Abstract

Este documento ofrece una guía detallada del proceso ETL implementado en el Instituto Privado de Investigación sobre Cambio Climático (ICC). Explora procedimientos y flujos de trabajo esenciales para la preparación y análisis de datos climáticos,

**Manual de Procesos ETL para el Instituto Privado De Investigación Sobre Cambio Climático-ICC**

**Contenido**

[**Introducción** 2](#_Toc151731536)

[**1.** **Diagrama de procesos ETL** 3](#_Toc151731537)

[**2.** **Fase carga de información al motor de base de datos** 4](#_Toc151731538)

[**3.** **Fase carga de información catálogo** 7](#_Toc151731539)

[**4.** **Fase control de calidad** 12](#_Toc151731540)

[**5.** **Fase de consultas** 22](#_Toc151731541)

# **Introducción**

Este manual presenta una detallada documentación sobre el proceso de Extracción, Transformación y Carga (ETL) implementado en el Instituto Privado de Investigación sobre Cambio Climático (ICC). La ETL desempeña un papel fundamental en la preparación y análisis de datos climáticos, permitiendo una operacionalización eficiente del control de calidad. A través de esta guía, se proporciona una visión integral de los procedimientos, componentes y flujos de trabajo involucrados en el ciclo ETL. El objetivo es facilitar la comprensión y gestión de este proceso esencial para el ICC, contribuyendo así a un manejo efectivo de la información.

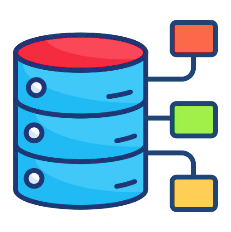
# **Diagrama de procesos ETL**

****

**1..Extracción**

1.2.Se carga información cruda en el esquema RedmetRaw

1.1. Se hace conexión entre API del REDMET y el server



****

2.2. Se calculan agregaciones diarias, semanales y anuales con la data ya modificada

2.1. La data que ingresa cruda, pasa por un tratamiento de control de calidad

**2.Transformación**



****

3.2. Después, la información se carga en la plataforma en línea, poniéndola a disposición de los usuarios que utilizan estos datos.

3.1. La data que es tratada con control de calidad ingresa al server en el esquema de datos Redmet

**3.Carga**

Este flujo de procesos de la ETL representa la **operacionalización** integral del ciclo. Inicia con la extracción de información a través de la conexión entre la API de REDMET y el servidor, seguido por el almacenamiento de esta data en el servidor. Posteriormente, la información cruda experimenta una transformación rigurosa mediante un proceso de control de calidad y cálculos específicos. Finalmente, los datos tratados son recargados en la plataforma en línea, garantizando su disponibilidad para los usuarios que aprovechan estos datos.

# **Fase carga de información al motor de base de datos**

El propósito de esta etapa es cargar la información histórica en dos fases: la primera que implica la carga de datos históricos en sin modificación y la segunda los datos históricos con un riguroso control de calidad. En el servidor, se distinguen dos esquemas clave: Redmet y RedmetRaw.

En Redmet, se almacena la información histórica que ha pasado por un exhaustivo control de calidad. Esta información está lista para ser cargada de nuevo en la plataforma en línea, estando así disponible para ser consumida por diversos usuarios.

Por otro lado, el segundo repositorio, RedmetRaw, se utiliza para la carga de la información histórica en su forma cruda, antes de someterse al proceso de control de calidad.

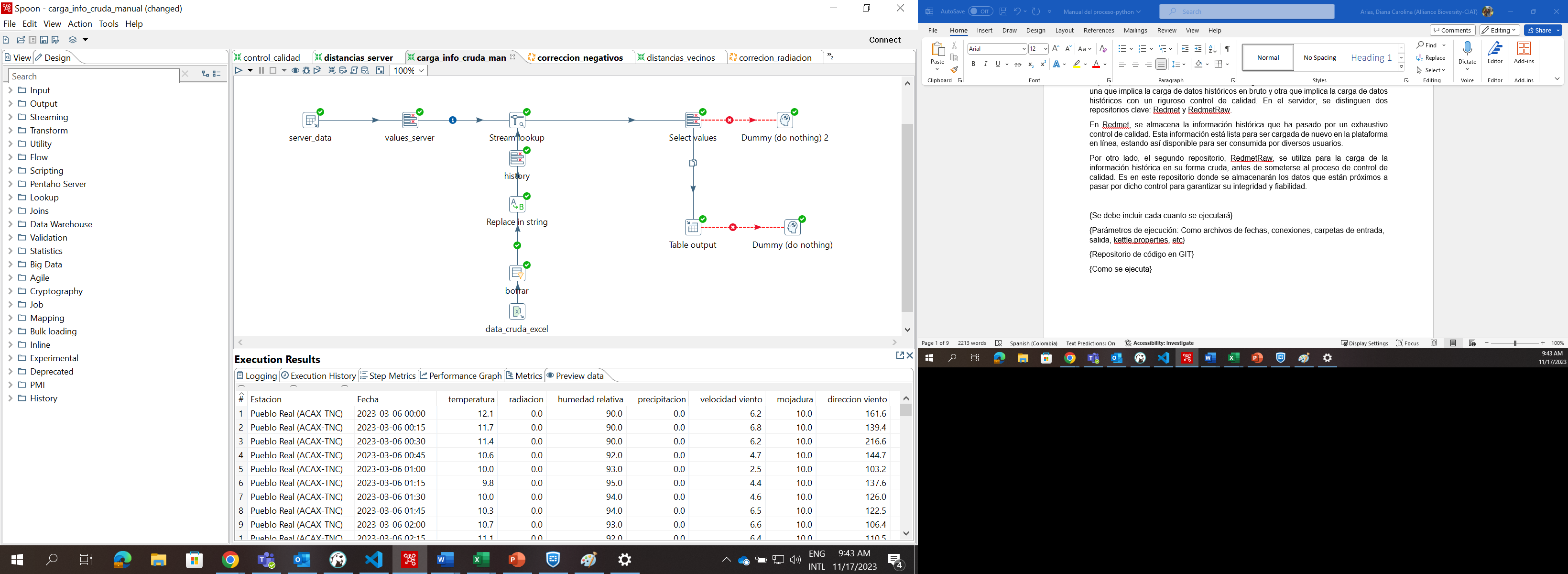


Ilustración 1 Flujo pentaho- Carga de información cruda

La transformación **load\_data\_raw.ktr** tiene la responsabilidad de cargar la información histórica en bruto en el servidor, específicamente en el repositorio RedmetRaw. Esta transformación está compuesta por los siguientes componentes:

1. **Table Input (Server\_data):** Este componente carga la tabla "Estacion" que reside en el servidor. Dicha tabla contiene información completa acerca de las estaciones meteorológicas automáticas supervisadas por ICC en Guatemala.
2. **Select Values (Values\_server):** Encargado de seleccionar las columnas necesarias de la tabla "Estacion".
3. **Microsoft Excel Input (data\_cruda\_excel):** Este componente se dedica a leer individualmente los archivos de Excel almacenados localmente en el equipo, consolidándolos en un único dataframe.
4. **Filter Rows (borrar):** Este componente realiza dos funciones. En primer lugar, limpia cada archivo de lectura, ya que estos archivos suelen tener filas adicionales al final que deben ser eliminadas. En segundo lugar, elimina de la carga de datos una estación identificada como "Miravalle", ya que no forma parte de los procesos actuales.
5. **Replace in String:** Este componente se encarga de ajustar los nombres de cinco estaciones, garantizando que estén escritos con los caracteres especiales correspondientes. Este proceso de reemplazo se aplica a la columna "Estacion", donde se identifican los nombres de las estaciones (Cengicana, El Balsamo, San Nicolas, Peten Oficina y Tulula) y se sustituyen por sus equivalentes con caracteres especiales (Cengicaña, El Bálsamo, San Nicolás, Petén Oficina y Tululá). Además, este componente realiza otro reemplazo en la columna "Fecha", identificando aquellas fechas con formato de guion (yyyy-mm-dd) y sustituyéndolas por un formato de barra inclinada (yyyy/mm/dd).
6. **Select Values (history):** Este componente convierte el formato de la columna "Fecha" a tipo string.
7. **Stream lookup :** Responsable de vincular la base de datos de "Estacion" con la información cruda previamente cargada. Esta acción es esencial, ya que las estaciones desde el servidor cuentan con un código de identificación, y es necesario agregar esos identificadores a la información cruda que se está cargando en el servidor.
8. **Select Values :** Este componente convierte el formato de la columna “Fecha” a tipo Date.
9. **Dummy (Errores de formato fecha):** Este componente dummy tiene la función de detectar los registros que presentan errores en el formato de fecha y que no han sido correctamente identificados. Estos registros requieren un tratamiento especial para realizar los ajustes pertinentes.
10. **Table Output (EstacionLectura):** Este componente es responsable de registrar toda la información en la tabla "EstacionLectura" ubicada en el servidor. Con la ejecución de este componente, se completa la carga de la información histórica en su forma cruda.
11. **Dummy:** Este componente se encarga de identificar todos los registros que no cumplen con los formatos adecuados para ser cargados en la tabla "EstacionLectura".

**Tiempo de ejecución:** Este flujo se ha llevado a cabo para cargar la información local almacenada en el equipo. En otras palabras, esta ejecución es única, ya que la información cruda que se carga en la tabla "EstacionLectura" del repositorio RedmetRaw será posteriormente cargada directamente mediante la conexión desde la API del Redmet.

A screenshot of a computer

Description automatically generated

Ilustración 2 Flujo pentaho- Carga de información histórica con control de calidad

La transformación **load\_data\_control.ktr** tiene la responsabilidad de cargar la información histórica de la data que ha pasado por control de calidad en el servidor, específicamente en el repositorio Redmet. Esta transformación está compuesta por los siguientes componentes:

1. **Table Input (Server\_data):** Este componente carga la tabla "Estacion" ubicada en el servidor, la cual contiene información completa sobre las estaciones meteorológicas automáticas supervisadas por ICC en Guatemala.
2. **Select Values (Values\_server):** Se encarga de seleccionar las variables necesarias de la tabla "Estacion".
3. **Text File Input:** Este componente se dedica a leer individualmente los archivos en formato CSV almacenados localmente en el equipo, consolidándolos en un único dataframe.
4. **Select Values (history):** Organiza y selecciona las columnas pertinentes de la tabla formada por todos los archivos CSV del componente anterior.
5. **Stream Lookup**: Es responsable de vincular la base de datos de "Estacion" con la información con control de calidad previamente cargada. Esta acción es esencial, ya que las estaciones desde el servidor cuentan con un código de identificación, y es necesario agregar esos identificadores a la información con control de calidad que se está cargando en el servidor.
6. **Select Values:** Ajusta el formato de la columna fecha, dejándolo en tipo Date y en el formato yyy/mm/dd HH:mm.
7. **Table Output:** Se encarga de insertar la información histórica con control de calidad en la tabla "EstacionLectura" del servidor en el repositorio Redmet.

**Tiempo de ejecución:** Este flujo se ha llevado a cabo para cargar la información local almacenada en el equipo. En otras palabras, esta ejecución es única, ya que la información historica con control de calidad que se cargó en la tabla "EstacionLectura" del repositorio Redmet es data local.

# **Fase carga de información catálogo**

Esta etapa tiene como objetivo cargar la información catálogo, que será base para realizar cálculos en la fase de agregaciones. Por lo tanto, se cuentas con unas tablas locales (N\_dayligth\_hours, radicion\_dia\_despejado, radiación\_global y constante\_ra), las cuales son cargadas al servidor en sus respectivas tablas del repositorio RedmetRaw.

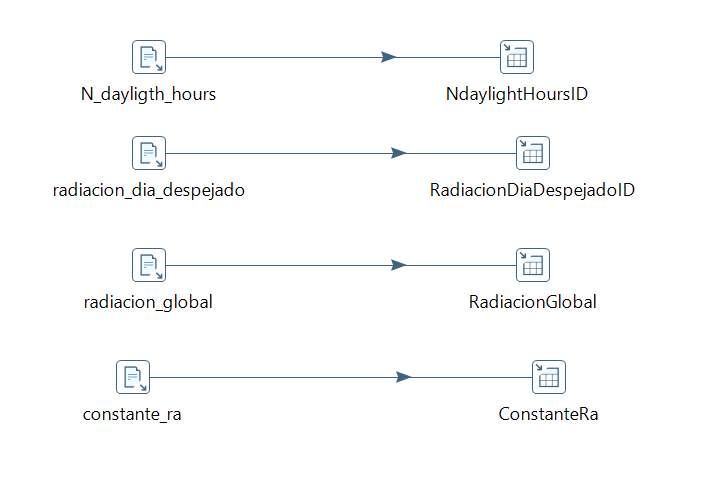
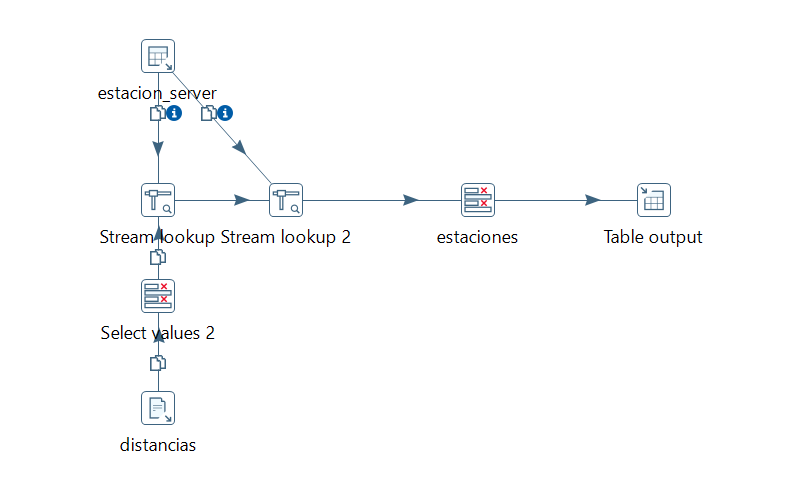


Ilustración 3 Carga de tablas catálogo

**Tiempo de ejecución:** Este flujo se ha llevado a cabo para cargar la información local de las tablas catalogo almacenadas en el equipo. En otras palabras, esta ejecución es única.



**Ilustración 4 Carga de tabla de distancias en el servidor**

La información esencial para ejecutar los procesos de transformación de datos incluye la tabla de distancias, calculada a través del método de vecinos más cercanos aplicado a la clasificación de clústeres. Para la carga de esta tabla, la transformación "distancias\_server.ktr" incorpora los siguientes componentes:

1. **Text file input (distancias):** Este componente se encarga de leer el archivo CSV que contiene información sobre las estaciones, sus estaciones cercanas, las distancias y el clúster al que pertenecen.
2. **Select values2:** En este componente, se renombran las columnas "estacion" y "estaciones\_cercanas" como "Codigo" y "código2", respectivamente. Este ajuste se realiza para facilitar el cruce de tablas en el componente de "stream lookup".
3. **Table input (estación\_server):** Aquí, se carga la tabla "Estacion" del esquema "RedmetRaw" del servidor.
4. **Stream lookup:** A través de este componente, se realiza el cruce entre la tabla "Estacion" y la tabla de distancias. El objetivo es asignar códigos de identificación a los nombres de las estaciones presentes en la tabla de distancias.
5. **Stream lookup2:** Dado que la columna "estaciones\_cercanas" también requiere identificación numérica, se establece una conexión entre la tabla "Estacion" (componente Table input: estación\_server) y este componente. Esto permite asignar identificadores a los nombres de las estaciones.
6. **Select values (estaciones):** En este componente, se ajusta el formato de la columna de distancia para aceptar dos cifras decimales.
7. **Table output:** Este componente se encarga de insertar la información de las distancias, adaptada con los ID de las estaciones, en la tabla "Distancias" ubicada en el esquema "RedmetRaw" del servidor.

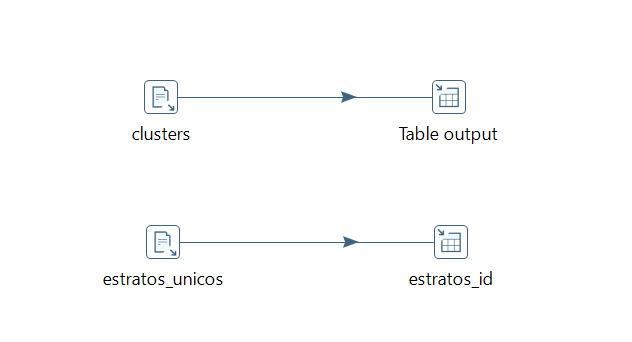


Ilustración 5 Carga de información única de las estaciones

Entre la información adicional directamente relacionada con los datos de las estaciones meteorológicas, se encuentran los estratos y los clusters. Para incorporar esta información de manera adecuada, se ha añadido a sus respectivas tablas en el esquema "RedmetRaw" del servidor.

Las transformaciones "cluster\_unicos.ktr" y "estratos\_unicos.ktr" incluyen como componentes el "text input", específicamente "clusters" y "estratos\_unicos". Estos componentes cargan archivos en formato TXT y CSV, respectivamente, que contienen exclusivamente información sobre los estratos y clusters existentes. Por ejemplo, la tabla de estratos consta de una columna con las categorías (bajo, alto, medio y litoral), mientras que la tabla de clusters contiene una columna con las clasificaciones (cluster0, cluster1, cluster2, cluster3, cluster4, cluster5).

Estos "text input" se conectan a sus correspondientes "table output", los cuales representan las tablas alojadas en el servidor.

**Tiempo de ejecución:** Este flujo se ha llevado a cabo para cargar la información local de las tablas catalogo almacenadas en el equipo. En otras palabras, esta ejecución es única. Llegado el caso se agreguen nuevas estaciones a la base de datos, la clasificación de clusters y estratos se deberá cargar nuevamente.

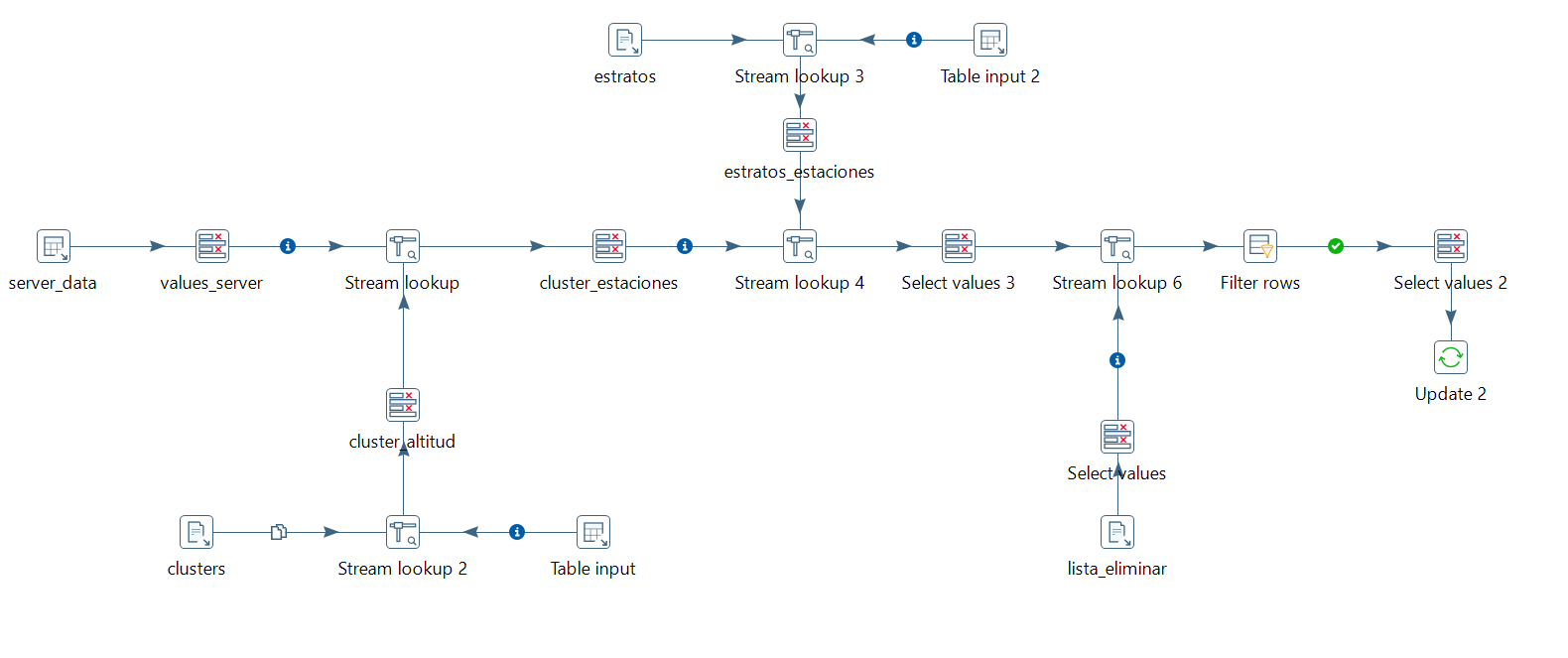


Ilustración 6 Actualización de data en la tabla Estacion del servidor

El propósito de esta transformación es añadir la clasificación de estratos y clusters a las estaciones, ya que esta información es única para cada estación. Las tablas de estratos y clusters tienen una columna con un código de identificación único para cada clasificación, y estos códigos se agregarán a la tabla "Estacion" en el esquema "RedmetRaw" del servidor. La transformación "cluster\_estratos.ktr" consta de los siguientes componentes:

1. **Text input (clusters):** Lee un archivo CSV que contiene los nombres de las estaciones, los clusters en los que están clasificadas y la altitud de cada estación.
2. **Table input:** Carga la tabla "Cluster" que reside en el servidor.
3. **Stream lookup:** Cruza la tabla local de clusters con la tabla del servidor para obtener los códigos ID de los clusters y asignarlos a las estaciones.
4. **Select values (cluster altitud):** Selecciona las columnas necesarias y renombra la variable de altitud.
5. **Table input (server data):** Carga la tabla "Estacion" del esquema "RedmetRaw" del servidor.
6. **Select values (values\_server):** Selecciona las variables necesarias de la tabla "Estacion".
7. **Stream lookup:** Cruza la tabla "Estacion" con la tabla de clusters para agregar los ClusterID a sus respectivas estaciones.
8. **Select values (cluster\_estaciones):** Selecciona todas las variables necesarias que complementan la tabla "Estacion".
9. **Text input (estratos):** Lee un archivo CSV que contiene los nombres de las estaciones y los estratos en los que están clasificadas.
10. **Table input2:** Carga la tabla "Estratos" que se encuentra en el servidor.
11. **Stream lookup3:** Cruza la tabla local de clusters con la tabla del servidor para obtener los códigos ID de los clusters y asignarlos a las estaciones.
12. **Select values (estratos\_estaciones):** Selecciona las variables necesarias del cruce anterior.
13. **Stream lookup4:** Cruza la tabla que viene del servidor junto con los clusters.
14. **Select values3:** Selecciona las variables que formarán parte de la tabla final.
15. **Text input (lista\_eliminar):** Carga un archivo de texto con la lista de estaciones que serán eliminadas de la tabla, ya que solo se incluyen estaciones meteorológicas de Guatemala.
16. **Select values:** Renombra la columna de la lista de estaciones a eliminar.
17. **Stream lookup6:** Cruza la tabla que se está construyendo en el flujo con la lista de estaciones que serán eliminadas.
18. **Filter rows:** Filtra las estaciones que serán eliminadas de la tabla.
19. **Select values2:** Selecciona las columnas que conformarán la tabla final.
20. **Update2:** Este componente actualiza la tabla "Estacion" en el servidor, es decir, carga solo la información nueva en la tabla.

**Tiempo de ejecución:** Este flujo se ha llevado a cabo para cargar la información local de las tablas catalogo almacenadas en el equipo. En otras palabras, esta ejecución es única. Llegado el caso se agreguen nuevas estaciones a la base de datos, la clasificación de clusters y estratos se deberá cargar nuavemente.

# **Fase control de calidad**

A screenshot of a computer

Description automatically generated

Ilustración 7 Preparación de la data

Esta etapa de "Control de Calidad" se encarga de preparar los datos previamente cargados en la tabla "EstacionLectura" del repositorio RedmetRaw, los cuales serán sometidos a un proceso de control de calidad. Este proceso implica la organización de la información y la incorporación de las columnas necesarias que permitirán la aplicación de filtros de verificación y corrección. Para llevar a cabo esta primera preparación de los datos, la transformación está compuesta de la siguiente manera:

1. **Table Input (Data\_cruda):** Este componente se encarga de cargar la tabla "EstacionLectura" ubicada en el repositorio RedmetRaw. Dado que esta tabla alberga tanto la data histórica cruda como los nuevos registros crudos, se deben establecer las fechas a las cuales se les realizará el control de calidad.
2. **Text File Input (fechas\_por\_correccion):** Este componente carga un archivo en formato txt para establecer las fechas en las cuales se desea llevar a cabo el control de calidad, como se mencionó en el componente anterior.
3. **Select Values:** En este componente, se ajusta el formato de la columna fecha a (yyy/MMydd HH:mm).
4. **Calculator (Segmentación fecha):** Este componente se encarga de segmentar la fecha, creando nuevas columnas como semana (indica el número de la semana del año), día (indica el número del día del mes), mes (indica el número del mes), año (indica el número del año), date (establece la fecha sin el formato de tiempo) y hora (indica el número de la hora).
5. **Table Input (Info\_estacion):** Este componente carga la tabla "Estacion" que se encuentra en el repositorio RemetRaw.
6. **Stream Lookup:** Encargado de combinar la tabla de datos crudos que se someterán a control de calidad con la tabla "Estacion". Este paso es esencial para enriquecer la información de las estaciones, preparándola para la aplicación de filtros de control en fases posteriores.
7. **Select Values 2:** En este componente, se lleva a cabo un ajuste en la columna "Codigo", renombrándola como "Estacion", mientras que la columna "ClusterID" se renombra como "Cluster" y se modifica su tipo de formato de entero a cadena.
8. **Replace in String:** Este componente se encarga de reemplazar los valores numéricos de la columna "Cluster" (1, 2, 3, 4, 5, 6) con agrupaciones de texto (Cluster0, Cluster1, Cluster2, Cluster3, Cluster4, Cluster5).
9. **Avro Output:** Este componente tiene la responsabilidad de almacenar la tabla de datos crudos que están a punto de ser sometidos a un control de calidad. La tabla generada en este componente se guarda en formato Avro.

**Tiempo de ejecución**: Durante la ejecución de esta transformación, se generará un archivo local en formato .avro. Este archivo se considerará temporal en el contexto de la operacionalización de la ETL y se eliminará en un plazo determinado. Su función es fundamental en el proceso de control de calidad, pero al tratarse de una tabla que no necesita ser conservada, se gestionará como un componente transitorio del flujo de trabajo. En cuanto a la frecuencia de ejecución de esta transformación, se llevará a cabo de forma diaria, permitiendo así que el control de calidad se opere de manera efectiva cada día.

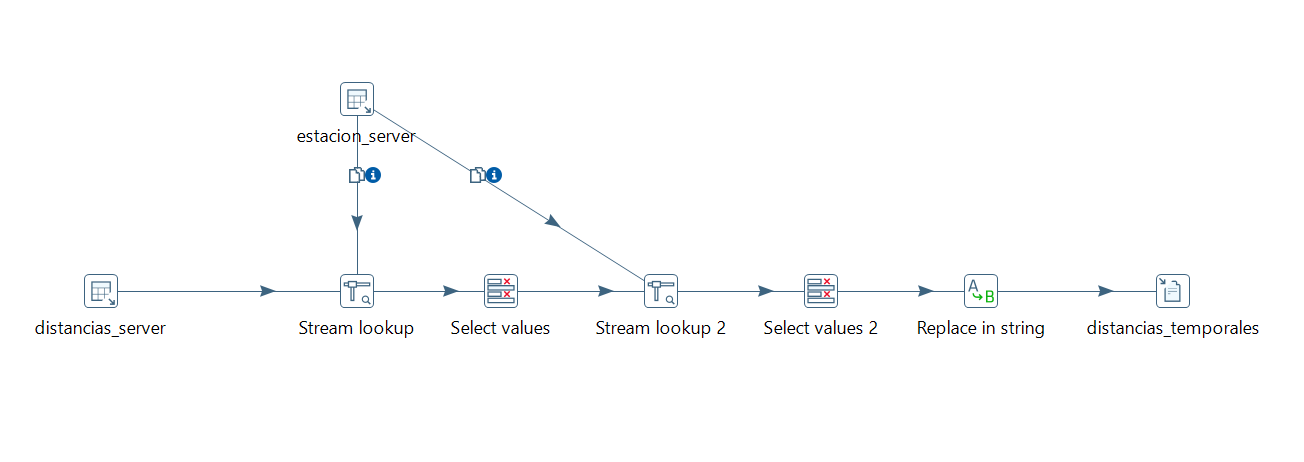
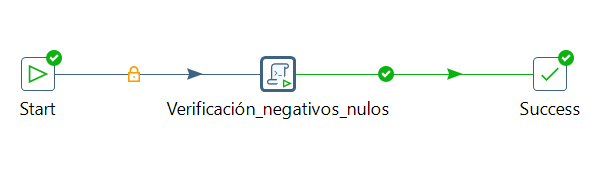


Ilustración 8 Tabla de distancias

Dentro de los flujos destinados al control de calidad, se encuentra la tabla de distancias, la cual fue detalladamente explicada en la Fase de Carga de Información del Catálogo. Dado que el control de calidad implica la ejecución de scripts en Python, y las funciones en Python requieren que las tablas tengan un formato específico, se ha desarrollado la transformación "distancias\_vecinos" con el objetivo de generar una tabla temporal que se adapte al código. Esta transformación consta de los siguientes componentes:

1. **Table input (distancias\_server):** Carga la tabla de distancias ubicada en el esquema "RedmetRaw" del servidor.
2. **Table input (estacion\_server):** Carga también la tabla "Estacion" del servidor.
3. **Stream lookup:** Cruza las dos tablas mencionadas anteriormente para identificar las estaciones no por su código ID, sino por su nombre.
4. **Select values:** Convierte a tipo string los códigos ID de la tabla de clusters, ya que la tabla de funciones en Python requiere que estos sean de este tipo.
5. **Stream lookup 2:** En este cruce, se cambian los códigos ID de la columna "estaciones cercanas" ya que lo necesario es el nombre de la estación.
6. **Select values 2:** Se renombran algunas columnas en este componente.
7. **Replace in string:** Se reemplazan los valores numéricos de los ID de los clusters, dejándolos como texto (1: cluster0, 2: cluster1, etc.).
8. **Text output (distancias temporales):** Finalmente, en este componente se genera un archivo de texto con la tabla en el formato necesario para ser utilizada en la fase de control de calidad.

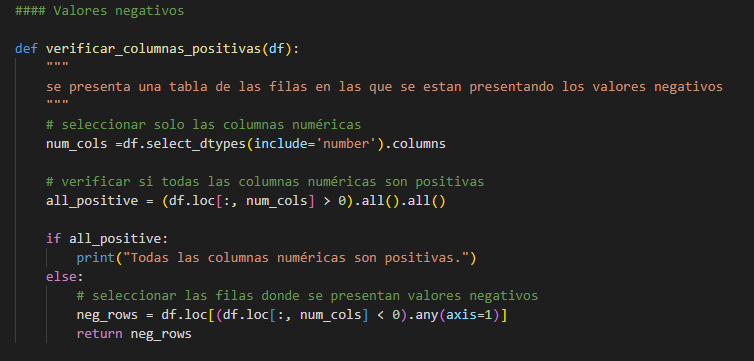
**Tiempo de ejecución:** Dado que esta tabla forma parte integral del proceso de control de calidad, se implementará durante la fase de operacionalización. En otras palabras, si el proceso de calidad está programado para llevarse a cabo diariamente, se generará esta tabla temporal de manera diaria. Sin embargo, es importante destacar que una vez cumplida su función, no se almacenará.

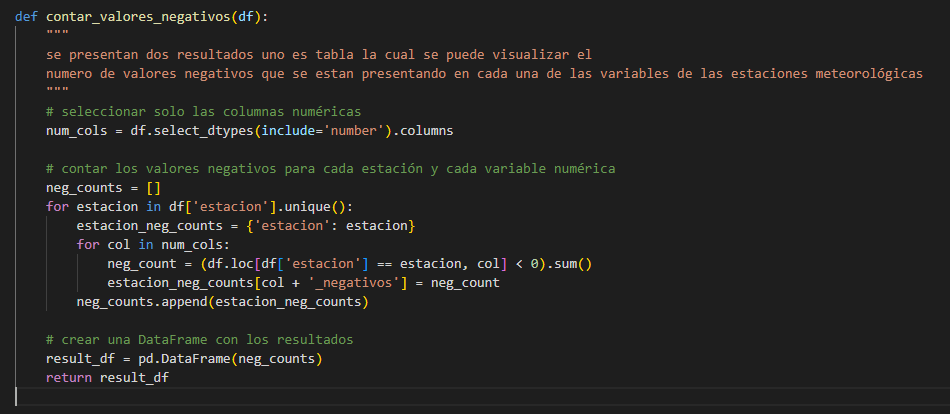


El módulo de verificación de negativos y nulos tiene como función principal orquestar un archivo .py que contiene diversas funciones. Estas funciones son diseñadas para analizar la información previamente preparada y determinar el estado de dichos datos. En otras palabras, este módulo se encarga de informar qué estaciones están registrando valores negativos en las variables, así como de identificar la presencia de valores nulos consecutivos y proporcionar información sobre su cantidad y otros detalles relevantes.

A continuación, se presentan las funciones definidas en el archivo .py:

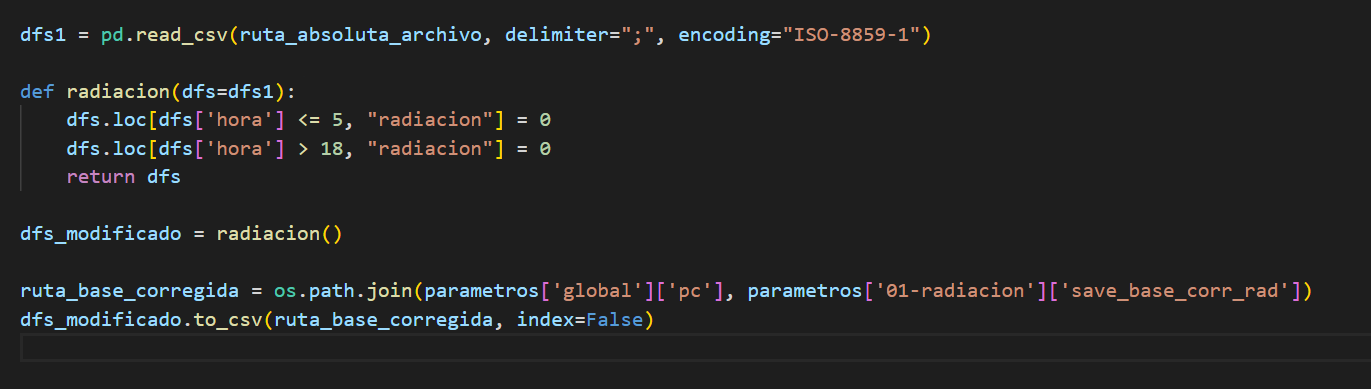
La función **Verificar\_columnas\_positivas** se ha diseñado como parte esencial del proceso de control de calidad de los datos climáticos recopilados por estaciones meteorológicas. Su objetivo principal es identificar la presencia de valores negativos en cada una de las variables que componen estos datos, permitiendo así una detección temprana de posibles errores en la captura o transmisión de información.



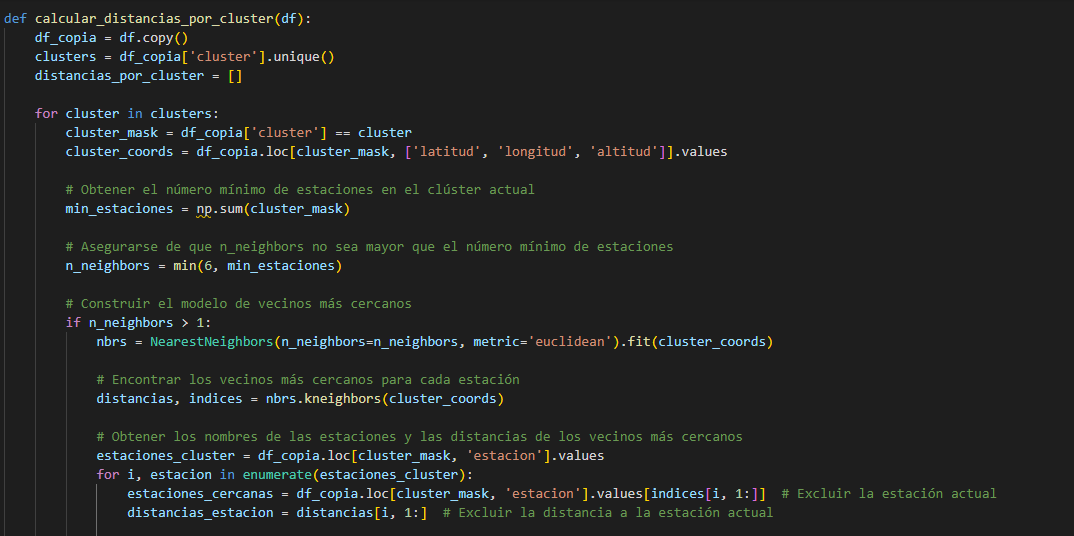


La función **contar\_valores\_negativos** ha sido diseñada como una herramienta esencial en el proceso de verificación de la calidad de los datos climáticos recopilados por las estaciones meteorológicas. Su propósito principal es generar un informe que cuantifique la presencia de valores negativos en cada una de las estaciones meteorológicas y en todas las variables numéricas registradas. En otras palabras, esta función complementa la verificación de los valores positivos y contribuye al informe final al proporcionar un recuento detallado de los valores negativos encontrados.

A continuación, se procede a realizar la corrección de errores sistemáticos, como el caso de la variable "radiación". En este contexto, la función denominada "radiacion" ha sido diseñada con el propósito de identificar y corregir todos los valores de radiación registrados durante las horas nocturnas. En otras palabras, esta función tiene la capacidad de localizar y ajustar los valores de radiación que se registran específicamente en este período. Su funcionamiento se describe mediante el siguiente código:



Esta función, al ser ejecutada, establece todos los valores de "radiación" registrados antes de las 5:00 a.m. y después de las 6:00 p.m. a un valor de 0, lo que corrige los errores relacionados con la radiación durante las horas nocturnas en los datos. Luego es guardada esta base con las correcciones realizadas en radiación.



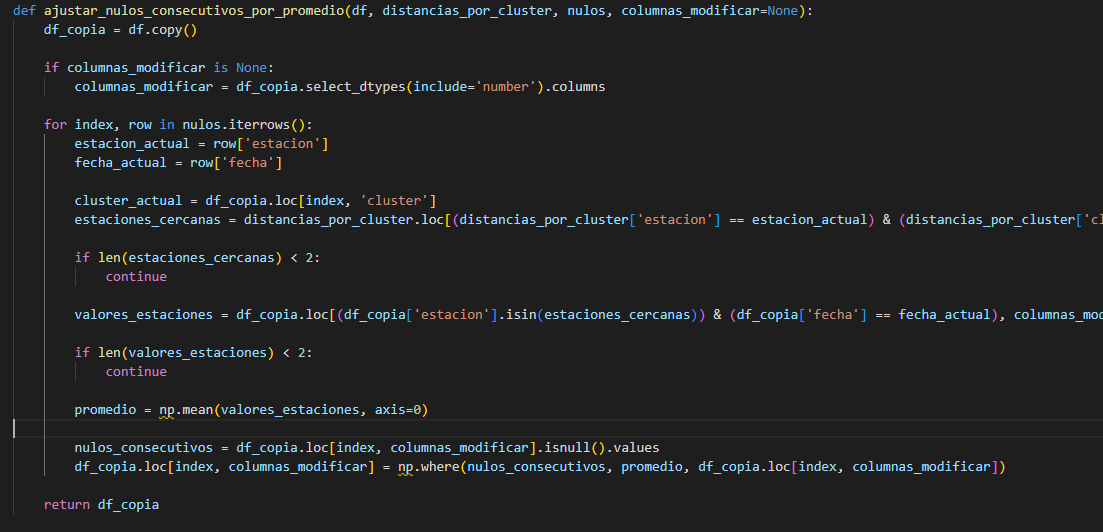
La **función calcular\_distancias\_por\_cluster**, está diseñada para calcular las distancias entre estaciones en el conjunto de datos de las estaciones.

1. **Copiar el DataFrame original:** Comienza copiando el DataFrame de entrada df en una nueva variable llamada df\_copia. Esto se hace para evitar modificar el DataFrame original durante el proceso.
2. **Obtener los clusters únicos:** La función identifica los valores únicos en la columna 'cluster' del DataFrame df\_copia y los almacena en la variable clusters. Esto sugiere que los datos están organizados en grupos o clusters, y se desea calcular las distancias dentro de cada uno de estos grupos.
3. **Inicializar una** **lista vacía para las distancias:** La variable distancias\_por\_cluster se inicializa como una lista vacía. Esta lista se utilizará para almacenar las distancias entre estaciones dentro de cada cluster.
4. **Iterar a través de los clusters:** La función recorre cada uno de los clusters identificados en el paso 2.
5. **Filtrar el DataFrame por cluster:** Para cada cluster, se crea una máscara booleana llamada cluster\_mask que selecciona las filas del DataFrame donde el valor de 'cluster' coincide con el cluster actual.
6. **Extraer las coordenadas del cluster:** Se extraen las coordenadas ('latitud', 'longitud' y 'altitud') de todas las estaciones que pertenecen al cluster actual y se almacenan en la variable cluster\_coords.
7. **Determinar el número mínimo de estaciones**: Se calcula el número mínimo de estaciones en el cluster actual utilizando la suma de la máscara booleana cluster\_mask. Esto se almacena en min\_estaciones.
8. **Limitar el número de vecinos más cercanos**: Se establece el número de vecinos más cercanos (n\_neighbors) como el mínimo entre 6 y el número mínimo de estaciones en el cluster. Esto limita el número de vecinos a calcular y asegura que no se soliciten más vecinos de los disponibles.
9. **Construir el modelo de vecinos más cercanos**: Se utiliza la biblioteca scikit-learn para construir un modelo de vecinos más cercanos (NearestNeighbors) utilizando las coordenadas de las estaciones en el cluster actual.
10. **Encontrar vecinos más cercanos**: Se encuentran los vecinos más cercanos para cada estación en el cluster actual utilizando el modelo de vecinos más cercanos. Se almacenan las distancias y los índices de las estaciones vecinas.
11. **Crear un DataFrame de distancias**: Para cada estación en el cluster, se crea un DataFrame que contiene información sobre la estación actual, las estaciones cercanas, las distancias y el cluster al que pertenecen.
12. **Agregar los DataFrames a la lista:** Estos DataFrames individuales se agregan a la lista distancias\_por\_cluster que se creó en el paso 3.
13. **Concatenar los DataFrames:** Finalmente, si la lista distancias\_por\_cluster no está vacía, se concatenan todos los DataFrames individuales en uno solo utilizando pd.concat(). Esto crea un DataFrame único que contiene las distancias entre estaciones dentro de todos los clusters.
14. **Devolver el DataFrame de distancias**: La función devuelve el DataFrame final que contiene las distancias entre estaciones dentro de cada cluster.



La función ajustar\_negativos\_por\_promedio está diseñada para ajustar valores negativos en la base de datos según el promedio de valores en estaciones cercanas en el mismo cluster.

1. **Copiar el DataFrame original:** La función comienza copiando el DataFrame de entrada df en una nueva variable llamada df\_copia. Esto se hace para evitar modificar el DataFrame original durante el proceso.
2. **Definir columnas a modificar:** Si no se proporciona un conjunto específico de columnas para modificar (columnas\_modificar), la función selecciona automáticamente todas las columnas numéricas del DataFrame df\_copia. Esto se hace utilizando select\_dtypes(include='number').columns.
3. **Iterar a través de las filas con valores negativos:** La función itera a través de las filas de un DataFrame llamado neg\_rows, que se espera que contenga información sobre filas que tienen valores negativos que deben ser ajustados.
4. **Obtener información de la estación actual:** Para cada fila en neg\_rows, se obtiene información sobre la estación actual y la fecha correspondiente.
5. **Identificar el cluster actual:** La función encuentra el cluster al que pertenece la estación actual utilizando df\_copia.loc[index, 'cluster'].
6. **Encontrar estaciones cercanas en el mismo cluster:** Se busca en el DataFrame distancias\_por\_cluster para encontrar las estaciones cercanas a la estación actual que están en el mismo cluster. Esto se hace filtrando las filas donde la 'estacion' coincide con la estación actual y el 'cluster' coincide con el cluster actual. Los nombres de las estaciones cercanas se almacenan en estaciones\_cercanas.
7. **Verificar la cantidad de estaciones cercanas:** Si hay menos de 2 estaciones cercanas, no se realiza ningún ajuste y la función continúa con la siguiente fila.
8. **Obtener los valores de las estaciones cercanas:** Para cada estación cercana, se obtienen los valores correspondientes de las columnas a modificar en la fecha actual. Estos valores se almacenan en valores\_estaciones.
9. **Verificar la cantidad de valores obtenidos:** Si hay menos de 2 conjuntos de valores (uno para la estación actual y al menos uno para una estación cercana), no se realiza ningún ajuste y la función continúa con la siguiente fila.
10. **Calcular el promedio de los valores de estaciones cercanas:** Los valores obtenidos de las estaciones cercanas se convierten en un arreglo NumPy y luego se calcula el promedio de estos valores a lo largo del eje 0. Esto significa que se calcula un promedio por columna.
11. **Realizar el ajuste:** Para la fila actual en el DataFrame df\_copia, se verifica si algún valor es menor que cero. Si es menor que cero, se reemplaza por el valor promedio calculado en el paso anterior. De lo contrario, los valores se mantienen sin cambios.
12. **Retornar el DataFrame ajustado:** Una vez que se han ajustado todos los valores negativos en el DataFrame df\_copia, se devuelve este DataFrame modificado como resultado de la función.



Esta función es similar a 'ajustar\_negativos\_por\_promedio', pero ajusta valores nulos (NaN) en lugar de valores negativos, utilizando el promedio de valores en estaciones cercana dentro del mismo cluster.

Teniendo en cuenta que la función ajustar\_nulos\_consecutivos\_por\_promedio, se utiliza para manejar los valores nulos en los datos y reemplazarlos por valores calculados a partir de estaciones cercanas en el mismo cluster. La imputación por media móvil busca poder suplir aquellos nulos que no fueron corregidos. Es decir que, la imputación por media móvil se aplica después de que la función anterior ha hecho parte de la correción. La idea detrás de esta imputación es tratar de llenar los valores nulos restantes que no fueron corregidos por la función anterior. Esto se hace calculando valores promedio basados en datos históricos dentro de una ventana de tiempo móvil, lo que ayuda a estimar los valores faltantes en función de tendencias pasadas.

A screen shot of a computer code

Description automatically generated

1. **Definición de las variables:** Se crea una lista llamada variables que contiene los nombres de las variables que se imputarán en el DataFrame. Estas variables son: 'temperatura', 'radiacion', 'humedad\_relativa', 'precipitacion', 'velocidad\_viento', 'mojadura' y 'direccion\_viento'.
2. **Copia del DataFrame:** Se crea una copia del DataFrame original llamado df\_ajusta1 y se almacena en una nueva variable llamada df\_tabla. Esto se hace para asegurarse de que la imputación se realice en una copia y no afecte al DataFrame original.
3. **Definición de la función de imputación por estación:** Se define una función llamada imputar\_por\_estacion que toma un grupo de datos (que corresponde a una estación meteorológica específica) como entrada. La función realiza la imputación de valores faltantes para las variables en la lista variables utilizando un enfoque de ventana deslizante.
4. **Bucle anidado:** Se utiliza un bucle anidado que recorre diferentes tamaños de ventana (de 2 a 6) y luego recorre las variables en la lista variables.
5. **Comprobación de valores faltantes**: Para cada variable en la lista, se verifica si existen valores faltantes en el grupo de datos correspondiente a la estación actual. Si hay valores faltantes, se procede con la imputación.
6. **Creación de una columna imputada:** Se crea una nueva columna en el grupo de datos con un nombre que incluye el nombre de la variable, la palabra 'imputed' y el tamaño de la ventana actual. Por ejemplo, si la variable es 'temperatura' y la ventana es 2, la columna se llamará 'temperatura\_imputed\_2'. Esta columna se llena utilizando el método rolling con la ventana especificada y se calcula la media móvil.
7. **Imputación en el lugar:** Se imputan los valores faltantes en la columna original de la variable utilizando los valores de la columna imputada creada en el paso anterior. Esto se hace utilizando el método fillna en la columna original.
8. **Retorno del grupo de datos:** La función retorna el grupo de datos (para una estación específica) con los valores faltantes imputados.
9. **Obtención de la lista de estaciones meteorológicas únicas:** Se obtiene una lista de las estaciones meteorológicas únicas presentes en el DataFrame df\_tabla y se almacenan en la variable estaciones.
10. **Aplicación de la imputación por estación:** Se aplica la función imputar\_por\_estacion a cada grupo de datos correspondiente a una estación meteorológica utilizando el método groupby. Esto significa que la función se ejecutará por separado para cada estación y se imputarán los valores faltantes dentro de cada estación.

# **Fase de consultas**

En esta fase se explica como realizar una consulta en el servidor para visualizar la tabla de agregaciones diarias la cual contiene los cálculos de las variables por el periodo de un día.

La tabla de agregaciones diarias se llama “GetAgregacionDiaria2”, al llamar la tabla se ingresa el periodo de fechas que se desea visualizar como se muestra a continuación en la Ilustración 9.

# 

Ilustración 9 Tabla Agregaciones diarias

Adicionalmente si se desea hacer una consulta más específica, como el poder visualizar los resultados de una estación en especifico se puede hacer como se presenta en la ilustración 10.

**A screenshot of a computer

Description automatically generated**

Ilustración 10 Consultas especificas