Group 6 Modeling Notebook

2025-03-22

Introduction

Project Goal
The project goal is to identify characteristics of customers that order above a specific threshold annually to determine what might indicate "high growth potential" in customers below the threshold.

Business Problem Summary
Swire Coca-Cola wants to optimize logistic transport costs by changing some direct delivery ("red truck") customers to a third-party delivery ("white truck"). They need to identify characteristics of customers which order greater than 400 gallons of product per year in order to determine which customers below this threshold might have "high growth potential". These "high growth potential" customers may exceed the 400 gallon threshold in the future if they continue with red truck delivery and business support services instead of being swapped to white truck.

Analytics Problem Summary
The analytics problem is to identify important features which indicate customers with “high growth potential”, based on historical sales data and customer characteristics. This may involve supervised predictive analytics such as classification models and unsupervised descriptive analytic algorithms.

Notebook Purpose
This notebook includes the modeling activities and findings. The primary language used in the notebook is R, however, some modeling sections are written in Python and will be marked accordingly.

Data Preparation

We utilized the same dataset created in our EDA, with the addition of 3 columns: “cases_total”, “ordered_total”, and “fountain_only”. We also created a copy of the final dataset subset to the customer segment of interest to Swire Coca-Cola: customers that are Local Market Partners, who do not order CO2, and who order fountain drinks only.

Load Packages and Data

Load all R packages.

```
library(corrplot)
library(ggcorrplot)
library(fastDummies)
library(tidyr)
library(dplyr)
library(ggplot2)
library(rlang)
library(tidyverse)
```

```
library(lubridate)
library(caret)
library(glmnet)
library(grf)
library(rsample)
library(ranger)
library(tidymodels)
library(themis)
```

Load the datasets.

```
customer_profile_raw <- read.csv("customer_profile.csv")
customer_address_and_zip_raw <- read.csv("customer_address_and_zip_mapping.csv")
transactional_data_raw <- read.csv("transactional_data.csv")
```

Join and Clean the Datasets

Joining Datasets into wide tables.

```
# join customer address to customer profile
customers_raw_joined <- left_join(customer_profile_raw, customer_address_and_zip_raw, by
=c('ZIP_CODE' = 'zip'))

# customer joined with transactions
customers_trans_joined <- left_join(transactional_data_raw, customers_raw_joined, by
='CUSTOMER_NUMBER')
```

Transform our character variables into factors or other datatypes where appropriate.

```
# get character columns
cust_tran_chr_cols <- names(customers_trans_joined)[sapply(customers_trans_joined,
is.character)]
str(customers_trans_joined[cust_tran_chr_cols])

# convert dates to dates
date_columns_tran_cust <- c('TRANSACTION_DATE', 'FIRST_DELIVERY_DATE', 'ON_BOARDING_DATE')
customers_trans_joined_dates <- customers_trans_joined %>%
  mutate(across(all_of(date_columns_tran_cust), ~ case_when(
    TRUE ~ as.Date(., format = "%m/%d/%Y")
  )))

# convert remain chr to factors excluding full address
customers_trans_joined_factored <- customers_trans_joined_dates %>%
  mutate(across(where(is.character) & !matches("full.address"), as.factor))
```

Correct for Nulls

```
# create copy of joined and factored dataset
stg_cust_tran <- customers_trans_joined_factored
stg_cust_tran$PRIMARY_GROUP_NUMBER <- factor(ifelse(is.na(stg_cust_tran$PRIMARY_GROUP_NUMBER),
# convert NAs to...
                                          "NA",            # ...a chr factor
                                          stg_cust_tran$PRIMARY_GROUP_NUMBER))

stg_cust_tran_clean <- stg_cust_tran
```

Feature Engineering

Create total ordered gallons.

```
# created ordered total column
stg_cust_tran_clean$ordered_Total <- stg_cust_tran$ORDERED_CASES +
stg_cust_tran$ORDERED_GALLONS
```

Calculate Percent Change from 2023 to 2024 and create a summary dataset from the wide table, grouped by customer.

```
# Calculate total order per year for each customer
customer_year_totals <- stg_cust_tran_clean %>%
  # Group by customer and year
  group_by(CUSTOMER_NUMBER, YEAR) %>%
  summarise(total_order_per_year = sum(ordered_Total, na.rm = TRUE), .groups = "drop")
# Sum for each group

# Calculate percent change year-over-year for each customer
customer_year_totals <- customer_year_totals %>%
  # Ensure the data is ordered by customer and year
  arrange(CUSTOMER_NUMBER, YEAR) %>%
  # Group by customer to calculate percentage change within each group
  group_by(CUSTOMER_NUMBER) %>%
  mutate(percent_change = ((total_order_per_year - lag(total_order_per_year)) /
lag(total_order_per_year)) * 100)
# Calculate YoY change
```

Pivot and create a binary indicator of growth (positive percent change).

```
# Pivot the data so each customer has 1 row with cols for each year
customer_year_summary <- customer_year_totals %>%
  select(CUSTOMER_NUMBER, YEAR, total_order_per_year) %>%
  pivot_wider(names_from = YEAR, values_from = total_order_per_year, names_prefix =
```

```
"total_order_") %>%
  rename(total_order_2023 = total_order_2023, total_order_2024 = total_order_2024)

# Calculate the percent change between 2023 and 2024
customer_year_summary <- customer_year_summary %>%
  mutate(percent_change = ((total_order_2024 - total_order_2023) / total_order_2023) *
100)
```

Convert the percent change into a categorical variable with grew, declined, or stayed the same.

```
# Categorize customers into 'grew', 'declined', or 'stayed_same'
customer_year_summary <- customer_year_summary %>%
  mutate(growth_category = case_when(
    percent_change > 0 ~ "grew",
    percent_change < 0 ~ "declined",
    percent_change == 0 | abs(percent_change) < 0.001 ~ "stayed_same",
    TRUE ~ "unknown"
  ))
```

Create variable for years since first delivery.

```
# Calculate the number of years since first delivery
customer_years <- stg_cust_tran_clean %>%
  select(CUSTOMER_NUMBER, FIRST_DELIVERY_DATE) %>%
  distinct() %>%     # Ensure one row per customer
  mutate(years_since_first_delivery = 2024 - year(as.Date(FIRST_DELIVERY_DATE)))

# Join with pivoted data containing percent change
customer_summary <- customer_year_summary %>%
  left_join(customer_years, by = "CUSTOMER_NUMBER")
```

Re-add customer trade channel to the new customer summary dataset with added features.

```
# Drop ALL columns named TRADE_CHANNEL before joining
customer_year_summary <- customer_year_summary %>%
  select(-contains("TRADE_CHANNEL"), everything())

# Extract unique CUSTOMER_NUMBER and TRADE_CHANNEL from original dataset
customer_trade_channel <- stg_cust_tran_clean %>%
  select(CUSTOMER_NUMBER, TRADE_CHANNEL) %>%
  distinct()

# Perform LEFT JOIN only once
customer_year_summary <- customer_year_summary %>%
  left_join(customer_trade_channel, by = "CUSTOMER_NUMBER")
```

Adding the CO2 column into the summary dataset.

```
# Extract unique CUSTOMER_NUMBER and CO2_CUSTOMER from original dataset
customer_co2 <- stg_cust_tran_clean %>%
  select(CUSTOMER_NUMBER, CO2_CUSTOMER) %>%
  distinct()

# LEFT JOIN to add CO2_CUSTOMER to the customer_year_summary dataset
customer_year_summary <- customer_year_summary %>%
  left_join(customer_co2, by = "CUSTOMER_NUMBER")
```

Add cold drink channel to the dataset.

```
# Extract unique CUSTOMER_NUMBER and COLD_DRINK_CHANNEL from original dataset
customer_cold_drink_channel <- stg_cust_tran_clean %>%
  select(CUSTOMER_NUMBER, COLD_DRINK_CHANNEL) %>%
  distinct()

# LEFT JOIN to add COLD_DRINK_CHANNEL to the customer_year_summary dataset
customer_year_summary <- customer_year_summary %>%
  left_join(customer_cold_drink_channel, by = "CUSTOMER_NUMBER")
```

Add Sub-trade Channel to the dataset.

```
# Extract unique CUSTOMER_NUMBER and SUB_TRADE_CHANNEL
customer_sub_trade_channel <- stg_cust_tran_clean %>%
  select(CUSTOMER_NUMBER, SUB_TRADE_CHANNEL) %>%
  distinct()

# LEFT JOIN to add SUB_TRADE_CHANNEL
customer_year_summary <- customer_year_summary %>%
  left_join(customer_sub_trade_channel, by = "CUSTOMER_NUMBER")
```

Add Local Market Partners.

```
# Extract unique CUSTOMER_NUMBER and LOCAL_MARKET_PARTNER
customer_local_market <- stg_cust_tran_clean %>%
  select(CUSTOMER_NUMBER, LOCAL_MARKET_PARTNER) %>%
  distinct()

# LEFT JOIN to add LOCAL_MARKET_PARTNER
customer_year_summary <- customer_year_summary %>%
  left_join(customer_local_market, by = "CUSTOMER_NUMBER")
```

Add Order Types.

```
# Extract unique CUSTOMER_NUMBER and FREQUENT_ORDER_TYPE
customer_frequent_order <- stg_cust_tran_clean %>%
  select(CUSTOMER_NUMBER, FREQUENT_ORDER_TYPE) %>%
  distinct()

# LEFT JOIN to add FREQUENT_ORDER_TYPE to customer_year_summary
customer_year_summary <- customer_year_summary %>%
  left_join(customer_frequent_order, by = "CUSTOMER_NUMBER")
```

Create new features to capture customer engagement across multiple order types, creating a column for each order type with the number of orders placed.

```
# Identify customers with more than one order type
multi_order_customers <- stg_cust_tran_clean %>%
  group_by(CUSTOMER_NUMBER) %>%
  summarise(distinct_order_types = n_distinct(ORDER_TYPE, na.rm = TRUE)) %>%
  filter(distinct_order_types > 1) %>%
  pull(CUSTOMER_NUMBER)

# Get the order types and their amounts for those customers
order_type_details <- stg_cust_tran_clean %>%
  filter(CUSTOMER_NUMBER %in% multi_order_customers) %>%
  group_by(CUSTOMER_NUMBER, ORDER_TYPE) %>%
  summarise(total_amount = sum(ordered_Total, na.rm = TRUE), .groups = "drop") %>%
  arrange(CUSTOMER_NUMBER, desc(total_amount))

# Pivot the data
order_type_details_pivot <- order_type_details %>%
  pivot_wider(names_from = ORDER_TYPE, values_from = total_amount, values_fill = 0)

# Left join the pivoted order type columns to the main dataset
customer_year_summary <- customer_year_summary %>%
  left_join(order_type_details_pivot, by = "CUSTOMER_NUMBER")
```

Create a new feature to indicate fountain only customers (no cans or bottles ordered).

```
# create boolean if customer is fountain only (no cases ordered)
fountain_only <- stg_cust_tran_clean %>%
  group_by(CUSTOMER_NUMBER) %>%
  summarise(cases_total = sum(ORDERED_CASES),
            fountain_total = sum(ORDERED_GALLONS)) %>%
  mutate(fountain_only = ifelse(cases_total == 0 & fountain_total >= 1,
                                TRUE,
                    # True if no cases ordered and some fountain ordered
                                FALSE))  # False otherwise
```

```
# add to dataset
customer_year_summary <- customer_year_summary %>%
  left_join(fountain_only, by = "CUSTOMER_NUMBER")
```

Final Cleaning

Remove introduced NAs.

```
# dropping nas from customer yearly summary
complete_data <- customer_year_summary %>%
  drop_na()
```

Create the subset of customers of interest to Swire Coca-Cola: customers that are Local Market Partners, who do not order CO2, and who order fountain drinks only.

```
# make filtered dataset
complete_data_subset <- complete_data %>%
  filter(LOCAL_MARKET_PARTNER == TRUE
         & CO2_CUSTOMER == FALSE
         & fountain_only == TRUE) %>%
  select(-c(LOCAL_MARKET_PARTNER, CO2_CUSTOMER, fountain_only))
```

Save both the final customer summary dataset and the subset to .csv files to ensure group members utilize the same data.

```
# write to csv files
write.csv(complete_data, file = "complete_data.csv", row.names = FALSE)
write.csv(complete_data_subset, file = "complete_data_subset.csv", row.names = FALSE)
```

Modeling Process

After completing feature engineering and data preparation, we explored a range of models with the highest potential for generating valuable insights. These included logistic regression (with LASSO and ridge regularization), XGBoost, random forest, causal forest, k-means and k-medoid clustering, and DBSCAN clustering. The following sections outline the modeling process, results, and key takeaways for each approach.

Logistic Regression

The goal in fitting the data to a logistic regression model was to understand which customer characteristics are associated with being a high-growth or low-growth customer. Since our customer growth variable,

growth_category, is a binary categorical outcome, logistic regression was an appropriate choice. It allowed us to model the probability of a customer belonging to the high-growth category based on predictor variables.

Additional Data Preparation

Before fitting the model, we needed to ensure the categorical variables were factors and ID-like variables were dropped, since those kinds of variables tend to create overfitting in the modeling while providing very little actual predictive power for regression models.

```
# Ensure categorical variables are factors
complete_data <- complete_data %>%
 mutate(across(where(is.character), as.factor))

# Drop ID-like columns
model_data <- complete_data %>%
 select(-CUSTOMER_NUMBER, -TRADE_CHANNEL.y)

# Ensure target variable is a factor
model_data$growth_category <- as.factor(model_data$growth_category)

# Split data
set.seed(42)
train_index <- createDataPartition(model_data$growth_category, p = 0.8, list = FALSE)
train_data <- model_data[train_index, ]
test_data <- model_data[-train_index, ]

# Check for missing values and handle them
train_data <- na.omit(train_data)
```

Original Logistic Regression Model

The first attempt at a simple logistic model, using growth_category as the target variable and train_data as the data set, resulted in an algorithm that didn't converge:

```
# Original model
logistic_model <- glm(growth_category ~ ., data = train_data, family = binomial)
summary(logistic_model)
```

```
Call:
glm(formula = growth_category ~ ., family = binomial, data = train_data)

Coefficients: (30 not defined because of singularities)
                                          Estimate Std. Error z value
(Intercept)                              6.836e+02  6.769e+05   0.001
total_order_2023                        -3.454e+00  9.034e+02  -0.004
total_order_2024                         3.385e+00  1.158e+03   0.003
percent_change                           2.203e+00  1.178e+03   0.002
TRADE_CHANNEL.xACTIVITIES               -1.104e+02  4.914e+05   0.000
TRADE_CHANNEL.xBULK TRADE               -1.023e+03  5.290e+05  -0.002
TRADE_CHANNEL.xCOMPREHENSIVE DINING      4.421e+00  3.155e+05   0.000
```

Lasso and Ridge Logistic Regression

To try to fix the non-converging original model, we next attempted leveraging lasso and ridge regularization with the logistic model. The result was a model that was functional unlike the original, but which had an accuracy of 0 in classifying customers in growth categories.

```
# Prepare the data
X <- model.matrix(growth_category ~ ., data = train_data)[, -1]  # Remove intercept
y <- train_data$growth_category  # Response variable

# Convert y to a factor (if needed)
y <- as.factor(y)

# Define a sequence of lambda values to test
lambda_seq <- 10^seq(3, -3, by = -0.1)

# Fit Lasso Logistic Regression (alpha = 1 for Lasso)
lasso_model <- cv.glmnet(X, y, family = "binomial", alpha = 1, lambda = lambda_seq)

# Fit Ridge Logistic Regression (alpha = 0 for Ridge)
ridge_model <- cv.glmnet(X, y, family = "binomial", alpha = 0, lambda = lambda_seq)

# Find optimal lambda values
best_lambda_lasso <- lasso_model$lambda.min
best_lambda_ridge <- ridge_model$lambda.min

# Print best lambda values
print(paste("Best Lambda for Lasso:", best_lambda_lasso))
print(paste("Best Lambda for Ridge:", best_lambda_ridge))
```

```
[1] "Best Lambda for Lasso: 0.001"
[1] "Best Lambda for Ridge: 0.001"
```

```
# Get coefficients for the best Lasso model
lasso_coef <- coef(lasso_model, s = "lambda.min")
# print(lasso_coef) # lengthy output suppressed

# Get coefficients for the best Ridge model
ridge_coef <- coef(ridge_model, s = "lambda.min")
# print(ridge_coef) # lengthy output suppressed

# Prepare the test data and ensure no NA values in the response variable
# Subset test_data to remove rows with NA in 'growth_category' (if any)
test_data_subset <- test_data[!is.na(test_data$growth_category), ]

# Check that both test data and predictions align
X_test <- model.matrix(growth_category ~ ., data = test_data_subset)[, -1]
y_test <- test_data_subset$growth_category

# Predict on test data
```

```r
lasso_preds <- predict(lasso_model, s = best_lambda_lasso, newx = X_test, type = "response")
ridge_preds <- predict(ridge_model, s = best_lambda_ridge, newx = X_test, type = "response")

# Convert predictions to binary outcome (ensure same length)
lasso_class <- ifelse(lasso_preds > 0.5, 1, 0)
ridge_class <- ifelse(ridge_preds > 0.5, 1, 0)

# Ensure that the predicted and actual labels have the same length
lasso_class <- lasso_class[1:length(y_test)]
ridge_class <- ridge_class[1:length(y_test)]

# Evaluate accuracy (assuming test_data$growth_category is the true label)
lasso_accuracy <- mean(lasso_class == y_test)
ridge_accuracy <- mean(ridge_class == y_test)

# Print the accuracies
print(paste("Lasso Accuracy:", lasso_accuracy))
print(paste("Ridge Accuracy:", ridge_accuracy))
```

```
[1] "Lasso Accuracy: 0"
[1] "Ridge Accuracy: 0"
```

Subsequent attempts included changing the prediction type from response to class, and changing the method of creating lambda values from lambda.min to lambda.1se, none of which were successful in creating a model with any predictive accuracy.

XG Boost

Note: All code in this section is written in Python.

Complete Dataset Without Grid Search

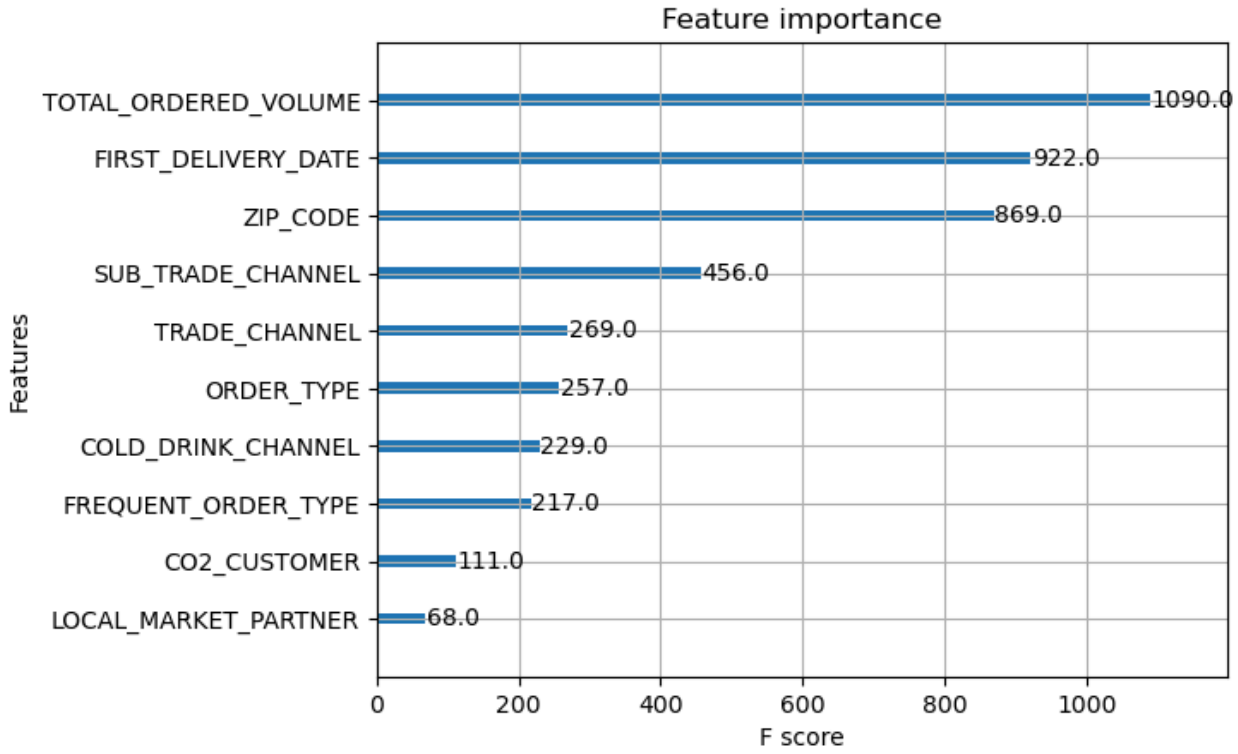
Perform XG Boost on the whole dataset of customers, without using a grid search.

```python
# Copying dataset
df2 = full_customer_data.copy()

# Drop unwanted columns (dates and unnecessary features)
df2 = df2.drop(columns=[ "ON_BOARDING_DATE", "PERCENT_CHANGE", "GROWTH_SAME_DECLINE", 2023.0,
2024.0, "CUSTOMER_NUMBER"])

# Convert datetime columns to the number of days since a reference date
reference_date = pd.to_datetime("2022-01-01")  # You can change this reference

df2["FIRST_DELIVERY_DATE"] = (pd.to_datetime(df2["FIRST_DELIVERY_DATE"]) -
reference_date).dt.days

# Convert categorical columns to numeric
```

```
label_encoders = {}
for col in df2.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df2[col] = le.fit_transform(df2[col])
    label_encoders[col] = le  # Store encoder for future use

# Define features and target variable
X = df2.drop(columns=['GROWTH_DECLINE'])
y = df2['GROWTH_DECLINE']

# Split data into training and testing sets
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the XGBoost model (correct approach)
model = XGBClassifier(
    objective="binary:logistic",
    eval_metric="logloss",
    max_depth=6,
    learning_rate=0.1,
    n_estimators=100  # Now correctly used
)

# Train the model
model.fit(X_train2, y_train2)

# Make predictions
y_pred2 = model.predict(X_test2)

# Print accuracy
accuracy = accuracy_score(y_test2, y_pred2)
print(f"Accuracy: {accuracy:.4f}")

# Print classification report
print(classification_report(y_test2, y_pred2))
```

```
Accuracy: 0.6316
              precision    recall  f1-score   support

           0       0.63      0.89      0.74      2933
           1       0.64      0.28      0.38      2111

    accuracy                           0.63      5044
   macro avg       0.63      0.58      0.56      5044
weighted avg       0.63      0.63      0.59      5044
```

```
# Feature importance plot
xgb.plot_importance(model)
plt.show()
```



Using the whole dataset without Grid Search, the feature importance ranking highlights TOTAL_ORDERED_VOLUME (1090.0), FIRST_DELIVERY_DATE (922.0), and ZIP_CODE (869.0) as the dominant predictors of growth. Compared to previous models, these features have significantly higher importance, suggesting that incorporating the full dataset reinforces their influence. The classification report shows an accuracy of 63.16%, which is an improvement over earlier runs.

For non-growth cases (class 0), the model achieves precision of 0.63 and recall of 0.89, meaning it correctly identifies most non-growth instances. However, for growth cases (class 1), precision is 0.64, but recall remains low at 0.28, leading to an F1-score of 0.38, indicating continued difficulty in capturing growth patterns. The macro average (0.58 F1-score) and weighted average (0.59 F1-score) confirm an overall improvement but highlight persistent class imbalance.

While the model benefits from more data, the imbalance in recall for class 1 remains an issue, suggesting the need for Grid Search optimization to better capture growth trends.

Complete Dataset With Grid Search

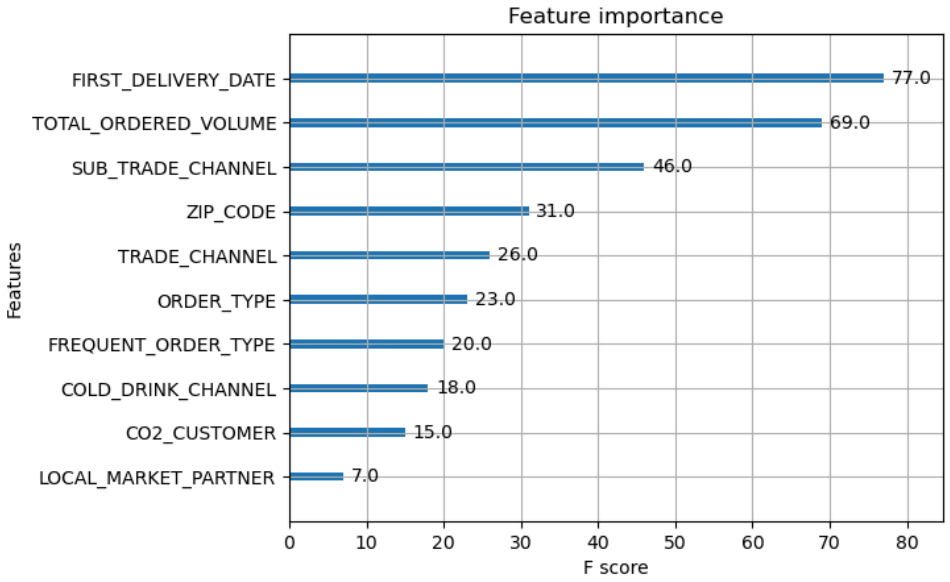
Perform XG Boost on the whole dataset of customers, applying a grid search for hyperparameter optimization.

```
#  define hyperparamater grid
param_grid = {
    'max_depth': [3, 6, 9],              # Controls tree complexity
    'learning_rate': [0.01, 0.1, 0.2],   # How much each tree contributes
    'n_estimators': [50, 100, 200],      # Number of boosting rounds
    'subsample': [0.8, 1.0],             # Fraction of data per boosting round
    'colsample_bytree': [0.8, 1.0],      # Fraction of features per tree
    'reg_lambda': [0, 1, 10],            # L2 regularization
```

```
    'reg_alpha': [0, 1, 10]             # L1 regularization
}

# set up grid serch with 5 folds cross validation
grid_search = GridSearchCV(
    XGBClassifier(objective="binary:logistic", eval_metric="logloss"),
    param_grid,
    scoring='accuracy',  # You can use 'f1' if recall is important
    cv=5,                # 5-fold cross-validation
    n_jobs=-1,           # Use all CPU cores
    verbose=2
)

# train with grid search
grid_search.fit(X_train2, y_train2)

# Get best parameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# train the best model
best_model = XGBClassifier(**best_params)
best_model.fit(X_train2, y_train2)

# Make predictions
y_pred2 = best_model.predict(X_test)

# Evaluate the final model
accuracy = accuracy_score(y_test2, y_pred2)
print(f"Optimized Accuracy: {accuracy:.4f}")
print(classification_report(y_test2, y_pred2))
```

```
Fitting 5 folds for each of 972 candidates, totalling 4860 fits
Best Hyperparameters: {'colsample_bytree': 1.0, 'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 50, 'reg_alpha': 10, 'reg_lambda': 0, 'subs
ample': 1.0}
Optimized Accuracy: 0.6279
              precision    recall  f1-score   support

           0       0.62      0.94      0.75      2933
           1       0.70      0.19      0.30      2111

    accuracy                           0.63      5044
   macro avg       0.66      0.57      0.52      5044
weighted avg       0.65      0.63      0.56      5044
```

```
# Plot new feature importance
xgb.plot_importance(best_model)
plt.show()
```



After applying Grid Search to the whole dataset, the feature importance analysis shows that FIRST_DELIVERY_DATE (77.0) and TOTAL_ORDERED_VOLUME (69.0) remain the most influential factors, while SUB_TRADE_CHANNEL (46.0) and ZIP_CODE (31.0) also play significant roles. The optimized model achieves an accuracy of 62.79%, indicating a slight improvement over the previous version.

For non-growth cases (class 0), the model performs well with precision (0.62) and recall (0.94), leading to an F1-score of 0.75. However, for growth cases (class 1), precision improves to 0.70, but recall remains very low at 0.19, resulting in a weak F1-score of 0.30. The macro average F1-score (0.52) and weighted average (0.56) highlight the ongoing class imbalance issue.

While Grid Search helped optimize hyperparameters, the model still struggles with identifying growth cases due to low recall.

Subset Dataset Without Grid Search

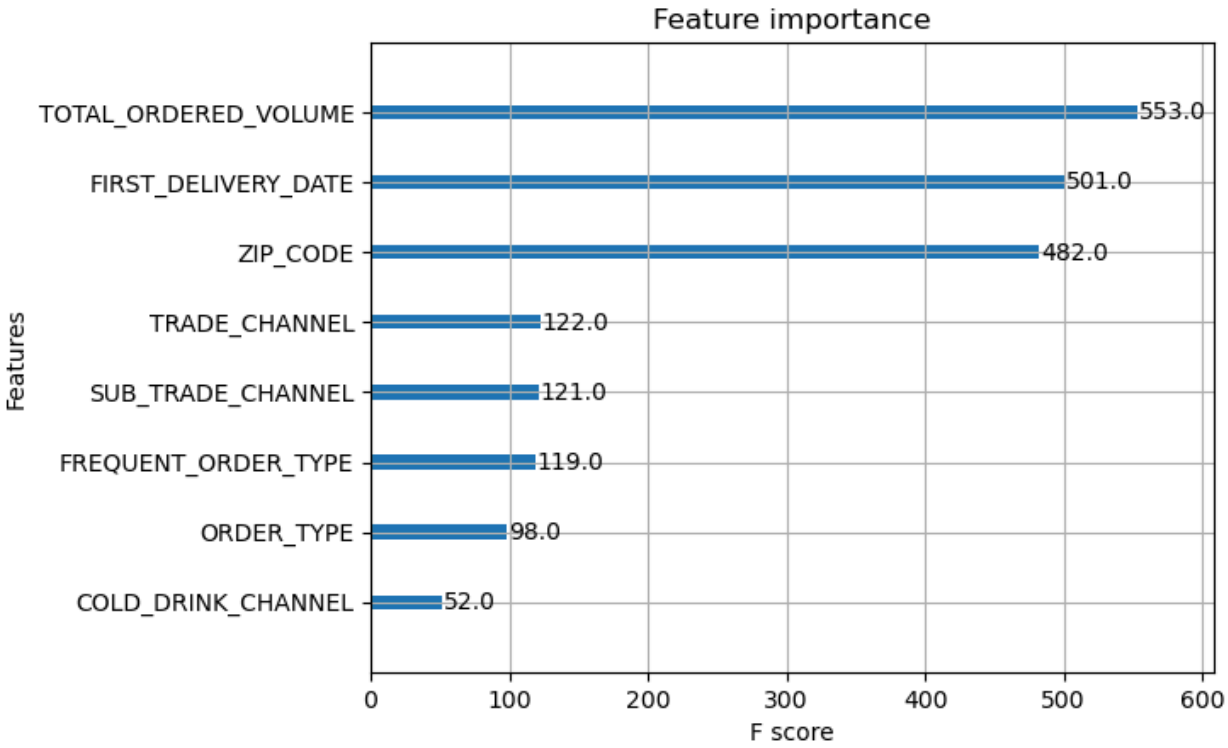
Perform XG Boost on the subset of customers, without using a grid search.

```
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Copying dataset
df = local_market_data.copy()

# Drop unwanted columns (dates and unnecessary features)
df = df.drop(columns=[ "ON_BOARDING_DATE", "PERCENT_CHANGE", "GROWTH_SAME_DECLINE", 2023.0,
2024.0, "CUSTOMER_NUMBER"])

# Convert datetime columns to the number of days since a reference date
```

```
reference_date = pd.to_datetime("2022-01-01")  # You can change this reference

df["FIRST_DELIVERY_DATE"] = (pd.to_datetime(df["FIRST_DELIVERY_DATE"]) - reference_date).dt.days

# Convert categorical columns to numeric
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le  # Store encoder for future use

# Define features and target variable
X = df.drop(columns=['GROWTH_DECLINE'])
y = df['GROWTH_DECLINE']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the XGBoost model (correct approach)
model = XGBClassifier(
    objective="binary:logistic",
    eval_metric="logloss",
    max_depth=6,
    learning_rate=0.1,
    n_estimators=100  # Now correctly used
)

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

# Make predictions
y_pred = model.predict(X_test)

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.5938
              precision    recall  f1-score   support

           0       0.65      0.74      0.69       119
           1       0.46      0.36      0.40        73

    accuracy                           0.59       192
   macro avg       0.55      0.55      0.55       192
weighted avg       0.58      0.59      0.58       192
```

```
# Feature importance plot
xgb.plot_importance(model)
plt.show()
```



The feature importance analysis shows that TOTAL_ORDERED_VOLUME, FIRST_DELIVERY_DATE, and ZIP_CODE are the most influential factors in predicting customer growth or decline. The classification report indicates a moderate accuracy of 59.38%, with a precision of 0.65 and recall of 0.74 for non-growth (class 0), meaning the model predicts this class well. However, for growth cases (class 1), precision drops to 0.46 and recall to 0.36, showing difficulty in identifying these instances. The macro and weighted averages (both around 0.55-0.59) confirm this imbalance. This suggests a need for better handling of class imbalance, feature engineering, or hyperparameter tuning, which we addressed in the next step with Grid Search.

Subset Dataset With Grid Search

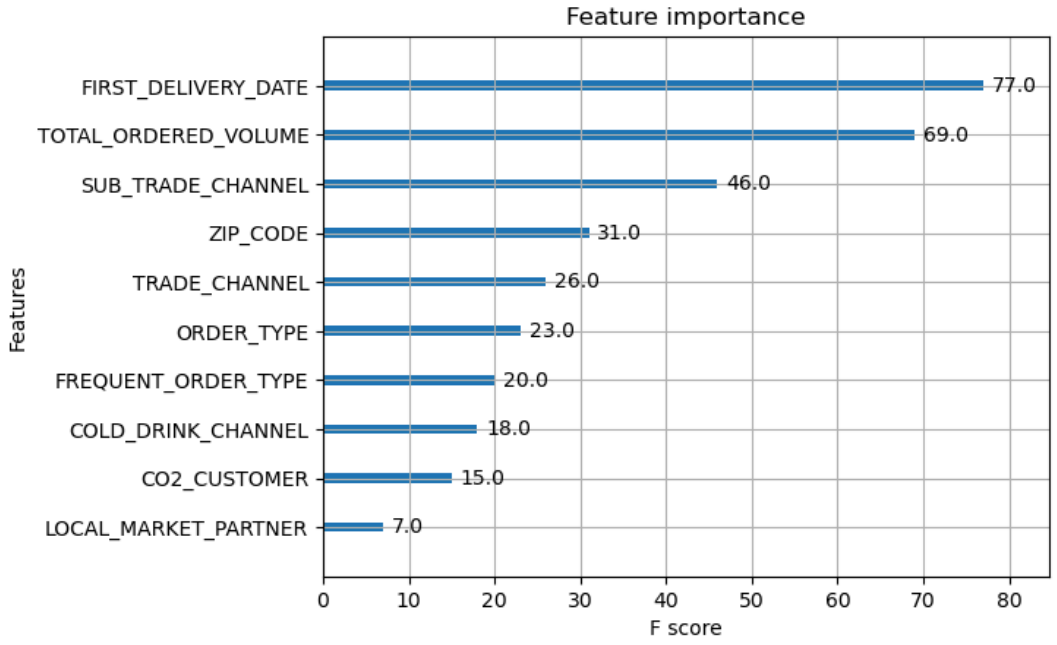
Perform XG Boost on the subset of customers, applying a grid search for hyperparameter optimization.

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

#  define hyperparamater grid
param_grid = {
    'max_depth': [3, 6, 9],              # Controls tree complexity
    'learning_rate': [0.01, 0.1, 0.2],   # How much each tree contributes
    'n_estimators': [50, 100, 200],      # Number of boosting rounds
    'subsample': [0.8, 1.0],             # Fraction of data per boosting round
    'colsample_bytree': [0.8, 1.0],      # Fraction of features per tree
    'reg_lambda': [0, 1, 10],            # L2 regularization
```

```
    'reg_alpha': [0, 1, 10]             # L1 regularization
}

# set up grid search with 5 folds cross validation
grid_search = GridSearchCV(
    XGBClassifier(objective="binary:logistic", eval_metric="logloss"),
    param_grid,
    scoring='accuracy',  # You can use 'f1' if recall is important
    cv=5,                # 5-fold cross-validation
    n_jobs=-1,           # Use all CPU cores
    verbose=2
)

# train with grid search
grid_search.fit(X_train, y_train)

# Get best parameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# train the best model
best_model = XGBClassifier(**best_params)
best_model.fit(X_train, y_train)

# Make predictions
y_pred = best_model.predict(X_test)

# Evaluate the final model
accuracy = accuracy_score(y_test, y_pred)
print(f"Optimized Accuracy: {accuracy:.4f}")
print(classification_report(y_test, y_pred))
```

```
Optimized Accuracy: 0.6279
              precision    recall  f1-score   support

           0       0.62      0.94      0.75      2933
           1       0.70      0.19      0.30      2111

    accuracy                           0.63      5044
   macro avg       0.66      0.57      0.52      5044
weighted avg       0.65      0.63      0.56      5044
```

```
# Plot new feature importance
xgb.plot_importance(best_model)
plt.show()
```



After applying Grid Search, the model's optimized feature importance ranking shows that FIRST_DELIVERY_DATE (77.0) and TOTAL_ORDERED_VOLUME (69.0) remain the most influential variables, but SUB_TRADE_CHANNEL (46.0) has gained more significance compared to the previous model.

The classification report indicates an improved accuracy of 62.79%, showing better overall performance. However, the imbalance between class 0 (non-growth) and class 1 (growth) remains evident. Class 0 achieves high recall (0.94) and a strong F1-score (0.75), meaning the model predicts this class well. However, for class 1, precision (0.70) is better, but recall remains low (0.19), leading to a weak F1-score (0.30). The macro and weighted averages (0.66 and 0.65, respectively) confirm the imbalance. While Grid Search improved accuracy, the model still struggles with recall for growth cases, suggesting that class weighting or resampling techniques might be needed for further enhancement.

Random Forest

In this analysis, the goal was to understand which customer characteristics are associated with being a high-growth or low-growth customer. Given that our outcome variable, *growth_category*, is binary, a random forest classifier was an appropriate choice. This ensemble method allowed us to capture complex interactions between predictor variables while reducing the risk of overfitting. By aggregating multiple decision trees, the model provided robust predictions and insights into the most influential customer characteristics.

Complete Dataset

```
# Split the data into train and test datasets
set.seed(2319)

data_split = initial_split(complete_data, strata = growth_category, prop = 0.75)
data_train = training(data_split)
data_test = testing(data_split)
```

```r
# Set the ratio to not be over-represented
data_recipe = recipe(growth_category ~ ., data = complete_data) |>
  step_downsample(growth_category, under_ratio = 1.5)

# Create the model
model_data = rand_forest(
  mtry = tune(),
  trees = tune(),
  min_n = tune()) |>
  set_mode("classification") |>
  set_engine("ranger")

# Set the number of cross validation folds
backorder_folds = vfold_cv(data_train, v = 5)

# Create the model
data_wf = workflow() |>
  add_model(model_data) |>
  add_recipe(data_recipe)

# Tune the model
doParallel::registerDoParallel(cores = 10)

data_tune = data_wf |>
  tune_grid(
    resamples = backorder_folds,
    grid = 50
  )

# Select the best hyperparameter combination
best_model = data_tune |>
  select_best(metric = "roc_auc")

# Set the final model workflow
final_wf =
  data_wf |>
  finalize_workflow(best_model)

# Fit the final model
final_fit = final_wf |>
  last_fit(split = data_split)

# Produce the model metrics
final_fit |>
  collect_metrics()
```

.metric <chr>	.estimator <chr>	.estimate <dbl>	.config <chr>
accuracy	multiclass	0.99791715	Preprocessor1_Model1
roc_auc	hand_till	0.99986487	Preprocessor1_Model1
brier_class	multiclass	0.02177747	Preprocessor1_Model1

```
# Produce a confusion matrix
final_fit |>
  collect_predictions() |>
  conf_mat(truth = growth_category, estimate = .pred_class)
```

```
             Truth
Prediction    declined grew stayed_same
  declined        2362    0           0
  grew               0 1925           0
  stayed_same        0    9          25
```

From this we learn that the model had 99.79% accuracy in predicting the correct class for whether or not the customer's orders grew, declined or stayed the same. It also received a 99.98% ROC, indicating the model has near-perfect discriminatory power, meaning it can almost flawlessly distinguish between the classes. The confusion matrix shows that only 0.208% of predictions were incorrect using this model.

Looking at only customers that are local market partners and only order fountain drinks, the code was adjusted to fit this dataset.

Subset Dataset

```
# Split the data into train and test datasets
set.seed(2319)

d_split = initial_split(subset_data, strata = growth_category, prop = 0.75)
train = training(d_split)
test = testing(d_split)

# Set the ratio to not be over-represented
data_recipe_local = recipe(growth_category ~ ., data = subset_data) |>
  step_downsample(growth_category, under_ratio = 1.5)

# Create the model
model_data_local = rand_forest(
  mtry = tune(),
  trees = tune(),
  min_n = tune()) |>
  set_mode("classification") |>
  set_engine("ranger")

# Set the number of cross validation folds
backorder_folds_local = vfold_cv(train, v = 5)

# Create the model
data_wf_local = workflow() |>
  add_model(model_data_local) |>
  add_recipe(data_recipe_local)
```

```
# Tune the model
doParallel::registerDoParallel(cores = 10)

data_tune_local = data_wf_local |>
  tune_grid(
    resamples = backorder_folds_local,
    grid = 50
  )

# Select the best hyperparameter combination
best_model_local = data_tune_local |>
  select_best(metric = "roc_auc")

# Set the final model workflow
final_wf_local =
  data_wf_local |>
  finalize_workflow(best_model_local)

# Fit the final model
final_fit_local = final_wf_local |>
  last_fit(split = d_split)

# Produce the model metrics
final_fit_local |>
  collect_metrics()
```

.metric <chr>	.estimator <chr>	.estimate <dbl>	.config <chr>
accuracy	multiclass	0.98639456	Preprocessor1_Model1
roc_auc	hand_till	0.99623352	Preprocessor1_Model1
brier_class	multiclass	0.01368302	Preprocessor1_Model1

```
# Produce a confusion matrix
final_fit_local |>
  collect_predictions() |>
  conf_mat(truth = growth_category, estimate = .pred_class)
```

```
             Truth
Prediction    declined grew stayed_same
  declined          85    0           0
  grew               0   57           0
  stayed_same        0    2           3
```

From these results, we can see that there was also a very high accuracy and ROC, 98.6% and 99.6% respectively. The confusion matrix is slightly worse than the full dataset, with 1.36% of customers categorized incorrectly with this model.

These results, while initially appearing promising, are actually showing that the model is heavily overfitting the data. It is fitting too heavily to the training data, meaning that any results found from this model should not be used on further data, or new customers may be categorized incorrectly, leading to the incorrect categorization of high and low growth customers.

Causal Forest

In this analysis, the goal was to understand which customer characteristics influence the likelihood of being a high-growth or low-growth customer. To uncover heterogeneous treatment effects, we used a causal forest model. This approach could allow us to estimate how different customer attributes impact growth outcomes while accounting for potential confounding factors. By leveraging an ensemble of decision trees, the causal forest could provide nuanced insights into which characteristics drive customer growth and how these effects vary across the population.

Complete Dataset

First, causal forest was run on the full dataset.

```
# Convert dataset to factor variables properly
cd <- complete_data
cd[] <- lapply(cd, function(x) if (is.character(x)) as.factor(x) else x)

# Make growth_category a binary variable
cd$growth_category <- ifelse(cd$growth_category == "grew", 1, 0)

# Select covariates (excluding percent_change and growth_category)
X <- cd %>% select(-percent_change, -growth_category)

# Convert categorical variables in X to numeric (one-hot encoding)
X <- model.matrix(~ . -1, data = X)  # One-hot encode factors & remove intercept

# Ensure X, percent_change, and growth_category have no missing values
if (any(is.na(X)) | any(is.na(cd$percent_change)) | any(is.na(cd$growth_category))) {
  stop("Missing values detected after preprocessing.")
}

# Train the causal forest
cf <- causal_forest(
  X = X,                          # Covariates as matrix (fully numeric)
  Y = cd$percent_change,          # Outcome variable (percent change)
  W = cd$growth_category,         # Treatment indicator
  num.trees = 2000
)

# Get Individual Treatment Effects (ITE) for first 10 observations
ite_predictions <- predict(cf)$predictions
ite_predictions[1:10]
```

```r
# Get Average Treatment Effect (ATE)
ate_result <- average_treatment_effect(cf, target.sample = "all")

# Print ATE and confidence interval
print(ate_result)
```

```
 estimate  std.err
      NaN       NA
```

Subset Dataset

Causal Forest was then run on the subset with local market partners that only purchase fountain drinks:

```r
# Convert dataset to factor variables properly
cd_local <- subset_data
cd_local[] <- lapply(cd_local, function(x) if (is.character(x)) as.factor(x) else x)

# Convert growth_category into a binary variable
cd_local$growth_category <- ifelse(cd_local$growth_category == "grew", 1, 0)

# Select covariates (excluding percent_change and growth_category)
X <- cd_local %>% select(-percent_change, -growth_category)

# Convert categorical variables in X to numeric (one-hot encoding)
X <- model.matrix(~ . -1, data = X)  # One-hot encode factors & remove intercept

# Ensure X, percent_change, and growth_category have no missing values
if (any(is.na(X)) | any(is.na(cd_local$percent_change)) | any(is.na(cd_local$growth_category)))
{
  stop("Missing values detected after preprocessing.")
}

# Train the causal forest
cf_local <- causal_forest(
  X = X,                          # Covariates as matrix (fully numeric)
  Y = cd_local$percent_change,         # Outcome variable (percent change)
  W = cd_local$growth_category,         # Treatment indicator
  num.trees = 2000
)

# Get Individual Treatment Effects (ITE) for first 10 observations
ite_predictions <- predict(cf_local)$predictions
ite_predictions[1:10]

# Get Average Treatment Effect (ATE)
ate_result_local <- average_treatment_effect(cf, target.sample = "all")

# Print ATE and confidence interval
print(ate_result_local)
```

```
estimate  std.err
     NaN       NA
```

The results from both causal forest models indicates that the ATE could not be properly calculated. The standard error and confidence intervals could not be calculated because the models produced no ATE. This is an indicator that the model was potentially overfitting the training data when running the models. It could also mean that the features used were highly correlated or irrelevant, meaning more feature engineering may be needed for a causal forest to be useful in this analysis. Both causal forest models were run with multiple different numbers of trees and confirming no NA's in the data, yielding only the result of NaN.

Ultimately, the inability of random forest and causal forest models to produce meaningful insights strongly suggests that either more feature engineering is needed or a different analytical approach is needed to derive valuable conclusions.

Partition Clustering

Partition clustering was used to see if there were potential groups of customers with similar characteristics. If groups were identified that captured a lot of variation within the dataset, we could try to figure out which characteristics were associated with different groups and perhaps identify high-potential or low-potential clusters of customers. K-means, k-medoid, and kernel k-means using radial basis function clustering was tested. The kernel k-means was not included in this notebook as initial runs with a slightly different version of the dataset had poor results, and it required extremely high processing and run time.

Additional Data Prep for Clustering

Recode categorical variables into dummy variables or remove them, then scale all columns.

```
# Remove columns captured by engineered fields
complete_trimmed <- complete_data %>%
  select(-c(CUSTOMER_NUMBER,           # ID column
            TRADE_CHANNEL.y            # duplicate
          )) %>%
  mutate(percent_change = ifelse(is.na(percent_change),        # convert NAs to...
                                 0,                            # 0
                                 percent_change),              # otherwise leave the same
         percent_change = ifelse(is.infinite(percent_change),  # convert Infs to...
                                 0,                            # 0
                                 percent_change)               # otherwise leave the same
         )

# Extract identifier column
CUSTOMER_NUMBER_col <- complete_data$CUSTOMER_NUMBER

# Extract non-categorical cols
complete_num <- complete_trimmed %>%
  select(-c(growth_category, TRADE_CHANNEL.x, CO2_CUSTOMER, COLD_DRINK_CHANNEL,
            SUB_TRADE_CHANNEL, LOCAL_MARKET_PARTNER, FREQUENT_ORDER_TYPE,
```

```
            fountain_only))

# Recode categorical into dummy variables
complete_recoded <- complete_trimmed %>%
  # create dummy variables model
  dummyVars(data = ., formula = ~ growth_category + TRADE_CHANNEL.x
            + CO2_CUSTOMER + COLD_DRINK_CHANNEL + SUB_TRADE_CHANNEL
            + LOCAL_MARKET_PARTNER + FREQUENT_ORDER_TYPE + fountain_only
            ) %>%
  # apply dummy variables model
  predict(object = .,
          newdata = complete_trimmed)

# combine and scale all columns
complete_dummy_scaled <- cbind(complete_num, complete_recoded) %>%
  scale(x = .,
        scale = TRUE,
        center = TRUE) %>%
  as.data.frame()
```

Do the same for the subset of customers.

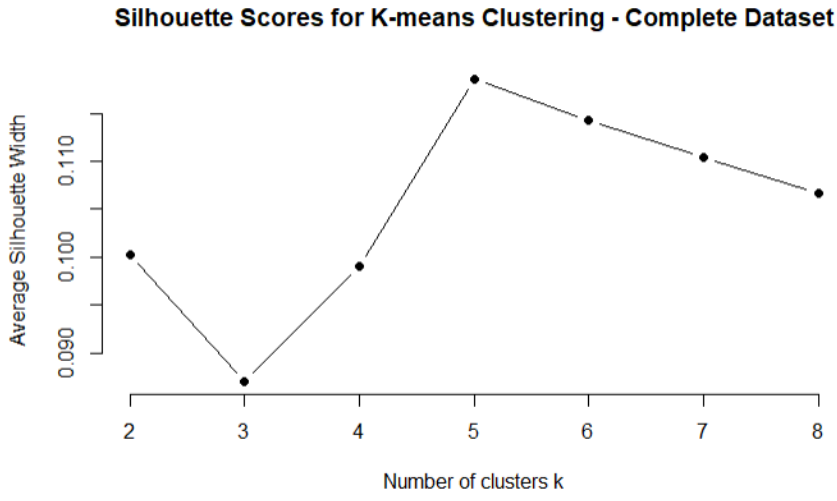
```
# Remove columns captured by engineered fields
subset_trimmed <- complete_data_subset %>%
  select(-c(CUSTOMER_NUMBER,           # ID column
            TRADE_CHANNEL.y,           # duplicate
            Cases_total, EDI           # no variation
          )) %>%
  mutate(percent_change = ifelse(is.na(percent_change),        # convert NAs to...
                                 0,                            # 0
                                 percent_change),              # otherwise leave the same
         percent_change = ifelse(is.infinite(percent_change),  # convert Infs to...
                                 0,                            # 0
                                 percent_change)               # otherwise leave the same
         )

# Extract identifier column
CUSTOMER_NUMBER_col_subset <- complete_data_subset$CUSTOMER_NUMBER

# Extract non-categorical cols
subset_num <- subset_trimmed %>%
  select(-c(growth_category, TRADE_CHANNEL.x, COLD_DRINK_CHANNEL,
            SUB_TRADE_CHANNEL, FREQUENT_ORDER_TYPE))

# Recode categorical into dummy variables
subset_recoded <- subset_trimmed %>%
  # create dummy variables model
  dummyVars(data = ., formula = ~ growth_category + TRADE_CHANNEL.x
            + COLD_DRINK_CHANNEL + SUB_TRADE_CHANNEL + FREQUENT_ORDER_TYPE
            ) %>%
```

```
  # apply dummy variables model
  predict(object = .,
          newdata = subset_trimmed)

# combine and scale all columns
subset_dummy_scaled <- cbind(subset_num, subset_recoded) %>%
  scale(x = .,
        scale = TRUE,
        center = TRUE) %>%
  as.data.frame()
```

K-means Clustering

Try k means partition clustering on full dataset with dummy variables, using Euclidean distance.

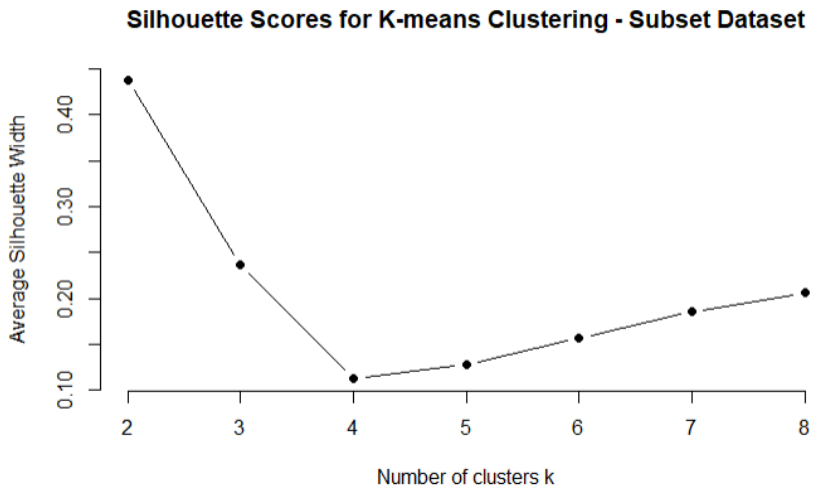
```
# set seed
set.seed(25463)

# define function to calculate silhouette widths
silhouette_widths_kmean <- function(k) {
  kmeans <- kmeans(complete_dummy_scaled, centers = k, nstart = 20, iter.max = 100) # kmeans
with k centers
  dist <- dist(complete_dummy_scaled, method = "euclidean")
  sil <- silhouette(kmeans$cluster, dist)
  mean(sil[ ,3]) # calculate silhouette for each and take mean
}

# Calculate average silhouette width for different numbers of clusters
avg_silwidth_kmean <- sapply(2:8, silhouette_widths_kmean) # 8 = Max reasonable # of groups

# Plot the Silhouette Method
plot(2:8, avg_silwidth_kmean, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of clusters k",
     ylab = "Average Silhouette Width",
     main = "Silhouette Scores for K-means Clustering - Complete Dataset")
```



The best value looks around 0.16 or so, well below the ideal silhouette score of 0.5 or more. This model is not accurately describing variation in the dataset.

Try k means partition clustering on the subset with dummy variables, using Euclidean distance.

```
# set seed
set.seed(25463)

# define function to calculate silhouette widths
silhouette_widths_kmean <- function(k) {
  kmeans <- kmeans(subset_dummy_scaled, centers = k, nstart = 20, iter.max = 100) # kmeans with
k centers
  dist <- dist(subset_dummy_scaled, method = "euclidean")
  sil <- silhouette(kmeans$cluster, dist)
  mean(sil[ ,3]) # calculate silhouette for each and take mean
}

# Calculate average silhouette width for different numbers of clusters
avg_silwidth_kmean <- sapply(2:8, silhouette_widths_kmean) # 8 = Max reasonable # of customer
groups

# Plot the Silhouette Method
plot(2:8, avg_silwidth_kmean, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of clusters k",
     ylab = "Average Silhouette Width",
     main = "Silhouette Scores for K-means Clustering - Subset Dataset")
```

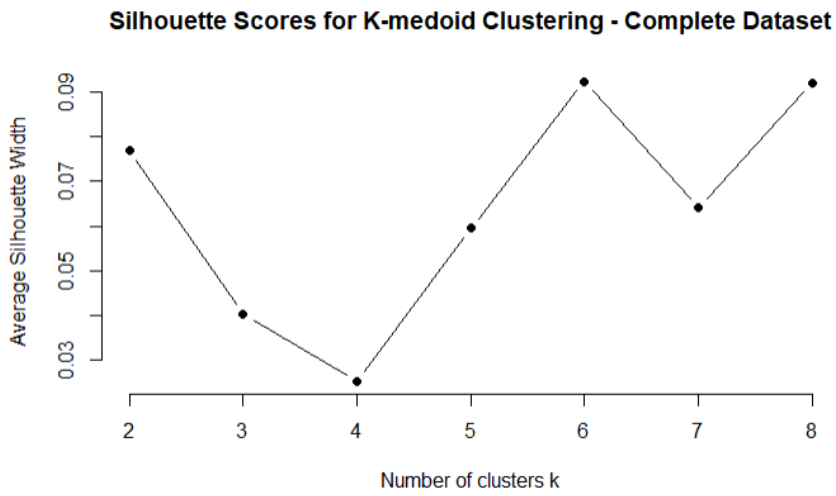


The best score is below 0.45, this time barely below the ideal threshold of 0.5.

K-medoid Clustering

Try k-medoid partition clustering on full dataset with dummy variables, using Manhattan distance.

```
# set seed
set.seed(25463)

# define function to calculate silhouette widths
silhouette_widths_kmed_c <- function(k) {
  kmedoid <- pam(complete_dummy_scaled, k) # k-medoid with k centers
  kmedoid$silinfo$avg.width
}

# Calculate average silhouette width for different numbers of clusters
avg_silwidth_kmed_c <- sapply(2:8, silhouette_widths_kmed_c) # 8 = Max reasonable # of customer
groups

# Plot the Silhouette Method
plot(2:8, avg_silwidth_kmed_c, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of clusters k",
     ylab = "Average Silhouette Width",
     main = "Silhouette Scores for K-medoid Clustering - Complete Dataset")
```

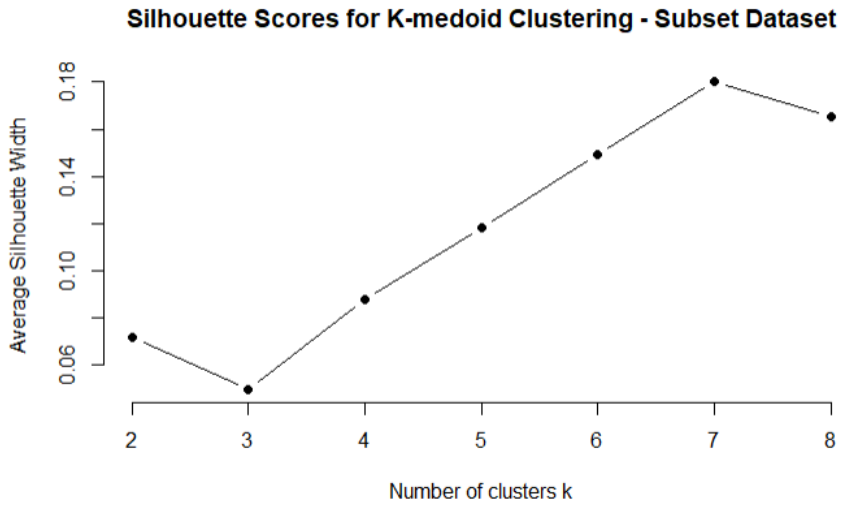


This model is about 0.1 silhouette score, again, not capturing variation at all.

Try k-medoid partition clustering on a subset dataset with dummy variables, using Manhattan distance.

```
# set seed
set.seed(25463)

# define function to calculate silhouette widths
silhouette_widths_kmed_s <- function(k) {
  kmedoid <- pam(subset_dummy_scaled, k) # k-medoid with k centers
  kmedoid$silinfo$avg.width
}

# Calculate average silhouette width for different numbers of clusters
avg_silwidth_kmed_s <- sapply(2:8, silhouette_widths_kmed_s) # 8 = Max reasonable # of groups
```

```r
# Plot the Silhouette Method
plot(2:8, avg_silwidth_kmed_s, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of clusters k",
     ylab = "Average Silhouette Width",
     main = "Silhouette Scores for K-medoid Clustering - Subset Dataset")
```



The elbow is at 0.18 silhouette score for 7 clusters, which is again not capturing variation in the dataset.

DBSCAN Clustering and Random Forest Modeling

Note: All code in this section is written in Python.

For my analysis, I am doing density based clustering utilizing an rbf_kernel distance matrix to capture nonlinear and complex relationships between observations. After clustering, I will investigate the characteristics of the clusters to uncover insights about variables that may contribute to growth or decline of order volumes. I also utilize a random forest model to determine the features that contributed the most to the clusters to offer context to the clustering and potential insights into the clusters' characteristics.

```python
#loading in packages
import pandas as pd
from sklearn.preprocessing import StandardScaler
import numpy as np
from sklearn.cluster import DBSCAN
from scipy.spatial.distance import mahalanobis
from sklearn.metrics import pairwise_distances
from google.colab import drive
```

Cleaning Data Further

Next, I cleaned the data a bit further than the initial EDA by creating dummy variables from our categorical variables so it is compatible with the clustering algorithm. I also scale the numeric features of case_total, fountain_total, and percent_change so that these variables do not overly influence the clustering algorithm.

```
raw_completed_dummies[['CO2_CUSTOMER', 'LOCAL_MARKET_PARTNER']] =
raw_completed_dummies[['CO2_CUSTOMER', 'LOCAL_MARKET_PARTNER']].astype(int)
  except:
    print('segmented data... continuing')
  #drop duplicate column from join
  stg_completed = raw_completed_dummies.drop(['TRADE_CHANNEL.y'],axis=1)
  #replace infinity values with nulls
  stg_completed.replace([np.inf, -np.inf], np.nan, inplace=True)
  # Drop rows with NaN values
  stg_completed = stg_completed.dropna()
  #scale percent change growth for clustering
  scaler = StandardScaler()
  stg_completed['percent_change'] = scaler.fit_transform(stg_completed[['percent_change']])
  stg_completed['fountain_total'] = scaler.fit_transform(stg_completed[['fountain_total']])
  stg_completed['cases_total'] = scaler.fit_transform(stg_completed[['cases_total']])
  return stg_completed

stg_completed=clean_data(raw_completed).drop('fountain_only', axis=1)
stg_complete_segment = clean_data(raw_subset)
```

DBSCAN Clustering

Here, we will create our density based clustering algorithm using an RBF kernel. We use this instead of the traditional euclidean distance since we suspect the data to have non-linear relationship (RBF transforming data into a higher dimensional space to identify clusters with complex boundaries), to capture similarity along with distance (use of the similarity score in computing distance), and because RBF is better suited for high dimensional data, which we have with our ~90 dummy variables, due to its ability to focus on local relationships. RBF is also more robust to noise, which we seem to have plenty of within our dataset.

```
def clusterData(gamma:float, df:pd.DataFrame, minsamps:int, eps:float):
  initial_df = df
  #get array of feature values
  X = initial_df.values
  from sklearn.metrics.pairwise import rbf_kernel

  # Compute the RBF kernel similarity matrix
  similarity_matrix = rbf_kernel(X, gamma=0.1)

  # Convert similarity to distance (1 - similarity)
  distance_matrix = 1 - similarity_matrix
  #played around with epsilion and min samples, .3 and 200 seemed to balance groups the best
  dbscan = DBSCAN(eps=eps, min_samples=minsamps, metric='precomputed')
  #create cluster labels
  labels = dbscan.fit_predict(distance_matrix)
  initial_df['Cluster'] = labels
  clustered_data = initial_df.copy(deep=True)
  #output counts per cluster
  print(clustered_data['Cluster'].value_counts())
  #remove unclassified
  clustered_cleaned = clustered_data[clustered_data['Cluster'] != -1]
```

```
  return clustered_cleaned

clustered_completed = clusterData(0.1, stg_completed, 200, .3)
```

```
Cluster
 1     3526
 8     2729
-1     1648
 9     1424
 3     1375
 0     1169
 6      898
 2      876
 7      813
 4      682
 5      480
 12     404
 14     346
 11     311
 10     291
 13     270
Name: count, dtype: int64
```

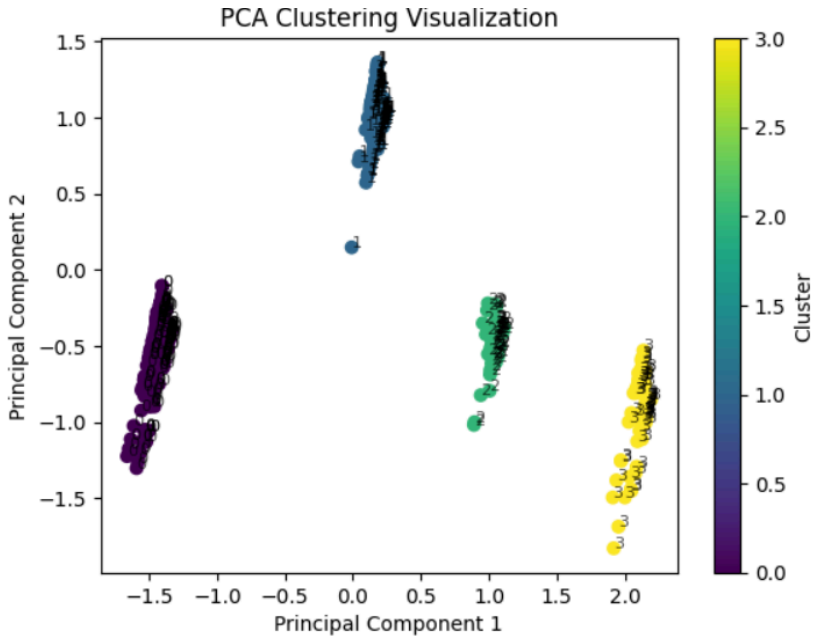
```
clustered_subset = clusterData(.1, stg_complete_segment, 20, .2)
```

```
Cluster
 1    166
 0    153
-1    145
 3     62
 2     58
Name: count, dtype: int64
```

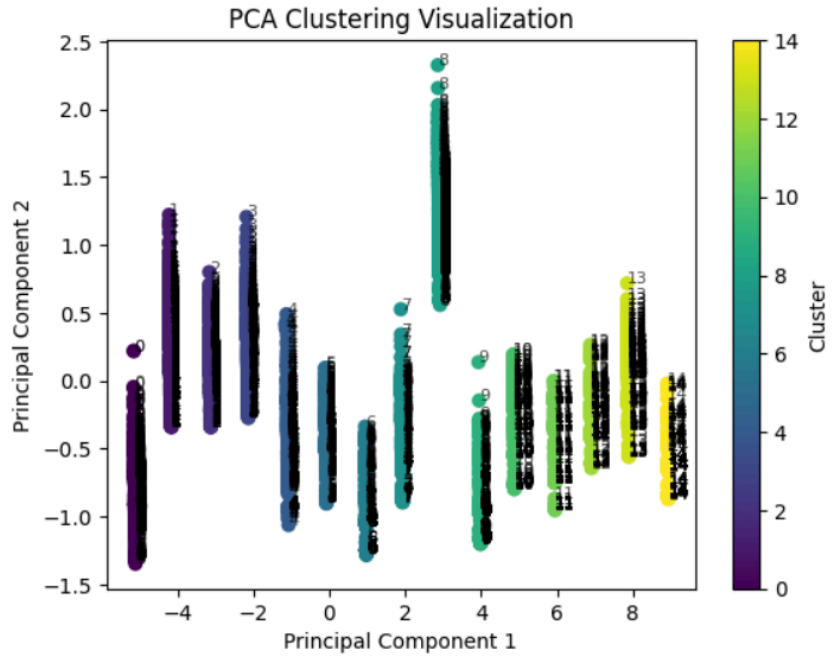
PCA Plotting of Clusters

Now, we will reduce the feature dimensionality using PCA to identify how the clusters are formed in relation to each other. Here in the chart, we are looking to see clusters spaced out from each other, which indicates that each cluster possesses unique characteristics.

```
clustered_subset = clusterData(.1, stg_complete_segment, 20, .2)
```




```
plot_PCA(clustered_completed)
```



The PCA chart reveals distinct groupings across the principal components in both the segmented and completed datasets. This is a positive indication, as it suggests clear evidence that the clusters are well-defined and separate from one another. Now, let's take a look at some of the summary statistics of each.

Random Forest: Variable Contribution to Clusters

Next, we run a random forest classifier to determine which features seemed to contribute the most to the forming of the clusters.

```
from sklearn.ensemble import RandomForestClassifier

def get_feature_importance(df: pd.DataFrame):
  # Separate features and cluster labels
  X = df.drop('Cluster', axis=1)
  y = df['Cluster']

  # Train a random forest
  rf = RandomForestClassifier(random_state=42)
  rf.fit(X, y)

  # Get feature importances
  feature_importances = pd.Series(rf.feature_importances_, index=X.columns)
  print(feature_importances.sort_values(ascending=False).iloc[0:5])
  return feature_importances

complete_feature_importance = get_feature_importance(clustered_completed)
```

```
TRADE_CHANNEL.x_FAST CASUAL DINING      0.094742
TRADE_CHANNEL.x_COMPREHENSIVE DINING    0.084353
SUB_TRADE_CHANNEL_FSR - MISC            0.074269
COLD_DRINK_CHANNEL_DINING               0.068318
TRADE_CHANNEL.x_GENERAL RETAILER        0.048437
dtype: float64
```

```
subset_feature_importance = get_feature_importance(clustered_subset)
```

```
TRADE_CHANNEL.x_LICENSED HOSPITALITY           0.174963
TRADE_CHANNEL.x_COMPREHENSIVE DINING           0.161862
SUB_TRADE_CHANNEL_FSR - MISC                   0.152927
SUB_TRADE_CHANNEL_OTHER LICENSED HOSPITALITY   0.143088
TRADE_CHANNEL.x_OTHER DINING & BEVERAGE        0.106045
dtype: float64
```

From this analysis, we identify the five most influential features in determining clusters for the complete dataset: the trade channels of fast casual dining, comprehensive dining, and general retail. Additionally, clustering was significantly impacted by the sub-trade channel and the cold drink channel of dining.

For the subset group, clustering appears to be driven by the trade channels of licensed hospitality, comprehensive dining, other dining and beverage, as well as the sub-trade channels for miscellaneous and other licensed hospitality.

We will save the top 10 characteristics to an object for further exploration of their distribution across clusters.

```
top10characteristics_subset =
subset_feature_importance.sort_values(ascending=False).iloc[0:10].index.tolist()
top10characteristics_complete =
complete_feature_importance.sort_values(ascending=False).iloc[0:10].index.tolist()
```

Investigating 10 largest clusters

In the code below, we create a cluster profile for each of the datasets. This profile simply shows the mean values of each of the characteristics. This can be utilized to deep dive how a cluster compares on certain features compared to other clusters.

```
def cluster_profiles(df: pd.DataFrame):
  # Count the number of points in each cluster
  cluster_sizes = df['Cluster'].value_counts()

  # Select the top 10 largest clusters
  largest_clusters = cluster_sizes.head(10).index
  print("Largest clusters:", largest_clusters)
  # Filter the DataFrame for the largest 10 clusters
  largest_clusters_data = df[df['Cluster'].isin(largest_clusters)]
  cluster_profiles = top10characteristics_subset =
subset_feature_importance.sort_values(ascending=False).iloc[0:10].index.tolist()
top10characteristics_complete =
complete_feature_importance.sort_values(ascending=False).iloc[0:10].index.tolist().groupby('Clus
ter').agg(['mean'])
  return cluster_profiles

subset_profiles = cluster_profiles(clustered_subset)
subset_profiles
```

	percent_change	cases_total	fountain_total	growth_category_declined	growth_category_grew	growth_category_stayed_same	TRADE_CHANNEL.x_ACCOMMODAT
	mean	mean	mean	mean	mean	mean	n
Cluster							
0	-0.171615	0.0	0.115600	0.581699	0.398693	0.019608	
1	-0.131268	0.0	-0.269973	0.596386	0.385542	0.018072	
2	-0.313498	0.0	-0.206973	0.827586	0.120690	0.051724	
3	-0.059286	0.0	0.026702	0.548387	0.419355	0.032258	

4 rows × 67 columns

* See ipynb “Dan_Modeling” for full chart

```
complete_profiles = cluster_profiles(clustered_completed)
complete_profiles
```

```
Largest clusters: Index([1, 8, 9, 3, 0, 6, 2, 7, 4, 5], dtype='int64', name='Cluster')
```

	percent_change	CO2_CUSTOMER	LOCAL_MARKET_PARTNER	cases_total	fountain_total	growth_category_declined	growth_category_grew	growth_category_stayed_same	TRADE_CHANNEL.x_ACADEMIC INSTITUTION	TRAI
	mean	mean	mean	mean	mean	mean	mean	mean	mean	
Cluster										
0	-0.025317	0.439692	0.871685	-0.043222	-0.070740	0.511548	0.471343	0.017109	0.0	
1	-0.018407	0.473057	0.861032	-0.051794	-0.034630	0.585366	0.409529	0.005105	0.0	
2	-0.028961	0.530822	0.985160	-0.111778	-0.060006	0.536530	0.454338	0.009132	0.0	
3	0.009179	0.548364	0.941091	-0.095899	0.015984	0.531636	0.459636	0.008727	0.0	
4	-0.020766	0.376833	0.521994	0.193449	-0.071601	0.466276	0.530792	0.002933	0.0	
5	-0.036570	0.625000	0.966667	-0.093454	-0.128454	0.552083	0.441667	0.006250	0.0	
6	-0.034192	0.091314	0.996659	-0.098700	-0.257324	0.503341	0.488864	0.007795	0.0	
7	-0.030031	0.318573	0.772448	-0.053771	-0.172544	0.578106	0.414514	0.007380	0.0	
8	-0.030485	0.652254	0.904727	-0.103452	0.131481	0.562111	0.430561	0.007329	0.0	
9	-0.032312	0.045646	0.996489	-0.018357	-0.271799	0.577949	0.416433	0.005618	0.0	

10 rows × 94 columns

* See ipynb “Dan_Modeling” for full chart

For this analysis, though, we will just look at which clusters had the highest means for each of the top 10 most "important" characteristics from our random forest model earlier.

```
# Example: Find the highest mean value for each feature across the top clusters
def get_highest_means(df: pd.DataFrame, top10characteristics:list):
  dominant_characteristics = df.idxmax(axis=0)  # Identify clusters with max mean for each
feature

  pd.set_option('display.max_rows', None)  # Show all rows
  pd.set_option('display.max_columns', None) #show all columns
  summary_clusters =
pd.DataFrame(dominant_characteristics).reset_index(drop=False).rename({'level_0':
'Variable','level_1': 'summary Stat', 0:'Cluster with Highest Mean:'}, axis=1)
  summary_clusters_top_10 =
summary_clusters[summary_clusters['Variable'].isin(top10characteristics)]
  return summary_clusters, summary_clusters_top_10

top10characteristics_subset.append('percent_change')

top10characteristics_subset.append('growth_category_declined')

subset_all_summary, subset_top10_summary = get_highest_means(subset_profiles,
top10characteristics_subset)
subset_top10_summary.sort_values(by='Cluster with Highest Mean:')
```

	Variable	summary Stat	Cluster with Highest Mean:
2	fountain_total	mean	0
9	TRADE_CHANNEL.x_COMPREHENSIVE DINING	mean	0
39	SUB_TRADE_CHANNEL_FSR - MISC	mean	0
16	TRADE_CHANNEL.x_LICENSED HOSPITALITY	mean	1
52	SUB_TRADE_CHANNEL_OTHER LICENSED HOSPITALITY	mean	1
11	TRADE_CHANNEL.x_FAST CASUAL DINING	mean	2
41	SUB_TRADE_CHANNEL_MEXICAN FAST FOOD	mean	2
3	growth_category_declined	mean	2
46	SUB_TRADE_CHANNEL_OTHER FAST FOOD	mean	2
0	percent_change	mean	3
18	TRADE_CHANNEL.x_OTHER DINING & BEVERAGE	mean	3
45	SUB_TRADE_CHANNEL_OTHER DINING	mean	3

In the subsetted dataset, Group 0 had the highest means for fountain total, comprehensive dining, and FSR-misc. Another notable cluster was Cluster 2, which had the highest means for fast food groups but also the most observations experiencing a decline in growth. Cluster 3 had the highest mean for percent change, with the largest proportions of dining and beverage, as well as other dining.

```
top10characteristics_complete.append('percent_change')

top10characteristics_complete.append('growth_category_declined')

complete_all_summary, complete_top_10_summary = get_highest_means(complete_profiles,
top10characteristics_complete)
complete_top_10_summary.sort_values(by='Cluster with Highest Mean:')
```

	Variable	summary Stat	Cluster with Highest Mean:
75	SUB_TRADE_CHANNEL_OTHER OUTDOOR ACTIVITIES	mean	0
5	growth_category_declined	mean	1
36	COLD_DRINK_CHANNEL_DINING	mean	1
15	TRADE_CHANNEL.x_FAST CASUAL DINING	mean	1
21	TRADE_CHANNEL.x_LICENSED HOSPITALITY	mean	2
23	TRADE_CHANNEL.x_OTHER DINING & BEVERAGE	mean	3
66	SUB_TRADE_CHANNEL_OTHER DINING	mean	3
0	percent_change	mean	3
38	COLD_DRINK_CHANNEL_GOODS	mean	6
12	TRADE_CHANNEL.x_COMPREHENSIVE DINING	mean	8
53	SUB_TRADE_CHANNEL_FSR - MISC	mean	8
17	TRADE_CHANNEL.x_GENERAL RETAILER	mean	9

In the complete dataset, Groups 1, 8, and 3 exhibited the most distinct characteristics. Group 3 stood out as the dominant group for other dining and other dining and beverage, while also having the highest mean

percent change in growth. Group 1 had the highest means for dining and fast casual dining but also the largest proportion of observations experiencing declining growth. This may further suggest that fast food chains require additional support, whereas other dining categories may be growing at a faster rate.

Next, we deep dive the clusters containing the highest proportion of declining-growth customers and those containing the highest-growing customers in each of the datasets.

Deep Diving Clusters

First we will look at the fast food/declining group clusters in both of our datasets.

Declining Cluster

```
subset_all_summary[subset_all_summary['Cluster with Highest Mean:'] == 2]
```

	Variable	summary Stat	Cluster with Highest Mean:
3	growth_category_declined	mean	2
5	growth_category_stayed_same	mean	2
11	TRADE_CHANNEL.x_FAST CASUAL DINING	mean	2
34	SUB_TRADE_CHANNEL_ASIAN FAST FOOD	mean	2
41	SUB_TRADE_CHANNEL_MEXICAN FAST FOOD	mean	2
46	SUB_TRADE_CHANNEL_OTHER FAST FOOD	mean	2
59	SUB_TRADE_CHANNEL_PIZZA FAST FOOD	mean	2
61	SUB_TRADE_CHANNEL_SANDWICH FAST FOOD	mean	2
63	FREQUENT_ORDER_TYPE_MYCOKE LEGACY	mean	2
65	FREQUENT_ORDER_TYPE_OTHER	mean	2

Cluster 2 in the subsetted data shows the highest proportions of customer growth staying the same and declining of all the clusters. Additionally, it contained all the fast food groups for trade channels. Additionally, it also contained the highest proportions of MYCOKE legacy and other frequent order types of the clusters.

Let's compare it to the overall data for a similar cluster, 1.

```
complete_all_summary[complete_all_summary['Cluster with Highest Mean:'] == 1]
```

	Variable	summary Stat	Cluster with Highest Mean:
5	growth_category_declined	mean	1
15	TRADE_CHANNEL.x_FAST CASUAL DINING	mean	1
36	COLD_DRINK_CHANNEL_DINING	mean	1
42	SUB_TRADE_CHANNEL_ASIAN FAST FOOD	mean	1
45	SUB_TRADE_CHANNEL_BURGER FAST FOOD	mean	1
47	SUB_TRADE_CHANNEL_CHICKEN FAST FOOD	mean	1
58	SUB_TRADE_CHANNEL_MEXICAN FAST FOOD	mean	1
67	SUB_TRADE_CHANNEL_OTHER FAST FOOD	mean	1
81	SUB_TRADE_CHANNEL_PIZZA FAST FOOD	mean	1
87	SUB_TRADE_CHANNEL_SANDWICH FAST FOOD	mean	1
90	FREQUENT_ORDER_TYPE_MYCOKE LEGACY	mean	1

Again, we see many of the same characteristics, with cluster 1 having the highest proportions of fast food trade channels and using Mycoke legacy. This could offer evidence that fast food chains may be declining in overall ordered gallons, potentially due to them using legacy systems.

Let's examine our high growth clusters in each of the datasets to determine if there are characteristics shared by high performing clusters.

Rising Cluster

Finally, we will dive into the clusters with the highest proportions of increasing growth.

```
subset_all_summary[subset_all_summary['Cluster with Highest Mean:'] == 3]
```

	Variable	summary Stat	Cluster with Highest Mean:
0	percent_change	mean	3
4	growth_category_grew	mean	3
18	TRADE_CHANNEL.x_OTHER DINING & BEVERAGE	mean	3
45	SUB_TRADE_CHANNEL_OTHER DINING	mean	3
64	FREQUENT_ORDER_TYPE_MYCOKE360	mean	3

In the subset dataset for the cluster with the highest mean for percent change, we see it also has the highest mean for the binary variable of growth_category_grew, indicating that the clustering algorithm is picking up on the correlation between those two variables. Additionally, we also see the other dining channels from the top 10 features analysis. Most interestingly, however, is the cluster containing the highest proportion of Mycoke360. Seeing as the declining growth cluster had the highest proportion of Mycoke360 legacy versions, this could offer insight that mycoke360 upgrades could improve growth as high performers seem to be using it.

Finally, let's look at the overall data's cluster with the highest mean growth, which was also cluster 3.

```
complete_all_summary[complete_all_summary['Cluster with Highest Mean:'] == 3]
```

	Variable	summary Stat	Cluster with Highest Mean:
0	percent_change	mean	3
23	TRADE_CHANNEL.x_OTHER DINING & BEVERAGE	mean	3
66	SUB_TRADE_CHANNEL_OTHER DINING	mean	3

Here, we observe that Cluster 3 was the highest-performing group only for the other channels and did not have the highest proportion of MyCoke360, unlike in the subsetted data. Before concluding our cluster analysis, we will examine whether the group with MyCoke360 in the overall dataset tended to outperform the average in terms of growth.

```
complete_all_summary[complete_all_summary['Variable'] == 'FREQUENT_ORDER_TYPE_MYCOKE360']
```

	Variable	summary Stat	Cluster with Highest Mean:
91	FREQUENT_ORDER_TYPE_MYCOKE360	mean	5

```
#join complete profiles with mean of means with 5's means for each variable
five_v_all = pd.concat([complete_profiles.mean(), complete_profiles.loc[5].sort_values()],
axis=1)
#make delta column
complete_all_summary[complete_all_summary['Variable'] ==
'FREQUENT_ORDER_TYPE_MYCOKE360']five_v_all['delta'] = five_v_all[5] - five_v_all[0]
#filter to delta values that are not 0 and sort
five_v_all[five_v_all['delta'] != 0].sort_values(by='delta', key=abs,
ascending=False).rename({0:'Mean of Means of all Other Clusters', 5:'Cluster 5 mean'}, axis=1)
```

		Mean of Means of all Other Clusters	Cluster 5 mean	delta
TRADE_CHANNEL.x_PROFESSIONAL SERVICES	mean	0.100000	1.000000	0.900000
COLD_DRINK_CHANNEL_WORKPLACE	mean	0.100000	1.000000	0.900000
SUB_TRADE_CHANNEL_OTHER PROFESSIONAL SERVICES	mean	0.100000	1.000000	0.900000
COLD_DRINK_CHANNEL_DINING	mean	0.400611	0.000000	-0.400611
FREQUENT_ORDER_TYPE_SALES REP	mean	0.693944	0.352083	-0.341861
FREQUENT_ORDER_TYPE_OTHER	mean	0.182965	0.447917	0.264951
CO2_CUSTOMER	mean	0.410155	0.625000	0.214845
COLD_DRINK_CHANNEL_GOODS	mean	0.200000	0.000000	-0.200000
TRADE_CHANNEL.x_OUTDOOR ACTIVITIES	mean	0.100000	0.000000	-0.100000
TRADE_CHANNEL.x_ACCOMMODATION	mean	0.100000	0.000000	-0.100000
TRADE_CHANNEL.x_VEHICLE CARE	mean	0.100000	0.000000	-0.100000
SUB_TRADE_CHANNEL_COMPREHENSIVE PROVIDER	mean	0.100000	0.000000	-0.100000
TRADE_CHANNEL.x_OTHER DINING & BEVERAGE	mean	0.100000	0.000000	-0.100000
TRADE_CHANNEL.x_COMPREHENSIVE DINING	mean	0.100000	0.000000	-0.100000
TRADE_CHANNEL.x_GENERAL	mean	0.100000	0.000000	-0.100000
growth_category_declined	mean	0.540495	0.552083	0.011589
FREQUENT_ORDER_TYPE_MYCOKE360	mean	0.101084	0.175000	0.073916

*for full chart, see Dan_Modeling.ipynb

Examining the characteristics of Cluster 5, we find that it had a slightly higher mean of declining customers compared to the average cluster. While MyCoke360 may support growth in local fountain customers, this effect may not extend to the overall population.

Model Results

XG Boost Results

For both the complete and subset, the top 4 influential features are: FIRST_DELIVERY_DATE, TOTAL_ORDERED_VOLUME, while SUB_TRADE_CHANNEL and ZIP_CODE.

Using the target variable of “growth” or “no growth”, there was a strong target imbalance toward non-growth. The precision of the XG Boost models for both customer groups was good, but the recall was very low. This means that customers that will grow may be predicted to not grow using these models, so Swire Coca-Cola would have missed opportunities if these incorrect predictions led them to move these customers to white truck delivery.

DBSCAN Clustering Results

From the clustering analysis, we found that in the local market partner and fountain drink only subset, the cluster with the highest growth contained the highest proportions of MyCoke360 users in the other channels.

Whereas the cluster with the highest proportion of declining growth also had the highest proportions of fast food trade channels and the use of MyCoke360 - Legacy.

In the overall population, we saw that the cluster with the highest average growth also had the highest proportions of other trade channels as well, but did not use MyCoke360 more than other clusters. When investigating the group that did have the highest proportion of MyCoke360 order types, it did not seem to exhibit as much growth as the other clusters. This may offer evidence that upgrading MyCoke360 may be worthwhile in the subset, but may not have as drastic of an effect in the general population of customers.

Additionally, we found that both the subsetted and complete datasets seemed to cluster the "OTHER" trade channels in the highest-mean-growth cluster, which may indicate promise for these companies for growing order volumes. Additionally, both datasets showed the cluster with the highest proportion of declining-order-volume customers also contained the highest proportions of many of the "FAST FOOD" order channels. This could indicate that the fast food channels may be declining in order volume and could benefit through coaching or upgrading their ordering systems, as suggested by the subsetted analysis showing a distinct difference in growth in the clusters most utilizing MyCoke360 vs. MyCoke360 Legacy.

Non-informative Models

The models that did not have useful results were: logistic regression, lasso regression, ridge regression, random forest, causal forest, k-means clustering, and k-medoid clustering.

Group Member Contribution

Andy Spendlove
- Logistic Regression
- Lasso & Ridge Regressions
- Model results summary & final review

Dan Powell
- DBSCAN and PCA analysis of clusters
- Random Forest for Cluster feature importance
- Model results summary & final review

Jessica Kersey
- K-means & K-medoid clustering
- Formatting Google Doc
- Model results summary & final review

Melissa Messervy
- Random Forest model
- Causal Forest model
- Model results summary & final review

Tommaso Pascucci
- XG Boost models
- Model results summary & final review