Automatización de Scripts en Python

Los **scripts** en Python **son archivos .py que nos permiten realizar tareas repetitivas de forma automática**, ahorrándonos así mucho tiempo.

Nos permite **automatizar una amplia variedad de tareas**, desde procesamiento de datos y manipulación de archivos hasta interacciones con servicios web y más.

Formas de ejecutar los Scripts

Podemos ejecutar las automatizaciones de forma **manual** o de forma **programada**.

**Módulo OS**

El módulo **os** es **esencial para tareas de automatización que requieren interacción con el sistema operativo**.

Nos permite **manipular archivos y directorios**, **ejecutar comandos del sistema** y **gestionar aspectos relacionados con el entorno del sistema operativo** desde tus scripts de Python.

1. **Manipulación de directorios y archivos**: Puedes crear, eliminar, mover y renombrar directorios y archivos utilizando funciones del módulo **os** como **os.mkdir()**, **os.rmdir()**, **os.rename()**, **os.remove()**, **os.listdir()**, entre otros.
2. **Navegación en el sistema de archivos**: El módulo **os** permite obtener información sobre rutas de archivos, como la verificación de si un archivo o directorio existe (**os.path.exists()**), la obtención del directorio actual (**os.getcwd()**), y la construcción de rutas de archivos de manera plataforma independiente (**os.path.join()**).
3. **Ejecución de comandos del sistema**: Puedes ejecutar comandos del sistema operativo desde un script de Python utilizando la función **os.system()**. Esto te permite automatizar tareas que normalmente se realizarían desde la línea de comandos.
4. **Variables de entorno**: Puedes acceder y manipular variables de entorno del sistema operativo a través de **os.environ**. Esto es útil cuando necesitas configurar variables de entorno específicas para tu script.
5. **Cambiar el directorio de trabajo**: Utilizando **os.chdir()**, puedes cambiar el directorio de trabajo actual de tu script, lo que es útil para navegar entre diferentes carpetas en tu sistema de archivos.
6. **Gestión de permisos y propietarios de archivos**: Puedes usar funciones del módulo **os** para establecer o verificar los permisos y el propietario de archivos y directorios, aunque ten en cuenta que la capacidad de hacerlo puede depender de los permisos del usuario que ejecuta el script.
7. **Otras operaciones de sistema**: **os** también proporciona funciones para obtener información sobre el sistema operativo en el que se ejecuta el script, como el nombre del sistema (**os.name**), el acceso a las funciones específicas del sistema operativo (por ejemplo, **os.uname()** en sistemas Unix), y más.

import os

#-----1) acceder al nombre de la ruta del script en el que estamos-----

print("Nombre del directorio actual: ")

print(os.getcwd()) #nos devuelve la ruta en forma de string

#-----2) listar los archivos y carpetas que están dentro de la carpeta (directorio) en la que estamos-----

print("")

print("Lista de los archivos del directorio actual:")

print(os.listdir()) #obtener los archivos del directorio actual

print("Lista de los archivos del directorio especificado:")

print(os.listdir('carpeta\_para\_pruebas')) #obtener los archivos de un directorio en específico

#-----3) concatenar rutas (paths)-----

#como primer parámetro pasamos la ruta actual y luego lo que deseamos concatenarle

print("")

print("Concatenación entre la ruta actual y la carpeta\_para\_pruebas:")

print(os.path.join(os.getcwd(), 'carpeta\_para\_pruebas')) #NO es una creación de carpetas, sencillamente es una concatenación de strings

#-----4) crear archivos y carpetas-----

print("")

print("make directories (mkdir) utilizado para crear una carpeta")

#os.mkdir('carpetas\_creadas') #si ya existe una carpeta con el mismo nombre nos generará un error

#crear carpetas dentro de carpetas (en el primer parámetro ponemos la carpeta donde crearemos la nueva y en el segundo pondremos la carpeta que crearemos dentro de la anterior)

print("")

print("makedirs utilizado para crear carpetas dentro de otras carpetas")

#os.makedirs(os.path.join('carpetas\_creadas', 'carpeta\_mkdir'))

#os.makedirs(os.path.join('carpetas\_creadas', 'carpeta\_makedirs'))

#-----5) renombrar archivos o carpetas-----

print("")

print("os.rename() utilizado para renombrar a la carpeta\_pruebas")

#os.rename('carpeta\_pruebas', 'carpeta\_para\_pruebas')

#otra forma de renombrar es buscando coincidencias dentro de un bucle for que contenga todos los archivos y carpetas del directorio

#en este ejemplo renombraremos todos los archivos y carpetas del directorio actual que terminen en .csv

#for archivo in os.listdir() :

#    if archivo.endswith('.csv') :

#        os.rename(archivo, f'rename\_de\_{archivo}')

#-----6) comprobar si un archivo o carpeta existe en el directorio actual-----

print("")

print("os.path.exists() utilizado para verificar la existencia en el directorio actual")

print(os.path.exists('carpeta\_makedirs')) #en este caso será falso ya que esta carpeta no está en este directorio, sino que es una subcarpeta

#-----7) Metadata (data que describe a otra data)-----

#una de las funcionalidades es la de obtener la ruta absoluta de un archivo o carpeta:

print("")

print("La ruta absoluta de la carpeta\_pruebas es: ")

print(os.path.abspath('carpeta\_pruebas'))

#además de esta hay varias funcionalidades mas

**openpyxl**

Una de las bibliotecas más populares para **automatizar tareas repetitivas con hojas de cálculo de Excel utilizando Python** es **openpyxl**.

1. Instala la biblioteca **openpyxl** si aún no lo has hecho. Puedes hacerlo utilizando **pip**:

pip install openpyxl

1. Importa la biblioteca **openpyxl** en tu script Python:

import openpyxl

1. Abre un archivo Excel existente o crea uno nuevo. Por ejemplo, para abrir un archivo existente:

archivo\_excel = openpyxl.load\_workbook('nombre\_del\_archivo.xlsx')

1. Accede a la hoja de cálculo en la que deseas trabajar:

hoja = archivo\_excel['Nombre de la Hoja']

1. Realiza operaciones en la hoja de cálculo. Puedes leer y escribir datos, dar formato, copiar, eliminar, etc. Aquí hay algunos ejemplos:
   * Leer datos de una celda:

valor = hoja['A1'].value

* + Escribir datos en una celda:

hoja['B2'] = "Nuevo Valor"

* + Iterar a través de filas y columnas:

for fila in hoja.iter\_rows(min\_row=2, max\_row=hoja.max\_row, min\_col=1, max\_col=hoja.max\_column): for celda in fila: print(celda.value)

1. Guarda los cambios en el archivo Excel:

archivo\_excel.save('nombre\_del\_archivo.xlsx')

1. Cierra el archivo Excel:

archivo\_excel.close()

1. Automatiza tareas específicas según tus necesidades. Puedes combinar estas operaciones para realizar tareas más complejas, como filtrar datos, calcular estadísticas, crear gráficos, etc.

Este es solo un ejemplo básico de cómo automatizar tareas en Excel con Python. La biblioteca **openpyxl** es muy versátil y te permite realizar una amplia gama de tareas. Sin embargo, también hay otras bibliotecas como **pandas** y **xlwings** que ofrecen funcionalidades adicionales para trabajar con datos de Excel de manera eficiente. La elección de la biblioteca depende de tus necesidades específicas.

**Automatización de reportes de Excel con Python**

**Pasos para automatizar archivos Excel utilizando pandas y openpyxl**

En resumen, utilizamos **pandas para la lectura y manejo de archivos Excel, creación de tablas pivote y exportación de las mismas a archivos Excel**.

En cambio usamos **openpyxl para hacer reportes en Excel**.

**Importamos las librerías pandas y openpyxl**

import pandas as pd

import openpyxl

**Leer archivos Excel (pandas)**

#leemos el archivo excel (o df, lo que prefiramos) y lo asignamos a una variable

archivo\_excel = pd.read\_excel('supermarket\_sales.xlsx')

**Crear tablas pivote con la función pivot\_table (pandas)**

Este término se utiliza comúnmente en el contexto de análisis de datos para describir una **tabla que se ha reorganizado para facilitar el resumen, la agregación o el análisis de datos de una manera más conveniente**.

La función **pivot\_table** permite reorganizar un DataFrame para resumir datos en función de los valores de una o más columnas. Esto facilita el análisis y la visualización de datos de una manera más estructurada y conveniente. Los argumentos clave, como **values**, **index**, **columns** y **aggfunc**, se utilizan para controlar cómo se realiza la agregación y la organización de los datos en la tabla pivote.

**df:** Esto es el DataFrame de pandas en el que se encuentra el conjunto de datos que deseas pivotar.

1. **values**: Esta es la columna que se utilizará para calcular los valores de la tabla pivote.
2. **index**: Esta es la columna que se utilizará como el índice (filas) de la tabla pivote. Cada valor único en esta columna se convierte en una fila en la tabla pivote.
3. **columns**: Esta es la columna que se utilizará para crear las columnas de la tabla pivote. Cada valor único en esta columna se convierte en una columna en la tabla pivote.
4. **aggfunc**: Esta es la función de agregación que se aplicará a los valores de las celdas cuando hay duplicados en la intersección de una fila y una columna en la tabla pivote.

**Algunas de las funciones comunes del parámetro aggfunc para los conjuntos de datos son:**

**'sum'** suma los valores.

**'mean'** calcula el promedio.

**'max'** encuentra el valor máximo.

**'min'** encuentra el valor mínimo.

**'count'** Cuenta el número de valores no nulos.

**'size'** Similar a count, pero incluye valores nulos en el conteo. Cuenta el número total de valores (nulos y no nulos).

**'first'** Devuelve el primer valor no nulo en la columna. Es útil cuando deseas conservar el primer valor en caso de duplicados.

**'last'** Devuelve el último valor no nulo en la columna. Se utiliza para conservar el último valor en caso de duplicados.

#creamos una tabla pivote en la que ordenaremos el total gastado en cada linea de productos separados por género

tabla\_pivote = archivo\_excel.pivot\_table(index='Gender', columns='Product line', values='Total', aggfunc='sum').round(0)

#round(0) es para redondear los decimales

print(tabla\_pivote)

**Exportar tablas pivote a archivos Excel (pandas)**

#exportamos la tabla pivote como archivo Excel y le ponemos el nombre que tendrá el archivo Excel (con su terminación .xlsx)

tabla\_pivote.to\_excel('total\_por\_género.xlsx', startrow=0, sheet\_name='Reporte por género')

#startrow es el parámetro que controla en que fila (row) del Excel comenzará a mostrarse la tabla\_pivote

#sheet\_name es el nombre que le pondremos a la pestaña de excel

#Tanto startrow como sheet\_name son parámetros opcionales

En este ejemplo el archivo Excel resultante es:

**Con esto terminamos con la librería pandas, ahora toca trabajar con openpyxl en la creación de reportes en Excel**.

**Hacer reportes en Excel utilizando openpyxl**

**Crear filas y columnas de referencia (openpyxl)**

Para leer un archivo Excel con openpyxl usaremos load\_wordbook.

from openpyxl import load\_workbook

Cargamos el excel de la tabla pivote e ingresamos la pestaña con la que trabajaremos (sheet\_name)

#leemos el Excel de la tabla pivote con load\_workbook (en general le daremos el valor de lectura a una variable wb(WorkBook))

wb = load\_workbook('total\_por\_género.xlsx')

#usamos una variable pestaña y ponemos el nombre de la pestaña o ventana que usaremos del archivo que estamos leyendo (el que pusimos de sheet\_name al exportar la tabla pivote a Excel)

pestaña = wb['Reporte por género']

Para automatizar cualquier excel, debemos hacer que el programa detecte la cantidad de filas y columnas activas del mismo, y eso permite que funcione de igual forma sin importar que luego el archivo cambie su tamaño, ya que lo detectará automáticamente.

#ahora debemos saber que filas y columnas están activas en el excel que estamos utilizando para tener las referencias

#para eso debemos conocer las filas máximas y mínimas y las columnas máximas y mínimas que se están utilizando

min\_col = wb.active.min\_column

max\_col = wb.active.max\_column

min\_fila = wb.active.min\_row

max\_fila = wb.active.max\_row

**Crear gráficos de Excel mediante Python (openpyxl)**

Debemos importar: from openpyxl.chart import BarChart, Reference

Definimos la data y las categorías del **wb** que usaremos para el gráfico (especificando la pestaña y los valores mínimos y máximos de las filas y columnas, modificándolas dependiendo de si lo hacemos con la data o categorías)

*#queremos que la data tome también los headers o encabezados de la tabla\_pivote de excel, por ende le agregamos 1 a la min\_columna calculada previamente*

data = Reference(pestaña, min\_col=min\_columna+1, max\_col=max\_columna, min\_row=min\_fila, max\_row=max\_fila)

*#queremos que las categorías tomen también los valores de las categorías de la tabla\_pivote de excel, (por ende le agregamos 1 a la min\_fila calculada previamente y también hacemos que la min\_col y la max\_col tengan el mismo valor de min\_columna, ya que solo queremos que tome la columna de las categorías, que es la primera)*

categorias = Reference(pestaña, min\_col=min\_columna, max\_col=min\_columna, min\_row=min\_fila+1, max\_row=max\_fila)

Creamos el gráfico

*#creación del gráfico*

barchart = BarChart()

Cargamos la data y las categorías dentro del gráfico y le ponemos un título y un estilo al mismo

*#agregamos la data (el parámetro titles\_from\_data = True sirve para indicarle al programa que también debe tomar los headers o encabezados)*

barchart.add\_data(data, titles\_from\_data=True)

*#agregamos las categorías*

barchart.set\_categories(categorias)

*#podemos agregarle un título y un estilo al gráfico*

barchart.title = 'Total de ventas por género'

barchart.style = 2

Agregamos el gráfico a la pestaña de excel con la que estamos trabajando en la posición que deseemos

*#agregamos el gráfico barchart a la pestaña de excel con la que estamos trabajando, en este caso la posición K1 (el parámetro sirve para indicar en qué celda de la pestaña pondremos el gráfico)*

pestaña.add\_chart(barchart, 'K1')

Por último, guardamos el archivo (esto SIEMPRE VA ÚLTIMO DE TODO, si agregamos mas funcionalidades al código debemos ponerlas antes del guardado del wb)

*#por último, guardamos el archivo wb (el archivo excel creado a partir de la tabla\_pivote)*

wb.save('total\_por\_género.xlsx')

**Aplicar fórmulas de Excel mediante Python (openpyxl)**

**Podemos realizar las mismas acciones y cálculos que hacemos en excel en Python para poder automatizar los reportes**.

**Estructura de una fórmula común en Excel**

**Pestaña[‘ColumnaFila’] = ‘=Formula(ColumnaFila\_inferior : ColumnaFila\_superior)’**

**Si queremos realizar una suma de columnas o filas y mostrarla en una posición específica del wb ponemos:**

*#en la posición B4 se mostrará la suma de la posición B2 hasta la B3*

pestaña['B4'] = '=SUM(B2:B3)'

pestaña['B4'].style = 'Currency' *#este estilo hace que se muestre en formato moneda*

**En este caso realizaremos la suma de los totales de cada columna del wb**.

Importamos string

*#importamos string para utilizar string.ascii\_uppercase*

import string

Debemos conocer cuáles son las columnas que debemos usar para nuestro objetivo

*#en este caso sumaremos los totales de cada columna de la pestaña de excel que venimos usando, para eso usaremos un bucle for*

*#pero antes del for debemos conocer cuáles son las columnas que usaremos, y según vemos en el archivo excel del wb debemos usar de la columna B hasta la G, por eso debemos realizar dos pasos antes del for*

*#primero usaremos una variable abecedario para que contenga todos las letras en MAYÚSCULAS (ya que los índices de Excel son en mayúsculas)*

abecedario = list(string.ascii\_uppercase)

*#luego nos quedaremos solo con las letras que necesitamos (para eso usamos el slicing o segmentación)*

*#en este caso solo debemos usar de la A hasta la G (max\_columna)*

abecedario\_excel = abecedario[0:max\_columna]

Sumamos los totales de cada columna con un For

*#utilizamos un bucle for para sumar los totales de cada columna*

for i in abecedario\_excel :

*# i debe ser distinto de 'A' ya que ahí se encuentran los géneros, y no queremos sumar eso*

    if i != 'A' :

*#---sumamos cada total de las columnas especificadas y ponemos el resultado en la posición correspondiente---*

*#al hacer que se guarde en la columna i y en la fila max\_fila+1 nos aseguramos que el programa seguirá funcionando correctamente por más que el excel sea modificado*

*#al hacer que se realice la suma entre la columna i y la fila min\_fila+1 y la columna i y la fila max\_fila nos aseguramos que el programa seguirá realizando las sumas correctamente por más que el excel sea modificado*

        pestaña[f'{i}{max\_fila+1}'] = f'=SUM({i}{min\_fila+1}:{i}{max\_fila})'

        pestaña[f'{i}{max\_fila+1}'].style = 'Currency' *#este estilo hace que se muestre en formato moneda*

Agregamos el nombre ‘Total’ en la fila donde se muestran los resultados de las sumas de cada columna

*#agregamos el nombre 'Total' (para mostrar que en esa fila se mostrarán los totales de todas las sumas anteriores) en la primera columna de la fila max\_fila+1 para asegurarnos el correcto funcionamiento sin importar si el excel es modificado*

pestaña[f'{abecedario\_excel[0]}{max\_fila+1}'] = 'Total'

**Con esto, GENERAMOS UNA AUTOMATIZACIÓN QUE SEGUIRÁ FUNCIONANDO CORRECTAMENTE SIN IMPORTAR SI EL WB ES MODIFICADO.**

**Dar formato a los reportes de Excel (openpyxl)**

Por último le damos un poco de formato al excel al poner un par de títulos o textos descriptivos.

Importamos Font

*#importamos Font para darle formato final al excel del wb*

from openpyxl.styles import Font

Ponemos títulos o textos descriptivos y sus fuentes

*#Por último le damos un poco de formato y estilo al excel al poner un par de títulos o textos descriptivos.*

*#-----textos-----*

pestaña['A1'] = 'Reporte automatizado'

pestaña['A2'] = '2023'

*#-----fuentes de los textos-----*

*#de parámetros ponemos el nombre de la fuente, si será negrita o no, y el tamaño*

pestaña['A1'].font = Font('Arial', bold=True, size=20)

pestaña['A2'].font = Font('Arial', bold=False, size=12)

**Automatizar los reportes con una función en Python**

Creamos otro script de Python y pegamos todo el código dentro de una función de Python.

Además lo modificamos para que pueda automatizar todos los archivos Excel que tengan el nombre “Ventas\_mes.xlsx”

"""ESTE PROGRAMA AUTOMATIZA LOS REPORTES DE LAS VENTAS DE CADA MES. ES OBLIGATORIO QUE EL ARCHIVO DE INGRESO (INPUT) TENGA EL FORMATO Ventas\_mes.xlsx PARA EL CORRECTO FUNCIONAMIENTO DEL PROGRAMA.

EL MISMO DEVUELVE COMO SALIDA (OUTPUT) UN ARCHIVO CON EL FORMATO Reporte\_mes.xlsx CON SU RESPECTIVO GRÁFICO"""

import pandas as pd *#usamos pandas para la lectura y manejo de archivos Excel, creación de tablas pivote y exportación de las mismas a archivos Excel.*

from openpyxl import load\_workbook *#usamos openpyxl para hacer reportes en Excel (usamos load\_workbook para leer los archivos Excel con openpyxl)*

from openpyxl.chart import BarChart, Reference

*#importamos string para utilizar string.ascii\_uppercase*

import string

*#importamos Font para darle formato final al excel del wb*

from openpyxl.styles import Font

*#creamos la función que nos permitirá automatizar los reportes del Excel*

def automatizar\_excel(nombre\_archivo) :

    """Input Ventas\_mes.xlsx / Output reporte\_mes.xlsx"""

*#leemos el archivo excel (o df, lo que prefiramos) y lo asignamos a una variable*

    archivo\_excel = pd.read\_excel(nombre\_archivo) *#modificamos el nombre original por esta variable para que se pueda usar ante cualquier archivo y funcione la automatización*

*#creamos una tabla pivote en la que ordenaremos el total gastado en cada linea de productos separados por género*

    tabla\_pivote = archivo\_excel.pivot\_table(index='Gender', columns='Product line', values='Total', aggfunc='sum').round(0)

*#round(0) es para redondear los decimales*

*#tomamos como variable el mes y la extensión del archivo para poder hacer el reporte de cada mes que ingrese*

*#tomará lo que está luego del \_*

    mes\_extension = nombre\_archivo.split('\_')[1]

*#exportamos la tabla pivote como archivo Excel y le ponemos el nombre que tendrá el archivo Excel (con su terminación .xlsx) y su mes\_extension*

    tabla\_pivote.to\_excel(f'Reporte\_{mes\_extension}', startrow=3, sheet\_name='Reporte por género')

*#startrow es el parámetro que controla en que fila (row) del Excel comenzará a mostrarse la tabla\_pivote*

*#sheet\_name es el nombre que le pondremos a la pestaña o ventana de excel*

*#------------------------------Creación de reportes de Excel con openpyxl------------------------------*

*#leemos el Excel de la tabla pivote con load\_workbook (en general le daremos el valor de lectura a una variable wb(WorkBook))*

    wb = load\_workbook(f'Reporte\_{mes\_extension}')

*#usamos una variable pestaña y ponemos el nombre de la pestaña o ventana que usaremos del archivo que estamos leyendo (el que pusimos de sheet\_name al exportar la tabla pivote a Excel)*

    pestaña = wb['Reporte por género']

*#ahora debemos saber que filas y columnas están activas en el excel que estamos utilizando para tener las referencias*

*#para eso debemos conocer las filas maximas y minimas y las columnas maximas y minimas que se estan utilizando*

    min\_columna = wb.active.min\_column

    max\_columna = wb.active.max\_column

    min\_fila = wb.active.min\_row

    max\_fila = wb.active.max\_row

*#CON ESTO PODEMOS AUTOMATIZAR CUALQUIER REPORTE, YA QUE AUTOMATICAMENTE EL PROGRAMA TOMARÁ LOS VALORES DE LAS FILAS Y COLUMNAS ACTIVAS, SIN IMPORTAR SI CAMBIAN LUEGO DE TAMAÑO*

*#especificamos que la data y categorias tomarán los valores de la pestaña con la que se está trabajando, y los valores mínimos y máximos de las filas y columnas que calculamos antes.*

*#queremos que la data tome también los headers o encabezados de la tabla\_pivote de excel, por ende le agregamos 1 a la min\_columna calculada previamente*

    data = Reference(pestaña, min\_col=min\_columna+1, max\_col=max\_columna, min\_row=min\_fila, max\_row=max\_fila)

*#queremos que las categorias tomen también los valores de las categorias de la tabla\_pivote de excel, (por ende le agregamos 1 a la min\_fila calculada previamente y también hacemos que la min\_col y la max\_col tengan el mismo valor de min\_columna, ya que solo queremos que tome la columna de las categorias, que es la primera)*

    categorias = Reference(pestaña, min\_col=min\_columna, max\_col=min\_columna, min\_row=min\_fila+1, max\_row=max\_fila)

*#-----ahora solo nos falta crear el gráfico y cargar las referencias dentro del mismo-----*

*#creación del gráfico*

    barchart = BarChart()

*#agregamos la data (el parámetro titles\_from\_data = True sirve para indicarle al programa que también debe tomar los headers o encabezados)*

    barchart.add\_data(data, titles\_from\_data=True)

*#agregamos las categorias*

    barchart.set\_categories(categorias)

*#podemos agregarle un título y un estilo al gráfico*

    barchart.title = 'Total de ventas por género'

    barchart.style = 2

*#agregamos el gráfico barchart a la pestaña de excel con la que estamos trabajando, en este caso la posición C10 (el parámetro sirve para indicar en qué celda de la pestaña pondremos el gráfico)*

    pestaña.add\_chart(barchart, 'C10')

*#----------aplicamos fórmulas de Excel para automatizar los reportes----------*

*#si quisiéramos sumar columnas o filas y poner el resultado en una posición específica debemos poner:*

*#en la posición B4 se mostrará la suma de la posición B5 hasta la B6*

*#pestaña['B7'] = '=SUM(B5:B6)'*

*#pestaña['B7'].style = 'Currency' #este estilo hace que se muestre en formato moneda*

*#en este caso sumaremos los totales de cada columna de la pestaña de excel que venimos usando, para eso usaremos un bucle for*

*#pero antes del for debemos conocer cuáles son las columnas que usaremos, y según vemos en el archivo excel del wb debemos usar de la columna B hasta la G, por eso debemos realizar dos pasos antes del for*

*#primero usaremos una variable abecedario para que contenga todos las letras en MAYÚSCULAS (ya que los índices de Excel son en mayúsculas)*

    abecedario = list(string.ascii\_uppercase)

*#luego nos quedaremos solo con las letras que necesitamos (para eso usamos el slicing o segmentación)*

*#en este caso solo debemos usar de la A hasta la G (max\_columna)*

    abecedario\_excel = abecedario[0:max\_columna]

*#utilizamos un bucle for para sumar los totales de cada columna*

    for i in abecedario\_excel :

*# i debe ser distinto de 'A' ya que ahí se encuentran los géneros, y no queremos sumar eso*

        if i != 'A' :

*#---sumamos cada total de las columnas especificadas y ponemos el resultado en la posición correspondiente---*

*#al hacer que se guarde en la columna i y en la fila max\_fila+1 nos aseguramos que el programa seguirá funcionando correctamente por más que el excel sea modificado*

*#al hacer que se realice la suma entre la columna i y la fila min\_fila+1 y la columna i y la fila max\_fila nos aseguramos que el programa seguirá realizando las sumas correctamente por más que el excel sea modificado*

            pestaña[f'{i}{max\_fila+1}'] = f'=SUM({i}{min\_fila+1}:{i}{max\_fila})'

            pestaña[f'{i}{max\_fila+1}'].style = 'Currency' *#este estilo hace que se muestre en formato moneda*

*#agregamos el nombre 'Total' (para mostrar que en esa fila se mostrarán los totales de todas las sumas anteriores) en la primera columna de la fila max\_fila+1 para asegurarnos el correcto funcionamiento sin importar si el excel es modificado*

    pestaña[f'{abecedario\_excel[0]}{max\_fila+1}'] = 'Total'

*#con esto, GENERAMOS UNA AUTOMATIZACIÓN QUE NO AFECTARÁ SU RENDIMIENTO SI EL EXCEL DEL WB ES MODIFICADO*

*#de esta forma, en nuestro wb veremos que la última columna se llamará Total y en esa fila estarán todos los totales de las sumas*

*#Por último le damos un poco de formato y estilo al excel al poner un par de títulos o textos descriptivos.*

*#-----textos-----*

    pestaña['A1'] = 'Reporte automatizado'

*#conseguimos el mes para usarlo (tomamos lo que está antes del punto en mes\_extension)*

    mes = mes\_extension.split('.')[0]

*#hacemos que el mes aparezca en la posicion A2*

    pestaña['A2'] = mes

*#-----fuentes de los textos-----*

*#de parámetros ponemos el nombre de la fuente, si será negrita o no, y el tamaño*

    pestaña['A1'].font = Font('Arial', bold=True, size=20)

    pestaña['A2'].font = Font('Arial', bold=False, size=12)

*#por último, guardamos el archivo wb (el archivo excel creado a partir de la tabla\_pivote)*

    wb.save(f'Reporte\_{mes\_extension}')

*#probamos la función con cualquier archivo excel que tenga el formato pedido (en este caso el formato de entrada debe ser Ventas\_mes.xlsx)*

*#automatizar reportes mensuales*

automatizar\_excel('Ventas\_enero.xlsx')

automatizar\_excel('Ventas\_febrero.xlsx')

De esta forma, este programa toma los archivos Excel llamados “Ventas\_mes.xlsx” y los transforma en tablas pivote de Excel llamadas “Reporte\_mes.xlsx” junto con su respectivo gráfico.

Es importante aclarar que si debemos leer otro formato de excel que no contenga las columnas utilizadas para formar la tabla pivote indicada, debemos modificar esa parte del código para tomar las columnas existentes.