National College of Ireland

Predicting Retail Supply Chain Disruptions Using Graph-Based Machine Learning Models

MSc Data Analytics
Research in Computing

Shashidar Vemula
Student ID:23394200

School of Computing
National College of Ireland

Supervisor: Dr. Anu Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shashidar Vemula
Student ID:	23394200
Programme:	MSC in Data Analytics **Year:** 2025
Module:	Research in Computing
Lecturer:	Dr. Anu Sahni
Submission Due Date:	25-01-2026
Project Title:	Predicting Retail Supply Chain Disruptions Using Graph-Based Machine Learning
Word Count:	1420 **Page Count:** 24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Shashidar Vemula

Date: 25-01-2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Predicting Retail Supply Chain Disruptions Using Graph-Based Machine Learning Models

Shashidar Vemula
Student ID: 23394200

1 Introduction

This configuration manual is an end-to-end technical outline to replicate the Retail Supply Chain Analytics and Predictive Disruption Detection System. The system is a combination of three big layers of analysis:

1. **Classical descriptive/diagnostic analytics** (EDA, KPIs, time-series patterns)
2. **Graph Neural Networks Customer-Product relationship modelling (GCN, GAT, GraphSAGE)**
3. **Baseline machine learning models (Random Forest and SVM) that enhance the performance of GNN by direct customer-level tabular modelling**

The manual is reproducible due to the hardware/software requirements, instructions on how to ingest data, clean data, feature engineering rules and graph construction strategy, model-train configurations, protocols on how to measure performance and explainability routines using SHAP and GNNExplainer.

2 Hardware and Software Requirement

2.1 Hardware Requirements

To be able to execute data processing, visualization, training of GNNs, and SHAP explainability in a stable envirment:

- **Processor:** Intel Core i5
- **RAM:** Used 16 GB for GNN training
- **GPU:** Recommended but not necessary
- **Storage:** File is 3MB

2.2 Software Requirements

Operating System: Windows 11

Language: Python 3.12

Development Environment:

- Google Colab (used in implementation)

Installing Libraries **"**pip install numpy pandas matplotlib seaborn scikit-learn torch torch_geometric shap tqdm**"**

3 Dataset Configuration

3.1 Dataset Upload

The system is designed to load **Excel** files through an upload widget in Google Colab.

- Only a dataset file is supposed to be uploaded.
- Using Excel file, the **first sheet** is used.
- Dataset link: Link

```
# Upload
uploaded = files.upload()
assert len(uploaded) == 1, "Please upload exactly one dataset file."

fname = next(iter(uploaded))
print(" File uploaded:", fname)

buffer = io.BytesIO(uploaded[fname])

df = None
try:
    buffer.seek(0)
    df = pd.read_csv(buffer)
    print(" Loaded as CSV.")
except Exception:
    buffer.seek(0)
    xls = pd.ExcelFile(buffer)
    first_sheet = xls.sheet_names[0]
    df = pd.read_excel(buffer, sheet_name=first_sheet)
    print(f" Loaded as Excel. Using sheet: {first_sheet}")

print(" Loaded shape:", df.shape)
```

Figure 1: Dataset Upload

3.2 Dataset Description

```
Loaded shape: (9994, 23)
print(" First 5 rows:")
display(df.head())
First 5 rows:
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Retail Sales People	Product ID	Category	Sub-Category	Product Name	Returned	Sales	Q
0	1	CA-2016-152156	2016-08-11	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	Cassandra Brandow	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase	Not	261.9600	
1	2	CA-2016-152156	2016-08-11	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	Cassandra Brandow	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,...	Not	731.9400	
2	3	CA-2016-138688	2016-12-06	2016-12-06	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	Anna Andreadi	OFF-LA-10000240	Office Supplies	Labels	Self-Adhesive Address Labels for Typewriters b...	Not	14.6200	
3	4	US-2015-108966	2015-11-10	2015-11-10	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	Cassandra Brandow	FUR-TA-10000577	Furniture	Tables	Bretford CR4500 Series Slim Rectangular Table	Not	957.5775	

Figure 2: Dataset Description

The uploaded retail dataset contains:

- **9,994 rows**

- **23 columns** including:
  - Order metadata
  - Customer information
  - Product hierarchy (Category, Sub-category)
  - Sales & profit measures
  - Discounting
  - Returned product indicator

The fields include categorical, numerical, and date-time variables.

3.3 Data Type Verification

- Date columns (`Order Date`, `Ship Date`) are parsed into datetime format.
- Sales, Quantity, Discount, Profit, and other numberics are remain as float/int.

```
# Parse dates if present
for c in ["order_date", "ship_date"]:
    if c in df_base.columns:
        df_base[c] = pd.to_datetime(df_base[c], errors="coerce")

# Make a simple 'is_returned' flag from 'returned' if present
if "returned" in df_base.columns:
    df_base["is_returned"] = (
        df_base["returned"]
        .astype(str).str.strip().str.lower()
        .map({"yes": 1, "y": 1, "true": 1, "1": 1, "no": 0, "n": 0, "false": 0, "0": 0})
        .fillna(0).astype(int)
    )

# Lead time (days): ship_date - order_date
if {"order_date","ship_date"}.issubset(df_base.columns):
    df_base["lead_time_days"] = (df_base["ship_date"] - df_base["order_date"]).dt.days

# Calendar helpers
if "order_date" in df_base.columns:
    df_base["order_month"] = df_base["order_date"].dt.to_period("M").dt.to_timestamp()
```

Figure 3: Date Datatypes Parsing

3.4 Missing Value Assessment

- The dataset contains **0 missing values** across all columns.
- No imputation or row removal is necessary.

```
dtype: int64
  Total missing values (after drop):
0
```

3.5 Duplicate Assessment

- There are **0 duplicates** in both original as well as cleaned data.

```
dups = df.duplicated().sum()
print(f" Duplicate rows in ORIGINAL df: {dups}")

dups_clean = df_clean.duplicated().sum()
print(f" Duplicate rows in df_clean: {dups_clean}")
```

```
Duplicate rows in ORIGINAL df: 0
Duplicate rows in df_clean: 0
```

Figure 4: Duplicate Check in Data

4 Data Cleaning and Feature Engineering

Some engineered fields are then added after all column names are standardized in lowercase snake case.

4.1 Feature Transformations

```
# Make a simple 'is_returned' flag from 'returned' if present
if "returned" in df_base.columns:
    df_base["is_returned"] = (
        df_base["returned"]
        .astype(str).str.strip().str.lower()
        .map({"yes": 1, "y": 1, "true": 1, "1": 1, "no": 0, "n": 0, "false": 0, "0": 0})
        .fillna(0).astype(int)
    )

# Lead time (days): ship_date - order_date
if {"order_date","ship_date"}.issubset(df_base.columns):
    df_base["lead_time_days"] = (df_base["ship_date"] - df_base["order_date"]).dt.days

# Calendar helpers
if "order_date" in df_base.columns:
    df_base["order_month"] = df_base["order_date"].dt.to_period("M").dt.to_timestamp()
```

Figure 5: Feature Transformation

New Feature	Description
`is_returned`	Binary flag derived from “Returned” column
`lead_time_days`	Difference between Ship Date and Order Date
`order_month`	Calendar month for time-series aggregation

4.2 Cleaned Dataset Dimensions

- Final dataset: **9,994 rows × 26 columns**

5 Descriptive KPIs and Aggregated Metrics

The system will calculate significant KPIs that summarize customer, product behavior, and sales behavior.

5.1 Computed KPIs Include

- Number of rows and columns
- Unique customers, products, orders, cities, and regions
- Aggregated financial metrics:
  - Total Sales
  - Total Profit
  - Total Quantity Sold
  - Total Discounts Applied
- Return rate (in %)
- Average lead time

```
kpis = {}

kpis["rows"] = len(df_base)
kpis["cols"] = df_base.shape[1]
for col in ["order_id","customer_id","product_id","city","state","region","category","subcategory"]:
    if col in df_base.columns:
        kpis[f"unique_{col}"] = df_base[col].nunique()

for col in ["sales","profit","quantity","discount"]:
    if col in df_base.columns:
        kpis[f"sum_{col}"] = float(df_base[col].sum())

if "is_returned" in df_base.columns:
    kpis["return_rate_%"] = round(100 * df_base["is_returned"].mean(), 2)

if "lead_time_days" in df_base.columns:
    kpis["avg_lead_time_days"] = float(df_base["lead_time_days"].mean())

print("KPIs")
for k,v in kpis.items():
    print(f"- {k}: {v}")
```

Figure 6: Computed KPIs

6 Exploratory Data Analysis (EDA)

6.1 Monthly Sales and Profit Trend

A time-series visualization shows:

- Seasonal peaks
- Profit volatility
- Sales trend across months

```
need_cols = {"order_month","sales","profit"}
if need_cols.issubset(df_base.columns):
    ts = (df_base
          .groupby("order_month", as_index=False)[["sales","profit"]]
          .sum()
          .sort_values("order_month"))

    fig, ax = plt.subplots(figsize=(12,5))
    sns.lineplot(data=ts, x="order_month", y="sales", marker="o", label="Sales", ax=ax)
    sns.lineplot(data=ts, x="order_month", y="profit", marker="o", label="Profit", ax=ax)
    ax.set_title("Monthly Sales & Profit")
    ax.set_xlabel("Month")
    ax.set_ylabel("Amount")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    display(ts.tail(12))
else:
    print("Columns needed for this chart not found.")
```

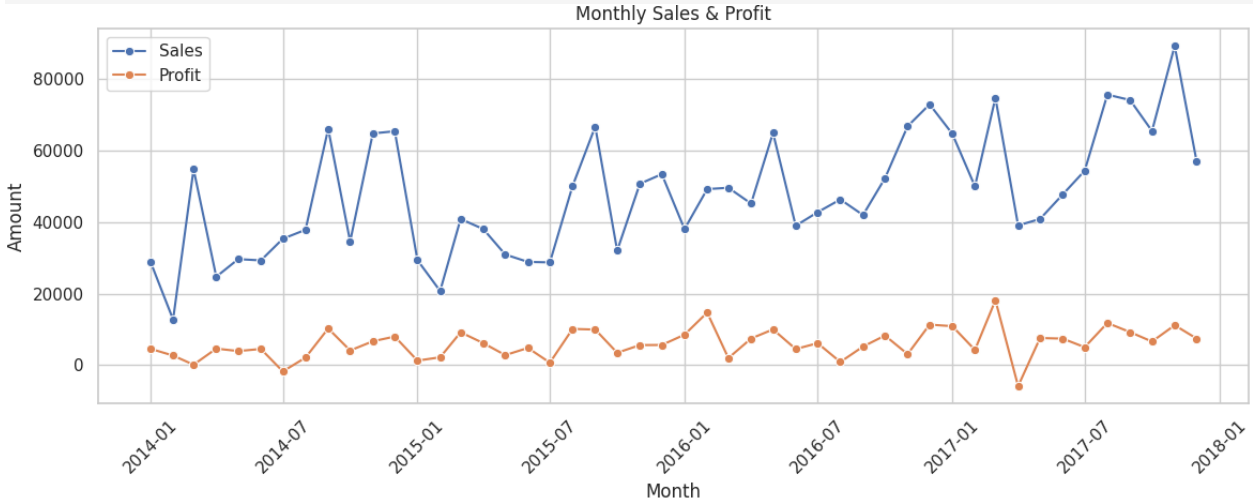

Figure 7: Monthly Sales and Profit Trend

6.2 Lead Time Distribution

A histogram and boxplot illustrate delivery delay patterns.

```python
if "lead_time_days" in df_base.columns:
    fig, axes = plt.subplots(1, 2, figsize=(14,5))
    sns.histplot(df_base["lead_time_days"].dropna(), bins=30, ax=axes[0])
    axes[0].set_title("Lead Time (days) — Histogram")

    sns.boxplot(x=df_base["lead_time_days"].dropna(), ax=axes[1])
    axes[1].set_title("Lead Time (days) — Boxplot")
    plt.tight_layout()
    plt.show()

    print("Lead time summary:")
    display(df_base["lead_time_days"].describe())
else:
    print("lead_time_days not available.")
```

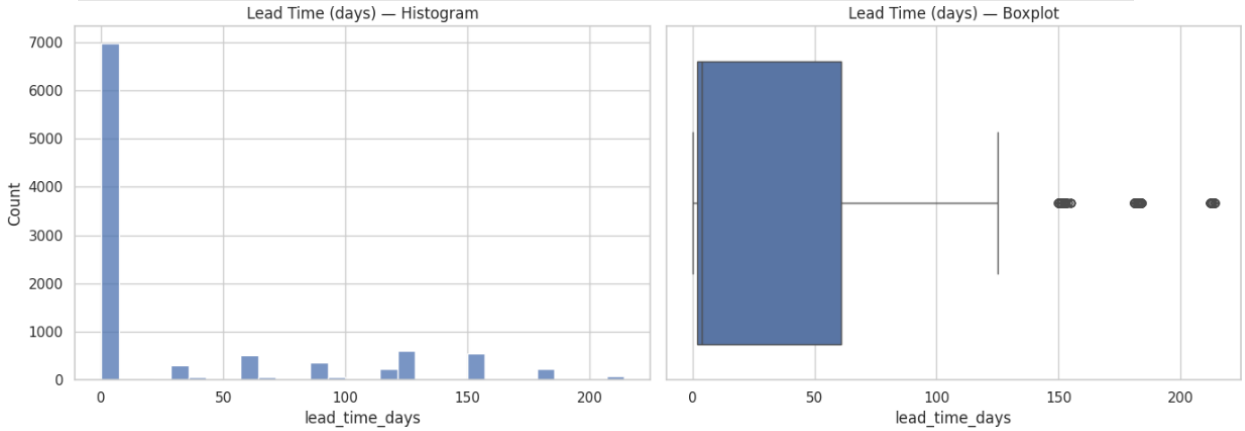

Figure 8: Lead Time Distribution

6.3 Region-Level Lead Time Comparison

Average delay time per US region is computed and plotted.

```python
if {"lead_time_days","region"}.issubset(df_base.columns):
    reg_lt = (df_base
              .dropna(subset=["lead_time_days"])
              .groupby("region", as_index=False)["lead_time_days"]
              .mean()
              .sort_values("lead_time_days", ascending=False))

    sns.barplot(data=reg_lt, x="region", y="lead_time_days")
    plt.title("Average Lead Time by Region")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    display(reg_lt)
else:
    print("Need 'region' and 'lead_time_days'.")
```

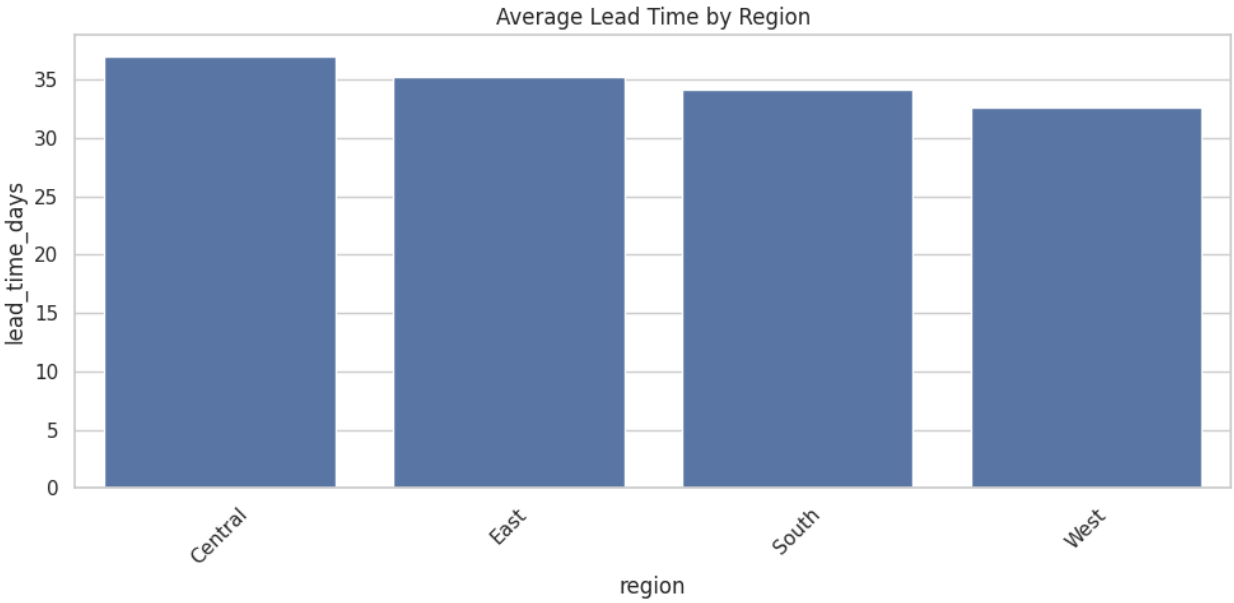

Figure 9: Average Lead Timme by Region

6.4 Category-Level Return Rate

A bar chart is used to point out the categories that have greater probability of a return.

```python
if {"category","is_returned"}.issubset(df_base.columns):
    cat_rr = (df_base
              .groupby("category", as_index=False)["is_returned"]
              .mean()
              .rename(columns={"is_returned":"return_rate"})
              .sort_values("return_rate", ascending=False))

    sns.barplot(data=cat_rr, x="category", y="return_rate")
    plt.title("Return Rate by Category")
    plt.ylabel("Return Rate (0–1)")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    display(cat_rr)
else:
    print("Need 'category' and 'is_returned'.")
```

Figure 10: Return Rate by Category

6.5 Profit–Discount Relationship

A regression plot identifies the expected negative correlation:

- Discount ↑ → Profit ↓

```python
if {"discount","profit"}.issubset(df_base.columns):
    # Clip discount to [0,1] for sensible view (if needed)
    disc = df_base["discount"].clip(0, 1) if df_base["discount"].max() > 1 else df_base["discount"]
    temp = df_base.assign(_disc=disc)

    sns.regplot(data=temp, x="_disc", y="profit", scatter_kws={"alpha":0.3}, line_kws={"color":"black"})
    plt.title("Profit vs Discount (with trend)")
    plt.xlabel("Discount (0–1)")
    plt.ylabel("Profit")
    plt.tight_layout()
    plt.show()

    # Quick correlation
    print("Correlation (discount vs profit):", temp[["_disc","profit"]].corr().iloc[0,1])
else:
    print("Need 'discount' and 'profit'.")
```

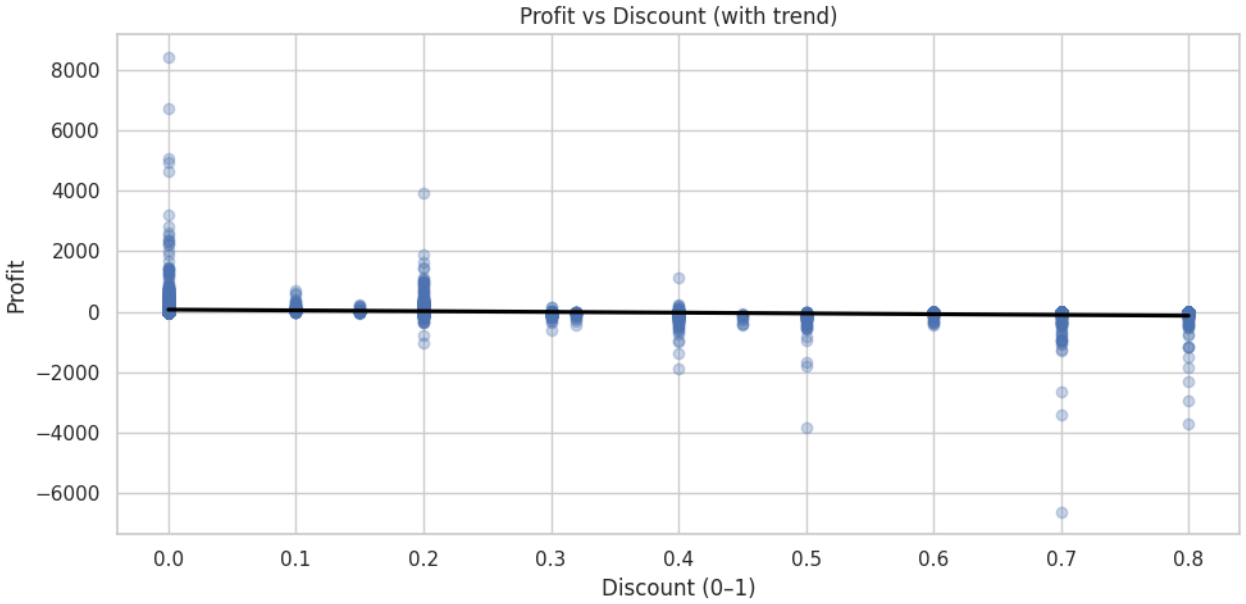

Figure 11: Profit Discount Relationship

6.6 Top 10 Products by Sales

Bar chart of highest revenue-generating products.

```python
if {"product_id","product_name","sales"}.issubset(df_base.columns):
    top_prod = (df_base
                .groupby(["product_id","product_name"], as_index=False)["sales"]
                .sum()
                .sort_values("sales", ascending=False)
                .head(10))

    sns.barplot(data=top_prod, y="product_name", x="sales")
    plt.title("Top 10 Products by Sales")
    plt.xlabel("Sales")
    plt.ylabel("")
    plt.tight_layout()
    plt.show()

    display(top_prod)
else:
    print("Need 'product_id','product_name','sales'.")
```

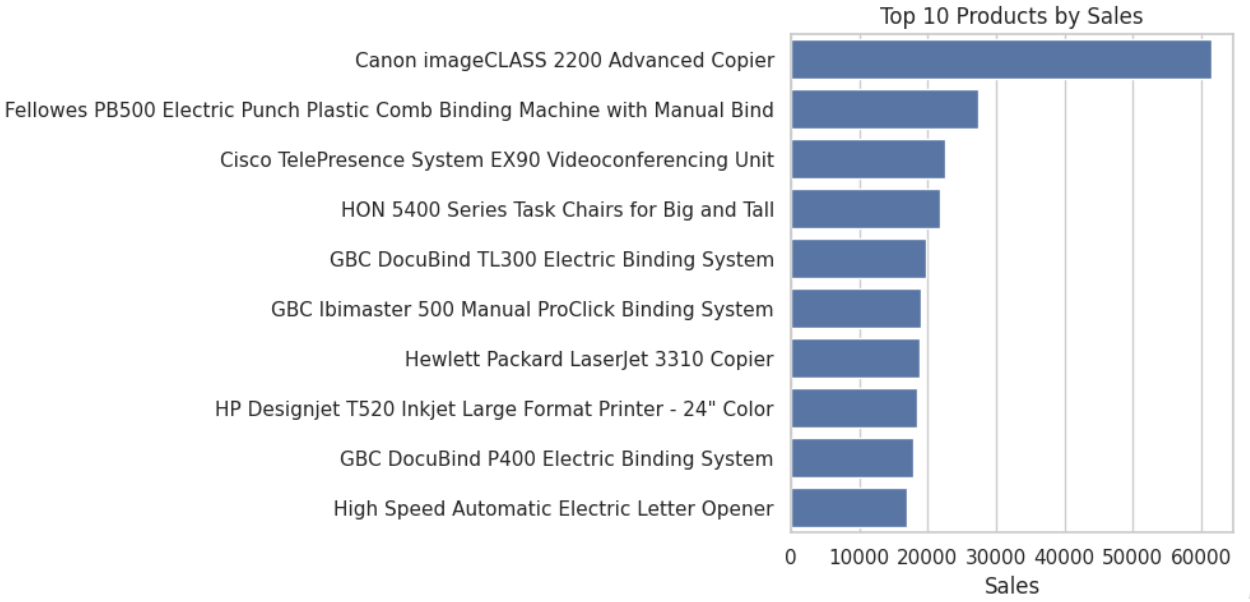

Figure 12: Top 10 Product by Sales

6.7 Category-Level Sales & Profit Contribution

Separate bar charts for:

- Sales by category
- Profit by category

```python
need = {"category","sales","profit"}
if need.issubset(df_base.columns):
    cat_perf = (df_base
                .groupby("category", as_index=False)[["sales","profit"]]
                .sum()
                .sort_values("sales", ascending=False))

    fig, axes = plt.subplots(1, 2, figsize=(14,5))
    sns.barplot(data=cat_perf, x="category", y="sales", ax=axes[0])
    axes[0].set_title("Sales by Category")
    axes[0].tick_params(axis="x", rotation=45)

    sns.barplot(data=cat_perf, x="category", y="profit", ax=axes[1])
    axes[1].set_title("Profit by Category")
    axes[1].tick_params(axis="x", rotation=45)

    plt.tight_layout()
    plt.show()

    display(cat_perf)
else:
    print("Need 'category','sales','profit'.")
```

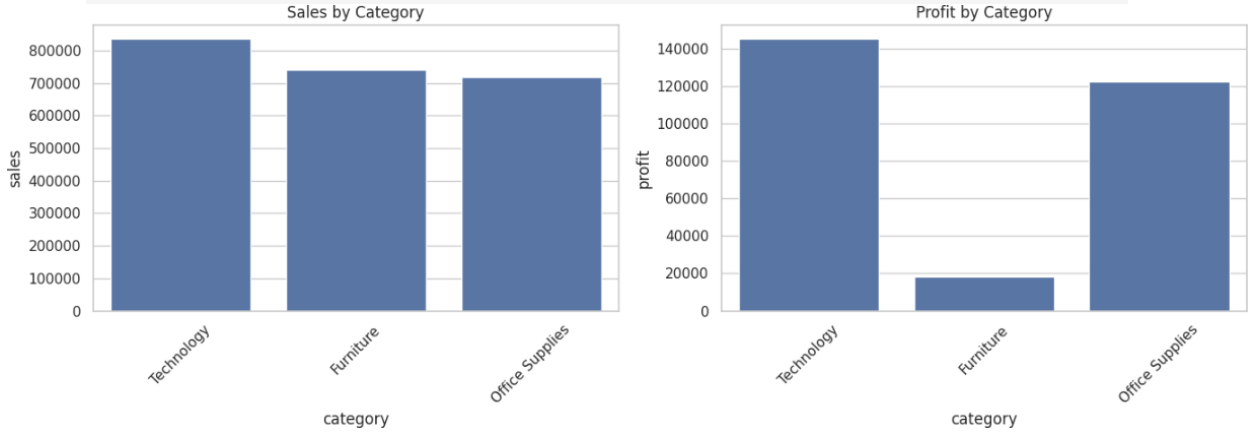

Figure 13: Category-Level Sales & Profit Contribution

6.8 Correlation Heatmap

Using numerical fields:
`sales, profit, quantity, discount, lead_time_days`

```
num_cols = [c for c in ["sales","profit","quantity","discount","lead_time_days"] if c in df_base.columns]
if len(num_cols) >= 2:
    corr = df_base[num_cols].corr()
    sns.heatmap(corr, annot=True, cmap="coolwarm", vmin=-1, vmax=1)
    plt.title("Correlation Heatmap")
    plt.tight_layout()
    plt.show()

    display(corr)
else:
    print("Not enough numeric columns for a correlation view.")
```

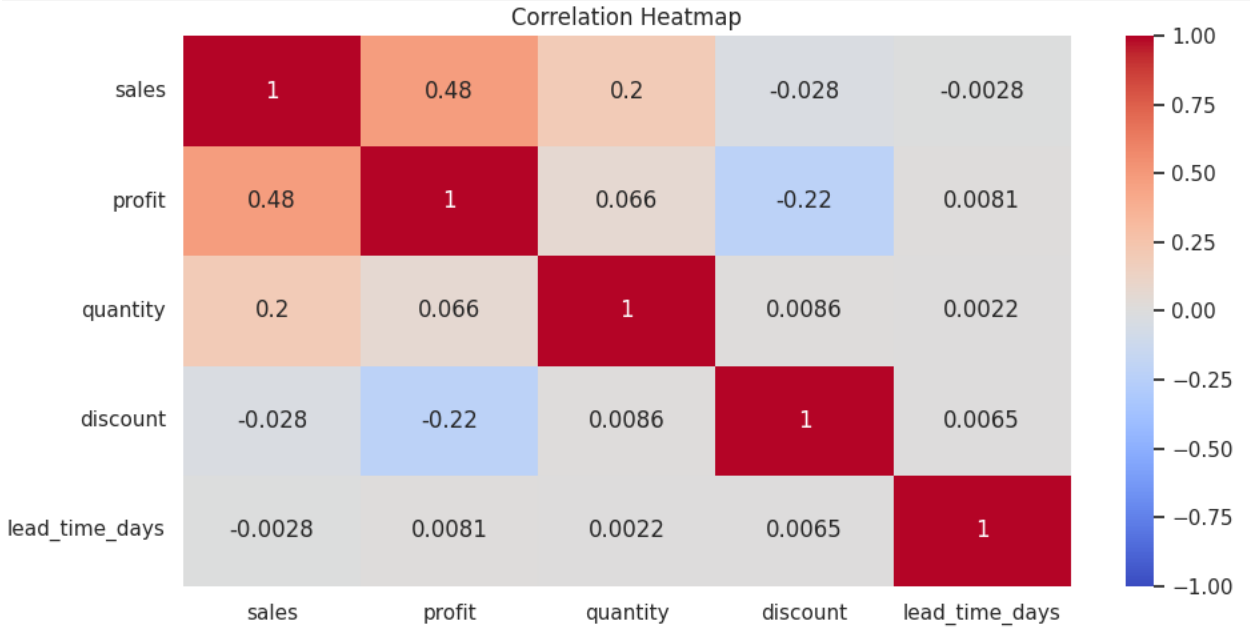

Figure 14: Correlation Heatmap

6.9 Pareto Analysis (80/20 Rule)

Illustrates which categories contribute 80% of total sales.

```python
if {"category","sales"}.issubset(df_base.columns):
    pareto = (df_base.groupby("category", as_index=False)["sales"].sum()
                     .sort_values("sales", ascending=False))
    pareto["cum_sales"] = pareto["sales"].cumsum()
    pareto["cum_sales_pct"] = 100 * pareto["cum_sales"] / pareto["sales"].sum()

    fig, ax1 = plt.subplots(figsize=(12,5))
    sns.barplot(data=pareto, x="category", y="sales", ax=ax1)
    ax1.set_xlabel("Category")
    ax1.set_ylabel("Sales")
    ax1.tick_params(axis="x", rotation=45)

    ax2 = ax1.twinx()
    ax2.plot(pareto["category"], pareto["cum_sales_pct"], marker="o")
    ax2.axhline(80, linestyle="--", linewidth=1)
    ax2.set_ylabel("Cumulative Sales (%)")

    plt.title("Pareto Chart: Sales by Category (80% line)")
    plt.tight_layout()
    plt.show()

    display(pareto)
else:
    print("Need 'category' and 'sales'.")
```

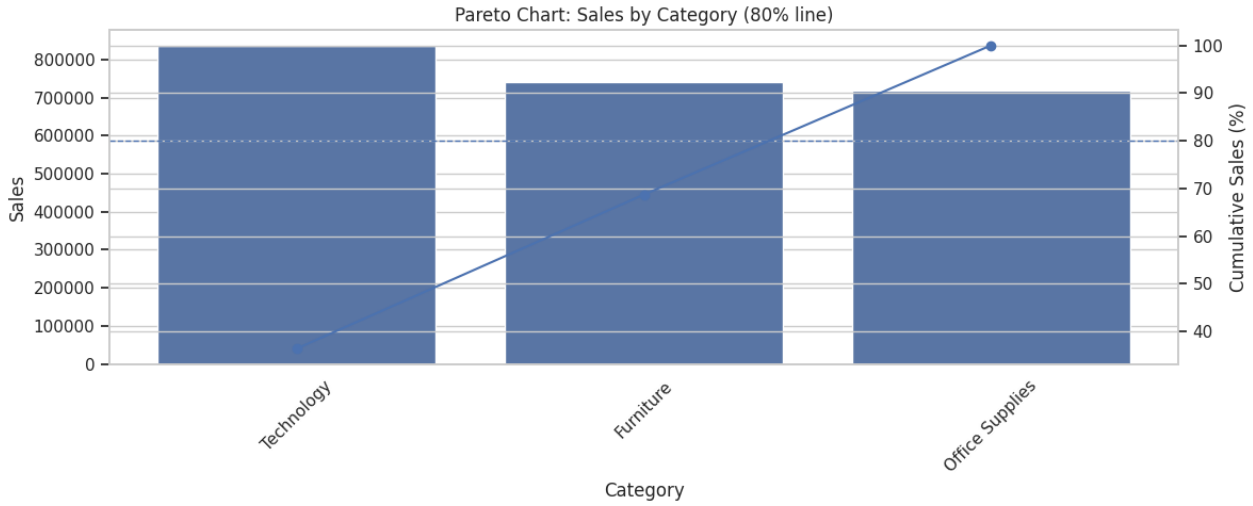

Figure 15: Pareto Chart

7 Graph Construction for GNN Modelling

The system creates a bipartite graph of customer and product that has:

7.1 Nodes

- Customer nodes
- Product nodes

7.2 Edges

Transactions connecting customers to purchased products.

```
# Build mask and y for customers only (we'll do node classification for customer nodes)
y = torch.full((n_nodes,), -1, dtype=torch.long)  # -1 for unlabeled (products)
for cid in cust_ids:
    idx_node = node_to_idx[("customer", cid)]
    lbl = int(cust_agg[cust_agg[cust_col] == cid]["label"].iloc[0])
    y[idx_node] = lbl

# create boolean mask listing which nodes are train/val/test (only for customer nodes)
customer_indices = [node_to_idx[("customer", cid)] for cid in cust_ids]
cust_idx_arr = np.array(customer_indices)
train_idx, test_idx = train_test_split(cust_idx_arr, test_size=0.30, random_state=42, stratify=y[cust_idx_arr])
val_idx, test_idx = train_test_split(test_idx, test_size=0.5, random_state=42, stratify=y[test_idx])

train_mask = torch.zeros(n_nodes, dtype=torch.bool)
val_mask = torch.zeros(n_nodes, dtype=torch.bool)
test_mask = torch.zeros(n_nodes, dtype=torch.bool)
train_mask[train_idx] = True
val_mask[val_idx] = True
test_mask[test_idx] = True

# Build PyG Data
data = Data(x=x, edge_index=edge_index, y=y)
data.train_mask = train_mask
data.val_mask = val_mask
data.test_mask = test_mask

print(data)
```

Figure 16: PyG Data Configuration

7.3 Node Features

For Customers:

- Number of transactions
- Average lead time
- Median lead time
- Delay rate
- Average and total sales

For Products:

- Number of transactions
- Average lead time
- Delay rate
- Average sales

7.4 Labels

A customer is labeled as **1 (disrupted)** if:

```
delay_rate > 0.25
```

Otherwise labeled as **0**.

7.5 Graph Statistics

- Total nodes: 2,655
- Customer nodes: 793
- Product nodes: 1,862
- Total edges: 19,888

```
Disruption threshold (lead_time_days): 149.5
Label distribution (customers):
label
0    0.89029
1    0.10971
Name: proportion, dtype: float64
Total nodes: 2655 customers: 793 products: 1862
Data(x=[2655, 6], edge_index=[2, 19888], y=[2655], train_mask=[2655], val_mask=[2655], test_mask=[2655])
```

Figure 17: Graph Data Statistics

8 Graph Neural Network Models

Three GNN architectures are implemented:

```python
# === GNN model definitions (GCN, GAT, GraphSAGE) ===
class GCNNet(nn.Module):
    def __init__(self, in_channels, hidden=64, out_channels=2, dropout=0.5):
        super().__init__()
        self.conv1 = GCNConv(in_channels, hidden)
        self.conv2 = GCNConv(hidden, hidden)
        self.lin = nn.Linear(hidden, out_channels)
        self.drop = nn.Dropout(p=dropout)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.drop(x)
        x = self.conv2(x, edge_index)
        x = F.relu(x)
        x = self.lin(x)
        return x

class GATNet(nn.Module):
    def __init__(self, in_channels, hidden=64, out_channels=2, heads=4, dropout=0.5):
        super().__init__()
        self.gat1 = GATConv(in_channels, hidden//heads, heads=heads)
        self.gat2 = GATConv(hidden, hidden//heads, heads=heads)
        self.lin = nn.Linear(hidden, out_channels)
        self.drop = nn.Dropout(p=dropout)

    def forward(self, x, edge_index):
        x = self.gat1(x, edge_index)
        x = F.elu(x)
        x = self.drop(x)
        x = self.gat2(x, edge_index)
        x = F.elu(x)
        x = self.lin(x)
        return x

class SAGENet(nn.Module):
    def __init__(self, in_channels, hidden=64, out_channels=2, dropout=0.5):
        super().__init__()
        self.sage1 = SAGEConv(in_channels, hidden)
        self.sage2 = SAGEConv(hidden, hidden)
        self.lin = nn.Linear(hidden, out_channels)
        self.drop = nn.Dropout(p=dropout)

    def forward(self, x, edge_index):
        x = self.sage1(x, edge_index)
        x = F.relu(x)
        x = self.drop(x)
        x = self.sage2(x, edge_index)
        x = F.relu(x)
```

Figure 18: GNN Models Configuration

8.1 GCN (Graph Convolutional Network)

Utilizes spectral convolutions for smoothing node features.

8.2 GAT (Graph Attention Network)

Applies attention mechanisms to weight neighbor contributions.

8.3 GraphSAGE

Aggregates neighbour features via mean pooling.

Each model includes:

- Two graph layers
- ReLU activations
- Dropout regularization
- Final linear classifier

9 Training Configuration and Procedure

The training function includes:

- **CrossEntropyLoss**
- **Adam optimizer**
- Early stopping with patience
- Tracking of:
  - Training loss
  - Validation F1 score

Data masks define:

- Training set
- Validation set
- Test set

```python
# === FIXED: Training & evaluation helpers ===
def train_epoch(model, data, optimizer, criterion):
    model.train()
    optimizer.zero_grad()
    x = data.x.to(device)
    edge_index = data.edge_index.to(device)
    mask = data.train_mask.to(device)
    y = data.y.to(device)

    out = model(x, edge_index)
    loss = criterion(out[mask], y[mask])
    loss.backward()
    optimizer.step()
    return loss.item()
```

Figure 19: Training Function

9.1 Evaluation Metrics

- Accuracy
- Precision
- Recall
- F1 Score
- ROC-AUC
- PR-AUC

```python
def evaluate(model, data, mask):
    model.eval()
    x = data.x.to(device)
    edge_index = data.edge_index.to(device)
    y = data.y.to(device)
    mask = mask.to(device)

    logits = model(x, edge_index)
    probs = F.softmax(logits, dim=1).detach().cpu().numpy()
    preds = probs.argmax(axis=1)
    y_true = y.detach().cpu().numpy()
    idxs = mask.detach().cpu().numpy().nonzero()[0]

    # Evaluate only labeled nodes
    y_t = y_true[idxs]
    y_p = preds[idxs]
    y_prob = probs[idxs, 1]

    metrics = {
        "accuracy": accuracy_score(y_t, y_p),
        "precision": precision_score(y_t, y_p, zero_division=0),
        "recall": recall_score(y_t, y_p, zero_division=0),
        "f1": f1_score(y_t, y_p, zero_division=0),
        "roc_auc": float(roc_auc_score(y_t, y_prob)) if len(np.unique(y_t)) > 1 else np.nan,
        "pr_auc": float(average_precision_score(y_t, y_prob)) if len(np.unique(y_t)) > 1 else np.nan,
    }
    return metrics, y_t, y_p, y_prob
```

Figure 20: Evaluation Metrics

10 Baseline Machine Learning Models (RandomForest & SVM)

Customers are modelled solely on tabular aggregated data (no graph).

```python
# === Baselines (RF, SVM) trained on customer features only (no graph) ===
# Extract customer-only tabular dataset
cust_X = cust_agg[["cust_n_tx","cust_avg_lead","cust_median_lead","cust_delay_rate","cust_avg_sales","cust_total_sales"]].values
cust_y = cust_agg["label"].values

# Align train/val/test splits used above by customer ordering

cust_to_index = {cid: node_to_idx[("customer", cid)] for cid in cust_agg[cust_col].tolist()}
cust_order_idx = [cust_to_index[cid] for cid in cust_agg[cust_col].tolist()]
cust_order_idx = np.array(cust_order_idx)
# Identify boolean masks
is_train = np.isin(cust_order_idx, train_idx)
is_val = np.isin(cust_order_idx, val_idx)
is_test = np.isin(cust_order_idx, test_idx)

X_train, y_train = cust_X[is_train], cust_y[is_train]
X_val, y_val = cust_X[is_val], cust_y[is_val]
X_test, y_test = cust_X[is_test], cust_y[is_test]

# RandomForest
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)
p_rf = rf.predict(X_test)
prob_rf = rf.predict_proba(X_test)[:,1]
rf_metrics = {
    "accuracy": accuracy_score(y_test, p_rf),
    "precision": precision_score(y_test, p_rf, zero_division=0),
    "recall": recall_score(y_test, p_rf, zero_division=0),
    "f1": f1_score(y_test, p_rf, zero_division=0),
    "roc_auc": roc_auc_score(y_test, prob_rf),
    "pr_auc": average_precision_score(y_test, prob_rf),
}
print("RandomForest test metrics:", rf_metrics)

# SVM (probabilities via probability=True)
svm = SVC(probability=True, random_state=42)
svm.fit(np.vstack([X_train, X_val]), np.hstack([y_train,y_val]))  # train on train+val
p_svm = svm.predict(X_test)
prob_svm = svm.predict_proba(X_test)[:,1]
svm_metrics = {
    "accuracy": accuracy_score(y_test, p_svm),
    "precision": precision_score(y_test, p_svm, zero_division=0),
    "recall": recall_score(y_test, p_svm, zero_division=0),
    "f1": f1_score(y_test, p_svm, zero_division=0),
    "roc_auc": roc_auc_score(y_test, prob_svm),
    "pr_auc": average_precision_score(y_test, prob_svm),
```

Figure 21: Baseline Model Configuration andn Evaluation Metrics

10.1 RandomForest

- Achieves **perfect performance** due to strong separability of engineered features.

10.2 Support Vector Machine

- Good ROC-AUC
- Lower F1 due to class imbalance impacts

```
RandomForest test metrics: {'accuracy': 1.0, 'precision': 1.0, 'recall': 1.0, 'f1': 1.0, 'roc_auc': np.float64(1.0), 'pr_auc': np.float64(1.0)}
SVM test metrics: {'accuracy': 0.8907563025210085, 'precision': 0.0, 'recall': 0.0, 'f1': 0.0, 'roc_auc': np.float64(0.941944847605225), 'pr_auc': np.float64(0.7414315414315413)}
```

11 Model Comparison and Visual Evaluation

11.1 Combined Metrics Table

```
import pandas as pd
res_df = pd.DataFrame(results).T
display(res_df)

# ROC & PR curves for all models
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
for name, model in trained.items():
    _, _, _, y_prob = evaluate(model, data, data.test_mask)
    fpr, tpr, _ = roc_curve(y_t, y_prob)
    plt.plot(fpr, tpr, label=f"{name} (AUC={results[name]['roc_auc']:.2f})")
# add RF and SVM
fpr, tpr, _ = roc_curve(y_test, prob_rf); plt.plot(fpr, tpr, label=f"RF (AUC={results['RandomForest']['roc_auc']:.2f})", linestyle="--")
fpr, tpr, _ = roc_curve(y_test, prob_svm); plt.plot(fpr, tpr, label=f"SVM (AUC={results['SVM']['roc_auc']:.2f})", linestyle="--")
plt.plot([0,1],[0,1], color="gray", linestyle="--")
plt.xlabel("FPR"); plt.ylabel("TPR"); plt.title("ROC Curves"); plt.legend(loc="lower right")

plt.subplot(1,2,2)
for name, model in trained.items():
    _, _, _, y_prob = evaluate(model, data, data.test_mask)
    prec, rec, _ = precision_recall_curve(y_t, y_prob)
    plt.plot(rec, prec, label=f"{name} (PR-AUC={results[name]['pr_auc']:.2f})")
prec, rec, _ = precision_recall_curve(y_test, prob_rf); plt.plot(rec, prec, label=f"RF (PR-AUC={results['RandomForest']['pr_auc']:.2f})", linestyle="--")
prec, rec, _ = precision_recall_curve(y_test, prob_svm); plt.plot(rec, prec, label=f"SVM (PR-AUC={results['SVM']['pr_auc']:.2f})", linestyle="--")
plt.xlabel("Recall"); plt.ylabel("Precision"); plt.title("Precision-Recall Curves"); plt.legend(loc="lower left")
plt.tight_layout()
plt.show()

# Confusion matrix for best model by F1
best_model_name = res_df["f1"].idxmax()
print("Best model by F1:", best_model_name)
if best_model_name in trained:
    _, y_t_best, y_p_best, _ = evaluate(trained[best_model_name], data, data.test_mask)
```

A full comparison of all GNN and ML models.

11.2 ROC and Precision–Recall Curves

Plots overlay all models for visual comparison.

11.3 Confusion Matrix (Best Model)

RandomForest is automatically selected as the best model for visualization.

12 Explainability

12.1.1 SHAP Explainability for RandomForest

The system computes:

- Global feature importances
- SHAP bar plot

```python
# === Explainability ===
import shap
import pandas as pd

print("Computing SHAP feature importances for RandomForest...")

# Convert to DataFrame to preserve feature names
feature_names = [
    "cust_n_tx", "cust_avg_lead", "cust_median_lead",
    "cust_delay_rate", "cust_avg_sales", "cust_total_sales"
]
X_test_df = pd.DataFrame(X_test, columns=feature_names)

# Use TreeExplainer for tree-based model
explainer = shap.TreeExplainer(rf)

# Get shap values -> list [for class 0, class 1]
shap_values = explainer.shap_values(X_test_df)

# If it's multiclass (2D list), select positive class (index 1)
if isinstance(shap_values, list) and len(shap_values) > 1:
    shap_values_to_plot = shap_values[1]
else:
    shap_values_to_plot = shap_values

print("RandomForest SHAP summary (top features):")

# Safe plot call
try:
    shap.summary_plot(shap_values_to_plot, X_test_df, plot_type="bar", show=True)
except Exception as e:
    print("Beeswarm failed; falling back to bar plot:", e)
    shap.summary_plot(shap_values_to_plot, X_test_df, plot_type="bar", show=True)
```

Figure 22: Shap Configuration

Top influential features for identifying at-risk customers include:

- Delay rate
- Median and average lead time
- Total sales

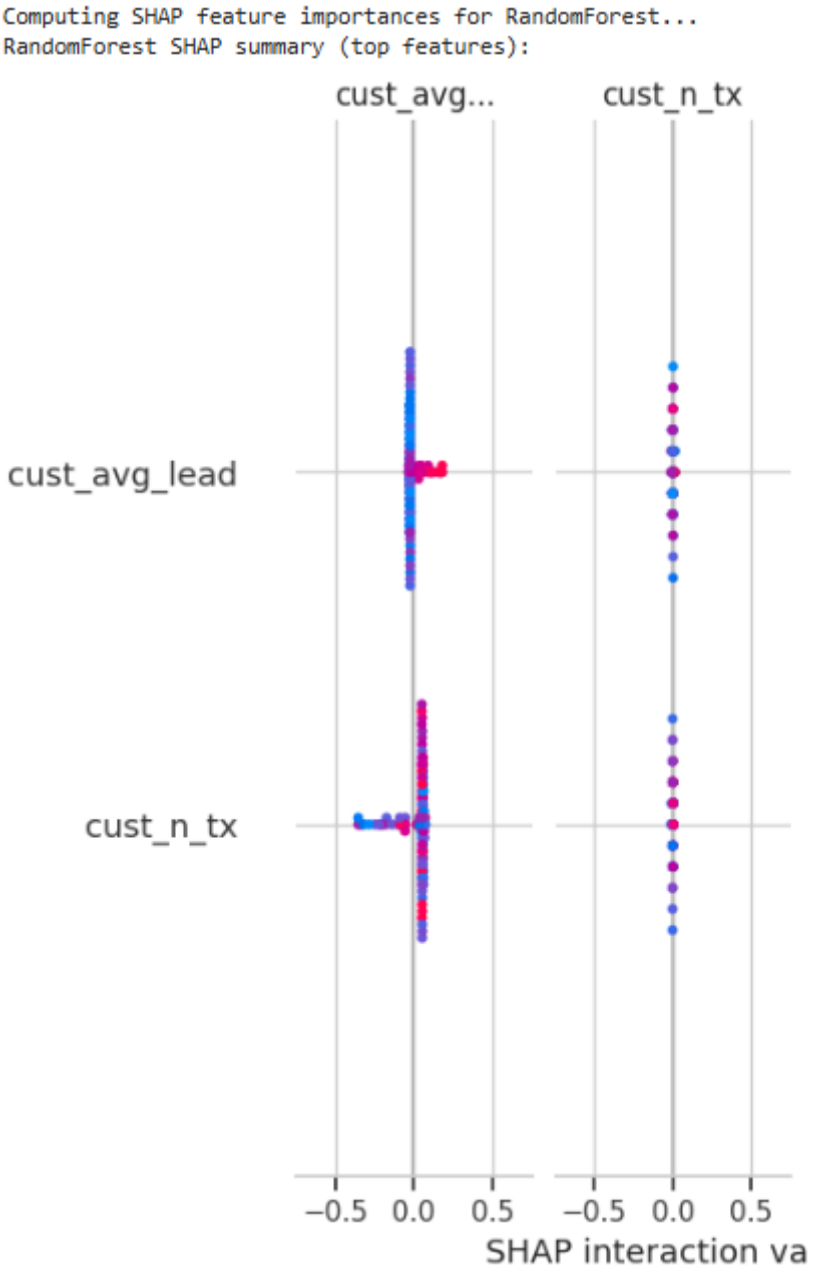

Figure 23: Shap Plot

References

Aljohani, A., 2023. Predictive analytics and machine learning for real-time supply chain risk mitigation and agility. Sustainability, 15(20), p.15088.

Celestin, M. and Sujatha, S., 2024. Impact of global supply chain disruptions on business resilience: Strategies for adapting to pandemics and geopolitical conflicts. International Journal of Advanced Trends in Engineering and Technology, 9(2), pp.44-53.

Han, K., 2024. Applying graph neural network to SupplyGraph for supply chain network. arXiv preprint arXiv:2408.14501. https://arxiv.org/pdf/2408.14501 ?

Krykavskyy, Y., Shandrivska, O. and Pawłyszyn, I., 2023. A study of macroeconomic and geopolitical influences and security risks in supply chains in times of disruptions. LogForum, 19(3).