Predicting Procurement Compliance Using KPI-Driven Machine Learning Models

Brittany M.D. Dowdle

Northwest Missouri State University
School of Computer Science and Information Systems
Maryville MO 64468, USA
S574362@nwmissouri.edu

Abstract. Compliance with procurement procedures such as timely delivery, defect-free units, and adherence to negotiated pricing is a critical driver of operational efficiency and cost control. This capstone project used a real-world dataset of 777 purchase orders spanning 2022–2023, featuring KPIs such as defect rate, price differential, order volume, and lead time, to predict supplier compliance outcomes. Data cleaning was performed that included median lead time imputation and outlier analysis. Then exploratory analysis using trend charts, histograms, and correlation heat maps. Following that, two machine learning classifiers were evaluated: Logistic Regression (AUC = 0.718) and Random Forest (AUC = 0.741). The defect rate and the price differential emerged as the strongest predictors. The calibration curves (Brier = 0.134) highlighted a mild overconfidence and the confusion matrices highlighted overfitting; this suggested the need for more advanced handling of class imbalances than a stratified split. Although class imbalance limited non-compliant detection, feature-focused modeling modestly improved minority-class recall. Future work will explore advanced resampling, temporal modeling, and optimization for minority class detection.

Keywords: detection · procurement · compliance · defect · class

1 Introduction

This project explored procurement performance using real-world data; the goal was to predict whether a purchase order would result in supplier compliance. Order compliance is critical in performance metrics. It reflects whether supplier contracts are meeting agreed-upon delivery schedules, product quality, and pricing terms. By building a predictive model based on attributes such as quantity, price deviation, or defect rates, the objective was to provide procurement teams with data-driven insights. This is useful to predict risk, improve supplier relationships, and enhance operational efficiency.

1.1 Goal of This Project

The final deliverables of this project include the following:

✓ Machine learning model predicting order compliance with supporting visualizations [6]
✓ A PDF report written in LaTeX via Overleaf
✓ A fully documented GitHub repository with Jupyter notebooks including EDA [6]

1.2 Project Resources

– **GitHub Repository:** https://github.com/Dowdle_Analytics_Capstone
– **Overleaf Report:** https://www.overleaf.com/read/bszyhdxsnrsf
– **Pro Analytics 01:** https://github.com/denisecase/pro-analytics-01
The guide used to follow a repeatable workflow for professional python projects [1]

2 Collect and Describe Data

The data set analyzed consists of 777 purchase order records with 11 original columns. Each row in the data set represents a unique transaction with attributes that relate to the identity of the supplier, the characteristics of the order, the pricing, the defects, and whether or not the supplier was compliant. The data set was publicly accessible on Kaggle; the link is in Section 2.2 Dataset Resource.

2.1 Dataset Overview

- **Data Type:** Structured (Avg of 10 key features per PO)
- **Source:** Kaggle
- **Size:** 68 KB
- **Rows:** 777
- **Columns:** 11
- **File Extension:** `.csv`
- **Tool for Ingestion:** `pandas` in Python

The data set was downloaded from Kaggle [4] and moved to the project folder. Then it was added to the GitHub project repository in the "Data" folder. Finally, it was read into three Jupyter notebooks using the pandas library (Cleaning, EDA, and Modeling).

2.2 Data Attribute Dictionary

Column Name	Description	Data Type	Example
po_id	Unique identifier for the purchase order	String	PO-10231
supplier	Supplier name or identifier	String	Supplier_A
order_date	Date the order was placed	Date	2024-01-03
delivery_date	Date the order was delivered	Date	2024-01-11
item_category	Category or type of item ordered	String	Raw Materials
order_status	Status of the order (e.g., Delivered, Pending)	String	Delivered
quantity	Quantity of units ordered	Integer	500
unit_price	Price per unit paid	Float	12.75
negotiated_price	Contractually agreed price per unit	Float	12.00
defective_units	Number of defective units in the delivery	Integer	5
compliance	Binary indicator of contract compliance	Integer	1

Table 1. Original data attributes and examples.

2.3 Domain and Professional Description

Domain: Business Operations
Subdomain: Procurement / Supply Chain

This project would be important to:
1. Supply Chain Analysts - to identify patterns in KPI metrics
2. Procurement Managers - for supplier scorecards and vendor decisions
3. Chief Purchasing Officer (CPO) - to support strategic sourcing and policy decisions

This data set falls within the field of procurement analytics. Procurement professionals use analytics to track KPIs such as on-time deliveries, cost savings, and defect rates to measure supplier performance. When a supplier consistently does not meet the expected performance level, they are a financial and operational risk. Suppliers can be considered non-compliant for late deliveries, defective products, and violations of negotiated pricing. By identifying patterns in procurement data and predicting compliance outcomes, organizations can optimize sourcing strategies, negotiate better contracts, and reduce supply-side risk. [5]

2.4 Dataset Resource

Procurement KPI Analysis Dataset:
https://www.kaggle.com/datasets/shahriarkabir/procurement-kpi-analysis-dataset

As mentioned in the Kaggle Data Card, this data set is anonymized to protect company and supplier identities and provides real-world transactions of 5 different suppliers from 2022-2023. This data set reflects challenges such as supplier delays, compliance gaps, defects, and inflationary price trends over time. It is not expected to be updated. [4]

3 Clean and Prepare Data

The raw procurement data set had to be cleaned and preprocessed to ensure consistency, accuracy, and usability. This section outlines the data cleaning and preparation steps completed in the Jupyter notebook named `cleaning.ipynb`. The goal was to ensure that it was ready for EDA in the next section. The diagram below 1 summarizes the pre-processing pipeline [3].

Fig. 1. Visual overview of the data cleaning and feature engineering process applied to procurement records (created by the author in PowerPoint).

3.1 Data Formatting and Standardization

To ensure consistency in data types and naming conventions:

– `order_date` and `delivery_date` were converted to datetime objects using

```
pd.to_datetime()
```

– All column names were standardized to lowercase letters and any white space was removed using

```
str.strip().str.lower()
```

3.2 Handling Missing Values

Missing data was identified and addressed with conditional logic using

```
df.info
```

– `defective_units` missing values were assumed to have no defects and replaced with 0 using

```
.fillna(0)
```

– Missing `delivery_date` effected 87 rows and that seemed like a significant loss of data. So, instead, the rows that had an `order_status` of Delivered were kept and used a combined median imputation and a flag column to maintain data integrity.
– Each supplier's median `lead_time_days` was calculated using

```
.dropna(subset=['lead_time_days']).groupby('supplier')['
    lead_time_days'].median()
```

- The column to flag missing delivery date was created using

```
df['delivery_date'].isnull() & (df['order_status'].str.
    lower() == 'delivered')
```

- The column to flag missing delivery date was converted to an integer using

```
.astype(int)
```

- Imputed `delivery_date` used `order_date` + median lead time per supplier.
- 19 rows still had missing `delivery_date` after this and had an `order_status` of `"cancelled"`, `"pending"`, or `"partially delivered"`. They were removed because they represent incomplete transactions and could introduce bias or noise into the model.

3.3 Feature Engineering

Two more attributes were created to allow for a deeper analysis of procurement efficiency and quality because they will support more meaningful comparisons between suppliers and orders. These attributes were selected as independent variables for the model. The dependent variable for the model was `compliance`.

- `price_diff` calculates the difference between `unit_price` and `negotiated_price`.
- `defect_rate` calculates the proportion of `defective_units` relative to the `quantity` ordered.

3.4 Outlier Detection and Treatment

Outliers were identified using the IQR method for the following 4 attributes: `quantity`, `unit_price`, `price_diff`, and `defect_rate`. Outliers were identified in 15 of 777 rows. This was equal to 1.93% and was assumed to be a natural variance. The treatment decided for these rows was to leave as is.

3.5 Exporting Cleaned Dataset

The cleaned data set contained 15 attributes and 758 records. It was exported as a CSV file using

```
df.to_csv()
```

It is saved as `cleaned_procurement_data.csv` in the project repository Data folder. The image below 2 shows the first five rows of the cleaned dataset as output from the Jupyter notebook [2]. This preview was generated using:

```
print(df.head())
```

	po_id	supplier	order_date	delivery_date	item_category
0	PO-00001	Alpha_Inc	2023-10-17	2023-10-25	Office Supplies
1	PO-00002	Delta_Logistics	2022-04-25	2022-05-05	Office Supplies
2	PO-00003	Gamma_Co	2022-01-26	2022-02-15	MRO
3	PO-00004	Beta_Supplies	2022-10-09	2022-10-28	Packaging
4	PO-00005	Delta_Logistics	2022-09-08	2022-09-20	Raw Materials

	order_status	quantity	unit_price	negotiated_price	defective_units
0	Cancelled	1176	20.13	17.81	0.0
1	Delivered	1509	39.32	37.34	235.0
2	Delivered	910	95.51	92.26	41.0
3	Delivered	1344	99.85	95.52	112.0
4	Delivered	1180	64.07	60.53	171.0

	compliance	lead_time_days	delivery_date_missing_flag	price_diff
0	Yes	8.0	0	2.32
1	Yes	10.0	0	1.98
2	Yes	20.0	0	3.25
3	Yes	19.0	0	4.33
4	No	12.0	0	3.54

	defect_rate
0	0.000000
1	0.155732
2	0.045055
3	0.083333
4	0.144915

Fig. 2. First five rows of the cleaned procurement dataset, shown after all transformations confirming standardized formats, added features, and imputation results (created by the author in VSCode).

4 Exploratory Data Analysis (EDA)

EDA was performed using Python libraries including `pandas`, `seaborn`, and `matplotlib`. The goal was to understand the structure, distribution, and relationships in the data set. All supporting code and visualizations are available in the Jupyter notebook named EDA.ipynb to identify correlations useful for modeling supplier compliance.

4.1 Trend Line Charts

Time-series line plots were created to explore the patterns for price difference and defect rate. The visualizations revealed when supplier performance deviated from expectations and highlighted potential predictors of compliance.

- A line graph of the quarterly average `price_diff` over time reflected more contract violations in late-year periods, indicating a potential seasonal trend.

– A line graph of the quarterly average `defect_rate` showed spikes in Q3 and Q4, suggesting quality degradation later in the year.

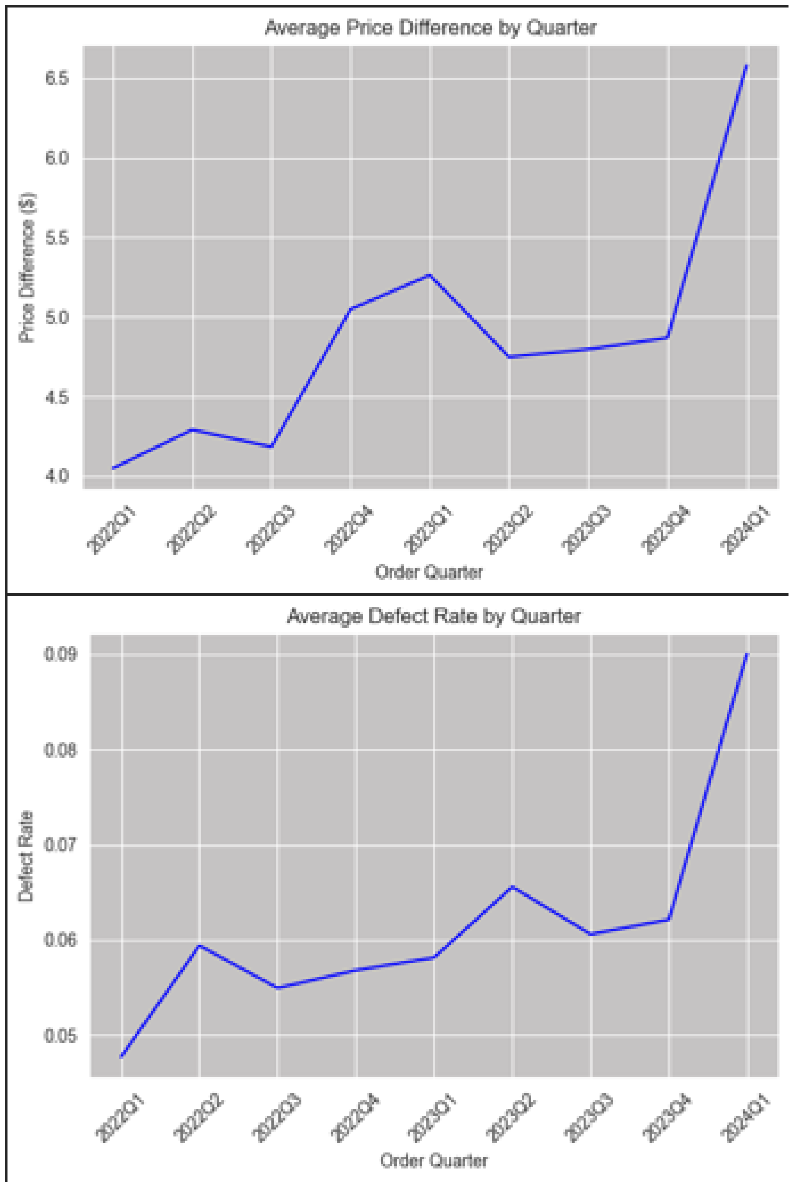

Fig. 3. Average price differential (top) by quarter peaks in Q4, signaling year-end negotiation breakdowns. Average defect rate (bottom) by quarter shows dips in quality in Q3-Q4, indicating when to investigate supplier processes (created by the author in VSCode).

4.2 Numerical Column Distributions

The distributions of the numeric attributes were plotted using histograms to assess skewness and any further transformation needs. In the previous section, outliers were determined to be well below 10% of the total rows and will remain in the set for modeling.

Table 2. Distribution Patterns of Numeric Features

Feature	Distribution Type	Comment
`quantity`	Slightly right-skewed	Long tail with some large orders near 5000 units.
`unit_price`	Uniform	Values spread across the full range with no clear peak.
`negotiated_price`	Slightly bimodal	Highest values in both of the end ranges.
`defective_units`	Strong right-skewed	Most orders have few defects; rare cases with over 300.
`price_diff`	Right-skewed	Most orders have small differences; large deviations are uncommon.
`defect_rate`	Strong right-skewed	Defect rate close to zero for most; some high outliers.
`lead_time_days`	Mild right-skewed	Clustered around 10–12 days with moderate variability.

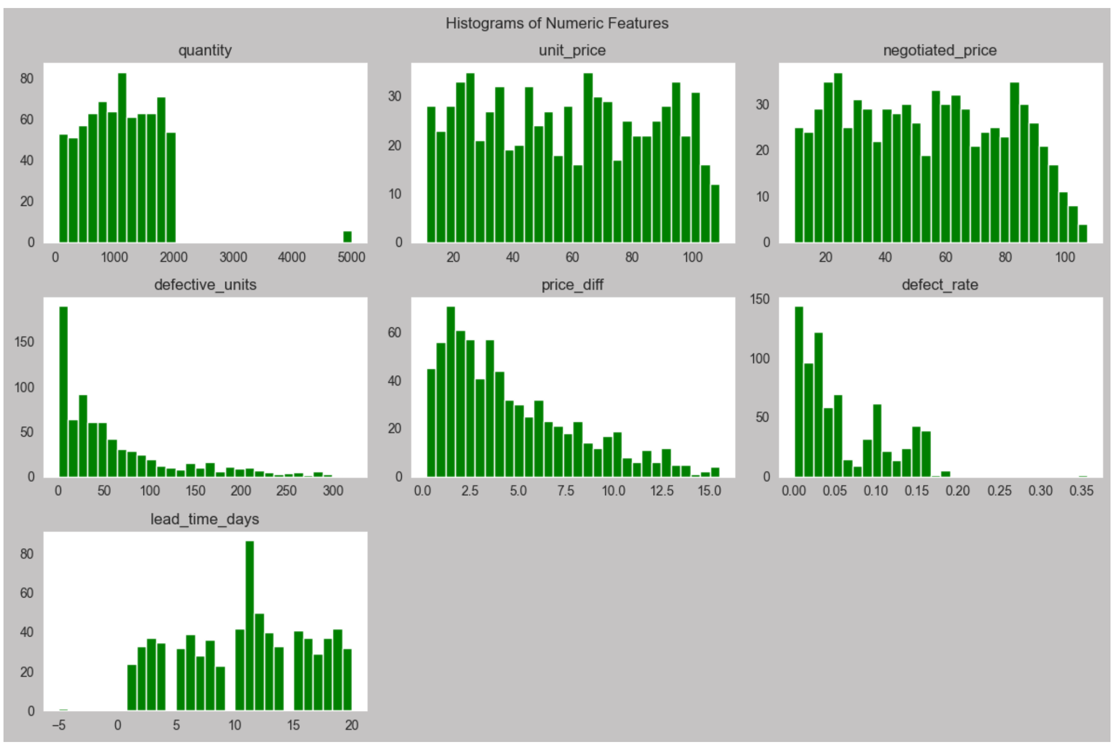

Fig. 4. Distributions of numerical columns like unit price with a uniform spread, defect rate with a heavy right skew, and lead time days clustering at 10-12. Procurement should monitor large order outliers separately and consider log-transforming defect rate to help stabilize the variance (created by the author in VSCode).

4.3 Categorical Column Distributions

Distributions were analyzed using count plots and value frequencies. Delta Logistics and Epsilon Group handled the most orders. Office Supplies peaked as the most ordered category. More than 500 orders have a Delivered status. And just over 80% of the orders were compliant, which indicates how the data set will need to be split to use in the model in the next section.

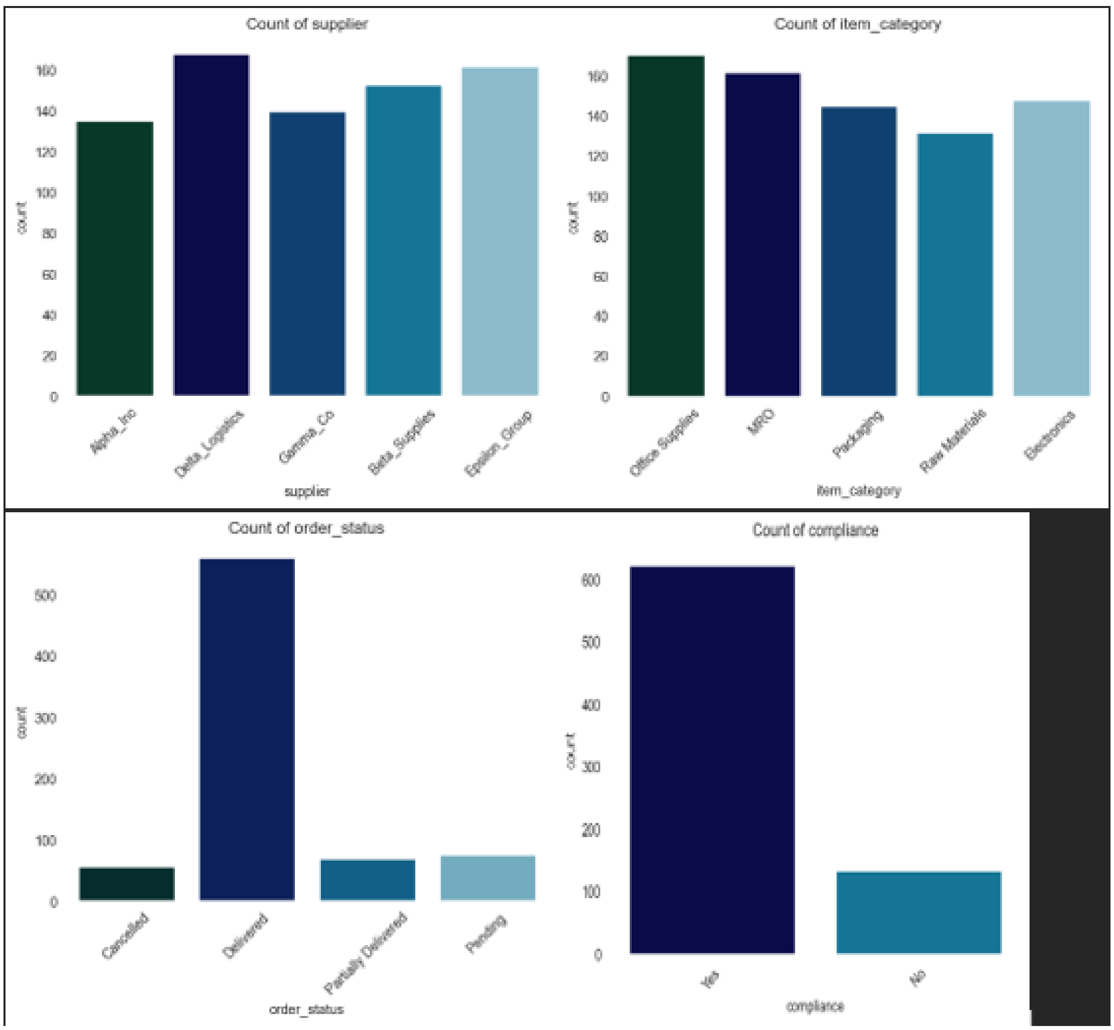

Fig. 5. Count plots of categorical column frequencies using bar charts, highlighted Delta Logistics handled 250 orders, suggesting priority for vendor scorecards. Order status shows 85% delivered, indicating limited non-compliance data. And identified class imbalance will need handled before modeling (created by the author in VSCode).

4.4 Correlation Matrix

A heatmap visualization was created to examine the relationships between numeric attributes and their potential predictive value for the binary compliance variable. A mask was used to highlight the column with correlations with `compliance`.

- `defect_rate` had the strongest (negative) correlation with compliance.
- The rest of the attributes had weak positive correlations and were unlikely to drive the performance of the model individually.
- `quantity` was not considered a high predictor but was included in the model due to this section.

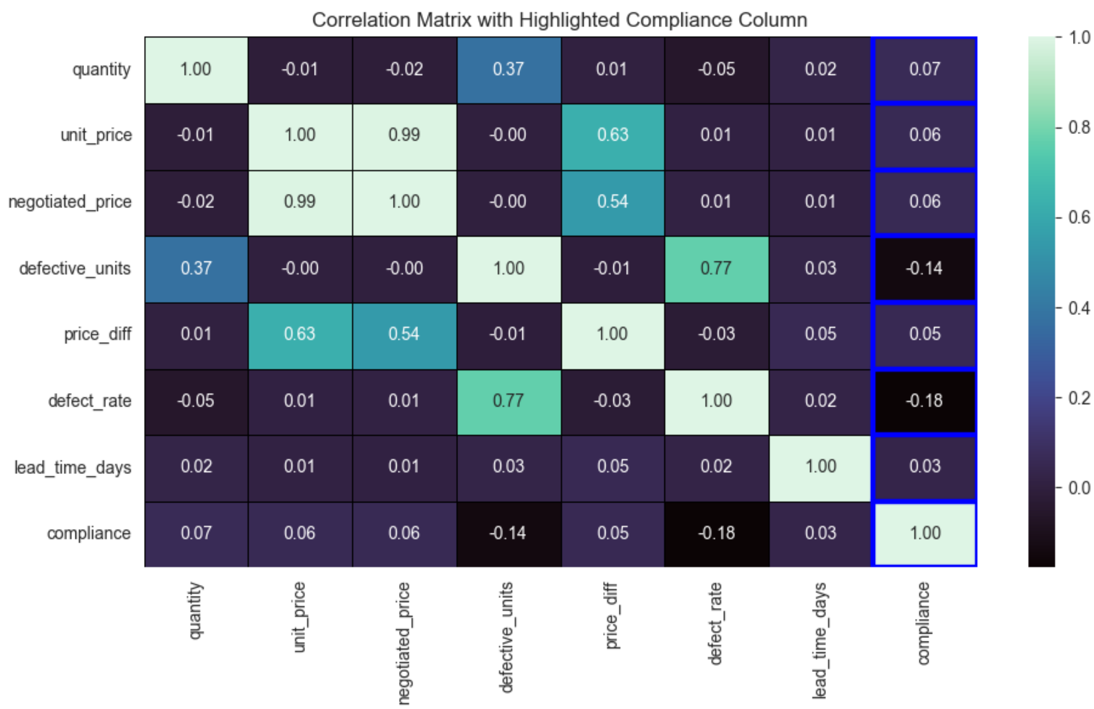

Fig. 6. Heatmap showing pairwise correlation coefficients among numeric features. The strongest correlations are negative, defect rate (-0.18) and defective units (-0.14). Price diff (0.05) offers a secondary signal for cost containment efforts (created by the author in VSCode).

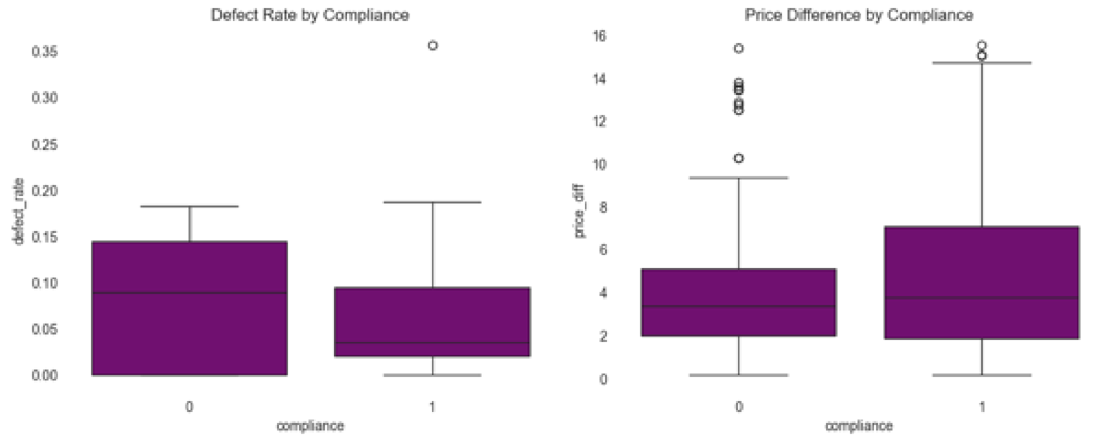

Fig. 7. Box Plots of attributes with highest correlation to compliance. Both metrics demonstrate clear patterns that support their use as predictive features, with non-compliant orders consistently showing higher defect rates and greater price volatility (created by the author in VSCode).

4.5 Vendor Comparisons

Bar plots and aggregated statistics were used to compare vendor performance. Grouped by `supplier`, average values were calculated and visualized for key metrics. Delta Logistics had the highest average defect rate, suggesting quality concerns. Beta Supplies had the longest average lead time. And Alpha Inc showed the largest average price overcharge despite low defect rates. These comparisons offer procurement teams a basis for supplier evaluation and suggest which vendors may require closer monitoring.

Table 3. Average Supplier KPIs

Supplier	Avg Defect Rate	Avg Lead Time (Days)	Avg Price Diff ($)
Alpha	0.02	10.2	4.8
Beta	0.08	10.7	4.6
Delta	0.10	10.4	4.3
Epsilon	0.03	10.4	4.7
Gemma	0.04	10.0	4.5

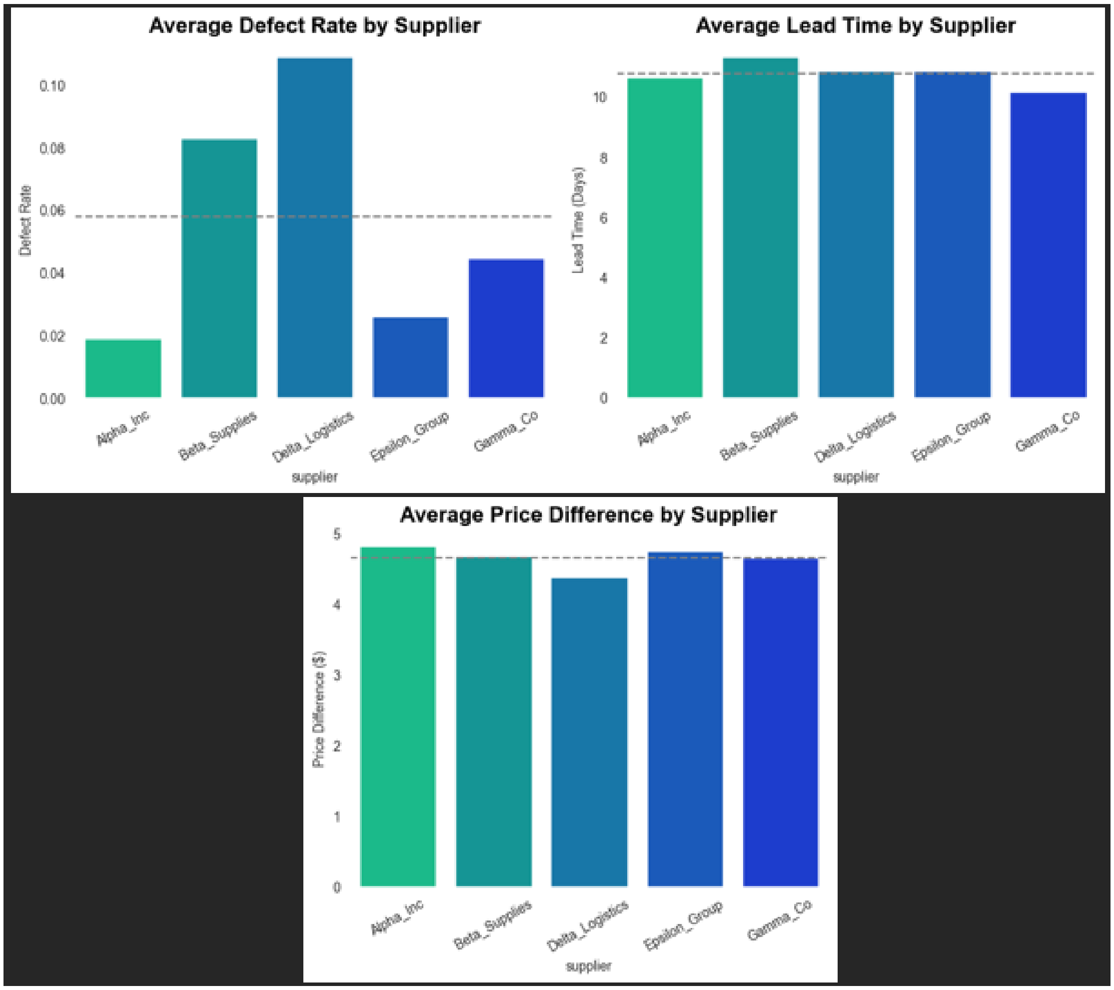

Fig. 8. Bar Charts comparing average Defect Rate, Lead Time Days, and Price Diff by supplier. Delta Logistics showed the highest defect rate (0.10) and Alpha Inc's price diff (4.8) signals negotiation leverage potential in contract renewals (created by the author in VSCode).

5 Model and Generate Insights

This section details the predictive modeling workflow, hyperparameter tuning, and performance evaluation performed to forecast procurement order compliance. All model code and results are available in the modeling.ipynb notebook in the project repository.

5.1 Data Preparation and Split

Due to the previous section showing a large class imbalance, a stratified train-test split was applied using `StratifiedShuffleSplit` to preserve the distribution of the binary target `compliance` (encoded 1 = Yes, 0 = No). The dataset (n=758) was split (80/20) into a training set with 606 records and a testing set with 152 records.

```
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
    random_state=42)
for train_idx, test_idx in split.split(df[categorical_cols +
    numeric_cols], df['compliance']):
     X_train = df.iloc[train_idx][categorical_cols +
         numeric_cols]
     y_train = df.iloc[train_idx]['compliance']
     X_test = df.iloc[test_idx][categorical_cols +
         numeric_cols]
     y_test = df.iloc[test_idx]['compliance']
```

5.2 Models Evaluated

Two classification algorithms were compared:

– **Logistic Regression** – provides a linear baseline with interpretable coefficients.
– **Random Forest Classifier** - captures nonlinear interactions and reports feature importance.

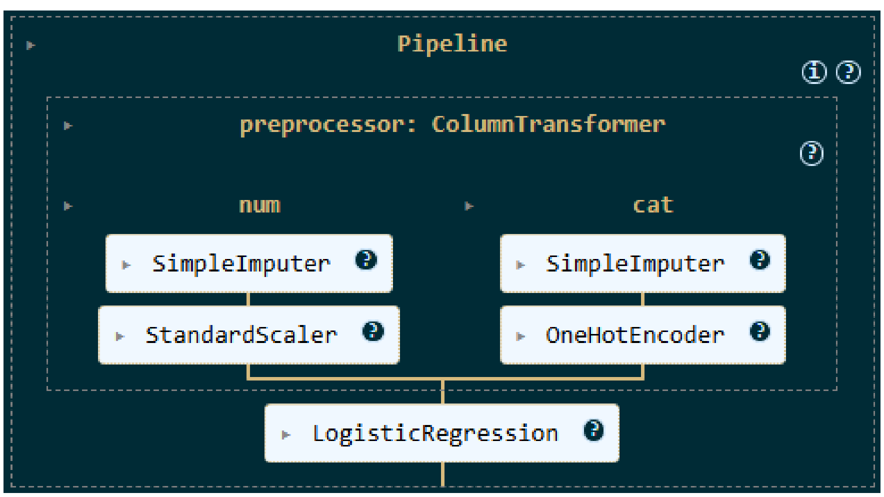

Fig. 9. Pipeline architecture for preprocessing and logistic regression modeling. (created by the author in VSCode.)

5.3 Hyperparameter Tuning

A `Pipeline` combining preprocessing (imputation, scaling, one-hot encoding) and classification was tuned via `GridSearchCV` (5-fold cross validation) to maximize the F1 score. Both models were set up using the same preprocessor and classifier in the code below. The selected Random Forest parameters were:

```
rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])
# Set parameters
rf_params = {
    'classifier__n_estimators': [100],
    'classifier__max_depth': [5],
    'classifier__min_samples_split': [4]
}
```

5.4 Evaluation Metrics

Both models were evaluated on the test set using the following:

- **Accuracy** for overall correctness.
- **Precision/Recall** for class specific performance.
- **F1 Score** for mean of precision and recall.
- **ROC AUC Score** for ranking quality across thresholds.

The random forest ROC AUC of 0.741 outperformed logistic regression's 0.718 by 0.023, which suggested a slightly better ability to distinguish between classes. Logistic regression had better recall and accuracy, but this was due to predicting only the majority class.

Model	Label	Precision	Recall	F1-Score	Support	ROC AUC
Logistic Regression (all features)	0	0.00	0.00	0.00	27	0.718
	1	0.82	1.00	0.90	125	
	Accuracy			0.82	152	
	Macro Avg	0.41	0.50	0.45	152	
	Weighted Avg	0.68	0.82	0.74	152	
Random Forest (all features)	0	0.00	0.00	0.00	27	0.741
	1	0.82	0.98	0.89	125	
	Accuracy			0.81	152	
	Macro Avg	0.41	0.49	0.45	152	
	Weighted Avg	0.67	0.81	0.74	152	

6 Present Results

This section consolidates the key outcomes of the predictive modeling effort, illustrates the performance of the model with tables and charts, and explores additional experiments to deepen the insights. All figures can be found in the Visualizations folder of the project repository.

6.1 Calibration Curve Analysis

Both models show reasonable calibration in the higher probability bins, with the Random Forest curve exhibiting more variance. These are moderately low scores, which indicate that both models are reasonably calibrated overall. The majority of both models are overconfident and would benefit from further tuning or calibration methods to improve reliability in uncertain regions.

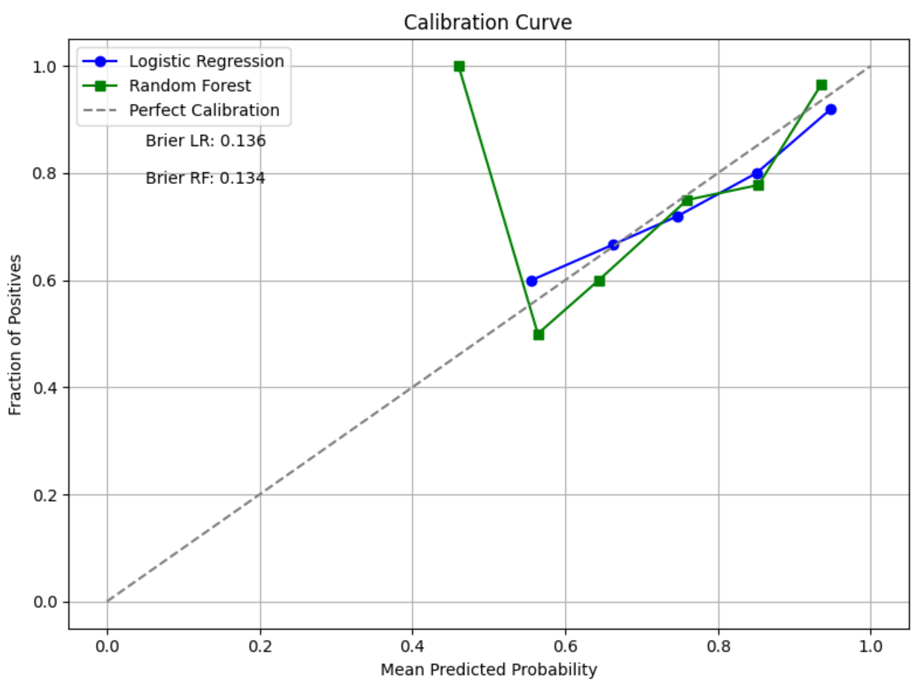

Fig. 10. Calibration Curves for Logistic Regression and Random Forest. Brier scores indicate the overall accuracy of probabilistic predictions, with lower values being better. Random Forest had a slightly better Brier score (0.134) compared to Logistic Regression (0.136).

6.2 Confusion Matrix Analysis

Both models classified all or nearly all instances as Class 1, demonstrating complete inability to identify non-compliant orders. All 27 non-compliant orders were predicted as false positives. This pattern underscores the extreme class imbalance and indicates that additional strategies are needed to recover accurate predictions of the minority class.

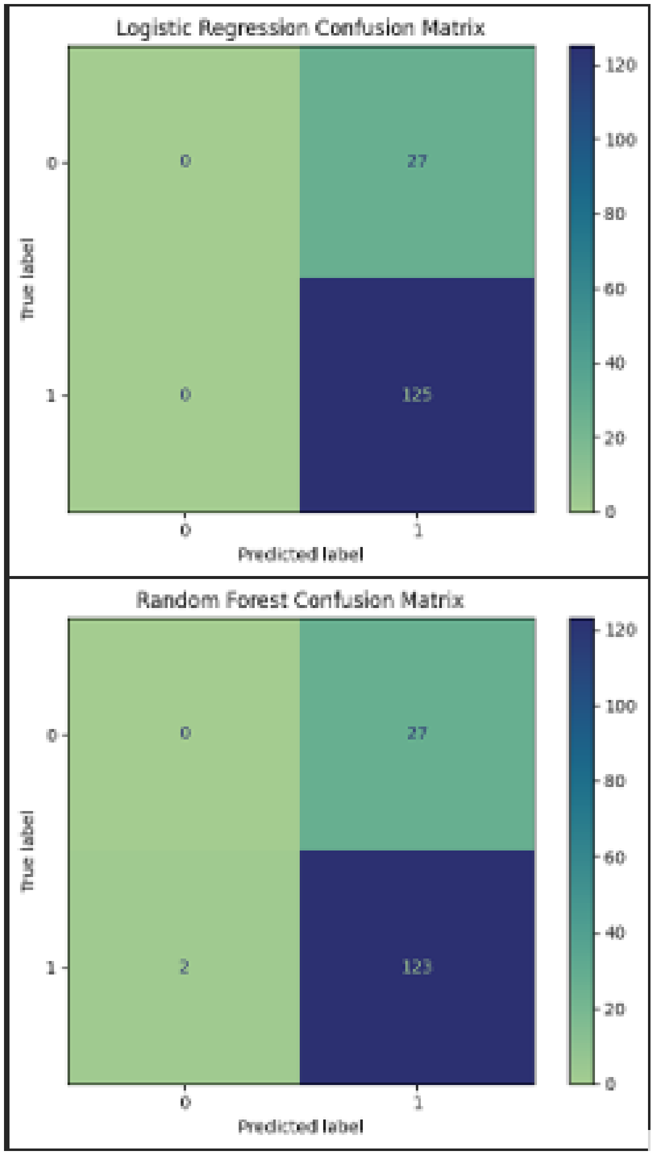

Fig. 11. Confusion matrices for Logistic Regression (top) and Random Forest (bottom on the test set. The models are essentially "blind" to non-compliant orders (created by the author in VSCode).

6.3 ROC Curve Analysis

ROC curves quantify the discrimination of each model. Although both curves lie above the diagonal, the modest AUCs reflect the challenge of separating compliant from non-compliant purchases based solely on the selected KPIs. Random Forest scored slightly higher here, which differs from the classification report results.

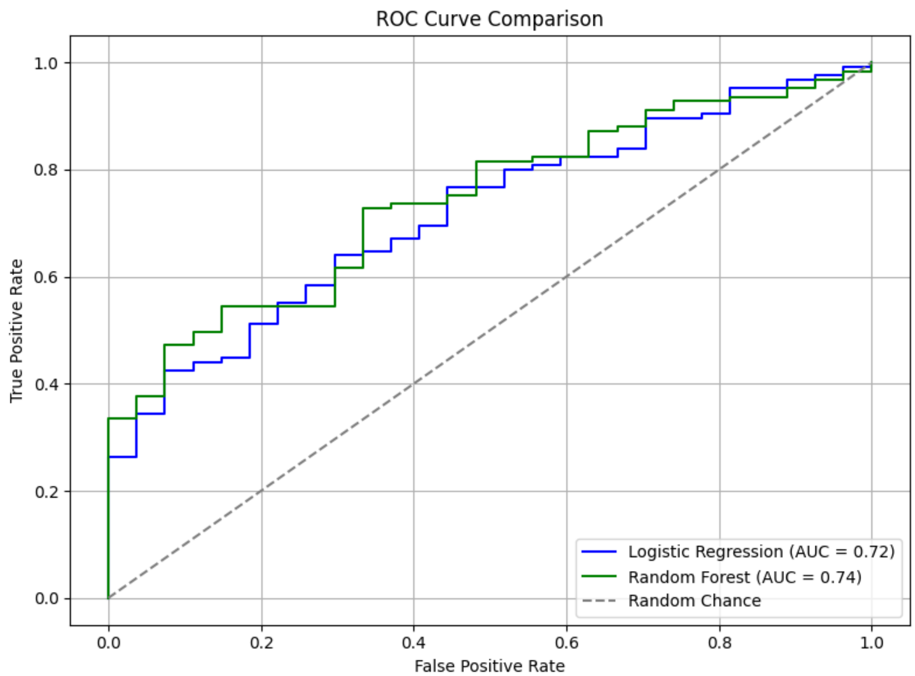

Fig. 12. ROC curve comparison: Logistic Regression AUC = 0.72, Random Forest AUC = 0.74 (created by the author in VSCode).

6.4 Feature Importance Analysis

Random Forest feature importance revealed the top predictive variables were:

1. **defect_rate** - proportion of defective units. (Next highest is defective_units but it would be redundant to include in the model)
2. **price_diff** - negotiation effectiveness
3. **quantity** - order volume
4. **lead_time_days** - delivery timeline

A follow-up experiment trained the models using only these selected features to test whether dropping weaker variables improved performance.

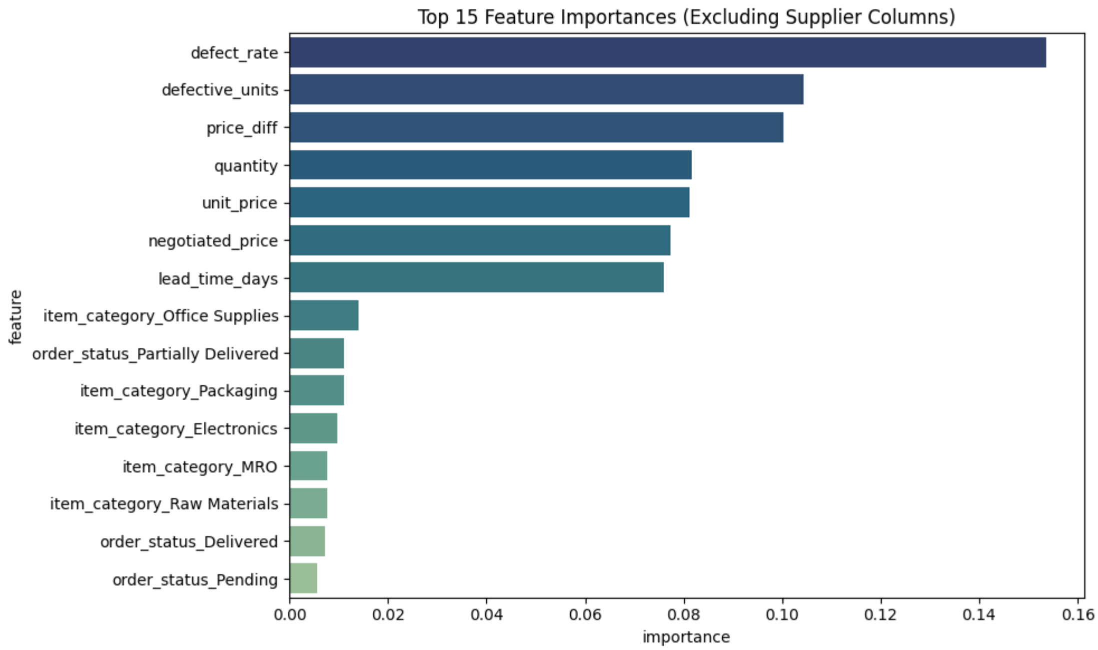

Fig. 13. Top 15 Feature Importances from the Random Forest Classifier. The most predictive features include `defect_rate`, `defective_units`, and `price_diff`, which align with prior exploratory analysis.

6.5 Dimensionality Reduction Experiment

The same models were re-trained using only the top features (defect_rate, price_diff, quantity, and lead_time_days). Both models show deviations from perfect calibration, particularly at the extremes. The Logistic Regression model tends to be *overconfident*, predicting probabilities closer to 1 even when the observed frequency is lower. The Random Forest model also exhibits pockets of *overconfidence*, but with more variability between bins. Random forest was able to predict a single non-compliant order, which was better than the original score of zero. The dramatic performance degradation, particularly for Logistic Regression, suggested that feature interactions and the full feature space are critical for model performance.

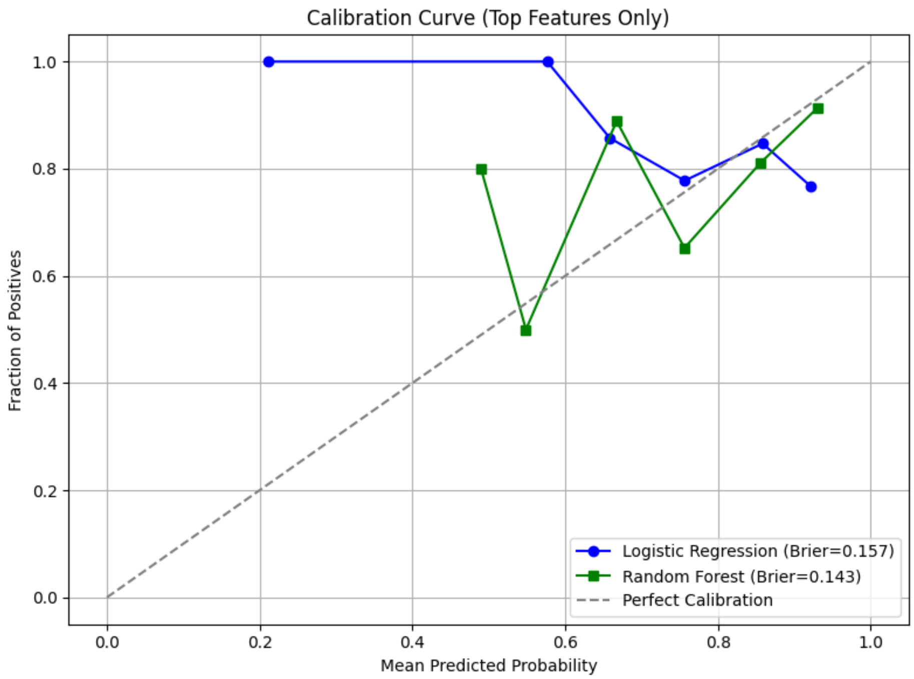

Fig. 14. The Brier score using only the top indicators for Random Forest (0.143) was lower than for Logistic Regression (0.157), indicating slightly better probability calibration. A perfectly calibrated model would lie along the dashed diagonal.

Model	Original AUC	SF AUC	Performance Change
Logistic Regression	0.718	0.468	-0.25
Random Forest	0.741	0.671	-0.07

Table 4. AUC score comparison with original model using all features and SF model using dimensionality reduction.

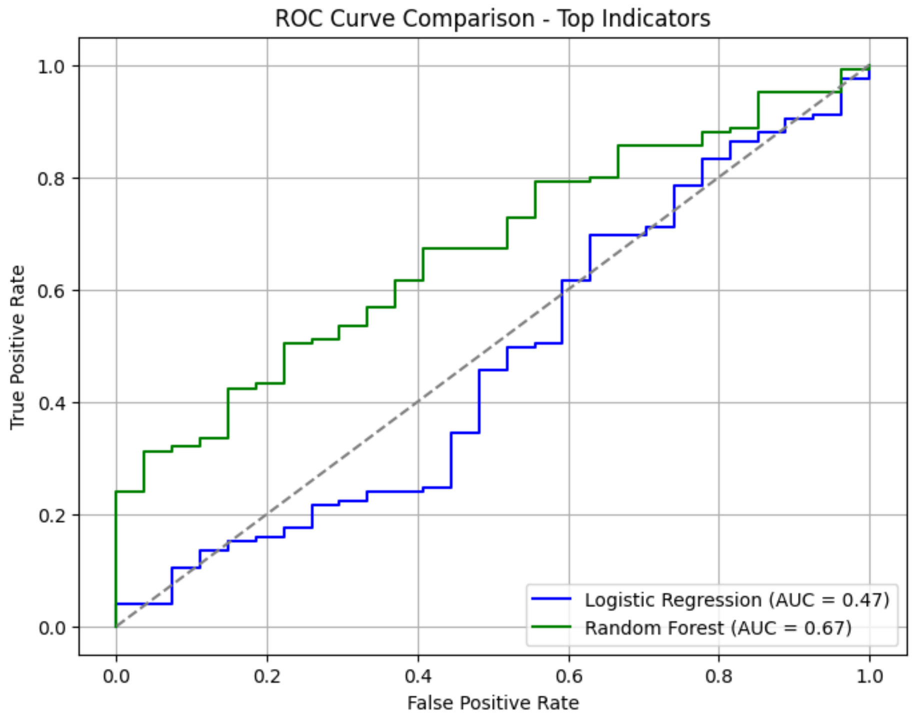

Fig. 15. ROC Curve for Models Using Only Top Indicators: `defect_rate`, `price_diff`, `quantity`, and `lead_time_days`. Significant performance degradation on the Logistic Regression model.

7 Final Deliverables

All final deliverables identified in the introduction of this report have been completed.

7.1 GitHub Repo Completeness

1. Structure: The repository contains clear top-level folders for Data/, Notebooks/, and Visualizations/.
2. README.md: Introduces the project objectives, data source links, environment setup instructions, and clickable links to both the Overleaf PDF, individual notebooks, and all visualizations.
3. Commit History: Chronologically documents data ingestion, cleaning, EDA, and modeling milestones.
4. Artifacts: Includes raw and cleaned CSVs, .ipynb notebooks, and a readable version of the report.

7.2 OverLeaf Report Completeness

1. Sections: The seven modules are presented with detailed prose, figures, tables, and code excerpts.
2. Navigation: Hyperlinked contents and clickable URLs to GitHub and Kaggle.
3. References: Complete @misc citations for all online resources and in-text Figure/Table citations.
4. Formatting: Consistent typography, captioned visuals, and professional layout free of grammar or spelling errors.

7.3 Limitations

- Class Imbalance: Resulting in zero true negatives, this severely limits the ability to detect the minority class, and the mathematically optimal strategy inflates overall accuracy.
- Default Algorithm Parameters: Standard configurations prioritize majority class performance, and standard regularization techniques do not address class imbalance directly.
- Feature Engineering: Limited domain-specific feature creation may have missed important patterns. The delivery date dates were imputed using the median lead time per supplier. Although this preserved more records, it may introduce bias if true lead time variability is high.
- Temporal Aspects: The analysis did not consider time-series patterns in compliance behavior. Covers a limited period, and any changes in supplier performance outside that window cannot be captured, reducing the long-term applicability of the model.

7.4 Future Work

- Imbalance Handling: SMOTE (synthetic minority oversampling) or ensemble methods such as BalancedRandomForest to improve the detection of non-compliant orders without sacrificing precision on the majority class.
- Parameter Settings: Modify the parameters to call out `class_weight='balanced'` or lowering the probability threshold below 0.5 could improve precision/recall on class 0.
- Custom Loss Functions: Implement specialized functions like Focal Loss or Weighted Cross-Entropy that explicitly optimize for minority class detection.

Peer reviews provided the feedback to implement one of the future work options to strengthen the model's real-world applicability. The Jupyter notebook PeerReview.ipynb contains the work and comparison of the original Random Forest Model. The following is a table of the comparison:

Table 5. Comparison of Random Forest vs. Balanced Random Forest Performance

Model	Class	Precision	Recall	F1-Score	Support
Random Forest	0	0.00	0.00	0.00	27
	1	0.82	0.98	0.89	125
	Accuracy	-	-	**0.81**	152
	Macro Avg	0.41	0.49	0.45	-
	Weighted Avg	0.67	0.81	0.74	-
	ROC AUC	-	-	**0.741**	-
	Brier Score	-	-	**0.134**	-
Balanced RF	0	0.39	0.56	0.46	27
	1	0.89	0.82	0.85	125
	Accuracy	-	-	**0.77**	152
	Macro Avg	0.64	0.69	0.66	-
	Weighted Avg	0.81	0.77	0.78	-
	ROC AUC	-	-	**0.746**	-
	Brier Score	-	-	**0.166**	-

The balanced-class Random Forest model achieved a substantial improvement in identifying non-compliant orders (class 0) while maintaining strong overall performance on compliant orders (class 1). Precision for class 0 rose to 0.39, Recall to 0.56, and F1-score to 0.46. The original model all scored 0. Out of 27 non-compliant orders, 15 were correctly identified by applying class weighting.

References

1. Case, D.: Git add, commit, push guide. https://github.com/denisecase/pro-analytics-01/blob/main/03-repeatable-workflow/06-git-add-commit-push.md (2023), accessed: 2025-06-29
2. Dowdle, B.: Cleaned dataset preview from jupyter notebook (2025), created using `df.head()` in VSCode
3. Dowdle, B.: Cleaning workflow diagram (2025), created in Microsoft PowerPoint
4. Kabir, S.: Procurement kpi analysis dataset. https://www.kaggle.com/datasets/shahriarkabir/procurement-kpi-analysis-dataset (2022), accessed: 2025-06-23
5. Team, T.E.: Why procurement is a good career: Key reasons to consider this path. https://www.techneeds.com/2025/04/05/why-procurement-is-a-good-career-key-reasons-to-consider-this-path/ (April 2025), accessed: 2025-06-29
6. Team, U.E.: Top 100 data science project ideas for beginners. https://www.upgrad.com/blog/data-science-project-ideas-topics-beginners/ (2024), accessed: 2025-06-23