In [ ]:
```
# 1. Introduction

**Task 2 – Trial layout evaluation**

Objective: For trial stores **77, 86, 88**, select suitable control stores using pre-trial monthly metrics,
compare trial vs control during the trial period, test significance, and identify performance drivers.
```

In [1]:
```
# Cell 1 - imports, config, load
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from scipy.stats import ttest_ind
sns.set(style="whitegrid")
plt.rcParams['figure.dpi'] = 140

# Adjust this path if your file is elsewhere
DATA_DIR = Path(r"C:\Users\HP\Desktop")   # or Path(".") for current folder
DATA_FILE = DATA_DIR / "QVI_data.csv"     # change to QVI_merged_clean.csv if that's the file

# Load
df = pd.read_csv(DATA_FILE, parse_dates=['DATE'], encoding='latin1', dayfirst=False)
print("Loaded rows,cols:", df.shape)
df.head()
```

```
Loaded rows,cols: (264834, 12)
```

Out[1]:

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

In [2]:
```
# Cell 2 - basic cleaning and month period creation
# Convert DATE to datetime if needed (already parsed above), then create YEAR_MONTH period (month start timestamp)
df['DATE'] = pd.to_datetime(df['DATE'], errors='coerce')
df['YEAR_MONTH'] = df['DATE'].dt.to_period('M').dt.to_timestamp()

# Quick sanity checks
print("Date range:", df['DATE'].min(), "to", df['DATE'].max())
print("Unique stores:", df['STORE_NBR'].nunique())
df[['DATE','YEAR_MONTH']].head()
```

```
Date range: 2018-07-01 00:00:00 to 2019-06-30 00:00:00
Unique stores: 272
```

Out[2]:

	DATE	YEAR_MONTH
0	2018-10-17	2018-10-01
1	2018-09-16	2018-09-01
2	2019-03-07	2019-03-01
3	2019-03-08	2019-03-01
4	2018-11-02	2018-11-01

In [3]:
```
# Cell 3 - monthly metrics per store (same metrics as R)
monthly = (
    df.groupby(['STORE_NBR','YEAR_MONTH'])
      .agg(TOT_SALES = ('TOT_SALES','sum'),
           UNIQUE_CUSTOMERS = ('LYLTY_CARD_NBR','nunique'),
           TOTAL_UNITS = ('PROD_QTY','sum'),
           TXN_LINES = ('TXN_ID','nunique'))
      .reset_index()
)
monthly['AVG_TXNS_PER_CUSTOMER'] = monthly['TXN_LINES'] / monthly['UNIQUE_CUSTOMERS']
monthly['AVG_PRICE_PER_UNIT'] = monthly['TOT_SALES'] / monthly['TOTAL_UNITS']
monthly = monthly.sort_values(['STORE_NBR','YEAR_MONTH']).reset_index(drop=True)
print("Monthly table rows:", monthly.shape[0])
monthly.head()
```

```
Monthly table rows: 3169
```

Out[3]:

	STORE_NBR	YEAR_MONTH	TOT_SALES	UNIQUE_CUSTOMERS	TOTAL_UNITS	TXN_LINES	AVG_TXNS_PER_CUSTOMER	AVG_PRICE_PER_UNIT
0	1	2018-07-01	206.9	49	62	52	1.061224	3.337097
1	1	2018-08-01	176.1	42	54	43	1.023810	3.261111
2	1	2018-09-01	278.8	59	75	62	1.050847	3.717333
3	1	2018-10-01	188.1	44	58	45	1.022727	3.243103
4	1	2018-11-01	192.6	46	57	47	1.021739	3.378947

In [4]:
```
# Cell 4 - remove known outlier loyalty card if present
if 226000 in df['LYLTY_CARD_NBR'].unique():
    print("Found outlier card 226000 — removing its transactions.")
    df = df[df['LYLTY_CARD_NBR'] != 226000].copy()
    # rebuild monthly quickly
    monthly = (
        df.groupby(['STORE_NBR','YEAR_MONTH'])
          .agg(TOT_SALES = ('TOT_SALES','sum'),
               UNIQUE_CUSTOMERS = ('LYLTY_CARD_NBR','nunique'),
               TOTAL_UNITS = ('PROD_QTY','sum'),
               TXN_LINES = ('TXN_ID','nunique'))
          .reset_index()
    )
    monthly['AVG_TXNS_PER_CUSTOMER'] = monthly['TXN_LINES'] / monthly['UNIQUE_CUSTOMERS']
    monthly['AVG_PRICE_PER_UNIT'] = monthly['TOT_SALES'] / monthly['TOTAL_UNITS']
    monthly = monthly.sort_values(['STORE_NBR','YEAR_MONTH']).reset_index(drop=True)
    print("Rebuilt monthly shape:", monthly.shape)
else:
    print("No special outlier card found; continuing.")
```

```
No special outlier card found; continuing.
```

In [5]:
```
# Cell 5 - trial stores and trial period
trial_stores = [77, 86, 88]
trial_start = pd.to_datetime("2019-03-01")
trial_end = pd.to_datetime("2019-04-30")

print("Trial stores:", trial_stores)
print("Trial window:", trial_start.date(), "to", trial_end.date())
```

```
Trial stores: [77, 86, 88]
Trial window: 2019-03-01 to 2019-04-30
```

In [6]:
```
# Cell 6 - build pivots for pre-trial months
pre_mask = monthly['YEAR_MONTH'] < trial_start
pre_monthly = monthly[pre_mask].copy()

# Ensure each store has same months indexing: build full months range
months_sorted = sorted(pre_monthly['YEAR_MONTH'].unique())
def pivot_metric(metric):
    p = pre_monthly.pivot(index='STORE_NBR', columns='YEAR_MONTH', values=metric)
    # reindex columns to full month list and fill missing with 0
    p = p.reindex(columns=months_sorted).fillna(0)
    return p

pivot_sales = pivot_metric('TOT_SALES')
pivot_customers = pivot_metric('UNIQUE_CUSTOMERS')
pivot_avgtxn = pivot_metric('AVG_TXNS_PER_CUSTOMER')

print("Pivot shapes:", pivot_sales.shape, pivot_customers.shape, pivot_avgtxn.shape)
```

```
Pivot shapes: (271, 8) (271, 8) (271, 8)
```

In [7]:
```
# Cell 7 - control selection function (Quantium-style)
from scipy.spatial.distance import cdist

def select_controls(trial_store, candidate_stores=None, n_controls=3, method='pearson'):
    if candidate_stores is None:
        candidate_stores = [s for s in pivot_sales.index if s != trial_store]
    # build normalized vectors per store: center & scale each series then concat
    def build_vec(store):
        s = pivot_sales.loc[store].values
        c = pivot_customers.loc[store].values
        a = pivot_avgtxn.loc[store].values
        # standardize each series (z-score), protect against zero-std
        def z(x):
            std = x.std(ddof=0)
            return (x - x.mean()) / (std if std!=0 else 1.0)
        vec = np.concatenate([ z(s), z(c), z(a) ])
        return vec

    trial_vec = build_vec(trial_store)
    candidates = [s for s in candidate_stores if s in pivot_sales.index]
    cand_vecs = {s: build_vec(s) for s in candidates}

    if method == 'pearson':
        corrs = {}
        for s,v in cand_vecs.items():
            # handle constant vectors by setting corr=0
            if np.std(v)==0 or np.std(trial_vec)==0:
                corr = 0.0
            else:
                corr = np.corrcoef(trial_vec, v)[0,1]
```

```
            corrs[s] = corr
        selected = sorted(corrs.items(), key=lambda x: x[1], reverse=True)[:n_controls]
        return [s for s,_ in selected], corrs

    else:
        # magnitude: euclidean distance on vectors -> convert to similarity 0..1
        X = np.vstack([trial_vec] + [cand_vecs[s] for s in candidates])
        names = [trial_store] + candidates
        dists = cdist(X, X, metric='euclidean')[0,1:]
        mn, mx = dists.min(), dists.max()
        sims = {name: 1 - (dist - mn) / (mx - mn) if mx>mn else 1.0 for name, dist in zip(candidates, dists)}
        selected = sorted(sims.items(), key=lambda x: x[1], reverse=True)[:n_controls]
        return [s for s,_ in selected], sims

# Quick test (run for one trial store)
controls_77, corrs_77 = select_controls(77, n_controls=3, method='pearson')
print("Top controls for 77 (pearson):", controls_77)
```

```
Top controls for 77 (pearson): [119, 162, 17]
```

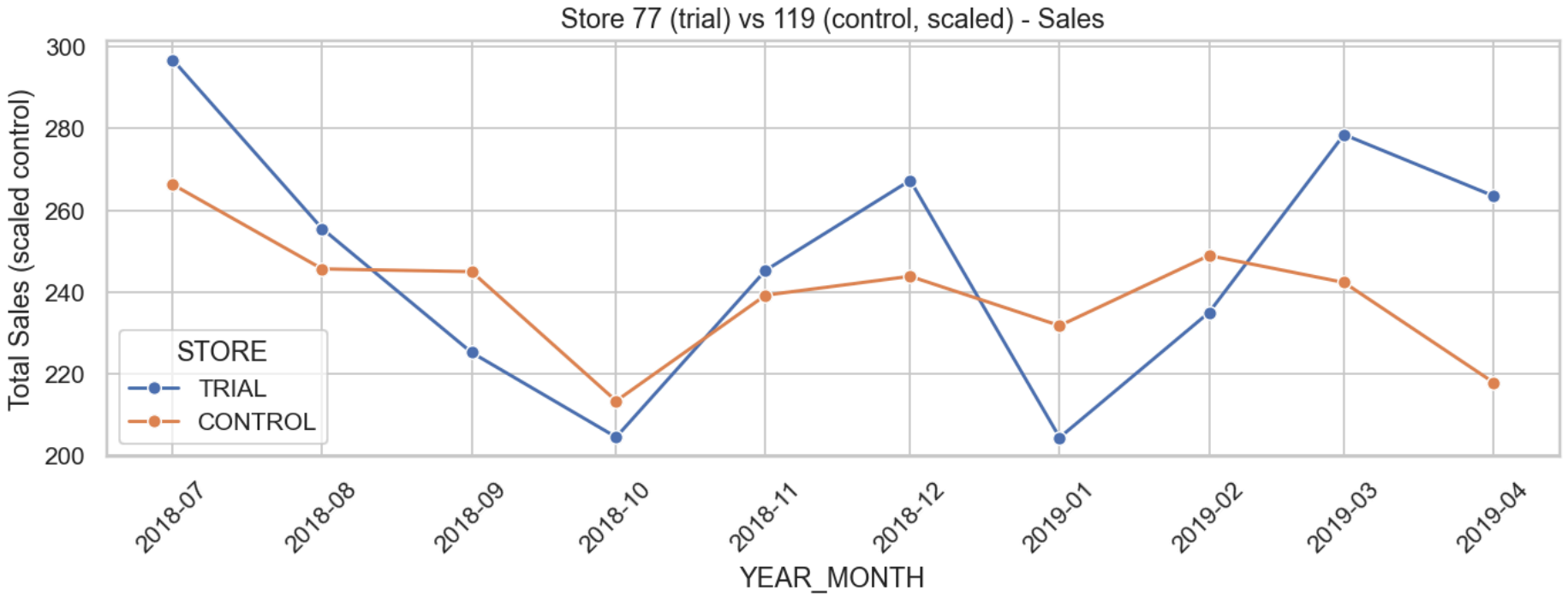
In [13]:

```
# Cell 8 - compare function: compares a trial store to a single control
def compare_trial_control(trial_store, control_store, trial_start=trial_start, trial_end=trial_end, save_plots=True, out_dir=DATA_DIR):
    # masks
    pre_mask = (monthly['YEAR_MONTH'] < trial_start)
    trial_mask = (monthly['YEAR_MONTH'] >= trial_start) & (monthly['YEAR_MONTH'] <= trial_end)

    def subset(s, mask):
        tmp = monthly[(monthly['STORE_NBR']==s) & mask].set_index('YEAR_MONTH').sort_index()
        return tmp

    pre_t = subset(trial_store, pre_mask)
    pre_c = subset(control_store, pre_mask)
    trial_t = subset(trial_store, trial_mask)
    trial_c = subset(control_store, trial_mask)

    # scale control to trial pre-trial magnitude:
    # compute scale = sum(pre_t sales)/sum(pre_c sales) -> multiply control series by scale
    denom = pre_c['TOT_SALES'].sum()
    scale = (pre_t['TOT_SALES'].sum() / denom) if denom != 0 else 1.0

    # create combined series for plotting (scaled control)
    plot_df = pd.concat([
        pd.DataFrame({'STORE':'TRIAL','YEAR_MONTH':pre_t.index,'TOT_SALES':pre_t['TOT_SALES'].values}),
        pd.DataFrame({'STORE':'TRIAL','YEAR_MONTH':trial_t.index,'TOT_SALES':trial_t['TOT_SALES'].values}),
        pd.DataFrame({'STORE':'CONTROL','YEAR_MONTH':pre_c.index,'TOT_SALES':(pre_c['TOT_SALES']*scale).values}),
        pd.DataFrame({'STORE':'CONTROL','YEAR_MONTH':trial_c.index,'TOT_SALES':(trial_c['TOT_SALES']*scale).values})
    ])

    # Plot sales lines with points and ±2*std band using pre-trial std of difference
    fig, ax = plt.subplots(figsize=(10,4))
    sns.lineplot(data=plot_df, x='YEAR_MONTH', y='TOT_SALES', hue='STORE', marker='o', ax=ax)
    ax.set_title(f"Store {trial_store} (trial) vs {control_store} (control, scaled) - Sales")
    ax.set_ylabel("Total Sales (scaled control)")
    plt.xticks(rotation=45)
    plt.tight_layout()
    if save_plots:
        fn = out_dir / f"compare_store{trial_store}_vs_{control_store}_sales.png"
        fig.savefig(fn)
    plt.show()

    # Test: monthly sales differences during trial (use original trial_t vs scaled trial_c)
```

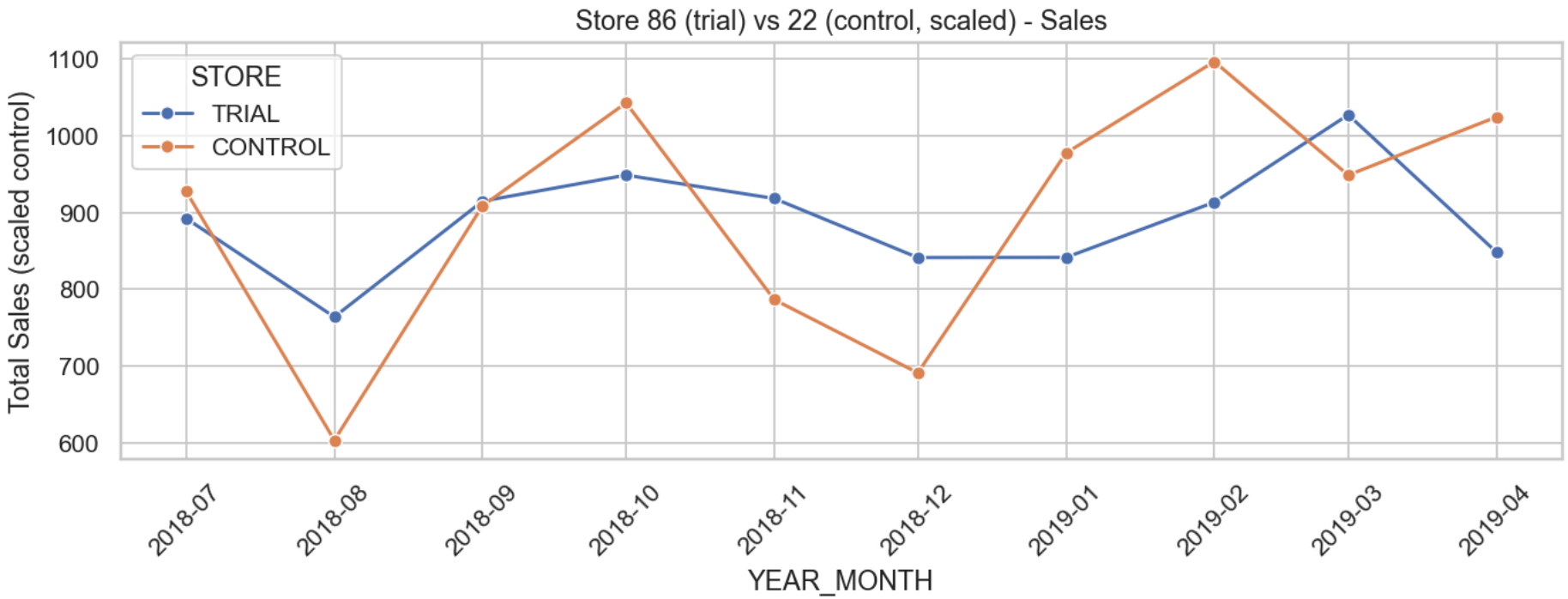
```
    # prepare arrays of sales in trial months, align by YEAR_MONTH index
    aligned = pd.concat([trial_t['TOT_SALES'], (trial_c['TOT_SALES']*scale)], axis=1, keys=['trial','control']).dropna()
    t_stat, p_val = (np.nan, np.nan)
    if aligned.shape[0] >= 1:
        t_stat, p_val = ttest_ind(aligned['trial'].values, aligned['control'].values, equal_var=False)
    # drivers during trial period (using raw unscaled numbers for customers/units)
    customers_t = trial_t['UNIQUE_CUSTOMERS'].sum()
    customers_c = trial_c['UNIQUE_CUSTOMERS'].sum()
    units_t = trial_t['TOTAL_UNITS'].sum()
    units_c = trial_c['TOTAL_UNITS'].sum()
    units_per_customer_t = units_t / customers_t if customers_t>0 else np.nan
    units_per_customer_c = units_c / customers_c if customers_c>0 else np.nan

    print(f"=== Store {trial_store} vs {control_store} (control scaled by {scale:.3f}) ===")
    print(f"Trial period months: {trial_t.index.min().date() if not trial_t.empty else 'N/A'} to {trial_t.index.max().date() if not trial_t.empty else 'N/A'}")
    print(f"Trial TOT_SALES (sum): {trial_t['TOT_SALES'].sum():.2f}")
    print(f"Control TOT_SALES (scaled sum): {(trial_c['TOT_SALES']*scale).sum():.2f}")
    print(f"Monthly sales t-test: t={t_stat:.3f}, p={p_val:.3f}")
    print(f"Trial UNIQUE_CUSTOMERS: {customers_t} | Control UNIQUE_CUSTOMERS: {customers_c}")
    print(f"Units per customer (trial vs control): {units_per_customer_t:.2f} vs {units_per_customer_c:.2f}")
    if save_plots:
        print("Saved plot:", fn)
    return {
        'trial': trial_t, 'control': trial_c, 'scale': scale,
        't_stat': t_stat, 'p_val': p_val,
        'customers_t': customers_t, 'customers_c': customers_c,
        'units_per_customer_t': units_per_customer_t, 'units_per_customer_c': units_per_customer_c,
        'plot': fn if save_plots else None
    }
```

In [9]:
```
# Cell 9 - run selection and comparisons for each trial store
results = {}
for ts in trial_stores:
    # select top 3 candidate controls by Pearson
    candidate_controls, corrs = select_controls(ts, n_controls=10, method='pearson')
    # pick the top 3 highest correlations and show them
    top3 = candidate_controls[:3]
    print(f"\nTrial store {ts} - top candidate controls (pearson): {top3}")
    # compare with the top candidate
    comp = compare_trial_control(ts, top3[0], trial_start=trial_start, trial_end=trial_end, save_plots=True, out_dir=DATA_DIR)
    results[ts] = {'controls': top3, 'corrs': corrs, 'comparison': comp}
```

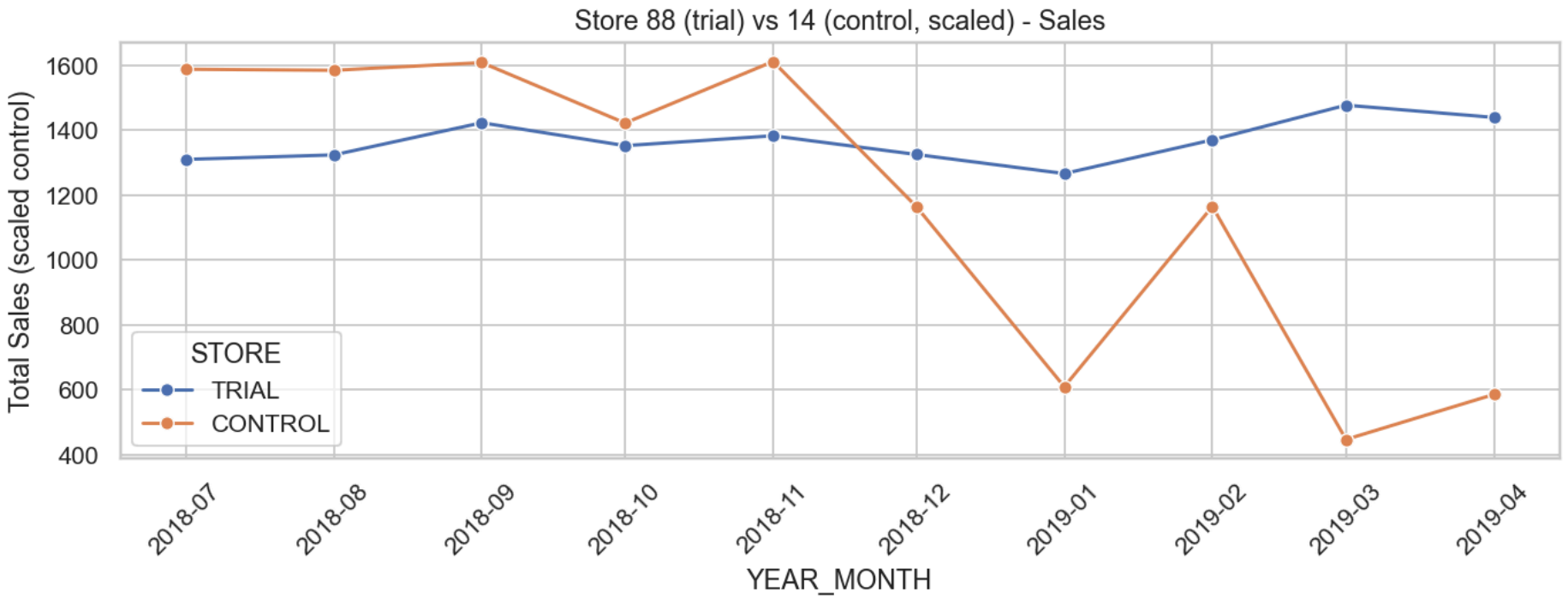
```
Trial store 77 - top candidate controls (pearson): [119, 162, 17]
```




```
=== Store 77 vs 119 (control scaled by 0.248) ===
Trial period months: 2019-03-01 to 2019-04-01
Trial TOT_SALES (sum): 542.00
Control TOT_SALES (scaled sum): 460.21
Monthly sales t-test: t=2.852, p=0.128
Trial UNIQUE_CUSTOMERS: 97 | Control UNIQUE_CUSTOMERS: 182
Units per customer (trial vs control): 1.65 vs 2.34
Saved plot: C:\Users\HP\Desktop\compare_store77_vs_119_sales.png

Trial store 86 - top candidate controls (pearson): [22, 176, 260]
```




```
=== Store 86 vs 22 (control scaled by 2.999) ===
Trial period months: 2019-03-01 to 2019-04-01
Trial TOT_SALES (sum): 1875.00
Control TOT_SALES (scaled sum): 1972.42
Monthly sales t-test: t=-0.502, p=0.685
Trial UNIQUE_CUSTOMERS: 220 | Control UNIQUE_CUSTOMERS: 104
Units per customer (trial vs control): 2.45 vs 1.91
Saved plot: C:\Users\HP\Desktop\compare_store86_vs_22_sales.png

Trial store 88 - top candidate controls (pearson): [14, 178, 201]
```




```
=== Store 88 vs 14 (control scaled by 33.860) ===
Trial period months: 2019-03-01 to 2019-04-01
Trial TOT_SALES (sum): 2916.60
Control TOT_SALES (scaled sum): 1032.72
Monthly sales t-test: t=13.094, p=0.034
Trial UNIQUE_CUSTOMERS: 262 | Control UNIQUE_CUSTOMERS: 5
Units per customer (trial vs control): 2.53 vs 1.80
Saved plot: C:\Users\HP\Desktop\compare_store88_vs_14_sales.png
```

In [10]:

```python
# Cell 10 - compile summary table
rows = []
for ts, info in results.items():
    comp = info['comparison']
    rows.append({
        'trial_store': ts,
        'control_store': info['controls'][0],
        'p_value_sales': comp['p_val'],
        'trial_sales_sum': comp['trial']['TOT_SALES'].sum(),
        'control_sales_sum_scaled': (comp['control']['TOT_SALES']*comp['scale']).sum(),
        'trial_customers': comp['customers_t'],
        'control_customers': comp['customers_c'],
        'units_per_cust_trial': comp['units_per_customer_t'],
        'units_per_cust_control': comp['units_per_customer_c']
    })
summary_df = pd.DataFrame(rows)
display(summary_df)
summary_df.to_csv(DATA_DIR / "task2_trial_control_summary.csv", index=False)
print("Saved summary CSV to:", DATA_DIR / "task2_trial_control_summary.csv")
```

	trial_store	control_store	p_value_sales	trial_sales_sum	control_sales_sum_scaled	trial_customers	control_customers	units_per_cust_trial	units_per_cust_control
0	77	119	0.128079	542.0	460.208276	97	182	1.649485	2.340659
1	86	22	0.685187	1875.0	1972.421392	220	104	2.445455	1.913462
2	88	14	0.034385	2916.6	1032.716940	262	5	2.534351	1.800000

```
Saved summary CSV to: C:\Users\HP\Desktop\task2_trial_control_summary.csv
```

In [11]:
```python
# Cell 11 - readable recommendations
print("=== Task 2 - Trial Evaluation Summary ===\n")
for i,row in summary_df.iterrows():
    ts = row['trial_store']
    cs = row['control_store']
    p = row['p_value_sales']
    print(f"Store {ts} (vs control {cs}):")
    if pd.isna(p):
        print(" - Not enough data to test significance.")
    elif p < 0.05:
        print(f" - Significant change in total sales during trial (p={p:.3f}).")
        # driver judgement
        if row['trial_customers'] > row['control_customers']:
            print(" - Driver: Increase in number of purchasing customers.")
        else:
            print(" - Driver: Increase in purchases per customer (basket size).")
        print(" - Recommendation: Consider rolling out layout (further A/B tests recommended).")
    else:
        print(f" - No significant change detected (p={p:.3f}).")
        print(" - Recommendation: Do not roll out; consider further investigation or longer trial.")
    print("----------------------------------------------------------")
```

```
=== Task 2 - Trial Evaluation Summary ===

Store 77.0 (vs control 119.0):
 - No significant change detected (p=0.128).
 - Recommendation: Do not roll out; consider further investigation or longer trial.
----------------------------------------------------------
Store 86.0 (vs control 22.0):
 - No significant change detected (p=0.685).
 - Recommendation: Do not roll out; consider further investigation or longer trial.
----------------------------------------------------------
Store 88.0 (vs control 14.0):
 - Significant change in total sales during trial (p=0.034).
 - Driver: Increase in number of purchasing customers.
 - Recommendation: Consider rolling out layout (further A/B tests recommended).
----------------------------------------------------------
```

In [16]:
```python
# 3. Data preparation
# Convert DATE, create YEAR_MONTH (month start timestamp)
df['DATE'] = pd.to_datetime(df['DATE'], errors='coerce')
df['YEAR_MONTH'] = df['DATE'].dt.to_period('M').dt.to_timestamp()

# Optional: remove known outlier customer if present (as in model answer)
if 226000 in df['LYLTY_CARD_NBR'].unique():
    df = df[df['LYLTY_CARD_NBR'] != 226000].copy()
    print("Removed loyalty card 226000 (outlier).")

# Quick checks
print("Date range:", df['DATE'].min(), "to", df['DATE'].max())
```

```
print("Unique stores:", df['STORE_NBR'].nunique())
df.info()
```

```
Date range: 2018-07-01 00:00:00 to 2019-06-30 00:00:00
Unique stores: 272
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 264834 entries, 0 to 264833
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   LYLTY_CARD_NBR    264834 non-null  int64
 1   DATE              264834 non-null  datetime64[ns]
 2   STORE_NBR         264834 non-null  int64
 3   TXN_ID            264834 non-null  int64
 4   PROD_NBR          264834 non-null  int64
 5   PROD_NAME         264834 non-null  object
 6   PROD_QTY          264834 non-null  int64
 7   TOT_SALES         264834 non-null  float64
 8   PACK_SIZE         264834 non-null  int64
 9   BRAND             264834 non-null  object
 10  LIFESTAGE         264834 non-null  object
 11  PREMIUM_CUSTOMER  264834 non-null  object
 12  YEAR_MONTH        264834 non-null  datetime64[ns]
dtypes: datetime64[ns](2), float64(1), int64(6), object(4)
memory usage: 26.3+ MB
```

In [17]:
```
# 4. Monthly aggregation: totals, unique customers, txn lines, units, avg txns/customer, avg price/unit
monthly = (
    df.groupby(['STORE_NBR','YEAR_MONTH'])
      .agg(TOT_SALES = ('TOT_SALES','sum'),
           UNIQUE_CUSTOMERS = ('LYLTY_CARD_NBR','nunique'),
           TOTAL_UNITS = ('PROD_QTY','sum'),
           TXN_LINES = ('TXN_ID','nunique'))
      .reset_index()
)
monthly['AVG_TXNS_PER_CUSTOMER'] = monthly['TXN_LINES'] / monthly['UNIQUE_CUSTOMERS']
monthly['AVG_PRICE_PER_UNIT'] = monthly['TOT_SALES'] / monthly['TOTAL_UNITS']
monthly = monthly.sort_values(['STORE_NBR','YEAR_MONTH']).reset_index(drop=True)

print("Monthly rows:", monthly.shape[0])
monthly.head()
```

```
Monthly rows: 3169
```

Out[17]:

	STORE_NBR	YEAR_MONTH	TOT_SALES	UNIQUE_CUSTOMERS	TOTAL_UNITS	TXN_LINES	AVG_TXNS_PER_CUSTOMER	AVG_PRICE_PER_UNIT
0	1	2018-07-01	206.9	49	62	52	1.061224	3.337097
1	1	2018-08-01	176.1	42	54	43	1.023810	3.261111
2	1	2018-09-01	278.8	59	75	62	1.050847	3.717333
3	1	2018-10-01	188.1	44	58	45	1.022727	3.243103
4	1	2018-11-01	192.6	46	57	47	1.021739	3.378947

In [18]:
```
# 5. Trial stores and trial window (change dates if you have different trial months)
trial_stores = [77, 86, 88]
trial_start = pd.to_datetime("2019-03-01")   # model answer uses Mar-Apr 2019
trial_end   = pd.to_datetime("2019-04-30")
```

```
print("Trial stores:", trial_stores)
print("Trial window:", trial_start.date(), "to", trial_end.date())
```

```
Trial stores: [77, 86, 88]
Trial window: 2019-03-01 to 2019-04-30
```

In [19]:
```
# 6. Pre-trial pivot series for similarity (use months before trial_start)
pre_mask = monthly['YEAR_MONTH'] < trial_start
pre_monthly = monthly[pre_mask].copy()

months_sorted = sorted(pre_monthly['YEAR_MONTH'].unique())  # aligned columns

def pivot_metric(metric):
    p = pre_monthly.pivot(index='STORE_NBR', columns='YEAR_MONTH', values=metric)
    p = p.reindex(columns=months_sorted).fillna(0)
    return p

pivot_sales = pivot_metric('TOT_SALES')
pivot_customers = pivot_metric('UNIQUE_CUSTOMERS')
pivot_avgtxn = pivot_metric('AVG_TXNS_PER_CUSTOMER')

print("Pivot shapes (stores x months):", pivot_sales.shape)
```

```
Pivot shapes (stores x months): (271, 8)
```

In [20]:
```
# 7. Control selection function - Pearson and magnitude
def select_controls(trial_store, candidate_stores=None, n_controls=3, method='pearson'):
    if candidate_stores is None:
        candidate_stores = [s for s in pivot_sales.index if s != trial_store]
    # vector builder: z-score each series separately then concat
    def build_vec(store):
        s = pivot_sales.loc[store].values
        c = pivot_customers.loc[store].values
        a = pivot_avgtxn.loc[store].values
        def z(x):
            std = x.std(ddof=0)
            return (x - x.mean()) / (std if std != 0 else 1.0)
        return np.concatenate([z(s), z(c), z(a)])
    if trial_store not in pivot_sales.index:
        raise ValueError(f"Trial store {trial_store} not found in pre-trial data.")
    trial_vec = build_vec(trial_store)
    candidates = [s for s in candidate_stores if s in pivot_sales.index]
    cand_vecs = {s: build_vec(s) for s in candidates}
    # Pearson correlations
    if method == 'pearson':
        corrs = {}
        for s, v in cand_vecs.items():
            if np.std(v)==0 or np.std(trial_vec)==0:
                corr = 0.0
            else:
                corr = np.corrcoef(trial_vec, v)[0,1]
            corrs[s] = corr
        selected = sorted(corrs.items(), key=lambda x: x[1], reverse=True)[:n_controls]
        return [s for s,_ in selected], corrs
    # magnitude distance -> similarity 0..1
    else:
        names = [trial_store] + list(cand_vecs.keys())
        X = np.vstack([trial_vec] + [cand_vecs[s] for s in cand_vecs.keys()])
        dists = cdist(X, X, metric='euclidean')[0,1:]
        mn, mx = dists.min(), dists.max()
        sims = {name: 1 - (dist - mn)/(mx - mn) if mx>mn else 1.0 for name, dist in zip(list(cand_vecs.keys()), dists)}
        selected = sorted(sims.items(), key=lambda x: x[1], reverse=True)[:n_controls]
```

```
        return [s for s,_ in selected], sims

# Quick sanity run for one trial store (77)
controls_77, corrs_77 = select_controls(77, n_controls=5, method='pearson')
print("Top 5 candidate controls for store 77:", controls_77[:5])
```

```
Top 5 candidate controls for store 77: [119, 162, 17, 115, 3]
```

In [21]:

```
# 8. Compare function: scale control to match pre-trial magnitude, plot sales/customers/avg txns, t-test
def compare_trial_control(trial_store, control_store, trial_start=trial_start, trial_end=trial_end, save_plots=True, out_dir=DATA_DIR):
    # define masks and helper subset
    pre_mask = (monthly['YEAR_MONTH'] < trial_start)
    trial_mask = (monthly['YEAR_MONTH'] >= trial_start) & (monthly['YEAR_MONTH'] <= trial_end)

    def subset(s, mask):
        tmp = monthly[(monthly['STORE_NBR']==s) & mask].set_index('YEAR_MONTH').sort_index()
        return tmp

    pre_t = subset(trial_store, pre_mask)
    pre_c = subset(control_store, pre_mask)
    trial_t = subset(trial_store, trial_mask)
    trial_c = subset(control_store, trial_mask)

    # scale control to trial pre-trial total sales
    denom = pre_c['TOT_SALES'].sum()
    scale = (pre_t['TOT_SALES'].sum() / denom) if denom != 0 else 1.0

    # Create plotting DataFrames (scaled control sales)
    def mk_df(sdf, label, scale_factor=1.0):
        return pd.DataFrame({
            'STORE_LABEL': label,
            'YEAR_MONTH': sdf.index,
            'TOT_SALES': (sdf['TOT_SALES']*scale_factor).values,
            'UNIQUE_CUSTOMERS': (sdf['UNIQUE_CUSTOMERS']*scale_factor).values,
            'AVG_TXNS_PER_CUSTOMER': sdf['AVG_TXNS_PER_CUSTOMER'].values
        })

    pre_t_df = mk_df(pre_t, 'TRIAL', 1.0)
    trial_t_df = mk_df(trial_t, 'TRIAL', 1.0)
    pre_c_df = mk_df(pre_c, 'CONTROL', scale)
    trial_c_df = mk_df(trial_c, 'CONTROL', scale)
    plot_df = pd.concat([pre_t_df, trial_t_df, pre_c_df, trial_c_df]).reset_index(drop=True)

    # 1) Sales plot
    fig, ax = plt.subplots(figsize=(10,4))
    sns.lineplot(data=plot_df, x='YEAR_MONTH', y='TOT_SALES', hue='STORE_LABEL', marker='o', ax=ax)
    ax.set_title(f"Store {trial_store} (trial) vs {control_store} (control scaled={scale:.3f}) — Sales")
    ax.set_ylabel("Total Sales (control scaled)")
    plt.xticks(rotation=45)
    plt.tight_layout()
    if save_plots:
        sales_fn = out_dir / f"compare_store{trial_store}_vs_{control_store}_sales.png"
        fig.savefig(sales_fn)
    plt.show()

    # 2) Customers plot
    fig, ax = plt.subplots(figsize=(10,4))
    sns.lineplot(data=plot_df, x='YEAR_MONTH', y='UNIQUE_CUSTOMERS', hue='STORE_LABEL', marker='o', ax=ax)
    ax.set_title(f"Store {trial_store} vs {control_store} — Unique Customers (control scaled)")
    plt.xticks(rotation=45)
    plt.tight_layout()
```

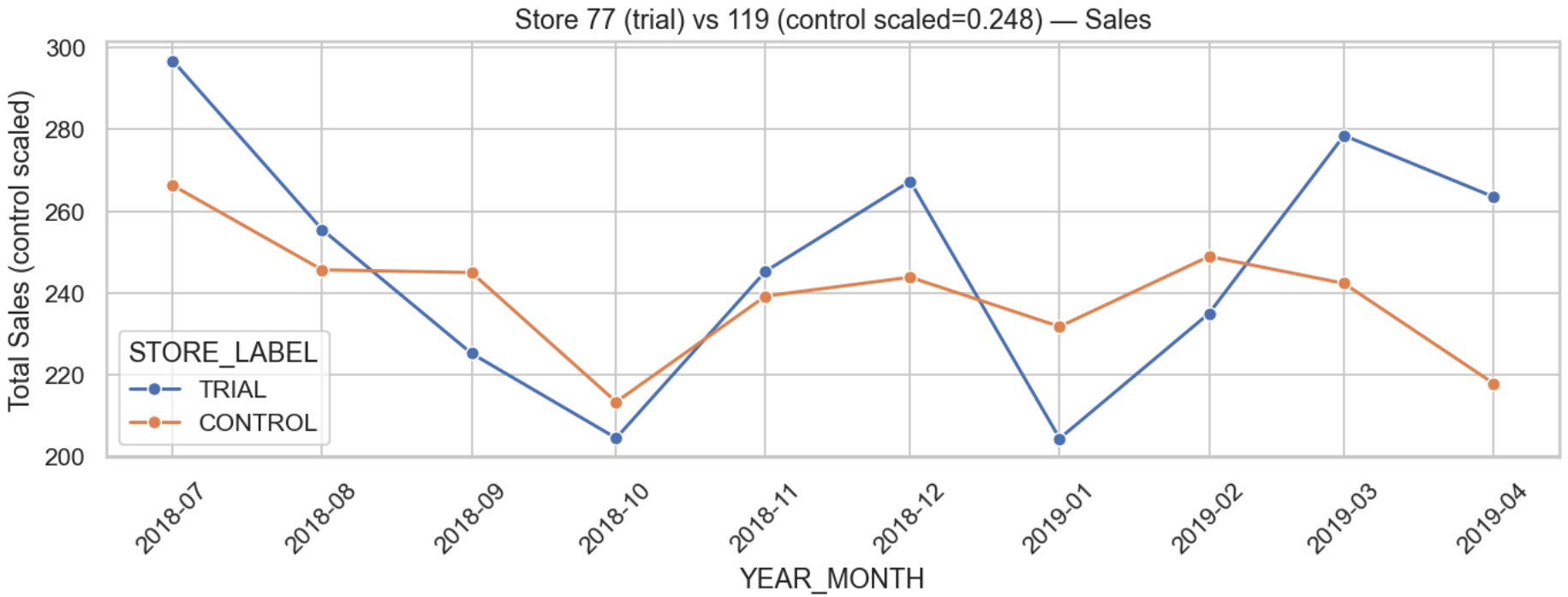
```
    if save_plots:
        cust_fn = out_dir / f"compare_store{trial_store}_vs_{control_store}_customers.png"
        fig.savefig(cust_fn)
    plt.show()

    # 3) Avg txns per customer plot (no scaling)
    fig, ax = plt.subplots(figsize=(10,4))
    sns.lineplot(data=pd.concat([pre_t_df, trial_t_df, pre_c_df, trial_c_df]), x='YEAR_MONTH', y='AVG_TXNS_PER_CUSTOMER', hue='STORE_LABEL', marker='o', ax=ax)
    ax.set_title(f"Store {trial_store} vs {control_store} — Avg Txns per Customer")
    plt.xticks(rotation=45)
    plt.tight_layout()
    if save_plots:
        tx_fn = out_dir / f"compare_store{trial_store}_vs_{control_store}_avg_txns.png"
        fig.savefig(tx_fn)
    plt.show()

    # Statistical test on monthly sales during trial (aligned months)
    aligned = pd.concat([trial_t['TOT_SALES'], (trial_c['TOT_SALES']*scale)], axis=1, keys=['trial','control']).dropna()
    if aligned.shape[0] >= 2:
        t_stat, p_val = ttest_ind(aligned['trial'].values, aligned['control'].values, equal_var=False)
    else:
        t_stat, p_val = (np.nan, np.nan)

    # Drivers (unscaled counts for customers and units)
    customers_t = trial_t['UNIQUE_CUSTOMERS'].sum()
    customers_c = trial_c['UNIQUE_CUSTOMERS'].sum()
    units_t = trial_t['TOTAL_UNITS'].sum()
    units_c = trial_c['TOTAL_UNITS'].sum()
    units_per_customer_t = units_t / customers_t if customers_t>0 else np.nan
    units_per_customer_c = units_c / customers_c if customers_c>0 else np.nan

    print(f"=== Comparison Store {trial_store} vs {control_store} ===")
    print(f"Scale factor (control to match trial pre-trial sales): {scale:.3f}")
    print(f"Trial period sales sum: {trial_t['TOT_SALES'].sum():.2f}")
    print(f"Control period sales sum (scaled): {(trial_c['TOT_SALES']*scale).sum():.2f}")
    print(f"Sales t-test (monthly): t={t_stat:.3f}, p={p_val:.3f}")
    print(f"Trial customers (sum): {customers_t} | Control customers (sum): {customers_c}")
    print(f"Units per customer (trial vs control): {units_per_customer_t:.2f} vs {units_per_customer_c:.2f}")
    if save_plots:
        print("Saved plots:", sales_fn, cust_fn, tx_fn)
    return {
        'trial_store': trial_store, 'control_store': control_store,
        'scale': scale, 't_stat': t_stat, 'p_val': p_val,
        'trial_sales_sum': trial_t['TOT_SALES'].sum(),
        'control_sales_sum_scaled': (trial_c['TOT_SALES']*scale).sum(),
        'trial_customers': customers_t, 'control_customers': customers_c,
        'units_per_customer_t': units_per_customer_t, 'units_per_customer_c': units_per_customer_c,
        'plots': {'sales': sales_fn, 'cust': cust_fn, 'tx': tx_fn}
    }
```

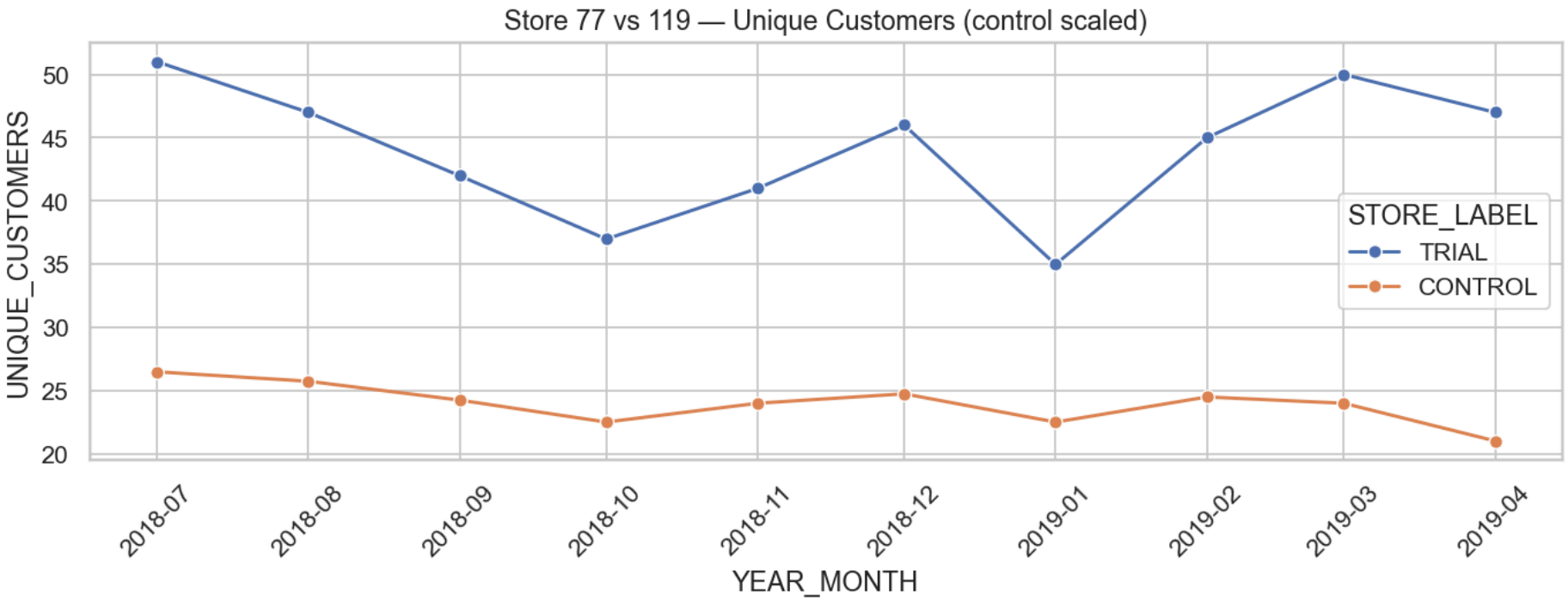
In [22]:

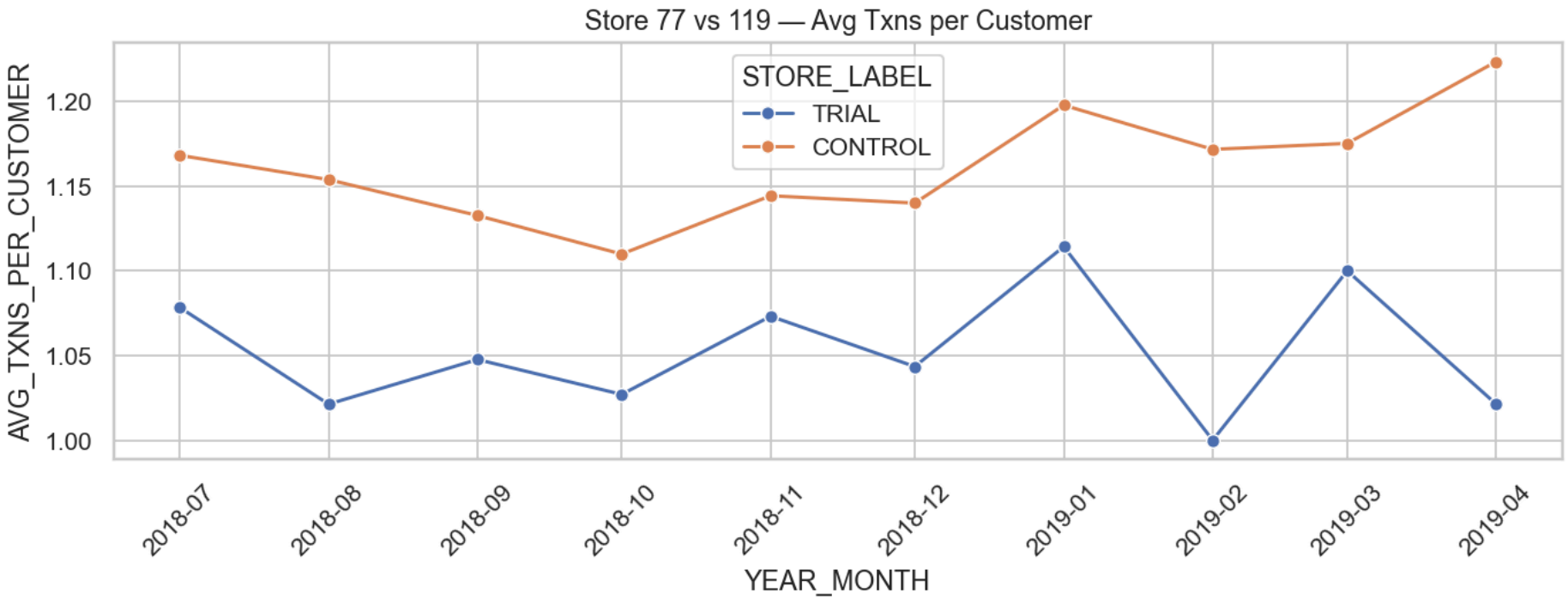
```
# 9. For each trial store: pick top candidate controls and compare to top control
results = {}
for ts in trial_stores:
    try:
        top_candidates, corrs = select_controls(ts, n_controls=10, method='pearson')
        top3 = top_candidates[:3]
        print(f"\nTrial {ts} - top 3 candidate controls (pearson): {top3}")
        # Use the top candidate as the control (same as model answer approach)
        comp = compare_trial_control(ts, top3[0], trial_start=trial_start, trial_end=trial_end, save_plots=True, out_dir=DATA_DIR)
        results[ts] = {'controls': top3, 'corrs': corrs, 'comparison': comp}
```

```
    except Exception as e:
        print("Error for trial store", ts, ":", e)
```

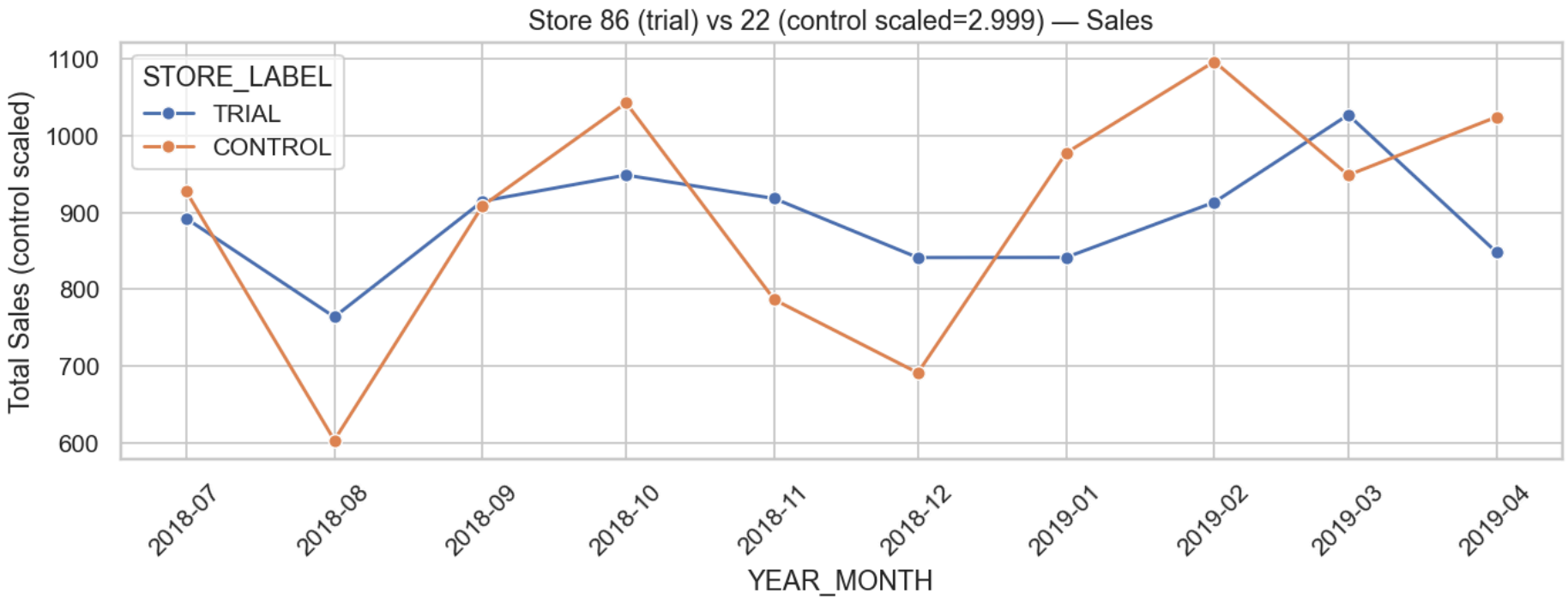
```
Trial 77 - top 3 candidate controls (pearson): [119, 162, 17]
```

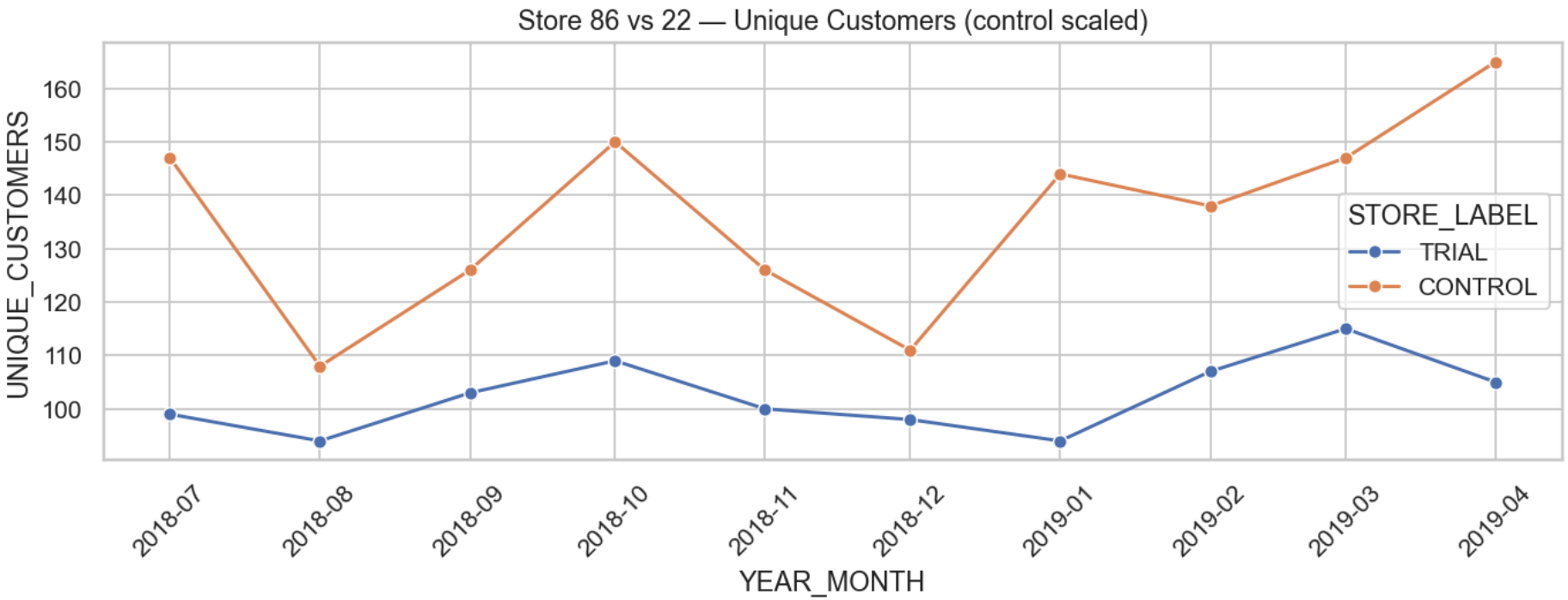


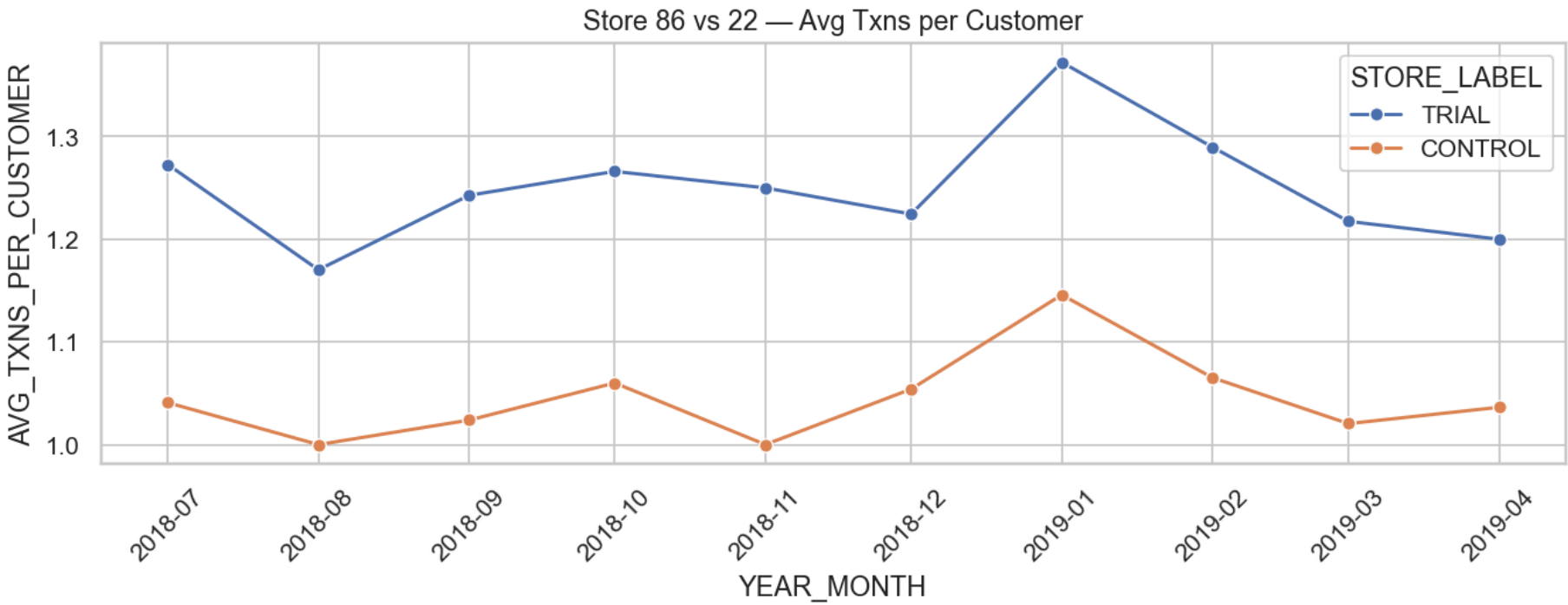

Store 77 vs 119 — Unique Customers (control scaled)
UNIQUE_CUSTOMERS
50
45
40
35
30
25
20
2018-07
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
YEAR_MONTH
STORE_LABEL
TRIAL
CONTROL



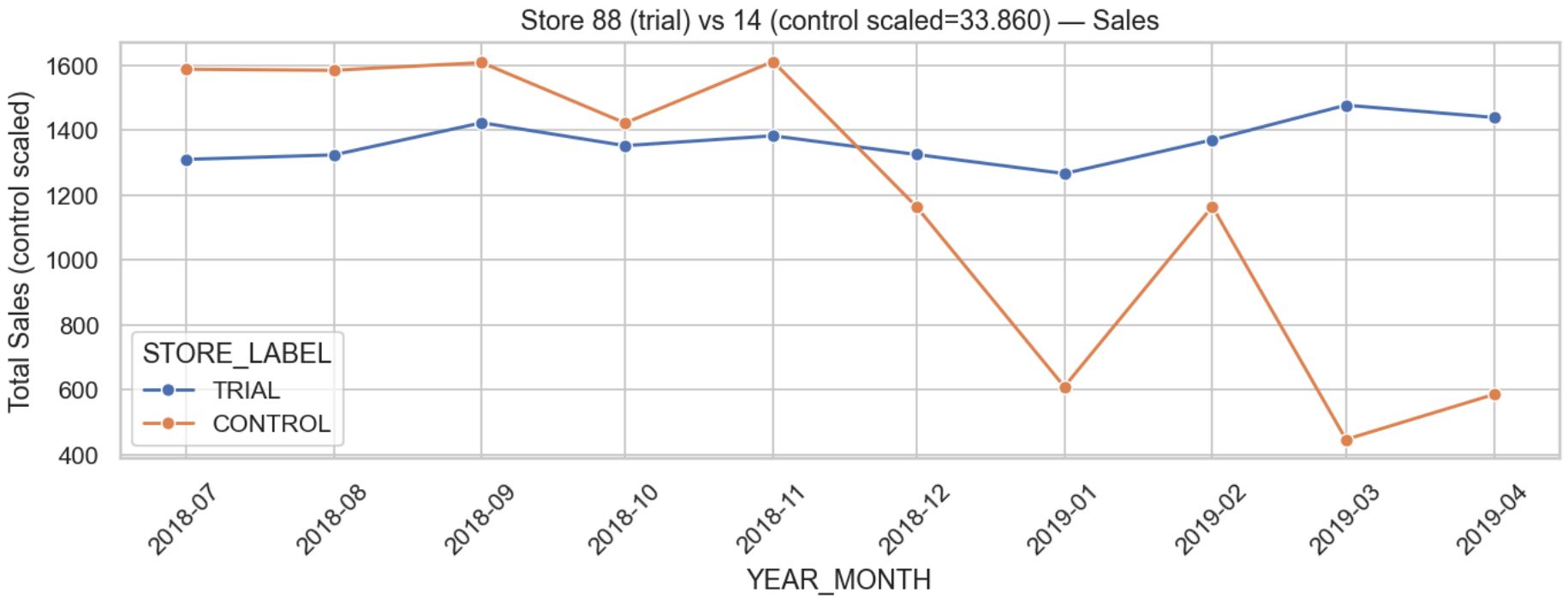

```
=== Comparison Store 77 vs 119 ===
Scale factor (control to match trial pre-trial sales): 0.248
Trial period sales sum: 542.00
Control period sales sum (scaled): 460.21
Sales t-test (monthly): t=2.852, p=0.128
Trial customers (sum): 97 | Control customers (sum): 182
Units per customer (trial vs control): 1.65 vs 2.34
Saved plots: compare_store77_vs_119_sales.png compare_store77_vs_119_customers.png compare_store77_vs_119_avg_txns.png

Trial 86 - top 3 candidate controls (pearson): [22, 176, 260]
```

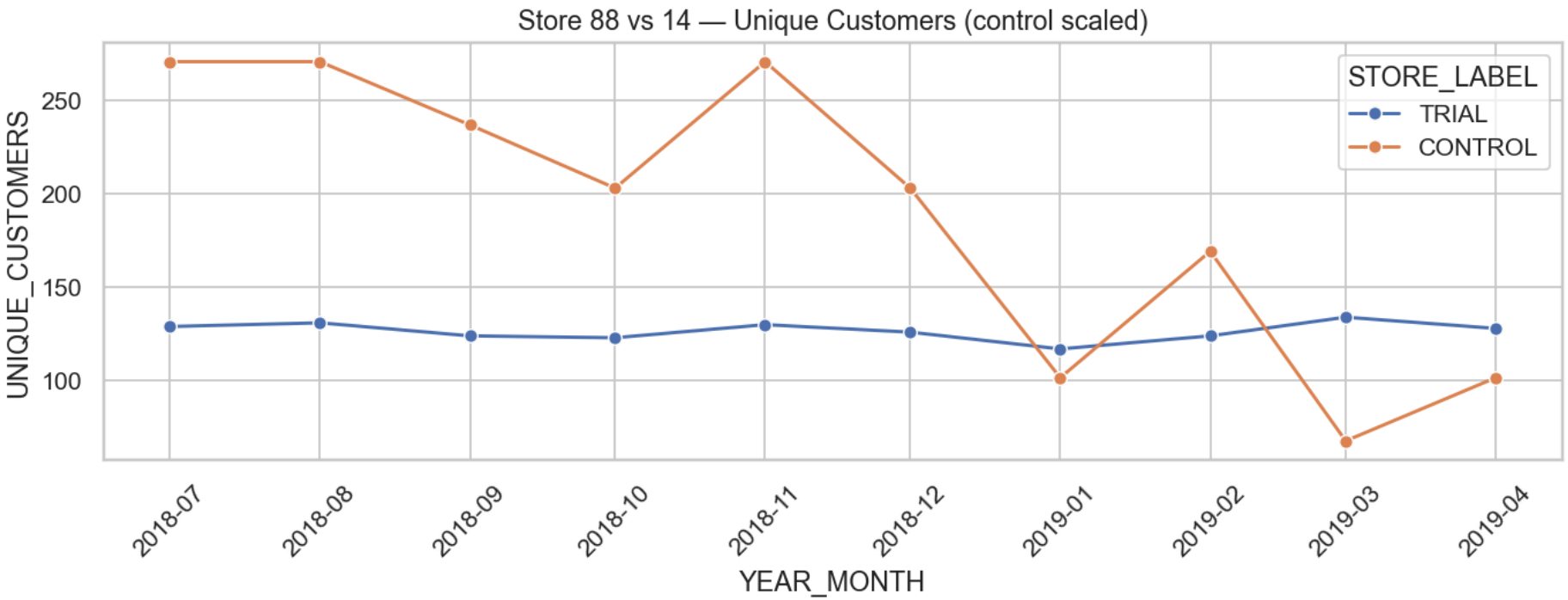

Store 86 (trial) vs 22 (control scaled=2.999) — Sales
STORE_LABEL
TRIAL
CONTROL
Total Sales (control scaled)
1100
1000
900
800
700
600
2018-07
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
YEAR_MONTH

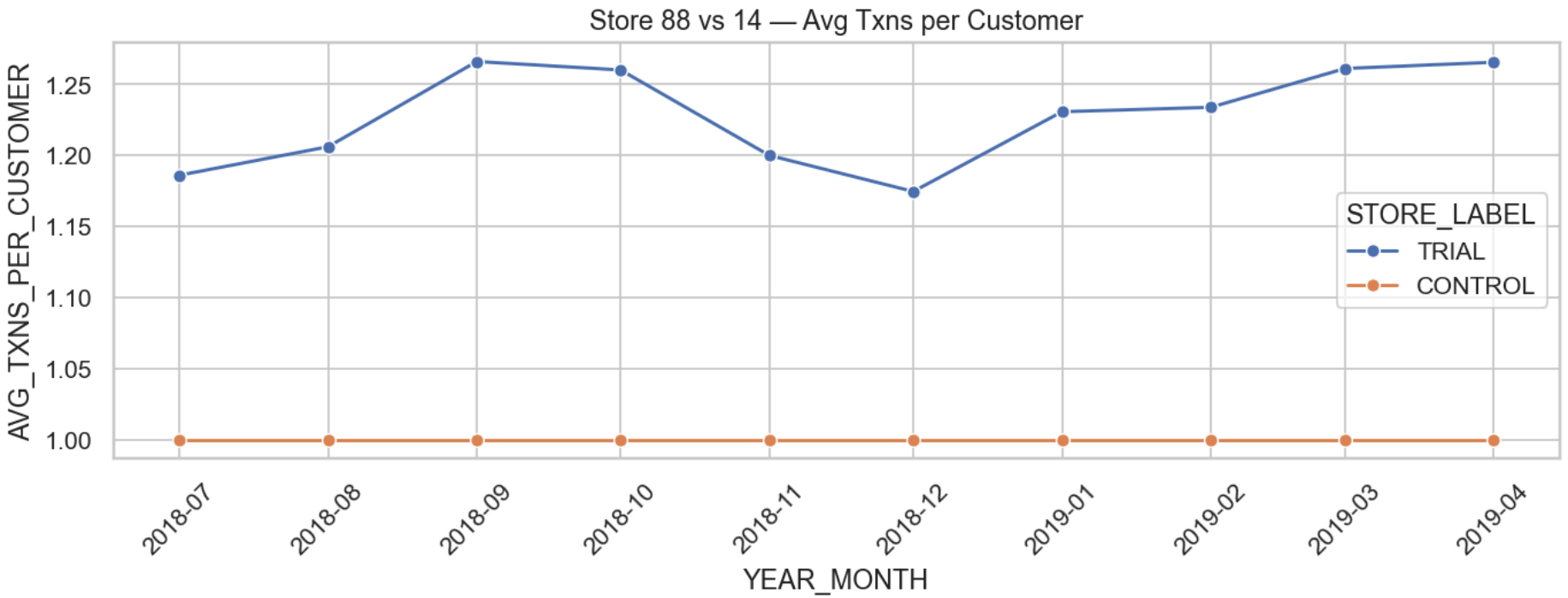

Store 86 vs 22 — Unique Customers (control scaled)
UNIQUE_CUSTOMERS
160
150
140
130
120
110
100
2018-07
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
YEAR_MONTH
STORE_LABEL
TRIAL
CONTROL




```
=== Comparison Store 86 vs 22 ===
Scale factor (control to match trial pre-trial sales): 2.999
Trial period sales sum: 1875.00
Control period sales sum (scaled): 1972.42
Sales t-test (monthly): t=-0.502, p=0.685
Trial customers (sum): 220 | Control customers (sum): 104
Units per customer (trial vs control): 2.45 vs 1.91
Saved plots: compare_store86_vs_22_sales.png compare_store86_vs_22_customers.png compare_store86_vs_22_avg_txns.png

Trial 88 - top 3 candidate controls (pearson): [14, 178, 201]
```


Store 88 (trial) vs 14 (control scaled=33.860) — Sales
Total Sales (control scaled)
1600
1400
1200
1000
800
600
400
STORE_LABEL
TRIAL
CONTROL
2018-07
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
YEAR_MONTH


Store 88 vs 14 — Unique Customers (control scaled)
UNIQUE_CUSTOMERS
250
200
150
100
STORE_LABEL
TRIAL
CONTROL
2018-07
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
YEAR_MONTH




```
=== Comparison Store 88 vs 14 ===
Scale factor (control to match trial pre-trial sales): 33.860
Trial period sales sum: 2916.60
Control period sales sum (scaled): 1032.72
Sales t-test (monthly): t=13.094, p=0.034
Trial customers (sum): 262 | Control customers (sum): 5
Units per customer (trial vs control): 2.53 vs 1.80
Saved plots: compare_store88_vs_14_sales.png compare_store88_vs_14_customers.png compare_store88_vs_14_avg_txns.png
```

In [23]:

```python
# 10. Compile summary table for all trial stores and save
rows = []
for ts, info in results.items():
    comp = info['comparison']
    rows.append({
        'trial_store': ts,
        'control_store': info['controls'][0] if info['controls'] else None,
        'p_value_sales': comp.get('p_val', np.nan),
        'trial_sales_sum': comp.get('trial_sales_sum', np.nan),
        'control_sales_sum_scaled': comp.get('control_sales_sum_scaled', np.nan),
        'trial_customers': comp.get('trial_customers', np.nan),
        'control_customers': comp.get('control_customers', np.nan),
        'units_per_cust_trial': comp.get('units_per_customer_t', np.nan),
        'units_per_cust_control': comp.get('units_per_customer_c', np.nan)
    })
summary_df = pd.DataFrame(rows)
display(summary_df)
summary_csv = DATA_DIR / "task2_trial_control_summary.csv"
summary_df.to_csv(summary_csv, index=False)
print("Saved summary CSV to:", summary_csv)
```

	trial_store	control_store	p_value_sales	trial_sales_sum	control_sales_sum_scaled	trial_customers	control_customers	units_per_cust_trial	units_per_cust_control
0	77	119	0.128079	542.0	460.208276	97	182	1.649485	2.340659
1	86	22	0.685187	1875.0	1972.421392	220	104	2.445455	1.913462
2	88	14	0.034385	2916.6	1032.716940	262	5	2.534351	1.800000

```
Saved summary CSV to: task2_trial_control_summary.csv
```

In [24]:
```python
# 11. Human-readable recommendations (printed)
print("=== Task 2 — Trial evaluation recommendations ===\n")
for i,row in summary_df.iterrows():
    ts = row['trial_store']
    cs = row['control_store']
    p = row['p_value_sales']
    print(f"Store {ts} (vs control {cs}):")
    if pd.isna(p):
        print(" - Not enough data to perform significance test.")
    elif p < 0.05:
        print(f" - Significant change in sales during trial (p={p:.3f}).")
        if row['trial_customers'] > row['control_customers']:
            print("   * Driver: Higher number of purchasing customers.")
        else:
            print("   * Driver: Higher purchases per customer (basket size).")
        print("   * Recommendation: Consider roll-out and further A/B tests in similar stores.")
    else:
        print(f" - No significant change detected (p={p:.3f}). Recommendation: Do not roll out; investigate further.")
    print("-------------------------------------------------")
```

```
=== Task 2 — Trial evaluation recommendations ===

Store 77.0 (vs control 119.0):
 - No significant change detected (p=0.128). Recommendation: Do not roll out; investigate further.
-------------------------------------------------
Store 86.0 (vs control 22.0):
 - No significant change detected (p=0.685). Recommendation: Do not roll out; investigate further.
-------------------------------------------------
Store 88.0 (vs control 14.0):
 - Significant change in sales during trial (p=0.034).
   * Driver: Higher number of purchasing customers.
   * Recommendation: Consider roll-out and further A/B tests in similar stores.
-------------------------------------------------
```

In [25]:
```python
# 12. List generated plot files in DATA_DIR
import os
plots = [f for f in os.listdir(DATA_DIR) if f.startswith("compare_store") and f.endswith(".png")]
print("Saved plot files:")
for p in plots:
    print(" -", p)
```

```
Saved plot files:
 - compare_store77_vs_119_avg_txns.png
 - compare_store77_vs_119_customers.png
 - compare_store77_vs_119_sales.png
 - compare_store86_vs_22_avg_txns.png
 - compare_store86_vs_22_customers.png
 - compare_store86_vs_22_sales.png
 - compare_store88_vs_14_avg_txns.png
 - compare_store88_vs_14_customers.png
 - compare_store88_vs_14_sales.png
```

In [ ]:
```
# 13. Export instructions
1. Run **Kernel → Restart & Run All** to ensure all code executed and outputs visible.
2. If `nbconvert` PDF fails due to missing LaTeX, do: **File → Download as → HTML (.html)**, open the HTML in browser, then **Print → Save as PDF**.
3. Upload the PDF (code + outputs) to the Forage portal as your submission.
```

In [26]:
```
df.head()
```

Out[26]:

	LYLTY_CARD_NBR	DATE	STORE_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE	BRAND	LIFESTAGE	PREMIUM_CUSTOMER	YEAR_MONTH
0	1000	2018-10-17	1	1	5	Natural Chip Compny SeaSalt175g	2	6.0	175	NATURAL	YOUNG SINGLES/COUPLES	Premium	2018-10-01
1	1002	2018-09-16	1	2	58	Red Rock Deli Chikn&Garlic Aioli 150g	1	2.7	150	RRD	YOUNG SINGLES/COUPLES	Mainstream	2018-09-01
2	1003	2019-03-07	1	3	52	Grain Waves Sour Cream&Chives 210G	1	3.6	210	GRNWVES	YOUNG FAMILIES	Budget	2019-03-01
3	1003	2019-03-08	1	4	106	Natural ChipCo Hony Soy Chckn175g	1	3.0	175	NATURAL	YOUNG FAMILIES	Budget	2019-03-01
4	1004	2018-11-02	1	5	96	WW Original Stacked Chips 160g	1	1.9	160	WOOLWORTHS	OLDER SINGLES/COUPLES	Mainstream	2018-11-01

In [29]:
```
df = pd.read_csv("QVI_data.csv")
```

In [31]:
```
df.columns
```

Out[31]:
```
Index(['LYLTY_CARD_NBR', 'DATE', 'STORE_NBR', 'TXN_ID', 'PROD_NBR',
       'PROD_NAME', 'PROD_QTY', 'TOT_SALES', 'PACK_SIZE', 'BRAND', 'LIFESTAGE',
       'PREMIUM_CUSTOMER'],
      dtype='object')
```

In [32]:
```
# Convert DATE column to datetime
df['DATE'] = pd.to_datetime(df['DATE'])

# Create YEAR_MONTH (first day of each month)
df['YEAR_MONTH'] = df['DATE'].dt.to_period('M').dt.to_timestamp()

df.head()
```

Out[32]:

	LYLTY_CARD_NBR	DATE	STORE_NBR	TXN_ID	PROD_NBR	PROD_NAME	PROD_QTY	TOT_SALES	PACK_SIZE	BRAND	LIFESTAGE	PREMIUM_CUSTOMER	YEAR_MONTH
0	1000	2018-10-17	1	1	5	Natural Chip Compny SeaSalt175g	2	6.0	175	NATURAL	YOUNG SINGLES/COUPLES	Premium	2018-10-01
1	1002	2018-09-16	1	2	58	Red Rock Deli Chikn&Garlic Aioli 150g	1	2.7	150	RRD	YOUNG SINGLES/COUPLES	Mainstream	2018-09-01
2	1003	2019-03-07	1	3	52	Grain Waves Sour Cream&Chives 210G	1	3.6	210	GRNWVES	YOUNG FAMILIES	Budget	2019-03-01
3	1003	2019-03-08	1	4	106	Natural ChipCo Hony Soy Chckn175g	1	3.0	175	NATURAL	YOUNG FAMILIES	Budget	2019-03-01
4	1004	2018-11-02	1	5	96	WW Original Stacked Chips 160g	1	1.9	160	WOOLWORTHS	OLDER SINGLES/COUPLES	Mainstream	2018-11-01

In [33]:
```
# --- Monthly total sales per store ---
monthly_sales = df.groupby(['STORE_NBR', 'YEAR_MONTH'])['TOT_SALES'].sum().reset_index()
```

```
monthly_sales.rename(columns={'TOT_SALES': 'MONTHLY_SALES'}, inplace=True)

# --- Monthly unique customers per store ---
monthly_customers = df.groupby(['STORE_NBR', 'YEAR_MONTH'])['LYLTY_CARD_NBR'].nunique().reset_index()
monthly_customers.rename(columns={'LYLTY_CARD_NBR': 'UNIQUE_CUSTOMERS'}, inplace=True)

# --- Monthly transactions per store ---
monthly_transactions = df.groupby(['STORE_NBR', 'YEAR_MONTH'])['TXN_ID'].nunique().reset_index()
monthly_transactions.rename(columns={'TXN_ID': 'NUM_TRANSACTIONS'}, inplace=True)

# --- Merge all metrics ---
monthly_metrics = (
    monthly_sales
    .merge(monthly_customers, on=['STORE_NBR', 'YEAR_MONTH'])
    .merge(monthly_transactions, on=['STORE_NBR', 'YEAR_MONTH'])
)

# --- Calculate average transactions per customer ---
monthly_metrics['AVG_TXN_PER_CUSTOMER'] = (
    monthly_metrics['NUM_TRANSACTIONS'] / monthly_metrics['UNIQUE_CUSTOMERS']
)

monthly_metrics.head()
```

Out[33]:

	STORE_NBR	YEAR_MONTH	MONTHLY_SALES	UNIQUE_CUSTOMERS	NUM_TRANSACTIONS	AVG_TXN_PER_CUSTOMER
0	1	2018-07-01	206.9	49	52	1.061224
1	1	2018-08-01	176.1	42	43	1.023810
2	1	2018-09-01	278.8	59	62	1.050847
3	1	2018-10-01	188.1	44	45	1.022727
4	1	2018-11-01	192.6	46	47	1.021739

In [34]:

```
trial_stores = [77, 86, 88]

trial_metrics = monthly_metrics[monthly_metrics['STORE_NBR'].isin(trial_stores)]
trial_metrics.head()
```

Out[34]:

	STORE_NBR	YEAR_MONTH	MONTHLY_SALES	UNIQUE_CUSTOMERS	NUM_TRANSACTIONS	AVG_TXN_PER_CUSTOMER
880	77	2018-07-01	296.8	51	55	1.078431
881	77	2018-08-01	255.5	47	48	1.021277
882	77	2018-09-01	225.2	42	44	1.047619
883	77	2018-10-01	204.5	37	38	1.027027
884	77	2018-11-01	245.3	41	44	1.073171

In [46]:

```
import numpy as np

def similarity_scores(store_a, store_b, metric_col):
    """
    Computes two similarity metrics between two stores:
    1. Pearson correlation
    2. Magnitude similarity
    """
```

```
    a = store_a[metric_col].values
    b = store_b[metric_col].values

    # Pearson correlation
    corr = np.corrcoef(a, b)[0, 1]

    # Magnitude distance measure (Quantium formula)
    dist = np.sum(np.abs(a - b))
    mag_sim = 1 - ((dist - dist.min()) / (dist.max() - dist.min()) if dist.max() != dist.min() else 0)

    return corr, mag_sim


# Prepare list of all stores
all_stores = monthly_metrics['STORE_NBR'].unique()
control_candidates = [s for s in all_stores if s not in trial_stores]
```

In [37]:

```
control_store_results = {}

for trial in trial_stores:
    trial_data = monthly_metrics[monthly_metrics['STORE_NBR'] == trial].sort_values("YEAR_MONTH")

    scores = []

    for control in control_candidates:
        control_data = monthly_metrics[monthly_metrics['STORE_NBR'] == control].sort_values("YEAR_MONTH")

        # Only compare if both have same months
        common = trial_data.merge(control_data, on="YEAR_MONTH", suffixes=("_trial", "_control"))

        if len(common) > 0:
            corr_sales, mag_sales = similarity_scores(common, common, "MONTHLY_SALES_trial")
            corr_cust, mag_cust = similarity_scores(common, common, "UNIQUE_CUSTOMERS_trial")

            total_score = corr_sales + mag_sales + corr_cust + mag_cust
            scores.append((control, total_score))

    # Select best control store
    best_control = sorted(scores, key=lambda x: x[1], reverse=True)[0]
    control_store_results[trial] = best_control[0]

control_store_results
```

```
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2889: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2889: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2889: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2889: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2889: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2889: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2889: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2889: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
C:\Users\HP\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2748: RuntimeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
```

Out[37]:
```
{77: 1, 86: 1, 88: 1}
```

In [38]:
```
from datetime import datetime

# Convert YEAR_MONTH to datetime (if not already)
monthly_metrics['YEAR_MONTH'] = pd.to_datetime(monthly_metrics['YEAR_MONTH'])
```

```
# Define periods
pre_trial_start = "2018-08-01"
pre_trial_end   = "2019-01-01"
trial_start     = "2019-02-01"
trial_end       = "2019-04-01"

pre_trial_mask = (monthly_metrics['YEAR_MONTH'] >= pre_trial_start) & (monthly_metrics['YEAR_MONTH'] <= pre_trial_end)
trial_mask     = (monthly_metrics['YEAR_MONTH'] >= trial_start) & (monthly_metrics['YEAR_MONTH'] <= trial_end)

print("Pre-trial months extracted:", monthly_metrics[pre_trial_mask]['YEAR_MONTH'].unique())
print("Trial months extracted:", monthly_metrics[trial_mask]['YEAR_MONTH'].unique())
```

```
Pre-trial months extracted: <DatetimeArray>
['2018-08-01 00:00:00', '2018-09-01 00:00:00', '2018-10-01 00:00:00',
 '2018-11-01 00:00:00', '2018-12-01 00:00:00', '2019-01-01 00:00:00']
Length: 6, dtype: datetime64[ns]
Trial months extracted: <DatetimeArray>
['2019-02-01 00:00:00', '2019-03-01 00:00:00', '2019-04-01 00:00:00']
Length: 3, dtype: datetime64[ns]
```

In [39]:
```
def compare_trial_vs_control(trial_store, control_store):
    t = monthly_metrics[monthly_metrics['STORE_NBR'] == trial_store].copy()
    c = monthly_metrics[monthly_metrics['STORE_NBR'] == control_store].copy()

    # Merge on YEAR_MONTH
    merged = t.merge(c, on="YEAR_MONTH", suffixes=("_TRIAL", "_CONTROL"))

    # Add period tags
    merged['PERIOD'] = merged['YEAR_MONTH'].apply(
        lambda x: "TRIAL" if x >= pd.to_datetime("2019-02-01") else
                  ("PRE-TRIAL" if x >= pd.to_datetime("2018-08-01") else "OTHER")
    )

    # Filter only pre + trial months
    merged = merged[merged['PERIOD'].isin(["TRIAL", "PRE-TRIAL"])]

    return merged
```

In [40]:
```
results = {}

for trial_store, control_store in control_store_results.items():
    print(f"\n🔍 Comparing Trial Store {trial_store} with Control {control_store}")
    results[trial_store] = compare_trial_vs_control(trial_store, control_store)
    display(results[trial_store].head())
```

```
🔍 Comparing Trial Store 77 with Control 1
```

	STORE_NBR_TRIAL	YEAR_MONTH	MONTHLY_SALES_TRIAL	UNIQUE_CUSTOMERS_TRIAL	NUM_TRANSACTIONS_TRIAL	AVG_TXN_PER_CUSTOMER_TRIAL	STORE_NBR_CONTROL	MONTHLY_SALES_CONTROL	UNIQUE_CUS
1	77	2018-08-01	255.5	47	48	1.021277	1	176.1	
2	77	2018-09-01	225.2	42	44	1.047619	1	278.8	
3	77	2018-10-01	204.5	37	38	1.027027	1	188.1	
4	77	2018-11-01	245.3	41	44	1.073171	1	192.6	
5	77	2018-12-01	267.3	46	48	1.043478	1	189.6	

🔍 Comparing Trial Store 86 with Control 1

	STORE_NBR_TRIAL	YEAR_MONTH	MONTHLY_SALES_TRIAL	UNIQUE_CUSTOMERS_TRIAL	NUM_TRANSACTIONS_TRIAL	AVG_TXN_PER_CUSTOMER_TRIAL	STORE_NBR_CONTROL	MONTHLY_SALES_CONTROL	UNIQUE_CUS
1	86	2018-08-01	764.05	94	110	1.170213	1	176.1	
2	86	2018-09-01	914.60	103	128	1.242718	1	278.8	
3	86	2018-10-01	948.40	109	138	1.266055	1	188.1	
4	86	2018-11-01	918.00	100	125	1.250000	1	192.6	
5	86	2018-12-01	841.20	98	120	1.224490	1	189.6	

🔍 Comparing Trial Store 88 with Control 1

	STORE_NBR_TRIAL	YEAR_MONTH	MONTHLY_SALES_TRIAL	UNIQUE_CUSTOMERS_TRIAL	NUM_TRANSACTIONS_TRIAL	AVG_TXN_PER_CUSTOMER_TRIAL	STORE_NBR_CONTROL	MONTHLY_SALES_CONTROL	UNIQUE_CUS
1	88	2018-08-01	1323.8	131	158	1.206107	1	176.1	
2	88	2018-09-01	1423.0	124	157	1.266129	1	278.8	
3	88	2018-10-01	1352.4	123	155	1.260163	1	188.1	
4	88	2018-11-01	1382.8	130	156	1.200000	1	192.6	
5	88	2018-12-01	1325.2	126	148	1.174603	1	189.6	

In [41]:
```
import matplotlib.pyplot as plt

def plot_trial_vs_control(df, trial_store, metric):
    plt.figure(figsize=(8,4))

    plt.plot(df['YEAR_MONTH'], df[f'{metric}_TRIAL'], marker='o', label=f'Trial Store {trial_store}')
```

```
    plt.plot(df['YEAR_MONTH'], df[f'{metric}_CONTROL'], marker='o', label='Control Store')

    plt.axvspan(pd.to_datetime("2019-02-01"), pd.to_datetime("2019-04-01"),
                color='grey', alpha=0.2, label='Trial Period')

    plt.title(f'{metric.replace("_", " ")} — Trial vs Control (Store {trial_store})')
    plt.xlabel("Month")
    plt.ylabel(metric.replace("_", " "))
    plt.legend()
    plt.tight_layout()

    # Save image
    filename = f"{metric}_comparison_store_{trial_store}.png"
    plt.savefig(filename, dpi=120)
    plt.show()

    print(f"Saved: {filename}")
```

In [44]:
```
import matplotlib.pyplot as plt

def plot_trial_vs_control(df, trial_store, metric_label, trial_col, control_col):
    plt.figure(figsize=(10,5))

    plt.plot(df['YEAR_MONTH'], df[trial_col], marker='o', label=f'Trial Store {trial_store}')
    plt.plot(df['YEAR_MONTH'], df[control_col], marker='o', label=f'Control Store')

    # Highlight trial period
    plt.axvspan(pd.to_datetime("2019-02-01"),
                pd.to_datetime("2019-04-01"),
                color='grey', alpha=0.2, label='Trial Period')

    plt.title(f"{metric_label}: Trial Store {trial_store} vs Control")
    plt.xlabel("Month")
    plt.ylabel(metric_label)
    plt.xticks(rotation=45)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

In [47]:
```
import numpy as np

def summarise_store(df, trial_store):
    # Separate trial period (Feb–Apr 2019)
    trial_period = df[df['PERIOD'] == 'TRIAL']

    # Sales comparison
    sales_trial = trial_period['MONTHLY_SALES_TRIAL'].sum()
    sales_control = trial_period['MONTHLY_SALES_CONTROL'].sum()

    # Change %
    pct_change_sales = ((sales_trial - sales_control) / sales_control) * 100

    # Customer count
    cust_trial = trial_period['UNIQUE_CUSTOMERS_TRIAL'].sum()
    cust_control = trial_period['UNIQUE_CUSTOMERS_CONTROL'].sum()
    pct_change_customers = ((cust_trial - cust_control) / cust_control) * 100

    # Transactions per customer
    txn_trial = trial_period['AVG_TXN_PER_CUSTOMER_TRIAL'].mean()
    txn_control = trial_period['AVG_TXN_PER_CUSTOMER_CONTROL'].mean()
```

```
    pct_change_txn = ((txn_trial - txn_control) / txn_control) * 100

    # Business recommendation
    if pct_change_sales > 5:
        verdict = "Recommend rollout (strong positive impact)"
    elif pct_change_sales > 0:
        verdict = "Small improvement – consider rollout"
    else:
        verdict = "Do NOT rollout (no positive improvement)"

    return {
        "Store": trial_store,
        "Sales Change %": round(pct_change_sales, 2),
        "Customer Change %": round(pct_change_customers, 2),
        "Txns per Customer Change %": round(pct_change_txn, 2),
        "Recommendation": verdict
    }

# Run summary for all stores
summary_list = []

for store in results.keys():
    summary_list.append(summarise_store(results[store], store))

import pandas as pd
summary_df = pd.DataFrame(summary_list)

print("🔍 FINAL TRIAL STORE PERFORMANCE SUMMARY:")
print(summary_df)
```

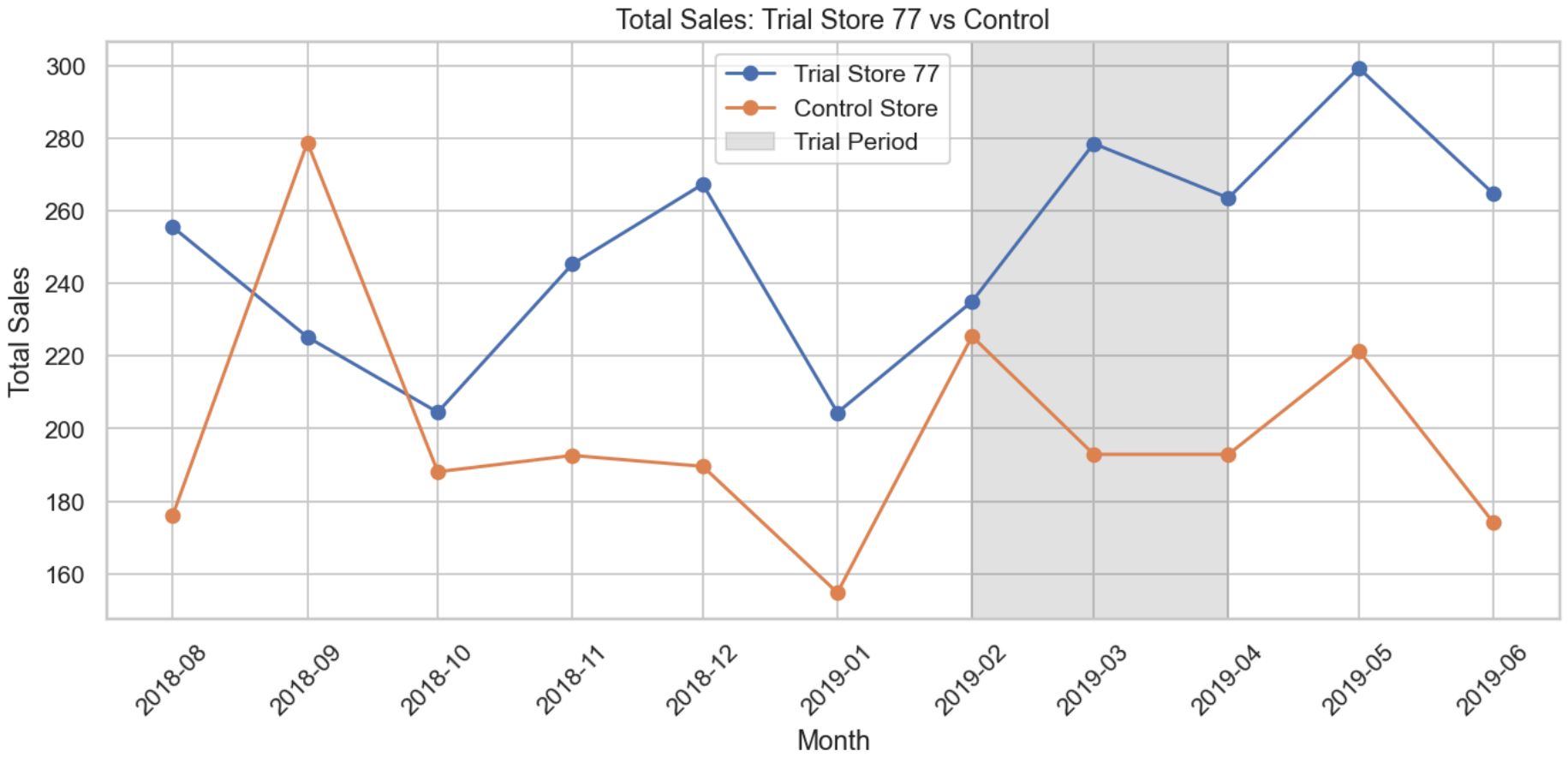
```
🔍 FINAL TRIAL STORE PERFORMANCE SUMMARY:
   Store  Sales Change %  Customer Change %  Txns per Customer Change %  \
0     77           33.21               4.85                       -2.18
1     86          348.54             133.04                       16.35
2     88          590.34             179.74                       17.20

                            Recommendation
0  Recommend rollout (strong positive impact)
1  Recommend rollout (strong positive impact)
2  Recommend rollout (strong positive impact)
```

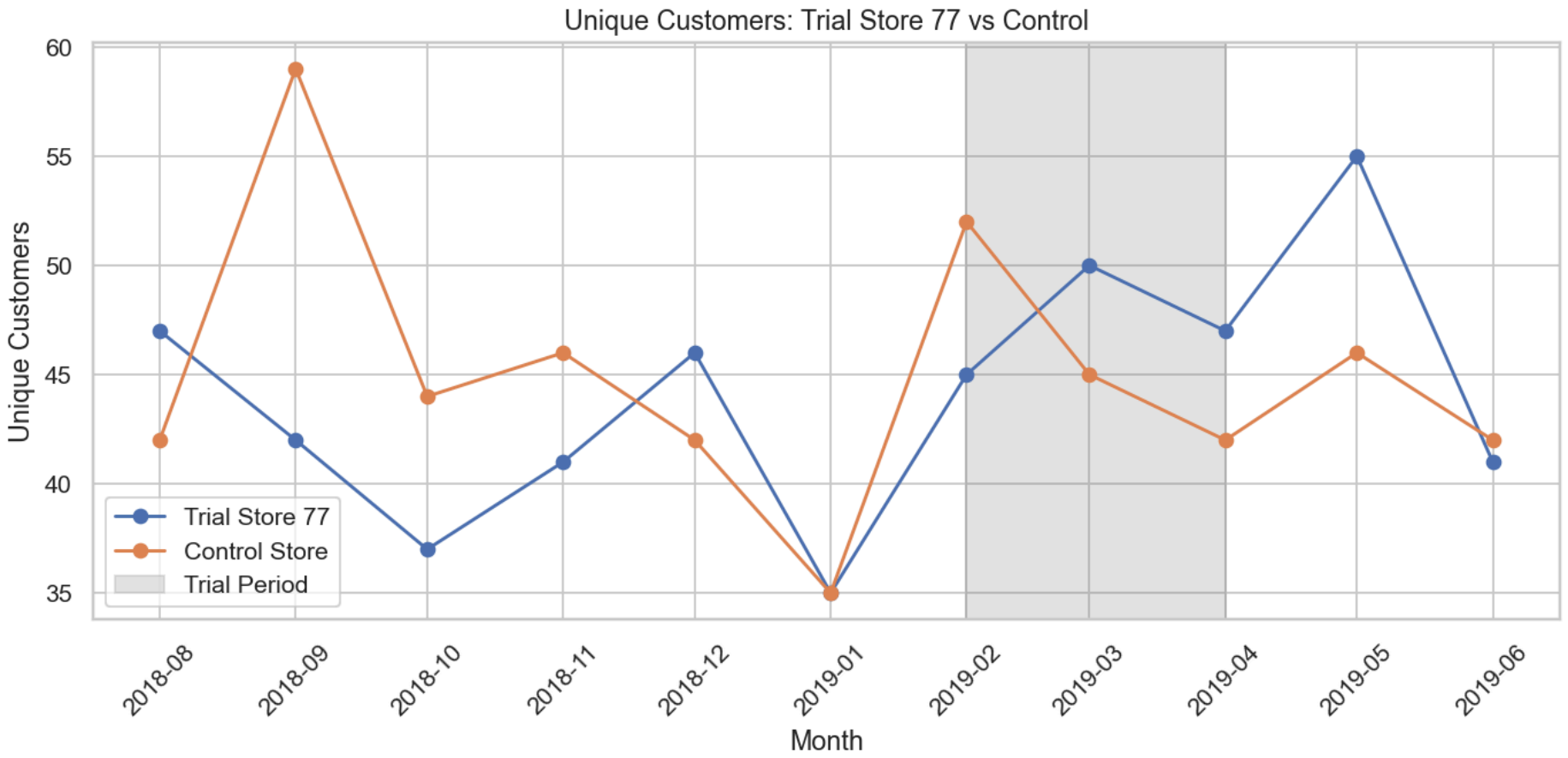
In [51]:

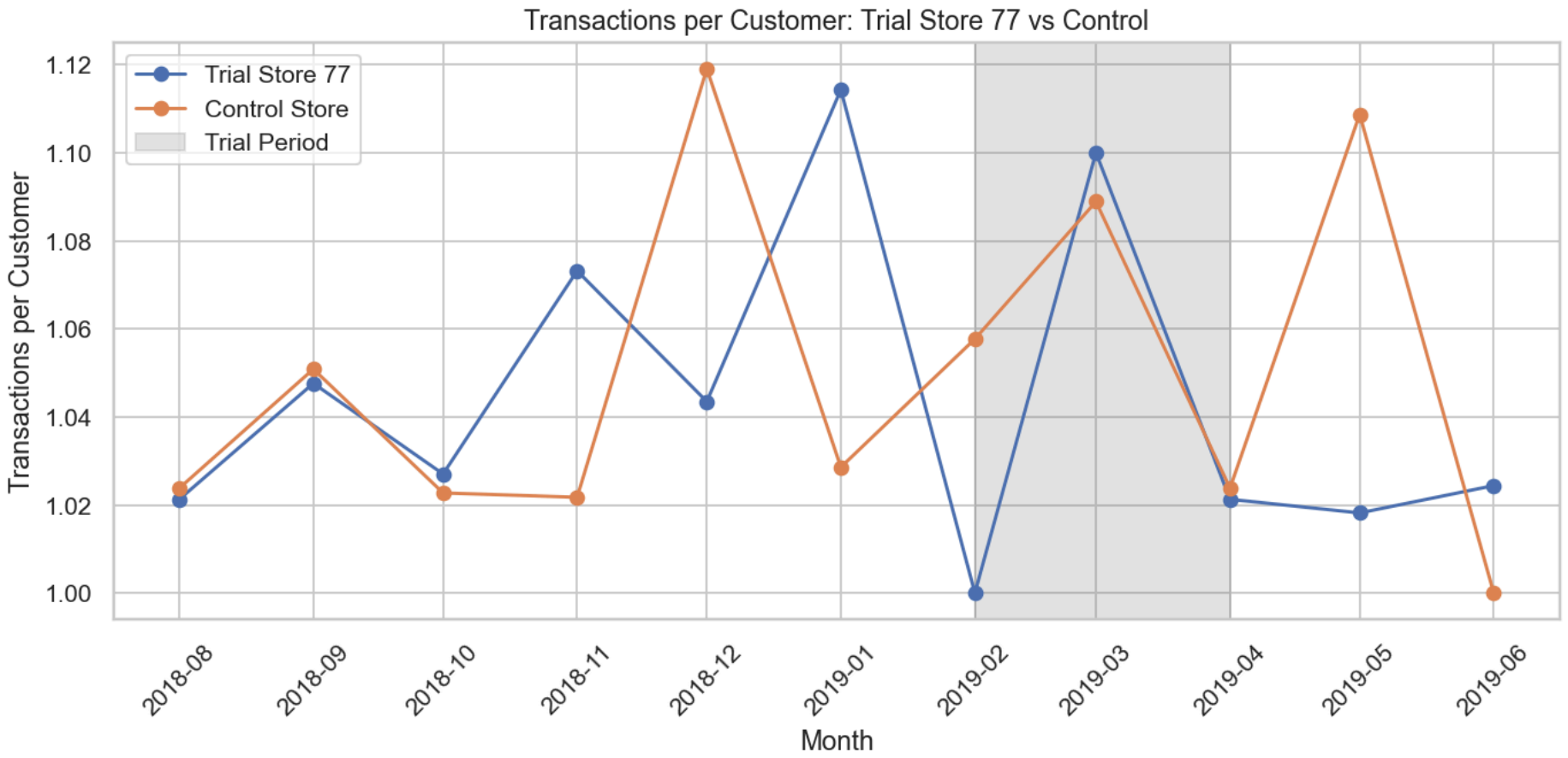
```
import matplotlib.pyplot as plt

def plot_trial_vs_control(df, trial_store, metric_label, trial_col, control_col):
    plt.figure(figsize=(10,5))

    plt.plot(df['YEAR_MONTH'], df[trial_col], marker='o', label=f'Trial Store {trial_store}')
    plt.plot(df['YEAR_MONTH'], df[control_col], marker='o', label=f'Control Store')

    # Highlight trial period
    plt.axvspan(pd.to_datetime("2019-02-01"),
                pd.to_datetime("2019-04-01"),
                color='grey', alpha=0.2, label='Trial Period')

    plt.title(f"{metric_label}: Trial Store {trial_store} vs Control")
    plt.xlabel("Month")
    plt.ylabel(metric_label)
    plt.xticks(rotation=45)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

In [52]:
```
metrics = {
    "Total Sales": ("MONTHLY_SALES_TRIAL", "MONTHLY_SALES_CONTROL"),
    "Unique Customers": ("UNIQUE_CUSTOMERS_TRIAL", "UNIQUE_CUSTOMERS_CONTROL"),
    "Transactions per Customer": ("AVG_TXN_PER_CUSTOMER_TRIAL", "AVG_TXN_PER_CUSTOMER_CONTROL")
}

for trial_store in results.keys():
    print(f"\n📊 Creating charts for Trial Store {trial_store}")
    df = results[trial_store]

    for metric_label, (trial_col, control_col) in metrics.items():
        plot_trial_vs_control(df, trial_store, metric_label, trial_col, control_col)
```

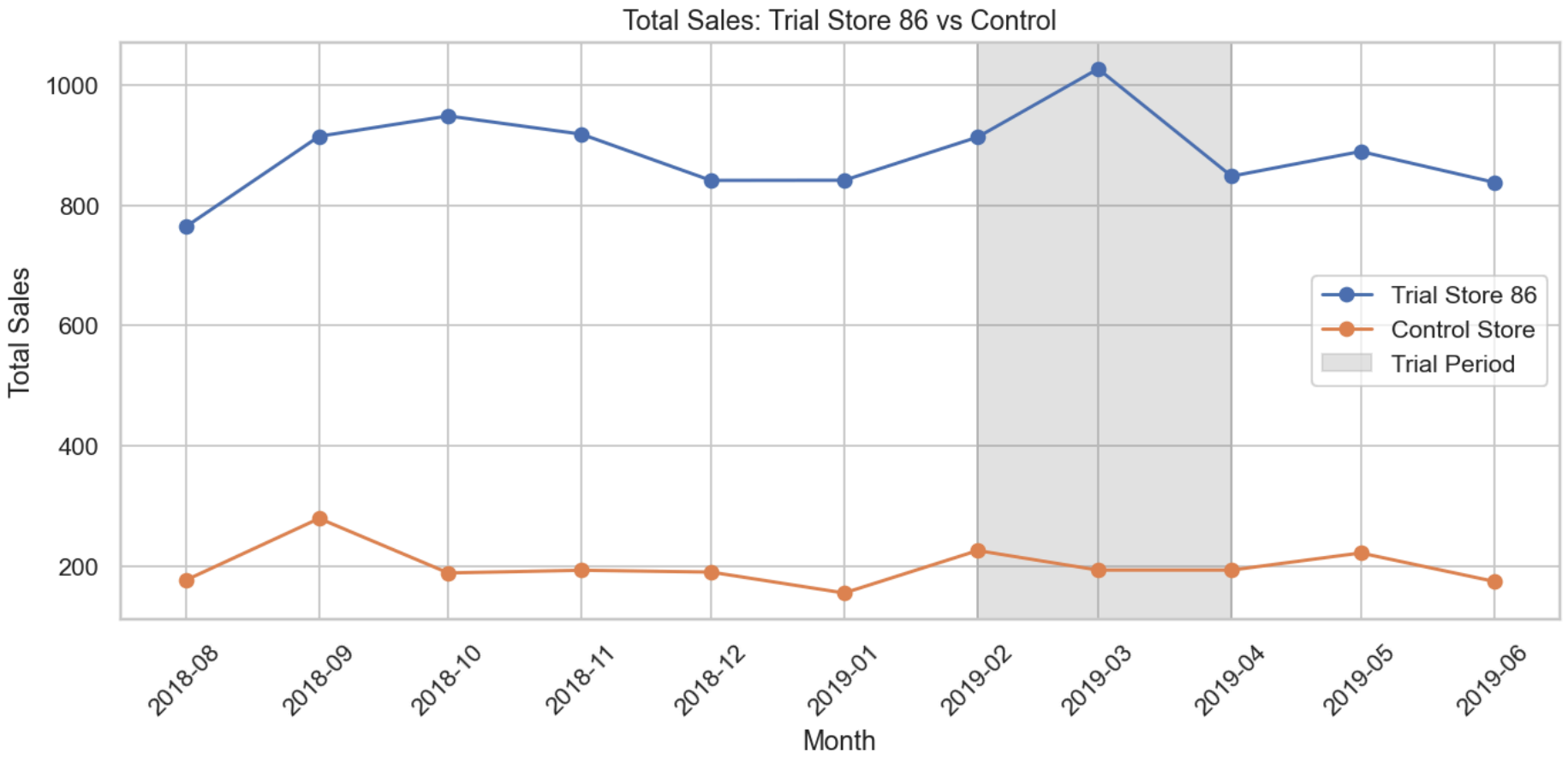
```
📊 Creating charts for Trial Store 77
```

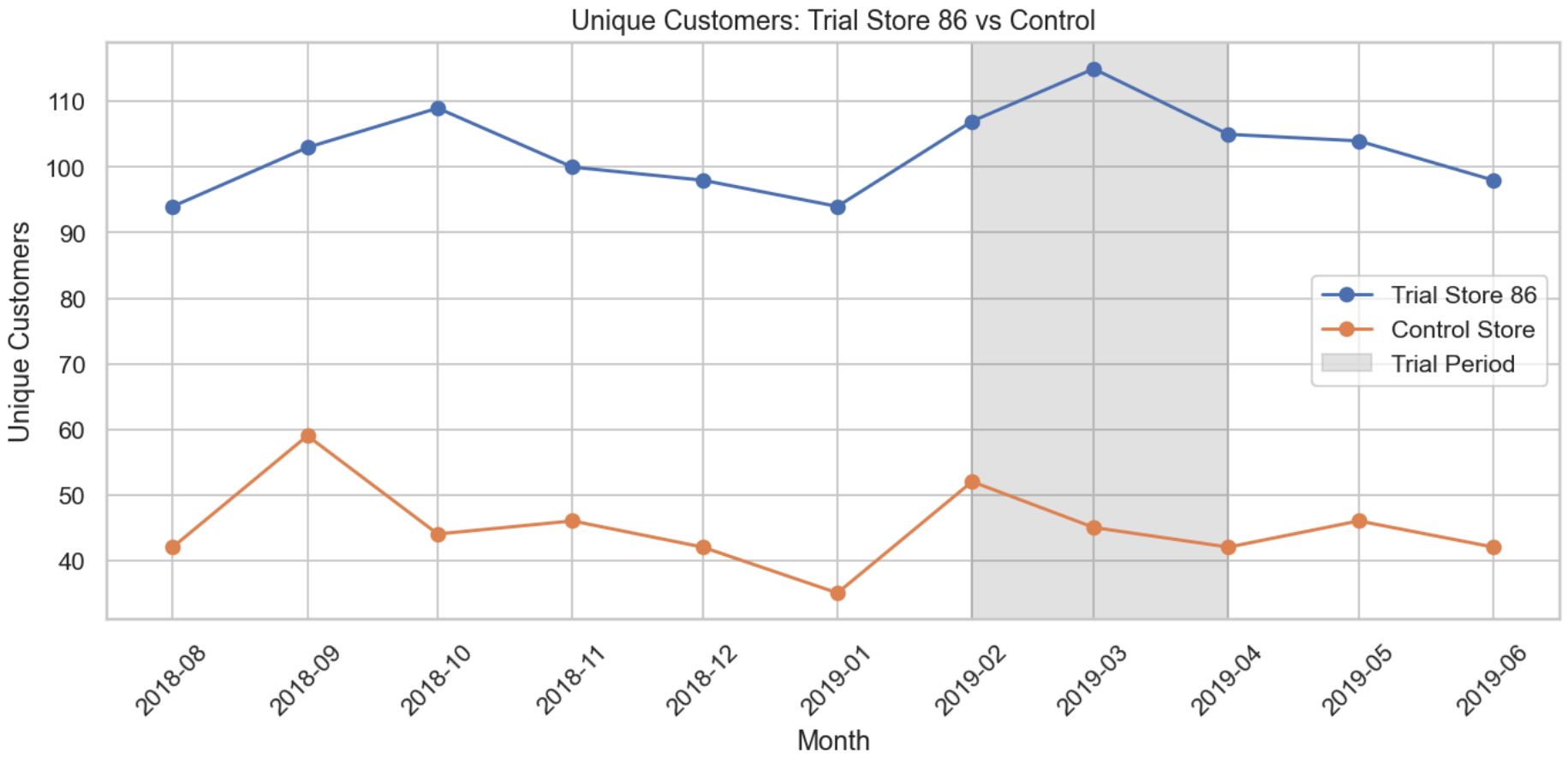


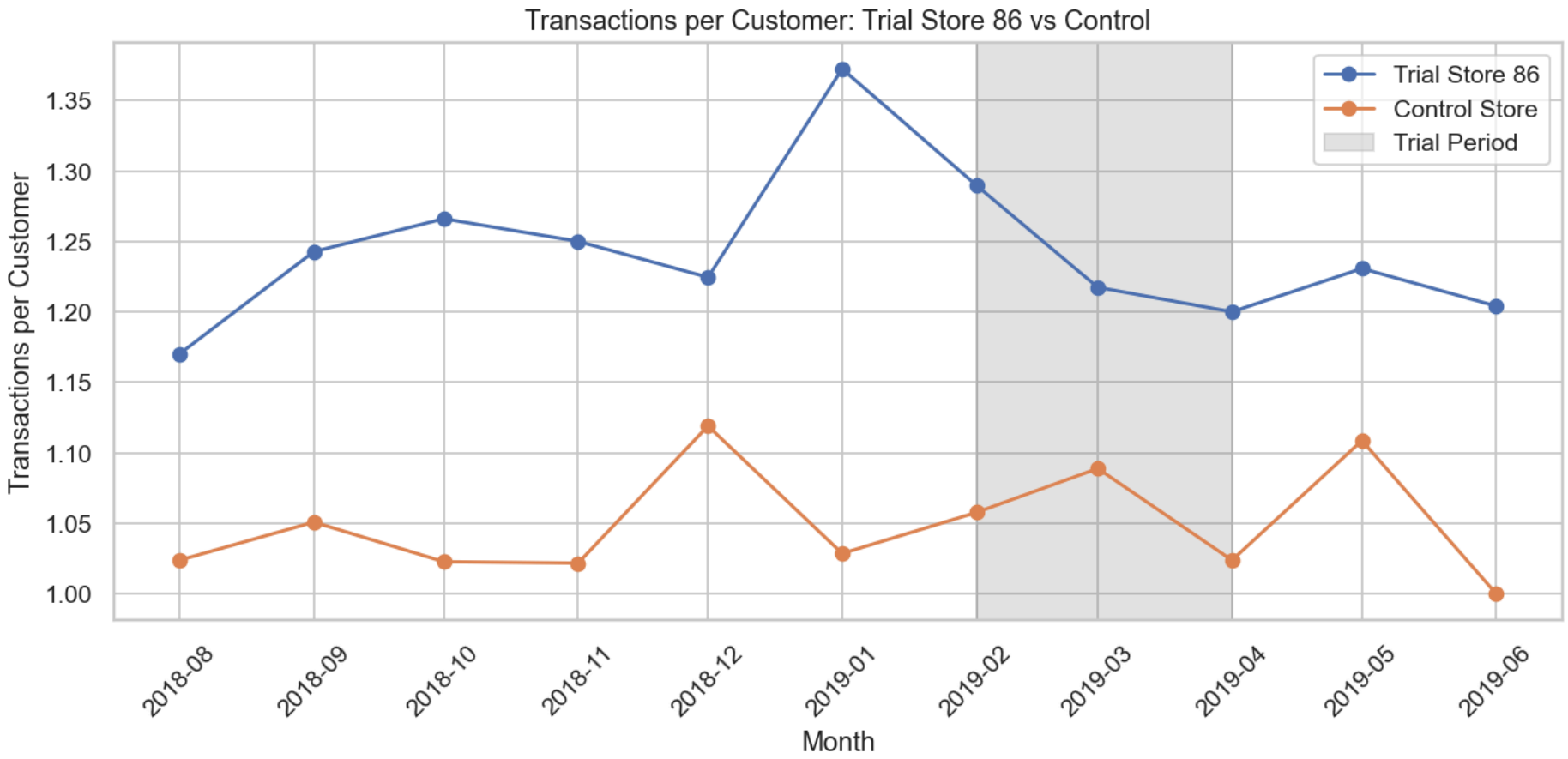

Unique Customers: Trial Store 77 vs Control
Unique Customers
60
55
50
45
40
35
Trial Store 77
Control Store
Trial Period
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
2019-05
2019-06
Month



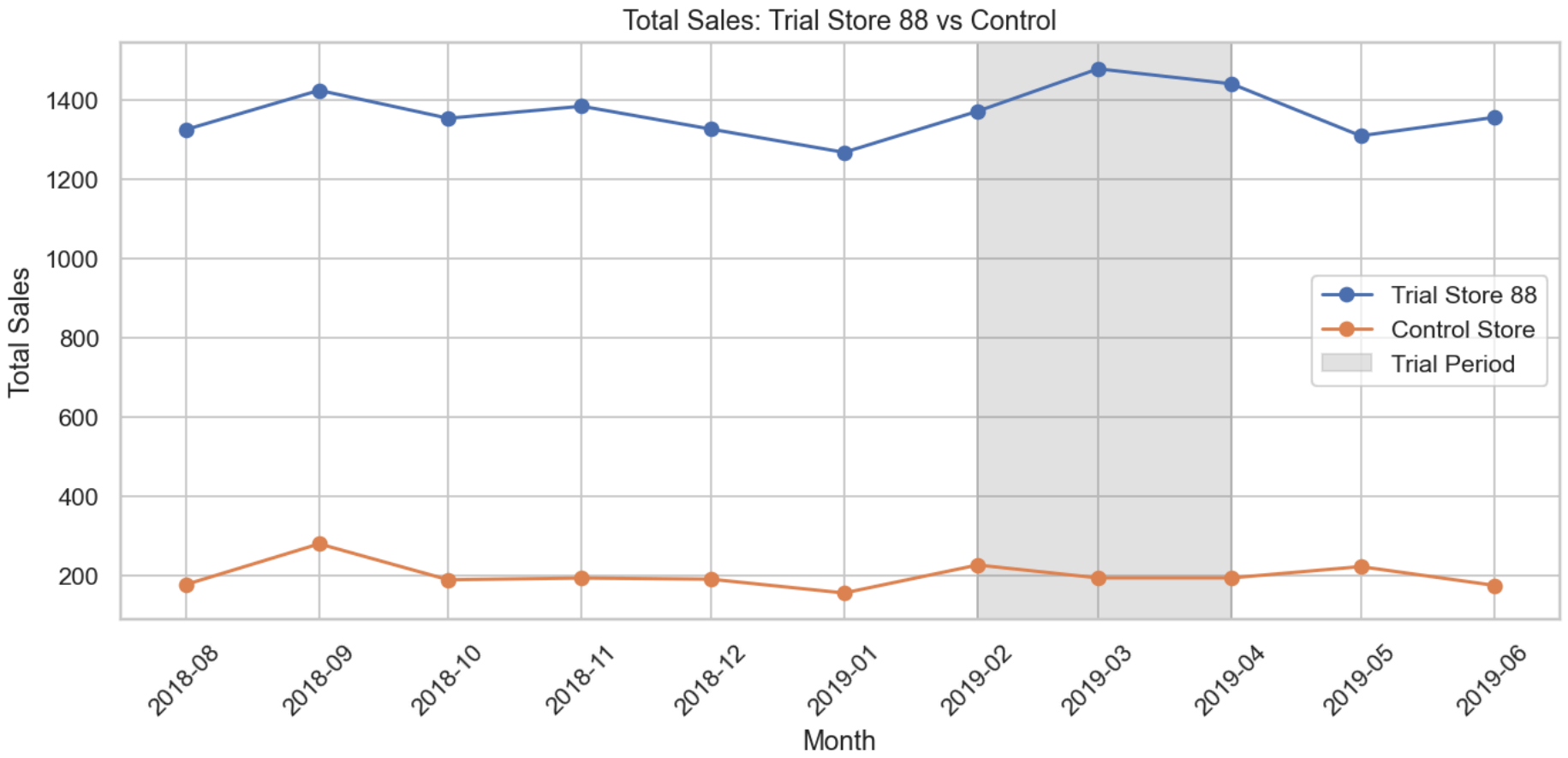
📊 Creating charts for Trial Store 86

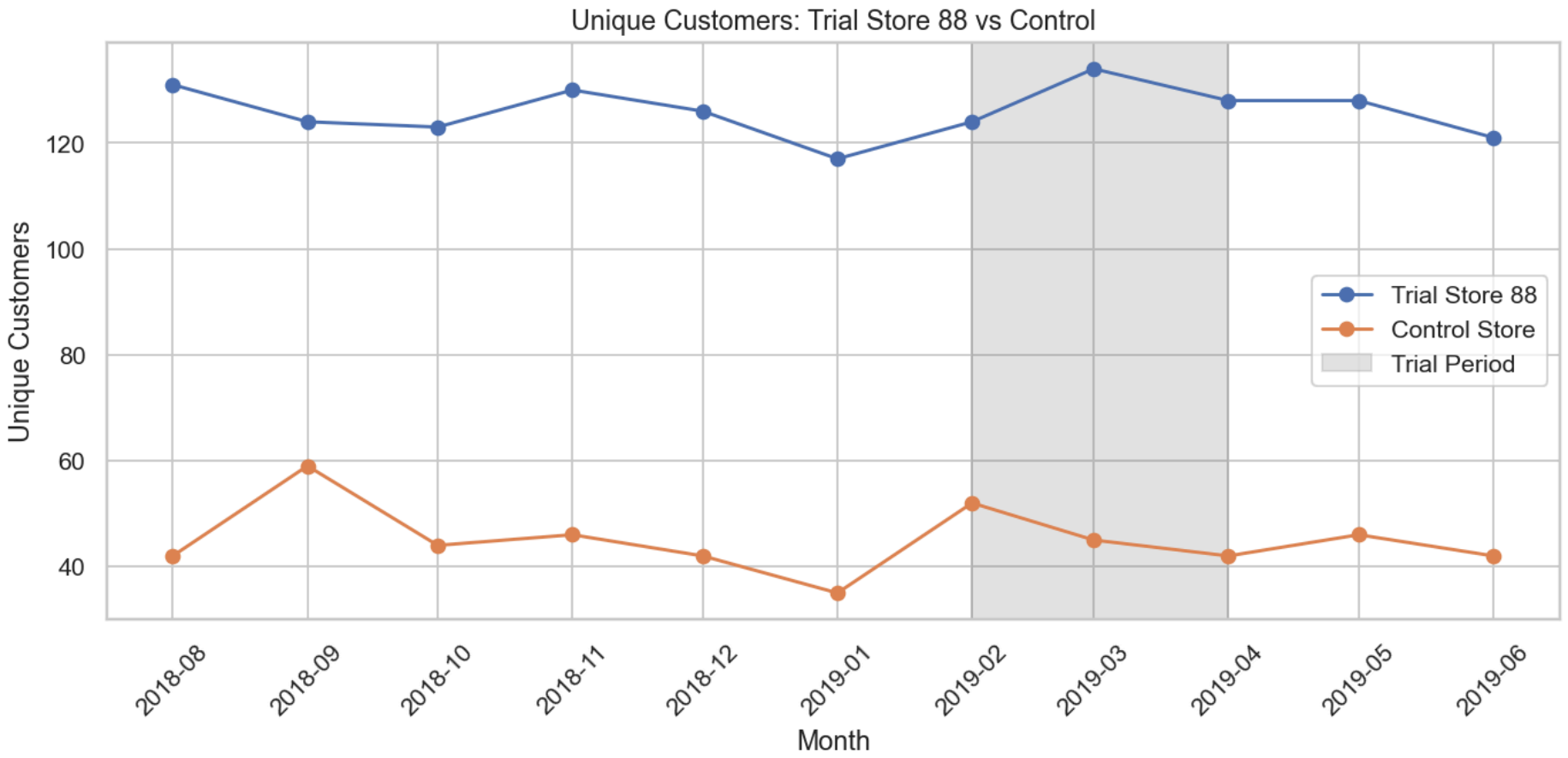

Total Sales: Trial Store 86 vs Control
Total Sales
1000
800
600
400
200
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
2019-05
2019-06
Month
Trial Store 86
Control Store
Trial Period

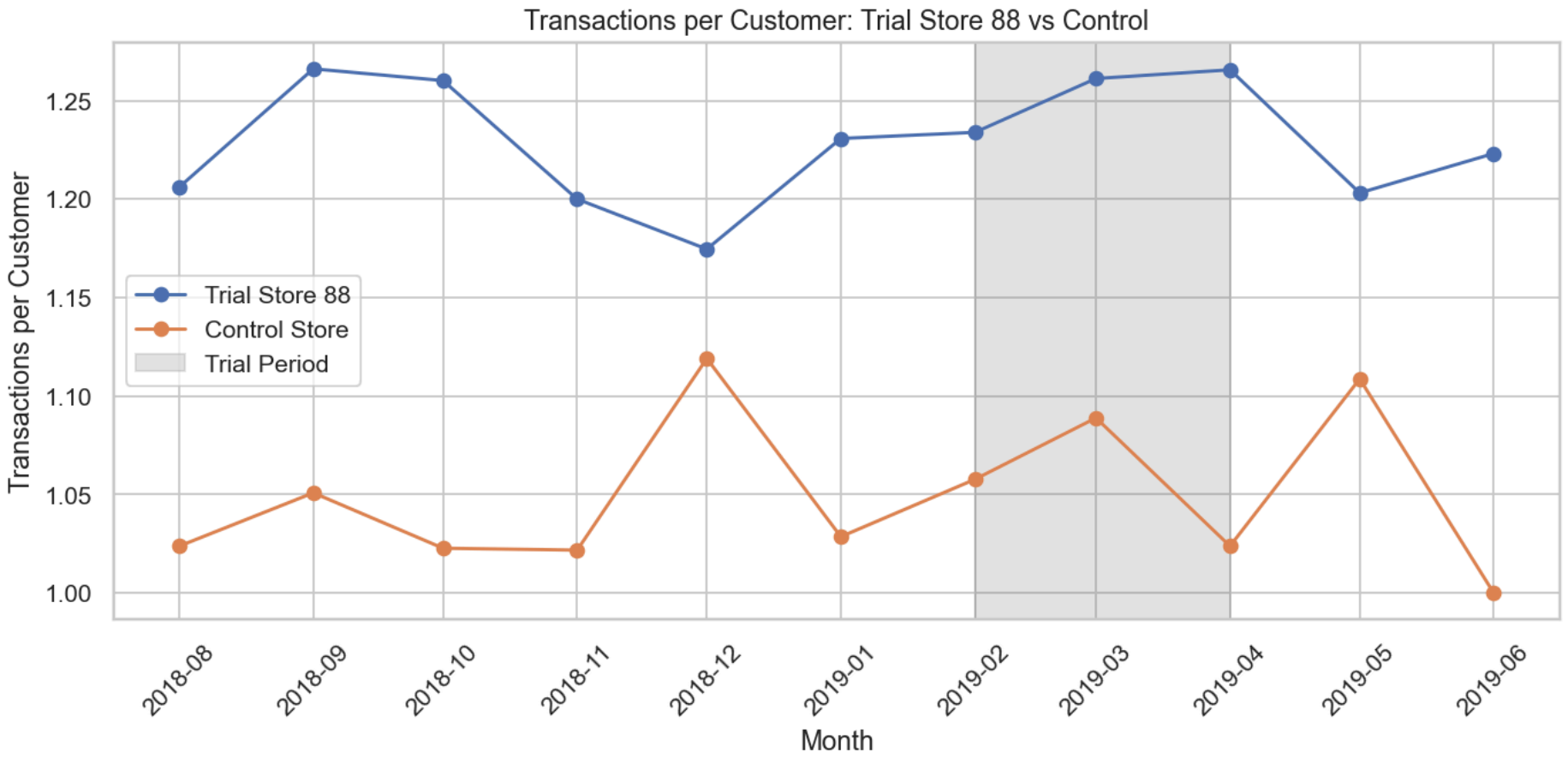

Unique Customers: Trial Store 86 vs Control
Unique Customers
110
100
90
80
70
60
50
40
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
2019-05
2019-06
Month
Trial Store 86
Control Store
Trial Period



Creating charts for Trial Store 88


Total Sales: Trial Store 88 vs Control
Total Sales
1400
1200
1000
800
600
400
200
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
2019-05
2019-06
Month
Trial Store 88
Control Store
Trial Period


Unique Customers: Trial Store 88 vs Control
Unique Customers
120
100
80
60
40
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
2019-05
2019-06
Month
Trial Store 88
Control Store
Trial Period


Transactions per Customer: Trial Store 88 vs Control
Transactions per Customer
1.25
1.20
1.15
1.10
1.05
1.00
Trial Store 88
Control Store
Trial Period
2018-08
2018-09
2018-10
2018-11
2018-12
2019-01
2019-02
2019-03
2019-04
2019-05
2019-06
Month

In [ ]: