ETL ICDF

Desarrollar un proceso ETL (Extracción, Transformación y Carga) robusto y eficiente para recolectar datos de las estaciones automáticas de monitoreo climático, limpiar y transformar dichos datos para asegurar su calidad, y finalmente integrarlos en la App Agrointeligente para su uso en la toma de decisiones agrícolas.

El proceso se ejecuta cada hora según los procesos de carga de datos nuevos de las estaciones.

{Parámetros de ejecución: Como archivos de fechas, conexiones, carpetas de entrada, salida, kettle.properties, etc}

Repositorio de código en GIT: https://github.com/alliance-datascience/agrilac-gt-icdf

{Como se ejecuta} Emplea Python para toda su ejecución.

Componentes del proceso

1. Extracción

1.1 Librerías requeridas y otros

```
import json
import requests
import time
from datetime import datetime
```

Acá se puede observar las principales librerías de Python requeridas para que el script.

Se importan los módulos json, requests, time y datetime para manejar datos JSON, hacer solicitudes HTTP, trabajar con marcas de tiempo y gestionar fechas y horas, respectivamente

1.2 Ingreso de credenciales

```
# Your WeatherLink Account API Credentials
API_KEY = ""
API_SECRET = ""
```

En API_KEY y API_SECRET se colocan los token previamente generados.

La clave API es una identificación única que identifica al usuario de API que realiza la llamada API. La clave API se debe pasar como parámetro de consulta en todas las solicitudes de API.

La API WeatherLink v2 utiliza una clave API y un secreto API para autenticar solicitudes API.

Secreto de API es un valor secreto que se pasa como un encabezado denominado X-Api-Secret (no distingue entre mayúsculas y minúsculas).

Si su secreto de API se ve comprometido, permitirá que otros accedan a la API haciéndose pasar por usted. Si esto ocurre, regrese a la página Cuenta en https://www.weatherlink.com/account y haga clic nuevamente en el botón Generar clave v2. Esto eliminará el antiguo secreto de API y creará un nuevo secreto de API. Luego deberá actualizar su código para utilizar el nuevo API Secret.

Tenga en cuenta que la clave API no se cambiará y no se podrá eliminar al generar un nuevo secreto API.

1.3 Definición de Endpoint

```
# API v2 Endpoint URL
# https://weatherlink.github.io/v2-api/api-reference
endpoint = "historic"
```

Se puede elegir, según lo que se requiera entre:

- historic

```
{
    "station_id_uuid": "66124387-f696-4ec8-8bae-a65e19eb36fa",
    "sensors": [
        {
            "lsid": 659098,
            "data": [
                {
                    "tz_offset": -21600,
                    "moist_soil_last": 140.9,
                    "ts": 1716476400
                },
                {
                    "tz_offset": -21600,
                    "moist_soil_last": 141.3,
                    "ts": 1716480000
                },
                {
                    "tz_offset": -21600,
                    "moist_soil_last": 141.1,
                    "ts": 1716483600
                }
            ],
            "sensor_type": 108,
            "data_structure_type": 9
        },
```


Figura 3: Ejemplo de Información de la estación 169529 utilizando ‘historic’.

- sensors: El campo sensor_types es una matriz de objetos JSON, cada uno de los cuales contiene información de metadatos sobre un tipo de sensor.

```
{
    "sensor_type": 115,
    "manufacturer": "Sentek",
    "product_name": "Drill & Drop 60 cm (24\"); 6 sensors",
    "product_number": "00620",
    "category": "Soil Moisture",
    "configuration": {
        "depth_unit": "in",
        "depths": {
            "ring1": 4,
            "ring2": 8,
            "ring3": 12,
            "ring4": 16,
            "ring5": 20,
            "ring6": 24
        }
    },
    "data_structure": {
        "moist_soil_last_1": {
            "type": "float",
            "units": "percent"
        },
```


Figura 2: Ejemplo de Información utilizando ‘sensors’.

- current: información en el momento

```
{
    "station_id_uuid": "66124387-f696-4ec8-8bae-a65e19eb36fa",
    "sensors": [
        {
            "lsid": 659098,
            "data": [
                {
                    "tz_offset": -21600,
                    "moist_soil_last": 141.1,
                    "ts": 1716483600
                }
            ],
            "sensor_type": 108,
            "data_structure_type": 9
        },
        {
            "lsid": 659099,
            "data": [
                {
                    "tz_offset": -21600,
                    "moist_soil_last": 56.9,
                    "ts": 1716483600
                }
            ],
            "sensor_type": 108,
...
    ],
```


Figura 2: Ejemplo de Información de la estación 169529 utilizando ‘current’.

- stations

```
{
    "stations": [
        {
            "station_id": 169484,
            "station_id_uuid": "3fa66637-8d6b-4fa5-a703-33d4e60b435c",
            "station_name": "San Jer\u00f3nimo Misi\u00f3n Taiw\u00e1n",
            "gateway_id": 8429813,
            "gateway_id_hex": "001D0A80A0F5",
            "product_number": "6803A",
            "username": "ICDF",
            "user_email": "info@labconet.com",
            "company_name": "Laboratorio de Comunicaciones, S.A.",
            "active": true,
            "private": false,
            "recording_interval": 60,
            "firmware_version": null,
            "imei": "862771041006829",
            "registered_date": 1696625970,
            "subscription_end_date": 1728248370,
            "time_zone": "America/Guatemala",
            "city": "San Jer\u00f3nimo",
            "region": "Baja Verapaz Department",
            "country": "Guatemala",
            "latitude": 15.07635,
            "longitude": -90.25007,
...
        }
```


Figura 1: Ejemplo de metadata descargada con API usando ‘stations’.

En general una respuesta típica incluye los siguientes componentes:

- station_id: Es el identificador único de la estación meteorológica. Este ID se puede obtener utilizando el endpoint de la API de metadatos /stations.
- sensors: Es una lista de sensores asociados con la estación. Cada sensor en la lista contiene los datos de observación que recoge, y estos datos se explican con más detalle más adelante.
- generated_at: Es una marca de tiempo en formato Unix que indica cuándo se generó la respuesta de la API.

El formato de la respuesta de la API de WeatherLink v2 es JSON.

Cada sensor que aparece en la sección de sensores incluye la siguiente información:

- lsid: Este es el ID lógico del sensor, único para cada uno. Puedes obtener todos los IDs de sensores de tus estaciones usando el endpoint de la API de metadatos /sensors.
- sensor_type: El tipo de sensor. Este dato se puede usar para consultar el Catálogo de Sensores, donde encontrarás información sobre los nombres de los campos y los tipos de datos que cada tipo de sensor puede registrar.
- data_structure_type: Este campo indica el tipo de estructura de datos y ayuda a especificar la naturaleza del registro de datos de observación meteorológica, especialmente cuando el sensor puede generar varios tipos de registros. Para más detalles, se puede consultar la página de Tipos de Estructuras de Datos.
- data: Es una lista (o matriz) de registros de datos de observación meteorológica. Para los datos de condiciones actuales, la lista puede contener uno o ningún registro. Para los datos históricos, la lista puede contener desde cero hasta muchos registros, dependiendo del período de tiempo que se haya solicitado.

Los dispositivos WeatherLink Live incluyen un campo adicional en el registro de datos llamado **tx_id**. Este campo es específico para estos dispositivos y representa el ID del transmisor con el que el sensor está configurado en ese momento. El ID del transmisor no se almacena históricamente, así que siempre verás el valor actual del transmisor, incluso si se consulta datos históricos.

1.4 Definición de estaciones a descargar

```
# API Path Parameters
# Add the necessary _path_ parameters necessary for the API endpoint that you are querying
pathParameters = [
  169484,
  169523,
  169524,
  169529,
  178223
]
```

Se realiza un diccionario con los ID de las estaciones. Solo lo que esté definido en este espacio será los datos que se descarguen.

**Si se tiene una estación nueva es necesario agregarla en este diccionario.

1.5 Creación de timestamp para query

Parámetros de consulta que definen el rango de tiempo para los datos solicitados (desde una hora antes del tiempo actual hasta el tiempo actual).

```
# API Query String Parameters
# Add the necessary _query string_ parameters necessary for the API endpoint that you are querying
queryParameters = [
  ("start-timestamp", str(int(time.time()) - 3600)),
  ("end-timestamp", str(int(time.time())))
]
```

1.6 Bucle para la generación del request de la API

- Se itera a través de cada ID de estación en pathParameters.
- Para cada ID, se construye la URL de la solicitud incluyendo el endpoint, el ID de la estación, la clave de API y los parámetros de consulta.
- Se realiza la solicitud HTTP GET a la API utilizando la URL construida y se añade el secreto de la API en el encabezado de la solicitud.
- La respuesta de la API se convierte en un objeto JSON.

```
# Loop through each path parameter and make API requests
for pathParam in pathParameters:
    # Create final API URL for each path parameter
    api_url = BASE_URL + endpoint
    api_url += f"/{pathParam}"
    api_url += "?api-key=" + API_KEY
    api_url += ''.join("&" + str(x) + "=" + str(y) for (x, y) in queryParameters)[0:]

    # Make call to API and pretty-print returned data
    api_results = requests.get(
      headers={
        "X-Api-Secret": API_SECRET
      },
      url=api_url,
      verify=True,
    )
```

1.7 Generación de documento .json

- Se define la ruta de salida para guardar el archivo JSON correspondiente a cada ID de estación.
- Los datos JSON se guardan en la ruta especificada utilizando el formato JSON con una indentación de 4 espacios para mejorar la legibilidad.

Mensaje de Éxito:

- Se imprime un mensaje de éxito en la consola, indicando que el script se ejecutó correctamente para la estación actual, junto con la fecha y hora actuales.

```
# Parse the API response
api_data = json.loads(api_results.text)

# Define the output path for the JSON file
output_path = f"//{pathParam}.json"

# Save the JSON data to the specified path
with open(output_path, 'w') as json_file:
    json.dump(api_data, json_file, indent=4)

# Print a success message with the current time
current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
print(f"Script executed successfully for station {pathParam} at {current_time}")
```

1.8 Generación de documento .csv

Dado que los datos se encuentran en un archivo JSON, es necesario pasarlos a .CSV para los siguientes procesos. Por lo tanto, se tiene el siguiente script que funciona de manera parecida al anterior y permite generar un documento por ID facilitando la gestión y el análisis.

Primero, lectura de módulos(librerías) y toma el diccionario de ID.

```
import json
import pandas as pd
from datetime import datetime

# Lista de archivos JSON que ya existen
pathParameters = [
    169484,
    169523,
    169524,
    169529,
    178223
]

# Ruta base para los archivos JSON y CSV
base_path = ""

# Diccionario para almacenar los DataFrames
dfs = {}
```

Los guarda en la ruta que se prefiera. Se van almacenando los dataframes. Tomando cada .json y se va iterando según los sensores para obtener la data.

```
# Procesar cada archivo JSON
for pathParam in pathParameters:
    json_file_path = f"{base_path}{pathParam}.json"

    # Cargar el archivo JSON
    with open(json_file_path, 'r') as file:
        data = json.load(file)

    # Inicializar una lista para almacenar los datos
    records = []

    # Iterar a través de cada sensor y sus datos
    for sensor in data['sensors']:
        sensor_id = sensor['lsid']
        sensor_type = sensor['sensor_type']
        data_structure_type = sensor['data_structure_type']

        for entry in sensor['data']:
            entry_record = {
                'station_id': data['station_id'],
                'sensor_id': sensor_id,
                'sensor_type': sensor_type,
                'data_structure_type': data_structure_type,
                'generated_at': data['generated_at']
            }
            entry_record.update(entry)
            records.append(entry_record)

    # Convertir la lista de registros a un DataFrame de Pandas
    df = pd.DataFrame(records)
```

Una vez generados los dataframes con la fecha en un formato legible se crean los archivos CSV, utilizando el separador de punto y coma (;).

```
# Convertir la lista de registros a un DataFrame de Pandas
df = pd.DataFrame(records)

# Convertir las columnas de marca de tiempo a un formato legible
df['ts'] = pd.to_datetime(df['ts'], unit='s')
df['generated_at'] = pd.to_datetime(df['generated_at'], unit='s')

# Guardar el DataFrame en un archivo CSV
csv_file_path = f"{base_path}{pathParam}.csv"
df.to_csv(csv_file_path, index=False, sep=';')

# Asignar el DataFrame al diccionario usando el código como clave
dfs[f"df_{pathParam}"] = df

# Print a success message with the current time
current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
print(f"CSV file created for station {pathParam} at {current_time}")
```

Mensaje de Éxito:

- Se imprime un mensaje de éxito en la consola, indicando que el script se ejecutó correctamente para la estación actual, junto con la fecha y hora actuales.

```
CSV file created for station 169484 at 2024-06-06 10:57:17
CSV file created for station 169523 at 2024-06-06 10:57:17
CSV file created for station 169524 at 2024-06-06 10:57:17
CSV file created for station 169529 at 2024-06-06 10:57:17
CSV file created for station 178223 at 2024-06-06 10:57:17
```

Figura : Ejemplo de mensaje de éxito para la generación del documento CSV.

Los mensajes de éxito permiten verificar que el proceso se está llevando a cabo y llevar un control del mismo.

—----------

SE DEBE AGREGAR LA PARTE DE LA BASE DE DATOS

—----------

2. **ETL**
   2.1. Se crea un dataframe por cada ID de las estaciones.

—----------

esta creado así porque se esta considerando leer desde la base de datos

—------

```
#Creación de nuevos dataframes a partir de los existentes por estación
df_169484 = dfs['df_169484']
df_169523 = dfs['df_169523']
df_169524 = dfs['df_169524']
df_169529 = dfs['df_169529']
df_178223 = dfs['df_178223']
```

2.2 Se crea la lista de las columnas que se emplearán, dado que no todas las columnas reportan datos o son de interés. La estación de Granados reporta una cantidad menor de variables, por esta razón se tiene una lista aparte para esta estación.

```
# Lista de columnas a seleccionar
columns_to_select = [
    'station_id','ts', 'sensor_type', 'wind_speed_avg', 'uv_dose', 'wind_speed_hi', 'wind_dir_of_hi',
    'wind_chill', 'solar_rad_hi', 'deg_days_heat', 'thw_index', 'bar', 'hum_out',
    'tz_offset', 'uv_index_hi', 'temp_out', 'temp_out_lo', 'wet_bulb', 'temp_out_hi',
    'solar_rad_avg', 'bar_alt', 'arch_int', 'wind_run', 'solar_energy', 'dew_point_out',
    'rain_rate_hi_clicks', 'wind_dir_of_prevail', 'et', 'air_density', 'rainfall_in',
    'heat_index_out', 'thsw_index', 'rainfall_mm', 'night_cloud_cover', 'deg_days_cool',
    'rain_rate_hi_in', 'uv_index_avg', 'wind_num_samples', 'emc', 'rain_rate_hi_mm',
    'rev_type', 'rainfall_clicks', 'abs_press','moist_soil_last','bar_trend_3_hr','pressure_last'
]

columns_to_select_GRANADOS = [
    'station_id','ts', 'sensor_type','wind_speed_avg',
    'uv_dose','wind_speed_hi','wind_dir_of_hi','wind_chill',
    'solar_rad_hi','deg_days_heat','thw_index','bar',
    'hum_out','uv_index_hi','temp_out','temp_out_lo',
    'wet_bulb','temp_out_hi','solar_rad_avg','bar_alt',
    'arch_int','wind_run','solar_energy','dew_point_out',
    'rain_rate_hi_clicks','wind_dir_of_prevail','et',
    'air_density','rainfall_in','heat_index_out',
    'thsw_index','rainfall_mm','night_cloud_cover',
    'deg_days_cool','rain_rate_hi_in','uv_index_avg',
    'wind_num_samples','emc','rain_rate_hi_mm',
    'rev_type','rainfall_clicks','abs_press', 'bar_trend_3_hr','pressure_last'
]
```

2.3 Se genera dataframes solo con las columnas de las listas.

```
# Crear el nuevo dataframe con solo las columnas seleccionadas
df_JERONIMO = df_169484[columns_to_select]
df_SALAMA = df_169523[columns_to_select]
df_CUNBAV = df_169524[columns_to_select]
df_CUBULCO = df_169529[columns_to_select]
df_GRANADOS = df_178223[columns_to_select_GRANADOS]
```

```
df_JERONIMO.head()
```

	station_id	ts	sensor_type	wind_speed_avg	uv_dose	wind_speed_hi	wind_dir_of_hi	wind_chill	solar_rad_hi	deg_days_heat	...
0	169484	2024-06-06 16:00:00	30	2.0	NaN	7.0	10.0	82.9	594.0	0.0	...
1	169484	2024-06-06 16:00:00	501	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
2	169484	2024-06-06 16:15:00	501	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
3	169484	2024-06-06 16:30:00	501	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
4	169484	2024-06-06 16:45:00	501	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

5 rows × 46 columns

Figura : Ejemplo de dataframe de la estación Jeronimo filtrado según variables de interés.

2.4 Para cada estación tiene sensores definidos que son los que reportan los datos. Por tal razón, se filtra los datos según los sensores activos.

Además, la fecha requiere una modificación de retroceder 6 horas para todas las estaciones, se crea una nueva columna con el valor.

```
# Filtrar el dataframe para dejar solo las filas donde sensor_type es 30 o 108
df_JERONIMO_filtrado = df_JERONIMO[(df_JERONIMO['sensor_type'] == 30) | (df_JERONIMO['sensor_type'] == 108)]
# Restar 6 horas a la columna ts
df_JERONIMO_filtrado['ts'] = pd.to_datetime(df_JERONIMO_filtrado['ts']) - pd.Timedelta(hours=6)
```

```
# Filtrar el dataframe para dejar solo las filas donde sensor_type es 30 o 108
df_CUNBAV_filtrado = df_CUNBAV[(df_CUNBAV['sensor_type'] == 30) | (df_CUNBAV['sensor_type'] == 108)]
# Restar 6 horas a la columna ts
df_CUNBAV_filtrado['ts'] = pd.to_datetime(df_CUNBAV_filtrado['ts']) - pd.Timedelta(hours=6)
```

```
# Filtrar el dataframe para dejar solo las filas donde sensor_type es 30 o 108
df_CUBULCO_filtrado = df_CUBULCO[(df_CUBULCO['sensor_type'] == 30) | (df_CUBULCO['sensor_type'] == 108)]
# Restar 6 horas a la columna ts
df_CUBULCO_filtrado['ts'] = pd.to_datetime(df_CUBULCO_filtrado['ts']) - pd.Timedelta(hours=6)
```

```
# Filtrar el dataframe para dejar solo las filas donde sensor_type es 30 o 108
df_GRANADOS_filtrado = df_GRANADOS[(df_GRANADOS['sensor_type'] == 30) | (df_GRANADOS['sensor_type'] == 3)]
# Restar 6 horas a la columna ts
df_GRANADOS_filtrado['ts'] = pd.to_datetime(df_GRANADOS_filtrado['ts']) - pd.Timedelta(hours=6)
```

2.5 Ahora no es necesario la columna de sensor_type y la de tz_offset, por tal razón, se eliminan.

```
# Eliminar la columna 'sensor_type'
df_JERONIMO_filtrado.drop(columns=['sensor_type','tz_offset'], inplace=True)
df_SALAMA_filtrado.drop(columns=['sensor_type','tz_offset'], inplace=True)
df_CUNBAV_filtrado.drop(columns=['sensor_type','tz_offset'], inplace=True)
df_CUBULCO_filtrado.drop(columns=['sensor_type','tz_offset'], inplace=True)
df_GRANADOS_filtrado.drop(columns=['sensor_type'], inplace=True)
```

2.6 Para un manejo eficiente se despivotea en los dataframes de cada estación.

```
# Despivoteando las columnas seleccionadas, manteniendo 'station_id' y 'ts' sin despivotear
df_JERONIMO_unpivot = pd.melt(df_JERONIMO_filtrado, id_vars=['station_id', 'ts'], var_name='variable', value_name='value')
df_SALAMA_unpivot = pd.melt(df_SALAMA_filtrado, id_vars=['station_id', 'ts'], var_name='variable', value_name='value')
df_CUNBAV_unpivot = pd.melt(df_CUNBAV_filtrado, id_vars=['station_id', 'ts'], var_name='variable', value_name='value')
df_CUBULCO_unpivot = pd.melt(df_CUBULCO_filtrado, id_vars=['station_id', 'ts'], var_name='variable', value_name='value')
df_GRANADOS_unpivot = pd.melt(df_GRANADOS_filtrado, id_vars=['station_id', 'ts'], var_name='variable', value_name='value')
```

2.7 Se realiza la unión de los dataframe para tener uno con todos los valores.

```
# Lista de dataframes
dfs_unpivot = [df_JERONIMO_unpivot, df_SALAMA_unpivot, df_CUNBAV_unpivot, df_CUBULCO_unpivot, df_GRANADOS_unpivot]

# Unir todos los dataframes
df_todas_estaciones= pd.concat(dfs_unpivot, ignore_index=True)
```

2.8 Si la estación no registra valor entonces, tendrá nulo (NaN). Este dato se elimina.

```
# Eliminar los NaN de la columna 'value'
df_todas_estaciones_sin_nan = df_todas_estaciones.dropna(subset=['value'])
```

```
df_todas_estaciones_sin_nan.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 189 entries, 0 to 584
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   station_id  189 non-null    int64
 1   ts          189 non-null    datetime64[ns]
 2   variable    189 non-null    object
 3   value       189 non-null    float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(1)
memory usage: 7.4+ KB
```

	station_id	ts	variable	value
0	169484	2024-06-06 10:00:00	wind_speed_avg	2.0
6	169484	2024-06-06 10:00:00	wind_speed_hi	7.0
9	169484	2024-06-06 10:00:00	wind_dir_of_hi	10.0
12	169484	2024-06-06 10:00:00	wind_chill	82.9
15	169484	2024-06-06 10:00:00	solar_rad_hi	594.0

Figura : Ejemplo de dataframe con todas las estaciones sin nulos.

2.9 Debido a que las variables al ser descargadas con la API vienen con dimensionales diferentes a las que se emplean normalmente se hace la transformación de variables a través de la definición transformar_valor de la siguiente forma:

- inHg a mbar
- Fahrenheit a Celsius
- millas/h a Km/h
- in/día a mm/día

```python
# Definir función para transformar los valores
def transformar_valor(row):
    if row['variable'] == 'bar':
        return row['value'] * 33.8639  # convertir inHg a mbar
    elif row['variable'] == 'temp_out':
        return (row['value'] - 32) * 5.0/9.0  #convertir de Fahrenheit a Celsius
    elif row['variable'] == 'temp_out_hi':
        return (row['value'] - 32) * 5.0/9.0  #convertir de Fahrenheit a Celsius
    elif row['variable'] == 'temp_out_lo':
        return (row['value'] - 32) * 5.0/9.0  #convertir de Fahrenheit a Celsius
    elif row['variable'] == 'dew_point_out':
        return (row['value'] - 32) * 5.0/9.0  #convertir de Fahrenheit a Celsius
    elif row['variable'] == 'wet_bulb':
        return (row['value'] - 32) * 5.0/9.0  #convertir de Fahrenheit a Celsius
    elif row['variable'] == 'heat_index_out':
        return (row['value'] - 32) * 5.0/9.0  #convertir de Fahrenheit a Celsius
    elif row['variable'] == 'thw_index':
        return (row['value'] - 32) * 5.0/9.0  #convertir de Fahrenheit a Celsius
    elif row['variable'] == 'thsw_index':
        return (row['value'] - 32) * 5.0/9.0  #convertir de Fahrenheit a Celsius
    elif row['variable'] == 'wind_chill':
        return (row['value'] - 32) * 5.0/9.0  #convertir de Fahrenheit a Celsius
    elif row['variable'] == 'wind_speed_avg':
        return row['value'] * 1.60934  # convertir millas/h a Km/h
    elif row['variable'] == 'wind_speed_hi':
        return row['value'] * 1.60934  # convertir millas/h a Km/h
    elif row['variable'] == 'wind_run':
        return row['value'] * 1.60934  # convertir millas/h a Km/h
    elif row['variable'] == 'et':
        return row['value'] * 25.4  # convertir in/día a mm/día
    else:
        return row['value']

# Aplicar la función al DataFrame y crear la nueva columna 'valor'
df_todas_estaciones_sin_nan['valor'] = df_todas_estaciones_sin_nan.apply(transformar_valor, axis=1)
```

2.10 Se crea un archivo CSV con separador de punto y coma (;).

```
df_todas_estaciones_sin_nan.to_csv('', index=False, sep=';')
df_todas_estaciones_sin_nan.info()
df_todas_estaciones_sin_nan.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 189 entries, 0 to 584
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   station_id  189 non-null    int64
 1   ts          189 non-null    datetime64[ns]
 2   variable    189 non-null    object
 3   value       189 non-null    float64
 4   valor       189 non-null    float64
dtypes: datetime64[ns](1), float64(2), int64(1), object(1)
memory usage: 8.9+ KB
```

	station_id	ts	variable	value	valor
0	169484	2024-06-06 10:00:00	wind_speed_avg	2.0	3.218680
6	169484	2024-06-06 10:00:00	wind_speed_hi	7.0	11.265380
9	169484	2024-06-06 10:00:00	wind_dir_of_hi	10.0	10.000000
12	169484	2024-06-06 10:00:00	wind_chill	82.9	28.277778
15	169484	2024-06-06 10:00:00	solar_rad_hi	594.0	594.000000

Figura : Ejemplo de dataframe con todas las estaciones con la columna valor con los datos con las nuevas dimensionales.

2.11 Se lee el documento Excel llamado LIMITES_GENERALES, donde como su nombre lo indica tiene los límites máximo y mínimo que permitirán posteriormente comparar los valores entrantes.

```
df_limites = pd.read_excel('.xlsx')
```

```
df_limites.info()
df_limites.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24 entries, 0 to 23
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   VARIABLE  24 non-null     object
 1   MAX       24 non-null     float64
 2   MIN       24 non-null     float64
dtypes: float64(2), object(1)
memory usage: 708.0+ bytes
```

	VARIABLE	MAX	MIN
0	bar	1083.8	910.0
1	temp_out	37.9	0.0
2	temp_out_hi	40.0	5.0
3	temp_out_lo	37.6	0.0
4	hum_out	100.0	7.0

Figura : Ejemplo de dataframe de Límites Generales.

2.12 Se elimina la columna value, ya que tiene los datos en las dimensionales que no se requieren. Se le da un nombre en español que sea más explicativo.

```
# Eliminar la columna 'value'
df_todas_estaciones_sin_nan.drop(columns=['value'], inplace=True)

# Renombrar las columnas
df_todas_estaciones_sin_nan.rename(columns={
    'station_id': 'ID',
    'ts': 'FECHA',
    'variable': 'VARIABLE',
    'valor': 'VALOR'
}, inplace=True)
```

```
df_todas_estaciones_sin_nan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 189 entries, 0 to 584
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ID        189 non-null    int64
 1   FECHA     189 non-null    datetime64[ns]
 2   VARIABLE  189 non-null    object
 3   VALOR     189 non-null    float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(1)
memory usage: 7.4+ KB
```

Figura : Ejemplo del dataframe con los nuevos encabezados.

2.13 Se crea una copia del dataframe anterior(contiene los datos de todas las estaciones). Además, se hace la unión entre la tabla con todos los datos y los límites generales.

```
df_valores_para_comparar = df_todas_estaciones_sin_nan
```

```
# Realizar el merge
df_merged = pd.merge(df_valores_para_comparar, df_limites, on='VARIABLE', how='left')
```

2.14 Se realiza la comparación entre los limite

```
df_merged2 = df_merged
# Realiza la validación de rango
df_merged2['VALOR_final'] = np.where(
    (df_merged2['VALOR'] <= df_merged2['MAX']) &
    (df_merged2['VALOR'] >= df_merged2['MIN']),
    df_merged2['VALOR'],
    np.nan
)
```

```
df_merged2.info()
df_merged2.head()
```

```
df_merged2.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 189 entries, 0 to 188
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   ID            189 non-null    int64
 1   FECHA         189 non-null    datetime64[ns]
 2   VARIABLE      189 non-null    object
 3   VALOR         189 non-null    float64
 4   MAX           122 non-null    float64
```

```
df_merged2.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 189 entries, 0 to 188
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   ID            189 non-null    int64
 1   FECHA         189 non-null    datetime64[ns]
 2   VARIABLE      189 non-null    object
 3   VALOR         189 non-null    float64
 4   MAX           122 non-null    float64
 5   MIN           122 non-null    float64
 6   VALOR_final   122 non-null    float64
dtypes: datetime64[ns](1), float64(4), int64(1), object(1)
memory usage: 10.5+ KB
```

	ID	FECHA	VARIABLE	VALOR	MAX	MIN	VALOR_final
0	169484	2024-06-06 10:00:00	wind_speed_avg	3.218680	35.4	0.0	3.218680
1	169484	2024-06-06 10:00:00	wind_speed_hi	11.265380	120.0	0.0	11.265380
2	169484	2024-06-06 10:00:00	wind_dir_of_hi	10.000000	360.0	0.0	10.000000
3	169484	2024-06-06 10:00:00	wind_chill	28.277778	39.7	5.6	28.277778
4	169484	2024-06-06 10:00:00	solar_rad_hi	594.000000	1171.0	0.0	594.000000

```
df_datos_validados_rangos = df_merged2
# Redondear la columna "VALOR" a 2 decimales
df_datos_validados_rangos['VALOR_final'] = df_datos_validados_rangos['VALOR_final'].round(2)
df_datos_validados_rangos.to_csv('.csv', index=False, sep=';')
```

```
df_datos_validados_rangos_v2 =df_datos_validados_rangos
```

```
print(df_datos_validados_rangos_v2.columns)
```

```
Index(['ID', 'FECHA', 'VARIABLE', 'VALOR', 'MAX', 'MIN', 'VALOR_final'], dtype='object')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 189 entries, 0 to 188
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ID           189 non-null    int64
 1   FECHA        189 non-null    datetime64[ns]
 2   VARIABLE     189 non-null    object
 3   VALOR        189 non-null    float64
 4   MAX          122 non-null    float64
 5   MIN          122 non-null    float64
 6   VALOR_final  122 non-null    float64
dtypes: datetime64[ns](1), float64(4), int64(1), object(1)
memory usage: 10.5+ KB
```

	ID	FECHA	VARIABLE	VALOR	MAX	MIN	VALOR_final
0	169484	2024-06-06 10:00:00	wind_speed_avg	3.218680	35.4	0.0	3.218680
1	169484	2024-06-06 10:00:00	wind_speed_hi	11.265380	120.0	0.0	11.265380
2	169484	2024-06-06 10:00:00	wind_dir_of_hi	10.000000	360.0	0.0	10.000000
3	169484	2024-06-06 10:00:00	wind_chill	28.277778	39.7	5.6	28.277778
4	169484	2024-06-06 10:00:00	solar_rad_hi	594.000000	1171.0	0.0	594.000000

```
[ ] df_datos_validados_rangos = df_merged2
    # Redondear la columna "VALOR" a 2 decimales
    df_datos_validados_rangos['VALOR_final'] = df_datos_validados_rangos['VALOR_final'].round(2)
    df_datos_validados_rangos.to_csv('.csv', index=False, sep=';')
```

```
[ ] df_datos_validados_rangos_v2 =df_datos_validados_rangos
```

```
[ ] print(df_datos_validados_rangos_v2.columns)
```

```
Index(['ID', 'FECHA', 'VARIABLE', 'VALOR', 'MAX', 'MIN', 'VALOR_final'], dtype='object')
```

```
# Eliminar las columnas "VALOR", "MAX" y "MIN"
df_datos_validados_rangos_v2.drop(columns=['VALOR', 'MAX', 'MIN'], inplace=True)
```

```
# Pivoteo del DataFrame
df_pivot = df_datos_validados_rangos_v2.pivot_table(index=['ID', 'FECHA'], columns='VARIABLE', values='VALOR_final').reset_index()
```

```
df_pivot.info()
df_pivot.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 26 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               5 non-null      int64
 1   FECHA            5 non-null      datetime64[ns]
 2   bar              5 non-null      float64
 3   deg_days_cool    5 non-null      float64
 4   deg_days_heat    5 non-null      float64
 5   dew_point_out    5 non-null      float64
 6   et               5 non-null      float64
 7   heat_index_out   5 non-null      float64
 8   hum_out          5 non-null      float64
 9   moist_soil_last  4 non-null      float64
 10  rain_rate_hi_mm  5 non-null      float64
 11  rainfall_mm      5 non-null      float64
 12  solar_energy     5 non-null      float64
 13  solar_rad_avg    5 non-null      float64
 14  solar_rad_hi     5 non-null      float64
```

```
df_pivot.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 26 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               5 non-null      int64
 1   FECHA            5 non-null      datetime64[ns]
 2   bar              5 non-null      float64
 3   deg_days_cool    5 non-null      float64
 4   deg_days_heat    5 non-null      float64
 5   dew_point_out    5 non-null      float64
 6   et               5 non-null      float64
 7   heat_index_out   5 non-null      float64
 8   hum_out          5 non-null      float64
 9   moist_soil_last  4 non-null      float64
 10  rain_rate_hi_mm  5 non-null      float64
 11  rainfall_mm      5 non-null      float64
 12  solar_energy     5 non-null      float64
 13  solar_rad_avg    5 non-null      float64
 14  solar_rad_hi     5 non-null      float64
 15  temp_out         5 non-null      float64
 16  temp_out_hi      5 non-null      float64
 17  temp_out_lo      5 non-null      float64
 18  thsw_index       4 non-null      float64
 19  thw_index        5 non-null      float64
 20  wet_bulb         5 non-null      float64
 21  wind_chill       5 non-null      float64
 22  wind_dir_of_hi   5 non-null      float64
 23  wind_run         5 non-null      float64
 24  wind_speed_avg   5 non-null      float64
 25  wind_speed_hi    5 non-null      float64
dtypes: datetime64[ns](1), float64(24), int64(1)
memory usage: 1.1 KB
```

VARIABLE	ID	FECHA	bar	deg_days_cool	deg_days_heat	dew_point_out	et	heat_index_out	hum_out	moist_soil_last	...
0	169484	2024-06-06 10:00:00	1017.00	0.75	0.0	18.95	0.38	29.47	57.0	32.35	...
1	169523	2024-06-06 10:00:00	1015.92	0.67	0.0	20.61	0.36	29.09	67.0	27.95	...

```
df_pivot.to_csv('C:/Users/stzor/Desktop/Junio/ICDF/df_pivot_11jun2024.csv', index=False, sep=';')
```

```
df_revision_temps = df_pivot
```

```
# Comprobación de temp_out_lo no mayor que temp_out_hi
comprobacion = df_revision_temps['temp_out_lo'] > df_revision_temps['temp_out_hi']
df_revision_temps.loc[comprobacion, 'temp_out_lo'] = None
```

```
df_revision_temps.to_csv('C:/Users/stzor/Desktop/Junio/ICDF/df_revision_temps11jun.csv', index=False, sep=';')
```

```
comprobacion.info()
comprobacion.head()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 5 entries, 0 to 4
Series name: None
Non-Null Count  Dtype
--------------  -----
5 non-null      bool
```

```
df_correccion_radiacion = df_revision_temps
```

```
# Función para verificar el rango de horas
def es_hora_noche(hora):
    return (hora >= 19) or (hora < 4)

# Aplicar la función y obtener una máscara
correcion = df_correccion_radiacion['FECHA'].dt.hour.apply(es_hora_noche)

# Volver nulos los valores en las columnas específicas donde la máscara es verdadera
df_correccion_radiacion.loc[correcion, ['solar_energy', 'solar_rad_avg', 'solar_rad_hi']] = 0
```

```
df_correccion_radiacion.info()
df_correccion_radiacion.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 26 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               5 non-null      int64
 1   FECHA            5 non-null      datetime64[ns]
 2   bar              5 non-null      float64
 3   deg_days_cool    5 non-null      float64
 4   deg_days_heat    5 non-null      float64
 5   dew_point_out    5 non-null      float64
 6   et               5 non-null      float64
```

```
df_correccion_radiacion.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 26 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ID               5 non-null      int64
 1   FECHA            5 non-null      datetime64[ns]
 2   bar              5 non-null      float64
 3   deg_days_cool    5 non-null      float64
 4   deg_days_heat    5 non-null      float64
 5   dew_point_out    5 non-null      float64
 6   et               5 non-null      float64
 7   heat_index_out   5 non-null      float64
 8   hum_out          5 non-null      float64
 9   moist_soil_last  4 non-null      float64
 10  rain_rate_hi_mm  5 non-null      float64
 11  rainfall_mm      5 non-null      float64
 12  solar_energy     5 non-null      float64
 13  solar_rad_avg    5 non-null      float64
 14  solar_rad_hi     5 non-null      float64
 15  temp_out         5 non-null      float64
 16  temp_out_hi      5 non-null      float64
 17  temp_out_lo      5 non-null      float64
 18  thsw_index       4 non-null      float64
 19  thw_index        5 non-null      float64
 20  wet_bulb         5 non-null      float64
 21  wind_chill       5 non-null      float64
 22  wind_dir_of_hi   5 non-null      float64
 23  wind_run         5 non-null      float64
 24  wind_speed_avg   5 non-null      float64
 25  wind_speed_hi    5 non-null      float64
dtypes: datetime64[ns](1), float64(24), int64(1)
memory usage: 1.1 KB
```

VARIABLE	ID	FECHA	bar	deg_days_cool	deg_days_heat	dew_point_out	et	heat_index_out	hum_out	moist_soil_last	...
0	169484	2024-06-06 10:00:00	1017.00	0.75	0.0	18.95	0.38	29.47	57.0	32.35	...
1	169523	2024-06-06 10:00:00	1015.92	0.67	0.0	20.61	0.36	29.09	67.0	27.95	...