22.403 · Programación para la ciencia de datos

Grado en Ciencia de Datos Aplicada

Estudios de Informática, Multimedia y Telecomunicación

Programación para la ciencia de datos - PEC4

En este Notebook encontraréis el ejercicio que supone la cuarta y última actividad de evaluación continuada (PEC) de la asignatura. Esta PEC intenta presentaros un pequeño proyecto en el cual debéis resolver diferentes ejercicios, que engloba muchos de los conceptos cubiertos durante la asignatura.

El objetivo de este ejercicio será desarrollar un **paquete de Python** fuera del entorno de Notebooks, que nos permita resolver el problema dado. Trabajaréis en archivos Python planos `.py`. Éste tendrá que incluir el correspondiente código organizado lógicamente (separado por módulos, organizados por funcionalidad,...), la documentación del código (*docstrings*) y tests. Además, tendréis que incluir los correspondientes archivos de documentación de alto nivel (`README`) así como los archivos de licencia y dependencias (`requirements.txt`) comentados en la teoría.

Hacer un `setup.py` es opcional, pero si se hace se valorará positivamente de cara a la nota de la práctica y del curso.

Rendimiento de los estudiantes universitarios en Cataluña

Descripción del Dataset: Rendimiento de los estudiantes universitarios en Cataluña

El conjunto de datos recoge los principales indicadores vinculados a la medida del rendimiento académico de los estudiantes, como son las tasas de rendimiento, de graduación, de eficiencia y de abandono. Los datos han sido descargados desde el siguiente enlace:

- https://recercaiuniversitats.gencat.cat/ca/03_ambits_dactuacio/xifres/docencia/Rendiment-dels-estudiants/

Se trabajará con dos datasets principales:

1. Tasa de rendimiento (rendiment_estudiants.xlsx)

Para un determinado curso, es el cociente entre el número de créditos superados respecto al número de créditos matriculados en los estudios de grados y másteres.

- En este cálculo solo se tienen en cuenta los **créditos ordinarios**, que son los propios de un estudio.
- Se excluyen los créditos reconocidos, los adaptados y los transferidos de otros planes de estudios.

2. Tasa de abandono (taxa_abandonament.xlsx)

Para un determinado curso, es la proporción de estudiantes de grado o máster que abandonan el Sistema Universitario Catalán (SUC) en el primer curso sobre el total de nuevo acceso de aquel curso.

- Se establece que el estudiante ha abandonado el SUC si no se matricula en el próximo curso en ninguna de las universidades catalanas.

Proyecto Python, funcionalidad

Para hacer la entrega más fácil y homogénea os pedimos que organicéis el código de tal manera que **desde el fichero principal retorne todas las respuestas que se os pida en la PEC** haciendo uso de funciones que tendréis que definir en módulos. Para ello, en cada ejercicio os indicaremos el formato que tiene que tener cada respuesta, de tal manera que ejecutando `main.py` se vaya respondiendo a toda la PEC.

Por defecto, `main.py` debe ejecutar todas las funciones de la PEC mostrando cómo funcionan pero también debe permitir ejecutarlas una a una si se desea (entendiendo que para ejecutar la segunda se tiene que haber ejecutado la primera, etc). Debéis documentarlo todo muy bien en el *README* para que el profesor pueda ejecutar el código sin problemas y sin dudas. En el README también tendréis que indicar cómo instalar el proyecto, cómo ejecutarlo, generar la documentación, ejecutar los tests y la cobertura, comprobar el cumplimiento de las reglas de estilos.

Ejercicio 1. Load dataset y EDA

- Crear una función que acepte una ruta de archivo como argumento opcional. Si se proporciona la ruta, la función debe leer el fichero correspondiente. Si no se

proporciona ninguna entrada, la función debe preguntar al usuario qué dataset de los dos disponibles desea cargar. Esta función se deberá reutilizar en los siguientes ejercicios.

- Exploración del dataset. Para el dataset que el usuario seleccione, se debe:
    - 1.1. Mostrar las 5 primeras filas.
    - 1.2. Mostrar las columnas del dataframe.
    - 1.3. Mostrar la información (*info()*)

Ejercicio 2: limpieza de los datos y filtrado

Se pretende hacer un análisis de los dos datasets que se proporcionan. Para ello, lo primero que queremos hacer es limpiar y unir los datasets. Como habrás podido comprobar en el EDA, los datasets tienen estructuras similares pero no completamente iguales. Los objetivos de este ejercicio son:

- Homogeneizar ambos datasets: implica que las columnas tengan el mismo nombre y que los datos que se muestren tengan el mismo origen
- Actualmente cada línea de datos muestra la información de un estudio en concreto por curso académico y sexo (para diferenciar los resultados entre hombres y mujeres). Para simplificar el análisis posterior, queremos agrupar los estudios por rama (Branca), de tal forma que tengamos una línea por curso académico y rama de estudio, donde la tasa de rendimiento y el % de abandono sea la media de los estudios que contienen esa rama.
- Finalmente queremos crear un dataset fusionado a partir de ambos datasets

Por lo tanto, este ejercicio debe:

- 2.1. Renombrar las columnas del dataset taxa_abandonament.xlsx para que coincida con el dataset rendiment_estudiants.xlsx
- 2.2. Eliminar las columnas de "Universitat", "Unitat" en ambos dataframes, y también "Crèdits ordinaris superats" y "Crèdits ordinaris matriculats" en el caso del dataset de rendimiento.
- 2.3. Crear y aplicar a los datasets una función que agrupe todas las filas que compartan las mismas características (excepto el nombre del estudio) para ambos datasets. La función debe devolver un nuevo dataset con:
    - Una fila por cada combinación únicas de las columnas ['Curs Acadèmic', 'Tipus universitat', 'Sigles', 'Tipus Estudi', 'Branca', 'Sexe', 'Integrat S/N']
    - Una columna con el rendimiento medio, en el caso del dataset de rendimiento y con la tasa media de abandono en el caso del dataset de abandono.

- 2.4. Crear una función para fusionar ambos datasets. El dataset resultante solo debe contener las filas coincidentes entre ambos datasets. A partir de ahora utilizaréis este datasets en los ejercicios futuros.

Nota: Para agrupar los datasets podéis utilizar el método *groupby* de pandas, y para fusionar ambos datasets, el método *merge* con la propiedad *inner*.

Ejercicio 3: Análisis Visual de Tendencias Temporales

3.1. Generación de Gráficos de Series Temporales

Crear una función que genere visualizaciones de series temporales con las siguientes características:

- Crear **dos gráficos** (subplots) en una misma figura:
  - **Gráfico 1**: Evolución del % de Abandonamiento por curso académico
  - **Gráfico 2**: Evolución de la Tasa de Rendimiento por curso académico
- Cada gráfico debe mostrar:
  - Una línea diferente para cada rama de estudio (*Branca*)
  - Leyenda identificando cada rama
  - Grid para facilitar la lectura
  - Etiquetas de ejes apropiadas
  - Título descriptivo

Visualización:

- Tamaño de figura recomendado: 14x10 pulgadas
- Utilizar colores distintos para cada rama (sugerencia: `plt.cm.tab10`)
- Rotar etiquetas del eje X si es necesario para mejor legibilidad

3.2. Guardar las Visualizaciones

- Guardar la figura generada en el directorio `src/img/`
- Nombre del archivo: `evolucion_nombre_alumno.png`
- Resolución recomendada: 300 dpi
- Crear el directorio si no existe

Ejercicio 4: Análisis estadístico automatizado

En este ejercicio, crearás una función llamada `analyze_dataset()` que se encargará de realizar un análisis estadístico completo del dataset fusionado que obtuviste en el ejercicio 2. El objetivo es calcular diferentes métricas y estadísticas

que nos permitan entender mejor el comportamiento del rendimiento y el abandono universitario, y guardar todos estos resultados en un archivo JSON bien estructurado.

Estructura del análisis JSON:

El archivo JSON que debes generar (`src/report/analisi_estadistic.json`) debe contener los siguientes apartados. Observa cómo está organizado en secciones lógicas que van de lo general (metadata y estadísticas globales) a lo específico (análisis por rama). Podrás ver un ejemplo de la estructura en (`examples/analisi_estadistic_example.json`) (*Nota: Los valores del ejemplo no son los correctos*).

4.1. Sección Metadata:

Esta sección debe contener información básica sobre el análisis que estás realizando. Piensa en ella como la "ficha técnica" de tu análisis:

- **fecha_analisis**: La fecha actual en formato ISO (YYYY-MM-DD). Usa `datetime.now().strftime("%Y-%m-%d")` para obtenerla automáticamente.
- **num_registros**: El número total de registros en tu dataset fusionado. Simplemente usa `len(merged_df)`.
- **periodo_temporal**: Una lista ordenada con todos los cursos académicos únicos que aparecen en tus datos. Puedes obtenerla con `sorted(merged_df['Curs Acadèmic'].unique())`.

4.2. Estadísticas Globales:

Aquí calcularás las métricas que resumen el comportamiento general de todo el sistema universitario catalán, sin diferenciar por ramas:

- **abandono_medio**: La tasa media de abandono en primer curso, calculada como la media de la columna `'% Abandonament a primer curs'`.
- **rendimiento_medio**: La tasa media de rendimiento, calculada como la media de la columna `'Taxa rendiment'`.
- **correlacion_abandono_rendimiento**: Este es un valor muy interesante que nos indica si existe relación entre el abandono y el rendimiento. Para calcularlo, debes usar la correlación de Pearson de scipy:

```
from scipy.stats import pearsonr

corr, p_value = pearsonr(
    merged_df['% Abandonament a primer curs'].dropna(),
    merged_df['Taxa rendiment'].dropna()
)
```

Una correlación negativa (como -0.68) nos indica que a mayor abandono, menor rendimiento, lo cual tiene sentido intuitivo.

4.3. Análisis por Rama:

Esta es la sección más completa del análisis. Para cada rama de estudios (Arts i humanitats, Ciències, Ciències de la salut, etc.), debes calcular:

- **Estadísticas descriptivas básicas:**
  - Media y desviación estándar del porcentaje de abandono
  - Media y desviación estándar de la tasa de rendimiento
- **Detección de tendencias temporales:**

  Aquí es donde aplicarás la regresión lineal para detectar si el abandono en cada rama está mejorando (tendencia decreciente), empeorando (tendencia creciente), o se mantiene estable en el tiempo.

  El proceso es el siguiente:

  a) Primero, agrupa los datos por año académico para cada rama:

```
branch_data = merged_df[merged_df['Branca'] == branch]
branch_by_year = branch_data.groupby('Curs Acadèmic').agg(
    '% Abandonament a primer curs': 'mean'
}).reset_index()
```

  b) Extrae las listas de años y valores:

```
years = branch_by_year['Curs Acadèmic'].tolist()
valores_abandono = branch_by_year['% Abandonament a primer
```

  c) Calcula la regresión lineal usando scipy:

```
from scipy.stats import linregress

slope, intercept, r_value, p_value, std_err = linregress(
    range(len(years)),  # Posiciones: 0, 1, 2, 3...
    valores_abandono
)
```

d) Interpreta la pendiente (slope):

- Si `slope > 0.01`: la tendencia es "creciente" (el abandono aumenta con el tiempo)
- Si `slope < -0.01`: la tendencia es "decreciente" (el abandono disminuye)
- Si está entre -0.01 y 0.01: la tendencia es "estable" (sin cambios significativos)

4.4. Rankings:

Para terminar el análisis, identifica qué ramas tienen los mejores y peores resultados. Esto es útil para tomar decisiones sobre dónde enfocar recursos o investigar qué están haciendo bien las ramas con mejores resultados:

- Rama con mejor rendimiento (tasa más alta)
- Rama con peor rendimiento (tasa más baja)
- Rama con mayor abandono (porcentaje más alto)
- Rama con menor abandono (porcentaje más bajo)

Puedes calcular los rankings ordenando las ramas según cada métrica y tomando el primer/último elemento.

Requerimientos del proyecte Python

README, LICENCE, requirements.txt

- **README**: Tenéis que añadir el fichero README (README.txt, README.md), que presente el proyecto, la estructura de carpetas, y explique cómo instalarlo y ejecutarlo. También cómo ejecutar los tests, generar la documentación y comprobar el linting. La explicación de cómo se ejecuta dentro de un entorno virtual es opcional.
- Incluir el fichero **LICENSE** de la licencia bajo la que se distribuye el código (podéis escoger la que queráis).
- **requirements.txt**: fichero de requisitos que contenga la lista de librerías necesarias parar ejecutar el código. Han de ser las librerías propias de este proyecto (no hace falta incluir, aunque se puede, las librerías accesorias para ejecutar los tests, linter, documentación).

Modularidad

El código tiene que ser modular, tanto en las funciones que se definan (programación estructurada), como en la organización del código en módulos. Una posibilidad es

separar cada ejercicio en un fichero. Otro alumno preferirá que los ficheros de los módulos sean funcionalidades. Típicamente habrá el fichero *main.py*, y este fichero importará los módulos y ejecutará los diferentes ejercicios. La idea es que toda la funcionalidad esté en los módulos, y el fichero *main.py* sirva como punto de entrada para leer las opciones y ejecutar los módulos/ejercicios.

Opciones en la línea de comandos

Puedes utilizar directamente la librería *sys*, o bien alguna otra posibilidad (*getopt* o *argparse*, que es la que se utiliza en la solución), tanto da. Lo que se pide es que haya la opción de ayuda (-h, --help), y ejecutar los ejercicios individualmente (*-ex num*). Si pasamos *-ex 5* presupone que se ejecutan del ejercicio 1 al 5, pues los ejercicios son progresivos. Si no pasamos argumentos, se ejecutan todos los ejercicios.
Documentar en el *README*.

Documentación

Todas las funciones de los ejercicios de esta PEC tendrán que estar debidamente documentadas utilizando docstrings (en el formato que prefiera). Se debe generar la documentación html a partir de los docstrings, que quedará en la carpeta *doc/*. Documentar en *README* y hacer captura de pantalla en la carpeta *screenshots/* de la raíz del proyecto (hay que visualizar el nombre del alumno).

Testing y coverage

El código debe contener una o varias suites de test que permitan comprobar que el código funciona correctamente, con un mínimo del 50% de cobertura. Puede hacer un archivo de test para cada ejercicio, contemplando varios casos. Documentar en *README* cómo se ejecutan los tests y cómo se ejecuta la cobertura (**importante**). Realizad una captura de pantalla en la carpeta *screenshots/* de la raíz del proyecto de cómo se ha ejecutado los tests y la cobertura (hay que visualizar el nombre del alumno).

Linter

El código debe seguir la guía de estilo de Python (PEP8), exceptuando los casos en los que hacerlo complique la legibilidad del código. Puedes utilizar el linter *pylint*, y puedes poner las excepciones que no te parezcan correctas según tu criterio en el archivo *.pylintrc*, de forma que consigas unos valores superiores a 9 en tus scripts. Documentar en el *README* y hacer captura de pantalla en la carpeta *screenshots/* de la raíz del proyecto (hay que visualizar el nombre del alumno).

Capturas de pantalla

Entregarás la carpeta *screenshots/* tal como se ha pedido a lo largo del enunciado.

NOTA: como siempre, no utilicéis rutas absolutas.

NOTA: tendréis que entregar un zip con el contenido del proyecto. Típicamente el proyecto contendrá las carpetas *src/*, *src/modules/*, *tests/*, *doc/*, *screenshots/* (aunque algún alumno puede tomar decisiones diferentes). En la raíz del proyecto se espera encontrar los archivos *README* (formato texto o *README.md* formato markdown), *requirements.txt*, *LICENSE*, y quizás algún otro como *.pylintrc*, etc.

Rúbrica

Funcionalidad (4 p)

- **Ejercicio 1 (0,5 p)**
- **Ejercicio 2 (0,75 p)**
- **Ejercicio 3 (0,75 p)**
- **Ejercicio 4 (2,00 p)**

Aspectos formales del proyecto Python (6 p)

- **README, LICENCE, requirements.txt (1 p)**
- **Modularidad (1 p)**
- **Opciones en la línea de comandos (ejecutar los ejercicios individualmente y ayuda). (1 p)**
- **Generar documentación (formato html) (1 p)**
- **Testing y coverage (1 p)**
- **Linter (pylint) (0,50 p)**
- **Capturas de pantalla (visualizar el nombre del alumno) (0,50 p)**

Haz doble clic (o pulsa Intro) para editar