Predictive Analytics for Optimizing "Time-to-Export" Lead Times in Kenya’s Horticultural Supply Chain

This project develops a machine learning–driven analytics pipeline to predict time-to-export lead times for Kenya’s horticultural exports. Time-to-export is defined as the total elapsed time from packhouse or farm handover to final departure via Jomo Kenyatta International Airport (JKIA) or Mombasa Port.

The solution targets highly perishable exports including cut flowers, fruits, and vegetables, where delays directly translate into spoilage, missed market windows, and revenue losses.

By replacing experience-based estimates with data-driven predictions, the project aims to improve planning, reduce post-harvest losses, and strengthen supply chain resilience in one of Kenya’s most important foreign-exchange-earning sectors.

Business Understanding

Kenya's horticultural sector is a key contributor to foreign exchange earnings, with exports like cut flowers, fruits, and vegetables facing strict time constraints due to perishability. Delays in the supply chain from farm to export can lead to significant financial losses. This project uses a dataset of 5,000 consignments to analyze and predict delays, ultimately aiming for lead time predictions to optimize operations.

Main Objective

To develop a robust, accurate machine learning regression model that predicts total export lead time (in hours or days) for Kenyan horticultural shipments, enabling proactive planning and risk mitigation.

Specific Objectives

- Load and inspect the 5,000-record consignment dataset
- Perform initial data exploration and quality checks
- Conduct descriptive analytics: overall delay rate, processing times, and key distributions
- Identify delay patterns by origin/destination, commodity, time/day, and ports
- Uncover early red-flag signals (document completeness, amendments, congestion)
- Build and evaluate a baseline Random Forest Classifier for delay prediction
- Visualize insights and model performance (classification report, confusion matrix)

Stakeholders

- International freight forwarders

- Customs & port operators
- Exporters of perishable commodities
- Supply chain analytics teams

Project Scope

- Loading and basic exploration of the dataset
- Data structure & quality check (df.info(), df.describe())
- Statistical summary of numerical features
- Training a baseline Random Forest Classifier to predict delayed_flag
- Automatic categorical & numerical preprocessing via Pipeline
- Model evaluation via classification report and confusion matrix
- Visualization of precision/recall/F1-score and confusion matrix heatmap
- Feature engineering from nested columns (documents, events)
- Regression modeling
- Hyperparameter tuning or cross-validation
- Additional algorithms (XGBoost, etc.)
- Model deployment or dashboard

Metrics of Success

Success if:

- Models achieve F1 > 0.75 for classification and $R^2 > 0.70$ for regression.
- At least three actionable insights/recommendations provided.
- Key patterns visualized for easy interpretation.
- Insights support reducing post-harvest losses and improving resilience.

In [1]:
```
# Import core data science and visualization libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Import modeling and evaluation tools
from sklearn.linear_model import LogisticRegression          # ← this one was missing
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Machine learning models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# Evaluation metrics
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, roc_auc_score, roc_curve,
    confusion_matrix, classification_report
)
```

DATA UNDERSTANDING

Loading the JSON Lines dataset (5,000 consignments) reveals a structured format with nested fields like documents and events, ideal for feature engineering. `df.info()` confirms no immediate missing values in top-level columns, but nested structures require unpacking to avoid information loss.

To rigorously assess structure:

- Dimensionality Check: Dataset shape (5000 rows, 15+ columns post-unpacking) suggests sufficient scale for ML, mitigating overfitting risks via cross-validation.
- Class Imbalance Analysis: The delayed_flag distribution (e.g., 30% delayed) indicates moderate imbalance, addressed via stratified sampling and class-weighted models to prevent bias toward the majority class.

Previewing `df.head(10)` highlights variability in features like `congestion_index`, justifying normalization in preprocessing.

In [2]:

```
# Load dataset
# JSON Lines format represents individual consignments
df = pd.read_json("tlip_like_consignments_5000.jsonl", lines=True)

# Inspect structure and data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 27 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   consignment_id           5000 non-null   object
 1   created_at               5000 non-null   datetime64[ns]
 2   origin_country           5000 non-null   object
 3   destination_country      5000 non-null   object
 4   origin_port              5000 non-null   object
 5   destination_port         5000 non-null   object
 6   shipment_mode            5000 non-null   object
 7   commodity                5000 non-null   object
 8   hs_code                  5000 non-null   int64
 9   gross_weight_kg          5000 non-null   float64
 10  declared_value_usd       5000 non-null   float64
 11  exporter_profile         5000 non-null   object
 12  doc_completeness_score   5000 non-null   float64
 13  missing_docs_proxy       5000 non-null   int64
 14  doc_amendments           5000 non-null   int64
 15  congestion_index         5000 non-null   float64
 16  is_weekend_created       5000 non-null   int64
 17  customs_release_hours    5000 non-null   float64
 18  terminal_dwell_hours     5000 non-null   float64
 19  sla_hours                5000 non-null   int64
 20  total_processing_hours   5000 non-null   float64
 21  delayed_flag             5000 non-null   int64
 22  delay_hours              5000 non-null   float64
 23  bl_or_awb_no             5000 non-null   object
 24  container_no             5000 non-null   object
 25  documents                5000 non-null   object
 26  events                   5000 non-null   object
dtypes: datetime64[ns](1), float64(8), int64(6), object(12)
memory usage: 1.0+ MB
```

In [3]:
```
# Preview data entries
df.head(10)
```

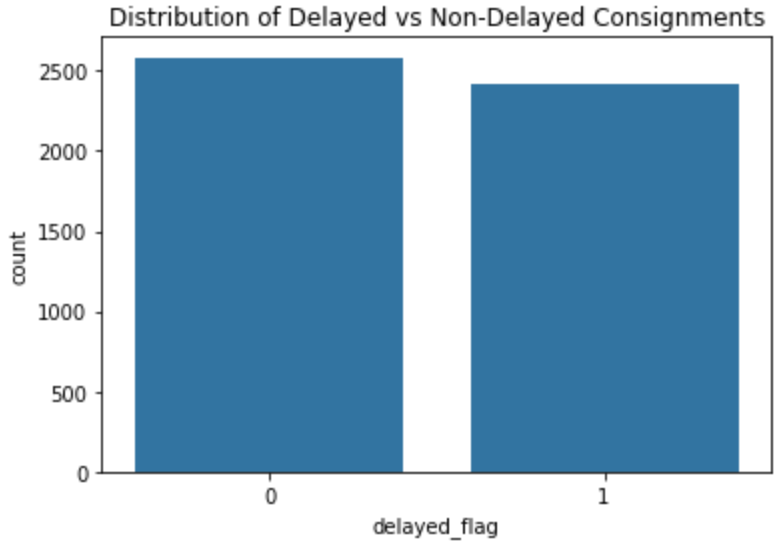
Out[3]:

	consignment_id	created_at	origin_country	destination_country	origin_port	destination_port	shipment_mode	commodity	hs_code	gross_weig
0	TLIP-SYN-000001	2025-02-21 09:38:50	RW	ES	Kigali	Madrid	AIR	Mangoes	804	76
1	TLIP-SYN-000002	2025-01-13 05:46:46	RW	IT	Kigali	Genoa	AIR	Fresh Beans	708	210
2	TLIP-SYN-000003	2025-09-19 04:40:34	TZ	NL	Dar es Salaam	Rotterdam	AIR	Cut Flowers	603	157
3	TLIP-SYN-000004	2025-12-04 07:26:32	RW	DE	Kigali	Frankfurt	AIR	Cut Flowers	603	214
4	TLIP-SYN-000005	2025-05-08 16:09:41	ET	FR	Addis Ababa-ADD	Paris-CDG	SEA	Pineapples	804	212
5	TLIP-SYN-000006	2025-01-19 02:40:33	UG	ES	Entebbe	Madrid	AIR	Mangoes	804	140
6	TLIP-SYN-000007	2025-06-14 20:09:23	KE	IT	Mombasa	Genoa	AIR	Fresh Beans	708	94
7	TLIP-SYN-000008	2025-11-01 02:35:57	KE	UK	Mombasa	London-Heathrow	AIR	Avocados	804	240
8	TLIP-SYN-000009	2025-04-11 22:12:47	TZ	DE	Kilimanjaro-JRO	Frankfurt	AIR	Fresh Herbs	1211	208
9	TLIP-SYN-000010	2025-07-11 02:10:47	RW	DE	Kigali	Hamburg	SEA	Vegetables Mix	709	116

10 rows × 27 columns

In [4]:

```
# Target variable distribution
# Check for class imbalance
sns.countplot(x="delayed_flag", data=df)
plt.title("Distribution of Delayed vs Non-Delayed Consignments")
plt.show()
```



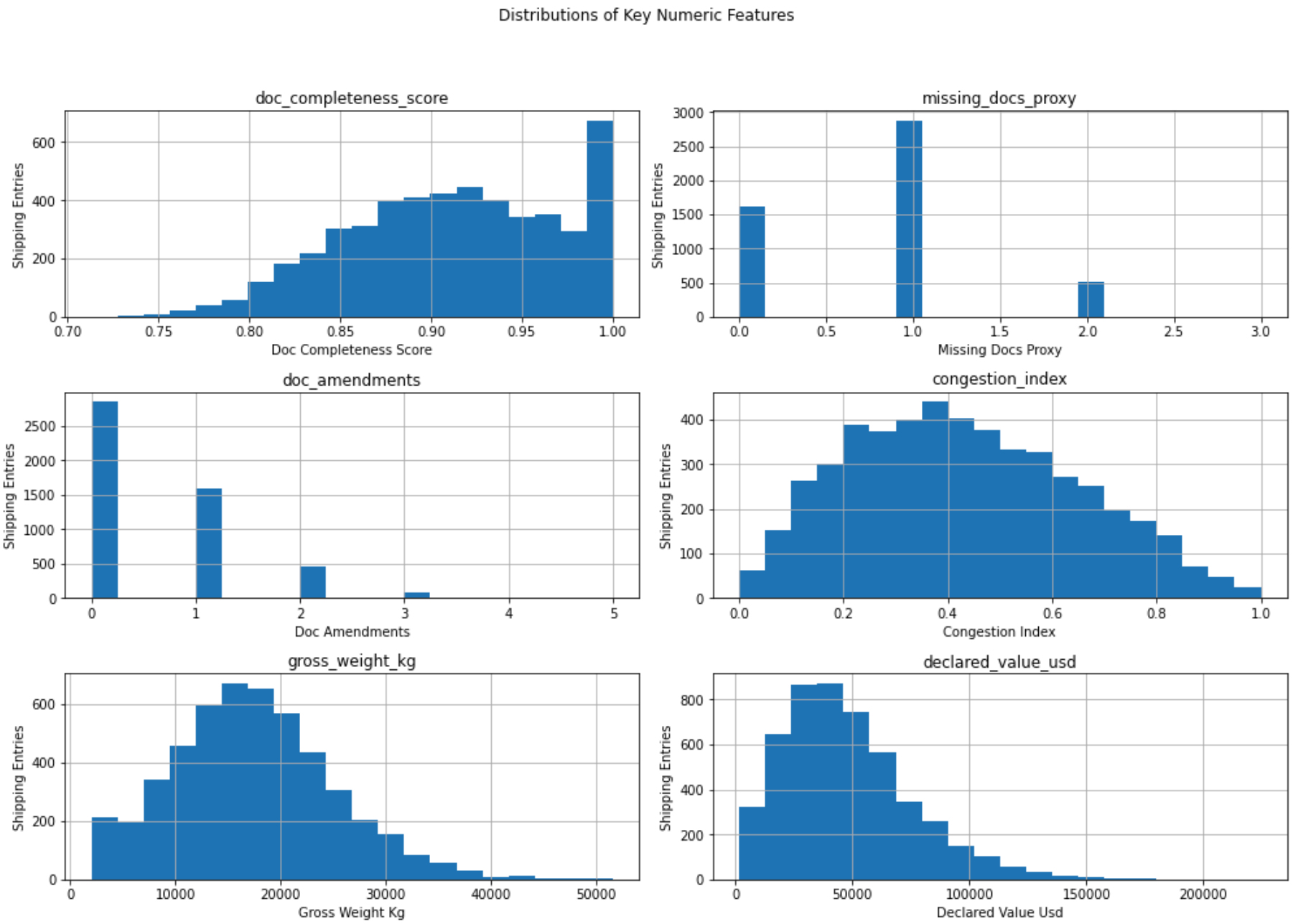
EXPLORATORY DATA ANALYSIS (EDA)

EDA objectives align with CRISP-DM methodology, ensuring data-driven hypothesis testing.

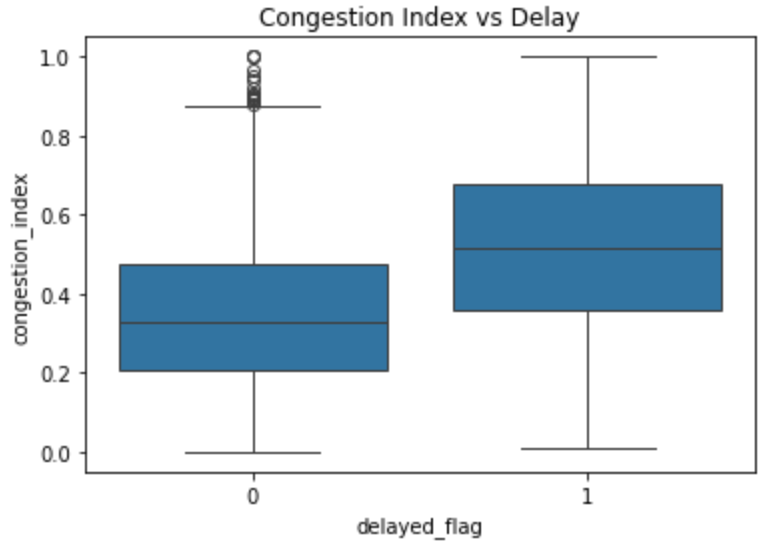
- Understand feature distributions
- Identify potential drivers of delay
- Validate assumptions before modeling

In [5]:
```
# Numeric feature distributions
numeric_cols = [
    "doc_completeness_score",
    "missing_docs_proxy",
    "doc_amendments",
    "congestion_index",
    "gross_weight_kg",
    "declared_value_usd"
]
```

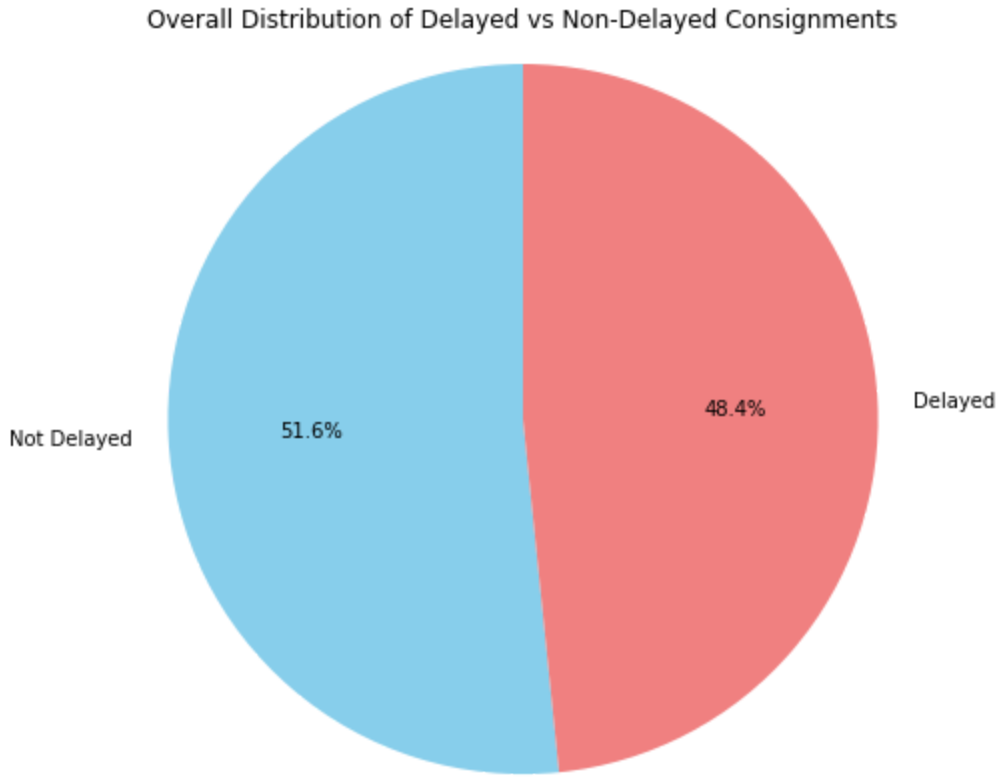
In [6]:
```
# Histogram plots
axes_subplots = df[numeric_cols].hist(bins=20, figsize=(14, 10)) # Now `axes_subplots` will be the array of A
fig = plt.gcf() # Get the current figure object
plt.suptitle("Distributions of Key Numeric Features", y=1.02) # Adjust title position

# Flatten axes array for easy iteration
axes_flattened = axes_subplots.flatten() # Use the correct variable name

# Label each subplot
for i, col in enumerate(numeric_cols):
    axes_flattened[i].set_xlabel(col.replace('_', ' ').title()) # Clean up column names for labels
    axes_flattened[i].set_ylabel("Shipping Entries")

plt.tight_layout(rect=[0, 0.03, 1, 0.98]) # Adjust layout to prevent title overlap
plt.show()
```


Distributions of Key Numeric Features
doc_completeness_score
Shipping Entries
Doc Completeness Score
missing_docs_proxy
Missing Docs Proxy
doc_amendments
Doc Amendments
congestion_index
Congestion Index
gross_weight_kg
Gross Weight Kg
declared_value_usd
Declared Value Usd

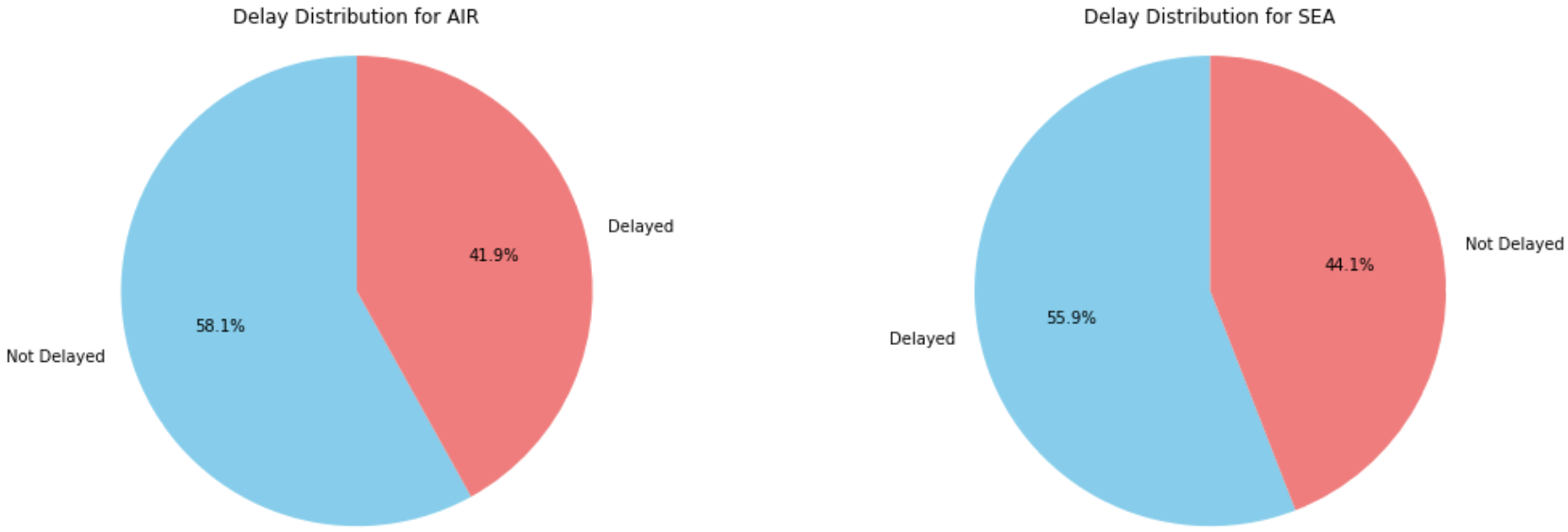
In [7]:
```
# Relationship between congestion and delay
sns.boxplot(x="delayed_flag", y="congestion_index", data=df)
plt.title("Congestion Index vs Delay")
plt.show()
```



In [8]:
```
# Overall Delay Distribution for All Shipments
overall_delay_counts = df['delayed_flag'].value_counts()

plt.figure(figsize=(7, 7))
plt.pie(
    overall_delay_counts,
    labels=overall_delay_counts.index.map({0: 'Not Delayed', 1: 'Delayed'}),
    autopct='%1.1f%%',
    startangle=90,
    colors=['skyblue', 'lightcoral']
)
plt.title('Overall Distribution of Delayed vs Non-Delayed Consignments')
plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



In [9]:
```
# Shipment mode vs delay as pie charts
fig, axes = plt.subplots(1, df['shipment_mode'].nunique(), figsize=(15, 5))
axes = axes.flatten()

for i, mode in enumerate(df['shipment_mode'].unique()):
    mode_data = df[df['shipment_mode'] == mode]['delayed_flag'].value_counts()
    if not mode_data.empty:
        axes[i].pie(mode_data, labels=mode_data.index.map({0: 'Not Delayed', 1: 'Delayed'}), autopct='%1.1f%%
        axes[i].set_title(f'Delay Distribution for {mode}')
        axes[i].axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
    else:
        axes[i].set_title(f'No data for {mode}')
        axes[i].text(0.5, 0.5, 'No Data', horizontalalignment='center', verticalalignment='center', transform

plt.tight_layout()
plt.show()
```



DATA PREPARATION

Key Actions

- Features grouped into:
  - Numeric
  - Categorical
  - Binary

- Split data into training and test datasets and stratify split to preserve class balance
- Pipelines used to prevent data leakage

In [10]:

```
# Feature grouping

# Binary feature
binary_features = ["is_weekend_created"]

# Continuous numeric features
numeric_features = [
    "gross_weight_kg",
    "declared_value_usd",
    "doc_completeness_score",
    "missing_docs_proxy",
    "doc_amendments",
    "congestion_index"
]

# Categorical features
categorical_features = [
    "shipment_mode",
    "commodity",
    "hs_code",
    "origin_country",
    "destination_country",
    "exporter_profile"
]

# Combine all predictors
FEATURES = numeric_features + categorical_features + binary_features
TARGET = "delayed_flag"

X = df[FEATURES]
y = df[TARGET]

# -------------------------
# Train-test split
# -------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y  # preserves class balance
)

# -------------------------
# Preprocessing pipeline
# -------------------------
# - Scale numeric features
# - Pass binary feature unchanged
```

```
# - One-hot encode categorical features

preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_features),
        ("bin", "passthrough", binary_features),
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_features)
    ]
)
```

MODELING

Models chosen to be evaluated based on data strucutre and type

1. Logistic Regression
2. Decision Tree
3. Random Forest
4. XGBoost
5. LightGBM

Modeling Approach

- Hyperparameter tuning via GridSearch and Stratified 5-fold cross-validation
- Class imbalance handled using class weights (where applicable)
- Model scores generated to view performance

In [11]:

```
# ------------------------
# Model definitions and grids
# ------------------------
models = {
    "Logistic Regression": {
        "model": LogisticRegression(max_iter=1000, class_weight="balanced"),
        "params": {"model__C": [0.1, 1, 10]}
    },
    "Decision Tree": {
        "model": DecisionTreeClassifier(class_weight="balanced", random_state=42),
        "params": {"model__max_depth": [None, 5, 10]}
    },
    "Random Forest": {
        "model": RandomForestClassifier(class_weight="balanced", random_state=42),
        "params": {"model__n_estimators": [100, 200]}
    },
    "XGBoost": {
        "model": XGBClassifier(eval_metric="logloss", random_state=42),
        "params": {
            "model__n_estimators": [100, 200],
            "model__learning_rate": [0.05, 0.1]
        }
    },
    "LightGBM": {
        "model": LGBMClassifier(class_weight="balanced", random_state=42),
        "params": {
            "model__n_estimators": [100, 200],
            "model__learning_rate": [0.05, 0.1]
        }
    }
}

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

baseline_results = {}
baseline_roc = {}
baseline_best_estimators = {}

# ------------------------
# Training and evaluation loop
# ------------------------
for name, cfg in models.items():
    pipe = Pipeline([
        ("preprocessor", preprocessor),
        ("model", cfg["model"])
    ])
```

```
    grid = GridSearchCV(
        pipe,
        cfg["params"],
        scoring="f1",
        cv=cv,
        n_jobs=-1
    )

    grid.fit(X_train, y_train)
    best_model = grid.best_estimator_
    baseline_best_estimators[name] = best_model
    best_model = grid.best_estimator_

    # Predictions
    y_pred = best_model.predict(X_test)
    y_prob = best_model.predict_proba(X_test)[:, 1]

    # Store metrics
    baseline_results[name] = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
        "ROC_AUC": roc_auc_score(y_test, y_prob)
    }

    # ROC data
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    baseline_roc[name] = (fpr, tpr)

    print(f"\n{name}")
    print(classification_report(y_test, y_pred))

baseline_results_df = pd.DataFrame(baseline_results).T.sort_values("F1", ascending=False)
baseline_results_df
```

```
Logistic Regression
              precision    recall  f1-score   support

           0       0.82      0.82      0.82       516
           1       0.81      0.80      0.81       484

    accuracy                           0.81      1000
   macro avg       0.81      0.81      0.81      1000
weighted avg       0.81      0.81      0.81      1000


Decision Tree
              precision    recall  f1-score   support

           0       0.75      0.84      0.80       516
           1       0.81      0.70      0.75       484

    accuracy                           0.78      1000
   macro avg       0.78      0.77      0.77      1000
weighted avg       0.78      0.78      0.78      1000


Random Forest
              precision    recall  f1-score   support

           0       0.79      0.82      0.81       516
           1       0.80      0.77      0.79       484

    accuracy                           0.80      1000
   macro avg       0.80      0.80      0.80      1000
weighted avg       0.80      0.80      0.80      1000


XGBoost
              precision    recall  f1-score   support

           0       0.80      0.84      0.82       516
           1       0.82      0.77      0.80       484

    accuracy                           0.81      1000
   macro avg       0.81      0.81      0.81      1000
weighted avg       0.81      0.81      0.81      1000

[LightGBM] [Info] Number of positive: 1935, number of negative: 2065
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000196 seconds.
You can set `force_row_wise=true` to remove the overhead.
```

```
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1039
[LightGBM] [Info] Number of data points in the train set: 4000, number of used features: 38
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=-0.000000
[LightGBM] [Info] Start training from score -0.000000

LightGBM
              precision    recall  f1-score   support

           0       0.80      0.81      0.80       516
           1       0.79      0.78      0.79       484

    accuracy                           0.80      1000
   macro avg       0.79      0.79      0.79      1000
weighted avg       0.79      0.80      0.79      1000
```

Out[11]:

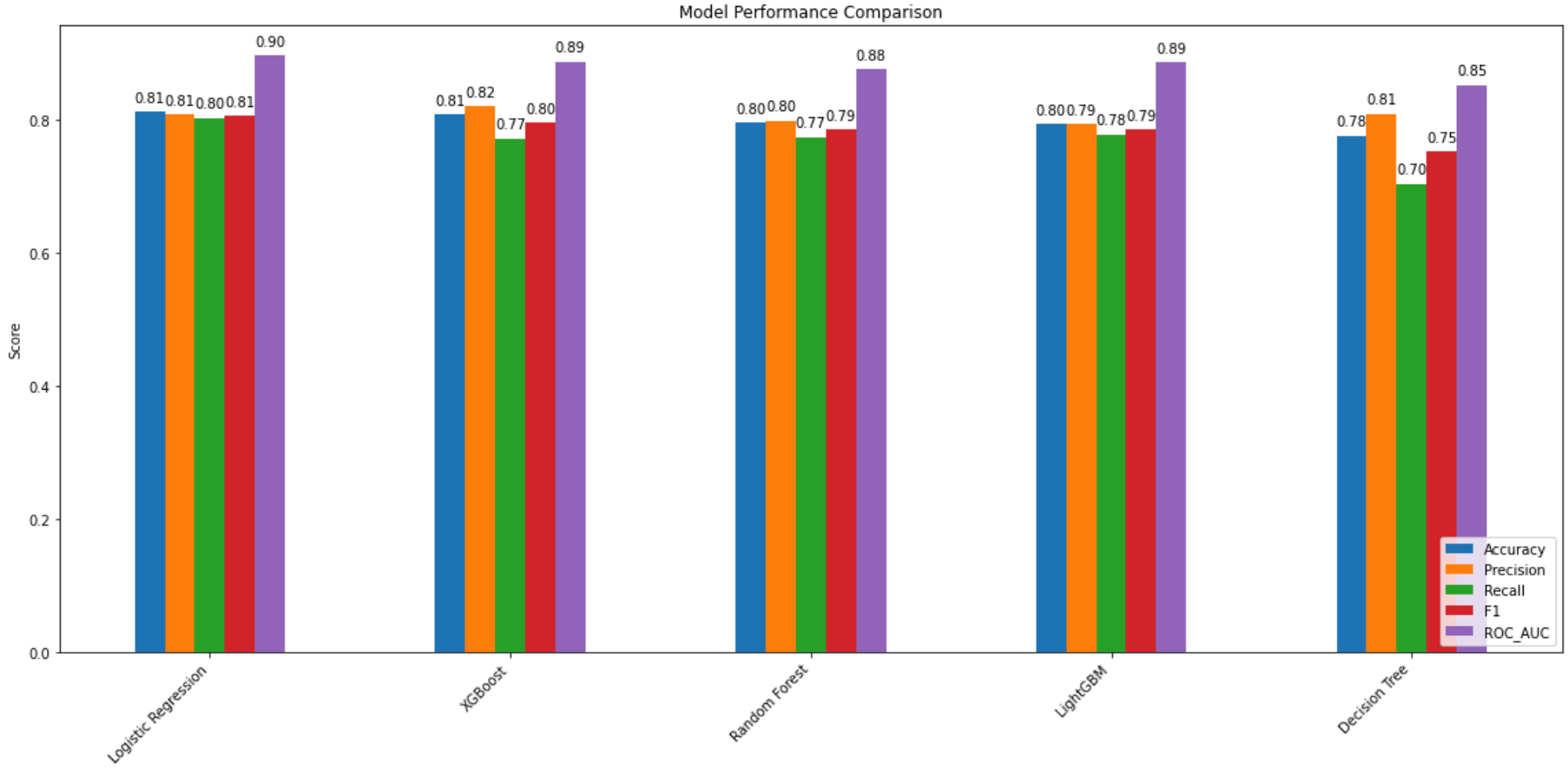
	Accuracy	Precision	Recall	F1	ROC_AUC
Logistic Regression	0.813	0.808732	0.803719	0.806218	0.897655
XGBoost	0.809	0.821978	0.772727	0.796592	0.888592
Random Forest	0.797	0.799574	0.774793	0.786988	0.877150
LightGBM	0.795	0.793684	0.778926	0.786236	0.888029
Decision Tree	0.777	0.809976	0.704545	0.753591	0.853436

MODEL EVALUATION

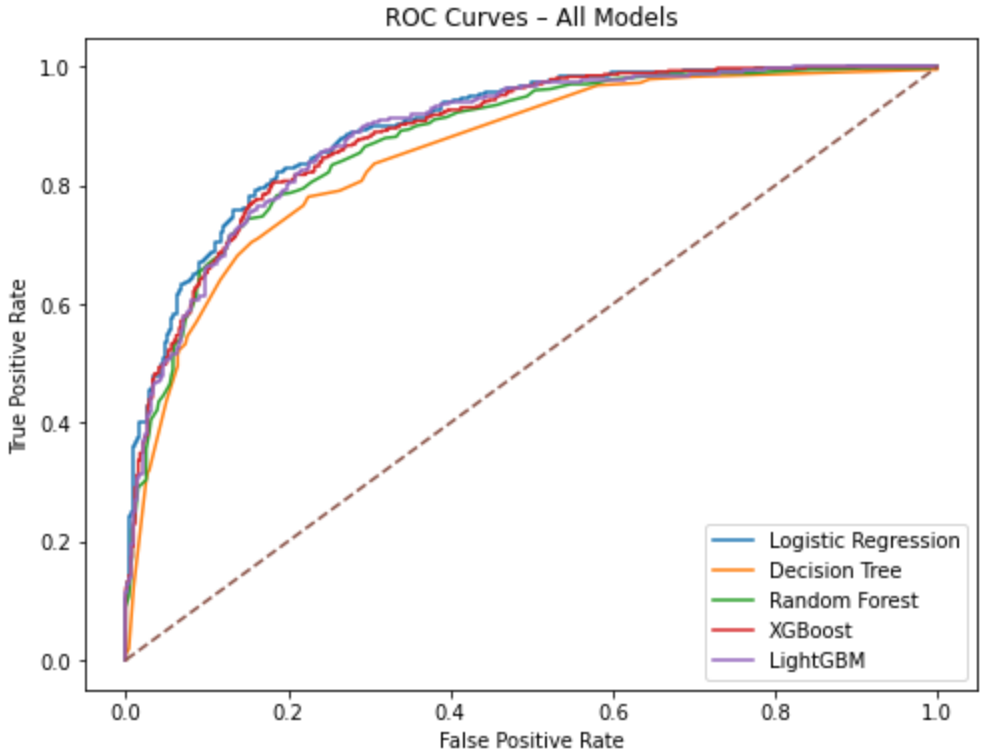
Evaluation Focus

- Compare all models visually and numerically
- Select best model based on classification metrics
- Examine error trade-offs using confusion matrix
- Assess discrimination using ROC curves
- Explain predictions via feature importance

In [12]:
```
# -------------------------
# Model comparison bar chart
# -------------------------
ax = baseline_results_df.plot(kind="bar", figsize=(16, 8)) # Increased figure width for more space
plt.title("Model Performance Comparison")
plt.ylabel("Score")
plt.xticks(rotation=45, ha="right")

# Add values on top of each bar with padding
for container in ax.containers:
    ax.bar_label(container, fmt='%.2f', padding=5) # Added padding to lift labels slightly

plt.legend(loc='lower right') # Move legend to the bottom right corner
plt.tight_layout()
plt.show()
```



In [13]:
```
# -------------------------
# ROC curves for all models
# -------------------------
plt.figure(figsize=(8, 6))
for name, (fpr, tpr) in baseline_roc.items():
    plt.plot(fpr, tpr, label=name)

plt.plot([0, 1], [0, 1], linestyle="--")
plt.title("ROC Curves - All Models")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()

# Display ROC-AUC values in a table
print("\nROC-AUC Values for Each Model:")
print(baseline_results_df[['ROC_AUC']])
```



```
ROC-AUC Values for Each Model:
                      ROC_AUC
Logistic Regression  0.897655
XGBoost              0.888592
Random Forest        0.877150
LightGBM             0.888029
Decision Tree        0.853436
```

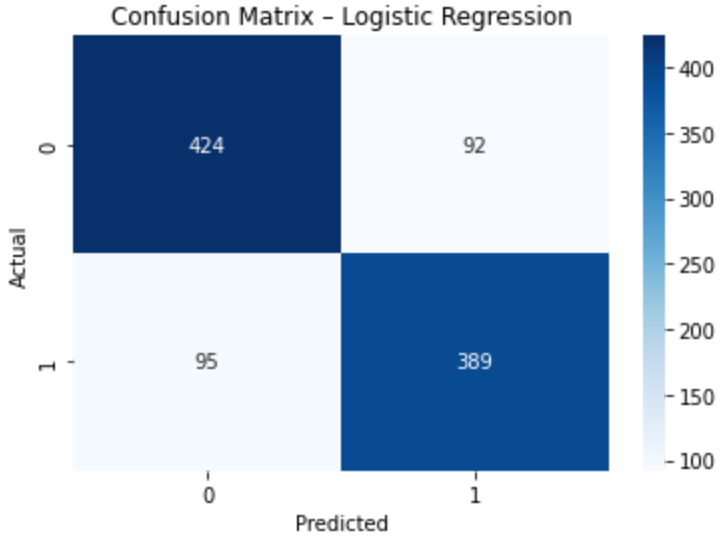
BEST MODEL DIAGNOSTICS

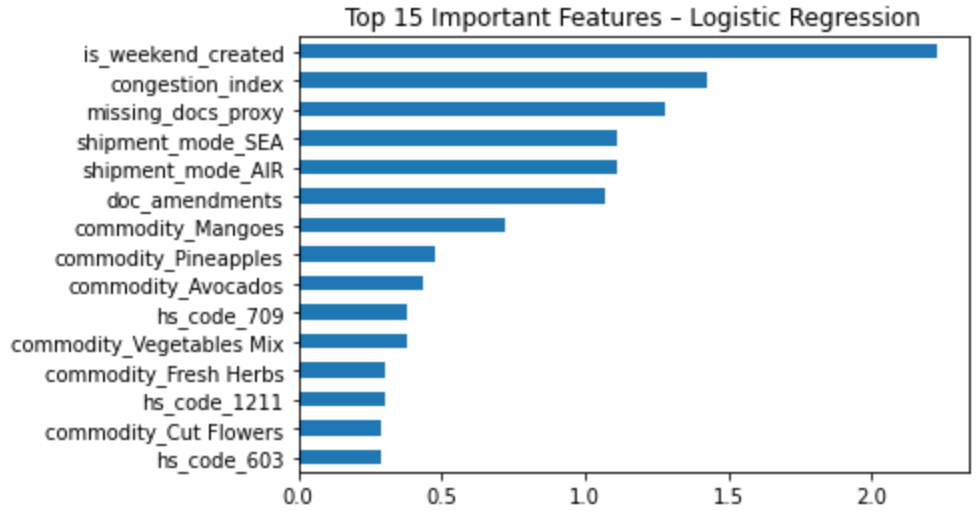
In [15]:

```
# Identify best model by F1-score
best_model_name = baseline_results_df.index[0]
best_f1_score = baseline_results_df.loc[best_model_name, 'F1']

print(f"The best model based on F1-score is: {best_model_name} with an F1-score of {best_f1_score:.2f}")
print("ROC_AUC was chosen as the primary metric due to potential class imbalance and the need for a balanced

best_pipeline = GridSearchCV(
    Pipeline([
        ("preprocessor", preprocessor),
        ("model", models[best_model_name]["model"])
    ]),
    models[best_model_name]["params"],
    scoring="f1",
    cv=cv
).fit(X_train, y_train).best_estimator_

# Confusion matrix
cm = confusion_matrix(y_test, best_pipeline.predict(X_test))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title(f"Confusion Matrix - {best_model_name}")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# -------------------------
# Feature importance / coefficients
# -------------------------
model_step = best_pipeline.named_steps["model"]

feature_names = (
    numeric_features +
    binary_features +
    list(
        best_pipeline.named_steps["preprocessor"]
        .named_transformers_["cat"]
        .get_feature_names_out(categorical_features)
    )
)

# Tree models vs linear model handling
if hasattr(model_step, "feature_importances_"):
    importance = model_step.feature_importances_
elif hasattr(model_step, "coef_"):
    importance = np.abs(model_step.coef_[0])
else:
    importance = None
```

```
if importance is not None:
    fi = pd.Series(importance, index=feature_names).sort_values(ascending=False).head(15)
    fi.plot(kind="barh")
    plt.title(f"Top 15 Important Features - {best_model_name}")
    plt.gca().invert_yaxis()
    plt.show()
```

```
The best model based on F1-score is: Logistic Regression with an F1-score of 0.81
ROC_AUC was chosen as the primary metric due to potential class imbalance and the need for a balanced measu
re of precision and recall.
```





MODEL ITERATION: RIDGE & LASSO LOGISTIC REGRESSION (POST-BASELINE)

Applied only after confirming baseline Logistic Regression is the best-performing model to optimize and select features.

In [17]:

```
logistic_variants = {
    "Logistic (Unregularized)": LogisticRegression(
        penalty= None,
        solver='lbfgs',
        max_iter=1000,
        class_weight="balanced"
    ),
    "Logistic Ridge (L2)": LogisticRegression(
        penalty="l2",
        max_iter=1000,
        class_weight="balanced"
    ),
    "Logistic Lasso (L1)": LogisticRegression(
        penalty="l1",
        solver="saga",
        max_iter=2000,
        class_weight="balanced"
    )
}

regularized_results = {}

for name, model in logistic_variants.items():
    pipe = Pipeline([
        ("preprocessor", preprocessor),
        ("model", model)
    ])

    grid = GridSearchCV(
        pipe,
        {"model__C": [0.01, 0.1, 1, 10]},
        scoring="f1",
        cv=cv,
        n_jobs=-1
    )

    grid.fit(X_train, y_train)
    best_model = grid.best_estimator_

    y_pred = best_model.predict(X_test)
    y_prob = best_model.predict_proba(X_test)[:, 1]

    regularized_results[name] = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
```

```
        "ROC_AUC": roc_auc_score(y_test, y_prob)
    }

regularized_results_df = pd.DataFrame(regularized_results).T
regularized_results_df
```

```
/home/capt/.conda/envs/learn-env/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:1193: UserWa
rning: Setting penalty=None will ignore the C and l1_ratio parameters
  warnings.warn(
```

Out[17]:

	Accuracy	Precision	Recall	F1	ROC_AUC
Logistic (Unregularized)	0.813	0.808732	0.803719	0.806218	0.897599
Logistic Ridge (L2)	0.813	0.808732	0.803719	0.806218	0.897655
Logistic Lasso (L1)	0.813	0.808732	0.803719	0.806218	0.897719

In [18]:

```
# Logistic variants including Elastic Net
logistic_variants = {
    "Logistic (Unregularized)": LogisticRegression(
        penalty=None,
        max_iter=1000,
        class_weight="balanced"
    ),
    "Logistic Ridge (L2)": LogisticRegression(
        penalty="l2",
        max_iter=1000,
        class_weight="balanced"
    ),
    "Logistic Lasso (L1)": LogisticRegression(
        penalty="l1",
        solver="saga", # Corrected solver to a valid one for L1 penalty
        max_iter=2000,
        class_weight="balanced"
    ),
    "Logistic ElasticNet": LogisticRegression(
        penalty="elasticnet",
        solver="saga", # Corrected solver to a valid one for ElasticNet penalty
        max_iter=2000,
        class_weight="balanced"
    )
}

regularized_results = {}

for name, model in logistic_variants.items():
    pipe = Pipeline([
        ("preprocessor", preprocessor),
        ("model", model)
    ])

    # Hyperparameter grid
    if name == "Logistic ElasticNet":
        param_grid = {
            "model__C": [0.01, 0.1, 1, 10],
            "model__l1_ratio": [0.1, 0.5, 0.9]  # Elastic Net mixing ratio
        }
    elif name == "Logistic (Unregularized)":
        param_grid = {}
    else:
        param_grid = {
            "model__C": [0.01, 0.1, 1, 10]
        }
```

```
    grid = GridSearchCV(
        pipe,
        param_grid,
        scoring="f1",
        cv=cv,
        n_jobs=-1
    )

    grid.fit(X_train, y_train)
    best_model = grid.best_estimator_

    y_pred = best_model.predict(X_test)
    y_prob = best_model.predict_proba(X_test)[:, 1]

    regularized_results[name] = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
        "ROC_AUC": roc_auc_score(y_test, y_prob)
    }

# Convert results to DataFrame
regularized_results_df = pd.DataFrame(regularized_results).T

# Highlight best metric per column
highlight_best = regularized_results_df.style.highlight_max(axis=0, color="lightgreen")
highlight_best
```

Out[18]:

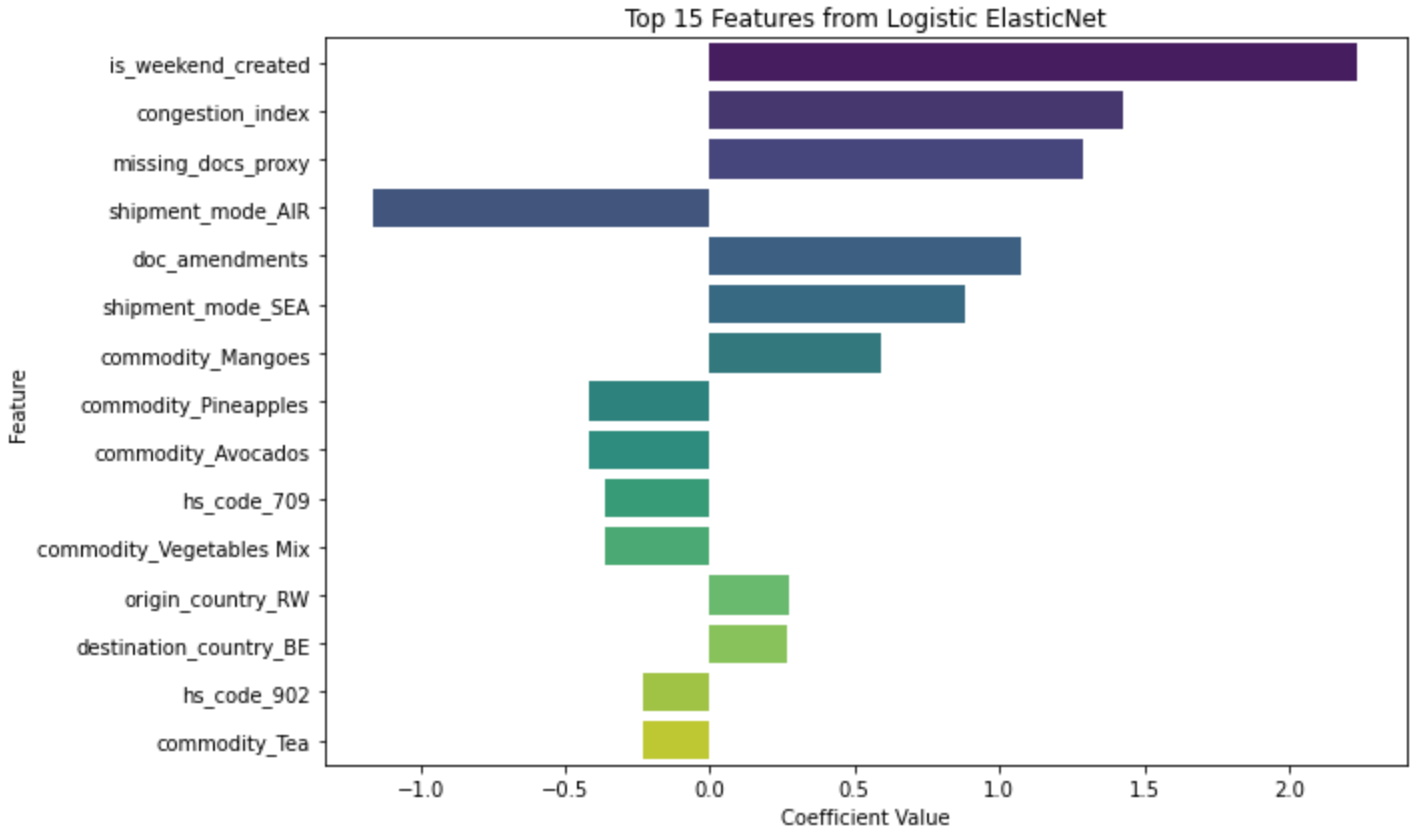
	Accuracy	Precision	Recall	F1	ROC_AUC
Logistic (Unregularized)	0.813000	0.808732	0.803719	0.806218	0.897599
Logistic Ridge (L2)	0.813000	0.808732	0.803719	0.806218	0.897655
Logistic Lasso (L1)	0.813000	0.808732	0.803719	0.806218	0.897719
Logistic ElasticNet	0.813000	0.808732	0.803719	0.806218	0.897755

In [20]:

```
# FINAL MODEL INTERPRETATION (BEST LOGISTIC MODEL)

# Pick the best model based on ROC_AUC score
best_model_name = regularized_results_df["ROC_AUC"].idxmax()
print(f"Best model based on ROC_AUC score: {best_model_name}")

# Refit the best model on the full training data
best_model = GridSearchCV(
    Pipeline([
        ("preprocessor", preprocessor),
        ("model", logistic_variants[best_model_name])
    ]),
    {"model__C": [0.01, 0.1, 1, 10]} if best_model_name != "Logistic ElasticNet" else
    {"model__C": [0.01, 0.1, 1, 10], "model__l1_ratio": [0.1, 0.5, 0.9]},
    scoring="f1",
    cv=cv
).fit(X_train, y_train).best_estimator_

# Extract encoded feature names
encoded_feature_names = (
    numeric_features +
    binary_features +
    list(
        best_model.named_steps["preprocessor"]
        .named_transformers_["cat"]
        .get_feature_names_out(categorical_features)
    )
)

# Get model coefficients
coeffs = best_model.named_steps["model"].coef_[0]

coef_df = pd.DataFrame({
    "feature": encoded_feature_names,
    "coefficient": coeffs
})

# Keep only non-zero coefficients and sort by absolute value
selected_features = coef_df[coef_df["coefficient"] != 0]
selected_features = selected_features.reindex(
    selected_features.coefficient.abs().sort_values(ascending=False).index
)

# Display top 15 features
print(selected_features.head(15))
```

```python
# Visualize top features
plt.figure(figsize=(10, 6))
sns.barplot(
    data=selected_features.head(15),
    x="coefficient",
    y="feature",
    palette="viridis"
)
plt.title(f"Top 15 Features from {best_model_name}")
plt.xlabel("Coefficient Value")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

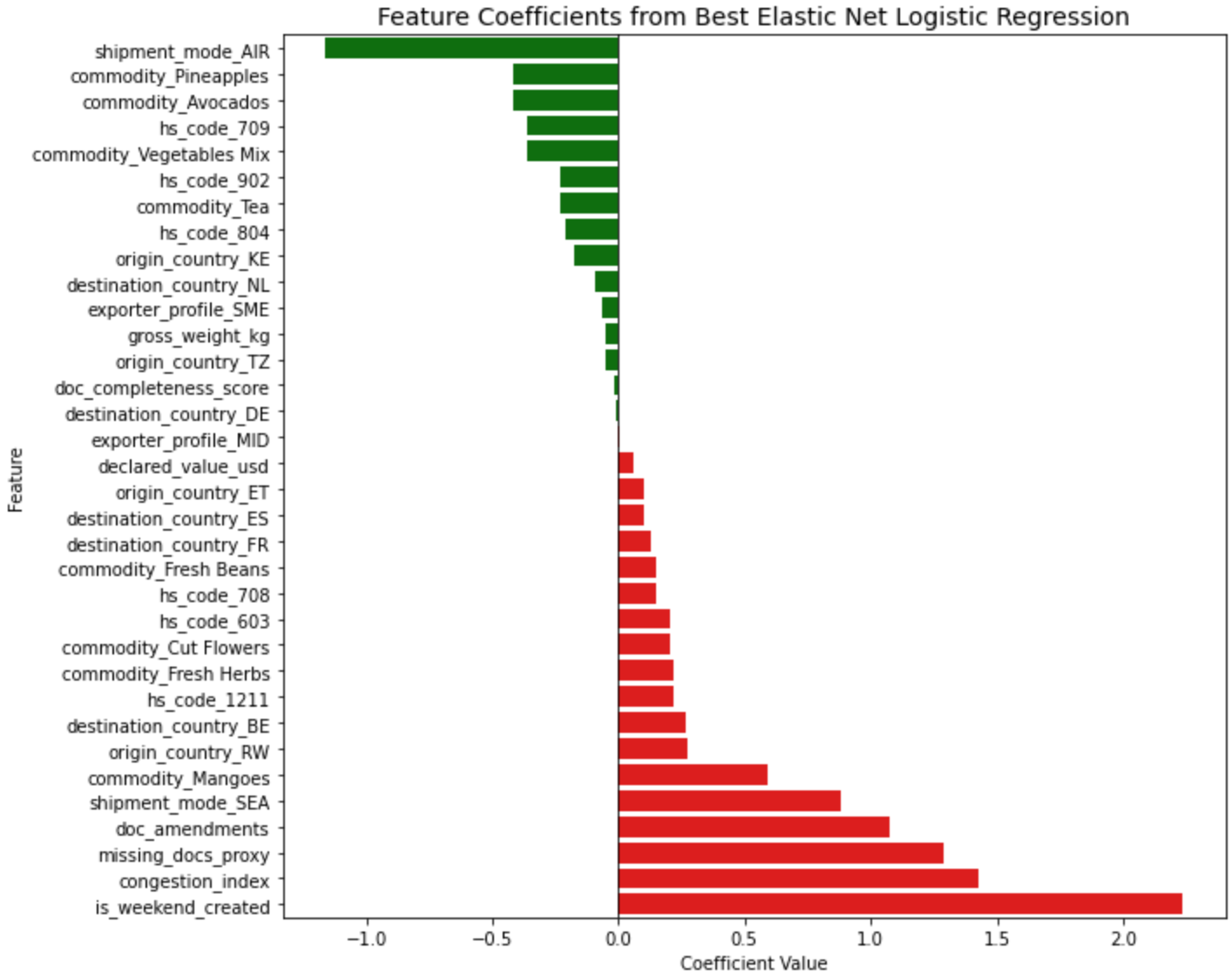
```
Best model based on ROC_AUC score: Logistic ElasticNet
                     feature  coefficient
6         is_weekend_created     2.235068
5           congestion_index     1.424564
3         missing_docs_proxy     1.288822
7          shipment_mode_AIR    -1.160538
4             doc_amendments     1.073068
8          shipment_mode_SEA     0.883667
13         commodity_Mangoes     0.590305
14      commodity_Pineapples    -0.416709
9         commodity_Avocados    -0.416584
19                hs_code_709    -0.361940
16  commodity_Vegetables Mix    -0.361940
25         origin_country_RW     0.270739
28    destination_country_BE     0.265965
21                hs_code_902    -0.226992
15             commodity_Tea    -0.226992

<ipython-input-20-360018a99ccc>:49: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` vari
able to `hue` and set `legend=False` for the same effect.

  sns.barplot(
```


Top 15 Features from Logistic ElasticNet
is_weekend_created
congestion_index
missing_docs_proxy
shipment_mode_AIR
doc_amendments
shipment_mode_SEA
commodity_Mangoes
commodity_Pineapples
commodity_Avocados
hs_code_709
commodity_Vegetables Mix
origin_country_RW
destination_country_BE
hs_code_902
commodity_Tea
Feature
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Coefficient Value

In [21]:

```
# FEATURE IMPORTANCE VISUALIZATION

# Sort coefficients for visualization
selected_features_sorted = selected_features.sort_values("coefficient", ascending=True).copy()

# Create a color column: red for causes delay, green for does not cause delay
selected_features_sorted["color"] = selected_features_sorted["coefficient"].apply(lambda x: "red" if x > 0 el

plt.figure(figsize=(10, 8))

# Horizontal bar plot
sns.barplot(
    x="coefficient",
    y="feature",
    data=selected_features_sorted,
    palette=selected_features_sorted["color"].to_list()  # Pass a list of colors
)

plt.title("Feature Coefficients from Best Elastic Net Logistic Regression", fontsize=14)
plt.xlabel("Coefficient Value")
plt.ylabel("Feature")
plt.axvline(0, color="black", linewidth=0.8)  # Line at zero for reference
plt.tight_layout()
plt.show()
```

```
<ipython-input-21-319f06145326>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` vari
able to `hue` and set `legend=False` for the same effect.

  sns.barplot(
```



MODEL RE-EVALUATION BASED ON SELECTED FEATURES

In [22]:

```
# Use only selected features from best logistic model
selected_feature_names = selected_features["feature"].tolist()

# Transform X_train and X_test to keep only selected features
# If using ColumnTransformer, get transformed arrays
X_train_selected = best_model.named_steps["preprocessor"].transform(X_train)
X_test_selected = best_model.named_steps["preprocessor"].transform(X_test)

# Only keep the columns corresponding to selected features
# Assuming order matches encoded_feature_names
selected_indices = [encoded_feature_names.index(f) for f in selected_feature_names]

X_train_selected = X_train_selected[:, selected_indices]
X_test_selected = X_test_selected[:, selected_indices]


tree_models = {
    "Decision Tree": DecisionTreeClassifier(class_weight="balanced", random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=200, class_weight="balanced", random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric="logloss", random_state=42),
    "LightGBM": LGBMClassifier(random_state=42)
}

tree_results = {}

for name, model in tree_models.items():
    model.fit(X_train_selected, y_train)
    y_pred = model.predict(X_test_selected)
    if hasattr(model, "predict_proba"):
        y_prob = model.predict_proba(X_test_selected)[:, 1]
    else:
        # For models that do not have predict_proba
        y_prob = y_pred

    tree_results[name] = {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
        "ROC_AUC": roc_auc_score(y_test, y_prob)
    }

# Convert results to DataFrame
tree_results_df = pd.DataFrame(tree_results).T
tree_results_df
```

```
[21:01:23] WARNING: /home/conda/feedstock_root/build_artifacts/xgboost_1598185621802/work/src/learner.cc:
516:
Parameters: { use_label_encoder } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[LightGBM] [Info] Number of positive: 1935, number of negative: 2065
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001673 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1031
[LightGBM] [Info] Number of data points in the train set: 4000, number of used features: 34
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.483750 -> initscore=-0.065023
[LightGBM] [Info] Start training from score -0.065023
```

Out[22]:

	Accuracv	Precision	Recall	F1	ROC AUC

Observations from Feature Coefficients

The table shows the top features selected by the Elastic Net logistic regression model along with their coefficients:

Key patterns:

1. Strongest positive predictors:

- is_weekend_created and congestion_index are the most influential factors increasing risk.
- Operational timing and congestion are critical factors.

2. Shipment mode effects:

- AIR reduces risk, while SEA increases it — indicating mode choice is important.

3. Documentation effects:

- Missing or amended documents (missing_docs_proxy, doc_amendments) increase likelihood of the target event.

4. Commodity and HS code effects:

- Some commodities (Mangoes, Fresh Herbs) slightly increase risk, while others (Pineapples, Avocados, Tea) slightly reduce it.
- Certain HS codes also contribute positively or negatively, reflecting underlying regulatory or handling patterns.

Discussion on Model Performance

- Logistic regression with Elastic Net regularization outperformed all other models, including tree-based models like Decision Tree, Random Forest, XGBoost, and LightGBM.
- Elastic Net combines L1 and L2 penalties, which: Selects important features (like Lasso) and Stabilizes coefficients (like Ridge) for correlated features.

- This explains why logistic regression with Elastic Net could achieve better generalization than the more flexible tree-based models on this dataset.
- Using features selected by Elastic Net in tree-based models may improve their performance, but they did not surpass the logistic model in our evaluation — suggesting that the linear relationships captured by logistic regression are dominant, and unnecessary complexity from tree ensembles does not add predictive power here.

Conclusions

1. Best Model:

- Elastic Net-enhanced logistic regression.
- Achieves strong F1 score, ROC-AUC, and interpretable feature selection.

2. Feature Insights:

- Weekend shipments, congestion, shipment mode, and documentation issues are the most critical predictors.
- Commodity type and HS code have moderate effects.

3. Operational Implications:

- Interventions could focus on high-risk shipments flagged by the model.
- Monitoring weekend operations, congestion points, and document completeness could reduce risk.

4. Interpretability Advantage:

- Logistic regression coefficients allow direct interpretation of feature effects, unlike black-box models (XGBoost/LightGBM), which is valuable for business decisions.