```
#!/usr/bin/env python3
"""
Enhanced Chip Analysis Project
Comprehensive analysis of QVI purchase behavior and transaction data
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
import warnings
warnings.filterwarnings('ignore')

# Set up plotting style
plt.style.use('default')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 8)

#
================================================================================
===
# STEP 1: LOAD AND INSPECT THE DATA
#
================================================================================
===

def load_and_inspect_data():
    """Load and inspect the QVI datasets"""

    print("🔍 STEP 1: LOADING AND INSPECTING DATA")
    print("=" * 60)

    # Load purchase behavior data
    print("📊 Loading QVI purchase behavior data...")
    purchase_data = pd.read_csv("QVI_purchase_behaviour.csv")

    # Load transaction data
    print("📊 Loading QVI transaction data...")
    transaction_data = pd.read_excel("QVI_transaction_data.xlsx")

    print(f"✅ Purchase behavior data: {purchase_data.shape}")
    print(f"✅ Transaction data: {transaction_data.shape}")

    # Inspect purchase behavior data
    print("\n📋 Purchase Behavior Data Info:")
    print(f"   • Shape: {purchase_data.shape}")
    print(f"   • Columns: {list(purchase_data.columns)}")
    print(f"   • Missing values: {purchase_data.isnull().sum().sum()}")
    print(f"   • Data types:")
    print(purchase_data.dtypes)

    # Inspect transaction data
```

```
    print("\n📋 Transaction Data Info:")
    print(f"   • Shape: {transaction_data.shape}")
    print(f"   • Columns: {list(transaction_data.columns)}")
    print(f"   • Missing values:
{transaction_data.isnull().sum().sum()}")
    print(f"   • Data types:")
    print(transaction_data.dtypes)

    # Show sample data
    print("\n📋 Sample Purchase Behavior Data:")
    print(purchase_data.head())

    print("\n📋 Sample Transaction Data:")
    print(transaction_data.head())

    return purchase_data, transaction_data

#
==========================================================================
===
# STEP 2: CLEAN THE DATA
#
==========================================================================
===

def clean_data(purchase_data, transaction_data):
    """Clean and preprocess the data"""

    print("\n🧹 STEP 2: CLEANING DATA")
    print("=" * 60)

    # Create copies
    purchase_clean = purchase_data.copy()
    transaction_clean = transaction_data.copy()

    # Clean purchase behavior data
    print("🧹 Cleaning purchase behavior data...")

    # Check for duplicates
    purchase_duplicates = purchase_clean.duplicated().sum()
    if purchase_duplicates > 0:
        print(f"   • Removing {purchase_duplicates} duplicates")
        purchase_clean = purchase_clean.drop_duplicates()

    # Check for missing values
    purchase_missing = purchase_clean.isnull().sum()
    if purchase_missing.sum() > 0:
        print(f"   • Missing values found:
{purchase_missing[purchase_missing > 0]}")
        # For categorical data, we'll fill with mode
        for col in purchase_clean.columns:
            if purchase_clean[col].isnull().sum() > 0:
                if purchase_clean[col].dtype == 'object':
```

```
                    mode_val = purchase_clean[col].mode()[0]
                    purchase_clean[col].fillna(mode_val, inplace=True)
                    print(f"   • Filled missing values in {col} with
mode: {mode_val}")

    # Clean transaction data
    print("🧹 Cleaning transaction data...")

    # Convert DATE to datetime
    if transaction_clean['DATE'].dtype == 'object':
        transaction_clean['DATE'] =
pd.to_datetime(transaction_clean['DATE'])

    # Check for duplicates
    transaction_duplicates = transaction_clean.duplicated().sum()
    if transaction_duplicates > 0:
        print(f"   • Removing {transaction_duplicates} duplicates")
        transaction_clean = transaction_clean.drop_duplicates()

    # Check for missing values
    transaction_missing = transaction_clean.isnull().sum()
    if transaction_missing.sum() > 0:
        print(f"   • Missing values found:
{transaction_missing[transaction_missing > 0]}")
        # For numeric data, we'll fill with median
        for col in transaction_clean.columns:
            if transaction_clean[col].isnull().sum() > 0:
                if transaction_clean[col].dtype in ['int64', 'float64']:
                    median_val = transaction_clean[col].median()
                    transaction_clean[col].fillna(median_val,
inplace=True)
                    print(f"   • Filled missing values in {col} with
median: {median_val}")

    # Remove outliers in sales (transactions with very high values)
    Q1 = transaction_clean['TOT_SALES'].quantile(0.25)
    Q3 = transaction_clean['TOT_SALES'].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = ((transaction_clean['TOT_SALES'] < lower_bound) |
                (transaction_clean['TOT_SALES'] > upper_bound)).sum()
    if outliers > 0:
        print(f"   • Removing {outliers} sales outliers")
        transaction_clean = transaction_clean[
            (transaction_clean['TOT_SALES'] >= lower_bound) &
            (transaction_clean['TOT_SALES'] <= upper_bound)
        ]

    print(f"✅ Clean purchase data: {purchase_clean.shape}")
    print(f"✅ Clean transaction data: {transaction_clean.shape}")

    return purchase_clean, transaction_clean
```

```
#
================================================================================
===
# STEP 3: MERGE DATASETS
#
================================================================================
===

def merge_datasets(purchase_clean, transaction_clean):
    """Merge purchase behavior and transaction data"""

    print("\n🔗 STEP 3: MERGING DATASETS")
    print("=" * 60)

    # Merge on customer ID (LYLTY_CARD_NBR)
    print("🔗 Merging purchase behavior and transaction data...")
    merged_data = transaction_clean.merge(
        purchase_clean,
        on='LYLTY_CARD_NBR',
        how='inner'
    )

    print(f"✅ Merged data shape: {merged_data.shape}")
    print(f"✅ Columns in merged data: {list(merged_data.columns)}")

    # Add derived features
    print("🔗 Adding derived features...")

    # Ensure DATE is datetime
    if merged_data['DATE'].dtype != 'datetime64[ns]':
        merged_data['DATE'] = pd.to_datetime(merged_data['DATE'])

    # Extract date components
    merged_data['YEAR'] = merged_data['DATE'].dt.year
    merged_data['MONTH'] = merged_data['DATE'].dt.month
    merged_data['DAY_OF_WEEK'] = merged_data['DATE'].dt.day_name()
    merged_data['QUARTER'] = merged_data['DATE'].dt.quarter

    # Calculate transaction metrics
    merged_data['SALES_PER_UNIT'] = merged_data['TOT_SALES'] /
merged_data['PROD_QTY']

    # Create customer segments
    merged_data['CUSTOMER_SEGMENT'] = merged_data['LIFESTAGE'] + '_' +
merged_data['PREMIUM_CUSTOMER']

    print("✅ Derived features added:")
    print("   • Date components (YEAR, MONTH, DAY_OF_WEEK, QUARTER)")
    print("   • Sales per unit")
    print("   • Customer segment (LIFESTAGE + PREMIUM_CUSTOMER)")

    # Save merged data
```

```
    merged_data.to_csv("merged_chip_data.csv", index=False)
    print("✅ Merged data saved to: merged_chip_data.csv")

    return merged_data

#
================================================================================
===
# STEP 4: ANALYZE CUSTOMER SEGMENTS
#
================================================================================
===

def analyze_customer_segments(merged_data):
    """Analyze customer segments and their behavior"""

    print("\n📊 STEP 4: ANALYZING CUSTOMER SEGMENTS")
    print("=" * 60)

    # Customer segment analysis
    print("📊 Analyzing customer segments...")

    segment_analysis = merged_data.groupby('CUSTOMER_SEGMENT').agg({
        'LYLTY_CARD_NBR': 'nunique',  # Unique customers
        'TXN_ID': 'count',           # Total transactions
        'TOT_SALES': ['sum', 'mean'], # Total and average sales
        'PROD_QTY': ['sum', 'mean'],  # Total and average quantity
        'SALES_PER_UNIT': 'mean'      # Average price per unit
    }).round(2)

    # Flatten column names
    segment_analysis.columns = [
        'unique_customers', 'total_transactions', 'total_sales',
        'avg_transaction_value', 'total_quantity', 'avg_quantity',
'avg_price_per_unit'
    ]

    # Calculate additional metrics
    segment_analysis['transactions_per_customer'] = (
        segment_analysis['total_transactions'] /
segment_analysis['unique_customers']
    ).round(2)

    segment_analysis['sales_per_customer'] = (
        segment_analysis['total_sales'] /
segment_analysis['unique_customers']
    ).round(2)

    segment_analysis['quantity_per_customer'] = (
        segment_analysis['total_quantity'] /
segment_analysis['unique_customers']
    ).round(2)

    # Sort by total sales
```

```
    segment_analysis = segment_analysis.sort_values('total_sales',
ascending=False)

    print("📊 Customer Segment Summary:")
    print(segment_analysis)

    # Save segment analysis
    segment_analysis.to_csv("segment_summary.csv")
    print("✅ Segment analysis saved to: segment_summary.csv")

    return segment_analysis

#
================================================================================
===
# STEP 5: VISUALIZE INSIGHTS
#
================================================================================
===

def visualize_insights(merged_data, segment_analysis):
    """Create visualizations to understand chip buying patterns"""

    print("\n📈 STEP 5: CREATING VISUALIZATIONS")
    print("=" * 60)

    # Create a comprehensive dashboard
    fig, axes = plt.subplots(2, 3, figsize=(20, 12))
    fig.suptitle('QVI Chip Analysis - Customer Insights Dashboard',
fontsize=16, fontweight='bold')

    # 1. Total Sales by Customer Segment
    print("📊 Creating total sales by segment chart...")
    top_segments = segment_analysis.head(10)  # Top 10 segments
    axes[0,0].barh(range(len(top_segments)), top_segments['total_sales'])
    axes[0,0].set_yticks(range(len(top_segments)))
    axes[0,0].set_yticklabels(top_segments.index, fontsize=8)
    axes[0,0].set_title('Total Sales by Customer Segment')
    axes[0,0].set_xlabel('Total Sales ($)')

    # 2. Customer Distribution by Life Stage
    print("📊 Creating customer distribution by life stage...")
    lifestage_counts = merged_data['LIFESTAGE'].value_counts()
    axes[0,1].pie(lifestage_counts.values, labels=lifestage_counts.index,
autopct='%1.1f%%')
    axes[0,1].set_title('Customer Distribution by Life Stage')

    # 3. Customer Distribution by Premium Status
    print("📊 Creating customer distribution by premium status...")
    premium_counts = merged_data['PREMIUM_CUSTOMER'].value_counts()
    axes[0,2].bar(premium_counts.index, premium_counts.values)
    axes[0,2].set_title('Customer Distribution by Premium Status')
    axes[0,2].tick_params(axis='x', rotation=45)
```

```
    # 4. Sales Distribution
    print("📊 Creating sales distribution...")
    axes[1,0].hist(merged_data['TOT_SALES'], bins=50, alpha=0.7,
edgecolor='black')
    axes[1,0].set_title('Distribution of Transaction Values')
    axes[1,0].set_xlabel('Transaction Value ($)')
    axes[1,0].set_ylabel('Frequency')

    # 5. Sales by Day of Week
    print("📊 Creating sales by day of week...")
    day_sales = merged_data.groupby('DAY_OF_WEEK')['TOT_SALES'].sum()
    day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
    day_sales = day_sales.reindex(day_order)
    axes[1,1].bar(day_sales.index, day_sales.values)
    axes[1,1].set_title('Total Sales by Day of Week')
    axes[1,1].tick_params(axis='x', rotation=45)

    # 6. Average Transaction Value by Segment
    print("📊 Creating average transaction value by segment...")
    top_segments_avg = segment_analysis.head(8)  # Top 8 for better
visibility
    axes[1,2].barh(range(len(top_segments_avg)),
top_segments_avg['avg_transaction_value'])
    axes[1,2].set_yticks(range(len(top_segments_avg)))
    axes[1,2].set_yticklabels(top_segments_avg.index, fontsize=8)
    axes[1,2].set_title('Average Transaction Value by Segment')
    axes[1,2].set_xlabel('Average Transaction Value ($)')

    plt.tight_layout()

    # Save the dashboard
    plt.savefig("total_sales_by_segment.png", dpi=300,
bbox_inches='tight')
    plt.close()

    print("✅ Dashboard saved to: total_sales_by_segment.png")

    # Create additional insights
    print("📊 Creating additional insights...")

    # Product analysis
    product_analysis = merged_data.groupby('PROD_NBR').agg({
        'TXN_ID': 'count',
        'TOT_SALES': 'sum',
        'PROD_QTY': 'sum'
    }).sort_values('TOT_SALES', ascending=False)

    print(f"\n📦 Top 5 Products by Sales:")
    print(product_analysis.head())

    # Store analysis
```

```
    store_analysis = merged_data.groupby('STORE_NBR').agg({
        'TXN_ID': 'count',
        'TOT_SALES': 'sum'
    }).sort_values('TOT_SALES', ascending=False)

    print(f"\n🏪 Top 5 Stores by Sales:")
    print(store_analysis.head())

#
================================================================================
===
# STEP 6: INTERPRET AND RECOMMEND
#
================================================================================
===

def interpret_and_recommend(merged_data, segment_analysis):
    """Interpret findings and provide recommendations"""

    print("\n💡 STEP 6: INTERPRETING FINDINGS AND RECOMMENDATIONS")
    print("=" * 60)

    # Key insights
    total_customers = merged_data['LYLTY_CARD_NBR'].nunique()
    total_transactions = merged_data['TXN_ID'].nunique()
    total_sales = merged_data['TOT_SALES'].sum()
    avg_transaction = merged_data['TOT_SALES'].mean()

    # Top performing segments
    top_segments = segment_analysis.head(5)

    # Customer behavior insights
    customer_frequency =
merged_data.groupby('LYLTY_CARD_NBR')['TXN_ID'].count()
    avg_frequency = customer_frequency.mean()

    # Premium vs non-premium analysis
    premium_analysis = merged_data.groupby('PREMIUM_CUSTOMER').agg({
        'LYLTY_CARD_NBR': 'nunique',
        'TOT_SALES': 'sum',
        'TOT_SALES': 'mean'
    })

    # Life stage analysis
    lifestage_analysis = merged_data.groupby('LIFESTAGE').agg({
        'LYLTY_CARD_NBR': 'nunique',
        'TOT_SALES': 'sum',
        'TOT_SALES': 'mean'
    })

    # Create summary report
    summary = f"""
QVI CHIP ANALYSIS - EXECUTIVE SUMMARY
=====================================
```

```
OVERVIEW:
- Total Customers: {total_customers:,}
- Total Transactions: {total_transactions:,}
- Total Sales: ${total_sales:,.2f}
- Average Transaction Value: ${avg_transaction:.2f}
- Average Transactions per Customer: {avg_frequency:.1f}

TOP CUSTOMER SEGMENTS (by Total Sales):
{top_segments[['total_sales', 'unique_customers',
'avg_transaction_value']].to_string()}

KEY INSIGHTS:

1. WHO BUYS CHIPS:
   - Most active customers: {top_segments.index[0]} with
${top_segments.iloc[0]['total_sales']:,.2f} in sales
   - Customer distribution by life stage:
{dict(lifestage_analysis['LYLTY_CARD_NBR'])}
   - Premium vs non-premium: {dict(premium_analysis['LYLTY_CARD_NBR'])}

2. WHAT DRIVES CHIP SALES:
   - Average transaction value: ${avg_transaction:.2f}
   - Customer frequency: {avg_frequency:.1f} transactions per customer
   - Top performing segments show higher transaction values and frequency

3. TARGETING RECOMMENDATIONS:
   - Focus on {top_segments.index[0]} segment (highest sales)
   - Develop loyalty programs for {top_segments.index[1]} segment (second
highest)
   - Consider premium customer strategies for {top_segments.index[2]}
segment
   - Target marketing campaigns based on life stage preferences

BUSINESS RECOMMENDATIONS:
1. Develop targeted marketing campaigns for top-performing customer
segments
2. Implement loyalty programs to increase customer frequency
3. Consider product bundling strategies for high-value segments
4. Optimize store layouts and product placement based on customer
preferences
5. Develop premium product lines for high-value customer segments
"""

    # Save summary
    with open("initial_findings.txt", "w") as f:
        f.write(summary)

    print("✅ Executive summary saved to: initial_findings.txt")
    print("\n" + summary)

#
==========================================================================
===
```

```
# MAIN ANALYSIS FUNCTION
#
================================================================================
===

def main():
    """Main analysis function"""

    print("🚀 ENHANCED QVI CHIP ANALYSIS PROJECT")
    print("=" * 60)

    # Step 1: Load and inspect data
    purchase_data, transaction_data = load_and_inspect_data()

    # Step 2: Clean data
    purchase_clean, transaction_clean = clean_data(purchase_data,
transaction_data)

    # Step 3: Merge datasets
    merged_data = merge_datasets(purchase_clean, transaction_clean)

    # Step 4: Analyze customer segments
    segment_analysis = analyze_customer_segments(merged_data)

    # Step 5: Visualize insights
    visualize_insights(merged_data, segment_analysis)

    # Step 6: Interpret and recommend
    interpret_and_recommend(merged_data, segment_analysis)

    print("\n" + "=" * 60)
    print("🎉 ANALYSIS COMPLETE!")
    print("=" * 60)
    print("📁 Generated files:")
    print("   • merged_chip_data.csv - Cleaned and merged data")
    print("   • segment_summary.csv - Customer segment metrics")
    print("   • total_sales_by_segment.png - Visualization dashboard")
    print("   • initial_findings.txt - Executive summary")

    return merged_data, segment_analysis

#
================================================================================
===
# RUN THE ANALYSIS
#
================================================================================
===

if __name__ == "__main__":
    merged_data, segment_analysis = main()
```