
UNIVERSIDAD DE GRANADA

# Práctica 4. Desarrollo de un Sistema de Recomendación basado en Filtrado Colaborativo.

*Gestión de Información en la Web*

Máster Profesional en Ingeniería Informática

Curso académico 2022/2023

**Autor**

*José Alberto Gómez García*

*modej@correo.ugr.es*

*26514779B*

# Índice

# 1. Introducción.

En esta segunda práctica de la asignatura “Gestión de Información en la Web” se va a realizar el desarrollo de un Sistema de Recomendación basado en filtrado colaborativo. Trabajaremos haciendo uso de la base de datos *MovieLens*, con unas 100.000 valoraciones de 943 usuarios para 1682 películas.

En esta práctica se pide la construcción de una aplicación que pida al usuario que valore un determinado número de películas (20 por defecto), tras lo cual deberá construir un vecindario (por defecto de tamaño 10) de usuarios con gustos similares. Por último, y basado en las valoraciones de estos usuarios similares, se proporcionará una lista de recomendaciones de películas para el usuario activo de la aplicación. Estas recomendaciones deberán contar con una valoración de mínimo 4 estrellas.

# 2. Desarrollo del software.

El lenguaje de programación escogido para desarrollar el proyecto ha sido Python (en su versión 3.10), ya que tengo cierta experiencia con él y es sencillo de utilizar, facilitando en gran medida las operaciones de cálculo y manejo de datos que necesitaremos hacer. Además, se utilizarán las librerías Pandas (en su versión 1.5.3) y Numpy (en su versión 1.24.2). Se ha utilizado el IDE PyCharm en su versión 2022.2.4 sobre un sistema con Windows 11 Pro.

El software desarrollado se divide en 3 ficheros, los cuales comentaremos en las siguientes subsecciones.

## 2.1. Configuración.

Configuration.py contiene una “dataclass”, empleada para almacenar la configuración del recomendador. Se proporcionan los siguientes parámetros de configuración:

- DATA_DIR: directorio de donde leer los ficheros con la base de datos de *MovieLens*.
- RESULTS_DIR: directorio donde guardar el fichero con las recomendaciones realizadas.
- MOVIES_FILE: fichero por defecto de donde leer la base de datos de película. Hace uso de DATA_DIR.
- RATINGS_FILE: fichero por defecto de donde leer la base de datos de las valoraciones. Hace uso de DATA_DIR.
- RESULTS_FILE: fichero por defecto donde guardar las recomendaciones realizadas. Hace uso de RESULTS_DIR.
- MOVIES_TO_BE_RATED: indica el número de películas que el usuario debe valorar al comienzo del programa. Por defecto, de acuerdo con el guion, son 20.

- NEIGHBOURS: tamaño del vecindario de usuarios a tener en cuenta. Por defecto, de acuerdo con el guion, son 10.
- USE_MAX_RESULTS: valor booleano que permite limitar el número de recomendaciones mostradas al usuario. Por defecto se encuentra activado.
- MAX_RESULTS: número máximo de recomendaciones mostradas al usuario. Sólo se utilizará de estar activado USE_MAX_RESULTS.
- DELETE_IRRELEVANT_USERS: valor booleano que nos permite decidir si obviar usuarios que no hayan valorado un determinado número de películas de entre aquellas que valoró nuestro usuario al principio del programa. Por defecto se encuentra activado.
- THRESHOLD_TO_BE_IRRELEVANT: número de películas que deberá un usuario de la base de datos tener en común con nosotros para tenerlo en cuenta de cara al cálculo del vecindario. Por defecto se elige el valor de 3 (pensado para el caso en que el usuario activo haya valorado 20 películas).
- SHOW_DEBUG_INFO: valor booleano para elegir si deseamos imprimir el vecindario seleccionado (índice de usuario y correlación de Pearson asociada) durante la ejecución del programa. Por defecto se encuentra desactivado.

## 2.2. Recomendador.

En el fichero “recomendador.py” se recoge toda la lógica del sistema de recomendación, encapsulada en una única clase. Esta clase contiene una serie de métodos, cada uno de ellos responsable de realizar una parte del proceso de recomendación, y una serie de atributos encargados de almacenar la información necesaria para poder realizar dicho proceso. Veamos cada uno de ellos.

En el constructor de la clase “Recomendador” se pregunta al usuario si desea utilizar la configuración por defecto. En caso negativo, se le preguntará por el número de películas que deberá valorar el usuario, el tamaño del vecindario, si desea limitar el número de recomendaciones (y en caso afirmativo, en qué número), si desea utilizar sólo usuarios con un mínimo de películas valoradas en común con nosotros (y en caso afirmativo qué número). No se adjunta captura de este código dado el carácter repetitivo del mismo.

Finalmente, se le pide al usuario la ruta de los ficheros con la base de datos de películas y valoraciones, respectivamente. Para evitar la tortuosa tarea de introducir rutas de forma manual vía texto se ha creado una pequeña función que nos permite hacer uso de la típica interfaz gráfica para seleccionar ficheros. El código de esta función se puede ver en la imagen 1.

```python
@staticmethod
def pedir_fichero(title, initialdir=configuracion.DATA_DIR, filetypes=None):
    if filetypes is None:
        filetypes = [('Todos los ficheros', '.*')]
    root = Tk()
    root.withdraw()  # Ocultar ventana padre de Tk y mostrar la nuestra como popup
    root.attributes('-topmost', True)
    try:
        fichero = filedialog.askopenfilename(
            initialdir=initialdir, title=title,
            filetypes=filetypes)
        root.destroy()
    except (OSError, FileNotFoundError):
        print('No se ha podido abrir el fichero.')
        sys.exit(1)
    except Exception as e:
        print(f'Ha ocurrido un error inesperado: {e}')
        sys.exit(1)
    if len(fichero) == 0 or fichero is None:
        print('No se ha seleccionado ningún fichero.')
        sys.exit(1)

    return fichero
```

*Imagen 1. Método para pedir ficheros al usuario.*

Leídos los ficheros con las bases de datos, cargamos su contenido en dos DataFrames de Pandas haciendo uso de la función de lectura de ficheros CSV de Pandas. Aunque estos ficheros no son CSV, podemos utilizar los parámetros de esta función para adaptar la lectura a nuestras necesidades. Cabe destacar que se ha usado el encoding “mbcs” dado que los ficheros no se encontraban codificados en UTF-8, si no en el formato propio de Windows.

Posteriormente, crearemos el resto de atributos que necesitaremos. Entre estos se encuentra el número de películas a valorar por el usuario, el tamaño del vecindario, y las estructuras de datos (diccionarios) que almacenaran el vecindario de usuarios, las valoraciones proporcionadas por el usuario del programa y las recomendaciones que se le ofrecerán finalmente.

Para cambiar la ruta donde se guardarán las recomendaciones y activar o desactivar el modo que imprime características del vecindario (llamado DEBUG), deberá acudirse al fichero de configuración.

```
        self.peliculas = pd.read_csv(
            filepath_or_buffer=self.configuracion.MOVIES_FILE,
            sep='|',
            header=None,
            engine='python',
            encoding='mbcs',  # ANSI encoding in Windows, el fichero no está en UTF-8
            usecols=[0, 1, 2],
            names=['idMovie', 'title', 'date'])

        self.valoraciones = pd.read_csv(
            filepath_or_buffer=self.configuracion.RATINGS_FILE,
            sep='\t',
            header=None,
            engine='python',
            encoding='mbcs',  # ANSI encoding in Windows, el fichero no está en UTF-8
            usecols=[0, 1, 2],
            names=['idUser', 'idMovie', 'rating'])

        self.num_peliculas_a_valorar = self.configuracion.MOVIES_TO_BE_RATED
        self.tamanio_vecindario = self.configuracion.NEIGHBOURS
        self.vecindario = {}
        self.valoraciones_por_usuario = {}
        self.recomendaciones = {}
```

*Imagen 2. Creación de los DataFrames y resto de atributos necesarios.*

Construido el objeto, haremos uso de una función para pedir al usuario la valoración de un determinado número de películas elegidas aleatoriamente. El código se adjunta en la imagen 3 que se muestra a continuación.

```
def pedir_valoraciones_al_usuario(self):
    peliculas_a_valorar = self.peliculas.sample(n=self.num_peliculas_a_valorar, replace=False)
    print('Por favor, valora las siguientes películas del 1 al 5:')

    contador = 1
    for _, row in peliculas_a_valorar.iterrows():
        titulo = row['title']
        while True:
            print(f'Película {contador} de {self.num_peliculas_a_valorar}: {titulo}')
            valoracion = input('Valoración: ')
            if valoracion == "" or int(valoracion) not in range(1, 6):
                print('Valoración no válida. Estas deben estar entre 1 y 5 estrellas.')
            else:
                break
        self.valoraciones_por_usuario[row['idMovie']] = int(valoracion)
        contador += 1

    print('Gracias por tu colaboración al valorar las películas. \n')
```

*Imagen 3. Función para pedir al usuario la valoración de películas.*

A continuación, calcularemos el vecindario de usuarios más afines al usuario del programa. Para ello, crearemos un tercer DataFrame, que contiene las valoraciones de cada usuario (filas) para cada película (columnas), teniendo un cero para aquellos casos en los que un usuario no haya valorado una determinada película. Véase el código de la imagen 4.

En la imagen 5 se muestra el proceso de eliminación de usuarios que no hayan valorado un determinado número de películas en común con el usuario. También, se calcula la matriz de correlación (usando el coeficiente de Pearson) entre usuarios. Nos quedaremos con la correlación de los usuarios de la base de datos con nuestro usuario (índice 0) y las ordenaremos de mayor a menor a correlación. Guardaremos en el diccionario del vecindario la pareja (idUsuario, correlación), de los usuarios con mayor correlación.

```python
def calcular_vecindario(self):
    print('Calculando vecindario...\n')

    usuarios = self.valoraciones['idUser'].unique()
    usuarios = list(np.insert(usuarios, 0, 0))  # Mi usuario es el 0, la DB empieza por 1.

    # Crear matriz que agrupe los usuarios y las películas
    matriz_valoraciones = pd.DataFrame(index=usuarios, columns=self.peliculas['idMovie'])
    matriz_valoraciones = matriz_valoraciones.fillna(0)

    # Rellenar la matriz con las valoraciones del usuario
    for movie in self.valoraciones_por_usuario.keys():
        matriz_valoraciones.iloc[0, movie - 1] = self.valoraciones_por_usuario[movie]
    # Añadir las valoraciones de los demás usuarios
    for usuario in usuarios[1:]:
        valoraciones_usuario = self.valoraciones[self.valoraciones['idUser'] == usuario]
        for _, row in valoraciones_usuario.iterrows():
            matriz_valoraciones.iloc[usuario, row['idMovie'] - 1] = row['rating']
```

```python
    # Elimino a los usuarios con menos de X coincidencias, quedandome así con los de más de X coincidencias.
    if self.configuracion.DELETE_IRRELEVANT_USERS:
        usuarios_a_eliminar = list()
        grupos = self.valoraciones.groupby('idUser')['idMovie'].apply(list)
        for usuario, peliculas in grupos.items():
            common = set(self.valoraciones_por_usuario.keys()).intersection(set(peliculas))
            if len(common) < self.configuracion.THRESHOLD_TO_BE_IRRELEVANT:
                usuarios_a_eliminar.append(usuario)
        matriz_valoraciones.drop(matriz_valoraciones[usuarios_a_eliminar], axis=0, inplace=True)
        # Obtener el numero real de filas, al eliminar usuarios puede ser menor que el tamaño del vecindario
        num_filas = matriz_valoraciones.shape[0]
        if num_filas < self.tamanio_vecindario:
            self.tamanio_vecindario = num_filas - 1

    correlation_matrix = np.corrcoef(matriz_valoraciones)
    usuarios_ordenados = np.argsort(correlation_matrix[0])[::-1]

    for i in range(1, self.tamanio_vecindario + 1):
        self.vecindario[usuarios_ordenados[i]] = correlation_matrix[0][usuarios_ordenados[i]]

    if self.configuracion.SHOW_DEBUG_INFO:
        print('Vecindario (idUser : correlation):\n', self.vecindario)
```

*Imágenes 4 y 5. Función para crear el vecindario de usuarios afines.*

La función mostrada en la imagen 6 es la responsable de calculas las recomendaciones al usuario a partir del vecindario y valoraciones obtenidos anteriormente.

Para calcular el grado de recomendación (o valoración prevista) de cada película haremos uso de la fórmula mostrada en la parte inferior de la diapositiva 28 del tema 2 de esta asignatura, la cual se muestra a continuación.

$$\hat{r}(u,i)=\bar{r}(u)+C\sum_{v\in N_u^k} sim(u,v)(r(v,i)-\bar{r}(v))$$

$$C=\frac{1}{\sum_{v\in N_u^k}|sim(u,v)|}$$

Así pues, tenemos en cuenta las valoraciones de otros usuarios para dicha película, la media de las valoraciones de cada uno de los demás usuarios y el grado de correlación de cada uno de dichos usuarios con nosotros. También tenemos en cuenta la media de nuestras valoraciones.

Por último, fuerzo que el valor máximo sea de 5 (pues he tenido algún problema de redondeo), selecciono aquellas películas con un grado de recomendación mínimo de 4 y las ordeno de manera descendente.

```python
def calcular_recomendaciones(self):
    print('Calculando recomendaciones...\n')
    peliculas_valoradas = self.valoraciones_por_usuario.keys()
    valoraciones_medias = np.mean(list(self.valoraciones_por_usuario.values()))
    vecindario_media = {}
    for usuario in self.vecindario.keys():
        valoracion_vecindario = self.valoraciones[self.valoraciones['idUser'] == usuario]
        media_vecindario = np.mean(valoracion_vecindario['rating'])
        vecindario_media[usuario] = media_vecindario
    # Para cada pelicula no vista por el usuario, calcular el posible interés.
    for pelicula in self.peliculas['idMovie']:
        if pelicula not in peliculas_valoradas:
            similitud, corr_acum = 0, 0
            for usuario in self.vecindario.keys():
                correlacion = self.vecindario[usuario]
                valoracion = self.valoraciones[(self.valoraciones['idUser'] == usuario)
                                               & (self.valoraciones['idMovie'] == pelicula)]['rating']
                if len(valoracion) > 0:
                    similitud += correlacion * (valoracion.values[0] - vecindario_media[usuario])
                    corr_acum += abs(correlacion)
            interes = 0
            if corr_acum > 0:
                interes = (similitud / corr_acum) + valoraciones_medias
                interes = min(5, interes)  # El interés no puede ser mayor que 5 estrellas
            if interes >= 4:
                self.recomendaciones[pelicula] = interes
    # Ordenar las recomendaciones ordenadas por interés descendente
    self.recomendaciones = dict(sorted(self.recomendaciones.items(), key=lambda item: item[1], reverse=True))
```

*Imagen 6. Función para calcular las recomendaciones que mostrar al usuario.*

Las dos funciones mostradas en la imagen 7 implementan la lógica necesaria para imprimir las recomendaciones por terminal de texto y/o salvar dichas recomendaciones en un fichero de texto. Nótese que al mostrar los resultados por terminal se puede aplicar el límite especificado en la configuración, mientras que a la hora de exportar se utilizarán todos los resultados obtenidos.

```
def imprimir_recomendaciones(self):
    contador = 1
    for pelicula in self.recomendaciones.keys():
        if self.configuracion.USE_MAX_RESULTS and contador > self.configuracion.MAX_RESULTS:
            break
        titulo = self.peliculas[self.peliculas['idMovie'] == pelicula]['title'].values[0]
        print(f'{contador}. {titulo}. \tPuntuacion: {self.recomendaciones[pelicula]:.3f}')
        contador += 1

def exportar_recomendaciones(self):
    contador = 1
    directorio = self.configuracion.RESULTS_DIR
    if not os.path.exists(directorio):
        os.makedirs(directorio)

    with open(self.configuracion.RESULTS_FILE, 'x') as f:
        for pelicula in self.recomendaciones.keys():
            titulo = self.peliculas[self.peliculas['idMovie'] == pelicula]['title'].values[0]
            f.write(f'{contador}. {titulo}. \tPuntuación: {self.recomendaciones[pelicula]:.3f}\n')
            contador += 1
```

*Imagen 7. Funciones para imprimir las recomendaciones por terminal y/o exportarlas a un fichero.*

Finalmente, se adjunta una captura de pantalla del método “ejecutar”, el cual recoge todo el proceso anteriormente mencionado. Es este método el que el usuario debería utilizar tras crear un objeto de la clase “Recomendador”.

```
def ejecutar(self):
    self.pedir_valoraciones_al_usuario()
    self.calcular_vecindario()
    self.calcular_recomendaciones()
    while True:
        imprimir = input('¿Desea imprimir las recomendaciones? (S/N): ')
        imprimir.upper()
        if imprimir == 'S' or imprimir == 'N':
            break
        else:
            print('Opción no válida. Introduzca S o N.')
    if imprimir == 'S':
        self.imprimir_recomendaciones()
    while True:
        exportar = input('¿Desea exportar las recomendaciones? (S/N): ')
        exportar.upper()
        if exportar == 'S' or exportar == 'N':
            break
        else:
            print('Opción no válida. Introduzca S o N.')
    if exportar == 'S':
        try:
            self.exportar_recomendaciones()
            print('Recomendaciones exportadas correctamente.')
        except FileExistsError:
            print('Error al exportar las recomendaciones.')
    print('Fin del programa. Gracias por usar el sistema de recomendación.')
```

*Imagen 8. Función responsable del proceso completo de recomendación.*

### 2.3. Programa principal.

El fichero “main.py” es simplemente un pequeño “wrapper” encargado de la creación de un objeto de la clase “Recomendador” y la ejecución del proceso principal detallado al final de la sección anterior.

```
from src.recomendador import Recomendador

if __name__ == '__main__':
    print('Bienvenido al recomendador de películas.\n')

    recomendador = Recomendador()
    recomendador.ejecutar()
```

*Imagen 9. Programa principal.*

## 3. Mejoras e innovaciones.

Además de cumplir con los requisitos especificados en el guion de la práctica, se incorporan las siguientes mejoras e “innovaciones”.

- Se proporciona una amplia capacidad de personalización del funcionamiento de la aplicación, recogida en un sencillo fichero de configuración.
- Se ofrece una configuración por defecto, de manera que el usuario no tenga que preocuparse por nada y pueda comenzar a usar el sistema directamente.
- En caso de querer usarse otros valores, la mayoría de ellos pueden ser introducidos por interfaz de texto, proporcionándose una cómoda interfaz gráfica para seleccionar los ficheros que contienen las bases de datos de películas y valoraciones.
- Se permite exportar las recomendaciones generadas (junto a su grado de recomendación) en un fichero de texto.

## 4. Manual de usuario.

El proyecto ha sido desarrollado haciendo uso de Python 3.10, por lo que se deberá tener instalado el mismo en el sistema. En principio, el programa debería ser retrocompatible hasta Python 3.6, pero esto no ha sido testeado.

Para ejecutar el proyecto se necesita tener instalado las librerías de Pandas y Numpy. Dado el fichero de requisitos adjunto, estas pueden instalarse fácilmente haciendo uso de:

“pip install -r requirements.txt”

En un intento de seguir las buenas prácticas se recomienda hacer uso de entornos virtuales. En este tutorial no se explicará como crearlos ni gestionarlos, pero el lector interesado puede consultar [6].

Tras satisfacer los requisitos especificados, podremos ejecutar nuestro proyecto sin ningún problema. Dadas las rutas definidas por defecto en el fichero de configuración, el proyecto deberá ejecutarse ubicados en la carpeta resultado de descomprimir la entrega (teniendo *src, data* y *documentación* como hijas). De encontrarnos en un IDE como PyCharm, simplemente deberemos importar la carpeta padre.

Al ejecutar, se nos preguntará si deseamos utilizar la configuración por defecto. En caso negativo se pedirán diferentes parámetros al usuario, como se puede ver en la imagen 10. Nótese que no se muestran capturas de pantalla del menú de selección de archivos.

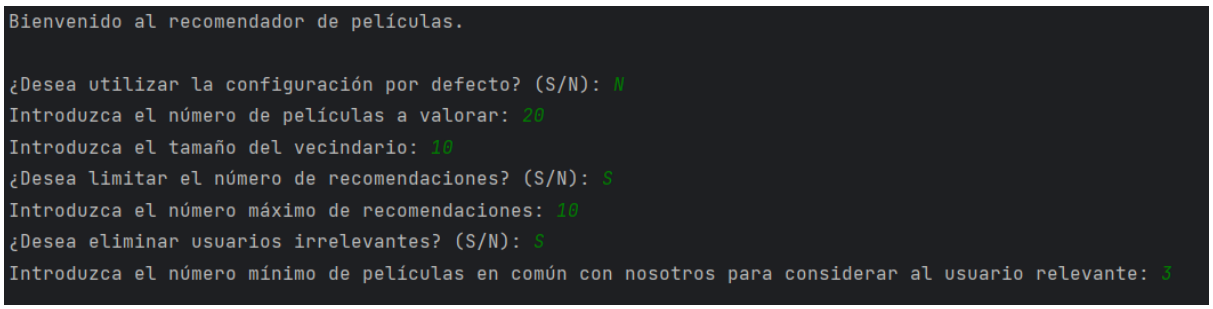


*Imagen 10. Petición de parámetros al usuario.*

Posteriormente, se pedirá al usuario la valoración de un determinado número de películas. Este proceso, aunque simple y repetitivo, se puede ver en las imágenes 11 y 12.



*Imágenes 11 y 12. Petición de valoraciones al usuario.*

Posteriormente, se calculará el vecindario y las recomendaciones. En la imagen 13 se puede ver los índices del top 5 usuarios del vecindario junto a su coeficiente de correlación de Pearson. Nótese que la opción SHOW_DEBUG_INFO está activada, lo cual no es el comportamiento por defecto.

Posteriormente, se preguntará al usuario si desea ver las recomendaciones generadas. En caso afirmativo, se mostrará un listado con el título de la película y la valoración calculada. Finalmente, se preguntará si se desean exportar los resultados en un fichero de texto. Si observamos la figura 14, veremos las recomendaciones entre los puestos 90 y 105, con valoraciones entre los valores 4.33 y 4.43. En total se generaron 163 recomendaciones.

```
Calculando vecindario...

Vecindario (idUser : correlation):
 {183: 0.1184948105318245, 15: 0.11097747380579606, 56: 0.10551335414456821, 11: 0.10361335385783499, 142: 0.09070835213228645,
Calculando recomendaciones...

¿Desea imprimir las recomendaciones? (S/N): S
1. Good, The Bad and The Ugly, The (1966).  Puntuacion: 5.000
2. Restoration (1995).  Puntuacion: 5.000
3. Rosewood (1997).     Puntuacion: 5.000
4. Mrs. Brown (Her Majesty, Mrs. Brown) (1997).     Puntuacion: 5.000
5. FairyTale: A True Story (1997).  Puntuacion: 5.000
6. Kiss the Girls (1997).   Puntuacion: 5.000
7. Day the Earth Stood Still, The (1951).   Puntuacion: 5.000
8. Nixon (1995).    Puntuacion: 5.000
9. Cry, the Beloved Country (1995).     Puntuacion: 5.000
10. My Fair Lady (1964).    Puntuacion: 5.000
¿Desea exportar las recomendaciones? (S/N): S
Recomendaciones exportadas correctamente.
Fin del programa. Gracias por usar el sistema de recomendación.
```

*Imagen 13. Mostrar vecindario y top 10 recomendaciones.*

```
90. Remains of the Day, The (1993).       Puntuación: 4.434
91. Young Frankenstein (1974).      Puntuación: 4.434
92. Courage Under Fire (1996).      Puntuación: 4.429
93. Chasing Amy (1997).       Puntuación: 4.411
94. Apollo 13 (1995).   Puntuación: 4.399
95. Sabrina (1995).     Puntuación: 4.375
96. Swingers (1996).    Puntuación: 4.372
97. Ghost (1990).       Puntuación: 4.367
98. Pulp Fiction (1994).      Puntuación: 4.350
99. Wrong Trousers, The (1993).     Puntuación: 4.345
100. In & Out (1997).   Puntuación: 4.341
101. Seven Years in Tibet (1997).   Puntuación: 4.341
102. Michael Collins (1996).  Puntuación: 4.341
103. Dolores Claiborne (1994).      Puntuación: 4.335
104. Searching for Bobby Fischer (1993).  Puntuación: 4.335
105. Wallace & Gromit: The Best of Aardman Animation (1996).      Puntuación: 4.335
```

*Imagen 14. Mostrar otras recomendaciones, mostradas en el fichero de texto exportado.*

# 5. Bibliografía.

Durante la realización se han consultado las siguientes fuentes de información:


- [1] Recomendaciones basado en filtrado colaborativo en Python. Stat Developer
- [2] Sistemas de recomendación en Python. Blog.findemor
- [3] MovieLens project 1-2 Collaborative Filtering. Kaggle
- [4] Pearson Correlation Coefficient (r) | Guide & Examples. Scribbr
- [5] Documentación oficial de Pandas 1.5.3.
- [6] Creación de entornos virtuales en Python.