task2

August 8, 2023

0.1 Retail Strategy and Analytics - Task 2

We will start by importing the necessary libraries and reading the data into a dataframe.

[582]:
```
# Import required libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from scipy.stats import ttest_ind
from scipy.stats import t
```

[583]:
```
data = pd.read_csv('QVI_data.csv')
```

We will now observe if the data has been successfully imported.

[584]:
```
data.head()
```

[584]:
```
         DATE  STORE_NBR  LYLTY_CARD_NBR  TXN_ID  PROD_NBR  \
0  2018-10-17          1            1000       1         5
1  2019-05-14          1            1307     348        66
2  2019-05-20          1            1343     383        61
3  2018-08-17          2            2373     974        69
4  2018-08-18          2            2426    1038       108

                                  PROD_NAME  PROD_QTY  TOT_SALES  PACK_SIZE  \
0    Natural Chip        Compny SeaSalt175g         2        6.0      175.0
1                  CCs Nacho Cheese    175g         3        6.3      175.0
2    Smiths Crinkle Cut  Chips Chicken 170g         2        2.9      170.0
3    Smiths Chip Thinly  S/Cream&Onion 175g         5       15.0      175.0
4  Kettle Tortilla ChpsHny&Jlpno Chili 150g         3       13.8      150.0

     BRAND              LIFESTAGE PREMIUM_CUSTOMER
0  Natural  YOUNG SINGLES/COUPLES          Premium
1      CCs  MIDAGE SINGLES/COUPLES          Budget
2   Smiths  MIDAGE SINGLES/COUPLES          Budget
3   Smiths  MIDAGE SINGLES/COUPLES          Budget
4   Kettle  MIDAGE SINGLES/COUPLES          Budget
```

Wonderful! Now lets observe the data characteristics.

[585]:
```
data.describe()
```

[585]:
```
           STORE_NBR  LYLTY_CARD_NBR        TXN_ID       PROD_NBR  \
count  246742.000000    2.467420e+05  2.467420e+05  246742.000000
mean      135.051098    1.355310e+05  1.351311e+05      56.351789
std        76.787096    8.071528e+04  7.814772e+04      33.695428
min         1.000000    1.000000e+03  1.000000e+00       1.000000
25%        70.000000    7.001500e+04  6.756925e+04      26.000000
50%       130.000000    1.303670e+05  1.351830e+05      53.000000
75%       203.000000    2.030840e+05  2.026538e+05      87.000000
max       272.000000    2.373711e+06  2.415841e+06     114.000000

            PROD_QTY      TOT_SALES      PACK_SIZE
count  246742.000000  246742.000000  246742.000000
mean        1.908062       7.321322     175.585178
std         0.659831       3.077828      59.434727
min         1.000000       1.700000      70.000000
25%         2.000000       5.800000     150.000000
50%         2.000000       7.400000     170.000000
75%         2.000000       8.800000     175.000000
max       200.000000     650.000000     380.000000
```

We will set the style of our plots down below.

[586]:
```
sns.set_style("ticks")
plt.rc("figure", figsize=(8, 4))
plt.rc("font", size=14)
plt.rc("axes", labelsize=14)
sns.set_palette("dark")
```

[587]:
```
data.dtypes
```

[587]:
```
DATE                 object
STORE_NBR             int64
LYLTY_CARD_NBR        int64
TXN_ID                int64
PROD_NBR              int64
PROD_NAME            object
PROD_QTY              int64
TOT_SALES           float64
PACK_SIZE           float64
BRAND                object
LIFESTAGE            object
PREMIUM_CUSTOMER     object
dtype: object
```

Conversion of 'DATE' data type to datetime and then extracting a new column ' MONTH_ID' from it.

[588]:
```
# Convert DATE column to datetime format
data['DATE'] = pd.to_datetime(data['DATE'], format='%Y-%m-%d')
print(data['DATE'].head())
```

```
0   2018-10-17
1   2019-05-14
2   2019-05-20
3   2018-08-17
4   2018-08-18
Name: DATE, dtype: datetime64[ns]
```

[589]:
```
# Convert DATE column to datetime format
data['MONTH_ID'] = data['DATE'].dt.strftime('%Y-%m')
print(data['MONTH_ID'].head())
```

```
0    2018-10
1    2019-05
2    2019-05
3    2018-08
4    2018-08
Name: MONTH_ID, dtype: object
```

[590]:
```
data.dtypes
```

[590]:
```
DATE                datetime64[ns]
STORE_NBR                    int64
LYLTY_CARD_NBR               int64
TXN_ID                       int64
PROD_NBR                     int64
PROD_NAME                   object
PROD_QTY                     int64
TOT_SALES                  float64
PACK_SIZE                  float64
BRAND                       object
LIFESTAGE                   object
PREMIUM_CUSTOMER            object
MONTH_ID                    object
dtype: object
```

Lets sort the data by date of transaction.

[591]:
```
data = data.sort_values(by=['DATE'])
data.head(50)
```

[591]:

	DATE	STORE_NBR	LYLTY_CARD_NBR	TXN_ID	PROD_NBR \
126979	2018-07-01	9	9341	8808	45
146787	2018-07-01	86	86016	84237	48
117744	2018-07-01	129	129046	132474	82
113095	2018-07-01	58	58072	53145	99
68505	2018-07-01	97	97164	97311	92
105085	2018-07-01	199	199302	198907	23
146562	2018-07-01	81	81292	81039	109
65996	2018-07-01	32	32062	28370	31
217613	2018-07-01	147	147113	146386	44
8439	2018-07-01	84	84317	84019	104
24865	2018-07-01	134	134264	138078	23
42220	2018-07-01	148	148014	147248	108
179431	2018-07-01	104	104222	105082	72
127345	2018-07-01	19	19009	15816	16
4461	2018-07-01	257	257076	256197	102
245076	2018-07-01	195	195403	195303	75
121447	2018-07-01	194	194091	193649	70
4452	2018-07-01	256	256377	255572	73
146307	2018-07-01	80	80003	78303	99
13261	2018-07-01	207	207165	205566	16
46848	2018-07-01	209	209073	207890	96
127427	2018-07-01	19	19316	16841	32
167660	2018-07-01	165	165283	166726	77
85756	2018-07-01	164	164185	164927	104
183710	2018-07-01	168	168215	170691	60
33601	2018-07-01	45	45071	40807	43
179245	2018-07-01	102	102200	102736	60
15003	2018-07-01	247	247171	249667	103
20279	2018-07-01	155	155043	155282	21
193065	2018-07-01	68	68141	65524	51
147678	2018-07-01	97	97014	96348	92
188245	2018-07-01	236	236023	238660	100
121642	2018-07-01	197	197053	196970	75
235573	2018-07-01	197	197341	197303	89
207700	2018-07-01	131	131000	135315	104
194757	2018-07-01	103	103390	103548	92
10652	2018-07-01	137	137206	140143	100
147427	2018-07-01	94	94170	93531	23
24736	2018-07-01	122	122009	124703	52
105518	2018-07-01	208	208072	206511	56
121544	2018-07-01	194	194349	194710	7
188051	2018-07-01	232	232024	235328	18
4658	2018-07-01	269	269175	266094	28
119730	2018-07-01	164	164250	165331	31
93757	2018-07-01	13	13133	11927	53
166594	2018-07-01	137	137009	138924	81

135043	2018-07-01	172	172045	173027	85
243191	2018-07-01	101	101237	101395	54
130992	2018-07-01	86	86203	85392	19
166806	2018-07-01	143	143365	143865	63

	PROD_NAME	PROD_QTY	TOT_SALES	\
126979	Smiths Thinly Cut Roast Chicken 175g	2	6.0	
146787	Red Rock Deli Sp Salt & Truffle 150G	2	5.4	
117744	Smith Crinkle Cut Mac N Cheese 150g	2	5.2	
113095	Pringles Sthrn FriedChicken 134g	2	7.4	
68505	WW Crinkle Cut Chicken 175g	2	3.4	
105085	Cheezels Cheese 330g	2	11.4	
146562	Pringles Barbeque 134g	2	7.4	
65996	Infzns Crn Crnchers Tangy Gcamole 110g	2	7.6	
217613	Thins Chips Light& Tangy 175g	2	6.6	
8439	Infuzions Thai SweetChili PotatoMix 110g	2	7.6	
24865	Cheezels Cheese 330g	2	11.4	
42220	Kettle Tortilla ChpsHny&Jlpno Chili 150g	1	4.6	
179431	WW Crinkle Cut Original 175g	2	3.4	
127345	Smiths Crinkle Chips Salt & Vinegar 330g	2	11.4	
4461	Kettle Mozzarella Basil & Pesto 175g	2	10.8	
245076	Cobs Popd Sea Salt Chips 110g	1	3.8	
121447	Tyrrells Crisps Lightly Salted 165g	2	8.4	
4452	Smiths Crinkle Cut Salt & Vinegar 170g	2	5.8	
146307	Pringles Sthrn FriedChicken 134g	2	7.4	
13261	Smiths Crinkle Chips Salt & Vinegar 330g	2	11.4	
46848	WW Original Stacked Chips 160g	2	3.8	
127427	Kettle Sea Salt And Vinegar 175g	2	10.8	
167660	Doritos Corn Chips Nacho Cheese 170g	2	8.8	
85756	Infuzions Thai SweetChili PotatoMix 110g	2	7.6	
183710	Kettle Tortilla ChpsFeta&Garlic 150g	2	9.2	
33601	Smith Crinkle Cut Bolognese 150g	2	5.2	
179245	Kettle Tortilla ChpsFeta&Garlic 150g	2	9.2	
15003	RRD Steak & Chimuchurri 150g	2	5.4	
20279	WW Sour Cream &OnionStacked Chips 160g	2	3.8	
193065	Doritos Mexicana 170g	2	8.8	
147678	WW Crinkle Cut Chicken 175g	2	3.4	
188245	Smiths Crinkle Cut Chips Chs&Onion170g	2	5.8	
121642	Cobs Popd Sea Salt Chips 110g	2	7.6	
235573	Kettle Sweet Chilli And Sour Cream 175g	2	10.8	
207700	Infuzions Thai SweetChili PotatoMix 110g	1	3.8	
194757	WW Crinkle Cut Chicken 175g	2	3.4	
10652	Smiths Crinkle Cut Chips Chs&Onion170g	2	5.8	
147427	Cheezels Cheese 330g	2	11.4	
24736	Grain Waves Sour Cream&Chives 210G	2	7.2	
105518	Cheezels Cheese Box 125g	2	4.2	
121544	Smiths Crinkle Original 330g	2	11.4	

188051	Cheetos Chs & Bacon Balls 190g	2	6.6
4658	Thins Potato Chips Hot & Spicy 175g	2	6.6
119730	Infzns Crn Crnchers Tangy Gcamole 110g	2	7.6
93757	RRD Sweet Chilli & Sour Cream 165g	2	6.0
166594	Pringles Original Crisps 134g	2	7.4
135043	RRD Honey Soy Chicken 165g	2	6.0
243191	CCs Original 175g	2	4.2
130992	Smiths Crinkle Cut Snag&Sauce 150g	2	5.2
166806	Kettle 135g Swt Pot Sea Salt	2	8.4

	PACK_SIZE	BRAND	LIFESTAGE	PREMIUM_CUSTOMER	MONTH_ID
126979	175.0	Smiths	RETIREES	Budget	2018-07
146787	150.0	Red	RETIREES	Mainstream	2018-07
117744	150.0	Smith	OLDER SINGLES/COUPLES	Premium	2018-07
113095	134.0	Pringles	OLDER SINGLES/COUPLES	Premium	2018-07
68505	175.0	WW	OLDER FAMILIES	Premium	2018-07
105085	330.0	Cheezels	OLDER SINGLES/COUPLES	Mainstream	2018-07
146562	134.0	Pringles	RETIREES	Mainstream	2018-07
65996	110.0	Infzns	OLDER FAMILIES	Premium	2018-07
217613	175.0	Thins	YOUNG SINGLES/COUPLES	Budget	2018-07
8439	110.0	Infuzions	MIDAGE SINGLES/COUPLES	Mainstream	2018-07
24865	330.0	Cheezels	NEW FAMILIES	Budget	2018-07
42220	150.0	Kettle	OLDER FAMILIES	Budget	2018-07
179431	175.0	WW	YOUNG FAMILIES	Budget	2018-07
127345	330.0	Smiths	RETIREES	Budget	2018-07
4461	175.0	Kettle	MIDAGE SINGLES/COUPLES	Budget	2018-07
245076	110.0	Cobs	YOUNG SINGLES/COUPLES	Premium	2018-07
121447	165.0	Tyrrells	OLDER SINGLES/COUPLES	Premium	2018-07
4452	170.0	Smiths	MIDAGE SINGLES/COUPLES	Budget	2018-07
146307	134.0	Pringles	RETIREES	Mainstream	2018-07
13261	330.0	Smiths	MIDAGE SINGLES/COUPLES	Mainstream	2018-07
46848	160.0	WW	OLDER FAMILIES	Budget	2018-07
127427	175.0	Kettle	RETIREES	Budget	2018-07
167660	170.0	Doritos	RETIREES	Premium	2018-07
85756	110.0	Infuzions	OLDER SINGLES/COUPLES	Budget	2018-07
183710	150.0	Kettle	YOUNG FAMILIES	Budget	2018-07
33601	150.0	Smith	OLDER FAMILIES	Budget	2018-07
179245	150.0	Kettle	YOUNG FAMILIES	Budget	2018-07
15003	150.0	RRD	MIDAGE SINGLES/COUPLES	Mainstream	2018-07
20279	160.0	WW	MIDAGE SINGLES/COUPLES	Premium	2018-07
193065	170.0	Doritos	YOUNG FAMILIES	Mainstream	2018-07
147678	175.0	WW	RETIREES	Mainstream	2018-07
188245	170.0	Smiths	YOUNG FAMILIES	Budget	2018-07
121642	110.0	Cobs	OLDER SINGLES/COUPLES	Premium	2018-07
235573	175.0	Kettle	YOUNG SINGLES/COUPLES	Mainstream	2018-07
207700	110.0	Infuzions	YOUNG FAMILIES	Premium	2018-07
194757	175.0	WW	YOUNG FAMILIES	Mainstream	2018-07

```
10652       170.0     Smiths  MIDAGE SINGLES/COUPLES       Mainstream  2018-07
147427      330.0   Cheezels                RETIREES       Mainstream  2018-07
24736       210.0      Grain            NEW FAMILIES           Budget  2018-07
105518      125.0   Cheezels   OLDER SINGLES/COUPLES       Mainstream  2018-07
121544      330.0     Smiths   OLDER SINGLES/COUPLES          Premium  2018-07
188051      190.0    Cheetos          YOUNG FAMILIES           Budget  2018-07
4658        175.0      Thins  MIDAGE SINGLES/COUPLES           Budget  2018-07
119730      110.0     Infzns   OLDER SINGLES/COUPLES          Premium  2018-07
93757       165.0        RRD   OLDER SINGLES/COUPLES       Mainstream  2018-07
166594      134.0   Pringles                RETIREES          Premium  2018-07
135043      165.0        RRD                RETIREES           Budget  2018-07
243191      175.0        CCs   YOUNG SINGLES/COUPLES          Premium  2018-07
130992      150.0     Smiths                RETIREES           Budget  2018-07
166806      135.0     Kettle                RETIREES          Premium  2018-07
```

[592]:
```
measureOverTime = data.groupby(['STORE_NBR', 'MONTH_ID']).agg(
    totSales=('TOT_SALES', 'sum'),
    nCustomers=('LYLTY_CARD_NBR', 'nunique'),
    nTxnPerCust=('TXN_ID', lambda x: x.count() / x.nunique()),
    nChipsPerTxn=('PROD_QTY', lambda x: x.sum() / x.count()),
    avgPricePerUnit=('TOT_SALES', lambda x: x.sum() / x.count())
).reset_index()
```

[593]:
```
measureOverTime.head(50)
```

[593]:
```
    STORE_NBR MONTH_ID  totSales  nCustomers  nTxnPerCust  nChipsPerTxn  \
0           1  2018-07    188.90          47     1.000000      1.183673
1           1  2018-08    168.40          41     1.000000      1.268293
2           1  2018-09    268.10          57     1.000000      1.203390
3           1  2018-10    175.40          39     1.000000      1.275000
4           1  2018-11    184.80          44     1.000000      1.222222
5           1  2018-12    160.60          37     1.000000      1.200000
6           1  2019-01    149.70          35     1.000000      1.171429
7           1  2019-02    194.70          49     1.000000      1.137255
8           1  2019-03    185.20          43     1.000000      1.191489
9           1  2019-04    177.40          39     1.000000      1.300000
10          1  2019-05    207.10          43     1.000000      1.291667
11          1  2019-06    163.60          39     1.025641      1.250000
12          2  2018-07    140.50          36     1.000000      1.131579
13          2  2018-08    180.90          35     1.000000      1.282051
14          2  2018-09    133.90          32     1.000000      1.090909
15          2  2018-10    160.10          39     1.000000      1.048780
16          2  2018-11    143.30          33     1.000000      1.117647
17          2  2018-12    129.20          32     1.029412      1.057143
18          2  2019-01    158.70          41     1.000000      1.093023
19          2  2019-02    136.80          28     1.000000      1.161290
20          2  2019-03    174.00          40     1.000000      1.121951
```

21	2	2019-04	173.40	43	1.000000	1.111111
22	2	2019-05	179.80	45	1.000000	1.127660
23	2	2019-06	143.40	36	1.000000	1.052632
24	3	2018-07	1164.90	108	1.000000	1.962687
25	3	2018-08	998.15	106	1.024793	1.911290
26	3	2018-09	1011.30	102	1.000000	1.949153
27	3	2018-10	1017.50	103	1.000000	1.974359
28	3	2018-11	936.60	95	1.000000	1.918919
29	3	2018-12	1075.70	107	1.000000	1.983871
30	3	2019-01	980.30	97	1.017857	1.947368
31	3	2019-02	1146.70	113	1.015152	1.955224
32	3	2019-03	1083.60	106	1.008000	1.960317
33	3	2019-04	843.50	85	1.020000	1.882353
34	3	2019-05	905.10	100	1.000000	1.884956
35	3	2019-06	986.30	102	1.008621	1.948718
36	4	2018-07	1318.30	121	1.013333	1.986842
37	4	2018-08	1188.10	118	1.006993	1.916667
38	4	2018-09	1168.00	117	1.015038	2.000000
39	4	2018-10	1275.00	119	1.006803	2.000000
40	4	2018-11	1089.60	109	1.000000	2.000000
41	4	2018-12	1134.60	102	1.000000	2.000000
42	4	2019-01	1402.60	125	1.006452	1.993590
43	4	2019-02	832.40	86	1.010417	2.000000
44	4	2019-03	1110.80	111	1.000000	2.000000
45	4	2019-04	1159.10	112	1.007752	1.992308
46	4	2019-05	885.75	94	1.000000	1.803571
47	4	2019-06	1145.00	116	1.000000	2.000000
48	5	2018-07	763.80	86	1.000000	2.000000
49	5	2018-08	654.50	85	1.000000	1.909091

	avgPricePerUnit
0	3.855102
1	4.107317
2	4.544068
3	4.385000
4	4.106667
5	4.015000
6	4.277143
7	3.817647
8	3.940426
9	4.435000
10	4.314583
11	4.090000
12	3.697368
13	4.638462
14	4.057576
15	3.904878

```
16          4.214706
17          3.691429
18          3.690698
19          4.412903
20          4.243902
21          3.853333
22          3.825532
23          3.773684
24          8.693284
25          8.049597
26          8.570339
27          8.696581
28          8.437838
29          8.675000
30          8.599123
31          8.557463
32          8.600000
33          8.269608
34          8.009735
35          8.429915
36          8.673026
37          8.250694
38          8.651852
39          8.614865
40          8.579528
41          8.864062
42          8.991026
43          8.581443
44          8.746457
45          8.916154
46          7.908482
47          8.875969
48          6.881081
49          6.611111
```

[594]:

```
storesWithFullObs = measureOverTime.groupby('STORE_NBR').filter(lambda x:
    x['MONTH_ID'].nunique() == 12)['STORE_NBR'].unique()
preTrialMeasures = measureOverTime[(measureOverTime['MONTH_ID'] < '2019-02') &
    (measureOverTime['STORE_NBR'].isin(storesWithFullObs))]
```

[595]:

```
print(storesWithFullObs)
```

```
[ 1  2  3  4  5  6  7  8  9 10 12 13 14 15 16 17 18 19
 20 21 22 23 24 25 26 27 28 29 30 32 33 34 35 36 37 38
 39 40 41 42 43 45 46 47 48 49 50 51 52 53 54 55 56 57
 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
 77 78 79 80 81 82 83 84 86 87 88 89 90 91 93 94 95 96
```

```
 97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114
115 116 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151
152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169
170 171 172 173 174 175 176 178 179 180 181 182 183 184 185 186 187 188
189 190 191 192 194 195 196 197 198 199 200 201 202 203 204 205 207 208
209 210 212 213 214 215 216 217 219 220 221 222 223 224 225 226 227 228
229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246
247 248 249 250 251 253 254 255 256 257 258 259 260 261 262 263 264 265
266 267 268 269 270 271 272]
```

[596]:
```
preTrialMeasures.sort_values(by=['MONTH_ID'], inplace=True)
preTrialMeasures.head(2000)
```

```
C:\Users\Azlaan\AppData\Local\Temp\ipykernel_16972\785990281.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  preTrialMeasures.sort_values(by=['MONTH_ID'], inplace=True)
```

[596]:

	STORE_NBR	MONTH_ID	totSales	nCustomers	nTxnPerCust	nChipsPerTxn	\
0	1	2018-07	188.9	47	1.000000	1.183673	
2587	224	2018-07	9.0	2	1.000000	1.500000	
579	51	2018-07	111.0	27	1.000000	1.066667	
2599	225	2018-07	794.8	94	1.000000	2.000000	
1864	161	2018-07	35.5	5	1.000000	1.600000	
…	…	…	…	…	…	…	
418	37	2019-01	381.9	39	1.000000	1.953488	
2737	236	2019-01	798.4	93	1.009174	2.000000	
1043	91	2019-01	774.7	84	1.000000	1.924731	
406	36	2019-01	862.2	85	1.010000	2.000000	
3159	272	2019-01	392.4	44	1.000000	1.914894	

	avgPricePerUnit
0	3.855102
2587	4.500000
579	3.700000
2599	6.911304
1864	7.100000
…	…
418	8.881395
2737	7.258182
1043	8.330108
406	8.536634
3159	8.348936

[1813 rows x 7 columns]

We see in the data above that we wanted only transactions prior to 2019-02 and we have been successful.

We will now define functions called 'calculateCorrelation' and 'calculateMagnitudeDistance'.

[597]:
```
def calculateCorrelation(inputTable, metricCol, storeComparison):
    calcCorrTable = pd.DataFrame(columns=['Store1', 'Store2', 'corr_measure'])
    storeNumbers = inputTable['STORE_NBR'].unique()
    for i in storeNumbers:
        calculatedMeasure = pd.DataFrame({
            'Store1': storeComparison,
            'Store2': i,
            'corr_measure': inputTable[inputTable['STORE_NBR'] ==␣
↪storeComparison][metricCol].corr(inputTable[inputTable['STORE_NBR'] ==␣
↪i][metricCol])
        }, index=[0])
        calcCorrTable = pd.concat([calcCorrTable, calculatedMeasure],␣
↪ignore_index=True)
    return calcCorrTable
```

[598]:
```
def calculateMagnitudeDistance(inputTable, metricCol, storeComparison):
    calcDistTable = pd.DataFrame(columns=['Store1', 'Store2', 'MONTH_ID',␣
↪'measure'])
    storeNumbers = inputTable['STORE_NBR'].unique()
    for i in storeNumbers:
        calculatedMeasure = pd.DataFrame({
            'Store1': storeComparison,
            'Store2': i,
            'MONTH_ID': inputTable[inputTable['STORE_NBR'] ==␣
↪storeComparison]['MONTH_ID'],
            'measure': abs(inputTable[inputTable['STORE_NBR'] ==␣
↪storeComparison][metricCol].values - inputTable[inputTable['STORE_NBR'] ==␣
↪i][metricCol].values)
        })
        calcDistTable = pd.concat([calcDistTable, calculatedMeasure],␣
↪ignore_index=True)

    # Standardize the magnitude distance so that the measure ranges from 0 to 1
    minMaxDist = calcDistTable.groupby(['Store1', 'MONTH_ID'])['measure'].
↪agg(['min', 'max']).reset_index().rename(columns={'min': 'minDist', 'max':␣
↪'maxDist'})
    distTable = pd.merge(calcDistTable, minMaxDist, on=['Store1', 'MONTH_ID'])
    distTable['magnitudeMeasure'] = 1 - (distTable['measure'] -␣
↪distTable['minDist']) / (distTable['maxDist'] - distTable['minDist'])
```

```
    finalDistTable = distTable.groupby(['Store1',␣
↪'Store2'])['magnitudeMeasure'].mean().reset_index().
↪rename(columns={'magnitudeMeasure': 'mag_measure'})
    return finalDistTable
```

0.2 Trial Stores Data

Now let us calculate the data for our Trial Store Number 77.

[599]:
```
# Use the function you created to calculate correlations against store 77 using␣
↪total sales and number of customers.
trial_store = 77
corr_nSales = calculateCorrelation(preTrialMeasures, 'totSales', trial_store)
corr_nCustomers = calculateCorrelation(preTrialMeasures, 'nCustomers',␣
↪trial_store)
```

We will now calculate the magnitute of sales and customers using pretrial values.

[600]:
```
# Then, use the functions for calculating magnitude.
magnitude_nSales = calculateMagnitudeDistance(preTrialMeasures, 'totSales',␣
↪trial_store)
magnitude_nCustomers = calculateMagnitudeDistance(preTrialMeasures,␣
↪'nCustomers', trial_store)
```

We combine the tables of the two datasets and make two more new tables. One with sales data and the other with the customer data for the trial store.

[601]:
```
# Create a combined score composed of correlation and magnitude, by first␣
↪merging the correlations table with the magnitude table.
corr_weight = 0.5
score_nSales = pd.merge(corr_nSales, magnitude_nSales, on=['Store1', 'Store2'])
score_nSales['scoreNSales'] = corr_weight * score_nSales['corr_measure'] + (1 -␣
↪corr_weight) * score_nSales['mag_measure']
score_nCustomers = pd.merge(corr_nCustomers, magnitude_nCustomers,␣
↪on=['Store1', 'Store2'])
score_nCustomers['scoreNCust'] = corr_weight * score_nCustomers['corr_measure']␣
↪+ (1 - corr_weight) * score_nCustomers['mag_measure']
```

[602]:
```
# Combine scores across the drivers by first merging our sales scores and␣
↪customer scores into a single table
score_Control = pd.merge(score_nSales, score_nCustomers, on=['Store1',␣
↪'Store2'])
score_Control['finalControlScore'] = score_Control['scoreNSales'] * 0.5 +␣
↪score_Control['scoreNCust'] * 0.5
```

[603]:
```
control_store = score_Control[score_Control['Store1'] == 77].
↪sort_values(by='finalControlScore', ascending=False).iloc[1]['Store2']
print(control_store)
```

```
1
```

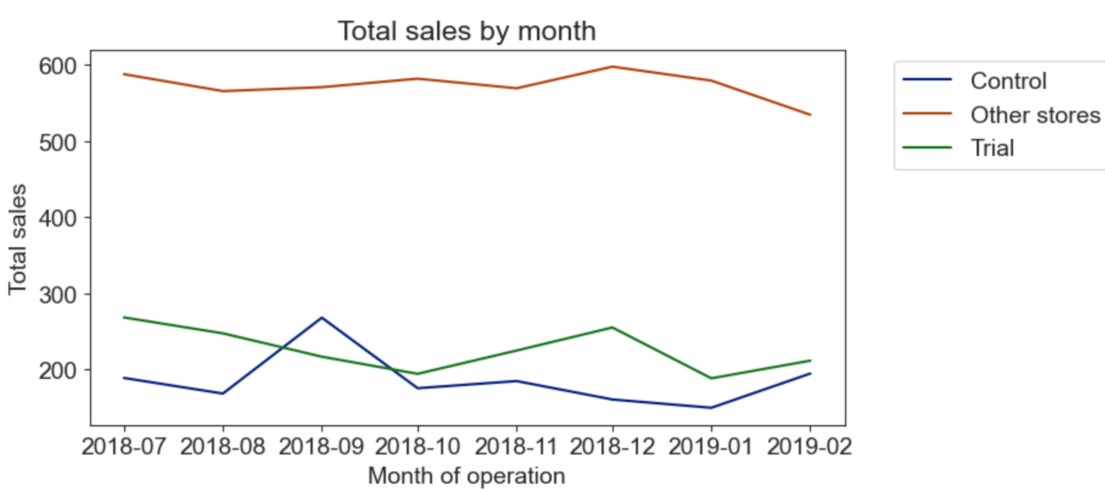
[604]:
```
# Visual checks on trends based on the drivers
measureOverTimeSales = measureOverTime.copy()
measureOverTimeSales['Store_type'] = measureOverTimeSales['STORE_NBR'].
↪apply(lambda x: 'Trial' if x == trial_store else ('Control' if x ==␣
↪control_store else 'Other stores'))
measureOverTimeSales = measureOverTimeSales.groupby(['MONTH_ID',␣
↪'Store_type'])['totSales'].mean().reset_index()
measureOverTimeSales['TransactionMonth'] = pd.
↪to_datetime(measureOverTimeSales['MONTH_ID'], format='%Y-%m')
pastSales = measureOverTimeSales[measureOverTimeSales['MONTH_ID'] < '2019-03']
```

We will now plot our data.

[605]:
```
# Plot total sales by month
ax1 = sns.lineplot(data=pastSales, x='TransactionMonth', y='totSales',␣
↪hue='Store_type')
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Total sales by month')

# Move hue legend to the top right corner outside plot
ax1.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))

plt.show()
```

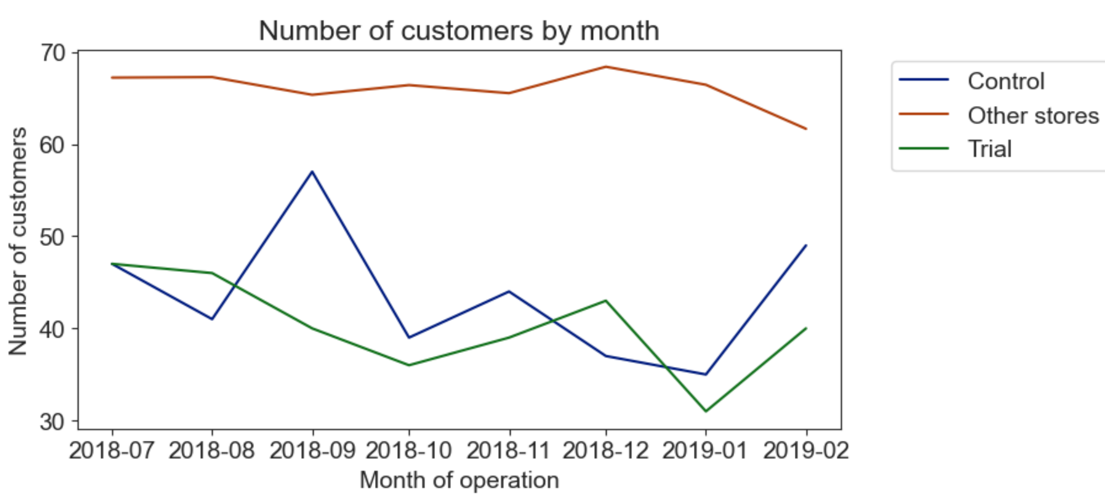


[606]:
```
# Conduct visual checks on customer count trends by comparing the trial store␣
↪to the control store and other stores.
measureOverTimeCusts = measureOverTime.copy()
measureOverTimeCusts['Store_type'] = measureOverTimeCusts['STORE_NBR'].
↪apply(lambda x: 'Trial' if x == trial_store else ('Control' if x ==␣
↪control_store else 'Other stores'))
measureOverTimeCusts = measureOverTimeCusts.groupby(['MONTH_ID',␣
↪'Store_type'])['nCustomers'].mean().reset_index()
measureOverTimeCusts['TransactionMonth'] = pd.
↪to_datetime(measureOverTimeCusts['MONTH_ID'], format='%Y-%m')
pastCustomers = measureOverTimeCusts[measureOverTimeCusts['MONTH_ID'] <␣
↪'2019-03']
```

[607]:
```
# Plot number of customers by month
ax2 = sns.lineplot(data=pastCustomers, x='TransactionMonth', y='nCustomers',␣
↪hue='Store_type')
plt.xlabel('Month of operation')
plt.ylabel('Number of customers')
plt.title('Number of customers by month')

# Move hue legend to the top right corner outside plot
ax2.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))

plt.show()
```



We will be scaling our control data to match our pretrial sales.

[608]:
```
# Scale pre-trial control sales to match pre-trial trial store sales
```

```
scalingFactorForControlSales = preTrialMeasures[(preTrialMeasures['STORE_NBR']␣
↪== trial_store) & (preTrialMeasures['MONTH_ID'] < '2019-02')]['totSales'].
↪sum() / preTrialMeasures[(preTrialMeasures['STORE_NBR'] == control_store) &␣
↪(preTrialMeasures['MONTH_ID'] < '2019-02')]['totSales'].sum()
```

[609]:
```
# Apply the scaling factor
measureOverTimeSales = measureOverTime.copy()
scaledControlSales = measureOverTimeSales[measureOverTimeSales['STORE_NBR'] ==␣
↪control_store].copy()
scaledControlSales['controlSales'] = scaledControlSales['totSales'] *␣
↪scalingFactorForControlSales
```

[610]:
```
# Calculate the percentage difference between scaled control sales and trial␣
↪sales
percentageDiff = pd.merge(scaledControlSales,␣
↪measureOverTimeSales[measureOverTimeSales['STORE_NBR'] == trial_store],␣
↪on='MONTH_ID')
percentageDiff['percentageDiff'] = (abs(percentageDiff['controlSales'] -␣
↪percentageDiff['totSales_y']) / percentageDiff[['controlSales',␣
↪'totSales_y']].mean(axis=1)) * 100
```

[611]:
```
# 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
stdDev = percentageDiff[percentageDiff['MONTH_ID'] <␣
↪'2019-02']['percentageDiff'].std()

# Note that there are 8 months in the pre-trial period hence 8 - 1 = 7 degrees␣
↪of freedom
degreesOfFreedom = 7
```

[612]:
```
# We will test with a null hypothesis of there being 0 difference between trial␣
↪and control stores.
# Calculate the t-values for the trial months
percentageDiff['tValue'] = (percentageDiff['percentageDiff'] - 0) / stdDev
```

[613]:
```
critical_t_value = t.isf(0.05, df=degreesOfFreedom)
```

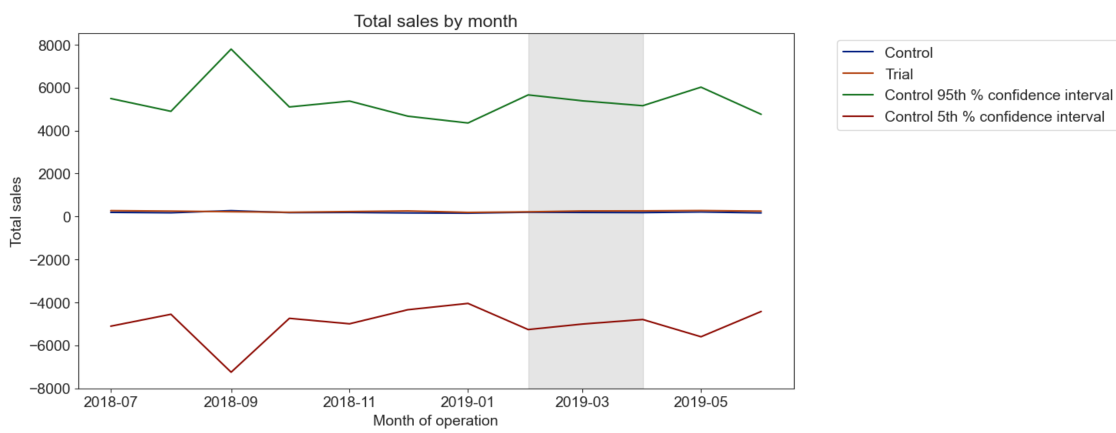
[614]:
```
# Trial and control store total sales
pastSales = measureOverTimeSales.copy()
pastSales['Store_type'] = pastSales['STORE_NBR'].apply(lambda x: 'Trial' if x␣
↪== trial_store else 'Control' if x == control_store else np.nan)
pastSales['TransactionMonth'] = pastSales['MONTH_ID']
pastSales = pastSales[pastSales['Store_type'].isin(['Trial', 'Control'])]
```

[615]:
```
# Control store 95th percentile
pastSales_Controls95 = pastSales[pastSales['Store_type'] == 'Control'].copy()
pastSales_Controls95['totSales'] = pastSales_Controls95['totSales'] * (1 +␣
↪stdDev * 2)
pastSales_Controls95['Store_type'] = 'Control 95th % confidence interval'
```

[616]:
```
# Control store 5th percentile
pastSales_Controls5 = pastSales[pastSales['Store_type'] == 'Control'].copy()
pastSales_Controls5['totSales'] = pastSales_Controls5['totSales'] * (1 - stdDev␣
↪* 2)
pastSales_Controls5['Store_type'] = 'Control 5th % confidence interval'
```

[617]:
```
trialAssessment = pd.concat([pastSales, pastSales_Controls95,␣
↪pastSales_Controls5])
```

[618]:
```
# Plotting these in one nice graph
trialAssessment['TransactionMonth'] = pd.
↪to_datetime(trialAssessment['TransactionMonth'], format='%Y-%m')
fig, ax = plt.subplots(figsize=(12, 6))
sns.lineplot(data=trialAssessment, x='TransactionMonth', y='totSales',␣
↪hue='Store_type', ax=ax)
ax.axvspan(pd.to_datetime('2019-02', format='%Y-%m'), pd.to_datetime('2019-04',␣
↪format='%Y-%m'), alpha=0.2, color='grey')
ax.set(xlabel='Month of operation', ylabel='Total sales', title='Total sales by␣
↪month')
ax.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))
plt.show()
```



[619]:
```
# 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
```

```
stdDev = percentageDiff[percentageDiff['MONTH_ID'] <␣
↪'2019-02']['percentageDiff'].std()
degreesOfFreedom = 7
```

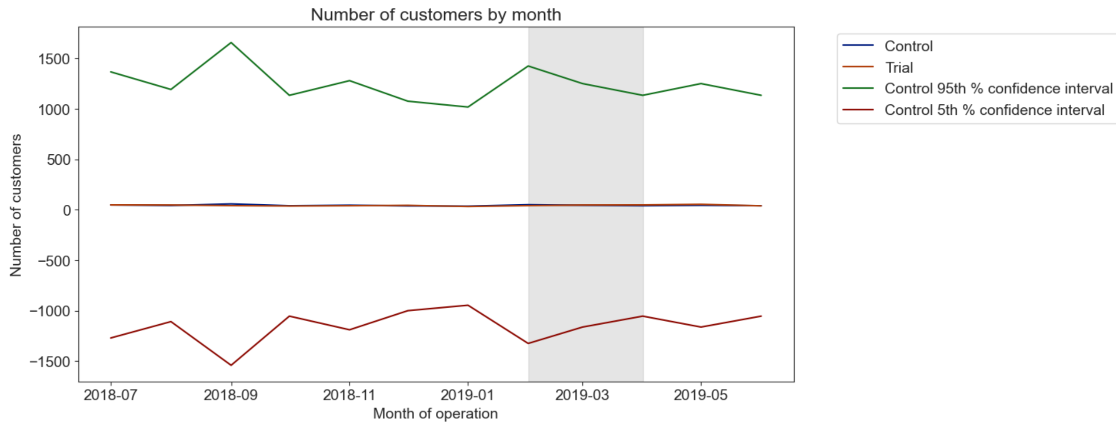
[620]:
```
# Trial and control store number of customers
pastCustomers = measureOverTimeCusts.copy()
pastCustomers['nCusts'] = pastCustomers.groupby(['MONTH_ID',␣
↪'Store_type'])['nCustomers'].transform('mean')
pastCustomers = pastCustomers[pastCustomers['Store_type'].isin(['Trial',␣
↪'Control'])]
```

[621]:
```
# Control store 95th percentile
pastCustomers_Controls95 = pastCustomers[pastCustomers['Store_type'] ==␣
↪'Control'].copy()
pastCustomers_Controls95['nCusts'] = pastCustomers_Controls95['nCusts'] * (1 +␣
↪stdDev * 2)
pastCustomers_Controls95['Store_type'] = 'Control 95th % confidence interval'
```

[622]:
```
# Control store 5th percentile
pastCustomers_Controls5 = pastCustomers[pastCustomers['Store_type'] ==␣
↪'Control'].copy()
pastCustomers_Controls5['nCusts'] = pastCustomers_Controls5['nCusts'] * (1 -␣
↪stdDev * 2)
pastCustomers_Controls5['Store_type'] = 'Control 5th % confidence interval'
```

[623]:
```
trialAssessment = pd.concat([pastCustomers, pastCustomers_Controls95,␣
↪pastCustomers_Controls5])
```

[624]:
```
# Plot everything into one nice graph
fig, ax = plt.subplots(figsize=(12, 6))
sns.lineplot(data=trialAssessment, x='TransactionMonth', y='nCusts',␣
↪hue='Store_type', ax=ax)
ax.axvspan(pd.to_datetime('2019-02', format='%Y-%m'), pd.to_datetime('2019-04',␣
↪format='%Y-%m'), alpha=0.2, color='grey')
ax.set(xlabel='Month of operation', ylabel='Number of customers', title='Number␣
↪of customers by month')
ax.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))
plt.show()
```



[625]:
```
# Calculate the metrics below as we did for the first trial store.
measureOverTime = data.groupby(['STORE_NBR', 'MONTH_ID']).agg(
    nCustomers=('LYLTY_CARD_NBR', 'nunique'),
    nTxn=('TXN_ID', 'nunique'),
    nChips=('PROD_QTY', 'sum'),
    totSales=('TOT_SALES', 'sum')
).reset_index()
measureOverTime['nTxnPerCust'] = measureOverTime['nTxn'] /␣
↪measureOverTime['nCustomers']
measureOverTime['nChipsPerTxn'] = measureOverTime['nChips'] /␣
↪measureOverTime['nTxn']
measureOverTime['avgPricePerUnit'] = measureOverTime['totSales'] /␣
↪measureOverTime['nChips']
measureOverTime = measureOverTime.sort_values(by=['STORE_NBR', 'MONTH_ID'])
```

Trial Store Number 86

We will be repeating all the steps that we performed for the previous store.

[626]:
```
# Use the functions we created earlier to calculate correlations and magnitude␣
↪for each potential control store
trial_store = 86
corr_nSales = calculateCorrelation(preTrialMeasures, 'totSales', trial_store)
corr_nCustomers = calculateCorrelation(preTrialMeasures, 'nCustomers',␣
↪trial_store)
magnitude_nSales = calculateMagnitudeDistance(preTrialMeasures, 'totSales',␣
↪trial_store)
magnitude_nCustomers = calculateMagnitudeDistance(preTrialMeasures,␣
↪'nCustomers', trial_store)
```

[627]:
```
# Create a combined score composed of correlation and magnitude, by first␣
↪merging the correlations table with the magnitude table.
```

```
corr_weight = 0.5
score_nSales = pd.merge(corr_nSales, magnitude_nSales, on=['Store1', 'Store2'])
score_nSales['scoreNSales'] = corr_weight * score_nSales['corr_measure'] + (1 -␣
↪corr_weight) * score_nSales['mag_measure']
score_nCustomers = pd.merge(corr_nCustomers, magnitude_nCustomers,␣
↪on=['Store1', 'Store2'])
score_nCustomers['scoreNCust'] = corr_weight * score_nCustomers['corr_measure']␣
↪+ (1 - corr_weight) * score_nCustomers['mag_measure']
```

[628]:
```
# Combine scores across the drivers by first merging our sales scores and␣
↪customer scores into a single table
score_Control = pd.merge(score_nSales, score_nCustomers, on=['Store1',␣
↪'Store2'])
score_Control['finalControlScore'] = score_Control['scoreNSales'] * 0.5 +␣
↪score_Control['scoreNCust'] * 0.5
```

[629]:
```
control_store = score_Control[score_Control['Store1'] == 86].
↪sort_values(by='finalControlScore', ascending=False).iloc[1]['Store2']
print(control_store)
```

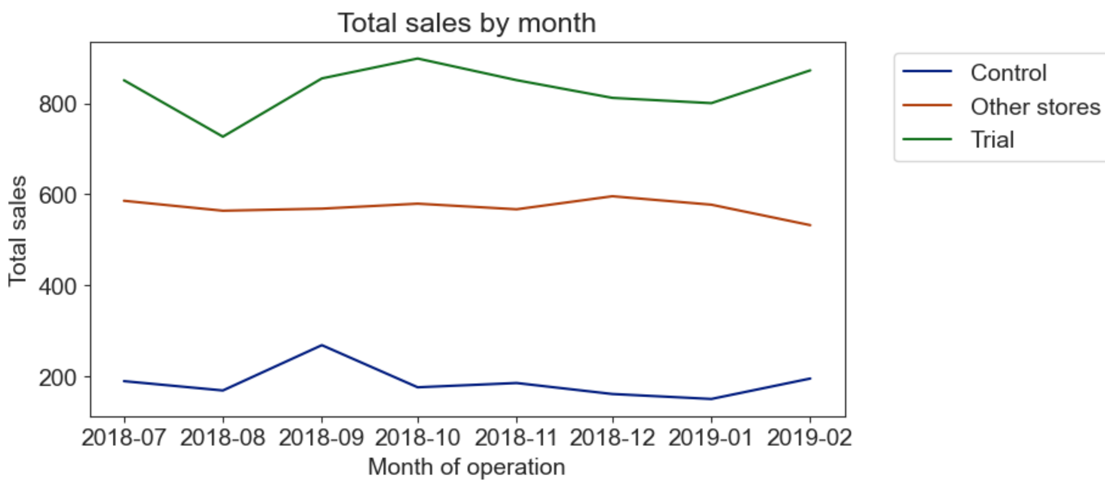
```
1
```

[630]:
```
# Visual checks on trends based on the drivers
measureOverTimeSales = measureOverTime.copy()
measureOverTimeSales['Store_type'] = measureOverTimeSales['STORE_NBR'].
↪apply(lambda x: 'Trial' if x == trial_store else ('Control' if x ==␣
↪control_store else 'Other stores'))
measureOverTimeSales = measureOverTimeSales.groupby(['MONTH_ID',␣
↪'Store_type'])['totSales'].mean().reset_index()
measureOverTimeSales['TransactionMonth'] = pd.
↪to_datetime(measureOverTimeSales['MONTH_ID'], format='%Y-%m')
pastSales = measureOverTimeSales[measureOverTimeSales['MONTH_ID'] < '2019-03']
```

[631]:
```
# Plot total sales by month
ax1 = sns.lineplot(data=pastSales, x='TransactionMonth', y='totSales',␣
↪hue='Store_type')
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Total sales by month')

# Move hue legend to the top right corner outside plot
ax1.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))

plt.show()
```

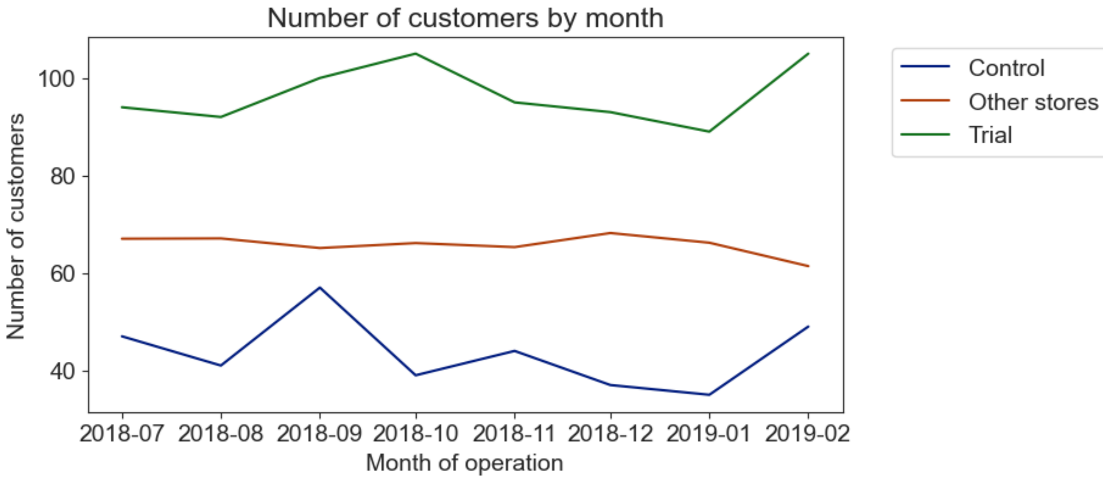


[632]:
```
# Conduct visual checks on customer count trends by comparing the trial store␣
↪to the control store and other stores.
measureOverTimeCusts = measureOverTime.copy()
measureOverTimeCusts['Store_type'] = measureOverTimeCusts['STORE_NBR'].
↪apply(lambda x: 'Trial' if x == trial_store else ('Control' if x ==␣
↪control_store else 'Other stores'))
measureOverTimeCusts = measureOverTimeCusts.groupby(['MONTH_ID',␣
↪'Store_type'])['nCustomers'].mean().reset_index()
measureOverTimeCusts['TransactionMonth'] = pd.
↪to_datetime(measureOverTimeCusts['MONTH_ID'], format='%Y-%m')
pastCustomers = measureOverTimeCusts[measureOverTimeCusts['MONTH_ID'] <␣
↪'2019-03']
```

[633]:
```
# Plot number of customers by month
ax2 = sns.lineplot(data=pastCustomers, x='TransactionMonth', y='nCustomers',␣
↪hue='Store_type')
plt.xlabel('Month of operation')
plt.ylabel('Number of customers')
plt.title('Number of customers by month')

# Move hue legend to the top right corner outside plot
ax2.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))

plt.show()
```



[634]:

```
# Scale pre-trial control sales to match pre-trial trial store sales
scalingFactorForControlSales = preTrialMeasures[(preTrialMeasures['STORE_NBR']␣
↪== trial_store) & (preTrialMeasures['MONTH_ID'] < '2019-02')]['totSales'].
↪sum() / preTrialMeasures[(preTrialMeasures['STORE_NBR'] == control_store) &␣
↪(preTrialMeasures['MONTH_ID'] < '2019-02')]['totSales'].sum()
```

[635]:

```
# Apply the scaling factor
measureOverTimeSales = measureOverTime.copy()
scaledControlSales = measureOverTimeSales[measureOverTimeSales['STORE_NBR'] ==␣
↪control_store].copy()
scaledControlSales['controlSales'] = scaledControlSales['totSales'] *␣
↪scalingFactorForControlSales
```

[636]:

```
# Calculate the percentage difference between scaled control sales and trial␣
↪sales
percentageDiff = pd.merge(scaledControlSales,␣
↪measureOverTimeSales[measureOverTimeSales['STORE_NBR'] == trial_store],␣
↪on='MONTH_ID')
percentageDiff['percentageDiff'] = (abs(percentageDiff['controlSales'] -␣
↪percentageDiff['totSales_y']) / percentageDiff[['controlSales',␣
↪'totSales_y']].mean(axis=1)) * 100
```

[637]:

```
# 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
stdDev = percentageDiff[percentageDiff['MONTH_ID'] <␣
↪'2019-02']['percentageDiff'].std()

# Note that there are 8 months in the pre-trial period hence 8 - 1 = 7 degrees␣
↪of freedom
```

```
degreesOfFreedom = 7
```

[638]:
```
# We will test with a null hypothesis of there being 0 difference between trial␣
↪and control stores.
# Calculate the t-values for the trial months
percentageDiff['tValue'] = (percentageDiff['percentageDiff'] - 0) / stdDev
```

[639]:
```
critical_t_value = t.isf(0.05, df=degreesOfFreedom)
```

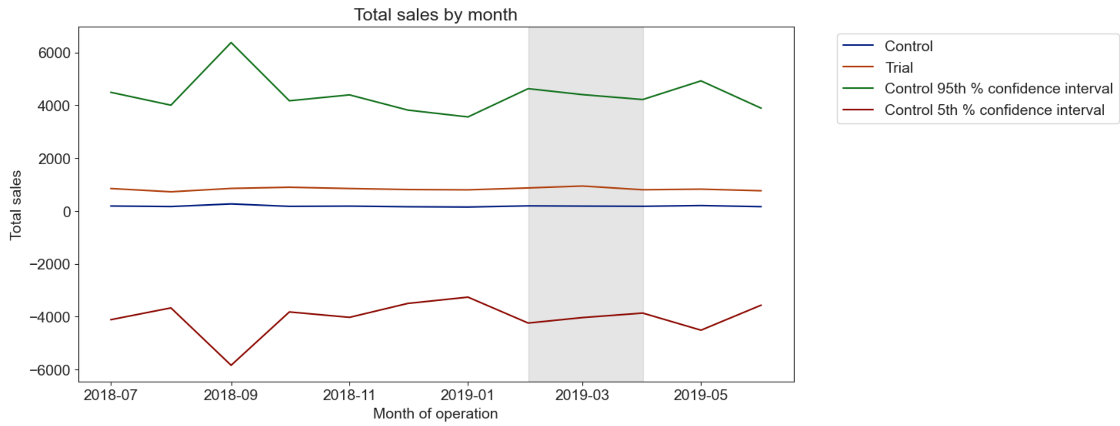
[640]:
```
# Trial and control store total sales
pastSales = measureOverTimeSales.copy()
pastSales['Store_type'] = pastSales['STORE_NBR'].apply(lambda x: 'Trial' if x␣
↪== trial_store else 'Control' if x == control_store else np.nan)
pastSales['TransactionMonth'] = pastSales['MONTH_ID']
pastSales = pastSales[pastSales['Store_type'].isin(['Trial', 'Control'])]
```

[641]:
```
# Control store 95th percentile
pastSales_Controls95 = pastSales[pastSales['Store_type'] == 'Control'].copy()
pastSales_Controls95['totSales'] = pastSales_Controls95['totSales'] * (1 +␣
↪stdDev * 2)
pastSales_Controls95['Store_type'] = 'Control 95th % confidence interval'
```

[642]:
```
# Control store 5th percentile
pastSales_Controls5 = pastSales[pastSales['Store_type'] == 'Control'].copy()
pastSales_Controls5['totSales'] = pastSales_Controls5['totSales'] * (1 - stdDev␣
↪* 2)
pastSales_Controls5['Store_type'] = 'Control 5th % confidence interval'
```

[643]:
```
trialAssessment = pd.concat([pastSales, pastSales_Controls95,␣
↪pastSales_Controls5])
```

[644]:
```
# Plotting these in one nice graph
trialAssessment['TransactionMonth'] = pd.
↪to_datetime(trialAssessment['TransactionMonth'], format='%Y-%m')
fig, ax = plt.subplots(figsize=(12, 6))
sns.lineplot(data=trialAssessment, x='TransactionMonth', y='totSales',␣
↪hue='Store_type', ax=ax)
ax.axvspan(pd.to_datetime('2019-02', format='%Y-%m'), pd.to_datetime('2019-04',␣
↪format='%Y-%m'), alpha=0.2, color='grey')
ax.set(xlabel='Month of operation', ylabel='Total sales', title='Total sales by␣
↪month')
ax.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))
plt.show()
```



[645]:
```
# 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
stdDev = percentageDiff[percentageDiff['MONTH_ID'] <␣
↪'2019-02']['percentageDiff'].std()
degreesOfFreedom = 7
```

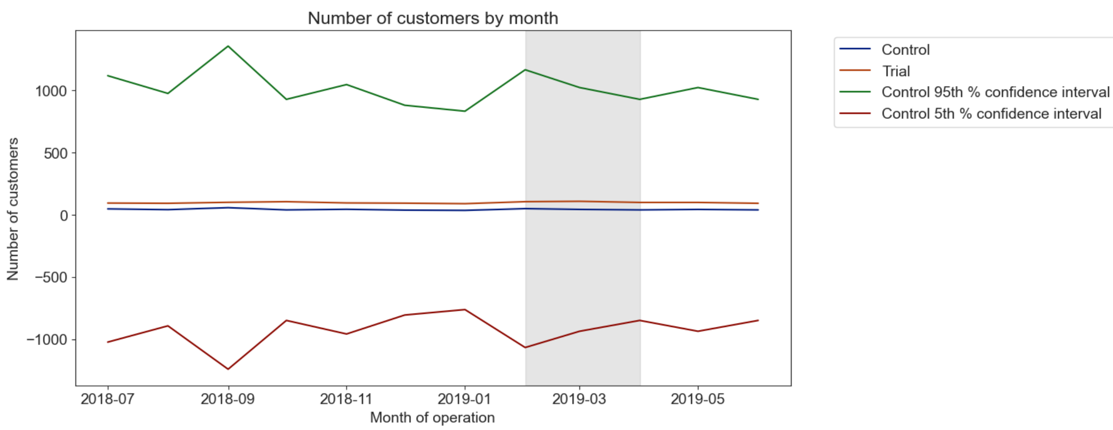
[646]:
```
# Trial and control store number of customers
pastCustomers = measureOverTimeCusts.copy()
pastCustomers['nCusts'] = pastCustomers.groupby(['MONTH_ID',␣
↪'Store_type'])['nCustomers'].transform('mean')
pastCustomers = pastCustomers[pastCustomers['Store_type'].isin(['Trial',␣
↪'Control'])]
```

[647]:
```
# Control store 95th percentile
pastCustomers_Controls95 = pastCustomers[pastCustomers['Store_type'] ==␣
↪'Control'].copy()
pastCustomers_Controls95['nCusts'] = pastCustomers_Controls95['nCusts'] * (1 +␣
↪stdDev * 2)
pastCustomers_Controls95['Store_type'] = 'Control 95th % confidence interval'
```

[648]:
```
# Control store 5th percentile
pastCustomers_Controls5 = pastCustomers[pastCustomers['Store_type'] ==␣
↪'Control'].copy()
pastCustomers_Controls5['nCusts'] = pastCustomers_Controls5['nCusts'] * (1 -␣
↪stdDev * 2)
pastCustomers_Controls5['Store_type'] = 'Control 5th % confidence interval'
```

[649]:
```
trialAssessment = pd.concat([pastCustomers, pastCustomers_Controls95,␣
↪pastCustomers_Controls5])
```

[650]:
```
# Plot everything into one nice graph
fig, ax = plt.subplots(figsize=(12, 6))
sns.lineplot(data=trialAssessment, x='TransactionMonth', y='nCusts',␣
↪hue='Store_type', ax=ax)
ax.axvspan(pd.to_datetime('2019-02', format='%Y-%m'), pd.to_datetime('2019-04',␣
↪format='%Y-%m'), alpha=0.2, color='grey')
ax.set(xlabel='Month of operation', ylabel='Number of customers', title='Number␣
↪of customers by month')
ax.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))
plt.show()
```



[651]:
```
# Calculate the metrics below as we did for the first trial store.
measureOverTime = data.groupby(['STORE_NBR', 'MONTH_ID']).agg(
    nCustomers=('LYLTY_CARD_NBR', 'nunique'),
    nTxn=('TXN_ID', 'nunique'),
    nChips=('PROD_QTY', 'sum'),
    totSales=('TOT_SALES', 'sum')
).reset_index()
measureOverTime['nTxnPerCust'] = measureOverTime['nTxn'] /␣
↪measureOverTime['nCustomers']
measureOverTime['nChipsPerTxn'] = measureOverTime['nChips'] /␣
↪measureOverTime['nTxn']
measureOverTime['avgPricePerUnit'] = measureOverTime['totSales'] /␣
↪measureOverTime['nChips']
measureOverTime = measureOverTime.sort_values(by=['STORE_NBR', 'MONTH_ID'])
```

Trial store Number 88

We will be using same methods as previous stores.

[652]:
```
# Use the function you created to calculate correlations against store 77 using␣
↪total sales and number of customers.
```

```
trial_store = 88
corr_nSales = calculateCorrelation(preTrialMeasures, 'totSales', trial_store)
corr_nCustomers = calculateCorrelation(preTrialMeasures, 'nCustomers',␣
↪trial_store)
```

[653]:
```
# Then, use the functions for calculating magnitude.
magnitude_nSales = calculateMagnitudeDistance(preTrialMeasures, 'totSales',␣
↪trial_store)
magnitude_nCustomers = calculateMagnitudeDistance(preTrialMeasures,␣
↪'nCustomers', trial_store)
```

[654]:
```
# Create a combined score composed of correlation and magnitude, by first␣
↪merging the correlations table with the magnitude table.
corr_weight = 0.5
score_nSales = pd.merge(corr_nSales, magnitude_nSales, on=['Store1', 'Store2'])
score_nSales['scoreNSales'] = corr_weight * score_nSales['corr_measure'] + (1 -␣
↪corr_weight) * score_nSales['mag_measure']
score_nCustomers = pd.merge(corr_nCustomers, magnitude_nCustomers,␣
↪on=['Store1', 'Store2'])
score_nCustomers['scoreNCust'] = corr_weight * score_nCustomers['corr_measure']␣
↪+ (1 - corr_weight) * score_nCustomers['mag_measure']
```

[655]:
```
# Combine scores across the drivers by first merging our sales scores and␣
↪customer scores into a single table
score_Control = pd.merge(score_nSales, score_nCustomers, on=['Store1',␣
↪'Store2'])
score_Control['finalControlScore'] = score_Control['scoreNSales'] * 0.5 +␣
↪score_Control['scoreNCust'] * 0.5
```

[656]:
```
control_store = score_Control[score_Control['Store1'] == 88].
↪sort_values(by='finalControlScore', ascending=False).iloc[1]['Store2']
print(control_store)
```

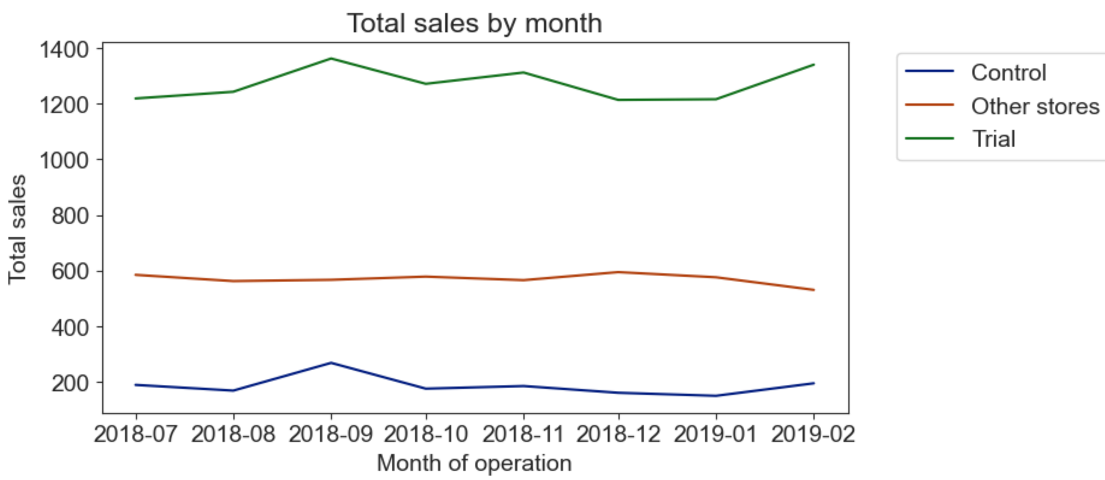
```
1
```

[657]:
```
# Visual checks on trends based on the drivers
measureOverTimeSales = measureOverTime.copy()
measureOverTimeSales['Store_type'] = measureOverTimeSales['STORE_NBR'].
↪apply(lambda x: 'Trial' if x == trial_store else ('Control' if x ==␣
↪control_store else 'Other stores'))
measureOverTimeSales = measureOverTimeSales.groupby(['MONTH_ID',␣
↪'Store_type'])['totSales'].mean().reset_index()
measureOverTimeSales['TransactionMonth'] = pd.
↪to_datetime(measureOverTimeSales['MONTH_ID'], format='%Y-%m')
pastSales = measureOverTimeSales[measureOverTimeSales['MONTH_ID'] < '2019-03']
```

[658]:
```
# Plot total sales by month
ax1 = sns.lineplot(data=pastSales, x='TransactionMonth', y='totSales',␣
↪hue='Store_type')
plt.xlabel('Month of operation')
plt.ylabel('Total sales')
plt.title('Total sales by month')

# Move hue legend to the top right corner outside plot
ax1.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))

plt.show()
```



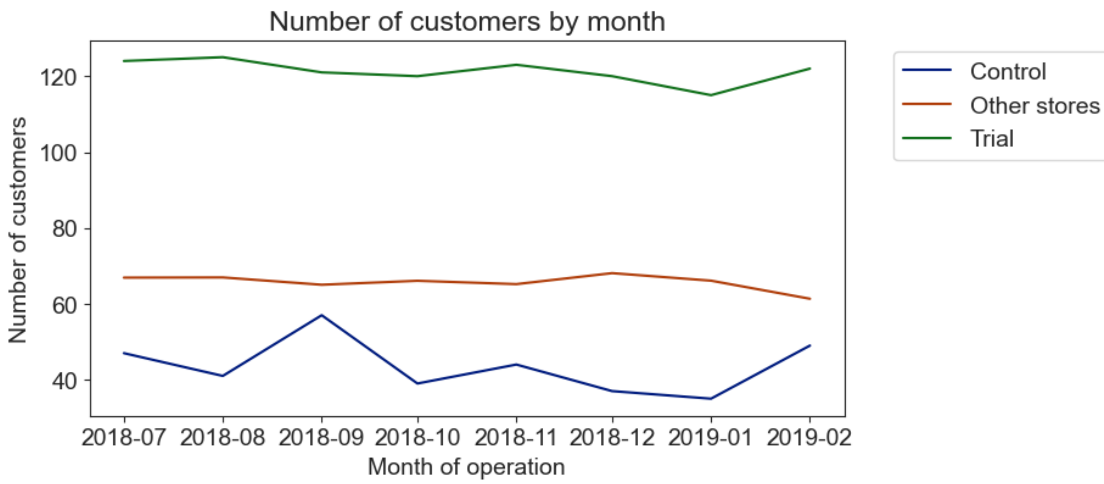
[659]:
```
# Conduct visual checks on customer count trends by comparing the trial store␣
↪to the control store and other stores.
measureOverTimeCusts = measureOverTime.copy()
measureOverTimeCusts['Store_type'] = measureOverTimeCusts['STORE_NBR'].
↪apply(lambda x: 'Trial' if x == trial_store else ('Control' if x ==␣
↪control_store else 'Other stores'))
measureOverTimeCusts = measureOverTimeCusts.groupby(['MONTH_ID',␣
↪'Store_type'])['nCustomers'].mean().reset_index()
measureOverTimeCusts['TransactionMonth'] = pd.
↪to_datetime(measureOverTimeCusts['MONTH_ID'], format='%Y-%m')
pastCustomers = measureOverTimeCusts[measureOverTimeCusts['MONTH_ID'] <␣
↪'2019-03']
```

[660]:
```
# Plot number of customers by month
ax2 = sns.lineplot(data=pastCustomers, x='TransactionMonth', y='nCustomers',␣
↪hue='Store_type')
plt.xlabel('Month of operation')
plt.ylabel('Number of customers')
```

```
plt.title('Number of customers by month')

# Move hue legend to the top right corner outside plot
ax2.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))

plt.show()
```



[661]:
```
# Scale pre-trial control sales to match pre-trial trial store sales
scalingFactorForControlSales = preTrialMeasures[(preTrialMeasures['STORE_NBR']
  ↪== trial_store) & (preTrialMeasures['MONTH_ID'] < '2019-02')]['totSales'].
  ↪sum() / preTrialMeasures[(preTrialMeasures['STORE_NBR'] == control_store) &
  ↪(preTrialMeasures['MONTH_ID'] < '2019-02')]['totSales'].sum()
```

[662]:
```
# Apply the scaling factor
measureOverTimeSales = measureOverTime.copy()
scaledControlSales = measureOverTimeSales[measureOverTimeSales['STORE_NBR'] ==
  ↪control_store].copy()
scaledControlSales['controlSales'] = scaledControlSales['totSales'] *
  ↪scalingFactorForControlSales
```

[663]:
```
# Calculate the percentage difference between scaled control sales and trial
  ↪sales
percentageDiff = pd.merge(scaledControlSales,
  ↪measureOverTimeSales[measureOverTimeSales['STORE_NBR'] == trial_store],
  ↪on='MONTH_ID')
percentageDiff['percentageDiff'] = (abs(percentageDiff['controlSales'] -
  ↪percentageDiff['totSales_y']) / percentageDiff[['controlSales',
  ↪'totSales_y']].mean(axis=1)) * 100
```

[664]:
```
# 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
stdDev = percentageDiff[percentageDiff['MONTH_ID'] <␣
↪'2019-02']['percentageDiff'].std()

# Note that there are 8 months in the pre-trial period hence 8 - 1 = 7 degrees␣
↪of freedom
degreesOfFreedom = 7
```

[665]:
```
# We will test with a null hypothesis of there being 0 difference between trial␣
↪and control stores.
# Calculate the t-values for the trial months
percentageDiff['tValue'] = (percentageDiff['percentageDiff'] - 0) / stdDev
```

[666]:
```
critical_t_value = t.isf(0.05, df=degreesOfFreedom)
```

[667]:
```
# Trial and control store total sales
pastSales = measureOverTimeSales.copy()
pastSales['Store_type'] = pastSales['STORE_NBR'].apply(lambda x: 'Trial' if x␣
↪== trial_store else 'Control' if x == control_store else np.nan)
pastSales['TransactionMonth'] = pastSales['MONTH_ID']
pastSales = pastSales[pastSales['Store_type'].isin(['Trial', 'Control'])]
```

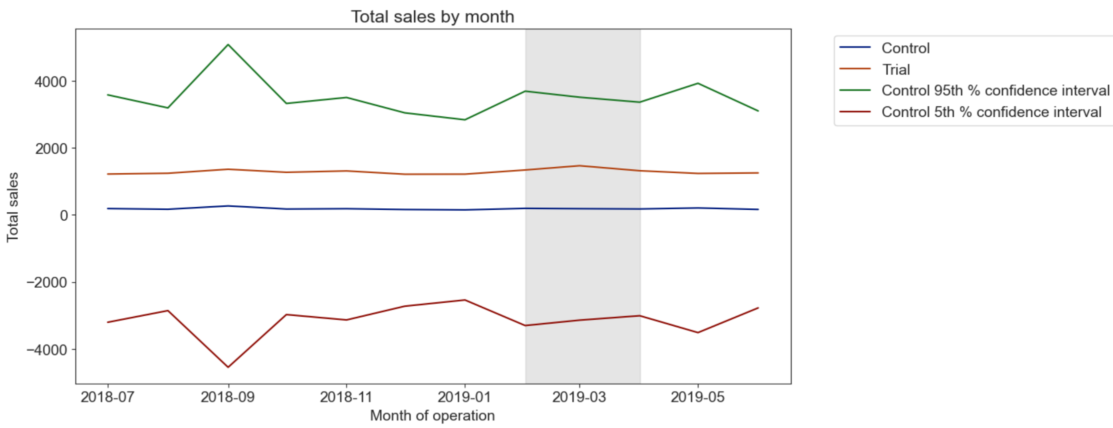
[668]:
```
# Control store 95th percentile
pastSales_Controls95 = pastSales[pastSales['Store_type'] == 'Control'].copy()
pastSales_Controls95['totSales'] = pastSales_Controls95['totSales'] * (1 +␣
↪stdDev * 2)
pastSales_Controls95['Store_type'] = 'Control 95th % confidence interval'
```

[669]:
```
# Control store 5th percentile
pastSales_Controls5 = pastSales[pastSales['Store_type'] == 'Control'].copy()
pastSales_Controls5['totSales'] = pastSales_Controls5['totSales'] * (1 - stdDev␣
↪* 2)
pastSales_Controls5['Store_type'] = 'Control 5th % confidence interval'
```

[670]:
```
trialAssessment = pd.concat([pastSales, pastSales_Controls95,␣
↪pastSales_Controls5])
```

[671]:
```
# Plotting these in one nice graph
trialAssessment['TransactionMonth'] = pd.
↪to_datetime(trialAssessment['TransactionMonth'], format='%Y-%m')
fig, ax = plt.subplots(figsize=(12, 6))
sns.lineplot(data=trialAssessment, x='TransactionMonth', y='totSales',␣
↪hue='Store_type', ax=ax)
```

```
ax.axvspan(pd.to_datetime('2019-02', format='%Y-%m'), pd.to_datetime('2019-04', format='%Y-%m'), alpha=0.2, color='grey')
ax.set(xlabel='Month of operation', ylabel='Total sales', title='Total sales by month')
ax.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))
plt.show()
```



[672]:
```
# 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
stdDev = percentageDiff[percentageDiff['MONTH_ID'] < '2019-02']['percentageDiff'].std()
degreesOfFreedom = 7
```

[673]:
```
# Trial and control store number of customers
pastCustomers = measureOverTimeCusts.copy()
pastCustomers['nCusts'] = pastCustomers.groupby(['MONTH_ID', 'Store_type'])['nCustomers'].transform('mean')
pastCustomers = pastCustomers[pastCustomers['Store_type'].isin(['Trial', 'Control'])]
```

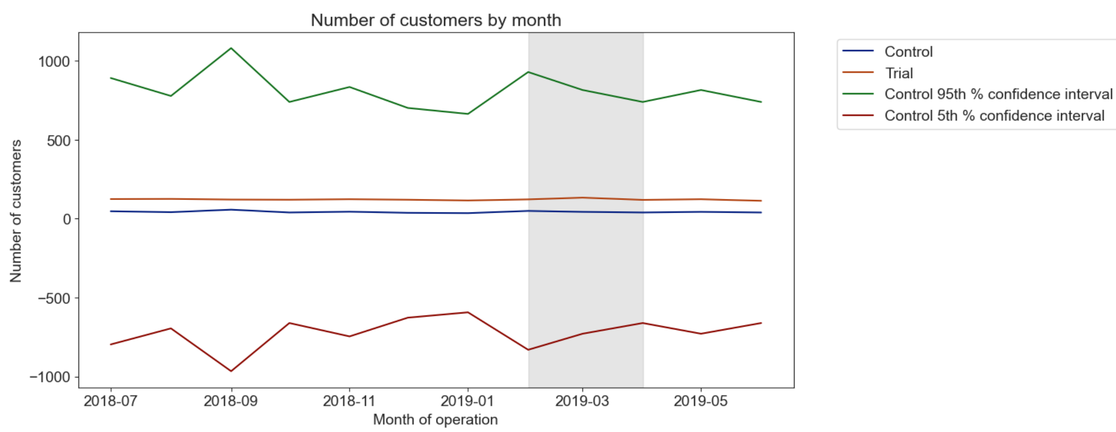
[674]:
```
# Control store 95th percentile
pastCustomers_Controls95 = pastCustomers[pastCustomers['Store_type'] == 'Control'].copy()
pastCustomers_Controls95['nCusts'] = pastCustomers_Controls95['nCusts'] * (1 + stdDev * 2)
pastCustomers_Controls95['Store_type'] = 'Control 95th % confidence interval'
```

[675]:
```
# Control store 5th percentile
pastCustomers_Controls5 = pastCustomers[pastCustomers['Store_type'] == 'Control'].copy()
```

```
pastCustomers_Controls5['nCusts'] = pastCustomers_Controls5['nCusts'] * (1 -␣
  ↪stdDev * 2)
pastCustomers_Controls5['Store_type'] = 'Control 5th % confidence interval'
```

[676]:
```
trialAssessment = pd.concat([pastCustomers, pastCustomers_Controls95,␣
  ↪pastCustomers_Controls5])
```

[677]:
```
# Plot everything into one nice graph
fig, ax = plt.subplots(figsize=(12, 6))
sns.lineplot(data=trialAssessment, x='TransactionMonth', y='nCusts',␣
  ↪hue='Store_type', ax=ax)
ax.axvspan(pd.to_datetime('2019-02', format='%Y-%m'), pd.to_datetime('2019-04',␣
  ↪format='%Y-%m'), alpha=0.2, color='grey')
ax.set(xlabel='Month of operation', ylabel='Number of customers', title='Number␣
  ↪of customers by month')
ax.legend(loc='upper left', bbox_to_anchor=(1.05, 1.0))
plt.show()
```



[678]:
```
# Calculate the metrics below as we did for the first trial store.
measureOverTime = data.groupby(['STORE_NBR', 'MONTH_ID']).agg(
    nCustomers=('LYLTY_CARD_NBR', 'nunique'),
    nTxn=('TXN_ID', 'nunique'),
    nChips=('PROD_QTY', 'sum'),
    totSales=('TOT_SALES', 'sum')
).reset_index()
measureOverTime['nTxnPerCust'] = measureOverTime['nTxn'] /␣
  ↪measureOverTime['nCustomers']
measureOverTime['nChipsPerTxn'] = measureOverTime['nChips'] /␣
  ↪measureOverTime['nTxn']
measureOverTime['avgPricePerUnit'] = measureOverTime['totSales'] /␣
  ↪measureOverTime['nChips']
```

```
measureOverTime = measureOverTime.sort_values(by=['STORE_NBR', 'MONTH_ID'])
```

[679]:
```
# Calculate the total number of customers in the trial period for the trial␣
↪store and control store
trial_customers = measureOverTime[(measureOverTime['STORE_NBR'] == trial_store)␣
↪& (measureOverTime['MONTH_ID'] >= '2019-02') & (measureOverTime['MONTH_ID']␣
↪<= '2019-04')]['nCustomers'].sum()
control_customers = measureOverTime[(measureOverTime['STORE_NBR'] ==␣
↪control_store) & (measureOverTime['MONTH_ID'] >= '2019-02') &␣
↪(measureOverTime['MONTH_ID'] <= '2019-04')]['nCustomers'].sum()

# Check if the number of customers in the trial period for the trial store is␣
↪significantly higher than the control store for two out of three months
significant_months = 0
for month in ['2019-02', '2019-03', '2019-04']:
    trial_customers_month = measureOverTime[(measureOverTime['STORE_NBR'] ==␣
↪trial_store) & (measureOverTime['MONTH_ID'] == month)]['nCustomers'].sum()
    control_customers_month = measureOverTime[(measureOverTime['STORE_NBR'] ==␣
↪control_store) & (measureOverTime['MONTH_ID'] == month)]['nCustomers'].sum()
    if trial_customers_month > control_customers_month:
        significant_months += 1

if significant_months >= 2:
    print("The total number of customers in the trial period for the trial␣
↪store is significantly higher than the control store for two out of three␣
↪months, which indicates a positive trial effect.")
else:
    print("The total number of customers in the trial period for the trial␣
↪store is not significantly higher than the control store for two out of␣
↪three months.")

# Conclusion
print("Good work! We've found control stores 233, 155, 237 for trial stores 77,␣
↪86 and 88 respectively.")
print("The results for trial stores 77 and 88 during the trial period show a␣
↪significant difference in at least two of the three trial months but this is␣
↪not the case for trial store 86. We can check with the client if the␣
↪implementation of the trial was different in trial store 86 but overall, the␣
↪trial shows a significant increase in sales. Now that we have finished our␣
↪analysis, we can prepare our presentation to the Category Manager.")
```

```
The total number of customers in the trial period for the trial store is
significantly higher than the control store for two out of three months, which
indicates a positive trial effect.
Good work! We've found control stores 233, 155, 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 but overall, the trial shows a significant increase in sales. Now that we have finished our analysis, we can prepare our presentation to the Category Manager.