DOCUMENTACION SCRIPT DATABASE VEHÍCULO / CO_2

GRUPO 1

Contenido

Introducción

Este módulo toma tickets de recarga eléctrica (EV) y tickets de combustibles (ICE), calcula emisiones de CO_2e por consumo (kWh/litros) usando factores de emisión por comercializadora/operador o por combustible, y genera agregados por empresa, usuario, vehículo y mes.

Exporta resultados en `JSON` y `JSONL` listos para informes de sostenibilidad corporativa.

Entradas, salidas y factores.

Archivos de entrada

- EV: `data/tickets_ev_sinteticos.json`
- ICE: `data/tickets_sinteticos.json`

Archivos de salida

- `data/company_sustainability.json` y `data/company_sustainability.jsonl`
- `data/company_sustainability_month.json` y `data/company_sustainability_month.jsonl`

Factores (constantes)

- `KGCO2_PER_L_GASOLINA = 2.35`
- `KGCO2_PER_L_DIESEL = 2.69`
- `GRID_KGCO2_PER_KWH = 0.283` (mix electrico generico)
- `LOSS_FACTOR_TD = 1.096` (perdidas T&D aprox 9,6%)

Normalización y mapeos

Normalización de texto

`_norm(s) -> str.upper() sin acentos`

`_printable(s) -> elimina saltos de línea y colapsa espacios`

Mapeo Operador → Comercializadora (CPO o similar)

`CPO_TO_SUPPLIER`: asigna operador/gestor de carga a la **comercializadora** responsable (`normalizado con _norm`).

Comercializadoras COR (PVPC)

`COR_SUPPLIERS`: comercializadoras de referencia; se **fuerza** el factor genérico de red (`GRID_KGCO2_PER_KWH`).

Factores por comercializadora

`SUPPLIER_FACTOR_KG_PER_KWH`: factores específicos ($kgCO_2/kWh$) por comercializadora (si no está, cae a GRID).

Funciones clave:

```
_supplier_factor(supplier)    # factor por comercializadora (o
GRID)

_factor_por_operador(operador)  # factor desde
operador→supplier→factor

_factor_combustible(fuel)     # 2.69 diésel / 2.35 gasolina (por
defecto)
```

Armonización de columnas y claves

`_alias_norm(df)` : Renombra columnas heterogéneas a nombres estándar → Si existe `mes`, lo fuerza a `str`.

```
idEmpresa, idUsuario, idVehiculo, propulsion, mes,
kwh, litros, fuel, supplier, operador
```

`_ensure_idVehiculo(df)`: Asegura columnas mínima y construye `idVehiculo` si falla

```
idVehiculo = f"{idEmpresa}-{idUsuario}-{propulsion}"
```

Cálculo de emisiones

`_compute_emissions_if_missing(df)`

- Crea (si faltan): `kwh, litros, factor_ele_kg_per_kwh, loss_TD, factor_comb_kg_per_l.`
- Ev (propulsion == "EV"):
    - Donde `factor_ele_kg_per_kwh` se obtiene por `supplier` o por `operador` (CPO→Supplier).

```
kgCO2e_ev = kwh * LOSS_FACTOR_TD * factor_ele_kg_per_kwh
```

- Ice (propulsion == "ICE"): (factor por `fuel` si está; si no, gasolina por defecto para ICE).

```
kgCO2_ice = litros * factor_comb_kg_per_l
```

- Total

```
kgCO2_total = kgCO2e_ev + kgCO2_ice  # con skipna
```

Preparación de datos EV/ICE desde tickets

Los JSON de tickets pueden no traer `kwh/litros` explícitos. Se incluyen extractores robustos desde la lista `lineas`.

EV: `_prep_ev(df)`

- Si falta `kwh`, suma desde `lineas` los ítems con "`producto`" que contenga "`Electric`".
- mes desde `fechaEmision (YYYY-MM)`.
- Fija `propulsion = "EV"`.
- Agrupa por `idEmpresa`, `idUsuario`, `propulsion`, `mes` y suma `kwh`, tomando el primer no nulo de `operador/supplier`.

ICE: `_prep_ice(df)` (versión robusta)

- Si falta `litros`, intenta en orden: `litros → volumen → cantidad` (si unidad ~ litro o producto fuel) → `importe/precio`.
- Infiera `fuel` desde `lineas` si falta (`diesel` / `gasolina`).
- mes desde `fechaEmision`.
- Fija `propulsion = "ICE"`.
- Agrupa por `idEmpresa, idUsuario, propulsion, mes` y suma `litros`.

Extractores auxiliares (ICE):

```
_is_fuel_product(txt)
_is_liter_unit(txt)
_sum_ice_litros_from_lineas(lineas)
_infer_fuel_from_lineas(lineas)
```

Agrupacions de salida

1. Por empresa-usuario-vehículo

```
build_company_user_vehicle_df(df_in)
```

- Estructura de salida

```
idEmpresa, idUsuario, idVehiculo, propulsion,
total_kwh_ev, total_litros_ice,
ev_kgCO2e_total, ice_kgCO2_total, kgCO2_total
```

---

2. Por empresa-usuario-vehículo-mes

```
build_company_user_vehicle_month_df(df_in)
```

- Estructura de salida:

```
idEmpresa, idUsuario, idVehiculo, propulsion, mes,
ev_kwh_mes, ice_litros_mes, ev_kgCO2e_mes, ice_kgCO2_mes,
kgCO2_mes_total
```

Ambas funciones:

- Normalizan alias (`_alias_norm`)
- Aseguran idVehiculo
- Calculan emisiones (`_compute_emissions_if_missing`)
- Agrupan y ordenan columnas

Exportación

```
export_company_df(company_df, out_dir="data",
base_name="company_sustainability")
```

- Valida no vacío.
- Convierte `NaN` → `null` para JSON.
- Escribe:
  - `data/company_sustainability.json`
  - `data/company_sustainability.jsonl` (una línea por registro)

Flujo de generacion

1. Crea `data/` si no existe
2. Lee EV/ICE (`_read_json`).Construir índice por provincia → Tramos candidatos
3. Prepara `ev_agg = _prep_ev(...), ice_agg = _prep_ice(...).`
4. Concatena en `df_in` (rellena `kwh/litros` faltantes con `NaN`).
5. Calcula `company_df = build_company_user_vehicle_df(df_in).`
6. Exporta `company_sustainability`.
7. Muestra archivos exportados y retorna `company_df`.