Carlos Landa Vázquez

carlos@carloslanda.com
(34) 662944300

IMDB Top 25 Extractor

2 de noviembre del 2024

Visión general

Este proyecto es un script de Python que toma como entrada un archivo CSV con el top 50 de películas (por defecto IMDB_Top50.csv) y produce un archivo de salida en formato JSON con el top 25 de las películas. El JSON resultante contiene el nombre de la película, su duración en minutos y su posición en la lista. Este proyecto se encuentra alojado en github.

Características

- Entrada: Acepta un archivo CSV con el listado de las películas. Este CSV debe tener formato especificado como el que se encuentra en el archivo de ejemplo.
- Salida: Genera un archivo JSON con el top 25 de películas, con el formato pedido.
- Flags:

  --input: Especifica el archivo CSV de entrada. Si no se proporciona, se utiliza IMDB_Top50.csv por defecto.

  --output: Define el nombre o ruta del archivo JSON de salida. Si no se proporciona, se guarda con el nombre IMDB.json.

  --input-url: Permite descargar un archivo CSV desde una URL. Este archivo se guarda con el nombre especificado en --input y se procesa de la misma manera.

Formato de salida

El archivo JSON de salida contiene una lista de objetos, cada uno representando una película del top 25 con el siguiente formato:

```
{
  "peliculas":[
  {
      "pelicula": "Nombre de la película", "duracion": 120, "puesto": 1
    }, {
      "pelicula": "Otra película", "duracion": 150, "puesto": 2
    }, ...
  ]
}
```

Requisitos

- Python 3.13.0 (establecido en entorno virtual adjunto)
- librería utils (adjunta) para gestión de logs

Uso

Ejecutar el script de la siguiente manera:

```
python main.py [--input IMDB_Top50.csv] [--output salida.json] [--input-url URL]
```

Ejemplos

1. Ejecutar con el archivo CSV por defecto (IMDB_Top50.csv):

```
python main.py
```

2. Especificar un archivo CSV de entrada:

```
python main.py --input otro_archivo.csv
```

3. Especificar un archivo JSON de salida:

```
python main.py --output mi_salida.json
```

4. Descargar un archivo CSV desde una URL:

```
python main.py --input-url
https://drive.google.com/uc?export=download&id=1zue8yX7khIwjm0ooXLKLfyPNRaC
3OFMY --input IMDB_Descargado.csv
```

Nota: En este caso, el archivo CSV se descargará y guardará con el nombre IMDB_Descargado.csv y luego será utilizado como entrada para el proceso.

Código en un solo archivo:

Todo el desarrollo se ha llevado a cabo a través del uso de módulo, pero se ha simplificado para que todo el codigo esté disponible independientemente en main_only.py de forma que también se puede ejecutar por separado usando el comando:

```
python main_only.py
```

Dicho documento contiene el siguiente código:

```
import argparse
import csv
import json
import os
import datetime
import logging
import os
import traceback
from time import sleep
from urllib import request


class LoggerFile:

    logFile = f'./logs/errorlog -
{datetime.datetime.now().strftime('%d-%m-%Y %H:%M:%S')}.log'
    logFolders = []

    def __init__(self, filename=f'./logs/errorlog -
{datetime.datetime.now().strftime('%d-%m-%Y %H:%M:%S')}.log'):
        self.logFile = filename



    def __set_logger(self, lv=logging.DEBUG):
        #Definimos el formato de loggings por defecto
        LOG_FORMAT = '[{asctime}] {levelname} - {message}'

        log_path = self.logFile
        log_directory = ''

        ###Creamos la ruta de directoios de log si no existe
        folders = log_path.split('/')
        folders.pop(-1)
        #print (folders)
```

```
        for x in range(1, len(folders)):
            folders[x]=os.path.join(folders[x-1],folders[x])

        if folders[0]=='.':
            folders.pop(0)

        self.logFolders=folders

        #Creamos las carpetas de los logs en cascada
        for folder in folders:
            try:
                os.stat(folder)
            except:
                os.mkdir(folder)
        ###


        logger = logging.getLogger(__name__)
        logger.setLevel(lv)

        #log_path = os.path.join(log_directory, log_filename)


        file_handler = logging.FileHandler(log_path, encoding='utf-8')
        file_handler.setLevel(lv)

        formatter = logging.Formatter(LOG_FORMAT, style="{")
        file_handler.setFormatter(formatter)

        if(logger.hasHandlers()):
            logger.handlers.clear()

        logger.addHandler(file_handler)
        return logger

    def deleteLogFile(self):
        os.remove(self.logFile)
```

```
    @classmethod
    def add_to_log(cls, level, message):
        try:
            logger = cls.__set_logger(cls)
            if(level=="critical"):
                logger.critical(message)
            elif(level=="debug"):
                logger.debug(message)
            elif(level=="info"):
                logger.info(message)
            elif(level=="warning"):
                logger.warning(message)
            elif(level=="error"):
                logger.error(message)
        except Exception as ex:
            print(traceback.format_exc())
            print(ex)

    @classmethod
    def info(self,msg):
        self.add_to_log(level="info", message=msg)
    @classmethod
    def debug(self,msg):
        self.add_to_log(level="debug", message=msg)
    @classmethod
    def warning(self,msg):
        self.add_to_log(level="warning", message=msg)
    @classmethod
    def error(self,msg):
        self.add_to_log(level="error", message=msg)
    @classmethod
    def critical(self,msg):
        self.add_to_log(level="critical", message=msg)

class ColorLogger:


    def __set_logger(self, lv=logging.DEBUG):
```

```
        #Definimos el formato de loggings por defecto
        LOG_FORMAT = '[{asctime}] {levelname} - {message}'
        #Definimos los colores en ASCII para cada nivel
        FORMATS={
            logging.DEBUG: f"\33[32m{LOG_FORMAT}\33[0m",
            logging.INFO: f"\33[34m{LOG_FORMAT}\33[0m",
            logging.WARNING: f"\33[33m{LOG_FORMAT}\33[0m",
            logging.ERROR: f"\33[31m{LOG_FORMAT}\33[0m",
            logging.CRITICAL: f"\33[35m{LOG_FORMAT}\33[0m"
        }

        class CustomFormatter(logging.Formatter):
            def format(self, record):
                format = FORMATS[record.levelno]
                formatter = logging.Formatter(format, style="{")
                return formatter.format(record)

        logger = logging.getLogger(__name__)
        logger.setLevel(lv)

        handlers = logging.StreamHandler()
        handlers.setFormatter(CustomFormatter())

        if(logger.hasHandlers()):
            logger.handlers.clear()

        logger.addHandler(handlers)
        return logger

        logging.basicConfig(
            level=lv,
            handlers=[handlers]
        )


    @classmethod
    def add_to_log(cls, level, message):
        try:
```

```
            logger = cls.__set_logger(cls)
            if(level=="critical"):
                logger.critical(message)
            elif(level=="debug"):
                logger.debug(message)
            elif(level=="info"):
                logger.info(message)
            elif(level=="warning"):
                logger.warning(message)
            elif(level=="error"):
                logger.error(message)
        except Exception as ex:
            print(traceback.format_exc())
            print(ex)

    @classmethod
    def info(self,msg):
        self.add_to_log(level="info", message=msg)
    @classmethod
    def debug(self,msg):
        self.add_to_log(level="debug", message=msg)
    @classmethod
    def warning(self,msg):
        self.add_to_log(level="warning", message=msg)
    @classmethod
    def error(self,msg):
        self.add_to_log(level="error", message=msg)
    @classmethod
    def critical(self,msg):
        self.add_to_log(level="critical", message=msg)



logfile = LoggerFile()


def createJson(data, fname = "IMDB.json"):
    try:
```

```
        #comprobación si existe el archivo anteriormente
        if os.path.isfile(fname):
            ColorLogger.info(f'Borrando archivo {fname} para regenerarlo')
            logfile.info(f'Borrando archivo {fname} para regenerarlo')
            os.remove(fname)

        #generación del archivo
        ColorLogger.info(f"Generando {fname}")
        logfile.info(f"Generando {fname}")



        out_path = fname
        ###Creamos la ruta de directoios de out si no existe
        folders = out_path.split('/')
        folders.pop(-1)
        ColorLogger.debug(f"Carpetas a crear:{folders}")
        logfile.debug(f"Carpetas a crear:{folders}")
        if folders!=[]:
            for x in range(1, len(folders)):
                folders[x]=os.path.join(folders[x-1],folders[x])

            if folders[0]=='.':
                folders.pop(0)

            #Creamos las carpetas de los output en cascada
            ColorLogger.debug(f"Carpetas a crear:{folders}")
            logfile.debug(f"Carpetas a crear:{folders}")
            for folder in folders:
                try:
                    os.stat(folder)
                except:
                    os.mkdir(folder)
        ###
```

```
        with open(fname, "w") as j:
            json.dump(data, j, indent=4)
    except Exception as ex:
        ColorLogger.error(f"Error al crear el archivo {fname}.")
        logfile.error(f"Error al leer el archivo {fname}.")
        exit(1)

def csvToArray(csv_file='IMDB_Top50.csv', csv_file_url = '',
top_length=25):
    #añadir comprobción de que el archivo IMDB_Top.csv existe y sino ir a
buscarlos a internet
(https://drive.google.com/file/d/1zue8yX7khIwjm0ooXLKLfyPNRaC3OFMY/view?usp
=sharing)

    #leyendo archivo csv
    ColorLogger.info(f'Leyendo archivo {csv_file}')
    logfile.info(f'Leyendo archivo {csv_file}')
    try:
        array = []
        in_path = csv_file
        ###Creamos la ruta de directoios de input si no existe
        folders = in_path.split('/')
        folders.pop(-1)
        ColorLogger.debug(f"Carpetas a crear:{folders}")
        logfile.debug(f"Carpetas a crear:{folders}")
        if folders!=[]:
            for x in range(1, len(folders)):
                folders[x]=os.path.join(folders[x-1],folders[x])

            if folders[0]=='.':
                folders.pop(0)

            #Creamos las carpetas de los output en cascada
            ColorLogger.debug(f"Carpetas a crear:{folders}")
            logfile.debug(f"Carpetas a crear:{folders}")
            for folder in folders:
                try:
                    os.stat(folder)
```

```
                except:
                    os.mkdir(folder)
        ###


        if (csv_file_url!=''):
            request.urlretrieve(csv_file_url, csv_file)

        if (not os.path.exists(csv_file)):
            csv_file_url =
"https://drive.google.com/uc?export=download&id=1zue8yX7khIwjm0ooXLKLfyPNRa
C3OFMY"
            ColorLogger.info(f'No se ha podido encontrar el archivo
{csv_file}, asi que procederé a descargarlo de {csv_file_url}')
            logfile.info(f'No se ha podido encontrar el archivo {csv_file},
asi que procederé a descargarlo de {csv_file_url}')
            request.urlretrieve(csv_file_url, csv_file)


        with open(file=csv_file) as f:
            csv_reader = csv.reader(f, delimiter=',')
            position = 0
            for row in csv_reader:
                if position:
                    split_time = row[0].split("H")
                    split_time[0] = int(split_time[0])*60
                    split_time[1] = int(split_time[1][:-1])
                    minutes = split_time[0]+split_time[1]

                    if (position <= top_length):
                        ColorLogger.debug("Posicion: {0}, Duración: {1},
Nombre: {2}".format(position, minutes, row[1]))
                        logfile.debug("Posicion: {0}, Duración: {1},
Nombre: {2}".format(position, minutes, row[1]))
                        array.append({"pelicula":row[1],
"duracion":minutes, "puesto":position})
                position += 1
            return array
```

```
    except Exception as ex:
        ColorLogger.error(f"Error al leer el archivo {csv_file}.")
        logfile.error(f"Error al leer el archivo {csv_file}.")
        exit(1)

def main():
    #paso de parametros al script
    parse = argparse.ArgumentParser()
    parse.add_argument("--input",
type=str,default="assets/IMDB_Top50.csv",help="ruta de acceso al archivo
.csv del cual se sacan los datos.")
    parse.add_argument("--input-url", type=str,default="",help="url al
archivo .csv del cual se sacan los datos.")
    parse.add_argument("--output",
type=str,default="assets/IMDB.json",help="ruta de acceso al archivo .json
al cual creará con los datos de salida.")
    args=parse.parse_args()
    ColorLogger.debug("Parametros de entrada: {0}".format(args))
    logfile.debug("Parametros de entrada: {0}".format(args))
    #print(args)


    #ejecución del programa
    data = {"peliculas":csvToArray(csv_file=str(args.input),
csv_file_url=str(args.input_url))}
    createJson(data, fname=str(args.output))
    ColorLogger.info("----------------------DATOS DE
SALIDA----------------------")
    print(data)
    logfile.info("----------------------DATOS DE
SALIDA----------------------")
    logfile.info(data)
    #borramos el logfile si no ha ocurrido ningun error que haga exit(1)
    logfile.deleteLogFile()

if __name__ == '__main__':
    main()
```