CEROP: Cross-Border Enterprise Risk & Operations Predictor

Week 4: Preparing Data

Group 4

Arthunya Kanoklertwongse

Aswin Shriram Thiagarajan

Divya Gopal

Ankit Kumar

Westcliff University

AIT 506: Machine Learning

Professor Oueichek Ibba

Preparing Data

Financial Distress Layer - Taiwan Bankruptcy Dataset

Overview

This section prepares the Taiwanese Bankruptcy dataset (https://archive.ics.uci.edu/dataset/572/taiwanese+bankruptcy+prediction) for modeling, building on the data quality findings from Week 3. That earlier analysis sorted the dataset's 95 financial ratio features into three groups: fifteen minor-outlier features, one borderline feature, and eight severely corrupted features, plus one pair of multicollinear features. Below, each group gets a tiered cleaning treatment, followed by feature engineering and a leakage-safe train/test split.

Data Copy and Reusable Function Design

All cleaning was done on a copy of the original dataframe, so the raw data stays intact. Each transformation such as capping, dropping, feature engineering was written as a standalone function rather than inline code. That way the same logic runs on both the training and test sets without duplicating it. This also forces a strict fit/apply separation: capping thresholds are learned only from the training data, then applied unchanged to the test data. No information from the test set touches the cleaning process.

```python
# ---- 2. Reusable function: fit cap thresholds (call this on TRAIN only) ----
def fit_cap_thresholds(df, columns, upper_quantile=0.99):
    """Learn the capping threshold for each column from training data only."""
    thresholds = {}
    for col in columns:
        thresholds[col] = df[col].quantile(upper_quantile)
    return thresholds


def apply_caps(df, thresholds):
    """Apply pre-fitted cap thresholds to any dataframe (train or test)."""
    df = df.copy()
    for col, cap_value in thresholds.items():
        df[col] = df[col].clip(upper=cap_value)
    return df
```

Code Snippet 1. Reusable functions to fit capping thresholds on training data and apply them to any dataset.

Outlier and Sentinel Value Treatment

The 24 features flagged in Week 3 for sentinel values were treated per tiers set at time:

Tier	# Features	Action	Rationale
Minor (<2% affected)	15	Capped at 99th percentile	Isolated data-entry errors; capping removes the corrupted values without losing the feature
Borderline (3.24% affected)	1 (Interest-bearing debt interest rate)	Capped, same treatment as minor tier	Behaves closer to the minor tier than the severe one; worth keeping as a predictor
Severe (18–88% affected)	8	Dropped entirely	Too little reliable data left to justify imputing it

Table 1 Sentinel Value Treatment Tiers and Feature Actions

"Average Collection Days," one of the fifteen minor-outlier features, shows the capping effect clearly. Before treatment, the training set contained a sentinel value as extreme as 9.73 billion, consistent with the sentinel pattern flagged in Week 3. After the 99th-percentile cap (threshold = 0.021), the maximum dropped to 0.021, putting the feature back in a business-plausible range without discarding the record.

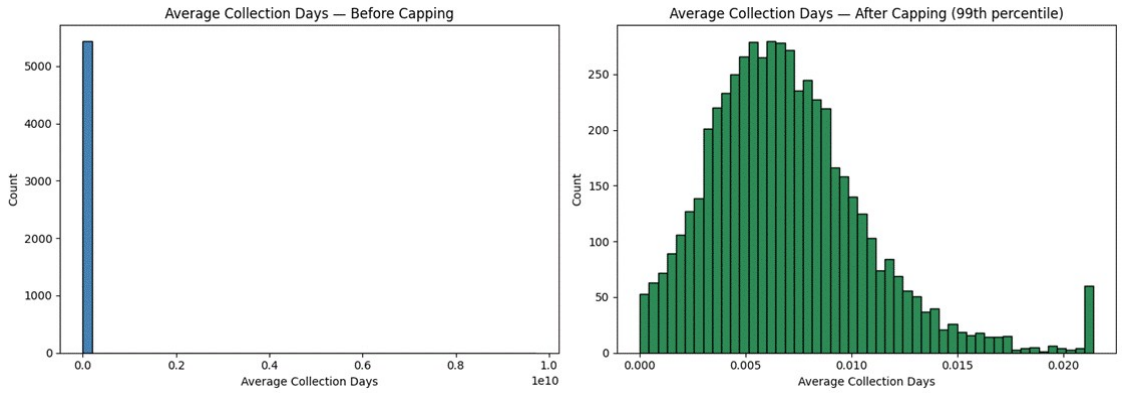


Figure 1. Distribution of "Average Collection Days" before and after 99th-percentile capping.

The full cleaning pipeline, including capping, dropping, and feature engineering, is combined into one master function:

```python
# Reusable financial preprocessing functions

def verify_exact_duplicate(df, keep_col, drop_col, atol=1e-12):
    """Confirm two columns are exact duplicates before dropping one automatically."""
    keep_values = pd.to_numeric(df[keep_col], errors="coerce").to_numpy()
    drop_values = pd.to_numeric(df[drop_col], errors="coerce").to_numpy()
    identical = np.allclose(
        keep_values,
        drop_values,
        equal_nan=True,
        atol=atol,
        rtol=0,
    )
    max_abs_diff = np.nanmax(np.abs(keep_values - drop_values))
    if not identical:
        raise ValueError(
            f"{keep_col!r} and {drop_col!r} are not exact duplicates; "
            "do not drop one automatically."
        )
    return float(max_abs_diff)


def fit_cap_thresholds(df, columns, upper_quantile=0.99):
    """Learn capping thresholds from the training data."""
    thresholds = {}

    for col in columns:
        thresholds[col] = df[col].quantile(upper_quantile)

    return thresholds


def apply_caps(df, thresholds):
    """Apply the training capping thresholds to a dataframe."""
    df = df.copy()

    for col, cap_value in thresholds.items():
        df[col] = df[col].clip(upper=cap_value)

    return df
```

```python
def drop_unreliable_columns(df, severe_cols, col_to_drop_from_pair):
    """Drop severely corrupted and multicollinear features."""
    cols_to_drop = severe_cols + [col_to_drop_from_pair]

    return df.drop(
        columns=cols_to_drop,
        errors="ignore"
    )


ROA_COLS = [
    "ROA(A) before interest and % after tax",
    "ROA(B) before interest and depreciation after tax",
    "ROA(C) before interest and depreciation before interest",
]
```

```python
def engineer_financial_features(df):
    """Create two new financial-risk features.

    Liquidity_Buffer was removed: it is an exact linear transformation of
    Net worth/Assets and Debt ratio % (both already in the feature set) and
    adds no new information. The three ROA variants are replaced by a
    single Profitability_Composite instead of being kept alongside it.
    """
    df = df.copy()

    df["Profitability_Composite"] = df[ROA_COLS].mean(axis=1)
    df = df.drop(columns=ROA_COLS)

    df["Compounding_Leverage_Risk"] = (
        df["Current Liability to Assets"] *
        df["Borrowing dependency"]
    )

    return df
```

Code Snippet 2. Master cleaning function combining capping, dropping, and feature engineering.

Multicollinearity Resolution

Week 3 found that Current Liabilities/Equity and Current Liability to Equity returned identical correlation with the target (r = 0.154). The code then verified that both columns were identical across all 6,819 records, with a maximum absolute difference of 0.0. Current Liability to Equity was dropped; Current Liabilities/Equity stays as the representative feature.

Feature Engineering

Two new features were built from existing ratios to capture relationships no single ratio covers on its own:

Feature	Formula	Business Rationale
Profitability_Composite	Mean of ROA(A), ROA(B), ROA(C)	Averages three ROA variants into one signal, cutting noise from any single calculation method
Compounding_Leverage_Risk	Current Liability to Assets × Borrowing dependency	Flags firms carrying heavy short-term liabilities while also leaning on borrowing. Neither factor alone looks alarming; together they compound distress risk

Table 2 Summary of Composite Features Built From Financial Ratios

Train/Test Split Methodology

The dataset was split 80/20 using stratified sampling on the target variable (Bankrupt?), preserving the class imbalance (96.77%/3.23%) in both sets. As above, capping thresholds were learned from the training set only and applied to the test set without re-fitting, so that no test-set information leaks into cleaning.

```
financial_train_clean, financial_cap_thresholds = clean_financial_layer(
    financial_train_raw,
    is_training=True
)

financial_test_clean = clean_financial_layer(
    financial_test_raw,
    cap_thresholds=financial_cap_thresholds,
    is_training=False
)

print("Train:", financial_train_clean.shape)
print("Test:", financial_test_clean.shape)
```

```
Verified exact duplicate pair; max absolute difference: 0.0
Verified exact duplicate pair; max absolute difference: 0.0
Train: (5455, 86)
Test: (1364, 86)
```

Code Snippet 3. Stratified 80/20 train/test split and full pipeline execution.

Final Dataset Summary

The cleaned dataset holds 85 features, down from 95 (−8 severely corrupted, −1 exact duplicate, −3 ROA variables, +2 engineered). Training set: 5,455 records. Test set: 1,364 records, consistent with an 80/20 split of the original 6,819 firms. A post-cleaning check found zero missing values in either set, so no transformation dropped data unintentionally.

Implications for Week 5

The cleaned dataset is ready for model selection. The two engineered features deserve a close look during feature importance analysis. Compounding_Leverage_Risk especially, since it combines two moderately correlated risk signals that no single original ratio captures. Profitability_Composite should also be reviewed as the replacement for the three original ROA variables.

The financial distress layer is fully prepared for Week 5 by applying the cleaning strategy with training and test data. Combined with the operations layer and the unsupervised component scoped this week. As the result, the project has now complete on preparation foundation across all three risk signals.

Operations Layer - DataCo Smart Supply Chain Dataset

Overview

In this report we documents the preprocessing pipeline built for the supply chain risk component of our Cross Border Enterprise Risk & Operations Predictor (CEROP) project, using

the DataCo Smart Supply Chain for Big Data Analysis dataset to predict Late_delivery_risk. The report follows the standard data preparation checklist: cleaning, feature selection, feature engineering, feature scaling and also added a section on unsupervised learning analysis of few features. From the previous week's EDA, we found the dataset consisted of few duplicate columns, high cardinality features like order or customer ID, and other issues that require attention. This week we will address all the data issues, and prepare it for ML modeling in the next week. The dataset initially has 53 columns and contains both numerical and categorical values.

Working Procedure

We made all data transformations as reusable functions using SKLearn library's preprocessors, encoders and data splits. We first split the data into training and test splits, followed by encoding categorical variables using one-hot encoders, and then finally performed standard scaling on numeric columns. The main goal was to clean the data, ensure there are no missing values, remove/replace outliers from features, drop unwanted columns, and select features based on domain knowledge and to avoid multi collinearity. The use of reusable methods ensures ease of constructing the pipeline across different datasets, and combine it with the financial layer effectively.

We also created a separate feature scaling and preprocessing step to perform unsupervised clustering using DBSCAN on a subset of features. This is to identify any underlying patterns/grouping and use it in other downstream layers.

Data Cleaning

We performed the following data cleaning on the supply chain datset. 1. Fill in missing values, 2. Look for duplicates in columns and drop those rows, 3. Check for redundant columns and

remove them/keep one, 4. Eliminate high cardinal features like PII, IDs, etc which will not contribute much in model building, 5. Used Windsor method to eliminate outliers and replace it with boundary values. Missing values were checked directly using (df[<col>].isna().sum()) after the IDs and redundant columns were removed. Some columns like Order Country and Customer City were in Spanish, so we converted them into English for evenness and possible join with other financial layers. Several columns were also dropped to prevent data leakage, remove redundancy, and manage cardinality.

- Leakage columns: Delivery Status, Days for shipping (real), and shipping date (DateOrders) were removed since they provide information on delivery outcome, which would leak the target variable (Late_delivery_risk).
- PII and free text columns: Features containing PII data or unstructured text/image data, such as Customer Email, Customer First name, Customer Last name, Customer Password, Customer Street, Product Image, and Product Description, were dropped.
- Redundant ID Columns: Columns like Order Customer Id, Order Item Cardprod Id, Product Category Id were found to be duplicates of Customer Id, Product Card Id, Category Id respectively and were removed. Also other features like Latitude and Longitude were dropped as they were providing redundant information.
- High Cardinality Columns: Columns with large num. of unique values such as Order Id, Order Item Id, Customer Zipcode, Order Zipcode, Customer City, Order City, Order State, Product Name, and Customer Id were also dropped.
- Price and profit related redundant variables: Order Item Total and Order Item Product Price were removed due to their high correlation with Sales and Product Price. Order Profit Per Order was also dropped as it was found to be perfectly correlated (r = 1.0) with Benefit per

order confirmed by identical outlier bounds and clipped-value counts during outlier treatment. Sales per customer was dropped as a near-duplicate of Sales ($r \approx 0.99$).

- Translation: Spanish country names in Order Country and Customer City were translated to English for consistent data joining.
- Removal of data after 2017-10-01: The data after this date had sudden spikes in order counts and sales, and we removed about 8557 rows to ensure data consistency. This is because the dataset was cut off after this date and had very few entries from 2018.

The main outlier handling was applied to the training dataset for specific numeric columns: Benefit per order, and Order Item Profit Ratio. The Interquartile Range (IQR) method also called Windsor method was used with a factor of 1.5 to eliminate outliers by clipping it to calculated upper and lower bounds. The reusable method is shown below along with a sample of box plots before and after outlier treatment.

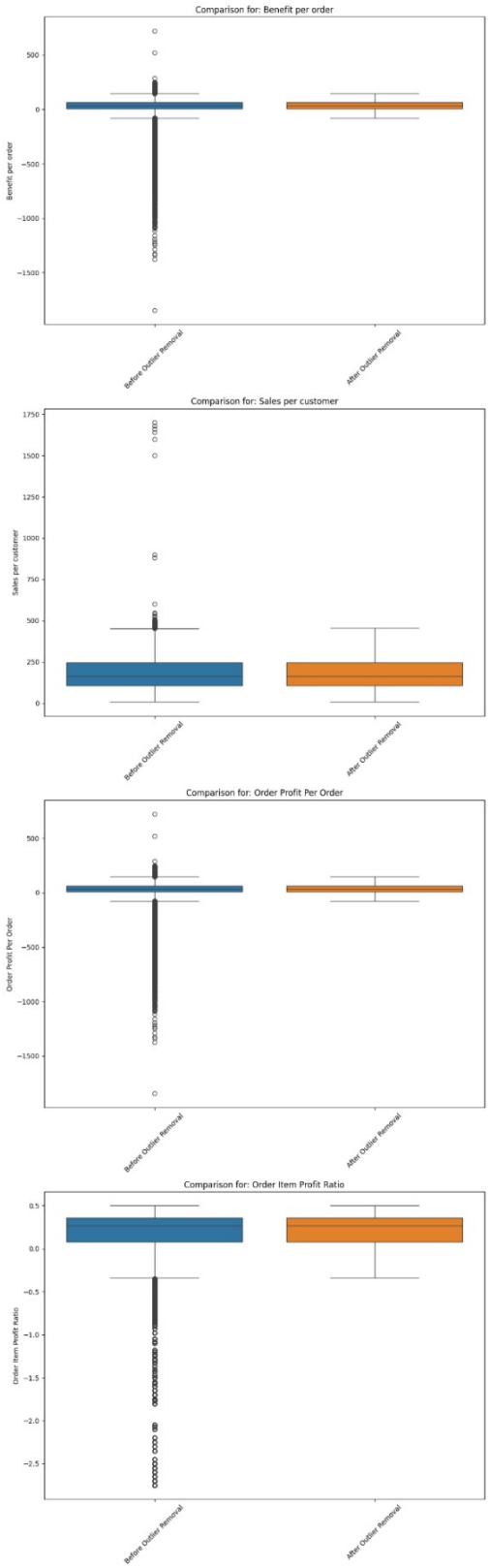


Figure 2. Box plots showing before and after outlier removal

Outlier handling

```python
def handle_outliers_iqr(df, cols, factor=1.5, method="clip"):
    """
    Handle outliers in numeric columns using the IQR method.

    Args:
        df: Input DataFrame.
        cols: Numeric columns to check.
        factor: IQR multiplier (1.5 = standard, 3.0 = lenient).
        method: "remove" (drop rows) or "clip" (cap extreme values).

    Returns:
        DataFrame with outliers handled. Bounds are computed independently
        for each column using the original input data.
    """

    result = df.copy()
    bounds = {}
    for col in cols:
        Q1, Q3 = df[col].quantile([0.25, 0.75])
        IQR = Q3 - Q1
        bounds[col] = (Q1 - factor * IQR, Q3 + factor * IQR)

    if method == "remove":
        mask = pd.Series(True, index=result.index)
        for col, (lower, upper) in bounds.items():
            col_mask = result[col].between(lower, upper)
            removed = (~col_mask & mask).sum()
            print(
                f"{col}: removing {removed} outlier rows (bounds: {lower:.1f} to {upper:.1f})"
            )
            mask &= col_mask
        result = result[mask]
        print(
            f"\nTotal rows removed: {len(df) - len(result)} ({len(df)} -> {len(result)})"
        )

    elif method == "clip":
        for col, (lower, upper) in bounds.items():
            n_clipped = (~df[col].between(lower, upper)).sum()
            result[col] = result[col].clip(lower, upper)
            print(
                f"{col}: clipped {n_clipped} values to bounds ({lower:.1f} to {upper:.1f})"
            )
        print(f"\nRow count unchanged: {len(result)}")

    else:
        raise ValueError("method must be 'remove' or 'clip'")

    return result
```

Code Snippet 4. Reusable functions to clip or remove outliers on numeric columns in any dataset

Feature Selection

For this task, we used a bit of domain knowledge and correlation based analysis to select only the features that contribute to the efficient prediction of target variable. Attributes providing no useful information were categorized in four groups. In the previous section we discussed target leakage columns, as we believe these features such as Delivery Status (contains Late Delivery, Shipping cancelled, etc), Days for shipping (real) - tells us if it was delayed or not, and shipping date (DateOrders) are only available after a delivery is made and cannot be used in predicting Late_Delivery_Risk. Personally identifiable and non-tabular columns (customer name, email, password, street, product image and description) were marked irrelevant and weren't retained. Duplicate IDs were reduced to one pair in the final data (Customer Id/Order Customer Id, Product Card Id/Order Item Cardprod Id, Category Id/Product Category Id). Zipcode columns were also dropped as we already have other geo identifiers such as City/State. We retained Order Status as we felt it describes order and payment status which known before delivery. Final selected columns before feature engineering: ['Type', 'Days for shipment (scheduled)', 'Benefit per order', 'Category Name', 'Customer Country', 'Customer Segment', 'Customer State', 'Department Name', 'Market', 'Order Country', 'Order Item Discount', 'Order Item Discount Rate', 'Order Item Profit Ratio', 'Order Item Quantity', 'Sales', 'Order Region', 'Order Status', 'Product Price', 'Shipping Mode']

Feature Engineering

Feature decomposition: We decomposed the order date into components: year, month, day of week and hour and then dropped the raw datetime column. These datetime components can reveal the underlying seasonal patterns useful for modeling. In those components Year was considered as weaker candidate for the final feature set, since the dataset only consists of 3 years of data which is not enough to generalize.

Feature Aggregation: We applied scikit learn's TargetEncoder to the high cardinality cat. columns (City, State, Country, Category Name, Department Name, Product Name). We kept a threshold of 8 or less unique values for one hot encoding vs. > 8 unique values for target encoding. This is useful in converting the categorical variables into numerical values while also not blowing up the number of new columns created.

Feature transformations: We converted the Order Country and Customer City into English to keep them homogenous with other datasets we are working.

After the Feature Engineering the final number of features are: ['Type', 'Days for shipment (scheduled)', 'Benefit per order', 'Category Name', 'Customer Country', 'Customer Segment', 'Customer State', 'Department Name', 'Market', 'Order Country', 'Order Item Discount', 'Order Item Discount Rate', 'Order Item Profit Ratio', 'Order Item Quantity', 'Sales', 'Order Region', 'Order Status', 'Product Price', 'Shipping Mode', 'Order_Year', 'Order_Month', 'Order_DayOfWeek', 'Order_Hour']

Feature Scaling

Since we are not considering a time series model for this task, we converted all numeric columns from the steps above along with the date components, were standardized using scikit learn's StandardScaler within the same ColumnTransformer as the other encoders. The whole pipeline works sequentially by applying each feature processing step on each type of columns like numeric and categorical.



Code Snippet 5. Reusable method to perform feature pre-processing on columns

Unsupervised Clustering and Future Experimentation

We added a DBSCAN based component to flag orders with anomalous profit and discount patterns which we identified earlier in the project while performing EDA. We chose a feature subset (Benefit per order, Order Item Profit Ratio, Sales, Order Item Discount Rate) and scaled it independently of the main pipeline, then fit on the training data split and applied to test. Since DBSCAN's distance based clustering needs its own comparable scale, separate from the

full one-hot and target-encoded feature space used by the classifiers. We noticed some orders that did not resemble any dense neighborhood were labeled as noise, proving this method as an effective unsupervised method for detecting anomaly (potentially fraud orders). We found 3 clusters from our analysis and there were 18 noisy transactions. The K-dist plot below shows how dense the points are located across the clusters.

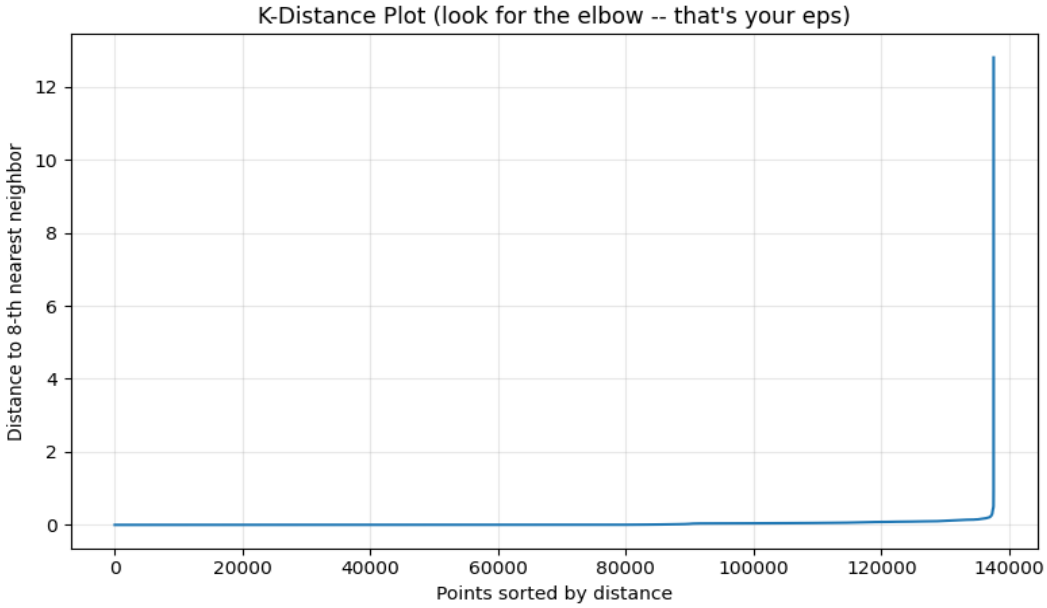


Figure 3. K-Distance plot for the DBSCAN model

Conclusion

The resulting data processing pipeline removes unreliable spikes in dataset, excludes target leakage and PII columns, resolves redundant identifiers, encodes categorical features by a cardinality rule, handles outliers without removing the data and ensures class balance, and adds an unsupervised anomaly detection component. This pipeline will directly connect with the Financial layer and will contribute to the overall ML modeling helping us with our business use case.

For coding references https://colab.research.google.com/drive/1vXsOQ_Txj5z1T8bR3tj-eSiHmyguRbeY?usp=sharing

Pipeline Integration

Overview

This section covers the integration step for Week 4: taking the two cleaned layers produced independently this week and running them through a single preprocessing pipeline that produces model-ready matrices for Week 5. The financial layer comes from the Taiwanese Bankruptcy dataset (Taiwanese Bankruptcy Prediction, 2020), with Bankrupt? as the target. The operations layer comes from the DataCo Smart Supply Chain dataset (Constante et al., 2019), with Late_delivery_risk as the target. The GSCPI series is merged into the operations layer only. The integration work itself was less about writing new cleaning logic than about three things: making sure both layers split their data before anything gets fitted, making sure the numbers coming out of the pipeline match the numbers in the two layer notebooks and saving the results so Week 5 can start from files instead of re-running everything.

Integration Approach

The financial cleaning steps were taken directly from the financial-layer notebook. For the operations layer, the same column removals, feature engineering, outlier treatment and encoding decisions from the operations notebook were followed in the combined pipeline. Since the two notebooks were created separately, the integration was checked by comparing the number of rows, the train-test split sizes and the final feature counts with the original results. The numbers matched, which confirmed that both layers were brought together correctly.

```python
def clean_financial_layer(
    df,
    cap_thresholds=None,
    is_training=False
):
    """Run the complete financial cleaning process."""
    df = df.copy()

    if is_training:
        cap_thresholds = fit_cap_thresholds(
            df,
            cols_to_cap,
            upper_quantile=0.99
        )

    if cap_thresholds is None:
        raise ValueError(
            "Training cap thresholds are required for test data."
        )

    df = apply_caps(df, cap_thresholds)

    max_duplicate_diff = verify_exact_duplicate(
        df,
        keep_col="Current Liabilities/Equity",
        drop_col=COL_TO_DROP_FROM_PAIR,
    )
    print("Verified exact duplicate pair; max absolute difference:", max_duplicate_diff)

    df = drop_unreliable_columns(
        df,
        SEVERE_COLS,
        COL_TO_DROP_FROM_PAIR
    )

    df = engineer_financial_features(df)

    if is_training:
        return df, cap_thresholds

    return df
```

Code Snippet 6. The financial master cleaning function, carried over unchanged, with its fit/apply separation.

Leakage-Safe Train/Test Split Methodology

Both datasets are split 80/20 with stratified sampling on their target before any data-dependent step runs. Everything that learns a parameter from data is fitted on the training split only and then applied to the test split without re-fitting. That covers five things: the 99th-percentile cap thresholds on the financial side, the IQR bounds on the operations side, StandardScaler on both,

the OneHotEncoder category vocabulary and the TargetEncoder mappings. The pipeline uses StandardScaler, OneHotEncoder and TargetEncoder from scikit-learn to prepare the data for modeling (Pedregosa et al., 2011).

One operation deliberately runs before the split. The November to December 2017 tail of the DataCo data was removed from the full dataset first, because it is a data-quality filter rather than something learned from the data. No parameter is estimated, the same date cutoff would apply to any future batch and leaving the artifact in the test set would have meant scoring Week 5 models against records we already know are unreliable. The financial layer has no equivalent filter.

TargetEncoder deserves a specific note since it is the one transformer that touches the labels. It is fitted with cv=5 cross-fitting and automatic smoothing, so no row contributes to its own encoded value during training. Categories that appear only a handful of times get pulled toward the global mean instead of memorising a rate from three orders.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, TargetEncoder

operations_preprocessor = ColumnTransformer(
    transformers=[
        (
            "numeric",
            StandardScaler(),
            operations_numeric_cols
        ),
        (
            "low_cardinality",
            OneHotEncoder(
                drop="first",
                handle_unknown="ignore",
                sparse_output=False
            ),
            operations_low_cardinality_cols
        ),
        (
            "high_cardinality",
            TargetEncoder(
                target_type="binary",
                smooth="auto",
                cv=5,
                random_state=42
            ),
            operations_high_cardinality_cols
        )
    ],
    remainder="drop",
    verbose_feature_names_out=True
)
```

Code Snippet 7. The operations ColumnTransformer,

showing the three encoder groups fitted on the training split only.

Financial Layer Integration

The financial layer runs exactly as specified in the layer notebook. Fifteen minor-outlier features plus the borderline Interest-bearing debt interest rate are capped at the 99th percentile using thresholds learned from training data, eight features with severe outlier problems were dropped, Current Liability to Equity is dropped after verifying it is an exact duplicate, the three ROA variables are replaced by Profitability_Composite, and Compounding_Leverage_Risk is added.

The 6,819 firms were split into 5,455 training records and 1,364 test records. The same class balance was kept in both sets, with 96.77% of the firms in the non-bankrupt group and 3.23% in the bankrupt group. After cleaning, the financial dataset had 85 features. During integration, the scaled results were converted from a NumPy array into a pandas DataFrame so the original feature names would still be available when we review feature importance in Week 5.

Operations Layer Integration

We left out the November and December 2017 records as that part of the data was identified as unreliable. The dataset then went from 180,519 rows to 171,962, which was the same as shown in the operations notebook. From there, 137,569 records were used for training and 34,393 for testing, with stratified sampling holding the class balance at 54.82% late against 45.18% on time.

Structural cleaning drops the three leakage columns (Delivery Status, Days for shipping (real), shipping date (DateOrders)), the PII and free-text columns, the confirmed duplicate ID pairs, the high-cardinality identifiers and geographic name columns and the redundant price and profit columns - Order Item Total, Order Item Product Price, Order Profit Per Order (a perfect duplicate of Benefit per order, $r = 1.0$) and Sales per customer (a near-duplicate of Sales, $r \approx 0.99$). The order date is decomposed into year, month, day of week and hour, plus a month-start key used for the GSCPI join. The raw timestamp is then dropped.

Outliers in Benefit per order and Order Item Profit Ratio are clipped at $1.5 \times$ IQR. The bounds are computed on the training split alone and then applied to both splits. This differs from the operations layer notebook, which clipped the training split only. Applying the same bounds to test keeps the two matrices on a consistent scale. Because the bounds carry no test-set information, the split protocol still holds.

Encoding Strategy

Categorical columns were handled based on what they represent rather than only by the number of categories they contain. Transaction-related columns, including Type, Customer Country, Customer Segment, Order Status and Shipping Mode, were one-hot encoded using drop="first". These have few levels and they are read directly by anyone looking at the model. Shipping Mode matters most here, since Week 3 found it was the strongest driver of late delivery. Dropping the first level avoids the collinear dummy block that would otherwise distort the logistic regression baseline.

Columns that describe geography or product taxonomy (Category Name, Department Name, Market, Order Country, Order Region) are target encoded. For these, the useful summary is the historical late-delivery rate associated with the category rather than 160-odd indicator columns. Market sits in this group despite having only five levels, on the same reasoning. It is a geographic rollup whose encoded value summarizes the smoothed historical late-delivery tendency associated with that regional category.

This gives 35 features: 13 numeric (including GSCPI), 17 one-hot columns and 5 target-encoded columns. The operations layer notebook, which assigned Market to one-hot and Customer Country to target encoding, reported 37. The difference in feature count comes from the two encoder reassignments and the number of dummy columns they produce.

Customer State was left out of the final model because the same location information was already available through the country and region fields. Keeping all three would have added overlapping information without giving the model much additional value.

Encoder	Columns	Output

		columns
StandardScaler	Days for shipment (scheduled), Benefit per order, Order Item Discount, Order Item Discount Rate, Order Item Profit Ratio, Order Item Quantity, Sales, Product Price, Order_Year, Order_Month, Order_DayOfWeek, Order_Hour, GSCPI	13
OneHotEncoder(drop="first")	Type, Customer Country, Customer Segment, Order Status, Shipping Mode	17
TargetEncoder	Category Name, Department Name, Market, Order Country, Order Region	5
Dropped	Customer State	0

Table 3. Column groups and encoder assignment.

Scope Note: Unsupervised Features

The integration does not include DBSCAN_Cluster or Is_Anomalous_Profit. These variables were created through a separate unsupervised anomaly-analysis step in the operations notebook, but the original fitted DBSCAN helper was not available as a reusable component in the integration workflow. Rather than independently recreating clusters that might not match the source notebook, these features were left out and documented for possible reintegration before model comparison. The 3 clusters and 18 noise points reported in the operations layer notebook remain valid as a standalone unsupervised analysis; they are simply not yet part of the shared feature matrices produced by this integration step.

GSCPI Macro Enrichment

The GSCPI file from the Federal Reserve Bank of New York contains 342 monthly records from January 1998 through June 2026. For the integration, the date field was converted into a monthly key and matched with Order_YearMonth in the DataCo dataset. The merge was successful, with no missing GSCPI values in either the training or test set. This was expected because the DataCo records cover 2015 through September 2017, which falls within the GSCPI date range. The merge was completed after the train-test split, and this was still safe because GSCPI is an external published series rather than a value calculated from the project data.

GSCPI was added only to the operations layer as the Taiwanese Bankruptcy dataset does not have a reporting date or fiscal year that could be used for a reliable match.

The Week 3 analysis also showed that GSCPI may not have a strong relationship with late-delivery risk. The monthly correlation was only about -0.06 across 37 months, while the late-delivery rate stayed within a fairly narrow range of 51.9% to 56.8%. Even though the relationship was weak, GSCPI was still kept as a candidate feature so its value can be tested during Week 5 modeling instead of removing it based only on the initial correlation result.

Missing Data

The financial dataset did not have any missing values before or after cleaning, so no imputation was needed for that layer. In the operations dataset, the main missing-data issues came from Product Description, which was completely empty, and Order Zipcode, which was missing in about 86% of the records. Both columns were removed during structural cleaning because they did not add useful information for modeling. After preprocessing, all four training and test matrices were checked and confirmed to contain no missing or infinite values.

Scope Note: The Delivery-Gap Feature

The Week 4 task list included creating a feature that compares scheduled shipping time with actual shipping time. This feature was not added because actual shipping time is only known after an order has been delivered, while CEROP is intended to predict late-delivery risk when the order is placed. Late_delivery_risk also appears to be based on whether actual shipping time exceeds scheduled shipping time, which means the delivery gap would closely reproduce the target and create data leakage. For this reason, Days for shipping (real) was removed, while Days for shipment (scheduled) was retained because it is known at the time of the order.

Verification and Saved Artifacts

Each layer runs the same assertions before anything is written: train and test have identical column counts, neither contains nulls and all values are finite. The pipeline then saves ten files: processed X and y files for the training and test portions of both layers, along with the fitted financial scaler and operations preprocessor. Saving the fitted transformers matters as much as the data, because Week 5 needs the same encoder object to transform any new data and refitting it would silently produce a different feature space.

Final Dataset Summary

Layer	Rows in	Rows after cleaning	Train	Test	Features

Financial (Taiwan)	6,819	6,819	5,455	1,364	85
Operations (DataCo)	180,519	171,962	137,569	34,393	35

Table 4. Row and feature counts for both layers after preprocessing.

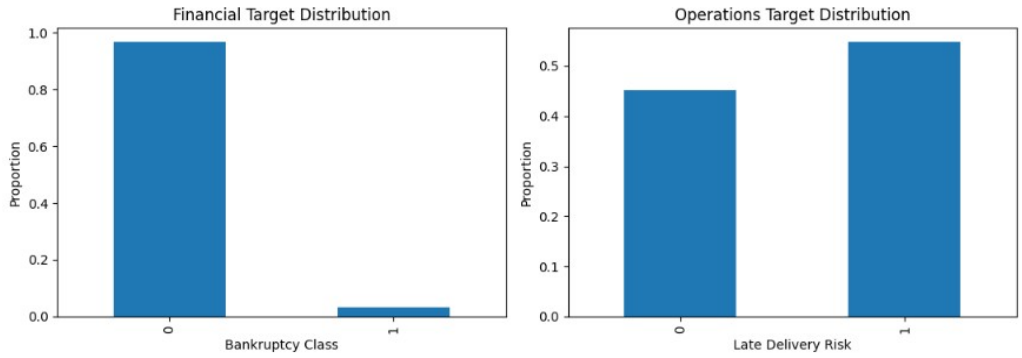


Figure 4. Target class distribution for the financial and operations training datasets.

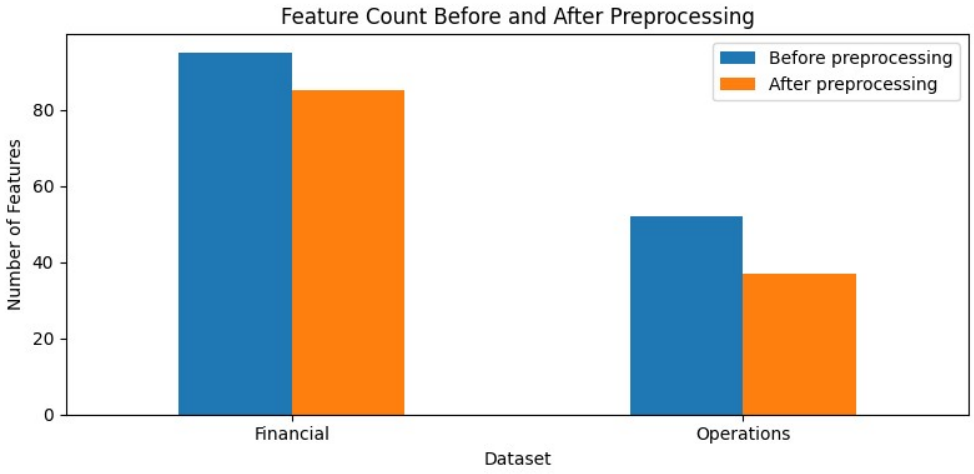


Figure 5. Feature count before and after preprocessing for both layers.

Documented Divergences From the Layer Notebooks

Three integration decisions differ from the layer notebooks. These differences are documented to clarify why the integrated results do not exactly match the layer notebooks.

Decision	Layer notebook	Integration	Reason
IQR clipping scope	Train only	Train and test, using train bounds	Keeps both matrices on one scale; bounds carry no test information
Market encoding	One-hot	Target encoded	Treated as a geographic rollup with the other geography columns
Customer Country encoding	Target encoded	One-hot	Only two levels, so one indicator column is simpler and readable

Table 5. Integration decisions that differ from the layer notebooks.

Conclusion

Both cleaned layers have been saved and are ready for modeling in Week 5. The financial layer will need extra attention because only 3.23% of the records belong to the bankrupt class. Accuracy alone may therefore be misleading, so the models should also be evaluated using precision, recall and F1-score, along with class weighting or resampling where needed. GSCPI should be tested to determine whether it adds useful information to the operations model, since the Week 3 analysis showed only a weak relationship with late-delivery risk. The target-encoded features should also be reviewed carefully during feature-importance analysis to make sure they do not overshadow more practical operational variables such as Shipping Mode. Overall, the

integration produced consistent training and test datasets for both layers while keeping the preprocessing process leakage-safe.

Scope Note: Incorporating an Unsupervised Component

During the Week 4 class, the instructor recommended to include an unsupervised learning component into the project, which is clustering (k-Means, DBSCAN) and anomaly detection. In response, the team has extended the operations risk layer to include an unsupervised anomaly detection component by using DBSCAN to applied on the order level financial features in the DataCo dataset (e.g., Benefit per order, Order Item Profit Ratio). Which has been observed on Week 3 regarding the data quality. This was noted these features on "show wide variance, including losses on some orders, which may correlate with cancellations or fraud flagged orders."

This additional updates Frame the Problem on item 4 (originally framed as purely supervised) to reflect a instructor's suggestions. The two primary classification layers remain supervised, while the new component addresses a gap those layers cannot detect anomalous transactions that do not fit either labeled risk category. Model exploration and tuning for this component will occur in Weeks 5 and 6.

References

Constante, F., Silva, F., & Pereira, A. (2019). *DataCo smart supply chain for big data analysis* [Data set]. Mendeley Data. https://doi.org/10.17632/8gx2fvg2k6.5

Federal Reserve Bank of New York. (n.d.). *Global Supply Chain Pressure Index (GSCPI)*. Retrieved July 8, 2026, from https://www.newyorkfed.org/research/policy/gscpi

Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830. https://jmlr.org/papers/v12/pedregosa11a.html

Taiwanese Bankruptcy Prediction [Data set]. (2020). UCI Machine Learning Repository. https://doi.org/10.24432/C5004D