TP Final: Documentación de Análisis

Algoritmos 1

Grupo 7

Librería para el análisis de datos en JAVA

El objetivo del proyecto es la creación de una librería que permita manipular y analizar datos en forma tabular (2 dimensiones) para el lenguaje Java.

El alcance del proyecto incluye el diseño, desarrollo e implementación de las siguientes funcionalidades en la librería: Información básica (cantidad y etiquetas de filas y columnas), acceso indexado a las filas y columnas, carga y descarga de archivos en formato CSV, visualización de la información en forma de tabla, generación y modificación de tablas, selección parcial de la estructura tabular, filtrado de los datos de la tabla utilizando comparadores combinados con operadores lógicos, copia independiente de la estructura tabular, concatenación de tablas, ordenamiento de las filas según un criterio, aplicación de operaciones de sumarización estadística, modificación de las celdas con valores faltantes con cierto valor literal indicado, muestreo de los datos de la tabla en función de un porcentaje del total de la estructura.

Users Stories:

- Como usuario, quiero presentar la información de forma tabular para poder trabajar con los datos.
    - Las tablas van a ser listas de columnas, que a su vez son listas de celdas.
    - Los tipos de datos soportados para una columna son: Numérico, Booleano y Cadena.
    - A la hora de generar la tabla, vamos a tener distintos métodos constructores, uno en el que reciba de argumento un archivo CSV y otro que instancie una tabla vacía. También voy a poder generar tablas recibiendo como argumento matrices nativas de Java.
        - **DoD:**
            - Puedo instanciar la clase Tabla a partir de un archivo csv o de una matriz nativa.
            - Puede aplicarsele métodos para su análisis y visualización.

- Como usuario, quiero saber la cantidad de filas y columnas de la tabla para conocer las dimensiones de mi dataset.
    - Las columnas van a ser listas, por lo que podemos calcular la cantidad de filas como la longitud de las listas.
    - Para calcular la cantidad de columnas, calculamos la cantidad de listas que componen la tabla.
        - **DoD:**
            - Cuando llamo al método *obtenerCantidadFilas()* me devuelve el número total de filas. Si la tabla está vacía, devuelve 0
            - Cuando llamo al método *obtenerCantidadColumnas()* me devuelve el número total de columnas. Si la tabla está vacía, devuelve 0

- Como usuario, quiero saber las etiquetas de las columnas y filas, y los tipos de datos en las columnas para entender mejor la composición de la tabla.
    - Las etiquetas de las columnas son la primera línea del archivo si tiene encabezado, o una lista incremental inicializada en 0 si no lo tiene.
    - Las etiquetas de las filas son la primera columna del archivo si se especifica que tiene etiquetas de filas, o una lista incremental inicializada en 0 en caso contrario.
        - **DoD:**
            - Cuando llamo al método obtenerEtiquetasColumnas(), devuelve una lista de tipo de dato etiquetas con las etiquetas de las columnas. Si la tabla está vacía, devuelve una lista vacía.
            - Cuando llamo al método obtenerEtiquetasFilas(), devuelve una lista de tipo de dato etiquetas con las etiquetas de las filas. Si la tabla está vacía, devuelve una lista vacía.

- El método obtenerTipoDatoColumna(Etiqueta etiquetaColumna), recibe como parámetro la etiqueta de una columna y devuelve el tipo de dato de dicha columna. Si no existe la etiqueta que se indicó como parámetro, se mostrará el mensaje de error pertinente.
- El método obtenerTipoDeDato(), devuelve una lista de los tipos de datos de la tabla. Si la tabla está vacía, devuelve una lista vacía.

● Como usuario, quiero acceder a una fila completa seleccionando la etiqueta de la fila, para poder tener acceso más directo a la información que quiero ver.

- Para acceder a una fila con la etiqueta, busco en todas las columnas el elemento que está en el índice de la etiqueta.
  - **DoD:**
    - El método obtenerFila(Etiqueta etiquetaFila) devuelve una Fila con los datos de las celdas en la fila indicada.
    - La lista está completa y correcta con los datos de la fila elegida.
    - Si no existe etiquetaFila o la tabla está vacía, se mostrará el mensaje de error pertinente.

● Como usuario, quiero acceder a una columna completa seleccionando la etiqueta de la columna, para poder tener acceso más directo a la información que quiero ver.

- Para acceder a una columna con su etiqueta, accedo a la columna desde la etiqueta.
  - **DoD:**
    - El método obtenerColumna(Etiqueta etiquetaColumna) devuelve una instancia de Columna que contiene los datos de dicha columna.
    - La lista está completa y correcta con los datos de la columna elegida.
    - Si la etiqueta no existe o la tabla está vacía, se mostrará el error pertinente.

● Como usuario, quiero acceder a una celda específica indicando las etiquetas de fila y columna simultáneamente, para poder tener acceso más directo a la información que quiero ver.

- Para acceder a una celda específica, busco en la columna que tiene la etiqueta de columna indicada, el valor definido por la etiqueta de fila.
  - El método obtenerCelda(Etiqueta etiquetaColumna, Etiqueta etiquetaFila) devuelve el valor de la celda que vive en la ubicación indicada.
  - El valor es correcto con el buscado.
  - Si la posición buscada no existe, se mostrará el error pertinente.

- Como usuario, quiero que el programa me permita trabajar con archivos CSV en disco.
    - Itero sobre la primera línea y la traduzco en un array de Celda. La longitud de ese array me va a indicar la cantidad de Columnas que tendrá la tabla. Recorro ese array y agrego las celdas a las columnas de mi tabla.
    - Luego sigo iterando sobre las demas líneas del CSV y voy agregando los datos Celda a las Columnas.
    - Para poder trabajar con archivos CSV, leo el archivo y genero objetos de la clase columna por cada iteración que tiene como atributos la etiqueta, con las columnas instancio una Tabla.
    - Al iterar sobre el CSV, detecta el tipo de dato sobre el que está iterando para definir si la columna es ColumnaNumerica, ColumnaBool o ColumnaString. Lo detecta viendo si comienza con comillas, o de forma numérica.
    - Guardando los elementos en las columnas definidas- que están separados por coma o por otro elemento especificado por el usuario.
    - Si el usuario indica que tiene encabezado, la primera línea del CSV va a definir el atributo colLabel, si no lo indica va a ser un índice numérico de enteros inicializado en 0.
    - Si el usuario indica que tiene índice, el primer elemento de cada línea del CSV va a definir el atributo rowLabel, si no lo indica va a ser un índice numérico de enteros inicializado en 0.
    - Si hay valores faltantes en alguna instancia, se le asignará el valor *“null”.*
        - El método cargarCSV(LectorCSV archivo) devuelve una instancia de Tabla con los datos del archivo CSV, y las rowLabels y colLabels serán índices numericos inicializados en 0 de forma ascendente.
        - El método cargarCSV(LectorCSV archivo, Boolean encabezado , Boolean indice) devuelve una instancia de Tabla con los datos del archivo CSV, si encabezado == True las colLabels son definidas por la primera línea del CSV, y si índice == True las rowLabels también son las indicadas por el archivo CSV. Si los Boolean son False, se generará la etiqueta numérica.
        - Si archivo no es una instancia de LectorCSV se mostrará un mensaje con el error indicado.

- Como usuario, quiero visualizar en formato de texto la información de la tabla, para poder ver la información de mi archivo.
    - Para visualizar implementaremos el método toString().
        - **DoD:**
            - Cada fila se muestra en una línea separada y los valores de las columnas están alineados
            - Los valores de las columnas de cada fila están separados por “|”
            - Las etiquetas de las filas y las columnas se incluyen en la

visualización de la tabla si no son numéricas.

- Como usuario, quiero hacer una copia de una tabla para modificarla sin perder el original.
    - Itero sobre la tabla original y copio todos sus datos, y los guardo en una nueva instancia de Tabla independiente de la original.
        - **DoD:**
            - El método copiarTabla() devuelve una instancia de tabla nueva con los mismos datos que la tabla original.
            - Los cambios en la nueva tabla no afectan a la tabla original.
            - Si tablaOriginal está vacía, genero una nueva tabla vacía.

- Como usuario, quiero poder modificar el valor de una celda especifica para actualizar el valor.
    - Accediendo a un celda usando las etiquetas de fila y columna, puedo asignarle un nuevo valor y modificar la tabla, validando que el tipo de dato sea el mismo que el de la columna. Si no pasa la validación, se debe indicar al usuario que no es posible modificar el nuevo valor porque el tipo de dato es incorrecto.
        - **DoD:**
            - El método cambiarValor(Etiqueta etiquetaColumna, Etiqueta etiquetaFila, Object nuevoValor) modifica el valor que está en la tabla original en la posición indicada por nuevoValor.
            - Si el tipo de dato de nuevoValor no coincide con el tipo de dato de la columna, se mostrará un mensaje con el error indicado.
- Como usuario quiero agregar columnas a tablas existentes para sumar información sin crear nuevas tablas
    - Voy a poder agregarle una nueva columna a la tabla utilizando un método agregarColumna, que implemente el método ".add" de la colección List<>, ya sea un objeto columna o una lista nativa de java (en este caso, hacemos que la lista sea del tipo de dato columna). Para poder realizar esto, es necesario realizar una validación de que la columna sea del mismo tamaño que el de las columnas de la tabla.
    - Si la columna tiene definido un atributo etiqueta se actualiza el atributo colLabels de la tabla. Si no, se define que la etiqueta sea numérica.
    - Al agregar columnas a la tabla el usuario puede elegir en qué posición se ubica a partir de la etiqueta de la columna original.
        - **DoD:**
            - El método agregarColumna(List<E> columnaNueva) instancia una columna con los datos de columnaNueva y la agrega a la tabla, y actualiza colLabels.
            - Si columnaNueva no es del mismo tamaño que las columnas de la tabla, se mostrará un mensaje de error pertinente.

- Como usuario quiero eliminar una columna, para poder ver mi tabla con los datos que me interesan.
    - Para eliminar una columna, suprimo la columna a partir de su etiqueta, y actualizo el atributo *colLabels* de la tabla.
    - Al visualizar una tabla, el usuario puede definir los tamaños máximos configurables y ver la cantidad de filas y columnas que quiera, como también la cantidad de caracteres por celda. Por defecto, se mostrarán las primeras 10 filas y 10 columnas, y 20 caracteres por celda.
        - **DoD:**
            - El método eliminarColumna(Etiqueta etiquetaColumna) elimina la columna indicada.
            - El atributo colLabels está correctamente actualizado, sin la etiqueta de la columna que borramos.
            - Las columnas que se encontraban a la derecha de la columna que borramos, se movieron un espacio a la izquierda para estar acorde a la actualización de colLabels.
- Como usuario, quiero poder eliminar filas para descartar la información que no interesa.
    - Para eliminar una fila, suprimo todos elementos de las columnas que se encuentren en el lugar de la etiqueta de fila, iterando sobre todas las columnas y accediendo al valor indicado con la etiqueta de la fila.
    - Actualizo el atributo *rowLabels* de la tabla
        - El método eliminarFila(Etiqueta etiquetaFila) elimina la fila indicada.
        - El atributo rowLabels está correctamente actualizado, moviendo un lugar hacia arriba todas las filas y sus etiquetas que se encontraban por debajo de la fila borrada.

- Como usuario, quiero poder seleccionar un fragmento de la tabla para mostrar o manipular solo una parte específica de los datos originales sin alterar la tabla principal.
    - Para generar una vista, referencio la sección de la estructura a partir de las etiquetas pedidas por el usuario.
        - El método seleccionarSubtabla(Etiqueta[] etiquetasColumnas, Etiqueta[] etiquetasFilas) selecciona una porción de la tabla solo con las columnas y etiquetas elegidas por el usuario.
        - Si alguna etiqueta no existe, se debe indicar un mensaje con el error pertinente.

- Como usuario, quiero poder seleccionar parte de mi tabla a partir de alguna condición que cumplan los valores dentro de las celdas para poder filtrar la información.
    - Para filtrar, accedo a los valores de las celdas según las etiquetas que ponga el usuario y con el valor lo comparo a la condición pedida. Así, genero una vista con los datos que cumplan dicha condición.
    - Para filtrar, utilizo las etiquetas de columnas y filas para acceder a los valores de las celdas y mediante operadores lógicos se establecen las condiciones específicas para comparar dichos valores y generar una vista filtrada que

incluye solo los datos que cumplen con esos criterios.
        - El método mayorQue()

● Como usuario, quiero concatenar dos tablas en una para tener más datos.
    - Para concatenar tablas, creo un método que recorra las etiquetas de las columnas de cada tabla y los tipos de datos almacenados en las celdas de cada tabla, y una vez validado que son iguales, una las columnas con igual etiqueta.
    - El atributo rowLabel de la tabla resultante se resetea a un rango numérico ascendente comenzando en 0.
    - Si quiero concatenar dos tablas, tengo que verificar que las columnas de ambas estructuras coincidan, tanto en cantidad de columnas como también orden, tipo de datos y etiquetas asociadas. Si no hacen, informo al usuario del problema.
        - El método concatenarTabla(Tabla nuevaTabla) devuelve una nueva tabla con la tabla que se le aplica el método concatenada con nuevaTabla. La instancia rowLabels en la nueva tabla es un rango numérico inicializado en 0.
        - Si las columnas de nuevaTabla no son del mismo tamaño, se mostrará un mensaje con el error indicado.
        - Si las colLabels de nuevaTabla no coinciden, se mostrará un mensaje con el error indicado.
        - Si nuevaTabla es vacía, devuelve la tabla original.
        - Si se aplica el método a una tabla vacía, devuelve nuevaTabla.

● Como usuario, quiero ordenar las tablas según alguna condición ascendente o descendente de los valores de una columna o más, para revisar mejor los datos.
    - Para ordenar, recorro todos los valores de las celdas de la columna o columnas pedidas por el usuario y ordeno las filas según la condición (orden alfabético, mayor/menor) de dichos valores.
    - Al ordenar las tablas, si el usuario elige más de una columna, se toma el orden ingresado de las columnas como precedencia para ordenar las filas.
    - Si la columna pedida es de tipo ColumnaNumerica ordeno de mayor a menor
    - Si la columna pedida es de tipo ColumnaString ordeno alfabeticamente
    - Si la columna pedida es de tipo ColumnaBool pongo True primero
        - El método ordenarFilas(Columna columna) verifica el tipo de dato de columna y ordena de manera acorde al tipo de dato. Si columna no existe en la tabla, se mostrará un mensaje con el error indicado.
        - El método ordenarFilas(Columna[] columnas) ordena dando prioridad al orden ingresado de columnas, y en caso de que el valor de la primera columna sea el mismo entre filas, se decide con la siguiente columna. Si una o más columnas no existen en

la tabla, se mostrará un mensaje con el error indicado.

● Como usuario, quiero dividir las filas en diferentes grupos según una o más columnas para poder analizar los datos particulares de esas columnas.

- Para agrupar, creo una copia independiente de la tabla y luego la recorro y elimino las columnas que no le interesan al usuario. Con los valores de las columnas que interesan, realizo una operación de sumarización estadística. Devuelvo una tabla con el nombre del grupo y los resultados de las operaciones que dependerán de la cantidad de columnas agrupadas.

● Como usuario, quiero poder rellenar los valores vacíos para que no me afecten los análisis.

- Recorrer las columnas bajo una condición donde, si el valor del dato es nulo, lo reemplace por un valor predeterminado.
  - El método imputacion(Columna columna, String valor) modifica todos los valores N/A por valor. Si la columna no es del tipo ColumnaString, se mostrará un mensaje con el error indicado.
  - El método imputacion(Columna columna, Integer valor) modifica todos los valores N/A por valor. Si la columna no es del tipo ColumnaNum, se mostrará un mensaje con el error indicado.
  - El método imputacion(Columna columna, Boolean valor) modificó todos los valores N/A por valor. Si la columna no es del tipo ColumnaBool, se mostrará un mensaje con el error indicado.

● Como usuario, quiero seleccionar aleatoriamente una cantidad de filas de la tabla para poder analizarlas de forma separada.

- A partir de un porcentaje que indique el usuario, calculo el número de filas que voy a separar operando ese porcentaje con la longitud de las columnas, y luego genero una lista de enteros de tamaño de esa operación y que tome valores aleatorios que vayan del 0 a la longitud de la columna. Luego, creo una selección de la tabla con las filas que tengan los índices de la lista recién generada.
  - El método muestreo(Integer porcentaje) devuelve una tabla con una cantidad de filas igual al porcentaje de la cantidad de filas de la tabla, y las filas son todas aleatorias de la tabla original.
  - Si porcentaje no es un valor entre 0 y 100, se mostrará un mensaje con el error pertinente.
  - Si porcentaje es 0, devuelvo una lista vacía.
  - Si porcentaje es 100, devuelvo la tabla original.

Requerimientos no funcionales:

- El lenguaje de programación debe ser Java 8 o superior para garantizar la compatibilidad con la mayoría de las aplicaciones Java modernas.
- La librería debería contemplar todas las situaciones de error y tratarlas con excepciones para notificar correctamente el problema. Las excepciones deben ser fáciles de comprender y localizar.
- Se documentará la interfaz pública con la herramienta Javadocs.
- La interfaz de usuario de la librería debe ser intuitiva, con nombres de métodos y clases descriptivos que reflejen claramente su funcionalidad.
- La documentación debe incluir tutoriales para las operaciones más comunes y ejemplos de código para cada función proporcionada por la librería.
- La librería es eficiente en términos de uso de memoria y procesamiento sin perder velocidad ni requerir demasiados recursos al sistema.
- La librería debe estar protegida contra posibles vulnerabilidades de seguridad, como accesos no autorizados a los datos.
- La librería podrá ser usada en cualquier momento del día una vez desplegada.
- El código debe seguir los principios de diseño y patrones de codificación estándar de Java para garantizar su mantenibilidad a largo plazo.

.