In [1]:
```
# ==== [0] Setup & Load ====
import os, warnings, json
import numpy as np
import pandas as pd

from sklearn.model_selection import KFold, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_scor

from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor, HistGradientBoostingRegr
from sklearn.multioutput import MultiOutputRegressor

warnings.filterwarnings("ignore")
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)

# Paths — adjust if needed
DF1_PATH   = "df1.csv"
DF2_PATH   = "df2.csv"
DFVAL_PATH = "dfval.csv"
XTEST_PATH = "Xtest.csv"

df1   = pd.read_csv(DF1_PATH)
df2   = pd.read_csv(DF2_PATH)
dfval = pd.read_csv(DFVAL_PATH)
Xtest = pd.read_csv(XTEST_PATH)

ALL_FEATURES = [
    "RhythmScore",
    "AudioLoudness",
    "VocalContent",
    "AcousticQuality",
    "InstrumentalScore",
    "LivePerformanceLikelihood",
    "MoodScore",
    "TrackDurationMs",
    "BeatsPerMinute",
]
TARGET = "Energy"

# Infer feature sets actually present
features_df1 = [c for c in df1.columns if c in ALL_FEATURES]
features_df2 = [c for c in df2.columns if c in ALL_FEATURES]
features_val = [c for c in dfval.columns if c in ALL_FEATURES]
features_tst = [c for c in Xtest.columns if c in ALL_FEATURES]

common_feats = sorted(list(set(features_df1) & set(features_df2) & set(featu
full_feats   = sorted(list(set(ALL_FEATURES) & set(features_df2) & set(featu
missing_from_df1 = [f for f in full_feats if f not in features_df1]

print("common_feats:", common_feats)
```

```
print("full_feats:", full_feats)
print("missing_from_df1:", missing_from_df1)

# Train/val matrices for different strategies
# Strategy A: common features on df1 ∪ df2
X_A_train = pd.concat([df1[common_feats], df2[common_feats]], axis=0)
y_A_train = pd.concat([df1[TARGET], df2[TARGET]], axis=0)
X_A_val   = dfval[common_feats]
y_A_val   = dfval[TARGET]

# Strategy B: full features on df2
X_B_train = df2[full_feats]
y_B_train = df2[TARGET]
X_B_val   = dfval[full_feats]
y_B_val   = dfval[TARGET]

# Strategy C prep inputs (imputation needed later)
X_C_df2_common = df2[common_feats].copy()
Y_C_df2_missing = df2[missing_from_df1].copy() if len(missing_from_df1) > 0

# Preprocessors
def linear_preproc():
    return Pipeline([("imp", SimpleImputer(strategy="median")),
                     ("sc", StandardScaler())])

def tree_preproc():
    # trees don't need scaling
    return Pipeline([("imp", SimpleImputer(strategy="median"))])

# Utility: metrics
def metrics(y_true, y_pred, label=""):
    # y_true / y_pred to 1D arrays (defensive)
    y_true = np.asarray(y_true).ravel()
    y_pred = np.asarray(y_pred).ravel()
    # RMSE: prefer squared=False if available; otherwise sqrt(MSE)
    try:
        rmse = mean_squared_error(y_true, y_pred, squared=False)
    except TypeError:
        rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r2  = r2_score(y_true, y_pred)
    return {"label": label, "RMSE": rmse, "MAE": mae, "R2": r2}
def print_metrics_table(rows, title):
    dfm = pd.DataFrame(rows).set_index("label").sort_values("RMSE")
    print("\n" + title)
    print(dfm)
    return dfm

# Utility: cross-validated grid-search on *training only*
def cv_grid_search(pipe, param_grid, X, y, splits=5):
    n = len(X)
    n_splits = min(max(2, splits), max(2, n))  # safe for tiny n
    cv = KFold(n_splits=n_splits, shuffle=True, random_state=RANDOM_STATE)
    gs = GridSearchCV(pipe, param_grid=param_grid, cv=cv,
                      scoring="neg_root_mean_squared_error",
                      n_jobs=-1, refit=True)
```

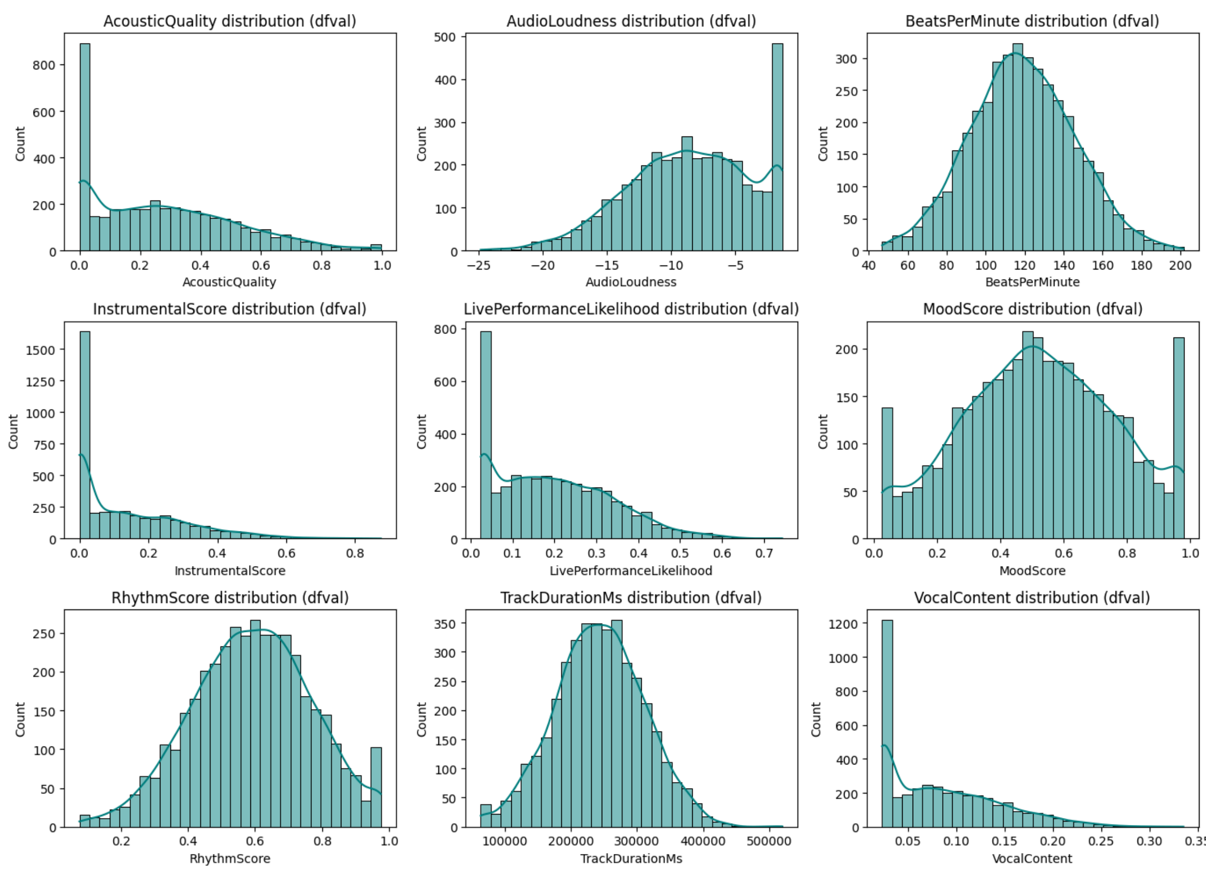
```
    gs.fit(X, y)
    return gs

# Storage for results across sections
MODEL_STORE = {}          # name -> fitted estimator (or tuple if blender)
VAL_PRED_STORE = {}       # name -> predictions on dfval
RESULTS = []              # list of metrics dicts (train & dfval)
```

```
common_feats: ['AudioLoudness', 'InstrumentalScore', 'TrackDurationMs', 'VocalContent']
full_feats: ['AcousticQuality', 'AudioLoudness', 'BeatsPerMinute', 'InstrumentalScore', 'LivePerformanceLikelihood', 'MoodScore', 'RhythmScore', 'TrackDurationMs', 'VocalContent']
missing_from_df1: ['AcousticQuality', 'BeatsPerMinute', 'LivePerformanceLikelihood', 'MoodScore', 'RhythmScore']
```

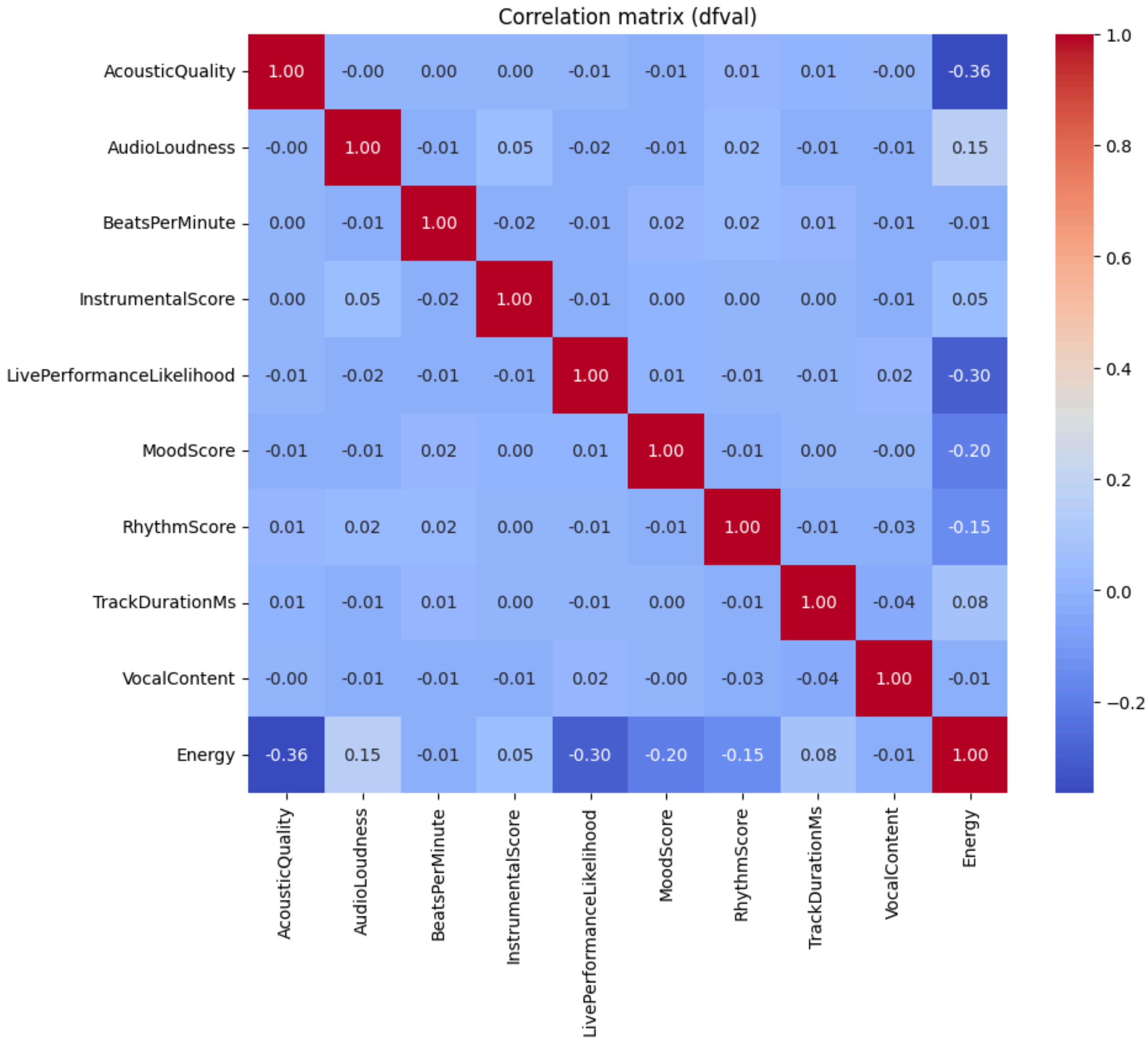
In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(14, 10))
for i, col in enumerate(full_feats, 1):
    plt.subplot(3, 3, i)
    sns.histplot(dfval[col], kde=True, color="teal", bins=30)
    plt.title(f"{col} distribution (dfval)")
plt.tight_layout()
plt.show()
```

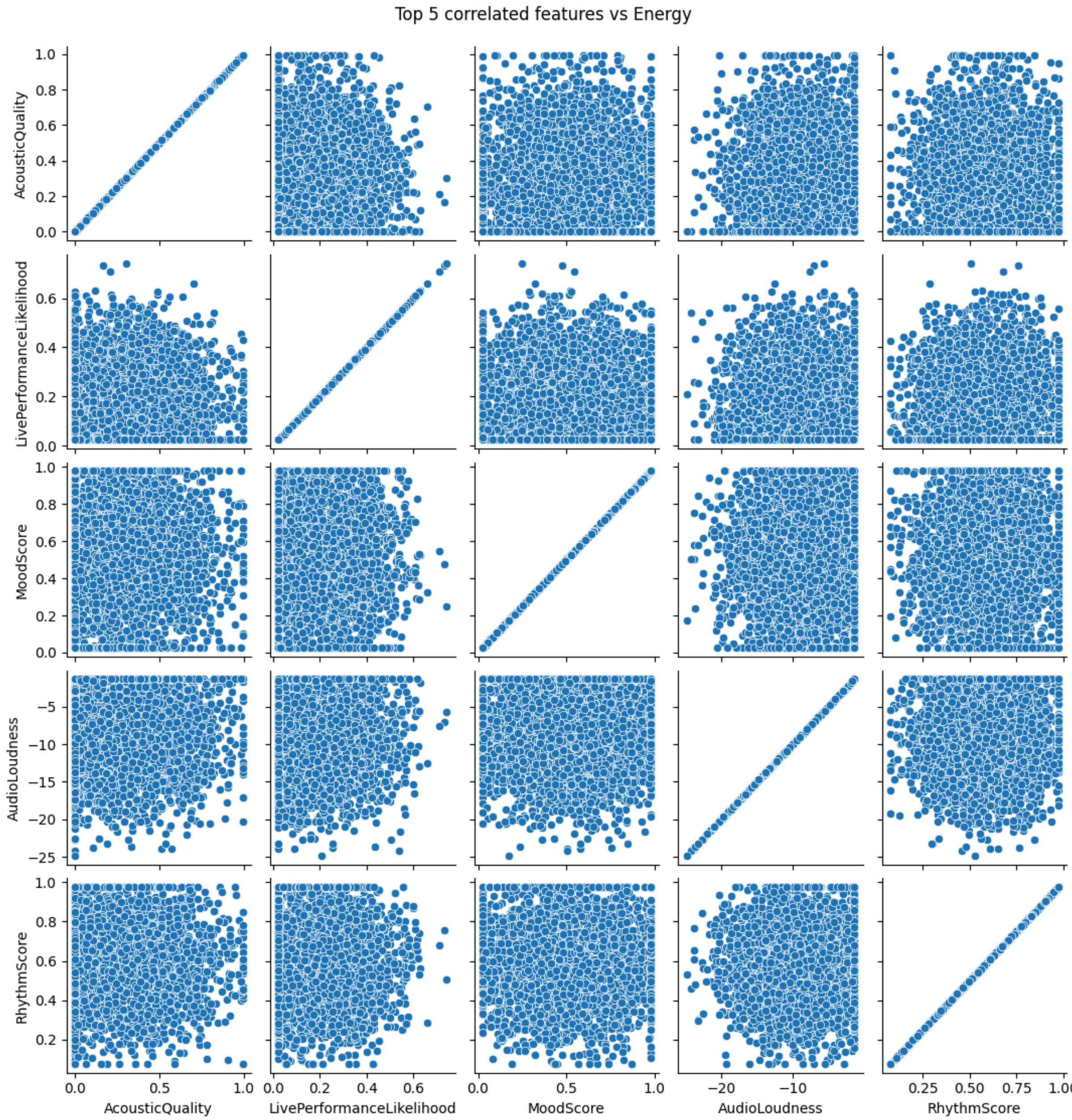


In [3]:
```
plt.figure(figsize=(10, 8))
corr = dfval[full_feats + [TARGET]].corr()
sns.heatmap(corr, cmap="coolwarm", annot=True, fmt=".2f", square=True)
```

```
plt.title("Correlation matrix (dfval)")
plt.show()
```



In [4]:
```
top_corrs = corr[TARGET].abs().sort_values(ascending=False)[1:6].index
sns.pairplot(dfval, vars=top_corrs, y_vars=[TARGET], kind="scatter", diag_ki
plt.suptitle("Top 5 correlated features vs Energy", y=1.02)
plt.show()
```



In [5]:
```
# ==== [1A] Strategy A — Ridge on common features ====
pipe = Pipeline([
    ("prep", linear_preproc()),
    ("est", Ridge(random_state=RANDOM_STATE))
])

grid = {"est__alpha": [0.1, 1.0, 3.0, 10.0]}

gs = cv_grid_search(pipe, grid, X_A_train, y_A_train, splits=5)
best = gs.best_estimator_

# Fit and evaluate
best.fit(X_A_train, y_A_train)
pred_tr = best.predict(X_A_train)
pred_val = best.predict(X_A_val)

RESULTS += [
    metrics(y_A_train, pred_tr, "A_Ridge | In-sample"),
```

```
    metrics(y_A_val,   pred_val, "A_Ridge | dfval"),
]
MODEL_STORE["A_Ridge"] = best
VAL_PRED_STORE["A_Ridge"] = pred_val

print_metrics_table([r for r in RESULTS if r["label"].startswith("A_Ridge")]
                    "Ridge (Strategy A) metrics")

# Optional: write Xtest predictions
# pd.DataFrame({"Energy": best.predict(Xtest[common_feats])}).to_csv("A_Ridg
```

```
Ridge (Strategy A) metrics
                         RMSE       MAE        R2
label
A_Ridge | In-sample  0.280837  0.241595  0.020102
A_Ridge | dfval      0.288830  0.250225  0.023690
```

Out[5]:

label	RMSE	MAE	R2
A_Ridge | In-sample	0.280837	0.241595	0.020102
A_Ridge | dfval	0.288830	0.250225	0.023690

In [6]:

```
# ==== [1B] Strategy A — Lasso on common features ====
pipe = Pipeline([
    ("prep", linear_preproc()),
    ("est", Lasso(random_state=RANDOM_STATE, max_iter=10000))
])

grid = {"est__alpha": [0.001, 0.01, 0.1, 1.0]}

gs = cv_grid_search(pipe, grid, X_A_train, y_A_train, splits=5)
best = gs.best_estimator_

best.fit(X_A_train, y_A_train)
pred_tr = best.predict(X_A_train)
pred_val = best.predict(X_A_val)

RESULTS += [
    metrics(y_A_train, pred_tr, "A_Lasso | In-sample"),
    metrics(y_A_val,   pred_val, "A_Lasso | dfval"),
]
MODEL_STORE["A_Lasso"] = best
VAL_PRED_STORE["A_Lasso"] = pred_val

print_metrics_table([r for r in RESULTS if r["label"].startswith("A_Lasso")]
                    "Lasso (Strategy A) metrics")

# Optional export:
# pd.DataFrame({"Energy": best.predict(Xtest[common_feats])}).to_csv("A_Lass
```

```
Lasso (Strategy A) metrics
                          RMSE       MAE        R2
label
A_Lasso | In-sample  0.280844  0.241653  0.020059
A_Lasso | dfval      0.288793  0.250230  0.023943
```

Out[6]:

label	RMSE	MAE	R2
A_Lasso | In-sample	0.280844	0.241653	0.020059
A_Lasso | dfval	0.288793	0.250230	0.023943

In [7]:

```
# ==== [1C] Strategy A — Elastic Net on common features ====
pipe = Pipeline([
    ("prep", linear_preproc()),
    ("est", ElasticNet(random_state=RANDOM_STATE, max_iter=10000))
])

grid = {
    "est__alpha": [0.01, 0.1, 1.0],
    "est__l1_ratio": [0.2, 0.5, 0.8]
}

gs = cv_grid_search(pipe, grid, X_A_train, y_A_train, splits=5)
best = gs.best_estimator_

best.fit(X_A_train, y_A_train)
pred_tr = best.predict(X_A_train)
pred_val = best.predict(X_A_val)

RESULTS += [
    metrics(y_A_train, pred_tr, "A_ElasticNet | In-sample"),
    metrics(y_A_val,   pred_val, "A_ElasticNet | dfval"),
]
MODEL_STORE["A_ElasticNet"] = best
VAL_PRED_STORE["A_ElasticNet"] = pred_val

print_metrics_table([r for r in RESULTS if r["label"].startswith("A_ElasticN
                    "Elastic Net (Strategy A) metrics")

# Optional export:
# pd.DataFrame({"Energy": best.predict(Xtest[common_feats])}).to_csv("A_Elas
```

```
Elastic Net (Strategy A) metrics
                               RMSE       MAE        R2
label
A_ElasticNet | In-sample  0.280866  0.241737  0.019901
A_ElasticNet | dfval      0.288788  0.250272  0.023974
```

Out[7]:

label	RMSE	MAE	R2
A_ElasticNet | In-sample	0.280866	0.241737	0.019901
A_ElasticNet | dfval	0.288788	0.250272	0.023974

In [8]:

```
# ==== [1D] Strategy A — Random Forest on common features ====
pipe = Pipeline([
    ("prep", tree_preproc()),
    ("est", RandomForestRegressor(random_state=RANDOM_STATE, n_jobs=-1))
])

grid = {
    "est__n_estimators": [200],
    "est__max_depth": [None, 6, 10],
    "est__min_samples_leaf": [1, 3, 5]
}

gs = cv_grid_search(pipe, grid, X_A_train, y_A_train, splits=5)
best = gs.best_estimator_

best.fit(X_A_train, y_A_train)
pred_tr = best.predict(X_A_train)
pred_val = best.predict(X_A_val)

RESULTS += [
    metrics(y_A_train, pred_tr, "A_RF | In-sample"),
    metrics(y_A_val,   pred_val, "A_RF | dfval"),
]
MODEL_STORE["A_RF"] = best
VAL_PRED_STORE["A_RF"] = pred_val

print_metrics_table([r for r in RESULTS if r["label"].startswith("A_RF")],
                    "Random Forest (Strategy A) metrics")

# Optional export:
# pd.DataFrame({"Energy": best.predict(Xtest[common_feats])}).to_csv("A_RF_X
```

```
Random Forest (Strategy A) metrics
                      RMSE       MAE        R2
label
A_RF | In-sample  0.238242  0.205192  0.294810
A_RF | dfval      0.284273  0.244386  0.054257
```

Out[8]:

label	RMSE	MAE	R2
A_RF | In-sample	0.238242	0.205192	0.294810
A_RF | dfval	0.284273	0.244386	0.054257

In [9]:

```
# ==== [1E] Strategy A — HistGradientBoosting on common features ====
pipe = Pipeline([
```

```
    ("prep", tree_preproc()),
    ("est", HistGradientBoostingRegressor(random_state=RANDOM_STATE))
])

grid = {
    "est__max_depth": [None, 6],
    "est__min_samples_leaf": [20, 50],
    "est__learning_rate": [0.05, 0.1],
}

gs = cv_grid_search(pipe, grid, X_A_train, y_A_train, splits=5)
best = gs.best_estimator_

best.fit(X_A_train, y_A_train)
pred_tr = best.predict(X_A_train)
pred_val = best.predict(X_A_val)

RESULTS += [
    metrics(y_A_train, pred_tr, "A_HGB | In-sample"),
    metrics(y_A_val,   pred_val, "A_HGB | dfval"),
]
MODEL_STORE["A_HGB"] = best
VAL_PRED_STORE["A_HGB"] = pred_val

print_metrics_table([r for r in RESULTS if r["label"].startswith("A_HGB")],
                    "HistGradientBoosting (Strategy A) metrics")

# Optional export:
# pd.DataFrame({"Energy": best.predict(Xtest[common_feats])}).to_csv("A_HGB_
```

```
HistGradientBoosting (Strategy A) metrics
                       RMSE       MAE        R2
label
A_HGB | In-sample  0.242255  0.207563  0.270854
A_HGB | dfval      0.284231  0.243286  0.054534
```

Out[9]:

label	RMSE	MAE	R2
A_HGB | In-sample	0.242255	0.207563	0.270854
A_HGB | dfval	0.284231	0.243286	0.054534

In [10]:

```
# ==== [2A] Strategy B — Ridge on full 9 features (df2 only) ====
pipe = Pipeline([
    ("prep", linear_preproc()),
    ("est", Ridge(random_state=RANDOM_STATE))
])

grid = {"est__alpha": [0.1, 1.0, 3.0, 10.0]}

gs = cv_grid_search(pipe, grid, X_B_train, y_B_train, splits=5)
best = gs.best_estimator_

best.fit(X_B_train, y_B_train)
pred_tr = best.predict(X_B_train)
```

```
pred_val = best.predict(X_B_val)

RESULTS += [
    metrics(y_B_train, pred_tr, "B_Ridge | In-sample"),
    metrics(y_B_val,   pred_val, "B_Ridge | dfval"),
]
MODEL_STORE["B_Ridge"] = best
VAL_PRED_STORE["B_Ridge"] = pred_val

print_metrics_table([r for r in RESULTS if r["label"].startswith("B_Ridge")]
                    "Ridge (Strategy B) metrics")

# Optional export:
# pd.DataFrame({"Energy": best.predict(Xtest[full_feats])}).to_csv("B_Ridge_
```

```
Ridge (Strategy B) metrics
                         RMSE       MAE        R2
label
B_Ridge | In-sample  0.229945  0.190094  0.417242
B_Ridge | dfval      0.263456  0.216947  0.187699
```

Out[10]:

label	RMSE	MAE	R2
B_Ridge | In-sample	0.229945	0.190094	0.417242
B_Ridge | dfval	0.263456	0.216947	0.187699

In [11]:

```
# ==== [2B] Strategy B — Elastic Net on full 9 features (df2 only) ====
pipe = Pipeline([
    ("prep", linear_preproc()),
    ("est", ElasticNet(random_state=RANDOM_STATE, max_iter=10000))
])

grid = {
    "est__alpha": [0.01, 0.1, 1.0],
    "est__l1_ratio": [0.2, 0.5, 0.8]
}

gs = cv_grid_search(pipe, grid, X_B_train, y_B_train, splits=5)
best = gs.best_estimator_

best.fit(X_B_train, y_B_train)
pred_tr = best.predict(X_B_train)
pred_val = best.predict(X_B_val)

RESULTS += [
    metrics(y_B_train, pred_tr, "B_ElasticNet | In-sample"),
    metrics(y_B_val,   pred_val, "B_ElasticNet | dfval"),
]
MODEL_STORE["B_ElasticNet"] = best
VAL_PRED_STORE["B_ElasticNet"] = pred_val

print_metrics_table([r for r in RESULTS if r["label"].startswith("B_ElasticN
                    "Elastic Net (Strategy B) metrics")
```

```
# Optional export:
# pd.DataFrame({"Energy": best.predict(Xtest[full_feats])}).to_csv("B_Elasti
```

```
Elastic Net (Strategy B) metrics
                              RMSE       MAE        R2
label
B_ElasticNet | In-sample  0.230092  0.190302  0.416496
B_ElasticNet | dfval      0.262705  0.216425  0.192321
```

Out[11]:

label	RMSE	MAE	R2
B_ElasticNet | In-sample	0.230092	0.190302	0.416496
B_ElasticNet | dfval	0.262705	0.216425	0.192321

In [12]:

```
# ==== [2C] Strategy B — HistGradientBoosting on full 9 features (df2 only)
pipe = Pipeline([
    ("prep", tree_preproc()),
    ("est", HistGradientBoostingRegressor(random_state=RANDOM_STATE))
])

grid = {
    "est__max_depth": [None, 6],
    "est__min_samples_leaf": [5, 20, 50],
    "est__learning_rate": [0.05, 0.1],
}

gs = cv_grid_search(pipe, grid, X_B_train, y_B_train, splits=5)
best = gs.best_estimator_

best.fit(X_B_train, y_B_train)
pred_tr = best.predict(X_B_train)
pred_val = best.predict(X_B_val)

RESULTS += [
    metrics(y_B_train, pred_tr, "B_HGB | In-sample"),
    metrics(y_B_val,   pred_val, "B_HGB | dfval"),
]
MODEL_STORE["B_HGB"] = best
VAL_PRED_STORE["B_HGB"] = pred_val

print_metrics_table([r for r in RESULTS if r["label"].startswith("B_HGB")],
                    "HistGradientBoosting (Strategy B) metrics")

# Optional export:
# pd.DataFrame({"Energy": best.predict(Xtest[full_feats])}).to_csv("B_HGB_Xt
```

```
HistGradientBoosting (Strategy B) metrics
                       RMSE       MAE        R2
label
B_HGB | In-sample  0.209634  0.171864  0.515644
B_HGB | dfval      0.266118  0.224043  0.171200
```

Out[12]:

label	RMSE	MAE	R2
B_HGB | In-sample	0.209634	0.171864	0.515644
B_HGB | dfval	0.266118	0.224043	0.171200

In [13]:

```
# ==== [3A] Strategy C — Impute (4->5) + Ridge on augmented 9D ====
if len(missing_from_df1) == 0:
    raise RuntimeError("No missing features in df1 relative to full_feats; S

# 1) Fit imputer on df2: map common_feats -> missing_from_df1
imputer = Pipeline([
    ("prep", linear_preproc()),
    ("est", MultiOutputRegressor(Ridge(alpha=1.0, random_state=RANDOM_STATE)
])
imputer.fit(X_C_df2_common, Y_C_df2_missing)

# 2) Impute df1's missing 5, build augmented 9D training set
imputed_missing = imputer.predict(df1[common_feats])
imputed_missing_df = pd.DataFrame(imputed_missing, columns=missing_from_df1,
df1_aug = pd.concat([df1[common_feats], imputed_missing_df, df1[[TARGET]]],
df1_aug = df1_aug[full_feats + [TARGET]]  # order

train_aug = pd.concat([df2[full_feats + [TARGET]], df1_aug], axis=0)
X_C_train = train_aug[full_feats].copy()
y_C_train = train_aug[TARGET].copy()

# weights: down-weight imputed rows (those after df2)
w = np.ones(len(train_aug))
w[len(df2):] = 0.5

# 3) Ridge on augmented 9D
pipe = Pipeline([
    ("prep", linear_preproc()),
    ("est", Ridge(random_state=RANDOM_STATE))
])
grid = {"est__alpha": [0.1, 1.0, 3.0, 10.0]}

gs = cv_grid_search(pipe, grid, X_C_train, y_C_train, splits=5)
best = gs.best_estimator_
best.fit(X_C_train, y_C_train, **{"est__sample_weight": w} if "est__sample_w

# Evaluate on dfval (no leakage)
pred_tr = best.predict(X_C_train)
pred_val = best.predict(dfval[full_feats])

RESULTS += [
    metrics(y_C_train, pred_tr, "C_Ridge | In-sample (augmented)"),
    metrics(dfval[TARGET], pred_val, "C_Ridge | dfval"),
]
MODEL_STORE["C_Ridge"] = (best, imputer)  # store imputer too
VAL_PRED_STORE["C_Ridge"] = pred_val
```

```
print_metrics_table([r for r in RESULTS if r["label"].startswith("C_Ridge")]
                    "Ridge (Strategy C) metrics")

# Optional export for Xtest (need to impute Xtest missing 5 from its common
Xtest_missing = imputer.predict(Xtest[common_feats])
Xtest_full = pd.concat([Xtest[common_feats],
                        pd.DataFrame(Xtest_missing, columns=missing_from_df1
# pd.DataFrame({"Energy": best.predict(Xtest_full)}).to_csv("C_Ridge_Xtest.c
```

```
Ridge (Strategy C) metrics
                                     RMSE       MAE        R2
label
C_Ridge | dfval                  0.266845  0.218232  0.166664
C_Ridge | In-sample (augmented)  0.278971  0.240076  0.033085
```

In [14]:

```
# ==== [3B] Strategy C — Elastic Net on augmented 9D ====
# Reuse imputer, X_C_train, y_C_train, w, full_feats from previous cell

pipe = Pipeline([
    ("prep", linear_preproc()),
    ("est", ElasticNet(random_state=RANDOM_STATE, max_iter=10000))
])

grid = {
    "est__alpha": [0.01, 0.1, 1.0],
    "est__l1_ratio": [0.2, 0.5, 0.8]
}

gs = cv_grid_search(pipe, grid, X_C_train, y_C_train, splits=5)
best = gs.best_estimator_
best.fit(X_C_train, y_C_train, **{"est__sample_weight": w} if "est__sample_w

pred_tr = best.predict(X_C_train)
pred_val = best.predict(dfval[full_feats])

RESULTS += [
    metrics(y_C_train, pred_tr, "C_ElasticNet | In-sample (augmented)"),
    metrics(dfval[TARGET], pred_val, "C_ElasticNet | dfval"),
]
MODEL_STORE["C_ElasticNet"] = (best, imputer)
VAL_PRED_STORE["C_ElasticNet"] = pred_val

print_metrics_table([r for r in RESULTS if r["label"].startswith("C_ElasticN
                    "Elastic Net (Strategy C) metrics")

# Optional export:
# Xtest_missing = imputer.predict(Xtest[common_feats])
# Xtest_full = pd.concat([Xtest[common_feats],
#                         pd.DataFrame(Xtest_missing, columns=missing_from_c
# pd.DataFrame({"Energy": best.predict(Xtest_full)}).to_csv("C_ElasticNet_Xt
```

```
Elastic Net (Strategy C) metrics
                                          RMSE       MAE        R2
label
C_ElasticNet | dfval                  0.251808  0.212191  0.257937
C_ElasticNet | In-sample (augmented)  0.279201  0.240444  0.031488
```

Out[14]:

label	RMSE	MAE	R2
C_ElasticNet | dfval	0.251808	0.212191	0.257937
C_ElasticNet | In-sample (augmented)	0.279201	0.240444	0.031488

In [15]:

```
# ==== [3C] Strategy C — HistGradientBoosting on augmented 9D ====
pipe = Pipeline([
    ("prep", tree_preproc()),
    ("est", HistGradientBoostingRegressor(random_state=RANDOM_STATE))
])

grid = {
    "est__max_depth": [None, 6],
    "est__min_samples_leaf": [20, 50],
    "est__learning_rate": [0.05, 0.1],
}

gs = cv_grid_search(pipe, grid, X_C_train, y_C_train, splits=5)
best = gs.best_estimator_
best.fit(X_C_train, y_C_train, **{"est__sample_weight": w} if "est__sample_w

pred_tr = best.predict(X_C_train)
pred_val = best.predict(dfval[full_feats])

RESULTS += [
    metrics(y_C_train, pred_tr, "C_HGB | In-sample (augmented)"),
    metrics(dfval[TARGET], pred_val, "C_HGB | dfval"),
]
MODEL_STORE["C_HGB"] = (best, imputer)
VAL_PRED_STORE["C_HGB"] = pred_val

print_metrics_table([r for r in RESULTS if r["label"].startswith("C_HGB")],
                    "HistGradientBoosting (Strategy C) metrics")

# Optional export:
# Xtest_missing = imputer.predict(Xtest[common_feats])
# Xtest_full = pd.concat([Xtest[common_feats],
#                         pd.DataFrame(Xtest_missing, columns=missing_from_c
# pd.DataFrame({"Energy": best.predict(Xtest_full)}).to_csv("C_HGB_Xtest.csv
```

```
HistGradientBoosting (Strategy C) metrics
                                   RMSE       MAE        R2
label
C_HGB | In-sample (augmented)  0.234765  0.200893  0.315239
C_HGB | dfval                  0.263276  0.226940  0.188804
```

Out[15]:

label	RMSE	MAE	R2
C_HGB | In-sample (augmented)	0.234765	0.200893	0.315239
C_HGB | dfval	0.263276	0.226940	0.188804

In [16]:
```
# ==== [4] Strategy D — Blend best A and best B (fixed idxmin + guards) ====

# Helper: list the dfval rows we actually have (debug aid)
def _dfval_rows(prefix=None):
    rows = [r for r in RESULTS if r.get("label","").endswith("| dfval")]
    if prefix:
        rows = [r for r in rows if r["label"].startswith(prefix)]
    return pd.DataFrame(rows)

# Pick the best (lowest dfval RMSE) model by prefix
def best_by_prefix(prefix: str):
    dfm = _dfval_rows(prefix)
    if dfm.empty:
        return None
    # Coerce RMSE to numeric and drop NaNs if any
    dfm = dfm.copy()
    dfm["RMSE"] = pd.to_numeric(dfm["RMSE"], errors="coerce")
    dfm = dfm.dropna(subset=["RMSE"])
    if dfm.empty:
        return None
    winner_row = dfm.loc[dfm["RMSE"].idxmin()]
    return winner_row["label"].split(" | ")[0]  # strip the " | dfval"

best_A = best_by_prefix("A_")
best_B = best_by_prefix("B_")

print("Best A:", best_A)
print("Best B:", best_B)

if best_A is None or best_B is None:
    print("\nNo eligible models found to blend.")
    print("Make sure you ran at least one Strategy A cell AND one Strategy B
    print("Current dfval entries:\n", _dfval_rows())
    raise RuntimeError("Run Strategy A and Strategy B training cells before

A_model = MODEL_STORE[best_A]
B_model = MODEL_STORE[best_B]

# 2) Train blender on df2 using predictions from A (common feats) and B (ful
from sklearn.linear_model import Ridge as RidgeBlender
A_on_df2 = A_model.predict(df2[common_feats])
B_on_df2 = B_model.predict(df2[full_feats])
Z_train = np.vstack([A_on_df2, B_on_df2]).T
y_train_blend = df2[TARGET].values

blender = RidgeBlender(alpha=0.1, random_state=RANDOM_STATE)
blender.fit(Z_train, y_train_blend)

# 3) Evaluate on dfval
A_on_val = A_model.predict(dfval[common_feats])
B_on_val = B_model.predict(dfval[full_feats])
Z_val = np.vstack([A_on_val, B_on_val]).T
pred_val = blender.predict(Z_val)

RESULTS += [
```

```
    metrics(y_train_blend, blender.predict(Z_train), "D_Blend(A,B) | In-samp
    metrics(dfval[TARGET], pred_val,              "D_Blend(A,B) | dfval"),
]
MODEL_STORE["D_Blend(A,B)"] = (A_model, B_model, blender)
VAL_PRED_STORE["D_Blend(A,B)"] = pred_val

print("\nBlend (Strategy D) metrics:")
display(pd.DataFrame([r for r in RESULTS if r["label"].startswith("D_Blend")
```

```
Best A: A_HGB
Best B: B_ElasticNet

Blend (Strategy D) metrics:
```

label	RMSE	MAE	R2
D_Blend(A,B) | In-sample (df2)	0.206213	0.162419	0.531325
D_Blend(A,B) | dfval	0.253001	0.209384	0.250886

In [17]:

```
# ==== [14] Transform helpers ====
import numpy as np
import pandas as pd

# Heavily right-skewed features from your EDA
SKEWED_COLS = [
    "AcousticQuality",
    "InstrumentalScore",
    "VocalContent",
    "LivePerformanceLikelihood",
    "TrackDurationMs",
]

def apply_log1p(df: pd.DataFrame, cols: list) -> pd.DataFrame:
    df2 = df.copy()
    for c in cols:
        if c in df2.columns:
            # guard against negatives (shouldn't happen per your audit)
            df2[c] = np.log1p(np.clip(df2[c], a_min=0, a_max=None))
    return df2

# Transformed matrices for each strategy
X_A_train_tf = apply_log1p(pd.concat([df1[common_feats], df2[common_feats]],
X_A_val_tf   = apply_log1p(dfval[common_feats], SKEWED_COLS)

X_B_train_tf = apply_log1p(df2[full_feats], SKEWED_COLS)
X_B_val_tf   = apply_log1p(dfval[full_feats], SKEWED_COLS)

# For C: imputer learns mapping from COMMON (transformed) -> MISSING (raw)
X_C_df2_common_tf = apply_log1p(df2[common_feats], SKEWED_COLS)
```

In [20]:

```
# ==== [15] XGBoost for B (df2 only) and C (augmented 9D) ====
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_scor
```

```
try:
    import xgboost as xgb
    HAS_XGB = True
except Exception as e:
    HAS_XGB = False
    print("xgboost not available; falling back to HistGradientBoostingRegres

def rmse(y_true, y_pred):
    """Compute RMSE compatible with older sklearn versions."""
    return mean_squared_error(y_true, y_pred) ** 0.5


# ---------- B_XGB: train on df2 (9 features) ----------
if HAS_XGB:
    xgb_b = xgb.XGBRegressor(
        random_state=RANDOM_STATE,
        tree_method="hist",  # fast/robust
        n_estimators=600,
        max_depth=3,
        subsample=0.8,
        colsample_bytree=0.8,
        learning_rate=0.05,
        reg_alpha=0.0,
        reg_lambda=2.0,
        min_child_weight=5,
    )
else:
    from sklearn.ensemble import HistGradientBoostingRegressor
    xgb_b = HistGradientBoostingRegressor(
        max_depth=6, learning_rate=0.05, min_samples_leaf=20, random_state=R
    )

xgb_b.fit(X_B_train_tf, y_B_train)
pred_tr_b = xgb_b.predict(X_B_train_tf)
pred_val_b = xgb_b.predict(X_B_val_tf)

RESULTS += [
    {"label": "B_XGB | In-sample", "RMSE": rmse(y_B_train, pred_tr_b), "MAE"
    {"label": "B_XGB | dfval",     "RMSE": rmse(y_B_val,   pred_val_b), "MAE
]
MODEL_STORE["B_XGB"] = xgb_b
VAL_PRED_STORE["B_XGB"] = pred_val_b

print("B_XGB dfval RMSE:", rmse(y_B_val, pred_val_b))

# ---------- C_XGB: augmented 9D (impute df1's missing 5, then train on df2
# 1) Imputer: common(tf) -> missing(raw) using df2 only
from sklearn.multioutput import MultiOutputRegressor
from sklearn.linear_model import Ridge

imputer_c = Pipeline([
    ("prep", linear_preproc()),  # median + scale on common
    ("est", MultiOutputRegressor(Ridge(alpha=1.0, random_state=RANDOM_STATE)
])
imputer_c.fit(X_C_df2_common_tf, df2[missing_from_df1])  # note: targets are
```

```
# 2) Impute df1's missing 5
X1_common_tf = apply_log1p(df1[common_feats], SKEWED_COLS)
imputed_missing = imputer_c.predict(X1_common_tf)
imputed_df = pd.DataFrame(imputed_missing, columns=missing_from_df1, index=c

# 3) Build augmented 9D training set
train_aug_c = pd.concat(
    [df2[full_feats + [TARGET]], pd.concat([df1[common_feats], imputed_df, c
    axis=0
)
X_C_train_tf = apply_log1p(train_aug_c[full_feats], SKEWED_COLS)
y_C_train    = train_aug_c[TARGET]
X_C_val_tf   = apply_log1p(dfval[full_feats], SKEWED_COLS)

# down-weight imputed rows
w_c = np.ones(len(train_aug_c))
w_c[len(df2):] = 0.5

if HAS_XGB:
    xgb_c = xgb.XGBRegressor(
        random_state=RANDOM_STATE,
        tree_method="hist",
        n_estimators=900,
        max_depth=4,
        subsample=0.9,
        colsample_bytree=0.9,
        learning_rate=0.03,
        reg_alpha=0.0,
        reg_lambda=2.0,
        min_child_weight=3,
    )
else:
    from sklearn.ensemble import HistGradientBoostingRegressor
    xgb_c = HistGradientBoostingRegressor(
        max_depth=6, learning_rate=0.05, min_samples_leaf=20, random_state=R
    )

# NOTE: xgboost supports sample_weight directly; HGBR also supports sample_w
xgb_c.fit(X_C_train_tf, y_C_train, sample_weight=w_c)
pred_tr_c = xgb_c.predict(X_C_train_tf)
pred_val_c = xgb_c.predict(X_C_val_tf)

RESULTS += [
    {"label": "C_XGB | In-sample (augmented)", "RMSE": rmse(y_C_train, pred_
    {"label": "C_XGB | dfval",                 "RMSE": rmse(dfval[TARGET], p
]
MODEL_STORE["C_XGB"] = (xgb_c, imputer_c)
VAL_PRED_STORE["C_XGB"] = pred_val_c

print("C_XGB dfval RMSE:", rmse(dfval[TARGET], pred_val_c))
```

```
B_XGB dfval RMSE: 0.2704140945463544
C_XGB dfval RMSE: 0.2577286043761442
```

In [22]:
```
# ==== [16] Stacking / multi-model blend on df2 predictions ====
from sklearn.linear_model import Ridge as RidgeBlender

# --- Ensure required base models are trained (skip missing ones gracefully)
base_models = []

# Helper: safely fetch model and a predictor function
def _add_base(name, use_feats):
    if name not in MODEL_STORE:
        print(f"[stack] Skipping {name} (not found in MODEL_STORE).")
        return
    mdl = MODEL_STORE[name]
    if name.startswith("C_") and isinstance(mdl, tuple):
        # (estimator, imputer)
        est, imp = mdl
        def pred_fn_df2(df):
            # df has all 9; for C_* we don't need to re-impute df2 to predic
            X = apply_log1p(df[full_feats], SKEWED_COLS)
            return est.predict(X)
        def pred_fn_any(df):
            # for dfval/Xtest: just transformed 9D
            X = apply_log1p(df[full_feats], SKEWED_COLS)
            return est.predict(X)
    elif name.startswith("B_"):
        est = mdl
        def pred_fn_df2(df):
            X = apply_log1p(df[full_feats], SKEWED_COLS)
            return est.predict(X)
        def pred_fn_any(df):
            X = apply_log1p(df[full_feats], SKEWED_COLS)
            return est.predict(X)
    elif name.startswith("A_"):
        est = mdl
        def pred_fn_df2(df):
            X = apply_log1p(df[common_feats], SKEWED_COLS)
            return est.predict(X)
        def pred_fn_any(df):
            X = apply_log1p(df[common_feats], SKEWED_COLS)
            return est.predict(X)
    else:
        print(f"[stack] Unrecognized base '{name}', skipping.")
        return
    base_models.append((name, pred_fn_df2, pred_fn_any))

# Add candidates you have trained (names must match your earlier sections)
for cand in ["A_HGB", "B_ElasticNet", "B_XGB", "C_ElasticNet", "C_XGB"]:
    _add_base(cand, full_feats)

if len(base_models) < 2:
    raise RuntimeError("Need at least 2 base models for stacking. Train more

# --- Build Z_train from df2 predictions (no leakage)
Z_train = []
for (nm, pred_df2, _) in base_models:
    Z_train.append(pred_df2(df2))
```

```
Z_train = np.vstack(Z_train).T  # shape (len(df2), n_models)
y_train_blend = df2[TARGET].values

# --- Fit small ridge blender
blender = RidgeBlender(alpha=0.1, random_state=RANDOM_STATE)
blender.fit(Z_train, y_train_blend)

# --- Evaluate on dfval (out-of-sample)
Z_val = []
for (nm, _, pred_any) in base_models:
    Z_val.append(pred_any(dfval))
Z_val = np.vstack(Z_val).T
pred_val_blend = blender.predict(Z_val)

RESULTS += [
    {"label": "STACK | In-sample (df2)", "RMSE": rmse(y_train_blend, blender
    {"label": "STACK | dfval",           "RMSE": rmse(dfval[TARGET], pred_va
]
MODEL_STORE["STACK"] = (base_models, blender)
VAL_PRED_STORE["STACK"] = pred_val_blend

print("STACK dfval RMSE:", rmse(dfval[TARGET], pred_val_blend))
```

```
STACK dfval RMSE: 0.25353976784635635
```