# Encoding Categorical Variables

#Since machine learning models only work with numbers, we must convert categorical features into numerical values.

from sklearn.preprocessing import LabelEncoder

# Initialize label encoder

encoder = LabelEncoder()

# List of categorical columns to encode

categorical\_cols = ['Mode\_of\_Shipment', 'Warehouse\_block', 'Product\_importance', 'Gender']

# Apply encoding

for col in categorical\_cols:

    df[col] = encoder.fit\_transform(df[col])

# Display the first few rows to confirm changes

print(df.head())

ID Warehouse\_block Mode\_of\_Shipment Customer\_care\_calls \

0 1 3 0 4

1 2 4 0 4

2 3 0 0 2

3 4 1 0 3

4 5 2 0 2

Customer\_rating Cost\_of\_the\_Product Prior\_purchases Product\_importance \

0 2 177 3 1

1 5 216 2 1

2 2 183 4 1

3 3 176 4 2

4 2 184 3 2

Gender Discount\_offered Weight\_in\_gms Reached.on.Time\_Y.N

0 0 44 1233 1

1 1 59 3088 1

2 1 48 3374 1

3 1 10 1177 1

4 0 46 2484 1

from sklearn.preprocessing import MinMaxScaler

# Initialize MinMaxScaler

scaler = MinMaxScaler()

# Scale the numerical columns

num\_cols = ['Cost\_of\_the\_Product', 'Weight\_in\_gms', 'Discount\_offered']

df[num\_cols] = scaler.fit\_transform(df[num\_cols])

# Confirm scaling

print(df.head())

ID Warehouse\_block Mode\_of\_Shipment Customer\_care\_calls \

0 1 3 0 4

1 2 4 0 4

2 3 0 0 2

3 4 1 0 3

4 5 2 0 2

Customer\_rating Cost\_of\_the\_Product Prior\_purchases Product\_importance \

0 2 0.378505 3 1

1 5 0.560748 2 1

2 2 0.406542 4 1

3 3 0.373832 4 2

4 2 0.411215 3 2

Gender Discount\_offered Weight\_in\_gms Reached.on.Time\_Y.N

0 0 0.671875 0.033893 1

1 1 0.906250 0.304894 1

2 1 0.734375 0.346676 1

3 1 0.140625 0.025712 1

4 0 0.703125 0.216654 1

from sklearn.model\_selection import train\_test\_split

# Define features (X) and target variable (y)

X = df.drop(columns=['Reached.on.Time\_Y.N'])  # Features

y = df['Reached.on.Time\_Y.N']  # Target variable

# Split the dataset into 80% training and 20% testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# Confirm split sizes

print("Training Set Size:", X\_train.shape)

print("Test Set Size:", X\_test.shape)

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

# Split the dataset again (if needed)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the Decision Tree Classifier

dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

# Evaluate the model

y\_pred = dt\_model.predict(X\_test)

from sklearn.metrics import accuracy\_score, classification\_report

print("Updated Decision Tree Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Updated Classification Report:\n", classification\_report(y\_test, y\_pred))

Updated Decision Tree Accuracy: 0.6518181818181819

Updated Classification Report:

precision recall f1-score support

0 0.58 0.54 0.56 895

1 0.70 0.73 0.71 1305

accuracy 0.65 2200

macro avg 0.64 0.63 0.64 2200

weighted avg 0.65 0.65 0.65 2200

from sklearn.model\_selection import GridSearchCV

# Define parameter grid

param\_grid = {

    'criterion': ['gini', 'entropy'],

    'max\_depth': [2, 4, 6, 8, 10],

    'min\_samples\_split': [2, 5, 10],

    'min\_samples\_leaf': [1, 2, 4]

}

# GridSearchCV

grid\_search = GridSearchCV(DecisionTreeClassifier(random\_state=42), param\_grid, cv=5, scoring='accuracy', n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

# Best parameters

print("Best Parameters:", grid\_search.best\_params\_)

# Train with best parameters

best\_dt = DecisionTreeClassifier(\*\*grid\_search.best\_params\_, random\_state=42)

best\_dt.fit(X\_train, y\_train)

# Evaluate again

y\_pred\_best = best\_dt.predict(X\_test)

print("Optimized Decision Tree Accuracy:", accuracy\_score(y\_test, y\_pred\_best))

print("Optimized Classification Report:\n", classification\_report(y\_test, y\_pred\_best))

Best Parameters: {'criterion': 'gini', 'max\_depth': 2, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2}

Optimized Decision Tree Accuracy: 0.6945454545454546

Optimized Classification Report:

precision recall f1-score support

0 0.57 1.00 0.73 895

1 1.00 0.49 0.65 1305

accuracy 0.69 2200

macro avg 0.79 0.74 0.69 2200

weighted avg 0.83 0.69 0.68 2200

import matplotlib.pyplot as plt

import numpy as np

# Feature importance

feature\_importances = best\_dt.feature\_importances\_

# Plot

plt.figure(figsize=(10, 5))

plt.bar(X.columns, feature\_importances, color='skyblue')

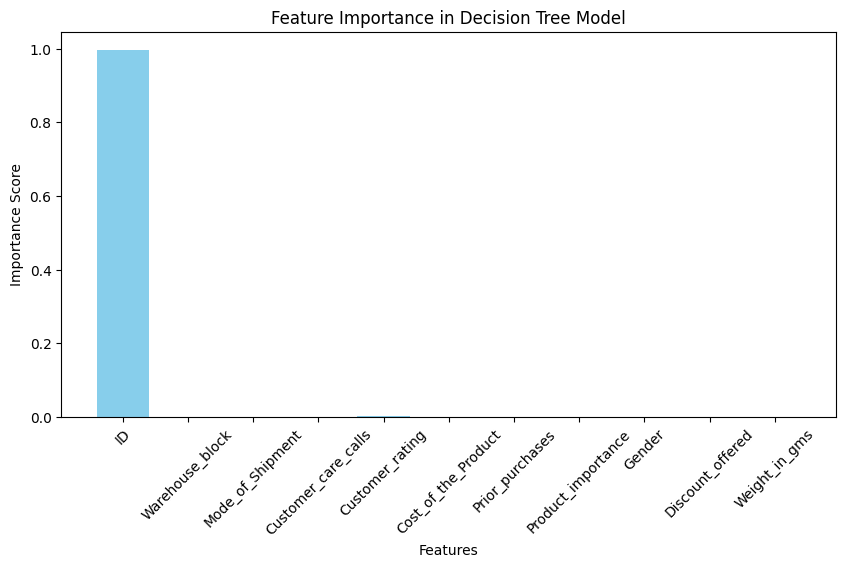
plt.xlabel('Features')

plt.ylabel('Importance Score')

plt.title('Feature Importance in Decision Tree Model')

plt.xticks(rotation=45)

plt.show()



from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Compute confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Plot confusion matrix

plt.figure(figsize=(5,4))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['On Time', 'Late'], yticklabels=['On Time', 'Late'])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix - Decision Tree")

plt.show()

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.model\_selection import train\_test\_split, GridSearchCV

import seaborn as sns

import matplotlib.pyplot as plt

# Initialize Random Forest Model

rfc = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the Model

rfc.fit(X\_train, y\_train)

# Predictions

y\_pred\_rfc = rfc.predict(X\_test)

# Split Data Again

X = df.drop(columns=['Reached.on.Time\_Y.N'])  # Features

y = df['Reached.on.Time\_Y.N']  # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Random Forest Again

rfc = RandomForestClassifier(n\_estimators=100, random\_state=42)

rfc.fit(X\_train, y\_train)

# Accuracy Score

accuracy\_rfc = accuracy\_score(y\_test, y\_pred\_rfc)

print(f'Random Forest Accuracy: {accuracy\_rfc:.2f}')

# Classification Report

print('Classification Report:\n', classification\_report(y\_test, y\_pred\_rfc))

# Confusion Matrix

conf\_matrix\_rfc = confusion\_matrix(y\_test, y\_pred\_rfc)

sns.heatmap(conf\_matrix\_rfc, annot=True, fmt="d", cmap="Blues", xticklabels=["On Time", "Late"], yticklabels=["On Time", "Late"])

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.title("Random Forest Confusion Matrix")

plt.show()

  RandomForestClassifier[?](https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html)i

RandomForestClassifier(random\_state=42)

Random Forest Accuracy: 0.68

Classification Report:

precision recall f1-score support

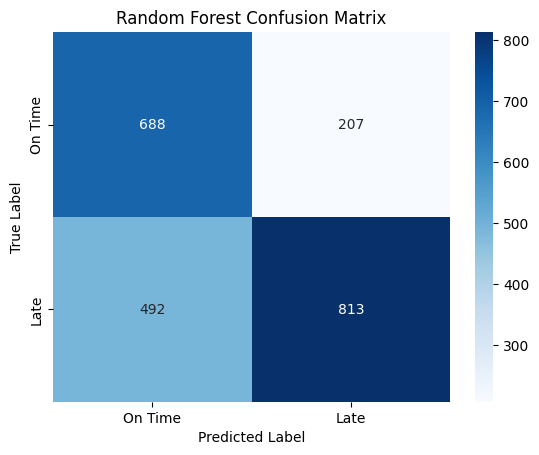
0 0.58 0.77 0.66 895

1 0.80 0.62 0.70 1305

accuracy 0.68 2200

macro avg 0.69 0.70 0.68 2200

weighted avg 0.71 0.68 0.68 2200



# Random Forest Feature Importance

feature\_importances\_rfc = rfc.feature\_importances\_

plt.figure(figsize=(10,5))

sns.barplot(x=feature\_importances\_rfc, y=features, palette="plasma")

plt.xlabel("Feature Importance Score")

plt.ylabel("Features")

plt.title("Feature Importance in Random Forest (After Removing ID)")

plt.show()

plt.figure(figsize=(8,5))

sns.boxplot(x=df['Reached.on.Time\_Y.N'], y=df['Weight\_in\_gms'], palette="coolwarm")

plt.xlabel("Delivery Status (0 = On Time, 1 = Late)")

plt.ylabel("Weight in Grams")

plt.title("Distribution of Package Weight by Delivery Status")

plt.show()

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Initialize the Logistic Regression model

log\_reg = LogisticRegression(max\_iter=1000, solver='lbfgs', random\_state=42)

# Train the model

log\_reg.fit(X\_train, y\_train)

# Predict on test data

y\_pred\_log = log\_reg.predict(X\_test)

# Evaluate the model

accuracy\_log = accuracy\_score(y\_test, y\_pred\_log)

print(f"Logistic Regression Accuracy: {accuracy\_log:.4f}")

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred\_log))

Logistic Regression Accuracy: 0.6545

Classification Report:

precision recall f1-score support

0 0.57 0.59 0.58 895

1 0.71 0.70 0.71 1305

accuracy 0.65 2200

macro avg 0.64 0.64 0.64 2200

weighted avg 0.66 0.65 0.66 2200

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Initialize the KNN model (default: k=5)

knn = KNeighborsClassifier(n\_neighbors=5)

# Train the model

knn.fit(X\_train, y\_train)

# Predict on test data

y\_pred\_knn = knn.predict(X\_test)

# Evaluate the model

accuracy\_knn = accuracy\_score(y\_test, y\_pred\_knn)

print(f"KNN Accuracy: {accuracy\_knn:.4f}")

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred\_knn))

KNN Accuracy: 0.6309

Classification Report:

precision recall f1-score support

0 0.54 0.58 0.56 895

1 0.70 0.67 0.68 1305

accuracy 0.63 2200

macro avg 0.62 0.62 0.62 2200

weighted avg 0.63 0.63 0.63 2200

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import RandomizedSearchCV

# Define the parameter grid

param\_grid = {

    'n\_estimators': [500, 1000, 1500],

    'max\_depth': [10, 20, 30, None],

    'min\_samples\_split': [2, 5, 10],

    'min\_samples\_leaf': [1, 2, 4],

    'bootstrap': [True, False]

}

# Initialize the model

rf = RandomForestClassifier(random\_state=42)

# Perform Randomized Search

random\_search = RandomizedSearchCV(

    estimator=rf,

    param\_distributions=param\_grid,

    n\_iter=50,

    cv=5,

    verbose=2,

    n\_jobs=-1,

    scoring='accuracy'

)

# Fit the model

random\_search.fit(X\_train, y\_train)

# Print the best parameters and accuracy

print("Best Parameters:", random\_search.best\_params\_)

print("Optimized Random Forest Accuracy:", random\_search.best\_score\_)

ssss

from sklearn.model\_selection import GridSearchCV

# Define parameter grid

param\_grid = {

    'n\_estimators': [1200, 1500, 1800],

    'max\_depth': [None, 20, 25],

    'min\_samples\_split': [2, 3, 4],

    'min\_samples\_leaf': [1, 2, 3],

}

# Initialize GridSearch

grid\_search = GridSearchCV(

    RandomForestClassifier(bootstrap=True, random\_state=42),

    param\_grid,

    cv=5,

    scoring='accuracy',

    n\_jobs=-1,

    verbose=2

)

# Fit GridSearch

grid\_search.fit(X\_train, y\_train)

# Get the best parameters

best\_params = grid\_search.best\_params\_

best\_model = grid\_search.best\_estimator\_

# Test final accuracy

final\_accuracy = best\_model.score(X\_test, y\_test)

print("Best Parameters:", best\_params)

print("Final Optimized Accuracy:", final\_accuracy)

# Feature Engineering - Creating new features

X\_train["Discount\_Per\_Cost"] = X\_train["Discount\_offered"] / X\_train["Cost\_of\_the\_Product"]

X\_test["Discount\_Per\_Cost"] = X\_test["Discount\_offered"] / X\_test["Cost\_of\_the\_Product"]

X\_train["Weight\_Cost\_Ratio"] = X\_train["Weight\_in\_gms"] / X\_train["Cost\_of\_the\_Product"]

X\_test["Weight\_Cost\_Ratio"] = X\_test["Weight\_in\_gms"] / X\_test["Cost\_of\_the\_Product"]

import numpy as np

# Replace infinite values with NaN

X\_train.replace([np.inf, -np.inf], np.nan, inplace=True)

X\_test.replace([np.inf, -np.inf], np.nan, inplace=True)

# Check if NaNs exist

print("Missing values in X\_train:\n", X\_train.isnull().sum())

print("Missing values in X\_test:\n", X\_test.isnull().sum())

# Fill NaNs with median

X\_train.fillna(X\_train.median(), inplace=True)

X\_test.fillna(X\_test.median(), inplace=True)

X\_train["Discount\_Per\_Cost"].fillna(X\_train["Discount\_Per\_Cost"].median(), inplace=True)

X\_test["Discount\_Per\_Cost"].fillna(X\_test["Discount\_Per\_Cost"].median(), inplace=True)

X\_train["Weight\_Cost\_Ratio"].fillna(X\_train["Weight\_Cost\_Ratio"].median(), inplace=True)

X\_test["Weight\_Cost\_Ratio"].fillna(X\_test["Weight\_Cost\_Ratio"].median(), inplace=True)

print("Missing values after fixing:\n", X\_train.isnull().sum())

print("Missing values after fixing:\n", X\_test.isnull().sum())

from sklearn.ensemble import RandomForestClassifier

# Train new model with added features

rf\_final = RandomForestClassifier(

    n\_estimators=1500,

    max\_depth=20,

    min\_samples\_split=2,

    min\_samples\_leaf=3,

    bootstrap=True,

    random\_state=42

)

rf\_final.fit(X\_train, y\_train)

accuracy = rf\_final.score(X\_test, y\_test)

print("Updated Random Forest Accuracy:", accuracy)

imators': [1500, 2000, 2500],

    'max\_depth': [20, 25, None],

    'min\_samples\_split': [2, 5, 10],

    'min\_samples\_leaf': [1, 2, 3]

}

grid\_search = GridSearchCV(

    RandomForestClassifier(random\_state=42),

    param\_grid,

    cv=5,

    n\_jobs=-1,

    verbose=2

)

grid\_search.fit(X\_train, y\_train)

print("Best Parameters:", grid\_search.best\_params\_)

rf\_best = grid\_search.best\_estimator\_

accuracy = rf\_best.score(X\_test, y\_test)

print("Optimized Random Forest Accuracy:", accuracy)

# Safe drop: only drop features that exist in the DataFrame

features\_to\_drop = ['Product\_importance', 'Prior\_purchases', 'Warehouse\_block', 'Mode\_of\_Shipment']

features\_to\_drop = [col for col in features\_to\_drop if col in X\_train.columns]

X\_train\_clean = X\_train.drop(columns=features\_to\_drop)

X\_test\_clean = X\_test.drop(columns=features\_to\_drop)

# Re-train Random Forest

from sklearn.ensemble import RandomForestClassifier

rf\_final = RandomForestClassifier(

    n\_estimators=2000,

    max\_depth=20,

    min\_samples\_split=2,

    min\_samples\_leaf=3,

    bootstrap=True,

    random\_state=42

)

rf\_final.fit(X\_train\_clean, y\_train)

accuracy = rf\_final.score(X\_test\_clean, y\_test)

print("Cleaned Features Accuracy:", accuracy)

Cleaned Features Accuracy: 0.6827272727272727

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score

xgb\_model = XGBClassifier(

    n\_estimators=1000,

    max\_depth=6,

    learning\_rate=0.1,

    subsample=0.8,

    colsample\_bytree=0.8,

    random\_state=42,

    use\_label\_encoder=False,

    eval\_metric='logloss'

)

xgb\_model.fit(X\_train\_clean, y\_train)

y\_pred = xgb\_model.predict(X\_test\_clean)

accuracy = accuracy\_score(y\_test, y\_pred)

print("XGBoost Accuracy:", accuracy)

[C:\Users\imogen\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12\_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\xgboost\training.py:183](file:///C:\Users\imogen\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\xgboost\training.py:183): UserWarning: [16:10:42] WARNING: [C:\actions-runner\\_work\xgboost\xgboost\src\learner.cc:738](file:///C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738):

Parameters: { "use\_label\_encoder" } are not used.

bst.update(dtrain, iteration=i, fobj=obj)

XGBoost Accuracy: 0.6631818181818182