DOCUMENTACION SCRIPTS INGESTA

Desafío de Tripulaciones – Data Science PT 2025

GRUPO 1

Contenido

Introduccion: ... 3

API empleada: ... 3

OCM_Ingest - Ingesta y agregación de precios EV por provincia ... 3

Flujo de datos ... 3

Estructura de carpetas ... 4

Dependencias clave ... 4

Interfaz de línea de comandos (CLI) ... 4

Archivos de salida y esquemas de columnas ... 5

QC por provincia y mes → data/qc/qc_{provincia}_{YYYY-MM}.csv ... 5

Filas válidas del mes (todas las provincias) → data/processed/ocm_rows_{YYYY-MM}.csv ... 5

Agregado mensual → data/processed/ocm_agg_{YYYY-MM}.csv ... 5

Agregado annual → data/processed/ocm_agg_{YYYY}.csv ... 6

Detalle de funciones relevantes ... 6

Consideraciones y limitaciones ... 7

Config_provincias - Generación y caché de *bounding boxes* provinciales (España) ... 7

Flujo de datos ... 7

Salida principal ... 8

Estructura de archivos y caché ... 8

Dependencias ... 8

Funciones del modulo ... 8

Limitaciones y consideraciones ... 9

Ocm_curvas - Curvas mensuales en formato "wide" (AC/DC) por provincia ... 9

Entradas y supuestos ... 9

Salidas ... 10

Flujo de trabajo ... 10

Funciones del script ... 10

Limitaciones y consideraciones ... 11

Build_ev_proxy - Histórico proxy EV por CCAA usando índice PVPC ... 11

Entradas y supuestos ... 11

Salidas ... 12

Flujo de datos ... 12

Funciones del script ... 13

Consideraciones y limitaciones ... 13

Test de normalización de UsageCost (OpenChargeMap) por CCAA ... 13

Resumen de funcionamiento ... 14

Entradas y dependencias ... 14

Logica del parser "usagecost_to_eur_per_kwh(text)" ... 14

Introduccion:

El siguiente documento recoge la información sobre los script empleado para la generación del archivo CSV “ocm_agg_2025” el cual corresponde a la curva de precios histórica mensual de Kw/h para corriente continua y corriente alterna, para cada una de las provincias.

Este CSV, se empleará más adelante en conjunto con un script de generación de archivo JSON con información correspondiente a tickets de facturación de recarga de coches eléctricos, en distintos puntos de carga.

API empleada:

OpenChargeMap (OCM), concretamente el endpoint “GET https://api.openchargemap.io/v3/poi”, que devuelve un array de puntos de recarga (POIs) con su “AddressInfo” (lat/lon, dirección, operador) y una lista de “Connections” (cada conector con potencia PowerKW, nivel, tipo de corriente, etc.). La autenticación se hace con API key, que puedes pasar tanto en cabecera (“X-API-Key”) como en query (“key”). Para acotar resultados empleas “countrycode=E”S y un bounding box en formato WGS84 "(north,west),(south,east)"; además ajustas “maxresults”, y sueles activar “compact=true” y “verbose=false” para respuestas ligeras. Para enriquecer/interpretar “IDs” cuando trabajas en modo compacto, está el “endpoint GET /v3/referencedata”, que expone catálogos (“Levels”, “ConnectionTypes”, “Operators”, etc.).

OCM_Ingest - Ingesta y agregación de precios EV por provincia

Este script descarga, normaliza y agrega precios de carga de vehículos eléctricos en España desde la API de **OpenChargeMap (OCM)**, a nivel de **provincia** y **mes**.
Para cada provincia (definida por un *bounding box*), obtiene los POIs y sus *connections*, intenta **parsear UsageCost a €/kWh**, clasifica el conector en **AC/DC** por potencia, genera ficheros **raw**, **QC** (calidad de parseo) y **procesados** (filas y agregados mensuales), y finalmente puede **combinar** todos los meses de un año en un único CSV anual.

Flujo de datos

1. **Entrada de parámetros (CLI):** mes o año, provincias a incluir, “—overwrite”, “–maxresults”, “--free0”.
2. Cargar API key (OCM_API_KEY desde .env).
3. Iterar provincias (según PROVINCIAS_BOUNDING_BOXES):
    - Descarga JSON de OCM (o lo reutiliza si existe y no hay “—overwrite”).
    - **Normaliza**: por cada “Connection” del POI:
        - Extrae” lat”/”lon”, operador, potencia kW.
        - Clasifica **AC** si “PowerKW” ≤ 22, si no **DC**.
        - Intenta convertir “UsageCost” → **€/kWh** (con razones de fallo/éxito).
        - Guarda siempre una fila de **QC** (diagnóstico) y, si hay €/kWh y banda, una fila “válida” para métricas.
4. Agrega por mes: mediana, media y recuento por (“month”, “provincia”, “band”).

Guarda

- data/raw/ocm_{provincia}_{YYYY-MM}.json
- data/qc/qc_{provincia}_{YYYY-MM}.csv
- data/processed/ocm_rows_{YYYY-MM}.csv → (todas las filas válidas del mes)
- data/processed/ocm_agg_{YYYY-MM}.csv → (agregado mensual)

Estructura de carpetas

data/

- raw/ → Respuestas JSON de la API por provincia y mes
- qc/ → Diagnóstico de parseo de precios por provincia y mes
- processed/ → Filas válidas y agregados mensuales/anuales

Dependencias clave

- API: https://api.openchargemap.io/v3/poi
- Auth: variable de entorno OCM_API_KEY en .env (usando python-dotenv).
- Provincias: config_provincias.py debe exponer **"PROVINCIAS_BOUNDING_BOXES":** Dict[str, Dict[str, float]] con north, south, east, west por provincia. → **VER MAS ADELANTE**

Interfaz de línea de comandos (CLI)

python ocm_ingesta.py [--month YYYY-MM | --year YYYY]

[--provincias PROV ...]

[--overwrite]

[--maxresults N]

- "--month YYYY-MM" → procesa solo ese mes.
- "--year YYYY" → procesa todos los meses desde enero hasta el mes actual (UTC) si es el año en curso.
- "—provincias" (alias "--only-provincias", "—only") → limita a una lista concreta de provincias (nombres exactos como en config_provincias.py).
- "—overwrite" → fuerza re-descarga aunque ya exista el JSON raw.
- "—maxresults" (por defecto 10000) → máximo de registros que solicita a OCM por bounding box y petición.
- "--free0" → trata textos con free/gratis como 0.0 €/kWh (si no se pasa, los marca como "free" y no generan fila válida).

Archivos de salida y esquemas de columnas

QC por provincia y mes → data/qc/qc_{provincia}_{YYYY-MM}.csv

Columna	Descripcion
month	Etiqueta “YYYY-MM”.
provincia	Nombre (clave) de la “provincia”.
operator	Operador (según OperatorInfo.Title).
power_kW	Potencia anunciada de la *connection*.
band	“AC” si “power_kW ≤ 22”, si no “DC”.
usagecost_original	Cadena de “UsageCost” tal cual viene del POI.
parsed	Valor parseado en €/kWh (o “None” si no se pudo/convino).
reason	Motivo del resultado (“ok”, “free”,” free->0”, “per_minute”, “per_session”, “no_kwh”, “parse_error”, “out_of_range”, “empty”).

Filas válidas del mes (todas las provincias) → data/processed/ocm_rows_{YYYY-MM}.csv

Contiene solo las filas con precio parseado válido y banda definida.

Columna	Descripcion
month	“YYYY-MM”.
provincia	Provincia
band	“AC / DC” por regla de potencia.
eur_kWh	Precio en €/kWh parseado.
lat,lon	Coordenadas del POI.
operator	Operador
power_kW	Potencia de la *connection*.

Agregado mensual → data/processed/ocm_agg_{YYYY-MM}.csv

Agregado por (“month”, “provincial”, “band”):

Columna	Descripcion
month	“YYYY-MM”.
provincia	Provincia
band	“AC / DC”
median eur_kWh	Mediana de “eur_kWh”.
mean_eur_kWh	Media de “eur_kWh”
n	Nº de connections validas para ese grupo

Agregado annual → data/processed/ocm_agg_{YYYY}.csv
Concatenación ordenada de todos los ocm_agg_{YYYY-MM}.csv existentes.

Detalle de funciones relevantes

“usagecost_to_eur_per_kwh(text, treat_free_as_zero=False) -> (valor, razón)”

- Limpia el texto y busca patrones de “€/kWh”:
  - Principal: ([0-9]+[.,]?[0-9]*)\s*(€|EUR)\s*/?\s*kW?h
  - Secundario: número + €/EUR + … + kWh
- Detecta y excluye tarifas por minuto o por **sesión**.
- Manejo de “free/gratis”:
  - Si –“-free0”, devuelve “0.0” con razón “free->0”.
  - Si no, marca “free” y no genera fila válida.
- Rango aceptado: 0.05 ≤ €/kWh ≤ 2.0 (si no, “out_of_range”).
- Devuelve (float, "ok") o (None, <reason>).

“band_from_power_kw(power_kw, level_id=None)”

- “AC” si “power_kw ≤ 22”; si no, ”DC.” *(El level_id se ignora; podría usarse en mejoras.)*

“fetch_bbox(bbox, api_key, maxresults=10000, sleep_s=1.0) -> List[Dict]”

- Llama a OCM con:
  - “countrycode=ES, boundingbox=(north,west),(south,east), maxresults, compact=true, verbose=false”.
  - Envía **API key** tanto por query (“key”) como por cabecera (“X-API-Key").
- Reintenta una vez si hay 429/503 y **duerme** sleep_s tras cada petición.”

“normalize_rows(pois_json, prov_name, month_label, treat_free_as_zero=False)

- Recorre POIs y sus *connections* y:
  - Calcula “band”, parsea “eur_kWh”, añade fila QC.
  - Si “parsed” y “band” no son “None”, añade fila válida (para métricas).
- Exporta QC por provincia/mes y devuelve la **lista de filas válidas**.

“aggregate_month(df) -> DataFrame”

- Agrega por (“month”, “provincial”, “band”) → “median_eur_kWh”,” mean_eur_kWh”, “n”.

“process_one_month(...) -> (df_rows, df_agg)”

- Gestiona descarga/reuso del raw por provincia.
- Normaliza y agrega.
- Guarda “rows” y “agg” del mes.

“combine_year(year) -> DataFrame|None”

- Une todos los ocm_agg_{year}-*.csv existentes → “ocm_agg_{year}.csv.”

Consideraciones y limitaciones

- Paginación OCM: se confía en un “maxresults” grande; si OCM limita resultados por bbox, podrían quedar POIs fuera.
- Clasificación AC/DC: basada solo en potencia (“≤22 kW → AC”).
- Precios no en €/kWh: tarifas por minuto/sesión se excluyen de métricas
- Múltiples conexiones por POI: cada *connection* genera fila. n refleja nº de conectores con precio válido, no nº de estaciones.
- Fechas: el etiquetado “month” es lógico,no la fecha real de la tarifa. OCM no garantiza que “UsageCost” sea vigente para ese mes simplemente es una estimacion.
- Dependencia geográfica: requiere “PROVINCIAS_BOUNDING_BOXES” correctos; si cubren zonas fuera de Esapaña lo solapan, puede haber ruido.
- Codificación: se usa UTF-8 en todos los CSV/JSON.

Config_provincias - Generación y caché de *bounding boxes* provinciales (España)

Este módulo obtiene los polígonos de provincias españolas desde OpenDataSoft (dataset “georef-spain-provincia”), calcula sus cajas envolventes (*bounding boxes*, EPSG:4326), normaliza nombres de provincias y cachea el resultado en disco para usos posteriores (p. ej., consultas a APIs por provincia mediante “boundingbox”).

Flujo de datos

1. Comprobar caché en “data/processed/provincias_bbox.json”.
2. Si existe, cargar y exponer “PROVINCIAS_BOUNDING_BOXES”.
3. Si no existe:
   - Descargar GeoJSON de OpenDataSoft.
   - Calcular bounding boxes por provincia.
   - Normalizar nombres (acentos/variantes oficiales/cooficiales).
   - Guardar en caché y exponer “PROVINCIAS_BOUNDING_BOXES”.

Salida principal

- PROVINCIAS_BOUNDING_BOXES: Dict[str, Dict[str, float]] Diccionario {provincia: {north, south, west, east}} con 6 decimales (WGS84).

Estructura de archivos y caché

- Caché JSON: “data/processed/provincias_bbox.json” → Se crea automaticamente si no existe
- Directorios garantizados por “_ensure_dirs()”.

Dependencias

- Red**:** descarga desde https://public.opendatasoft.com/api/explore/v2.1/catalog/datasets/georef-spain-provincia/exports/geojson?lang=es&timezone=UTC
- Paquetes Python:
    - “requests” (HTTP)
    - “shapely” (cálculo de *bounds* de geometrías GeoJSON)

“shapely” se importa **solo dentro** de _build_from_opendatasoft() para evitar coste si ya hay caché.

Funciones del modulo

“_ensure_dirs()”

Crea “data/processed”/ (y padres) si no existen.

“_load_cached() -> dict | None”

Devuelve el contenido JSON de caché si existe; si no, “None”.

“_save_cache(obj) -> None”

Guarda el diccionario de *bboxes* en” _CACHE” (UTF-8, con indentado).

“_build_from_opendatasoft() -> dict”

1. Descarga el GeoJSON.
2. Recorre “features” y obtiene el nombre de provincia desde la primera propiedad disponible en este orden:
    - “prov_name” → “name” → “provincia” → “label_es” → “label”
3. Construye la geometría con “shapely.geometry.shape(geom)” (EPSG:4326).

4. Calcula “minx”, “miny”, “maxx”, “maxy = g.bounds” y devuelve:
    - “north=maxy”, “south=miny”, “west=minx”, “east=maxx” (redondeo a 6 decimales).
5. Normaliza nombres mediante “renames” y reglas específicas para Ceuta y Melilla (manejo de variantes en los metadatos del dataset).

También contempla equivalencias/cooficiales como Araba/Álava → Álava, València → Valencia, Castelló → Castellón, Illes Balears, A Coruña, Ourense….

“refresh_cache() -> dict”

Fuerza reconstrucción desde OpenDataSoft, guarda y devuelve el resultado.

Limitaciones y consideraciones

- Un *bounding box* es una envolvente rectangular mínima; puede incluir mar en provincias costeras y espacios entre islas (p. ej., Illes Balears).
- Nombres oficiales/cooficiales: aunque se normalizan algunos, pueden existir otras variantes
- Dependencia del dataset externo: cambios de esquema/URL podrían romper “_build_from_opendatasoft().”

Ocm_curvas - Curvas mensuales en formato “wide” (AC/DC) por provincia

Este script toma todos los agregados mensuales de un año (salidas ocm_agg_YYYY-MM.csv generadas por la ingesta OCM) y producr dos tablas “wide” con una fila por (mes, territorio) y dos columnas de valores (AC y DC), una para mediana y otra para media del precio en €/kWh.

Entradas y supuestos

- Directorio de entrada: data/processed/
- Ficheros esperados (uno por mes):
    - “ocm_agg_{YYYY}-{MM}.csv,” p. ej.:
    - “ocm_agg_2025-01.csv”, “ocm_agg_2025-02.csv”, …
- Esquema mínimo de columnas en esos CSV:
    - “month” (formato YYYY-MM)
    - “provincia”
    - “band” (AC o DC)
    - “median_eur_kWh” y “mean_eur_kWh”

Salidas

- “data/processed/ocm_curvas_median_{YYYY}_wide.csv” → Mediana mensual en formato ancho: columnas month, <nivel>, AC, DC.
- “data/processed/ocm_curvas_mean_{YYYY}_wide.csv” → Media mensual en formato ancho: columnas month, <nivel>, AC, DC.

Flujo de trabajo

1. Cargar todos los agregados mensuales del año (“load_monthlies_for_year”):
    - Busca “data/processed/ocm_agg_{year}-*.csv”.
    - Concatena en un único DataFrame.
    - Determina el nivel territorial: provincia → si no existe, ccaa.
    - Convierte month a datetime y ordena por month, <nivel>, band.
2. Generar dos pivotes wide (“make_wide”):
    - Para “median_eur_kWh” y para “mean_eur_kWh”:
        - “pivot_table(index=["month", <nivel>], columns="band", values=value_col)”
        - Rellena columnas faltantes (AC o DC) con NA si una banda no aparece en algún territorio/mes.
        - Formatea “month” otra vez como texto “YYYY-MM”.
3. Guardar CSVs en “data/processed/”.

Funciones del script

“load_monthlies_for_year(year: int) -> pd.DataFrame”

- Carga y concatena todos los “ocm_agg_{year}-*.csv”.
- Verifica que haya algún archivo, si no aborta.
- Normaliza “month” a “datetime” y ordena.

“make_wide(df: pd.DataFrame, value_col: str) -> pd.DataFrame”

- “value_col “ ∈ { "median_eur_kWh", "mean_eur_kWh" }.
- Genera la tabla con columnas AC y DC.
- Si una banda no existe para algún grupo, crea la columna y la rellena con NA.
- Devuelve columnas: “month (YYYY-MM)”, “<nivel>”, “AC”, “DC.”

“main()”

- CLI con “--year YYYY” (requerido).
- Crea “data/processed/” si no existe.
- Llama a las funciones anteriores y guarda los dos CSV resultantes.

Limitaciones y consideraciones

- El orden final es “month”, “<nivel>”.” Month” se devuelve como **string YYYY-MM** para facilitar su uso.
- Si en un mes/territorio no hay una de las bandas, la columna correspondiente queda como NA.
- El script no recalcula estadísticas: se limita a reformatear los agregados mensuales ya computados por la ingesta OCM.

Build_ev_proxy - Histórico proxy EV por CCAA usando índice PVPC

Reconstruir un histórico mensual proxy de precios EV (€/kWh) por CCAA y banda (AC/DC) a lo largo de un año, escalando los valores de un mes base real mediante un índice PVPC.

La idea: conservar el nivel (baseline) por CCAA/banda del mes base y aplicar a cada mes la variación relativa del PVPC → Pese a que esta calculado para CCAA nos sirve a nivel de provincia dado que la variación relativa proviene del PVPC, por lo tanto tiene la misma forma y es posible aplicar para obtener una idea aproximada de la tendencia de precios

Entradas y supuestos

- Repositorio y rutas
  - El script localiza la raíz del repo (“REPO_ROOT”) buscando “.git”, “ocm_ingest.py” o “config.py”, o con “git rev-parse”.
  - Los datos se escriben en “REPO_ROOT/data/processed/” (se crea si no existe).
- Mes base (“--base-month, formato YYYY-MM”)
  - Debe existir data/processed/ocm_agg_{base-month}.csv (agregado mensual real) con columnas:
    - “month” (YYYY-MM)
    - “ccaa”
    - “band” (“AC”/”DC”)
    - “median_eur_kWh” y/o “mean_eur_kWh”

- PVPC:
    - Si “--pvpc-file” no se indica y “--year == 2025”, se usa el diccionario embebido “PVPC_2025_DEFAULT” (enero–septiembre 2025).
    - Para años ≠ 2025, es obligatorio “--pvpc-file” (CSV con columnas “month”,”pvpc_eur_kWh”).
    - Métrica: “—metric” define si el baseline usa “median_eur_kWh” o “mean_eur_kWh”.

Este método no recalcula precios reales; genera un proxy escalando el mes base con el PVPC.

Salidas

- **Wide (siempre)**: “data/processed/ocm_proxy_{YYYY}_wide.csv”
    - Columnas: “month (YYYY-MM)”, “ccaa”, “AC”, “DC”.
- **Long (opcional --long)**: “data/processed/ocm_proxy_{YYYY}_long.csv”
    - Columnas: “month”, “ccaa”, “band”, “eur_kWh”.

Flujo de datos

- Encontrar “REPO_ROOT” (“find_repo_root”) y preparar “DATA_DIR”.
- Cargar PVPC (“load_pvpc”):
    - Si hay “--pvpc-file”, lee CSV y valida columnas (“month”, “pvpc_eur_kWh”).
    - Si no, y “year==2025”, usa “PVPC_2025_DEFAULT”.
    - Convierte “month” a datetime, ordena y filtra por “year”.
- Construit indicie PVPC:
    - Comprueba que “base-month” este en PVPC
    - Calcula “pvpc_index = pvpc_eur_kWH / pvp_eur_kWh(base-month)”
- Leer baseline del mes base
    - Abre “data/proccesed/ocm_agg_{base-month}”
    - Selecciona coluna de valor según “—metric (“median_eur_kWh” o “mean_eur_kWh”)
    - Validsa que existan “ccaa”, “band” y la columna de valor.
    - Pivota a wide por CCAA → columnas “AC” y “DC”
- Generar histórico proxy
    - Producto caretesiano baseline x meses PVPC
    - Para cada banda: “valor = baseline_banda * pvpc_index”
- Guardar resultados
    - Wide: “month” formateafo “YYYY-MM”, columnas “month”,”ccaa”,”AC”,”DC”
    - Long (si “—long”): apila AC/DC en “band”, con “eur_kWh”.

Funciones del script

“find_repo_root(start: Path | None = None) -> Path”

Detecta la raíz del repositorio buscando marcadores o consultando a Git. Devuelve un Path.

“load_pvpc(year: int, pvpc_file: str | None) -> pd.DataFrame”

Carga PVPC mensual:

- CSV externo (month, pvpc_eur_kWh) o diccionario embebido (solo 2025).
- Devuelve columnas month (datetime), pvpc_eur_kWh.

“main()”

Parsea flags, carga PVPC, construye índice, extrae baseline del agregado mensual base, genera el histórico proxy (wide y opcionalmente long) y escribe los CSVs.

Consideraciones y limitaciones

Curvas con misma forma: al aplicar un único índice PVPC a todas las CCAA, las curvas mensuales comparten la misma variación relativa; difieren en el nivel según el baseline.

Cobertura PVPC: el diccionario por defecto cubre ene–sep 2025. Para otros meses/años, aporta --pvpc-file.

Dependencia del baseline: si una CCAA carece de AC o DC en el mes base, esa banda quedará NA todo el año.

Nivel territorial: el script espera CCAA en el CSV base. Si tus agregados están por provincia, debes convertirlos previamente a CCAA o adaptar el script.

Test de normalización de UsageCost (OpenChargeMap) por CCAA

Probar de forma rápida la **extracción y normalización** de precios de carga (`UsageCost`) en **€/kWh** a partir de POIs de **OpenChargeMap (OCM)**, acotando por **Comunidad Autónoma** (CCAA) mediante un *bounding box*.
El script imprime en consola pares **“Original → Normalizado”** para una muestra de POIs.

Resumen de funcionamiento

1. Lee argumentos CLI y exige o bien --ccaa o bien --provincia (mutuamente excluyentes), más --n (tamaño de muestra).
2. Carga OCM_API_KEY desde .env.
3. Busca el bounding box del territorio:
    - CCAA_BOUNDING_BOXES (para CCAA)
    - PROVINCIAS_BOUNDING_BOXES (para provincia)
4. Consulta OCM con ese bbox y maxresults = n.
5. Para cada POI:
    - Obtiene UsageCost (texto libre).
    - Aplica el parser mínimo a €/kWh (regex).
    - Imprime: Original: '...' -> Normalizado: X.YZ (o None si no se pudo).

Entradas y dependencias

Entradas

- "—ccaa" *(exclusivo con "—provincia")*: nombre **exacto** según config.py.
- "—provincia" *(exclusivo con" –ccaa")*: nombre **exacto** según config_provincias.py.
- "—n" *(opcional, por defecto 10)*: nº de POIs máximos a recuperar.

Dependencias

- "requests" (HTTP)
- "python-dotenv" (cargar ".env")
- "re" (regex)
- "config.py" → "CCAA_BOUNDING_BOXES"
- "config_provincias.py" → "PROVINCIAS_BOUNDING_BOXES"

Logica del parser "usagecost_to_eur_per_kwh(text)"

- Limpieza → Colapsa espacios
- Regex (case insensitive)
- Conversión: coma → punto, "float"
- Rango válido: "0.05 ≤ €/kWh ≤ 2.0"

Flujo de datos

1. Args: grupo mutuamente " --ccaa/--provincia"; valida nombre en el diccionario correspondiente.
2. Autenticación: carga "OCM_API_KEY" (si falta, lanza error claro).
3. Petición a OCM:
    - "countrycode=ES"
    - "boundingbox=(north,west),(south,east)"
    - "maxresults=n"
    - compact=true, verbose=false
    - API key por query y cabecera ("X-API-Key")
4. Procesado: recorre los POIs, aplica parser, imprime pares "Original → Normalizado".
5. (Sugerido): imprime conteo final "parseables/total (porcentaje)" para medir calidad.

Limitaciones

- **Muestreo**: solo los primeros “n” resultados; no evalúa exhaustivamente todo el territorio.
- **Parser minimalista**: no cubre “€/min”, “€/sesión”, ““gratis/free””, varios precios, condiciones por tiempo, etc.
- **Sin AC/DC**: no clasifica por banda ni usa “Connections”; es un test de campo “UsageCost”.
- **BBox**: como caja envolvente, puede incluir áreas sin POIs (p. ej., mar/entre islas).
- **Rate limiting**: no hay reintentos/backoff para “429/503”.