22.503 - Programación para la ciencia de datos PEC 4

En este documento encontraréis el enunciado de la cuarta actividad de evaluación continua (PEC) de la asignatura. Esta PEC consistirá en 6 ejercicios a resolver que engloban muchos de los conceptos que hemos visto a lo largo del curso.

Es recomendable hacer los ejercicios en orden, aunque es posible saltárselo en algunas ocasiones. En particular, es importante hacer primero el 1, ya que se trata de funciones que se usarán en el resto de ejercicios. También se recomienda dejar el 6 para el final ya que es un problema más abierto que los anteriores y resultará más sencillo si ya se ha trabajado el resto de la PEC.

En el aula podréis encontrar, además, un vídeo en el que se repasan cuestiones prácticas relacionadas con la programación fuera del entorno de los *notebooks*. Recomendamos encarecidamente su visionado antes de empezar la PEC.

A diferencia de otras PECs, veréis que esta incluye un conjunto de tests que permiten comprobar que las funciones que se piden hacen lo que se espera de ellas. Podéis encontrar más información sobre cómo ejecutar los tests correctamente al final de esta PEC y en el vídeo del aula.

También se incluye información, tanto al final del documento como en el vídeo, sobre *pylint*, la herramienta que usaremos para comprobar que se sigue la guía de estilo PEP8.

Contexto

Trabajáis para una empresa que proporciona soluciones “Data Science” para todo tipo de proyectos. El equipo directivo quiere centrarse en los próximos años en el incipiente mundo de los e-sports y ha decidido empezar con uno que no les suena tan extraño, el FIFA.

Nos han pedido que diseñemos una serie de herramientas que permitan trabajar con este tipo de datos a largo plazo, además de realizar unos análisis preliminares de los mismos. Para ello, nos han proporcionado un conjunto de datos que incluye toda la información sobre los jugadores y las jugadoras incluídas en el videojuego desde el año 2016 hasta el 2022.

Ejercicio 1: lectura y preprocesado

El primer paso es crear las funciones necesarias para leer y procesar los datos de los que disponemos. Para ello, se pide implementar:

a) Una función que nos permita leer un fichero de jugadores/as y añadir información sobre el género y año al que corresponden en dos nuevas columnas: *gender* y *year*. La función debe devolver el *dataframe* resultante.

read_add_year_gender(filepath: str, gender: str, year: int) -> pd.DataFrame

- filepath: string con la ruta del fichero que queremos leer
- gender: puede ser 'M' o 'F'
- year: año al que corresponden los datos en formato XXXX (por ejemplo, 2020)

b) Una función que cree un único *dataframe* con los datos de todos los jugadores y jugadoras de un mismo año. El *dataframe* debería contener información sobre el género y año al que se corresponden los datos de cada futbolista.

join_male_female(path: str, year: int) -> pd.DataFrame

- path: ruta hasta la carpeta en la que están los datos
- year: año que se quiere leer en formato XXXX

c) Una función que lea la información correspondiente a futbolistas de ambos géneros durante varios años, devolviendo un único dataframe. Este debería contener, además, información sobre el género y año al que se corresponden los datos de cada futbolista.

join_datasets_year(path: str, years: list) -> pd.DataFrame

- path: ruta hasta la carpeta en la que están los datos
- years: lista de años que se quieren incluir en formato [XXXX,...]

Ejercicio 2: estadística básica

Queremos crear una serie de funciones que nos permitan obtener ciertas estadísticas básicas del conjunto de futbolistas. Se nos pide:

a) Una función que, dado el nombre de una columna numérica, nos devuelva la(s) fila(s) en las que su valor es máximo. Además, la función recibirá como argumento una lista de columnas. Las filas que devuelva la función deben contener únicamente estas columnas.

find_max_col(df: pd.DataFrame, filter_col: str, cols_to_return: list) -> pd.DataFrame

- df: *dataframe* con los datos
- filter_col: nombre de la columna de la que queremos averiguar el máximo
- cols_to_return: lista de columnas que hay que devolver

b) Una función que nos permita realizar filtros avanzados. La función recibirá una query en forma de tupla. El primer elemento será una lista de columnas sobre las que queremos filtrar. El segundo elemento será la lista de valores que queremos usar en el filtro. Si la columna es categórica, el valor será un string. Si es numérica, será una tupla con el valor mínimo y máximo (ambos deben ser incluídos en el filtro).

Por ejemplo, la query

([“league_name”, “weight_kg”], [“English Premier League”, (60, 70)])

Nos permitiría encontrar los jugadores y las jugadoras de la liga “English Premier League” con un peso entre 60 y 70 kilos (incluídos).

Al igual que en el ejercicio 2a, se incluye un argumento que nos indica qué columnas queremos que nos devuelva la función.

find_rows_query(df: pd.DataFrame, query: tuple, cols_to_return: list) -> pd.DataFrame

- df: dataframe con los datos
- query: tupla con los datos de la query
- cols_to_return: lista de columnas que hay que devolver

c) Partiendo de todo el conjunto de datos proporcionado (desde el año 2016 hasta el 2022, ambos incluídos), mostrad por pantalla el “short_name”, “year”, “age”, “overall” y “potential” de:

- Los jugadores de nacionalidad belga menores de 25 años con máximo *potential* en el fútbol masculino.
- Las porteras mayores de 28 años con *overall* superior a 85 en el fútbol femenino.

Ejercicio 3: BMI

Las/os deportistas profesionales siguen un riguroso control de su dieta y realizan mayor esfuerzo físico que la población general. Queremos estudiar si esto resulta en un índice de masa corporal (BMI por sus siglas en Inglés) distinto al de la población general.

a) En primer lugar, debéis crear una función que, dado un dataframe con los datos, un género y un año, devuelva un dataframe que incluya una columna con el índice de masa corporal (BMI) de cada futbolista de dicho género y año. La función también recibirá como argumento una lista de columnas. Las filas que devuelva la función deben contener estas columnas, además de la nueva columna BMI.

   El BMI de una persona viene dado por:

$$BMI = \frac{peso}{altura * altura}$$

   Nota: El peso debe de darse en kgs y la altura en metros.

   calculate_BMI(df: pd.DataFrame, gender: str, year: int, cols_to_return: list) -> pd.DataFrame

   - df: dataframe con los datos
   - gender: género que queremos estudiar
   - year: año que nos interesa en formato XXXX (por ejemplo 2020)
   - cols_to_return: lista de columnas que hay que devolver (sin incluir BMI)

b) Mostrad un gráfico con el BMI máximo por país. Filtrad por género masculino y año 2022. La información sobre el país en el que juega cada futbolista (no confundir con la nacionalidad del jugador) se puede extraer de la columna *club_flag_url*. Por ejemplo, la url https://cdn.sofifa.net/flags/fr.png corresponde a Francia (fr). No es necesario obtener el nombre completo del país, con la abreviatura es suficiente.

   Teniendo en cuenta la siguiente clasificación:

Category	BMI
underweight	< 18.5
normal weight	[18.5, 25)
overweight	[25, 30)
obese	≥ 30

¿Os resultan llamativos los valores obtenidos? ¿Por qué?

c) Comparad en una gráfica el BMI del conjunto de futbolistas que deseéis con el de la población española. Podéis descargar los datos de la página del INE:

https://www.ine.es/jaxiPx/Tabla.htm?path=/t15/p420/a2019/p03/l0/&file=01001.px&L=1

Notad que los datos del INE están separados entre hombres y mujeres. Notad también que el BMI varía con la edad. Cualquier comparación que hagáis debe de ser entre datos lo más cercanos posible. No obstante, no es necesario que consideréis información sobre la nacionalidad o el país en el que juegan.

Indicad claramente en la leyenda de la gráfica qué datos habéis elegido para hacer la comparación.

Ejercicio 4: diccionarios

Queremos poder seguir la evolución de los/las jugadores/as que aparecen en distintos años en nuestro conjunto de datos. En un primer paso crearemos una función que nos permitirá seleccionar con qué jugadores/as (identificados con la columna “sofifa_id”) queremos trabajar y qué columnas queremos mostrar:

a) Cread una función que, dado un *dataframe* con los datos, una lista de identificadores y una lista de columnas:

players_dict(df: pd.DataFrame, ids: list, cols:list) -> dict

- df: *dataframe* con los datos
- ids: lista de identificadores “sofifa_id”
- cols: lista de columnas de la que hay que guardar la información

devuelva un diccionario donde las claves sean los valores de la columna “sofifa_id” contenidos en la lista “ids” y los valores asociados sean diccionarios con la información correspondiente. Las claves de cada uno de estos diccionarios serán los nombres de las columnas incluidas en “col” y sus valores serán la información de todos los años disponibles en el *dataframe* para cada futbolista.

Nota 1: la columna “sofifa_id debe usarse como clave siempre y no se incluirá en la lista cols en ningún caso.

El diccionario anterior contiene valores redundantes. Por ejemplo, la columna “short_name” contendrá tantas repeticiones como años esté esa persona en los datos. En otros casos, como la columna “player_positions”, el diccionario contendrá información parcialmente repetida pero con la posibilidad de incluir nuevos valores. Finalmente, en columnas como “potential”, cada uno de los valores obtenidos representa información relevante.

b) Para eliminar esta información redundante, cread la función:

clean_up_players_dict(player_dict: dict, col_query: list) -> dict

- player_dict: diccionario con el formato de *players_dict*
- col_query: lista de tuplas con detalles sobre la información a simplificar

La función debe recorrer uno a uno los elementos del diccionario y efectuar los cambios indicados por la lista de tuplas col_query. Cada tupla de la lista estará compuesta por dos valores: el nombre de una columna y una cadena de caracteres. Esta cadena puede contener dos posibles valores, los cuales nos indican la operación a realizar sobre la columna/clave del diccionario:

- “one”: debemos quedarnos solo con el primer valor que aparezca
- “del_rep”: primero debemos descomponer la información y después eliminar las repeticiones

La información referente a las columnas que no aparecen en col_query se retornará sin cambios.

Por ejemplo si tuviéramos la query

[("player_positions","del_rep"), ("short_name","one")]

y la entrada de diccionario

41: {'short_name': ['Iniesta', 'Iniesta', 'Iniesta'], 'overall': [88, 88, 87], 'potential': [88, 88, 87], 'player_positions': ['CM', 'CM', 'CM, LM'], 'year': [2016, 2017, 2018]}

Nuestra función deberá retornar:

41: {'short_name': 'Iniesta', 'overall': [88, 88, 87], 'potential': [88, 88, 87], 'player_positions': {'CM', 'LM'}, 'year': [2016, 2017, 2018]}

c) Partiendo del *dataframe* con ambos géneros y los años 2016, 2017 y 2018, mostrad por pantalla:

- El diccionario construido con la función del apartado *4a* con la información de las columnas ["short_name", "overall", “potential”, "player_positions", "year"] y los ids = [226328, 192476, 230566].
- La *query* que pasaríais a la función del apartado *4b* para limpiar este diccionario.
- El diccionario limpio.

Nota: se recomienda utilizar el módulo pprint para mostrar los diccionarios.

Ejercicio 5: evolución

Queremos poder estudiar la evolución de ciertas características. Para ello, se pide:

a) Implementad la función

top_average_column(data: dict, identifier: str, col: str, threshold: int) -> list

- data: diccionario limpio con la información de varios sofifa_id
- identifier: columna/clave que se usará como identificador
- col: nombre de una columna/clave numérica
- threshold: mínimo número de datos necesario

que:

- Para cada sofifa_id, calcule el valor promedio de la característica *col* si tiene información de *threshold* o más años. Si no, se ignora dicho sofifa_id. Si algún elemento de la lista tiene el valor NaN, también se ignora dicho sofifa_id.
- Devuelva una lista de tuplas formadas por tres elementos: valor de la columna identifier; promedio de la característica; y un diccionario compuesto por la clave *year* que contenga la lista de años correspondientes a los valores y la clave *value* con dichos valores.
- La lista debe estar ordenada de mayor a menor según el promedio calculado.

Por ejemplo, si tuviéramos identifier = “short_name” y col = “shooting”, un posible elemento de la lista sería

('L. Schelin', 85.0, {'value': [87.0, 84.0, 84.0], 'year': [2016, 2017, 2018]})

b) Utilizad la función anterior para obtener la evolución de los/as 4 futbolistas con mejor promedio de movement_sprint_speed entre 2016 y 2022 (incluidos). Utilizad “short_name” como identificador y mostrad el resultado por pantalla. Representad gráficamente la evolución obtenida.

Ejercicio 6: La mejor defensa.

En este ejercicio final de la asignatura os pedimos que utilicéis vuestras habilidades como científicos/as de datos para:

a) Formalizar un problema a partir de objetivos generales.
b) Implementar una solución al problema formalizado siguiendo los principios vistos en esta asignatura.
c) Encontrar una solución al problema inicial y comunicar los resultados de manera efectiva.

Lo que sigue es un enunciado con algo de literatura que describe el problema "general". Al final del ejercicio encontraréis un resumen de los puntos más importantes:

Uno de vuestros clientes representa a un misterioso grupo de inversión que ha adquirido un conocido club de fútbol "en horas bajas". El cliente os indica que ha decidido empezar la reconstrucción del equipo por la defensa y que, por tanto, necesita saber qué futbolistas forman la mejor línea defensiva del mundo, el dinero no es obstáculo.

El cliente os deja muy claros los siguientes puntos:

- ***Fuente de Datos:*** *La mejor manera de juzgar la calidad de las/os futbolistas es consultar la base de datos con la que hemos estado trabajando en esta PEC. Concretamente, el cliente está interesado únicamente en los datos del 2022 (female_players_22.csv y players_22.csv).*
- ***Definición de "defensa"****. Una línea defensiva se compone de 4 jugadores/as: 2 defensas centrales (CB según la notación de la columna "player_positions"); 1 defensa lateral derecho (RB); y 1 defensa lateral izquierdo (LB).*
- ***No se deben repetir jugadores/as:*** *Si un jugador/a puede jugar en más de una de estas posiciones se le puede considerar en más de un rol, pero siempre en uno como máximo en cada línea defensiva. Por ejemplo, en el archivo female_players_22.csv se indica que Magdalena Lilly Eriksson puede ocupar las posiciones "LB" y "CB". Esta jugadora podrá, por tanto, optar a ser parte de nuestra "defensa ideal" tanto como lateral derecho como central, pero una alineación que la incluya a la vez en las dos posiciones será considerada errónea.*
- ***Masculino/Femenino/Veteran@s****: El club dispone de un equipo masculino en horas particularmente bajas, y un equipo femenino entre los mejores del continente. Por tanto, el cliente quiere conocer las mejores líneas defensivas masculina y femenina. Además, el cliente sabe de buena tinta que un proyecto*

llamado European VetLeague, donde competirán equipos mixtos de jugadoras/es a partir de 30 años, está a punto de ver la luz. Por ello, quiere conocer también la mejor línea defensiva de jugadores/as de cualquier sexo que tengan 30 años o más.

Al intentar preguntar al cliente qué criterios definen la calidad de una línea defensiva, este gesticula vagamente en dirección de las columnas F a B7 de nuestro dataset abierto en excel y nos indica que debemos elegir las (entre 3 y 5) características más importantes a tener en cuenta para cada posición (LB, RB,CB).

El cliente quiere tener una estimación de qué aportación produce cada posible línea defensiva en los tres criterios siguientes: 1) defensa; 2) control de balón (posesión); y 3) ataque. A ser posible, la defensa y el ataque deberían estar divididos en tres zonas: izquierdo, central y derecho.

Vuestros intentos de conseguir un poco más de concreción de vuestro jefe resultan infructuosos ya que se encuentra muy ocupado contando fajos de billetes que extrae de la bolsa de deporte que el cliente ha dejado. Solamente os indica que este es el proyecto más importante de la empresa hasta la fecha y que vuestro puesto de trabajo depende enteramente de su resultado, por no hablar de la felicidad de los cientos de miles de hinchas del club en cuestión.

Puntos a tener en cuenta:

a) **Formalización del problema:** Lo primero que debéis hacer es determinar qué futbolistas pueden jugar en cada una de las posiciones de interés. Del mismo modo, debéis decidir, para cada posición, qué variables (de las representadas por las columnas del dataset) son más importantes. Os recomendamos elegir un reducido número de candidatos (no más de 50) a ocupar cada una de las 4 posiciones siguiendo algún criterio de vuestra elección.

   Obtenida la lista de candidatos para cada posición, el siguiente paso sería determinar todas las posibles líneas defensivas que se puedan constituir con ellos. Después, habrá que hacer una estimación de la contribución al ataque, posesión y defensa de cada línea en base a vuestro criterio propio. Así, podréis ordenar las líneas de mejor a peor y quedaros con la mejor.

b) **Implementación:** Vuestro entrega debe incluir el código necesario para encontrar la solución al problema implementando los criterios subjetivos que hayáis definido. Recordad que se valorarán los mismos aspectos de modularidad, documentación y eficiencia que en el resto de la PEC.

c) **Presentación de los resultados:** Vuestra entrega debe incluir también un pequeño informe (de no más de 2 paginas, en formato pdf) que explique qué criterios habéis utilizado para modelar cada uno de los aspectos del problema y que presente los resultados obtenidos para las mejores líneas defensivas masculina, femenina y de jugadores/as veteranos/as. Se valorará positivamente la inclusión de algún gráfico que acompañe la explicación.

d) No es necesario tener conocimientos avanzados de fútbol para realizar este ejercicio. Tampoco existe una solución única ni se comparará el resultado con otros. Lo que se valorará es que la idea planteada tenga sentido, que el informe comunique de manera efectiva el trabajo realizado y que la implementación sea correcta.

Tests

La PEC incluye un archivo con un conjunto de tests públicos en tests/test_public.py. Para su ejecución, es necesario tener instalados los módulos unittest y HTMLTestRunner. Este último se puede instalar ejecutando

pip install HTMLTestRunner-rv

Para realizar los tests debéis ejecutar el siguiente comando desde una terminal situada en el directorio en el que se encuentra main.py:

python3 -m tests.test_public

De forma automática se creará una carpeta llamada reports con información sobre la ejecución de los tests en html.

Para corregir un apartado es imprescindible que pase satisfactoriamente todos los tests públicos correspondientes. Os animamos a abrir el archivo y a explorar los test para entender qué se espera que den las funciones. Eso sí, no se permite ninguna modificación del archivo test_public.py. De hecho, para corregir la PEC sustituiremos el archivo con nuestra propia copia, así que no sirve de nada intentar modificar a mano estos tests.

Tened en cuenta que los tests proporcionados no incluyen pruebas para todas las funcionalidades que se piden en esta PEC. A la hora de corregir pasaremos un conjunto de tests privados que comprobarán otros aspectos de las funciones. Esto, además, previene que hagáis funciones pensadas para pasar únicamente los tests públicos.

Por último, os proponemos que creéis vuestros propios tests para comprobar alguna de estas funcionalidades no contenidas en los públicos. Para ello, se incluye el archivo test_custom.py que podéis modificar como deseéis. Aunque podéis crear todos los que queráis, para conseguir la nota máxima en este apartado debéis crear, como mínimo, los siguientes tests:

a) Comprobar que la función 2a da el resultado correcto cuando cols_to_return contiene, como mínimo, dos columnas (por ejemplo, short_name y potential).
b) Comprobar que la función 2a da el resultado correcto si el resultado incluye más de una fila.
c) Comprobar que la función calculate_BMI da el resultado correcto cuando gender = ‘F’
d) Comprobar que la función clean_up_players_dict da el resultado correcto si la *query* contiene la operación “one”.

Importante: el archivo test_public.py necesita importar las funciones que hagáis. Para evitar que tengáis que modificar el archivo (recordad que lo sustituiremos por nuestra copia), se incluye el fichero testing_imports.py. Tendréis que poner ahí todos los *imports* que sean necesarios (en el vídeo se muestra un ejemplo).

Linter

Dado que ya no estamos en el entorno de los Jupyter notebooks, para evaluar si el código sigue la guía de estilo PEP8 utilizaremos el *linter* pylint. Para instalarlo en la máquina virtual simplemente tenéis que ejecutar en una terminal:

```
sudo apt install pylint
```

Después, para ejecutarlo, basta con ir al directorio en el que se encuentren los archivos *.py y ejecutar en una terminal:

```
pylint *.py
```

El resultado será una lista de errores (si los hubiera) junto con una nota numérica sobre 10. La puntuación correspondiente al apartado de estilo en esta PEC se basará en esta nota.

Nota: no se evaluará el estilo de los archivos del directorio test.

Criterios de corrección

Esta PEC se valorará siguiendo los criterios siguientes:

- Funcionalidad (7 puntos): Se valorará que el código implemente correctamente lo que se pide en el enunciado. Para ello, se usará un conjunto de test públicos, privados y se valorarán los resultados gráficos propuestos. **Es imprescindible para corregir un ejercicio que pase todos los test públicos correspondientes.**
    - Tarea 1 (0.75 puntos)
    - Tarea 2 (1.25 puntos)
    - Tarea 3 (1.25 puntos)
    - Tarea 4 (1 punto)
    - Tarea 5 (0.75 puntos)
    - Tarea 6 (2 puntos)
- **Estilo y documentación** (0.75 puntos): Se usará un analizador automático de código, pylint, el cual proporciona una nota sobre 10. Esta será la nota de dicho apartado. Tened en cuenta que el analizador también evalúa la presencia de docstrings.
- **Modularidad** (0.25 puntos): Se valorará la organización del código en módulos temáticos siguiendo algún criterio razonable.
- **Tests** (1 punto): Se propone implementar 4 tests que permitan evaluar la funcionalidad de partes del código que los tests públicos no comprueban. Todos ellos tienen el mismo peso en este apartado.
- **Licencia + Requerimientos** (0.5 puntos): El proyecto deberá incluir la licencia bajo la cual se distribuye el código (podéis elegir la que queráis). Es necesario incluir un fichero de requerimientos que liste únicamente las librerías necesarias para ejecutar el código.
- **README** (0.5 puntos): El proyecto deberá incluir un fichero README en formato markdown que explique cómo instalarlo y ejecutarlo.