# README

Fabricio Manuel Pérez Toledo

January 2019

# Contents

# 1 Introducción

La tarea encomendada en el departamento de ingeniería ha sido la creación de un script en python capaz de generar un informe a partir de las medidas obtenidas por el D1 y D2. Dicho informe debe de cumplir una serie de requisitos para alcanzar su objetivo, qué es la comparación desde una perspectiva estadística de las diferencias entre ambos. Como bien se ha visto en la anterior apartado, ambos sistemas debería de obtener medidas muy similares siendo despreciables las diferencias entre los dos sistemas. A día de hoy, se ha comprobado la similitud de ambos sistemas para periodos cortos de tiempo y es necesario hacerlo para periodos mayores. Así como otros objetivos a largo plazo que ahora mismo no son relevantes y no se han tenido en cuenta a la hora de desarrollar el script.

Es notorio que mi labor ha sido de programar durante todo el periodo de prácticas. Por esta razón, mis actividades se han reducido a crear desde cero el script que se usará para la obtención del informe. Descartando otras actividades complementarias sobre el DIMMA a excepción de un software creado con LabView con el fin de obtener las medidas obtenidas por D1 ya que este no genera de forma automática un archivo que sea guardado en un directorio.

# 2 Definiciones

Con el objetivo de facilitar el proceso de comprensión de texto, es necesario realizar unas definiciones aclaratorias sobre ciertas abreviaturas empleadas:

- D1 : Comprende al sistema DIMM, una versión antigua del actual en pruebas. Que dispone de otros softwares, sistemas de automatización parcial entre otras diferencias.
- D2 : Correspondel con el sistema DIMM actual. Que es completamente automático y por ello se añade una A al final. Resultando el DIMMA. Además, tiene otros software y otras mejoras frente a D1. Recalcar que es nuestro sujeto de estudio y el futuro sucesor de D1.
- DataFrame : Estructura de datos tabular de dos dimensiones, mutable, potencialmente heterogénea con ejes etiquetados (filas y columnas). Las operaciones aritméticas se alinean en ambas etiquetas de fila y columna. Puede considerarse como un contenedor tipo dict para objetos de la serie. La principal estructura de datos de Pandas.

A continuación indico las categorías de clasificación de las medidas:

- Válidas: Son todas aquellas medidas que tienen un 'Seeing' inferior a 7, el número de 'Muestras' es superior a 190 y 'Return' es igual a 0.
- No válidas: No cumplen las condiciones mencionadas.
- guales: Consiste en comprobar que la información contenida fila a fila para una misma variable en ambos sistemas es igual. Por ejemplo, al comparar las columnas 'Objeto_D1' y 'Objeto_D2' debe de coincidir el nombre de los objetos fila a fila.
- No iguales : No cumplen la condición anterior.

Definimos el nombre de las sheets del documentos en XLSX y de algunas tablas del documentos PDF.

- DS1 : Corresponde a la colección de medidas que cumplen el criterio de válidas y no iguales.
- DS2 : Corresponde a la colección de medidas que cumplen el crterio de válidas e iguales.
- DS3 : Corresponde a la colección de medidas que cumplen el criterio de no iguales.
- DS4 : Corresponde a la colección de medidas que cumplen el cirterio de no válidas.
- COMP-DS1 : Corresponde a la colección de diferencias de medidas que cumplen el criterio de válidas y no iguales.

- COMP-DS2 : Corresponde a la colección de diferencias de medidas que cumplen el crterio de válidas e iguales.
- COMP-DS3 : Corresponde a la colección de diferencias de medidas que cumplen el criterio de no iguales.
- COMP-DS4 : Corresponde a la colección de diferencias de medidas que cumplen el cirterio de no válidas.

# 3 Objetivos del script

Los objetivos planteado para el script, y por ende para el informe, son los siguientes:

- Debe ser capaz de leer simultáneamente dos archivo con gran cantidad de información. De ellos, deberá en primer lugar condicionarlos para su análisis y comparación de forma clara.
- Debe contener unos filtros que permitan clasificar las medidas según sea su calidad.
- Debe poder generar archivos que guarden los resultados obtenidos. Un archivo XLSX y otro PDF.
- Debe realizar análisis estadístico.
- Debe generar gráficos y guardarlos.
- Debe añadir el lugar de obtención de medidas, la fecha de inicio, fecha de finalización, el software empleado para D1 y para D2.
- Debe crear una tabla con el número de imágenes y porcentajes de cada categoría.

# 4 Estructura del script

La creación del script no ha sido labor sencilla debido a la inexperiencia en el campo de la programación. Han habido diversos diseños estructurales que han dado errores complejos, que han llevado horas de investigación sobre sus causas y posibles soluciones. Por esta razón, el script tiene cierta complejidad dado que se compone de tres archivos : REPORT_DIMMA_init.py, REPORT_DIMMA_func.py y report_templates.htlm. La combinación funcional de los mismos podría estar bien representada por el organigrama (fig).

Como bien se puede observar, está constituido por dos archivo PYTHON y uno HTML. El archivo más importante es REPORT_DIMMA_func.py al que me referiré mediante RDFPY. Este script es una macro función que a su vez contiene otras funciones en su interior y las instrucciones para generar los archivos finales. Es el corazón del script que contiene desde las funciones para filtrar las medidas, crear las diferentes categorías, crear DataFrame, los archivos finales, etc.

El siguiente archivo PYTHON es el REPORT_DIMMA_init.py (RDIPY) cuya función en este script es cargar y crear un generador de archivos ordenados por fecha, así como llamar al script RDFPY para la obtención de los archivos finales y su posterior modificación de nombre acorde a los archivos de entrada.

Luego el archivo HTML de nombre report_templates.html (RT) tiene como función ser la estructura que contendrá la información del análisis estadístico en formato de tablas y que será convertido en un archivo PDF al final del proceso. Además, contiene información sobre definiciones de las distintas categorías y contenido de las tablas.

En la Fig. 5, se observa como el script comienza con RDIPY que una vez realizado unas operaciones previos, crea un generador que devuelve dos archivos de entrada que corresponden a D1 y D2 para una misma fecha. Acto seguido, llama a la funcion Report() contenida en RDFPY, que a su vez llamará al archivo HTML donde almacenará la información estadística en formato de tablas y unas determinadas figuras sobre la evolución temporal del seeing. Tras este proceso, RDFPY convierte al archivo HTML en

un archivo PDF y junto a un archivo XLSX con varios hojas son devueltos a RFIPY. Finalmente, modificará los nombres y almacenará los resultados en un nuevo directorio. A partir de este punto, se imprime en pantalla el archivo PDF obtenido y se generan dos nuevos archivos de entrada. El ciclo se repite hasta procesar todos los archivos disponibles en el directorio de entrada.

La razón última por la que ha sido necesario adquirir esta estructura se debe a la necesidad de que el script sea capaz de leer todo el contenido de un directorio sin tener que intervenir continuamente en los archivos de entrada. No obstante, no es la única opción viable, ya que también habría sido posible reemplazar en la línea 106 (función Report()) en RDIPY, por todo el contenido de RDFPY. Simplemente habría sido necesario cuidar la correcta identaxión.

Otras posibles estructuras podrían haber sido posibles, mediante la creación de clases entre otras opciones. Sin embargo, el conocimiento disponible actualmente, el tiempo para la realización de las pŕacticas, así como resultar ser la solución más sencilla.

# 5 Requerimiento técnicos

Para el correcto funcionamiento del script es necesario que se hayan cumplido los siguiente requerimientos:

- python = 3.6
- numpy = 1.15.0
- pandas = 0.23.4
- jinja2 = 2.10
- WeasyPrint = 43
- matplotlib = 2.2.2
- pypdf2 = 1.26.0
- re = 2.2.1
- os = no tiene versión.
- glob = no tiene versión.

Cabe citar por experiencia propia que el posible encontrar algunos problemas según sea la versión del módulo. Modificaciones en el código han hecho posible omitir este inconveniente.

Además, es conveniente advertir que la instalación de WeasyPrint puede generar problemas especialmente en Windows. Para la correcta solución es conveniente consulta la bibliografía presente en la página web del módulo [1].

# 6 RDIPY antes de RDFPY

En esta sección explicaré detalladamente los pasos seguidos para la creación del script generador de reports. Para ello, seguiré la ruta de ejecución del script para un ciclo completo. No obstante, no tendré en cuenta el archivo HTML ya que no se realiza ninguna operación dentro de este. Simplemente tiene función estructural, indicando en que orden deben aparecer la información creada en RDFPY.

Para comenzar, partimos de la estructura de directorios adecuada para el correcto funcionamiento del script. Lo ideal es un directorio creado en ./home/username que por defecto se llamará /DIMMA-SCRIPT. Dentro de este directorio, habrá otros dos directorios /MEDIDAS y /REPORTS . En el directorio /DIMMA-SCRIPT estará los archivos RDIPY, RDFPY y RT. Esto es importante tenerlo en cuenta para la correcta ejecución del script.

[1] https://weasyprint.readthedocs.io/en/stable/install.html

A partir de este punto comienzo a explicar paso a paso la ejecución del script RDIPY, su salto intermedio a RDFPY y posterior retorno a RDIPY.

```
#librear as usadas y otros scripts empleados.
import pandas as pd #Manejo de gran cantidad de datos.
import glob # Para poder usar todos los archivos de un directorio.
import re # Para cambiar nombres, directorios, etc.
from REPORT_DIMMA_func import * # Script encargado de generar cada report
#de forma individual.
import os #Para cambiar nombre de archivos.
```

Listing 1: Librerías usadas en RDIPY.

Comenzando por definir que librerías de python son necesarias para la ejecución del script. Sin embargo, los comentarios añadidos al código ya aportan la información necesaria sobre los motivos de su aplicación. Recalcar que es neceario tratar a RDFPY como una librería externa que debe ser cargada de la misma forma que otras librerías.

```
#Constantes
#Lista de nombres de columnas que sustituir n las generadas por #defecto.
stand = ['Fecha','Ubicacion','Objeto','Seeing','Maire','Flujo1','Flujo2','Var_l','Var_t','
    Centelleo1','Centelleo2','r0l','r0t','fwhml'
          ,'fwmlt','CCD','Software','Ventana','EscalaPlaca','Distancia'
          ,'Diametro','WinSpot','Gain','Texp','Sampling','N','Muestras'
          , 'Return']

#Conjunto de columnas que ser  seleccionadas por medio de  ndice .
use_cols = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]
```

Listing 2: Constantes y listas predeterminadas.

La finalidad de Listing 2 es agrupar de forma clara aquellas constantes y listas predeterminadas. Aunque en este caso, debido a necesidad explicadas próximamente, use_cols se añadirá un elemento.

```
# INPUTS
#Comentarios de entrada empleados para informar al usuario.
print("\n"
      "The following script has been desgined to produce reports that \n"
      "compares D1's measurements and D2's measurements. The statistical \n"
      " results provide us information whether both systems are comparables.")

print("\n"
      "The following example shows how you should write the directories \n "
      " and files format. IF YOU USE OTHER FORMAT, YOU WILL FIND PROBLEMS. \n"
      "More information in README document.")

print("\n"
      "The directory for input files :./MEDIDAS/"
      "\n"
      "The directory for output files:./REPORTS/"
      "\n"
      "Generic name for input files of D1:IAC_TEST_*.txt"
      "\n"
      "Generic name for input files of D2:RMMSX_*.txt")
```

Listing 3: Prints

El contenido de Listing 3 es información de interés para el usuario. Compuesto por un breve resumen de la finalidad del script, una breve advertencia si no se siguen las indicaciones y un ejemplo de como debe ser escrito los datos que se pedirán al ejecutar el script.

```
dir_in = input('The directory for input files :') #Directorio de entrada
#definido por el usuario.
dir_out = input('The directory for output files:')# Directorio de salida
#definido por el usuario.
file_D1 = input('Generic name for input files of D1:') #Nombre gen rico de
#los archivos de entrada para D1.
file_D2 = input('Generic name for input files of D2:')# Nombre gen rico de los
# archivos de entrada para D2.
```

Listing 4: Información necesaria en la entrada del script.

Listing 4 contiene el código que pide en el terminal que introduzcamos información sobre el directorio de los archivos de entrada, directorio de salida y definir los nombres de los archivos de entrada.

```
listD2A = glob.glob(dir_in + file_D2) #Genera un lista con todos los archivos
listD1A = glob.glob(dir_in + file_D1) #que se localizan en el directorio.
```

Listing 5: Lista de archivos de entrada.

Listing 5 genera una lista con los archivos de D1 y D2.

```
for g in listD2A: # Genera un archivo .csv por cada .txt para D2.
    df = pd.read_csv(g ,delimiter='\t', index_col=None) #Lee el archivo.
    returnindex = df.columns.get_loc('Return') #Devuelve el  ndice  de 'Return'.
    use_cols.append(returnindex + 1)# Sumamos 1 al  ndice  de 'Return' y lo insertamos en
    use_cols.
    q = re.sub('\.txt$','.csv',g)#Sustituye la ra z del nombre.
    df.to_csv(q) #Guarda el archivo en formato CSV.
    df = pd.read_csv(q, decimal='.', usecols=use_cols)#Relee el archivo
    #seleccionando un conjunto determinado de columnas.
    colnameD2 = df.columns #Creamos una variable con la lista de columnas
    #por defecto.
    for i in range(len(stand)): #Empleamos un for para cambiar el nombre de
        #las columnas una a una, ya que el otro script necesita unos nombre
        # determinados.
        df = df.rename(columns={colnameD2[i] : stand[i]}) #Cambia los nombres de las
    columnas.
    df.to_csv(q, index=False) #Guardamos el resultado en CSV.
```

Listing 6: Conversor TXT a CSV.

En Listing 6, la finalidad del código es convertir los archivos TXT en CSV porque son más fáciles de manejar con las librerías de pandas. Además, al leer el archivo hay que prestar especial atención al delimiter.

Dado que el siguiente fragmento de código es prácticamente igual que el anterior excepto que es para D1, será omitido. De forma que indicaré los fragmentos más relevantes.

```
for fileD1, fileD2 in zip(sorted(glob.iglob(dir_in + 'IAC_TEST_*.csv')), sorted(glob.iglob(
    dir_in + 'RMMSX_*.csv'))):
    #Su funci n es crear un for que funciona como un generador de archivos.
    #De esta forma, en cada paso, el for devuelve dos nuevos archivos
    #que pueden ser le dos por las siguientes l neas y llegar al final del
    #proceso sin generar errores.
    #El zip se debe a que son archivos grandes, el sorted ya que
    #deben estar ordenados por fecha y iglob crea el generador de archivos.
    D2 = pd.read_csv(fileD2, decimal='.', parse_dates=['Fecha'])
    #Lee el archivo generado para D2.
    D1 = pd.read_csv(fileD1, decimal='.', parse_dates=['Fecha'])
    #Lee el archivo generado para D1.
    Report(D2,D1) # Llama a la funci n contenido en el script REPORT_DIMMA_func.py
```

Listing 7: Generador de archivos y posterior carga.

En Listing 7, contiene uno de los fragmentos más importantes. Esto se debe a que ha sido complejo encontrar los elementos que permitieran cargar los archivos CSV (son pesados) de forma ordenada, y además que cargara un archivo por sistema. Si en vez de emplear iglob hubiera sido glob, habría obtenido una lista con todos los archivos previamente cargados. Eso supondría sobrecargar la capacidad del ordenador.

Luego, tras leer los archivos CSV de D1 y D2, se llama al función Report(). Es por ello que a continuación pasamos al script RDFPY

# 7 RDFPY

Ahora continuamos con la ejecución de RDFPY.

```
#Librer as y su descendencia.
import numpy as np # Multitud de funciones matem ticas.
import pandas as pd #Ya citado.
from jinja2 import Environment, FileSystemLoader # Para operaciones con HTML.
from weasyprint import HTML #Otras operaciones para HTML.
import matplotlib.pyplot as plt #Generar gr ficos.
from matplotlib.backends.backend_pdf import PdfPages #Guardar gr ficos en PDF.
from PyPDF2 import PdfFileMerger #Operaciones con PDF.
```

Listing 8: Librerías de RDFPY

De las liberías presentes Listing 8, jinja2 es muy importante porque permite convertir los DataFrames obtenidas por el script en tablas en html. Otra de la libería fundamental es WeasyPrint ya que permite convertir un archivo en HTML en un documento PDF. Y finalmente PyPDF2 permite fusionar archivos PDF y así crear un único archivo que contenga toda la información.

```
#Constantes y sus madres.
N = 200. #El n mero m ximo de muestras.
Nmin=190. #El n mero m nimo de muestras para considerar v lida la medida.
seemax = 6. #El valor m ximo del seeing para considerar v lidad la medida.

#Conjunto de nombre que heredar n las columnas de los archivos de D1 una vez
#se haya llamado a la funci n REPORT().
ListD1=['Fecha_D1', 'Ubicacion_D1', 'Objeto_D1', 'Seeing_D1', 'Maire_D1', 'Flujo1_D1', '
    Flujo2_D1',
        'Var_l_D1', 'Var_t_D1', 'Centelleo1_D1', 'Centelleo2_D1', 'r0l_D1', 'r0t_D1', '
    fwhml_D1',
        'fwmlt_D1', 'CCD_D1', 'Software_D1', 'Ventana_D1', 'EscalaPlaca_D1', 'Distancia_D1',
        'Diametro_D1', 'WinSpot_D1', 'Gain_D1', 'Texp_D1', 'Sampling_D1', 'N_D1', '
    Muestras_D1',
        'Return_D1']
#Lo mismo para D2.
ListD2=['Fecha_D2', 'Ubicacion_D2', 'Objeto_D2', 'Seeing_D2', 'Maire_D2', 'Flujo1_D2', '
    Flujo2_D2',
        'Var_l_D2', 'Var_t_D2', 'Centelleo1_D2', 'Centelleo2_D2', 'r0l_D2', 'r0t_D2', '
    fwhml_D2',
        'fwmlt_D2', 'CCD_D2', 'Software_D2', 'Ventana_D2', 'EscalaPlaca_D2', 'Distancia_D2',
        'Diametro_D2', 'WinSpot_D2', 'Gain_D2', 'Texp_D2', 'Sampling_D2', 'N_D2', '
    Muestras_D2',
        'Return_D2']
#Nombre por defecto que contienen los archivos antes de llamar a REPORT().
ListD1D2=['Fecha', 'Ubicacion', 'Objeto', 'Seeing', 'Maire', 'Flujo1', 'Flujo2',
        'Var_l', 'Var_t', 'Centelleo1', 'Centelleo2', 'r0l', 'r0t', 'fwhml',
        'fwmlt', 'CCD', 'Software', 'Ventana', 'EscalaPlaca', 'Distancia',
        'Diametro', 'WinSpot', 'Gain', 'Texp', 'Sampling', 'N', 'Muestras',
        'Return']

shListD1=[ 'Seeing_D1', 'Maire_D1', 'Flujo1_D1', 'Flujo2_D1',
        'Var_l_D1', 'Var_t_D1', 'Centelleo1_D1', 'Centelleo2_D1', 'r0l_D1', 'r0t_D1', '
    fwhml_D1',
        'fwmlt_D1', 'EscalaPlaca_D1', 'Distancia_D1',
```

```
        'Diametro_D1', 'WinSpot_D1', 'Gain_D1', 'Texp_D1', 'Sampling_D1', 'N_D1', '
    Muestras_D1',
        'Return_D1']
shListD2 = [ 'Seeing_D2', 'Maire_D2', 'Flujo1_D2', 'Flujo2_D2',
        'Var_l_D2', 'Var_t_D2', 'Centelleo1_D2', 'Centelleo2_D2', 'r0l_D2', 'r0t_D2', '
    fwhml_D2',
        'fwmlt_D2', 'EscalaPlaca_D2', 'Distancia_D2',
        'Diametro_D2', 'WinSpot_D2', 'Gain_D2', 'Texp_D2', 'Sampling_D2', 'N_D2', '
    Muestras_D2',
        'Return_D2']

#Nombre que heredar n las columnas cuando se generen los sheets en el archivo
# XLSX para la comparaci n entre D1 y D2.
diffD1D2=[ 'Dif_Seeing','Dif_Maire', 'Dif_Flujo1', 'Dif_Flujo2','Dif_Var_l'\
         , 'Dif_Var_t', 'Dif_Centelleo1'\
         , 'Dif_Centelleo2', 'Dif_r0l', 'Dif_r0t', 'Dif_fwhml', 'Dif_fwmlt'\
         ,'Dif_EscalaPlaca', 'Dif_Distancia','Dif_Diametro','Dif_WinSpot','Dif_Gain'\
         ,'Dif_Texp','Dif_Sampling', 'Dif_N', 'Dif_Muestras','Dif_Return']

#Nombre de columnas que recibir n un an lisis estad stico determinado.
shdiffD1D2=[ 'Dif_Seeing','Dif_Maire', 'Dif_Flujo1', 'Dif_Flujo2','Dif_Var_l'\
         , 'Dif_Var_t', 'Dif_Centelleo1'\
         , 'Dif_Centelleo2', 'Dif_r0l', 'Dif_r0t', 'Dif_fwhml', 'Dif_fwmlt'\
         ,'Dif_Muestras']

#Nombre de columnas que recibir n un an lisis estad stico determinado.
ListD1D2_EST = ['Seeing_D1','Seeing_D2','Flujo1_D1','Flujo1_D2','Flujo2_D1','Flujo2_D2' \
           , 'Var_l_D1','Var_l_D2','Var_t_D1','Var_t_D2','Centelleo1_D1','Centelleo1_D2'\
           ,'Centelleo2_D1','Centelleo2_D2']

#Nombre de la columnas empleadas en la funci n CompCol1().
comp1 = [ 'Objeto', 'CCD'] # NOTA: El Software no es igual. Y la ventana la tienen que
    reparar.

comp2 = [ 'EscalaPlaca', 'Distancia','Diametro', 'WinSpot', 'Gain' \
         , 'Texp', 'Sampling', 'N', 'Muestras', 'Return']

#Nombre de la columnas empleadas en la funci n CompCol2(). A la espera de
#solucionar unos problemas en DIMMA. Una vez resueltos emplear comp2.
comp3 = [  'Gain' \
         , 'Texp', 'Sampling', 'N', 'Muestras', 'Return']
```

Listing 9: Listas y constantes RDFPY.

Cabe destacar la importancia de las constantes que aquí se almacenan porque son necesarias para la clasificación por categorías de medidas. También las listas presentes son fundamentales dado que son usadas para diferenciar los datos procedentes de D1 y D2. Por otra parte, permiten realizar operaciones complejas como crear DataFrame de la combinación de otros dos o comparar que el contenido de una misma variable fila a fila por las funciones CompCol1 y ComCol2.

```
def Report(x,y):
    """
    La funcion Report() se ha creado para generar un estudio estad stico entre las medidas
    obtenidas por D1 y D2. As  mismo, se encarga de clasificar las medidas en funci n de
    filtros en el interior de la funci n.
    Estas clasificaciones son: validas, validas-iguales, no-iguales y
    no-validas. Para m s informaci n consultar README.pdf.
    """
    #Crea un documento gen rico donde introducir los resultados. Posteriormente
    #se cambiar  de nombre.
    writer = pd.ExcelWriter('datasheet.xlsx', datetime_format='yyyy-mm-dd hh:mm:ss') #
    Definimos formato fecha.
    D2 = x # arvhivos de D2.
    D1 = y # archivos de D1.
    D21 = D2
    D11 = D1
```

Listing 10: Función Report()

La función Report() se encarga prácticamente de caer de RDIPY la información correspondiente de los archivos CSV de D1 y D2. Además, que dentro de si misma alberga el resto de funciones. De hecho el resto de funciones se encuentra indexadas dentro de ésta.

```
def SeeVal(z):
        """
        Esta funci n SeeVal(z) lee fila por fila el archivo CSV ejecutado
        y compara para determinadas columnas que se cumplen unos criterios
        para considerar las medidas v lidas. Si no es as , ser  eliminadas
        del DataFrame.
        M s informaci n en el README.pdf.
        """
        errSeeing = [] #crea una lista vac a.
        for q in z.index: #itera en funci n de los valores del  ndice .
            #Si no se cumple la condici n la iteraci n continua, en caso
            #contrario.
            if (z.Seeing[q] > seemax == True) or (((z.Muestras[q] < Nmin).any() == True) \
                or ((z.Return[q] != 0).any() == True)):
                #print (q)
                #print('Valor de Seeing malo')
                errSeeing.append(q) #Se a ade el valor del  ndice  en la lista.
        return z.drop(errSeeing) #Los  ndices  recopilados se eliminan del DataFrame.
```

Listing 11: Función SeeVal(z)

La función SeeVal(z) se pueden considerar el primer filtro. Permite descriminar entre válidas y no-válidas. Como bien esos criterios ya han sido mencionados, el seeing no debe ser superior a 7, las muestras no pueden ser inferiores a 190 y la columna de return debe ser igual a 0. Si una medida no cumple alguna de estas condiciones será eliminada del DataFrame.

```
    def CompTiempo(n,l):
        """
        CompTiempo(n,l) tiene por cometido sincronizar el DataFrame
        correspondiente a D1 con D2, dado que el momento de adquisici n
        de la medida deben ser iguales.
        M s informaci n en README.pdf.
        """
        #Cambiamos el formato de contenido de la columna 'Fecha' y cambiamos
        # de calendario gregoriano a calendario juliana por facilidad de las
        # operaciones de comparaci n.
        n = pd.DatetimeIndex(n['Fecha'].values).to_julian_date()
        l = pd.DatetimeIndex(l['Fecha'].values).to_julian_date()
        interA = np.intersect1d(n,l, return_indices=True) #Compara la columna
        #'Fecha' en ambos DataFrame e indica valores coinciden y cu l es su
        #  ndice .
        return interA[1].tolist() #Convertimos los  ndeces  recopilados en una
        #lista.
```

Listing 12: Función CompTiempo(n l)

La misión de la función CompTiempo(n,l) es sincronizar ambos DataFrame. Bien porque al adquirir las medidas no fueron tomadas todas al mismo tiempo o bien la función SeeVal(z) elimina índices en uno de los DataFrames mientras que permanecen en el otro. Por lo tanto, al comparar que valores del tiempo se encuentran en ambos DataFrame al realizar la intersección podemos conocer su índice y seleccionar sólo los que coinciden.

```
    def CompCol1(f,g):
        """
        CompCol1(f,g) compara fila por fila que las columnas contenidas en
        comp1 sea su contenido igual.
        M s informaci n en README.pdf.
        """
        h = [] #Crea una lista vac a.
        for q in range(len(comp1)): # Genera una iteraci n.
```

```
            fcomp = (f[comp1[q]].str.strip()) #Elimina los espacios dentro de
            # la celda a los lados del contenido.
            fcomp = fcomp.str.replace(' ','') #Reemplaza los espacios dentro
            #de la celda.
            #print(fcomp)
            gcomp = (g[comp1[q]].str.strip())
            gcomp = gcomp.str.replace(' ','')
            #print(gcomp)
            for r in f.index: #Genera una iteraci n en funci n del  ndice  de f.
                #print(r)
                if fcomp[r] != gcomp[r]: #Compara fila por fila
                    #print('Hay algo que eliminar')
                    h.append(r) #Si no es igual el contenido, entonces el
                    # ndice  se agraga ah .
        return h #Devuelve la lista de  ndices  de filas que no son iguales.
```

Listing 13: Función CompCol1(f g)

En Listing 13 es suficiente con la información de los comentarios del script. No obstante, recalcar que corresponde con el filtro que permite descriminar entre iguales y noiguales. Esta función debido a problemas con los espacios de los string que contiene las columnas es una versión distinta de Listing 14 que opera con floats.

```
    #En la siguiente funci n se pide lo mismo que en la anterior. No obstante,
    #se aplica a columnas que no tienen problemas con los espacios.
    def CompCol2(f,g):
        """
        CompCol2(f,g) compara fila por fila que las columnas contenidas en
        comp1 sea su contenido igual.
        M s informaci n en README.pdf.
        """
        h = []
        for q in range(len(comp3)):
            for r in f.index:
                fcomp = (f[comp3[q]])
                gcomp = (g[comp3[q]])
                if fcomp[r] != gcomp[r]:
                    #print('Hay algo que eliminar') #Inf.
                    h.append(r)
        return h
```

Listing 14: Función CompCol2(f g)

Realiza la misma tarea que Listing 13. Por ello, no es necesario añadir más información.

```
    def CreateDF1(j,k):
        """
        CreateDF1(j,k) introduce las columnas de j y k en un nuevo DataFrame vac o
        colocando primero una columnas de D1 seguido de la columna equivalente
        de D2. Ej. primero la columna 'Seeing_D1' y luego la columna 'Seeing_D2'
        M s informaci n en README.pdf.
        """
        cloe = pd.DataFrame() #DataFrame vac o.
        for i in range(len(ListD1)): #Generamos un bucle.
            cloe[ListD1[i]] = j[ListD1D2[i]] #Selecciona una columna en j y la a ade.
            cloe[ListD2[i]] = k[ListD1D2[i]] #Selecciona una columna en k y la a ade.
        return cloe #Devuelve el nuevo DataFrame.
```

Listing 15: Función CreateDF1(j k)

La función CreateDF1(j k) tiene por cometido crear un nuevo DataFrame de forma que introduce primero una columna de D1 para una variable como puede ser 'Seeing_D1' y posteriormente introduce la columna 'Seeing_D2' de las medidas de D2. Esta función tiene gran importancia junto CreateDF2(j) porque devuelve un DataFrame que puede ser convertido en una tabla de HTML o ser insertada en un archivo XLSX.

```
    def CreateDF2(j):
        """
        CreateDF2(j) es una funci n creada para generar un DataFrame que contenga
        la diferencia entre columnas con el mismo tipo de informaci n. Ej.
        'Seeing_D1' - 'Seeing_D2'.
        M s informaci n en README.pdf.
        """
        maria = pd.DataFrame() #DataFrame vac o
        maria['Fecha'] = j['Fecha_D1'] #A ade la columna con la fecha y hora.
        maria['Objeto'] = j['Objeto_D1'] #A ade la columna con el objeto.
        for a in range(len(shListD1)): #Genera una iteraci n para operar fila a fila.
            #Si el nombre de las columnas coinciden con la indicadas se
            # crear  un string que crea un c digo de error.
            if (shListD1[a]=='Return_D1') & (shListD2[a]=='Return_D2'):
                #Genera un c digo de error.
                maria[diffD1D2[a]] = (j[shListD1[a]].astype(int)).astype(str)+'-'+(j[
shListD2[a]].astype(int)).astype(str)
            else:
                #Realiza una operaci n de resta entre las columnas.
                maria[diffD1D2[a]] = abs(j[shListD1[a]]-j[shListD2[a]])

        return maria #Devuelve un DataFrame con la diferencia.
```

Listing 16: Función CreateDF2(j)

La función CreateDF2(j) es diferente a CreateDF1(j,k) debido a que calcula la diferencia entre las columnas de una misma variable, ej. la masa de aire, de D1 y D2. Al final, devuelve un DataFrame con la diferencia de las variables tipo float o int. Por lo general, esta función está diseñada para operar con el DataFrame fruto de CreateDF1(j,k).

```
    def EstSee(s):
        """

        EstSee(s) es una funci n que realiza operaciones estad sticas como
        obtener la media, desviaci n est ndar, valor m ximo y valor m nimo
        para las columnas indicadas.
        M s informaci n en README.pdf.
        """
        #Se crea un DataFrame vac o pero con el  ndice  definido.
        EST = pd.DataFrame(index=['Media','Desviacion estandar','Max','Min'])
        for t in range(len(shdiffD1D2)): #Creamos un bucle.
            estad = np.array([]) #Crea un array vac o.
            #La siguiente condici n se ha creado en el caso de que se desee
            #a adir cifras decimales a los datos correspondientes al flujo.
            if (shdiffD1D2[t]=='Dif_Flujo1') or (shdiffD1D2[t]=='Dif_Flujo2'):
                #Se a ade en el array vac o el valor calculado para una
                #determinada operaci n estad stica.
                estad = np.append(estad,np.mean(abs(s[shdiffD1D2[t]]))).round(4))
                estad = np.append(estad,np.std(abs(s[shdiffD1D2[t]]))).round(4))
                estad = np.append(estad,np.max(abs(s[shdiffD1D2[t]]))).round(4))
                estad = np.append(estad,np.min(abs(s[shdiffD1D2[t]]))).round(4))
            else:
                estad = np.append(estad,np.mean(abs(s[shdiffD1D2[t]]))).round(4))
                estad = np.append(estad,np.std(abs(s[shdiffD1D2[t]]))).round(4))
                estad = np.append(estad,np.max(abs(s[shdiffD1D2[t]]))).round(4))
                estad = np.append(estad,np.min(abs(s[shdiffD1D2[t]]))).round(4))
                #print(estad)
            #Ahora generamos una Serie predefiniendo el  ndice .
            d = pd.Series(estad.tolist(), index=['Media'\
                          ,'Desviacion estandar','Max','Min']).to_frame(name=diffD1D2[t])
            #Lo introducimos dentro del DataFrame.
            EST.insert(t,shdiffD1D2[t],d)
        return EST
```

Listing 17: Función EstSee(s)

De la función EstSee(s) es otra de las funciones más importantes de este script dado que permite calcular distintos parámetros estadísticos de las columnas del DataFrame deseado. Aunque evidentemente sólo se puede aplicar a aquellas que cuyos datos son floats o int.

```
    #La siguiente funci n es igual que la anterior a excepci n de que realiza
    #la operaci n de mediana.
    def EstCol(s):
        """
        EstCol(s) es una funci n que realiza operaciones estad sticas como
        obtener la media, mediana, desviaci n est ndar, valor m ximo y
        valor m nimo para las columnas indicadas.
        M s informaci n en README.pdf.
        """
        EST1 = pd.DataFrame(index=['Media','Mediana','Desviacion estandar','Max','Min'])
        for t in range(len(ListD1D2_EST)):
            estad = np.array([])
            estad = np.append(estad,np.mean(abs(s[ListD1D2_EST[t]])).round(4))
            estad = np.append(estad,np.median(abs(s[ListD1D2_EST[t]])).round(4))
            estad = np.append(estad,np.std(abs(s[ListD1D2_EST[t]])).round(4))
            estad = np.append(estad,np.max(abs(s[ListD1D2_EST[t]])).round(4))
            estad = np.append(estad,np.min(abs(s[ListD1D2_EST[t]])).round(4))
            d = pd.Series(estad.tolist(), index=['Media'\
                        ,'Mediana','Desviacion estandar','Max','Min']).to_frame(name=
    ListD1D2_EST[t])
            EST1.insert(t,ListD1D2_EST[t],d)
        return EST1
```

Listing 18: Función EstCol(s)

La función EstCol(s) hace prácticamente lo mismo que EstSee(s). No obstante, está aplicado a otros DataFrame.

Una vez explicados el funcionamiento de cada función procedemos a la lista de instrucciones que indican el proceso de obtención de los resultados que devuelve el script. Recordar que la función Report(x,y) trae a RDFPY los DataFrame cargados en RDIPY correspondientes a D1 y D2. Comenzamos con la eliminación de la medidas no-válidas.

```
    D2 = SeeVal(D2) #Llama SeeVal() para eliminar las medidas no-v lidas.
    D1 = SeeVal(D1)
```

Listing 19: Selección de medidas válidas.

Tras aplicar la función SeeVal(z) al DataFrame de D2 se habrán seleccionado únicamente las medidas válidas. Lo mismo sucede para D1.

```
    intA = CompTiempo(D2, D1) #Obtenemos los  ndices  de los valores que est n
    intB = CompTiempo(D1, D2) #sincronizados.
```

Listing 20: Índice de medidas sincronizadas.

Después de haber obtenido las medidas válidas no nos garantiza que no hayan índices desparejados entre D1 y D2. Es por esto que aplicamos la función en busca de fechas que coincidan entre ambos DataFrames. Luego obtendremos una lista con los índices de las fechas que coinciden entre ambos.

```
    D2clean = D2.iloc[intA] #Seleccionamos s lo las medidas v lidas.
    D1clean = D1.iloc[intB]
```

Listing 21: Selección de medidas sincronizadas.

A continuación, en Listing 21 se seleccionan mediante el índices aquellas medidas que están sincronizadas.

```
    D2clean3 = D2clean #Renombramos la misma variable para otro fines.
    D1clean3 = D1clean
```

Listing 22: Renombramiento de variables.

```
NoCoinciden1 = CompCol1(D1clean,D2clean) # Obtenemos la lista de  ndices .
D2clean1 = D2clean.drop(NoCoinciden1) #Eliminamos dichos  ndices  del
D1clean1 = D1clean.drop(NoCoinciden1) # DataFrame inicial.
```

Listing 23: Eliminamos medidas no-iguales.

En Listing 23 se aplica la función CompCol1() con la finalidad de obtener los índices de la medidas que no son iguales. Mediante la comparación de columnas, como Ubicacion_D1 y Ubicacion_D2, se determina si ambas son iguales fila a fila. En caso de que sean diferente se añade el índice a la lista. Posteriormente, dichos índices serán eliminados del DataFrame correspondiente. Por lo que el resultado final serán medidas válidas-iguales.

```
D2conf1 = D2clean3.loc[NoCoinciden1,:] #Ahora seleccionamos los datos
D1conf1 = D1clean3.loc[NoCoinciden1,:] # indicados por los  ndices .
```

Listing 24: Seleccionamos medidas no-iguales.

En Listing 24 hacemos justo lo contrario, seleccionamos las medidas no-iguales para almacenar en otro DataFrame para su uso posterior.

```
NoCoinciden2 = CompCol2(D1clean1,D2clean1) #Realizamos la misma
D2clean2 = D2clean1.drop(NoCoinciden2) # operaci n para la siguiente
D1clean2 = D1clean1.drop(NoCoinciden2) # funci n.
D2clean = D2clean2
D1clean = D1clean2
```

Listing 25: Eliminamos medidas no-iguales con CompCol2.

Volvemos hacer los mismo que en Listing 23 pero aplicado a columnas que contienen int o floats. Ej. EscalaPlaca_D1 y EscalaPlaca_D2.

```
D2conf2 = D2clean3.loc[NoCoinciden2,:]
D1conf2 = D1clean3.loc[NoCoinciden2,:]
D2conf = pd.concat([D2conf1,D2conf2]) #Fusionamos contenidos.
D1conf = pd.concat([D1conf1, D1conf2])
```

Listing 26: Seleccionamos medidas no-iguales y fusionamos resultados.

En Listing 26 seleccionamos de nuevo de cada DataFrame las medidas no-válidas y posteriormente fusionamos con los resultados obtenidos en Listing 24. Por lo tanto en D2conf y D1conf se almacena todas las medidas que se han declarado no-iguales.

```
intC = D2.index #Llama al  ndice  de las medidas v lidas.
ErrD21 = D21.drop(intC) #Se eliminan dichas medidas.
ErrD11 = D11.drop(intC)
ErrD2 = ErrD21.reset_index().loc[CompTiempo(ErrD21, ErrD11)]
ErrD1 = ErrD11.reset_index().loc[CompTiempo(ErrD11, ErrD21)]
ErrD2 = ErrD2.dropna(axis=0)
ErrD1 = ErrD1.dropna(axis=0)
```

Listing 27: Seleccionamos medidas no-válidas.

En Listing 27, intC se define como los índices de las medidas válidas de D2. No obstante, si estos índices son eliminados a D21 y D11 obtendremos la medidas no válidas. A continuación se le aplicar la función CompTiempo() ya que puede contener medidas desincronizadas que deben ser eliminadas dado que no entran en la clasificación. Y finalmente se aplica dropna() para borrar aquellos huecos vacíos.

```
    valigu = CreateDF1(D1clean,D2clean) #Crea un DataFrame con las validas-iguales.
    valigu.index.name = 'DS2' #A adimos el nombre de la sheet a la tabla.
    noigu = CreateDF1(D1conf,D2conf) #Crea un DataFrame con las no-iguales.
    noigu.index.name = 'DS3'
    val = CreateDF1(D1clean3,D2clean3) # Crea un DataFrame con las validas.
    val.index.name = 'DS1'
    noval = CreateDF1(ErrD1,ErrD2) #Crea un DataFrame con las no-validas.
    noval.index.name = 'DS4'
```

Listing 28: Creando DataFrame combinados.

En el fragmento de Listing 28 se lleva acabo la aplicación de la función CreateDF1() que genera un DataFrame compuesto siempre por una columnas de D1 y posteriormente por una columna de D2 para la misma variable. Se repite continuamente la estructura resultando Seeing_D1, Seeing_D2, Maire_D1, Maire_D2, etc . Añadir que además se crea un DataFrame para cada categoría. Siendo de este modo, valigu correspondiente a válidas-iguales, val a válidas, noigu a no-iguales y noval a no-válidas.

```
    val.to_excel(writer,'DS1') #Se guardan en el archivo XLSX creado.
    valigu.to_excel(writer,'DS2')
    noigu.to_excel(writer,'DS3')
    noval.to_excel(writer, 'DS4')
```

Listing 29: Guardamos los resultados en el archivo excel.

En el fragmento Listing 29, se guardan los DataFrame en formato tabla en un archivo XLSX. Dónde a cada categoría le corresponde un sheet. Los DataFrames corresponden a Listing 28.

```
    difvaligu = CreateDF2(valigu) #Crea un DataFrame con la comparaci n de las validas-
    iguales.
    difval = CreateDF2(val) #Crea un DataFrame con la comparaci n de las validas.
    difnoigu = CreateDF2(noigu) #Crea un DataFrame con la comparaci n de las no-iguales.
    difnoval = CreateDF2(noval) #Crea un DataFrame con la comparaci n de las no-validas.
```

Listing 30: Crea los DataFrames de las medidas comparadas.

En Listing 30 ocurre lo mismo que Listing 28 a excepción de que se crean los DataFrames para las diferencias entre D1 y D2. Se conservan las categorías.

```
    difval.to_excel(writer,'COMP-DS1') #Guardamos resultados en el archivo XLSX.
    difval.index.name = 'COMP-DS1' #A adimos el nombre de la sheet a la tabla.
    difvaligu.to_excel(writer,'COMP-DS2')
    difvaligu.index.name = 'COMP-DS2'
    difnoigu.to_excel(writer,'COMP-DS3')
    difnoigu.index.name = 'COMP-DS3'
    difnoval.to_excel(writer,'COMP-DS4')
    difnoval.index.name = 'COMP-DS4'
    writer.save() #Guardamos el archivo XLSX.
```

Listing 31: Guardamos los resultados en el archivo excel.

En Listing 31 se escriben las líneas que indican que los DataFrames deben ser guardados en el mismo archivo XLSX pero en otros sheets en lo que se indica en su nombre que son fruto de una comparación. Además, se conservan las categorías.

A partir de este punto, los DataFrames que se obtienen son con la finalidad de crear el report final con los resultados estadísticos entre otros.

```
    pd.options.display.float_format = "{0:.6g}".format #Definimos el formato de los n meros
    .
    Val = EstSee(difval) #Crea un DataFrame con los resultados estad sticos
    # de la comparaci n de las medidas validas.
    Val.index.name = 'COMP-DS1'
```

```
    Valigu = EstSee(difvaligu) # Lo mismo para las medidas validas-iguales.
    Valigu.index.name = 'COMP-DS2'
    Noigu = EstSee(difnoigu) # Para las no-iguales.
    Noigu.index.name = 'COMP-DS3'
    Noval = EstSee(difnoval) # Para las no-validas.
    Noval.index.name = 'COMP-DS4'
```

Listing 32: Estadísticos de comparaciones entre columnas de una misma variable.

Ahora en Listing 32 se realizan la creación de los DataFrames con los resultados estadísticos para cada categoría a partir de los DataFrames de diferencias.

```
    pd.options.display.float_format = "{0:.6g}".format
    Val_est = EstCol(val) #Crea un DataFrame con los resultados estad sticos
    # de las medidas validas.
    Val_est.index.name = 'DS1'
    Valigu_est = EstCol(valigu)#Crea un DataFrame con los resultados estad sticos
    #de las medidas validas-iguales.
    Valigu_est.index.name = 'DS2'
```

Listing 33: Estadísticos de DS1 y DS2

En este fragmento también se obtiene los DataFrames con los resultados estadísticos para valigu y val. Al aplicar la función EstCol() se obtiene otros parámetros estadísticos extra. Además, los parámetros no se obtiene para las diferencias entre D1 y D2. En cambio se obtienen los resultados estadísticos para ambos. Ej. para Seeing_D1 y Seeing_D2.

Ahora continuamos con la estimación del número de medidas por categorías incluyendo a desincronizadas, así como porcentaje para D1 y D2.

```
    Nvaligu = difvaligu.shape[0] # Definimos el n mero de filas.
    Nnoigu = difnoigu.shape[0]
    Nnoval = difnoval.shape[0]
    Nval = difval.shape[0]
    NTD2 = D21.shape[0]
    NTD1 = D11.shape[0]
```

Listing 34: Número de medidas en cada categoría.

En Listing 34, mediante la aplicación de .shape al DataFrame de cada categoría se obtiene el número de medidas.

```
    DsincD2 = NTD2 - Nvaligu - Nnoigu - Nnoval #Definimos el n mero de medidas
    desincronizadas.
    DsincD1 = NTD1 - Nvaligu - Nnoigu - Nnoval
    #Obtenemos los porcentajes de medidas.
    PTEST = (np.asarray([Nval,Nvaligu,Nnoigu,Nnoval,DsincD1,NTD1])*(100./NTD1))
    PSOFT = (np.asarray([Nval,Nvaligu,Nnoigu,Nnoval,DsincD2,NTD2])*(100./NTD2))
    PTEST = PTEST.round(1).tolist()
    PSOFT = PSOFT.round(1).tolist()
```

Listing 35: Número de medidas en cada categoría.

Posteriormente, conociendo el total de medidas y realizando una operación sencilla averiguamos el número de desincronizadas. Después creamos un array que a través de unas operaciones podemos establecer el porcentaje de medidas que pertenecen a cada categoría.

```
    #Generamos a DataFrame con los resultados.
    Num = pd.DataFrame({'N   de medidas D1':[Nval,Nvaligu,Nnoigu,Nnoval,DsincD1,NTD1] \
                        ,'Porcentajes de D1':PTEST\
                        ,'N   de Medidas D2':[Nval,Nvaligu,Nnoigu,Nnoval,DsincD2,NTD2] \
                        ,'Porcentaje de D2': PSOFT\
                        }, index=['DS1','DS2' ,'DS3','DS4','Desincronizadas','Total'])
```

Listing 36: Número de medidas en cada categoría.

En Listing 36 creamos un DataFrame que contiene el número de medidas de cada categoría con su correspondiente porcentaje para D1 y D2.

```
    #Creamos un DataFrame para la cebecera del report que indique la ubicaci n
    #del DIMMA, la fecha de inicio, la fecha de finalizaci n, el software
    #usado para D1 y el software usado para D2.
    U = val['Ubicacion_D2'].index[0] #El primer valor del  nidice .
    V = val['Ubicacion_D2'].index[-1] #El  ltimo  valor del  nidice .
    Datos=pd.DataFrame({'Ubicaci n':[val['Ubicacion_D2'][U]] \
                        ,'Fecha inicial':[val['Fecha_D2'][U]]\
                        ,'Fecha final':[val['Fecha_D2'][V]]
                        ,'D1':[val['Software_D1'][U]]\
                        ,'D2':[val['Software_D2'][U]]})
```

Listing 37: Información de la cabecera.

En Listing 37, se crea un nuevo DataFrame que contiene información importante para cada report como es la ubicación del DIMMA, fecha de inicio, fecha de finalización, software empleado en D1 y software empleado en D2.

Con Listing 37 hemos finalizado la creación de DataFrames que se introducirán en la estructura del archivo HTML para la reación del report. Para ello, a continuación se explica como se lleva acabo el proceso de creación del archivo HTML y su posterior conversión a PDF.

```
    env = Environment(loader=FileSystemLoader('.'))
    template = env.get_template("report_templates.html") #Carga el archivo HTML
    # report_templates.html para tener una estructura d nde insertar los DataFrames
    # creados anteriormente. Adem s, el archivo contiene especificaciones sobre
    # el tama o de la fuente, tipo de borde, etc.
```

Listing 38: Carga el archivo HTML.

En Listing 38 lo que se ha hecho es crear mediante Jinja2 un environment (un medio) que es posible aplicar para varios fines. En mi caso, me interesa que lea el archivo HTML que contiene la estructura dónde deben ser ubicados las tablas. Para ello se define en el archivo HTML mediante una variable (Ver Anexo X).

```
    Valigu.style.set_properties(**{'font-size':'8pt'}).render() #Definimos el tama o de la
    fuente.
    Val.style.set_properties(**{'font-size':'8pt'}).render()
    Noigu.style.set_properties(**{'font-size':'8pt'}).render()
    Noval.style.set_properties(**{'font-size':'8pt'}).render()
    Num.style.set_properties(**{'font-size':'8pt'}).render()
    Datos.style.set_properties(**{'font-size':'8pt'}).render()
```

Listing 39: Define propiedades del tamaño de la fuente.

En Listing 39 se define las propiedades de la fuente que tendrán en el archivo HTML. En este caso, se específica únicamente el tamaño de la fuente.

```
    template_vars = {"title" : "Comparativa D1 y D2",
                     "tabla_cero": Datos.to_html(table_id="t01") \
                     ,"primera_tabla": Num.to_html(table_id="t01") \
                     , "segunda_tabla": (Val_est.transpose()).to_html(table_id="t01") \
                     , "tercera_tabla":(Valigu_est.transpose()).to_html(table_id="t01") \
                     , "cuarta_tabla":(Val.transpose()).to_html(table_id="t01") \
                     , "quinta_tabla":(Valigu.transpose()).to_html(table_id="t01")\
                     , "sexta_tabla":(Noigu.transpose()).to_html(table_id="t01")\
                     , "septima_tabla":(Noval.transpose()).to_html(table_id="t01")}
```

Listing 40: Insertamos los DataFrames en el archivo HTML.

En Listing 40, contiene el fragmento de código que permite la conexión con el archivo HTML, ya que sus variables están relacionadas espacialmente con el orden expuesto. Además, se puede ver el código que permite la conversión a HTML de la tabla y table_id permite elegir el tipo de tabla.

```
    html_out = template.render(template_vars)
    #print(html_out)
    HTML(string=html_out).write_pdf("report-DIMMA.pdf") #Guardamos el contenido en PDF.
```

Listing 41: Crea el archivo HTML

En Listing 41, el código permite la creación del archivo HTML con las tablas incluidas mediante el comando template.render(template_vars). Posteriormente aplicando Py2PDF y su descendencia de HTML podemos convertir el archivo HTML con las tablas en un documento PDF.

```
    pp = PdfPages('imagenes.pdf') #Crea un PDF donde guardar la gr ficas.
```

Listing 42: Creación de documento PDF para guardar figuras.

En Listing 42, se aplica la librería Py2PDF para la creación de un documento PDF que permita guardar las figuras creadas mediante la descendencia PdfPages.

```
    #Ahora se generan los gr ficos con los resultados de los DataFrames para
    #v lidas y v lidas-comparadas.
    plt.figure(1,figsize=(8.27, 11.69), dpi=100) #Definimos el tama o y calidad de las
    gr ficas.
    ax1=plt.subplot(211) #Crea una sub gr fica.
    #Genera el gr fico.
    plt.plot(val['Fecha_D1'], val['Seeing_D1'],val['Fecha_D1'], val['Seeing_D2'])
    plt.legend(('D1','D2')) #Crea una leyenda.
    plt.xticks(rotation='vertical') # Los valores en el eje x est n en posici n vertical.
    plt.ylabel('Seeing in arcsec') # Nombre del eje y.
    plt.title('Measurements of seeing for DS1') #T tulo de la gr fica.
    plt.setp(ax1.get_xticklabels(), visible=False) # No muestra el contenido del eje x.
    ax1.yaxis.grid() #A ade un grid s lo para y.

    ax2=plt.subplot(212, sharex=ax1) # Indica que el eje x es compartido por las dos
    gr ficas.
    plt.plot(val['Fecha_D1'], difval['Dif_Seeing'])
    plt.xticks(rotation='vertical')
    plt.ylabel('Dif. Seeing in arcsec ')
    plt.setp(ax2.get_xticklabels(), fontsize=6) #Define el tama o de los n meros.
    plt.xlabel('Time')
    plt.title('Diference of seeing between D1 and D2, for DS1')
    ax2.yaxis.grid()

    pp.savefig() #Guardamos las gr ficas en el archivo PDF.
```

Listing 43: Gráficos de DS1.

En Listing 43 se pretende en generar dos gráficas que compartan el eje X. Ambas contienen información sobre la evolución temporal del seeing. Mientras que en el eje Y, en la primera se muestra el valor de seeing para D1 y D2. En cambio, para la segunda figura se muestra la diferencia del seeing.

Entre los comandos empleado en Listing 43, destacan el ajuste al tamaño de un hoja A4 y la calidad de la imagen. Recalcar que se elegido la opción de generar dos gráficas mediante subplot para poder vincular los ejes X de cada una. Aunque en realidad consiste en indicar al script que oculte los valores de referencia en el eje X de la primera gráfica mediante visible=False. Además, para evitar la superposición de los valores temporales se han rotado 90º plt.xticks(rotation=90). Los comandos restantes son los usuales, así como el nombre de los ejes, añadir una leyenda, generar la figura o añadir un grid, que en este caso es sólo horizontal.

```
    #Borra el contenido de las gr ficas para evitar sobreescritura.
    plt.clf()
    plt.cla()
    plt.close()
```

Listing 44: Borra el contenido de las figuras.

En Listing 44, se pretende vaciar el contenido guardado en las figuras. Ya que al crear una nueva figura puede suceder que se introducan valores de la anterior. Con ello se evita que en las sucesivas iteraciones se acumule información y la última figura creada sea una combinación de todas las iteraciones anteriores.

```
    pp.close() #Cerramos el documento PDF.
```

Listing 45: Cierra el documento PDF.

A continuación, en Listing 45 se cierra el documento PDF que contiene a las figuras.

```
    pdfs = ['report-DIMMA.pdf','imagenes.pdf'] #Crea una lista con ambos
    # archivos PDF.
    merger = PdfFileMerger() #Permite fusionar archivos PDF.
    for pdf in pdfs:
        merger.append(pdf) #Fusi n de archivos PDF.
    merger.write('resultados.pdf') #Guarda el resultado en otro archivo PDF.

    return writer, merger #Devuelve el archivo XLSX 'datasheet.xlsx' y
    #el archivo PDF 'resultados.pdf'.
```

Listing 46: Fusión de archivos PDF.

En Listing 46, el objetivo es crear un único documento PDF que albergue el contenido de las tablas con la información estadística y las figuras. Es por ello que es necesario unir ambos documentos PDF. Se lleva acabo mediante el empleo de WeasyPrint con la descendencia de PdfFileMerger(). Con ayuda de un bucle for y una previa lista creada con los dos documentos PDF creados, se pueden unir en uno sólo. Finalmente, el documento se guarda con un nombre genérico, y el return de la función Report() devuelve a RDIPY los archivos XLSX (con otro nombre genérico) y el documentos PDF completo.

# 8 RDIPY después de RDFPY

Una vez finalizado la creación del documento PDF con tablas y figuras que será el report final, y el archivo XLSX con el contenido de las medidas para cada categoría. Es necesario volver al script RDIPY ya que se encargará de cambiar el nombre de los archivos XLSX y documento PDF para evitar su confusión dentro del directorio de salida.

```
    outputfilexlsx = re.sub('.csv','.xlsx',fileD2)#Cambia el nombre de la ra z.
    outputfilepdf = re.sub('.csv','.pdf',fileD2)
    outputfilexlsx = re.sub(dir_in,dir_out,outputfilexlsx)#Cambia el directorio.
    outputfilepdf = re.sub(dir_in,dir_out,outputfilepdf)
    os.rename('datasheet.xlsx', outputfilexlsx)#Modifica el nombre del
    #directorio donde ser  guardado el archivo final.
    os.rename('resultados.pdf', outputfilepdf)
    print("\n"
          "Completed the analysis of {}".format(outputfilepdf))
    #Informa de que se ha finalizado la creaci n del documentos para el
    #archivo generado.
```

Listing 47: Cambia el nombre del archivo PDF y lo guarda en el directorio correspondiente.

Para modificar el nombre, se parte del nombre de entrada del archivo en CSV para D2. Se modificia la raíz de .csv por .xlsx o .pdf según sea el caso. También, tener en cuenta que el nombre que contiene fileD2 tiene incluido el camino al directorio de entrada, que debe ser sustituido para guardar el archivo en el directorio correspondiente. Después, los nombres genéricos de los archivos XLSX y PDF son sustituidos por nombres específicos acorde a los archivos de entrada con os.rename(). Y al final se imprime en pantalla el nombre del documento pdf con su nombre específico indicando la obtención del report.

Una vez alcanzado este punto, el generador en Listing 7 dará dos nuevos archivos de entrada sucesivamente hasta analizar todos los archivos disponibles en el directorio. Generando un report específico para cada par de archivos de entrada.

Añado enlace a video para saber como usar el script en Linux [2] y en Windows [3].

[2] https://drive.google.com/open?id=1AIhjRmeKWHuDMTpKXZku_PvcWj-NNXEQ
[3] https://drive.google.com/open?id=1al0lqiC3Hcqf8kYesN1F11CR5gWcLu04