In [ ]:

```
"""
The Task:
Define metrics to select control stores.
Analyze trial stores against controls.
Use R/Python for data analysis and visualization and summarise findings and provide re

Details:

You are part of Quantium’s retail analytics team and have been approached by your clie

You have received the following email from Zilinka.

Hi,

Thanks for your feedback earlier, I’m glad you find my follow up emails helpful in ens

For this part of the project we will be examining the performance in trial vs control

Select control stores - explore the data and define metrics for your control store sel

Assessment of the trial - this one should give you some interesting insights into each

Collate findings - summarise your findings for each store and provide an recommendatio

Remember when working with a client visualisations are key to helping them understand

Keep up the good work!

Thanks,

Zilinka'

Here is your task
Julia has asked us to evaluate the performance of a store trial which was performed in

We have chosen to complete this task in R, however, you will also find Python to be a

To get started, use the QVI_data dataset below or your output from task 1 and consider

This can be broken down by:

total sales revenue
total number of customers
average number of transactions per customer

Create a measure to compare different control stores to each of the trial stores to do

Once you have selected your control stores, compare each trial and control pair during

As we are in the early stages of this analysis Zilinka has asked us to submit our init

"""
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import pearsonr

# Load the dataset
file_path = "QVI_data.csv"
data = pd.read_csv(file_path)
```

In [2]:
```
data.head()
```

Out[2]:

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

In [3]:
```
data.shape
```

Out[3]:
```
(264834, 12)
```

In [4]:
```
data['STORE_NBR'].unique()
```

Out[4]:
```
array([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
        14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,
        27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,
        40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,
        53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,
        66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,
        79,  80,  81,  82,  83,  84,  85,  86, 155,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
       117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
       130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
       143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 156,
       157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
       170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182,
       183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
       196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208,
       209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221,
       222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234,
       235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247,
       248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260,
       261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272],
      dtype=int64)
```

In [3]:
```
#Process the date column to yyyymm format
data['YEARMONTH'] = data['DATE'].apply(lambda x: x[:7].replace("-", ""))
```

In [6]:
```
data.head()
```

Out[6]:

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

In [4]:
```
# Calculate monthly metrics of  total sales, number of customers, transactions per cust
#chips per transaction, and average price per unit for each store and month
monthly_metrics = data.groupby(['STORE_NBR', 'YEARMONTH']).agg(
    totSales=('TOT_SALES', 'sum'),
    nCustomers=('LYLTY_CARD_NBR', 'nunique'),
    nTxnPerCust=('TXN_ID', 'count'),
    nChipsPerTxn=('PROD_QTY', 'mean'),
    avgPricePerUnit=('TOT_SALES', 'mean')
).reset_index()
```

In [5]:
```
#Select stores that have data for the full pre-trial period
stores_with_full_obs = monthly_metrics.groupby('STORE_NBR').filter(lambda x: len(x) ==
pre_trial_measures = stores_with_full_obs[stores_with_full_obs['YEARMONTH'] < '201902'
```

In [6]:

```
#Functions to Calculate Correlation and Magnitude of similarity of each control store t


def calculate_correlation(input_table, metric_col, store_comparison):
    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:
            corr, _ = pearsonr(
                input_table[input_table['STORE_NBR'] == store_comparison][metric_col],
                input_table[input_table['STORE_NBR'] == store][metric_col]
            )
            new_row = pd.DataFrame({'Store1': [store_comparison], 'Store2': [store], '
            corr_table = pd.concat([corr_table, new_row], ignore_index=True)

    return corr_table

def calculate_magnitude_distance(input_table, metric_col, store_comparison):
    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:
            measure_diff = abs(
                input_table[input_table['STORE_NBR'] == store_comparison][metric_col].
                input_table[input_table['STORE_NBR'] == store][metric_col].values
            )
            new_rows = pd.DataFrame({
                'Store1': store_comparison,
                'Store2': store,
                'YEARMONTH': input_table[input_table['STORE_NBR'] == store_comparison]
                'measure': measure_diff
            })
            dist_table = pd.concat([dist_table, new_rows], ignore_index=True)

    min_max_dist = dist_table.groupby(['Store1', 'YEARMONTH']).agg(
        minDist=('measure', 'min'), maxDist=('measure', 'max')
    ).reset_index()
    dist_table = dist_table.merge(min_max_dist, on=['Store1', 'YEARMONTH'])
    dist_table['magnitudeMeasure'] = 1 - (dist_table['measure'] - dist_table['minDist'

    final_dist_table = dist_table.groupby(['Store1', 'Store2']).agg(
        mag_measure=('magnitudeMeasure', 'mean')
    ).reset_index()

    return final_dist_table
```

In [16]:
```
#Calculate Correlation and Magnitude scores for both sales and customes
trial_store = 77
corr_n_sales = calculate_correlation(pre_trial_measures, 'totSales', trial_store)
corr_n_customers = calculate_correlation(pre_trial_measures, 'nCustomers', trial_store
magnitude_n_sales = calculate_magnitude_distance(pre_trial_measures, 'totSales', trial
magnitude_n_customers = calculate_magnitude_distance(pre_trial_measures, 'nCustomers',
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_10192\4097937659.py:17: FutureWarning: Th
e 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 determinin
g the result dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  corr_table = pd.concat([corr_table, new_row], ignore_index=True)
C:\Users\Tonye\AppData\Local\Temp\ipykernel_10192\4097937659.py:17: FutureWarning: Th
e 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 determinin
g the result dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  corr_table = pd.concat([corr_table, new_row], ignore_index=True)
C:\Users\Tonye\AppData\Local\Temp\ipykernel_10192\4097937659.py:37: FutureWarning: Th
e 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 determinin
g the result dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  dist_table = pd.concat([dist_table, new_rows], ignore_index=True)
```

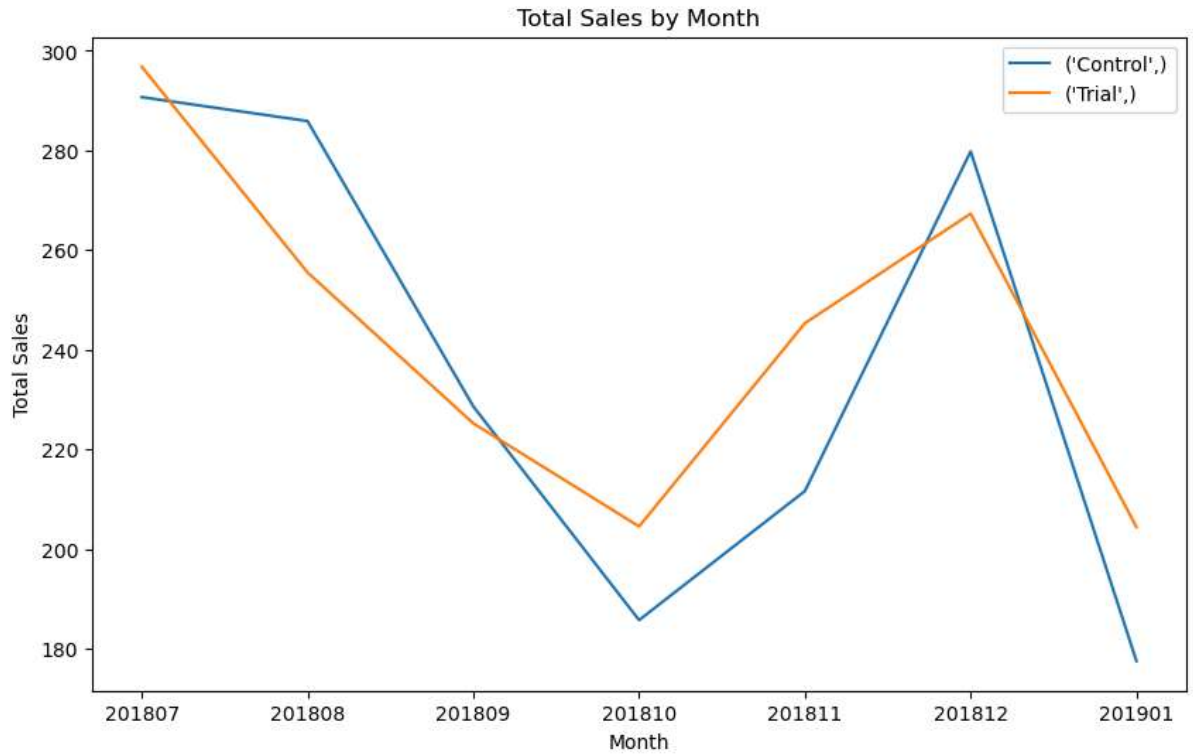
In [17]:
```
# Create a combined score composed of correlation and magnitude, by first merging the
corr_weight = 0.5
score_n_sales = corr_n_sales.merge(magnitude_n_sales, on=['Store1', 'Store2'])
score_n_sales['scoreNSales'] = corr_weight * score_n_sales['corr_measure'] + (1 - corr

score_n_customers = corr_n_customers.merge(magnitude_n_customers, on=['Store1', 'Store
score_n_customers['scoreNCust'] = corr_weight * score_n_customers['corr_measure'] + (1

score_control = score_n_sales.merge(score_n_customers, on=['Store1', 'Store2'])
score_control['finalControlScore'] = score_control['scoreNSales'] * 0.5 + score_contro
```

In [18]:
```
#Select Control Stores based on the control score combining correlation and magnitude
control_store = score_control.loc[score_control['finalControlScore'].idxmax(), 'Store2
print("Control Store:", control_store)
```

```
Control Store: 233
```

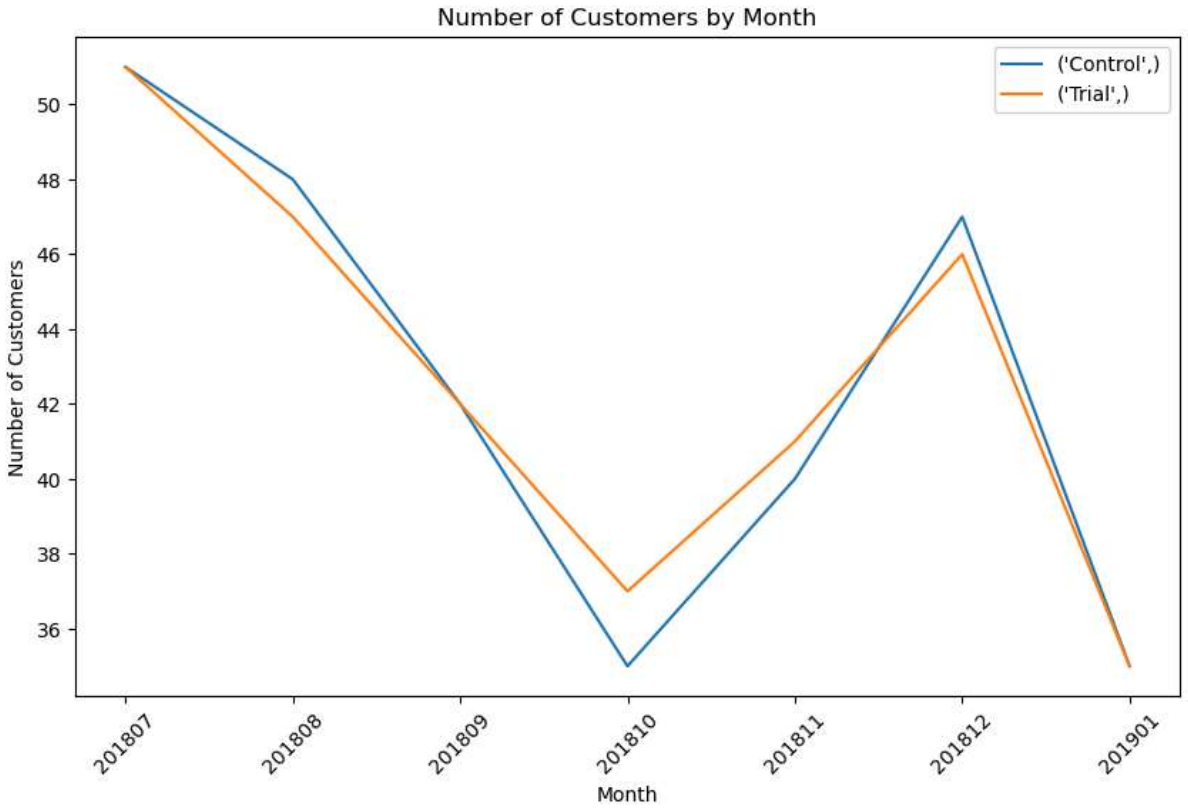
In [19]:
```
#Visual check to see if the control store is indeed similar to the trial store before t
#Visual Check on Sales

# Plot total sales
pre_trial_sales = monthly_metrics[
    (monthly_metrics['STORE_NBR'].isin([trial_store, control_store])) &
    (monthly_metrics['YEARMONTH'] < '201902')
]
pre_trial_sales['Store_type'] = pre_trial_sales['STORE_NBR'].apply(lambda x: 'Trial' i

plt.figure(figsize=(10, 6))
for key, grp in pre_trial_sales.groupby(['Store_type']):
    plt.plot(grp['YEARMONTH'], grp['totSales'], label=key)
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.title('Total Sales by Month')
plt.legend(loc='best')
plt.show()
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_10192\4020865328.py:9: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  pre_trial_sales['Store_type'] = pre_trial_sales['STORE_NBR'].apply(lambda x: 'Tria
l' if x == trial_store else 'Control')
```



In [20]:
```
#Visualise number of customers

# Define the trial and control store numbers
#trial_store = 77
#control_store = <control_store_number>  # Replace with the control store number you've

# Plot number of customers
pre_trial_customers = monthly_metrics[
    (monthly_metrics['STORE_NBR'].isin([trial_store, control_store])) &
    (monthly_metrics['YEARMONTH'] < '201902')
]
pre_trial_customers['Store_type'] = pre_trial_customers['STORE_NBR'].apply(lambda x: '

plt.figure(figsize=(10, 6))
for key, grp in pre_trial_customers.groupby(['Store_type']):
    plt.plot(grp['YEARMONTH'], grp['nCustomers'], label=key)
plt.xlabel('Month')
plt.ylabel('Number of Customers')
plt.title('Number of Customers by Month')
plt.legend(loc='best')
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_10192\1492363467.py:12: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  pre_trial_customers['Store_type'] = pre_trial_customers['STORE_NBR'].apply(lambda
x: 'Trial' if x == trial_store else 'Control')
```



In [21]:

```
#Assess trail store performance
#with scaling the control store's sales to a level similar to control for any differen
#outside of the trial period.

#Scale store sales
scaling_factor = pre_trial_measures[pre_trial_measures['STORE_NBR'] == trial_store]['t
                 pre_trial_measures[pre_trial_measures['STORE_NBR'] == control_store][
scaled_control_sales = monthly_metrics[monthly_metrics['STORE_NBR'] == control_store]
scaled_control_sales['scaledSales'] = scaled_control_sales['totSales'] * scaling_facto
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_10192\3882563808.py:9: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  scaled_control_sales['scaledSales'] = scaled_control_sales['totSales'] * scaling_fa
ctor
```

In [22]:

```
#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
trial_period = monthly_metrics[(monthly_metrics['YEARMONTH'] >= '201902') & (monthly_m
percentage_diff = trial_period.merge(scaled_control_sales, on='YEARMONTH')
percentage_diff['percentageDiff'] = (percentage_diff['totSales_x'] - percentage_diff['
```

In [33]:

```
# Define trial and control stores
trial_store = 77
control_store = 233

# Define the pre-trial and trial periods
pre_trial_period_end = '201901'
trial_period_start = '201902'
trial_period_end = '201904'

# Calculate the total sales in the pre-trial period for both trial and control stores
pre_trial_sales_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

pre_trial_sales_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

# Calculate the scaling factor
scaling_factor = pre_trial_sales_trial / pre_trial_sales_control

# Copy the original dataframe to measure_over_time_sales
measure_over_time_sales = monthly_metrics.copy()

# Apply the scaling factor to the control store sales during the trial period
measure_over_time_sales.loc[
    (measure_over_time_sales['STORE_NBR'] == control_store) &
    (measure_over_time_sales['YEARMONTH'] >= trial_period_start) &
    (measure_over_time_sales['YEARMONTH'] <= trial_period_end),
    'scaledControlSales'
] = measure_over_time_sales['totSales'] * scaling_factor

# Merge the trial and scaled control sales data
percentage_diff = measure_over_time_sales[
    (measure_over_time_sales['STORE_NBR'].isin([trial_store, control_store])) &
    (measure_over_time_sales['YEARMONTH'] >= trial_period_start) &
    (measure_over_time_sales['YEARMONTH'] <= trial_period_end)
].pivot_table(
    index='YEARMONTH',
    columns='STORE_NBR',
    values=['totSales', 'scaledControlSales']
).reset_index()

# Print the columns to understand the structure
print(percentage_diff.columns)

# Adjust column renaming and calculation accordingly
# The column names in the DataFrame will be multi-indexed, so we need to handle that
percentage_diff.columns = ['YEARMONTH', 'controlSales', 'trialSales', 'scaledControlSa

# Calculate the percentage difference
percentage_diff['percentageDiff'] = (
    (percentage_diff['trialSales'] - percentage_diff['scaledControlSales']) /
    percentage_diff['scaledControlSales']
) * 100

# Print the result
print(percentage_diff)
```

```
MultiIndex([(         'YEARMONTH',  ''),
            ('scaledControlSales', 233),
            (          'totSales',  77),
            (          'totSales', 233)],
           names=[None, 'STORE_NBR'])
```

	YEARMONTH	controlSales	trialSales	scaledControlSales	percentageDiff
0	201902	249.762622	235.0	244.0	-3.688525
1	201903	203.802205	278.5	199.1	39.879458
2	201904	162.345704	263.5	158.6	66.141236

In [39]:

```
#Anayse the significance of the percentage difference through the t-value and standard


from scipy.stats import t

# Define trial and control stores
trial_store = 77
control_store = 233  # Assuming 233 is the control store for this example

# Define the pre-trial and trial periods
pre_trial_period_end = '201901'
trial_period_start = '201902'
trial_period_end = '201904'

# Calculate the total sales in the pre-trial period for both trial and control stores
pre_trial_sales_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

pre_trial_sales_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

# Calculate the scaling factor
scaling_factor = pre_trial_sales_trial / pre_trial_sales_control

# Copy the original dataframe to measure_over_time_sales
measure_over_time_sales = monthly_metrics.copy()

# Apply the scaling factor to the control store sales during the pre-trial and trial pe
measure_over_time_sales.loc[
    (measure_over_time_sales['STORE_NBR'] == control_store),
    'scaledControlSales'
] = measure_over_time_sales['totSales'] * scaling_factor

# Filter for the trial and control stores only
filtered_sales = measure_over_time_sales[
    measure_over_time_sales['STORE_NBR'].isin([trial_store, control_store])
]

# Check the structure of the filtered_sales DataFrame
print(filtered_sales.head())

# Create a pivot table
pre_trial_percentage_diff = filtered_sales[
    (filtered_sales['YEARMONTH'] <= pre_trial_period_end)
].pivot_table(
    index='YEARMONTH',
    columns='STORE_NBR',
    values=['totSales', 'scaledControlSales'],
    aggfunc='sum'
).reset_index()

# Flatten the multi-level columns
pre_trial_percentage_diff.columns = ['YEARMONTH'] + [f"{col[0]}_{col[1]}" for col in p

# Check the structure of the pivot table
print(pre_trial_percentage_diff.head())

# Rename columns for clarity
```

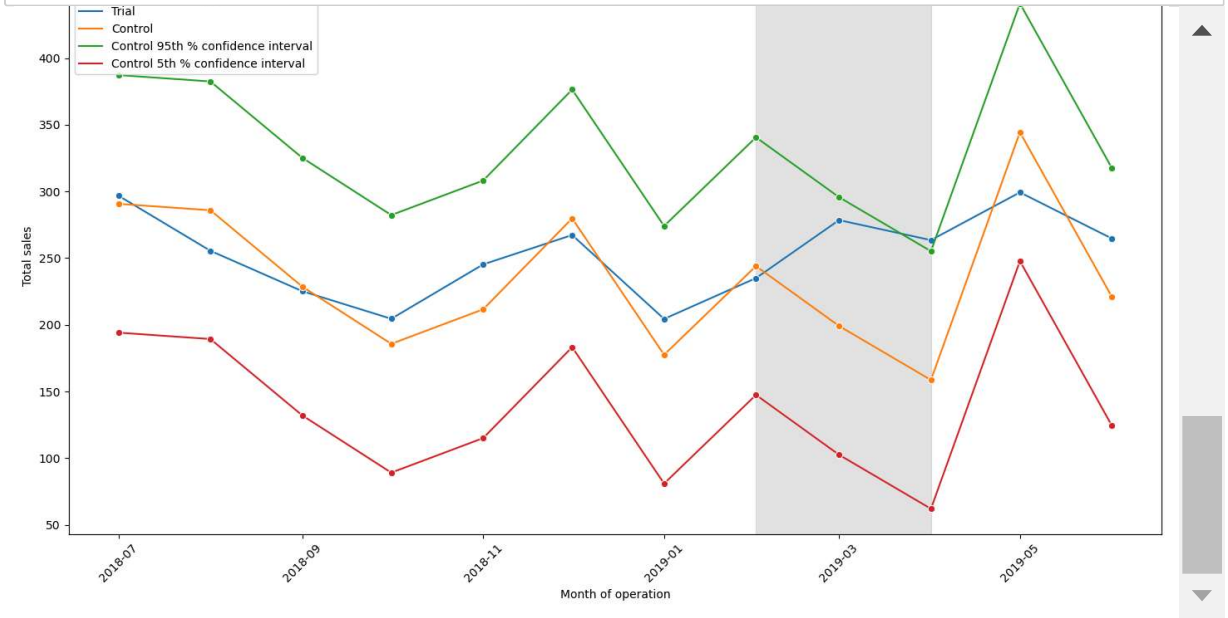
```
pre_trial_percentage_diff.rename(columns={
    'totSales_77': 'trialSales',
    'scaledControlSales_233': 'scaledControlSales'
}, inplace=True)

# Calculate the percentage difference for the pre-trial period
pre_trial_percentage_diff['percentageDiff'] = (
    (pre_trial_percentage_diff['trialSales'] - pre_trial_percentage_diff['scaledContro
    pre_trial_percentage_diff['scaledControlSales']
) * 100

# Calculate standard deviation of pre-trial percentage differences
std_dev = pre_trial_percentage_diff['percentageDiff'].std()
degrees_of_freedom = len(pre_trial_percentage_diff) - 1

# Merge the trial and scaled control sales data for the trial period
trial_percentage_diff = filtered_sales[
    (filtered_sales['YEARMONTH'] >= trial_period_start) &
    (filtered_sales['YEARMONTH'] <= trial_period_end)
].pivot_table(
    index='YEARMONTH',
    columns='STORE_NBR',
    values=['totSales', 'scaledControlSales'],
    aggfunc='sum'
).reset_index()

# Flatten the multi-level columns
trial_percentage_diff.columns = ['YEARMONTH'] + [f"{col[0]}_{col[1]}" for col in trial

# Check the structure of the pivot table for the trial period
print(trial_percentage_diff.head())

# Rename columns for clarity
trial_percentage_diff.rename(columns={
    'totSales_77': 'trialSales',
    'scaledControlSales_233': 'scaledControlSales'
}, inplace=True)

# Calculate the percentage difference for the trial period
trial_percentage_diff['percentageDiff'] = (
    (trial_percentage_diff['trialSales'] - trial_percentage_diff['scaledControlSales']
    trial_percentage_diff['scaledControlSales']
) * 100

# Calculate t-values for the trial months
trial_percentage_diff['tValue'] = (
    trial_percentage_diff['percentageDiff'] / std_dev
)

# Add transaction month for clarity
trial_percentage_diff['TransactionMonth'] = trial_percentage_diff['YEARMONTH']

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

# Print results
print("Standard Deviation of Pre-trial Percentage Differences:", std_dev)
print("Degrees of Freedom:", degrees_of_freedom)
print("t Critical Value (95th percentile):", t_critical)
print("Percentage Differences and t-Values for Trial Months:")
print(trial_percentage_diff[['TransactionMonth', 'percentageDiff', 'tValue']])
```

```
     STORE_NBR YEARMONTH  totSales  nCustomers  nTxnPerCust  nChipsPerTxn  \
880         77    201807     296.8          51           55      1.527273
881         77    201808     255.5          47           48      1.541667
882         77    201809     225.2          42           44      1.590909
883         77    201810     204.5          37           38      1.368421
884         77    201811     245.3          41           44      1.522727

     avgPricePerUnit  scaledControlSales
880         5.396364                 NaN
881         5.322917                 NaN
882         5.118182                 NaN
883         5.381579                 NaN
884         5.575000                 NaN
  YEARMONTH  scaledControlSales_77  scaledControlSales_233  totSales_77  \
0    201807                    0.0              297.565550        296.8
1    201808                    0.0              292.652187        255.5
2    201809                    0.0              233.998916        225.2
3    201810                    0.0              190.085733        204.5
4    201811                    0.0              216.597421        245.3

   totSales_233
0         290.7
1         285.9
2         228.6
3         185.7
4         211.6
  YEARMONTH  scaledControlSales_77  scaledControlSales_233  totSales_77  \
0    201902                    0.0              249.762622        235.0
1    201903                    0.0              203.802205        278.5
2    201904                    0.0              162.345704        263.5

   totSales_233
0         244.0
1         199.1
2         158.6
Standard Deviation of Pre-trial Percentage Differences: 9.958646884078389
Degrees of Freedom: 6
t Critical Value (95th percentile): 1.9431802803927816
Percentage Differences and t-Values for Trial Months:
  TransactionMonth  percentageDiff    tValue
0           201902       -5.910661 -0.593520
1           201903       36.652103  3.680430
2           201904       62.307960  6.256669
```

In [45]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming monthly_metrics is already defined as your main DataFrame
# Convert YEARMONTH to integer if it's not already
monthly_metrics['YEARMONTH'] = monthly_metrics['YEARMONTH'].astype(int)

# Define trial and control stores
trial_store = 77
control_store = 233

# Define the pre-trial and trial periods
pre_trial_period_end = 201901
trial_period_start = 201902
trial_period_end = 201904

# Calculate the total sales in the pre-trial period for both trial and control stores
pre_trial_sales_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

pre_trial_sales_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

# Calculate the scaling factor
scaling_factor = pre_trial_sales_trial / pre_trial_sales_control

# Print scaling factor for debugging
print(f"Scaling factor: {scaling_factor}")

# Apply the scaling factor to the control store sales during the pre-trial and trial pe
monthly_metrics.loc[
    (monthly_metrics['STORE_NBR'] == control_store),
    'scaledControlSales'
] = monthly_metrics['totSales'] * scaling_factor

# Filter for the trial and control stores only
filtered_sales = monthly_metrics[
    monthly_metrics['STORE_NBR'].isin([trial_store, control_store])
]

# Create a new column for store type
filtered_sales['Store_type'] = np.where(
    filtered_sales['STORE_NBR'] == trial_store, 'Trial', 'Control'
)

# Create a new column for TransactionMonth
filtered_sales['TransactionMonth'] = pd.to_datetime(filtered_sales['YEARMONTH'].astype

# Calculate the 95th and 5th percentiles for the control store
std_dev = filtered_sales[
    (filtered_sales['Store_type'] == 'Control') &
    (filtered_sales['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].std()

# Print standard deviation for debugging
print(f"Standard Deviation: {std_dev}")
```

```
filtered_sales_95 = filtered_sales[filtered_sales['Store_type'] == 'Control'].copy()
filtered_sales_95['totSales'] = filtered_sales_95['totSales'] + (std_dev * 2)
filtered_sales_95['Store_type'] = 'Control 95th % confidence interval'

filtered_sales_5 = filtered_sales[filtered_sales['Store_type'] == 'Control'].copy()
filtered_sales_5['totSales'] = filtered_sales_5['totSales'] - (std_dev * 2)
filtered_sales_5['Store_type'] = 'Control 5th % confidence interval'

# Combine all the data
trial_assessment = pd.concat([filtered_sales, filtered_sales_95, filtered_sales_5])

# Print the first few rows of the combined data for debugging
print(trial_assessment.head())

# Plotting
plt.figure(figsize=(14, 8))
sns.lineplot(data=trial_assessment, x='TransactionMonth', y='totSales', hue='Store_typ

# Highlight the trial period
plt.axvspan(pd.to_datetime('201902', format='%Y%m'), pd.to_datetime('201904', format='

# Labels and title
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Total sales by month')
plt.legend(title='Store Type')
plt.xticks(rotation=45)
plt.tight_layout()

# Show plot
plt.show()
```



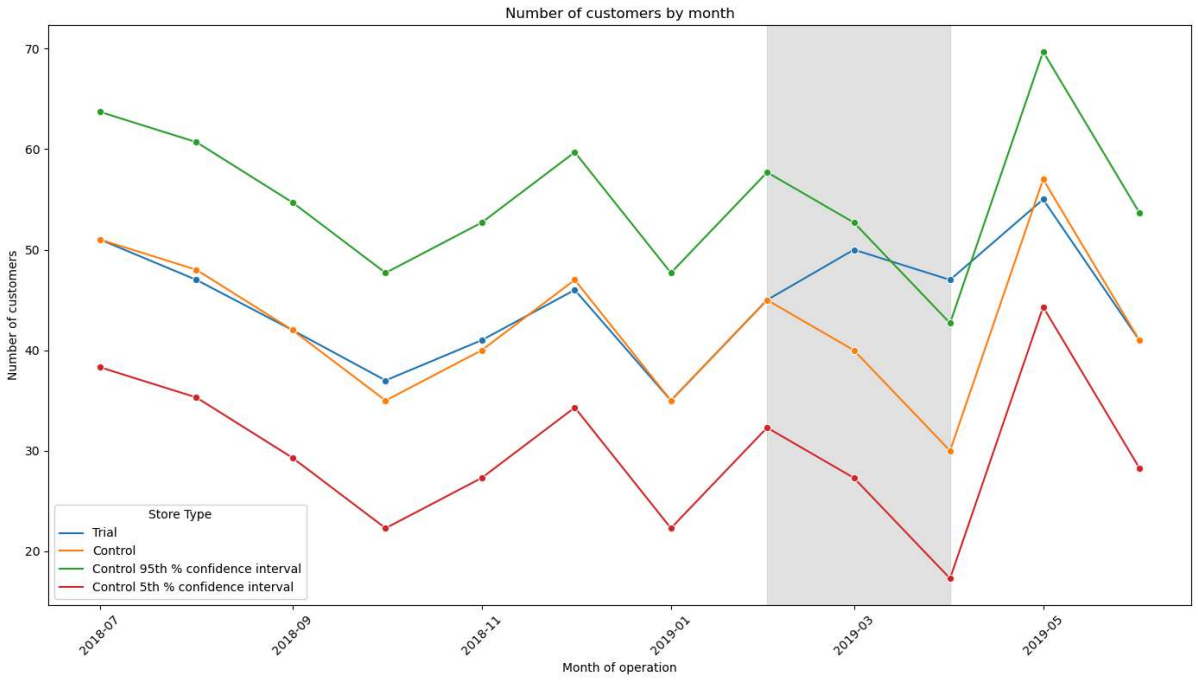
In [46]:

```
# Calculate the total number of customers in the pre-trial period for both trial and c
pre_trial_customers_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['nCustomers'].sum()

pre_trial_customers_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['nCustomers'].sum()

# Calculate the scaling factor
scaling_factor_customers = pre_trial_customers_trial / pre_trial_customers_control

# Print scaling factor for debugging
print(f"Scaling factor for customers: {scaling_factor_customers}")

# Apply the scaling factor to the control store customers during the pre-trial and tri
monthly_metrics.loc[
    (monthly_metrics['STORE_NBR'] == control_store),
    'scaledControlCustomers'
] = monthly_metrics['nCustomers'] * scaling_factor_customers

# Filter for the trial and control stores only
filtered_customers = monthly_metrics[
    monthly_metrics['STORE_NBR'].isin([trial_store, control_store])
]

# Create a new column for store type
filtered_customers['Store_type'] = np.where(
    filtered_customers['STORE_NBR'] == trial_store, 'Trial', 'Control'
)

# Create a new column for TransactionMonth
filtered_customers['TransactionMonth'] = pd.to_datetime(filtered_customers['YEARMONTH'

# Calculate the 95th and 5th percentiles for the control store
std_dev_customers = filtered_customers[
    (filtered_customers['Store_type'] == 'Control') &
    (filtered_customers['YEARMONTH'] <= pre_trial_period_end)
]['nCustomers'].std()

# Print standard deviation for debugging
print(f"Standard Deviation for customers: {std_dev_customers}")

filtered_customers_95 = filtered_customers[filtered_customers['Store_type'] == 'Contro
filtered_customers_95['nCustomers'] = filtered_customers_95['nCustomers'] + (std_dev_c
filtered_customers_95['Store_type'] = 'Control 95th % confidence interval'

filtered_customers_5 = filtered_customers[filtered_customers['Store_type'] == 'Control
filtered_customers_5['nCustomers'] = filtered_customers_5['nCustomers'] - (std_dev_cus
filtered_customers_5['Store_type'] = 'Control 5th % confidence interval'

# Combine all the data
trial_assessment_customers = pd.concat([filtered_customers, filtered_customers_95, fil

# Print the first few rows of the combined data for debugging
print(trial_assessment_customers.head())

# Plotting
plt.figure(figsize=(14, 8))
sns.lineplot(data=trial_assessment_customers, x='TransactionMonth', y='nCustomers', hu
```

```
# Highlight the trial period
plt.axvspan(pd.to_datetime('201902', format='%Y%m'), pd.to_datetime('201904', format='%

# Labels and title
plt.xlabel('Month of operation')
plt.ylabel('Number of customers')
plt.title('Number of customers by month')
plt.legend(title='Store Type')
plt.xticks(rotation=45)
plt.tight_layout()

# Show plot
plt.show()
```

```
Scaling factor for customers: 1.0033557046979866
Standard Deviation for customers: 6.347102826149446
     STORE_NBR  YEARMONTH  totSales  nCustomers  nTxnPerCust  nChipsPerTxn  \
880         77     201807     296.8        51.0           55      1.527273
881         77     201808     255.5        47.0           48      1.541667
882         77     201809     225.2        42.0           44      1.590909
883         77     201810     204.5        37.0           38      1.368421
884         77     201811     245.3        41.0           44      1.522727

     avgPricePerUnit  scaledControlSales  scaledControlCustomers Store_type  \
880         5.396364                 NaN                     NaN      Trial
881         5.322917                 NaN                     NaN      Trial
882         5.118182                 NaN                     NaN      Trial
883         5.381579                 NaN                     NaN      Trial
884         5.575000                 NaN                     NaN      Trial

    TransactionMonth
880       2018-07-01
881       2018-08-01
882       2018-09-01
883       2018-10-01
884       2018-11-01
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_10192\3103596605.py:30: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_customers['Store_type'] = np.where(
C:\Users\Tonye\AppData\Local\Temp\ipykernel_10192\3103596605.py:35: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_customers['TransactionMonth'] = pd.to_datetime(filtered_customers['YEARMON
TH'].astype(str), format='%Y%m')
C:\Users\Tonye\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Tonye\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [7]:
```
#Analysis for Store 86

#Defining the trial store
#Calculate Correlation and Magnitude scores for both sales and customes
trial_store = 86
corr_n_sales = calculate_correlation(pre_trial_measures, 'totSales', trial_store)
corr_n_customers = calculate_correlation(pre_trial_measures, 'nCustomers', trial_store
magnitude_n_sales = calculate_magnitude_distance(pre_trial_measures, 'totSales', trial
magnitude_n_customers = calculate_magnitude_distance(pre_trial_measures, 'nCustomers',
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\4097937659.py:17: FutureWarning: Th
e 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 determinin
g the result dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  corr_table = pd.concat([corr_table, new_row], ignore_index=True)
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\4097937659.py:17: FutureWarning: Th
e 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 determinin
g the result dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  corr_table = pd.concat([corr_table, new_row], ignore_index=True)
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\4097937659.py:37: FutureWarning: Th
e 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 determinin
g the result dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  dist_table = pd.concat([dist_table, new_rows], ignore_index=True)
```

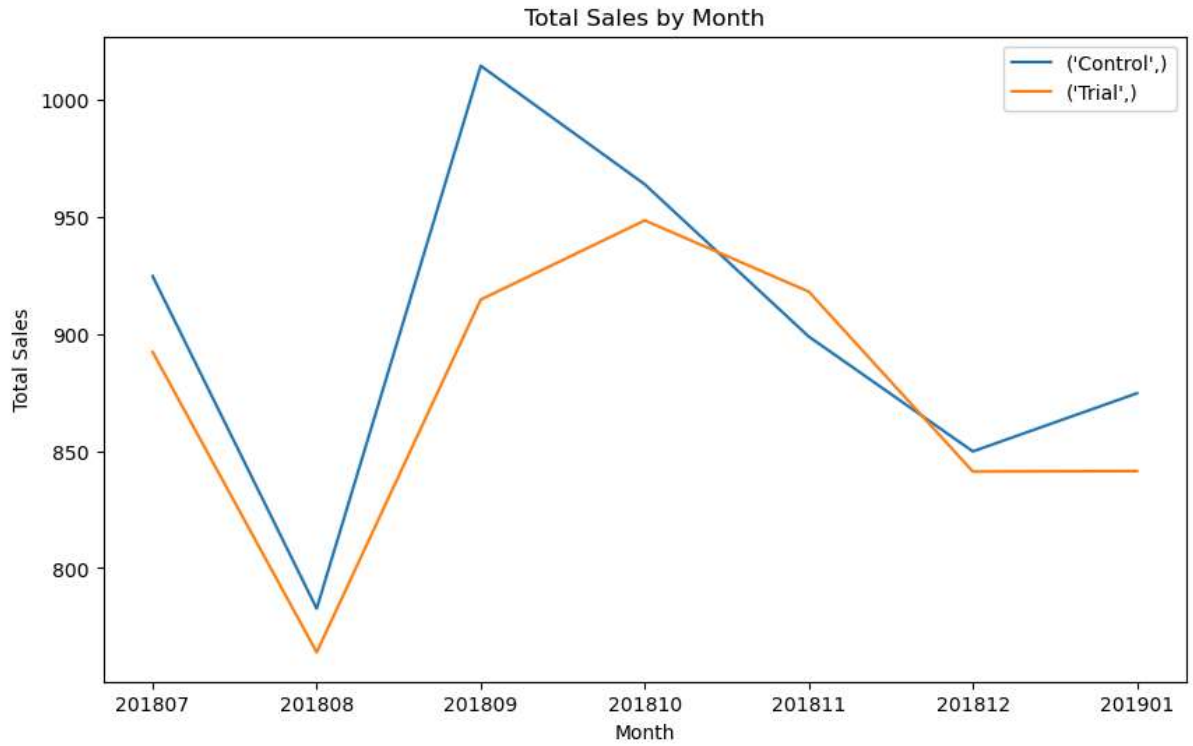
In [8]:
```
# Create a combined score composed of correlation and magnitude, by first merging the
corr_weight = 0.5
score_n_sales = corr_n_sales.merge(magnitude_n_sales, on=['Store1', 'Store2'])
score_n_sales['scoreNSales'] = corr_weight * score_n_sales['corr_measure'] + (1 - corr

score_n_customers = corr_n_customers.merge(magnitude_n_customers, on=['Store1', 'Store
score_n_customers['scoreNCust'] = corr_weight * score_n_customers['corr_measure'] + (1

score_control = score_n_sales.merge(score_n_customers, on=['Store1', 'Store2'])
score_control['finalControlScore'] = score_control['scoreNSales'] * 0.5 + score_contro
```

In [9]:
```
#Select Control Stores based on the control score combining correlation and magnitude
control_store = score_control.loc[score_control['finalControlScore'].idxmax(), 'Store2
print("Control Store:", control_store)
```

```
Control Store: 155
```

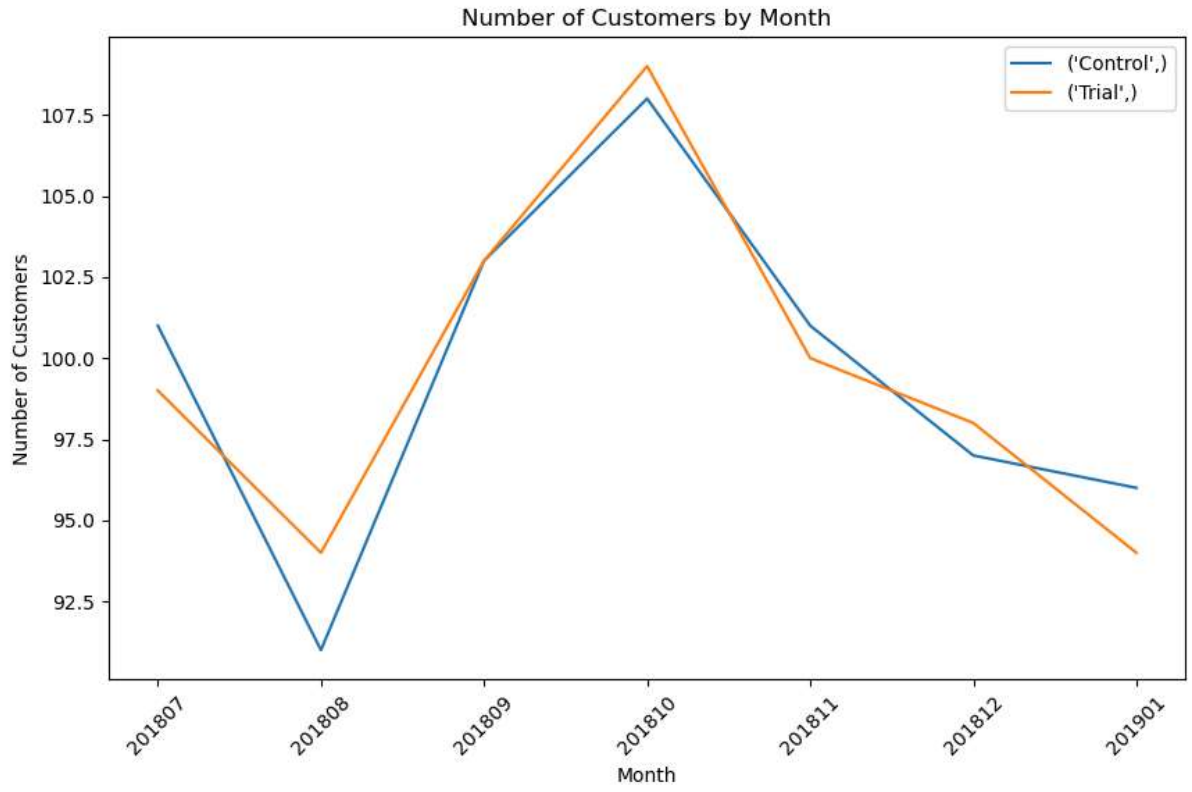
In [10]:
```
#Visual check to see if the control store is indeed similar to the trial store before t
#Visual Check on Sales

# Plot total sales
pre_trial_sales = monthly_metrics[
    (monthly_metrics['STORE_NBR'].isin([trial_store, control_store])) &
    (monthly_metrics['YEARMONTH'] < '201902')
]
pre_trial_sales['Store_type'] = pre_trial_sales['STORE_NBR'].apply(lambda x: 'Trial' i

plt.figure(figsize=(10, 6))
for key, grp in pre_trial_sales.groupby(['Store_type']):
    plt.plot(grp['YEARMONTH'], grp['totSales'], label=key)
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.title('Total Sales by Month')
plt.legend(loc='best')
plt.show()
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\4020865328.py:9: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  pre_trial_sales['Store_type'] = pre_trial_sales['STORE_NBR'].apply(lambda x: 'Tria
l' if x == trial_store else 'Control')
```



In [11]:
```
#Visualise number of customers



# Plot number of customers
pre_trial_customers = monthly_metrics[
    (monthly_metrics['STORE_NBR'].isin([trial_store, control_store])) &
    (monthly_metrics['YEARMONTH'] < '201902')
]
pre_trial_customers['Store_type'] = pre_trial_customers['STORE_NBR'].apply(lambda x: '

plt.figure(figsize=(10, 6))
for key, grp in pre_trial_customers.groupby(['Store_type']):
    plt.plot(grp['YEARMONTH'], grp['nCustomers'], label=key)
plt.xlabel('Month')
plt.ylabel('Number of Customers')
plt.title('Number of Customers by Month')
plt.legend(loc='best')
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\1492363467.py:12: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  pre_trial_customers['Store_type'] = pre_trial_customers['STORE_NBR'].apply(lambda
x: 'Trial' if x == trial_store else 'Control')
```



In [12]:
```
#Assess trail store performance
#with scaling the control store's sales to a level similar to control for any differen
#outside of the trial period.

#Scale store sales
scaling_factor = pre_trial_measures[pre_trial_measures['STORE_NBR'] == trial_store]['t
                 pre_trial_measures[pre_trial_measures['STORE_NBR'] == control_store][
scaled_control_sales = monthly_metrics[monthly_metrics['STORE_NBR'] == control_store]
scaled_control_sales['scaledSales'] = scaled_control_sales['totSales'] * scaling_facto
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\225660109.py:9: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  scaled_control_sales['scaledSales'] = scaled_control_sales['totSales'] * scaling_fa
ctor
```

In [13]:
```
#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
trial_period = monthly_metrics[(monthly_metrics['YEARMONTH'] >= '201902') & (monthly_m
percentage_diff = trial_period.merge(scaled_control_sales, on='YEARMONTH')
percentage_diff['percentageDiff'] = (percentage_diff['totSales_x'] - percentage_diff['
```

In [17]:

```
# Define trial and control stores
trial_store = 86
control_store = 155

# Define the pre-trial and trial periods
pre_trial_period_end = '201901'
trial_period_start = '201902'
trial_period_end = '201904'

# Calculate the total sales in the pre-trial period for both trial and control stores
pre_trial_sales_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

pre_trial_sales_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

# Calculate the scaling factor
scaling_factor = pre_trial_sales_trial / pre_trial_sales_control

# Copy the original dataframe to measure_over_time_sales
measure_over_time_sales = monthly_metrics.copy()

# Apply the scaling factor to the control store sales during the trial period
measure_over_time_sales.loc[
    (measure_over_time_sales['STORE_NBR'] == control_store) &
    (measure_over_time_sales['YEARMONTH'] >= trial_period_start) &
    (measure_over_time_sales['YEARMONTH'] <= trial_period_end),
    'scaledControlSales'
] = measure_over_time_sales['totSales'] * scaling_factor

# Merge the trial and scaled control sales data
percentage_diff = measure_over_time_sales[
    (measure_over_time_sales['STORE_NBR'].isin([trial_store, control_store])) &
    (measure_over_time_sales['YEARMONTH'] >= trial_period_start) &
    (measure_over_time_sales['YEARMONTH'] <= trial_period_end)
].pivot_table(
    index='YEARMONTH',
    columns='STORE_NBR',
    values=['totSales', 'scaledControlSales']
).reset_index()

# Print the columns to understand the structure
print(percentage_diff.columns)

# Adjust column renaming and calculation accordingly
# The column names in the DataFrame will be multi-indexed, so we need to handle that
percentage_diff.columns = ['YEARMONTH', 'controlSales', 'trialSales', 'scaledControlSa

# Calculate the percentage difference
percentage_diff['percentageDiff'] = (
    (percentage_diff['trialSales'] - percentage_diff['scaledControlSales']) /
    percentage_diff['scaledControlSales']
) * 100

# Print the result
print(percentage_diff)
```

```
MultiIndex([(         'YEARMONTH',  ''),
            ('scaledControlSales', 155),
            (          'totSales',  86),
            (          'totSales', 155)],
           names=[None, 'STORE_NBR'])
```

	YEARMONTH	controlSales	trialSales	scaledControlSales	percentageDiff
0	201902	864.522060	913.2	891.2	2.468582
1	201903	780.320405	1026.8	804.4	27.647936
2	201904	819.317024	848.2	844.6	0.426237

In [19]:

```
#Anayse the significance of the percentage difference through the t-value and standard


# Define trial and control stores
trial_store = 86
control_store = 155

# Define the pre-trial and trial periods
pre_trial_period_end = '201901'
trial_period_start = '201902'
trial_period_end = '201904'

# Calculate the total sales in the pre-trial period for both trial and control stores
pre_trial_sales_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

pre_trial_sales_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

# Calculate the scaling factor
scaling_factor = pre_trial_sales_trial / pre_trial_sales_control

# Copy the original dataframe to measure_over_time_sales
measure_over_time_sales = monthly_metrics.copy()

# Apply the scaling factor to the control store sales during the pre-trial and trial pe
measure_over_time_sales.loc[
    (measure_over_time_sales['STORE_NBR'] == control_store),
    'scaledControlSales'
] = measure_over_time_sales['totSales'] * scaling_factor

# Filter for the trial and control stores only
filtered_sales = measure_over_time_sales[
    measure_over_time_sales['STORE_NBR'].isin([trial_store, control_store])
]

# Check the structure of the filtered_sales DataFrame
print(filtered_sales.head())

# Create a pivot table
pre_trial_percentage_diff = filtered_sales[
    (filtered_sales['YEARMONTH'] <= pre_trial_period_end)
].pivot_table(
    index='YEARMONTH',
    columns='STORE_NBR',
    values=['totSales', 'scaledControlSales'],
    aggfunc='sum'
).reset_index()

# Flatten the multi-level columns
pre_trial_percentage_diff.columns = ['YEARMONTH'] + [f"{col[0]}_{col[1]}" for col in p

# Check the structure of the pivot table
print(pre_trial_percentage_diff.head())

# Rename columns for clarity
pre_trial_percentage_diff.rename(columns={
    'totSales_86': 'trialSales',
```

```
    'scaledControlSales_155': 'scaledControlSales'
}, inplace=True)

# Calculate the percentage difference for the pre-trial period
pre_trial_percentage_diff['percentageDiff'] = (
    (pre_trial_percentage_diff['trialSales'] - pre_trial_percentage_diff['scaledContro
    pre_trial_percentage_diff['scaledControlSales']
) * 100

# Calculate standard deviation of pre-trial percentage differences
std_dev = pre_trial_percentage_diff['percentageDiff'].std()
degrees_of_freedom = len(pre_trial_percentage_diff) - 1

# Merge the trial and scaled control sales data for the trial period
trial_percentage_diff = filtered_sales[
    (filtered_sales['YEARMONTH'] >= trial_period_start) &
    (filtered_sales['YEARMONTH'] <= trial_period_end)
].pivot_table(
    index='YEARMONTH',
    columns='STORE_NBR',
    values=['totSales', 'scaledControlSales'],
    aggfunc='sum'
).reset_index()

# Flatten the multi-level columns
trial_percentage_diff.columns = ['YEARMONTH'] + [f"{col[0]}_{col[1]}" for col in trial

# Check the structure of the pivot table for the trial period
print(trial_percentage_diff.head())

# Rename columns for clarity
trial_percentage_diff.rename(columns={
    'totSales_86': 'trialSales',
    'scaledControlSales_155': 'scaledControlSales'
}, inplace=True)

# Calculate the percentage difference for the trial period
trial_percentage_diff['percentageDiff'] = (
    (trial_percentage_diff['trialSales'] - trial_percentage_diff['scaledControlSales']
    trial_percentage_diff['scaledControlSales']
) * 100

# Calculate t-values for the trial months
trial_percentage_diff['tValue'] = (
    trial_percentage_diff['percentageDiff'] / std_dev
)

# Add transaction month for clarity
trial_percentage_diff['TransactionMonth'] = trial_percentage_diff['YEARMONTH']

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

# Print results
print("Standard Deviation of Pre-trial Percentage Differences:", std_dev)
print("Degrees of Freedom:", degrees_of_freedom)
print("t Critical Value (95th percentile):", t_critical)
print("Percentage Differences and t-Values for Trial Months:")
print(trial_percentage_diff[['TransactionMonth', 'percentageDiff', 'tValue']])
```

```
     STORE_NBR YEARMONTH  totSales  nCustomers  nTxnPerCust  nChipsPerTxn  \
977         86    201807    892.20          99          126      1.992063
978         86    201808    764.05          94          112      1.919643
979         86    201809    914.60         103          129      2.000000
980         86    201810    948.40         109          138      2.000000
981         86    201811    918.00         100          127      2.000000

     avgPricePerUnit  scaledControlSales
977         7.080952                 NaN
978         6.821875                 NaN
979         7.089922                 NaN
980         6.872464                 NaN
981         7.228346                 NaN
  YEARMONTH  scaledControlSales_86  scaledControlSales_155  totSales_86  \
0    201807                    0.0              896.922236       892.20
1    201808                    0.0              759.269991       764.05
2    201809                    0.0              984.034086       914.60
3    201810                    0.0              934.948790       948.40
4    201811                    0.0              871.894555       918.00

   totSales_155
0         924.6
1         782.7
2        1014.4
3         963.8
4         898.8
  YEARMONTH  scaledControlSales_86  scaledControlSales_155  totSales_86  \
0    201902                    0.0              864.522060        913.2
1    201903                    0.0              780.320405       1026.8
2    201904                    0.0              819.317024        848.2

   totSales_155
0         891.2
1         804.4
2         844.6
Standard Deviation of Pre-trial Percentage Differences: 3.768532790008376
Degrees of Freedom: 6
t Critical Value (95th percentile): 1.9431802803927816
Percentage Differences and t-Values for Trial Months:
  TransactionMonth  percentageDiff    tValue
0           201902        5.630619  1.494114
1           201903       31.586973  8.381769
2           201904        3.525250  0.935444
```

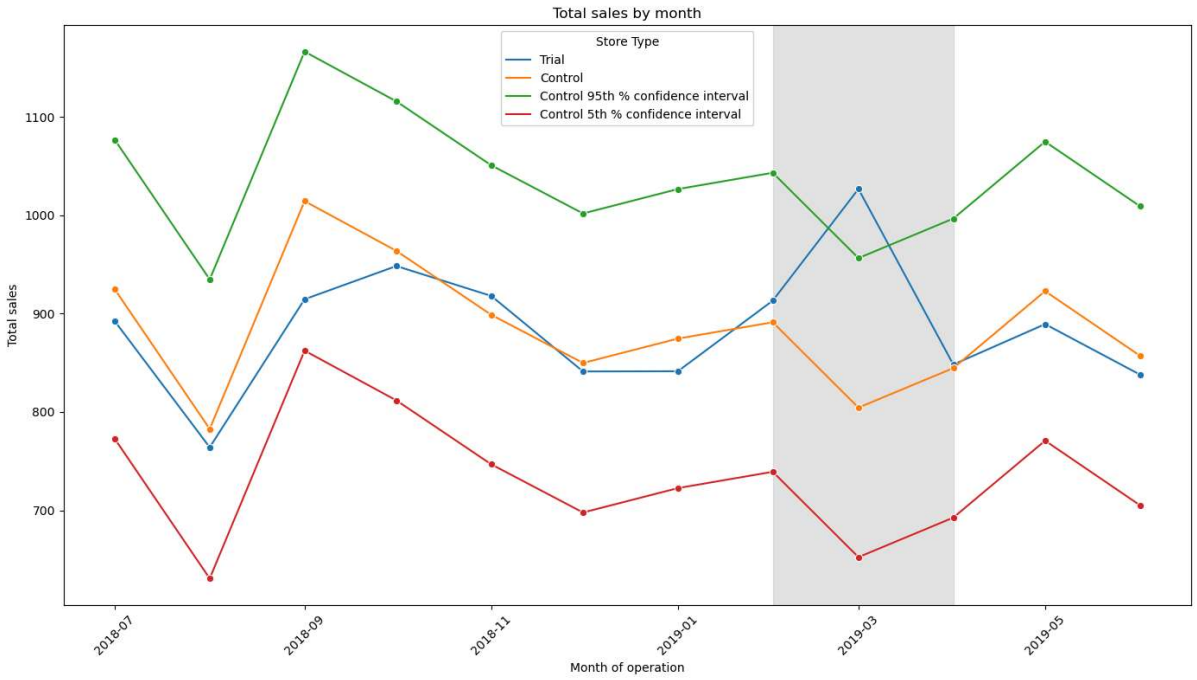
In [20]:

```
# Assuming monthly_metrics is already defined as your main DataFrame
# Convert YEARMONTH to integer if it's not already
monthly_metrics['YEARMONTH'] = monthly_metrics['YEARMONTH'].astype(int)

# Define trial and control stores
trial_store = 86
control_store = 155

# Define the pre-trial and trial periods
pre_trial_period_end = 201901
trial_period_start = 201902
trial_period_end = 201904

# Calculate the total sales in the pre-trial period for both trial and control stores
pre_trial_sales_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

pre_trial_sales_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

# Calculate the scaling factor
scaling_factor = pre_trial_sales_trial / pre_trial_sales_control

# Print scaling factor for debugging
print(f"Scaling factor: {scaling_factor}")

# Apply the scaling factor to the control store sales during the pre-trial and trial pe
monthly_metrics.loc[
    (monthly_metrics['STORE_NBR'] == control_store),
    'scaledControlSales'
] = monthly_metrics['totSales'] * scaling_factor

# Filter for the trial and control stores only
filtered_sales = monthly_metrics[
    monthly_metrics['STORE_NBR'].isin([trial_store, control_store])
]

# Create a new column for store type
filtered_sales['Store_type'] = np.where(
    filtered_sales['STORE_NBR'] == trial_store, 'Trial', 'Control'
)

# Create a new column for TransactionMonth
filtered_sales['TransactionMonth'] = pd.to_datetime(filtered_sales['YEARMONTH'].astype

# Calculate the 95th and 5th percentiles for the control store
std_dev = filtered_sales[
    (filtered_sales['Store_type'] == 'Control') &
    (filtered_sales['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].std()

# Print standard deviation for debugging
print(f"Standard Deviation: {std_dev}")

filtered_sales_95 = filtered_sales[filtered_sales['Store_type'] == 'Control'].copy()
filtered_sales_95['totSales'] = filtered_sales_95['totSales'] + (std_dev * 2)
```

```
filtered_sales_95['Store_type'] = 'Control 95th % confidence interval'

filtered_sales_5 = filtered_sales[filtered_sales['Store_type'] == 'Control'].copy()
filtered_sales_5['totSales'] = filtered_sales_5['totSales'] - (std_dev * 2)
filtered_sales_5['Store_type'] = 'Control 5th % confidence interval'

# Combine all the data
trial_assessment = pd.concat([filtered_sales, filtered_sales_95, filtered_sales_5])

# Print the first few rows of the combined data for debugging
print(trial_assessment.head())

# Plotting
plt.figure(figsize=(14, 8))
sns.lineplot(data=trial_assessment, x='TransactionMonth', y='totSales', hue='Store_typ

# Highlight the trial period
plt.axvspan(pd.to_datetime('201902', format='%Y%m'), pd.to_datetime('201904', format='

# Labels and title
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Total sales by month')
plt.legend(title='Store Type')
plt.xticks(rotation=45)
plt.tight_layout()

# Show plot
plt.show()
```

```
Scaling factor: 0.9700651481287743
Standard Deviation: 76.02420353946184
     STORE_NBR  YEARMONTH  totSales  nCustomers  nTxnPerCust  nChipsPerTxn  \
977         86     201807    892.20          99          126      1.992063
978         86     201808    764.05          94          112      1.919643
979         86     201809    914.60         103          129      2.000000
980         86     201810    948.40         109          138      2.000000
981         86     201811    918.00         100          127      2.000000

     avgPricePerUnit  scaledControlSales Store_type TransactionMonth
977         7.080952                 NaN      Trial       2018-07-01
978         6.821875                 NaN      Trial       2018-08-01
979         7.089922                 NaN      Trial       2018-09-01
980         6.872464                 NaN      Trial       2018-10-01
981         7.228346                 NaN      Trial       2018-11-01
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\237553517.py:48: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_sales['Store_type'] = np.where(
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\237553517.py:53: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_sales['TransactionMonth'] = pd.to_datetime(filtered_sales['YEARMONTH'].ast
ype(str), format='%Y%m')
C:\Users\Tonye\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Tonye\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



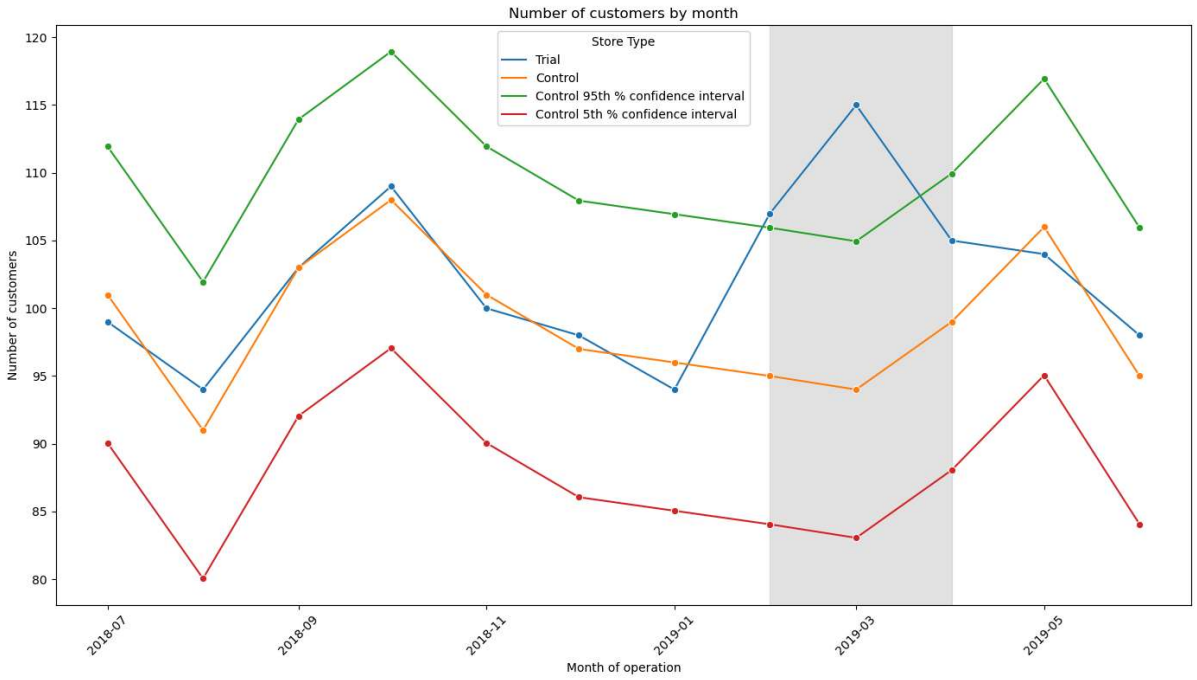
In [21]:

```
# Calculate the total number of customers in the pre-trial period for both trial and c
pre_trial_customers_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['nCustomers'].sum()

pre_trial_customers_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['nCustomers'].sum()

# Calculate the scaling factor
scaling_factor_customers = pre_trial_customers_trial / pre_trial_customers_control

# Print scaling factor for debugging
print(f"Scaling factor for customers: {scaling_factor_customers}")

# Apply the scaling factor to the control store customers during the pre-trial and tri
monthly_metrics.loc[
    (monthly_metrics['STORE_NBR'] == control_store),
    'scaledControlCustomers'
] = monthly_metrics['nCustomers'] * scaling_factor_customers

# Filter for the trial and control stores only
filtered_customers = monthly_metrics[
    monthly_metrics['STORE_NBR'].isin([trial_store, control_store])
]

# Create a new column for store type
filtered_customers['Store_type'] = np.where(
    filtered_customers['STORE_NBR'] == trial_store, 'Trial', 'Control'
)

# Create a new column for TransactionMonth
filtered_customers['TransactionMonth'] = pd.to_datetime(filtered_customers['YEARMONTH'

# Calculate the 95th and 5th percentiles for the control store
std_dev_customers = filtered_customers[
    (filtered_customers['Store_type'] == 'Control') &
    (filtered_customers['YEARMONTH'] <= pre_trial_period_end)
]['nCustomers'].std()

# Print standard deviation for debugging
print(f"Standard Deviation for customers: {std_dev_customers}")

filtered_customers_95 = filtered_customers[filtered_customers['Store_type'] == 'Contro
filtered_customers_95['nCustomers'] = filtered_customers_95['nCustomers'] + (std_dev_c
filtered_customers_95['Store_type'] = 'Control 95th % confidence interval'

filtered_customers_5 = filtered_customers[filtered_customers['Store_type'] == 'Control
filtered_customers_5['nCustomers'] = filtered_customers_5['nCustomers'] - (std_dev_cus
filtered_customers_5['Store_type'] = 'Control 5th % confidence interval'

# Combine all the data
trial_assessment_customers = pd.concat([filtered_customers, filtered_customers_95, fil

# Print the first few rows of the combined data for debugging
print(trial_assessment_customers.head())

# Plotting
plt.figure(figsize=(14, 8))
sns.lineplot(data=trial_assessment_customers, x='TransactionMonth', y='nCustomers', hu
```

```
# Highlight the trial period
plt.axvspan(pd.to_datetime('201902', format='%Y%m'), pd.to_datetime('201904', format='%

# Labels and title
plt.xlabel('Month of operation')
plt.ylabel('Number of customers')
plt.title('Number of customers by month')
plt.legend(title='Store Type')
plt.xticks(rotation=45)
plt.tight_layout()

# Show plot
plt.show()
```

```
Scaling factor for customers: 1.0
Standard Deviation for customers: 5.472876844254852
     STORE_NBR  YEARMONTH  totSales  nCustomers  nTxnPerCust  nChipsPerTxn  \
977         86     201807    892.20        99.0          126      1.992063
978         86     201808    764.05        94.0          112      1.919643
979         86     201809    914.60       103.0          129      2.000000
980         86     201810    948.40       109.0          138      2.000000
981         86     201811    918.00       100.0          127      2.000000

     avgPricePerUnit  scaledControlSales  scaledControlCustomers Store_type  \
977         7.080952                 NaN                     NaN      Trial
978         6.821875                 NaN                     NaN      Trial
979         7.089922                 NaN                     NaN      Trial
980         6.872464                 NaN                     NaN      Trial
981         7.228346                 NaN                     NaN      Trial

    TransactionMonth
977       2018-07-01
978       2018-08-01
979       2018-09-01
980       2018-10-01
981       2018-11-01
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\3103596605.py:30: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_customers['Store_type'] = np.where(
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\3103596605.py:35: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_customers['TransactionMonth'] = pd.to_datetime(filtered_customers['YEARMON
TH'].astype(str), format='%Y%m')
C:\Users\Tonye\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Tonye\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [22]:
```
#Repeating the process for Trial Store 88

#Calculate Correlation and Magnitude scores for both sales and customes
trial_store = 88
corr_n_sales = calculate_correlation(pre_trial_measures, 'totSales', trial_store)
corr_n_customers = calculate_correlation(pre_trial_measures, 'nCustomers', trial_store
magnitude_n_sales = calculate_magnitude_distance(pre_trial_measures, 'totSales', trial
magnitude_n_customers = calculate_magnitude_distance(pre_trial_measures, 'nCustomers',
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\4097937659.py:17: FutureWarning: Th
e 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 determinin
g the result dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  corr_table = pd.concat([corr_table, new_row], ignore_index=True)
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\4097937659.py:17: FutureWarning: Th
e 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 determinin
g the result dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  corr_table = pd.concat([corr_table, new_row], ignore_index=True)
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\4097937659.py:37: FutureWarning: Th
e 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 determinin
g the result dtypes. To retain the old behavior, exclude the relevant entries before
the concat operation.
  dist_table = pd.concat([dist_table, new_rows], ignore_index=True)
```

In [23]:
```
# Create a combined score composed of correlation and magnitude, by first merging the
corr_weight = 0.5
score_n_sales = corr_n_sales.merge(magnitude_n_sales, on=['Store1', 'Store2'])
score_n_sales['scoreNSales'] = corr_weight * score_n_sales['corr_measure'] + (1 - corr

score_n_customers = corr_n_customers.merge(magnitude_n_customers, on=['Store1', 'Store
score_n_customers['scoreNCust'] = corr_weight * score_n_customers['corr_measure'] + (1

score_control = score_n_sales.merge(score_n_customers, on=['Store1', 'Store2'])
score_control['finalControlScore'] = score_control['scoreNSales'] * 0.5 + score_contro
```

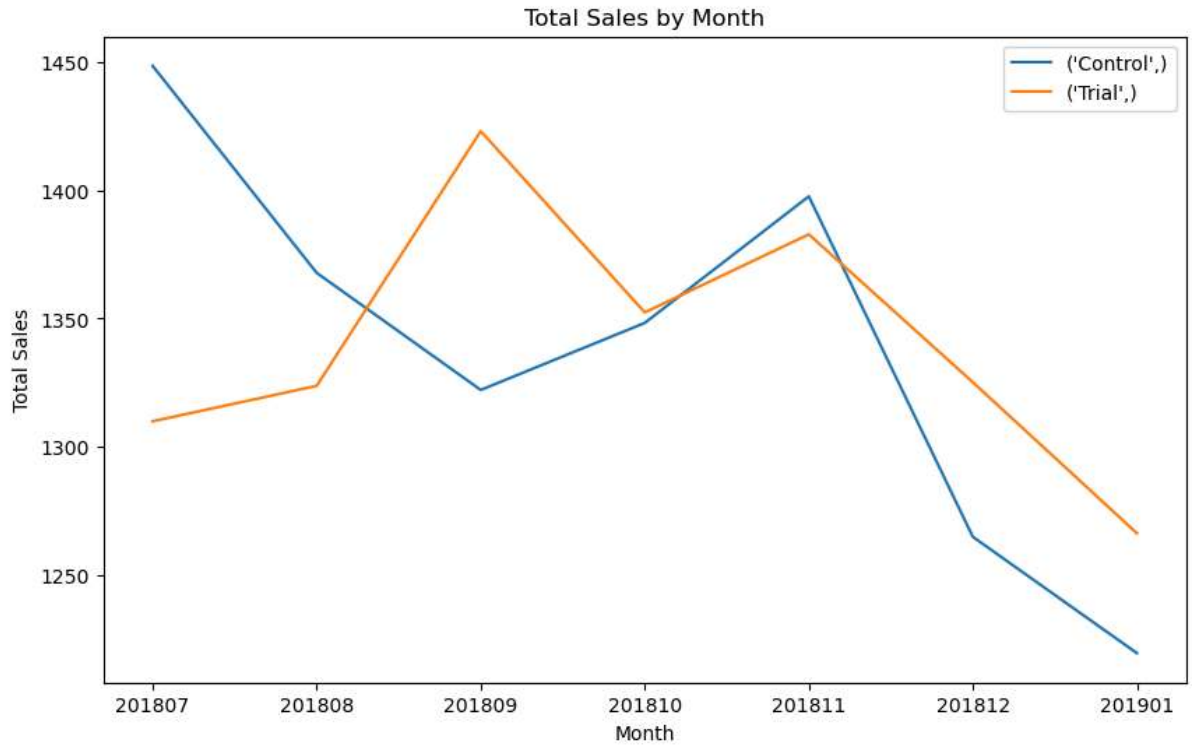
In [24]:
```
#Select Control Stores based on the control score combining correlation and magnitude
control_store = score_control.loc[score_control['finalControlScore'].idxmax(), 'Store2
print("Control Store:", control_store)
```

```
Control Store: 237
```

In [27]:

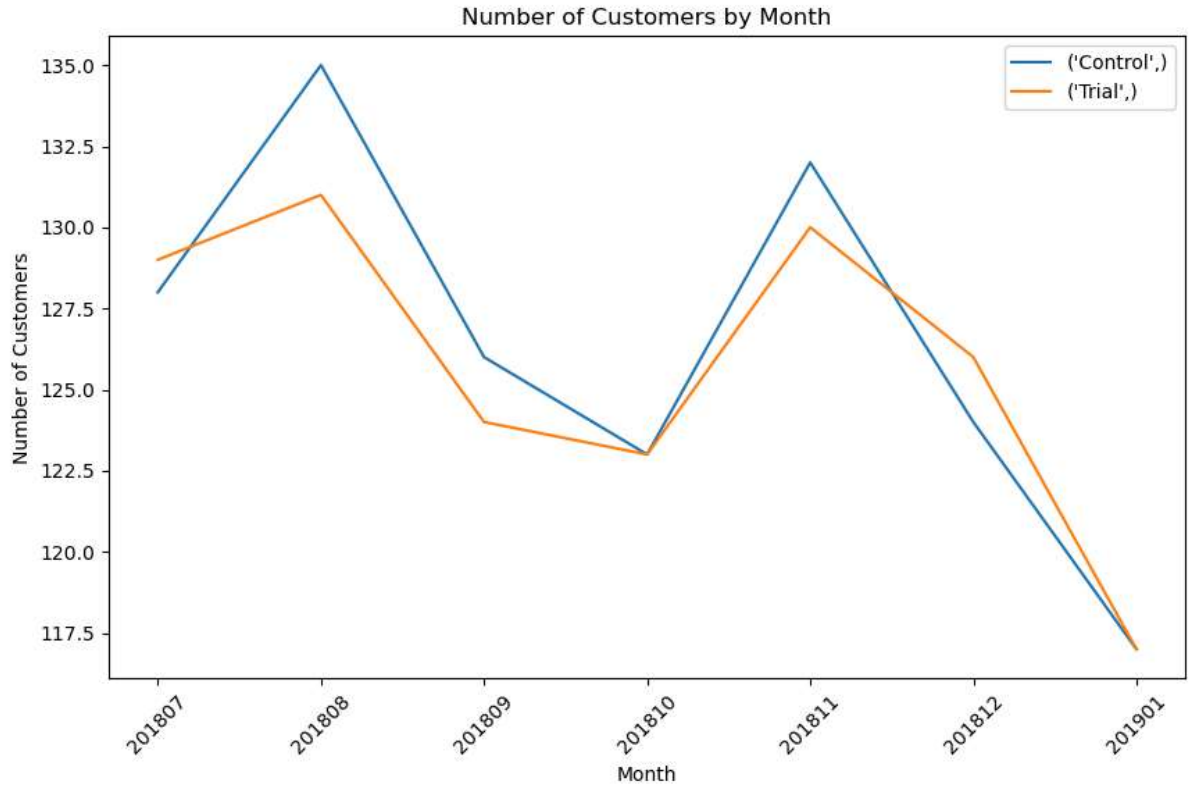
```
#Visual check to see if the control store is indeed similar to the trial store before t

# Convert YEARMONTH to string for comparison
monthly_metrics['YEARMONTH'] = monthly_metrics['YEARMONTH'].astype(str)

#Visual Check on Sales

# Plot total sales
pre_trial_sales = monthly_metrics[
    (monthly_metrics['STORE_NBR'].isin([trial_store, control_store])) &
    (monthly_metrics['YEARMONTH'] < '201902')
]
pre_trial_sales['Store_type'] = pre_trial_sales['STORE_NBR'].apply(lambda x: 'Trial' i

plt.figure(figsize=(10, 6))
for key, grp in pre_trial_sales.groupby(['Store_type']):
    plt.plot(grp['YEARMONTH'], grp['totSales'], label=key)
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.title('Total Sales by Month')
plt.legend(loc='best')
plt.show()
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\2328495815.py:13: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  pre_trial_sales['Store_type'] = pre_trial_sales['STORE_NBR'].apply(lambda x: 'Tria
l' if x == trial_store else 'Control')
```



In [28]:
```
#Visualise number of customers


# Plot number of customers
pre_trial_customers = monthly_metrics[
    (monthly_metrics['STORE_NBR'].isin([trial_store, control_store])) &
    (monthly_metrics['YEARMONTH'] < '201902')
]
pre_trial_customers['Store_type'] = pre_trial_customers['STORE_NBR'].apply(lambda x: '

plt.figure(figsize=(10, 6))
for key, grp in pre_trial_customers.groupby(['Store_type']):
    plt.plot(grp['YEARMONTH'], grp['nCustomers'], label=key)
plt.xlabel('Month')
plt.ylabel('Number of Customers')
plt.title('Number of Customers by Month')
plt.legend(loc='best')
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\507553536.py:9: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  pre_trial_customers['Store_type'] = pre_trial_customers['STORE_NBR'].apply(lambda
x: 'Trial' if x == trial_store else 'Control')
```



In [29]:
```
#Assess trail store performance
#with scaling the control store's sales to a level similar to control for any differen
#outside of the trial period.

#Scale store sales
scaling_factor = pre_trial_measures[pre_trial_measures['STORE_NBR'] == trial_store]['t
                 pre_trial_measures[pre_trial_measures['STORE_NBR'] == control_store][
scaled_control_sales = monthly_metrics[monthly_metrics['STORE_NBR'] == control_store]
scaled_control_sales['scaledSales'] = scaled_control_sales['totSales'] * scaling_facto
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\225660109.py:9: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  scaled_control_sales['scaledSales'] = scaled_control_sales['totSales'] * scaling_fa
ctor
```

In [30]:
```
#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
trial_period = monthly_metrics[(monthly_metrics['YEARMONTH'] >= '201902') & (monthly_m
percentage_diff = trial_period.merge(scaled_control_sales, on='YEARMONTH')
percentage_diff['percentageDiff'] = (percentage_diff['totSales_x'] - percentage_diff['
```

In [31]:

```
# Define trial and control stores
trial_store = 88
control_store = 237

# Define the pre-trial and trial periods
pre_trial_period_end = '201901'
trial_period_start = '201902'
trial_period_end = '201904'

# Calculate the total sales in the pre-trial period for both trial and control stores
pre_trial_sales_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

pre_trial_sales_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

# Calculate the scaling factor
scaling_factor = pre_trial_sales_trial / pre_trial_sales_control

# Copy the original dataframe to measure_over_time_sales
measure_over_time_sales = monthly_metrics.copy()

# Apply the scaling factor to the control store sales during the trial period
measure_over_time_sales.loc[
    (measure_over_time_sales['STORE_NBR'] == control_store) &
    (measure_over_time_sales['YEARMONTH'] >= trial_period_start) &
    (measure_over_time_sales['YEARMONTH'] <= trial_period_end),
    'scaledControlSales'
] = measure_over_time_sales['totSales'] * scaling_factor

# Merge the trial and scaled control sales data
percentage_diff = measure_over_time_sales[
    (measure_over_time_sales['STORE_NBR'].isin([trial_store, control_store])) &
    (measure_over_time_sales['YEARMONTH'] >= trial_period_start) &
    (measure_over_time_sales['YEARMONTH'] <= trial_period_end)
].pivot_table(
    index='YEARMONTH',
    columns='STORE_NBR',
    values=['totSales', 'scaledControlSales']
).reset_index()

# Print the columns to understand the structure
print(percentage_diff.columns)

# Adjust column renaming and calculation accordingly
# The column names in the DataFrame will be multi-indexed, so we need to handle that
percentage_diff.columns = ['YEARMONTH', 'controlSales', 'trialSales', 'scaledControlSa

# Calculate the percentage difference
percentage_diff['percentageDiff'] = (
    (percentage_diff['trialSales'] - percentage_diff['scaledControlSales']) /
    percentage_diff['scaledControlSales']
) * 100

# Print the result
print(percentage_diff)
```

```
MultiIndex([(         'YEARMONTH',  ''),
            ('scaledControlSales', 237),
            (          'totSales',  88),
            (          'totSales', 237)],
           names=[None, 'STORE_NBR'])
```

	YEARMONTH	controlSales	trialSales	scaledControlSales	percentageDiff
0	201902	1406.989143	1370.2	1404.8	-2.462984
1	201903	1210.082775	1477.2	1208.2	22.264526
2	201904	1206.477165	1439.4	1204.6	19.491948

In [32]:

```
#Anayse the significance of the percentage difference through the t-value and standard


# Define trial and control stores
trial_store = 88
control_store = 237

# Define the pre-trial and trial periods
pre_trial_period_end = '201901'
trial_period_start = '201902'
trial_period_end = '201904'

# Calculate the total sales in the pre-trial period for both trial and control stores
pre_trial_sales_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

pre_trial_sales_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

# Calculate the scaling factor
scaling_factor = pre_trial_sales_trial / pre_trial_sales_control

# Copy the original dataframe to measure_over_time_sales
measure_over_time_sales = monthly_metrics.copy()

# Apply the scaling factor to the control store sales during the pre-trial and trial pe
measure_over_time_sales.loc[
    (measure_over_time_sales['STORE_NBR'] == control_store),
    'scaledControlSales'
] = measure_over_time_sales['totSales'] * scaling_factor

# Filter for the trial and control stores only
filtered_sales = measure_over_time_sales[
    measure_over_time_sales['STORE_NBR'].isin([trial_store, control_store])
]

# Check the structure of the filtered_sales DataFrame
print(filtered_sales.head())

# Create a pivot table
pre_trial_percentage_diff = filtered_sales[
    (filtered_sales['YEARMONTH'] <= pre_trial_period_end)
].pivot_table(
    index='YEARMONTH',
    columns='STORE_NBR',
    values=['totSales', 'scaledControlSales'],
    aggfunc='sum'
).reset_index()

# Flatten the multi-level columns
pre_trial_percentage_diff.columns = ['YEARMONTH'] + [f"{col[0]}_{col[1]}" for col in p

# Check the structure of the pivot table
print(pre_trial_percentage_diff.head())

# Rename columns for clarity
pre_trial_percentage_diff.rename(columns={
    'totSales_88': 'trialSales',
```

```
    'scaledControlSales_237': 'scaledControlSales'
}, inplace=True)

# Calculate the percentage difference for the pre-trial period
pre_trial_percentage_diff['percentageDiff'] = (
    (pre_trial_percentage_diff['trialSales'] - pre_trial_percentage_diff['scaledContro
    pre_trial_percentage_diff['scaledControlSales']
) * 100

# Calculate standard deviation of pre-trial percentage differences
std_dev = pre_trial_percentage_diff['percentageDiff'].std()
degrees_of_freedom = len(pre_trial_percentage_diff) - 1

# Merge the trial and scaled control sales data for the trial period
trial_percentage_diff = filtered_sales[
    (filtered_sales['YEARMONTH'] >= trial_period_start) &
    (filtered_sales['YEARMONTH'] <= trial_period_end)
].pivot_table(
    index='YEARMONTH',
    columns='STORE_NBR',
    values=['totSales', 'scaledControlSales'],
    aggfunc='sum'
).reset_index()

# Flatten the multi-level columns
trial_percentage_diff.columns = ['YEARMONTH'] + [f"{col[0]}_{col[1]}" for col in trial

# Check the structure of the pivot table for the trial period
print(trial_percentage_diff.head())

# Rename columns for clarity
trial_percentage_diff.rename(columns={
    'totSales_88': 'trialSales',
    'scaledControlSales_237': 'scaledControlSales'
}, inplace=True)

# Calculate the percentage difference for the trial period
trial_percentage_diff['percentageDiff'] = (
    (trial_percentage_diff['trialSales'] - trial_percentage_diff['scaledControlSales']
    trial_percentage_diff['scaledControlSales']
) * 100

# Calculate t-values for the trial months
trial_percentage_diff['tValue'] = (
    trial_percentage_diff['percentageDiff'] / std_dev
)

# Add transaction month for clarity
trial_percentage_diff['TransactionMonth'] = trial_percentage_diff['YEARMONTH']

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

# Print results
print("Standard Deviation of Pre-trial Percentage Differences:", std_dev)
print("Degrees of Freedom:", degrees_of_freedom)
print("t Critical Value (95th percentile):", t_critical)
print("Percentage Differences and t-Values for Trial Months:")
print(trial_percentage_diff[['TransactionMonth', 'percentageDiff', 'tValue']])
```

```
      STORE_NBR YEARMONTH  totSales  nCustomers  nTxnPerCust  nChipsPerTxn  \
1001         88    201807    1310.0         129          153       2.00000
1002         88    201808    1323.8         131          160       1.89375
1003         88    201809    1423.0         124          159       2.00000
1004         88    201810    1352.4         123          158       2.00000
1005         88    201811    1382.8         130          157       2.00000

      avgPricePerUnit  scaledControlSales  scaledControlCustomers
1001         8.562092                 NaN                     NaN
1002         8.273750                 NaN                     NaN
1003         8.949686                 NaN                     NaN
1004         8.559494                 NaN                     NaN
1005         8.807643                 NaN                     NaN
  YEARMONTH  scaledControlSales_88  scaledControlSales_237  totSales_88  \
0    201807                   0.0             1450.657086       1310.0
1    201808                   0.0             1369.931485       1323.8
2    201809                   0.0             1324.260425       1423.0
3    201810                   0.0             1350.401097       1352.4
4    201811                   0.0             1399.777923       1382.8

   totSales_237
0        1448.4
1        1367.8
2        1322.2
3        1348.3
4        1397.6
  YEARMONTH  scaledControlSales_88  scaledControlSales_237  totSales_88  \
0    201902                   0.0             1406.989143       1370.2
1    201903                   0.0             1210.082775       1477.2
2    201904                   0.0             1206.477165       1439.4

   totSales_237
0        1404.8
1        1208.2
2        1204.6
Standard Deviation of Pre-trial Percentage Differences: 5.724965451900226
Degrees of Freedom: 6
t Critical Value (95th percentile): 1.9431802803927816
Percentage Differences and t-Values for Trial Months:
  TransactionMonth  percentageDiff    tValue
0           201902       -2.614742 -0.456726
1           201903       22.074294  3.855795
2           201904       19.306029  3.372253
```

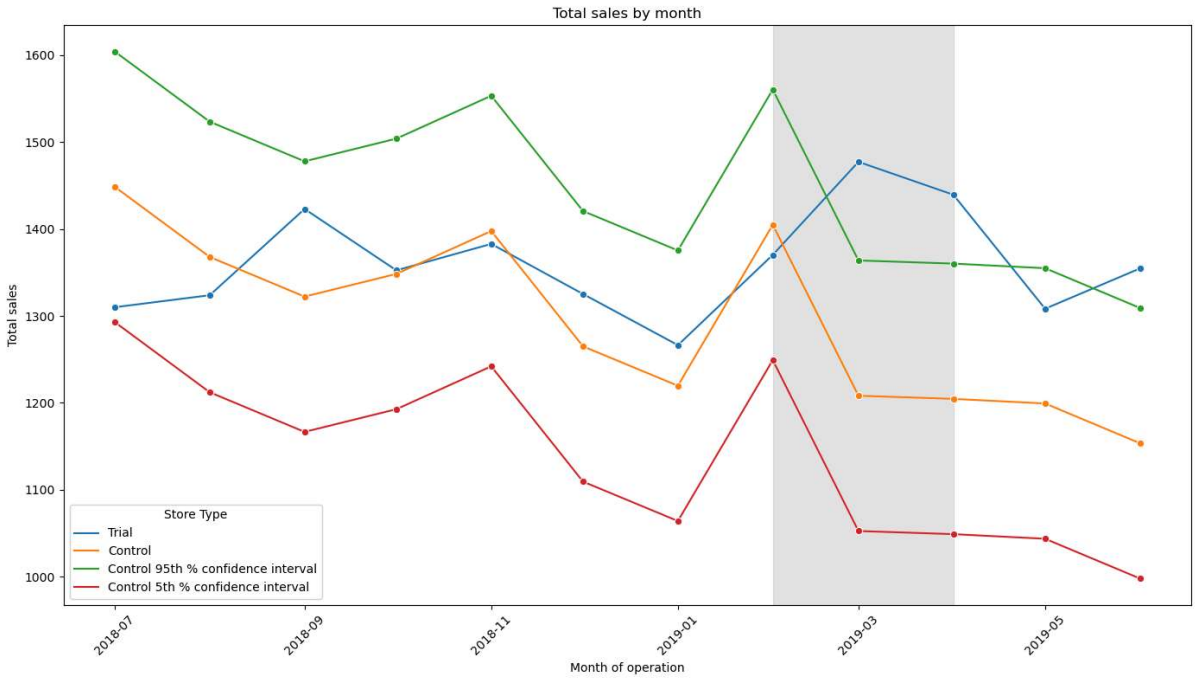
In [33]:

```
# Assuming monthly_metrics is already defined as your main DataFrame
# Convert YEARMONTH to integer if it's not already
monthly_metrics['YEARMONTH'] = monthly_metrics['YEARMONTH'].astype(int)

# Define trial and control stores
trial_store = 88
control_store = 237

# Define the pre-trial and trial periods
pre_trial_period_end = 201901
trial_period_start = 201902
trial_period_end = 201904

# Calculate the total sales in the pre-trial period for both trial and control stores
pre_trial_sales_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

pre_trial_sales_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].sum()

# Calculate the scaling factor
scaling_factor = pre_trial_sales_trial / pre_trial_sales_control

# Print scaling factor for debugging
print(f"Scaling factor: {scaling_factor}")

# Apply the scaling factor to the control store sales during the pre-trial and trial pe
monthly_metrics.loc[
    (monthly_metrics['STORE_NBR'] == control_store),
    'scaledControlSales'
] = monthly_metrics['totSales'] * scaling_factor

# Filter for the trial and control stores only
filtered_sales = monthly_metrics[
    monthly_metrics['STORE_NBR'].isin([trial_store, control_store])
]

# Create a new column for store type
filtered_sales['Store_type'] = np.where(
    filtered_sales['STORE_NBR'] == trial_store, 'Trial', 'Control'
)

# Create a new column for TransactionMonth
filtered_sales['TransactionMonth'] = pd.to_datetime(filtered_sales['YEARMONTH'].astype

# Calculate the 95th and 5th percentiles for the control store
std_dev = filtered_sales[
    (filtered_sales['Store_type'] == 'Control') &
    (filtered_sales['YEARMONTH'] <= pre_trial_period_end)
]['totSales'].std()

# Print standard deviation for debugging
print(f"Standard Deviation: {std_dev}")

filtered_sales_95 = filtered_sales[filtered_sales['Store_type'] == 'Control'].copy()
filtered_sales_95['totSales'] = filtered_sales_95['totSales'] + (std_dev * 2)
```

```
filtered_sales_95['Store_type'] = 'Control 95th % confidence interval'

filtered_sales_5 = filtered_sales[filtered_sales['Store_type'] == 'Control'].copy()
filtered_sales_5['totSales'] = filtered_sales_5['totSales'] - (std_dev * 2)
filtered_sales_5['Store_type'] = 'Control 5th % confidence interval'

# Combine all the data
trial_assessment = pd.concat([filtered_sales, filtered_sales_95, filtered_sales_5])

# Print the first few rows of the combined data for debugging
print(trial_assessment.head())

# Plotting
plt.figure(figsize=(14, 8))
sns.lineplot(data=trial_assessment, x='TransactionMonth', y='totSales', hue='Store_typ

# Highlight the trial period
plt.axvspan(pd.to_datetime('201902', format='%Y%m'), pd.to_datetime('201904', format='

# Labels and title
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Total sales by month')
plt.legend(title='Store Type')
plt.xticks(rotation=45)
plt.tight_layout()

# Show plot
plt.show()
```

```
Scaling factor: 1.001558330664959
Standard Deviation: 77.78891768295607
      STORE_NBR  YEARMONTH  totSales  nCustomers  nTxnPerCust  nChipsPerTxn  \
1001         88     201807    1310.0         129          153       2.00000
1002         88     201808    1323.8         131          160       1.89375
1003         88     201809    1423.0         124          159       2.00000
1004         88     201810    1352.4         123          158       2.00000
1005         88     201811    1382.8         130          157       2.00000

      avgPricePerUnit  scaledControlSales  scaledControlCustomers Store_type  \
1001         8.562092                 NaN                     NaN      Trial
1002         8.273750                 NaN                     NaN      Trial
1003         8.949686                 NaN                     NaN      Trial
1004         8.559494                 NaN                     NaN      Trial
1005         8.807643                 NaN                     NaN      Trial

     TransactionMonth
1001       2018-07-01
1002       2018-08-01
1003       2018-09-01
1004       2018-10-01
1005       2018-11-01
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\1041189794.py:48: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_sales['Store_type'] = np.where(
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\1041189794.py:53: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_sales['TransactionMonth'] = pd.to_datetime(filtered_sales['YEARMONTH'].ast
ype(str), format='%Y%m')
C:\Users\Tonye\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Tonye\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



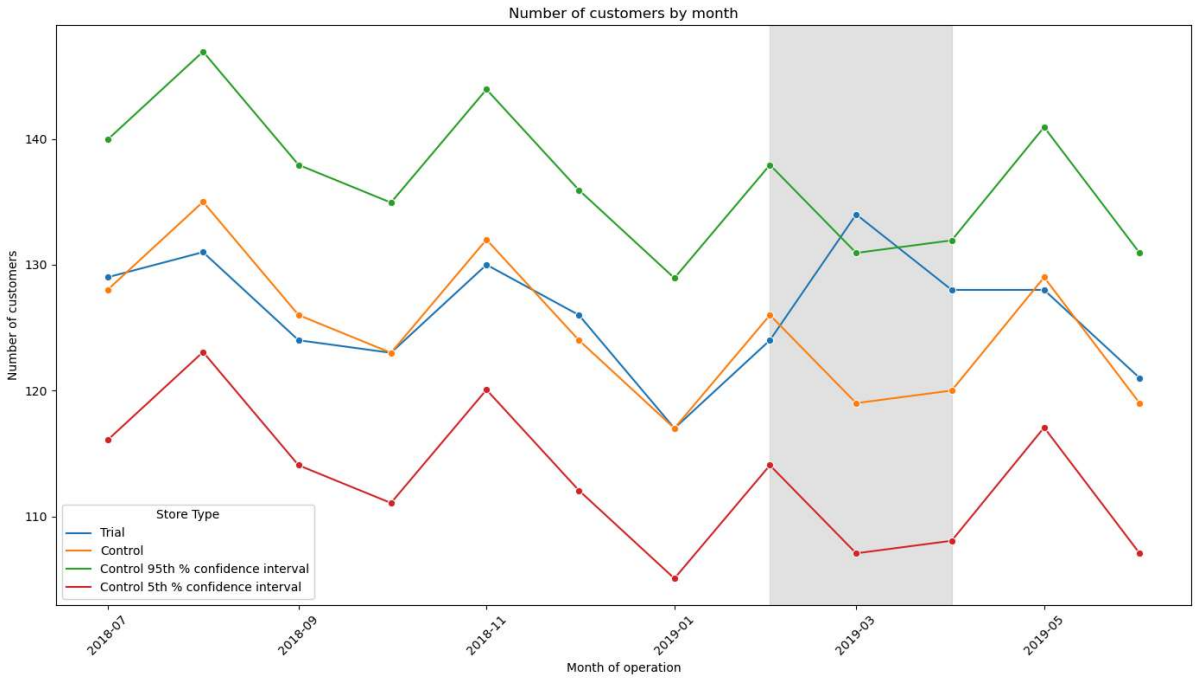
In [34]:

```
# Calculate the total number of customers in the pre-trial period for both trial and c
pre_trial_customers_trial = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == trial_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['nCustomers'].sum()

pre_trial_customers_control = monthly_metrics[
    (monthly_metrics['STORE_NBR'] == control_store) &
    (monthly_metrics['YEARMONTH'] <= pre_trial_period_end)
]['nCustomers'].sum()

# Calculate the scaling factor
scaling_factor_customers = pre_trial_customers_trial / pre_trial_customers_control

# Print scaling factor for debugging
print(f"Scaling factor for customers: {scaling_factor_customers}")

# Apply the scaling factor to the control store customers during the pre-trial and tri
monthly_metrics.loc[
    (monthly_metrics['STORE_NBR'] == control_store),
    'scaledControlCustomers'
] = monthly_metrics['nCustomers'] * scaling_factor_customers

# Filter for the trial and control stores only
filtered_customers = monthly_metrics[
    monthly_metrics['STORE_NBR'].isin([trial_store, control_store])
]

# Create a new column for store type
filtered_customers['Store_type'] = np.where(
    filtered_customers['STORE_NBR'] == trial_store, 'Trial', 'Control'
)

# Create a new column for TransactionMonth
filtered_customers['TransactionMonth'] = pd.to_datetime(filtered_customers['YEARMONTH'

# Calculate the 95th and 5th percentiles for the control store
std_dev_customers = filtered_customers[
    (filtered_customers['Store_type'] == 'Control') &
    (filtered_customers['YEARMONTH'] <= pre_trial_period_end)
]['nCustomers'].std()

# Print standard deviation for debugging
print(f"Standard Deviation for customers: {std_dev_customers}")

filtered_customers_95 = filtered_customers[filtered_customers['Store_type'] == 'Contro
filtered_customers_95['nCustomers'] = filtered_customers_95['nCustomers'] + (std_dev_c
filtered_customers_95['Store_type'] = 'Control 95th % confidence interval'

filtered_customers_5 = filtered_customers[filtered_customers['Store_type'] == 'Control
filtered_customers_5['nCustomers'] = filtered_customers_5['nCustomers'] - (std_dev_cus
filtered_customers_5['Store_type'] = 'Control 5th % confidence interval'

# Combine all the data
trial_assessment_customers = pd.concat([filtered_customers, filtered_customers_95, fil

# Print the first few rows of the combined data for debugging
print(trial_assessment_customers.head())

# Plotting
plt.figure(figsize=(14, 8))
sns.lineplot(data=trial_assessment_customers, x='TransactionMonth', y='nCustomers', hu
```

```
# Highlight the trial period
plt.axvspan(pd.to_datetime('201902', format='%Y%m'), pd.to_datetime('201904', format='%

# Labels and title
plt.xlabel('Month of operation')
plt.ylabel('Number of customers')
plt.title('Number of customers by month')
plt.legend(title='Store Type')
plt.xticks(rotation=45)
plt.tight_layout()

# Show plot
plt.show()
```

```
Scaling factor for customers: 0.9943502824858758
Standard Deviation for customers: 5.968169536721256
      STORE_NBR  YEARMONTH  totSales  nCustomers  nTxnPerCust  nChipsPerTxn  \
1001         88     201807    1310.0       129.0          153       2.00000
1002         88     201808    1323.8       131.0          160       1.89375
1003         88     201809    1423.0       124.0          159       2.00000
1004         88     201810    1352.4       123.0          158       2.00000
1005         88     201811    1382.8       130.0          157       2.00000

      avgPricePerUnit  scaledControlSales  scaledControlCustomers Store_type  \
1001         8.562092                 NaN                     NaN      Trial
1002         8.273750                 NaN                     NaN      Trial
1003         8.949686                 NaN                     NaN      Trial
1004         8.559494                 NaN                     NaN      Trial
1005         8.807643                 NaN                     NaN      Trial

     TransactionMonth
1001       2018-07-01
1002       2018-08-01
1003       2018-09-01
1004       2018-10-01
1005       2018-11-01
```

```
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\3103596605.py:30: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_customers['Store_type'] = np.where(
C:\Users\Tonye\AppData\Local\Temp\ipykernel_15652\3103596605.py:35: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  filtered_customers['TransactionMonth'] = pd.to_datetime(filtered_customers['YEARMON
TH'].astype(str), format='%Y%m')
C:\Users\Tonye\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Tonye\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [ ]:

In [ ]: