submission

August 22, 2025

0.1 Full Repo

https://github.com/MostafaBelo/Konecta_Assignments/tree/main

0.2 Imports

[1]:
```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.cm as cm

import pandas as pd

from sklearn.base import BaseEstimator, ClusterMixin
from sklearn.cluster import KMeans, DBSCAN, MeanShift
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV,␣
 ↪PredefinedSplit
from sklearn.metrics import silhouette_score, davies_bouldin_score,␣
 ↪calinski_harabasz_score
```

0.3 Loading csv

[2]:
```
df = pd.read_csv("./Wholesale customers data.csv")

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Channel           440 non-null    int64
 1   Region            440 non-null    int64
 2   Fresh             440 non-null    int64
 3   Milk              440 non-null    int64
```

```
 4   Grocery           440 non-null    int64
 5   Frozen            440 non-null    int64
 6   Detergents_Paper  440 non-null    int64
 7   Delicassen        440 non-null    int64
dtypes: int64(8)
memory usage: 27.6 KB
```

[3]:
```
df.describe()
```

[3]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	1.322727	2.543182	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	1524.870455
std	0.468052	0.774272	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	2820.105937
min	1.000000	1.000000	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	1.000000	2.000000	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	1.000000	3.000000	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965.500000
75%	2.000000	3.000000	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	1820.250000
max	2.000000	3.000000	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

[4]:
```
df.head()
```

[4]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

[5]:
```
df.keys()
```

[5]:
```
Index(['Channel', 'Region', 'Fresh', 'Milk', 'Grocery', 'Frozen',
       'Detergents_Paper', 'Delicassen'],
      dtype='object')
```

[6]:
```
def get_uniques(df: pd.DataFrame, limit=-1):
    for key in df.keys():
        uqs = df[key].unique()
```

```
        print(key)
        if limit >=0 and len(uqs) > limit:
            print("Too Many Values")
        else:
            print(uqs)
        print()
get_uniques(df)
```

```
Channel
[2 1]

Region
[3 1 2]

Fresh
[ 12669   7057   6353  13265  22615   9413  12126   7579   5963   6006
   3366  13146  31714  21217  24653  10253   1020   5876  18601   7780
  17546   5567  31276  26373  22647  16165   9898  14276   4113  43088
  18815   2612  21632  29729   1502    688  29955  15168   4591  56159
  24025  19176  10850    630   9670   5181   3103  44466  11519   4967
   6269   3347  40721    491  27329   5264   4098   5417  13779   6137
   8590  35942   7823   9396   4760     85      9  19913   2446   8352
  16705  18291   4420  19899   8190  20398    717  12205  10766   1640
   7005    219  10362  20874  11867  16117  22925  43265   7864  24904
  11405  12754   9198  11314   5626      3     23    403    503   9658
  11594   1420   2932  56082  14100  15587   1454   8797   1531   1406
  11818  12579  19046  14438  18044  11134  11173   6990  20049   8258
  17160   4020  12212  11170  36050  76237  19219  21465    140  42312
   7149   2101  14903   9434   7388   6300   4625   3087  13537   5387
  17623  30379  37036  10405  18827  22039   7769   9203   5924  31812
  16225   1289  18840   3463    622   1989   3830  17773   2861    355
   1725  12434  15177   5531   5224  15615   4822   2926   5809   5414
    260    200    955    514    286   2343  45640  12759  11002   3157
  12356 112151    694  36847    327   8170   3009   2438   8040    834
  16936  13624   5509    180   7107  17023  30624   2427  11686   3067
   4484  25203    583   1956   1107   6373   2541   1537   5550  18567
  12119   7291   3317   2362   2806   2532     18   4155  14755   5396
   5041   2790   7274  12680  20782   4042   1869   8656  11072   2344
  25962    964  15603   1838   8635  18692   7363  47493  22096  24929
  18226  11210   6202   3062   8885  13569  15671   3191   6134   6623
  29526  10379  31614  11092   8475  56083  53205   9193   7858  23257
   2153   1073   5909    572  20893  11908  15218   4720   2083  36817
    894    680  27901   9061  11693  17360  12238  49063  25767  68951
  40254  15354  16260  42786   2708   6022   2838   3996  21273   7588
  19087   8090   6758    444  16448   5283   2886   2599    161    243
   6468  17327   6987    918   7034  29635   2137   9784  10617   1479
   7127   1182  11800   9759   1774   9155  15881  13360  25977  32717
   4414    542  16933   5113   9790  11223  22321   8565  16823  27082
```

```
 13970  9351  2617   381  2320   255  1689  3043  1198  2771
 27380  3428  5981  3521  1210   608   117 14039   190 22686
    37   759   796 19746  4734  2121  4627  2615  4692  9561
  3477 22335  6211 39679 20105  3884 15076  6338  5841  3136
 38793  3225  4048 28257 17770 34454  1821 10683 11635  1206
 20918  9785  9385  3352  2647   518 23632 12377  9602  4515
 11535 11442  9612  4446 27167 26539 25606 18073  6884 25066
  7362  8257  8708  6633  2126    97  4983  5969  7842  4389
  5065   660  8861  4456 17063 26400 17565 16980 11243 13134
 31012  3047  8607  3097  8533 21117  1982 16731 29703 39228
 14531 10290  2787]
```

```
Milk
[ 9656  9810  8808  1196  5410  8259  3199  4956  3648 11093  5403  1124
 12319  6208  9465  1114  8816  6157  6327  2495  4519   871  1917 36423
  9776  4230   961   803 20484  2100  3610  4339  1318  4786  1979  5491
  4362 10556 15729   555  4332  3065  7555 11095  7027 22044 14069 54259
  6152 21412  1095  4051  3916 10473  1449  3683 29892  9933  1970  5360
  3045 38369  6245 11601  1227 20959  1534  6759  7260  2820  2037  1266
  5139  5332  6343  1137  3587 12697  1175  3259   829  9540  9232  1563
  3327 46197 73498  5025   542  3836   596  2762 27472  3090 12220  2920
  2616   254   112  2182  7779 10810  6459  3504  2132  1014  6337 10646
  8397 16729  1648 11114  2770  2295  1080   793  2521  3880  1891  2344
  1200  3234   201 10769  1642  3473  1840  7243  8847   926  2428   589
  2032  1042  1882  1289  8579  8080  4257  4979  4280 13252  7152  1596
  3677  8384  1936  3373   584  1433  1825  3328  1371  9250    55 10690
  5291  1366  6570  7704  3651   540  2024 15726  7603 12653  6721  3195
   735   717  8675 25862  5479  7677  1208  7845  6958  7330  7075  4888
  6036 29627  8533 43950   918  6448   521  8002  7639 11577  6250   295
  1461  3485  1012  7209  7097  2154  2280 13240 14399 11487   685   891
 11711   780  4737  3748 12729  1895 28326  6602  6551 10765 16599  1475
  7504   367   899  7503  1115  2527   659  3243  5921  2204   577  2746
  5989 10678  1780  4984  2703  6380   820  3838   475  2567  3575  1801
  3576  7775  6154   346  5279  3795  1993 23133  1860  7961 17972   489
  5008  1931  4563  4959  4885  1110  1372  9679 23527  9763  1222  8053
   258  1032  5007  8323  1703  1610  3749  2317  6200  2884  7108  3965
  3613  4411   640  2247  2102   594   286  2160  3354  3086 11103  2013
  1897  1304  4560   879  6243 13316  5302  3688  7460 12939 12867  2374
  1020 20655  1492  2335  3737   925  1795 14982  1375  3088  2713 25071
  3696   713   944 16784  2209  1486  1786 14881  3216  4980   928  6817
  1511  1347   333  1188  4025  5763  5758  6964  1172  2602  6939  7184
  2380 14641  1099 10044  1106  6264  7393   727   134  1275 18664  5878
  2872   607  1601   997   873  6128  2217   894   337  3944  1887  3801
  6257  2256  1450  8630  3154  3294  5164  4591  7435  1364 21858   922
  3620  1916   848  1530  1181  2761  4180  6730   865  1316 11991  1666
   906  2801  4753 11006  4613  1046  5010 12844  3634  2096  3289  3605
  4859  1990  6046 10940  5499  8494  3783  5266  4847  1377  3686  2408
  9347 16687  5970  1750  5506  1162  3218  3922 12051  1431 15488  1981
```

```
  1698]

Grocery
[ 7561  9568  7684  4221  7198  5126  6975  9426  6192 18881 12974  4523
 11757 14982 12091  3821 12121  2933 10099  9464  4602  2010  4469 22019
 13792  7595  2861  3045 25957  2609 11107  3133  2886  7326  2262 11091
  5428 12477 16709   902  4757  5956 14961 23998 10471 21531 21955 55571
 10868 28921  1980  6996  5876 11532  1947  5005 26866 10487  1648  8040
  7854 59598  6544 15775  3250 45828  7417 13462  3993  1293  3202 21042
  2661  8713  9794     3  6532 28540  2067  3655  3009 14403 11009  1783
  4814 92780 32114  8117  4042  5330  1638  2530 32034  2062 11323  6252
  8118   610   778  1909 12144 16267  7677  8906  3445  3970 10704 14886
  6981 28986  1694 17569  2469  1733  2000  2988  3355  5380  2362  2147
  3412  1498   245  8814  2961  7102  1658 10685  3823  1510   699   314
  2479  1235  2174  2591  7030  8282  5034  3343  7305  5189  8253  1096
  1988 34792  2177  2707   542  1651  1765  2022  3135  2368   137 19460
 14855  2474  9618 14682 12822   283  3810 26870  8584 19858  9170  3268
   803  2155 13430 19816  6536 19805  5241 11874  4533  4945  2500  8887
 18148 10518 20170  4710  1139   854  9819 11687 11522  1981  1381  2251
 20292  2974  5230  4897 10391  6824  2112 23127 24708  9490  2216  5226
 23596   950  6089  5838 16767  1393 39694  6861 11364 15538 36486  2046
 15205  1390  1382 10646  2856  5265  1499  4157  9212  1563   572  2501
  5615  3828  3838  3316  3833  2824  3047   593   585  3779  7041  2475
  2914  5119 10817 13916  1777   489  2406  2070  1799 33586  4740 16966
  4748  1495  5249  1883  2124  7336  2157  1094  1677  6684 15445 13699
 22182  2576 19847  1138   975  6869  1493  1841   223  6964   683  2543
  9694  2431  6235  4252  2013 12609  3600  1242  2828  1296   471  2642
  3261  4329 12469  6550  5234  3643  6986  9965  2060  6360 20399  9785
 13829 24773  8852 21570  2842  3007 13567  2405  8280 19172  7647 11924
  2201  6114  3558 17645  2280  5167  3315 11593  2464 13626  1431  1664
  3389  4583  5109 26839  1447 67298  2743 10790  1330  2611  7021  5332
  9670 11238  5923 26316  1763  8335 15541 12311  2028 20521  1997 22294
  1533 21203  2548  2012   218 22272  1660  2109  2006   864  2453  4438
  1524  8025   534  4955  1939  1641  7398  1668  1162 13586  2648  1902
  2146  1617  8469  3450 15400  1614  2857  1573  1172  1422  1328  2313
  3842  3204  1263  9345  1428   582   935  1238  2128  5091  4604  3444
  1167  5026 18683  6407  6100  4563  3281 12400  6633  3417  8552 10908
 11055 18622  2223 13227  9053  4172  4657 12232  2593 14316  5429  4910
  3580 16483  5160  4754  7994 16027   764 30243  2232  2510]

Frozen
[  214  1762  2405  6404  3915   666   480  1669   425  1159  4400  1420
   287  3095   294   397   134   839  2205   669  1066  3383  9408  5154
  2915   201  3151   485  1158  1200  1148  2088   266  6130   833  1729
  1920    33 10002  9510  2033   188   787   541  1740  1668  7782   584
  1798  3860   239   532   744  2436  1057  2616    38   596   129    96
  3254  4154  2896  3724    36   175  1256  5870   779 10643  5373  8872
  8132  1285  4407  7530   869  2096   868   430   283   737  2320  1178
```

```
  1026   987  6312  9735  3443  3347  8693  3232 35009   206   440   145
   774   895  5639  3252  1593  2561 18028  1336   910   133  2471   247
   673  2276   805  8853  3220  2555  2715  1517  1647  5343  3896  2417
  2395  1991  2194  4787 16538  8195   880   142  1718  6316   346   576
   436   720  1170  4575   661   155   825  2279   321  2995  8425   118
    42   926  1286  4052   800   853   531  3001    75   233   317  3378
   930   398   824  1092  2665  2367  2540  4425   993   405  1393  2399
  1116   651   333   937  2515    52  7368  1752  1152  4477   402 16745
   443 36534    74  2181  3470  6269  2758   275  7332   890   547   959
   806  7888 18711  1127  3527   520  3941  3549  5065   469  1383   955
   878  2946  1859   864  1801  4736  1291  1329   913  1374   179  2532
  2306  1765    91  7496  5612   784   660  1759  2286   950  6845  8321
  1439   638  4260  1218  2312  4634  1112  5243 11422  2216  3752   561
  1183   230  1777  2077   559  6340  1730  6746  7683   432  4686  3242
   453  5004  6422  3012   327  6818   982  4324    61 10155  2221  3975
  1069  2516  5500  1120   529  4802   862  4479 16919  5845  1293   977
  1093  5970 10303  8692  1042  1619  8366   848  1388   502  2507  3838
   902   909   417  3045  1455   934   264  1809   364   492   617   799
  1840  1149   416  1465 12569  3046  1274  4447  1483   662  2679   978
  2121  1128   514  2714  3703   915  2369 60869  3498   414  7849  5127
  3570  1234  2208   131 11559  1365   650  8170 15601  9584   388   767
   349  1456  2234  2693  2809  1341  2005  1796  1741   830   228  6386
   245  3157   137  6114   340  2601  1206   560   191  1103  1173  1457
  2046  1089  1364  8164   876  1504  1492   597  5641  1034   282   130
  3881  9927  4006  3635  2583  1945  1960  1677  3019  5502   907   659
  8620  1398  2921  2644  6838  5390  1601  3576 13223   220   127  2069
  9806  2854  1646  2349  1389  1535    98 17866  5679  1691   633    25
  1031  1059   874 15348  3141 15082  2198    47   575 13486   269  1541
   688 13135  4510   437  1038    65]
```

Detergents_Paper

```
[ 2674  3293  3516   507  1777  1795  3140  3321  1716  7425  5977   549
  3881  6707  5058   964  4508   370  2767  2518  2259   375  2381  4337
  4482  4003   242   100  8604  1107  2134   820   918   361   483  4239
   862  6506  6956   212  1145  2575  6899  9529  4618  7353  6792 24171
  5121 13583   609  1538  2587  5611   204  2024 17740  7572   227  3084
  4095 26701  4074  7677  1247 24231  3468  5141   788   656   116  4173
  1321   764  1901     3   529 12034   301  1202   610  7818  3537   550
  3837 40827 20070  1579   165   454    69   627 18906    71  5038   223
  3874    54    56   215  8035  6766  4573  1480  1491   139  6830  8969
  2505   836   169  6457   585   118   276   310   319   411   266   174
   264    25  1976   500   778   349  2386  1062   410   395    70   955
   256    47   199  2447   721   249   637   960    51    20   399   516
 12591    73  1082   283   113   170   255   352   302     7 11577  6694
   811  4004  8077  4424   232 13726  3674  7108  4973  1680    79  7015
  8773  2840  9836   153  4196  1532   120   273  1382  4948  6907   239
   334    58   949  3459  6839  4027    43   187  5618   355   330   763
  4314   592   402  9959 14235   284   954     5  9265   288  5316  3381
```

```
 12420   244 19410   240  3961  5957  5828 13308   130  4797    86  4167
   761  2568   263  4762   694  1566   409   325  1216   415    28    72
   828   343   412   586  1682  3143  8933   430    44   562   234 18594
   205   363  1547   111   392  3593   730   967   780    49   429  2894
  5980   830  4882   737  6374   333   197   147    93   210   759    96
   603   621   274  3620   167  2328  1041   314   751   436  1226   386
   445    32   965   825  5952  2208   710  3712  4538   290  2662  8752
  6236 10069 11783  3909  7558   351   257  6846   299   371 17120   183
   857  3891    83   821   706 12408   275   228  1470  1679   140  1272
   387    88   492   182  9606   178 38102   332  4111   146   442    15
   573  7271  5162  4595 15469   217  3843  6600  4621  1184 12218   173
 12638    90  8682  1333   184     9  6747   536   468   159   179  1335
   514  4515   222   252   101    41   523   716   397  1916   311   476
  4666    68   813   600   246  1711   282   192   353   231   200    95
   122   385   149   841  3378    64    74   469    92    10   632   914
   593  1092  7883  2730  2123  1860   235  2970   912  1135  3540  6728
  3485  6740  1580  6818  3415   948  1803  3213   108  5079   439   850
    84   241  1377  1328   356  2371 14841   168   477]
```

Delicassen

```
[ 1338  1776  7844  1788  5185  1451   545  2566   750  2098  1744   497
  2931   602  2168   412  1080  4478  3181   501  2124   569  4334 16523
  5778    57   833   518  5206   823  2963   985   405  1083   395   436
  4626   714   433  2916  5864  2802    46    72    65  4985  1452  6465
  1476  1163  2162   301  1278   224  1333  1130  1340  1282  1603   225
  2017   964  1295  1145  1423    27   834  3095   144  1365 14472   181
   648  1780   975   894  1009   167  1653   529   156  2342   772   120
  2944   903 14351  3178   360  1117  5130  2698   244   709   217    63
   132   323  3029  1838  1386  2498   548  1378  1831  1438  1236     3
  1647  1519  2708  1561  1266   610   222  1160   933   635  1136   255
   860   143  1621   918   483  2749  1819   911   310   328   396   537
   326  1542    36  3271   929  2616  1450   318   201  4430   520   526
   434  1440  1067  1774   184  1627     8  2153  3182   418  1682   303
  2157  2233   446   238  2379  3637   693   429  6250   707   716  1442
  1697   230  2631  2165  2794  8550 47943    11   247   727   404  1856
    64    84   409   666  1142  1755  2876  1468   697   347   731  1681
  6854    18  1328   710   285   806   797  2100  2870  1775  1215   791
  2388   674  1158  6372   130   749   239   375  1360   659   786  1553
   689   203   980  2137   490     7  2563   295   216  2253  2564  1047
   578  2398  1970  2784   572   291  5121  1693  1391  3265   615   373
   987  3321   818   287   655   411  1265  3636  3628   698   204    56
  1550  1040  1824  1153   379  2503   139  1409  1721  1104  2079  1404
  1384  2406   128  1027   258    22  1522   686  1060   741  1854   254
   898   531  1037   259  2005   172   555    59  2410   211  1543   925
   656   117   142   297  1233  3508  1059  1637    51  1625  1113   229
   573  1092  5609   522  1534   739  1043  1102  2602  3486  2139   778
   868   550  1942  1371  2158    37  1115  1022   665   445   995  3137
   195  1111  2341   127   110  4100   776   503   712   314   468  3105
```

```
  447   342   558   296  2235   790  4829  3113    70  1426  1242  1114
  179   270   532  2893   361  5120  1068   967   961   406   684  1000
 1827   654   819   452   290  2213   743  1014  1902   340   288   715
  378   960   553   344  5137  1892  4365    62  2435  1874   993  1063
 1521  1393  1784  1218   668   249  1886  1894   317  2501  2080  1498
 1449   838  2204  2346  1867  2125    52]
```

0.4 Preprocessing

[7]:
```
# All values seem to be non-null
```

[8]:
```
# Outlier check

df_z = (df - df.mean())/df.std()

(df_z.abs() > 3).sum()
```

[8]:
```
Channel             0
Region              0
Fresh               7
Milk                9
Grocery             7
Frozen              6
Detergents_Paper    10
Delicassen          4
dtype: int64
```

[9]:
```
# Few outliers so we are safe to remove them

df.drop(df[(df_z.abs() > 3).any(axis=1)].index, inplace=True)

df.describe()
```

[9]:
```
          Channel      Region         Fresh          Milk       Grocery  \
count  414.000000  414.000000    414.000000    414.000000    414.000000
mean     1.314010    2.548309  10711.758454   4871.920290   6814.043478
std      0.464682    0.769475   9819.217756   4555.665546   6456.160715
min      1.000000    1.000000      3.000000     55.000000      3.000000
25%      1.000000    2.000000   3063.250000   1477.750000   2116.000000
50%      1.000000    3.000000   8040.000000   3530.000000   4528.000000
75%      2.000000    3.000000  15657.000000   6908.500000   9762.250000
max      2.000000    3.000000  49063.000000  25862.000000  34792.000000

             Frozen  Detergents_Paper   Delicassen
count    414.000000        414.000000   414.000000
mean    2549.898551       2373.393720  1237.939614
std     2916.683284       3208.707909  1217.399162
```

```
min       25.000000          3.000000     3.000000
25%      676.750000        252.750000   395.000000
50%     1447.000000        772.000000   881.000000
75%     3204.250000       3660.500000  1681.750000
max    16919.000000      17120.000000  7844.000000
```

0.4.1 Scaling Features

[10]:
```
df_scaled = (df - df.mean()) / df.std()

df_scaled.describe()
```

[10]:
```
            Channel        Region         Fresh          Milk       Grocery  \
count  4.140000e+02  4.140000e+02  4.140000e+02  4.140000e+02  4.140000e+02
mean  -1.716287e-16 -4.462346e-16 -4.719789e-17  1.029772e-16 -6.865147e-17
std    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
min   -6.757520e-01 -2.012163e+00 -1.090592e+00 -1.057347e+00 -1.054968e+00
25%   -6.757520e-01 -7.125758e-01 -7.789326e-01 -7.450438e-01 -7.276838e-01
50%   -6.757520e-01  5.870118e-01 -2.720948e-01 -2.945608e-01 -3.540871e-01
75%    1.476258e+00  5.870118e-01  5.036289e-01  4.470433e-01  4.566501e-01
max    1.476258e+00  5.870118e-01  3.905733e+00  4.607467e+00  4.333529e+00

             Frozen  Detergents_Paper    Delicassen
count  4.140000e+02      4.140000e+02  4.140000e+02
mean   4.290717e-17     -1.716287e-17 -9.439577e-17
std    1.000000e+00      1.000000e+00  1.000000e+00
min   -8.656746e-01     -7.387378e-01 -1.014408e+00
25%   -6.422187e-01     -6.609027e-01 -6.924102e-01
50%   -3.781345e-01     -4.990774e-01 -2.931985e-01
75%    2.243478e-01      4.011292e-01  3.645562e-01
max    4.926521e+00      4.595808e+00  5.426372e+00
```

[11]:
```
X = df_scaled
```

0.5 Cluster Training and evaluation

[12]:
```
def search(X, params, model_class, method="grid", metric="silhouette",␣
 ↪n_iter=30):
    metric_func = {
        "silhouette": silhouette_score,
        "dbi": davies_bouldin_score,
        "chi": calinski_harabasz_score,
    }[metric]

    class WrapperModel(BaseEstimator, ClusterMixin):
        def __init__(self, **kwargs):
            #  Store kwargs so sklearn knows about them
```

```
            self.kwargs = kwargs

        def set_params(self, **params):
            self.kwargs.update(params)
            return self

        def get_params(self, deep=True):
            return self.kwargs.copy()

        def fit(self, X, y=None):
            self.model_ = model_class(**self.kwargs)
            self.labels_ = self.model_.fit_predict(X)
            return self

        def score(self, X, y=None):
            if len(set(self.labels_)) <= 1:
                return -1000 if metric in ["silhouette", "chi"] else 1000
            score = metric_func(X, self.labels_)
            return -score if metric == "dbi" else score

    dummy_cv = [(np.arange(len(X)), np.arange(len(X)))]

    if method == "grid":
        searcher = GridSearchCV(
            estimator=WrapperModel(),
            param_grid=params,
            verbose=1,
            cv=dummy_cv,
            n_jobs=-1,
        )
    else:
        searcher = RandomizedSearchCV(
            estimator=WrapperModel(),
            param_distributions=params,
            verbose=1,
            cv=dummy_cv,
            n_iter=n_iter,
            n_jobs=-1,
        )

    searcher.fit(X)

    print("Best params:", searcher.best_params_)
    print(f"Best {metric} score:", searcher.best_score_)
    print()
    return searcher.best_estimator_, searcher.best_params_, searcher.best_score_
```

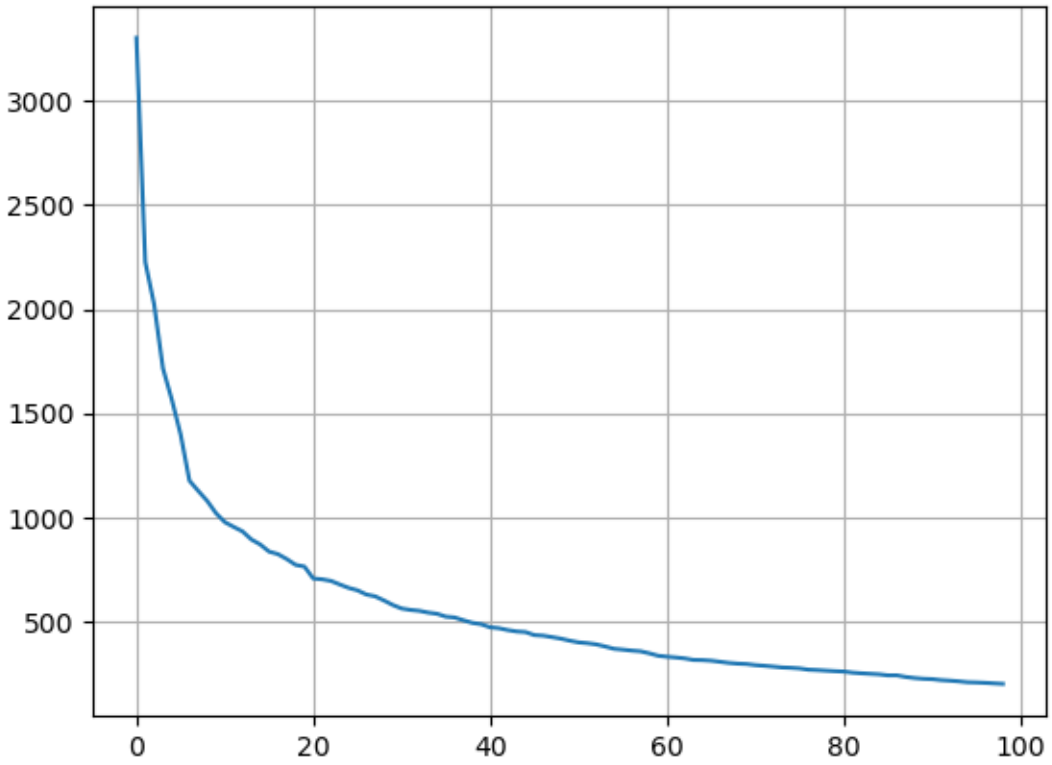
[13]:
```
models = {}

def eval(Name, model, X, labels):
    score_sh = silhouette_score(X, labels)
    score_db = davies_bouldin_score(X, labels)
    score_ch = calinski_harabasz_score(X, labels)

    print("Clusters Count:", len(set(labels)))

    print("Silhouette Score:", score_sh)
    print("Davies-Bouldin Index:", score_db)
    print("Calinski-Harabasz Index:", score_ch)

    models[Name] = {
        "Name": Name,
        "Model": model,
        "labels": labels,
        "metrics": {
            "score_sh": score_sh,
            "score_db": score_db,
            "score_ch": score_ch,
        }
    }
```

0.5.1 KMeans

[14]:
```
params = {
    "n_clusters": list(range(2, 100)),
    "random_state": [42],
}

search(X, params, KMeans, method="grid", metric="silhouette");
search(X, params, KMeans, method="grid", metric="dbi");
search(X, params, KMeans, method="grid", metric="chi");
```

```
Fitting 1 folds for each of 98 candidates, totalling 98 fits
Best params: {'n_clusters': 2, 'random_state': 42}
Best silhouette score: 0.35102559061514965

Fitting 1 folds for each of 98 candidates, totalling 98 fits
Best params: {'n_clusters': 99, 'random_state': 42}
Best dbi score: -0.7900541120322981

Fitting 1 folds for each of 98 candidates, totalling 98 fits
Best params: {'n_clusters': 2, 'random_state': 42}
Best chi score: 199.47723211994588
```

[15]:
```
intertias = []

for k in range(1,100):
    kmean = KMeans(n_clusters=k, random_state=42)
    kmean.fit(X)
    # labels = kmean.labels_
    # centers = kmean.cluster_centers_
    intertias.append(kmean.inertia_)

plt.plot(intertias);
plt.grid();

# 20 is a good elbow
```



[16]:
```
kmean = KMeans(n_clusters=20, random_state=42)
kmean.fit(X)
labels = kmean.labels_

eval("KMeans", kmean, X, labels)
```

```
Clusters Count: 20
Silhouette Score: 0.22442835965126814
```

```
Davies-Bouldin Index: 1.445003682140416
Calinski-Harabasz Index: 69.1979890629256
```

0.5.2 Hierarchical

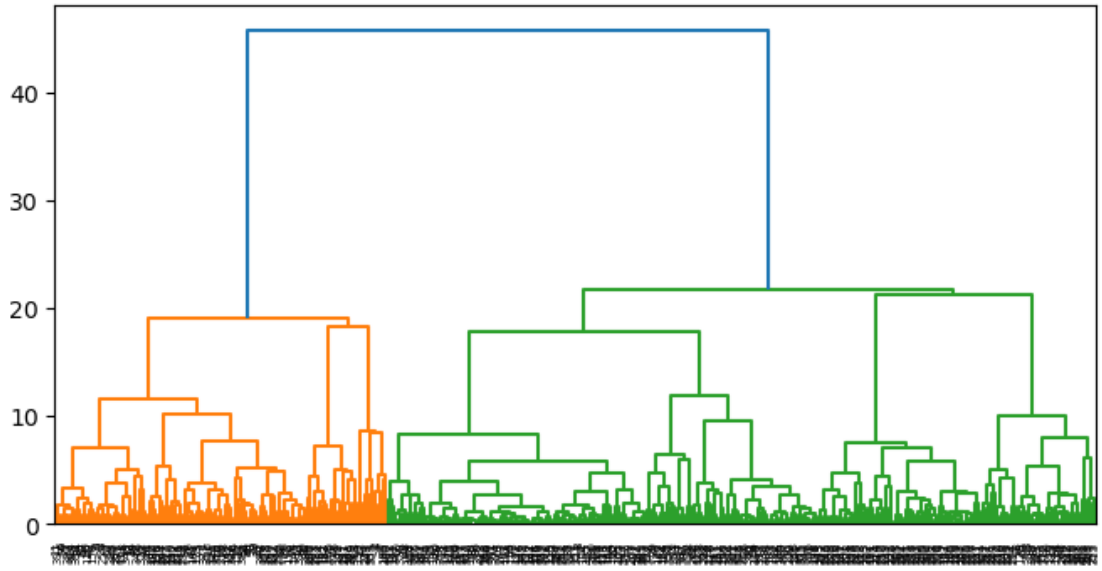
[17]:
```python
class HierarchicalWrapper:
    def __init__(self, t=2, method="ward"):
        self.t = t
        self.method = method
        self._Z = None   # will hold the linkage matrix
        self._X = None   # keep X if needed

    def fit(self, X, y=None):
        self._X = X
        self._Z = linkage(X, method=self.method)
        return self

    @property
    def labels_(self):
        if self._Z is None:
            raise ValueError("Model not fitted yet. Call .fit(X) first.")
        return fcluster(self._Z, t=self.t, criterion="maxclust")

    def fit_predict(self, X, y=None):
        self.fit(X, y)
        return self.labels_

params = {
    "t": list(range(2, 10)),
    "method": ["ward", "complete", "average", "single"]
}

search(X, params, HierarchicalWrapper, method="grid", metric="silhouette");
search(X, params, HierarchicalWrapper, method="grid", metric="dbi");
search(X, params, HierarchicalWrapper, method="grid", metric="chi");
```

```
Fitting 1 folds for each of 32 candidates, totalling 32 fits
Best params: {'method': 'single', 't': 2}
Best silhouette score: 0.39825637913430123

Fitting 1 folds for each of 32 candidates, totalling 32 fits
Best params: {'method': 'single', 't': 2}
Best dbi score: -0.4651900503730936

Fitting 1 folds for each of 32 candidates, totalling 32 fits
Best params: {'method': 'ward', 't': 2}
Best chi score: 191.34688484634316
```

[18]:
```
# Perform hierarchical clustering (Ward's method)
Z = linkage(X, method='ward')

clusters = fcluster(Z, t=2, criterion='maxclust')

eval("Hierarchical", Z, X, clusters)
```

```
Clusters Count: 2
Silhouette Score: 0.3464660865788527
Davies-Bouldin Index: 1.2868043821600255
Calinski-Harabasz Index: 191.34688484634316
```

[19]:
```
# Plot dendrogram
plt.figure(figsize=(8, 4))
dendrogram(Z);
```



0.5.3 DBSCAN

[20]:
```
params = {
    "eps": np.linspace(0.1, 2.0, 20),
    "min_samples": list(range(1,10)),
}

best_model, best_params, best_score = search(X, params, DBSCAN, method="grid",␣
 ↪metric="silhouette")
```

```
Fitting 1 folds for each of 180 candidates, totalling 180 fits
Best params: {'eps': np.float64(2.0), 'min_samples': 4}
```

```
Best silhouette score: 0.33160555947846404
```

[21]:
```
db = DBSCAN(eps=2, min_samples=4)
db.fit(X)
labels = db.labels_

eval("DBSCAN", db, X, labels)
```

```
Clusters Count: 3
Silhouette Score: 0.33160555947846404
Davies-Bouldin Index: 2.1360379799360394
Calinski-Harabasz Index: 108.90110333762323
```

0.5.4 MeanShift

[22]:
```
ms = MeanShift(bandwidth=None)
ms.fit(X)
labels = ms.predict(X)

eval("MeanShift", ms, X, labels)
```

```
Clusters Count: 5
Silhouette Score: 0.28536086529846033
Davies-Bouldin Index: 1.5009312018373087
Calinski-Harabasz Index: 33.7644296397798
```

0.6 Evaluation

[23]:
```
comparison = {
    k: v["metrics"] for k, v in models.items()
}

comparison = pd.DataFrame.from_dict(comparison, orient="index")

comparison_sh = comparison.sort_values(by="score_sh", ascending=False)
comparison_db = comparison.sort_values(by="score_db", ascending=True)
comparison_ch = comparison.sort_values(by="score_ch", ascending=False)
```

[24]:
```
comparison_sh
```

[24]:

	score_sh	score_db	score_ch
Hierarchical	0.346466	1.286804	191.346885
DBSCAN	0.331606	2.136038	108.901103
MeanShift	0.285361	1.500931	33.764430
KMeans	0.224428	1.445004	69.197989

[25]:
```
comparison_db
```

[25]:

	score_sh	score_db	score_ch
Hierarchical	0.346466	1.286804	191.346885
KMeans	0.224428	1.445004	69.197989
MeanShift	0.285361	1.500931	33.764430
DBSCAN	0.331606	2.136038	108.901103

[26]:
```
comparison_ch
```

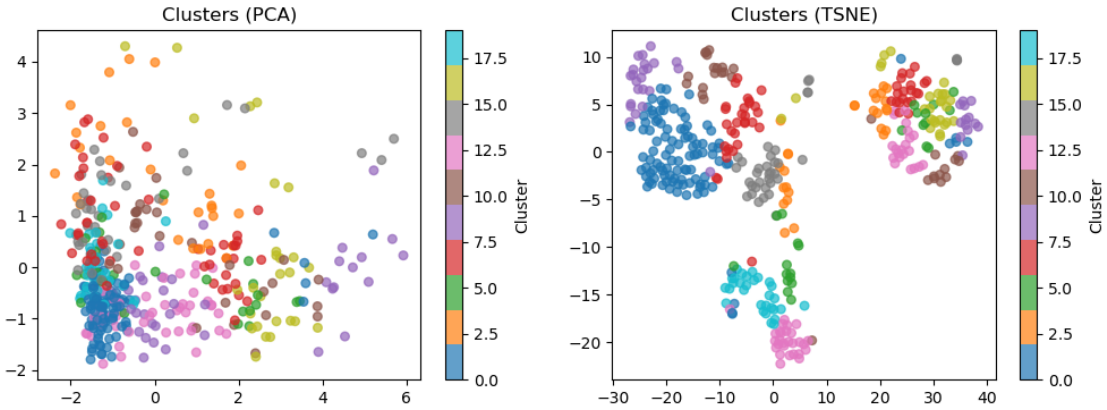
[26]:

	score_sh	score_db	score_ch
Hierarchical	0.346466	1.286804	191.346885
DBSCAN	0.331606	2.136038	108.901103
KMeans	0.224428	1.445004	69.197989
MeanShift	0.285361	1.500931	33.764430

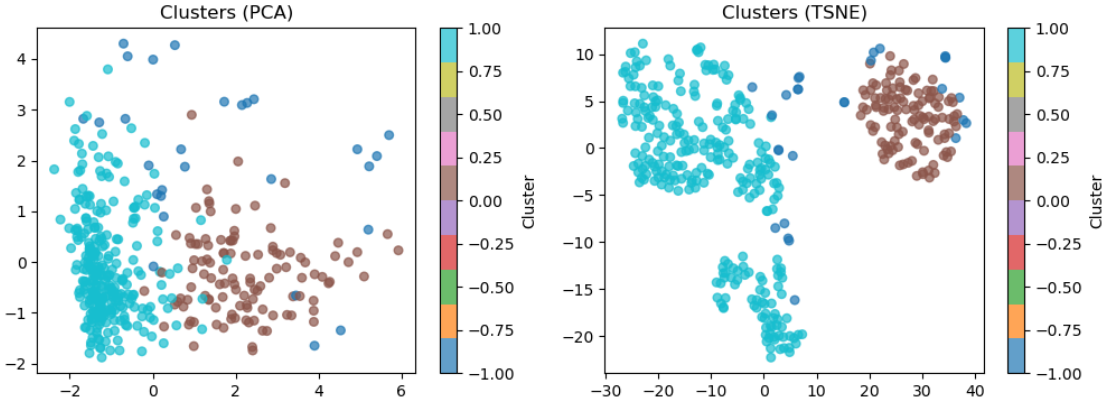
0.7 Visualization

[27]:
```
def plot_clusters(X, labels, method=None):
    assert method in ["pca", "tsne", None]

    if method is None:
        plt.figure(figsize=(12,4))
        plt.subplot(1,2,1)
        plot_clusters(X, labels, "pca")
        plt.subplot(1,2,2)
        plot_clusters(X, labels, "tsne")

        return

    if method == "pca":
        model = PCA(n_components=2)
    elif method == "tsne":
        model = TSNE(n_components=2, perplexity=30, random_state=42)
    X_2d = model.fit_transform(X)

    scatter = plt.scatter(X_2d[:,0], X_2d[:,1], c=labels, cmap="tab10", s=30,␣
↪alpha=0.7)
    plt.colorbar(scatter, label="Cluster")

    plt.title(f"Clusters ({method.upper()})")
```

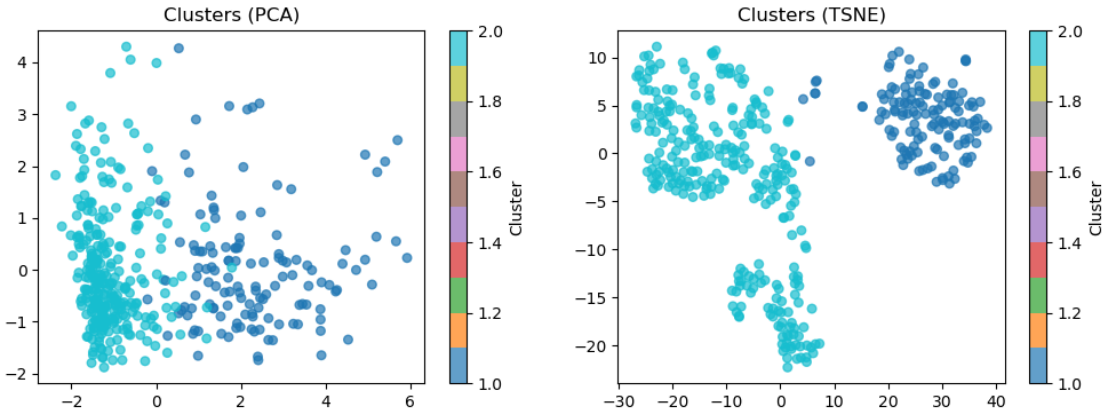
[28]:
```
plot_clusters(X, models["KMeans"]["labels"])
```



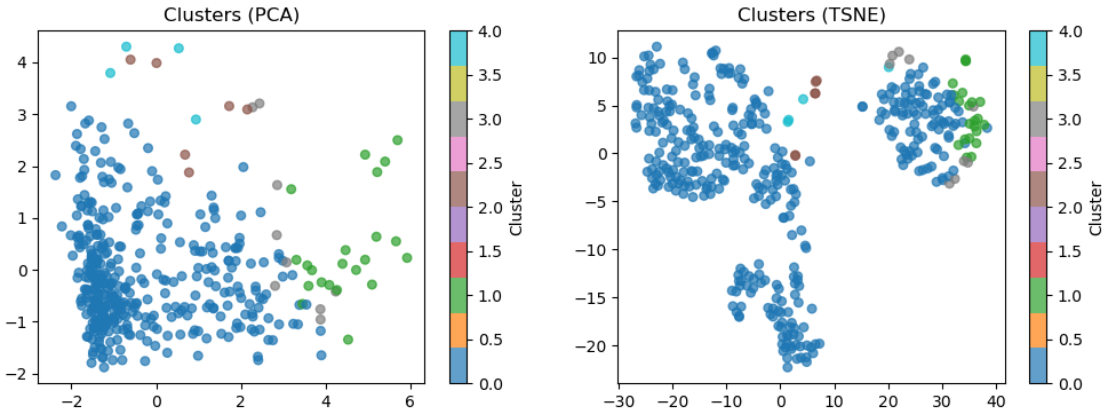
[29]:
```
plot_clusters(X, models["DBSCAN"]["labels"])
```



[30]:
```
plot_clusters(X, models["Hierarchical"]["labels"])
```



[31]:
```
plot_clusters(X, models["MeanShift"]["labels"])
```



0.8 Interpretation

KMeans using the elbow method showed by large cluster count of 20, unlike the rest which when optimizing the relevant scores opted for 2, 3, and 5 cluster counts.

When looking at the visuals, TSNE seems to be doing a better job at visualizing a good seperation of axes unlike PCA.

[32]:
```
def cluster_summary(X:pd.DataFrame, labels, csv_path=None):
    df = X.copy()
    df["cluster"] = labels

    # group by cluster and compute stats
    summary = df.groupby("cluster").agg(["mean", "min", "max", "count"])

    if csv_path is not None:
        summary.to_csv(csv_path, index=True)

    return summary
```

[33]:
```
cluster_summary(df, models["KMeans"]["labels"], "kmeans_summary.csv")

# The summary is hard to interpret due to the many clusters
# Some of these clusters have <10 data points, indicating a high degree of␣
 ↪overfitting the data without producing usefull analysis information
```

[33]:
```
           Channel                    Region                          Fresh         \
              mean min max count         mean min max count            mean    min
cluster
```

0	1.000000	1	1	99	2.959596	2	3	99	6829.202020	3
1	2.000000	2	2	5	2.000000	2	2	5	4781.400000	918
2	2.000000	2	2	15	2.866667	2	3	15	23727.866667	15076
3	1.083333	1	2	12	2.583333	1	3	12	18309.833333	3
4	2.000000	2	2	11	2.818182	2	3	11	4448.545455	243
5	1.153846	1	2	13	1.230769	1	2	13	6346.846154	2153
6	1.000000	1	1	30	2.900000	1	3	30	32196.966667	19746
7	2.000000	2	2	21	3.000000	3	3	21	5464.666667	381
8	2.000000	2	2	13	2.615385	1	3	13	6780.538462	161
9	1.000000	1	1	23	3.000000	3	3	23	4434.086957	9
10	1.923077	1	2	13	1.076923	1	2	13	4604.307692	572
11	1.000000	1	1	17	3.000000	3	3	17	12156.058824	2126
12	1.000000	1	1	28	1.035714	1	2	28	5610.785714	514
13	2.000000	2	2	21	2.809524	2	3	21	8624.142857	23
14	1.428571	1	2	7	2.428571	1	3	7	5317.714286	200
15	1.034483	1	2	29	2.965517	2	3	29	14154.137931	717
16	2.000000	2	2	21	3.000000	3	3	21	4491.809524	37
17	1.666667	1	2	6	2.333333	1	3	6	16810.166667	18
18	1.000000	1	1	5	1.800000	1	2	5	15191.600000	1182
19	1.000000	1	1	25	1.440000	1	2	25	17329.360000	7127

	…	Frozen		Detergents_Paper				Delicassen	\
	…	max	count	mean	min	max	count	mean	
cluster	…								
0	…	4052	99	420.646465	7	1860	99	676.101010	
1	…	1840	5	7891.000000	3891	12408	5	1530.800000	
2	…	3095	15	3443.600000	523	6707	15	1992.400000	
3	…	16919	12	541.916667	15	2381	12	1852.250000	
4	…	2194	11	4731.545455	836	6956	11	603.818182	
5	…	9584	13	927.384615	49	3593	13	932.769231	
6	…	7368	30	517.133333	9	1711	30	1022.733333	
7	…	5641	21	4654.238095	1901	8035	21	2119.571429	
8	…	3549	13	13186.230769	9606	17120	13	1540.153846	
9	…	6269	23	2715.565217	120	6907	23	785.173913	
10	…	3941	13	7066.538462	2568	12420	13	1225.153846	
11	…	5870	17	769.235294	3	2575	17	3282.882353	
12	…	2312	28	1098.428571	5	4762	28	725.642857	
13	…	4154	21	3253.809524	549	6236	21	533.619048	
14	…	10155	7	3989.285714	282	8773	7	4651.714286	
15	…	10643	29	478.344828	3	1333	29	928.137931	
16	…	4425	21	7478.523810	5618	9836	21	935.809524	
17	…	9510	6	2666.833333	284	4797	6	6316.166667	
18	…	1483	5	906.200000	363	1679	5	1337.600000	
19	…	4634	25	347.520000	28	1470	25	1071.560000	

	min	max	count

```
cluster
0           8  2162    99
1         172  3508     5
2         602  3181    15
3         139  4334    12
4           3  1856    11
5          56  2137    13
6           3  2893    30
7        1371  3637    21
8          37  4430    13
9           3  2080    23
10         59  2784    13
11       2125  5137    17
12          7  2398    28
13         57  1451    21
14       3265  6250     7
15         46  2708    29
16         46  2379    21
17       5185  7844     6
18        573  1854     5
19         51  3628    25

[20 rows x 32 columns]
```

[34]:
```
cluster_summary(df, models["Hierarchical"]["labels"], "hierarchical_summary.
 ↪csv")

# The summary shows two groups
# Group A spend more on categories: Milk, Grocery, Frozen, Delicassen
# Group B spend more on categories: Fresh, Detergents_Paper
```

[34]:
```
           Channel                   Region                          Fresh      …  \
              mean min max count      mean min max count           mean min  …
cluster                                                                      …
1         1.962121   1   2   132  2.628788   1   3   132   8319.00000  18  …
2         1.010638   1   2   282  2.510638   1   3   282  11831.77305   3  …

         Frozen       Detergents_Paper                    Delicassen      \
            max count             mean  min    max count         mean min
cluster
1         10155   132      5812.704545  282  17120   132  1654.204545   3
2         16919   282       763.503546    3   6907   282  1043.092199   3


          max count
cluster
1         7844   132
```

```
2        5864   282

[2 rows x 32 columns]
```

[35]:
```
cluster_summary(df, models["DBSCAN"]["labels"], "dbscan_summary.csv")

# The summary shows three groups
# Group A are few in count, but are considered overspenders as they typically␣
 ↪spend more than the other two in most categories
# Then Groups B and C which are somewhat comparable and mainly differ in their␣
 ↪channel
```

[35]:
```
           Channel                    Region                          Fresh      \
              mean min max count      mean min max count           mean min
cluster
-1        1.607143   1   2    28  2.250000   1   3    28  15832.642857  18
 0        2.000000   2   2   113  2.654867   1   3   113   7693.221239  23
 1        1.000000   1   1   273  2.534799   1   3   273  11435.970696   3

          … Frozen       Detergents_Paper                     Delicassen      \
          …    max count             mean  min    max count         mean min
cluster   …
-1        …  15082    28      3881.321429  182  17120    28  3293.928571   3
 0        …   5641   113      5875.884956  549  14841   113  1314.575221  46
 1        …  16919   273       768.985348    3   6907   273   995.347985   3


          max count
cluster
-1       7844    28
 0       5185   113
 1       5137   273

[3 rows x 32 columns]
```

[36]:
```
cluster_summary(df, models["MeanShift"]["labels"], "meanshift_summary.csv")

# The summary shows five groups
# Each group specializes in purchasing a certain category except for Group A␣
 ↪which are low spenders in general

# Group B specializes in Detergents_Paper
# Group C specializes in Milk and Frozen
# Group D specializes in Grocery
# Group E specializes in Fresh
```

[36]:

	Channel				Region				Fresh	
	mean	min	max	count	mean	min	max	count	mean	min
cluster										
0	1.262735	1	2	373	2.568365	1	3	373	10885.978552	3
1	2.000000	2	2	22	2.636364	1	3	22	6440.227273	200
2	1.000000	1	1	6	2.333333	1	3	6	14740.833333	759
3	2.000000	2	2	9	1.666667	1	3	9	4549.888889	18
4	1.250000	1	2	4	2.500000	1	3	4	25779.750000	22615

	…	Frozen		Detergents_Paper				Delicassen	
	…	max	count	mean	min	max	count	mean	
cluster	…								
0	…	16919	373	1831.187668	3	17120	373	1054.873995	
1	…	4425	22	10280.909091	836	15469	22	2024.909091	
2	…	15082	6	636.000000	182	1547	6	3248.000000	
3	…	2915	9	7107.666667	3516	12420	9	3640.666667	
4	…	9510	4	1396.750000	284	2381	4	5559.250000	

	min	max	count
cluster			
0	3	5137	373
1	3	6250	22
2	1163	5120	6
3	710	7844	9
4	4334	6854	4

[5 rows x 32 columns]