Programación en Entorno de Datos: Memoria

Sergio Soler Rocha
Universidad Nacional de Educación a Distancia

1. Enunciado

A lo largo de los años, los profesores de diversas asignaturas que forman parte de un mismo departamento han estado recopilando las calificaciones de todos los exámenes realizados por sus estudiantes. Las calificaciones de cada asignatura se encuentran en ficheros .csv independientes con la siguiente información:

La primera línea contiene el nombre de la asignatura. El resto de líneas del fichero contienen información sobre cada examen de esta asignatura que se ha realizado, con los siguientes campos separados por comas:

- Curso académico en el que se realizó el examen con el formato: yyyy/yyyy+1. Por ejemplo 2021/2022.
- Convocatoria en la que se realizó el examen. Tiene uno de los siguientes valores: febrero, junio, septiembre.
- Un identificador único para cada estudiante.
- Calificación obtenida en el examen en forma de un valor real entre 0 y 10 con una precisión de una única cifra decimal. Por ejemplo: 3.8, 9.6, 8.0, etc.

En el último Consejo de Departamento se comentó la necesidad de analizar toda esta información para obtener información de interés. Y es aquí donde entramos nosotros como expertos en tratamiento de datos.

2. Trabajo a realizar

2.1. Ejercicio 1

Para cargar los datos creamos la función `loadData(file_paths)` siendo `file_paths` una lista que contiene rutas de archivos CSV.

- Primero define una lista vacía `dfs` que se utilizará para almacenar los DataFrames temporales.
- Define los nombres de las columnas de los archivos CSV: 'Curso académico', 'Convocatoria', 'Estudiante' y 'Calificación obtenida'.
- Itera sobre cada ruta de archivo en `file_paths` verificando si el nombre del archivo comienza con 'Asignatura' y termina con '.csv'. Si cumple con esa condición, lee el archivo CSV usando Pandas con `pd.read_csv()`. Define los nombres de las columnas utilizando `names=column_names` y salta la primera fila de cada archivo usando `skiprows=1`. Añade una nueva columna al DataFrame llamada 'Asignatura', donde se registra el nombre del archivo CSV(eliminando la extensión '.csv' para identificar la asignatura a la que pertenecen los datos. Agrega este DataFrame a la lista `dfs`.
- En la línea `dtype='Calificación obtenida': 'float32'`, especificamos que la columna 'Calificación obtenida' se debe almacenar como números de punto flotante de 32 bits (float32), en lugar del valor por defecto float64. Esto reduce el uso de memoria, ya que los valores pueden ser representados con una precisión de 32 bits.

- En la línea `df['Convocatoria'] = df['Convocatoria'].astype('category')`, convertimos la columna 'Convocatoria' a tipo categórico. Este cambio optimiza el uso de memoria, ya que Pandas almacena las categorías de manera más eficiente que una columna de texto, especialmente debido a que solo hay tres valores únicos en esa columna.
- Después de procesar todos los archivos, utiliza `pd.concat()` para combinar todos los DataFrames almacenados en dfs en un solo DataFrame grande llamado `combined_df`.
- Devuelve el DataFrame combinado que contiene todos los datos de los archivos CSV proporcionados en `file_paths`.

Una vez definida la función `loadData(file_paths)`, creamos una lista de rutas utilizando la librería 'os' y se la proporcionamos a la función. Esto nos permite cargar los datos utilizando dicha función. A continuación mostramos una imagen del procedimiento:

```
def loadData(file_paths):
    dfs = []
    column_names = ['Curso académico', 'Convocatoria', 'Estudiante', 'Calificación obtenida']

    for path in file_paths:
        file_name = os.path.basename(path)
        if file_name.startswith('Asignatura') and file_name.endswith('.csv'):
            df = pd.read_csv(path, names=column_names, skiprows=1, dtype={'Calificación obtenida': 'float32'})
            df['Asignatura'] = file_name[:-4]

            # Convertir la columna 'Convocatoria' a categórica
            df['Convocatoria'] = df['Convocatoria'].astype('category')

            dfs.append(df)

    combined_df = pd.concat(dfs, ignore_index=True)
    return combined_df

# Directorio donde se encuentran los archivos
directory = 'Data'
file_paths = [os.path.join(directory, file) for file in os.listdir(directory)]

df = loadData(file_paths)
```

Figura 1: Implementación del ejercicio 1

2.2. Ejercicio 2

Para llevar a cabo el ejercicio 2, primero definiremos una función que nos resultará útil. Esta función devolverá una tupla con la cantidad de suspensos, aprobados, notables y sobresalientes de un DataFrame dado. La función es la siguiente:

```
def calculate_counts_score(df):
    # Conteos de cada puntuación
    suspensos = df[df['Calificación obtenida'] < 5.0].values.shape[0]
    aprobados = df[(df['Calificación obtenida'] >= 5.0) & (df['Calificación obtenida'] < 7.0)].values.shape[0]
    notables = df[(df['Calificación obtenida'] >= 7.0) & (df['Calificación obtenida'] < 9.0)].values.shape[0]
    sobresalientes = df[df['Calificación obtenida'] >= 9.0].values.shape[0]

    return suspensos, aprobados, notables, sobresalientes
```

Figura 2: Función que nos devuelve el número de suspensos, aprobados, notables y sobresalientes

Para el caso de suspensos `df['Calificación obtenida'] < 5.0` crea una máscara booleana que es True para cada fila donde la columna 'Calificación obtenida' tiene un valor menor a 5.0 y False en caso contrario. `df[df['Calificación obtenida'] < 5.0` Aplica la máscara booleana al DataFrame, devolviendo solo las filas donde la condición es True. El método `.values` convierte el DataFrame resultante a un array de valores NumPy. `.shape[0]` devuelve la longitud (número de filas) del array NumPy resultante. El mismo procedimiento se extrapola a aprobados, notables y sobresalientes.

Una vez creada la función *calculate_counts_score*, las funciones pedidas por el enunciado se crean de la siguiente forma:

```python
def tableAll(df, a):
    df_asignatura = df[df['Asignatura'] == a]
    return calculate_counts_score(df_asignatura)

def tableYear(df, a, y):
    df_year = df[(df['Asignatura'] == a) & (df['Curso académico'] == y)]
    return calculate_counts_score(df_year)

def tableYearConv(df, a, y, c):
    df_year_conv = df[(df['Asignatura'] == a) & (df['Curso académico'] == y) & (df['Convocatoria'] == c)]
    return calculate_counts_score(df_year_conv)

def tableConv(df, a, c):
    df_conv = df[(df['Asignatura'] == a) & (df['Convocatoria'] == c)]
    return calculate_counts_score(df_conv)
```

Figura 3: Funciones del ejercicio 2

En todas las funciones creamos máscaras booleanas según lo que se nos pida en cada una de ellas.

2.3. Ejercicio 3

En el ejercicio 3 queremos conocer es la media del número de veces que los estudiantes se presentan a una misma asignatura, para eso creamos la siguiente función:

```python
def meansExams(df,a):
    mean_exams = df[df['Asignatura']== a].groupby(['Estudiante']).size().mean()
    return np.round(mean_exams, decimals=2)

meansExams(df, a='Asignatura01')
```

Figura 4: Función que nos devuelve la media con la que un alumno se presenta a una asignatura dada

`df[df['Asignatura']== a]` filtra el DataFrame df para seleccionar solo las filas donde la columna 'Asignatura' es igual a la variable a (asignatura que elijamos). `.groupby(['Estudiante']).size()` agrupa el DataFrame resultante por la columna 'Estudiante' y se calcula el número de ocurrencias (número de veces que aparece dicho estudiante) de cada grupo. Finalmente `.mean()` calcula la media de los tamaños de los grupos resultantes (las veces que cada alumno se presenta a la asignatura).

2.4. Ejercicio 4

En el ejercicio 4 hay que programar la función *allSubjects(df)*, que recibe el DataFrame df de Pandas con los datos cargados y devuelve una lista con los identificadores de aquellos estudiantes que han aprobado los exámenes de todas las asignaturas del departamento.

Si asumimos que, una vez que un alumno aprueba una asignatura, no se presenta nuevamente a la misma (por ejemplo, para mejorar la nota), podríamos realizar un filtrado por aprobados. Luego, agruparíamos por alumnos, contaríamos las ocurrencias y seleccionaríamos a aquellos que aparezcan tantas veces como el número de asignaturas que tiene el departamento. El problema con esta suposición es que no tiene en cuenta la posibilidad de que un alumno pueda presentarse más de una vez a una asignatura aprobada. Por lo tanto, si este fuera el caso, el método mencionado anteriormente dejaría de ser válido. En este escenario, un alumno podría aprobar más exámenes de los que tiene el número total de asignaturas, o no aprobar todas ellas, pero al presentarse varias veces a una misma asignatura podría alcanzar la cantidad deseada. Por tanto, para evitar esos posibles problemas usamos el siguiente método:

```python
def allSubjects(df):
    # Filtrar por aprobados
    df_aprobados= df[df['Calificación obtenida'] >= 5]

    # Agrupar por Estudiantes y Asignaturas (se cuenta el número de veces que se presenta a la asignatura aprobada)
    aprobados_grouped = df_aprobados.groupby(['Estudiante', 'Asignatura']).size().reset_index(name = 'Numero de veces peresentado')

    # Número de asignaturas total que hay en el departamento
    total_subjects = df['Asignatura'].nunique()

    # Reagrupar el DataFrame aprobados_grouped y contar el número de veces que aparece
    aprobados_grouped_subjects = aprobados_grouped.groupby('Estudiante').size().reset_index(name = 'Asignaturas aprobadas')

    # Filtrar por los alumnos que aparezcan tantas veces como asignaturas hay en el departamento
    students_aprobados_all = aprobados_grouped_subjects[aprobados_grouped_subjects['Asignaturas aprobadas'] == total_subjects]['Estudiante'].tolist()

    return students_aprobados_all
```

Figura 5: Definición de la función *allSubjects*

En primer lugar, filtramos por aprobados. A continuación, agrupamos por 'Estudiante' y por 'Asignatura' y al contarlos, obtenemos el número de veces que un alumno se presenta a una asignatura aprobada. Con *reset_index*, transformamos la Serie a DataFrame para poder seguir agrupando.

En *total_subjects*, calculamos el número total de asignaturas que hay en el departamento. En *aprobados_grouped_subjects*, agrupamos el DataFrame obtenido en el paso anterior por 'Estudiante'; ahora, al contarlos, podemos saber el número de asignaturas que ha aprobado cada alumno de manera individual, sin preocuparnos de que se haya presentado más de una vez aun habiendo aprobado.

En el último paso, filtramos por los alumnos que han llegado al número total de asignaturas del departamento.

2.5. Ejercicio 5

Hay que programar la función *allSubjectsFirstYear(df)*, que recibe el DataFrame df de Pandas con los datos cargados y devuelve una lista con los identificadores de aquellos estudiantes que han aprobado los exámenes de todas las asignaturas del departamento el primer año que se presentaron al examen de dicha asignatura. Teniendo en cuenta que hay tres convocatorias de exámenes, habrá que considerar a aquellos estudiantes que aprobaron en cualquiera de ellas, siempre que no se hubieran presentado a examen un año anterior.

El código para crear la función *allSubjectsFirstYear(df)* es el siguiente:

```python
def allSubjectsFirstYear(df):
    # Agrupar por Estudiante y Asignatura contando el número de años académicos únicos para cada combinación de 'Estudiante' y 'Asignatura'
    years_per_subject = df.groupby(['Estudiante', 'Asignatura'])['Curso académico'].nunique()

    # Filtrar por el número de veces que se ha presentando una vez
    students_all_subjects_one_year = years_per_subject[years_per_subject == 1]

    # Crear un DataFrame de la Serie anterior y agrupor por Estudiante para contarlos
    students_one_year_total = students_all_subjects_one_year.reset_index().groupby('Estudiante').size()

    # Número de asignaturas total que hay en el departamento
    total_subjects = df['Asignatura'].nunique()

    # Filtrar por los que coincida con el número total de asignaturas del departamento
    students_one_year_total_all = students_one_year_total[students_one_year_total == total_subjects].index.tolist()

    # Asegurarse de que hayan aprobado
    students_one_year_total_all_passed = allSubjects(df[df['Estudiante'].isin(students_one_year_total_all)])


    return students_one_year_total_all_passed
```

Figura 6: Definición de la función *allSubjectsFirstYear*

1. Agrupamos por Estudiante y Asignatura con *groupby* y contamos el número de años académicos únicos para cada combinación de 'Estudiante' y 'Asignatura'. En otras palabras, obtenemos el número de cursos que un alumno ha hecho un examen de una asignatura dada.

2. Filtramos a los estudiantes que solo se han presentado una vez (solo un año). Solo nos quedamos las combinaciones 'Estudiante' y 'Asignatura' donde el número de años académicos únicos es igual a 1.

3. Contamos a los estudiantes con una asignatura por año. Utilizamos *reset_index* para convertir la Serie en un DataFrame y luego agrupamos por 'Estudiante' para contar cuántos estudiantes tienen una asignatura por año.

4. Contamos el número total de asignaturas del departamento

5. Filtramos a los 'Estudiante' que tienen una asignatura por año y su número total coincide con el número total de asignaturas del departamento.

6. Ya hemos identificado a los estudiantes que se han presentado únicamente durante un curso a todas las asignaturas del departamento. Sin embargo, este criterio por sí solo no es suficiente. Existe la posibilidad de que un alumno se haya presentado a todas las asignaturas durante un único año, pero no haya aprobado alguna de ellas y haya decidido abandonar (aunque sea algo prácticamente improbable, es importante tenerlo en cuenta). Por esta razón, aplicaremos la función allSubjects(df) a estos estudiantes para asegurarnos de que han aprobado todas las asignaturas. Con `df['Estudiante'].isin(students_one_year_total_all)` Pasamos la lista del punto anterior y nos devuelve una máscara booleana que nos servirá paa filtrar por esos alumnos y ver si han aprobado todas las asignaturas.

2.6. Ejercicio 6

Hay que crear la función *academicRecord(df)*, que recibe el DataFrame df de Pandas con los datos cargados y devuelve otro DataFrame en el que en cada fila se encuentra el identificador de un estudiante asociado a la calificación que tiene en cada asignatura (se entiende que será la del último examen que haya realizado de dicha asignatura).

Para la resolución del ejercicio definimos la siguiente función:

```
def academicRecord(df):
    # Ordenar el DataFrame por Estudiante, Asignatura y Curso académico
    df_sorted = df.sort_values(by=['Estudiante', 'Asignatura', 'Curso académico'])

    # Obtener el último examen realizado por cada estudiante en cada asignatura
    last_exam_grades = df_sorted.groupby(['Estudiante', 'Asignatura'])['Calificación obtenida'].last().reset_index()

    # Crear un nuevo DataFrame con los expedientes académicos
    academic_record = last_exam_grades.pivot(index='Estudiante', columns='Asignatura', values='Calificación obtenida')

    return academic_record
```

Figura 7: Definición de la función *academicRecord*

1. Ordenamos el DataFrame original por las columnas 'Estudiante', 'Asignatura' y 'Curso académico' utilizando el método *sort_values*. Esto es para asegurar que los datos estén ordenados de manera que el examen más reciente de cada estudiante para cada asignatura sea el último en aparecer en el DataFrame.

2. Obtenemos el último examen realizado, para eso utilizamos *groupby* para agrupar el DataFrame ordenado por 'Estudiante' y 'Asignatura'. Luego, aplicamos el método last() para obtener el último examen realizado por cada estudiante en cada asignatura.

3. Utilizamos el método *.pivot()* para reorganizar el DataFrame `last_exam_grades`. Se utiliza 'Estudiante' como el índice, 'Asignatura' como columnas y 'Calificación obtenida' como los valores de la tabla resultante. Esto da como resultado un nuevo DataFrame llamado *academic_record* que representa los expedientes académicos de los estudiantes, donde las filas son estudiantes, las columnas son asignaturas y los valores son las calificaciones obtenidas en el último examen de cada asignatura.