Load required libraries and datasets

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
```

In [2]:
```
# Load dataset
df = pd.read_csv(r'resources/QVI_data.csv')
```

In [3]:
```
# Print first few rows of the dataframe
df.head()
```

Out[3]:

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

In [4]:
```
# Create YEARMONTH column and change the date format to YYYYMM
df['DATE'] = pd.to_datetime(df['DATE'])
df['YEARMONTH'] = df['DATE'].dt.strftime('%Y%m')
```

Select control stores

The client has selected store numbers 77, 86 and 88 as trial stores and want control stores to be established stores that are operational for the entire observation period.

We would want to match trial stores to control stores that are similar to the trial store prior to the trial period of Feb 2019 in terms of :

- Monthly overall sales revenue
- Monthly number of customers
- Monthly number of transactions per customer

Let's first create the metrics of interest and filter to stores that are present throughout the pre-trial and trial period

In [5]:
```
# Group by 'STORE_NBR' and 'YEARMONTH' and calculate the metrics
measure_overtime = df.groupby(['STORE_NBR', 'YEARMONTH']).agg(
    tot_sales=pd.NamedAgg(column='TOT_SALES', aggfunc='sum'),
    n_customers=pd.NamedAgg(column='LYLTY_CARD_NBR', aggfunc='nunique'),
    n_txn_per_cust=pd.NamedAgg(column='TXN_ID', aggfunc='count'),
    n_chips_per_txn=pd.NamedAgg(column='PROD_QTY', aggfunc='sum'),
    avg_price_per_unit=pd.NamedAgg(column='TOT_SALES', aggfunc='mean')
).reset_index()
```

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

In [6]:
```
# Calculate 'n_txn_per_cust' and 'n_chips_per_txn' per customer
measure_overtime['n_txn_per_cust'] /= measure_overtime['n_customers'].round(5)
measure_overtime['n_chips_per_txn'] /= measure_overtime['n_customers'].round(5)

# Create a boolean mask for stores with full observation periods
mask = measure_overtime.groupby('STORE_NBR')['STORE_NBR'].transform('size') == 12

# Identify stores with full observation periods
stores_with_full_obs = measure_overtime.loc[mask, 'STORE_NBR'].unique()

# Filter to overtime period and stores with full observation periods
overtime_period = measure_overtime['YEARMONTH'] >= '201807'

# Filter to the pre-trial period and stores with full observation periods
pre_trial_period = measure_overtime['YEARMONTH'] < '201902'
pre_trial_measures = measure_overtime[pre_trial_period & (measure_overtime['STORE_NBR'].isin(stores_with

# Filter to the trial period and stores with full observation periods
trial_period = (measure_overtime['YEARMONTH'] >= '201902')
trial_measures = measure_overtime[trial_period & (measure_overtime['STORE_NBR'].isin(stores_with_full_ob
```

In [7]:
```
measure_overtime
```

Out[7]:

	STORE_NBR	YEARMONTH	tot_sales	n_customers	n_txn_per_cust	n_chips_per_txn	avg_price_per_unit
0	1	201807	206.9	49	1.061224	1.265306	3.978846
1	1	201808	176.1	42	1.023810	1.285714	4.095349
2	1	201809	278.8	59	1.050847	1.271186	4.496774
3	1	201810	188.1	44	1.022727	1.318182	4.180000
4	1	201811	192.6	46	1.021739	1.239130	4.097872
...	...	...	...	...	...	...	...
3164	272	201902	395.5	45	1.066667	2.022222	8.239583
3165	272	201903	442.3	50	1.060000	2.020000	8.345283
3166	272	201904	445.1	54	1.037037	1.944444	7.948214
3167	272	201905	314.6	34	1.176471	2.088235	7.865000
3168	272	201906	312.1	34	1.088235	2.058824	8.435135

3169 rows × 7 columns

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

In [8]:
```
# Create correlation function
def calculate_correlation(input_table, metric_col, store_comparison):
    # Initialize an empty list to store the results
    calc_corr_table = []

    # Get the unique store numbers
    store_numbers = input_table['STORE_NBR'].unique()

    # Loop through each store number
    for i in store_numbers:
        # Calculate the correlation measure
        corr_measure = np.corrcoef(
            input_table.loc[input_table['STORE_NBR'] == store_comparison, metric_col],
            input_table.loc[input_table['STORE_NBR'] == i, metric_col]
        )[0, 1]

        # Append the result to the list
        calc_corr_table.append({
            'Store1': store_comparison,
            'Store2': i,
            'corr_measure': corr_measure
        })
```

```
    # Convert the list to a DataFrame
    calc_corr_table = pd.DataFrame(calc_corr_table)

    return calc_corr_table
```

Apart from correlation, we can also calculate a standardised metric based on the absolute difference between the trial store's performance and each control store's performance.

In [9]:
```
def calculate_magnitude_distance(input_table, metric_col, store_comparison):
    # Initialize an empty list to store the results
    calc_dist_table = []

    # Get the unique store numbers
    store_numbers = input_table['STORE_NBR'].unique()

    # Loop through each store number
    for i in store_numbers:
        # Calculate the absolute difference measure for each month
        for month in input_table['YEARMONTH'].unique():
            measure = abs(
                input_table.loc[(input_table['STORE_NBR'] == i) & (input_table['YEARMONTH'] == month), m
                - input_table.loc[(input_table['STORE_NBR'] == store_comparison) & (input_table['YEARMON
            )

            # Append the result to the list
            calc_dist_table.append({
                'Store1': store_comparison,
                'Store2': i,
                'YEARMONTH': month,
                'measure': measure
            })

    # Convert the list to a DataFrame
    calc_dist_table = pd.DataFrame(calc_dist_table)

    # Calculate the min and max distance for each 'Store1' and 'YEARMONTH'
    min_max_dist = calc_dist_table.groupby(['Store1', 'YEARMONTH'])['measure'].agg(['min', 'max']).reset

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

    # Calculate the magnitude measure
    dist_table['magnitude_measure'] = 1 - (dist_table['measure'] - dist_table['min']) / (dist_table['max

    # Calculate the mean magnitude measure for each 'Store1' and 'Store2'
    final_dist_table = dist_table.groupby(['Store1', 'Store2'])['magnitude_measure'].mean().reset_index(

    return final_dist_table
```

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

Trial store 77

In [10]:
```
# Define the trial store
trial_store = 77

# Calculate correlations for total sales and number of customers
corr_n_sales = calculate_correlation(pre_trial_measures, 'tot_sales', trial_store)
corr_n_customers = calculate_correlation(pre_trial_measures, 'n_customers', trial_store)

# Calculate magnitude distances for total sales and number of customers
magnitude_n_sales = calculate_magnitude_distance(pre_trial_measures, 'tot_sales', trial_store)
magnitude_n_customers = calculate_magnitude_distance(pre_trial_measures, 'n_customers', trial_store)
```

We'll need to combine the all the scores calculated using our function to create a composite score to rank on. Let's take a simple average of the correlation and magnitude scores for each driver. Note that if we consider it more important for the trend of the drivers to be similar, we can increase the weight of the correlation score (a simple average gives a weight of 0.5 to the corr_weight) or if we consider the absolute size of the drivers to be more important, we can lower the weight of the correlation score.

In [11]:
```
# Merge the correlation and magnitude distance dataframes
score_n_sales = pd.merge(corr_n_sales, magnitude_n_sales, on=['Store1', 'Store2'])
score_n_customers = pd.merge(corr_n_customers, magnitude_n_customers, on=['Store1', 'Store2'])

# Define the correlation weight
corr_weight = 0.5

# Calculate the composite scores
score_n_sales['score_n_sales'] = corr_weight * score_n_sales['corr_measure'] + (1 - corr_weight) * score
score_n_customers['score_n_cust'] = corr_weight * score_n_customers['corr_measure'] + (1 - corr_weight)
```

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

In [12]:
```
# Merge the sales and customer scores
control_store = pd.merge(score_n_sales, score_n_customers, on=['Store1', 'Store2'])

# Calculate the final control score
control_store['final_control_store'] = 0.5 * control_store['score_n_sales'] + 0.5 * control_store['score

# Exclude the trial stores from consideration
filtered_control_store = control_store[control_store['Store2'] != trial_store]

# Sort the stores based on the final control score in descending order
sorted_control_store = filtered_control_store.sort_values(by='final_control_store', ascending=False)
```

Select control stores based on the highest matching store (closest to 1 but not the store itself, i.e. the second ranked highest store)

In [13]:
```
# Use the selected control store
control_store = int(sorted_control_store.iloc[0]['Store2'])

# Print the trial and control store
print(f"Trial store = {trial_store}")
print(f"Control store = {control_store}")
```

```
Trial store = 77
Control store = 233
```

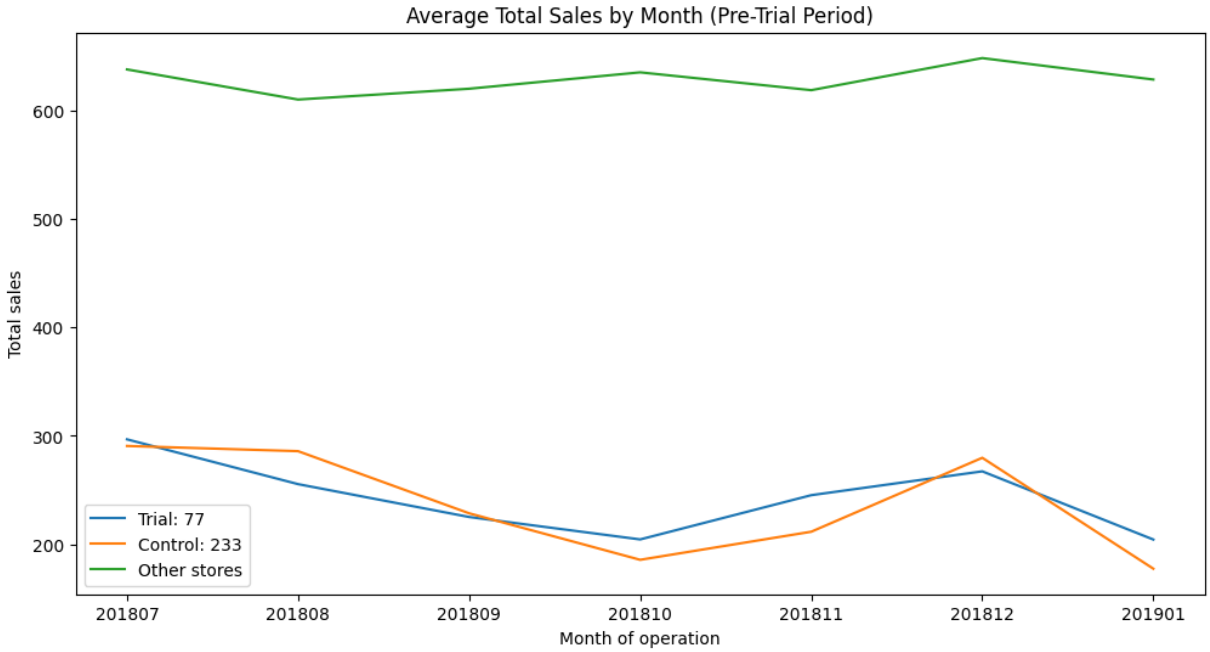
Now that we have found a control store for trial store 77 which is store 233, let's check visually if the drivers are indeed similar in the period before the trial

In [14]:
```
# Create a new column 'store_type' to label the trial and control stores
measure_overtime['store_type'] = measure_overtime['STORE_NBR'].apply(lambda x: 'Trial' if x == trial_sto

# Filter to the pre-trial period and stores with full observation periods
pre_trial_period = measure_overtime['YEARMONTH'] < '201902'
pre_trial_measures = measure_overtime[pre_trial_period & (measure_overtime['STORE_NBR'].isin(stores_with

# Filter to the trial period and stores with full observation periods
trial_period = (measure_overtime['YEARMONTH'] >= '201902')
trial_measures = measure_overtime[trial_period & (measure_overtime['STORE_NBR'].isin(stores_with_full_ob
```

We'll look at total sales first. Visual checks on trends based on the drivers

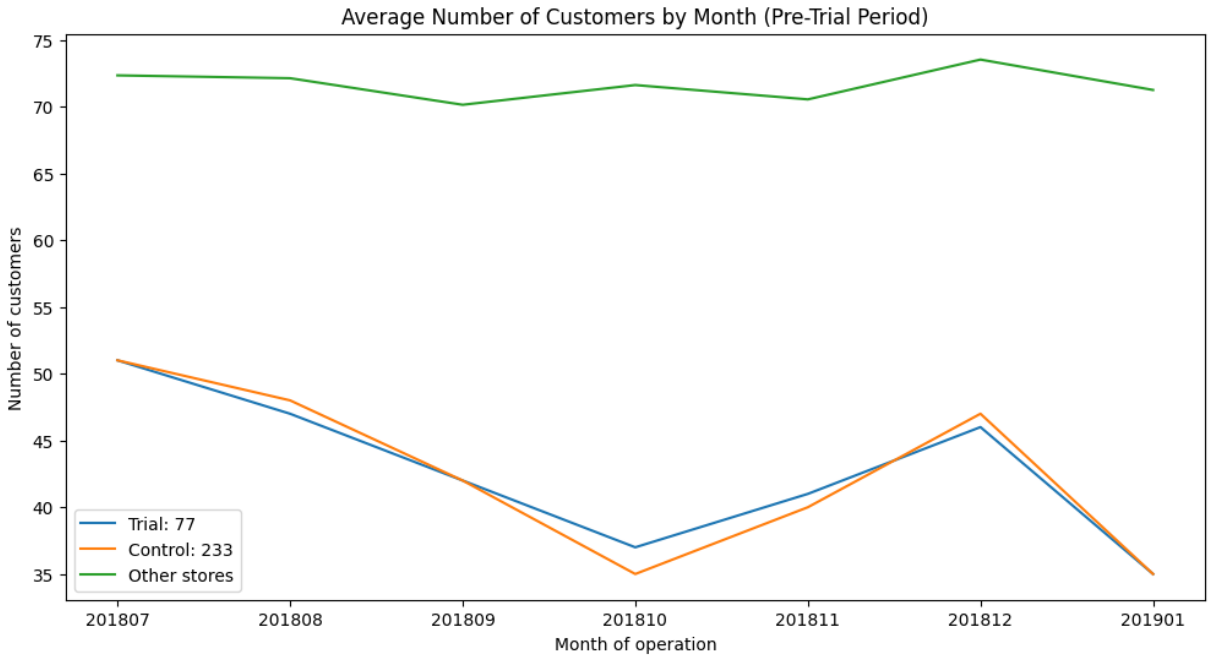
In [15]:
```
# Calculate the mean total sales for each 'YEARMONTH' and 'store_type'
past_sales = pre_trial_measures.groupby(['YEARMONTH', 'store_type'])['tot_sales'].mean().reset_index()

# Plot the total sales
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Other stores']:
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
```

```
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(past_sales.loc[past_sales['store_type'] == store_type, 'YEARMONTH'],
             past_sales.loc[past_sales['store_type'] == store_type, 'tot_sales'],
             label=store_label)

plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Average Total Sales by Month (Pre-Trial Period)')
plt.legend()
plt.show()
```



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

In [16]:
```
# Calculate the mean number of customers for each 'YEARMONTH' and 'store_type'
past_customers = pre_trial_measures.groupby(['YEARMONTH', 'store_type'])['n_customers'].mean().reset_ind

# Plot the number of customers
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Other stores']:
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(past_customers.loc[past_customers['store_type'] == store_type, 'YEARMONTH'],
             past_customers.loc[past_customers['store_type'] == store_type, 'n_customers'], label=store_
plt.xlabel('Month of operation')
plt.ylabel('Number of customers')
plt.title('Average Number of Customers by Month (Pre-Trial Period)')
plt.legend()
plt.show()
```



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

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

In [17]:
```
def scale_control():

    # Copy measure_overtime dataframe
    scaled_control = measure_overtime.copy()

    # Dictionary to store scaling factors
    scaling_factors = {}

    # Calculate scaling factors based on pre-trial period
    scaling_factors_sales = pre_trial_measures.loc[(pre_trial_measures['store_type'] == 'Trial'), 'tot_s
    scaling_factors_customers = pre_trial_measures.loc[(pre_trial_measures['store_type'] == 'Trial'), 'n_

    # Apply the scaling factors
    scaled_control.loc[scaled_control['store_type'] == 'Control', 'control_sales'] = scaled_control.loc[
    scaled_control.loc[scaled_control['store_type'] == 'Control', 'control_customers'] = scaled_control.

    return scaled_control
```

In [18]:
```
# Apply scale_control function and assign to scaled_control variable
scaled_control = scale_control()
```

In [19]:
```
def calculate_percentage_diff(type):

    # Merge the trial store's sales with the control store's scaled sales
    percentage_diff = pd.merge(
        scaled_control.loc[scaled_control['store_type'] == 'Trial', ['YEARMONTH', 'tot_sales', 'n_custom
        scaled_control.loc[scaled_control['store_type'] == 'Control', ['YEARMONTH', 'control_sales', 'co
    on='YEARMONTH'
    )

    if type == 'sales':
        # Calculate the percentage difference
        percentage_diff['percentage_diff_sales'] = (percentage_diff['tot_sales'] - percentage_diff['cont

    if type == 'customers':
        # Calculate the percentage difference
        percentage_diff['percentage_diff_customers'] = (percentage_diff['n_customers'] - percentage_diff

    return percentage_diff
```

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.

In [20]:
```
# Apply calculate_percentage_diff function for trial store 77 sales
percentage_diff_sales = calculate_percentage_diff('sales')
```

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

In [21]:
```
# Filter percentage_diff to the pre_trial_period
filtered_data = percentage_diff_sales.loc[pre_trial_period].copy()

# Calculate the standard deviation of the percentage difference in the pre-trial period
std_dev = filtered_data['percentage_diff_sales'].std()

# Filter percentage_diff to the trial_period
filtered_data = percentage_diff_sales.loc[trial_period].copy()

# Round the 'control_sales' values to one decimal place
filtered_data['control_sales'] = filtered_data['control_sales'].round(1)

# Calculate the t-values for the trial months
sample_size = len(filtered_data)
filtered_data['t_value'] = (filtered_data['percentage_diff_sales'] - 0) / (std_dev / (sample_size ** 0.5

# Degrees of freedom (df) for the t-distribution. Note that there are 8 months in the pre-trial period
degrees_of_freedom = 8 - 1

# Probability value corresponding to the 95th percentile
percentile_value = 1 - 0.05

# Calculate the 95th percentile of the t-distribution
t_95_percentile = stats.t.ppf(percentile_value, df=degrees_of_freedom)

# Add 't_95_percentile' to the percentage_diff dataframe
filtered_data['t_95_percentile'] = t_95_percentile
filtered_data
```

Out[21]:

	YEARMONTH	tot_sales	n_customers	control_sales	control_customers	percentage_diff_sales	t_value	t_95_
7	201902	235.0	45	249.8	45.151007	-0.059107	-1.327152	
8	201903	278.5	50	203.8	40.134228	0.366521	8.229692	
9	201904	263.5	47	162.3	30.100671	0.623080	13.990338	
10	201905	299.3	55	352.5	57.191275	-0.151003	-3.390559	
11	201906	264.7	41	226.2	41.137584	0.170103	3.819411	

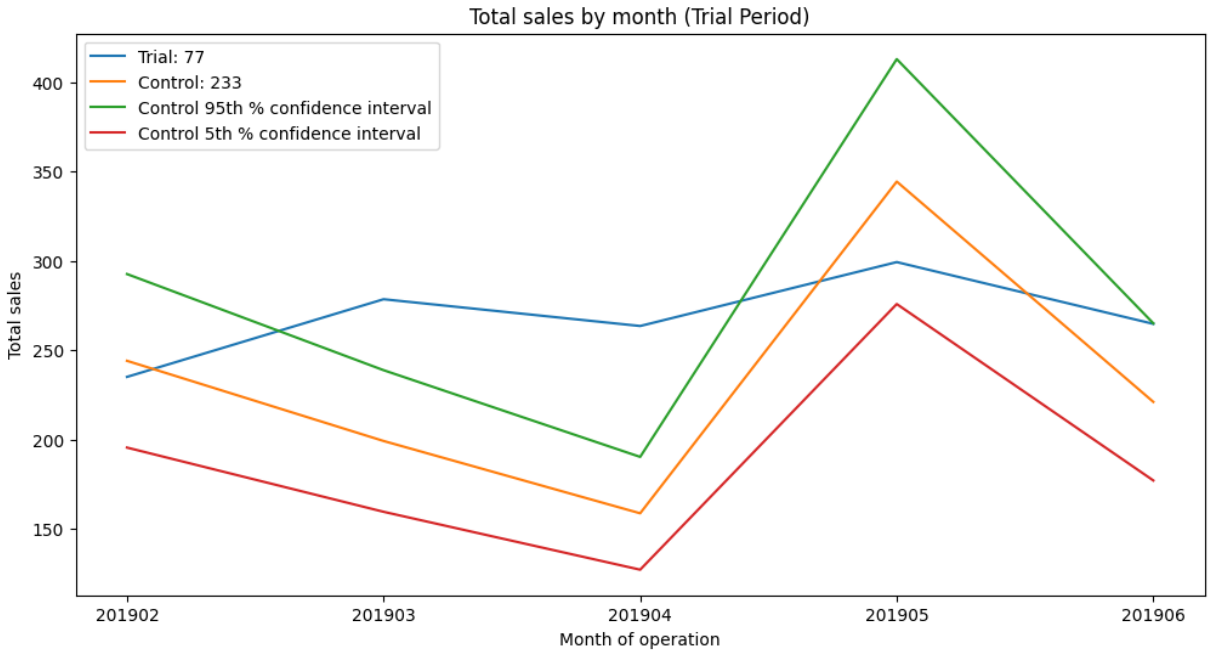
We can observe that the t-value is much larger than the 95th percentile value of the t-distribution for March and April - i.e. the increase in sales in the trial store in March and April is statistically greater than in the control store

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

In [22]:
```
# Calculate the mean total sales for each 'YEARMONTH' and 'store_type'
trial_sales = trial_measures.groupby(['YEARMONTH', 'store_type'])['tot_sales'].mean().reset_index()

# Calculate the control store 95th percentile
controls_sales_95 = trial_sales.loc[trial_sales['store_type'] == 'Control'].copy()
controls_sales_95['tot_sales'] *= (1 + std_dev * 2)
controls_sales_95['store_type'] = 'Control 95th % confidence interval'

# Calculate the control store 5th percentile
controls_sales_5 = trial_sales.loc[trial_sales['store_type'] == 'Control'].copy()
controls_sales_5['tot_sales'] *= (1 - std_dev * 2)
controls_sales_5['store_type'] = 'Control 5th % confidence interval'
```

```
# Combine the DataFrames
trial_assessment = pd.concat([trial_sales, controls_sales_95, controls_sales_5])
```

In [23]:
```
# Plot the total sales
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Control 95th % confidence interval', 'Control 5th % confidence i
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(trial_assessment.loc[trial_assessment['store_type'] == store_type, 'YEARMONTH'],
             trial_assessment.loc[trial_assessment['store_type'] == store_type, 'tot_sales'], label=stor
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Total sales by month (Trial Period)')
plt.legend()
plt.show()
```



The results show that the trial in store 77 is significantly different to its control store in the trial period as the trial store performance lies outside the 5% to 95% confidence interval of the control store in two of the three trial months.

Let's have a look at assessing this for number of customers as well

In [24]:
```
# Apply calculate_percentage_diff function for trial store 77 customers
percentage_diff_customers = calculate_percentage_diff('customers')
```

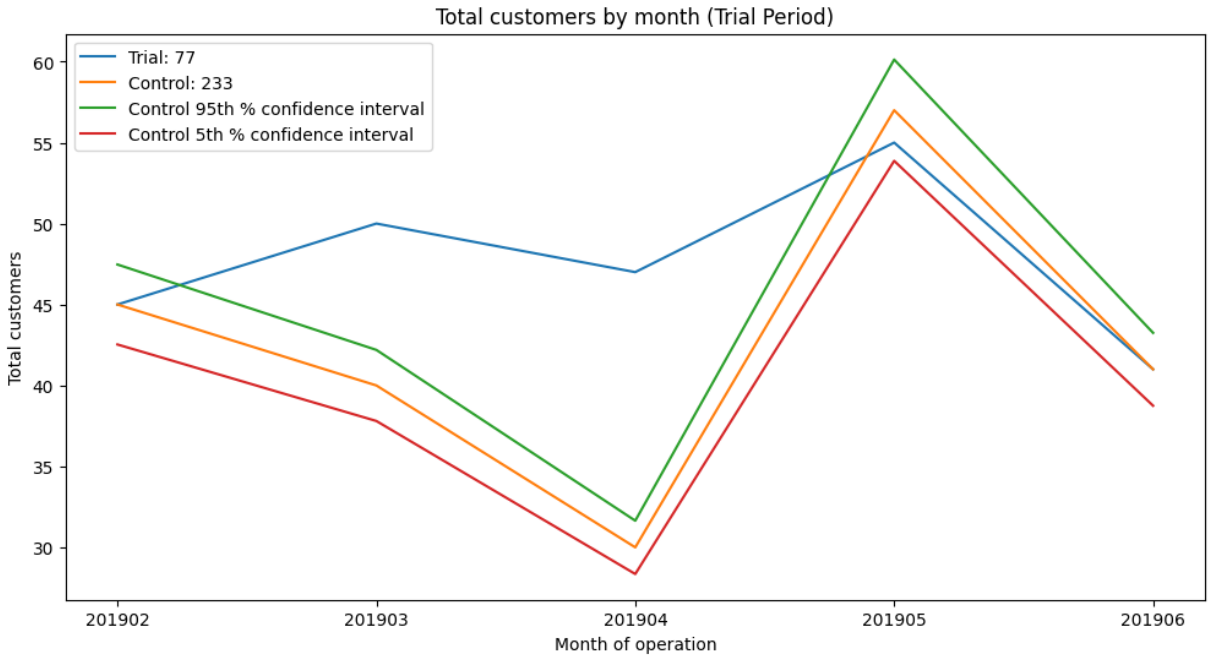
Let's again see if the difference is significant visually!. As our null hypothesis is that the trial period is the same as the pre-trial period, let's take the standard deviation based on the scaled percentage difference in the pre-trial period

In [25]:
```
# Filter percentage_diff to the pre_trial_period
filtered_data = percentage_diff_customers.loc[pre_trial_period].copy()

# Calculate the standard deviation of the percentage difference in the pre-trial period
std_dev = filtered_data['percentage_diff_customers'].std()

# Calculate the t-values for the trial months
sample_size = len(filtered_data)
filtered_data['t_value'] = (filtered_data['percentage_diff_customers'] - 0) / (std_dev / (sample_size **

# Degrees of freedom (df) for the t-distribution. Note that there are 8 months in the pre-trial period
degrees_of_freedom = 8 - 1

# Probability value corresponding to the 95th percentile
percentile_value = 1 - 0.05
```

```
# Calculate the 95th percentile of the t-distribution
t_95_percentile = stats.t.ppf(percentile_value, df=degrees_of_freedom)

# Add 't_95_percentile' to the percentage_diff dataframe
filtered_data['t_95_percentile'] = t_95_percentile
```

In [26]:
```
# Create a more visual version of this by plotting the sales of the control store,
# the sales of the trial stores and the 95th percentile value of sales of the control store.

# Calculate the mean total sales for each 'YEARMONTH' and 'store_type'
trial_customers = trial_measures.groupby(['YEARMONTH', 'store_type'])['n_customers'].mean().reset_index(

# Calculate the control store 95th percentile
controls_customers_95 = trial_customers.loc[trial_customers['store_type'] == 'Control'].copy()
controls_customers_95['n_customers'] *= (1 + std_dev * 2)
controls_customers_95['store_type'] = 'Control 95th % confidence interval'

# Calculate the control store 5th percentile
controls_customers_5 = trial_customers.loc[trial_customers['store_type'] == 'Control'].copy()
controls_customers_5['n_customers'] *= (1 - std_dev * 2)
controls_customers_5['store_type'] = 'Control 5th % confidence interval'

# Combine the DataFrames
trial_assessment = pd.concat([trial_customers, controls_customers_95, controls_customers_5])
```

In [27]:
```
# Plot the total sales
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Control 95th % confidence interval', 'Control 5th % confidence i
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(trial_assessment.loc[trial_assessment['store_type'] == store_type, 'YEARMONTH'],
             trial_assessment.loc[trial_assessment['store_type'] == store_type, 'n_customers'], label=st
plt.xlabel('Month of operation')
plt.ylabel('Total customers')
plt.title('Total customers by month (Trial Period)')
plt.legend()
plt.show()
```



Let's repeat finding the control store and assessing the impact of the trial for each of the other two trial stores.

Trial store 86

In [28]:
```
# Define the trial store
trial_store = 86

# Calculate correlations for total sales and number of customers
corr_n_sales = calculate_correlation(pre_trial_measures, 'tot_sales', trial_store)
corr_n_customers = calculate_correlation(pre_trial_measures, 'n_customers', trial_store)

# Calculate magnitude distances for total sales and number of customers
magnitude_n_sales = calculate_magnitude_distance(pre_trial_measures, 'tot_sales', trial_store)
magnitude_n_customers = calculate_magnitude_distance(pre_trial_measures, 'n_customers', trial_store)
```

We'll need to combine the all the scores calculated using our function to create a composite score to rank on. Let's take a simple average of the correlation and magnitude scores for each driver. Note that if we consider it more important for the trend of the drivers to be similar, we can increase the weight of the correlation score (a simple average gives a weight of 0.5 to the corr_weight) or if we consider the absolute size of the drivers to be more important, we can lower the weight of the correlation score.

In [29]:
```
# Merge the correlation and magnitude distance dataframes
score_n_sales = pd.merge(corr_n_sales, magnitude_n_sales, on=['Store1', 'Store2'])
score_n_customers = pd.merge(corr_n_customers, magnitude_n_customers, on=['Store1', 'Store2'])

# Define the correlation weight
corr_weight = 0.5

# Calculate the composite scores
score_n_sales['score_n_sales'] = corr_weight * score_n_sales['corr_measure'] + (1 - corr_weight) * score
score_n_customers['score_n_cust'] = corr_weight * score_n_customers['corr_measure'] + (1 - corr_weight)
```

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

In [30]:
```
# Merge the sales and customer scores
control_store = pd.merge(score_n_sales, score_n_customers, on=['Store1', 'Store2'])

# Calculate the final control score
control_store['final_control_store'] = 0.5 * control_store['score_n_sales'] + 0.5 * control_store['score

# Exclude the trial stores from consideration
filtered_control_store = control_store[control_store['Store2'] != trial_store]

# Sort the stores based on the final control score in descending order
sorted_control_store = filtered_control_store.sort_values(by='final_control_store', ascending=False)
```

Select control stores based on the highest matching store (closest to 1 but not the store itself, i.e. the second ranked highest store)

In [31]:
```
# Use the selected control store
control_store = int(sorted_control_store.iloc[0]['Store2'])

# Print the trial and control store
print(f"Trial store = {trial_store}")
print(f"Control store = {control_store}")
```

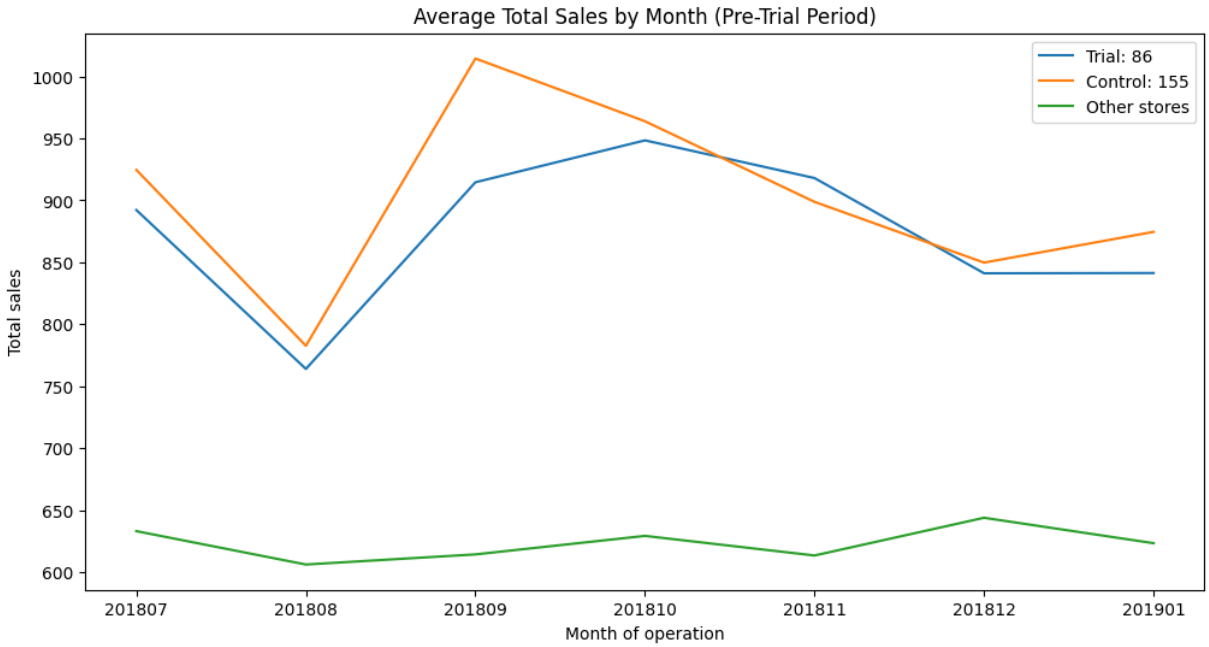
```
Trial store = 86
Control store = 155
```

Now that we have found a control store for trial store 86 which is store 155, let's check visually if the drivers are indeed similar in the period before the trial.

In [32]:
```
# Create a new column 'store_type' to label the trial and control stores
measure_overtime['store_type'] = measure_overtime['STORE_NBR'].apply(lambda x: 'Trial' if x == trial_sto

# Filter to the pre-trial period and stores with full observation periods
pre_trial_period = measure_overtime['YEARMONTH'] < '201902'
pre_trial_measures = measure_overtime[pre_trial_period & (measure_overtime['STORE_NBR'].isin(stores_with

# Filter to the trial period and stores with full observation periods
trial_period = (measure_overtime['YEARMONTH'] >= '201902')
trial_measures = measure_overtime[trial_period & (measure_overtime['STORE_NBR'].isin(stores_with_full_ob
```

We'll look at total sales first. Visual checks on trends based on the drivers

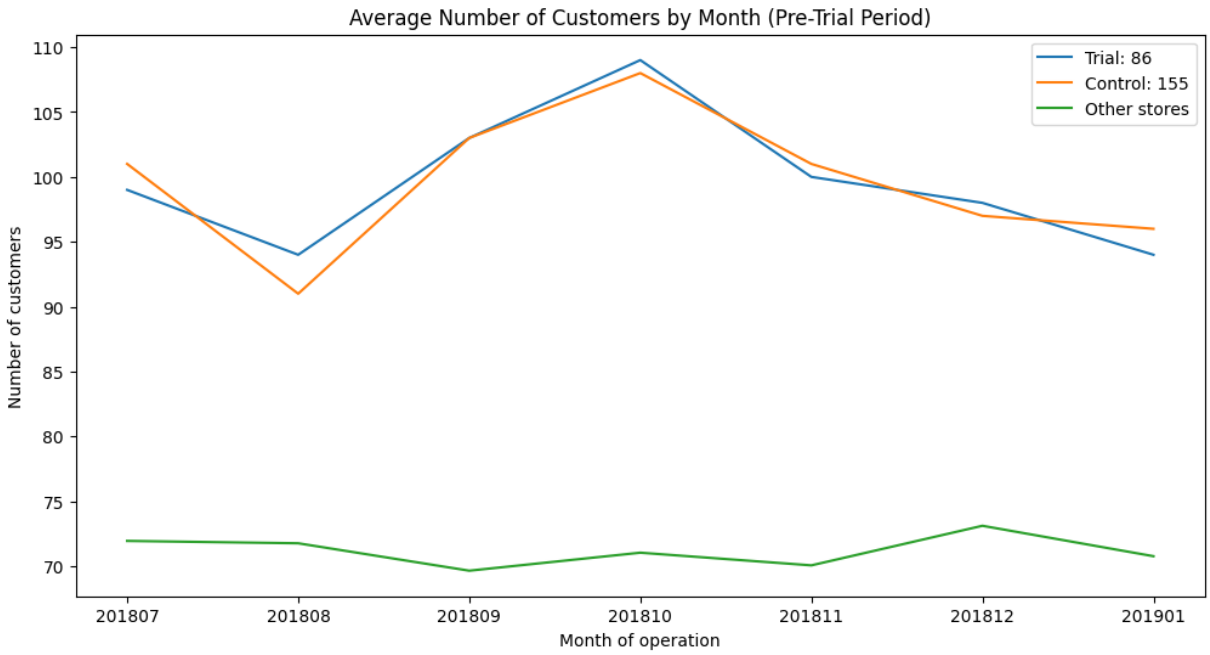
In [33]:
```
# Calculate the mean total sales for each 'YEARMONTH' and 'store_type'
past_sales = pre_trial_measures.groupby(['YEARMONTH', 'store_type'])['tot_sales'].mean().reset_index()

# Plot the total sales
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Other stores']:
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(past_sales.loc[past_sales['store_type'] == store_type, 'YEARMONTH'],
             past_sales.loc[past_sales['store_type'] == store_type, 'tot_sales'],
             label=store_label)

plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Average Total Sales by Month (Pre-Trial Period)')
plt.legend()
plt.show()
```



Sales are trending in a similar way. Next, number of customers. Visual checks on customer count trends by comparing the trial store to the control store and other stores.

In [34]:
```
# Calculate the mean number of customers for each 'YEARMONTH' and 'store_type'
past_customers = pre_trial_measures.groupby(['YEARMONTH', 'store_type'])['n_customers'].mean().reset_ind

# Plot the number of customers
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Other stores']:
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(past_customers.loc[past_customers['store_type'] == store_type, 'YEARMONTH'],
             past_customers.loc[past_customers['store_type'] == store_type, 'n_customers'], label=store_
plt.xlabel('Month of operation')
plt.ylabel('Number of customers')
plt.title('Average Number of Customers by Month (Pre-Trial Period)')
```

```
plt.legend()
plt.show()
```



The trend in number of customers is also similar. Let's now assess the impact of the trial on sales.

In [35]:
```
# Apply scale_control function for trial store 86
scaled_control = scale_control()
```

In [36]:
```
# Apply calculate_percentage_diff function for trial store 86
percentage_diff_sales = calculate_percentage_diff('sales')
```

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

In [37]:
```
# Filter percentage_diff to the pre_trial_period
filtered_data = percentage_diff_sales.loc[pre_trial_period].copy()

# Calculate the standard deviation of the percentage difference in the pre-trial period
std_dev = filtered_data['percentage_diff_sales'].std()

# Round the 'control_sales' values to one decimal place
filtered_data['control_sales'] = filtered_data['control_sales'].round(1)

# Calculate the t-values for the trial months
sample_size = len(filtered_data)
filtered_data['t_value'] = (filtered_data['percentage_diff_sales'] - 0) / (std_dev / (sample_size ** 0.5
# Degrees of freedom (df) for the t-distribution. Note that there are 8 months in the pre-trial period
degrees_of_freedom = 8 - 1

# Probability value corresponding to the 95th percentile
percentile_value = 1 - 0.05

# Calculate the 95th percentile of the t-distribution
t_95_percentile = stats.t.ppf(percentile_value, df=degrees_of_freedom)

# Add 't_95_percentile' to the percentage_diff dataframe
filtered_data['t_95_percentile'] = t_95_percentile

filtered_data
```

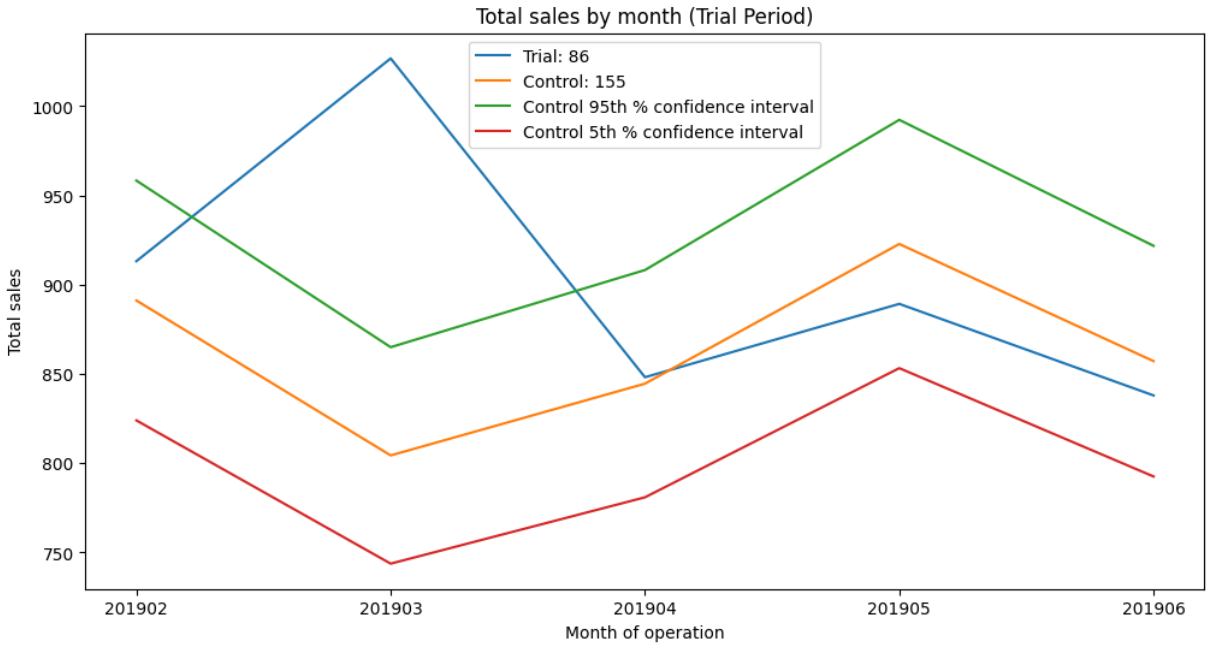
Out[37]:

	YEARMONTH	tot_sales	n_customers	control_sales	control_customers	percentage_diff_sales	t_value	t_95_pe
0	201807	892.20	99	896.9	101.0	-0.005265	-0.369632	1
1	201808	764.05	94	759.3	91.0	0.006296	0.441987	1
2	201809	914.60	103	984.0	103.0	-0.070561	-4.953810	1
3	201810	948.40	109	934.9	108.0	0.014387	1.010067	1
4	201811	918.00	100	871.9	101.0	0.052880	3.712487	1
5	201812	841.20	98	824.4	97.0	0.020426	1.434056	1
6	201901	841.40	94	848.4	96.0	-0.008273	-0.580818	1

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

In [38]:
```
# Calculate the mean total sales for each 'YEARMONTH' and 'store_type'
trial_sales = trial_measures.groupby(['YEARMONTH', 'store_type'])['tot_sales'].mean().reset_index()

# Calculate the control store 95th percentile
controls_sales_95 = trial_sales.loc[trial_sales['store_type'] == 'Control'].copy()
controls_sales_95['tot_sales'] *= (1 + std_dev * 2)
controls_sales_95['store_type'] = 'Control 95th % confidence interval'

# Calculate the control store 5th percentile
controls_sales_5 = trial_sales.loc[trial_sales['store_type'] == 'Control'].copy()
controls_sales_5['tot_sales'] *= (1 - std_dev * 2)
controls_sales_5['store_type'] = 'Control 5th % confidence interval'

# Combine the DataFrames
trial_assessment = pd.concat([trial_sales, controls_sales_95, controls_sales_5])
```

In [39]:
```
# Plot the total sales
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Control 95th % confidence interval', 'Control 5th % confidence i
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(trial_assessment.loc[trial_assessment['store_type'] == store_type, 'YEARMONTH'],
             trial_assessment.loc[trial_assessment['store_type'] == store_type, 'tot_sales'], label=stor
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Total sales by month (Trial Period)')
plt.legend()
plt.show()
```



The results show that the trial in store 86 is not significantly different to its control store in the trial period as the trial store performance lies inside the 5% to 95% confidence interval of the control store in two of the three trial months

Let's have a look at assessing this for the number of customers as well.

In [40]:
```
# Apply calculate_percentage_diff function for trial store 86
percentage_diff_customers = calculate_percentage_diff('customers')
percentage_diff_customers
```

Out[40]:

	YEARMONTH	tot_sales	n_customers	control_sales	control_customers	percentage_diff_customers
0	201807	892.20	99	896.922236	101.0	-0.019802
1	201808	764.05	94	759.269991	91.0	0.032967
2	201809	914.60	103	984.034086	103.0	0.000000
3	201810	948.40	109	934.948790	108.0	0.009259
4	201811	918.00	100	871.894555	101.0	-0.009901
5	201812	841.20	98	824.361363	97.0	0.010309
6	201901	841.40	94	848.418979	96.0	-0.020833
7	201902	913.20	107	864.522060	95.0	0.126316
8	201903	1026.80	115	780.320405	94.0	0.223404
9	201904	848.20	105	819.317024	99.0	0.060606
10	201905	889.30	104	895.224622	106.0	-0.018868
11	201906	838.00	98	831.539845	95.0	0.031579

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

In [41]:
```
# Filter percentage_diff to the pre_trial_period
filtered_data = percentage_diff_customers.loc[pre_trial_period].copy()

# Calculate the standard deviation of the percentage difference in the pre-trial period
std_dev = filtered_data['percentage_diff_customers'].std()

# Calculate the t-values for the trial months
sample_size = len(filtered_data)
filtered_data['t_value'] = (filtered_data['percentage_diff_customers'] - 0) / (std_dev / (sample_size **

# Degrees of freedom (df) for the t-distribution. Note that there are 8 months in the pre-trial period
degrees_of_freedom = 8 - 1
```

```
# Probability value corresponding to the 95th percentile
percentile_value = 1 - 0.05

# Calculate the 95th percentile of the t-distribution
t_95_percentile = stats.t.ppf(percentile_value, df=degrees_of_freedom)

# Add 't_95_percentile' to the percentage_diff dataframe
filtered_data['t_95_percentile'] = t_95_percentile


filtered_data
```

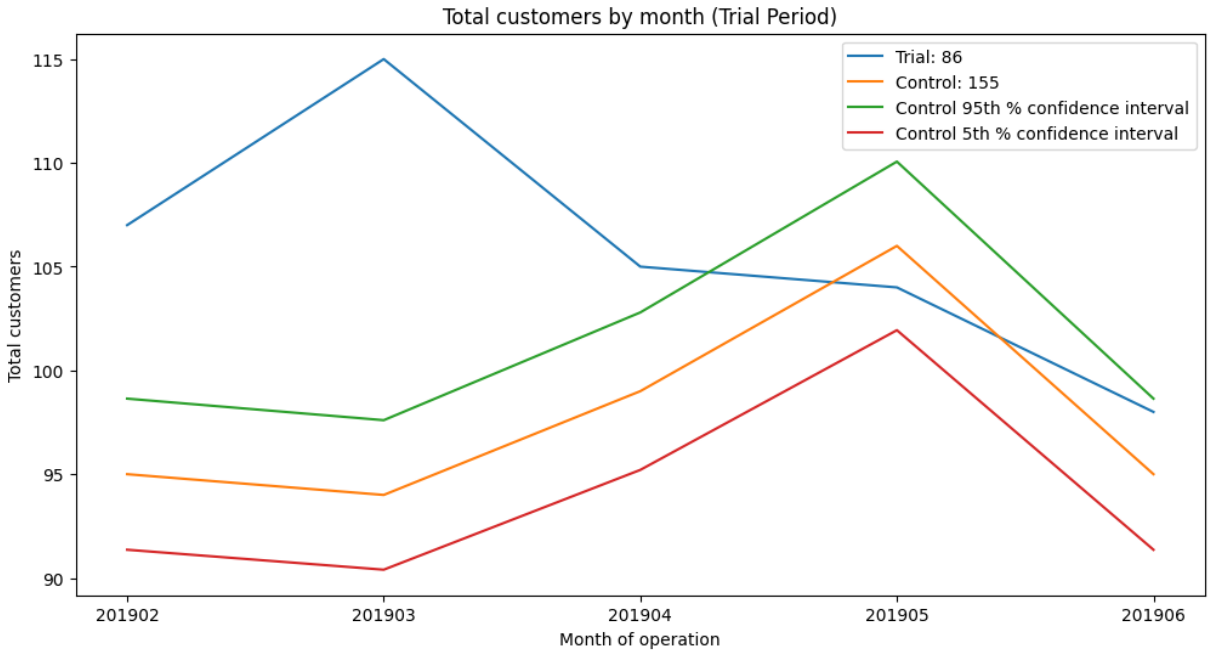
Out[41]:

	YEARMONTH	tot_sales	n_customers	control_sales	control_customers	percentage_diff_customers	t_value	t_
0	201807	892.20	99	896.922236	101.0	-0.019802	-2.734518	
1	201808	764.05	94	759.269991	91.0	0.032967	4.552521	
2	201809	914.60	103	984.034086	103.0	0.000000	0.000000	
3	201810	948.40	109	934.948790	108.0	0.009259	1.278640	
4	201811	918.00	100	871.894555	101.0	-0.009901	-1.367259	
5	201812	841.20	98	824.361363	97.0	0.010309	1.423641	
6	201901	841.40	94	848.418979	96.0	-0.020833	-2.876940	

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

In [42]:
```
# Calculate the mean total sales for each 'YEARMONTH' and 'store_type'
trial_customers = trial_measures.groupby(['YEARMONTH', 'store_type'])['n_customers'].mean().reset_index(

# Calculate the control store 95th percentile
controls_customers_95 = trial_customers.loc[trial_customers['store_type'] == 'Control'].copy()
controls_customers_95['n_customers'] *= (1 + std_dev * 2)
controls_customers_95['store_type'] = 'Control 95th % confidence interval'

# Calculate the control store 5th percentile
controls_customers_5 = trial_customers.loc[trial_customers['store_type'] == 'Control'].copy()
controls_customers_5['n_customers'] *= (1 - std_dev * 2)
controls_customers_5['store_type'] = 'Control 5th % confidence interval'

# Combine the DataFrames
trial_assessment = pd.concat([trial_customers, controls_customers_95, controls_customers_5])
```

In [43]:
```
# Plot the total customers
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Control 95th % confidence interval', 'Control 5th % confidence i
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(trial_assessment.loc[trial_assessment['store_type'] == store_type, 'YEARMONTH'],
             trial_assessment.loc[trial_assessment['store_type'] == store_type, 'n_customers'], label=st
plt.xlabel('Month of operation')
plt.ylabel('Total customers')
plt.title('Total customers by month (Trial Period)')
plt.legend()
plt.show()
```



It looks like the number of customers is significantly higher in all of the three months. This seems to suggest that the trial had a significant impact on increasing the number of customers in trial store 86 but as we saw, sales were not significantly higher. We should check with the Category Manager if there were special deals in the trial store that were may have resulted in lower prices, impacting the results.

Trial store 88

In [44]:
```
# Define the trial store
trial_store = 88

# Calculate correlations for total sales and number of customers
corr_n_sales = calculate_correlation(pre_trial_measures, 'tot_sales', trial_store)
corr_n_customers = calculate_correlation(pre_trial_measures, 'n_customers', trial_store)

# Calculate magnitude distances for total sales and number of customers
magnitude_n_sales = calculate_magnitude_distance(pre_trial_measures, 'tot_sales', trial_store)
magnitude_n_customers = calculate_magnitude_distance(pre_trial_measures, 'n_customers', trial_store)
```

We'll need to combine the all the scores calculated using our function to create a composite score to rank on. Let's take a simple average of the correlation and magnitude scores for each driver. Note that if we consider it more important for the trend of the drivers to be similar, we can increase the weight of the correlation score (a simple average gives a weight of 0.5 to the corr_weight) or if we consider the absolute size of the drivers to be more important, we can lower the weight of the correlation score.

In [45]:
```
# Merge the correlation and magnitude distance dataframes
score_n_sales = pd.merge(corr_n_sales, magnitude_n_sales, on=['Store1', 'Store2'])
score_n_customers = pd.merge(corr_n_customers, magnitude_n_customers, on=['Store1', 'Store2'])

# Define the correlation weight
corr_weight = 0.5

# Calculate the composite scores
score_n_sales['score_n_sales'] = corr_weight * score_n_sales['corr_measure'] + (1 - corr_weight) * score
score_n_customers['score_n_cust'] = corr_weight * score_n_customers['corr_measure'] + (1 - corr_weight)
```

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

In [46]:
```
# Merge the sales and customer scores
control_store = pd.merge(score_n_sales, score_n_customers, on=['Store1', 'Store2'])

# Calculate the final control score
control_store['final_control_store'] = 0.5 * control_store['score_n_sales'] + 0.5 * control_store['score

# Exclude the trial stores from consideration
```

```
filtered_control_store = control_store[control_store['Store2'] != trial_store]

# Sort the stores based on the final control score in descending order
sorted_control_store = filtered_control_store.sort_values(by='final_control_store', ascending=False)
```

Select control stores based on the highest matching store (closest to 1 but not the store itself, i.e. the second ranked highest store)

In [47]:
```
# Use the selected control store
control_store = int(sorted_control_store.iloc[0]['Store2'])

# Print the trial and control store
print(f"Trial store = {trial_store}")
print(f"Control store = {control_store}")
```

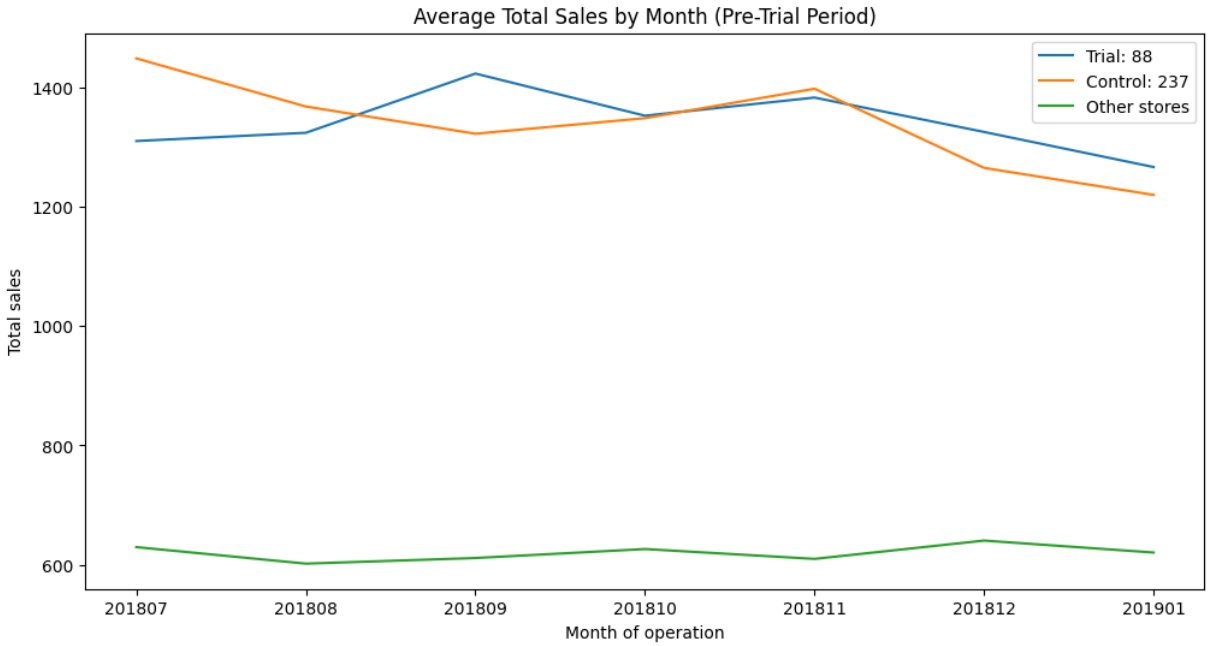
```
Trial store = 88
Control store = 237
```

Now that we have found a control store for trial store 88 which is store 237, let's check visually if the drivers are indeed similar in the period before the trial.

In [48]:
```
# Create a new column 'store_type' to label the trial and control stores
measure_overtime['store_type'] = measure_overtime['STORE_NBR'].apply(lambda x: 'Trial' if x == trial_sto

# Filter to the pre-trial period and stores with full observation periods
pre_trial_period = measure_overtime['YEARMONTH'] < '201902'
pre_trial_measures = measure_overtime[pre_trial_period & (measure_overtime['STORE_NBR'].isin(stores_with

# Filter to the trial period and stores with full observation periods
trial_period = (measure_overtime['YEARMONTH'] >= '201902')
trial_measures = measure_overtime[trial_period & (measure_overtime['STORE_NBR'].isin(stores_with_full_ob
```

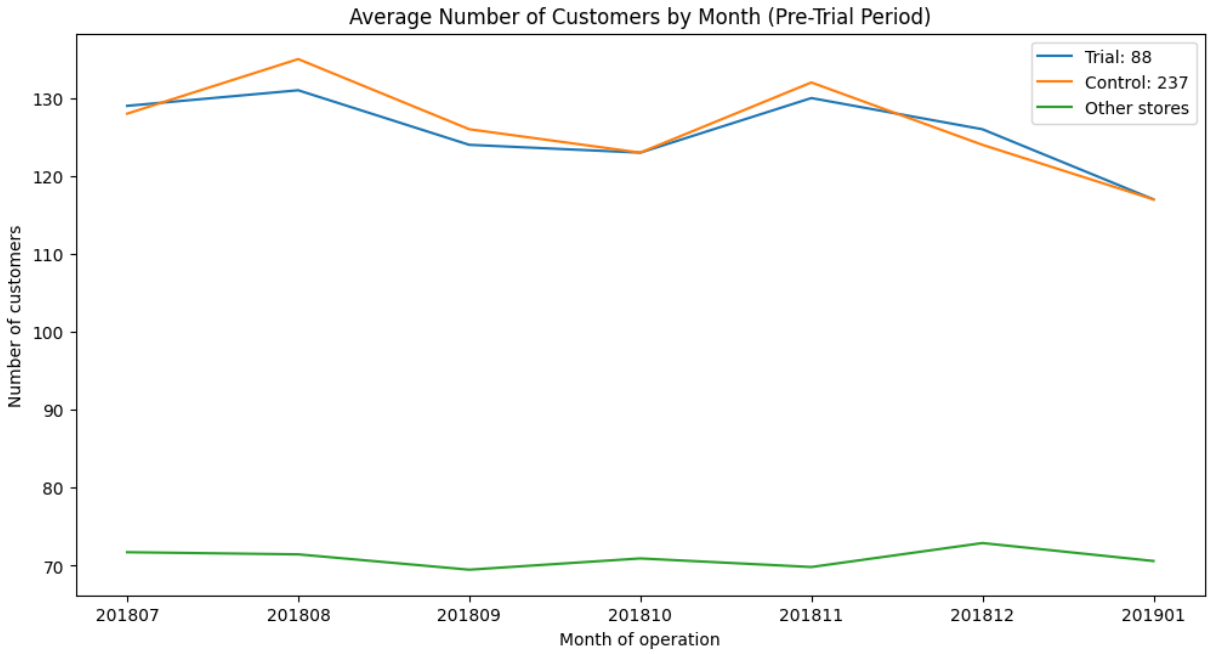
We'll look at total sales first. Visual checks on trends based on the drivers

In [49]:
```
# Calculate the mean total sales for each 'YEARMONTH' and 'store_type'
past_sales = pre_trial_measures.groupby(['YEARMONTH', 'store_type'])['tot_sales'].mean().reset_index()

# Plot the total sales
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Other stores']:
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(past_sales.loc[past_sales['store_type'] == store_type, 'YEARMONTH'],
             past_sales.loc[past_sales['store_type'] == store_type, 'tot_sales'],
             label=store_label)

plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Average Total Sales by Month (Pre-Trial Period)')
plt.legend()
plt.show()
```



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

In [50]:
```
# Calculate the mean number of customers for each 'YEARMONTH' and 'store_type'
past_customers = pre_trial_measures.groupby(['YEARMONTH', 'store_type'])['n_customers'].mean().reset_ind

# Plot the number of customers
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Other stores']:
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(past_customers.loc[past_customers['store_type'] == store_type, 'YEARMONTH'],
             past_customers.loc[past_customers['store_type'] == store_type, 'n_customers'], label=store_
plt.xlabel('Month of operation')
plt.ylabel('Number of customers')
plt.title('Average Number of Customers by Month (Pre-Trial Period)')
plt.legend()
plt.show()
```



Total number of customers of the control and trial stores are also similar. Let's now assess the impact of the trial on sales.

In [51]:
```
# Apply scale_control function for trial store 86
scaled_control = scale_control()
```

In [52]:
```
# Apply calculate_percentage_diff function for trial store 86
percentage_diff_sales = calculate_percentage_diff('sales')
```

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

In [53]:
```
# Filter percentage_diff to the pre_trial_period
filtered_data = percentage_diff_sales.loc[pre_trial_period].copy()

# Calculate the standard deviation of the percentage difference in the pre-trial period
std_dev = filtered_data['percentage_diff_sales'].std()

# Round the 'control_sales' values to one decimal place
filtered_data['control_sales'] = filtered_data['control_sales'].round(1)

# Calculate the t-values for the trial months
sample_size = len(filtered_data)
filtered_data['t_value'] = (filtered_data['percentage_diff_sales'] - 0) / (std_dev / (sample_size ** 0.5
# Degrees of freedom (df) for the t-distribution. Note that there are 8 months in the pre-trial period
degrees_of_freedom = 8 - 1

# Probability value corresponding to the 95th percentile
percentile_value = 1 - 0.05

# Calculate the 95th percentile of the t-distribution
t_95_percentile = stats.t.ppf(percentile_value, df=degrees_of_freedom)

# Add 't_95_percentile' to the percentage_diff dataframe
filtered_data['t_95_percentile'] = t_95_percentile

filtered_data
```

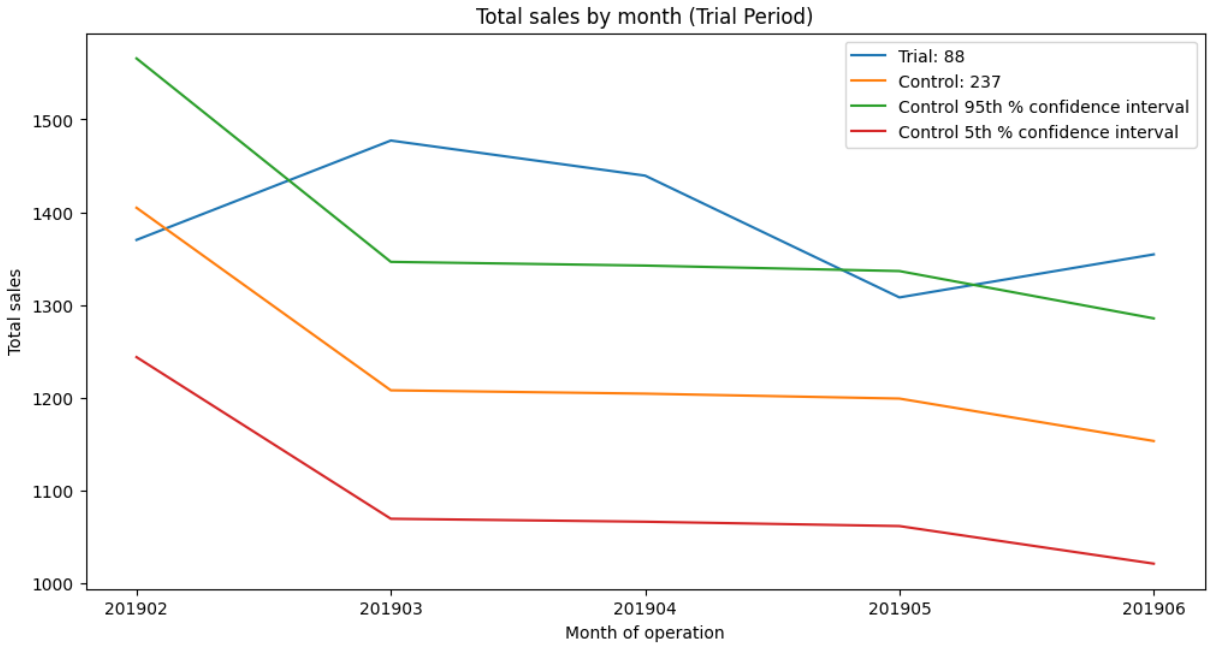
Out[53]:

	YEARMONTH	tot_sales	n_customers	control_sales	control_customers	percentage_diff_sales	t_value	t_95_pe
0	201807	1310.0	129	1450.7	127.276836	-0.096961	-4.480980	1
1	201808	1323.8	131	1369.9	134.237288	-0.033674	-1.556233	1
2	201809	1423.0	124	1324.3	125.288136	0.074562	3.445831	1
3	201810	1352.4	123	1350.4	122.305085	0.001480	0.068408	1
4	201811	1382.8	130	1399.8	131.254237	-0.012129	-0.560533	1
5	201812	1325.2	126	1267.0	123.299435	0.045959	2.123961	1
6	201901	1266.4	117	1221.6	116.338983	0.036673	1.694799	1

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

In [54]:
```
# Calculate the mean total sales for each 'YEARMONTH' and 'store_type'
trial_sales = trial_measures.groupby(['YEARMONTH', 'store_type'])['tot_sales'].mean().reset_index()

# Calculate the control store 95th percentile
controls_sales_95 = trial_sales.loc[trial_sales['store_type'] == 'Control'].copy()
controls_sales_95['tot_sales'] *= (1 + std_dev * 2)
controls_sales_95['store_type'] = 'Control 95th % confidence interval'

# Calculate the control store 5th percentile
controls_sales_5 = trial_sales.loc[trial_sales['store_type'] == 'Control'].copy()
controls_sales_5['tot_sales'] *= (1 - std_dev * 2)
controls_sales_5['store_type'] = 'Control 5th % confidence interval'

# Combine the DataFrames
trial_assessment = pd.concat([trial_sales, controls_sales_95, controls_sales_5])
```

In [55]:
```
# Plot the total sales
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Control 95th % confidence interval', 'Control 5th % confidence i
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(trial_assessment.loc[trial_assessment['store_type'] == store_type, 'YEARMONTH'],
             trial_assessment.loc[trial_assessment['store_type'] == store_type, 'tot_sales'], label=stor
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Total sales by month (Trial Period)')
plt.legend()
plt.show()
```



The results show that the trial in store 88 is significantly different to its control store in the trial period as the trial store performance lies outside of the 5% to 95% confidence interval of the control store in two of the three trial months.

Let's have a look at assessing this for the number of customers as well.

In [56]:
```
# Apply calculate_percentage_diff function for trial store 86
percentage_diff_customers = calculate_percentage_diff('customers')
percentage_diff_customers
```

Out[56]:

	YEARMONTH	tot_sales	n_customers	control_sales	control_customers	percentage_diff_customers
0	201807	1310.00	129	1450.657086	127.276836	0.013539
1	201808	1323.80	131	1369.931485	134.237288	-0.024116
2	201809	1423.00	124	1324.260425	125.288136	-0.010281
3	201810	1352.40	123	1350.401097	122.305085	0.005682
4	201811	1382.80	130	1399.777923	131.254237	-0.009556
5	201812	1325.20	126	1266.971288	123.299435	0.021902
6	201901	1266.40	117	1221.600696	116.338983	0.005682
7	201902	1370.20	124	1406.989143	125.288136	-0.010281
8	201903	1477.20	134	1210.082775	118.327684	0.132448
9	201904	1439.40	128	1206.477165	119.322034	0.072727
10	201905	1308.25	128	1201.168906	128.271186	-0.002114
11	201906	1354.60	121	1155.397690	118.327684	0.022584

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

In [57]:
```
# Filter percentage_diff to the pre_trial_period
filtered_data = percentage_diff_customers.loc[pre_trial_period].copy()

# Calculate the standard deviation of the percentage difference in the pre-trial period
std_dev = filtered_data['percentage_diff_customers'].std()

# Calculate the t-values for the trial months
sample_size = len(filtered_data)
filtered_data['t_value'] = (filtered_data['percentage_diff_customers'] - 0) / (std_dev / (sample_size **
```

```
# Degrees of freedom (df) for the t-distribution. Note that there are 8 months in the pre-trial period
degrees_of_freedom = 8 - 1

# Probability value corresponding to the 95th percentile
percentile_value = 1 - 0.05

# Calculate the 95th percentile of the t-distribution
t_95_percentile = stats.t.ppf(percentile_value, df=degrees_of_freedom)

# Add 't_95_percentile' to the percentage_diff dataframe
filtered_data['t_95_percentile'] = t_95_percentile


filtered_data
```

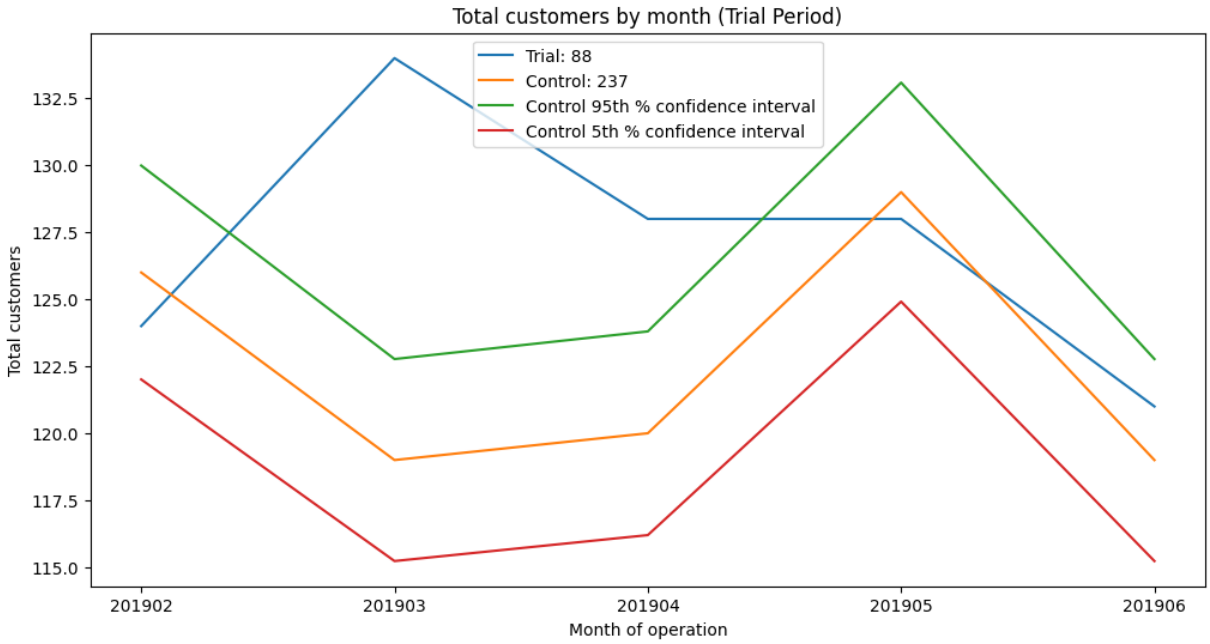
Out[57]:

	YEARMONTH	tot_sales	n_customers	control_sales	control_customers	percentage_diff_customers	t_value	t_
0	201807	1310.0	129	1450.657086	127.276836	0.013539	2.261671	
1	201808	1323.8	131	1369.931485	134.237288	-0.024116	-4.028657	
2	201809	1423.0	124	1324.260425	125.288136	-0.010281	-1.717528	
3	201810	1352.4	123	1350.401097	122.305085	0.005682	0.949160	
4	201811	1382.8	130	1399.777923	131.254237	-0.009556	-1.596315	
5	201812	1325.2	126	1266.971288	123.299435	0.021902	3.658859	
6	201901	1266.4	117	1221.600696	116.338983	0.005682	0.949160	

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

In [58]:
```
# Calculate the mean total sales for each 'YEARMONTH' and 'store_type'
trial_customers = trial_measures.groupby(['YEARMONTH', 'store_type'])['n_customers'].mean().reset_index(

# Calculate the control store 95th percentile
controls_customers_95 = trial_customers.loc[trial_customers['store_type'] == 'Control'].copy()
controls_customers_95['n_customers'] *= (1 + std_dev * 2)
controls_customers_95['store_type'] = 'Control 95th % confidence interval'

# Calculate the control store 5th percentile
controls_customers_5 = trial_customers.loc[trial_customers['store_type'] == 'Control'].copy()
controls_customers_5['n_customers'] *= (1 - std_dev * 2)
controls_customers_5['store_type'] = 'Control 5th % confidence interval'

# Combine the DataFrames
trial_assessment = pd.concat([trial_customers, controls_customers_95, controls_customers_5])
```

In [59]:
```
# Plot the total customers
plt.figure(figsize=(12, 6))

for store_type in ['Trial', 'Control', 'Control 95th % confidence interval', 'Control 5th % confidence i
    if store_type == 'Trial':
        store_label = f"{store_type}: {trial_store}"
    elif store_type == 'Control':
        store_label = f"{store_type}: {control_store}"
    else:
        store_label = store_type

    plt.plot(trial_assessment.loc[trial_assessment['store_type'] == store_type, 'YEARMONTH'],
             trial_assessment.loc[trial_assessment['store_type'] == store_type, 'n_customers'], label=st
plt.xlabel('Month of operation')
plt.ylabel('Total customers')
plt.title('Total customers by month (Trial Period)')
plt.legend()
plt.show()
```



Total number of customers in the trial period for the trial store is significantly higher than the control store for two out of three months, which indicates a positive trial effect.

Conclusion

The results for trial stores 77 and 88 during the trial period show a significant difference in at least two of the three trial months but this is not the case for trial store 86. We can check with the client if the implementation of the trial was different in trial store 86 but overall, the trial shows a significant increase in sales. Now that we have finished our analysis, we can prepare our presentation to the Category Manager.