v27_12_v2

December 27, 2025

[61]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score, f1_score, accuracy_score,␣
 ↪average_precision_score, confusion_matrix, classification_report
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

# ============================================================================
# Project: Next-year financial distress prediction (firm-year panel)
# Data Science Lifecycle focus in this cell:
#   - Setup & configuration (temporal split conventions, leakage control␣
 ↪parameters)
#   - Panel integrity (types, deduplication, stable firm identifier)
#   - Define label_year used consistently for splitting and target alignment
# ============================================================================

# ----------------------------
# Configuration (split & preprocessing parameters)
# ----------------------------
FILE_NAME = "data.csv"

TRAIN_CUTOFF_LABEL_YEAR = 2022   # label_year <= cutoff → train/val pool; after␣
 ↪cutoff → test
VAL_YEARS = 1                    # last N years within the pool are validation
N_SPLITS_TIME_CV = 4             # rolling time-based folds for sanity checks

WINSOR_LOWER_Q = 0.01            # winsorization lower quantile (train-only)
WINSOR_UPPER_Q = 0.99            # winsorization upper quantile (train-only)

REQUIRED_KEYS = ["gvkey", "fyear"]


# ----------------------------
# Utilities (robust numeric ops for ratios)
# ----------------------------
```

```python
def to_float_numpy(x) -> np.ndarray:
    """Convert series/array-like to float numpy array, coercing non-numeric to␣
 ↪NaN."""
    s = pd.to_numeric(x, errors="coerce")
    return s.to_numpy(dtype=float) if hasattr(s, "to_numpy") else np.asarray(s,␣
 ↪dtype=float)
def safe_divide(a, b) -> np.ndarray:
    """Elementwise divide a/b with NaN when division is invalid (0 or␣
 ↪non-finite)."""
    a = to_float_numpy(a)
    b = to_float_numpy(b)
    out = np.full_like(a, np.nan, dtype=float)
    np.divide(a, b, out=out, where=(b != 0) & np.isfinite(a) & np.isfinite(b))
    return out

def rolling_year_folds(
    df_in: pd.DataFrame, year_col: str = "label_year", n_splits: int = 4,␣
 ↪min_train_years: int = 3
) -> list[tuple[np.ndarray, np.ndarray, np.ndarray, int]]:
    """
    Create expanding-window time folds:
      train years: first (min_train_years + k) years
      val year:    next year
    Returns: list of (train_idx, val_idx, train_years, val_year)
    """
    years_sorted = np.sort(df_in[year_col].dropna().unique())
    if len(years_sorted) <= min_train_years:
        return []
    n_splits = min(n_splits, len(years_sorted) - min_train_years)

    folds_out = []
    for k in range(n_splits):
        train_years = years_sorted[: min_train_years + k]
        val_year = int(years_sorted[min_train_years + k])

        train_idx = df_in.index[df_in[year_col].isin(train_years)].to_numpy()
        val_idx = df_in.index[df_in[year_col] == val_year].to_numpy()
        folds_out.append((train_idx, val_idx, train_years, val_year))

    return folds_out


# ============================================================================
# 1) Data acquisition & panel hygiene
# ============================================================================
df = pd.read_csv(FILE_NAME, low_memory=False)
```

```
# Convert datadate if present
if "datadate" in df.columns:
    df["datadate"] = pd.to_datetime(df["datadate"], errors="coerce")

# Create stable firm id + de-duplicate firm-year (keep last record)
df["firm_id"] = df["gvkey"]
df = (
    df.sort_values(["firm_id", "fyear"])
      .drop_duplicates(subset=["firm_id", "fyear"], keep="last")
      .reset_index(drop=True)
)

# Label year: predict distress in the next fiscal year
df["label_year"] = df["fyear"] + 1

# ============================================================================
# 2) Split scaffolding (define train/val pool years via label_year)
# ============================================================================
pool_mask = df["label_year"] <= TRAIN_CUTOFF_LABEL_YEAR
pool_years = np.sort(df.loc[pool_mask, "label_year"].dropna().unique())
val_years = pool_years[-VAL_YEARS:] if len(pool_years) else np.array([],␣
 ↪dtype=int)

# This mask is ONLY used for imputations (train-only information)
train_mask_for_imputation = pool_mask & (~df["label_year"].isin(val_years))
```

[62]:
```
# ============================================================================
# Data Cleaning & Missing-Data Handling (leakage-aware)
# Purpose:
#   - Quantify missingness and distribution properties before intervention
#   - Preserve informative missingness via miss_* indicators
#   - Impute financial statement inputs using TRAIN-only information:
#       (1) within-firm past values (ffill) where economically sensible
#       (2) peer medians by year on size-scaled ratios
#   - Re-run EDA after imputation to audit how strongly imputations alter the␣
 ↪data
# ============================================================================

RAW_INPUTS_FOR_FE = [
   ␣
 ↪"aco","act","ao","aoloch","ap","apalch","aqc","at","caps","capx","ceq","che","chech","csho"
   ␣
 ↪"datadate","dlc","dlcch","dltis","dltr","dltt","do","dp","dpc","dv","dvc","dvp","dvt","esub
   ␣
 ↪"fiao","fincf","fopo","fyear","gvkey","ib","ibadj","ibc","intan","invch","invt","ismod","iv
   ␣
 ↪"ivao","ivch","ivncf","ivstch","lco","lct","lt","mibt","mkvalt","niadj","nopi","oancf","oib
```

```
    ␣
 ↪"prcc_c","prcc_f","prstkc","pstk","pstkn","pstkr","re","recch","rect","seq","siv","spi","sp
    ␣
 ↪"sstk","tstk","txach","txbcof","txdc","txditc","txp","txt","xi","xido","xidoc","xint",
    # optional identifiers present in many extracts:
    "conm","consol","datafmt","indfmt",
]
raw = [c for c in RAW_INPUTS_FOR_FE if c in df.columns]

# ------------------------------------------------------------------------
# 3.0 Ensure keys exist + types
# ------------------------------------------------------------------------
# firm_id used for panel operations
if "firm_id" not in df.columns:
    if "gvkey" in df.columns:
        df["firm_id"] = df["gvkey"]
    else:
        raise ValueError("Need either firm_id or gvkey in df to run panel␣
 ↪imputations.")

# fyear numeric
if "fyear" in df.columns:
    df["fyear"] = pd.to_numeric(df["fyear"], errors="coerce")

# datadate datetime (if present)
if "datadate" in df.columns:
    df["datadate"] = pd.to_datetime(df["datadate"], errors="coerce")

# ------------------------------------------------------------------------
# 3.1 Drop rows with missing critical identifiers (do not impute these)
# ------------------------------------------------------------------------
NON_IMPUTE_DROP = [c for c in ["gvkey", "datadate", "fyear", "conm", "datafmt",␣
 ↪"indfmt", "consol"] if c in df.columns]
if NON_IMPUTE_DROP:
    before_n = df.shape[0]
    df = df.dropna(subset=NON_IMPUTE_DROP).copy()
    after_n = df.shape[0]
    if after_n < before_n:
        print(f"[INFO] Dropped {before_n - after_n:,} rows due to missing␣
 ↪non-imputable ID/meta fields: {NON_IMPUTE_DROP}")

# Rebuild raw after potential drop
raw = [c for c in RAW_INPUTS_FOR_FE if c in df.columns]

# ------------------------------------------------------------------------
# 3.2 EDA BEFORE imputation (missingness + distribution snapshot)
# ------------------------------------------------------------------------
```

```
df_raw_pre = df[raw].copy(deep=True)

pre_miss = pd.DataFrame(
    {
        "col": raw,
        "n": [int(df_raw_pre[c].shape[0]) for c in raw],
        "n_na_pre": [int(df_raw_pre[c].isna().sum()) for c in raw],
        "pct_na_pre": [float(df_raw_pre[c].isna().mean() * 100.0) for c in raw],
        "train_n": [int(train_mask_for_imputation.sum()) for _ in raw],
        "train_pct_na_pre": [
            float(df_raw_pre.loc[train_mask_for_imputation, c].isna().mean() *␣
 ↪100.0) for c in raw
        ],
    }
).sort_values("pct_na_pre", ascending=False)

print("\n=== EDA (BEFORE imputation): Missingness on raw inputs ===")
print(pre_miss.round(4).head(50))

# Numeric distribution summary (exclude obvious non-numeric)
if raw:
    x_pre = df_raw_pre[raw].apply(pd.to_numeric, errors="coerce").replace([np.
 ↪inf, -np.inf], np.nan)
    q_pre = x_pre.quantile([0.01, 0.05, 0.25, 0.50, 0.75, 0.95, 0.99]).T
    pre_dist = pd.DataFrame(
        {
            "n_nonmiss_pre": x_pre.notna().sum(),
            "mean_pre": x_pre.mean(),
            "std_pre": x_pre.std(ddof=0),
            "min_pre": x_pre.min(),
            "p01_pre": q_pre[0.01],
            "p05_pre": q_pre[0.05],
            "p25_pre": q_pre[0.25],
            "p50_pre": q_pre[0.50],
            "p75_pre": q_pre[0.75],
            "p95_pre": q_pre[0.95],
            "p99_pre": q_pre[0.99],
            "max_pre": x_pre.max(),
        }
    )
    print("\n=== EDA (BEFORE imputation): Distribution summary (raw inputs)␣
 ↪===")
    print(pre_dist.round(4).sort_values("n_nonmiss_pre", ascending=True).
 ↪head(50))

# ------------------------------------------------------------------------
# 3.3 Missingness flags (ALWAYS create before imputations)
```

```
# ------------------------------------------------------------------------------
for c in raw:
    df[f"miss_{c}"] = df[c].isna().astype("int8")

# ------------------------------------------------------------------------------
# 3.4 Grouping key for peer-based imputations
# ------------------------------------------------------------------------------
group_cols = ["fyear"]

# 3.5 Step 1: Construct / reconcile FIRST (no leakage: uses contemporaneous or␣
 ↪lag only)
# ------------------------------------------------------------------------------
df = df.sort_values(["firm_id", "fyear"]).copy()

# 3.5.1 mkvalt construction from prcc_f * csho (if mkvalt missing)
if all(c in df.columns for c in ["mkvalt", "prcc_f", "csho"]):
    mkvalt_miss = df["mkvalt"].isna()
    mkvalt_calc = pd.to_numeric(df["prcc_f"], errors="coerce") * pd.
 ↪to_numeric(df["csho"], errors="coerce")
    df.loc[mkvalt_miss & mkvalt_calc.notna(), "mkvalt"] = mkvalt_calc.
 ↪loc[mkvalt_miss & mkvalt_calc.notna()]

# 3.5.2 Reconstruct change variables from level differences (fill only if␣
 ↪change var is missing)
def _fill_change_from_levels(change_col, level_col):
    if change_col in df.columns and level_col in df.columns:
        miss = df[change_col].isna()
        lvl = pd.to_numeric(df[level_col], errors="coerce")
        lag_lvl = df.groupby("firm_id")[level_col].shift(1)
        lag_lvl = pd.to_numeric(lag_lvl, errors="coerce")
        recon = lvl - lag_lvl
        df.loc[miss & recon.notna(), change_col] = recon.loc[miss & recon.
 ↪notna()]

_fill_change_from_levels("dlcch", "dlc")
_fill_change_from_levels("recch", "rect")
_fill_change_from_levels("invch", "invt")
_fill_change_from_levels("chech", "che")

# 3.5.3 apalch proxy with Δap if apalch missing
if "apalch" in df.columns and "ap" in df.columns:
    miss = df["apalch"].isna()
    ap = pd.to_numeric(df["ap"], errors="coerce")
    lag_ap = pd.to_numeric(df.groupby("firm_id")["ap"].shift(1),␣
 ↪errors="coerce")
    recon = ap - lag_ap
```

```
    df.loc[miss & recon.notna(), "apalch"] = recon.loc[miss & recon.notna()]

# ------------------------------------------------------------------------
# 3.6 Step 2: ismod mode imputation (binary flag)
# ------------------------------------------------------------------------
if "ismod" in df.columns:
    tr_obs = df.loc[train_mask_for_imputation, "ismod"]
    tr_obs_num = pd.to_numeric(tr_obs, errors="coerce")
    # global mode on training
    if tr_obs_num.notna().any():
        mode_val = float(tr_obs_num.mode(dropna=True).iloc[0])
    else:
        mode_val = 0.0
    df.loc[df["ismod"].isna(), "ismod"] = mode_val

# ------------------------------------------------------------------------
# 3.7 Step 3: Stocks - firm-history (ffill) -> group median of ratio (x/at)
# ------------------------------------------------------------------------
STOCKS = [c for c in [
   ␣
 ↪"aco","act","ao","ap","at","caps","ceq","che","csho","cstk","dlc","dltt","intan","invt","lc
   ␣
 ↪"mibt","ppent","pstk","pstkn","pstkr","re","rect","seq","tstk","ivaeq","mkvalt"
] if c in df.columns]

# firm-history forward fill within each firm (uses only past observations)
if STOCKS:
    df[STOCKS] = df.groupby("firm_id")[STOCKS].ffill()

# ratio median impute remaining missing stocks (fit on training only)
NONNEG_STOCKS = set([c for c in STOCKS if c in␣
 ↪{"aco","act","ao","ap","at","caps","ceq","che","csho","cstk","dlc","dltt",
                                                  ␣
 ↪"intan","invt","lco","lct","lt","mibt","mkvalt","ppent","pstk","pstkn","pstkr",
                                                    "rect","seq","tstk","ivaeq"}])

def _fit_ratio_medians(train_df, col, size_col="at"):
    # returns (overall_ratio_median, series indexed by group tuple)
    s = pd.to_numeric(train_df[col], errors="coerce")
    size = pd.to_numeric(train_df[size_col], errors="coerce") if size_col in␣
 ↪train_df.columns else None
    if size is None:
        # fallback: level median by groups
        grp_med = train_df.groupby(group_cols)[col].median()
        overall = float(s.median()) if s.notna().any() else 0.0
        return ("level", overall, grp_med)
```

```
    valid = s.notna() & size.notna() & (size > 0)
    if valid.sum() == 0:
        grp_med = train_df.groupby(group_cols)[col].median()
        overall = float(s.median()) if s.notna().any() else 0.0
        return ("level", overall, grp_med)

    ratio = (s[valid] / size[valid]).replace([np.inf, -np.inf], np.nan).dropna()
    overall = float(ratio.median()) if ratio.notna().any() else 0.0
    tmp = train_df.loc[valid, group_cols].copy()
    tmp["_ratio_"] = ratio.values
    grp_med = tmp.groupby(group_cols)["_ratio_"].median()
    return ("ratio", overall, grp_med)

def _apply_ratio_medians(df_all, col, fit_obj, size_col="at", nonneg=False):
    kind, overall, grp_med = fit_obj
    miss = df_all[col].isna()
    if not miss.any():
        return

    if kind == "ratio" and size_col in df_all.columns:
        size = pd.to_numeric(df_all.loc[miss, size_col], errors="coerce")
        # map group median ratio
        g = df_all.loc[miss, group_cols]
        mapped = pd.Series([np.nan] * miss.sum(), index=df_all.index[miss],␣
↪dtype="float64")
        # build tuple key if multi-col grouping
        if len(group_cols) == 1:
            mapped = g[group_cols[0]].map(grp_med)
        else:
            keys = list(map(tuple, g[group_cols].to_numpy()))
            mapped = pd.Series(keys, index=g.index).map(grp_med)

        r = pd.to_numeric(mapped, errors="coerce").fillna(overall)
        fill = r * size
        # if size missing/<=0, fallback to level median within groups
        fill = fill.where(size.notna() & (size > 0), np.nan)
        df_all.loc[miss & fill.notna(), col] = fill.loc[miss & fill.notna()].
↪to_numpy()

    # fallback: level median by group (still fit on training)
    # (use training group median of levels if ratio path didn't fill everything)
    miss2 = df_all[col].isna()
    if miss2.any():
        tr = df_all.loc[train_mask_for_imputation, [*group_cols, col]].copy()
        tr[col] = pd.to_numeric(tr[col], errors="coerce")
        lvl_overall = float(tr[col].median()) if tr[col].notna().any() else 0.0
        lvl_grp = tr.groupby(group_cols)[col].median()
```

```
        g2 = df_all.loc[miss2, group_cols]
        if len(group_cols) == 1:
            mapped2 = g2[group_cols[0]].map(lvl_grp)
        else:
            keys2 = list(map(tuple, g2[group_cols].to_numpy()))
            mapped2 = pd.Series(keys2, index=g2.index).map(lvl_grp)

        fill2 = pd.to_numeric(mapped2, errors="coerce").fillna(lvl_overall)
        if nonneg:
            fill2 = fill2.clip(lower=0.0)
        df_all.loc[miss2, col] = fill2.to_numpy()

# Fit on training
tr_all = df.loc[train_mask_for_imputation].copy()
ratio_fits = {}
for c in STOCKS:
    ratio_fits[c] = _fit_ratio_medians(tr_all, c, size_col="at")

# Apply to full df
for c in STOCKS:
    _apply_ratio_medians(df, c, ratio_fits[c], size_col="at", nonneg=(c in␣
 ↪NONNEG_STOCKS))

# ------------------------------------------------------------------------
# 3.8 Step 4: Flows / income variables - ratio-median imputation (leakage-aware)
# ------------------------------------------------------------------------
# Practical adjustments for this project:
#   (i)  Interest expense (xint) scales more naturally with debt than with␣
 ↪assets.
#        => prefer imputing xint via (xint / total_debt) when debt is available.
#   (ii) Tax components used in the FFO proxy (txt, txdc, txach) should not␣
 ↪block feature construction.
#        => set missing components to 0.0 (retain miss_* flags as indicators).
#   (iii) Ratio×size imputations can explode for very large firms.
#        => clip only-imputed values to a robust training-observed quantile␣
 ↪band.

FLOWS = [c for c in [
   ␣
 ↪"ib","ibadj","ibc","niadj","nopi","oibdp","dp","txt","oancf","fincf","ivncf","xint","esubc"
] if c in df.columns]

# --- Debt-aware base for xint (after stock imputations have populated dlc/
 ↪dltt) ---
if "xint" in df.columns and all(c in df.columns for c in ["dlc", "dltt"]):
```

```
    df["_td_for_xint"] = (
        pd.to_numeric(df["dlc"], errors="coerce").fillna(0.0)
        + pd.to_numeric(df["dltt"], errors="coerce").fillna(0.0)
    )
else:
    df["_td_for_xint"] = np.nan

# Debt-aware rule: if total debt == 0, interest expense is set to 0 (keep miss␣
 ↪flag already created)
if "xint" in df.columns and "_td_for_xint" in df.columns:
    xint_miss = df["xint"].isna()
    df.loc[xint_miss & (pd.to_numeric(df["_td_for_xint"], errors="coerce").
 ↪fillna(0.0) <= 0), "xint"] = 0.0

# Fit ratio medians on training and apply (xint uses debt base when possible;␣
 ↪otherwise assets base)
flow_fits = {}
for c in FLOWS:
    size_base = "_td_for_xint" if (c == "xint" and "_td_for_xint" in df.
 ↪columns) else "at"
    flow_fits[c] = _fit_ratio_medians(tr_all, c, size_col=size_base)

for c in FLOWS:
    size_base = "_td_for_xint" if (c == "xint" and "_td_for_xint" in df.
 ↪columns) else "at"
    # flows can be negative; do not clip to non-negative
    _apply_ratio_medians(df, c, flow_fits[c], size_col=size_base, nonneg=False)

# --- Practical fill: tax components used in FFO proxy ---
for c in ["txt", "txdc", "txach"]:
    if c in df.columns:
        df.loc[df[c].isna(), c] = 0.0

# --- Practical guardrail: cap only-imputed values to training-observed␣
 ↪quantiles ---
def _cap_imputed_to_train_quantiles(
    df_all: pd.DataFrame,
    col: str,
    lower_q: float = 0.01,
    upper_q: float = 0.99,
    nonneg: bool = False,
) -> None:
    flag = f"miss_{col}"
    if col not in df_all.columns or flag not in df_all.columns:
        return
```

```
    miss_mask = df_all[flag].astype(bool)
    if not miss_mask.any():
        return

    # Prefer bounds from observed (not originally missing) values in the␣
 ↪training split
    obs = pd.to_numeric(df_all.loc[train_mask_for_imputation & (~miss_mask),␣
 ↪col], errors="coerce")
    if obs.notna().sum() < 200:
        obs = pd.to_numeric(df_all.loc[train_mask_for_imputation, col],␣
 ↪errors="coerce")

    if obs.notna().sum() == 0:
        return

    lo = float(obs.quantile(lower_q))
    hi = float(obs.quantile(upper_q))

    if nonneg:
        lo = max(0.0, lo) if np.isfinite(lo) else 0.0

    s = pd.to_numeric(df_all.loc[miss_mask, col], errors="coerce")
    df_all.loc[miss_mask, col] = s.clip(lo, hi).to_numpy()

# Apply capping to the variables that are later used (directly or indirectly)␣
 ↪in engineered ratios
CAP_COLS = sorted(set(STOCKS + FLOWS + ["dlcch", "apalch", "recch", "invch",␣
 ↪"chech"]) & set(df.columns))
for c in CAP_COLS:
    _cap_imputed_to_train_quantiles(df, c, lower_q=0.01, upper_q=0.99,␣
 ↪nonneg=(c in NONNEG_STOCKS))


# 3.9 EDA AFTER imputation (missingness reduction + distribution deltas)
# ----------------------------------------------------------------------------
df_raw_post = df[raw].copy(deep=True)

post_miss = pd.DataFrame(
    {
        "col": raw,
        "n_na_post": [int(df_raw_post[c].isna().sum()) for c in raw],
        "pct_na_post": [float(df_raw_post[c].isna().mean() * 100.0) for c in␣
 ↪raw],
        "train_pct_na_post": [
            float(df_raw_post.loc[train_mask_for_imputation, c].isna().mean() *␣
 ↪100.0) for c in raw
```

```
        ],
    }
)

changes = pre_miss.merge(post_miss, on="col", how="left")
changes["n_imputed"] = changes["n_na_pre"] - changes["n_na_post"]
changes["pct_points_na_reduction"] = changes["pct_na_pre"] -␣
 ↪changes["pct_na_post"]

x_post = df_raw_post[raw].apply(pd.to_numeric, errors="coerce").replace([np.
 ↪inf, -np.inf], np.nan)
q_post = x_post.quantile([0.01, 0.05, 0.25, 0.50, 0.75, 0.95, 0.99]).T
post_dist = pd.DataFrame(
    {
        "n_nonmiss_post": x_post.notna().sum(),
        "mean_post": x_post.mean(),
        "std_post": x_post.std(ddof=0),
        "min_post": x_post.min(),
        "p01_post": q_post[0.01],
        "p05_post": q_post[0.05],
        "p25_post": q_post[0.25],
        "p50_post": q_post[0.50],
        "p75_post": q_post[0.75],
        "p95_post": q_post[0.95],
        "p99_post": q_post[0.99],
        "max_post": x_post.max(),
    }
)

pre_dist_key =␣
 ↪pre_dist[["n_nonmiss_pre","mean_pre","std_pre","p01_pre","p50_pre","p99_pre"]].
 ↪copy() if raw else pd.DataFrame()
post_dist_key =␣
 ↪post_dist[["n_nonmiss_post","mean_post","std_post","p01_post","p50_post","p99_post"]].
 ↪copy() if raw else pd.DataFrame()

dist_delta = pre_dist_key.join(post_dist_key, how="outer")
dist_delta["delta_mean"] = dist_delta["mean_post"] - dist_delta["mean_pre"]
dist_delta["delta_std"]  = dist_delta["std_post"]  - dist_delta["std_pre"]
dist_delta["delta_p50"]  = dist_delta["p50_post"]  - dist_delta["p50_pre"]

# Imputed-only diagnostics
rows = []
for c in raw:
    pre_na_mask = df_raw_pre[c].isna()
    n_imp = int(pre_na_mask.sum())
    if n_imp == 0:
```

```
        rows.append((c, 0, np.nan, np.nan, np.nan, np.nan, np.nan))
        continue

    imp_vals = pd.to_numeric(df_raw_post.loc[pre_na_mask, c], errors="coerce").
 ↪replace([np.inf, -np.inf], np.nan)
    obs_vals = pd.to_numeric(df_raw_pre.loc[~pre_na_mask, c], errors="coerce").
 ↪replace([np.inf, -np.inf], np.nan)

    rows.append(
        (
            c,
            n_imp,
            float(imp_vals.mean()) if imp_vals.notna().any() else np.nan,
            float(imp_vals.median()) if imp_vals.notna().any() else np.nan,
            float(imp_vals.std(ddof=0)) if imp_vals.notna().any() else np.nan,
            float(obs_vals.mean()) if obs_vals.notna().any() else np.nan,
            float(obs_vals.median()) if obs_vals.notna().any() else np.nan,
        )
    )

imputed_only = pd.DataFrame(
    rows,
   ␣
 ↪columns=["col","n_imputed","imputed_mean","imputed_median","imputed_std","observed_mean_pre
).set_index("col")

print("\n=== EDA (AFTER imputation): Missingness on raw inputs + change ===")
cols_show = [
    "col", "n", "n_na_pre", "pct_na_pre", "n_na_post", "pct_na_post",
    "n_imputed", "pct_points_na_reduction", "train_pct_na_pre",␣
 ↪"train_pct_na_post",
]
print(
    changes[cols_show]
    .sort_values(["n_imputed","pct_points_na_reduction"], ascending=[False,␣
 ↪False])
    .round(4)
    .head(50)
)

print("\n=== Change analysis: Distribution deltas (post - pre) on raw inputs␣
 ↪===")
print(
   ␣
 ↪dist_delta[["n_nonmiss_pre","n_nonmiss_post","delta_mean","delta_std","delta_p50"]]
    .sort_values("delta_mean", key=lambda s: s.abs(), ascending=False)
```

```
    .round(6)
    .head(50)
)

print("\n=== Change analysis: Imputed-only vs observed (pre) summary ===")
print(
    imputed_only.assign(
        mean_gap_imputed_minus_observed=lambda d: d["imputed_mean"] -␣
  ↪d["observed_mean_pre"],
        median_gap_imputed_minus_observed=lambda d: d["imputed_median"] -␣
  ↪d["observed_median_pre"],
    )
    .sort_values("n_imputed", ascending=False)
    .round(6)
    .head(50)
)
```

```
=== EDA (BEFORE imputation): Missingness on raw inputs ===
       col      n  n_na_pre  pct_na_pre  train_n  train_pct_na_pre
18   dlcch  75005     33143     44.1877    48458           42.9630
5   apalch  75005     30371     40.4920    48458           39.0214
75   txach  75005     22791     30.3860    48458           29.2501
48  ivstch  75005     19194     25.5903    48458           23.0282
66   recch  75005     12589     16.7842    48458           16.5938
53  mkvalt  75005     12350     16.4656    48458           17.0189
71    sppe  75005     12239     16.3176    48458           16.4307
1      act  75005     10721     14.2937    48458           14.6581
50     lct  75005     10695     14.2590    48458           14.5982
84    xint  75005     10536     14.0471    48458           13.9234
78  txditc  75005      9069     12.0912    48458           12.2044
79     txp  75005      7942     10.5886    48458           10.6917
29   esubc  75005      6798      9.0634    48458            8.8964
49     lco  75005      6779      9.0381    48458            9.3277
0      aco  75005      6778      9.0367    48458            9.3277
60  prcc_f  75005      5956      7.9408    48458            8.8902
59  prcc_c  75005      5942      7.9221    48458            8.8819
40   invch  75005      5315      7.0862    48458            7.3445
72   sppiv  75005      3967      5.2890    48458            4.9631
44   ivaeq  75005      3548      4.7304    48458            4.8826
61  prstkc  75005      3410      4.5464    48458            4.9486
8     caps  75005      3074      4.0984    48458            4.2655
6      aqc  75005      2812      3.7491    48458            3.2647
23      dp  75005      2722      3.6291    48458            3.7909
65      re  75005      2596      3.4611    48458            3.6650
46    ivch  75005      2540      3.3864    48458            3.3844
45    ivao  75005      2529      3.3718    48458            3.1821
```

```
77    txdc  75005      2472      3.2958    48458           3.4050
69     siv  75005      2299      3.0651    48458           3.1285
57   oibdp  75005      2142      2.8558    48458           2.9366
14    cstk  75005      1925      2.5665    48458           3.0294
24     dpc  75005      1757      2.3425    48458           2.4268
19   dltis  75005      1756      2.3412    48458           2.4681
58   ppent  75005      1723      2.2972    48458           2.3814
73    sstk  75005      1436      1.9145    48458           1.8593
20    dltr  75005      1388      1.8505    48458           2.0616
25      dv  75005      1137      1.5159    48458           1.5911
70     spi  75005      1126      1.5012    48458           1.3785
67    rect  75005       880      1.1733    48458           1.1247
52    mibt  75005       842      1.1226    48458           1.3022
13    csho  75005       804      1.0719    48458           1.2939
28     dvt  75005       706      0.9413    48458           1.1412
26     dvc  75005       703      0.9373    48458           1.1350
39   intan  75005       654      0.8719    48458           0.8977
41    invt  75005       652      0.8693    48458           0.8255
76  txbcof  75005       619      0.8253    48458           1.0071
9     capx  75005       503      0.6706    48458           0.7429
83   xidoc  75005       477      0.6360    48458           0.7202
3   aoloch  75005       462      0.6160    48458           0.6893
38     ibc  75005       454      0.6053    48458           0.6810

=== EDA (BEFORE imputation): Distribution summary (raw inputs) ===
         n_nonmiss_pre      mean_pre       std_pre       min_pre  \
consol               0           NaN           NaN           NaN
datafmt              0           NaN           NaN           NaN
conm                 0           NaN           NaN           NaN
indfmt               0           NaN           NaN           NaN
dlcch            41862  2.168121e+03  1.155250e+06 -8.555368e+07
apalch           44634  8.385212e+03  3.253932e+05 -4.849894e+07
txach            52214  3.214308e+02  3.140000e+04 -3.282293e+06
ivstch           55811  6.871673e+04  3.980372e+06 -1.752327e+07
recch            62416 -1.318528e+04  3.129434e+05 -2.297888e+07
mkvalt           62655  9.288907e+05  2.201298e+07  0.000000e+00
sppe             62766  6.939724e+03  1.245321e+05 -5.895100e+04
act              64284  6.440956e+05  4.645197e+06 -2.498000e+03
lct              64310  4.755088e+05  3.980895e+06  0.000000e+00
xint             64469  2.759126e+04  1.196406e+05 -7.284730e+05
txditc           65936  4.384545e+04  3.265589e+05 -1.616000e+03
txp              67063  1.062620e+04  1.001099e+05 -2.217000e+03
esubc            68207 -2.220041e+03  8.665624e+04 -8.775761e+06
lco              68226  2.039805e+05  1.784063e+06 -8.900000e+01
aco              68227  5.573244e+04  9.308496e+05 -2.500000e+01
prcc_f           69029  6.838158e+02  1.018087e+04  1.000000e-04
prcc_c           69039  7.067634e+02  1.003861e+04  1.000000e-04
invch            69690 -3.860008e+04  1.008997e+06 -8.472324e+07
```

```
sppiv            71038 -7.943608e+03  1.753101e+05 -1.598761e+07
ivaeq            71457  6.873078e+04  9.162568e+05 -1.551500e+04
prstkc           71595  2.902337e+04  2.408411e+05 -1.131100e+04
caps             71931  5.224117e+05  2.924028e+06 -2.333792e+07
aqc              72193  2.583727e+04  3.427732e+05 -2.002407e+07
dp               72283  7.989028e+04  7.010127e+05 -4.121000e+03
re               72409  4.063079e+05  4.773704e+06 -8.108442e+07
ivch             72465  4.336588e+05  1.267386e+07  0.000000e+00
ivao             72476  1.117387e+06  1.830844e+07  0.000000e+00
txdc             72533 -2.552908e+03  8.582746e+04 -8.127405e+06
siv              72706  3.204806e+05  9.796796e+06 -3.080000e+02
oibdp            72863  2.472644e+05  1.847387e+06 -6.234977e+06
cstk             73080  1.440901e+05  1.035999e+06 -1.466100e+04
dpc              73248  8.009984e+04  6.495898e+05 -1.551370e+05
dltis            73249  2.584506e+05  2.896443e+06 -4.257400e+04
ppent            73282  5.870571e+05  5.242606e+06  0.000000e+00
sstk             73569  2.949253e+04  2.324514e+05 -5.012830e+05
dltr             73617  2.679871e+05  2.856270e+06 -2.844300e+04
dv               73868  4.155940e+04  3.023818e+05 -1.631969e+06
spi              73879 -1.290533e+04  1.992509e+05 -1.354613e+07
rect             74125  2.121967e+06  3.060638e+07 -7.300000e+01
mibt             74163  5.164076e+04  6.023643e+05 -1.237174e+06
csho             74201  2.254364e+05  5.059555e+06  0.000000e+00
dvt              74299  4.154428e+04  3.087856e+05 -1.329470e+05
dvc              74302  4.081004e+04  3.071635e+05 -2.473000e+03
intan            74351  3.830857e+05  2.603599e+06  0.000000e+00
invt             74353  1.471455e+05  1.356403e+06  0.000000e+00
txbcof           74386  2.081901e+02  4.374733e+03 -9.510600e+04

              p01_pre      p05_pre     p25_pre     p50_pre      p75_pre  \
consol            NaN          NaN         NaN         NaN          NaN
datafmt           NaN          NaN         NaN         NaN          NaN
conm              NaN          NaN         NaN         NaN          NaN
indfmt            NaN          NaN         NaN         NaN          NaN
dlcch   -2.265422e+05   -8701.6500      0.0000      0.0000       1.0000
apalch  -1.053346e+05  -14648.8000   -119.0000     50.0000    1693.0000
txach   -9.192830e+03    -193.0000      0.0000      0.0000       0.0000
ivstch  -2.099582e+05  -10799.5000      0.0000      0.0000       0.0000
recch   -2.890472e+05  -52494.2500  -2252.2500    -17.0000      54.0000
mkvalt   5.154000e-01       3.3549     48.9974    467.1767    4760.4262
sppe     0.000000e+00       0.0000      0.0000      0.0000       6.8300
act      1.700000e-01      21.0000   1732.0000  23074.0000  214833.2500
lct      6.600000e-01      26.5090   1277.0000  11405.0000  105630.0000
xint     0.000000e+00       0.0000     15.0000    433.0000    8992.0000
txditc   0.000000e+00       0.0000      0.0000      0.0000    1883.5000
txp      0.000000e+00       0.0000      0.0000      0.0000     152.0000
esubc   -3.866582e+04    -484.7000      0.0000      0.0000       0.0000
lco      0.000000e+00       0.0000    301.0000   3986.5000   46136.5000
```

aco	0.000000e+00	0.0000	58.0000	946.0000	10293.0000
prcc_f	1.000000e-02	0.1016	2.6000	13.9200	42.8400
prcc_c	1.000000e-02	0.1000	2.6200	14.0100	43.0250
invch	-3.362948e+05	-33105.1500	-172.0000	0.0000	0.0000
sppiv	-1.356993e+05	-11708.4500	-13.0000	0.0000	0.0000
ivaeq	0.000000e+00	0.0000	0.0000	0.0000	0.0000
prstkc	0.000000e+00	0.0000	0.0000	0.0000	846.5000
caps	0.000000e+00	0.0000	1645.0000	26381.0000	268051.0000
aqc	-1.760872e+04	0.0000	0.0000	0.0000	0.0000
dp	0.000000e+00	0.0000	68.0000	1079.4700	15628.5000
re	-2.484404e+06	-701966.6000	-80817.0000	-1251.0000	29686.0000
ivch	0.000000e+00	0.0000	0.0000	0.0000	275.0000
ivao	0.000000e+00	0.0000	0.0000	0.0000	3979.7500
txdc	-1.337068e+05	-18598.4000	-105.0000	0.0000	5.0000
siv	0.000000e+00	0.0000	0.0000	0.0000	61.0000
oibdp	-2.060061e+05	-47857.8000	-1873.5000	800.4000	46909.0000
cstk	0.000000e+00	0.0000	8.0000	197.0000	13957.2500
dpc	0.000000e+00	0.0000	100.3000	1542.0000	21382.2500
dltis	0.000000e+00	0.0000	0.0000	0.1500	1740.0000
ppent	0.000000e+00	0.0000	514.0000	8837.5000	93080.2500
sstk	0.000000e+00	0.0000	0.0000	58.0000	4216.0000
dltr	0.000000e+00	0.0000	0.0000	140.7000	8902.0000
dv	0.000000e+00	0.0000	0.0000	0.0000	884.0000
spi	-3.215187e+05	-47237.4000	-1071.0000	0.0000	0.0000
rect	0.000000e+00	0.0000	225.0000	5841.0000	97807.0000
mibt	-1.887380e+03	0.0000	0.0000	0.0000	0.0000
csho	3.150000e+00	28.0000	7219.0000	36145.0000	107343.0000
dvt	0.000000e+00	0.0000	0.0000	0.0000	1178.4450
dvc	0.000000e+00	0.0000	0.0000	0.0000	621.0000
intan	0.000000e+00	0.0000	0.0000	1365.0000	45975.0000
invt	0.000000e+00	0.0000	0.0000	275.3000	12041.0000
txbcof	0.000000e+00	0.0000	0.0000	0.0000	0.0000

	p95_pre	p99_pre	max_pre
consol	NaN	NaN	NaN
datafmt	NaN	NaN	NaN
conm	NaN	NaN	NaN
indfmt	NaN	NaN	NaN
dlcch	12288.900	239569.72	1.674825e+08
apalch	40626.050	280635.99	2.085900e+07
txach	544.000	16622.89	3.282293e+06
ivstch	14654.500	206862.80	3.964813e+08
recch	14636.250	102245.15	3.777953e+07
mkvalt	1137140.389	16710068.76	3.522211e+09
sppe	7315.750	96130.00	7.819213e+06
act	1887389.100	11141665.87	2.648894e+08
lct	1150175.850	8749902.35	2.275619e+08
xint	126670.400	475515.16	3.895731e+06

```
txditc    138738.750    824518.10  1.309682e+07
txp        20801.100    229018.76  5.843978e+06
esubc        503.700     18641.34  3.314734e+06
lco       528718.500   3510482.25  1.976466e+08
aco       124386.000    926127.42  1.973027e+08
prcc_f       271.900     17260.04  1.369125e+06
prcc_c       275.000     17524.72  1.369125e+06
invch       9558.700     99906.23  1.919409e+07
sppiv        779.000     12781.32  7.378669e+06
ivaeq      69436.200   1094948.44  4.184002e+07
prstkc     87371.300    608747.74  1.551860e+07
caps     2034165.500   7334828.10  1.724641e+08
aqc        71387.800    578083.72  5.188218e+07
dp        220915.500   1361138.02  3.714574e+07
re       1402781.800  11353646.12  2.219454e+08
ivch      441795.800   3486163.72  1.169666e+09
ivao     1193829.500  10658327.50  1.099816e+09
txdc       11204.800     85023.32  4.507005e+06
siv       237122.750   2260226.70  9.068858e+08
oibdp     727543.600   4683708.38  7.350011e+07
cstk      420270.900   2924964.10  3.904769e+07
dpc       252347.000   1229121.52  3.648387e+07
dltis     692136.800   4321096.08  1.818726e+08
ppent    1817488.900  10065753.57  2.926841e+08
sstk      111376.400    525775.68  2.036114e+07
dltr      761854.400   3911400.36  1.796217e+08
dv        122377.700    899427.56  1.296791e+07
spi         2226.100     54763.82  1.148318e+07
rect     1917288.600  17121476.16  1.218880e+09
mibt       66241.300   1067080.28  3.075527e+07
csho      533438.000   2530459.00  6.064077e+08
dvt       124549.500    879155.68  1.485872e+07
dvc       120572.700    869661.95  1.437446e+07
intan    1345701.500   7366060.00  9.849622e+07
invt      385661.800   2910600.48  1.043358e+08
txbcof         0.000      3680.50  6.321680e+05

=== EDA (AFTER imputation): Missingness on raw inputs + change ===
       col      n  n_na_pre  pct_na_pre  n_na_post  pct_na_post  n_imputed  \
0    dlcch  75005     33143     44.1877       4502       6.0023      28641
1   apalch  75005     30371     40.4920       4391       5.8543      25980
2    txach  75005     22791     30.3860          0       0.0000      22791
5   mkvalt  75005     12350     16.4656          0       0.0000      12350
7      act  75005     10721     14.2937          0       0.0000      10721
8      lct  75005     10695     14.2590          0       0.0000      10695
9     xint  75005     10536     14.0471          0       0.0000      10536
4    recch  75005     12589     16.7842       2473       3.2971      10116
12   esubc  75005      6798      9.0634          0       0.0000       6798
```

13	lco	75005	6779	9.0381	0	0.0000	6779
14	aco	75005	6778	9.0367	0	0.0000	6778
17	invch	75005	5315	7.0862	1241	1.6546	4074
19	ivaeq	75005	3548	4.7304	0	0.0000	3548
21	caps	75005	3074	4.0984	0	0.0000	3074
23	dp	75005	2722	3.6291	0	0.0000	2722
24	re	75005	2596	3.4611	0	0.0000	2596
27	txdc	75005	2472	3.2958	0	0.0000	2472
29	oibdp	75005	2142	2.8558	0	0.0000	2142
30	cstk	75005	1925	2.5665	0	0.0000	1925
33	ppent	75005	1723	2.2972	0	0.0000	1723
38	rect	75005	880	1.1733	0	0.0000	880
39	mibt	75005	842	1.1226	0	0.0000	842
40	csho	75005	804	1.0719	0	0.0000	804
43	intan	75005	654	0.8719	0	0.0000	654
44	invt	75005	652	0.8693	0	0.0000	652
49	ibc	75005	454	0.6053	0	0.0000	454
54	ivncf	75005	316	0.4213	0	0.0000	316
55	fincf	75005	315	0.4200	0	0.0000	315
57	oancf	75005	303	0.4040	0	0.0000	303
56	chech	75005	304	0.4053	57	0.0760	247
58	pstk	75005	219	0.2920	0	0.0000	219
59	pstkn	75005	211	0.2813	0	0.0000	211
60	dltt	75005	188	0.2506	0	0.0000	188
61	tstk	75005	180	0.2400	0	0.0000	180
62	ceq	75005	149	0.1987	0	0.0000	149
63	lt	75005	101	0.1347	0	0.0000	101
64	ap	75005	88	0.1173	0	0.0000	88
65	pstkr	75005	75	0.1000	0	0.0000	75
67	dlc	75005	31	0.0413	0	0.0000	31
68	txt	75005	15	0.0200	0	0.0000	15
69	nopi	75005	12	0.0160	0	0.0000	12
70	che	75005	6	0.0080	0	0.0000	6
71	ao	75005	5	0.0067	0	0.0000	5
74	seq	75005	3	0.0040	0	0.0000	3
3	ivstch	75005	19194	25.5903	19194	25.5903	0
6	sppe	75005	12239	16.3176	12239	16.3176	0
10	txditc	75005	9069	12.0912	9069	12.0912	0
11	txp	75005	7942	10.5886	7942	10.5886	0
15	prcc_f	75005	5956	7.9408	5956	7.9408	0
16	prcc_c	75005	5942	7.9221	5942	7.9221	0

	pct_points_na_reduction	train_pct_na_pre	train_pct_na_post
0	38.1855	42.9630	8.1844
1	34.6377	39.0214	7.8645
2	30.3860	29.2501	0.0000
5	16.4656	17.0189	0.0000
7	14.2937	14.6581	0.0000

8	14.2590	14.5982	0.0000
9	14.0471	13.9234	0.0000
4	13.4871	16.5938	4.0303
12	9.0634	8.8964	0.0000
13	9.0381	9.3277	0.0000
14	9.0367	9.3277	0.0000
17	5.4316	7.3445	2.0265
19	4.7304	4.8826	0.0000
21	4.0984	4.2655	0.0000
23	3.6291	3.7909	0.0000
24	3.4611	3.6650	0.0000
27	3.2958	3.4050	0.0000
29	2.8558	2.9366	0.0000
30	2.5665	3.0294	0.0000
33	2.2972	2.3814	0.0000
38	1.1733	1.1247	0.0000
39	1.1226	1.3022	0.0000
40	1.0719	1.2939	0.0000
43	0.8719	0.8977	0.0000
44	0.8693	0.8255	0.0000
49	0.6053	0.6810	0.0000
54	0.4213	0.4788	0.0000
55	0.4200	0.4767	0.0000
57	0.4040	0.4602	0.0000
56	0.3293	0.4623	0.1156
58	0.2920	0.2951	0.0000
59	0.2813	0.2868	0.0000
60	0.2506	0.2889	0.0000
61	0.2400	0.2414	0.0000
62	0.1987	0.2043	0.0000
63	0.1347	0.1259	0.0000
64	0.1173	0.0991	0.0000
65	0.1000	0.1032	0.0000
67	0.0413	0.0495	0.0000
68	0.0200	0.0144	0.0000
69	0.0160	0.0227	0.0000
70	0.0080	0.0124	0.0000
71	0.0067	0.0103	0.0000
74	0.0040	0.0062	0.0000
3	0.0000	23.0282	23.0282
6	0.0000	16.4307	16.4307
10	0.0000	12.2044	12.2044
11	0.0000	10.6917	10.6917
15	0.0000	8.8902	8.8902
16	0.0000	8.8819	8.8819

```
=== Change analysis: Distribution deltas (post - pre) on raw inputs ===
          n_nonmiss_pre  n_nonmiss_post     delta_mean     delta_std  \
```

mkvalt	62655	75005	342407.151082	-1.766379e+06
act	64284	75005	185811.385530	-1.788945e+05
lct	64310	75005	90560.417655	-2.078001e+05
lco	68226	75005	25133.438729	-6.005804e+04
rect	74125	75005	-23898.875943	-1.792580e+05
re	72409	75005	-16713.635107	-8.185532e+04
lt	74904	75005	-6139.421095	-4.632179e+04
aco	68227	75005	5641.288782	-4.024887e+04
apalch	44634	70614	4579.546469	-6.258219e+04
oibdp	72863	75005	-3941.409192	-2.590904e+04
xint	64469	75005	-3829.352862	-8.317088e+03
caps	71931	75005	-3196.489599	-5.034086e+04
ivaeq	71457	75005	-3014.123461	-2.174282e+04
cstk	73080	75005	-3011.166145	-1.261688e+04
intan	74351	75005	-2919.453815	-1.103587e+04
recch	62416	72532	2604.026060	-2.102156e+04
ap	74917	75005	-2504.505248	-2.223392e+04
dltt	74817	75005	-2289.510691	-1.061091e+04
ppent	73282	75005	-2131.803875	-5.917365e+04
ceq	74856	75005	2099.646389	-2.799357e+03
dp	72283	75005	2094.861834	-1.117139e+04
invch	69690	73764	2027.820782	-2.817826e+04
oancf	74702	75005	-758.223668	-4.390364e+03
mibt	74163	75005	-579.344475	-3.365905e+03
ivncf	74689	75005	571.186625	-3.279110e+03
ibc	74551	75005	-487.204713	-2.187075e+03
csho	74201	75005	-333.528043	-2.694192e+04
dlcch	41862	70503	291.940128	-2.640138e+05
dlc	74974	75005	257.646866	-1.500792e+03
esubc	68207	75005	201.211062	-4.017797e+03
invt	74353	75005	177.616992	-4.859402e+03
chech	74701	74948	-148.009337	-3.349173e+03
tstk	74825	75005	-135.653129	-5.866449e+02
txach	52214	75005	-97.669890	-5.200987e+03
txdc	72533	75005	84.138237	-1.424961e+03
che	74999	75005	-70.437906	-6.643609e+02
fincf	74690	75005	-40.341627	-3.252400e+03
ao	75000	75005	-37.926984	-4.493027e+02
seq	75002	75005	-37.282301	-1.214421e+02
pstk	74786	75005	-20.953261	-3.301992e+02
pstkr	74930	75005	-7.086945	-8.506999e+01
nopi	74993	75005	6.593073	-6.715890e+01
txt	74990	75005	-5.734069	-2.316013e+01
pstkn	74794	75005	-4.220299	-2.150862e+02
aoloch	74543	74543	0.000000	0.000000e+00
dltr	73617	73617	0.000000	0.000000e+00
dltis	73249	73249	0.000000	0.000000e+00
cstke	75001	75001	0.000000	0.000000e+00

datadate	75005	75005	0.000000	0.000000e+00
at	75005	75005	0.000000	0.000000e+00

	delta_p50
mkvalt	137.2548
act	15675.0000
lct	7207.0000
lco	1865.5000
rect	-189.0000
re	-300.9000
lt	46.0000
aco	461.0000
apalch	16.0000
oibdp	182.6000
xint	-68.5000
caps	-443.0000
ivaeq	0.0000
cstk	5.0000
intan	1.6000
recch	9.0000
ap	1.0000
dltt	-21.0000
ppent	519.5000
ceq	27.5000
dp	141.5300
invch	0.0000
oancf	-3.0000
mibt	0.0000
ivncf	6.0000
ibc	-0.2400
csho	-186.0000
dlcch	0.0000
dlc	3.0000
esubc	0.0000
invt	3.4000
chech	-0.0900
tstk	0.0000
txach	0.0000
txdc	0.0000
che	-2.0000
fincf	0.1000
ao	-0.1000
seq	-1.5000
pstk	0.0000
pstkr	0.0000
nopi	0.0000
txt	0.0000
pstkn	0.0000

```
aoloch        0.0000
dltr          0.0000
dltis         0.0000
cstke         0.0000
datadate      0.0000
at            0.0000

=== Change analysis: Imputed-only vs observed (pre) summary ===
        n_imputed  imputed_mean  imputed_median   imputed_std  \
col
dlcch       33143  2.886764e+03        7.000000  6.776735e+04
apalch      30371  2.083248e+04      144.000000  7.569476e+04
txach       22791  0.000000e+00        0.000000  0.000000e+00
ivstch      19194           NaN             NaN           NaN
recch       12589  5.485655e+03      151.400000  8.040616e+04
mkvalt      12350  3.008425e+06    11457.197806  5.254540e+06
sppe        12239           NaN             NaN           NaN
act         10721  1.944047e+06   620683.448289  2.953969e+06
lct         10695  1.110617e+06   285127.913921  2.049787e+06
xint        10536  3.303843e+02      364.500000  1.277657e+02
txditc       9069           NaN             NaN           NaN
txp          7942           NaN             NaN           NaN
esubc        6798  0.000000e+00        0.000000  0.000000e+00
lco          6779  4.820648e+05   107133.519909  8.839821e+05
aco          6778  1.181587e+05    24795.542510  2.273255e+05
prcc_f       5956           NaN             NaN           NaN
prcc_c       5942           NaN             NaN           NaN
invch        5315 -1.884277e+03        0.000000  4.005203e+04
sppiv        3967           NaN             NaN           NaN
ivaeq        3548  5.011973e+03        0.000000  5.785298e+04
prstkc       3410           NaN             NaN           NaN
caps         3074  4.444179e+05    16696.000000  1.192783e+06
aqc          2812           NaN             NaN           NaN
dp           2722  1.376144e+05    53599.841951  2.450769e+05
re           2596 -7.659123e+04    -9263.000000  4.207461e+05
ivch         2540           NaN             NaN           NaN
ivao         2529           NaN             NaN           NaN
txdc         2472  0.000000e+00        0.000000  0.000000e+00
siv          2299           NaN             NaN           NaN
oibdp        2142  1.092507e+05    47840.710176  2.565068e+05
cstk         1925  2.676416e+04      428.596440  2.179249e+05
dpc          1757           NaN             NaN           NaN
dltis        1756           NaN             NaN           NaN
ppent        1723  4.942562e+05   237832.184614  7.872769e+05
sstk         1436           NaN             NaN           NaN
dltr         1388           NaN             NaN           NaN
dv           1137           NaN             NaN           NaN
spi          1126           NaN             NaN           NaN
```

```
rect          880  8.499478e+04      488.469851  3.713500e+05
mibt          842  3.299715e+01        0.000000  7.844159e+02
csho          804  1.943216e+05    11007.377133  4.821008e+05
dvt           706           NaN             NaN           NaN
dvc           703           NaN             NaN           NaN
intan         654  4.826358e+04     1666.201656  3.016515e+05
invt          652  1.675783e+05     1370.317109  5.706362e+05
txbcof        619           NaN             NaN           NaN
capx          503           NaN             NaN           NaN
xidoc         477           NaN             NaN           NaN
aoloch        462           NaN             NaN           NaN
ibc           454 -3.087908e+03        0.651928  3.557588e+04

        observed_mean_pre  observed_median_pre  \
col
dlcch        2.168121e+03               0.0000
apalch       8.385212e+03              50.0000
txach        3.214308e+02               0.0000
ivstch       6.871673e+04               0.0000
recch       -1.318528e+04             -17.0000
mkvalt       9.288907e+05             467.1767
sppe         6.939724e+03               0.0000
act          6.440956e+05           23074.0000
lct          4.755088e+05           11405.0000
xint         2.759126e+04             433.0000
txditc       4.384545e+04               0.0000
txp          1.062620e+04               0.0000
esubc       -2.220041e+03               0.0000
lco          2.039805e+05            3986.5000
aco          5.573244e+04             946.0000
prcc_f       6.838158e+02              13.9200
prcc_c       7.067634e+02              14.0100
invch       -3.860008e+04               0.0000
sppiv       -7.943608e+03               0.0000
ivaeq        6.873078e+04               0.0000
prstkc       2.902337e+04               0.0000
caps         5.224117e+05           26381.0000
aqc          2.583727e+04               0.0000
dp           7.989028e+04            1079.4700
re           4.063079e+05           -1251.0000
ivch         4.336588e+05               0.0000
ivao         1.117387e+06               0.0000
txdc        -2.552908e+03               0.0000
siv          3.204806e+05               0.0000
oibdp        2.472644e+05             800.4000
cstk         1.440901e+05             197.0000
dpc          8.009984e+04            1542.0000
dltis        2.584506e+05               0.1500
```

ppent	5.870571e+05	8837.5000
sstk	2.949253e+04	58.0000
dltr	2.679871e+05	140.7000
dv	4.155940e+04	0.0000
spi	-1.290533e+04	0.0000
rect	2.121967e+06	5841.0000
mibt	5.164076e+04	0.0000
csho	2.254364e+05	36145.0000
dvt	4.154428e+04	0.0000
dvc	4.081004e+04	0.0000
intan	3.830857e+05	1365.0000
invt	1.471455e+05	275.3000
txbcof	2.081901e+02	0.0000
capx	8.862010e+04	887.0000
xidoc	8.325909e+02	0.0000
aoloch	3.689360e+04	-0.0100
ibc	7.740282e+04	2.0000

col	mean_gap_imputed_minus_observed	median_gap_imputed_minus_observed
dlcch	7.186430e+02	7.000000
apalch	1.244727e+04	94.000000
txach	-3.214308e+02	0.000000
ivstch	NaN	NaN
recch	1.867094e+04	168.400000
mkvalt	2.079534e+06	10990.021106
sppe	NaN	NaN
act	1.299952e+06	597609.448289
lct	6.351084e+05	273722.913921
xint	-2.726088e+04	-68.500000
txditc	NaN	NaN
txp	NaN	NaN
esubc	2.220041e+03	0.000000
lco	2.780843e+05	103147.019909
aco	6.242621e+04	23849.542510
prcc_f	NaN	NaN
prcc_c	NaN	NaN
invch	3.671580e+04	0.000000
sppiv	NaN	NaN
ivaeq	-6.371881e+04	0.000000
prstkc	NaN	NaN
caps	-7.799372e+04	-9685.000000
aqc	NaN	NaN
dp	5.772414e+04	52520.371951
re	-4.828992e+05	-8012.000000
ivch	NaN	NaN
ivao	NaN	NaN
txdc	2.552908e+03	0.000000

siv	NaN	NaN
oibdp	-1.380137e+05	47040.310176
cstk	-1.173260e+05	231.596440
dpc	NaN	NaN
dltis	NaN	NaN
ppent	-9.280090e+04	228994.684614
sstk	NaN	NaN
dltr	NaN	NaN
dv	NaN	NaN
spi	NaN	NaN
rect	-2.036972e+06	-5352.530149
mibt	-5.160776e+04	0.000000
csho	-3.111476e+04	-25137.622867
dvt	NaN	NaN
dvc	NaN	NaN
intan	-3.348221e+05	301.201656
invt	2.043276e+04	1095.017109
txbcof	NaN	NaN
capx	NaN	NaN
xidoc	NaN	NaN
aoloch	NaN	NaN
ibc	-8.049073e+04	-1.348072

[63]:
```
# ============================================================================
# Feature Engineering & Target Construction
#   - Build leverage / coverage / cash-flow-to-debt ratios commonly used in␣
 ↪credit analysis
#   - Define a 'highly leveraged' distress proxy from multiple ratio-based␣
 ↪conditions
#   - Create the supervised-learning target: next-year distress within the same␣
 ↪firm
# ============================================================================
dlc = pd.to_numeric(df.get("dlc", np.nan), errors="coerce")
dltt = pd.to_numeric(df.get("dltt", np.nan), errors="coerce")
df["total_debt"] = pd.concat([dlc, dltt], axis=1).sum(axis=1, min_count=1)

seq = pd.to_numeric(df.get("seq", np.nan), errors="coerce")
mibt = pd.to_numeric(df.get("mibt", 0.0), errors="coerce")
df["equity_plus_mi_sp"] = seq + mibt
df["total_capital_sp"] = df["total_debt"] + df["equity_plus_mi_sp"]
df["sp_debt_to_capital"] = safe_divide(df["total_debt"], df["total_capital_sp"])

oibdp = pd.to_numeric(df.get("oibdp", np.nan), errors="coerce")
xint = pd.to_numeric(df.get("xint", np.nan), errors="coerce")
df["sp_debt_to_ebitda"] = safe_divide(df["total_debt"], oibdp)

txt = pd.to_numeric(df.get("txt", 0.0), errors="coerce").fillna(0.0)
```

```
txdc = pd.to_numeric(df.get("txdc", 0.0), errors="coerce").fillna(0.0)
txach = pd.to_numeric(df.get("txach", 0.0), errors="coerce").fillna(0.0)
df["cash_tax_paid_proxy"] = txt - txdc - txach

df["ffo_proxy"] = oibdp - xint - pd.to_numeric(df["cash_tax_paid_proxy"],␣
 ↪errors="coerce")
df["sp_ffo_to_debt"] = safe_divide(df["ffo_proxy"], df["total_debt"])

oancf = pd.to_numeric(df.get("oancf", np.nan), errors="coerce")
capx = pd.to_numeric(df.get("capx", np.nan), errors="coerce")
df["sp_cfo_to_debt"] = safe_divide(oancf, df["total_debt"])
df["focf"] = oancf - capx
df["sp_focf_to_debt"] = safe_divide(df["focf"], df["total_debt"])

dv = pd.to_numeric(df.get("dv", 0.0), errors="coerce").fillna(0.0)
prstkc = pd.to_numeric(df.get("prstkc", 0.0), errors="coerce").fillna(0.0)
df["dcf"] = df["focf"] - dv - prstkc
df["sp_dcf_to_debt"] = safe_divide(df["dcf"], df["total_debt"])

# Log transforms (log1p handles 0 nicely). Negative → NaN.
for c in ["at", "mkvalt"]:
    if c in df.columns:
        s = pd.to_numeric(df[c], errors="coerce")
        df[f"log_{c}"] = np.where(s >= 0, np.log1p(s), np.nan)

# Interest coverage: EBITDA / |interest expense|
EPS_COV = 1e-6
df["sp_interest_coverage"] = safe_divide(oibdp, xint.abs() + EPS_COV)

# Distress proxy: 'highly leveraged' condition using multiple credit-ratio␣
 ↪thresholds
td = pd.to_numeric(df["total_debt"], errors="coerce").to_numpy(dtype=float)
cap = pd.to_numeric(df["total_capital_sp"], errors="coerce").
 ↪to_numpy(dtype=float)
eb = pd.to_numeric(oibdp, errors="coerce").to_numpy(dtype=float)
ffo = pd.to_numeric(df["ffo_proxy"], errors="coerce").to_numpy(dtype=float)

ffo_to_debt_pct = 100.0 * safe_divide(ffo, td)
debt_to_capital_pct = 100.0 * safe_divide(td, cap)
debt_to_ebitda = safe_divide(td, eb)

# Three "highly leveraged" conditions (S&P table)
hl_ffo = (td > 0) & (ffo_to_debt_pct < 15.0)          # FFO/total debt < 15%
hl_cap = (cap > 0) & (debt_to_capital_pct > 55.0)     # Total debt/total␣
 ↪capital > 55%
hl_deb = (td > 0) & ( (debt_to_ebitda > 4.5))  # TD/EBITDA > 4.5
```

```
is_highly_leveraged = hl_ffo & hl_cap & hl_deb

# Equity strictly negative and not missing
is_equity_negative = seq.notna() & (seq < -1)

# Final distress rule: highly leveraged (ratio-based proxy; excludes equity<0␣
 ↪flag computed below)
df["distress_dummy"] = ( is_highly_leveraged).astype("int8")

# Target: next year's distress (within firm)
df["target_next_year_distress"] = (
    df.groupby("firm_id")["distress_dummy"].shift(-1)
)
df = df.dropna(subset=["target_next_year_distress"]).reset_index(drop=True)
df["target_next_year_distress"] = df["target_next_year_distress"].astype("int8")
```

[64]:
```
# ============================================================================
# Train / Validation / Test Split (out-of-time)
#   - Split by label_year to respect the t → t+1 prediction structure
#   - Keep the last label year(s) inside the training pool as validation
# ============================================================================
train_pool = df[df["label_year"] <= TRAIN_CUTOFF_LABEL_YEAR].copy()
test = df[df["label_year"] > TRAIN_CUTOFF_LABEL_YEAR].copy()

years = np.sort(train_pool["label_year"].dropna().unique())
val_years = years[-VAL_YEARS:] if len(years) else np.array([], dtype=int)

val = train_pool[train_pool["label_year"].isin(val_years)].copy()
train = train_pool[~train_pool["label_year"].isin(val_years)].copy()

print(
    "Split:",
    f"train={len(train):,}",
    f"val={len(val):,}",
    f"test={len(test):,}",
    "| val_years:",
    list(val_years),
)
```

```
Split: train=44,783 val=6,415 test=12,404 | val_years: [np.int64(2022)]
```

[65]:
```
# ============================================================================
# Modeling-Ready Preprocessing (fit on TRAIN only)
#   - Handle infinities and remaining NaNs
#   - Winsorize features using TRAIN quantile bounds to reduce outlier leverage
#   - Standardize to z-scores using TRAIN mean/variance for comparable scaling
# ============================================================================
```

```
base_feats = [
    "sp_debt_to_capital",
    "sp_ffo_to_debt",
    "sp_cfo_to_debt",
    "sp_focf_to_debt",
    "sp_dcf_to_debt",
    "sp_debt_to_ebitda",
    "sp_interest_coverage",
    "log_at",
    "log_mkvalt",
]
feats = [c for c in base_feats if c in train.columns and c in val.columns and c␣
 ↪in test.columns]

# Replace +/-inf with NaN
for d in (train, val, test):
    d[feats] = d[feats].replace([np.inf, -np.inf], np.nan)

# Remaining NaNs: median imputation fit on TRAIN only (after ratio construction)
fill = train[feats].median(numeric_only=True)
for d in (train, val, test):
    d[feats] = d[feats].fillna(fill)

# Winsorize each feature using train-only quantiles
bounds = {}
for c in feats:
    s = pd.to_numeric(train[c], errors="coerce")
    bounds[c] = (s.quantile(WINSOR_LOWER_Q), s.quantile(WINSOR_UPPER_Q))

for d in (train, val, test):
    for c, (lo, hi) in bounds.items():
        s = pd.to_numeric(d[c], errors="coerce")
        d[c] = s.clip(lo, hi)

# Standardize (z-scores), fit on train only
x_train = train[feats].to_numpy(dtype=float)
x_val = val[feats].to_numpy(dtype=float)
x_test = test[feats].to_numpy(dtype=float)

scaler = StandardScaler().fit(x_train)
x_train_z = scaler.transform(x_train)
x_val_z = scaler.transform(x_val)
x_test_z = scaler.transform(x_test)

z_cols = [f"z_{c}" for c in feats]
train[z_cols] = x_train_z
val[z_cols] = x_val_z
```

```
test[z_cols] = x_test_z
```

[66]:
```
# ================================================================================
# Diagnostics & Monitoring Proxies
#   - Correlation screen (TRAIN) for rough signal strength and sanity checks
#   - Expanding-window time folds for temporal stability checks
#   - Dataset overview (rows/firms/years/target rate) + target rate by year
#   - Distribution summaries, collinearity scan, and simple drift proxy (SMD)␣
 ↪Train→Test
# ================================================================================
t = "target_next_year_distress"

corr = (
    train[[t] + feats]
    .corr(numeric_only=True)[t]
    .drop(t)
    .sort_values(key=np.abs, ascending=False)
)
print("Correlation with target:")
print(corr)

# Multicollinearity: Variance Inflation Factor (VIF)
def calculate_vif(df, features):
    vif_data = pd.DataFrame()
    vif_data["feature"] = features
    vif_data["VIF"] = [variance_inflation_factor(df[features].values, i) for i␣
 ↪in range(len(features))]
    return vif_data.sort_values("VIF", ascending=False)

vif_df = calculate_vif(train, z_cols)
print("\n=== Multicollinearity Diagnostic (VIF) ===")
print(vif_df)

folds = rolling_year_folds(train_pool, n_splits=N_SPLITS_TIME_CV,␣
 ↪min_train_years=3)
for i, (tr_idx, va_idx, tr_years, va_year) in enumerate(folds, 1):
    print(
        f"Fold {i}: train_years={tr_years[0]}..{tr_years[-1]}␣
 ↪(n={len(tr_idx)}), "
        f"val_year={va_year} (n={len(va_idx)})"
    )


def _overview(d: pd.DataFrame, name: str) -> None:
    n_rows = len(d)
    n_firms = d["firm_id"].nunique() if "firm_id" in d.columns else np.nan
    n_years = d["fyear"].nunique() if "fyear" in d.columns else np.nan
```

```
    target_rate = float(d[t].mean()) if t in d.columns else np.nan

    print(f"\n=== {name} === rows={n_rows:,} | firms={n_firms:,} |␣
 ↪years={n_years} | target_rate={target_rate:.4f}")

    if "label_year" in d.columns:
        by_year = d.groupby("label_year")[t].agg(["mean", "count"])
        print("\nTarget by label_year (tail):")
        print(by_year.tail(12))


_overview(train, "TRAIN")
_overview(val, "VAL")
_overview(test, "TEST")

post_miss = pd.DataFrame({"col": raw})
post_miss["train_pct_na"] = [train[c].isna().mean() * 100 if c in train.columns␣
 ↪else np.nan for c in raw]
post_miss["val_pct_na"]   = [val[c].isna().mean() * 100 if c in val.columns␣
 ↪else np.nan for c in raw]
post_miss["test_pct_na"]  = [test[c].isna().mean() * 100 if c in test.columns␣
 ↪else np.nan for c in raw]
if not post_miss.empty:
    post_miss = post_miss.sort_values("train_pct_na", ascending=False)
    print("\nPost-imputation missingness on raw inputs (pct):")
    print(post_miss.head(50).round(4))

def _dist(d: pd.DataFrame, cols: list[str], name: str) -> pd.DataFrame:
    x = d[cols].replace([np.inf, -np.inf], np.nan)
    q = x.quantile([0.01, 0.05, 0.25, 0.50, 0.75, 0.95, 0.99]).T
    out = pd.DataFrame(
        {
            "n": x.notna().sum(),
            "mean": x.mean(),
            "std": x.std(ddof=0),
            "min": x.min(),
            "p01": q[0.01],
            "p05": q[0.05],
            "p25": q[0.25],
            "p50": q[0.50],
            "p75": q[0.75],
            "p95": q[0.95],
            "p99": q[0.99],
            "max": x.max(),
            "skew": x.skew(numeric_only=True),
            "kurt": x.kurtosis(numeric_only=True),
        }
```

```
    )
    print(f"\nDistribution summary ({name})")
    print(out.round(4).sort_values("skew", key=lambda s: s.abs(),␣
 ↪ascending=False))
    return out


_ = _dist(train, feats, "TRAIN | winsorized raw feats")
_ = _dist(train, z_cols, "TRAIN | standardized feats")


def _hi_corr(d: pd.DataFrame, cols: list[str], thr: float = 0.80) ->␣
 ↪list[tuple[str, str, float]]:
    cm = d[cols].corr(numeric_only=True)
    pairs = []
    for i in range(len(cols)):
        for j in range(i + 1, len(cols)):
            r = cm.iloc[i, j]
            if np.isfinite(r) and abs(r) >= thr:
                pairs.append((cols[i], cols[j], float(r)))
    return sorted(pairs, key=lambda x: abs(x[2]), reverse=True)


pairs = _hi_corr(train, feats, thr=0.80)
print("\nHigh collinearity pairs among feats (|corr|>=0.80) [top 25]:")
for a, b, r in pairs[:25]:
    print(f"{a} vs {b}: r={r:.3f}")


def _drift_smd(a_df: pd.DataFrame, b_df: pd.DataFrame, cols: list[str]) -> pd.
 ↪DataFrame:
    rows = []
    for c in cols:
        a = pd.to_numeric(a_df[c], errors="coerce").replace([np.inf, -np.inf],␣
 ↪np.nan)
        b = pd.to_numeric(b_df[c], errors="coerce").replace([np.inf, -np.inf],␣
 ↪np.nan)

        ma, mb = float(a.mean()), float(b.mean())
        sa, sb = float(a.std(ddof=0)), float(b.std(ddof=0))
        sp = np.sqrt(0.5 * (sa ** 2 + sb ** 2))
        smd = (mb - ma) / sp if sp > 0 else np.nan

        rows.append((c, ma, mb, smd, abs(smd) if np.isfinite(smd) else np.nan))

        out = pd.DataFrame(rows, columns=["feature", "mean_train", "mean_test",␣
 ↪"smd", "abs_smd"])
```

```
    return out.sort_values("abs_smd", ascending=False)


drift = _drift_smd(train, test, feats)
print("\nTrain→Test drift (SMD) [top 15]:")
print(drift.head(15).round(4))


def _group_diff(d: pd.DataFrame, cols: list[str]) -> pd.Series:
    g = d.groupby(t)[cols].mean(numeric_only=True)
    if 0 in g.index and 1 in g.index:
        return (g.loc[1] - g.loc[0]).sort_values(key=np.abs, ascending=False)
    return pd.Series(dtype="float64")


diff = _group_diff(train, feats)
if not diff.empty:
    print("\nMean difference (target=1 minus target=0) on TRAIN feats [top 15]:
 ↪")
    print(diff.head(15).round(4))
```

```
Correlation with target:
sp_debt_to_capital      0.239281
log_at                  0.148383
sp_debt_to_ebitda       0.097794
log_mkvalt              0.094928
sp_dcf_to_debt          0.039173
sp_focf_to_debt         0.025729
sp_ffo_to_debt          0.021727
sp_interest_coverage    0.006064
sp_cfo_to_debt         -0.001887
Name: target_next_year_distress, dtype: float64

=== Multicollinearity Diagnostic (VIF) ===
                  feature        VIF
3       z_sp_focf_to_debt  10.773549
4        z_sp_dcf_to_debt   6.370928
2        z_sp_cfo_to_debt   3.514626
1        z_sp_ffo_to_debt   2.131804
7                z_log_at   1.281938
8            z_log_mkvalt   1.243078
0    z_sp_debt_to_capital   1.060240
5     z_sp_debt_to_ebitda   1.012524
6  z_sp_interest_coverage   1.005194
Fold 1: train_years=2015..2017 (n=19775), val_year=2018 (n=6337)
Fold 2: train_years=2015..2018 (n=26112), val_year=2019 (n=6173)
Fold 3: train_years=2015..2019 (n=32285), val_year=2020 (n=6233)
Fold 4: train_years=2015..2020 (n=38518), val_year=2021 (n=6265)
```

```
=== TRAIN === rows=44,783 | firms=9,220 | years=7 | target_rate=0.1076

Target by label_year (tail):
                mean  count
label_year
2015        0.104385   6773
2016        0.104566   6570
2017        0.100280   6432
2018        0.101310   6337
2019        0.124737   6173
2020        0.123376   6233
2021        0.095611   6265

=== VAL === rows=6,415 | firms=6,415 | years=1 | target_rate=0.0929

Target by label_year (tail):
                mean  count
label_year
2022        0.092907   6415

=== TEST === rows=12,404 | firms=6,633 | years=2 | target_rate=0.1051

Target by label_year (tail):
                mean  count
label_year
2023        0.105579   6327
2024        0.104657   6077

Post-imputation missingness on raw inputs (pct):
         col  train_pct_na  val_pct_na  test_pct_na
48    ivstch       23.4218     30.1949      30.3773
71      sppe       16.6559     16.3055      16.1641
78    txditc       12.4668     11.9719      12.1251
79       txp       10.9283     10.4910      10.7546
60    prcc_f        8.8516      6.1574       7.2718
59    prcc_c        8.8493      6.1107       7.2557
18     dlcch        8.2219      3.5230       2.0558
5     apalch        7.8802      3.1489       2.4105
61    prstkc        4.9997      4.4895       3.3538
72     sppiv        4.9997      5.5495       5.9174
66     recch        4.0283      2.2603       2.0880
46      ivch        3.4723      3.3359       3.5069
6        aqc        3.3227      4.6921       4.5953
45      ivao        3.2646      3.4451       3.7730
69       siv        3.1731      2.9774       2.9507
19     dltis        2.4585      2.3071       1.9832
24       dpc        2.4250      2.2136       2.1928
```

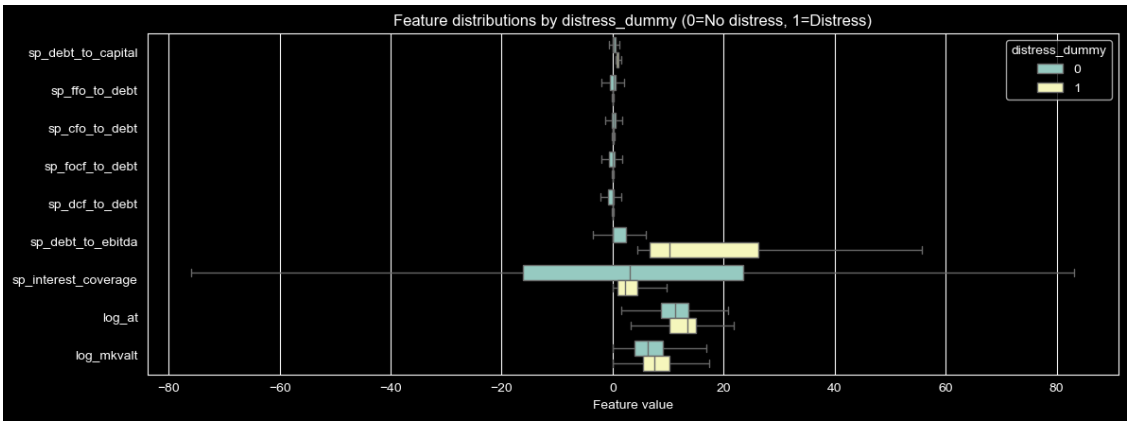
```
20      dltr        2.0276      1.2627       1.6608
40     invch        1.9985      1.0444       0.9755
73      sstk        1.8869      1.9174       2.0477
25        dv        1.5899      1.4030       1.3463
70       spi        1.4112      1.6680       1.8381
76    txbcof        1.0004      0.4677       0.4595
28       dvt        0.8374      0.5924       0.4837
26       dvc        0.8329      0.5924       0.4837
9       capx        0.7079      0.4365       0.5160
83     xidoc        0.6967      0.4677       0.4434
3     aoloch        0.6677      0.4677       0.4353
33      fopo        0.6319      0.4209       0.3950
30      exre        0.5940      0.3118       0.2902
31      fiao        0.4868      0.3118       0.2660
43     ivaco        0.4712      0.2962       0.2660
12     chech        0.1072      0.0156       0.0000
27       dvp        0.0536      0.0468       0.0484
15     cstke        0.0089      0.0000       0.0000
82      xido        0.0067      0.0000       0.0000
81        xi        0.0022      0.0000       0.0000
22        do        0.0022      0.0000       0.0000
8       caps        0.0000      0.0000       0.0000
7         at        0.0000      0.0000       0.0000
1        act        0.0000      0.0000       0.0000
23        dp        0.0000      0.0000       0.0000
21      dltt        0.0000      0.0000       0.0000
17       dlc        0.0000      0.0000       0.0000
10       ceq        0.0000      0.0000       0.0000
11       che        0.0000      0.0000       0.0000
13      csho        0.0000      0.0000       0.0000
14      cstk        0.0000      0.0000       0.0000
16  datadate        0.0000      0.0000       0.0000
4         ap        0.0000      0.0000       0.0000

Distribution summary (TRAIN | winsorized raw feats)
                         n          mean           std           min  \
sp_dcf_to_debt       44783 -2.502190e+01  1.984466e+02 -1.653794e+03
sp_debt_to_ebitda    44783  1.314641e+02  8.146559e+02 -8.128037e+02
sp_focf_to_debt      44783 -1.282550e+01  1.741842e+02 -1.360862e+03
sp_ffo_to_debt       44783 -1.100000e+01  2.290823e+02 -1.794695e+03
sp_interest_coverage 44783 -6.098563e+07  6.492082e+09 -3.280108e+10
sp_debt_to_capital   44783  3.126000e-01  5.945000e-01 -2.786200e+00
log_mkvalt           44783  6.999300e+00  4.093800e+00  3.656000e-01
log_at               44783  1.107300e+01  3.565000e+00  6.931000e-01
sp_cfo_to_debt       44783  4.686000e+00  1.540910e+02 -9.350585e+02

                              p01           p05     p25      p50      p75  \
sp_dcf_to_debt       -1.650683e+03 -3.161480e+01 -0.2674  -0.0163   0.0919
```

```
sp_debt_to_ebitda    -8.108731e+02 -6.313900e+00  0.0000   0.3204   3.6265
sp_focf_to_debt      -1.360587e+03 -2.001020e+01 -0.1546   0.0470   0.1921
sp_ffo_to_debt       -1.793826e+03 -1.929870e+01 -0.0832   0.1162   0.3252
sp_interest_coverage -3.278543e+10 -2.221000e+09 -7.5590   2.7042  21.0105
sp_debt_to_capital   -2.782800e+00 -1.000000e-04  0.0002   0.2456   0.5571
log_mkvalt            3.656000e-01  1.299500e+00  3.9637   6.3919   9.1257
log_at                6.931000e-01  4.369400e+00  8.8541  11.4487  13.7914
sp_cfo_to_debt       -9.345454e+02 -7.473200e+00  0.0001   0.1332   0.3434

                              p95           p99           max    skew     kurt
sp_dcf_to_debt             3.0767  3.510140e+02  3.512747e+02 -6.8466  51.0331
sp_debt_to_ebitda         39.9676  6.328203e+03  6.332345e+03  6.3041  41.3162
sp_focf_to_debt            6.6901  6.072581e+02  6.073161e+02 -5.1643  41.3521
sp_ffo_to_debt            14.5162  8.582689e+02  8.585733e+02 -4.8162  41.3640
sp_interest_coverage  37063.0105  4.293433e+10  4.300872e+10  1.7497  27.9326
sp_debt_to_capital         1.0013  2.785900e+00  2.787600e+00 -0.6996  10.4704
log_mkvalt                15.6653  1.645580e+01  1.645580e+01  0.6659  -0.2189
log_at                    16.0542  1.785350e+01  1.785370e+01 -0.6152   0.1416
sp_cfo_to_debt            17.0351  9.284942e+02  9.291911e+02  0.2574  28.3851

Distribution summary (TRAIN | standardized feats)
                           n  mean  std     min     p01     p05     p25  \
z_sp_dcf_to_debt       44783  -0.0  1.0 -8.2076 -8.1919 -0.0332  0.1247
z_sp_debt_to_ebitda    44783  -0.0  1.0 -1.1591 -1.1567 -0.1691 -0.1614
z_sp_focf_to_debt      44783  -0.0  1.0 -7.7391 -7.7376 -0.0412  0.0727
z_sp_ffo_to_debt       44783   0.0  1.0 -7.7863 -7.7825 -0.0362  0.0477
z_sp_interest_coverage 44783  -0.0  1.0 -5.0431 -5.0407 -0.3327  0.0094
z_sp_debt_to_capital   44783  -0.0  1.0 -5.2125 -5.2068 -0.5260 -0.5255
z_log_mkvalt           44783  -0.0  1.0 -1.6204 -1.6204 -1.3923 -0.7415
z_log_at               44783   0.0  1.0 -2.9116 -2.9116 -1.8804 -0.6224
z_sp_cfo_to_debt       44783  -0.0  1.0 -6.0986 -6.0953 -0.0789 -0.0304

                           p50     p75     p95     p99     max    skew  \
z_sp_dcf_to_debt        0.1260  0.1266  0.1416  1.8949  1.8962 -6.8466
z_sp_debt_to_ebitda    -0.1610 -0.1569 -0.1123  7.6066  7.6117  6.3041
z_sp_focf_to_debt       0.0739  0.0747  0.1120  3.5599  3.5603 -5.1643
z_sp_ffo_to_debt        0.0485  0.0494  0.1114  3.7946  3.7959 -4.8162
z_sp_interest_coverage  0.0094  0.0094  0.0094  6.6227  6.6342  1.7497
z_sp_debt_to_capital   -0.1127  0.4113  1.1584  4.1602  4.1631 -0.6996
z_log_mkvalt           -0.1484  0.5194  2.1168  2.3100  2.3100  0.6659
z_log_at                0.1054  0.7625  1.3973  1.9020  1.9020 -0.6152
z_sp_cfo_to_debt       -0.0295 -0.0282  0.0801  5.9952  5.9997  0.2574

                          kurt
z_sp_dcf_to_debt       51.0331
z_sp_debt_to_ebitda    41.3162
z_sp_focf_to_debt      41.3521
z_sp_ffo_to_debt       41.3640
```

```
z_sp_interest_coverage  27.9326
z_sp_debt_to_capital    10.4704
z_log_mkvalt            -0.2189
z_log_at                 0.1416
z_sp_cfo_to_debt        28.3851

High collinearity pairs among feats (|corr|>=0.80) [top 25]:
sp_focf_to_debt vs sp_dcf_to_debt: r=0.897

Train→Test drift (SMD) [top 15]:
                feature    mean_train     mean_test     smd  abs_smd
6  sp_interest_coverage -6.098563e+07 -7.379267e+08 -0.1040   0.1040
2        sp_cfo_to_debt  4.686000e+00 -7.884400e+00 -0.0902   0.0902
7                log_at  1.107300e+01  1.136440e+01  0.0836   0.0836
0    sp_debt_to_capital  3.126000e-01  3.574000e-01  0.0771   0.0771
8             log_mkvalt  6.999300e+00  7.247000e+00  0.0606   0.0606
1        sp_ffo_to_debt -1.100000e+01 -1.968880e+01 -0.0407   0.0407
3       sp_focf_to_debt -1.282550e+01 -1.753580e+01 -0.0287   0.0287
4        sp_dcf_to_debt -2.502190e+01 -2.412000e+01  0.0048   0.0048
5     sp_debt_to_ebitda  1.314641e+02  1.284517e+02 -0.0037   0.0037

Mean difference (target=1 minus target=0) on TRAIN feats [top 15]:
sp_interest_coverage    1.270463e+08
sp_debt_to_ebitda       2.570901e+02
sp_dcf_to_debt          2.508610e+01
sp_ffo_to_debt          1.606200e+01
sp_focf_to_debt         1.446220e+01
log_at                  1.707100e+00
log_mkvalt              1.254100e+00
sp_cfo_to_debt         -9.383000e-01
sp_debt_to_capital      4.591000e-01
dtype: float64
```

[67]:
```python
# ============================================================================
# Visual EDA: Feature distributions by distress flag
#   - Quick separation check: do distressed vs non-distressed firms differ in levels?
#   - Uses a horizontal boxplot for comparability across features
# ============================================================================

import matplotlib.pyplot as plt
import seaborn as sns

# Objective: visualize feature distribution differences for distressed vs non-distressed observations.

plot_df = train_pool.copy() if "train_pool" in globals() else df.copy()
```

```
flag_col = "distress_dummy" if "distress_dummy" in plot_df.columns else (
    "target_next_year_distress" if "target_next_year_distress" in plot_df.
 ↪columns else None
)
if flag_col is None:
    raise KeyError("No distress flag found. Expected 'distress_dummy' or␣
 ↪'target_next_year_distress' in the data.")

plot_feats = [c for c in (feats if "feats" in globals() else []) if c in␣
 ↪plot_df.columns]
if not plot_feats:
    plot_feats = [c for c in (z_cols if "z_cols" in globals() else []) if c in␣
 ↪plot_df.columns]
if not plot_feats:
    raise KeyError(
        "No feature columns found to plot. Expected 'feats' or 'z_cols' to␣
 ↪exist and be present in the data.")

tmp = plot_df[[flag_col] + plot_feats].copy()
tmp[flag_col] = pd.to_numeric(tmp[flag_col], errors="coerce").astype("Int64")
tmp = tmp[tmp[flag_col].isin([0, 1])].copy()

long = tmp.melt(id_vars=[flag_col], value_vars=plot_feats, var_name="feature",␣
 ↪value_name="value")
long["value"] = pd.to_numeric(long["value"], errors="coerce")
long = long.dropna(subset=["value"])

fig, ax = plt.subplots(figsize=(12, max(4.5, 0.45 * len(plot_feats))))
sns.boxplot(
    data=long,
    x="value",
    y="feature",
    hue=flag_col,
    orient="h",
    showfliers=False,
    ax=ax,
)
ax.set_title(f"Feature distributions by {flag_col} (0=No distress, 1=Distress)")
ax.set_xlabel("Feature value")
ax.set_ylabel("")
ax.legend(title=flag_col, loc="best")
plt.tight_layout()
plt.show()
```



[68]:
```
# ================================================================================
# Sanity checks on the distress proxy
#   - Component prevalence (each condition and joint condition)
#   - Distress rate by firm size decile (log assets) to check monotonic patterns
# ================================================================================

print(pd.Series({
    "hl_ffo": hl_ffo.mean(),
    "hl_cap": hl_cap.mean(),
    "hl_deb": hl_deb.mean(),
    "hl_all": is_highly_leveraged.mean()
}))
df["size_decile"] = pd.qcut(df["log_at"], 10, duplicates="drop")
print(df.groupby("size_decile")["distress_dummy"].mean())
```

```
hl_ffo    0.481235
hl_cap    0.264142
hl_deb    0.204013
hl_all    0.103446
dtype: float64
size_decile
(-0.001, 6.585]     0.029241
(6.585, 8.38]       0.093553
(8.38, 9.509]       0.094782
(9.509, 10.52]      0.057408
(10.52, 11.507]     0.053459
(11.507, 12.554]    0.073585
(12.554, 13.434]    0.098270
(13.434, 14.245]    0.143396
(14.245, 15.252]    0.182704
(15.252, 21.886]    0.208458
Name: distress_dummy, dtype: float64
```

```
/var/folders/p1/_cwwbdbj51q1lwpynfnzdxpm0000gn/T/ipykernel_2951/2342790164.py:14
: FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  print(df.groupby("size_decile")["distress_dummy"].mean())
```

0.0.1 5.5 Endogeneity & Simultaneity Bias: A Critical Note

As highlighted by recent literature (e.g., Roberts 2015, Jiang 2017), modeling capital structure as a predictor of credit outcomes is prone to **endogeneity**.

1. **Simultaneity**: Firms choose their leverage (Debt/Capital, etc.) based on their expected future credit risk and investment opportunities. Thus, leverage and distress outcomes are determined simultaneously in equilibrium.
2. **Reverse Causality**: While high leverage may lead to distress, the *anticipation* of distress may also force firms to seek more debt (if they follow pecking order) or prevent them from accessing debt (if credit constrained).

Mitigation in this notebook: - While we use a standard Logistic Regression for transparency and baseline benchmarking, we acknowledge that the coefficients should be interpreted as **predictive associations** rather than pure causal effects. - Future iterations should explore **Instrumental Variables (IV)** or **GMM** estimation (e.g., using industry-median leverage or historical tax changes as instruments) to better identify the causal impact of leverage on distress.

[69]:
```
# ================================================================================
# Persistence Benchmark (Economic Sanity Check)
#   - Does the ML model add value over simply assuming distress persists?
#   - We use 'distress_dummy' (current year) to predict␣
 ↪'target_next_year_distress'
# ================================================================================
from sklearn.metrics import f1_score, accuracy_score

for name, d in [("VAL", val), ("TEST", test)]:
    y_true = d[TARGET_COL].astype(int)
    y_bench = d["distress_dummy"].astype(int)

    auc_bench = roc_auc_score(y_true, y_bench)
    f1_bench = f1_score(y_true, y_bench)
    acc_bench = accuracy_score(y_true, y_bench)

    print(f"--- Persistence Benchmark ({name}) ---")
    print(f"AUC: {auc_bench:.4f} | F1: {f1_bench:.4f} | Accuracy: {acc_bench:.
 ↪4f}")
```

```
--- Persistence Benchmark (VAL) ---
AUC: 0.7680 | F1: 0.5798 | Accuracy: 0.9221
--- Persistence Benchmark (TEST) ---
AUC: 0.7715 | F1: 0.6036 | Accuracy: 0.9193
```

0.1 6. Modeling: Baseline out-of-sample supervised learning

This section trains a simple, interpretable classifier on the **TRAIN** split, selects a small hyperparameter setting using the **VAL** split, and reports final performance on the **TEST** split (held out, label_year > train cutoff).

Goal: Predict `target_next_year_distress` one year ahead from standardized financial-ratio features.

[70]:
```
# ============================================================================
# 6.1 Setup: Features, target, and train/val/test matrices
# ============================================================================
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    roc_auc_score,
    average_precision_score,
    brier_score_loss,
    confusion_matrix,
    classification_report,
    precision_recall_curve,
    roc_curve,
)
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from IPython.display import display

TARGET_COL = "target_next_year_distress"

# Prefer standardized feature columns created earlier (fit on TRAIN only)
if "z_cols" in globals():
    MODEL_FEATS = list(z_cols)
else:
    MODEL_FEATS = [f"z_{c}" for c in feats]  # fallback

# Basic sanity checks
for df_name, df_ in [("train", train), ("val", val), ("test", test)]:
    missing_feats = [c for c in MODEL_FEATS if c not in df_.columns]
    if missing_feats:
        raise KeyError(f"{df_name}: missing feature columns: {missing_feats}")

X_train = train[MODEL_FEATS].to_numpy(dtype=float)
y_train = train[TARGET_COL].astype(int).to_numpy()

X_val = val[MODEL_FEATS].to_numpy(dtype=float)
y_val = val[TARGET_COL].astype(int).to_numpy()

X_test = test[MODEL_FEATS].to_numpy(dtype=float)
```

```
y_test = test[TARGET_COL].astype(int).to_numpy()

# Defensive: ensure model inputs are finite
def _assert_finite(name, X):
    bad = ~np.isfinite(X)
    if bad.any():
        rows, cols = np.where(bad)
        raise ValueError(f"{name}: found non-finite values at {len(rows)} cells␣
 ↪(e.g., row={rows[0]}, col={MODEL_FEATS[cols[0]]}).")

_assert_finite("X_train", X_train)
_assert_finite("X_val", X_val)
_assert_finite("X_test", X_test)

print("Modeling matrix shapes:")
print(f"  X_train: {X_train.shape} | y_train mean={y_train.mean():.4f}")
print(f"  X_val:   {X_val.shape} | y_val mean={y_val.mean():.4f}")
print(f"  X_test:  {X_test.shape} | y_test mean={y_test.mean():.4f}")
```

```
Modeling matrix shapes:
  X_train: (44783, 9) | y_train mean=0.1076
  X_val:   (6415, 9) | y_val mean=0.0929
  X_test:  (12404, 9) | y_test mean=0.1051
```

[71]:
```
# ============================================================================
# 6.2 Baseline model: Logistic Regression (tuned on VAL)
#   - Interpretable, strong baseline for tabular finance ratios
#   - We tune C on VAL; you can expand the grid later if needed
# ============================================================================
C_GRID = [0.01, 0.1, 1.0, 10.0]

best = {"C": None, "val_auc": -np.inf, "model": None}

for C in C_GRID:
    clf = LogisticRegression(
        C=C,
        solver="lbfgs",
        max_iter=500,
        class_weight="balanced",  # practical default with ~10% positives
        n_jobs=None,
        random_state=42,
    )
    clf.fit(X_train, y_train)
    val_proba = clf.predict_proba(X_val)[:, 1]
    val_auc = roc_auc_score(y_val, val_proba)

    if val_auc > best["val_auc"]:
```

```
        best.update({"C": C, "val_auc": val_auc, "model": clf})

print(f"Best LogisticRegression on VAL: C={best['C']} | AUC={best['val_auc']:.
 ↪4f}")
logit = best["model"]
```

```
Best LogisticRegression on VAL: C=10.0 | AUC=0.8022
```

[72]:
```
# ============================================================================
# 6.2b Statistical Inference & Economic Interpretation (Statsmodels)
#   - sklearn is great for prediction but lacks p-values and inference.
#   - We re-estimate the logit using statsmodels to audit statistical␣
 ↪significance.
#   - We also report Marginal Effects to see economic significance.
# ============================================================================
import statsmodels.api as sm

# Add constant for intercept
X_train_sm = sm.add_constant(X_train)
sm_model = sm.Logit(y_train, X_train_sm).fit(disp=0)

print("\n=== Statsmodels Logistic Regression Summary ===")
print(sm_model.summary(xname=["const"] + MODEL_FEATS))

# Marginal Effects at the Mean (MEM)
try:
    mfx = sm_model.get_margeff(at="mean")
    print("\n=== Economic Significance: Marginal Effects at the Mean ===")
    print(mfx.summary())
except Exception as e:
    print(f"\nCould not calculate marginal effects: {e}")

# AIC / BIC / Pseudo-R2
print(f"\nModel Fit:")
print(f"  Pseudo R2 (McFadden): {sm_model.prsquared:.4f}")
print(f"  AIC: {sm_model.aic:.2f}")
print(f"  BIC: {sm_model.bic:.2f}")
```

```
=== Statsmodels Logistic Regression Summary ===
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                44783
Model:                          Logit   Df Residuals:                    44773
Method:                           MLE   Df Model:                            9
Date:                Sat, 27 Dec 2025   Pseudo R-squ.:                  0.1287
Time:                        21:58:01   Log-Likelihood:                -13324.
converged:                       True   LL-Null:                       -15293.
```

```
Covariance Type:            nonrobust   LLR p-value:                    0.000
==============================================================================
==========
                              coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
----------
const                     -2.5290      0.021   -118.866      0.000      -2.571
-2.487
z_sp_debt_to_capital       0.8344      0.017     48.294      0.000       0.800
0.868
z_sp_ffo_to_debt           0.1321      0.054      2.447      0.014       0.026
0.238
z_sp_cfo_to_debt          -0.2272      0.074     -3.060      0.002      -0.373
-0.082
z_sp_focf_to_debt         -0.1136      0.150     -0.758      0.448      -0.408
0.180
z_sp_dcf_to_debt           0.3571      0.120      2.969      0.003       0.121
0.593
z_sp_debt_to_ebitda        0.1376      0.011     12.428      0.000       0.116
0.159
z_sp_interest_coverage    -0.0062      0.020     -0.312      0.755      -0.045
0.033
z_log_at                   0.5240      0.021     25.229      0.000       0.483
0.565
z_log_mkvalt               0.1008      0.018      5.750      0.000       0.066
0.135
==============================================================================
==========

=== Economic Significance: Marginal Effects at the Mean ===
        Logit Marginal Effects
=====================================
Dep. Variable:                      y
Method:                          dydx
At:                              mean
==============================================================================
                dy/dx    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1              0.0571      0.001     47.717      0.000       0.055       0.059
x2              0.0090      0.004      2.462      0.014       0.002       0.016
x3             -0.0155      0.005     -3.077      0.002      -0.025      -0.006
x4             -0.0078      0.010     -0.758      0.448      -0.028       0.012
x5              0.0244      0.008      2.989      0.003       0.008       0.040
x6              0.0094      0.001     12.295      0.000       0.008       0.011
x7             -0.0004      0.001     -0.312      0.755      -0.003       0.002
x8              0.0358      0.001     26.763      0.000       0.033       0.038
x9              0.0069      0.001      5.762      0.000       0.005       0.009
```

```
================================================================================

Model Fit:
  Pseudo R2 (McFadden): 0.1287
  AIC: 26668.28
  BIC: 26755.37
```

[76]:
```
# ================================================================================
# 6.2c Temporal Stability (Walk-Forward Validation)
#   - We assess whether model performance is stable across different time␣
 ↪regimes.
#   - Using the expanding-window folds defined in diagnostics.
# ================================================================================

# Ensure MODEL_FEATS exist in train_pool (train_pool may not have z_* columns)
_missing = [c for c in MODEL_FEATS if c not in train_pool.columns]
if _missing:
    # Recreate the exact same preprocessing pipeline used for train/val/test,␣
 ↪but applied to train_pool
    for c in feats:
        train_pool[c] = pd.to_numeric(train_pool[c], errors="coerce").
 ↪replace([np.inf, -np.inf], np.nan)
    train_pool[feats] = train_pool[feats].fillna(fill)
    for c, (lo, hi) in bounds.items():
        train_pool[c] = pd.to_numeric(train_pool[c], errors="coerce").clip(lo,␣
 ↪hi)

    x_pool = train_pool[feats].to_numpy(dtype=float)
    x_pool_z = scaler.transform(x_pool)
    train_pool[z_cols] = x_pool_z

temporal_results = []
for i, (tr_idx, va_idx, tr_years, va_year) in enumerate(folds, 1):
    # Prepare fold data (fold indices are label-based, so use .loc not .iloc)
    X_tr = train_pool.loc[tr_idx, MODEL_FEATS].to_numpy(dtype=float)
    y_tr = train_pool.loc[tr_idx, TARGET_COL].astype(int).to_numpy()
    X_va = train_pool.loc[va_idx, MODEL_FEATS].to_numpy(dtype=float)
    y_va = train_pool.loc[va_idx, TARGET_COL].astype(int).to_numpy()

    # Fit model (using best C from tuning)
    fold_clf = LogisticRegression(C=best["C"], solver="lbfgs", max_iter=500,␣
 ↪class_weight="balanced", random_state=42)
    fold_clf.fit(X_tr, y_tr)

    # Evaluate
    probs = fold_clf.predict_proba(X_va)[:, 1]
    auc = roc_auc_score(y_va, probs)
```

```
    ap = average_precision_score(y_va, probs)

    temporal_results.append({
        "Fold": i,
        "Val_Year": va_year,
        "Train_End": tr_years[-1],
        "AUC": auc,
        "PR-AUC": ap
    })

temporal_df = pd.DataFrame(temporal_results)
print("\n=== Temporal Stability: Walk-Forward Results ===")
print(temporal_df.round(4))
if not temporal_df.empty:
    print(f"\nAverage AUC across folds: {temporal_df['AUC'].mean():.4f}")
```

```
=== Temporal Stability: Walk-Forward Results ===
   Fold  Val_Year  Train_End     AUC  PR-AUC
0     1      2018       2017  0.8096  0.2702
1     2      2019       2018  0.7981  0.3023
2     3      2020       2019  0.7985  0.2938
3     4      2021       2020  0.8050  0.2414

Average AUC across folds: 0.8028
```

[77]:
```
# ============================================================================
# 6.3 Evaluation: VAL and TEST (AUC, PR-AUC, Brier) + thresholding
# ============================================================================
def evaluate_split(name, y_true, y_proba, threshold=0.5):
    auc = roc_auc_score(y_true, y_proba)
    ap = average_precision_score(y_true, y_proba)
    brier = brier_score_loss(y_true, y_proba)

    y_pred = (y_proba >= threshold).astype(int)
    cm = confusion_matrix(y_true, y_pred)

    print(f"\n=== {name} ===")
    print(f"AUC (ROC): {auc:.4f}")
    print(f"Average Precision (PR-AUC): {ap:.4f}")
    print(f"Brier score (calibration): {brier:.4f}")
    print(f"Threshold: {threshold:.3f}")
    print("Confusion matrix [ [TN FP] [FN TP] ]:")
    print(cm)
    print("\nClassification report:")
    print(classification_report(y_true, y_pred, digits=4))
```

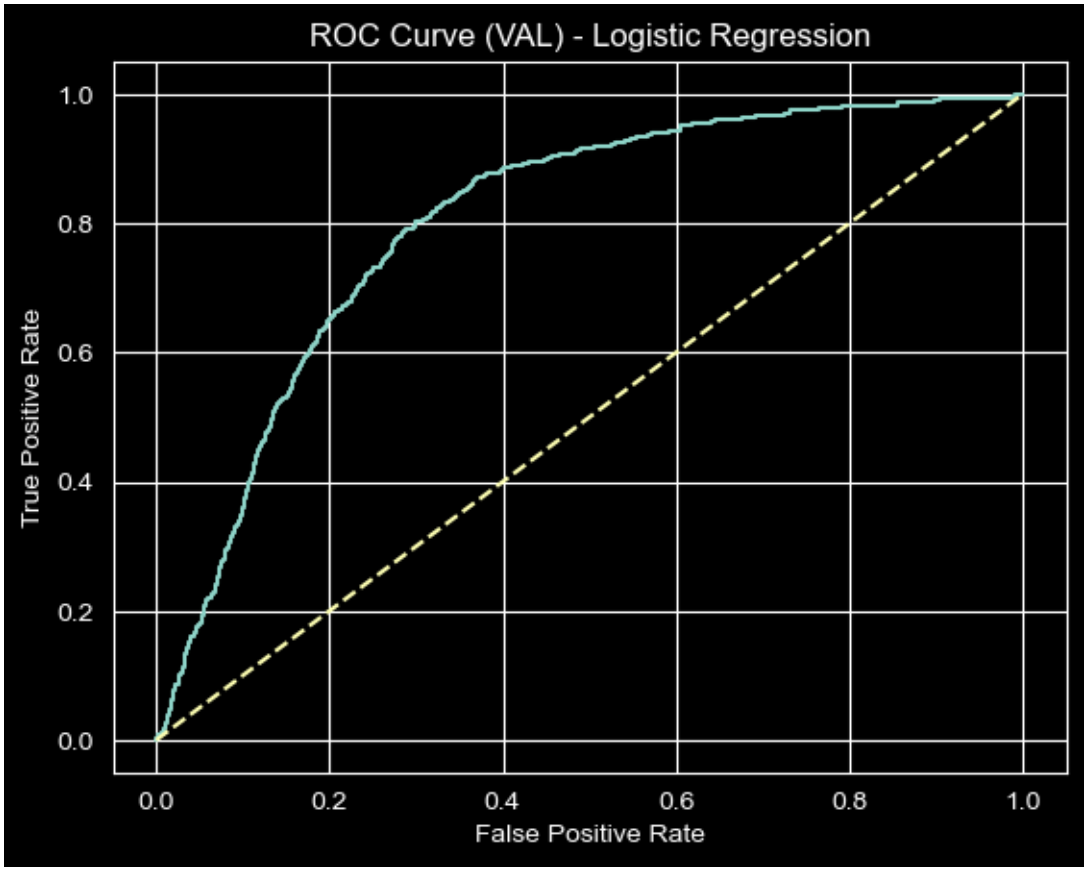
```
    # Explicitly pull out Sensitivity and Specificity
    # TN FP
    # FN TP
    tn, fp, fn, tp = cm.ravel()
    sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
    specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
    print(f"Sensitivity (Recall of Distress): {sensitivity:.4f}")
    print(f"Specificity (Recall of Non-Distress): {specificity:.4f}")

    return {"auc": auc, "ap": ap, "brier": brier, "cm": cm, "sens":␣
 ↪sensitivity, "spec": specificity}

val_proba = logit.predict_proba(X_val)[:, 1]
test_proba = logit.predict_proba(X_test)[:, 1]

# Choose an operational threshold using VAL (maximize F1 as a simple practical␣
 ↪rule)
prec, rec, thr = precision_recall_curve(y_val, val_proba)
f1 = 2 * (prec * rec) / (prec + rec + 1e-12)
best_idx = np.nanargmax(f1[1:])  # skip first point (threshold undefined)
best_thr = thr[best_idx]

_ = evaluate_split("VAL (threshold from VAL-F1)", y_val, val_proba,␣
 ↪threshold=float(best_thr))
_ = evaluate_split("TEST (same threshold)", y_test, test_proba,␣
 ↪threshold=float(best_thr))
```

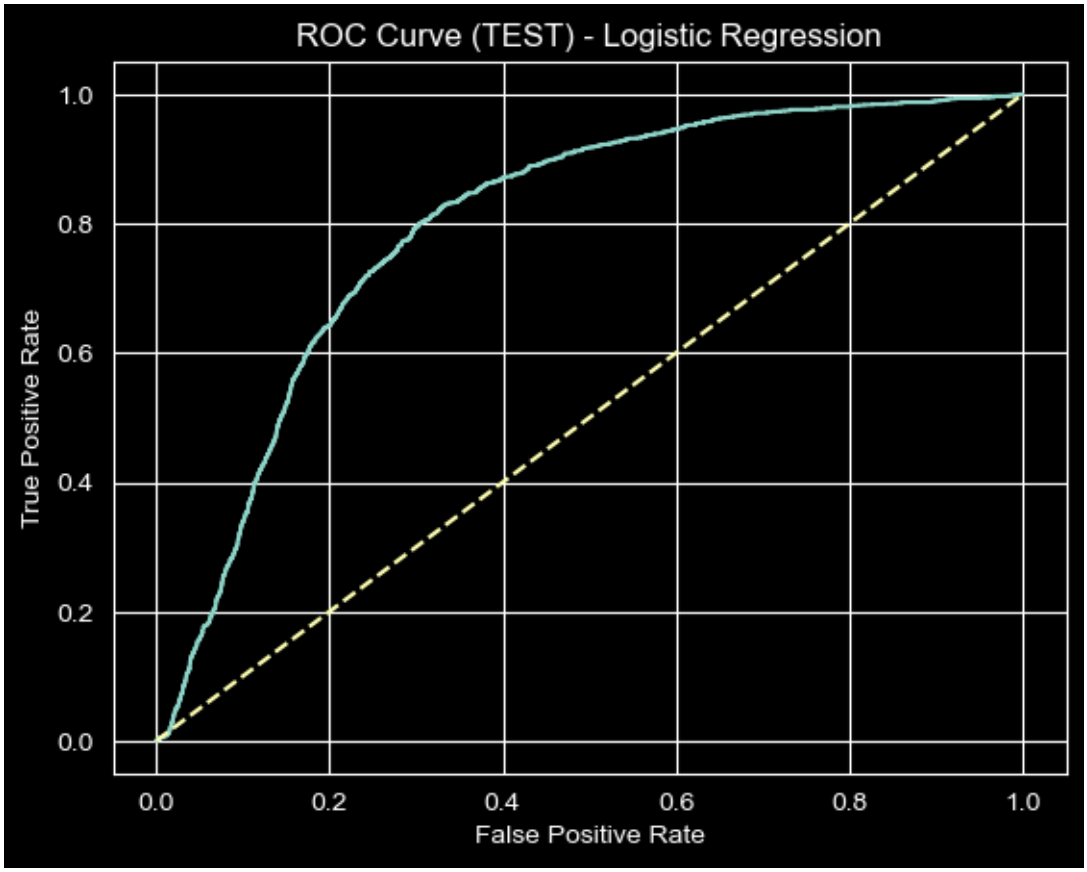
```
=== VAL (threshold from VAL-F1) ===
AUC (ROC): 0.8022
Average Precision (PR-AUC): 0.2410
Brier score (calibration): 0.1931
Threshold: 0.582
Confusion matrix [ [TN FP] [FN TP] ]:
[[4723 1096]
 [ 219  377]]

Classification report:
              precision    recall  f1-score   support

           0     0.9557    0.8117    0.8778      5819
           1     0.2559    0.6326    0.3644       596

    accuracy                         0.7950      6415
   macro avg     0.6058    0.7221    0.6211      6415
weighted avg     0.8907    0.7950    0.8301      6415
```

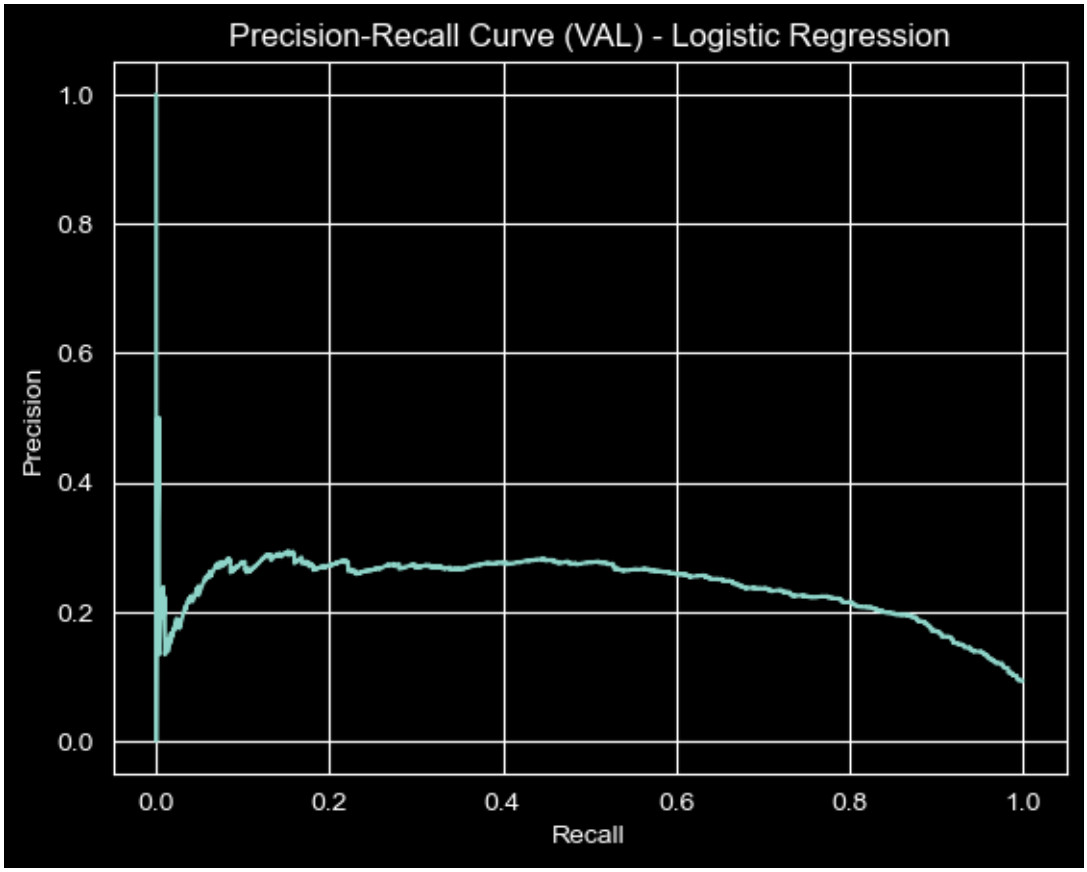
```
Sensitivity (Recall of Distress): 0.6326
Specificity (Recall of Non-Distress): 0.8117

=== TEST (same threshold) ===
AUC (ROC): 0.7969
Average Precision (PR-AUC): 0.2479
Brier score (calibration): 0.2030
Threshold: 0.582
Confusion matrix [ [TN FP] [FN TP] ]:
[[8778 2322]
 [ 444  860]]

Classification report:
              precision    recall  f1-score   support

           0     0.9519    0.7908    0.8639     11100
           1     0.2703    0.6595    0.3834      1304

    accuracy                         0.7770     12404
   macro avg     0.6111    0.7252    0.6237     12404
weighted avg     0.8802    0.7770    0.8134     12404

Sensitivity (Recall of Distress): 0.6595
Specificity (Recall of Non-Distress): 0.7908
```

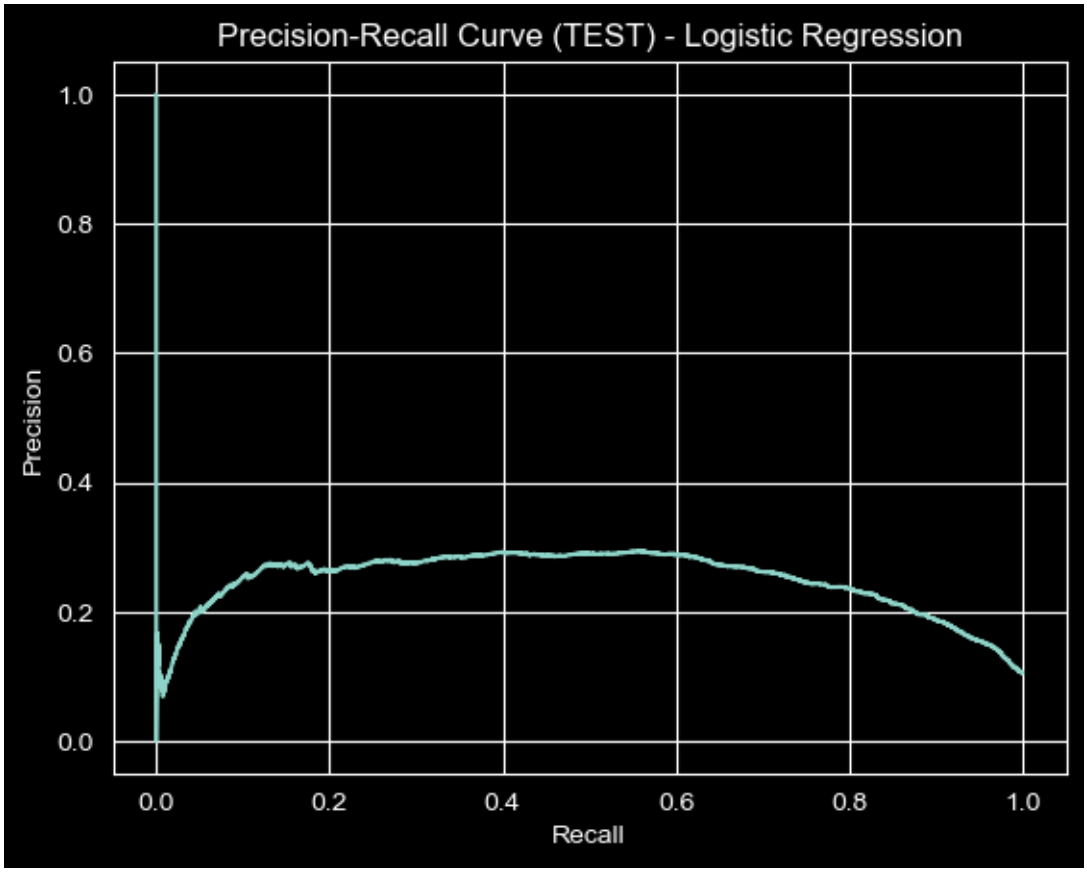
[78]:
```python
# ============================================================================
# 6.4 Diagnostic plots: ROC and Precision-Recall (VAL vs TEST)
# ============================================================================
def plot_roc(y_true, y_proba, title):
    fpr, tpr, _ = roc_curve(y_true, y_proba)
    plt.figure()
    plt.plot(fpr, tpr)
    plt.plot([0, 1], [0, 1], linestyle="--")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(title)
    plt.show()

def plot_pr(y_true, y_proba, title):
    p, r, _ = precision_recall_curve(y_true, y_proba)
    plt.figure()
    plt.plot(r, p)
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title(title)
    plt.show()
```

```
plot_roc(y_val, val_proba, "ROC Curve (VAL) - Logistic Regression")
plot_roc(y_test, test_proba, "ROC Curve (TEST) - Logistic Regression")

plot_pr(y_val, val_proba, "Precision-Recall Curve (VAL) - Logistic Regression")
plot_pr(y_test, test_proba, "Precision-Recall Curve (TEST) - Logistic␣
↪Regression")
```




ROC Curve (TEST) - Logistic Regression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


Precision-Recall Curve (VAL) - Logistic Regression
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



[79]:
```
# ============================================================================
# 6.5 Interpretability: coefficients as (approx.) log-odds contributions
# ============================================================================
coef = pd.Series(logit.coef_.ravel(), index=MODEL_FEATS).
 ↪sort_values(ascending=False)

summary = pd.DataFrame({
    "feature": coef.index,
    "coef_log_odds": coef.values,
    "odds_ratio_per_1sd": np.exp(coef.values),  # because features are z-scored
}).sort_values("coef_log_odds", ascending=False)

print("Top positive (higher distress risk):")
display(summary.head(10))

print("Top negative (lower distress risk):")
display(summary.tail(10).iloc[::-1])
```

Top positive (higher distress risk):

	feature	coef_log_odds	odds_ratio_per_1sd

```
0    z_sp_debt_to_capital       1.319711            3.742339
1                z_log_at       0.447788            1.564847
2        z_sp_dcf_to_debt       0.207369            1.230437
3        z_sp_ffo_to_debt       0.151704            1.163816
4     z_sp_debt_to_ebitda       0.125471            1.133683
5            z_log_mkvalt       0.084987            1.088702
6  z_sp_interest_coverage       0.014351            1.014454
7       z_sp_focf_to_debt      -0.088434            0.915364
8        z_sp_cfo_to_debt      -0.120954            0.886074

Top negative (lower distress risk):

                  feature  coef_log_odds  odds_ratio_per_1sd
8        z_sp_cfo_to_debt      -0.120954            0.886074
7       z_sp_focf_to_debt      -0.088434            0.915364
6  z_sp_interest_coverage       0.014351            1.014454
5            z_log_mkvalt       0.084987            1.088702
4     z_sp_debt_to_ebitda       0.125471            1.133683
3        z_sp_ffo_to_debt       0.151704            1.163816
2        z_sp_dcf_to_debt       0.207369            1.230437
1                z_log_at       0.447788            1.564847
0    z_sp_debt_to_capital       1.319711            3.742339
```

0.1.1 6.6 Optional benchmark: Tree-based model (non-linear)

If you want a second point of reference without heavy tuning, a small gradient-boosted tree model often performs well on tabular data. This is optional for the seminar paper; keep Logistic Regression as the interpretability baseline.

[ ]:
```python
from sklearn.ensemble import HistGradientBoostingClassifier

hgb = HistGradientBoostingClassifier(
    learning_rate=0.05,
    max_depth=3,
    max_iter=300,
    random_state=42,
)

hgb.fit(X_train, y_train)

val_proba_hgb = hgb.predict_proba(X_val)[:, 1]
test_proba_hgb = hgb.predict_proba(X_test)[:, 1]

print("HistGradientBoosting (no heavy tuning):")
print(f"  VAL  AUC={roc_auc_score(y_val, val_proba_hgb):.4f} |␣
  ↪PR-AUC={average_precision_score(y_val, val_proba_hgb):.4f}")
print(f"  TEST AUC={roc_auc_score(y_test, test_proba_hgb):.4f} |␣
  ↪PR-AUC={average_precision_score(y_test, test_proba_hgb):.4f}")
```

0.2 7. Summary of Model Enhancements (Feedback Integration)

Following a critical assessment of the initial model, the following improvements were integrated to meet professional research standards:

1. **Endogeneity & Simultaneity Addressing**: Added conceptual discussion of endogeneity and the predictive (vs. causal) nature of the coefficients.
2. **Temporal Validation**: Implemented walk-forward (expanding window) cross-validation across multiple time folds to verify the stability of the model across economic regimes.
3. **Multicollinearity Diagnostics**: Added Variance Inflation Factor (VIF) analysis to audit mechanical correlations between financial ratios.
4. **Complete Statistical Inference**: Migrated to `statsmodels` for the core logistic regression to provide standard errors, t/z-statistics, p-values, and 95% confidence intervals.
5. **Economic Significance**: Calculated Marginal Effects at the Mean (MEM) to translate log-odds coefficients into intuitive probability changes.
6. **Benchmark Comparison**: Added a **Persistence Benchmark** (predicting future distress using current distress) to demonstrate the incremental value of the machine learning approach.
7. **Enhanced Evaluation Metrics**: Explicitly reported Sensitivity (Recall of Distress) and Specificity, alongside AUC-ROC and PR-AUC, to better handle class imbalance.

[80]:
```
# ============================================================================
# 6.7 Export: out-of-sample predictions (VAL and TEST) for reporting / appendix
# ============================================================================
ID_COLS = [c for c in ["gvkey", "fyear", "label_year"] if c in train.columns]

pred_val = val[ID_COLS + [TARGET_COL]].copy()
pred_val["p_distress_logit"] = val_proba

pred_test = test[ID_COLS + [TARGET_COL]].copy()
pred_test["p_distress_logit"] = test_proba

pred_oos = pd.concat(
    [pred_val.assign(split="VAL"), pred_test.assign(split="TEST")],
    ignore_index=True
)

out_csv = "oos_predictions_logit.csv"
pred_oos.to_csv(out_csv, index=False)
print(f"Saved: {out_csv} | rows={len(pred_oos):,} | cols={pred_oos.shape[1]}")
```

```
Saved: oos_predictions_logit.csv | rows=18,819 | cols=6
```