Deezer Visualización y Análisis de Datos Musicales

**Profesor: Simón Poblete Fuentes**

**Alumnos: Cristian Caro Raio**

**Gonzalo Fernández**

**Luis Pérez**

**Vicente López**

Introducción

Este código es una aplicación web interactiva desarrollada utilizando Streamlit y varias bibliotecas de Python, como requests, pandas, plotly.express, y numpy. El objetivo de la aplicación es mostrar los Top Charts de Deezer, es decir, las canciones, álbumes, artistas, listas de reproducción y podcasts más populares según los datos de Deezer. A continuación, se proporciona una explicación detallada de cada parte del código.

1. **Importación de Librerías**

# importacion de las librerias

1. import streamlit as st
2. import requests
3. import pandas as pd
4. import plotly.express as px
5. import numpy as np

* **Streamlit (st)**: Se usa para crear la interfaz de usuario de la aplicación web.
* **Requests**: Permite realizar solicitudes HTTP para obtener los datos desde una API.
* **Pandas (pd)**: Se utiliza para manejar los datos en forma de DataFrame.
* **Plotly Express (px)**: Se usa para generar gráficos interactivos.
* **NumPy (np)**: Utilizado para operaciones numéricas, especialmente en la creación de matrices y cálculos de correlación.

**2. Configuración de la Página**

Esta línea configura la página web de la aplicación:

* **page\_title**: El título que aparecerá en la pestaña del navegador.
* **page\_icon**: El ícono que aparecerá junto al título en la pestaña (en este caso, un ícono musical 🎵).
* **layout**: Define el diseño de la página, en este caso, en formato "wide" (amplio).
* # Configuración de la página
* st.set\_page\_config(page\_title="Deezer Top Charts", page\_icon="🎵", layout="wide")

**3. Título y Descripción**

* **st.title**: Muestra el título principal de la aplicación.
* **st.markdown**: Permite agregar texto en formato Markdown, lo que incluye una breve descripción de la aplicación.
* # Título y descripción
* st.title("🎵 Deezer Top Charts")
* st.markdown("Explora las canciones, álbumes, artistas y más populares según los charts oficiales de Deezer.")

**4. Sidebar: Selección de Categoría**

st.sidebar.header: Agrega un encabezado en la barra lateral de la aplicación.

st.sidebar.selectbox: Permite al usuario seleccionar una categoría de entre las opciones proporcionadas. Esto puede ser una de las siguientes categorías:

* Canciones
* Álbumes
* Artistas
* Playlists (Listas de reproducción)
* Podcasts
* # Sidebar: Selección de categoría
* st.sidebar.header("Opciones")
* opcion\_categoria = st.sidebar.selectbox("Selecciona una categoría",
* ["Canciones", "Álbumes", "Artistas", "Playlists", "Podcasts"])

5. **Mapeo de Categorías a Campos de API**

* Aquí, se define un diccionario que mapea cada opción de categoría seleccionada por el usuario a su correspondiente campo de la API. Según la categoría seleccionada (por ejemplo, "Canciones"), se asigna el campo adecuado ("tracks").
* endpoint\_fields = {
* "Canciones": "tracks",
* "Álbumes": "albums",
* "Artistas": "artists",
* "Playlists": "playlists",
* "Podcasts": "podcasts",
* }
* campo\_api = endpoint\_fields[opcion\_categoria]

1. **Función para Obtener Datos del Chart**

* **@st.cache**: Decorador de Streamlit que almacena en caché el resultado de la función para evitar solicitudes repetidas.
* **requests.get(url)**: Realiza una solicitud HTTP GET a una API local que proporciona datos sobre el chart de Deezer.
* **response.json()**: Convierte la respuesta de la solicitud a un formato JSON.
* **get(campo\_api, {})**: Obtiene los datos correspondientes a la categoría seleccionada.
* Si no se encuentran datos para esa categoría, se devuelve un DataFrame vacío.
* # Función para obtener datos del chart
* @st.cache
* def obtener\_datos\_chart(campo\_api):
* try:
* url = "http://localhost:8080?endpoint=chart"  # URL del backend
* response = requests.get(url)
* response.raise\_for\_status()
* data = response.json().get(campo\_api, {}).get("data",[])

1. **Validación y Visualización de Datos**

* **df.empty**: Verifica si el DataFrame obtenido está vacío (es decir, si no se encontraron datos para la categoría seleccionada).
* Si está vacío, la aplicación se detiene con **st.stop()**.
* # Obtener datos de la categoría seleccionada
* df = obtener\_datos\_chart(campo\_api)
* # Validar si hay datos
* if df.empty:
* st.stop()

**9. Mostrar Tabla Interactiva**

*  **st.subheader**: Muestra un subtítulo con el nombre de la categoría seleccionada.
*  **st.dataframe**: Muestra los datos obtenidos como una tabla interactiva.
* # Mostrar tabla interactiva
* st.subheader(f"Top {opcion\_categoria}")
* st.dataframe(df, use\_container\_width=True)
  1. **Generación de Gráficos para "Canciones"**
* Si la categoría seleccionada es **"Canciones"**, la aplicación genera varios gráficos interactivos utilizando **Plotly Express**:
* **Histograma de Duración**: Muestra la distribución de las duraciones de las canciones en un histograma.
* fig\_hist = px.histogram(df, x="Duración (s)", nbins=10, title="Distribución de la Duración de Canciones")
* st.plotly\_chart(fig\_hist, use\_container\_width=True)
* **Gráfico de Barras (Duración por Canción)**: Muestra un gráfico de barras donde se visualiza la duración de las canciones agrupadas por artista.
* st.subheader("Gráfico de Barras: Duración por Canción")
* fig\_bar = px.bar(df, x="Título", y="Duración (s)", color="Artista",
* title="Duración de Canciones por Artista", labels={"Duración (s)": "Duración (segundos)"})
* st.plotly\_chart(fig\_bar, use\_container\_width=True)
* **Gráfico de Líneas (Duración por Posición)**: Representa cómo la duración de las canciones cambia a medida que varía su posición en el ranking.
* # Gráfico de líneas
* st.subheader("Gráfico de Líneas: Duración por Posición")
* fig\_line = px.line(df, x="Posición", y="Duración (s)", title="Duración por Posición en el Ranking")
* st.plotly\_chart(fig\_line, use\_container\_width=True)
* **Diagrama de Dispersión (Posición vs. Duración)**: Muestra la relación entre la posición en el ranking y la duración de las canciones.
* # Diagrama de dispersión
* st.subheader("Dispersión: Posición vs. Duración")
* fig\_scatter = px.scatter(df, x="Posición", y="Duración (s)", color="Artista",
* title="Dispersión: Duración vs. Posición", size="Duración (s)")
* st.plotly\_chart(fig\_scatter, use\_container\_width=True)
* **Mapa de Calor (Correlación entre Variables)**: Si hay suficientes columnas numéricas, se genera un mapa de calor que muestra las correlaciones entre las variables del DataFrame.
* # Mapa de calor
* st.subheader("Mapa de Calor: Correlación entre Variables")
* if len(df.columns) > 2:  # Asegurarse de tener suficientes columnas
* numeric\_cols = df.select\_dtypes(include=[np.number])
* correlation\_matrix = numeric\_cols.corr()
* fig\_heatmap = px.imshow(correlation\_matrix, text\_auto=True, title="Mapa de Calor: Correlaciones")
* st.plotly\_chart(fig\_heatmap, use\_container\_width=True)

Conclusion

Este código permite interactuar con los datos de Deezer y explorar sus **Top Charts** a través de una interfaz sencilla, donde el usuario puede seleccionar la categoría de interés (canciones, álbumes, artistas, playlists o podcasts). Además, para la categoría de canciones, genera una variedad de gráficos interactivos que facilitan la visualización de la duración de las canciones y su relación con otras variables. La aplicación es eficiente gracias al uso de la caché en Streamlit, y proporciona una experiencia visual y dinámica a través de los gráficos generados con **Plotly**.