Manual Librería Tablas

	Responsable	Cargo/Rol	Fecha
Preparado	Capeletti, Gian Franco	Developer	18/05/2021
Revisado	Ortiz, Rodrigo Nicolas	Lider técnico	18/05/2021
Aprobado			

CONTROL DE EDICIONES

VERSION	FECHA	MODIFICACIÓN	AUTOR
0.1	18/05/2021	Creación del documento	Capeletti, Gian Franco
0.2	09/06/2021	Modificación del documento	Argolo, Luciano Agustín
0.3			
0.4			
0.5			

Contenido

1. Tablas.py

1.1 Descripción

La librería **Tablas** está diseñada para estructurar y manejar tablas de datos de forma rápida y flexible. Integrada por un sistema de columnas y filas, otorga diversas funciones para manejar las tablas (Denominadas **DataFrame**) y los datos que las componen, diseñado completamente en Python.

1.2 Detalles y características

- Creación de objetos **DataFrame** para la manipulación de datos;
- **Indexado** de filas para un mejor manejo;
- **Escritura y lectura de datos** a partir de archivos de formato CSV;
- **Reestructuración** de las tablas;
- **Transformación** de datos, tal como comparación, modificación, operaciones matemáticas/lógicas o eliminación;
- Capacidad de **agrupamiento** por campos;
- **Generación de subtablas** a través de condiciones lógicas;
- **Ordenamiento de datos** según uno o varios campos;
- **Exportación de tablas** a archivos de formato CSV.

1.3 Objetivo

Con el uso de esta librería, se busca otorgar la capacidad de manejar diferentes orígenes de información y realizar análisis de datos de forma práctica con una herramienta flexible acoplada en Python puro.

2. Estructura

2.1 DataFrame – Clase

tablas.DataFrame(df, filas, columnas)

Permite crear instancias de tablas de dos dimensiones para el almacenamiento, manejo y proceso de datos.

Posee un formato de columnas y filas etiquetadas, permite manejar diferentes tipos de datos para ejecutar los procesos que sean necesarios.

Esta clase suele iniciarse a través del uso de funciones de lectura de archivos (Tal como **leer_csv**)

2.1.1 Parámetros

- **df : diccionario**
  Refiere a la estructura de tipo diccionario que contendrá los datos.
- **filas: numérico – int**
  Cantidad de registros que existen actualmente en el **DataFrame**.
- **columnas : diccionario o listado de strings – list**
  Listado que contiene el nombre de las columnas del **DataFrame**.

2.1.2 Métodos

acumular(columna, nuevaCol)	Genera una nueva columna con el nombre indicado, a partir de la suma acumulada de los valores de la columna definida.
agregarCol(nuevaCol, rellenar)	Genera una nueva columna con el nombre indicado
agregarFila(otro, índices)	Agrega al DataFrame actual la/las filas indicadas en parámetro, completando las columnas no definidas.
agrupar(columna)	Agrupa los registros según los valores de la/las columnas indicadas y crea nuevas subtablas.
anexar(otro, índices)	Permite agregar al DataFrame registros individuales u otros DataFrames.
buscar(condiciones)	Busca aquellos registros que cumplan con las condiciones indicadas y los devuelve en una subtabla.
cambiarColumnas(lista)	Cambia el nombre de las columnas del DataFrame.
cambiarValor(columna, valor, inplace, condiciones)	Cambia los valores de una o varias columnas a un valor fijo o modifica el valor de cada celda a través de una operación.
cambiarTipo(diccionarioCambio)	Modifica el tipo de datos de las columnas indicadas
contiene(columna, valor)	Comprueba cuales registros del DataFrame contienen la subcadena indicada.
copiar(deep)	Realiza una copia del DataFrame que puede o no ser anexada.
eliminar(indices, columnas, inplace)	Elimina las columnas y filas indicada.
exportarCSV(ruta, separador, columnas, índice)	Crea un archivo de texto de formato CSV y exporta todos los registros del DataFrame.
final(cantidad, columnasSelect)	Devuelve los últimos N registros del DataFrame.
indices()	Devuelve el número de índice de cada registro.
iterar(*args)	Recorre cada registro del DataFrame y devuelve uno por uno de forma iterativa.
ordenar(columnas, ascendente, indices)	Ordena el DataFrame datos según la/las columnas indicadas de forma ascendente o descendente.
principio(cantidad, columnasSelect)	Devuelve los primeros N registros de la tabla.

redondear(kwargs)	Redondea los registros numéricos de la/las columnas indicadas.
reiniciarIndices(inplace)	Reinicia los indices de los registros de la tabla.
remplazar(columna, cadena, remplazo)	Remplaza la subcadena de todos los registros de la columna indicada por otra cadena de texto,
separar(columna, valor, posición)	Separa la cadena de texto de cada registro de la columna indicada según el separador indicado.
vacio()	Comprueba si el DataFrame posee registros.
valores(*args)	Devuelve la lista de valores de cada columna.
colMax(columnmax)	Devuelve el valor máximo de una columna
colMin(columnmin)	Devuelve el valor mínimo de una columna
Acum(columacum)	Devuelve el valor acumulado de una columna
unique(columunique)	Devuelve una lista de los valores únicos de una columna
nunique(columunique)	Devuelve un valor equivalente a ala longitud de la lista devuelta por unique()
innerJoin(dfderecha,columnmatch, columnas_izq, columnas_der)	Devuelve un DataFrame definido por los parámetros resultado de dos DataFrame's unidos por las columnas donde hubiera datos en común.

2.2 Operadores – Clase

Clase diseñada para permitir la ejecución de operaciones lógicas/matemáticas entre los valores de los DataFrame.

2.1.1 Atributos – variables

Las variables de clase definidas hacen referencia a los operadores lógicos/matemáticos que se utilizaran. Estos son:

- **igualdad: ==**
- **desigualdad: !=**
- **menor: <**
- **menorIgual: <=**
- **mayor: >**
- **mayorIgual: >=**
- **suma: +**
- **resta: -**
- **multiplicación: ***
- **división: /**

2.1.2 Métodos

funcIgual(*args)	Ejecuta la operación Igualdad entre dos valores.
funcDistinto(*args)	Ejecuta la operación Desigualdad entre dos valores.
funcMenor(*args)	Ejecuta la operación Menor entre dos valores.

funcMenorIgual(*args)	Ejecuta la operación Menor o igual entre dos valores.
funcMayor(*args)	Ejecuta la operación Mayor entre dos valores.
funcMayorIgual(*args)	Ejecuta la operación Mayor o igual entre dos valores.
funcSuma(*args)	Ejecuta la operación Suma entre dos valores.
funcResta(*args)	Ejecuta la operación Resta entre dos valores.

funcMultiplicacion(*args)	Ejecuta la operación Multiplicación entre dos valores.
funcDivision(*args)	Ejecuta la operación División entre dos valores.

3. Funciones

3.1 crearDataFrame(guía = None)

Objetivo

- Genera una instancia de un objeto DataFrame vacío para su posterior uso. Puede brindarse por parámetro un diccionario que funcionara como guía para crear columnas predefinidas.

Parámetros

- **guía : dict – *Default: None***
  Diccionario cuyas claves se usarán para definir las claves del nuevo objeto DataFrame.

Salida

- Genera una nueva instancia de objeto DataFrame.

Proceso

- Recibe de forma opcional un diccionario por parámetro.
- Establece los atributos base de cualquier DataFrame.
- Si se brindó un diccionario por parámetro, obtiene las claves del mismo.
- Crea el nuevo objeto DataFrame y lo retorna.

Ejemplo

Creando un DataFrame completamente vacio.

```
>>> df = tablas.crearDataFrame()
>>> df
Indice
```

Creando un DataFrame a partir de un diccionario.

```
>>> df = tablas.crearDataFrame({“ID”: [], “StartTime”: [], “Outweight”: []})
>>> df
Indice     ID    StartTime     Outweight
```

3.2 definirColumnas (archivo, separador)

Objetivo

- Define un diccionario con la misma cantidad de claves como campos de valores que posea el archivo de formato CSV que se le defina. Este método se usa en conjunto con ***leer_csv*** *para crear un objeto de tipo DataFrame a partir de un archivo CSV.*

Parámetros

- **archivo : File object type**
  Instancia de un objeto File que referencia al archivo de se desea usar.
- **separador : string**
  Carácter que se utiliza en el archivo CSV para separar los campos.

Salida

- Retorna un diccionario con la misma cantidad de claves como de campos en el CSV.

Proceso

- Recibe por parámetro la ruta del archivo y el separador utilizado en el archivo CSV.
- Procede a leer la primera línea del archivo.
- Define el número de columnas como la cantidad de veces que se encuentra el separador en la primera línea aumentado en 1.
- Crea el listado de columnas con nombres por defecto y lo retorna.

Ejemplo

Se utiliza un archivo que posee tres campos separados por coma.

```
>>> columnas = tablas.definirColumnas("archivo.csv", ",")
>>> columnas
{"col1": [], "col2": [], "col3": []}
```

3.3 ejecutarOperacion(valor, operacion)

Objetivo

- Permite realizar operaciones lógicas/matemáticas utilizando los valores del DataFrame.

Parámetros

- **valor: int/float/string**
  Valor sobre el que se realizara la operación necesaria. Puede tratarse de un valor numérico o de texto.
- **operación: list**
  Lista de dos valores: El primero se trata del operador que referencia a la operación que se realizara; El segundo será el valor adicional que se aplicara en la operación. Puede ser un valor numérico o de texto.

Salida

- Si se trata de una operación lógica, retorna un valor booleano de Verdadero/Falso; Si se trata de una operación matemática, devuelve el resultado de dicha operación.

Proceso

- Recibe por parámetro el primer valor de operación, y el listado que contiene el operador y el segundo valor de operación.
- Realiza un llamado a un diccionario de operaciones:
  - Las claves de este diccionario son las diferentes operaciones lógicas/matemáticas (Igualdad: '=='; Desigualdad: '!='; Menor: '<'; MenorIgual: '<='; Mayor: '>'; MayorIgual: '>='; Suma: '+'; Resta: '-'; Multiplicación: '*'; División: '/')
  - Cada una de estas claves ejecuta su operación correspondiente, tomando por parámetros los dos valores con los que se quiere operar y así devolver el resultado de la operación.

Ejemplo

Ejecución de una operación matemática.

```
>>> resultado = tablas.ejecutarOperacion(15, ["*", 3])
>>> resultado
45
```

Ejecución de una operación lógica.

```
>>> resultado = tablas.ejecutarOperacion(22.7, ["<", 26.1])
>>> resultado
True
```

Ejecución de una operación de concatenación.

```
>>> resultado = tablas.ejecutarOperacion("Hola", ["+", " Mundo"])
>>> resultado
Hola Mundo
```

3.4 leer_csv(archivo, separador)

Objetivo

- Crea una instancia de un DataFrame a través de un archivo de formato CSV, con la misma cantidad de columnas que de campos en el archivo y agregando cada uno de los registros que tenga.

Parámetros

- **archivo : string**
  Ruta del archivo que se utilizara.
- **separador : string**
  Carácter que se utiliza en el archivo CSV para separar los campos.

Salida

- Retorna una instancia de un objeto DataFrame.

Proceso

- Recibe por parámetro la ruta del archivo CSV y el separador que utiliza.
- Define los parámetros base del objeto DataFrame.
- Actualiza la cantidad de columnas que posee el archivo CSV.
- Por cada línea que lee en el archivo:
  - Registra el índice del nuevo registro
  - De forma iterativa, escribe cada valor del registro en su columna correspondiente secuencialmente.
  - Actualiza la cantidad de registros que posee el DataFrame
- Crea la instancia del nuevo objeto DataFrame y la retorna.

Ejemplo

Se lee un nuevo archivo CSV.

```
>>> df = tablas.leer_csv("archivoCSV.txt", ";")
>>> df
Indice    Col1    Col2        Col3
0         1       20201208    15.0
1         2       20201209    21.4
2         3       20201209    12.7
```

4. Métodos

4.1 Operadores

4.1.1 funcIgual(*args)

Objetivo

- Se encarga de comparar si los valores ingresados a la función son iguales, devolviendo un valor Booleano, True o False, dependiendo de si son iguales o no, respectivamente.

Parámetros

- ***args : int/float/string**
  Los argumentos pasados como parámetro generan una lista, formada por int, floats o strings. Son los valores a comparar.

Salida

- Valor de verdad.

Proceso

- Recibe por parámetro una lista de valores.
- Compara los valores y verifica si son o no iguales, siendo True si son iguales, y False cuando no.
- Devuelve el resultado booleano de la comparación.

Ejemplo

Se ingresan dos valores distintos a la función.

```
>>> valor = Operadores.funcIgual(3, 4)
>>> valor
False
```

Se ingresan dos valores iguales a la función.

```
>>> valor = Operadores.funcIgual(3, 3)
>>> valor
True
```

4.1.2 funcDistinto(*args)

Objetivo

- Se encarga de comparar si los valores ingresados a la función son distintos, devolviendo un valor Booleano, True o False, dependiendo de si son distintos o si no lo son, respectivamente.

Parámetros

- ***args : int/float/string**
  Los argumentos pasados como parámetro generan una lista, formada por int, floats o strings. Son los valores a comparar.

Salida

- Valor de verdad.

Proceso

- Recibe por parámetro una lista de valores.
- Compara los valores y verifica si son o no distintos, siendo True si no lo son, y False cuando son iguales.
- Devuelve el resultado booleano de la comparación.

Ejemplo

Se ingresan dos valores distintos a la función.

```
>>> valor = Operadores.funcDistinto(3, 4)
>>> valor
True
```

Se ingresan dos valores iguales a la función.

```
>>> valor = Operadores.funcDistinto(3, 3)
>>> valor
False
```

4.1.3 funcMenor(*args)

Objetivo

- Se encarga de comparar si el primer valor es menor al segundo valor ingresado, devolviendo un valor Booleano, True o False, dependiendo de si es menor o no lo es, respectivamente.

Parámetros

- ***args : int/float/string**
  Los argumentos pasados como parámetro generan una lista, formada por int, floats o strings. Son los valores a comparar.

Salida

- Valor de verdad.

Proceso

- Recibe por parámetro una lista de valores.
- Compara los valores y verifica si el primero es menor al segundo, siendo True si lo es, y False cuando no lo es.
- Devuelve el resultado booleano de la comparación.

Ejemplo

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcMenor(3, 4)
>>> valor
True
```

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcMenor(3, 3)
>>> valor
False
```

4.1.4 funcMenorIgual(*args)

Objetivo

- Se encarga de comparar si el primer valor es menor o igual al segundo valor ingresado, devolviendo un valor Booleano, True o False, dependiendo de si es menor/igual o no lo es, respectivamente.

Parámetros

- ***args : int/float/string**
  Los argumentos pasados como parámetro generan una lista, formada por int, floats o strings. Son los valores a comparar.

Salida

- Valor de verdad.

Proceso

- Recibe por parámetro una lista de valores.
- Compara los valores y verifica si el primero es menor o igual al segundo, siendo True si lo es, y False cuando no lo es.
- Devuelve el resultado booleano de la comparación.

Ejemplo

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcMenorIgual(3, 2)
>>> valor
False
```

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcMenorIgual(3, 3)
>>> valor
True
```

4.1.5 funcMayor(*args)

Objetivo

- Se encarga de comparar si el primer valor es mayor al segundo valor ingresado, devolviendo un valor Booleano, True o False, dependiendo de si es mayor o no lo es, respectivamente.

Parámetros

- ***args : int/float/string**
  Los argumentos pasados como parámetro generan una lista, formada por int, floats o strings. Son los valores a comparar.

Salida

- Valor de verdad.

Proceso

- Recibe por parámetro una lista de valores.
- Compara los valores y verifica si el primero es mayor al segundo, siendo True si lo es, y False cuando no lo es.
- Devuelve el resultado booleano de la comparación.

Ejemplo

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcMayor(3, 4)
>>> valor
False
```

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcMayor(3, 1)
>>> valor
True
```

4.1.6 funcMayorIgual(*args)

Objetivo

- Se encarga de comparar si el primer valor es mayor o igual al segundo valor ingresado, devolviendo un valor Booleano, True o False, dependiendo de si es mayor/igual o no lo es, respectivamente.

Parámetros

- ***args : int/float/string**
  Los argumentos pasados como parámetro generan una lista, formada por int, floats o strings. Son los valores a comparar.

Salida

- Valor de verdad.

Proceso

- Recibe por parámetro una lista de valores.
- Compara los valores y verifica si el primero es mayor o igual al segundo, siendo True si lo es, y False cuando no lo es.

- Devuelve el resultado booleano de la comparación.

Ejemplo

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcMayorIgual(3, 4)
>>> valor
False
```

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcMayorIgual(3, 3)
>>> valor
True
```

4.1.7 funcSuma(*args)

Objetivo

- Se encarga de sumarle al primer valor el segundo valor ingresado, devolviendo el resultado de la operación.

Parámetros

- ***args : int/float/string**
  Los argumentos pasados como parámetro generan una lista, formada por int, floats o strings. Son los valores de la operacion.

Salida

- Valor de la operación matemática/concatenación.

Proceso

- Recibe por parámetro una lista de valores.
- Suma el primer valor con el segundo si son numéricos, o los concatena si alguno de ellos es un string.
- Devuelve el resultado de la operación.

Ejemplo

Se ingresan dos valores numéricos a la función.

```
>>> valor = Operadores.funcSuma(3, 4)
>>> valor
7
```

Se ingresan dos valores de texto a la función.

```
>>> valor = Operadores.funcSuma("Hola", "Mundo")
>>> valor
HolaMundo
```

4.1.8 funcResta(*args)

Objetivo

- Se encarga de restarle al primer valor el segundo valor ingresado, devolviendo el resultado de la operación.

Parámetros

- ***args : int/float**
  Los argumentos pasados como parámetro generan una lista, formada por int o floats. Son los valores de la operacion.

Salida

- Valor de la operación matemática.

Proceso

- Recibe por parámetro una lista de valores.
- Le resta al primer valor el segundo.
- Devuelve el resultado de la operación.

Ejemplo

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcResta(3, 4)
>>> valor
-1
```

4.1.9 funcMultiplicacion(*args)

Objetivo

- Se encarga de multiplicar el primer valor por el segundo valor ingresado, devolviendo el resultado de la operación.

Parámetros

- ***args : int/float**
  Los argumentos pasados como parámetro generan una lista, formada por int o floats. Son los valores de la operacion.

Salida

- Valor de la operación matemática.

Proceso

- Recibe por parámetro una lista de valores.
- Multiplica el primer valor con el segundo valor.
- Devuelve el resultado de la operación.

Ejemplo

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcMultiplicacion(3, 4)
>>> valor
12
```

4.1.10 funcDivision(*args)

Objetivo

- Se encarga de dividir el primer valor por el segundo valor ingresado, devolviendo el resultado de la operación.

Parámetros

- ***args : int/float**
  Los argumentos pasados como parámetro generan una lista, formada por int o floats. Son los valores de la operacion.

Salida

- Valor de la operación matemática.

Proceso

- Recibe por parámetro una lista de valores.
- Divide el primer valor por el segundo.
- Devuelve el resultado de la operación.

Ejemplo

Se ingresan dos valores a la función.

```
>>> valor = Operadores.funcDivision(12, 4)
>>> valor
3
```

4.2 DataFrame

4.2.1 acumular(nuevaCol, rellenar = True)

Objetivo

- Se encarga de agregar una nueva columna al DataFrame, que acumulará iterativamente los valores que se encuentren en la columna seleccionada que ya se encuentra en el diccionario.

Parámetros

- **columna: string**
  Columna de la cual la nuevaCol tomará los valores para generar los valores acumulados.
- **nuevalCol: string**
  Nombre de la nueva columna a ingresar en el DataFrame.

Proceso

- Recibe por parámetro dos columnas, una que ya se encuentra en el DataFrame, y otra columna nueva.
- Adquiere los valores de la columna del DataFrame, y los almacena acumulativamente, en el nuevo campo.

Ejemplo

Partiendo de un DataFrame (df) ya existente.

```
>>> df
Indice    ID    Outweight
0         1     22.1
1         2     12.1
2         3     20.7
```

Se ejecuta la función

```
>>> df.acumular("Outweight", "PesoAcumulado")
Indice    ID    Outweight    PesoAcumulado
0         1     22,1         22.1
1         2     12,1         34.2
2         3     20,7         54.9
```

4.2.2 agregarCol(nuevaCol, rellenar = True)

Objetivo

- Permite agregar al DataFrame una nueva columna. Si esta columna ya existiera, se eliminarán los valores de sus registros.

Parámetros

- **nuevaCol: string**
  Nombre de la nueva columna que se quiere agregar. Si la columna ya existiese, esta quedara vacía.
- **rellenar: boolean – *Default: True***
  Define si se completan los registros con un espacio vacío o no. Este parámetro se utiliza cuando este método se llama a través de otros métodos. Se recomienda dejar en **True**.

Proceso

- Recibe por parámetro el nombre de la nueva columna y si se van a rellenar los valores.
- Crea el espacio vacio para los nuevos campos de los registros.
- Si la columna no existía, la agrega al registro de columnas del DataFrame.
- Se rellanan los valores de los registros con un dato vacio si así se lo especifico.

Ejemplo

Partiendo de un DataFrame (df) ya existente.

```
>>> df
Indice    Id    StartTime    Peso
0         1     20201208     15.0
1         2     20201209     21.4
2         3     20201209     12.7
```

Se ejecuta la función

```
>>> df.agregarCol("Linea")
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    ""
1         2     20201209     21.4    ""
2         3     20201209     12.7    ""
```

Si se especifica que no se rellene la nueva columna

```
>>> df.agregarCol("Linea", rellenar = False)
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0
1         2     20201209     21.4
2         3     20201209     12.7
```

Nota: Esto puede causar errores al momento de ejecución si no se completan los registros. Se recomienda dejar en **True** al utilizar el método.

4.2.3 agregarFila(otro, indices = False)

Objetivo

*- Permite agregar un nuevo registro al DataFrame. Puede recibir tanto una lista de valores como un diccionario. Aquellos valores que no se completen, se dejaran en vacio. Para concatenar los registros de un DataFrame con otro, ver el método **anexar**.*

Parámetros

- **otro: list; dict**
  Lista/diccionario que contenga los valores del nuevo registro. Las claves del diccionario deben coincidir con el nombre de la columna a la que se insertara, mientras los valores de la lista se agregaran de forma ordenada según su posición.
- **indices: boolean – Default: False**
  Valor de verdad que permite definir si se reiniciaran los indices de los registros. Se recomienda reiniciarlos para evitar duplicaciones y mantener los registros ordenados.

Proceso

- Recibe por parámetro el nuevo registro a agregar y si se reiniciaran los indices.
- Si se recibe un diccionario:
  - Se verifica que haya registros existentes en el DataFrame. Si es así, por cada valor que se agrega, se verifica que el tipo de dato de los registros del DataFrame coincida con el del

dato en esa columna para luego agregarlo. Si no existieran registros, se agrega el dato sin ningún tipo de impedimento.
    - Si el **Índice** no estuviera dentro de las columnas del diccionario, se agrega de forma manual aumentando en uno el ultimo índice registrado.
    - Verifica que columnas no fueron completadas y rellena su valor con un dato vacio.
    - Aumenta en uno la cantidad de registros en el DataFrame.
    - Se reinician los indices si así se lo indico.
- Si se recibe una lista:
    - Se agrega de forma manual el índice del nuevo registro, aumentando en uno el ultimo índice registrado.
    - Se verifica que no se estén ingresando columnas de más.
    - Se verifica que haya registros existentes en el DataFrame. Si es así, por cada valor que se agrega, se verifica que el tipo de dato de los registros del DataFrame coincida con el del dato en esa columna para luego agregarlo. Si no existieran registros, se agrega el dato sin ningún tipo de impedimento.
    - Verifica que columnas no fueron completadas y rellena su valor con un dato vacio.
    - Aumenta en uno la cantidad de registros en el DataFrame.

Ejemplo

Se parte de un DataFrame (df) base.

```
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201209     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
```

Llamado al método.

```
>>> df.agregarFila({"Indice": 4, "Id": "5", "Linea": "A3_JUV", "Peso": 10.3})
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201209     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
4         5     ""           10.3    A3_JUV
```

Si se pide reiniciar los indices.

```
>>> df.agregarFila({“Indice”: 7, “Id”: “10”, “Linea”: “A2_JUV”}, indices = True)
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201209     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
3         10    “”           0       A2_JUV
```

Si se utiliza una lista.

```
>>> df.agregarFila([“5”, ”20201211”, 17.9])
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201209     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
3         5     20201211     17.9    “”
```

4.2.4 agrupar(columnas)

Objetivo

- Agrupa los registros según los valores en común de la/las columnas indicadas y genera nuevas subtablas a partir de ellos.

Parámetros

- **columnas: list**
  Lista de los nombres de las columnas por las que se quiere realizar la agrupación.

Salida

- Devuelve de forma iterativa una subtabla y un listado con las claves de agrupación de dicha tabla.

Proceso

- Recibe por parámetro el listado de columnas por las que se quiere agrupar.
- Mientras haya registros en el DataFrame actual:
  - Recupera el primer registro del DataFrame

- Toma la primera clave (columna) de agrupación y anota el valor que se recuperó del registro en dicha clave.
- Genera dos subtablas: La subtabla con los registros que coincidan con el valor anotado; La subtabla con los registros que no coincidan con el valor anotado.
- Verifica si existen más claves de agrupación, en caso de que existan:
  - Se vuelve a **agrupar** la tabla de los registros equivalentes.
  - Por cada listado de claves y subtabla que se genere, se arma un listado con la clave actual en adición al listado de claves que se hayan generado.
  - Se devuelve el nuevo listado de claves y la subtabla actuales.
- En caso de que no existan:
  - Se devuelve la clave actual como una lista junto con la subtabla con los registros equivalentes.
- El DataFrame de los registros que no coinciden pasa a ser el registro actual, para así volver a agrupar hasta que no queden más registros diferentes.

Ejemplo

Se parte de un DataFrame (df) base.

```
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201208     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
3         4     20201210     18.3    A2_JUV
```

Llamado al método.

```
>>> for clave, subdf in df.agrupar(["Linea"])
>>>     print(clave, "\n", subdf)
["A1_JUV"]
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201208     21.4    A1_JUV
["A2_JUV"]
Indice    Id    StartTime    Peso    Linea
2         3     20201209     12.7    A2_JUV
3         4     20201210     18.3    A2_JUV
```

Si se agrupa por más de dos columnas.

```
>>> for clave, subdf in df.agrupar(["Linea", "StartTime"])
>>>     print(clave, "\n", subdf)
["A1_JUV", "20201208"]
Indice   Id   StartTime   Peso   Linea
0        1    20201208    15.0   A1_JUV
1        2    20201208    21.4   A1_JUV
["A2_JUV", "20201209"]
Indice   Id   StartTime   Peso   Linea
2        3    20201209    12.7   A2_JUV
["A2_JUV", "20201210"]
Indice   Id   StartTime   Peso   Linea
3        4    20201210    18.3   A2_JUV
```

4.2.5 anexar(otro, indices = False)

Objetivo

- *Permite unir los registros de dos DataFrames u otro registro.*

Parámetros

- **otro: DataFrame; Dict; List**
  DataFrame, diccionario o lista que contenga datos de uno o varios registros.
- **indices: boolean – *Default: False***
  Valor de verdad que permite reiniciar los indices.

Salida

- DataFrame actual con los nuevos registros añadidos.

Proceso

- Recibe por parámetro el DataFrame/diccionario/lista y si se reiniciaran los indices.
- Verifica si se recibió un DataFrame: Si es así, itera sobre cada registro para añadirlo al DataFrame actual.
- Si se recibió un diccionario o lista, añade este al DataFrame actual.
- Tras añadir el/los registros, retorna el DataFrame.

Ejemplo

Se parte de un DataFrame (df) base.

```
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201209     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
```

Llamado al método.

```
>>> df.anexar({"Indice": 4, "Id": "5", "Linea": "A3_JUV", "Peso": 10.3})
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201209     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
4         5     ""           10.3    A3_JUV
```

Si se pide reiniciar los indices.

```
>>> df.anexar({"Indice": 7, "Id": "10", "Linea": "A2_JUV"}, indices = True)
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201209     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
3         10    ""           0       A2_JUV
```

Si se utiliza una lista.

```
>>> df.anexar(["5", "20201211", 17.9])
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201209     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
3         5     20201211     17.9    ""
```

Si se utiliza otro DataFrame (df2)

```
>>> df2
Indice    Id    StartTime    Peso    Linea
0         8     20201215     19.1    A3_JUV
1         9     20201216     9.6     A1_JUV
2         10    20201216     24.7    A2_JUV
```

Llamado al método realizado sobre el DataFrame original.

```
>>> df.anexar(df2, True)
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201209     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
3         8     20201215     19.1    A3_JUV
4         9     20201216     9.6     A1_JUV
5         10    20201216     24.7    A2_JUV
```

4.2.6 buscar(condiciones = None)

Objetivo

- Genera una subtabla con aquellos registros que cumplan con una condición lógica especifica.

Parámetros

- **condiciones: dict – *Default: None***
  Diccionario cuyas claves son las columnas en las que se realizara una condición y sus valores son una tupla de dos valores: Un operador lógico y un valor a comparar.

Salida

- Retorna una subtabla con todos los registros que cumplan con la condición.

Proceso

- Recibe por parámetro el diccionario de condiciones definidas
- Si hay condiciones:
  - Itera sobre cada fila del registro actual

- Por cada condición que el diccionario de condiciones posea, verifica si esa fila la cumple.
- Si llegase a haber una condición que no se cumpliese, la fila no se incluirá en la subtabla. En caso contrario, se la agregara y se aumentara en uno la cantidad de registros de la subtabla.
- Crea los atributos bases para la nueva subtabla y crea una nueva instancia de DataFrame que se le agregan los registros que cumplen con la condición para luego retornarlo.

Ejemplo

Se parte de un DataFrame (df) base.

```
>>> df
Indice    Id    StartTime    Peso    Linea
0         1     20201208     15.0    A1_JUV
1         2     20201209     21.4    A1_JUV
2         3     20201209     12.7    A2_JUV
```

Llamado al método.

```
>>> dfFiltrado = df.buscar({"Linea": ["==", "A1_JUV"], "Peso": [">", 20]})
>>> dfFiltrado
Indice    Id    StartTime    Peso    Linea
1         2     20201209     21.4    A1_JUV
```

4.2.7 cambiarColumnas(lista)

Objetivo

- Se encarga de modificar el nombre de las columnas con los de la lista de nombres pasados, reemplazándolos en orden respectivo en la tabla.

Parámetros

- **Lista : list**
  Lista de Strings/texto con los nombres de las nuevas columnas, debe tener el mismo largo que las columnas utilizadas previamente.

Proceso

- Recibe por parámetro una lista de nombres para reemplazar la lista de nombres de las columnas del diccionario.

- Modifica los valores de los nombres de las columnas viejas, con los nuevos pasados (respectivamente 1 a 1).
- Retorna el diccionario con los nombres de las columnas modificados.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    Col0    Col1
0         1       22,1
1         2       12,1
2         3       20,7
```

Utilizamos la función para cambiar las columnas de la tabla.

```
>>> df.cambiarColumnas(["ID", "Outweight"])
Indice    ID      Outweight
0         1       22,1
1         2       12,1
2         3       20,7
```

4.2.8 cambiarValor(columna, valor, inplace = False, condiciones = None)

Objetivo

- Permite cambiar los valores de cada celda de una columna, ya sea a un valor fijo o realizando una operación matemática. Así mismo, permite cambiar los valores de aquellos registros que cumplan con una o varias condiciones.

Parámetros

- **columna: string**
  Columna en la cual se realizará el cambio de valores.
- **valor: list, str, int, float, boolean**
  Puede tratarse de una lista de operaciones de la forma:
  valor = [*operación, operación, operación,* …]
  operación = [*operador, valor*]
  Ejemplo: [[“+”, 15], [“-“, 4], [“/”, 2]]

También puede definirse un valor fijo que se aplicara sobre cada registro o una lista normal cuyos valores se irán aplicando en el orden en que llegan.

- **inplace: boolean – *Default: False***
  Si es **True**, se realizarán los cambios sobre el mismo DataFrame. En caso contrario, se devolverá una copia no vinculada del DataFrame con los datos actualizados.
- **condiciones: dict– *Default: None***
  Diccionario cuyas claves son las columnas en las que se realizara una condición y sus valores son una tupla de dos valores: Un operador lógico y un valor a comparar.

Salida

- Si **inplace = False**, retorna una copia del DataFrame con los cambios realizados.

Proceso

- Recibe por parámetro la columna y el valor a aplicar en los registros. Opcionalmente, puede recibir si se realizara los cambios en el DataFrame o en una copia, y si los registros tienen que cumplir ciertas condiciones.
- Si hay condiciones, se **busca** que registros cumplen con las mismas.
- Si el valor a aplicar es una lista de operaciones, se **ejecutará cada operación** de izquierda a derecha.
- Si es un listado normal, se cambiar los valores en el orden que vengan. Si no hay suficientes valores en la lista, se mantendrá el valor original de los registros restantes
- Si es un valor único, se aplicará sobre todos los registros.
- Retornara el DataFrame si se esta trabajando con una copia.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      Outweight      Linea
0         1       22.1           A1_JUV
1         2       12.1           A1_JUV
2         3       20.7           A2_JUV
```

Se realiza el llamado al método.

```
>>> dfCambios = df.cambiarValor("Linea", [["+", 5]])
>>> dfCambios
Indice    ID      Outweight      Linea
0         1       27.1           A1_JUV
1         2       17.1           A1_JUV
2         3       25.7           A2_JUV
```

Se llama a la función y se realiza el cambio sobre el DataFrame original.

```
>>> df.cambiarValor("Linea", [["+", 5]], True)
>>> df
Indice    ID    Outweight    Linea
0         1     27.1         A1_JUV
1         2     17.1         A1_JUV
2         3     25.7         A2_JUV
```

Si se ejecuta con condiciones.

```
>>> df.cambiarValor("Linea", "A3_JUV", True, {"Linea": ["==", "A1_JUV"]})
>>> df
Indice    ID    Outweight    Linea
0         1     27.1         A3_JUV
1         2     17.1         A3_JUV
2         3     25.7         A2_JUV
```

4.2.9 cambiarTipo(diccionarioCambio)

Objetivo

- Se encarga de modificar el tipo de dato de una columna.

Parámetros

- **cambiarTipo: dict**
  Diccionario de clave valor:
  - Clave: Columna a modificar
  - Valor: Tipo de dato al que se modificara la columna.

Proceso

- Recibe por parámetro el diccionario con la columna y un tipo de dato.
- Modifica el tipo de dato de los valores en la columna solicitada mediante un mapeo.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Id              int
Outweight       string

Indice    Id      Outweight
0         1       “22.1”
```

Utilizamos la función para cambiar las columnas de la tabla.

```
>>> df.cambiarTipo({"Outweight": float})
Id              int
Outweight       float

Indice    Id      Outweight
0         1       22.1
```

4.2.10 contiene(columna, valor)

Objetivo

- Se encarga de devolver una lista de valores booleanos (True o False) en tanto los valores que sean ingresados, se encuentren dentro de la columna indicada.

Parámetros

- **columna : string**
  Columna del DataFrame de la cual se buscará la subcadena.
- **valor : string**
  La cadena que se verificara si se encuentra dentro de los valores de la columna.

Salida

- Lista de valores booleanos (Verdadero/Falso) referentes a cada registro del DataFrame.

Proceso

- Recibe por parámetro una columna que se encuentre en el diccionario y la cadena de texto que se buscara.
- Verifica si la cadena se encuentra dentro de los valores de la columna, devolviendo True si el valor se encuentra, o False si no.
- Devuelve una lista de valores True/False.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      Outweight
0         1       22,1
1         2       12.1
2         3       20,7
```

Utilizamos la función para cambiar las columnas de la tabla.

```
>>> lista = df.contiene("Outweight", ",")
>>> lista
[True, False, True]
```

4.2.11 copiar(deep = True)

Objetivo

- *Se encarga de realizar una copia del diccionario, ya sea una copia "superficial" la cual modifica el diccionario original, y una copia "profunda", que es independiente del diccionario original.*

Parámetros

- **Deep: boolean – *Default: True***
  Define si la copia no estará vinculada o si lo estará.

Salida

- Retorna la copia del DataFrame.

Proceso

- Recibe por parámetro la condición de si se realizara una copia vinculada.
- Verifica el valor ingresado: Si es True, realiza una copia "profunda", esto es independiente del diccionario original; Si el valor ingresado es "False", realiza una copia "superficial", a la cual, si se le realizan modificaciones, modificará también los valores del diccionario original.
- Devuelve la copia de un diccionario.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
```

Se realiza una copia no vinculada de la tabla.

```
>>> dfCopia = df.copiar(True)
>>> dfCopia
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
```

Si se modifica la copia.

```
>>> dfCopia.cambiarTipo({"Outweight": str})
Indice    ID      Outweight
0         1       "22.1"
1         2       "12.1"
2         3       "20.7"
```

El DataFrame se mantiene intacto.

```
>>> df
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
```

En cambio, si se realiza una copia vinculada de la tabla.

```
>>> dfCopia = df.copiar(False)
>>> dfCopia
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
```

Si se modifica la copia.

```
>>> dfCopia.cambiarTipo({"Outweight": str})
Indice    ID      Outweight
0         1       "22.1"
1         2       "12.1"
2         3       "20.7"
```

El DataFrame se verá modificado.

```
>>> df
Indice    ID      Outweight
0         1       "22.1"
1         2       "12.1"
2         3       "20.7"
```

4.2.12 eliminar(indices = None, columnas = None, inplace = False)

Objetivo

- *Elimina las columnas y/o registros indicados.*

Parámetros

- **indices: list – *Default: None***
  Listado de indices de los registros que se eliminaran del DataFrame.
- **columnas: None – *Default: None***
  Listado de nombre de columnas que se eliminaran del DataFrame.
- **inplace boolean – *Default: False***
  Determina si los cambios se realizaran sobre el DataFrame original o sobre una copia.

Salida

- Devuelve una copia del DataFrame si así se lo indico.

Proceso

- Recibe por parámetro el listado de indices y/o columnas a eliminar, como así la opción de si se realizaran los cambios sobre el DataFrame original o sobre una copia.
- Se realiza una copia del DataFrame si así se lo desea.
- Por cada índice que se indicó, se lo busca dentro del DataFrame y se lo elimina, decrementando en uno la cantidad de registros del DataFrame.

- Por cada columna que se indicó, se la busca dentro del DataFrame y se la elimina, al igual que todos los valores en ella.
- Al finalizar, si se esta trabajando con una copia, se devuelve la misma.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID        Outweight
0         1         22.1
1         2         12.1
2         3         20.7
```

Se realiza el llamado al método.

```
>>> dfEliminados = df.eliminar([0], ["Outweight"])
>>> dfEliminados
Indice    ID
1         2
2         3
```

Si se trabaja sobre el DataFrame original.

```
>>> df.eliminar([0], ["Outweight"], True)
>>> df
Indice    ID
1         2
2         3
```

4.2.13 exportarCSV(ruta, separador = ",", columnas = False, índice = False)

Objetivo

- Se encarga de crear un archivo CSV a partir del DataFrame.

Parámetros

- **ruta: string**
  Directorio o Path a donde se quiere guardar el CSV.
- **separador: string – *Default: “,”***
  Carácter que separara los campos en el archivo CSV. Por regla general, establecido como coma (**,**).
- **columnas : boolean – *Default: False***
  Define si se escriben el nombre de las columnas en el primer registro del archivo CSV.
- **índice : boolean – *Default: False***
  Define si se escriben los indices de los registros del CSV.

Proceso

- Recibe por parámetro, la ruta/directorio donde se quiere guardar el CSV, el separador y si se escribirán los nombres de las columnas y los indices.
- Genera el archivo CSV en el directorio solicitado.
- Si se solicitó, adquiere los nombres de las columnas del Dataframe y los escribirá en el archivo CSV con el separador de por medio.
- Por cada registro en el DataFrame, escribe sus datos en el archivo CSV, incluyendo el índice si así se indicó.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
```

Se realiza la exportación a un archivo CSV.

```
>>> df.exportarCSV(“directorio”)
1,22.1
2,12.1
3,20.7
```

También se puede realizar la exportación cambiando el separador y agregando el nombre de las columnas y/o los indices de los registros.

```
>>> df.exportarCSV("directorio", ";", True, True)
Indice;ID;Outweight
0;1;22.1
1;2;12.1
2;3;20.7
```

4.2.14 final(cantidad = 5, columnasSelect = None)

Objetivo

- *Devuelve los últimos N registros del DataFrame. Por defecto, devuelve 5.*

Parámetros

- **cantidad: int – *Default: 5***
  Cantidad de registros a mostrar.
- **columnasSelect: list**
  Listado de columnas que se quiera mostrar.

Salida

- Retorna una copia del DataFrame con los últimos N registros que se quieren mostrar.

Proceso

- Se puede recibir por parámetro la cantidad de registros a mostrar y que columnas se mostraran.
- Se realiza una copia del DataFrame.
- Se eliminan aquellas columnas que no se quieren mostrar.
- Recupera las ultimas N filas y las carga en el nuevo DataFrame.
- Retorna el DataFrame con las ultimas N filas.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID        Outweight
0         1         22.1
1         2         12.1
2         3         20.7
3         4         21.1
4         5         18.3
5         6         27.0
```

Se realiza el llamado al método.

```
>>> dfFinal = df.final()
>>> dfFinal
Indice    ID        Outweight
1         2         12.1
2         3         20.7
3         4         21.1
4         5         18.3
5         6         27.0
```

Se realiza el llamado a la función especificando parámetros.

```
>>> dfFinal = df.final(3, "ID")
>>> dfFinal
Indice    ID
3         4
4         5
5         6
```

4.2.15 indices()

Objetivo

- *Genera el listado de los índices de cada registro del DataFrame.*

Salida

- Retorna el listado de indices

Proceso

- Itera sobre cada registro del DataFrame actual.
- Recupera el índice actual de cada registro.
- Devuelve el listado de índices.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID        Outweight
0         1         22.1
1         2         12.1
2         3         20.7
```

Se realiza el llamado al método.

```
>>> listaIndices = df.indices()
>>> listaIndices
[0, 1, 2]
```

4.2.16 iterar(*args)

Objetivo

- *Itera sobre los registros del DataFrame.*

Parámetros

- ***args: X - strings**
  Permite recibir X cantidad de parámetros que serán los nombres de las columnas que se quieran visualizar.

Salida

- Retorna uno por uno cada registro del DataFrame.

Proceso

- Se puede recibir un listado de parámetros que serán los nombres de las columnas a visualizar.
- Si se detallaron que columnas se desean ver, se extraerán únicamente los datos de los registros de dichas columnas. En caso contrario, devuelve todos los campos.
- Retorna la fila actual y continua hasta devolver el ultimo registro del DataFrame.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
```

Se realiza el llamado al método.

```
>>> for fila in df.iterar():
>>>     print(fila)
{'Indice': 0, 'ID': '1', 'Outweight', 22.1}
{'Indice': 1, 'ID': '2', 'Outweight', 12.1}
{'Indice': 2, 'ID': '3', 'Outweight', 20.7}
```

Si se especifican columnas.

```
>>> for fila in df.iterar("ID"):
>>>     print(fila)
{'Indice': 0, 'ID': '1'}
{'Indice': 1, 'ID': '2'}
{'Indice': 2, 'ID': '3'}
```

4.2.17 ordenar(columnas, ascendente = True, indices = False)

Objetivo

- Ordena el DataFrame actual según la/las columnas indicadas, ya sea de forma ascendente o descendente.

Parámetros

- **columnas: list**
  Listado de columnas por las que se realizara el ordenamiento.
- **ascendente: boolean – *Default: True***
  Valor de verdad. Si es **True**, ordena de forma ascendente. En caso contrario, descendente.
- **indices: boolean – *Default: False***
  Indica si se reiniciaran los indices una vez ordenado el DataFrame.

Salida

- Retorna el DataFrame ordenado.

Proceso

- Recibe por parámetro el listado de columnas por el que se ordenara y, opcionalmente, si se ordenara de forma ascendente/descendente y si se reiniciaran los indices.
- Se verifica que haya mas de un registro a ordenar y que se hayan recibido columnas.
- Recupera el primer registro del DataFrame para usarlo de pivot y la primera columna por la que se ordenara.
- Se generan tres subtablas:
  1) La tabla con todos los registros que sean menores al valor del pivot.
  2) La tabla con todos los registros que sean iguales al valor del pivot.
  3) La tabla con todos los registros que sean mayores al valor del pivot.
- Se **ordenan** cada una de las tablas generadas anteriormente: las tablas de mayores y menores se ordenan por todas las columnas indicadas; La tabla pivot se ordenará por el resto de columnas en el listado sin contar la columna actual.
- Si se ordenara de forma ascendente, se **anexarán** las tablas en el siguiente orden: tabla menores -> tabla pivot -> tabla mayores
  Si se ordenara de forma descendente, se **anexarán** las tablas en el siguiente orden: tabla mayores -> tabla pivot -> tabla menores
- Se reiniciarán los indices si así se lo indico.
- Retorna el DataFrame ordenado.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
```

Se realiza el llamado al método.

```
>>> dfOrdenado = df.ordenar("Outweight"):
>>> dfOrdenado
Indice    ID      Outweight
1         2       12.1
2         3       20.7
0         1       22.1
```

Si se solicita que se reinicien los indices.

```
>>> dfOrdenado = df.ordenar("Outweight", indices = True):
>>> dfOrdenado
Indice    ID      Outweight
0         2       12.1
1         3       20.7
2         1       22.1
```

4.2.18 principio(cantidad = 5, columnasSelect = None)

Objetivo

- *Devuelve los primeros N registros del DataFrame. Por defecto, devuelve 5.*

Parámetros

- **cantidad: int – *Default: 5***
  Cantidad de registros a mostrar.
- **columnasSelect: list**
  Listado de columnas que se quiera mostrar.

Salida

- Retorna una copia del DataFrame con los primeros N registros que se quieren mostrar.

Proceso

- Se puede recibir por parámetro la cantidad de registros a mostrar y que columnas se mostraran.
- Se realiza una copia del DataFrame.
- Se eliminan aquellas columnas que no se quieren mostrar.
- Recupera las primeras N filas y las carga en el nuevo DataFrame.
- Retorna el DataFrame con las primeras N filas.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
3         4       21.1
4         5       18.3
5         6       27.0
```

Se realiza el llamado al método.

```
>>> dfFinal = df.principio()
>>> dfFinal
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
3         4       21.1
4         5       18.3
```

Se realiza el llamado a la función especificando parámetros.

```
>>> dfFinal = df.final(3, "ID")
>>> dfFinal
Indice    ID
0         1
1         2
2         3
```

4.2.19 redondear(kwargs)

Objetivo

- Redondea los valores de las columnas seleccionadas del DataFrame a partir de la según la cantidad de decimales que se deseen.

Parámetros

- **kwargs: dict**
  Diccionario cuyas claves son las columnas a modificar, y el valor es un entero que representa la cantidad de decimales del redondeo.

Proceso

- Recibe por parámetro el diccionario de columnas y la cantidad de decimales para cada una.
- Lee la columna solicitada y su primer valor, para corroborar que el dato sea un número (caso contrario da aviso de error).
- Modifica los valores de la columna solicitada, redondeándolos a la cantidad de decimales indicados.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      Outweight
0         1       22.102
1         2       12.1965
2         3       20.734
```

Se realiza el llamado al método.

```
>>> df.indices({"Outweight": 1})
>>> df
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
```

4.2.20 reiniciarIndices(inplace = False)

Objetivo

- Reinicia los indices del DataFrame para que figuren de la forma: 0, 1, 2, … N-1.

Parámetros

- **inplace: boolean – *Default: False***
  Especifica si la acción se aplicara sobre el DataFrame original o en una copia del mismo.

Salida

- Retorna una copia del DataFrame si así se lo indica.

Proceso

- Puede recibir por parámetro si se decide trabajar sobre el DataFrame original o no.
- Elimina todos los indices actuales de los registros.
- Empieza a agendar los nuevos indices de forma ordenada según la cantidad de filas en el DataFrame.
- Se devuelve la copia del DataFrame si se está trabajando con una.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID        Outweight
5         1         22.1
1         2         12.1
2         3         20.7
3         4         21.1
8         5         18.3
```

Se realiza el llamado al método.

```
>>> df.reiniciarIndices(inplace = True)
>>> df
Indice    ID        Outweight
0         1         22.1
1         2         12.1
2         3         20.7
3         4         21.1
4         5         18.3
```

4.2.21 remplazar(columna, cadena, remplazo)

Objetivo

- Remplaza la cadena de texto indicada que posean todos los registros en una columna por un valor explicito.

Parámetros

- **columna : string**
  Nombre de la columna en la cual se realizará el remplazo.
- **cadena : string**
  Valor que se busca remplazar dentro de cada registro.
- **remplazo: string**
  Valor que reemplazará la cadena de texto indicada si se encuentra dentro del registro.

Proceso

- Recibe por parámetro el nombre de la columna, la cadena de texto a remplazar y el valor que la reemplazara.
- Verifica si se encuentra la cadena dentro de cada registro de la columna especificada en el DataFrame. Si es así, reemplazará el valor previo con el nuevo valor.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      Outweight
0         1       22,1
1         2       12.1
2         3       20,7
```

Se realiza el llamado al método.

```
>>> df.remplazar("Outweight", ",", ".")
>>> df
Indice    ID      Outweight
0         1       22.1
1         2       12.1
2         3       20.7
```

4.2.22 separar(columna, valor, posicion = None)

Objetivo

- Se encarga de separar los valores dentro de una columna según una subcadena o carácter y generar una lista a partir de ellos.

Parámetros

- **columna: string**
  Nombre de la columna en la que se efectuara la separación.

- **valor : string**
  Cadena o carácter de texto que se utilizará para separar el texto de los registros en la columna.
- **posicion: int – *Default: None***
  Numero entero que indica que porción de la cadena dividida se tiene que devolver. Si se deja por defecto, devuelve todo el listado de separaciones.

Salida

- Listado de cadenas divididas. Cada item de la lista se trata de otra lista con las subdivisiones de cada registro.

Proceso

- Recibe por parámetro una columna que se encuentre en el diccionario, el separador y, opcionalmente, la porción de la separación que se quiere devolver.
- Divide o separa los valores de la columna, mediante el separador indicado.
- Devuelve una lista de listas con todos los valores de la separación en caso de no haberse indicado una posición. En caso de seleccionar posición, devuelve una lista de los valores correspondientes a dicha posición en cada registro.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      TipoCampana
0         1       5132_MA
1         2       5132_MA
2         3       5132_MA
```

Se realiza el llamado al método.

```
>>> listaSeparaciones = df.separar("TipoCampana", "_")
>>> listaSeparaciones
[["5132","MA"], ["5132","MA"], ["5132","MA"]]
```

Si se especifica la posición que se quiere devolver.

```
>>> listaSeparaciones = df.separar("TipoCampana", "_", 0)
>>> listaSeparaciones
["5132", "5132", "5132"]
```

4.2.23 vacio()

Objetivo

- Se encarga de verificar si el DataFrame tiene registros o no.

Salida

- Devuelve un valor de verdadero/falso, según corresponda.

Proceso

- Verifica si la cantidad de filas en el DataFrame es diferente de 0. Si es así, devuelve **False**. En caso contrario, **True**.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID        TipoCampana
0         1         5132_MA
1         2         5132_MA
2         3         5132_MA
```

Se realiza el llamado al método.

```
>>> valor = df.vacio()
>>> valor
False
```

4.2.24 valores(*args)

Objetivo

- Se encarga de generar una lista de valores de cada columna.

Parámetros

- ***args: X - strings**
  Permite recibir X cantidad de parámetros que serán los nombres de las columnas que se quieran visualizar

Salida

- Devuelve una lista que contiene un listado de valores por cada columna del DataFrame.

Proceso

- Puede recibir por parámetro un listado de columnas a visualizar.
- Genera un listado de los valores que posee cada columna seleccionada. Si no se especificaron columnas, genera una lista de los valores por cada columna del DataFrame.
- Retorna el listado de valores

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice   ID      TipoCampana     Outweight
0        1       5132_MA         22.1
1        2       5132_MA         12.1
2        3       5132_MA         20.7
```

Se realiza el llamado al método.

```
>>> listaValores = df.valores()
>>> listaValores
[[“1”, “2”, “3”], [“5132_MA”, “5132_MA”, “5132_MA”], [22.1, 12.1, 20.7]]
```

4.2.25 colMax(columnmax = None)

Objetivo

- *Realizar una comparación entre todos los valores numéricos de la columna, devolviendo el mayor.*

Parámetros

- **columnmax: str**
  Nombre de la columna del DataFrame a la que se le realizará la comparación de valores.

Salida

- Devuelve el valor máximo.

Proceso

- Recibe por parámetro una columna.
- Recorre cada valor de la columna del DataFrame, comparando los valores uno a uno y quedándose siempre con el mayor.

- Devuelve el valor máximo.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID        Outweight
0         1         22.1
1         2         12.1
2         3         20.7
```

Se realiza el llamado al método.

```
>>> maximo = df.colMax( "Outweight")
>>> maximo
22.1
```

4.2.25 colMin(columnmin = None)

Objetivo

- *Realizar una comparación entre todos los valores numéricos de la columna, devolviendo el menor.*

Parámetros

- **columnmin: str**
  Nombre de la columna del DataFrame a la que se le realizará la comparación de valores.

Salida

- Devuelve el valor mínimo.

Proceso

- Recibe por parámetro una columna.
- Recorre cada valor de la columna del DataFrame, comparando los valores uno a uno y quedándose siempre con el menor.

- Devuelve el valor mínimo.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID        Outweight
0         1         22.1
1         2         12.1
2         3         20.7
```

Se realiza el llamado al método.

```
>>> mínimo = df.colMin( "Outweight")
>>> mínimo
12.1
```

4.2.27 Acum(columacum = None)

Objetivo

- *Realizar una suma acumulativa de los valores de una columna.*

Parámetros

- **columacum: str**
  Nombre de la columna del DataFrame a la que se le realizará una suma acumulativa.

Salida

- Devuelve el total acumulado.

Proceso

- Recibe por parámetro una columna.
- Recorre cada valor de la columna del DataFrame, sumando de forma acumulativa.

- Devuelve la suma acumulada.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID        Outweight
0         1         22.1
1         2         12.1
2         3         20.7
```

Se realiza el llamado al método.

```
>>> acumulado = df.Acum( "Outweight")
>>> acumulado
54.9
```

4.2.28 unique(columunique = None)

Objetivo

- *Realizar una comparación de los valores que hay en la columna, y devuelve una lista de los mismos sin repetirlos.*

Parámetros

- **columunique: str**
  Nombre de la columna del DataFrame a la que se le realizará la comparación.

Salida

- Devuelve una lista de los distintos valores que hay en la columna.

Proceso

- Recibe por parámetro una columna.
- Recorre cada valor de la columna del DataFrame, comparando valores, y agregando a la lista aquellos que no se encuentran.

- Devuelve una lista de los distintos valores.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID        Linea
0         1         'A3_JUV'
1         2         'GT2_JUV'
2         3         'GT2_JUV'
```

Se realiza el llamado al método.

```
>>> val_unicos = df.unique( "Linea")
>>> val_unicos
['A3_JUV','GT2_JUV']
```

4.2.29 nunique(columunique = None)

Objetivo

*- Devuelve la longitud de la lista de valores únicos generados por el método **unique()**.*

Parámetros

- **Columunique: str**
  Nombre de la columna del DataFrame a la que se le realizará la comparación.

Proceso

- Recibe por parámetro una columna.
- Llama a la función **unique()**, pasándole la columna de interés. Obtiene los distintos valores únicos de la lista, y verifica su longitud.
- Devuelve la longitud de la lista.

Salida

- Devuelve un valor numérico correspondiente a la longitud, de los valores únicos.

Ejemplo

Partiendo de un Dataframe (df).

```
>>> df
Indice    ID      Linea
0         1       'A3_JUV'
1         2       'GT2_JUV'
2         3       'GT2_JUV'
```

Se realiza el llamado al método.

```
>>> num_unicos = df.nunique( "Linea")
>>> num_unicos
2
```

4.2.30 innerjoin(dfderecha , columnmatch, columnas_izq = None, columnas_der = None)

Objetivo

- *Devuelve un DataFrame sobre el cual se realiza la unión de dos DataFrames.*

Parámetros

- **dfderecha: DataFrame**
  Objeto tipo DataFrame.
- **columnmatch: dict**
  Diccionario formado por llaves que son las columnas del DataFrame que ya tenemos, con los valores que son las columnas del DataFrame(**dfderecha**) que estamos pasando.
- **columnas_izq: list**
  Parámetro opcional tipo lista correspondiente al DataFrame sobre el que llamamos el método.
- **columnas_der: list**
  Parámetro opcional tipo lista correspondiente al DataFrame(**dfderecha**) que pasamos.

Proceso

- Recibe por parámetro un DataFrame, un diccionario, y puede recibir una, dos o ninguna lista.
- Guarda una lista de las columnas que se hayan pasado como parámetro, por ejemplo columnas_izq o columnas_der (o ambas), en caso de que no reciba

ninguna de estas listas como parámetro, toma por default una lista de las columnas en los DataFrames sobre los que se están trabajando .

- Trabaja sobre el diccionario **columnmatch** pasado como parámetro, , siendo las llaves de ese diccionario las columnas correspondientes al DataFrame sobre el que se trabaja, y los valores las columnas del DataFrame(**dfderecha**) que pasa como parámetro.
- Elimina Columnas duplicadas en caso de que las haya.
- Usa la función **buscar(),** se comparan los valores pasados en **columnmatch** y se trae un nuevo DataFrame con las filas que cumplen únicamente el criterio solicitado.
- Agrega las filas recién obtenidas a un nuevo DataFrame.
- Devuelve el DataFrame unido.

Salida

- Se obtiene como Output un nuevo DataFrame.

Ejemplo 1, sin parámetro de columnas.

Partiendo de dos DataFrames:

```
>>> df
Indice    ID      Linea        PesoMax
0         1       'A3_JUV'      3000
1         2       'GT2_JUV'     3500
2         3       'GT2_JUV'     4000
```

```
>>> df2
Indice    ID      Madre        PesoSalida
0         1       '7771_MA'     1200
1         2       '7772_MA'     1300
2         3       '7773_MA'     1400
```

Se realiza el llamado al método. Y el DataFrame resultante es:

```
>>> df_unido = df.innerJoin(df2, {"ID":"ID"})
>>> df_unido
>>> df2
Indice    ID      Linea        PesoMax      Madre        PesoSalida
0         1       'A3_JUV'      3000       '7771_MA'     1200
1         2       'GT2_JUV'     3500       '7772_MA'     1300
2         3       'GT2_JUV'     4000       '7773_MA'     1400
```

Ejemplo 2, con parámetro de columnas_der.

Teniendo los mismos **df** y **df2** de antes. Se realiza el llamado al método. Y el DataFrame resultante es:

```
>>> df_unido = df.innerJoin(df2, {"ID":"ID"}, columnas_der = ["Madre"])
>>> df_unido
>>> df2
```

Indice	ID	Linea	PesoMax	Madre
0	1	'A3_JUV'	3000	'7771_MA'
1	2	'GT2_JUV'	3500	'7772_MA'
2	3	'GT2_JUV'	4000	'7773_MA'

Ejemplo 3, con parámetro de columnas_izq.

Teniendo los mismos **df** y **df2** de antes. Se realiza el llamado al método. Y el DataFrame resultante es:

```
>>> df_unido = df.innerJoin(df2, {"ID":"ID"}, columnas_izq = ["Linea"])
>>> df_unido
```

Indice	Linea	ID	Madre	PesoSalida
0	'A3_JUV'	1	'7771_MA'	1200
1	'GT2_JUV'	2	'7772_MA'	1300
2	'GT2_JUV'	3	'7773_MA'	1400

Ejemplo 4, con parámetro de columnas_izq y columnas_der.

Teniendo los mismos **df** y **df2** de antes. Se realiza el llamado al método. Y el DataFrame resultante es:

```
>>> df_unido = df.innerJoin(df2, {"ID":"ID"}, columnas_izq = ["Linea"], columnas_der =
["Madre"] )
>>> df_unido
```

Indice	Linea	Madre
0	'A3_JUV'	'7771_MA'
1	'GT2_JUV'	'7772_MA'
2	'GT2_JUV'	'7773_MA'