Tools folder code:A screenshot of a computer

Description automatically generated

# tools/make\_figs\_overlay.py

import os

import re

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from pathlib import Path

# Project helpers

from tools.sim\_support import build\_sim\_config, run\_daily

ROOT = Path(".")

TABLES = ROOT / "tables"

FIGS = ROOT / "figs"

SUMMARY\_CSV = TABLES / "daily\_energy\_all.csv" # already built

MANIFEST\_CSV = ROOT / "selection" / "exemplars\_manifest.csv" # optional for captions

def \_theta\_star\_from\_summary(site, date):

df = pd.read\_csv(SUMMARY\_CSV)

row = df[(df["site"].str.upper()==site.upper()) & (df["date"]==date)]

if row.empty:

raise RuntimeError(f"No row in {SUMMARY\_CSV} for {site} {date}")

r = row.iloc[0]

return float(r["offset\_opt\_deg"]), float(r["E\_perp\_kWh"]), float(r["E\_opt\_kWh"]), float(r["delta\_kWh"]), float(r["delta\_pct"]), str(r.get("bin","")), str(r.get("season",""))

def sweep\_plot\_with\_annotation(site, date):

"""

Read tables/Eday\_sweep\_<SITE>\_<DATE>.csv, convert Wh->kWh, annotate Œ∏‚òÖ + Œî%.

Save figs/Eday\_sweep\_<SITE>\_<DATE>\_annot.pdf

"""

sweep\_path = TABLES / f"Eday\_sweep\_{site}\_{date}.csv"

if not sweep\_path.exists():

raise FileNotFoundError(sweep\_path)

df = pd.read\_csv(sweep\_path)

if not {"offset\_deg","e\_day\_Wh"} <= set(df.columns):

raise RuntimeError(f"{sweep\_path} missing required columns.")

df = df.copy()

df["e\_day\_kWh"] = df["e\_day\_Wh"] / 1000.0

# Get Œ∏‚òÖ and Œî from summary

theta\_star, E0\_kWh, Eopt\_kWh, dE\_kWh, d\_pct, binlab, season = \_theta\_star\_from\_summary(site, date)

# Plot

fig, ax = plt.subplots(figsize=(6.4, 4.2))

ax.plot(df["offset\_deg"], df["e\_day\_kWh"], lw=1.8)

# mark Œ∏‚òÖ on curve

if theta\_star in df["offset\_deg"].values:

ystar = float(df.loc[df["offset\_deg"]==theta\_star, "e\_day\_kWh"].iloc[0])

else:

# fallback to numeric peak

idx = df["e\_day\_kWh"].idxmax()

theta\_star = float(df.loc[idx,"offset\_deg"])

ystar = float(df.loc[idx,"e\_day\_kWh"])

ax.scatter([theta\_star],[ystar], s=45, zorder=5)

# Labels / title

ax.set\_xlabel("Constant offset $\\delta$ [deg]")

ax.set\_ylabel("Daily energy [kWh]")

title\_bin = (binlab.split(":")[0] if isinstance(binlab,str) and ":" in binlab else binlab).strip()

ax.set\_title(f"{site} {date} ({title\_bin}, {season})")

# Annotation

ax.annotate(fr"$\theta^\star={theta\_star:.0f}^\circ$, $\Delta={d\_pct:.1f}\%$",

xy=(theta\_star, ystar),

xytext=(theta\_star+1, ystar\*1.01),

arrowprops=dict(arrowstyle="->", lw=0.8),

fontsize=10)

ax.grid(True, ls=":", lw=0.6)

fig.tight\_layout()

out = FIGS / f"Eday\_sweep\_{site}\_{date}\_annot.pdf"

fig.savefig(out, bbox\_inches="tight")

plt.close(fig)

print("Wrote", out)

def pac\_overlay(site, date, base\_dir="."):

"""

Recompute AC power for Œ¥=0¬∞ and Œ¥=Œ∏‚òÖ using your run\_daily(),

overlay both traces with legend, save to figs/Pac\_<SITE>\_<DATE>\_overlay.pdf

"""

theta\_star, E0\_kWh, Eopt\_kWh, dE\_kWh, d\_pct, binlab, season = \_theta\_star\_from\_summary(site, date)

sim = build\_sim\_config(site, base\_dir=base\_dir)

# Resolve weather filename by site

wmap = {"NC":"data\_raw/NC\_2024\_POWER\_qc.csv", "WC":"data\_raw/WC\_2024\_POWER\_qc.csv"}

wpath = wmap.get(site.upper())

if wpath is None or not Path(wpath).exists():

# try a generic guess

guess = list(Path("data\_raw").glob(f"{site}\_\*qc\*.csv"))

if not guess:

raise FileNotFoundError(f"Weather CSV not found for {site}.")

wpath = str(guess[0])

# Run both offsets

out0 = run\_daily(sim, wpath, date\_local=date, offset\_deg=0.0)

outS = run\_daily(sim, wpath, date\_local=date, offset\_deg=theta\_star)

# Plot

fig, ax = plt.subplots(figsize=(6.4, 4.2))

ax.plot(out0["time"], out0["pac"], label=r"$\delta=0^\circ$ (baseline)", lw=1.6)

ax.plot(outS["time"], outS["pac"], label=fr"$\delta=\theta^\star={theta\_star:.0f}^\circ$", lw=1.6)

ax.set\_ylabel("AC power [W]")

ax.set\_xlabel("Local time")

title\_bin = (binlab.split(":")[0] if isinstance(binlab,str) and ":" in binlab else binlab).strip()

ax.set\_title(f"{site} {date} ({title\_bin}, {season})")

ax.grid(True, ls=":", lw=0.6)

ax.legend(loc="upper right")

fig.autofmt\_xdate()

fig.tight\_layout()

out = FIGS / f"Pac\_{site}\_{date}\_overlay.pdf"

fig.savefig(out, bbox\_inches="tight")

plt.close(fig)

print("Wrote", out)

def make\_for\_examples(pairs):

for site, date in pairs:

sweep\_plot\_with\_annotation(site, date)

pac\_overlay(site, date)

if \_\_name\_\_ == "\_\_main\_\_":

# Default demo set ‚Äî edit to taste

examples = [

("WC","2024-03-02"), # high gain

("NC","2024-08-29"), # high gain

("NC","2024-03-21"), # low gain

("WC","2024-10-23"), # low gain

]

make\_for\_examples(examples)

# NASA POWER hourly fetcher (hourly-safe variables only).

# Usage (run as a single line):

# python tools/power\_fetch.py --lat -28.5 --lon 21.0 --start 2024-01-01 --end 2024-12-31 --out data\_raw/NC\_2024\_POWER.csv

#

# Notes:

# - We request only vars the hourly endpoint serves reliably to avoid 422.

# - We'll compute any extra diagnostics downstream.

import argparse, csv, datetime as dt, requests

# Hourly-safe list (drop CLRSKY\_\*, DNR, DIF, ALLSKY\_SFC\_LW\_DWN)

VARS = ["ALLSKY\_SFC\_SW\_DWN", "T2M", "WS10M", "RH2M", "PS", "ALLSKY\_SRF\_ALB"]

def fetch\_hourly(lat, lon, start, end):

base = "https://power.larc.nasa.gov/api/temporal/hourly/point"

params = {

"parameters": ",".join(VARS),

"community": "RE",

"longitude": lon,

"latitude": lat,

"start": start.strftime("%Y%m%d"),

"end": end.strftime("%Y%m%d"),

"format": "JSON",

}

r = requests.get(base, params=params, timeout=60)

r.raise\_for\_status()

return r.json()

def flatten(js):

props = js.get("properties", {})

param = props.get("parameter", {})

# collect all time keys present across vars

timekeys = set()

for v in param.values():

timekeys |= set(v.keys())

rows = []

for tk in sorted(timekeys):

key = tk.replace(":", "")

if len(key) == 10: # 2024010101

ts = dt.datetime.strptime(key, "%Y%m%d%H")

elif len(key) == 11: # 20240101:01 (rare)

ts = dt.datetime.strptime(key, "%Y%m%d:%H")

else:

continue

rows.append({

"time\_utc": ts.strftime("%Y-%m-%d %H:00"),

"ghi\_wm2": param.get("ALLSKY\_SFC\_SW\_DWN", {}).get(tk),

"temp\_air\_c": param.get("T2M", {}).get(tk),

"wind\_speed\_ms": param.get("WS10M", {}).get(tk),

"rel\_humidity\_pct": param.get("RH2M", {}).get(tk),

"pressure\_pa": (param.get("PS", {}).get(tk) \* 100) if param.get("PS", {}).get(tk) is not None else None,

"albedo": param.get("ALLSKY\_SRF\_ALB", {}).get(tk),

})

return rows

def main():

ap = argparse.ArgumentParser()

ap.add\_argument("--lat", type=float, required=True)

ap.add\_argument("--lon", type=float, required=True)

ap.add\_argument("--start", type=str, required=True) # YYYY-MM-DD

ap.add\_argument("--end", type=str, required=True) # YYYY-MM-DD

ap.add\_argument("--out", type=str, required=True)

args = ap.parse\_args()

start = dt.datetime.strptime(args.start, "%Y-%m-%d").date()

end = dt.datetime.strptime(args.end, "%Y-%m-%d").date()

js = fetch\_hourly(args.lat, args.lon, start, end)

rows = flatten(js)

with open(args.out, "w", newline="") as f:

writer = csv.DictWriter(f, fieldnames=[

"time\_utc","ghi\_wm2","temp\_air\_c","wind\_speed\_ms","rel\_humidity\_pct","pressure\_pa","albedo"

])

writer.writeheader()

for r in rows:

writer.writerow(r)

print(f"Wrote: {args.out} (rows={len(rows)})")

if \_\_name\_\_ == "\_\_main\_\_":

main()

#!/usr/bin/env python3

import re, sys

from pathlib import Path

import pandas as pd

import numpy as np

BASE = Path(".")

SRC\_DIR = BASE / "data\_raw" / "validation\_system\_1430"

OUT = SRC\_DIR / "scada\_pvdaq1430\_2017\_noNov.csv"#!/usr/bin/env python3

import re, sys

from pathlib import Path

import pandas as pd

import numpy as np

BASE = Path(".")

SRC\_DIR = BASE / "data\_raw" / "validation\_system\_1430"

OUT = SRC\_DIR / "scada\_pvdaq1430\_2017\_noNov.csv"

EXCLUDE\_MONTHS = {98} # exclude November

# Column candidates in PVDAQ exports

TIME\_COLS = ["measured\_on", "time\_local", "Timestamp"]

POWER\_COLS = ["ac\_power\_\_5074", "P\_AC\_kW", "power\_kw", "Pac\_kW"] # AC power

def parse\_date\_from\_name(name: str) -> str:

m = re.search(r"date\_(\d{4})\_(\d{2})\_(\d{2})", name)

if not m: raise ValueError(f"Cannot parse date from {name}")

y, mm, dd = m.groups(); return f"{y}-{mm}-{dd}"

def power\_to\_kW(series: pd.Series) -> pd.Series:

"""Return AC power in kW; infer if values are W or kW by magnitude."""

s = pd.to\_numeric(series, errors="coerce")

q95 = s.quantile(0.95)

if pd.isna(q95): return s

# If the plant is tens of kW, PVDAQ often logs AC power in W.

# Heuristic: if 95th percentile > 2000, it's likely W ‚Üí convert to kW.

return s/1000.0 if q95 > 2000 else s

def integrate\_day\_kWh(df: pd.DataFrame):

pcol = next((c for c in POWER\_COLS if c in df.columns), None)

if pcol is None: raise ValueError("No AC power column found")

tcol = next((c for c in TIME\_COLS if c in df.columns), None)

if tcol is None:

# Fall back to fixed 15-min spacing if no timestamp

p\_kw = power\_to\_kW(df[pcol])

return float((p\_kw.fillna(0) \* 0.25).sum()), pd.NaT, pd.NaT, int(len(p\_kw)), f"sum15:{pcol}"

# Integrate AC power vs actual timestamps (trapezoid)

t = pd.to\_datetime(df[tcol], errors="coerce")

p\_kw = power\_to\_kW(df[pcol])

ok = (~t.isna()) & (~p\_kw.isna())

if ok.sum() < 2:

return 0.0, t.min(), t.max(), int(ok.sum()), f"trapz:{pcol}@na"

tt\_h = (t[ok].astype("int64")/1e9)/3600.0

pp = p\_kw[ok].astype(float).values

e\_kWh = float(np.trapz(y=pp, x=tt\_h))

return max(0.0, e\_kWh), t.min(), t.max(), int(ok.sum()), f"trapz:{pcol}"

def main():

files = sorted((SRC\_DIR).glob("system\_1430\_\_date\_\*.csv"))

if not files:

print(f"No daily files in {SRC\_DIR}", file=sys.stderr); sys.exit(1)

rows = []

for f in files:

date\_local = parse\_date\_from\_name(f.name)

if int(date\_local[5:7]) in EXCLUDE\_MONTHS:

continue

try:

df = pd.read\_csv(f)

except Exception as e:

print(f"[SKIP] {f.name}: read error {e}", file=sys.stderr); continue

try:

e\_kWh, t0, t1, n, method = integrate\_day\_kWh(df)

except Exception as e:

print(f"[SKIP] {f.name}: {e}", file=sys.stderr); continue

rows.append({

"site":"PVDAQ1430",

"date\_local":date\_local,

"start\_local":t0,

"end\_local":t1,

"samples":n,

"energy\_kWh":round(e\_kWh,3),

"energy\_source":method

})

if not rows:

print("No rows extracted.", file=sys.stderr); sys.exit(2)

out = pd.DataFrame(rows).sort\_values("date\_local").reset\_index(drop=True)

out.to\_csv(OUT, index=False)

print(f"Wrote: {OUT} (rows={len(out)})")

print(out.head(10).to\_string(index=False))

if \_\_name\_\_ == "\_\_main\_\_":

main()

# Column candidates in PVDAQ exports

TIME\_COLS = ["measured\_on", "time\_local", "Timestamp"]

POWER\_COLS = ["ac\_power\_\_5074", "P\_AC\_kW", "power\_kw", "Pac\_kW"] # AC power

def parse\_date\_from\_name(name: str) -> str:

m = re.search(r"date\_(\d{4})\_(\d{2})\_(\d{2})", name)

if not m: raise ValueError(f"Cannot parse date from {name}")

y, mm, dd = m.groups(); return f"{y}-{mm}-{dd}"

def power\_to\_kW(series: pd.Series) -> pd.Series:

"""Return AC power in kW; infer if values are W or kW by magnitude."""

s = pd.to\_numeric(series, errors="coerce")

q95 = s.quantile(0.95)

if pd.isna(q95): return s

# If the plant is tens of kW, PVDAQ often logs AC power in W.

# Heuristic: if 95th percentile > 2000, it's likely W ‚Üí convert to kW.

return s/1000.0 if q95 > 2000 else s

def integrate\_day\_kWh(df: pd.DataFrame):

pcol = next((c for c in POWER\_COLS if c in df.columns), None)

if pcol is None: raise ValueError("No AC power column found")

tcol = next((c for c in TIME\_COLS if c in df.columns), None)

if tcol is None:

# Fall back to fixed 15-min spacing if no timestamp

p\_kw = power\_to\_kW(df[pcol])

return float((p\_kw.fillna(0) \* 0.25).sum()), pd.NaT, pd.NaT, int(len(p\_kw)), f"sum15:{pcol}"

# Integrate AC power vs actual timestamps (trapezoid)

t = pd.to\_datetime(df[tcol], errors="coerce")

p\_kw = power\_to\_kW(df[pcol])

ok = (~t.isna()) & (~p\_kw.isna())

if ok.sum() < 2:

return 0.0, t.min(), t.max(), int(ok.sum()), f"trapz:{pcol}@na"

tt\_h = (t[ok].astype("int64")/1e9)/3600.0

pp = p\_kw[ok].astype(float).values

e\_kWh = float(np.trapz(y=pp, x=tt\_h))

return max(0.0, e\_kWh), t.min(), t.max(), int(ok.sum()), f"trapz:{pcol}"

def main():

files = sorted((SRC\_DIR).glob("system\_1430\_\_date\_\*.csv"))

if not files:

print(f"No daily files in {SRC\_DIR}", file=sys.stderr); sys.exit(1)

rows = []

for f in files:

date\_local = parse\_date\_from\_name(f.name)

if int(date\_local[5:7]) in EXCLUDE\_MONTHS:

continue

try:

df = pd.read\_csv(f)

except Exception as e:

print(f"[SKIP] {f.name}: read error {e}", file=sys.stderr); continue

try:

e\_kWh, t0, t1, n, method = integrate\_day\_kWh(df)

except Exception as e:

print(f"[SKIP] {f.name}: {e}", file=sys.stderr); continue

rows.append({

"site":"PVDAQ1430",

"date\_local":date\_local,

"start\_local":t0,

"end\_local":t1,

"samples":n,

"energy\_kWh":round(e\_kWh,3),

"energy\_source":method

})

if not rows:

print("No rows extracted.", file=sys.stderr); sys.exit(2)

out = pd.DataFrame(rows).sort\_values("date\_local").reset\_index(drop=True)

out.to\_csv(OUT, index=False)

print(f"Wrote: {OUT} (rows={len(out)})")

print(out.head(10).to\_string(index=False))

if \_\_name\_\_ == "\_\_main\_\_":

main()

from \_\_future\_\_ import annotations

import argparse, pandas as pd, numpy as np

from pathlib import Path

# Columns we might see

TIME\_CANDS = ["timestamp","time","datetime","localtime","utc\_time","date\_time","DateTime","Date"]

PWR\_CANDS = ["ac\_power","ac\_power\_w","ac\_kw","power\_ac\_w","p\_ac","kw\_ac","power\_kw","ac power","kw"]

# Already-daily energy variants

ENERGY\_DAILY\_CANDS = [

"ac\_energy\_daily\_sum", "daily\_ac\_energy", "energy\_kwh", "energy\_mwh",

"AC Energy (kWh)", "Daily Yield (kWh)", "daily\_energy\_kwh"

]

def pick(colnames, candidates):

low = {c.lower(): c for c in colnames}

for k in candidates:

if k.lower() in low: return low[k.lower()]

return None

def to\_daily\_from\_power(df: pd.DataFrame, tz="Africa/Johannesburg") -> pd.DataFrame | None:

tcol = pick(df.columns, TIME\_CANDS)

pcol = pick(df.columns, PWR\_CANDS)

if not tcol or not pcol:

return None

ts = pd.to\_datetime(df[tcol], errors="coerce", utc=True)

p = pd.to\_numeric(df[pcol], errors="coerce")

good = ts.notna() & p.notna()

if not good.any():

return None

s = pd.Series(p[good].values, index=ts[good]).sort\_index()

# Heuristic units: if median<50 ‚Üí kW, else W

sW = s\*1000.0 if np.nanmedian(s.values) < 50 else s

# Regularize to 15-min then integrate

sW = sW.resample("15min").interpolate(limit\_direction="both")

eWh = sW \* 0.25 # Wh per 15-min

eWh.index = eWh.index.tz\_convert(tz)

daily = (eWh.resample("1D").sum()/1000.0).rename("energy\_kWh").reset\_index()

daily.rename(columns={"index":"date\_local", "time":"date\_local"}, inplace=True)

daily["date\_local"] = pd.to\_datetime(daily["date\_local"]).dt.strftime("%Y-%m-%d")

return daily

def to\_daily\_from\_energy(df: pd.DataFrame) -> pd.DataFrame | None:

# Already daily aggregates: a date column + an energy column (kWh or MWh)

dcol = pick(df.columns, TIME\_CANDS)

ecol = pick(df.columns, ENERGY\_DAILY\_CANDS)

if not dcol or not ecol:

# Try: first col is date, 4th is energy (common PVDAQ daily export)

if len(df.columns) >= 4:

dcol = dcol or df.columns[0]

ecol = ecol or df.columns[3]

else:

return None

out = pd.DataFrame({

"date\_local": pd.to\_datetime(df[dcol], errors="coerce"),

"energy": pd.to\_numeric(df[ecol], errors="coerce")

}).dropna()

if out.empty: return None

# If values look like MWh, convert to kWh

med = out["energy"].median()

if np.isfinite(med) and med < 200: # many MW plants have ~10‚Äì50 MWh/day

# assume MWh ‚Üí convert to kWh

energy\_kWh = out["energy"] \* 1000.0

else:

# assume already kWh

energy\_kWh = out["energy"]

out = pd.DataFrame({

"date\_local": out["date\_local"].dt.strftime("%Y-%m-%d"),

"energy\_kWh": energy\_kWh

})

return out

def main():

ap = argparse.ArgumentParser()

ap.add\_argument("--indir", required=True, help="Folder with PVDAQ CSVs (e.g., pvdaq\_system\_9068)")

ap.add\_argument("--site", required=True, help="Label to write as 'site' in output CSV (e.g., PVDAQ9068)")

ap.add\_argument("--tz", default="America/Denver", help="Local timezone for daily aggregation (default America/Denver)")

args = ap.parse\_args()

indir = Path(args.indir)

outs = []

for p in sorted(indir.rglob("\*.csv")):

try:

df = pd.read\_csv(p)

except Exception:

continue

d = to\_daily\_from\_energy(df)

if d is None:

d = to\_daily\_from\_power(df, tz=args.tz)

if d is not None and not d.empty:

outs.append(d)

if not outs:

raise SystemExit("No usable CSVs found. Consider adjusting column candidates in the script.")

daily = pd.concat(outs, ignore\_index=True)

daily = (daily.groupby("date\_local")["energy\_kWh"].sum()

.reset\_index().sort\_values("date\_local"))

daily.insert(0, "site", args.site)

out = Path("scada\_daily\_template.csv")

daily.to\_csv(out, index=False)

print(f"Wrote {out} with {len(daily)} rows.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

# Weather QC script (compatible with hourly-safe POWER columns).

# Usage (one line):

# python tools/qc\_weather.py --in data\_raw/NC\_2024\_POWER.csv --out qc/NC\_2024\_qc\_report.json --timeshift qc/NC\_2024\_timeshift\_check.txt

import argparse, json, pandas as pd

import numpy as np

IRR\_COLS = ["ghi\_wm2"] # hourly-safe fetcher only has GHI

NUM\_COLS = ["ghi\_wm2","temp\_air\_c","wind\_speed\_ms","rel\_humidity\_pct","pressure\_pa","albedo"]

def qc\_dataframe(df: pd.DataFrame, interpolate\_small\_gaps=True):

report = {"input\_rows": int(len(df))}

# Ensure expected cols exist

for c in NUM\_COLS:

if c not in df.columns:

df[c] = np.nan

# Drop rows where all irradiances are NaN (here: just GHI)

mask\_all\_nan = df[IRR\_COLS].isna().all(axis=1)

report["dropped\_all\_irr\_nan"] = int(mask\_all\_nan.sum())

df = df.loc[~mask\_all\_nan].copy()

# Clamp negative irradiance to 0

for c in IRR\_COLS:

neg = (df[c].fillna(0) < 0).sum()

report[f"clamped\_neg\_{c}"] = int(neg)

df.loc[df[c] < 0, c] = 0.0

# Bounds

def clamp\_bounds(col, lo=None, hi=None):

n = 0

if lo is not None:

n += int((df[col].dropna() < lo).sum())

df.loc[df[col] < lo, col] = lo

if hi is not None:

n += int((df[col].dropna() > hi).sum())

df.loc[df[col] > hi, col] = hi

return n

report["bounded\_temp\_outliers"] = clamp\_bounds("temp\_air\_c", -40, 60)

report["bounded\_wind\_outliers"] = clamp\_bounds("wind\_speed\_ms", 0, None)

report["bounded\_rh\_outliers"] = clamp\_bounds("rel\_humidity\_pct", 0, 100)

# Optional interpolation for single missing hours

if interpolate\_small\_gaps:

before = df.isna().sum().to\_dict()

df = df.sort\_values("time\_utc").reset\_index(drop=True)

cols\_to\_interp = ["ghi\_wm2","temp\_air\_c","wind\_speed\_ms","rel\_humidity\_pct","pressure\_pa","albedo"]

df[cols\_to\_interp] = df[cols\_to\_interp].interpolate(limit=1)

after = df.isna().sum().to\_dict()

report["interpolated\_counts"] = {k:int(before[k])-int(after[k]) for k in before.keys()}

else:

report["interpolated\_counts"] = {}

report["output\_rows"] = int(len(df))

return df, report

def main():

ap = argparse.ArgumentParser()

ap.add\_argument("--in", dest="inp", required=True)

ap.add\_argument("--out", required=True)

ap.add\_argument("--timeshift", required=True)

args = ap.parse\_args()

df = pd.read\_csv(args.inp, parse\_dates=["time\_utc"])

df, rep = qc\_dataframe(df, interpolate\_small\_gaps=True)

# Save QC report

with open(args.out, "w") as f:

json.dump(rep, f, indent=2)

# Clean CSV alongside (same name with \_qc.csv)

out\_csv = args.inp.replace(".csv","\_qc.csv")

df.to\_csv(out\_csv, index=False)

# Timeshift checklist

with open(args.timeshift, "w") as f:

f.write(

"Timeshift sanity checklist (UTC -> Africa/Johannesburg):\n"

"- Confirm solar noon (UTC) +2h aligns with expected SAST solar noon.\n"

"- Plot GHI diurnal curve; check sunrise/sunset transitions.\n"

"- If offset seen, shift timestamps accordingly before analysis.\n"

)

print(f"QC report: {args.out}")

print(f"Clean CSV : {out\_csv}")

print(f"Wrote timeshift checklist: {args.timeshift}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

from \_\_future\_\_ import annotations

import argparse

from pathlib import Path

import pandas as pd

import numpy as np

from tools.sim\_support import build\_sim\_config, run\_daily

ROOT = Path(".")

SEL = ROOT/"selection"

TABLES = ROOT/"tables"

DATA = ROOT/"data\_raw"

TABLES.mkdir(exist\_ok=True, parents=True)

def load\_daylist(site: str) -> pd.DataFrame:

fn = SEL/f"{site}\_2024\_day\_bins.csv"

df = pd.read\_csv(fn)

if "date\_local" not in df.columns and "date" in df.columns:

df = df.rename(columns={"date":"date\_local"})

if "bin" not in df.columns and "bin\_label" in df.columns:

df = df.rename(columns={"bin\_label":"bin"})

if "season" not in df.columns:

m = pd.to\_datetime(df["date\_local"]).dt.month

season = pd.cut(m, bins=[0,2,5,8,11,12],

labels=["Summer (DJF)","Autumn (MAM)","Winter (JJA)","Spring (SON)","Summer (DJF)"],

right=True, include\_lowest=True)

df["season"] = season.values

df["site"]=site

return df[["site","date\_local","bin","season"]].copy()

def site\_weather(site: str) -> str:

return str(DATA/f"{site}\_2024\_POWER\_qc.csv")

def bin\_median\_offsets() -> pd.DataFrame:

summ = pd.read\_csv(TABLES/"daily\_energy\_all.csv")

summ["bin\_code"] = summ["bin"].str.split(":").str[0]

g = summ.groupby(["site","bin\_code"])

med = g[["offset\_opt\_deg","delta\_pct"]].median().reset\_index()

med = med.rename(columns={"bin\_code":"bin","offset\_opt\_deg":"theta\_med\_deg","delta\_pct":"delta\_med\_pct"})

return med

def compute\_day\_energy(site: str, date: str, delta\_deg: float) -> float:

sim = build\_sim\_config(site, base\_dir=".")

out = run\_daily(sim, site\_weather(site), date\_local=date, offset\_deg=float(delta\_deg))

return float(out["pac"].clip(lower=0).sum()/1000.0) # kWh

def main():

ap = argparse.ArgumentParser()

ap.add\_argument("--scada", type=str, default=None, help="Optional SCADA daily CSV with columns: site,date\_local,energy\_kWh")

args = ap.parse\_args()

# If you only want validation (no SA sites set up), we can skip NC/WC gracefully

have\_nc = (SEL/"NC\_2024\_day\_bins.csv").exists()

have\_wc = (SEL/"WC\_2024\_day\_bins.csv").exists()

daylists = []

if have\_nc: daylists.append(load\_daylist("NC"))

if have\_wc: daylists.append(load\_daylist("WC"))

if daylists:

daylists = pd.concat(daylists, ignore\_index=True)

daylists["date\_local"] = pd.to\_datetime(daylists["date\_local"]).dt.strftime("%Y-%m-%d")

# Bin medians for SA sites (if present)

med = bin\_median\_offsets() if (TABLES/"daily\_energy\_all.csv").exists() else pd.DataFrame(columns=["site","bin","theta\_med\_deg","delta\_med\_pct"])

pol\_rows, daily\_rows = [], []

# Run SA (NC/WC) annual policy comparison if we have day lists

for site, g in (daylists.groupby("site") if isinstance(daylists, pd.DataFrame) and not daylists.empty else []):

th\_map = med[med["site"]==site].set\_index("bin")["theta\_med\_deg"].to\_dict()

b4\_theta = th\_map.get("B4", 0.0)

E0 = E\_b4 = E\_bin = 0.0

for \_, r in g.iterrows():

date = r["date\_local"]

b = (r["bin"].split(":")[0] if ":" in r["bin"] else r["bin"]).strip()

e0 = compute\_day\_energy(site, date, 0.0)

eb4 = compute\_day\_energy(site, date, b4\_theta if b=="B4" else 0.0)

ebin = compute\_day\_energy(site, date, th\_map.get(b, 0.0))

E0 += e0; E\_b4 += eb4; E\_bin += ebin

daily\_rows.append({"site":site,"date":date,"bin":b,

"E0\_kWh":e0,"E\_policyB4\_kWh":eb4,"E\_idealBin\_kWh":ebin,

"theta\_B4\_deg":b4\_theta,"theta\_bin\_deg":th\_map.get(b,0.0)})

if E0>0:

pol\_rows.append({"site":site,

"E0\_year\_kWh":E0,

"E\_policyB4\_year\_kWh":E\_b4,

"E\_idealBin\_year\_kWh":E\_bin,

"gain\_policyB4\_pct":100\*(E\_b4-E0)/E0,

"gain\_idealBin\_pct":100\*(E\_bin-E0)/E0})

if daily\_rows:

daily = pd.DataFrame(daily\_rows)

daily.to\_csv(TABLES/"annual\_policy\_daily\_log.csv", index=False)

if pol\_rows:

annual = pd.DataFrame(pol\_rows)

annual.to\_csv(TABLES/"annual\_policy\_by\_site.csv", index=False)

print("\nAnnual energy by site (kWh) and gains vs baseline (%):")

print(annual.to\_string(index=False))

else:

print("\nNo NC/WC annual policy run (day lists not found) ‚Äî proceeding to SCADA validation only.")

# SCADA validation (baseline model vs measured)

if args.scada:

sc = pd.read\_csv(args.scada)

sc["date\_local"]=pd.to\_datetime(sc["date\_local"]).dt.strftime("%Y-%m-%d")

# Validate for any site present in SCADA that also has a row in inputs/sites.csv

sites = sc["site"].unique().tolist()

vals = []

for site in sites:

# we need the site to exist in inputs/sites.csv so build\_sim\_config can run

try:

# model daily AC for all scada dates

dates = sc[sc["site"]==site]["date\_local"].tolist()

E0s = []

for d in dates:

try:

E0s.append({"date\_local":d, "E0\_kWh": compute\_day\_energy(site, d, 0.0)})

except Exception:

# if any missing weather day, skip that date

pass

if not E0s:

continue

mod = pd.DataFrame(E0s)

m = mod.merge(sc[sc["site"]==site][["date\_local","energy\_kWh"]], on="date\_local", how="inner")

if m.empty:

continue

y = m["energy\_kWh"].to\_numpy()

yhat = m["E0\_kWh"].to\_numpy()

mbe = float((yhat - y).mean())

nrmse = float(np.sqrt(np.mean((yhat-y)\*\*2)) / (y.mean() if y.mean()!=0 else 1.0))

vals.append({"site":site,"n\_days":len(m),"MBE\_kWh":mbe,"NRMSE\_frac":nrmse})

except Exception as e:

continue

if vals:

v = pd.DataFrame(vals)

v.to\_csv(TABLES/"validity\_stats.csv", index=False)

print("\nValidation vs SCADA (baseline model ‚Üí measured):")

print(v.to\_string(index=False))

else:

print("\nSCADA validation ran, but no overlapping dates/site config were usable (check inputs/sites.csv and weather files).")

else:

print("\nSCADA file not provided: run with --scada scada\_daily\_template.csv when ready.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

"""

Run Step 2 daily results:

- For each exemplar (from selection/exemplars\_manifest.csv), generate:

\* Tc(t) plot -> figs/Tc\_{site}\_{date}.pdf

\* Pac(t) plot -> figs/Pac\_{site}\_{date}.pdf

\* E\_day(Œ¥) sweep CSV and plot -> tables/Eday\_sweep\_{site}\_{date}.csv, figs/Eday\_sweep\_{site}\_{date}.pdf

Usage:

python tools/run\_step2\_daily.py --site NC --weather data\_raw/NC\_2024\_POWER\_qc.csv

python tools/run\_step2\_daily.py --site WC --weather data\_raw/WC\_2024\_POWER\_qc.csv

"""

import argparse, os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sim\_support import build\_sim\_config, run\_daily, sweep\_offsets

def ensure\_dirs():

os.makedirs("figs", exist\_ok=True)

os.makedirs("tables", exist\_ok=True)

def plot\_series(x, y, ylabel, title, outpath):

fig, ax = plt.subplots(figsize=(10,4))

ax.plot(x, y, lw=1.6)

ax.set\_xlabel("Time (local)")

ax.set\_ylabel(ylabel)

ax.set\_title(title)

fig.autofmt\_xdate()

fig.tight\_layout()

fig.savefig(outpath, bbox\_inches="tight")

plt.close(fig)

def plot\_sweep(df, site, date, outpath):

fig, ax = plt.subplots(figsize=(8,4))

ax.plot(df["offset\_deg"], df["e\_day\_Wh"], marker="o")

ax.set\_xlabel("Offset Œ¥ [deg]")

ax.set\_ylabel("E\_day [Wh]")

ax.set\_title(f"E\_day(Œ¥) ‚Äî {site} {date}")

fig.tight\_layout()

fig.savefig(outpath, bbox\_inches="tight")

plt.close(fig)

def main():

ap = argparse.ArgumentParser()

ap.add\_argument("--site", required=True, help="NC or WC")

ap.add\_argument("--weather", required=True, help="Path to \*\_POWER\_qc.csv for the site")

ap.add\_argument("--manifest", default="selection/exemplars\_manifest.csv")

ap.add\_argument("--min\_deg", type=float, default=-30)

ap.add\_argument("--max\_deg", type=float, default=30)

ap.add\_argument("--step\_deg", type=float, default=2)

args = ap.parse\_args()

ensure\_dirs()

sim = build\_sim\_config(args.site, base\_dir=".")

ex = pd.read\_csv(args.manifest)

ex = ex[ex["site"].str.upper()==args.site.upper()].copy()

for \_, row in ex.iterrows():

date = row["date\_local"]

# Baseline Œ¥=0

out = run\_daily(sim, args.weather, date, offset\_deg=0.0)

plot\_series(out["time"], out["tc"], "Cell temp Tc [¬∞C]", f"Tc(t) ‚Äî {args.site} {date} (Œ¥=0¬∞)", f"figs/Tc\_{args.site}\_{date}.pdf")

plot\_series(out["time"], out["pac"], "P\_AC [W]", f"P\_AC(t) ‚Äî {args.site} {date} (Œ¥=0¬∞)", f"figs/Pac\_{args.site}\_{date}.pdf")

# E\_day sweep

sweep = sweep\_offsets(sim, args.weather, date, args.min\_deg, args.max\_deg, args.step\_deg)

sweep.to\_csv(f"tables/Eday\_sweep\_{args.site}\_{date}.csv", index=False)

plot\_sweep(sweep, args.site, date, f"figs/Eday\_sweep\_{args.site}\_{date}.pdf")

print(f"Done. Figures in figs/, tables in tables/.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

#!/usr/bin/env python3

# Exemplar-day selection with auto unit-detect & per-day q95 clear-sky proxy

# Usage (one line):

# python tools/select\_bins.py --site WC --in data\_raw/WC\_2024\_POWER\_qc.csv --out selection/WC\_2024\_day\_bins.csv --fig figs/day\_bin\_table\_WC.pdf

# python tools/select\_bins.py --site NC --in data\_raw/NC\_2024\_POWER\_qc.csv --out selection/NC\_2024\_day\_bins.csv --fig figs/day\_bin\_table.pdf

import argparse, os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import pytz

def season\_from\_month(m: int) -> str:

if m in (12,1,2): return "Summer (DJF)"

if m in (3,4,5): return "Autumn (MAM)"

if m in (6,7,8): return "Winter (JJA)"

return "Spring (SON)"

def classify\_day(r\_med, ws\_med, ta\_med, wind\_split, hot\_split):

if pd.isna(r\_med): sky = "mixed"

elif r\_med >= 0.8: sky = "clear"

elif r\_med >= 0.4: sky = "mixed"

else: sky = "overcast"

wind = "low" if (pd.notna(ws\_med) and ws\_med < wind\_split) else "high"

therm = "hot" if (pd.notna(ta\_med) and ta\_med >= hot\_split) else "cool"

return sky, wind, therm

def main():

ap = argparse.ArgumentParser()

ap.add\_argument("--site", required=True)

ap.add\_argument("--in", dest="inp", required=True)

ap.add\_argument("--out", required=True)

ap.add\_argument("--fig", required=True)

ap.add\_argument("--wind\_split\_ms", type=float, default=3.0)

ap.add\_argument("--hot\_split\_c", type=float, default=25.0)

# you can override, but script auto-detects units and sets its own daylight threshold

args = ap.parse\_args()

tz = pytz.timezone("Africa/Johannesburg")

if not os.path.exists(args.inp):

raise FileNotFoundError(args.inp)

# --- load & parse timestamps ---

df = pd.read\_csv(args.inp)

if "time\_utc" not in df.columns:

raise RuntimeError("Column 'time\_utc' not found.")

df["time\_utc"] = pd.to\_datetime(df["time\_utc"].astype(str).str.strip(), errors="coerce", utc=True)

df = df.dropna(subset=["time\_utc"]).reset\_index(drop=True)

# ensure numeric columns exist & numeric

for c in ["ghi\_wm2","temp\_air\_c","wind\_speed\_ms","rel\_humidity\_pct","pressure\_pa","albedo"]:

if c not in df.columns: df[c] = np.nan

for c in ["ghi\_wm2","temp\_air\_c","wind\_speed\_ms"]:

df[c] = pd.to\_numeric(df[c], errors="coerce")

# --- AUTO UNIT DETECT / NORMALIZE to W/m¬≤ ---

# Look at high-end irradiance: if the 95th percentile is very small (<=5), assume it's kW/m¬≤ or kWh/m¬≤ per hour

# and multiply by 1000 to get W/m¬≤. (Typical peak sun ‚âà 800‚Äì1100 W/m¬≤)

q95\_raw = np.nanpercentile(df["ghi\_wm2"], 95) if df["ghi\_wm2"].notna().any() else np.nan

scale = 1.0

if np.isfinite(q95\_raw) and q95\_raw <= 5.0:

scale = 1000.0

df["ghi\_wm2"] = df["ghi\_wm2"] \* scale

# time zone & local date

df["time\_local"] = df["time\_utc"].dt.tz\_convert(tz)

df["date\_local"] = df["time\_local"].dt.date

# --- daylight threshold: adaptive ---

# Use overall q90 after scaling; daylight threshold = max(20 W/m¬≤, 10% of overall q90)

if df["ghi\_wm2"].notna().any():

q90 = np.nanpercentile(df["ghi\_wm2"], 90)

daylight\_thr = max(20.0, 0.10 \* q90)

else:

raise RuntimeError("No valid GHI values found.")

df["daylight"] = df["ghi\_wm2"] > daylight\_thr

# --- per-day q95 clear-sky proxy using only daylight rows ---

day = df[df["daylight"]].copy()

if day.empty:

raise RuntimeError("No daylight rows found even after adaptive thresholding. Check the input file.")

q95 = day.groupby("date\_local")["ghi\_wm2"].quantile(0.95).rename("q95")

df = df.merge(q95, on="date\_local", how="left")

df["q95"] = np.clip(df["q95"], 1.0, None)

df["r"] = np.where(df["daylight"], df["ghi\_wm2"] / df["q95"], np.nan)

# --- daily aggregates over daylight only ---

daylight\_only = df[df["daylight"]]

agg = daylight\_only.groupby("date\_local").agg(

r\_median = ("r", "median"),

wind\_median = ("wind\_speed\_ms", "median"),

Ta\_median = ("temp\_air\_c", "median"),

ghi\_day\_Whm2 = ("ghi\_wm2", "sum"),

).reset\_index()

# Convert Wh/m¬≤ to MJ/m¬≤ (hourly W/m¬≤ summed over hours = Wh/m¬≤)

agg["ghi\_day\_MJm2"] = agg["ghi\_day\_Whm2"] \* 0.0036

# season + classification

agg["month"] = pd.to\_datetime(agg["date\_local"]).dt.month

agg["season"] = agg["month"].apply(season\_from\_month)

sky, wind, therm = [], [], []

for \_, row in agg.iterrows():

s, w, t = classify\_day(row["r\_median"], row["wind\_median"], row["Ta\_median"],

args.wind\_split\_ms, args.hot\_split\_c)

sky.append(s); wind.append(w); therm.append(t)

agg["sky"] = sky; agg["wind"] = wind; agg["thermal"] = therm

# bin labels

def bin\_label(s,w,t):

if s=="clear" and w=="low" and t=="hot": return "B1: Clear / Low-wind / Hot"

if s=="clear" and w=="high" and t=="hot": return "B2: Clear / High-wind / Hot"

if s=="mixed" and w=="low" and t=="hot": return "B3: Mixed / Low-wind / Hot"

if s=="mixed" and w=="high" and t=="cool": return "B4: Mixed / High-wind / Cool"

if s=="overcast" and w=="low" and t=="cool": return "B5: Overcast / Low-wind / Cool"

return None

agg["bin"] = [bin\_label(s,w,t) for s,w,t in zip(agg["sky"], agg["wind"], agg["thermal"])]

cand = agg.dropna(subset=["bin"]).copy()

# exemplar per (bin, season): highest daily GHI

cand["rank"] = cand.groupby(["bin","season"])["ghi\_day\_MJm2"].rank(ascending=False, method="first")

selected = cand[cand["rank"]==1].copy().sort\_values(["bin","season"])

# save CSV

os.makedirs(os.path.dirname(args.out), exist\_ok=True)

out\_cols = ["date\_local","bin","season","r\_median","wind\_median","Ta\_median","ghi\_day\_MJm2"]

selected.to\_csv(args.out, index=False, columns=out\_cols)

# render compact table figure

os.makedirs(os.path.dirname(args.fig), exist\_ok=True)

fig, ax = plt.subplots(figsize=(10, 2 + 0.3\*max(1,len(selected))))

ax.axis('off')

header = ["Date","Bin","Season","Median r","Median WS [m/s]","Median Ta [¬∞C]","GHI day [MJ/m¬≤]"]

table\_data = [header]

for \_, row in selected.iterrows():

table\_data.append([

str(row["date\_local"]),

row["bin"],

row["season"],

f'{row["r\_median"]:.2f}' if pd.notna(row["r\_median"]) else "",

f'{row["wind\_median"]:.2f}' if pd.notna(row["wind\_median"]) else "",

f'{row["Ta\_median"]:.1f}' if pd.notna(row["Ta\_median"]) else "",

f'{row["ghi\_day\_MJm2"]:.1f}' if pd.notna(row["ghi\_day\_MJm2"]) else ""

])

table = ax.table(cellText=table\_data, loc='center')

table.auto\_set\_font\_size(False); table.set\_fontsize(10); table.scale(1, 1.4)

fig.tight\_layout(); fig.savefig(args.fig, bbox\_inches="tight"); plt.close(fig)

print(f"[{args.site}] wrote: {args.out}")

print(f"[{args.site}] wrote: {args.fig}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

"""

Simulation helpers for daily PV modelling with pvlib (with graceful fallbacks).

Features:

- Loads project configs (models/module/inverter/tracker)

- Builds a SimConfig with site metadata (lat/lon/tz)

- Loads QC'd weather (tz-aware local index)

- Decomposes GHI -> (DNI, DHI) using ERBS

- Computes POA on single-axis tracker (Hay-Davies) with constant offset Œ¥

- Faiman module temperature

- DC (DeSoto) and AC (Sandia/CEC) power

- Daily energy integration using actual time deltas

- Offset sweep utility for Œ¥ in [min, max] with step

Requirements:

pip install pvlib pandas pyyaml matplotlib numpy

"""

from \_\_future\_\_ import annotations

import pandas as pd

import numpy as np

import yaml

from dataclasses import dataclass

from typing import Dict, Any, Tuple

# Try pvlib import early so we can fail fast with a clear message

try:

import pvlib

from pvlib.temperature import faiman

from pvlib import location, irradiance, pvsystem, tracking

PVLIB\_AVAILABLE = True

except Exception as e:

PVLIB\_AVAILABLE = False

PVLIB\_IMPORT\_ERROR = str(e)

# -----------------------------

# Data container for simulation

# -----------------------------

@dataclass

class SimConfig:

site\_name: str

lat: float

lon: float

tz: str

models: Dict[str, Any]

module: Dict[str, Any]

inverter: Dict[str, Any]

tracker: Dict[str, Any]

# -----------------------------

# Config + site loading

# -----------------------------

def load\_yaml(path: str) -> Dict[str, Any]:

with open(path, "r") as f:

return yaml.safe\_load(f)

def load\_project\_configs(base\_dir: str = ".") -> Dict[str, Any]:

models = load\_yaml(f"{base\_dir}/configs/models.yaml")

module = load\_yaml(f"{base\_dir}/configs/module\_trina415.yaml")

inverter = load\_yaml(f"{base\_dir}/configs/inverter\_sg2500hv.yaml")

tracker = load\_yaml(f"{base\_dir}/configs/tracker\_sat.yaml")

return {"models": models, "module": module, "inverter": inverter, "tracker": tracker}

def load\_site\_row(sites\_csv: str, site: str) -> Dict[str, Any]:

df = pd.read\_csv(sites\_csv)

row = df.loc[df["site"].str.upper() == site.upper()]

if row.empty:

raise RuntimeError(f"Site '{site}' not found in {sites\_csv}")

r = row.iloc[0].to\_dict()

lat = float(r.get("lat") or r.get("latitude"))

lon = float(r.get("lon") or r.get("longitude"))

tz = r.get("tz") or r.get("timezone") or "Africa/Johannesburg"

return {"site": site.upper(), "lat": lat, "lon": lon, "tz": tz}

def build\_sim\_config(site: str, base\_dir=".") -> SimConfig:

if not PVLIB\_AVAILABLE:

raise RuntimeError(f"pvlib is required for simulation but could not be imported: {PVLIB\_IMPORT\_ERROR}")

cfgs = load\_project\_configs(base\_dir)

site\_row = load\_site\_row(f"{base\_dir}/inputs/sites.csv", site)

return SimConfig(

site\_name=site\_row["site"],

lat=site\_row["lat"],

lon=site\_row["lon"],

tz=site\_row["tz"],

models=cfgs["models"],

module=cfgs["module"],

inverter=cfgs["inverter"],

tracker=cfgs["tracker"],

)

def \_ensure\_pvlib():

if not PVLIB\_AVAILABLE:

raise RuntimeError(f"pvlib is required for simulation but could not be imported: {PVLIB\_IMPORT\_ERROR}")

# -----------------------------

# Weather I/O & preprocessing

# -----------------------------

def load\_weather\_qc(path\_csv: str, tz: str) -> pd.DataFrame:

"""

Load QC'd hourly weather and return tz-aware index in local time (hour-begin).

Expected columns (non-fatal if missing; they will be created as NaN):

time\_utc, ghi\_wm2, temp\_air\_c, wind\_speed\_ms, rel\_humidity\_pct, pressure\_pa, albedo

"""

df = pd.read\_csv(path\_csv, parse\_dates=["time\_utc"])

df["time\_local"] = pd.to\_datetime(df["time\_utc"], utc=True).dt.tz\_convert(tz)

df = df.set\_index("time\_local").sort\_index()

for c in ["ghi\_wm2", "temp\_air\_c", "wind\_speed\_ms", "rel\_humidity\_pct", "pressure\_pa", "albedo"]:

if c not in df.columns:

df[c] = np.nan

df[c] = pd.to\_numeric(df[c], errors="coerce")

# Sensible defaults where totally missing

if df["albedo"].isna().all():

df["albedo"] = 0.2

if df["pressure\_pa"].isna().all():

df["pressure\_pa"] = 101325.0

return df

# -----------------------------

# Irradiance decomposition

# -----------------------------

def erbs\_decompose(df\_local: pd.DataFrame, loc: "pvlib.location.Location") -> pd.DataFrame:

"""

Decompose GHI->DNI/DHI using ERBS with solar position.

Adds columns: dni\_wm2, dhi\_wm2; applies night mask (zenith >= 90 deg).

"""

\_ensure\_pvlib()

# Solar position (tz-aware index is required)

solpos = loc.get\_solarposition(df\_local.index)

zen = solpos["apparent\_zenith"].clip(upper=90)

ghi = df\_local["ghi\_wm2"].clip(lower=0).astype(float)

# ERBS decomposition (works well for hourly data)

erbs = irradiance.erbs(ghi, zen, df\_local.index)

dni = erbs["dni"].fillna(0.0).clip(lower=0.0)

dhi = erbs["dhi"].fillna(0.0).clip(lower=0.0)

# Night cleanup

night = zen >= 90

dni = dni.where(~night, 0.0)

dhi = dhi.where(~night, 0.0)

out = df\_local.copy()

out["dni\_wm2"] = dni

out["dhi\_wm2"] = dhi

return out

# -----------------------------

# POA on single-axis tracker

# -----------------------------

def poa\_on\_tracker(df: pd.DataFrame, sim: SimConfig, backtrack: bool = True, offset\_deg: float = 0.0) -> Tuple[pd.DataFrame, pd.DataFrame]:

"""

Compute POA components for a single-axis tracker with a constant tilt offset Œ¥.

Uses Hay‚ÄìDavies transposition and provides dni\_extra (required).

Returns: (poa\_df, tracker\_df)

"""

\_ensure\_pvlib()

# Location / solar position

loc = location.Location(sim.lat, sim.lon, tz=sim.tz)

solpos = loc.get\_solarposition(df.index)

zen = solpos["apparent\_zenith"].clip(upper=90)

# Tracker geometry (from configs)

axis\_tilt = float(sim.tracker.get("axis\_tilt\_deg", 0.0))

axis\_azimuth = float(sim.tracker.get("axis\_azimuth\_deg", 0.0))

max\_angle = float(sim.tracker.get("max\_angle\_deg", 60.0))

gcr = float(sim.tracker.get("gcr", 0.30))

trk = tracking.singleaxis(

apparent\_zenith=zen,

apparent\_azimuth=solpos["azimuth"],

axis\_tilt=axis\_tilt,

axis\_azimuth=axis\_azimuth,

max\_angle=max\_angle,

backtrack=backtrack,

gcr=gcr

)

# Apply constant offset Œ¥ to the surface tilt (clip to [0, max\_angle])

trk["surface\_tilt"] = (trk["surface\_tilt"] + float(offset\_deg)).clip(lower=0, upper=max\_angle)

# Required extraterrestrial DNI for Hay‚ÄìDavies

dni\_extra = irradiance.get\_extra\_radiation(df.index, method="spencer")

# Transposition (Hay‚ÄìDavies)

poa = irradiance.get\_total\_irradiance(

surface\_tilt=trk["surface\_tilt"],

surface\_azimuth=trk["surface\_azimuth"],

dni=df["dni\_wm2"],

ghi=df["ghi\_wm2"],

dhi=df["dhi\_wm2"],

solar\_zenith=zen,

solar\_azimuth=solpos["azimuth"],

dni\_extra=dni\_extra, # <-- this fixes the previous error

albedo=df.get("albedo", 0.2),

model="haydavies"

)

return poa, trk

# -----------------------------

# Temperature & Power chains

# -----------------------------

def faiman\_cell\_temp(poa\_irr: pd.Series, temp\_air: pd.Series, wind: pd.Series, u0=25.0, u1=6.0) -> pd.Series:

"""

Faiman model for module temperature (cell/backsheet proxy).

u0 [W/m^2/K], u1 [W/(m^2\*K)/(m/s)].

"""

return faiman(poa\_global=poa\_irr, temp\_air=temp\_air, wind\_speed=wind, u0=u0, u1=u1)

def dc\_ac\_power(sim: SimConfig, poa: pd.DataFrame, tcell: pd.Series) -> Tuple[pd.DataFrame, pd.Series]:

"""

Compute DC (De Soto single-diode) and inverter AC.

- Uses cec/sapm-like dicts from configs where available.

- Falls back gracefully if some coefficients are missing.

Returns: (dc\_df with p\_dc & p\_mp\_raw, pac series)

"""

\_ensure\_pvlib()

# Module params (prefer CEC params; otherwise accept SAPM)

mod = sim.module.get("cec\_params", {}) or sim.module.get("sapm\_params", {})

gamma\_p = float(sim.module.get("gamma\_p", -0.0034)) # per ¬∞C for Pmp

# Effective irradiance using SAPM formulation if possible; fallback to POA global

try:

ee = pvsystem.sapm\_effective\_irradiance(

poa\_direct=poa["poa\_direct"],

poa\_diffuse=poa["poa\_diffuse"],

airmass\_absolute=None,

airmass\_relative=None,

aoi=poa.get("aoi", None),

module=mod

)

ee = ee.fillna(poa["poa\_global"])

except Exception:

ee = poa["poa\_global"]

# De Soto single-diode parameters

il, io, rs, rsh, nNsVth = pvsystem.calcparams\_desoto(

effective\_irradiance=ee,

temp\_cell=tcell,

alpha\_sc=mod.get("alpha\_sc", 0.0045),

a\_ref=mod.get("a\_ref", 1.3),

I\_L\_ref=mod.get("I\_L\_ref", 9.5),

I\_o\_ref=mod.get("I\_o\_ref", 1e-10),

R\_sh\_ref=mod.get("R\_sh\_ref", 200.0),

R\_s=mod.get("R\_s", 0.5),

EgRef=1.121,

dEgdT=-0.0002677

)

sd = pvsystem.singlediode(il, io, rs, rsh, nNsVth)

p\_dc = sd["p\_mp"] # W per module (approx)

# Temperature correction (gamma\_p relative to 25¬∞C)

delta\_T = tcell - 25.0

p\_dc\_corr = p\_dc \* (1.0 + gamma\_p \* delta\_T)

# Inverter model (Sandia/CEC); fallback to simple clipping

inv = sim.inverter.get("coefficients", {})

pac\_nameplate = float(sim.inverter.get("pac\_nameplate\_w", 2\_500\_000.0))

if inv:

try:

ac = pvsystem.snlinverter(p\_dc\_corr, v\_dc=None, inverter=inv)

pac = ac.clip(lower=0).clip(upper=pac\_nameplate)

except Exception:

pac = p\_dc\_corr.clip(lower=0).clip(upper=pac\_nameplate)

else:

pac = p\_dc\_corr.clip(lower=0).clip(upper=pac\_nameplate)

return pd.DataFrame({"p\_dc": p\_dc\_corr, "p\_mp\_raw": p\_dc}), pac

# -----------------------------

# Daily simulation & sweeping

# -----------------------------

def \_integrate\_energy\_Wh(power\_W: pd.Series) -> float:

"""

Integrate power(t) over the index using actual time deltas to get energy in Wh.

Handles irregular sampling; assumes tz-aware datetime index.

"""

if power\_W.empty:

return 0.0

s = power\_W.copy()

# dt to next sample in hours; last sample gets median dt to avoid drop

idx = s.index

dt\_next = (pd.Series(idx[1:].append(idx[-1:])).reset\_index(drop=True) - pd.Series(idx).reset\_index(drop=True))

dt\_next = dt\_next.dt.total\_seconds().astype(float) / 3600.0

if len(dt\_next) > 1:

dt\_next.iloc[-1] = float(np.nanmedian(dt\_next[:-1]))

else:

dt\_next.iloc[-1] = 1.0 # assume hourly if single point

dt\_h = pd.Series(dt\_next.values, index=s.index)

energy\_Wh = float((s.clip(lower=0) \* dt\_h).sum())

return energy\_Wh

def run\_daily(sim: SimConfig, weather\_csv: str, date\_local: str, offset\_deg: float) -> Dict[str, Any]:

"""

Run the full stack for one local date and return dict with series & summaries.

Steps:

- load local tz weather

- slice the day

- decompose (ERBS)

- tracker + POA (Hay‚ÄìDavies with dni\_extra)

- Faiman temperature

- DC/AC power

- integrate to Wh using actual dt

"""

\_ensure\_pvlib()

loc = location.Location(sim.lat, sim.lon, tz=sim.tz)

df\_loc = load\_weather\_qc(weather\_csv, sim.tz)

# Select the 24h window for date\_local (local midnight to next midnight)

target\_date = pd.to\_datetime(date\_local).date()

day\_mask = df\_loc.index.date == target\_date

day = df\_loc.loc[day\_mask].copy()

if day.empty:

raise RuntimeError(f"No rows found for {date\_local} in {weather\_csv}")

# Decompose + POA on tracker

dfd = erbs\_decompose(day, loc)

poa, trk = poa\_on\_tracker(dfd, sim, backtrack=True, offset\_deg=float(offset\_deg))

# Faiman (from configs)

try:

u0 = sim.models["temperature"]["baseline"]["parameters"]["u0\_W\_m2K"]

u1 = sim.models["temperature"]["baseline"]["parameters"]["u1\_W\_s\_m3K"]

except Exception:

# Sensible defaults if not present

u0, u1 = 25.0, 6.0

tc = faiman\_cell\_temp(poa["poa\_global"], dfd["temp\_air\_c"], dfd["wind\_speed\_ms"], u0, u1)

# DC/AC

dc\_df, pac = dc\_ac\_power(sim, poa, tc)

# Daily energy (Wh)

e\_day\_Wh = \_integrate\_energy\_Wh(pac)

return {

"time": day.index,

"poa": poa,

"tc": tc,

"pac": pac,

"dc": dc\_df,

"trk": trk,

"e\_day\_wh": float(e\_day\_Wh)

}

def sweep\_offsets(sim: SimConfig, weather\_csv: str, date\_local: str, min\_deg: int = -30, max\_deg: int = 30, step\_deg: int = 2) -> pd.DataFrame:

"""

Compute E\_day(Œ¥) sweep for a given date. Returns a DataFrame with columns:

offset\_deg, e\_day\_Wh

"""

rows = []

for d in range(int(min\_deg), int(max\_deg) + 1, int(step\_deg)):

try:

out = run\_daily(sim, weather\_csv, date\_local, offset\_deg=float(d))

rows.append({"offset\_deg": d, "e\_day\_Wh": out["e\_day\_wh"]})

except Exception:

rows.append({"offset\_deg": d, "e\_day\_Wh": np.nan})

return pd.DataFrame(rows)

#!/usr/bin/env python3

import argparse, os, numpy as np, pandas as pd, matplotlib.pyplot as plt

from sim\_support import build\_sim\_config, run\_daily

def parse\_args():

ap = argparse.ArgumentParser()

ap.add\_argument("--site", required=True)

ap.add\_argument("--scada", required=True)

ap.add\_argument("--weather", required=True)

ap.add\_argument("--offset-deg", type=float, default=0.0)

ap.add\_argument("--scale-sim", type=float, default=1.0)

ap.add\_argument("--auto-scale", action="store\_true")

return ap.parse\_args()

def load\_scada(path):

df = pd.read\_csv(path)

for req in ["site","date\_local","energy\_kWh"]:

if req not in df.columns: raise RuntimeError(f"SCADA missing {req}")

df["date\_local"] = pd.to\_datetime(df["date\_local"]).dt.date

df["E\_scada\_Wh"] = pd.to\_numeric(df["energy\_kWh"], errors="coerce")\*1000.0

return df[["site","date\_local","E\_scada\_Wh"]]

def main():

a = parse\_args()

os.makedirs("tables", exist\_ok=True); os.makedirs("figs", exist\_ok=True)

# Build sim for PVDAQ1430 (Denver) using your single-axis tracker

simcfg = build\_sim\_config(a.site, base\_dir=".")

scada = load\_scada(a.scada)

# Simulate each day in 2017 (dates from SCADA)

rows=[]

for \_, r in scada.iterrows():

d = r["date\_local"].isoformat()

try:

out = run\_daily(simcfg, a.weather, d, offset\_deg=a.offset\_deg) # <-- your tracker

e\_sim = float(out["e\_day\_wh"])

except Exception:

e\_sim = np.nan

rows.append({"date\_local":d,"E\_scada\_Wh":float(r["E\_scada\_Wh"]), "E\_sim\_Wh\_raw":e\_sim})

res = pd.DataFrame(rows)

# Auto-scale if requested

scale = float(a.scale\_sim)

if a.auto\_scale:

calib = res.dropna(subset=["E\_scada\_Wh","E\_sim\_Wh\_raw"]).copy()

if not calib.empty and (calib["E\_sim\_Wh\_raw"]>0).any():

s = (calib.E\_sim\_Wh\_raw\*calib.E\_scada\_Wh).sum() / (calib.E\_sim\_Wh\_raw\*\*2).sum()

if np.isfinite(s) and s>0: scale = float(s)

res["E\_sim\_Wh"] = res["E\_sim\_Wh\_raw"] \* scale

res["diff\_Wh"] = res["E\_sim\_Wh"] - res["E\_scada\_Wh"]

res["date\_local"] = pd.to\_datetime(res["date\_local"]).dt.date

res.to\_csv(f"tables/validation\_{a.site}.csv", index=False)

# Metrics

val = res.dropna(subset=["E\_scada\_Wh","E\_sim\_Wh"]).copy()

metrics={}

if not val.empty:

y, yhat = val["E\_scada\_Wh"].values, val["E\_sim\_Wh"].values

err = yhat - y

mbe = err.mean(); rmse = np.sqrt((err\*\*2).mean())

r2 = 1 - ((err\*\*2).sum())/((y-y.mean())\*\*2).sum() if len(y)>1 else np.nan

nrmse = rmse / (y.mean() if y.mean()!=0 else 1.0)

metrics = {"n":int(len(val)), "scale\_applied":float(scale),

"MBE\_Wh":float(mbe), "NMBE\_pct":float(100\*mbe/(y.mean() or 1.0)),

"RMSE\_Wh":float(rmse), "NRMSE\_pct":float(100\*nrmse), "R2":float(r2)}

pd.DataFrame([metrics]).to\_csv(f"tables/validation\_metrics\_{a.site}.csv", index=False)

# Plots

if not val.empty:

x = val["E\_scada\_Wh"]/1000.0; yk = val["E\_sim\_Wh"]/1000.0

lim=[0, max(x.max(), yk.max())\*1.05]

fig,ax=plt.subplots(figsize=(5,5)); ax.scatter(x,yk,s=45); ax.plot(lim,lim,"--",lw=1,color="gray")

if len(x)>=2:

m,b=np.polyfit(x,yk,1); xx=np.linspace(\*lim,200); ax.plot(xx,m\*xx+b,lw=1)

if metrics:

ax.text(0.02,0.98,f"scale={scale:.2f}\n"

f"n={metrics['n']}\n"

f"R¬≤={metrics['R2']:.3f}\n"

f"NMBE={metrics['NMBE\_pct']:.1f}%\n"

f"NRMSE={metrics['NRMSE\_pct']:.1f}%",

transform=ax.transAxes, va="top")

ax.set\_xlim(lim); ax.set\_ylim(lim)

ax.set\_xlabel("SCADA $E\_{day}$ [kWh]"); ax.set\_ylabel("Simulated $E\_{day}$ [kWh]")

ax.set\_title(f"Daily Energy Validation ‚Äî {a.site}")

fig.tight\_layout(); fig.savefig(f"figs/validation\_scatter\_{a.site}.pdf", bbox\_inches="tight"); plt.close(fig)

vts = res.sort\_values("date\_local").copy()

vts["date\_local"] = pd.to\_datetime(vts["date\_local"])

fig,ax=plt.subplots(figsize=(8.5,3.6))

ax.plot(vts["date\_local"], vts["E\_scada\_Wh"]/1000.0, marker="o", label="SCADA")

ax.plot(vts["date\_local"], vts["E\_sim\_Wh"]/1000.0, marker="s", label=f"Sim √ó {scale:.2f}")

ax.set\_ylabel("$E\_{day}$ [kWh]"); ax.set\_xlabel("Date (local)"); ax.legend()

ax.set\_title(f"{a.site} ‚Äî Daily Energy (SCADA vs Sim)")

fig.autofmt\_xdate(); fig.tight\_layout(); fig.savefig(f"figs/validation\_timeseries\_{a.site}.pdf", bbox\_inches="tight"); plt.close(fig)

print(f"Wrote: tables/validation\_{a.site}.csv")

if metrics: print("Metrics:", metrics)

print(f"Scatter: figs/validation\_scatter\_{a.site}.pdf")

print(f"Timeseries: figs/validation\_timeseries\_{a.site}.pdf")

if \_\_name\_\_ == "\_\_main\_\_": main()

from \_\_future\_\_ import annotations

import argparse

from pathlib import Path

import pandas as pd

import numpy as np

from tools.sim\_support import build\_sim\_config, run\_daily

ROOT = Path(".")

SEL = ROOT/"selection"

TABLES = ROOT/"tables"

DATA = ROOT/"data\_raw"

TABLES.mkdir(exist\_ok=True, parents=True)

def load\_daylist(site: str) -> pd.DataFrame:

fn = SEL/f"{site}\_2024\_day\_bins.csv"

df = pd.read\_csv(fn)

if "date\_local" not in df.columns and "date" in df.columns:

df = df.rename(columns={"date":"date\_local"})

if "bin" not in df.columns and "bin\_label" in df.columns:

df = df.rename(columns={"bin\_label":"bin"})

if "season" not in df.columns:

m = pd.to\_datetime(df["date\_local"]).dt.month

season = pd.cut(m, bins=[0,2,5,8,11,12],

labels=["Summer (DJF)","Autumn (MAM)","Winter (JJA)","Spring (SON)","Summer (DJF)"],

right=True, include\_lowest=True)

df["season"] = season.values

df["site"]=site

return df[["site","date\_local","bin","season"]].copy()

def site\_weather(site: str) -> str:

return str(DATA/f"{site}\_2024\_POWER\_qc.csv")

def bin\_median\_offsets() -> pd.DataFrame:

summ = pd.read\_csv(TABLES/"daily\_energy\_all.csv")

summ["bin\_code"] = summ["bin"].str.split(":").str[0]

g = summ.groupby(["site","bin\_code"])

med = g[["offset\_opt\_deg","delta\_pct"]].median().reset\_index()

med = med.rename(columns={"bin\_code":"bin","offset\_opt\_deg":"theta\_med\_deg","delta\_pct":"delta\_med\_pct"})

return med

def compute\_day\_energy(site: str, date: str, delta\_deg: float) -> float:

sim = build\_sim\_config(site, base\_dir=".")

out = run\_daily(sim, site\_weather(site), date\_local=date, offset\_deg=float(delta\_deg))

return float(out["pac"].clip(lower=0).sum()/1000.0) # kWh

def main():

ap = argparse.ArgumentParser()

ap.add\_argument("--scada", type=str, default=None, help="Optional SCADA daily CSV with columns: site,date\_local,energy\_kWh")

args = ap.parse\_args()

# If you only want validation (no SA sites set up), we can skip NC/WC gracefully

have\_nc = (SEL/"NC\_2024\_day\_bins.csv").exists()

have\_wc = (SEL/"WC\_2024\_day\_bins.csv").exists()

daylists = []

if have\_nc: daylists.append(load\_daylist("NC"))

if have\_wc: daylists.append(load\_daylist("WC"))

if daylists:

daylists = pd.concat(daylists, ignore\_index=True)

daylists["date\_local"] = pd.to\_datetime(daylists["date\_local"]).dt.strftime("%Y-%m-%d")

# Bin medians for SA sites (if present)

med = bin\_median\_offsets() if (TABLES/"daily\_energy\_all.csv").exists() else pd.DataFrame(columns=["site","bin","theta\_med\_deg","delta\_med\_pct"])

pol\_rows, daily\_rows = [], []

# Run SA (NC/WC) annual policy comparison if we have day lists

for site, g in (daylists.groupby("site") if isinstance(daylists, pd.DataFrame) and not daylists.empty else []):

th\_map = med[med["site"]==site].set\_index("bin")["theta\_med\_deg"].to\_dict()

b4\_theta = th\_map.get("B4", 0.0)

E0 = E\_b4 = E\_bin = 0.0

for \_, r in g.iterrows():

date = r["date\_local"]

b = (r["bin"].split(":")[0] if ":" in r["bin"] else r["bin"]).strip()

e0 = compute\_day\_energy(site, date, 0.0)

eb4 = compute\_day\_energy(site, date, b4\_theta if b=="B4" else 0.0)

ebin = compute\_day\_energy(site, date, th\_map.get(b, 0.0))

E0 += e0; E\_b4 += eb4; E\_bin += ebin

daily\_rows.append({"site":site,"date":date,"bin":b,

"E0\_kWh":e0,"E\_policyB4\_kWh":eb4,"E\_idealBin\_kWh":ebin,

"theta\_B4\_deg":b4\_theta,"theta\_bin\_deg":th\_map.get(b,0.0)})

if E0>0:

pol\_rows.append({"site":site,

"E0\_year\_kWh":E0,

"E\_policyB4\_year\_kWh":E\_b4,

"E\_idealBin\_year\_kWh":E\_bin,

"gain\_policyB4\_pct":100\*(E\_b4-E0)/E0,

"gain\_idealBin\_pct":100\*(E\_bin-E0)/E0})

if daily\_rows:

daily = pd.DataFrame(daily\_rows)

daily.to\_csv(TABLES/"annual\_policy\_daily\_log.csv", index=False)

if pol\_rows:

annual = pd.DataFrame(pol\_rows)

annual.to\_csv(TABLES/"annual\_policy\_by\_site.csv", index=False)

print("\nAnnual energy by site (kWh) and gains vs baseline (%):")

print(annual.to\_string(index=False))

else:

print("\nNo NC/WC annual policy run (day lists not found) ‚Äî proceeding to SCADA validation only.")

# SCADA validation (baseline model vs measured)

if args.scada:

sc = pd.read\_csv(args.scada)

sc["date\_local"]=pd.to\_datetime(sc["date\_local"]).dt.strftime("%Y-%m-%d")

# Validate for any site present in SCADA that also has a row in inputs/sites.csv

sites = sc["site"].unique().tolist()

vals = []

for site in sites:

# we need the site to exist in inputs/sites.csv so build\_sim\_config can run

try:

# model daily AC for all scada dates

dates = sc[sc["site"]==site]["date\_local"].tolist()

E0s = []

for d in dates:

try:

E0s.append({"date\_local":d, "E0\_kWh": compute\_day\_energy(site, d, 0.0)})

except Exception:

# if any missing weather day, skip that date

pass

if not E0s:

continue

mod = pd.DataFrame(E0s)

m = mod.merge(sc[sc["site"]==site][["date\_local","energy\_kWh"]], on="date\_local", how="inner")

if m.empty:

continue

y = m["energy\_kWh"].to\_numpy()

yhat = m["E0\_kWh"].to\_numpy()

mbe = float((yhat - y).mean())

nrmse = float(np.sqrt(np.mean((yhat-y)\*\*2)) / (y.mean() if y.mean()!=0 else 1.0))

vals.append({"site":site,"n\_days":len(m),"MBE\_kWh":mbe,"NRMSE\_frac":nrmse})

except Exception as e:

continue

if vals:

v = pd.DataFrame(vals)

v.to\_csv(TABLES/"validity\_stats.csv", index=False)

print("\nValidation vs SCADA (baseline model ‚Üí measured):")

print(v.to\_string(index=False))

else:

print("\nSCADA validation ran, but no overlapping dates/site config were usable (check inputs/sites.csv and weather files).")

else:

print("\nSCADA file not provided: run with --scada scada\_daily\_template.csv when ready.")

if \_\_name\_\_ == "\_\_main\_\_":

main()