SOP_96_Well

September 22, 2025

1 Import Packages

[2]:
```
import os, re
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter, LogLocator, NullFormatter
```

2 Setting Up the Workflow

2.0.1 This is the main section you will need to edit

Set working directory and bring in your data

[ ]:
```
WORKDIR = "EDITPATH"
FILENAME = "96_well_aggregated_data.xlsx"
SHEET = 0
OUT_BASENAME = "96_Well"
SAVE_TABLES = True
```

Verify the working directory is correct. This should be the file that contains

(1) This SOP

(2) The file titled 96_well_aggregated_data.xlsx

- Note, it might be alluring to change the name of the aggregated data file. If you do this you must insert the entire PATH.

[ ]:
```
sns.set_style("white")
if WORKDIR:
    os.makedirs(WORKDIR, exist_ok=True)
    os.chdir(WORKDIR)
print("CWD:", os.getcwd())
```

Write out your microbes and order

[ ]:
```
KNOWN_MICROBES = {"Sterile","MG1655","IGC16","IGC17"}
MICROBE_ORDER  = ["Sterile","MG1655","IGC16","IGC17"]
```

What do you want to plot?

[ ]:
```
# Choose one microbe and condition example to run at the bottom
SELECT_MICROBE   = "MG1655" # this will show all conditions for one microbe
SELECT_CONDITION = "R2A7"     # this will show all microbes for one condition
```

Plotting Style

[ ]:
```
import seaborn as sns, hashlib

PLOT = {
    # labels
    "x_label": "Days",
    "y_label": "OD600",

    # Y axis (log)
    "y_min": 1e-3,
    "y_max": 2.0,
    "y_lock": True,
    "yticks": [0.01, 0.05, 0.1, 0.5, 1, 2],

    # X axis ticks/labels
    "x_ticks_mode": "data",
    "x_tick_int_labels": True,

    # Style
    "line_style": "solid",
    "line_width": 2.5,
    "marker": "o",
    "marker_size": 10,
    "marker_edge_width": 1.2,
    "error_style": "band",
    "stats_source": "plot",
    "legend_loc": "upper center",
    "legend_ncol": 3,
    "title_size": 18,
    "label_size": 16,
    "tick_size": 13,
    "axes_linewidth": 2.0,
    "axes_edgecolor": "#CFCFCF",

# Choosing Colors: You can just keep the default colorblind one or ddefine␣
 ↪using the optional overrides
    "base_palette_microbe": "colorblind",
    "base_palette_condition": "tab20",
    "color_pool_microbe": 12,
    "color_pool_condition": 48,

# Optional color overrides with random examples. Not currently in effect.
```

```
    "microbe_colors": {
        # "Sterile": "#595959", "MG1655": "#1f77b4", "IGC16": "#2ca02c",␣
↪"IGC17": "#ff7f0e",
    },
    "condition_colors": {
        # "R2A7": "#7b8da8", "R2A9": "#66c2a5", "R2A3": "#a6cee3",
        # "Aluminum Chloride 2 ppm": "#e45756", "Aluminum Chloride 20 ppm":␣
↪"#f58518",
        # "Aluminum Chloride 200 ppm": "#72b7b2", "Ampicillin 0.5  g/mL":␣
↪"#7e57c2",
        # "Ampicillin 5  g/mL": "#54a24b",
    },
}
# Defining some color things here, but no editing needed
def _build_pool(base: str, pool_size: int):
    """Make a pool of visually distinct colors of length pool_size."""
    pal = []
    try:
        base_max = 20 if base == "tab20" else (10 if base == "colorblind" else␣
↪pool_size)
        pal.extend(sns.color_palette(base, n_colors=min(pool_size, base_max)))
    except Exception:
        pass
    if len(pal) < pool_size:
        pal.extend(sns.husl_palette(pool_size - len(pal), s=.80, l=.52))  #␣
↪fill with evenly spaced hues
    return pal[:pool_size]

def _stable_color_map_unique(keys, base_palette, overrides=None, pool_size=32):
    """Deterministic, collision-free mapping from labels → distinct colors."""
    keys = sorted([str(k) for k in keys], key=str)
    pool = _build_pool(base_palette, pool_size)
    taken = set()
    cmap = {}
    for k in keys:
        if overrides and k in overrides and overrides[k]:
            cmap[k] = overrides[k]
            continue
        h = int(hashlib.blake2b(k.encode(), digest_size=2).hexdigest(), 16)
        idx = h % len(pool)
        start = idx
        while idx in taken:
            idx = (idx + 1) % len(pool)
            if idx == start: break
        taken.add(idx)
        cmap[k] = pool[idx]
    return cmap
```

```
def _color_map_for(keys, kind="condition"):
    if kind == "microbe":
        return _stable_color_map_unique(
            keys,
            base_palette=PLOT.get("base_palette_microbe", "colorblind"),
            overrides=PLOT.get("microbe_colors"),
            pool_size=PLOT.get("color_pool_microbe", 12),
        )
    else:
        return _stable_color_map_unique(
            keys,
            base_palette=PLOT.get("base_palette_condition", "tab20"),
            overrides=PLOT.get("condition_colors"),
            pool_size=PLOT.get("color_pool_condition", 48),
        )
```

3 Misc Helper Cells

You should not need to edit this

[ ]:
```
def to_micro_symbols(s):
    if not isinstance(s, str): return s
    s = s.replace("_", " ")
    s = re.sub(r"(?i)\bug\b", "µg", s)
    s = re.sub(r"(?i)\buL\b", "µL", s)
    return s

def normalize_day(series):
    s = series.copy()
    if s.dropna().shape[0] and pd.api.types.is_datetime64_any_dtype(s.dropna()):
        base = s.dropna().min()
        return (s - base).dt.days.astype("Int64")
    s_str = s.astype(str)
    m = s_str.str.extract(r"(-?\d+)")[0]
    if m.notna().any():
        try: return m.astype("Int64")
        except: pass
    ymd = s_str.str.replace(r"[^\d]", "", regex=True)
    if ymd.str.len().between(6,8).all():
        try: return ymd.astype("Int64")
        except: pass
    codes, _ = pd.factorize(s_str)
    return pd.Series(codes, index=s.index, dtype="Int64")

def pick_col_fuzzy_exclude(candidates, cols,␣
 ↪exclude=("SampleMatrix","Condition")):
```

```
    for c in candidates:
        if c in cols and c not in exclude: return c
    canon = {re.sub(r"[\s_]+","", str(c).lower()): c for c in cols if c not in␣
 ↪exclude}
    for c in candidates:
        key = re.sub(r"[\s_]+","", c.lower())
        if key in canon: return canon[key]
    for c in candidates:
        key = re.sub(r"[\s_]+","", c.lower())
        for col in cols:
            if col in exclude: continue
            if key in re.sub(r"[\s_]+","", str(col).lower()): return col
    return None

def fmt_num(x):
    if pd.isna(x): return None
    try:
        v = float(x)
        return str(int(v)) if v.is_integer() else str(v)
    except Exception:
        return str(x)

def microfix(s: str) -> str:
    if not isinstance(s, str): return s
    s = s.replace("_", " ")
    s = s.replace("ug", " g").replace(" uL", "  L").replace("uL", " L")
    return s.strip()
```

[ ]:
```
def load_table(fname, sheet):
    try:
        return pd.read_excel(fname, sheet_name=sheet, header=1), 1
    except Exception:
        return pd.read_excel(fname, sheet_name=sheet, header=0), 0

df, header_used = load_table(FILENAME, SHEET)
print(f"Loaded {FILENAME} (header={header_used}); shape={df.shape}")

# Clean text columns
for c in df.select_dtypes(include=["object"]).columns:
    df[c] = df[c].map(to_micro_symbols)

# Detect key columns
cols = list(df.columns)
day_raw = next((c for c in ["Metadata_4","Day","Day_norm","Time","Timepoint"]␣
 ↪if c in cols), None)
od_col  = next((c for c in ["OD600","OD_600","OD","OD_600nm"] if c in cols),␣
 ↪None)
```

```
mic_col = next((c for c in ["Microbe","Strain","Group","Condition"] if c in␣
 ↪cols), None)

# Normalize Day → Day_norm
day_col = None
if day_raw:
    df["Day_norm"] = normalize_day(df[day_raw])
    day_col = "Day_norm"

print("Detected:", {"day": day_raw, "od": od_col, "microbe": mic_col, "day_col":
 ↪ day_col})
#df.head(3)
```

[ ]:
```
raw_cols = list(df.columns)

MATRIX_SRC = pick_col_fuzzy_exclude(["Matrix","Medium","Media","Sample␣
 ↪Matrix","Metadata_1"], raw_cols)
VALUE_SRC  = pick_col_fuzzy_exclude(["Final_Concentration","Final␣
 ↪Concentration","FinalConc","Conc","Concentration","Value","Metadata_2"],␣
 ↪raw_cols)
UNIT_SRC   = pick_col_fuzzy_exclude(["Unit","Units","Measurement␣
 ↪Unit","Metadata_3"], raw_cols)

# Only ffill the matrix/name (do NOT ffill value/unit)
if MATRIX_SRC and MATRIX_SRC in df.columns:
    df[MATRIX_SRC] = df[MATRIX_SRC].ffill()

def build_condition_from_raw(row):
    name = row.get(MATRIX_SRC) if MATRIX_SRC else None
    val  = row.get(VALUE_SRC)  if VALUE_SRC  else None
    unit = row.get(UNIT_SRC)   if UNIT_SRC   else None
    parts = []
    if pd.notna(name) and str(name).strip().lower() not in {"na","nan","-"}:
        parts.append(str(name).strip())
    v = fmt_num(val)
    if v and str(v).strip().lower() not in {"na","nan","-"}:
        parts.append(v)
    if pd.notna(unit) and str(unit).strip().lower() not in {"na","nan","-"}:
        parts.append(str(unit).strip())
    label = " ".join(parts).strip()
    return microfix(label if label else "-")

df["Condition_raw"] = df.apply(build_condition_from_raw, axis=1)

print("\n[RAW Condition sources]")
print("  MATRIX_SRC:", MATRIX_SRC)
print("  VALUE_SRC :", VALUE_SRC)
```

```
print("  UNIT_SRC  :", UNIT_SRC)
print("[RAW Condition counts (top 25)]")
print(df["Condition_raw"].value_counts().head(25).to_string())
#df[[MATRIX_SRC, VALUE_SRC, UNIT_SRC, "Condition_raw"]].head(10)
```

[ ]:
```
long_like = (od_col is not None) and (mic_col is not None)

if long_like:
    df_long = df.copy()
    used_od_col = od_col
    used_microbe_col = mic_col
    df_long["Condition"] = df["Condition_raw"]
else:
    microbe_cols = [c for c in cols if c in KNOWN_MICROBES]
    if not microbe_cols:
        not_microbe = {day_col} if day_col else set()
        not_microbe |= {"Row","Col","Well","Plate","Replicate", MATRIX_SRC or␣
 ↪"", VALUE_SRC or "", UNIT_SRC or "", "Condition_raw"}
        microbe_cols = [c for c in cols if c not in not_microbe and c]
    id_vars = [c for c in ["Row","Col","Well","Plate","Replicate"] if c in cols]
    id_vars += [x for x in [MATRIX_SRC, VALUE_SRC, UNIT_SRC, "Condition_raw"]␣
 ↪if x and x in cols]
    if day_col: id_vars.append(day_col)
    df_long = df.melt(id_vars=id_vars, value_vars=microbe_cols,
                      var_name="Microbe", value_name="OD600")
    used_od_col = "OD600"; used_microbe_col = "Microbe"
    df_long["Condition"] = df_long.apply(build_condition_from_raw, axis=1)

# log OD
df_long[used_od_col] = pd.to_numeric(df_long[used_od_col], errors="coerce")
df_long["OD600_raw"]  = df_long[used_od_col]
df_long["OD600_plot"] = df_long["OD600_raw"].where(df_long["OD600_raw"] > 0)


microbe_levels = list(pd.Index(df_long[used_microbe_col].dropna().unique()))
hue_order = MICROBE_ORDER if all(m in microbe_levels for m in MICROBE_ORDER)␣
 ↪else sorted(microbe_levels, key=lambda x: str(x))

print("\n[Overall conditions (top 25)]")
print(df_long["Condition"].value_counts().head(25).to_string())

if SELECT_MICROBE in df_long[used_microbe_col].unique():
    sub0 = df_long[df_long[used_microbe_col]==SELECT_MICROBE]
    print(f"\n[{SELECT_MICROBE}] conditions (top 25)")
    print(sub0["Condition"].value_counts().head(25).to_string())

#df_long.head(5)
```

[ ]:
```
row_present = "Row" in df_long.columns
col_present = "Col" in df_long.columns

well_like_col = None
if not (row_present and col_present):
    for c in df_long.columns:
        if c in {used_microbe_col, used_od_col, "OD600_raw", "OD600_plot",␣
 ↪"Condition", day_col} or c is None:
            continue
        vals = df_long[c].dropna().astype(str)
        if len(vals) and (vals.str.match(r"^[A-H](?:[1-9]|1[0-2])$").mean() > 0.
 ↪6):
            well_like_col = c
            break

if row_present:
    df_long["RowLetter"] = df_long["Row"].astype(str).str.strip().str.upper()
elif well_like_col:
    df_long["RowLetter"] = df_long[well_like_col].astype(str).str[0].str.upper()
else:
    df_long["RowLetter"] = pd.NA

if col_present:
    df_long["ColNum"] = pd.to_numeric(df_long["Col"], errors="coerce").
 ↪astype("Int64")
elif well_like_col:
    df_long["ColNum"] = pd.to_numeric(df_long[well_like_col].astype(str).str[1:
 ↪], errors="coerce").astype("Int64")
else:
    df_long["ColNum"] = pd.Series([pd.NA]*len(df_long), dtype="Int64")

group_keys = [k for k in [day_col, used_microbe_col, "Condition"] if (k is not␣
 ↪None and k in df_long.columns)]

def axis_levels_eq3(axis_col):
    if axis_col not in df_long.columns: return False
    tmp = (df_long.dropna(subset=[axis_col])
                  .groupby(group_keys, dropna=False, observed=True)[axis_col]
                  .nunique())
    return (len(tmp) > 0) and ((tmp == 3).mean() >= 0.8)

# Auto-pick replicate axis (or set REPLICATE_AXIS manually in Cell 1)
REPLICATE_AXIS = "auto" if "REPLICATE_AXIS" not in globals() else REPLICATE_AXIS
if REPLICATE_AXIS in {"RowLetter","ColNum"}:
    replicate_axis = REPLICATE_AXIS
else:
    prefer_col = axis_levels_eq3("ColNum")
```

```
    prefer_row = axis_levels_eq3("RowLetter")
    if   prefer_col and not prefer_row: replicate_axis = "ColNum"
    elif prefer_row and not prefer_col: replicate_axis = "RowLetter"
    elif prefer_col and prefer_row:     replicate_axis = "ColNum"
    else:                               replicate_axis = None

# Collapse across the *other* axis → ~3 replicates per group
if replicate_axis is not None and replicate_axis in df_long.columns:
    agg_keys = group_keys + [replicate_axis]
    df_reps = (df_long.groupby(agg_keys, dropna=False,␣
 ↪observed=True)["OD600_raw"]
                      .mean().reset_index())
    df_reps = df_reps.sort_values(agg_keys)
    df_reps["ReplicateID"] = df_reps.groupby(group_keys).cumcount() + 1
else:
    df_reps = df_long.sort_values(group_keys).copy()
    df_reps["ReplicateID"] = df_reps.groupby(group_keys).cumcount() + 1
    df_reps = df_reps[df_reps["ReplicateID"] <= 3].copy()

# Add log-safe values to reps too
df_reps["OD600_plot"] = df_reps["OD600_raw"].where(df_reps["OD600_raw"] > 0)

# Summary + optional saves
has_day = (day_col is not None) and (day_col in df_reps.columns)
time_levels = sorted([x for x in df_reps[day_col].dropna().unique()]) if␣
 ↪has_day else []
groupers_counts = [used_microbe_col, "Condition"]
if has_day: groupers_counts = [day_col] + groupers_counts
rep_counts = (df_reps.groupby(groupers_counts, dropna=False,␣
 ↪observed=True)["OD600_raw"].size())
min_rep = int(rep_counts.min()) if len(rep_counts) else 0
max_rep = int(rep_counts.max()) if len(rep_counts) else 0
uniform_reps = bool(len(rep_counts) and (min_rep == max_rep))

print(f"There are {len(time_levels) if has_day else 'N/A'} timepoints denoted␣
 ↪by {day_col if has_day else 'N/A'}{': ' + str(time_levels) if has_day else␣
 ↪''}.")
print(f"Microbes detected: {sorted(df_reps[used_microbe_col].dropna().unique(),␣
 ↪key=str)}")
if len(rep_counts):
    msg = (f"There are {min_rep} datapoints for each (timepoint × microbe ×␣
 ↪Condition)."
           if uniform_reps else f"Replicates per group range from {min_rep} to␣
 ↪{max_rep}.")
    print(msg)
```

```
if SAVE_TABLES:
    df_long.to_csv(f"{OUT_BASENAME}_cleaned_long.csv", index=False)
    df_reps.to_csv(f"{OUT_BASENAME}_replicates_table.csv", index=False)
    rep_counts.reset_index(name="N").sort_values(groupers_counts).
 ↪to_csv(f"{OUT_BASENAME}_replicate_counts.csv", index=False)
    print("Saved cleaned_long, replicates_table, replicate_counts CSVs.")

#df_reps.head(3)
```

[ ]:
```
def _first_present(cols, df):
    for c in cols:
        if c and c in df.columns and df[c].notna().any():
            return c
    return None

# Prefer your computed Day_norm; fallback to other time-ish columns; else an␣
 ↪index
DAY_FOR_PLOT = None
candidates = [day_col, "Day_norm", "Day", "Timepoint", "Time", "Metadata_4"]
DAY_FOR_PLOT = _first_present(candidates, df_reps)

if DAY_FOR_PLOT is None:
    # build a simple increasing index per (Microbe × Condition)
    df_reps = df_reps.sort_values([ "Condition", "Microbe"]).copy()
    df_reps["Day_index"] = df_reps.groupby(["Microbe", "Condition"]).cumcount()
    DAY_FOR_PLOT = "Day_index"
    print("No day/time column detected → using synthetic index Day_index.")
else:
    # normalize if it looks like dates/strings
    try:
        from pandas.api.types import is_datetime64_any_dtype
        if is_datetime64_any_dtype(df_reps[DAY_FOR_PLOT]):
            base = df_reps[DAY_FOR_PLOT].dropna().min()
            df_reps["Day_norm_auto"] = (df_reps[DAY_FOR_PLOT] - base).dt.days.
 ↪astype("Int64")
            DAY_FOR_PLOT = "Day_norm_auto"
        elif df_reps[DAY_FOR_PLOT].dtype == object:
            # try to coerce to ints
            tmp = pd.to_numeric(df_reps[DAY_FOR_PLOT], errors="coerce")
            if tmp.notna().any():
                df_reps["Day_norm_auto"] = tmp.astype("Int64")
                DAY_FOR_PLOT = "Day_norm_auto"
    except Exception:
        pass

print("Using X-axis column:", DAY_FOR_PLOT)
```

```
# Ensure a log-safe y column exists in reps (zeros → NaN)
if "OD600_plot" not in df_reps.columns:
    df_reps["OD600_plot"] = df_reps["OD600_raw"].where(df_reps["OD600_raw"] > 0)

# Helpful: pick a real condition that exists if your default didn't
if SELECT_CONDITION not in set(df_reps["Condition"].astype(str)):
    SELECT_CONDITION = sorted(df_reps["Condition"].dropna().astype(str).
 ↪unique(), key=str)[0]
    print("SELECT_CONDITION not found; using:", SELECT_CONDITION)
if SELECT_MICROBE not in set(df_reps["Microbe"].astype(str)):
    SELECT_MICROBE = sorted(df_reps["Microbe"].dropna().astype(str).unique(),␣
 ↪key=str)[0]
    print("SELECT_MICROBE not found; using:", SELECT_MICROBE)
```

4 Main plotting functions

[ ]:
```
def plot_microbe_all_conditions(
    df_reps, microbe_name, day_col="Day_norm",
    od_col="OD600_plot", microbe_col="Microbe",
    out_base="96_Well_"
):
    sub = df_reps[df_reps[microbe_col] == microbe_name].copy()
    if sub.empty:
        raise ValueError(f"No rows for microbe '{microbe_name}'.")
    sub["Condition"] = sub["Condition"].astype(str)

    ystat = "OD600_plot" if (PLOT["stats_source"] == "plot" and "OD600_plot" in␣
 ↪sub.columns) else "OD600_raw"
    summ = (sub.groupby([day_col, "Condition"], dropna=False,␣
 ↪observed=True)[ystat]
              .agg(N="count", mean="mean", sd="std").reset_index())

    conds = sorted(summ["Condition"].dropna().unique(), key=str)
    cmap = _color_map_for(conds, kind="condition")


    fig, ax = plt.subplots(figsize=(7.2, 6.6), dpi=180)

    # choose x ticks
    if PLOT["x_ticks_mode"] == "fixed":
        x_ticks = list(PLOT["x_ticks_fixed"])
    else:
        x_ticks = sorted(pd.unique(summ[day_col].dropna()))
    # optional integer labels
    if PLOT["x_tick_int_labels"]:
        ax.set_xticks(x_ticks)
```

```
        ax.set_xticklabels([f"{int(x)}" for x in x_ticks])
    else:
        ax.set_xticks(x_ticks)

    # draw lines
    for cond in conds:
        d = summ[summ["Condition"] == cond].sort_values(by=day_col)
        x  = d[day_col].to_numpy()
        y  = d["mean"].to_numpy()
        sd = d["sd"].fillna(0).to_numpy()

        # SD band/bars
        if PLOT["error_style"] == "band" and len(x) >= 2:
            y1 = np.clip(y - sd, a_min=PLOT["y_min"], a_max=None)
            y2 = np.clip(y + sd, a_min=PLOT["y_min"], a_max=None)
            ax.fill_between(x, y1, y2, alpha=0.25, color=cmap[cond],␣
↪linewidth=0, zorder=4)

        ax.plot(x, y, label=str(cond),
                color=cmap[cond], linewidth=PLOT["line_width"],␣
↪linestyle=PLOT["line_style"],
                marker=PLOT["marker"], markersize=PLOT["marker_size"],
                markerfacecolor=cmap[cond], markeredgecolor="white",
                markeredgewidth=PLOT["marker_edge_width"], alpha=0.98, zorder=3)

        if PLOT["error_style"] == "bars" or len(x) < 2:
            ax.errorbar(x, y, yerr=sd, fmt="none", ecolor=cmap[cond],␣
↪elinewidth=3.0,
                        capsize=6, capthick=3.0, alpha=0.95, zorder=4)

    # Y axis: log + fixed ticks/limits
    ax.set_yscale("log")
    if PLOT.get("y_lock") and PLOT.get("y_max"):
        ax.set_ylim(PLOT["y_min"], PLOT["y_max"])
    else:
        ymax = float(np.nanmax(summ["mean"])) if len(summ) else PLOT["y_min"]*10
        ax.set_ylim(bottom=PLOT["y_min"], top=max(PLOT["y_min"]*10, ymax*1.3))

    ymin_now, ymax_now = ax.get_ylim()
    ax.set_yticks([t for t in PLOT["yticks"] if ymin_now <= t <= ymax_now])
    ax.yaxis.set_major_formatter(ScalarFormatter())
    ax.yaxis.set_minor_locator(LogLocator(base=10.0, subs=(), numticks=3))
    ax.yaxis.set_minor_formatter(NullFormatter())

    # Cosmetics
    ax.set_xlabel(PLOT["x_label"]); ax.set_ylabel(PLOT["y_label"])
```

```
    ax.set_title(f"{microbe_name}: {PLOT['y_label']} vs {day_col} by␣
↪condition", pad=12)
    for s in ["top","right"]: ax.spines[s].set_visible(False)
    ax.tick_params(length=0)

    ax.legend(title="Condition", loc=PLOT["legend_loc"], bbox_to_anchor=(0.5,␣
↪-0.18),
              ncol=PLOT["legend_ncol"], frameon=True, fancybox=True,␣
↪framealpha=0.85)

    plt.tight_layout(); fig.subplots_adjust(bottom=0.25)
    slug = re.sub(r"[^A-Za-z0-9]+","_", str(microbe_name)).strip("_")
    plt.savefig(f"{out_base}_{slug}_byCondition.png", dpi=300)
    plt.savefig(f"{out_base}_{slug}_byCondition.pdf")
    plt.show()


def plot_condition_all_microbes(
    df_reps, condition_name, day_col="Day_norm",
    microbe_col="Microbe", od_col="OD600_plot",
    microbe_order=None, out_base="IGC_style_from_96W"
):
    sub = df_reps[df_reps["Condition"].astype(str) == str(condition_name)].
↪copy()
    if sub.empty:
        raise ValueError(f"No rows for condition '{condition_name}'.")

    ystat = "OD600_plot" if (PLOT["stats_source"] == "plot" and "OD600_plot" in␣
↪sub.columns) else "OD600_raw"
    summ = (sub.groupby([day_col, microbe_col], dropna=False,␣
↪observed=True)[ystat]
              .agg(N="count", mean="mean", sd="std").reset_index())

    microbes = list(summ[microbe_col].dropna().unique())
    if microbe_order:
        order = [m for m in microbe_order if m in microbes] + [m for m in␣
↪microbes if m not in (microbe_order or [])]
    else:
        order = sorted(microbes, key=str)

    cmap = _color_map_for(order, kind="microbe")


    fig, ax = plt.subplots(figsize=(7.2, 6.6), dpi=180)

    # choose x ticks
```

```
    if PLOT["x_ticks_mode"] == "fixed":
        x_ticks = list(PLOT["x_ticks_fixed"])
    else:
        x_ticks = sorted(pd.unique(summ[day_col].dropna()))
    if PLOT["x_tick_int_labels"]:
        ax.set_xticks(x_ticks)
        ax.set_xticklabels([f"{int(x)}" for x in x_ticks])
    else:
        ax.set_xticks(x_ticks)

    # draw lines
    for m in order:
        d  = summ[summ[microbe_col] == m].sort_values(by=day_col)
        x  = d[day_col].to_numpy()
        y  = d["mean"].to_numpy()
        sd = d["sd"].fillna(0).to_numpy()

        if PLOT["error_style"] == "band" and len(x) >= 2:
            y1 = np.clip(y - sd, a_min=PLOT["y_min"], a_max=None)
            y2 = np.clip(y + sd, a_min=PLOT["y_min"], a_max=None)
            ax.fill_between(x, y1, y2, alpha=0.25, color=cmap[m], linewidth=0,␣
↪zorder=4)

        ax.plot(x, y, label=str(m),
                color=cmap[m], linewidth=PLOT["line_width"],␣
↪linestyle=PLOT["line_style"],
                marker=PLOT["marker"], markersize=PLOT["marker_size"],
                markerfacecolor=cmap[m], markeredgecolor="white",
                markeredgewidth=PLOT["marker_edge_width"], alpha=0.98, zorder=3)

        if PLOT["error_style"] == "bars" or len(x) < 2:
            ax.errorbar(x, y, yerr=sd, fmt="none", ecolor=cmap[m], elinewidth=3.
↪0,
                        capsize=6, capthick=3.0, alpha=0.95, zorder=4)

    # Y axis: log + fixed ticks/limits
    ax.set_yscale("log")
    if PLOT.get("y_lock") and PLOT.get("y_max"):
        ax.set_ylim(PLOT["y_min"], PLOT["y_max"])
    else:
        ymax = float(np.nanmax(summ["mean"])) if len(summ) else PLOT["y_min"]*10
        ax.set_ylim(bottom=PLOT["y_min"], top=max(PLOT["y_min"]*10, ymax*1.3))

    ymin_now, ymax_now = ax.get_ylim()
    ax.set_yticks([t for t in PLOT["yticks"] if ymin_now <= t <= ymax_now])
    ax.yaxis.set_major_formatter(ScalarFormatter())
    ax.yaxis.set_minor_locator(LogLocator(base=10.0, subs=(), numticks=3))
```

```
    ax.yaxis.set_minor_formatter(NullFormatter())

    ax.set_xlabel(PLOT["x_label"]); ax.set_ylabel(PLOT["y_label"])
    ax.set_title(f"{condition_name}: {PLOT['y_label']} vs {day_col} by␣
 ↪microbe", pad=12)
    for s in ["top","right"]: ax.spines[s].set_visible(False)
    ax.tick_params(length=0)

    ax.legend(title="Microbe", loc=PLOT["legend_loc"], bbox_to_anchor=(0.5, -0.
 ↪18),
              ncol=PLOT["legend_ncol"], frameon=True, fancybox=True,␣
 ↪framealpha=0.85)

    plt.tight_layout(); fig.subplots_adjust(bottom=0.25)
    slug = re.sub(r"[^A-Za-z0-9]+","_", str(condition_name)).strip("_")
    plt.savefig(f"{out_base}_COND_{slug}_byMicrobe.png", dpi=300)
    plt.savefig(f"{out_base}_COND_{slug}_byMicrobe.pdf")
    plt.show()

# This cell will fail unless you want to do work with just one micorbe
# One microbe, lines = conditions
plot_microbe_all_conditions(
    df_reps,
    microbe_name=SELECT_MICROBE,
    day_col=DAY_FOR_PLOT,          # << use the safe x-axis
    od_col="OD600_plot",
    microbe_col="Microbe",
    out_base=OUT_BASENAME
)

# One condition, lines = microbes
plot_condition_all_microbes(
    df_reps,
    condition_name=SELECT_CONDITION,
    day_col=DAY_FOR_PLOT,          # << use the safe x-axis
    microbe_col="Microbe",
    od_col="OD600_plot",
    microbe_order=MICROBE_ORDER,
    out_base=OUT_BASENAME
)
```

[ ]:
```
import os, re
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter, LogLocator, NullFormatter
from matplotlib.backends.backend_pdf import PdfPages
```

```python
def _save_fig(
    fig, base_stub, slug, suffix,
    png_dir=None, pdf_dir=None, save_individual_pdf=False
):
    """
    Save one figure as PNG (always) and optionally as an individual PDF.
    Returns (png_path, pdf_path_or_None).
    """
    os.makedirs(png_dir or ".", exist_ok=True)
    if pdf_dir:
        os.makedirs(pdf_dir, exist_ok=True)

    stem = f"{base_stub}_{slug}_{suffix}"
    png_path = os.path.join(png_dir or ".", f"{stem}.png")
    fig.savefig(png_path, dpi=300)

    pdf_path = None
    if save_individual_pdf and pdf_dir:
        pdf_path = os.path.join(pdf_dir, f"{stem}.pdf")
        fig.savefig(pdf_path)

    return png_path, pdf_path


def plot_microbe_all_conditions(
    df_reps, microbe_name, day_col="Day_norm",
    od_col="OD600_plot", microbe_col="Microbe",
    out_stub="IGC_style_from_96W",
    png_dir=None, pdf_dir=None,                # << new
    save_individual_pdf=False,                 # << new
    pdf_pages=None,                            # << new (PdfPages aggregator)
    show=True
):
    sub = df_reps[df_reps[microbe_col] == microbe_name].copy()
    if sub.empty:
        raise ValueError(f"No rows for microbe '{microbe_name}'.")
    sub["Condition"] = sub["Condition"].astype(str)

    ystat = "OD600_plot" if (PLOT["stats_source"] == "plot" and "OD600_plot" in␣
 ↪sub.columns) else "OD600_raw"
    summ = (sub.groupby([day_col, "Condition"], dropna=False,␣
 ↪observed=True)[ystat]
              .agg(N="count", mean="mean", sd="std").reset_index())

    conds = sorted(summ["Condition"].dropna().unique(), key=str)
    cmap  = _color_map_for(conds, kind="condition")
```

```
    fig, ax = plt.subplots(figsize=(7.2, 6.6), dpi=180)

    # X ticks
    x_ticks = (sorted(pd.unique(summ[day_col].dropna()))
               if PLOT["x_ticks_mode"] == "data"
               else list(PLOT.get("x_ticks_fixed", [])))
    ax.set_xticks(x_ticks)
    if PLOT.get("x_tick_int_labels", True):
        ax.set_xticklabels([f"{int(round(x))}" for x in x_ticks])

    # Lines
    ls = "-" if PLOT.get("line_style") in (None, "solid") else␣
↪PLOT["line_style"]
    for cond in conds:
        d  = summ[summ["Condition"] == cond].sort_values(by=day_col)
        x  = d[day_col].to_numpy()
        y  = d["mean"].to_numpy()
        sd = d["sd"].fillna(0).to_numpy()

        if PLOT["error_style"] == "band" and len(x) >= 2:
            y1 = np.clip(y - sd, a_min=PLOT["y_min"], a_max=None)
            y2 = np.clip(y + sd, a_min=PLOT["y_min"], a_max=None)
            ax.fill_between(x, y1, y2, alpha=0.25, color=cmap[cond],␣
↪linewidth=0, zorder=4)

        ax.plot(x, y, label=str(cond),
                color=cmap[cond], linewidth=PLOT["line_width"], linestyle=ls,
                marker=PLOT["marker"], markersize=PLOT["marker_size"],
                markerfacecolor=cmap[cond], markeredgecolor="white",
                markeredgewidth=PLOT["marker_edge_width"], alpha=0.98, zorder=3)

        if PLOT["error_style"] == "bars" or len(x) < 2:
            ax.errorbar(x, y, yerr=sd, fmt="none", ecolor=cmap[cond],␣
↪elinewidth=3.0,
                        capsize=6, capthick=3.0, alpha=0.95, zorder=4)

    # Y axis
    ax.set_yscale("log")
    if PLOT.get("y_lock") and PLOT.get("y_max"):
        ax.set_ylim(PLOT["y_min"], PLOT["y_max"])
    else:
        ymax = float(np.nanmax(summ["mean"])) if len(summ) else PLOT["y_min"]*10
        ax.set_ylim(bottom=PLOT["y_min"], top=max(PLOT["y_min"]*10, ymax*1.3))

    ymin_now, ymax_now = ax.get_ylim()
    ax.set_yticks([t for t in PLOT["yticks"] if t > 0 and ymin_now <= t <=␣
↪ymax_now])
```

```
    ax.yaxis.set_major_formatter(ScalarFormatter())
    ax.yaxis.set_minor_locator(LogLocator(base=10.0, subs=(), numticks=3))
    ax.yaxis.set_minor_formatter(NullFormatter())

    ax.set_xlabel(PLOT["x_label"]); ax.set_ylabel(PLOT["y_label"])
    ax.set_title(f"{microbe_name}: {PLOT['y_label']} vs {day_col} by␣
 ↪condition", pad=12)
    for s in ["top","right"]: ax.spines[s].set_visible(False)
    ax.tick_params(length=0)
    ax.legend(title="Condition", loc=PLOT["legend_loc"], bbox_to_anchor=(0.5,␣
 ↪-0.18),
              ncol=PLOT["legend_ncol"], frameon=True, fancybox=True,␣
 ↪framealpha=0.85)

    plt.tight_layout(); fig.subplots_adjust(bottom=0.25)

    slug = re.sub(r"[^A-Za-z0-9]+", "_", str(microbe_name)).strip("_")
    png, _ = _save_fig(
        fig, out_stub, slug, "byCondition",
        png_dir=png_dir, pdf_dir=pdf_dir,
        save_individual_pdf=save_individual_pdf
    )
    if pdf_pages is not None:
        pdf_pages.savefig(fig)

    if show: plt.show()
    else: plt.close(fig)
    return png


def plot_condition_all_microbes(
    df_reps, condition_name, day_col="Day_norm",
    microbe_col="Microbe", od_col="OD600_plot",
    microbe_order=None,
    out_stub="IGC_style_from_96W",
    png_dir=None, pdf_dir=None,
    save_individual_pdf=False,
    pdf_pages=None,
    show=True
):
    sub = df_reps[df_reps["Condition"].astype(str) == str(condition_name)].
 ↪copy()
    if sub.empty:
        raise ValueError(f"No rows for condition '{condition_name}'.")
    sub = sub.dropna(subset=[od_col, day_col, microbe_col]).copy()
```

```
    ystat = "OD600_plot" if (PLOT["stats_source"] == "plot" and "OD600_plot" in␣
↪sub.columns) else "OD600_raw"
    summ = (sub.groupby([day_col, microbe_col], dropna=False,␣
↪observed=True)[ystat]
              .agg(N="count", mean="mean", sd="std").reset_index())

    microbes = list(summ[microbe_col].dropna().unique())
    if microbe_order:
        order = [m for m in microbe_order if m in microbes] + [m for m in␣
↪microbes if m not in microbe_order]
    else:
        order = sorted(microbes, key=str)

    cmap = _color_map_for(order, kind="microbe")
    fig, ax = plt.subplots(figsize=(7.2, 6.6), dpi=180)

    x_ticks = (sorted(pd.unique(summ[day_col].dropna()))
               if PLOT["x_ticks_mode"] == "data"
               else list(PLOT.get("x_ticks_fixed", [])))
    ax.set_xticks(x_ticks)
    if PLOT.get("x_tick_int_labels", True):
        ax.set_xticklabels([f"{int(round(x))}" for x in x_ticks])

    ls = "-" if PLOT.get("line_style") in (None, "solid") else␣
↪PLOT["line_style"]
    for m in order:
        d  = summ[summ[microbe_col] == m].sort_values(by=day_col)
        x  = d[day_col].to_numpy()
        y  = d["mean"].to_numpy()
        sd = d["sd"].fillna(0).to_numpy()

        if PLOT["error_style"] == "band" and len(x) >= 2:
            y1 = np.clip(y - sd, a_min=PLOT["y_min"], a_max=None)
            y2 = np.clip(y + sd, a_min=PLOT["y_min"], a_max=None)
            ax.fill_between(x, y1, y2, alpha=0.25, color=cmap[m], linewidth=0,␣
↪zorder=4)

        ax.plot(x, y, label=str(m),
                color=cmap[m], linewidth=PLOT["line_width"], linestyle=ls,
                marker=PLOT["marker"], markersize=PLOT["marker_size"],
                markerfacecolor=cmap[m], markeredgecolor="white",
                markeredgewidth=PLOT["marker_edge_width"], alpha=0.98, zorder=3)

        if PLOT["error_style"] == "bars" or len(x) < 2:
            ax.errorbar(x, y, yerr=sd, fmt="none", ecolor=cmap[m], elinewidth=3.
↪0,
```

```
                        capsize=6, capthick=3.0, alpha=0.95, zorder=4)

    ax.set_yscale("log")
    if PLOT.get("y_lock") and PLOT.get("y_max"):
        ax.set_ylim(PLOT["y_min"], PLOT["y_max"])
    else:
        ymax = float(np.nanmax(summ["mean"])) if len(summ) else PLOT["y_min"]*10
        ax.set_ylim(bottom=PLOT["y_min"], top=max(PLOT["y_min"]*10, ymax*1.3))

    ymin_now, ymax_now = ax.get_ylim()
    ax.set_yticks([t for t in PLOT["yticks"] if t > 0 and ymin_now <= t <=␣
↪ymax_now])
    ax.yaxis.set_major_formatter(ScalarFormatter())
    ax.yaxis.set_minor_locator(LogLocator(base=10.0, subs=(), numticks=3))
    ax.yaxis.set_minor_formatter(NullFormatter())

    ax.set_xlabel(PLOT["x_label"]); ax.set_ylabel(PLOT["y_label"])
    ax.set_title(f"{condition_name}: {PLOT['y_label']} vs {day_col} by␣
↪microbe", pad=12)
    for s in ["top","right"]: ax.spines[s].set_visible(False)
    ax.tick_params(length=0)
    ax.legend(title="Microbe", loc=PLOT["legend_loc"], bbox_to_anchor=(0.5, -0.
↪18),
              ncol=PLOT["legend_ncol"], frameon=True, fancybox=True,␣
↪framealpha=0.85)

    plt.tight_layout(); fig.subplots_adjust(bottom=0.25)

    slug = re.sub(r"[^A-Za-z0-9]+", "_", str(condition_name)).strip("_")
    png, _ = _save_fig(
        fig, out_stub, f"COND_{slug}", "byMicrobe",
        png_dir=png_dir, pdf_dir=pdf_dir,
        save_individual_pdf=save_individual_pdf
    )
    if pdf_pages is not None:
        pdf_pages.savefig(fig)

    if show: plt.show()
    else: plt.close(fig)
    return png
```

[ ]:
```
from datetime import datetime

BATCH = {
    "run_microbe_by_condition": True,     # one page per microbe (lines =␣
↪conditions)
```

```
    "run_condition_by_microbe": True,      # one page per condition (lines =␣
 ↪microbes)
    "png_dir": "plots_png",                # all PNGs here
    "pdf_dir": "plots_pdf",                # all PDFs here
    "base_stub": OUT_BASENAME,             # used in filenames
    "show": False,
    "save_individual_pdf": False,
    "min_points": 1,
}

os.makedirs(BATCH["png_dir"], exist_ok=True)
os.makedirs(BATCH["pdf_dir"], exist_ok=True)

all_microbes    = sorted(df_reps["Microbe"].dropna().astype(str).unique(),␣
 ↪key=str)
all_conditions = sorted(df_reps["Condition"].dropna().astype(str).unique(),␣
 ↪key=str)

def _panel_has_enough(sub, day_col, min_points):
    try:
        return (sub[day_col].notna() & sub["OD600_plot"].notna()).sum() >=␣
 ↪min_points
    except Exception:
        return len(sub) >= min_points

made_png = []

pp_microbes    = PdfPages(os.path.join(BATCH["pdf_dir"],␣
 ↪f"{BATCH['base_stub']}_AllMicrobes_byCondition.pdf")) \
                if BATCH["run_microbe_by_condition"] else None
pp_conditions = PdfPages(os.path.join(BATCH["pdf_dir"],␣
 ↪f"{BATCH['base_stub']}_AllConditions_byMicrobe.pdf")) \
                if BATCH["run_condition_by_microbe"] else None

if BATCH["run_microbe_by_condition"]:
    print(f"\nExporting microbe panels → conditions ({len(all_microbes)}␣
 ↪microbes)")
    for m in all_microbes:
        sub = df_reps[df_reps["Microbe"].astype(str) == m]
        if not _panel_has_enough(sub, DAY_FOR_PLOT, BATCH["min_points"]):
            print(f"  skip {m}: not enough points"); continue
        try:
            png = plot_microbe_all_conditions(
                df_reps, m, day_col=DAY_FOR_PLOT,
                out_stub=BATCH["base_stub"],
                png_dir=BATCH["png_dir"], pdf_dir=BATCH["pdf_dir"],
```

```
                save_individual_pdf=BATCH["save_individual_pdf"],
                pdf_pages=pp_microbes,
                show=BATCH["show"]
            )
            made_png.append(png); print(f"    {m}")
        except Exception as e:
            print(f"    {m} → {e}")

# Condition → Microbes
if BATCH["run_condition_by_microbe"]:
    print(f"\nExporting condition panels → microbes ({len(all_conditions)}␣
 ↪conditions)")
    for c in all_conditions:
        sub = df_reps[df_reps["Condition"].astype(str) == c]
        if not _panel_has_enough(sub, DAY_FOR_PLOT, BATCH["min_points"]):
            print(f"  skip {c}: not enough points"); continue
        try:
            png = plot_condition_all_microbes(
                df_reps, c, day_col=DAY_FOR_PLOT,
                microbe_order=MICROBE_ORDER,
                out_stub=BATCH["base_stub"],
                png_dir=BATCH["png_dir"], pdf_dir=BATCH["pdf_dir"],
                save_individual_pdf=BATCH["save_individual_pdf"],
                pdf_pages=pp_conditions,
                show=BATCH["show"]
            )
            made_png.append(png); print(f"    {c}")
        except Exception as e:
            print(f"    {c} → {e}")

if pp_microbes:   pp_microbes.close()
if pp_conditions: pp_conditions.close()

print("\nDone.")
print(f"PNGs saved in: {os.path.abspath(BATCH['png_dir'])}")
print(f"PDFs saved in: {os.path.abspath(BATCH['pdf_dir'])}")
print("Compiled PDFs:")
if BATCH["run_microbe_by_condition"]:
    print(" -", os.path.join(BATCH["pdf_dir"],␣
 ↪f"{BATCH['base_stub']}_AllMicrobes_byCondition.pdf"))
if BATCH["run_condition_by_microbe"]:
    print(" -", os.path.join(BATCH["pdf_dir"],␣
 ↪f"{BATCH['base_stub']}_AllConditions_byMicrobe.pdf"))
```

[ ]: