In [87]:

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

Stores Doing Trial: 77, 86, 88

In [4]:

```
df = pd.read_csv('QVI_clean.scv')
```

In [5]:

```
df.head(4)
```

Out[5]:

	Unnamed: 0	LYLTY_CARD_NBR	LIFESTAGE	PREMIUM_CUSTOMER	DATE	STORE_NBR	TXN_ID	PROD_NBR	PROD_NA
0	0	47142.0	MIDAGE SINGLES/COUPLES	Budget	2018-07-01	47.0	42540.0	14.0	Smiths Cr Chip O Big Bag 3
1	1	55073.0	MIDAGE SINGLES/COUPLES	Budget	2018-07-01	55.0	48884.0	99.0	Pringles S FriedChic 1
2	2	55073.0	MIDAGE SINGLES/COUPLES	Budget	2018-07-01	55.0	48884.0	91.0	CCs T Cheese 1
3	3	58351.0	MIDAGE SINGLES/COUPLES	Budget	2018-07-01	58.0	54374.0	102.0	K Mozzar Basil & P 1

Remove Stores With Too Little Transactional Data Across the Year

We are interested in stores that have data throughout the entire year. So lets filter out the stores which staisfy this

In [6]:

```
df.groupby(by='STORE_NBR')['DATE'].nunique().sort_values(ascending=False).head(5)
```

Out[6]:

```
STORE_NBR
133.0    362
152.0    362
226.0    362
213.0    362
259.0    362
Name: DATE, dtype: int64
```

In [485…

```
# Stores with missing months

stores_with_missing_months = df.groupby(by='STORE_NBR')['MONTH'].nunique()
filtered_stores = stores_with_missing_months[stores_with_missing_months < 10].index

print(f'Stores with fewer than 12 months:: {filtered_stores}')
```

```
Stores with fewer than 12 months:: Index([], dtype='float64', name='STORE_NBR')
```

In [486…

```
# Drop these
df = df[~df['STORE_NBR'].isin(filtered_stores)]
```

In [487…

```
# There will still be stores with only a few transaction p/month.
# We can filter these by removing sotres who have.
stores_with_low_monthly_trans = df.groupby(by=['STORE_NBR', 'MONTH'])['DATE'].nunique()
filtered_stores = stores_with_low_monthly_trans[stores_with_low_monthly_trans < 15]
# Get the unique FIRST index.
filtered_stores = filtered_stores.index.get_level_values(0).unique()

# Drop these
df = df[~df['STORE_NBR'].isin(filtered_stores)]
```

```
/var/folders/b3/8dnps2js0rz1xgw1gkx248p40000gn/T/ipykernel_1504/2360524443.py:3: FutureWarning: The default of o
bserved=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to re
tain current behavior or observed=True to adopt the future default and silence this warning.
  stores_with_low_monthly_trans = df.groupby(by=['STORE_NBR', 'MONTH'])['DATE'].nunique()
```

In [488…

```
# How many stores are left?
df['STORE_NBR'].nunique()
```

Out[488…

```
235
```

In [489…

```
# Lets make sure we have the stores doing the trial: 77, 86, 88
```

```
df[(df['STORE_NBR'] == 78) | (df['STORE_NBR'] == 88) | (df['STORE_NBR'] == 86)]
```

Out[489…

	Unnamed: 0	LYLTY_CARD_NBR	LIFESTAGE	PREMIUM_CUSTOMER	DATE	STORE_NBR	TXN_ID	PROD_NBR	
29	29	78115.0	MIDAGE SINGLES/COUPLES	Mainstream	2018-07-01	78.0	76138.0	87.0	Infı I
31	31	88076.0	MIDAGE SINGLES/COUPLES	Mainstream	2018-07-01	88.0	86585.0	9.0	Chp
32	32	88140.0	MIDAGE SINGLES/COUPLES	Mainstream	2018-07-01	88.0	86914.0	25.0	So
115	115	86082.0	OLDER FAMILIES	Budget	2018-07-01	86.0	84643.0	3.0	Ke Ca
160	160	88155.0	OLDER FAMILIES	Mainstream	2018-07-01	88.0	86995.0	68.0	P
...	...	...	...	...	...	...	...	...	
249452	249452	78129.0	YOUNG FAMILIES	Budget	2019-06-30	78.0	76215.0	58.0	Ch
249453	249453	78176.0	YOUNG FAMILIES	Budget	2019-06-30	78.0	76482.0	61.0	Sm
249496	249496	86198.0	YOUNG FAMILIES	Mainstream	2019-06-30	86.0	85372.0	58.0	Ch
249594	249594	86045.0	YOUNG SINGLES/COUPLES	Mainstream	2019-06-30	86.0	84426.0	98.0	NCC
249595	249595	88342.0	YOUNG SINGLES/COUPLES	Mainstream	2019-06-30	88.0	87918.0	23.0	Ch

4524 rows × 17 columns

Explore Monthly Trends for the Trial Stores

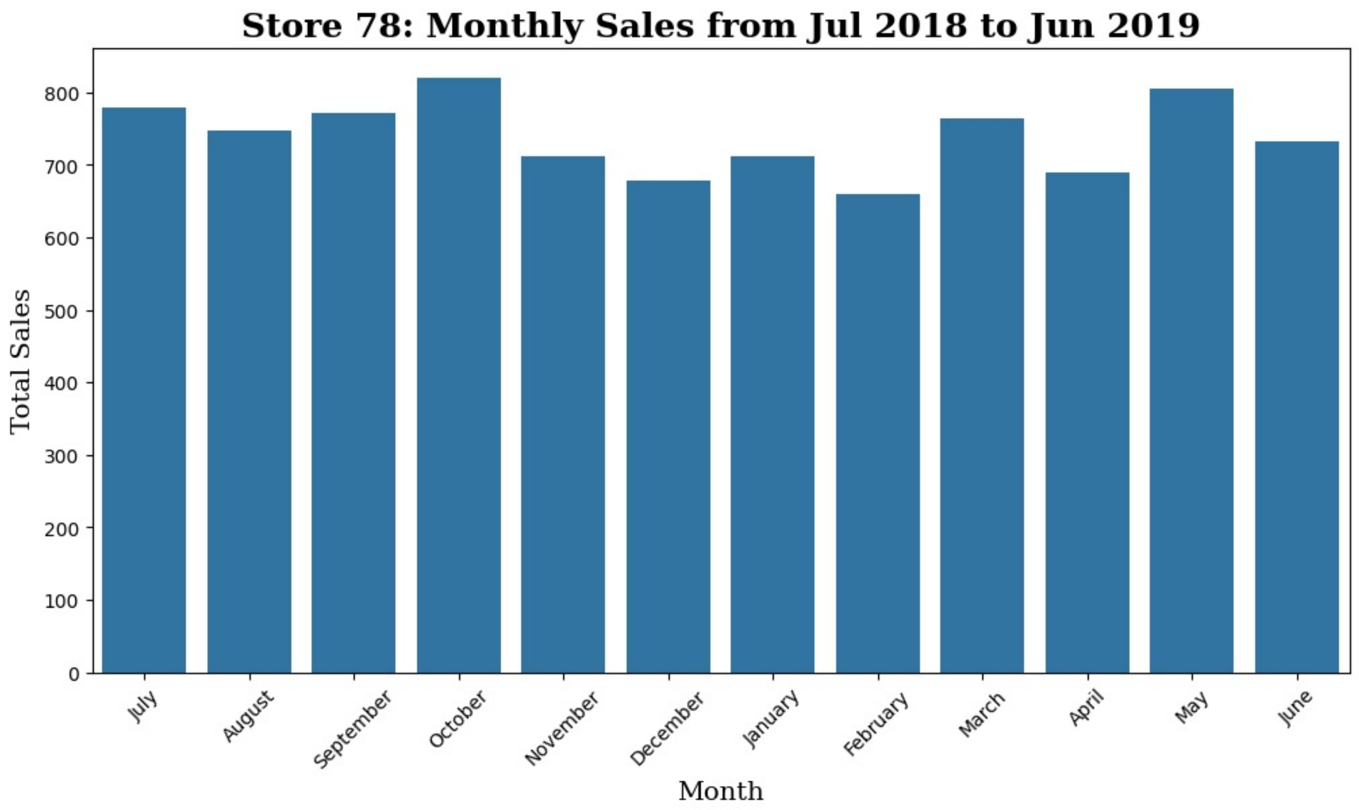
In [490…

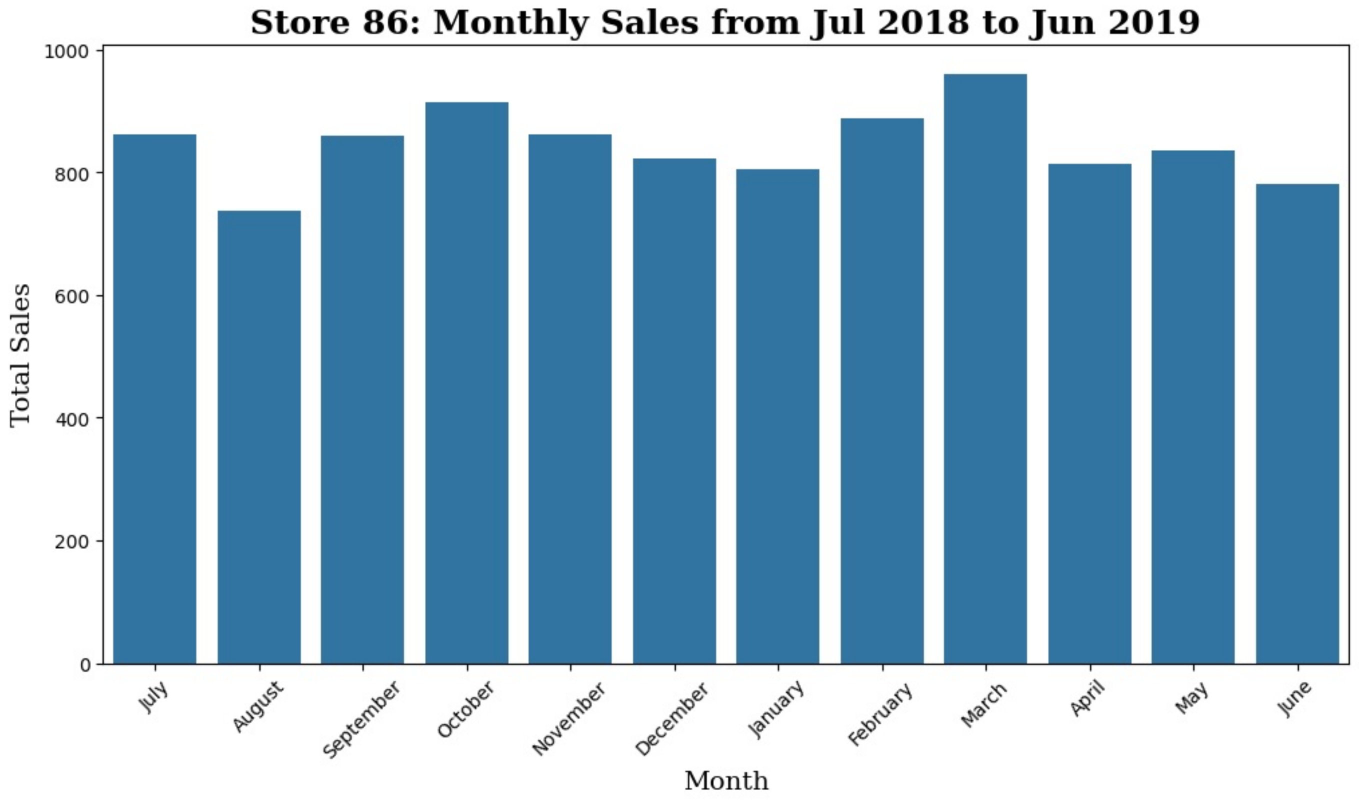
```
# Seems like none of the data types save after exporting and importing my csv so I have to do it again here
df.dropna(axis=0, inplace=True)

month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
               'August', 'September', 'October', 'November', 'December']
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
df['MONTH'] = pd.Categorical(df['MONTH'], categories=month_order, ordered=True)
df['DAY'] = pd.Categorical(df['DAY'], categories=day_order, ordered=True)
df['SIZE_IN_GRAMS'] = df['PROD_NAME'].str.extract(r'(\d+)[gG]').astype('int')
df['LIFESTAGE'] = df['LIFESTAGE'].astype('category')
df['PREMIUM_CUSTOMER'] = pd.Categorical(df['PREMIUM_CUSTOMER'], categories=['Budget', 'Mainstream', 'Premium'
df['DATE'] = pd.to_datetime(df['DATE'])
```

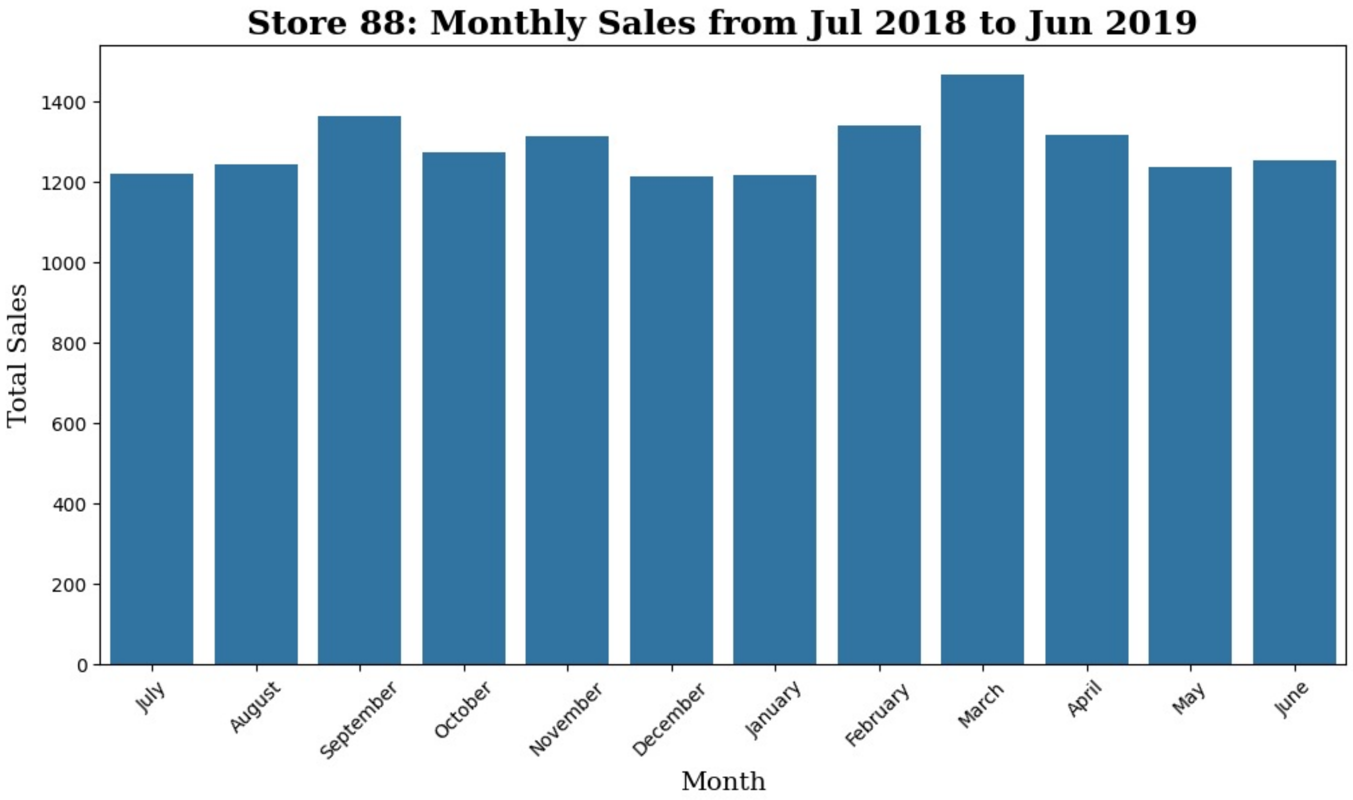
In [491…

```
month_order = ['July', 'August', 'September', 'October', 'November', 'December',
               'January', 'February', 'March', 'April', 'May', 'June']

monthly_sales = df[df['STORE_NBR'] == 78].groupby('MONTH', observed=False)['TOT_SALES'].sum()

plt.figure(figsize=(12,6))
sns.barplot(x=monthly_sales.index, y=monthly_sales, order=month_order)

plt.title('Store 78: Monthly Sales from Jul 2018 to Jun 2019', fontsize=18, fontweight='bold', fontfamily='seri
plt.xlabel('Month', fontsize=14, fontfamily='serif')
plt.ylabel('Total Sales', fontsize=14, fontfamily='serif')
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.show()
plt.close()


monthly_sales = df[df['STORE_NBR'] == 86].groupby('MONTH', observed=False)['TOT_SALES'].sum()
# monthly_sales = monthly_sales[month_order]

plt.figure(figsize=(12,6))
sns.barplot(x=monthly_sales.index, y=monthly_sales, order=month_order)

plt.title('Store 86: Monthly Sales from Jul 2018 to Jun 2019', fontsize=18, fontweight='bold', fontfamily='seri
plt.xlabel('Month', fontsize=14, fontfamily='serif')
```

```python
plt.ylabel('Total Sales', fontsize=14, fontfamily='serif')
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.show()
plt.close()


monthly_sales = df[df['STORE_NBR'] == 88].groupby('MONTH', observed=False)['TOT_SALES'].sum()
# monthly_sales = monthly_sales[month_order]

plt.figure(figsize=(12,6))
sns.barplot(x=monthly_sales.index, y=monthly_sales, order=month_order)

plt.title('Store 88: Monthly Sales from Jul 2018 to Jun 2019', fontsize=18, fontweight='bold', fontfamily='seri
plt.xlabel('Month', fontsize=14, fontfamily='serif')
plt.ylabel('Total Sales', fontsize=14, fontfamily='serif')
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.show()
```







Calculate Total Revenue p/Month, # of Customers p/Month, and Avg Transactions p/Customer p/Month.

In [888…

```
# Filter data for the period of interest (e.g., July 2018 - June 2019)
start_date = '2018-07-01'
end_date = '2019-01-31'
filtered_df = df[(df['DATE'] >= start_date) & (df['DATE'] <= end_date)]


# Get TOTAL SALES per store per month
monthly_sales = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')])['TOT_SALES'].sum().unstacl
monthly_sales = monthly_sales.T

# NO. of CUSTOMERS P/MONTH.
monthly_customers = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')])['LYLTY_CARD_NBR'].nun:
monthly_customers = monthly_customers.T

# Average number of transactions per customer per month.
monthly_transactions = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')]).size().unstack(fil
monthly_customers = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')])['LYLTY_CARD_NBR'].nun:
avg_transactions_per_customer = monthly_transactions / monthly_customers  # Calculate the average number of trai
avg_transactions_per_customer = avg_transactions_per_customer.T
```

Define function to find best Pearsons Correlation between test store and control stores

In [889…

```
# Stores
stores = monthly_sales.columns
# Transpose this again for functions
monthly_customers = monthly_customers.T
#Define trial stores
trial_stores = [77, 86, 88]
```

In [890…

```
# Create a function that minimises
def min_corr(trial_store, stores):
    # Highest correlation
    correlations = {}  # {trial_store: {control_store: (a, b, c)}}
    for j in trial_store:
        correlations[j] = {}  # Initialize inner dictionary for each trial store
        for i in stores:
            if i not in trial_store:
                a = monthly_sales.loc[:, j].corr(monthly_sales.loc[:, i])
                b = monthly_customers.loc[:, j].corr(monthly_customers.loc[:, i])
                c = avg_transactions_per_customer.loc[:, j].corr(avg_transactions_per_customer.loc[:, i])
                correlations[j][i] = (a, b, c)
            else:
                pass
       # Compute min correlations per control store
        min_scores = {store: min(corrs) for store, corrs in correlations[j].items()}

        #  # Compute highest average
        # min_scores = {store: np.median(corrs) for store, corrs in correlations[j].items()}

        best_store = max(min_scores, key=min_scores.get)
        best_corrs = correlations[j][best_store]
        print(f'Best control store for trial store {j} is store {best_store} with correlations: {best_corrs}')
    return correlations  # Return for later use
```

In [891…

```
# Find control stores with biggest correlation
correlations = min_corr(trial_store=trial_stores, stores=stores)
```

```
Best control store for trial store 77 is store 84.0 with correlations: (np.float64(0.6211394206078755), np.float
64(0.8086899604018802), np.float64(0.6622933061287001))
Best control store for trial store 86 is store 176.0 with correlations: (np.float64(0.6294791954372881), np.floa
t64(0.7558204875430532), np.float64(0.7900571473834768))
Best control store for trial store 88 is store 145.0 with correlations: (np.float64(0.5119102374038524), np.floa
t64(0.46630617823260456), np.float64(0.5467904521576857))
```

Define function to find lowest standardised metric between test stores and control stores

In [892…

```
# Create a function to calculate the standardized metric
def standardized_metric(trial_store, stores):
    # Store for tracking the absolute differences
    performance_diff = {}  # {trial_sotre: {store: (sales_diff, customer_diff, transactions_diff)}}

    # Iterate over the trial stores
    for j in trial_store:
        performance_diff[j] = {}  # Initialize inner dictionary for each trial store
        for i in stores:
            if i not in trial_store:
                # Calculate absolute differences for each metric
                sales_diff = np.abs(monthly_sales.loc[:, j] - monthly_sales.loc[:, i])
                customer_diff = np.abs(monthly_customers.loc[:, j] - monthly_customers.loc[:, i])
                transactions_diff = np.abs(avg_transactions_per_customer.loc[:, j] - avg_transactions_per_custo

                # Standardize the differences by dividing by the trial store's std deviation
                sales_std = monthly_sales.loc[:, j].std()
                customer_std = monthly_customers.loc[:, j].std()
                transactions_std = avg_transactions_per_customer.loc[:, j].std()

                standardized_sales_diff = sales_diff / sales_std if sales_std != 0 else sales_diff
                standardized_customer_diff = customer_diff / customer_std if customer_std != 0 else customer_di
                standardized_transactions_diff = transactions_diff / transactions_std if transactions_std != 0

                '''
                # LESS ROBUST
                # Calculate the total standardized difference (could sum or average them)
                total_diff = standardized_sales_diff + standardized_customer_diff + standardized_transactions_d
                performance_diff[j][i] = total_diff.mean()  # Store the mean of the total difference
                '''

                # Calculate the total standardized difference
                performance_diff[j][i] = np.median([
                    standardized_sales_diff.median(),
                        standardized_customer_diff.median(),
                            standardized_transactions_diff.median()
                                ])

        # Now, find the control store with the minimum standardized difference
        best_control_store = min(performance_diff[j], key=performance_diff[j].get)
        best_diff = performance_diff[j][best_control_store]

        print(f'For trial store {j}: Best control store is {best_control_store} with standardized difference: {
        # return(performance_diff)

    return performance_diff  # for later use
```

In [893…

```
performance_use = standardized_metric(trial_store=trial_stores, stores=stores)
```

```
For trial store 77: Best control store is 233.0 with standardized difference: 0.436
For trial store 86: Best control store is 219.0 with standardized difference: 0.505
For trial store 88: Best control store is 237.0 with standardized difference: 0.697
```

Create function to find control store based on the highest correlation and lowest standardised metric

In [908…

```
def decide_best_control_store(trial_store, stores, correlations, standardized_differences, w1=0.4, w2=0.6): # C
    best_control = None
    best_score = -float('inf')  # Set an initial best score

    for store in stores:
        if store != trial_store:
            # Get correlation and standardized difference for current store
            corr = min(correlations[trial_store].get(store, (0,0,0)))
            std_diff = standardized_differences[trial_store].get(store, float('inf'))

            # Calculate weighted score
            score = w1 * corr + w2 * (1 / std_diff if std_diff != 0 else float('inf'))  # Avoid divide by zero

            # Update best control store based on score
            if score > best_score:
                best_score = score
                best_control = store

    return best_control, best_score
```

In [909…

```
# Example usage for each trial store
for trial_store in trial_stores:
    best_control, best_score = decide_best_control_store(trial_store, stores, correlations, performance_use)
    print(f"For trial store {trial_store}, the best control store is {best_control} with a score of {best_score
```

```
For trial store 77, the best control store is 233.0 with a score of 1.256
For trial store 86, the best control store is 219.0 with a score of 1.193
For trial store 88, the best control store is 237.0 with a score of 0.823
```

The stores with the a combination of the highest correlation and lowest standardised metric for our metrics

- Total revenue p/month.
- No. of customers per month.
- Avg transaction p/customer p/month.

are:

- For trial store 77, the best control store is 233
- For trial store 86, the best control store is 219
- For trial store 88, the best control store is 237

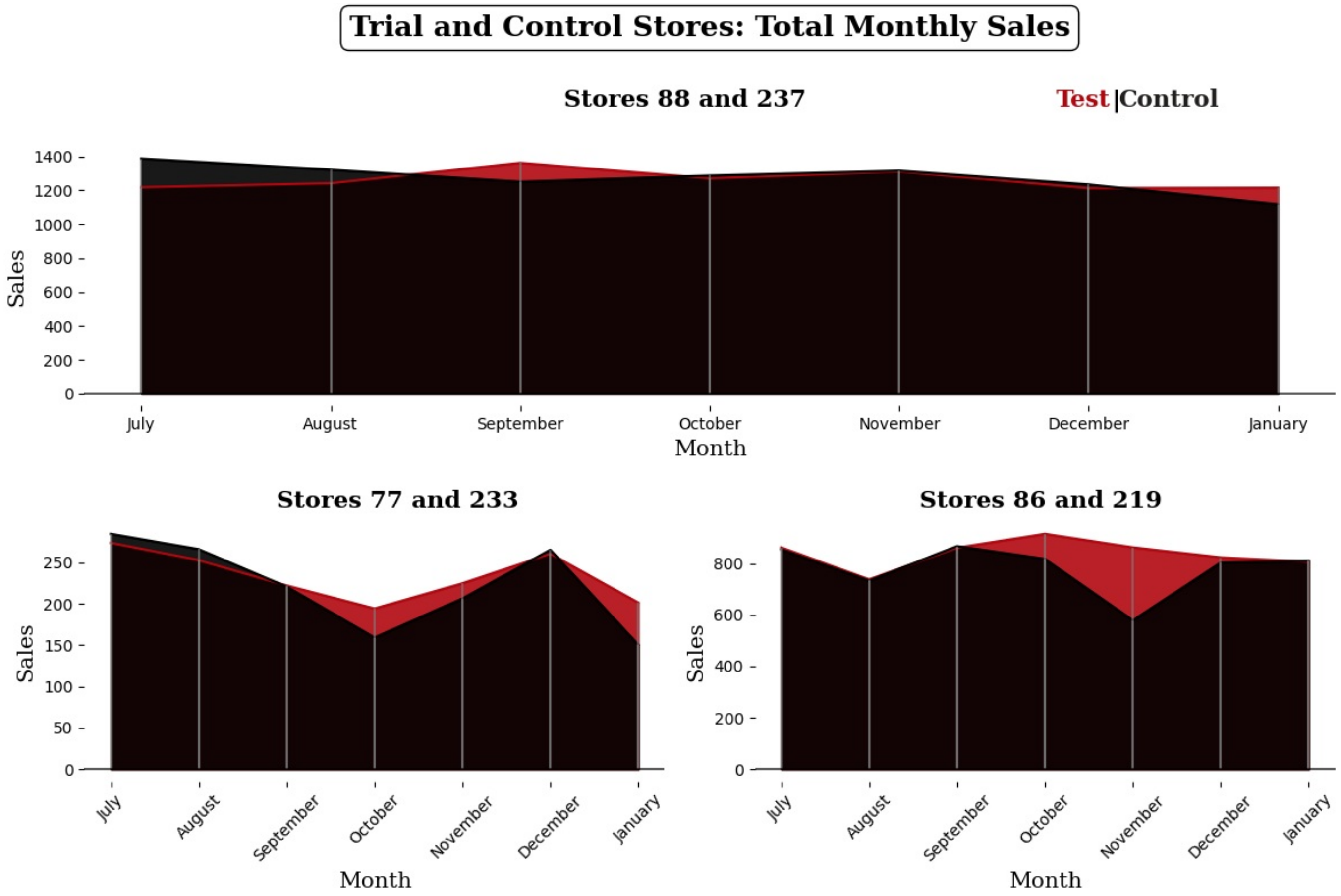
Let's do some visualisations of month-to-month metrics of the test and control scores

Total monthly sales.

In [910…

```
# sns.lineplot(y=monthly_sales.loc[:,77], x=monthly_sales.index)
color = ["#b20710", "#221f1f"]

fig, ax = plt.subplots(2, 2, figsize=(12, 8))

# Plot
ax[1,0].plot(monthly_sales.index.month_name(), monthly_sales.loc[:,77], color=color[0], label='Test Store')
ax[1,0].fill_between(monthly_sales.index.month_name(), 0, monthly_sales.loc[:,77], color=color[0], alpha=0.9)
ax[1,0].plot(monthly_sales.index.month_name(), monthly_sales.loc[:,233], color='black', label='Control Store')
ax[1,0].fill_between(monthly_sales.index.month_name(), 0, monthly_sales.loc[:,233], color='black', alpha=0.9)
# Axis labels
ax[1,0].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,0].set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,0].set_xticks(monthly_sales.index.month_name())
ax[1,0].set_xticklabels(monthly_sales.index.month_name(), rotation=45)
for date in monthly_sales.index:
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 77], colors='grey', alpha=0.6)
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 233], colors='grey', alpha=0.6)

# Plot
ax[1,1].plot(monthly_sales.index.month_name(), monthly_sales.loc[:,86], color=color[0], label='Test Store')
ax[1,1].fill_between(monthly_sales.index.month_name(), 0, monthly_sales.loc[:,86], color=color[0], alpha=0.9)
ax[1,1].plot(monthly_sales.index.month_name(), monthly_sales.loc[:,219], color='black', label='Control Store')
ax[1,1].fill_between(monthly_sales.index.month_name(), 0, monthly_sales.loc[:,219], color='black', alpha=0.9)
# Axis labels
ax[1,1].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,1].set_ylabel('Sales', fontsize=14, fontfamily='serif')
```

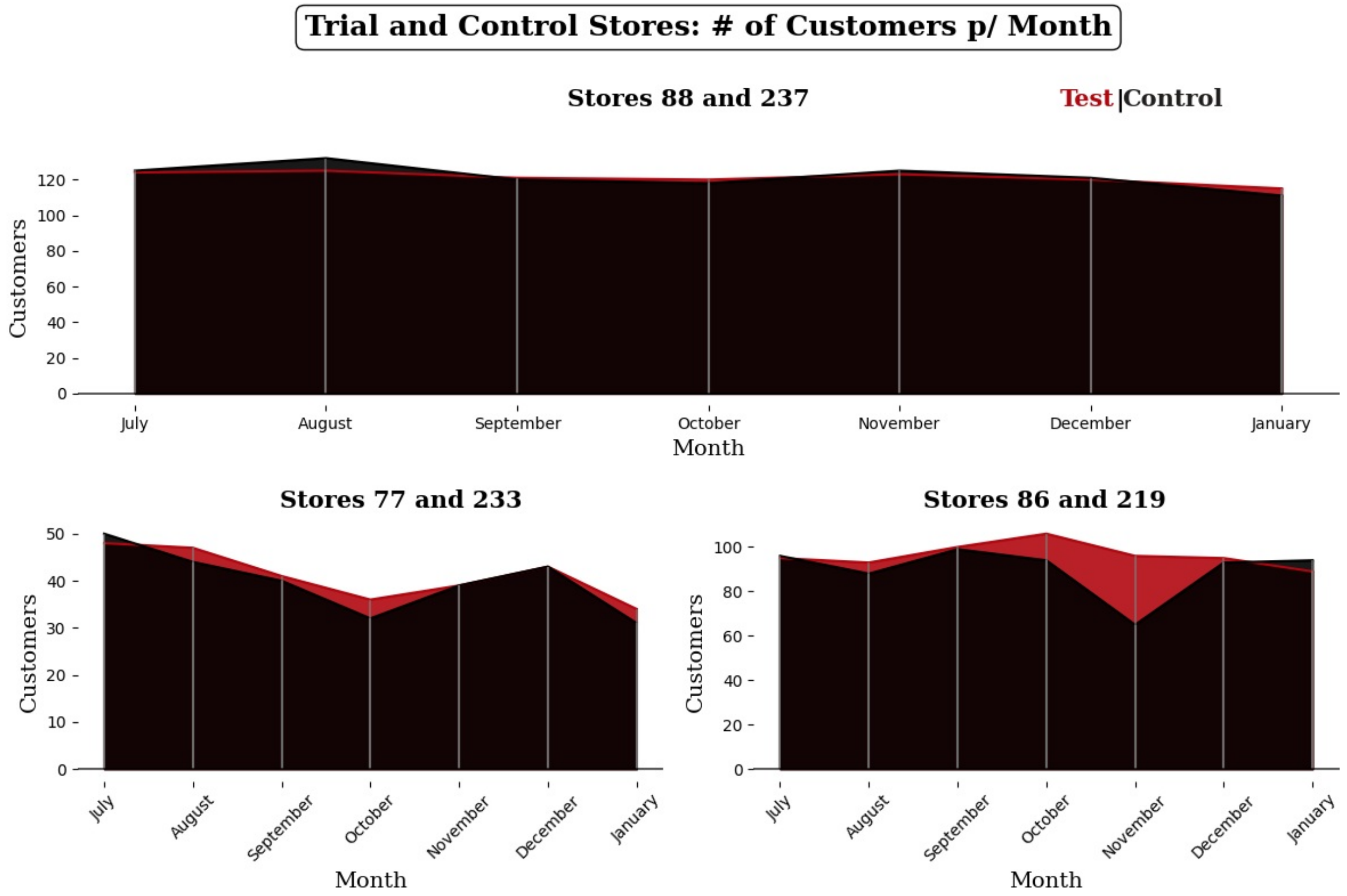
```
# Fix x-ticks
ax[1,1].set_xticks(monthly_sales.index.month_name())
ax[1,1].set_xticklabels(monthly_sales.index.month_name(), rotation=45)
for date in monthly_sales.index:
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 86], colors='grey', alpha=0.6)
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 219], colors='grey', alpha=0.6)

# # Add gridlines
# for i in range(2):
#     ax[1,i].grid(axis='y', color='gray', alpha=.7)
# Horizontal line on x-axis
for i in range(2):
    ax[1,i].yaxis.tick_left()
    ax[1,i].axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)
# Remove spines
for i in range(2):
    for s in ['top', 'right','bottom','left']:
        ax[1,i].spines[s].set_visible(False)





# Remove unnecessary subplots
fig.delaxes(ax[0, 1])  # Remove empty subplot
fig.delaxes(ax[0, 0])  # Remove empty subplot

# Add a big top plot
ax_top = fig.add_subplot(2, 1, 1)
# Plot
ax_top.plot(monthly_sales.index.month_name(), monthly_sales.loc[:,88], color=color[0], label='Test Store')
ax_top.fill_between(monthly_sales.index.month_name(), 0, monthly_sales.loc[:,88], color=color[0], alpha=0.9)
ax_top.plot(monthly_sales.index.month_name(), monthly_sales.loc[:,237], color='black', label='Test Store')
ax_top.fill_between(monthly_sales.index.month_name(), 0, monthly_sales.loc[:,237], color='black', alpha=0.9)
for date in monthly_sales.index:
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 88], colors='grey', alpha=0.6)
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 237], colors='grey', alpha=0.6)

# Plot title
ax_top.set_title('Trial and Control Stores: Total Monthly Sales', pad=70, fontsize=18, fontfamily='serif', fontw
                 bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3'))
# Axis labels
ax_top.set_xlabel('Month', fontsize=14, fontfamily='serif')
ax_top.set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax_top.set_xticks(monthly_sales.index.month_name())
ax_top.set_xticklabels(monthly_sales.index.month_name(), rotation=0)
# # Add gridlines
# ax_top.grid(axis='y', color='gray', alpha=.7)
# Horizontal line on x-axis
ax_top.axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)

# Remove spines
for s in ['top', 'right','bottom','left']:
    ax_top.spines[s].set_visible(False)


# Add titles
fig.text(.42, .88, 'Stores 88 and 237', fontsize=15, fontweight='bold', fontfamily='serif')
fig.text(.21, .44, 'Stores 77 and 233', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(.68, .44, 'Stores 86 and 219', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(0.78,0.88,"Test", fontweight="bold", fontfamily='serif', fontsize=15, color='#b20710')
fig.text(0.82,0.88,"|", fontweight="bold", fontfamily='serif', fontsize=15, color='black')
fig.text(0.825,0.88,"Control", fontweight="bold", fontfamily='serif', fontsize=15, color='#221f1f')


plt.tight_layout()
plt.subplots_adjust(hspace=.45)
```



No. of Monthly Customers

In [911…

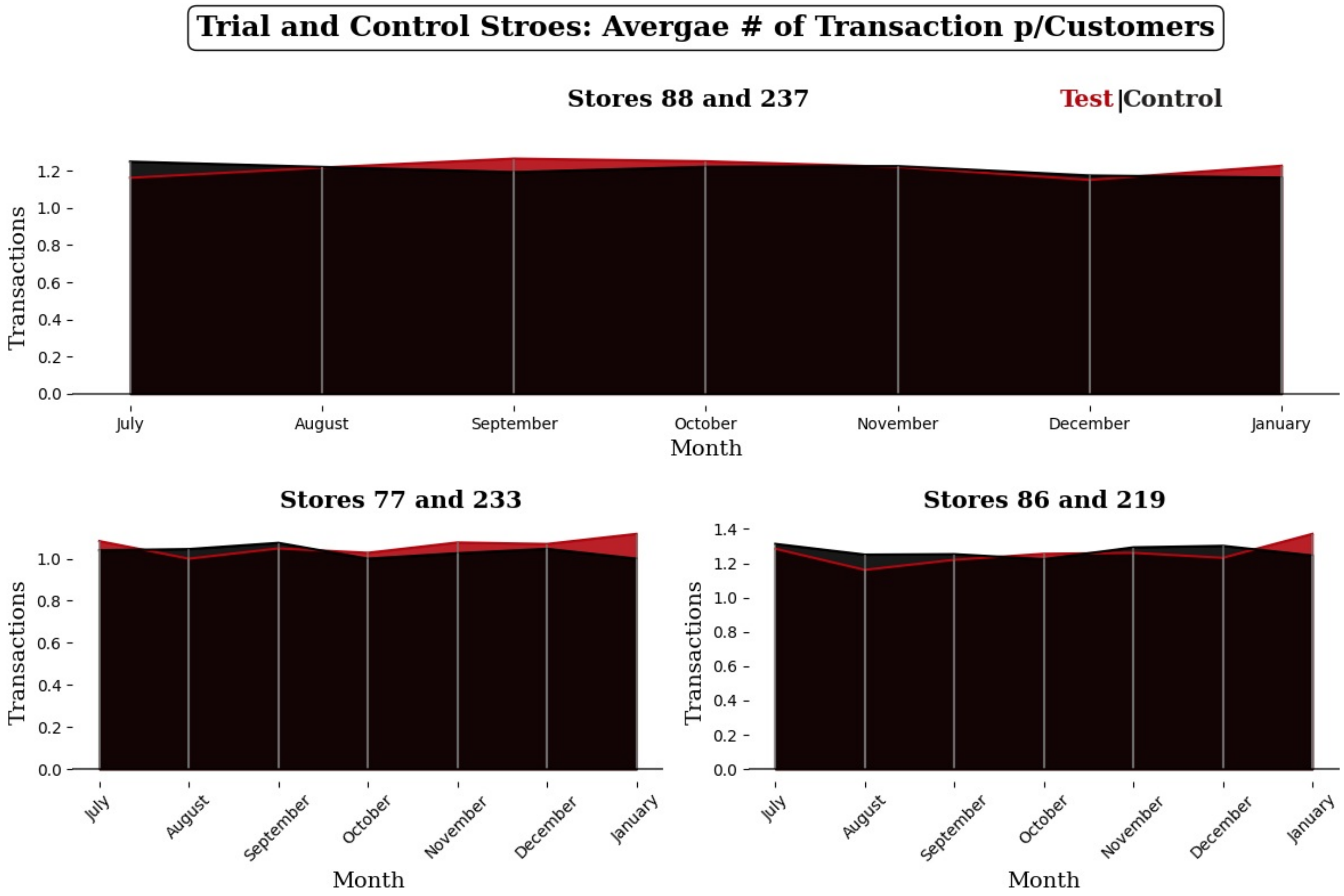
```
color = ["#b20710", "#221f1f"]

fig, ax = plt.subplots(2, 2, figsize=(12, 8))

# Plot
ax[1,0].plot(monthly_customers.index.month_name(), monthly_customers.loc[:,77], color=color[0], label='Test Sto
ax[1,0].fill_between(monthly_customers.index.month_name(), 0, monthly_customers.loc[:,77], color=color[0], alpha
ax[1,0].plot(monthly_customers.index.month_name(), monthly_customers.loc[:,233], color='black', label='Control S
ax[1,0].fill_between(monthly_customers.index.month_name(), 0, monthly_customers.loc[:,233], color='black', alpha
# Axis labels
ax[1,0].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,0].set_ylabel('Customers', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,0].set_xticks(monthly_customers.index.month_name())
ax[1,0].set_xticklabels(monthly_customers.index.month_name(), rotation=45)
for date in monthly_sales.index:
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 77], colors='grey', alpha=0.6)
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 233], colors='grey', alpha=0.6


# Plot
ax[1,1].plot(monthly_customers.index.month_name(), monthly_customers.loc[:,86], color=color[0], label='Test Sto
ax[1,1].fill_between(monthly_customers.index.month_name(), 0, monthly_customers.loc[:,86], color=color[0], alpha
ax[1,1].plot(monthly_customers.index.month_name(), monthly_customers.loc[:,219], color='black', label='Control S
ax[1,1].fill_between(monthly_customers.index.month_name(), 0, monthly_customers.loc[:,219], color='black', alpha
# Axis labels
ax[1,1].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,1].set_ylabel('Customers', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,1].set_xticks(monthly_customers.index.month_name())
ax[1,1].set_xticklabels(monthly_customers.index.month_name(), rotation=45)
for date in monthly_sales.index:
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 86], colors='grey', alpha=0.6)
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 219], colors='grey', alpha=0.6

# # Add gridlines
# for i in range(2):
#     ax[1,i].grid(axis='y', color='gray', alpha=.7)
# Horizontal line on x-axis
for i in range(2):
    ax[1,i].yaxis.tick_left()
    ax[1,i].axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)
# Remove spines
for i in range(2):
    for s in ['top', 'right','bottom','left']:
        ax[1,i].spines[s].set_visible(False)



# Remove unnecessary subplots
fig.delaxes(ax[0, 1])  # Remove empty subplot
fig.delaxes(ax[0, 0])  # Remove empty subplot

# Add a big top plot
ax_top = fig.add_subplot(2, 1, 1)
# Plot
ax_top.plot(monthly_customers.index.month_name(), monthly_customers.loc[:,88], color=color[0], label='Test Stor
ax_top.fill_between(monthly_customers.index.month_name(), 0, monthly_customers.loc[:,88], color=color[0], alpha=
ax_top.plot(monthly_customers.index.month_name(), monthly_customers.loc[:,237], color='black', label='Test Stor
ax_top.fill_between(monthly_customers.index.month_name(), 0, monthly_customers.loc[:,237], color='black', alpha=
for date in monthly_sales.index:
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 88], colors='grey', alpha=0.6)
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 237], colors='grey', alpha=0.6)

# Plot title
ax_top.set_title('Trial and Control Stores: # of Customers p/ Month', pad=70, fontsize=18, fontfamily='serif', 
                 bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3'))
# Axis labels
ax_top.set_xlabel('Month', fontsize=14, fontfamily='serif')
ax_top.set_ylabel('Customers', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax_top.set_xticks(monthly_customers.index.month_name())
ax_top.set_xticklabels(monthly_customers.index.month_name(), rotation=0)
# # Add gridlines
# ax_top.grid(axis='y', color='gray', alpha=.7)
# Horizontal line on x-axis
ax_top.axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)

# Remove spines
for s in ['top', 'right','bottom','left']:
    ax_top.spines[s].set_visible(False)


# Add titles
fig.text(.42, .88, 'Stores 88 and 237', fontsize=15, fontweight='bold', fontfamily='serif')
fig.text(.21, .44, 'Stores 77 and 233', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(.68, .44, 'Stores 86 and 219', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(0.78,0.88,"Test", fontweight="bold", fontfamily='serif', fontsize=15, color='#b20710')
fig.text(0.82,0.88,"|", fontweight="bold", fontfamily='serif', fontsize=15, color='black')
fig.text(0.825,0.88,"Control", fontweight="bold", fontfamily='serif', fontsize=15, color='#221f1f')


plt.tight_layout()
plt.subplots_adjust(hspace=.45)
```



Average Transactions p/ Customer

In [912…

```
color = ["#b20710", "#221f1f"]

fig, ax = plt.subplots(2, 2, figsize=(12, 8))

# Plot
ax[1,0].plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,77], color=c
ax[1,0].fill_between(avg_transactions_per_customer.index.month_name(), 0, avg_transactions_per_customer.loc[:,7
ax[1,0].plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,233], color=
ax[1,0].fill_between(avg_transactions_per_customer.index.month_name(), 0, avg_transactions_per_customer.loc[:,2
# Axis labels
ax[1,0].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,0].set_ylabel('Transactions', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,0].set_xticks(avg_transactions_per_customer.index.month_name())
ax[1,0].set_xticklabels(avg_transactions_per_customer.index.month_name(), rotation=45)
for date in monthly_sales.index:
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 77], colors='grey'
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 233], colors='grey

# Plot
ax[1,1].plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,86], color=c
ax[1,1].fill_between(avg_transactions_per_customer.index.month_name(), 0, avg_transactions_per_customer.loc[:,8
ax[1,1].plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,219], color=
ax[1,1].fill_between(avg_transactions_per_customer.index.month_name(), 0, avg_transactions_per_customer.loc[:,2
# Axis labels
ax[1,1].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,1].set_ylabel('Transactions', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,1].set_xticks(avg_transactions_per_customer.index.month_name())
ax[1,1].set_xticklabels(avg_transactions_per_customer.index.month_name(), rotation=45)
for date in monthly_sales.index:
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 86], colors='grey'
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 219], colors='grey

# Add gridlines
# for i in range(2):
#     ax[1,i].grid(axis='y', color='gray', alpha=.7)
# Horizontal line on x-axis
for i in range(2):
    ax[1,i].yaxis.tick_left()
    ax[1,i].axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)
# Remove spines
for i in range(2):
    for s in ['top', 'right','bottom','left']:
        ax[1,i].spines[s].set_visible(False)




# Remove unnecessary subplots
fig.delaxes(ax[0, 1])  # Remove empty subplot
fig.delaxes(ax[0, 0])  # Remove empty subplot

# Add a big top plot
ax_top = fig.add_subplot(2, 1, 1)
# Plot
ax_top.plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,88], color=co
ax_top.fill_between(avg_transactions_per_customer.index.month_name(), 0, avg_transactions_per_customer.loc[:,88
ax_top.plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,237], color='
ax_top.fill_between(avg_transactions_per_customer.index.month_name(), 0, avg_transactions_per_customer.loc[:,23
for date in monthly_sales.index:
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 88], colors='grey',
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 237], colors='grey'


# Plot title
ax_top.set_title('Trial and Control Stroes: Avergae # of Transaction p/Customers', pad=70, fontsize=18, fontfam
                 bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3'))
# Axis labels
ax_top.set_xlabel('Month', fontsize=14, fontfamily='serif')
ax_top.set_ylabel('Transactions', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax_top.set_xticks(avg_transactions_per_customer.index.month_name())
ax_top.set_xticklabels(avg_transactions_per_customer.index.month_name(), rotation=0)
# # Add gridlines
# ax_top.grid(axis='y', color='gray', alpha=.7)
# Horizontal line on x-axis
ax_top.axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)

# Remove spines
for s in ['top', 'right','bottom','left']:
    ax_top.spines[s].set_visible(False)


# Add titles
fig.text(.42, .88, 'Stores 88 and 237', fontsize=15, fontweight='bold', fontfamily='serif')
fig.text(.21, .44, 'Stores 77 and 233', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(.68, .44, 'Stores 86 and 219', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(0.78,0.88,"Test", fontweight="bold", fontfamily='serif', fontsize=15, color='#b20710')
fig.text(0.82,0.88,"|", fontweight="bold", fontfamily='serif', fontsize=15, color='black')
fig.text(0.825,0.88,"Control", fontweight="bold", fontfamily='serif', fontsize=15, color='#221f1f')


plt.tight_layout()
plt.subplots_adjust(hspace=.45)
```



Trial and Control Stroes: Avergae # of Transaction p/Customers

Stores 88 and 237

Stores 77 and 233

Stores 86 and 219

Check the trial period total sales

In [913…

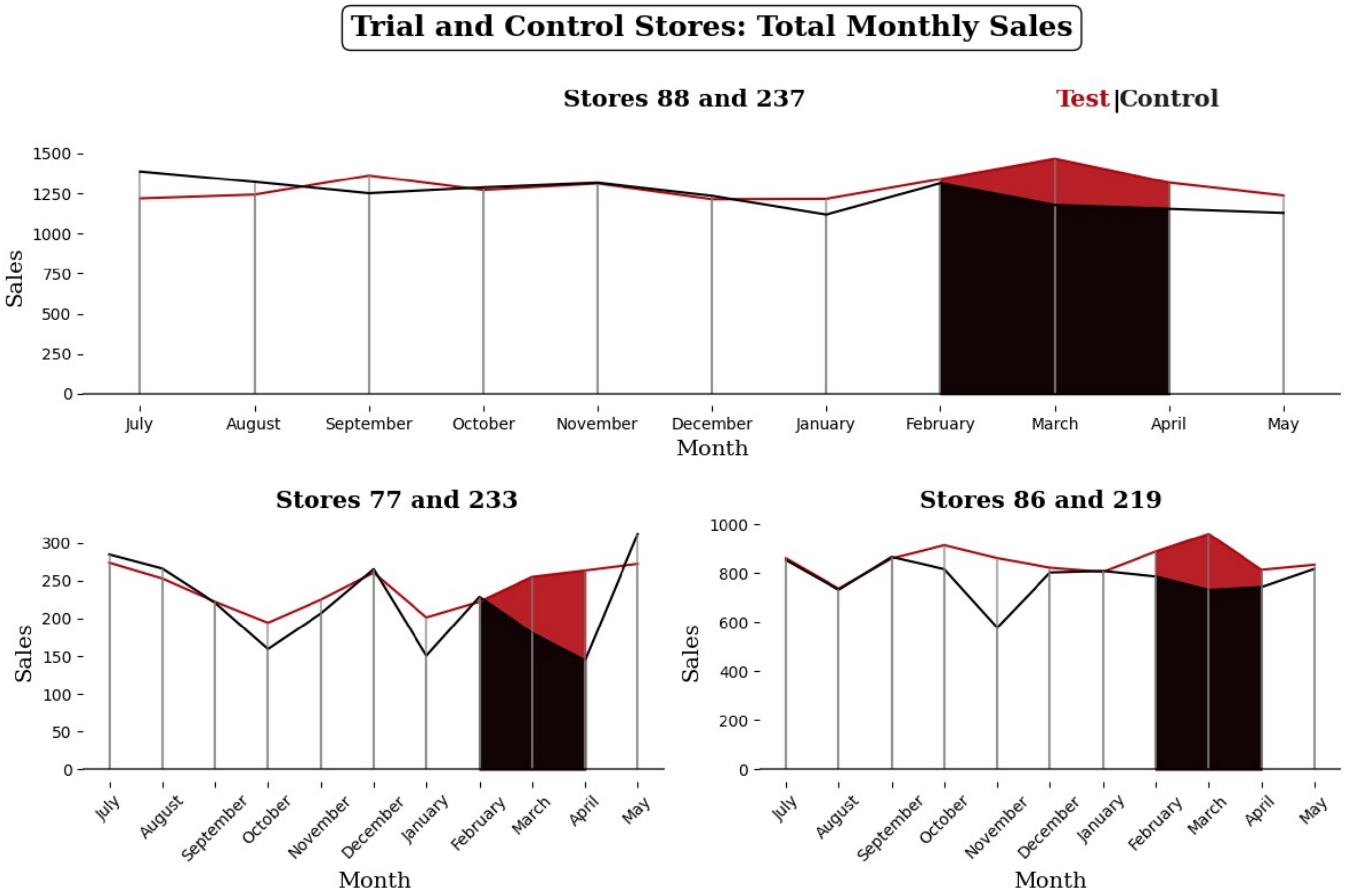
```
start_date = '2018-07-01'
end_date = '2019-05-31'

# Filter the trial date
filtered_df = df[(df['DATE'] >= start_date) & (df['DATE'] <= end_date)]
# Get TOTAL SALES per store per month
monthly_sales = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')])['TOT_SALES'].sum().unstac
monthly_sales = monthly_sales.T

# NO. of CUSTOMERS P/MONTH.
monthly_customers = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')])['LYLTY_CARD_NBR'].nun
monthly_customers = monthly_customers.T

# Average number of transactions per customer per month.
monthly_transactions = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')]).size().unstack(fil
monthly_customers = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')])['LYLTY_CARD_NBR'].nun
avg_transactions_per_customer = monthly_transactions / monthly_customers  # Calculate the average number of tra
avg_transactions_per_customer = avg_transactions_per_customer.T


# NO. of CUSTOMERS P/MONTH.
monthly_customers = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')])['LYLTY_CARD_NBR'].nun
monthly_customers = monthly_customers.T
```

Total Monthly Sales

In [914…

```
color = ["#b20710", "#221f1f"]

fig, ax = plt.subplots(2, 2, figsize=(12, 8))

# Plot
ax[1,0].plot(monthly_sales.index.month_name(), monthly_sales.loc[:,77], color=color[0], label='Test Store')
# Fill only between start and end date
ax[1,0].fill_between(monthly_sales.index.month_name(), monthly_sales.loc[:,77], 0, where=(monthly_sales.index>'
ax[1,0].plot(monthly_sales.index.month_name(), monthly_sales.loc[:,233], color='black', label='Control Store')
ax[1,0].fill_between(monthly_sales.index.month_name(), monthly_sales.loc[:,233], 0, where=(monthly_sales.index>
# Axis labels
ax[1,0].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,0].set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,0].set_xticks(monthly_sales.index.month_name())
ax[1,0].set_xticklabels(monthly_sales.index.month_name(), rotation=45)
```

```
for date in monthly_sales.index:
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 77], colors='grey', alpha=0.6)
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 233], colors='grey', alpha=0.6)


# Plot
ax[1,1].plot(monthly_sales.index.month_name(), monthly_sales.loc[:,86], color=color[0], label='Test Store')
ax[1,1].fill_between(monthly_sales.index.month_name(), monthly_sales.loc[:,86], 0, where=(monthly_sales.index>'2
ax[1,1].plot(monthly_sales.index.month_name(), monthly_sales.loc[:,219], color='black', label='Control Store')
ax[1,1].fill_between(monthly_sales.index.month_name(), monthly_sales.loc[:,219], 0, where=(monthly_sales.index>
# Axis labels
ax[1,1].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,1].set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,1].set_xticks(monthly_sales.index.month_name())
ax[1,1].set_xticklabels(monthly_sales.index.month_name(), rotation=45)


# Add gridlines
# for i in range(2):
    # ax[1,i].grid(axis='y', color='gray', alpha=.7)

for date in monthly_sales.index:
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 86], colors='grey', alpha=0.6)
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 219], colors='grey', alpha=0.6)
# Horizontal line on x-axis
for i in range(2):
    ax[1,i].yaxis.tick_left()
    ax[1,i].axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)
# Remove spines
for i in range(2):
    for s in ['top', 'right','bottom','left']:
        ax[1,i].spines[s].set_visible(False)




# Remove unnecessary subplots
fig.delaxes(ax[0, 1])  # Remove empty subplot
fig.delaxes(ax[0, 0])  # Remove empty subplot

# Add a big top plot
ax_top = fig.add_subplot(2, 1, 1)
# Plot
ax_top.plot(monthly_sales.index.month_name(), monthly_sales.loc[:,88], color=color[0], label='Test Store')
ax_top.fill_between(monthly_sales.index.month_name(), monthly_sales.loc[:,88], 0, where=(monthly_sales.index>'20
for date in monthly_sales.index:
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 88], colors='grey', alpha=0.6)
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=monthly_sales.loc[date, 237], colors='grey', alpha=0.6)
ax_top.plot(monthly_sales.index.month_name(), monthly_sales.loc[:,237], color='black', label='Test Store')
ax_top.fill_between(monthly_sales.index.month_name(), monthly_sales.loc[:,237], 0, where=(monthly_sales.index>'2


# Plot title
ax_top.set_title('Trial and Control Stores: Total Monthly Sales', pad=70, fontsize=18, fontfamily='serif', fontw
                 bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3'))
# Axis labels
ax_top.set_xlabel('Month', fontsize=14, fontfamily='serif')
ax_top.set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax_top.set_xticks(monthly_sales.index.month_name())
ax_top.set_xticklabels(monthly_sales.index.month_name(), rotation=0)
# Add gridlines
# ax_top.grid(axis='y', color='gray', alpha=.7)
# Horizontal line on x-axis
ax_top.axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)

# Remove spines
for s in ['top', 'right','bottom','left']:
    ax_top.spines[s].set_visible(False)


# Add titles
fig.text(.42, .88, 'Stores 88 and 237', fontsize=15, fontweight='bold', fontfamily='serif')
fig.text(.21, .44, 'Stores 77 and 233', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(.68, .44, 'Stores 86 and 219', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(0.78,0.88,"Test", fontweight="bold", fontfamily='serif', fontsize=15, color='#b20710')
fig.text(0.82,0.88,"|", fontweight="bold", fontfamily='serif', fontsize=15, color='black')
fig.text(0.825,0.88,"Control", fontweight="bold", fontfamily='serif', fontsize=15, color='#221f1f')


plt.tight_layout()
plt.subplots_adjust(hspace=.45)
```



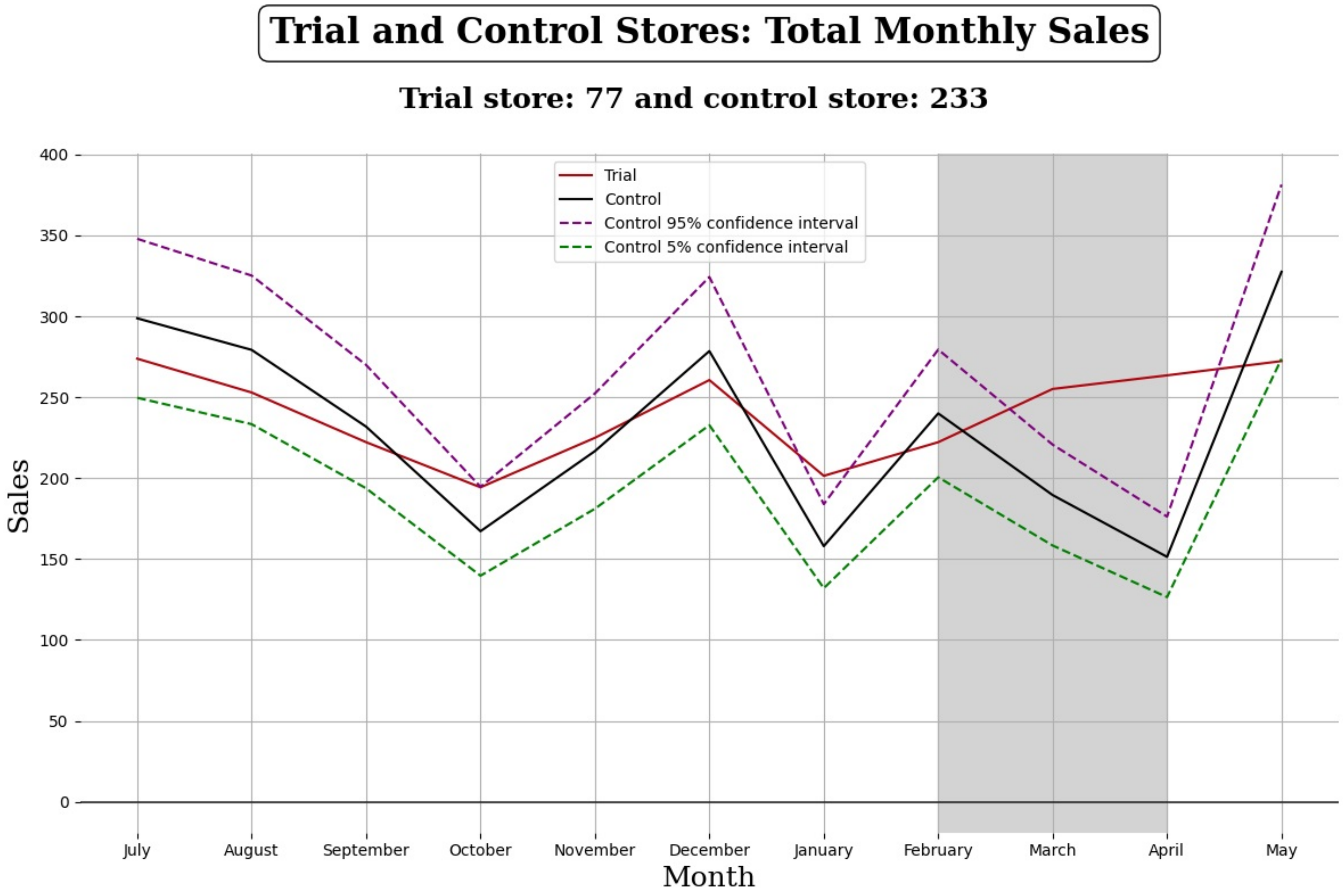
Define function to check significance difference during trial period and plot.

In [ ]:

```
# 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

def significance_check(trial_store, control_store, pre_trial_cutoff = '2019-02', trial_end = '2019-04-30'):
    # 1. Create YEARMONTH column for grouping
    df['YEARMONTH'] = df['DATE'].dt.to_period('M')

    # 2. Aggregate total sales per month per store
    monthly_sales = df.groupby(['YEARMONTH', 'STORE_NBR'])['TOT_SALES'].sum().reset_index()

    # 3. Compute scaling factor from pre-trial months
    trial_pre_total = monthly_sales[(monthly_sales['STORE_NBR'] == trial_store) &
                                    (monthly_sales['YEARMONTH'] < pre_trial_cutoff)]['TOT_SALES'].sum()

    control_pre_total = monthly_sales[(monthly_sales['STORE_NBR'] == control_store) &
                                    (monthly_sales['YEARMONTH'] < pre_trial_cutoff)]['TOT_SALES'].sum()

    scaling_factor = trial_pre_total / control_pre_total

    # 4. Apply scaling factor to control store
    monthly_sales['scaled_sales'] = None
    monthly_sales.loc[monthly_sales['STORE_NBR'] == control_store, 'scaled_sales'] = \
        monthly_sales.loc[monthly_sales['STORE_NBR'] == control_store, 'TOT_SALES'] * scaling_factor

    # 5. Pivot for comparison (use only relevant stores)
    comparison_df = monthly_sales[monthly_sales['STORE_NBR'].isin([trial_store, control_store])]
    pivoted = comparison_df.pivot(index='YEARMONTH', columns='STORE_NBR', values=['TOT_SALES', 'scaled_sales'])

    # 6. Flatten column names
    pivoted.columns = [f"{metric}_{store}" for metric, store in pivoted.columns]
    pivoted = pivoted.reset_index()

    # 7. Calculate percentage difference from pre-trial months
    pivoted['percentage_diff'] = np.abs(pivoted[f'TOT_SALES_{trial_store}.0'] - pivoted[f'scaled_sales_{control_

    # 8. Std deviation from pre-trial period
    pre_trial = pivoted[pivoted['YEARMONTH'] < pre_trial_cutoff]
    std_dev = pre_trial['percentage_diff'].std()
    df_degrees = len(pre_trial) - 1


    # 9. Compute t-values during trial period (e.g. Feb–Apr or Mar–Jun)
    trial_period = pivoted[(pivoted['YEARMONTH'] >= pre_trial_cutoff) & (pivoted['YEARMONTH'] <= trial_end)].co
    trial_period['t_value'] = trial_period['percentage_diff'] / std_dev
    trial_period

    # 10. Critical value for 95% confidence
    t_critical = stats.t.ppf(0.95, df_degrees)

    # 11. Significance check
    trial_period['is_significant'] = trial_period['t_value'] > t_critical


    ###########################################################################################################


    # Now Plot
    start_date = '2018-07-01'
    end_date = '2019-05-31'

    # Filter the trial date
    filtered_df = df[(df['DATE'] >= start_date) & (df['DATE'] <= end_date)]
    # Get TOTAL SALES per store per month
    monthly_sales = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')])['TOT_SALES'].sum().uns
    monthly_sales = monthly_sales.T

    color = ["#b20710", "#221f1f"]

    fig = plt.figure(figsize=(12, 8))
    ax = plt.gca()

    # Plot
    ax.plot(monthly_sales.index.month_name(), monthly_sales.loc[:,trial_store], color=color[0], label='Trial')
    ax.plot(monthly_sales.index.month_name(), scaling_factor * monthly_sales.loc[:,control_store], color='black
    ax.plot(monthly_sales.index.month_name(), scaling_factor * monthly_sales.loc[:,control_store] + scaling_fac
            color='purple', linestyle='--', label=r'Control 95% confidence interval')
    ax.plot(monthly_sales.index.month_name(), scaling_factor * monthly_sales.loc[:,control_store] - scaling_fac
            color='green', linestyle='--', label=r'Control 5% confidence interval')
    # ax_top.fill_between(monthly_sales.index.month_name(), monthly_sales.loc[:,237], 0, where=(monthly_sales.in
    ax.axvspan('February', 'April', color='lightgray')

    # Plot title
    ax.set_title('Trial and Control Stores: Total Monthly Sales', pad=70, fontsize=22, fontfamily='serif', fontw
                    bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3'))
    # Axis labels
    ax.set_xlabel('Month', fontsize=18, fontfamily='serif')
    ax.set_ylabel('Sales', fontsize=18, fontfamily='serif')
    # Fix x-ticks
    ax.set_xticks(monthly_sales.index.month_name())
    ax.set_xticklabels(monthly_sales.index.month_name(), rotation=0)
    # Horizontal line on x-axis
    ax.axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)
    # Gridlines
    ax.grid()
    fig.text(.3, .88, f'Trial store: {trial_store} and control store: {control_store}', fontsize=18, fontweight=

    # Remove spines
    for s in ['top', 'right','bottom','left']:
        ax.spines[s].set_visible(False)

    plt.tight_layout()
    ax.legend(loc='best')
    plt.show()

    return(trial_period)
```

In [924…

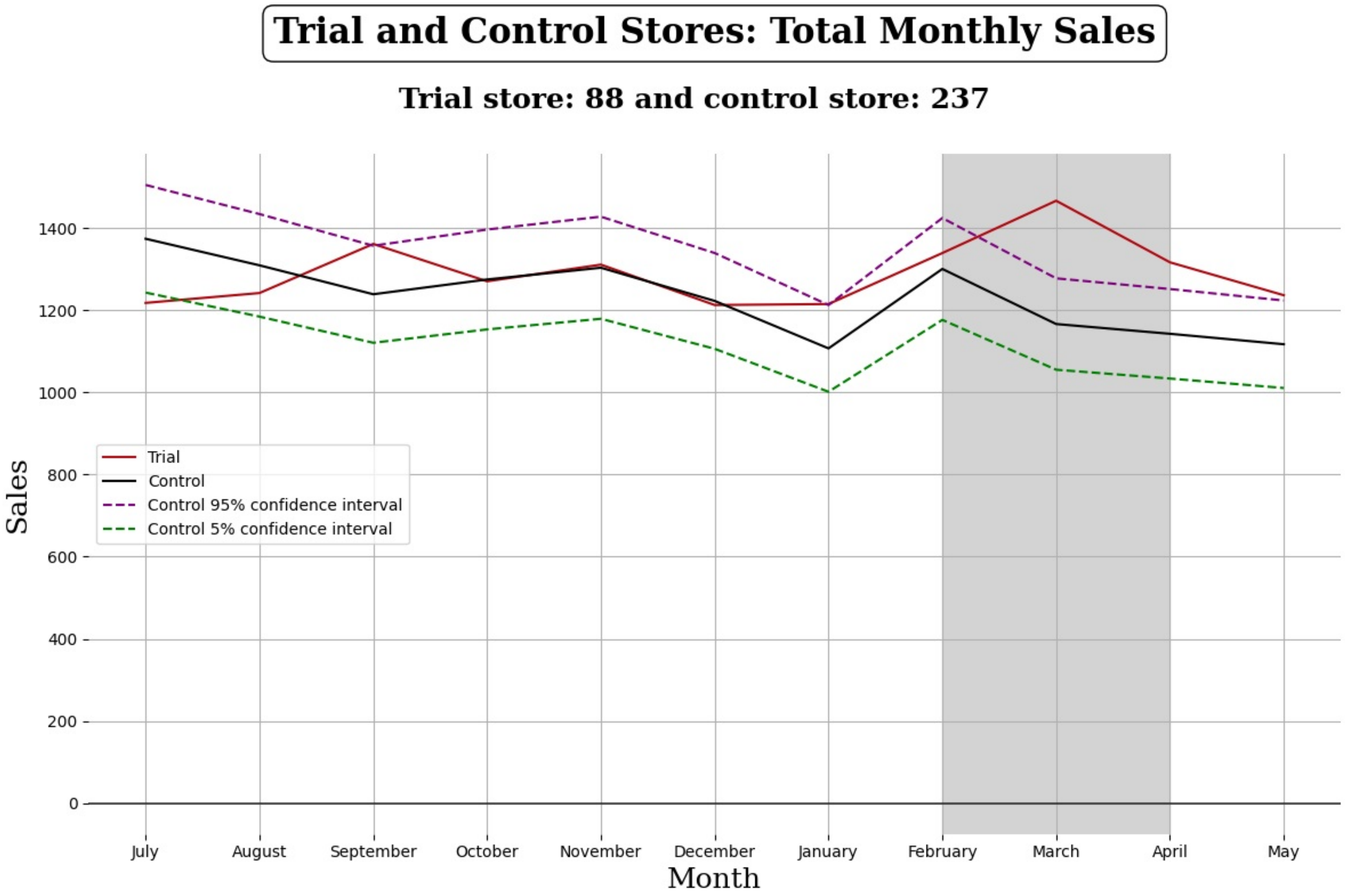
```
print(significance_check(77,233))
```



```
  YEARMONTH TOT_SALES_77.0 TOT_SALES_233.0 scaled_sales_77.0  \
7   2019-02          222.2           228.7              None   
8   2019-03          255.1           180.6              None   
9   2019-04          263.5           144.2              None   

  scaled_sales_233.0 percentage_diff   t_value  is_significant  
7         239.992191        0.074137  0.876099           False  
8         189.517227        0.346052  4.089422            True  
9         151.319956        0.741343  8.760729            True  
```

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.

```
In [ ]: print(significance_check(88,237))
```

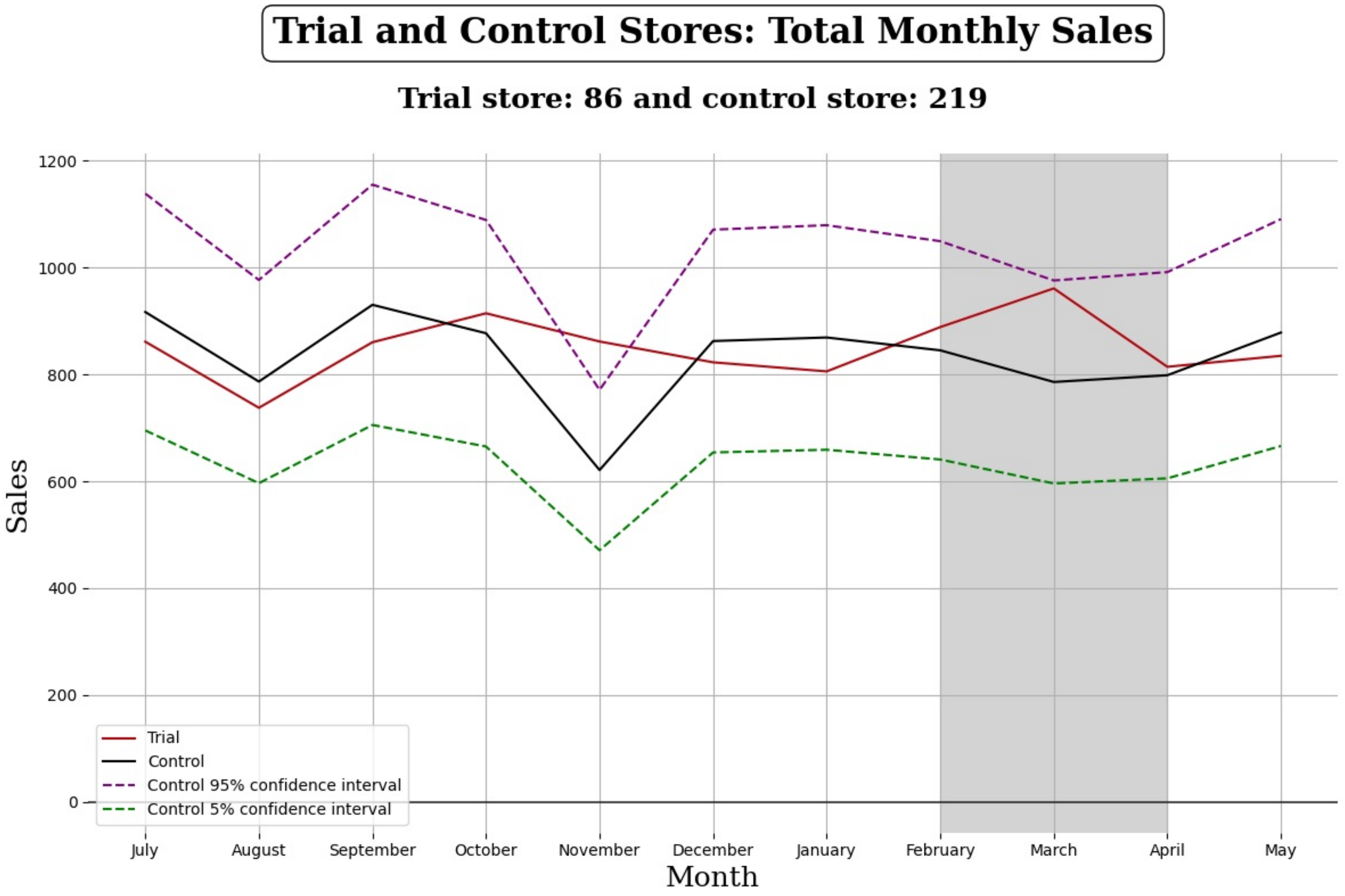


```
  YEARMONTH TOT_SALES_88.0 TOT_SALES_237.0 scaled_sales_88.0  \
7   2019-02         1339.6          1313.0              None
8   2019-03         1467.0          1177.6              None
9   2019-04         1317.0          1153.6              None

  scaled_sales_237.0 percentage_diff   t_value  is_significant
7        1300.879003        0.029765  0.606487           False
8        1166.728952        0.257361   5.24391            True
9        1142.950509        0.152281  3.102824            True
```

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 the 5% to 95% confidence interval of the control store in two of the three trial months.

In [927…

```
print(significance_check(86,219))
```



```
  YEARMONTH TOT_SALES_86.0 TOT_SALES_219.0 scaled_sales_86.0  \
7   2019-02          888.8           787.2              None   
8   2019-03          961.2           732.0              None   
9   2019-04          814.4           743.8              None   

  scaled_sales_219.0 percentage_diff   t_value  is_significant  
7         845.248739        0.051525  0.414062           False  
8         785.978249        0.222935  1.791539           False  
9          798.64839        0.019723  0.158496           False  
```

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 all three trial months.

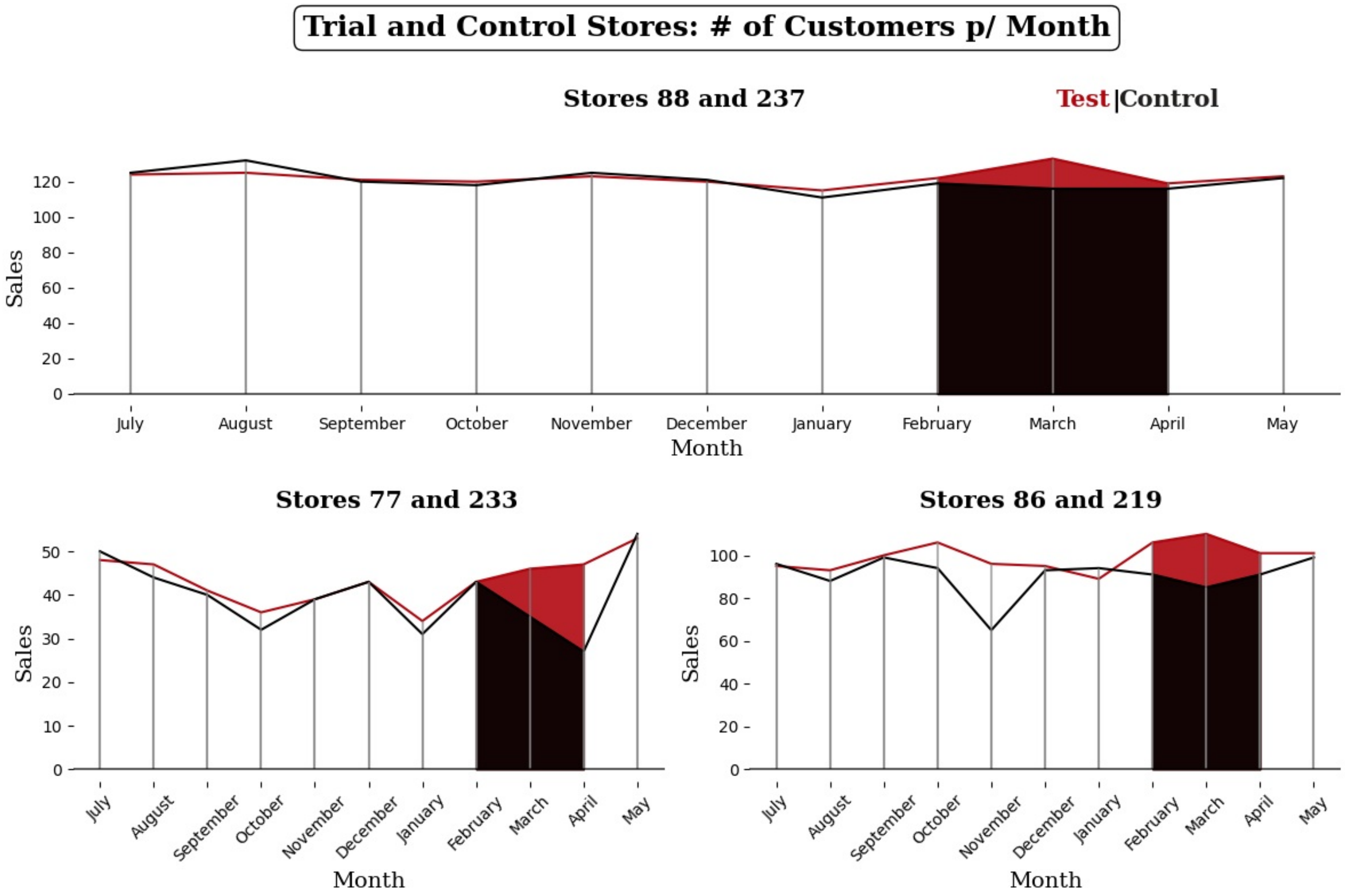
Let's Find Out Why?

No. of Monthly Customers

In [928…

```
color = ["#b20710", "#221f1f"]

fig, ax = plt.subplots(2, 2, figsize=(12, 8))

# Plot
ax[1,0].plot(monthly_customers.index.month_name(), monthly_customers.loc[:,77], color=color[0], label='Test Sto
# Fill only between start and end date
ax[1,0].fill_between(monthly_customers.index.month_name(), monthly_customers.loc[:,77], 0, where=(monthly_custor
ax[1,0].plot(monthly_customers.index.month_name(), monthly_customers.loc[:,233], color='black', label='Control S
ax[1,0].fill_between(monthly_customers.index.month_name(), monthly_customers.loc[:,233], 0, where=(monthly_custo
# Axis labels
ax[1,0].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,0].set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,0].set_xticks(monthly_customers.index.month_name())
ax[1,0].set_xticklabels(monthly_customers.index.month_name(), rotation=45)

for date in monthly_customers.index:
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 77], colors='grey', alpha=0.6)
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 233], colors='grey', alpha=0.6


# Plot
ax[1,1].plot(monthly_customers.index.month_name(), monthly_customers.loc[:,86], color=color[0], label='Test Sto
ax[1,1].fill_between(monthly_customers.index.month_name(), monthly_customers.loc[:,86], 0, where=(monthly_custor
ax[1,1].plot(monthly_customers.index.month_name(), monthly_customers.loc[:,219], color='black', label='Control S
ax[1,1].fill_between(monthly_customers.index.month_name(), monthly_customers.loc[:,219], 0, where=(monthly_custo
# Axis labels
ax[1,1].set_xlabel('Month', fontsize=14, fontfamily='serif')
```

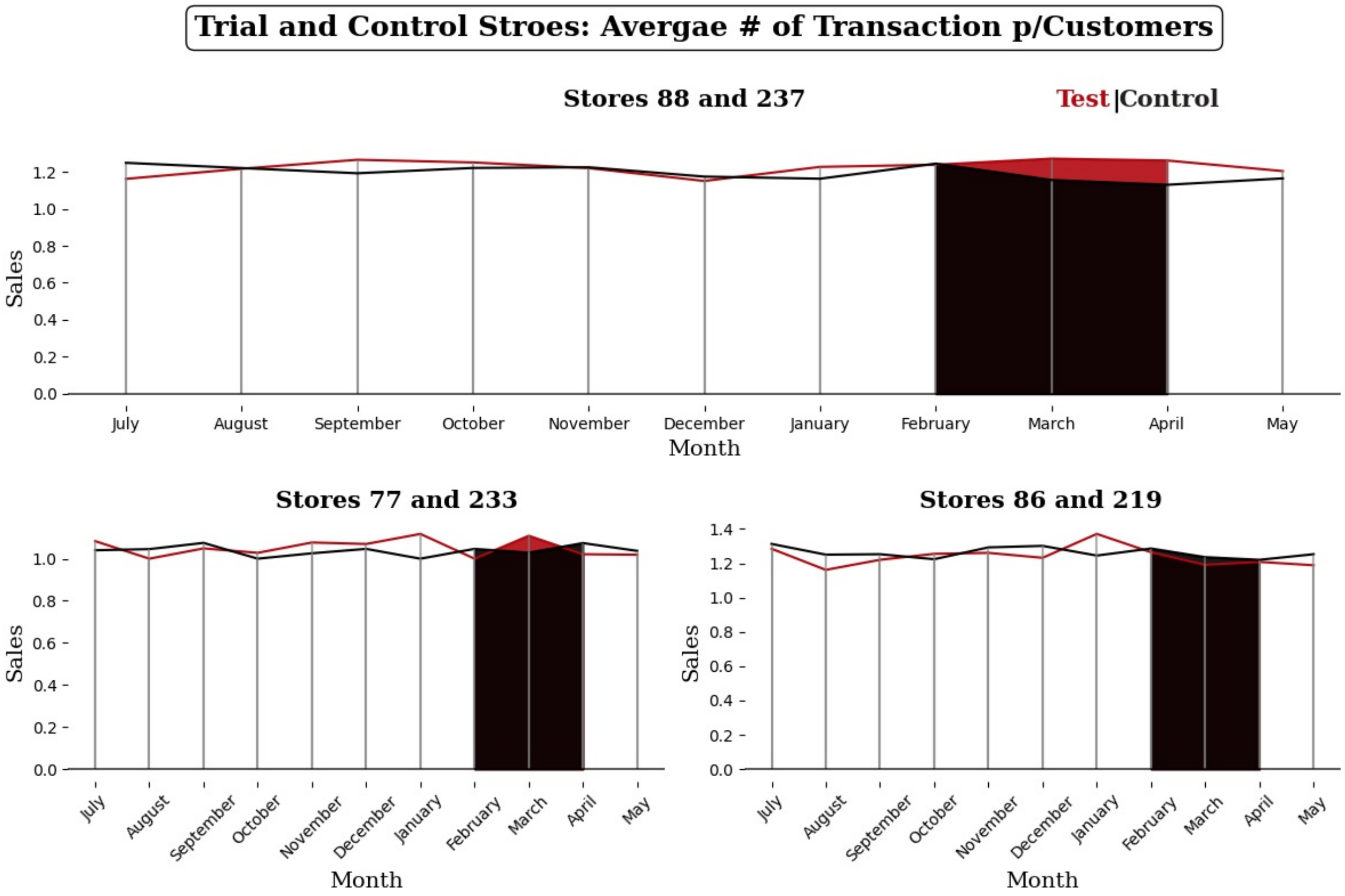
```python
ax[1,1].set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,1].set_xticks(monthly_customers.index.month_name())
ax[1,1].set_xticklabels(monthly_customers.index.month_name(), rotation=45)


# Add gridlines
# for i in range(2):
    # ax[1,i].grid(axis='y', color='gray', alpha=.7)

for date in monthly_customers.index:
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 86], colors='grey', alpha=0.6)
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 219], colors='grey', alpha=0.6
# Horizontal line on x-axis
for i in range(2):
    ax[1,i].yaxis.tick_left()
    ax[1,i].axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)
# Remove spines
for i in range(2):
    for s in ['top', 'right','bottom','left']:
        ax[1,i].spines[s].set_visible(False)




# Remove unnecessary subplots
fig.delaxes(ax[0, 1])  # Remove empty subplot
fig.delaxes(ax[0, 0])  # Remove empty subplot

# Add a big top plot
ax_top = fig.add_subplot(2, 1, 1)
# Plot
ax_top.plot(monthly_customers.index.month_name(), monthly_customers.loc[:,88], color=color[0], label='Test Store
ax_top.fill_between(monthly_customers.index.month_name(), monthly_customers.loc[:,88], 0, where=(monthly_custome
for date in monthly_customers.index:
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 88], colors='grey', alpha=0.6)
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=monthly_customers.loc[date, 237], colors='grey', alpha=0.6)
ax_top.plot(monthly_customers.index.month_name(), monthly_customers.loc[:,237], color='black', label='Test Store
ax_top.fill_between(monthly_customers.index.month_name(), monthly_customers.loc[:,237], 0, where=(monthly_custon


# Plot title
ax_top.set_title('Trial and Control Stores: # of Customers p/ Month', pad=70, fontsize=18, fontfamily='serif', 
                 bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3'))
# Axis labels
ax_top.set_xlabel('Month', fontsize=14, fontfamily='serif')
ax_top.set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax_top.set_xticks(monthly_customers.index.month_name())
ax_top.set_xticklabels(monthly_customers.index.month_name(), rotation=0)
# Add gridlines
# ax_top.grid(axis='y', color='gray', alpha=.7)
# Horizontal line on x-axis
ax_top.axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)

# Remove spines
for s in ['top', 'right','bottom','left']:
    ax_top.spines[s].set_visible(False)


# Add titles
fig.text(.42, .88, 'Stores 88 and 237', fontsize=15, fontweight='bold', fontfamily='serif')
fig.text(.21, .44, 'Stores 77 and 233', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(.68, .44, 'Stores 86 and 219', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(0.78,0.88,"Test", fontweight="bold", fontfamily='serif', fontsize=15, color='#b20710')
fig.text(0.82,0.88,"|", fontweight="bold", fontfamily='serif', fontsize=15, color='black')
fig.text(0.825,0.88,"Control", fontweight="bold", fontfamily='serif', fontsize=15, color='#221f1f')


plt.tight_layout()
plt.subplots_adjust(hspace=.45)
```



Average Transactions p/Customer

In [929…

```
color = ["#b20710", "#221f1f"]

fig, ax = plt.subplots(2, 2, figsize=(12, 8))

# Plot
ax[1,0].plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,77], color=c
# Fill only between start and end date
ax[1,0].fill_between(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,77],
ax[1,0].plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,233], color=
ax[1,0].fill_between(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,233]
# Axis labels
ax[1,0].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,0].set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,0].set_xticks(avg_transactions_per_customer.index.month_name())
ax[1,0].set_xticklabels(avg_transactions_per_customer.index.month_name(), rotation=45)

for date in avg_transactions_per_customer.index:
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 77], colors='grey'
    ax[1,0].vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 233], colors='grey


# Plot
ax[1,1].plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,86], color=c
ax[1,1].fill_between(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,86],
ax[1,1].plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,219], color=
ax[1,1].fill_between(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,219]
# Axis labels
ax[1,1].set_xlabel('Month', fontsize=14, fontfamily='serif')
ax[1,1].set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax[1,1].set_xticks(avg_transactions_per_customer.index.month_name())
ax[1,1].set_xticklabels(avg_transactions_per_customer.index.month_name(), rotation=45)


# Add gridlines
# for i in range(2):
    # ax[1,i].grid(axis='y', color='gray', alpha=.7)

for date in avg_transactions_per_customer.index:
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 86], colors='grey'
    ax[1,1].vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 219], colors='grey
# Horizontal line on x-axis
for i in range(2):
    ax[1,i].yaxis.tick_left()
    ax[1,i].axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)
# Remove spines
for i in range(2):
    for s in ['top', 'right','bottom','left']:
        ax[1,i].spines[s].set_visible(False)




# Remove unnecessary subplots
fig.delaxes(ax[0, 1])  # Remove empty subplot
fig.delaxes(ax[0, 0])  # Remove empty subplot

# Add a big top plot
ax_top = fig.add_subplot(2, 1, 1)
# Plot
ax_top.plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,88], color=co
ax_top.fill_between(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,88], (
for date in avg_transactions_per_customer.index:
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 88], colors='grey',
    ax_top.vlines(x=date.month_name(), ymin=0, ymax=avg_transactions_per_customer.loc[date, 237], colors='grey'
ax_top.plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,237], color='
ax_top.fill_between(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,237],


# Plot title
ax_top.set_title('Trial and Control Stroes: Avergae # of Transaction p/Customers', pad=70, fontsize=18, fontfami
                 bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3'))
# Axis labels
ax_top.set_xlabel('Month', fontsize=14, fontfamily='serif')
ax_top.set_ylabel('Sales', fontsize=14, fontfamily='serif')
# Fix x-ticks
ax_top.set_xticks(avg_transactions_per_customer.index.month_name())
ax_top.set_xticklabels(avg_transactions_per_customer.index.month_name(), rotation=0)
# Add gridlines
# ax_top.grid(axis='y', color='gray', alpha=.7)
# Horizontal line on x-axis
ax_top.axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)

# Remove spines
for s in ['top', 'right','bottom','left']:
    ax_top.spines[s].set_visible(False)


# Add titles
fig.text(.42, .88, 'Stores 88 and 237', fontsize=15, fontweight='bold', fontfamily='serif')
fig.text(.21, .44, 'Stores 77 and 233', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(.68, .44, 'Stores 86 and 219', fontsize=15, fontfamily='serif', fontweight='bold')
fig.text(0.78,0.88,"Test", fontweight="bold", fontfamily='serif', fontsize=15, color='#b20710')
fig.text(0.82,0.88,"|", fontweight="bold", fontfamily='serif', fontsize=15, color='black')
fig.text(0.825,0.88,"Control", fontweight="bold", fontfamily='serif', fontsize=15, color='#221f1f')


plt.tight_layout()
plt.subplots_adjust(hspace=.45)
```



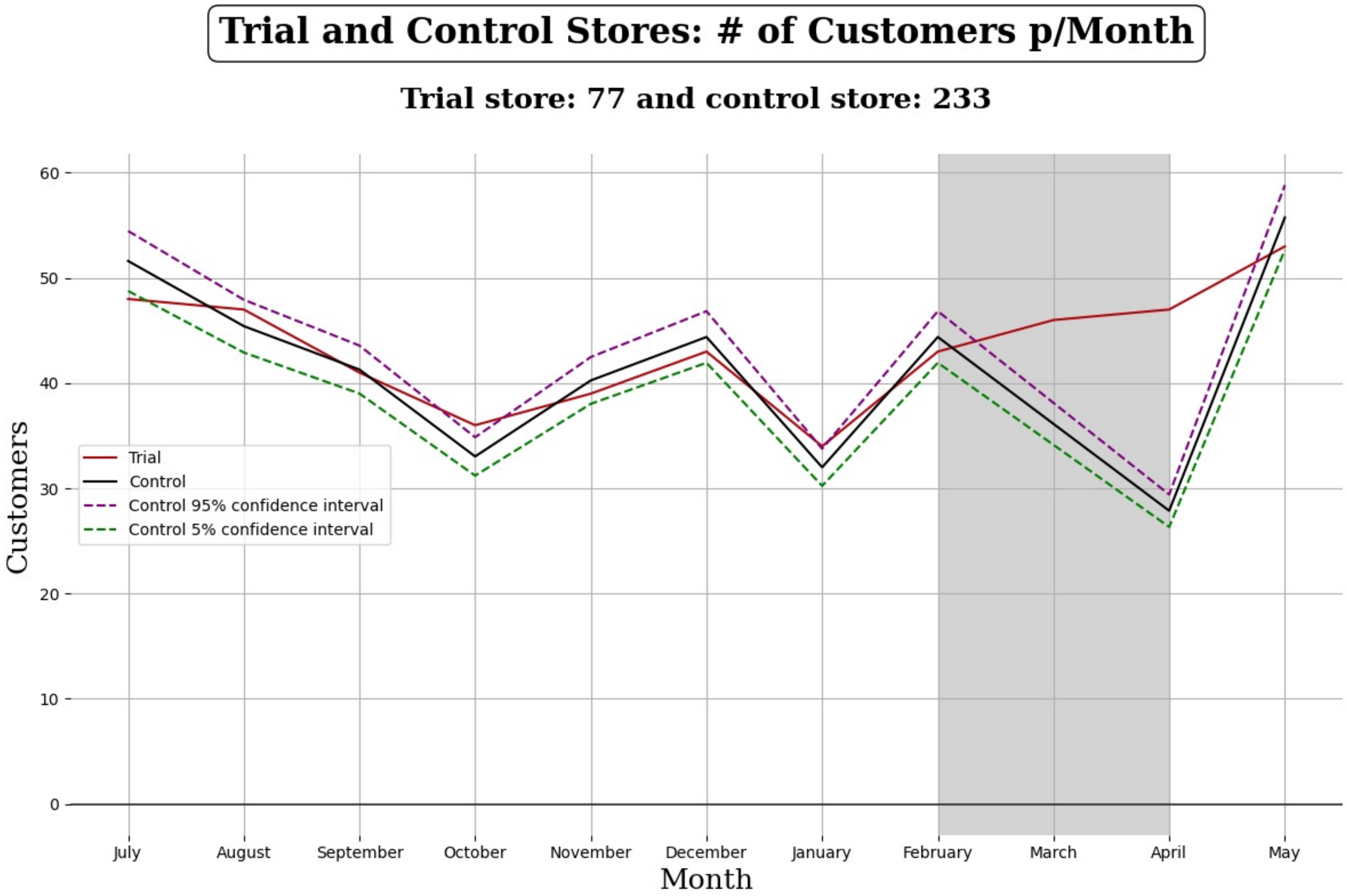
Significance of # of Customers.

```
In [ ]: def significance_check_customers(trial_store, control_store, pre_trial_cutoff = '2019-02', trial_end = '2019-04
            # 1. Create YEARMONTH column for grouping
            df['YEARMONTH'] = df['DATE'].dt.to_period('M')

            # 2. Aggregate NO. customers per month per store
            monthly_customers = df.groupby(['YEARMONTH', 'STORE_NBR'])['LYLTY_CARD_NBR'].nunique().reset_index()

            # 3. Compute scaling factor from pre-trial months
            trial_pre_total = monthly_customers[(monthly_customers['STORE_NBR'] == trial_store) &
                                                (monthly_customers['YEARMONTH'] < pre_trial_cutoff)]['LYLTY_CARD_NBR'].sum(

            control_pre_total = monthly_customers[(monthly_customers['STORE_NBR'] == control_store) &
                                                (monthly_customers['YEARMONTH'] < pre_trial_cutoff)]['LYLTY_CARD_NBR'].sum(

            scaling_factor = trial_pre_total / control_pre_total

            # 4. Apply scaling factor to control store
            monthly_customers['scaled_customers'] = None
            monthly_customers.loc[monthly_customers['STORE_NBR'] == control_store, 'scaled_customers'] = \
                monthly_customers.loc[monthly_customers['STORE_NBR'] == control_store, 'LYLTY_CARD_NBR'] * scaling_fact

            # 5. Pivot for comparison (use only relevant stores)
            comparison_df = monthly_customers[monthly_customers['STORE_NBR'].isin([trial_store, control_store])]
            pivoted = comparison_df.pivot(index='YEARMONTH', columns='STORE_NBR', values=['LYLTY_CARD_NBR', 'scaled_cus

            # 6. Flatten column names
            pivoted.columns = [f"{metric}_{store}" for metric, store in pivoted.columns]
            pivoted = pivoted.reset_index()

            # 7. Calculate percentage difference from pre-trial months
            pivoted['percentage_diff'] = np.abs(pivoted[f'LYLTY_CARD_NBR_{trial_store}.0'] - pivoted[f'scaled_customers_

            # 8. Std deviation from pre-trial period
            pre_trial = pivoted[pivoted['YEARMONTH'] < pre_trial_cutoff]
            std_dev = pre_trial['percentage_diff'].std()
            df_degrees = len(pre_trial) - 1

            # 9. Compute t-values during trial period (e.g. Feb–Apr or Mar–Jun)
            trial_period = pivoted[(pivoted['YEARMONTH'] >= pre_trial_cutoff) & (pivoted['YEARMONTH'] <= trial_end)].co
            trial_period['t_value'] = trial_period['percentage_diff'] / std_dev
            trial_period

            # 10. Critical value for 95% confidence
```

```
    t_critical = stats.t.ppf(0.95, df_degrees)

    # 11. Significance check
    trial_period['is_significant'] = trial_period['t_value'] > t_critical


    ###########################################################################################################


    # Now Plot
    start_date = '2018-07-01'
    end_date = '2019-05-31'

    # Filter the trial date
    filtered_df = df[(df['DATE'] >= start_date) & (df['DATE'] <= end_date)]
    # NO. of CUSTOMERS P/MONTH.
    monthly_customers = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')])['LYLTY_CARD_NBR']
    monthly_customers = monthly_customers.T

    color = ["#b20710", "#221f1f"]

    fig = plt.figure(figsize=(12, 8))
    ax = plt.gca()

    # Plot
    ax.plot(monthly_customers.index.month_name(), monthly_customers.loc[:,trial_store], color=color[0], label='
    ax.plot(monthly_customers.index.month_name(), scaling_factor * monthly_customers.loc[:,control_store], colo
    ax.plot(monthly_customers.index.month_name(), scaling_factor * monthly_customers.loc[:,control_store] + sca
            color='purple', linestyle='--', label=r'Control 95% confidence interval')
    ax.plot(monthly_customers.index.month_name(), scaling_factor * monthly_customers.loc[:,control_store] - sca
            color='green', linestyle='--', label=r'Control 5% confidence interval')
    ax.axvspan('February', 'April', color='lightgray')

    # Plot title
    ax.set_title('Trial and Control Stores: # of Customers p/Month', pad=70, fontsize=22, fontfamily='serif', f
                    bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3'))
    # Axis labels
    ax.set_xlabel('Month', fontsize=18, fontfamily='serif')
    ax.set_ylabel('Customers', fontsize=18, fontfamily='serif')
    # Fix x-ticks
    ax.set_xticks(monthly_customers.index.month_name())
    ax.set_xticklabels(monthly_customers.index.month_name(), rotation=0)
    # Horizontal line on x-axis
    ax.axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)
    # Gridlines
    ax.grid()
    fig.text(.3, .88, f'Trial store: {trial_store} and control store: {control_store}', fontsize=18, fontweight

    # Remove spines
    for s in ['top', 'right','bottom','left']:
        ax.spines[s].set_visible(False)

    plt.tight_layout()
    ax.legend(loc='best')
    plt.show()

    return(trial_period)
```

In [ ]:

```
significance_check_customers(77,233)
```



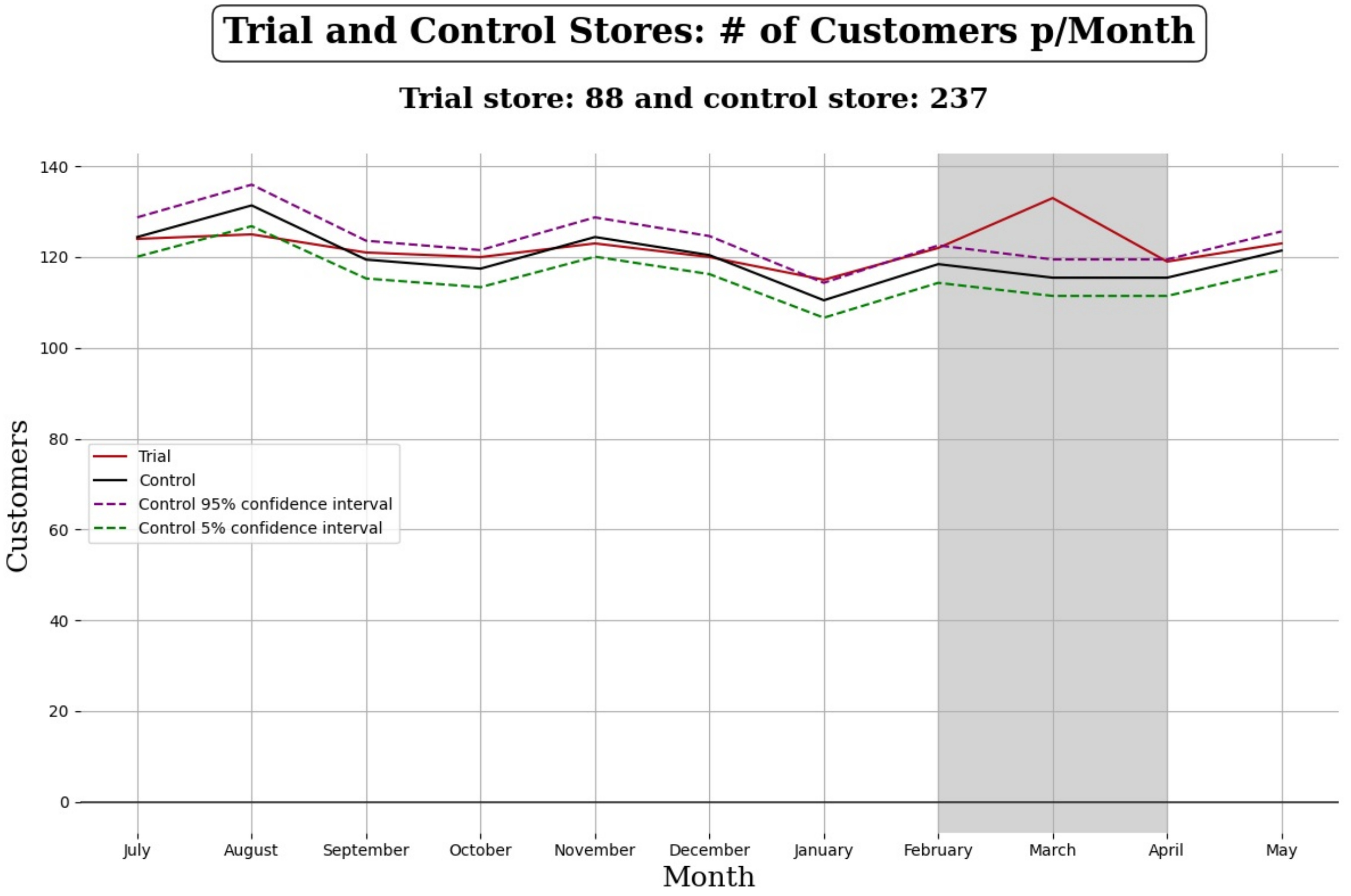
Out[ ]:

	YEARMONTH	LYLTY_CARD_NBR_77.0	LYLTY_CARD_NBR_233.0	scaled_customers_77.0	scaled_customers_233.0	percentage_di
7	2019-02	43	43	None	44.387097	0.0312
8	2019-03	46	35	None	36.129032	0.27321
9	2019-04	47	27	None	27.870968	0.68634

It looks like the number of customers is significantly higher in two of the three months. This seems to suggest that the trial had a significant impact on increasing the number of customers in trial store 77 and had an overall positive trial effect.

In [ ]:

```
significance_check_customers(88,237)
```



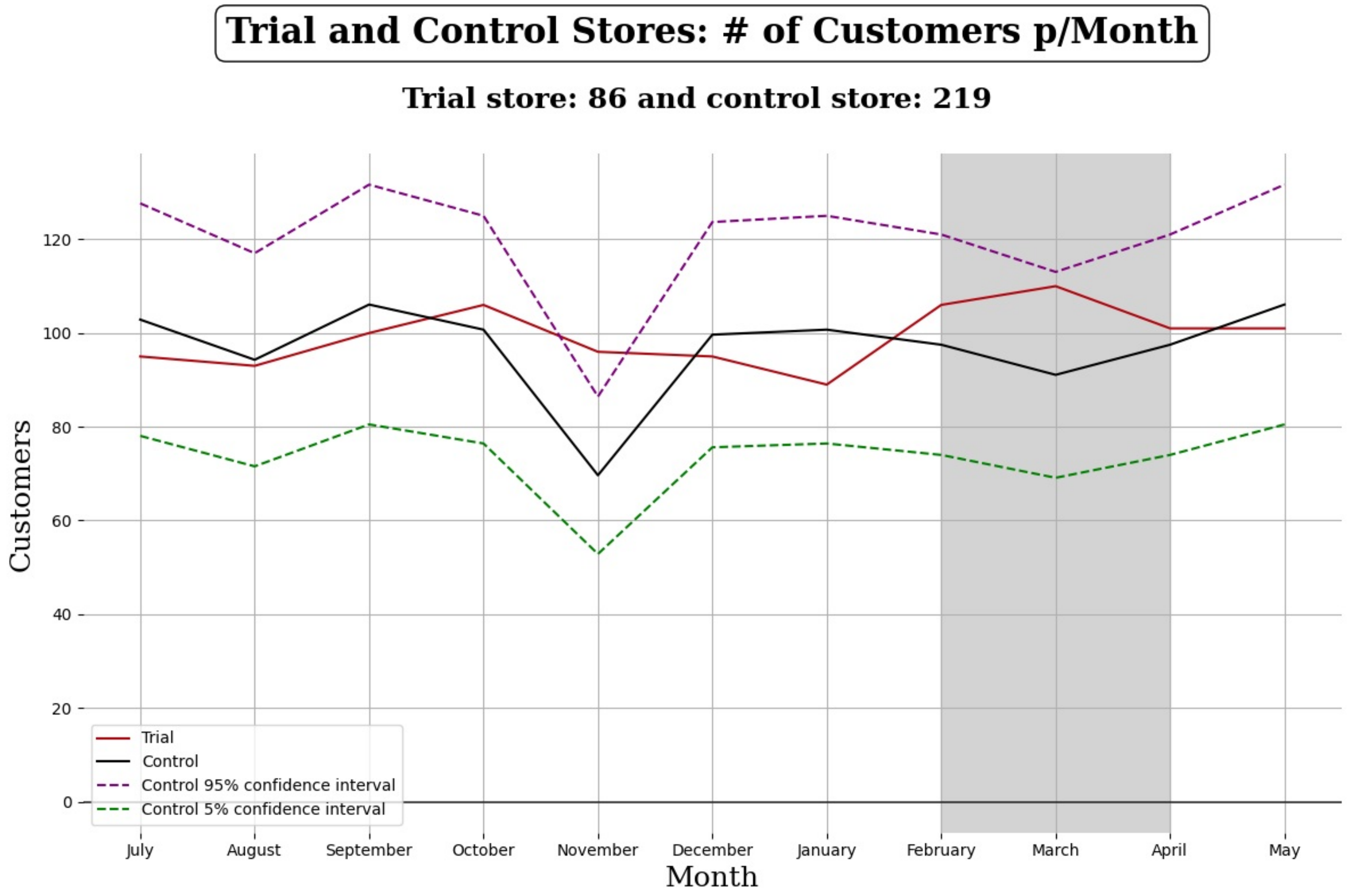
Out[ ]:

	YEARMONTH	LYLTY_CARD_NBR_88.0	LYLTY_CARD_NBR_237.0	scaled_customers_88.0	scaled_customers_237.0	percentage_di
7	2019-02	122	119	None	118.441315	0.03004
8	2019-03	133	116	None	115.455399	0.1519
9	2019-04	119	116	None	115.455399	0.03070

It looks like the number of customers is in trial store 88 is not significantly different to its control store as two of the three months lie inside the 5% to 95% confidence interval of the control store. This seems to suggest that the trial had a significant impact on increasing the spending by existing customers.

In [ ]:

```
significance_check_customers(86,219)
```



Out[ ]:

	YEARMONTH	LYLTY_CARD_NBR_86.0	LYLTY_CARD_NBR_219.0	scaled_customers_86.0	scaled_customers_219.0	percentage_di
7	2019-02	106	91	None	97.510334	0.08706
8	2019-03	110	85	None	91.081081	0.20771
9	2019-04	101	91	None	97.510334	0.03578

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 all three trial months.

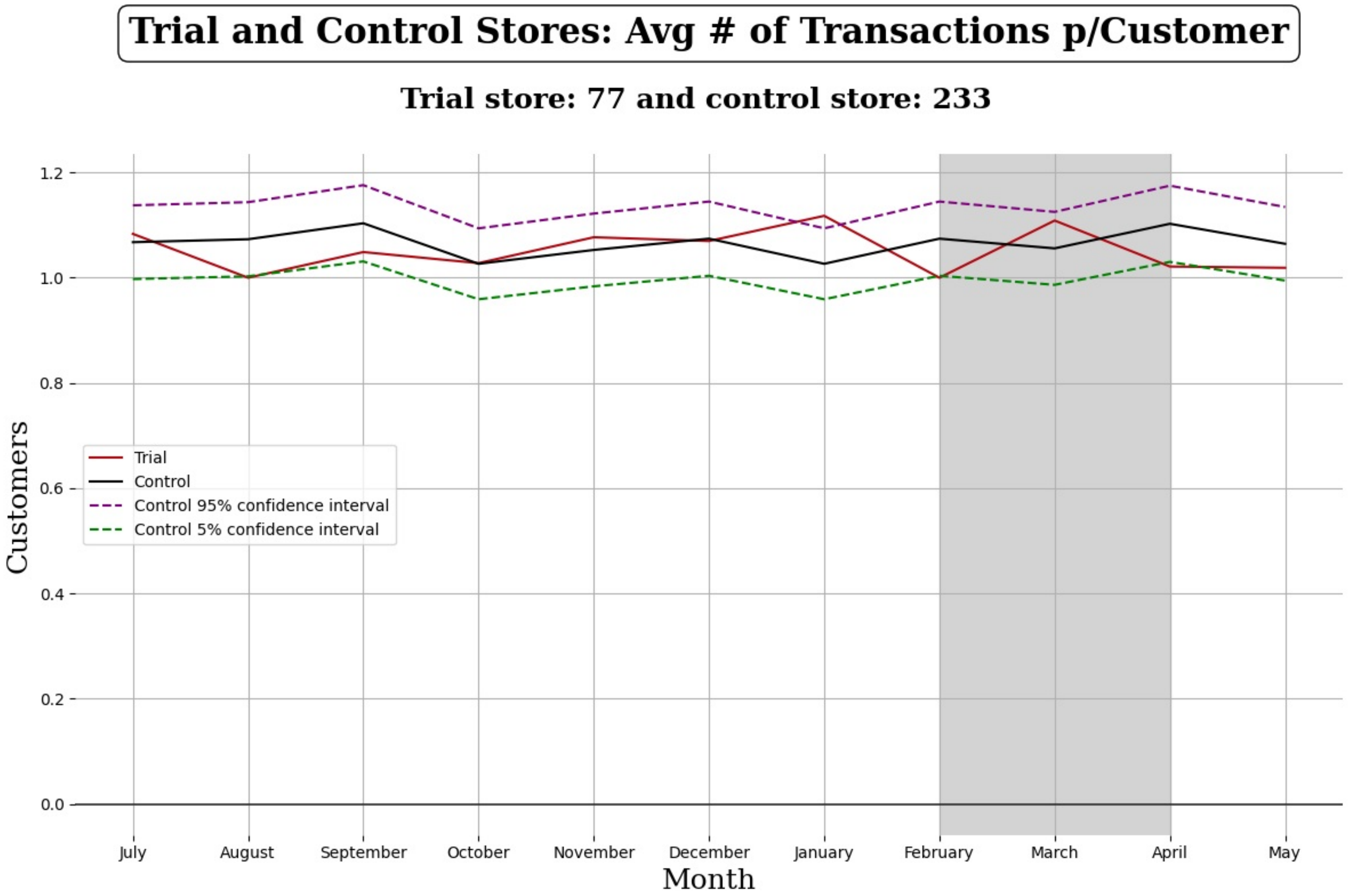
Significance of # of Transactions p/Cusotmer

In [ ]:

```python
def significance_check_transactions(trial_store, control_store, pre_trial_cutoff = '2019-02', trial_end = '2019
    # 1. Create YEARMONTH column for grouping
    df['YEARMONTH'] = df['DATE'].dt.to_period('M')

    # 2. Aggregate NO. customers per month per store
    monthly_transactions = df.groupby(['YEARMONTH', 'STORE_NBR']).size().reset_index(name='transactions')
    monthly_customers = df.groupby(['YEARMONTH', 'STORE_NBR'])['LYLTY_CARD_NBR'].nunique().reset_index(name='un
    # Merge the two DataFrames on YEARMONTH and STORE_NBR
    merged = pd.merge(monthly_transactions, monthly_customers, on=['YEARMONTH', 'STORE_NBR'])
    avg_transactions_per_customer = merged
    avg_transactions_per_customer['avg_transactions'] = avg_transactions_per_customer['transactions']/avg_transa

    # 3. Compute scaling factor from pre-trial months
    trial_pre_total = avg_transactions_per_customer[(avg_transactions_per_customer['STORE_NBR'] == trial_store)
                                                    (avg_transactions_per_customer['YEARMONTH'] < pre_trial_cutoff)]['avg_transa

    control_pre_total = avg_transactions_per_customer[(avg_transactions_per_customer['STORE_NBR'] == control_sto
                                                    (avg_transactions_per_customer['YEARMONTH'] < pre_trial_cutoff)]['avg_transa

    scaling_factor = trial_pre_total / control_pre_total

    # 4. Apply scaling factor to control store
    avg_transactions_per_customer['scaled_transactions'] = None
    avg_transactions_per_customer.loc[avg_transactions_per_customer['STORE_NBR'] == control_store, 'scaled_trans
        avg_transactions_per_customer.loc[avg_transactions_per_customer['STORE_NBR'] == control_store, 'avg_tran

    # 5. Pivot for comparison (use only relevant stores)
    comparison_df = avg_transactions_per_customer[avg_transactions_per_customer['STORE_NBR'].isin([trial_store,
    pivoted = comparison_df.pivot(index='YEARMONTH', columns='STORE_NBR', values=['avg_transactions', 'scaled_t

    # 6. Flatten column names
    pivoted.columns = [f"{metric}_{store}" for metric, store in pivoted.columns]
    pivoted = pivoted.reset_index()
    # 7. Calculate percentage difference from pre-trial months
    pivoted['percentage_diff'] = np.abs(pivoted[f'avg_transactions_{trial_store}.0'] - pivoted[f'scaled_transac

    # 8. Std deviation from pre-trial period
    pre_trial = pivoted[pivoted['YEARMONTH'] < pre_trial_cutoff]
    std_dev = pre_trial['percentage_diff'].std()
    df_degrees = len(pre_trial) - 1

    # 9. Compute t-values during trial period (e.g. Feb–Apr or Mar–Jun)
    trial_period = pivoted[(pivoted['YEARMONTH'] >= pre_trial_cutoff) & (pivoted['YEARMONTH'] <= trial_end)].co
    trial_period['t_value'] = trial_period['percentage_diff'] / std_dev
    trial_period

    # 10. Critical value for 95% confidence
    t_critical = stats.t.ppf(0.95, df_degrees)

    # 11. Significance check
    trial_period['is_significant'] = trial_period['t_value'] > t_critical


    # ##########################################################################################################



    # Now Plot
    start_date = '2018-07-01'
    end_date = '2019-05-31'

    # Filter the trial date
    filtered_df = df[(df['DATE'] >= start_date) & (df['DATE'] <= end_date)]
    # Average number of transactions per customer per month.
    monthly_transactions = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')]).size().unstack
    monthly_customers = filtered_df.groupby(['STORE_NBR', pd.Grouper(key='DATE', freq='ME')])['LYLTY_CARD_NBR']
    avg_transactions_per_customer = monthly_transactions / monthly_customers  # Calculate the average number of
    avg_transactions_per_customer = avg_transactions_per_customer.T

    color = ["#b20710", "#221f1f"]

    fig = plt.figure(figsize=(12, 8))
    ax = plt.gca()

    # Plot
    ax.plot(avg_transactions_per_customer.index.month_name(), avg_transactions_per_customer.loc[:,trial_store],
    ax.plot(avg_transactions_per_customer.index.month_name(), scaling_factor * avg_transactions_per_customer.lo
    ax.plot(avg_transactions_per_customer.index.month_name(), scaling_factor * avg_transactions_per_customer.lo
            color='purple', linestyle='--', label=r'Control 95% confidence interval')
    ax.plot(avg_transactions_per_customer.index.month_name(), scaling_factor * avg_transactions_per_customer.lo
            color='green', linestyle='--', label=r'Control 5% confidence interval')
    ax.axvspan('February', 'April', color='lightgray')

    # Plot title
    ax.set_title('Trial and Control Stores: Avg # of Transactions p/Customer', pad=70, fontsize=22, fontfamily=
                    bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.3'))
    # Axis labels
    ax.set_xlabel('Month', fontsize=18, fontfamily='serif')
    ax.set_ylabel('Customers', fontsize=18, fontfamily='serif')
    # Fix x-ticks
    ax.set_xticks(avg_transactions_per_customer.index.month_name())
    ax.set_xticklabels(avg_transactions_per_customer.index.month_name(), rotation=0)
    # Horizontal line on x-axis
    ax.axhline(y = 0, color = 'black', linewidth = 1.3, alpha = .7)
    # Gridlines
    ax.grid()
    fig.text(.3, .88, f'Trial store: {trial_store} and control store: {control_store}', fontsize=18, fontweight

    # Remove spines
    for s in ['top', 'right','bottom','left']:
        ax.spines[s].set_visible(False)

    plt.tight_layout()
    ax.legend(loc='best')
    plt.show()

    return(trial_period)
```

In [994…

```
significance_check_transactions(77,233)
```



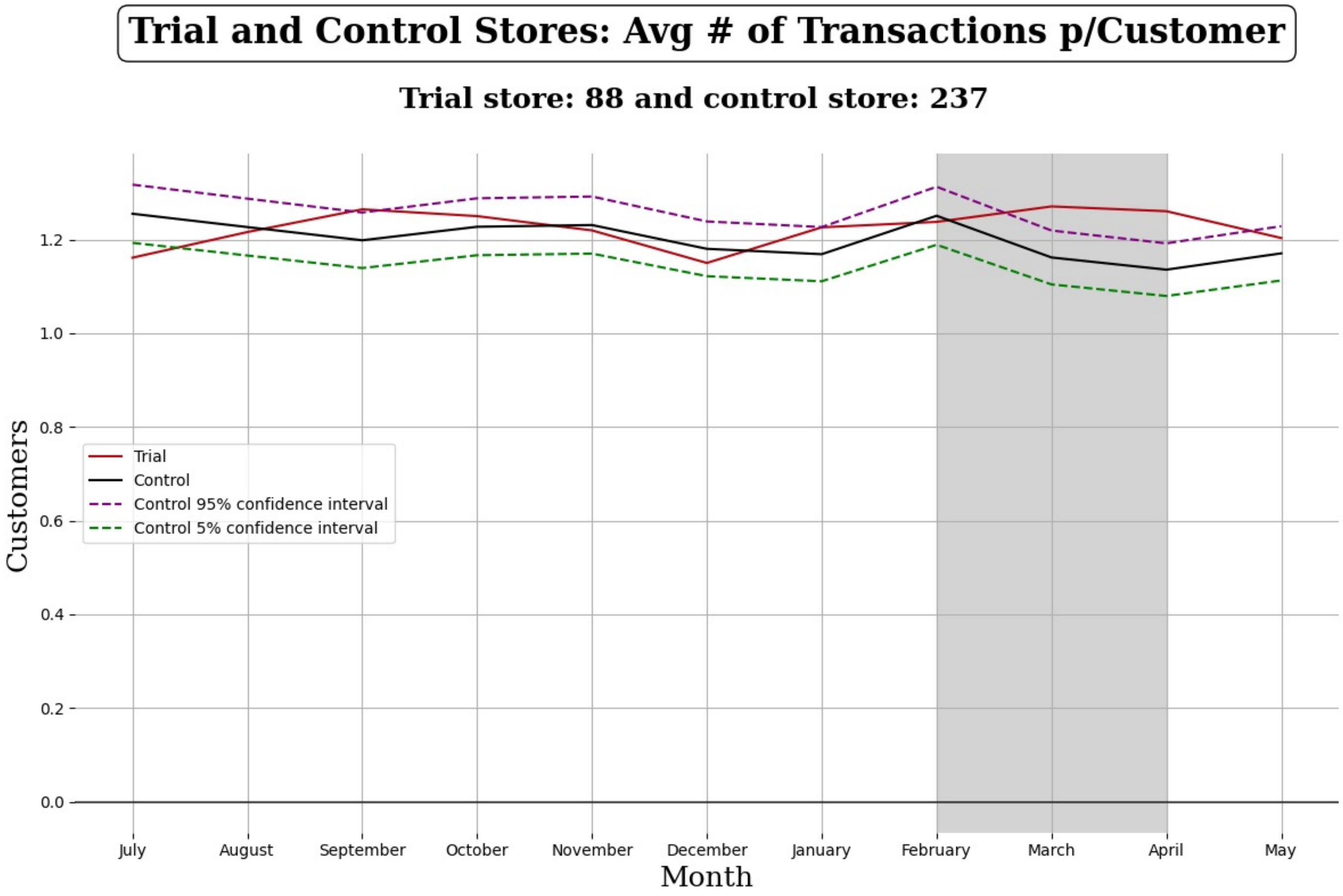
Out[994…

	YEARMONTH	avg_transactions_77.0	avg_transactions_233.0	scaled_transactions_77.0	scaled_transactions_233.0	percentage_dif
7	2019-02	1.0	1.046512	None	1.074238	0.06910
8	2019-03	1.108696	1.028571	None	1.055823	0.05007
9	2019-04	1.021277	1.074074	None	1.102531	0.07369

Here we see the number of transactions p/customer in the trial period is not significantly different to the control store. This suggests that while the trial period was overall successful in generating more revenue, it did so by attracting new customers and increasing the # of customers p/month rather than increasing transactions of existing customers.

In [995…

```
significance_check_transactions(88,237)
```



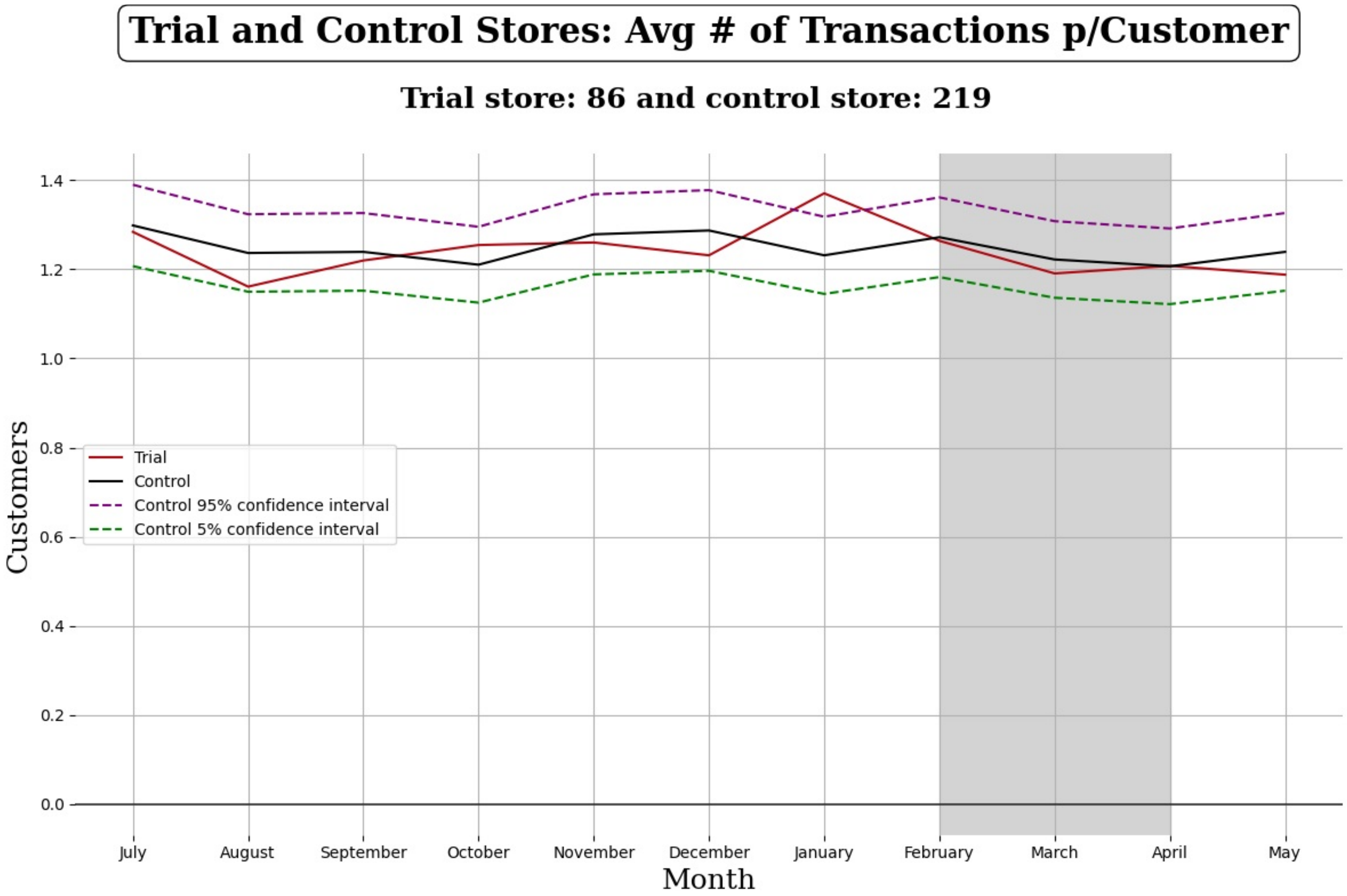
Out[995…

	YEARMONTH	avg_transactions_88.0	avg_transactions_237.0	scaled_transactions_88.0	scaled_transactions_237.0	percentage_dif
7	2019-02	1.237705	1.243697	None	1.250761	0.01043
8	2019-03	1.270677	1.155172	None	1.161734	0.09377
9	2019-04	1.260504	1.12931	None	1.135725	0.10986

As we had suggested earlier in the # of custoemrs p/Month, it seems the trial run was successful in getting existing customers to make more transactions. This hypothesis is confirmed here as we see the number of transactions p/customer is significantly higher in two of the three months. This suggests that the trial had a significant impact on increasing the number of transactions p/customers in trial store 88 and had an overall positive trial effect.

In [996…

```
significance_check_transactions(86,219)
```



Out[996…

	YEARMONTH	avg_transactions_86.0	avg_transactions_219.0	scaled_transactions_86.0	scaled_transactions_219.0	percentage_dif
7	2019-02	1.264151	1.285714	None	1.272172	0.006305
8	2019-03	1.190909	1.235294	None	1.222283	0.025668
9	2019-04	1.207921	1.21978	None	1.206933	0.000819

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 all three trial months.

Conclusion

We’ve found control stores 233, 219, 237 for trial stores 77, 86 and 88 respectively.

The results for trial stores 77 and 88 during the trial period show a significant difference in at least two of the three trial months but this is not the case for trial store 86.

We can check with the client if the implementation of the trial was different in trial store 86.

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.