

```
import pandas as pd

# Upload file from your system
from google.colab import files
uploaded = files.upload()

# Suppose your file is named "data.csv"
df = pd.read_csv("QVI_data.csv")

# Quick check
print(df.head())
print(df.info())
```

Choose Files

QVI\_data.csv

• **QVI\_data.csv**(text/csv) - 29019945 bytes, last modified: 8/26/2025 - 100% done

Saving QVI\_data.csv to QVI\_data (1).csv

	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

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 264834 entries, 0 to 264833

Data columns (total 12 columns):

#	Column	Non-Null Count	Dtype
0	LYLTY_CARD_NBR	264834 non-null	int64
1	DATE	264834 non-null	object
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

dtypes: float64(1), int64(6), object(5)

memory usage: 24.2+ MB

None

```
# Check total NULLs per column
print(df.isnull().sum())

# Check percentage of NULLs per column
print((df.isnull().mean()*100).round(2))

# Show rows with any NULL values
null_rows = df[df.isnull().any(axis=1)]
print(null_rows.head())
```

LYLTY\_CARD\_NBR

0

DATE

0

STORE\_NBR

0

TXN\_ID

0

PROD\_NBR

0

PROD\_NAME

0

PROD\_QTY

0

TOT\_SALES

0

PACK\_SIZE

0

BRAND

0

LIFESTAGE

0

PREMIUM\_CUSTOMER

0

```

dtype: int64
LYLTY_CARD_NBR      0.0
DATE                0.0
STORE_NBR           0.0
TXN_ID              0.0
PROD_NBR            0.0
PROD_NAME           0.0
PROD_QTY            0.0
TOT_SALES           0.0
PACK_SIZE           0.0
BRAND               0.0
LIFESTAGE           0.0
PREMIUM_CUSTOMER    0.0
dtype: float64
Empty DataFrame
Columns: [LYLTY_CARD_NBR, DATE, STORE_NBR, TXN_ID, PROD_NBR, PROD_NAME, PROD_QTY, TOT_SALES, PACK_SIZE, BRAND, LIFESTAGE, PREMIUM_CUSTOM
Index: []

```

```

# Check total duplicate rows
print("Total Duplicates:", df.duplicated().sum())

```

```

# Show duplicate rows
duplicates = df[df.duplicated()]
print(duplicates.head())

```

```

# Remove duplicates (if needed)
df_no_dup = df.drop_duplicates()

```

```

print("Shape before:", df.shape)
print("Shape after :", df_no_dup.shape)

```

```

↗ Total Duplicates: 0
Empty DataFrame
Columns: [LYLTY_CARD_NBR, DATE, STORE_NBR, TXN_ID, PROD_NBR, PROD_NAME, PROD_QTY, TOT_SALES, PACK_SIZE, BRAND, LIFESTAGE, PREMIUM_CUSTOM
Index: []
Shape before: (264833, 12)
Shape after : (264833, 12)

```

```

# Delete duplicate rows (keep first occurrence)
df.drop_duplicates(inplace=True)

```

```

# Reset index after deletion
df.reset_index(drop=True, inplace=True)

```

```

print("After removing duplicates:", df.shape)

```

```

↗ After removing duplicates: (264833, 12)

```

```

import numpy as np

```

```

# Example: checking outliers in SALES_VALUE
Q1 = df['TOT_SALES'].quantile(0.25)
Q3 = df['TOT_SALES'].quantile(0.75)
IQR = Q3 - Q1

```

```

# Define bounds
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

```

```

# Find outliers
outliers = df[(df['TOT_SALES'] < lower) | (df['TOT_SALES'] > upper)]
print("Total outliers:", outliers.shape[0])
print(outliers.head())

```

```

↗ Total outliers: 576
  LYLTY_CARD_NBR  DATE  STORE_NBR  TXN_ID  PROD_NBR  \
952           2373  2018-08-17         2      974      69
4122           4347  2019-05-16         4     4220      14
4630           5063  2018-08-20         5     4742      30
4640           5065  2019-05-15         5     4756      15
4719           5080  2019-05-17         5     4842      79

  PROD_NAME  PROD_QTY  TOT_SALES  PACK_SIZE  \
952  Smiths Chip Thinly S/Cream&Onion 175g         5        15.0        175
4122  Smiths Crnkle Chip Orgnl Big Bag 380g         3        17.7        380

```

4630	Doritos Corn Chips	Cheese Supreme	170g	5	22.0	170
4640		Twisties Cheese	270g	4	18.4	270
4719	Smiths Chip Thinly	CutSalt/Vinegr	175g	5	15.0	175

	BRAND		LIFESTAGE	PREMIUM_CUSTOMER
952	SMITHS	MIDAGE	SINGLES/COUPLES	Budget
4122	SMITHS		YOUNG FAMILIES	Budget
4630	DORITOS		YOUNG FAMILIES	Premium
4640	TWISTIES		OLDER FAMILIES	Premium
4719	SMITHS		OLDER FAMILIES	Mainstream

```
# IQR method for SALES_VALUE
```

```
Q1 = df['TOT_SALES'].quantile(0.25)
```

```
Q3 = df['TOT_SALES'].quantile(0.75)
```

```
IQR = Q3 - Q1
```

```
lower = Q1 - 1.5 * IQR
```

```
upper = Q3 + 1.5 * IQR
```

```
# Outlier values
```

```
outliers = df[(df['TOT_SALES'] < lower) | (df['TOT_SALES'] > upper)]
```

```
# Distinct values in comma-separated form
```

```
distinct_outliers = ", ".join(map(str, sorted(outliers['TOT_SALES'].unique())))
```

```
print(distinct_outliers)
```

```
15.0, 15.2, 15.3, 15.5, 15.6, 16.2, 16.25, 16.5, 16.8, 17.1, 17.2, 17.6, 17.7, 18.0, 18.4, 18.5, 19.0, 19.5, 20.4, 21.0, 21.5, 21.6, 22.
```

```
# Calculate IQR
```

```
Q1 = df['TOT_SALES'].quantile(0.25)
```

```
Q3 = df['TOT_SALES'].quantile(0.75)
```

```
IQR = Q3 - Q1
```

```
lower = Q1 - 1.5 * IQR
```

```
upper = Q3 + 1.5 * IQR
```

```
# Remove outliers
```

```
df_no_outliers = df[(df['TOT_SALES'] >= lower) & (df['TOT_SALES'] <= upper)].copy()
```

```
print("Before:", df.shape)
```

```
print("After removing outliers:", df_no_outliers.shape)
```

```
Before: (264833, 12)
After removing outliers: (264257, 12)
```

```
import pandas as pd
```

```
# Ensure DATE is datetime
```

```
df['DATE'] = pd.to_datetime(df['DATE'])
```

```
# Extract month-year
```

```
df['Month'] = df['DATE'].dt.to_period('M')
```

```
# Filter only required stores
```

```
stores_df = df[df['STORE_NBR'].isin([77, 86, 88])]
```

```
# Group by store and month for sales revenue
```

```
monthly_revenue = (stores_df
                    .groupby(['STORE_NBR', 'Month'])['']
                    .sum()
                    .reset_index(name='Monthly_Revenue'))
```

```
print(monthly_revenue)
```

	STORE_NBR	Month	Monthly_Revenue
0	77	2018-07	296.80
1	77	2018-08	255.50
2	77	2018-09	225.20
3	77	2018-10	204.50
4	77	2018-11	245.30
5	77	2018-12	267.30
6	77	2019-01	204.40
7	77	2019-02	235.00
8	77	2019-03	278.50
9	77	2019-04	263.50
10	77	2019-05	299.30

11	77	2019-06	264.70
12	86	2018-07	892.20
13	86	2018-08	764.05
14	86	2018-09	914.60
15	86	2018-10	948.40
16	86	2018-11	918.00
17	86	2018-12	841.20
18	86	2019-01	841.40
19	86	2019-02	913.20
20	86	2019-03	1026.80
21	86	2019-04	848.20
22	86	2019-05	889.30
23	86	2019-06	838.00
24	88	2018-07	1310.00
25	88	2018-08	1323.80
26	88	2018-09	1423.00
27	88	2018-10	1352.40
28	88	2018-11	1382.80
29	88	2018-12	1325.20
30	88	2019-01	1266.40
31	88	2019-02	1370.20
32	88	2019-03	1477.20
33	88	2019-04	1439.40
34	88	2019-05	1308.25
35	88	2019-06	1354.60

""" Monthly unique 'PREMIUM\_CUSTOMER' count per store  
NEED TO RE-CHECK """

```
monthly_customers = (stores_df
    .groupby(['STORE_NBR', 'Month'])['PREMIUM_CUSTOMER']
    .nunique()
    .reset_index(name='Total_Customers'))
```

```
print(monthly_customers)
```

	STORE_NBR	Month	Total_Customers
0	77	2018-07	3
1	77	2018-08	3
2	77	2018-09	3
3	77	2018-10	3
4	77	2018-11	3
5	77	2018-12	3
6	77	2019-01	3
7	77	2019-02	3
8	77	2019-03	3
9	77	2019-04	3
10	77	2019-05	3
11	77	2019-06	3
12	86	2018-07	3
13	86	2018-08	3
14	86	2018-09	3
15	86	2018-10	3
16	86	2018-11	3
17	86	2018-12	3
18	86	2019-01	3
19	86	2019-02	3
20	86	2019-03	3
21	86	2019-04	3
22	86	2019-05	3
23	86	2019-06	3
24	88	2018-07	3
25	88	2018-08	3
26	88	2018-09	3
27	88	2018-10	3
28	88	2018-11	3
29	88	2018-12	3
30	88	2019-01	3
31	88	2019-02	3
32	88	2019-03	3
33	88	2019-04	3
34	88	2019-05	3
35	88	2019-06	3

# Average no. of transactions per customer #

```
monthly_txn = (stores_df
    .groupby(['STORE_NBR', 'Month'])['TXN_ID']
    .nunique())
```

```

.reset_index(name='Total_Transactions'))


# Monthly unique customers
monthly_customers = (stores_df
    .groupby(['STORE_NBR', 'Month'])['PREMIUM_CUSTOMER']
    .nunique()
    .reset_index(name='Total_Customers'))

# Merge both
monthly_stats = monthly_txn.merge(monthly_customers, on=['STORE_NBR', 'Month'])

# Calculate avg transactions per customer
monthly_stats['Avg_Txn_per_Customer'] = (monthly_stats['Total_Transactions'] /
    monthly_stats['Total_Customers']).round(2)

print(monthly_stats)

```



	STORE_NBR	Month	Total_Transactions	Total_Customers
12	88	2018-07	140	3
13	86	2018-08	110	3
14	86	2018-09	128	3
15	86	2018-10	138	3
16	86	2018-11	125	3
17	86	2018-12	120	3
18	86	2019-01	129	3
19	86	2019-02	138	3
20	86	2019-03	140	3
21	86	2019-04	126	3
22	86	2019-05	128	3
23	86	2019-06	118	3
24	88	2018-07	153	3
25	88	2018-08	158	3
26	88	2018-09	157	3
27	88	2018-10	155	3
28	88	2018-11	156	3
29	88	2018-12	148	3
30	88	2019-01	144	3
31	88	2019-02	153	3
32	88	2019-03	169	3
33	88	2019-04	162	3
34	88	2019-05	154	3
35	88	2019-06	148	3

	Avg_Txn_per_Customer
0	18.33
1	16.00
2	14.67
3	12.67
4	14.67
5	16.00
6	13.00
7	15.00
8	18.33
9	16.00
10	18.67
11	14.00
12	42.00
13	36.67
14	42.67
15	46.00
16	41.67
17	40.00
18	43.00
19	46.00
20	46.67
21	42.00
22	42.67
23	39.33
24	51.00
25	52.67
26	52.33
27	51.67
28	52.00
29	49.33
30	48.00
31	51.00
32	56.33

```

# Define pre-trial period
pre_trial_period = (df['Month'] >= '2018-07') & (df['Month'] <= '2018-11')

trial_store = 77
trial_data = (df[pre_trial_period & (df['STORE_NBR']==trial_store)])

```

```

        .groupby('Month')['TOT_SALES'].sum())

scores = {}
for store in df['STORE_NBR'].unique():
    if store == trial_store:
        continue
    control_data = (df[pre_trial_period & (df['STORE_NBR']==store)]
                    .groupby('Month')['TOT_SALES'].sum())
    if len(control_data) == len(trial_data):
        scores[store] = trial_data.corr(control_data)

best_control = max(scores, key=scores.get)
print("Best control store for", trial_store, "is", best_control)

➡ Best control store for 77 is 115

# Example trial store
trial = np.array([100, 120, 130, 140])

# Example control stores
controls = {
    101: np.array([98, 119, 128, 141]),
    102: np.array([90, 110, 115, 120]),
    103: np.array([130, 140, 150, 160])
}

df_scores = calculate_similarity(trial, controls)
print(df_scores)

➡
   store_id  pearson_corr  magnitude_sim  final_score
0         101         0.998397         1.00000         0.999198
1         102         0.983338         0.40088         0.692109
2         103         0.982708         0.00000         0.491354

# Define the best control stores based on the previous analysis
best_controls = {
    77: best_77,
    86: best_86,
    88: best_88
}

# Define trial period (edit if different)
trial_start = '2019-02-01'
trial_end = '2019-04-30'

for trial, control in best_controls.items():
    if control is None:
        print(f"\n⚠ No valid control store found for Trial Store {trial}. Skipping comparison.")
        continue

    results = compare_trial_vs_control(
        df,
        trial_store=trial,
        control_store=control,
        trial_start=trial_start,
        trial_end=trial_end
    )

    print(f"\n📊 Trial Store {trial} vs Control Store {control} during Trial Period ({trial_start} to {trial_end})")
    display(results)  # shows the table

```

Trial Store 77 vs Control Store 115 during Trial Period (2019-02-01 to 2019-04-30)

	Month	Trial_Sales	Control_Sales	
0	2019-02	235.0	295.4	
1	2019-03	278.5	282.8	
2	2019-04	263.5	389.0	

Trial Store 86 vs Control Store 49 during Trial Period (2019-02-01 to 2019-04-30)

	Month	Trial_Sales	Control_Sales	
0	2019-02	913.2	987.3	
1	2019-03	1026.8	1261.8	
2	2019-04	848.2	1024.0	

Trial Store 88 vs Control Store 125 during Trial Period (2019-02-01 to 2019-04-30)

	Month	Trial_Sales	Control_Sales	
0	2019-02	1370.2	915.8	
1	2019-03	1477.2	1235.4	
2	2019-04	1439.4	1108.2	

Next steps:

Generate code with results

View recommended plots

New interactive sheet

Generate code with results

View recommended plots

Start coding or [generate](#) with AI.

Start coding or [generate](#) with AI.