Quantium Virtual Internship - Retail Strategy and Analytics - Task 2

Title:

Evaluation of the Impact of New Store Layouts on Sales Performance: A Comparative Analysis of Trial vs. Control Stores

Objective:

The objective of this analysis is to assess the effectiveness of new store layouts trialed in selected stores (Stores 77, 86, and 88) by comparing their sales performance against control stores. The goal is to provide data-driven recommendations on whether the new layouts should be rolled out to all stores based on their impact on key sales metrics, including total sales revenue, customer footfall, and transaction frequency.

Problem Statement:

Evaluating the impact of new store layouts trialed in Stores 77, 86, and 88. The primary challenge is to determine whether these new layouts significantly improve sales performance compared to control stores. This involves selecting appropriate control stores, developing a robust methodology for comparison, and analyzing whether the observed changes in sales metrics during the trial period are statistically significant. The outcome of this analysis will inform the decision on whether to implement the new layouts across all stores.

Load required libraries and datasets

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import numpy as np
from scipy import stats

# Set plot styles
plt.rcParams['figure.figsize'] = (10, 6)

# Read in data from previous module
data = pd.read_csv("QVI_data.csv")
```

Select control stores

The client has selected store numbers 77, 86 and 88 as trial stores and want control stores to be established stores that are operational for the entire observation period. We would want to match trial stores to control stores that are similar to the trial store prior to the trial period of Feb 2019 in terms of : throughout the pre-trial period.

- Monthly overall sales revenue
- Monthly number of customers
- Monthly number of transactions per customer Let's first create the metrics of interest and filter to stores that are present

Calculate these measures over time for each store

In [ ]:
```
data['YEARMONTH'] = pd.to_datetime(data['DATE']).dt.to_period('M')
```

Next, we define the measure calculations to use during the analysis. For each store and month calculate total sales, number of customers, transactions per customer, chips per customer and the average price per unit.

In [ ]:
```
# Calculate measures over time for each store
measure_over_time = data.groupby(['STORE_NBR', 'YEARMONTH']).agg({
    'TOT_SALES': 'sum',
    'LYLTY_CARD_NBR': 'nunique',
    'TXN_ID': 'count',
    'PROD_QTY': 'sum'
}).reset_index()
```

In [ ]:
```
measure_over_time = measure_over_time.assign(
    nTxnPerCust=lambda x: x['TXN_ID'] / x['LYLTY_CARD_NBR'],
    nChipsPerTxn=lambda x: x['PROD_QTY'] / x['TXN_ID'],
    avgPricePerUnit=lambda x: x['TOT_SALES'] / x['PROD_QTY']
)
```

Filter to the pre-trial period and stores with full observation periods

In [ ]:
```
# 12 means 1 year
stores_with_full_obs = measure_over_time.groupby('STORE_NBR').size()
```

In [ ]:
```
# stores that were operational the whole 12 months
stores_with_full_obs = stores_with_full_obs[stores_with_full_obs == 12].index
```

In [ ]:
```
# stores that were operational prior to Feb 2019 (trial start date)
pre_trial_measures = measure_over_time[
    (measure_over_time['YEARMONTH'] < pd.Period('2019-02')) &
    (measure_over_time['STORE_NBR'].isin(stores_with_full_obs))
]
```

Now we need to work out a way of ranking how similar each potential control store is to the trial store. We can calculate how correlated the performance of each store is to the trial store.

In [ ]:
```
def calculate_correlation(input_table, metric_col, store_comparison):
    calc_corr_table = pd.DataFrame(columns=['Store1', 'Store2', 'corr_measure'])
    store_numbers = input_table['STORE_NBR'].unique()

    for store in store_numbers:
        if store != store_comparison:
            # Extract data for both stores
            store_data = input_table[input_table['STORE_NBR'] == store][['YEARMONTH', metric_col]]
            comparison_data = input_table[input_table['STORE_NBR'] == store_comparison][['YEARMONTH',

            # Align both stores on the same YEARMONTH (time period)
            aligned_data = pd.merge(store_data, comparison_data, on='YEARMONTH', suffixes=('', '_compa

            # Calculate correlation
            correlation = aligned_data[metric_col].corr(aligned_data[f'{metric_col}_comparison'])

            # Append directly using loc
            calc_corr_table.loc[len(calc_corr_table)] = [store_comparison, store, correlation]

    return calc_corr_table
```

In [ ]:
```
def calculate_magnitude_distance(input_table, metric_col, store_comparison):
    calc_dist_table = pd.DataFrame(columns=['Store1', 'Store2', 'YEARMONTH', 'measure'])
    store_numbers = input_table['STORE_NBR'].unique()

    for store in store_numbers:
        if store != store_comparison:
            # Extract data for both stores
            store_data = input_table[input_table['STORE_NBR'] == store][['YEARMONTH', metric_col]]
            comparison_data = input_table[input_table['STORE_NBR'] == store_comparison][['YEARMONTH',

            # Align the stores on the same YEARMONTH
            aligned_data = pd.merge(store_data, comparison_data, on='YEARMONTH', suffixes=('', '_compa

            # Calculate magnitude distance
            magnitude_distance = abs(aligned_data[metric_col] - aligned_data[f'{metric_col}_comparison

            # Append the data to the DataFrame using .loc[]
```

```
            for idx in range(len(magnitude_distance)):
                calc_dist_table.loc[len(calc_dist_table)] = [store_comparison, store, aligned_data['YE

    # Standardize the magnitude distance
    min_max_dist = calc_dist_table.groupby(['Store1', 'YEARMONTH']).agg({
        'measure': ['min', 'max']
    }).reset_index()
    min_max_dist.columns = ['Store1', 'YEARMONTH', 'minDist', 'maxDist']

    dist_table = pd.merge(calc_dist_table, min_max_dist, on=['Store1', 'YEARMONTH'])
    dist_table['magnitude_measure'] = 1 - (dist_table['measure'] - dist_table['minDist']) / (dist_tabl

    final_dist_table = dist_table.groupby(['Store1', 'Store2'])['magnitude_measure'].mean().reset_inde
    return final_dist_table
```

Now let's use the functions to find the control stores! We'll select control stores based on how similar monthly total sales in dollar amounts and monthly number of customers are to the trial stores. So we will need to use our functions to get four scores, two for each of total sales and total customers.

Use the function we created to calculate correlations against store 77 using total sales and number of customers.

In [ ]:
```
trial_store = 77
corr_sales = calculate_correlation(pre_trial_measures, 'TOT_SALES', trial_store)
```

In [ ]:
```
corr_customers = calculate_correlation(pre_trial_measures, 'LYLTY_CARD_NBR', trial_store)
```

Then, use the functions for calculating magnitude

In [ ]:
```
magnitude_sales = calculate_magnitude_distance(pre_trial_measures, 'TOT_SALES', trial_store)
```

In [ ]:
```
magnitude_customers = calculate_magnitude_distance(pre_trial_measures, 'LYLTY_CARD_NBR', trial_store)
```

Create a combined score composed of correlation and magnitude, by first merging the correlations table with the magnitude table

In [ ]:
```
corr_weight = 0.5
score_sales = pd.merge(corr_sales, magnitude_sales, on=['Store1', 'Store2'])
# weighted average of the correlation and magnitude distance. The corr_weight determines how much impo
score_sales['score_sales'] = corr_weight * score_sales['corr_measure'] + (1 - corr_weight) * score_sal
```

In [ ]:
```
score_customers = pd.merge(corr_customers, magnitude_customers, on=['Store1', 'Store2'])
score_customers['score_customers'] = corr_weight * score_customers['corr_measure'] + (1 - corr_weight)
```

Now we have a score for each of total number of sales and number of customers. Let's combine the two via a simple average

Combine scores across the drivers by first merging our sales scores and customer scores into a single table

In [ ]:
```
score_control = pd.merge(score_sales[['Store1', 'Store2', 'score_sales']],
                         score_customers[['Store1', 'Store2', 'score_customers']],
                         on=['Store1', 'Store2'])
score_control['final_control_score'] = 0.5 * score_control['score_sales'] + 0.5 * score_control['score
```

The store with the highest score is then selected as the control store since it is most similar to the trial store

Select control stores based on the highest matching store (closest to 1 but not the store itself, i.e. the second ranked highest store) Select the most appropriate control store for trial store 77 by finding the store with the highest final score.

In [ ]:
```
# Select control store
control_store = score_control[score_control['Store1'] == trial_store].sort_values('final_control_score
print(f"The control store for trial store {trial_store} is store {control_store}")
```

```
The control store for trial store 77 is store 233.0
```

To find the most appropriate control store for trial store 77, we should look for the Store2 value that has the highest final_control_score, which is closest to 1. This score represents how closely another store (Store2) matches the trial store (Store1, which is store 77) in terms of sales and customer behavior.

Steps we used to Identify the Best Control Store: Review the final_control_score Column: This column contains the combined score based on both sales and customer metrics. The store with the highest score is the most similar to the trial store.

Ignore Store1 (77): Since we want to find a control store that is not the trial store itself, we should look at the Store2 column to find the matching store.

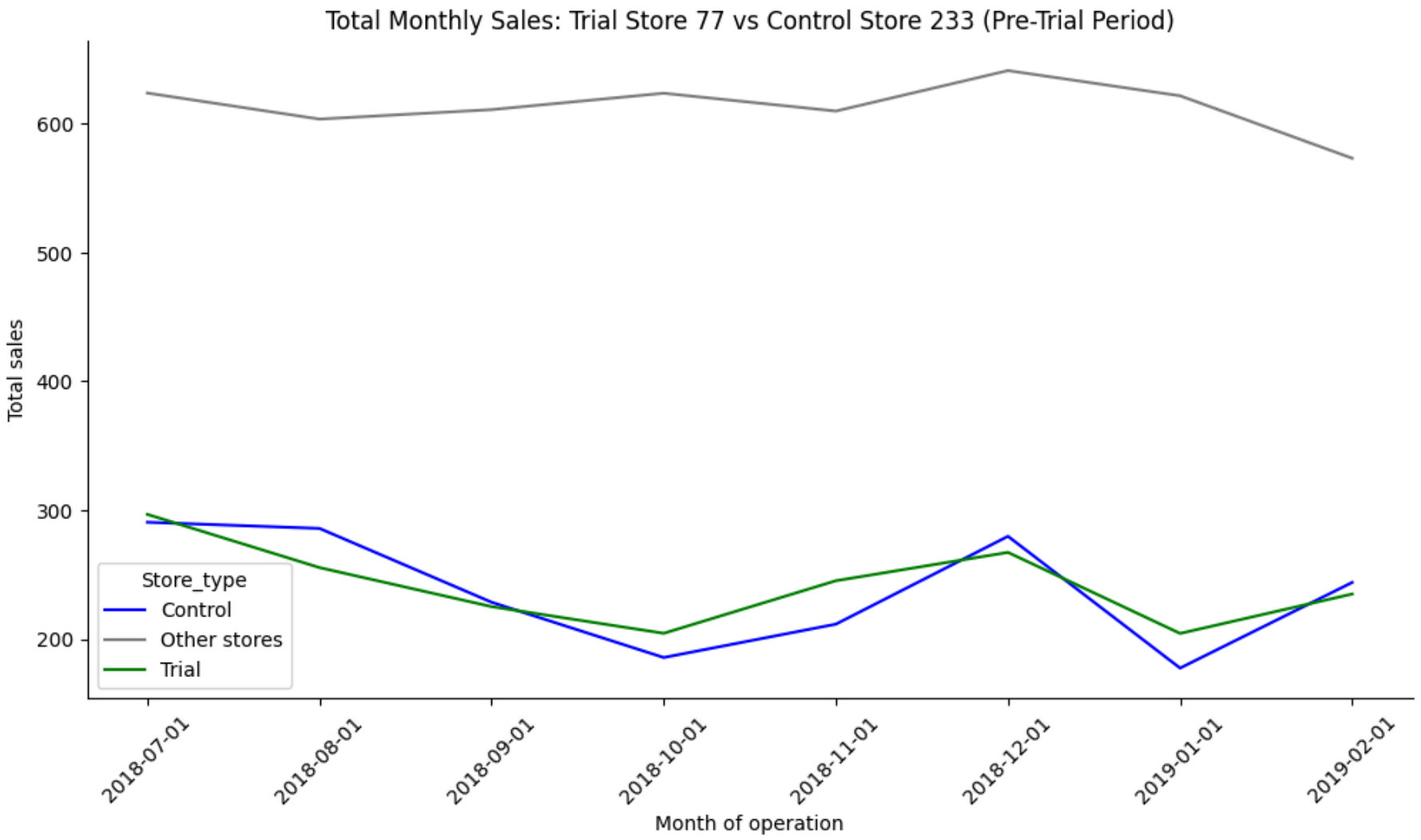
Select the Highest final_control_score: The store corresponding to the highest final_control_score in the Store2 column is the best match for store 77.

Identifying the Best Control Store: Store 233 has the highest final_control_score of 0.968499. Therefore, Store 233 is the best match and should be selected as the control store for the trial store 77.

Now that we have found a control store, let's check visually if the drivers are indeed similar in the period before the trial

We'll look at total sales first.

In [ ]:
```
measure_over_time_sales = measure_over_time.copy()

# Mapping the stores to 'Trial', 'Control', and 'Other stores'
measure_over_time_sales['Store_type'] = measure_over_time_sales['STORE_NBR'].map(
    {trial_store: 'Trial', control_store: 'Control'}
).fillna('Other stores')

# Convert 'YEARMONTH' to timestamp if it is in PeriodDtype format
if isinstance(measure_over_time_sales['YEARMONTH'].dtype, pd.core.dtypes.dtypes.PeriodDtype):
    measure_over_time_sales['YEARMONTH'] = measure_over_time_sales['YEARMONTH'].dt.to_timestamp()

# Converting 'YEARMONTH' to string format for plotting
measure_over_time_sales['YEARMONTH'] = measure_over_time_sales['YEARMONTH'].astype(str)

# Filtering the data to include only periods before March 2019
past_sales = measure_over_time_sales[measure_over_time_sales['YEARMONTH'] < '2019-03']

# Grouping by 'YEARMONTH' and 'Store_type' to calculate the mean of 'TOT_SALES'
past_sales = past_sales.groupby(['YEARMONTH', 'Store_type'])['TOT_SALES'].mean().reset_index()

color_palette = {
            'Trial': 'green',
            'Control': 'blue',
            'Other stores' : 'grey'
        }

# Plotting the data
plt.figure(figsize=(12, 6))
sns.lineplot(data=past_sales, x='YEARMONTH', y='TOT_SALES', hue='Store_type', palette=color_palette)
plt.title("Total Monthly Sales: Trial Store 77 vs Control Store 233 (Pre-Trial Period)")
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.xticks(rotation=45)  # Rotate x-axis labels if necessary
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



The graph we've generated shows the total sales over time for three categories: the Control store, the Trial store, and Other stores. Here's how to interpret the graph:

Control Store (Blue Line):

The total sales in the control store generally show a declining trend from July 2018 to December 2018. However, there is a slight increase in January 2019, followed by another dip in February 2019. This trend indicates some fluctuations in sales, with the overall trend being downward during the observed period. Trial Store (Green Line):

The trial store follows a pattern that is quite similar to the control store, which is a positive sign, as this similarity is what we want when selecting a control store. The sales decline from July 2018, hitting a low around November 2018, and then there is a small peak in December 2018. Similar to the control store, there is a dip in January 2019 and a slight rise in February 2019.

Other Stores (Orange Line):

The "Other stores" category has consistently higher sales compared to both the control and trial stores. The sales in this group are relatively stable with a slight upward trend through 2018, followed by a gradual decline starting in January 2019. This trend shows that the other stores do not follow the same pattern as the trial and control stores, which is expected since these stores are not directly involved in the trial.

Key Takeaways:

Similarity Between Control and Trial Stores: The control and trial stores display similar sales trends, which is crucial for the validity of the trial. If the trial is to be evaluated based on the impact on sales, it's important that the control store provides a realistic comparison. The graph indicates that the control store we've chosen is a good match for the trial store, as their sales trends are closely aligned.

Different Behavior of Other Stores:

The "Other stores" category behaves differently, with higher and more stable sales. This difference further validates that the selected control store is unique and more comparable to the trial store, rather than to the general population of stores.

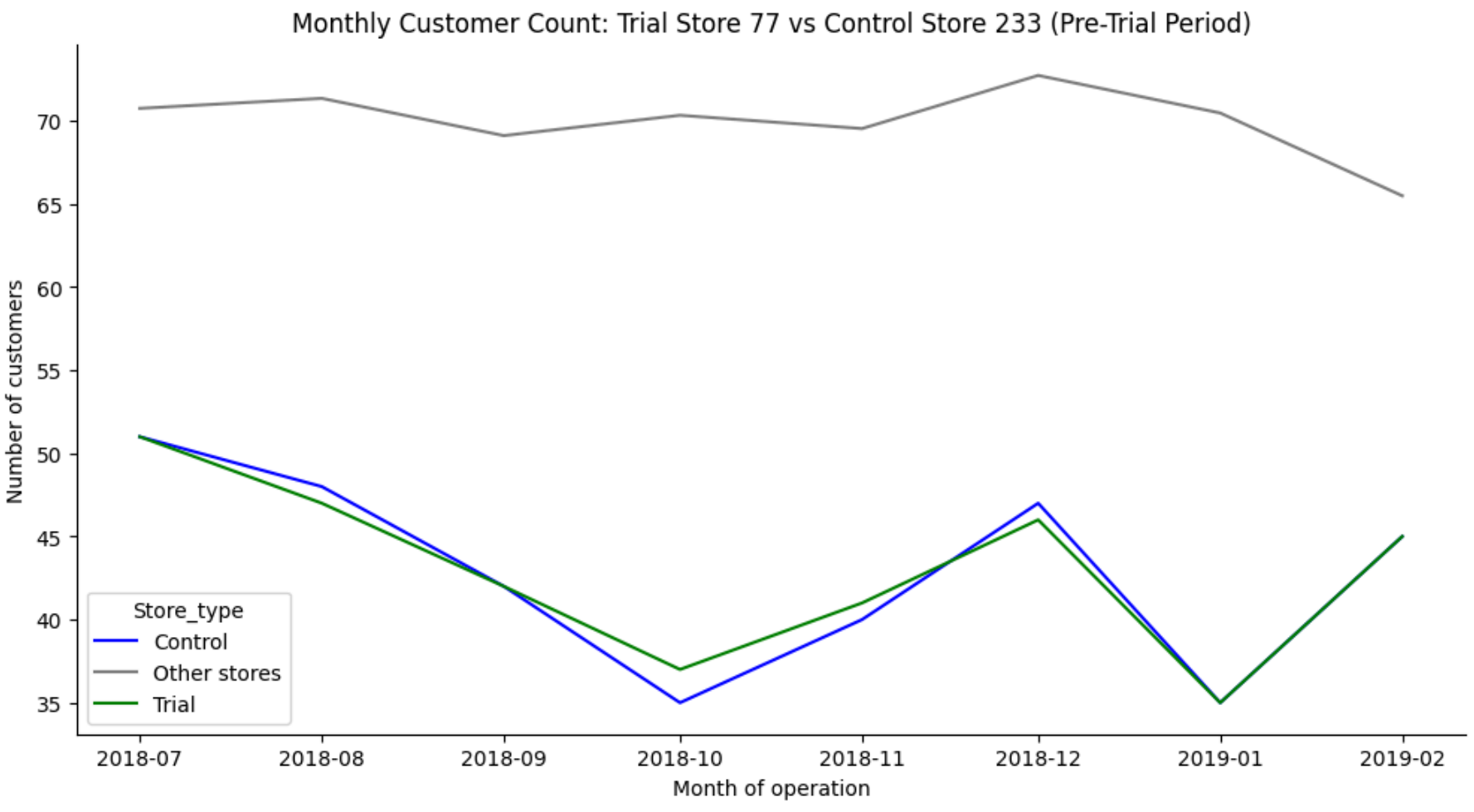
Month-to-Month Variability:

Both the control and trial stores exhibit month-to-month variability, with visible dips and peaks. This variability should be considered when analyzing the impact of the trial, as external factors (like seasonality) might be affecting these fluctuations.

In summary, this graph confirms that the selected control store has a similar sales trend to the trial store, making it a suitable choice for comparison during the trial period.

Next, number of customers. Visual checks on customer count trends by comparing the trial store to the control store and other stores.

In [ ]:
```
# Create a copy of the measure_over_time data
measure_over_time_customers = measure_over_time.copy()

# Map the store numbers to store types: Trial, Control, and Other stores
measure_over_time_customers['Store_type'] = measure_over_time_customers['STORE_NBR'].map(
    {trial_store: 'Trial', control_store: 'Control'}
).fillna('Other stores')

# Convert 'YEARMONTH' to a timestamp if it is a PeriodDtype
if isinstance(measure_over_time_customers['YEARMONTH'].dtype, pd.PeriodDtype):
    measure_over_time_customers['YEARMONTH'] = measure_over_time_customers['YEARMONTH'].dt.to_timestam


# Filter the data to include only the periods before March 2019
past_customers = measure_over_time_customers[measure_over_time_customers['YEARMONTH'] < '2019-03-01']

# Group by 'YEARMONTH' and 'Store_type' and calculate the mean number of customers
past_customers = past_customers.groupby(['YEARMONTH', 'Store_type'])['LYLTY_CARD_NBR'].mean().reset_in

color_palette = {
            'Trial': 'green',
            'Control': 'blue',
            'Other stores' : 'grey'
        }

# Plotting the number of customers over time
plt.figure(figsize=(12, 6))
sns.lineplot(data=past_customers, x='YEARMONTH', y='LYLTY_CARD_NBR', hue='Store_type', palette=color_p
plt.title("Monthly Customer Count: Trial Store 77 vs Control Store 233 (Pre-Trial Period)")
plt.xlabel('Month of operation')
plt.ylabel('Number of customers')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



The line graph we've generated shows the number of customers by month for three different store types: Control, Other stores, and Trial, from July 2018 to February 2019. Here's an interpretation of the trends observed in the graph:

Control Store (Blue Line):

The number of customers in the control store started at around 50 in July 2018. There was a gradual decline until November 2018, where it reached a low point of approximately 35 customers. After November 2018, the number of customers increased slightly in December before dropping again in January 2019. The number of customers then recovered slightly in February 2019.

Trial Store (Green Line):

The trend in the trial store closely mirrors the control store's trend, with a starting point just below the control store's customer count in July 2018. Similar to the control store, the trial store experienced a decline in customers until November 2018. The trial store's customer count remained slightly below the control store's throughout most of the period. However, in February 2019, the trial store's customer count surpassed that of the control store. In summary, this graph confirms that the selected control store has a similar customer count to the trial store, making it a suitable choice for comparison during the trial period.

The trial period goes from the start of February 2019 to April 2019. We now want to see if there has been an uplift in overall chip sales

We'll start with scaling the control store's sales to a level similar to control for any differences between the two stores outside of the trial period.

In [ ]:
```
# Assessment of trial
# Scale pre-trial control sales to match pre-trial trial store sales
scaling_factor_sales = (
    pre_trial_measures[pre_trial_measures['STORE_NBR'] == trial_store]['TOT_SALES'].sum() /
    pre_trial_measures[pre_trial_measures['STORE_NBR'] == control_store]['TOT_SALES'].sum()
)
scaling_factor_sales
```

Out[ ]:
```
1.023617303289553
```

Apply the scaling factor

In [ ]:
```
measure_over_time_sales = measure_over_time.copy()
scaled_control_sales = measure_over_time_sales[measure_over_time_sales['STORE_NBR'] == control_store].
scaled_control_sales['control_sales'] = scaled_control_sales['TOT_SALES'] * scaling_factor_sales
```

Now that we have comparable sales figures for the control store, we can calculate the percentage difference between the scaled control sales and the trial store's sales during the trial period.

Calculate the percentage difference between scaled control sales and trial sales

In [ ]:
```
# Calculate percentage difference
percentage_diff = pd.merge(
    scaled_control_sales[['YEARMONTH', 'control_sales']],
    measure_over_time_sales[measure_over_time_sales['STORE_NBR'] == trial_store][['YEARMONTH', 'TOT_SA
    on='YEARMONTH'
)
percentage_diff['percentage_diff'] = (percentage_diff['TOT_SALES'] - percentage_diff['control_sales'])
```

Let's see if the difference is significant!

As our null hypothesis is that the trial period is the same as the pre-trial period, let's take the standard deviation based on the scaled percentage difference in the pre-trial period

In [ ]:
```
std_dev = percentage_diff[percentage_diff['YEARMONTH'] < pd.Period('2019-02')]['percentage_diff'].std(
std_dev
```

Out[ ]:
```
0.09958646884078388
```

In [ ]:
```
degrees_of_freedom = len(percentage_diff[percentage_diff['YEARMONTH'] <= pd.Period('2019-02')]) - 1
degrees_of_freedom
```

Out[ ]: 7

We will test with a null hypothesis of there being 0 difference between trial and control stores

Calculate the t-values for the trial months. After that, find the 95th percentile of the t distribution with the appropriate degrees of freedom to check whether the hypothesis is statistically significant. The test statistic here is (x - u)/standard deviation

In [ ]:
```
# Calculate t-values for trial periods
t_values = percentage_diff[(percentage_diff['YEARMONTH'] >= pd.Period('2019-02')) &
                           (percentage_diff['YEARMONTH'] <= pd.Period('2019-04'))].copy()
t_values['t_value'] = (t_values['percentage_diff'] - 0) / std_dev
t_values['significant'] = abs(t_values['t_value']) > stats.t.ppf(0.95, degrees_of_freedom)
print(t_values[['YEARMONTH', 't_value', 'significant']])
```

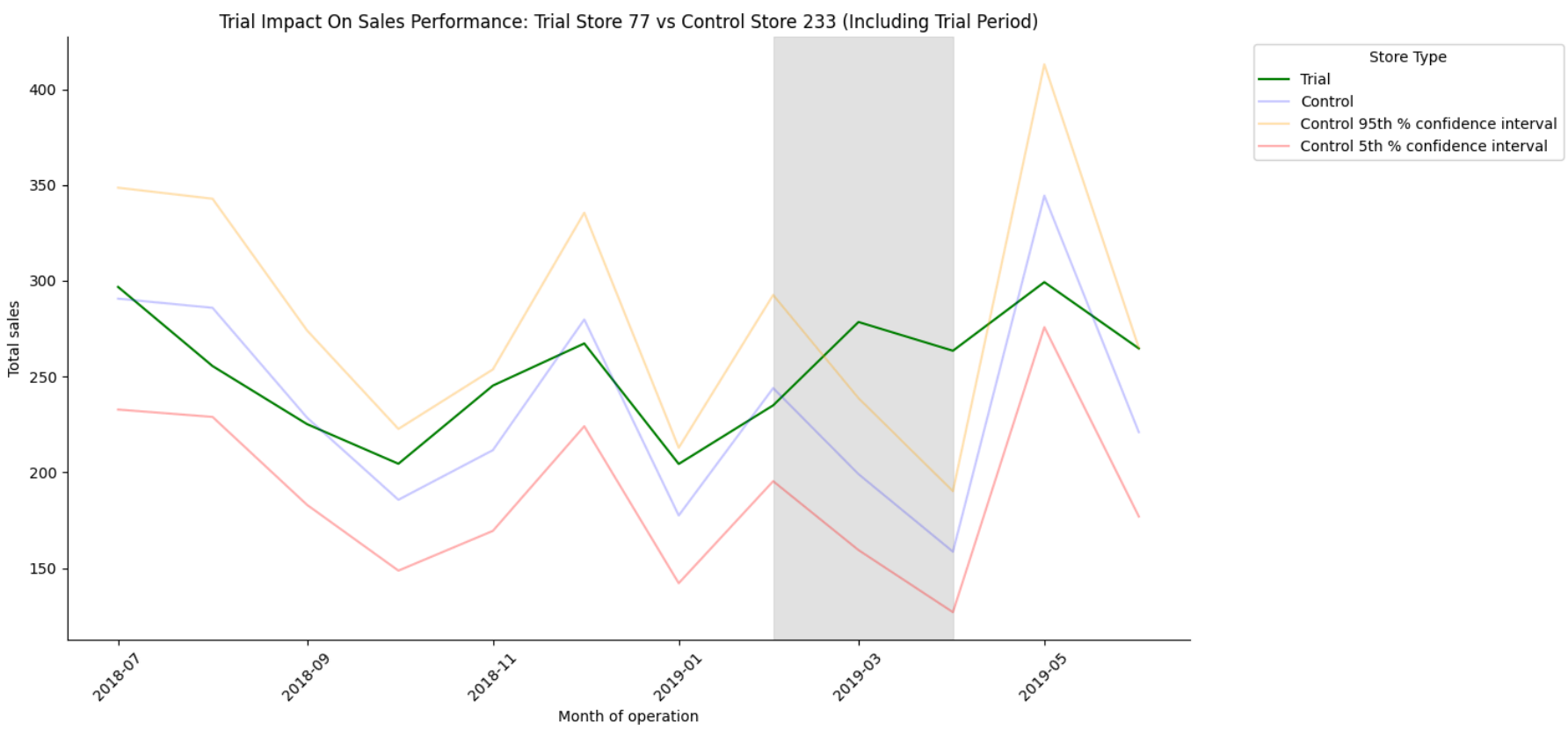
```
  YEARMONTH   t_value  significant
7   2019-02 -0.593520        False
8   2019-03  3.680430         True
9   2019-04  6.256669         True
```

We can observe that the t-value is much larger than the 95th percentile value of the t-distribution for March and April - i.e. the increase in sales in the trial store in March and April is statistically greater than in the control store.

Let's create a more visual version of this by plotting the sales of the control store, the sales of the trial stores and the 95th percentile value of sales of the control store.

In [ ]:
```
# Ensure YEARMONTH is in the correct datetime format
measure_over_time_sales['YEARMONTH'] = pd.to_datetime(measure_over_time_sales['YEARMONTH'].astype(str)

# Ensure TOT_SALES is numeric
measure_over_time_sales['TOT_SALES'] = pd.to_numeric(measure_over_time_sales['TOT_SALES'], errors='coe

# Drop any rows with NaN values that might have been introduced
measure_over_time_sales.dropna(subset=['TOT_SALES', 'YEARMONTH'], inplace=True)

# Copy data and label store types
past_sales = measure_over_time_sales.copy()
past_sales['Store_type'] = past_sales['STORE_NBR'].apply(lambda x: 'Trial' if x == trial_store else 'C

# Filter only Trial and Control stores
past_sales = past_sales[past_sales['Store_type'].isin(['Trial', 'Control'])]

# Calculate the 95th percentile and 5th percentile for the Control store
past_sales_control_95 = past_sales[past_sales['Store_type'] == 'Control'].copy()
past_sales_control_95['TOT_SALES'] = past_sales_control_95['TOT_SALES'] * (1 + std_dev * 2)
past_sales_control_95['Store_type'] = 'Control 95th % confidence interval'

past_sales_control_5 = past_sales[past_sales['Store_type'] == 'Control'].copy()
past_sales_control_5['TOT_SALES'] = past_sales_control_5['TOT_SALES'] * (1 - std_dev * 2)
past_sales_control_5['Store_type'] = 'Control 5th % confidence interval'

# Combine data for plotting
trial_assessment = pd.concat([past_sales, past_sales_control_95, past_sales_control_5])

# Define the custom color palette
color_palette = {
                'Trial': 'green',
                'Control': (0, 0, 1, 0.2),  # Blue with 0.3 alpha
                'Control 95th % confidence interval': (1, 0.65, 0, 0.3),  # Orange with 0.3 alpha
                'Control 5th % confidence interval': (1, 0, 0, 0.3)  # Red with 0.3 alpha
            }
# Plot the results for Total Sales
plt.figure(figsize=(15, 7))
sns.lineplot(data=trial_assessment, x='YEARMONTH', y='TOT_SALES', hue='Store_type', palette=color_pale
plt.axvspan(pd.to_datetime('2019-02'), pd.to_datetime('2019-04'), alpha=0.2, color='gray')
plt.title("Trial Impact On Sales Performance: Trial Store 77 vs Control Store 233 (Including Trial Per
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.xticks(rotation=45)
plt.legend(title='Store Type', bbox_to_anchor=(1.05, 1), loc='upper left')
```

```
plt.tight_layout()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



March 2019: The trial store’s sales (blue line) are indeed above the 95th percentile confidence interval of the control store (green line). April 2019: The trial store’s sales again exceed the 95th percentile confidence interval of the control store. This means that the trial store's performance lies outside the 5% to 95% confidence interval of the control store in two of the three trial months (March and April 2019).

Therefore, the statement that the trial in store 77 is significantly different from its control store during the trial period, as the trial store's performance lies outside the 5% to 95% confidence interval in two of the three trial months, is indeed true.

Combined Functions created to calculate control store, visualize similarities between control store and trial store, analysis of impact of the trial on the over sales and number of customers of both store to establish significance

In [ ]:
```
def find_control_store(pre_trial_data, trial_store, metric_cols=['TOT_SALES', 'LYLTY_CARD_NBR']):
    """
    Find the best control store for a given trial store.

    :param pre_trial_data: DataFrame containing the store data
    :param trial_store: Store number of the trial store
    :param metric_cols: List of metric columns to use for comparison
    :return: Best control store number
    """

    def calculate_correlation(input_table, metric_col, store_comparison):
        calc_corr_table = pd.DataFrame(columns=['Store1', 'Store2', 'corr_measure'])
        store_numbers = input_table['STORE_NBR'].unique()

        for store in store_numbers:
            if store != store_comparison:
                store_data = input_table[input_table['STORE_NBR'] == store][['YEARMONTH', metric_col]]
                comparison_data = input_table[input_table['STORE_NBR'] == store_comparison][['YEARMONT
                aligned_data = pd.merge(store_data, comparison_data, on='YEARMONTH', suffixes=('', '_c
                correlation = aligned_data[metric_col].corr(aligned_data[f'{metric_col}_comparison'])
                calc_corr_table.loc[len(calc_corr_table)] = [store_comparison, store, correlation]

        return calc_corr_table

    def calculate_magnitude_distance(input_table, metric_col, store_comparison):
        calc_dist_table = pd.DataFrame(columns=['Store1', 'Store2', 'YEARMONTH', 'measure'])
        store_numbers = input_table['STORE_NBR'].unique()

        for store in store_numbers:
            if store != store_comparison:
```

```
            store_data = input_table[input_table['STORE_NBR'] == store][['YEARMONTH', metric_col]]
            comparison_data = input_table[input_table['STORE_NBR'] == store_comparison][['YEARMONT
            aligned_data = pd.merge(store_data, comparison_data, on='YEARMONTH', suffixes=('', '_c
            magnitude_distance = abs(aligned_data[metric_col] - aligned_data[f'{metric_col}_compar

            for idx in range(len(magnitude_distance)):
                calc_dist_table.loc[len(calc_dist_table)] = [store_comparison, store, aligned_data

    # Standardize the magnitude distance
    min_max_dist = calc_dist_table.groupby(['Store1', 'YEARMONTH']).agg({
        'measure': ['min', 'max']
    }).reset_index()
    min_max_dist.columns = ['Store1', 'YEARMONTH', 'minDist', 'maxDist']

    dist_table = pd.merge(calc_dist_table, min_max_dist, on=['Store1', 'YEARMONTH'])
    dist_table['magnitude_measure'] = 1 - (dist_table['measure'] - dist_table['minDist']) / (dist_

    final_dist_table = dist_table.groupby(['Store1', 'Store2'])['magnitude_measure'].mean().reset_
    return final_dist_table

corr_weight = 0.5
scores = pd.DataFrame()

for metric in metric_cols:
    corr = calculate_correlation(pre_trial_data, metric, trial_store)
    mag = calculate_magnitude_distance(pre_trial_data, metric, trial_store)
    score = pd.merge(corr, mag, on=['Store1', 'Store2'])
    score[f'score_{metric}'] = corr_weight * score['corr_measure'] + (1 - corr_weight) * score['ma
    if scores.empty:
        scores = score[['Store1', 'Store2', f'score_{metric}']]
    else:
        scores = pd.merge(scores, score[['Store1', 'Store2', f'score_{metric}']], on=['Store1', 'S

scores['final_score'] = scores[[f'score_{metric}' for metric in metric_cols]].mean(axis=1)

# Select control store, ensure we are filtering the correct store (Store1 is trial store)
control_store = scores[scores['Store1'] == trial_store].sort_values('final_score', ascending=False
print(f"The control store for trial store {trial_store} is store {int(control_store)} ")
return int(control_store)

# ************************************************************************************************

def generate_driver_plots(measure_over_time, control_store, trial_store, metric_cols=['TOT_SALES', 'LY
    """
    Compare the similarities of control store to trial store.

    :param measure_over_time: DataFrame containing the store data
    :param control_store: Control store for the trial store
    :param metric_cols: List of metric columns to use for comparison
    :return: Graphs that show similarities of the control to the trial store
    """
    for metric in metric_cols:
        measure_over_time_metric = measure_over_time.copy()

        # Mapping the stores to 'Trial', 'Control', and 'Other stores'
        measure_over_time_metric['Store_type'] = measure_over_time_metric['STORE_NBR'].map(
            {trial_store: 'Trial', control_store: 'Control'}
        ).fillna('Other stores')

        # Convert 'YEARMONTH' to timestamp if it is in PeriodDtype format
        if isinstance(measure_over_time_metric['YEARMONTH'].dtype, pd.core.dtypes.dtypes.PeriodDtype):
            measure_over_time_metric['YEARMONTH'] = measure_over_time_metric['YEARMONTH'].dt.to_timest

        # Filtering the data to include only periods before March 2019
        past_metric_col = measure_over_time_metric[measure_over_time_metric['YEARMONTH'] < '2019-03']

        # Grouping by 'YEARMONTH' and 'Store_type' to calculate the mean of the metric
        past_metric_col = past_metric_col.groupby(['YEARMONTH', 'Store_type'])[metric].mean().reset_in

        # Define the color palette
        color_palette = {
            'Trial': 'green',
            'Control': 'blue',
```

```
            'Other stores': 'grey'
        }

        # Plot the data
        plt.figure(figsize=(12, 6))
        sns.lineplot(data=past_metric_col, x='YEARMONTH', y=metric, hue='Store_type', palette=color_pa

        if metric == 'TOT_SALES':
            plt.title(f'Total Monthly Sales: Trial Store {trial_store} vs Control Store {control_store
            plt.ylabel('Total sales')
        elif metric == 'LYLTY_CARD_NBR':
            plt.title(f'Monthly Customer Count: Trial Store {trial_store} vs Control Store {control_st
            plt.ylabel('Number of customers')

        plt.xlabel('Month of operation')
        plt.xticks(rotation=45)  # Rotate x-axis labels if necessary
        plt.gca().spines['top'].set_visible(False)
        plt.gca().spines['right'].set_visible(False)
        plt.show()


    # ************************************************************************************************

def analyze_trial_impact(pre_trial_measures, trial_store, control_store, measure_over_time, metric_col
    """
    Analyse the impact of the trial on the total sales and number of customers on the trial store.

    :param pre_trial_measures: DataFrame containing the store data
    :param trial_store: Store that is being tried
    :param control_store: Control store for the trial store
    :param pre_trial_measures: DataFrame containing the store data
    :param metric_cols: List of metric columns to use for comparison
    :return: analysis of significance and a more visual version of this by plotting the metrics of the
    """
    for metric in metric_cols:
        pre_trial_measures_metric = pre_trial_measures.copy()
        scaling_factor_for_metric = (
            pre_trial_measures_metric[pre_trial_measures_metric['STORE_NBR'] == trial_store][f'{metric
            pre_trial_measures_metric[pre_trial_measures_metric['STORE_NBR'] == control_store][f'{metr
        )
        measure_over_time_for_metric = measure_over_time.copy()
        scaled_control_for_metric =  measure_over_time_for_metric[measure_over_time_for_metric['STORE_
        scaled_control_for_metric['control_metric'] = scaled_control_for_metric[f'{metric}'] * scaling
        # Calculate percentage difference
        percentage_diff = pd.merge(
            scaled_control_for_metric[['YEARMONTH', 'control_metric']],
            measure_over_time_for_metric[measure_over_time_for_metric['STORE_NBR'] == trial_store][['Y
            on='YEARMONTH'
        )
        percentage_diff['percentage_diff'] = (percentage_diff[f'{metric}'] - percentage_diff['control_
        std_dev = percentage_diff[percentage_diff['YEARMONTH'] < pd.Period('2019-02')]['percentage_dif
        degrees_of_freedom = len(percentage_diff[percentage_diff['YEARMONTH'] <= pd.Period('2019-02')]
        # The trial period goes from the start of February 2019 to April 2019.
        t_values = percentage_diff[(percentage_diff['YEARMONTH'] >= pd.Period('2019-02')) &
                                   (percentage_diff['YEARMONTH'] <= pd.Period('2019-04'))].copy()
        t_values['t_calculated'] = (t_values['percentage_diff'] - 0) / std_dev
        t_values['significant'] = abs(t_values['t_calculated']) > stats.t.ppf(0.95, degrees_of_freedom
        t_values['t_critical'] = stats.t.ppf(0.95, degrees_of_freedom)
        if metric == 'TOT_SALES':
            print('Impact of Trial on Sales')
        else:
            print('Impact of Trial on Customers')
        print(t_values[['YEARMONTH', 't_calculated', 'significant', 't_critical' ]])
        # Ensure YEARMONTH is in the correct datetime format
        measure_over_time_for_metric['YEARMONTH'] = pd.to_datetime(measure_over_time_for_metric['YEARM

        # Ensure TOT_SALES is numeric
        measure_over_time_for_metric[f'{metric}'] = pd.to_numeric(measure_over_time_for_metric[f'{metr

        # Drop any rows with NaN values that might have been introduced
        measure_over_time_for_metric.dropna(subset=[f'{metric}', 'YEARMONTH'], inplace=True)

        # Copy data and label store types
```

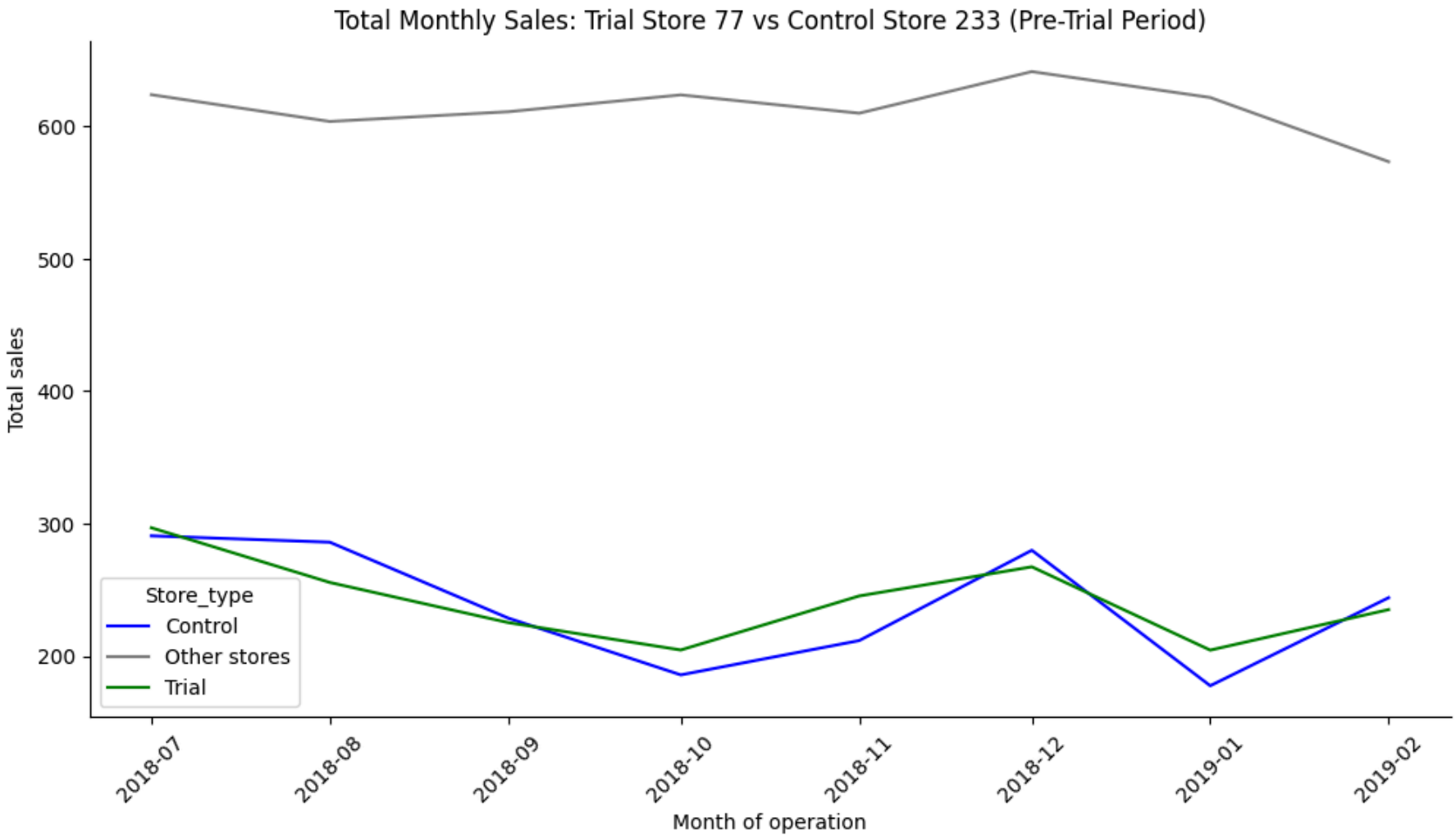
```
        past_metric = measure_over_time_for_metric.copy()
        past_metric['Store_type'] = past_metric['STORE_NBR'].apply(lambda x: 'Trial' if x == trial_sto

        # Filter only Trial and Control stores
        past_metric = past_metric[past_metric['Store_type'].isin(['Trial', 'Control'])]

        # Calculate the 95th percentile and 5th percentile for the Control store
        past_metric_control_95 = past_metric[past_metric['Store_type'] == 'Control'].copy()
        past_metric_control_95[f'{metric}'] = past_metric_control_95[f'{metric}'] * (1 + std_dev * 2)
        past_metric_control_95['Store_type'] = 'Control 95th % confidence interval'

        past_metric_control_5 = past_metric[past_metric['Store_type'] == 'Control'].copy()
        past_metric_control_5[f'{metric}'] =  past_metric_control_5[f'{metric}'] * (1 - std_dev * 2)
        past_metric_control_5['Store_type'] = 'Control 5th % confidence interval'

        # Combine data for plotting
        trial_assessment = pd.concat([past_metric, past_metric_control_95, past_metric_control_5])

        # Define color palette with alpha values
        color_palette = {
                'Trial': 'green',
                'Control': (0, 0, 1, 0.5),  # Blue with 0.3 alpha
                'Control 95th % confidence interval': (1, 0.65, 0, 0.5),  # Orange with 0.3 alpha
                'Control 5th % confidence interval': (1, 0, 0, 0.5)  # Red with 0.3 alpha
            }

        if metric == 'TOT_SALES':
            # Plot the results
            plt.figure(figsize=(15, 7))
            for store_type in trial_assessment['Store_type'].unique():
                data = trial_assessment[trial_assessment['Store_type'] == store_type]
                if store_type == 'Trial':
                    plt.plot(data['YEARMONTH'], data[metric], color=color_palette[store_type], label=s
                else:
                    plt.plot(data['YEARMONTH'], data[metric], color=color_palette[store_type], label=s

            plt.axvspan(pd.to_datetime('2019-02'), pd.to_datetime('2019-04'), alpha=0.2, color='gray')
            plt.title(f'Trial Impact On Sales Performance: Trial Store {trial_store} vs Control Store
            plt.xlabel('Month of operation')
            plt.ylabel('Total sales')
            plt.xticks(rotation=45)
            plt.legend(title='Store Type', bbox_to_anchor=(1.05, 1), loc='upper left')
            plt.tight_layout()
            plt.gca().spines['top'].set_visible(False)
            plt.gca().spines['right'].set_visible(False)
            plt.show()
        elif metric == 'LYLTY_CARD_NBR':
            # Plot the results
            plt.figure(figsize=(15, 7))
            for store_type in trial_assessment['Store_type'].unique():
                data = trial_assessment[trial_assessment['Store_type'] == store_type]
                if store_type == 'Trial':
                    plt.plot(data['YEARMONTH'], data['LYLTY_CARD_NBR'], color=color_palette[store_type
                else:
                    plt.plot(data['YEARMONTH'], data['LYLTY_CARD_NBR'], color=color_palette[store_type

            plt.axvspan(pd.to_datetime('2019-02'), pd.to_datetime('2019-04'), alpha=0.2, color='gray')
            plt.title(f'Trial Impact On Monthly Customer Count: Trial Store {trial_store} vs Control S
            plt.xlabel('Month of operation')
            plt.ylabel('Number of Customers')
            plt.xticks(rotation=45)
            plt.legend(title='Store Type', bbox_to_anchor=(1.05, 1), loc='upper left')
            plt.tight_layout()
            plt.gca().spines['top'].set_visible(False)
            plt.gca().spines['right'].set_visible(False)
            plt.show()
```

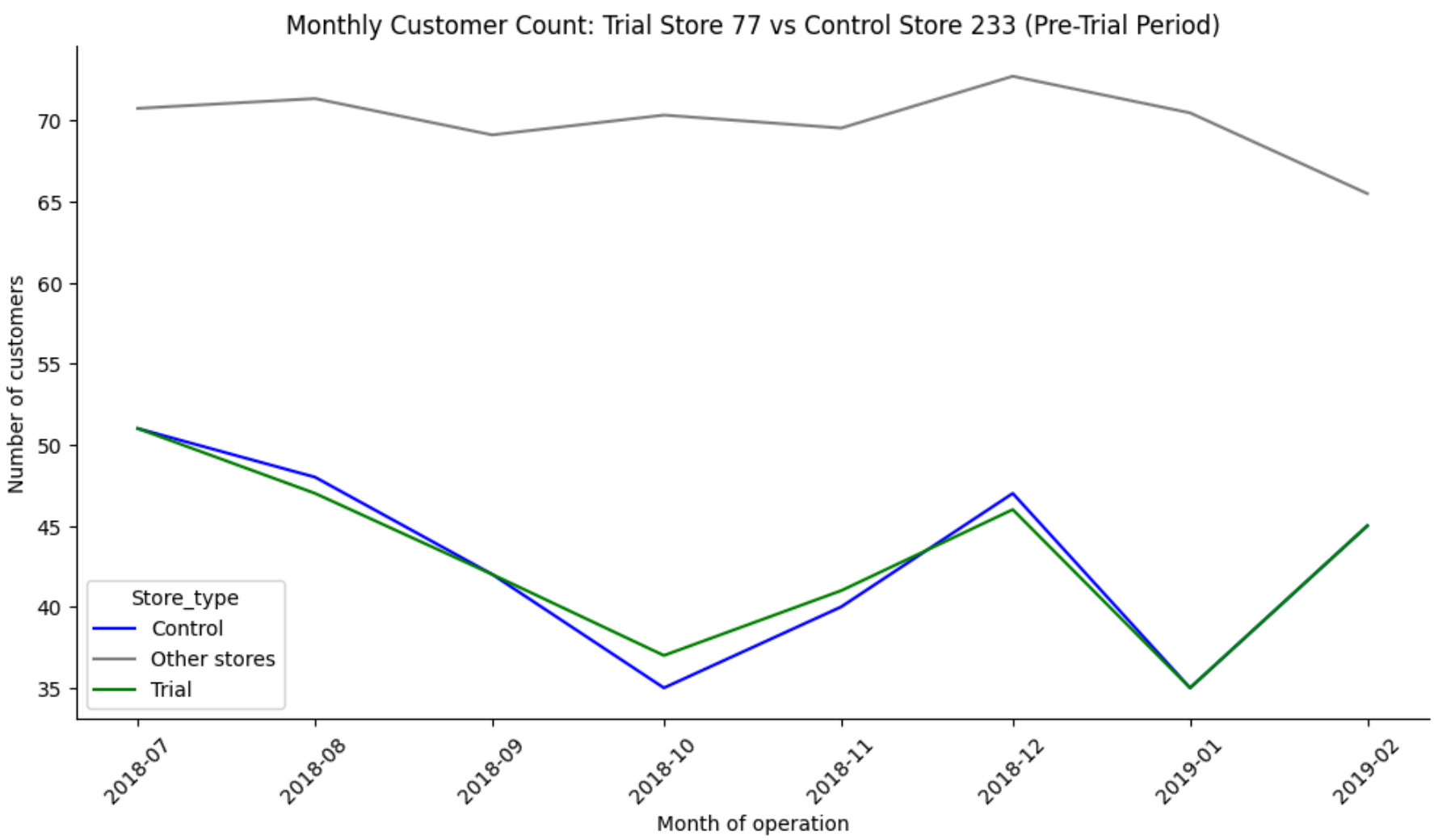
Now lets work on the other trial stores

In [ ]:
```
trial_stores = [77, 88, 86]
for trial_store in trial_stores:
    control_store = find_control_store(pre_trial_measures, trial_store)
```

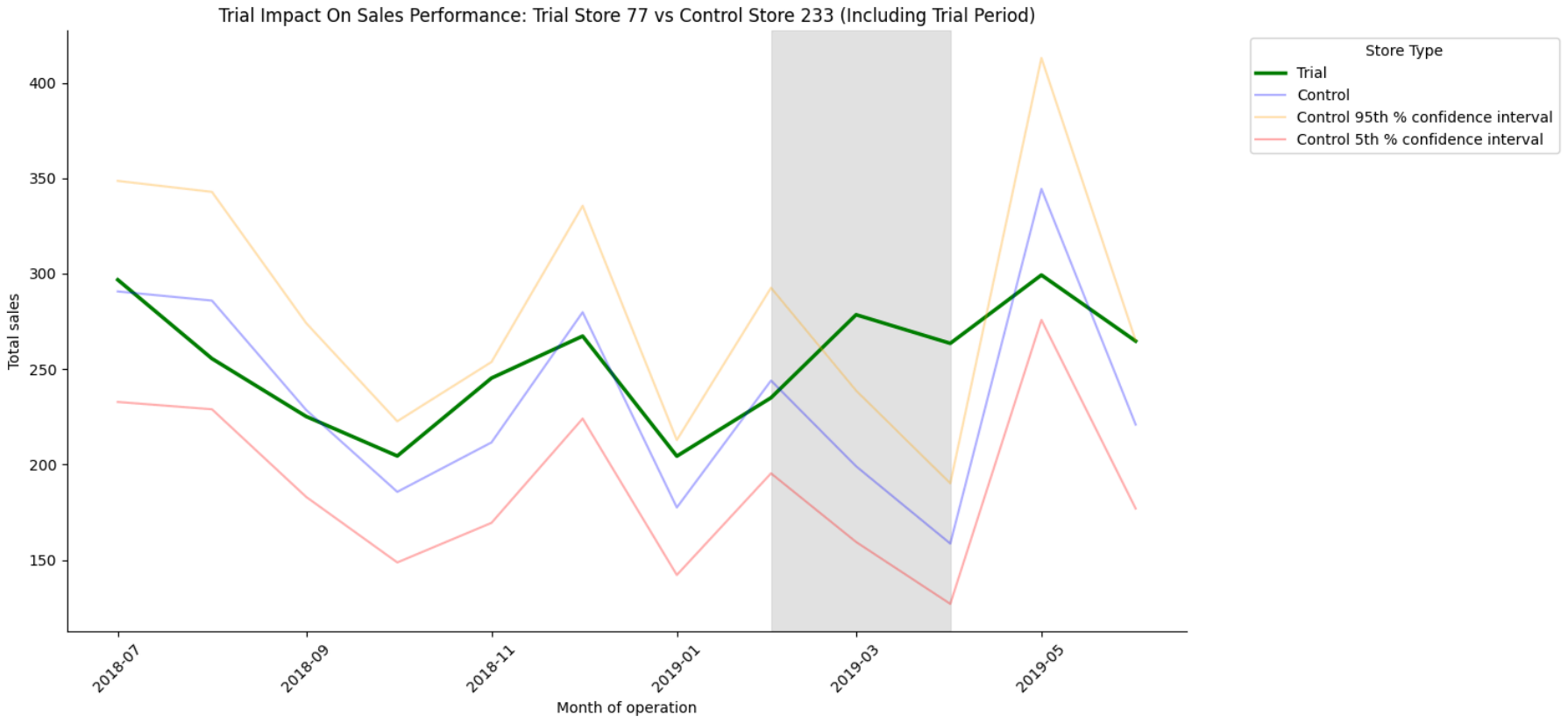
```
    generate_driver_plots(measure_over_time, control_store, trial_store)
    analyze_trial_impact(pre_trial_measures, trial_store, control_store, measure_over_time)
```

```
The control store for trial store 77 is store 233
```



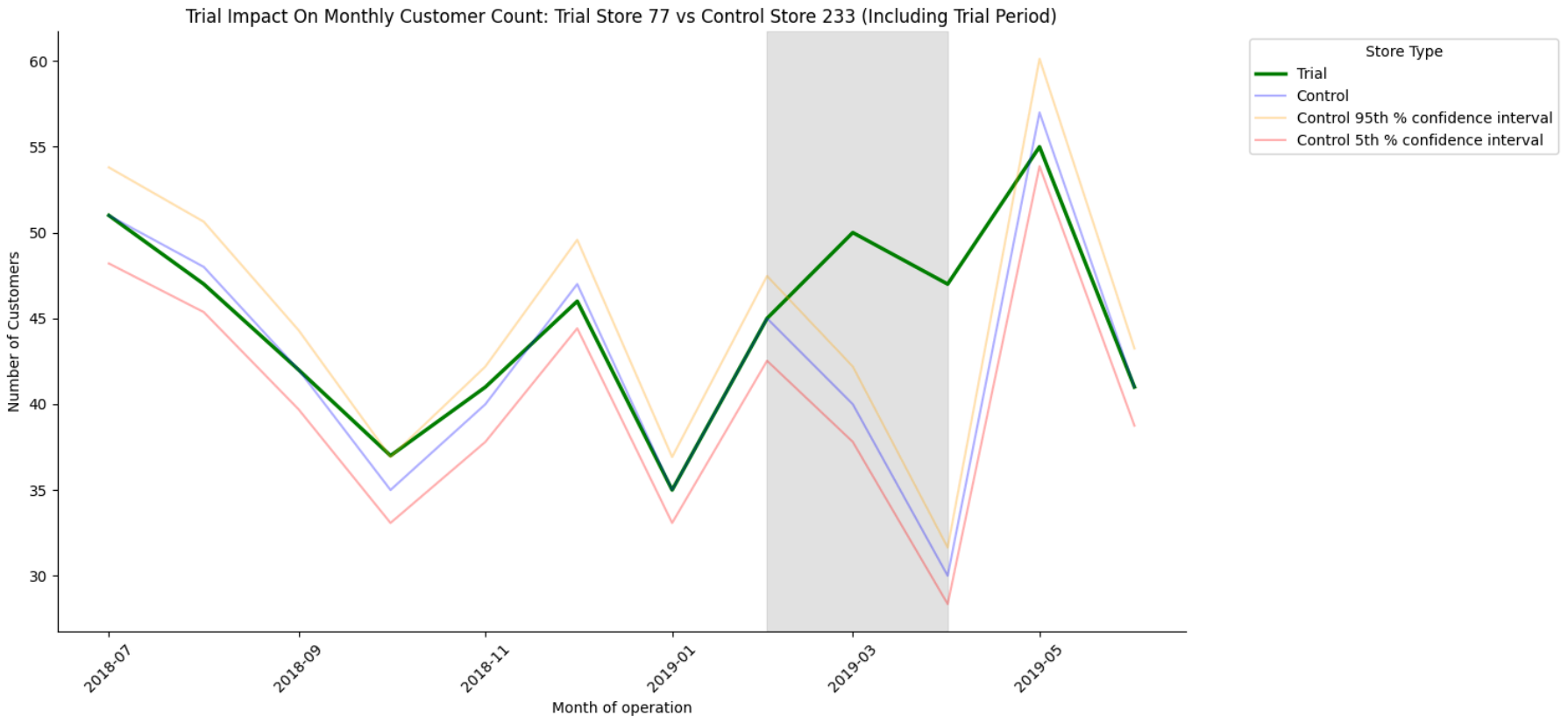



```
Impact of Trial on Sales
  YEARMONTH  t_calculated  significant  t_critical
7   2019-02     -0.593520        False    1.894579
8   2019-03      3.680430         True    1.894579
9   2019-04      6.256669         True    1.894579
```

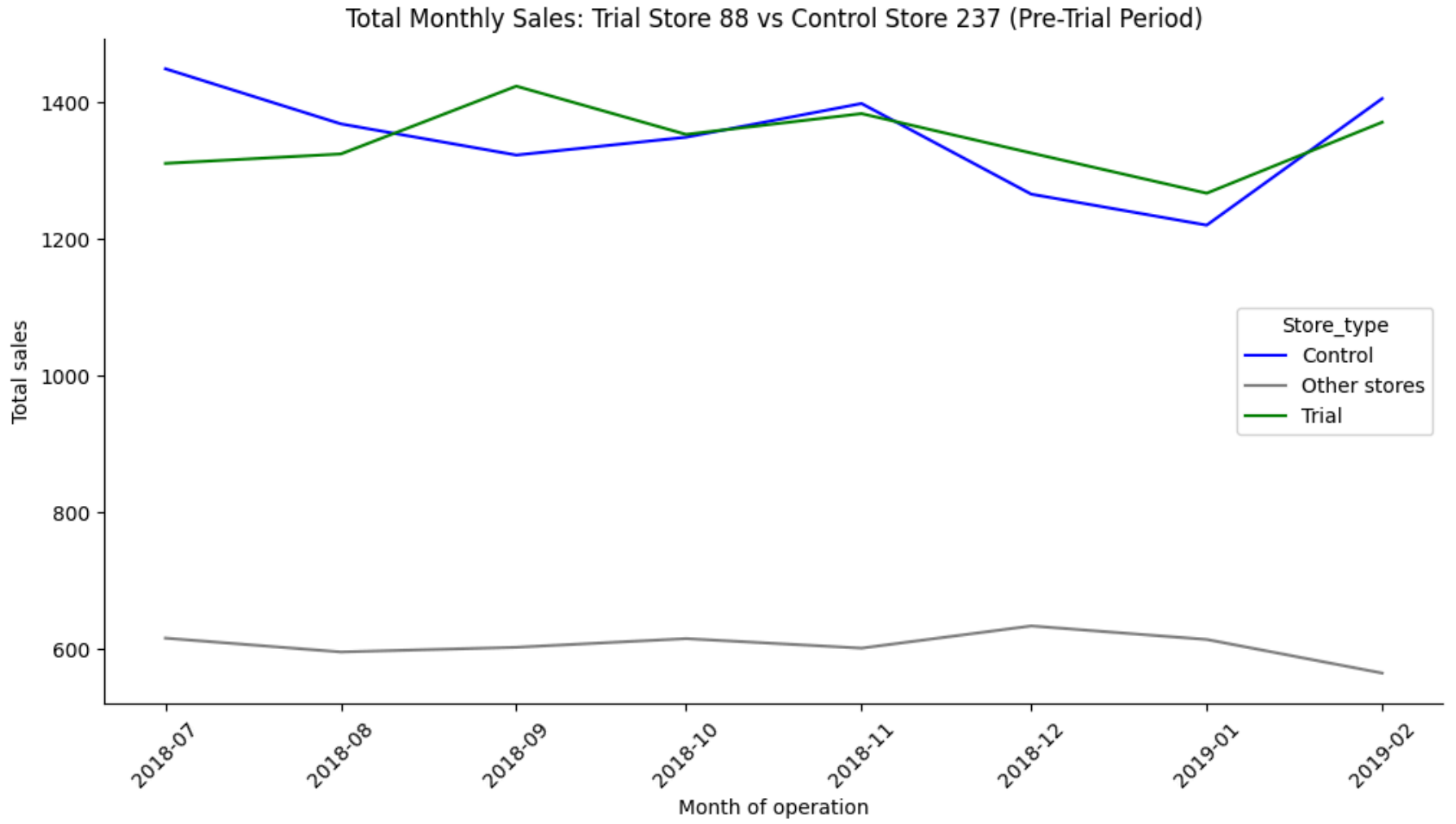


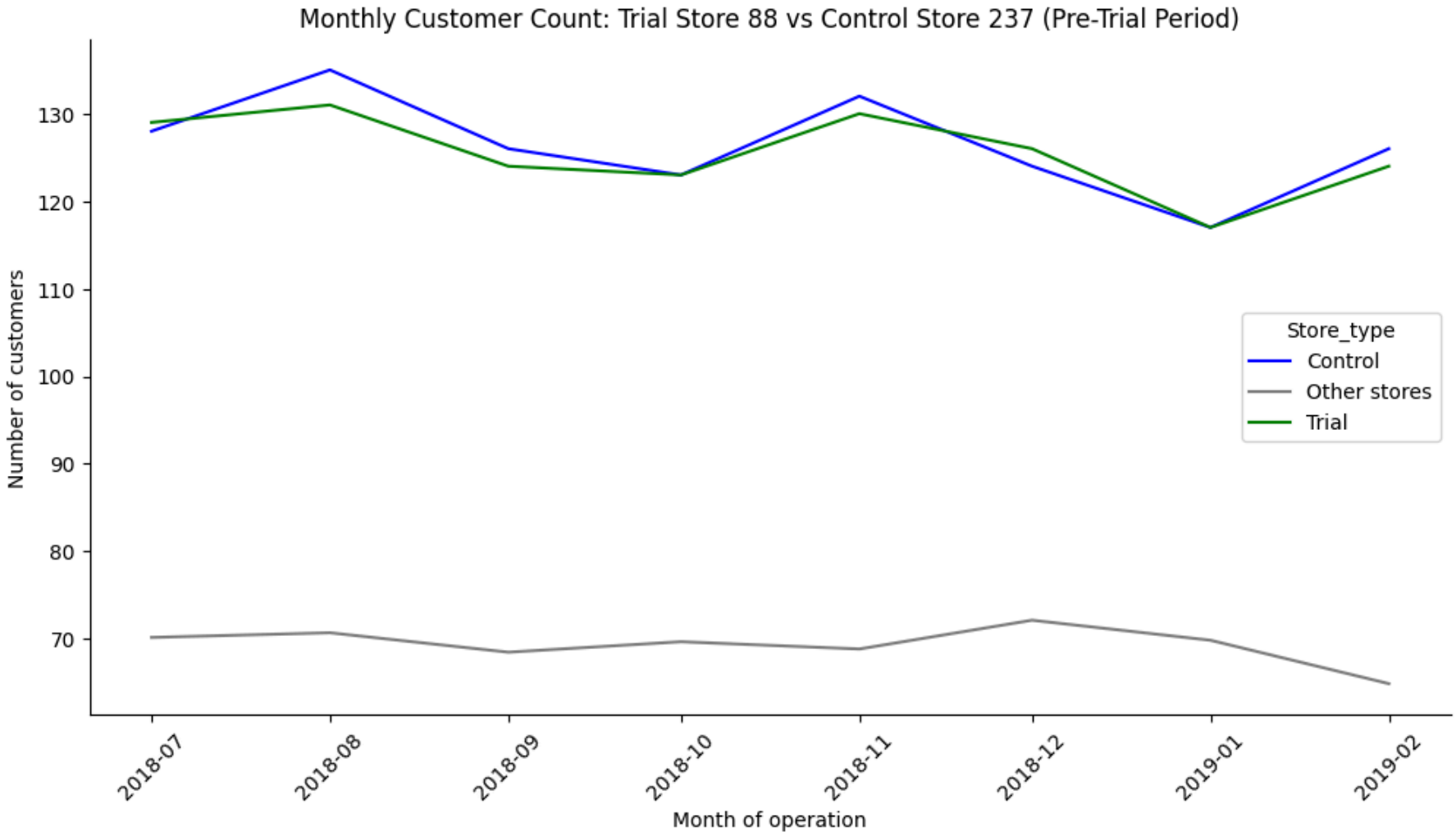
Impact of Trial on Customers

	YEARMONTH	t_calculated	significant	t_critical
7	2019-02	-0.121884	False	1.894579
8	2019-03	8.958446	True	1.894579
9	2019-04	20.460197	True	1.894579



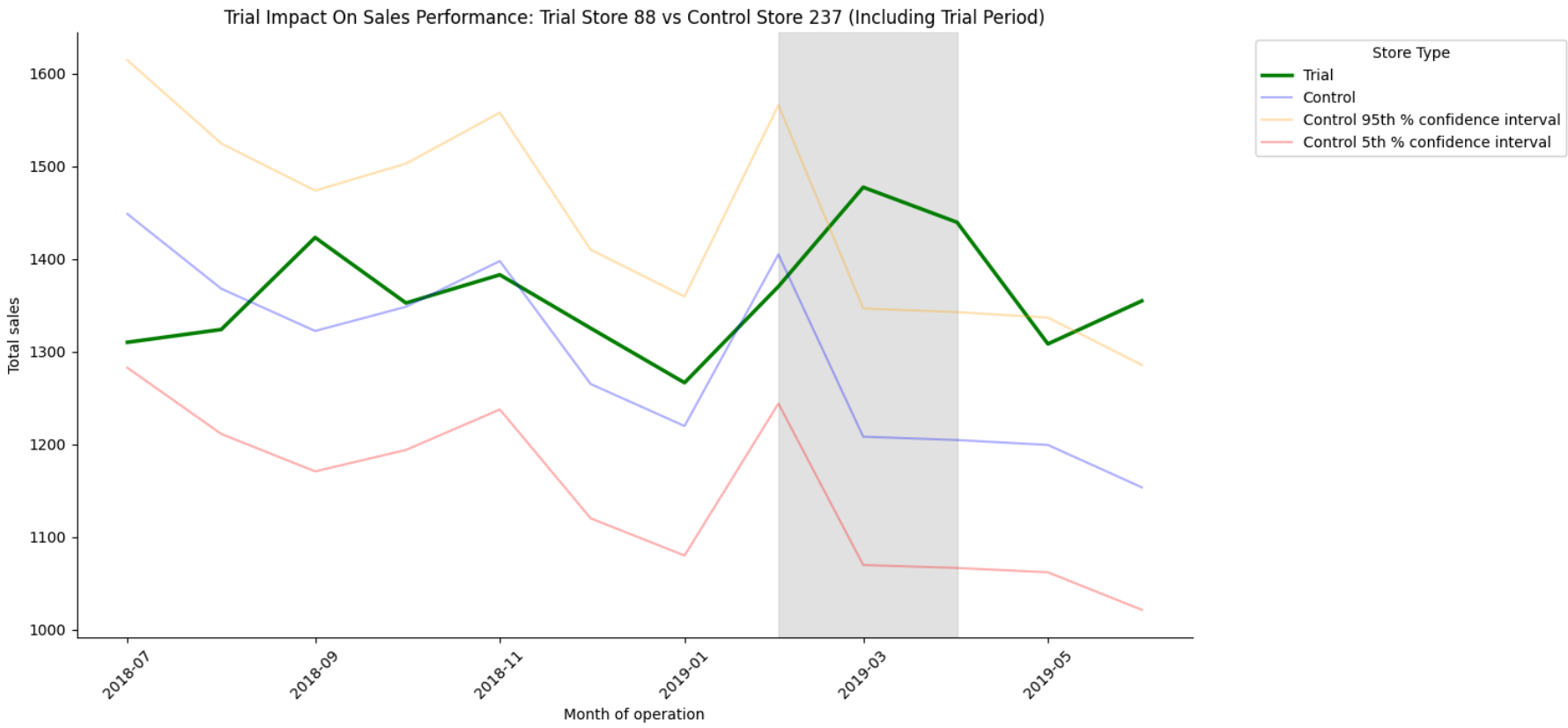
The control store for trial store 88 is store 237





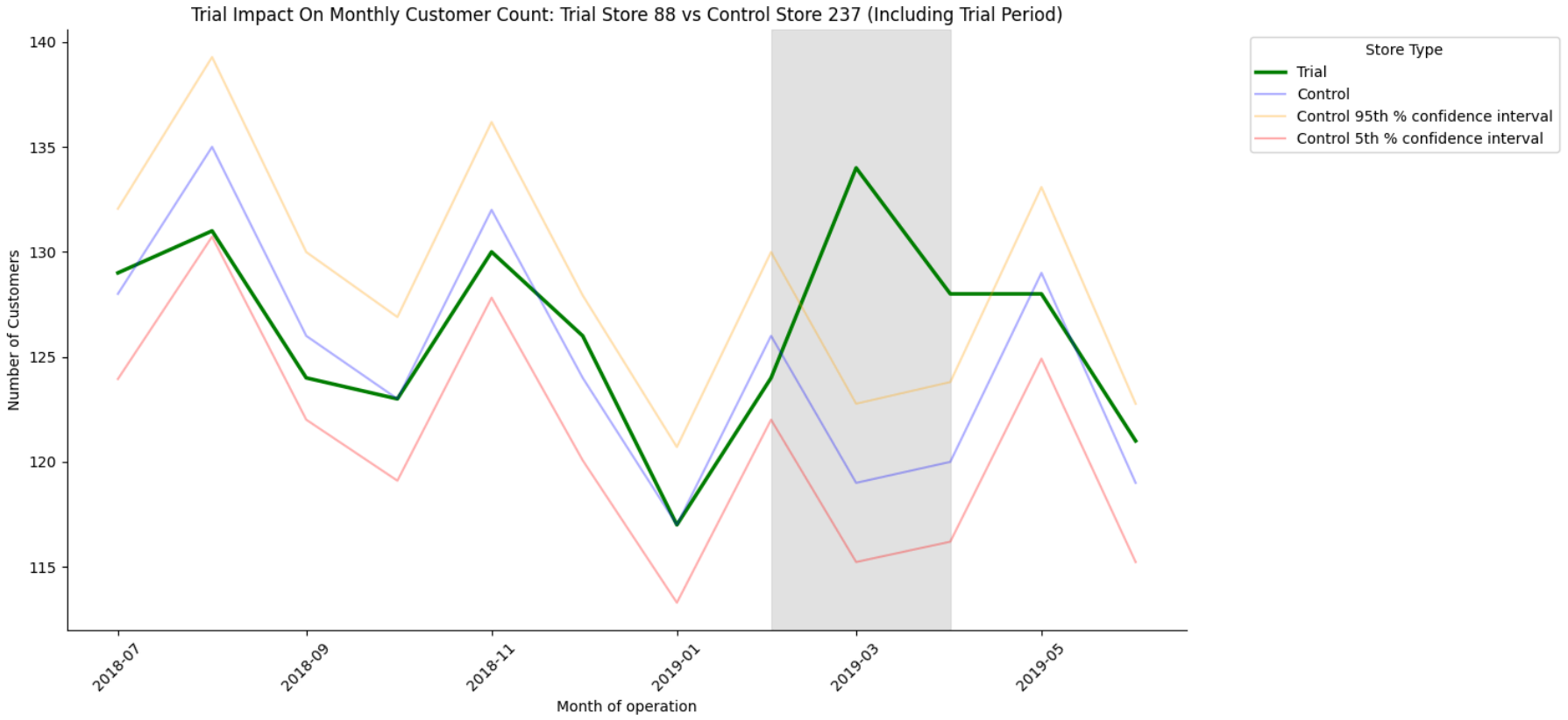
Impact of Trial on Sales

	YEARMONTH	t_calculated	significant	t_critical
7	2019-02	-0.456726	False	1.894579
8	2019-03	3.855795	True	1.894579
9	2019-04	3.372253	True	1.894579

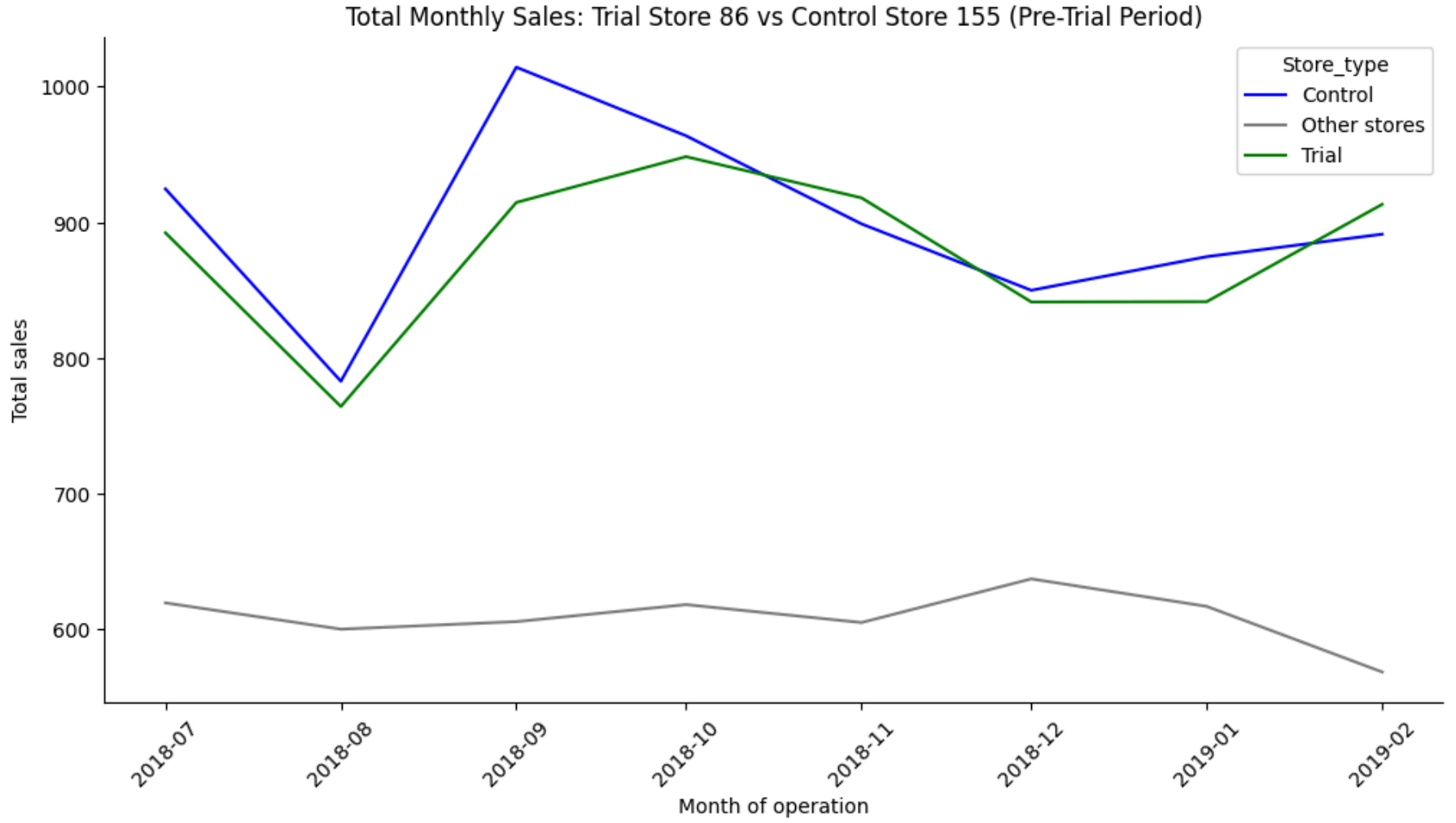


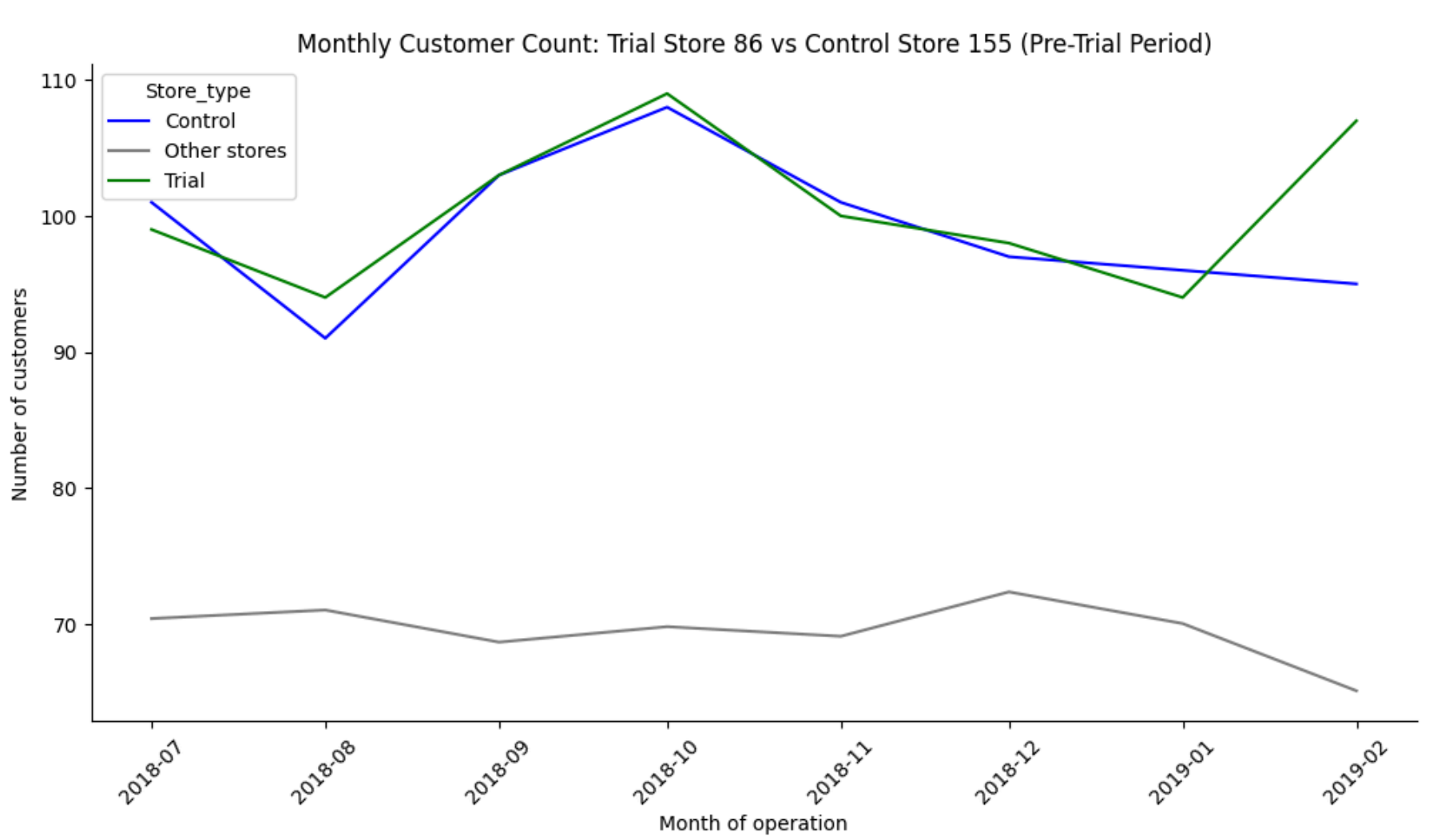
Impact of Trial on Customers

	YEARMONTH	t_calculated	significant	t_critical
7	2019-02	-0.649164	False	1.894579
8	2019-03	8.362766	True	1.894579
9	2019-04	4.591985	True	1.894579

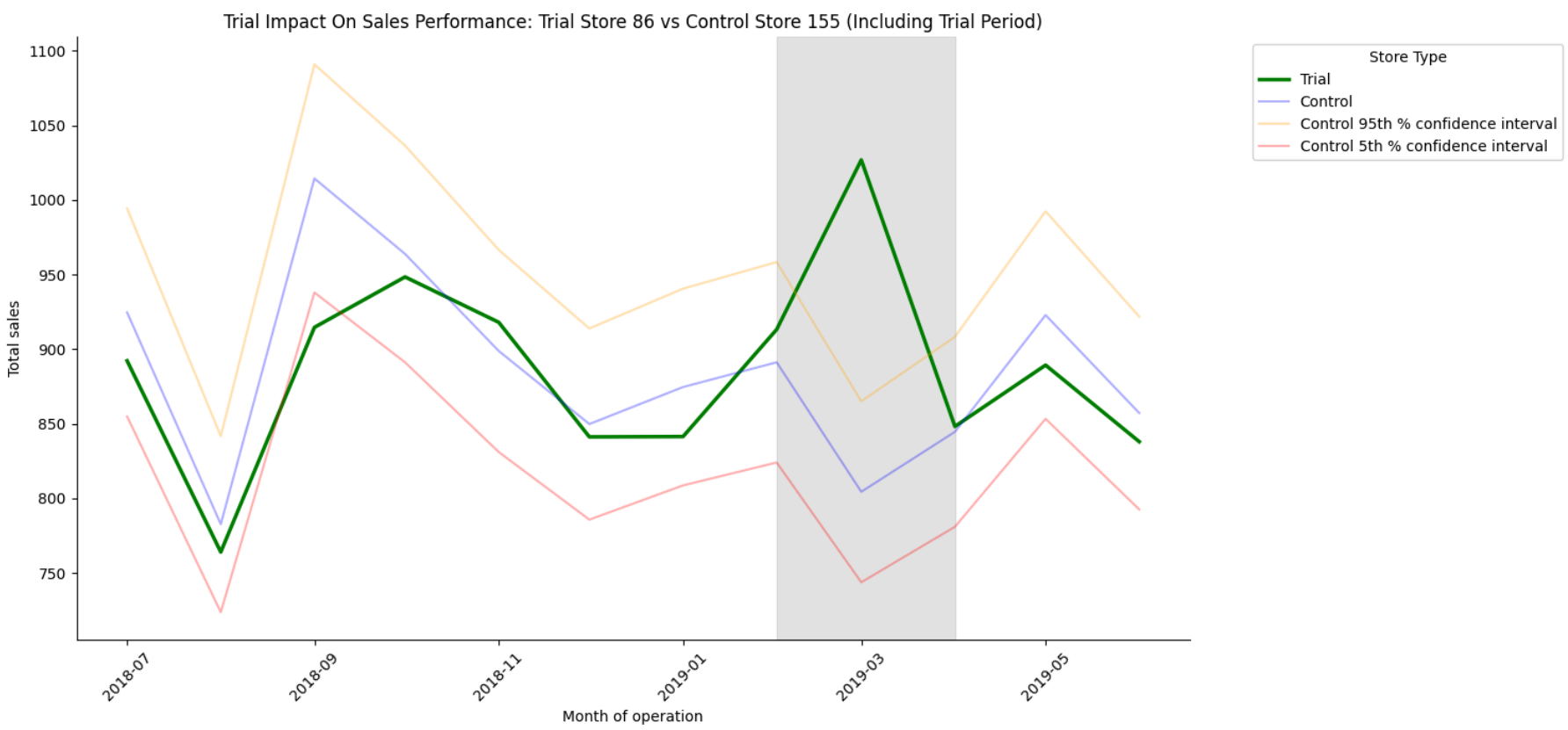


The control store for trial store 86 is store 155

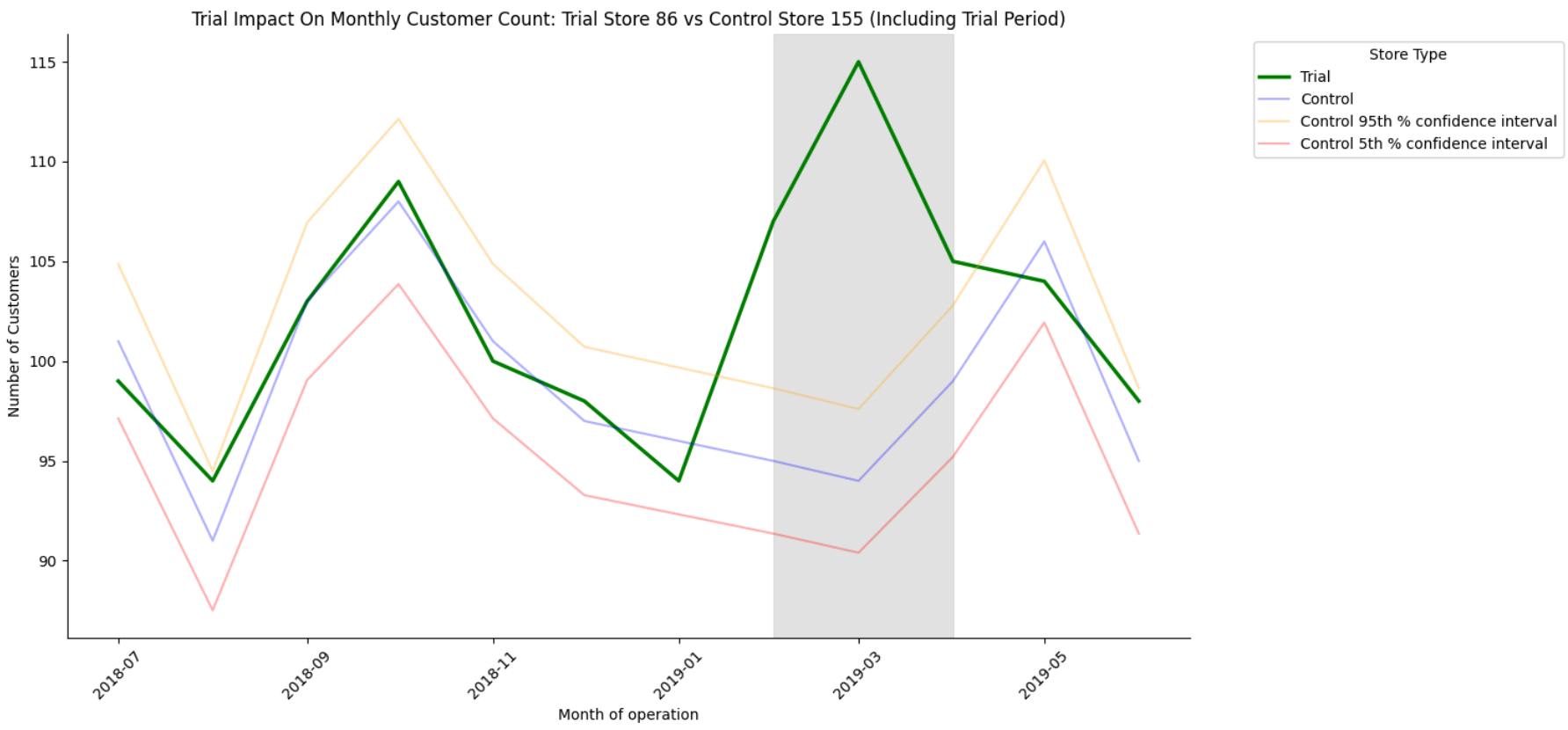




```
Impact of Trial on Sales
  YEARMONTH  t_calculated  significant  t_critical
7   2019-02      1.494114        False    1.894579
8   2019-03      8.381769         True    1.894579
9   2019-04      0.935444        False    1.894579
```




```
Impact of Trial on Customers
  YEARMONTH  t_calculated  significant  t_critical
7   2019-02      6.592964         True    1.894579
8   2019-03     11.660429         True    1.894579
9   2019-04      3.163291         True    1.894579
```



We've found control stores 233, 155, 237 for trial stores 77, 86 and 88 respectively. The results for trial stores 77 and 88 during the trial period show a significant difference in at least two of the three trial months but this is not the case for trial store 86. We can check with the client if the implementation of the trial was different in trial store 86 but overall, the trial shows a significant increase in sales.

In [ ]: