```
# Quantum Retail Analytics: Trial Store Performance Analysis
# Analysis of trial vs control stores to evaluate new store layouts

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from datetime import datetime
import calendar
from google.colab import files


# Set the style for our visualizations
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("Set2")
pd.set_option('display.max_columns', None)


# Upload the data file
print("Please upload the QVI_data file when prompted.")
uploaded = files.upload()
```

```
Please upload the QVI_data file when prompted.
Choose Files QVI_data.csv
• QVI_data.csv(text/csv) - 29019945 bytes, last modified: 4/20/2025 - 100% done
Saving QVI_data.csv to QVI_data.csv
```

```
# Read the uploaded file
for filename in uploaded.keys():
    df = pd.read_csv("/content/QVI_data.csv")
    print(f"Uploaded {filename}, shape: {df.shape}")
```

```
Uploaded QVI_data.csv, shape: (264834, 12)
```

```
# Display first few rows of the data
print("\nFirst few rows of the data:")
df.head()
```

```
First few rows of the data:
```

	LYLTY_CARD_NBR	DATE	STORE_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE	BRAND	LIFESTAG
0	1000	2018-10-17	1	1	5	Natural Chip Compny SeaSalt175g	2	6.0	175	NATURAL	YOUN SINGLES/COUPLE
1	1002	2018-09-16	1	2	58	Red Rock Deli Chikn&Garlic Aioli 150g	1	2.7	150	RRD	YOUN SINGLES/COUPLE
2	1003	2019-03-07	1	3	52	Grain Waves Sour Cream&Chives 210G	1	3.6	210	GRNWVES	YOUNG FAMILIE
3	1003	2019-03-08	1	4	106	Natural ChipCo Hony Soy Chckn175g	1	3.0	175	NATURAL	YOUNG FAMILIE
4	1004	2018-11-02	1	5	96	WW Original Stacked Chips 160g	1	1.9	160	WOOLWORTHS	OLDE SINGLES/COUPLE

```
# Check data types and missing values
print("\nData information:")
df.info()

print("\nMissing values:")
print(df.isnull().sum())
```

```
Data information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 264834 entries, 0 to 264833
Data columns (total 12 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   LYLTY_CARD_NBR    264834 non-null  int64
 1   DATE              264834 non-null  object
 2   STORE_NBR         264834 non-null  int64
```

```
 3   TXN_ID            264834 non-null  int64
 4   PROD_NBR          264834 non-null  int64
 5   PROD_NAME         264834 non-null  object
 6   PROD_QTY          264834 non-null  int64
 7   TOT_SALES         264834 non-null  float64
 8   PACK_SIZE         264834 non-null  int64
 9   BRAND             264834 non-null  object
 10  LIFESTAGE         264834 non-null  object
 11  PREMIUM_CUSTOMER  264834 non-null  object
dtypes: float64(1), int64(6), object(5)
memory usage: 24.2+ MB

Missing values:
LYLTY_CARD_NBR      0
DATE                0
STORE_NBR           0
TXN_ID              0
PROD_NBR            0
PROD_NAME           0
PROD_QTY            0
TOT_SALES           0
PACK_SIZE           0
BRAND               0
LIFESTAGE           0
PREMIUM_CUSTOMER    0
dtype: int64
```

```
# Convert DATE column to datetime format
df['DATE'] = pd.to_datetime(df['DATE'])


# Extract month and year from DATE
df['MONTH'] = df['DATE'].dt.month
df['YEAR'] = df['DATE'].dt.year
df['MONTH_YEAR'] = df['DATE'].dt.to_period('M')
```

Data preprocessing

```
# Check for and handle outliers, duplicates, etc.
print("\nChecking for duplicate records:")
print(df.duplicated().sum())
```

```
Checking for duplicate records:
1
```

```
# Create monthly metrics for each store
# Group by STORE_NBR and MONTH_YEAR
monthly_metrics = df.groupby(['STORE_NBR', 'MONTH_YEAR']).agg(
    TOTAL_SALES = ('TOT_SALES', 'sum'),
    TOTAL_CUSTOMERS = ('LYLTY_CARD_NBR', 'nunique'),
    TOTAL_TRANSACTIONS = ('TXN_ID', 'nunique')
).reset_index()


# Calculate transactions per customer
monthly_metrics['TRANSACTIONS_PER_CUSTOMER'] = monthly_metrics['TOTAL_TRANSACTIONS'] / monthly_metrics['TOTAL_CUSTOMERS']

print("\nMonthly metrics summary:")
monthly_metrics.describe()
```

```
Monthly metrics summary:
```

	STORE_NBR	TOTAL_SALES	TOTAL_CUSTOMERS	TOTAL_TRANSACTIONS	TRANSACTIONS_PER_CUSTOMER
count	3169.000000	3169.000000	3169.000000	3169.000000	3169.000000
mean	136.802461	610.007889	69.826128	83.031556	1.141307
std	78.418604	389.699478	36.817352	47.918428	0.111341
min	1.000000	1.500000	1.000000	1.000000	1.000000
25%	68.000000	260.000000	41.000000	43.000000	1.041667
50%	137.000000	674.600000	79.000000	96.000000	1.125000
75%	204.000000	928.200000	102.000000	127.000000	1.233645
max	272.000000	1659.600000	150.000000	188.000000	1.468085

```
# Identify the trial stores and period
trial_stores = [77, 86, 88]
# Assuming trial period is the last month(s) in the dataset
```

```
trial_period_end = monthly_metrics['MONTH_YEAR'].max()
print(f"\nTrial stores: {trial_stores}")
print(f"Trial period ends at: {trial_period_end}")
```

```
Trial stores: [77, 86, 88]
Trial period ends at: 2019-06
```

```
# Determine the trial period (assuming last 3 months are trial period)
all_periods = sorted(monthly_metrics['MONTH_YEAR'].unique())
trial_period = all_periods[-3:]  # Last 3 months
pre_trial_period = all_periods[:-3]  # All months before the trial

print(f"\nTrial period: {trial_period}")
print(f"Pre-trial period: {pre_trial_period}")
```

```
Trial period: [Period('2019-04', 'M'), Period('2019-05', 'M'), Period('2019-06', 'M')]
Pre-trial period: [Period('2018-07', 'M'), Period('2018-08', 'M'), Period('2018-09', 'M'), Period('2018-10', 'M'), Period('2018-11',
```

```
# Function to calculate correlation and magnitude distance for control store selection
def calculate_similarity_metrics(trial_store_data, control_store_data, metric):
    """
    Calculate correlation and magnitude distance between trial and control store
    for a given metric (e.g., TOTAL_SALES, TOTAL_CUSTOMERS).

    Returns a tuple of (correlation, magnitude_score)
    """
    # Merge data for comparison
    comparison_data = pd.merge(
        trial_store_data[['MONTH_YEAR', metric]],
        control_store_data[['MONTH_YEAR', metric]],
        on='MONTH_YEAR',
        suffixes=('_trial', '_control')
    )

    # Calculate Pearson correlation
    correlation = comparison_data[f"{metric}_trial"].corr(comparison_data[f"{metric}_control"])
    # Calculate magnitude distance
    trial_mean = comparison_data[f"{metric}_trial"].mean()
    control_mean = comparison_data[f"{metric}_control"].mean()
    # Calculate normalized magnitude distance
    # 1 - abs(trial_mean - control_mean) / max(trial_mean, control_mean)
    # This gives a score from 0 to 1, where 1 means identical magnitudes
    magnitude_diff = abs(trial_mean - control_mean)
    max_magnitude = max(trial_mean, control_mean)
    magnitude_score = 1 - (magnitude_diff / max_magnitude)

    return correlation, magnitude_score


# Function to select the best control store for a given trial store
def select_control_store(trial_store, all_stores, monthly_data, pre_trial_data, metrics):
    """
    Select the best control store for a given trial store based on correlations
    and magnitude distances for multiple metrics.

    Parameters:
    - trial_store: Store number of the trial store
    - all_stores: List of all store numbers
    - monthly_data: DataFrame with monthly metrics for all stores
    - pre_trial_data: List of pre-trial periods
    - metrics: List of metrics to consider ['TOTAL_SALES', 'TOTAL_CUSTOMERS', etc.]

    Returns:
    - best_control_store: Store number of the best control store
    - scores: DataFrame with scores for all potential control stores
    """
    # Filter pre-trial data
    pre_trial_monthly = monthly_data[monthly_data['MONTH_YEAR'].isin(pre_trial_data)]

    # Get trial store data
    trial_store_data = pre_trial_monthly[pre_trial_monthly['STORE_NBR'] == trial_store]

    # Initialize scores DataFrame
    potential_controls = [store for store in all_stores if store != trial_store and store not in trial_stores]
    scores = pd.DataFrame({'STORE_NBR': potential_controls})
        # Calculate scores for each metric
    for metric in metrics:
        corr_scores = []
        mag_scores = []
```

```
            for control_store in potential_controls:
                control_store_data = pre_trial_monthly[pre_trial_monthly['STORE_NBR'] == control_store]

                if not control_store_data.empty and not trial_store_data.empty:
                    corr, mag = calculate_similarity_metrics(trial_store_data, control_store_data, metric)
                    corr_scores.append(corr)
                    mag_scores.append(mag)
                else:
                    # If either store has no data, assign low scores
                    corr_scores.append(-1)
                    mag_scores.append(0)

            scores[f'{metric}_CORRELATION'] = corr_scores
            scores[f'{metric}_MAGNITUDE'] = mag_scores
                # Calculate composite score (average of all metrics)
        score_columns = [col for col in scores.columns if col != 'STORE_NBR']
        scores['COMPOSITE_SCORE'] = scores[score_columns].mean(axis=1)

        # Sort by composite score
        scores = scores.sort_values('COMPOSITE_SCORE', ascending=False).reset_index(drop=True)

        return scores.iloc[0]['STORE_NBR'], scores


    # Function to visualize trial vs control store metrics
    def plot_trial_vs_control(trial_store, control_store, metric, monthly_data, trial_period):
        """
        Plot the performance of trial store vs control store for a given metric.
        """
        # Filter data for the specific stores
        trial_data = monthly_data[monthly_data['STORE_NBR'] == trial_store]
        control_data = monthly_data[monthly_data['STORE_NBR'] == control_store]

        # Create a figure
        plt.figure(figsize=(12, 6))

        # Convert period to string for plotting
        trial_data = trial_data.copy()
        control_data = control_data.copy()
        trial_data['MONTH_YEAR_STR'] = trial_data['MONTH_YEAR'].astype(str)
        control_data['MONTH_YEAR_STR'] = control_data['MONTH_YEAR'].astype(str)

        # Plot the metric over time
        plt.plot(trial_data['MONTH_YEAR_STR'], trial_data[metric], marker='o', linewidth=2, label=f'Trial Store {trial_store}')
        plt.plot(control_data['MONTH_YEAR_STR'], control_data[metric], marker='s', linewidth=2, label=f'Control Store {control_store}')

        # Add vertical line for trial period start
        trial_start = str(trial_period[0])
        plt.axvline(x=trial_start, color='red', linestyle='--', alpha=0.7, label='Trial Start')

        # Customize the plot
        plt.title(f'{metric} Comparison: Trial Store {trial_store} vs Control Store {control_store}')
        plt.xlabel('Month-Year')
        plt.ylabel(metric)
        plt.xticks(rotation=45)
        plt.legend()
        plt.tight_layout()

        return plt


    # Function to analyze statistical significance
    def analyze_significance(trial_store, control_store, metric, monthly_data, pre_trial_period, trial_period):
        """
        Analyze if there's a significant difference in performance between trial and control stores
        during the trial period compared to pre-trial period.
        """
        # Filter data
        trial_data = monthly_data[monthly_data['STORE_NBR'] == trial_store]
        control_data = monthly_data[monthly_data['STORE_NBR'] == control_store]

        # Pre-trial averages
        trial_pre = trial_data[trial_data['MONTH_YEAR'].isin(pre_trial_period)][metric].mean()
        control_pre = control_data[control_data['MONTH_YEAR'].isin(pre_trial_period)][metric].mean()

        # Trial period averages
        trial_during = trial_data[trial_data['MONTH_YEAR'].isin(trial_period)][metric].mean()
        control_during = control_data[control_data['MONTH_YEAR'].isin(trial_period)][metric].mean()

        # Calculate percentage changes
        if control_pre > 0 and trial_pre > 0:
            control_pct_change = (control_during - control_pre) / control_pre
```

```
            trial_pct_change = (trial_during - trial_pre) / trial_pre

            # Performance difference (how much better/worse the trial store performed compared to control)
            performance_diff = trial_pct_change - control_pct_change

            return {
                'trial_pre': trial_pre,
                'trial_during': trial_during,
                'control_pre': control_pre,
                'control_during': control_during,
                'control_pct_change': control_pct_change * 100,  # Convert to percentage
                'trial_pct_change': trial_pct_change * 100,      # Convert to percentage
                'performance_diff': performance_diff * 100       # Convert to percentage
            }
        else:
            return None


    # Get a list of all stores
    all_stores = monthly_metrics['STORE_NBR'].unique()


    # Define metrics for control store selection
    control_metrics = ['TOTAL_SALES', 'TOTAL_CUSTOMERS', 'TRANSACTIONS_PER_CUSTOMER']



    # Select control stores for each trial store
    control_store_mapping = {}
    control_store_scores = {}


    for trial_store in trial_stores:
        best_control, scores = select_control_store(
            trial_store,
            all_stores,
            monthly_metrics,
            pre_trial_period,
            control_metrics
        )
        control_store_mapping[trial_store] = best_control
        control_store_scores[trial_store] = scores
```

```
  c /= stddev[None, :]
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:2922: RuntimeWarning: invalid value encountered in divid
  c /= stddev[:, None]
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:2923: RuntimeWarning: invalid value encountered in divid
  c /= stddev[None, :]
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:2922: RuntimeWarning: invalid value encountered in divid
  c /= stddev[:, None]
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:2923: RuntimeWarning: invalid value encountered in divid
  c /= stddev[None, :]
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:2922: RuntimeWarning: invalid value encountered in divid
  c /= stddev[:, None]
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:2923: RuntimeWarning: invalid value encountered in divid
  c /= stddev[None, :]
```

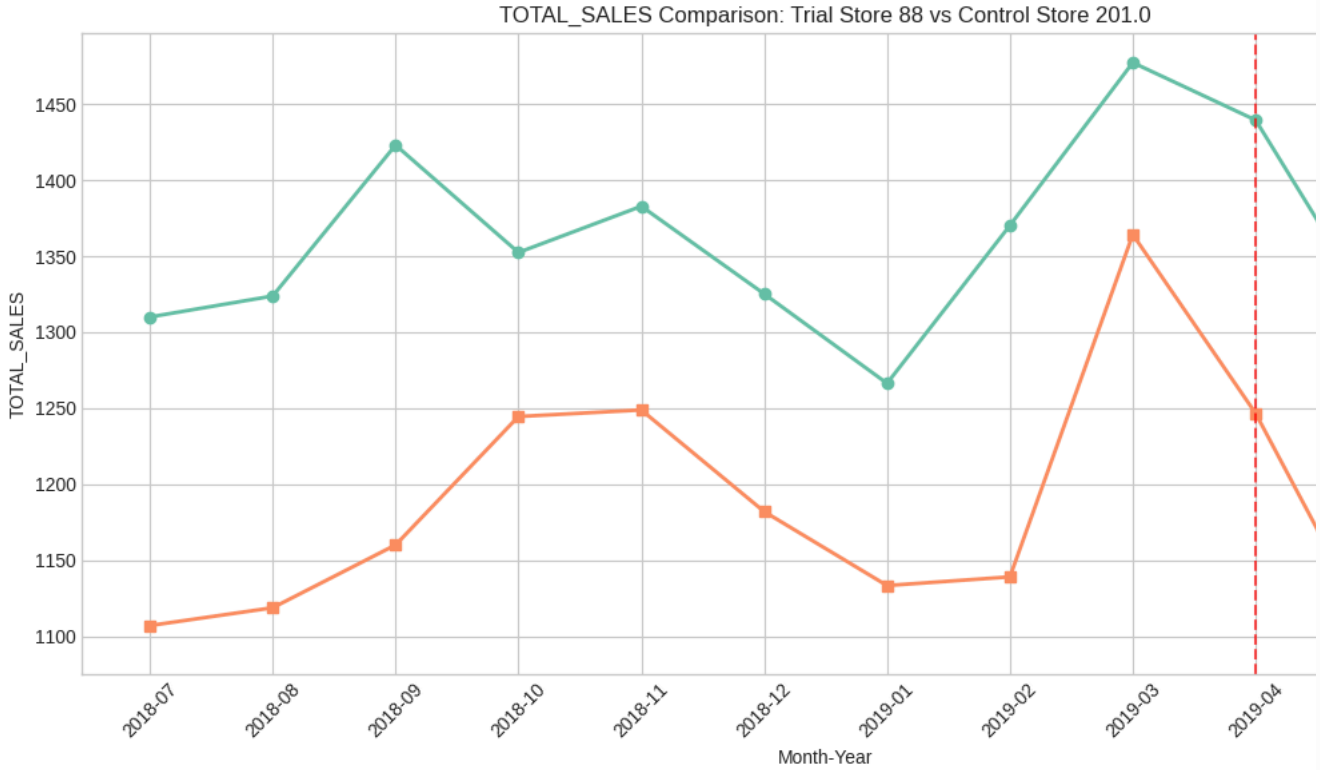
```
print("\nSelected control stores for each trial store:")
for trial_store, control_store in control_store_mapping.items():
    print(f"Trial Store {trial_store} -> Control Store {control_store}")
```

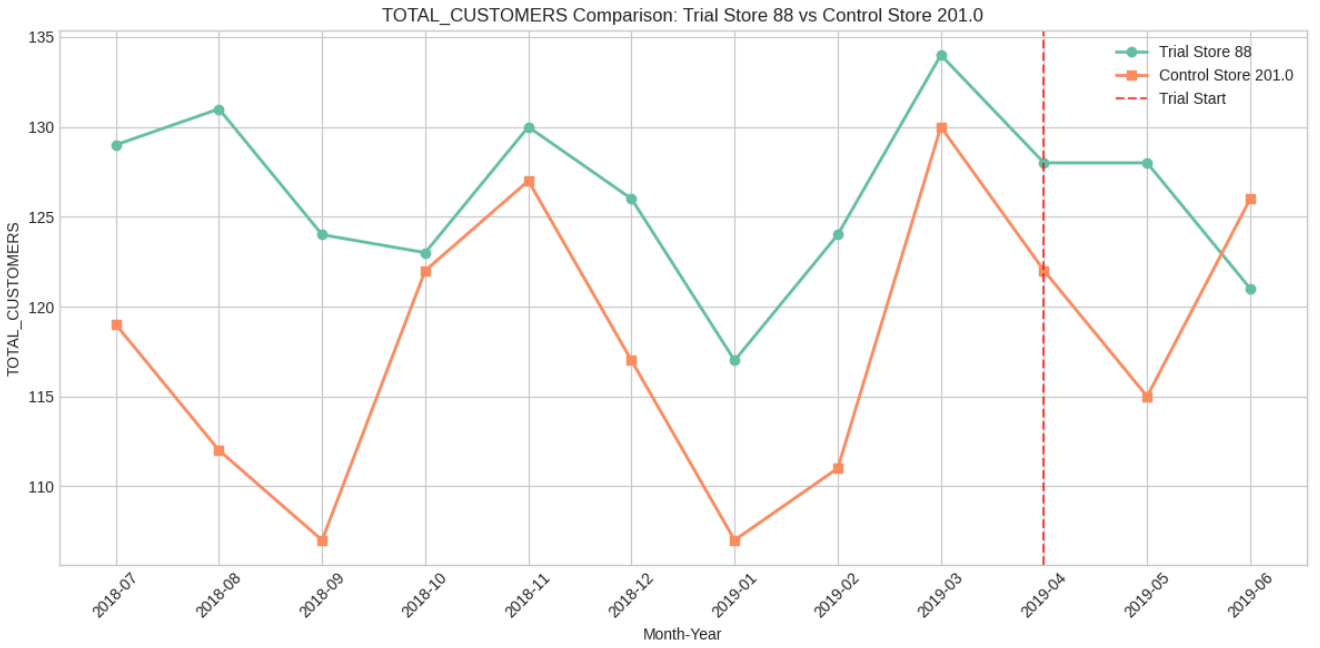
```
Selected control stores for each trial store:
Trial Store 77 -> Control Store 17.0
Trial Store 86 -> Control Store 13.0
Trial Store 88 -> Control Store 201.0
```

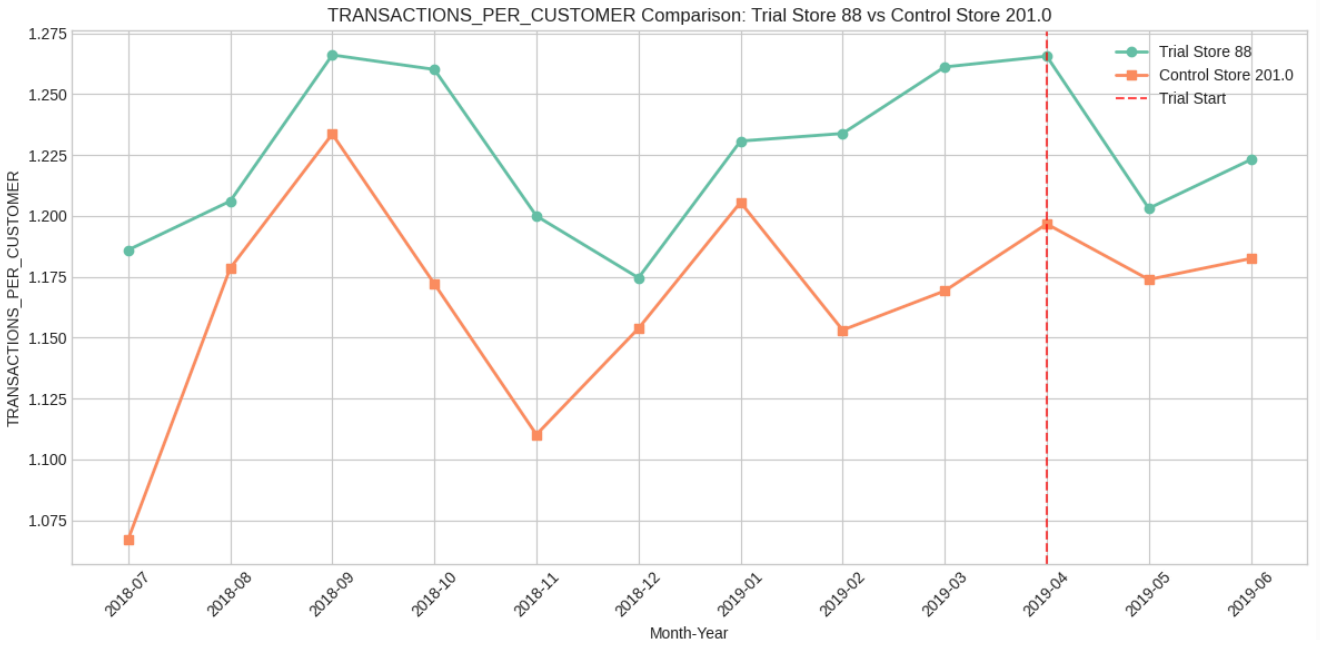
```
# Plot comparisons for each trial-control pair
for trial_store, control_store in control_store_mapping.items():
    print(f"\nAnalyzing Trial Store {trial_store} vs Control Store {control_store}")
```

```
Analyzing Trial Store 77 vs Control Store 17.0

Analyzing Trial Store 86 vs Control Store 13.0

Analyzing Trial Store 88 vs Control Store 201.0
```

```
    # Visualize each metric
    for metric in control_metrics:
        plot = plot_trial_vs_control(trial_store, control_store, metric, monthly_metrics, trial_period)
        plt.show()
         # Analyze significance
        result = analyze_significance(trial_store, control_store, metric, monthly_metrics, pre_trial_period, trial_period)
```


TOTAL_SALES Comparison: Trial Store 88 vs Control Store 201.0
TOTAL_SALES
Month-Year


TOTAL_CUSTOMERS Comparison: Trial Store 88 vs Control Store 201.0
Trial Store 88
Control Store 201.0
Trial Start
TOTAL_CUSTOMERS
Month-Year


TRANSACTIONS_PER_CUSTOMER Comparison: Trial Store 88 vs Control Store 201.0
Trial Store 88
Control Store 201.0
Trial Start
TRANSACTIONS_PER_CUSTOMER
Month-Year

```
    if result:
        print(f"\n{metric} Analysis:")
        print(f"Trial Store {trial_store}:")
        print(f"  - Pre-trial average: {result['trial_pre']:.2f}")
        print(f"  - Trial period average: {result['trial_during']:.2f}")
        print(f"  - Percentage change: {result['trial_pct_change']:.2f}%")

        print(f"Control Store {control_store}:")
        print(f"  - Pre-trial average: {result['control_pre']:.2f}")
        print(f"  - Trial period average: {result['control_during']:.2f}")
        print(f"  - Percentage change: {result['control_pct_change']:.2f}%")

        print(f"Performance difference: {result['performance_diff']:.2f}%")
```

```
TRANSACTIONS_PER_CUSTOMER Analysis:
Trial Store 88:
  - Pre-trial average: 1.22
  - Trial period average: 1.23
  - Percentage change: 0.52%
Control Store 201.0:
  - Pre-trial average: 1.16
  - Trial period average: 1.18
  - Percentage change: 2.07%
Performance difference: -1.55%
```

```
        # Interpretation
        if result['performance_diff'] > 5:
            print(f"✅ Trial store outperformed control store significantly for {metric}")
        elif result['performance_diff'] > 0:
            print(f"✓ Trial store performed slightly better than control store for {metric}")
        elif result['performance_diff'] > -5:
            print(f"⚠️ Trial store performed slightly worse than control store for {metric}")
        else:
            print(f"❌ Trial store underperformed compared to control store for {metric}")
```

```
⚠️ Trial store performed slightly worse than control store for TRANSACTIONS_PER_CUSTOMER
```

```
# Summary of findings and recommendations
print("\n===== SUMMARY OF FINDINGS =====")
summary_results = {}

for trial_store in trial_stores:
    control_store = control_store_mapping[trial_store]
    store_results = {}

    print(f"\nTrial Store {trial_store} (Control: Store {control_store}):")

    for metric in control_metrics:
        result = analyze_significance(trial_store, control_store, metric, monthly_metrics, pre_trial_period, trial_period)
        if result:
            store_results[metric] = result['performance_diff']
            print(f"- {metric}: {result['performance_diff']:.2f}% {'better' if result['performance_diff'] > 0 else 'worse'} than control"
               # Overall recommendation
    avg_performance = sum(store_results.values()) / len(store_results)

    if avg_performance > 5:
        recommendation = "Strong positive impact. Recommend rolling out new layout."
    elif avg_performance > 0:
        recommendation = "Slight positive impact. Consider rolling out new layout with minor adjustments."
    elif avg_performance > -5:
        recommendation = "Minimal negative impact. May need further testing or adjustments before rollout."
    else:
        recommendation = "Significant negative impact. Do not recommend rolling out new layout."

    print(f"Overall performance: {avg_performance:.2f}%")
    print(f"Recommendation: {recommendation}")

    summary_results[trial_store] = {
        'control_store': control_store,
        'metrics': store_results,
        'avg_performance': avg_performance,
        'recommendation': recommendation
```

```
    }

print("\n===== FINAL RECOMMENDATIONS =====")
positive_trials = [store for store, data in summary_results.items() if data['avg_performance'] > 0]
negative_trials = [store for store, data in summary_results.items() if data['avg_performance'] <= 0]

if len(positive_trials) > len(negative_trials):
    print(f"Overall, {len(positive_trials)} out of {len(trial_stores)} trial stores showed positive results.")
    print("The new layout appears to have a positive impact on performance metrics.")
    if negative_trials:
        print(f"However, stores {negative_trials} did not show improvement and may need special consideration.")
    print("Recommendation: Proceed with rollout to all stores, with possible adjustments for specific store characteristics.")
else:
    print(f"Overall, {len(negative_trials)} out of {len(trial_stores)} trial stores showed negative results.")
    print("The new layout appears to have a negative impact on performance metrics.")
    if positive_trials:
        print(f"However, stores {positive_trials} showed improvement and could provide insights for redesign.")
    print("Recommendation: Do not proceed with rollout. Further testing and redesign is needed.")
```

```
 ===== SUMMARY OF FINDINGS =====

 Trial Store 77 (Control: Store 17.0):
 - TOTAL_SALES: 27.38% better than control
 - TOTAL_CUSTOMERS: 16.63% better than control
 - TRANSACTIONS_PER_CUSTOMER: -1.05% worse than control
 Overall performance: 14.32%
 Recommendation: Strong positive impact. Recommend rolling out new layout.
```