https://github.com/Fetiiii/evds-datahub
Detaylı bir şekilde buradan bakabilirsiniz.
Datalar githubta mevcut, çekildikten sonra githuba push edildi.

```
#!/usr/bin/env python3

import os
import re
import sys
import time
import requests
import pandas as pd
from datetime import datetime, timedelta
from evds import evdsAPI

# ---------- AYARLAR ----------
API_KEY = os.getenv("EVDS_API_KEY")
DATA_DIR = "data"
LOG_DIR = "logs"
SLEEP_BETWEEN_SERIES = 1
UPDATE_MODE = "--update" in sys.argv
UPDATE_DAYS = 3  # Güncelleme modunda son X gün
# -----------------------------

if not API_KEY:
    raise SystemExit(" EVDS_API_KEY bulunamadı. Ortam değişkeni olarak
tanımla.")

os.makedirs(DATA_DIR, exist_ok=True)
os.makedirs(LOG_DIR, exist_ok=True)
evds = evdsAPI(API_KEY)

# ---------- FONKSİYONLAR ----------

def log_failed_series(code, serie_name, category, reason):
    """Başarısız olan serileri logs/failed_series.txt dosyasına
kaydeder."""
    log_file = os.path.join(LOG_DIR, "failed_series.txt")
    with open(log_file, "a", encoding="utf-8") as f:
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        f.write(f"[{timestamp}] {category} | {serie_name} | {code} |
{reason}\n")

def safe_get_main_categories():
    try:
        return evds.main_categories
    except Exception as e:
        print(f" main_categories hata: {e}")
        return None

def safe_get_sub_categories(cat_id):
    try:
```

```
        return evds.get_sub_categories(cat_id)
    except Exception as e:
        print(f" get_sub_categories hata (CATEGORY_ID={cat_id}): {e}")
        return None

def safe_get_series(datagroup_code):
    try:
        return evds.get_series(datagroup_code)
    except Exception as e:
        print(f" get_series hata (DATAGROUP_CODE={datagroup_code}): {e}")
        return None

def safe_get_data(code, serie_name=None, category=None, retries=3,
delay=5):
    """EVDS'ten veri çeker, bağlantı hatalarında tekrar dener."""
    for attempt in range(retries):
        try:
            if UPDATE_MODE:
                end = datetime.now()
                start = end - timedelta(days=UPDATE_DAYS)
                start_str = start.strftime("%d-%m-%Y")
                end_str = end.strftime("%d-%m-%Y")
                print(f"      ↪ Güncelleme aralığı: {start_str} →
{end_str}")
            else:
                start_str = "01-01-2000"
                end_str = datetime.now().strftime("%d-%m-%Y")

            return evds.get_data([code], startdate=start_str,
enddate=end_str)

        except requests.exceptions.ConnectionError as e:
            print(f" Bağlantı hatası ({code}), {attempt+1}. deneme: {e}")
            time.sleep(delay)

        except Exception as e:
            if "Too Many Requests" in str(e):
                print(" API limitine ulaşıldı, 60 sn bekleniyor...")
                time.sleep(60)
                continue
            print(f" get_data hata ({code}): {e}")
            break

    print(f" {code}: {retries} denemede veri alınamadı.")
    log_failed_series(code, serie_name or "Bilinmiyor", category or
"Bilinmiyor", "ConnectionError veya boş veri")
    return None

def normalize_df(df, code):
    if df is None or df.empty:
        return None

    if "Tarih" not in df.columns and "DATE" in df.columns:
        df = df.rename(columns={"DATE": "Tarih"})
```

```
    if "Tarih" not in df.columns:
        print(f" {code}: Tarih sütunu yok.")
        return None

    df["Tarih"] = pd.to_datetime(df["Tarih"], dayfirst=True,
errors="coerce")
    df = df.dropna(subset=["Tarih"])

    other_cols = [c for c in df.columns if c != "Tarih"]
    if not other_cols:
        print(f" {code}: Veri sütunu bulunamadı.")
        return None

    series_col = other_cols[0]
    df = df.rename(columns={series_col: code.replace(".", "_")})
    return df[["Tarih", code.replace(".", "_")]]

def clean_filename(name):
    """Dosya veya klasör isimlerinde sorun çıkaracak karakterleri
temizler"""
    return re.sub(r'[\\/*?:"<>|]', "", str(name))

def append_or_create_csv(series_name, df, main_category, sub_category):
    """CSV dosyasını klasör yapısına göre kaydeder veya günceller"""
    main_dir = os.path.join(DATA_DIR, clean_filename(main_category))
    sub_dir = os.path.join(main_dir, clean_filename(sub_category))
    os.makedirs(sub_dir, exist_ok=True)

    fname = os.path.join(sub_dir, f"{clean_filename(series_name)}.csv")
    # Uzun yol desteği (Windows)
    if os.name == "nt":
        fname = "\\\\?\\" + os.path.abspath(fname)

    #  Skip kontrolü: Dosya zaten varsa ve update modunda değilsek, atla
    if os.path.exists(fname) and not UPDATE_MODE:
        print(f"     {series_name}: zaten mevcut, atlanıyor.")
        return

    if df is None or df.empty:
        print(f"     {series_name}: yeni veri yok.")
        return

    if not os.path.exists(fname):
        df.to_csv(fname, index=False, encoding="utf-8")
        print(f"     Oluşturuldu: {fname} ({len(df)} satır)")
        return

    old = pd.read_csv(fname, parse_dates=["Tarih"], dayfirst=True)
    combined = pd.concat([old, df], ignore_index=True)
    combined["Tarih"] = pd.to_datetime(combined["Tarih"], dayfirst=True,
errors="coerce")
    combined = (
        combined.dropna(subset=["Tarih"])
        .sort_values("Tarih")
```

```
        .drop_duplicates(subset=["Tarih"], keep="last")
        .reset_index(drop=True)
    )
    combined.to_csv(fname, index=False, encoding="utf-8")
    print(f"     Güncellendi: {fname} (toplam {len(combined)} satır)")

# ---------- ANA PROGRAM ----------

def fetch_all_series():
    mode = "GÜNCELLEME MODU" if UPDATE_MODE else "FULL MOD"
    print(f" {mode} başlatılıyor...")

    main_cats = safe_get_main_categories()
    if main_cats is None or main_cats.empty:
        print(" Ana kategori alınamadı.")
        return

    for main_cat in main_cats.itertuples():
        cat_id = main_cat.CATEGORY_ID
        cat_name = getattr(main_cat, "TOPIC_TITLE_TR", "Bilinmiyor")
        print(f"\n Ana Kategori: {cat_name} (ID: {cat_id})")

        sub_cats = safe_get_sub_categories(cat_id)
        if sub_cats is None or sub_cats.empty:
            continue

        for sub_cat in sub_cats.itertuples():
            datagroup_code = sub_cat.DATAGROUP_CODE
            sub_name = getattr(sub_cat, "DATAGROUP_NAME", "Bilinmiyor")
            print(f"  ▪ Alt Kategori: {sub_name} (Code:
{datagroup_code})")

            series_df = safe_get_series(datagroup_code)
            if series_df is None or series_df.empty:
                continue

            for s in series_df.itertuples():
                serie_name = getattr(s, "SERIE_NAME", "Bilinmiyor")
                code = getattr(s, "SERIE_CODE", None)
                if not code:
                    continue

                print(f"    • Seri: {serie_name} ({code})")
                df_raw = safe_get_data(code, serie_name=serie_name,
category=sub_name)
                df = normalize_df(df_raw, code)
                append_or_create_csv(
                    series_name=serie_name,
                    df=df,
                    main_category=cat_name,
                    sub_category=sub_name
                )
                time.sleep(SLEEP_BETWEEN_SERIES)
```

```
# ---------- ANA ÇALIŞTIRMA ----------
if __name__ == "__main__":
    start = time.time()
    try:
        fetch_all_series()
    except KeyboardInterrupt:
        print("\n Kullanıcı tarafından durduruldu.")
    except Exception as e:
        print(f" Genel hata: {e}")
    finally:
        elapsed = time.time() - start
        print(f"\n İşlem tamamlandı. Süre: {elapsed:.0f} s")
```