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 de un único ejercicio a resolver que engloba muchos de los conceptos que hemos visto a lo largo del curso.

El contexto

La empresa UOC Sound Dynamics os ha contratado como científicos de datos para un proyecto cuya finalidad es crear una aplicación para descubrir artistas musicales. De momento, el proyecto se encuentra en una fase inicial dónde se deben analizar los datos existentes y así ayudar a definir el producto final que más tarde implementarán los desarrolladores. Vuestro trabajo es crear un paquete de Python que lea los datos, los prepare para el análisis, calcule algunas estadísticas básicas y finalmente, cree visualizaciones para comparar diferentes artistas.

Los datos que se utilizaran en el análisis contienen información sobre canciones, sobre los álbumes que las contienen y sobre los artistas que las han creado. A parte de detalles básicos como por ejemplo, el nombre de los artistas o el título de las canciones, los datos más interesantes que serán la base del análisis son las audio features que nos proporciona Spotify. Podéis ver más detalles sobre los datos proporcionados en la sección *Datos*.

Objetivos del proyecto

El Data Lead del proyecto ya ha avanzado el trabajo de especificación y ha diseñado una primera fase que empieza el **17/12/2021** y acaba el **10/01/2022**. Durante esta fase se deben realizar las tareas descritas en la sección *Tareas*. Cada tarea contiene una descripción y un listado de criterios de aceptación que se tienen que cumplir para dar la tarea por finalizada. Además, se han definido algunos aspectos generales a tener en cuenta:

- El paquete será utilizado en futuras fases del proyecto y por lo tanto, la implementación se tiene que hacer en ficheros planos de Python y no en Jupyter Notebooks.
- El código se debe implementar en funciones que sean lo más generales posible, siempre que esto sea posible y tenga sentido. De esta manera, dichas funciones se podrán utilizar en fases posteriores del proyecto.
- El paquete debe incluir el correspondiente código organizado por bloques lógicos (separado en módulos y organizado por funcionalidad), así como documentación

(docstrings) y tests. Además, se deberán incluir los correspondientes archivos de documentación de alto nivel (README), así como los archivos de licencia y dependencias (requirements.txt). Tener un archivo setup.py es opcional, pero sí se valorará positivamente.

Los datos

Los datos de canciones, álbumes y artistas se encuentran en 3 CSVs diferentes dentro del archivo comprimido *data.zip*. Fijaros que los datos están normalizados y en el listado de *tracks* (canciones) se encuentran los identificadores del artista (*artist_id*) y del álbum (*album_id*) para poder relacionarlos.

tracks_norm.csv

artist_id: identificador del artista **album_id**: identificador del álbum **track_id**: identificador de la canción **track_sp_id**: identificador de la canción a Spotify
name: título de la canción **number**: número de canción dentro del álbum **disc_number**: número del álbum (en el caso de álbumes dobles) **popularity**: indicador de popularidad **preview_url**: URL para poder escuchar un trozo de la canción **duration_ms**: duración en ms
audio features de la canción: danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time_signature

albums_norm.csv

artist_id: identificador del artista **album_id**: identificador del álbum **album_sp_id**: identificador del álbum a Spotify
name: título del álbum **popularity**: indicador de popularidad **release_year**: año de publicación **total_tracks**: número de canciones en el álbum

artists_norm.csv

artist_id: identificador del artista **artist_sp_id**: identificador del artista a Spotify
name: nombre del artista **popularity**: indicador de popularidad

followers: número de seguidores a Spotify
total_albums: número de álbumes

Tareas

Tarea 1: Crear dataset de tracks des-normalizado

Procesad los tres archivos originales contenidos al archivo comprimido *dades.zip* con tal de crear un dataset final de *tracks* que incluya también los datos de los álbumes y artistas correspondientes.

Se debe tener en cuenta que debido a errores de extracción de los datos de los sistemas orígen:

- Algunos nombres de artistas no tienen el formato correcto. El nombre debería tener cada palabra del nombre en mayúsculas (por ejemplo, 'the beatles' debería estar registrado como 'The Beatles').
- Algunos registros de *popularity* del dataset de *tracks* no tienen ningún valor. Para este primer análisis podemos asumir que las *tracks* que no tienen indicado el valor de *popularity* tendrán el valor medio de todas las *tracks*.

Otros aspectos a tener en cuenta:

- El archivo con los datasets originales es un archivo comprimido en formato ZIP y se deberá descomprimir de forma programática.
- Se recomienda utilizar métodos de la librería *pandas* para leer y procesar los archivos anteriores.

Criterios de aceptación:

- Mostrar por pantalla un comentario con el número de *tracks* totales y el número de columnas del dataset final.
- Mostrar por pantalla un comentario con el número de *tracks* que no tienen un valor de *popularity*.

Tarea 2: Explorar alternativas de lectura de ficheros

Explorad métodos alternativos para leer columnas de un fichero CSV. A pesar de que los archivos actuales tienen un tamaño reducido, queremos observar el comportamiento cuando el tamaño de dichos archivos sobrepasa 1Gb.

Para este estudio queremos encontrar una manera más eficiente de leer una columna, comparado con hacer la misma acción mediante la librería *pandas*. Para hacerlo se deben implementar dos funciones con la siguiente nomenclatura:

- *get_column_pandas*: implementará la lectura mediante *pandas*.
- *get_column_{your_method}*: implementará la lectura mediante el método escogido por vosotros. Debéis sustituir *{your_method}* con un nombre que identifique de alguna manera el algoritmo escogido.

Las dos funciones recibirán como parámetros de entrada:
- la ruta del archivo CSV a leer.
- el nombre de la columna del archivo CSV que queremos leer.

Las dos funciones deberán retornar una lista con todos los valores de la columna.

Queremos comparar el tiempo de ejecución de cada una de las versiones variando los parámetros de entrada según el que se especifica en los criterios de aceptación. Para poder evaluar el tiempo de ejecución de archivos de diferente tamaño, os proponemos utilizar los 3 archivos originales que se han utilizado en la Tarea 1, dado que son de tamaños diferentes (el de artistas siendo el más pequeño y el de canciones siendo el más grande). Realizad ejecuciones para las dos versiones utilizando los 3 datasets originales disponibles leyendo en cada caso las siguientes columnas:
- artists_norm.csv: *artist_id*
- albums_norm.csv: *album_id*
- tracks_norm.csv: *track_id*

Para facilitar el análisis y la comparación, se pide crear un gráfico que compare las dos versiones de la función de lectura con el número de filas leídas en el eje horizontal y el tiempo de ejecución en el eje vertical.

Criterios de aceptación:
- Mostrar por pantalla el gráfico que muestra la comparación entre las dos versiones de la función.

Tarea 3: Filtrado y contadores básicos

Realizad un análisis exploratorio inicial de los datos utilizando el dataset de *tracks* creado en la Tarea 1 respondiendo a las siguientes preguntas:
A. ¿Cuántas *tracks* hay del artista *Radiohead*?
B. ¿Cuántas *tracks* contienen la palabra *'police'* en el título?
C. ¿Cuántas *tracks* son de álbumes publicados en la década del 1990?
D. ¿Cuál es la *track* con más popularidad de los últimos 10 años?
E. ¿Qué artistas tienen *tracks* en cada una de las décadas desde el 1960?

Se debe remarcar que el objetivo es implementar las funciones necesarias para responder las preguntas anteriores, pero que a la vez puedan ser reutilizadas en futuras fases del proyecto y que hagan el mantenimiento sencillo.

Criterios de aceptación:

- Mostrar por pantalla un comentario para cada una de las preguntas anteriores.

Tarea 4: Análisis inicial de *audio features*

En esta tarea queremos centrarnos en el análisis de los datos de *audio features* para empezar a entenderlas e interpretarlas.

Tenéis que:

A. Calcular el mínimo, la media y el máximo de la feature *energy* de todas las tracks de *Metallica.*
B. Calcular la media de la feature *danceability* de cada álbum de *Coldplay* y crear una gráfica para visualizar el resultado.

Criterios de aceptación:

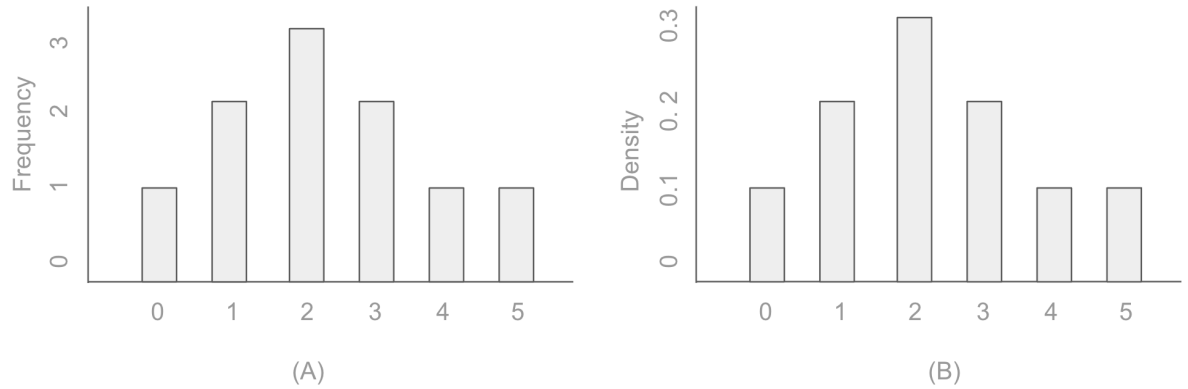
- Mostrar por pantalla un comentario con las estadísticas básicas de A.
- Mostrar por pantalla la gráfica generada en B.

Tarea 5: Histograma de una *audio feature* de un artista

Aparte del análisis estadístico básico realizado en la tarea anterior, necesitamos una manera visual de mostrar los datos de una *audio feature* del artista en cuestión. Se ha decidido explorar la visualización en formato histograma donde en el eje horizontal se muestran los diferentes valores que puede tomar la *audio feature* y en el eje vertical se muestra la densidad de probabilidad de cada valor. Es importante entender que la densidad de probabilidad no es la frecuencia (el número de veces que se repite un valor concreto) sino una normalización de la frecuencia.

A modo de ejemplo, en la siguiente figura se muestran dos histogramas de los datos contenidos en la siguiente lista: [0, 1, 1, 2, 2, 2, 3, 3, 4, 5] de N = 10 elementos.

- (A) histograma de frecuencia: cuántas veces se repite cada número.
- (B) histograma de densidad de probabilidad: cuántas veces se repite cada número dividido por el número total de muestras.



Ejemplo de histograma de (A) frecuencia y (B) de densidad de probabilidad. Datos de N=10 muestras

Para más detalles podéis consultar aquí.

El objetivo de la tarea es crear una función que reciba los datos y configuración necesaria para crear el histograma y visualizarlo.

Criterios de aceptación:

- Mostrar por pantalla un histograma que muestre la *acousticness* de las canciones de *Ed Sheeran*.

Tarea 6: Comparar artistas visualmente

Una vez tenemos el histograma de una única *audio feature* y de un único artista, la parte interesante es utilizar este tipo de visualizaciones para comparar dos artistas.

Se ha de crear una versión del histograma anterior que incluya los datos de dos artistas superponiendo los dos histogramas el uno sobre el otro. Es importante buscar una forma de visualizar los dos histogramas superpuestos sin que se tapen el uno al otro.

Dado que estamos comparando dos artistas, aquí cobra mucha importancia utilizar un histograma que muestre la densidad de probabilidad y no la frecuencia ya que si comparamos dos artistas con un número muy diferente de tracks, la visualización no ayudará a analizarlos porque las alturas de los histogramas serían muy diferentes. En cambio, visualizando la densidad de probabilidad eliminamos el efecto de dicha diferencia.

Igual que en las tareas anteriores, es importante hacer una implementación utilizando funciones parametrizadas que puedan ser reutilizadas de forma fácil.

Criterios de aceptación:

- Mostrar por pantalla un histograma que compare la *energy* de *Adele* y la de *Extremoduro*.

Tarea 7: Calcular similitud entre artistas

Otra forma de comparar dos artistas es hacerlo utilizando un cálculo matemático. Dado que por cada track tenemos 12 *audio features*, podemos obtener la media de las *audio features* para cada artista (un vector de 12 valores) y después comparar dos artistas comparando sus dos vectores de *audio features*.

Se ha de explorar la mejor manera de comparar artistas y para ello se analizarán dos formas de calcular la similitud entre dichos artistas:

- Similitud euclidiana
- Similitud cosinus

Para cada una de las métricas se ha de construir una matriz con los valores de similitud entre todos los pares de artistas. Es decir, en la posición (1, 3) encontramos la similitud entre el artista 1 y el artista 3. Una vez calculada la matriz, se ha de visualizar en una imágen tipo *heatmap* para poder ver de forma rápida la similitud entre todos los artistas.

Es importante tener en cuenta que la implementación para crear la matriz ha de estar pensada para el futuro y debería ser sencillo poder incluir una nueva métrica.

Criterios de aceptación:

- Mostrar por pantalla els *heatmap* de la similitud euclidiana i de la similitud cosinus comparando los artistas *Metallica*, *Extremoduro*, *AC/DC* y *Hans Zimmer*.

Tarea 8 (Opcional): Llamadas a API externa

Para completar el dataset original de artistas nos interesan algunos datos que se pueden obtener a través de la API de AudioDB: el año de formación del artista y la ciudad de origen.

El objetivo de la tarea es crear un dataset independiente con las columnas *artist_name*, *formed_year*, *country*. Dicho dataset se ha de guardar en un archivo CSV con el nombre de *artists_audiodb.csv*.

Se ha de tener en cuenta que la implementación debe ser eficiente en el caso que se quiera buscar muchos artistas, la cual implica un gran volumen de llamadas a la API remota y por lo tanto, muchos instantes de espera hasta obtener la respuesta.

Criterios de aceptación:

- Mostrar por pantalla los datos obtenidos (*artist_name*, *formed_year*, *country*) para los artistas *Radiohead*, *David Bowie* y Måneskin.
- Evaluad vuestra implementación descargando la información de todos los artistas del dataset original y comentad brevemente por qué creéis que vuestra implementación es eficiente y cómo creéis que escalará si se tienen que buscar miles de artistas simultáneamente.

Otras consideraciones

Tests

Se pide también que el código esté debidamente documentado y testeado, asegurando una cobertura del código fuente de como mínimo el 50%. Se recomienda el uso de la librería *unittest* debido a su uso en anteriores proyectos.

El cálculo de la cobertura de los tests se utiliza para evaluar la eficacia de los tests propuestos. En particular, sirve para determinar la calidad de los tests desarrollados y para determinar las partes críticas del código que no han sido testeadas. Para mesurar dicho valor os proponemos el uso de la herramienta Coverage.py. En la documentación podéis encontrar cómo cómo instalarla y cómo usarla.

Guía de estilo

Se pide también que el código siga la guía de estilo de PEP8. Para hacerlo se recomienda el uso de la herramienta black ya que nos permite automatizar el formateo de nuestro código siguiendo la guía de estilo de Python. Podéis encontrar más detalles sobre la instalación y configuración de dicha herramienta en el siguiente enlace.

Uso de Git

Con tal de poner en práctica lo que hemos aprendido en la Unidad 6 sobre Git, os proponemos el uso de Github Classroom para desarrollar vuestro paquete de Python. Github Classroom es una herramienta gratuita de código abierto que ayuda a simplificar el uso educativo de GitHub. Hemos utilizado Github Classroom para crear un aula donde hemos creado una tarea para la PAC4. Para hacer uso de este espacio que hemos creado, os aconsejamos seguir los pasos indicados en la guia adjunta a la descripción de la actividad del aula, donde explicamos cómo crear un repositorio para trabajar en la tarea que hemos preparado. El enlace para inscribiros lo encontraréis en el mensaje de la PAC4 publicado en el tablón del aula.

El uso de esta herramienta no es obligatorio para la evaluación de la PAC4, pero creemos que es una muy buena oportunidad para poner en práctica vuestros conocimientos en un entorno vital para todo aquel que trabaje o quiera trabajar en el ámbito de la ciencia de datos.

Criterios de corrección

Esta PAC se valorará siguiendo los criterios siguientes:

- **Funcionalidad** (6 puntos): Se valorará que el código implementa correctamente lo que se pide en el enunciado.
    - Tarea 1 (1 punto)
    - Tarea 2 (0.75 puntos)
    - Tarea 3 (0.5 puntos)
    - Tarea 4 (0.5 puntos)
    - Tarea 5 (0.75 punto)
    - Tarea 6 (0.75 punto)
    - Tarea 7 (1 punto)
    - Visualizaciones (0.75 puntos)
    - Tarea 8 (1 punto extra)
- **Documentación** (0.5 puntos): Todas las funciones de los ejercicios de la PEC deben estar correctamente documentadas utilizando docstrings.
- **Modularidad** (0.5 puntos): Se valorará la modularidad del código (tanto la organización del código en ficheros como la creación de funciones).
- **Estilo** (0.5 puntos): El código tiene que seguir la guía de estilo de Python (PEP8), exceptuando los casos donde hacerlo complique la legibilidad del código.
- **Tests** (1.5 puntos): El código tiene que contener una o varias *suites* de tests que permitan comprobar el buen funcionamiento de las funciones implementadas, obteniendo un mínimo del 50% de cobertura.
- **Requerimientos** (0.5 puntos): Es necesario crear un fichero de requerimientos que liste (sólo) las librerías necesarias para ejecutar el código.
- **README** i **licencia** (0.5 puntos): Se valorará la creación de un fichero README, que presente el proyecto y explique cómo ejecutarlo, así como la inclusión de la licencia bajo la cual se distribuye el código (podéis elegir la que queráis).

Importante

Nota 1: Del mismo modo que en las PECs anteriores, los criterios transversales se valorarán de manera proporcional a la parte de la funcionalidad implementada.

Por ejemplo, si el código sólo implementa la mitad de la funcionalidad requerida, y la documentación de esta parte es perfecta, entonces la puntuación correspondiente a la parte de documentación sería de 0.25.

Nota 2: Es imprescindible que el paquete que entreguéis se ejecute correctamente y que el fichero de README explique claramente cómo se tiene que ejecutar vuestro código para generar las gráficas resultantes del análisis, ejecutar los tests y comprobar la cobertura.

Nota 3: Entregad el paquete como un único archivo .zip en el Registro de Evaluación Continua**. El código Python tendrá que estar escrito en ficheros planos de Python.**