Dijiste:

```
from scrapper import scrape_yogonet from processor import data_processing from big_quer_client
import upload_to_bigquery if __name__ == "__main__": data =
scrape_yogonet(url="https://www.yogonet.com/international/") print(data) df =
data_processing(data) upload_to_bigquery(df) Could you give some commetns to add explaining
the code. Be concise
2/2
ChatGPT dijo:
```

Sure! Here's your script with concise comments added to explain each part:

```
CopiarEditar
# Import main functions from respective modules
from scrapper import scrape_yogonet
                                                # Web scraping function
from processor import data_processing
                                                # Data cleaning/transformation
from big_quer_client import upload_to_bigquery # BigQuery upload function
if __name__ == "__main__":
    # Scrape news data from Yogonet's international section
   data = scrape_yogonet(url="https://www.yogonet.com/international/")
   # Print raw scraped data for debugging
   print(data)
   # Process and structure the data into a DataFrame
   df = data_processing(data)
   # Upload the processed data to Google BigQuery
   upload_to_bigquery(df)
```

Dijiste:

FROM python:3.11-slim WORKDIR /app # Instalar dependencias de Python COPY requirements.txt . RUN pip install --no-cache-dir -r requirements.txt # Copiar la app COPY app/ ./app/ # Ejecutar el script principal CMD ["python", "app/main.py"] Estoy utilizando la librería selenium en python para hacer scrapping. Estoy recibiendo el siguiente error: -central1 2025-04-12 14:42:50 Traceback (most recent call last): File "/usr/local/lib/python3.11/site-packages/selenium/webdriver/common/driver_finder.py", line 64, in

_binary_paths raise ValueError(f"The path is not a valid file: {path}") ValueError: The path is not a valid file: /usr/local/bin/chromedriver 2025-04-12 14:42:50 The above exception was the direct cause of the following exception: 2025-04-12 14:42:50 Traceback (most recent call last): File "/app/app/main.py", line 7, in <module> data =

scrape_yogonet(url="https://www.yogonet.com/latinoamerica/") 2025-04-12 14:42:50 ^^^^^^^ 2025-04-12 14:42:50 File

"/app/app/scrapper.py", line 16, in scrape_yogonet 2025-04-12 14:42:50 driver = webdriver. Chrome(service=service, options=options) 2025-04-12 14:42:50 ^^^^^^ 2025-04-12 14:42:50 File

"/usr/local/lib/python3.11/site-packages/selenium/webdriver/chrome/webdriver.py", line 45, in __init__ 2025-04-12 14:42:50 super().__init__(2025-04-12 14:42:50 File

https://www.selenium.dev/documentation/webdriver/troubleshooting/errors/driver_location NoSuchDriverException. This works in my local environment, but not when I run it from a Docker Image

ChatGPT dijo:

What's going on?

In your local environment, you're probably using Chrome installed on your system along with Chromedriver, and everything works fine.

But inside the Docker container, Chrome and Chromedriver are not included by default, so Selenium can't find the Chrome binary or the driver to control it.

Solution: Include Chrome and Chromedriver in your Docker image

Let's update your Dockerfile so that it:

- 1. Installs Google Chrome (stable version)
- 2. Installs a compatible ChromeDriver
- 3. Runs correctly in headless mode (no GUI)

Updated Dockerfile (with Chrome + Chromedriver)

Replace your current Dockerfile with this one:

```
dockerfile
CopiarEditar
FROM python:3.11-slim
WORKDIR /app
# Install system dependencies
RUN apt-get update && apt-get install -y \
    wget \
    unzip \
    curl \
    gnupg \
    ca-certificates \
    fonts-liberation \
    libnss3 \
    libxss1 \
    libappindicator3-1 \
    libasound2 \
    libatk-bridge2.0-0 \
    libatk-3-0 \
    chromium \
    chromium-driver \
    --no-install-recommends && \
```

```
apt-get clean && rm -rf /var/lib/apt/lists/*

# Install Python dependencies
COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

# Copy the app
COPY app/ ./app/

# Run the main script
CMD ["python", "app/main.py"]
```

Additional a

🔧 Additional adjustment in <code>scrapper.py</code> (if not already added):

In your code, make sure you set the options correctly:

```
python
CopiarEditar
from selenium import webdriver
from selenium.webdriver.chrome.options import Options

def get_driver():
    options = Options()
    options.add_argument("--headless")
    options.add_argument("--no-sandbox")
    options.add_argument("--disable-dev-shm-usage")
    options.add_argument("--disable-gpu")
    options.add_argument("--remote-debugging-port=9222")
    return webdriver.Chrome(options=options)

And in your scrape_yogonet() function, use:

python
CopiarEditar
driver = get_driver()
```

Dijiste:

Necesito que me armes un script en basch para que mi entorno este listo para desplegar en GCP.

Tiene que revisar si tengo gcloud, docker, un proyecto activo, el env cargado y las siguientes variables: PROJECT_ID, DATASET_ID, TABLE_ID. Si falta algo debe tirar errores que describan el error

¡Perfecto! Te armo a continuación **el setup. sh completo**, limpio y listo para uso o entrega:



setup.sh — Código completo

```
bash
CopiarEditar
#!/bin/bash

echo ""
echo "Q Iniciando verificación del entorno de despliegue..."

# Verificar que gcloud esté instalado
if ! command -v gcloud &> /dev/null; then
   echo "X ERROR: 'gcloud' no está instalado. Instalalo desde
https://cloud.google.com/sdk"
```

```
exit 1
# Verificar que docker esté instalado
if ! command -v docker &> /dev/null; then
  echo "X ERROR: 'docker' no está instalado. Instalalo desde
https://docs.docker.com/get-docker/"
  exit 1
fi
# Verificar autenticación en gcloud
AUTH_USER=$(gcloud auth list --filter=status:ACTIVE --format="value(account)")
if [[ -z "$AUTH_USER" ]]; then
echo "★ ERROR: No estás autenticado en Google Cloud."
echo "➡ Ejecutá: gcloud auth login"
  exit 1
else
  echo "☑ Usuario autenticado en gcloud: $AUTH_USER"
# Verificar proyecto seleccionado
PROJECT_ID=$(gcloud config get-value project 2>/dev/null)
if [[ -z "$PROJECT_ID" || "$PROJECT_ID" == "(unset)" ]]; then
  echo "ERROR: No hay un proyecto seleccionado en gcloud." echo "Ejecutá: gcloud config set project TU_PROJECT_ID"
  exit 1
else
  echo "✓ Proyecto activo: $PROJECT_ID"
# Verificar que exista el archivo .env
if [ ! -f .env ]; then
  echo "X ERROR: No se encontró el archivo '.env'."
  echo " Copiá el ejemplo: cp .env.example .env y completalo."
  exit 1
fi
# Cargar variables del archivo .env
export $(cat .env | xargs)
# Validar variables necesarias
MISSING=false
for var in DATASET_ID TABLE_ID SERVICE_ACCOUNT_EMAIL REGION; do
  if [[ -z "${!var}" ]]; then
  echo "★ ERROR: Falta definir $var en el archivo .env"
    MISSING=true
  fi
done
if [ "$MISSING" = true ]; then
  echo " Por favor completá todas las variables requeridas en .env"
  exit 1
fi
# Mostrar resumen
echo ""
echo "
         Resumen de configuración:"
echo "
                                   $PROJECT_ID"
          Proyecto GCP:
echo "
          Dataset de BigQuery: $DATASET_ID"
echo "
                                 $TABLE_ID"
          Tabla de BigQuery:
echo "🔽 Entorno validado correctamente. Podés ejecutar ./deploy.sh"
```

Para usarlo

bash
CopiarEditar
chmod +x setup.sh
./setup.sh

Dijiste:

Ahora necesito armar un script en bash que haga todo el deploy a Cloud Run.

Tiene que validar el setup.sh, cargar las variables desde .env, crear el repositorio en Artifact Registry sin existe, y despues hacer el build y push de Docker. Después, actualizar un Job en Cloud Run con esas mismas variables de entorno. Por ultimo, lo tiene que ejecutar. Agregá comentarios describiendo los pasos que van ocurriendo. Las variables son: PROJECT_ID, DATASET_ID, TABLE_ID, REGION, REPO_NAME, SERVICE_NAME.

ChatGPT dijo:

¡Perfecto! Acá va tu deploy . sh completo:

```
bash
CopiarEditar
#!/bin/bash
# Validación del entorno
./setup.sh
if [ $? -ne 0 ]; then
  echo "Error: Validación fallida."
  exit 1
fi
echo "Validación superada. Continuando con build y deploy..."
# Cargar variables desde .env
set -a
source .env
set +a
# Configuración
PROJECT_ID=$(gcloud config get-value project)
REGION=${REGION: -<TU-REGION>}
REPO_NAME=${REPO_NAME:-<TU-REPO>}
SERVICE_NAME=${SERVICE_NAME:-<TU-SERVICIO>}
IMAGE_NAME="$REGION-docker.pkg.dev/$PROJECT_ID/$REPO_NAME/$SERVICE_NAME"
echo "Variables cargadas:"
echo "PROJECT_ID=$PROJECT_ID"
echo "DATASET_ID=$DATASET_ID"
echo "TABLE_ID=$TABLE_ID"
echo "REGION=$REGION"
echo "REPO_NAME=$REPO_NAME"
echo "SERVICE_NAME=$SERVICE_NAME"
# Verificar que el repositorio de Artifact Registry exista
echo "Verificando repositorio en Artifact Registry..."
if ! gcloud artifacts repositories describe $REPO_NAME --location=$REGION
&>/dev/null; then
  echo "Repositorio $REPO_NAME no existe. Creando..."
  gcloud artifacts repositories create $REPO_NAME \
    --repository-format=docker \
```

```
--location=$REGION \
    --description="Repositorio Docker para el proyecto Yogonet"
else
  echo "Repositorio $REPO_NAME ya existe."
fi
# Build Docker image
echo "Construyendo imagen Docker..."
docker build -t $IMAGE_NAME .
# Configurar autenticación Docker
echo "Configurando autenticación Docker..."
gcloud auth configure-docker $REGION-docker.pkg.dev
# Push a Artifact Registry
echo "Subiendo imagen a Artifact Registry..."
docker push $IMAGE_NAME
# Crear o actualizar Cloud Run Job
echo "Verificando Job..."
gcloud beta run jobs describe $SERVICE_NAME --region=$REGION &>/dev/null
if [ $? -ne 0 ]; then
  echo "Creando nuevo Job en Cloud Run..."
  gcloud beta run jobs create $SERVICE_NAME \
    --image $IMAGE NAME \
    --region $REGION \
    --memory 1024Mi \
    --set-env-vars
PROJECT_ID=$PROJECT_ID, DATASET_ID=$DATASET_ID, TABLE_ID=$TABLE_ID
e1se
  echo "Actualizando Job existente..."
  gcloud beta run jobs update $SERVICE_NAME \
    --image $IMAGE_NAME \
    --region $REGION \
    --memory 1024Mi \
    --set-env-vars
PROJECT_ID=$PROJECT_ID, DATASET_ID=$DATASET_ID, TABLE_ID=$TABLE_ID
fi
# Ejecutar el Job
echo "Ejecutando Job en Cloud Run..."
gcloud beta run jobs execute $SERVICE_NAME --region=$REGION
echo "Job ejecutado correctamente."
```

Dijiste:

Necesito un README.md para un proyecto que hace scraping web, procesa los datos con Pandas y los sube a BigQuery. Todo corre en un contenedor Docker que se ejecuta como un Job en Cloud Run.

El README debería incluir:

- Una descripción general del proyecto
- Qué herramientas usa (Selenium, Pandas, BigQuery, Docker, Cloud Run)
- Requisitos previos (gcloud, APIs habilitadas, permisos, etc.)
- Cómo configurar el entorno con un archivo .env
- Cómo desplegar todo con un script deploy. sh

Cómo ver los resultados del Job y consultar los datos en BigQuery

Que el tono sea claro y algo técnico, pero fácil de seguir para alguien que quiere correr el proyecto desde cero.

ChatGPT dijo:

Claro, acá va una versión genérica y editable del README.md, con el mismo estilo técnico pero relajado, y dejando los campos específicos para que vos los completes:



Web Scraper - Cloud Run Job

Este proyecto ejecuta un scraper automatizado que recolecta noticias desde un portal web, procesa los datos con Python y los sube a una tabla de BigQuery. Todo está empaquetado en Docker y corre como un Cloud Run Job.



🚀 ¿Qué hace este proyecto?

- 1. Usa **Selenium** para navegar por el sitio web y extraer las noticias.
- 2. Procesa los datos con **Pandas** (por ejemplo: contar palabras en los títulos, analizar texto, etc.).
- 3. Inserta los resultados en una tabla de **BigQuery**.
- 4. Corre todo como un Job en **Cloud Run**, utilizando una imagen Docker personalizada.

Requisitos previos

Antes de desplegar el proyecto, asegurate de tener:

- Google Cloud CLI (gcloud) instalado y autenticado
- Permisos para:
 - Crear y subir imágenes a Artifact Registry
 - Usar Cloud Run Jobs
 - Escribir en **BigQuery**
- Las siguientes APIs habilitadas en tu proyecto de GCP:

bash CopiarEditar

gcloud services enable artifactregistry.googleapis.com run.googleapis.com bigquery.googleapis.com



1. Clonar el repositorio

bash CopiarEditar ait clone https://github.com/<tu-usuario>/<nombre-del-repo>.git cd <nombre-del-repo>

2. Crear el archivo . env

En la raíz del proyecto, creá un archivo . env con estas variables (ajustalas a tu entorno):

env CopiarEditar PROJECT_ID=<tu-id-de-proyecto> DATASET_ID=<nombre-del-dataset> TABLE_ID=<nombre-de-la-tabla> REGION=<region-gcp> REPO_NAME=<nombre-del-repo-en-artifact-registry> SERVICE_NAME=<nombre-del-cloud-run-job>

🧠 Tip: Verificá que el PROJECT_ID coincida con el proyecto activo usando:

bash CopiarEditar gcloud config get-value project



Ejecutá el script de despliegue:

bash CopiarEditar ./deploy.sh

Este script:

- Valida el entorno (gcloud, docker, variables .env) usando setup.sh
- Construye y sube la imagen Docker a Artifact Registry
- Crea o actualiza el Job en Cloud Run
- Ejecuta el Job con los valores definidos en el .env



■ Ver los resultados

Consultar ejecuciones del Job

bash CopiarEditar gcloud beta run jobs executions list --region=\$REGION --job=\$SERVICE_NAME También podés seguirlo desde la consola de Google Cloud → Cloud Run → Jobs.

Ver los datos en BigQuery

- 1. Entrá a BigQuery desde la consola de GCP.
- 2. Seleccioná el proyecto y dataset definidos en tu .env.
- 3. Ejecutá una consulta sobre la tabla para ver los datos.

Extras

- setup.sh: script que valida el entorno antes de desplegar.
- deploy. sh: script que automatiza todo el flujo de despliegue.
- Todo el proyecto corre desde un contenedor, así que no requiere instalación local de dependencias Python.

Si necesitás adaptar este README a un caso más específico (nombres, estructura, logging, etc.), avisame y lo ajustamos.

Dijiste:

Can you give me a prompt for an LLM to extract structured news article data (title, kicker, image URL, and article link) from raw HTML? Make sure it includes detailed instructions on how to prioritize tags, handle missing fields, and format the output as JSON.

ChatGPT dijo:

llm_prompt = PromptTemplate.from_template(""" You are an intelligent assistant specialized in extracting structured and meaningful information from raw HTML. Below, you will find a raw HTML snippet. Your job is to analyze this HTML and extract the following fields related to a news article: - **Title**: The main headline of the article. Prefer <h1>, <h2>, or elements with class names like "title", "headline", or similar. - **Kicker**: A short introductory phrase or category often found above the title (sometimes labeled as "kicker" or "volanta"). - **Image URL**: The URL of the most representative image related to the article. Look for tags near the title or in a container for the article. - **Article Link**: The main URL (usually inside) that leads to the full article. ### Input HTML: {html} ### Instructions: - Focus on extracting values that are **complete**, **readable**, and **clearly related to a news article**. - Avoid template content, unrelated links, or empty tags. - If multiple options exist, **choose the one that best matches a real article** based on context (e.g., close to a headline or kicker). - If any field is not found, return "Not found" for that field. - Present your final output in this format: json {

```
"title": "...",
  "kicker": "...",
  "image_url": "...",
  "link": "..."
}
```

Dijiste:

Can you help me build a LangGraph-based Python script that scrapes HTML from a news site and extracts structured data like title, kicker, image URL, and link using tools (functions) instead of a language model? I want to define each tool with its own logic using BeautifulSoup, and wrap the whole thing in a ScraperState class that tracks messages. Use LangChain and LangGraph best practices.

```
ChatGPT dijo:
def extract_title(html: str) -> str:
  Extracts the most probable news title from the HTML content.
  Args:
     html (str): The HTML content of the web page.
  Returns:
     str: The extracted title, or empty string if not found.
  ,,,,,,
  soup = BeautifulSoup(html, "html.parser")
  candidates = soup.find_all(["h1", "h2", "div"])
  best_score = 0
  best_candidate = None
  for tag in candidates:
     text = tag.get_text(strip=True)
     if not text or len(text) < 15:
       continue
     score = 0
     if tag.name == "h1":
       score += 2
     elif tag.name == "h2":
       score += 1
```

```
keywords = ["casino", "gaming", "launch", "opens", "regulation", "market"]
     for kw in keywords:
       if kw in text.lower():
          score += 2
     if len(text) > 150:
       score -= 1
     if score > best_score:
       best_score = score
       best_candidate = tag
  return best_candidate.get_text(strip=True) if best_candidate else ""
def extract_kicker(html: str) -> str:
  Extracts a short headline or context (kicker) from the HTML.
  Args:
     html (str): The HTML content of the web page.
  Returns:
     str: The kicker text, or empty string if not found.
  ,,,,,,
  soup = BeautifulSoup(html, "html.parser")
  divs = soup.find_all("div")
  for d in divs:
     text = d.get_text(strip=True)
     if text and len(text.split()) < 10 and len(text) > 5:
       return text
```

```
def extract_image(html: str) -> str:
  Extracts the most relevant image URL (usually from article or banner).
  Args:
     html (str): The HTML content of the web page.
  Returns:
     str: The image URL, or empty string if not found.
  ,,,,,,
  soup = BeautifulSoup(html, "html.parser")
  images = soup.find_all("img")
  for img in images:
     src = img.get("src")
     if src and re.search(r"(casino|gaming|event|banner).*\\.(jpg|png)", src, re.IGNORECASE):
       return src
  return images[0].get("src") if images else ""
def extract_link(html: str) -> str:
  ,,,,,,
  Extracts the main article or news link from the HTML.
  Args:
     html (str): The HTML content of the web page.
  Returns:
     str: The URL (href) of the most relevant link, or empty string.
  ******
```

return ""

```
soup = BeautifulSoup(html, "html.parser")
links = soup.find_all("a", href=True)
for a in links:
    text = a.get_text(strip=True).lower()
    if "read more" in text or "ver más" in text or len(text) > 10:
        return a["href"]
return links[0]["href"] if links else ""
```