QuantiumPart2

August 25, 2024

1 Analysis of Trial and Control Stores

[193]:

```python
import pandas as pd
import numpy as np
from numpy import where
from matplotlib import pyplot
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import randint
import scipy.stats as stats
```

[194]:

```python
df = pd.read_csv('QVI_data.csv')
df.head()
```

[194]:

```
   LYLTY_CARD_NBR        DATE  STORE_NBR  TXN_ID  PROD_NBR  \
0            1000  2018-10-17          1       1         5
1            1002  2018-09-16          1       2        58
2            1003  2019-03-07          1       3        52
3            1003  2019-03-08          1       4       106
4            1004  2018-11-02          1       5        96

                                PROD_NAME  PROD_QTY  TOT_SALES  PACK_SIZE  \
0  Natural Chip        Compny SeaSalt175g         2        6.0        175
1   Red Rock Deli Chikn&Garlic Aioli 150g         1        2.7        150
2   Grain Waves Sour    Cream&Chives 210G         1        3.6        210
3  Natural ChipCo      Hony Soy Chckn175g         1        3.0        175
4          WW Original Stacked Chips 160g         1        1.9        160

        BRAND              LIFESTAGE PREMIUM_CUSTOMER
0     NATURAL  YOUNG SINGLES/COUPLES          Premium
1         RRD  YOUNG SINGLES/COUPLES       Mainstream
2     GRNWVES         YOUNG FAMILIES           Budget
3     NATURAL         YOUNG FAMILIES           Budget
4  WOOLWORTHS  OLDER SINGLES/COUPLES       Mainstream
```

[195]:

```python
# Adding new MONTH ID column in the format yyyymm
df['DATE'] = pd.to_datetime(df['DATE'])
```

```
df['Month_ID'] = df['DATE'].dt.strftime('%Y%m')
df.head()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 264834 entries, 0 to 264833
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   LYLTY_CARD_NBR    264834 non-null  int64
 1   DATE              264834 non-null  datetime64[ns]
 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
 12  Month_ID          264834 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(6), object(5)
memory usage: 26.3+ MB
```

[196]:

```
# Create a new dataframe with new metrics
# Calculate total sales for each store and month
total_sales = df.groupby(['STORE_NBR', 'Month_ID'])['TOT_SALES'].sum().
 ↪reset_index(name='Total_Sales')

# Calculate the number of unique customers for each store and month
num_customers = df.groupby(['STORE_NBR', 'Month_ID'])['LYLTY_CARD_NBR'].
 ↪nunique().reset_index(name='Number_of_Customers')

# Calculate the number of transactions for each store and month
num_transactions = df.groupby(['STORE_NBR', 'Month_ID'])['TXN_ID'].nunique().
 ↪reset_index(name='Transactions')

# Calculate the total number of chips (or units) for each store and month
total_chips = df.groupby(['STORE_NBR', 'Month_ID'])['PROD_QTY'].sum().
 ↪reset_index(name='Total_Chips')

# Calculate the average price per unit for each store and month
avg_price_per_unit = df.groupby(['STORE_NBR', 'Month_ID']).apply(lambda x:␣
 ↪(x['TOT_SALES'].sum() / x['PROD_QTY'].sum())).
 ↪reset_index(name='Avg_Price_per_Unit')
```

```python
# Merge all the calculated metrics into a single DataFrame
metrics_df = pd.merge(total_sales, num_customers, on=['STORE_NBR', 'Month_ID'])
metrics_df = pd.merge(metrics_df, num_transactions, on=['STORE_NBR',␣
 ↪'Month_ID'])
metrics_df = pd.merge(metrics_df, total_chips, on=['STORE_NBR', 'Month_ID'])
metrics_df = pd.merge(metrics_df, avg_price_per_unit, on=['STORE_NBR',␣
 ↪'Month_ID'])

# Calculate transactions per customer and chips per customer
metrics_df['Transactions_per_Customer'] = metrics_df['Transactions'] /␣
 ↪metrics_df['Number_of_Customers']
metrics_df['Chips_per_Customer'] = metrics_df['Total_Chips'] /␣
 ↪metrics_df['Number_of_Customers']
```

```
C:\Users\Alden\AppData\Local\Temp\ipykernel_21008\2744485053.py:15:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns.
This behavior is deprecated, and in a future version of pandas the grouping
columns will be excluded from the operation. Either pass `include_groups=False`
to exclude the groupings or explicitly select the grouping columns after groupby
to silence this warning.
  avg_price_per_unit = df.groupby(['STORE_NBR', 'Month_ID']).apply(lambda x:
(x['TOT_SALES'].sum() /
x['PROD_QTY'].sum())).reset_index(name='Avg_Price_per_Unit')
```

[197]:

```python
# Identify stores with full observation periods (12 months in the pre-trial␣
 ↪period)
stores_with_full_obs = metrics_df.groupby('STORE_NBR').filter(lambda x: len(x)␣
 ↪== 12)['STORE_NBR'].unique()

# Filter to the pre-trial period (before February 2019)
pre_trial_df = metrics_df[metrics_df['Month_ID'] < '201902']

# Filter the data to include only the stores with full observation periods
pre_trial_measures = pre_trial_df[pre_trial_df['STORE_NBR'].
 ↪isin(stores_with_full_obs)]
```

[198]:

```python
def calculate_correlation(input_table, metric_col, store_comparison):
    """ Calculate the correlation of a metric between a trial store and␣
 ↪potential control stores. """
    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:
            store1_data = input_table[input_table['STORE_NBR'] ==␣
 ↪store_comparison][metric_col].values
```

```
            store2_data = input_table[input_table['STORE_NBR'] ==␣
↪store][metric_col].values

            # Check if both stores have data for correlation calculation
            if len(store1_data) > 0 and len(store2_data) > 0:
                correlation_measure = pd.Series(store1_data).corr(pd.
↪Series(store2_data))

            calculated_measure = pd.DataFrame({
                'Store1': [store_comparison],
                'Store2': [store],
                'corr_measure': [correlation_measure]
            })

            calc_corr_table = pd.concat([calc_corr_table, calculated_measure],␣
↪ignore_index=True)

    return calc_corr_table
```

```
[199]: def calculate_magnitude_distance(input_table, metric_col, store_comparison):
    """Calculate the standardized magnitude distance for a measure, looping␣
↪through each control store."""
    calc_dist_table = pd.DataFrame(columns=['Store1', 'Store2', 'YEARMONTH',␣
↪'measure'])
    store_numbers = input_table['STORE_NBR'].unique()

    for store in store_numbers:
        # Calculate the magnitude distance between the trial store and each␣
↪control store
        store1_data = input_table[input_table['STORE_NBR'] ==␣
↪store_comparison][[metric_col, 'Month_ID']].reset_index(drop=True)
        store2_data = input_table[input_table['STORE_NBR'] ==␣
↪store][[metric_col, 'Month_ID']].reset_index(drop=True)

        # Ensure both stores have the same YEARMONTH for valid comparison
        if len(store1_data) > 0 and len(store2_data) > 0:
            merged_data = pd.merge(store1_data, store2_data, on='Month_ID',␣
↪suffixes=('_store1', '_store2'))
            merged_data['measure'] = np.abs(merged_data[metric_col + '_store1']␣
↪- merged_data[metric_col + '_store2'])

            calculated_measure = pd.DataFrame({
                'Store1': store_comparison,
                'Store2': store,
                'YEARMONTH': merged_data['Month_ID'],
                'measure': merged_data['measure']
            })

            calc_dist_table = pd.concat([calc_dist_table, calculated_measure],␣
 ↪ignore_index=True)

    # Calculate the minimum and maximum distances by Store1 and YEARMONTH
    min_max_dist = calc_dist_table.groupby(['Store1', 'YEARMONTH']).agg(
        minDist=('measure', 'min'),
        maxDist=('measure', 'max')
    ).reset_index()

    # Merge the min and max distances with the original distance table
    dist_table = pd.merge(calc_dist_table, min_max_dist, on=['Store1',␣
 ↪'YEARMONTH'])

    # Standardize the magnitude measure so that it ranges from 0 to 1
    dist_table['magnitudeMeasure'] = 1 - (dist_table['measure'] -␣
 ↪dist_table['minDist']) / (dist_table['maxDist'] - dist_table['minDist'])

    # Calculate the mean of the standardized magnitude measure by Store1 and␣
 ↪Store2
    final_dist_table = dist_table.groupby(['Store1', 'Store2']).agg(
        mag_measure=('magnitudeMeasure', 'mean')
    ).reset_index()

    return final_dist_table
```

2 Trial Store = 77

[200]:

```
# Calculate the correlation for the trial store 77 for its total sales and␣
 ↪number of customers
input_table = pre_trial_measures
metric_col_sales = 'Total_Sales'
metric_col_ncustomers = 'Number_of_Customers'
store_comparison = 77
corr_sales = calculate_correlation(input_table, metric_col_sales,␣
 ↪store_comparison)
corr_ncustomer = calculate_correlation(input_table, metric_col_ncustomers,␣
 ↪store_comparison)

# Calculate the magnitude distance for the trial store 77 for its total sales␣
 ↪and number of customers
magdist_sales = calculate_magnitude_distance(input_table, metric_col_sales,␣
 ↪store_comparison)
magdist_ncustomer = calculate_magnitude_distance(input_table,␣
 ↪metric_col_ncustomers, store_comparison)
```

```
C:\Users\Alden\AppData\Local\Temp\ipykernel_21008\2769831981.py:21:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  calc_corr_table = pd.concat([calc_corr_table, calculated_measure],
ignore_index=True)
C:\Users\Alden\AppData\Local\Temp\ipykernel_21008\2769831981.py:21:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  calc_corr_table = pd.concat([calc_corr_table, calculated_measure],
ignore_index=True)
C:\Users\Alden\AppData\Local\Temp\ipykernel_21008\985414664.py:23:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  calc_dist_table = pd.concat([calc_dist_table, calculated_measure],
ignore_index=True)
```

[201]:

```
# Calculates the score for control store sales
score_n_sales = pd.merge(corr_sales, magdist_sales, on=['Store1', 'Store2'])
score_n_customers = pd.merge(corr_ncustomer, magdist_ncustomer, on=['Store1',␣
  ↪'Store2'])
score_n_sales, score_n_customers

# Merge sales scores and customer scores into a single table
score_control = pd.merge(score_n_sales, score_n_customers, on=['Store1',␣
  ↪'Store2'])
score_control

# Calculate the final control score as a simple average of sales and customer␣
  ↪scores
score_control['finalControlScore'] = (0.5 * ( 0.5 *␣
  ↪score_control['corr_measure_x'] + 0.5 * score_control['mag_measure_x']) +
                                      0.5 * ( 0.5 *␣
  ↪score_control['corr_measure_y'] + 0.5 * score_control['mag_measure_y']))
```

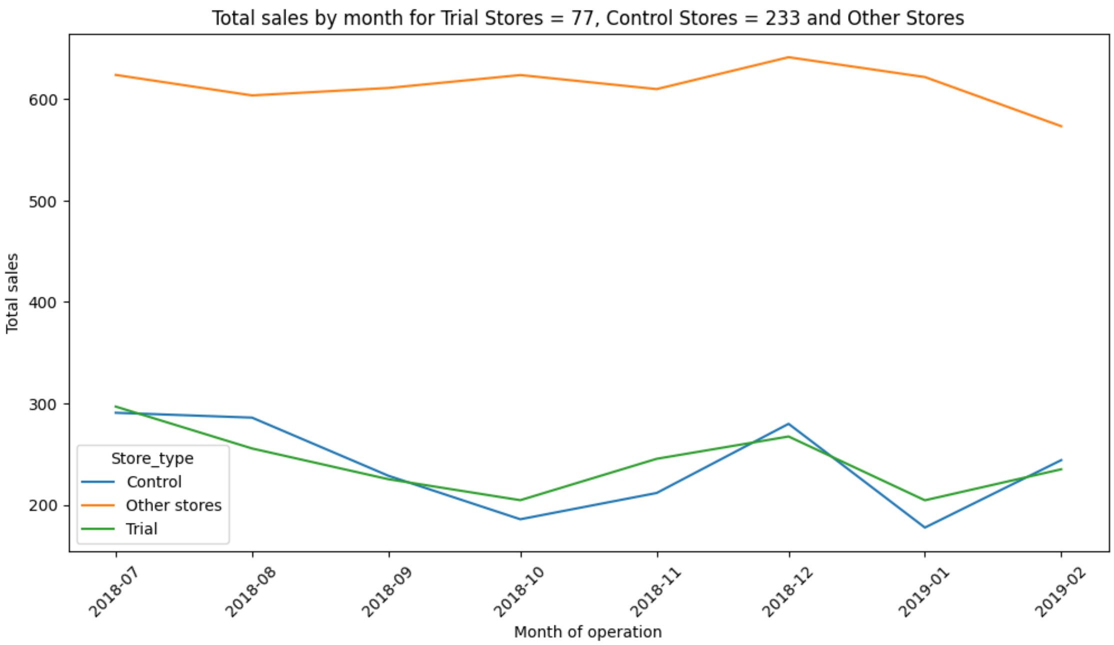
[202]:

```
# Retrieves the control score number
max_row = score_control.loc[score_control['finalControlScore'].idxmax()]
control_store = 233
```

[203]:

```
# Plot the Total sales by Month
trial_store = 77
control_store = 233
```

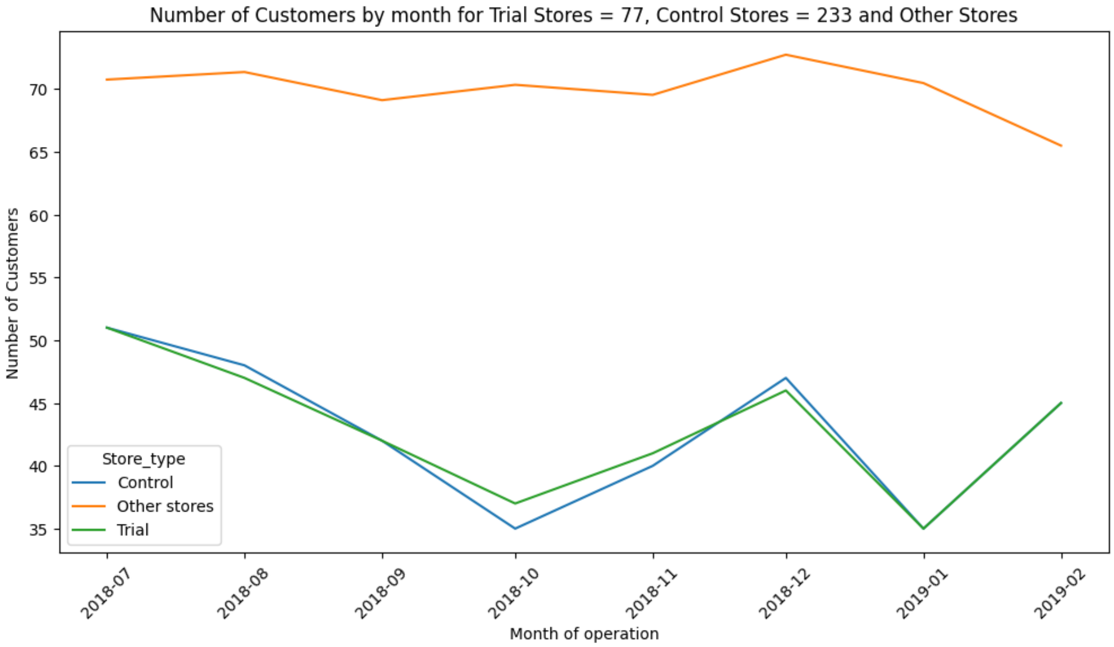
```
#  Create a 'Store_type' column to classify stores
measure_over_time_sales = metrics_df.copy()
measure_over_time_sales['Store_type'] = measure_over_time_sales['STORE_NBR'].
  ↪apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
  ↪else 'Other stores'))

# Calculate the mean of 'totSales' by 'YEARMONTH' and 'Store_type'
past_sales = measure_over_time_sales.groupby(['Month_ID', 'Store_type'],␣
  ↪as_index=False).agg({'Total_Sales': 'mean'})

# Create a 'TransactionMonth' column with the format YYYY-MM-DD
past_sales['TransactionMonth'] = pd.to_datetime(past_sales['Month_ID'].
  ↪astype(str) + '01', format='%Y%m%d')

# Filter data to only include months before March 2019
past_sales = past_sales[past_sales['Month_ID'] < '201903']

# Plotting the sales trends
plt.figure(figsize=(12, 6))
sns.lineplot(data=past_sales, x='TransactionMonth', y='Total_Sales',␣
  ↪hue='Store_type')
plt.title('Total sales by month for Trial Stores = 77, Control Stores = 233 and␣
  ↪Other Stores')
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.xticks(rotation=45)
plt.show()
```



[204]:

```
# Plot number of customer per month
trial_store = 77
control_store = 233

# Create a 'Store_type' column to classify stores
measure_over_time_custs = metrics_df.copy()
measure_over_time_custs['Store_type'] = measure_over_time_custs['STORE_NBR'].
  ↪apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
  ↪else 'Other stores'))

# Calculate the mean of 'totSales' by 'YEARMONTH' and 'Store_type'
past_custs = measure_over_time_custs.groupby(['Month_ID', 'Store_type'],␣
  ↪as_index=False).agg({'Number_of_Customers': 'mean'})

# Create a 'TransactionMonth' column with the format YYYY-MM-DD
past_custs['TransactionMonth'] = pd.to_datetime(past_sales['Month_ID'].
  ↪astype(str) + '01', format='%Y%m%d')

# Filter data to only include months before March 2019
past_custs = past_custs[past_custs['Month_ID'] < '201903']

# Plotting the sales trends
plt.figure(figsize=(12, 6))
```

```
sns.lineplot(data=past_custs, x='TransactionMonth', y='Number_of_Customers',␣
 ↪hue='Store_type')
plt.title('Number of Customers by month for Trial Stores = 77, Control Stores =␣
 ↪233 and Other Stores')
plt.xlabel('Month of operation')
plt.ylabel('Number of Customers')
plt.xticks(rotation=45)
plt.show()
```



3 Total Sales for Trial Store = 77 and Control Store = 233

[205]:

```
# Filter the pre-trial data for the trial store and control store
pre_trial_sales_trial_store =␣
 ↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == trial_store) &
                                                    ␣
 ↪(pre_trial_measures['Month_ID'] < '201902')]['Total_Sales'].sum()

pre_trial_sales_control_store =␣
 ↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == control_store) &
                                                      ␣
 ↪(pre_trial_measures['Month_ID'] < '201902')]['Total_Sales'].sum()

# Calculate the scaling factor
```

```
scaling_factor_for_control_sales = pre_trial_sales_trial_store /␣
 ↪pre_trial_sales_control_store
scaling_factor_for_control_sales
```

[205]:
```
1.023617303289553
```

[206]:
```
measure_over_time_sales = metrics_df.copy()
# Apply the scaling factor only to the rows where 'STORE_NBR' equals␣
 ↪'control_store'
scaled_control_sales =␣
 ↪measure_over_time_sales[measure_over_time_sales['STORE_NBR'] ==␣
 ↪control_store].copy()

# Create the 'controlSales' column by multiplying 'totSales' with the scaling␣
 ↪factor
scaled_control_sales['controlSales'] = scaled_control_sales['Total_Sales'] *␣
 ↪scaling_factor_for_control_sales
```

[207]:
```
# Percentage difference between the scaled control sales and the trial store's␣
 ↪sales during the trial period
trial_store_sales = metrics_df[metrics_df['STORE_NBR'] == 77]
# Merge the trial sales and control sales based on 'YEARMONTH'
percentage_diff = pd.merge(trial_store_sales[['Month_ID', 'Total_Sales']],
                           scaled_control_sales[['Month_ID', 'controlSales']],
                           on='Month_ID')

# Calculate the percentage difference
percentage_diff['percentageDiff'] = 100 * (percentage_diff['Total_Sales'] -␣
 ↪percentage_diff['controlSales']) / percentage_diff['controlSales']
```

[208]:
```
# Filter the percentage difference for the pre-trial period (YEARMONTH < 201902)
pre_trial_percentage_diff = percentage_diff[percentage_diff['Month_ID'] <␣
 ↪'201902']

# Calculate the standard deviation of the percentage difference during the␣
 ↪pre-trial period
stdDev = np.std(pre_trial_percentage_diff['percentageDiff'])

# Display the result
print(f"Standard Deviation of Percentage Difference (Pre-Trial Period):␣
 ↪{stdDev}")
```

```
Standard Deviation of Percentage Difference (Pre-Trial Period):
9.219915451817062
```

[209]:
```
# Calculate the t-values for the trial months
percentage_diff['tValue'] = percentage_diff['percentageDiff'] / stdDev
```

```

# Convert YEARMONTH to a datetime format to represent the TransactionMonth
percentage_diff['TransactionMonth'] = pd.
 ↪to_datetime(percentage_diff['Month_ID'].astype(str) + '01', format='%Y%m%d')

degrees_of_freedom = 7

# Find the 95th percentile of the t distribution
t_critical = stats.t.ppf(0.95, df=degrees_of_freedom)

# Display the calculated t-values and the critical t-value
print("T-Values during the Trial Period:")
print(percentage_diff[['TransactionMonth', 'tValue']])
print(f"Critical T-Value (95th Percentile, df={degrees_of_freedom}):␣
 ↪{t_critical}")
```

```
T-Values during the Trial Period:
   TransactionMonth    tValue
0        2018-07-01 -0.027904
1        2018-08-01 -1.376910
2        2018-09-01 -0.407839
3        2018-10-01  0.822462
4        2018-11-01  1.437278
5        2018-12-01 -0.723612
6        2019-01-01  1.355547
7        2019-02-01 -0.641075
8        2019-03-01  3.975319
9        2019-04-01  6.757975
10       2019-05-01 -1.637796
11       2019-06-01  1.844951
Critical T-Value (95th Percentile, df=7): 1.894578605061305
```

[210]:

```
# Create new dataframe for past sales
trial_store = 77
control_store = 233
measure_over_time_sales = metrics_df.copy()
# Create the Store_type column
measure_over_time_sales['Store_type'] = measure_over_time_sales['STORE_NBR'].
 ↪apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
 ↪else 'Other'))

measure_over_time_sales['Total_Sales'] = measure_over_time_sales['Total_Sales']

# Create the TransactionMonth column by converting 'YEARMONTH' to a datetime␣
 ↪format
measure_over_time_sales['TransactionMonth'] = pd.to_datetime(
    measure_over_time_sales['Month_ID'].astype(str) + '01', format='%Y%m%d')

# Filter the data to include only "Trial" and "Control" stores
pastSales = measure_over_time_sales[measure_over_time_sales['Store_type'].
 ↪isin(['Trial', 'Control'])]
```

[211]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
past_sales_controls_95 = pastSales[pastSales['Store_type'] == "Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_sales_controls_95['Total_Sales'] = past_sales_controls_95['Total_Sales'] *␣
 ↪(1 + stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
 ↪interval"
past_sales_controls_95['Store_type'] = "Control 95th % confidence interval"
```

[212]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
past_sales_controls_5 = pastSales[pastSales['Store_type'] == "Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_sales_controls_5['Total_Sales'] = past_sales_controls_5['Total_Sales'] *␣
 ↪(1 - stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
 ↪interval"
past_sales_controls_5['Store_type'] = "Control 5th % confidence interval"
```

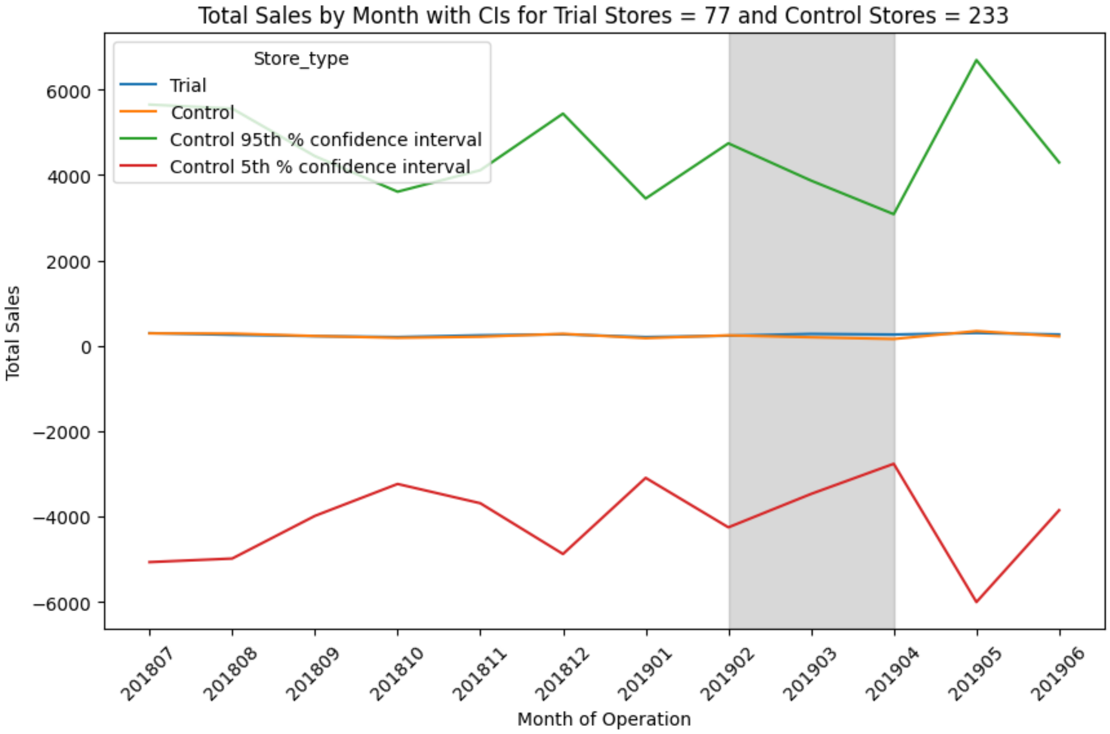
[213]:

```
# Concatenate the DataFrames
trial_assessment = pd.concat([pastSales, past_sales_controls_95,␣
 ↪past_sales_controls_5], ignore_index=True)
```

[215]:

```
# Convert 'TransactionMonth' to datetime if not already done
trial_assessment['TransactionMonth'] = pd.
 ↪to_datetime(trial_assessment['TransactionMonth'])

# Plotting
plt.figure(figsize=(10, 6))
sns.lineplot(
    data=trial_assessment,
    x='Month_ID',
    y='Total_Sales',
    hue='Store_type')
```

```
plt.axvspan(
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].min(),
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].max(),
    color='grey', alpha=0.3)

# Add labels and title
plt.xlabel('Month of Operation')
plt.ylabel('Total Sales')
plt.title('Total Sales by Month with CIs for Trial Stores = 77 and Control␣
 ↪Stores = 233')

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)
plt.show()
```



4 Number of Customers for Trial Store = 77 and Control Store = 233

[216]:

```
# Scaling Factor for Control Number for Customers

# Filter the pre-trial data for the trial store and control store
pre_trial_ncustomers_trial_store =␣
 ↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == trial_store) &
                                                     ␣
 ↪(pre_trial_measures['Month_ID'] < '201902')]['Number_of_Customers'].sum()

pre_trial_ncustomers_control_store =␣
 ↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == control_store) &
                                                       ␣
 ↪(pre_trial_measures['Month_ID'] < '201902')]['Number_of_Customers'].sum()

# Calculate the scaling factor
scaling_factor_for_control_ncustomers = pre_trial_ncustomers_trial_store /␣
 ↪pre_trial_ncustomers_control_store
scaling_factor_for_control_ncustomers
```

[216]:

```
1.0033557046979866
```

[218]:

```
measure_over_time_sales = metrics_df.copy()
# Apply the scaling factor only to the rows where 'STORE_NBR' equals␣
 ↪'control_store'
scaled_control_ncustomers =␣
 ↪measure_over_time_sales[measure_over_time_sales['STORE_NBR'] ==␣
 ↪control_store].copy()

scaled_control_ncustomers['controlNumCustomers'] =␣
 ↪scaled_control_ncustomers['Number_of_Customers'] *␣
 ↪scaling_factor_for_control_ncustomers
```

[219]:

```
# Percentage difference between the scaled control num customers and the trial␣
 ↪store's num customers during the trial period
trial_store_ncustomers = metrics_df[metrics_df['STORE_NBR'] == 77]
# Merge the trial num customers and control num customers based on 'YEARMONTH'
percentage_diff = pd.merge(trial_store_ncustomers[['Month_ID',␣
 ↪'Number_of_Customers']],
                           scaled_control_ncustomers[['Month_ID',␣
 ↪'controlNumCustomers']],
                           on='Month_ID')

# Calculate the percentage difference
```

```
percentage_diff['percentageDiff'] = 100 *␣
  ↪(percentage_diff['Number_of_Customers'] -␣
  ↪percentage_diff['controlNumCustomers']) /␣
  ↪percentage_diff['controlNumCustomers']
```

[220]:

```
# Filter the percentage difference for the pre-trial period (YEARMONTH < 201902)
pre_trial_percentage_diff = percentage_diff[percentage_diff['Month_ID'] <␣
  ↪'201902']

# Calculate the standard deviation of the percentage difference during the␣
  ↪pre-trial period
stdDev = np.std(pre_trial_percentage_diff['percentageDiff'])

# Display the result
print(f"Standard Deviation of Percentage Difference (Pre-Trial Period):␣
  ↪{stdDev}")
```

```
Standard Deviation of Percentage Difference (Pre-Trial Period):
2.540446658899978
```

[221]:

```
# Calculate the t-values for the trial months
percentage_diff['tValue'] = percentage_diff['percentageDiff'] / stdDev

# Convert YEARMONTH to a datetime format to represent the TransactionMonth
percentage_diff['TransactionMonth'] = pd.
  ↪to_datetime(percentage_diff['Month_ID'].astype(str) + '01', format='%Y%m%d')

degrees_of_freedom = 7

# Find the 95th percentile of the t distribution
t_critical = stats.t.ppf(0.95, df=degrees_of_freedom)

# Display the calculated t-values and the critical t-value
print("T-Values during the Trial Period:")
print(percentage_diff[['TransactionMonth', 'tValue']])
print(f"Critical T-Value (95th Percentile, df={degrees_of_freedom}):␣
  ↪{t_critical}")
```

```
T-Values during the Trial Period:
   TransactionMonth    tValue
0        2018-07-01 -0.131649
1        2018-08-01 -0.948972
2        2018-09-01 -0.131649
3        2018-10-01  2.110151
4        2018-11-01  0.849138
5        2018-12-01 -0.966362
6        2019-01-01 -0.131649
7        2019-02-01 -0.131649
```

```
8        2019-03-01   9.676227
9        2019-04-01  22.099538
10       2019-05-01  -1.508193
11       2019-06-01  -0.131649
Critical T-Value (95th Percentile, df=7): 1.894578605061305
```

[222]:

```
# Create new dataframe for past customers
trial_store = 77
control_store = 233
measure_over_time_ncustomers = metrics_df.copy()
# Create the Store_type column
measure_over_time_ncustomers['Store_type'] =␣
 ↪measure_over_time_ncustomers['STORE_NBR'].apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
 ↪else 'Other'))

measure_over_time_ncustomers['Number_of_Customers'] =␣
 ↪measure_over_time_ncustomers['Number_of_Customers']

# Create the TransactionMonth column by converting 'YEARMONTH' to a datetime␣
 ↪format
measure_over_time_ncustomers['TransactionMonth'] = pd.to_datetime(
    measure_over_time_ncustomers['Month_ID'].astype(str) + '01',␣
 ↪format='%Y%m%d')

# Filter the data to include only "Trial" and "Control" stores
pastnCustomers =␣
 ↪measure_over_time_ncustomers[measure_over_time_ncustomers['Store_type'].
 ↪isin(['Trial', 'Control'])]
```

[223]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
past_ncustomers_controls_95 = pastnCustomers[pastnCustomers['Store_type'] ==␣
 ↪"Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_ncustomers_controls_95['Number_of_Customers'] =␣
 ↪past_ncustomers_controls_95['Number_of_Customers'] * (1 + stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
 ↪interval"
past_ncustomers_controls_95['Store_type'] = "Control 95th % confidence interval"
```

[224]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
```

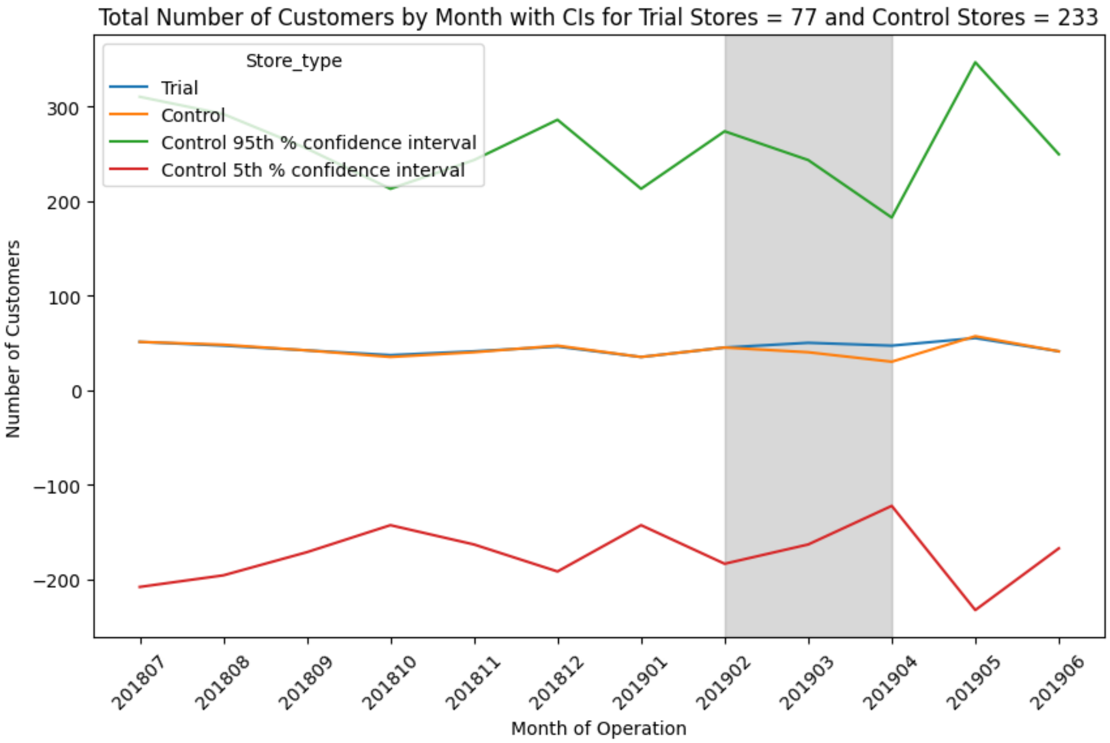
```
past_ncustomers_controls_5 = pastnCustomers[pastnCustomers['Store_type'] ==␣
 ↪"Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_ncustomers_controls_5['Number_of_Customers'] =␣
 ↪past_ncustomers_controls_5['Number_of_Customers'] * (1 - stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
 ↪interval"
past_ncustomers_controls_5['Store_type'] = "Control 5th % confidence interval"
```

[225]:

```
# Concatenate the DataFrames
trial_assessment = pd.concat([pastnCustomers, past_ncustomers_controls_95,␣
 ↪past_ncustomers_controls_5], ignore_index=True)
```

[227]:

```
# Convert 'TransactionMonth' to datetime if not already done
trial_assessment['TransactionMonth'] = pd.
 ↪to_datetime(trial_assessment['TransactionMonth'])

# Plotting
plt.figure(figsize=(10, 6))
sns.lineplot(
    data=trial_assessment,
    x='Month_ID',
    y='Number_of_Customers',
    hue='Store_type')

plt.axvspan(
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].min(),
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].max(),
    color='grey', alpha=0.3)

# Add labels and title
plt.xlabel('Month of Operation')
plt.ylabel('Number of Customers')
plt.title('Total Number of Customers by Month with CIs for Trial Stores = 77␣
 ↪and Control Stores = 233')
plt.xticks(rotation=45)
plt.show()
```



5 Trial Store 86

[228]:

```
# Calculate the correlation for the trial store 86 for its total sales and␣
  ↪number of customers
input_table = pre_trial_measures
metric_col_sales = 'Total_Sales'
metric_col_ncustomers = 'Number_of_Customers'
store_comparison = 86
corr_sales = calculate_correlation(input_table, metric_col_sales,␣
  ↪store_comparison)
corr_ncustomer = calculate_correlation(input_table, metric_col_ncustomers,␣
  ↪store_comparison)

# Calculate the magnitude distance for the trial store 86 for its total sales␣
  ↪and number of customers
magdist_sales = calculate_magnitude_distance(input_table, metric_col_sales,␣
  ↪store_comparison)
magdist_ncustomer = calculate_magnitude_distance(input_table,␣
  ↪metric_col_ncustomers, store_comparison)
```

```
C:\Users\Alden\AppData\Local\Temp\ipykernel_21008\2769831981.py:21:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
```

```
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  calc_corr_table = pd.concat([calc_corr_table, calculated_measure],
ignore_index=True)
C:\Users\Alden\AppData\Local\Temp\ipykernel_21008\2769831981.py:21:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  calc_corr_table = pd.concat([calc_corr_table, calculated_measure],
ignore_index=True)
C:\Users\Alden\AppData\Local\Temp\ipykernel_21008\985414664.py:23:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  calc_dist_table = pd.concat([calc_dist_table, calculated_measure],
ignore_index=True)
```

[230]:

```
# Calculates the score for control store sales
score_n_sales = pd.merge(corr_sales, magdist_sales, on=['Store1', 'Store2'])
score_n_customers = pd.merge(corr_ncustomer, magdist_ncustomer, on=['Store1',␣
 ↪'Store2'])
score_n_sales, score_n_customers

# Merge sales scores and customer scores into a single table
score_control = pd.merge(score_n_sales, score_n_customers, on=['Store1',␣
 ↪'Store2'])
score_control

# Calculate the final control score as a simple average of sales and customer␣
 ↪scores
score_control['finalControlScore'] = (0.5 * ( 0.5 *␣
 ↪score_control['corr_measure_x'] + 0.5 * score_control['mag_measure_x']) +
                                      0.5 * ( 0.5 *␣
 ↪score_control['corr_measure_y'] + 0.5 * score_control['mag_measure_y']))
```

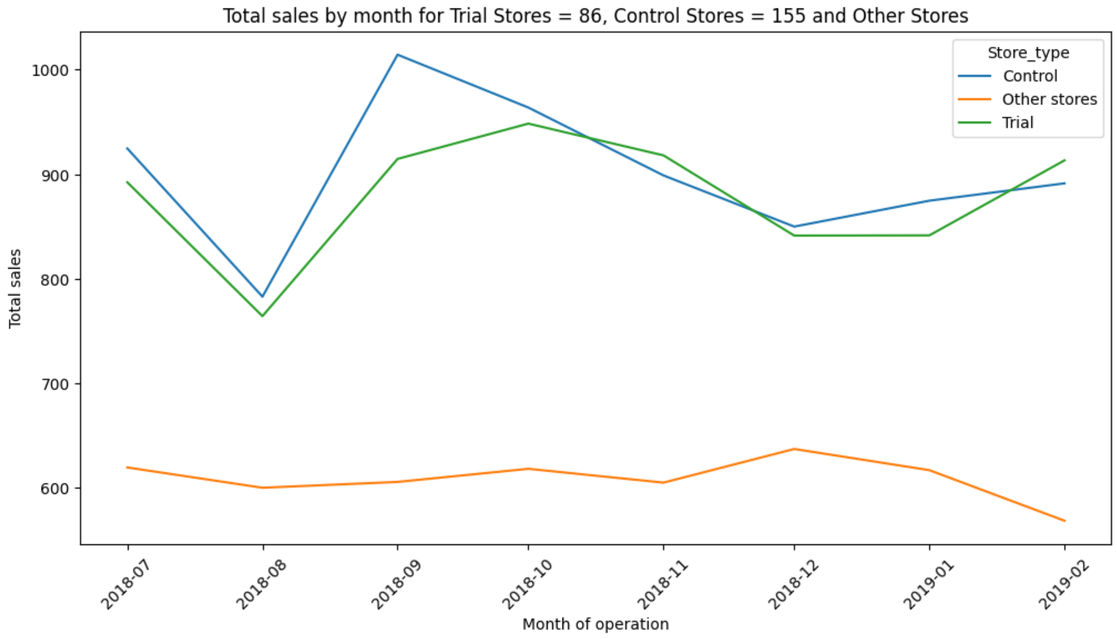
[231]:

```
# Retrieves the control score number
max_row = score_control.loc[score_control['finalControlScore'].idxmax()]
control_store = 155
```

[233]:

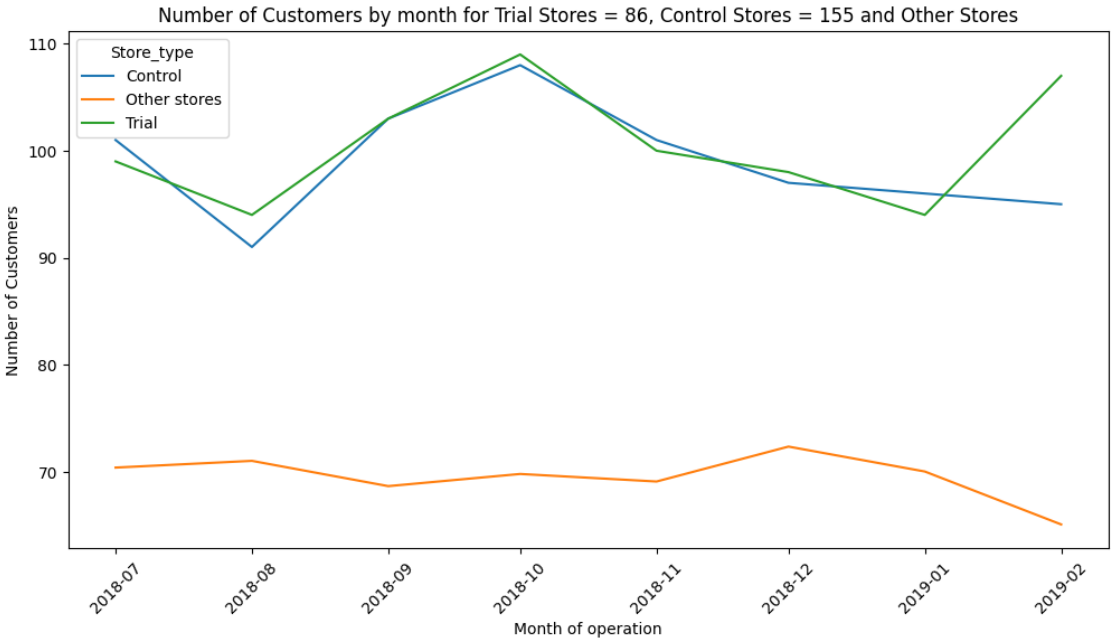
```
# Plot the Total sales by Month
trial_store = 86
control_store = 155


#  Create a 'Store_type' column to classify stores
```

```
measure_over_time_sales = metrics_df.copy()
measure_over_time_sales['Store_type'] = measure_over_time_sales['STORE_NBR'].apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store else 'Other stores'))

# Calculate the mean of 'totSales' by 'YEARMONTH' and 'Store_type'
past_sales = measure_over_time_sales.groupby(['Month_ID', 'Store_type'], as_index=False).agg({'Total_Sales': 'mean'})

# Create a 'TransactionMonth' column with the format YYYY-MM-DD
past_sales['TransactionMonth'] = pd.to_datetime(past_sales['Month_ID'].astype(str) + '01', format='%Y%m%d')

# Filter data to only include months before March 2019
past_sales = past_sales[past_sales['Month_ID'] < '201903']

# Plotting the sales trends
plt.figure(figsize=(12, 6))
sns.lineplot(data=past_sales, x='TransactionMonth', y='Total_Sales', hue='Store_type')
plt.title('Total sales by month for Trial Stores = 86, Control Stores = 155 and Other Stores')
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.xticks(rotation=45)
plt.show()
```



[235]:

```
# Plot number of customer per month
trial_store = 86
control_store = 155

# Create a 'Store_type' column to classify stores
measure_over_time_custs = metrics_df.copy()
measure_over_time_custs['Store_type'] = measure_over_time_custs['STORE_NBR'].
  ↪apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
  ↪else 'Other stores'))

# Calculate the mean of 'totSales' by 'YEARMONTH' and 'Store_type'
past_custs = measure_over_time_custs.groupby(['Month_ID', 'Store_type'],␣
  ↪as_index=False).agg({'Number_of_Customers': 'mean'})

# Create a 'TransactionMonth' column with the format YYYY-MM-DD
past_custs['TransactionMonth'] = pd.to_datetime(past_sales['Month_ID'].
  ↪astype(str) + '01', format='%Y%m%d')

# Filter data to only include months before March 2019
past_custs = past_custs[past_custs['Month_ID'] < '201903']

# Plotting the sales trends
plt.figure(figsize=(12, 6))
sns.lineplot(data=past_custs, x='TransactionMonth', y='Number_of_Customers',␣
  ↪hue='Store_type')
plt.title('Number of Customers by month for Trial Stores = 86, Control Stores =␣
  ↪155 and Other Stores')
plt.xlabel('Month of operation')
plt.ylabel('Number of Customers')
plt.xticks(rotation=45)
plt.show()
```



6 Total Sales for Trial Store 86 and Control Store 155

[241]:

```
# Filter the pre-trial data for the trial store and control store
pre_trial_sales_trial_store =␣
 ↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == trial_store) &
                                                      ␣
 ↪(pre_trial_measures['Month_ID'] < '201902')]['Total_Sales'].sum()

pre_trial_sales_control_store =␣
 ↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == control_store) &
                                                        ␣
 ↪(pre_trial_measures['Month_ID'] < '201902')]['Total_Sales'].sum()

# Calculate the scaling factor
scaling_factor_for_control_sales = pre_trial_sales_trial_store /␣
 ↪pre_trial_sales_control_store
scaling_factor_for_control_sales
```

[241]:

```
0.9700651481287743
```

[242]:

```
measure_over_time_sales = metrics_df.copy()
# Apply the scaling factor only to the rows where 'STORE_NBR' equals␣
 ↪'control_store'
```

```
scaled_control_sales =␣
 ↪measure_over_time_sales[measure_over_time_sales['STORE_NBR'] ==␣
 ↪control_store].copy()

# Create the 'controlSales' column by multiplying 'totSales' with the scaling␣
 ↪factor
scaled_control_sales['controlSales'] = scaled_control_sales['Total_Sales'] *␣
 ↪scaling_factor_for_control_sales
```

[243]:

```
# Percentage difference between the scaled control sales and the trial store's␣
 ↪sales during the trial period
trial_store_sales = metrics_df[metrics_df['STORE_NBR'] == 86]
# Merge the trial sales and control sales based on 'YEARMONTH'
percentage_diff = pd.merge(trial_store_sales[['Month_ID', 'Total_Sales']],
                           scaled_control_sales[['Month_ID', 'controlSales']],
                           on='Month_ID')

# Calculate the percentage difference
percentage_diff['percentageDiff'] = 100 * (percentage_diff['Total_Sales'] -␣
 ↪percentage_diff['controlSales']) / percentage_diff['controlSales']
```

[244]:

```
# Filter the percentage difference for the pre-trial period (YEARMONTH < 201902)
pre_trial_percentage_diff = percentage_diff[percentage_diff['Month_ID'] <␣
 ↪'201902']

# Calculate the standard deviation of the percentage difference during the␣
 ↪pre-trial period
stdDev = np.std(pre_trial_percentage_diff['percentageDiff'])

# Display the result
print(f"Standard Deviation of Percentage Difference (Pre-Trial Period):␣
 ↪{stdDev}")
```

```
Standard Deviation of Percentage Difference (Pre-Trial Period):
3.4889834036416865
```

[245]:

```
# Calculate the t-values for the trial months
percentage_diff['tValue'] = percentage_diff['percentageDiff'] / stdDev

# Convert YEARMONTH to a datetime format to represent the TransactionMonth
percentage_diff['TransactionMonth'] = pd.
 ↪to_datetime(percentage_diff['Month_ID'].astype(str) + '01', format='%Y%m%d')

degrees_of_freedom = 7

# Find the 95th percentile of the t distribution
t_critical = stats.t.ppf(0.95, df=degrees_of_freedom)
```

```
# Display the calculated t-values and the critical t-value
print("T-Values during the Trial Period:")
print(percentage_diff[['TransactionMonth', 'tValue']])
print(f"Critical T-Value (95th Percentile, df={degrees_of_freedom}):␣
  ↪{t_critical}")
```

```
T-Values during the Trial Period:
   TransactionMonth    tValue
0        2018-07-01 -0.150902
1        2018-08-01  0.180440
2        2018-09-01 -2.022384
3        2018-10-01  0.412358
4        2018-11-01  1.515617
5        2018-12-01  0.585451
6        2019-01-01 -0.237118
7        2019-02-01  1.613828
8        2019-03-01  9.053346
9        2019-04-01  1.010395
10       2019-05-01 -0.189684
11       2019-06-01  0.222670
Critical T-Value (95th Percentile, df=7): 1.894578605061305
```

[247]:

```
# Create new dataframe for past sales
trial_store = 86
control_store = 155
measure_over_time_sales = metrics_df.copy()
# Create the Store_type column
measure_over_time_sales['Store_type'] = measure_over_time_sales['STORE_NBR'].
  ↪apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
  ↪else 'Other'))

measure_over_time_sales['Total_Sales'] = measure_over_time_sales['Total_Sales']

# Create the TransactionMonth column by converting 'YEARMONTH' to a datetime␣
  ↪format
measure_over_time_sales['TransactionMonth'] = pd.to_datetime(
    measure_over_time_sales['Month_ID'].astype(str) + '01', format='%Y%m%d')

# Filter the data to include only "Trial" and "Control" stores
pastSales = measure_over_time_sales[measure_over_time_sales['Store_type'].
  ↪isin(['Trial', 'Control'])]
```

[248]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
```

```
past_sales_controls_95 = pastSales[pastSales['Store_type'] == "Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_sales_controls_95['Total_Sales'] = past_sales_controls_95['Total_Sales'] *␣
 ↪(1 + stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
 ↪interval"
past_sales_controls_95['Store_type'] = "Control 95th % confidence interval"
```

[249]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
past_sales_controls_5 = pastSales[pastSales['Store_type'] == "Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_sales_controls_5['Total_Sales'] = past_sales_controls_5['Total_Sales'] *␣
 ↪(1 - stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
 ↪interval"
past_sales_controls_5['Store_type'] = "Control 5th % confidence interval"
```

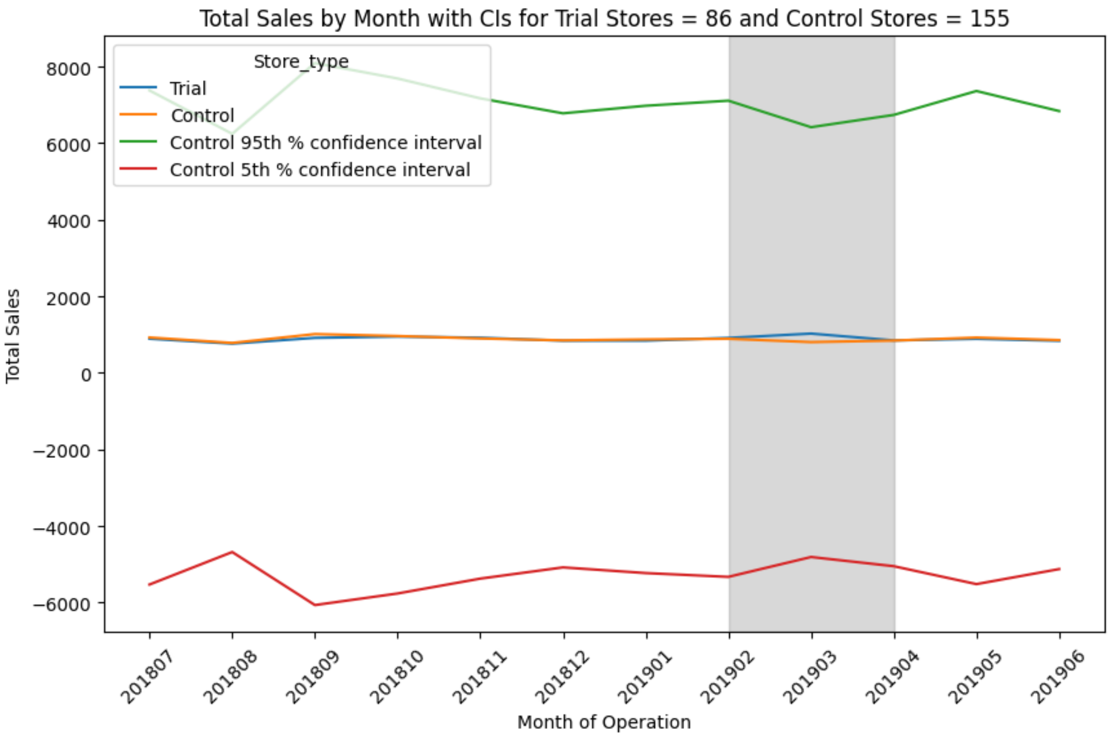
[250]:

```
# Concatenate the DataFrames
trial_assessment = pd.concat([pastSales, past_sales_controls_95,␣
 ↪past_sales_controls_5], ignore_index=True)
```

[252]:

```
# Convert 'TransactionMonth' to datetime if not already done
trial_assessment['TransactionMonth'] = pd.
 ↪to_datetime(trial_assessment['TransactionMonth'])

# Plotting
plt.figure(figsize=(10, 6))
sns.lineplot(
    data=trial_assessment,
    x='Month_ID',
    y='Total_Sales',
    hue='Store_type')

plt.axvspan(
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].min(),
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].max(),
    color='grey', alpha=0.3)

# Add labels and title
```

```
plt.xlabel('Month of Operation')
plt.ylabel('Total Sales')
plt.title('Total Sales by Month with CIs for Trial Stores = 86 and Control␣
 ↪Stores = 155')

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)
plt.show()
```



7 Number of Customers for Trial Store = 86 and Control Store = 155

[257]:

```
# Scaling Factor for Control Number for Customers

# Filter the pre-trial data for the trial store and control store
pre_trial_ncustomers_trial_store =␣
 ↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == trial_store) &
                                                      ␣
 ↪(pre_trial_measures['Month_ID'] < '201902')]['Number_of_Customers'].sum()

pre_trial_ncustomers_control_store =␣
 ↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == control_store) &
```

```
                                                                       ␣
 ↪(pre_trial_measures['Month_ID'] < '201902')]['Number_of_Customers'].sum()

# Calculate the scaling factor
scaling_factor_for_control_ncustomers = pre_trial_ncustomers_trial_store /␣
 ↪pre_trial_ncustomers_control_store
scaling_factor_for_control_ncustomers
```

[257]:
```
1.0
```

[258]:
```
measure_over_time_sales = metrics_df.copy()
# Apply the scaling factor only to the rows where 'STORE_NBR' equals␣
 ↪'control_store'
scaled_control_ncustomers =␣
 ↪measure_over_time_sales[measure_over_time_sales['STORE_NBR'] ==␣
 ↪control_store].copy()

scaled_control_ncustomers['controlNumCustomers'] =␣
 ↪scaled_control_ncustomers['Number_of_Customers'] *␣
 ↪scaling_factor_for_control_ncustomers
```

[259]:
```
# Percentage difference between the scaled control num customers and the trial␣
 ↪store's num customers during the trial period
trial_store_ncustomers = metrics_df[metrics_df['STORE_NBR'] == 86]
# Merge the trial num customers and control num customers based on 'YEARMONTH'
percentage_diff = pd.merge(trial_store_ncustomers[['Month_ID',␣
 ↪'Number_of_Customers']],
                           scaled_control_ncustomers[['Month_ID',␣
 ↪'controlNumCustomers']],
                           on='Month_ID')

# Calculate the percentage difference
percentage_diff['percentageDiff'] = 100 *␣
 ↪(percentage_diff['Number_of_Customers'] -␣
 ↪percentage_diff['controlNumCustomers']) /␣
 ↪percentage_diff['controlNumCustomers']
```

[260]:
```
# Filter the percentage difference for the pre-trial period (YEARMONTH < 201902)
pre_trial_percentage_diff = percentage_diff[percentage_diff['Month_ID'] <␣
 ↪'201902']

# Calculate the standard deviation of the percentage difference during the␣
 ↪pre-trial period
stdDev = np.std(pre_trial_percentage_diff['percentageDiff'])

# Display the result
```

```
print(f"Standard Deviation of Percentage Difference (Pre-Trial Period):␣
 ↪{stdDev}")
```

```
Standard Deviation of Percentage Difference (Pre-Trial Period):
1.7737954293019782
```

[261]:

```
# Calculate the t-values for the trial months
percentage_diff['tValue'] = percentage_diff['percentageDiff'] / stdDev

# Convert YEARMONTH to a datetime format to represent the TransactionMonth
percentage_diff['TransactionMonth'] = pd.
 ↪to_datetime(percentage_diff['Month_ID'].astype(str) + '01', format='%Y%m%d')

degrees_of_freedom = 7

# Find the 95th percentile of the t distribution
t_critical = stats.t.ppf(0.95, df=degrees_of_freedom)

# Display the calculated t-values and the critical t-value
print("T-Values during the Trial Period:")
print(percentage_diff[['TransactionMonth', 'tValue']])
print(f"Critical T-Value (95th Percentile, df={degrees_of_freedom}):␣
 ↪{t_critical}")
```

```
T-Values during the Trial Period:
   TransactionMonth     tValue
0        2018-07-01  -1.116362
1        2018-08-01   1.858559
2        2018-09-01   0.000000
3        2018-10-01   0.522003
4        2018-11-01  -0.558181
5        2018-12-01   0.581199
6        2019-01-01  -1.174506
7        2019-02-01   7.121215
8        2019-03-01  12.594702
9        2019-04-01   3.416745
10       2019-05-01  -1.063704
11       2019-06-01   1.780304
Critical T-Value (95th Percentile, df=7): 1.894578605061305
```

[263]:

```
# Create new dataframe for past customers
trial_store = 86
control_store = 155
measure_over_time_ncustomers = metrics_df.copy()
# Create the Store_type column
measure_over_time_ncustomers['Store_type'] =␣
 ↪measure_over_time_ncustomers['STORE_NBR'].apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
  ↪else 'Other'))

measure_over_time_ncustomers['Number_of_Customers'] =␣
  ↪measure_over_time_ncustomers['Number_of_Customers']

# Create the TransactionMonth column by converting 'YEARMONTH' to a datetime␣
  ↪format
measure_over_time_ncustomers['TransactionMonth'] = pd.to_datetime(
    measure_over_time_ncustomers['Month_ID'].astype(str) + '01',␣
  ↪format='%Y%m%d')

# Filter the data to include only "Trial" and "Control" stores
pastnCustomers =␣
  ↪measure_over_time_ncustomers[measure_over_time_ncustomers['Store_type'].
  ↪isin(['Trial', 'Control'])]
```

[264]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
past_ncustomers_controls_95 = pastnCustomers[pastnCustomers['Store_type'] ==␣
  ↪"Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_ncustomers_controls_95['Number_of_Customers'] =␣
  ↪past_ncustomers_controls_95['Number_of_Customers'] * (1 + stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
  ↪interval"
past_ncustomers_controls_95['Store_type'] = "Control 95th % confidence interval"
```

[265]:

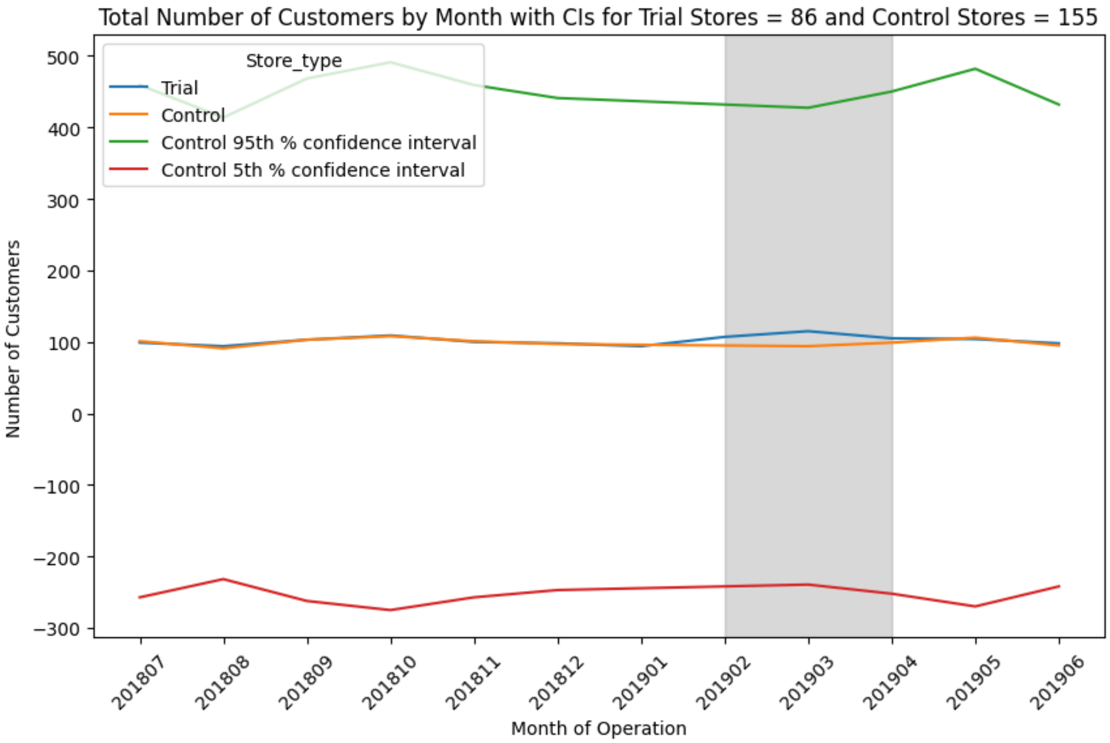
```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
past_ncustomers_controls_5 = pastnCustomers[pastnCustomers['Store_type'] ==␣
  ↪"Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_ncustomers_controls_5['Number_of_Customers'] =␣
  ↪past_ncustomers_controls_5['Number_of_Customers'] * (1 - stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
  ↪interval"
past_ncustomers_controls_5['Store_type'] = "Control 5th % confidence interval"
```

[266]:

```
# Concatenate the DataFrames
trial_assessment = pd.concat([pastnCustomers, past_ncustomers_controls_95,␣
 ↪past_ncustomers_controls_5], ignore_index=True)
```

[269]:

```
# Convert 'TransactionMonth' to datetime if not already done
trial_assessment['TransactionMonth'] = pd.
 ↪to_datetime(trial_assessment['TransactionMonth'])

# Plotting
plt.figure(figsize=(10, 6))
sns.lineplot(
    data=trial_assessment,
    x='Month_ID',
    y='Number_of_Customers',
    hue='Store_type')

plt.axvspan(
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].min(),
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].max(),
    color='grey', alpha=0.3)

# Add labels and title
plt.xlabel('Month of Operation')
plt.ylabel('Number of Customers')
plt.title('Total Number of Customers by Month with CIs for Trial Stores = 86␣
 ↪and Control Stores = 155')
plt.xticks(rotation=45)
plt.show()
```



8 Trial Store 88

[275]:

```
# Calculate the correlation for the trial store 88 for its total sales and␣
  ↪number of customers
input_table = pre_trial_measures
metric_col_sales = 'Total_Sales'
metric_col_ncustomers = 'Number_of_Customers'
store_comparison = 88
corr_sales = calculate_correlation(input_table, metric_col_sales,␣
  ↪store_comparison)
corr_ncustomer = calculate_correlation(input_table, metric_col_ncustomers,␣
  ↪store_comparison)

# Calculate the magnitude distance for the trial store 88 for its total sales␣
  ↪and number of customers
magdist_sales = calculate_magnitude_distance(input_table, metric_col_sales,␣
  ↪store_comparison)
magdist_ncustomer = calculate_magnitude_distance(input_table,␣
  ↪metric_col_ncustomers, store_comparison)
```

```
C:\Users\Alden\AppData\Local\Temp\ipykernel_21008\2769831981.py:21:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
```

```
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  calc_corr_table = pd.concat([calc_corr_table, calculated_measure],
ignore_index=True)
C:\Users\Alden\AppData\Local\Temp\ipykernel_21008\2769831981.py:21:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  calc_corr_table = pd.concat([calc_corr_table, calculated_measure],
ignore_index=True)
C:\Users\Alden\AppData\Local\Temp\ipykernel_21008\985414664.py:23:
FutureWarning: The behavior of DataFrame concatenation with empty or all-NA
entries is deprecated. In a future version, this will no longer exclude empty or
all-NA columns when determining the result dtypes. To retain the old behavior,
exclude the relevant entries before the concat operation.
  calc_dist_table = pd.concat([calc_dist_table, calculated_measure],
ignore_index=True)
```

[276]:

```
# Calculates the score for control store sales
score_n_sales = pd.merge(corr_sales, magdist_sales, on=['Store1', 'Store2'])
score_n_customers = pd.merge(corr_ncustomer, magdist_ncustomer, on=['Store1',␣
 ↪'Store2'])
score_n_sales, score_n_customers

# Merge sales scores and customer scores into a single table
score_control = pd.merge(score_n_sales, score_n_customers, on=['Store1',␣
 ↪'Store2'])
score_control

# Calculate the final control score as a simple average of sales and customer␣
 ↪scores
score_control['finalControlScore'] = (0.5 * ( 0.5 *␣
 ↪score_control['corr_measure_x'] + 0.5 * score_control['mag_measure_x']) +
                                      0.5 * ( 0.5 *␣
 ↪score_control['corr_measure_y'] + 0.5 * score_control['mag_measure_y']))
```

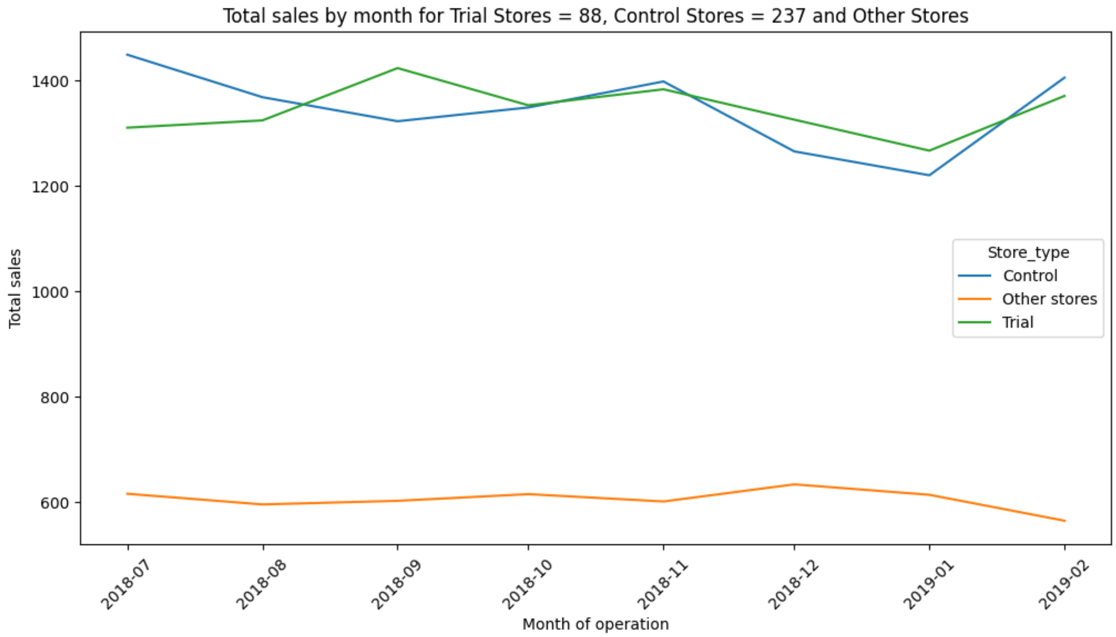
[277]:

```
# Retrieves the control score number
max_row = score_control.loc[score_control['finalControlScore'].idxmax()]
control_store = 237
```

[278]:

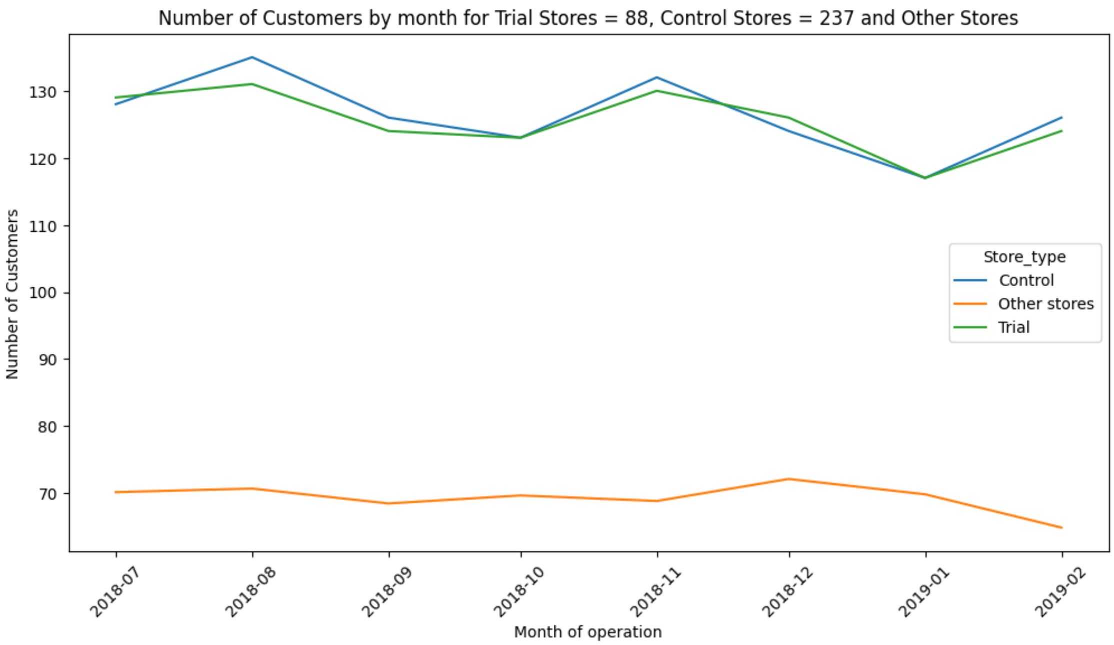
```
# Plot the Total sales by Month
trial_store = 88
control_store = 237

#  Create a 'Store_type' column to classify stores
```

```
measure_over_time_sales = metrics_df.copy()
measure_over_time_sales['Store_type'] = measure_over_time_sales['STORE_NBR'].
  ↪apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
  ↪else 'Other stores'))

# Calculate the mean of 'totSales' by 'YEARMONTH' and 'Store_type'
past_sales = measure_over_time_sales.groupby(['Month_ID', 'Store_type'],␣
  ↪as_index=False).agg({'Total_Sales': 'mean'})

# Create a 'TransactionMonth' column with the format YYYY-MM-DD
past_sales['TransactionMonth'] = pd.to_datetime(past_sales['Month_ID'].
  ↪astype(str) + '01', format='%Y%m%d')

# Filter data to only include months before March 2019
past_sales = past_sales[past_sales['Month_ID'] < '201903']

# Plotting the sales trends
plt.figure(figsize=(12, 6))
sns.lineplot(data=past_sales, x='TransactionMonth', y='Total_Sales',␣
  ↪hue='Store_type')
plt.title('Total sales by month for Trial Stores = 88, Control Stores = 237 and␣
  ↪Other Stores')
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.xticks(rotation=45)
plt.show()
```



[283]:

```
# Plot number of customer per month
trial_store = 88
control_store = 237

# Create a 'Store_type' column to classify stores
measure_over_time_custs = metrics_df.copy()
measure_over_time_custs['Store_type'] = measure_over_time_custs['STORE_NBR'].
 ↪apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
 ↪else 'Other stores'))

# Calculate the mean of 'totSales' by 'YEARMONTH' and 'Store_type'
past_custs = measure_over_time_custs.groupby(['Month_ID', 'Store_type'],␣
 ↪as_index=False).agg({'Number_of_Customers': 'mean'})

# Create a 'TransactionMonth' column with the format YYYY-MM-DD
past_custs['TransactionMonth'] = pd.to_datetime(past_sales['Month_ID'].
 ↪astype(str) + '01', format='%Y%m%d')

# Filter data to only include months before March 2019
past_custs = past_custs[past_custs['Month_ID'] < '201903']

# Plotting the sales trends
plt.figure(figsize=(12, 6))
sns.lineplot(data=past_custs, x='TransactionMonth', y='Number_of_Customers',␣
 ↪hue='Store_type')
plt.title('Number of Customers by month for Trial Stores = 88, Control Stores =␣
 ↪237 and Other Stores')
plt.xlabel('Month of operation')
plt.ylabel('Number of Customers')
plt.xticks(rotation=45)
plt.show()
```



9 Total Sales for Trial Store = 88 and Control Store = 237

[292]:

```
# Filter the pre-trial data for the trial store and control store
pre_trial_sales_trial_store =␣
 ↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == trial_store) &
                                                     ␣
 ↪(pre_trial_measures['Month_ID'] < '201902')]['Total_Sales'].sum()

pre_trial_sales_control_store =␣
 ↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == control_store) &
                                                       ␣
 ↪(pre_trial_measures['Month_ID'] < '201902')]['Total_Sales'].sum()

# Calculate the scaling factor
scaling_factor_for_control_sales = pre_trial_sales_trial_store /␣
 ↪pre_trial_sales_control_store
scaling_factor_for_control_sales
```

[292]:

```
1.001558330664959
```

[293]:

```
measure_over_time_sales = metrics_df.copy()
# Apply the scaling factor only to the rows where 'STORE_NBR' equals␣
 ↪'control_store'
```

```
scaled_control_sales =␣
 ↪measure_over_time_sales[measure_over_time_sales['STORE_NBR'] ==␣
 ↪control_store].copy()

# Create the 'controlSales' column by multiplying 'totSales' with the scaling␣
 ↪factor
scaled_control_sales['controlSales'] = scaled_control_sales['Total_Sales'] *␣
 ↪scaling_factor_for_control_sales
```

[294]:

```
# Percentage difference between the scaled control sales and the trial store's␣
 ↪sales during the trial period
trial_store_sales = metrics_df[metrics_df['STORE_NBR'] == 88]
# Merge the trial sales and control sales based on 'YEARMONTH'
percentage_diff = pd.merge(trial_store_sales[['Month_ID', 'Total_Sales']],
                           scaled_control_sales[['Month_ID', 'controlSales']],
                           on='Month_ID')

# Calculate the percentage difference
percentage_diff['percentageDiff'] = 100 * (percentage_diff['Total_Sales'] -␣
 ↪percentage_diff['controlSales']) / percentage_diff['controlSales']
```

[295]:

```
# Filter the percentage difference for the pre-trial period (YEARMONTH < 201902)
pre_trial_percentage_diff = percentage_diff[percentage_diff['Month_ID'] <␣
 ↪'201902']

# Calculate the standard deviation of the percentage difference during the␣
 ↪pre-trial period
stdDev = np.std(pre_trial_percentage_diff['percentageDiff'])

# Display the result
print(f"Standard Deviation of Percentage Difference (Pre-Trial Period):␣
 ↪{stdDev}")
```

```
Standard Deviation of Percentage Difference (Pre-Trial Period):
5.300288085872677
```

[296]:

```
# Calculate the t-values for the trial months
percentage_diff['tValue'] = percentage_diff['percentageDiff'] / stdDev

# Convert YEARMONTH to a datetime format to represent the TransactionMonth
percentage_diff['TransactionMonth'] = pd.
 ↪to_datetime(percentage_diff['Month_ID'].astype(str) + '01', format='%Y%m%d')

degrees_of_freedom = 7

# Find the 95th percentile of the t distribution
t_critical = stats.t.ppf(0.95, df=degrees_of_freedom)
```

```
# Display the calculated t-values and the critical t-value
print("T-Values during the Trial Period:")
print(percentage_diff[['TransactionMonth', 'tValue']])
print(f"Critical T-Value (95th Percentile, df={degrees_of_freedom}):␣
↪{t_critical}")
```

```
T-Values during the Trial Period:
   TransactionMonth    tValue
0        2018-07-01 -1.829352
1        2018-08-01 -0.635330
2        2018-09-01  1.406755
3        2018-10-01  0.027927
4        2018-11-01 -0.228837
5        2018-12-01  0.867103
6        2019-01-01  0.691899
7        2019-02-01 -0.493321
8        2019-03-01  4.164735
9        2019-04-01  3.642449
10       2019-05-01  1.681935
11       2019-06-01  3.252845
Critical T-Value (95th Percentile, df=7): 1.894578605061305
```

[297]:

```
# Create new dataframe for past sales
trial_store = 88
control_store = 237
measure_over_time_sales = metrics_df.copy()
# Create the Store_type column
measure_over_time_sales['Store_type'] = measure_over_time_sales['STORE_NBR'].
↪apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
↪else 'Other'))

measure_over_time_sales['Total_Sales'] = measure_over_time_sales['Total_Sales']

# Create the TransactionMonth column by converting 'YEARMONTH' to a datetime␣
↪format
measure_over_time_sales['TransactionMonth'] = pd.to_datetime(
    measure_over_time_sales['Month_ID'].astype(str) + '01', format='%Y%m%d')

# Filter the data to include only "Trial" and "Control" stores
pastSales = measure_over_time_sales[measure_over_time_sales['Store_type'].
↪isin(['Trial', 'Control'])]
```

[298]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
```

```
past_sales_controls_95 = pastSales[pastSales['Store_type'] == "Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_sales_controls_95['Total_Sales'] = past_sales_controls_95['Total_Sales'] *␣
 ↪(1 + stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
 ↪interval"
past_sales_controls_95['Store_type'] = "Control 95th % confidence interval"
```

[299]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
past_sales_controls_5 = pastSales[pastSales['Store_type'] == "Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_sales_controls_5['Total_Sales'] = past_sales_controls_5['Total_Sales'] *␣
 ↪(1 - stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
 ↪interval"
past_sales_controls_5['Store_type'] = "Control 5th % confidence interval"
```

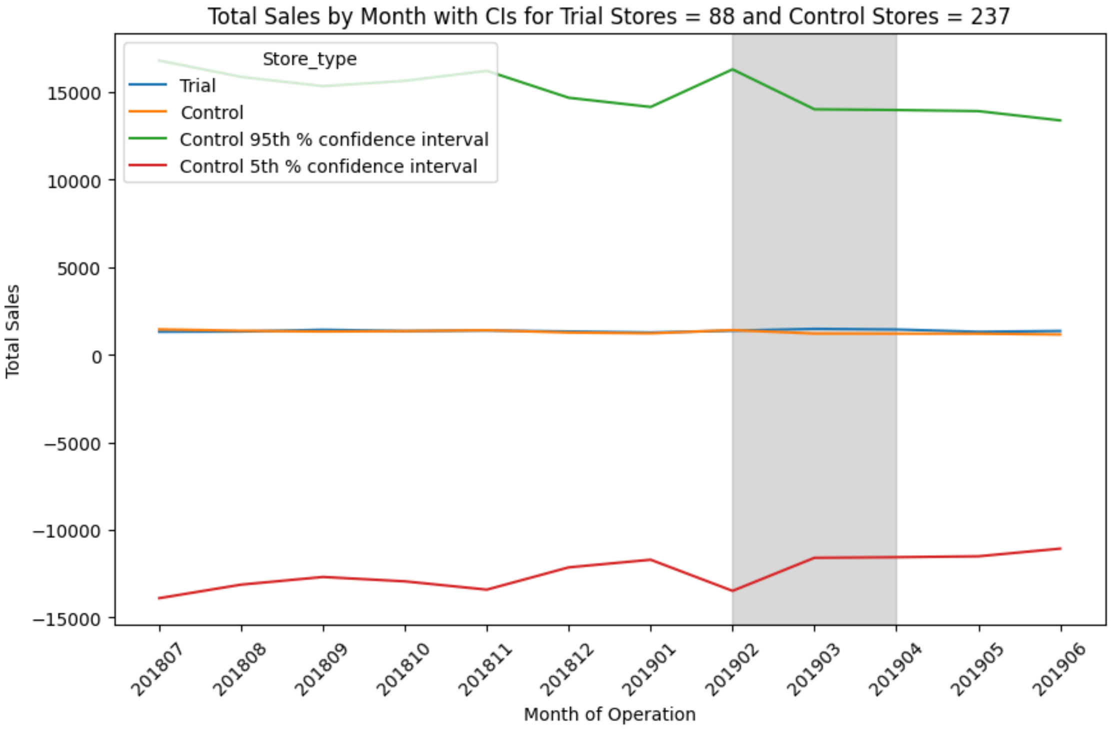
[300]:

```
# Concatenate the DataFrames
trial_assessment = pd.concat([pastSales, past_sales_controls_95,␣
 ↪past_sales_controls_5], ignore_index=True)
```

[301]:

```
# Convert 'TransactionMonth' to datetime if not already done
trial_assessment['TransactionMonth'] = pd.
 ↪to_datetime(trial_assessment['TransactionMonth'])

# Plotting
plt.figure(figsize=(10, 6))
sns.lineplot(
    data=trial_assessment,
    x='Month_ID',
    y='Total_Sales',
    hue='Store_type')

plt.axvspan(
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].min(),
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].max(),
    color='grey', alpha=0.3)

# Add labels and title
```

```
plt.xlabel('Month of Operation')
plt.ylabel('Total Sales')
plt.title('Total Sales by Month with CIs for Trial Stores = 88 and Control␣
↪Stores = 237')
plt.xticks(rotation=45)
plt.show()
```



10 Number of Customers for Trial Store = 88 and Control Store = 237

[302]:
```
# Scaling Factor for Control Number for Customers

# Filter the pre-trial data for the trial store and control store
pre_trial_ncustomers_trial_store =␣
↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == trial_store) &
                                                                           ␣
↪(pre_trial_measures['Month_ID'] < '201902')]['Number_of_Customers'].sum()

pre_trial_ncustomers_control_store =␣
↪pre_trial_measures[(pre_trial_measures['STORE_NBR'] == control_store) &
                                                                             ␣
↪(pre_trial_measures['Month_ID'] < '201902')]['Number_of_Customers'].sum()
```

```
# Calculate the scaling factor
scaling_factor_for_control_ncustomers = pre_trial_ncustomers_trial_store /␣
 ↪pre_trial_ncustomers_control_store
scaling_factor_for_control_ncustomers
```

[302]:
```
0.9943502824858758
```

[304]:
```
measure_over_time_sales = metrics_df.copy()
# Apply the scaling factor only to the rows where 'STORE_NBR' equals␣
 ↪'control_store'
scaled_control_ncustomers =␣
 ↪measure_over_time_sales[measure_over_time_sales['STORE_NBR'] ==␣
 ↪control_store].copy()

scaled_control_ncustomers['controlNumCustomers'] =␣
 ↪scaled_control_ncustomers['Number_of_Customers'] *␣
 ↪scaling_factor_for_control_ncustomers
```

[306]:
```
# Percentage difference between the scaled control num customers and the trial␣
 ↪store's num customers during the trial period
trial_store_ncustomers = metrics_df[metrics_df['STORE_NBR'] == 88]
# Merge the trial num customers and control num customers based on 'YEARMONTH'
percentage_diff = pd.merge(trial_store_ncustomers[['Month_ID',␣
 ↪'Number_of_Customers']],
                           scaled_control_ncustomers[['Month_ID',␣
 ↪'controlNumCustomers']],
                           on='Month_ID')

# Calculate the percentage difference
percentage_diff['percentageDiff'] = 100 *␣
 ↪(percentage_diff['Number_of_Customers'] -␣
 ↪percentage_diff['controlNumCustomers']) /␣
 ↪percentage_diff['controlNumCustomers']
```

[307]:
```
# Filter the percentage difference for the pre-trial period (YEARMONTH < 201902)
pre_trial_percentage_diff = percentage_diff[percentage_diff['Month_ID'] <␣
 ↪'201902']

# Calculate the standard deviation of the percentage difference during the␣
 ↪pre-trial period
stdDev = np.std(pre_trial_percentage_diff['percentageDiff'])

# Display the result
print(f"Standard Deviation of Percentage Difference (Pre-Trial Period):␣
 ↪{stdDev}")
```

```
Standard Deviation of Percentage Difference (Pre-Trial Period):
1.4663021905247504
```

[308]:

```
# Calculate the t-values for the trial months
percentage_diff['tValue'] = percentage_diff['percentageDiff'] / stdDev

# Convert YEARMONTH to a datetime format to represent the TransactionMonth
percentage_diff['TransactionMonth'] = pd.
 ↪to_datetime(percentage_diff['Month_ID'].astype(str) + '01', format='%Y%m%d')

degrees_of_freedom = 7

# Find the 95th percentile of the t distribution
t_critical = stats.t.ppf(0.95, df=degrees_of_freedom)

# Display the calculated t-values and the critical t-value
print("T-Values during the Trial Period:")
print(percentage_diff[['TransactionMonth', 'tValue']])
print(f"Critical T-Value (95th Percentile, df={degrees_of_freedom}):␣
 ↪{t_critical}")
```

```
T-Values during the Trial Period:
   TransactionMonth    tValue
0        2018-07-01  0.923323
1        2018-08-01 -1.644692
2        2018-09-01 -0.701178
3        2018-10-01  0.387493
4        2018-11-01 -0.651693
5        2018-12-01  1.493723
6        2019-01-01  0.387493
7        2019-02-01 -0.701178
8        2019-03-01  9.032820
9        2019-04-01  4.959910
10       2019-05-01 -0.144183
11       2019-06-01  1.540203
Critical T-Value (95th Percentile, df=7): 1.894578605061305
```

[310]:

```
# Create new dataframe for past customers
trial_store = 88
control_store = 237
measure_over_time_ncustomers = metrics_df.copy()
# Create the Store_type column
measure_over_time_ncustomers['Store_type'] =␣
 ↪measure_over_time_ncustomers['STORE_NBR'].apply(
    lambda x: 'Trial' if x == trial_store else ('Control' if x == control_store␣
 ↪else 'Other'))
```

```
measure_over_time_ncustomers['Number_of_Customers'] =␣
 ↪measure_over_time_ncustomers['Number_of_Customers']

# Create the TransactionMonth column by converting 'YEARMONTH' to a datetime␣
 ↪format
measure_over_time_ncustomers['TransactionMonth'] = pd.to_datetime(
    measure_over_time_ncustomers['Month_ID'].astype(str) + '01',␣
 ↪format='%Y%m%d')

# Filter the data to include only "Trial" and "Control" stores
pastnCustomers =␣
 ↪measure_over_time_ncustomers[measure_over_time_ncustomers['Store_type'].
 ↪isin(['Trial', 'Control'])]
```

[311]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
past_ncustomers_controls_95 = pastnCustomers[pastnCustomers['Store_type'] ==␣
 ↪"Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_ncustomers_controls_95['Number_of_Customers'] =␣
 ↪past_ncustomers_controls_95['Number_of_Customers'] * (1 + stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
 ↪interval"
past_ncustomers_controls_95['Store_type'] = "Control 95th % confidence interval"
```

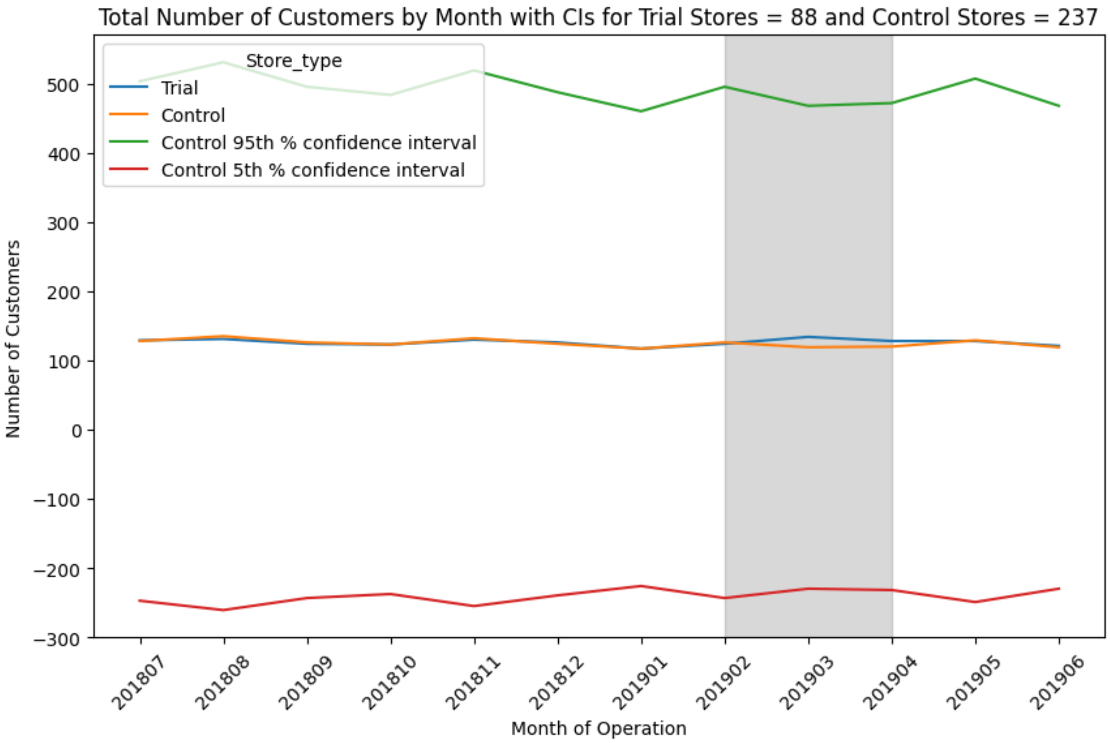
[312]:

```
# Control store percentiles

# Filter the DataFrame for rows where Store_type is "Control"
past_ncustomers_controls_5 = pastnCustomers[pastnCustomers['Store_type'] ==␣
 ↪"Control"].copy()

# Adjust the totSales column by applying the standard deviation factor
past_ncustomers_controls_5['Number_of_Customers'] =␣
 ↪past_ncustomers_controls_5['Number_of_Customers'] * (1 - stdDev * 2)

# Update the Store_type column to indicate the "Control 95th % confidence␣
 ↪interval"
past_ncustomers_controls_5['Store_type'] = "Control 5th % confidence interval"
```

[313]:

```
# Concatenate the DataFrames
trial_assessment = pd.concat([pastnCustomers, past_ncustomers_controls_95,␣
 ↪past_ncustomers_controls_5], ignore_index=True)
```

```
[314]: # Convert 'TransactionMonth' to datetime if not already done
trial_assessment['TransactionMonth'] = pd.
 ↪to_datetime(trial_assessment['TransactionMonth'])

# Plotting
plt.figure(figsize=(10, 6))
sns.lineplot(
    data=trial_assessment,
    x='Month_ID',
    y='Number_of_Customers',
    hue='Store_type')

plt.axvspan(
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].min(),
    trial_assessment[(trial_assessment['Month_ID'] < '201905') &␣
 ↪(trial_assessment['Month_ID'] > '201901')]['Month_ID'].max(),
    color='grey', alpha=0.3)

# Add labels and title
plt.xlabel('Month of Operation')
plt.ylabel('Number of Customers')
plt.title('Total Number of Customers by Month with CIs for Trial Stores = 88␣
 ↪and Control Stores = 237')
plt.xticks(rotation=45)
plt.show()
```


Total Number of Customers by Month with CIs for Trial Stores = 88 and Control Stores = 237
Store_type
Trial
Control
Control 95th % confidence interval
Control 5th % confidence interval
Number of Customers
500
400
300
200
100
0
−100
−200
−300
201807
201808
201809
201810
201811
201812
201901
201902
201903
201904
201905
201906
Month of Operation

[ ]:

[ ]:

[ ]:

[ ]: