ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MÁSTER EN INGENIERIA INDUSTRIAL

THE IMPACT OF CLIMATE, GEOGRAPHICAL LOCATION, AND HUMAN BEHAVIOR ON USAGE PATTERNS OF PROGRAMMABLE THERMOSTATS

Author: Ignacio Pérez de Rojas

Director: José Ramón Vázquez Canteli

Madrid

June 2018

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Ignacio Pérez de Rojas

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: "El impacto del clima, posición y conducta humana en los patrones de uso de termostatos programables", que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que

pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

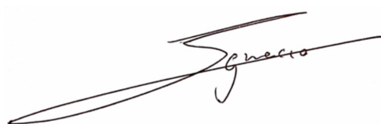
6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 28 de Junio de 2018

ACEPTA



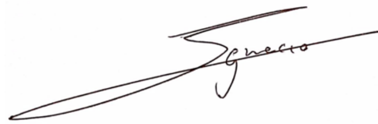
Fdo Ignacio Pérez de Rojas

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

"El impacto del clima, posición geográfica y conducta humana en los patrones de uso de termostatos programables"

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2017/2018 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Ignacio Pérez de Rojas Fecha: 28/06/2018



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: José Ramón Vázquez Canteli Fecha: 28/06/2018

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MÁSTER EN INGENIERIA INDUSTRIAL

THE IMPACT OF CLIMATE, GEOGRAPHICAL LOCATION, AND HUMAN BEHAVIOR ON USAGE PATTERNS OF PROGRAMMABLE THERMOSTATS

Author: Ignacio Pérez de Rojas

Director: José Ramón Vázquez Canteli

Madrid

June 2018

EL IMPACTO DEL CLIMA, POSICIÓN GEOGRÁFICA Y CONDUCTA HUMANA EN LOS PATRONES DE USO DE TERMOSTATOS PROGRAMABLES

Autor: Pérez de Rojas, Ignacio

Director: Vázquez Canteli, José Ramón

Entidad Colaboradora: The University of Texas at Austin

RESUMEN

Los sistemas de climatización representan aproximadamente el 21% del consumo total de energía en el sector residencial en EEUU. Pese a que su impacto a veces se pasa por alto, los termostatos residenciales tienen un gran efecto en el consumo total de energía de los hogares de EEUU.

Este documento analiza los datos de más de 6500 termostatos inteligentes para comprender cómo los ocupantes se comportan e interactúan con sus termostatos, y tiene tres objetivos principales: (1) proporcionar un análisis exploratorio de los datos para visualizar los diversos patrones de uso del termostato; (2) analizar las correlaciones lineales y no lineales entre los setpoints de temperatura en hogares, con variables de entrada como la temperatura exterior, la humedad o la ubicación geográfica; (3) clasificar a los usuarios de acuerdo con sus preferencias personales y localización para resaltar las diferencias entre los hogares acostumbrados a diferentes condiciones climáticas.

Introducción

En un mundo donde las técnicas de automatización e inteligencia artificial están liderando el camino para facilitar la experiencia del usuario y aumentar la eficiencia general de una amplia gama de industrias, la industria del termostato residencial aún tiene dificultades para alcanzar los objetivos de ahorro de energía y confort que el termostato programable estaba destinado a lograr.

Los termostatos tienen un objetivo simple: mantener la temperatura de una casa en un valor deseado o "setpoint". Sin embargo, varios conceptos erróneos acerca de cómo funcionan los termostatos programables han llevado a los usuarios a operarlos de forma manual, obteniendo así un rendimiento subóptimo [1].

Al mismo tiempo, los termostatos programables están programados para seguir los horarios de ocupación promedio en lugar de modelos de ocupación más realistas y flexibles. Esto ha impulsado aún más la idea de que su uso por lo general no conduce a grandes ahorros de energía o una mejor experiencia de los ocupantes. De acuerdo con la EPA de EEUU [2] sobre este tema, “Estudios disponibles muestran ausencia de ahorro tras la instalación de termostatos programables (PT). Algunos estudios indican un ligero incremento en el consumo”.

En este contexto, los termostatos inteligentes, que aprenden de las preferencias personales de los ocupantes para predecir la temperatura que de otro modo tendrían que establecer, apuntan a resolver estos problemas. Al quitar al ocupante de la ecuación del control de su termostato, los termostatos inteligentes pueden proporcionar las mismas temperaturas preferidas a la vez que ahorran energía y mejoran la comodidad del ocupante.

Después de un análisis exploratorio de los datos disponibles, este proyecto pretende predecir los setpoints que un ocupante elegiría para su casa en función de sus preferencias personales y las variables externas actuales, como temperatura exterior, humedad, ocupación, etc. Este objetivo es útil al modelar el consumo total futuro de energía relacionada con la climatización de un edificio o un grupo de edificios cuando se desconoce la información sobre los setpoints del termostato. Un modelo de simulación que sea capaz de predecir cómo se comportará cada hogar durante un período de tiempo es mucho más preciso que uno que simplemente tenga en cuenta los horarios promedio.

Finalmente, este proyecto también tiene como objetivo clasificar a los usuarios de acuerdo con sus preferencias de setpoint para identificar si existen diferencias entre los comportamientos de los hogares acostumbrados a diferentes climas y posiblemente a diferentes estilos de vida. Este objetivo resulta útil cuando se trata de desarrollar un termostato inteligente que no solo aprenda de cada usuario, sino que también ofrezca un comportamiento de setpoint suficientemente adaptado (basado en la localización del hogar) desde el primer momento.

Metodología

1. Predicción del setpoint: Dada la información de la base de datos mencionada anteriormente, este proyecto analiza la predictibilidad de los setpoints de temperatura utilizando métodos no lineales con diferentes variables climáticas como entradas.

El primer modelo de predicción de setpoint desarrollado combina la regresión multinomial y técnicas de clustering. En lugar de pasar todos los hogares a través de la misma regresión multinomial, primero se agrupan en clústers según su localización para representar los diferentes climas a los que cada uno está acostumbrado. Este análisis de clustering permite que el modelo de regresión se refine para cada grupo, adaptando las predicciones de cada ocupante dado el comportamiento de los otros en su mismo grupo.

El método de regresión logística se caracteriza por el hecho de que predice una variable de salida booleana con una cierta probabilidad, dadas variables de entrada continuas o categóricas. La regresión multinomial es una generalización del método de logística ya que es capaz de predecir una variable de salida categórica (con un número finito de categorías), dadas las mismas variables de entrada.

El segundo modelo de predicción de setpoints desarrollado utiliza el mismo análisis de clustering pero se basa en una red neuronal LSTM en lugar de un análisis de regresión multinomial para hacer sus predicciones. Las redes neuronales LSTM son un tipo de redes neuronales recurrentes (RNN) que se utilizan ampliamente en modelos secuenciales, como el reconocimiento de voz o la traducción automática, debido a su capacidad para

"recordar" valores en diferentes intervalos de tiempo. La naturaleza secuencial de ambos setpoints (series de tiempo) fue la razón por la cual se eligió este tipo de red neuronal. El LSTM presenta la ventaja en comparación con el modelo de regresión multinomial de trabajar con variables de salida continuas.

2. Clasificación de viviendas según su comportamiento de setpoint: Se analizan los datos de la base de datos para probar o refutar la hipótesis de que personas acostumbradas a diferentes climas eligen diferentes setpoints del termostato para sus hogares.

La primera técnica utilizada para probar esta hipótesis es el análisis de componentes principales (PCA). Los datos se dividen en días y cada día se traza por separado. Para un solo día, se traza un gráfico con los 2 componentes principales (PC) que contienen la mayor varianza de los datos en cada eje (con el componente principal en el eje X y el segundo componente principal en el eje Y) en el que cada vivienda está coloreada con el clúster al que pertenece. El análisis de clustering sirve para distinguir visualmente cualquier diferencia en los patrones de uso. Si cada clúster tiene realmente un comportamiento diferente, la figura PCA debe separar cada clúster en una zona diferente.

Después de este análisis, se utiliza Factor Analysis para encontrar cualquier variable latente oculta que pueda explicar mejor los datos que el análisis PCA, ayudando así a probar o refutar la misma hipótesis.

Resultados

1. Predicción del setpoint: Después de entrenar y probar el modelo de regresión multinomial con los datos contenidos en la base de datos, la mejor predicción del modelo para cada período de tiempo fue predecir que no hubiera cambios cada hora. Aunque esta estrategia lleva a la solución de evitar predecir seleccionando la categoría más probable cada vez, el modelo no pudo encontrar ningún patrón con las variables de entrada lo suficientemente fuertes como para poder predecir confiablemente cuándo ocurren los cambios del setpoint. Se obtuvieron resultados muy similares cambiando el método de clustering, el número (y criterios) de las categorías de setpoint y el número de hogares utilizados como datos de entrenamiento.

Para rechazar la hipótesis de simplemente tener un modelo que funciona mal o uno que se ha programado de tal manera que nunca va a proporcionar datos significativos, se realizó un otro análisis en el que se introdujo una nueva variable de entrada: "Schedule". "Schedule" es una variable que se creó artificialmente para tener en cuenta los horarios o patrones de uso que algunos hogares podrían tener (y por lo tanto, que podrían ser buenos predictores del comportamiento del setpoint). Analiza el día anterior de información de cada setpoint para encontrar cualquier patrón que pueda ayudar al modelo de regresión a predecir cómo se comportará en el futuro. De esta forma, si un hogar tiene un horario diario de 24 horas establecido para cualquiera de los setpoints, esta variable puede identificarlo y predecirlo antes de que ocurra, comenzando en el segundo día.

Aunque esta variable tiene sus limitaciones (que se explican mejor más adelante en este documento), los resultados obtenidos con esta variable como variable de entrada fueron

lo suficientemente buenos como para rechazar la hipótesis de tener un modelo que funcionara mal. Esto implica que el modelo de regresión multinomial simplemente no es adecuado para este modelo y, por lo tanto, fue reemplazado por una red neuronal LSTM.

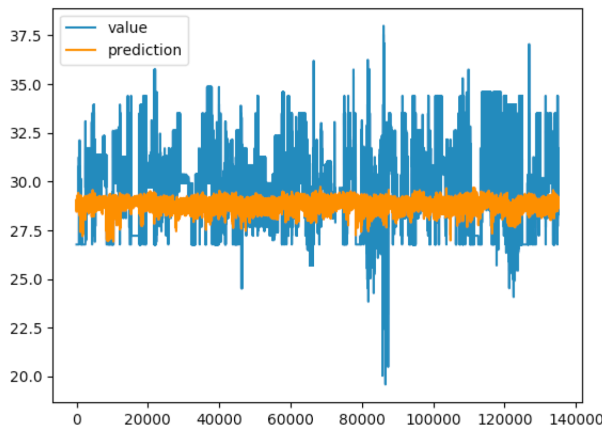


Figura 1. Predicción hora a hora del test data para el modelo LSTM con Tout, Tctrl y Humedad como entrada. El eje Y representa la temperatura [ºC] y el eje X horas [h]

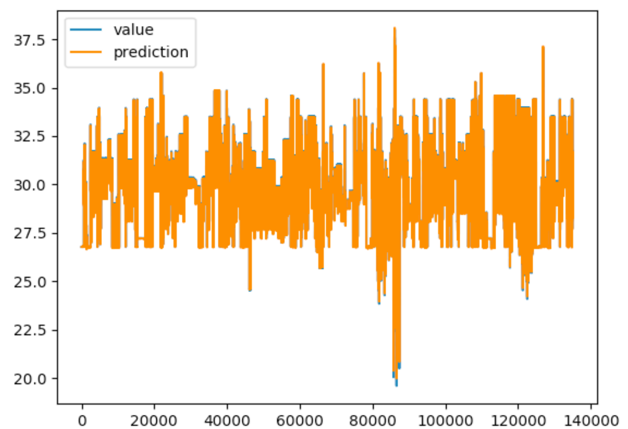


Figura 2. Predicción hora a hora del test data para el modelo LSTM con pasado de Tstp_cool como entrada. El eje Y representa la temperatura [ºC] y el eje X horas [h]

La Figura 1 muestra que el rendimiento de un clúster para el modelo LSTM dada la temperatura exterior, la temperatura interna y la humedad relativa como entradas es subóptimo. Para probar que el modelo funcionaba correctamente, se decidió ejecutar las mismas comprobaciones de calidad que con el modelo de regresión multinomial (tamaño de datos de capacitación, diferentes clústeres y ajuste de parámetros del modelo). Luego se decidió incorporar la información pasada de la variable de salida para distinguir si el problema estaba en el modelo o en los datos de entrada.

Para cada unidad de tiempo particular, t, la red neuronal LSTM predice cómo debería comportarse el setpoint para (t + 1) basándose no solo en las condiciones climáticas (temperatura exterior e interior, humedad relativa ...) sino también la propia información sobre cómo el setpoint se comportó en el pasado. La información pasada del resultado es útil para predecir sus valores futuros cuando sigue un patrón.

La Figura 2 muestra el rendimiento de este modelo, que es significativamente mejor que el mostrado en la Figura 1. Esto implica que en lugar de tener un modelo mal entrenado o que funciona mal, los datos de entrada de este modelo están demasiado débilmente correlacionados con la variable de salida para poder predecir correctamente.

2. Clasificación de viviendas según su comportamiento de setpoint: Se utilizó el formato de día debido a la inconsistencia del número de días que cada hogar tenía. La Figura 3 presenta los resultados del análisis del PCA del 25 de junio de 2016, ignorando todos los hogares que carecían la información de este día y trazando el componente principal 1 en el eje x y el componente principal 2 en el eje y. Si los hogares en diferentes clústers presentaran diferentes comportamientos de setpoints, los puntos de la Figura 3 se ordenarían por color (con la mayoría de los hogares que pertenecen al mismo grupo agrupados). Sin embargo, esta figura muestra que la diferencia de comportamientos de setpoint no está correlacionada con el clúster al que pertenecen.

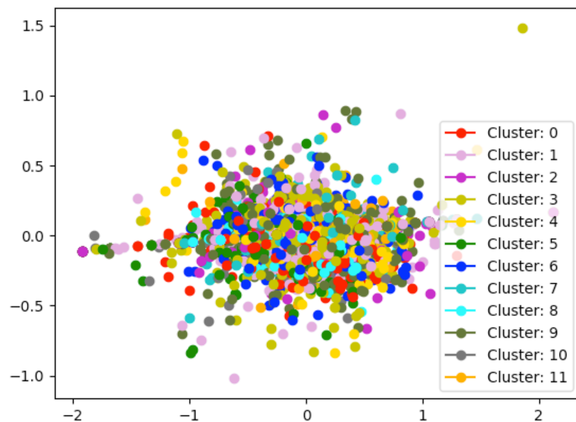


Figura 3. Resultado de PCA por clusters para un día. Cada punto representa una vivienda

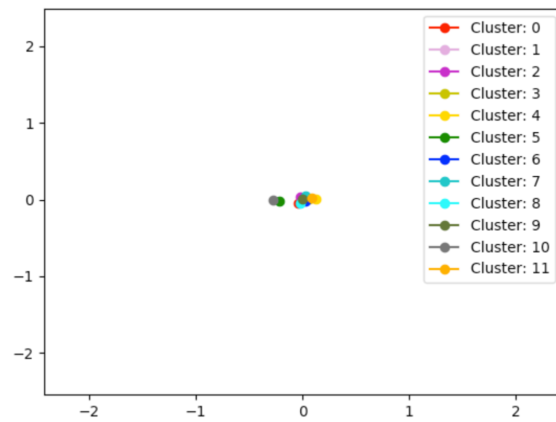


Figura 4 Resultado que muestra solo centros de clusters. Cada punto representa un el centro medio

La Figura 4 presenta el análisis en el cual los centros de cada grupo están lo suficientemente cerca como para mantener la hipótesis de un comportamiento similar para cada clúster. Se es consciente del sesgo que este análisis tiene debido a los diferentes climas que cada grupo trata en un día determinado. Si los resultados hubieran sido diferentes (con los puntos de datos más agrupados en grupos y, por lo tanto, los centros más separados), se habría realizado un análisis más preciso donde se habrían comparado días con una temperatura exterior más similar.

Conclusiones

En primer lugar, se ha combinado un modelo de regresión multinomial y una red neuronal con un análisis de agrupación para tratar de predecir cómo se comporta la variable de setpoint en función de la humedad, la hora actual y la temperatura exterior e interior. Estos modelos han fortalecido la hipótesis de que no existe ningún tipo de correlación entre los datos de entrada y salida, lo que hace que sea imposible para cualquier modelo predecir con exactitud cualquier variable de salida basada en el ruido.

En segundo lugar, este documento también ha presentado múltiples análisis para clasificar a los usuarios de acuerdo con sus preferencias personales y ubicación geográfica. El objetivo en este caso fue destacar cualquier diferencia entre los comportamientos de setpoint de los hogares acostumbrados a diferentes climas. Después de usar PCA y Factor Analysis para definir los diferentes tipos de comportamientos de setpoint, la heterogeneidad de los clusters de este análisis y los clusters obtenidos por ubicación no arrojaron resultados significativos para demostrar una diferencia significativa en los comportamientos de setpoint.

Referencias

[1] Diana Elliott Bryan Urban and Olga Sachs. Towards better modeling of residential thermostats. *Fifth National Conference of IBPSA-USA*, 2012.

[2] David Shiller. Programmable thermostat program proposal.

THE IMPACT OF CLIMATE, GEOGRAPHICAL LOCATION, AND HUMAN BEHAVIOR ON USAGE PATTERNS OF PROGRAMMABLE THERMOSTATS

Author: Pérez de Rojas, Ignacio

Director: Vázquez Canteli, José Ramón

Collaborating entity: The University of Texas at Austin

ABSTRACT

HVAC systems account for about 21% of the total energy consumption in the residential sector in the U.S. Although their impact is sometimes overlooked, residential thermostats have a great effect on the overall energy consumption of U.S. households.

This document analyzes the data from more than 6500 smart thermostats to better understand how occupants behave and interact with their thermostats, and has a three-folded objective: (1) providing an exploratory analysis of the data to visualize the various patterns of thermostat usage; (2) analyzing the linear and non-linear correlations among the temperature setpoints in many households with other variables such as outdoor temperature, relative humidity, or the geographical location; (3) classifying users according to their personal preferences and geographical location to highlight any differences between households accustomed to different weather conditions.

Introduction

In a world where automation and machine learning techniques are leading the way in easing the user experience and increasing the overall efficiency of a wide range of industries, the residential thermostat industry still struggles to achieve the energy saving and comfort goals that the programmable thermostat that was once set to achieve.

Thermostats have a simple objective: maintaining a house's temperature at a desired value or "setpoint". Yet, several misconceptions about how HVAC systems work have led users to operate their thermostats manually, and thus obtaining a suboptimal performance [1].

At the same time, programmable thermostats are programmed to follow average occupancy schedules instead of more realistic and flexible occupancy models. This has further pushed the idea that their use usually does not lead to much (if any) energy savings or an improved occupant's experience. According to US EPA [2] on this subject, "Available studies indicate no savings from programmable thermostat (PT) installation. Some studies indicate slight increased consumption."

In this context, smart thermostats, which learn from the occupant's personal preferences to predict the temperature that they would otherwise have to set, aim to solve these issues. By taking away from the occupant the role letting them decide setpoints and average schedules based on their best guess, smart thermostats are able to provide the same preferred temperatures while also saving energy and improving the occupant's comfort.

After an explanatory analysis of the data available, this project aims to predict the setpoints that an occupant would choose for their house based on their personal preferences and current external variables such as outside temperature, humidity, occupancy, etc. This objective is useful when modeling the future total HVAC-related energy consumption of a building or a group of buildings when the information about the thermostat's setpoints is not known. A simulation model that is able to predict how each household behaves throughout a period of time is much more accurate than one that simply takes into account average schedules.

Finally, this project also aims to classify users according to their setpoint preferences to identify if there are any differences between households' behaviors accustomed to different weathers and possibly different lifestyles. This objective proves useful when trying to develop a smart thermostat that not only learns from each user, but also comes with a tailored-enough setpoint behavior (based on the household's localization) right out of the box.

Methodology

1. Setpoint Predictability: Given the information from the before mentioned database, this project analyses the predictability of the temperature setpoints by using linear and non-linear methods and different weather variables as inputs.

The first setpoint prediction model developed combines multinomial regression and clustering techniques to work. Instead of running all the households through the same multinomial regression, they are first grouped into clusters according to their geographical location in order to account for the different climates each one is accustomed to. This clustering analysis allows the regression model to be refined for each particular cluster, tailoring the predictions of each occupant given how the other occupants in their cluster behave.

The logistic regression method is characterized by the fact that it predicts a boolean output variable with a certain probability, given continuous or categorical input variables. The multinomial regression is a generalization of the logistic method since it is capable of predicting a categorical output variable (with a finite number of categories), given the same input variables.

The second setpoint prediction model developed uses the same clustering analysis but relies on a LSTM neural network instead of a multinomial regression analysis to make its predictions. LSTM neural networks are a type of Recurrent Neural Networks (RNNs) that are widely used in sequential models such as voice recognition or machine translation due to their ability of "remembering" values over different time intervals. The sequential nature of both setpoints (time series) was the reason this type of neural network was chosen. The LSTM presents the advantage compared with the multinomial regression model of working with continuous output variables.

2. Household setpoint behavior classification: The data of the database is analyzed to prove or disprove the hypothesis that people accustomed to different weather choose different thermostat setpoints for their homes.

The first technique used to test this hypothesis is Principal Component Analysis (PCA). The data is split into days and each day is plotted separately. Once all the households are plotted in a chart with the 2 Principal Components (PCs) that contain the higher variance of the data in each axis (with the principal component in the X-axis and the second principal component in the Y-axis) each household is colored according to the cluster that they belong to. The clustering analysis serves to visually distinguish any differences in the setpoint patterns. If each does in fact have a different prominent behavior, the PCA chart should separate each cluster into a different area.

After this analysis, factor analysis techniques are used to find any hidden latent variables that might explain the data better than the PCA analysis to help prove or disprove the same hypothesis.

Results

1. Setpoint Predictability: After training and testing the multinomial regression model with the data contained in the database, the model's best prediction for each period of time was to predict no change each hour. Even though this strategy leads to the solution of avoiding predicting by selecting the most likely category each time, the model was unable to find any pattern with the input variables strong enough to be able to reliably predict when setpoint changes occur. Very similar results were obtained by changing the clustering method, the number (and criteria) of the setpoint categories and the number of households used as training data.

In order to avoid the hypothesis of simply having a malfunctioning model or one that has been programmed in such a way that is never going to provide meaningful data, a second analysis was done in which a new variable was introduced as input data: schedule.

"Schedule" is a variable that was artificially created to account for schedules or patterns of use that some households might have (and thus, could be good predictors of setpoint behavior). It looks at the past day of information of each setpoint to find any patterns that could help the regression model predict how it behaves in the future. This way, if a household has a 24h daily schedule set for either setpoint, this variable is able to identify it and predict it before it happens, starting on the second day.

Although this variable has its limitations (which are better explained later in the document), the results obtained with this variable as input data were strong enough to reject the malfunctioning model hypothesis. This implies that the multinomial regression model is simply not fitting for this model and so, it was replaced with a LSTM neural network.

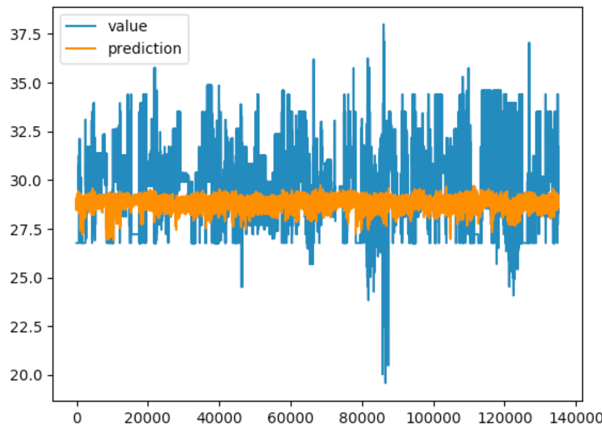


Figure 1. Hour-by-hour prediction of test data for LSTM model with T_{out}, T_{ctrl} and Humidity as input. Y-axis represents temperatura [ºC] and X-axis periods of time [h]

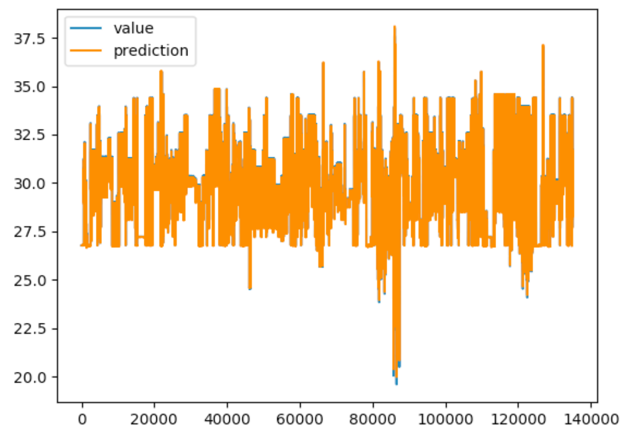


Figure 2. Hour-by-hour prediction of test data for LSTM model with past of T_{stp_cool} as input. Y-axis represents temperatura [ºC] and X-axis periods of time [h]

Figure 1 shows the performance of one cluster for the LSTM model given outside temperature, inside temperature and relative humidity as inputs, which is suboptimal. To prove that the model was working correctly, we decided to run through the same quality checks as with the multinomial regression model (training data size, different clusters and model parameters tuning). It was then decided to incorporate the past information of the output variable to distinguish if the issue was on the model or the input data.

For each particular unit of time, t, the LSTM neural network predicts how the setpoint should behave for (t+1) given not only weather conditions (outside and inside temperature, relative humidity…) but also the own information of how the setpoint itself behaved in the past. Past information of the output is helpful to predict its future values when it follows a pattern. This is often the case of programmable thermostats.

Figure 2 shows the performance of this model, which is significantly better to the one shown in Figure 1. This implies that rather than having a poorly trained or malfunctioning model, the input data of this model is simply too weakly correlated to the output variable that tries to predict.

2. Household setpoint behavior classification: Day-format was used for this analysis to account for the inconsistency of the number of days that each household had Figure 3 presents the results of the PCA analysis of June 25th, 2016, ignoring all the households that lacked the information of this day and plotting principal component 1 on the x-axis and principal component 2 on the y-axis.

If households in different clusters presented different setpoint behaviors, the data points of Figure 3 would be arranged by color (with most of the households that belong to the same cluster near each other). By contrast, this figure shows that the difference of setpoint behaviors are uncorrelated with the cluster they belong to.

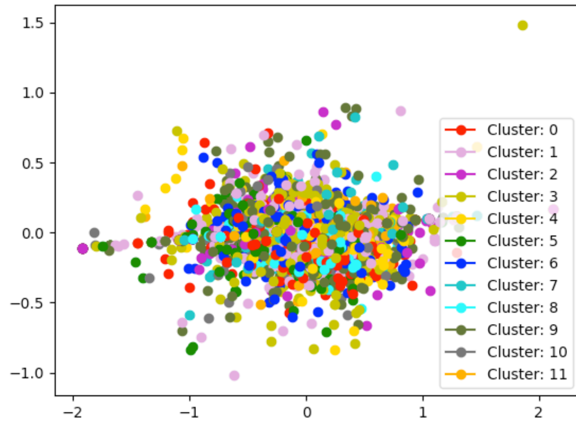


Figure 3. PCA result by clusters for one day. Each data point represents one household

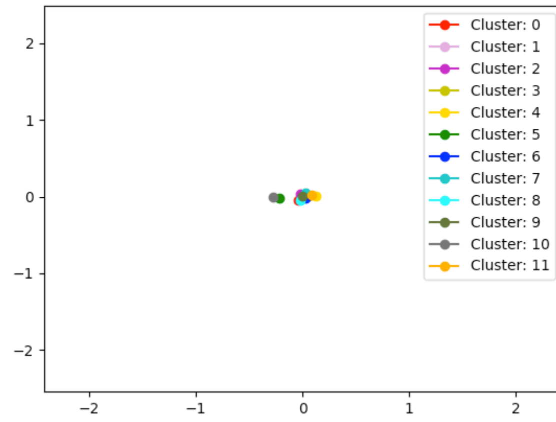


Figure 4 Result showing only clusters' centers. Each data point represents one cluster center

Figure 4 presents the analysis in which the centers of each cluster are close enough to maintain the hypothesis of equal behavior for each cluster. We are aware of the bias that this analysis has due to the fact the different climates that each cluster deals with for any given day. Should the results have been different (with the data points more clustered into groups and thus the centers more separated), a more accurate analysis would have been done where days with a more similar outdoor temperature would have been compared.

Conclusions

First, both a multinomial regression model and a neural network have been combined with a clustering analysis to try to predict how the setpoint variable behaves based on humidity, current time and outdoor and indoor temperature. These models have strengthened the hypothesis that there is no type of correlation between the input and output data, making it impossible for any model to accurately predict any output variable based on noise.

Second, this document has also presented multiple analyses to classify users according to their personal preferences and geographic location. The aim in this case was to highlight any differences between setpoint behaviors from households accustomed to different weathers. After using both PCA and Factor Analysis to define the different types of setpoint behaviors, the heterogeneity of the clusters of this analysis and the clusters obtained by location have yielded no meaningful results to prove significant difference on the setpoint behaviors.

References

[1] Diana Elliott Bryan Urban and Olga Sachs. Towards better modeling of residential thermostats. *Fifth National Conference of IBPSA-USA*, 2012.

[2] David Shiller. Programmable thermostat program proposal.

Table of Contents

1. Introduction **27**

2. Data Exploration **31**

2.1. Data Exploration 31

2.2. Code 36

3. Methodology **41**

3.1. Predictability of setpoint temperatures 41

3.1.1. Clustering 41

3.1.2. First Setpoint Prediction Model 42

3.1.3. Second Setpoint Prediction Model 44

3.1.4. Code 45

3.2. Household Behavior Setpoint Classification 49

3.2.1. Code 50

4. Results **53**

4.1. Predictability of setpoint temperatures 53

4.1.1. First Setpoint Prediction Model 53

4.1.2. Second Setpoint Prediction Model 56

4.2. Household Setpoint Behavior Classification 61

5. Conclusion **67**

6. Bibliography **69**

7. Appendices **73**

A. Data Exploration 73

B. Multinomial Regression 79

C. LSTM Neural Network 85

D. LSTM Neural Network with Setpoint variable as input 94

E. Household Setpoint Behavior Analysis 100

F. Library 107

a) Clean_nan.py 107

b) Cuantify_change.py 107

c) DB_generator.py 108

d) Detect_change.py 109

e) Fill_one_whole_household.py 110

f) Fill_one_whole_household_v2.py 112

g) Geodata_collection_dum.py 115

h) Get_data.py 116
i) Number_clusters_optimization.py 117
j) Plot_one_month.py 118
k) Plot_one_month_v2.py 122
l) Plot_one_year.py 125
m) Supervised_format.py 129
n) Transform_data.py 130
o) Transform_data_inv.py 130
p) Transform_data_inv_v2.py 131
q) Transform_data_inv_1D.py 131
r) Transform_data_multinom.py 132
s) Transform_pre_clean.py 133
t) Transform_post_clean.py 133

List of Figures

Figure 1. Hour-by-hour prediction of test data for LSTM model with T_{out}, T_{ctrl} and Humidity as input. Y-axis represents temperatura [ºC] and X-axis periods of time [h]. 18

Figure 2. Hour-by-hour prediction of test data for LSTM model with past of T_{stp_cool} as input. Y-axis represents temperatura [ºC] and X-axis periods of time [h] 18

Figure 3. PCA result by clusters for one day. Each data point represents one household 19

Figure 4 Result showing only clusters' centers. Each data point represents one cluster center 19

Figure 5. 30-day data visualization for a sample household.. 32

Figure 6. Boxplot of T_{out} by cluster 33

Figure 7. Boxplot of the temperature setpoint by cluster... 34

Figure 8. Boxplot of T_{stp_heat} by cluster 34

Figure 9. T_{stp_cool} and T_{out} chart 35

Figure 10. T_{stp_heat} and T_{out} chart 35

Figure 11. Data transformation for clustering. Each row in the original format (left matrix) represents data from 1 hour while each row on the new format (right matrix) represents 1 day.. 42

Figure 12. Setpoint predictor structure which combines clustering and multinomial regression model 43

Figure 13. Data clustering by household localization... 43

Figure 14. Recurrent feature of RNNs networks representation................................... 44

Figure 15. LSTM neural network 45

Figure 16. Cost function error of training and test data for LSTM model with T_{out}, T_{ctrl} and Humidity as inputs 57

Figure 17. Hour-by-hour prediction of test data for LSTM model with T_{out}, T_{ctrl} and Humidity as input.. 57

Figure 18. Cost function error of training and test data for LSTM model with past of T_{stp_cool} as input.. 59

Figure 19. Hour-by-hour prediction of test data for LSTM model with past of T_{stp_cool} as input 59

Figure 20. Close-up of prediction of test data for LSTM model with past of T_{stp_cool} as input 60
Figure 21. PCA result by clusters for one day 61
Figure 22. PCA result showing only clusters' centers 62
Figure 23. PCA results for one day clustered 63
Figure 24. PCA result from centers for 2016 65
Figure 25. Factor analysis result by clusters for one day 65
Figure 26. Factor analysis result of only clusters' centers 66
Figure 27. Factor analysis result from centers for 2016 66

List of Tables

Table 1. Correlation matrix of cooling setpoint temperature, humidity and outdoor and indoor temperature 32
Table 2. Kruskal-Wallis test for Tout, Tstp_cool and Tstp_heat........ 33
Table 3. Confusion matrix results for cluster 0 of multinomial regression model with original input variables 53
Table 4. Classification matrix for cluster 0 of multinomial regression model with original input variables 54
Table 5. Confusion matrix results for cluster 0 of multinomial regression model with original input variables and variable Schedule 55
Table 6. Classification matrix for cluster 0 of multinomial regression model with original input variables and variable schedule........ 56
Table 7. Clustering comparison 64

[This page was intentionally left blank]

1. Introduction

Recent technological innovations have increased the complexity and the amount of features that residential thermostats have, most of them in hopes of providing the final user with useful data to make better decisions and ultimately save energy. Yet, multiple misconceptions about how HVAC systems work have led users to operate their thermostats manually and sometimes erratically, obtaining a suboptimal performance.

These misconceptions include:

- "Thermostat is simply an on/off switch
- Thermostat is a dimmer switch for heat (valve theory)
- Turning down the thermostat does not reduce energy consumption (or not substantially)
- Boiler thermostat is used to change the temperature in the room (as if it is a room thermostat)
- People are afraid of using PTs (unknown terrible consequences)" (Urban et al. 2012)

The difficulty or ignorance of how to properly use PTs is one of the main factors that refrain users from leveraging the technological abilities of these devices. As the UK Government found in one of their studies:

"However, a major residual problem was controlling the central-heating system. A third of all respondents over 60 reported difficulty with programmers, with a majority of these saying they were too complicated; "I don't understand it," "I'm not very technical – unsure what to do." There were three types of response; first leaving the system as originally set, "I never touch the controls;" second, asking friends, family members or neighbors to adjust the setting; third, resorting to manual settings, "My husband switches it on when he gets up" However, in [all] these cases, such coping strategies were evidently not successful in securing warm homes." (Boait et al. 2010)

At the same time, the fact that programmable thermostats, which are programmed to follow average occupancy schedules instead of more realistic and flexible occupancy

models has further pushed the idea that their use usually does not lead to much (if any) energy savings or an improved occupant's experience. According to US EPA on this subject, "Available studies indicate no savings from programmable thermostat (PT) installation. Some studies indicate slight increased consumption."

In this context, programmable thermostats are gradually being replaced by a new generation of thermostats. The so-called smart thermostats, which learn from the occupant's personal setpoint preferences and occupancy behavior to predict the temperature that they would otherwise have to set. The intrinsic advantage of these thermostats is that they no longer require the user to manually control them indefinitely. Smart thermostats take control after a short period of adaptation and don't require to be readjusted by the owner afterwards. By taking away the human role from this equation, smart thermostats do not only increase their owners' overall comfortability and quality of life, but also help to achieve the energy-saving goals that the PTs never fully-achieved.

Finally, smart thermostats also present the advantage of being more predictable than the before mentioned human behavior. Thermostat predictability is especially useful when predicting a building's HVAC-related energy consumption, which is one of the objectives of this document. Should a simulation model try to estimate the total heating and cooling energy consumption of a building, it will need to be able to reliably predict how the thermostat's setpoints will behave or otherwise it will be forced to use an average scheduled behavior that might differ greatly from the real one.

Using the information from an already existing real-world database, an initial multinomial regression model will be developed to predict the setpoint behavior of every occupant in the dataset, using only the exterior and interior temperature, humidity and time of day as inputs, since those variables are the ones that the simulation software (which scope is out of this document) possesses about a number of buildings. Once this is done, a clustering analysis of each household will define different clusters based on how different types of occupants decide their thermostat setpoints. This classification will allow the initial model to be refined for each particular cluster, tailoring the predictions of each occupant given their preferences and how the other occupants in their cluster behave. Finally, a neural

network will be developed to replace the multinomial model in order to try to improve the predictions of each setpoint.

Therefore, we train data-driven regression models of the thermostats, which have already been clustered, using recurrent neural networks to predict the behavior of cooling and heating setpoints based on variables such as inside and outside temperature, relative humidity and current time of the day. The objective is to build regression models capable of generalizing among the different ways the thermostats of different households operate.

Regarding the other objective cited in the abstract, households will be characterized and analyzed combining clustering and PCA analysis techniques according to their setpoint preferences and geographic location. The objective is to identify if there are any differences between households' behaviors accustomed to different weathers and possibly different lifestyles. This objective will prove useful when taking the previous setpoint prediction model to the next level. That is, trying to develop a smart thermostat that not only learns from each user, but also comes with a tailored-enough setpoint behavior (based on the household's localization) right out of the box. This way, the learning curve and initial adaptation period of the smart thermostat will be much shorter, being especially useful for people who refuse to or just ignore how to use their thermostats in the first place. Another use for this objective would be for commercial buildings that are visited on a daily basis by the public: Should they respond equally regardless of their localization or should they take into account the different behaviors and preferences of the people of that particular area?

Woo Moon et al. (2010) in their study titled "ANN-based thermal control models for residential buildings" also aim to solve the problem of predicting thermostats' setpoints using neural networks. Having no real-world thermostat information, their study differs from this one in the sense that they test their network on an experimental household setting especially prepared for such study. This document has the limitation of having less information from each household (since the data collected from the smart thermostats used in this study is limited) but will be able, if successful, to train a much realistic neural network in order to implement it in the future.

Boait et al. (2009) tried a different approach in their study titled "A method for fully automatic operation of domestic heating". Using Bayesian statistics, an occupancy model was developed in order to automate a thermostat's settings. This way, users would no longer require to manually change the schedule of their thermostat, as it would be automatically done. This approach presents the limitation of having to operate between a couple of rigid scheduled setpoints, instead of the more flexible model that a neural network or any other could provide.

Peffer et al. (2011) conducted an extensive study about the adoption of smart thermostats across the U.S. and how most households still fail to use them correctly. No other documents related to how external variables impact the behavior of residential thermostats' setpoints were found. The lack of more specific studies about this issue motivates this document to analyze the data collected by ecobee's smart thermostats from more than 6500 households.

The contributions of this paper are three: (1) providing a further insight on this dataset and performing an exploratory analysis of the data to visualize the various patterns of thermostat usage; (2) an analysis the correlations among the temperature setpoints in many households with other variables such as outdoor temperature, relative humidity, or the geographical location. We do not only analyze the existence of linear correlations, but also study the viability of non-linear regression methods for predicting the temperature setpoints using the aforementioned variables; and (3) classifying users according to their personal preferences and geographical location to highlight any differences between households accustomed to different weather conditions.

2. Data Exploration

2.1. Data Exploration

The first step that was taken in order to develop the thermostat setpoint regression model was to understand which data was available to use.

The before mentioned database is the result of the data stored by 6739 different ecobee thermostats, currently stored as a single HDF5 file and containing (but not limited to) the following variables for each household:

- Time, with a resolution of 1 hour
- T_ctrl: Household inside temperature
- T_out: Outside temperature
- T_stp_heat: Heating temperature setpoint
- T_stp_cool: Cooling temperature setpoint
- Humidity
- Event and Schedule the thermostat might be operating in
- Remote sensors motion
- Remote sensors temperature
- Activation of heating or cooling system
- Thermostat temperature

Furthermore, this database also contains different metadata information for each household (house age, number of floors, localization…)

Due to the nature of the simulation software that would use the regression model to predict household setpoint behavior, only the current time, humidity and inside and outside temperature will be used in the model as input data.

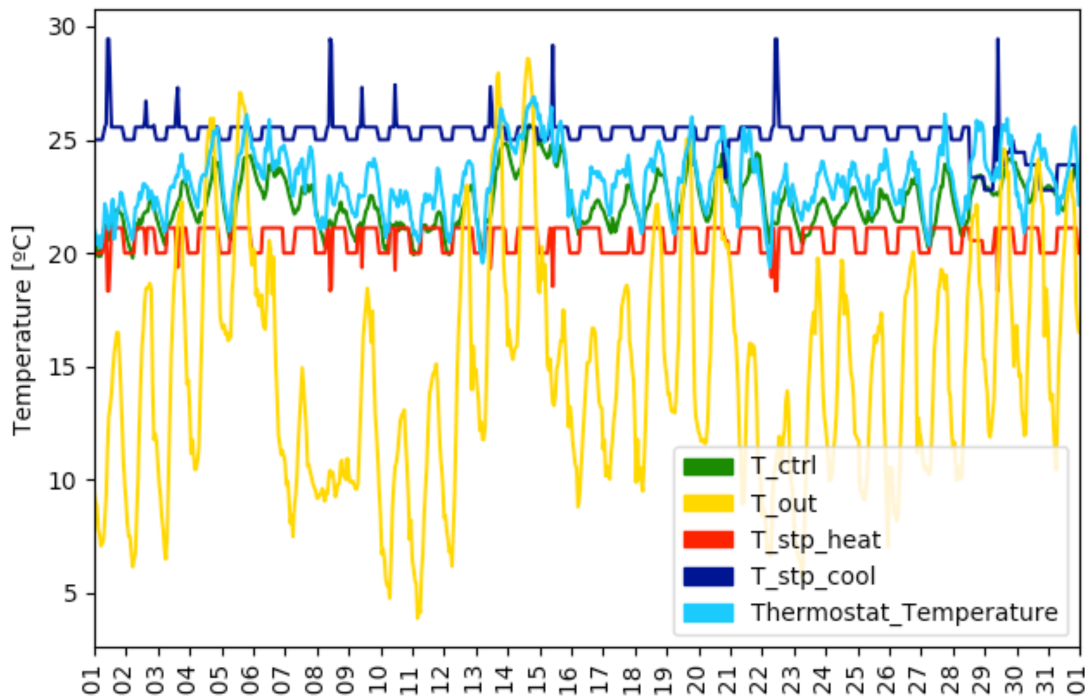


Figure 5. 30-day data visualization for a sample household

A correlation matrix was then built to get an idea of how correlated the setpoint variable was compared to the input variables. As shown in Table 1, the linear correlation is weak, which implies that there is not much of a linear relationship between the setpoint behavior and the rest of the input data. This is a good indicator that linear regression models wouldn't probably work best in this case.

	T_{out}	T_{ctrl}	Humidity	T_{stp_cool}
T_{out}	1	0.674	0.486	0.067
T_{ctrl}	0.674	1	0.238	0.198
Humidity	0.486	0.238	1	-0.001
T_{stp_cool}	0.067	0.198	-0.001	1

Table 1. Correlation matrix of cooling setpoint temperature, humidity and outdoor and indoor temperature

The objective of doing a preliminary analysis of the data is two-folded: First, it is a good way to know what kind of data is in the database and second, it helps to view any inconsistencies or outliers that the data might have.

Figures 2, 3, and 4 show a boxplot of every cluster for Tout, Tstp_cool and Tstp_heat, respectively. Table 2 contains the results of the Kruskal-Wall test which rejects the null

hypothesis of equal means, proving the relevance of dividing the original data into clusters.

At the same time, these figures illustrate that the majority of the households studied chose both their cooling and heating setpoints in accordance with ASHRAE guidelines, which are set between 19.4 ºC and 27.8 ºC. Nonetheless, there was a significant number of outliers, mainly due to the fact that many households still struggle to properly program their thermostats.

	H-statistic	**p-value**
T_{out}	4966563.998927	0.0
T_{stp_cool}	1011935.795565	0.0
T_{stp_heat}	520014.6961706	0.0

Table 2. Kruskal-Wallis test for T_{out}, T_{stp_cool} and T_{stp_heat}

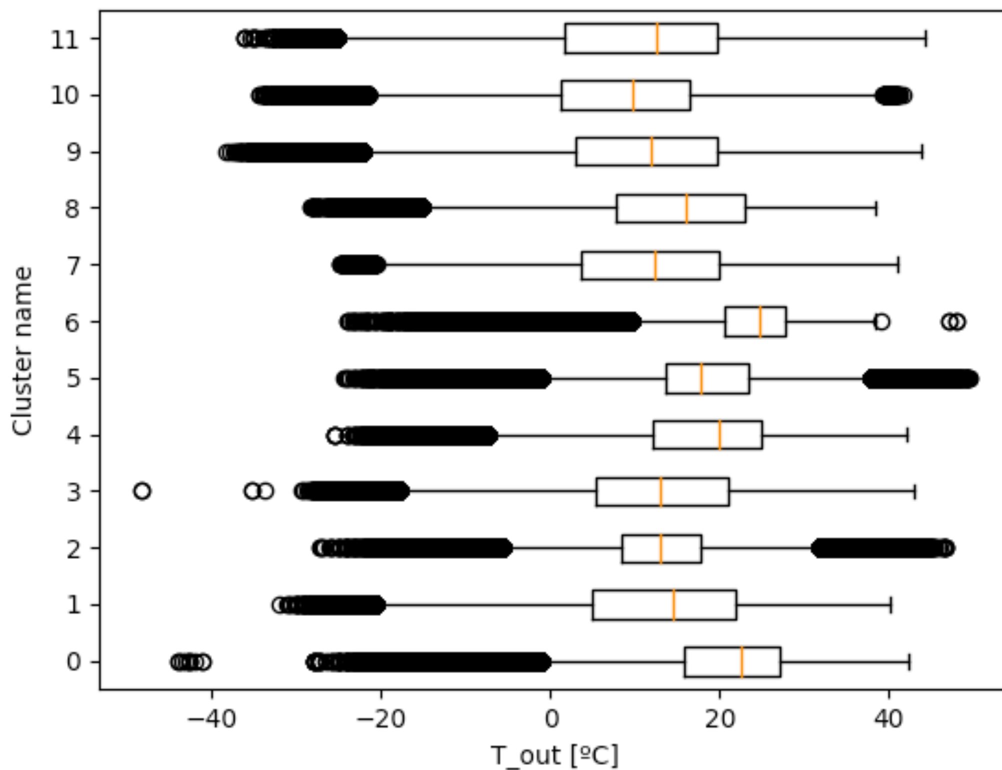


Figure 6. Boxplot of T_{out} by cluster

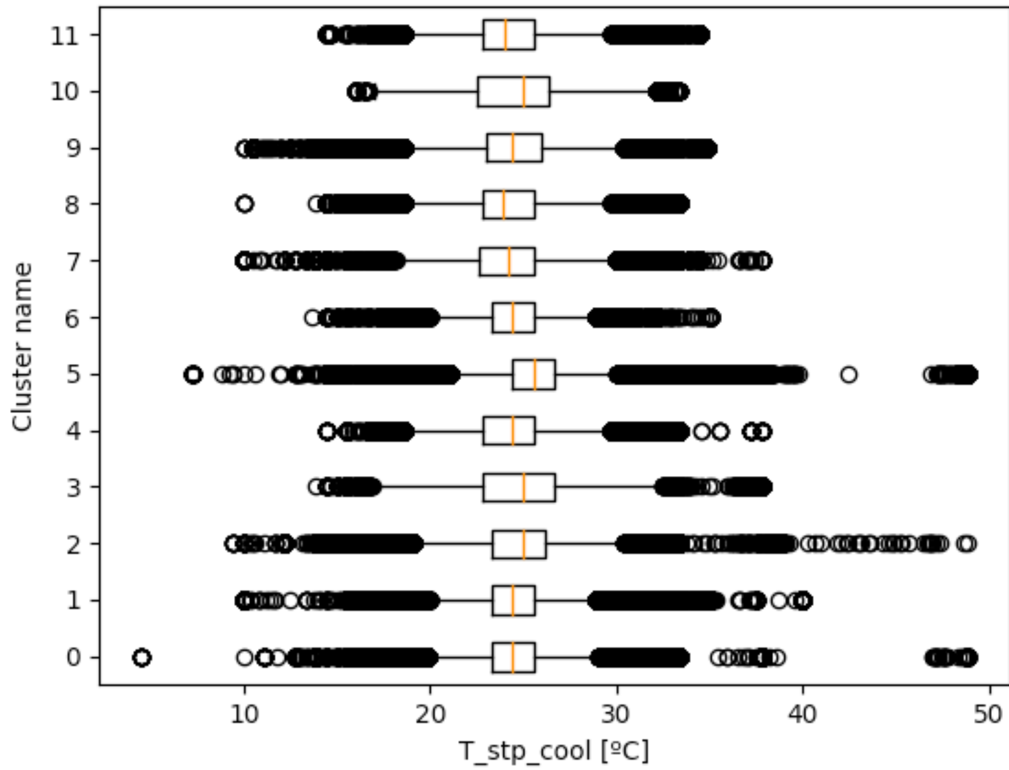


Figure 7. Boxplot of the temperature setpoint by cluster

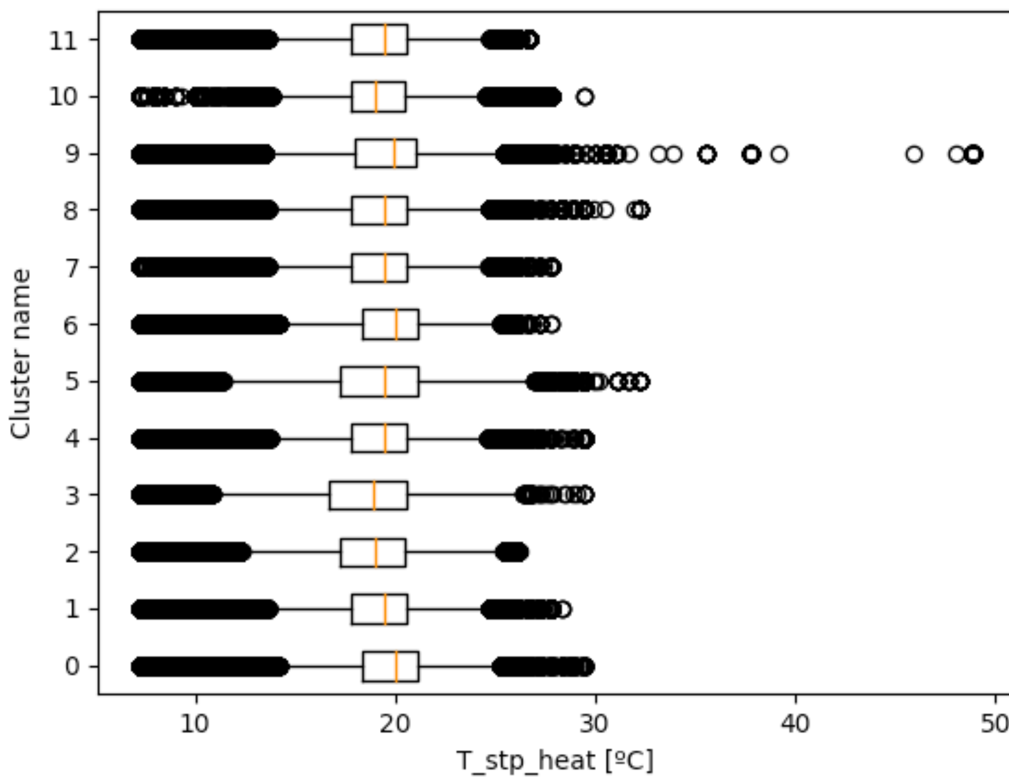


Figure 8. Boxplot of T_{stp_heat} by cluster

Figure 9 and Figure 10 show an apparent weak correlation between setpoints and T_{out}.

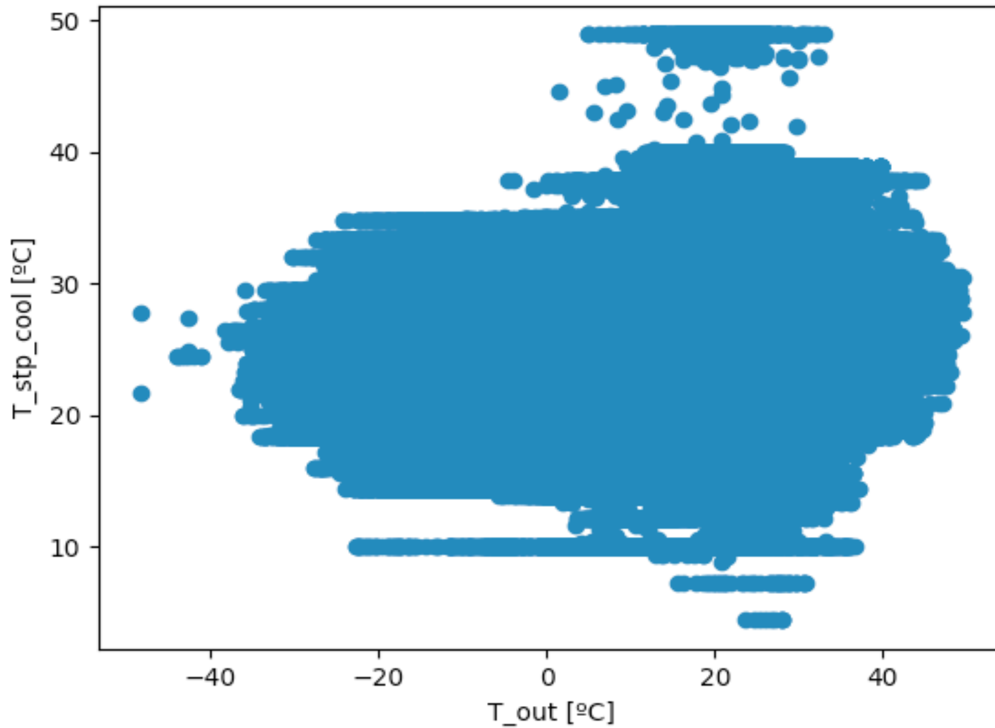


Figure 9. T_{stp_cool} and T_{out} chart

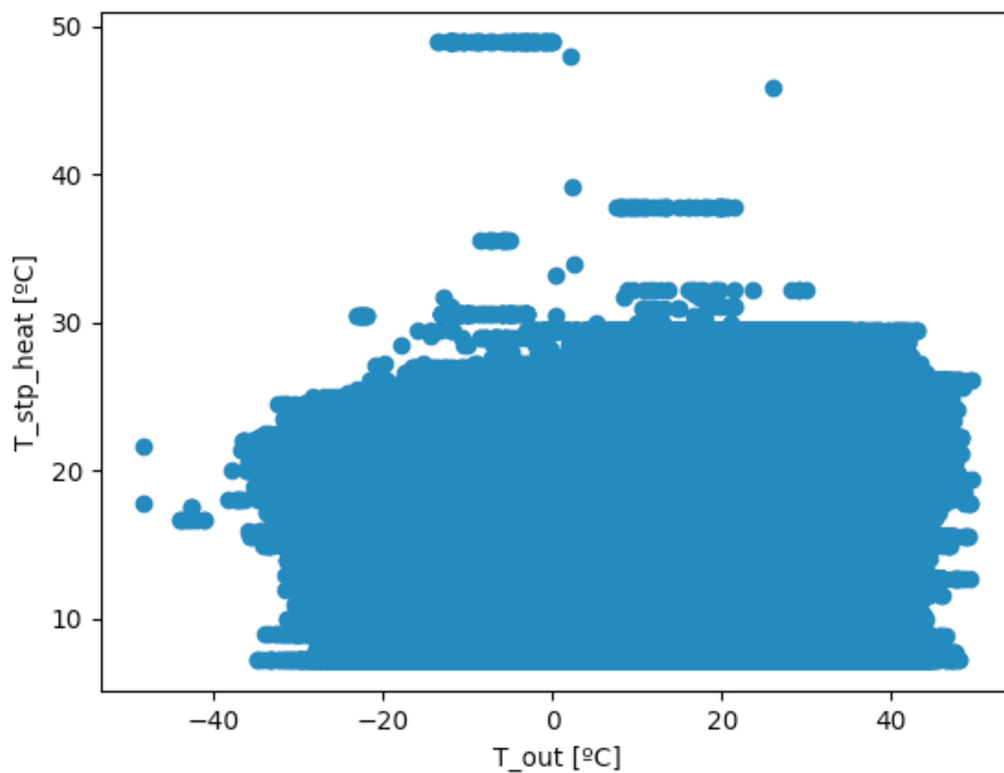


Figure 10. T_{stp_heat} and T_{out} chart

2.2. Code

Data acquisition from the database is the first step in the code. The below snip (labeled as "GET DATA") is used in every script developed for this project, so it will only be explained once for simplicity.

```
1.  ################################ GET DATA ################################
2.
3.  start = datetime.now()
4.  print("start: ",start)
5.  data = []
6.  data = get_data('npy', households, st_household, st_year, fin_year, st_month,
    fin_month)
7.  print(data.shape)
8.  print("Time elapsed: ", datetime.now()-start)
9.
10. ## Fahrenheit to Celsius
11. data[:,5] = copy.copy(F2C(data[:,5]))
12. data[:,6] = copy.copy(F2C(data[:,6]))
13. data[:,8] = copy.copy(F2C(data[:,8]))
14. data[:,9] = copy.copy(F2C(data[:,9]))
15.
16. print("Data acquisition completed: ",datetime.now(),"\n")
17.
18. ##########################################################################
```

First, the code calls the function *get_data* which was designed specifically for this project. This function queries the database month by month (since the data is stored that way) and then proceeds to clean all the days that contain at least one null value (NaN). This is done by changing the data to day-format (as shown in Figure 11), eliminating all the rows that contain at least one NaN value and then switching back to the original format.

Then, the values from the columns that correspond to temperatures are converted into Celsius since they are originally stored in Fahrenheit.

Once the data is cleaned and stored in a variable, the code proceeds to cluster the households according to their location in order to be able to then plot Figure 13. The below snip of code is responsible for this task.

First, the code reads the location (latitude and longitude) of each household from a text file. Since there are some households with unknown location or a location outside of North America, a list is created with the intention of saving the household IDs of all the households that have a valid location.

```
1.  ############################### CLUSTERING ###############################
2.
3.  ## Clustering method 2 (metadata)
4.  ## Read metadata
5.  with open('/Users/user/Documents/Project/Metadata_full_2.txt', 'r') as ff:
6.      household_latitude_full.append([float(i) for i in tuple(zip(*[line.split(";") for line in ff]))[4]])
7.      ff.seek(0)
8.      household_longitude_full.append([float(i) for i in tuple(zip(*[line.split(";") for line in ff]))[5]])
9.  household_latitude_full = np.array(household_latitude_full)
10. household_longitude_full = np.array(household_longitude_full)
11. household_latitude_full = household_latitude_full.reshape(-1,1)
12. household_longitude_full = household_longitude_full.reshape(-1,1)
13.
14. print("latitude, longitude full shape: ", household_latitude_full.shape, household_longitude_full.shape)
15.
16. ## Household Selection
17. for i in range(0,len(household_longitude_full)):
18.     if(household_longitude_full[i] > -130 and household_longitude_full[i] < -50 and household_latitude_full[i] > 20 and household_latitude_full[i] < 60):
19.         sel_buildings = np.append(sel_buildings, i)
20.
21. sel_buildings = np.array([int(i) for i in sel_buildings])
22. data = data[np.where(np.in1d(data[:,0].astype(int), sel_buildings[:].astype(int)))]
23. print("data shape: ", data.shape)
24. print("sel buildings shape: ", sel_buildings.shape)
25.
26. for i in sel_buildings:
27.     household_latitude = np.append(household_latitude, household_latitude_full[int(i)])
28.     household_longitude = np.append(household_longitude, household_longitude_full[int(i)])
29. household_latitude = household_latitude.reshape(-1,1)
30. household_longitude = household_longitude.reshape(-1,1)
31.
32. x_clus = np.hstack((household_latitude, household_longitude))
33. scaler_clus = MinMaxScaler(feature_range=(0,1))
34. x_clus = scaler_clus.fit_transform(x_clus)
35.
36. ## Clustering model
37. kmmodel = KMeans(n_clusters = clusters, random_state = 0).fit(x_clus)
38. labels = kmmodel.labels_
39. labels_box = np.hstack((sel_buildings.reshape(-1,1),labels.reshape(-1,1)))
40. print("Labels: ",labels.shape)
41.
42. ## Plot KMeans Inertia to select number of clusters
43. #number_clusters_optimization(x_clus)
44.
45. print("Clustering: ",datetime.now())
46.
47. ##########################################################################
```

This “household selection” list is then used to differentiate the households that will be clustered and those that will be ignored. The clustering model uses K-Means and the

number of clusters is selected from plotting at the model's inertia and finding where is the elbow.

Once the clustering analysis is done and there is one label per selected household, the code is ready to plot Figure 13. Using the library matplotlib, the household's longitude is given as the X-axis and the household's latitude as the Y-axis to get the scatter plot.

```
1.  ############################## PLOT CLUSTERS ##############################
2.  
3.  ## Plot
4.  fig, ax = plt.subplots()
5.  for i in range(0, len(labels)):
6.      cluster = labels[i]
7.      ax.scatter(household_longitude[i], household_latitude[i], color=LABEL_COLO
    R_MAP[cluster], label = 'household '+str(int(i)))
8.  
9.  ## Plot legend and colors
10. legend_elements = []
11. label_elements = []
12. for i in range(0,clusters):
13.     legend_elements = np.append(legend_elements, Line2D([0],[0],marker='o', co
    lor=LABEL_COLOR_MAP[i], label='Cluster: '+str(int(i))))
14.     label_elements = np.append(label_elements, 'Cluster: '+str(int(i)))
15. #ax.legend(legend_elements, label_elements)
16. 
17. plt.show()
18. 
19. ###########################################################################
```

The below code is responsible for plotting Figure 9 and Figure 10. While it is similar to the previous code (since both of them are scatter plots and both use the matplotlib library), this plots requires the data to be normalized. This is done using the *MinMaxScaler* which is included in the sklearn library.

```
1.  ################################ POINT PLOT ###############################
2.  
3.  ## Variable declaration
4.  clean_T_out = copy.copy(data[:,5])
5.  clean_T_stp = copy.copy(data[:,8])
6.  
7.  x_pp = copy.copy(clean_T_out).reshape(-1,1)
8.  y_pp = copy.copy(clean_T_stp).reshape(-1,1)
9.  
10. ## Data normalization
11. x_clus = np.hstack((x_pp, y_pp))
12. scaler_clus = MinMaxScaler(feature_range=(0,1))
13. x_clus = scaler_clus.fit_transform(x_clus)
14. 
15. ## Plot
16. fig = plt.figure()
17. ax = fig.add_subplot(1,1,1)
```

```
18.
19. ## Axis labels
20. ax.scatter(x_pp,y_pp)
21. ax.set_ylabel("T_stp_cool [ºC]")
22. ax.set_xlabel("T_ctrl [ºC]")
23. plt.show()
24.
25. ############################################################################
```

Finally, the boxplots in Figure 6, Figure 7 and Figure 8 are plotted by clusters. This means that the original label list (which contains 1 label per household) needs to be transform so that there is 1 label per day. Once this is accomplished, the data (whether it is the outside temperature, inside temperature or relative humidity) is also split by cluster so that the matplotlib library can plot the boxplots independently.

```
1.  ############################## BOX PLOT T_OUT ##############################
2.
3.  ## Variable to use in box plot
4.  variable_to_plot = 9
5.
6.  ## Labels column creation
7.  x_labels = []
8.  start_data = datetime.now()
9.  x_labels = transform_data_inv_v2(labels_box,data[:,:2],only_labels = True, by_
    days = days_flag)
10. x_labels = x_labels.reshape(-1,1)
11. print("Time elapsed -----> ",datetime.now() - start_data)
12. print("x_labels: ",x_labels.shape)
13.
14. ## Plot
15. fig = plt.figure()
16. ax = fig.add_subplot(1,1,1)
17.
18. ## Split data by cluster
19. for i in range(0,clusters):
20.     exec("T_out_%s = copy.copy(data[np.where(x_labels[:] == %s)[0],variable_to
    _plot])" %(i,i))
21.
22. ## Axis labels
23. string = ""
24. for x in range(0,clusters):
25.     if(x == clusters-1):
26.         string += "eval(\"T_out_" + str(x) + "\")"
27.     else:
28.         string += "eval(\"T_out_" + str(x) + "\"),"
29. eval("ax.boxplot([%s], vert = False, positions = np.arange(0,clusters,1))" %(s
    tring))
30. ax.set_ylabel("Cluster name")
31. ax.set_xlabel("T_stp_heat [ºC]")
32.
33. plt.show()
34.
35. ############################################################################
```

[This page was intentionally left blank]

3. Methodology

3.1. Predictability of setpoint temperatures

The predictability of the temperature setpoints is analyzed using linear and non-linear methods and different weather variables as inputs. The scope of this section is to analyze the viability of predicting the temperature setpoints using the data available.

Given that a thermostat's setpoint only changes so many times in a particular day, its behavior is most similar to a square function. Thus, the behavior of the setpoint will be modelled as a constant variable with a certain probability to change or remain the same for each period (t+1). This fact, plus the correlation matrix shown in Table 1, lead to avoid using any type of linear regression models all together and try using logistic or multinomial regression techniques. The logistic regression classification method is characterized by the fact that it predicts a boolean output variable with a certain probability, given continuous or categorical input variables. The multinomial regression is a generalization of the logistic method since it is capable of predicting a categorical output variable (with a finite number of categories), given the same input variables.

3.1.1. Clustering

As mentioned, the multinomial regression model will be combined with an initial clustering analysis of the different types of households, based on how different types of occupants decide their thermostat setpoints. This classification will allow the initial model to be refined for each particular cluster, specializing the predictions of each occupant given their preferences and how the other occupants in their cluster behave.

The chosen clustering technique is K-means and the variables that are used to cluster are T_ctrl, T_out and Humidity. Due to the inconsistency of the data available for each household (some households lack the information of one or more variables for one or more months), the data of each user was split into days, and each day was clustered independently, as shown in Figure 11. Each household then belongs to the cluster that represents the most of its days

3.1.2. First Setpoint Prediction Model

The first setpoint prediction model combines the before mentioned multinomial regression model and clustering analysis, as shown in Figure 12. It is important to remark that due to the fact that the prediction model provides a setpoint prediction for each hour (and not each day), the labels that are obtained after the clustering analysis, will have to go through a new data transformation that is strictly inverse to the process shown in Figure 11. Each label obtained after clustering (right matrix) corresponds to the cluster that the whole day belongs to, so each label will have to be repeated 24 times, once for each hour, in the original data format (left matrix).

The second clustering method that was develop was using each household location (as latitude and longitude). Although the location of each household was unknown, the metadata of the database provided the both the city and the state that each household belonged to, the geopy library allowed to obtain an approximate location of each household by providing the latitude and longitude of each city and using it as the household's. This second clustering method also involved transforming the original clustering's labels list, although this time each household had a different number of rows in the original format (left matrix) due to the fact that each household has a different number of recorded days in the database.

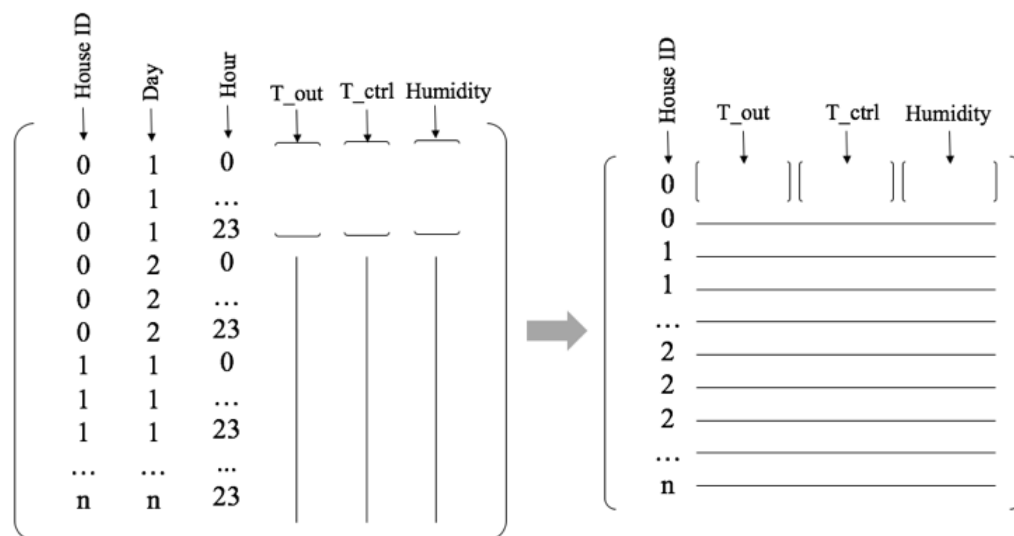


Figure 11. Data transformation for clustering. Each row in the original format (left matrix) represents data from 1 hour while each row on the new format (right matrix) represents 1 day

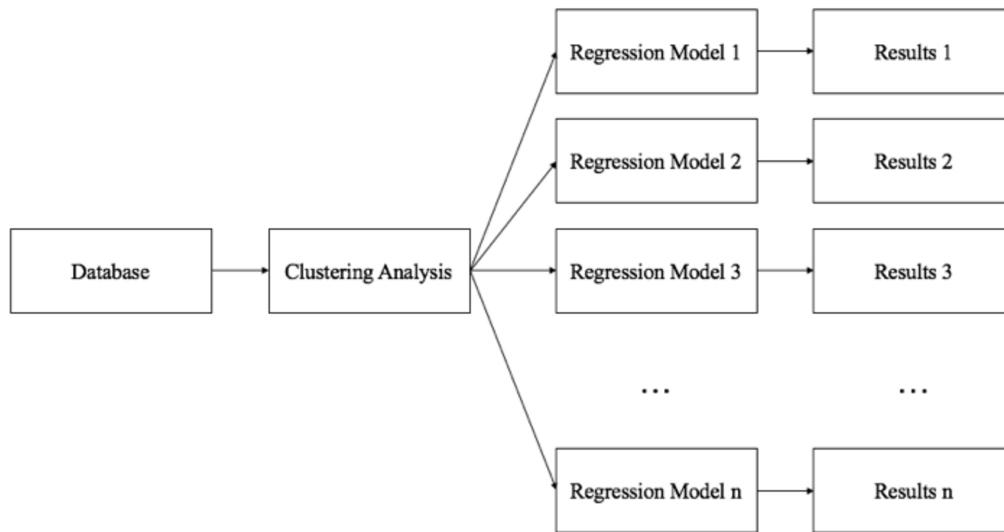


Figure 12. Setpoint predictor structure which combines clustering and multinomial regression model

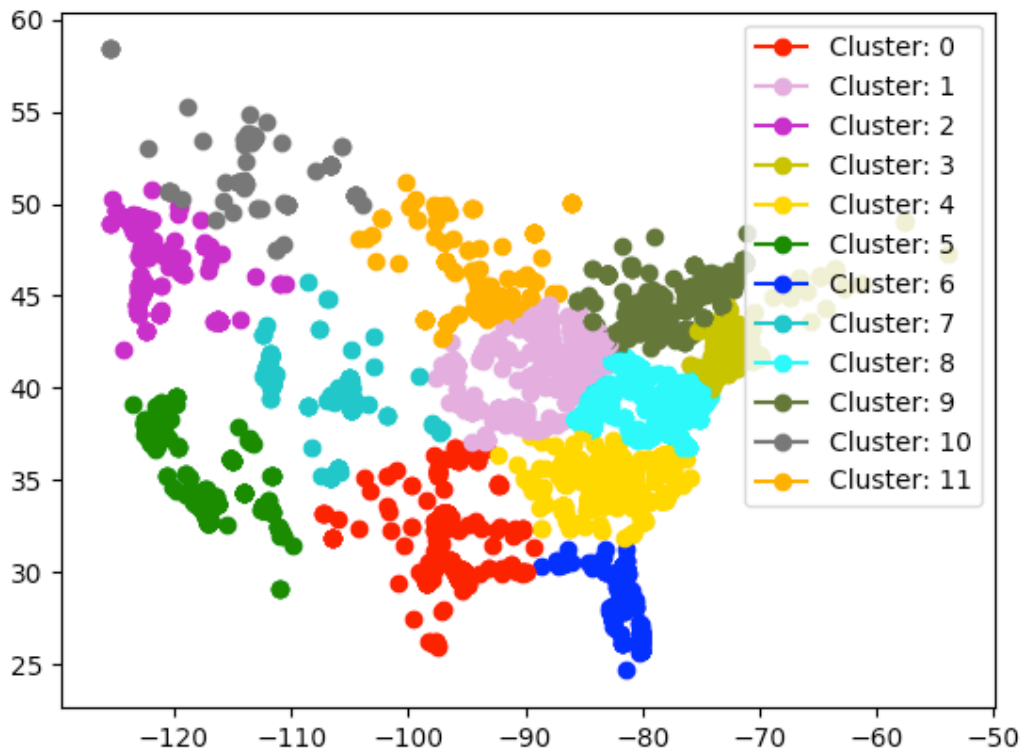


Figure 13. Data clustering by household localization

3.1.3. Second Setpoint Prediction Model

The second setpoint prediction model uses the same clustering techniques as the first model, but relies on a LSTM neural network to predict instead of a regression model.

Due to the sequential nature of the variable that this document is trying to predict (thermostat setpoint), the initial neural network type that was proposed to use was a simple Recurrent Neural Network or RNN (Figure 14). However, this type of neural network had the limitation of long-term persistence, which is important when predicting residential thermostat setpoints as most of them usually have a temporal pattern that can be ignored or forgotten if the neural network does not have a strong long-term memory.

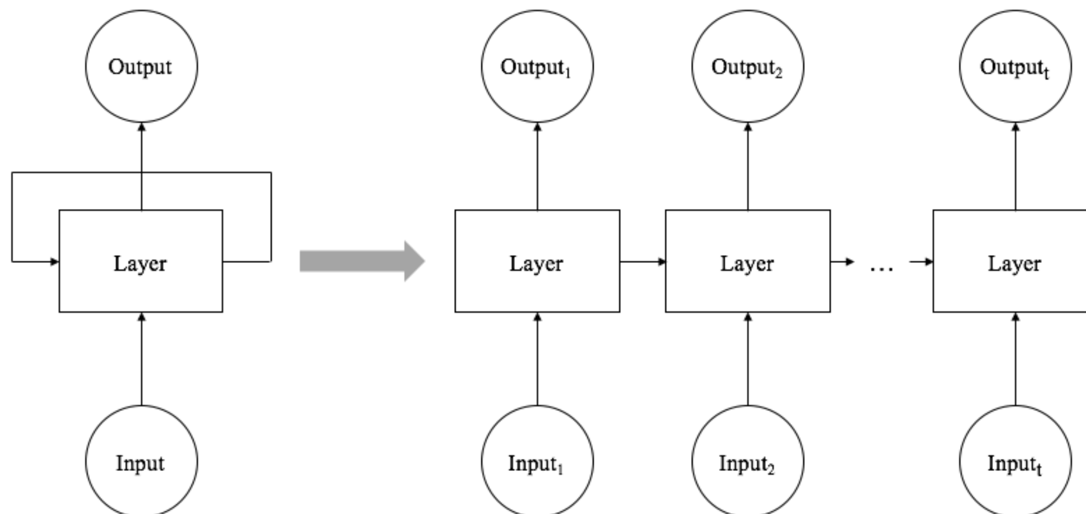


Figure 14. Recurrent feature of RNNs networks representation

In this particular LSTM network, the input layer contained the exact same data as the regression model (inside and outside temperature, time and humidity) and was trained using directly the temperature setpoint as output, as shown in Figure 15.

Due to their recurrence, LSTMs typically only use one hidden layer. Although Figure 15 only shows 6 neurons in the hidden layer, it is worth noting that the real neural network had 100 neurons.

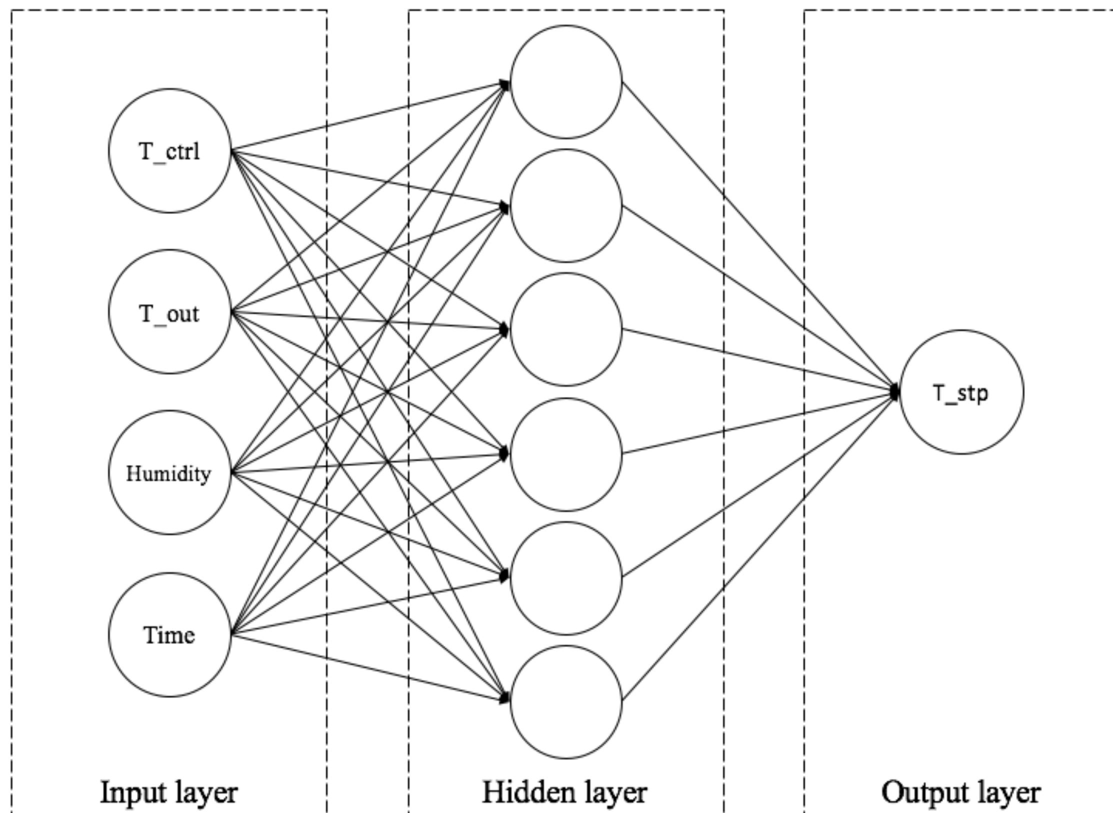


Figure 15. LSTM neural network

3.1.4. Code

3.1.4.1. First Setpoint Prediction Model

The Multinomial regression model uses the "get data" and "clustering" parts similar to Data Exploration (section 2.2.). Once the data is extracted from the database and clustered according to the location of each household, it then proceeds to split the data from the input/output variables (outside temperature, inside temperature, humidity, time and setpoint) into the previously defined clusters.

Then, variable x is defined as the input variable (by grouping all the individual input variables) and y as the output variable (setpoint). Each one is then separated into training and test sets, with the training set being 70% the size of the original dataset. The regression model is then ready to be trained and ultimately tested. The results from testing the model are printed on the console.

```
1.  ################################ GET DATA ################################
2.  [...]
3.  
4.  
5.  ############################### CLUSTERING ###############################
6.  [...]
7.  
8.  
9.  ######################### MULTINOMIAL REGRESSION #########################
10. 
11. ## Inverse transformation for clustering labels previously obtained
12. T_out_m = transform_data_multinom(labels, clean_T_out)
13. T_ctrl_m = transform_data_multinom(labels, clean_T_ctrl)
14. Humidity_m = transform_data_multinom(labels, clean_Humidity)
15. Hour_m = transform_data_multinom(labels, clean_Hour)
16. schedule_m = transform_data_multinom(labels, clean_schedule)
17. T_stp_cool_md = transform_data_multinom(labels, dummy_T_stp_cool)
18. 
19. 
20. for cluster in range(0,clusters):
21.     ## Split variables according to the cluster they belong to
22.     exec("T_out_m%s = copy.copy(T_out_m[np.where(T_out_m[:,0] == %s),1])" %(cl
    uster, cluster))
23.     exec("T_ctrl_m%s = copy.copy(T_ctrl_m[np.where(T_ctrl_m[:,0] == %s),1])" %
    (cluster, cluster))
24.     exec("Humidity_m%s = copy.copy(Humidity_m[np.where(Humidity_m[:,0] == %s),
    1])" %(cluster, cluster))
25.     exec("schedule_m%s = copy.copy(schedule_m[np.where(schedule_m[:,0] == %s),
    1])" %(cluster, cluster))
26.     exec("Hour_m%s = copy.copy(Hour_m[np.where(Hour_m[:,0] == %s),1])" %(clust
    er, cluster))
27.     exec("T_stp_cool_md%s = copy.copy(T_stp_cool_md[np.where(T_stp_cool_md[:,0
    ] == %s), 1])" %(cluster, cluster))
28. 
29.     ## Reshape (needed for regression)
30.     exec("T_out_m%s = T_out_m%s.reshape((len(T_out_m%s[0,:]),1))" %(cluster, c
    luster, cluster))
31.     exec("T_ctrl_m%s = T_ctrl_m%s.reshape((len(T_ctrl_m%s[0,:]),1))" %(cluster
    , cluster, cluster))
32.     exec("Humidity_m%s = Humidity_m%s.reshape((len(Humidity_m%s[0,:]),1))" %(c
    luster, cluster, cluster))
33.     exec("schedule_m%s = schedule_m%s.reshape((len(schedule_m%s[0,:]),1))" %(c
    luster, cluster, cluster))
34.     exec("Hour_m%s = Hour_m%s.reshape((len(Hour_m%s[0,:]),1))" %(cluster, clus
    ter, cluster))
35.     exec("T_stp_cool_md%s = T_stp_cool_md%s.reshape((len(T_stp_cool_md%s[0,:])
    ,1))" %(cluster, cluster, cluster))
36. 
37. 
38. for a in range(0, clusters):
39.     try:
40.         ## Format input and output variables and then split into train / test
    sets
41.         exec("x_%s = np.hstack((T_out_m%s, T_ctrl_m%s, Humidity_m%s, Hour_m%s,
     schedule_m%s))" %(a,a,a,a,a,a))
42.         exec("y_%s = T_stp_cool_md%s" %(a,a))
```

```
43.         exec("x_train_%s, x_test_%s, y_train_%s, y_test_%s = train_test_split(
    x_%s, np.ravel(y_%s), test_size=0.3, random_state=42)" %(a,a,a,a,a,a))
44.         exec("model_%s = LogisticRegression(multi_class='multinomial',solver =
    'newton-cg',max_iter=500000).fit(x_train_%s, y_train_%s)" %(a, a, a))
45.         exec("modelpred_%s = model_%s.predict(x_test_%s)" %(a,a,a))
46.         exec("modelprob_%s = model_%s.predict_proba(x_test_%s)" %(a,a,a))
47.
48.         ## Results
49.         print("------------------------ CLUSTER ", a, "-----------------------
    -")
50.         print(eval("metrics.confusion_matrix(y_test_%s,modelpred_%s)" %(a,a)))

51.         print(eval("metrics.classification_report(y_test_%s,modelpred_%s)" %(a
    ,a)))
52.     except:
53.         pass
54.
55. ############################################################################
```

3.1.4.2. Second Setpoint Prediction Model

The second prediction model uses once again the same “get data” and “clustering” parts as the first regression model, but it uses a LSTM neural network instead of regression techniques.

```
1.  ################################# GET DATA ################################
2.  [...]
3.
4.
5.  ################################ CLUSTERING ###############################
6.  [...]
7.
8.
9.  ############################## NEURAL NETWORK #############################
10.
11. ## Inverse data transformation for clustering labels
12. x_labels = []
13. x_labels = transform_data_inv(labels,x,only_labels = True, by_days = days_flag
    )
14. x_labels = x_labels.reshape(len(x_labels),1)
15. print("x_labels: ",x_labels.shape)
16.
17. ## Normalization
18. scaler_x = MinMaxScaler(feature_range=(0, 1))
19. scaler_y = MinMaxScaler(feature_range=(0, 1))
20. scaled_x = scaler_x.fit_transform(x)
21. scaled_y = scaler_y.fit_transform(y.reshape(-1,1))
22.
23. ## Supervised learning
24. reframed_x = supervised_format(scaled_x, look_back, 1)
25. reframed_y = supervised_format(scaled_y, look_back, 1)
26. reframed_y.drop(reframed_y.columns[0:-1],axis=1,inplace=True)
27. print(reframed_x.head())
28. print(reframed_y.head())
29.
30. ## Split by cluster
```

```
31. for i in range(0,clusters):
32.     exec("x_%s = copy.copy(x[np.where(x_labels[:] == %s)[0],:])" %(i,i))
33.     exec("x_%s = scaler_x.fit_transform(x_%s)" %(i,i))
34.     exec("y_%s = copy.copy(y[np.where(x_labels[:] == %s)[0]])" %(i,i))
35.     exec("y_%s = scaler_y.fit_transform(y_%s.reshape(-1,1))" %(i,i))
36.
37. ## Split into train / test sets
38. for i in range(0,clusters):
39.     n_train_hours = int(len(eval("x_%s[:,0]"%(i)))*0.7)
40.     exec("train_X_%s = x_%s[:n_train_hours,:]" %(i,i))
41.     exec("test_X_%s = x_%s[n_train_hours:,:]" %(i,i))
42.     exec("train_y_%s = y_%s[:n_train_hours]" %(i,i))
43.     exec("test_y_%s = y_%s[n_train_hours:]" %(i,i))
44.     print("Cluster ",i," ----
    >",eval("train_X_%s" %(i)).shape, eval("test_X_%s" %(i)).shape, eval("train_y_
    %s" %(i)).shape, eval("test_y_%s" %(i)).shape)
45.
46. ## Reshape to required 3D format [samples, timesteps, features]
47. for i in range(0,clusters):
48.     exec("train_X_%s = train_X_%s.reshape((train_X_%s.shape[0], 1, train_X_%s.
    shape[1]))" %(i,i,i,i))
49.     exec("test_X_%s = test_X_%s.reshape((test_X_%s.shape[0], 1, test_X_%s.shap
    e[1]))" %(i,i,i,i))
50.
51. ## LSTM model
52. for i in range(0,clusters):
53.     exec("model_%s = Sequential()" %(i))
54.     exec("model_%s.add(LSTM(neurons, input_shape=(train_X_%s.shape[1], train_X
    _%s.shape[2])))" %(i,i,i))
55.     exec("model_%s.add(Dense(1))" %(i))
56.     exec("model_%s.compile(loss='mean_squared_error', optimizer='adam')" %(i))

57.     exec("history_%s = model_%s.fit(train_X_%s, train_y_%s, epochs=epochs, bat
    ch_size=batch_size, validation_data=(test_X_%s, test_y_%s), verbose=2, shuffle
    =False)" %(i,i,i,i,i,i))
58.
59. ## Show model summary
60. #model.summary()
61.
62. ###########################################################################
```

Some of the steps followed are very similar to the ones taken with the regression model: splitting the data by clusters, splitting then the input and output variables into training and test sets or reshaping the format of the variables to be compliant with the requirements of the pre-defined functions.

3.2. Household Behavior Setpoint Classification

Once the before mentioned neural network is trained and is capable of functioning as a the "brain" of a smart thermostat, the database data will be analyzed once again, this time to prove or disprove the hypothesis that people accustomed to different weather will choose different thermostat setpoints for their homes.

The objective of proving this hypothesis is developing a smart thermostat that would be pre-programmed to work with a certain schedule and behavior right out of the box (with no interaction with the user at all) based on the household location and any other kind of metadata that might classify households according to the different setpoint preferences. If successful, this feature would prove useful in households with people that simply ignore their thermostat or do not know how to properly use it.

The first technique that will be used to test this hypothesis will be PCA. Given, once again, the inconsistency of information from each household, the data will be split into days and each day will be plotted separately. Once all the days are plotted in 2D 2PCs axis (with the principal component in the X-axis and the second principal component in the Y-axis), each day will be colored with the cluster that their household belongs to. In this case, the clustering analysis does not intervene in the PCA and only serves to distinguish graphically if there is any difference in septoint behavior from each cluster. If each does in fact have a different prominent behavior, the PCA chart should separate each cluster into a different area.

After this analysis, factor analysis will be used to find any hidden latent variables that might explain the data better. The same chart will be developed, using the same methodology and also splitting the data into days. This latent variable, might be "preference of warmer/colder temperatures", "day/night mode" or "temperature with or without occupant" would serve as an empirical proof that people that live in different weathers have different preferences when choosing what each one would consider a comfortable temperature, which would be really useful when developing the algorithm that would set the initial smart thermostat setpoint behavior when first bought.

3.2.1. Code

The household behavior setpoint classification also uses the “get data” and “clustering” parts to extract the data from the database and cluster it depending the location of each household.

Since Figure 22 requires the centers’ of each cluster to be calculated each day for a whole year, the code below uses a loop to do such calculations one day at-a-time. Figure 21 is a simplified version of this in which only one day is calculated.

```
1.  ################################ GET DATA ################################
2.  [...]
3.
4.
5.  ############################### CLUSTERING ###############################
6.  [...]
7.
8.
9.  ######################### PCA / Factor Analysis #########################
10.
11. ## PCA calculation (loop required to go day by day)
12. for year in range(st_year,fin_year+1):
13.     for month in range(st_month, fin_month+1):
14.         exec("days = month_days_%s[month]" %(year))
15.         for day in range(1,eval("days")+1):
16.             print(year, month," day ", day, " of ",eval("days"))
17.
18.             ## Get today's data
19.             data_sel = []
20.             data_sel = data[np.where(data[:,1] == year)]
21.             data_sel = data_sel[np.where(data_sel[:,2] == month)]
22.             data_sel = data_sel[np.where(data_sel[:,3] == day)]
23.             sel_buildings = np.unique(data_sel[:,0])
24.             print("sel buildings: ",sel_buildings.shape)
25.
26.             ## Get latitude and longitude of buildings with data from today
27.             day_latitude = []
28.             day_longitude = []
29.             for i in sel_buildings:
30.                 day_latitude = np.append(day_latitude, household_latitude_full
    [int(i)])
31.                 day_longitude = np.append(day_longitude, household_longitude_f
    ull[int(i)])
32.             day_latitude = day_latitude.reshape(-1,1)
33.             day_longitude = day_longitude.reshape(-1,1)
34.
35.             ## Format model input
36.             x = np.hstack((day_latitude, day_longitude))
37.             x = scaler_clus.transform(x)
38.             labels = kmmodel.predict(x)
39.
40.             ############################### PCA ANALYSIS #####################
    ########
```

```
41.
42.             pca_flag = True
43.             T_stp_cool_clus = []
44.             T_stp_cool_clus = transform_data(np.hstack((data_sel[:,4].reshape(
    -1,1) , data_sel[:,8].reshape(-1,1))))
45.             x = copy.copy(T_stp_cool_clus[:,:])
46.
47.             scaler_x = MinMaxScaler(feature_range=(0,1))
48.             x = scaler_x.fit_transform(x)
49.             model_pca = PCA(n_components = 2, copy = True)
50.             x_pca = model_pca.fit_transform(x)
51.
52.             ## Centroids
53.             for i in range(0,clusters):
54.                 exec("centroid_pca_data%s = x_pca[np.in1d(labels,i)]" %(i))
55.                 exec("centroid_pca_%s = centroidnp(centroid_pca_data%s)" %(i,i
    ))
56.                 exec("full_centr_pca_%s = np.append(full_centr_pca_%s, centroi
    d_pca_%s)" %(i,i,i))
57.             ##################################################################
    ########
58.
59. ##########################################################################
```

First, the loop extracts from the data the information that is relevant to the day that corresponds in each iteration. Then, it does the same thing for the latitude and longitude of the households that have valid information for such day.

Once this information is known, the PCA model is fitted and the centers of each cluster are calculated for that particular day. In order to finally get the center of all the daily centers for each cluster, all the centers obtained in each iteration are stored in one list. These centers' of centers are calculated only once the loop has finished all of its iterations.

[This page was intentionally left blank]

4. Results

4.1. Predictability of setpoint temperatures

4.1.1. First Setpoint Prediction Model

Following Figure 12 schematic design, the first thing that had to be done was to separate the initial data into clusters to analyze each cluster independently. In this case, the clustering analysis is based on the household's location, as shown in Figure 13. Once this is done, the output variable (which is in this case, either T_stp_cool or T_stp_heat) needs to be discretized into a set of categories for the multinomial regression model to work.

For this task, the variable T_stp_cool has been compared each period of time t to its predecessor (t-1). Let:

$$\Delta T = T_{stp_cool}(t) - T_{stp_cool}\,(t-1) \tag{1}$$

$$I_1 = \Delta T \in (-\infty\,, -4.5] \tag{2}$$

$$I_2 = \Delta T \in (-4.5\,, -0.5) \tag{3}$$

$$I_3 = \Delta T \in [-0.5\,, +0.5] \tag{4}$$

$$I_4 = \Delta T \in (-0.5\,, +4.5) \tag{5}$$

$$I_5 = \Delta T \in [+4.5\,, +\infty) \tag{6}$$

3000 households were used as training and test data for the regression model. Input variables were T_out, T_ctrl, Humidity and current time.

ΔT	I_1	I_2	I_3	I_4	I_5
I_1	0	0	6592	0	0
I_2	0	0	39434	0	0
I_3	0	0	437392	0	0
I_4	0	0	39116	0	0
I_5	0	0	6710	0	0

Table 3. Confusion matrix results for cluster 0 of multinomial regression model with original input variables

	precision	recall	f1-score	support
I_1	0.00	0.00	0.00	6592
I_2	0.00	0.00	0.00	39434
I_3	0.83	1.00	0.90	437392
I_4	0.00	0.00	0.00	39116
I_5	0.00	0.00	0.00	6710
total	0.69	0.83	0.74	529244

Table 4. Classification matrix for cluster 0 of multinomial regression model with original input variables

Table 3 and Table 4 are the results for cluster 0. The result from clusters 1 to 11 are not shown since there are very similar to cluster 0. As shown, the multinomial regression model best guess is to simply predict no change each hour. Even though this strategy only leads to the very rudimentary solution of avoiding predicting by simply picking the most likely category each single time, it seems as if it is the best one that the model is able to come up with. Very similar results were obtained by changing the clustering method, the number (and criteria) of the T_stp_cool categories and the number of households used as training data.

So, in order to avoid the hypothesis of simply working with a model that is somehow malfunctioning or has been programmed in such a way that is never going to provide meaningful data, a second analysis was done in which a new variable was introduced as input data: schedule.

"Schedule" is a variable that was artificially created by taking the past information of each setpoint's own past data. Schedule searches each period for the change that occurred in each household the day before at the same time (so, 24 periods ago) and if the temperature of (t-1) is equal to the temperature of that time the day before (t-25), it assigns itself the value of the change that happened the day before at that time. This way, if a household has a 24h daily schedule set for either setpoint, this variable will be able to identify it and predict it before it happens, starting on the second day.

This variable has its obvious limitations. To start, it only accounts for schedules that are 24h long, so any other schedules with different periods are not identified. It also fails in

the eventual case that the setpoint happens to be the same as the day before but the change that occurred 24h ago had nothing to do with a schedule (and so it does not reproduce the following day), and so it introduces noise to the model. Finally, the schedule variable will give little to no information when the household uses no clear schedule and the thermostat is primarily driven by human decision. It is important to note that this variable was simply created with the intention of proving whether the regression model was able to work given a good-enough input variable.

The same 3000 households were used as training and test data this time for the multinomial regression model. Input variables were T_out, T_ctrl, Humidity, current time and schedule.

Even though the results are not optimal (and could further be improved by changing the definition of schedule and the number of training households), Table 5 and Table 6 clearly show that the regression model is capable of predicting the output variable given some reasonable input information.

This overall conclusion led to replacing the multinomial regression with a more advanced technique in order to develop the setpoint prediction model. This is why Long short-term memory neural networks (LTSMs) have been used with the same purpose as the regression model. Given the flexibility and learning ability of these networks, it was though that a LSTM would have a better chance to find some kind of relationship between the setpoint variables and the input data.

ΔT	I_1	I_2	I_3	I_4	I_5
I_1	936	927	4678	48	3
I_2	88	17376	21692	260	18
I_3	345	3685	427503	5528	331
I_4	22	285	20382	18380	47
I_5	1	21	4654	145	1889

Table 5. Confusion matrix results for cluster 0 of multinomial regression model with original input variables and variable Schedule

	precision	**recall**	**f1-score**	**support**
I_1	0.67	0.14	0.23	6592
I_2	0.78	0.44	0.56	39434
I_3	0.89	0.98	0.93	437392
I_4	0.75	0.47	0.58	39116
I_5	0.83	0.28	0.42	6710
total	0.87	0.88	0.86	529244

Table 6. Classification matrix for cluster 0 of multinomial regression model with original input variables and variable schedule

Even though the results are not optimal (and could further be improved by changing the definition of schedule and the number of training households), Table 5 and Table 6 clearly show that the regression model is capable of predicting the output variable given some reasonable input information.

This overall conclusion led to replacing the multinomial regression with a more advanced technique in order to develop the setpoint prediction model. This is why Long short-term memory neural networks (LTSMs) have been used with the same purpose as the regression model. Given the flexibility and learning ability of these networks, it was though that a LSTM would have a better chance to find some kind of relationship between the setpoint variables and the input data.

4.1.2. Second Setpoint Prediction Model

The Second Setpoint prediction model also uses the same clustering analysis as the one that produces the clusters shown in Figure 13. With such clusters already defined in the first setpoint, the second prediction model has its main focus on developing the best neural network possible.

This time, given the flexibility of the output variable of LSTMs networks, it was not necessary to categorize neither T_stp_cool nor T_stp_heat into the 5 categories described in the first model. Simply using the same input and output variables as they come from the database is enough to train the network.

After an iterative tuning process, the final version of the LSTMs had 100 neurons, 5 epochs and a batch size equal to 72. 1000 households were used as training and test data for the model. Input variables were T_out, T_ctrl, Humidity and current time.

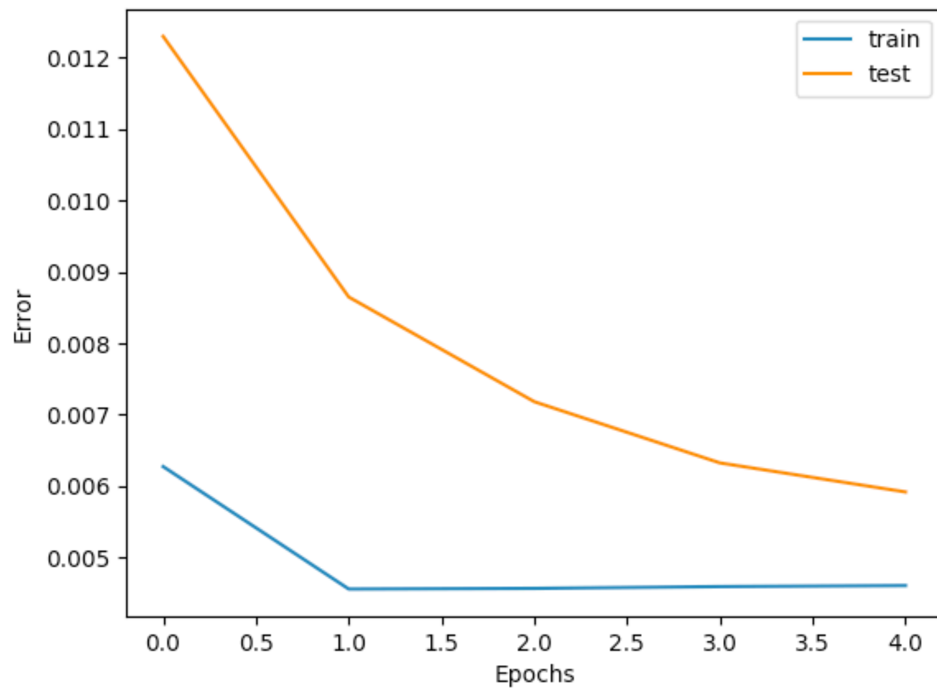


Figure 16. Cost function error of training and test data for LSTM model with T_{out}, T_{ctrl} and Humidity as inputs

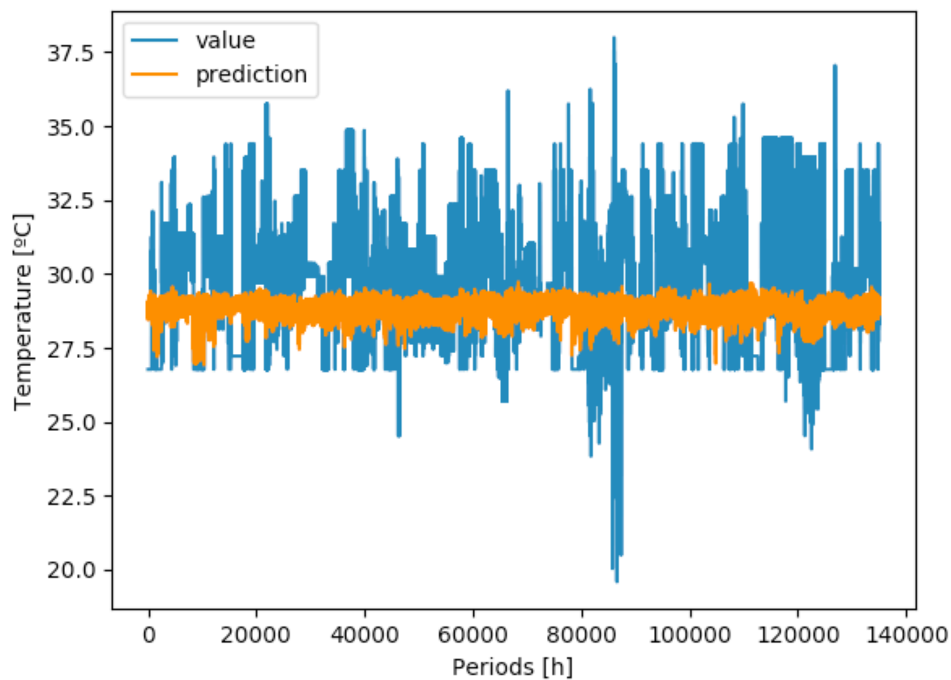


Figure 17. Hour-by-hour prediction of test data for LSTM model with T_{out}, T_{ctrl} and Humidity as input

Total RSME of cluster 0: 1.79 ºC degrees.

Even though the prediction is significantly better than the one obtained by the first model (which in essence was not predicting at all), it is still far from good. So, after running through the same quality checks as with the first model (training data size, different clusters and model parameters tuning), it was decided to once again incorporate the past information of the output variable to distinguish if the issue was on the model or the input data.

Since in this case the output variable (T_stp_cool) is going to be introduced into the model as an input variable, it is worth noting that it is only the past information of such output variable what the model will be able to use as input for each period of time.

This is what happens in the most typical case of a smart thermostat. For each particular unit of time, t, the thermostat predicts how the setpoint should behave for (t+1) given not only the internal and external weather conditions (T_out, T_ctrl, Humidity…) but also the own information of how the setpoint itself behaved in the past. This ability to reflect upon the past of the output information is especially useful when predicting how the setpoint will behave in cases where there is a strong pattern that works regardless of any external weather condition (as it can be the case of a schedule in a household where nobody knows how to use the thermostat).

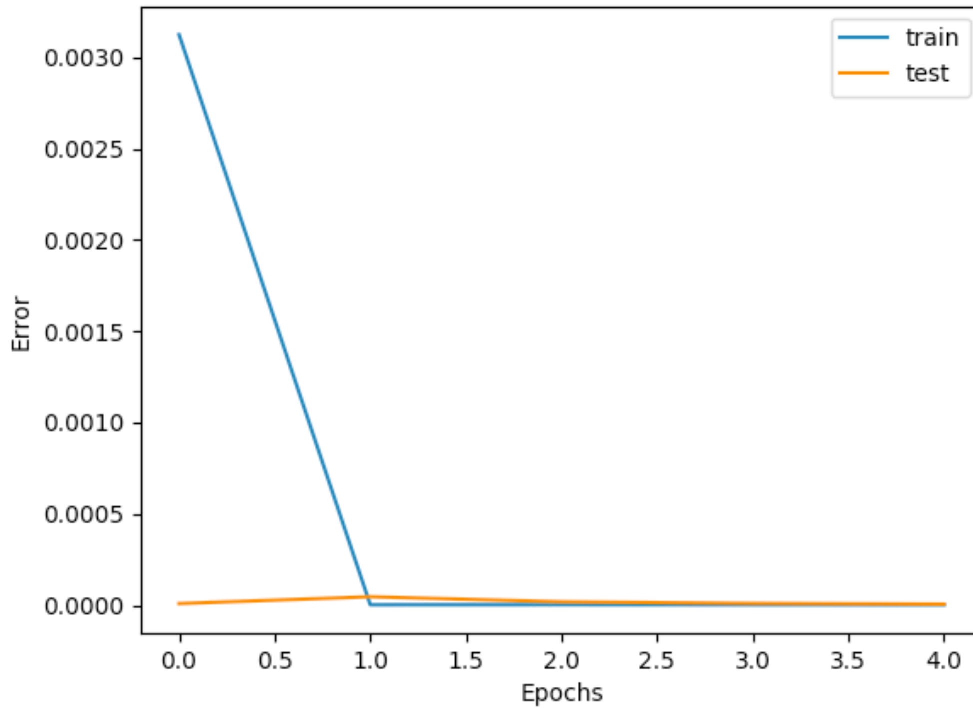


Figure 18. Cost function error of training and test data for LSTM model with past of T_{stp_cool} as input

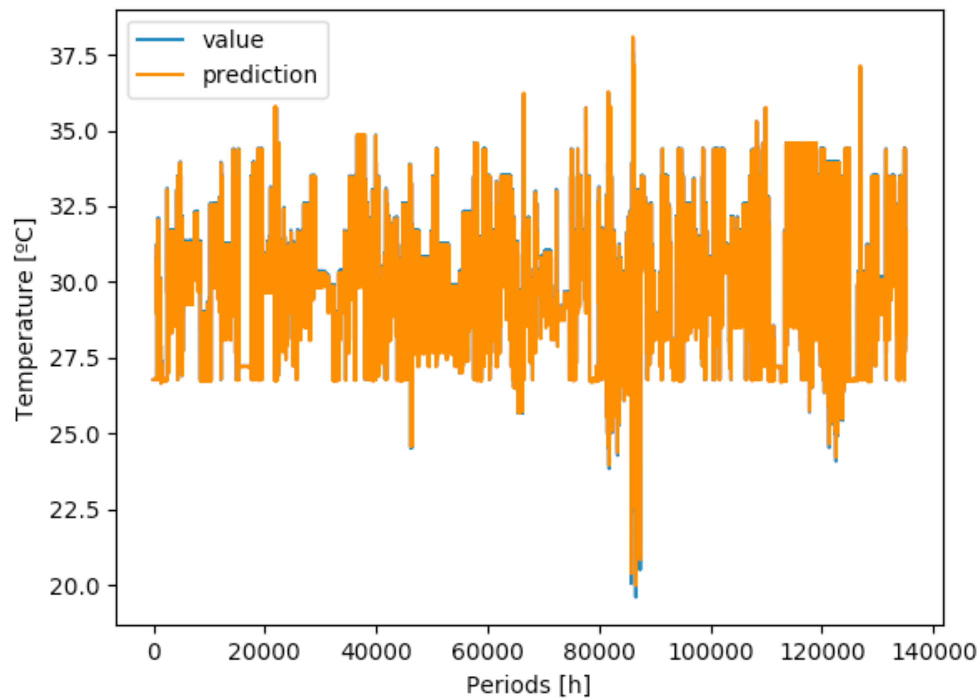


Figure 19. Hour-by-hour prediction of test data for LSTM model with past of T_{stp_cool} as input

Total RSME of cluster 0: 0.03 F degrees.

As shown in Figure 18 and Figure 19, the results obtained this time are significantly better. Remarkably, this model works almost as well with a single cluster as it does with the clustering analysis. This implies that, after trying clustering, regression and neural network techniques to predict how a residential thermostat setpoint behaves, the behavior of this variable is very independent from weather conditions and mostly depends on its own past.

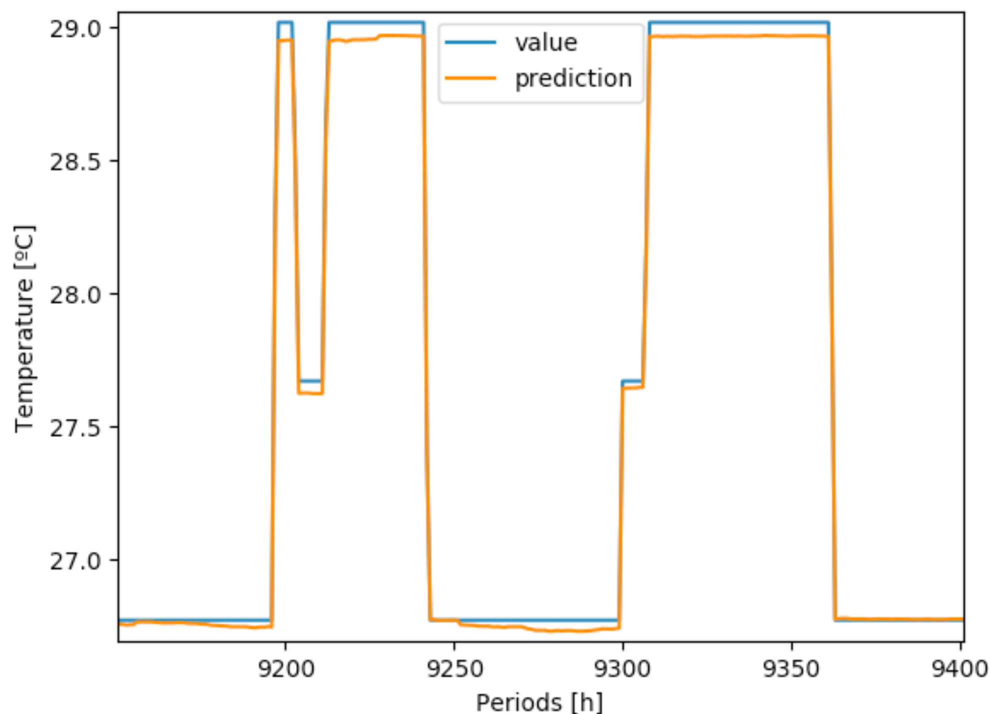


Figure 20. Close-up of prediction of test data for LSTM model with past of T_{stp_cool} as input

Or put other way: the outside temperature, inside temperature and humidity are almost complete noise when predicting the temperature setpoint of a residential thermostat. And so, predicting how the setpoint behaves based on these input variables is harder and harder the higher number of households that are being tried to be predicted.

So, although the initial setpoint prediction model failed to be developed (it might be possible, given other set of input data that this document hasn't had the access to or has overlooked) for the simulation software to use, a LSTM model has been developed with the objective of developing a smart thermostat algorithm, called in this document as the "third setpoint prediction model".

4.2. Household Setpoint Behavior Classification

With the smart thermostat's algorithm developed, the final step is to classify users according to their setpoint personal preferences and geographic location to highlight any differences between households accustomed to different weathers. The main objective, as mentioned, is to find if people in the U.S. accustomed to different weathers also have different preferences when selecting the setpoint thermostat that is "comfortable" to them.

In this case, the data was once again grouped into days (Figure 11) and plotted according to the cluster that they belong to.

The reason behind grouping the data into days was once again the inconsistency of the number of days that each household had. So, in order to be fair for each household and not compare days of summer with days of winter, only one day was initially tested. Figure 21 presents the results of the PCA analysis of the 25th of June of 2016, ignoring all the households that lacked the information of this day and plotting principal component 1 on the x-axis and principal component 2 on the y-axis.

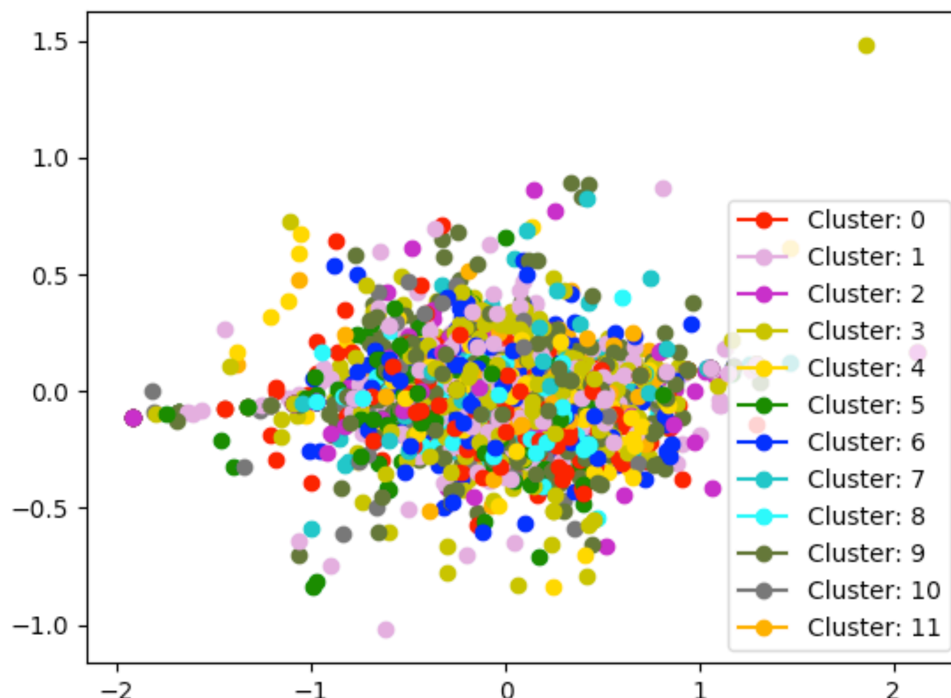


Figure 21. PCA result by clusters for one day

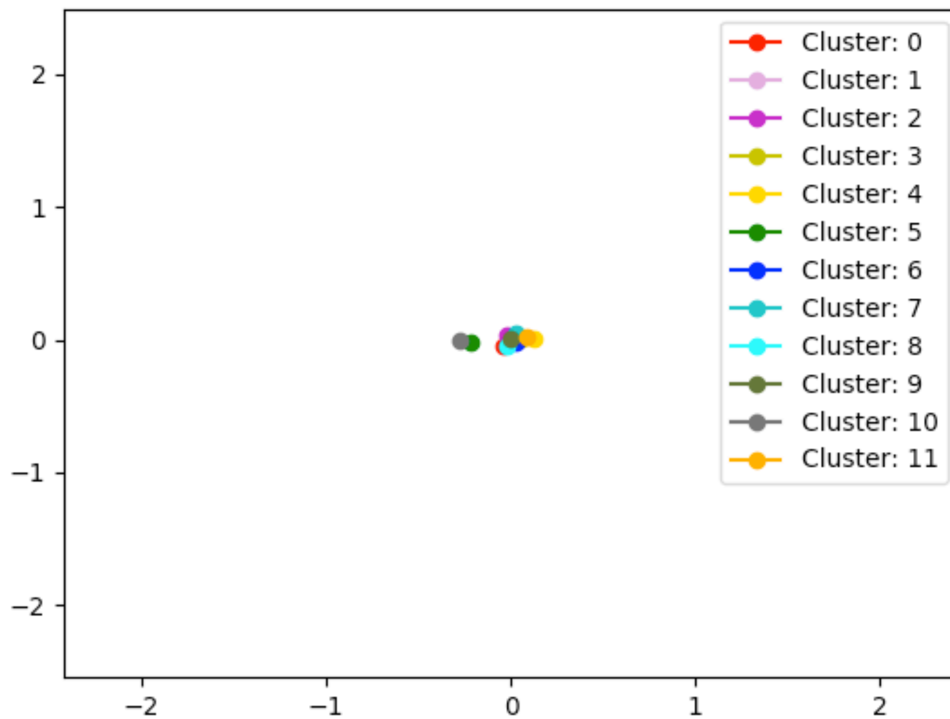


Figure 22. PCA result showing only clusters' centers

As shown in Figure 21, the households appear to be concentrated around the (0,0) –as expected– with no clear behavior pattern for any of the clusters. Nevertheless, even if each cluster appears to be represented randomly around the (0,0), the centers of each cluster were plotted in a new figure in order to identify if this impression was indeed correct since the great number of colors in the figure can sometimes mislead the naked eye. Figure 22 presents this analysis, in which the centers of each cluster are close enough to maintain the initial statement of equal behavior for each cluster.

It is true that the data of this analysis is biased by the fact that each cluster deals with different temperatures for the same day. A better analysis would be to compare days that are similar to one another even if they belong to different months or seasons. But, due to the fact that the results showed no difference between clusters with the bias in the data, this analysis was not done.

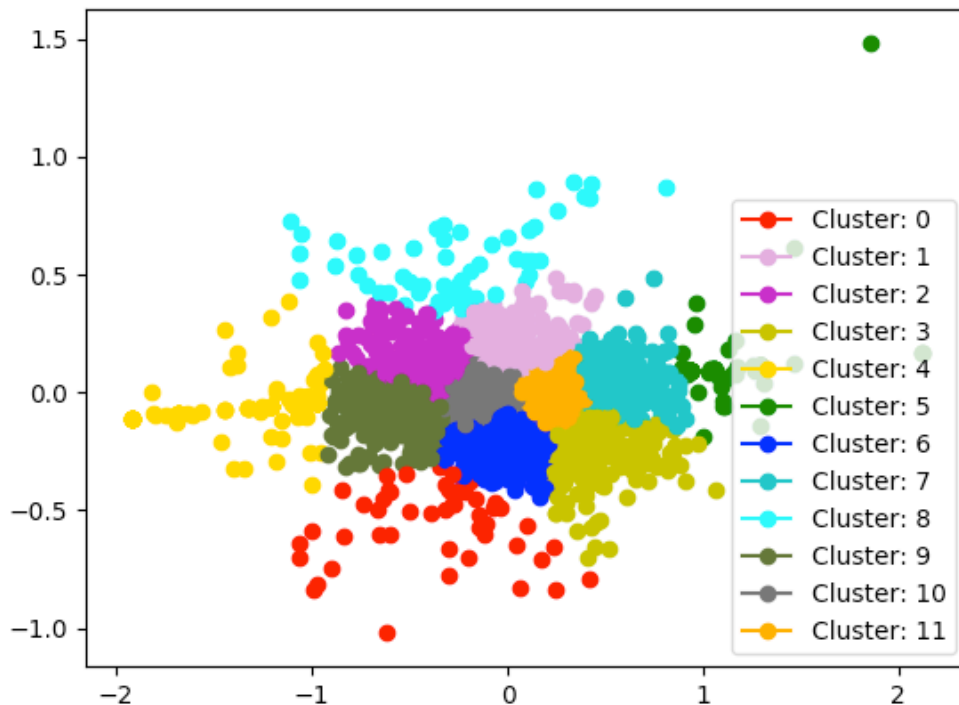


Figure 23. PCA results for one day clustered

Even if the centers are very much alike (as Figure 22 illustrates), the variability of each cluster and the impact that outliers have on the centers' final position could be playing a role in the setpoint behavior that would be hard to identify with the previous analysis. This is why, we decided to take a second approach to further validate and quantify the mismatch that exists between the localization setpoint and setpoint behavior clusters.

Given the PCA results of each household for a single day (Figure 21), we ran a KMeans clustering analysis of the data itself with the same number of clusters (12) as the localization clustering analysis. Figure 23 illustrates this analysis. This way, it would be possible to compare how well these two different clustering analysis agree.

We used both the adjusted rand index and the adjusted mutual information scores to quantify such agreement in order to account for chance. Table 7 shows the results, where it is easy to realize how different each analysis is from the other.

	Adjusted Rand index	Adjusted Mutual Info
Score	0.01238	0.02121

Table 7. Clustering comparison

The second major limitation of the previous analysis is the fact that only one day was taken into account. To address this issue, the cluster centers of each day in 2016 were calculated. Then, the centers of such centers were calculated for each cluster and plotted in Figure 24. Again, the proximity of the centers shows a great similarity in the average day setpoint behavior for each cluster in 2016. The only cluster that slightly differs from the rest is cluster 5 (mainly California-based) but is mainly due to the great number of outliers that it has.

Finally, a factor analysis was done to discover any latent variables (such as "aversion to hotness/coldness" or "preference of warmer/colder temperatures") that might help explain setpoint behavior differences that the PCA analysis might have overlooked.

In this case, the procedure is very similar to the one followed with the PCA analysis. First, the data was grouped into days to avoid the data inconsistency present in the original data and the same was picked, ignoring all the households that lacked the information of this day.

The one-day results (as shown in Figure 25) yield no clear indication of any setpoint behavior pattern, with all the datapoints scattered around the (0,0) center and no clear differentiation of clusters aside from a couple of outliers.

In Figure 26, the difference between each other is still too small to differentiate any cluster-lead setpoint behavior. Figure 27 represents the results of the factor analysis for the whole 2016, avoiding the one-day bias.

Should the results have been different (with the datapoints more clustered into groups and thus the centers more separated), a more accurate analysis would have been done where days with a more similar outside temperature would have been compared.

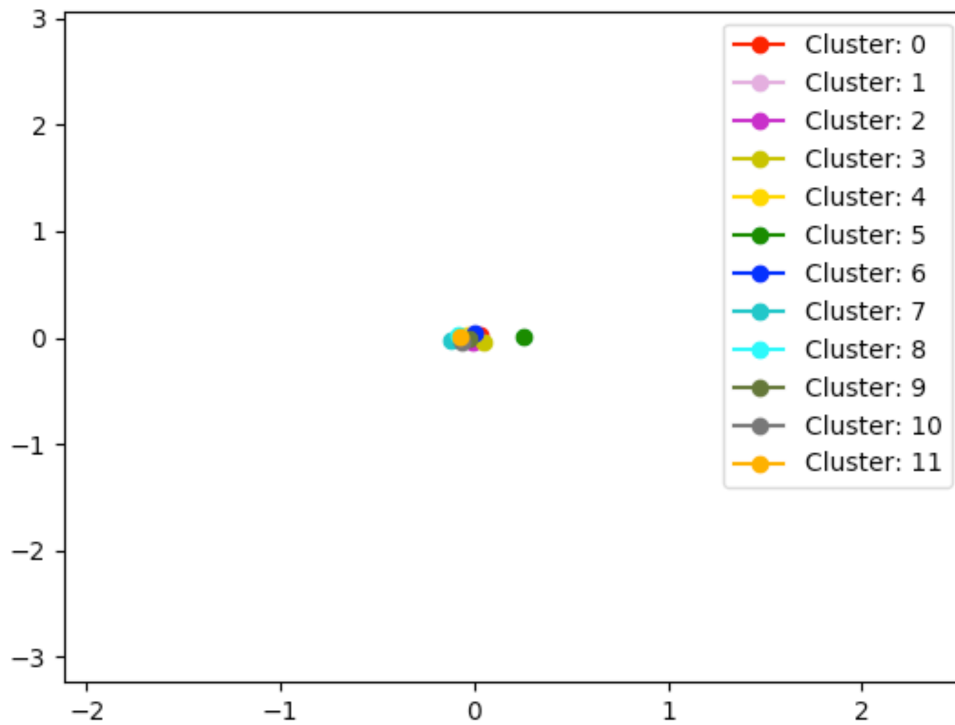


Figure 24. PCA result from centers for 2016

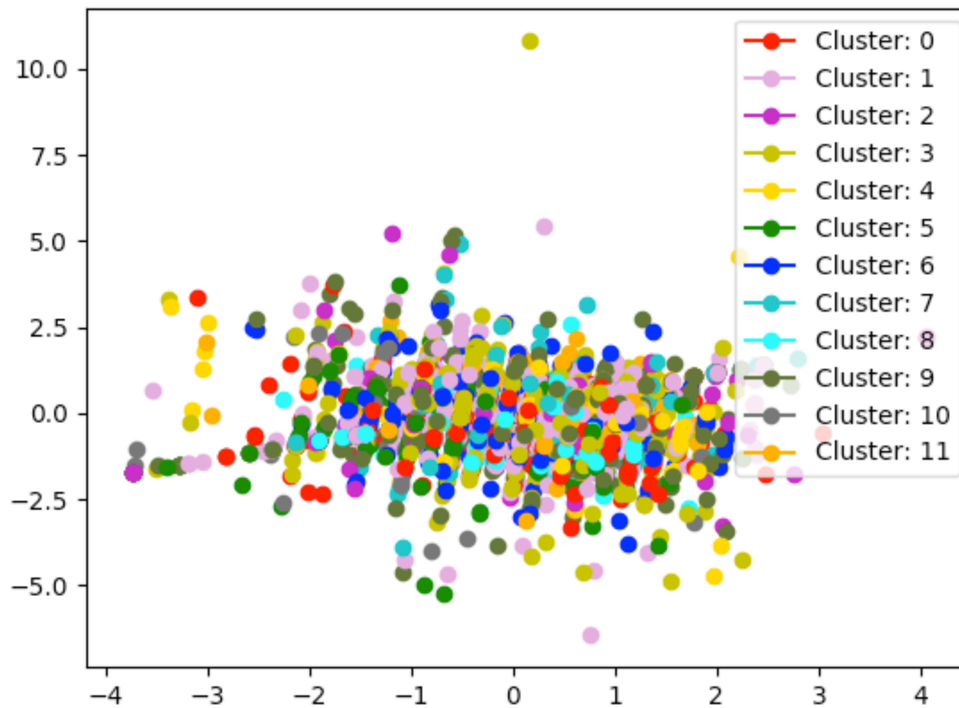


Figure 25. Factor analysis result by clusters for one day

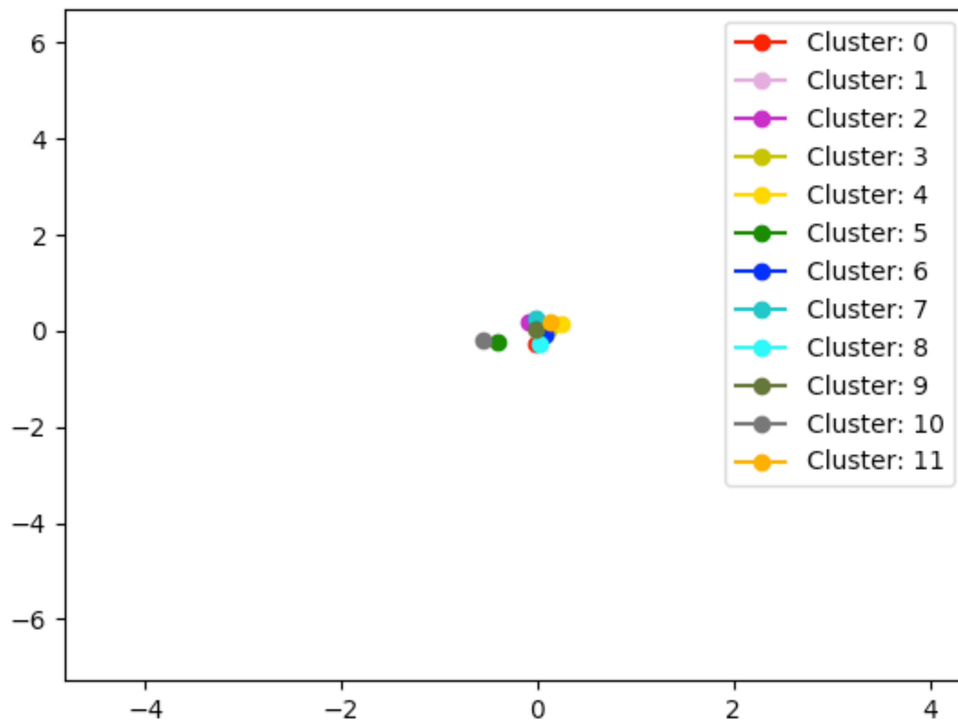


Figure 26. Factor analysis result of only clusters' centers

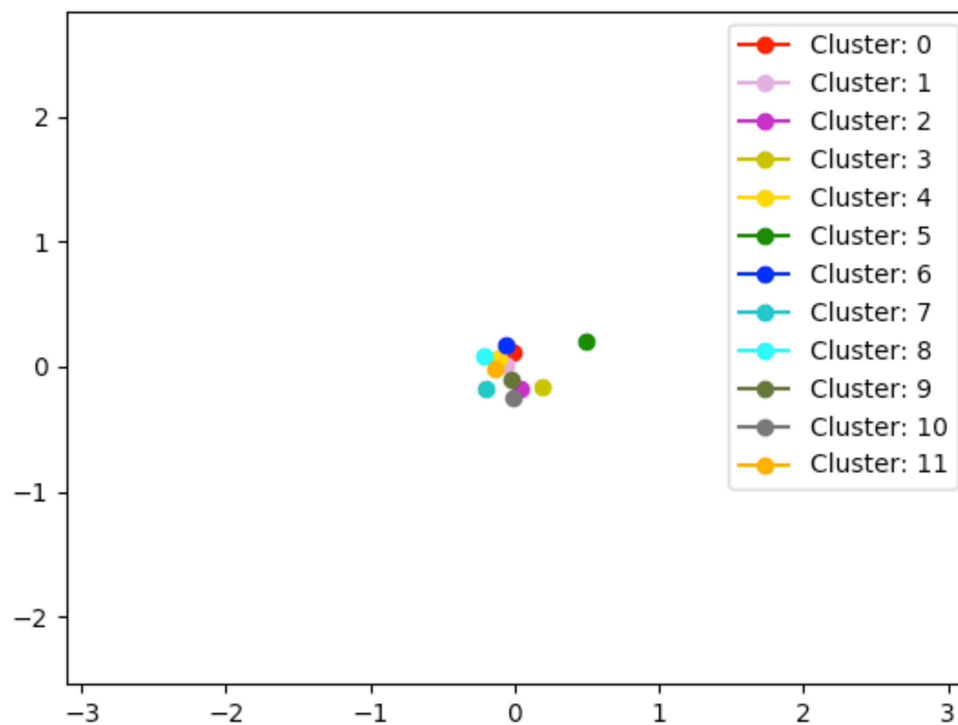


Figure 27. Factor analysis result from centers for 2016

5. Conclusion

This document has presented the analysis and results in consonance with the objective of better understanding how residential households interact with their thermostats', in particular with their heating and cooling setpoints. First, both a multinomial regression model and a neural network have been combined with a clustering analysis to try to predict how the setpoint variable behaves based on humidity, current time and outdoor and indoor temperature. These models have strengthened the hypothesis that there is no type of correlation between the input and output data, making it impossible for any model to accurately predict any output variable based on noise.

Second, this document has also presented multiple analyses to classify users according to their personal preferences and geographic location. The aim in this case was to highlight any differences between setpoint behaviors from households accustomed to different weathers. After using both PCA and Factor Analysis to define the different types of setpoint behaviors, the heterogeneity of the clusters of this analysis and the clusters obtained by location have yielded no meaningful results to prove that there is a difference on the setpoint behaviors

[This page was intentionally left blank]

6. Bibliography

[WOO10] Woo Moon J., Kim J., "ANN-based thermal control models for residential buildings", *Building and Environment 45*(2010), 1612–1625.

[LI17] Li Z., Dong B., "A new modeling approach for short-term prediction of occupancy in residential buildings", *Building and Environment 121*(2017), 277–290.

[KIM18] Kim J., Zhou Y., Schiavon S., Raftery, P., Brager, G., "Personal comfort models: Predicting individuals' thermal preference using occupant heating and cooling behavior and machine learning", Building *and Environment* 129(2018), 96–106.

[URBA12] Urban B., Elliott D., Sachs O., "Towards better modeling of residential thermostats", *Fifth National Conference of IBPSA-USA. Madison, USA.*

[BOAI10] Boait P.J., Rylatt R.M., "A method for fully automatic operation of domestic heating", *Energy and Buildings 42*(2010), 11–16.

[PEFF11] Peffer T., Pritoni M., Meier A., Aragon C., Perry D., "How people use thermostats in homes: A review", *Building and Environment 46*(2010), 2529–2541.

[HOYT15] Hoyt T., Arens E., Zhang H., "Extending air temperature setpoints: Simulated energy savings and design considerations for new and retrofit buildings", *Building and Environment 88*(2015), 89–96.

[MEIE10] Meier A., Hurwitz B., Dhawal M., Daniel P., Therese P., Marco P., "How people actually use thermostats", *Center for the Built Environment, UC Berkeley* (2010). https://escholarship.org/uc/item/37j43258

[WOOD06] Woods J., "Fiddling with thermostats: Energy implications of heating and cooling set point behavior", *ACEEE Summer Study on Energy Efficiency in Buildings* (7) 278-287.

[SHIP11] Shipworth M., "Thermostat settings in English houses: No evidence of change between 1984 and 2007", *Building and Environment 46*(2011) 635–642.

[HUCH18] Huchuk B., O'Brien W., Sanner S., "A longitudinal study of thermostat behaviors based on climate, seasonal, and energy price considerations using connected thermostat data", *Building and Environment* (2018) 30263-4.

[LU10] Lu L., Sookor T., Srinivasan V., Gao G., Holben B., Stankovic J., Field E., Whitehouse K., "The Smart Thermostat: Using Occupancy Sensors to Save Energy in Homes", *Sensys '10 Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems* (2010) 211-224

[ASHR16] ASHRAE., "What are the recommended indoor temperature and humidity levels for homes?", https://www.ashrae.org/File%20Library/Technical%20Resources/Technical%20FAQs/TC-02.01-FAQ-92.pdf

[SHIL06] Shiller D., "Programmable thermostat program proposal", US EPA. https://www.energystar.gov/ia/partners/prod_development/revisions/downloads/thermostats/Proposal_011106.pdf

[BARK12] Barker, S., Mishra A., Irwin D., Cecchet E., Shenoy P., "Smart *: An Open Data Set and Tools for Enabling Research in Sustainable Homes", *University of Massachusetts Amherst* (2012)

[This page was intentionally left blank]

7. Appendices

A. Data Exploration

```
1.  import h5py
2.  import numpy as np
3.  import string
4.
5.  import statsmodels.api as sm
6.  from statsmodels.formula.api import ols
7.  import pandas as pd
8.
9.  from math import sqrt
10. from numpy import concatenate
11. from matplotlib import pyplot as plt
12. from matplotlib.lines import Line2D
13. from matplotlib.pyplot import boxplot
14. from pandas import read_csv
15. from pandas import DataFrame
16. from pandas import concat
17.
18. from sklearn.preprocessing import MinMaxScaler
19. from sklearn.preprocessing import LabelEncoder
20. from sklearn.metrics import mean_squared_error
21. from sklearn.cluster import KMeans
22. from sklearn.decomposition import FactorAnalysis
23. from sklearn.decomposition import PCA
24.
25. from keras.models import Sequential
26. from keras.layers import Dense
27. from keras.layers import LSTM
28.
29. from datetime import datetime
30.
31. import sys
32. sys.path.insert(0, '/Users/user/Documents/Project/IP_library')
33. from Fill_one_whole_household import fill_one_whole_household
34. from Fill_one_whole_household_v2 import fill_one_whole_household_v2
35. from Transform_pre_clean import transform_pre_clean
36. from Transform_post_clean import transform_post_clean
37. from Transform_data import transform_data
38. from Transform_data_inv import transform_data_inv
39. from Transform_data_inv_v2 import transform_data_inv_v2
40. from Number_clusters_optimization import number_clusters_optimization
41. from Get_data import get_data
42.
43. import copy
44. from scipy.constants import F2C
45. from scipy import stats
46.
47. ################################################################################
    ################################################################################
    ######
48. ################################################################################
    ################################################################################
    ######
```

```
49. ################################################################################
    ################################################################################
    ######
50. ############################################
                                            ########################################
    ######
51. ############################################                                 DAT
    A EXPLORATION                           ########################################
    ######
52. ############################################
                                            ########################################
    ######
53. ################################################################################
    ################################################################################
    ######
54. ################################################################################
    ################################################################################
    ######
55. ################################################################################
    ################################################################################
    ######
56.
57. """
58. This script has the objective of visually represent some of the data
59. that is contained in the h5df database.
60.
61. First, the "get data" section reads, cleans and stores the information
62. contained in the hd5f database to appropriated variables.
63.
64. Once this is done, the data is clustered according to the location of
65. the household they belong to, clustering each user with nearby households
66. that will likely experience a similar weather. The number of clusters is
67. defined at the beginning of the script.
68.
69. There are 3 main plots in this script:
70. 1) Plot Clusters: this plot illustrates the location of each household and
71. the cluster they belong to. This plot is helpful to check that the kmeans
72. clustering model works correctly while also checking that each cluster
73. represents one climate
74.
75. 2) Point plot: this plot illustrates the relationship that exists between
76. two different variables. The selected variables for this script where
77. outside temperature versus cooling/heating setpoint (one plot for each
78. case). This plot is helpful to see any preliminary linear or non-linear
79. relationship that each setpoint might have with the outside temperature.
80.
81. 3) Box plot: this plot illustrates the distribution that a particular
82. variable has for each cluster. Whether that variable is outside temperature,
83. cooling or heating setpoint, this plot is useful to reject the null
84. hypothesis of equal means between variables.
85.
86. These 3 plots are all commented since the script works best when only one
87. of them is uncommeted at a time.
88. """
89.
90. #########################
91. households = 6739   # Number of households to extract from database
92. st_household = 0    # Starting household of DB to extract
93. st_year = 2015      # Starting year to extract for each household
```

```
94. fin_year = 2016     # Final year to extract for each household
95. st_month = 1        # Starting month to extract for each household
96. fin_month = 12      # Final month to extract for each household
97.
98. sel_buildings = []  # Selected buildings for data analysis
99. k_labels = []       # Clustering labels
100.        clusters = 12       # Number of clusters
101.        BASE_DIR = '/Users/user/Documents/Project/Exports/'
102.        LABEL_COLOR_MAP = {0:'r', 1:'plum', 2:'m', 3:'y', 4:'gold', 5:'g', 6:'b
    ', 7:'c', 8:'cyan', 9:'darkolivegreen', 10:'dimgray', 11: 'orange', 12:'k', 13
    :'navy'}   # Color map for plots
103.        #########################
104.
105.
106.        ################################ GET DATA #############################
    ###
107.
108.        start = datetime.now()
109.        print("start: ",start)
110.        data = []
111.        data = get_data('npy', households, st_household, st_year, fin_year, st_
    month, fin_month)
112.        print(data.shape)
113.        print("Time elapsed: ", datetime.now()-start)
114.
115.        ## Fahrenheit to Celsius
116.        data[:,5] = copy.copy(F2C(data[:,5]))
117.        data[:,6] = copy.copy(F2C(data[:,6]))
118.        data[:,8] = copy.copy(F2C(data[:,8]))
119.        data[:,9] = copy.copy(F2C(data[:,9]))
120.
121.        print("Data acquisition completed: ",datetime.now(),"\n")
122.
123.        #######################################################################
    ###
124.
125.
126.        ############################### CLUSTERING ############################
    ###
127.
128.        T_out_clus = []
129.        T_ctrl_clus = []
130.        T_stp_cool_clus = []
131.        Humidity_clus = []
132.        x_clus = []
133.
134.        ## Clustering method 2 (metadata)
135.        days_flag = False
136.        household_latitude = []
137.        household_longitude = []
138.        households_ids = np.unique(data[:,0])
139.        print(len(households_ids))
140.        household_latitude_full = []
141.        household_longitude_full = []
142.
143.        ## Read metadata
144.        with open('/Users/user/Documents/Project/Metadata_full_2.txt', 'r') as
    ff:
```

```
145.            household_latitude_full.append([float(i) for i in tuple(zip(*[line.
    split(";") for line in ff]))[4]])
146.            ff.seek(0)
147.            household_longitude_full.append([float(i) for i in tuple(zip(*[line
    .split(";") for line in ff]))[5]])
148.        household_latitude_full = np.array(household_latitude_full)
149.        household_longitude_full = np.array(household_longitude_full)
150.        household_latitude_full = household_latitude_full.reshape(-1,1)
151.        household_longitude_full = household_longitude_full.reshape(-1,1)
152.
153.        print("latitude, longitude full shape: ", household_latitude_full.shape
    , household_longitude_full.shape)
154.
155.        ## Household Selection
156.        for i in range(0,len(household_longitude_full)):
157.            if(household_longitude_full[i] > -
    130 and household_longitude_full[i] < -
    50 and household_latitude_full[i] > 20 and household_latitude_full[i] < 60):
158.                sel_buildings = np.append(sel_buildings, i)
159.
160.        sel_buildings = np.array([int(i) for i in sel_buildings])
161.        data = data[np.where(np.in1d(data[:,0].astype(int), sel_buildings[:].as
    type(int)))]
162.        print("data shape: ", data.shape)
163.        print("sel buildings shape: ", sel_buildings.shape)
164.
165.        for i in sel_buildings:
166.            household_latitude = np.append(household_latitude, household_latitu
    de_full[int(i)])
167.            household_longitude = np.append(household_longitude, household_long
    itude_full[int(i)])
168.        household_latitude = household_latitude.reshape(-1,1)
169.        household_longitude = household_longitude.reshape(-1,1)
170.
171.        x_clus = np.hstack((household_latitude, household_longitude))
172.        scaler_clus = MinMaxScaler(feature_range=(0,1))
173.        x_clus = scaler_clus.fit_transform(x_clus)
174.
175.        ## Clustering model
176.        kmmodel = KMeans(n_clusters = clusters, random_state = 0).fit(x_clus)
177.        labels = kmmodel.labels_
178.        labels_box = np.hstack((sel_buildings.reshape(-1,1),labels.reshape(-
    1,1)))
179.        print("Labels: ",labels.shape)
180.
181.        ## Plot KMeans Inertia to select number of clusters
182.        #number_clusters_optimization(x_clus)
183.
184.        print("Clustering: ",datetime.now())
185.
186.        ########################################################################
    ###
187.
188.        """
189.        ############################# PLOT CLUSTERS ###########################
    ###
190.
191.        ## Plot
192.        fig, ax = plt.subplots()
```

```
193.        for i in range(0, len(labels)):
194.            cluster = labels[i]
195.            ax.scatter(household_longitude[i], household_latitude[i], color=LAB
    EL_COLOR_MAP[cluster], label = 'household '+str(int(i)))
196.
197.        ## Plot legend and colors
198.        legend_elements = []
199.        label_elements = []
200.        for i in range(0,clusters):
201.            legend_elements = np.append(legend_elements, Line2D([0],[0],marker=
    'o', color=LABEL_COLOR_MAP[i], label='Cluster: '+str(int(i))))
202.            label_elements = np.append(label_elements, 'Cluster: '+str(int(i)))

203.        #ax.legend(legend_elements, label_elements)
204.
205.        plt.show()
206.
207.        #######################################################################
    ###
208.        """
209.
210.        """
211.        ############################### POINT PLOT ############################
    ###
212.
213.        ## Variable declaration
214.        clean_T_out = copy.copy(data[:,5])
215.        clean_T_stp = copy.copy(data[:,8])
216.
217.        x_pp = copy.copy(clean_T_out).reshape(-1,1)
218.        y_pp = copy.copy(clean_T_stp).reshape(-1,1)
219.
220.        ## Data normalization
221.        x_clus = np.hstack((x_pp, y_pp))
222.        scaler_clus = MinMaxScaler(feature_range=(0,1))
223.        x_clus = scaler_clus.fit_transform(x_clus)
224.
225.        ## Plot
226.        fig = plt.figure()
227.        ax = fig.add_subplot(1,1,1)
228.
229.        ## Axis labels
230.        ax.scatter(x_pp,y_pp)
231.        ax.set_ylabel("T_stp_cool [ºC]")
232.        ax.set_xlabel("T_ctrl [ºC]")
233.        plt.show()
234.
235.
236.        #######################################################################
    ###
237.        """
238.
239.        """
240.        ############################# BOX PLOT T_OUT ##########################
    ###
241.
242.        ## Variable to use in box plot
243.        variable_to_plot = 9
244.
```

```
245.        ## Labels column creation
246.        x_labels = []
247.        start_data = datetime.now()
248.        x_labels = transform_data_inv_v2(labels_box,data[:,:2],only_labels = Tr
    ue, by_days = days_flag)
249.        x_labels = x_labels.reshape(-1,1)
250.        print("Time elapsed -----> ",datetime.now() - start_data)
251.        print("x_labels: ",x_labels.shape)
252.
253.        ## Plot
254.        fig = plt.figure()
255.        ax = fig.add_subplot(1,1,1)
256.
257.        ## Split data by cluster
258.        for i in range(0,clusters):
259.            exec("T_out_%s = copy.copy(data[np.where(x_labels[:] == %s)[0],vari
    able_to_plot])" %(i,i))
260.
261.        ## Axis labels
262.        string = ""
263.        for x in range(0,clusters):
264.            if(x == clusters-1):
265.                string += "eval(\"T_out_" + str(x) + "\")"
266.            else:
267.                string += "eval(\"T_out_" + str(x) + "\"),"
268.        eval("ax.boxplot([%s], vert = False, positions = np.arange(0,clusters,1
    ))" %(string))
269.        ax.set_ylabel("Cluster name")
270.        ax.set_xlabel("T_stp_heat [ºC]")
271.
272.        plt.show()
273.
274.        ## Kruskal-Wallis (ANOVA)
275.        exec("test = stats.kruskal(%s)" %(string))
276.        print(eval("test"))
277.
278.        ########################################################################
    ###
279.        """
```

B. Multinomial Regression

```
1.  import h5py
2.  import numpy as np
3.
4.  import matplotlib.pyplot as plt
5.
6.  from sklearn.linear_model import LogisticRegression
7.  from sklearn.model_selection import train_test_split
8.  from sklearn import metrics
9.  from sklearn.feature_selection import RFE
10. from sklearn.ensemble import ExtraTreesClassifier
11. from sklearn.cluster import KMeans
12. from sklearn.preprocessing import MinMaxScaler
13.
14. from datetime import datetime
15.
16. import sys
17. sys.path.insert(0,'/Users/user/Documents/Project/IP_library')
18. from Plot_one_month import Plot_one_month
19. from Plot_one_year import Plot_one_year
20. from Detect_change import detect_change
21. from Clean_nan import clean_nan
22. from Plot_model_prediction import Plot_model_prediction
23. from Cuantify_change import cuantify_change
24. from Transform_data import transform_data
25. from Transform_data_multinom import transform_data_multinom
26. from Fill_one_whole_household import fill_one_whole_household
27. from Transform_pre_clean import transform_pre_clean
28. from Transform_post_clean import transform_post_clean
29. from Get_data import get_data
30.
31. import copy
32.
33. ################################################################################
    ################################################################################
    ######
34. ################################################################################
    ################################################################################
    ######
35. ################################################################################
    ################################################################################
    ######
36. ############################################
                                        ########################################
    ######
37. ############################################                       MULTINOMI
    AL REGRESSION MODEL                 ########################################
    ######
38. ############################################
                                        ########################################
    ######
39. ################################################################################
    ################################################################################
    ######
40. ################################################################################
    ################################################################################
    ######
```

```
41. ##########################################################################################################################################################################
42.
43. """
44. This script has the objective of running a multinomial regression analysis
45. given a chosen number of input variables to predict how either the cooling
46. or heating setpoint variables behave.
47.
48. First, the "get data" section reads, cleans and stores the information
49. contained in the hd5f database to appropriated variables. The "dummy"
50. variables created in this section are nothing but a discretisation of the
51. output variable in order for the multinomial regression model to work
52. (see "Cuantify_change.py")
53.
54. Once this is done, the data is clustered according to their humidity and
55. outside and inside temperature. The number of clusters is defined at the
56. beginning of the script.
57.
58. Finally, the regression analysis is done once for each cluster. Given that
59. the data was transformed into day format for the clustering analysis, the
60. first step in this section is to reverse this transformation for the
61. clustering labels to have the correct size. Each variable is then split
62. into training / test set before running the regression model.
63. """
64.
65.
66. ## Open file
67. fileName = '/Users/user/Documents/Project/hourlyData.h5'
68. f = h5py.File(fileName, libver='latest')
69. file_names = list(f.keys())
70. i = 4435
71.
72. full_name = '/' + file_names[i] + '/2016/05/table'
73. semi_full_name = '/' + file_names[i] + '/2016/05'
74.
75. ## Plot household timeseries for data exploration purposes
76. #Plot_one_month(2015,1,1,i,'month')
77. #Plot_one_year(2015,3347)
78. f.close()
79.
80.
81. ###########################
82. households = 3000    # Number of households to extract from database
83. st_household = 1     # Starting household of DB to extract
84. st_year = 2015       # Starting year to extract for each household
85. fin_year = 2016      # Final year to extract for each household
86. st_month = 1         # Starting month to extract for each household
87. fin_month = 12       # Final month to extract for each houeshold
88. clusters = 12        # Number of clusters for clustering analysis
89. ###########################
90.
91.
92. ################################ GET DATA ################################
93.
94. print("start: ",datetime.now())
95. data = []
96. data = get_data('npy', households, st_household, st_year, fin_year, st_month,
    fin_month)
```

```
97. print(data.shape)
98.
99. clean_household = copy.copy(data[:,0])
100.        clean_Year = copy.copy(data[:,1])
101.        clean_Month = copy.copy(data[:,2])
102.        clean_Day = copy.copy(data[:,3])
103.        clean_Hour = copy.copy(data[:,4])
104.        clean_T_out = copy.copy(data[:,5])
105.        clean_T_ctrl = copy.copy(data[:,6])
106.        clean_Humidity = copy.copy(data[:,7])
107.        clean_T_stp_cool = copy.copy(data[:,8])
108.        clean_T_stp_heat = copy.copy(data[:,9])
109.        clean_ID = copy.copy(data[:,10])
110.
111.        ## Schedule variable (created to test reliability of regression model)

112.        clean_schedule = []
113.        clean_schedule = np.zeros((len(clean_T_stp_cool)))
114.        for i in range(0, len(clean_schedule)):
115.            try:
116.                past_1 = round(clean_T_stp_cool[i-
    24]) - round(clean_T_stp_cool[i-25])
117.                if(past_1 != 0 and round(clean_T_stp_cool[i-
    1]) == round(clean_T_stp_cool[i-25])):
118.                    if(past_1 > 0 and past_1 < 5):
119.                        clean_schedule[i] = 1
120.                    if(past_1 < 0 and past_1 > -5):
121.                        clean_schedule[i] = -1
122.                    if(past_1 > 5):
123.                        clean_schedule[i] = 5
124.                    if(past_1 < -5):
125.                        clean_schedule[i] = -5
126.            except:
127.                pass
128.
129.        ## Output variables
130.        dummy_T_stp_cool = []
131.        dummy_T_stp_cool = copy.copy(clean_T_stp_cool)
132.        dummy_T_stp_cool = cuantify_change(dummy_T_stp_cool)
133.
134.        dummy_T_stp_heat = []
135.        dummy_T_stp_heat = copy.copy(clean_T_stp_heat)
136.        dummy_T_stp_heat = cuantify_change(dummy_T_stp_heat)
137.
138.        print("Data acquisition completed: ",datetime.now())
139.
140.        ########################################################################
    ###
141.
142.
143.
144.        ############################### CLUSTERING ############################
    ###
145.
146.        T_out_clus = []
147.        T_ctrl_clus = []
148.        Humidity_clus = []
149.        print(clean_Hour.shape, clean_T_out.shape)
150.
```

```
151.        ## Data transformation into "day-format" for clustering
152.        T_out_clus = transform_data(np.hstack((clean_Hour.reshape(-
    1,1) , clean_T_out.reshape(-1,1))))
153.        T_ctrl_clus = transform_data(np.hstack((clean_Hour.reshape(-
    1,1) , clean_T_ctrl.reshape(-1,1))))
154.        Humidity_clus = transform_data(np.hstack((clean_Hour.reshape(-
    1,1) , clean_Humidity.reshape(-1,1))))
155.
156.        print("Transformation done: ",datetime.now())
157.
158.        ## Data normalization
159.        x = []
160.        x = np.hstack((T_out_clus,T_ctrl_clus,Humidity_clus))
161.        x = (x-np.min(x,axis=0))/(np.max(x, axis=0)-np.min(x, axis=0))
162.
163.        ## KMeans
164.        kmmodel = KMeans(n_clusters = clusters, random_state = 0).fit(x)
165.        labels = kmmodel.labels_
166.
167.        print("Clustering: ",datetime.now())
168.
169.        ######################################################################
    ###
170.
171.
172.
173.        ######################### MULTINOMIAL REGRESSION ######################
    ###
174.
175.        for i in range(0,clusters):
176.            exec("T_stp_cool_md%s = []" %(i))
177.            exec("T_out_m%s = []" %(i))
178.            exec("T_ctrl_m%s = []" %(i))
179.            exec("Humidity_m%s = []" %(i))
180.            exec("Hour_m%s = []" %(i))
181.            exec("schedule_m%s = []" %(i))
182.            exec("clean_Remote_sensor_m%s = []" %(i))
183.
184.        T_out_m = []
185.        T_ctrl_m = []
186.        Humidity_m = []
187.        Hour_m = []
188.        T_stp_cool_md = []
189.        schedule_m = []
190.        Remote_sensor_m = []
191.
192.        print("Regression step 1: ",datetime.now())
193.
194.        ## Inverse transformation for clustering labels previously obtained
195.        T_out_m = transform_data_multinom(labels, clean_T_out)
196.        T_ctrl_m = transform_data_multinom(labels, clean_T_ctrl)
197.        Humidity_m = transform_data_multinom(labels, clean_Humidity)
198.        Hour_m = transform_data_multinom(labels, clean_Hour)
199.        schedule_m = transform_data_multinom(labels, clean_schedule)
200.        T_stp_cool_md = transform_data_multinom(labels, dummy_T_stp_cool)
201.
202.
203.        print("Regression step 2: ",datetime.now())
204.
```

```
205.        for cluster in range(0,clusters):
206.            ## Split variables according to the cluster they belong to
207.            exec("T_out_m%s = copy.copy(T_out_m[np.where(T_out_m[:,0] == %s),1]
    )" %(cluster, cluster))
208.            exec("T_ctrl_m%s = copy.copy(T_ctrl_m[np.where(T_ctrl_m[:,0] == %s)
    ,1])" %(cluster, cluster))
209.            exec("Humidity_m%s = copy.copy(Humidity_m[np.where(Humidity_m[:,0]
    == %s),1])" %(cluster, cluster))
210.            exec("schedule_m%s = copy.copy(schedule_m[np.where(schedule_m[:,0]
    == %s),1])" %(cluster, cluster))
211.            exec("Hour_m%s = copy.copy(Hour_m[np.where(Hour_m[:,0] == %s),1])"
    %(cluster, cluster))
212.            exec("T_stp_cool_md%s = copy.copy(T_stp_cool_md[np.where(T_stp_cool
    _md[:,0] == %s), 1])" %(cluster, cluster))
213.
214.            ## Reshape (needed for regression)
215.            exec("T_out_m%s = T_out_m%s.reshape((len(T_out_m%s[0,:]),1))" %(clu
    ster, cluster, cluster))
216.            exec("T_ctrl_m%s = T_ctrl_m%s.reshape((len(T_ctrl_m%s[0,:]),1))" %(
    cluster, cluster, cluster))
217.            exec("Humidity_m%s = Humidity_m%s.reshape((len(Humidity_m%s[0,:]),1
    ))" %(cluster, cluster, cluster))
218.            exec("schedule_m%s = schedule_m%s.reshape((len(schedule_m%s[0,:]),1
    ))" %(cluster, cluster, cluster))
219.            exec("Hour_m%s = Hour_m%s.reshape((len(Hour_m%s[0,:]),1))" %(cluste
    r, cluster, cluster))
220.            exec("T_stp_cool_md%s = T_stp_cool_md%s.reshape((len(T_stp_cool_md%
    s[0,:]),1))" %(cluster, cluster, cluster))
221.
222.
223.        print("Regression step 3: ",datetime.now())
224.
225.        for a in range(0, clusters):
226.            try:
227.                ## Format input and output variables and then split into train
    / test sets
228.                exec("x_%s = np.hstack((T_out_m%s, T_ctrl_m%s, Humidity_m%s, Ho
    ur_m%s, schedule_m%s))" %(a,a,a,a,a,a))
229.                exec("y_%s = T_stp_cool_md%s" %(a,a))
230.                exec("x_train_%s, x_test_%s, y_train_%s, y_test_%s = train_test
    _split(x_%s, np.ravel(y_%s), test_size=0.3, random_state=42)" %(a,a,a,a,a,a))

231.                exec("model_%s = LogisticRegression(multi_class='multinomial',s
    olver ='newton-cg',max_iter=500000).fit(x_train_%s, y_train_%s)" %(a, a, a))
232.                exec("modelpred_%s = model_%s.predict(x_test_%s)" %(a,a,a))
233.                exec("modelprob_%s = model_%s.predict_proba(x_test_%s)" %(a,a,a
    ))
234.
235.                ## Results
236.                print("------------------------ CLUSTER ", a, "----------------
    --------")
237.                print(eval("metrics.confusion_matrix(y_test_%s,modelpred_%s)" %
    (a,a)))
238.                print(eval("metrics.classification_report(y_test_%s,modelpred_%
    s)" %(a,a)))
239.            except:
240.                pass
241.
242.
```

```
243.        ## Save txt of output (double-check solution or debug)
244.        #np.savetxt("/Users/user/Documents/Project/Exports/probs_0_"+str(datetime.now().strftime('%d%m%y-%H%M')),eval("modelprob_0"),fmt = '%s',delimiter=",")
245.        #np.savetxt("/Users/user/Documents/Project/Exports/y_dumm_0_"+str(datetime.now().strftime('%d%m%y-%H%M')),eval("y_test_0"),fmt = '%s',delimiter=",")
246.        #np.savetxt("/Users/user/Documents/Project/Exports/probs_retrof_0_"+str(datetime.now().strftime('%d%m%y-%H%M')),eval("model_0.predict_proba(x_train_0)"),fmt = '%s',delimiter=",")
247.
248.        ###########################################################################
```

C. LSTM Neural Network

```
1.  import h5py
2.  import numpy as np
3.  
4.  from math import sqrt
5.  from numpy import concatenate
6.  from matplotlib import pyplot as plt
7.  from pandas import read_csv
8.  from pandas import DataFrame
9.  from pandas import concat
10. 
11. from sklearn.preprocessing import MinMaxScaler
12. from sklearn.preprocessing import LabelEncoder
13. from sklearn.metrics import mean_squared_error
14. from sklearn.cluster import KMeans
15. 
16. from keras.models import Sequential
17. from keras.layers import Dense
18. from keras.layers import LSTM
19. 
20. from datetime import datetime
21. 
22. import sys
23. sys.path.insert(0, '/Users/user/Documents/Project/IP_library')
24. from Fill_one_whole_household import fill_one_whole_household
25. from Transform_pre_clean import transform_pre_clean
26. from Transform_post_clean import transform_post_clean
27. from Supervised_format import supervised_format
28. from Transform_data import transform_data
29. from Transform_data_inv import transform_data_inv
30. from Transform_data_inv_1D import transform_data_inv_1D
31. from Number_clusters_optimization import number_clusters_optimization
32. from Get_data import get_data
33. 
34. import copy
35. from scipy.constants import F2C
36. 
37. 
38. ########################################################################################################################################################################
39. ########################################################################################################################################################################
40. ########################################################################################################################################################################
41. ############################################                                        ##################################################
42. ############################################                FIRST NEURAL NETWORK MODEL (LSTM)                ##################################################
43. ############################################                                        ##################################################
```

```
44. ################################################################################
    ######
45. ################################################################################
    ######
46. ################################################################################
    ######
47.
48. """
49. This script has the objective of running a LSTM neural network given a
50. chosen number of input variables to predict how either the cooling or
51. heating setpoint variables behave.
52.
53. First, the "get data" section reads, cleans and stores the information
54. contained in the hd5f database to appropriated variables.
55.
56. Once this is done, the data is clustered according to their humidity and
57. outside and inside temperature. There is a second clustering method
58. (commented) which can be used instead of this one. This method clusters
59. information according to the location of the household they belong to,
60. clustering each user with nearby households that will likely experience
61. a similar weather. The number of clusters is defined at the beginning
62. of the script.
63.
64. Finally, a different neural network is trained for each cluster. Given that
65. the data was transformed into day format for the clustering analysis, the
66. first step in this section is to reverse this transformation for the
67. clustering labels to have the correct size. Each variable is then split
68. into training / test set before running the regression model.
69. """
70.
71. #########################
72. households = 1000   # Number of households to extract from database
73. st_household = 0    # Starting household of DB to extract
74. st_year = 2015      # Starting year to extract for each household
75. fin_year = 2016     # Final year to extract for each household
76. st_month = 1        # Starting month to extract for each household
77. fin_month = 12      # Final month to extract for each houeshold
78.
79. clusters = 12       # Number of clusters for clustering analysis
80. neurons = 100       # Number of neurons for LSTM model
81. epochs = 5          # Numbber of epochs for LSTM model
82. batch_size = 72     # Batch size for LSTM model
83. look_back = 1       # Number of past hours given as input for LSTM model
84.
85. BASE_DIR = '/Users/user/Documents/Project/Exports/'
86. #########################
87.
88.
89. ################################ GET DATA ################################
90.
91. start = datetime.now()
92. print("start: ",start)
93. data = []
94. data = get_data('npy', households, st_household, st_year, fin_year, st_month,
    fin_month)
95. print(data.shape)
```

```
96. print("Time elapsed: ", datetime.now()-start)
97.
98. ## Fahrenheit to Celsius
99. data[:,5] = copy.copy(F2C(data[:,5]))
100.        data[:,6] = copy.copy(F2C(data[:,6]))
101.        data[:,8] = copy.copy(F2C(data[:,8]))
102.        data[:,9] = copy.copy(F2C(data[:,9]))
103.
104.        ## Select x and y from data
105.        x = copy.copy(data[:,4:8])
106.        y = copy.copy(data[:,8])
107.
108.        print("Data acquisition completed: ",datetime.now(),"\n")
109.
110.        #######################################################################
    ###
111.
112.
113.        ############################### CLUSTERING ############################
    ###
114.
115.        T_out_clus = []
116.        T_ctrl_clus = []
117.        Humidity_clus = []
118.        x_clus = []
119.
120.        ## Clustering method 1 (by days)
121.        days_flag = True
122.        T_out_clus = transform_data(np.hstack((data[:,4].reshape(-
    1,1), data[:,5].reshape(-1,1))))
123.        T_ctrl_clus = transform_data(np.hstack((data[:,4].reshape(-
    1,1), data[:,6].reshape(-1,1))))
124.        Humidity_clus = transform_data(np.hstack((data[:,4].reshape(-
    1,1), data[:,7].reshape(-1,1))))
125.        T_stp_cool_clus = transform_data(np.hstack((data[:,4].reshape(-
    1,1), data[:,8].reshape(-1,1))))
126.
127.        x_clus = np.hstack((T_out_clus,T_ctrl_clus,Humidity_clus,T_stp_cool_clu
    s))
128.        scaler_clus = MinMaxScaler(feature_range=(0,1))
129.        x_clus = scaler_clus.fit_transform(x_clus)
130.        print("Transformation done: ",datetime.now())
131.
132.        """
133.        ## Clustering method 2 (metadata)
134.        days_flag = False
135.        household_size = []
136.        household_age = []
137.        household_latitude = []
138.        household_longitude = []
139.        household_latitude_full = []
140.        household_longitude_full = []
141.        households_ids = np.unique(clean_household)
142.
143.        ## Read metadata
144.        with h5py.File('/Users/user/Documents/Project/hourlyData.h5', 'r') as f
    :
145.            file_names = list(f.keys())
146.            for i in households_ids:
```

```
147.                household_size = np.append(household_size, f['/'+file_names[int
    (i)]].attrs['Floor Area [ft2]'])
148.                household_age = np.append(household_age, f['/'+file_names[int(i
    )]].attrs['Age of Home [years]'])
149.        household_size = household_size.reshape(-1,1)
150.        household_age = household_age.reshape(-1,1)
151.
152.        ## Get household latitude and longitude
153.        with open('/Users/user/Documents/Project/Metadata_full.txt', 'r') as ff
    :
154.            household_latitude_full.append([float(i) for i in tuple(zip(*[line.
    split(";") for line in ff]))[4]])
155.            ff.seek(0)
156.            household_longitude_full.append([float(i) for i in tuple(zip(*[line
    .split(";") for line in ff]))[5]])
157.        household_latitude_full = np.array(household_latitude_full)
158.        household_longitude_full = np.array(household_longitude_full)
159.        household_latitude_full = household_latitude_full.reshape(-1,1)
160.        household_longitude_full = household_longitude_full.reshape(-1,1)
161.
162.        ## Get latitude and longitude only of selected household ids
163.        for i in households_ids:
164.            household_latitude = np.append(household_latitude, household_latitu
    de_full[int(i)])
165.            household_longitude = np.append(household_longitude, household_long
    itude_full[int(i)])
166.        household_latitude = household_latitude.reshape(-1,1)
167.        household_longitude = household_longitude.reshape(-1,1)
168.
169.        x_clus = np.hstack((household_latitude, household_longitude, household_
    size, household_age))
170.        scaler_clus = MinMaxScaler(feature_range=(0,1))
171.        x_clus = scaler_clus.fit_transform(x_clus)
172.        """
173.
174.        ## Clustering model
175.        kmmodel = KMeans(n_clusters = clusters, random_state = 0).fit(x_clus)
176.        labels = kmmodel.labels_
177.
178.        ## Plot KMeans Inertia to select number of clusters
179.        #number_clusters_optimization(x_clus)
180.
181.        print("Clustering: ",datetime.now())
182.
183.        #######################################################################
    ###
184.
185.
186.
187.        ############################# NEURAL NETWORK ##########################
    ###
188.
189.        ## Inverse data transformation for clustering labels
190.        x_labels = []
191.        x_labels = transform_data_inv(labels,x,only_labels = True, by_days = da
    ys_flag)
192.        x_labels = x_labels.reshape(len(x_labels),1)
193.        print("x_labels: ",x_labels.shape)
194.
```

```
195.        ## Normalization
196.        scaler_x = MinMaxScaler(feature_range=(0, 1))
197.        scaler_y = MinMaxScaler(feature_range=(0, 1))
198.        scaled_x = scaler_x.fit_transform(x)
199.        scaled_y = scaler_y.fit_transform(y.reshape(-1,1))
200.
201.        ## Supervised learning
202.        reframed_x = supervised_format(scaled_x, look_back, 1)
203.        reframed_y = supervised_format(scaled_y, look_back, 1)
204.        reframed_y.drop(reframed_y.columns[0:-1],axis=1,inplace=True)
205.        print(reframed_x.head())
206.        print(reframed_y.head())
207.
208.        ## Split by cluster
209.        for i in range(0,clusters):
210.            exec("x_%s = copy.copy(x[np.where(x_labels[:] == %s)[0],:])" %(i,i)
    )
211.            exec("x_%s = scaler_x.fit_transform(x_%s)" %(i,i))
212.            exec("y_%s = copy.copy(y[np.where(x_labels[:] == %s)[0]])" %(i,i))

213.            exec("y_%s = scaler_y.fit_transform(y_%s.reshape(-1,1))" %(i,i))
214.
215.        ## Split into train / test sets
216.        for i in range(0,clusters):
217.            n_train_hours = int(len(eval("x_%s[:,0]"%(i)))*0.7)
218.            exec("train_X_%s = x_%s[:n_train_hours,:]" %(i,i))
219.            exec("test_X_%s = x_%s[n_train_hours:,:]" %(i,i))
220.            exec("train_y_%s = y_%s[:n_train_hours]" %(i,i))
221.            exec("test_y_%s = y_%s[n_train_hours:]" %(i,i))
222.            print("Cluster ",i," ----
    >",eval("train_X_%s" %(i)).shape, eval("test_X_%s" %(i)).shape, eval("train_y_
    %s" %(i)).shape, eval("test_y_%s" %(i)).shape)
223.
224.        ## Reshape to required 3D format [samples, timesteps, features]
225.        for i in range(0,clusters):
226.            exec("train_X_%s = train_X_%s.reshape((train_X_%s.shape[0], 1, trai
    n_X_%s.shape[1]))" %(i,i,i,i))
227.            exec("test_X_%s = test_X_%s.reshape((test_X_%s.shape[0], 1, test_X_
    %s.shape[1]))" %(i,i,i,i))
228.
229.        ## LSTM model
230.        for i in range(0,clusters):
231.            exec("model_%s = Sequential()" %(i))
232.            exec("model_%s.add(LSTM(neurons, input_shape=(train_X_%s.shape[1],
    train_X_%s.shape[2])))" %(i,i,i))
233.            exec("model_%s.add(Dense(1))" %(i))
234.            exec("model_%s.compile(loss='mean_squared_error', optimizer='adam')
    " %(i))
235.            exec("history_%s = model_%s.fit(train_X_%s, train_y_%s, epochs=epoc
    hs, batch_size=batch_size, validation_data=(test_X_%s, test_y_%s), verbose=2,
    shuffle=False)" %(i,i,i,i,i,i))
236.
237.        ## Plot training
238.        for i in range(0,clusters):
239.            fig, ax = plt.subplots()
240.            exec("ax.plot(history_%s.history['loss'], label='train')" %(i))
241.            exec("ax.plot(history_%s.history['val_loss'], label='test')" %(i))

242.            ax.legend()
```

```
243.            plot_title = "Cluster: " + str(i)
244.            plt.title(plot_title)
245.            ax.set_xlabel("Epochs")
246.            ax.set_ylabel("Error")
247.            plt.show()
248.
249.        ## Predict
250.        for i in range(0,clusters):
251.            exec("yhat_%s = model_%s.predict(test_X_%s)" %(i,i,i))
252.            exec("test_X_%s = test_X_%s.reshape((test_X_%s.shape[0], test_X_%s.
    shape[2]))" %(i,i,i,i))
253.
254.        ## Invert prediction
255.        for i in range(0,clusters):
256.            exec("inv_yhat_%s = yhat_%s" %(i,i))
257.            exec("inv_yhat_%s = scaler_y.inverse_transform(inv_yhat_%s)" %(i,i)
    )
258.
259.        ## Invert scaling (actual)
260.        for i in range(0,clusters):
261.            exec("test_y_%s = test_y_%s.reshape((len(test_y_%s), 1))" %(i,i,i))

262.            exec("inv_y_%s = test_y_%s" %(i,i))
263.            exec("inv_y_%s = scaler_y.inverse_transform(inv_y_%s)" %(i,i))
264.
265.        print("--------------------------------------------")
266.        ## RMSE
267.        for i in range(0,clusters):
268.            exec("rmse_%s = sqrt(mean_squared_error(inv_y_%s, inv_yhat_%s))" %(
    i,i,i))
269.            print("Test RMSE cluster ",i," :", eval("rmse_%s" %(i)))
270.
271.        ## Plot prediction
272.        for i in range(0,clusters):
273.            fig, ax = plt.subplots()
274.            exec("ax.plot(inv_y_%s, label='value')" %(i))
275.            exec("ax.plot(inv_yhat_%s, label='prediction')" %(i))
276.            ax.legend()
277.            plot_title_2 = "Cluster: " + str(i)
278.            plt.title(plot_title_2)
279.            ax.set_xlabel("Periods [h]")
280.            ax.set_ylabel("Temperature [ºC]")
281.            plt.show()
282.
283.        ## Show model summary
284.        #model.summary()
285.
286.        #######################################################################
    ###
287.
288.
289.        """
290.        ####################### PREDICT CENTROIDS METHOD 1 ####################
    ##
291.
292.        # Get centroids
293.        centroids = kmmodel.cluster_centers_
294.        centroids = np.array(centroids)
295.
```

```
296.        print("---------------------------------------------")
297.        print(centroids.shape)
298.        np.savetxt(BASE_DIR+"centroids_1_"+str(datetime.now().strftime('%d%m%y-
    %H%M')),centroids,fmt = '%s',delimiter=",")
299.
300.        # get test x and test y out of centroids
301.        for i in range(0,clusters):
302.            exec("test1_x_%s = np.hstack((centroids[i,:24].reshape(-
    1,1) , centroids[i,24:48].reshape(-1,1) , centroids[i,48:72].reshape(-
    1,1)))" %(i))
303.            exec("test1_y_%s = copy.copy(centroids[i,72:].reshape(-
    1,1))" %(i))
304.
305.            # in case that T_stp (y) is not part of the input
306.            #exec("test1_y_%s = np.zeros((24,1))" %(i))
307.            #y_days = int(len(eval("y_%s[:,0]" %(i)))/24)
308.            #for j in range(0,y_days):
309.            #    for k in range(0,24):
310.            #        exec("test1_y_%s[k,0] += y_%s[24*j+k,0]/y_days" %(i,i))
311.
312.            exec("np.savetxt(BASE_DIR+\"x_test1_clus%s_\"+str(datetime.now().st
    rftime('%%d%%m%%y-
    %%H%%M')),eval(\"test1_x_%s\"),fmt = '%%s',delimiter=\",\")" %(i,i))
313.            exec("np.savetxt(BASE_DIR+\"y_test1_clus%s_\"+str(datetime.now().st
    rftime('%%d%%m%%y-
    %%H%%M')),eval(\"test1_y_%s\"),fmt = '%%s',delimiter=\",\")" %(i,i))
314.
315.        # make a prediction
316.        for i in range(0,clusters):
317.            exec("test1_x_%s = test1_x_%s.reshape((test1_x_%s.shape[0], 1, test
    1_x_%s.shape[1]))" %(i,i,i,i))
318.            exec("yhat1_%s = model_%s.predict(test1_x_%s)" %(i,i,i))
319.            exec("test1_x_%s = test1_x_%s.reshape((test1_x_%s.shape[0], test1_x
    _%s.shape[2]))" %(i,i,i,i))
320.
321.        # invert scaling for forecast
322.        for i in range(0,clusters):
323.            exec("inv1_yhat_%s = yhat1_%s" %(i,i))
324.            exec("inv1_yhat_%s = scaler_y.inverse_transform(inv1_yhat_%s)" %(i,
    i))
325.
326.        # invert scaling for actual
327.        for i in range(0,clusters):
328.            exec("test1_y_%s = test1_y_%s.reshape((len(test1_y_%s), 1))" %(i,i,
    i))
329.            exec("inv1_y_%s = test1_y_%s" %(i,i))
330.            exec("inv1_y_%s = scaler_y.inverse_transform(inv1_y_%s)" %(i,i))
331.
332.        print("---------------------------------------------")
333.        # calculate RMSE
334.        for i in range(0,clusters):
335.            exec("rmse1_%s = sqrt(mean_squared_error(inv1_y_%s, inv1_yhat_%s))"
     %(i,i,i))
336.            print("Test RMSE for centroid of cluster ",i," :", eval("rmse1_%s"
    %(i)))
337.
338.        # plot predictions
339.        for i in range(0,clusters):
340.            plt.figure()
```

```
341.            exec("plt.plot(inv1_y_%s)" %(i))
342.            exec("plt.plot(inv1_yhat_%s)" %(i))
343.            plot_title_3 = "Prediction for centroid of cluster: " + str(i)
344.            plt.title(plot_title_3)
345.            plt.show()
346.
347.
348.        ########################################################################
    ###
349.        """
350.
351.
352.
353.        """
354.        ####################### PREDICT CENTROIDS METHOD 2 ####################
    ##
355.
356.        centroids = kmmodel.cluster_centers_
357.        centroids = np.array(centroids)
358.        labels = labels.reshape(-1,1)
359.
360.        distance = []
361.
362.        for i in range(0,len(labels)):
363.            sum_dis = 0
364.            for j in range(0,len(x_clus[0,:])):
365.                sum_dis = sum_dis + (x_clus[i,j]-centroids[labels[i],j])**2
366.            distance = np.append(distance, sqrt(sum_dis))
367.
368.        distance = distance.reshape(-1,1)
369.        x_distance = transform_data_inv(distance,x,only_labels = True, by_days
    = False)
370.        x_distance = x_distance.reshape(-1,1)
371.
372.        for i in range(0,clusters):
373.            exec("x_distance_%s = copy.copy(x_distance[np.where(x_labels[:] ==
    %s)])" %(i,i))
374.
375.        print("---------------------------------------------")
376.        for i in range(0,clusters):
377.            exec("min_dist_%s = copy.copy(distance[np.where(labels == i)].min()
    )" %(i))
378.            print(i, "   ", eval("min_dist_%s" %(i)))
379.
380.        for i in range(0,clusters):
381.            exec("test2_x_%s = copy.copy(x_%s[np.where(x_distance_%s == min_dis
    t_%s)])" %(i,i,i,i))
382.            exec("test2_y_%s = copy.copy(y_%s[np.where(x_distance_%s == min_dis
    t_%s)])" %(i,i,i,i))
383.
384.        # make a prediction
385.        for i in range(0,clusters):
386.            exec("test2_x_%s = copy.copy(x_%s[np.where(x_distance_%s == min_dis
    t_%s)])" %(i,i,i,i))
387.            exec("test2_x_%s = test2_x_%s.reshape((test2_x_%s.shape[0], 1, test
    2_x_%s.shape[1]))" %(i,i,i,i))
388.            exec("yhat2_%s = model_%s.predict(test2_x_%s)" %(i,i,i))
389.            exec("test2_x_%s = test2_x_%s.reshape((test2_x_%s.shape[0], test2_x
    _%s.shape[2]))" %(i,i,i,i))
```

```
390.
391.        # invert scaling for forecast
392.        for i in range(0,clusters):
393.            exec("inv2_yhat_%s = yhat2_%s" %(i,i))
394.            exec("inv2_yhat_%s = scaler_y.inverse_transform(inv2_yhat_%s)" %(i,
    i))
395.
396.        # invert scaling for actual
397.        for i in range(0,clusters):
398.            exec("test2_y_%s = test2_y_%s.reshape((len(test2_y_%s), 1))" %(i,i,
    i))
399.            exec("inv2_y_%s = test2_y_%s" %(i,i))
400.            exec("inv2_y_%s = scaler_y.inverse_transform(inv2_y_%s)" %(i,i))
401.
402.        print("---------------------------------------------")
403.        # calculate RMSE
404.        for i in range(0,clusters):
405.            exec("rmse2_%s = sqrt(mean_squared_error(inv2_y_%s, inv2_yhat_%s))"
     %(i,i,i))
406.            print("Test RMSE for centroid of cluster ",i," :", eval("rmse2_%s"
    %(i)))
407.
408.        # plot predictions
409.        for i in range(0,clusters):
410.            plt.figure()
411.            exec("plt.plot(inv2_y_%s)" %(i))
412.            exec("plt.plot(inv2_yhat_%s)" %(i))
413.            plot_title_3 = "Prediction for closets to centroid of cluster: " +
    str(i)
414.            plt.title(plot_title_3)
415.            plt.show()
416.
417.
418.        #######################################################################
    ###
419.        """
```

D. LSTM Neural Network with Setpoint variable as input

```
1.  import h5py
2.  import numpy as np
3.
4.  import math
5.
6.  import pandas as pd
7.  import matplotlib.pyplot as plt
8.
9.  import plotly
10. import plotly.plotly as py
11. import plotly.graph_objs as go
12.
13. from sklearn.linear_model import LogisticRegression
14. from sklearn import metrics
15. from sklearn.cluster import KMeans
16.
17. from datetime import datetime
18. import copy
19. from scipy.constants import F2C
20.
21. from pandas import read_csv
22. from keras.models import Sequential
23. from keras.layers import Dense
24. from keras.layers import LSTM
25. from sklearn.preprocessing import MinMaxScaler
26. from sklearn.metrics import mean_squared_error
27.
28. import sys
29. sys.path.insert(0, '/Users/user/Documents/Project/IP_library')
30. from Fill_one_whole_household import fill_one_whole_household
31. from Transform_pre_clean import transform_pre_clean
32. from Transform_post_clean import transform_post_clean
33. from Supervised_format import supervised_format
34. from Transform_data import transform_data
35. from Transform_data_inv_1D import transform_data_inv_1D
36. from Get_data import get_data
37.
38. from math import sqrt
39. from numpy import concatenate
40. from matplotlib import pyplot as plt
41. from pandas import read_csv
42. from pandas import DataFrame
43. from pandas import concat
44. from sklearn.preprocessing import MinMaxScaler
45. from sklearn.preprocessing import LabelEncoder
46. from sklearn.metrics import mean_squared_error
47. from keras.models import Sequential
48. from keras.layers import Dense
49. from keras.layers import LSTM
50.
51. ##############################################################################
    ##############################################################################
    ######
52. ##############################################################################
    ##############################################################################
    ######
```

```
53. ##############################################################################
    ######
54. ############################################
                                        ########################################
    ######
55. ############################################                    NEURAL NETWORK
    MODEL (PAST AS INPUT)               ########################################
    ######
56. ############################################
                                        ########################################
    ######
57. ##############################################################################
    ######
58. ##############################################################################
    ######
59. ##############################################################################
    ######
60.
61. """
62. This script has the objective of running a LSTM neural network given the
63. past information of the output setpoint as input to predict how either
64. the cooling or heating setpoint variables behave.
65.
66. First, the "get data" section reads, cleans and stores the information
67. contained in the hd5f database to appropriated variables.
68.
69. Once this is done, the data is clustered according to the location of
70. the household they belong to, clustering each user with nearby households
71. that will likely experience a similar weather. The number of clusters is
72. defined at the beginning of the script.
73.
74. Finally, a different neural network is trained for each cluster. Given that
75. the data was transformed into day format for the clustering analysis, the
76. first step in this section is to reverse this transformation for the
77. clustering labels to have the correct size. Each variable is then split
78. into training / test set before running the regression model.
79. """
80.
81. #########################
82. households = 1000    # Number of households to extract from database
83. st_household = 0     # Starting household of DB to extract
84. st_year = 2015       # Starting year to extract for each household
85. fin_year = 2016      # Final year to extract for each household
86. st_month = 1         # Starting month to extract for each household
87. fin_month = 12       # Final month to extract for each houseshold
88.
89. clusters = 1         # Number of clusters for clustering analysis
90. neurons = 100        # Number of neurons for LSTM model
91. epochs = 5           # Numbber of epochs for LSTM model
92. batch_size = 70      # Batch size for LSTM model
93. look_back = 72       # Number of past hours given as input for LSTM model
94.
95. BASE_DIR = '/Users/user/Documents/Project/Exports/'
96. #########################
97.
```

```
98.
99. print(np.__version__)
100.
101.        ################################ GET DATA #############################
    ###
102.
103.        start = datetime.now()
104.        print("start: ",start)
105.        data = []
106.        data = get_data('npy', households, st_household, st_year, fin_year, st_
    month, fin_month)
107.        print(data.shape)
108.        print("Time elapsed: ", datetime.now()-start)
109.
110.        ## Fahrenheit to Celsius
111.        data[:,5] = copy.copy(F2C(data[:,5]))
112.        data[:,6] = copy.copy(F2C(data[:,6]))
113.        data[:,8] = copy.copy(F2C(data[:,8]))
114.        data[:,9] = copy.copy(F2C(data[:,9]))
115.
116.        ## Select x and y from data
117.        x = copy.copy(data[:,5])
118.        y = copy.copy(data[:,5])
119.
120.        print("Data acquisition completed: ",datetime.now(),"\n")
121.
122.        #######################################################################
    ###
123.
124.
125.        ############################### CLUSTERING ############################
    ###
126.
127.        T_out_clus = []
128.        T_ctrl_clus = []
129.        Humidity_clus = []
130.        x_clus = []
131.
132.        ## Clustering method 2 (metadata)
133.        household_size = []
134.        household_age = []
135.        household_latitude = []
136.        household_longitude = []
137.        household_latitude_full = []
138.        household_longitude_full = []
139.        households_ids = np.unique(data[:,0])
140.
141.        ## Read metadata
142.        with h5py.File('/Users/user/Documents/Project/hourlyData.h5', 'r') as f
    :
143.            file_names = list(f.keys())
144.            for i in households_ids:
145.                household_size = np.append(household_size, f['/'+file_names[int
    (i)]].attrs['Floor Area [ft2]'])
146.                household_age = np.append(household_age, f['/'+file_names[int(i
    )]].attrs['Age of Home [years]'])
147.        household_size = household_size.reshape(-1,1)
148.        household_age = household_age.reshape(-1,1)
149.
```

```
150.        ## Get household latitude and longitude
151.        with open('/Users/user/Documents/Project/Metadata_full.txt', 'r') as ff
    :
152.            household_latitude_full.append([float(i) for i in tuple(zip(*[line.
    split(";") for line in ff]))[4]])
153.            ff.seek(0)
154.            household_longitude_full.append([float(i) for i in tuple(zip(*[line
    .split(";") for line in ff]))[5]])
155.        household_latitude_full = np.array(household_latitude_full)
156.        household_longitude_full = np.array(household_longitude_full)
157.        household_latitude_full = household_latitude_full.reshape(-1,1)
158.        household_longitude_full = household_longitude_full.reshape(-1,1)
159.
160.        ## Get latitude and longitude only of selected household ids
161.        for i in households_ids:
162.            household_latitude = np.append(household_latitude, household_latitu
    de_full[int(i)])
163.            household_longitude = np.append(household_longitude, household_long
    itude_full[int(i)])
164.        household_latitude = household_latitude.reshape(-1,1)
165.        household_longitude = household_longitude.reshape(-1,1)
166.
167.        x_clus = np.hstack((household_latitude, household_longitude, household_
    size, household_age))
168.        scaler_clus = MinMaxScaler(feature_range=(0,1))
169.        x_clus = scaler_clus.fit_transform(x_clus)
170.
171.        ## Clustering model
172.        kmmodel = KMeans(n_clusters = clusters, random_state = 0).fit(x_clus)
173.        labels = kmmodel.labels_
174.
175.        print("Clustering: ",datetime.now())
176.
177.        ########################################################################
    ###
178.
179.
180.
181.        ############################# NEURAL NETWORK ##########################
    ###
182.
183.        ## Inverse data transformation for clustering labels
184.        x_labels = []
185.        x_labels = transform_data_inv_1D(labels,x,only_labels = True, by_days =
     False)
186.        x_labels = x_labels.reshape(len(x_labels),1)
187.        print("x_labels: ",x_labels.shape)
188.
189.        ## Normalization
190.        scaler_x = MinMaxScaler(feature_range=(0, 1))
191.        scaler_y = MinMaxScaler(feature_range=(0, 1))
192.        scaled_x = scaler_x.fit_transform(x.reshape(-1,1))
193.        scaled_y = scaler_y.fit_transform(y.reshape(-1,1))
194.
195.        ## Supervised learning
196.        data_supervised = supervised_format(scaled_x, look_back, 1)
197.        reframed_x = copy.copy(data_supervised)
198.        reframed_y = copy.copy(data_supervised)
199.        reframed_x.drop(reframed_x.columns[-1],axis=1,inplace=True)
```

```
200.        reframed_y.drop(reframed_y.columns[0:-1],axis=1,inplace=True)
201.        print(reframed_x.head())
202.        print(reframed_y.head())
203.
204.        ## Split by cluster
205.        for i in range(0,clusters):
206.            exec("x_%s = copy.copy(x[np.where(x_labels[:] == %s)[0]])" %(i,i))

207.            exec("x_%s = scaler_x.fit_transform(x_%s.reshape(-1,1))" %(i,i))
208.            exec("y_%s = copy.copy(y[np.where(x_labels[:] == %s)[0]])" %(i,i))

209.            exec("y_%s = scaler_y.fit_transform(y_%s.reshape(-1,1))" %(i,i))
210.            exec("np.savetxt(\"/Users/user/Documents/Project/Exports/x_clus%s_\
    "+str(datetime.now().strftime('%%d%%m%%y-
    %%H%%M')),eval(\"x_%s\"),fmt = '%%s',delimiter=\",\")" %(i,i))
211.
212.        ## Split into train / test sets
213.        for i in range(0,clusters):
214.            n_train_hours = int(len(eval("x_%s[:,0]"%(i)))*0.7)
215.            exec("train_X_%s = x_%s[:n_train_hours]" %(i,i))
216.            exec("test_X_%s = x_%s[n_train_hours:]" %(i,i))
217.            exec("train_y_%s = y_%s[:n_train_hours]" %(i,i))
218.            exec("test_y_%s = y_%s[n_train_hours:]" %(i,i))
219.            print("Cluster ",i," ----
    >",eval("train_X_%s" %(i)).shape, eval("test_X_%s" %(i)).shape, eval("train_y_
    %s" %(i)).shape, eval("test_y_%s" %(i)).shape)
220.
221.        ## Reshape to required 3D format [samples, timesteps, features]
222.        for i in range(0,clusters):
223.            exec("train_X_%s = train_X_%s.reshape((train_X_%s.shape[0], 1, trai
    n_X_%s.shape[1]))" %(i,i,i,i))
224.            exec("test_X_%s = test_X_%s.reshape((test_X_%s.shape[0], 1, test_X_
    %s.shape[1]))" %(i,i,i,i))
225.
226.        ## LSTM model
227.        for i in range(0,clusters):
228.            exec("model_%s = Sequential()" %(i))
229.            exec("model_%s.add(LSTM(neurons, input_shape=(train_X_%s.shape[1],
    train_X_%s.shape[2])))" %(i,i,i))
230.            exec("model_%s.add(Dense(1))" %(i))
231.            exec("model_%s.compile(loss='mean_squared_error', optimizer='adam')
    " %(i))
232.            exec("history_%s = model_%s.fit(train_X_%s, train_y_%s, epochs=epoc
    hs, batch_size=batch_size, validation_data=(test_X_%s, test_y_%s), verbose=2,
    shuffle=False)" %(i,i,i,i,i,i))
233.
234.        ## Plot training
235.        for i in range(0,clusters):
236.            exec("plt.plot(history_%s.history['loss'], label='train')" %(i))
237.            exec("plt.plot(history_%s.history['val_loss'], label='test')" %(i))

238.            plt.legend()
239.            plot_title = "Cluster: " + str(i)
240.            plt.title(plot_title)
241.            plt.show()
242.
243.        ## Predict
244.        for i in range(0,clusters):
245.            exec("yhat_%s = model_%s.predict(test_X_%s)" %(i,i,i))
```

```
246.            exec("test_X_%s = test_X_%s.reshape((test_X_%s.shape[0], test_X_%s.
    shape[2]))" %(i,i,i,i))
247.
248.        ## Invert prediction
249.        for i in range(0,clusters):
250.            exec("inv_yhat_%s = yhat_%s" %(i,i))
251.            exec("inv_yhat_%s = scaler_y.inverse_transform(inv_yhat_%s)" %(i,i)
    )
252.
253.        ## Invert scaling (actual)
254.        for i in range(0,clusters):
255.            exec("test_y_%s = test_y_%s.reshape((len(test_y_%s), 1))" %(i,i,i))

256.            exec("inv_y_%s = test_y_%s" %(i,i))
257.            exec("inv_y_%s = scaler_y.inverse_transform(inv_y_%s)" %(i,i))
258.
259.        print("---------------------------------------------")
260.        ## RMSE
261.        for i in range(0,clusters):
262.            exec("rmse_%s = sqrt(mean_squared_error(inv_y_%s, inv_yhat_%s))" %(
    i,i,i))
263.            print("Test RMSE cluster ",i," :", eval("rmse_%s" %(i)))
264.
265.        ## Plot prediction
266.        for i in range(0,clusters):
267.            plt.figure()
268.            exec("plt.plot(inv_y_%s)" %(i))
269.            exec("plt.plot(inv_yhat_%s)" %(i))
270.            plot_title_2 = "Cluster: " + str(i)
271.            plt.title(plot_title_2)
272.            plt.show()
273.
274.        ## Show model summary
275.        #model.summary()
276.
277.        ########################################################################
    ###
```

E. Household Setpoint Behavior Analysis

```
1.  import h5py
2.  import numpy as np
3.
4.  from math import sqrt
5.  from numpy import concatenate
6.  from matplotlib import pyplot as plt
7.  from matplotlib.lines import Line2D
8.  from pandas import read_csv
9.  from pandas import DataFrame
10. from pandas import concat
11.
12. from sklearn.preprocessing import MinMaxScaler
13. from sklearn.preprocessing import LabelEncoder
14. from sklearn.metrics import mean_squared_error
15. from sklearn.cluster import KMeans
16. from sklearn.decomposition import FactorAnalysis
17. from sklearn.decomposition import PCA
18.
19. from keras.models import Sequential
20. from keras.layers import Dense
21. from keras.layers import LSTM
22.
23. from datetime import datetime
24.
25. import sys
26. sys.path.insert(0, '/Users/user/Documents/Project/IP_library')
27. from Fill_one_whole_household import fill_one_whole_household
28. from Fill_one_whole_household_v2 import fill_one_whole_household_v2
29. from Transform_pre_clean import transform_pre_clean
30. from Transform_post_clean import transform_post_clean
31. from Transform_data import transform_data
32. from Transform_data_inv import transform_data_inv
33. from Number_clusters_optimization import number_clusters_optimization
34. from Get_data import get_data
35.
36. import copy
37.
38. def centroidnp(arr):
39.     length = arr.shape[0]
40.     sum_x = np.sum(arr[:, 0])
41.     sum_y = np.sum(arr[:, 1])
42.     return sum_x/length, sum_y/length
43.
44. ###############################################################################
    ###############################################################################
    ######
45. ###############################################################################
    ###############################################################################
    ######
46. ###############################################################################
    ###############################################################################
    ######
47. ###########################################
                                           #######################################
    ######
```

```
48. ############################################                      HOUSEHOLD SET
    POINT BEHAVIOR ANALYSIS                ########################################
    ######
49. ############################################
                                           ########################################
    ######
50. ################################################################################
    ################################################################################
    ######
51. ################################################################################
    ################################################################################
    ######
52. ################################################################################
    ################################################################################
    ######
53.
54. """
55. This script has the objective of running a PCA / Factor Analysis to
56. test the hypothesis of equal setpoint behaviors for households in
57. different climates.
58.
59. First, the "get data" section reads, cleans and stores the information
60. contained in the hd5f database to appropriated variables.
61.
62. Once this is done, the data is clustered according to the location of
63. the household they belong to, clustering each user with nearby households
64. that will likely experience a similar weather. The number of clusters is
65. defined at the beginning of the script.
66.
67. Finally, the PCA / Factor Analysis is done using a loop in order to get
68. the average centroids for the time period selected. The Factor Analysis
69. section is commented since the script works best when only one of the
70. two sections is uncommeted at a time.
71. """
72.
73. #########################
74. households = 6739    # Number of households to extract from database
75. st_household = 0     # Starting household of DB to extract
76. st_year = 2015       # Starting year to extract for each household
77. fin_year = 2016      # Final year to extract for each household
78. st_month = 1         # Starting month to extract for each household
79. fin_month = 12       # Final month to extract for each houeshold
80.
81. sel_buildings = []  # Selected buildings for data analysis
82. k_labels = []        # Clustering labels
83. clusters = 12        # Number of clusters
84. BASE_DIR = '/Users/user/Documents/Project/Exports/'
85. LABEL_COLOR_MAP = {0:'r', 1:'plum', 2:'m', 3:'y', 4:'gold', 5:'g', 6:'b', 7:'c
    ', 8:'cyan', 9:'darkolivegreen', 10:'dimgray', 11: 'orange', 12:'k', 13:'navy'
    }   # Color map for plots
86. month_days_2015 = {1 : 31, 2 : 28, 3 : 31, 4 : 30, 5 : 31, 6 : 30, 7 : 31, 8 :
     31, 9 : 30, 10 : 31, 11 : 30, 12 : 31 }  # Days per month in 2015
87. month_days_2016 = {1 : 31, 2 : 29, 3 : 31, 4 : 30, 5 : 31, 6 : 30, 7 : 31, 8 :
     31, 9 : 30, 10 : 31, 11 : 30, 12 : 31 }  # Days per month in 2016
88. #########################
89.
90.
91. ################################ GET DATA ################################
92.
```

```
93. start = datetime.now()
94. print("start: ",start)
95. data = []
96. data = get_data('npy', households, st_household, st_year, fin_year, st_month,
    fin_month)
97. print(data.shape)
98. print("Time elapsed: ", datetime.now()-start)
99.
100.        ## Fahrenheit to Celsius
101.        data[:,5] = copy.copy(F2C(data[:,5]))
102.        data[:,6] = copy.copy(F2C(data[:,6]))
103.        data[:,8] = copy.copy(F2C(data[:,8]))
104.        data[:,9] = copy.copy(F2C(data[:,9]))
105.
106.        print("Data acquisition completed: ",datetime.now(),"\n")
107.
108.        #######################################################################
    ###
109.
110.
111.        ############################### CLUSTERING ############################
    ###
112.
113.        T_out_clus = []
114.        T_ctrl_clus = []
115.        T_stp_cool_clus = []
116.        Humidity_clus = []
117.        x_clus = []
118.
119.        ## Clustering method 2 (metadata)
120.        days_flag = False
121.        household_latitude = []
122.        household_longitude = []
123.        households_ids = np.unique(data[:,0])
124.        print(len(households_ids))
125.        household_latitude_full = []
126.        household_longitude_full = []
127.
128.        ## Read metadata
129.        with open('/Users/user/Documents/Project/Metadata_full_2.txt', 'r') as
    ff:
130.            household_latitude_full.append([float(i) for i in tuple(zip(*[line.
    split(";") for line in ff]))[4]])
131.            ff.seek(0)
132.            household_longitude_full.append([float(i) for i in tuple(zip(*[line
    .split(";") for line in ff]))[5]])
133.        household_latitude_full = np.array(household_latitude_full)
134.        household_longitude_full = np.array(household_longitude_full)
135.        household_latitude_full = household_latitude_full.reshape(-1,1)
136.        household_longitude_full = household_longitude_full.reshape(-1,1)
137.
138.        print("latitude, longitude full shape: ", household_latitude_full.shape
    , household_longitude_full.shape)
139.
140.        ## Household Selection
141.        for i in range(0,len(household_longitude_full)):
142.            if(household_longitude_full[i] > -
    130 and household_longitude_full[i] < -
    50 and household_latitude_full[i] > 20 and household_latitude_full[i] < 60):
```

```
143.                sel_buildings = np.append(sel_buildings, i)
144. 
145.        sel_buildings = np.array([int(i) for i in sel_buildings])
146.        data = data[np.where(np.in1d(data[:,0].astype(int), sel_buildings[:].as
    type(int)))]
147.        print("data shape: ", data.shape)
148.        print("sel buildings shape: ", sel_buildings.shape)
149. 
150.        household_latitude = []
151.        household_longitude = []
152.        for i in sel_buildings:
153.            household_latitude = np.append(household_latitude, household_latitu
    de_full[int(i)])
154.            household_longitude = np.append(household_longitude, household_long
    itude_full[int(i)])
155.        household_latitude = household_latitude.reshape(-1,1)
156.        household_longitude = household_longitude.reshape(-1,1)
157. 
158.        x_clus = np.hstack((household_latitude, household_longitude))
159.        scaler_clus = MinMaxScaler(feature_range=(0,1))
160.        x_clus = scaler_clus.fit_transform(x_clus)
161. 
162.        ## Clustering model
163.        kmmodel = KMeans(n_clusters = clusters, random_state = 0).fit(x_clus)
164. 
165.        #######################################################################
    ###
166. 
167. 
168.        ######################### PCA / Factor Analysis ######################
    ###
169. 
170.        ## Variable declaration
171.        for i in range(0,clusters):
172.            exec("full_centr_pca_%s = []" %(i))
173.            exec("full_centr_FA_%s = []" %(i))
174. 
175.        ## PCA or FA calculation (loop required to go day by day)
176.        for year in range(st_year,fin_year+1):
177.            for month in range(st_month, fin_month+1):
178.                exec("days = month_days_%s[month]" %(year))
179.                for day in range(1,eval("days")+1):
180.                    print(year, month," day ", day, " of ",eval("days"))
181. 
182.                    ## Get today's data
183.                    data_sel = []
184.                    data_sel = data[np.where(data[:,1] == year)]
185.                    data_sel = data_sel[np.where(data_sel[:,2] == month)]
186.                    data_sel = data_sel[np.where(data_sel[:,3] == day)]
187.                    sel_buildings = np.unique(data_sel[:,0])
188.                    print("sel buildings: ",sel_buildings.shape)
189. 
190.                    ## Get latitude and longitude of buildings with data from t
    oday
191.                    day_latitude = []
192.                    day_longitude = []
193.                    for i in sel_buildings:
194.                        day_latitude = np.append(day_latitude, household_latitu
    de_full[int(i)])
```

```
195.                        day_longitude = np.append(day_longitude, household_long
    itude_full[int(i)])
196.                    day_latitude = day_latitude.reshape(-1,1)
197.                    day_longitude = day_longitude.reshape(-1,1)
198.
199.                    ## Format model input
200.                    x = np.hstack((day_latitude, day_longitude))
201.                    x = scaler_clus.transform(x)
202.                    labels = kmmodel.predict(x)
203.
204.        """
205.                    ############################### PCA ANALYSIS ##############
    ###############
206.
207.                    pca_flag = True
208.                    T_stp_cool_clus = []
209.                    T_stp_cool_clus = transform_data(np.hstack((data_sel[:,4].r
    eshape(-1,1) , data_sel[:,8].reshape(-1,1))))
210.                    x = copy.copy(T_stp_cool_clus[:,:])
211.
212.                    scaler_x = MinMaxScaler(feature_range=(0,1))
213.                    x = scaler_x.fit_transform(x)
214.                    model_pca = PCA(n_components = 2, copy = True)
215.                    x_pca = model_pca.fit_transform(x)
216.
217.                    ## Centroids
218.                    for i in range(0,clusters):
219.                        exec("centroid_pca_data%s = x_pca[np.in1d(labels,i)]" %
    (i))
220.                        exec("centroid_pca_%s = centroidnp(centroid_pca_data%s)
    " %(i,i))
221.                        exec("full_centr_pca_%s = np.append(full_centr_pca_%s,
    centroid_pca_%s)" %(i,i,i))
222.                    ###########################################################
    ###############
223.        """
224.
225.                    ############################## FACTOR ANALYSIS #############
    ###############
226.
227.                    pca_flag = False
228.                    T_stp_cool_clus = []
229.                    T_stp_cool_clus = transform_data(np.hstack((data_sel[:,4].r
    eshape(-1,1) , data_sel[:,8].reshape(-1,1))))
230.                    x = copy.copy(T_stp_cool_clus[:,:])
231.
232.                    scaler_x = MinMaxScaler(feature_range=(0,1))
233.                    x = scaler_x.fit_transform(x)
234.
235.                    model_FA = FactorAnalysis(n_components = 2, copy = True)
236.                    x_FA = model_FA.fit_transform(x)
237.
238.                    ## Centroids
239.                    for i in range(0,clusters):
240.                        exec("centroid_FA_data%s = x_FA[np.in1d(labels,i)]" %(i
    ))
241.                        exec("centroid_FA_%s = centroidnp(centroid_FA_data%s)"
    %(i,i))
```

```
242.                        exec("full_centr_FA_%s = np.append(full_centr_FA_%s, ce
    ntroid_FA_%s)" %(i,i,i))
243.
244.                    #############################################################
    ###############
245.
246.        ## Plot all Centroids
247.        fig, ax = plt.subplots()
248.        legend_elements = []
249.        label_elements = []
250.        for i in range(0,clusters):
251.            if(pca_flag):
252.                exec("total_FA_%s = centroidnp(centroid_FA_data%s)" %(i,i))
253.                ax.scatter(eval("total_FA_%s[0]" %(i)),eval("total_FA_%s[1]" %(
    i)),color=LABEL_COLOR_MAP[i])
254.            else:
255.                exec("total_pca_%s = centroidnp(centroid_pca_data%s)" %(i,i))
256.                ax.scatter(eval("total_pca_%s[0]" %(i)),eval("total_pca_%s[1]"
    %(i)),color=LABEL_COLOR_MAP[i])
257.        for i in range(0,clusters):
258.            legend_elements = np.append(legend_elements, Line2D([0],[0],marker=
    'o', color=LABEL_COLOR_MAP[i], label='Cluster: '+str(int(i))))
259.            label_elements = np.append(label_elements, 'Cluster: '+str(int(i)))

260.        ax.legend(legend_elements, label_elements)
261.        plt.show()
262.
263.        #######################################################################
    ###
264.
265.
266.        """
267.        ########################### 1-
    DAY PCA ANALYSIS ############################
268.
269.        ### Day ###
270.        year = 2016
271.        month = 6
272.        day = 25
273.        ###########
274.
275.        data_sel = []
276.        data_sel = data[np.where(data[:,1] == year)]
277.        data_sel = data_sel[np.where(data_sel[:,2] == month)]
278.        data_sel = data_sel[np.where(data_sel[:,3] == day)]
279.        sel_buildings = np.unique(data_sel[:,0])
280.        print("sel buildings: ",sel_buildings.shape)
281.
282.        day_latitude = []
283.        day_longitude = []
284.        for i in sel_buildings:
285.            day_latitude = np.append(day_latitude, household_latitude_full[int(
    i)])
286.            day_longitude = np.append(day_longitude, household_longitude_full[i
    nt(i)])
287.        day_latitude = day_latitude.reshape(-1,1)
288.        day_longitude = day_longitude.reshape(-1,1)
289.
290.        x = np.hstack((day_latitude, day_longitude))
```

```
291.        x = scaler_clus.transform(x)
292.
293.        labels_loc = kmmodel.predict(x)
294.
295.        T_stp_cool_clus = []
296.        T_stp_cool_clus = transform_data(np.hstack((data_sel[:,4].reshape(-
    1,1) , data_sel[:,8].reshape(-1,1))))
297.        x = copy.copy(T_stp_cool_clus[:,:])
298.
299.        scaler_x = MinMaxScaler(feature_range=(0,1))
300.        x = scaler_x.fit_transform(x)
301.        model_pca = PCA(n_components = 2, copy = True)
302.
303.        x_pca = model_pca.fit_transform(x)
304.        scaler_pca = MinMaxScaler(feature_range=(0,1))
305.        x_pca_mm = scaler_pca.fit_transform(x_pca)
306.
307.        kmmodel_pca = KMeans(n_clusters = clusters, random_state = 0).fit(x_pca
    _mm)
308.        labels_pca = kmmodel_pca.labels_
309.
310.        # Plot PCA
311.        fig, ax = plt.subplots()
312.        legend_elements = []
313.        label_elements = []
314.        for i in range(0,len(x_pca[:,0])):
315.            cluster = labels_pca[i]
316.            ax.scatter(x_pca[i,0],x_pca[i,1],color=LABEL_COLOR_MAP[cluster])
317.        for i in range(0,clusters):
318.            legend_elements = np.append(legend_elements, Line2D([0],[0],marker=
    'o', color=LABEL_COLOR_MAP[i], label='Cluster: '+str(int(i))))
319.            label_elements = np.append(label_elements, 'Cluster: '+str(int(i)))

320.        ax.legend(legend_elements, label_elements)
321.        plt.show()
322.
323.        print(labels_loc.shape)
324.        print(labels_pca.shape)
325.
326.        from sklearn.metrics import adjusted_rand_score
327.        from sklearn.metrics import adjusted_mutual_info_score
328.
329.        print(adjusted_rand_score(labels_loc, labels_pca))
330.        print(adjusted_mutual_info_score(labels_loc, labels_pca))
331.        #######################################################################
    ###
332.        """
```

F. Library

a) Clean_nan.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import math
5.
6.  import pandas as pd
7.  import matplotlib.pyplot as plt
8.
9.  import datetime
10. import matplotlib.dates as mdates
11. import matplotlib.patches as mpatches
12.
13. import plotly
14. import plotly.plotly as py
15. import plotly.graph_objs as go
16.
17. """
18. Delete rows containing NaN values from a list
19. """
20.
21. def clean_nan(variable_to_clean):
22.
23.     clean_variable = []
24.     clean_variable = variable_to_clean
25.
26.     for i in range(0, len(variable_to_clean)):
27.         if(math.isnan(variable_to_clean[i])):
28.             if(i==0):
29.                 clean_variable[i] = 0
30.             else:
31.                 clean_variable[i] = clean_variable[i-1]
32.
33.
34.     return clean_variable
```

b) Cuantify_change.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import math
5.
6.  import pandas as pd
7.  import matplotlib.pyplot as plt
8.
9.  import datetime
10. import matplotlib.dates as mdates
11. import matplotlib.patches as mpatches
12.
13. import plotly
14. import plotly.plotly as py
15. import plotly.graph_objs as go
16.
17. """
```

```
18. Discretization of a variable (usually setpoint variable)
19. into categories for multinomial regression to work
20.
21. It works by simply comparing the value of the variable at
22. a particular period with the previous period. That value
23. is then grouped into 5 categories (big negative, small
24. negative, zero, small positive and big positive)
25. """
26.
27. def cuantify_change(variable_to_dummy):
28.
29.     dummy = []
30.     dummy = np.zeros(len(variable_to_dummy))
31.
32.     for i in range(0,len(variable_to_dummy)):
33.         if(i == 0):
34.             dummy[i] = 0
35.         else:
36.             dummy[i] = int(np.round(variable_to_dummy[i]) - np.round(variable_
    to_dummy[i-1]))
37.
38.     for i in range(0, len(dummy)):
39.         if(dummy[i] > 0 and dummy[i] < 5):
40.             dummy[i] = 1
41.
42.         if(dummy[i] < 0 and dummy[i] > -5):
43.             dummy[i] = -1
44.
45.         if(dummy[i] > 5):
46.             dummy[i] = 5
47.
48.         if(dummy[i] < -5):
49.             dummy[i] = -5
50.
51.
52.     return dummy
```

c) DB_generator.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import math
5.  from datetime import datetime
6.
7.  import sys
8.  sys.path.insert(0, '/Users/user/Documents/Project/IP_library')
9.  from Fill_one_whole_household_v2 import fill_one_whole_household_v2
10. from Transform_pre_clean import transform_pre_clean
11. from Transform_post_clean import transform_post_clean
12.
13. """
14. Script to generate a simplified version of the hd5f database,
15. containing only the variables that are interesting for the project
16. and erasing all the days that contain NaN values
17. """
18.
19. ## Open file
```

```
20. fileName = '/Users/user/Documents/Project/hourlyData.h5'
21. f = h5py.File(fileName, libver='latest')
22. file_names = list(f.keys())
23.
24. #########################
25. households = 6739   # Number of households to extract from database
26. st_household = 0    # Starting household of DB to extract
27. st_year = 2015      # Starting year to extract for each household
28. fin_year = 2016     # Final year to extract for each household
29. st_month = 1        # Starting month to extract for each household
30. fin_month = 12      # Final month to extract for each houeshold
31. BASE_DIR = '/Users/user/Documents/Project/Exports/'    # Base directory where
    the file will be saved
32. #########################
33.
34.
35. ################################ GET DATA ################################
36.
37. print("start: ",datetime.now())
38. data = []
39. data = fill_one_whole_household_v2(households, st_household, st_year, fin_year
    , st_month, fin_month)
40. data = transform_pre_clean(data)
41. data = data[~np.isnan(data).any(axis=1)]
42. data = transform_post_clean(data)
43. data = data[np.argsort(data[:,len(data[0,:])-1]),:]
44. np.save("/Users/user/Documents/Project/Exports/data_DDBB_npy_"+str(datetime.no
    w().strftime('%d%m%y-%H%M')),data)
45. np.savetxt("/Users/user/Documents/Project/Exports/data_DDBB_txt_"+str(datetime
    .now().strftime('%d%m%y-%H%M')),data,fmt = '%s',delimiter=",")
46.
47. print("Data acquisition completed: ",datetime.now(),"\n")
48.
49. ##########################################################################
50.
51. f.close()
```

d) Detect_change.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import math
5.
6.  import pandas as pd
7.  import matplotlib.pyplot as plt
8.
9.  import datetime
10. import matplotlib.dates as mdates
11. import matplotlib.patches as mpatches
12.
13. import plotly
14. import plotly.plotly as py
15. import plotly.graph_objs as go
16.
17. """
18. Simplified version of "Cuantify_change" in which
19. the given variable is simply split into two
```

```
20. categories: change or no change
21.
22. Change = variable[t] - variable[t-1]
23. """
24.
25. def detect_change(variable_to_dummy):
26.
27.     dummy = []
28.     dummy = variable_to_dummy
29.
30.     for i in range(0, len(variable_to_dummy)):
31.         if(i == len(variable_to_dummy)-1):
32.             dummy[i] = 0
33.         else:
34.             if(dummy[i+1]-dummy[i] == 0):
35.                 dummy[i] = 0
36.             else:
37.                 dummy[i] = 1
38.
39.
40.     return dummy
```

e) Fill_one_whole_household.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import math
5.
6.  import pandas as pd
7.  import matplotlib.pyplot as plt
8.
9.  import datetime
10. import matplotlib.dates as mdates
11. import matplotlib.patches as mpatches
12.
13. import plotly
14. import plotly.plotly as py
15. import plotly.graph_objs as go
16.
17. import copy
18.
19. """
20. Calls database to create a matrix with the data asked
21. takes as arguments:
22. iter_size: number of households to extract
23. s_household: Starting household of DB to extract
24. s_year: Starting year to extract for each household
25. f_year: Final year to extract for each household
26. s_month: Starting month to extract for each household
27. f_month: Final month to extract for each houeshold
28. """
29.
30. def fill_one_whole_household(iter_size, s_household, s_year, f_year, s_month,
    f_month):
31.
32.     ## Open DB
33.     fileName = '/Users/user/Documents/Project/hourlyData.h5'
```

```
34.     f = h5py.File(fileName, libver='latest')
35.     file_names = list(f.keys())
36.
37.     ## Fields to get
38.     fields = ['Household', 'Hour', 'T_out [oF]', 'T_ctrl [oF]', 'Humidity','T_
    stp_cool [oF]', 'T_stp_heat [oF]', 'Remote Sensor 1 Motion', 'ID']
39.     month_days_2015 = {1 : 31, 2 : 28, 3 : 31, 4 : 30, 5 : 31, 6 : 30, 7 : 31,
     8 : 31, 9 : 30, 10 : 31, 11 : 30, 12 : 31 }
40.     month_days_2016 = {1 : 31, 2 : 29, 3 : 31, 4 : 30, 5 : 31, 6 : 30, 7 : 31,
     8 : 31, 9 : 30, 10 : 31, 11 : 30, 12 : 31 }
41.
42.     ## Variable declaration
43.     x_var = []
44.     sel_buildings = []
45.     first_entr = 0
46.     string = ""
47.     T_stp_cool = []
48.     count = 0
49.     for element in range(0,len(fields)):
50.         exec("x_var%s = []" %(element), globals())
51.
52.     ## Household ids
53.     sel_buildings = np.arange(0,len(file_names))
54.     sel_buildings = np.array(sel_buildings)[s_household:,]
55.
56.     ## Get data
57.     for limit, name in enumerate(sel_buildings):
58.         if(limit == iter_size):
59.             break
60.         for pos, field in enumerate(fields):
61.             for year in range(s_year,f_year+1):
62.                 for month in range(s_month, f_month+1):
63.                     try:
64.                         if(field == 'Household'):
65.                             exec("names_x = []")
66.                             exec("names_x = np.empty((24 * month_days_%s[month
    ],1))" %(year))
67.                             exec("names_x[:] = int(name)")
68.                             exec("x_var%s = np.append(x_var%s, names_x)" %(pos
    , pos))
69.                         elif(field == 'Hour'):
70.                             exec("hours_x = []")
71.                             exec("days = month_days_%s[month]" %(year))
72.                             exec("hours_x = np.empty((24 * days,1))")
73.                             for day in range(0,eval("days")):
74.                                 exec("hours_x[24*day:24*day+24,0] = np.arange(
    0,24,1)")
75.                             exec("hours_x = list(map(int, hours_x))")
76.                             exec("x_var%s = np.append(x_var%s, hours_x)" %(pos
    , pos))
77.                         elif(field == 'ID'):
78.                             exec("IDs_x = []")
79.                             exec("IDs_x = np.empty((24 * month_days_%s[month],
    1))" %(year))
80.                             for i in range(0, eval("len(IDs_x)")):
81.                                 exec("IDs_x[i] = count")
82.                                 count = count +1
83.                             exec("x_var%s = np.append(x_var%s, IDs_x)" %(pos,
    pos))
```

```
84.                             else:
85.                                 exec("x_var%s = np.append(x_var%s, f['/' + file_na
    mes[name] + '/' + str(year).zfill(2) + '/' + str(month).zfill(2) +'/table'][fi
    eld])" %(pos, pos))
86.
87.                         except:
88.                             exec("x_var_x = []", globals())
89.                             exec("x_var_x = np.empty((24 * month_days_%s[month],1)
    )" %(year))
90.                             exec("x_var_x[:] = np. nan")
91.                             exec("x_var%s = np.append(x_var%s, x_var_x)" %(pos, po
    s))
92.
93.     ## Group all columns into a single matrix
94.     for x in range(0,len(fields)):
95.         if(x == len(fields)-1):
96.             string += "eval(\"x_var" + str(x) + "\").reshape((" +  str(int(len
    (eval("x_var%s" %(x))))) + ",1))"
97.         else:
98.             string += "eval(\"x_var" + str(x) + "\").reshape((" +  str(int(len
    (eval("x_var%s" %(x))))) + ",1)),"
99.
100.         f.close()
101.
102.         return eval("np.hstack((%s))" %(string))
```

f) Fill_one_whole_household_v2.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import math
5.
6.  import pandas as pd
7.  import matplotlib.pyplot as plt
8.
9.  import datetime
10. from datetime import datetime
11.
12. import matplotlib.dates as mdates
13. import matplotlib.patches as mpatches
14.
15. import plotly
16. import plotly.plotly as py
17. import plotly.graph_objs as go
18.
19. import copy
20.
21. """
22. Calls database to create a matrix with the data asked
23. takes as arguments:
24. iter_size: number of households to extract
25. s_household: Starting household of DB to extract
26. s_year: Starting year to extract for each household
27. f_year: Final year to extract for each household
28. s_month: Starting month to extract for each household
29. f_month: Final month to extract for each houeshold
30.
```

```
31. Very similar to "fill_one_whole_household". The main
32. difference is the method that each one uses to fill the
33. new variable with the data from the database. This version
34. is quicker (and thus, preferable) since it avoids using
35. np.appends, which is very time-consuming.
36. """
37.
38. def fill_one_whole_household_v2(iter_size, s_household, s_year, f_year, s_mont
    h, f_month):
39.
40.     ## Open DB
41.     fileName = '/Users/user/Documents/Project/hourlyData.h5'
42.     f = h5py.File(fileName, libver='latest')
43.     file_names = list(f.keys())
44.
45.     ## Fields to get
46.     fields = ['Household', 'Year', 'Month', 'Day', 'Hour', 'T_out [oF]', 'T_ct
    rl [oF]', 'Humidity','T_stp_cool [oF]', 'T_stp_heat [oF]', 'ID']
47.     month_days_2015 = {1 : 31, 2 : 28, 3 : 31, 4 : 30, 5 : 31, 6 : 30, 7 : 31,
     8 : 31, 9 : 30, 10 : 31, 11 : 30, 12 : 31 }
48.     month_days_2016 = {1 : 31, 2 : 29, 3 : 31, 4 : 30, 5 : 31, 6 : 30, 7 : 31,
     8 : 31, 9 : 30, 10 : 31, 11 : 30, 12 : 31 }
49.
50.     ## Variable declaration
51.     x_var = []
52.     sel_buildings = []
53.     first_entr = 0
54.     string = ""
55.     T_stp_cool = []
56.     count = 0
57.     exec("total_rows = 0")
58.     exec("row_s = 0")
59.     exec("row_f = 0")
60.
61.     ## Create empty matrix
62.     for year in range(s_year,f_year+1):
63.         for month in range(s_month, f_month+1):
64.             exec("total_rows += 24 * month_days_%s[month]" %(year))
65.     data = np.zeros((iter_size * eval("total_rows") , len(fields)))
66.     data[:] = np.nan
67.
68.     ## Household ids
69.     for element in range(0,len(fields)):
70.         exec("x_var%s = []" %(element), globals())
71.
72.     sel_buildings = np.arange(0,len(file_names))
73.     sel_buildings = np.array(sel_buildings)[s_household:,]
74.
75.     ## Get data
76.     for limit, name in enumerate(sel_buildings):
77.         if(limit == iter_size):
78.             break
79.         start = datetime.now()
80.         for pos, field in enumerate(fields):
81.             for year in range(s_year,f_year+1):
82.                 for month in range(s_month, f_month+1):
83.                     try:
84.
85.                         exec("col = pos")
```

```
86.                             exec("row_s = row_f")
87.                             exec("row_f = row_s + (24 * month_days_%s[month])" %(y
    ear))
88.
89.                             #print(field,year,month,eval("row_s"),eval("row_f"),ev
    al("col"))
90.
91.                             if(field == 'Household'):
92.                                 exec("data[row_s:row_f,col] = int(name)")
93.
94.                             elif(field == 'Year'):
95.                                 exec("dates_x = []")
96.                                 exec("years_x = []")
97.                                 exec("dates_x = f['/' + file_names[name] + '/' + s
    tr(year).zfill(2) + '/' + str(month).zfill(2) +'/table']['Date'].astype('U13')
    ")
98.                                 exec("years_x = [i.split(\"-
    \")[0] for i in dates_x]")
99.                                 exec("data[row_s:row_f,col] = years_x")
100.
101.                                elif(field == 'Month'):
102.                                    exec("month_x = []")
103.                                    exec("dates_x = f['/' + file_names[name] +
    '/' + str(year).zfill(2) + '/' + str(month).zfill(2) +'/table']['Date'].astype
    ('U13')")
104.                                    exec("month_x = [i.split(\"-
    \")[1] for i in dates_x]")
105.                                    exec("data[row_s:row_f,col] = month_x")
106.
107.                                elif(field == 'Day'):
108.                                    exec("days_x = []")
109.                                    exec("dates_x = f['/' + file_names[name] +
    '/' + str(year).zfill(2) + '/' + str(month).zfill(2) +'/table']['Date'].astype
    ('U13')")
110.                                    exec("days_x = [i.split(\"-
    \")[2][:2] for i in dates_x]")
111.                                    exec("data[row_s:row_f,col] = days_x")
112.
113.                                elif(field == 'Hour'):
114.                                    exec("hours_x = []")
115.                                    exec("days = month_days_%s[month]" %(year))

116.                                    exec("hours_x = np.empty((24 * days,1))")
117.                                    for day in range(0,eval("days")):
118.                                        exec("hours_x[24*day:24*day+24,0] = np.
    arange(0,24,1)")
119.                                    exec("hours_x = list(map(int, hours_x))")
120.                                    exec("data[row_s:row_f,col] = hours_x")
121.
122.                                elif(field == 'ID'):
123.                                    exec("IDs_x = []")
124.                                    exec("IDs_x = np.empty((24 * month_days_%s[
    month]))" %(year))
125.                                    for i in range(0, eval("len(IDs_x)")):
126.                                        exec("IDs_x[i] = count")
127.                                        count = count +1
128.                                    exec("data[row_s:row_f,col] = IDs_x")
129.
130.                                else:
```

```
131.                                    exec("data[row_s:row_f,col] = f['/' + file_names[name] + '/' + str(year).zfill(2) + '/' + str(month).zfill(2) +'/table'][field]")
132.
133.                            except:
134.                                pass
135.
136.                    exec("row_s = 17544 * (limit)")
137.                    exec("row_f = 17544 * (limit)")
138.                exec("row_s = 17544 * (limit+1)")
139.                exec("row_f = 17544 * (limit+1)")
140.        #        print(limit, " --> Time elapsed: ",datetime.now()-start)
141.        #            print("end field. row_s: ",eval("row_s"))
142.
143.            f.close()
144.
145.            return eval("data")
```

g) Geodata_collection_dum.py

```
1.  """
2.  Generate a .txt file with the longitude and latitiude of each household
3.  """
4.
5.
6.  #################################################################
7.  def getLocation(name):
8.      geolocator = Nominatim(scheme='http')
9.      try:
10.         location = geolocator.geocode(name, timeout=5)
11.         return location
12.     except GeocoderTimedOut as e:
13.         print("Error: geocode failed on input %s with message %s" % (e.msg))
14.
15. sel = []
16.
17. for pos,i in enumerate(file_names[10:27],start = 10):
18.     try:
19.         if(np.isnan(f['/'+i].attrs['Country'])):
20.             print("nan!")
21.             location = getLocation(f['/'+i].attrs['City']+" "+f['/'+i].attrs['Province Or State'])
22.         else:
23.             location = getLocation(f['/'+i].attrs['City']+" "+f['/'+i].attrs['Province Or State']+" "+f['/'+i].attrs['Country'])
24.
25.         print(pos, ";", f['/'+i].attrs['Country'], ";",f['/'+i].attrs['Province Or State'], ";", f['/'+i].attrs['City'], ";", location.latitude, ";",location.longitude, ";",location.address)
26.         sel.append(str(pos)+";"+str(f['/'+i].attrs['Country'])+";"+str(f['/'+i].attrs['Province Or State'])+";"+ str(f['/'+i].attrs['City'])+";"+str(location.latitude)+";"+str(location.longitude)+ ";"+str(location.address))
27.     except:
28.         print(pos, ";", f['/'+i].attrs['Country'], ";",f['/'+i].attrs['Province Or State'], ";", f['/'+i].attrs['City'], ";", "NOT FOUND")
29.         sel.append(str(pos)+";"+str(f['/'+i].attrs['Country'])+";"+str(f['/'+i].attrs['Province Or State'])+";"+str(f['/'+i].attrs['City'])+";0;0;NOT FOUND")
```

```
30.
31.
32. sel = np.array(sel).reshape((len(sel),1))
33.
34. np.savetxt("/Users/user/Documents/Project/Exports/Metadata_new_v1"+str(datetim
    e.now().strftime('%d%m%y-%H%M')),sel,fmt='%s',delimiter=";")
35.
36. ##################################################################
```

h) Get_data.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import math
5.  from datetime import datetime
6.
7.  import sys
8.  sys.path.insert(0, '/Users/user/Documents/Project/IP_library')
9.  from Fill_one_whole_household_v2 import fill_one_whole_household_v2
10. from Transform_pre_clean import transform_pre_clean
11. from Transform_post_clean import transform_post_clean
12.
13. """
14. Get data from Database. 3 Modes:
15.
16. 1) dynamic: Script calls hdf5 file and then cleans it. This option is slower
17. since it has to restart each time the script restarts (memory lost between
18. compilations)
19.
20. 2) npy: Using "DB_generator.py", it is possible to create a .npy file with the

21. "cleaned" version of the database. This option simply loads such file, which
22. is much faster than rerunning the "dynamic" option each time.
23.
24. 3) txt: Same as "npy" but with a .txt file.
25. """
26.
27. def get_data(mode, households, st_household, st_year, fin_year, st_month, fin_
    month):
28.
29.     data = []
30.     BASE_DIR = '/Users/user/Documents/Project/'
31.
32.     if (mode == 'dynamic'):
33.         data = fill_one_whole_household_v2(households, st_household, st_year,
    fin_year, st_month, fin_month)
34.         data = transform_pre_clean(data)
35.         data = data[~np.isnan(data).any(axis=1)]
36.         data = transform_post_clean(data)
37.         data = data[np.argsort(data[:,len(data[0,:])-1]),:]
38.
39.     elif (mode == 'npy'):
40.         data = np.load(BASE_DIR+'/DDBB/Full_DB.npy')
41.         data = data[np.where(data[:,0] <= households)]
42.
43.     elif (mode == 'txt'):
44.
```

```
45.         with open('/Users/user/Documents/Project/DDBB/Full_DB', 'r') as ff:
46.             data = ff.read().splitlines()
47.
48.         data = [i.split(",") for i in data]
49.         data = np.array(data).astype(float)
50.
51.     else:
52.         print("Error. Mode not recognized. (options: 'dynamic', 'npy', 'txt')"
    )
53.
54.     return data
```

i) Number_clusters_optimization.py

```
1.  import h5py
2.  import numpy as np
3.  import copy
4.
5.  from sklearn.cluster import KMeans
6.  from matplotlib import pyplot as plt
7.
8.  """
9.  Plot KMeans clustering model inertia
10. depending on number of clusters.
11.
12. Useful to optimize the number of clusters
13. to use.
14. """
15.
16. def number_clusters_optimization(x_clus):
17.
18.     ## Variable declaration
19.     n_clusters = []
20.     inertia = []
21.     j = 1
22.     max_clusters = 10
23.
24.
25.     for i in range (0,max_clusters):
26.         n_clusters.append(j)
27.         kmeans = KMeans(n_clusters= n_clusters[i], random_state=0).fit(x_clus)

28.         inertia.append(kmeans.inertia_)
29.         j+=1
30.
31.     X_clus_check = []
32.     Y_clus_check = []
33.
34.     for n in range(0,max_clusters):
35.         X_clus_check.insert(n,n_clusters[n])
36.         Y_clus_check.insert(n,inertia[n])
37.
38.         if(n==9):
39.             plt.figure()
40.             plt.plot(X_clus_check, Y_clus_check, 'ro')
41.             plt.show()
42.
43.     return
```

j) Plot_one_month.py

```
1. import h5py
2. import numpy as np
3.
4. import pandas as pd
5. import matplotlib.pyplot as plt
6.
7. import datetime
8. import matplotlib.dates as mdates
9. import matplotlib.patches as mpatches
10.
11. import plotly
12. import plotly.plotly as py
13. import plotly.graph_objs as go
14.
15. """
16. Plot one month/day worth of data from a single household
17. """
18.
19. def Plot_one_month(year, month, day, selection, mode):
20.     ## Open file
21.     fileName = '/Users/user/Documents/Project/hourlyData.h5'
22.     f = h5py.File(fileName, libver='latest')
23.
24.     ## Get data
25.     T_ctrl     = []
26.     T_stp_cool = []
27.     T_stp_heat = []
28.     T_out      = []
29.     Thermostat_Temperature = []
30.     Humidity    = []
31.     HumidityExpectedLow    = []
32.     HumidityExpectedHigh   = []
33.
34.     file_names = list(f.keys())
35.
36.     full_name = '/' + file_names[selection] + '/' + str(year) + '/' + str(mont
    h).zfill(2) + '/table'
37.     semi_full_name = '/' + file_names[selection] + '/' + str(year) + '/' + str
    (month).zfill(2)
38.
39.     month_days_int = {1 : 31, 2 : 28, 3 : 31, 4 : 30, 5 : 31, 6 : 30, 7 : 31,
    8 : 31, 9 : 30, 10 : 31, 11 : 30, 12 : 31 }
40.     month_days_str = {"January" : 31, "February" : 28, "March" : 31, "April" :
     30, "May" : 31, "June" : 30, "July" : 31, "August" : 31, "September" : 30, "O
    ctober" : 31, "November" : 30, "December" : 31}
41.
42.     try:
43.         T_ctrl = f[full_name][:,'T_ctrl [oF]']
44.     except:
45.         T_ctrl = np.empty((24 * month_days_int[month],1))
46.         T_ctrl[:] = np.nan
47.
48.     try:
49.         T_stp_cool = f[full_name][:,'T_stp_cool [oF]']
50.     except:
51.         T_stp_cool = np.empty((24 * month_days_int[month],1))
52.         T_stp_cool[:] = np.nan
```

```
53.
54.     try:
55.         T_stp_heat = f[full_name][:,'T_stp_heat [oF]']
56.     except:
57.         T_stp_heat = np.empty((24 * month_days_int[month],1))
58.         T_stp_heat[:] = np.nan
59.
60.     try:
61.         T_out = f[full_name][:,'T_out [oF]']
62.     except:
63.         T_out = np.empty((24 * month_days_int[month],1))
64.         T_out[:] = np.nan
65.
66.     try:
67.         Thermostat_Temperature = f[full_name][:,'Thermostat Temperature [oF]']

68.     except:
69.         Thermostat_Temperature = np.empty((24 * month_days_int[month],1))
70.         Thermostat_Temperature[:] = np.nan
71.
72.     try:
73.         Humidity = f[full_name][:,'Humidity']
74.     except:
75.         Humidity = np.empty((24 * month_days_int[month],1))
76.         Humidity[:] = np.nan
77.
78.     try:
79.         HumidityExpectedHigh = f[full_name][:,'HumidityExpectedHigh']
80.     except:
81.         HumidityExpectedHigh = np.empty((24 * month_days_int[month],1))
82.         HumidityExpectedHigh[:] = np.nan
83.
84.     try:
85.         HumidityExpectedLow = f[full_name][:,'HumidityExpectedLow']
86.     except:
87.         HumidityExpectedLow = np.empty((24 * month_days_int[month],1))
88.         HumidityExpectedLow[:] = np.nan
89.
90.     ## Plot
91.     fig = plt.figure()
92.     ax = fig.add_subplot(2,1,1)
93.     ax2 = fig.add_subplot(2,1,2)
94.     plot_title = 'Temperature [oF] variables for ' + file_names[selection]
95.     plot_title_2 = 'Humidity for ' + file_names[selection]
96.
97.     ## Axis
98.     ax.xaxis_date()
99.     ax.set_title(plot_title)
100.
101.
102.         if(mode == 'day'):
103.             #ax.set_xlabel('Date: ' + str(year) + '-
    ' + str(month).zfill(2) + '-' +str(day).zfill(2)+'\n')
104.             ax.xaxis.set_major_locator(mdates.HourLocator())
105.             ax.xaxis.set_major_formatter(mdates.DateFormatter('  %H:%M'))
106.             datemin = datetime.date(year,month,day)
107.             datemax = datetime.date(year,month,day+1)
108.         else:
```

```
109.                #ax.set_xlabel('Date: ' + str(year) + '-
    ' + str(month).zfill(2))
110.                ax.xaxis.set_major_locator(mdates.DayLocator())
111.                ax.xaxis.set_major_formatter(mdates.DateFormatter('  %d'))
112.                datemin = datetime.date(year,month,1)
113.                datemax = datetime.date(year,month+1,1)
114.
115.            ax.set_xlim(datemin, datemax)
116.            plt.xticks(rotation=90)
117.
118.
119.            if(mode == 'day'):
120.                starting_day = str(year) + '-' + str(month).zfill(2) + '-
    ' + str(day).zfill(2) + 'T00:00'
121.                final_day = str(year) + '-' + str(month).zfill(2) + '-
    ' + str(day+1).zfill(2) + 'T00:00'
122.                starting_pos = (day-1) * 24
123.                final_pos = starting_pos + 24
124.
125.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_ctrl[starting_pos:final_pos],color='green')
126.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_stp_cool[starting_pos:final_pos],color='navy')
127.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_stp_heat[starting_pos:final_pos],color='red')
128.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_out[starting_pos:final_pos],color='gold')
129.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),Thermostat_Temperature[starting_pos:final_pos],color='deepskyblue')
130.
131.            else:
132.                starting_day = str(year) + '-' + str(month).zfill(2) + '-
    ' + str(day).zfill(2) + 'T00:00'
133.                final_day = str(year) + '-' + str(month+1).zfill(2) + '-
    ' + str(day).zfill(2) + 'T00:00'
134.
135.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_ctrl,color='green')
136.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_stp_cool,color='navy')
137.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_stp_heat,color='red')
138.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_out,color='gold')
139.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),Thermostat_Temperature,color='deepskyblue')
140.
141.            green_patch = mpatches.Patch(color='green', label='T_ctrl')
142.            gold_patch = mpatches.Patch(color='gold', label='T_out')
143.            red_patch = mpatches.Patch(color='red', label='T_stp_heat')
144.            navy_patch = mpatches.Patch(color='navy', label='T_stp_cool')
145.            deepskyblue_patch = mpatches.Patch(color='deepskyblue', label='Ther
    mostat_Temperature')
146.            ax.legend(handles=[green_patch, gold_patch, red_patch, navy_patch,
    deepskyblue_patch])
147.
148.
149.            ## Axis 2
150.            ax2.xaxis_date()
```

```
151.            ax2.set_title(plot_title_2)
152.
153.
154.            if(mode == 'day'):
155.                ax2.set_xlabel('Date: ' + str(year) + '-
    ' + str(month).zfill(2) + '-' +str(day).zfill(2))
156.                ax2.xaxis.set_major_locator(mdates.HourLocator())
157.                ax2.xaxis.set_major_formatter(mdates.DateFormatter('  %H:%M'))

158.                datemin_2 = datetime.date(year,month,day)
159.                datemax_2 = datetime.date(year,month,day+1)
160.
161.            else:
162.                ax2.set_xlabel('Date: ' + str(year) + '-
    ' + str(month).zfill(2))
163.                ax2.xaxis.set_major_locator(mdates.DayLocator())
164.                ax2.xaxis.set_major_formatter(mdates.DateFormatter('  %d'))
165.                datemin_2 = datetime.date(year,month,1)
166.                datemax_2 = datetime.date(year,month+1,1)
167.
168.            ax2.set_xlim(datemin_2, datemax_2)
169.            plt.xticks(rotation=90)
170.
171.
172.            if(mode == 'day'):
173.                starting_day_2 = str(year) + '-' + str(month).zfill(2) + '-
    ' + str(day).zfill(2) + 'T00:00'
174.                final_day_2 = str(year) + '-' + str(month).zfill(2) + '-
    ' + str(day+1).zfill(2) + 'T00:00'
175.                starting_pos_2 = (day-1) * 24
176.                final_pos_2 = starting_pos_2 + 24
177.
178.                ax2.plot(np.arange(starting_day_2, final_day_2,  60, dtype='dat
    etime64[m]'),HumidityExpectedLow[starting_pos_2:final_pos_2],color='navy')
179.                ax2.plot(np.arange(starting_day_2, final_day_2,  60, dtype='dat
    etime64[m]'),HumidityExpectedHigh[starting_pos_2:final_pos_2],color='red')
180.                ax2.plot(np.arange(starting_day_2, final_day_2,  60, dtype='dat
    etime64[m]'),Humidity[starting_pos_2:final_pos_2],color='gold')
181.
182.            else:
183.                starting_day_2 = str(year) + '-' + str(month).zfill(2) + '-
    ' + str(day).zfill(2) + 'T00:00'
184.                final_day_2 = str(year) + '-' + str(month+1).zfill(2) + '-
    ' + str(day).zfill(2) + 'T00:00'
185.
186.                ax2.plot(np.arange(starting_day_2, final_day_2,  60, dtype='dat
    etime64[m]'),HumidityExpectedLow,color='navy')
187.                ax2.plot(np.arange(starting_day_2, final_day_2,  60, dtype='dat
    etime64[m]'),HumidityExpectedHigh,color='red')
188.                ax2.plot(np.arange(starting_day_2, final_day_2,  60, dtype='dat
    etime64[m]'),Humidity,color='gold')
189.
190.            gold_patch = mpatches.Patch(color='gold', label='Humidity')
191.            red_patch = mpatches.Patch(color='red', label='HumidityExpectedHigh
    ')
192.            navy_patch = mpatches.Patch(color='navy', label='HumidityExpectedLo
    w')
193.            ax2.legend(handles=[ gold_patch, red_patch, navy_patch])
194.
```

```
195.            #plt.tight_layout(pad=0.01, w_pad=0.01, h_pad=0.01)
196.
197.            plt.show()
198.
199.            f.close()
200.
201.            return
```

k) Plot_one_month_v2.py

```
1.  import h5py
2.  import numpy as np
3.  import copy
4.
5.  import pandas as pd
6.  import matplotlib.pyplot as plt
7.
8.  import datetime
9.  import matplotlib.dates as mdates
10. import matplotlib.patches as mpatches
11.
12. import plotly
13. import plotly.plotly as py
14. import plotly.graph_objs as go
15.
16. from scipy.constants import F2C
17.
18. """
19. Plot one month/day worth of data from a single household
20. """
21.
22. def Plot_one_month_v2(year, month, day, selection, mode):
23.     #Open file
24.     fileName = '/Users/user/Documents/Project/hourlyData.h5'
25.     f = h5py.File(fileName, libver='latest')
26.
27.     #Get data
28.     T_ctrl      = []
29.     T_stp_cool = []
30.     T_stp_heat = []
31.     T_out       = []
32.     Thermostat_Temperature = []
33.     Humidity    = []
34.     HumidityExpectedLow    = []
35.     HumidityExpectedHigh   = []
36.
37.     file_names = list(f.keys())
38.
39.     full_name = '/' + file_names[selection] + '/' + str(year) + '/' + str(mont
    h).zfill(2) + '/table'
40.     semi_full_name = '/' + file_names[selection] + '/' + str(year) + '/' + str
    (month).zfill(2)
41.
42.     month_days_int = {1 : 31, 2 : 28, 3 : 31, 4 : 30, 5 : 31, 6 : 30, 7 : 31,
    8 : 31, 9 : 30, 10 : 31, 11 : 30, 12 : 31 }
43.     month_days_str = {"January" : 31, "February" : 28, "March" : 31, "April" :
     30, "May" : 31, "June" : 30, "July" : 31, "August" : 31, "September" : 30, "O
    ctober" : 31, "November" : 30, "December" : 31}
```

```
44.
45.     try:
46.         T_ctrl = f[full_name][:,'T_ctrl [oF]']
47.     except:
48.         T_ctrl = np.empty((24 * month_days_int[month],1))
49.         T_ctrl[:] = np.nan
50.
51.     try:
52.         T_stp_cool = f[full_name][:,'T_stp_cool [oF]']
53.     except:
54.         T_stp_cool = np.empty((24 * month_days_int[month],1))
55.         T_stp_cool[:] = np.nan
56.
57.     try:
58.         T_stp_heat = f[full_name][:,'T_stp_heat [oF]']
59.     except:
60.         T_stp_heat = np.empty((24 * month_days_int[month],1))
61.         T_stp_heat[:] = np.nan
62.
63.     try:
64.         T_out = f[full_name][:,'T_out [oF]']
65.     except:
66.         T_out = np.empty((24 * month_days_int[month],1))
67.         T_out[:] = np.nan
68.
69.     try:
70.         Thermostat_Temperature = f[full_name][:,'Thermostat Temperature [oF]']

71.     except:
72.         Thermostat_Temperature = np.empty((24 * month_days_int[month],1))
73.         Thermostat_Temperature[:] = np.nan
74.
75.     try:
76.         Humidity = f[full_name][:,'Humidity']
77.     except:
78.         Humidity = np.empty((24 * month_days_int[month],1))
79.         Humidity[:] = np.nan
80.
81.     try:
82.         HumidityExpectedHigh = f[full_name][:,'HumidityExpectedHigh']
83.     except:
84.         HumidityExpectedHigh = np.empty((24 * month_days_int[month],1))
85.         HumidityExpectedHigh[:] = np.nan
86.
87.     try:
88.         HumidityExpectedLow = f[full_name][:,'HumidityExpectedLow']
89.     except:
90.         HumidityExpectedLow = np.empty((24 * month_days_int[month],1))
91.         HumidityExpectedLow[:] = np.nan
92.
93.     T_ctrl = copy.copy(F2C(T_ctrl))
94.     T_stp_cool = copy.copy(F2C(T_stp_cool))
95.     T_stp_heat = copy.copy(F2C(T_stp_heat))
96.     T_out = copy.copy(F2C(T_out))
97.     Thermostat_Temperature = copy.copy(F2C(Thermostat_Temperature))
98.
99.     #Plot
100.         fig = plt.figure()
101.         ax = fig.add_subplot(1,1,1)
```

```
102.        #     ax2 = fig.add_subplot(2,1,2)
103.            plot_title = 'Temperature [oF] variables for ' + file_names[selecti
    on]
104.            plot_title_2 = 'Humidity for ' + file_names[selection]
105.
106.            #ax
107.            ax.xaxis_date()
108.        #     ax.set_title(plot_title)
109.
110.
111.            if(mode == 'day'):
112.                #ax.set_xlabel('Date: ' + str(year) + '-
    ' + str(month).zfill(2) + '-' +str(day).zfill(2)+'\n')
113.                ax.xaxis.set_major_locator(mdates.HourLocator())
114.                ax.xaxis.set_major_formatter(mdates.DateFormatter('  %H:%M'))
115.                datemin = datetime.date(year,month,day)
116.                datemax = datetime.date(year,month,day+1)
117.            else:
118.                #ax.set_xlabel('Date: ' + str(year) + '-
    ' + str(month).zfill(2))
119.                ax.xaxis.set_major_locator(mdates.DayLocator())
120.                ax.xaxis.set_major_formatter(mdates.DateFormatter('  %d'))
121.                datemin = datetime.date(year,month,1)
122.                datemax = datetime.date(year,month+1,1)
123.
124.            ax.set_xlim(datemin, datemax)
125.            plt.xticks(rotation=90)
126.
127.
128.            if(mode == 'day'):
129.                starting_day = str(year) + '-' + str(month).zfill(2) + '-
    ' + str(day).zfill(2) + 'T00:00'
130.                final_day = str(year) + '-' + str(month).zfill(2) + '-
    ' + str(day+1).zfill(2) + 'T00:00'
131.                starting_pos = (day-1) * 24
132.                final_pos = starting_pos + 24
133.
134.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_ctrl[starting_pos:final_pos],color='green')
135.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_stp_cool[starting_pos:final_pos],color='navy')
136.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_stp_heat[starting_pos:final_pos],color='red')
137.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_out[starting_pos:final_pos],color='gold')
138.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),Thermostat_Temperature[starting_pos:final_pos],color='deepskyblue')
139.
140.            else:
141.                starting_day = str(year) + '-' + str(month).zfill(2) + '-
    ' + str(day).zfill(2) + 'T00:00'
142.                final_day = str(year) + '-' + str(month+1).zfill(2) + '-
    ' + str(day).zfill(2) + 'T00:00'
143.
144.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_ctrl,color='green')
145.                ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_stp_cool,color='navy')
```

```
146.            ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_stp_heat,color='red')
147.            ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),T_out,color='gold')
148.            ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime
    64[m]'),Thermostat_Temperature,color='deepskyblue')
149.
150.        green_patch = mpatches.Patch(color='green', label='T_ctrl')
151.        gold_patch = mpatches.Patch(color='gold', label='T_out')
152.        red_patch = mpatches.Patch(color='red', label='T_stp_heat')
153.        navy_patch = mpatches.Patch(color='navy', label='T_stp_cool')
154.        deepskyblue_patch = mpatches.Patch(color='deepskyblue', label='Ther
    mostat_Temperature')
155.        ax.legend(handles=[green_patch, gold_patch, red_patch, navy_patch,
    deepskyblue_patch])
156.        ax.set_xlabel("day")
157.        ax.set_ylabel("Temperature [ºC]")
158.
159.
160.        plt.show()
161.
162.        f.close()
163.
164.        return
```

l) Plot_one_year.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import pandas as pd
5.  import matplotlib.pyplot as plt
6.
7.  import datetime
8.  import matplotlib.dates as mdates
9.  import matplotlib.patches as mpatches
10.
11. import plotly
12. import plotly.plotly as py
13. import plotly.graph_objs as go
14.
15. """
16. Plot one year worth of data from a single household
17. """
18.
19. def Plot_one_year(year, selection):
20.     #Open file
21.     fileName = '/Users/user/Documents/Project/hourlyData.h5'
22.     f = h5py.File(fileName, libver='latest')
23.
24.     #Get data
25.     T_ctrl      = []
26.     T_stp_cool = []
27.     T_stp_heat = []
28.     T_out       = []
29.     Thermostat_Temperature = []
30.     Humidity    = []
31.     HumidityExpectedLow    = []
```

```
32.     HumidityExpectedHigh  = []
33.
34.     file_names = list(f.keys())
35.
36.     if (year == 2016):
37.         month_days_int = {1 : 31, 2 : 29, 3 : 31, 4 : 30, 5 : 31, 6 : 30, 7 :
    31, 8 : 31, 9 : 30, 10 : 31, 11 : 30, 12 : 31 }
38.         month_days_str = {"January" : 31, "February" : 29, "March" : 31, "Apri
    l" : 30, "May" : 31, "June" : 30, "July" : 31, "August" : 31, "September" : 30
    , "October" : 31, "November" : 30, "December" : 31}
39.     else:
40.         month_days_int = {1 : 31, 2 : 28, 3 : 31, 4 : 30, 5 : 31, 6 : 30, 7 :
    31, 8 : 31, 9 : 30, 10 : 31, 11 : 30, 12 : 31 }
41.         month_days_str = {"January" : 31, "February" : 28, "March" : 31, "Apri
    l" : 30, "May" : 31, "June" : 30, "July" : 31, "August" : 31, "September" : 30
    , "October" : 31, "November" : 30, "December" : 31}
42.
43.     for i in range(1, 13):
44.         try:
45.             T_ctrl = np.append(T_ctrl, f['/' + file_names[selection] + '/' + s
    tr(year).zfill(2) + '/' + str(i).zfill(2) +'/table']['T_ctrl [oF]'])
46.         except:
47.             T_ctrl_x = []
48.             T_ctrl_x = np.empty((24 * month_days_int[i],1))
49.             T_ctrl_x[:] = np. nan
50.             T_ctrl = np.append(T_ctrl, T_ctrl_x)
51.
52.     for i in range(1, 13):
53.         try:
54.             T_out = np.append(T_out, f['/' + file_names[selection] + '/' + str
    (year).zfill(2) + '/' + str(i).zfill(2) +'/table']['T_out [oF]'])
55.         except:
56.             T_out_x = []
57.             T_out_x = np.empty((24 * month_days_int[i],1))
58.             T_out_x[:] = np. nan
59.             T_out = np.append(T_out, T_out_x)
60.
61.     for i in range(1, 13):
62.         try:
63.             T_stp_cool = np.append(T_stp_cool, f['/' + file_names[selection] +
     '/' + str(year).zfill(2) + '/' + str(i).zfill(2) +'/table']['T_stp_cool [oF]'
    ])
64.         except:
65.             T_stp_cool_x = []
66.             T_stp_cool_x = np.empty((24 * month_days_int[i],1))
67.             T_stp_cool_x[:] = np. nan
68.             T_stp_cool = np.append(T_stp_cool, T_stp_cool_x)
69.
70.     for i in range(1, 13):
71.         try:
72.             T_stp_heat = np.append(T_stp_heat, f['/' + file_names[selection] +
     '/' + str(year).zfill(2) + '/' + str(i).zfill(2) +'/table']['T_stp_heat [oF]'
    ])
73.         except:
74.             T_stp_heat_x = []
75.             T_stp_heat_x = np.empty((24 * month_days_int[i],1))
76.             T_stp_heat_x[:] = np. nan
77.             T_stp_heat = np.append(T_stp_heat, T_stp_heat_x)
78.
```

```
79.     for i in range(1, 13):
80.         try:
81.             Thermostat_Temperature = np.append(Thermostat_Temperature, f['/' +
 file_names[selection] + '/' + str(year).zfill(2) + '/' + str(i).zfill(2) +'/t
able']['Thermostat_Temperature [oF]'])
82.         except:
83.             Thermostat_Temperature_x = []
84.             Thermostat_Temperature_x = np.empty((24 * month_days_int[i],1))
85.             Thermostat_Temperature_x[:] = np. nan
86.             Thermostat_Temperature = np.append(Thermostat_Temperature, Thermos
tat_Temperature_x)
87.
88.     for i in range(1, 13):
89.         try:
90.             Humidity = np.append(Humidity, f['/' + file_names[selection] + '/'
 + str(year).zfill(2) + '/' + str(i).zfill(2) +'/table']['Humidity'])
91.         except:
92.             Humidity_x = []
93.             Humidity_x = np.empty((24 * month_days_int[i],1))
94.             Humidity_x[:] = np. nan
95.             Humidity = np.append(Humidity, Humidity_x)
96.
97.     for i in range(1, 13):
98.         try:
99.             HumidityExpectedLow = np.append(HumidityExpectedLow, f['/' + file_
names[selection] + '/' + str(year).zfill(2) + '/' + str(i).zfill(2) +'/table']
['HumidityExpectedLow'])
100.            except:
101.                HumidityExpectedLow_x = []
102.                HumidityExpectedLow_x = np.empty((24 * month_days_int[i],1)
)
103.                HumidityExpectedLow_x[:] = np. nan
104.                HumidityExpectedLow = np.append(HumidityExpectedLow, Humidi
tyExpectedLow_x)
105.
106.        for i in range(1, 13):
107.            try:
108.                HumidityExpectedHigh = np.append(HumidityExpectedHigh, f['/
' + file_names[selection] + '/' + str(year).zfill(2) + '/' + str(i).zfill(2) +
'/table']['HumidityExpectedHigh'])
109.            except:
110.                HumidityExpectedHigh_x = []
111.                HumidityExpectedHigh_x = np.empty((24 * month_days_int[i],1
))
112.                HumidityExpectedHigh_x[:] = np. nan
113.                HumidityExpectedHigh = np.append(HumidityExpectedHigh, Humi
dityExpectedHigh_x)
114.
115.
116.        #Plot
117.        fig = plt.figure()
118.        ax = fig.add_subplot(2,1,1)
119.        ax2 = fig.add_subplot(2,1,2)
120.        plot_title = 'Temperature [oF] variables for ' + file_names[selecti
on]
121.        plot_title_2 = 'Humidity for ' + file_names[selection]
122.
123.        #ax
124.        ax.xaxis_date()
```

```
125.            ax.set_title(plot_title)
126.
127.            #ax.set_xlabel('Date: ' + str(year) + '-' + str(month).zfill(2))
128.            ax.xaxis.set_major_locator(mdates.MonthLocator())
129.            ax.xaxis.set_major_formatter(mdates.DateFormatter('   %m'))
130.            datemin = datetime.date(year,1,1)
131.            datemax = datetime.date(year,12,31)
132.
133.            ax.set_xlim(datemin, datemax)
134.            starting_day = str(year).zfill(2) + '-01-01T00:00'
135.            final_day = str(year+1).zfill(2) + '-01-01T00:00'
136.
137.            ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime64[m
    ]'),T_ctrl,color='green')
138.            ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime64[m
    ]'),T_stp_cool,color='navy')
139.            ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime64[m
    ]'),T_stp_heat,color='red')
140.            ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime64[m
    ]'),T_out,color='gold')
141.            ax.plot(np.arange(starting_day, final_day,  60, dtype='datetime64[m
    ]'),Thermostat_Temperature,color='deepskyblue')
142.
143.            green_patch = mpatches.Patch(color='green', label='T_ctrl')
144.            gold_patch = mpatches.Patch(color='gold', label='T_out')
145.            red_patch = mpatches.Patch(color='red', label='T_stp_heat')
146.            navy_patch = mpatches.Patch(color='navy', label='T_stp_cool')
147.            deepskyblue_patch = mpatches.Patch(color='deepskyblue', label='Ther
    mostat_Temperature')
148.            ax.legend(handles=[green_patch, gold_patch, red_patch, navy_patch,
    deepskyblue_patch])
149.
150.
151.            #ax2
152.            ax2.xaxis_date()
153.            ax2.set_title(plot_title_2)
154.
155.            ax2.set_xlabel('Date: ' + str(year))
156.            ax2.xaxis.set_major_locator(mdates.MonthLocator())
157.            ax2.xaxis.set_major_formatter(mdates.DateFormatter('   %m'))
158.            datemin_2 = datetime.date(year,1,1)
159.            datemax_2 = datetime.date(year,12,1)
160.
161.            ax2.set_xlim(datemin_2, datemax_2)
162.            ax2.plot(np.arange(starting_day, final_day,  60, dtype='datetime64[
    m]'),HumidityExpectedLow,color='navy')
163.            ax2.plot(np.arange(starting_day, final_day,  60, dtype='datetime64[
    m]'),HumidityExpectedHigh,color='red')
164.            ax2.plot(np.arange(starting_day, final_day,  60, dtype='datetime64[
    m]'),Humidity,color='gold')
165.
166.            gold_patch = mpatches.Patch(color='gold', label='Humidity')
167.            red_patch = mpatches.Patch(color='red', label='HumidityExpectedHigh
    ')
168.            navy_patch = mpatches.Patch(color='navy', label='HumidityExpectedLo
    w')
169.            ax2.legend(handles=[ gold_patch, red_patch, navy_patch])
170.
171.            plt.show()
```

```
172. 
173.            f.close()
174. 
175.            return
```

m) Supervised_format.py

```
1.  import h5py
2.  import numpy as np
3.  
4.  from math import sqrt
5.  from numpy import concatenate
6.  from matplotlib import pyplot as plt
7.  from pandas import read_csv
8.  from pandas import DataFrame
9.  from pandas import concat
10. from sklearn.preprocessing import MinMaxScaler
11. from sklearn.preprocessing import LabelEncoder
12. from sklearn.metrics import mean_squared_error
13. from keras.models import Sequential
14. from keras.layers import Dense
15. from keras.layers import LSTM
16. 
17. """
18. Transform variable series to supervised format
19. """
20. 
21. def supervised_format(data, n_in=1, n_out=1, dropnan=True):
22.     n_vars = 1 if type(data) is list else data.shape[1]
23.     df = DataFrame(data)
24.     cols, names = list(), list()
25. 
26.     ## Sequence to (t-n, ... t-1) format
27.     for i in range(n_in, 0, -1):
28.         cols.append(df.shift(i))
29.         names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
30. 
31.     ## Forecast
32.     for i in range(0, n_out):
33.         cols.append(df.shift(-i))
34.         if i == 0:
35.             names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
36.         else:
37.             names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
38. 
39.     ## Consolidate
40.     agg = concat(cols, axis=1)
41.     agg.columns = names
42. 
43.     ## Delete NaN rows
44.     if dropnan:
45.         agg.dropna(inplace=True)
46.     return agg
```

n) Transform_data.py

```
1.  import h5py
2.  import numpy as np
3.  import copy
4.
5.  """
6.  Transform data from original database format to day-by-day format
7.  """
8.
9.  def transform_data(x_var):
10.
11.     x_transformed = []
12.     string = ""
13.
14.     for hour in range(0,24):
15.         exec("x_var_h%s = []" %(hour), globals())
16.         exec("x_var_h%s = copy.copy(x_var[np.where(x_var[:,0] == %s),1])" %(ho
    ur, hour))
17.
18.     for x in range(0,24):
19.         if(x == 23):
20.             string += "eval(\"x_var_h" + str(x) + "\").reshape((" +  str(int(l
    en(x_var[:,0])/24)) + ",1))"
21.         else:
22.             string += "eval(\"x_var_h" + str(x) + "\").reshape((" +  str(int(l
    en(x_var[:,0])/24)) + ",1)),"
23.
24.     return eval("np.hstack((%s))" %(string))
```

o) Transform_data_inv.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import copy
5.
6.  """
7.  Inverse transformation. From day-format to original database format
8.  """
9.
10. def transform_data_inv(labels, x_var, only_labels=True, by_days=True):
11.
12.     output = []
13.     x_transformed = np.zeros((len(x_var[:,0]),len(x_var[0,:])+1))
14.     x_transformed[:,1:] = copy.copy(x_var)
15.
16.     if by_days:
17.         for i in range(0, len(labels)):
18.             for j in range(0, 24):
19.                 x_transformed[24*i + j,0] = labels[i]
20.     else:
21.         h_uniques = np.unique(x_var[:,0])
22.         for i in range(0,len(labels)):
23.             x_transformed[np.where(x_var[:,0] == h_uniques[i]),0] = labels[i]

24.
25.
```

```
26.     if only_labels:
27.         output = copy.copy(x_transformed[:,0])
28.     else:
29.         output = copy.copy(x_transformed)
30.
31.     return output
```

p) Transform_data_inv_v2.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import copy
5.
6.  from datetime import datetime
7.
8.  """
9.  Inverse transformation. From day-format to original database format
10. Case for Boxplot figure
11. """
12.
13. def transform_data_inv_v2(labels, x_var, only_labels=True, by_days=True):
14.
15.     BASE_DIR = '/Users/user/Documents/Project/Exports/'
16.
17.     output = []
18.     x_transformed = np.zeros((len(x_var[:,0]),len(x_var[0,:])+1))
19.     x_transformed[:,1:] = copy.copy(x_var)
20.     h_uniques = np.unique(x_var[:,0])
21.     c_labels = np.zeros(len(h_uniques))
22.
23.     for i in range(0,len(c_labels)):
24.         c_labels[i] = labels[np.where(np.in1d(labels[:,0], h_uniques[i])),1]
25.
26.     if by_days:
27.         for i in range(0, len(labels)):
28.             for j in range(0, 24):
29.                 x_transformed[24*i + j,0] = labels[i]
30.     else:
31.         for i in range(0,len(x_var[:,0])):
32.             x_transformed[i,0] = c_labels[np.where(x_var[i,0] == h_uniques)]
33.
34.     if only_labels:
35.         output = copy.copy(x_transformed[:,0])
36.     else:
37.         output = copy.copy(x_transformed)
38.
39.     return output
```

q) Transform_data_inv_1D.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import copy
5.
6.  """
```

```
7.  Inverse transformation. From day-format to original database format
8.  Case when input variable is 1D
9.  """
10.
11. def transform_data_inv_1D(labels, x_var, only_labels=True, by_days=True):
12.
13.     output = []
14.     x_var = x_var.reshape(-1,1)
15.     x_transformed = np.zeros((len(x_var),2))
16.     x_transformed[:,1:] = copy.copy(x_var)
17.
18.     print(x_transformed.shape)
19.     print(labels.shape)
20.     print(len(labels))
21.
22.     if by_days:
23.         for i in range(0, len(labels)):
24.             for j in range(0, 24):
25.                 print(i,j,labels[i])
26.                 x_transformed[24*i + j,0] = labels[i]
27.     else:
28.         for i in range(0,len(labels)):
29.             x_transformed[np.where(x_var[:] == np.unique(x_var[:])[i]),0] = la
    bels[i]
30.
31.
32.     if only_labels:
33.         output = copy.copy(x_transformed[:,0])
34.     else:
35.         output = copy.copy(x_transformed)
36.
37.     return output
```

r) Transform_data_multinom.py

```
1.  import h5py
2.  import numpy as np
3.
4.  import copy
5.
6.  from Clean_nan import clean_nan
7.  from Fill_one_household import fill_one_household
8.
9.  """
10. Inverse transformation. From day-format to original database format
11. Case for multinomial regression
12. """
13.
14. def transform_data_multinom(labels, x_var):
15.
16.     x_transformed = np.zeros((len(x_var),2))
17.     x_transformed[:,1] = copy.copy(x_var)
18.
19.     for i in range(0, len(labels)):
20.         for j in range(0, 24):
21.             x_transformed[24*i + j,0] = labels[i]
22.
23.     return x_transformed
```

s) Transform_pre_clean.py

```
1. import h5py
2. import numpy as np
3.
4. import plotly
5. import plotly.plotly as py
6. import plotly.graph_objs as go
7.
8. import matplotlib.pyplot as plt
9. import copy
10.
11. """
12. Data transformation from original database format to day-by-day format
13. to delete rows with NaN values from the data extracted from the database
14. """
15.
16. def transform_pre_clean(x_var):
17.
18.     x_transformed = []
19.     string = ""
20.     string_2 = ""
21.     columns = len(x_var[0,:])
22.     variable_x = []
23.
24.     for column in range(0,columns):
25.
26.         exec("variable%s = []" %(column), globals())
27.
28.         for hour in range(0,24):
29.             exec("x_var_h%s = []" %(hour), globals())
30.             exec("x_var_h%s = copy.copy(x_var[np.where(x_var[:,4] == %s),%s])"
   %(hour, hour, column))
31.
32.         for x in range(0,24):
33.             if(x == 23):
34.                 string += "eval(\"x_var_h" + str(x) + "\").reshape((-1,1))"
35.             else:
36.                 string += "eval(\"x_var_h" + str(x) + "\").reshape((-1,1)),"
37.
38.         variable_x = eval("np.hstack((%s))" %(string))
39.         exec("variable%s = copy.copy(variable_x)" %(column))
40.         string = ""
41.
42.     for x in range(0,columns):
43.         if(x == columns-1):
44.             string_2 += "eval(\"variable" + str(x) + "\")"
45.         else:
46.             string_2 += "eval(\"variable" + str(x) + "\"),"
47.
48.     return eval("np.hstack((%s))" %(string_2))
```

t) Transform_post_clean.py

```
1. import h5py
2. import numpy as np
3.
4. import plotly
```

```
5.  import plotly.plotly as py
6.  import plotly.graph_objs as go
7.
8.  import matplotlib.pyplot as plt
9.  import copy
10.
11. """
12. Once the NaN rows from the day-by-day format have been removed
13. from the data variable, this inverts the Transform_pre_clean
14. transformation to leave the data in the original format
15. """
16.
17. def transform_post_clean(x_var):
18.
19.     x_transformed = []
20.     string = ""
21.     columns = len(x_var[0,:])
22.     variable_x = []
23.
24.
25.     for variable in range(0, int(len(x_var[0,:])/24)):
26.         exec("x_var_h%s = []" %(variable), globals())
27.         exec("x_var_h%s = np.transpose(x_var[:,24*variable:24*variable+24])" %
    (variable))
28.         exec("x_var_h%s = x_var_h%s.reshape(len(x_var_h%s[:,0]) * len(x_var_h%
    s[0,:]), 1)" %(variable, variable, variable, variable))
29.
30.
31.     for x in range(0,int(len(x_var[0,:])/24)):
32.         if(x == int(len(x_var[0,:])/24)-1):
33.             string += "eval(\"x_var_h" + str(x) + "\")"
34.         else:
35.             string += "eval(\"x_var_h" + str(x) + "\"),"
36.
37.     return eval("np.hstack((%s))" %(string))
```

[End of document]