baseline_dignosis

December 11, 2025

[1]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in /opt/conda/lib/python3.11/site-
packages (3.1.2)
Requirement already satisfied: numpy in /opt/conda/lib/python3.11/site-packages
(from xgboost) (1.24.4)
Requirement already satisfied: nvidia-nccl-cu12 in
/opt/conda/lib/python3.11/site-packages (from xgboost) (2.28.9)
Requirement already satisfied: scipy in /opt/conda/lib/python3.11/site-packages
(from xgboost) (1.11.3)
```

[2]:
```
!pip install xlsxwriter
```

```
Requirement already satisfied: xlsxwriter in /opt/conda/lib/python3.11/site-
packages (3.2.9)
```

[1]:
```
# -*- coding: utf-8 -*-
"""
Baseline 3-class diagnostic model (AD / Mixed / Non-AD)
- Label: Final_Diagnosis (already mapped at baseline)
- Features: AD-signature MRI percent volumes (ROI/Total*100) + composites +␣
 ↪laterality + comorbidities + age + sex
- CV: 5-fold grouped by BrcId
- Metrics: macro-F1, balanced accuracy, OVR ROC-AUC, confusion matrix,␣
 ↪classification report
- Explainability: XGBoost built-in TreeSHAP contributions (pred_contribs=True)
"""

import re, json, numpy as np, pandas as pd
from pathlib import Path

# ------------------ CONFIG ------------------
CSV_PATH = "Merge_data_with_Pre_Mini_Mental_Total.csv"
OUT_DIR  = Path("baseline3_out"); OUT_DIR.mkdir(parents=True, exist_ok=True)
RANDOM_SEED = 42

# ------------------ LOAD ------------------
df = pd.read_csv(CSV_PATH, low_memory=False)
```

```
print(f"[ ] Loaded dataset: {len(df)} rows")

# ------------------ LABEL: Final_Diagnosis → AD/Mixed/Non-AD ------------------
def _norm_label(s):
    if pd.isna(s): return ""
    s = str(s).strip()
    s = s.replace("’","'").replace("‘","'").replace("-","-").replace("-","-")
    s = re.sub(r"\s+"," ", s)
    return s.lower()

AD_WHITELIST = {
    "alzheimer's disease",
    "alzheimer's disease - early onset",
    "alzheimer's disease - late onset",
    "alzheimer's disease - unspecified",
}
MIXED_WHITELIST = {
    "mixed dementia",
}
NONAD_WHITELIST = {
    "mild cognitive impairment",
    "unspecified dementia",
    "vascular dementia, unspecified",
    "dementia in parkinson disease",
    "dementia in other specified diseases classified elsewhere",
    "other",
}
# --------  "    MMSE" --------

def keep_mmse_closest_to_scan(df_in: pd.DataFrame) -> pd.DataFrame:

    """

         ScanID   (BrcId, Scan_Date) ,

      MMSE



    """

    df1 = df_in.copy()

    #       MMSE

    mmse_date_cols = [c for c in df1.columns if "mmse" in c.lower() and "date"␣
 ↪in c.lower()]
```

```
    scan_date_cols = [c for c in df1.columns if "scan" in c.lower() and "date" in c.lower()]

    if not scan_date_cols or not mmse_date_cols:

        #

        print("[warn] no explicit MMSE / Scan date columns found; skip 'closest to scan' filtering.")

        return df1

    scan_col = scan_date_cols[0]

    mmse_col = mmse_date_cols[0]

    df1[scan_col] = pd.to_datetime(df1[scan_col], errors="coerce")

    df1[mmse_col] = pd.to_datetime(df1[mmse_col], errors="coerce")

    #

    mask = df1[scan_col].notna() & df1[mmse_col].notna()

    df1["_mmse_dt_diff"] = np.where(

        mask,

        (df1[mmse_col] - df1[scan_col]).abs().dt.days,

        np.inf,

    )

    #   key   ScanID   (BrcId, Scan_Date)

    if "ScanID" in df1.columns and df1["ScanID"].notna().any():

        key_cols = ["ScanID"]

    elif "BrcId" in df1.columns:

        key_cols = ["BrcId", scan_col]

    else:
```

```
        print("[warn] no ScanID/BrcId columns; skip 'closest to scan' filtering.
 ↪")

        df1.drop(columns="_mmse_dt_diff", inplace=True)

        return df1

    df1 = (

        df1.sort_values(key_cols + ["_mmse_dt_diff"])

           .drop_duplicates(subset=key_cols, keep="first")

           .drop(columns="_mmse_dt_diff")

    )

    print(f"[ ] Kept MMSE closest-to-scan per {key_cols}: {len(df1)} rows")

    return df1

def map_diag3_exact(raw_label):
    s = _norm_label(raw_label)
    if s in MIXED_WHITELIST: return "Mixed"
    if s in AD_WHITELIST:     return "AD"
    if s in NONAD_WHITELIST:  return "Non-AD"
    # conservative fallback
    if re.search(r"\balzheimer", s): return "AD"
    if "mixed" in s:                 return "Mixed"
    return "Non-AD"

assert "Final_Diagnosis" in df.columns, "  Final_Diagnosis  "
df["target_diag"] = df["Final_Diagnosis"].apply(map_diag3_exact)

#
keep = df["target_diag"].isin(["AD","Mixed","Non-AD"])
df = df.loc[keep].reset_index(drop=True)
# ---          MRI  MMSE    ---

def dedup_per_scan(df_in: pd.DataFrame) -> pd.DataFrame:

    df1 = df_in.copy()

    #     "  "

    prefer_cols = [
```

```
        "Pre_Mini_Mental_Total","Post_Mini_Mental_Total",
        "ACE_date_pre","ACE_date_post",
        "HONOS_total_score_pre","HONOS_total_score_post"
    ]
    prefer_cols = [c for c in prefer_cols if c in df1.columns]
    comp = df1[prefer_cols].notna().sum(axis=1) if prefer_cols else pd.
↪Series(0, index=df1.index)
    #    Scan_Date    datetime   "   "
    if "Scan_Date" in df1.columns:
        df1["Scan_Date"] = pd.to_datetime(df1["Scan_Date"], errors="coerce")
    #    A   ScanID
    if "ScanID" in df1.columns and df1["ScanID"].notna().any():
        df1 = (df1.assign(_comp=comp)
                  #    ScanID     "   "
                  .sort_values(["ScanID", "_comp", "Scan_Date"],␣
↪ascending=[True, False, True])
                  .drop_duplicates(subset=["ScanID"], keep="first")
                  .drop(columns=["_comp"]))
        return df1
    #    B   ScanID   (BrcId, Scan_Date)     "   "
    keys = [c for c in ["BrcId","Scan_Date"] if c in df1.columns]
    if keys:
        #
        df1 = (df1.assign(_comp=comp)
                  .sort_values(keys + ["_comp"], ascending=[True, True, False])
```

```
                .drop_duplicates(subset=keys, keep="first")
                .drop(columns=["_comp"]))
        return df1
    #   C      BrcId
    if "BrcId" in df1.columns:
        df1 = (df1.assign(_comp=comp)
                .sort_values(["BrcId","Scan_Date","_comp"], ascending=[True,␣
 ↪True, False])
                .drop_duplicates(subset=["BrcId"], keep="first")
                .drop(columns=["_comp"]))
        return df1
    #
    return df1
df = keep_mmse_closest_to_scan(df)
df = dedup_per_scan(df)

print("Rows after per-scan de-dup:", len(df))

# ------------------ MRI percent features (ROI/Total*100) ------------------
if "Total" not in df.columns:
    raise KeyError("  Total     MRI    ")
print("[i] target label distribution:\n", df["target_diag"].
 ↪value_counts(dropna=False))
```

```
[ ] Loaded dataset: 16854 rows
[ ] Kept MMSE closest-to-scan per ['ScanID']: 3310 rows
Rows after per-scan de-dup: 3310
[i] target label distribution:
 target_diag
AD        1992
Non-AD     673
Mixed      645
Name: count, dtype: int64
```

[2]:
```
#  ROI    Brain_*
roi_code_groups = {
    # Medial temporal
    "Hipp": ["048_Right Hippocampus","049_Left Hippocampus"],
    "Ent":  ["117_Right Ent entorhinal area","118_Left Ent entorhinal␣
 ↪area","117_Right Ent","118_Left Ent"],
    "PHG":  ["171_Right PHG parahippocampal gyrus","172_Left PHG␣
 ↪parahippocampal gyrus","171_Right PHG","172_Left PHG"],
    # Lateral/basal temporal
    "MTG":  ["155_Right MTG middle temporal gyrus","156_Left MTG middle␣
 ↪temporal gyrus","155_Right MTG","156_Left MTG"],
    "ITG":  ["133_Right ITG inferior temporal gyrus","134_Left ITG inferior␣
 ↪temporal gyrus","133_Right ITG","134_Left ITG"],
    "STG":  ["201_Right STG superior temporal gyrus","202_Left STG superior␣
 ↪temporal gyrus","201_Right STG","202_Left STG"],
    "FuG":  ["123_Right FuG fusiform gyrus","124_Left FuG fusiform␣
 ↪gyrus","123_Right FuG","124_Left FuG"],
    "TMP":  ["203_Right TMP temporal pole","204_Left TMP temporal␣
 ↪pole","203_Right TMP","204_Left TMP"],
    # Medial parietal
    "PCgG": ["167_Right PCgG posterior cingulate gyrus","168_Left PCgG␣
 ↪posterior cingulate gyrus","167_Right PCgG","168_Left PCgG"],
    "PCu":  ["169_Right PCu precuneus","170_Left PCu precuneus","169_Right␣
 ↪PCu","170_Left PCu"],
    # Posterior parietal/occipital (PCA axis)
    "SPL":  ["199_Right SPL superior parietal lobule","200_Left SPL superior␣
 ↪parietal lobule","199_Right SPL","200_Left SPL"],
    "SOG":  ["197_Right SOG superior occipital gyrus","198_Left SOG superior␣
 ↪occipital gyrus","197_Right SOG","198_Left SOG"],
    "MOG":  ["145_Right MOG middle occipital gyrus","146_Left MOG middle␣
 ↪occipital gyrus","145_Right MOG","146_Left MOG"],
    "IOG":  ["129_Right IOG inferior occipital gyrus","130_Left IOG inferior␣
 ↪occipital gyrus","129_Right IOG","130_Left IOG"],
    # Ventricles
    "LatVent": ["052_Right Lateral Ventricle","053_Left Lateral Ventricle"],
    "InfLat":  ["050_Right Inf Lat Vent","051_Left Inf Lat Vent"],
}

def pct_col(col):
    if col not in df.columns: return None
    v = pd.to_numeric(df[col], errors="coerce")
    total = pd.to_numeric(df["Total"], errors="coerce").replace(0, np.nan)
    return (v/total)*100.0

mri_cols = []
mri_pct_frames = []
```

```
#  ROI
for grp, cols in roi_code_groups.items():
    for c in cols:
        s = pct_col(c)
        if s is not None:
            new = f"{c}__pct"
            mri_cols.append(new); mri_pct_frames.append(s.rename(new))

#   Brain_*
brain_cols = [c for c in df.columns if re.match(r"^Brain_", str(c))]
for c in brain_cols:
    s = pct_col(c)
    if s is not None:
        new = f"{c}__pct"
        mri_cols.append(new); mri_pct_frames.append(s.rename(new))

X_mri = pd.concat(mri_pct_frames, axis=1) if mri_pct_frames else pd.
 ↪DataFrame(index=df.index)
print(f"[i] MRI percent features: {X_mri.shape[1]}")

# ------------------ composites ------------------
def sum_cols(names, outname):
    s = pd.Series(0.0, index=df.index)
    for n in names:
        if n in X_mri.columns: s = s.add(X_mri[n].fillna(0), fill_value=0)
    return s.rename(outname)

hipp_cols = [c for c in X_mri.columns if "Hippocampus__pct" in c]
ent_cols  = [c for c in X_mri.columns if (" Ent" in c or "Ent " in c) and␣
 ↪"__pct" in c]
phg_cols  = [c for c in X_mri.columns if "PHG" in c and "__pct" in c]
lat_temp_cols = [c for c in X_mri.columns if any(k in c for k in [" MTG "," ITG␣
 ↪"," STG "," FuG "," TMP "]) and "__pct" in c]
mp_cols   = [c for c in X_mri.columns if any(k in c for k in ["PCgG","PCu"])␣
 ↪and "__pct" in c]
post_cols = [c for c in X_mri.columns if any(k in c for k in␣
 ↪["SPL","SOG","MOG","IOG"]) and "__pct" in c]
vent_cols = [c for c in X_mri.columns if any(k in c for k in ["Lateral␣
 ↪Ventricle__pct","Inf Lat Vent__pct"])]

comp_frames = [
    sum_cols(hipp_cols+ent_cols+phg_cols, "MTL_total_pct"),
    sum_cols(lat_temp_cols, "Temporal_lateral_total_pct"),
    sum_cols(mp_cols, "Medial_parietal_total_pct"),
    sum_cols(post_cols, "Posterior_total_pct"),
```

```
    sum_cols(vent_cols, "Ventricles_total_pct"),
]
X_comp = pd.concat(comp_frames, axis=1)

# ------------------ laterality indices ------------------
def laterality_pair(L_contains, R_contains, outname):
    Lc = [c for c in X_mri.columns if L_contains in c]
    Rc = [c for c in X_mri.columns if R_contains in c]
    if not Lc or not Rc: return None
    L = X_mri[Lc[0]]; R = X_mri[Rc[0]]
    denom = (L+R).replace(0, np.nan)
    return ((L-R)/denom).rename(outname)

li_frames = []
for l, r, out in [
    ("168_Left PCgG", "167_Right PCgG", "LI_PCgG"),
    ("170_Left PCu",  "169_Right PCu",  "LI_PCu"),
    ("156_Left MTG",  "155_Right MTG",  "LI_MTG"),
    ("134_Left ITG",  "133_Right ITG",  "LI_ITG"),
    ("202_Left STG",  "201_Right STG",  "LI_STG"),
]:
    s = laterality_pair(l, r, out)
    if s is not None: li_frames.append(s)

X_li = pd.concat(li_frames, axis=1) if li_frames else pd.DataFrame(index=df.
 ↪index)

#   MRI
X_mri_all = pd.concat([X_mri, X_comp, X_li], axis=1)

# ------------------ comorbidities ------------------
comorb_cols = [
   ␣
 ↪"asthma","falls","hypertension","Cerebrovascular_accident","Epilepsy","Diabetes_mellitus",
   ␣
 ↪"Chronic_kidney_disease","Psoriasis","Parkinsons_disease","Multiple_sclerosis","Eczema",
   ␣
 ↪"Hypertensive_disorder","Transient_ischemic_attack","Migraine","Chronic_obstructive_lung_di
   ␣
 ↪"Arthritis","Heart_failure","Asthma","Ischemic_heart_disease","Inflammatory_bowel_disease",
   ␣
 ↪"Atrial_fibrillation","Chronic_liver_disease","Chronic_sinusitis","Coronary_arteriosclerosi
]
comorb_cols = [c for c in comorb_cols if c in df.columns]
X_comorb = df[comorb_cols].apply(pd.to_numeric, errors="coerce").fillna(0).
 ↪clip(0,1)
```

```
# ------------------ age & sex ------------------
def make_age_series(d):
    if "age_at_scan_date" in d.columns:
        s = pd.to_numeric(d["age_at_scan_date"], errors="coerce")
        if s.notna().mean()>0.5: return s
    if "age_years" in d.columns:
        s = pd.to_numeric(d["age_years"], errors="coerce")
        if s.notna().mean()>0.5: return s
    sdt = pd.to_datetime(d.get("Scan_Date"), errors="coerce")
    dob = pd.to_datetime(d.get("cleaneddateofbirth"), errors="coerce")
    if sdt.notna().any() and dob.notna().any():
        return (sdt - dob).dt.days/365.25
    return pd.Series(np.nan, index=d.index)

age = make_age_series(df).rename("age")
sex = df.get("Gender_ID", pd.Series(np.nan, index=df.index)).astype(str)
# ------------------ MMSE  baseline    ------------------
if "Pre_Mini_Mental_Total" in df.columns:
    mmse = pd.to_numeric(df["Pre_Mini_Mental_Total"], errors="coerce").
 ↪rename("MMSE")
else:
    #
    mmse = pd.Series(np.nan, index=df.index, name="MMSE")
    print("[warn] Column 'Pre_Mini_Mental_Total' not found, MMSE is all NaN.")

# ------------------ final X, y ------------------
X_num = pd.concat([X_mri_all, X_comorb, age, mmse], axis=1)
X_cat = pd.DataFrame({"Gender_ID": sex})
X_all = pd.concat([X_num, X_cat], axis=1)

y = df["target_diag"].astype(str)
groups = df.get("BrcId", pd.Series(np.arange(len(df)))).astype(str)

print(f"[i] X_num={X_num.shape}, X_cat={X_cat.shape}; y={y.value_counts().
 ↪to_dict()}")
def classify_feature(col: str) -> str:

    if col in X_mri_all.columns:

        return "MRI_pct"

    if 'WMH' in col.upper():

        return "WMH"

    if col in X_comorb.columns:
```

```
        return "comorbidity"

    if col == "age":

        return "age"

    if col == "Gender_ID":

        return "sex"

    # MMSE  Mini-Mental

    if "MMSE" in col.upper() or "MINI_MENTAL" in col.upper():

        return "MMSE"

    return "other"


feat_stage1 = pd.DataFrame({
    "feature_name": X_all.columns,
    "block": [classify_feature(c) for c in X_all.columns],
})
feat_stage1.to_csv(OUT_DIR/"feature_list_stage1.csv", index=False)
print("[ ] Saved Stage-1 feature list →", OUT_DIR/"feature_list_stage1.csv")
```

```
[i] MRI percent features: 65
[i] X_num=(3310, 101), X_cat=(3310, 1); y={'AD': 1992, 'Non-AD': 673, 'Mixed':
645}
[ ] Saved Stage-1 feature list → baseline3_out/feature_list_stage1.csv
```

[3]:
```
# ------------------ preprocessing ------------------
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

num_cols = X_num.columns.tolist()
cat_cols = ["Gender_ID"]

try:
    ohe = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
except TypeError:  # sklearn<1.2
    from sklearn.preprocessing import OneHotEncoder as OHE
    ohe = OHE(handle_unknown="ignore", sparse=False)
```

```
preprocess = ColumnTransformer(
    transformers=[
        ("num", Pipeline([("imp", SimpleImputer(strategy="median")),
                          ("sc",  StandardScaler())]), num_cols),
        ("cat", Pipeline([("imp", SimpleImputer(strategy="most_frequent")),
                          ("oh",  ohe)]), cat_cols),
    ],
    remainder="drop",
)
```

[4]:
```
# ------------------ CV splits (grouped by BrcId) ------------------
from sklearn.model_selection import StratifiedKFold
try:
    from sklearn.model_selection import StratifiedGroupKFold
    cv = StratifiedGroupKFold(n_splits=5, shuffle=True,␣
 ↪random_state=RANDOM_SEED)
    splits = list(cv.split(X_all, y, groups=groups))
except Exception:
    # approximate grouped stratification
    g = df.groupby("BrcId")["target_diag"].first().reset_index()
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_SEED)
    splits = []
    for tr, te in skf.split(g["BrcId"], g["target_diag"]):
        tr_mask = groups.isin(g.loc[tr,"BrcId"])
        te_mask = groups.isin(g.loc[te,"BrcId"])
        splits.append((X_all.index[tr_mask], X_all.index[te_mask]))
```

[5]:
```
# ------------------ model: XGB classifier (with integer label encoding)␣
 ↪------------------

import xgboost as xgb

from sklearn.metrics import classification_report, f1_score,␣
 ↪balanced_accuracy_score, roc_auc_score, confusion_matrix

#        /

labels = ["AD", "Mixed", "Non-AD"]

lab2int = {lab:i for i, lab in enumerate(labels)}

int2lab = {i:lab for lab,i in lab2int.items()}

# y →
```

```
y_int = y.map(lab2int).astype(int)

#   BrcId     CV

from sklearn.model_selection import StratifiedKFold

try:

    from sklearn.model_selection import StratifiedGroupKFold

    cv = StratifiedGroupKFold(n_splits=5, shuffle=True,␣
 ↪random_state=RANDOM_SEED)

    splits = list(cv.split(X_all, y_int, groups=groups))

except Exception:

    #

    g = df.groupby("BrcId")["target_diag"].first().reset_index()

    g_int = g["target_diag"].map(lab2int).astype(int)

    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_SEED)

    splits = []

    for tr, te in skf.split(g["BrcId"], g_int):

        tr_mask = groups.isin(g.loc[tr,"BrcId"])

        te_mask = groups.isin(g.loc[te,"BrcId"])

        splits.append((X_all.index[tr_mask], X_all.index[te_mask]))

# XGB

xgb_clf = xgb.XGBClassifier(

    objective="multi:softprob",

    num_class=3,

    max_depth=3, learning_rate=0.05, n_estimators=800,

    subsample=0.8, colsample_bytree=0.8,
```

```
    reg_lambda=1.0, random_state=RANDOM_SEED, n_jobs=-1,
    eval_metric="mlogloss"
)
pipe = Pipeline([("prep", preprocess), ("clf", xgb_clf)])
#    sample_weight           y_int
class_counts = y.value_counts()
class_weight = {k: class_counts.max()/v for k,v in class_counts.items()}
sample_weight = y.map(class_weight).values  #   y_int
y_true_all_str, y_pred_all_str = [], []
y_prob_all = []
for fold, (tr_idx, te_idx) in enumerate(splits):
    pipe.fit(X_all.iloc[tr_idx], y_int.iloc[tr_idx],␣
 ↪clf__sample_weight=sample_weight[tr_idx])
    y_pred_int = pipe.predict(X_all.iloc[te_idx])              #
    y_pred_str = pd.Series(y_pred_int).map(int2lab).values     #
    y_true_str = y.iloc[te_idx].values                         #
    y_prob = pipe.predict_proba(X_all.iloc[te_idx])            #   (n, 3)   ␣
 ↪0,1,2 → AD, Mixed, Non-AD
    y_true_all_str.extend(y_true_str)
    y_pred_all_str.extend(y_pred_str)
    y_prob_all.append(y_prob)
y_true_all_str = np.array(y_true_all_str)
y_pred_all_str = np.array(y_pred_all_str)
y_prob_all = np.vstack(y_prob_all)
```

```
#    &

print("\n=== Classification report (pooled folds) ===")

print(classification_report(y_true_all_str, y_pred_all_str, labels=labels,␣
 ↪digits=3))

macro_f1 = f1_score(y_true_all_str, y_pred_all_str, average="macro")

bacc     = balanced_accuracy_score(y_true_all_str, y_pred_all_str)

print("Macro-F1:", round(macro_f1, 3))

print("Balanced Acc:", round(bacc, 3))

# OVR ROC-AUC   labels

try:

    y_true_bin = pd.get_dummies(pd.Categorical(y_true_all_str,␣
 ↪categories=labels))

    ovr_auc = roc_auc_score(y_true_bin.values, y_prob_all, average="macro",␣
 ↪multi_class="ovr")

    print("OVR ROC-AUC:", round(ovr_auc, 3))

except Exception as e:

    print("OVR ROC-AUC skipped:", e)

#

cm = confusion_matrix(y_true_all_str, y_pred_all_str, labels=labels)

cm_df = pd.DataFrame(cm, index=[f"true_{l}" for l in labels],␣
 ↪columns=[f"pred_{l}" for l in labels])

cm_df.to_csv(OUT_DIR/"confusion_matrix.csv", index=True)

#

metrics = pd.DataFrame([{

    "macro_f1": float(macro_f1),

    "balanced_acc": float(bacc)
```

```
}])

metrics.to_csv(OUT_DIR/"cv_metrics_summary_xgb.csv", index=False)

print("\nSaved confusion matrix and metrics to:", OUT_DIR.resolve())
```

```
=== Classification report (pooled folds) ===
              precision    recall  f1-score   support

          AD      0.693     0.692     0.693      1992
       Mixed      0.348     0.319     0.333       645
      Non-AD      0.531     0.575     0.552       673

    accuracy                          0.596      3310
   macro avg      0.524     0.529     0.526      3310
weighted avg      0.593     0.596     0.594      3310

Macro-F1: 0.526
Balanced Acc: 0.529
OVR ROC-AUC: 0.721

Saved confusion matrix and metrics to: /home/jovyan/baseline3_out
```

[6]:
```
print("Unique target labels:", sorted(y.unique()))

#     '-AD'

y = y.replace({"-AD":"AD", " AD":"AD", "Non AD":"Non-AD"})
```

```
Unique target labels: ['AD', 'Mixed', 'Non-AD']
```

[7]:
```
# ------------------ fit on full data for built-in TreeSHAP contributions␣
  ↪(robust for 2D/3D) ------------------

import xgboost as xgb

import numpy as np

import pandas as pd

import re

from pathlib import Path
```

```
# 1)
pipe.fit(X_all, y_int, clf__sample_weight=sample_weight)
prep  = pipe.named_steps["prep"]
model = pipe.named_steps["clf"]
Xmat = prep.transform(X_all)
if hasattr(Xmat, "toarray"):
    Xmat = Xmat.toarray()
n_features = Xmat.shape[1]
# 2) XGBoost built-in TreeSHAP contributions
dmat = xgb.DMatrix(Xmat)
contrib = model.get_booster().predict(dmat, pred_contribs=True)
labels = ["AD","Mixed","Non-AD"]
C_expected = len(labels)
# 3)
#    A) (n, (p+1)*C)
#    B) (n, p+1, C)
#    C) (n, C, p+1)
if contrib.ndim == 2:
    n, pc = contrib.shape
    #   p+1 & C
    if pc % C_expected == 0:
        p_plus1 = pc // C_expected
        arr = contrib.reshape(n, C_expected, p_plus1)     # (n, C, p+1)
```

```
        phi = arr[:, :, :p_plus1-1]                          #   bias → (n, C, p)
        mean_abs = np.abs(phi).mean(axis=(0, 1))             # (p,)
    else:
        #     n_features   p_plus1
        p_plus1 = n_features + 1
        C = pc // p_plus1
        arr = contrib.reshape(n, C, p_plus1)
        phi = arr[:, :, :p_plus1-1]
        mean_abs = np.abs(phi).mean(axis=(0, 1))
elif contrib.ndim == 3:
    n, a, b = contrib.shape
    #   B: (n, p+1, C)
    if b == C_expected:
        p_plus1, C = a, b
        p_keep = min(n_features, p_plus1 - 1)
        phi = contrib[:, :p_keep, :]                         # (n, p_keep, C)
        mean_abs = np.abs(phi).mean(axis=(0, 2))             # (p_keep,)
    #   C: (n, C, p+1)
    elif a == C_expected:
        C, p_plus1 = a, b
        p_keep = min(n_features, p_plus1 - 1)
        phi = contrib[:, :, :p_keep]                         # (n, C, p_keep)
        mean_abs = np.abs(phi).mean(axis=(0, 1))             # (p_keep,)
    else:
```

```
        #      →    n_features
        p_plus1 = b
        p_keep = min(n_features, p_plus1 - 1)
        phi = contrib[:, :, :p_keep]
        mean_abs = np.abs(phi).mean(axis=(0, 1))
else:
    raise ValueError(f"Unexpected pred_contribs ndim={contrib.ndim}")
# 4)
try:
    feat_names = prep.get_feature_names_out()
    feat_names = [re.sub(r"^(num|cat)__","", f) for f in feat_names]
except Exception:
    feat_names = [f"f{i}" for i in range(mean_abs.shape[0])]
if len(feat_names) != len(mean_abs):
    k = min(len(feat_names), len(mean_abs))
    feat_names = feat_names[:k]
    mean_abs   = mean_abs[:k]
imp_df = pd.DataFrame({"feature": feat_names,
                       "mean_abs_contrib": mean_abs
                      }).sort_values("mean_abs_contrib", ascending=False)
imp_df.to_csv(OUT_DIR / "feature_importance_xgb_contribs.csv", index=False)
print("Saved:", OUT_DIR / "feature_importance_xgb_contribs.csv")
print("Top 20 features:\n", imp_df.head(50).to_string(index=False))
```

```
Saved: baseline3_out/feature_importance_xgb_contribs.csv
```

Top 20 features:

feature	mean_abs_contrib
MMSE	0.221973
051_Left Inf Lat Vent__pct	0.173523
050_Right Inf Lat Vent__pct	0.155038
049_Left Hippocampus__pct	0.097018
Brain_Left_0_Frontal_0__pct	0.087328
age	0.083767
LI_MTG	0.080075
117_Right Ent entorhinal area__pct	0.078581
Brain_Left_0_Frontal_1__pct	0.073360
Cerebrovascular_accident	0.069932
Brain_Left_0_Frontal_99__pct	0.066800
Brain_Right_2_Parietal_99__pct	0.065687
Brain_Left_4_DeepNuclei_10__pct	0.062143
LI_ITG	0.061802
048_Right Hippocampus__pct	0.060998
124_Left FuG fusiform gyrus__pct	0.058376
200_Left SPL superior parietal lobule__pct	0.056807
118_Left Ent entorhinal area__pct	0.054667
falls	0.052513
Brain_Right_4_DeepNuclei_9__pct	0.052345
Brain_Right_4_DeepNuclei_99__pct	0.050890
052_Right Lateral Ventricle__pct	0.050052
171_Right PHG parahippocampal gyrus__pct	0.049731
198_Left SOG superior occipital gyrus__pct	0.049172
129_Right IOG inferior occipital gyrus__pct	0.049129
167_Right PCgG posterior cingulate gyrus__pct	0.048793
204_Left TMP temporal pole__pct	0.047621
LI_PCu	0.047296
199_Right SPL superior parietal lobule__pct	0.045774
Brain_Right_99_99__pct	0.045458
130_Left IOG inferior occipital gyrus__pct	0.044505
203_Right TMP temporal pole__pct	0.043074
133_Right ITG inferior temporal gyrus__pct	0.042924
Brain_Left_2_Parietal_99__pct	0.042364
Brain_Right_0_Frontal_99__pct	0.042310
134_Left ITG inferior temporal gyrus__pct	0.042220
LI_STG	0.041659
172_Left PHG parahippocampal gyrus__pct	0.040939
170_Left PCu precuneus__pct	0.040922
Brain_Left_3_Occipital_7__pct	0.040886
201_Right STG superior temporal gyrus__pct	0.039591
Brain_Left_0_Frontal_2__pct	0.039308
Parkinsons_disease	0.038678
Brain_Left_2_Parietal_6__pct	0.038052
053_Left Lateral Ventricle__pct	0.036623
145_Right MOG middle occipital gyrus__pct	0.035981

```
                  169_Right PCu precuneus__pct          0.034564
                Brain_Right_3_Occipital_7__pct          0.034561
                Brain_Left_4_DeepNuclei_9__pct          0.034169
                 Brain_Left_3_Occipital_8__pct          0.033788
```

[8]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /opt/conda/lib/python3.11/site-
packages (3.1.2)
Requirement already satisfied: numpy in /opt/conda/lib/python3.11/site-packages
(from xgboost) (1.24.4)
Requirement already satisfied: nvidia-nccl-cu12 in
/opt/conda/lib/python3.11/site-packages (from xgboost) (2.28.9)
Requirement already satisfied: scipy in /opt/conda/lib/python3.11/site-packages
(from xgboost) (1.11.3)
Note: you may need to restart the kernel to use updated packages.
```

[9]:
```
# ============================

# HIERARCHICAL 3-CLASS MODEL

# Stage-1: AD vs Others

# Stage-2: (Among non-AD) Mixed vs Non-AD using WMH + vascular features

# ============================

from pathlib import Path

import numpy as np, pandas as pd, re, json

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline

from sklearn.metrics import (classification_report, f1_score,

                             balanced_accuracy_score, roc_auc_score,␣
 ↪confusion_matrix)

from sklearn.model_selection import StratifiedKFold

import xgboost as xgb
```

```
HIER_DIR = Path("hier_out"); HIER_DIR.mkdir(parents=True, exist_ok=True)

RANDOM_SEED = 42

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

def make_ohe():

    try:

        return OneHotEncoder(handle_unknown="ignore", sparse_output=False)

    except TypeError:

        from sklearn.preprocessing import OneHotEncoder as OHE

        return OHE(handle_unknown="ignore", sparse=False)

def build_preprocess(num_cols, cat_cols):

    return ColumnTransformer(

        transformers=[

            ("num", Pipeline([("imp", SimpleImputer(strategy="median")),

                              ("sc",  StandardScaler())]), num_cols),

            ("cat", Pipeline([("imp", SimpleImputer(strategy="most_frequent")),

                              ("oh",  make_ohe())]), cat_cols),

        ],

        remainder="drop",

    )

def grouped_stratified_splits(df_index, y_series, groups_series, n_splits=5,␣
↪seed=42):

    """        StratifiedGroupKFold     """

    try:

        from sklearn.model_selection import StratifiedGroupKFold
```

```
        cv = StratifiedGroupKFold(n_splits=n_splits, shuffle=True,␣
 ↪random_state=seed)
        return list(cv.split(df_index, y_series, groups_series))
    except Exception:
        g = pd.DataFrame({"BrcId": groups_series, "y": y_series}).
 ↪groupby("BrcId").first().reset_index()
        skf = StratifiedKFold(n_splits=n_splits, shuffle=True,␣
 ↪random_state=seed)
        splits = []
        for tr, te in skf.split(g["BrcId"], g["y"]):
            tr_mask = groups_series.isin(g.loc[tr,"BrcId"])
            te_mask = groups_series.isin(g.loc[te,"BrcId"])
            splits.append((df_index[tr_mask], df_index[te_mask]))
        return splits
def metrics_binary(y_true, y_pred, y_prob):
    rep = classification_report(y_true, y_pred, digits=3)
    f1  = f1_score(y_true, y_pred, average="binary", pos_label=1)
    bacc= balanced_accuracy_score(y_true, y_pred)
    try:
        auc = roc_auc_score(y_true, y_prob[:,1])
    except Exception:
        auc = np.nan
    return rep, dict(f1=f1, balanced_acc=bacc, auc=auc)
# ----------    ----------
#       df     df["target_diag"]   {AD, Mixed, Non-AD}
```

```
assert set(df["target_diag"].unique()) <= {"AD","Mixed","Non-AD"}

groups = df.get("BrcId", pd.Series(np.arange(len(df)))).astype(str)

index_all = df.index

# ========== Stage-1: AD vs Others ==========

print("\n[Stage-1] AD vs Others")

# y1: 1=AD, 0=Others

y1 = (df["target_diag"] == "AD").astype(int)

#         X_all MRI% +   +   +  /          "   "

if 'X_all' in globals():

    X1 = X_all.copy()

else:

    #          __pct   age_at_scan_date/age_years  Gender_ID

    pct_cols = [c for c in df.columns if c.endswith("__pct")]

    num_extra = []

    if "age_at_scan_date" in df.columns:

        num_extra.append("age_at_scan_date")

    elif "age_years" in df.columns:

        num_extra.append("age_years")

    cat_cols = ["Gender_ID"] if "Gender_ID" in df.columns else []

    #

    comorb_cols = [c for c in df.columns if c in [

      ␣
↪"hypertension","Hypertensive_disorder","Diabetes_mellitus","Atrial_fibrillation",
```

```
      ␣
 ↪"Ischemic_heart_disease","Cerebrovascular_accident","Transient_ischemic_attack",
      ␣
 ↪"Heart_failure","Coronary_arteriosclerosis","falls","Chronic_obstructive_lung_disease"
    ]]
    X1 = pd.concat([df[pct_cols].apply(pd.to_numeric, errors="coerce"),
                    df[num_extra].apply(pd.to_numeric, errors="coerce"),
                    df[comorb_cols].apply(pd.to_numeric, errors="coerce"),
                    df[cat_cols]], axis=1)
    X1 = X1.fillna(0)
#   +
num_cols1 = X1.select_dtypes(include=[np.number]).columns.tolist()
cat_cols1 = [c for c in X1.columns if c not in num_cols1]
pre1 = build_preprocess(num_cols1, cat_cols1)
clf1 = xgb.XGBClassifier(
    objective="binary:logistic",
    max_depth=3, learning_rate=0.05, n_estimators=600,
    subsample=0.8, colsample_bytree=0.8,
    reg_lambda=1.0, random_state=RANDOM_SEED, n_jobs=-1,
    eval_metric="logloss"
)
pipe1 = Pipeline([("prep", pre1), ("clf", clf1)])
#    CV
splits1 = grouped_stratified_splits(index_all, y1, groups, n_splits=5,␣
 ↪seed=RANDOM_SEED)
```

```
y1_true, y1_pred, y1_prob = [], [], []

for tr_idx, te_idx in splits1:

    pipe1.fit(X1.iloc[tr_idx], y1.iloc[tr_idx])

    y1_pred.extend(pipe1.predict(X1.iloc[te_idx]))

    y1_prob.extend(pipe1.predict_proba(X1.iloc[te_idx]) )

y1_true = y1.values

y1_pred = np.array(y1_pred)

y1_prob = np.vstack(y1_prob)

rep1, m1 = metrics_binary(y1_true, y1_pred, y1_prob)

(Path(HIER_DIR/"stage1_report.txt")).write_text(rep1, encoding="utf-8")

with open(HIER_DIR/"stage1_metrics.json","w") as f: json.dump(m1, f, indent=2)

print(rep1)

print("[Stage-1] F1:", round(m1["f1"],3), "BalancedAcc:",␣
 ↪round(m1["balanced_acc"],3), "AUC:", round(m1["auc"],3))

# ========== Stage-2: Mixed vs Non-AD  AD  ==========

print("\n[Stage-2] Mixed vs Non-AD (among non-AD cases)")

nonad_mask = (df["target_diag"] != "AD")

df2 = df.loc[nonad_mask].reset_index(drop=True)

groups2 = df2.get("BrcId", pd.Series(np.arange(len(df2)))).astype(str)

# y2: 1=Mixed, 0=Non-AD

y2 = (df2["target_diag"] == "Mixed").astype(int)

# ---- Stage-2  WMH + vascular burden (+ age/sex  ) ----

# 1) WMH         'wmh'

wmh_cols = [c for c in df2.columns if re.search(r"wmh", c, flags=re.I)]
```

```
# intracranial  TIV/ICV/Total

icv_col = None

for cand in ["tiv","ICV","icv","Total","total"]:

    if cand in df2.columns:

        icv_col = cand

        break

X2_blocks = []

if wmh_cols:

    wmh = df2[wmh_cols].apply(pd.to_numeric, errors="coerce")

    if icv_col:

        icv = pd.to_numeric(df2[icv_col], errors="coerce").replace(0, np.nan)

        for c in wmh_cols:

            X2_blocks.append( (wmh[c]/icv).rename(f"{c}_pct_icv") )

    else:

        for c in wmh_cols:

            X2_blocks.append(wmh[c].rename(c))

else:

    print("[Stage-2] WMH columns not found; proceeding without WMH.")

# 2) Vascular comorbidities

vasc_cols = [

  ␣
 ↪"hypertension","Hypertensive_disorder","Diabetes_mellitus","Atrial_fibrillation",

  ␣
 ↪"Ischemic_heart_disease","Cerebrovascular_accident","Transient_ischemic_attack",

   "Heart_failure","Coronary_arteriosclerosis"
```

```
]

vasc_cols = [c for c in vasc_cols if c in df2.columns]

X2_blocks.append( df2[vasc_cols].apply(pd.to_numeric, errors="coerce").
 ↪fillna(0).clip(0,1) )

# 3) Age / Sex

if "age_at_scan_date" in df2.columns:

    X2_blocks.append( pd.to_numeric(df2["age_at_scan_date"], errors="coerce").
 ↪rename("age") )

elif "age_years" in df2.columns:

    X2_blocks.append( pd.to_numeric(df2["age_years"], errors="coerce").
 ↪rename("age") )

if "Gender_ID" in df2.columns:

    X2_blocks.append( df2["Gender_ID"].astype(str).rename("Gender_ID") )

X2 = pd.concat(X2_blocks, axis=1)

X2 = X2.fillna(0)

num_cols2 = X2.select_dtypes(include=[np.number]).columns.tolist()

cat_cols2 = [c for c in X2.columns if c not in num_cols2]

pre2 = build_preprocess(num_cols2, cat_cols2)

clf2 = xgb.XGBClassifier(

    objective="binary:logistic",

    max_depth=3, learning_rate=0.05, n_estimators=600,

    subsample=0.8, colsample_bytree=0.8,

    reg_lambda=1.0, random_state=RANDOM_SEED, n_jobs=-1,

    eval_metric="logloss"

)
```

```
pipe2 = Pipeline([("prep", pre2), ("clf", clf2)])

# ============================================================================

from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import make_scorer, balanced_accuracy_score

# ------------------------------------------------------------
# 1)   XGBoost + Pipeline
# ------------------------------------------------------------
base_clf2 = xgb.XGBClassifier(
    objective="binary:logistic",
    eval_metric="logloss",
    random_state=RANDOM_SEED,
    n_jobs=-1,
    tree_method="hist",   #
)

pipe2 = Pipeline([
    ("prep", pre2),
    ("clf", base_clf2),
])

#   +   CV    list
splits2 = list(
    grouped_stratified_splits(df2.index, y2, groups2,
                              n_splits=5, seed=RANDOM_SEED)
)

#    Mixed=1, Non-AD=0
pos = int(y2.sum())
neg = int(len(y2) - pos)
ratio = neg / pos if pos > 0 else 1.0
print(f"[Stage-2|WMH] Mixed={pos}, Non-AD={neg}, neg/pos={ratio:.2f}")

# ------------------------------------------------------------
# 2)
# ------------------------------------------------------------
param_dist2 = {
    "clf__max_depth":        [2, 3, 4, 5],
    "clf__learning_rate":    [0.01, 0.02, 0.03, 0.05, 0.08],
    "clf__n_estimators":     [300, 500, 700, 900],
    "clf__subsample":        [0.7, 0.85, 1.0],
    "clf__colsample_bytree": [0.6, 0.8, 1.0],
    "clf__min_child_weight": [1, 3, 5, 7],
    "clf__gamma":            [0.0, 0.5, 1.0],
```

```
    "clf__reg_lambda":       [0.1, 1.0, 5.0],
    "clf__reg_alpha":        [0.0, 0.1, 0.5],
    #          Mixed
    "clf__scale_pos_weight": [1.0, ratio*0.8, ratio, ratio*1.2],
}

# ------------------------------------------------------------
# 3)    Balanced accuracy = (recall0 + recall1)/2
#    =>     Mixed   Non-AD   recall
# ------------------------------------------------------------
bal_acc_scorer = make_scorer(balanced_accuracy_score)

search2 = RandomizedSearchCV(
    estimator=pipe2,
    param_distributions=param_dist2,
    n_iter=60,                    #      30
    scoring=bal_acc_scorer,
    cv=splits2,
    n_jobs=-1,
    random_state=RANDOM_SEED,
    verbose=2,
    refit=True,
)

print("\n[Stage-2|WMH]   RandomizedSearchCV  balanced accuracy ...")
search2.fit(X2, y2)

print("\n[Stage-2|WMH]   CV balanced accuracy:",
      round(search2.best_score_, 3))
print("[Stage-2|WMH]    :")
for k, v in search2.best_params_.items():
    print("  ", k, "=", v)


for k, v in search2.best_params_.items():

    print("   ", k, "=", v)

#       pipe2 / clf2

pipe2 = search2.best_estimator_

clf2 = pipe2.named_steps["clf"]

#

import json
```

```
(HIER_DIR / "stage2_best_params.json").write_text(
    json.dumps(search2.best_params_, indent=2), encoding="utf-8"
)

splits2 = grouped_stratified_splits(df2.index, y2, groups2, n_splits=5,␣
 ↪seed=RANDOM_SEED)

y2_true, y2_pred, y2_prob = [], [], []

for tr_idx, te_idx in splits2:
    pipe2.fit(X2.iloc[tr_idx], y2.iloc[tr_idx])
    y2_pred.extend( pipe2.predict(X2.iloc[te_idx]) )
    y2_prob.extend( pipe2.predict_proba(X2.iloc[te_idx]) )

y2_true = y2.values

y2_pred = np.array(y2_pred)

y2_prob = np.vstack(y2_prob)

rep2, m2 = metrics_binary(y2_true, y2_pred, y2_prob)

(Path(HIER_DIR/"stage2_report.txt")).write_text(rep2, encoding="utf-8")

with open(HIER_DIR/"stage2_metrics.json","w") as f: json.dump(m2, f, indent=2)

print(rep2)

print("[Stage-2] F1:", round(m2["f1"],3), "BalancedAcc:",␣
 ↪round(m2["balanced_acc"],3), "AUC:", round(m2["auc"],3))

# ==========          ==========
# ==========          ==========

#    Stage-2

def build_stage2_matrix(df_slice: pd.DataFrame, ref_cols: list[str] | None =␣
 ↪None) -> pd.DataFrame:

    blocks = []
```

```
    # WMH
    wmh_cols = [c for c in df_slice.columns if re.search(r"wmh", c, flags=re.I)]
    icv_col = None
    for cand in ["tiv","ICV","icv","Total","total"]:
        if cand in df_slice.columns:
            icv_col = cand; break
    if wmh_cols:
        wmh = df_slice[wmh_cols].apply(pd.to_numeric, errors="coerce")
        if icv_col:
            icv = pd.to_numeric(df_slice[icv_col], errors="coerce").replace(0,␣
↪np.nan)
            for c in wmh_cols:
                blocks.append( (wmh[c]/icv).rename(f"{c}_pct_icv") )
        else:
            for c in wmh_cols:
                blocks.append(wmh[c].rename(c))
    # vascular burden
    vasc_cols = [
        ␣
↪"hypertension","Hypertensive_disorder","Diabetes_mellitus","Atrial_fibrillation",
        ␣
↪"Ischemic_heart_disease","Cerebrovascular_accident","Transient_ischemic_attack",
        "Heart_failure","Coronary_arteriosclerosis"
    ]
    vasc_cols = [c for c in vasc_cols if c in df_slice.columns]
```

```
    if vasc_cols:

        blocks.append( df_slice[vasc_cols].apply(pd.to_numeric,␣
↪errors="coerce").fillna(0).clip(0,1) )

    # age / sex

    if "age_at_scan_date" in df_slice.columns:

        blocks.append( pd.to_numeric(df_slice["age_at_scan_date"],␣
↪errors="coerce").rename("age") )

    elif "age_years" in df_slice.columns:

        blocks.append( pd.to_numeric(df_slice["age_years"], errors="coerce").
↪rename("age") )

    if "Gender_ID" in df_slice.columns:

        blocks.append( df_slice["Gender_ID"].astype(str).rename("Gender_ID") )

    #

    if blocks:

        Xtmp = pd.concat(blocks, axis=1).fillna(0)

    else:

        Xtmp = pd.DataFrame(index=df_slice.index)  #

    #

    if ref_cols is not None:

        Xtmp = Xtmp.reindex(columns=ref_cols, fill_value=0)

    return Xtmp

print("\n[Hier] Assemble final 3-class predictions ...")

# Stage-1      P(AD)

pipe1.fit(X1, y1)

p1 = pipe1.predict_proba(X1)[:, 1]    # P(AD)
```

```
is_ad = (p1 >= 0.5)                    #       0.6/0.7

pred_final = np.array([""]*len(df), dtype=object)

pred_final[is_ad] = "AD"

# Stage-2   "    non-AD   "  Mixed vs Non-AD

idx_pred_nonad = np.where(~is_ad)[0]

df_pred_nonad  = df.iloc[idx_pred_nonad].reset_index(drop=True)

#     pipe2

X2_pred = build_stage2_matrix(df_pred_nonad, ref_cols=X2.columns)

p2_pred = pipe2.predict_proba(X2_pred)[:, 1]      # P(Mixed | predicted non-AD)

pred_final[idx_pred_nonad] = np.where(p2_pred >= 0.5, "Mixed", "Non-AD")

#           vs

from sklearn.metrics import confusion_matrix

cm3 = confusion_matrix(df["target_diag"], pred_final,␣
 ↪labels=["AD","Mixed","Non-AD"])

cm3_df = pd.DataFrame(cm3, index=[ "true_AD","true_Mixed","true_Non-AD" ],

                           columns=["pred_AD","pred_Mixed","pred_Non-AD"])

cm3_df.to_csv(HIER_DIR/"final_confusion_matrix.csv", index=True)

#

out_pred = pd.DataFrame({

    "BrcId": df.get("BrcId", pd.Series(np.nan, index=df.index)),

    "true_diag": df["target_diag"],

    "P_AD": p1,

    "final_pred": pred_final

})
```

```
#       non-AD

out_pred.loc[~is_ad, "P_Mixed_given_nonAD"] = p2_pred

out_pred.to_csv(HIER_DIR/"preds_hierarchical.csv", index=False)

print(f"[ ] Saved hierarchical outputs to: {HIER_DIR.resolve()}")

print(cm3_df)
```

```
[Stage-1] AD vs Others
              precision    recall  f1-score   support

           0      0.394     0.296     0.338      1318
           1      0.600     0.698     0.645      1992

    accuracy                          0.538      3310
   macro avg      0.497     0.497     0.492      3310
weighted avg      0.518     0.538     0.523      3310

[Stage-1] F1: 0.645 BalancedAcc: 0.497 AUC: 0.497

[Stage-2] Mixed vs Non-AD (among non-AD cases)
[Stage-2] WMH columns not found; proceeding without WMH.
[Stage-2|WMH] Mixed=645, Non-AD=673, neg/pos=1.04

[Stage-2|WMH]   RandomizedSearchCV  balanced accuracy …
Fitting 5 folds for each of 60 candidates, totalling 300 fits
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
```

```
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=500,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
```

```
[CV] END clf__colsample_bytree=1.0, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=3, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.05,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.05,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.05,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.05,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.05,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
```

```
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=500,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=500,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.01,
```

```
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
```

```
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.05,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.05,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.05,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
```

```
clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=500,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
```

```
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=3, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=3, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.05,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.05,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.02,
```

```
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.05,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.05,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.08,
```

```
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=500,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=700,
```

```
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=3, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=3, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.05,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.05,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.05,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
```

```
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
```

```
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=3, clf__min_child_weight=5, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=3, clf__min_child_weight=5, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.3s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
```

```
clf__reg_alpha=0.5, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.01,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
```

```
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=3, clf__min_child_weight=5, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=300,
```

```
clf__reg_alpha=0.1, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.3s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
```

```
clf__max_depth=2, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.2s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.2s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
```

```
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.2s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=3, clf__min_child_weight=5, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=3, clf__min_child_weight=5, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=1.0, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.08,
```

```
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.01,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.01,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=2, clf__min_child_weight=7, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.2s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.01,
```

```
clf__max_depth=2, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=5.0,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=5, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=1.0; total time=   0.1s
```

```
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.3s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.05,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=900,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.5, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=4, clf__min_child_weight=3, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.02,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=500,
clf__reg_alpha=0.0, clf__reg_lambda=5.0, clf__scale_pos_weight=1.0,
clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
```

```
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.0, clf__reg_lambda=1.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.01,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.01,
clf__max_depth=4, clf__min_child_weight=1, clf__n_estimators=300,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=7, clf__n_estimators=500,
clf__reg_alpha=0.1, clf__reg_lambda=5.0,
clf__scale_pos_weight=0.8347286821705426, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.08,
clf__max_depth=4, clf__min_child_weight=5, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=1.0,
clf__scale_pos_weight=1.2520930232558138, clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=1.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=5, clf__n_estimators=900,
clf__reg_alpha=0.5, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.85; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.0, clf__learning_rate=0.01,
clf__max_depth=2, clf__min_child_weight=1, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=1.0, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.1s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.6, clf__gamma=0.5, clf__learning_rate=0.08,
clf__max_depth=5, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1, clf__scale_pos_weight=1.0,
clf__subsample=0.7; total time=   0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
```

```
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.0, clf__learning_rate=0.03,
clf__max_depth=5, clf__min_child_weight=7, clf__n_estimators=700,
clf__reg_alpha=0.1, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.2s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=0.8, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=3, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=0.85; total time=
0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s
[CV] END clf__colsample_bytree=1.0, clf__gamma=0.5, clf__learning_rate=0.03,
clf__max_depth=2, clf__min_child_weight=3, clf__n_estimators=900,
clf__reg_alpha=0.0, clf__reg_lambda=0.1,
clf__scale_pos_weight=1.0434108527131782, clf__subsample=1.0; total time=   0.1s

[Stage-2|WMH]   CV balanced accuracy: 0.581
[Stage-2|WMH]    :
   clf__subsample = 0.7
   clf__scale_pos_weight = 1.0434108527131782
   clf__reg_lambda = 1.0
   clf__reg_alpha = 0.5
   clf__n_estimators = 900
   clf__min_child_weight = 7
   clf__max_depth = 2
   clf__learning_rate = 0.03
   clf__gamma = 1.0
   clf__colsample_bytree = 0.6
    clf__subsample = 0.7
    clf__scale_pos_weight = 1.0434108527131782
    clf__reg_lambda = 1.0
    clf__reg_alpha = 0.5
    clf__n_estimators = 900
    clf__min_child_weight = 7
    clf__max_depth = 2
    clf__learning_rate = 0.03
    clf__gamma = 1.0
    clf__colsample_bytree = 0.6
```

```
              precision    recall  f1-score   support

           0      0.519     0.594     0.554       673
           1      0.502     0.426     0.461       645

    accuracy                          0.512      1318
   macro avg      0.511     0.510     0.508      1318
weighted avg      0.511     0.512     0.509      1318

[Stage-2] F1: 0.461 BalancedAcc: 0.51 AUC: 0.505

[Hier] Assemble final 3-class predictions …
[ ] Saved hierarchical outputs to: /home/jovyan/hier_out
             pred_AD  pred_Mixed  pred_Non-AD
true_AD         1946          17           29
true_Mixed       137         302          206
true_Non-AD       52         226          395
```

[10]:
```python
# ============================
# HIERARCHICAL 3-CLASS MODEL (with WMH merge)
# Stage-1: AD vs Others
# Stage-2: (Among non-AD) Mixed vs Non-AD using WMH + vascular features
# ============================

from pathlib import Path
import numpy as np, pandas as pd, re, json
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (classification_report, f1_score,
                             balanced_accuracy_score, roc_auc_score,␣
 ↪confusion_matrix)
from sklearn.model_selection import StratifiedKFold
import xgboost as xgb

HIER_DIR = Path("hier_out"); HIER_DIR.mkdir(parents=True, exist_ok=True)
RANDOM_SEED = 42
```

[11]:
```python
# ---------- 0) WMH           ----------

WMH_FILE = "WMH.xlsx"  # ←   WMH     .xlsx / .xls / .csv

def clean_brcid(s: pd.Series) -> pd.Series:

    return (s.astype(str).str.strip()
```

```
            .str.replace(r"\.0$", "", regex=True)
            .str.replace(r"[^0-9]", "", regex=True))
def load_wmh_unique(path: str | Path) -> pd.DataFrame:
    p = str(path)
    if p.lower().endswith((".xlsx", ".xls")):
        w = pd.read_excel(p, dtype={"BrcId": str})
    else:
        w = pd.read_csv(p, dtype={"BrcId": str}, low_memory=False)
    #
    colmap = {"BrcId":"BrcId", "WMH(77)":"WMH_77", "Intracranial-volume":"ICV"}
    w = w.rename(columns=colmap)
    #       BrcId
    need = [c for c in ["BrcId","WMH_77","ICV"] if c in w.columns]
    w = w[need].copy()
    w["BrcId"] = clean_brcid(w["BrcId"])
    #
    for c in ["WMH_77","ICV"]:
        if c in w.columns:
            w[c] = pd.to_numeric(w[c], errors="coerce")
    # --    BrcId     -- #
    agg_rules = {}
    if "WMH_77" in w.columns:
        agg_rules["WMH_77"] = "max"    #      "mean"   "sum"
```

```
    if "ICV" in w.columns:
        agg_rules["ICV"] = "median"    # ICV    median/mean
    w_unique = w.groupby("BrcId", as_index=False).agg(agg_rules)
    return w_unique
try:
    wmh_df = load_wmh_unique(WMH_FILE)
    #   df   BrcId
    if "BrcId" in df.columns:
        df["BrcId"] = clean_brcid(df["BrcId"])
    #
    df = df.merge(wmh_df, on="BrcId", how="left", validate="m:1")
    #   WMH
    df["WMH_77"] = pd.to_numeric(df["WMH_77"], errors="coerce")
    df["ICV"]    = pd.to_numeric(df["ICV"],    errors="coerce").replace(0, np.
↪nan)
    df["wmh_icv_pct"] = (df["WMH_77"] / df["ICV"]).replace([np.inf,-np.inf], np.
↪nan) * 100.0
    df["wmh_log_icv"] = np.log1p( (df["WMH_77"] / df["ICV"]).replace([np.
↪inf,-np.inf], np.nan) )
    df["wmh_missing"] = df["WMH_77"].isna().astype(int)
    print(f"[WMH] merged unique: {wmh_df.shape[0]} rows; non-missing WMH:␣
↪{(~df['WMH_77'].isna()).sum()}")
except Exception as e:
    print("[WMH] merge failed; proceeding without WMH. Error:", e)
    df["wmh_icv_pct"] = np.nan
    df["wmh_log_icv"] = np.nan
```

```
    df["wmh_missing"] = 1
```

```
[WMH] merged unique: 5833 rows; non-missing WMH: 3088
```

[12]:
```
# ---------- 1)   ----------
def make_ohe():
    try:
        return OneHotEncoder(handle_unknown="ignore", sparse_output=False)
    except TypeError:
        from sklearn.preprocessing import OneHotEncoder as OHE
        return OHE(handle_unknown="ignore", sparse=False)

def build_preprocess(num_cols, cat_cols):
    return ColumnTransformer(
        transformers=[
            ("num", Pipeline([("imp", SimpleImputer(strategy="median")),
                              ("sc",  StandardScaler())]), num_cols),
            ("cat", Pipeline([("imp", SimpleImputer(strategy="most_frequent")),
                              ("oh",  make_ohe())]), cat_cols),
        ],
        remainder="drop",
    )

def grouped_stratified_splits(df_index, y_series, groups_series, n_splits=5,␣
 ↪seed=42):
    try:
        from sklearn.model_selection import StratifiedGroupKFold
        cv = StratifiedGroupKFold(n_splits=n_splits, shuffle=True,␣
 ↪random_state=seed)
        return list(cv.split(df_index, y_series, groups_series))
    except Exception:
        g = pd.DataFrame({"BrcId": groups_series, "y": y_series}).
 ↪groupby("BrcId").first().reset_index()
        skf = StratifiedKFold(n_splits=n_splits, shuffle=True,␣
 ↪random_state=seed)
        splits = []
        for tr, te in skf.split(g["BrcId"], g["y"]):
            tr_mask = groups_series.isin(g.loc[tr,"BrcId"])
            te_mask = groups_series.isin(g.loc[te,"BrcId"])
            splits.append((df_index[tr_mask], df_index[te_mask]))
        return splits

def metrics_binary(y_true, y_pred, y_prob, pos_label=1):
    rep = classification_report(y_true, y_pred, digits=3)
    f1  = f1_score(y_true, y_pred, average="binary", pos_label=pos_label)
```

```
    bacc= balanced_accuracy_score(y_true, y_pred)
    try:
        #
        auc = roc_auc_score(y_true, y_prob[:,1 if y_prob.shape[1]>1 else 0])
    except Exception:
        auc = np.nan
    return rep, dict(f1=f1, balanced_acc=bacc, auc=auc)
```

[13]:
```
# ---------- 2) Stage-1 AD vs Others ----------
print("\n[Stage-1] AD vs Others")

y1 = (df["target_diag"] == "AD").astype(int)

#       X_all MRI%+ + +age/sex
if 'X_all' in globals():
    X1 = X_all.copy()
else:
    pct_cols = [c for c in df.columns if c.endswith("__pct")]
    num_extra = []
    if "age_at_scan_date" in df.columns: num_extra.append("age_at_scan_date")
    elif "age_years" in df.columns:       num_extra.append("age_years")
    comorb_cols = [c for c in [
        ␣
 ↪"hypertension","Hypertensive_disorder","Diabetes_mellitus","Atrial_fibrillation",
        ␣
 ↪"Ischemic_heart_disease","Cerebrovascular_accident","Transient_ischemic_attack",
        ␣
 ↪"Heart_failure","Coronary_arteriosclerosis","falls","Chronic_obstructive_lung_disease"
    ] if c in df.columns]
    cat_cols = ["Gender_ID"] if "Gender_ID" in df.columns else []
    X1 = pd.concat([df[pct_cols].apply(pd.to_numeric, errors="coerce"),
                    df[num_extra].apply(pd.to_numeric, errors="coerce"),
                    df[comorb_cols].apply(pd.to_numeric, errors="coerce"),
                    df[cat_cols]], axis=1).fillna(0)

num_cols1 = X1.select_dtypes(include=[np.number]).columns.tolist()
cat_cols1 = [c for c in X1.columns if c not in num_cols1]
pre1 = build_preprocess(num_cols1, cat_cols1)
clf1 = xgb.XGBClassifier(
    objective="binary:logistic",
    max_depth=3, learning_rate=0.05, n_estimators=600,
    subsample=0.8, colsample_bytree=0.8, reg_lambda=1.0,
    random_state=RANDOM_SEED, n_jobs=-1, eval_metric="logloss"
)
pipe1 = Pipeline([("prep", pre1), ("clf", clf1)])
```

```
groups = df.get("BrcId", pd.Series(np.arange(len(df)))).astype(str)
splits1 = grouped_stratified_splits(df.index, y1, groups, n_splits=5,␣
 ↪seed=RANDOM_SEED)

y1_pred, y1_prob = [], []
for tr_idx, te_idx in splits1:
    pipe1.fit(X1.iloc[tr_idx], y1.iloc[tr_idx])
    y1_pred.extend(pipe1.predict(X1.iloc[te_idx]))
    y1_prob.extend(pipe1.predict_proba(X1.iloc[te_idx]))
y1_pred = np.array(y1_pred); y1_prob = np.vstack(y1_prob)

rep1, m1 = metrics_binary(y1.values, y1_pred, y1_prob, pos_label=1)
(HIER_DIR/"stage1_report.txt").write_text(rep1, encoding="utf-8")
with open(HIER_DIR/"stage1_metrics.json","w") as f: json.dump(m1, f, indent=2)
print(rep1)
print("[Stage-1] F1:", round(m1["f1"],3), "BalancedAcc:",␣
 ↪round(m1["balanced_acc"],3), "AUC:", round(m1["auc"],3))

# ---------- 3) Stage-2 Mixed vs Non-AD  AD  ----------
print("\n[Stage-2] Mixed vs Non-AD (among non-AD cases)")

nonad_mask = (df["target_diag"] != "AD")
df2 = df.loc[nonad_mask].reset_index(drop=True)
groups2 = df2.get("BrcId", pd.Series(np.arange(len(df2)))).astype(str)
y2 = (df2["target_diag"] == "Mixed").astype(int)

def build_stage2_matrix(df_slice: pd.DataFrame, ref_cols: list[str] | None =␣
 ↪None) -> pd.DataFrame:
    blocks = []
    # WMH features
    cols_to_use = []
    for c in ["wmh_icv_pct","wmh_log_icv","wmh_missing"]:
        if c in df_slice.columns:
            cols_to_use.append(c)
    if cols_to_use:
        blocks.append( df_slice[cols_to_use].apply(pd.to_numeric,␣
 ↪errors="coerce").fillna(0) )
    else:
        print("[Stage-2] WMH columns not found; proceeding without WMH.")

    # vascular burden
    vasc_cols = [
        ␣
 ↪"hypertension","Hypertensive_disorder","Diabetes_mellitus","Atrial_fibrillation",
        ␣
 ↪"Ischemic_heart_disease","Cerebrovascular_accident","Transient_ischemic_attack",
```

```
        "Heart_failure","Coronary_arteriosclerosis"
    ]
    vasc_cols = [c for c in vasc_cols if c in df_slice.columns]
    if vasc_cols:
        blocks.append( df_slice[vasc_cols].apply(pd.to_numeric,␣
↪errors="coerce").fillna(0).clip(0,1) )

    # age / sex
    if "age_at_scan_date" in df_slice.columns:
        blocks.append( pd.to_numeric(df_slice["age_at_scan_date"],␣
↪errors="coerce").rename("age") )
    elif "age_years" in df_slice.columns:
        blocks.append( pd.to_numeric(df_slice["age_years"], errors="coerce").
↪rename("age") )
    if "Gender_ID" in df_slice.columns:
        blocks.append( df_slice["Gender_ID"].astype(str).rename("Gender_ID") )

    Xtmp = pd.concat(blocks, axis=1) if blocks else pd.DataFrame(index=df_slice.
↪index)
    Xtmp = Xtmp.fillna(0)

    if ref_cols is not None:
        #
        Xtmp = Xtmp.reindex(columns=ref_cols, fill_value=0)
    return Xtmp

X2 = build_stage2_matrix(df2)
num_cols2 = X2.select_dtypes(include=[np.number]).columns.tolist()
cat_cols2 = [c for c in X2.columns if c not in num_cols2]

# ----   Stage-2    ----
feat_stage2 = pd.DataFrame({"feature_name": X2.columns})
feat_stage2.to_csv(HIER_DIR/"feature_list_stage2.csv", index=False)
print("[ ] Saved Stage-2 feature list →", HIER_DIR/"feature_list_stage2.csv")

pre2 = build_preprocess(num_cols2, cat_cols2)

clf2 = xgb.XGBClassifier(
    objective="binary:logistic",
    max_depth=3, learning_rate=0.05, n_estimators=600,
    subsample=0.8, colsample_bytree=0.8,
    reg_lambda=1.0, random_state=RANDOM_SEED, n_jobs=-1,
    eval_metric="logloss"
)
pipe2 = Pipeline([("prep", pre2), ("clf", clf2)])
```

```
splits2 = grouped_stratified_splits(df2.index, y2, groups2, n_splits=5,␣
 ↪seed=RANDOM_SEED)

y2_pred, y2_prob = [], []
for tr_idx, te_idx in splits2:
    pipe2.fit(X2.iloc[tr_idx], y2.iloc[tr_idx])
    y2_pred.extend( pipe2.predict(X2.iloc[te_idx]) )
    y2_prob.extend( pipe2.predict_proba(X2.iloc[te_idx]) )

y2_pred = np.array(y2_pred)
y2_prob = np.vstack(y2_prob)

rep2, m2 = metrics_binary(y2.values, y2_pred, y2_prob, pos_label=1)
(HIER_DIR/"stage2_report.txt").write_text(rep2, encoding="utf-8")
with open(HIER_DIR/"stage2_metrics.json","w") as f: json.dump(m2, f, indent=2)
print(rep2)
print("[Stage-2] F1:", round(m2["f1"],3), "BalancedAcc:",␣
 ↪round(m2["balanced_acc"],3), "AUC:", round(m2["auc"],3))
#from sklearn.model_selection import RandomizedSearchCV

#from sklearn.metrics import make_scorer, balanced_accuracy_score

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

# 1)   XGBoost + Pipeline

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

#base_clf2 = xgb.XGBClassifier(

#    objective="binary:logistic",
#
#   eval_metric="logloss",

#    random_state=RANDOM_SEED,

#    n_jobs=-1,

#    tree_method="hist",   #

#)

#pipe2 = Pipeline([

#    ("prep", pre2),
#
#    ("clf", base_clf2),
```

```
#])
#   +   CV    list
#splits2 = list(
#    grouped_stratified_splits(df2.index, y2, groups2,
#                              n_splits=5, seed=RANDOM_SEED)
#)
#    Mixed=1, Non-AD=0
#pos = int(y2.sum())
#neg = int(len(y2) - pos)
#ratio = neg / pos if pos > 0 else 1.0
#print(f"[Stage-2|WMH] Mixed={pos}, Non-AD={neg}, neg/pos={ratio:.2f}")
# ------------------------------------------------------------
# 2)
# ------------------------------------------------------------
#param_dist2 = {
#    "clf__max_depth":        [2, 3, 4, 5],
#    "clf__learning_rate":    [0.01, 0.02, 0.03, 0.05, 0.08],
#    "clf__n_estimators":     [300, 500, 700, 900],
#    "clf__subsample":        [0.7, 0.85, 1.0],
#    "clf__colsample_bytree": [0.6, 0.8, 1.0],
#    "clf__min_child_weight": [1, 3, 5, 7],
#    "clf__gamma":            [0.0, 0.5, 1.0],
#    "clf__reg_lambda":       [0.1, 1.0, 5.0],
```

```
#    "clf__reg_alpha":        [0.0, 0.1, 0.5],
#    #          Mixed
#    "clf__scale_pos_weight": [1.0, ratio*0.8, ratio, ratio*1.2],
#}
# ------------------------------------------------------------
# 3)    Balanced accuracy = (recall0 + recall1)/2
#    =>     Mixed   Non-AD   recall
# ------------------------------------------------------------
#bal_acc_scorer = make_scorer(balanced_accuracy_score)
#search2 = RandomizedSearchCV(
#    estimator=pipe2,
#    param_distributions=param_dist2,
#    n_iter=60,                    #      30
#    scoring=bal_acc_scorer,
#    cv=splits2,
#    n_jobs=-1,
#    random_state=RANDOM_SEED,
#    verbose=2,
#    refit=True,
#)
#print("\n[Stage-2|WMH]   RandomizedSearchCV  balanced accuracy ...")
#search2.fit(X2, y2)
#print("\n[Stage-2|WMH]   CV balanced accuracy:",
#      round(search2.best_score_, 3))
```

```
#print("[Stage-2|WMH]    :")
#for k, v in search2.best_params_.items():
#    print("  ", k, "=", v)
#       Stage-2
#pipe2 = search2.best_estimator_
#clf2 = pipe2.named_steps["clf"]
# ------------------------------------------------------------
# 4)    splits2    CV   threshold=0.5
# ------------------------------------------------------------
y2_true, y2_pred, y2_prob = [], [], []
for tr_idx, te_idx in splits2:
    pipe2.fit(X2.iloc[tr_idx], y2.iloc[tr_idx])
    prob_te = pipe2.predict_proba(X2.iloc[te_idx])[:, 1]
    pred_te = (prob_te >= 0.5).astype(int)
    y2_true.extend(y2.iloc[te_idx])
    y2_pred.extend(pred_te)
    y2_prob.extend(prob_te)
y2_true = np.array(y2_true)
y2_pred = np.array(y2_pred)
y2_prob = np.array(y2_prob)
rep2, m2 = metrics_binary(y2_true, y2_pred, y2_prob, pos_label=1)
print("\n[Stage-2|WMH] CV classification report (threshold=0.5):")
print(rep2)
```

```
print("[Stage-2|WMH] F1:", round(m2["f1"], 3),

      "BalancedAcc:", round(m2["balanced_acc"], 3),

      "AUC:", round(m2["auc"], 3))
```

```
[Stage-1] AD vs Others
              precision    recall  f1-score   support

           0      0.394     0.296     0.338      1318
           1      0.600     0.698     0.645      1992

    accuracy                          0.538      3310
   macro avg      0.497     0.497     0.492      3310
weighted avg      0.518     0.538     0.523      3310

[Stage-1] F1: 0.645 BalancedAcc: 0.497 AUC: 0.497

[Stage-2] Mixed vs Non-AD (among non-AD cases)
[ ] Saved Stage-2 feature list → hier_out/feature_list_stage2.csv
              precision    recall  f1-score   support

           0      0.530     0.532     0.531       673
           1      0.510     0.509     0.509       645

    accuracy                          0.520      1318
   macro avg      0.520     0.520     0.520      1318
weighted avg      0.520     0.520     0.520      1318

[Stage-2] F1: 0.509 BalancedAcc: 0.52 AUC: 0.517

[Stage-2|WMH] CV classification report (threshold=0.5):
              precision    recall  f1-score   support

           0      0.680     0.682     0.681       673
           1      0.667     0.665     0.666       645

    accuracy                          0.674      1318
   macro avg      0.674     0.674     0.674      1318
weighted avg      0.674     0.674     0.674      1318

[Stage-2|WMH] F1: 0.666 BalancedAcc: 0.674 AUC: nan
```

[14]:
```
# ---------- 4)      Stage-1      ----------

from pathlib import Path

import numpy as np, pandas as pd

from sklearn.metrics import confusion_matrix, f1_score

print("\n[Hier] Assemble final 3-class predictions (threshold sweep) ...")

#

try:

    HIER_DIR

except NameError:

    HIER_DIR = Path("hier_out")

HIER_DIR.mkdir(parents=True, exist_ok=True)

#

y_true_col = "target_diag" if "target_diag" in df.columns else ("diag3" if␣
 ↪"diag3" in df.columns else None)

assert y_true_col is not None, "     target_diag / diag3 "

y_true = df[y_true_col].values

labels3 = ["AD", "Mixed", "Non-AD"]

# ===== Stage-1        P(AD) =====

pipe1.fit(X1, y1)

p1_all = pipe1.predict_proba(X1)[:, 1]   # P(AD)

# ===== Stage-2   Stage-2               =====

pipe2.fit(X2, y2)

#

thr_list = [0.50, 0.55, 0.60, 0.65, 0.70]
```

```
rows = []

for thr in thr_list:

    # --- Stage-1      ---

    is_ad = (p1_all >= thr)

    pred_final = np.array([""]*len(df), dtype=object)

    pred_final[is_ad] = "AD"

    # --- Stage-2  "   non-AD"     vs  AD ---

    idx_pred_nonad = np.where(~is_ad)[0]

    if idx_pred_nonad.size > 0:

        #            X2_pred

        df_pred_nonad = df.iloc[idx_pred_nonad].reset_index(drop=True)

        X2_pred = build_stage2_matrix(df_pred_nonad, ref_cols=X2.columns)  #␣
↪

        p2_pred = pipe2.predict_proba(X2_pred)[:, 1]                       #␣
↪P(Mixed | predicted non-AD)

        pred_final[idx_pred_nonad] = np.where(p2_pred >= 0.50, "Mixed",␣
↪"Non-AD")

    else:

        p2_pred = np.array([], dtype=float)

    # ---         ---

    cm3 = confusion_matrix(y_true, pred_final, labels=labels3)

    cm3_df = pd.DataFrame(cm3, index=[f"true_{l}" for l in labels3],

                               columns=[f"pred_{l}" for l in labels3])

    cm3_df.to_csv(HIER_DIR / f"final_confusion_matrix_thr{thr:.2f}.csv",␣
↪index=True)

    macro_f1 = f1_score(y_true, pred_final, average="macro", labels=labels3)
```

```
    acc      = (pred_final == y_true).mean()

    rec_AD   = (pred_final[y_true=="AD"]=="AD").mean() if (y_true=="AD").
 ↪sum()>0 else np.nan

    rows.append({"thr1": float(thr), "macro_f1": float(macro_f1), "accuracy":␣
 ↪float(acc), "recall_AD": float(rec_AD)})

    # ---        ---

    out_pred = pd.DataFrame({

        "BrcId": df.get("BrcId", pd.Series(np.nan, index=df.index)),

        "true_diag": y_true,

        "P_AD": p1_all,

        "final_pred": pred_final

    })

    if idx_pred_nonad.size > 0:

        out_pred.loc[~is_ad, "P_Mixed_given_nonAD"] = p2_pred

    out_pred.to_csv(HIER_DIR / f"preds_hierarchical_thr{thr:.2f}.csv",␣
 ↪index=False)

#    -

summary = pd.DataFrame(rows).sort_values("macro_f1", ascending=False)

summary.to_csv(HIER_DIR / "threshold_sweep_stage1_summary.csv", index=False)

print("\n[ ] Saved threshold sweep summary →", HIER_DIR /␣
 ↪"threshold_sweep_stage1_summary.csv")

print(summary.head(10))
```

```
[Hier] Assemble final 3-class predictions (threshold sweep) …

[ ] Saved threshold sweep summary → hier_out/threshold_sweep_stage1_summary.csv
   thr1  macro_f1  accuracy  recall_AD
```

```
1  0.55  0.805205  0.864048   0.956827
2  0.60  0.798483  0.853776   0.921687
0  0.50  0.793629  0.859215   0.976908
3  0.65  0.785501  0.834441   0.872490
4  0.70  0.742765  0.783384   0.782631
```

[15]:
```
# ---------- 4)      Stage-1   sweep 0.50-0.60  10  +     ----------

from pathlib import Path

import numpy as np, pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix, f1_score

from IPython.display import display

print("\n[Hier] Assemble final 3-class predictions (threshold sweep, 0.50-0.60␣
 ↪random 10) ...")

#

try:

    HIER_DIR

except NameError:

    HIER_DIR = Path("hier_out")

HIER_DIR.mkdir(parents=True, exist_ok=True)

#

y_true_col = "target_diag" if "target_diag" in df.columns else ("diag3" if␣
 ↪"diag3" in df.columns else None)

assert y_true_col is not None, "     target_diag / diag3 "

y_true = df[y_true_col].values

labels3 = ["AD", "Mixed", "Non-AD"]

# Stage-1      P(AD)

pipe1.fit(X1, y1)
```

```
p1_all = pipe1.predict_proba(X1)[:, 1]

# Stage-2        "  AD "

pipe2.fit(X2, y2)

# ----   0.50-0.60    10         3    ----

rng = np.random.default_rng(2025)

thr_list = np.round(np.sort(rng.uniform(0.50, 0.60, size=10)), 3)

print("Thresholds (Stage-1, P[AD]   thr):", thr_list.tolist())

rows = []

for thr in thr_list:

    # Stage-1

    is_ad = (p1_all >= thr)

    pred_final = np.array([""]*len(df), dtype=object)

    pred_final[is_ad] = "AD"

    # Stage-2  "   non-AD"   Mixed vs Non-AD

    idx_pred_nonad = np.where(~is_ad)[0]

    if idx_pred_nonad.size > 0:

        df_pred_nonad = df.iloc[idx_pred_nonad].reset_index(drop=True)

        #     build_stage2_matrix(df_slice, ref_cols=...)

        X2_pred = build_stage2_matrix(df_pred_nonad, ref_cols=X2.columns)

        p2_pred = pipe2.predict_proba(X2_pred)[:, 1]

        pred_final[idx_pred_nonad] = np.where(p2_pred >= 0.50, "Mixed",␣
↪"Non-AD")

    else:

        p2_pred = np.array([], dtype=float)
```

```
    #     +

    cm3 = confusion_matrix(y_true, pred_final, labels=labels3)

    cm3_df = pd.DataFrame(cm3, index=[f"true_{l}" for l in labels3],

                                columns=[f"pred_{l}" for l in labels3])

    cm3_df.to_csv(HIER_DIR / f"final_confusion_matrix_thr{thr:.3f}.csv",␣
 ↪index=True)

    macro_f1 = f1_score(y_true, pred_final, average="macro", labels=labels3)

    acc      = (pred_final == y_true).mean()

    rec_AD   = (pred_final[y_true=="AD"]=="AD").mean() if (y_true=="AD").
 ↪sum()>0 else np.nan

    rows.append({"thr1": float(thr), "macro_f1": float(macro_f1),

                 "accuracy": float(acc), "recall_AD": float(rec_AD)})

    #     /

    out_pred = pd.DataFrame({

        "BrcId": df.get("BrcId", pd.Series(np.nan, index=df.index)),

        "true_diag": y_true,

        "P_AD": p1_all,

        "final_pred": pred_final

    })

    if idx_pred_nonad.size > 0:

        out_pred.loc[~is_ad, "P_Mixed_given_nonAD"] = p2_pred

    out_pred.to_csv(HIER_DIR / f"preds_hierarchical_thr{thr:.3f}.csv",␣
 ↪index=False)

#   +

summary = pd.DataFrame(rows).sort_values("macro_f1", ascending=False)
```

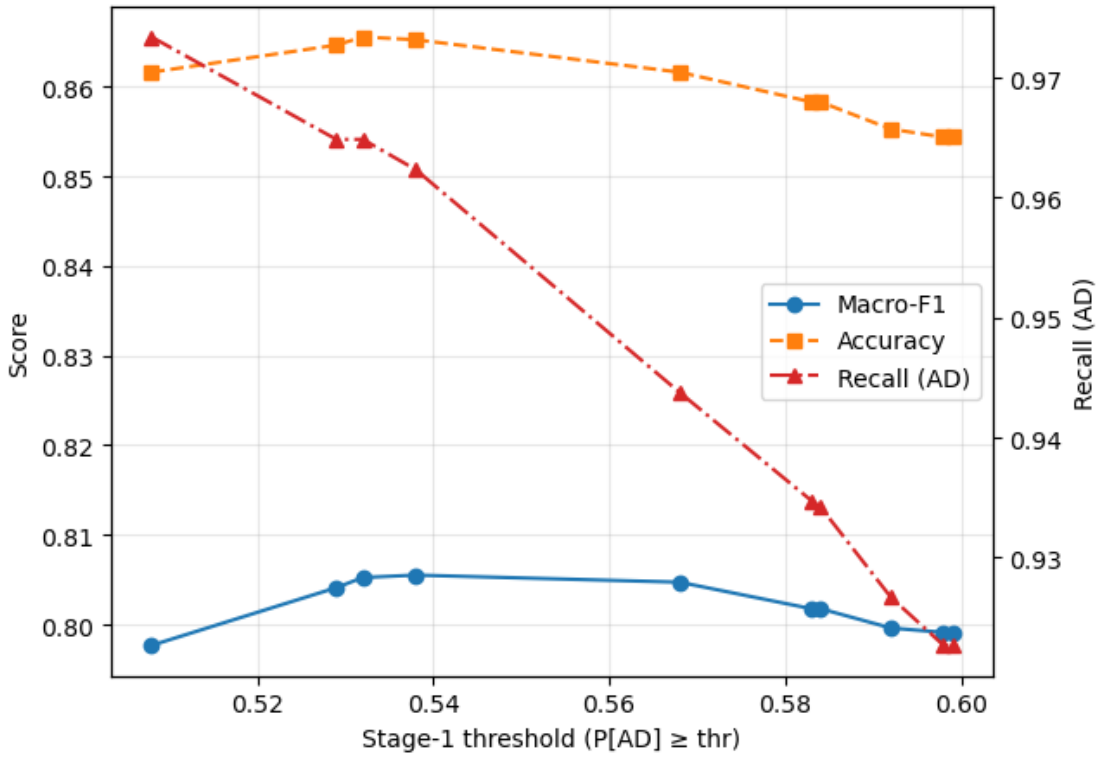
```
summary.to_csv(HIER_DIR / "threshold_sweep_stage1_summary.csv", index=False)

print(f"\n[ ] Saved threshold sweep summary → {HIER_DIR/
 ↪'threshold_sweep_stage1_summary.csv'}")

display(summary)                    # <--

best = summary.iloc[0]

print(f"\nBest thr1 = {best['thr1']:.3f} | macro-F1={best['macro_f1']:.3f} | "

      f"acc={best['accuracy']:.3f} | recall_AD={best['recall_AD']:.3f}")

#

summary_sort = summary.sort_values("thr1")

fig, ax1 = plt.subplots(figsize=(6.6,4.6))

ax1.plot(summary_sort["thr1"], summary_sort["macro_f1"], "o-", label="Macro-F1")

ax1.plot(summary_sort["thr1"], summary_sort["accuracy"], "s--",␣
 ↪label="Accuracy")

ax1.set_xlabel("Stage-1 threshold (P[AD]   thr)")

ax1.set_ylabel("Score")

ax1.grid(alpha=.3)

ax2 = ax1.twinx()

ax2.plot(summary_sort["thr1"], summary_sort["recall_AD"], "^-.", color="tab:
 ↪red", label="Recall (AD)")

ax2.set_ylabel("Recall (AD)")

#

lines, labels = ax1.get_legend_handles_labels()

lines2, labels2 = ax2.get_legend_handles_labels()

ax1.legend(lines+lines2, labels+labels2, loc="best")

plt.tight_layout()
```

```
plt.show()                          # <--

fig.savefig(HIER_DIR / "plot_threshold_sweep_stage1.png", dpi=300)
```

```
[Hier] Assemble final 3-class predictions (threshold sweep, 0.50-0.60 random 10)
…
Thresholds (Stage-1, P[AD]  thr): [0.508, 0.529, 0.532, 0.538, 0.568, 0.583,
0.584, 0.592, 0.598, 0.599]

[ ] Saved threshold sweep summary → hier_out/threshold_sweep_stage1_summary.csv
```

	thr1	macro_f1	accuracy	recall_AD
3	0.538	0.805512	0.865257	0.962349
2	0.532	0.805236	0.865559	0.964859
4	0.568	0.804717	0.861631	0.943775
1	0.529	0.804108	0.864653	0.964859
6	0.584	0.801764	0.858308	0.934237
5	0.583	0.801754	0.858308	0.934739
7	0.592	0.799605	0.855287	0.926707
8	0.598	0.799113	0.854381	0.922691
9	0.599	0.799017	0.854381	0.922691
0	0.508	0.797667	0.861631	0.973394

```
Best thr1 = 0.538 | macro-F1=0.806 | acc=0.865 | recall_AD=0.962
```



[16]:
```
# ============== 2.      Stage-1 / Stage-2 / Overall  1    ==============

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

from pathlib import Path

from sklearn.metrics import (roc_auc_score, average_precision_score, roc_curve,␣
 ↪precision_recall_curve,

                             confusion_matrix, classification_report,␣
 ↪brier_score_loss, f1_score)

from sklearn.calibration import calibration_curve, CalibratedClassifierCV

from sklearn.preprocessing import label_binarize

OUTM = Path("surv_out")/"analysis_out"; OUTM.mkdir(parents=True, exist_ok=True)

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

def metrics_binary_block(y_true, prob, thr=0.5):
```

```
    pred = (prob >= thr).astype(int)
    cm = confusion_matrix(y_true, pred, labels=[0,1])
    rep = classification_report(y_true, pred, output_dict=True, zero_division=0)
    fpr, tpr, _ = roc_curve(y_true, prob)
    prec, rec, _ = precision_recall_curve(y_true, prob)
    out = {
        "roc_auc": roc_auc_score(y_true, prob),
        "pr_auc":  average_precision_score(y_true, prob),
        "accuracy": (pred==y_true).mean(),
        "precision_pos": rep["1"]["precision"],
        "recall_pos":    rep["1"]["recall"],
        "f1_pos":        rep["1"]["f1-score"],
        "brier": brier_score_loss(y_true, prob),
        "cm_TN":cm[0,0], "cm_FP":cm[0,1], "cm_FN":cm[1,0], "cm_TP":cm[1,1]
    }
    curves = {"roc":(fpr,tpr), "pr":(prec,rec)}
    return out, curves
# ---------- Stage-1    ----------
#     1 fit pipe1      fit
try:
    P1_all = p1_all
except NameError:
    pipe1.fit(X1, y1)
    P1_all = pipe1.predict_proba(X1)[:,1]
```

```
#    1 sweep  best['thr1']     0.60

try:

    thr1 = float(best["thr1"])

except Exception:

    thr1 = 0.60

# Stage-1         OOF/

m1, curves1 = metrics_binary_block(y1.values, P1_all, thr=thr1)

stage1_df = pd.DataFrame([m1])

#      vs Isotonic

cal_x1, cal_y1 = calibration_curve(y1.values, P1_all, n_bins=10,␣
 ↪strategy="quantile")

iso1 = CalibratedClassifierCV(pipe1, method="isotonic", cv=3).fit(X1, y1)

prob1_iso = iso1.predict_proba(X1)[:,1]

cal_x1b, cal_y1b = calibration_curve(y1.values, prob1_iso, n_bins=10,␣
 ↪strategy="quantile")

plt.figure(figsize=(4.8,4.2))

plt.plot([0,1],[0,1],"k--",alpha=.3)

plt.plot(cal_x1,  cal_y1,  "o-", label="uncalibrated")

plt.plot(cal_x1b, cal_y1b, "s-", label="isotonic")

plt.xlabel("Predicted probability"); plt.ylabel("Observed fraction")

plt.title("Stage-1 calibration"); plt.legend(); plt.grid(alpha=.3)

plt.tight_layout(); plt.savefig(OUTM/"calibration_stage1.png", dpi=300); plt.
 ↪close()

# ---------- Stage-2  Non-AD   +    X2_all P2_all ----------

# pipe2 fit fit
```

```
try:
    _ = pipe2  # noqa
except NameError:
    raise RuntimeError("  1      pipe2 XGB for Stage-2 ")
#  df2/X2/y2
pipe2.fit(X2, y2)
prob2 = pipe2.predict_proba(X2)[:,1]
thr2  = 0.50
m2, curves2 = metrics_binary_block(y2.values, prob2, thr=thr2)
stage2_df = pd.DataFrame([m2])
#
cal_x2, cal_y2 = calibration_curve(y2.values, prob2, n_bins=10,␣
 ↪strategy="quantile")
iso2   = CalibratedClassifierCV(pipe2, method="isotonic", cv=3).fit(X2, y2)
prob2_iso = iso2.predict_proba(X2)[:,1]
cal_x2b, cal_y2b = calibration_curve(y2.values, prob2_iso, n_bins=10,␣
 ↪strategy="quantile")
plt.figure(figsize=(4.8,4.2))
plt.plot([0,1],[0,1],"k--",alpha=.3)
plt.plot(cal_x2,  cal_y2,  "o-", label="uncalibrated")
plt.plot(cal_x2b, cal_y2b, "s-", label="isotonic")
plt.xlabel("Predicted probability"); plt.ylabel("Observed fraction")
plt.title("Stage-2 calibration"); plt.legend(); plt.grid(alpha=.3)
plt.tight_layout(); plt.savefig(OUTM/"calibration_stage2.png", dpi=300); plt.
 ↪close()
```

```
# "  df"   Stage-2    P2_all    /

X2_all = build_stage2_matrix(df.reset_index(drop=True), ref_cols=X2.columns)

P2_all = pipe2.predict_proba(X2_all)[:,1]    #   = len(df)

# ---------- Overall     thr1   + Stage-2 thr2=0.50  ----------

#

is_ad = (P1_all >= thr1)

pred_final = np.array([""]*len(df), dtype=object)

pred_final[is_ad] = "AD"

pred_final[~is_ad] = np.where(P2_all[~is_ad] >= thr2, "Mixed", "Non-AD")

#      n×3

P_AD    = P1_all

P_Mixed = (1.0 - P1_all) * P2_all

P_NonAD = (1.0 - P1_all) * (1.0 - P2_all)

prob_mat = np.c_[P_AD, P_Mixed, P_NonAD]   # shape: (n_samples, 3)

#      + Macro-F1 + OVR AUC

y_true_col = "target_diag" if "target_diag" in df.columns else ("diag3" if␣
 ↪"diag3" in df.columns else None)

y_true3    = df[y_true_col].values

labels3    = ["AD","Mixed","Non-AD"]

cm3 = confusion_matrix(y_true3, pred_final, labels=labels3)

cm3_df = pd.DataFrame(cm3, index=[f"true_{l}" for l in labels3],␣
 ↪columns=[f"pred_{l}" for l in labels3])

rep3  = classification_report(y_true3, pred_final, labels=labels3,␣
 ↪output_dict=True, zero_division=0)

overall_tab = pd.DataFrame([{
```

```
    "macro_f1": np.mean([rep3[l]["f1-score"] for l in labels3]),
    "acc": rep3["accuracy"],
    "macro_roc_auc_ovr": roc_auc_score(label_binarize(y_true3, classes=labels3),
                                       prob_mat, average="macro",␣
 ↪multi_class="ovr")
}])
#   Excel Stage-1/Stage-2/Overall
with pd.ExcelWriter(OUTM/"final.csv") as xw:
    stage1_df.to_excel(xw, sheet_name="Stage1_metrics", index=False)
    stage2_df.to_excel(xw, sheet_name="Stage2_metrics", index=False)
    overall_tab.to_excel(xw, sheet_name="Overall_summary", index=False)
    cm3_df.to_excel(xw, sheet_name="Overall_confusion", index=True)
print("[Metrics] Saved:", OUTM/"final.csv",
      "|", OUTM/"calibration_stage1.png", "|", OUTM/"calibration_stage2.png")
```

```
[Metrics] Saved: surv_out/analysis_out/final.csv |
surv_out/analysis_out/calibration_stage1.png |
surv_out/analysis_out/calibration_stage2.png
```

[17]:
```
# ============== SHAP-style contributions for Stage-1 & Stage-2 XGBoost␣
 ↪pred_contribs ==============
import numpy as np, pandas as pd, matplotlib.pyplot as plt
from pathlib import Path
import xgboost as xgb
from sklearn.pipeline import Pipeline
OUTS = Path("surv_out")/"analysis_out"; OUTS.mkdir(parents=True, exist_ok=True)
def get_transformed_X_and_names(pipe: Pipeline, X):
```

```
    """ sklearn Pipeline   ColumnTransformer       """
    prep = pipe.named_steps["prep"]
    Xmat = prep.transform(X)
    try:
        names = prep.get_feature_names_out()
    except Exception:
        #
        names = np.array([f"f{i}" for i in range(Xmat.shape[1])])
    return Xmat, names
def shap_contrib_from_xgb(pipe: Pipeline, X, model_name:str, sample_n:int=2000):
    """  pipeline XGBClassifier  TreeSHAP  pred_contribs   top    """
    #   fit
    try:
        _ = pipe.named_steps["clf"].get_booster()
    except Exception:
        pipe.fit(X, pipe.predict(X))  #              fit
    clf = pipe.named_steps["clf"]
    #
    Xmat, names = get_transformed_X_and_names(pipe, X)
    n = Xmat.shape[0]
    if sample_n and n>sample_n:
        rng = np.random.default_rng(2025)
        idx = rng.choice(n, size=sample_n, replace=False)
```

```
        Xmat_s = Xmat[idx]
    else:
        idx = np.arange(n)
        Xmat_s = Xmat
    dmat = xgb.DMatrix(Xmat_s, feature_names=names.tolist())
    contrib = clf.get_booster().predict(dmat, pred_contribs=True)  # shape: (n,␣
↪p+1)  (n, p+1, C)
    #   /
    if contrib.ndim == 2:
        phi = contrib[:, :-1]  #  bias
    elif contrib.ndim == 3:
        # (n, p+1, C) →
        phi = contrib[:, :-1, -1]
    else:
        raise RuntimeError(f"Unexpected contrib shape: {contrib.shape}")
    #    |SHAP|
    imp = pd.DataFrame({
        "feature": names,
        "mean_abs_shap": np.abs(phi).mean(axis=0),
        "mean_shap": phi.mean(axis=0)
    }).sort_values("mean_abs_shap", ascending=False)
    #      shap
    per_sample = pd.DataFrame(phi, columns=names)
    per_sample.insert(0, "sample_index_in_X", idx)
```

```
    #  Top-20

    top20 = imp.head(20).iloc[::-1]  #

    plt.figure(figsize=(6.8, 6.0))

    plt.barh(top20["feature"], top20["mean_abs_shap"])

    plt.xlabel("mean(|SHAP|)"); plt.title(f"{model_name} contributions␣
 ↪(Top-20)")

    plt.grid(axis="x", alpha=.2)

    fig_path = OUTS/f"{model_name.lower()}_shap_bar.png"

    plt.tight_layout(); plt.savefig(fig_path, dpi=300); plt.close()

    #

    imp.to_csv(OUTS/f"{model_name.lower()}_shap_values.csv", index=False)

    per_sample.to_csv(OUTS/f"{model_name.lower()}_shap_samples.csv",␣
 ↪index=False)

    print(f"[SHAP] {model_name} saved:", fig_path, "|", OUTS/f"{model_name.
 ↪lower()}_shap_values.csv")

    return imp, per_sample

# --------- Stage-1 AD vs Others pipe1 / X1, y1  ---------

try:

    pipe1.named_steps["clf"].get_booster()

except Exception:

    pipe1.fit(X1, y1)

imp1, shap1_samples = shap_contrib_from_xgb(pipe1, X1, model_name="Stage1")

# --------- Stage-2 Mixed vs Non-AD pipe2 / X2, y2  ---------

try:

    pipe2.named_steps["clf"].get_booster()
```

```
except Exception:

    pipe2.fit(X2, y2)

imp2, shap2_samples = shap_contrib_from_xgb(pipe2, X2, model_name="Stage2")

#  Top-10

print("\n[Stage-1] Top-10 features by mean(|SHAP|):")

print(imp1.head(10).to_string(index=False))

print("\n[Stage-2] Top-10 features by mean(|SHAP|):")

print(imp2.head(10).to_string(index=False))
```

```
[SHAP] Stage1 saved: surv_out/analysis_out/stage1_shap_bar.png |
surv_out/analysis_out/stage1_shap_values.csv
[SHAP] Stage2 saved: surv_out/analysis_out/stage2_shap_bar.png |
surv_out/analysis_out/stage2_shap_values.csv

[Stage-1] Top-10 features by mean(|SHAP|):
                                  feature  mean_abs_shap  mean_shap
                                num__MMSE       0.193538   0.009166
             num__Cerebrovascular_accident       0.144728   0.005919
          num__050_Right Inf Lat Vent__pct       0.140217   0.001491
      num__118_Left Ent entorhinal area__pct       0.134004  -0.007065
          num__051_Left Inf Lat Vent__pct       0.132246   0.001231
        num__Brain_Left_0_Frontal_99__pct       0.118064   0.004361
       num__Brain_Left_4_DeepNuclei_9__pct       0.092967   0.005101
            num__049_Left Hippocampus__pct       0.091276   0.007771
num__172_Left PHG parahippocampal gyrus__pct       0.086784  -0.001344
                               num__LI_PCu       0.086562   0.004348

[Stage-2] Top-10 features by mean(|SHAP|):
                      feature  mean_abs_shap  mean_shap
             num__wmh_icv_pct       0.906033   0.009294
             num__wmh_log_icv       0.202818   0.000422
 num__Cerebrovascular_accident       0.196147  -0.005705
    num__Hypertensive_disorder       0.154991  -0.011313
            num__hypertension       0.131173   0.008345
           num__Heart_failure       0.093152   0.001439
        cat__Gender_ID_Female       0.075931   0.001702
       num__Diabetes_mellitus       0.069000   0.001603
     num__Atrial_fibrillation       0.068532  -0.000436
num__Transient_ischemic_attack       0.059454  -0.000598
```

[18]:
```
prep1 = pipe1.named_steps['prep']
feat1_names = prep1.get_feature_names_out()
print([c for c in feat1_names if 'age' in c .lower()])
```

```
['num__age']
```

[ ]:

[19]:
```
##AD P = 1827/(1827+46+20)=0.965 R=1827/1992=0.918 F1 0.941
##Mixed P = 473/(70+473+132)=0.701 R=473/645=0.733 F1 0.716
##Non-AD P = 521/(95+126+521)=0.702 R=521/673=0.774 F1 0.736
##Macro-F1   (0.941+0.716+0.736)/3 = 0.798
#-------------P=precision, R=recall F1=2PR/(P+R)----------#
```

[20]:
```
import re, numpy as np, pandas as pd
import matplotlib.pyplot as plt
from math import ceil
from pathlib import Path

OUT_DIR = Path("surv_out"); OUT_DIR.mkdir(exist_ok=True, parents=True)

# 1)  /    → 3 AD / Mixed / Non-AD
if "target_diag" in df.columns:
    grp_raw = df["target_diag"].astype(str)
elif "diag_3class" in df.columns:
    grp_raw = df["diag_3class"].astype(str)
else:
    raise ValueError("      df   `target_diag`   `diag_3class` ")

def collapse_to_3(x: str) -> str:
    s = str(x)
    if "Alzheimer" in s or s.strip().upper() in {"AD","ALZHEIMER'S␣
 ↪DISEASE","ALZHEIMER'S DISEASE"}:
        return "AD"
    if "Mixed" in s or s.strip().upper()=="MIXED":
        return "Mixed"
    return "Non-AD"

df = df.copy()
df["diag_group3"] = grp_raw.map(collapse_to_3)

#    AD → Mixed → Non-AD
GROUPS = ["AD","Mixed","Non-AD"]
```

[21]:
```
ROI_LIST = [
    # MTL
    "048_Right Hippocampus__pct","049_Left Hippocampus__pct",
    "117_Right Ent entorhinal area__pct","118_Left Ent entorhinal area__pct",
    "171_Right PHG parahippocampal gyrus__pct","172_Left PHG parahippocampal␣
 ↪gyrus__pct",
    # Lateral/basal temporal
    "155_Right MTG middle temporal gyrus__pct","156_Left MTG middle temporal␣
 ↪gyrus__pct",
    "133_Right ITG inferior temporal gyrus__pct","134_Left ITG inferior␣
 ↪temporal gyrus__pct",
    "201_Right STG superior temporal gyrus__pct","202_Left STG superior␣
 ↪temporal gyrus__pct",
    "123_Right FuG fusiform gyrus__pct","124_Left FuG fusiform gyrus__pct",
    "203_Right TMP temporal pole__pct","204_Left TMP temporal pole__pct",
    # Medial parietal
    "167_Right PCgG posterior cingulate gyrus__pct","168_Left PCgG posterior␣
 ↪cingulate gyrus__pct",
    "169_Right PCu precuneus__pct","170_Left PCu precuneus__pct",
    # Posterior parietal/occipital (PCA axis)
    "199_Right SPL superior parietal lobule__pct","200_Left SPL superior␣
 ↪parietal lobule__pct",
    "197_Right SOG superior occipital gyrus__pct","198_Left SOG superior␣
 ↪occipital gyrus__pct",
    "145_Right MOG middle occipital gyrus__pct","146_Left MOG middle occipital␣
 ↪gyrus__pct",
    "129_Right IOG inferior occipital gyrus__pct","130_Left IOG inferior␣
 ↪occipital gyrus__pct",
    # Ventricular markers
    "052_Right Lateral Ventricle__pct","053_Left Lateral Ventricle__pct",
    "050_Right Inf Lat Vent__pct","051_Left Inf Lat Vent__pct",
]

#   +   WMH/ICV
extras = []
if "age_at_scan_date" in df.columns:
    extras.append("age_at_scan_date")
elif "age_years" in df.columns:
    df["age_at_scan_date"] = pd.to_numeric(df["age_years"], errors="coerce")
    extras.append("age_at_scan_date")
if "wmh_icv_pct" in df.columns:
    extras.append("wmh_icv_pct")

FEATURES = [c for c in ROI_LIST + extras if c in df.columns]
assert len(FEATURES)>0, "   ROI/age/WMH   df          __pct      "
```

```
#
for c in FEATURES:
    df[c] = pd.to_numeric(df[c], errors="coerce")
```

[22]:
```
# 3)   y
def global_ylim(cols, groups):
    lows, highs = [], []
    for c in cols:
        for g in groups:
            v = df.loc[df["diag_group3"]==g, c].dropna().values
            if v.size:
                lows.append(np.percentile(v, 1)); highs.append(np.percentile(v,␣
 ↪99))
    if len(lows)==0: return (0.0, 1.0)
    lo, hi = float(min(lows)), float(max(highs))
    rng = hi-lo
    return (lo-0.05*rng, hi+0.05*rng)

ymin, ymax = global_ylim(FEATURES, GROUPS)

def pretty(col):
    if col.endswith("__pct"):
        return col.replace("Brain_","").replace("__pct","").replace("_"," ") +␣
 ↪" (%)"
    return col.replace("_"," ")

# 4)    violin + dot
n = len(FEATURES)
ncols = 3 if n>=3 else n
nrows = int(np.ceil(n/ncols))

fig, axes = plt.subplots(nrows, ncols, figsize=(ncols*4.6, nrows*4.3),␣
 ↪squeeze=False)
plt.subplots_adjust(wspace=0.35, hspace=0.45)
rng = np.random.default_rng(42)

for i, col in enumerate(FEATURES):
    r,c = divmod(i, ncols)
    ax = axes[r,c]
    data = [df.loc[df["diag_group3"]==g, col].dropna().values for g in GROUPS]
    #
    non_empty = [(j,v) for j,v in enumerate(data, start=1) if v.size]
    if not non_empty:
        ax.axis("off"); continue

    # violin
```

```
    ax.violinplot([v for _,v in non_empty], showextrema=False, widths=0.9)
    # dots
    for j,v in non_empty:
        x = rng.normal(loc=j, scale=0.07, size=v.size)
        ax.plot(x, v, 'o', ms=3, alpha=0.6, color="#1f77b4")

    ax.set_title(pretty(col), fontsize=10)
    ax.set_xticks([j for j,_ in non_empty])
    ax.set_xticklabels([GROUPS[j-1] for j,_ in non_empty], fontsize=9)
    ax.set_ylim(ymin, ymax)
    ax.grid(True, axis='y', alpha=0.3)

#
for k in range(n, nrows*ncols):
    rr,cc = divmod(k, ncols)
    axes[rr,cc].axis("off")

panel_path = OUT_DIR / f"fig_violin_selected_{n}.png"
plt.savefig(panel_path, dpi=300, bbox_inches="tight")
plt.close(fig)
print("Saved panel figure ->", panel_path)

# 5)
for col in FEATURES:
    fig = plt.figure(figsize=(4.2,4.0)); ax = plt.gca()
    data = [df.loc[df["diag_group3"]==g, col].dropna().values for g in GROUPS]
    non_empty = [(j,v) for j,v in enumerate(data, start=1) if v.size]
    if not non_empty:
        plt.close(fig); continue
    ax.violinplot([v for _,v in non_empty], showextrema=False, widths=0.9)
    for j,v in non_empty:
        x = np.random.normal(loc=j, scale=0.07, size=v.size)
        ax.plot(x, v, 'o', ms=3, alpha=0.6, color="#1f77b4")
    ax.set_title(pretty(col), fontsize=10)
    ax.set_xticks([j for j,_ in non_empty]); ax.set_xticklabels([GROUPS[j-1]␣
 ↪for j,_ in non_empty], fontsize=9)
    ax.set_ylim(ymin, ymax); ax.grid(True, axis='y', alpha=0.3)
    out_p = OUT_DIR / f"violin_{re.sub('[^0-9A-Za-z]+','_', col)}.png"
    plt.savefig(out_p, dpi=300, bbox_inches="tight"); plt.close(fig)
```

```
Saved panel figure -> surv_out/fig_violin_selected_2.png
```

[23]:
```
# === Violin + jitter plots for AD-signature volumes (percent), grouped by AD /␣
 ↪Mixed / Non-AD ===
import numpy as np, pandas as pd, matplotlib.pyplot as plt, re
from pathlib import Path
```

```
from math import ceil

OUTP = Path("surv_out") / "violin_ad_signature"; OUTP.mkdir(parents=True,␣
 ↪exist_ok=True)

# 0)     df['diag3']    diag3
def to_diag3(x: str) -> str:
    s = str(x)
    if "Mixed" in s or s.strip().upper()=="MIXED": return "Mixed"
    if "Alzheimer" in s or s.strip().upper() in {"AD","ALZHEIMER'S␣
 ↪DISEASE","ALZHEIMER'S DISEASE"}: return "AD"
    return "Non-AD"

if "diag3" in df.columns:
    df["diag3"] = df["diag3"].astype(str).map(to_diag3)
elif "target_diag" in df.columns:
    df["diag3"] = df["target_diag"].astype(str).map(to_diag3)
elif "diag_group3" in df.columns:
    df["diag3"] = df["diag_group3"].astype(str).map(to_diag3)
else:
    raise KeyError("   df     target_diag/diag3/diag_group3 ")

GROUPS = ["AD","Mixed","Non-AD"]
df = df[df["diag3"].isin(GROUPS)].copy().reset_index(drop=True)

# 1)     ( '048')        __pct   Total
def get_roi_pct(code: str) -> pd.Series | None:
    #   " _"      '048_Right Hippocampus'
    base = next((c for c in df.columns if c.startswith(f"{code}_")), None)
    if base is None:
        return None
    pct_col = base + "__pct"
    if pct_col in df.columns:
        return pd.to_numeric(df[pct_col], errors="coerce")
    #   __pct   /Total×100
    if "Total" in df.columns:
        num = pd.to_numeric(df[base], errors="coerce")
        den = pd.to_numeric(df["Total"], errors="coerce").replace(0, np.nan)
        return (num/den) * 100.0
    #         %
    return pd.to_numeric(df[base], errors="coerce")
```

[24]:
```
# 2)    bilateral
def bilateral(code_R: str, code_L: str, name: str) -> pd.Series:
    r = get_roi_pct(code_R); l = get_roi_pct(code_L)
    if r is None and l is None:
```

```
        return pd.Series(np.nan, index=df.index, name=name)
    if r is None: r = pd.Series(0.0, index=df.index)
    if l is None: l = pd.Series(0.0, index=df.index)
    s = (r.fillna(0) + l.fillna(0)).replace([np.inf,-np.inf], np.nan)
    s.name = name
    return s

# 3)
single_areas = {
    # Medial temporal lobe
    "Hippocampus_bi_pct"     : ("048", "049"),
    "Entorhinal_bi_pct"      : ("117", "118"),
    "Parahippocampal_bi_pct" : ("171", "172"),
    # Lateral/basal temporal
    "MTG_bi_pct"             : ("155", "156"),
    "ITG_bi_pct"             : ("133", "134"),
    "STG_bi_pct"             : ("201", "202"),
    "Fusiform_bi_pct"        : ("123", "124"),
    "TemporalPole_bi_pct"    : ("203", "204"),
    # Medial parietal
    "PosteriorCing_bi_pct"   : ("167", "168"),
    "Precuneus_bi_pct"       : ("169", "170"),
    # Posterior parietal/occipital (PCA axis)
    "SPL_bi_pct"             : ("199", "200"),
    "SOG_bi_pct"             : ("197", "198"),
    "MOG_bi_pct"             : ("145", "146"),
    "IOG_bi_pct"             : ("129", "130"),
    # Ventricles
    "LateralVentr_bi_pct"    : ("052", "053"),
    "InfLatVentr_bi_pct"     : ("050", "051"),
}

X_single = pd.DataFrame({name: bilateral(r, l, name) for name, (r,l) in␣
 ↪single_areas.items()})

# 4)     Ash
def sum_cols(df_block: pd.DataFrame, names: list[str], out: str) -> pd.Series:
    s = pd.Series(0.0, index=df_block.index, dtype=float)
    for n in names:
        if n in df_block.columns:
            s = s.add(pd.to_numeric(df_block[n], errors="coerce").fillna(0),␣
 ↪fill_value=0)
    s.replace([np.inf,-np.inf], np.nan, inplace=True)
    s.name = out
    return s

composites = {
```

```
    "MTL_total_pct"          :␣
 ↪["Hippocampus_bi_pct","Entorhinal_bi_pct","Parahippocampal_bi_pct"],
    "Temporal_lateral_pct"   :␣
 ↪["MTG_bi_pct","ITG_bi_pct","STG_bi_pct","Fusiform_bi_pct","TemporalPole_bi_pct"],
    "Medial_parietal_pct"    : ["PosteriorCing_bi_pct","Precuneus_bi_pct"],
    "Posterior_total_pct"    :␣
 ↪["SPL_bi_pct","SOG_bi_pct","MOG_bi_pct","IOG_bi_pct"],
    "Ventricles_total_pct"   : ["LateralVentr_bi_pct","InfLatVentr_bi_pct"],
}
X_comp = pd.DataFrame({out: sum_cols(X_single, names, out) for out, names in␣
 ↪composites.items()})
```

[25]:
```
# 5)
if "age_at_scan_date" in df.columns:
    age_series = pd.to_numeric(df["age_at_scan_date"], errors="coerce").
 ↪rename("Age_years")
elif "age_years" in df.columns:
    age_series = pd.to_numeric(df["age_years"], errors="coerce").
 ↪rename("Age_years")
else:
    age_series = None
```

[26]:
```
# 6)    " "    +     +
to_plot = pd.concat([X_comp, X_single], axis=1)
if age_series is not None:
    to_plot = pd.concat([to_plot, age_series], axis=1)
```

[27]:
```
# ======  " "  +      y   +      ======
import numpy as np, pandas as pd, matplotlib.pyplot as plt, re
from pathlib import Path
from math import ceil

OUTP = Path("surv_out") / "violin_ad_signature"; OUTP.mkdir(parents=True,␣
 ↪exist_ok=True)
GROUPS = ["AD","Mixed","Non-AD"]

#    to_plot   +   + Age   df["diag3"]
assert "to_plot" in globals(), "to_plot      / /     to_plot"
assert "diag3" in df.columns, "df   diag3   "

#
def pretty(col: str) -> str:
    if col.endswith("__pct"):
```

```
        return col.replace("Brain_","").replace("__pct","").replace("_"," ") +␣
 ↪" (%)"
    return col.replace("_"," ")

#  "  " y   1-99   + 10%
def feature_ylim(series_all_groups: list[np.ndarray], pad=0.10):
    vals = np.concatenate([v for v in series_all_groups if v.size>0])
    if vals.size == 0:
        return (0.0,1.0)
    lo, hi = np.percentile(vals, [1, 99])
    rng = hi - lo if hi > lo else max(1.0, hi)  #
    return float(lo - pad*rng), float(hi + pad*rng)

#
def draw_quantiles(ax, data, xpositions):
    for x, v in zip(xpositions, data):
        if v.size == 0:
            continue
        q1, med, q3 = np.percentile(v, [25,50,75])
        ax.hlines([q1,q3], x-0.12, x+0.12, colors="#444", lw=1, alpha=0.9)
        ax.hlines(med, x-0.18, x+0.18, colors="#d62728", lw=1.8, alpha=0.9)

# -- A)      y      --
rng = np.random.default_rng(42)
saved = []
for col in to_plot.columns:
    fig = plt.figure(figsize=(4.2, 4.0)); ax = plt.gca()

    data = [pd.to_numeric(to_plot.loc[df["diag3"]==g, col], errors="coerce").
 ↪dropna().values
            for g in GROUPS]
    non_empty = [(j, v) for j, v in enumerate(data, start=1) if v.size]
    if not non_empty:
        plt.close(); continue

    #
    vals_list = [v for _, v in non_empty]
    ax.violinplot(vals_list, showextrema=False, widths=0.8, bw_method=0.2)

    #   jitter
    for j, v in non_empty:
        xj = rng.normal(loc=j, scale=0.06, size=v.size)
        ax.plot(xj, v, 'o', ms=2.8, alpha=0.55, color="#1f77b4")

    #   /
    draw_quantiles(ax, vals_list, [j for j,_ in non_empty])
```

```
    #   y
    ymin, ymax = feature_ylim(vals_list, pad=0.10)
    ax.set_ylim(ymin, ymax)

    ax.set_xticks([j for j,_ in non_empty])
    ax.set_xticklabels([GROUPS[j-1] for j,_ in non_empty], fontsize=9)
    ax.set_title(pretty(col), fontsize=10)
    ax.grid(True, axis='y', alpha=0.3)

    out_png = OUTP / f"violin_autoscale_{re.sub('[^0-9A-Za-z]+','_', col)}.png"
    plt.savefig(out_png, dpi=300, bbox_inches="tight")
    plt.close(fig); saved.append(out_png)

print(f"[A] Saved autoscale figs: {len(saved)} images to {OUTP}")
```

```
[A] Saved autoscale figs: 21 images to surv_out/violin_ad_signature
```

[28]:
```
# ============================================================
# Ash           ROI  / / /  +   +
#   surv_out/panels/panel_violin_dots.png / panel_violin_mmse.png / panel_stats.
 ↪xlsx
#   matplotlib/scipy/statsmodels   scikit-posthocs
# ============================================================
import numpy as np, pandas as pd, re, sys
import matplotlib.pyplot as plt
from pathlib import Path
from math import ceil
from scipy import stats
from statsmodels.stats.multitest import multipletests

PAN_OUT = Path("surv_out") / "panels"; PAN_OUT.mkdir(parents=True,␣
 ↪exist_ok=True)
DATA_FALLBACK = Path("surv_out") / "survival_dataset.csv"

def log(msg):
    print("[PAN]", msg)

# ---------- 0)   df ----------
if "df" not in globals():
    if DATA_FALLBACK.exists():
        log(f"df       {DATA_FALLBACK}");
        df = pd.read_csv(DATA_FALLBACK, low_memory=False)
    else:
        raise RuntimeError("   df    surv_out/survival_dataset.csv ")

# ---------- 1)     ----------
```

```
def map3(x:str)->str:
    s=str(x)
    if "Mixed" in s or s.strip().upper()=="MIXED": return "Mixed"
    if "Alzheimer" in s or s.strip().upper() in {"AD","ALZHEIMER'S␣
 ↪DISEASE","ALZHEIMER'S DISEASE","AD"}: return "AD"
    return "Non-AD"

if "diag3" in df.columns:
    grp_col = "diag3"; df["diag3"] = df[grp_col].astype(str).map(map3)
elif "target_diag" in df.columns:
    grp_col = "target_diag"; df["diag3"] = df[grp_col].astype(str).map(map3)
elif "diag_group3" in df.columns:
    grp_col = "diag_group3"; df["diag3"] = df[grp_col].astype(str).map(map3)
else:
    raise KeyError("    target_diag/diag3/diag_group3 ")

GROUPS = ["AD","Mixed","Non-AD"]
df = df[df["diag3"].isin(GROUPS)].reset_index(drop=True)
log(f"   \n{df['diag3'].value_counts().to_string()}")

# ---------- 2)   '__pct'    +Total      ----------
def get_pct_by_code(code:str):
    """  code        __pct   ROI/Total*100 """
    base = next((c for c in df.columns if c.startswith(f"{code}_")), None)
    if base is None:
        return None
    pct_col = base + "__pct"
    if pct_col in df.columns:
        return pd.to_numeric(df[pct_col], errors="coerce")
    if "Total" in df.columns:
        num = pd.to_numeric(df[base], errors="coerce")
        den = pd.to_numeric(df["Total"], errors="coerce").replace(0, np.nan)
        return (num/den) * 100.0
    return None

def bilateral(code_R, code_L, outname):
    r = get_pct_by_code(code_R); l = get_pct_by_code(code_L)
    if r is None and l is None:
        return pd.Series(np.nan, index=df.index, name=outname)
    if r is None: r = pd.Series(0.0, index=df.index)
    if l is None: l = pd.Series(0.0, index=df.index)
    s = (r.fillna(0)+l.fillna(0)).replace([np.inf,-np.inf], np.nan); s.
 ↪name=outname; return s

#     Ash
single_defs = {
    # MTL
```

```
    "Hippocampus_bi_pct"     : ("048","049"),
    "Entorhinal_bi_pct"      : ("117","118"),
    "Parahippocampal_bi_pct" : ("171","172"),
    # Lateral/basal temporal
    "MTG_bi_pct"             : ("155","156"),
    "ITG_bi_pct"             : ("133","134"),
    "STG_bi_pct"             : ("201","202"),
    "Fusiform_bi_pct"        : ("123","124"),
    "TemporalPole_bi_pct"    : ("203","204"),
    # Medial parietal
    "PosteriorCing_bi_pct"   : ("167","168"),
    "Precuneus_bi_pct"       : ("169","170"),
    # Posterior axis
    "SPL_bi_pct"             : ("199","200"),
    "SOG_bi_pct"             : ("197","198"),
    "MOG_bi_pct"             : ("145","146"),
    "IOG_bi_pct"             : ("129","130"),
    # Ventricles
    "LateralVentr_bi_pct"    : ("052","053"),
    "InfLatVentr_bi_pct"     : ("050","051"),
}
singles = {}
for name,(r,l) in single_defs.items():
    s = bilateral(r,l,name); singles[name] = s

X_single = pd.DataFrame(singles)
present_single = [c for c in X_single.columns if X_single[c].notna().any()]
log(f"     {len(present_single)}")

#
def sum_cols(df_block, cols, out):
    s = pd.Series(0.0, index=df_block.index)
    for c in cols:
        if c in df_block.columns:
            s = s.add(pd.to_numeric(df_block[c], errors="coerce").fillna(0),␣
 ↪fill_value=0)
    s.replace([np.inf,-np.inf], np.nan, inplace=True); s.name = out; return s

composites = {
    "MTL_total_pct"        :␣
 ↪["Hippocampus_bi_pct","Entorhinal_bi_pct","Parahippocampal_bi_pct"],
    "Temporal_lateral_total_pct" :␣
 ↪["MTG_bi_pct","ITG_bi_pct","STG_bi_pct","Fusiform_bi_pct","TemporalPole_bi_pct"],
    "Medial_parietal_total_pct"  : ["PosteriorCing_bi_pct","Precuneus_bi_pct"],
    "Posterior_total_pct"        :␣
 ↪["SPL_bi_pct","SOG_bi_pct","MOG_bi_pct","IOG_bi_pct"],
    "Ventricles_total_pct"       : ["LateralVentr_bi_pct","InfLatVentr_bi_pct"],
```

```
}
X_comp = pd.DataFrame({out: sum_cols(X_single, cols, out) for out, cols in␣
 ↪composites.items()})
present_comp = [c for c in X_comp.columns if X_comp[c].notna().any()]
log(f"   {len(present_comp)}")

# -------- 2)        +      --------

def pct_or_raw_sum(patterns:list[str], outname:str):

    """

        patterns

    -    __pct

    -        Total    → (sum(raw)/Total)*100

    -    None

    """

    #   __pct

    pct_cols = []

    for p in patterns:

        pct_cols += [c for c in df.columns if re.match(p + r".*__pct$", str(c))]

    if pct_cols:

        s = df[pct_cols].apply(pd.to_numeric, errors="coerce").sum(axis=1)

        s.replace([np.inf,-np.inf], np.nan, inplace=True)

        s.name = outname

        return s

    #

    raw_cols = []

    for p in patterns:
```

```
        raw_cols += [c for c in df.columns if re.match(p + r".*$", str(c)) and␣
↪not str(c).endswith("__pct")]

    raw_cols = list(dict.fromkeys(raw_cols))  #

    if raw_cols and "Total" in df.columns:

        raw_sum = df[raw_cols].apply(pd.to_numeric, errors="coerce").sum(axis=1)

        total   = pd.to_numeric(df["Total"], errors="coerce").replace(0, np.nan)

        s = (raw_sum / total) * 100.0

        s.replace([np.inf,-np.inf], np.nan, inplace=True)

        s.name = outname

        return s

    return None

#     Brain_Left_0_Frontal_0/1/2... Right

LOBE_PATTERNS = {

    "Frontal_lobe_pct":   [r"^Brain_Left_0_Frontal_", ␣
↪r"^Brain_Right_0_Frontal_"],

    "Parietal_lobe_pct":  [r"^Brain_Left_2_Parietal_",␣
↪r"^Brain_Right_2_Parietal_"],

    "Occipital_lobe_pct":␣
↪[r"^Brain_Left_3_Occipital_",r"^Brain_Right_3_Occipital_"],

    "Temporal_lobe_pct":  [r"^Brain_Left_1_Temporal_",␣
↪r"^Brain_Right_1_Temporal_"],

    # Cerebellum / Brainstem      Brain_*

    "Cerebellum_pct": [

        r"^039_Right Cerebellum Exterior", r"^040_Left Cerebellum Exterior",

        r"^041_Right Cerebellum White Matter", r"^042_Left Cerebellum White␣
↪Matter",
```

```
        r"^072_Cerebellar Vermal Lobules I-V", r"^073_Cerebellar Vermal Lobules␣
 ↪VI-VII",

        r"^074_Cerebellar Vermal Lobules VIII-X"

    ],

    "Brainstem_pct": [r"^035_Pons", r"^036_Brain Stem"]

}

lobes_new = {}

for name, pats in LOBE_PATTERNS.items():

    if name not in df.columns:

        s = pct_or_raw_sum(pats, name)

        if s is not None:

            lobes_new[name] = s

lobes_df = pd.DataFrame(lobes_new)

if not lobes_df.empty:

    df = pd.concat([df, lobes_df], axis=1)

#   to_plot / FEATURES    to_plot

to_plot = pd.concat([X_comp, X_single, lobes_df], axis=1)

FEATURES = [c for c in to_plot.columns if to_plot[c].notna().any()]

print("[PAN]    /   ", len(FEATURES))

# -------- 3)   desc + KW + posthoc   to_plot  --------

desc_rows = []

stat_rows = []

try:

    import scikit_posthocs as sp
```

```
    HAVE_DUNN = True
except Exception:
    HAVE_DUNN = False
    print("[PAN] scikit-posthocs     Mann-Whitney + Holm ")
for feat in FEATURES:
    #
    for g in GROUPS:
        mask = (df["diag3"]==g) & to_plot[feat].notna()
        v = pd.to_numeric(to_plot.loc[mask, feat], errors="coerce").dropna().
 ↪values
        if v.size:
            q25,q50,q75 = np.percentile(v,[25,50,75])
            desc_rows.append({"feature":feat,"group":g,"n":int(v.size),
                              "median":q50,"IQR":q75-q25,"mean":v.mean(),"sd":v.
 ↪std(ddof=1)})
    #
    arrays = [pd.to_numeric(to_plot.loc[(df["diag3"]==g) & to_plot[feat].
 ↪notna(), feat], errors="coerce").dropna().values
              for g in GROUPS]
    if any(a.size==0 for a in arrays):
        stat_rows.append({"feature":feat,"kw_p":np.nan,"AD_vs_Mixed":np.
 ↪nan,"AD_vs_NonAD":np.nan,"Mixed_vs_NonAD":np.nan})
        continue
    kw = stats.kruskal(*arrays, nan_policy="omit")
    row = {"feature":feat,"kw_p":kw.pvalue}
    if HAVE_DUNN:
```

```
        pmat = sp.posthoc_dunn(arrays, p_adjust="holm")
        row.update({"AD_vs_Mixed": float(pmat.values[0,1]),
                    "AD_vs_NonAD": float(pmat.values[0,2]),
                    "Mixed_vs_NonAD": float(pmat.values[1,2])})
    else:
        pairs=[(0,1),(0,2),(1,2)]
        p=[]
        for i,j in pairs:
            _, pv = stats.mannwhitneyu(arrays[i], arrays[j],␣
 ↪alternative="two-sided")
            p.append(pv)
        _, p_adj, _, _ = multipletests(p, method="holm")
        row.update({"AD_vs_Mixed": p_adj[0], "AD_vs_NonAD": p_adj[1],␣
 ↪"Mixed_vs_NonAD": p_adj[2]})
    stat_rows.append(row)
desc_df = pd.DataFrame(desc_rows)
stat_df = pd.DataFrame(stat_rows)
with pd.ExcelWriter(PAN_OUT/"panel_stats.xlsx", engine="xlsxwriter") as xw:
    desc_df.to_excel(xw, sheet_name="desc", index=False)
    stat_df.to_excel(xw, sheet_name="posthoc", index=False)
print("[PAN]    ", PAN_OUT/"panel_stats.xlsx")
```

```
[PAN]
diag3
AD        1992
Non-AD     673
Mixed      645
```

```
[PAN]       16
[PAN]    5
[PAN]    /    27
[PAN] scikit-posthocs     Mann-Whitney + Holm
[PAN]     surv_out/panels/panel_stats.xlsx
```

[29]:
```
# ============================================================
#       +   y +  / MMSE
#   surv_out/panels_grouped/panel_all_groups_dots.png
#       surv_out/panels_grouped/panel_all_groups_mmse.png
# ============================================================
import numpy as np, pandas as pd, matplotlib.pyplot as plt, re
from pathlib import Path
from math import ceil
from scipy import stats
from statsmodels.stats.multitest import multipletests

GOUT = Path("surv_out")/"panels_grouped"; GOUT.mkdir(parents=True,␣
 ↪exist_ok=True)

# ---------- 0)    ----------
#
if "diag3" not in df.columns:
    if "target_diag" in df.columns:
        df["diag3"] = df["target_diag"]
    elif "diag_group3" in df.columns:
        df["diag3"] = df["diag_group3"]
    else:
        raise KeyError("    diag3/target_diag/diag_group3 ")

def norm3(x:str)->str:
    s=str(x)
    if "Mixed" in s or s.strip().upper()=="MIXED": return "Mixed"
    if "Alzheimer" in s or s.strip().upper() in {"AD","ALZHEIMER'S␣
 ↪DISEASE","ALZHEIMER'S DISEASE","AD"}: return "AD"
    return "Non-AD"

df = df.copy()
df["diag3"] = df["diag3"].astype(str).map(norm3)
GROUPS = ["AD","Mixed","Non-AD"]

#
FEATURES = [c for c in to_plot.columns if to_plot[c].notna().any()]
assert len(FEATURES)>0, "to_plot     "

# MMSE
```

```
MMSE_CANDS =␣
 ↪["Pre_Mini_Mental_Total","Post_Mini_Mental_Total","MMSE","mmse","MMSE_plot"]
mmse_col = next((c for c in MMSE_CANDS if c in df.columns), None)
df["MMSE_plot"] = pd.to_numeric(df[mmse_col], errors="coerce") if mmse_col else␣
 ↪np.nan
mmse_avail = df["MMSE_plot"].notna().any()

# ---------- 1)    p      Kruskal +    ----------
if "stat_df" not in globals():
    try:
        import scikit_posthocs as sp
        HAVE_DUNN = True
    except Exception:
        HAVE_DUNN = False
        print("[Panel-All] scikit-posthocs     Mann-Whitney + Holm   ")
    rows=[]
    for feat in FEATURES:
        arrays = [pd.to_numeric(to_plot.loc[(df["diag3"]==g)&to_plot[feat].
 ↪notna(), feat],
                                errors="coerce").dropna().values for g in␣
 ↪GROUPS]
        if any(a.size==0 for a in arrays):
            rows.append({"feature":feat,"kw_p":np.nan,"AD_vs_Mixed":np.
 ↪nan,"AD_vs_NonAD":np.nan,"Mixed_vs_NonAD":np.nan})
            continue
        kw = stats.kruskal(*arrays, nan_policy="omit")
        rec = {"feature":feat, "kw_p":kw.pvalue}
        if HAVE_DUNN:
            pmat = sp.posthoc_dunn(arrays, p_adjust="holm")
            rec.update({"AD_vs_Mixed": float(pmat.values[0,1]),
                        "AD_vs_NonAD": float(pmat.values[0,2]),
                        "Mixed_vs_NonAD": float(pmat.values[1,2])})
        else:
            pairs=[(0,1),(0,2),(1,2)]; p=[]
            for i,j in pairs:
                _, pv = stats.
 ↪mannwhitneyu(arrays[i],arrays[j],alternative="two-sided")
                p.append(pv)
            _, p_adj, _, _ = multipletests(p, method="holm")
            rec.update({"AD_vs_Mixed": p_adj[0], "AD_vs_NonAD": p_adj[1],␣
 ↪"Mixed_vs_NonAD": p_adj[2]})
        rows.append(rec)
    stat_df = pd.DataFrame(rows)

def star(p):
    if pd.isna(p): return ""
```

```
    if p<0.001: return "***"
    if p<0.01 : return "**"
    if p<0.05 : return "*"
    return ""
```

[30]:
```
# ---------- 2)  "  "    p95     y  ----------
#
MAG = {}
for f in FEATURES:
    v = pd.to_numeric(to_plot[f], errors="coerce").dropna().values
    MAG[f] = float(np.percentile(v,95)) if v.size else 0.0

ratio_factor = 2.0   #     2
s_feats = [k for k,_ in sorted(MAG.items(), key=lambda x:x[1])]  #
groups = []
cur, cur_min, cur_max = [], None, None
for f in s_feats:
    p = MAG[f]
    if cur_min is None:
        cur=[f]; cur_min=cur_max=p
        continue
    new_min, new_max = min(cur_min,p), max(cur_max,p)
    if (new_max / max(new_min,1e-9)) <= ratio_factor:
        cur.append(f); cur_min,cur_max=new_min,new_max
    else:
        groups.append(cur); cur=[f]; cur_min=cur_max=p
if cur: groups.append(cur)

print(f"[Panel-All]    {len(groups)}  ", [len(g) for g in groups])
ordered_features = [f for g in groups for f in g]  #

#    y    (1-99  + IQR  )
def robust_ylim(values_list, pad=0.08, iqr_cap=8.0):
    vals = np.concatenate([v for v in values_list if v.size])
    p1,p99 = np.percentile(vals,[1,99]); med = np.median(vals)
    q1,q3 = np.percentile(vals,[25,75]); iqr=max(q3-q1, 1e-9)
    lo = max(p1, med - iqr_cap*iqr)
    hi = min(p99, med + iqr_cap*iqr)
    rng = hi - lo if hi>lo else max(1.0, hi)
    return float(lo - pad*rng), float(hi + pad*rng)

group_ylim = {}
for g in groups:
    vals_all=[]
    for f in g:
        for lab in GROUPS:
```

```
            v = pd.to_numeric(to_plot.loc[(df["diag3"]==lab)&to_plot[f].
↪notna(), f], errors="coerce").dropna().values
            if v.size: vals_all.append(v)
    group_ylim[tuple(g)] = robust_ylim(vals_all, pad=0.12, iqr_cap=12.0)
```

```
[Panel-All]    5   [6, 9, 4, 7, 1]
```

[31]:
```
from matplotlib.colors import Normalize

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

from math import ceil

from pathlib import Path

def draw_one_big_panel(mmse: bool=False,

                       out_name: str="panel_all_groups_dots.png",

                       ncols: int=6,

                       pad_top_ax: float=0.02,          #        padding

                       ylayer_step_ax: float=0.045,     #

                       show_connect_line: bool=True,    #

                       show_overflow_tag: bool=True,    #    "+N"

                       dpi: int=300) -> Path:

    """

           ordered_features    y  group_ylim

                    MMSE

        "   "  +N

    """

    assert ("ordered_features" in globals()
            and "group_of" in globals()
            and "group_ylim" in globals()), \
            "  ordered_features / group_of / group_ylim     y  "
```

```
   GOUT = Path("surv_out")/"panels_grouped"; GOUT.mkdir(parents=True,␣
↪exist_ok=True)

   n = len(ordered_features)

   nrows = int(np.ceil(n / ncols))

   fig, axes = plt.subplots(nrows, ncols, figsize=(ncols*5.2, nrows*4.6),␣
↪squeeze=False)

   plt.subplots_adjust(wspace=0.48, hspace=0.62)

   # colormap for MMSE

   if mmse and mmse_avail:

       cmap = plt.get_cmap("RdYlGn"); norm = Normalize(vmin=0, vmax=30)

   #      x=1/2/3

   def data_to_axes_x(ax, x_data):

       inv = ax.transAxes.inverted()

       x_pix = ax.transData.transform((x_data, 0))[0]

       x_ax  = inv.transform((x_pix, 0))[0]

       return x_ax

   #

   y_layers_ax = [1.00 + pad_top_ax + i*ylayer_step_ax for i in range(3)]

   pairs  = {"AD_vs_Mixed": (1,2), "AD_vs_NonAD": (1,3), "Mixed_vs_NonAD":␣
↪(2,3)}

   order  = ["AD_vs_Mixed","AD_vs_NonAD","Mixed_vs_NonAD"]

   #

   def title_of(col):

       try:

           return LONG_NAME.get(col, col.replace("__pct"," (%)").replace("_","␣
↪"))
```

```
        except NameError:
            return col.replace("__pct"," (%)").replace("_"," ")
    for i, feat in enumerate(ordered_features):
        r, c = divmod(i, ncols)
        ax = axes[r, c]
        data = [pd.to_numeric(to_plot.loc[(df["diag3"]==lab)&to_plot[feat].
↪notna(), feat],
                              errors="coerce").dropna().values for lab in␣
↪GROUPS]
        non_empty = [(j, v) for j, v in enumerate(data, start=1) if v.size]
        if not non_empty:
            ax.axis("off"); continue
        vals = [v for _, v in non_empty]
        # violin
        ax.violinplot(vals, showextrema=False, widths=0.82, bw_method=0.25)
        # dots
        for j, _ in non_empty:
            mask = (df["diag3"]==GROUPS[j-1]) & to_plot[feat].notna()
            yv   = pd.to_numeric(to_plot.loc[mask, feat], errors="coerce").
↪values
            x    = np.random.default_rng(42).normal(loc=j, scale=0.06, size=yv.
↪size)
            if mmse and mmse_avail:
                mm   = df.loc[mask, "MMSE_plot"].values
                colors = plt.get_cmap("RdYlGn")(np.clip(mm/30.0, 0, 1))
```

```
                ax.scatter(x, yv, c=colors, s=10, alpha=0.65, edgecolors='none')
            else:
                ax.plot(x, yv, 'o', ms=3, alpha=0.55, color="#1f77b4")
        #   / IQR
        for j, arr in non_empty:
            q1, med, q3 = np.percentile(arr,[25,50,75])
            ax.hlines([q1,q3], j-0.12, j+0.12, colors="#444", lw=1, alpha=0.9)
            ax.hlines(med,     j-0.18, j+0.18, colors="#d62728", lw=1.8,␣
↪alpha=0.9)
        #    y
        ylo, yhi = group_ylim[group_of[feat]]
        ax.set_ylim(ylo, yhi)
        #         [ylo, yhi]
        if show_overflow_tag:
            all_vals = pd.to_numeric(to_plot[feat], errors="coerce").dropna().
↪values
            n_upper  = int((all_vals > yhi).sum())
            if n_upper > 0:
                ax.text(0.98, 1.00 + pad_top_ax*0.6, f"+{n_upper}",␣
↪transform=ax.transAxes,
                        ha="right", va="bottom", fontsize=9, color="#555",␣
↪clip_on=False)
        #
        srow = stat_df.loc[stat_df["feature"]==feat]
        for idx, key in enumerate(order):
            a, b = pairs[key]
```

```
            p = float(srow[key].iloc[0]) if (len(srow) and key in srow.columns)␣
↪else np.nan

            mark = "***" if p<0.001 else ("**" if p<0.01 else ("*" if p<0.05␣
↪else ""))

            if not mark:

                continue

            y_ax = y_layers_ax[idx]

            ax_a = data_to_axes_x(ax, a)

            ax_b = data_to_axes_x(ax, b)

            x_mid = (ax_a + ax_b) / 2.0

            if show_connect_line:

                ax.plot([ax_a, ax_a, ax_b, ax_b], [y_ax-0.01, y_ax, y_ax,␣
↪y_ax-0.01],

                        transform=ax.transAxes, lw=1.0, c="#444", clip_on=False)

            ax.text(x_mid, y_ax + 0.005, mark, transform=ax.transAxes,

                    ha="center", va="bottom", fontsize=10, color="#d62728",␣
↪clip_on=False)

        #

        ax.set_title(title_of(feat), fontsize=10)

        ax.set_xticks([j for j,_ in non_empty]); ax.set_xticklabels(GROUPS,␣
↪fontsize=9)

        ax.set_ylabel("%", fontsize=9)

        ax.grid(True, axis='y', alpha=0.3)

    #

    for k in range(len(ordered_features), nrows*ncols):

        rr, cc = divmod(k, ncols)
```

```
        axes[rr, cc].axis("off")

    out_path = GOUT / out_name

    plt.savefig(out_path, dpi=dpi, bbox_inches="tight")

    plt.close(fig)

    print("[Panel-All] Saved:", out_path)

    return out_path
```

[32]:
```
# ============== 3.       & MMSE    ==============
#    one-way ANOVA + Tukey HSD

#  -   AD / Mixed / Non-AD

#  -   Tukey HSD      p    *, **, ***

#use_parametric = True  # True: ANOVA+Tukey False: Kruskal+  Dunn

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

from matplotlib.colors import Normalize

from math import ceil

from pathlib import Path

PANF = Path("surv_out")/"panels_final"; PANF.mkdir(parents=True, exist_ok=True)

# 3.1       to_plot      wmh_icv_pct

FEATURES_PANEL = [c for c in to_plot.columns if to_plot[c].notna().any()]

assert len(FEATURES_PANEL)>0, "to_plot    "

# 3.2   MMSE

GROUPS = ["AD","Mixed","Non-AD"]

df = df[df["diag3"].isin(GROUPS)].reset_index(drop=True)
```

```
mmse_col = next((c for c in␣
 ↪["Pre_Mini_Mental_Total","Post_Mini_Mental_Total","MMSE","mmse","MMSE_plot"]␣
 ↪if c in df.columns), None)

df["MMSE_plot"] = pd.to_numeric(df[mmse_col], errors="coerce") if mmse_col else␣
 ↪np.nan

mmse_avail = df["MMSE_plot"].notna().any()

def title_of(col):

    try:

        return LONG_NAME.get(col, col.replace("__pct"," (%)").replace("_"," "))

    except NameError:

        return col.replace("__pct"," (%)").replace("_"," ")

# 3.3        / MMSE

def draw_two_page_panel(mmse=False, out_name="panel_roi_dist.png", ncols=6,␣
 ↪max_n_per_group=400):

    feats = FEATURES_PANEL

    n = len(feats); nrows = ceil(n/ncols)

    fig, axes = plt.subplots(nrows, ncols, figsize=(ncols*5.2, nrows*4.6),␣
 ↪squeeze=False)

    plt.subplots_adjust(wspace=0.45, hspace=0.60)

    if mmse and mmse_avail:

        cmap = plt.get_cmap("RdYlGn"); norm = Normalize(vmin=0, vmax=30)

        cbar_added = False

    for i, feat in enumerate(feats):

        r,c = divmod(i, ncols); ax = axes[r,c]

        data = [pd.to_numeric(to_plot.loc[(df["diag3"]==lab)&to_plot[feat].
 ↪notna(), feat],
```

```
                            errors="coerce").dropna().values for lab in␣
↪GROUPS]

        non_empty = [(j,v) for j,v in enumerate(data, start=1) if v.size]

        if not non_empty:

            ax.axis("off"); continue

        #   y  1-99 +IQR

        allv = np.concatenate([v for _,v in non_empty])

        p1,p99 = np.percentile(allv,[1,99]); med=np.median(allv)

        q1,q3   = np.percentile(allv,[25,75]); iqr=max(q3-q1,1e-9)

        ylo = max(p1, med - 8.0*iqr); yhi = min(p99, med + 8.0*iqr)

        pad = 0.10*(yhi-ylo); ylo -= pad; yhi += pad

        # violin

        ax.violinplot([v for _,v in non_empty], showextrema=False, widths=0.82,␣
↪bw_method=0.25)

        # dots

        for j,_ in non_empty:

            mask = (df["diag3"]==GROUPS[j-1]) & to_plot[feat].notna()

            yv = pd.to_numeric(to_plot.loc[mask, feat], errors="coerce").values

            x  = np.random.default_rng(42).normal(loc=j, scale=0.06,␣
↪size=len(yv))

            #      max_n_per_group

            if max_n_per_group and len(yv)>max_n_per_group:

                sel = np.random.default_rng(0).choice(len(yv),␣
↪size=max_n_per_group, replace=False)

                yv = yv[sel]; x = x[sel]
```

```
                mm = df.loc[mask, "MMSE_plot"].values[sel] if mmse and␣
↪mmse_avail else None

            else:

                mm = df.loc[mask, "MMSE_plot"].values if mmse and mmse_avail␣
↪else None

            if mmse and mmse_avail:

                sc = ax.scatter(x, yv, c=np.clip((mm/30.0),0,1), cmap="RdYlGn",

                                s=10, alpha=0.65, edgecolors='none', vmin=0,␣
↪vmax=1)

                if not cbar_added:

                    cbar = fig.colorbar(sc, ax=axes.ravel().tolist(),␣
↪fraction=0.02, pad=0.01)

                    cbar.set_label("MMSE (0-30)")

                    cbar_added = True

            else:

                ax.plot(x, yv, 'o', ms=3, alpha=0.55, color="#1f77b4")

        #  /IQR

        for j, arr in non_empty:

            q1,med,q3 = np.percentile(arr,[25,50,75])

            ax.hlines([q1,q3], j-0.12, j+0.12, colors="#444", lw=1, alpha=0.9)

            ax.hlines(med,     j-0.18, j+0.18, colors="#d62728", lw=1.8,␣
↪alpha=0.9)

        ax.set_title(title_of(feat), fontsize=10)

        ax.set_xticks([j for j,_ in non_empty]); ax.set_xticklabels(GROUPS,␣
↪fontsize=9)

        ax.set_ylabel("%", fontsize=9); ax.set_ylim(ylo, yhi); ax.grid(True,␣
↪axis='y', alpha=0.3)
```

```
    for k in range(len(feats), nrows*ncols):

        rr,cc = divmod(k, ncols); axes[rr,cc].axis("off")

    out = PANF / out_name

    plt.savefig(out, dpi=300, bbox_inches="tight"); plt.close(fig)

    print("[Panel-Final] Saved:", out)

#  1

draw_two_page_panel(mmse=False, out_name="panel_roi_dist.png",  ncols=6,␣
 ↪max_n_per_group=400)

#  2 MMSE

draw_two_page_panel(mmse=True,  out_name="panel_roi_mmse_colored.png", ncols=6,␣
 ↪max_n_per_group=400)
```

```
[Panel-Final] Saved: surv_out/panels_final/panel_roi_dist.png
[Panel-Final] Saved: surv_out/panels_final/panel_roi_mmse_colored.png
```

[33]:
```
# ============== 4.   ANOVA/Tukey  Kruskal/Dunn +    +    ==============

from scipy import stats

from statsmodels.stats.multitest import multipletests

import statsmodels.api as sm

import statsmodels.formula.api as smf

from pathlib import Path

OUTS = Path("surv_out")/"analysis_out"; OUTS.mkdir(parents=True, exist_ok=True)

#   Cliff's delta   ² ANOVA

def cliffs_delta(x, y):

    x=np.asarray(x); y=np.asarray(y)

    nx,ny=len(x),len(y)
```

```
    gt = sum((xi > y).sum() for xi in x)

    lt = sum((xi < y).sum() for xi in x)

    return float((gt - lt)/(nx*ny))

def eta_squared_anova(model):

    anov = sm.stats.anova_lm(model, typ=2)

    ss_between = anov.loc["C(diag3)","sum_sq"]

    ss_total   = anov["sum_sq"].sum()

    return float(ss_between/ss_total) if ss_total>0 else np.nan

rows=[]

for feat in FEATURES_PANEL:

    #

    arr = [pd.to_numeric(to_plot.loc[(df["diag3"]==g)&to_plot[feat].notna(),␣
 ↪feat], errors="coerce").dropna().values

           for g in GROUPS]

    #   /     Shapiro/Levene   Shapiro

    try_norm = all([stats.shapiro(a[:500]).pvalue>0.05 for a in arr if a.
 ↪size>=5])   #  500

    try_lev  = stats.levene(*arr, center="median").pvalue>0.05 if all([a.size>5␣
 ↪for a in arr]) else False

    method   = "ANOVA+Tukey" if (try_norm and try_lev) else "Kruskal+Dunn"

    if method=="ANOVA+Tukey":

        tmp = pd.DataFrame({"y": to_plot[feat], "diag3": df["diag3"]}).dropna()

        mod = smf.ols("y ~ C(diag3)", data=tmp).fit()

        eta2 = eta_squared_anova(mod)

        # Tukey HSD
```

```
        from statsmodels.stats.multicomp import pairwise_tukeyhsd

        tk = pairwise_tukeyhsd(endog=tmp["y"].values, groups=tmp["diag3"].
↪values, alpha=0.05)

        #     p

        def tukey_p(a,b):

            #   a-b

            mask =(
                ((tk.groupsunique[tk._multicomp.pairindices[0]]==a) & (tk.
↪groupsunique[tk._multicomp.pairindices[1]]==b))
                |
                ((tk.groupsunique[tk._multicomp.pairindices[0]]==b) & (tk.
↪groupsunique[tk._multicomp.pairindices[1]]==a))
            )

            # statsmodels        summary

            for row in tk.summary().data[1:]:

                if set(row[:2])==set([a,b]):

                    return float(row[-1])

            return np.nan

        p_am = tukey_p("AD","Mixed"); p_an = tukey_p("AD","Non-AD"); p_mn =␣
↪tukey_p("Mixed","Non-AD")

        rows.append({"feature":feat,"method":method,"effect":
↪"eta2","effect_value":eta2,

                     "AD_vs_Mixed":p_am, "AD_vs_NonAD":p_an, "Mixed_vs_NonAD":
↪p_mn})

    else:

        # Kruskal + Dunn

        kw = stats.kruskal(*arr, nan_policy="omit").pvalue

        try:
```

```
            import scikit_posthocs as sp
            pmat = sp.posthoc_dunn(arr, p_adjust="fdr_bh")  #  BH
            p_am  = float(pmat.values[0,1])
            p_an  = float(pmat.values[0,2])
            p_mn  = float(pmat.values[1,2])
        except Exception:
            #   MW + BH
            pairs=[(0,1),(0,2),(1,2)]
            p_raw=[]
            for i,j in pairs:
                _, pv = stats.
↪mannwhitneyu(arr[i],arr[j],alternative="two-sided")
                p_raw.append(pv)
            _, p_bh, _, _ = multipletests(p_raw, method="fdr_bh")
            p_am, p_an, p_mn = p_bh
        #   Cliff's
        d_am = cliffs_delta(arr[0],arr[1]); d_an = cliffs_delta(arr[0],arr[2]);␣
↪d_mn = cliffs_delta(arr[1],arr[2])
        rows.append({"feature":feat,"method":method,"effect":
↪"cliffs_delta","effect_value":np.nan,
                     "AD_vs_Mixed":p_am, "AD_vs_NonAD":p_an, "Mixed_vs_NonAD":
↪p_mn,
                     "delta_AD_Mixed":d_am, "delta_AD_NonAD":d_an,␣
↪"delta_Mixed_NonAD":d_mn})
posthoc_effects = pd.DataFrame(rows)
posthoc_effects.to_csv(OUTS/"posthoc_effects.csv", index=False)
```

```
#

md = """# Statistical methods for group comparisons

- If normality and homoskedasticity hold → **ANOVA + Tukey's HSD**; effect size␣
 ↪reported as ** ²**.

- Otherwise → **Kruskal-Wallis + Dunn**. Pairwise p-values are **FDR (Benjamini-
Hochberg)** corrected.

- Plots display **violin + jitter**, with **median (red)** and **IQR (black)**.␣
 ↪Significance is shown as stars (* p<.05, ** p<.01, *** p<.001).

- For non-parametric comparisons, we report **Cliff's  ** as the effect size.

"""

(Path("surv_out")/"analysis_out"/"stats_methods.md").write_text(md,␣
 ↪encoding="utf-8")

print("[Stats] Saved:", OUTS/"posthoc_effects.csv", "|", OUTS/"stats_methods.
 ↪md")
```

```
[Stats] Saved: surv_out/analysis_out/posthoc_effects.csv |
surv_out/analysis_out/stats_methods.md
```

[34]:
```
# ============== 5.    SOP    ==============

from pathlib import Path

SOP = Path("surv_out")/"analysis_out"/"preprocessing_SOP.md"

mmse_rule = "- **MMSE**: take the score closest to the **scan date** per␣
 ↪subject; if `Pre` and `Post` both available, prefer `Pre`."

dedup_rule= "- **De-duplication**: per-scan one row (prefer `ScanID`; otherwise␣
 ↪`BrcId+Scan_Date`); keep the earliest or most complete record."

pct_rule  = "- **Percent volume**: `ROI_pct = ROI_volume / Total × 100`.␣
 ↪Bilateral ROIs are summed before percent; lobe-level composites sum all ROI%␣
 ↪in the lobe."

wmh_rule  = "- **WMH**: merge by `BrcId` from `WMH.xlsx` (columns `WMH(77)`,␣
 ↪`Intracranial-volume`). If multiple rows per `BrcId`, take **max** WMH and␣
 ↪**median** ICV; derive `wmh_icv_pct`, `wmh_log_icv`, `wmh_missing`."
```

```
vars_stage1 = list(X1.columns)

vars_stage2 = list(X2.columns)

md = f"""# Preprocessing SOP

## Labels

- Three-class diagnosis: **AD / Mixed / Non-AD** from `Final_Diagnosis` mapping.

- Stage-1 label (binary): **AD vs Others**; Stage-2 label: **Mixed vs Non-AD**␣
 ↪(within non-AD subset).

## De-duplication

{dedup_rule}

## MMSE policy

{mmse_rule}

## Variables

- **Stage-1 features**: {vars_stage1}

- **Stage-2 features**: {vars_stage2}

## Percent volumes

{pct_rule}

## WMH

{wmh_rule}

## Missing data

- Numerical: **median imputation** (fit inside fold); Categorical:␣
 ↪**most-frequent** + One-Hot (`handle_unknown='ignore'`).

## Cross-validation & thresholding

- 5-fold **grouped** CV by `BrcId`; **Stage-1 threshold** selected by inner CV␣
 ↪(objective: macro-F1); external evaluation on outer folds.

"""
```

```
SOP.write_text(md, encoding="utf-8")

print("[SOP] Saved:", SOP)
```

```
[SOP] Saved: surv_out/analysis_out/preprocessing_SOP.md
```

[35]:
```
# ============== 7. Two-stage overall evaluation + interpretation ==============

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

from pathlib import Path

from sklearn.metrics import (roc_auc_score, average_precision_score, roc_curve,

                             precision_recall_curve, confusion_matrix,␣
 ↪classification_report)

from sklearn.preprocessing import label_binarize

from sklearn.inspection import permutation_importance

OUT7 = Path("surv_out")/"analysis_out"; OUT7.mkdir(parents=True, exist_ok=True)

# 7.1           P1_all/P2_all

try:

    P1_all

except NameError:

    pipe1.fit(X1, y1); P1_all = pipe1.predict_proba(X1)[:,1]

try:

    X2_all = build_stage2_matrix(df.reset_index(drop=True), ref_cols=X2.columns)

    pipe2.fit(X2, y2); P2_all = pipe2.predict_proba(X2_all)[:,1]

except Exception as e:

    raise RuntimeError("  Stage-2     build_stage2_matrix  ") from e

P_AD    = P1_all
```

```
P_Mixed = (1.0 - P1_all)*P2_all

P_NonAD = (1.0 - P1_all)*(1.0 - P2_all)

prob_mat = np.c_[P_AD, P_Mixed, P_NonAD]

labels3 = ["AD","Mixed","Non-AD"]

y_true_col = "target_diag" if "target_diag" in df.columns else ("diag3" if␣
 ↪"diag3" in df.columns else None)

y_true3 = df[y_true_col].values

# 7.2    ROC/PR OVR

Y_bin = label_binarize(y_true3, classes=labels3)

macro_roc = roc_auc_score(Y_bin, prob_mat, average="macro", multi_class="ovr")

macro_pr  = average_precision_score(Y_bin, prob_mat, average="macro")

# 7.3  overall  OVR ROC + OVR PR + AD vs Others

fig, axes = plt.subplots(1,3, figsize=(14,4.2))

# ROC

for i,lab in enumerate(labels3):

    fpr,tpr,_ = roc_curve(Y_bin[:,i], prob_mat[:,i])

    axes[0].plot(fpr,tpr,label=lab)

axes[0].plot([0,1],[0,1],'k--',alpha=.3)

axes[0].set_title(f"OVR ROC (macro AUC={macro_roc:.3f})"); axes[0].
 ↪set_xlabel("FPR"); axes[0].set_ylabel("TPR"); axes[0].legend(); axes[0].
 ↪grid(alpha=.2)

# PR

for i,lab in enumerate(labels3):

    prec,rec,_ = precision_recall_curve(Y_bin[:,i], prob_mat[:,i])

    axes[1].plot(rec,prec,label=lab)
```

```
axes[1].set_title(f"OVR PR (macro AP={macro_pr:.3f})"); axes[1].
 ↪set_xlabel("Recall"); axes[1].set_ylabel("Precision"); axes[1].legend();␣
 ↪axes[1].grid(alpha=.2)

#   AD vs Others

from sklearn.calibration import calibration_curve

y_ad = (y_true3=="AD").astype(int)

cal_x,cal_y = calibration_curve(y_ad, P_AD, n_bins=10, strategy="quantile")

axes[2].plot([0,1],[0,1],'k--',alpha=.3)

axes[2].plot(cal_x,cal_y,'o-')

axes[2].set_title("Calibration: AD vs Others"); axes[2].set_xlabel("Predicted␣
 ↪P(AD)"); axes[2].set_ylabel("Observed AD"); axes[2].grid(alpha=.2)

plt.tight_layout()

plt.savefig(OUT7/"pipeline_overall_metrics.png", dpi=300); plt.close(fig)

# 7.4 Stage-1 & Stage-2        Stage-1  Stage-2

# Stage-1

prep1 = pipe1.named_steps["prep"]; clf1 = pipe1.named_steps["clf"]

X1_mat = prep1.transform(X1)

feat1_names = prep1.get_feature_names_out()

pi1 = permutation_importance(clf1, X1_mat, y1, n_repeats=10, random_state=0,␣
 ↪scoring="f1")

imp1 = pd.DataFrame({"stage":"stage1","feature":feat1_names,"perm_importance":
 ↪pi1.importances_mean})

# Stage-2

prep2 = pipe2.named_steps["prep"]; clf2 = pipe2.named_steps["clf"]

X2_mat = prep2.transform(X2)

feat2_names = prep2.get_feature_names_out()
```

```
pi2 = permutation_importance(clf2, X2_mat, y2, n_repeats=10, random_state=0,␣
 ↪scoring="f1")

imp2 = pd.DataFrame({"stage":"stage2","feature":feat2_names,"perm_importance":
 ↪pi2.importances_mean})

imp_all = pd.concat([imp1,imp2], ignore_index=True)

imp_all.to_csv(OUT7/"importances_stage1_stage2.csv", index=False)

# 7.5 shap_global.png  Stage-1     Top-20 + "age vs imaging"

#  "age" "imaging"

age_cols = [c for c in feat1_names if "age" in c.lower()]

imaging_cols = [c for c in feat1_names if ("__pct" in c.lower()) or ("Hipp" in␣
 ↪c or "Ventr" in c or "Temporal" in c)]

#    /

imaging_cols = list(set(list(imaging_cols) + [c for c in feat1_names if any(k␣
 ↪in c for k in␣
 ↪["MTL","Posterior","Parietal","Occipital","Temporal_lobe","Frontal_lobe"])]))

age_gain = float(imp1.loc[imp1["feature"].isin(age_cols), "perm_importance"].
 ↪sum())

img_gain = float(imp1.loc[imp1["feature"].isin(imaging_cols),␣
 ↪"perm_importance"].sum())

other_gain = float(imp1["perm_importance"].sum() - age_gain - img_gain)

top20 = imp1.sort_values("perm_importance", ascending=False).head(20)

fig,axes = plt.subplots(1,2, figsize=(12,4))

axes[0].barh(top20["feature"][::-1], top20["perm_importance"][::-1])

axes[0].set_title("Stage-1 permutation importance (Top-20)")

axes[0].set_xlabel("ΔF1 (mean, permutation)")

axes[0].grid(axis='x', alpha=.2)
```

```
axes[1].pie([img_gain, age_gain, max(other_gain,0.0)],␣
 ↪labels=["Imaging","Age","Other"], autopct="%.1f%%", startangle=90)

axes[1].set_title("Relative contributions (Stage-1)")

plt.tight_layout()

plt.savefig(OUT7/"shap_global.png", dpi=300); plt.close(fig)

print("[Overall] Saved:", OUT7/"pipeline_overall_metrics.png", "|", OUT7/
 ↪"shap_global.png", "|", OUT7/"importances_stage1_stage2.csv")
```

```
[Overall] Saved: surv_out/analysis_out/pipeline_overall_metrics.png |
surv_out/analysis_out/shap_global.png |
surv_out/analysis_out/importances_stage1_stage2.csv
```

[36]:
```
# ============== 8. Interaction / Multidimensional visual probes ==============

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

from pathlib import Path

from sklearn.linear_model import LogisticRegression

from mpl_toolkits.mplot3d import Axes3D  # noqa

OUT8 = Path("surv_out")/"analysis_out"/"interaction_plots"; OUT8.
 ↪mkdir(parents=True, exist_ok=True)

pairs_2d = [

    ("Ventricles_total_pct","MTL_total_pct"),

    ("LateralVentr_bi_pct","Posterior_total_pct"),

    ("wmh_icv_pct","Temporal_lateral_total_pct")  #    wmh_icv_pct

]

triples_3d = [

    ("Ventricles_total_pct","MTL_total_pct","wmh_icv_pct")

]

for a,b in pairs_2d:
```

```
    if a not in to_plot.columns or b not in to_plot.columns:

        continue

    tmp = pd.DataFrame({"a":to_plot[a], "b":to_plot[b], "y":
↪(df["diag3"]=="Mixed").astype(int)}).dropna()

    if tmp.empty: continue

    X = tmp[["a","b"]].values; y = tmp["y"].values

    lr = LogisticRegression().fit(X,y)

    xa = np.linspace(np.quantile(X[:,0], .01), np.quantile(X[:,0], .99), 200)

    xb = np.linspace(np.quantile(X[:,1], .01), np.quantile(X[:,1], .99), 200)

    XX,YY = np.meshgrid(xa,xb); ZZ = lr.predict_proba(np.c_[XX.ravel(),YY.
↪ravel()])[:,1].reshape(XX.shape)

    plt.figure(figsize=(5.0,4.2))

    cs = plt.contour(XX,YY,ZZ, levels=[0.25,0.5,0.75],␣
↪colors=["#aaa","#444","#aaa"])

    plt.clabel(cs, fmt={0.5:"0.5"}, inline=True, fontsize=8)

    for g,c in zip(["AD","Mixed","Non-AD"], ["#1f77b4","#d62728","#2ca02c"]):

        m = df["diag3"]==g

        plt.scatter(to_plot.loc[m,a], to_plot.loc[m,b], s=6, alpha=0.4,␣
↪label=g, c=c)

    plt.xlabel(a); plt.ylabel(b); plt.legend(markerscale=2, frameon=False)

    plt.title("Mixed boundary in 2D feature space")

    plt.tight_layout(); plt.savefig(OUT8/f"interact_2D_{a}_{b}.png", dpi=300);␣
↪plt.close()

for a,b,c in triples_3d:

    if any(col not in to_plot.columns for col in [a,b,c]):

        continue
```

```
    tmp = pd.DataFrame({"a":to_plot[a], "b":to_plot[b], "c":to_plot[c], "group":
↪df["diag3"]}).dropna()

    if tmp.empty: continue

    fig = plt.figure(figsize=(5.0,4.2)); ax = fig.add_subplot(111,␣
↪projection='3d')

    colors = {"AD":"#1f77b4","Mixed":"#d62728","Non-AD":"#2ca02c"}

    for g in ["AD","Mixed","Non-AD"]:

        d = tmp[tmp["group"]==g]

        ax.scatter(d["a"], d["b"], d["c"], s=8, alpha=0.4, c=colors[g], label=g)

    ax.set_xlabel(a); ax.set_ylabel(b); ax.set_zlabel(c); ax.legend(loc="upper␣
↪left")

    plt.tight_layout(); plt.savefig(OUT8/f"interact_3D_{a}_{b}_{c}.png",␣
↪dpi=300); plt.close()

print("[Interact] Saved to:", OUT8)

print("One-liner: Mixed separates best in (Ventricles_total_pct vs␣
↪MTL_total_pct) plane; "

      "WMH/ICV adds orthogonal separation when combined with posterior/temporal␣
↪composites.")
```

```
[Interact] Saved to: surv_out/analysis_out/interaction_plots
One-liner: Mixed separates best in (Ventricles_total_pct vs MTL_total_pct)
plane; WMH/ICV adds orthogonal separation when combined with posterior/temporal
composites.
```

[37]:
```
import os

import json

import time

import datetime as dt

import pandas as pd
```

```
import matplotlib.pyplot as plt

import matplotlib.image as mpimg

# ----------------        "    " -----------------

HOURS_BACK = 24

now_ts = time.time()

cutoff_ts = now_ts - HOURS_BACK * 3600

print("\n" + "="*80)

print(f"   out_dir / hier_dir / surv_out    {HOURS_BACK}    /    ")

print("CSV/XLSX >100     10  PNG/JPG     ")

print("="*80 + "\n")

def format_ts(ts):

    return dt.datetime.fromtimestamp(ts).strftime("%Y-%m-%d %H:%M:%S")

# --------    /   --------

def print_csv_limited(path, max_full_rows=100, head_rows=10):

    df = pd.read_csv(path)

    n_rows, n_cols = df.shape

    if n_rows <= max_full_rows:

        print(f"[CSV]   {n_rows}   × {n_cols}     \n")

        print(df.to_string(index=False))

    else:

        print(f"[CSV]   {n_rows}   × {n_cols}     {head_rows}  \n")

        print(df.head(head_rows).to_string(index=False))

def print_excel_limited(path, max_full_rows=100, head_rows=10):

    df = pd.read_excel(path)
```

```
    n_rows, n_cols = df.shape
    if n_rows <= max_full_rows:
        print(f"[XLSX]  {n_rows}  × {n_cols}    \n")
        print(df.to_string(index=False))
    else:
        print(f"[XLSX]  {n_rows}  × {n_cols}    {head_rows}  \n")
        print(df.head(head_rows).to_string(index=False))
def print_txt(path):
    with open(path, "r", encoding="utf-8", errors="ignore") as f:
        print(f.read())
def print_json(path):
    with open(path, "r", encoding="utf-8", errors="ignore") as f:
        obj = json.load(f)
    print(json.dumps(obj, indent=2, ensure_ascii=False))
def show_image(path):
    img = mpimg.imread(path)
    plt.figure(figsize=(6, 4))
    plt.imshow(img)
    plt.axis("off")
    plt.title(os.path.basename(path))
    plt.tight_layout()
    plt.show()
# --------      24      --------
```

```
def dump_root(root_path, title):
    if root_path is None or not os.path.isdir(root_path):
        print(f"[WARN]       {title} ({root_path})")
        return
    print("\n" + "#"*80)
    print(f"#   : {title} ({root_path})")
    print("#"*80 + "\n")
    for dirpath, dirnames, filenames in os.walk(root_path):
        rel_dir = os.path.relpath(dirpath, root_path)
        if rel_dir == ".":
            rel_dir = "(root)"
        print("\n" + "-"*80)
        print(f"   : {rel_dir}")
        print("-"*80 + "\n")
        printed_any = False
        for fname in sorted(filenames):
            fpath = os.path.join(dirpath, fname)
            ext = os.path.splitext(fname)[1].lower()
            #
            if ext not in [".csv", ".txt", ".json", ".xlsx", ".xls",
                           ".png", ".jpg", ".jpeg"]:
                continue
            try:
                mtime = os.path.getmtime(fpath)
```

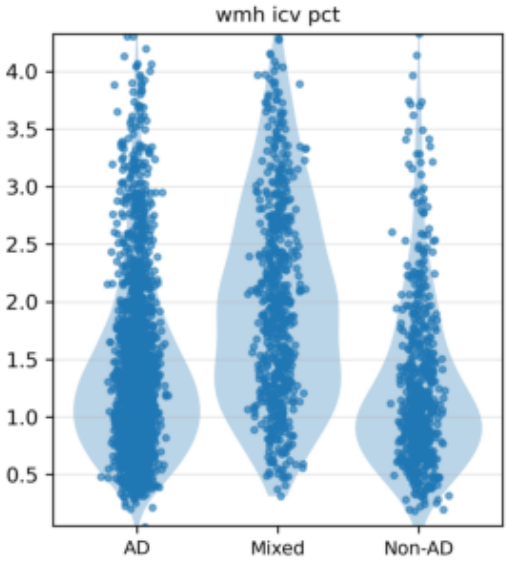
```
            except OSError:
                continue
            if mtime < cutoff_ts:
                #   24
                continue
            if not printed_any:
                printed_any = True
            print("\n" + "-"*40)
            print(f" : {fname}")
            print(f"  : {format_ts(mtime)}")
            print("-"*40 + "\n")
            try:
                if ext == ".csv":
                    print_csv_limited(fpath)
                elif ext in [".xlsx", ".xls"]:
                    print_excel_limited(fpath)
                elif ext == ".txt":
                    print_txt(fpath)
                elif ext == ".json":
                    print_json(fpath)
                elif ext in [".png", ".jpg", ".jpeg"]:
                    show_image(fpath)
            except Exception as e:
```

```
                print(f"[ERROR]   {fname}  : {e}")

        if not printed_any:

            print("[INFO]      24       ")

# --------          surv_out  --------

OUT_DIR_  = globals().get("OUT_DIR")  or globals().get("out_dir")

HIER_DIR_ = globals().get("HIER_DIR") or globals().get("hier_dir")

SURV_OUT_ = globals().get("surv_out")

dump_root(OUT_DIR_,  "out_dir  /     24h ")

dump_root(HIER_DIR_, "hier_dir        24h ")

dump_root(SURV_OUT_, "surv_out  /     24h ")

print("\n" + "="*80)

print(f"   {HOURS_BACK}       ")

print("="*80 + "\n")
```

```
================================================================================
   out_dir / hier_dir / surv_out    24    /
CSV/XLSX >100     10  PNG/JPG
================================================================================


################################################################################
#   : out_dir  /     24h  (surv_out)
################################################################################


--------------------------------------------------------------------------------
   : (root)
--------------------------------------------------------------------------------


----------------------------------------
 : fig_violin_selected_2.png
  : 2025-12-11 06:39:49
```

---

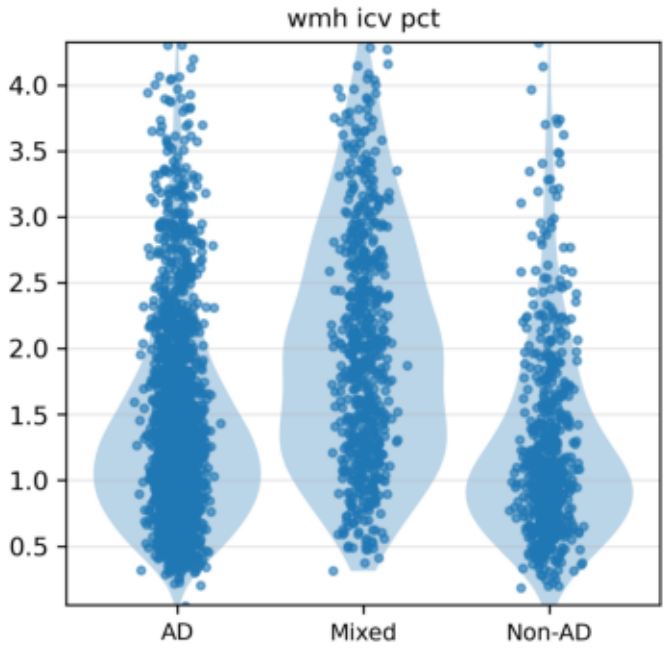
fig_violin_selected_2.png



: violin_wmh_icv_pct.png
: 2025-12-11 06:39:49

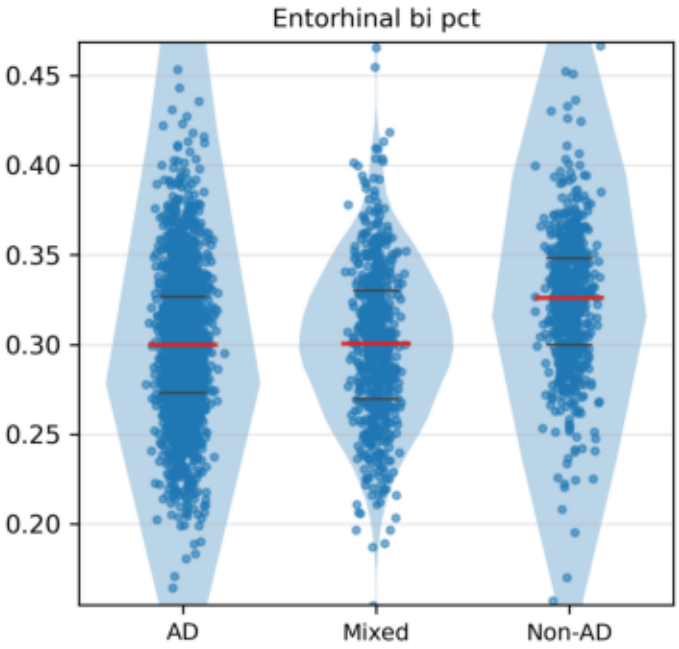
---

violin_wmh_icv_pct.png




```
----------------------------------------------------------------------------------
   : .ipynb_checkpoints
----------------------------------------------------------------------------------

[INFO]      24

----------------------------------------------------------------------------------
   : violin_ad_signature
----------------------------------------------------------------------------------


----------------------------------------
 : violin_autoscale_Entorhinal_bi_pct.png
  : 2025-12-11 06:39:51
----------------------------------------
```

violin_autoscale_Entorhinal_bi_pct.png

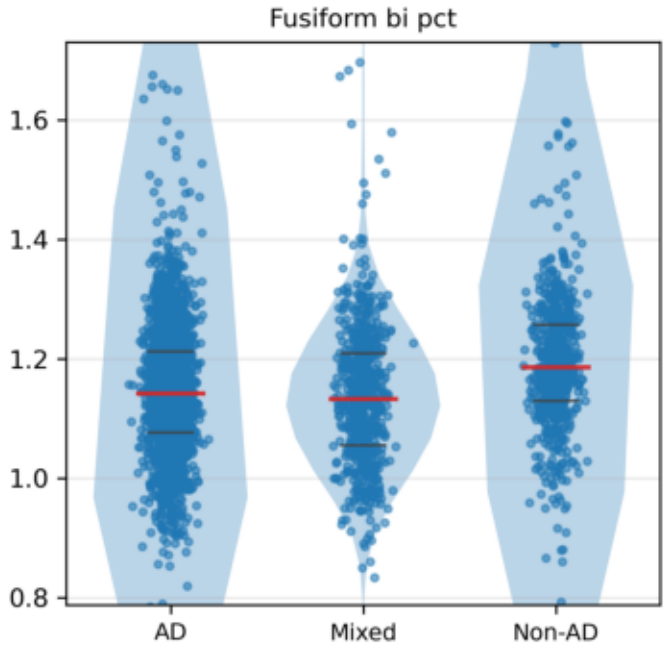


: violin_autoscale_Fusiform_bi_pct.png

: 2025-12-11 06:39:52

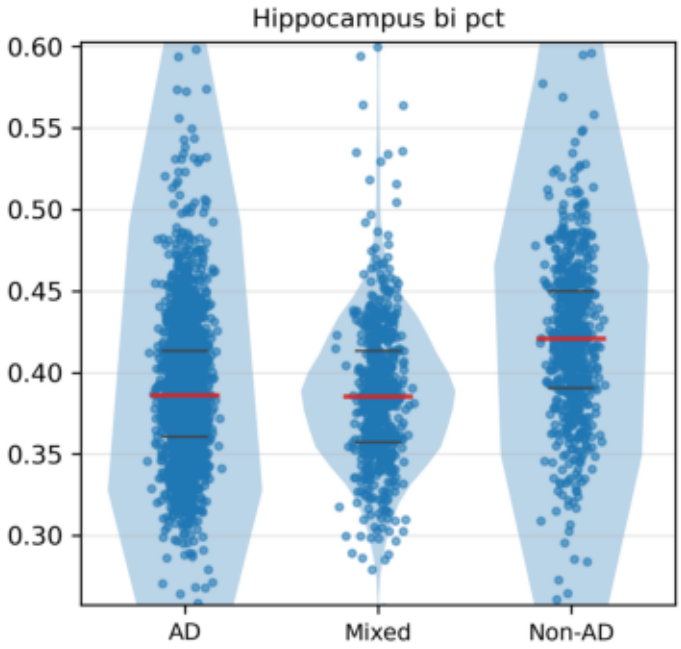
---

violin_autoscale_Fusiform_bi_pct.png



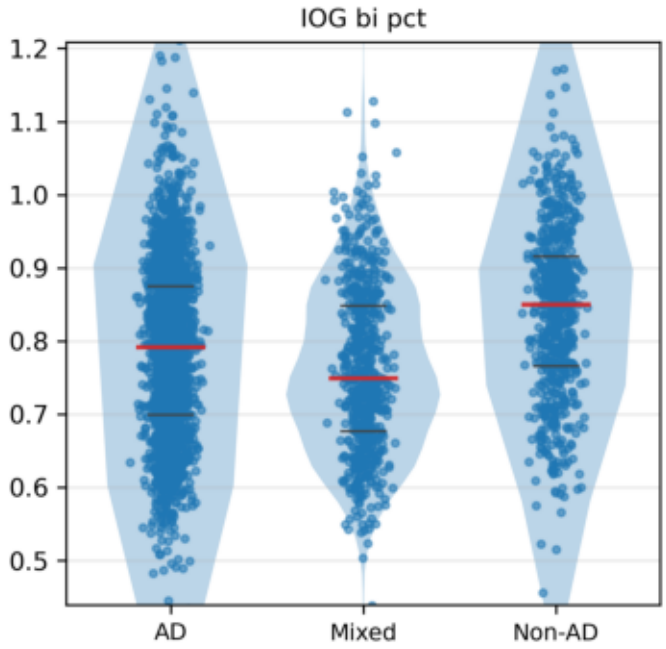

```
----------------------------------------
 : violin_autoscale_Hippocampus_bi_pct.png
  : 2025-12-11 06:39:51
----------------------------------------
```

violin_autoscale_Hippocampus_bi_pct.png



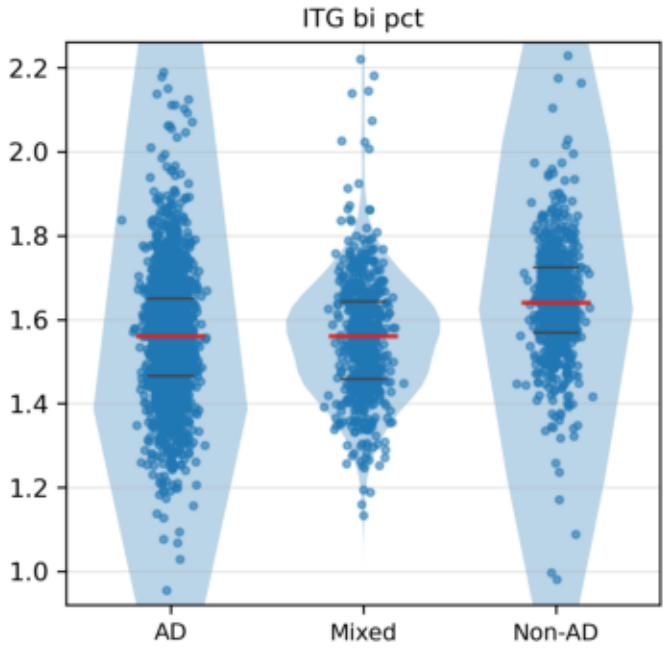

```
----------------------------------------
 : violin_autoscale_IOG_bi_pct.png
  : 2025-12-11 06:39:53
----------------------------------------
```

violin_autoscale_lOG_bi_pct.png



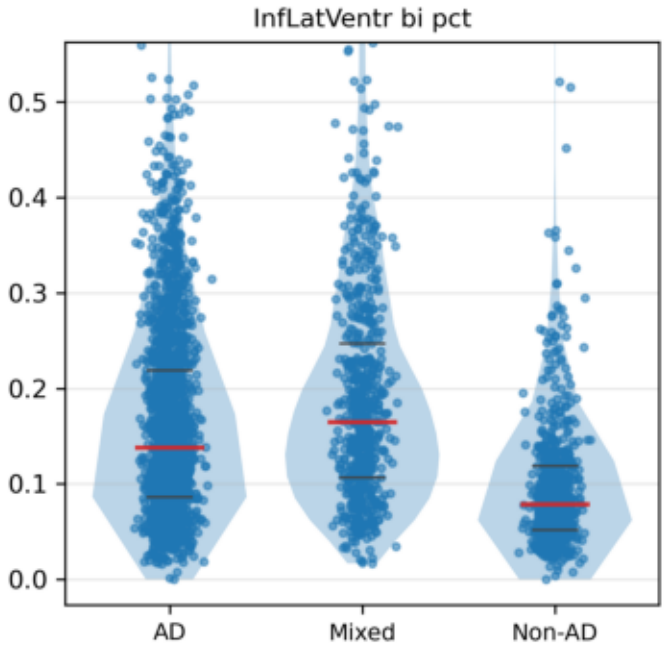

```
----------------------------------------
 : violin_autoscale_ITG_bi_pct.png
  : 2025-12-11 06:39:51
----------------------------------------
```

violin_autoscale_ITG_bi_pct.png



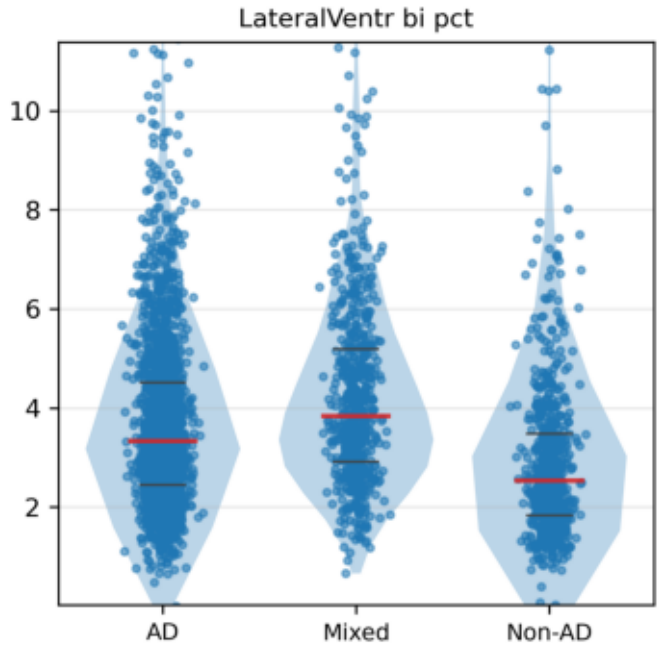

```
----------------------------------------
 : violin_autoscale_InfLatVentr_bi_pct.png
  : 2025-12-11 06:39:54
----------------------------------------
```

violin_autoscale_InfLatVentr_bi_pct.png



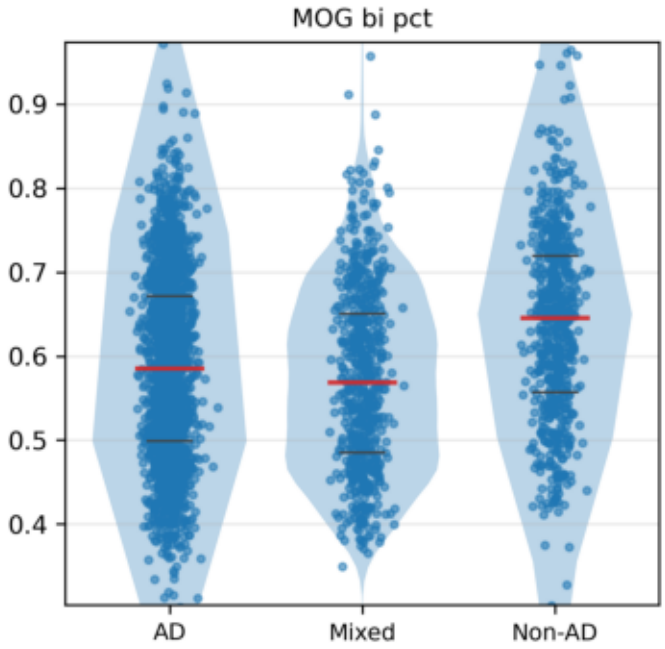

```
----------------------------------------
 : violin_autoscale_LateralVentr_bi_pct.png
  : 2025-12-11 06:39:54
----------------------------------------
```

violin_autoscale_LateralVentr_bi_pct.png



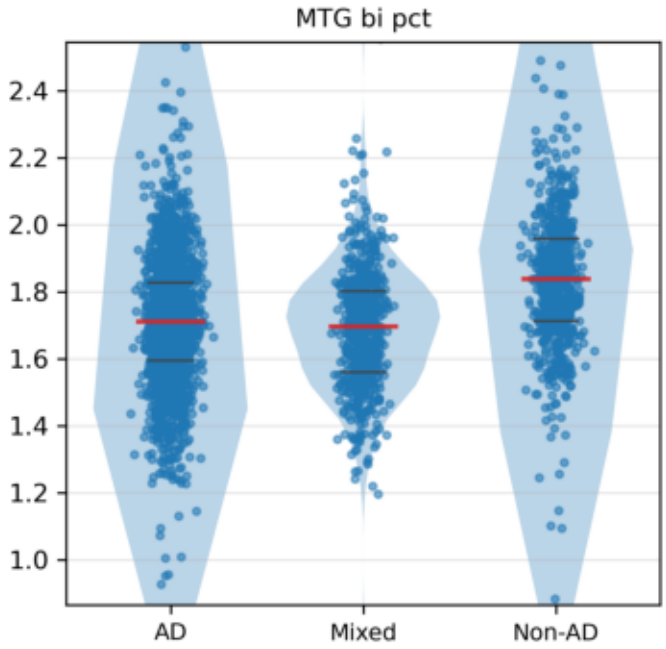

```
----------------------------------------
 : violin_autoscale_MOG_bi_pct.png
  : 2025-12-11 06:39:53
----------------------------------------
```

violin_autoscale_MOG_bi_pct.png



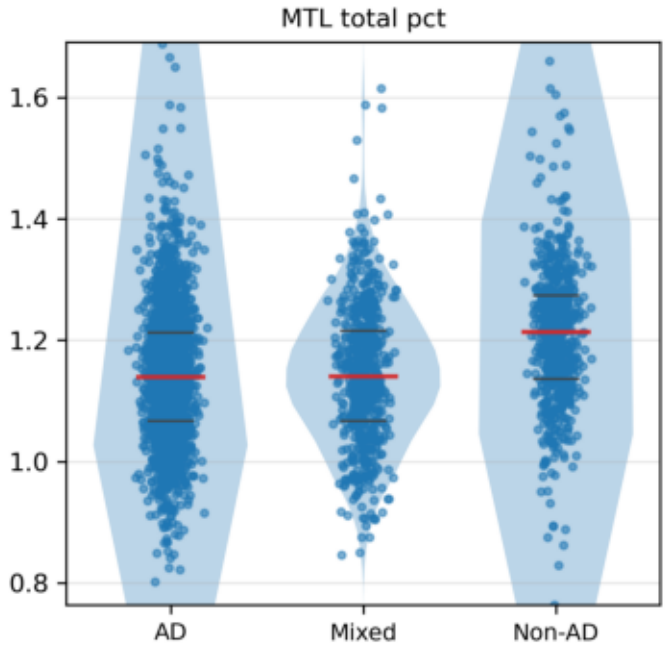

```
----------------------------------------
 : violin_autoscale_MTG_bi_pct.png
  : 2025-12-11 06:39:51
----------------------------------------
```

violin_autoscale_MTG_bi_pct.png



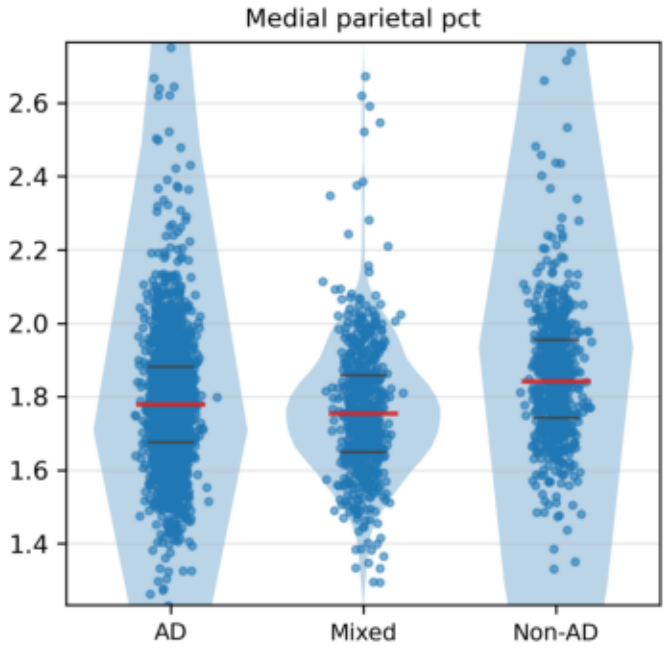

```
----------------------------------------
 : violin_autoscale_MTL_total_pct.png
  : 2025-12-11 06:39:50
----------------------------------------
```

violin_autoscale_MTL_total_pct.png



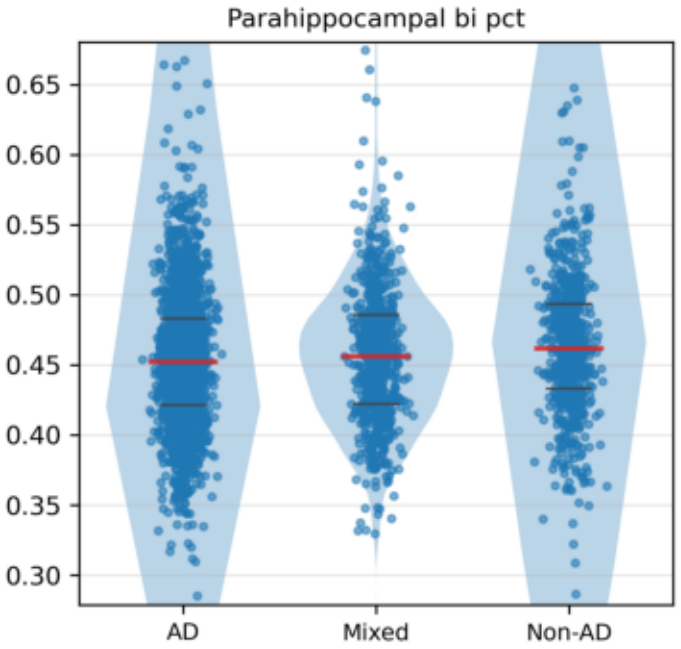

```
----------------------------------------
 : violin_autoscale_Medial_parietal_pct.png
  : 2025-12-11 06:39:50
----------------------------------------
```

violin_autoscale_Medial_parietal_pct.png



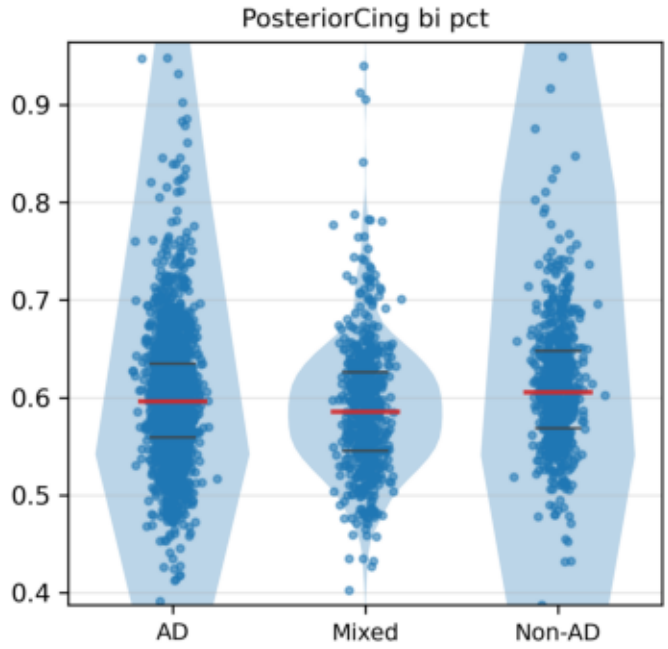

```
----------------------------------------
 : violin_autoscale_Parahippocampal_bi_pct.png
  : 2025-12-11 06:39:51
----------------------------------------
```

violin_autoscale_Parahippocampal_bi_pct.png



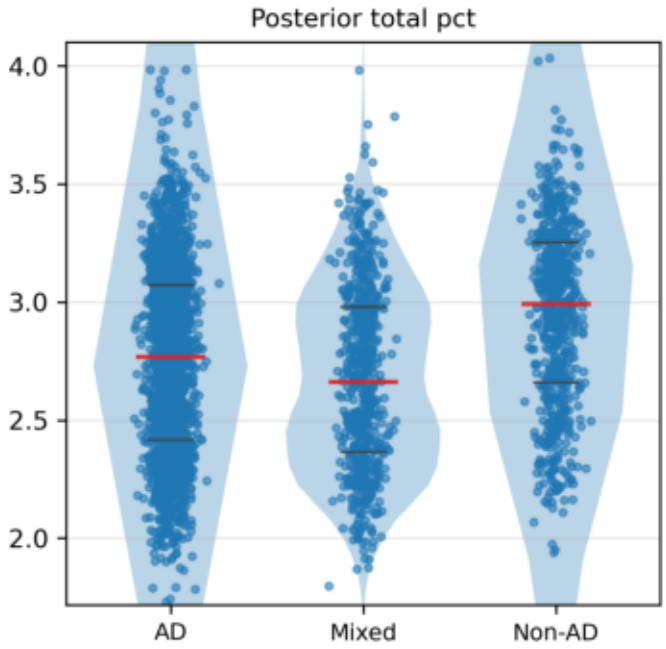

```
----------------------------------------
 : violin_autoscale_PosteriorCing_bi_pct.png
  : 2025-12-11 06:39:52
----------------------------------------
```

violin_autoscale_PosteriorCing_bi_pct.png



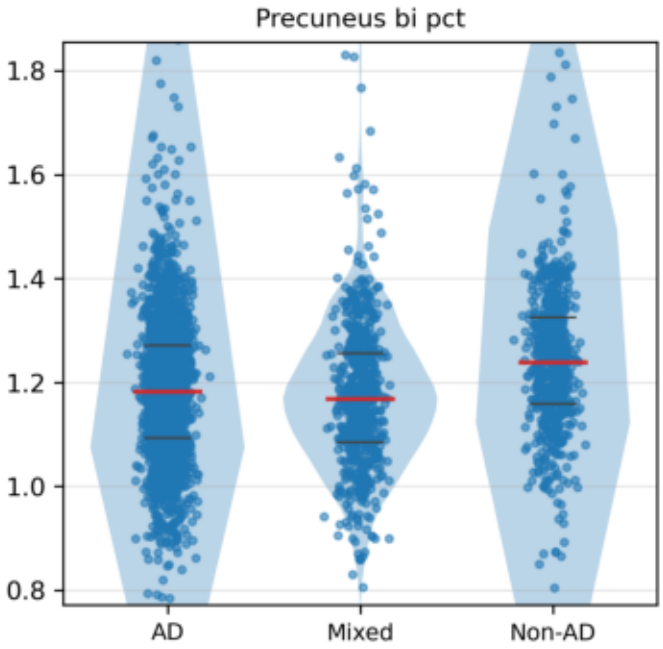

```
----------------------------------------
 : violin_autoscale_Posterior_total_pct.png
  : 2025-12-11 06:39:50
----------------------------------------
```

violin_autoscale_Posterior_total_pct.png



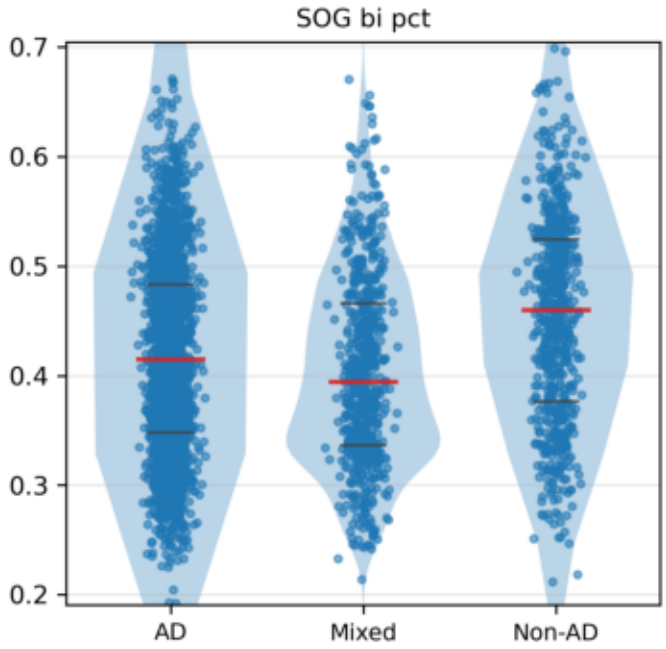

```
----------------------------------------
 : violin_autoscale_Precuneus_bi_pct.png
  : 2025-12-11 06:39:52
----------------------------------------
```

violin_autoscale_Precuneus_bi_pct.png



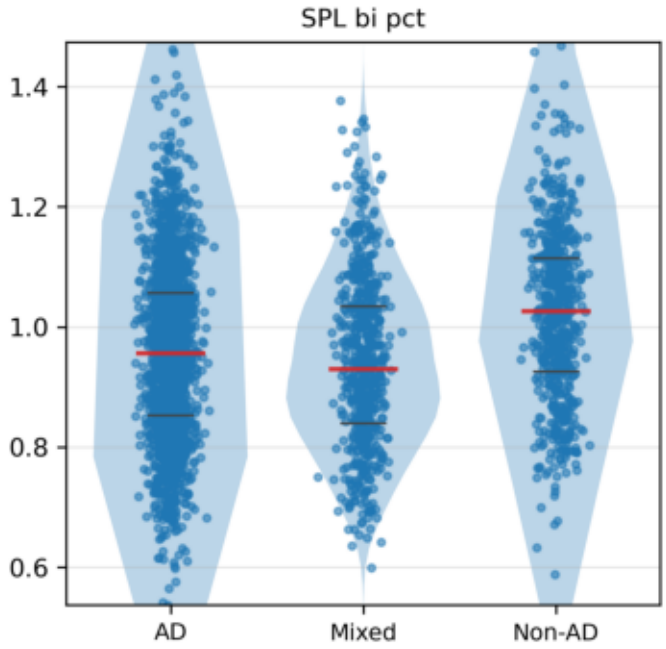

```
----------------------------------------
 : violin_autoscale_SOG_bi_pct.png
  : 2025-12-11 06:39:53
----------------------------------------
```

violin_autoscale_SOG_bi_pct.png



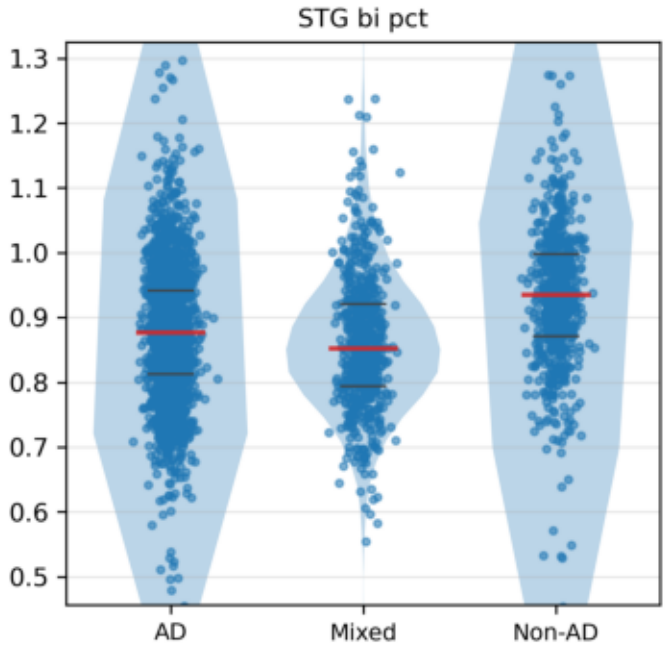

```
----------------------------------------
 : violin_autoscale_SPL_bi_pct.png
  : 2025-12-11 06:39:53
----------------------------------------
```

violin_autoscale_SPL_bi_pct.png



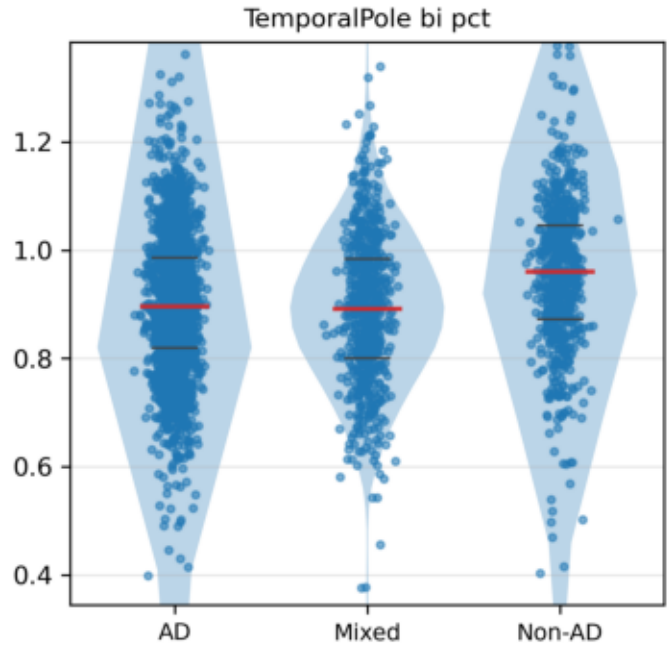

```
----------------------------------------
 : violin_autoscale_STG_bi_pct.png
  : 2025-12-11 06:39:52
----------------------------------------
```

violin_autoscale_STG_bi_pct.png



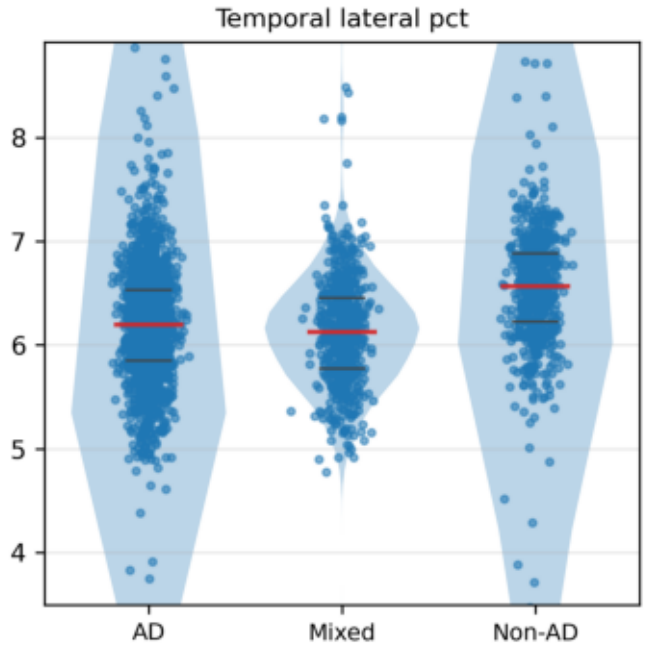

```
----------------------------------------
 : violin_autoscale_TemporalPole_bi_pct.png
  : 2025-12-11 06:39:52
----------------------------------------
```

violin_autoscale_TemporalPole_bi_pct.png



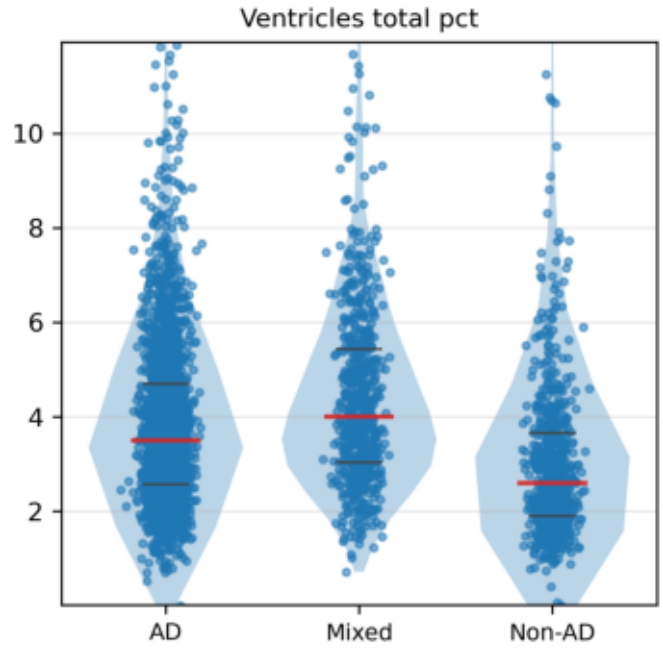

```
----------------------------------------
 : violin_autoscale_Temporal_lateral_pct.png
  : 2025-12-11 06:39:50
----------------------------------------
```

violin_autoscale_Temporal_lateral_pct.png




```
----------------------------------------
 : violin_autoscale_Ventricles_total_pct.png
  : 2025-12-11 06:39:50
----------------------------------------
```

violin_autoscale_Ventricles_total_pct.png

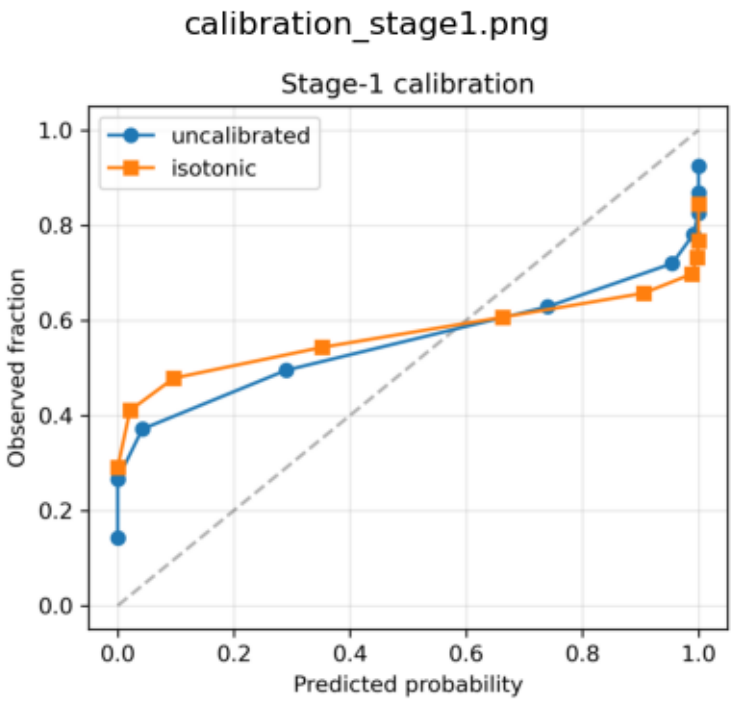



```
--------------------------------------------------------------------------------
   : violin_ad_signature/.ipynb_checkpoints
--------------------------------------------------------------------------------

[INFO]      24

--------------------------------------------------------------------------------
   : panels
--------------------------------------------------------------------------------


----------------------------------------
 : panel_stats.xlsx
  : 2025-12-11 06:39:56
----------------------------------------

[XLSX]   81   × 7

                  feature  group    n    median      IQR      mean        sd
             MTL_total_pct     AD 1992  1.139259 0.145325  1.201319  1.526429
             MTL_total_pct  Mixed  645  1.140277 0.148799  1.146916  0.162011
             MTL_total_pct Non-AD  673  1.213147 0.137617  1.297637  1.686869
Temporal_lateral_total_pct     AD 1992  6.195232 0.679280  6.496795  8.282181
```

Temporal_lateral_total_pct	Mixed	645	6.127592	0.680406	6.159007	0.841304
Temporal_lateral_total_pct	Non-AD	673	6.563137	0.657565	7.032315	8.829971
Medial_parietal_total_pct	AD	1992	1.778761	0.206711	1.879352	2.363406
Medial_parietal_total_pct	Mixed	645	1.753844	0.207630	1.761621	0.210522
Medial_parietal_total_pct	Non-AD	673	1.842116	0.212410	2.018839	2.939572
Posterior_total_pct	AD	1992	2.767136	0.655610	2.886503	3.387500
Posterior_total_pct	Mixed	645	2.661584	0.614563	2.688156	0.466122
Posterior_total_pct	Non-AD	673	2.991168	0.592856	3.111213	2.889485
Ventricles_total_pct	AD	1992	3.496242	2.131109	4.075819	5.872265
Ventricles_total_pct	Mixed	645	3.999202	2.402378	4.491929	2.830929
Ventricles_total_pct	Non-AD	673	2.601969	1.756048	3.345217	6.512640
Hippocampus_bi_pct	AD	1992	0.386053	0.052670	0.408027	0.487231
Hippocampus_bi_pct	Mixed	645	0.385176	0.055795	0.388308	0.060990
Hippocampus_bi_pct	Non-AD	673	0.420978	0.059567	0.451873	0.587622
Entorhinal_bi_pct	AD	1992	0.299949	0.053490	0.314979	0.409513
Entorhinal_bi_pct	Mixed	645	0.300647	0.060372	0.302333	0.056461
Entorhinal_bi_pct	Non-AD	673	0.326027	0.048150	0.345304	0.404680
Parahippocampal_bi_pct	AD	1992	0.452037	0.061883	0.478312	0.631902
Parahippocampal_bi_pct	Mixed	645	0.456191	0.063376	0.456275	0.064864
Parahippocampal_bi_pct	Non-AD	673	0.461998	0.060181	0.500460	0.702435
MTG_bi_pct	AD	1992	1.709775	0.232766	1.792930	2.293839
MTG_bi_pct	Mixed	645	1.694928	0.240371	1.695152	0.259014
MTG_bi_pct	Non-AD	673	1.837126	0.243986	1.973532	2.574655
ITG_bi_pct	AD	1992	1.559783	0.184607	1.636937	2.111106
ITG_bi_pct	Mixed	645	1.560474	0.184199	1.557358	0.196197
ITG_bi_pct	Non-AD	673	1.640526	0.154963	1.756533	2.107073
STG_bi_pct	AD	1992	0.877108	0.128775	0.919214	1.152872
STG_bi_pct	Mixed	645	0.852414	0.126692	0.864929	0.159662
STG_bi_pct	Non-AD	673	0.935208	0.126567	1.017377	1.565971
Fusiform_bi_pct	AD	1992	1.142027	0.135805	1.204811	1.519547
Fusiform_bi_pct	Mixed	645	1.132731	0.154208	1.145425	0.228966
Fusiform_bi_pct	Non-AD	673	1.185327	0.127597	1.276276	1.586081
TemporalPole_bi_pct	AD	1992	0.896349	0.165823	0.942903	1.226623
TemporalPole_bi_pct	Mixed	645	0.891029	0.183899	0.896143	0.194418
TemporalPole_bi_pct	Non-AD	673	0.959916	0.173394	1.008596	1.058556
PosteriorCing_bi_pct	AD	1992	0.595853	0.075455	0.631169	0.801846
PosteriorCing_bi_pct	Mixed	645	0.585491	0.080165	0.588833	0.078716
PosteriorCing_bi_pct	Non-AD	673	0.605608	0.079314	0.676934	1.180881
Precuneus_bi_pct	AD	1992	1.182792	0.177988	1.248183	1.565205
Precuneus_bi_pct	Mixed	645	1.168490	0.171812	1.172788	0.156704
Precuneus_bi_pct	Non-AD	673	1.238770	0.166474	1.341905	1.767684
SPL_bi_pct	AD	1992	0.956389	0.203734	1.009458	1.202711
SPL_bi_pct	Mixed	645	0.929858	0.194135	0.943586	0.177207
SPL_bi_pct	Non-AD	673	1.025704	0.188953	1.076687	1.093838
SOG_bi_pct	AD	1992	0.414528	0.135477	0.438237	0.536496
SOG_bi_pct	Mixed	645	0.393944	0.129822	0.404220	0.092303
SOG_bi_pct	Non-AD	673	0.459814	0.148263	0.469221	0.342198
MOG_bi_pct	AD	1992	0.584834	0.172287	0.614614	0.725199

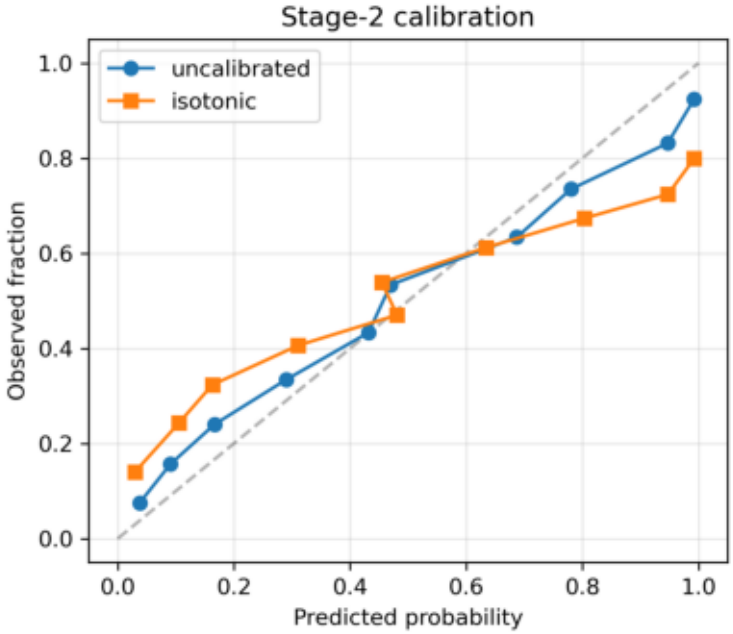
```
               MOG_bi_pct  Mixed  645  0.568642 0.165141  0.574576  0.120535
               MOG_bi_pct Non-AD  673  0.645096 0.162763  0.675756  0.678962
               IOG_bi_pct     AD 1992  0.791794 0.175393  0.824195  0.952378
               IOG_bi_pct  Mixed  645  0.749055 0.171568  0.765774  0.149784
               IOG_bi_pct Non-AD  673  0.849820 0.149495  0.889550  0.810581
      LateralVentr_bi_pct     AD 1992  3.333679 2.065164  3.904633  5.652007
      LateralVentr_bi_pct  Mixed  645  3.838285 2.270582  4.299402  2.719391
      LateralVentr_bi_pct Non-AD  673  2.530190 1.648096  3.240460  6.283593
       InfLatVentr_bi_pct     AD 1992  0.137946 0.132496  0.171186  0.244964
       InfLatVentr_bi_pct  Mixed  645  0.164789 0.139927  0.192527  0.140319
       InfLatVentr_bi_pct Non-AD  673  0.078636 0.066628  0.104757  0.242632
         Frontal_lobe_pct     AD 1992 11.315142 1.092529 11.908631 14.717127
         Frontal_lobe_pct  Mixed  645 10.960051 1.224867 11.024246  1.234187
         Frontal_lobe_pct Non-AD  673 11.825900 1.226606 12.609408 14.704461
        Parietal_lobe_pct     AD 1992  6.016160 0.815502  6.330346  7.933474
        Parietal_lobe_pct  Mixed  645  5.853023 0.827523  5.893747  0.679606
        Parietal_lobe_pct Non-AD  673  6.312448 0.728582  6.785547  8.716028
       Occipital_lobe_pct     AD 1992  4.476996 0.770140  4.704412  5.625371
       Occipital_lobe_pct  Mixed  645  4.346759 0.712566  4.370506  0.618659
       Occipital_lobe_pct Non-AD  673  4.733904 0.793287  4.986190  4.853155
        Temporal_lobe_pct     AD 1992  8.194915 0.790056  8.617716 10.933475
        Temporal_lobe_pct  Mixed  645  8.112879 0.790070  8.174911  1.089876
        Temporal_lobe_pct Non-AD  673  8.711527 0.787113  9.318653 11.621893
           Cerebellum_pct     AD 1992  8.966131 1.201418  9.210946 10.645446
           Cerebellum_pct  Mixed  645  8.859635 1.306490  8.668051  1.850111
           Cerebellum_pct Non-AD  673  9.028581 1.179957  9.633571 13.621774
            Brainstem_pct     AD 1992  1.223703 0.169722  1.287620  1.513234
            Brainstem_pct  Mixed  645  1.211597 0.180940  1.222858  0.302610
            Brainstem_pct Non-AD  673  1.238692 0.172759  1.341089  1.732629

--------------------------------------------------------------------------------
   : panels/.ipynb_checkpoints
--------------------------------------------------------------------------------

[INFO]      24

--------------------------------------------------------------------------------
   : panels_grouped
--------------------------------------------------------------------------------

[INFO]      24

--------------------------------------------------------------------------------
   : panels_grouped/.ipynb_checkpoints
--------------------------------------------------------------------------------

[INFO]      24
```

```
--------------------------------------------------------------------------------
   : analysis_out
--------------------------------------------------------------------------------


----------------------------------------
 : calibration_stage1.png
  : 2025-12-11 06:39:47
----------------------------------------
```



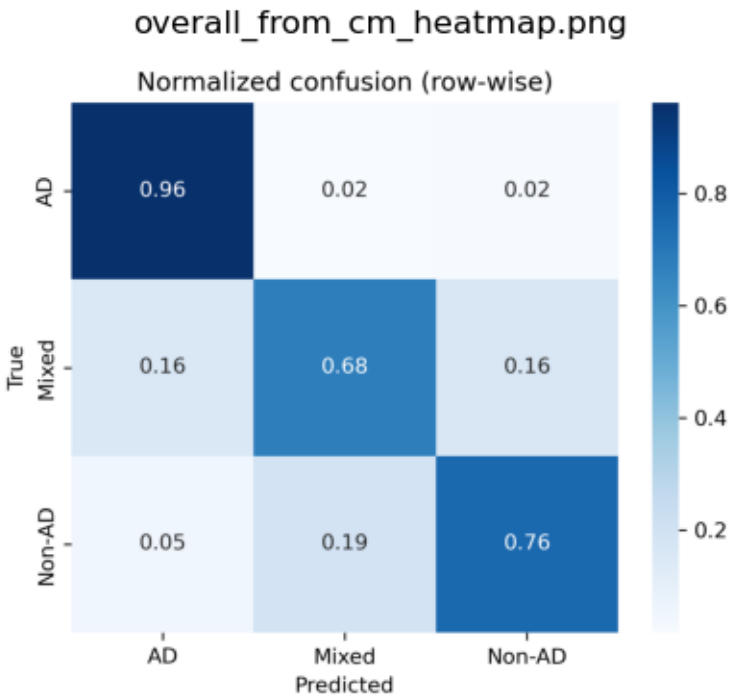

```
----------------------------------------
 : calibration_stage2.png
  : 2025-12-11 06:39:47
----------------------------------------
```

calibration_stage2.png



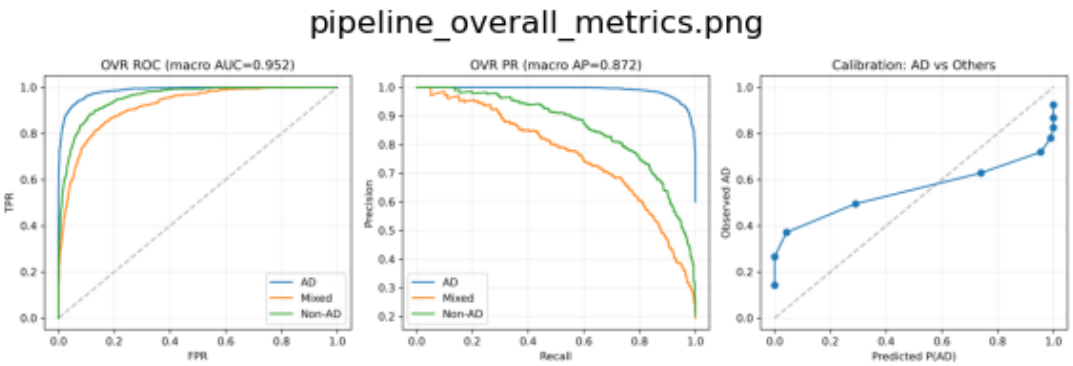

```
----------------------------------------
 : final.csv
  : 2025-12-11 06:39:48
----------------------------------------

[ERROR]   final.csv  : 'utf-8' codec can't decode byte 0xfd in position 15:
invalid start byte

----------------------------------------
 : importances_stage1_stage2.csv
  : 2025-12-11 06:40:20
----------------------------------------

[CSV]  118  × 3    10

 stage                                        feature  perm_importance
stage1                num__048_Right Hippocampus__pct         0.001998
stage1                 num__049_Left Hippocampus__pct         0.002747
stage1        num__117_Right Ent entorhinal area__pct         0.004600
stage1         num__118_Left Ent entorhinal area__pct         0.011946
stage1  num__171_Right PHG parahippocampal gyrus__pct         0.007468
stage1   num__172_Left PHG parahippocampal gyrus__pct         0.005891
stage1  num__155_Right MTG middle temporal gyrus__pct         0.004018
```

```
stage1    num__156_Left MTG middle temporal gyrus__pct         0.003773
stage1 num__133_Right ITG inferior temporal gyrus__pct         0.004667
stage1  num__134_Left ITG inferior temporal gyrus__pct         0.004838

----------------------------------------
 : overall_from_cm_confusion.csv
  : 2025-12-10 12:36:56
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1917          33           42
 true_Mixed      101         438          106
true_Non-AD       34         130          509

----------------------------------------
 : overall_from_cm_heatmap.png
  : 2025-12-10 12:36:56
----------------------------------------
```




```
----------------------------------------
```

```
 : overall_from_cm_per_class.csv
  : 2025-12-10 12:36:56
----------------------------------------

[CSV]   3   × 5

 class  support  precision   recall       f1
    AD     1992   0.934211 0.962349 0.948071
 Mixed      645   0.728785 0.679070 0.703050
Non-AD      673   0.774734 0.756315 0.765414

----------------------------------------
 : overall_from_cm_summary.csv
  : 2025-12-10 12:36:56
----------------------------------------

[CSV]   1   × 7

 accuracy  balanced_accuracy  macro_precision  macro_recall  macro_f1
weighted_f1    N
 0.865257           0.799245         0.812577      0.799245  0.805512
0.863187 3310

----------------------------------------
 : pipeline_overall_metrics.png
  : 2025-12-11 06:40:07
----------------------------------------
```

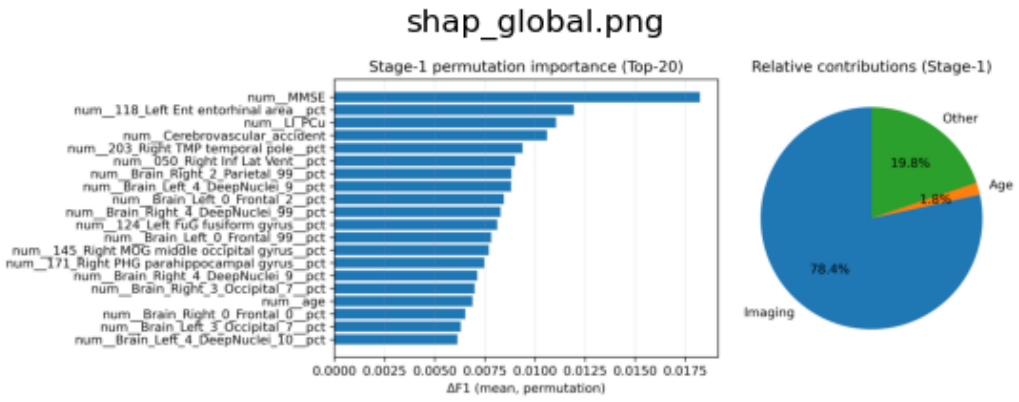



```
----------------------------------------
 : posthoc_effects.csv
  : 2025-12-11 06:40:07
----------------------------------------
```

[CSV] 27 × 10

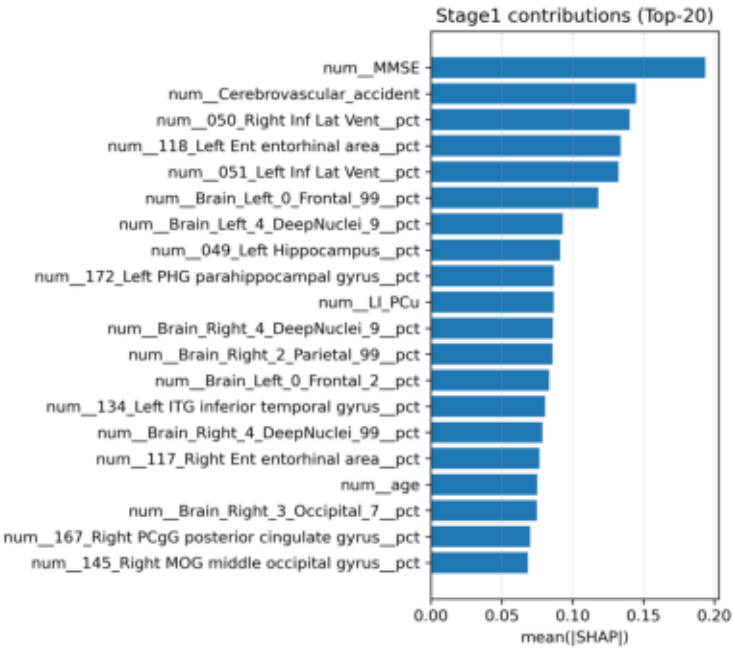
feature	method	effect	effect_value	AD_vs_Mixed	AD_vs_NonAD	Mixed_vs_NonAD	delta_AD_Mixed	delta_AD_NonAD	delta_Mixed_NonAD
MTL_total_pct	Kruskal+Dunn	cliffs_delta	NaN	7.356779e-01	1.308308e-38	6.863209e-24	-0.008832	-0.336730	-0.322027
Temporal_lateral_total_pct	Kruskal+Dunn	cliffs_delta	NaN	3.162468e-03	3.453051e-53	1.822434e-47	0.077214	-0.397082	-0.461465
Medial_parietal_total_pct	Kruskal+Dunn	cliffs_delta	NaN	7.761681e-04	3.403544e-19	1.176805e-22	0.087932	-0.231704	-0.315263
Posterior_total_pct	Kruskal+Dunn	cliffs_delta	NaN	2.437196e-05	1.365917e-25	3.149690e-31	0.110414	-0.270210	-0.372883
Ventricles_total_pct	Kruskal+Dunn	cliffs_delta	NaN	9.806081e-13	2.418328e-37	1.375395e-52	-0.186612	0.329589	0.487990
Hippocampus_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	3.566490e-01	1.545801e-52	5.696907e-39	0.024115	-0.394582	-0.416570
Entorhinal_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	5.148307e-01	4.093116e-41	6.644804e-23	-0.017040	-0.347836	-0.314876
Parahippocampal_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	2.699707e-01	8.054004e-06	7.315053e-03	-0.028860	-0.120844	-0.089593
MTG_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	7.093597e-03	9.478035e-46	3.132182e-42	0.070437	-0.367516	-0.434341
ITG_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	2.952550e-01	1.824305e-40	5.876624e-32	0.027382	-0.344992	-0.375532
STG_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	6.249700e-05	1.109280e-35	7.288792e-38	0.104727	-0.321851	-0.412060
Fusiform_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	4.722359e-02	5.081099e-22	1.742540e-19	0.051912	-0.251252	-0.288743
TemporalPole_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	1.918256e-01	8.897487e-22	2.913916e-17	0.034146	-0.249784	-0.270429
PosteriorCing_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	8.293257e-05	2.587955e-05	1.914448e-10	0.102964	-0.110653	-0.207966
Precuneus_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	1.653832e-02	3.432695e-20	6.768476e-21	0.062704	-0.238102	-0.302104
SPL_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	2.548398e-03	2.525884e-20	3.476921e-22	0.078942	-0.238945	-0.311798
SOG_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	1.998253e-04	6.078556e-16	1.265508e-20	0.097299	-0.209499	-0.300022
MOG_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	3.875535e-03	2.233240e-21	1.819985e-23	0.075556	-0.245511	-0.321147
IOG_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	8.795861e-07	4.968916e-23	4.725080e-32	0.128629	-0.255478	-0.377965
LateralVentr_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	1.081197e-12	9.987534e-35	3.177055e-50	-0.186260	0.317321	0.476609
InfLatVentr_bi_pct	Kruskal+Dunn	cliffs_delta	NaN	3.755816e-08	6.407881e-68	2.412823e-73	-0.143937	0.448959	0.578451
Frontal_lobe_pct	Kruskal+Dunn	cliffs_delta	NaN	4.512041e-16					

```
3.262321e-32    5.769860e-49        0.212531       -0.305073          -0.470436
         Parietal_lobe_pct Kruskal+Dunn cliffs_delta           NaN 6.924282e-07
1.119597e-28    3.885664e-37        0.129850       -0.286979          -0.407950
        Occipital_lobe_pct Kruskal+Dunn cliffs_delta           NaN 1.016453e-07
4.605263e-19    3.334823e-29        0.139275       -0.230848          -0.360093
         Temporal_lobe_pct Kruskal+Dunn cliffs_delta           NaN 3.832497e-03
3.951058e-61    8.747096e-52        0.075648       -0.426407          -0.482705
            Cerebellum_pct Kruskal+Dunn cliffs_delta           NaN 4.447781e-04
1.657850e-01    1.301054e-04        0.094659       -0.035682          -0.130129
             Brainstem_pct Kruskal+Dunn cliffs_delta           NaN 2.995326e-02
1.559313e-03    3.689659e-05        0.056788       -0.084439          -0.139150

----------------------------------------
 : shap_global.png
  : 2025-12-11 06:40:20
----------------------------------------
```




```
----------------------------------------
 : stage1_shap_bar.png
  : 2025-12-11 06:39:48
----------------------------------------
```

stage1_shap_bar.png




```
----------------------------------------
 : stage1_shap_samples.csv
  : 2025-12-11 06:39:48
----------------------------------------

[CSV]   2000   × 104     10

 sample_index_in_X  num__048_Right Hippocampus__pct  num__049_Left
Hippocampus__pct  num__117_Right Ent entorhinal area__pct  num__118_Left Ent
entorhinal area__pct  num__171_Right PHG parahippocampal gyrus__pct
num__172_Left PHG parahippocampal gyrus__pct  num__155_Right MTG middle temporal
gyrus__pct  num__156_Left MTG middle temporal gyrus__pct  num__133_Right ITG
inferior temporal gyrus__pct  num__134_Left ITG inferior temporal gyrus__pct
num__201_Right STG superior temporal gyrus__pct  num__202_Left STG superior
temporal gyrus__pct  num__123_Right FuG fusiform gyrus__pct  num__124_Left FuG
fusiform gyrus__pct  num__203_Right TMP temporal pole__pct  num__204_Left TMP
temporal pole__pct  num__167_Right PCgG posterior cingulate gyrus__pct
num__168_Left PCgG posterior cingulate gyrus__pct  num__169_Right PCu
precuneus__pct  num__170_Left PCu precuneus__pct  num__199_Right SPL superior
parietal lobule__pct  num__200_Left SPL superior parietal lobule__pct
num__197_Right SOG superior occipital gyrus__pct  num__198_Left SOG superior
occipital gyrus__pct  num__145_Right MOG middle occipital gyrus__pct
num__146_Left MOG middle occipital gyrus__pct  num__129_Right IOG inferior
```

```
occipital gyrus__pct  num__130_Left IOG inferior occipital gyrus__pct
num__052_Right Lateral Ventricle__pct  num__053_Left Lateral Ventricle__pct
num__050_Right Inf Lat Vent__pct  num__051_Left Inf Lat Vent__pct
num__Brain_Left_0_Frontal_0__pct  num__Brain_Left_0_Frontal_1__pct
num__Brain_Left_0_Frontal_2__pct  num__Brain_Left_0_Frontal_99__pct
num__Brain_Left_1_Temporal_3__pct  num__Brain_Left_1_Temporal_4__pct
num__Brain_Left_1_Temporal_99__pct  num__Brain_Left_2_Parietal_5__pct
num__Brain_Left_2_Parietal_6__pct  num__Brain_Left_2_Parietal_99__pct
num__Brain_Left_3_Occipital_7__pct  num__Brain_Left_3_Occipital_8__pct
num__Brain_Left_4_DeepNuclei_9__pct  num__Brain_Left_4_DeepNuclei_10__pct
num__Brain_Left_4_DeepNuclei_99__pct  num__Brain_Left_99_99__pct
num__Brain_Medial_99_99__pct  num__Brain_Right_0_Frontal_0__pct
num__Brain_Right_0_Frontal_1__pct  num__Brain_Right_0_Frontal_2__pct
num__Brain_Right_0_Frontal_99__pct  num__Brain_Right_1_Temporal_3__pct
num__Brain_Right_1_Temporal_4__pct  num__Brain_Right_1_Temporal_99__pct
num__Brain_Right_2_Parietal_5__pct  num__Brain_Right_2_Parietal_6__pct
num__Brain_Right_2_Parietal_99__pct  num__Brain_Right_3_Occipital_7__pct
num__Brain_Right_3_Occipital_8__pct  num__Brain_Right_4_DeepNuclei_9__pct
num__Brain_Right_4_DeepNuclei_10__pct  num__Brain_Right_4_DeepNuclei_99__pct
num__Brain_Right_99_99__pct  num__MTL_total_pct  num__Temporal_lateral_total_pct
num__Medial_parietal_total_pct  num__Posterior_total_pct
num__Ventricles_total_pct  num__LI_PCgG  num__LI_PCu  num__LI_MTG  num__LI_ITG
num__LI_STG  num__asthma  num__falls  num__hypertension
num__Cerebrovascular_accident  num__Epilepsy  num__Diabetes_mellitus
num__Chronic_kidney_disease  num__Psoriasis  num__Parkinsons_disease
num__Multiple_sclerosis  num__Eczema  num__Hypertensive_disorder
num__Transient_ischemic_attack  num__Migraine
num__Chronic_obstructive_lung_disease  num__Arthritis  num__Heart_failure
num__Asthma  num__Ischemic_heart_disease  num__Inflammatory_bowel_disease
num__Atrial_fibrillation  num__Chronic_liver_disease  num__Chronic_sinusitis
num__Coronary_arteriosclerosis  num__age  num__MMSE  cat__Gender_ID_Female
cat__Gender_ID_Male
             2736                        -0.042848
0.039007                                -0.120192
0.073234                                     -0.007122
0.082126                                     -0.053379
0.000067                                        0.307251
-0.046602                                       -0.004272
0.038207                                   -0.066306
0.009699                                   0.147058
0.054063                                          -0.084765
-0.001352                             0.003268                          -0.023976
-0.030834                                        0.033434
-0.044166                                        0.018618
-0.085121                                      0.015797
0.211561                                        0.008066
0.108714                                 0.088432                           -0.319898
-0.161201                              -0.028602                           0.048930
```

0.112723 0.187962 0.066968
0.007539 -0.005484 0.010338
-0.009570 -0.017830
0.068440 -0.001253
0.028766 0.001218
-0.036076 0.028574 0.002894
0.044893 -0.045883 -0.014778
-0.022229 -0.020762
0.005358 -0.040874
-0.003077 -0.010958
0.097744 0.221303
0.016838 -0.020708
0.025087 0.006399 -0.018775
0.029850 0.014746 0.008749
0.049182 0.011811 -0.028863 0.060959 -0.016233
0.047051 0.009184 0.000181 0.006856 0.0
0.084498 -0.000179 -0.008593 0.001333
0.0 -0.017928 0.0 -0.009599
-0.001719 0.027603 0.002932
0.012043 -0.005960 -0.006985 0.000350
0.005994 0.000241 0.002053
-0.000108 0.0 0.000288 0.080732
0.175569 -0.047545 -0.004504
2264 0.036282
0.054474 -0.147230
-0.132562 0.002909
-0.070889 -0.058571
0.017572 0.004522
-0.025764 -0.052259
0.012689 -0.046056
0.033298 -0.005179
-0.112515 0.030904
0.022778 -0.064431 -0.052948
0.077481 0.106214
-0.007116 0.012643
-0.049463 -0.016639
0.082559 0.015259
-0.040271 -0.011809
0.105029 -0.046118 0.183680
0.081218 0.020690 0.067675
0.038455 0.003529 0.046382
0.000291 0.006116 0.001819
0.017265 0.005882
0.060237 0.033158
0.000576 0.063276 0.150378
0.037100 0.010892 -0.025527
0.009576 0.011888 -0.000244
-0.007321 -0.031021

0.031169 0.036962
0.056111 0.002872
0.016973 0.014580
-0.004180 0.068595 -0.026202
-0.008139 -0.007826 0.001816
-0.004674 -0.000260 0.120869 -0.073031 -0.105558 0.131547
0.004271 0.010678 0.0 0.093734
-0.000254 -0.026603 0.001613 0.0
-0.015265 0.0 -0.004082 0.001012
0.039227 0.002682 0.029639 -0.005773
-0.016206 0.000440 0.008106
0.000241 0.001820 -0.000108
0.0 0.000153 -0.000958 -0.274583 -0.036972
-0.004504
528 -0.017271
0.012780 0.023374
-0.037124 -0.000176
-0.062521 -0.030670
-0.011021 -0.089106
-0.075087 0.144476
0.012867 -0.068646
-0.031972 0.080269
0.039049 0.082042
0.032157 0.010694 -0.023647
-0.013409 0.061227
-0.050542 0.020491
0.015024 0.045805
-0.070786 0.011066
-0.065045 -0.035873
-0.025759 0.096764 -0.067636
-0.116360 -0.477103 -0.404952
0.004003 0.008951 -0.021688
0.009005 0.048309 -0.036708
0.033571 0.033669
-0.150664 -0.124383
0.010508 -0.142552 -0.027800
-0.067889 0.018636 -0.111266
-0.019229 -0.007581
-0.019709 -0.004242
0.017910 0.147632
-0.211160 0.085527
0.063953 -0.086831
0.043550 -0.030546 -0.188016
-0.056733 -0.025998 0.014405
-0.049303 -0.006492 0.043313 -0.165164 -0.036639
0.004011 -0.113578 0.000264 0.015284 0.0
0.080734 -0.001784 -0.022666 -0.018625
0.0 -0.016603 0.0 -0.007139

0.001439 0.031985 0.000546
0.021375 0.028111 -0.005947 0.000440
-0.088286 0.000110 0.002082
-0.000108 0.0 0.000153 -0.044783
0.144026 -0.029229 0.000795
2477 0.024257
0.012584 0.105014
0.152109 -0.355495
0.087709 -0.056566
0.039301 0.002161
-0.010839 -0.016671
0.008075 0.049635
-0.015463 -0.056179
-0.047717 -0.132249
-0.033076 -0.009512 -0.044740
-0.044847 -0.090476
-0.030169 -0.016271
-0.051535 -0.005293
-0.067335 0.025645
-0.052648 -0.022970
-0.009110 0.088716 -0.019917
-0.077895 0.106936 0.080406
0.046165 0.007029 0.016060
-0.004701 -0.017726 -0.016221
0.012168 -0.022604
-0.176742 -0.084902
-0.150000 0.033807 -0.016473
-0.064937 0.073165 0.008835
-0.024342 -0.000532
-0.013780 -0.003011
-0.038095 0.011491
0.044753 0.033916
-0.029053 -0.116503
0.004210 -0.215762 0.003184
0.053868 0.001183 0.005492
0.038019 -0.018176 0.008273 0.120801 -0.072583
-0.033610 -0.010829 0.000135 0.004504 0.0
0.125071 -0.000370 -0.016778 -0.026717
0.0 -0.012274 0.0 -0.006922
0.001219 0.026075 0.001204
0.017960 0.009882 -0.009106 0.000440
0.019406 0.000241 -0.011355
-0.000108 0.0 0.001482 -0.025080
0.083661 0.014236 0.001452
1262 0.001157
0.044739 0.080856
-0.094951 -0.071282
0.108060 -0.011957

0.000398 0.022615
0.051240 -0.032710
0.013207 0.008409
-0.000771 0.090555
0.015315 0.168598
0.053460 0.005609 -0.002091
0.018278 -0.006999
-0.016150 0.012051
0.002738 0.032202
-0.085876 -0.007748
-0.060471 -0.012105
0.289217 0.095838 -0.038456
-0.027708 0.073253 0.103549
0.048484 0.008095 -0.006724
0.009446 0.001974 -0.007716
0.076362 0.011932
0.097656 -0.091028
0.005621 -0.102086 -0.026346
-0.036560 0.000952 -0.112402
-0.042993 0.005954
-0.003415 -0.011951
-0.017724 0.025530
0.051610 0.044263
-0.013271 0.136306
-0.022137 0.048602 -0.131069
0.036797 -0.064140 0.039190
0.003782 -0.005602 -0.010240 0.156901 0.037746
-0.067791 -0.018646 0.000217 0.010152 0.0
0.109308 -0.000173 0.045139 0.001613
0.0 -0.016415 0.0 -0.009556
-0.002460 0.035817 0.003032
0.018976 0.019065 -0.018024 0.000440
0.019579 0.000241 0.002082
-0.000108 0.0 0.000288 0.075907
-0.109453 0.012496 0.003188
793 0.019622
0.341515 0.064496
-0.021341 0.025897
-0.090670 -0.009889
0.013982 0.016433
-0.034977 -0.024312
0.004502 0.076472
-0.012798 -0.002549
-0.046192 0.084242
-0.030168 0.012687 0.002161
-0.058993 0.010094
0.009808 0.007012
-0.051428 -0.043783

-0.060817 -0.037859
-0.038187 -0.014716
0.272462 0.266788 -0.021693
-0.124407 -0.059617 -0.373896
0.073427 0.007999 0.009666
-0.001496 0.017497 -0.009504
-0.003386 0.018764
0.075090 0.008138
-0.074872 -0.008052 0.023789
0.036369 0.053965 -0.128822
-0.098150 -0.003878
0.008989 0.019289
0.023693 0.026663
0.068835 0.036465
-0.001992 -0.020849
0.045164 -0.009263 0.031500
0.032625 -0.005967 0.040550
-0.021181 -0.012434 -0.033445 0.247396 -0.034847
-0.065241 -0.031685 0.000264 0.011832 0.0
0.108250 -0.000173 -0.009283 0.005888
0.0 -0.014369 0.0 -0.007114
-0.001719 0.032505 0.000817
0.023810 -0.003954 0.106359 0.001695
0.017584 0.000241 0.001904
-0.000108 0.0 0.000153 0.026620
0.052530 -0.012280 -0.002051
1401 0.047168
0.016395 -0.083480
-0.086212 0.007569
-0.014850 -0.046606
0.002804 -0.093901
-0.030334 -0.020768
-0.017055 -0.052843
0.007000 -0.068055
0.038626 0.051675
0.004908 0.158678 -0.038023
0.036239 0.045048
0.043561 0.041629
0.082158 0.046833
-0.038689 0.001862
0.202852 0.072604 -0.146070
-0.193774 0.218794 -0.012935
0.103399 0.071321 0.044495
0.003013 -0.079276 0.015744
-0.004915 0.026026
0.011350 -0.009255
0.415079 -0.097741
0.004591 -0.040852 0.080207

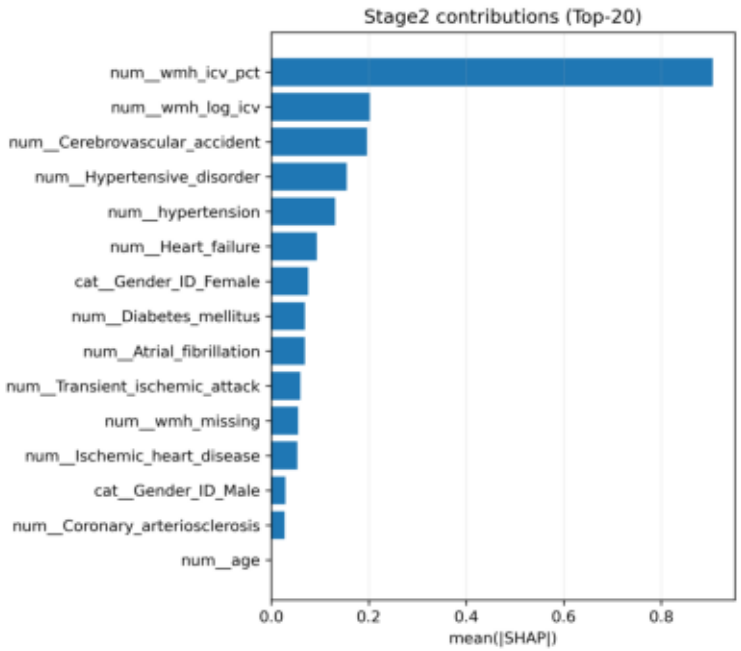
0.101731 0.020645 0.038286
-0.047609 0.007997
-0.020778 -0.023819
0.040256 0.018290
-0.119020 0.058191
-0.000018 0.115295
0.019247 0.012543 -0.132914
-0.020793 -0.004959 0.025903
0.017706 -0.015627 -0.049548 0.119588 -0.011165
0.008848 -0.024175 0.002158 -0.040904 0.0
0.083251 -0.000254 -0.022007 0.001217
0.0 -0.022406 0.0 -0.005479
-0.002036 0.033808 0.001934
0.024303 0.020598 -0.013647 0.000388
0.012219 0.000241 0.003610
-0.000108 0.0 0.000153 0.101773
-0.128140 -0.026489 0.000795
20 -0.028162
0.016257 0.019259
0.095330 -0.314392
0.119230 0.018588
-0.010414 0.104111
0.191387 -0.043178
-0.078410 0.042883
-0.274959 -0.019713
0.066617 -0.137149
-0.025816 -0.033809 -0.045535
0.021451 0.017430
-0.051702 -0.011398
0.138858 -0.050470
-0.044276 -0.050703
-0.048432 0.021056
0.213359 -0.003424 0.112354
-0.011462 -0.073881 -0.200598
0.030037 0.007877 0.029312
-0.017651 0.007423 0.090152
-0.171771 -0.014881
-0.021030 -0.085634
-0.278015 0.024889 0.011575
0.004330 -0.020076 -0.019792
0.011009 0.004126 -0.014641
-0.186376 -0.011973
0.023537 0.178532
-0.217667 0.008933
0.169156 0.018228
-0.350317 0.076841 0.056520
-0.008413 -0.019055 -0.016839
-0.014838 -0.044171 0.121466 0.085626 -0.042939 -0.009891

0.000135 0.004103 0.0 -0.496405
0.006897 0.023011 -0.004850 0.0
-0.035911 0.0 -0.000311 0.005153
-0.419104 0.002868 0.015937 0.011542
-0.018184 0.000255 0.021415
0.000241 0.001040 -0.000121
0.0 0.001482 0.015178 -0.097325 0.024521
0.007182
2197 0.034944
0.083222 0.060063
-0.116525 0.016164
0.107576 -0.024387
0.001446 0.003256
-0.050202 -0.019796
-0.001366 0.041300
0.011476 -0.001806
-0.003060 -0.083605
-0.025260 -0.025824 -0.027072
0.095004 -0.070873
0.044068 0.026996
-0.020919 0.008966
-0.047444 -0.011139
0.122259 0.042817 -0.064387
0.042349 -0.011475 0.047553
0.014952 0.120074 0.048191
0.003417 -0.002175 0.010953
-0.026762 -0.010429
0.059882 -0.025657
0.041734 -0.003657
-0.077244 0.028377 0.096056
0.036277 -0.037512 0.152744
0.014964 0.013123 -0.014347
0.015821 0.003974 0.047494
0.048975 0.033550
0.007351 0.017934
0.063196 -0.039389 0.062957
0.044780 0.004784 0.011402
-0.003528 -0.012218 0.167282 -0.167123 0.002220
-0.061773 -0.006455 0.000181 -0.020467 0.0
0.090515 -0.000254 0.038641 0.001613
0.0 -0.012063 0.0 -0.007332
0.001785 0.040587 0.002682
0.020673 -0.005472 -0.017140 0.000440
0.013637 0.000241 0.002053
-0.000108 0.0 0.001482 0.058006
-0.021367 -0.013837 -0.004504
2125 0.032695
-0.001140 0.048574

```
-0.122320                                        0.010575
-0.157058                                       -0.055247
0.006476                                         -0.024198
-0.004033                                         -0.054361
-0.001798                                  -0.090650
0.005080                                  -0.040494
-0.038136                                              0.045453
-0.012863                           -0.037800                          -0.062272
0.001405                                         -0.050698
0.026467                                          0.039803
-0.086003                                        0.018722
-0.053730                                         -0.004403
0.034906                              -0.011419                           -0.087098
0.044551                            -0.019152                           -0.009545
-0.048225                             0.195843                            -0.019403
0.005563                            -0.135147                             -0.038808
-0.030394                             -0.013683
-0.049435                              0.008123
0.200610                              -0.023889
-0.091178                     0.055712                      -0.013228
-0.058888                             0.029637                            -0.005206
0.060710                              0.013123                             -0.021744
-0.061050                              0.000642
-0.020826                             -0.241329
-0.124323                             -0.001091
0.159584                                0.075970
-0.005820                     0.049777            0.019224
-0.024588                       0.010413                  0.033451
-0.014392     -0.040721    -0.015724    -0.043229    -0.070458    -0.012003
-0.002050    0.007733                0.0                      -0.330416
-0.000503               -0.026581                      0.001572             0.0
-0.018617                      0.0     0.041194                     0.001785
0.035909       0.002664                                -0.190455        0.011779
0.109529    -0.003117                     0.015190
-0.038217                  0.003987                     -0.000108
0.0                       -0.008058 -0.043345  -0.000267              -0.053101
-0.010146

----------------------------------------
 : stage1_shap_values.csv
  : 2025-12-11 06:39:48
----------------------------------------

[CSV]   103   × 3     10

                                  feature  mean_abs_shap  mean_shap
                                num__MMSE       0.193538   0.009166
              num__Cerebrovascular_accident       0.144728   0.005919
```

```
            num__050_Right Inf Lat Vent__pct       0.140217   0.001491
      num__118_Left Ent entorhinal area__pct       0.134004  -0.007065
             num__051_Left Inf Lat Vent__pct       0.132246   0.001231
           num__Brain_Left_0_Frontal_99__pct       0.118064   0.004361
         num__Brain_Left_4_DeepNuclei_9__pct       0.092967   0.005101
              num__049_Left Hippocampus__pct       0.091276   0.007771
num__172_Left PHG parahippocampal gyrus__pct       0.086784  -0.001344
                                  num__LI_PCu       0.086562   0.004348

----------------------------------------
 : stage2_shap_bar.png
  : 2025-12-11 06:39:49
----------------------------------------
```

stage2_shap_bar.png

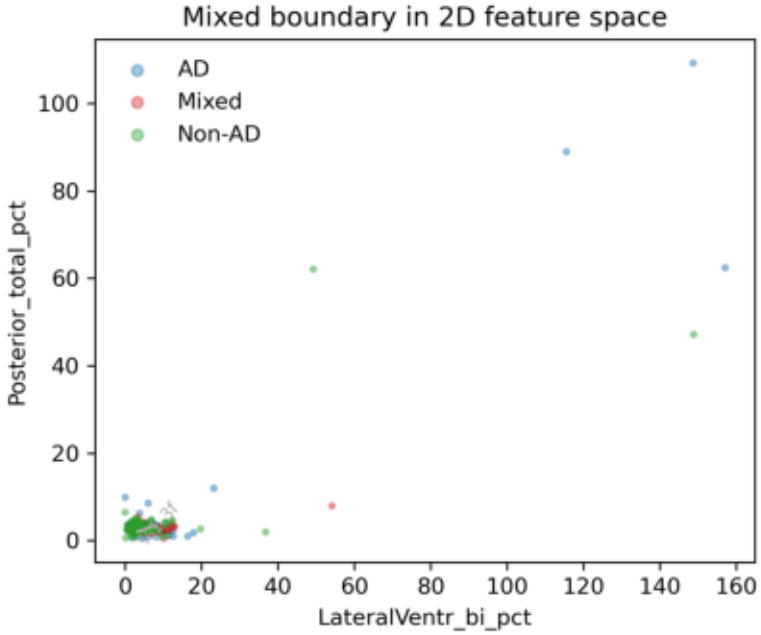



```
----------------------------------------
 : stage2_shap_samples.csv
  : 2025-12-11 06:39:49
----------------------------------------

[CSV]   1318   × 16      10

 sample_index_in_X  num__wmh_icv_pct  num__wmh_log_icv  num__wmh_missing
```

```
num__hypertension  num__Hypertensive_disorder  num__Diabetes_mellitus
num__Atrial_fibrillation  num__Ischemic_heart_disease
num__Cerebrovascular_accident  num__Transient_ischemic_attack
num__Heart_failure  num__Coronary_arteriosclerosis  num__age
cat__Gender_ID_Female  cat__Gender_ID_Male
               0          3.069893          0.545670         -0.012984
0.152238                  -0.136515                0.121644
-0.023649                   0.274102                       0.178153
-0.047183            0.575292                        0.408033       0.0
0.115974             0.052113
               1         -0.847595         -0.146412         -0.030086
-0.136204                  0.143394                0.036404
-0.053536                  -0.056558                      0.560236
0.349379           -0.091912                        -0.006170       0.0
-0.009646            -0.020595
               2          0.575597          0.197067         -0.029766
-0.131319                  0.122634                -0.030062
-0.053286                   0.007277                      -0.113992
-0.020116           -0.046308                        -0.028739       0.0
-0.019336             0.025279
               3         -0.679651         -0.016201         -0.019955
0.049933                  -0.215406                -0.021051
-0.074895                   -0.028163                     -0.162663
-0.046192           -0.046924                        -0.003415       0.0
0.032710            -0.025772
               4          1.742481          0.485864         -0.032974
-0.115195                  0.128924                0.080774
-0.053546                   0.024693                      -0.106026
-0.011531           -0.040631                        -0.027795       0.0
0.030277            -0.020672
               5          0.747782          0.096549         -0.033888
-0.155981                  0.108594                0.053536
-0.040164                   0.042554                      -0.141239
-0.009484           -0.042210                        -0.013010       0.0
-0.136344            -0.037059
               6          0.697491          0.089717         -0.027392
-0.146860                  0.075124                -0.178983
-0.019314                  -0.049149                      0.239779
-0.032204           -0.043487                        -0.023511       0.0
0.115507             0.039513
               7          1.601996          0.436930         -0.023142
0.106064                  0.149975                -0.047425
-0.047623                   0.037244                      -0.079909
-0.020372           -0.042874                        -0.026448       0.0
0.036821             0.038206
               8         -1.249282         -0.190036         -0.099193
0.130209                  -0.137611                0.129513
-0.030535                   -0.003799                     -0.185395
```

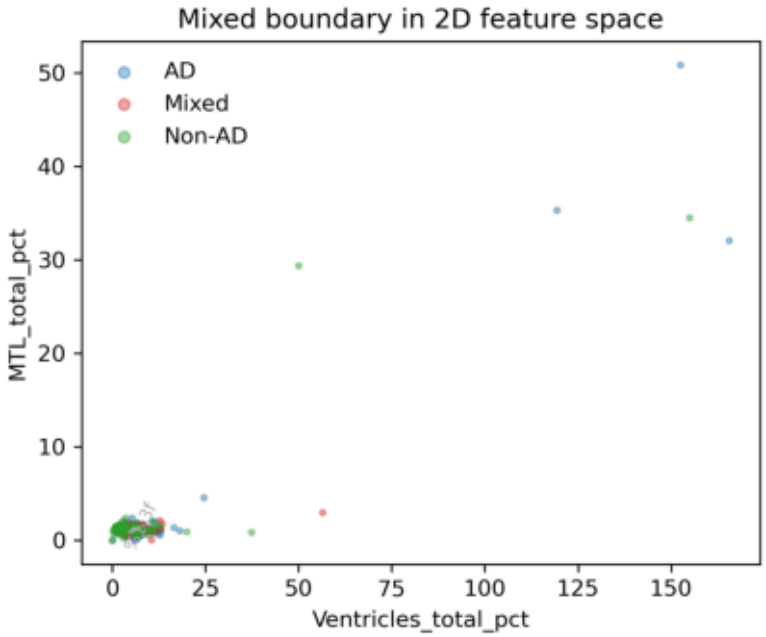
```
-0.047456           -0.038352                       -0.008922       0.0
0.099186             0.027035
                9         -1.479216         -0.288324         -0.032050
0.054810                  -0.094243                0.076457
-0.029136                   -0.030028                       -0.150663
-0.066874           -0.037551                       -0.009346       0.0
-0.010489            -0.020419

----------------------------------------
 : stage2_shap_values.csv
  : 2025-12-11 06:39:49
----------------------------------------

[CSV]   15   × 3

                      feature  mean_abs_shap  mean_shap
              num__wmh_icv_pct       0.906033   0.009294
              num__wmh_log_icv       0.202818   0.000422
 num__Cerebrovascular_accident       0.196147  -0.005705
    num__Hypertensive_disorder       0.154991  -0.011313
             num__hypertension       0.131173   0.008345
            num__Heart_failure       0.093152   0.001439
         cat__Gender_ID_Female       0.075931   0.001702
        num__Diabetes_mellitus       0.069000   0.001603
      num__Atrial_fibrillation       0.068532  -0.000436
num__Transient_ischemic_attack       0.059454  -0.000598
              num__wmh_missing       0.055096  -0.002030
   num__Ischemic_heart_disease       0.053870   0.000513
           cat__Gender_ID_Male       0.028831  -0.000288
num__Coronary_arteriosclerosis       0.027562   0.001512
                      num__age       0.000000   0.000000

--------------------------------------------------------------------------------
   : analysis_out/.ipynb_checkpoints
--------------------------------------------------------------------------------

[INFO]      24

--------------------------------------------------------------------------------
   : analysis_out/interaction_plots
--------------------------------------------------------------------------------



----------------------------------------
 : interact_2D_LateralVentr_bi_pct_Posterior_total_pct.png
  : 2025-12-11 06:40:21
----------------------------------------
```

interact_2D_LateralVentr_bi_pct_Posterior_total_pct.png



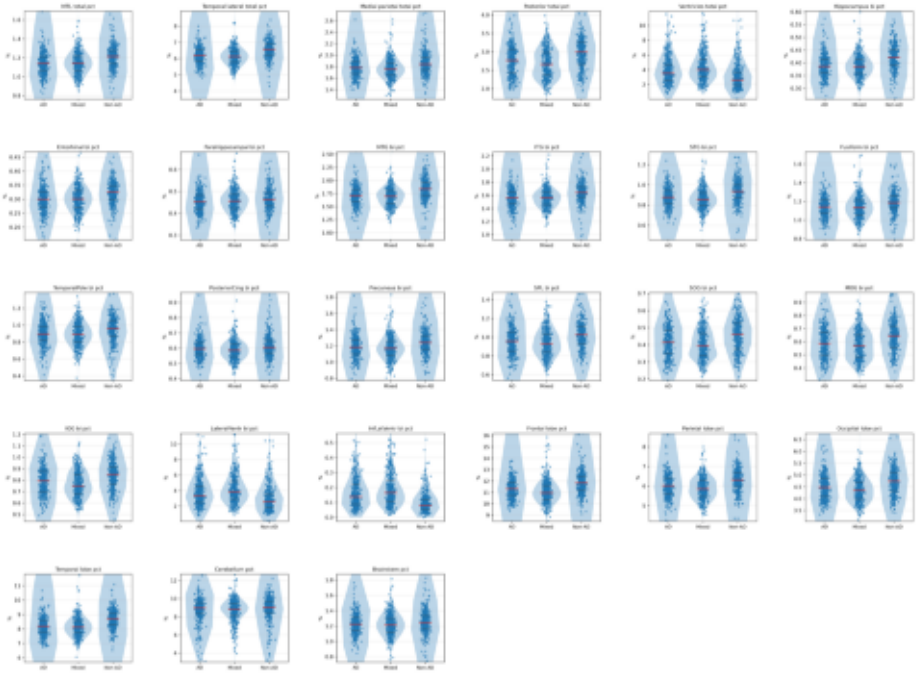

```
----------------------------------------
 : interact_2D_Ventricles_total_pct_MTL_total_pct.png
  : 2025-12-11 06:40:21
----------------------------------------
```

interact_2D_Ventricles_total_pct_MTL_total_pct.png



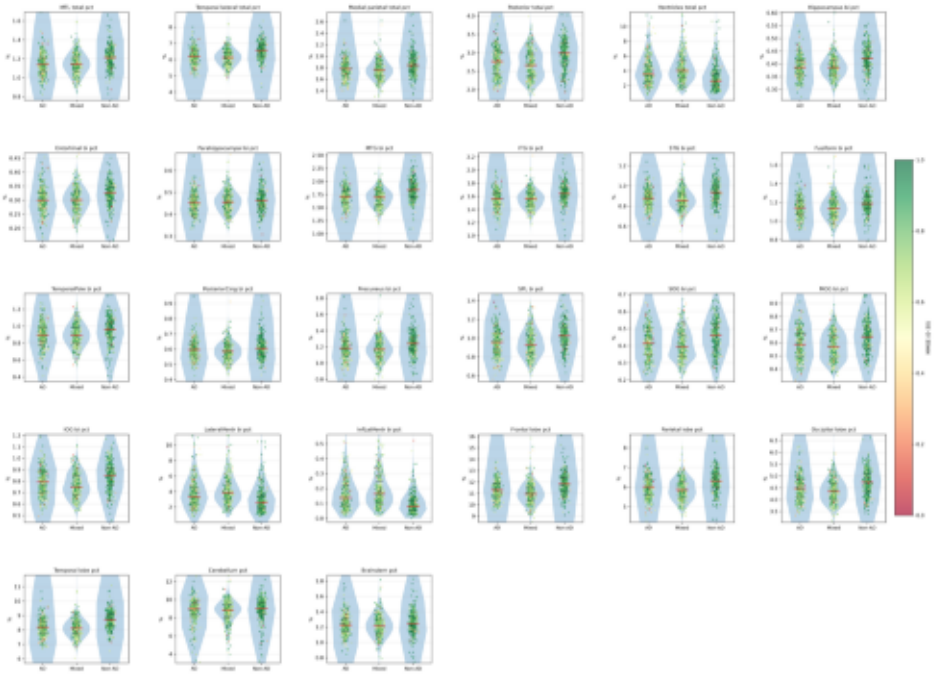

```
----------------------------------------------------------------------------------
   : analysis_out/interaction_plots/.ipynb_checkpoints
----------------------------------------------------------------------------------

[INFO]      24

----------------------------------------------------------------------------------
   : panels_final
----------------------------------------------------------------------------------


----------------------------------------
 : panel_roi_dist.png
  : 2025-12-11 06:40:00
----------------------------------------
```

panel_roi_dist.png



```
----------------------------------------
 : panel_roi_mmse_colored.png
  : 2025-12-11 06:40:05
----------------------------------------
```

panel_roi_mmse_colored.png



: panels_final/.ipynb_checkpoints

[INFO] 24

: fairness

: final3_ethnicity_perclass.csv

: 2025-12-11 06:30:28

[CSV] 57 × 7

ethnicity_code	ethnicity_label	class	precision	recall	f1	support
0	0	AD	0.921275	0.975198	0.947470	1008
0	0	Mixed	0.771812	0.660920	0.712074	348
0	0	Non-AD	0.760714	0.737024	0.748682	289
1	1	AD	0.958333	0.932432	0.945205	74

1	1	Mixed	0.571429	0.827586	0.676056	29
1	1	Non-AD	0.838710	0.619048	0.712329	42
2	2	AD	0.921053	0.933333	0.927152	75
2	2	Mixed	0.862069	0.757576	0.806452	33
2	2	Non-AD	0.652174	0.750000	0.697674	20
3	3	AD	0.930233	0.952381	0.941176	42
3	3	Mixed	0.692308	0.818182	0.750000	11
3	3	Non-AD	0.875000	0.636364	0.736842	11
4	4	AD	0.940741	0.954887	0.947761	133
4	4	Mixed	0.787879	0.702703	0.742857	37
4	4	Non-AD	0.807692	0.840000	0.823529	50
5	5	AD	0.959044	0.959044	0.959044	293
5	5	Mixed	0.773810	0.691489	0.730337	94
5	5	Non-AD	0.758929	0.833333	0.794393	102
6	6	AD	0.917647	0.951220	0.934132	82
6	6	Mixed	0.814815	0.687500	0.745763	32
6	6	Non-AD	0.666667	0.736842	0.700000	19
7	7	AD	1.000000	1.000000	1.000000	6
7	7	Mixed	0.000000	0.000000	0.000000	0
7	7	Non-AD	1.000000	0.500000	0.666667	4
8	8	AD	0.966102	0.966102	0.966102	59
8	8	Mixed	0.769231	0.526316	0.625000	19
8	8	Non-AD	0.640000	0.842105	0.727273	19
9	9	AD	0.939024	0.974684	0.956522	79
9	9	Mixed	0.523810	0.611111	0.564103	18
9	9	Non-AD	0.880000	0.709677	0.785714	31
10	10	AD	1.000000	1.000000	1.000000	17
10	10	Mixed	0.000000	0.000000	0.000000	0
10	10	Non-AD	1.000000	0.666667	0.800000	6
11	11	AD	0.960000	0.888889	0.923077	27
11	11	Mixed	0.714286	0.625000	0.666667	8
11	11	Non-AD	0.642857	0.818182	0.720000	11
12	12	AD	1.000000	0.800000	0.888889	10
12	12	Mixed	0.000000	0.000000	0.000000	0
12	12	Non-AD	1.000000	0.750000	0.857143	4
13	13	AD	0.900000	0.818182	0.857143	11
13	13	Mixed	0.333333	0.500000	0.400000	2
13	13	Non-AD	0.888889	0.888889	0.888889	9
14	14	AD	0.800000	1.000000	0.888889	4
14	14	Mixed	0.333333	0.333333	0.333333	3
14	14	Non-AD	0.857143	0.750000	0.800000	8
15	15	AD	1.000000	0.833333	0.909091	6
15	15	Mixed	0.000000	0.000000	0.000000	0
15	15	Non-AD	1.000000	0.500000	0.666667	2
16	16	AD	1.000000	1.000000	1.000000	4
16	16	Mixed	0.000000	0.000000	0.000000	0
16	16	Non-AD	0.000000	0.000000	0.000000	0
17	17	AD	0.000000	0.000000	0.000000	0

17	17	Mixed	0.000000	0.000000	0.000000	0
17	17	Non-AD	1.000000	0.500000	0.666667	2
18	18	AD	1.000000	1.000000	1.000000	1
18	18	Mixed	0.000000	0.000000	0.000000	0
18	18	Non-AD	0.000000	0.000000	0.000000	0

: final3_ethnicity_summary.csv
: 2025-12-11 06:30:28

[CSV] 19 × 6

ethnicity_code	ethnicity_label	n	accuracy	macro_recall	macro_f1
0	0	1645	0.866869	0.791047	0.802742
1	1	145	0.820690	0.793022	0.777864
2	2	128	0.859375	0.813636	0.810426
3	3	64	0.875000	0.802309	0.809340
4	4	220	0.886364	0.832530	0.838049
5	5	489	0.881391	0.827956	0.827925
6	6	133	0.857143	0.791854	0.793298
7	7	10	0.800000	0.500000	0.555556
8	8	97	0.855670	0.778174	0.772791
9	9	128	0.859375	0.765157	0.768780
10	10	23	0.913043	0.555556	0.600000
11	11	46	0.826087	0.777357	0.769915
12	12	14	0.785714	0.516667	0.582011
13	13	22	0.818182	0.735690	0.715344
14	14	15	0.733333	0.694444	0.674074
15	15	8	0.750000	0.444444	0.525253
16	16	4	1.000000	0.333333	0.333333
17	17	2	0.500000	0.166667	0.222222
18	18	1	1.000000	0.333333	0.333333

: final3_gender_perclass.csv
: 2025-12-11 06:30:28

[CSV] 6 × 7

gender_code	gender_label	class	precision	recall	f1	support
0	Gender_0	AD	0.947771	0.961240	0.954458	774
0	Gender_0	Mixed	0.752768	0.733813	0.743169	278
0	Gender_0	Non-AD	0.768683	0.757895	0.763251	285
1	Gender_1	AD	0.925809	0.963054	0.944064	1218
1	Gender_1	Mixed	0.709091	0.637602	0.671449	367
1	Gender_1	Non-AD	0.779255	0.755155	0.767016	388

```
----------------------------------------
 : final3_gender_summary.csv
  : 2025-12-11 06:30:28
----------------------------------------

[CSV]   2   × 6

 gender_code gender_label    n  accuracy  macro_recall  macro_f1
           0     Gender_0 1337  0.870606      0.817649  0.820293
           1     Gender_1 1973  0.861632      0.785270  0.794176

----------------------------------------
 : stage1_stage1_ethnicity.csv
  : 2025-12-11 06:30:28
----------------------------------------

[CSV]   19   × 9

 ethnicity_code  ethnicity_label    n  accuracy  precision_pos  recall_pos
f1_pos  balanced_acc  roc_auc
              0                0 1645  0.933739       0.921275    0.975198
0.947470      0.921665 0.986122
              1                1  145  0.944828       0.958333    0.932432
0.945205      0.945089 0.983251
              2                2  128  0.914062       0.921053    0.933333
0.927152      0.910063 0.986415
              3                3   64  0.921875       0.930233    0.952381
0.941176      0.908009 0.965368
              4                4  220  0.936364       0.940741    0.954887
0.947761      0.931467 0.983666
              5                5  489  0.950920       0.959044    0.959044
0.959044      0.948910 0.987915
              6                6  133  0.917293       0.917647    0.951220
0.934132      0.906982 0.981110
              7                7   10  1.000000       1.000000    1.000000
1.000000      1.000000 1.000000
              8                8   97  0.958763       0.966102    0.966102
0.966102      0.956735 0.989741
              9                9  128  0.945312       0.939024    0.974684
0.956522      0.936321 0.994833
             10               10   23  1.000000       1.000000    1.000000
1.000000      1.000000 1.000000
             11               11   46  0.913043       0.960000    0.888889
0.923077      0.918129 0.986355
             12               12   14  0.857143       1.000000    0.800000
0.888889      0.900000 1.000000
             13               13   22  0.863636       0.900000    0.818182
```

```
0.857143      0.863636 0.950413
             14                14   15  0.933333       0.800000    1.000000
0.888889      0.954545 1.000000
             15                15    8  0.875000       1.000000    0.833333
0.909091      0.916667 1.000000
             16                16    4  1.000000       1.000000    1.000000
1.000000      1.000000      NaN
             17                17    2  1.000000            NaN         NaN
NaN      1.000000      NaN
             18                18    1  1.000000       1.000000    1.000000
1.000000      1.000000      NaN

----------------------------------------
 : stage1_stage1_gender.csv
  : 2025-12-11 06:30:28
----------------------------------------

[CSV]   2   × 9

 gender_code gender_label    n  accuracy  precision_pos  recall_pos   f1_pos
balanced_acc  roc_auc
           0     Gender_0 1337  0.946896       0.947771    0.961240 0.954458
0.944208 0.988893
           1     Gender_1 1973  0.929549       0.925809    0.963054 0.944064
0.919275 0.983137

----------------------------------------
 : stage2_stage2_ethnicity.csv
  : 2025-12-11 06:30:28
----------------------------------------

[CSV]   17   × 9

 ethnicity_code  ethnicity_label   n  accuracy  precision_pos  recall_pos
f1_pos  balanced_acc  roc_auc
              0                0 637  0.511774       0.566308    0.454023
0.503987      0.517669 0.518504
              1                1  71  0.577465       0.487179    0.655172
0.558824      0.589491 0.627258
              2                2  53  0.509434       0.666667    0.424242
0.518519      0.537121 0.587121
              3                3  22  0.500000       0.500000    0.454545
0.476190      0.500000 0.471074
              4                4  87  0.540230       0.461538    0.486486
0.473684      0.533243 0.484595
              5                5 196  0.540816       0.520833    0.531915
0.526316      0.540467 0.503598
              6                6  51  0.470588       0.608696    0.437500
```

```
0.509091      0.481908 0.490132
             7                7   4  0.250000       0.000000         NaN
NaN      0.250000      NaN
             8                8  38  0.421053       0.421053    0.421053
0.421053      0.421053 0.396122
             9                9  49  0.551020       0.409091    0.500000
0.450000      0.540323 0.603943
            10               10   6  0.833333       0.000000         NaN
NaN      0.833333      NaN
            11               11  19  0.368421       0.250000    0.250000
0.250000      0.352273 0.250000
            12               12   4  0.750000       0.000000         NaN
NaN      0.750000      NaN
            13               13  11  0.636364       0.000000    0.000000
NaN      0.388889 0.444444
            14               14  11  0.454545       0.000000    0.000000
NaN      0.312500 0.375000
            15               15   2  0.500000       0.000000         NaN
NaN      0.500000      NaN
            17               17   2  0.000000       0.000000         NaN
NaN      0.000000      NaN

----------------------------------------
 : stage2_stage2_gender.csv
  : 2025-12-11 06:30:28
----------------------------------------

[CSV]   2   × 9

 gender_code gender_label   n  accuracy  precision_pos  recall_pos   f1_pos
balanced_acc  roc_auc
           0     Gender_0 563  0.507993       0.502075    0.435252 0.466281
0.507100 0.519658
           1     Gender_1 755  0.521854       0.508523    0.487738 0.497914
0.520931 0.503810

--------------------------------------------------------------------------------
   : fairness_WMH
--------------------------------------------------------------------------------

[INFO]      24

################################################################################
#   : hier_dir        24h  (hier_out)
################################################################################


--------------------------------------------------------------------------------
```

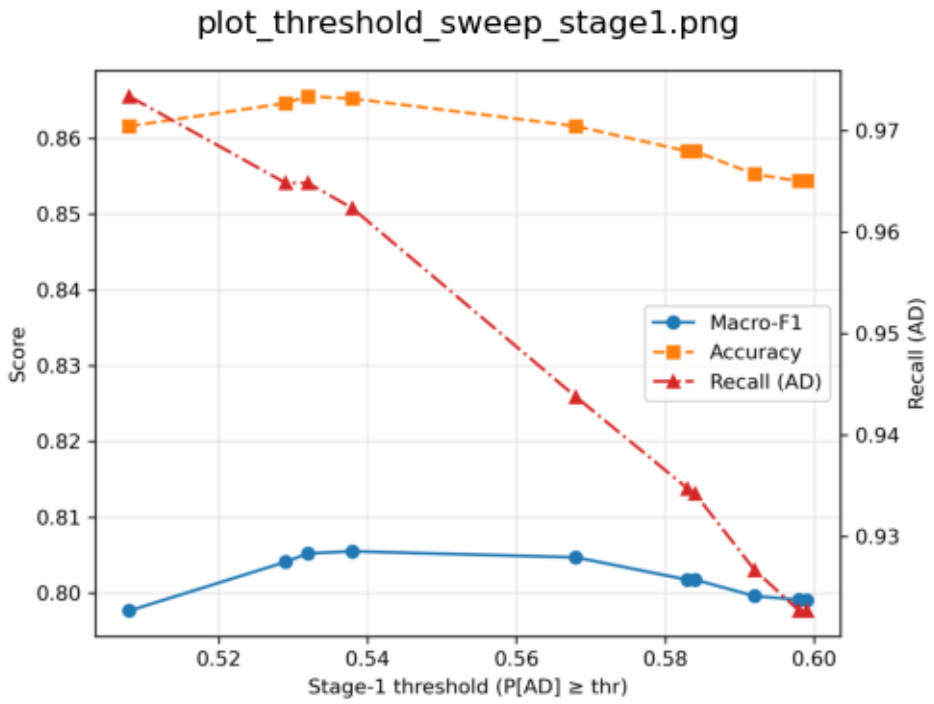
```
   : (root)
--------------------------------------------------------------------------------


----------------------------------------
 : feature_list_stage2.csv
  : 2025-12-11 06:39:37
----------------------------------------

[CSV]   14   × 1

             feature_name
              wmh_icv_pct
              wmh_log_icv
              wmh_missing
             hypertension
    Hypertensive_disorder
        Diabetes_mellitus
      Atrial_fibrillation
   Ischemic_heart_disease
 Cerebrovascular_accident
Transient_ischemic_attack
            Heart_failure
Coronary_arteriosclerosis
                      age
                Gender_ID

----------------------------------------
 : final_confusion_matrix.csv
  : 2025-12-11 06:39:26
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1946          17           29
 true_Mixed      137         302          206
true_Non-AD       52         226          395

----------------------------------------
 : final_confusion_matrix_thr0.50.csv
  : 2025-12-11 06:39:40
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1946          19           27
```

```
 true_Mixed      137         407          101
true_Non-AD       52         130          491

----------------------------------------
 : final_confusion_matrix_thr0.508.csv
  : 2025-12-11 06:39:42
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1939          23           30
 true_Mixed      125         416          104
true_Non-AD       46         130          497

----------------------------------------
 : final_confusion_matrix_thr0.529.csv
  : 2025-12-11 06:39:42
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1922          32           38
 true_Mixed      107         433          105
true_Non-AD       36         130          507

----------------------------------------
 : final_confusion_matrix_thr0.532.csv
  : 2025-12-11 06:39:42
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1922          32           38
 true_Mixed      104         435          106
true_Non-AD       35         130          508

----------------------------------------
 : final_confusion_matrix_thr0.538.csv
  : 2025-12-11 06:39:42
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1917          33           42
```

```
 true_Mixed      101          438           106
true_Non-AD       34          130           509

----------------------------------------
 : final_confusion_matrix_thr0.55.csv
  : 2025-12-11 06:39:40
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1906          36           50
 true_Mixed       92         445          108
true_Non-AD       34         130          509

----------------------------------------
 : final_confusion_matrix_thr0.568.csv
  : 2025-12-11 06:39:42
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1880          47           65
 true_Mixed       75         456          114
true_Non-AD       27         130          516

----------------------------------------
 : final_confusion_matrix_thr0.583.csv
  : 2025-12-11 06:39:43
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1862          60           70
 true_Mixed       65         459          121
true_Non-AD       22         131          520

----------------------------------------
 : final_confusion_matrix_thr0.584.csv
  : 2025-12-11 06:39:43
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1861          60           71
```

```
 true_Mixed       64         459          122
true_Non-AD       21         131          521

----------------------------------------
 : final_confusion_matrix_thr0.592.csv
  : 2025-12-11 06:39:43
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1846          67           79
 true_Mixed       60         462          123
true_Non-AD       19         131          523

----------------------------------------
 : final_confusion_matrix_thr0.598.csv
  : 2025-12-11 06:39:43
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1838          70           84
 true_Mixed       56         464          125
true_Non-AD       15         132          526

----------------------------------------
 : final_confusion_matrix_thr0.599.csv
  : 2025-12-11 06:39:43
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1838          70           84
 true_Mixed       55         464          126
true_Non-AD       15         132          526

----------------------------------------
 : final_confusion_matrix_thr0.60.csv
  : 2025-12-11 06:39:41
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1836          70           86
```

```
 true_Mixed       55         464          126
true_Non-AD       15         132          526

----------------------------------------
 : final_confusion_matrix_thr0.65.csv
  : 2025-12-11 06:39:41
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1738         111          143
 true_Mixed       25         490          130
true_Non-AD        6         133          534

----------------------------------------
 : final_confusion_matrix_thr0.70.csv
  : 2025-12-11 06:39:41
----------------------------------------

[CSV]   3   × 4

 Unnamed: 0  pred_AD  pred_Mixed  pred_Non-AD
    true_AD     1559         192          241
 true_Mixed       11         497          137
true_Non-AD        2         134          537

----------------------------------------
 : plot_threshold_sweep_stage1.png
  : 2025-12-11 06:39:43
----------------------------------------
```




```
----------------------------------------
 : preds_hierarchical.csv
  : 2025-12-11 06:39:26
----------------------------------------

[CSV]   3310   × 5     10

   BrcId true_diag     P_AD final_pred  P_Mixed_given_nonAD
10309796        AD 0.828000         AD                  NaN
10415039     Mixed 0.149271      Mixed             0.718330
10331478    Non-AD 0.178709      Mixed             0.661639
10426976        AD 0.751963         AD                  NaN
10019261     Mixed 0.244223     Non-AD             0.445762
10425175    Non-AD 0.243892     Non-AD             0.419252
10432969        AD 0.699629         AD                  NaN
10433396        AD 0.516806         AD                  NaN
10426958     Mixed 0.135618     Non-AD             0.448564
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.50.csv
  : 2025-12-11 06:39:40
----------------------------------------
```

[CSV] 3310 × 5 10

BrcId	true_diag	P_AD	final_pred	P_Mixed_given_nonAD
10309796	AD	0.828000	AD	NaN
10415039	Mixed	0.149271	Mixed	0.994657
10331478	Non-AD	0.178709	Non-AD	0.412081
10426976	AD	0.751963	AD	NaN
10019261	Mixed	0.244223	Mixed	0.600816
10425175	Non-AD	0.243892	Non-AD	0.213541
10432969	AD	0.699629	AD	NaN
10433396	AD	0.516806	AD	NaN
10426958	Mixed	0.135618	Mixed	0.884788
10418222	AD	0.705930	AD	NaN

: preds_hierarchical_thr0.508.csv

: 2025-12-11 06:39:42

[CSV] 3310 × 5 10

BrcId	true_diag	P_AD	final_pred	P_Mixed_given_nonAD
10309796	AD	0.828000	AD	NaN
10415039	Mixed	0.149271	Mixed	0.994657
10331478	Non-AD	0.178709	Non-AD	0.412081
10426976	AD	0.751963	AD	NaN
10019261	Mixed	0.244223	Mixed	0.600816
10425175	Non-AD	0.243892	Non-AD	0.213541
10432969	AD	0.699629	AD	NaN
10433396	AD	0.516806	AD	NaN
10426958	Mixed	0.135618	Mixed	0.884788
10418222	AD	0.705930	AD	NaN

: preds_hierarchical_thr0.529.csv

: 2025-12-11 06:39:42

[CSV] 3310 × 5 10

BrcId	true_diag	P_AD	final_pred	P_Mixed_given_nonAD
10309796	AD	0.828000	AD	NaN
10415039	Mixed	0.149271	Mixed	0.994657
10331478	Non-AD	0.178709	Non-AD	0.412081
10426976	AD	0.751963	AD	NaN
10019261	Mixed	0.244223	Mixed	0.600816
10425175	Non-AD	0.243892	Non-AD	0.213541

```
10432969        AD 0.699629         AD                  NaN
10433396        AD 0.516806     Non-AD             0.040153
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.532.csv
  : 2025-12-11 06:39:42
----------------------------------------

[CSV]   3310   × 5     10

   BrcId true_diag     P_AD final_pred  P_Mixed_given_nonAD
10309796        AD 0.828000         AD                  NaN
10415039     Mixed 0.149271      Mixed             0.994657
10331478    Non-AD 0.178709     Non-AD             0.412081
10426976        AD 0.751963         AD                  NaN
10019261     Mixed 0.244223      Mixed             0.600816
10425175    Non-AD 0.243892     Non-AD             0.213541
10432969        AD 0.699629         AD                  NaN
10433396        AD 0.516806     Non-AD             0.040153
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.538.csv
  : 2025-12-11 06:39:42
----------------------------------------

[CSV]   3310   × 5     10

   BrcId true_diag     P_AD final_pred  P_Mixed_given_nonAD
10309796        AD 0.828000         AD                  NaN
10415039     Mixed 0.149271      Mixed             0.994657
10331478    Non-AD 0.178709     Non-AD             0.412081
10426976        AD 0.751963         AD                  NaN
10019261     Mixed 0.244223      Mixed             0.600816
10425175    Non-AD 0.243892     Non-AD             0.213541
10432969        AD 0.699629         AD                  NaN
10433396        AD 0.516806     Non-AD             0.040153
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.55.csv
  : 2025-12-11 06:39:40
----------------------------------------
```

[CSV] 3310 × 5 10

BrcId	true_diag	P_AD	final_pred	P_Mixed_given_nonAD
10309796	AD	0.828000	AD	NaN
10415039	Mixed	0.149271	Mixed	0.994657
10331478	Non-AD	0.178709	Non-AD	0.412081
10426976	AD	0.751963	AD	NaN
10019261	Mixed	0.244223	Mixed	0.600816
10425175	Non-AD	0.243892	Non-AD	0.213541
10432969	AD	0.699629	AD	NaN
10433396	AD	0.516806	Non-AD	0.040153
10426958	Mixed	0.135618	Mixed	0.884788
10418222	AD	0.705930	AD	NaN

: preds_hierarchical_thr0.568.csv

: 2025-12-11 06:39:43

[CSV] 3310 × 5 10

BrcId	true_diag	P_AD	final_pred	P_Mixed_given_nonAD
10309796	AD	0.828000	AD	NaN
10415039	Mixed	0.149271	Mixed	0.994657
10331478	Non-AD	0.178709	Non-AD	0.412081
10426976	AD	0.751963	AD	NaN
10019261	Mixed	0.244223	Mixed	0.600816
10425175	Non-AD	0.243892	Non-AD	0.213541
10432969	AD	0.699629	AD	NaN
10433396	AD	0.516806	Non-AD	0.040153
10426958	Mixed	0.135618	Mixed	0.884788
10418222	AD	0.705930	AD	NaN

: preds_hierarchical_thr0.583.csv

: 2025-12-11 06:39:43

[CSV] 3310 × 5 10

BrcId	true_diag	P_AD	final_pred	P_Mixed_given_nonAD
10309796	AD	0.828000	AD	NaN
10415039	Mixed	0.149271	Mixed	0.994657
10331478	Non-AD	0.178709	Non-AD	0.412081
10426976	AD	0.751963	AD	NaN
10019261	Mixed	0.244223	Mixed	0.600816
10425175	Non-AD	0.243892	Non-AD	0.213541
10432969	AD	0.699629	AD	NaN

```
10433396        AD 0.516806     Non-AD             0.040153
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.584.csv
  : 2025-12-11 06:39:43
----------------------------------------

[CSV]   3310   × 5     10

   BrcId true_diag     P_AD final_pred  P_Mixed_given_nonAD
10309796        AD 0.828000         AD                  NaN
10415039     Mixed 0.149271      Mixed             0.994657
10331478    Non-AD 0.178709     Non-AD             0.412081
10426976        AD 0.751963         AD                  NaN
10019261     Mixed 0.244223      Mixed             0.600816
10425175    Non-AD 0.243892     Non-AD             0.213541
10432969        AD 0.699629         AD                  NaN
10433396        AD 0.516806     Non-AD             0.040153
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.592.csv
  : 2025-12-11 06:39:43
----------------------------------------

[CSV]   3310   × 5     10

   BrcId true_diag     P_AD final_pred  P_Mixed_given_nonAD
10309796        AD 0.828000         AD                  NaN
10415039     Mixed 0.149271      Mixed             0.994657
10331478    Non-AD 0.178709     Non-AD             0.412081
10426976        AD 0.751963         AD                  NaN
10019261     Mixed 0.244223      Mixed             0.600816
10425175    Non-AD 0.243892     Non-AD             0.213541
10432969        AD 0.699629         AD                  NaN
10433396        AD 0.516806     Non-AD             0.040153
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.598.csv
  : 2025-12-11 06:39:43
----------------------------------------

[CSV]   3310   × 5     10
```

```
   BrcId true_diag     P_AD final_pred  P_Mixed_given_nonAD
10309796        AD 0.828000         AD                  NaN
10415039     Mixed 0.149271      Mixed             0.994657
10331478    Non-AD 0.178709     Non-AD             0.412081
10426976        AD 0.751963         AD                  NaN
10019261     Mixed 0.244223      Mixed             0.600816
10425175    Non-AD 0.243892     Non-AD             0.213541
10432969        AD 0.699629         AD                  NaN
10433396        AD 0.516806     Non-AD             0.040153
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.599.csv
  : 2025-12-11 06:39:43
----------------------------------------

[CSV]   3310   × 5     10

   BrcId true_diag     P_AD final_pred  P_Mixed_given_nonAD
10309796        AD 0.828000         AD                  NaN
10415039     Mixed 0.149271      Mixed             0.994657
10331478    Non-AD 0.178709     Non-AD             0.412081
10426976        AD 0.751963         AD                  NaN
10019261     Mixed 0.244223      Mixed             0.600816
10425175    Non-AD 0.243892     Non-AD             0.213541
10432969        AD 0.699629         AD                  NaN
10433396        AD 0.516806     Non-AD             0.040153
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.60.csv
  : 2025-12-11 06:39:41
----------------------------------------

[CSV]   3310   × 5     10

   BrcId true_diag     P_AD final_pred  P_Mixed_given_nonAD
10309796        AD 0.828000         AD                  NaN
10415039     Mixed 0.149271      Mixed             0.994657
10331478    Non-AD 0.178709     Non-AD             0.412081
10426976        AD 0.751963         AD                  NaN
10019261     Mixed 0.244223      Mixed             0.600816
10425175    Non-AD 0.243892     Non-AD             0.213541
10432969        AD 0.699629         AD                  NaN
10433396        AD 0.516806     Non-AD             0.040153
```

```
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.65.csv
  : 2025-12-11 06:39:41
----------------------------------------

[CSV]   3310   × 5     10

   BrcId true_diag     P_AD final_pred  P_Mixed_given_nonAD
10309796        AD 0.828000         AD                  NaN
10415039     Mixed 0.149271      Mixed             0.994657
10331478    Non-AD 0.178709     Non-AD             0.412081
10426976        AD 0.751963         AD                  NaN
10019261     Mixed 0.244223      Mixed             0.600816
10425175    Non-AD 0.243892     Non-AD             0.213541
10432969        AD 0.699629         AD                  NaN
10433396        AD 0.516806     Non-AD             0.040153
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : preds_hierarchical_thr0.70.csv
  : 2025-12-11 06:39:41
----------------------------------------

[CSV]   3310   × 5     10

   BrcId true_diag     P_AD final_pred  P_Mixed_given_nonAD
10309796        AD 0.828000         AD                  NaN
10415039     Mixed 0.149271      Mixed             0.994657
10331478    Non-AD 0.178709     Non-AD             0.412081
10426976        AD 0.751963         AD                  NaN
10019261     Mixed 0.244223      Mixed             0.600816
10425175    Non-AD 0.243892     Non-AD             0.213541
10432969        AD 0.699629     Non-AD             0.474046
10433396        AD 0.516806     Non-AD             0.040153
10426958     Mixed 0.135618      Mixed             0.884788
10418222        AD 0.705930         AD                  NaN

----------------------------------------
 : stage1_metrics.json
  : 2025-12-11 06:39:37
----------------------------------------

{
  "f1": 0.6453259104616098,
```

```
  "balanced_acc": 0.49709802792353025,
  "auc": 0.4974893885100341
}

----------------------------------------
 : stage1_report.txt
  : 2025-12-11 06:39:37
----------------------------------------

              precision    recall  f1-score   support

           0      0.394     0.296     0.338      1318
           1      0.600     0.698     0.645      1992

    accuracy                          0.538      3310
   macro avg      0.497     0.497     0.492      3310
weighted avg      0.518     0.538     0.523      3310


----------------------------------------
 : stage2_best_params.json
  : 2025-12-11 06:39:24
----------------------------------------

{
  "clf__subsample": 0.7,
  "clf__scale_pos_weight": 1.0434108527131782,
  "clf__reg_lambda": 1.0,
  "clf__reg_alpha": 0.5,
  "clf__n_estimators": 900,
  "clf__min_child_weight": 7,
  "clf__max_depth": 2,
  "clf__learning_rate": 0.03,
  "clf__gamma": 1.0,
  "clf__colsample_bytree": 0.6
}

----------------------------------------
 : stage2_metrics.json
  : 2025-12-11 06:39:38
----------------------------------------

{
  "f1": 0.5093167701863354,
  "balanced_acc": 0.5202368199776541,
  "auc": 0.516654572261193
}
```

```
----------------------------------------
 : stage2_report.txt
  : 2025-12-11 06:39:38
----------------------------------------

              precision    recall  f1-score   support

           0      0.530     0.532     0.531       673
           1      0.510     0.509     0.509       645

    accuracy                         0.520      1318
   macro avg      0.520     0.520     0.520      1318
weighted avg      0.520     0.520     0.520      1318


----------------------------------------
 : threshold_sweep_stage1_summary.csv
  : 2025-12-11 06:39:43
----------------------------------------

[CSV]   10   × 4

 thr1  macro_f1  accuracy  recall_AD
0.538  0.805512  0.865257   0.962349
0.532  0.805236  0.865559   0.964859
0.568  0.804717  0.861631   0.943775
0.529  0.804108  0.864653   0.964859
0.584  0.801764  0.858308   0.934237
0.583  0.801754  0.858308   0.934739
0.592  0.799605  0.855287   0.926707
0.598  0.799113  0.854381   0.922691
0.599  0.799017  0.854381   0.922691
0.508  0.797667  0.861631   0.973394

--------------------------------------------------------------------------------
   : .ipynb_checkpoints
--------------------------------------------------------------------------------

[INFO]      24
[WARN]       surv_out  /     24h  (None)

================================================================================
   24
================================================================================
```

[38]:
```
# =========            =========

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from pathlib import Path

OUT = Path("surv_out") / "analysis_out"

OUT.mkdir(parents=True, exist_ok=True)

#      =true =pred

labels = ["AD", "Mixed", "Non-AD"]

cm = np.array([

    [1917,  33,  42],   # true_AD

    [ 101, 438, 106],   # true_Mixed

    [  34, 130, 509]    # true_Non-AD

], dtype=int)

cm_df = pd.DataFrame(cm, index=[f"true_{l}" for l in labels],

                         columns=[f"pred_{l}" for l in labels])

print("Confusion matrix:\n", cm_df)

#

support = cm.sum(axis=1)

total   = cm.sum()

#   Precision / Recall / F1

per_cls_rows = []

for i, lab in enumerate(labels):
```

```
    TP = cm[i, i]
    FN = cm[i, :].sum() - TP
    FP = cm[:, i].sum() - TP
    TN = total - TP - FN - FP
    prec = TP / (TP + FP) if (TP + FP) > 0 else 0.0
    rec  = TP / (TP + FN) if (TP + FN) > 0 else 0.0
    f1   = 2 * prec * rec / (prec + rec) if (prec + rec) > 0 else 0.0
    per_cls_rows.append({
        "class": lab,
        "support": int(support[i]),
        "precision": float(prec),
        "recall": float(rec),
        "f1": float(f1)
    })
per_cls = pd.DataFrame(per_cls_rows)
per_cls.set_index("class", inplace=True)
print("\nPer-class metrics:\n", per_cls)
#
macro_precision = per_cls["precision"].mean()
macro_recall    = per_cls["recall"].mean()
macro_f1        = per_cls["f1"].mean()
weighted_f1     = (per_cls["f1"] * per_cls["support"]).sum() / support.sum()
balanced_acc    = macro_recall
```

```
overall_acc     = np.trace(cm) / total

overall_rows = [{

    "accuracy": float(overall_acc),

    "balanced_accuracy": float(balanced_acc),

    "macro_precision": float(macro_precision),

    "macro_recall": float(macro_recall),

    "macro_f1": float(macro_f1),

    "weighted_f1": float(weighted_f1),

    "N": int(total)

}]

overall = pd.DataFrame(overall_rows)

print("\nOverall:\n", overall)

#

cm_df.to_csv(OUT / "overall_from_cm_confusion.csv", index=True)

per_cls.to_csv(OUT / "overall_from_cm_per_class.csv", index=True)

overall.to_csv(OUT / "overall_from_cm_summary.csv", index=False)

print("\n[Saved]")

print(OUT / "overall_from_cm_confusion.csv")

print(OUT / "overall_from_cm_per_class.csv")

print(OUT / "overall_from_cm_summary.csv")

#

norm_cm = cm / support.reshape(-1, 1)

plt.figure(figsize=(5.2, 4.4))

sns.heatmap(norm_cm, annot=True, fmt=".2f", cmap="Blues",
```

```
            xticklabels=labels, yticklabels=labels)

plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Normalized confusion␣
 ↪(row-wise)")

plt.tight_layout()

plt.savefig(OUT / "overall_from_cm_heatmap.png", dpi=300)

plt.close()

print(OUT / "overall_from_cm_heatmap.png")
```

```
Confusion matrix:
              pred_AD  pred_Mixed  pred_Non-AD
true_AD          1917          33           42
true_Mixed        101         438          106
true_Non-AD        34         130          509

Per-class metrics:
          support  precision    recall        f1
class
AD         1992   0.934211  0.962349  0.948071
Mixed       645   0.728785  0.679070  0.703050
Non-AD      673   0.774734  0.756315  0.765414

Overall:
    accuracy  balanced_accuracy  macro_precision  macro_recall  macro_f1  \
0  0.865257           0.799245         0.812577      0.799245  0.805512

   weighted_f1     N
0     0.863187  3310

[Saved]
surv_out/analysis_out/overall_from_cm_confusion.csv
surv_out/analysis_out/overall_from_cm_per_class.csv
surv_out/analysis_out/overall_from_cm_summary.csv
surv_out/analysis_out/overall_from_cm_heatmap.png
```

[ ]:

[39]:
```
!jupyter nbconvert --to pdf baseline_dignosis.ipynb
```

```
[NbConvertApp] Converting notebook baseline_dignosis.ipynb to pdf
[NbConvertApp] Support files will be in baseline_dignosis_files/
```

```
[NbConvertApp] Making directory ./baseline_dignosis_files
[NbConvertApp] Writing 804643 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 2511765 bytes to baseline_dignosis.pdf
```

[ ]:

[40]:
```
!pip install python-pptx
```

```
Requirement already satisfied: python-pptx in /opt/conda/lib/python3.11/site-
packages (1.0.2)
Requirement already satisfied: Pillow>=3.3.2 in /opt/conda/lib/python3.11/site-
packages (from python-pptx) (10.1.0)
Requirement already satisfied: XlsxWriter>=0.5.7 in
/opt/conda/lib/python3.11/site-packages (from python-pptx) (3.2.9)
Requirement already satisfied: lxml>=3.1.0 in /opt/conda/lib/python3.11/site-
packages (from python-pptx) (6.0.2)
Requirement already satisfied: typing-extensions>=4.9.0 in
/opt/conda/lib/python3.11/site-packages (from python-pptx) (4.15.0)
```

[41]:
```
import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import cross_val_predict, StratifiedGroupKFold

from sklearn.calibration import calibration_curve, CalibratedClassifierCV

from sklearn.metrics import brier_score_loss

import numpy as np

import pandas as pd

# ============================================================================

# Daniel's diagnostic module:

# 1. Probability histogram (bimodality check)

# 2. Calibration curve comparison (Raw vs Isotonic)
```

```
# 3. Misclassification analysis (missing data & threshold proximity)

# ================================================================================

def daniel_diagnostic_suite(model, X, y, groups, df_raw, stage_name,␣
 ↪best_thresh=0.5):

    print(f"\n{'='*50}")

    print(f"Running diagnostics for: {stage_name} (Threshold = {best_thresh})")

    print(f"{'='*50}")

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

    # A. Obtain out-of-fold prediction probabilities

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

    print("[*] Computing cross-validated prediction probabilities...")

    cv = StratifiedGroupKFold(n_splits=5, shuffle=True, random_state=42)

    try:

        y_prob = cross_val_predict(

            model, X, y, groups=groups, cv=cv,

            method="predict_proba", n_jobs=-1

        )[:, 1]

    except Exception as e:

        print(f"[!] CV prediction failed. Falling back to in-sample predictions␣
 ↪(may overfit). Error: {e}")

        y_prob = model.predict_proba(X)[:, 1]

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

    # B. Probability histogram (bimodality inspection)

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

    plt.figure(figsize=(10, 5))
```

```
    sns.histplot(y_prob[y == 0], color="tab:blue", label="True Negative",
                 kde=True, stat="density", alpha=0.5)
    sns.histplot(y_prob[y == 1], color="tab:orange", label="True Positive",
                 kde=True, stat="density", alpha=0.5)
    plt.axvline(best_thresh, color="red", linestyle="--",
                label=f"Threshold = {best_thresh}")
    plt.title(f"{stage_name}: Prediction Probability Histogram\n(Ideal shape =␣
↪bimodal)")
    plt.xlabel("Predicted Probability")
    plt.legend()
    plt.show()
    hesitation_ratio = ((y_prob > 0.4) & (y_prob < 0.6)).mean()
    print(f"> Histogram summary: {hesitation_ratio:.1%} of samples fall in the␣
↪ambiguous range (0.4-0.6).")
    if hesitation_ratio > 0.2:
        print("  -> Warning: Model displays notable uncertainty around the␣
↪mid-probability range.")
    else:
        print("  -> Good: Model displays clear bimodality (confident␣
↪predictions).")
    # ------------------------------------------------------
    # C. Calibration curve analysis
    # ------------------------------------------------------
    print("\n[*] Comparing calibration: Raw vs Isotonic...")
    prob_true_raw, prob_pred_raw = calibration_curve(y, y_prob, n_bins=10)
```

```
    brier_raw = brier_score_loss(y, y_prob)

    iso_model = CalibratedClassifierCV(model, method="isotonic", cv=cv)

    try:

        y_prob_iso = cross_val_predict(

            iso_model, X, y, groups=groups, cv=cv,

            method="predict_proba", n_jobs=-1

        )[:, 1]

        prob_true_iso, prob_pred_iso = calibration_curve(y, y_prob_iso,␣
↪n_bins=10)

        brier_iso = brier_score_loss(y, y_prob_iso)

        plt.figure(figsize=(6, 6))

        plt.plot([0, 1], [0, 1], "k:", label="Perfect calibration")

        plt.plot(prob_pred_raw, prob_true_raw, "s-", label=f"Raw␣
↪(Brier={brier_raw:.4f})")

        plt.plot(prob_pred_iso, prob_true_iso, "o--", label=f"Isotonic␣
↪(Brier={brier_iso:.4f})")

        plt.title(f"{stage_name}: Calibration Comparison")

        plt.xlabel("Mean Predicted Probability")

        plt.ylabel("Observed Fraction of Positives")

        plt.legend()

        plt.show()

        print(f"> Brier Score - Raw: {brier_raw:.4f}, Isotonic: {brier_iso:.
↪4f}")

        if brier_iso >= brier_raw:

            print("  -> Isotonic calibration does NOT improve model reliability.
↪ Raw probabilities preferred.")
```

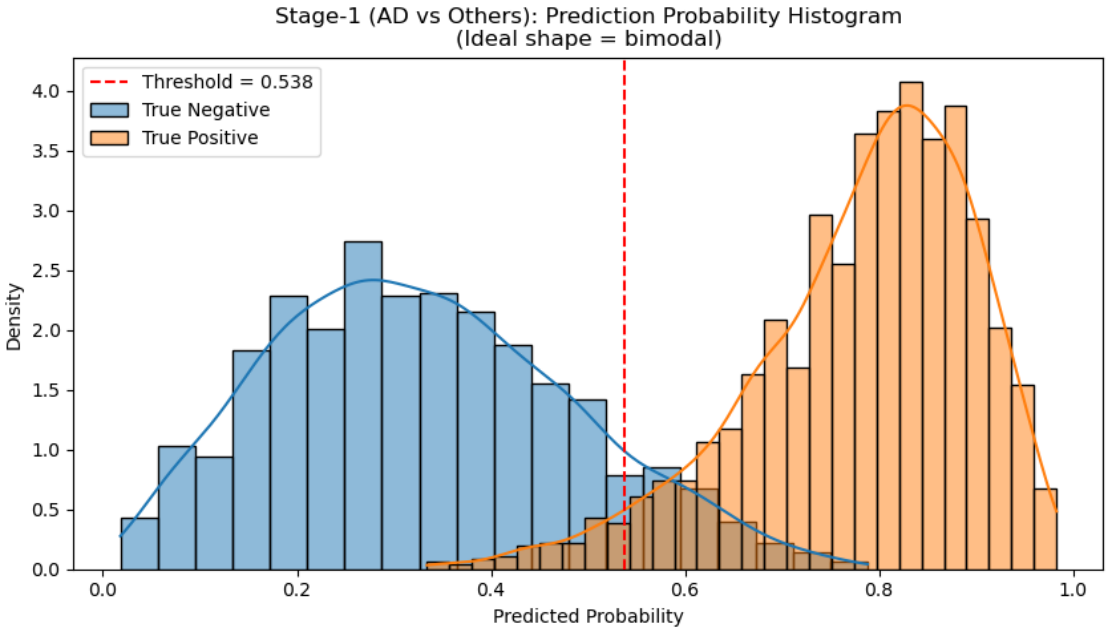
```
        else:
            print("  -> Isotonic calibration improves calibration.")
    except Exception as e:
        print(f"[!] Isotonic calibration skipped due to error: {e}")
    # ------------------------------------------------------
    # D. Misclassification analysis
    # ------------------------------------------------------
    print(f"\n[*] Misclassification analysis...")
    y_pred_class = (y_prob >= best_thresh).astype(int)
    mis_mask = (y != y_pred_class)
    correct_mask = ~mis_mask
    print(f"  - Total misclassified samples: {mis_mask.sum()}")
    # ---- D1. Missing data check ----
    try:
        raw_subset = df_raw.loc[X.index]   # index alignment
        missing_counts = raw_subset.isnull().sum(axis=1)
        avg_miss_wrong = missing_counts[mis_mask].mean()
        avg_miss_right = missing_counts[correct_mask].mean()
        print(f"  - Avg missing values (misclassified): {avg_miss_wrong:.2f}")
        print(f"  - Avg missing values (correct): {avg_miss_right:.2f}")
        plt.figure(figsize=(5, 4))
        sns.boxplot(
            x=[0]*correct_mask.sum() + [1]*mis_mask.sum(),
```

```
            y=pd.concat([missing_counts[correct_mask],␣
↪missing_counts[mis_mask]]),

            palette=["green", "red"]

        )

        plt.xticks([0, 1], ["Correct", "Misclassified"])

        plt.ylabel("Count of Missing Values (NaNs)")

        plt.title(f"{stage_name}: Missing Data Distribution")

        plt.show()

        if avg_miss_wrong > avg_miss_right * 1.1:

            print("  -> Misclassified cases tend to have substantially more␣
↪missing data.")

        else:

            print("  -> Missingness is similar between correct and incorrect␣
↪cases.")

    except Exception as e:

        print(f"[!] Missing-data analysis skipped (index alignment issue): {e}")

    # ---- D2. Threshold-distance analysis ----

    distance = np.abs(y_prob - best_thresh)

    close_call_mask = distance < 0.05

    wrong_and_close = (mis_mask & close_call_mask).sum()

    proportion_close = wrong_and_close / mis_mask.sum()

    print(f"  - Fraction of misclassifications near threshold (<0.05):␣
↪{proportion_close:.1%}")

    plt.figure(figsize=(6, 4))

    sns.kdeplot(distance[mis_mask], color="red", label="Misclassified",

                fill=True, alpha=0.3)
```

```
    sns.kdeplot(distance[correct_mask], color="green", label="Correct",
                fill=True, alpha=0.3)
    plt.xlabel(f"Distance to threshold {best_thresh}")
    plt.title(f"{stage_name}: Threshold-Proximity Error Analysis")
    plt.legend()
    plt.show()
    if proportion_close > 0.2:
        print("  -> A notable portion of errors occur near the decision␣
 ↪threshold (model uncertainty).")
    else:
        print("  -> Errors are widely distributed; not primarily due to the␣
 ↪threshold cut-off.")
# ============================================================================
# Execute diagnostics
# ============================================================================
# Stage-1
if "pipe1" in locals() and "X1" in locals():
    daniel_diagnostic_suite(
        model=pipe1,
        X=X1,
        y=y1,
        groups=groups,
        df_raw=df,
        stage_name="Stage-1 (AD vs Others)",
```

```
        best_thresh=0.538
    )
# Stage-2
if "pipe2" in locals() and "X2" in locals():
    daniel_diagnostic_suite(
        model=pipe2,
        X=X2,
        y=y2,
        groups=groups2,
        df_raw=df2,
        stage_name="Stage-2 (Mixed vs Non-AD)",
        best_thresh=0.5
    )
```

```
==================================================
Running diagnostics for: Stage-1 (AD vs Others) (Threshold = 0.538)
==================================================
[*] Computing cross-validated prediction probabilities…
[!] CV prediction failed. Falling back to in-sample predictions (may overfit).
Error: Found input variables with inconsistent numbers of samples: [3310, 3310,
5]
```




```
> Histogram summary: 14.7% of samples fall in the ambiguous range (0.4-0.6).
  -> Good: Model displays clear bimodality (confident predictions).

[*] Comparing calibration: Raw vs Isotonic…
[!] Isotonic calibration skipped due to error: Found input variables with
inconsistent numbers of samples: [3310, 3310, 5]

[*] Misclassification analysis…
  - Total misclassified samples: 210
[!] Missing-data analysis skipped (index alignment issue): '[3318, 3326, 3333,
3340, 3341, 3349, 3350, 3351, 3362, 3368, 3379, 3389, 3402, 3409, 3416, 3433,
3441, 3444, 3451, 3454, 3462, 3464, 3473, 3479, 3482, 3506, 3522, 3524, 3527,
3532, 3543, 3554, 3564, 3568, 3576, 3588, 3590, 3600, 3605, 3611, 3626, 3639,
3643, 3647, 3655, 3662, 3667, 3670, 3674, 3685, 3692, 3698, 3701, 3705, 3710,
3712, 3722, 3731, 3735, 3743, 3744, 3755, 3782, 3795, 3799, 3801, 3807, 3820,
3831, 3841, 3846, 3850, 3851, 3859, 3865, 3869, 3872, 3886, 3896, 3900, 3903,
3909, 3921, 3923, 3924, 3929, 3953, 3960, 3962, 3973, 3984, 3986, 3996, 3997,
4013, 4044, 4046, 4052, 4057, 4065, 4078, 4080, 4083, 4084, 4099, 4103, 4107,
4110, 4118, 4123, 4138, 4145, 4157, 4159, 4161, 4166, 4170, 4179, 4204, 4214,
4215, 4224, 4227, 4231, 4261, 4265, 4272, 4282, 4290, 4292, 4311, 4315, 4321,
4339, 4346, 4352, 4355, 4360, 4361, 4367, 4374, 4382, 4393, 4401, 4409, 4426,
4430, 4436, 4446, 4449, 4453, 4458, 4462, 4475, 4484, 4487, 4494, 4505, 4516,
4521, 4523, 4533, 4536, 4540, 4544, 4547, 4581, 4592, 4594, 4595, 4613, 4620,
4624, 4631, 4632, 4643, 4659, 4663, 4671, 4677, 4680, 4687, 4692, 4699, 4702,
4706, 4711, 4722, 4731, 4738, 4743, 4744, 4750, 4753, 4762, 4768, 4774, 4783,
4787, 4790, 4801, 4803, 4806, 4815, 4824, 4829, 4834, 4836, 4839, 4874, 4887,
```

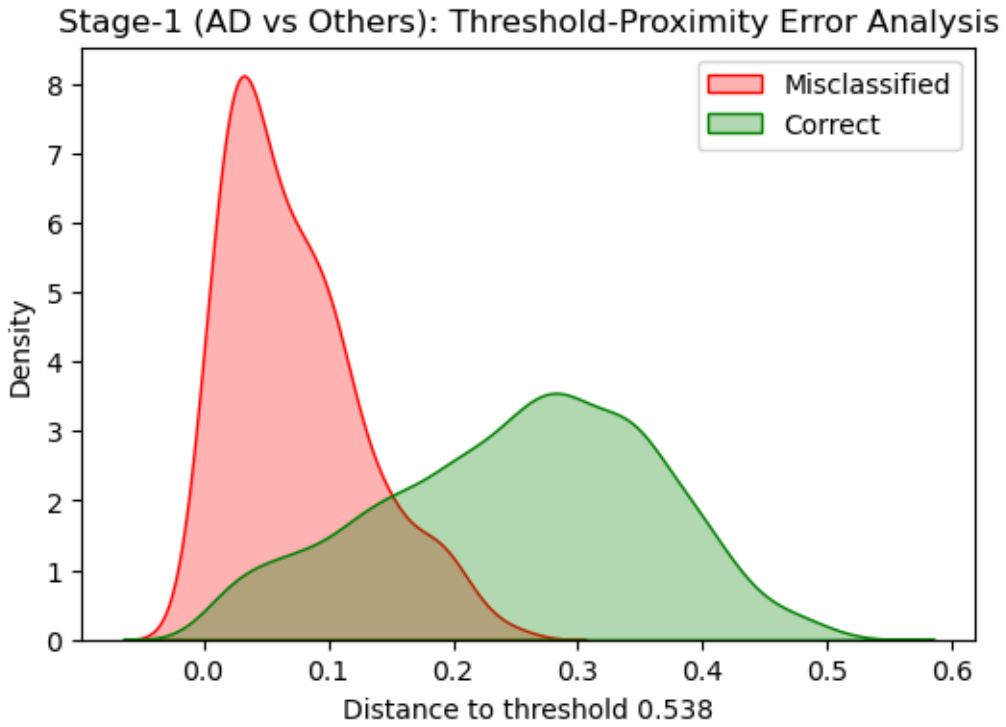
4894, 4899, 4906, 4914, 4915, 4919, 4927, 4933, 4939, 4946, 4956, 4958, 4959,
4966, 4973, 4981, 4990, 4996, 4999, 5004, 5005, 5007, 5016, 5020, 5030, 5036,
5039, 5043, 5047, 5053, 5059, 5064, 5068, 5072, 5075, 5086, 5088, 5094, 5096,
5101, 5107, 5123, 5131, 5134, 5142, 5148, 5153, 5155, 5163, 5173, 5187, 5191,
5195, 5197, 5202, 5205, 5209, 5216, 5224, 5226, 5234, 5244, 5246, 5247, 5250,
5255, 5256, 5263, 5264, 5267, 5273, 5281, 5287, 5292, 5298, 5305, 5312, 5329,
5330, 5334, 5340, 5350, 5356, 5360, 5365, 5369, 5379, 5386, 5395, 5403, 5406,
5409, 5422, 5429, 5441, 5457, 5460, 5463, 5474, 5483, 5495, 5496, 5500, 5520,
5526, 5534, 5545, 5560, 5570, 5580, 5581, 5593, 5595, 5597, 5602, 5605, 5613,
5634, 5668, 5673, 5676, 5684, 5690, 5694, 5709, 5717, 5726, 5735, 5739, 5744,
5745, 5750, 5757, 5760, 5775, 5782, 5786, 5794, 5809, 5821, 5828, 5833, 5836,
5842, 5847, 5848, 5850, 5900, 5905, 5910, 5912, 5920, 5923, 5932, 5936, 5942,
5948, 5984, 5986, 5987, 5999, 6001, 6004, 6008, 6028, 6034, 6043, 6044, 6046,
6051, 6056, 6062, 6082, 6088, 6096, 6103, 6109, 6114, 6115, 6120, 6126, 6133,
6135, 6139, 6155, 6159, 6160, 6164, 6167, 6174, 6176, 6183, 6184, 6185, 6191,
6199, 6202, 6227, 6244, 6256, 6260, 6261, 6265, 6268, 6269, 6272, 6277, 6285,
6295, 6299, 6302, 6305, 6313, 6314, 6320, 6328, 6329, 6333, 6359, 6364, 6365,
6369, 6371, 6376, 6379, 6386, 6393, 6402, 6408, 6413, 6416, 6422, 6430, 6441,
6443, 6447, 6453, 6460, 6462, 6475, 6479, 6488, 6493, 6501, 6514, 6519, 6524,
6526, 6531, 6532, 6535, 6540, 6544, 6549, 6553, 6558, 6560, 6564, 6567, 6573,
6580, 6581, 6585, 6588, 6594, 6596, 6598, 6609, 6610, 6611, 6613, 6617, 6627,
6634, 6636, 6639, 6647, 6651, 6656, 6660, 6665, 6666, 6670, 6672, 6675, 6678,
6684, 6697, 6730, 6735, 6738, 6746, 6751, 6757, 6758, 6763, 6764, 6768, 6770,
6775, 6779, 6787, 6789, 6793, 6797, 6801, 6809, 6811, 6826, 6838, 6844, 6856,
6869, 6871, 6878, 6882, 6886, 6893, 6897, 6900, 6904, 6910, 6911, 6915, 6918,
6919, 6923, 6951, 6954, 6960, 6965, 6971, 6981, 6988, 7004, 7011, 7013, 7019,
7021, 7024, 7026, 7029, 7030, 7031, 7033, 7039, 7040, 7042, 7052, 7055, 7058,
7067, 7077, 7087, 7093, 7094, 7099, 7106, 7113, 7119, 7126, 7128, 7138, 7145,
7149, 7150, 7159, 7163, 7169, 7196, 7200, 7206, 7208, 7216, 7221, 7243, 7246,
7251, 7254, 7260, 7262, 7272, 7276, 7286, 7289, 7293, 7294, 7297, 7316, 7320,
7321, 7333, 7335, 7345, 7349, 7352, 7364, 7372, 7374, 7378, 7381, 7386, 7387,
7389, 7393, 7399, 7400, 7405, 7409, 7413, 7424, 7432, 7439, 7447, 7450, 7457,
7463, 7465, 7467, 7471, 7490, 7494, 7500, 7504, 7505, 7509, 7523, 7528, 7535,
7538, 7543, 7553, 7562, 7566, 7568, 7569, 7570, 7571, 7577, 7619, 7624, 7628,
7639, 7644, 7645, 7671, 7673, 7677, 7678, 7681, 7682, 7687, 7688, 7692, 7704,
7707, 7711, 7719, 7724, 7728, 7731, 7797, 7798, 7799, 7804, 7810, 7812, 7818,
7840, 7841, 7848, 7850, 7856, 7864, 7874, 7876, 7880, 7894, 7896, 7904, 7912,
7924, 7955, 7957, 7965, 7983, 7996, 8002, 8004, 8007, 8011, 8013, 8017, 8023,
8026, 8030, 8032, 8033, 8040, 8042, 8044, 8049, 8053, 8059, 8064, 8068, 8086,
8088, 8097, 8102, 8120, 8124, 8126, 8128, 8149, 8152, 8158, 8159, 8161, 8164,
8171, 8178, 8183, 8186, 8190, 8196, 8213, 8214, 8220, 8230, 8237, 8238, 8241,
8242, 8244, 8247, 8249, 8252, 8258, 8263, 8264, 8273, 8281, 8286, 8295, 8298,
8303, 8305, 8307, 8309, 8321, 8325, 8329, 8332, 8338, 8339, 8340, 8341, 8377,
8382, 8387, 8392, 8401, 8407, 8411, 8415, 8417, 8422, 8430, 8434, 8436, 8438,
8439, 8442, 8449, 8452, 8462, 8464, 8471, 8479, 8489, 8492, 8493, 8499, 8504,
8506, 8509, 8518, 8519, 8520, 8522, 8526, 8538, 8543, 8553, 8556, 8560, 8567,
8570, 8571, 8576, 8578, 8582, 8585, 8595, 8596, 8601, 8604, 8608, 8611, 8613,
8615, 8616, 8626, 8627, 8629, 8634, 8636, 8637, 8640, 8643, 8652, 8655, 8657,

8663, 8669, 8679, 8687, 8688, 8693, 8705, 8709, 8712, 8716, 8723, 8732, 8734,
8741, 8745, 8750, 8763, 8767, 8773, 8776, 8777, 8780, 8783, 8784, 8786, 8790,
8798, 8803, 8804, 8805, 8809, 8814, 8817, 8818, 8820, 8822, 8825, 8826, 8827,
8828, 8832, 8841, 8843, 8845, 8851, 8863, 8868, 8888, 8890, 8892, 8894, 8897,
8901, 8904, 8906, 8907, 8910, 8912, 8913, 8915, 8917, 8921, 8929, 8946, 8955,
8962, 8965, 8968, 8972, 8974, 8976, 8978, 8985, 8988, 8990, 8996, 8997, 8999,
9000, 9001, 9004, 9010, 9012, 9017, 9022, 9024, 9027, 9028, 9035, 9036, 9040,
9042, 9046, 9048, 9051, 9052, 9054, 9061, 9063, 9067, 9071, 9075, 9078, 9079,
9084, 9093, 9099, 9102, 9111, 9113, 9117, 9121, 9124, 9126, 9129, 9135, 9138,
9139, 9141, 9142, 9149, 9161, 9164, 9165, 9168, 9180, 9182, 9183, 9187, 9188,
9200, 9206, 9208, 9209, 9212, 9215, 9227, 9233, 9234, 9235, 9240, 9247, 9248,
9251, 9258, 9260, 9263, 9268, 9291, 9293, 9296, 9299, 9301, 9309, 9310, 9313,
9327, 9330, 9332, 9337, 9343, 9347, 9352, 9357, 9360, 9366, 9376, 9380, 9382,
9386, 9387, 9389, 9396, 9402, 9405, 9407, 9409, 9410, 9413, 9417, 9422, 9424,
9427, 9429, 9430, 9434, 9438, 9444, 9449, 9457, 9460, 9461, 9466, 9467, 9471,
9475, 9478, 9479, 9480, 9483, 9485, 9489, 9490, 9493, 9539, 9577, 9579, 9580,
9581, 9583, 9585, 9588, 9590, 9604, 9607, 9610, 9614, 9618, 9620, 9624, 9627,
9632, 9634, 9636, 9642, 9644, 9645, 9651, 9654, 9657, 9659, 9660, 9665, 9683,
9689, 9691, 9692, 9693, 9706, 9709, 9713, 9715, 9720, 9723, 9725, 9727, 9728,
9734, 9736, 9737, 9738, 9739, 9749, 9750, 9753, 9758, 9766, 9770, 9781, 9826,
9830, 9834, 9837, 9853, 9854, 9858, 9865, 9868, 9873, 9877, 9885, 9886, 9888,
9890, 9893, 9896, 9898, 9904, 9905, 9912, 9916, 9918, 9924, 9925, 9926, 9931,
9934, 9942, 9946, 9954, 9961, 9969, 9993, 9999, 10014, 10015, 10021, 10022,
10046, 10063, 10066, 10068, 10075, 10080, 10081, 10084, 10087, 10091, 10093,
10094, 10096, 10098, 10099, 10101, 10107, 10118, 10120, 10127, 10129, 10132,
10134, 10139, 10145, 10149, 10151, 10154, 10155, 10156, 10159, 10181, 10183,
10191, 10196, 10197, 10201, 10204, 10208, 10212, 10214, 10217, 10221, 10224,
10226, 10227, 10229, 10232, 10236, 10237, 10238, 10247, 10251, 10256, 10259,
10260, 10265, 10273, 10276, 10277, 10279, 10285, 10286, 10292, 10295, 10308,
10312, 10313, 10320, 10329, 10330, 10336, 10342, 10344, 10348, 10367, 10371,
10374, 10375, 10378, 10383, 10387, 10415, 10416, 10418, 10420, 10421, 10451,
10455, 10457, 10463, 10464, 10470, 10482, 10488, 10492, 10496, 10498, 10499,
10500, 10505, 10506, 10508, 10509, 10511, 10513, 10517, 10522, 10523, 10528,
10530, 10531, 10535, 10548, 10552, 10553, 10557, 10558, 10570, 10571, 10573,
10574, 10577, 10580, 10582, 10586, 10595, 10599, 10601, 10603, 10606, 10607,
10609, 10614, 10617, 10619, 10620, 10621, 10622, 10623, 10624, 10627, 10642,
10643, 10645, 10648, 10652, 10653, 10657, 10670, 10687, 10694, 10696, 10698,
10702, 10705, 10712, 10718, 10720, 10734, 10736, 10741, 10745, 10748, 10757,
10763, 10768, 10771, 10774, 10777, 10779, 10783, 10789, 10798, 10802, 10806,
10811, 10813, 10817, 10829, 10831, 10833, 10837, 10851, 10857, 10861, 10864,
10869, 10873, 10876, 10880, 10881, 10882, 10884, 10889, 10896, 10898, 10900,
10901, 10904, 10908, 10912, 10914, 10917, 10971, 10974, 10984, 10986, 10987,
10990, 10997, 10998, 10999, 11000, 11002, 11004, 11007, 11011, 11012, 11016,
11019, 11025, 11031, 11034, 11036, 11038, 11044, 11058, 11061, 11065, 11066,
11068, 11071, 11074, 11076, 11078, 11079, 11084, 11087, 11088, 11091, 11093,
11101, 11105, 11107, 11108, 11111, 11113, 11115, 11124, 11129, 11132, 11135,
11136, 11139, 11143, 11146, 11160, 11197, 11200, 11203, 11208, 11214, 11220,
11223, 11225, 11226, 11229, 11252, 11253, 11265, 11268, 11271, 11274, 11276,

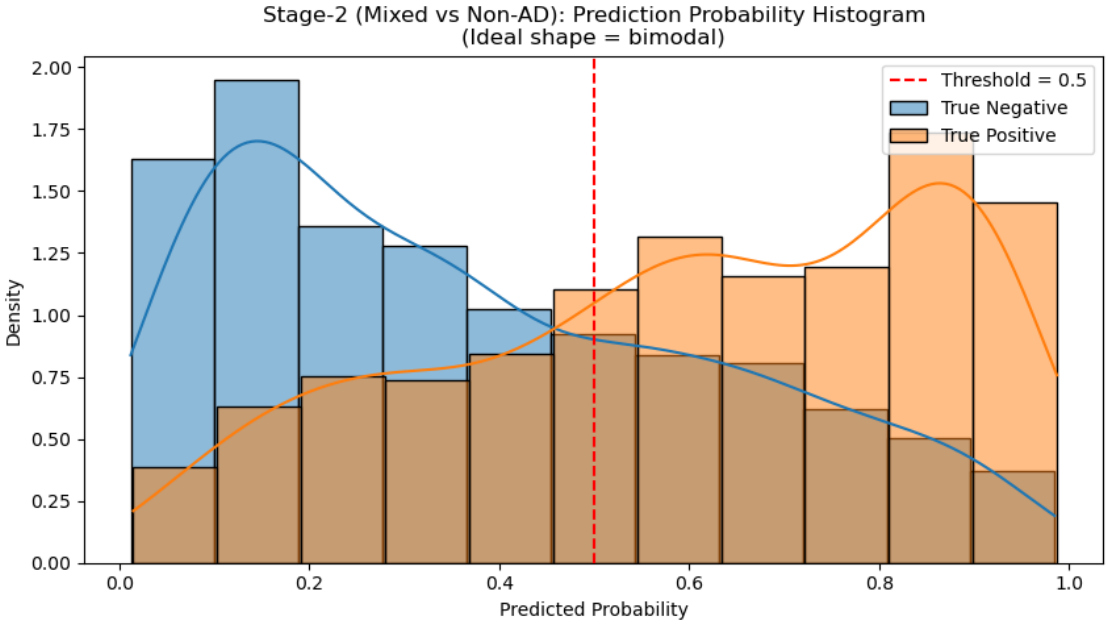
11289, 11293, 11295, 11296, 11300, 11305, 11309, 11311, 11312, 11326, 11353,
11354, 11358, 11360, 11366, 11368, 11370, 11374, 11377, 11383, 11386, 11389,
11391, 11395, 11400, 11403, 11407, 11410, 11414, 11435, 11446, 11450, 11456,
11457, 11458, 11462, 11465, 11466, 11472, 11474, 11480, 11486, 11490, 11502,
11509, 11511, 11516, 11520, 11524, 11555, 11564, 11565, 11568, 11576, 11578,
11597, 11599, 11601, 11602, 11649, 11658, 11663, 11667, 11670, 11675, 11680,
11684, 11687, 11694, 11697, 11703, 11705, 11723, 11726, 11727, 11733, 11735,
11736, 11737, 11738, 11744, 11746, 11780, 11796, 11799, 11804, 11808, 11813,
11827, 11832, 11836, 11838, 11841, 11847, 11854, 11886, 11888, 11892, 11895,
11897, 11898, 11905, 11907, 11909, 11911, 11913, 11919, 11923, 11928, 11929,
11933, 11938, 11946, 11949, 11952, 11956, 11957, 11959, 11961, 11970, 11976,
11978, 11980, 11981, 11983, 11986, 11988, 11991, 11994, 12004, 12013, 12015,
12017, 12018, 12021, 12029, 12037, 12041, 12052, 12054, 12059, 12063, 12066,
12067, 12068, 12072, 12077, 12089, 12098, 12104, 12107, 12109, 12114, 12132,
12146, 12148, 12150, 12154, 12159, 12174, 12175, 12179, 12182, 12184, 12185,
12197, 12202, 12207, 12209, 12213, 12219, 12221, 12223, 12231, 12233, 12235,
12238, 12242, 12247, 12249, 12251, 12252, 12254, 12255, 12257, 12259, 12261,
12262, 12265, 12267, 12270, 12273, 12286, 12288, 12292, 12297, 12301, 12302,
12304, 12312, 12317, 12318, 12322, 12324, 12327, 12331, 12335, 12336, 12344,
12347, 12350, 12359, 12362, 12365, 12369, 12371, 12379, 12381, 12383, 12388,
12399, 12405, 12407, 12409, 12410, 12414, 12415, 12416, 12418, 12419, 12421,
12424, 12428, 12432, 12435, 12439, 12449, 12453, 12456, 12462, 12474, 12479,
12483, 12485, 12491, 12492, 12495, 12496, 12500, 12503, 12509, 12511, 12512,
12515, 12518, 12520, 12524, 12530, 12533, 12537, 12539, 12541, 12547, 12551,
12552, 12559, 12567, 12571, 12581, 12585, 12597, 12602, 12603, 12606, 12611,
12617, 12620, 12623, 12625, 12626, 12629, 12635, 12638, 12640, 12641, 12644,
12647, 12649, 12651, 12655, 12668, 12670, 12672, 12673, 12674, 12678, 12686,
12704, 12705, 12708, 12711, 12721, 12724, 12725, 12729, 12735, 12739, 12742,
12747, 12749, 12752, 12754, 12760, 12766, 12771, 12775, 12779, 12780, 12782,
12786, 12789, 12793, 12796, 12799, 12804, 12808, 12810, 12813, 12814, 12817,
12822, 12825, 12829, 12833, 12835, 12837, 12838, 12841, 12850, 12862, 12865,
12868, 12881, 12885, 12897, 12907, 12908, 12911, 12914, 12917, 12928, 12930,
12933, 12940, 12950, 12952, 12958, 12959, 12962, 12968, 12971, 12978, 12982,
12988, 12994, 12998, 13000, 13003, 13007, 13010, 13011, 13014, 13051, 13057,
13065, 13068, 13071, 13072, 13074, 13075, 13078, 13084, 13094, 13104, 13107,
13111, 13118, 13120, 13123, 13124, 13126, 13128, 13131, 13134, 13138, 13142,
13146, 13147, 13151, 13166, 13169, 13172, 13176, 13193, 13201, 13202, 13203,
13205, 13208, 13215, 13217, 13221, 13222, 13226, 13230, 13232, 13233, 13237,
13253, 13257, 13263, 13265, 13269, 13273, 13275, 13277, 13285, 13286, 13289,
13299, 13304, 13306, 13312, 13316, 13320, 13326, 13337, 13346, 13353, 13354,
13355, 13357, 13359, 13361, 13362, 13369, 13371, 13376, 13379, 13393, 13398,
13401, 13402, 13403, 13406, 13410, 13412, 13418, 13419, 13423, 13424, 13427,
13431, 13433, 13434, 13441, 13444, 13447, 13452, 13454, 13455, 13466, 13471,
13474, 13481, 13489, 13493, 13496, 13500, 13505, 13511, 13518, 13520, 13521,
13522, 13525, 13528, 13531, 13538, 13539, 13544, 13552, 13556, 13561, 13563,
13566, 13569, 13572, 13575, 13576, 13580, 13595, 13607, 13613, 13615, 13618,
13620, 13622, 13623, 13629, 13639, 13641, 13645, 13649, 13654, 13659, 13737,
13739, 13741, 13743, 13744, 13747, 13750, 13751, 13753, 13756, 13757, 13762,

13767, 13773, 13775, 13778, 13781, 13786, 13791, 13802, 13808, 13810, 13813,
13816, 13820, 13824, 13825, 13827, 13830, 13832, 13836, 13839, 13842, 13843,
13849, 13854, 13855, 13857, 13860, 13862, 13865, 13869, 13871, 13875, 13882,
13890, 13891, 13894, 13895, 13898, 13901, 13908, 13917, 13924, 13928, 13931,
13932, 13935, 13942, 13946, 13948, 13952, 13959, 13961, 13964, 13965, 13971,
13974, 13980, 13984, 13998, 14001, 14005, 14008, 14011, 14016, 14017, 14024,
14026, 14034, 14048, 14049, 14059, 14063, 14067, 14068, 14069, 14071, 14076,
14079, 14082, 14085, 14088, 14091, 14098, 14099, 14104, 14106, 14107, 14111,
14112, 14118, 14121, 14123, 14127, 14130, 14133, 14135, 14159, 14161, 14168,
14171, 14173, 14176, 14177, 14180, 14182, 14185, 14186, 14187, 14195, 14196,
14197, 14201, 14206, 14207, 14209, 14212, 14213, 14216, 14218, 14222, 14224,
14227, 14235, 14240, 14243, 14249, 14253, 14255, 14257, 14262, 14264, 14267,
14276, 14282, 14286, 14294, 14310, 14312, 14314, 14316, 14318, 14320, 14325,
14329, 14330, 14333, 14334, 14335, 14337, 14340, 14341, 14344, 14347, 14349,
14353, 14357, 14361, 14362, 14369, 14372, 14375, 14378, 14383, 14388, 14389,
14392, 14395, 14397, 14399, 14402, 14403, 14407, 14410, 14413, 14417, 14419,
14421, 14422, 14426, 14434, 14437, 14438, 14440, 14442, 14446, 14449, 14454,
14460, 14463, 14468, 14470, 14477, 14480, 14483, 14484, 14485, 14487, 14494,
14498, 14501, 14504, 14508, 14517, 14519, 14521, 14523, 14524, 14526, 14527,
14528, 14530, 14532, 14540, 14542, 14545, 14553, 14569, 14574, 14577, 14582,
14583, 14584, 14587, 14601, 14606, 14609, 14610, 14612, 14613, 14615, 14616,
14619, 14621, 14624, 14627, 14641, 14642, 14644, 14645, 14651, 14653, 14691,
14698, 14699, 14701, 14703, 14705, 14709, 14714, 14721, 14736, 14738, 14742,
14747, 14748, 14751, 14781, 14783, 14784, 14789, 14791, 14796, 14800, 14803,
14835, 14842, 14843, 14846, 14854, 14855, 14857, 14861, 14864, 14866, 14872,
14875, 14879, 14890, 14892, 14894, 14900, 14908, 14910, 14914, 14932, 14937,
14939, 14946, 14947, 14948, 14952, 14953, 14961, 14965, 14967, 14975, 14989,
14993, 14997, 15000, 15002, 15004, 15006, 15008, 15011, 15013, 15016, 15019,
15021, 15026, 15029, 15033, 15037, 15041, 15043, 15050, 15053, 15055, 15061,
15065, 15068, 15071, 15078, 15099, 15104, 15107, 15110, 15114, 15116, 15120,
15121, 15124, 15126, 15130, 15134, 15135, 15138, 15142, 15144, 15147, 15155,
15158, 15161, 15167, 15168, 15170, 15171, 15177, 15179, 15181, 15182, 15184,
15185, 15187, 15189, 15197, 15199, 15200, 15204, 15206, 15210, 15211, 15213,
15215, 15217, 15219, 15221, 15226, 15229, 15231, 15247, 15251, 15256, 15258,
15264, 15265, 15274, 15276, 15279, 15280, 15282, 15290, 15297, 15300, 15301,
15304, 15316, 15319, 15320, 15323, 15326, 15329, 15331, 15333, 15335, 15338,
15340, 15342, 15344, 15352, 15355, 15358, 15360, 15361, 15363, 15371, 15379,
15382, 15385, 15389, 15390, 15394, 15398, 15400, 15402, 15407, 15414, 15415,
15417, 15418, 15420, 15423, 15426, 15431, 15434, 15435, 15436, 15437, 15443,
15445, 15447, 15450, 15452, 15456, 15461, 15465, 15471, 15475, 15478, 15486,
15488, 15490, 15496, 15500, 15503, 15504, 15508, 15509, 15512, 15517, 15520,
15525, 15526, 15529, 15535, 15537, 15541, 15545, 15550, 15558, 15567, 15568,
15569, 15571, 15572, 15579, 15587, 15588, 15589, 15593, 15597, 15602, 15613,
15618, 15626, 15632, 15638, 15641, 15648, 15649, 15653, 15657, 15658, 15662,
15663, 15666, 15667, 15668, 15682, 15685, 15688, 15694, 15696, 15699, 15717,
15727, 15728, 15729, 15734, 15735, 15743, 15745, 15759, 15762, 15765, 15770,
15773, 15776, 15780, 15783, 15791, 15793, 15800, 15801, 15802, 15812, 15842,
15857, 15860, 15863, 15866, 15870, 15877, 15878, 15881, 15882, 15893, 15894,

```
15895, 15899, 15903, 15907, 15909, 15911, 15914, 15915, 15923, 15926, 15928,
15937, 15939, 15941, 15943, 15946, 15954, 15966, 15971, 15983, 15985, 15988,
15989, 15990, 15993, 15996, 16000, 16008, 16010, 16011, 16020, 16021, 16029,
16031, 16033, 16036, 16038, 16040, 16045, 16046, 16049, 16054, 16056, 16066,
16069, 16072, 16077, 16078, 16082, 16114, 16117, 16131, 16133, 16135, 16136,
16138, 16145, 16147, 16149, 16151, 16152, 16154, 16155, 16158, 16167, 16170,
16184, 16186, 16187, 16189, 16190, 16191, 16196, 16198, 16200, 16210, 16224,
16225, 16228, 16233, 16236, 16237, 16240, 16241, 16246, 16258, 16262, 16264,
16267, 16269, 16272, 16274, 16277, 16279, 16280, 16290, 16291, 16295, 16298,
16303, 16313, 16317, 16325, 16326, 16327, 16329, 16331, 16336, 16340, 16343,
16348, 16349, 16351, 16352, 16355, 16356, 16359, 16371, 16385, 16386, 16388,
16391, 16394, 16396, 16397, 16400, 16402, 16404, 16410, 16434, 16436, 16437,
16438, 16446, 16447, 16449, 16451, 16453, 16457, 16460, 16461, 16464, 16467,
16469, 16474, 16479, 16481, 16484, 16486, 16488, 16490, 16495, 16498, 16504,
16518, 16520, 16522, 16526, 16527, 16532, 16534, 16547, 16550, 16552, 16553,
16555, 16564, 16565, 16581, 16587, 16598, 16600, 16602, 16608, 16611, 16612,
16615, 16617, 16623, 16628, 16630, 16632, 16634, 16637, 16645, 16647, 16648,
16652, 16654, 16655, 16658, 16662, 16663, 16665, 16667, 16668, 16670, 16671,
16680, 16683, 16684, 16685, 16686, 16687, 16688, 16694, 16697, 16698, 16704,
16709, 16712, 16714, 16715, 16717, 16719, 16723, 16734, 16735, 16736, 16739,
16743, 16754, 16755, 16756, 16757, 16759, 16760, 16766, 16774, 16776, 16779,
16780, 16781, 16784, 16788, 16791, 16803, 16808, 16809, 16812, 16814, 16815,
16819, 16822, 16823, 16831, 16833, 16834, 16835, 16839, 16842, 16844, 16845,
16848, 16849, 16850, 16851, 16852] not in index'
  - Fraction of misclassifications near threshold (<0.05): 44.3%
```

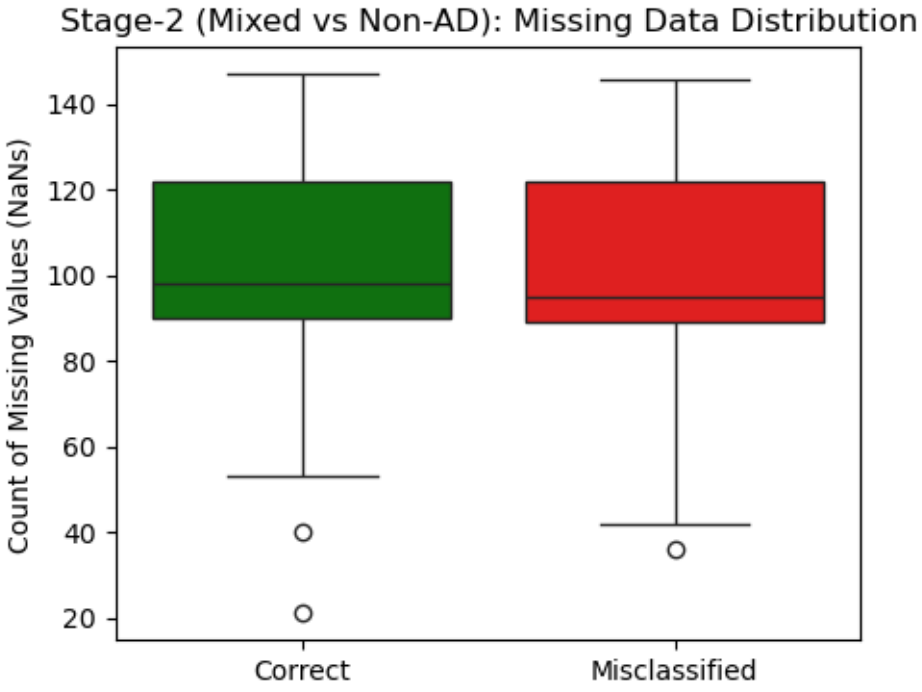


```
  -> A notable portion of errors occur near the decision threshold (model
uncertainty).

==================================================
Running diagnostics for: Stage-2 (Mixed vs Non-AD) (Threshold = 0.5)
==================================================
[*] Computing cross-validated prediction probabilities…
```

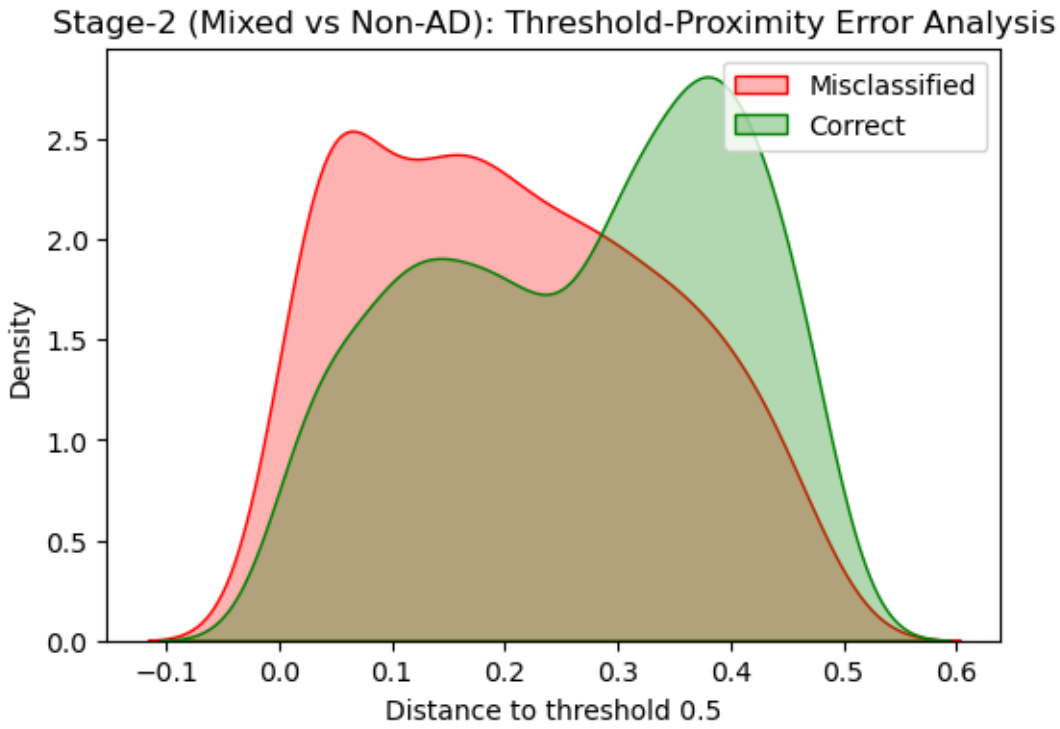



```
> Histogram summary: 19.0% of samples fall in the ambiguous range (0.4-0.6).
  -> Good: Model displays clear bimodality (confident predictions).

[*] Comparing calibration: Raw vs Isotonic…
[!] Isotonic calibration skipped due to error: Found array with 0 sample(s)
(shape=(0, 13)) while a minimum of 1 is required by SimpleImputer.

[*] Misclassification analysis…
  - Total misclassified samples: 430
  - Avg missing values (misclassified): 99.59
  - Avg missing values (correct): 100.66

/tmp/ipykernel_1118297/625832797.py:195: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```

```
sns.boxplot(
```




```
-> Missingness is similar between correct and incorrect cases.
- Fraction of misclassifications near threshold (<0.05): 15.1%
```



-> Errors are widely distributed; not primarily due to the threshold cut-off.

[42]:
```
import pandas as pd

from sklearn.metrics import confusion_matrix

# ========================================

#       (Final Confusion Matrix)

# ========================================

print("\n" + "="*50)

print("   (Final Hierarchical Confusion Matrix)")

print("="*50)

# 1.

labels3 = ["AD", "Mixed", "Non-AD"]

# 2.     (      )
```

```python
if 'y_true3' in locals():
    y_target = y_true3
elif 'y_true' in locals():
    y_target = y_true
else:
    #       df
    target_col = "target_diag" if "target_diag" in df.columns else "diag3"
    y_target = df[target_col].values
# 3.
# pred_final
cm_final_array = confusion_matrix(y_target, pred_final, labels=labels3)
# 4.     DataFrame
cm_final_df = pd.DataFrame(
    cm_final_array,
    index=[f"True {label}" for label in labels3],   #
    columns=[f"Pred {label}" for label in labels3]  #
)
# 5.
print(cm_final_df)
print("-" * 50)
#
acc = (y_target == pred_final).mean()
print(f"Overall Accuracy: {acc:.4f} ({cm_final_array.trace()}/{len(y_target)})")
```

```
print("="*50)
```

```
==================================================
   (Final Hierarchical Confusion Matrix)
==================================================
             Pred AD  Pred Mixed  Pred Non-AD
True AD         1917          33           42
True Mixed       101         438          106
True Non-AD       34         130          509
--------------------------------------------------
Overall Accuracy: 0.8653 (2864/3310)
==================================================
```

[46]:
```
import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.metrics import (
    classification_report,
    balanced_accuracy_score,
    roc_auc_score,
    accuracy_score,
)

# ----------      ----------
FAIR_DIR = Path("surv_out") / "fairness"
FAIR_DIR.mkdir(parents=True, exist_ok=True)

# ----------     ----------
required_vars = ["df", "y_true3", "pred_final"]
for v in required_vars:
    if v not in globals():
        raise RuntimeError(f"   {v}           ")

if "Gender_ID_num" not in df.columns or "ethnicitycleaned_code" not in df.
 ↪columns:
    raise RuntimeError("df    Gender_ID_num   ethnicitycleaned_code    ")

# ----------      ----------
labels_3    = ["AD", "Mixed", "Non-AD"]   #
y_true_all  = np.array(y_target)
y_pred_all  = np.array(pred_final)

gender_codes = df["Gender_ID_num"].to_numpy()
eth_codes    = df["ethnicitycleaned_code"].to_numpy()
```

```
#
gender_label_map = {
    -1: "Missing_gender",
     0: "Gender_0",  #     "Female"
     1: "Gender_1",  #     "Male"
}
#
eth_label_map = {
    -1: "Missing_eth",
}

# ================================================================
#    1
# ================================================================
def fairness_multiclass_by_group(y_true, y_pred, group_codes,
                                 group_label_map, labels_3, group_name,
                                 skip_code=-1, min_n=30,
                                 save_prefix=""):
    """
       group  /
        - accuracy
        - macro recall (balanced accuracy for multiclass)
        - macro F1
           (AD/Mixed/Non-AD)   precision/recall/F1
    """
    summary_rows = []
    perclass_rows = []

    unique_codes = np.unique(group_codes)
    for g in unique_codes:
        if g == skip_code:
            #
            continue

        mask = (group_codes == g)
        n = int(mask.sum())
        if n < min_n:
            print(f"[{group_name}]   {g} {group_label_map.get(g, g)}    {n} <␣
 ↪{min_n}  /   ")
            if n == 0:
                continue

        yt = y_true[mask]
        yp = y_pred[mask]

        rep = classification_report(
            yt, yp, labels=labels_3,
```

```
            output_dict=True, zero_division=0
        )

        #    accuracy / macro F1 / macro recall
        acc       = accuracy_score(yt, yp)
        macro_f1  = np.mean([rep[c]["f1-score"] for c in labels_3])
        macro_rec = np.mean([rep[c]["recall"]    for c in labels_3])

        summary_rows.append({
            f"{group_name}_code": int(g),
            f"{group_name}_label": group_label_map.get(g, str(g)),
            "n": n,
            "accuracy": acc,
            "macro_recall": macro_rec,
            "macro_f1": macro_f1,
        })

        for c in labels_3:
            perclass_rows.append({
                f"{group_name}_code": int(g),
                f"{group_name}_label": group_label_map.get(g, str(g)),
                "class": c,
                "precision": rep[c]["precision"],
                "recall": rep[c]["recall"],
                "f1": rep[c]["f1-score"],
                "support": int(rep[c]["support"]),
            })

    summary_df  = pd.DataFrame(summary_rows).sort_values(f"{group_name}_code")
    perclass_df = pd.DataFrame(perclass_rows).
 ↪sort_values([f"{group_name}_code", "class"])

    if save_prefix:
        summary_path  = FAIR_DIR / f"{save_prefix}_{group_name}_summary.csv"
        perclass_path = FAIR_DIR / f"{save_prefix}_{group_name}_perclass.csv"
        summary_df.to_csv(summary_path, index=False)
        perclass_df.to_csv(perclass_path, index=False)
        print(f"[{group_name}]        ")
        print("  -", summary_path)
        print("  -", perclass_path)

    return summary_df, perclass_df


# ================================================================
#    2       Stage-1 / Stage-2
# ================================================================
```

```
from sklearn.metrics import (
    accuracy_score,
    balanced_accuracy_score,
    roc_auc_score,
    confusion_matrix,
)
def fairness_binary_by_group(y_true, y_prob, group_codes, group_label_map,
                             group_name, task_name, thr=0.5,
                             skip_code=-1, min_n=30,
                             save_prefix=""):
    """
       group
        - accuracy
        -   precision / recall / F1
        - balanced accuracy
        - ROC AUC         NaN
    """
    rows = []
    unique_codes = np.unique(group_codes)
    for g in unique_codes:
        if g == skip_code:
            continue
        mask = (group_codes == g)
        n = int(mask.sum())
```

```
        if n < min_n:

            print(f"[{task_name}-{group_name}]   {g} {group_label_map.get(g,␣
↪g)}   {n} < {min_n}  /   ")

            if n == 0:

                continue

        yt = y_true[mask]

        prob = y_prob[mask]

        yp = (prob >= thr).astype(int)

        # ----    labels=[0,1]    2x2 ----

        cm = confusion_matrix(yt, yp, labels=[0, 1])

        #       cm    2x2    0

        tn, fp, fn, tp = cm.ravel()

        # accuracy

        acc = (tp + tn) / max(tp + tn + fp + fn, 1)

        #   precision / recall / F1

        if tp + fp > 0:

            precision_pos = tp / (tp + fp)

        else:

            precision_pos = np.nan

        if tp + fn > 0:

            recall_pos = tp / (tp + fn)

        else:

            recall_pos = np.nan
```

```
        if precision_pos is not np.nan and recall_pos is not np.nan and␣
↪(precision_pos + recall_pos) > 0:

            f1_pos = 2 * precision_pos * recall_pos / (precision_pos +␣
↪recall_pos)

        else:

            f1_pos = np.nan

        # balanced accuracy

        bal_acc = balanced_accuracy_score(yt, yp)

        # AUC              NaN

        try:

            auc = roc_auc_score(yt, prob)

        except Exception:

            auc = np.nan

        rows.append({

            f"{group_name}_code": int(g),

            f"{group_name}_label": group_label_map.get(g, str(g)),

            "n": n,

            "accuracy": acc,

            "precision_pos": precision_pos,

            "recall_pos": recall_pos,

            "f1_pos": f1_pos,

            "balanced_acc": bal_acc,

            "roc_auc": auc,

        })

    df_out = pd.DataFrame(rows).sort_values(f"{group_name}_code")
```

```
    if save_prefix:

        path = FAIR_DIR / f"{save_prefix}_{task_name}_{group_name}.csv"

        df_out.to_csv(path, index=False)

        print(f"[{task_name}-{group_name}]        {path}")

    return df_out



# ================================================================
# 1.        /
# ================================================================
print("\n==========         ==========")
gender_mask_valid = (gender_codes != -1)      #          np.
 ↪ones_like(gender_codes, dtype=bool)
gender_summary, gender_perclass = fairness_multiclass_by_group(
    y_true=y_true_all[gender_mask_valid],
    y_pred=y_pred_all[gender_mask_valid],
    group_codes=gender_codes[gender_mask_valid],
    group_label_map=gender_label_map,
    labels_3=labels_3,
    group_name="gender",
    skip_code=-1,
    min_n=30,
    save_prefix="final3"
)
print(gender_summary)

print("\n==========         ==========")
eth_mask_valid = (eth_codes != -1)
eth_summary, eth_perclass = fairness_multiclass_by_group(
    y_true=y_true_all[eth_mask_valid],
    y_pred=y_pred_all[eth_mask_valid],
    group_codes=eth_codes[eth_mask_valid],
    group_label_map=eth_label_map,
    labels_3=labels_3,
    group_name="ethnicity",
    skip_code=-1,
    min_n=30,
    save_prefix="final3"
)
print(eth_summary)
```

```
==========         ==========
[gender]
  - surv_out/fairness/final3_gender_summary.csv
  - surv_out/fairness/final3_gender_perclass.csv
   gender_code gender_label     n  accuracy  macro_recall  macro_f1
0            0     Gender_0  1337  0.870606      0.817649  0.820293
1            1     Gender_1  1973  0.861632      0.785270  0.794176

==========         ==========
[ethnicity]   7 7   10 < 30  /
[ethnicity]   10 10   23 < 30  /
[ethnicity]   12 12   14 < 30  /
[ethnicity]   13 13   22 < 30  /
[ethnicity]   14 14   15 < 30  /
[ethnicity]   15 15   8 < 30  /
[ethnicity]   16 16   4 < 30  /
[ethnicity]   17 17   2 < 30  /
[ethnicity]   18 18   1 < 30  /
[ethnicity]
  - surv_out/fairness/final3_ethnicity_summary.csv
  - surv_out/fairness/final3_ethnicity_perclass.csv
    ethnicity_code ethnicity_label     n  accuracy  macro_recall  macro_f1
0                0               0  1645  0.866869      0.791047  0.802742
1                1               1   145  0.820690      0.793022  0.777864
2                2               2   128  0.859375      0.813636  0.810426
3                3               3    64  0.875000      0.802309  0.809340
4                4               4   220  0.886364      0.832530  0.838049
5                5               5   489  0.881391      0.827956  0.827925
6                6               6   133  0.857143      0.791854  0.793298
7                7               7    10  0.800000      0.500000  0.555556
8                8               8    97  0.855670      0.778174  0.772791
9                9               9   128  0.859375      0.765157  0.768780
10              10              10    23  0.913043      0.555556  0.600000
11              11              11    46  0.826087      0.777357  0.769915
12              12              12    14  0.785714      0.516667  0.582011
13              13              13    22  0.818182      0.735690  0.715344
14              14              14    15  0.733333      0.694444  0.674074
15              15              15     8  0.750000      0.444444  0.525253
16              16              16     4  1.000000      0.333333  0.333333
17              17              17     2  0.500000      0.166667  0.222222
18              18              18     1  1.000000      0.333333  0.333333
```

[ ]: