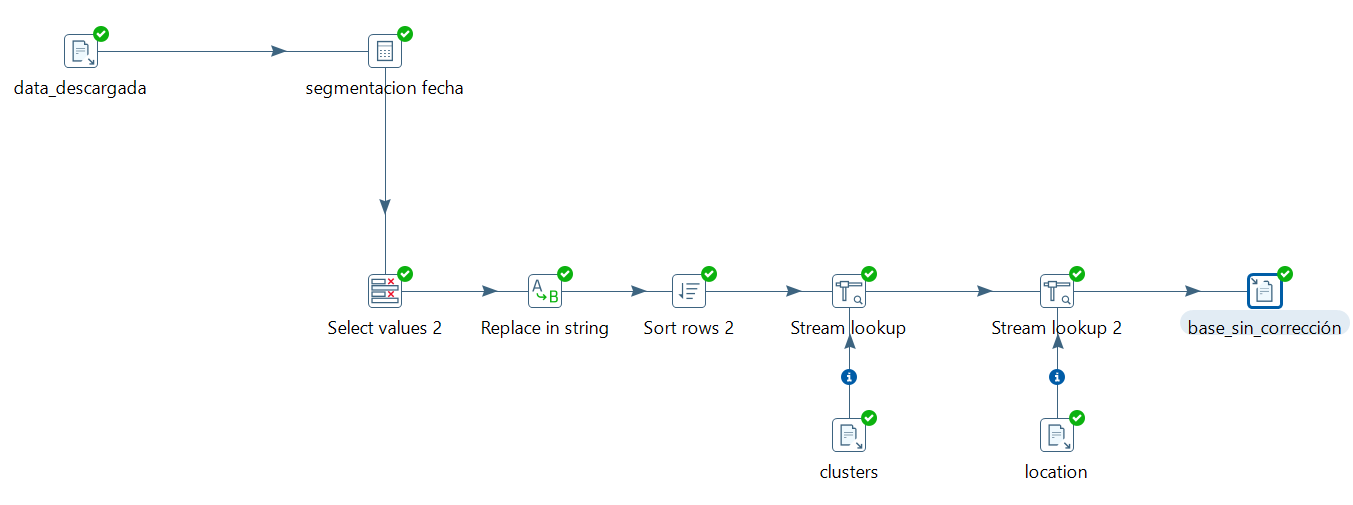
**Fase control de calidad**

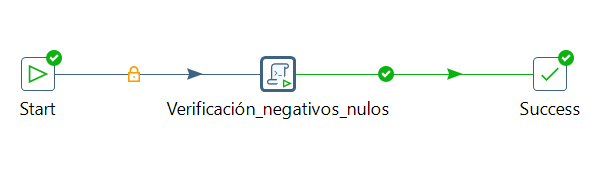
****

El módulo de "Control de Calidad", se encarga de preparar los datos que han sido previamente descargados y que serán sometidos a un proceso de control de calidad. Esto implica organizar la información y agregar las columnas necesarias que permitirán aplicar los filtros de verificación y corrección.

Para llevar a cabo este proceso, primero se realiza la lectura de la información descargada. A continuación, se aplica un componente que segmenta la fecha en semana, día, mes, año, date y hora. Utilizando el componente "Select Value", se ajustan los nombres de las variables y se aplican los formatos necesarios.

Luego, mediante el componente "Replace in String", se lleva a cabo la modificación de los nombres de las estaciones meteorológicas, eliminando caracteres especiales como guiones y tildes. Posteriormente, se utiliza el componente "Sort Rows" para ordenar la información de manera adecuada.

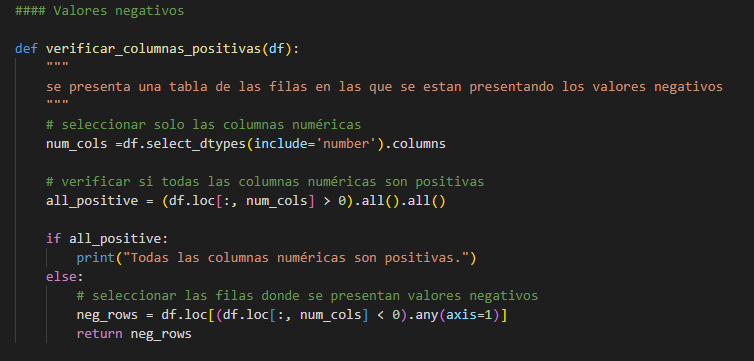
Después de la ordenación, se agrega una columna de "clusters" que indica a qué grupo pertenece cada estación. Seguidamente se agregan las columnas de localización Enel componente location. Finalmente, la base de datos se guarda en un archivo de texto (txt) para su utilización en los siguientes filtros de control y corrección.

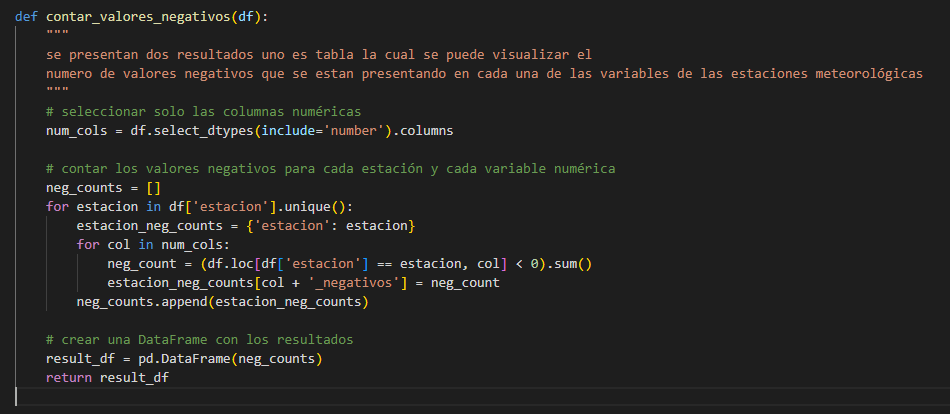


El módulo de verificación de negativos y nulos tiene como función principal orquestar un archivo .py que contiene diversas funciones. Estas funciones son diseñadas para analizar la información previamente preparada y determinar el estado de dichos datos. En otras palabras, este módulo se encarga de informar 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:

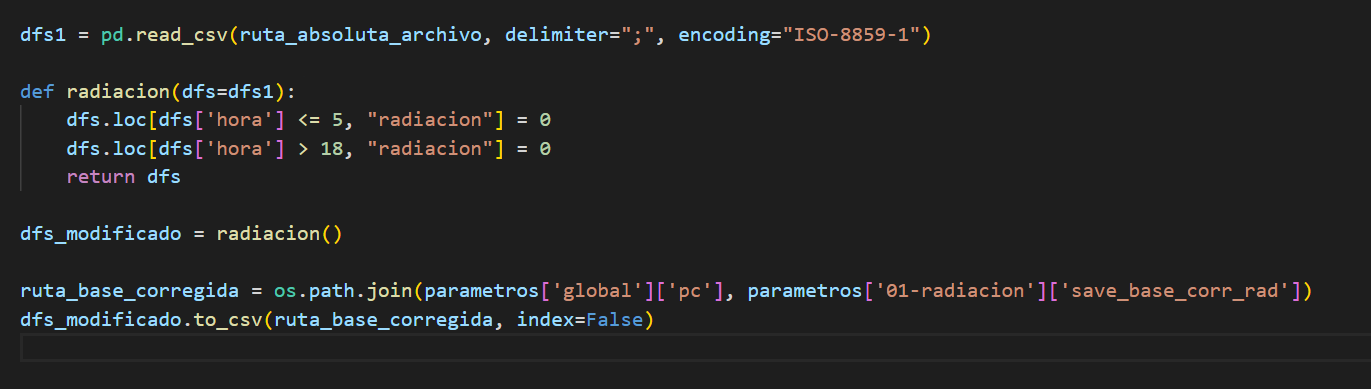
La función **Verificar\_columnas\_positivas** se ha diseñado como parte esencial del proceso de control de calidad de los datos climáticos recopilados por estaciones meteorológicas. Su objetivo principal es identificar la presencia de valores negativos en cada una de las variables que componen estos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



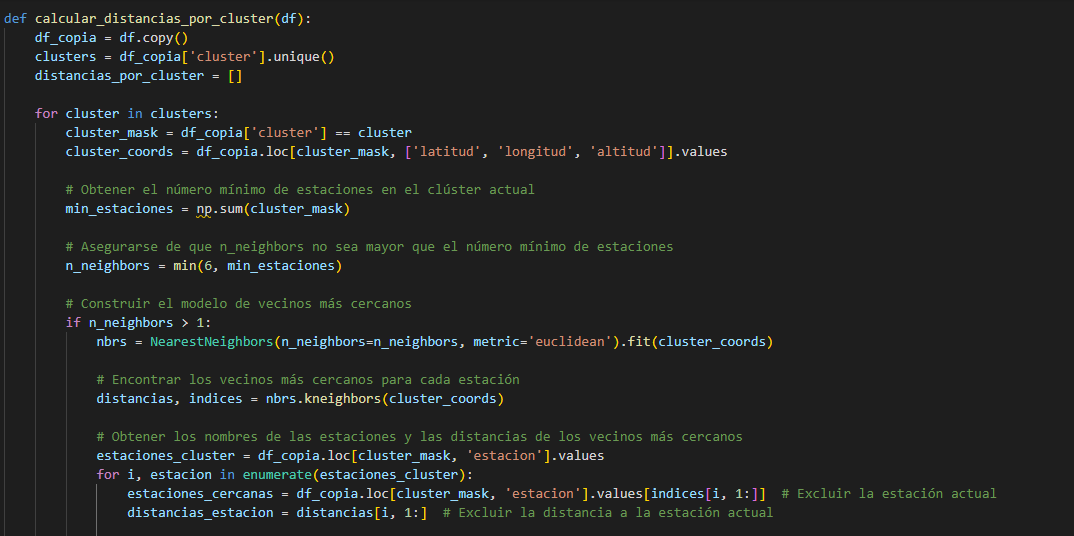


La función **contar\_valores\_negativos** ha sido diseñada como una herramienta esencial en el proceso de verificación de la calidad de los datos climáticos recopilados por las estaciones meteorológicas. Su propósito principal es generar un informe que cuantifique la presencia de valores negativos en 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.



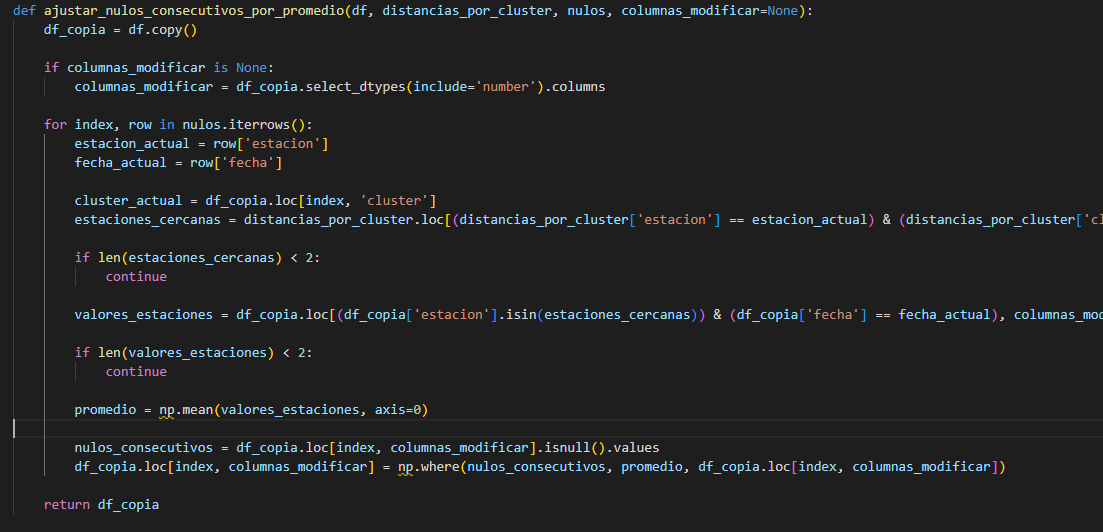
La **función calcular\_distancias\_por\_cluster**, está diseñada para calcular las distancias entre estaciones en el conjunto de datos de las estaciones.

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.