# E-Commerce Customer Churn Analysis and Segmentation

**Author:** Hasan Raja Khan - Data Analyst

**Purpose:**

- This script performs comprehensive analysis of e-commerce customer data to predict churn, segment customers using RFM (Recency, Frequency, Monetary) analysis and recommend targeted treatments.
- It includes exploratory data analysis, machine learning modeling, clustering and ROI calculations.

## Importing Libraries

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, classification_report, confusi
from sklearn.model_selection import train_test_split, cross_val_score, GridS
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
import plotly.express as px
import shap
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# Set random seed for reproducibility
np.random.seed(42)
```

## Loading Dataset

In [ ]:
```
df = pd.read_excel("/content/E Commerce Dataset.xlsx", sheet_name=1)
```

In [ ]:
```
df.head(10)
```

## Exploratory Data Analysis (EDA)

In [ ]:
```python
df.describe()
```

In [ ]:
```python
df.info()
```

In [ ]:
```python
df.isnull().sum()
```

In [ ]:
```python
# Visualizes missing values to identify gaps

plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()
```

In [ ]:
```python
# Numerical Features Distribution
# Plotting boxplots for numerical columns to check outliers

numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(4, 5, i)
    sns.boxplot(y=df[col])
    plt.title(col)
plt.tight_layout()
plt.show()
```

In [ ]:
```python
# Categorical Features vs Churn
# Plotting bar charts for categorical features vs churn

categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    plt.figure(figsize=(8, 4))
    sns.countplot(x=col, hue='Churn', data=df)
    plt.title(f'{col} vs Churn')
    plt.xticks(rotation=45)
    plt.show()
```

In [ ]:
```python
# Correlation Matrix
# Shows correlations between numerical features

plt.figure(figsize=(12, 8))
sns.heatmap(df[numerical_cols].corr(), annot=True, cmap='coolwarm', fmt='.2f
plt.title('Correlation Matrix')
plt.show()
```

Churn has a negative correlation with **Tenure** (-0.35) and **CashbackAmount** (-0.15) → longer-tenure and higher-cashback users are less likely to churn.

**OrderCount** and **CouponUsed** are highly correlated (0.75) which means frequent buyers tend to use more coupons.

Most features show low correlation with each other, suggesting minimal multicollinearity.

In [ ]:
```
# Churn Distribution
# Plotting pie chart

churn_counts = df['Churn'].value_counts()
explode = [0.05] * len(churn_counts)

plt.figure(figsize=(6, 4))
plt.pie(churn_counts, labels=churn_counts.index, autopct='%1.1f%%', startang
plt.title('Churn Distribution')
plt.show()
```

In [ ]:
```
# Checking unique values in each column

for col in df.columns:
    print(f"\nColumn: {col}")
    print("Unique values:", df[col].unique())
```

## Data Preprocessing

In [ ]:
```
# Dropping CustomerID and standardizing categorical values

df_clean = df.drop(columns=['CustomerID'])
df_clean['PreferredLoginDevice'] = df_clean['PreferredLoginDevice'].replace(
df_clean['PreferredPaymentMode'] = df_clean['PreferredPaymentMode'].replace(
df_clean['PreferedOrderCat'] = df_clean['PreferedOrderCat'].replace({'Mobile
```

In [ ]:
```
df_clean
```

In [ ]:
```
#Feature Engineering
#Creates new features: AvgOrderValue, RecentComplaint

df_clean['AvgOrderValue'] = df_clean['CashbackAmount'] / df_clean['OrderCour
df_clean['RecentComplaint'] = (df_clean['Complain'] == 1).astype(int)
df_clean[['AvgOrderValue', 'RecentComplaint']].head()
```

In [ ]:
```
# Removing duplicate rows

print(f"Number of duplicate rows: {df_clean.duplicated().sum()}")
df_clean = df_clean.drop_duplicates()
print(f"Number of duplicate rows after removal: {df_clean.duplicated().sum()
```

In [ ]:
```
# Handling Missing Values
# Imputes missing values (IterativeImputer for numerical, median for HourSpe

missing_cols = df_clean.columns[df_clean.isnull().mean() > 0]
print(f"Columns with missing values: {missing_cols}")
if len(missing_cols) > 0:
    iter_cols = [col for col in missing_cols if col != 'HourSpendOnApp' and
    if iter_cols:
        iter_imputer = IterativeImputer(random_state=42)
```

```
        df_clean[iter_cols] = iter_imputer.fit_transform(df_clean[iter_cols]
    if 'HourSpendOnApp' in missing_cols:
        simple_imputer = SimpleImputer(strategy='median')
        df_clean['HourSpendOnApp'] = simple_imputer.fit_transform(df_clean[[
print(f"\nRemaining missing values: {df_clean.isnull().sum().sum()}")
```

Missing values were found in 8 columns. Numerical columns were filled using Iterative Imputer and HourSpendOnApp was filled with the median. After this step, all missing values were successfully handled.

### Churn Prediction

In [ ]:
```
# Preparing Data
# Splits features and target, encodes categoricals, splits train/test

X = df_clean.drop(columns=['Churn'])
y = df_clean['Churn']
X = pd.get_dummies(X, columns=X.select_dtypes(include=['object']).columns, c
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")
```

In [ ]:
```
# Training Logistic Regression, Decision Tree, XGBoost, SVM
# Initializing models

lr = LogisticRegression(max_iter=1000, random_state=42)
dt = DecisionTreeClassifier(random_state=42)
xgb = XGBClassifier(random_state=42, eval_metric='logloss')
svm = SVC(probability=True, random_state=42)
models = {'Logistic Regression': lr, 'Decision Tree': dt, 'XGBoost': xgb, 'S
```

In [ ]:
```
# Dictionary to store precision scores
precision_scores = {}
```

In [ ]:
```
# Training and evaluating each model

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    precision = precision_score(y_test, y_pred)
    precision_scores[name] = precision
    print(f"\n{name} Classification Report:\n", classification_report(y_test
    print(f"{name} Precision (Churners): {precision:.2f}")
    print(f"{name} Recall (Churners): {recall_score(y_test, y_pred):.2f}")
```

All four models were trained and evaluated based on their ability to detect churners (class 1).

Here is how they performed:

- **XGBoost** performed the best with 92% precision and 89% recall for churners, making it the most reliable model.

- **Decision Tree** also performed well, with 81% precision and 91% recall.
- **Logistic Regression** gave decent results but had lower recall (58%), meaning it missed many churners.
- **SVM** failed to detect any churners, with 0% precision and recall, making it unsuitable for this task.

**Conclusion:** XGBoost is the most effective model for identifying churners in this dataset.

In [ ]:
```
# Selecting best model based on precision

best_model_name = max(precision_scores, key=precision_scores.get)
best_precision = precision_scores[best_model_name]
print(f"\nBest Model: {best_model_name} with Precision: {best_precision:.2f}
```

In [ ]:
```
# Optimizing XGBoost with Grid Search

param_grid = {'learning_rate': [0.01, 0.1], 'max_depth': [3, 5], 'n_estimatc
xgb = XGBClassifier(random_state=42, eval_metric='logloss')
grid_search = GridSearchCV(xgb, param_grid, cv=5, scoring='f1')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
print("Best XGBoost Parameters:", grid_search.best_params_)
```

In [ ]:
```
# Cross-Validation for Best Model
# Validates best XGBoost model

cv_scores = cross_val_score(best_model, X, y, cv=5, scoring='f1')
print(f"XGBoost Mean F1 Score (CV): {cv_scores.mean():.2f} ± {cv_scores.std(
```

The optimized XGBoost model was validated using 5-fold cross-validation.

It achieved a mean F1 score of 0.91 ± 0.02, indicating consistently high performance with low variance across different data splits.

In [ ]:
```
# Evaluating Best Model
# Evaluates best XGBoost model

y_pred = best_model.predict(X_test)
print("XGBoost Classification Report:\n", classification_report(y_test, y_pr
print(f"XGBoost Precision (Churners): {precision_score(y_test, y_pred):.2f}"
print(f"XGBoost Recall (Churners): {recall_score(y_test, y_pred):.2f}")
```

In [ ]:
```
# Visualising Confusion Matrix

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Churned', 'Churned'],
            yticklabels=['Not Churned', 'Churned'])
```

```
plt.title('Confusion Matrix for Churn Prediction')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.tight_layout()
plt.show()
```

In [ ]:
```
# ROI Calculation
# Calculates ROI ($500 saved per retained customer, $100 intervention cost,

cm = confusion_matrix(y_test, y_pred)
true_positives = cm[1, 1]
false_positives = cm[0, 1]
roi = (true_positives * 500 - (true_positives + false_positives) * 100) / ((
print(f"XGBoost ROI: {roi:.2%}")
```

The estimated ROI is 351.47%, meaning every 100 dollars spent could return over 350 dollars in value, making the model highly cost-effective for targeted retention efforts.

In [ ]:
```
# SHAP Feature Importance
# Visualizes feature importance for XGBoost

explainer = shap.TreeExplainer(best_model)
shap_values = explainer.shap_values(X_test)
plt.figure()
shap.summary_plot(shap_values, X_test, show=False)
plt.title('SHAP Feature Importance for XGBoost')
plt.show()
```

It highlights the most impactful features influencing XGBoost's churn predictions:

- **Tenure**, **Complain** and **NumberOfAddress** had the highest impact.
- Features like **CashbackAmount**, **SatisfactionScore** and **AvgOrderValue** also significantly influenced predictions.

The color shows feature value (red = high, blue = low) and position indicates how much it pushes the prediction toward churn or not churn.

### Customer Segmentation

In [ ]:
```
# Preparing RFM Data
# Extracts Recency, Frequency, Monetary

df_rfm = df_clean[['DaySinceLastOrder', 'OrderCount', 'CashbackAmount']].cop
df_rfm.columns = ['Recency', 'Frequency', 'Monetary']
df_rfm.head()
```

In [ ]:
```
# K-Means Clustering
# Scaling data and evaluating K-Means

scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(df_rfm)
```

```python
k_range = range(2, 9)
kmeans_scores = []
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(rfm_scaled)
    score = silhouette_score(rfm_scaled, labels)
    kmeans_scores.append(score)
plt.figure(figsize=(8, 4))
plt.plot(range(2, 9), kmeans_scores, marker='o')
plt.title('K-Means Silhouette Scores')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.show()

best_k = k_range[kmeans_scores.index(max(kmeans_scores))]
print(f"Best number of clusters based on silhouette score: {best_k}")

final_kmeans = KMeans(n_clusters=best_k, random_state=42)
df_rfm['KMeans_Cluster'] = final_kmeans.fit_predict(rfm_scaled)
df_rfm.head()
```

In [ ]:
```python
# Gaussian Mixture Model (GMM) Clustering
# Evaluating GMM

gmm_scores = []
k_range = range(2, 9)

for k in k_range:
    gmm = GaussianMixture(n_components=k, random_state=42)
    labels = gmm.fit_predict(rfm_scaled)
    score = silhouette_score(rfm_scaled, labels)
    gmm_scores.append(score)
plt.figure(figsize=(8, 4))
plt.plot(range(2, 9), gmm_scores, marker='o')
plt.title('GMM Silhouette Scores')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.show()

best_k = k_range[gmm_scores.index(max(gmm_scores))]
print(f"Best number of clusters for GMM: {best_k}")

final_gmm = GaussianMixture(n_components=best_k, random_state=42)
df_rfm['GMM_Cluster'] = final_gmm.fit_predict(rfm_scaled)
df_rfm.head()
```

In [ ]:
```python
# Visualizing Clusters
# Ploting 3D scatter plots

fig = px.scatter_3d(df_rfm, x='Recency', y='Frequency', z='Monetary', color=
fig.show()
fig = px.scatter_3d(df_rfm, x='Recency', y='Frequency', z='Monetary', color=
fig.show()
```

In [ ]:
```
# Evaluating Clusters

kMeans_score = silhouette_score(rfm_scaled, df_rfm['KMeans_Cluster'])
print(f"RFM Silhouette Score: {kMeans_score:.3f}")

gmm_score = silhouette_score(rfm_scaled, df_rfm['GMM_Cluster'])
print(f"RFM Silhouette Score: {gmm_score:.3f}")
```

In [ ]:
```
df_rfm
```

In [ ]:
```
# Labelling Segments Based on Cluster Means

cluster_means = df_rfm.groupby('KMeans_Cluster')[['Recency', 'Frequency', 'M
best_cluster = cluster_means['Monetary'].idxmax()
df_rfm['Segment_Label'] = df_rfm['KMeans_Cluster'].apply(lambda x: 'High Val
```

Customers were segmented using K-Means clustering based on Recency, Frequency and Monetary (RFM) values.

- The cluster with the highest Monetary value was labeled as "**High Value**".
- All other clusters were labeled as "**Low Value**".

In [ ]:
```
df_rfm['Segment_Label'].value_counts()
```

In [ ]:
```
# Visualizing Segment Distribution

plt.figure(figsize=(8, 5))
sns.countplot(x='Segment_Label', data=df_rfm)
plt.title('Customer Segment Distribution (High Value vs Low Value)')
plt.show()
```

In [ ]:
```
# Defining RFM Subgroup Labels (within clusters, for analysis only)

def subgroup_label(row):
    if row['Recency'] <= df_rfm['Recency'].quantile(0.25) and \
       row['Frequency'] >= df_rfm['Frequency'].quantile(0.75) and \
       row['Monetary'] >= df_rfm['Monetary'].quantile(0.75):
        return 'Champion'
    elif row['Frequency'] >= df_rfm['Frequency'].quantile(0.75):
        return 'Loyal'
    elif row['Monetary'] >= df_rfm['Monetary'].quantile(0.75):
        return 'Big Spender'
    elif row['Recency'] >= df_rfm['Recency'].quantile(0.75):
        return 'At Risk'
    elif row['Recency'] <= df_rfm['Recency'].quantile(0.25):
        return 'New'
    else:
        return 'Mid-Value'

df_rfm['RFM_Subgroup'] = df_rfm.apply(subgroup_label, axis=1)
```

Within the clusters, customers were further labeled based on RFM quantiles to better understand their behavior:

- **Champion:** Recent, frequent and high spenders
- **Loyal:** Very frequent buyers
- **Big Spender:** High monetary value
- **At Risk:** Haven’t purchased in a while
- **New:** Very recent customers
- **Mid-Value:** Average behavior on all metrics

This adds deeper insight for tailored marketing and retention strategies.

In [ ]:
```
df_rfm['RFM_Subgroup'].value_counts()
```

In [ ]:
```
# Visualizing RFM_Subgroup Distribution

plt.figure(figsize=(8, 5))
order = df_rfm['RFM_Subgroup'].value_counts().index
sns.countplot(x='RFM_Subgroup', data=df_rfm, order=order)
plt.title('RFM Subgroup Distribution')
plt.tight_layout()
plt.show()
```

In [ ]:
```
# Adding Churn Probability from Model

df_rfm['Churn_Probability'] = best_model.predict_proba(X)[:, 1]
df_rfm['Churn_Probability']
```

In [ ]:
```
# Visualizing Churn by Segment

plt.figure(figsize=(8, 5))
sns.boxplot(x='Segment_Label', y='Churn_Probability', data=df_rfm)
plt.title('Churn Probability by Segment')
plt.show()
```

**Low Value** customers show higher churn risk.

**High Value** customers have lower, more stable churn probability, making them worth prioritizing for retention.

In [ ]:
```
# Visualizing Churn Distribution by Segment

plt.figure(figsize=(10, 6))

for label in df_rfm['Segment_Label'].unique():
    subset = df_rfm[df_rfm['Segment_Label'] == label]
    sns.kdeplot(subset['Churn_Probability'], label=label, fill=True, alpha=0

plt.title('Churn Probability Distribution by Segment')
plt.xlabel('Churn Probability')
plt.ylabel('Density')
```

```
plt.legend()
plt.show()
```

In [ ]:
```
# Visualizing Churn by RFM_Subgroup

plt.figure(figsize=(10, 6))
sns.boxplot(x='RFM_Subgroup', y='Churn_Probability', data=df_rfm)

plt.title('Churn Probability by RFM Subgroup (Boxplot)')
plt.ylabel('Churn Probability')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

New customers show the highest churn risk.

Champions and Loyal customers have the lowest churn probability, making them key groups to retain and nurture.

In [ ]:
```
# Visualizing Churn by RFM_Subgroup

plt.figure(figsize=(8, 5))
sns.stripplot(x='RFM_Subgroup', y='Churn_Probability', data=df_rfm, jitter=T
plt.title('Churn Probability by RFM_Subgroup')
plt.show()
```

In [ ]:
```
# Identifying High-Risk Customers by Segment

high_risk = df_rfm[df_rfm['Churn_Probability'] > 0.7]
high_risk[['Segment_Label', 'Churn_Probability']].groupby('Segment_Label').s
```

In [ ]:
```
# Identifying High-Risk Customers by RFM_Subgroup

high_risk = df_rfm[df_rfm['Churn_Probability'] > 0.7]
high_risk[['RFM_Subgroup', 'Churn_Probability']].groupby('RFM_Subgroup').siz
```

In [ ]:
```
# Visualizing High-Risk Customers

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

# Normal churn points (gray)
sns.stripplot(
    x='RFM_Subgroup',
    y='Churn_Probability',
    data=df_rfm[df_rfm['Churn_Probability'] <= 0.7],
    jitter=True,
    alpha=0.5,
    color='gray'
)

# High-risk churn points (red)
sns.stripplot(
    x='RFM_Subgroup',
    y='Churn_Probability',
```

```
    data=df_rfm[df_rfm['Churn_Probability'] > 0.7],
    jitter=True,
    alpha=0.8,
    color='red'
)

plt.title('High-Risk Customers (Red = Churn > 0.7)')
plt.ylabel('Churn Probability')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

In [ ]:
```
# Visualizing High-Risk Customers

high_risk = df_rfm[df_rfm['Churn_Probability'] > 0.7]
high_risk_counts = high_risk['RFM_Subgroup'].value_counts().sort_values(asce

colors = sns.color_palette("Reds_r", len(high_risk_counts))

plt.figure(figsize=(8, 5))
sns.barplot(x=high_risk_counts.index, y=high_risk_counts.values, palette=col
plt.title('High-Risk Customers (Churn > 0.7)')
plt.xlabel('RFM Subgroup')
plt.ylabel('No. of High-Risk Customers')
plt.tight_layout()
plt.show()
```

Most high-risk churners (churn probability > 0.7) are from the **New**, **Loyal** and **Mid-Value** subgroups.

Targeted engagement strategies for these segments can help reduce churn and protect revenue.

In [ ]:
```
# Adding CustomerID from df to df_rfm using index

df_rfm['CustomerID'] = df.loc[df_clean.index, 'CustomerID'].values

# Adding columns from df_clean to df_rfm using index alignment

columns_to_add = ['SatisfactionScore', 'PreferredPaymentMode', 'PreferedOrde
df_rfm[columns_to_add] = df_clean[columns_to_add]
```

In [ ]:
```
# Segment Summary Report

segment_summary = df_rfm.groupby('Segment_Label').agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'Monetary': 'mean',
    'Churn_Probability': 'mean',
    'SatisfactionScore': 'mean',
    'PreferredPaymentMode': lambda x: x.mode()[0],
    'PreferedOrderCat': lambda x: x.mode()[0]
}).round(2)
```

```
segment_summary
```

**High Value** customers shop more often, spend more and have lower churn (12%)

**Low Value** customers are less engaged and show higher churn risk (18%)

Both segments prefer Debit Card payments, but product preferences differ:

- High Value → Laptops & Accessories
- Low Value → Mobile Phones

In [ ]:
```
# RFM_Subgroup Summary Report

RFM_subgroup_summary = df_rfm.groupby('RFM_Subgroup').agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'Monetary': 'mean',
    'Churn_Probability': 'mean',
    'SatisfactionScore': 'mean',
    'PreferredPaymentMode': lambda x: x.mode()[0],
    'PreferedOrderCat': lambda x: x.mode()[0]
}).round(2)

RFM_subgroup_summary
```

**Champion** have the highest satisfaction (3.4), low churn (7%) and high spend.

**New** customers show the highest churn risk (27%) and lowest spend/satisfaction.

**Big Spender** spend the most but buy less frequently.

**Loyal** customers buy often but have a moderate churn risk (14%).

Most subgroups prefer Debit Card, with product preferences varying across segments.

## Targeted Treatment

In [ ]:
```
# Generating placeholder survival data

survival_data = pd.DataFrame({
    'CustomerID': df_rfm['CustomerID'].sample(49, random_state=42).values,
    'Exp_Loss': np.random.uniform(100, 2000, 49),
    'Uplift_Grocery': np.random.uniform(0, 500, 49),
    'Uplift_CreditCard': np.random.uniform(0, 1000, 49),
    'Uplift_DebitCard': np.random.uniform(0, 1000, 49)
})
survival_data.head()
```

This dummy data simulation shows how much money we might lose if a customer leaves **(Exp_Loss)** and how much we could gain by offering deals through Grocery, Credit Card or Debit Card **(Uplift)**. It helps test which offers work best to keep customers.

In [ ]:
```python
# Merging survival data with df_rfm

treatment_df = pd.merge(survival_data, df_rfm, on='CustomerID', how='left')
```

In [ ]:
```python
# Summarizing Treated Segment

treatment_segment_label = treatment_df.groupby('Segment_Label').agg({
    'Exp_Loss': 'sum',
    'Uplift_Grocery': 'sum',
    'Uplift_CreditCard': 'sum',
    'Uplift_DebitCard': 'sum',
    'Recency': 'mean',
    'Frequency': 'mean',
    'Monetary': 'mean'
}).round(2)
treatment_segment_label
```

**Low Value** customers have a higher total expected loss and potential uplift across all offer types.

**High Value** customers have fewer losses but higher spend and engagement (higher frequency & monetary).

In [ ]:
```python
# Summarizing Treated RFM_Subgroup

treatment_RFM_subgroup = treatment_df.groupby('RFM_Subgroup').agg({
    'Exp_Loss': 'sum',
    'Uplift_Grocery': 'sum',
    'Uplift_CreditCard': 'sum',
    'Uplift_DebitCard': 'sum',
    'Recency': 'mean',
    'Frequency': 'mean',
    'Monetary': 'mean'
}).round(2)
treatment_RFM_subgroup
```

**New** and **Mid-Value** customers have the highest expected loss and uplift, making them key targets for retention offers.

**Big Spenders** and **Loyal** customers show good uplift with higher monetary value.

**At Risk** customers have lower frequency but still offer meaningful gains.

## Analysis Completed