```python
import pandas as pd
import numpy as np
import seaborn as sns

# Install dependencies as needed:
# pip install kagglehub[pandas-datasets]
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Set the path to the file you'd like to load
file_path = "QVI_data (1).csv"

# Load the latest version
df = kagglehub.dataset_load(
  KaggleDatasetAdapter.PANDAS,
  "preethis14/qvi-dataset",
  file_path,
  # Provide any additional arguments like
  # sql_query or pandas_kwargs. See the
  # documenation for more information:
  #
https://github.com/Kaggle/kagglehub/blob/main/README.md#kaggledataseta
dapterpandas
)

print("First 5 records:", df.head())
```

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

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

        BRAND              LIFESTAGE PREMIUM_CUSTOMER
0     NATURAL  YOUNG SINGLES/COUPLES          Premium
```

```
1         RRD  YOUNG SINGLES/COUPLES       Mainstream
2     GRNWVES          YOUNG FAMILIES           Budget
3     NATURAL          YOUNG FAMILIES           Budget
4  WOOLWORTHS  OLDER SINGLES/COUPLES       Mainstream
```

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

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

For this, let's first create the metrics of interest and filter to stores that are present throughout the pre-trial period.

```
df['DATE'] = pd.to_datetime(df['DATE'])
df['YEARMONTH'] = df['DATE'].dt.strftime('%Y%m')

pre_trial_months = pd.date_range('2018-07-01', '2019-01-31',
freq='MS').strftime('%Y%m')

store_month_counts = (
    df[df['YEARMONTH'].isin(pre_trial_months)]
    .groupby('STORE_NBR')['YEARMONTH']
    .nunique()
    .reset_index(name='MONTH_COUNT')
)

valid_stores = store_month_counts[store_month_counts['MONTH_COUNT'] ==
len(pre_trial_months)]['STORE_NBR']

filtered_df = df[df['STORE_NBR'].isin(valid_stores)]
```

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

```
measure_over_time = (
    df.groupby(['STORE_NBR', 'YEARMONTH'])
    .agg(
        totSales=('TOT_SALES', 'sum'),
        nCustomers=('LYLTY_CARD_NBR', 'nunique'),
        nTxn=('TXN_ID', 'nunique'),
        totalChips=('PROD_QTY', 'sum'),
        totalUnits=('PROD_QTY', 'sum'),
        totalSales=('TOT_SALES', 'sum')
    )
    .reset_index()
)

measure_over_time['nTxnPerCust'] = measure_over_time['nTxn'] /
```

```
measure_over_time['nCustomers']
measure_over_time['nChipsPerTxn'] = measure_over_time['totalChips'] /
measure_over_time['nTxn']
measure_over_time['avgPricePerUnit'] = measure_over_time['totalSales']
/ measure_over_time['totalUnits']

measure_over_time = measure_over_time[['STORE_NBR', 'YEARMONTH',
'totSales', 'nCustomers', 'nTxnPerCust', 'nChipsPerTxn',
'avgPricePerUnit']]

measure_over_time = measure_over_time.sort_values(by=['STORE_NBR',
'YEARMONTH'])

## Filter to pre-trial period

pre_trial = measure_over_time[(measure_over_time['YEARMONTH'] <
201902)]
full_obs_stores = measure_over_time.groupby('STORE_NBR')
['YEARMONTH'].nunique()
full_obs_stores = full_obs_stores[full_obs_stores == 12].index
pre_trial = pre_trial[pre_trial['STORE_NBR'].isin(full_obs_stores)]

def calculate_correlation(input_df, metric_col, store_comparison):
    trial_store_data = input_df[input_df['STORE_NBR'] ==
store_comparison][['YEARMONTH', metric_col]]
    trial_store_data = trial_store_data.rename(columns={metric_col:
'TRIAL_STORE_METRIC'})

    correlation_list = []

    for store in input_df['STORE_NBR'].unique():
        if store == store_comparison:
            continue

        control_store_data = input_df[input_df['STORE_NBR'] == store]
[['YEARMONTH', metric_col]]
        merged_data = trial_store_data.merge(control_store_data,
on='YEARMONTH')
        corr =
merged_data['TRIAL_STORE_METRIC'].corr(merged_data[metric_col])

        correlation_list.append({
            'Store1': store_comparison,
            'Store2': store,
            'corr_measure': corr
        })

    return pd.DataFrame(correlation_list)

## Magnitude Distance Function
```

```
def calculate_magnitude_distance(input_df, metric_col,
store_comparison):
    trial_data = input_df[input_df['STORE_NBR'] == store_comparison]
[['YEARMONTH', metric_col]]
    results = []
    for store in input_df['STORE_NBR'].unique():
        if store != store_comparison:
            control_data = input_df[input_df['STORE_NBR'] == store]
[['YEARMONTH', metric_col]]
            merged = pd.merge(trial_data, control_data,
on='YEARMONTH', suffixes=('_trial', '_control'))
            merged['measure'] = abs(merged[metric_col + '_trial'] -
merged[metric_col + '_control'])
            merged['Store1'] = store_comparison
            merged['Store2'] = store
            results.append(merged[['Store1', 'Store2', 'YEARMONTH',
'measure']])
    result_df = pd.concat(results)
    min_max = result_df.groupby(['Store1',
'YEARMONTH']).agg(minDist=('measure', 'min'), maxDist=('measure',
'max')).reset_index()
    result_df = pd.merge(result_df, min_max, on=['Store1',
'YEARMONTH'])
    result_df['magnitudeMeasure'] = 1 - ((result_df['measure'] -
result_df['minDist']) / (result_df['maxDist'] - result_df['minDist']))
    final_df = result_df.groupby(['Store1', 'Store2'])
['magnitudeMeasure'].mean().reset_index(name='mag_measure')
    return final_df


## Find control store (example for store 77)

trial_store = 77
corr_sales = calculate_correlation(pre_trial, 'totSales', trial_store)
corr_cust = calculate_correlation(pre_trial, 'nCustomers',
trial_store)

mag_sales = calculate_magnitude_distance(pre_trial, 'totSales',
trial_store)
mag_cust = calculate_magnitude_distance(pre_trial, 'nCustomers',
trial_store)

# Combine correlations and magnitude scores
score_sales = pd.merge(corr_sales, mag_sales, on=['Store1', 'Store2'])
score_sales['scoreNSales'] = 0.5 * score_sales['corr_measure'] + 0.5 *
score_sales['mag_measure']

score_cust = pd.merge(corr_cust, mag_cust, on=['Store1', 'Store2'])
score_cust['scoreNCust'] = 0.5 * score_cust['corr_measure'] + 0.5 *
score_cust['mag_measure']
```

```
final_score = pd.merge(score_sales[['Store1', 'Store2',
'scoreNSales']],
                       score_cust[['Store1', 'Store2', 'scoreNCust']],
                       on=['Store1', 'Store2'])
final_score['finalControlScore'] = 0.5 * final_score['scoreNSales'] +
0.5 * final_score['scoreNCust']

control_store = final_score.sort_values(by='finalControlScore',
ascending=False).iloc[1]['Store2']
print(f"Control store for trial store {trial_store}: {control_store}")
```

```
Control store for trial store 77: 41.0
```

```
## Visual Check: Total Sales Pre-trial

plot_data = measure_over_time.copy()
plot_data['Store_type'] = plot_data['STORE_NBR'].apply(lambda x:
'Trial' if x == trial_store else 'Control' if x == control_store else
'Other')
plot_data = plot_data.groupby(['YEARMONTH', 'Store_type'])
['totSales'].mean().reset_index()
plot_data['TransactionMonth'] =
pd.to_datetime(plot_data['YEARMONTH'].astype(str), format='%Y%m')

# Clean infinite values and drop NaNs to avoid warnings
plot_data.replace([np.inf, -np.inf], np.nan, inplace=True)
plot_data.dropna(inplace=True)

pre_trial_plot = plot_data[plot_data['YEARMONTH'] < 201903]

sns.lineplot(data=pre_trial_plot, x='TransactionMonth', y='totSales',
hue='Store_type')
plt.title('Total Sales by Month (Pre-trial)')
plt.xlabel('Month of Operation')
plt.ylabel('Total Sales')
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1075:
```

```
FutureWarning: When grouping with a length-1 list-like, you will need
to pass a length-1 tuple to get_group in a future version of pandas.
Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1075:
FutureWarning: When grouping with a length-1 list-like, you will need
to pass a length-1 tuple to get_group in a future version of pandas.
Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1075:
FutureWarning: When grouping with a length-1 list-like, you will need
to pass a length-1 tuple to get_group in a future version of pandas.
Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```