Can a machine learning model accurately predict company sectors by different financial ratios?

```
!pip install \
    numpy \
    pandas \
    scikit-learn \
    matplotlib \
    seaborn \
    xgboost \
    shap \
    umap-learn \
    liac-arff \
    ucimlrepo \
    statsmodels \
    openpyxl \
  textwrap3
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: xgboost in /usr/local/lib/python3.12/dist-packages (3.1.2)
Requirement already satisfied: shap in /usr/local/lib/python3.12/dist-packages (0.50.0)
Requirement already satisfied: umap-learn in /usr/local/lib/python3.12/dist-packages (0.5.9.post2)
Requirement already satisfied: liac-arff in /usr/local/lib/python3.12/dist-packages (2.5.0)
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.12/dist-packages (0.0.7)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.12/dist-packages (0.14.5)
Requirement already satisfied: openpyxl in /usr/local/lib/python3.12/dist-packages (3.1.5)
Requirement already satisfied: textwrap3 in /usr/local/lib/python3.12/dist-packages (0.9.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.61.0
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.12/dist-packages (from xgboost) (2.27.5)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.12/dist-packages (from shap) (4.67.1)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.12/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.12/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.12/dist-packages (from shap) (3.1.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.12/dist-packages (from shap) (4.15.0)
Requirement already satisfied: pynndescent>=0.5 in /usr/local/lib/python3.12/dist-packages (from umap-learn) (0.5.13)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.12/dist-packages (from ucimlrepo) (2025.1
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.12/dist-packages (from statsmodels) (1.0.2)
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.12/dist-packages (from openpyxl) (2.0.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.12/dist-packages (from numba>=0.5
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pand
```

```
import arff
import pandas as pd
import numpy as np
import re


ds = arff.load(open("/content/sector-classification/data/2020.arff", "r", encoding="utf-8", errors="ignore"))
cols = [a[0] for a in ds["attributes"]]
df = pd.DataFrame(ds["data"], columns=cols)


df.replace("m", np.nan, inplace=True)


for c in df.columns:
    try:
      df[c] = pd.to_numeric(df[c])
    except Exception:
      pass
```

```
df.dropna(axis=1, how="all", inplace=True)
if "S" in df.columns:
    df = df[df["S"].notna()]

df.reset_index(drop=True, inplace=True)
print("Loaded from one ARFF file →", df.shape)
df.head()
final_df = df
```

```
Loaded from one ARFF file → (450, 85)
```

```
rename_map = {
    'X1':  'NetProfit_to_TotalAssets',
    'X2':  'TotalLiabilities_to_TotalAssets',
    'X3':  'WorkingCapital_to_TotalAssets',
    'X4':  'CurrentAssets_to_ShortTermLiabilities',
    'X5':  'RetainedEarnings_to_TotalAssets',
    'X6':  'GrossProfit_to_TotalAssets',
    'X7':  'BookValueEquity_to_TotalLiabilities',
    'X8':  'NetSalesRevenue_to_TotalAssets',
    'X9':  'Equity_to_TotalAssets',
    'X10': 'GrossProfitPlusFinancialExp_to_TotalAssets',
    'X11': 'GrossProfit_to_ShortTermLiabilities',
    'X12': 'GrossProfitPlusDepreciation_to_Sales',
    'X13': 'EBIT_to_TotalOperatingCosts',
    'X14': 'GrossProfitPlusDepreciation_to_TotalLiabilities',
    'X15': 'TotalAssets_to_TotalLiabilities',
    'X16': 'EBIT_to_TotalLiabilities',
    'X17': 'GrossProfit_to_Sales',
    'X18': 'EBIT_to_TotalAssets',
    'X19': 'NetProfit_to_Sales',
    'X20': 'EquityMinusShareCapital_to_TotalAssets',
    'X21': 'NetProfitPlusDepreciation_to_TotalLiabilities',
    'X22': 'EBIT_to_FinancialExpenses',
    'X23': 'WorkingCapital_to_FixedAssets',
    'X24': 'Log_TotalAssets',
    'X25': 'TotalLiabilitiesMinusCash_to_Sales',
    'X26': 'EBIT_to_Equity',
    'X27': 'OperatingExpenses_to_ShortTermLiabilities',
    'X28': 'OperatingExpenses_to_TotalLiabilities',
    'X29': 'ProfitOnSales_to_TotalAssets',
    'X30': 'TotalOperatingRevenue_to_TotalAssets',
    'X31': 'CurrentAssetsMinusInventories_to_LongTermLiabilities',
    'X32': 'ConstantCapital_to_TotalAssets',
    'X33': 'ProfitOnSales_to_Sales',
    'X34': 'CurrentAssetsMinusInventoryMinusReceivables_to_ShortTermLiabilities',
    'X35': 'EBIT_to_Sales',
    'X36': 'NetProfit_to_Inventory',
    'X37': 'GrossProfit_to_ShortTermLiabilities_Alt',
    'X38': 'EBITDA_to_TotalAssets',
    'X39': 'EBITDA_to_Sales',
    'X40': 'CurrentAssets_to_TotalLiabilities',
    'X41': 'ShortTermLiabilities_to_TotalAssets',
    'X42': 'Equity_to_FixedAssets',
    'X43': 'ConstantCapital_to_FixedAssets',
    'X44': 'WorkingCapital',
    'X45': 'NetProfit_to_Equity',
    'X46': 'LongTermLiabilities_to_Equity',
    'X47': 'Sales_to_Inventory',
    'X48': 'Sales_to_Receivables',
    'X49': 'Sales_to_ShortTermLiabilities',
    'X50': 'Sales_to_FixedAssets',
    'X51': 'CA_minus_Inv_minus_STL_to_TotalOperatingRevenue',
    'X52': 'NetProfit_to_CashFlowOps',
    'X53': 'Depreciation_to_CashFlowOps',
    'X54': 'CashFlowOps_to_TotalAssets',
    'X55': 'CashFlowOps_to_Income',
    'X56': 'CashFlowOps_to_TotalLiabilities',
    'X57': 'CashFlowOps_to_LongTermLiabilities',
    'X58': 'CashFlowOps_to_ShortTermLiabilities',
    'X59': 'NetCashFlow',
    'X60': 'CashFlowOps_to_CurrentAssets',
    'X61': 'CashFlowOps_to_EBIT',
    'X62': 'NetProfit_perShare',
```

```
    'X63': 'Income_to_OutstandingShares',
    'X64': 'PricePerShare_to_NetProfitPerShare',
    'X65': 'Dividend_to_PricePerShare',
    'X66': 'MarketCap_to_BookValue',
    'X67': 'MarketCap_to_GrossProfit',
    'X68': 'MarketCap_to_EBITDA',
    'X69': 'MarketCap_to_EBIT',
    'X70': 'MarketCap_to_TotalAssets',
    'X71': 'MarketCap_to_CapitalEmployed',
    'X72': 'SalesRatio_n_to_SalesRatio_n_1',
    'X73': 'TotalSales_n_to_TotalSales_n_1',
    'X74': 'TotalAssets_n_to_TotalAssets_n_1',
    'X75': 'CurrentAssets_n_to_CurrentAssets_n_1',
    'X76': 'EBIT_n_to_EBIT_n_1',
    'X77': 'NetProfit_n_to_NetProfit_n_1',
    'X78': 'Inventory_n_to_Inventory_n_1',
    'X79': 'Receivables_n_to_Receivables_n_1',
    'X80': 'ShortTermLiabilities_n_to_ShortTermLiabilities_n_1',
    'X81': 'CashFlowOps_n_to_CashFlowOps_n_1',
    'X82': 'NetCashFlow_n_to_NetCashFlow_n_1',
    'S':   'Sector'
}


sector_map = {
    1: "Transportation & Warehousing",
    2: "Wholesale Trade",
    3: "Manufacturing",
    4: "Retail Trade",
    5: "Energy",
    6: "Construction"
}

final_df.rename(columns=rename_map, inplace=True)
final_df = final_df.sort_values(by='Num').reset_index(drop=True)


print(final_df.dtypes.value_counts())

pd.set_option("display.max_columns", None)
pd.set_option("display.width", 200)
pd.set_option("display.max_rows", None)


display(final_df.head())
```

```
float64    84
object      1
Name: count, dtype: int64
```

	Num	Country	NetProfit_to_TotalAssets	TotalLiabilities_to_TotalAssets	WorkingCapital_to_TotalAssets	CurrentAsse
0	1.0	Poland	0.04	0.49	0.02	
1	2.0	Hungary	-0.01	0.55	-0.01	
2	3.0	Poland	0.00	0.47	0.00	
3	4.0	Poland	0.01	0.88	-0.24	
4	5.0	Poland	0.04	0.51	0.02	

```
zero_counts = (final_df == 0).sum(axis=1)

print(zero_counts)

final_df['zero_count'] = zero_counts

print(final_df['zero_count'].describe())

zero_or_nan = ((final_df == 0) | (final_df.isna())).sum(axis=1)

threshold = 0.95 * final_df.shape[1]

final_df_clean = final_df[zero_or_nan < threshold]

print(f"Dropped {(len(final_df) - len(final_df_clean))} rows out of {len(final_df)} total.")
```

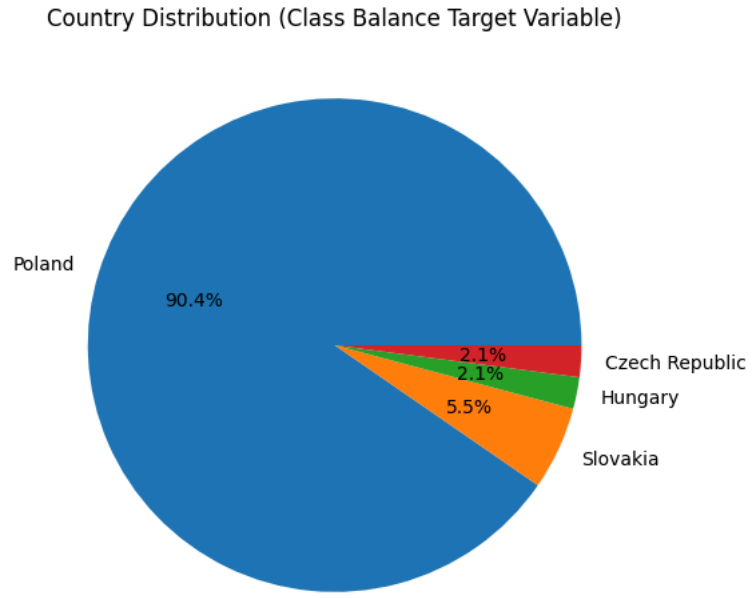
```
0       3
1       5
2      10
3       6
4      10
5       1
6      73
7      20
8       4
9       1
10      4
11      2
12      0
13      2
14      1
15      1
16      5
17      3
18     82
19      5
20      4
21     13
22      2
23      1
24      3
25      1
26     10
27      0
28     10
29     82
30      3
31     14
32      3
33      4
34      3
35     82
36      5
37     73
38      1
39      3
40      9
41      1
42      4
43     19
44      6
45      3
46      7
47      3
48     82
49      6
50      4
51      3
52      3
53      2
54      4
55      3
56      6
57      4
```

```
final_df_clean = final_df_clean.copy()
final_df_clean.rename(columns=rename_map, inplace=True)
final_df_clean = final_df_clean.sort_values(by='Num').reset_index(drop=True)
```

```
import matplotlib.pyplot as plt

d = final_df_clean['Country']

print(d.value_counts(normalize=True))

counts = d.value_counts()

plt.figure(figsize=(6,6))
plt.pie(counts.values, labels=counts.index, autopct='%1.1f%%')
plt.title("Country Distribution (Class Balance Target Variable)")
plt.savefig("/content/sector-classification/figures/country_distribution.png", dpi=300, bbox_inches="tight")
plt.show()
```

```
print('Imbalanced Data --> less than 5% in one class!')
```

```
Country
Poland            0.903896
Slovakia          0.054545
Hungary           0.020779
Czech Republic    0.020779
Name: proportion, dtype: float64
```




```
Imbalanced Data --> less than 5% in one class!
```

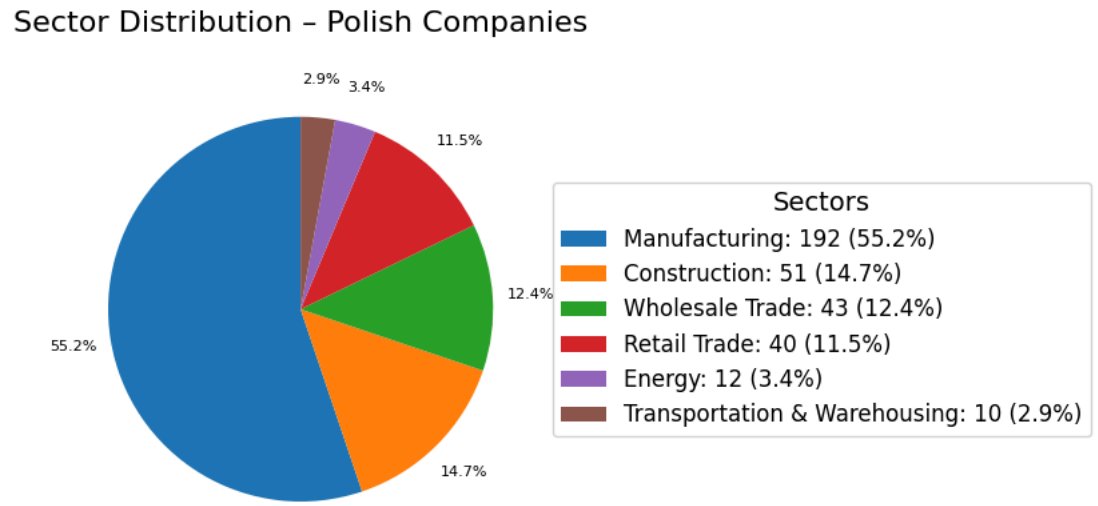
```
nonpoland_df = final_df_clean[final_df_clean['Country'] != 'Poland'].copy()
poland_df = final_df_clean[final_df_clean['Country'] == 'Poland'].copy()
```

```
X = poland_df.drop(columns=['Sector', 'Country', 'Num'])
y = poland_df['Sector']
```

```
y = poland_df['Sector']

print(y.value_counts(normalize=True))

counts = y.value_counts()

sector_labels = [sector_map.get(i, str(i)) for i in counts.index]
fig, ax = plt.subplots(figsize=(8, 8))

wedges, texts, autotexts = ax.pie(
    counts.values,
    labels=None,
    autopct='%1.1f%%',
    pctdistance=1.2,
    startangle=90,
    textprops={'fontsize': 8}
)

legend_labels = [
    f"{sector_labels[i]}: {counts.values[i]} ({counts.values[i]/counts.sum():.1%})"
    for i in range(len(counts))
]
ax.legend(
    wedges,
    legend_labels,
    title="Sectors",
    loc="center left",
    bbox_to_anchor=(1, 0.5),
    fontsize=12,
    title_fontsize=14
)
```

```
ax.set_title("Sector Distribution – Polish Companies", fontsize=16, pad=20)
plt.tight_layout()

fig.savefig("/content/sector-classification/figures/sector_distribution_poland_clean.png", dpi=300, bbox_inches="tig
plt.show()
```

```
Sector
3.0    0.551724
6.0    0.146552
2.0    0.123563
4.0    0.114943
5.0    0.034483
1.0    0.028736
Name: proportion, dtype: float64
```




```
X = X.drop(columns=['zero_count'])
num_ftrs = X.columns
print("Num features:", len(num_ftrs))
```

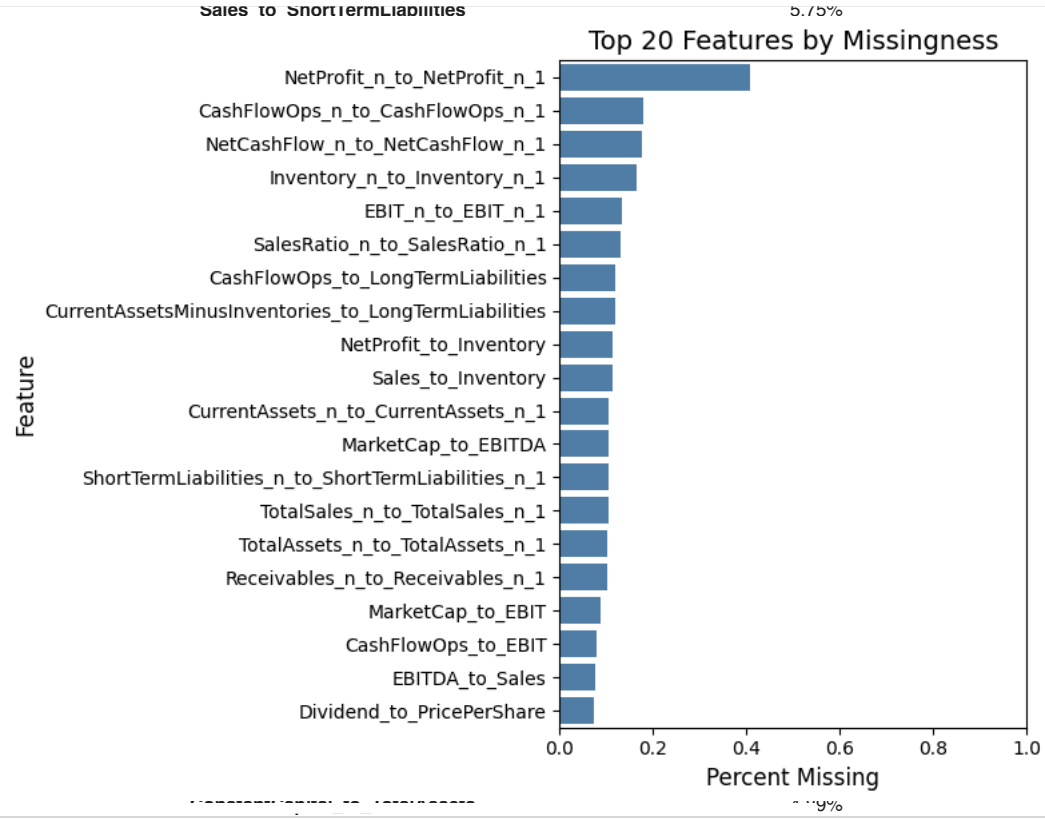
```
Num features: 82
```

```
missing_table = (
    poland_df.isnull().mean()
    .sort_values(ascending=False)
    .rename("percent_missing")
    .to_frame()
)

missing_table.style.format({"percent_missing": "{:.2%}"})
```

	percent_missing
NetProfit_n_to_NetProfit_n_1	40.80%
CashFlowOps_n_to_CashFlowOps_n_1	17.82%
NetCashFlow_n_to_NetCashFlow_n_1	17.53%
Inventory_n_to_Inventory_n_1	16.38%
EBIT_n_to_EBIT_n_1	13.22%
SalesRatio_n_to_SalesRatio_n_1	12.93%
CashFlowOps_to_LongTermLiabilities	11.78%
CurrentAssetsMinusInventories_to_LongTermLiabilities	11.78%
NetProfit_to_Inventory	11.21%
Sales_to_Inventory	11.21%
CurrentAssets_n_to_CurrentAssets_n_1	10.34%
MarketCap_to_EBITDA	10.34%
ShortTermLiabilities_n_to_ShortTermLiabilities_n_1	10.34%
TotalSales_n_to_TotalSales_n_1	10.34%
TotalAssets_n_to_TotalAssets_n_1	10.06%
Receivables_n_to_Receivables_n_1	10.06%
MarketCap_to_EBIT	8.62%
CashFlowOps_to_EBIT	7.76%
EBITDA_to_Sales	7.47%
Dividend_to_PricePerShare	7.18%
MarketCap_to_CapitalEmployed	7.18%
EBIT_to_FinancialExpenses	7.18%
CashFlowOps_to_Income	6.61%
TotalLiabilitiesMinusCash_to_Sales	6.61%
EBIT_to_Sales	6.61%
GrossProfit_to_Sales	6.61%
GrossProfitPlusDepreciation_to_Sales	6.61%
NetProfit_to_Sales	6.61%
ProfitOnSales_to_Sales	6.61%
Equity_to_FixedAssets	6.32%
WorkingCapital_to_FixedAssets	6.32%
Depreciation_to_CashFlowOps	6.32%
NetProfit_to_CashFlowOps	6.32%
ConstantCapital_to_FixedAssets	6.32%
Sales_to_FixedAssets	6.32%
MarketCap_to_GrossProfit	6.03%
OperatingExpenses_to_TotalLiabilities	5.75%
NetProfitPlusDepreciation_to_TotalLiabilities	5.75%
TotalAssets_to_TotalLiabilities	5.75%

```
import seaborn as sns

missing_table = poland_df.isnull().mean().sort_values(ascending=False)



k = 20
mt = missing_table[:k]

plt.figure(figsize=(8, 6))
```

```
sns.barplot(
    y=mt.index,
    x=mt.values,
    color="steelblue"
)

plt.xlabel("Percent Missing", fontsize=12)
plt.ylabel("Feature", fontsize=12)
plt.title(f"Top {k} Features by Missingness", fontsize=14)

plt.xlim(0, 1)

plt.tight_layout()
plt.savefig("/content/sector-classification/figures/missing_ftrs.png", dpi=300, bbox_inches="tight")
plt.show()
```

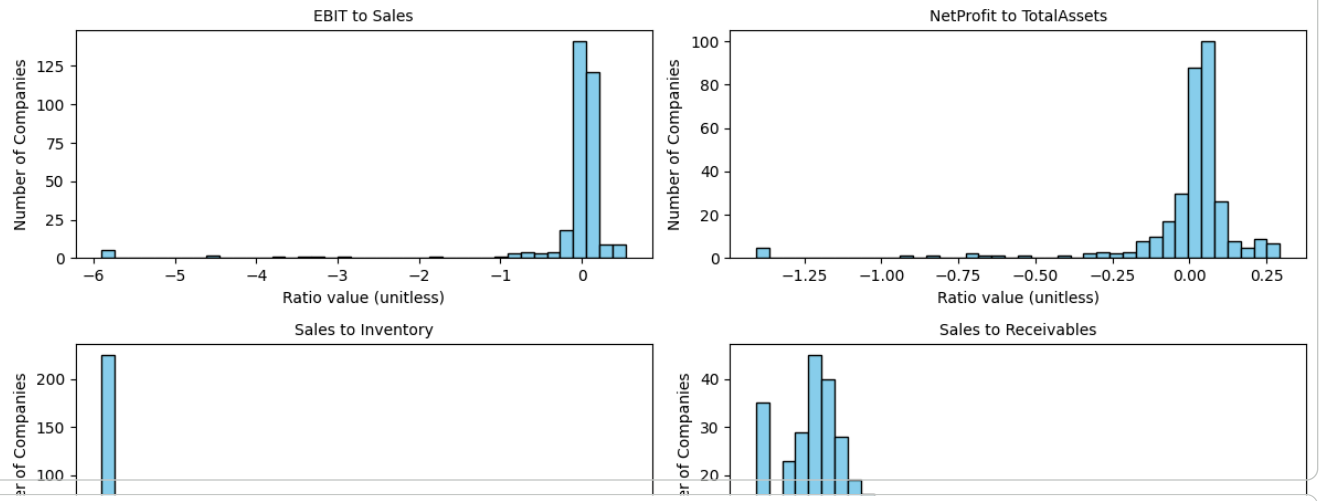



```
numeric_df = poland_df[num_ftrs]

corr = numeric_df.corr(method='spearman')

plt.figure(figsize=(16, 14))
sns.heatmap(
    corr,
    cmap='coolwarm',
    center=0,
    square=True,
    cbar_kws={'shrink': 0.5}
)
plt.title("Spearman Correlation Heatmap (Numeric Features)")
plt.savefig("/content/sector-classification/figures/correlation.png", dpi=300, bbox_inches="tight")
plt.show()
```


Spearman Correlation Heatmap (Numeric Features)
NetProfit_to_TotalAssets
WorkingCapital_to_TotalAssets
RetainedEarnings_to_TotalAssets
BookValueEquity_to_TotalLiabilities
Equity_to_TotalAssets
GrossProfit_to_ShortTermLiabilities
EBIT_to_TotalOperatingCosts
TotalAssets_to_TotalLiabilities
GrossProfit_to_Sales
NetProfit_to_Sales
NetProfitPlusDepreciation_to_TotalLiabilities
WorkingCapital_to_FixedAssets
TotalLiabilitiesMinusCash_to_Sales
OperatingExpenses_to_ShortTermLiabilities
ProfitOnSales_to_TotalAssets
CurrentAssetsMinusInventories_to_LongTermLiabilities
ProfitOnSales_to_Sales
EBIT_to_Sales
GrossProfit_to_ShortTermLiabilities_Alt
EBITDA_to_Sales
ShortTermLiabilities_to_TotalAssets
ConstantCapital_to_FixedAssets
NetProfit_to_Equity
Sales_to_Inventory
Sales_to_ShortTermLiabilities
CA_minus_Inv_minus_STL_to_TotalOperatingRevenue
Depreciation_to_CashFlowOps
CashFlowOps_to_Income
CashFlowOps_to_LongTermLiabilities
NetCashFlow
CashFlowOps_to_EBIT
Income_to_OutstandingShares
Dividend_to_PricePerShare
MarketCap_to_GrossProfit
MarketCap_to_EBIT
MarketCap_to_CapitalEmployed
TotalSales_n_to_TotalSales_n_1
CurrentAssets_n_to_CurrentAssets_n_1
NetProfit_n_to_NetProfit_n_1
Receivables_n_to_Receivables_n_1
CashFlowOps_n_to_CashFlowOps_n_1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

The correlation heatmap reveals several large clusters of highly correlated financial ratios, reflecting strong multicollinearity in the dataset. Profitability, leverage, liquidity, cash flow, and year-over-year growth features form tight internal blocks because they share common accounting denominators. This confirms the need for standardized features and regularization (L1/L2/ElasticNet) in logistic regression. Features with weaker correlations provide complementary signal but the overall structure is strongly collinear.

```
for col in num_ftrs:
    q1 = poland_df[col].quantile(0.01)
    q99 = poland_df[col].quantile(0.99)
    poland_df[col] = poland_df[col].clip(q1, q99)


for col in num_ftrs:
    poland_df[col] = pd.to_numeric(poland_df[col], errors='coerce')

features_to_plot = [
    "EBIT_to_Sales",
"NetProfit_to_TotalAssets",
"Sales_to_Inventory",
"Sales_to_Receivables",
"CashFlowOps_to_TotalAssets",
"TotalLiabilities_to_TotalAssets"

]

n_features = len(features_to_plot)
n_cols = 2
n_rows = (n_features + n_cols - 1) // n_cols

fig, axes = plt.subplots(n_rows, n_cols, figsize=(12, 3 * n_rows))
axes = axes.flatten()

for i, col in enumerate(features_to_plot):
    axes[i].hist(poland_df[col].dropna(), bins=40, color='skyblue', edgecolor='black')
    axes[i].set_title(col.replace("_", " "), fontsize=10)
    axes[i].set_xlabel("Ratio value (unitless)")
    axes[i].set_ylabel("Number of Companies")

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

fig.suptitle("Distribution of Selected Financial Ratios (Clipped at 1st and 99th Percentiles)", fontsize=14, y=1.02)
plt.tight_layout()
plt.savefig("/content/sector-classification/figures/selected_ratios.png", dpi=300, bbox_inches="tight")

plt.show()
```




```
from sklearn.model_selection import train_test_split
y_enc = y.astype(int) - 1

X_train, X_test, y_train, y_test = train_test_split(
    X, y_enc, test_size=0.2, stratify=y_enc, random_state=42
)
```

```
from sklearn.metrics import f1_score

majority_class = y_train.value_counts().idxmax()

y_pred_majority = np.full_like(y_test, fill_value=majority_class)

baseline_f1 = f1_score(y_test, y_pred_majority, average="macro")

print("Baseline F1-macro:", baseline_f1)
```

```
Baseline F1-macro: 0.11926605504587157
```

```
import numpy as np
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier


cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scoring = "f1_macro"

def run_grid(name, pipe, grid):
    gs = GridSearchCV(
        estimator=pipe,
        param_grid=grid,
        scoring=scoring,
        cv=cv,
        n_jobs=-1,
    )
    gs.fit(X_train, y_train)
    print(f"{name} | best {scoring}: {gs.best_score_:.3f} | best params: {gs.best_params_}")
    return gs
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
def imputation_uncertainty_scores(
    base_model,
    X_train,
    X_test,
    y_train,
    y_test,
    random_states=range(1, 5),
    imputer_max_iter=20,
):
    scores = []

    for rs in random_states:
        imputer = IterativeImputer(
            estimator=RandomForestRegressor(
                n_estimators=1,
                random_state=rs,
            ),
            max_iter=imputer_max_iter,
            initial_strategy="median",
        )

        pipe = Pipeline(steps=[
            ("imputer", imputer),
            ("scaler", StandardScaler()),
            ("model", base_model),
        ])

        pipe.fit(X_train, y_train)
        y_pred = pipe.predict(X_test)
        score = f1_score(y_test, y_pred, average="macro")
        scores.append(score)

    scores = np.array(scores)
    print(f"Mean: {scores.mean():.3f}, Std: {scores.std():.3f}, All: {np.round(scores, 3)}")
    return scores.mean(), scores.std(), scores
```

```
l1_pipe = Pipeline(steps=[
    ("imputer", IterativeImputer(
        random_state=42,
        max_iter=20,
        initial_strategy="median"
    )),
    ("scaler", StandardScaler()),
    ("model", LogisticRegression(
        max_iter=5000,
        solver="saga",
        tol=1e-3,
        n_jobs=-1,
        random_state=42,
    )),
])


l1_grid = {
    "model__C": np.logspace(-2, 2, 5),
    "model__penalty": ["l1"],
    "model__class_weight": ["balanced"],
}

l1_gs = run_grid("l1LR", l1_pipe, l1_grid)
```

```
l1LR | best f1_macro: 0.193 | best params: {'model__C': np.float64(10.0), 'model__class_weight': 'balanced', 'model__
```

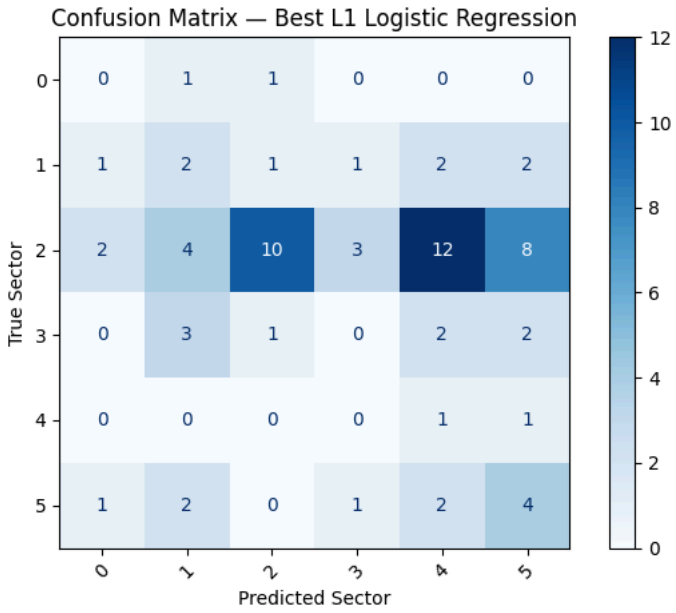
```
best_lrl1 = l1_gs.best_estimator_.named_steps["model"]
print("Best params:", l1_gs.best_params_)
print("n_iter per class:", best_lrl1.n_iter_)
print("Hit max_iter?", (best_lrl1.n_iter_ >= best_lrl1.max_iter).any())
```

```
Best params: {'model__C': np.float64(10.0), 'model__class_weight': 'balanced', 'model__penalty': 'l1'}
n_iter per class: [532]
Hit max_iter? False
```

```
imputation_uncertainty_scores(best_lrl1, X_train, X_test, y_train, y_test)
```

```
Mean: 0.240, Std: 0.012, All: [0.227 0.258 0.242 0.233]
(np.float64(0.23982972575188566),
 np.float64(0.011603765386120743),
 array([0.22707848, 0.25764895, 0.24207214, 0.23251933]))
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report

best_lrl1_model = l1_gs.best_estimator_
best_lrl1_model.fit(X_train, y_train)

y_pred = best_lrl1_model.predict(X_test)

labels = np.unique(y_test)

cm = confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap="Blues", xticks_rotation=45, values_format="d")
plt.title("Confusion Matrix — Best L1 Logistic Regression")
plt.xlabel("Predicted Sector")
plt.ylabel("True Sector")
plt.tight_layout()
plt.savefig("/content/sector-classification/figures/l1_cm.png", dpi=300, bbox_inches="tight")
plt.show()

print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=3))
```




```
Classification Report:
              precision    recall  f1-score   support

           0      0.000     0.000     0.000         2
           1      0.167     0.222     0.190         9
           2      0.769     0.256     0.385        39
           3      0.000     0.000     0.000         8
           4      0.053     0.500     0.095         2
           5      0.235     0.400     0.296        10

    accuracy                          0.243        70
   macro avg      0.204     0.230     0.161        70
weighted avg      0.485     0.243     0.284        70
```

```
l2_pipe = Pipeline(steps=[
    ("imputer", IterativeImputer(
        random_state=42,
        max_iter=20,
        initial_strategy="median"
    )),
    ("scaler", StandardScaler()),
    ("model", LogisticRegression(
```

```
            max_iter=5000,
            solver="saga",
            tol=1e-3,
            n_jobs=-1,
            random_state=42,
        )),
    ])


    l2_grid = {
        "model__C": np.logspace(-2, 2, 5),
        "model__penalty": ["l2"],
        "model__class_weight": ["balanced"],
    }

    l2_gs = run_grid("l2LR", l2_pipe, l2_grid)
```

```
l2LR | best f1_macro: 0.194 | best params: {'model__C': np.float64(10.0), 'model__class_weight': 'balanced', 'model__
```

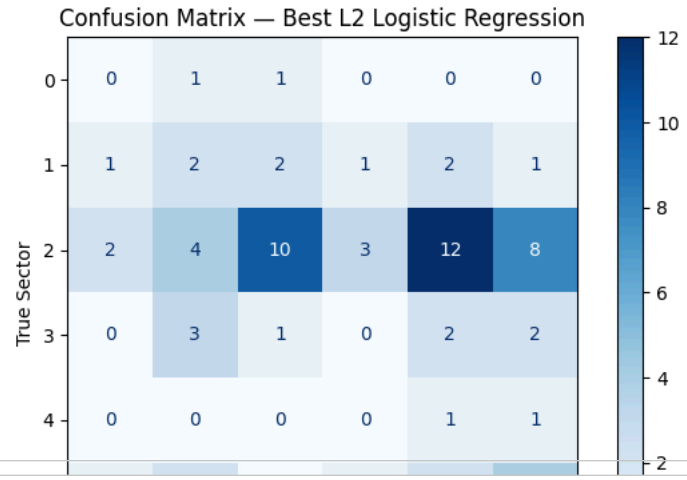
```
best_lrl2 = l2_gs.best_estimator_.named_steps["model"]
print("Best params:", l2_gs.best_params_)
print("n_iter per class:", best_lrl2.n_iter_)
print("Hit max_iter?", (best_lrl2.n_iter_ >= best_lrl2.max_iter).any())
```

```
Best params: {'model__C': np.float64(10.0), 'model__class_weight': 'balanced', 'model__penalty': 'l2'}
n_iter per class: [569]
Hit max_iter? False
```

```
imputation_uncertainty_scores(best_lrl2, X_train, X_test, y_train, y_test)
```

```
Mean: 0.231, Std: 0.008, All: [0.227 0.22  0.242 0.235]
(np.float64(0.23096390357084606),
 np.float64(0.008404472608244614),
 array([0.22707848, 0.21967244, 0.24207214, 0.23503256]))
```

```
best_lrl2_model = l2_gs.best_estimator_
best_lrl2_model.fit(X_train, y_train)

y_pred = best_lrl2_model.predict(X_test)

labels = np.unique(y_test)

cm = confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap="Blues", xticks_rotation=45, values_format="d")
plt.title("Confusion Matrix — Best L2 Logistic Regression")
plt.xlabel("Predicted Sector")
plt.ylabel("True Sector")
plt.tight_layout()
plt.savefig("/content/sector-classification/figures/l2_cm.png", dpi=300, bbox_inches="tight")
plt.show()

print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=3))
```




```
enet_pipe = Pipeline([
    ("imputer", IterativeImputer(random_state=42, initial_strategy="median", max_iter=20)),
    ("scaler", StandardScaler()),
    ("model", LogisticRegression(max_iter=5000, solver="saga",tol=1e-3, n_jobs=-1, random_state=42,))
        ])
enet_grid = {
    "model__penalty": ["elasticnet"],
    "model__C": np.logspace(-2, 2, 5),
    "model__l1_ratio": np.linspace(0.0, 1.0, 10),
    "model__class_weight": ["balanced"],
}
ElasticNetLR_gs = run_grid("ElasticNetLR", enet_pipe, enet_grid)
```

```
ElasticNetLR | best f1_macro: 0.216 | best params: {'model__C': np.float64(0.1), 'model__class_weight': 'balanced', '
4         0.053     0.500     0.095         2
5         0.250     0.400     0.308        10
```

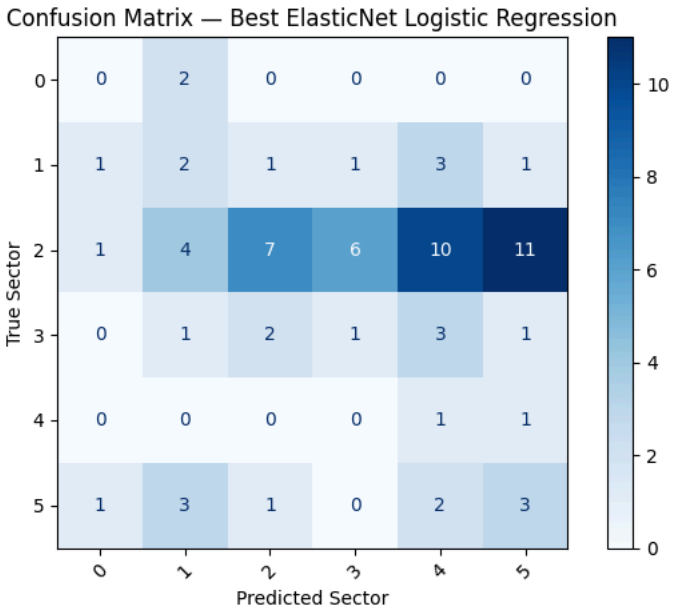
```
best_el = ElasticNetLR_gs.best_estimator_.named_steps["model"]
print("Best params:", ElasticNetLR_gs.best_params_)
print("n_iter per class:", best_el.n_iter_)
print("Hit max_iter?", (best_el.n_iter_ >= best_el.max_iter).any())
```

```
Best params: {'model__C': np.float64(0.1), 'model__class_weight': 'balanced', 'model__l1_ratio': np.float64(0.4444444
n_iter per class: [270]
Hit max_iter? False
```

```
imputation_uncertainty_scores(best_el, X_train, X_test, y_train, y_test)
```

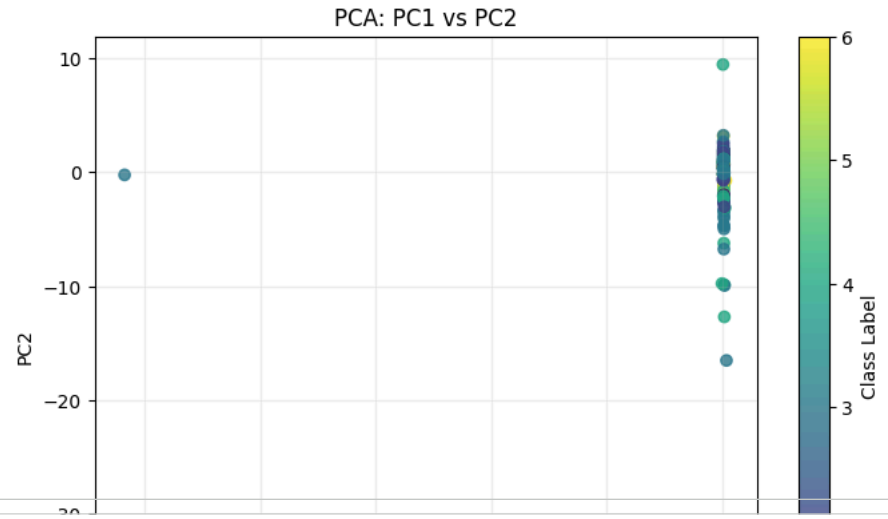
```
Mean: 0.199, Std: 0.009, All: [0.192 0.192 0.213 0.2  ]
(np.float64(0.1992161254812495),
 np.float64(0.008545814879592571),
 array([0.19171888, 0.19243992, 0.21299109, 0.19971461]))
```

```
best_el_model = ElasticNetLR_gs.best_estimator_
best_el_model.fit(X_train, y_train)

y_pred = best_el_model.predict(X_test)

labels = np.unique(y_test)

cm = confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap="Blues", xticks_rotation=45, values_format="d")
plt.title("Confusion Matrix — Best ElasticNet Logistic Regression")
plt.xlabel("Predicted Sector")
plt.ylabel("True Sector")
plt.tight_layout()
plt.savefig("/content/sector-classification/figures/elastic_cm.png", dpi=300, bbox_inches="tight")
plt.show()


print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=3))
```



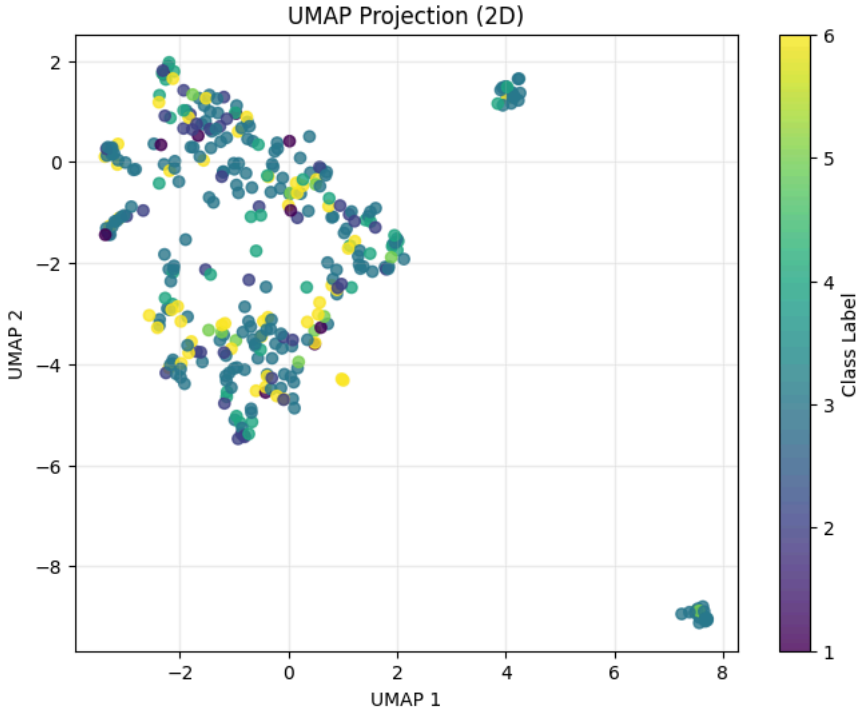

```
Classification Report:
              precision    recall  f1-score   support

           0      0.000     0.000     0.000         2
           1      0.167     0.222     0.190         9
           2      0.636     0.179     0.280        39
           3      0.125     0.125     0.125         8
           4      0.053     0.500     0.095         2
           5      0.176     0.300     0.222        10

    accuracy                          0.200        70
   macro avg      0.193     0.221     0.152        70
weighted avg      0.417     0.200     0.229        70
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import IterativeImputer

imp = IterativeImputer(random_state=42, initial_strategy="median", max_iter=20)
X_imputed = imp.fit_transform(X)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

pc1 = X_pca[:, 0]
pc2 = X_pca[:, 1]

plt.figure(figsize=(8,6))
scatter = plt.scatter(pc1, pc2, c=y, cmap="viridis", alpha=0.8)

plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("PCA: PC1 vs PC2")
plt.colorbar(scatter, label="Class Label")
plt.grid(True, alpha=0.2)
plt.savefig("/content/sector-classification/figures/PCA.png", dpi=300, bbox_inches="tight")
plt.show()
```




```
import umap

imp = IterativeImputer(random_state=42, initial_strategy="median", max_iter=20)
X_imputed = imp.fit_transform(X)


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)


reducer = umap.UMAP(
    n_neighbors=15,
    min_dist=0.1,
    n_components=2,
    metric="euclidean",
    random_state=42
)

X_umap = reducer.fit_transform(X_scaled)

u1 = X_umap[:, 0]
u2 = X_umap[:, 1]

plt.figure(figsize=(8,6))
scatter = plt.scatter(u1, u2, c=y, cmap="viridis", alpha=0.8)

plt.xlabel("UMAP 1")
plt.ylabel("UMAP 2")
plt.title("UMAP Projection (2D)")
plt.colorbar(scatter, label="Class Label")
plt.grid(True, alpha=0.2)
plt.savefig("/content/sector-classification/figures/UMAP.png", dpi=300, bbox_inches="tight")
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by setting ra
  warn(
```




```
xgb_pipe = Pipeline([
    ("imputer", IterativeImputer(random_state=42, initial_strategy="median", max_iter=20)),
    ("model", XGBClassifier(
        objective="multi:softprob",
        eval_metric="mlogloss",
        tree_method="hist",
        random_state=42
    ))
])
xgb_grid = {
    "model__n_estimators": [525, 550],
    "model__max_depth": [4, 5],
    "model__learning_rate": [0.03, 0.04],
    "model__subsample": [0.6, 0.65],
    "model__colsample_bytree": [0.8, 0.85],
}
XGB_gs = run_grid("XGB", xgb_pipe, xgb_grid)
```

```
XGB | best f1_macro: 0.323 | best params: {'model__colsample_bytree': 0.85, 'model__learning_rate': 0.04, 'model__max
```

```
best_xgb = XGB_gs.best_estimator_.named_steps["model"]

print("Best params:", XGB_gs.best_params_)
print("n_estimators:", best_xgb.n_estimators)
print("max_depth:", best_xgb.max_depth)
print("learning_rate:", best_xgb.learning_rate)
```

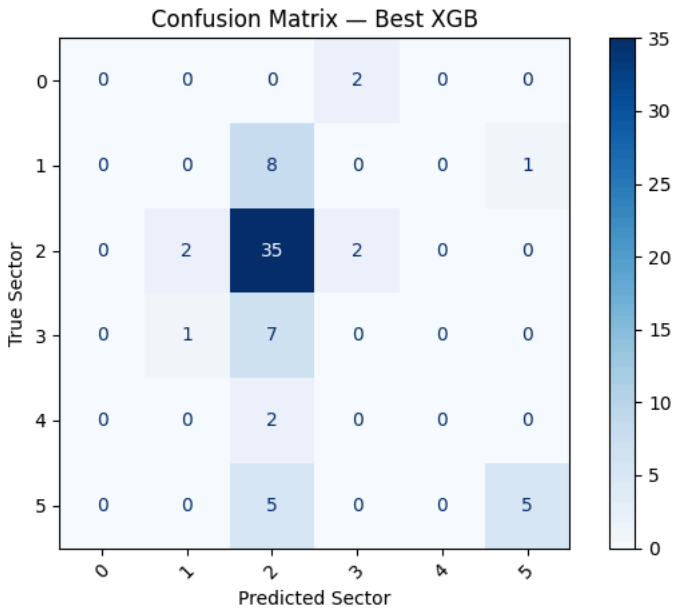
```
Best params: {'model__colsample_bytree': 0.85, 'model__learning_rate': 0.04, 'model__max_depth': 5, 'model__n_estimat
n_estimators: 550
max_depth: 5
learning_rate: 0.04
```

```
imputation_uncertainty_scores(best_xgb, X_train, X_test, y_train, y_test)
```

```
Mean: 0.220, Std: 0.010, All: [0.228 0.232 0.207 0.213]
(np.float64(0.22009178531434972),
 np.float64(0.010313152922921023),
 array([0.22813239, 0.23199666, 0.20694444, 0.21329365]))
```

```
best_xgb_model = XGB_gs.best_estimator_
best_xgb_model.fit(X_train, y_train)
```

```
y_pred = best_xgb_model.predict(X_test)

labels = np.unique(y_test)

cm = confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap="Blues", xticks_rotation=45, values_format="d")
plt.title("Confusion Matrix — Best XGB")
plt.xlabel("Predicted Sector")
plt.ylabel("True Sector")
plt.tight_layout()
plt.savefig("/content/sector-classification/figures/confusion_matrix_xgb.png", dpi=300, bbox_inches="tight")
plt.show()



print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=3))
```




```
Classification Report:
              precision    recall  f1-score   support

           0      0.000     0.000     0.000         2
           1      0.000     0.000     0.000         9
           2      0.614     0.897     0.729        39
           3      0.000     0.000     0.000         8
           4      0.000     0.000     0.000         2
           5      0.833     0.500     0.625        10

    accuracy                          0.571        70
   macro avg      0.241     0.233     0.226        70
weighted avg      0.461     0.571     0.496        70

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
rf_pipe = Pipeline([
    ("imputer", IterativeImputer(random_state=42, initial_strategy="median", max_iter=20)),
    ("model", RandomForestClassifier(random_state=42))
])
rf_grid = {
    "model__n_estimators": [550, 600, 650],
    "model__max_depth": [None, 1],
    "model__min_samples_leaf": [3,5, 6],
    "model__max_features": ["sqrt", "log2"],
    "model__class_weight": ["balanced_subsample"]
}
```

```
rf_gs= run_grid("RandomForest", rf_pipe, rf_grid)
```

```
RandomForest | best f1_macro: 0.269 | best params: {'model__class_weight': 'balanced_subsample', 'model__max_depth':
```

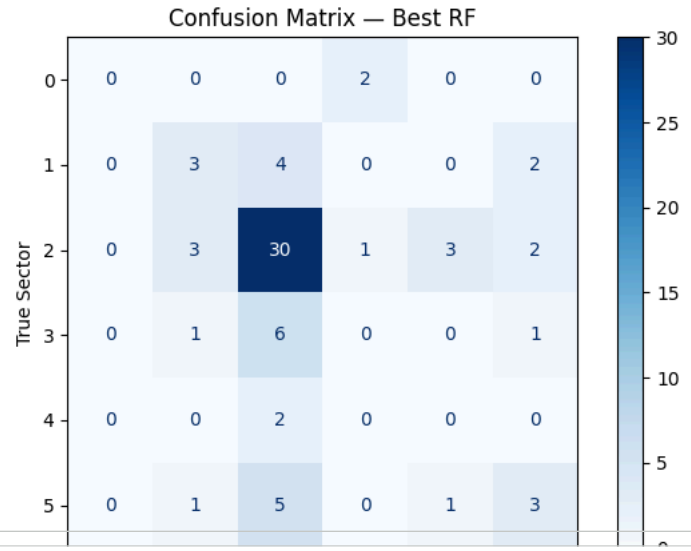
```
best_rf = rf_gs.best_estimator_.named_steps["model"]

print("Best RF params:", rf_gs.best_params_)
print("n_estimators:", best_rf.n_estimators)
print("max_depth:", best_rf.max_depth)
print("min_samples_split:", best_rf.min_samples_split)
print("min_samples_leaf:", best_rf.min_samples_leaf)
print("max_features:", best_rf.max_features)

print("Feature importances shape:", best_rf.feature_importances_.shape)
```

```
Best RF params: {'model__class_weight': 'balanced_subsample', 'model__max_depth': None, 'model__max_features': 'sqrt'
n_estimators: 550
max_depth: None
min_samples_split: 2
min_samples_leaf: 5
max_features: sqrt
Feature importances shape: (82,)
```

```
imputation_uncertainty_scores(best_rf, X_train, X_test, y_train, y_test)
```

```
Mean: 0.235, Std: 0.007, All: [0.227 0.235 0.245 0.232]
(np.float64(0.2346439353413457),
 np.float64(0.00653743461515374),
 array([0.22682888, 0.23525682, 0.24470899, 0.23178105]))
```

```
best_rf_model = rf_gs.best_estimator_
best_rf_model.fit(X_train, y_train)

y_pred = best_rf_model.predict(X_test)

labels = np.unique(y_test)

cm = confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap="Blues", xticks_rotation=45, values_format="d")
plt.title("Confusion Matrix – Best RF")
plt.xlabel("Predicted Sector")
plt.ylabel("True Sector")
plt.tight_layout()
plt.savefig("/content/sector-classification/figures/confusion_matrix_rf.png", dpi=300, bbox_inches="tight")
plt.show()

print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=3))
```




```
svm_pipe = Pipeline([
    ("imputer", IterativeImputer(random_state=42, initial_strategy="median", max_iter=20)),
    ("scaler", StandardScaler()),
    ("model", SVC(kernel="rbf"))
])
svm_grid = {
    "model__C": np.logspace(-2, 2, 5),
    "model__gamma": np.logspace(-3, 0, 4),
    "model__class_weight": ["balanced"]
}

svm_gs = run_grid("SVM_RBF", svm_pipe, svm_grid)
```

```
    accuracy                           0.514        70
```

```
best_svm = svm_gs.best_estimator_.named_steps["model"]

print("Best SVM params:", svm_gs.best_params_)
print("Kernel:", best_svm.kernel)
print("C:", best_svm.C)

if best_svm.kernel == "rbf":
    print("Gamma:", best_svm.gamma)
```

```
imputation_uncertainty_scores(best_svm, X_train, X_test, y_train, y_test)
```

```
Mean: 0.193, Std: 0.008, All: [0.186 0.202 0.183 0.201]
(np.float64(0.19323336298380372),
 np.float64(0.008373322873408966),
 array([0.18640718, 0.20239603, 0.18348697, 0.20064327]))
```

```
best_svm_model = svm_gs.best_estimator_
best_svm_model.fit(X_train, y_train)

y_pred = best_svm_model.predict(X_test)

labels = np.unique(y_test)

cm = confusion_matrix(y_test, y_pred, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap="Blues", xticks_rotation=45, values_format="d")
plt.title("Confusion Matrix — Best SVM")
plt.xlabel("Predicted Sector")
plt.ylabel("True Sector")
plt.tight_layout()
plt.savefig("/content/sector-classification/figures/confusion_matrix_svm.png", dpi=300, bbox_inches="tight")
plt.show()

print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=3))
```

```
models = {
    "L1_LogReg": l1_gs,
```

```
    "L2_LogReg": l2_gs,
    "ElasticNet_LogReg": ElasticNetLR_gs,
    "RandomForest": rf_gs,
    "XGBoost": XGB_gs,
    "SVM": svm_gs,
}

results = []

for name, gs in models.items():
    best_pipe = gs.best_estimator_
    cv_macro_f1 = gs.best_score_

    y_pred = best_pipe.predict(X_test)
    test_macro_f1 = f1_score(y_test, y_pred, average="macro")

    results.append((name, cv_macro_f1, test_macro_f1))

print("Model comparison (macro-F1):")
for name, cv_f1, test_f1 in results:
    print(f"{name:15s} | CV: {cv_f1:.3f} | Test: {test_f1:.3f}")

winner_name, winner_cv_f1, winner_test_f1 = max(results, key=lambda x: x[1])
winner_gs = models[winner_name]
winner_pipe = winner_gs.best_estimator_

print("\nWINNER (by CV macro-F1):")
print(f"{winner_name} | CV macro-F1 = {winner_cv_f1:.3f} | Test macro-F1 = {winner_test_f1:.3f}")
```

```
y_pred_winner = winner_pipe.predict(X_test)

print("\nClassification report for winner:")
print(classification_report(y_test, y_pred_winner, digits=3))

print("Confusion matrix for winner:")
print(confusion_matrix(y_test, y_pred_winner))
```

```
Classification report for winner:
              precision    recall  f1-score   support

           0      0.000     0.000     0.000         2
           1      0.000     0.000     0.000         9
           2      0.614     0.897     0.729        39
           3      0.000     0.000     0.000         8
           4      0.000     0.000     0.000         2
           5      0.833     0.500     0.625        10

    accuracy                          0.571        70
   macro avg      0.241     0.233     0.226        70
weighted avg      0.461     0.571     0.496        70

Confusion matrix for winner:
[[ 0  0  0  2  0  0]
 [ 0  0  8  0  0  1]
 [ 0  2 35  2  0  0]
 [ 0  1  7  0  0  0]
 [ 0  0  2  0  0  0]
 [ 0  0  5  0  0  5]]
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
from textwrap import fill


data = [
    ("Logistic Regression (L1)", "C, penalty, class_weight",
     "C ∈ logspace[−2,2] (5); penalty=l1; class_weight=balanced"),
    ("Logistic Regression (L2)", "C, penalty, class_weight",
     "C ∈ logspace[−2,2] (5); penalty=l2; class_weight=balanced"),
    ("Elastic Net Logistic Regression", "C, l1_ratio, penalty, class_weight",
     "C ∈ logspace[−2,2]; l1_ratio ∈ linspace[0,1] (10); penalty=elasticnet; class_weight=balanced"),
```

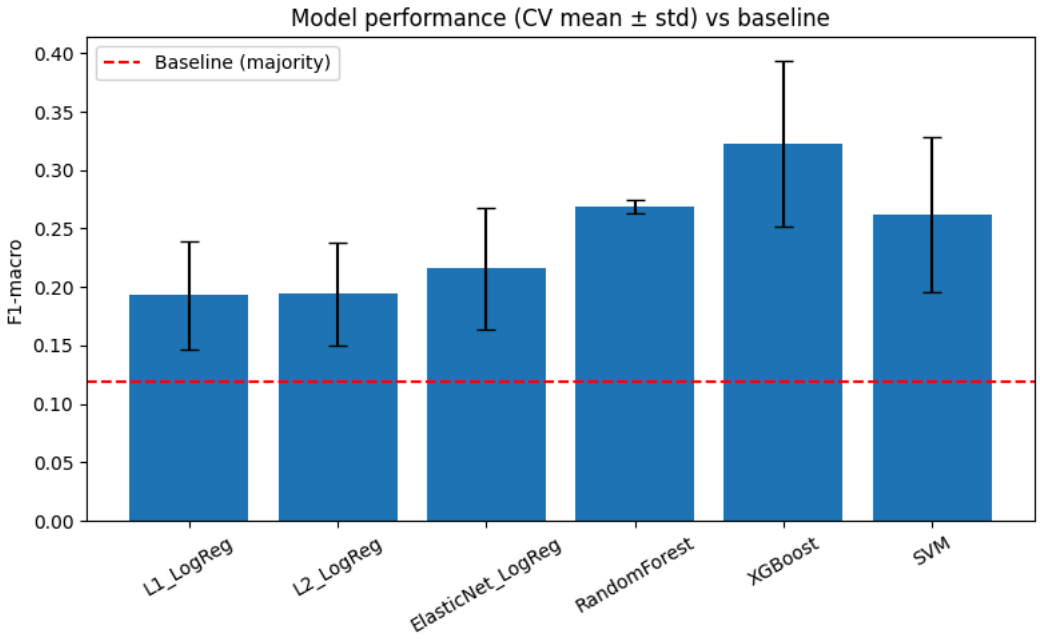
```
    ("Random Forest", "n_estimators, max_depth, min_samples_leaf, max_features, class_weight",
     "n_estimators=[550,600,650]; max_depth=[None,1]; min_samples_leaf=[3,5,6]; "
     "max_features=[sqrt,log2]; class_weight=balanced_subsample"),
    ("XGBoost", "n_estimators, max_depth, learning_rate, subsample, colsample_bytree",
     "n_estimators=[525,550]; max_depth=[4,5]; learning_rate=[0.03,0.04]; "
     "subsample=[0.6,0.65]; colsample_bytree=[0.8,0.85]"),
    ("SVM (RBF)", "C, gamma, class_weight",
     "C ∈ logspace[-2,2] (5); gamma ∈ logspace[-3,0] (4); class_weight=balanced"),
]

df = pd.DataFrame(data, columns=["Algorithm", "Parameters Tuned", "Search Space"])

def wrap(x, width=32):
    return fill(str(x), width=width)

df_wrapped = df.applymap(wrap)


line_counts = df_wrapped.applymap(lambda s: s.count("\n") + 1)



fig, ax = plt.subplots(figsize=(14, 10))
ax.axis('off')

table = ax.table(
    cellText=df_wrapped.values,
    colLabels=df_wrapped.columns,
    loc='center',
    cellLoc='left'
)

table.auto_set_font_size(False)
table.set_fontsize(11)


base_height = 0.045
header_height = 0.05

n_rows = df_wrapped.shape[0]
n_cols = df_wrapped.shape[1]


for col in range(n_cols):
    cell = table[(0, col)]
    cell.set_height(header_height)
    cell.set_text_props(va='center', ha='center', weight='bold')


for i in range(n_rows):

    max_lines = int(line_counts.iloc[i].max())
    this_height = base_height * max_lines

    for col in range(n_cols):
        cell = table[(i + 1, col)]
        cell.set_height(this_height)
        cell.set_text_props(va='center', ha='left')


for col in range(n_cols):
    for row in range(n_rows + 1):
        cell = table[(row, col)]
        cell.set_width(0.33)


plt.savefig("/content/sector-classification/figures/model_grid_summary.png", dpi=300, bbox_inches='tight')
print("Saved model_grid_summary.png")
```

```
models = {
    "L1_LogReg": l1_gs,
    "L2_LogReg": l2_gs,
    "ElasticNet_LogReg": ElasticNetLR_gs,
    "RandomForest": rf_gs,
    "XGBoost": XGB_gs,
```

```
    "SVM": svm_gs,
}

rows = []

for name, gs in models.items():
    best_idx = gs.best_index_
    mean_best = gs.cv_results_["mean_test_score"][best_idx]
    std_best  = gs.cv_results_["std_test_score"][best_idx]
    rows.append([name, mean_best, std_best])

summary_df = pd.DataFrame(rows, columns=["Model", "CV mean F1-macro", "CV std F1-macro"])
summary_df = summary_df.round(3)
print(summary_df)

fig, ax = plt.subplots(figsize=(6,2.5))
ax.axis('off')

tbl = ax.table(cellText=summary_df.values,
               colLabels=summary_df.columns,
               loc='center')

tbl.auto_set_font_size(False)
tbl.set_fontsize(8)
tbl.scale(1,1.3)

fig.savefig('/content/sector-classification/figures/cv_summary_table.png', dpi=300, bbox_inches='tight')
plt.close(fig)
```

```
              Model  CV mean F1-macro  CV std F1-macro
0          L1_LogReg             0.193            0.046
1          L2_LogReg             0.194            0.044
2  ElasticNet_LogReg             0.216            0.052
3       RandomForest             0.269            0.006
4            XGBoost             0.323            0.071
5                SVM             0.262            0.066
```

```
models = summary_df["Model"]
means  = summary_df["CV mean F1-macro"]
stds   = summary_df["CV std F1-macro"]

plt.figure(figsize=(8,5))
plt.bar(models, means, yerr=stds, capsize=5)
plt.axhline(baseline_f1, color="red", linestyle="--", label="Baseline (majority)")
plt.ylabel("F1-macro")
plt.title("Model performance (CV mean ± std) vs baseline")
plt.xticks(rotation=30)
plt.legend()
plt.tight_layout()
plt.savefig("/content/sector-classification/figures/model_performance.png", dpi=300, bbox_inches="tight")
plt.show()
```



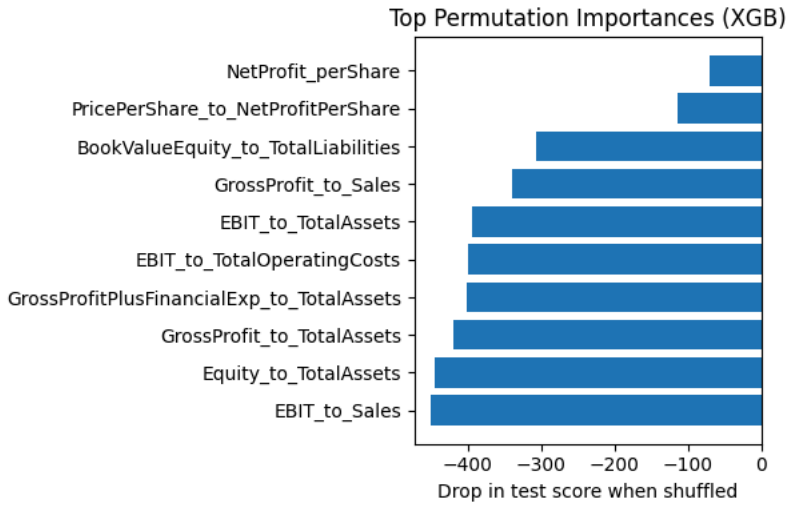

```
np.random.seed(42)

nr_runs = 10
scores = np.zeros((len(num_ftrs), nr_runs))
best_rf_pipe = rf_gs.best_estimator_


baseline_score = best_rf_pipe.score(X_test, y_test)
print("Baseline test score:", np.round(baseline_score, 3))

for i, ftr in enumerate(num_ftrs):
    print("Shuffling feature:", ftr)
    run_scores = []
    for j in range(nr_runs):
        X_test_shuffled = X_test.copy()
        X_test_shuffled[ftr] = np.random.permutation(X_test[ftr].values)
        run_scores.append(best_rf_pipe.score(X_test_shuffled, y_test))
```

```
Baseline test score: 0.514
Shuffling feature: NetProfit_to_TotalAssets
Shuffling feature: TotalLiabilities_to_TotalAssets
Shuffling feature: WorkingCapital_to_TotalAssets
Shuffling feature: CurrentAssets_to_ShortTermLiabilities
Shuffling feature: RetainedEarnings_to_TotalAssets
Shuffling feature: GrossProfit_to_TotalAssets
Shuffling feature: BookValueEquity_to_TotalLiabilities
Shuffling feature: NetSalesRevenue_to_TotalAssets
Shuffling feature: Equity_to_TotalAssets
Shuffling feature: GrossProfitPlusFinancialExp_to_TotalAssets
Shuffling feature: GrossProfit_to_ShortTermLiabilities
Shuffling feature: GrossProfitPlusDepreciation_to_Sales
Shuffling feature: EBIT_to_TotalOperatingCosts
Shuffling feature: GrossProfitPlusDepreciation_to_TotalLiabilities
Shuffling feature: TotalAssets_to_TotalLiabilities
Shuffling feature: EBIT_to_TotalLiabilities
Shuffling feature: GrossProfit_to_Sales
Shuffling feature: EBIT_to_TotalAssets
Shuffling feature: NetProfit_to_Sales
Shuffling feature: EquityMinusShareCapital_to_TotalAssets
Shuffling feature: NetProfitPlusDepreciation_to_TotalLiabilities
Shuffling feature: EBIT_to_FinancialExpenses
Shuffling feature: WorkingCapital_to_FixedAssets
Shuffling feature: Log_TotalAssets
Shuffling feature: TotalLiabilitiesMinusCash_to_Sales
Shuffling feature: EBIT_to_Equity
Shuffling feature: OperatingExpenses_to_ShortTermLiabilities
Shuffling feature: OperatingExpenses_to_TotalLiabilities
Shuffling feature: ProfitOnSales_to_TotalAssets
Shuffling feature: TotalOperatingRevenue_to_TotalAssets
Shuffling feature: CurrentAssetsMinusInventories_to_LongTermLiabilities
```

```
Shuffling feature: ConstantCapital_to_TotalAssets
Shuffling feature: ProfitOnSales_to_Sales
Shuffling feature: CurrentAssetsMinusInventoryMinusReceivables_to_ShortTermLiabilities
Shuffling feature: EBIT_to_Sales
Shuffling feature: NetProfit_to_Inventory
Shuffling feature: GrossProfit_to_ShortTermLiabilities_Alt
Shuffling feature: EBITDA_to_TotalAssets
Shuffling feature: EBITDA_to_Sales
Shuffling feature: CurrentAssets_to_TotalLiabilities
Shuffling feature: ShortTermLiabilities_to_TotalAssets
Shuffling feature: Equity_to_FixedAssets
Shuffling feature: ConstantCapital_to_FixedAssets
Shuffling feature: WorkingCapital
Shuffling feature: NetProfit_to_Equity
Shuffling feature: LongTermLiabilities_to_Equity
Shuffling feature: Sales_to_Inventory
Shuffling feature: Sales_to_Receivables
Shuffling feature: Sales_to_ShortTermLiabilities
Shuffling feature: Sales_to_FixedAssets
Shuffling feature: CA_minus_Inv_minus_STL_to_TotalOperatingRevenue
Shuffling feature: NetProfit_to_CashFlowOps
Shuffling feature: Depreciation_to_CashFlowOps
Shuffling feature: CashFlowOps_to_TotalAssets
Shuffling feature: CashFlowOps_to_Income
Shuffling feature: CashFlowOps_to_TotalLiabilities
Shuffling feature: CashFlowOps_to_LongTermLiabilities
```

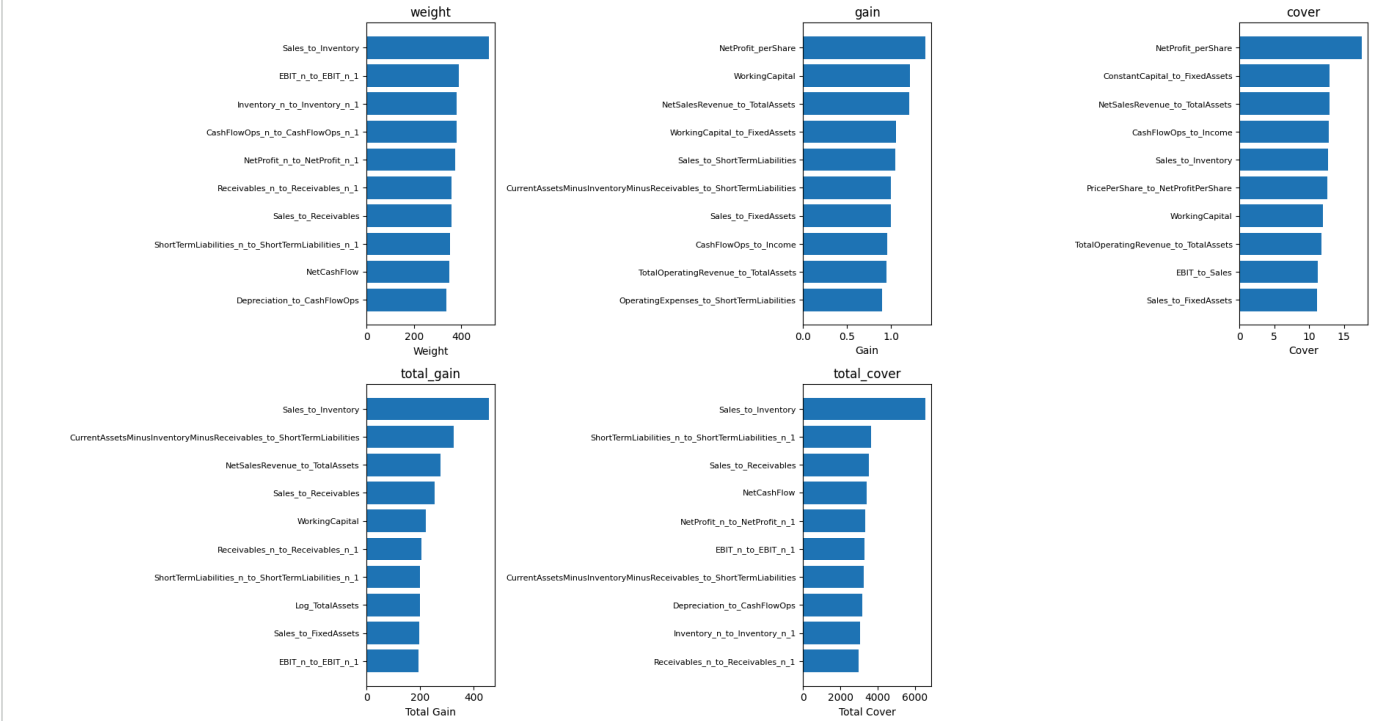
```
scores = np.asarray(scores)
num_ftrs = np.asarray(num_ftrs)

if scores.ndim == 1:
    if scores.size != len(num_ftrs):
        raise ValueError(f"scores length {scores.size} != number of features {len(num_ftrs)}")
    mean_scores = scores
elif scores.ndim == 2:
    if scores.shape[0] == len(num_ftrs):
        mean_scores = scores.mean(axis=1)
    elif scores.shape[1] == len(num_ftrs):
        mean_scores = scores.mean(axis=0)
    else:
        raise ValueError(f"scores shape {scores.shape} not aligned with features {len(num_ftrs)}")
else:
    raise ValueError(f"Unexpected scores.ndim={scores.ndim}")


drops = baseline_score - mean_scores


sorted_idx = np.argsort(drops)[::-1]
top_k = min(10, len(num_ftrs))
top_idx = sorted_idx[:top_k]
bottom_idx = sorted_idx[-top_k:]

top_features = num_ftrs[top_idx]
top_importances = drops[top_idx]

bottom_features = num_ftrs[bottom_idx]
bottom_importances = drops[bottom_idx]

print("\nTop features by permutation importance:")
for f, d in zip(top_features, top_importances):
    print(f"{f:30s} | drop = {d:.4f}")

print("\nLeast important features:")
for f, d in zip(bottom_features, bottom_importances):
    print(f"{f:30s} | drop = {d:.4f}")

plt.figure(figsize=(6, 4))
plt.barh(top_features[::-1], top_importances[::-1])
plt.xlabel("Drop in test score when shuffled")
plt.title("Top Permutation Importances (XGB)")
plt.tight_layout()
plt.savefig("/content/sector-classification/figures/permutation_importance.png", dpi=300, bbox_inches="tight")
plt.show()
```

```
Top features by permutation importance:
NetProfit_perShare             | drop = -69.6683
PricePerShare_to_NetProfitPerShare | drop = -113.4367
BookValueEquity_to_TotalLiabilities | drop = -306.3013
GrossProfit_to_Sales           | drop = -339.5959
EBIT_to_TotalAssets            | drop = -394.0189
EBIT_to_TotalOperatingCosts    | drop = -400.2067
GrossProfitPlusFinancialExp_to_TotalAssets | drop = -401.3914
GrossProfit_to_TotalAssets     | drop = -419.4860
Equity_to_TotalAssets          | drop = -445.1948
EBIT_to_Sales                  | drop = -450.1600

Least important features:
Receivables_n_to_Receivables_n_1 | drop = -2981.5370
Inventory_n_to_Inventory_n_1   | drop = -3041.3112
Depreciation_to_CashFlowOps    | drop = -3176.6556
CurrentAssetsMinusInventoryMinusReceivables_to_ShortTermLiabilities | drop = -3236.8270
EBIT_n_to_EBIT_n_1             | drop = -3283.3353
NetProfit_n_to_NetProfit_n_1   | drop = -3330.5651
NetCashFlow                    | drop = -3393.6283
Sales_to_Receivables           | drop = -3542.3614
ShortTermLiabilities_n_to_ShortTermLiabilities_n_1 | drop = -3641.5572
Sales_to_Inventory             | drop = -6550.5912
```



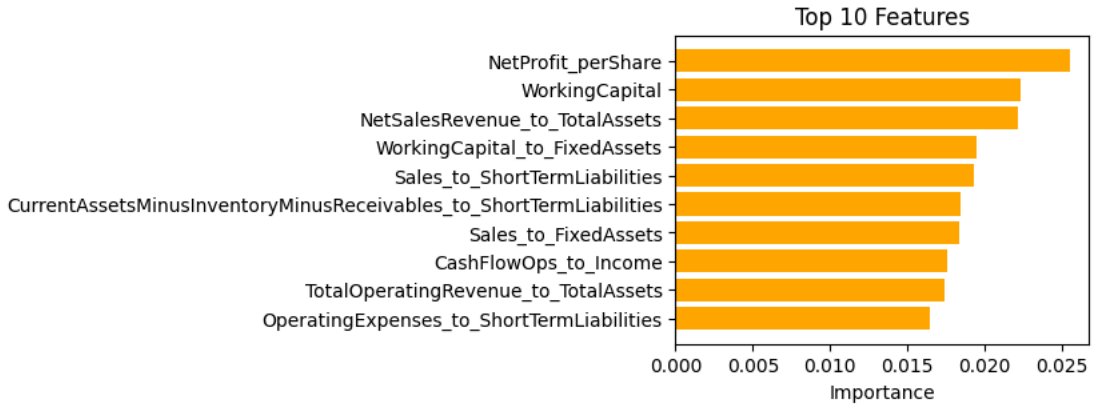

```
booster = XGB_gs.best_estimator_.named_steps["model"].get_booster()

feature_names = X_train.columns.tolist()

def importance_to_array(booster, importance_type, feature_names):
    raw = booster.get_score(importance_type=importance_type)
    arr = np.zeros(len(feature_names), dtype=float)


    if all(k.startswith('f') and k[1:].isdigit() for k in raw.keys()):
        for k, v in raw.items():
            idx = int(k[1:])
            if 0 <= idx < len(feature_names):
                arr[idx] = v
        return arr


    name_to_idx = {name: i for i, name in enumerate(feature_names)}
    for k, v in raw.items():
        if k in name_to_idx:
            arr[name_to_idx[k]] = v

    return arr

import numpy as np
import matplotlib.pyplot as plt

metrics = ["weight", "gain", "cover", "total_gain", "total_cover"]
topk = 10


fig, axs = plt.subplots(2, 3, figsize=(18, 10))
```

```
axs = axs.flatten()

for i, m in enumerate(metrics):
    scores = importance_to_array(booster, m, feature_names)

    idx = np.argsort(scores)[::-1][:topk]
    top_scores = scores[idx]
    top_names = np.array(feature_names)[idx]

    ax = axs[i]
    y = np.arange(len(idx))

    ax.barh(y, top_scores, align="center")
    ax.invert_yaxis()
    ax.set_yticks(y)
    ax.set_yticklabels(top_names, fontsize=8)
    ax.set_xlabel(m.replace("_", " ").title())
    ax.set_title(f"{m}", fontsize=12)

axs[-1].axis("off")

plt.tight_layout()
plt.savefig("/content/sector-classification/figures/xgb_feature_importance_grid.png", dpi=300, bbox_inches="tight")
plt.show()
```



```
xgb_best = XGB_gs.best_estimator_.named_steps["model"]
importances = xgb_best.feature_importances_

feat_imp = pd.DataFrame({
    "feature": X_train.columns,
    "importance": importances
}).sort_values("importance", ascending=False)

topk = 10
top = feat_imp.head(topk)


fig, ax = plt.subplots(figsize=(4,3))
ax.barh(top["feature"], top["importance"], color="orange")
ax.invert_yaxis()
ax.set_xlabel("Importance")
ax.set_title(f"Top {topk} Features")
plt.tight_layout()

fig.savefig("/content/sector-classification/figures/xgb_top10.png", dpi=300, bbox_inches="tight")
plt.show()
```

```
/tmp/ipython-input-3756544210.py:18: UserWarning: Tight layout not applied. The left and right margins cannot be made
  plt.tight_layout()
```




```
import shap
pipe = XGB_gs.best_estimator_
print(pipe.named_steps)

imputer = pipe.named_steps["imputer"]
model   = pipe.named_steps["model"]


X_test_imp = imputer.transform(X_test)


X_sample = X_test_imp[:1000]


explainer = shap.TreeExplainer(model)


ex_all = explainer(X_sample)


class_idx = 2
feature_names = X.columns


ex = shap.Explanation(
    values        = ex_all.values[:, :, class_idx],
    base_values   = ex_all.base_values[:, class_idx],
    data          = X_sample,
    feature_names = feature_names
)


plt.figure()
```

```
shap.plots.beeswarm(ex, max_display=20, show=False)
plt.title(f"SHAP Summary — Class {class_idx}")
fig = plt.gcf()
fig.savefig("/content/sector-classification/figures/shap_summary_2.png", dpi=300, bbox_inches="tight")
plt.show(fig)


plt.figure()
shap.plots.waterfall(ex[0], show=False)
plt.title(f"SHAP Waterfall — Example 0 (Class {class_idx})")
fig = plt.gcf()
fig.savefig("/content/sector-classification/figures/shap_waterfall.png", dpi=300, bbox_inches="tight")
plt.show(fig)
```