Creating Customer Segments

Edderic Ugaddan

In this project you, will analyze a dataset containing annual spending amounts for internal structure, to understand the variation in the different types of customers that a wholesale distributor interacts with.

Instructions:

- Run each code block below by pressing **Shift+Enter**, making sure to implement any steps marked with a TODO.
- Answer each question in the space provided by editing the blocks labeled "Answer:".
- When you are done, submit the completed notebook (.ipynb) with all code blocks executed, as well as a .pdf version (File > Download as).

In [22]:
```
# Import libraries: NumPy, pandas, matplotlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.externals import joblib

# Tell iPython to include plots inline in the notebook
%matplotlib inline

# Read dataset
data = pd.read_csv("wholesale-customers.csv")
print "Dataset has {} rows, {} columns".format(*data.shape)
print data.head()  # print the first 5 rows
```

```
Dataset has 440 rows, 6 columns
   Fresh  Milk  Grocery  Frozen  Detergents_Paper  Delicatessen
0  12669  9656     7561     214              2674          1338
1   7057  9810     9568    1762              3293          1776
2   6353  8808     7684    2405              3516          7844
3  13265  1196     4221    6404               507          1788
4  22615  5410     7198    3915              1777          5185
```

Exploratory Data Analysis

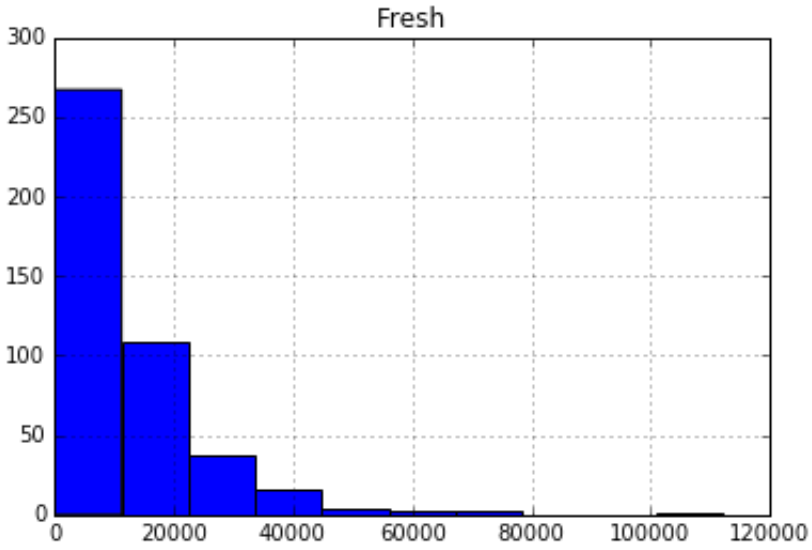
It seems that Fresh and Grocery products have the biggest variances.

Visual mode

There also seems to be two clusters...

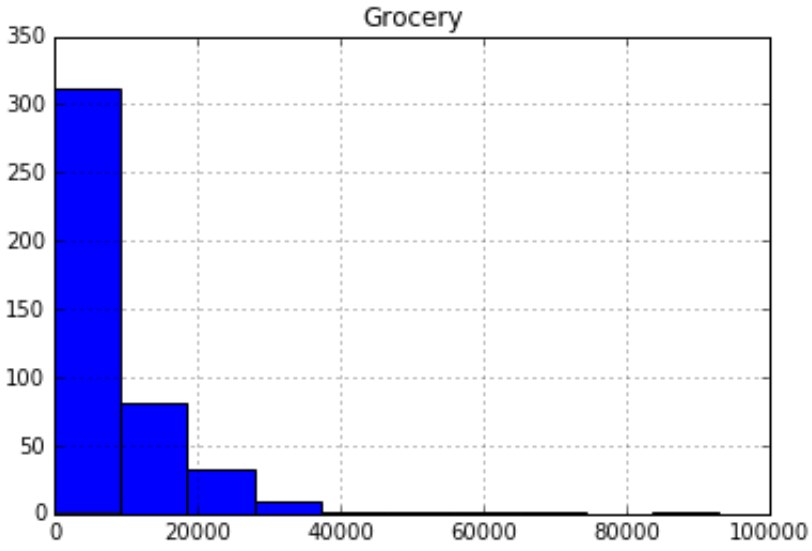
In [23]:
```
data[['Fresh']].hist()
```

Out[23]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x112cae690>]], dtype=object)
```



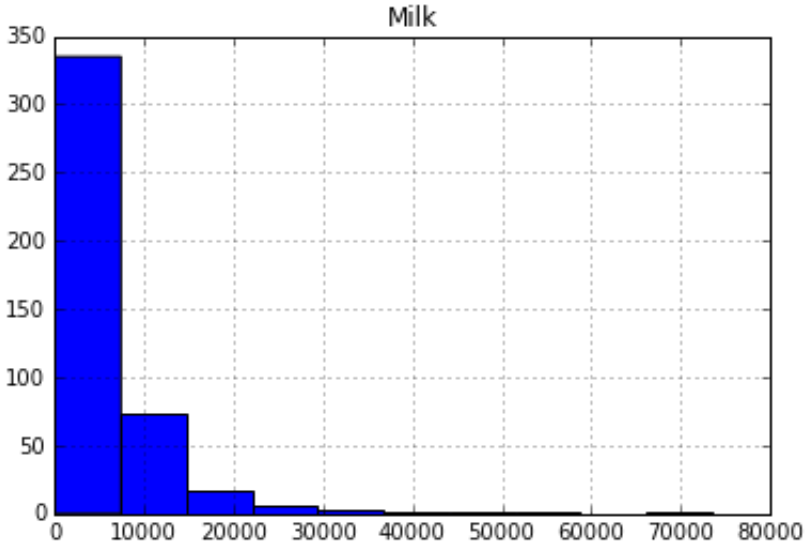
In [24]:
```
data[['Grocery']].hist()
```

Out[24]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x111f7ac50>]], dtype=object)
```



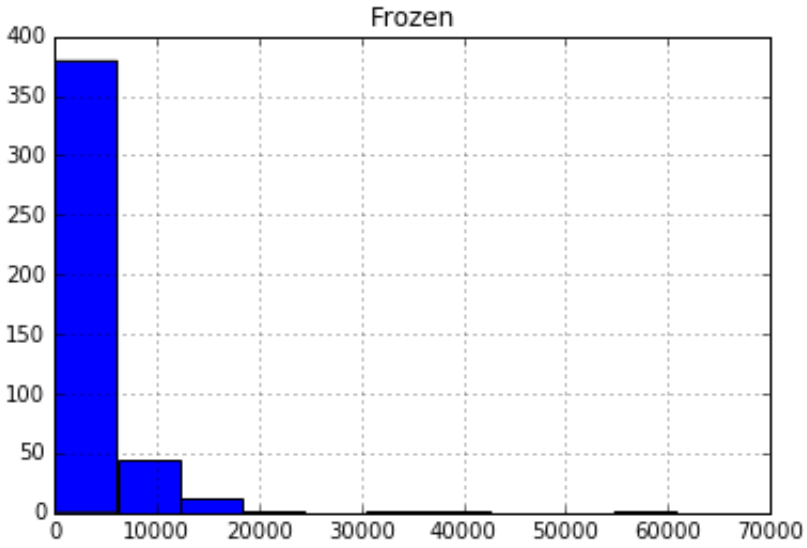
In [25]:
```
data[['Milk']].hist()
```

Out[25]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x11208f15
0>]], dtype=object)
```



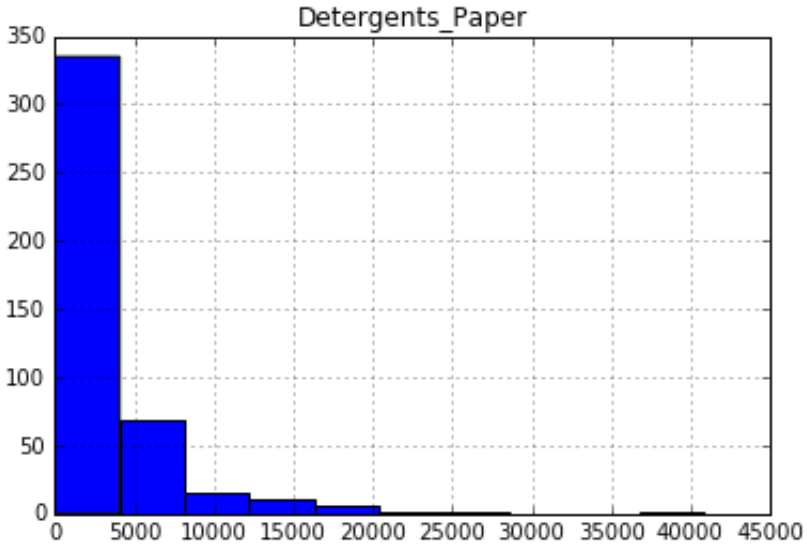
In [26]:
```
data[['Frozen']].hist()
```

Out[26]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x11317f7d
0>]], dtype=object)
```



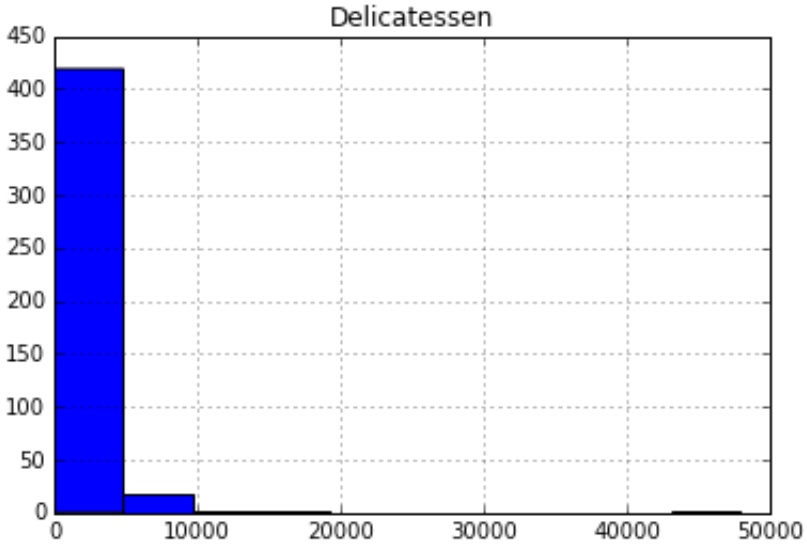
In [27]:
```
data[['Detergents_Paper']].hist()
```

Out[27]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x11346031
0>]], dtype=object)
```



In [28]:
```
data[['Delicatessen']].hist()
```

Out[28]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x11349bd5
0>]], dtype=object)
```



Feature Transformation

1) In this section you will be using PCA and ICA to start to understand the structure of the data. Before doing any computations, what do you think will show up in your computations? List one or two ideas for what might show up as the first PCA dimensions, or what type of vectors will show up as ICA dimensions.

Answer:

Answer:

Right now, I have a dataset of 440 rows and 6 columns (Fresh, Milk, Grocery, etc.) When I perform PCA, I expect the dataset to be transformed into another 440x6 matrix, where each customer's wholesale delivery is no longer represented explicitly in terms of the original columns (Fresh, Milk, Grocery, etc.) but as Principal Components (i.e. components that explain the variances in the data the best). Judging by the variances in the histograms of the products, I expect the first two principal components to correspond well to "Fresh" and "Grocery" products. In the histograms above, there seems to be a trend where order numbers for each category start really high, and then exponentially drop down to having no orders, and then followed again by some orders. This "gulf" suggests two clusters -- one for "small" businesses and another for "large" businesses.

On the other hand, with ICA, I guess we will get six statistically independent components (i.e. knowing about one component does not tell us anything about the other component). ICA is typically used for Blind Source Separation, where we are trying to extract independent signals from their mixtures. I suppose we might be able to represent products such as "Milk" and "Frozen" products by a vector that represents "refrigerated." Similarly, ICA might represent "Grocery" and "Detergents/Paper" products into a "dry" category.

PCA

In [124]:
```
# TODO: Apply PCA with the same number of dimensions as variables in t
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(data)

# Print the components and the amount of variance in the data containe
print pca.components_
print pca.explained_variance_ratio_
```

```
[[-0.97653685 -0.12118407 -0.06154039 -0.15236462  0.00705417 -0.068
10471]
 [-0.11061386  0.51580216  0.76460638 -0.01872345  0.36535076  0.057
07921]]
[ 0.45961362  0.40517227]
```

2) How quickly does the variance drop off by dimension? If you were to use PCA on this dataset, how many dimensions would you choose for your analysis? Why?

Answer:

Variance drops off relatively slowly from the 1st principal component to the 2nd (~5%). However, variance drops significantly from the 2nd principal component to the 3rd. I would just use the first two principal components since they account for 0.45961362 + 0.40517227 = 0.86478589, which is quite a significant amount of the variation. I would choose 2 principal components because of the clear "gap" between clusters, as seen in the histograms above.

3) What do the dimensions seem to represent? How can you use this information?

Answer:

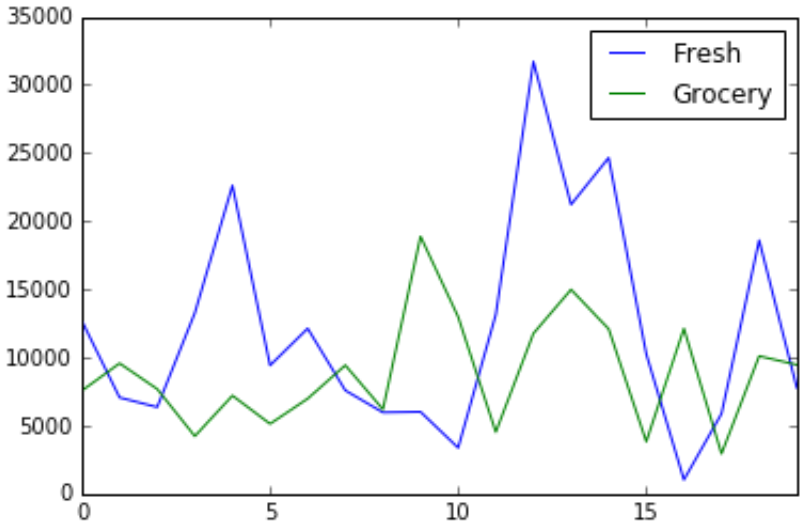
The first two dimensions seem to correspond strongly with annual customer spending on "fresh" and "grocery" items. By comparing the original graphs to the graphs of PCA for the first and last 20 customers, it seems that the first principal component is negatively correlated with annual customer spending on freshness (see below). We observe that peaks in "fresh" annual spending per customer correspond to troughs in terms of the first principal component, and similarly, the troughs in "fresh" annual spending per correspond to peaks in the latter. On the other hand, the second principal component seems to follow the general trend of annual spending on "grocery" items.

This information makes sense. When we look at README, we find out that annual spending on fresh items has the biggest standard deviation (12,647.239) and that annual spending per customer on grocery items has the second biggest standard deviation (9,503.163). PCA finds vectors to project onto so that variances of the data on them are maximized. Standard deviation is just the square root of the variance, so it follows that annual spending on "fresh" and "grocery" products are intimately related to the first and second principal components.

What this tells me is that if I really want the business to pay attention to only a couple of types of products, I would recommend them to focus on products that belong to the "fresh" and "grocery" labels. Together, they seem to capture much of the information about the whole dataset.

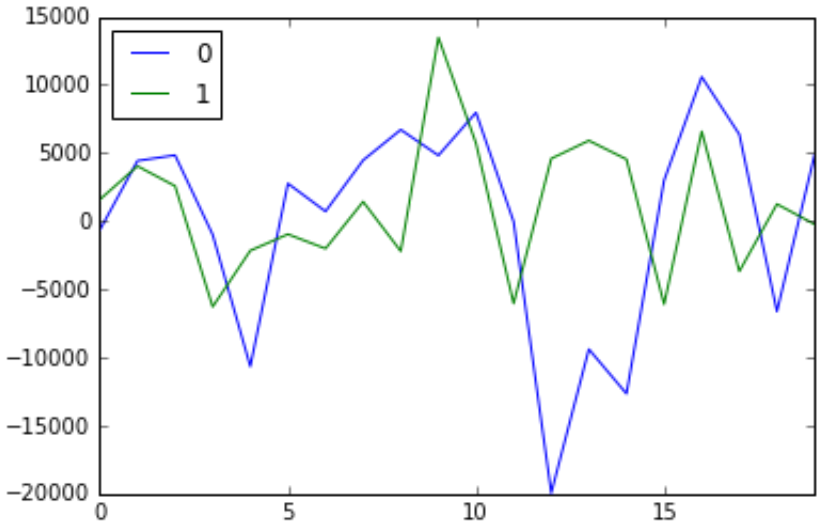
In [3]:
```
data[['Fresh', 'Grocery']].head(20).plot()
```

Out[3]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1108cc7d0>
```



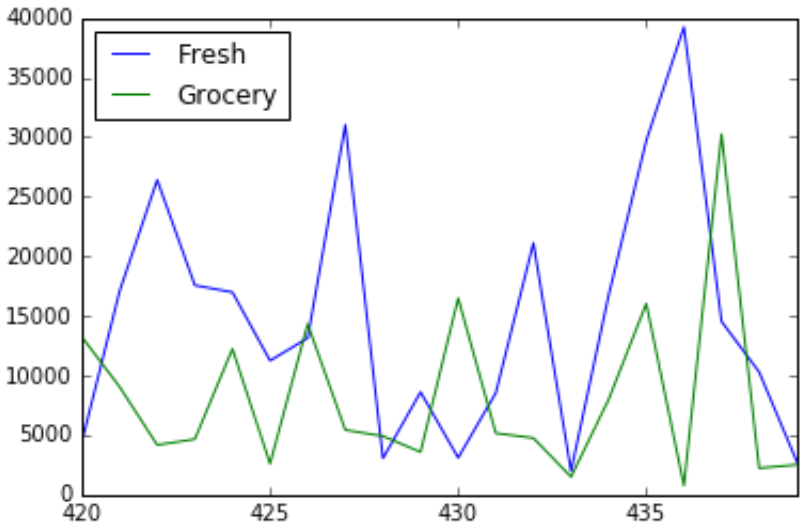
In [4]:
```
pd.DataFrame(pca.transform(data)).head(20).plot()
```

Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x110a9f610>



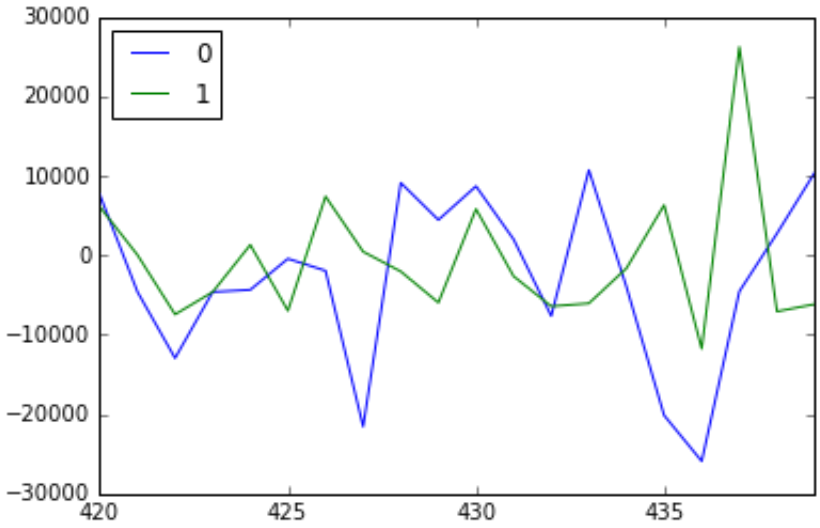
In [5]:
```
data[['Fresh', 'Grocery']].tail(20).plot()
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x1100697d0>



In [6]:
```
pd.DataFrame(pca.transform(data)).tail(20).plot()
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x110166a50>
```



ICA

In [7]:
```
means = pd.DataFrame({'Fresh': [12000.30],
                      'Milk': [5796.27],
                      'Grocery': [7951.28],
                      'Frozen': [3071.93],
                      'Detergents_Paper': [2881.49],
                      'Delicatessen': [1524.87]})

means
```

Out[7]:

	Delicatessen	Detergents_Paper	Fresh	Frozen	Grocery	Milk
0	1524.87	2881.49	12000.3	3071.93	7951.28	5796.27

In [8]:
```
columns = ['Fresh','Milk','Grocery','Frozen','Detergents_Paper','Delic
columns
```

Out[8]:
```
['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicate
ssen']
```

In [9]:
```
centered = pd.DataFrame(data[columns].values - means[columns].values,c
centered
```

Out[9]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	668.7	3859.73	-390.28	-2857.93	-207.49	-186.87
1	-4943.3	4013.73	1616.72	-1309.93	411.51	251.13
2	-5647.3	3011.73	-267.28	-666.93	634.51	6319.13
3	1264.7	-4600.27	-3730.28	3332.07	-2374.49	263.13
4	10614.7	-386.27	-753.28	843.07	-1104.49	3660.13
5	-2587.3	2462.73	-2825.28	-2405.93	-1086.49	-73.87
6	125.7	-2597.27	-976.28	-2591.93	258.51	-979.87
7	-4421.3	-840.27	1474.72	-1402.93	439.51	1041.13
8	-6037.3	-2148.27	-1759.28	-2646.93	-1165.49	-774.87
9	-5994.3	5296.73	10929.72	-1912.93	4543.51	573.13

In [10]:
```
data.head(6)
```

Out[10]:

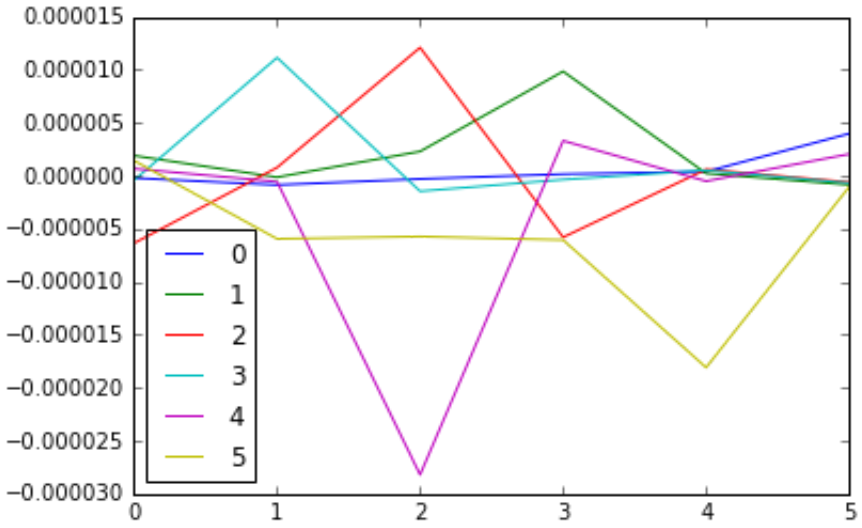
	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	12669	9656	7561	214	2674	1338
1	7057	9810	9568	1762	3293	1776
2	6353	8808	7684	2405	3516	7844
3	13265	1196	4221	6404	507	1788
4	22615	5410	7198	3915	1777	5185
5	9413	8259	5126	666	1795	1451

In [283]:
```
# TODO: Fit an ICA model to the data
# Note: Adjust the data to have center at the origin first!
from sklearn.decomposition import FastICA

ica = FastICA()


# Transform with ICA and then save to disk so that component ordering
# from sklearn.externals import joblib
# joblib.dump(ica, 'ica.pkl')

# Load saved ICA so that PDF report is synchronized with this particul
# ica = joblib.load('ica.pkl')

ica_transformed = ica.fit_transform(centered)
# Print the independent components
print ica.components_
```

```
[[ -2.11386360e-07   1.89486109e-06  -6.36590061e-06  -4.18937145e-0
7
    6.68190869e-07   1.42549974e-06]
 [ -8.65276868e-07  -1.40199306e-07   7.74364825e-07   1.11460794e-0
5
   -5.56447674e-07  -5.95249818e-06]
 [ -3.01172937e-07   2.29372400e-06   1.20938868e-05  -1.46078223e-0
6
   -2.82129448e-05  -5.73421732e-06]
 [  1.53330276e-07   9.84686886e-06  -5.80548986e-06  -3.64661925e-0
7
    3.30210222e-06  -6.05965886e-06]
 [  3.86371662e-07   2.19522462e-07   6.01632422e-07   5.22082298e-0
7
   -5.10896298e-07  -1.80928771e-05]
 [  3.97576537e-06  -8.57719350e-07  -6.17039015e-07  -6.78065373e-0
7
    2.04550191e-06  -1.04734235e-06]]
```

In [284]:
```
pd.DataFrame(ica.components_).plot()
```

Out[284]: <matplotlib.axes._subplots.AxesSubplot at 0x11b274690>



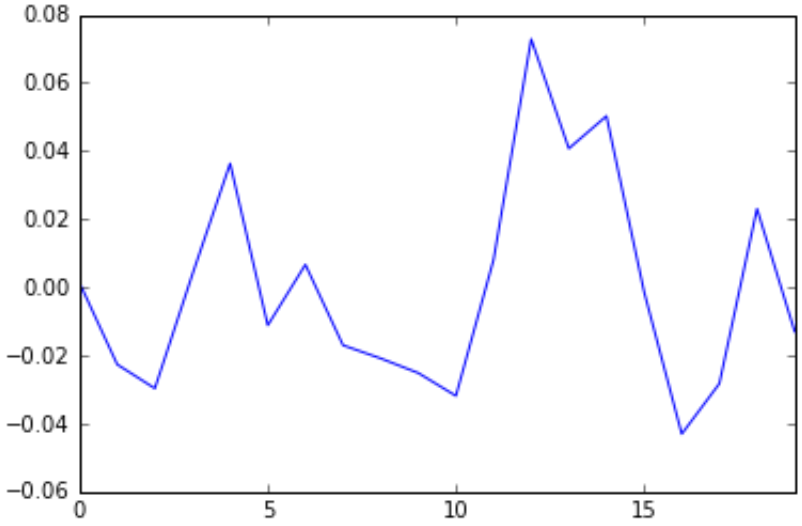
In [42]:
```
pd.DataFrame(ica_transformed)
```

19	-0.031473	-0.016007	-0.036073	-0.015938	-0.012957	-0.015202
20	0.027881	0.011716	0.002777	0.019213	0.024703	-0.032790
21	0.005301	-0.011600	-0.017595	0.026721	-0.022013	0.012206
22	0.067918	0.042752	-0.035968	0.012258	0.073923	0.035355
23	-0.105923	0.250285	0.135204	-0.013809	0.007946	-0.072723
24	-0.007078	0.069353	-0.013517	-0.024748	0.034229	-0.033201
25	0.028174	-0.025539	0.000949	-0.001519	0.023927	-0.027543
26	-0.006439	-0.008975	-0.022945	0.020849	-0.005800	0.005021
27	-0.016693	-0.015122	-0.022491	0.018941	0.013465	-0.026364
28	-0.073464	0.059534	0.036142	-0.075782	-0.046007	-0.027722
29	0.025357	-0.020623	-0.001522	0.019190	0.128469	-0.046214

Independent Component 4 corresponds to "Fresh" Products

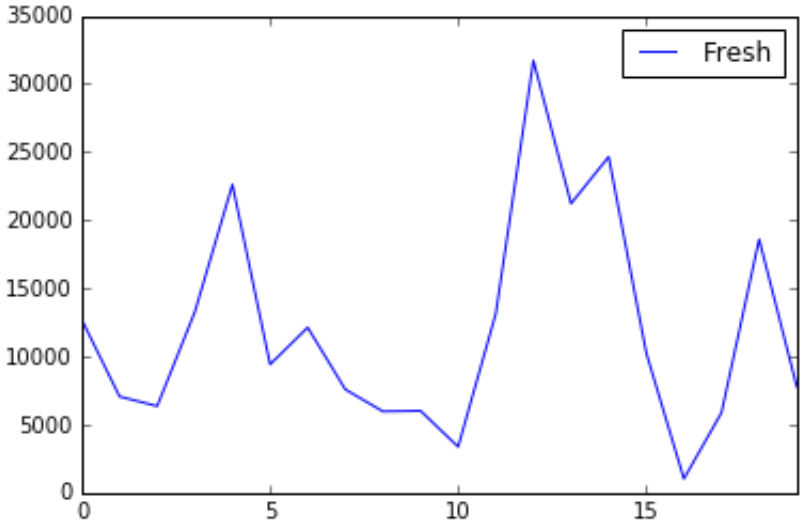
In [60]:
```
pd.DataFrame(ica_transformed)[4].head(20).plot()
```

Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x11519c2d0>



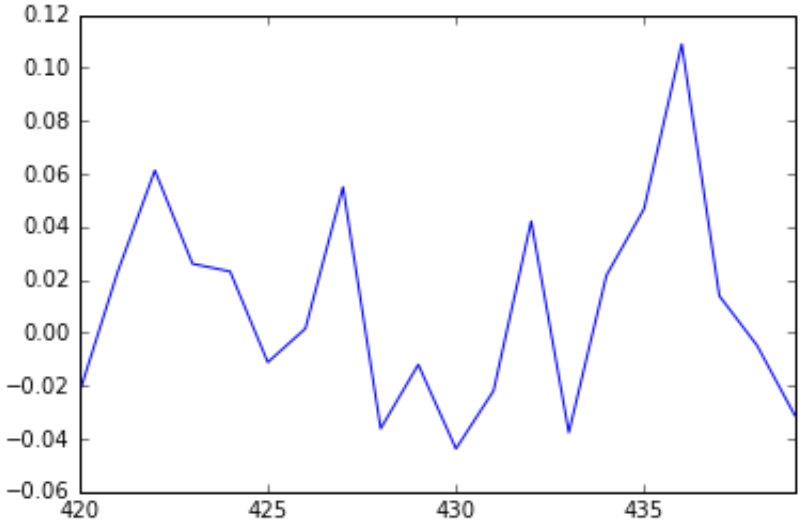
In [44]:
```
pd.DataFrame(data[['Fresh']].head(20)).plot()
```

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x1137e6d90>



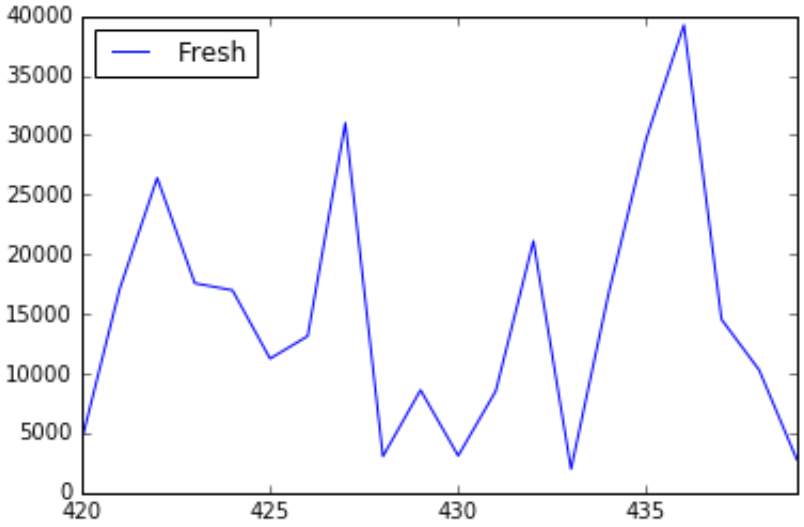
In [71]:
```
pd.DataFrame(ica_transformed)[4].tail(20).plot()
```

Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x116350210>



In [70]:
```
pd.DataFrame(data[['Fresh']].tail(20)).plot()
```

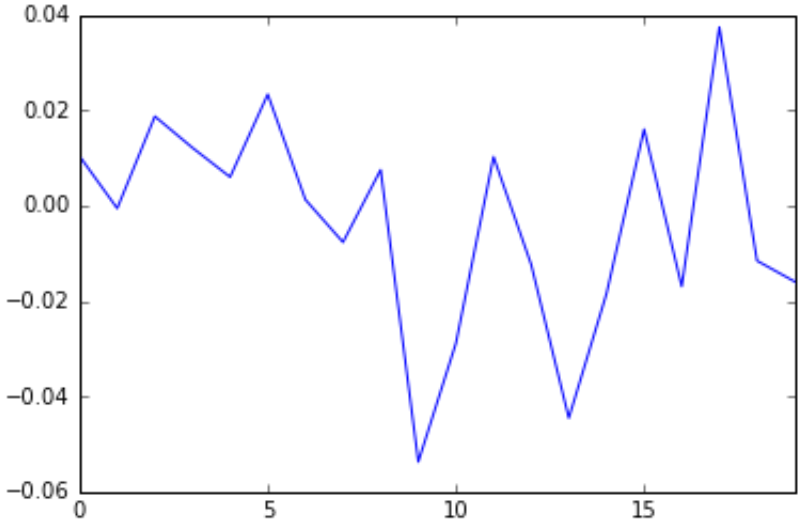
Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x116196450>

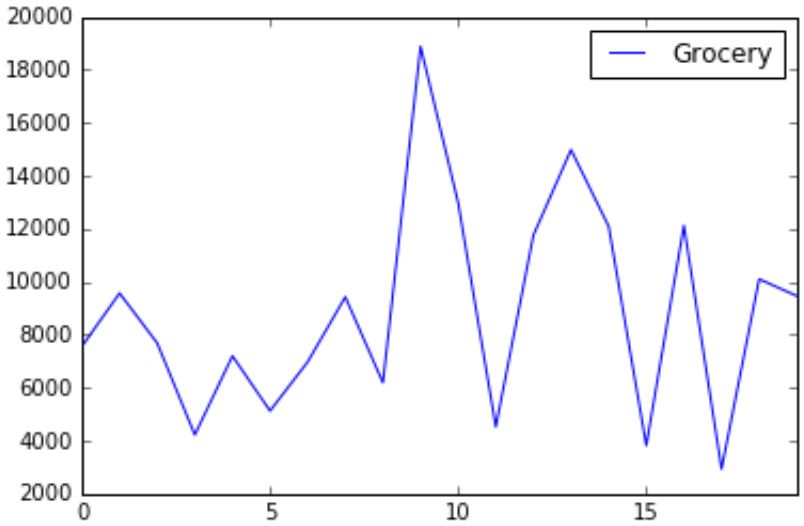


Independent Component 3 seems to be quite related negatively correlated with "Grocery" and "Detergent/Paper" products

In [87]:
```
pd.DataFrame(ica_transformed)[3].head(20).plot()
pd.DataFrame(data[['Grocery']].head(20)).plot()
```

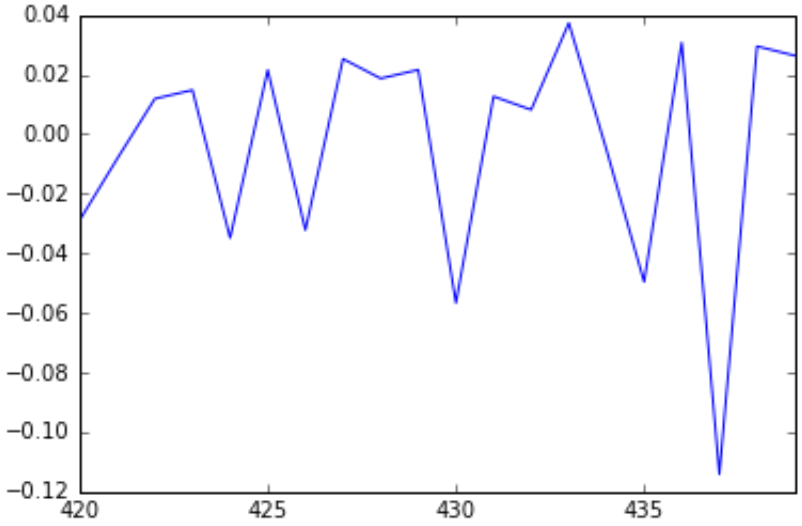
Out[87]: <matplotlib.axes._subplots.AxesSubplot at 0x11883a950>

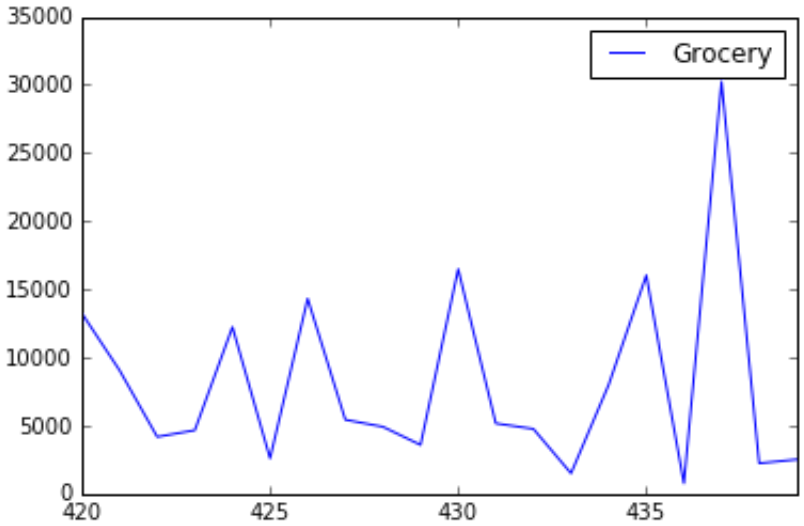




In [88]:
```
pd.DataFrame(ica_transformed)[3].tail(20).plot()
pd.DataFrame(data[['Grocery']].tail(20)).plot()
```

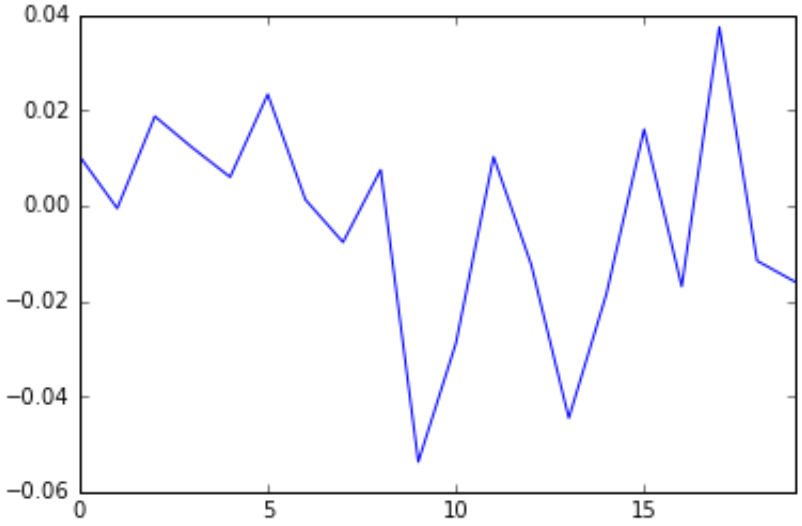
Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x118b2d2d0>

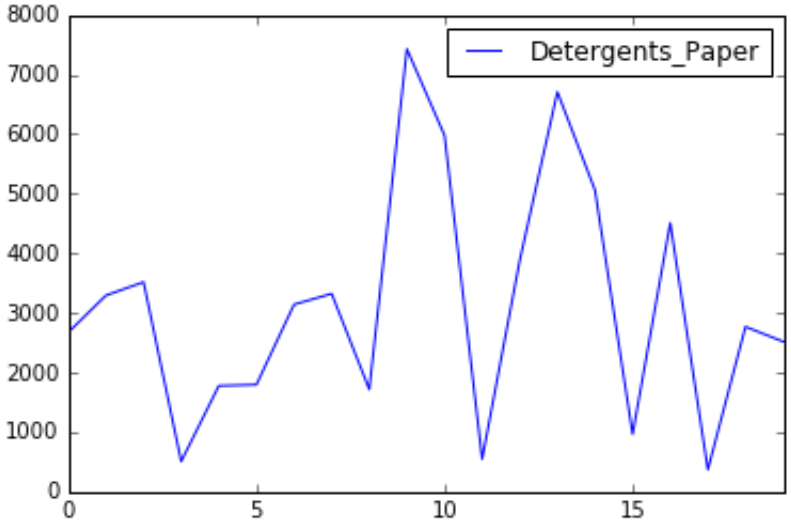




In [89]:
```
pd.DataFrame(ica_transformed)[3].head(20).plot()
pd.DataFrame(data[['Detergents_Paper']].head(20)).plot()
```

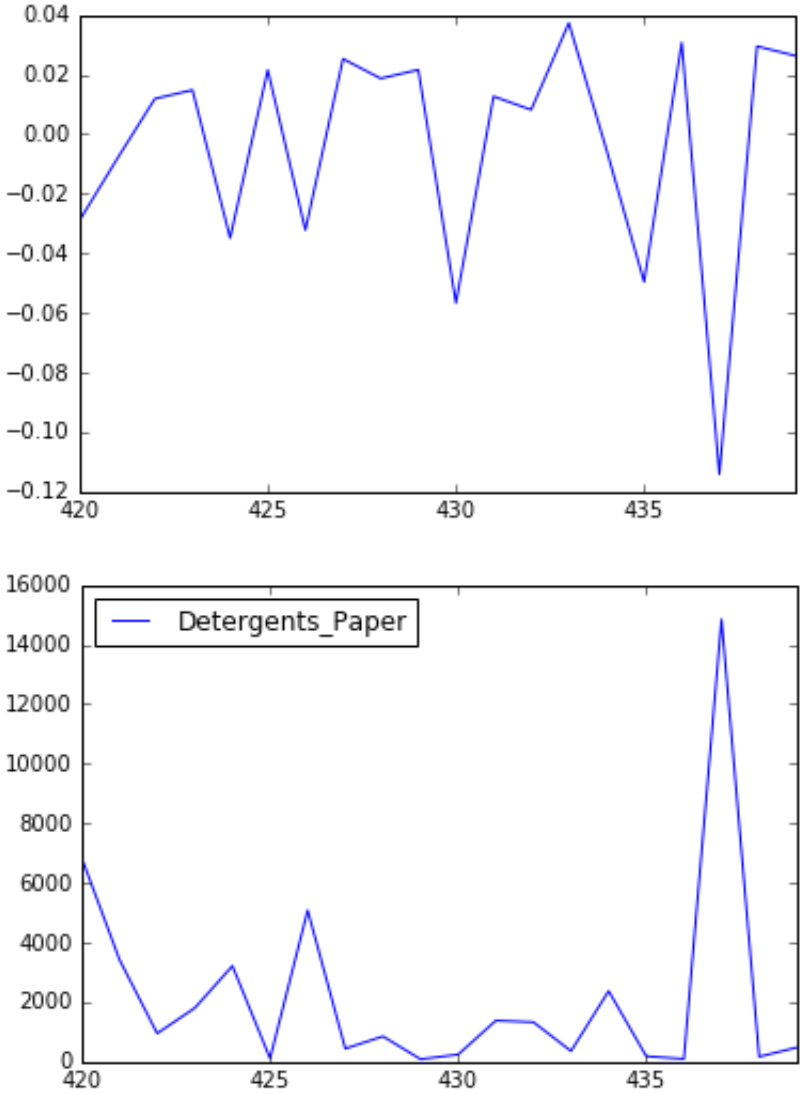
Out[89]: <matplotlib.axes._subplots.AxesSubplot at 0x118af3910>





In [113]:
```
pd.DataFrame(ica_transformed)[3].tail(20).plot()
pd.DataFrame(data[['Detergents_Paper']].tail(20)).plot()
```

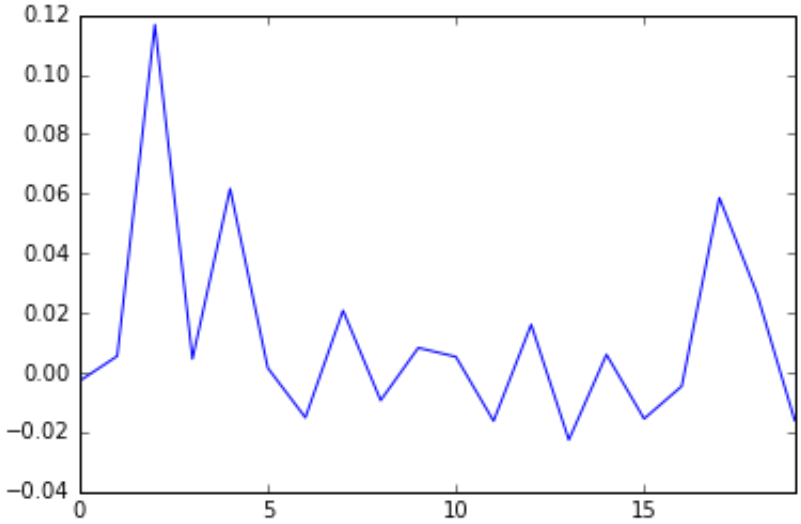
Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0x11b0a6510>



Independent Component 1 follows the general trend of "Delicatessen" products.

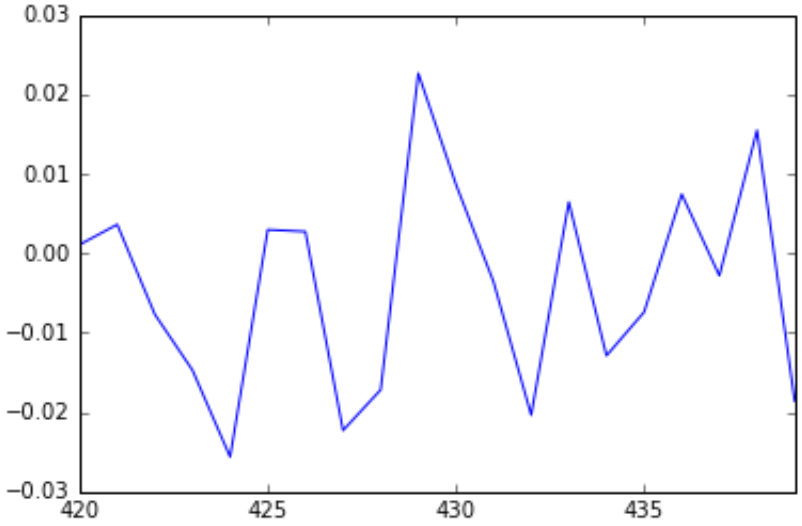
In [82]:
```
pd.DataFrame(ica_transformed)[1].head(20).plot()
pd.DataFrame(data[['Delicatessen']].head(20)).plot()
```

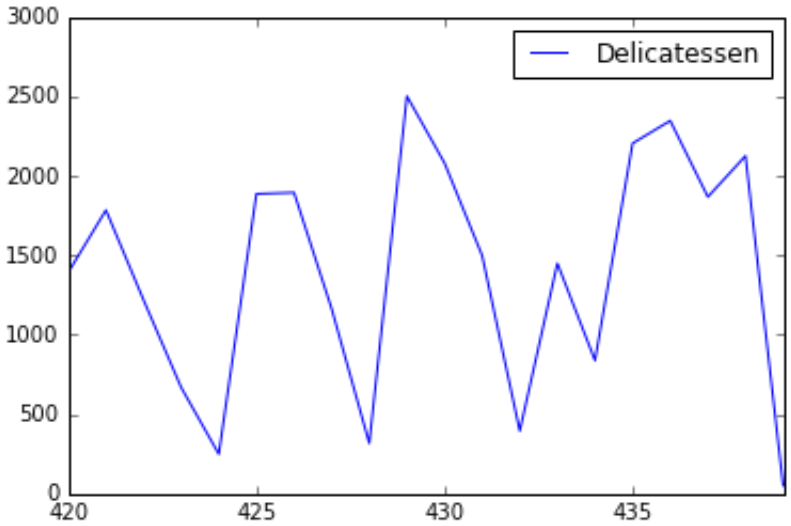
Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x117a15c90>



In [83]:
```
pd.DataFrame(ica_transformed)[1].tail(20).plot()
pd.DataFrame(data[['Delicatessen']].tail(20)).plot()
```

Out[83]: <matplotlib.axes._subplots.AxesSubplot at 0x117ce2250>

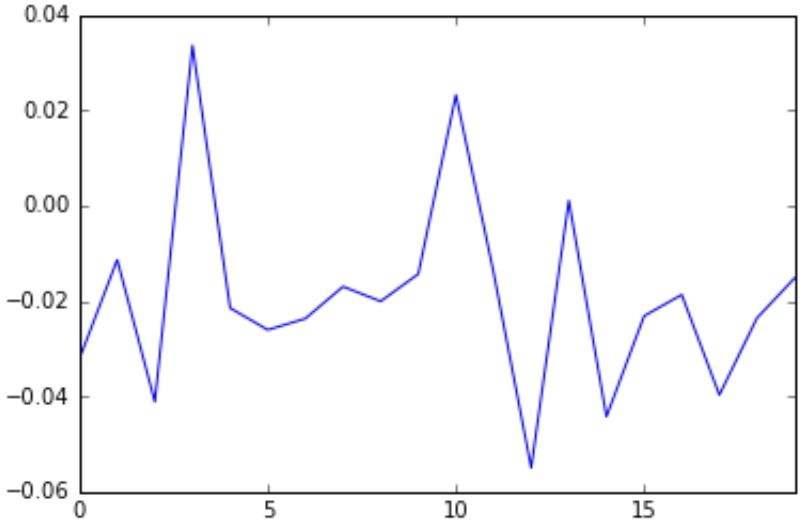


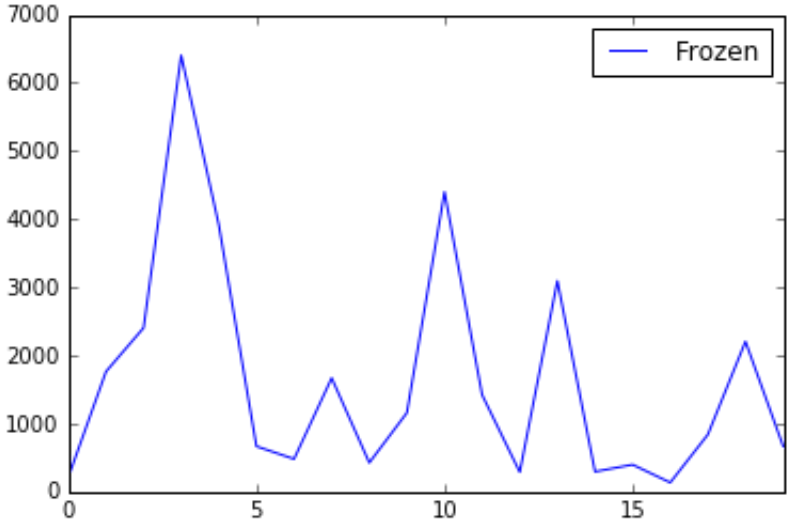


Independent Component 5 seems to correspond to "Frozen" and somewhat to "Milk" products

In [105]:
```
pd.DataFrame(ica_transformed)[5].head(20).plot()
pd.DataFrame(data[['Frozen']].head(20)).plot()
```

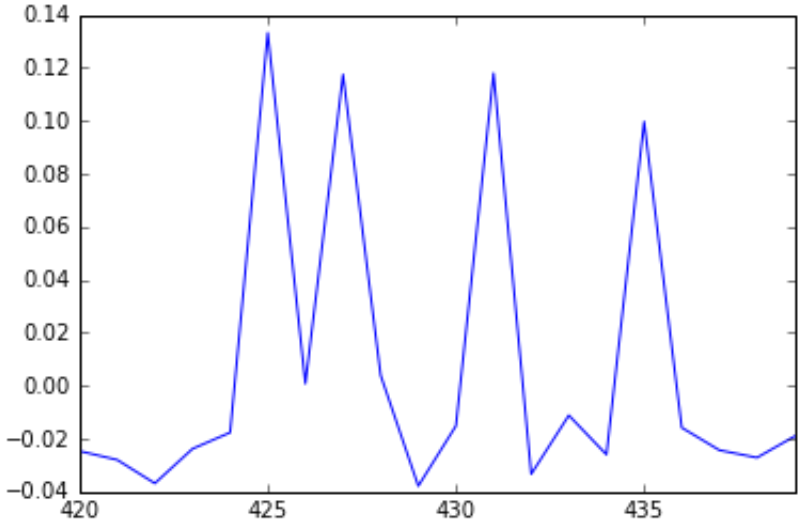
Out[105]: <matplotlib.axes._subplots.AxesSubplot at 0x1179f0250>

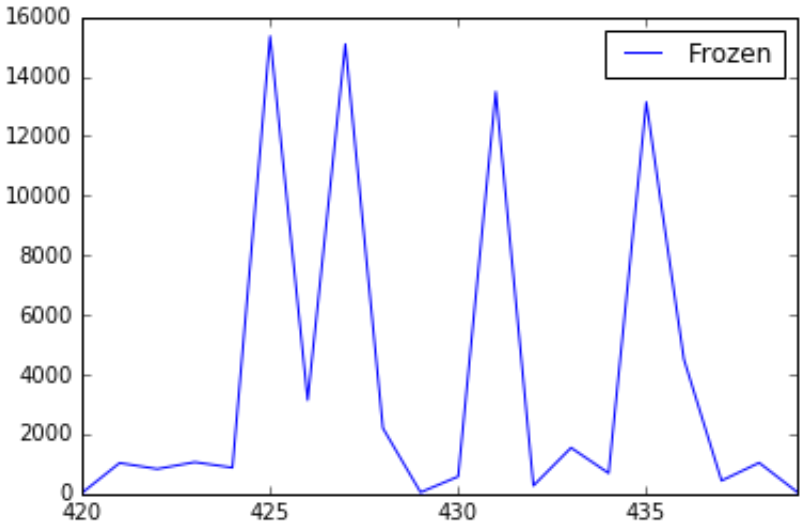




In [106]:
```
pd.DataFrame(ica_transformed)[5].tail(20).plot()
pd.DataFrame(data[['Frozen']].tail(20)).plot()
```

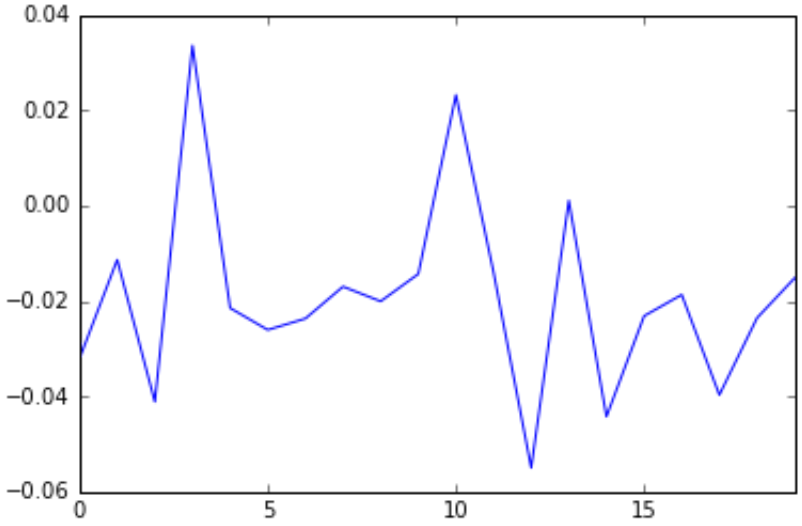
Out[106]: <matplotlib.axes._subplots.AxesSubplot at 0x118acac50>

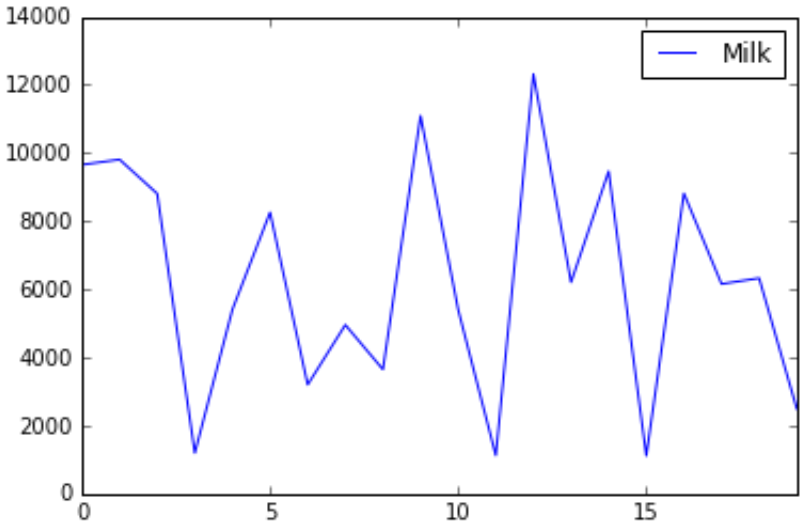




In [121]:
```
pd.DataFrame(ica_transformed)[5].head(20).plot()
pd.DataFrame(data[['Milk']].head(20)).plot()
```

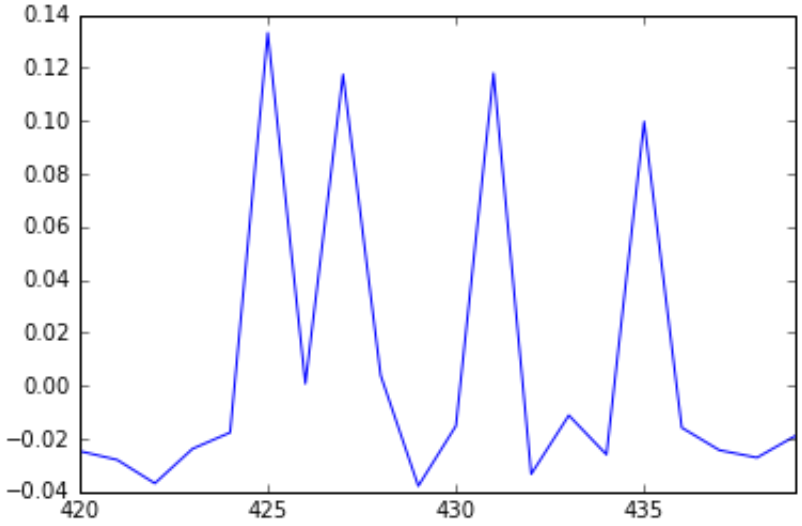
Out[121]: <matplotlib.axes._subplots.AxesSubplot at 0x11b6e2150>

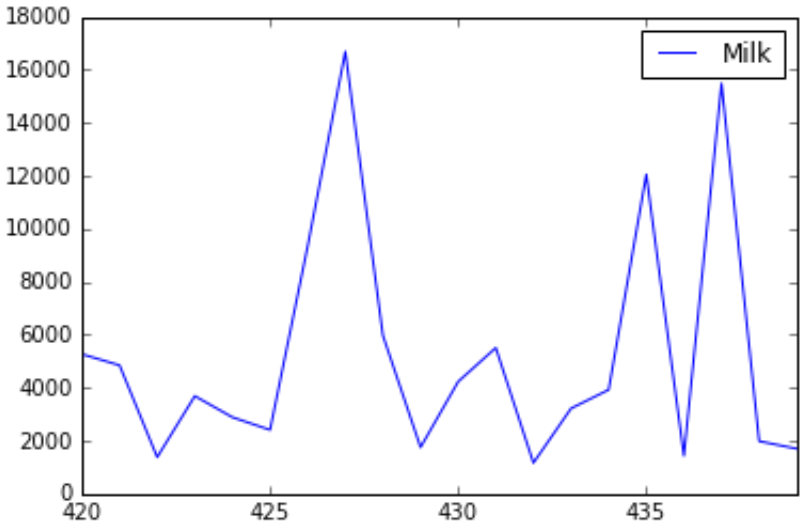




In [122]:
```
pd.DataFrame(ica_transformed)[5].tail(20).plot()
pd.DataFrame(data[['Milk']].tail(20)).plot()
```

Out[122]: <matplotlib.axes._subplots.AxesSubplot at 0x11b9b9650>





4) For each vector in the ICA decomposition, write a sentence or two explaining what sort of object or property it corresponds to. What could these components be used for?

Answer:

Each vector in the ICA decomposition is an independent component. ICA maps the data for each customer into new a new basis, where data in the new vectors are statistically independent from one another. ICA is usually used for separating superimposed signals (see http://scikit-learn.org/stable/modules/decomposition.html#ica (http://scikit-learn.org/stable/modules/decomposition.html#ica)).

It is interesting to note that one component seems to correspond with "fresh" products. A second component seems to correspond with "grocery" and "detergents/paper" products (i.e. dry products). A third component seems to be related to "frozen" and "milk" products

(i.e. refrigerated). The last represents "deli" orders. Overall, these vectors make me think of grocery stores, since most of the grocery stores that I've been have these four sections.

Clustering

In this section you will choose either K Means clustering or Gaussian Mixed Models clustering, which implements expectation-maximization. Then you will sample elements from the clusters to understand their significance.

Choose a Cluster Type

5) What are the advantages of using K Means clustering or Gaussian Mixture Models?

Answer:

The advantage of K Means Clustering is that it is relatively simple to understand. Once we select a number of clusters (k), the centroids are randomly placed on the feature space and are then iteratively "pulled by rubber bands" toward clumps of points. After several iterations, the centroids have hopefully stabilized on global minima (sometimes they don't), and points in the feature space are assigned to the centroid that is closest to them. Another is that K Means is significantly faster than GMM (see benchmarks below). For 2 clusters, on average, K Means (0.2775208 sec) ran about 27% faster than GMM (0.376286 sec).

The advantage of Gaussian Mixture Models is that their goals are similar to K Means, but they also take into account covariance structure in the data (see http://scikit-learn.org/stable/modules/mixture.html (http://scikit-learn.org/stable/modules/mixture.html)). Another advantage of Gaussian Mixture Models is that they do soft assignment (i.e. assign a probability to the class of the point of interest) instead classifying the point of interest as to belonging to only one class. This is nice because it reflects the uncertainty in classifying points that are right in between the decision boundaries, unlike K Means.

6) Below is some starter code to help you visualize some cluster data. The visualization is based on this demo (http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html) from the sklearn documentation.

In [128]:
```
# Import clustering modules
from sklearn.cluster import KMeans
from sklearn.mixture import GMM
```

In [230]:
```
import math
```

In [254]:
```
log_data = data.applymap(lambda x: math.log(x,2))
log_data
```

Out[254]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	13.629015	13.237210	12.884361	7.741467	11.384784	10.385862
1	12.784839	13.260037	13.224002	10.782998	11.685187	10.794416
2	12.633222	13.104599	12.907642	11.231821	11.779719	12.937374
3	13.695337	10.224002	12.043369	12.644758	8.985842	10.804131
4	14.464992	12.401413	12.813380	11.934797	10.795228	12.340128
5	13.200439	13.011751	12.323618	9.379378	10.809768	10.502832
6	13.565816	11.643405	12.767978	8.906891	11.616549	9.090112
7	12.887792	12.274960	13.202430	10.704768	11.697402	11.325305
8	12.541823	11.832890	12.596190	8.731319	10.744834	9.550747
9	12.552189	13.437362	14.204648	10.178665	12.858175	11.034799
10	11.716819	12.399545	13.663336	12.103288	12.545206	10.768184
11	13.682336	10.134426	12.143064	10.471675	9.100662	8.957102
12	14.952832	13.588598	13.521232	8.164907	11.922213	11.517177
13	14.372933	12.599913	13.870943	11.595724	12.711452	9.233620
14	14.589476	13.208387	13.561646	8.199672	12.304351	11.082149
15	13.323758	10.121534	11.899735	8.632995	9.912889	8.686501
16	9.994353	13.105909	13.565221	7.066089	12.138272	10.076816
17	12.520619	12.588012	11.518161	9.712527	8.531381	12.128639
18	14.183093	12.627306	13.301925	11.106563	11.434107	11.635265
19	12.925554	11.284824	13.208234	9.385862	11.298063	8.968667
20	14.098855	12.141788	12.168045	10.057992	11.141469	11.052568
21	12.442684	9.766529	10.972980	11.724087	8.550747	9.152285
22	14.932768	10.904635	12.125736	13.199672	11.217352	12.081483
23	14.686774	15.152562	14.426461	12.331477	12.082482	14.012188
24	14.467032	13.255029	13.751544	11.509280	12.129927	12.496354
25	13.980586	12.046442	12.890834	7.651052	11.966866	5.832890
26	13.272921	9.908393	11.482304	11.621594	7.918863	9.702173
27	13.801304	9.649256	11.572227	8.921841	6.643856	9.016808

28	12.005975	14.322210	14.663836	10.177420	13.070792	12.345960
29	15.394999	11.036174	11.349281	10.228819	10.112440	9.684749
...	...	...	...	...	...	...
410	12.695446	11.033423	12.155767	10.439831	10.861087	10.885696
411	11.053926	11.683433	11.679920	10.584023	7.876517	12.091766
412	6.599913	11.815784	13.598053	6.614710	11.536247	5.954196
413	12.282799	12.246444	12.695446	14.124929	9.832890	11.249706
414	12.543274	10.958553	11.738515	12.471421	10.148477	8.179909
415	12.937006	12.561765	13.062046	10.723661	11.789534	10.871905
416	12.099677	13.417325	13.413099	9.727920	12.715962	9.955650
417	12.306347	12.424954	13.432411	8.507795	11.766943	10.053926
418	9.366322	13.052228	14.184720	7.055282	12.718533	9.599913
419	13.113254	11.885315	11.118292	9.306062	10.625709	10.570804
420	12.121534	12.362492	13.691198	4.643856	12.735133	10.443980
421	14.058584	12.242876	13.144180	10.009829	11.737670	10.800900
422	14.688250	10.427313	12.026523	9.696968	9.888743	10.250298
423	14.100416	11.847840	12.185185	10.048487	10.816184	9.383704
424	14.051549	11.493855	13.578373	9.771489	11.649705	7.960002
425	13.456739	11.233620	11.340406	13.905763	6.754888	10.881114
426	13.681019	13.190288	13.805341	11.617008	12.310329	10.887221
427	14.920539	14.026437	12.406471	13.880540	8.778077	10.183635
428	11.573174	12.543515	12.261507	11.101976	9.731319	8.308339
429	13.071295	10.773139	11.805744	5.554589	6.392317	11.288289
430	11.596656	12.046442	14.008691	9.167418	7.912889	11.022368
431	13.058837	12.426789	12.333155	13.719175	10.427313	10.548822
432	14.366117	10.182394	12.214926	8.071462	10.375039	8.625709
433	10.952741	11.651949	10.543998	10.589651	8.475733	10.500842
434	14.030236	11.937374	12.964702	9.426265	11.211280	9.710806
435	14.858321	13.556865	13.968217	13.681129	7.507795	11.105909
436	15.259596	10.482808	9.577429	12.138912	6.539159	11.195987
437	13.826846	13.918863	14.884314	8.771489	13.857301	10.866506
438	13.328955	10.952013	11.124121	10.019591	7.392317	11.053247

439	11.444497	10.729621	11.293472	6.022368	8.897845	5.700440

440 rows × 6 columns

In [272]:
```
# TODO: First we reduce the data to two dimensions using PCA to captur
reduced_data = pca.transform(log_data)
print reduced_data[:10]  # print upto 10 elements
```

```
[[ 13444.53792159  -8804.4225931 ]
 [ 13444.84949531  -8803.98162552]
 [ 13444.82219707  -8804.13846858]
 [ 13444.09756691  -8807.57156392]
 [ 13443.05104379  -8805.18280252]
 [ 13444.75669019  -8805.15430001]
 [ 13444.71225598  -8805.33778374]
 [ 13444.84551114  -8804.48138185]
 [ 13445.689063    -8805.54703103]
 [ 13445.07883115  -8802.66103033]]
```

In [257]:
```
# TODO: Implement your clustering algorithm here, and fit it to the re
# The visualizer below assumes your clustering object is named 'cluste

def cluster(clusterer):
    clusterer.fit(reduced_data)
    clusters = clusterer
    print clusters
    return clusters
```

In [258]:
```
# Plot the decision boundary by building a mesh grid to populate a gra

def plot_boundary(clusters):
    x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].ma
    y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].ma
    hx = (x_max-x_min)/1000.
    hy = (y_max-y_min)/1000.
    xx, yy = np.meshgrid(np.arange(x_min, x_max, hx), np.arange(y_min,

    # Obtain labels for each point in mesh. Use last trained model.
    Z = clusters.predict(np.c_[xx.ravel(), yy.ravel()])
    return Z,xx,yy,x_min,x_max,y_min,y_max
```

In [259]:
```
# TODO: Find the centroids for KMeans or the cluster means for GMM

def cluster_means(clusters, func_name):
    centroids = getattr(clusters, func_name)
    print centroids
    return centroids
```

In [260]:
```
# Put the result into a color plot
def color_plot(clusters, Z,xx,yy,x_min,x_max,y_min,y_max,centroids):
    Z = Z.reshape(xx.shape)
    plt.figure(1)
    plt.clf()
    plt.imshow(Z, interpolation='nearest',
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               cmap=plt.cm.Paired,
               aspect='auto', origin='lower')

    plt.plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize=
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=169, linewidths=3,
                color='w', zorder=10)
    plt.title('Clustering on the wholesale grocery dataset (PCA-reduce
              'Centroids are marked with white cross')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xticks(())
    plt.yticks(())
    plt.show()
```

In [261]:
```
def cluster_and_plot(clusterer,func_name):
    clusters = cluster(clusterer)
    Z,xx,yy,x_min,x_max,y_min,y_max = plot_boundary(clusters)
    centroids = cluster_means(clusters,func_name)
    color_plot(clusters,Z,xx,yy,x_min,x_max,y_min,y_max,centroids)
```

In [262]:
```
import time
```

In [263]:
```
def benchmark(clusterer, func_name):
    num_times = 10
    total = 0.0
    for i in range(num_times):
        start = time.clock()
        cluster_and_plot(clusterer, func_name)
        end = time.clock()
        diff = end - start
        total += diff
    print "Benchmark:", total/num_times, "seconds"
```

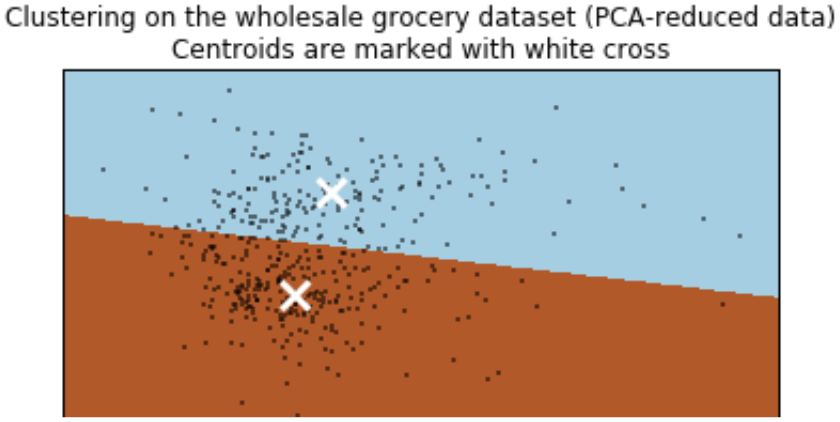
Benchmark

In [264]:
```
benchmark(GMM(n_components=2),'means_')
```

```
GMM(covariance_type='diag', init_params='wmc', min_covar=0.001,
  n_components=2, n_init=1, n_iter=100, params='wmc', random_state=N
one,
  thresh=None, tol=0.001)
[[ 13446.75651552  -8805.19081708]
 [ 13444.65945797  -8806.9119063 ]]
```

In [265]:
```
benchmark(KMeans(n_clusters=2), 'cluster_centers_')
```

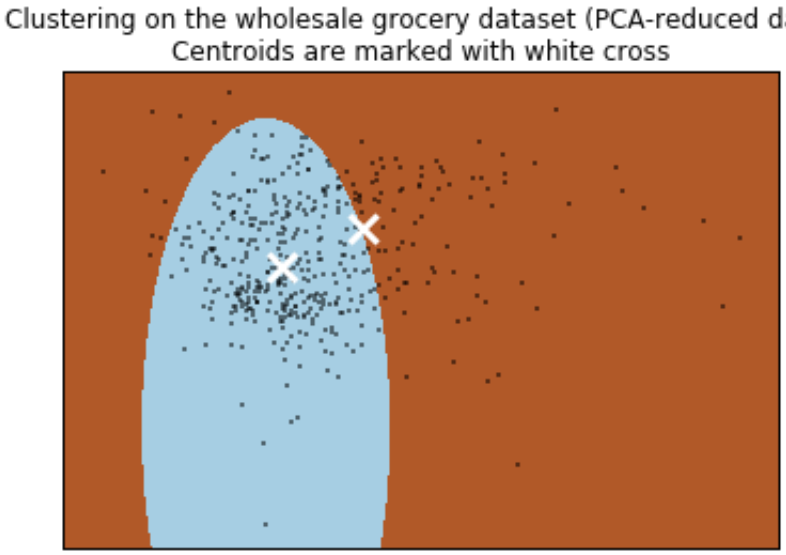
```
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=2,
n_init=10,
    n_jobs=1, precompute_distances='auto', random_state=None, to
l=0.0001,
    verbose=0)
[[ 13445.9224838   -8803.67098358]
 [ 13445.00494534  -8808.227584  ]]
```



Compare clusters

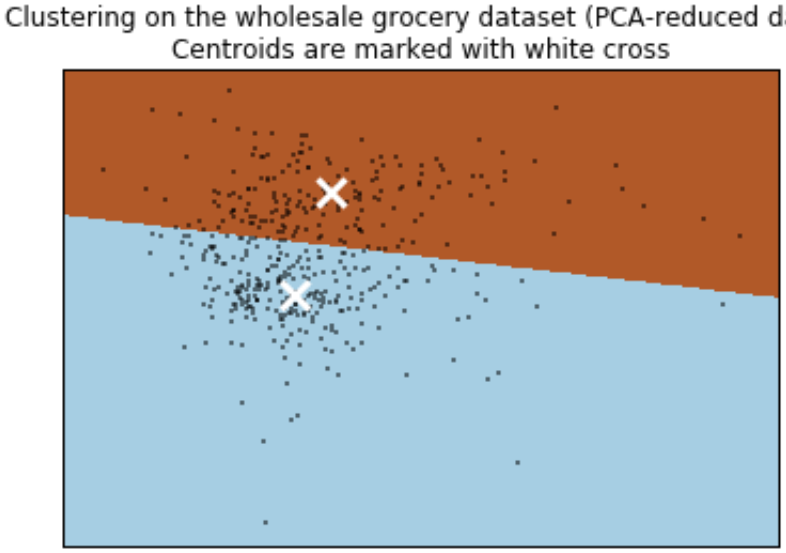
In [266]:
```
cluster_and_plot(GMM(n_components=2),'means_')
```

```
GMM(covariance_type='diag', init_params='wmc', min_covar=0.001,
  n_components=2, n_init=1, n_iter=100, params='wmc', random_state=N
one,
  thresh=None, tol=0.001)
[[ 13444.6591951   -8806.91510978]
 [ 13446.75167695  -8805.18922977]]
```



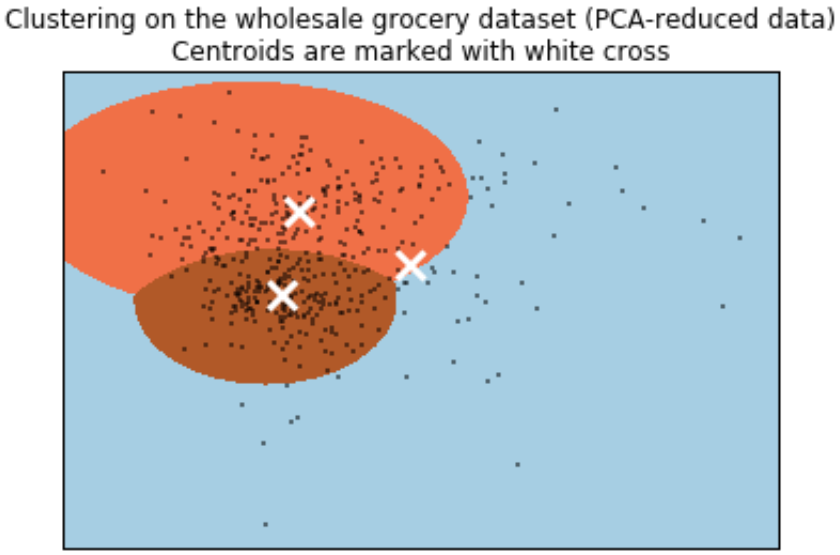
In [267]:
```
cluster_and_plot(KMeans(n_clusters=2), 'cluster_centers_')
```

```
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=2,
n_init=10,
    n_jobs=1, precompute_distances='auto', random_state=None, to
l=0.0001,
    verbose=0)
[[ 13445.00494534  -8808.227584  ]
 [ 13445.9224838   -8803.67098358]]
```



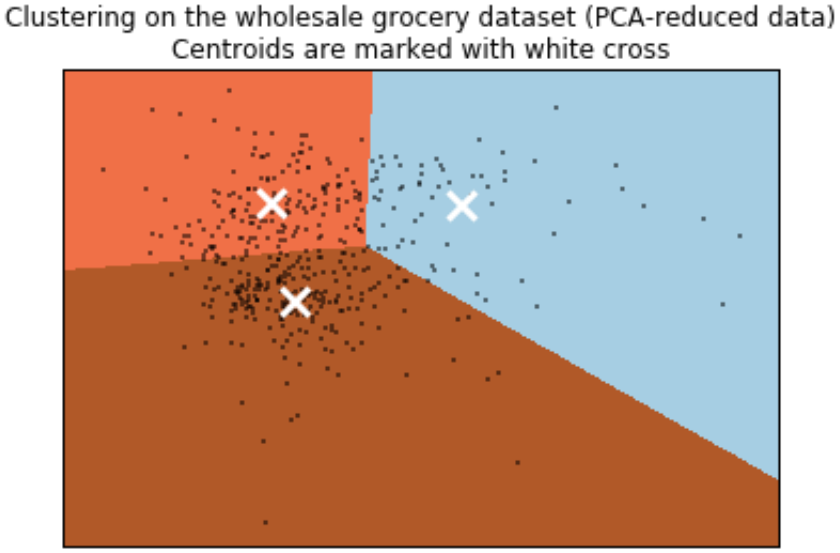
In [268]:
```
cluster_and_plot(GMM(n_components=3),'means_')
```

```
GMM(covariance_type='diag', init_params='wmc', min_covar=0.001,
  n_components=3, n_init=1, n_iter=100, params='wmc', random_state=N
one,
  thresh=None, tol=0.001)
[[ 13447.96738662  -8806.78423234]
 [ 13445.10491018  -8804.37781252]
 [ 13444.69465378  -8808.16172005]]
```



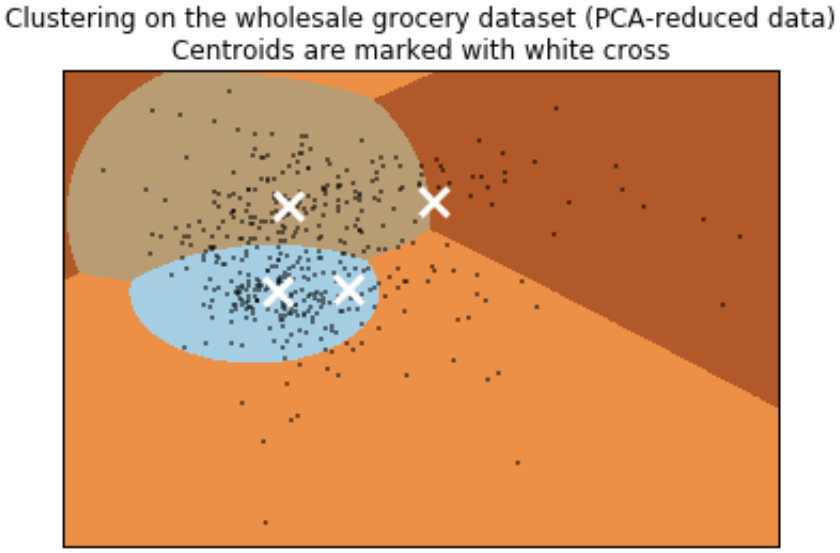
In [269]:
```
cluster_and_plot(KMeans(n_clusters=3), 'cluster_centers_')
```

```
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=3,
n_init=10,
    n_jobs=1, precompute_distances='auto', random_state=None, to
l=0.0001,
    verbose=0)
[[ 13449.2778563   -8804.248122  ]
 [ 13444.3967184   -8804.11428254]
 [ 13445.02048654  -8808.5521692 ]]
```



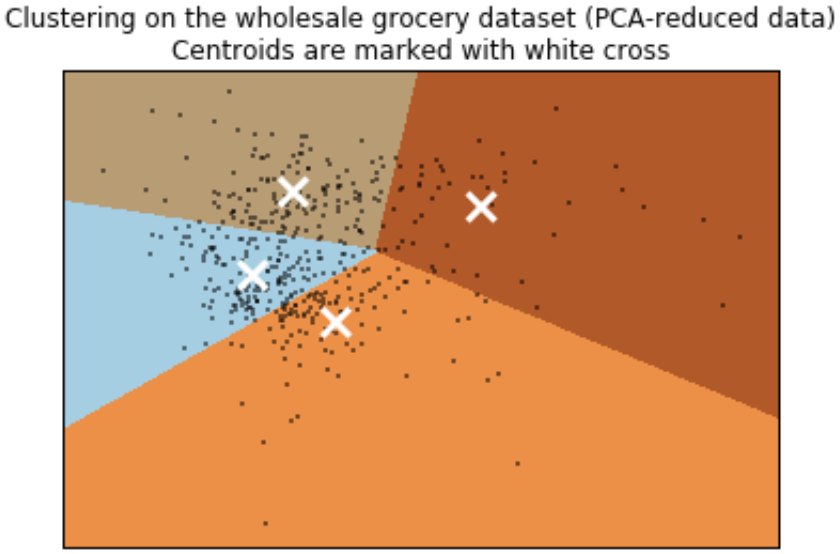
In [270]:
```
cluster_and_plot(GMM(n_components=4),'means_')
```

```
GMM(covariance_type='diag', init_params='wmc', min_covar=0.001,
  n_components=4, n_init=1, n_iter=100, params='wmc', random_state=N
one,
  thresh=None, tol=0.001)
[[ 13444.60196343  -8808.04786655]
 [ 13444.86948537  -8804.18977209]
 [ 13446.38097457  -8807.92156604]
 [ 13448.57383606  -8803.99540026]]
```



In [271]:
```
cluster_and_plot(KMeans(n_clusters=4), 'cluster_centers_')
```

```
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=4,
n_init=10,
    n_jobs=1, precompute_distances='auto', random_state=None, to
l=0.0001,
    verbose=0)
[[ 13443.9315741   -8807.28392791]
 [ 13444.94089245  -8803.59277804]
 [ 13446.0324121   -8809.42296143]
 [ 13449.76601576  -8804.26056814]]
```



7) What are the central objects in each cluster? Describe them as customers.

Answer:

Central objects in each cluster represent the centroids. In terms of customers, since they are "smack dab in the middle" of each cluster, they seem to represent the "average" customer in that grouping.

Conclusions

8) Which of these techniques did you feel gave you the most insight into the data?

Answer:

I felt that the best techniques to use were to log-transform the data, and then use PCA to summarize the data into two principal components that roughly correspond with "fresh" and "grocery" products respectively. The log transformation was helpful because it helped normalize the data, especially considering that there was huge variation in the annual spending of customers on "fresh" and "grocery" products. After doing PCA, using Gaussian Mixture Models with 2 or 3 components made sense. With 2 GMM components, I could

quickly see that one component captures some sort of "oval" clump, while the other captures everything that's not in the oval clump. Perhaps, these 2 clusters correspond to "big" and "small" businesses. With 3 GMM components, the rough oval shape became divided into two smaller ovals, while the 3rd component captured everything outside the two ovals.

In the 3-component clusters using GMM, we can identify 3 groups: one of customers that order lots of fresh products, another of customers that order lots of fresh products AND grocery items, and lastly, one of customers that don't belong in those two groups. My hypothesis is that unhappy customers probably belong to the cluster of customers that predominantly order only fresh products. Since the delivery schedule has changed from morning to evening, and now that it's in bulk, those customers are probably most affected. The products in the "fresh" category are probably not as "fresh" as they used to be. Assuming that they have regular hours of operation (i.e. they have more hours open during the day than during night), this means that the fresh products get stored at night, and probably have to wait till the next day to get bought. Also, "bulk" deliveries probably means we are not delivering to them as frequently. This further supports the idea that the "fresh" products are likely to become more stale because of the new changes. I would test this idea out, again, by figuring out how concentrated or spread out the unhappy customers are in the clusters.

9) How would you use that technique to help the company design new experiments?

Answer:

Now that we have some clusters, and given the information that some companies have complaints about the new change, I would try to project the customers that we know have issues with the new change onto the first two principal components as we've done above, and then figure out which cluster they belong. An interesting experiment would be to figure out if these unhappy customers are overwhelmingly concentrated in one cluster. For example, in the case of three GMM clusters, a null hypothesis could be that customers who have problems should be evenly distributed throughout the three clusters. Once we have gathered enough complaints, we could map each of those customers onto the new projection and test our hypothesis. If we can safely reject the null hypothesis (i.e. unhappy customers overwhelmingly are concentrated in one cluster), we could then reach out to the other customers who belong to that cluster, pay more attention to their needs and maybe give them special offers so that they keep doing business with us (such as restoring the daily morning deliveries).

10) How would you use that data to help you predict future customer needs?

Answer:

After the A/B test, if we got to the point of successfully identifying the cluster that most of the unhappy customers belong to, it's probably likely for those people in that cluster to prefer the old method of delivery. We predict that those customers in that cluster probably like more frequent, morning deliveries. In other words, we could assign them labels based on which cluster they belong to. After that, we could treat this as a supervised learning problem.

In [ ]: