Raw Data

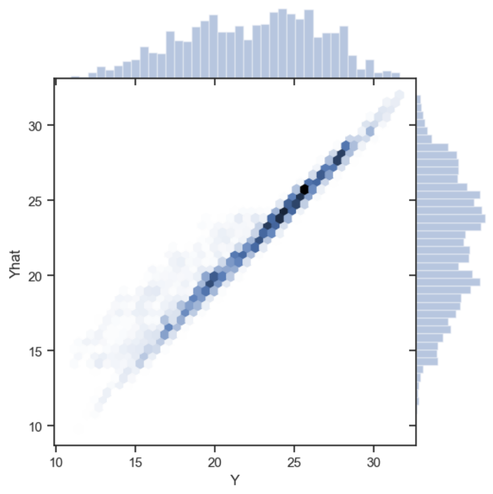


(b) Hex Plot

Figure B.15: Retail: District heating energy



(a) Raw Data



(b) Hex Plot

Figure B.16: Retail: ETS outlet temperature

B.3.2 Random Forest

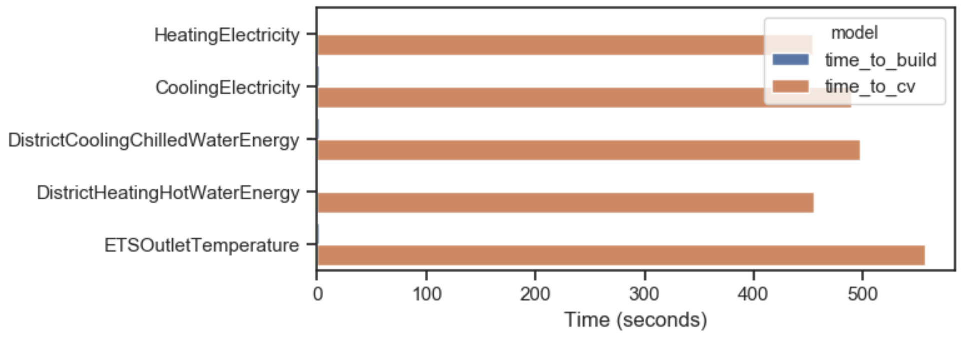
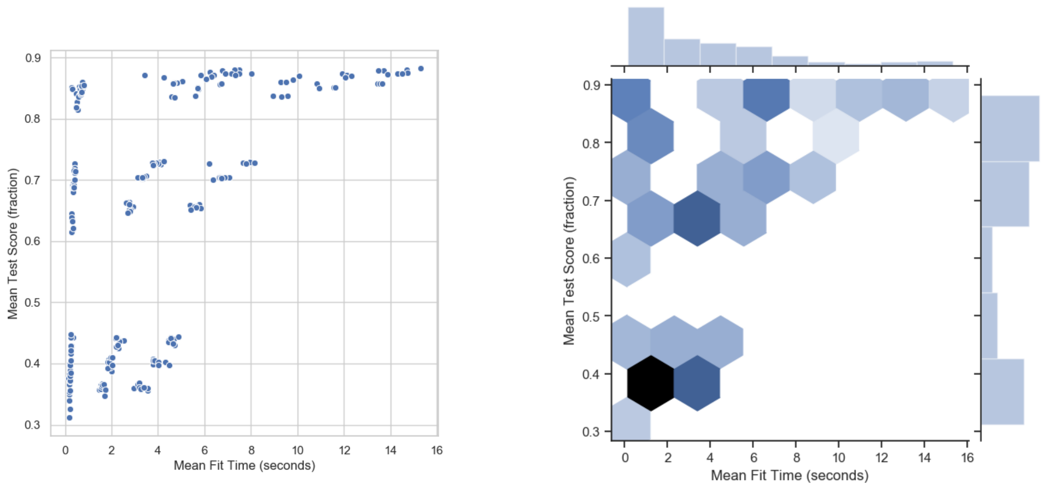


Figure B.17: Random forest build and cross-validation time



(a) Raw Data

(b) Hex Plot

Figure B.18: District heating energy build time compared to test score

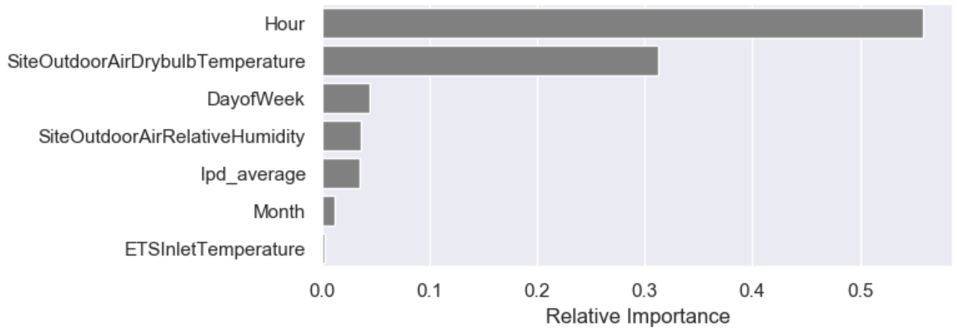


Figure B.19: Retail: Heating electricity relative importance

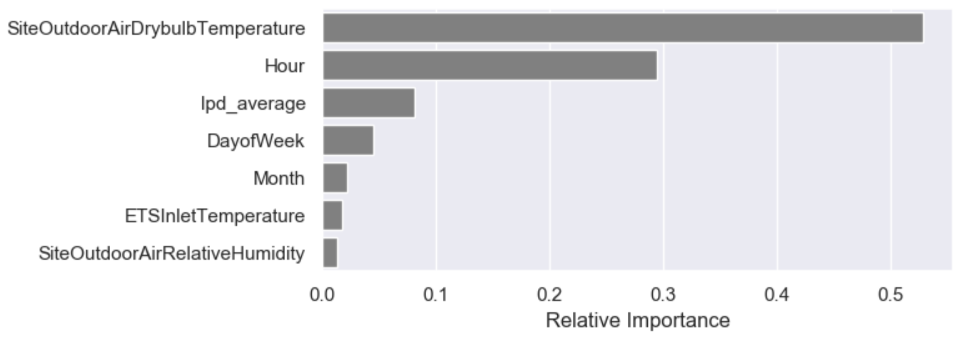


Figure B.20: Retail: Cooling electricity relative importance

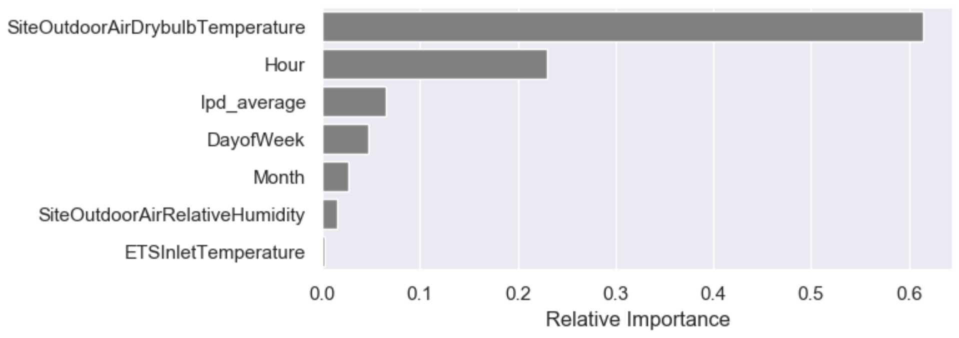


Figure B.21: Retail: District cooling energy relative importance

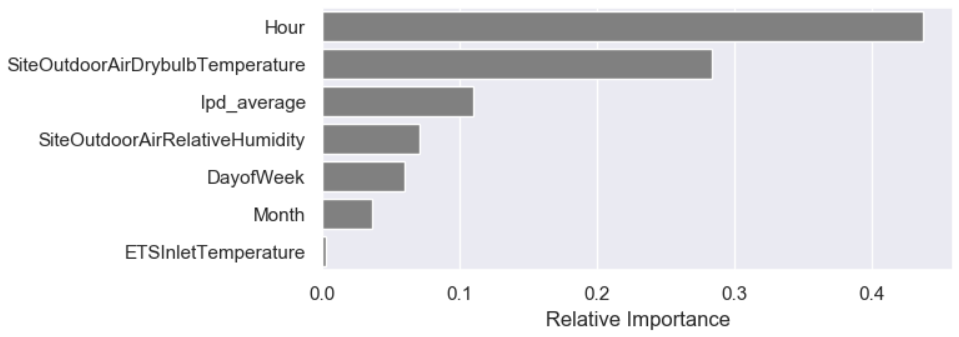


Figure B.22: Retail: District heating energy relative importance

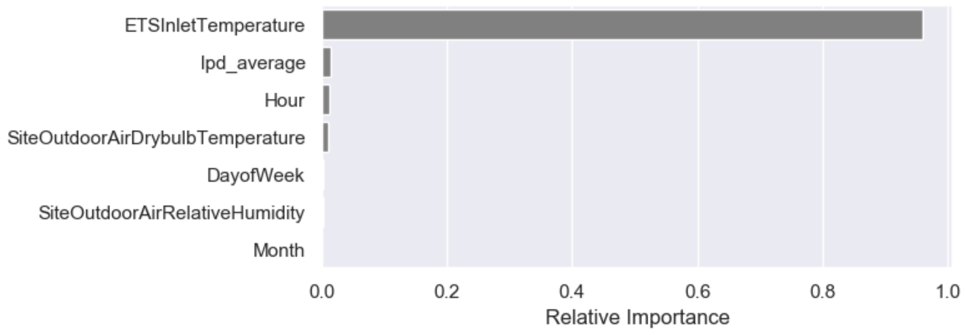
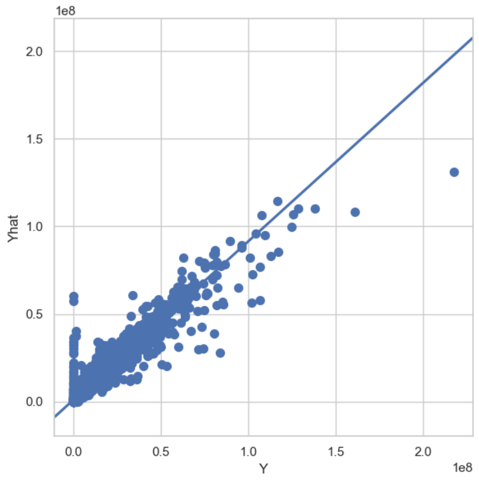
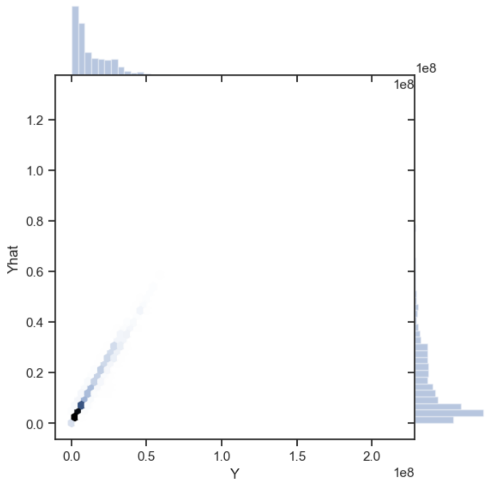


Figure B.23: Retail: ETS outlet temperature importance

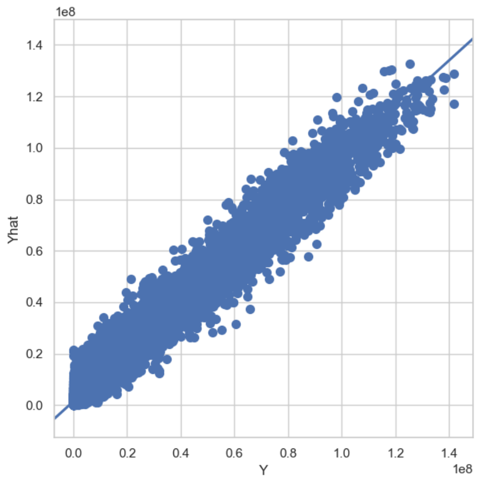


(a) Raw Data

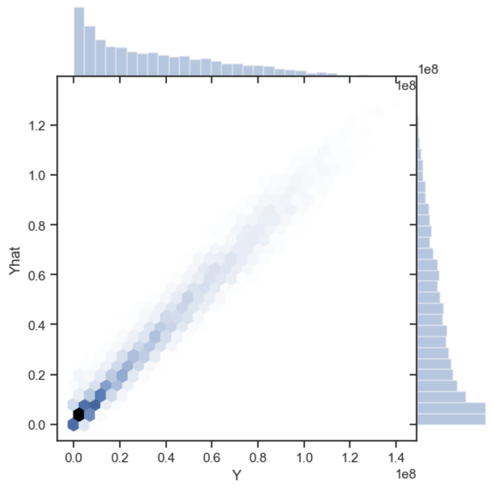


(b) Hex Plot

Figure B.24: Retail: Heating electricity

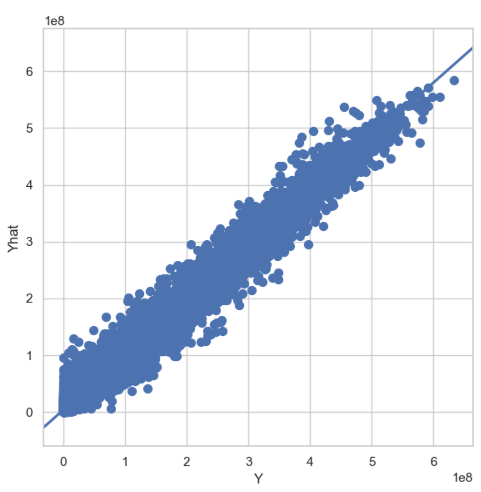


(a) Raw Data

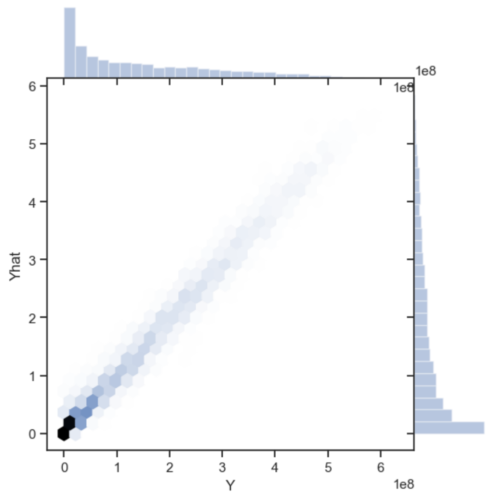


(b) Hex Plot

Figure B.25: Retail: Cooling electricity

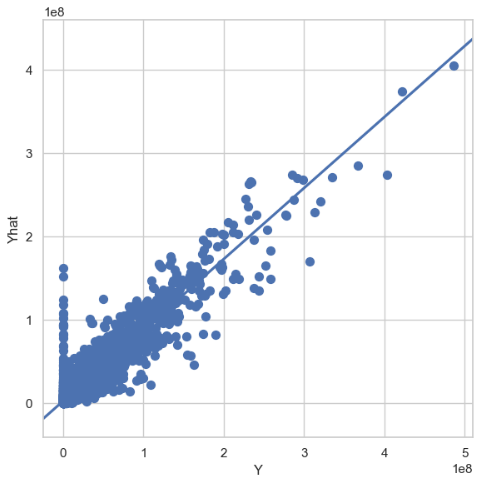


(a) Raw Data

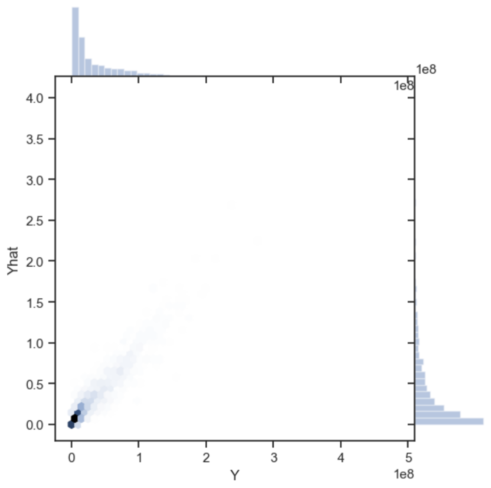


(b) Hex Plot

Figure B.26: Retail: District cooling energy

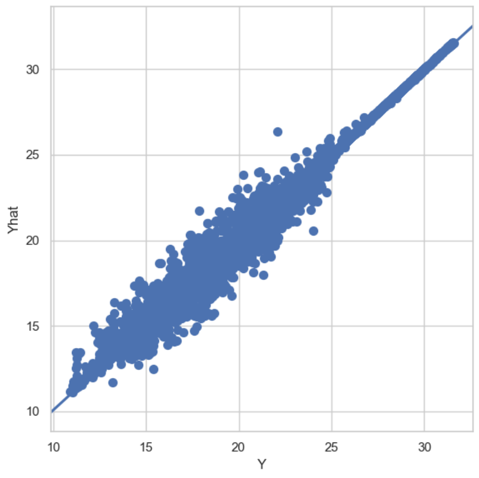


(a) Raw Data

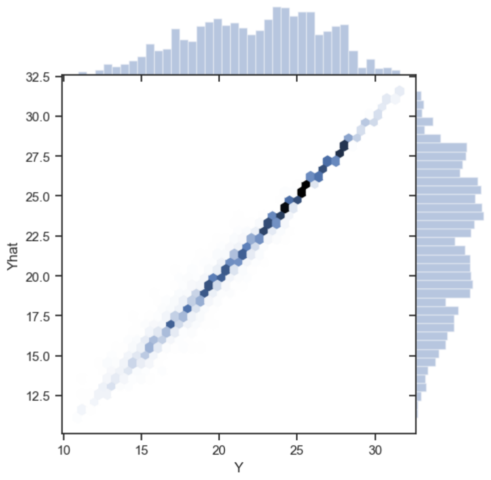


(b) Hex Plot

Figure B.27: Retail: District heating energy



(a) Raw Data



(b) Hex Plot

Figure B.28: Retail: ETS outlet temperature

B.3.3 Support Vector Regression

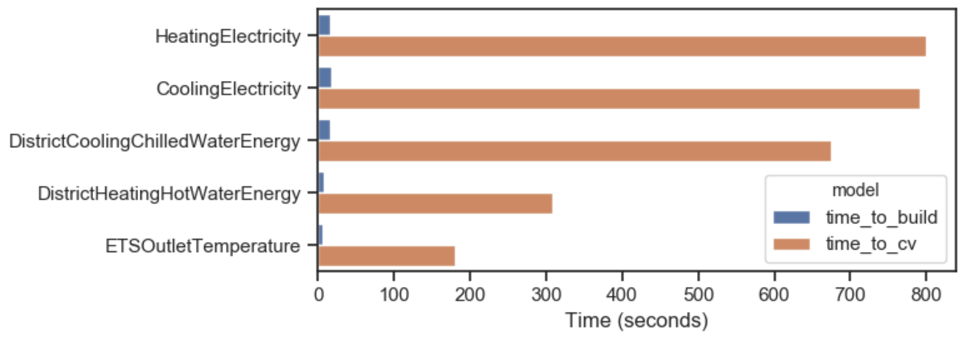


Figure B.29: Retail: SVR build and cross-validation time, for 5% down sampled data

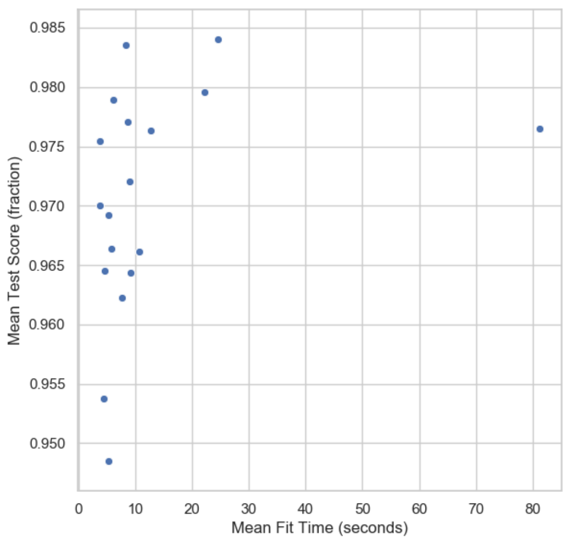
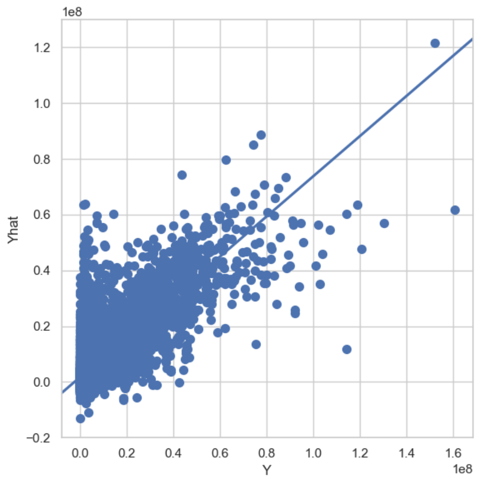
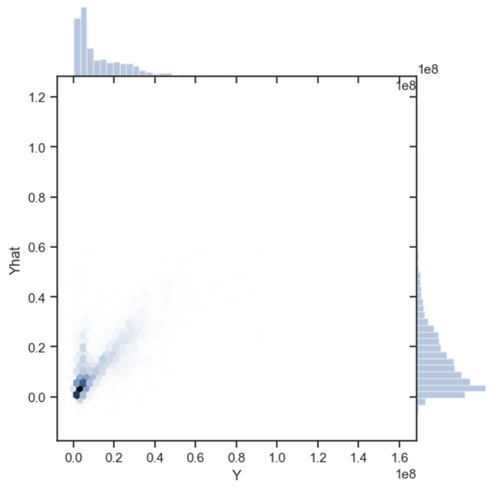


Figure B.30: Retail: SVR build and cross-validation time for ETS outlet temperature, 5% down sampled data

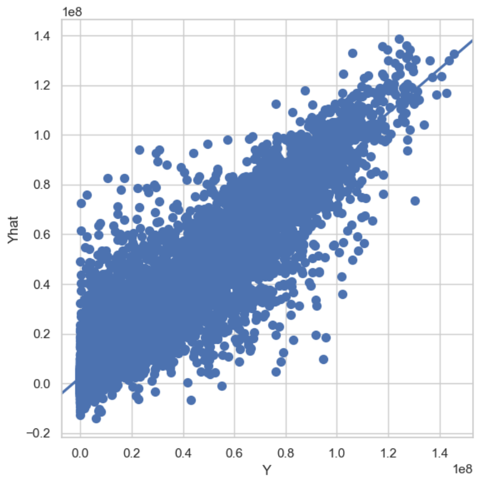


(a) Raw Data

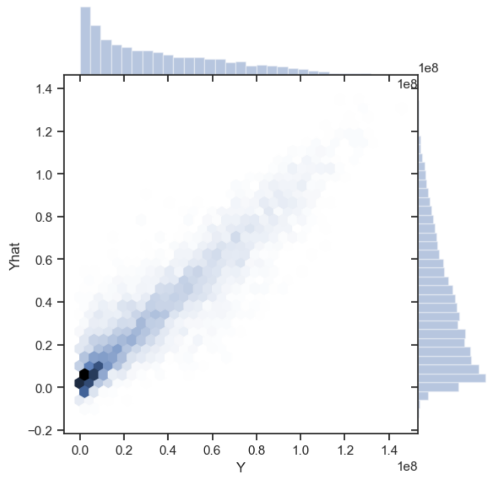


(b) Hex Plot

Figure B.31: Retail: Heating electricity

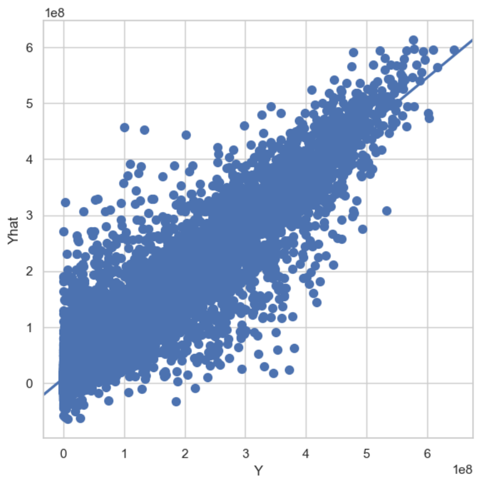


(a) Raw Data

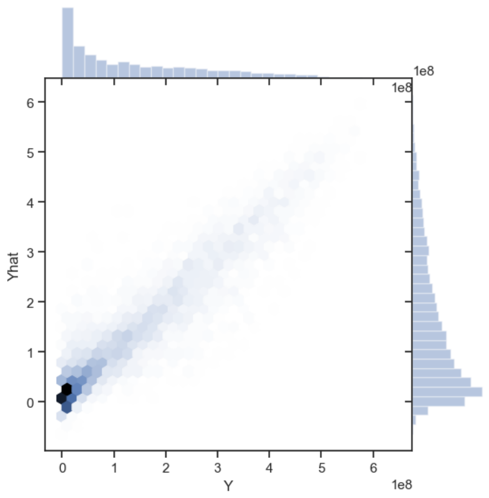


(b) Hex Plot

Figure B.32: Retail: Cooling electricity

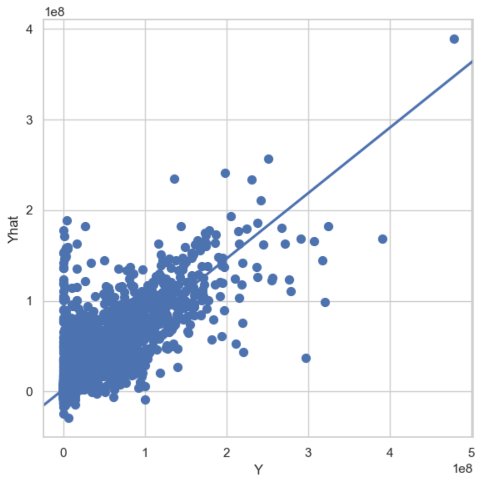


(a) Raw Data

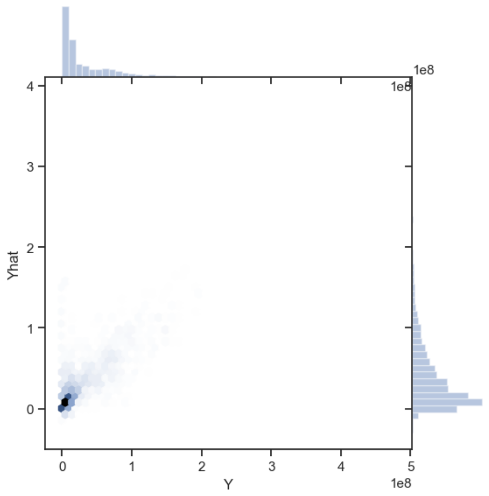


(b) Hex Plot

Figure B.33: Retail: District cooling energy

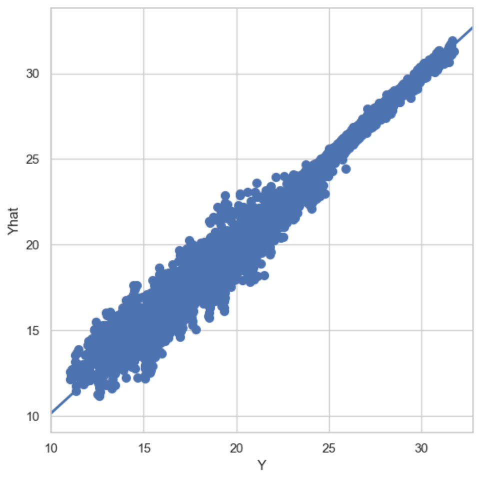


(a) Raw Data

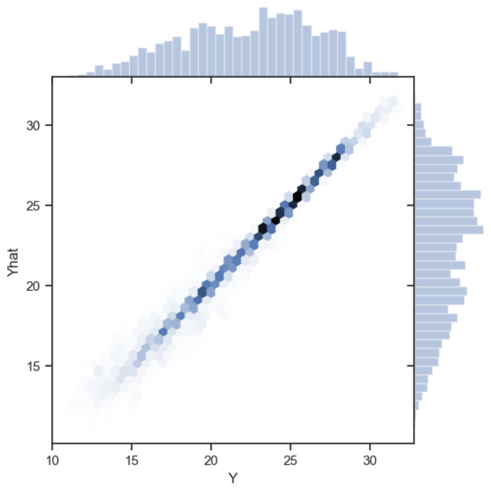


(b) Hex Plot

Figure B.34: Retail: District heating energy



(a) Raw Data



(b) Hex Plot

Figure B.35: Retail: ETS outlet temperature

B.4 Validation Results

Table B.1: Retail: PCC performance of reduced order models

Response	ROM Type	PCC
HeatingElectricity	LinearModel	0.974
HeatingElectricity	RandomForest	0.996
HeatingElectricity	SVR	0.992
CoolingElectricity	LinearModel	0.735
CoolingElectricity	RandomForest	0.987
CoolingElectricity	SVR	0.935
DistrictCoolingChilledWaterEnergy	LinearModel	0.739
DistrictCoolingChilledWaterEnergy	RandomForest	0.990
DistrictCoolingChilledWaterEnergy	SVR	0.942
DistrictHeatingHotWaterEnergy	LinearModel	0.400
DistrictHeatingHotWaterEnergy	RandomForest	0.947
DistrictHeatingHotWaterEnergy	SVR	0.832
ETSOutletTemperature	LinearModel	0.974
ETSOutletTemperature	RandomForest	0.996
ETSOutletTemperature	SVR	0.992

Table B.2: Retail: Performance of validation data

Response	ROM Type	NMBE	CVRMSE
HeatingElectricity	LinearModel	1.18	180
HeatingElectricity	RandomForest	1.81	43.7
HeatingElectricity	SVR	1.77	117
CoolingElectricity	LinearModel	4.6	114
CoolingElectricity	RandomForest	9.67	31.8
CoolingElectricity	SVR	9.93	58.6
DistrictCoolingChilledWaterEnergy	LinearModel	3.20	127
DistrictCoolingChilledWaterEnergy	RandomForest	2.03	26.0
DistrictCoolingChilledWaterEnergy	SVR	4.14	60.3
DistrictHeatingHotWaterEnergy	LinearModel	3.43	276
DistrictHeatingHotWaterEnergy	RandomForest	2.97	93.7
DistrictHeatingHotWaterEnergy	SVR	4.90	176
ETSOutletTemperature	LinearModel	1.05	180
ETSOutletTemperature	RandomForest	3.24	43.7
ETSOutletTemperature	SVR	1.75	117

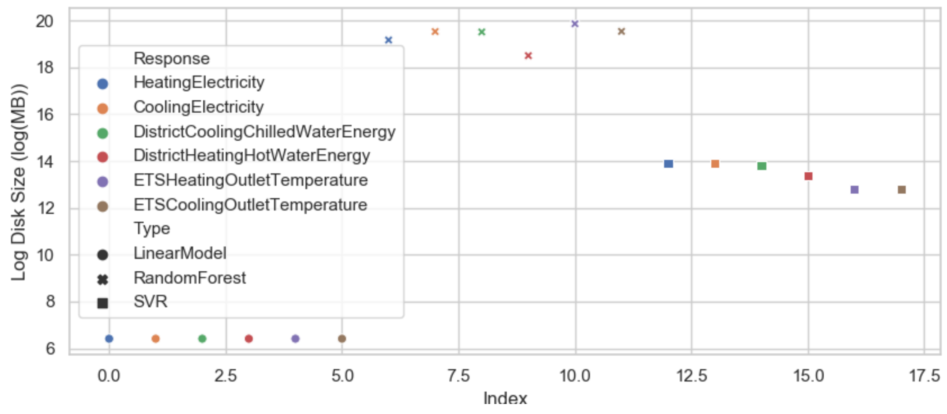


Figure B.36: Retail: Model disk size

Table B.3: Retail: Load time and runtime of reduced order models

Type	Avg Load Time (Sec)	Avg Run Time - Single (Sec)	Avg Run Time - 8760 (Sec)
Linear Model	0.0003	0.0028	0.0036
Random Forest	0.457	0.011	0.157
SVR	0.032	0.036	2.13

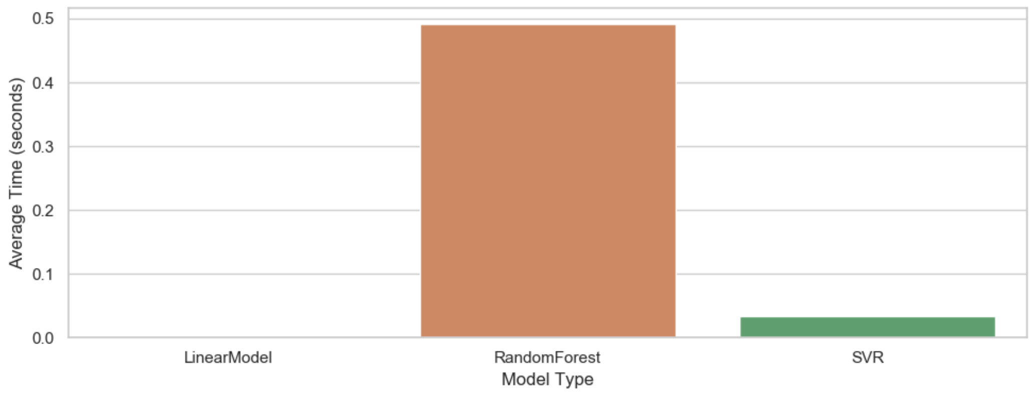


Figure B.37: Retail: Load time as a function of the model type

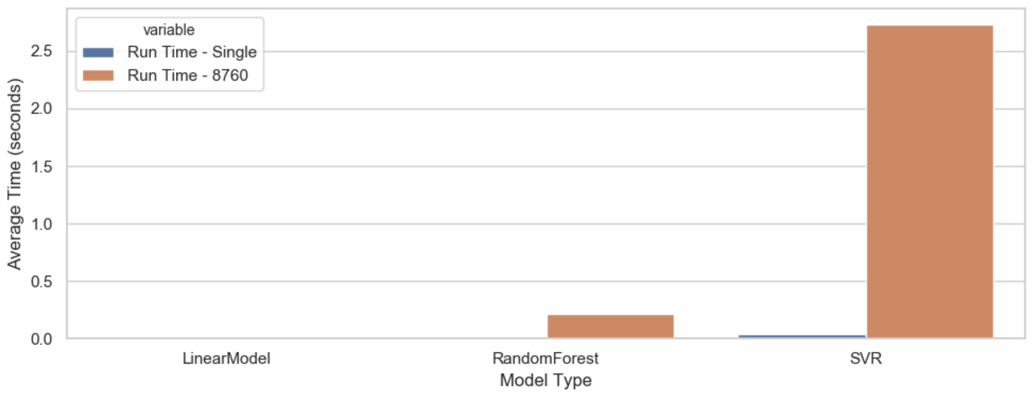


Figure B.38: Retail: Run times as a function of the model type

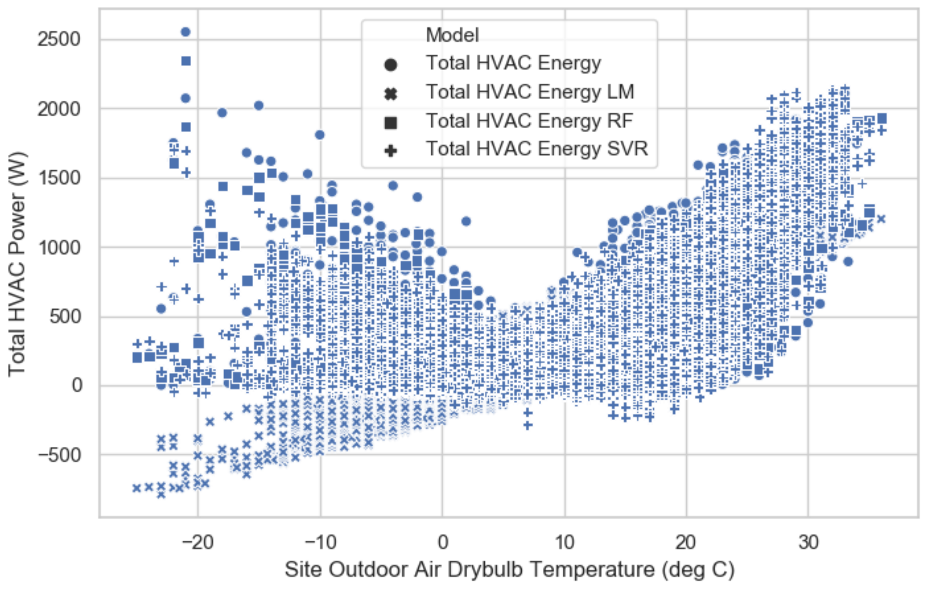


Figure B.39: Retail: Energy Consumption vs Outdoor Dry-Bulb Temperature

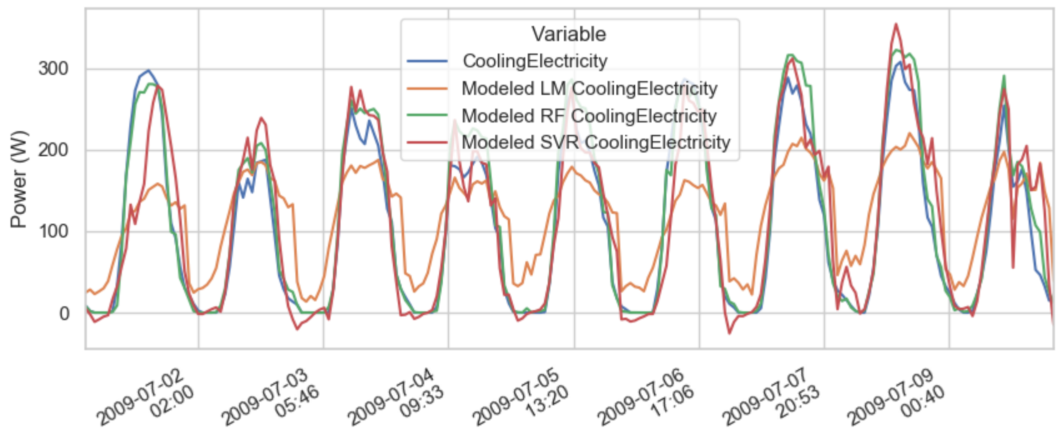


Figure B.40: Retail: Summer validation period cooling electricity performance

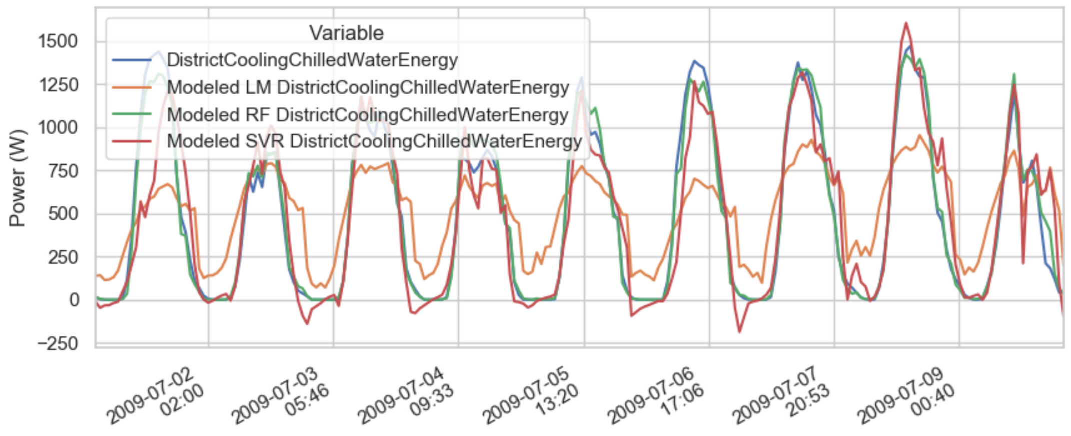


Figure B.41: Retail: Summer validation period district cooling energy

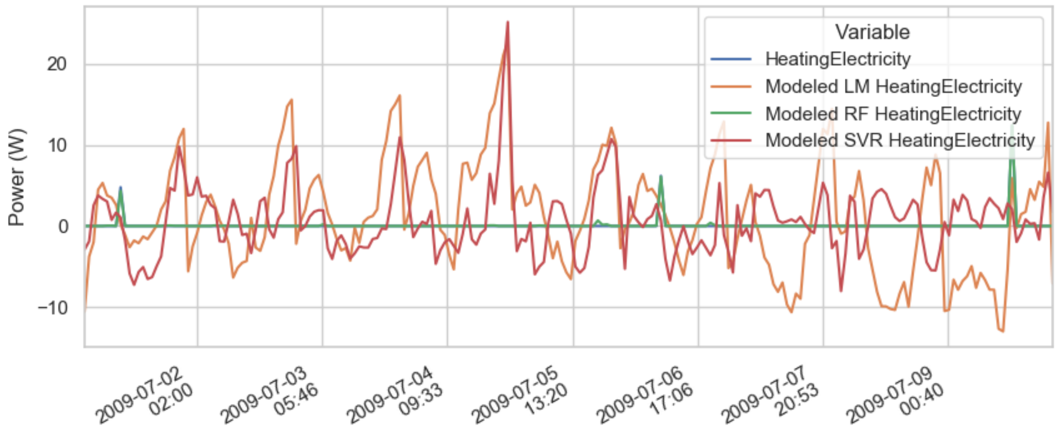


Figure B.42: Retail: Summer validation period heating electricity performance

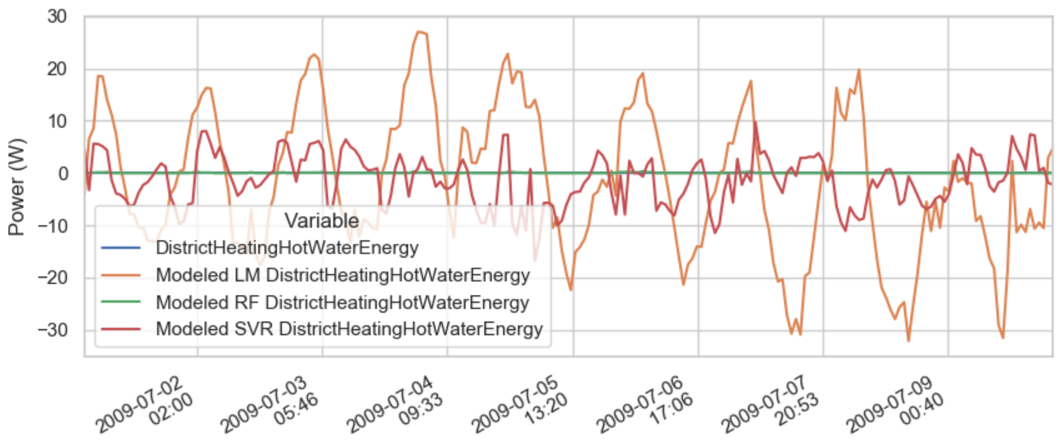


Figure B.43: Retail: Summer validation period district heating energy

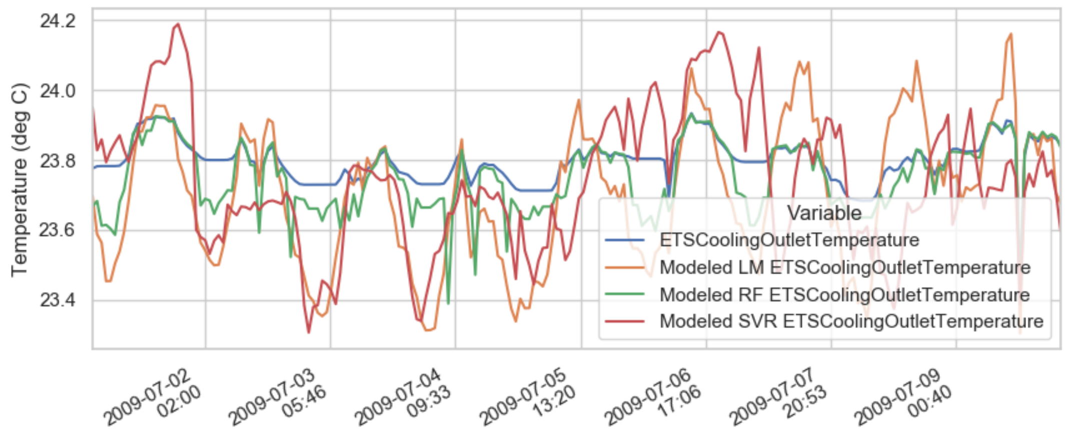


Figure B.44: Retail: Summer validation period ETS outlet temperature performance

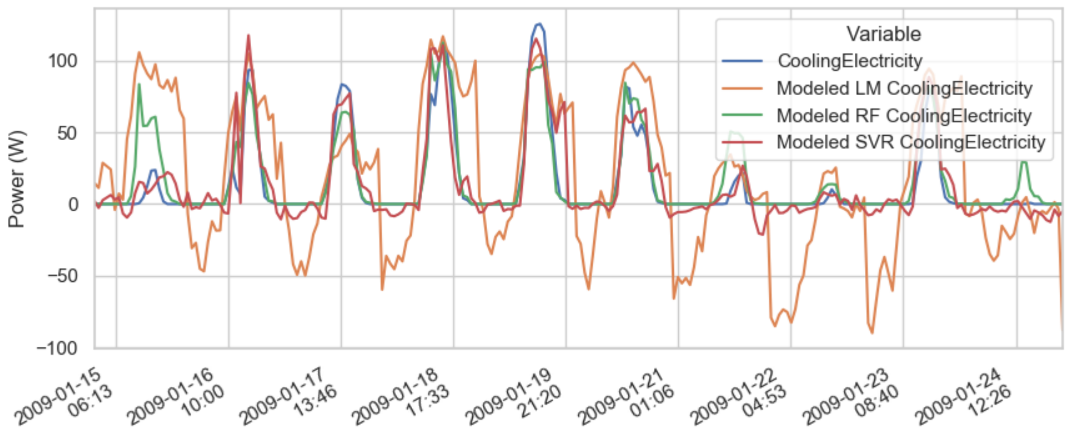


Figure B.45: Retail: Winter validation period cooling electricity performance

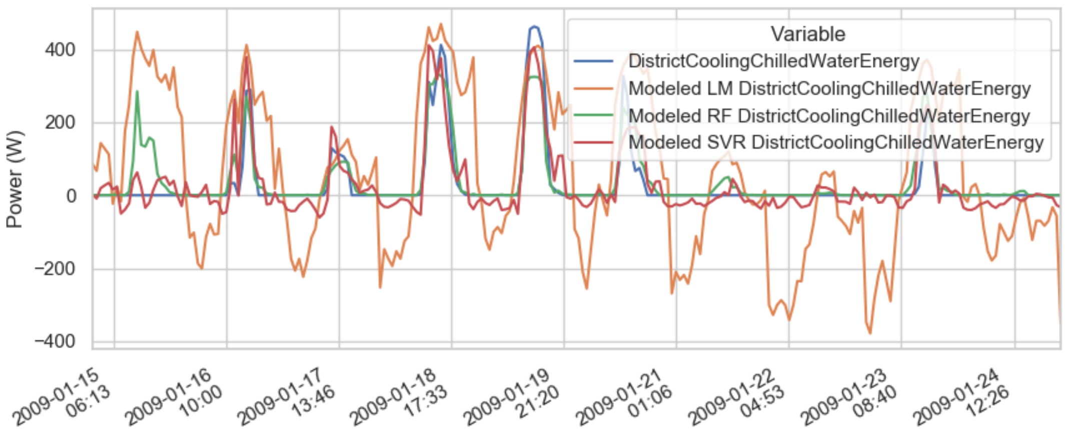


Figure B.46: Retail: Winter validation period district cooling energy

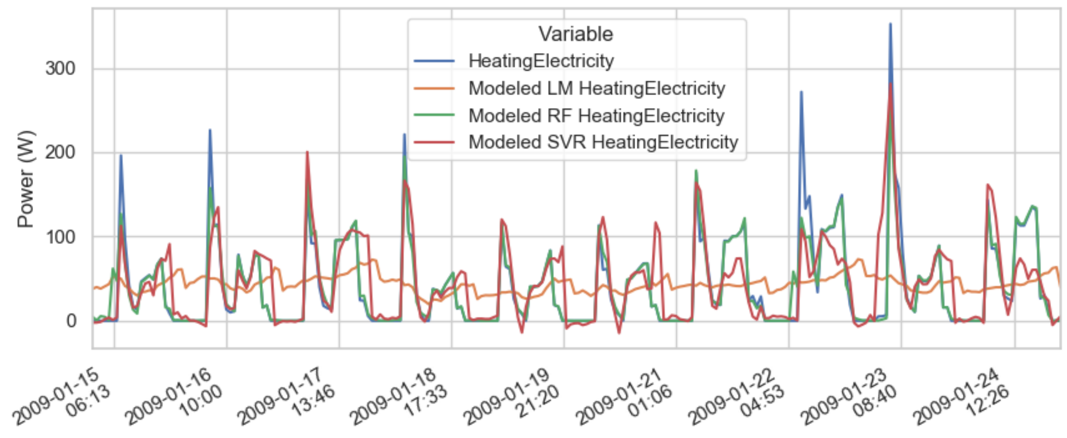


Figure B.47: Retail: Winter validation period heating electricity performance

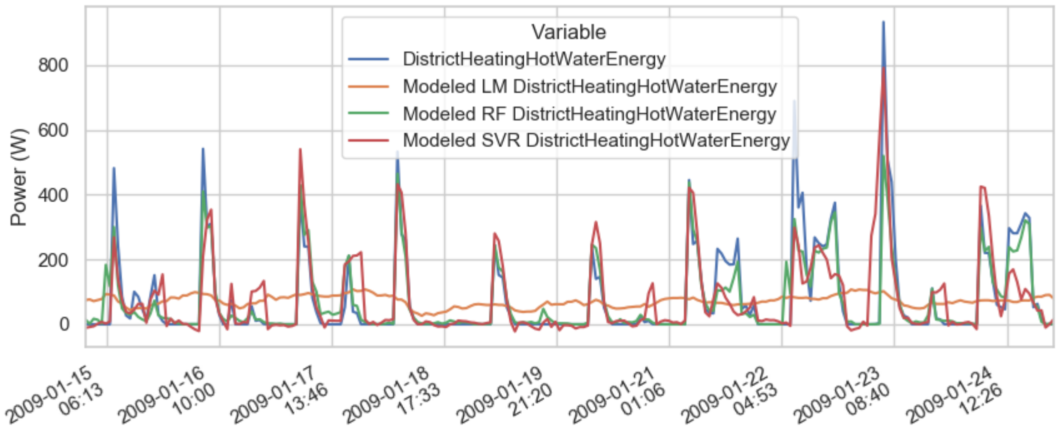


Figure B.48: Retail: Winter validation period district heating energy

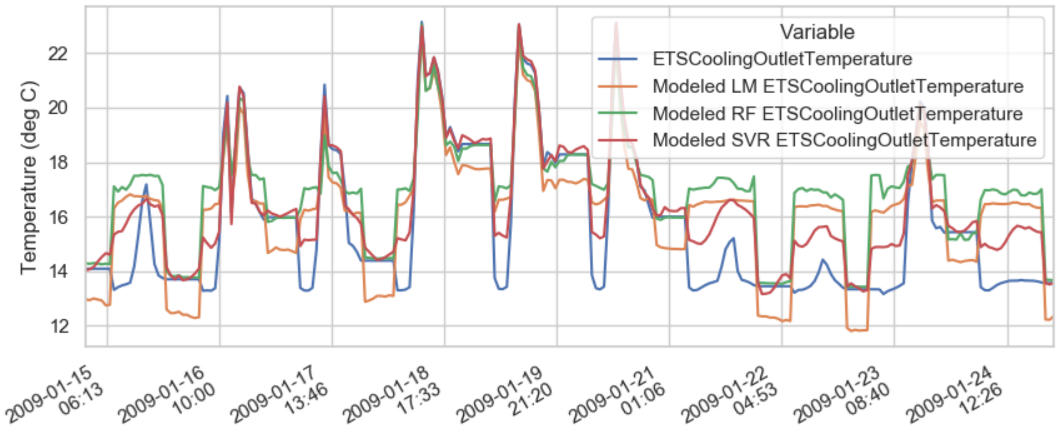


Figure B.49: Retail: Winter validation period ETS outlet temperature performance

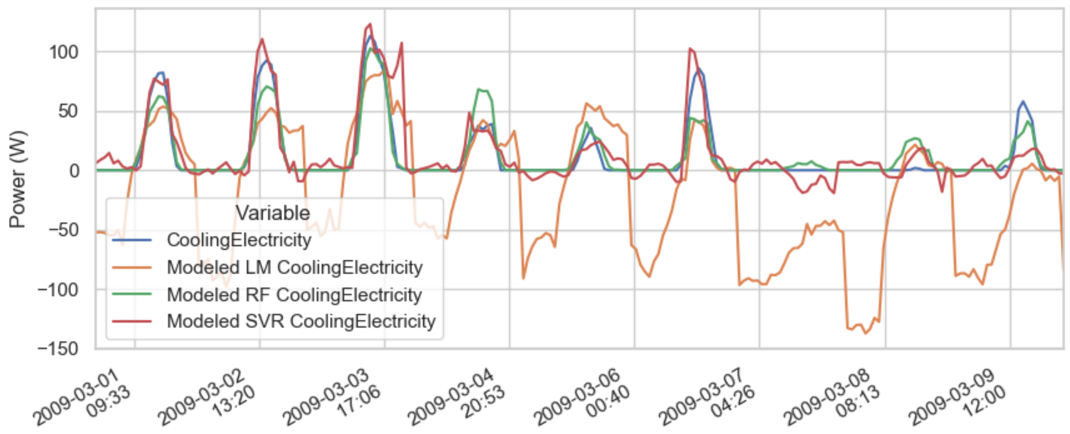


Figure B.50: Retail: Swing validation period cooling electricity performance

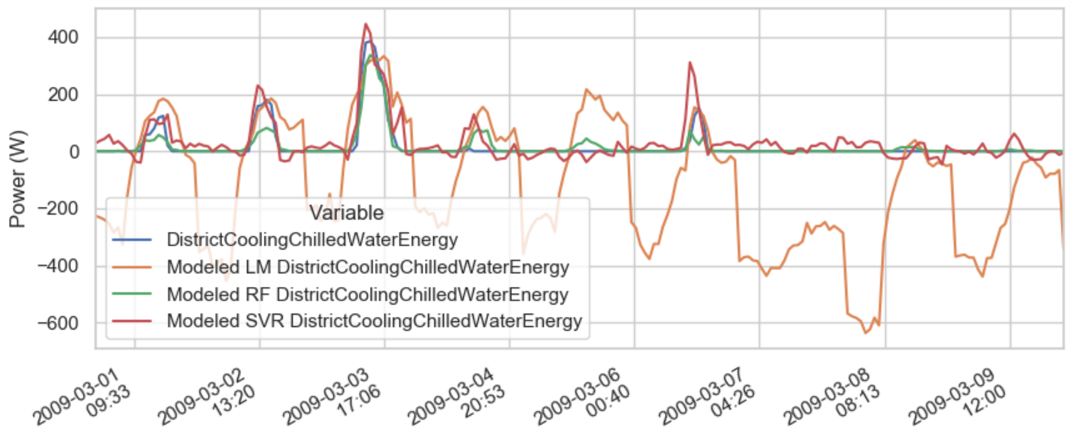


Figure B.51: Retail: Swing validation period district cooling energy

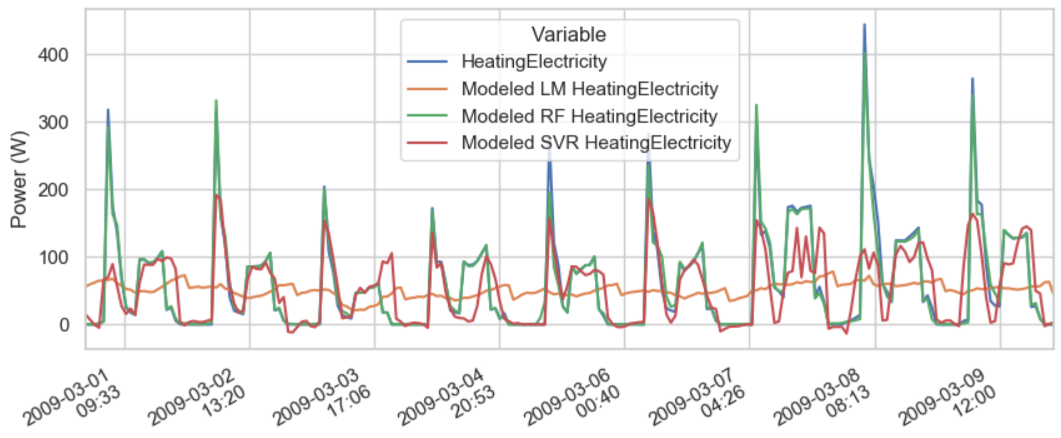


Figure B.52: Retail: Swing validation period heating electricity performance

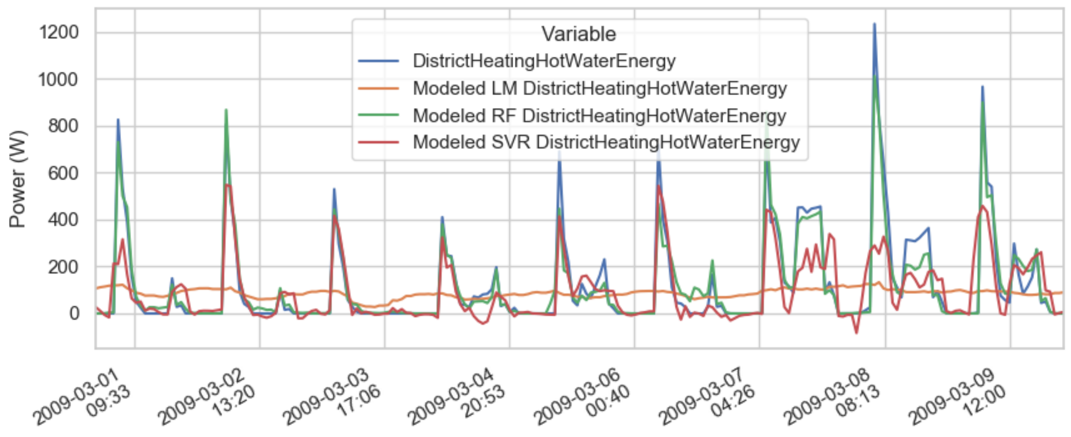


Figure B.53: Retail: Swing validation period district heating energy

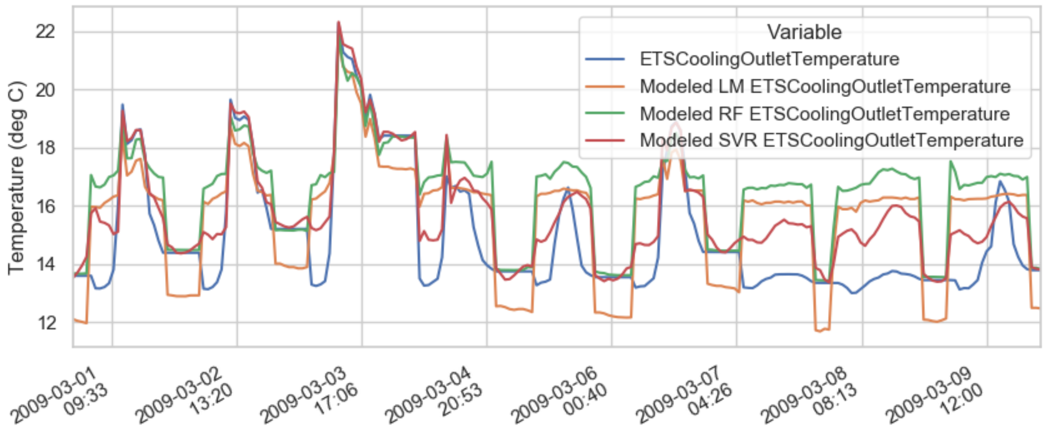


Figure B.54: Retail: Swing validation period ETS outlet temperature performance

Appendix C

OpenStudio Measures Source Code

C.1 Ambient Loop Prototype Building Measure

```
class AmbientLoopPrototypeBuilding < OpenStudio::Measure::
   ModelMeasure
  def name
    return "Ambient Loop Prototype Building"
  end

  def description
    return "Ambient Loop Prototype Building"
  end

  def modeler_description
    return "Ambient Loop Prototype Building"
  end

  def arguments(model)
    args = OpenStudio::Measure::OSArgumentVector.new

    # Make an argument for the building type
    building_type_chs = OpenStudio::StringVector.new
    building_type_chs << 'SecondarySchool'
    building_type_chs << 'PrimarySchool'
    building_type_chs << 'SmallOffice'
    building_type_chs << 'MediumOffice'
    building_type_chs << 'LargeOffice'
    building_type_chs << 'SmallHotel'
    building_type_chs << 'LargeHotel'
    building_type_chs << 'Warehouse'
    building_type_chs << 'RetailStandalone'
    building_type_chs << 'RetailStripmall'
    building_type_chs << 'QuickServiceRestaurant'
building_type_chs << 'FullServiceRestaurant'
building_type_chs << 'MidriseApartment'
building_type_chs << 'HighriseApartment'
building_type_chs << 'Hospital'
building_type_chs << 'Outpatient'
building_type = OpenStudio::Measure::OSArgument.
   makeChoiceArgument('building_type', building_type_chs, true
   )
building_type.setDisplayName('Building Type.')
building_type.setDefaultValue('SmallOffice')
args << building_type

# Make an argument for the template
template_chs = OpenStudio::StringVector.new
template_chs << 'DOE Ref Pre-1980'
template_chs << 'DOE Ref 1980-2004'
template_chs << '90.1-2004'
template_chs << '90.1-2007'
# template_chs << '189.1-2009'
template_chs << '90.1-2010'
template_chs << '90.1-2013'
template_chs << 'NECB 2011'
template = OpenStudio::Measure::OSArgument.makeChoiceArgument(
   'template', template_chs, true)
template.setDisplayName('Template.')
template.setDefaultValue('90.1-2010')
args << template

# Make an argument for the climate zone
climate_zone_chs = OpenStudio::StringVector.new
climate_zone_chs << 'ASHRAE 169-2006-1A'
# climate_zone_chs << 'ASHRAE 169-2006-1B'
climate_zone_chs << 'ASHRAE 169-2006-2A'
climate_zone_chs << 'ASHRAE 169-2006-2B'
climate_zone_chs << 'ASHRAE 169-2006-3A'
climate_zone_chs << 'ASHRAE 169-2006-3B'
climate_zone_chs << 'ASHRAE 169-2006-3C'
climate_zone_chs << 'ASHRAE 169-2006-4A'
climate_zone_chs << 'ASHRAE 169-2006-4B'
climate_zone_chs << 'ASHRAE 169-2006-4C'
climate_zone_chs << 'ASHRAE 169-2006-5A'
climate_zone_chs << 'ASHRAE 169-2006-5B'
# climate_zone_chs << 'ASHRAE 169-2006-5C'
climate_zone_chs << 'ASHRAE 169-2006-6A'
climate_zone_chs << 'ASHRAE 169-2006-6B'
climate_zone_chs << 'ASHRAE 169-2006-7A'
# climate_zone_chs << 'ASHRAE 169-2006-7B'
```

```
    climate_zone_chs << 'ASHRAE 169-2006-8A'
    # climate_zone_chs << 'ASHRAE 169-2006-8B'
    climate_zone_chs << 'NECB HDD Method'
    climate_zone = OpenStudio::Measure::OSArgument.
       makeChoiceArgument('climate_zone', climate_zone_chs, true)
    climate_zone.setDisplayName('Climate Zone.')
    climate_zone.setDefaultValue('ASHRAE 169-2006-2A')
    args << climate_zone

    return args
  end

  # define what happens when the measure is run
  def run(model, runner, user_arguments)
    super(model, runner, user_arguments)

    # Use the built-in error checking
    if !runner.validateUserArguments(arguments(model),
       user_arguments)
      return false
    end

    # Assign the user inputs to variables that can be accessed
       across the measure
    building_type = runner.getStringArgumentValue('building_type',
        user_arguments)
    template = runner.getStringArgumentValue('template',
       user_arguments)
    climate_zone = runner.getStringArgumentValue('climate_zone',
       user_arguments)


    # path is relative to the run directory
    run_dir = File.join('models')
    FileUtils.mkdir_p run_dir unless Dir.exist? run_dir
    runner.registerInfo "Found run dir to be #{run_dir}"

    new_model = OpenStudio::Model::Model.new
    new_model.create_prototype_building(building_type, template,
       climate_zone, 'USA_CO_Golden-NREL.724666_TMY3.epw')

    # For some reason new_model.save does not work inside the
       measure.
    File.open("#{run_dir}/prototype.osm", 'w') {|f| f << new_model
       .to_s}

    # remove existing objects from model
    handles = OpenStudio::UUIDVector.new
    model.objects.each do |obj|
      handles << obj.handle
    end
    model.removeObjects(handles)
    model.addObjects(new_model.toIdfFile.objects)

    # model change timestep to only one hour
    timestep = model.getTimestep
    timestep.setNumberOfTimestepsPerHour(1)

    # echo the new space's name back to the user
    runner.registerInfo("Model replaced.")

    # report final condition of model
    runner.registerFinalCondition("AmbientLoopSmallOffice Ran")

    return true
  end
end

AmbientLoopPrototypeBuilding.new.registerWithApplication
```

C.2 Change Building Location Measure

```
# Authors : Nicholas Long, David Goldwasser
# Simple measure to load the EPW file and DDY file

class ChangeBuildingLocation < OpenStudio::Measure::ModelMeasure

  Dir[File.dirname(__FILE__) + '/resources/*.rb'].each { |file|
     require file }

  # resource file modules
  include OsLib_HelperMethods

  # define the name that a user will see, this method may be
     deprecated as
  # the display name in PAT comes from the name field in measure.
     xml
  def name
    'ChangeBuildingLocation'
  end

  # define the arguments that the user will input
  def arguments(model)
    args = OpenStudio::Measure::OSArgumentVector.new
```

```
weather_file_name = OpenStudio::Measure::OSArgument.
   makeStringArgument('weather_file_name', true)
weather_file_name.setDisplayName('Weather File Name')
weather_file_name.setDescription('Name of the weather file to
   change to. This is the filename with the extension (e.g.
   NewWeather.epw). Optionally this can inclucde the full file
    path, but for most use cases should just be file name.')
args << weather_file_name

# make choice argument for climate zone
choices = OpenStudio::StringVector.new
choices << '1A'
choices << '1B'
choices << '2A'
choices << '2B'
choices << '3A'
choices << '3B'
choices << '3C'
choices << '4A'
choices << '4B'
choices << '4C'
choices << '5A'
choices << '5B'
choices << '5C'
choices << '6A'
choices << '6B'
choices << '7'
choices << '8'
choices << 'Lookup From Stat File'
climate_zone = OpenStudio::Measure::OSArgument.
   makeChoiceArgument('climate_zone', choices, true)
climate_zone.setDisplayName('Climate Zone.')
climate_zone.setDefaultValue('Lookup From Stat File')
args << climate_zone

# make an argument for use_upstream_args
use_upstream_args = OpenStudio::Measure::OSArgument.
   makeBoolArgument('use_upstream_args', true)
use_upstream_args.setDisplayName('Use Upstream Argument Values
   ')
use_upstream_args.setDescription('When true this will look for
    arguments or registerValues in upstream measures that
   match arguments from this measure, and will use the value
   from the upstream measure in place of what is entered for
   this measure.')
use_upstream_args.setDefaultValue(true)
```

```
  args << use_upstream_args

  args
end

# Define what happens when the measure is run
def run(model, runner, user_arguments)
  super(model, runner, user_arguments)

  # assign the user inputs to variables
  args = OsLib_HelperMethods.createRunVariables(runner, model,
     user_arguments, arguments(model))
  if !args then return false end

  # lookup and replace argument values from upstream measures
  if args['use_upstream_args'] == true
    args.each do |arg,value|
      next if arg == 'use_upstream_args' # this argument should
         not be changed
      value_from_osw = OsLib_HelperMethods.
         check_upstream_measure_for_arg(runner, arg)
      if !value_from_osw.empty?
        runner.registerInfo("Replacing argument named #{arg}
           from current measure with a value of #{value_from_osw
           [:value]} from #{value_from_osw[:measure_name]}.")
        new_val = value_from_osw[:value]
        # todo - make code to handle non strings more robust.
           check_upstream_measure_for_arg coudl pass bakc the
           argument type
        if arg == 'total_bldg_floor_area'
          args[arg] = new_val.to_f
        elsif arg == 'num_stories_above_grade'
          args[arg] = new_val.to_f
        elsif arg == 'zipcode'
          args[arg] = new_val.to_i
        else
          args[arg] = new_val
        end
      end
    end
  end

  # create initial condition
  if model.getWeatherFile.city != ''
    runner.registerInitialCondition("The initial weather file is
       #{model.getWeatherFile.city} and the model has #{model.
      getDesignDays.size} design day objects")
```

```
else
  runner.registerInitialCondition("No weather file is set. The
      model has #{model.getDesignDays.size} design day objects
     ")
end

# find weather file
osw_file = runner.workflow.findFile(args['weather_file_name'])
if osw_file.is_initialized
  weather_file = osw_file.get.to_s
else
  runner.registerError("Did not find #{args['weather_file_name
     ']} in paths described in OSW file.")
  return false
end

# Parse the EPW manually because OpenStudio can't handle
   multiyear weather files (or DATA PERIODS with YEARS)
epw_file = OpenStudio::Weather::Epw.load(weather_file)

weather_file = model.getWeatherFile
weather_file.setCity(epw_file.city)
weather_file.setStateProvinceRegion(epw_file.state)
weather_file.setCountry(epw_file.country)
weather_file.setDataSource(epw_file.data_type)
weather_file.setWMONumber(epw_file.wmo.to_s)
weather_file.setLatitude(epw_file.lat)
weather_file.setLongitude(epw_file.lon)
weather_file.setTimeZone(epw_file.gmt)
weather_file.setElevation(epw_file.elevation)
weather_file.setString(10, "file:///#{epw_file.filename}")

weather_name = "#{epw_file.city}_#{epw_file.state}_#{epw_file.
   country}"
weather_lat = epw_file.lat
weather_lon = epw_file.lon
weather_time = epw_file.gmt
weather_elev = epw_file.elevation

# Add or update site data
site = model.getSite
site.setName(weather_name)
site.setLatitude(weather_lat)
site.setLongitude(weather_lon)
site.setTimeZone(weather_time)
site.setElevation(weather_elev)
```

```
runner.registerInfo("city is #{epw_file.city}. State is #{
   epw_file.state}")

# Add SiteWaterMainsTemperature -- via parsing of STAT file.
stat_file = "#{File.join(File.dirname(epw_file.filename), File
   .basename(epw_file.filename, '.*'))}.stat"
unless File.exist? stat_file
  runner.registerInfo 'Could not find STAT file by filename,
     looking in the directory'
  stat_files = Dir["#{File.dirname(epw_file.filename)}/*.stat"
     ]
  if stat_files.size > 1
    runner.registerError('More than one stat file in the EPW
       directory')
    return false
  end
  if stat_files.empty?
    runner.registerError('Cound not find the stat file in the
       EPW directory')
    return false
  end

  runner.registerInfo "Using STAT file: #{stat_files.first}"
  stat_file = stat_files.first
end
unless stat_file
  runner.registerError 'Could not find stat file'
  return false
end

stat_model = EnergyPlus::StatFile.new(stat_file)
water_temp = model.getSiteWaterMainsTemperature
water_temp.setAnnualAverageOutdoorAirTemperature(stat_model.
   mean_dry_bulb)
water_temp.
   setMaximumDifferenceInMonthlyAverageOutdoorAirTemperatures(
   stat_model.delta_dry_bulb)
runner.registerInfo("mean dry-bulb is #{stat_model.
   mean_dry_bulb}")

# Remove all the Design Day objects that are in the file
model.getObjectsByType('OS:SizingPeriod:DesignDay'.
   to_IddObjectType).each(&:remove)

# find the ddy files
ddy_file = "#{File.join(File.dirname(epw_file.filename), File.
   basename(epw_file.filename, '.*'))}.ddy"
```

```
unless File.exist? ddy_file
  ddy_files = Dir["#{File.dirname(epw_file.filename)}/*.ddy"]
  if ddy_files.size > 1
    runner.registerError('More than one ddy file in the EPW
       directory')
    return false
  end
  if ddy_files.empty?
    runner.registerError('could not find the ddy file in the
       EPW directory')
    return false
  end

  ddy_file = ddy_files.first
end

unless ddy_file
  runner.registerError "Could not find DDY file for #{ddy_file
     }"
  return error
end

ddy_model = OpenStudio::EnergyPlus.loadAndTranslateIdf(
   ddy_file).get
ddy_model.getObjectsByType('OS:SizingPeriod:DesignDay'.
   to_IddObjectType).each do |d|
  # grab only the ones that matter
  ddy_list = /(Htg 99.6. Condns DB)|(Clg .4. Condns WB=>MDB)|(
     Clg .4% Condns DB=>MWB)/
  if d.name.get =~ ddy_list
    runner.registerInfo("Adding object #{d.name}")

    # add the object to the existing model
    model.addObject(d.clone)
  end
end

# Set climate zone
climateZones = model.getClimateZones
if args['climate_zone'] == 'Lookup From Stat File'

  # get climate zone from stat file
  text = nil
  File.open(stat_file) do |f|
    text = f.read.force_encoding('iso-8859-1')
  end
```

```
      # Get Climate zone.
      # - Climate type "3B" (ASHRAE Standard 196-2006 Climate Zone
         )**
      # - Climate type "6A" (ASHRAE Standards 90.1-2004 and
         90.2-2004 Climate Zone)**
      regex = /Climate type \"(.*?)\" \(ASHRAE Standards?(.*)\)
         \*\*/
      match_data = text.match(regex)
      if match_data.nil?
        runner.registerWarning("Can't find ASHRAE climate zone in
           stat file.")
      else
        args['climate_zone'] = match_data[1].to_s.strip
      end

    end
    # set climate zone
    climateZones.clear
    climateZones.setClimateZone('ASHRAE', args['climate_zone'])
    runner.registerInfo("Setting Climate Zone to #{climateZones.
       getClimateZones('ASHRAE').first.value}")

    # add final condition
    runner.registerFinalCondition("The final weather file is #{
       model.getWeatherFile.city} and the model has #{model.
       getDesignDays.size} design day objects.")

    true
  end
end

# This allows the measure to be use by the application
ChangeBuildingLocation.new.registerWithApplication
```

C.3 Ambient Loop Add ETS Measure

```
class AmbientLoopAddEtsSystem < OpenStudio::Measure::ModelMeasure
  def name
    return "Ambient Loop Add ETS System"
  end

  def description
    return "Apply an ETS system to a model"
  end

  def modeler_description
    return "This measure removes the existing HVAC system and
```

```
     replaces it with an energy transfer station."
end

# define the arguments that the user will input
def arguments(model)
  args = OpenStudio::Measure::OSArgumentVector.new

  return args
end

# Return the list of thermal zones that will have an ETS (
   exclude attics)
def get_thermal_zones(model)
  zones = []
  model.getThermalZones.each do |thermal_zone|
    add_zone = true
    thermal_zone.spaces.each do |space|
      next unless space.spaceType.is_initialized
      next unless space.spaceType.get.standardsSpaceType.
         is_initialized

      if space.spaceType.get.standardsSpaceType.get == 'Attic'
        add_zone = false
      end
    end
    zones << thermal_zone if add_zone
  end

  return zones
end

# define what happens when the measure is run
def run(model, runner, user_arguments)
  super(model, runner, user_arguments)

  # Use the built-in error checking
  if !runner.validateUserArguments(arguments(model),
     user_arguments)
    return false
  end

  # path is relative to the run directory
  run_dir = File.join('models')
  FileUtils.mkdir_p run_dir unless Dir.exist? run_dir

  runner.registerInfo "trying to remove HVAC equipment"
  model.remove_prm_hvac
```

```
    File.open("#{run_dir}/prototype-no-hvac.osm", 'w') {|f| f <<
       model.to_s}

    # add in the ambient loop model -- this is definitely not
       right. This adds a water to air heat pump
    model.add_energy_transfer_station("Water-to-Air Heat Pump",
       get_thermal_zones(model))

    File.open("#{run_dir}/final.osm", 'w') {|f| f << model.to_s}

    return true
  end
end

# register the measure to be used by the application
AmbientLoopAddEtsSystem.new.registerWithApplication
```

C.4 Ambient Loop Temperature Setpoint Measure

```
class AmbientLoopTemperatureSetpoint < OpenStudio::Ruleset::
   ModelUserScript
  def name
    return "Ambient Loop Temperature Setpoint"
  end

  def description
    return "Set the temperature of the ambient loop to a specific
       value."
  end

  def modeler_description
    return "There are naming restrictions in this measure. Plant
       loop must be named 'Ambient Loop'"
  end

  # define the arguments that the user will input
  def arguments(model)
    args = OpenStudio::Ruleset::OSArgumentVector.new

    # the name of the space to add to the model
    setpoint = OpenStudio::Ruleset::OSArgument.makeDoubleArgument(
       "setpoint_temperature", true)
    setpoint.setUnits("Degrees Celsius")
    setpoint.setDisplayName("Ambient Loop Temperature")
    setpoint.setDefaultValue(20)
    setpoint.setDescription("Temperature setpoint for Ambient Loop
     ")
  args << setpoint

  delta = OpenStudio::Ruleset::OSArgument.makeDoubleArgument("
     design_delta", true)
  delta.setUnits("Delta Temperature")
  delta.setDefaultValue(5.55) # 10 Deg F default delta
  delta.setDisplayName("Delta Design Loop Temperature")
  delta.setDescription("Set the delta design temperature for the
      ambient loop")
  args << delta

  return args
end

# define what happens when the measure is run
def run(model, runner, user_arguments)
  super(model, runner, user_arguments)

  # use the built-in error checking
  return false unless runner.validateUserArguments(arguments(
     model), user_arguments)

  # assign the user inputs to variables
  setpoint = runner.getDoubleArgumentValue("setpoint_temperature
     ", user_arguments)
  delta = runner.getDoubleArgumentValue("design_delta",
     user_arguments)

  # This measure only works with the predifined loop name of `
     Ambient Loop`
  plant_loop = model.getPlantLoopByName('Ambient Loop').get

  # try and set the temperature of the ambient loop - this
     includes setting the
  # plant loop min/max temperatures, the sizing plant objects,
     and the schedules
  loop_sizing = plant_loop.sizingPlant
  loop_sizing.setDesignLoopExitTemperature(setpoint)
  loop_sizing.setLoopDesignTemperatureDifference(delta)

  plant_loop.supplyOutletNode.setpointManagers.each {|sm| sm.
     remove}

  amb_loop_schedule = OpenStudio::Model::ScheduleRuleset.new(
     model)
  amb_loop_schedule.setName("Ambient Loop Temperature Ruleset")
```

```
    amb_loop_schedule.defaultDaySchedule.setName("Ambient Loop
       Temperature - Default")
    amb_loop_schedule.defaultDaySchedule.addValue(OpenStudio::Time
       .new(0, 24, 0, 0), setpoint)

    amb_stpt_manager = OpenStudio::Model::SetpointManagerScheduled
       .new(model, amb_loop_schedule)
    amb_stpt_manager.setName('Ambient Loop Setpoint Manager -
       Scheduled')
    amb_stpt_manager.setControlVariable("Temperature")
    amb_stpt_manager.addToNode(plant_loop.supplyOutletNode)

    # report final condition of model
    runner.registerFinalCondition("The final maximum loop
       temperature is: #{setpoint}")

    return true
  end
end

# register the measure to be used by the application
AmbientLoopTemperatureSetpoint.new.registerWithApplication
```

C.5 Hot Water Loop Temperature Setpoint Measure

```
class HotWaterLoopDesignTemperature < OpenStudio::Measure::
   ModelMeasure
  def name
    return "Hot Water Loop Design Temperature"
  end

  def description
    return "Set the design temperature of the Hot Water Loop to
       the specified value."
  end

  def modeler_description
    return "The name of the loop must be 'Hot Water Loop'"
  end

  # define the arguments that the user will input
  def arguments(model)
    args = OpenStudio::Ruleset::OSArgumentVector.new

    # the name of the space to add to the model
    temp = OpenStudio::Ruleset::OSArgument.makeDoubleArgument("
       hot_water_temperature", true)
  temp.setDisplayName("Hot Water Temperature")
  temp.setDescription("Design temperature of the hot water loop.
      Name must be Hot Water Loop")
  args << temp

  return args
end

# define what happens when the measure is run
def run(model, runner, user_arguments)
  super(model, runner, user_arguments)

  # use the built-in error checking
  if !runner.validateUserArguments(arguments(model),
     user_arguments)
    return false
  end

  # assign the user inputs to variables
  temp = runner.getDoubleArgumentValue("hot_water_temperature",
     user_arguments)

  # get the hot water loop
  # This measure only works with the predifined loop name of `
     Ambient Loop`
  plant_loop = model.getPlantLoopByName('Ambient Loop').get

  # try and set the temperature of the ambient loop - this
     includes setting the
  # plant loop min/max temperatures, the sizing plant objects,
     and the schedules
  loop_sizing = plant_loop.sizingPlant

  # report initial condition of model
  runner.registerInitialCondition("Hot water loop design
     temperature started with #{loop_sizing.
     designLoopExitTemperature}")

  loop_sizing.setDesignLoopExitTemperature(temp)

  # report final condition of model
  runner.registerFinalCondition("Hot water loop design
     temperature ended with #{loop_sizing.
     designLoopExitTemperature}")

  return true
end
```

```
end

# register the measure to be used by the application
HotWaterLoopDesignTemperature.new.registerWithApplication
```

C.6 Internal Loads Multiplier Measure

```
class InternalLoadsMultiplier < OpenStudio::Measure::ModelMeasure
  # require all .rb files in resources folder
  Dir[File.dirname(__FILE__) + '/resources/*.rb'].each {|file|
     require file}

  # resource file modules
  include OsLib_HelperMethods

  # human readable name
  def name
    return "Internal Loads Multiplier"
  end

  # human readable description
  def description
    return "Multipliers for LPD, EPD, and people densities."
  end

  # human readable description of modeling approach
  def modeler_description
    return "Multipliers for LPD, EPD, and people densities."
  end

  # define the arguments that the user will input
  def arguments(model)
    args = OpenStudio::Ruleset::OSArgumentVector.new

    lpd = OpenStudio::Ruleset::OSArgument.makeDoubleArgument("
       lpd_multiplier", true)
    lpd.setDisplayName("LPD Multiplier")
    lpd.setDefaultValue(1.0)
    lpd.setUnits("W/ft^2") # The resulting unit
    lpd.setDescription("Multiply the LPD in the building by this
       multiplier. Retail LPD is typically 1.3, Small Office 1.0")
    args << lpd

    epd = OpenStudio::Ruleset::OSArgument.makeDoubleArgument("
       epd_multiplier", true)
    epd.setDisplayName("Electric Equipment Power Density
       Multiplier")
  epd.setDefaultValue(1.0)
  epd.setUnits("W/ft^2") # The resulting unit
  epd.setDescription("Multiply the EPD in the building by this
     value")
  args << epd

  people_per_floor_area = OpenStudio::Ruleset::OSArgument.
     makeDoubleArgument("people_per_floor_area_multiplier", true
     )
  people_per_floor_area.setDisplayName("People per floor area
     multipleir")
  people_per_floor_area.setDefaultValue(1.0)
  people_per_floor_area.setUnits("People/ft^2") # The resulting
     unit
  args << people_per_floor_area

  return args
end

# define what happens when the measure is run
def run(model, runner, user_arguments)
  super(model, runner, user_arguments)

  # assign the user inputs to variables
  args = OsLib_HelperMethods.createRunVariables(runner, model,
     user_arguments, arguments(model))

  return false unless args

  # array of altered lighting defitinos (tracking so isn't
     altered twice)
  altered_light_defs = []

  ave_lpd = 0
  ave_lpd_count = 0
  ave_epd = 0
  ave_pd = 0
  ave_space_count = 0

  # loop through space types altering loads
  model.getSpaceTypes.each do |space_type|
    next if space_type.spaces.size == 0
    next if space_type.standardsSpaceType.get == 'Attic'

    # update lights
    space_type.lights.each do |light|
      light_def = light.lightsDefinition
```

```
        unless altered_light_defs.include? light_def
          light_def.setWattsperSpaceFloorArea(light_def.
             wattsperSpaceFloorArea.get * args['lpd_multiplier'])
          altered_light_defs << light_def
          ave_lpd += light_def.wattsperSpaceFloorArea.get
          ave_lpd_count += 1
        end
      end

      # replace electric equipment
      if space_type.electricEquipmentPowerPerFloorArea.
         is_initialized
        space_type.setElectricEquipmentPowerPerFloorArea(
           space_type.electricEquipmentPowerPerFloorArea.get *
           args['epd_multiplier'])
        ave_epd += space_type.electricEquipmentPowerPerFloorArea.
           get
      end

      # replace people
      if space_type.peoplePerFloorArea.is_initialized
        space_type.setPeoplePerFloorArea(space_type.
           peoplePerFloorArea.get * args['
           people_per_floor_area_multiplier'])
        ave_pd += space_type.peoplePerFloorArea.get
      end

      ave_space_count += 1
    end

    if ave_lpd_count > 0
      runner.registerValue('lpd_average', ave_lpd / ave_lpd_count,
          'W/m2')
    end

    if ave_space_count > 0
      runner.registerValue('epd_average', ave_epd /
         ave_space_count, 'W/m2')
      runner.registerValue('ppl_average', ave_pd / ave_space_count
         , 'People/m2')
    end

    return true
  end
end

# register the measure to be used by the application
```

```
InternalLoadsMultiplier.new.registerWithApplication
```

C.7 Ambient Loop Reports Measure

```
require 'erb'
require 'date'

class AmbientLoopReports < OpenStudio::Measure::ReportingMeasure
  def name
    return 'Ambient Loop Reports'
  end

  def description
    return 'Add report variables for post-processing the ambient
       loop data.'
  end

  def modeler_description
    return 'Reporting Variables for Ambient Loop'
  end

  def log(str)
    puts "#{Time.now}: #{str}"
  end

  # define the arguments that the user will input
  def arguments
    args = OpenStudio::Ruleset::OSArgumentVector.new

    # this measure does not require any user arguments, return an
       empty list
    return args
  end

  # return a vector of IdfObject's to request EnergyPlus objects
     needed by the run method
  def energyPlusOutputRequests(runner, user_arguments)
    super(runner, user_arguments)

    result = OpenStudio::IdfObjectVector.new

    # use the built-in error checking
    return result unless runner.validateUserArguments(arguments,
       user_arguments)

    # Output:Variable,*,Facility Heating Setpoint Not Met Time,
       hourly; !- Zone Sum [hr]
```

```
# Output:Variable,*,Facility Cooling Setpoint Not Met Time,
   hourly; !- Zone Sum [hr]
# Output:Variable,*,Facility Heating Setpoint Not Met While
   Occupied Time,hourly; !- Zone Sum [hr]
# Output:Variable,*,Facility Cooling Setpoint Not Met While
   Occupied Time,hourly; !- Zone Sum [hr]

result << OpenStudio::IdfObject.load('Output:Variable,,
   District Cooling Inlet Temperature,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,,
   District Cooling Outlet Temperature,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,,
   District Cooling Mass Flow Rate,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,,
   District Heating Inlet Temperature,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,,
   District Heating Outlet Temperature,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,,
   District Heating Mass Flow Rate,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,,
   District Heating Hot Water Energy,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,,
   District Cooling Chilled Water Energy,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,,Site
   Mains Water Temperature,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,,Site
   Outdoor Air Drybulb Temperature,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,,Site
   Outdoor Air Relative Humidity,timestep;').get
result << OpenStudio::IdfObject.load('Output:Meter,Cooling:
   Electricity,timestep;').get
result << OpenStudio::IdfObject.load('Output:Meter,Heating:
   Electricity,timestep;').get
result << OpenStudio::IdfObject.load('Output:Variable,*,Zone
   Predicted Sensible Load to Setpoint Heat Transfer Rate,
   hourly,timestep;').get
result << OpenStudio::IdfObject.load('Output:Meter,Heating:Gas
   ,timestep;').get
result << OpenStudio::IdfObject.load('Output:Meter,
   InteriorLights:Electricity,timestep;').get
result << OpenStudio::IdfObject.load('Output:Meter,Fans:
   Electricity,timestep;').get
result << OpenStudio::IdfObject.load('Output:Meter,
   InteriorEquipment:Electricity,timestep;').get
result << OpenStudio::IdfObject.load('Output:Meter,
   ExteriorLighting:Electricity,timestep;').get
result << OpenStudio::IdfObject.load('Output:Meter,Electricity
     :Facility,timestep;').get
  result << OpenStudio::IdfObject.load('Output:Meter,Gas:
     Facility,timestep;').get

  return result
end

def extract_timeseries_into_matrix(sqlfile, data, variable_name,
    key_value = nil)
  log "Executing query for #{variable_name}"
  if key_value
    ts = sqlfile.timeSeries('RUN PERIOD 1', 'Zone Timestep',
       variable_name, key_value)
  else
    ts = sqlfile.timeSeries('RUN PERIOD 1', 'Zone Timestep',
       variable_name)
  end
  log 'Iterating over timeseries'
  column = [variable_name.delete(':')] # Set the header of the
     data to the variable name, removing :
  unless ts.empty?
    ts = ts.get if ts.respond_to?(:get)
    ts = ts.first if ts.respond_to?(:first)

    start = Time.now
    # Iterating in OpenStudio can take up to 60 seconds with 10
       min data. The quick_proc takes 0.03 seconds.
    # for i in 0..ts.values.size - 1
    #   log "... at #{i}" if i % 10000 == 0
    #   column << ts.values[i]
    # end

    quick_proc = ts.values.to_s.split(',')

    # the first and last have some cleanup items because of the
       Vector method
    quick_proc[0] = quick_proc[0].gsub(/^.*\(/, '')
    quick_proc[-1] = quick_proc[-1].delete(')')
    column += quick_proc

    log "Took #{Time.now - start} to iterate"
  end

  log 'Appending column to data'

  # append the data to the end of the rows
  if column.size == data.size
```

```
    data.each_index do |index|
      data[index] << column[index]
    end
  end
  log "Finished extracting #{variable_name}"
end

# define what happens when the measure is run
def run(runner, user_arguments)
  super(runner, user_arguments)

  # use the built-in error checking
  return false unless runner.validateUserArguments(arguments,
     user_arguments)

  # get the last model and sql file
  model = runner.lastOpenStudioModel
  if model.empty?
    runner.registerError('Cannot find last model.')
    return false
  end
  model = model.get

  sqlFile = runner.lastEnergyPlusSqlFile
  if sqlFile.empty?
    runner.registerError('Cannot find last sql file.')
    return false
  end
  sqlFile = sqlFile.get
  model.setSqlFile(sqlFile)

  # create a new csv with the values and save to the reports
     direcoty.
  # assumptions:
  #   - all the variables exist
  #   - data are the same length

  # initialize the rows with the header
  puts 'Starting to process Timeseries data'
  rows = [
      # Initial header row
      ['Date Time', 'Month', 'Day', 'Day of Week', 'Hour', '
         Minute']
  ]

  # just grab one of the variables to get the date/time stamps
  ts = sqlFile.timeSeries('RUN PERIOD 1', 'Zone Timestep', '
     Cooling:Electricity')
  unless ts.empty?
    ts = ts.first

    # Save off the date time values
    ts.dateTimes.each_with_index do |dt, _index|
      rows << [DateTime.parse(dt.to_s).strftime('%m/%d/%Y %H:%M'
         ), dt.date.monthOfYear.value, dt.date.dayOfMonth, dt.
         date.dayOfWeek.value, dt.time.hours, dt.time.minutes]
    end
  end

  # add in the other variables by columns -- should really pull
     this from the report variables defined above
  extract_timeseries_into_matrix(sqlFile, rows, 'Site Outdoor
     Air Drybulb Temperature', 'Environment')
  extract_timeseries_into_matrix(sqlFile, rows, 'Site Outdoor
     Air Relative Humidity', 'Environment')
  extract_timeseries_into_matrix(sqlFile, rows, 'Heating:
     Electricity')
  extract_timeseries_into_matrix(sqlFile, rows, 'Heating:Gas')
  extract_timeseries_into_matrix(sqlFile, rows, 'Cooling:
     Electricity')
  extract_timeseries_into_matrix(sqlFile, rows, 'Electricity:
     Facility')
  extract_timeseries_into_matrix(sqlFile, rows, 'Gas:Facility')
  extract_timeseries_into_matrix(sqlFile, rows, 'District
     Heating Inlet Temperature', 'DISTRICT HEATING 1')
  extract_timeseries_into_matrix(sqlFile, rows, 'District
     Cooling Inlet Temperature', 'DISTRICT COOLING 1')
  extract_timeseries_into_matrix(sqlFile, rows, 'District
     Heating Outlet Temperature', 'DISTRICT HEATING 1')
  extract_timeseries_into_matrix(sqlFile, rows, 'District
     Cooling Outlet Temperature', 'DISTRICT COOLING 1')
  extract_timeseries_into_matrix(sqlFile, rows, 'District
     Heating Mass Flow Rate', 'DISTRICT HEATING 1')
  extract_timeseries_into_matrix(sqlFile, rows, 'District
     Cooling Mass Flow Rate', 'DISTRICT COOLING 1')
  extract_timeseries_into_matrix(sqlFile, rows, 'District
     Heating Hot Water Energy', 'DISTRICT HEATING 1')
  extract_timeseries_into_matrix(sqlFile, rows, 'District
     Cooling Chilled Water Energy', 'DISTRICT COOLING 1')

  # Figure out how to add this variable, probably by zone:
  # "Output:Variable,*,Zone Predicted Sensible Load to Setpoint
     Heat Transfer Rate,hourly,timestep;").get
```

```
# sum up a couple of the columns and create a new column
var_1 = nil
var_2 = nil
rows.each_with_index do |row, index|
  if index == 0
    runner.registerInfo(row.join(','))
    # Get the index of the columns to add
    var_1 = row.index('HeatingElectricity')
    var_2 = row.index('HeatingGas')

    if var_1 && var_2
      rows[index] << 'HeatingTotal'
      next
    else
      break
    end
  end

  runner.registerInfo("Index #{index}, Value 1 #{row[var_1]},
     Value 2 #{row[var_2]}, Class #{row[var_1]}")
  runner.registerInfo("rows[index] class #{rows[index]}")
  rows[index] << row[var_1].to_f + row[var_2].to_f
end

# convert this to CSV object
File.open('./report_timeseries.csv', 'w') do |f|
  rows.each do |row|
    f << row.join(',') << "\n"
  end
end

# Find the total runtime for energyplus and save it into a
   registerValue
total_time = -999
location_of_file = ['../eplusout.end', './run/eplusout.end']
first_index = location_of_file.map {|f| File.exist?(f)}.index(
   true)
if first_index
  match = File.read(location_of_file[first_index]).to_s.match
     (/Elapsed.Time=(.*)hr(.*)min(.*)sec/)
  total_time = match[1].to_i * 3600 + match[2].to_i * 60 +
     match[3].to_f
end

runner.registerValue('energyplus_runtime', total_time, 'sec')

return true
```

```
  ensure
    sqlFile.close if sqlFile
  end
end

# register the measure to be used by the application
AmbientLoopReports.new.registerWithApplication
```

Appendix D

ROM Framework Information

The ROM Framework is a Python package that was developed for this project. The ROM Framework was designed to be flexible allowing for new datasets and algorithms to be easily implemented and tested. The ROM Framework requires data in CSV format and a supporting JavaScript Object Notation (JSON) file that describes the covariates, response variables, and model parameters. The documentation of the ROM Framework can be found on Python's read the docs pages `https://reduced-order-modeling-framework.readthedocs.io/en/latest/`. The source code for the ROM Framework can be found on GitHub `https://github.com/nllong/ROM-Framework`.