Graph-Based Features for Late-Delivery Risk Prediction in Supply Chains: A Leakage-Controlled Benchmark

Amna Riaz

Computer Science, University of Sargodha, Sargodha, Pakistan

DOI: https://doi.org/10.51244/IJRSI.2026.1313CS031

Received: 08 October 2026; Accepted: 13 October 2026; Published: 24 October 2026

ABSTRACT

Predicting whether an order will be delivered late is a core task in supply chain analytics, and the public DataCo Smart Supply Chain dataset is widely used as a benchmark for it. Graph-based features and models are increasingly proposed to improve such predictions, but the dataset contains columns that leak the label and rows that share an order, which can inflate reported performance. This study asks whether hand-crafted graph features, appended to a tabular XGBoost model, improve late-delivery risk prediction once leakage is controlled. We audit the dataset, remove cancelled and fraud orders, and collapse 180,519 item rows into 62,897 orders. We build a heterogeneous graph of customers, products, categories, cities, countries and regions (24,268 nodes, 165,059 edges) with NetworkX, derive structural features, spectral embeddings and fold-safe label-aware encodings (a form of target encoding over graph neighbours), and evaluate them with XGBoost under order-level repeated cross-validation, a temporal split, Holm-corrected paired tests and SHAP. Leakage matters: a leaky column gives an AUC of 1.00, and a random row split overstates AUC by 0.024 (0.7985 versus 0.7749). Without leakage, the label is governed almost entirely by shipping mode and, for Same Day orders, by whether the order is placed after noon. A four-line lookup rule reaches an AUC of 0.774, and Standard and Second Class outcomes are indistinguishable from chance for every model (within-mode AUC about 0.50). Graph features add a nominal 0.003 AUC over XGBoost (uncorrected $p = 0.049$) that does not survive Holm correction ($p = 0.146$), cannot be distinguished from the rule (0.001, $p = 0.445$), and reverses on a temporal split (-0.005 against the rule). Entity-level late rates do not differ from shuffled labels. On DataCo, hand-crafted graph features therefore add no reliable signal; end-to-end graph neural networks were not tested. The whole graph pipeline costs about 5 s and under 1 GB of memory, so the problem is absence of signal, not cost. We recommend order-grouped, temporal evaluation with a rule baseline.

Keywords: late-delivery prediction; graph features; data leakage; supply chain analytics; XGBoost

INTRODUCTION

Late deliveries cost retailers money, damage customer trust and force expensive expediting. Predicting which orders will arrive late before they ship lets operations teams intervene early. The DataCo Smart Supply Chain dataset [1], with 180,519 order-item records and 53 columns, has become a common public benchmark for this task, and recent studies report strong results with tree ensembles [2] and graph neural architectures [3].

Two properties of this dataset make such results hard to interpret. First, it contains columns that are recorded only after shipment, including the actual shipping days and the delivery status, and the label can be recomputed from them. A model that uses them is not predicting anything. This is the textbook case of target leakage [4]. Second, one order is spread over up to five item rows that share a single label, so a random split over rows puts near-copies of the same order into both training and test sets and overstates accuracy, a known problem for data with grouped structure [5].

Graph-based features are attractive here because supply chains are networks of customers, products and places. Whether a graph adds information beyond a flat table has to be tested against a leakage-free baseline, and this paper does that.

We ask a narrow question: once leakage is removed, do hand-crafted graph features (structural statistics, spectral embeddings and label-aware encodings), appended to a tabular gradient-boosted model, improve late-delivery risk prediction on DataCo? We do not test end-to-end graph neural networks, which could in principle learn relational structure that hand-crafted features miss; Section 7 states this scope. Our contributions are:

1. A leakage audit of the dataset that quantifies how much each source of leakage inflates AUC.
2. A rule-based characterisation of the label: shipping mode, and for Same Day orders the order hour, explain all of the learnable signal.
3. A graph-feature ablation (structural features, spectral embeddings and fold-safe label-aware encodings) with 10-repeat cross-validation, Holm-corrected paired tests, a temporal split and a permutation test of entity effects.
4. A reproducible notebook that regenerates every number using only free tools, with no large language model and no graph database, and a measurement of the computational cost of each stage.
5. A checklist for practitioners deciding whether graph methods are worth trying on a supply chain prediction task (Section 6).

Related Work

Late-delivery prediction on DataCo. Dwivedi et al. [2] compare machine learning, statistical and time-series models on DataCo for several supply chain tasks and report that random forest, XGBoost and gradient boosting perform best for late delivery risk. Xue et al. [3] propose EAGLE, which combines a temporal encoder with an edge-aware graph attention network over 46 geographic nodes and reports an F1 of 0.876 and an AUC of 0.977. Their target differs from ours: a node is positive when its next-window average delay exceeds its own historical mean, and past realised delays are used as inputs, so their numbers are not comparable with per-order lateness. They also remove delivery status and actual shipping days as leakage, which we regard as good practice. Our study keeps the per-order label of the public file and asks the more basic question of whether graph features help at all.

Knowledge graphs and graph features. A knowledge graph represents entities and their relations as a graph and supports analytics such as link prediction and embedding [6]. For tabular prediction, graph information can enter as structural statistics (centrality, community membership), as low-dimensional embeddings, or as aggregates of neighbours' labels. We evaluate all three families. We use gradient-boosted trees as the learner because trees remain a strong default for medium-sized tabular data [7,8]. We separate these into two kinds. Structural statistics and spectral embeddings use graph structure only. Label-aware encodings aggregate labels over graph neighbours and are a form of target encoding [13] whose groups are graph nodes; they are different in kind, can leak if not computed out of fold, and are therefore reported separately in this paper.

Leakage and evaluation. Kaufman et al. [4] define leakage as the use of information that would not be legitimately available at prediction time and discuss how it produces overly optimistic models. Roberts et al. [5] show that random cross-validation is optimistic when observations are grouped or ordered in time, and recommend blocked schemes. We apply both ideas: order-grouped folds, a temporal split, and a simple rule as the baseline against which every model must be judged.

3. Data and Leakage Audit

3.1 Dataset. DataCo Smart Supply Chain [1] (Mendeley Data, version 5, 12 March 2019) records the provisioning, production, sales and distribution of a company selling clothing, sports and electronic supplies. We use the structured file, which has 180,519 order-item rows and 53 columns for orders placed between January 2015 and January 2018. The target, Late_delivery_risk, equals 1 when an order arrived later than scheduled. An order has 1 to 5 item rows (mean 2.75), and all items of an order share the same shipping mode, customer and label (no exception among 65,752 orders).

3.2 Sources of leakage. We found five issues, summarised in Table 1.

Table 1. Leakage sources in the public DataCo file and how each is handled.

Source	What it does	Treatment
Days for shipping (real)	For every shipped order, the label is 1 exactly when real days exceed scheduled days	Excluded from features
Delivery Status	Encodes the label directly ("Late delivery" is 1, every other status is 0)	Excluded from features
Shipping date	Known only after shipment; with the order date it gives the real shipping days	Excluded from features
Order Status CANCELED or SUSPECTED_FRAUD	7,754 item rows (2,855 orders) with label always 0, because nothing was delivered	Rows removed
Several item rows per order	Items of one order share a label, so random row splits place copies of an order in both train and test	One row per order, order-grouped folds

We also dropped columns that are masked or empty (customer e-mail, password, product description), a zip code that is 86% missing, and exact duplicates of other columns. Customer names and street addresses were not used.

3.3 Analysis table. After removing cancelled and fraud orders and collapsing items, 62,897 orders remain, of which 57.3% are late.

3.4 What determines the label. Shipping mode explains most of the label, and for Same Day orders the order hour explains the rest (Table 2, Figure 1). First Class orders are always late. Same Day orders are late exactly when placed at or after 12:00. Second Class and Standard Class orders are late at constant rates of 80.0% and 39.8%.

Table 2. Late-delivery rate by shipping mode (62,897 orders).

Shipping mode	Orders	Late rate
First Class	9,602	100%
Same Day, ordered before 12:00	1,759	0%
Same Day, ordered at or after 12:00	1,648	100%
Second Class	12,256	80.0%
Standard Class	37,632	39.8%

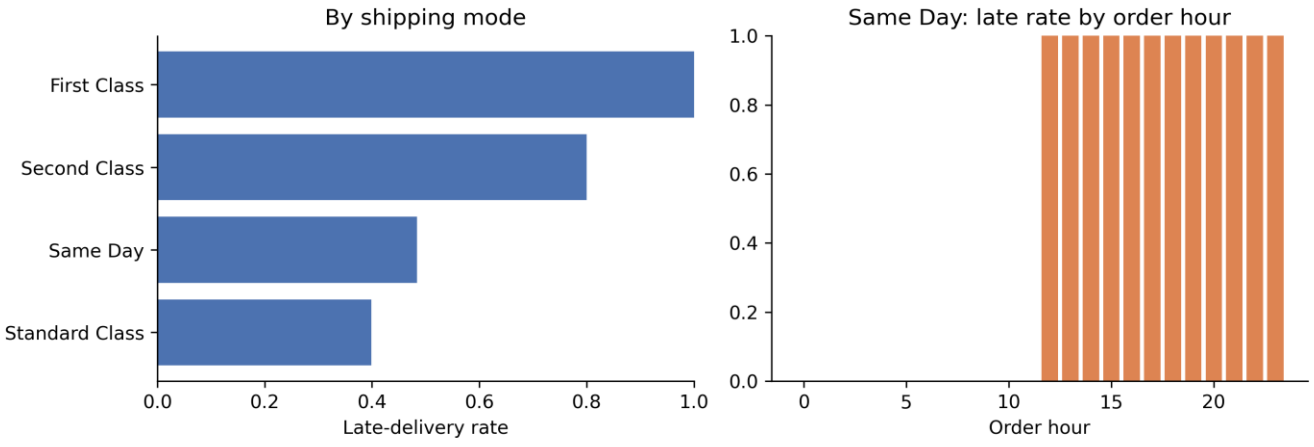


Figure 1. Late-delivery rate by shipping mode (left) and, for Same Day orders, by order hour (right), after removing cancelled and fraud orders.

METHODOLOGY

The pipeline builds an order table, a graph from it, three families of graph features, and a gradient-boosted classifier that is evaluated under leakage-free splits. Everything runs on the free CPU tier of Google Colab, with no large language model and no graph database.

4.1 Graph construction. We build one undirected, weighted, heterogeneous graph with NetworkX [9] from the cleaned item rows. Node types are customer, product, category, city, country and region. Edges link customer to product, customer to city, product to category, city to country and country to region, and the weight is the number of item rows supporting the pair. The graph has 24,268 nodes (20,261 customers, 3,651 cities, 164 countries, 118 products, 51 categories, 23 regions) and 165,059 edges. Labels and shipping modes are not used to build it.

4.2 Graph feature families.

The first two families are label-free: they use only the graph built from order attributes. The third family is different in kind. It is target encoding in which the grouping entities are graph nodes, so it uses training labels and must be computed out of fold. We therefore report label-free and label-aware features separately in the results.

- **Structural (10 features).** PageRank [10], weighted degree and Louvain community size [11] (46 communities) for the order's customer, city and country, plus the mean PageRank of the order's products.
- **Spectral embeddings (16 features).** Eight-dimensional truncated SVD [12] of the log-scaled customer-by-product and city-by-product count matrices, attached to each order through its customer and city.
- **Label-aware encodings (3 features).** The smoothed late rate of the order's customer, city and country, shrunk toward the training prior with weight 10 [13]. For training rows the encoding is computed out of fold inside the training set (inner 5-fold), and for test rows from the whole training fold, so no row ever sees its own label.

Structural features and embeddings use graph structure only, never labels. They are computed once over all orders, which is transductive; Section 7 discusses this.

4.3 Models. The tabular baseline has 16 features: shipping mode, scheduled days, payment type, customer segment, market, region, number of items, quantity, order total, mean discount, profit, number of distinct products, and the hour, weekday, month and year of the order. The learner is XGBoost [8] with 150 trees, depth 4, learning rate 0.1, row and column subsampling of 0.8, and default regularisation. Hyperparameters were fixed in advance and not tuned on the evaluation folds, because tuning there would add another source of optimism. scikit-learn [14] provides the folds, metrics and SVD.

Seven configurations are compared: (i) a lookup rule, (ii) tabular, (iii) tabular plus structure, (iv) tabular plus embeddings, (v) tabular plus structure and embeddings, (vi) tabular plus label-aware encodings, and (vii) tabular plus all graph features. The lookup rule predicts the training-fold late rate of one of five groups: First Class, Second Class, Standard Class, Same Day before noon, and Same Day at or after noon. It learns five numbers.

4.4 Evaluation protocol. The primary metric is ROC-AUC; PR-AUC and F1 at a 0.5 threshold are secondary.

- **Repeated cross-validation.** Each row is an order, so folds are shuffled 5-fold splits at order level. The ablation uses 3 repeats (15 folds). The final comparison of three pre-specified configurations (rule, tabular, all graph features) uses 10 new repeats (50 folds).
- **Significance.** Paired fold differences are tested with the Nadeau-Bengio corrected resampled t-test [15], which accounts for overlapping training sets. Holm correction [16] is applied across the six non-baseline configurations in the ablation and across the three pre-specified comparisons in the final experiment.

For k paired folds with AUC differences d_i, mean $\bar{d}$ and sample variance s^2, the corrected statistic is $t = \bar{d} / \sqrt{[(1/k + n_test/n_train) \cdot s^2]}$, compared with a t distribution with $k - 1$ degrees of freedom. With 5-fold cross-validation $n_test/n_train = 1/4$; $k = 15$ in the ablation and $k = 50$ in the final comparison.

- **Temporal split.** Train on orders before 1 January 2017 (39,950 orders) and test on later orders (22,947). The AUC difference against the rule gets a 95% interval from 500 bootstrap resamples of the test orders.
- **Within-mode AUC.** AUC computed per fold inside Second Class and Standard Class orders, where the rule predicts a constant (AUC 0.5).
- **Entity permutation test.** For customers (at least 5 orders), cities and countries (at least 20 orders), we compare the standard deviation of entity-mean residual late rates, after subtracting the rule's group mean, with a null from 300 label permutations.
- **Explainability.** SHAP [17] TreeExplainer on 3,000 random orders for the model with structure and embeddings.

RESULTS

5.1 Leakage audit. Leakage changes the answer more than any modelling choice (Table 3). Adding the actual shipping days gives a perfect AUC of 1.0000, because the label is a deterministic function of that column. With leak-free features, a random row split reports 0.7985, while the order-grouped split gives 0.7749, so the grouping error alone overstates AUC by 0.024.

Table 3. Effect of leakage on XGBoost AUC.

Setting	AUC
Leaky feature (Days for shipping, real), random row split	1.0000
Leak-free features, random row split (inflated)	0.7985
Leak-free features, order-grouped split (correct)	0.7749

5.2 Graph feature ablation. Table 4 and Figure 2 show the seven configurations over 3 repeats of 5-fold order-grouped cross-validation (15 folds). All AUCs lie between 0.7719 and 0.7760. The best graph variants (label-aware encodings, +0.0036; all graph features, +0.0041) have uncorrected p-values of 0.018 and 0.034, but after Holm correction over six comparisons they give 0.107 and 0.172. The structure and embedding families alone are not distinguishable from the tabular baseline (p of 0.31 to 0.49). The rule, which has five parameters, scores 0.7742, above the tabular model.

Table 4. Ablation results (mean AUC ± standard deviation over 15 folds; p from the Nadeau-Bengio corrected t-test against the tabular baseline).

Configuration	AUC	PR-AUC	F1	ΔAUC	p	Holm p
Rule lookup (mode + noon cutoff)	0.7742 ± 0.0024	0.8009	0.7070	+0.0023	0.195	0.781
Tabular	0.7719 ± 0.0037	0.8543	0.7073	–	–	–
Tabular + graph structure	0.7732 ± 0.0034	0.8550	0.7078	+0.0013	0.453	0.919
Tabular + spectral embeddings	0.7735 ± 0.0028	0.8552	0.7075	+0.0016	0.306	0.919
Tabular + structure + embeddings	0.7732 ± 0.0026	0.8549	0.7076	+0.0013	0.491	0.919
Tabular + label-aware encodings	0.7755 ± 0.0038	0.8560	0.7074	+0.0036	0.018	0.107
Tabular + all graph features	0.7760 ± 0.0030	0.8564	0.7090	+0.0041	0.034	0.172

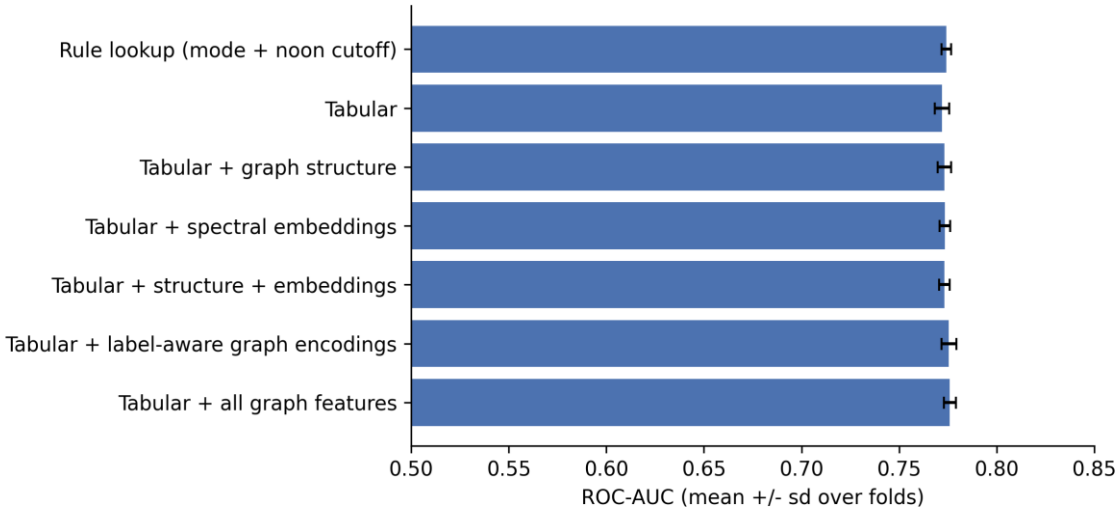


Figure 2. Mean AUC (± standard deviation over folds) of the seven configurations in the ablation. The differences are about one standard deviation.

The rule has a lower PR-AUC (0.8009) than the models because it outputs only five distinct scores, which coarsens the precision-recall curve; F1 is essentially the same for all configurations.

By family. The two label-free families give nominal gains of +0.0013 to +0.0016 AUC (p of 0.31 to 0.49), while label-aware encodings give +0.0036 (uncorrected p = 0.018, Holm p = 0.107). Only the label-aware family, which uses the training labels of graph neighbours, shows any nominal improvement, and it does not survive correction.

5.3 Final pre-specified comparison. To test the most favourable graph configuration without reusing the ablation folds, we ran 10 new repeats (50 folds) on three configurations fixed in advance (Table 5). Graph features improve on tabular by 0.0030 AUC (p = 0.0486, better in 44 of 50 folds), which is not significant after Holm correction over three comparisons (p = 0.146). Against the rule the difference is 0.0009 (p = 0.445). The rule also beats the tabular model by 0.0021 (Holm p = 0.146). The graph model therefore only recovers the rule's performance; it does not exceed it.

Table 5. Final comparison, 10 repeats × 5 folds (mean AUC ± standard deviation over 50 folds).

Configuration	AUC
Rule lookup (mode + noon cutoff)	0.7742 ± 0.0029
Tabular	0.7721 ± 0.0034
Tabular + all graph features	0.7751 ± 0.0038

Comparison	ΔAUC	p	Folds better	Holm p
All graph features − Tabular	+0.0030	0.0486	44/50	0.146
All graph features − Rule	+0.0009	0.445	31/50	0.445
Rule − Tabular	+0.0021	0.0499	41/50	0.146

5.4 Where the signal is. Table 6 gives per-fold AUC inside each shipping mode. For Same Day orders, every model is perfect (1.0000), as the rule is. For Second Class and Standard Class orders, whose labels are not determined by mode, AUC is about 0.50 for all models. The tabular model is slightly below chance in Second Class (0.4840), and the graph model sits at 0.4914 and 0.5032, so no model orders orders within a mode better than chance.

Table 6. Within-mode AUC per fold (0.5 means no signal; 10 repeats × 5 folds).

Mode	Rule	Tabular	All graph features
Second Class	0.5000 ± 0.0000	0.4840 ± 0.0139	0.4914 ± 0.0139
Standard Class	0.5000 ± 0.0000	0.4951 ± 0.0056	0.5032 ± 0.0063

The entity permutation test (Table 7) points the same way. After subtracting the mode-level rate, the spread of late rates across customers, cities and countries is indistinguishable from the spread produced by shuffled labels ($p = 0.31$ to 0.69), so there is no entity-level signal for the graph to carry.

Table 7. Entity permutation test (300 permutations).

Entity	Entities	Observed SD	Null SD	z	p
Customer	5,592	0.1697	0.1702	-0.32	0.633
City	712	0.0721	0.0710	0.48	0.307
Country	111	0.0434	0.0456	-0.54	0.690

5.5 Temporal split. Training on 39,950 orders placed before 2017 and testing on 22,947 later orders reverses the small nominal gain (Table 8). The rule scores 0.7741. Graph structure and embeddings are within noise of it, and the full graph model is worse by 0.0053, with a bootstrap interval of [-0.0087, -0.0016] that excludes zero. Label-aware encodings, which learn customer and city late rates, also fall below the rule (-0.0024, interval including zero), consistent with the absence of stable entity effects.

Table 8. Temporal split (train before 2017, test 2017 onward).

Configuration	Test AUC	ΔAUC vs rule	95% CI
Rule lookup (mode + noon cutoff)	0.7741	0.0000	–
Tabular	0.7733	-0.0008	[-0.0044, +0.0026]
Tabular + graph structure	0.7757	+0.0016	[-0.0019, +0.0052]
Tabular + spectral embeddings	0.7761	+0.0020	[-0.0015, +0.0057]
Tabular + structure + embeddings	0.7748	+0.0007	[-0.0029, +0.0045]
Tabular + label-aware encodings	0.7717	-0.0024	[-0.0059, +0.0013]
Tabular + all graph features	0.7689	-0.0053	[-0.0087, -0.0016]

5.6 Explainability. SHAP values for the model with structure and embeddings (Figure 3) show that shipping mode carries 66.9% of total mean absolute SHAP importance, scheduled shipping days 8.8% and order hour 6.3%. The highest-ranked graph features (customer embedding dimensions 3 and 7, city embedding dimension 7, mean product PageRank, city PageRank) each contribute under 1%.

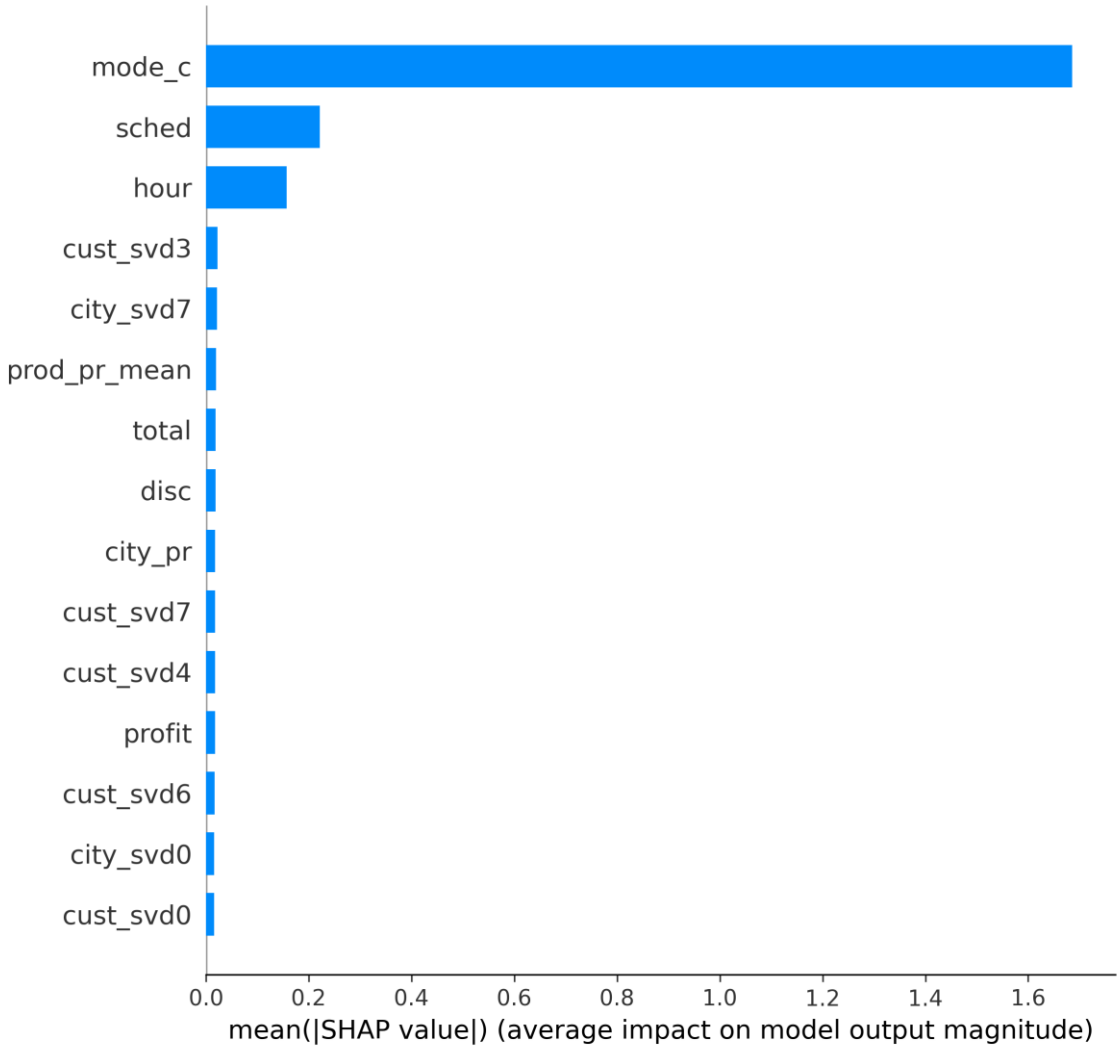


Figure 3. Share of total mean absolute SHAP importance by feature (3,000 orders). Shipping mode dominates; every graph feature is below 1%.

Direction of effects. Mean absolute SHAP values hide direction, so Figure 4 shows the signed values, in log-odds of a late delivery, for the same 3,000 orders. Shipping mode pushes predictions up for First Class (mean SHAP +6.74) and Second Class (+0.90) and down for Standard Class (−0.64); Same Day averages −0.06 because its two halves (before and after noon) have opposite signs. Longer scheduled shipping days lower the predicted risk (Spearman correlation between feature value and SHAP value −0.71), and later order hours raise it (+0.70), which is the noon rule for Same Day orders. The graph features have signed contributions of about ±0.3 log-odds or less, with high and low feature values on both sides of zero, so no graph feature has a consistent direction. Shipping mode is a category code, so its colour in Figure 4 is not an ordinal scale.

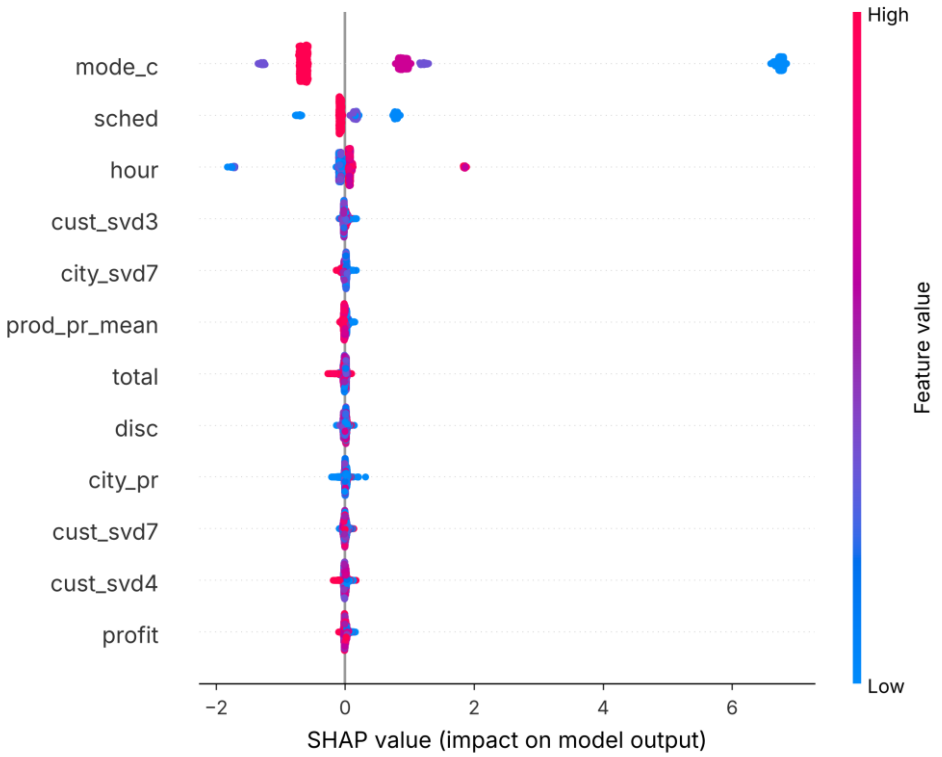


Figure 4. Signed SHAP values (log-odds of late delivery) for the 12 most important features of the model with structure and embeddings, 3,000 orders. Colour shows the feature value (blue low, red high).

5.7 Computational cost. Table 9 reports the measured cost of each stage. The measurements were taken on a CPU-only machine with 2 cores and 7 GB of memory, in a single process. The whole graph pipeline takes about 5 s once and 0.15 s per fold for the label-aware encodings, and raising the feature count from 16 to 45 doubles the time of one XGBoost fit from 0.28 s to 0.57 s. Peak memory for the entire workflow, including loading the 180,519-row file, was 0.74 GB. These costs are small in absolute terms, so on this dataset the argument against graph features is not cost but the absence of any gain: the overhead buys nothing measurable. On larger graphs, PageRank and Louvain community detection are likely to dominate, which we did not test.

Table 9. Measured cost per stage (2-core CPU, 7 GB RAM; one run, best of three for model fits).

Stage	Time
Load and clean 180,519 item rows	4.5 s
Build order table (62,897 orders)	0.6 s
Build graph (24,268 nodes, 165,059 edges), PageRank, Louvain	3.1 s
Structural lookups and two truncated SVDs	1.3 s
Label-aware encodings, per fold	0.15 s
XGBoost fit, 16 tabular features	0.28 s
XGBoost fit, 45 features (all graph features)	0.57 s
SHAP on 3,000 orders	0.19 s
Peak memory of whole workflow	0.74 GB

DISCUSSION

Why graph features add nothing here. The label is almost a function of shipping mode and, for Same Day orders, order hour. What remains in Second Class and Standard Class orders behaves like noise: within-mode AUC is 0.50, and customers, cities and countries do not differ from shuffled labels. A graph can only help if neighbours carry information about the target, and in this file they do not. The nominal gain of 0.003 AUC in

the repeated cross-validation is better read as recovery of the rule's score. The tabular model loses about 0.002 AUC against the rule because it fits noise in the noisy modes, and graph features, particularly label-aware encodings, give it enough extra inputs to land near the rule's level, not above it. The temporal split supports this reading: when the future differs from the past, the same features rank below the rule.

Reading the SHAP result. Shipping mode alone accounts for 66.9% of SHAP importance, and scheduled shipping days add 8.8%. These two are strongly related, because scheduled days are fixed largely by the shipping mode, so their shares overlap and together (about 76%) describe one underlying source of signal rather than two.

Relation to published results. High scores reported on DataCo, such as the AUC of 0.977 for EAGLE [3], are not comparable with ours, since they use a different target (next-window delay of a hub) and different inputs. Our point is narrower: on the per-order label, a five-number rule reaches 0.774, so any claimed improvement for a per-order model needs to be shown against that baseline and under grouped, temporal evaluation.

Practical recommendations. (i) Exclude every post-shipment column, in particular actual shipping days, delivery status and shipping date. (ii) Split by order, not by row, and report a temporal split. (iii) Always report a simple rule baseline and within-group AUC. (iv) Apply multiple-comparison correction when many feature families are tried.

Transfer to real operations. The label's regularity (First Class always late, Same Day decided exactly at noon, constant rates elsewhere) suggests that the public file was generated or heavily processed. Real shipments are likely to carry richer signal from carriers, routes, weather and inventory, which graph features could capture. Our negative result applies to this dataset and should not be generalised to every supply chain.

Why is the label so regular? Three explanations are possible. The delivery outcome may have been generated or heavily post-processed by rules tied to shipping mode and order time; the label may have been derived from scheduled and actual days under mode-specific delay distributions; or the underlying operation may really be rule-driven. The evidence favours a rule-like process: all 9,602 First Class orders are late, Same Day orders flip exactly at noon, Second and Standard Class rates are constant (80.0% and 39.8%), within-mode AUC is 0.50 for every model, and customer, city and country rates match shuffled labels. Real delivery data usually vary with carrier, route, weather and inventory, so this combination is unusual. The public documentation describes a company selling clothing, sports and electronic supplies, but we cannot verify from the file how the records were produced or whether they were anonymised or synthesised, and we do not know whether the regularity is specific to this sector. We also did not test other public supply chain datasets, so we make no claim about whether they share the problem. The checklist below can be run on any such dataset in minutes.

Checklist before investing in graph methods. (1) Build a rule baseline from the few dominant categorical fields and require every model to beat it under order-grouped, temporal evaluation. (2) Compute AUC inside groups where the rule is constant; a value near 0.50 means that no remaining signal has been found. (3) Run a permutation test on residual entity-level rates (customers, locations, and where available carriers, routes and warehouses); if the spread matches the null, neighbour-based features cannot help. (4) Check that the data contain entities that can carry relational signal at all; DataCo has no carrier, route or warehouse. (5) Decide in advance the smallest gain that would matter and check that the evaluation can detect it. Here the fold-to-fold standard deviation of AUC is 0.003 to 0.004, and a gain of 0.003 with 50 repeated folds on 62,897 orders gave an uncorrected p of 0.049, so gains of this size are at the edge of detectability and need more repeats or data. (6) Report the cost of the graph pipeline next to the gain (Section 5.7). (7) Only then try end-to-end graph learning, and compare it with the same rule baseline.

7. Limitations

- **Hand-crafted features only; no graph neural network.** We test graph features appended to a tabular model, not end-to-end graph learning. A graph neural network could learn higher-order relational structure that these features miss. The absence of entity effects in the permutation test suggests that there

is little relational signal among customers, cities and countries for any method, but this argument is not a proof, and our conclusion applies to hand-crafted features only.

- **One dataset.** All results come from DataCo, whose label has an unusually regular structure and contains no carrier, route or warehouse entities. Other logistics data may behave differently.
- **Transductive graph features.** Structural features and embeddings use all orders, including test orders, but never their labels. A stricter inductive version could only lower their scores.
- **One learner and one temporal cut.** We used XGBoost with fixed settings and a single split date (1 January 2017).
- **Cost measured on different hardware.** The costs in Section 5.7 were measured on a 2-core CPU machine and not on the machine used for the original runs; absolute times will differ.

CONCLUSION

We asked whether hand-crafted graph features improve late-delivery risk prediction on the DataCo dataset once leakage is controlled. They do not. A leaky column gives AUC 1.00 and a random row split overstates AUC by 0.024. Without these errors, shipping mode and the Same Day order hour explain all the learnable signal, and a five-number lookup rule matches the best model (AUC 0.774 against 0.775). Hand-crafted graph features give a nominal +0.003 AUC that disappears under Holm correction, is indistinguishable from the rule, and reverses on a temporal split. Within noisy shipping modes no model beats chance, and customer, city and country late rates match shuffled labels. For benchmarking on DataCo we recommend order-grouped, temporal evaluation against a rule baseline. The conclusion applies to hand-crafted features; graph neural networks remain untested. Future work should test graph methods on data with real network signal, such as carriers, routes and warehouses, and use inductive graph learning.

Declarations

Ethics. This study uses a public, anonymised dataset. Customer names, e-mail addresses and street addresses were not used. No human participants were involved.

Conflict of interest. The author declares no conflict of interest.

Funding. No external funding was received.

Data availability. The DataCo Smart Supply Chain dataset is public [1].

[https://www.kaggle.com/datasets/shashwatwork/dataco-smart-supply-chain-for-big-data-analysis]

Use of AI tools. No large language model is used as part of the proposed methodology. Claude (Anthropic) was used for writing assistance, code development, analysis scripts and formatting. The author reviewed all outputs and is responsible for the content.

REFERENCES

1. F. Constante, F. Silva and A. Pereira, "DataCo Smart Supply Chain for Big Data Analysis," Mendeley Data, V5, 2019. doi: 10.17632/8gx2fvg2k6.5.
2. V. K. Dwivedi, K. Kadian, B. Gardiner, R. Pandey and A. Lathwal, "Supply Chain Analytics: A Performance Evaluation of Machine Learning, Statistical, and Time-Series Models," Baltic Journal of Modern Computing, vol. 13, no. 2, pp. 453–485, 2025. doi: 10.22364/bjmc.2025.13.2.09.
3. Z. Xue, M. Huo and Y. Wang, "EAGLE: Edge-Aware Graph Learning for Proactive Delivery Delay Prediction in Smart Logistics Networks," arXiv:2604.05254, 2026.

4. S. Kaufman, S. Rosset, C. Perlich and O. Stitelman, "Leakage in data mining: Formulation, detection, and avoidance," ACM Transactions on Knowledge Discovery from Data, vol. 6, no. 4, article 15, 2012. doi: 10.1145/2382577.2382579.
5. D. R. Roberts et al., "Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure," Ecography, vol. 40, no. 8, pp. 913–929, 2017. doi: 10.1111/ecog.02881.
6. A. Hogan et al., "Knowledge graphs," ACM Computing Surveys, vol. 54, no. 4, article 71, 2021. doi: 10.1145/3447772.
7. L. Grinsztajn, E. Oyallon and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?," in Advances in Neural Information Processing Systems (Datasets and Benchmarks Track), 2022. arXiv:2207.08815.
8. T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 785–794. doi: 10.1145/2939672.2939785.
9. A. A. Hagberg, D. A. Schult and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in Proc. 7th Python in Science Conf. (SciPy 2008), 2008, pp. 11–15.
10. L. Page, S. Brin, R. Motwani and T. Winograd, "The PageRank citation ranking: Bringing order to the web," Stanford InfoLab Technical Report, 1999.
11. V. D. Blondel, J.-L. Guillaume, R. Lambiotte and E. Lefebvre, "Fast unfolding of communities in large networks," Journal of Statistical Mechanics: Theory and Experiment, article P10008, 2008. doi: 10.1088/1742-5468/2008/10/P10008.
12. N. Halko, P.-G. Martinsson and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," SIAM Review, vol. 53, no. 2, pp. 217–288, 2011. arXiv:0909.4061.
13. D. Micci-Barreca, "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," ACM SIGKDD Explorations Newsletter, vol. 3, no. 1, pp. 27–32, 2001. doi: 10.1145/507533.507538.
14. F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
15. C. Nadeau and Y. Bengio, "Inference for the generalization error," Machine Learning, vol. 52, no. 3, pp. 239–281, 2003. doi: 10.1023/A:1024068626366.
16. S. Holm, "A simple sequentially rejective multiple test procedure," Scandinavian Journal of Statistics, vol. 6, no. 2, pp. 65–70, 1979.
17. S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems 30, 2017, pp. 4765–4774. arXiv:1705.07874.