Swire Coca-Cola USA (SCCU) - Group Modeling Notebook

- Swire Coca-Cola USA (SCCU) - Group Modeling Notebook
  - Introduction
  - Main questions
  - Requirement
  - Thing to consider in modeling
  - Business Problem
  - Analytical Approach
  - Import Packages
  - Load Dataset
  - Pipeline Class of Routing Optimizer
  - Sequences of Pipeline
  - Customer Segmentation based on the transition status
    - Create Transition Status Variable
    - Dataset Cleaning
    - Iteratively Building Decision Trees
      - SMOTE
      - Undersampling
    - XGBoost Model
      - Grid Search and Model Training
      - Generate Predictions on Train and Test Set
      - Feature Importance
  - Delivery Cost Optimization

- Logistic Regression Model Training
- Number of Customers who Improved from Below to Above
- Cluster Improved Customers by ZIP Code and Volume
- Cluster Improved Customers data using Ordered Gallon
- Ordered cases and Gallons Comparison
- Multi-Dimensional Clustering
- Average Delivery Cost Per Case
- Before Vs After Clustering -Comparing Cost per Delivery
- Percentage Cost Savings
- Average Cost per Gallon Before Clustering
- Average cost per Gallon After Clustering
- Visuals for Average Cost per Case Before vs After Clustering
- Visuals for cost per Gallon Before vs After Clustering
  - Annual Delivery Cost Statistics
  - Modelling: Predicting potential growth-ready customers
    - Prepare the Data
  - Benchmark Model: Logistic Regression
  - Random Forest Model
  - GridSearchCV with Random Forest Classifier
  - GradientBoosting(Using XGBoost) with GridSearch
    - Characteristics of customer growth
  - Group1: Sensitivity Analysis
  - Group2: Analysis on Local Market Partners who only buy Fountain
- Result

Group Member Contribution

Introduction

This notebook includes the pipeline to predict the growth potential customer & the decision modeling to find the optimized threshold with Machine Learning framework.

Main questions

- (Richard Lim) What factors or characteristics distinguish customers with annual sales exceeding the determined volume threshold from those below this threshold?

- 2023 (Below vs Above): figure out factors differing between above vs below group
- 2024 (Below vs Above): figure out factors differing between above vs below group

- (Varun Selvam) How can SCCU uses historical sales data, or other Customer Characteristics to predict which ARTM customers have the potential to grow beyond the volume threshold annually?

- Which customer characteristics contribute to move customers from below threshold (2023) to above threshold (2024)?

- (Meenakshi H) How can these insights be integrated into the routing strategy to support long-term growth while maintaining logistical efficiency?

- logistical efficiency represents how much we can save delivery cost at maximum while increasing the number of growth customer (Below to Above Group)

- (Nikita Muddapati) What levers can be employed to accelerate volume and share growth at growth-ready, high-potential customers?

- Business Recommendation: accurate number + storytelling to take action
- Question: which threshold should be used for new logistical efficiency?
- Reason: Reasoning why the threshold is better than original one (400 annual threshold)

Requirement

- A statement of the business and analytic problems for the project.
- Documentation of your modeling process (as well as any additional data cleaning and preparation you did) along with an interpretation of your results. This writing belongs in the main sections of the notebook dedicated to the modeling process, and should go between code chunks.
- Code annotation. This writing goes within code chunks, and should explain what the code is doing.
- All the performance metrics for the best model: in-sample and estimated out-of-sample performance.
- A results section at the end where you summarize the key points of what you have learned through the group modeling process. Can modeling results be used to solve the business problem?

Thing to consider in modeling

- Identify a performance benchmark. What is a minimum threshold for model performance?
- Identify appropriate models to explore (given the business problem and the project objective).
- Do additional data preparation and feature engineering as necessary. Perform cross-validation to develop performance metrics for each model, appropriate for the context.
- Optimize model performance with hyperparameter tuning, if appropriate.
- Evaluate the strengths and weaknesses of each model and select the best one.
- Perform business validation of the model. Are your results sufficient to solve the business problem?

Business Problem

SCCU(Swire Coca-Cola United States) tries to optimize logistics by transitioning customers selling below a specific annual volume to an Alternate Route to Market (ARTM). There is an annual 400 gallons volume threshold used to distinguish the customers between the direct delivery route and ARTM.

However, SCCU is looking for a more cost-efficient strategy to decide new threshold for optimizing logistics which is driving better operational efficiency and more revenues.

Analytical Approach

This analysis will focus on building the classification model to predict who is going to be a growth customer segment (those who were below the threshold from 2023 but becoming above the threshold on following year (2024)) based on the 2023 historical data only, and which volume threshold would be more optimal compared to original threshold.

Import Packages

```
In [ ]: # import libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import FunctionTransformer
from sklearn.tree import DecisionTreeClassifier
from imblearn.over_sampling import SMOTE

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier

from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score,
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from collections import Counter
from sklearn.utils import resample
from sklearn.linear_model import LogisticRegression


# Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
#from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import xgboost as xgb

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split


# to impute NA's
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

from sklearn.metrics import classification_report, confusion_matrix,accuracy_score,


# DV
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Statistical Analysis
#!pip install statsmodels
import scipy.stats as stats

# Warnings
import warnings
warnings.filterwarnings("ignore")

import pickle
import joblib
import warnings

warnings.filterwarnings('ignore')

# Set plotting style
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette('viridis')
```

Load Dataset

In [ ]:
```python
# Load data
file_path = pd.read_csv('sccu_data.csv')
```

Pipeline Class of Routing Optimizer

This code includes all of the pipeline that implement the prediction modeling of growth potential customer (Below in 2023, Above in 2024) based on the only 2023 historical data, and redefining the threshold for optimizing the SCCU logistical routes.

In [ ]:
```python
class SCCURoutingOptimizer:
    """
    ML pipeline for SCCU route optimization:
    1. Predicts growth potential for ARTM customers
    2. Identifies optimal thresholds for routing decisions
    3. Provides actionable business insights
    """

    def __init__(self, data_path='sccu_data.csv'):
        """Initialize with path to data file"""
        self.data_path = data_path
        self.data = None
        self.X_train = None
        self.X_test = None
        self.y_train = None
        self.y_test = None
        self.models = {}
        self.best_model = None

    # Define the lambda function inside the method
    @staticmethod
    def bool_to_int_func(x):
        return x.astype(int)

    def load_data(self):
        """Load and perform initial data processing"""
        print("1. Loading and preprocessing data")

        # Load data
        self.data = pd.read_csv(self.data_path)

        # Basic info
        print(f"Dataset shape: {self.data.shape}")
        print(f"Missing values:\n{self.data[self.data.columns[self.data.isnull().su

        # Analyze current thresholds
        print("\n3.Analyzing current thresholds...")
        self.analyze_current_thresholds()

        return self.data

    def create_derived_features(self):
        """Create additional features for analysis and prediction"""

        # Target variable for growth prediction: Did customer move from below to ab
        self.data['GROWTH_TARGET'] = ((self.data['THRESHOLD_2023'] == 'below') &
                                      (self.data['THRESHOLD_2024'] == 'above')).astyp

        # Print derived features
        print(f"Added derived 1 variables. New shapes are: {self.data.shape}")

    def analyze_current_thresholds(self, number = 400):
        """Analyze the current threshold strategy effectiveness"""

        # Calculate additional derived features
        print("\n2.Creating derived features...")
        self.create_derived_features()

        # Current distribution
        print(f"\n{number} Annual Threshold distribution:")
        below_threshold_2023 = (self.data['THRESHOLD_2023'] == 'below').sum()
        above_threshold_2023 = (self.data['THRESHOLD_2023'] == 'above').sum()
        below_threshold_2024 = (self.data['THRESHOLD_2024'] == 'below').sum()
        above_threshold_2024 = (self.data['THRESHOLD_2024'] == 'above').sum()

        # The number of Customer Movement between thresholds
        below_to_above = ((self.data['THRESHOLD_2023'] == 'below') &
                          (self.data['THRESHOLD_2024'] == 'above')).sum()
        above_to_below = ((self.data['THRESHOLD_2023'] == 'above') &
                          (self.data['THRESHOLD_2024'] == 'below')).sum()
        below_to_below = ((self.data['THRESHOLD_2023'] == 'below') &
                          (self.data['THRESHOLD_2024'] == 'below')).sum()
        above_to_above = ((self.data['THRESHOLD_2023'] == 'above') &
                          (self.data['THRESHOLD_2024'] == 'above')).sum()
        entire = below_to_above + above_to_below + below_to_below + above_to_above
```

```python
        # Comparison for customers who changed categories
        growth_customers = self.data[self.data['GROWTH_TARGET'] == 1]
        decline_customers = self.data[((self.data['THRESHOLD_2023'] == 'above') &
                                      (self.data['THRESHOLD_2024'] == 'below'))]
        same_above_customer = self.data[(self.data['THRESHOLD_2023'] == 'above') &
                                              (self.data['THRESHOLD_2024'] ==
        same_below_customer= self.data[(self.data['THRESHOLD_2023'] == 'below') &
                                              (self.data['THRESHOLD_2024'] ==

        print("-----------------------------------------------------------")
        print(f"2023: Below threshold: {below_threshold_2023} ({below_threshold_202
              f"Above threshold: {above_threshold_2023} ({above_threshold_2023/len(
        print(f"2024: Below threshold: {below_threshold_2024} ({below_threshold_202
              f"Above threshold: {above_threshold_2024} ({above_threshold_2024/len(
        print("-----------------------------------------------------------")


        print("\nThe number of customer proportion between thresholds:")

        print("-----------------------------------------------------------")
        print(f"Below to Above (Growth): {below_to_above} "
              f"({below_to_above/entire * 100:.2f}% of total)")
        print(f"Above to Below (Decline): {above_to_below} "
              f"({above_to_below/entire * 100:.2f}% of total)")
        print(f"Below to Below (No Change): {below_to_below} "
              f"({below_to_below/entire * 100:.2f}% of total)")
        print(f"Above to Above (No Change): {above_to_above} "
              f"({above_to_above/entire * 100:.2f}% of total)")
        print("-----------------------------------------------------------")

        # Current threshold analysis
        print("\n3.Current threshold analysis:")

        print(f"\nIf {number} Annual Threshold were applied...")



        print("\n\nVolume Change")

        print("-----------------------------------------------------------")

        print("\nTotal annumal volume change from 2023 to 2024")
        print(f"2023: {self.data['ANNUAL_VOLUME_2023'].sum():.2f} units")
        print(f"2024: {self.data['ANNUAL_VOLUME_2024'].sum():.2f} units")
        print(f"Change: {self.data['ANNUAL_VOLUME_2024'].sum() - self.data['ANNUAL_

        print("-----------------------------------------------------------")

        print("\nTotal annual volume change for no change (below to below):")
        print(f"2023: {same_below_customer['ANNUAL_VOLUME_2023'].sum():.2f} units")
        print(f"2024: {same_below_customer['ANNUAL_VOLUME_2024'].sum():.2f} units")
        print(f"Change: {same_below_customer['ANNUAL_VOLUME_2024'].sum() - same_bel

        print("\nTotal annual volume change for growth customers (below to above):"
        print(f"2023: {growth_customers['ANNUAL_VOLUME_2023'].sum():.2f} units")
        print(f"2024: {growth_customers['ANNUAL_VOLUME_2024'].sum():.2f} units")
        print(f"Change: {growth_customers['ANNUAL_VOLUME_2024'].sum() - growth_cust

        print("\nTotal annual volume change for no change (above to above):")
        print(f"2023: {same_above_customer['ANNUAL_VOLUME_2023'].sum():.2f} units")
        print(f"2024: {same_above_customer['ANNUAL_VOLUME_2024'].sum():.2f} units")
        print(f"Change: {same_above_customer['ANNUAL_VOLUME_2024'].sum()- same_abov

        print("\nTotal annual volume change for decline customers (above to below):
        print(f"2023: {decline_customers['ANNUAL_VOLUME_2023'].sum():.2f} units")
        print(f"2024: {decline_customers['ANNUAL_VOLUME_2024'].sum():.2f} units")
        print(f"Change: {decline_customers['ANNUAL_VOLUME_2024'].sum() - decline_cu

        print("-----------------------------------------------------------")

        print("\n\nDelivery cost change")

        print("-----------------------------------------------------------")

        print("\nTotal annumal delivery cost change from 2023 to 2024")
        print(f"2023: ${self.data['DELIVERY_COST_2023'].sum():.2f}")
        print(f"2024: ${self.data['DELIVERY_COST_2024'].sum():.2f}")
        print(f"Change: ${self.data['DELIVERY_COST_2024'].sum() - self.data['DELIVE

        print("-----------------------------------------------------------")

        print("\nTotal delivery cost change for no change (below to below):")
        print(f"2023: ${same_below_customer['DELIVERY_COST_2023'].sum():.2f}")
        print(f"2024: ${same_below_customer['DELIVERY_COST_2024'].sum():.2f}")
        print(f"Change: ${same_below_customer['DELIVERY_COST_2024'].sum() - same_be

        print("\nTotal delivery cost change for growth customers (below to above):"
        print(f"2023: ${growth_customers['DELIVERY_COST_2023'].sum():.2f}")
        print(f"2024: ${growth_customers['DELIVERY_COST_2024'].sum():.2f}")
        print(f"Change: ${growth_customers['DELIVERY_COST_2024'].sum() - growth_cus

        print("\nTotal delivery cost change for no change (above to above):")
        print(f"2023: ${same_above_customer['DELIVERY_COST_2023'].sum():.2f}")
        print(f"2024: ${same_above_customer['DELIVERY_COST_2024'].sum():.2f}")
        print(f"Change: ${same_above_customer['DELIVERY_COST_2024'].sum() - same_ab

        print("\nTotal delivery cost change for decline customers (above to below):
        print(f"2023: ${decline_customers['DELIVERY_COST_2023'].sum():.2f}")
        print(f"2024: ${decline_customers['DELIVERY_COST_2024'].sum():.2f}")
        print(f"Change: ${decline_customers['DELIVERY_COST_2024'].sum() - decline_c

        print("-----------------------------------------------------------")

    def explore_data(self):
        """Perform exploratory data analysis for insights"""

        print("\n4.Exploring data for insights...")

        # Growth customer analysis
        growth_df = self.data[self.data['GROWTH_TARGET'] == 1]
        print(f"\nAnalyzing {len(growth_df)} growth customers (moved from below to
        # Trade channel distribution for growth customers
        trade_channel_growth = growth_df['TRADE_CHANNEL'].value_counts().head(10)
        print("\nTop 10 trade channels for growth customers:")
        print(trade_channel_growth)

        # Calculate growth rate by trade channel
        channel_growth_rates = []
        for channel in self.data['TRADE_CHANNEL'].unique():
            channel_customers = self.data[self.data['TRADE_CHANNEL'] == channel]
            below_customers = channel_customers[channel_customers['THRESHOLD_2023']
            growth_customers = below_customers[below_customers['THRESHOLD_2024'] ==

            if len(below_customers) > 0:
                growth_rate = len(growth_customers) / len(below_customers) * 100
                channel_growth_rates.append((channel, len(below_customers), len(gro

        # Sort by growth rate and show top channels
        channel_growth_df = pd.DataFrame(channel_growth_rates,
                                       columns=['Channel', 'Below Customers', 'Grow
        channel_growth_df = channel_growth_df.sort_values('Growth Rate', ascending=
        print("\nTop trade channels by growth rate (%):")
        print(channel_growth_df.head(10))

        # Cold drink channel analysis
        cold_drink_growth_rates = []
        for channel in self.data['COLD_DRINK_CHANNEL'].unique():
            channel_customers = self.data[self.data['COLD_DRINK_CHANNEL'] == channe
            below_customers = channel_customers[channel_customers['THRESHOLD_2023']
            growth_customers = below_customers[below_customers['THRESHOLD_2024'] ==

            if len(below_customers) > 0:
                growth_rate = len(growth_customers) / len(below_customers) * 100
                cold_drink_growth_rates.append((channel, len(below_customers), len(

        cold_drink_growth_df = pd.DataFrame(cold_drink_growth_rates,
                                          columns=['Channel', 'Below Customers', 'G
        cold_drink_growth_df = cold_drink_growth_df.sort_values('Growth Rate', asce
        print("\nCold drink channels by growth rate (%):")
        print(cold_drink_growth_df)

        # Partner status analysis
        partner_stats = self.data.groupby(['LOCAL_MARKET_PARTNER', 'THRESHOLD_2023'
        partner_growth = self.data[self.data['THRESHOLD_2023'] == 'below'].groupby(
        print("\nGrowth rate by local market partner status (%):")
        print(partner_growth)

        # CO2 customer analysis
        co2_stats = self.data.groupby(['CO2_CUSTOMER', 'THRESHOLD_2023']).size().un
        co2_growth = self.data[self.data['THRESHOLD_2023'] == 'below'].groupby('CO2
        print("\nGrowth rate by CO2 customer status (%):")
        print(co2_growth)

        # Transaction count analysis
        self.data['TRANS_COUNT_BUCKET'] = pd.cut(
            self.data['TRANS_COUNT_2023'],
            bins=[0, 5, 10, 20, 30, np.inf],
            labels=['0-5', '5-10', '10-20', '20-30', '30+']
        )

        trans_growth = self.data[self.data['THRESHOLD_2023'] == 'below'].groupby('T
        trans_growth['growth_rate'] = trans_growth['mean'] * 100
        print("\nGrowth rate by transaction count bucket (%):")
        print(trans_growth)

        # Average order volume analysis
        self.data['AVG_ORDER_BUCKET'] = pd.cut(
            self.data['AVG_ORDER_VOLUME_2023'],
            bins=[0, 10, 20, 30, 40, np.inf],
            labels=['0-10', '10-20', '20-30', '30-40', '40+']
        )

        order_growth = self.data[self.data['THRESHOLD_2023'] == 'below'].groupby('A
        order_growth['growth_rate'] = order_growth['mean'] * 100
        print("\nGrowth rate by average order volume bucket (%):")
        print(order_growth)

        return growth_df

    def _prepare_features_and_target(self):
        """Prepare features and target variable for modeling"""
        # Focus on customers below threshold in 2023
        below_threshold = self.data[self.data['THRESHOLD_2023'] == 'below'].copy()

        # Define features and target
        target = 'GROWTH_TARGET'

        # Numerical features
        numerical_features = [
            'TRANS_COUNT_2023',
            'ANNUAL_VOLUME_CASES_2023',
            'ANNUAL_VOLUME_GALLON_2023',
            'AVG_ORDER_VOLUME_2023',
            'DELIVERY_COST_2023_CASES',
            'DELIVERY_COST_2023_GALLON'
        ]

        # Categorical features
        categorical_features = [
            'STATE_SHORT',
            'TRADE_CHANNEL',
            'SUB_TRADE_CHANNEL',
            'COLD_DRINK_CHANNEL',
            'FREQUENT_ORDER_TYPE'
        ]

        # Boolean features
        boolean_features = [
            'LOCAL_MARKET_PARTNER',
            'CO2_CUSTOMER'
        ]

        # Feature lists
        feature_columns = numerical_features + categorical_features + boolean_featu

        # Print feature information
        print(f"\nUsing {len(numerical_features)} numerical features, "
              f"{len(categorical_features)} categorical features, and "
              f"{len(boolean_features)} boolean features")

        # Create feature matrix and target vector
        X = below_threshold[feature_columns].copy()
        y = below_threshold[target].copy()

        # Check for missing values
        missing_values = X.isnull().sum()
        print("\nFeatures with missing values:")
        print(missing_values[missing_values > 0])

        return X, y

    def prepare_data_for_modeling(self, test_size=0.2, random_state=42):
        """Prepare training and test datasets for modeling"""

        print("\nPreparing data for modeling...")

        # Get features and target
        X, y = self._prepare_features_and_target()

        # Split data
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            X, y, test_size=test_size, random_state=random_state, stratify=y
        )

        print(f"Training set: {len(self.X_train)} samples")
        print(f"Test set: {len(self.X_test)} samples")
        print(f"Positive class (growth) in train: {sum(self.y_train)} ({sum(self.y_
        print(f"Positive class (growth) in test: {sum(self.y_test)} ({sum(self.y_te

        return self.X_train, self.X_test, self.y_train, self.y_test

    def create_preprocessing_pipeline(self):
        """Create a preprocessing pipeline for numerical and categorical features""
        # Identify numerical and categorical columns
        numerical_columns = self.X_train.select_dtypes(include=['int64', 'float64']
        categorical_columns = self.X_train.select_dtypes(include=['object']).column
        boolean_columns = self.X_train.select_dtypes(include=['bool']).columns.toli

        # Create preprocessing steps
        numerical_transformer = Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', StandardScaler())
        ])

        categorical_transformer = Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('onehot', OneHotEncoder(handle_unknown='ignore'))
        ])
        boolean_transformer = Pipeline(steps=[
            ('bool_to_int', FunctionTransformer(self.bool_to_int_func)),  # Convert
            ('imputer', SimpleImputer(strategy='most_frequent'))
        ])

        # Create the column transformer
        preprocessor = ColumnTransformer(
            transformers=[
                ('num', numerical_transformer, numerical_columns),
                ('cat', categorical_transformer, categorical_columns),
                ('bool', boolean_transformer, boolean_columns)
            ],
            remainder='drop'
        )

        return preprocessor

    def train_models(self, use_smote=True):
        """Train multiple ML models to predict growth potential"""

        print("\nTraining models to predict growth potential...")

        # Create preprocessing pipeline
        preprocessor = self.create_preprocessing_pipeline()

        # Apply SMOTE to handle class imbalance if needed
        if use_smote:
            print("Applying SMOTE to handle class imbalance...")
            smote = SMOTE(random_state=42)
            X_train_processed = preprocessor.fit_transform(self.X_train)
            X_train_resampled, y_train_resampled = smote.fit_resample(X_train_proce
            # print(f"After SMOTE - Training samples: {len(X_train_resampled)}, "
            #       f"Positive class: {sum(y_train_resampled)} ({sum(y_train_resamp
            print(f"After SMOTE - Training samples: {X_train_resampled.shape[0]}, "
                  f"Positive class: {np.sum(y_train_resampled)}({np.sum(y_train_res
        else:
            X_train_resampled, y_train_resampled = self.X_train, self.y_train

        # Define models to train
        models = {
            'logistic_regression': Pipeline([
                ('preprocessor', preprocessor),
                ('classifier', LogisticRegression(class_weight='balanced', max_iter
            ]),
            'random_forest': Pipeline([
                ('preprocessor', preprocessor),
                ('classifier', RandomForestClassifier(n_estimators=100, class_weigh
            ]),
            'gradient_boosting': Pipeline([
                ('preprocessor', preprocessor),
                ('classifier', GradientBoostingClassifier(n_estimators=100, random_
            ]),
            'xgboost': Pipeline([
                ('preprocessor', preprocessor),
                ('classifier', XGBClassifier(n_estimators=100, use_label_encoder=Fa
            ])
        }

        # Train each model and store
        for name, model in models.items():
            print(f"\nTraining {name}...")
            if use_smote:
                # For SMOTE, fit preprocessor first, then classifier
                pipeline_model = model
                pipeline_model.fit(self.X_train, self.y_train)

                # Get the classifier from the pipeline
                classifier = pipeline_model.named_steps['classifier']

                # Replace the raw estimator with SMOTE-trained version
                if name == 'logistic_regression':
                    pipeline_model.named_steps['classifier'] = LogisticRegression(
                        class_weight='balanced', max_iter=1000, random_state=42
                    ).fit(X_train_resampled, y_train_resampled)
                elif name == 'random_forest':
                    pipeline_model.named_steps['classifier'] = RandomForestClassifi
                        n_estimators=100, class_weight='balanced', random_state=42
                    ).fit(X_train_resampled, y_train_resampled)
                elif name == 'gradient_boosting':
                    pipeline_model.named_steps['classifier'] = GradientBoostingClas
                        n_estimators=100, random_state=42
                    ).fit(X_train_resampled, y_train_resampled)
                elif name == 'xgboost':
                    pipeline_model.named_steps['classifier'] = XGBClassifier(
                        n_estimators=100, use_label_encoder=False, eval_metric='log
                    ).fit(X_train_resampled, y_train_resampled)

                self.models[name] = pipeline_model
            else:
                # Without SMOTE, fit the full pipeline
                model.fit(self.X_train, self.y_train)
                self.models[name] = model

        return self.models

    def evaluate_models(self):
        """Evaluate model performance on test data"""
        if not self.models:
            print("No trained models found. Call train_models() first.")
            return

        print("\nEvaluating model performance...")

        # Store evaluation metrics
        model_results = {}

        # Evaluate each model
        for name, model in self.models.items():
            print(f"\nEvaluating {name}...")

            # Make predictions
            y_pred = model.predict(self.X_test)
            y_pred_proba = model.predict_proba(self.X_test)[:, 1]

            # Calculate metrics
            classification_rep = classification_report(self.y_test, y_pred)
            conf_matrix = confusion_matrix(self.y_test, y_pred)
            roc_auc = roc_auc_score(self.y_test, y_pred_proba)

            # Print results
            print(f"ROC AUC: {roc_auc:.4f}")
            print("\nClassification Report:")
            print(classification_rep)
            print("\nConfusion Matrix:")
            print(conf_matrix)

            # Store results
            model_results[name] = {
                'roc_auc': roc_auc,
                'classification_report': classification_rep,
                'confusion_matrix': conf_matrix,
                'y_pred': y_pred,
                'y_pred_proba': y_pred_proba
            }

        # Identify best model based on ROC AUC
        best_model_name = max(model_results, key=lambda k: model_results[k]['roc_au
        self.best_model = self.models[best_model_name]

        print(f"\nBest model: {best_model_name} with ROC AUC: {model_results[best_m

        return model_results

    def analyze_feature_importance(self):
        """Analyze feature importance for the best model"""
        if not self.best_model:
            print("No best model identified. Call evaluate_models() first.")
            return

        print("\nAnalyzing feature importance...")

        # Get the classifier from the pipeline
        if hasattr(self.best_model, 'named_steps') and 'classifier' in self.best_mo
            classifier = self.best_model.named_steps['classifier']
            preprocessor = self.best_model.named_steps['preprocessor']

            # Get feature names after preprocessing
            numerical_columns = self.X_train.select_dtypes(include=['int64', 'float
            categorical_columns = self.X_train.select_dtypes(include=['object']).co
            boolean_columns = self.X_train.select_dtypes(include=['bool']).columns.

            # Get feature names after one-hot encoding for categorical features
            feature_names = []

            # Add numerical feature names
            feature_names.extend(numerical_columns)

            # Add categorical feature names with one-hot encoding
```

```
                for col in categorical_columns:
                    unique_values = self.X_train[col].dropna().unique()
                    for val in unique_values:
                        feature_names.append(f"{col}_{val}")

                # Add boolean feature names
                feature_names.extend(boolean_columns)

                # Get feature importance based on the model type
                if hasattr(classifier, 'feature_importances_'):  # For tree-based model
                    importance = classifier.feature_importances_
                elif hasattr(classifier, 'coef_'):  # For linear models
                    importance = np.abs(classifier.coef_[0])
                else:
                    print("Feature importance not available for this model type.")
                    return

                # Ensure lengths match (might need to be adjusted for actual encoded fe
                if len(importance) != len(feature_names):
                    print(f"Warning: Feature importance length ({len(importance)}) "
                          f"doesn't match feature names length ({len(feature_names)})")
                    # Use generic feature names
                    feature_names = [f"Feature_{i}" for i in range(len(importance))]

                # Create feature importance DataFrame
                importance_df = pd.DataFrame({
                    'Feature': feature_names[:len(importance)],
                    'Importance': importance
                })

                # Sort by importance
                importance_df = importance_df.sort_values('Importance', ascending=False

                # Print top features
                print("\nTop 20 important features:")
                print(importance_df.head(20))

                return importance_df
            else:
                print("Feature importance analysis not available for this model type.")
                return None

        def optimize_threshold(self, threshold_range=None):
            """
            Find the optimal volume threshold for routing decisions
            by balancing logistical costs and growth potential
            """

            print("\nOptimizing volume threshold for routing decisions...")

            if threshold_range is None:
                # Default range from 200 to 600 units
                threshold_range = range(100, 601, 50)

            threshold_metrics = []
            # Initialize variables outside the if block
            saving_cost = 0
            direct_cost = 0
            direct_volume = 0
            artm_volume = 0
            total_volume = 0

            for threshold in threshold_range:
                # Apply threshold
                self.data[f'THRESHOLD_{threshold}_2023'] = self.data[f'ANNUAL_VOLUME_20
                    lambda x: 'above' if x >= threshold else 'below')
                self.data[f'THRESHOLD_{threshold}_2024'] = self.data[f'ANNUAL_VOLUME_20
                    lambda x: 'above' if x >= threshold else 'below')

                # Count customers in each category
                below_to_below_count = ((self.data[f'THRESHOLD_{threshold}_2023'] == 'b
                above_to_above_count = ((self.data[f'THRESHOLD_{threshold}_2023'] == 'a
                below_to_above_count = ((self.data[f'THRESHOLD_{threshold}_2023'] == 'b
                above_to_below_count = ((self.data[f'THRESHOLD_{threshold}_2023'] == 'a


                if 'DELIVERY_COST_2023' in self.data.columns:
                    # Calculate total delivery cost by threshold
                    artm_cost_2023 = self.data[self.data[f'THRESHOLD_{threshold}_2023']
                    artm_cost_2024 = self.data[self.data[f'THRESHOLD_{threshold}_2024']
                    direct_cost_2023 = self.data[self.data[f'THRESHOLD_{threshold}_2023
                    direct_cost_2024 = self.data[self.data[f'THRESHOLD_{threshold}_2024


                    # Calculate annual volume by threshold
                    artm_volume_2023 = self.data[self.data[f'THRESHOLD_{threshold}_2023
                    artm_volume_2024 = self.data[self.data[f'THRESHOLD_{threshold}_2024
                    direct_volume_2023 = self.data[self.data[f'THRESHOLD_{threshold}_20
                    direct_volume_2024 = self.data[self.data[f'THRESHOLD_{threshold}_20


                    # Calculate total cost
                    artm_cost_change = artm_cost_2024 - artm_cost_2023
                    direct_cost_change = direct_cost_2024 - direct_cost_2023
                    direct_volume_change = direct_volume_2024 - direct_volume_2023
                    artm_volume_change = artm_volume_2024 - artm_volume_2023
                    total_volume_change = (direct_volume_2024 + artm_volume_2024) - (di
                    toal_cost_change = (direct_cost_2024 + artm_cost_2024) - (direct_co

                # Store metrics
                threshold_metrics.append({
                    'threshold': threshold,
                    'below_to_above_count': below_to_above_count,
                    'above_to_below_count': above_to_below_count,
                    'difference': below_to_above_count - above_to_below_count,
                    'below_to_above_percent': below_to_above_count / len(self.data) * 1
                    'above_to_below_percent': above_to_below_count / len(self.data) * 1
                    'direct_cost_2023': direct_cost_2023,
                    'direct_cost_2024': direct_cost_2024,
                    'direct_cost_change': direct_cost_change,
                    'artm_cost_change': artm_cost_change,
                    'direct_volume_change': direct_volume_change,
                    'artm_volume_change': artm_volume_change,
                    'total_volume_change': total_volume_change,
                    'toal_cost_change': toal_cost_change
                })

            # Convert to DataFrame
            threshold_df = pd.DataFrame(threshold_metrics)

            # Print results
            print("\nThreshold optimization results:")

            # visualize the comparison between original and optimized threshold
            self.compare_the_original_and_optimized_thresholds(df = threshold_df)

            return threshold_df

        def compare_the_original_and_optimized_thresholds(self, df = None):
            """Compare original and optimized thresholds"""
            if df is None:
              df = self.optimize_threshold()

            # visualize the comparison
            df.plot(x='threshold', y=['direct_cost_change', 'direct_volume_change'], ki

        def compare_the_effect_of_thresholds(self):
            """Compare the effect of thresholds"""

            self.analyze_current_thresholds()

            self.data['THRESHOLD_2023'] = self.data['ANNUAL_VOLUME_2023'].apply(lambda
            self.data['THRESHOLD_2024'] = self.data['ANNUAL_VOLUME_2024'].apply(lambda

            self.analyze_current_thresholds(number = 300)

        def analyze_the_metrics_and_business_impact(self, price = 5):
            """Analyze the metrics and business impact"""

            # Growth Customer
            print('when 400 threshold applied, the number of growth customer is 1268 (4
            print('when 300 threshold applied, the number of growth customer is 1620 (5

            # total saving delivery cost comparison
            # 400 threshold
            print('\n')
            print(f'when 400 threshold applied, direct delivery cost increases by: ${21
            # 300 threshold
            print(f'when 300 threshold applied, direct delivery cost increases by: ${24

            # total volume increase comparison
            # 400 threshold
            print('\n')
            print(f'when 400 threshold applied, growth customer volume increase by: {13
            # 300 threshold
            print(f'when 300 threshold applied, growth customer volume increase by: {13
            net_revenue = (1371834.82 - 1309052.46) * price - (2400822.11 - 2113387.14)
            print('\n')
            print(f'when price is ${price} and 300 threshold applied, net revenue per a



        def save_model(self, filename='sccu_growth_model.pkl'):
            """Save the best model to a file"""
            if not self.best_model:
                print("\nNo best model identified. Call evaluate_models() first.")
                return

            joblib.dump(self.best_model, filename)
            print(f"\nModel saved to {filename}")
```

Sequences of Pipeline

This codes includes the pipeline based on the below sequences.

1. Setting up the the pipeline classes
2. Loading the dataset and checkout data profile
3. Identifying the benchmark performance of original threshold (400 threshold)
4. Generating the classification model to predict the growth customer group
5. Selecting and engineering the variables used in the modeling
6. Training and evaluating the performance between the models and select the best model
7. Generating the comparison metrics between new threshold and old threshold

In [ ]:
```
# Initialize the pipeline
pipeline = SCCURoutingOptimizer()

# Step 1: Load and preprocess data
pipeline.load_data()

# Step 2: Explore data for insights
pipeline.explore_data()

# Step 3: Prepare data for modeling
pipeline.prepare_data_for_modeling()

# Step 4: Train predictive models
pipeline.train_models(use_smote=True)

# Step 5: Evaluate model performance
pipeline.evaluate_models()

# Step 8: Compare the new threshold with original threshold
pipeline.compare_the_effect_of_thresholds()

# Step 9: Analyze the metrics and business impact
pipeline.analyze_the_metrics_and_business_impact()
```

```
# Step 9: Save the best model
pipeline.save_model()
```

```
1. Loading and preprocessing data
Dataset shape: (30755, 37)
Missing values:
AVG_ORDER_VOLUME_2023         4291
AVG_ORDER_VOLUME_2024          724
PERCENT_CHANGE                 137
PRIMARY_GROUP_NUMBER         18341
DELIVERY_COST_2023_CASES        68
DELIVERY_COST_2024_CASES        68
DELIVERY_COST_2023_GALLON       57
DELIVERY_COST_2024_GALLON       57
DELIVERY_COST_2023              68
DELIVERY_COST_2024              68
dtype: int64

3.Analyzing current thresholds...

2.Creating derived features...
Added derived 1 variables. New shapes are: (30755, 38)

400 Annual Threshold distribution:
----------------------------------------------------------
2023: Below threshold: 22832 (74.24%), Above threshold: 7923 (25.76%)
2024: Below threshold: 22720 (73.87%), Above threshold: 8035 (26.13%)
----------------------------------------------------------

The number of customer proportion between thresholds:
----------------------------------------------------------
Below to Above (Growth): 1268 (4.12% of total)
Above to Below (Decline): 1156 (3.76% of total)
Below to Below (No Change): 21564 (70.12% of total)
Above to Above (No Change): 6767 (22.00% of total)
----------------------------------------------------------

3.Current threshold analysis:

If 400 Annual Threshold were applied...


Volume Change
----------------------------------------------------------

Total annumal volume change from 2023 to 2024
2023: 18937111.54 units
2024: 19928814.14 units
Change: 991702.59 units
----------------------------------------------------------

Total annual volume change for no change (below to below):
2023: 2373218.54 units
2024: 2518678.75 units
Change: 145460.21 units

Total annual volume change for growth customers (below to above):
2023: 219940.73 units
2024: 1528993.19 units
```

```
Change: 1309052.46 units

Total annual volume change for no change (above to above):
2023: 15560010.91 units
2024: 15599958.61 units
Change: 39947.69 units

Total annual volume change for decline customers (above to below):
2023: 783941.36 units
2024: 281183.60 units
Change: -502757.77 units
----------------------------------------------------------


Delivery cost change
----------------------------------------------------------

Total annumal delivery cost change from 2023 to 2024
2023: $34298477.62
2024: $36012190.37
Change: $1713712.75
----------------------------------------------------------

Total delivery cost change for no change (below to below):
2023: $11703364.74
2024: $12460361.75
Change: $756997.02

Total delivery cost change for growth customers (below to above):
2023: $960184.35
2024: $3073571.48
Change: $2113387.14

Total delivery cost change for no change (above to above):
2023: $19359555.34
2024: $19284711.41
Change: $-74843.93

Total delivery cost change for decline customers (above to below):
2023: $2275373.19
2024: $1193545.72
Change: $-1081827.47
----------------------------------------------------------

4.Exploring data for insights...

Analyzing 1268 growth customers (moved from below to above threshold)

Top 10 trade channels for growth customers:
TRADE_CHANNEL
FAST CASUAL DINING         282
OTHER DINING & BEVERAGE    160
GENERAL RETAILER           153
COMPREHENSIVE DINING       145
GENERAL                     90
OUTDOOR ACTIVITIES          77
```

```
ACCOMMODATION              58
RECREATION                 53
LICENSED HOSPITALITY       41
ACADEMIC INSTITUTION       36
Name: count, dtype: int64

Top trade channels by growth rate (%):
                 Channel  Below Customers  Growth Customers  Growth Rate
20            BULK TRADE               34                15    44.117647
24            SUPERSTORE                3                 1    33.333333
8                GENERAL              511                90    17.612524
22               DEFENSE               68                11    16.176471
16                TRAVEL               40                 6    15.000000
17  ACADEMIC INSTITUTION              251                36    14.342629
21            ACTIVITIES              170                17    10.000000
13            RECREATION              543                53     9.760589
4             HEALTHCARE              258                22     8.527132
1     FAST CASUAL DINING             4032               282     6.994048

Cold drink channels by growth rate (%):
         Channel  Below Customers  Growth Customers  Growth Rate
5     BULK TRADE              481                96    19.958420
3       WELLNESS              268                22     8.208955
7  PUBLIC SECTOR             1277                84     6.577917
0          EVENT             2127               138     6.488011
4  ACCOMMODATION              912                57     6.250000
1         DINING            11742               638     5.433487
6      WORKPLACE              989                44     4.448938
2          GOODS             4982               188     3.773585
8   CONVENTIONAL               54                 1     1.851852

Growth rate by local market partner status (%):
LOCAL_MARKET_PARTNER
False    11.725168
True      5.169125
Name: GROWTH_TARGET, dtype: float64

Growth rate by CO2 customer status (%):
CO2_CUSTOMER
False    5.387060
True     5.810329
Name: GROWTH_TARGET, dtype: float64

Growth rate by transaction count bucket (%):
                    size  sum      mean  growth_rate
TRANS_COUNT_BUCKET
0-5                 4971  211  0.042446     4.244619
5-10                4660  186  0.039914     3.991416
10-20               5469  236  0.043152     4.315231
20-30               2252  131  0.058171     5.817052
30+                 1189  102  0.085786     8.578638

Growth rate by average order volume bucket (%):
                  size  sum      mean  growth_rate
AVG_ORDER_BUCKET
0-10              8411  108  0.012840     1.284033
```

```
10-20             7049  323  0.045822     4.582210
20-30             1751  195  0.111365    11.136493
30-40              596   96  0.161074    16.107383
40+                591  144  0.243655    24.365482

Preparing data for modeling...

Using 6 numerical features, 5 categorical features, and 2 boolean features

Features with missing values:
AVG_ORDER_VOLUME_2023        4291
DELIVERY_COST_2023_CASES       54
DELIVERY_COST_2023_GALLON      54
dtype: int64
Training set: 18265 samples
Test set: 4567 samples
Positive class (growth) in train: 1014 (5.55%)
Positive class (growth) in test: 254 (5.56%)

Training models to predict growth potential...
Applying SMOTE to handle class imbalance...
After SMOTE - Training samples: 34502, Positive class: 17251(50.00%)

Training logistic_regression...

Training random_forest...

Training gradient_boosting...

Training xgboost...

Evaluating model performance...

Evaluating logistic_regression...
ROC AUC: 0.7916

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.77      0.86      4313
           1       0.15      0.67      0.24       254

    accuracy                           0.76      4567
   macro avg       0.56      0.72      0.55      4567
weighted avg       0.93      0.76      0.82      4567


Confusion Matrix:
[[3311 1002]
 [  84  170]]

Evaluating random_forest...
ROC AUC: 0.8091

Classification Report:
              precision    recall  f1-score   support
           0       0.96      0.95      0.95      4313
           1       0.24      0.26      0.25       254

    accuracy                           0.91      4567
   macro avg       0.60      0.61      0.60      4567
weighted avg       0.92      0.91      0.91      4567


Confusion Matrix:
[[4102  211]
 [ 187   67]]

Evaluating gradient_boosting...
ROC AUC: 0.8502

Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      4313
           1       0.59      0.07      0.13       254

    accuracy                           0.95      4567
   macro avg       0.77      0.54      0.55      4567
weighted avg       0.93      0.95      0.93      4567


Confusion Matrix:
[[4300   13]
 [ 235   19]]

Evaluating xgboost...
ROC AUC: 0.8459

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      4313
           1       0.55      0.14      0.23       254

    accuracy                           0.95      4567
   macro avg       0.75      0.57      0.60      4567
weighted avg       0.93      0.95      0.93      4567


Confusion Matrix:
[[4283   30]
 [ 218   36]]

Best model: gradient_boosting with ROC AUC: 0.8502

2.Creating derived features...
Added derived 1 variables. New shapes are: (30755, 40)

400 Annual Threshold distribution:
----------------------------------------------------------
```

```
2023: Below threshold: 22832 (74.24%), Above threshold: 7923 (25.76%)
2024: Below threshold: 22720 (73.87%), Above threshold: 8035 (26.13%)
----------------------------------------------------------

The number of customer proportion between thresholds:
----------------------------------------------------------
Below to Above (Growth): 1268 (4.12% of total)
Above to Below (Decline): 1156 (3.76% of total)
Below to Below (No Change): 21564 (70.12% of total)
Above to Above (No Change): 6767 (22.00% of total)
----------------------------------------------------------

3.Current threshold analysis:

If 400 Annual Threshold were applied...


Volume Change
----------------------------------------------------------

Total annumal volume change from 2023 to 2024
2023: 18937111.54 units
2024: 19928814.14 units
Change: 991702.59 units
----------------------------------------------------------

Total annual volume change for no change (below to below):
2023: 2373218.54 units
2024: 2518678.75 units
Change: 145460.21 units

Total annual volume change for growth customers (below to above):
2023: 219940.73 units
2024: 1528993.19 units
Change: 1309052.46 units

Total annual volume change for no change (above to above):
2023: 15560010.91 units
2024: 15599958.61 units
Change: 39947.69 units

Total annual volume change for decline customers (above to below):
2023: 783941.36 units
2024: 281183.60 units
Change: -502757.77 units
----------------------------------------------------------


Delivery cost change
----------------------------------------------------------

Total annual delivery cost change from 2023 to 2024
2023: $34298477.62
2024: $36012190.37
Change: $1713712.75
----------------------------------------------------------
```

```
Total delivery cost change for no change (below to below):
2023: $11703364.74
2024: $12460361.75
Change: $756997.02

Total delivery cost change for growth customers (below to above):
2023: $960184.35
2024: $3073571.48
Change: $2113387.14

Total delivery cost change for no change (above to above):
2023: $19359555.34
2024: $19284711.41
Change: $-74843.93

Total delivery cost change for decline customers (above to below):
2023: $2275373.19
2024: $1193545.72
Change: $-1081827.47
----------------------------------------------------------

2.Creating derived features...
Added derived 1 variables. New shapes are: (30755, 40)

300 Annual Threshold distribution:
----------------------------------------------------------
2023: Below threshold: 20801 (67.63%), Above threshold: 9954 (32.37%)
2024: Below threshold: 20641 (67.11%), Above threshold: 10114 (32.89%)
----------------------------------------------------------

The number of customer proportion between thresholds:
----------------------------------------------------------
Below to Above (Growth): 1620 (5.27% of total)
Above to Below (Decline): 1460 (4.75% of total)
Below to Below (No Change): 19181 (62.37% of total)
Above to Above (No Change): 8494 (27.62% of total)
----------------------------------------------------------

3.Current threshold analysis:

If 300 Annual Threshold were applied...


Volume Change
----------------------------------------------------------

Total annumal volume change from 2023 to 2024
2023: 18937111.54 units
2024: 19928814.14 units
Change: 991702.59 units
----------------------------------------------------------

Total annual volume change for no change (below to below):
2023: 1689274.22 units
2024: 1824583.22 units
```

```
Change: 135309.00 units

Total annual volume change for growth customers (below to above):
2023: 201061.05 units
2024: 1572895.87 units
Change: 1371834.82 units

Total annual volume change for no change (above to above):
2023: 16298531.30 units
2024: 16279094.34 units
Change: -19436.97 units

Total annual volume change for decline customers (above to below):
2023: 748244.97 units
2024: 252240.71 units
Change: -496004.26 units
----------------------------------------------------------


Delivery cost change
----------------------------------------------------------

Total annumal delivery cost change from 2023 to 2024
2023: $34298477.62
2024: $36012190.37
Change: $1713712.75
----------------------------------------------------------

Total delivery cost change for no change (below to below):
2023: $8868875.45
2024: $9595758.97
Change: $726883.52

Total delivery cost change for growth customers (below to above):
2023: $975255.34
2024: $3376077.45
Change: $2400822.11

Total delivery cost change for no change (above to above):
2023: $21957570.07
2024: $21806960.73
Change: $-150609.34

Total delivery cost change for decline customers (above to below):
2023: $2496776.77
2024: $1233393.22
Change: $-1263383.54
----------------------------------------------------------
when 400 threshold applied, the number of growth customer is 1268 (4.12% of total)
when 300 threshold applied, the number of growth customer is 1620 (5.27% of total)


when 400 threshold applied, direct delivery cost increases by: $2038543.21(2023 to 2
024)
when 300 threshold applied, direct delivery cost increases by: $2250212.77(2023 to 2
024)
```

```
when 400 threshold applied, growth customer volume increase by: 1309052.46 units (20
23 to 2024)
when 300 threshold applied, growth customer volume increase by: 1371834.82 units (20
23 to 2024)


when price is $5 and 300 threshold applied, net revenue per annual increases by $264
76.83 compared with that of 400 threshold.

Model saved to sccu_growth_model.pkl
```

Customer Segmentation based on the transition status

In [ ]:
```python
sccu_segmentation = file_path.copy() # Make copy of dataframe to avoid modifiying o
```

Create Transition Status Variable

One of our objectives is to determine the factors that enable a customer to move from Below Threshold in 2023 to Above Threshold in 2024. Thus, to help the model better capture that information, a new variable will be created. This new variable is called Transition Status and it tracks which customers moved from Below_Threshold in 2023 to Above_Threshold in 2024 via the "Up" level in the variable. After this new variable is created, it'll be remapped with "Up" getting a value of 1. This will allow Python to model "Up" response in the "Transition Status" for any machine learning models that are constructed.

In [ ]:
```python
# Define conditions
conditions = [
    (sccu_segmentation['THRESHOLD_2023'] == "below") & (sccu_segmentation['THRESHOL
    (sccu_segmentation['THRESHOLD_2023'] == "above") & (sccu_segmentation['THRESHOL
    (sccu_segmentation['THRESHOLD_2023'] == "above") & (sccu_segmentation['THRESHOL
    (sccu_segmentation['THRESHOLD_2023'] == "below") & (sccu_segmentation['THRESHOL
]

# Define corresponding labels
labels = ["Up", "Down", "Stays above Threshold", "Stays below Threshold"]

# Apply conditions to create the new column
sccu_segmentation['Transition_Status'] = np.select(conditions, labels, default="Unk

# Recode this column with numeric values for machine-learning purposes
sccu_segmentation['Transition_Status'] = sccu_segmentation['Transition_Status'].map
    'Up': 1,
    'Down': 0,
    'Stays above Threshold': 0,
    'Stays below Threshold': 0
})
```

```
print('Transition_Status' in sccu_segmentation.columns)
```

```
True
```

In [ ]:
```
sccu_segmentation.head() # See Dataframe and check that everything looks normal.
```

Out[ ]:

	Unnamed: 0	CUSTOMER_NUMBER	FIRST_TRANSACTION_DATE	LAST_TRANSACTION_DATE
0	1	500245678	2023-01-09	2024-11-20
1	2	500245685	2023-01-06	2024-08-16
2	3	500245686	2023-03-07	2024-12-17
3	4	500245687	2023-02-06	2024-10-28
4	5	500245689	2023-01-13	2024-12-26

5 rows × 38 columns

Dataset Cleaning

The dataset has some variables that must be cleaned before machine-learning models like Decision Trees, XGboost, etc. can be implemeneted. For instance, the dataset has several columns related to 2024. This however is problematic since it basically gives all of the answers to the model. The reason is because if the model has 2024 columns, it can directly map those columns to whether a model is above or below the threshold which causes the model to have very high accurate results. This however is not very accurate because the model has been given information about the 2024 columns which would strongly bias results.

Additionally in a real scenario, Swire Coca Cola would not have access to future data. For example, Swire would not have access to 2026 data at the time of this notebook's creation (2025). Thus, to model if a customer will transition from 2023 to 2024, the model should have any 2024 columns because it could be considered the "future" when viewed from the year 2023.

In [ ]:
```
# Create function to avoid retyping code later
def cleaning_data(dataset):
    # Drop these columns from the dataset, Threshold_2024 is target and the other v
    transformed_df = dataset.drop(columns=['FIRST_TRANSACTION_DATE', 'LAST_TRANSACT
                                  'TRANS_DAYS',"TRANS_COUNT_2024","ANNUAL_VOLUME_CAS
                                  'AVG_ORDER_VOLUME_2024','CHANGED_VOLUME','PERCENT_
                                  "DELIVERY_COST_2024","DELIVERY_COST_2024_CASES"])

    # Create list with following variables to one hot encode
    categorical_cols = ['THRESHOLD_2023', 'FREQUENT_ORDER_TYPE','COLD_DRINK_CHANNEL
    # Convert variables to one hot encoded dummy variables and assign back to trans
    transformed_df = pd.get_dummies(transformed_df, columns=categorical_cols, drop_

    # Return the Dataset
    return transformed_df

sccu_segmentation_01 = cleaning_data(sccu_segmentation)

sccu_segmentation_01.head()
```

Out[ ]:

	TRANS_COUNT_2023	ANNUAL_VOLUME_CASES_2023	ANNUAL_VOLUME_GALLON_2023
0	27	210.0	160.0
1	46	24.0	577.5
2	5	17.5	0.0
3	9	0.0	125.0
4	40	124.0	422.5

5 rows × 101 columns

The below code splits the data into a train and test set for cross-validation. The split is 80-20 with 80% of the data in the train set and 20% of the dataset in the test set.

In [ ]:
```
def cross_validation(dataset):
    X = dataset.drop(columns=['Transition_Status'])  # Features
    # Y dataset will contain our target variable
    y = dataset['Transition_Status']  # Target variable
    return X,y

X_dataset, target = cross_validation(sccu_segmentation_01)
```

In [ ]:
```
# Drop rows with NaN values from X and y
X_dataset = X_dataset.dropna()
target= target[X_dataset.index]  # Ensure that y is aligned with the cleaned X

# Check that there are no NaN values left in X and y
print(X_dataset.isna().sum())  # Should show 0 for all columns
print(target.isna().sum())  # Should show 0 for the target variable
```

```
TRANS_COUNT_2023                        0
ANNUAL_VOLUME_CASES_2023                0
ANNUAL_VOLUME_GALLON_2023               0
ANNUAL_VOLUME_2023                      0
AVG_ORDER_VOLUME_2023                   0
                                       ..
SUB_TRADE_CHANNEL_SANDWICH FAST FOOD    0
STATE_SHORT_KY                          0
STATE_SHORT_LA                          0
STATE_SHORT_MA                          0
STATE_SHORT_MD                          0
Length: 100, dtype: int64
0
```

Iteratively Building Decision Trees

Decision Trees will be built in an iterative manner by removing the top predictor each time and then analyzing the top predictors along with the ROC-AUC scores. The reason for building Decision Trees iteratively by removing the top predictors is to see if there are any common trends in the top predictors. For instance, are all the top predictors related to a particular trade channel, are the predictors, states, etc.

Additionally, if there is are particular predictors that heavily dominate, we can see what predictors are most influential when the heavy dominant predictors are removed.

```
In [ ]: def iterative_feature_importance(dataframe, target_variable, iterations=5, random_s
            importance_tracking = {}
            metrics_tracking = {}

            for i in range(iterations):
                # Split data with stratification to preserve class proportions
                X_train, X_test, y_train, y_test = train_test_split(
                    dataframe, target_variable, test_size=0.2, stratify=target_variable, ra
                )

                # Train decision tree
                dt_model = DecisionTreeClassifier(random_state=random_state)
                dt_model.fit(X_train, y_train)

                # Get feature importances
                feature_importances = pd.DataFrame({
                    'Feature': X_train.columns,
                    'Importance': dt_model.feature_importances_
                }).sort_values(by="Importance", ascending=False)

                # Store feature importance results
                importance_tracking[f'Iteration {i+1}'] = feature_importances

                # Make predictions
                y_train_pred = dt_model.predict(X_train)
                y_test_pred = dt_model.predict(X_test)

                # Calculate metrics
```

```
        metrics_tracking[f'Iteration {i+1}'] = {
            'Train Accuracy': accuracy_score(y_train, y_train_pred),
            'Test Accuracy': accuracy_score(y_test, y_test_pred),
            'Train F1 Score': f1_score(y_train, y_train_pred),
            'Test F1 Score': f1_score(y_test, y_test_pred),
            'Train Precision': precision_score(y_train, y_train_pred),
            'Test Precision': precision_score(y_test, y_test_pred),
            'Train Recall': recall_score(y_train, y_train_pred),
            'Test Recall': recall_score(y_test, y_test_pred),
            'Train ROC-AUC': roc_auc_score(y_train,y_train_pred),
            'Test ROC-AUC': roc_auc_score(y_test,y_test_pred)
        }

        # Drop the most important feature
        most_important = feature_importances.iloc[0]['Feature']
        dataframe = dataframe.drop(columns=[most_important])
        target_variable = target_variable  # Keep target variable unchanged

    return importance_tracking, metrics_tracking

# Utilizing the function by creating two variables to capture feature importance an
importance_results, metrics_results = iterative_feature_importance(X_dataset, targe

# Display feature importance results
for iteration, df in importance_results.items():
    print(f"\n{iteration} Feature Importances:\n")
    print(df)

# Display metrics results
for iteration, metrics in metrics_results.items():
    print(f"\n{iteration} Metrics:\n")
    for metric, value in metrics.items():
        print(f"{metric}: {value:.4f}")
```

```
Iteration 1 Feature Importances:

                                        Feature  Importance
4                         AVG_ORDER_VOLUME_2023    0.185146
3                            ANNUAL_VOLUME_2023    0.153472
9                            DELIVERY_COST_2023    0.099630
7                      DELIVERY_COST_2023_CASES    0.064812
1                      ANNUAL_VOLUME_CASES_2023    0.053348
..                                          ...         ...
16                 COLD_DRINK_CHANNEL_BULK TRADE    0.000000
59                  SUB_TRADE_CHANNEL_FRATERNITY    0.000000
64  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
66               SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
50        SUB_TRADE_CHANNEL_BULK BEVERAGE RETAIL    0.000000

[100 rows x 2 columns]

Iteration 2 Feature Importances:

                                        Feature  Importance
3                            ANNUAL_VOLUME_2023    0.214236
0                              TRANS_COUNT_2023    0.118523
8                            DELIVERY_COST_2023    0.113593
6                      DELIVERY_COST_2023_CASES    0.072382
1                      ANNUAL_VOLUME_CASES_2023    0.054325
..                                          ...         ...
56                      SUB_TRADE_CHANNEL_CRUISE    0.000000
58                  SUB_TRADE_CHANNEL_FRATERNITY    0.000000
63  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
65               SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
49        SUB_TRADE_CHANNEL_BULK BEVERAGE RETAIL    0.000000

[99 rows x 2 columns]

Iteration 3 Feature Importances:

                                        Feature  Importance
7                            DELIVERY_COST_2023    0.174514
0                              TRANS_COUNT_2023    0.131429
1                      ANNUAL_VOLUME_CASES_2023    0.110118
5                      DELIVERY_COST_2023_CASES    0.090365
2                     ANNUAL_VOLUME_GALLON_2023    0.078212
..                                          ...         ...
57                  SUB_TRADE_CHANNEL_FRATERNITY    0.000000
62  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
64               SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
68                SUB_TRADE_CHANNEL_ONLINE STORE    0.000000
49                  SUB_TRADE_CHANNEL_BULK TRADE    0.000000

[98 rows x 2 columns]

Iteration 4 Feature Importances:

                                        Feature  Importance
1                      ANNUAL_VOLUME_CASES_2023    0.184574
0                              TRANS_COUNT_2023    0.145315
```

```
5                     DELIVERY_COST_2023_CASES    0.125887
2                    ANNUAL_VOLUME_GALLON_2023    0.083382
6                    DELIVERY_COST_2023_GALLON    0.073204
..                                         ...         ...
50               SUB_TRADE_CHANNEL_CHAIN STORE    0.000000
54                    SUB_TRADE_CHANNEL_CRUISE    0.000000
56                SUB_TRADE_CHANNEL_FRATERNITY    0.000000
61  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
48                SUB_TRADE_CHANNEL_BULK TRADE    0.000000

[97 rows x 2 columns]

Iteration 5 Feature Importances:

                                      Feature  Importance
4                     DELIVERY_COST_2023_CASES    0.250694
0                             TRANS_COUNT_2023    0.152864
1                    ANNUAL_VOLUME_GALLON_2023    0.094890
5                    DELIVERY_COST_2023_GALLON    0.080379
59           SUB_TRADE_CHANNEL_HOME & HARDWARE    0.034464
..                                         ...         ...
76      SUB_TRADE_CHANNEL_OTHER LARGE RETAILER    0.000000
55                SUB_TRADE_CHANNEL_FRATERNITY    0.000000
60  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
22                    TRADE_CHANNEL_BULK TRADE    0.000000
51                      SUB_TRADE_CHANNEL_CLUB    0.000000

[96 rows x 2 columns]

Iteration 6 Feature Importances:

                                      Feature  Importance
0                             TRANS_COUNT_2023    0.275525
1                    ANNUAL_VOLUME_GALLON_2023    0.151715
4                    DELIVERY_COST_2023_GALLON    0.108037
93                              STATE_SHORT_MA    0.045012
10               FREQUENT_ORDER_TYPE_SALES REP    0.040049
..                                         ...         ...
54                SUB_TRADE_CHANNEL_FRATERNITY    0.000000
59  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
61             SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
65              SUB_TRADE_CHANNEL_ONLINE STORE    0.000000
52                    SUB_TRADE_CHANNEL_CRUISE    0.000000

[95 rows x 2 columns]

Iteration 7 Feature Importances:

                                      Feature  Importance
3                    DELIVERY_COST_2023_GALLON    0.211872
0                    ANNUAL_VOLUME_GALLON_2023    0.190956
92                              STATE_SHORT_MA    0.054642
2                                 CO2_CUSTOMER    0.052778
9                FREQUENT_ORDER_TYPE_SALES REP    0.051636
..                                         ...         ...
50   SUB_TRADE_CHANNEL_COMPREHENSIVE PROVIDER    0.000000
```

```
53               SUB_TRADE_CHANNEL_FRATERNITY    0.000000
58  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
60            SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
47              SUB_TRADE_CHANNEL_CHAIN STORE    0.000000

[94 rows x 2 columns]

Iteration 8 Feature Importances:

                                      Feature  Importance
0                  ANNUAL_VOLUME_GALLON_2023    0.411002
56          SUB_TRADE_CHANNEL_HOME & HARDWARE    0.047752
2                                CO2_CUSTOMER    0.047067
8              FREQUENT_ORDER_TYPE_SALES REP    0.044167
89                             STATE_SHORT_KY    0.040544
..                                        ...         ...
57  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
59            SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
60                     SUB_TRADE_CHANNEL_MISC    0.000000
63             SUB_TRADE_CHANNEL_ONLINE STORE    0.000000
46              SUB_TRADE_CHANNEL_CHAIN STORE    0.000000

[93 rows x 2 columns]

Iteration 9 Feature Importances:

                                   Feature  Importance
55       SUB_TRADE_CHANNEL_HOME & HARDWARE    0.140505
1                             CO2_CUSTOMER    0.099568
90                          STATE_SHORT_MA    0.081266
4        FREQUENT_ORDER_TYPE_MYCOKE LEGACY    0.067357
2                     THRESHOLD_2023_below    0.066978
..                                     ...         ...
27                TRADE_CHANNEL_INDUSTRIAL    0.000000
72  SUB_TRADE_CHANNEL_OTHER LARGE RETAILER    0.000000
31   TRADE_CHANNEL_OTHER DINING & BEVERAGE    0.000000
70      SUB_TRADE_CHANNEL_OTHER HEALTHCARE    0.000000
39                    TRADE_CHANNEL_TRAVEL    0.000000

[92 rows x 2 columns]

Iteration 10 Feature Importances:

                                   Feature  Importance
66  SUB_TRADE_CHANNEL_OTHER GENERAL RETAIL    0.135842
1                             CO2_CUSTOMER    0.094542
89                          STATE_SHORT_MA    0.079542
4        FREQUENT_ORDER_TYPE_MYCOKE LEGACY    0.071982
2                     THRESHOLD_2023_below    0.066978
..                                     ...         ...
42  SUB_TRADE_CHANNEL_BULK BEVERAGE RETAIL    0.000000
40              TRADE_CHANNEL_VEHICLE CARE    0.000000
38                TRADE_CHANNEL_SUPERSTORE    0.000000
37        TRADE_CHANNEL_SPECIALIZED GOODS    0.000000
45           SUB_TRADE_CHANNEL_CHAIN STORE    0.000000
```

```
[91 rows x 2 columns]

Iteration 1 Metrics:

Train Accuracy: 0.9998
Test Accuracy: 0.9515
Train F1 Score: 0.9971
Test F1 Score: 0.2242
Train Precision: 1.0000
Test Precision: 0.2357
Train Recall: 0.9942
Test Recall: 0.2139
Train ROC-AUC: 0.9971
Test ROC-AUC: 0.5952

Iteration 2 Metrics:

Train Accuracy: 0.9998
Test Accuracy: 0.9432
Train F1 Score: 0.9971
Test F1 Score: 0.1667
Train Precision: 1.0000
Test Precision: 0.1604
Train Recall: 0.9942
Test Recall: 0.1734
Train ROC-AUC: 0.9971
Test ROC-AUC: 0.5713

Iteration 3 Metrics:

Train Accuracy: 0.9998
Test Accuracy: 0.9513
Train F1 Score: 0.9971
Test F1 Score: 0.2678
Train Precision: 1.0000
Test Precision: 0.2640
Train Recall: 0.9942
Test Recall: 0.2717
Train ROC-AUC: 0.9971
Test ROC-AUC: 0.6230

Iteration 4 Metrics:

Train Accuracy: 0.9998
Test Accuracy: 0.9540
Train F1 Score: 0.9971
Test F1 Score: 0.3231
Train Precision: 1.0000
Test Precision: 0.3118
Train Recall: 0.9942
Test Recall: 0.3353
Train ROC-AUC: 0.9971
Test ROC-AUC: 0.6551

Iteration 5 Metrics:
```

```
Train Accuracy: 0.9998
Test Accuracy: 0.9523
Train F1 Score: 0.9971
Test F1 Score: 0.3226
Train Precision: 1.0000
Test Precision: 0.3015
Train Recall: 0.9942
Test Recall: 0.3468
Train ROC-AUC: 0.9971
Test ROC-AUC: 0.6598

Iteration 6 Metrics:

Train Accuracy: 0.9936
Test Accuracy: 0.9536
Train F1 Score: 0.8927
Test F1 Score: 0.2173
Train Precision: 0.9826
Test Precision: 0.2429
Train Recall: 0.8179
Test Recall: 0.1965
Train ROC-AUC: 0.9087
Test ROC-AUC: 0.5879

Iteration 7 Metrics:

Train Accuracy: 0.9850
Test Accuracy: 0.9589
Train F1 Score: 0.7162
Test F1 Score: 0.2439
Train Precision: 0.9412
Test Precision: 0.3070
Train Recall: 0.5780
Test Recall: 0.2023
Train ROC-AUC: 0.7884
Test ROC-AUC: 0.5934

Iteration 8 Metrics:

Train Accuracy: 0.9850
Test Accuracy: 0.9585
Train F1 Score: 0.7167
Test F1 Score: 0.2526
Train Precision: 0.9391
Test Precision: 0.3083
Train Recall: 0.5795
Test Recall: 0.2139
Train ROC-AUC: 0.7891
Test ROC-AUC: 0.5988

Iteration 9 Metrics:

Train Accuracy: 0.9707
Test Accuracy: 0.9669
Train F1 Score: 0.2463
Test F1 Score: 0.1706
```

```
Train Precision: 0.7891
Test Precision: 0.4737
Train Recall: 0.1460
Test Recall: 0.1040
Train ROC-AUC: 0.5723
Test ROC-AUC: 0.5501

Iteration 10 Metrics:

Train Accuracy: 0.9707
Test Accuracy: 0.9669
Train F1 Score: 0.2463
Test F1 Score: 0.1706
Train Precision: 0.7891
Test Precision: 0.4737
Train Recall: 0.1460
Test Recall: 0.1040
Train ROC-AUC: 0.5723
Test ROC-AUC: 0.5501
```

VOLUME related variables are still the most important predictors for determining whether a customer can transition from below threshold to above threshold. It is only around the 9th-10th iteration that other variable types start to appear. The accuracy is great, but all the other metrics that measure class differences suffer tremendously. For instance, in the last few iterations, the ROC-AUC score is slightly better than Random Guessing at 54-55% respectively.

To help solve this issue and make the model more generalizable, data-imbalancing techniques like SMOTE, Up-Sampling, Down-Sampling, etc. will have to be utilized. The importance is determined by how many times the Decision Tree used the variable to split the data.

SMOTE

SMOTE will create synthetic datapoints that give more representation to the minority class.

In [ ]:
```
print("Before SMOTE:", Counter(target))
print("target (minority class) proportion:", str(round(1250/(29072 + 1250),4)*100)
```

```
Before SMOTE: Counter({0: 25541, 1: 865})
target (minority class) proportion: 4.12%
 majority class proportion: 95.88%
```

In [ ]:
```
def ifis(dataframe, target_variable, iterations=5, random_state=42): # ifis = itera
    importance_tracking = {}
    metrics_tracking = {}

    for i in range(iterations):
        # Split data with stratification to preserve class proportions
        X_train, X_test, y_train, y_test = train_test_split(
            dataframe, target_variable, test_size=0.2, stratify=target_variable, ra
        )

        boolean_cols = X_train.select_dtypes(include=['bool']).columns
        X_train[boolean_cols] = X_train[boolean_cols].astype(int)

        # Apply SMOTE
        smote = SMOTE(random_state=42)
        X_resampled, y_resampled = smote.fit_resample(X_train, y_train.astype(int))

        # Train decision tree
        dt_model = DecisionTreeClassifier(random_state=random_state)
        dt_model.fit(X_resampled, y_resampled)

        # Get feature importances
        feature_importances = pd.DataFrame({
            'Feature': X_resampled.columns,
            'Importance': dt_model.feature_importances_
        }).sort_values(by="Importance", ascending=False)

        # Store feature importance results
        importance_tracking[f'Iteration {i+1}'] = feature_importances

        # Make predictions
        y_train_pred = dt_model.predict(X_train)
        y_test_pred = dt_model.predict(X_test)

        # Calculate metrics
        metrics_tracking[f'Iteration {i+1}'] = {
            'Train ROC-AUC': roc_auc_score(y_train,y_train_pred),
            'Test ROC-AUC': roc_auc_score(y_test,y_test_pred),
            'Train Accuracy': accuracy_score(y_train, y_train_pred),
            'Test Accuracy': accuracy_score(y_test, y_test_pred),
            'Train F1 Score': f1_score(y_train, y_train_pred),
            'Test F1 Score': f1_score(y_test, y_test_pred),
            'Train Precision': precision_score(y_train, y_train_pred),
            'Test Precision': precision_score(y_test, y_test_pred),
            'Train Recall': recall_score(y_train, y_train_pred),
            'Test Recall': recall_score(y_test, y_test_pred)
        }

        # Drop the most important feature
        most_important = feature_importances.iloc[0]['Feature']
        dataframe = dataframe.drop(columns=[most_important])
        target_variable = target_variable  # Keep target variable unchanged

    return importance_tracking, metrics_tracking

# Utilizing the function by creating two variables to capture feature importance an
importance_results, metrics_results = ifis(X_dataset, target, iterations=10)

# Display feature importance results
for iteration, df in importance_results.items():
    print(f"\n{iteration} Feature Importances:\n")
    print(df)

# Display metrics results
for iteration, metrics in metrics_results.items():
    print(f"\n{iteration} Metrics:\n")
    for metric, value in metrics.items():
        print(f"{metric}: {value:.4f}")
```

```
Iteration 1 Feature Importances:

                                   Feature  Importance
3                       ANNUAL_VOLUME_2023    0.308262
4                    AVG_ORDER_VOLUME_2023    0.239902
9                       DELIVERY_COST_2023    0.088222
15          FREQUENT_ORDER_TYPE_SALES REP    0.026962
14              FREQUENT_ORDER_TYPE_OTHER    0.026891
..                                     ...         ...
47                    TRADE_CHANNEL_TRAVEL    0.000000
46                TRADE_CHANNEL_SUPERSTORE    0.000000
41         TRADE_CHANNEL_PHARMACY RETAILER    0.000000
36      TRADE_CHANNEL_LARGE-SCALE RETAILER    0.000000
50  SUB_TRADE_CHANNEL_BULK BEVERAGE RETAIL    0.000000

[100 rows x 2 columns]

Iteration 2 Feature Importances:

                                      Feature  Importance
3                       AVG_ORDER_VOLUME_2023    0.261251
9                        THRESHOLD_2023_below    0.182614
8                          DELIVERY_COST_2023    0.095566
0                            TRANS_COUNT_2023    0.066198
1                     ANNUAL_VOLUME_CASES_2023    0.029182
..                                        ...         ...
79     SUB_TRADE_CHANNEL_OTHER LARGE RETAILER    0.000000
58               SUB_TRADE_CHANNEL_FRATERNITY    0.000000
63  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
65            SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
49     SUB_TRADE_CHANNEL_BULK BEVERAGE RETAIL    0.000000

[99 rows x 2 columns]

Iteration 3 Feature Importances:

                                      Feature  Importance
7                          DELIVERY_COST_2023    0.220201
8                        THRESHOLD_2023_below    0.182614
13              FREQUENT_ORDER_TYPE_SALES REP    0.053525
1                     ANNUAL_VOLUME_CASES_2023    0.052866
0                            TRANS_COUNT_2023    0.050527
..                                        ...         ...
15            COLD_DRINK_CHANNEL_CONVENTIONAL    0.000000
62  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
64            SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
65                     SUB_TRADE_CHANNEL_MISC    0.000000
56                    SUB_TRADE_CHANNEL_FAITH    0.000000

[98 rows x 2 columns]

Iteration 4 Feature Importances:

                                      Feature  Importance
7                        THRESHOLD_2023_below    0.182614
2                    ANNUAL_VOLUME_GALLON_2023    0.121642
```

```
1                    ANNUAL_VOLUME_CASES_2023    0.110691
5                    DELIVERY_COST_2023_CASES    0.056772
0                            TRANS_COUNT_2023    0.046452
..                                        ...         ...
52                     SUB_TRADE_CHANNEL_CLUB    0.000000
61  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
54                   SUB_TRADE_CHANNEL_CRUISE    0.000000
59              SUB_TRADE_CHANNEL_HIGH SCHOOL    0.000000
48               SUB_TRADE_CHANNEL_BULK TRADE    0.000000

[97 rows x 2 columns]

Iteration 5 Feature Importances:

                              Feature  Importance
1          ANNUAL_VOLUME_CASES_2023    0.217131
2         ANNUAL_VOLUME_GALLON_2023    0.107956
11    FREQUENT_ORDER_TYPE_SALES REP    0.083024
5          DELIVERY_COST_2023_CASES    0.070894
6         DELIVERY_COST_2023_GALLON    0.070179
..                              ...         ...
53         SUB_TRADE_CHANNEL_CRUISE    0.000000
47     SUB_TRADE_CHANNEL_BULK TRADE    0.000000
62  SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
42         TRADE_CHANNEL_SUPERSTORE    0.000000
49    SUB_TRADE_CHANNEL_CHAIN STORE    0.000000

[96 rows x 2 columns]

Iteration 6 Feature Importances:

                                      Feature  Importance
4                    DELIVERY_COST_2023_CASES    0.167010
10              FREQUENT_ORDER_TYPE_SALES REP    0.108827
1                   ANNUAL_VOLUME_GALLON_2023    0.095910
5                   DELIVERY_COST_2023_GALLON    0.083307
9                   FREQUENT_ORDER_TYPE_OTHER    0.064018
..                                        ...         ...
57              SUB_TRADE_CHANNEL_HIGH SCHOOL    0.000000
59  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
61            SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
65             SUB_TRADE_CHANNEL_ONLINE STORE    0.000000
52                   SUB_TRADE_CHANNEL_CRUISE    0.000000

[95 rows x 2 columns]

Iteration 7 Feature Importances:

                                   Feature  Importance
0                         TRANS_COUNT_2023    0.115831
1                ANNUAL_VOLUME_GALLON_2023    0.114489
4                DELIVERY_COST_2023_GALLON    0.075267
9            FREQUENT_ORDER_TYPE_SALES REP    0.073796
7            FREQUENT_ORDER_TYPE_MYCOKE360    0.054526
..                                     ...         ...
44  SUB_TRADE_CHANNEL_BULK BEVERAGE RETAIL    0.000000
```

```
49                  SUB_TRADE_CHANNEL_CLUB    0.000000
64          SUB_TRADE_CHANNEL_ONLINE STORE    0.000000
60         SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
47           SUB_TRADE_CHANNEL_CHAIN STORE    0.000000

[94 rows x 2 columns]

Iteration 8 Feature Importances:

                                      Feature  Importance
0                   ANNUAL_VOLUME_GALLON_2023    0.193431
3                   DELIVERY_COST_2023_GALLON    0.076114
8               FREQUENT_ORDER_TYPE_SALES REP    0.053409
6               FREQUENT_ORDER_TYPE_MYCOKE360    0.046120
92                             STATE_SHORT_MD    0.043662
..                                        ...         ...
48                     SUB_TRADE_CHANNEL_CLUB    0.000000
54              SUB_TRADE_CHANNEL_GAME CENTER    0.000000
57  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
59            SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
46              SUB_TRADE_CHANNEL_CHAIN STORE    0.000000

[93 rows x 2 columns]

Iteration 9 Feature Importances:

                                      Feature  Importance
2                   DELIVERY_COST_2023_GALLON    0.236027
7               FREQUENT_ORDER_TYPE_SALES REP    0.067845
5               FREQUENT_ORDER_TYPE_MYCOKE360    0.058635
91                             STATE_SHORT_MD    0.053271
88                             STATE_SHORT_KY    0.049715
..                                        ...         ...
51             SUB_TRADE_CHANNEL_FRATERNITY    0.000000
56  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
72     SUB_TRADE_CHANNEL_OTHER LARGE RETAILER    0.000000
58            SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
9             COLD_DRINK_CHANNEL_CONVENTIONAL    0.000000

[92 rows x 2 columns]

Iteration 10 Feature Importances:

                                      Feature  Importance
1                                CO2_CUSTOMER    0.104816
0                        LOCAL_MARKET_PARTNER    0.069459
2                    FREQUENT_ORDER_TYPE_EDI    0.055950
4               FREQUENT_ORDER_TYPE_MYCOKE360    0.042835
54           SUB_TRADE_CHANNEL_HOME & HARDWARE    0.039004
..                                        ...         ...
48                   SUB_TRADE_CHANNEL_CRUISE    0.000000
55  SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE    0.000000
57            SUB_TRADE_CHANNEL_MIDDLE SCHOOL    0.000000
71     SUB_TRADE_CHANNEL_OTHER LARGE RETAILER    0.000000
8             COLD_DRINK_CHANNEL_CONVENTIONAL    0.000000
```

```
[91 rows x 2 columns]

Iteration 1 Metrics:

Train ROC-AUC: 0.9971
Test ROC-AUC: 0.5968
Train Accuracy: 0.9998
Test Accuracy: 0.9385
Train F1 Score: 0.9971
Test F1 Score: 0.1975
Train Precision: 1.0000
Test Precision: 0.1724
Train Recall: 0.9942
Test Recall: 0.2312

Iteration 2 Metrics:

Train ROC-AUC: 0.9971
Test ROC-AUC: 0.6094
Train Accuracy: 0.9998
Test Accuracy: 0.9358
Train F1 Score: 0.9971
Test F1 Score: 0.2098
Train Precision: 1.0000
Test Precision: 0.1758
Train Recall: 0.9942
Test Recall: 0.2601

Iteration 3 Metrics:

Train ROC-AUC: 0.9971
Test ROC-AUC: 0.6282
Train Accuracy: 0.9998
Test Accuracy: 0.9398
Train F1 Score: 0.9971
Test F1 Score: 0.2429
Train Precision: 1.0000
Test Precision: 0.2065
Train Recall: 0.9942
Test Recall: 0.2948

Iteration 4 Metrics:

Train ROC-AUC: 0.9971
Test ROC-AUC: 0.6119
Train Accuracy: 0.9998
Test Accuracy: 0.9406
Train F1 Score: 0.9971
Test F1 Score: 0.2228
Train Precision: 1.0000
Test Precision: 0.1948
Train Recall: 0.9942
Test Recall: 0.2601

Iteration 5 Metrics:
```

```
Train ROC-AUC: 0.9971
Test ROC-AUC: 0.5775
Train Accuracy: 0.9998
Test Accuracy: 0.9388
Train F1 Score: 0.9971
Test F1 Score: 0.1697
Train Precision: 1.0000
Test Precision: 0.1528
Train Recall: 0.9942
Test Recall: 0.1908

Iteration 6 Metrics:

Train ROC-AUC: 0.9971
Test ROC-AUC: 0.5866
Train Accuracy: 0.9998
Test Accuracy: 0.9404
Train F1 Score: 0.9971
Test F1 Score: 0.1860
Train Precision: 1.0000
Test Precision: 0.1682
Train Recall: 0.9942
Test Recall: 0.2081

Iteration 7 Metrics:

Train ROC-AUC: 0.9324
Test ROC-AUC: 0.5945
Train Accuracy: 0.9854
Test Accuracy: 0.9339
Train F1 Score: 0.7968
Test F1 Score: 0.1865
Train Precision: 0.7310
Test Precision: 0.1562
Train Recall: 0.8757
Test Recall: 0.2312

Iteration 8 Metrics:

Train ROC-AUC: 0.8223
Test ROC-AUC: 0.5682
Train Accuracy: 0.9102
Test Accuracy: 0.8777
Train F1 Score: 0.3470
Test F1 Score: 0.1126
Train Precision: 0.2277
Test Precision: 0.0739
Train Recall: 0.7283
Test Recall: 0.2370

Iteration 9 Metrics:

Train ROC-AUC: 0.8228
Test ROC-AUC: 0.5710
Train Accuracy: 0.9098
Test Accuracy: 0.8777
```

```
Train F1 Score: 0.3465
Test F1 Score: 0.1151
Train Precision: 0.2272
Test Precision: 0.0754
Train Recall: 0.7298
Test Recall: 0.2428

Iteration 10 Metrics:

Train ROC-AUC: 0.6707
Test ROC-AUC: 0.5711
Train Accuracy: 0.7451
Test Accuracy: 0.7374
Train F1 Score: 0.1319
Test F1 Score: 0.0893
Train Precision: 0.0742
Test Precision: 0.0504
Train Recall: 0.5910
Test Recall: 0.3931
```

SMOTE did not really help that much, the AUC-ROC still has massive drops between the train and test sets. For instance, there is a 30-40% drop between the train and test sets for the first 9 iterations. The last iteration however has a drop of 10%. **VOLUME** related variables however still conintue to dominate the Decision Tree in terms of important predictors.

Undersampling

Undersampling will under sample from the majority class which are all the other responses in the Transition Variable. This will cause the majority class (all the other responses) to loose some information but it'll help the model better distinguish between customers who can transition from blow the threshold (2023) to above the threshold in 2024.

In [ ]:
```
def ifiu(dataframe, target_variable, iterations=5, random_state=42): # ifis = itera
    importance_tracking = {}
    metrics_tracking = {}

    for i in range(iterations):
        # Split data with stratification to preserve class proportions
        X_train, X_test, y_train, y_test = train_test_split(
            dataframe, target_variable, test_size=0.2, stratify=target_variable, ra
        )

        # Combine data into a single DataFrame
        df = pd.concat([X_train, y_train], axis=1)

        # Separate classes
        df_majority = df[df['Transition_Status'] == 0]
        df_minority = df[df['Transition_Status'] == 1]

        # Downsample majority class
        df_majority_downsampled = resample(df_majority,
                                           replace=False,    # Sample without replacement
                                           n_samples=len(df_minority), # Match minority siz
```

```
                                     random_state=42)

        # Combine back
        df_downsampled = pd.concat([df_majority_downsampled, df_minority])

        print(df_downsampled['Transition_Status'].value_counts())  # Now classes ar

        # Combine back
        df_upsampled = pd.concat([df_majority, df_majority_downsampled])
        print(df_upsampled['Transition_Status'].value_counts())  # # Now classes ar

        # Separate features (X) and target (y) from upsampled dataset
        X_train_upsampled = df_upsampled.drop(columns=['Transition_Status'])
        y_train_upsampled = df_upsampled['Transition_Status']

        # Initialize model
        dt_model = DecisionTreeClassifier(random_state=42)

        # Train the model on the upsampled dataset
        dt_model.fit(X_train_upsampled, y_train_upsampled)

        # Get feature importances
        feature_importances = pd.DataFrame({
            'Feature': X_train_upsampled.columns,
            'Importance': dt_model.feature_importances_
        }).sort_values(by="Importance", ascending=False)

        # Store feature importance results
        importance_tracking[f'Iteration {i+1}'] = feature_importances

        # Make predictions
        y_train_pred = dt_model.predict(X_train)
        y_test_pred = dt_model.predict(X_test)

        # Calculate metrics
        metrics_tracking[f'Iteration {i+1}'] = {
            'Train ROC-AUC': roc_auc_score(y_train,y_train_pred),
            'Test ROC-AUC': roc_auc_score(y_test,y_test_pred),
            'Train Accuracy': accuracy_score(y_train, y_train_pred),
            'Test Accuracy': accuracy_score(y_test, y_test_pred),
            'Train F1 Score': f1_score(y_train, y_train_pred),
            'Test F1 Score': f1_score(y_test, y_test_pred),
            'Train Precision': precision_score(y_train, y_train_pred),
            'Test Precision': precision_score(y_test, y_test_pred),
            'Train Recall': recall_score(y_train, y_train_pred),
            'Test Recall': recall_score(y_test, y_test_pred)
        }

        # Drop the most important feature
        most_important = feature_importances.iloc[0]['Feature']
        dataframe = dataframe.drop(columns=[most_important])
        target_variable = target_variable  # Keep target variable unchanged

    return importance_tracking, metrics_tracking

# Utilizing the function by creating two variables to capture feature importance an
```

```
importance_results, metrics_results = ifiu(X_dataset, target, iterations=10)

# Display feature importance results
for iteration, df in importance_results.items():
    print(f"\n{iteration} Feature Importances:\n")
    print(df)

# Display metrics results
for iteration, metrics in metrics_results.items():
    print(f"\n{iteration} Metrics:\n")
    for metric, value in metrics.items():
        print(f"{metric}: {value:.4f}")
```

```
Transition_Status
0    692
1    692
Name: count, dtype: int64
Transition_Status
0    21124
Name: count, dtype: int64
Transition_Status
0    692
1    692
Name: count, dtype: int64
Transition_Status
0    21124
Name: count, dtype: int64
Transition_Status
0    692
1    692
Name: count, dtype: int64
Transition_Status
0    21124
Name: count, dtype: int64
Transition_Status
0    692
1    692
Name: count, dtype: int64
Transition_Status
0    21124
Name: count, dtype: int64
Transition_Status
0    692
1    692
Name: count, dtype: int64
Transition_Status
0    21124
Name: count, dtype: int64
Transition_Status
0    692
1    692
Name: count, dtype: int64
Transition_Status
0    21124
Name: count, dtype: int64
Transition_Status
0    692
1    692
Name: count, dtype: int64
Transition_Status
0    21124
Name: count, dtype: int64
Transition_Status
0    692
1    692
Name: count, dtype: int64
Transition_Status
0    21124
Name: count, dtype: int64
```

```
Transition_Status
0    692
1    692
Name: count, dtype: int64
Transition_Status
0    21124
Name: count, dtype: int64
Transition_Status
0    692
1    692
Name: count, dtype: int64
Transition_Status
0    21124
Name: count, dtype: int64

Iteration 1 Feature Importances:

                                         Feature  Importance
0                                TRANS_COUNT_2023         0.0
63             SUB_TRADE_CHANNEL_HOME & HARDWARE         0.0
73                 SUB_TRADE_CHANNEL_OTHER DINING         0.0
72          SUB_TRADE_CHANNEL_OTHER ACCOMMODATION         0.0
71  SUB_TRADE_CHANNEL_OTHER ACADEMIC INSTITUTION         0.0
..                                           ...         ...
30              TRADE_CHANNEL_FAST CASUAL DINING         0.0
29                       TRADE_CHANNEL_EDUCATION         0.0
28                         TRADE_CHANNEL_DEFENSE         0.0
27            TRADE_CHANNEL_COMPREHENSIVE DINING         0.0
99                                STATE_SHORT_MD         0.0

[100 rows x 2 columns]

Iteration 2 Feature Importances:

                                         Feature  Importance
0                        ANNUAL_VOLUME_CASES_2023         0.0
74         SUB_TRADE_CHANNEL_OTHER GENERAL RETAIL         0.0
72                 SUB_TRADE_CHANNEL_OTHER DINING         0.0
71          SUB_TRADE_CHANNEL_OTHER ACCOMMODATION         0.0
70  SUB_TRADE_CHANNEL_OTHER ACADEMIC INSTITUTION         0.0
..                                           ...         ...
30                         TRADE_CHANNEL_GENERAL         0.0
29              TRADE_CHANNEL_FAST CASUAL DINING         0.0
28                       TRADE_CHANNEL_EDUCATION         0.0
27                         TRADE_CHANNEL_DEFENSE         0.0
98                                STATE_SHORT_MD         0.0

[99 rows x 2 columns]

Iteration 3 Feature Importances:

                                         Feature  Importance
0                       ANNUAL_VOLUME_GALLON_2023         0.0
73         SUB_TRADE_CHANNEL_OTHER GENERAL RETAIL         0.0
71                 SUB_TRADE_CHANNEL_OTHER DINING         0.0
70          SUB_TRADE_CHANNEL_OTHER ACCOMMODATION         0.0
```

```
69  SUB_TRADE_CHANNEL_OTHER ACADEMIC INSTITUTION         0.0
..                                           ...         ...
30                 TRADE_CHANNEL_GENERAL RETAILER         0.0
29                          TRADE_CHANNEL_GENERAL         0.0
28               TRADE_CHANNEL_FAST CASUAL DINING         0.0
27                        TRADE_CHANNEL_EDUCATION         0.0
97                                 STATE_SHORT_MD         0.0

[98 rows x 2 columns]

Iteration 4 Feature Importances:

                                  Feature  Importance
0                      ANNUAL_VOLUME_2023         0.0
49     SUB_TRADE_CHANNEL_BURGER FAST FOOD         0.0
71      SUB_TRADE_CHANNEL_OTHER FAST FOOD         0.0
70         SUB_TRADE_CHANNEL_OTHER DINING         0.0
69  SUB_TRADE_CHANNEL_OTHER ACCOMMODATION         0.0
..                                    ...         ...
30    TRADE_CHANNEL_GOURMET FOOD RETAILER         0.0
29         TRADE_CHANNEL_GENERAL RETAILER         0.0
28                  TRADE_CHANNEL_GENERAL         0.0
27       TRADE_CHANNEL_FAST CASUAL DINING         0.0
96                         STATE_SHORT_MD         0.0

[97 rows x 2 columns]

Iteration 5 Feature Importances:

                                  Feature  Importance
0                   AVG_ORDER_VOLUME_2023         0.0
1                     LOCAL_MARKET_PARTNER         0.0
70      SUB_TRADE_CHANNEL_OTHER FAST FOOD         0.0
69         SUB_TRADE_CHANNEL_OTHER DINING         0.0
68  SUB_TRADE_CHANNEL_OTHER ACCOMMODATION         0.0
..                                    ...         ...
29    TRADE_CHANNEL_GOURMET FOOD RETAILER         0.0
28         TRADE_CHANNEL_GENERAL RETAILER         0.0
27                  TRADE_CHANNEL_GENERAL         0.0
26       TRADE_CHANNEL_FAST CASUAL DINING         0.0
95                         STATE_SHORT_MD         0.0

[96 rows x 2 columns]

Iteration 6 Feature Importances:

                                  Feature  Importance
0                    LOCAL_MARKET_PARTNER         0.0
60    SUB_TRADE_CHANNEL_MEXICAN FAST FOOD         0.0
69      SUB_TRADE_CHANNEL_OTHER FAST FOOD         0.0
68         SUB_TRADE_CHANNEL_OTHER DINING         0.0
67  SUB_TRADE_CHANNEL_OTHER ACCOMMODATION         0.0
..                                    ...         ...
29               TRADE_CHANNEL_HEALTHCARE         0.0
28    TRADE_CHANNEL_GOURMET FOOD RETAILER         0.0
27         TRADE_CHANNEL_GENERAL RETAILER         0.0
```

```
26                   TRADE_CHANNEL_GENERAL         0.0
94                           STATE_SHORT_MD         0.0

[95 rows x 2 columns]

Iteration 7 Feature Importances:

                                    Feature  Importance
0                              CO2_CUSTOMER         0.0
59    SUB_TRADE_CHANNEL_MEXICAN FAST FOOD         0.0
68      SUB_TRADE_CHANNEL_OTHER FAST FOOD         0.0
67         SUB_TRADE_CHANNEL_OTHER DINING         0.0
66  SUB_TRADE_CHANNEL_OTHER ACCOMMODATION         0.0
..                                      ...         ...
29               TRADE_CHANNEL_INDUSTRIAL         0.0
28               TRADE_CHANNEL_HEALTHCARE         0.0
27    TRADE_CHANNEL_GOURMET FOOD RETAILER         0.0
26         TRADE_CHANNEL_GENERAL RETAILER         0.0
93                           STATE_SHORT_MD         0.0

[94 rows x 2 columns]

Iteration 8 Feature Importances:

                                    Feature  Importance
0                  DELIVERY_COST_2023_CASES         0.0
59         SUB_TRADE_CHANNEL_MIDDLE SCHOOL         0.0
68  SUB_TRADE_CHANNEL_OTHER GENERAL RETAIL         0.0
67       SUB_TRADE_CHANNEL_OTHER FAST FOOD         0.0
66          SUB_TRADE_CHANNEL_OTHER DINING         0.0
..                                      ...         ...
29      TRADE_CHANNEL_LARGE-SCALE RETAILER         0.0
28                TRADE_CHANNEL_INDUSTRIAL         0.0
27                TRADE_CHANNEL_HEALTHCARE         0.0
26     TRADE_CHANNEL_GOURMET FOOD RETAILER         0.0
92                           STATE_SHORT_MD         0.0

[93 rows x 2 columns]

Iteration 9 Feature Importances:

                                    Feature  Importance
0                 DELIVERY_COST_2023_GALLON         0.0
58         SUB_TRADE_CHANNEL_MIDDLE SCHOOL         0.0
67  SUB_TRADE_CHANNEL_OTHER GENERAL RETAIL         0.0
66       SUB_TRADE_CHANNEL_OTHER FAST FOOD         0.0
65          SUB_TRADE_CHANNEL_OTHER DINING         0.0
..                                      ...         ...
28      TRADE_CHANNEL_LARGE-SCALE RETAILER         0.0
27                TRADE_CHANNEL_INDUSTRIAL         0.0
26                TRADE_CHANNEL_HEALTHCARE         0.0
25     TRADE_CHANNEL_GOURMET FOOD RETAILER         0.0
91                           STATE_SHORT_MD         0.0

[92 rows x 2 columns]
```

```
Iteration 10 Feature Importances:

                                   Feature  Importance
0                       DELIVERY_COST_2023         0.0
68    SUB_TRADE_CHANNEL_OTHER GOURMET FOOD         0.0
66  SUB_TRADE_CHANNEL_OTHER GENERAL RETAIL         0.0
65       SUB_TRADE_CHANNEL_OTHER FAST FOOD         0.0
64          SUB_TRADE_CHANNEL_OTHER DINING         0.0
..                                     ...         ...
28      TRADE_CHANNEL_LICENSED HOSPITALITY         0.0
27      TRADE_CHANNEL_LARGE-SCALE RETAILER         0.0
26                TRADE_CHANNEL_INDUSTRIAL         0.0
25                TRADE_CHANNEL_HEALTHCARE         0.0
90                          STATE_SHORT_MD         0.0

[91 rows x 2 columns]

Iteration 1 Metrics:

Train ROC-AUC: 0.5000
Test ROC-AUC: 0.5000
Train Accuracy: 0.9672
Test Accuracy: 0.9672
Train F1 Score: 0.0000
Test F1 Score: 0.0000
Train Precision: 0.0000
Test Precision: 0.0000
Train Recall: 0.0000
Test Recall: 0.0000

Iteration 2 Metrics:

Train ROC-AUC: 0.5000
Test ROC-AUC: 0.5000
Train Accuracy: 0.9672
Test Accuracy: 0.9672
Train F1 Score: 0.0000
Test F1 Score: 0.0000
Train Precision: 0.0000
Test Precision: 0.0000
Train Recall: 0.0000
Test Recall: 0.0000

Iteration 3 Metrics:

Train ROC-AUC: 0.5000
Test ROC-AUC: 0.5000
Train Accuracy: 0.9672
Test Accuracy: 0.9672
Train F1 Score: 0.0000
Test F1 Score: 0.0000
Train Precision: 0.0000
Test Precision: 0.0000
Train Recall: 0.0000
Test Recall: 0.0000
```

```
Iteration 4 Metrics:

Train ROC-AUC: 0.5000
Test ROC-AUC: 0.5000
Train Accuracy: 0.9672
Test Accuracy: 0.9672
Train F1 Score: 0.0000
Test F1 Score: 0.0000
Train Precision: 0.0000
Test Precision: 0.0000
Train Recall: 0.0000
Test Recall: 0.0000

Iteration 5 Metrics:

Train ROC-AUC: 0.5000
Test ROC-AUC: 0.5000
Train Accuracy: 0.9672
Test Accuracy: 0.9672
Train F1 Score: 0.0000
Test F1 Score: 0.0000
Train Precision: 0.0000
Test Precision: 0.0000
Train Recall: 0.0000
Test Recall: 0.0000

Iteration 6 Metrics:

Train ROC-AUC: 0.5000
Test ROC-AUC: 0.5000
Train Accuracy: 0.9672
Test Accuracy: 0.9672
Train F1 Score: 0.0000
Test F1 Score: 0.0000
Train Precision: 0.0000
Test Precision: 0.0000
Train Recall: 0.0000
Test Recall: 0.0000

Iteration 7 Metrics:

Train ROC-AUC: 0.5000
Test ROC-AUC: 0.5000
Train Accuracy: 0.9672
Test Accuracy: 0.9672
Train F1 Score: 0.0000
Test F1 Score: 0.0000
Train Precision: 0.0000
Test Precision: 0.0000
Train Recall: 0.0000
Test Recall: 0.0000

Iteration 8 Metrics:

Train ROC-AUC: 0.5000
Test ROC-AUC: 0.5000
```

```
Train Accuracy: 0.9672
Test Accuracy: 0.9672
Train F1 Score: 0.0000
Test F1 Score: 0.0000
Train Precision: 0.0000
Test Precision: 0.0000
Train Recall: 0.0000
Test Recall: 0.0000

Iteration 9 Metrics:

Train ROC-AUC: 0.5000
Test ROC-AUC: 0.5000
Train Accuracy: 0.9672
Test Accuracy: 0.9672
Train F1 Score: 0.0000
Test F1 Score: 0.0000
Train Precision: 0.0000
Test Precision: 0.0000
Train Recall: 0.0000
Test Recall: 0.0000

Iteration 10 Metrics:

Train ROC-AUC: 0.5000
Test ROC-AUC: 0.5000
Train Accuracy: 0.9672
Test Accuracy: 0.9672
Train F1 Score: 0.0000
Test F1 Score: 0.0000
Train Precision: 0.0000
Test Precision: 0.0000
Train Recall: 0.0000
Test Recall: 0.0000
```

Undersampling also did not help. Although it made the ROC-AUC score 50% between train and test this is still not very good. This is because an ROC-AUC of 50% is no better than random guessing, even if makes the Decision Tree models more generalizable.XGB

XGBoost Model

Single Decision Trees are not very great at dealing with imbalance datasets despite techniques to handle the data imbalances. Hence, we'll instead try to utilize XGboost instead with a grid search to get the best hyperparameters. Although XGBoost is a blackbox model, it's possible to extract the top predictors that contribute to the model's predictions. Quantifying the predictors in terms of volume may be a bit limited from the XGBoost model.

Grid Search and Model Training

In [ ]:
```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(
    X_dataset, target, test_size=0.2, stratify=target, random_state=42
```

```
)

# Apply SMOTE to balance the training set
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Define the XGBoost model
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric="logloss", random_st

# Define hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200],       # Number of trees
    'max_depth': [3, 5, 7],               # Tree depth
    'learning_rate': [0.01, 0.1, 0.3],    # Step size shrinkage
    'subsample': [0.8, 1.0],              # Fraction of samples used per tree
    'colsample_bytree': [0.8, 1.0],       # Fraction of features used per tree
    'gamma': [0, 1, 5]                    # Minimum loss reduction for further spli
}

# Perform GridSearch with 5-fold cross-validation
grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    scoring='roc_auc',
    cv=5,
    n_jobs=-1,  # Use all available cores
    verbose=2
)

# Fit the model on resampled training data
grid_search.fit(X_train_resampled, y_train_resampled)

# Get the best model
best_xgb = grid_search.best_estimator_

# Evaluate on the test set
y_test_pred = best_xgb.predict_proba(X_test)[:, 1]  # Get probabilities
test_roc_auc = roc_auc_score(y_test, y_test_pred)

# Print results
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Test ROC-AUC Score: {test_roc_auc:.4f}")
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
Best Parameters: {'colsample_bytree': 0.8, 'gamma': 0, 'learning_rate': 0.3, 'max_de
pth': 7, 'n_estimators': 200, 'subsample': 0.8}
Test ROC-AUC Score: 0.9002
```

Generate Predictions on Train and Test Set

The model will be used to remake predictions on the train and test sets to compare the model's performance.

In [ ]:
```
print("Best Parameters:", grid_search.best_params_)
```

```
# Get predictions
y_train_pred_proba = best_xgb.predict_proba(X_train)[:, 1]  # Probabilities
y_test_pred_proba = best_xgb.predict_proba(X_test)[:, 1]

# Compute ROC-AUC
train_roc_auc = roc_auc_score(y_train, y_train_pred_proba)
test_roc_auc = roc_auc_score(y_test, y_test_pred_proba)

print(f"Train ROC-AUC: {train_roc_auc:.4f}")
print(f"Test ROC-AUC: {test_roc_auc:.4f}")
```

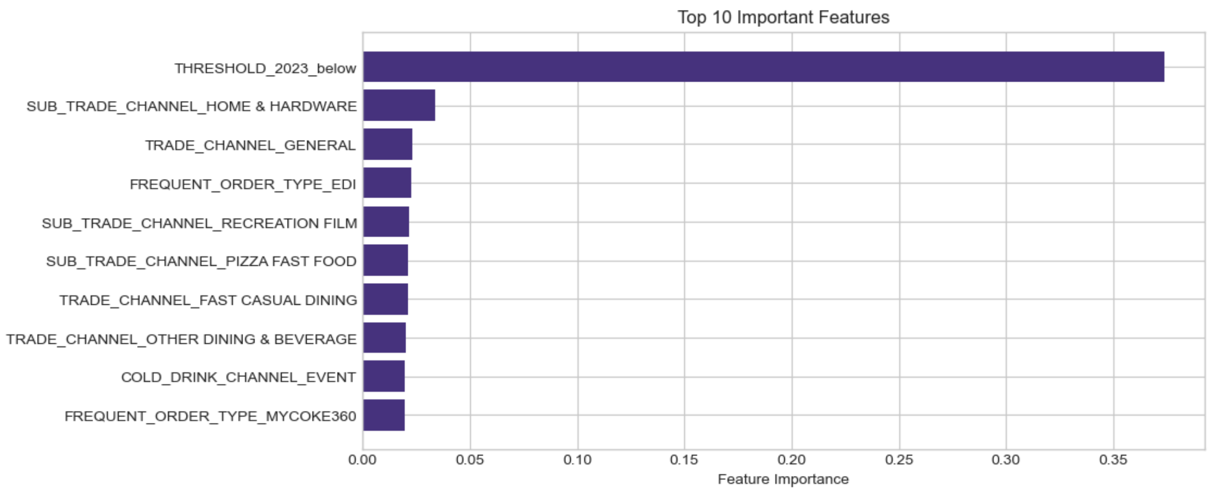
```
Best Parameters: {'colsample_bytree': 0.8, 'gamma': 0, 'learning_rate': 0.3, 'max_de
pth': 7, 'n_estimators': 200, 'subsample': 0.8}
Train ROC-AUC: 0.9997
Test ROC-AUC: 0.9002
```

The model had a drop of 10% from the Train to the Test set. This may seem steep however it's very good when compared to the other Decision Trees which had a drop of 30%. Moreover, this indicates that the predictors listed may be more reliable when compared to the Decision Tree.

Feature Importance

This code will extract the most important predictors from the Decision Tree.

```
In [ ]: # Extract feature importance
feature_importance = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': best_xgb.feature_importances_
}).sort_values(by='Importance', ascending=False)

# Plot feature importance
plt.figure(figsize=(10, 5))
plt.barh(feature_importance['Feature'][:10], feature_importance['Importance'][:10])
plt.gca().invert_yaxis()  # Flip highest importance to top
plt.xlabel("Feature Importance")
plt.title("Top 10 Important Features")
plt.show()
```



Threshold_2023 is the most important predictor with SUB_TRADE_CHANNEL_GAME_CENTER being a distant second. This again underscores the importance of VOLUME related variables for predicting transition status.

Delivery Cost Optimization

In [ ]:
```
sccu_segmentation = file_path.copy()
```

In [ ]:
```
# Load the Data
file_path = 'sccu_data.csv'
sccu_segmentation = file_path

df = pd.read_csv(sccu_segmentation)
print("Data loaded. Shape:", df.shape)
print(df.head())
```

```
Data loaded. Shape: (30755, 37)
   Unnamed: 0  CUSTOMER_NUMBER FIRST_TRANSACTION_DATE LAST_TRANSACTION_DATE  \
0           1        500245678             2023-01-09            2024-11-20
1           2        500245685             2023-01-06            2024-08-16
2           3        500245686             2023-03-07            2024-12-17
3           4        500245687             2023-02-06            2024-10-28
4           5        500245689             2023-01-13            2024-12-26

   TRANS_DAYS  TRANS_COUNT  TRANS_COUNT_2023  TRANS_COUNT_2024  \
0         682           44                27                17
1         589           63                46                17
2         652            9                 5                 4
3         631           18                 9                 9
4         714           85                40                45

   ANNUAL_VOLUME_CASES_2023  ANNUAL_VOLUME_GALLON_2023  ...  \
0                     210.0                      160.0  ...
1                      24.0                      577.5  ...
2                      17.5                        0.0  ...
3                       0.0                      125.0  ...
4                     124.0                      422.5  ...

   LOCAL_MARKET_PARTNER  CO2_CUSTOMER  ZIP_CODE  STATE_SHORT  \
0                  True          True     66508           KS
1                  True          True     21913           MD
2                  True         False      1350           MA
3                  True          True     42252           KY
4                  True         False     42031           KY

   DELIVERY_COST_2023_CASES  DELIVERY_COST_2024_CASES  \
0                938.785885                675.031755
1                206.051579                317.662851
2                128.200325                135.526058
3                  0.000000                  0.000000
4               1064.599825               1155.223773

   DELIVERY_COST_2023_GALLON  DELIVERY_COST_2024_GALLON DELIVERY_COST_2023  \
0                 428.063964                 622.030448        1366.849849
1                1163.368809                1126.805640        1369.420388
2                   0.000000                   0.000000         128.200325
3                 470.060235                 554.671077         470.060235
4                1069.832321                 921.629836        2134.432146

  DELIVERY_COST_2024
0        1297.062203
1        1444.468491
2         135.526058
3         554.671077
4        2076.853608

[5 rows x 37 columns]
```

In [ ]:

```python
# Create modeling features and target
df['Total_Ordered_Cases'] = df['ANNUAL_VOLUME_CASES_2023']
df['Order_Frequency_2023'] = df['TRANS_COUNT_2023']
df['Percentage_Drop'] = df['PERCENT_CHANGE']
```

```
df['Threshold_2023'] = df['THRESHOLD_2023'].str.lower()

# Drop rows with missing values in relevant columns
model_df = df[['Total_Ordered_Cases', 'Order_Frequency_2023', 'Percentage_Drop', 'T

print("Filtered for non-null rows. Shape:", model_df.shape)
```

```
Filtered for non-null rows. Shape: (30618, 4)
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

# Encode target variable
le = LabelEncoder()
model_df['Threshold_2023_encoded'] = le.fit_transform(model_df['Threshold_2023'])

# Define features and target
features = ['Total_Ordered_Cases', 'Order_Frequency_2023', 'Percentage_Drop']
X = model_df[features]
y = model_df['Threshold_2023_encoded']
```

In [ ]:
```
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

# Define features and target
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2

# Manual undersampling
train_df = X_train.copy()
train_df['target'] = y_train

majority = train_df[train_df['target'] == 0]
minority = train_df[train_df['target'] == 1]

# Balance the dataset by undersampling the majority class
minority_count = min(len(majority), len(minority))
majority_sampled = majority.sample(n=minority_count, random_state=42)
minority_sampled = minority.sample(n=minority_count, random_state=42)

balanced_df = pd.concat([majority_sampled, minority_sampled])

# Separate features and target from the balanced dataset
X_balanced = balanced_df[features]
y_balanced = balanced_df['target']

# Clean for infinity or large invalid values
X_balanced.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

# Drop rows with missing values (from balancing and cleaning)
X_balanced.dropna(inplace=True)
y_balanced = y_balanced.loc[X_balanced.index]

X_test.dropna(inplace=True)
y_test = y_test.loc[X_test.index]
```

```
# Final check
print("Balanced and cleaned dataset.")
print("Balanced class counts:\n", y_balanced.value_counts())
```

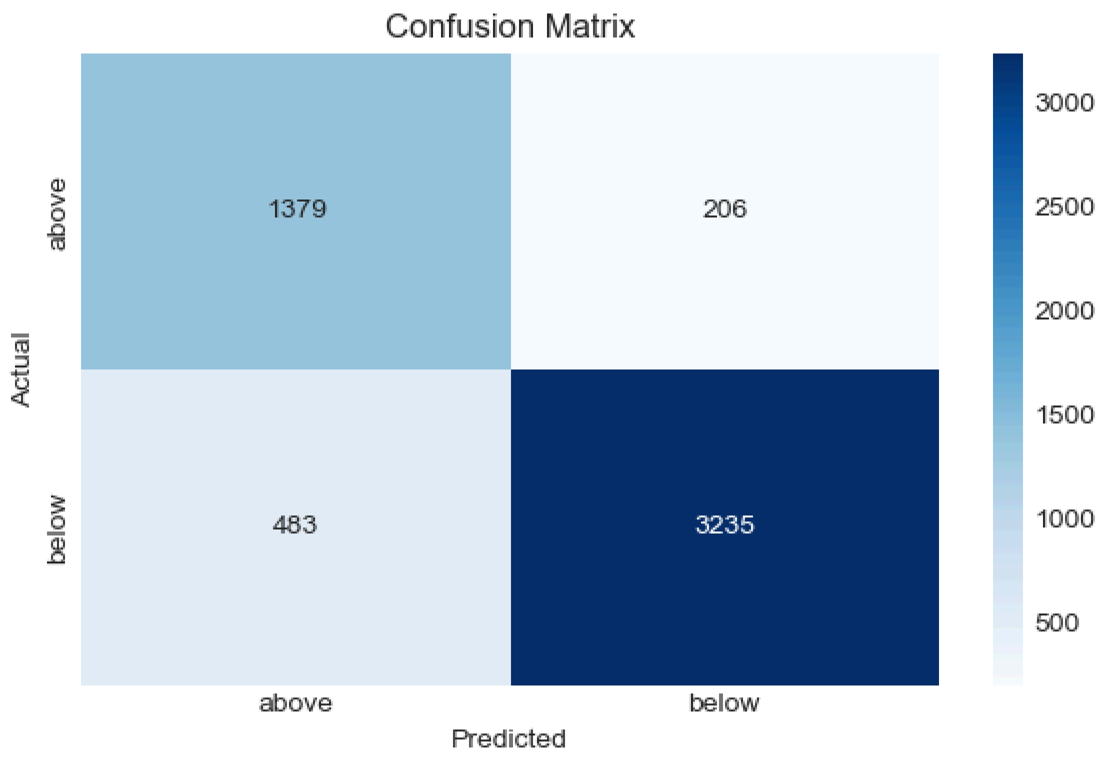
```
Balanced and cleaned dataset.
Balanced class counts:
 target
0    6338
1    5161
Name: count, dtype: int64
```

Logistic Regression Model Training

In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix


# Train Logistic Regression Model
lr = LogisticRegression(max_iter=1000)
lr.fit(X_balanced, y_balanced)
#Predict and Evaluate
y_pred = lr.predict(X_test)
report = classification_report(y_test, y_pred, target_names=le.classes_)
conf_matrix = confusion_matrix(y_test, y_pred)

print("\n📋 Classification Report:\n")
print(classification_report(y_test, y_pred, target_names=le.classes_))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

```
📋 Classification Report:

              precision    recall  f1-score   support

       above       0.74      0.87      0.80      1585
       below       0.94      0.87      0.90      3718

    accuracy                           0.87      5303
   macro avg       0.84      0.87      0.85      5303
weighted avg       0.88      0.87      0.87      5303
```



Model used is Logistic Regression where accuracy is 87%. Precision above is 0.74, below is 0.94. Recall above is 0.87 , below is 0.87. Model performs strongly on both classes, especially the majority 'below' class. Manual under sampling helped balance recall across classes.

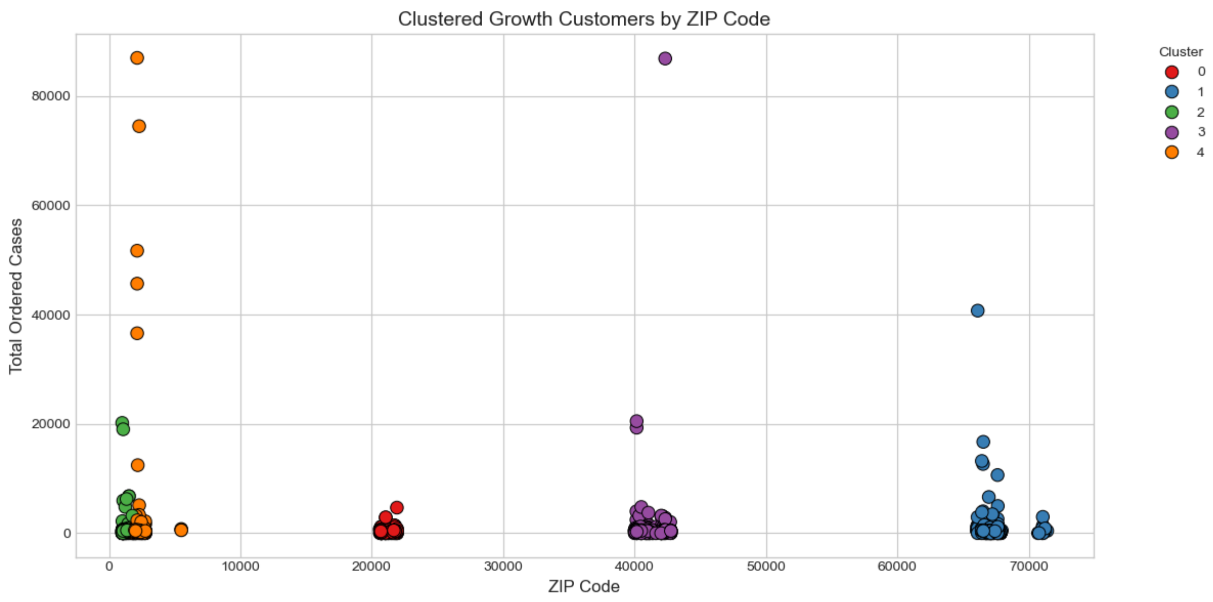
Number of Customers who Improved from Below to Above

```
In [ ]: # Count Customers Who Improved from 'below' to 'above'
        # Normalize casing for consistency
        df['THRESHOLD_2023'] = df['THRESHOLD_2023'].str.lower()
        df['THRESHOLD_2024'] = df['THRESHOLD_2024'].str.lower()

        # Identify improved customers
        improved_customers = df[
            (df['THRESHOLD_2023'] == 'below') &
            (df['THRESHOLD_2024'] == 'above')
        ]

        # Output the count
        num_improved = improved_customers.shape[0]
        print(f"\n Number of customers who improved from 'below' to 'above': {num_improved}
```

```
Number of customers who improved from 'below' to 'above': 1268
```

Cluster Improved Customers by ZIP Code and Volume

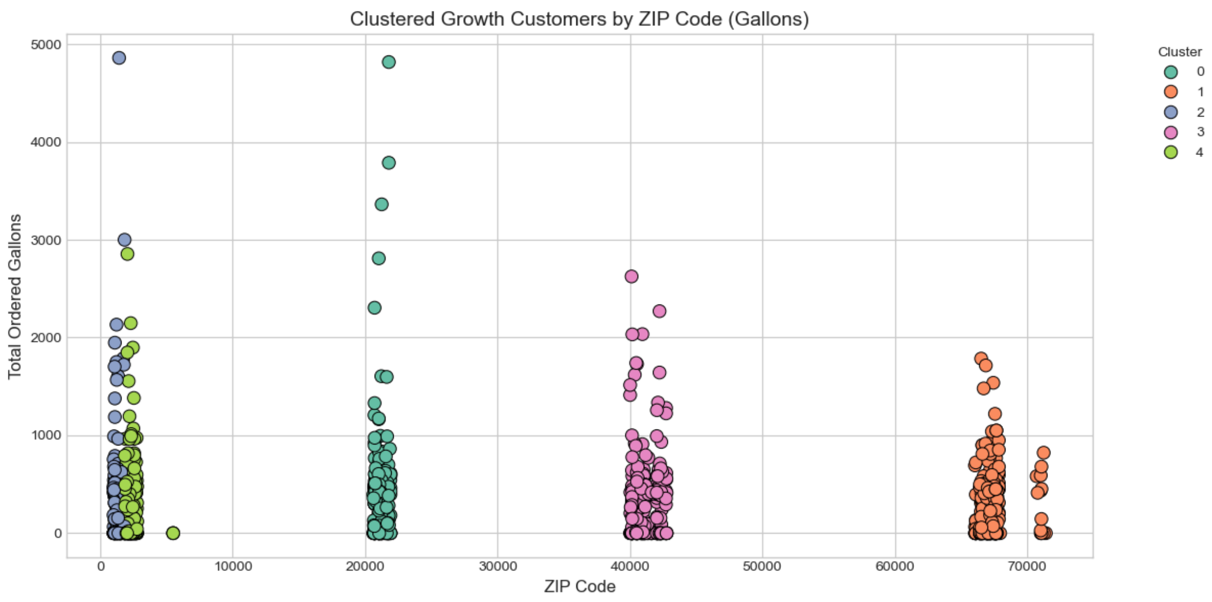
In [ ]:
```
#Cluster Improved Customers by ZIP Code and Volume

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans

# Prepare improved customers data again if needed
transitioned_customers = df[
    (df['THRESHOLD_2023'] == 'below') &
    (df['THRESHOLD_2024'] == 'above')
]

# Select ZIP and volume columns
growth_customers = transitioned_customers[["ZIP_CODE", "ANNUAL_VOLUME_CASES_2024"]]
growth_customers = growth_customers.rename(columns={"ANNUAL_VOLUME_CASES_2024": "To

# Ensure ZIP_CODE is numeric
growth_customers = growth_customers[growth_customers['ZIP_CODE'].apply(lambda x: st
growth_customers["ZIP_CODE"] = growth_customers["ZIP_CODE"].astype(int)

# Perform K-Means clustering
kmeans = KMeans(n_clusters=5, random_state=42)
growth_customers["Cluster"] = kmeans.fit_predict(growth_customers[["ZIP_CODE"]])

# Plot the clusters
plt.figure(figsize=(12, 6))
sns.scatterplot(
    data=growth_customers,
    x="ZIP_CODE",
    y="Total_Ordered_Cases",
    hue="Cluster",
    palette="Set1",
    s=80,
    edgecolor='black'
)

plt.title("Clustered Growth Customers by ZIP Code", fontsize=14)
plt.xlabel("ZIP Code", fontsize=12)
plt.ylabel("Total Ordered Cases", fontsize=12)
plt.legend(title="Cluster", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



X-axis: ZIP Code, Y-axis: Total Ordered Cases in 2024 (i.e., volume). Color: Represents the assigned Cluster (0–4) from K-Means. Each point = 1 customer who improved from below to above threshold Customers were grouped solely by ZIP Code whic h creates geographic clusters Largest-volume customers are spread across clusters which suggests volume alone isn’t geographically concentrated.

Cluster Improved Customers data using Ordered Gallon

In [ ]:
```
# Prepare improved customers data again( Used Ordered Gallons )
transitioned_customers = df[
    (df['THRESHOLD_2023'] == 'below') &
    (df['THRESHOLD_2024'] == 'above')
]

# Select ZIP and GALLON volume
growth_customers = transitioned_customers[["ZIP_CODE", "ANNUAL_VOLUME_GALLON_2024"]
growth_customers = growth_customers.rename(columns={"ANNUAL_VOLUME_GALLON_2024": "T

# Ensure ZIP_CODE is numeric
growth_customers = growth_customers[growth_customers['ZIP_CODE'].apply(lambda x: st
growth_customers["ZIP_CODE"] = growth_customers["ZIP_CODE"].astype(int)

# KMeans clustering
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5, random_state=42)
growth_customers["Cluster"] = kmeans.fit_predict(growth_customers[["ZIP_CODE"]])

# Plot
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 6))
sns.scatterplot(
    data=growth_customers,
    x="ZIP_CODE",
    y="Total_Ordered_Gallons",
    hue="Cluster",
    palette="Set2",
    s=80,
    edgecolor='black'
)
plt.title("Clustered Growth Customers by ZIP Code (Gallons)", fontsize=14)
plt.xlabel("ZIP Code", fontsize=12)
plt.ylabel("Total Ordered Gallons", fontsize=12)
plt.legend(title="Cluster", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```

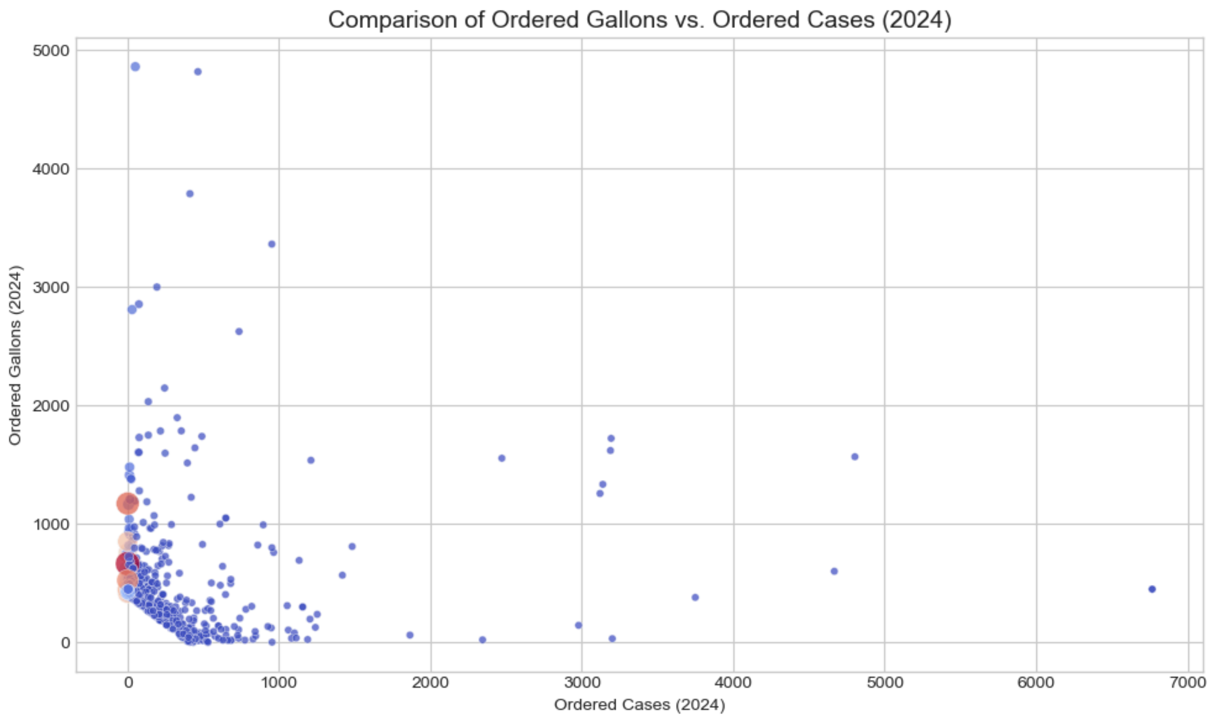


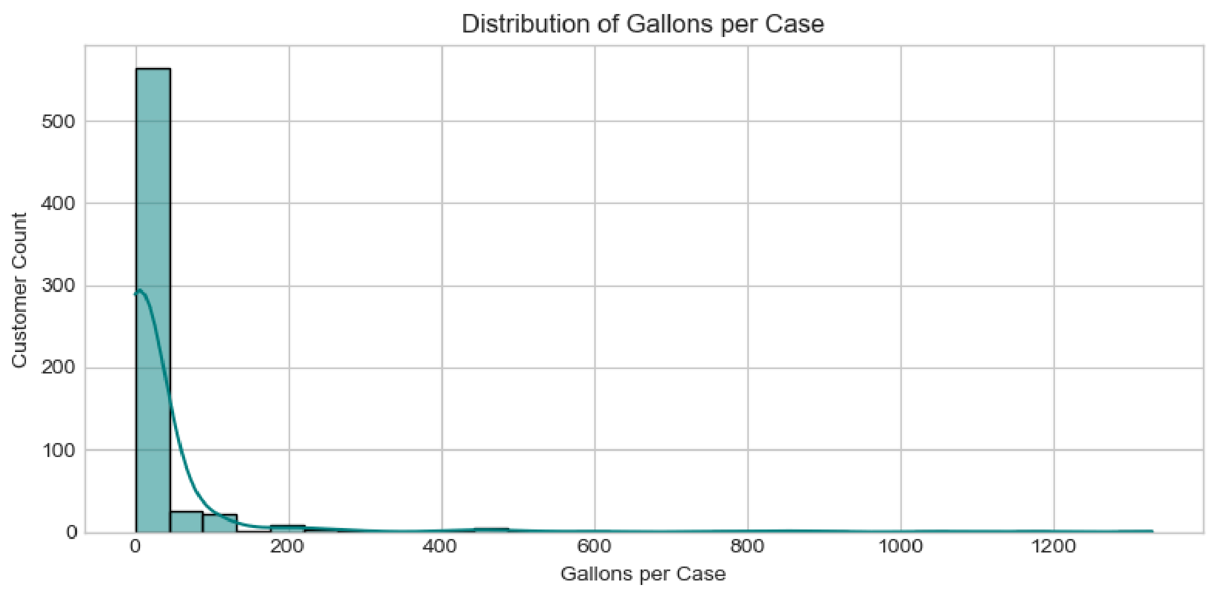
This gallon-based view might show new patterns not obvious from case count alone — e.g., ZIPs with small case counts but high gallon volume, possibly due to large-batch liquid orders.

Ordered cases and Gallons Comparison

```
In [ ]: # Comaprision between ordered cases and gallons

        # Step 1: Filter improved customers with valid data
        volume_df = transitioned_customers[
            (transitioned_customers["ANNUAL_VOLUME_CASES_2024"] > 0) &
            (transitioned_customers["ANNUAL_VOLUME_GALLON_2024"] > 0)
        ].copy()

        # Step 2: Create comparison column
        volume_df["Gallons_Per_Case"] = volume_df["ANNUAL_VOLUME_GALLON_2024"] / volume_df[

        # Step 3: Plot scatter of Gallons vs. Cases
        import matplotlib.pyplot as plt
```

```python
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=volume_df,
    x="ANNUAL_VOLUME_CASES_2024",
    y="ANNUAL_VOLUME_GALLON_2024",
    hue="Gallons_Per_Case",
    palette="coolwarm",
    size="Gallons_Per_Case",
    sizes=(20, 200),
    alpha=0.7,
    legend=False
)
plt.title("Comparison of Ordered Gallons vs. Ordered Cases (2024)", fontsize=14)
plt.xlabel("Ordered Cases (2024)")
plt.ylabel("Ordered Gallons (2024)")
plt.grid(True)
plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 4))
sns.histplot(volume_df["Gallons_Per_Case"], bins=30, kde=True, color='teal')
plt.title("Distribution of Gallons per Case")
plt.xlabel("Gallons per Case")
plt.ylabel("Customer Count")
plt.grid(True)
plt.tight_layout()
plt.show()
```



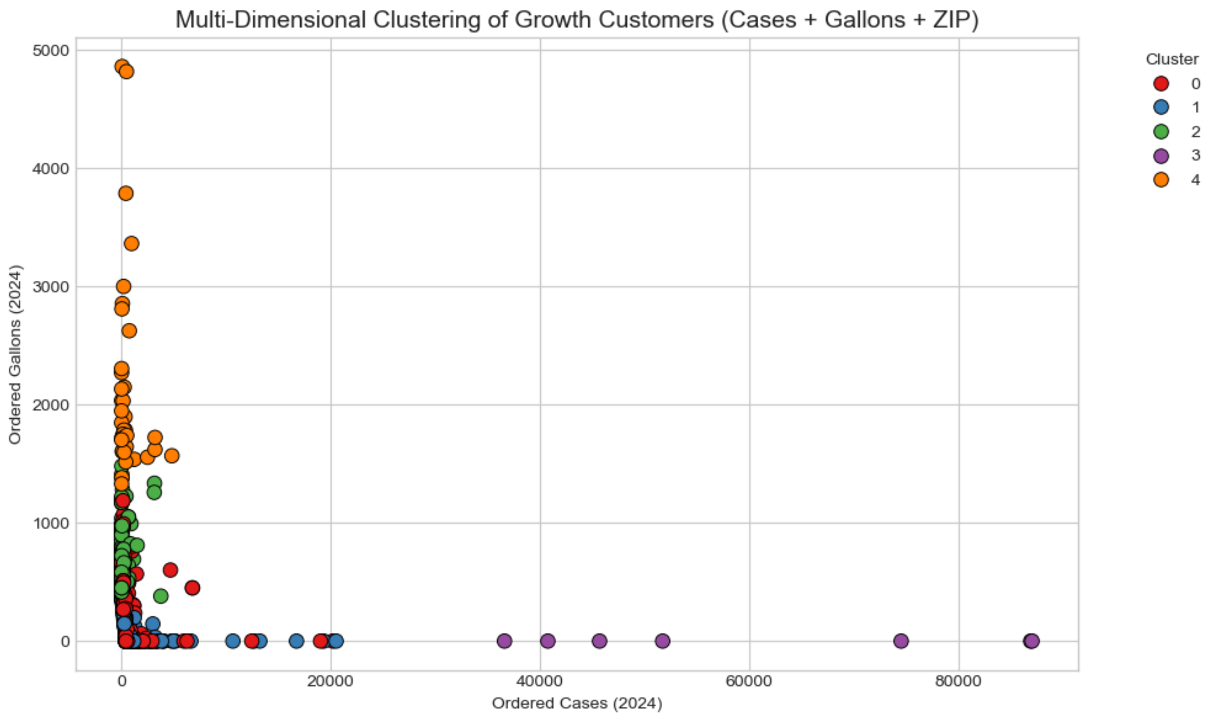


X: Ordered Cases (2024), Y: Ordered Gallons(2024)Dot Size & Color: Represent Gallons per Case(Higher = larger, redder dot).Most customers with low case counts also have low gallon volume — typical of small accounts.A diagonal trend exists for medium/large customers: more cases generally means more gallons. Outliers identified for High gallons but few case and High cases but low gallons

Multi-Dimensional Clustering

In [ ]:
```
#Multi-Dimensional Clustering (ZIP + Cases + Gallons)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Filter growth customers with valid data
transitioned_customers = df[
    (df['THRESHOLD_2023'].str.lower() == 'below') &
    (df['THRESHOLD_2024'].str.lower() == 'above')
]

cluster_df = transitioned_customers[
    ['ZIP_CODE', 'ANNUAL_VOLUME_CASES_2024', 'ANNUAL_VOLUME_GALLON_2024']
].dropna()

#  Clean & transform ZIP (ensure it's numeric)
cluster_df = cluster_df[cluster_df['ZIP_CODE'].apply(lambda x: str(x).isdigit())]
cluster_df['ZIP_CODE'] = cluster_df['ZIP_CODE'].astype(int)

#  Scale the features (KMeans is distance-based)
features = ['ZIP_CODE', 'ANNUAL_VOLUME_CASES_2024', 'ANNUAL_VOLUME_GALLON_2024']
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(cluster_df[features])

# Run KMeans clustering
kmeans = KMeans(n_clusters=5, random_state=42)
cluster_df['Cluster'] = kmeans.fit_predict(X_scaled)

# Visualize clusters – 2D projection (Cases vs Gallons colored by cluster)
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=cluster_df,
    x='ANNUAL_VOLUME_CASES_2024',
    y='ANNUAL_VOLUME_GALLON_2024',
    hue='Cluster',
    palette='Set1',
    s=70,
    edgecolor='black'
)
plt.title("Multi-Dimensional Clustering of Growth Customers (Cases + Gallons + ZIP)
plt.xlabel("Ordered Cases (2024)")
plt.ylabel("Ordered Gallons (2024)")
plt.legend(title="Cluster", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



Instead of clustering by a single feature, this approach uses ZIP code + case + gallon volume to create real-world customer segments based on both location and behavior.

Average Delivery Cost Per Case

```
In [ ]: import pandas as pd
        import numpy as np
```

```
# Step 1: Filter valid data where both delivery cost and volume are positive
valid_cost_data = transitioned_customers[
    (transitioned_customers["DELIVERY_COST_2024"] > 0) &
    (transitioned_customers["ANNUAL_VOLUME_CASES_2024"] > 0)
].copy()

# Step 2: Calculate cost per case for each customer
valid_cost_data["Cost_Per_Case"] = valid_cost_data["DELIVERY_COST_2024"] / valid_co

# Step 3: Take the average of all valid cost-per-case values
cost_per_case = valid_cost_data["Cost_Per_Case"].mean()

# Step 4: Output result
print(f"Actual Average Delivery Cost per Case (2024): ${cost_per_case:.2f}")
```

```
Actual Average Delivery Cost per Case (2024): $40.92
```

Actual average delivery cost per case (from 2024 data) is $40.92. I Multiplied by total cases in each cluster to estimate total delivery cost per cluster.

Before Vs After Clustering -Comparing Cost per Delivery

In [ ]:

```
#Compare Cost Per Delivery: Before vs. After Clustering

#  Calculate Average Cost Per Case BEFORE Clustering
# Filter improved customers with valid delivery cost and order volume
improved_customers = transitioned_customers[
    (transitioned_customers["DELIVERY_COST_2024"] > 0) &
    (transitioned_customers["ANNUAL_VOLUME_CASES_2024"] > 0)
].copy()  # use .copy() to avoid SettingWithCopyWarning

# Calculate cost per case for each improved customer
improved_customers["Cost_Per_Case"] = improved_customers["DELIVERY_COST_2024"] / im

# Compute the average cost per case before clustering
avg_cost_before = improved_customers["Cost_Per_Case"].mean()
print(f"Actual Average Cost Per Case BEFORE Clustering: ${avg_cost_before:.2f}")

# Calculate Average Cost Per Case AFTER Clustering (Validated)
# Merge the growth_customers (from clustering, Step 14) with actual cost data from
merged = growth_customers.merge(
    improved_customers[["ZIP_CODE", "DELIVERY_COST_2024", "ANNUAL_VOLUME_CASES_2024
    on="ZIP_CODE",
    how="left"
).dropna()

# Group by cluster and sum up the delivery cost and total cases
cluster_actuals = merged.groupby("Cluster").agg({
    "DELIVERY_COST_2024": "sum",
    "ANNUAL_VOLUME_CASES_2024": "sum"
}).reset_index()
```

```
# Compute average cost per case for each cluster and overall (weighted)
cluster_actuals["Cost_Per_Case"] = cluster_actuals["DELIVERY_COST_2024"] / cluster_

total_cost = cluster_actuals["DELIVERY_COST_2024"].sum()
total_cases = cluster_actuals["ANNUAL_VOLUME_CASES_2024"].sum()
validated_avg_cost_after = total_cost / total_cases

print(f" Validated Average Cost Per Case AFTER Clustering: ${validated_avg_cost_aft

# Calculate estimated savings per case due to clustering
savings = avg_cost_before - validated_avg_cost_after
print(f"Estimated Savings Per Case from Routing Clusters: ${savings:.2f}")

# Optional: Display Cluster-Level Details
print("\nCluster-Level Actuals:")
print(cluster_actuals[['Cluster', 'DELIVERY_COST_2024', 'ANNUAL_VOLUME_CASES_2024',
```

```
Actual Average Cost Per Case BEFORE Clustering: $40.92
 Validated Average Cost Per Case AFTER Clustering: $2.19
Estimated Savings Per Case from Routing Clusters: $38.73

Cluster-Level Actuals:
   Cluster  DELIVERY_COST_2024  ANNUAL_VOLUME_CASES_2024  Cost_Per_Case
0        0        1.372571e+06              3.311752e+05       4.144548
1        1        2.304221e+06              1.027345e+06       2.242891
2        2        1.220535e+06              3.616110e+05       3.375270
3        3        2.396896e+06              1.007614e+06       2.378786
4        4        3.472352e+06              2.180573e+06       1.592404
```

.total_cost = sum of DELIVERY_COST_2024 across all clustered customers .total_cases = sum of ANNUAL_VOLUME_CASES_2024 across those customers . Avg. Cost per Case = total_cost / total_cases = $2.19. Indicates major efficiency gains from optimizing customer grouping and routing.

Percentage Cost Savings

In [ ]:

```
# Show Cost Savings as a Percentage

savings_per_case = avg_cost_before - validated_avg_cost_after
percentage_savings = (savings_per_case / avg_cost_before) * 100

# Print full comparison with percentage
print(f"Actual Avg. Cost per Case BEFORE Clustering:  ${avg_cost_before:.2f}")
print(f" Avg. Cost per Case AFTER Clustering:         ${validated_avg_cost_after:.2
print(f" Estimated Savings per Case:                 ${savings_per_case:.2f}")
print(f" Percentage Cost Reduction:                  {percentage_savings:.2f}%")
```

```
Actual Avg. Cost per Case BEFORE Clustering:  $40.92
 Avg. Cost per Case AFTER Clustering:         $2.19
 Estimated Savings per Case:                 $38.73
 Percentage Cost Reduction:                  94.64%
```

After clustering, it dropped to $2.19, saving$38.73 per case — a 94.64% cost reduction driven by more efficient delivery through optimized customer grouping.

Average Cost per Gallon Before Clustering

In [ ]:
```
# Filter improved customers with valid data( for gallons)
valid_gallon_data = transitioned_customers[
    (transitioned_customers["DELIVERY_COST_2024"] > 0) &
    (transitioned_customers["ANNUAL_VOLUME_GALLON_2024"] > 0)
].copy()

# Calculate cost per gallon
valid_gallon_data["Cost_Per_Gallon"] = valid_gallon_data["DELIVERY_COST_2024"] / va
avg_cost_per_gallon_before = valid_gallon_data["Cost_Per_Gallon"].mean()

print(f"Actual Average Cost per Gallon BEFORE Clustering: ${avg_cost_per_gallon_bef
```

```
Actual Average Cost per Gallon BEFORE Clustering: $18.97
```

Calculates actual average delivery cost per gallon using 2024 data from improved customers.This means, on average, it cost $18.97 to deliver each gallon before any clustering or delivery optimization.

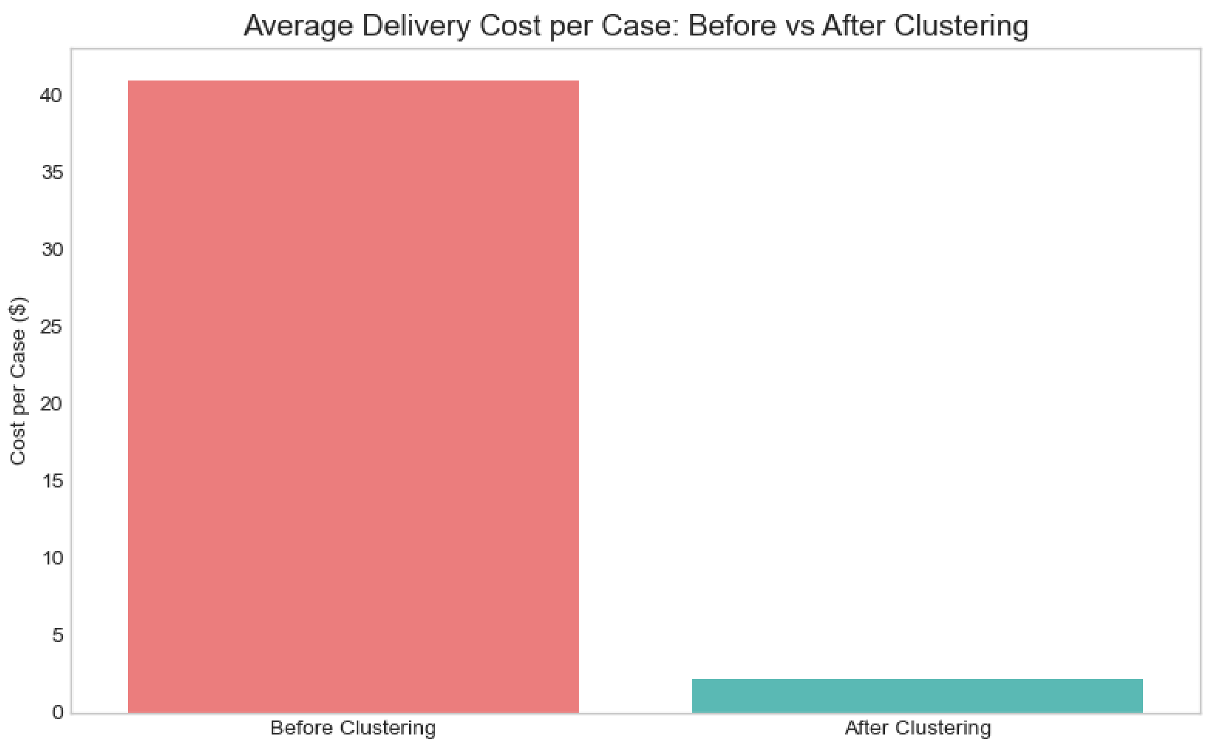
Average cost per Gallon After Clustering

In [ ]:
```
# Merge cost/gallon data into gallon-based clustered customers
gallon_clustered = growth_customers.merge(
    valid_gallon_data[["ZIP_CODE", "DELIVERY_COST_2024", "ANNUAL_VOLUME_GALLON_2024
    on="ZIP_CODE", how="left"
).dropna()

cluster_gallon_costs = gallon_clustered.groupby("Cluster").agg({
    "DELIVERY_COST_2024": "sum",
    "ANNUAL_VOLUME_GALLON_2024": "sum"
}).reset_index()

cluster_gallon_costs["Cost_Per_Gallon"] = cluster_gallon_costs["DELIVERY_COST_2024"

# Weighted average (after clustering)
total_cost_gallons = cluster_gallon_costs["DELIVERY_COST_2024"].sum()
total_gallons = cluster_gallon_costs["ANNUAL_VOLUME_GALLON_2024"].sum()
avg_cost_per_gallon_after = total_cost_gallons / total_gallons

print(f"Avg. Cost per Gallon AFTER Clustering: ${avg_cost_per_gallon_after:.2f}")

# Calculate savings
gallon_savings = avg_cost_per_gallon_before - avg_cost_per_gallon_after
percent_gallon_savings = (gallon_savings / avg_cost_per_gallon_before) * 100

print(f"Savings per Gallon: ${gallon_savings:.2f}")
print(f"Percentage Savings from Clustering (Gallons): {percent_gallon_savings:.2f}%
Avg. Cost per Gallon AFTER Clustering: $3.61
Savings per Gallon: $15.36
Percentage Savings from Clustering (Gallons): 80.95%
```

Avg. Cost per Gallon Before Clustering: $18.97. After Clustering$:3.61, Savings per Gallon: $15.36 (81% reduction).Indicates significant efficiency gains in liquid deliveries through optimized routing and grouping.

Visuals for Average Cost per Case Before vs After Clustering

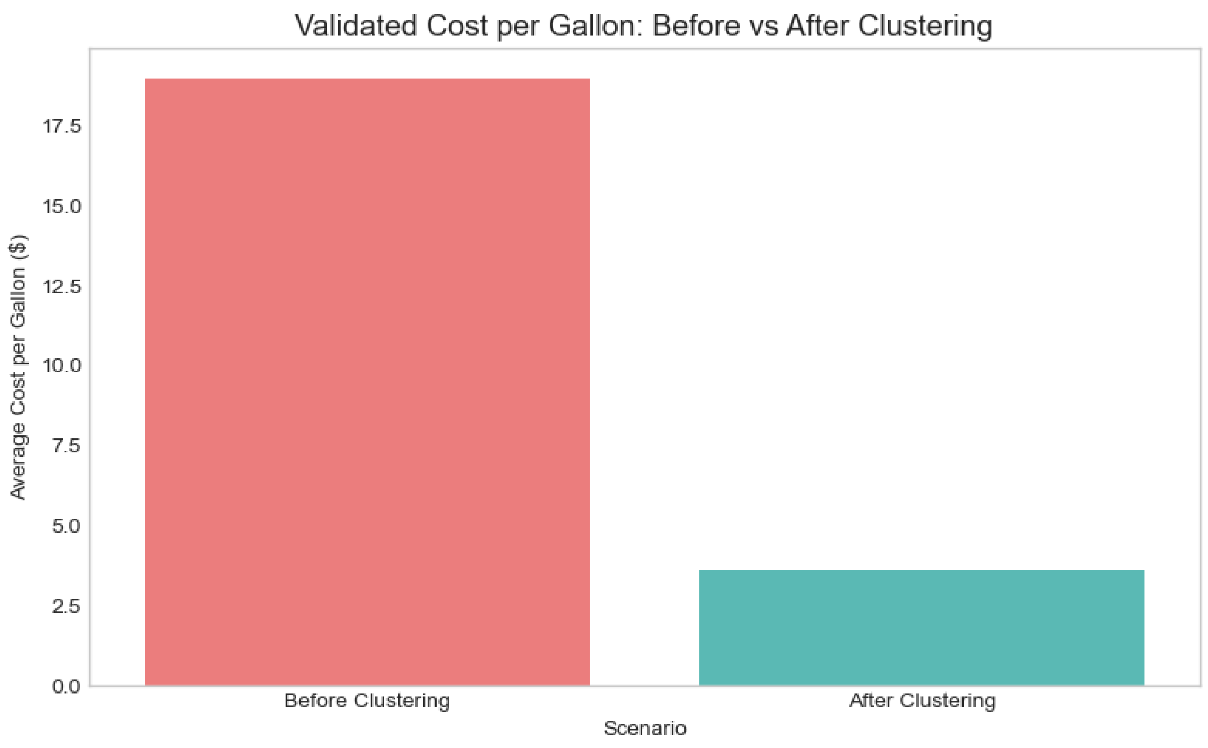
```
In [ ]: # Visualize Average Cost Per Case Before vs After Clustering

import matplotlib.pyplot as plt
import seaborn as sns

# Bar chart: before vs after clustering
plt.figure(figsize=(8, 5))
sns.barplot(
    x=["Before Clustering", "After Clustering"],
    y=[avg_cost_before, validated_avg_cost_after],
    palette=["#FF6B6B", "#4ECDC4"]
)
plt.title("Average Delivery Cost per Case: Before vs After Clustering", fontsize=14
plt.ylabel("Cost per Case ($)")
plt.grid(axis="y")
plt.tight_layout()
plt.show()

# Robustness Check – Distribution of Cost per Case

# Before clustering
before_dist = improved_customers[["ZIP_CODE", "Cost_Per_Case"]].copy()
before_dist["Scenario"] = "Before Clustering"

# After clustering
after_dist = merged[["ZIP_CODE", "DELIVERY_COST_2024", "ANNUAL_VOLUME_CASES_2024"]]
after_dist["Cost_Per_Case"] = after_dist["DELIVERY_COST_2024"] / after_dist["ANNUAL
after_dist["Scenario"] = "After Clustering"

# Combine for distribution comparison
distribution_df = pd.concat([
    before_dist[["ZIP_CODE", "Cost_Per_Case", "Scenario"]],
    after_dist[["ZIP_CODE", "Cost_Per_Case", "Scenario"]]
])

# Remove extreme outliers to keep boxplot readable
distribution_df = distribution_df[distribution_df["Cost_Per_Case"] < 150]
```



Visuals for cost per Gallon Before vs After Clustering

In [ ]:
```
# Ensure variables are consistent and re-run calculation to double-check
# Filter improved customers with valid delivery cost and gallon volume
valid_gallon_data = transitioned_customers[
    (transitioned_customers["DELIVERY_COST_2024"] > 0) &
    (transitioned_customers["ANNUAL_VOLUME_GALLON_2024"] > 0)
].copy()

# Calculate cost per gallon before clustering
valid_gallon_data["Cost_Per_Gallon"] = valid_gallon_data["DELIVERY_COST_2024"] / va
avg_cost_per_gallon_before = valid_gallon_data["Cost_Per_Gallon"].mean()

# Merge cluster assignments into valid_gallon_data
gallon_clustered = growth_customers.merge(
    valid_gallon_data[["ZIP_CODE", "DELIVERY_COST_2024", "ANNUAL_VOLUME_GALLON_2024
    on="ZIP_CODE", how="left"
).dropna()

# Group by cluster to calculate cost per gallon after clustering
cluster_gallon_costs = gallon_clustered.groupby("Cluster").agg({
    "DELIVERY_COST_2024": "sum",
    "ANNUAL_VOLUME_GALLON_2024": "sum"
}).reset_index()

# Calculate weighted average after clustering
total_cost_gallons = cluster_gallon_costs["DELIVERY_COST_2024"].sum()
total_gallons = cluster_gallon_costs["ANNUAL_VOLUME_GALLON_2024"].sum()
avg_cost_per_gallon_after = total_cost_gallons / total_gallons
```

```
# Bar plot: Before vs After clustering
import matplotlib.pyplot as plt
import seaborn as sns

gallon_cost_comparison = pd.DataFrame({
    "Scenario": ["Before Clustering", "After Clustering"],
    "Cost_Per_Gallon": [avg_cost_per_gallon_before, avg_cost_per_gallon_after]
})

plt.figure(figsize=(8, 5))
sns.barplot(data=gallon_cost_comparison, x="Scenario", y="Cost_Per_Gallon", palette
plt.title("Validated Cost per Gallon: Before vs After Clustering", fontsize=14)
plt.ylabel("Average Cost per Gallon ($)")
plt.grid(axis="y")
plt.tight_layout()
plt.show()
```



Annual Delivery Cost Statistics

In [ ]:
```
# Read the CSV file
final_df = pd.read_csv(file_path)

# Min and Max for 2023 and 2024
min_cost_2023 = float(final_df['DELIVERY_COST_2023'].min())  # Convert to float
max_cost_2023 = float(final_df['DELIVERY_COST_2023'].max())

min_cost_2024 = float(final_df['DELIVERY_COST_2024'].min())
max_cost_2024 = float(final_df['DELIVERY_COST_2024'].max())

# Average delivery cost per customer
avg_cost_2023 = float(final_df['DELIVERY_COST_2023'].mean())
avg_cost_2024 = float(final_df['DELIVERY_COST_2024'].mean())
```

```
# # Display results
print("Delivery Cost Summary:")
print(f"2023 → Min: ${min_cost_2023:.2f}, Max: ${max_cost_2023:.2f}, Avg: ${avg_cos
print(f"2024 → Min: ${min_cost_2024:.2f}, Max: ${max_cost_2024:.2f}, Avg: ${avg_cos
```

```
Delivery Cost Summary:
2023 → Min: $0.00, Max: $67412.32, Avg: $1117.69
2024 → Min: $0.00, Max: $94393.82, Avg: $1173.53
```

The minimum delivery cost per customer in both 2023 and 2024 was USD 0, indicating some customers incurred no delivery charges. The maximum delivery cost increased from USD 67,412.32 in 2023 to USD 94,393.82 in 2024, reflecting a USD 26,981.50 rise. On average, customers experienced a modest increase in delivery cost, from 1,117.69 in 2023 to 1173.53 USD in 2024, resulting in an average difference of about $56 in the costs between 2023 and 2024.

Modelling: Predicting potential growth-ready customers

Prepare the Data

```
In [ ]: # Get the Target Variable and variables to consider to model

# Create T.V1: growth-ready
final_df['GROWTH_READY'] = ((final_df['THRESHOLD_2023'] == 'below') & (final_df['TH

# Step 1: Strip whitespace from column names
final_df.columns = final_df.columns.str.strip()

# Step 2: Drop unwanted columns
drop_cols = ['Unnamed: 0','CUSTOMER_NUMBER', 'FIRST_TRANSACTION_DATE', 'LAST_TRANSA
             'FIRST_DELIVERY_DATE','ON_BOARDING_DATE','THRESHOLD_2023', 'THRESHOLD_
             'ANNUAL_VOLUME_CASES_2024', 'ANNUAL_VOLUME_GALLON_2024',
             'ANNUAL_VOLUME_2024','AVG_ORDER_VOLUME_2024','PRIMARY_GROUP_NUMBER',
             'CHANGED_VOLUME', 'PERCENT_CHANGE','DELIVERY_COST_2024_CASE','DELIVERY
             'DELIVERY_COST_2024_GALLON','DELIVERY_COST_2024', 'TRANS_COUNT','TRANS
             'TRANS_COUNT_2024']

df_model = final_df.drop(columns=drop_cols, errors='ignore')

# Step 3: Drop all columns that start with 'Vol Range'
vol_range_cols = [col for col in df_model.columns if col.strip().startswith('Vol Ra
df_model = df_model.drop(columns=vol_range_cols, errors='ignore')


# Step 4: One-hot encode categorical variables
df_model = pd.get_dummies(df_model, drop_first=True)

# Step 5: Separate features and target
X = df_model.drop(columns='GROWTH_READY')
y = df_model['GROWTH_READY']
```

```
# Step 6: Check balance of the target
print(y.value_counts(normalize=True))
```

```
GROWTH_READY
0    0.958771
1    0.041229
Name: proportion, dtype: float64
```

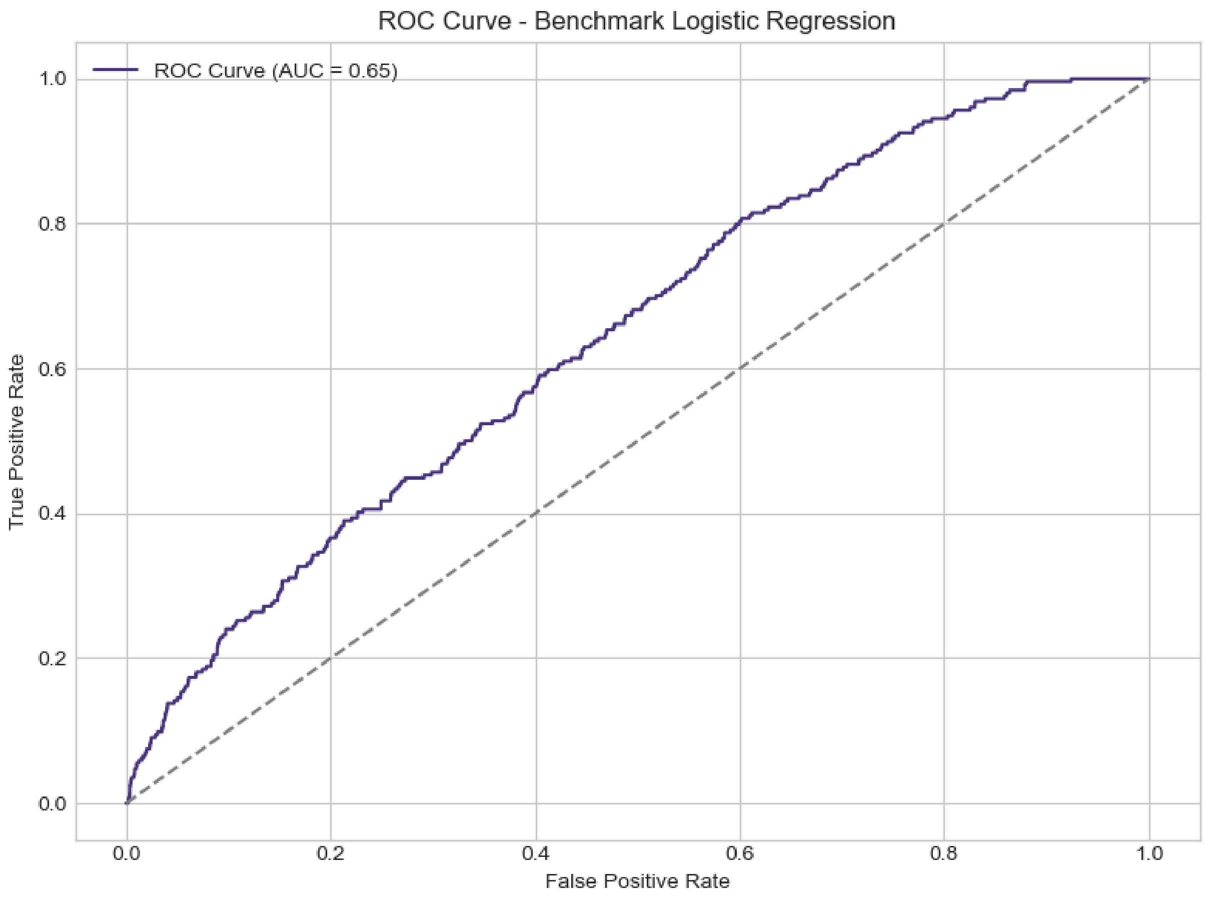
> About 4% of the customers from the historical data are growth-ready and 96% of them fall under any one of these categories - low potential, high potential, or declining growth.

Benchmark Model: Logistic Regression

In [ ]:

```
# Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

# Pipeline to impute NA's + model
benchmark_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('model', LogisticRegression(max_iter=1000, solver='liblinear', random_state=42
])

# Fit pipeline model
benchmark_pipeline.fit(X_train, y_train)

# Make predictions
y_pred_benchmark = benchmark_pipeline.predict(X_test)
y_pred_proba = benchmark_pipeline.predict_proba(X_test)[:, 1]

# Evaluation
print(" Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_benchmark))

print("\n Classification Report:")
print(classification_report(y_test, y_pred_benchmark))

print("\n Accuracy Score:")
print(accuracy_score(y_test, y_pred_benchmark))

print("\n ROC AUC Score:")
print(roc_auc_score(y_test, y_pred_proba))

# Plot ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)



plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc_score(y_test, y_pred_proba):.2
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Benchmark Logistic Regression')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
 Confusion Matrix:
[[5897    0]
 [ 254    0]]

 Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      5897
           1       0.00      0.00      0.00       254

    accuracy                           0.96      6151
   macro avg       0.48      0.50      0.49      6151
weighted avg       0.92      0.96      0.94      6151


 Accuracy Score:
0.9587059014794342

 ROC AUC Score:
0.6450216912643424
```



In [ ]:
```
# Get Model coefficients

model = benchmark_pipeline.named_steps['model']
imputed_X = benchmark_pipeline.named_steps['imputer'].transform(X)

coeff_df = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': model.coef_[0],
    'Odds_Ratio': np.exp(model.coef_[0])
}).sort_values(by='Odds_Ratio', ascending=False)

print("\n Model Summary (Top 10 Features):")
print(coeff_df.head(10))
```

```
 Model Summary (Top 10 Features):
                                     Feature  Coefficient  Odds_Ratio
63             SUB_TRADE_CHANNEL_HOME & HARDWARE     0.048862    1.050075
11                       FREQUENT_ORDER_TYPE_EDI     0.044022    1.045006
89             SUB_TRADE_CHANNEL_PIZZA FAST FOOD     0.041348    1.042215
16                 COLD_DRINK_CHANNEL_BULK TRADE     0.039709    1.040508
31                         TRADE_CHANNEL_GENERAL     0.032059    1.032579
56      SUB_TRADE_CHANNEL_COMPREHENSIVE PROVIDER     0.027957    1.028351
51                  SUB_TRADE_CHANNEL_BULK TRADE     0.010814    1.010873
26                      TRADE_CHANNEL_BULK TRADE     0.010719    1.010777
71  SUB_TRADE_CHANNEL_OTHER ACADEMIC INSTITUTION     0.010572    1.010628
52             SUB_TRADE_CHANNEL_BURGER FAST FOOD     0.007067    1.007092
```

The ROC-AUC curve for the benchmark model is 0.69, any model we build should perform better than this. We are not considering accuracy as there is a class imbalance due to bias. The accuracy of the model would naturally be biased towards the majority classifier which is the non-growth-ready customer segment. Our benchamark model shows a high accuracy of about 96% which is nothing but the proportion of the majority classifer.

According to the logistic regression model, for every customer who falls under the home and hardware sub trade channel, the log odds or the probability of the customer being growth-ready increases by about 11% compared to those who are not growth-ready. Similarly, being in the Pizza Fast Food sub-trade channel increases the log odds or the probability of being growth-ready by 10% compared to the reference group. Other signficant customer characteristics in increasing the likelihood of being growth-ready are those who use digital ordering platforms, fall under the - general retailer trade-channel, fast casual dining trade-channel, and the bulk trade cold drink channel.

Random Forest Model

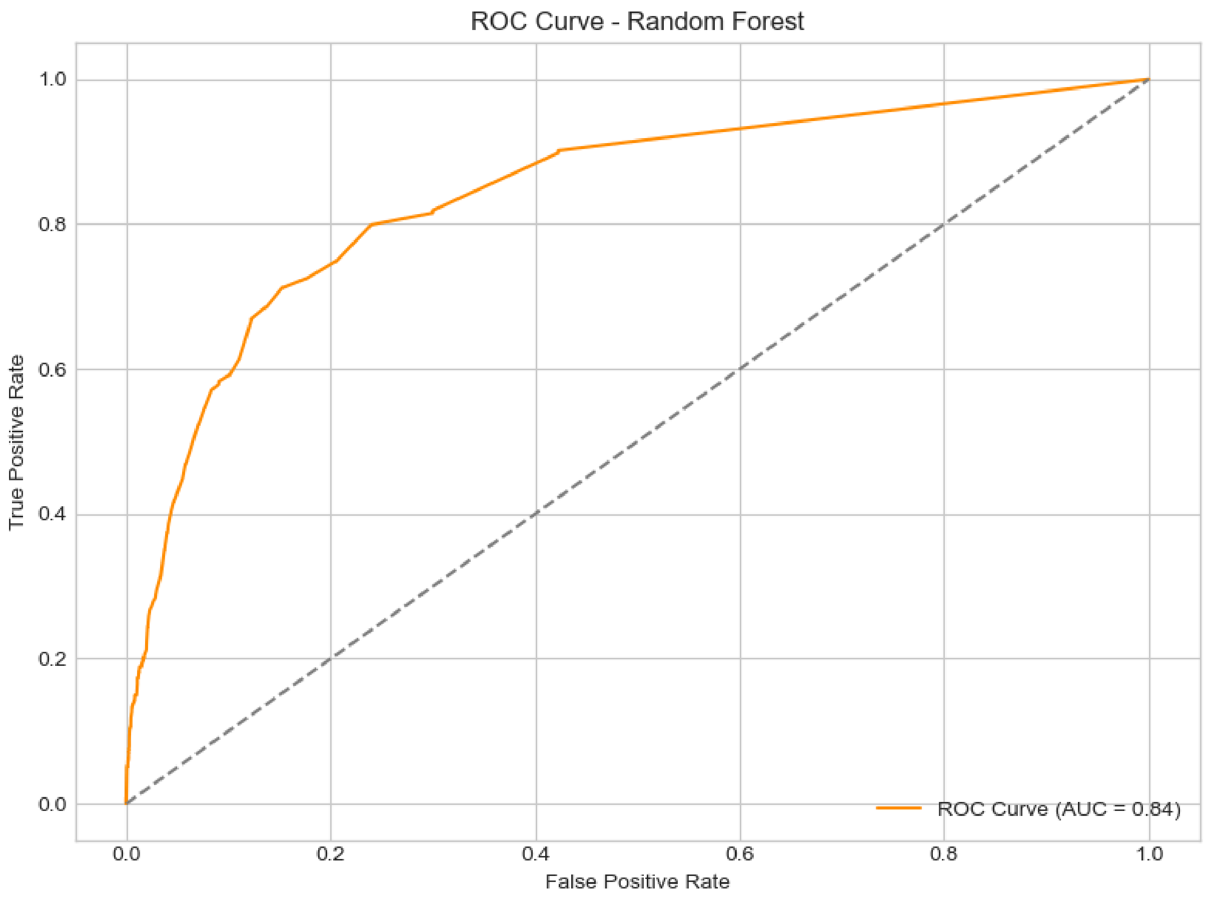
In [ ]:
```
print(X.columns)
```

```
Index(['TRANS_COUNT_2023', 'ANNUAL_VOLUME_CASES_2023',
       'ANNUAL_VOLUME_GALLON_2023', 'ANNUAL_VOLUME_2023',
       'AVG_ORDER_VOLUME_2023', 'LOCAL_MARKET_PARTNER', 'CO2_CUSTOMER',
       'ZIP_CODE', 'DELIVERY_COST_2023_CASES', 'DELIVERY_COST_2023_GALLON',
       'DELIVERY_COST_2023', 'FREQUENT_ORDER_TYPE_EDI',
       'FREQUENT_ORDER_TYPE_MYCOKE LEGACY', 'FREQUENT_ORDER_TYPE_MYCOKE360',
       'FREQUENT_ORDER_TYPE_OTHER', 'FREQUENT_ORDER_TYPE_SALES REP',
       'COLD_DRINK_CHANNEL_BULK TRADE', 'COLD_DRINK_CHANNEL_CONVENTIONAL',
       'COLD_DRINK_CHANNEL_DINING', 'COLD_DRINK_CHANNEL_EVENT',
       'COLD_DRINK_CHANNEL_GOODS', 'COLD_DRINK_CHANNEL_PUBLIC SECTOR',
       'COLD_DRINK_CHANNEL_WELLNESS', 'COLD_DRINK_CHANNEL_WORKPLACE',
       'TRADE_CHANNEL_ACCOMMODATION', 'TRADE_CHANNEL_ACTIVITIES',
       'TRADE_CHANNEL_BULK TRADE', 'TRADE_CHANNEL_COMPREHENSIVE DINING',
       'TRADE_CHANNEL_DEFENSE', 'TRADE_CHANNEL_EDUCATION',
       'TRADE_CHANNEL_FAST CASUAL DINING', 'TRADE_CHANNEL_GENERAL',
       'TRADE_CHANNEL_GENERAL RETAILER', 'TRADE_CHANNEL_GOURMET FOOD RETAILER',
       'TRADE_CHANNEL_HEALTHCARE', 'TRADE_CHANNEL_INDUSTRIAL',
       'TRADE_CHANNEL_LARGE-SCALE RETAILER',
       'TRADE_CHANNEL_LICENSED HOSPITALITY', 'TRADE_CHANNEL_MOBILE RETAIL',
       'TRADE_CHANNEL_OTHER DINING & BEVERAGE',
       'TRADE_CHANNEL_OUTDOOR ACTIVITIES', 'TRADE_CHANNEL_PHARMACY RETAILER',
       'TRADE_CHANNEL_PROFESSIONAL SERVICES',
       'TRADE_CHANNEL_PUBLIC SECTOR (NON-MILITARY)',
       'TRADE_CHANNEL_RECREATION', 'TRADE_CHANNEL_SPECIALIZED GOODS',
       'TRADE_CHANNEL_SUPERSTORE', 'TRADE_CHANNEL_TRAVEL',
       'TRADE_CHANNEL_VEHICLE CARE', 'SUB_TRADE_CHANNEL_BOOKS & OFFICE',
       'SUB_TRADE_CHANNEL_BULK BEVERAGE RETAIL',
       'SUB_TRADE_CHANNEL_BULK TRADE', 'SUB_TRADE_CHANNEL_BURGER FAST FOOD',
       'SUB_TRADE_CHANNEL_CHAIN STORE', 'SUB_TRADE_CHANNEL_CHICKEN FAST FOOD',
       'SUB_TRADE_CHANNEL_CLUB', 'SUB_TRADE_CHANNEL_COMPREHENSIVE PROVIDER',
       'SUB_TRADE_CHANNEL_CRUISE', 'SUB_TRADE_CHANNEL_FAITH',
       'SUB_TRADE_CHANNEL_FRATERNITY', 'SUB_TRADE_CHANNEL_FSR - MISC',
       'SUB_TRADE_CHANNEL_GAME CENTER', 'SUB_TRADE_CHANNEL_HIGH SCHOOL',
       'SUB_TRADE_CHANNEL_HOME & HARDWARE',
       'SUB_TRADE_CHANNEL_INDEPENDENT LOCAL STORE',
       'SUB_TRADE_CHANNEL_MEXICAN FAST FOOD',
       'SUB_TRADE_CHANNEL_MIDDLE SCHOOL', 'SUB_TRADE_CHANNEL_MISC',
       'SUB_TRADE_CHANNEL_MOBILE RETAIL',
       'SUB_TRADE_CHANNEL_NON-RESTAURANT EDUCATION',
       'SUB_TRADE_CHANNEL_ONLINE STORE',
       'SUB_TRADE_CHANNEL_OTHER ACADEMIC INSTITUTION',
       'SUB_TRADE_CHANNEL_OTHER ACCOMMODATION',
       'SUB_TRADE_CHANNEL_OTHER DINING', 'SUB_TRADE_CHANNEL_OTHER FAST FOOD',
       'SUB_TRADE_CHANNEL_OTHER GENERAL RETAIL',
       'SUB_TRADE_CHANNEL_OTHER GOODS', 'SUB_TRADE_CHANNEL_OTHER GOURMET FOOD',
       'SUB_TRADE_CHANNEL_OTHER HEALTHCARE',
       'SUB_TRADE_CHANNEL_OTHER INDUSTRIAL',
       'SUB_TRADE_CHANNEL_OTHER LARGE RETAILER',
       'SUB_TRADE_CHANNEL_OTHER LICENSED HOSPITALITY',
       'SUB_TRADE_CHANNEL_OTHER MILITARY',
       'SUB_TRADE_CHANNEL_OTHER OUTDOOR ACTIVITIES',
       'SUB_TRADE_CHANNEL_OTHER PROFESSIONAL SERVICES',
       'SUB_TRADE_CHANNEL_OTHER PUBLIC SECTOR',
       'SUB_TRADE_CHANNEL_OTHER RECREATION', 'SUB_TRADE_CHANNEL_OTHER TRAVEL',
       'SUB_TRADE_CHANNEL_OTHER VEHICLE CARE',
       'SUB_TRADE_CHANNEL_PIZZA FAST FOOD', 'SUB_TRADE_CHANNEL_PRIMARY SCHOOL',
       'SUB_TRADE_CHANNEL_RECREATION ARENA',
       'SUB_TRADE_CHANNEL_RECREATION FILM',
       'SUB_TRADE_CHANNEL_RECREATION PARK', 'SUB_TRADE_CHANNEL_RESIDENTIAL',
       'SUB_TRADE_CHANNEL_SANDWICH FAST FOOD', 'STATE_SHORT_KY',
       'STATE_SHORT_LA', 'STATE_SHORT_MA', 'STATE_SHORT_MD'],
      dtype='object')
```

In [ ]:

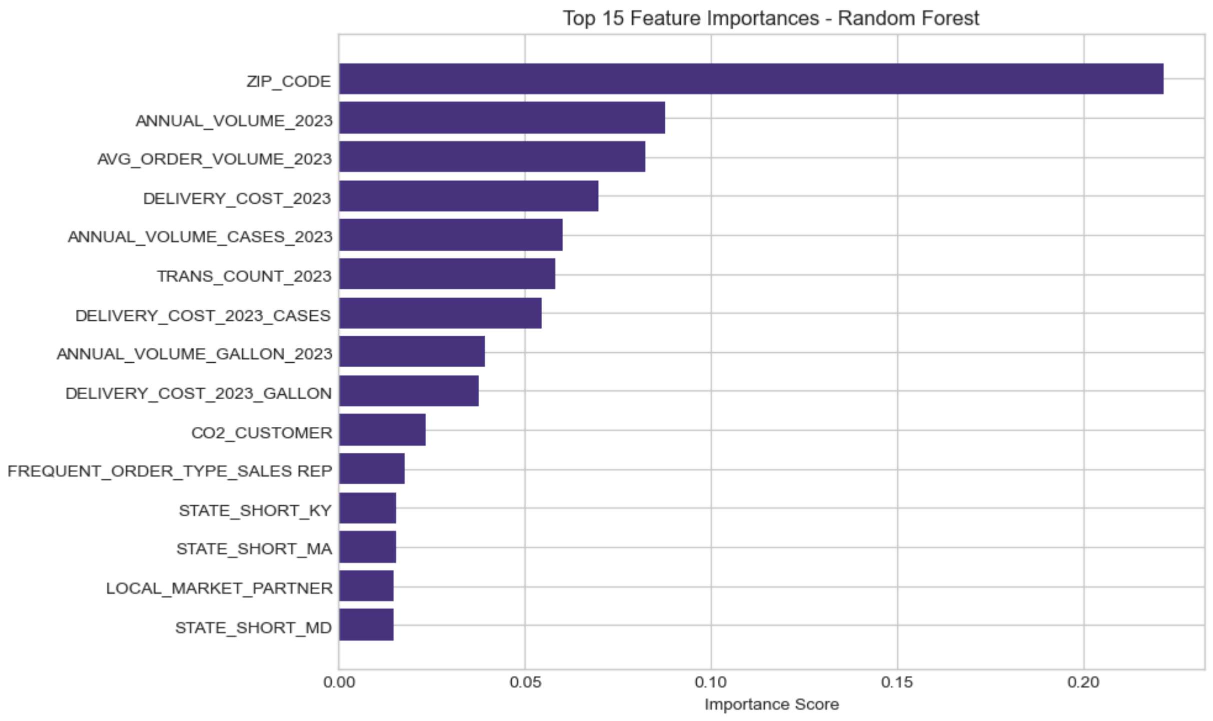
```
# check for any leakages in the target variable
#print([col for col in X.columns if 'THRESHOLD' in col or 'GROWTH_READY' in col])


# Define the pipeline: imputation (for NA's) + random forest
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),  # Replace missing values
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])

# Train the model using the pipeline
pipeline.fit(X_train, y_train)

# Predict on test data
y_pred = pipeline.predict(X_test)
y_prob = pipeline.predict_proba(X_test)[:, 1]  # Probabilities for ROC curve

# Evaluate
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\n Classification Report:")
print(classification_report(y_test, y_pred))

print("\n Accuracy Score:")
print(accuracy_score(y_test, y_pred))

# ROC-AUC
roc_auc = roc_auc_score(y_test, y_prob)
fpr, tpr, _ = roc_curve(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})', color='darkorange')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Random Forest")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Confusion Matrix:
[[5836   61]
 [ 216   38]]

 Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      5897
           1       0.38      0.15      0.22       254

    accuracy                           0.95      6151
   macro avg       0.67      0.57      0.60      6151
weighted avg       0.94      0.95      0.95      6151


 Accuracy Score:
0.9549666720858397
```



In [ ]:
```
### Which charecteristics drive the predictions

# Extract the trained model from pipeline
rf_model_gs = pipeline.named_steps['classifier']

# Get feature importances
importances = rf_model_gs.feature_importances_

# Map them to feature names
feature_names = X.columns
feat_imp_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feat_imp_df = feat_imp_df.sort_values(by='Importance', ascending=False)

# Show top 15 important features
print(" Top 15 Important Features:")
print(feat_imp_df.head(15))

# Plot
plt.figure(figsize=(10, 6))
plt.barh(feat_imp_df['Feature'].head(15), feat_imp_df['Importance'].head(15))
plt.gca().invert_yaxis()
plt.title('Top 15 Feature Importances - Random Forest')
plt.xlabel('Importance Score')
plt.tight_layout()
plt.show()
```

```
 Top 15 Important Features:
                        Feature  Importance
7                      ZIP_CODE    0.221298
3            ANNUAL_VOLUME_2023    0.087552
4         AVG_ORDER_VOLUME_2023    0.082434
10           DELIVERY_COST_2023    0.069633
1      ANNUAL_VOLUME_CASES_2023    0.060201
0               TRANS_COUNT_2023    0.058063
8      DELIVERY_COST_2023_CASES    0.054561
2     ANNUAL_VOLUME_GALLON_2023    0.039360
9     DELIVERY_COST_2023_GALLON    0.037506
6                  CO2_CUSTOMER    0.023296
15  FREQUENT_ORDER_TYPE_SALES REP    0.017694
96               STATE_SHORT_KY    0.015539
98               STATE_SHORT_MA    0.015404
5           LOCAL_MARKET_PARTNER    0.014893
99               STATE_SHORT_MD    0.014684
```

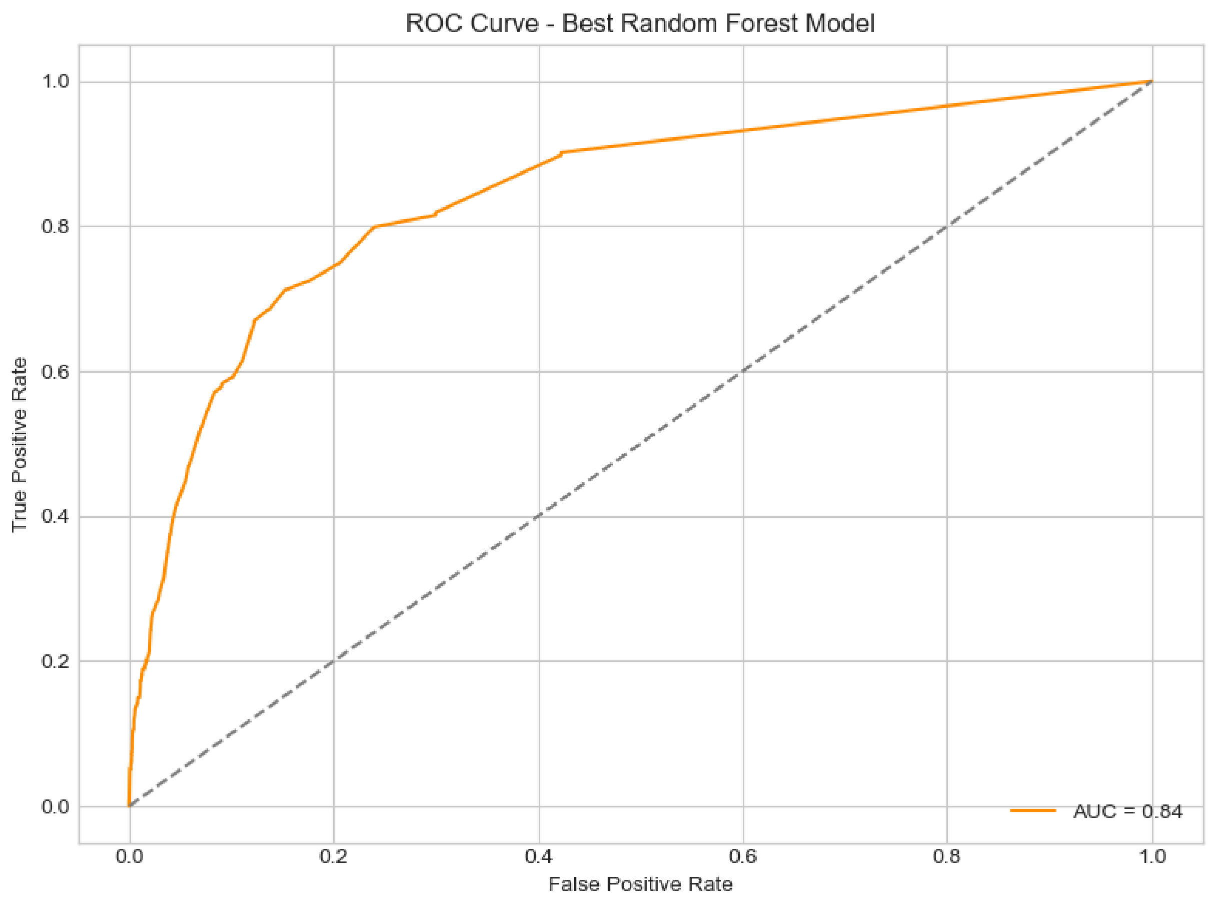


GridSearchCV with Random Forest Classifier

```
In [ ]: # Define pipeline: impute NA's + RF model
        pipeline = Pipeline([
            ('imputer', SimpleImputer(strategy='mean')),
            ('classifier', RandomForestClassifier(random_state=42))
        ])

        # Define hyperparameter grid
        #(note: prefix with classifier__ to access inside pipeline)
        param_grid = {
            'classifier__n_estimators': [50, 100], #no. of trees
            'classifier__max_depth': [None, 5, 10],
            'classifier__min_samples_split': [2, 5],
            'classifier__min_samples_leaf': [1, 2],
            'classifier__max_features': ['sqrt', 'log2']
        }

        # Set up GridSearchCV
        grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid,
                                   cv=5, scoring='f1', n_jobs=-1, verbose=1)

        # Fit model
        grid_search.fit(X_train, y_train)

        # Best model
        best_model = grid_search.best_estimator_

        # Predict on test set
        y_pred = best_model.predict(X_test)
        y_proba = best_model.predict_proba(X_test)[:, 1]  # get probabilities for class 1 (

        # Evaluate
        print("Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred))

        print("\n Classification Report:")
        print(classification_report(y_test, y_pred))

        print("\n Accuracy Score:")
        print(accuracy_score(y_test, y_pred))

        # Get best parameters
        print("\n Best Hyperparameters:")
        print(grid_search.best_params_)


        # Calculate ROC curve
        fpr, tpr, thresholds = roc_curve(y_test, y_proba)

        # Compute AUC
        roc_auc = roc_auc_score(y_test, y_proba)

        # Plot ROC Curve
```

```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}", color='darkorange')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Best Random Forest Model')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Confusion Matrix:
[[5836   61]
 [ 216   38]]

 Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      5897
           1       0.38      0.15      0.22       254

    accuracy                           0.95      6151
   macro avg       0.67      0.57      0.60      6151
weighted avg       0.94      0.95      0.95      6151


 Accuracy Score:
0.9549666720858397

 Best Hyperparameters:
{'classifier__max_depth': None, 'classifier__max_features': 'sqrt', 'classifier__min
_samples_leaf': 1, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 1
00}
```



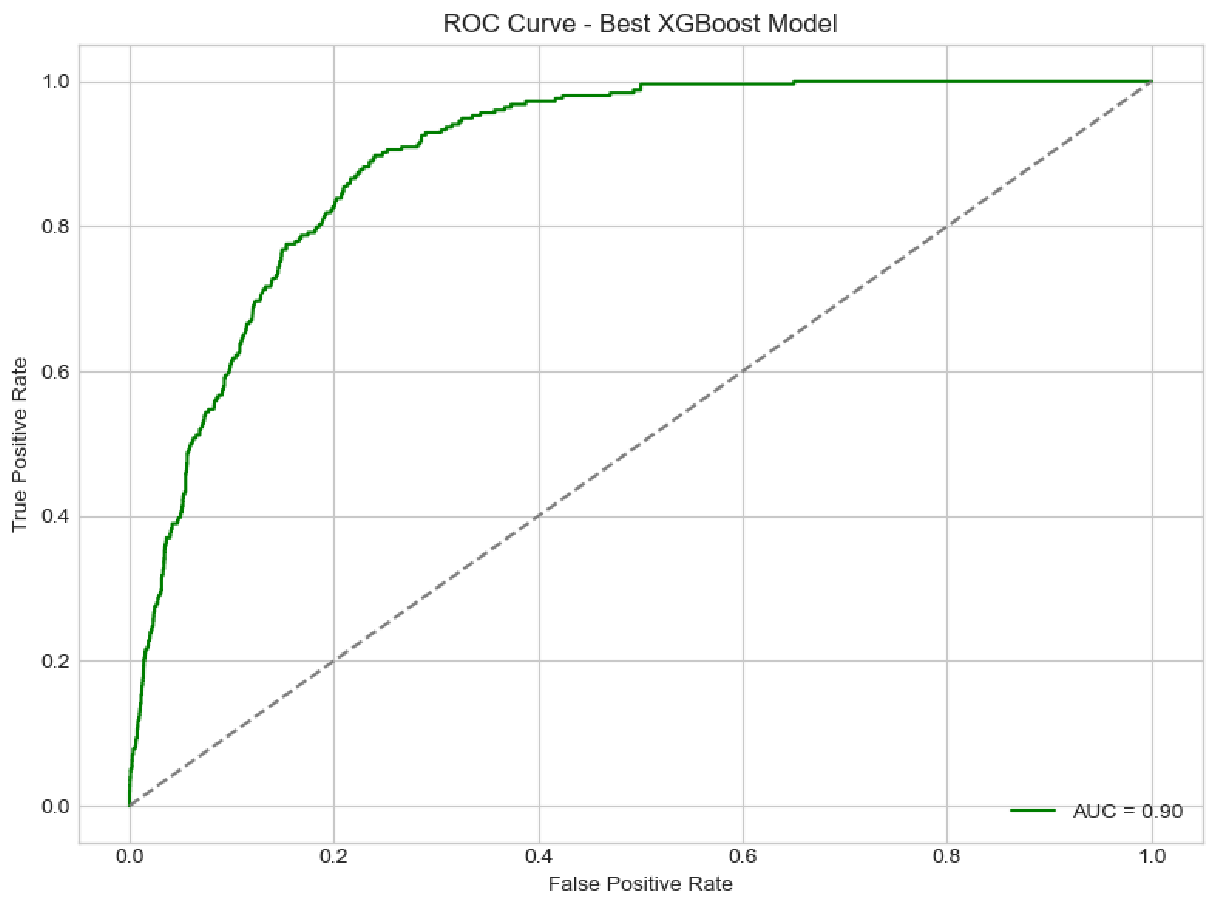
Both the Random Forest model and Random Forest with grid search achieved a ROC-AUC score of 0.84. The best random forest model used 100 trees with no depth limit, considered the square root of features at each split, and allowed nodes to split with at least 2 samples and leaves with at least 1 sample.

GradientBoosting(Using XGBoost) with GridSearch

In [ ]:
```
# Define parameter grid for XGBoost
param_grid_xgb = {
    'n_estimators': [50, 100],
    'max_depth': [3, 5],
    'learning_rate': [0.02, 0.1],
    'subsample': [0.8, 1.0]
}

# Create xgboost model
xgb_clf = xgb.XGBClassifier(
    random_state=42,
    eval_metric='logloss'
)

# Set up GridSearchCV
grid_search_xgb = GridSearchCV(estimator=xgb_clf, param_grid=param_grid_xgb,
                                cv=5, scoring='f1', verbose=1, n_jobs=-1)

# Fit the model
```

```
grid_search_xgb.fit(X_train, y_train)

#  Get the best model
best_xgb = grid_search_xgb.best_estimator_

# Predict on test set from best model
y_pred_xgb = best_xgb.predict(X_test)
y_proba_xgb = best_xgb.predict_proba(X_test)[:, 1]

# Evaluation Metrics
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_xgb))

print("\n Classification Report:")
print(classification_report(y_test, y_pred_xgb))

print("\n Accuracy Score:")
print(accuracy_score(y_test, y_pred_xgb))

# ROC-AUC
roc_auc = roc_auc_score(y_test, y_proba_xgb)
print(f"\n ROC AUC Score: {roc_auc:.4f}")

# Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_proba_xgb)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}", color='green')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Best XGBoost Model')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

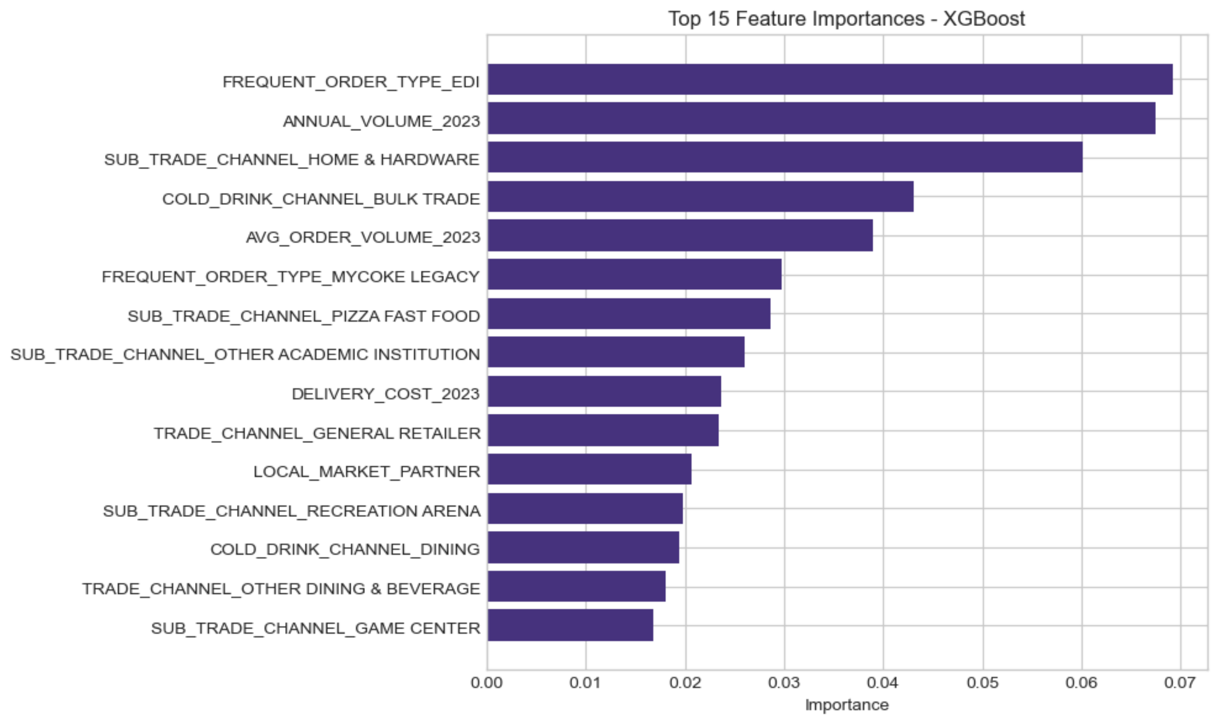
```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Confusion Matrix:
[[5869   28]
 [ 234   20]]

 Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      5897
           1       0.42      0.08      0.13       254

    accuracy                           0.96      6151
   macro avg       0.69      0.54      0.56      6151
weighted avg       0.94      0.96      0.94      6151


 Accuracy Score:
0.9574052999512275

 ROC AUC Score: 0.8954
```



In [ ]:
```
# Best parameters from XGboost gridsearch

print("\n Best Hyperparameters from GridSearchCV:")
print(grid_search_xgb.best_params_)
```

```
 Best Hyperparameters from GridSearchCV:
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100, 'subsample': 0.8}
```

In [ ]:
```
# Feature importance
importances = best_xgb.feature_importances_
feature_names = X.columns
feat_imp_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feat_imp_df = feat_imp_df.sort_values(by='Importance', ascending=False)

# Plot
plt.figure(figsize=(10, 6))
plt.barh(feat_imp_df['Feature'][:15], feat_imp_df['Importance'][:15])
plt.gca().invert_yaxis()
plt.title('Top 15 Feature Importances - XGBoost')
plt.xlabel('Importance')
plt.tight_layout()
plt.show()
```



Out of the 3 models built, the XGBoost using grid search performed the best with an ROC-AUC of 0.89. The best XGBoost model achieved optimal performance with a learning rate of 0.1, using 100 trees of maximum depth 5.

Customers who frequently use EDI (Electronic Data Interchange) for ordering are strong indicators of growth readiness. Features like ANNUAL_VOLUME_2023 and AVG_ORDER_VOLUME_2023 indicate that customers with steady (but not necessarily high) volumes and efficient, larger order sizes tend to grow more over time. Belonging to Bulk Trade and Pizza/Fast Food, Home/Hardware chains is a strong indicator of growth-readiness too. Being a local market partner and having reasonable delivery costs in 2023 also contribute positively to know whether the customer is growth-ready.

Characteristics of customer growth

```
In [ ]: # Mean values for each group
growth_ready_means = df_model[df_model['GROWTH_READY'] == 1].mean()
non_growth_ready_means = df_model[df_model['GROWTH_READY'] == 0].mean()

comparison = pd.DataFrame({
    'Growth-Ready Mean': growth_ready_means,
    'Non-Growth-Ready Mean': non_growth_ready_means,
    'Difference': growth_ready_means - non_growth_ready_means
})

# Sort by absolute difference
comparison = comparison.reindex(comparison['Difference'].abs().sort_values(ascendin

# View top features
comparison.head(10)
```

Out[ ]:

	Growth-Ready Mean	Non-Growth-Ready Mean	Difference
ANNUAL_VOLUME_2023	173.454835	634.760091	-461.305256
DELIVERY_COST_2023	757.840843	1133.184680	-375.343837
ANNUAL_VOLUME_CASES_2023	97.389949	462.443774	-365.053826
DELIVERY_COST_2023_CASES	539.575114	846.035029	-306.459915
ZIP_CODE	30159.637224	30262.626581	-102.989357
ANNUAL_VOLUME_GALLON_2023	76.064887	172.316317	-96.251431
DELIVERY_COST_2023_GALLON	218.265729	287.080417	-68.814687
TRANS_COUNT_2023	10.140379	17.746736	-7.606357
AVG_ORDER_VOLUME_2023	26.337003	24.667788	1.669215
GROWTH_READY	1.000000	0.000000	1.000000

- Some of the top contributors distinguishing growth-ready customers include 2023's annual volume, delivery cost, delivery cost by case, and average order volume.
- The average 2023 annual volume for growth-ready customers was 173 gallons whereas it was 634 gallons for those who are not growth-ready. So on average, growth-ready customers started much smaller in terms of volume.
- Their delivery cost was lower in 2023, with an average of 757 dollars, compared to 1133 dollars of non-growth-ready group, suggesting they were smaller accounts of these customers initially. Specifically for cases, the delivery cost of these customers was significantly lower, with an average of 539 dollars as compared to 846 dollars on average for non-growth-ready customers.
- Growth-ready customers also placed slightly higher number(26) of orders on average, compared to 24 orders on average for those who are non-growth-ready.

Group1: Sensitivity Analysis

In [ ]:
```
# Objective function: Composite Score = min cost + high count of high potential cus

# Define the range of potential thresholds for sensitivity analysis
threshold_range = range(300, 501, 50)

# Empty lists to store metrics
costs = []        # min cost
high_potential_counts = []   # high no. of high potential customers

# Function to compute both delivery cost and high-potential customer count
def compute_delivery_metrics(threshold):
    df = final_df.copy()
    # Categorize customers based on 2023 and 2024 purchase volume
    df['new_threshold_2023'] = np.where(df['ANNUAL_VOLUME_2023'] >= threshold, 'abo
    df['new_threshold_2024'] = np.where(df['ANNUAL_VOLUME_2024'] >= threshold, 'abo

    # Identify high potential customers and growth-ready customers
    df['high_potential'] = np.where(
        ((df['new_threshold_2023'] == 'above') & (df['new_threshold_2024'] == 'abov
        ((df['new_threshold_2023'] == 'below') & (df['new_threshold_2024'] == 'abov
        'high_potential', 'low_potential'
    )

    # Compute delivery costs only for high potential and growth-ready customers
    # Using the correct column names based on previous code
    df['delivery_cost_2023_mod'] = np.where(df['high_potential'] == 'high_potential
    df['delivery_cost_2024_mod'] = np.where(df['high_potential'] == 'high_potential

    # Total delivery cost
    total_cost = df['delivery_cost_2023_mod'].sum(skipna=True) + df['delivery_cost_

    # Count of high-potential customers
    high_potential_count = (df['high_potential'] == 'high_potential').sum()

    return total_cost, high_potential_count

# Run metrics for each threshold
for th in threshold_range:
    cost, high_count = compute_delivery_metrics(th)
    costs.append(cost)
    high_potential_counts.append(high_count)

# Create results DataFrame
sensitivity_df = pd.DataFrame({
    'threshold': threshold_range,
    'total_cost': costs,
    'high_potential_customers': high_potential_counts
})

# Normalize the metrics
sensitivity_df['norm_cost'] = (sensitivity_df['total_cost'] - sensitivity_df['total
sensitivity_df['norm_high_potential'] = (sensitivity_df['high_potential_customers']

# Define weights (tune as per priority)
w_cost = 0.5
w_high = 0.5

# Composite score = lower is better (we reverse high_potential score to reward high
sensitivity_df['composite_score'] = w_cost * sensitivity_df['norm_cost'] + w_high *

# Get best threshold
best_row = sensitivity_df.loc[sensitivity_df['composite_score'].idxmin()]
optimal_threshold = best_row['threshold']

# Shadow price
sensitivity_df['shadow_price'] = sensitivity_df['total_cost'].diff() / sensitivity_

# Results
```

```
print(" Sensitivity Analysis:")
print(sensitivity_df[['threshold', 'total_cost', 'high_potential_customers', 'compo

print(f"\n Optimal Threshold: {optimal_threshold}")
print("\n  Minimum Composite Score Row:")
print(best_row)

print("\n Maximum Shadow Price:")
print(sensitivity_df.loc[sensitivity_df['shadow_price'].idxmax()])
```

```
 Sensitivity Analysis:
   threshold    total_cost  high_potential_customers  composite_score  \
0        300  4.811586e+07                     10114         0.500000
1        350  4.540524e+07                      9030         0.515898
2        400  4.267802e+07                      8035         0.518168
3        450  4.039977e+07                      7275         0.509847
4        500  3.841222e+07                      6630         0.500000

   shadow_price
0           NaN
1 -54212.443132
2 -54544.376874
3 -45565.042835
4 -39751.039227

 Optimal Threshold: 300.0

  Minimum Composite Score Row:
threshold                   3.000000e+02
total_cost                  4.811586e+07
high_potential_customers    1.011400e+04
norm_cost                   1.000000e+00
norm_high_potential         1.000000e+00
composite_score             5.000000e-01
Name: 0, dtype: float64

 Maximum Shadow Price:
threshold                   5.000000e+02
total_cost                  3.841222e+07
high_potential_customers    6.630000e+03
norm_cost                   0.000000e+00
norm_high_potential         0.000000e+00
composite_score             5.000000e-01
shadow_price               -3.975104e+04
Name: 4, dtype: float64
```

For all customers—regardless of whether they are local market partners who buy fountain—the optimal threshold to distinguish high-potential customers in both 2023 and 2024 appears to be 300 gallons. If the threshold were lowered from 400 to 300 gallons, the total delivery cost across both years would minimize to approximately $47 million, while also retaining 3,380 more high-potential customers. This would help Swire increase the total number of high-potential customers to 9,841, striking a better balance between cost efficiency and customer growth potential.

Group2: Analysis on Local Market Partners who only buy Fountain

In [ ]:
```
# Ensure CO2_CUSTOMER and LOCAL_MARKET_PARTNER are converted to boolean
final_df['CO2_CUSTOMER'] = final_df['CO2_CUSTOMER'].astype(bool)
final_df['LOCAL_MARKET_PARTNER'] = final_df['LOCAL_MARKET_PARTNER'].astype(bool)

# Filter: only Local Market Partners who do NOT purchase cases and only buy gallons
loc_f_df = final_df[
    (~final_df['CO2_CUSTOMER']) &
    (final_df['LOCAL_MARKET_PARTNER']) &
    (final_df['ANNUAL_VOLUME_CASES_2023'] == 0) &
    (final_df['ANNUAL_VOLUME_CASES_2024'] == 0)
]

# Summarize total purchases for 2023, 2024, and both years combined
summary = {
    'TOTAL_PURCHASE_2023': loc_f_df['ANNUAL_VOLUME_GALLON_2023'].sum(skipna=True),
    'TOTAL_PURCHASE_2024': loc_f_df['ANNUAL_VOLUME_GALLON_2024'].sum(skipna=True),
    'TOTAL_PURCHASE_2023_2024': (loc_f_df['ANNUAL_VOLUME_GALLON_2023'] + loc_f_df['
}

# Display summary
pd.DataFrame([summary])
```

Out[ ]:

	TOTAL_PURCHASE_2023	TOTAL_PURCHASE_2024	TOTAL_PURCHASE_2023_2024
0	287627.83351	297081.459553	584709.293063

- In 2023, the total annual volume purchased by local market partners who buy fountain is 287627.83 gallons.
- In 2024, the total annual volume purchased by local market partners who buy fountain is 297081.45 gallons.
- A total of 584709 gallons in fountain drinks were purchased by local market partners in both the years.

In [ ]:
```
# Average annual gallon volume in 2023 and 2024
avg_summary = {
    'AVG_PURCHASE_VOLUME_2023': loc_f_df['ANNUAL_VOLUME_GALLON_2023'].mean(skipna=T
    'AVG_PURCHASE_VOLUME_2024': loc_f_df['ANNUAL_VOLUME_GALLON_2024'].mean(skipna=T
}

# Display the result
pd.DataFrame([avg_summary])
```

Out[ ]:

	AVG_PURCHASE_VOLUME_2023	AVG_PURCHASE_VOLUME_2024
0	196.60139	203.063199

In [ ]:
```
# High potential customers in either 2023 or 2024

high_potential_customers = loc_f_df[
    (loc_f_df['ANNUAL_VOLUME_GALLON_2023'] >= 400) |
    (loc_f_df['ANNUAL_VOLUME_GALLON_2024'] >= 400)
][['CUSTOMER_NUMBER', 'ANNUAL_VOLUME_GALLON_2023', 'ANNUAL_VOLUME_GALLON_2024']]

high_potential_customers.head()
```

Out[ ]:

	CUSTOMER_NUMBER	ANNUAL_VOLUME_GALLON_2023	ANNUAL_VOLUME_GALLON_202
202	500266661	737.5	792
212	500267538	730.0	702
213	500267539	810.0	792
249	500269416	942.5	1002
332	500276735	1067.5	992

In [ ]:
```
# count of high potential customers in 2023, 2024, and count of growth-ready custom

high_potential_summary = {
    'ABOVE_THRES_2023': (loc_f_df['THRESHOLD_2023'] == 'above').sum(),
    'ABOVE_THRES_2024': (loc_f_df['THRESHOLD_2024'] == 'above').sum(),
    'GROWTH_READY_CUSTOMERS': ((loc_f_df['THRESHOLD_2023'] != 'above') &
                               (loc_f_df['THRESHOLD_2024'] == 'above')).sum()
}

# Display as a DataFrame
pd.DataFrame([high_potential_summary])
```

Out[ ]:

	ABOVE_THRES_2023	ABOVE_THRES_2024	GROWTH_READY_CUSTOMERS
0	206	192	26

- The average annual volume purchased by local market partners buying fountain was about 196 gallons in 2023, and about 203 gallons in 2024.
- 206 customers in this segment appeared to be high volume customers in 2023, and 192 customers appeared to be high volume customers in 2024.
- 26 customers were found to be growth-ready in this customer segment.

In [ ]:
```
# Characteristics of top 10 highest growth customers:

#(only the customers with a drastic increase in purchase volume rather than thresho


# Step 1: Calculate growth in gallons from 2023 to 2024
loc_f_df['GROWTH'] = loc_f_df['ANNUAL_VOLUME_GALLON_2024'] - loc_f_df['ANNUAL_VOLUM
```

```
# Step 2: Get the top 10 growth customers
top_10_growth_customers = loc_f_df.sort_values(by='GROWTH', ascending=False).head(1

# Step 3: Extract their characteristics and growth info
top_10_summary = top_10_growth_customers[[
    'CUSTOMER_NUMBER',
    'ANNUAL_VOLUME_GALLON_2023',
    'ANNUAL_VOLUME_GALLON_2024',
    'GROWTH',
    'COLD_DRINK_CHANNEL',
    'TRADE_CHANNEL',
    'SUB_TRADE_CHANNEL',
    'FREQUENT_ORDER_TYPE',
    'ZIP_CODE'
]]

# Show the result
top_10_summary.reset_index(drop=True, inplace=True)
top_10_summary
```

Out[ ]:

	CUSTOMER_NUMBER	ANNUAL_VOLUME_GALLON_2023	ANNUAL_VOLUME_GALLON_2024
0	501516973	500.0	6565.0
1	501579393	0.0	2270.0
2	501676564	0.0	1947.5
3	501457271	4777.5	6110.0
4	501494585	0.0	770.0
5	500956234	4417.5	5117.5
6	501569602	132.5	652.5
7	501676506	0.0	515.0
8	501628282	0.0	507.5
9	501654755	0.0	452.5

- Customer 501516973 showed the highest growth from 2023 to 2024 with an increase in purchase of 6065 gallons of fountain drinks. This customer belongs to DINING cold drink channel and often uses new digital ordering platform (MYCOKE360).

Result

- If we are assumed that the price if 1 unit volume is 5 dollars, growth customers net revenue per annual in 300 threshold increases by 26476.83 dollars compared to that of 400 original threshold.

- Volume Related Variables are very important for determining if a customer can transition from Below the Threshold to Above the Threshold. For instance, the top variable in the XGBoost model for predicting transition was Threshold_2023_below which had an importance score 50%. This is much higher than the other predictors which were all less than 10%. (Although the exact units of the importance score are murky, we can still determine how important a predictor is by comparing it's importance score to the other predictors.)

- Logistic Regression model achieved 87% accuracy. Strong performance in identifying likely growth customers → supports targeted marketing and sales.

- Customers were clustered by ZIP code, ordered cases, and gallons into 5 optimized groups.Enables region-based route planning and load balancing based on customer behavior.

- Routing optimization reduces cost across both unit-based (cases) and volume-based (gallons) deliveries, which supports scalable, efficient logistics planning. it enables targeted growth strategies through predictive modeling like below.

% Savings	Before Delivery Cost	After Delivery Cost	Percentaage
Per Case	$40.92	$2.19	94.64%
Per Gallon	$18.97	$3.61	80.95%

- Delivery costs increased by an average of about 56 dollars from 2023 to 2024.

- Only about 4% of customers were growth-ready. These customers started smaller in annual volume and had lower delivery costs, but placed slightly more orders than others.

- For Group 1 customers, lowering the threshold to 300 gallons to distinguish high-potential customers, serves the purpose of maintaining Swire’s logistic efficiency. Over the 2 years, this would minimize the total delivery cost to $47 million while also increasing the number of high potential customers to 9,841.

Group Member Contribution

- Richard Lim: Generating the pipeline to obtain what is the optimal threshold compared to the original threshold, and how much efficiency is gained out of threshold transition.

- Varun Selvam: Determining top factors that enable customers to transition from Below The Threshold in 2023 to Above the Threshold in 2024. Performed Customer Segmentation as well.

- Meenakshi H : Customer growth Identification, Logistic Regression, ZIP based Clustering , Cases vs Gallons Comparison , Validating Cost Efficiency,Cluster level cost summary.

- Nikita Muddapati: Modelling, Growth-ready Customer Characteristics, Customer Segment 1 Optimal Threshold, Customer Segment 2 Analysis.