

## Site\_8\_Gam\_Tuned\_8020

October 12, 2025

```
[3]: # CSV-based plotting with user-controlled MAE/RMSE decimals in the metrics box.
# How to control decimals:
#   - Set `metrics_fmt_mae` and/or `metrics_fmt_rmse` to a Python format string.
#   - Examples:
#       metrics_fmt_mae="{:.1f}", metrics_fmt_rmse="{:.1f}"    -> 1 decimal (e.g.
#       ↪, 6.2, 9.4)
#       metrics_fmt_mae="{:.2f}", metrics_fmt_rmse="{:.3f}"    -> 2 and 3
#       ↪decimals
#       metrics_fmt_mae="{:g}", metrics_fmt_rmse="{:g}"        -> trimmed (no
#       ↪trailing zeros)
#
# You can ALSO force the legend locations and keep all prior behavior (red
# ↪markers for True==0 dates).

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.offsetbox import AnchoredText
from typing import Optional
from math import sqrt

def _legend_dedup(ax, loc="best"):
    handles, labels = ax.get_legend_handles_labels()
    if not labels:
        return
    dd = {}
    for h, l in zip(handles, labels):
        if l not in dd:
            dd[l] = h
    ax.legend(list(dd.values()), list(dd.keys()), loc=loc)

def plot_test_series_csv(csv_path: str,
                        title: str = "Adults_8_Col",
                        # If not provided, will try CSV columns 'MAE'/'RMSE';
                        ↪else compute
                        mae: Optional[float] = None,
```

```

rmse: Optional[float] = None,
# >>> Set your decimals here (format strings) <<<
metrics_fmt_mae: Optional[str] = "{:g}",
metrics_fmt_rmse: Optional[str] = "{:g}",
# Layout / formatting
date_pad_days: int = 3,
scatter_pad_frac: float = 0.06,
scatter_min_pad: float = 0.5,
metrics_loc: str = "upper right",      # MAE/RMSE box
↪position

time_legend_loc: str = "upper left",  # regular legend
↪on Plot 1

other_legend_loc: str = "best",       # legend on
↪other plots

    parse_dates_col: str = "Date"):
# ---- ingest CSV (auto-sep sniff) ----
df = pd.read_csv(csv_path, sep=None, engine="python")
if parse_dates_col not in df.columns:
    raise ValueError(f"Column '{parse_dates_col}' not found in CSV.")
df[parse_dates_col] = pd.to_datetime(df[parse_dates_col])
df.columns = [c.strip() for c in df.columns]
req = {"True", "Predicted"}
if not req.issubset(df.columns):
    raise ValueError(f"CSV must contain columns: {sorted(list(req))}")
df = df.sort_values(parse_dates_col, ignore_index=True)

# Mask for "Residual NaN True was 0 (held-out on TEST)"
if "was_missing" in df.columns:
    mask_nan = df["was_missing"].astype(str).str.lower().map({"true": True,
↪"false": False}).fillna(False)
    else:
        mask_nan = (df["True"] == 0)

# Residuals (recompute if missing)
if "Residual" not in df.columns or df["Residual"].isna().all():
    df["Residual"] = np.where(~mask_nan, df["Predicted"] - df["True"], np.
↪nan)

# ---- resolve metrics (prefer args, else CSV, else compute) ----
def _first_numeric(col):
    if col in df.columns:
        s = pd.to_numeric(df[col], errors="coerce").dropna()
        if not s.empty:
            return float(s.iloc[0])
    return None

mae_val = float(mae) if mae is not None else (_first_numeric("MAE"))

```

```

rmse_val = float(rmse) if rmse is not None else (_first_numeric("RMSE"))

y_true = df.loc[~mask_nan, "True"].to_numpy()
y_pred = df.loc[~mask_nan, "Predicted"].to_numpy()
if mae_val is None:
    mae_val = float(np.mean(np.abs(y_pred - y_true))) if y_true.size else
float("nan")
if rmse_val is None:
    rmse_val = float(sqrt(np.mean((y_pred - y_true) ** 2))) if y_true.size
else float("nan")

# ---- metrics box helper (uses your per-metric formats) ----
def _fmt(x, fmt):
    try:
        return fmt.format(float(x)) if fmt is not None else f"{float(x):g}"
    except Exception:
        return str(x)

def _metrics_box(ax):
    txt = f"MAE: {_fmt(mae_val, metrics_fmt_mae)}\nRMSE: {_fmt(rmse_val,
metrics_fmt_rmse)}"
    at = AnchoredText(txt, loc=metrics_loc, prop=dict(size=17),
frameon=True, borderpad=0.8)
    at.patch.set_alpha(0.75)
    ax.add_artist(at)

# ---- axis pads ----
date_pad = pd.Timedelta(days=int(date_pad_days))
x_min = df[parse_dates_col].min() - date_pad
x_max = df[parse_dates_col].max() + date_pad

# ----- Plot 1: True vs Predicted over time -----
fig, ax = plt.subplots(figsize=(11, 5))
ax.plot(df[parse_dates_col], df["True"], marker="o", linewidth=1.5,
label="True")
ax.plot(df[parse_dates_col], df["Predicted"], marker="s", linewidth=1.5,
label="Predicted")
if mask_nan.any():
    ax.scatter(df.loc[mask_nan, parse_dates_col], df.loc[mask_nan, "True"],
marker="o", s=90, color="red", zorder=5, label="True=0
(Residual NaN)")
    ax.scatter(df.loc[mask_nan, parse_dates_col], df.loc[mask_nan,
"Predicted"],
marker="s", s=90, color="red", zorder=5, label="Predicted at
True=0")
ax.set_xlim(x_min, x_max)

```

```

ax.margins(x=0.0, y=0.06)
ax.set_title(f"{title}: True vs Predicted")
ax.set_xlabel("Date")
ax.set_ylabel("Count")
ax.xaxis.set_major_locator(mdates.MonthLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter("%b"))
ax.grid(True, alpha=0.3)
_metrics_box(ax)
_legend_dedup(ax, loc=time_legend_loc)
plt.tight_layout()

# ----- Plot 2: Residuals over time -----
fig, ax = plt.subplots(figsize=(11, 4))
ax.axhline(0, linestyle="--", linewidth=1)
ax.plot(df.loc[~mask_nan, parse_dates_col], df.loc[~mask_nan, "Residual"],
↪marker="o", linewidth=1.5)
ax.set_xlim(x_min, x_max)
ax.margins(x=0.0, y=0.08)
ax.set_title(f"{title}: Residuals Over Time (Predicted - True)")
ax.set_xlabel("Date")
ax.set_ylabel("Residual")
ax.xaxis.set_major_locator(mdates.MonthLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter("%b"))
ax.grid(True, alpha=0.3)
_metrics_box(ax)
_legend_dedup(ax, loc=other_legend_loc)
plt.tight_layout()

# ----- Plot 3: Residual distribution -----
fig, ax = plt.subplots(figsize=(7, 4))
ax.hist(df.loc[~mask_nan, "Residual"].dropna(), bins="auto")
ax.set_title(f"{title}: Residual Distribution (Predicted - True)")
ax.set_xlabel("Residual")
ax.set_ylabel("Frequency")
ax.grid(True, alpha=0.3)
_metrics_box(ax)
_legend_dedup(ax, loc=other_legend_loc)
plt.tight_layout()

# ----- Plot 4: True vs Predicted (scatter) -----
fig, ax = plt.subplots(figsize=(6.8, 6.8))
x_true = df["True"].to_numpy()
y_pred = df["Predicted"].to_numpy()
ax.scatter(x_true[~mask_nan], y_pred[~mask_nan], marker="o", alpha=0.9,
↪label="Observed")
if mask_nan.any():

```

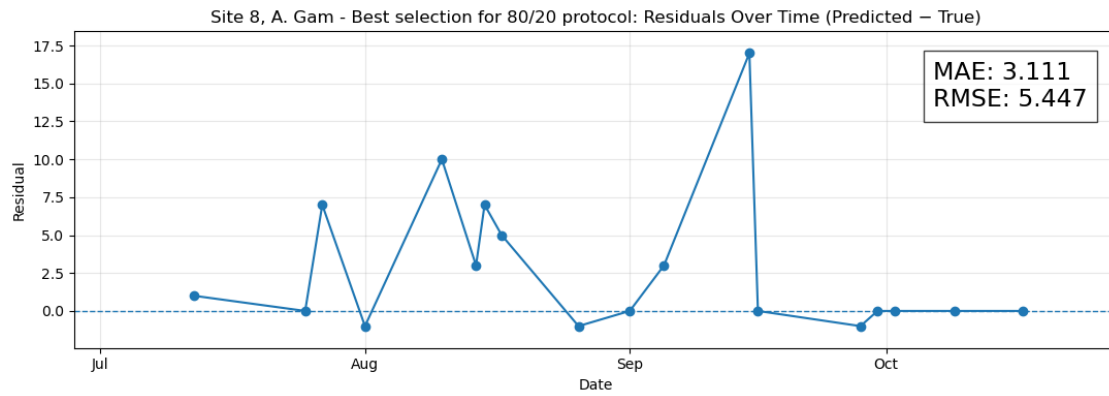
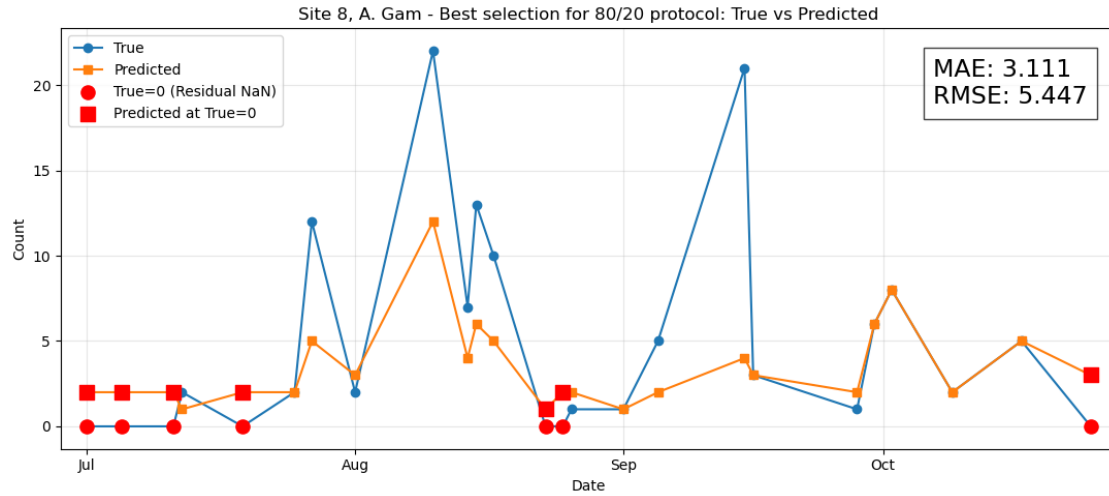
```

        ax.scatter(x_true[mask_nan], y_pred[mask_nan], marker="o", s=90,
↪color="red", zorder=5,
                    label="NaN Residual (True=0)")
    data_min = float(min(x_true.min(), y_pred.min()))
    data_max = float(max(x_true.max(), y_pred.max()))
    span = max(1e-9, data_max - data_min)
    pad_val = max(float(scatter_min_pad), float(scatter_pad_frac) * span)
    lims = (data_min - pad_val, data_max + pad_val)
    ax.plot(lims, lims, linestyle="--", linewidth=1)
    ax.set_xlim(lims)
    ax.set_ylim(lims)
    ax.set_xlabel("True")
    ax.set_ylabel("Predicted")
    ax.set_title(f"{title}: True vs Predicted (Scatter)")
    ax.grid(True, alpha=0.3)
    _metrics_box(ax)
    _legend_dedup(ax, loc=other_legend_loc)
    plt.tight_layout()

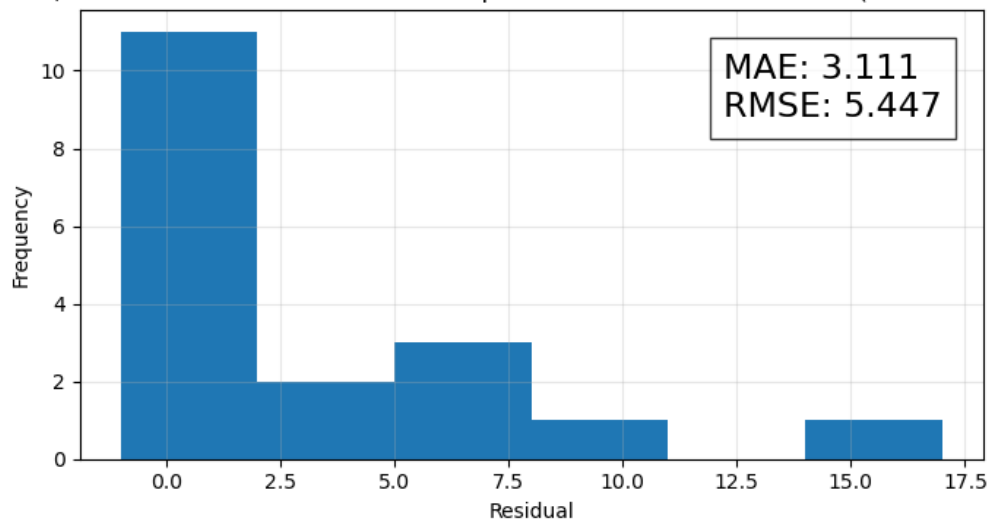
    plt.show()

# ----- Example usage -----
if __name__ == "__main__":
    # Example: 1 decimal for both MAE and RMSE (e.g., 6.2 and 9.4)
    plot_test_series_csv(
        "/Users/pradumchauhan/Desktop/MPI/Final_Data/Code/Pipeline_MICE/
↪pipeline_80_20_Tuned/pipeline_80_20_2015_S8_Tuned /results/
↪8020__both_lags__train_keeps_zeros__2015-07-01_2015-10-31__tuned__Adults_8_Gam0__preds__202
↪csv",
        title="Site 8, A. Gam - Best selection for 80/20 protocol",
        mae=3.111 ,
        rmse= 5.447,
        metrics_fmt_mae="{:.3f}",
        metrics_fmt_rmse="{:.3f}",
    )

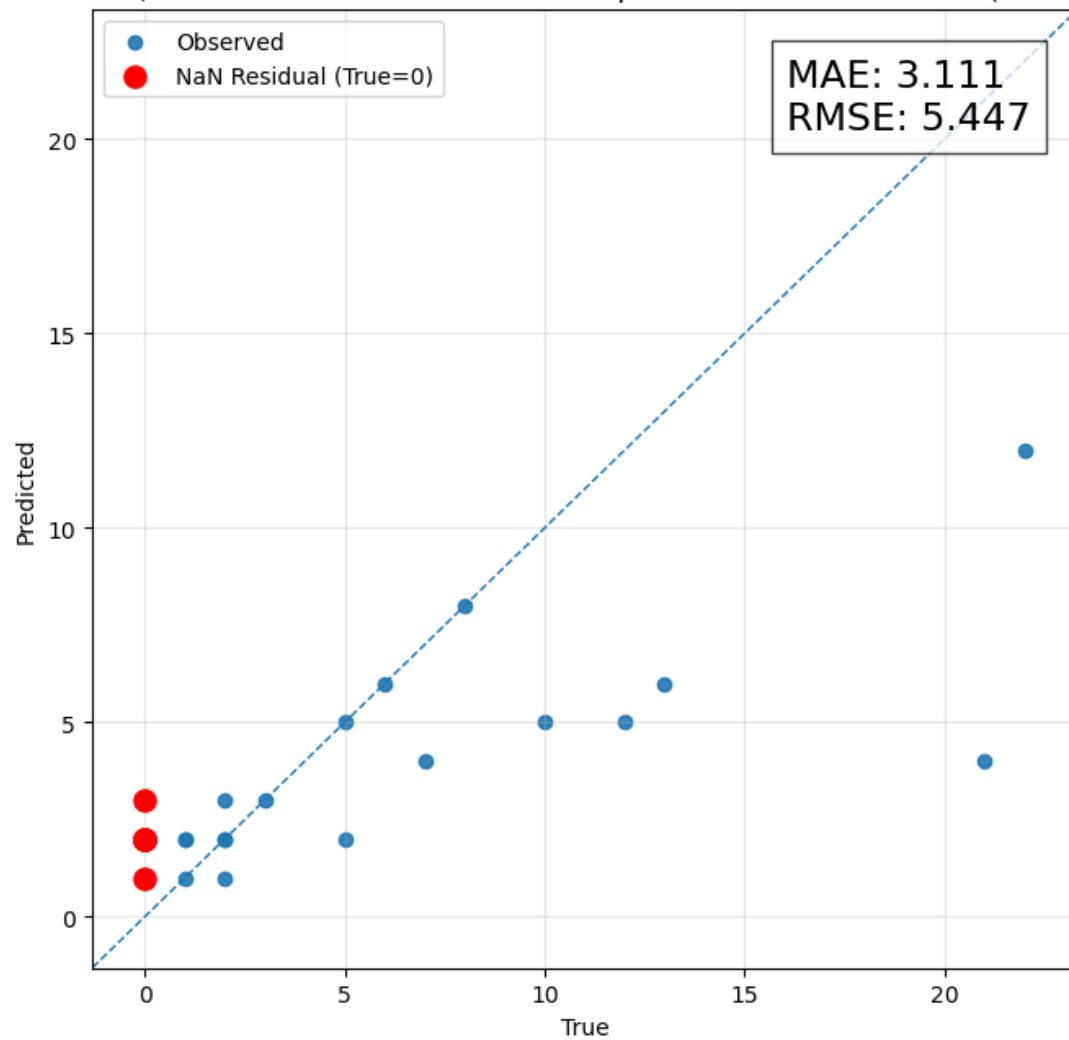
```



Site 8, A. Gam - Best selection for 80/20 protocol: Residual Distribution (Predicted – True)



Site 8, A. Gam - Best selection for 80/20 protocol: True vs Predicted (Scatter)



[ ]: