customer_segments

March 5, 2016

1 Creating Customer Segments

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 [85]:
```
# Import libraries: NumPy, pandas, matplotlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

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

import warnings
warnings.filterwarnings('ignore')

pd.options.display.float_format = '{:.5f}'.format
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

# Read dataset
data = pd.read_csv("wholesale-customers.csv")
num_features=data.shape[1]
num_data_points=data.shape[0]
print "Dataset has {} rows, {} columns".format(num_data_points,num_features)
print data.head()  # print the first 5 rows
print data.describe()

'''
TODOs:
1. Create 3-D plot for pca vectors.
2. Apply PCA on ICA-demixed-transformed data.
3. Draw elbow graph to identify k in k-means.
'''
```

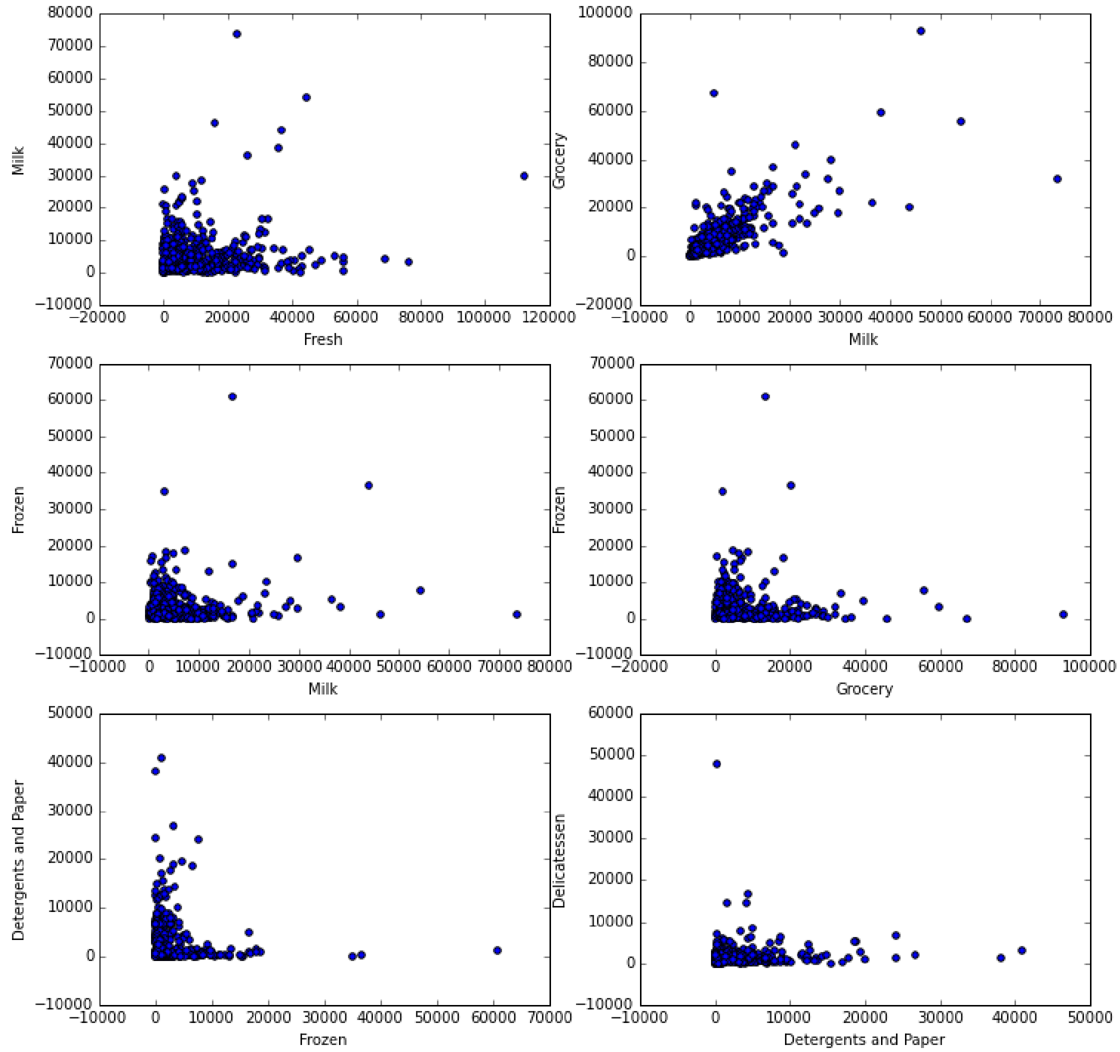
```
Dataset has 440 rows, 6 columns
   Fresh  Milk  Grocery  Frozen  Detergents_Paper  Delicatessen
```

```
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
              Fresh         Milk     Grocery       Frozen  Detergents_Paper  Delicatessen
count    440.00000    440.00000    440.00000    440.00000         440.00000     440.00000
mean   12000.29773   5796.26591   7951.27727   3071.93182        2881.49318    1524.87045
std    12647.32887   7380.37717   9503.16283   4854.67333        4767.85445    2820.10594
min        3.00000     55.00000      3.00000     25.00000           3.00000       3.00000
25%     3127.75000   1533.00000   2153.00000    742.25000         256.75000     408.25000
50%     8504.00000   3627.00000   4755.50000   1526.00000         816.50000     965.50000
75%    16933.75000   7190.25000  10655.75000   3554.25000        3922.00000    1820.25000
max   112151.00000  73498.00000  92780.00000  60869.00000       40827.00000   47943.00000
```

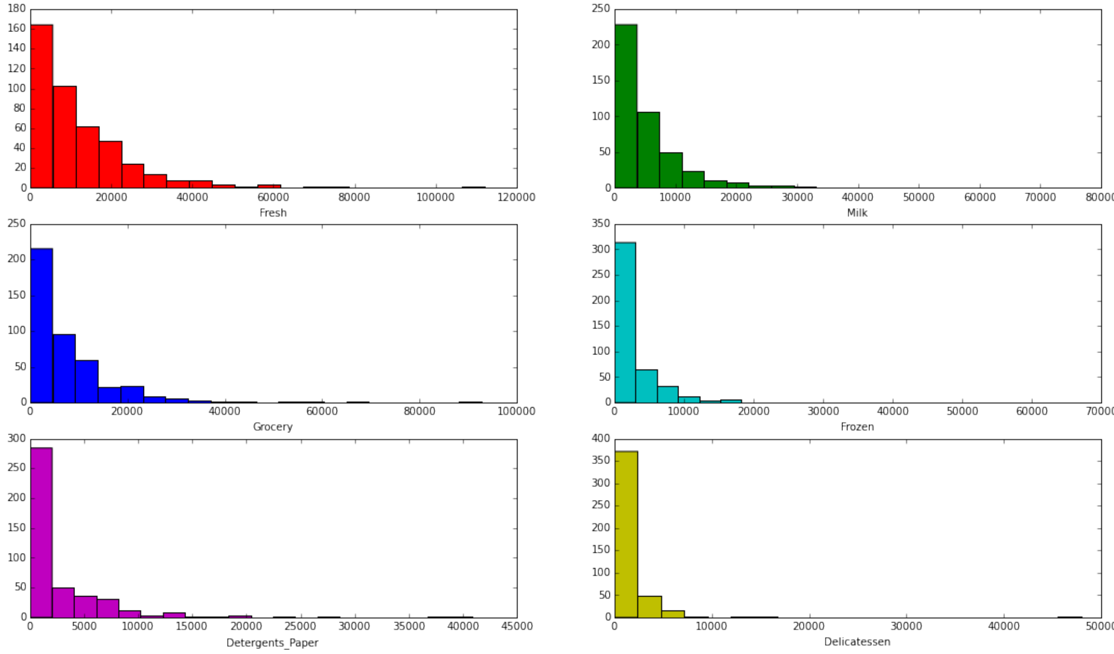
Out[85]: '\nTODOs:\n1. Create 3-D plot for pca vectors.\n2. Apply PCA on ICA-demixed-transformed data.\n

In [86]:
```
## Cleanup data, remove outliers.

f, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots(3, 2)
ax1.scatter(data.iloc[:,0],data.iloc[:,1])
ax1.set_xlabel('Fresh')
ax1.set_ylabel('Milk')

ax2.scatter(data.iloc[:,1],data.iloc[:,2])
ax2.set_xlabel('Milk')
ax2.set_ylabel('Grocery')


ax3.scatter(data.iloc[:,1],data.iloc[:,3])
ax3.set_xlabel('Milk')
ax3.set_ylabel('Frozen')

ax4.scatter(data.iloc[:,2],data.iloc[:,3])
ax4.set_xlabel('Grocery')
ax4.set_ylabel('Frozen')


ax5.scatter(data.iloc[:,3],data.iloc[:,4])
ax5.set_xlabel('Frozen')
ax5.set_ylabel('Detergents and Paper')


ax6.scatter(data.iloc[:,4],data.iloc[:,5])
ax6.set_xlabel('Detergents and Paper')
ax6.set_ylabel('Delicatessen')

fig = plt.gcf()
fig.set_size_inches(12, 12)
#fig.set_size_inches(18.5, 10.5, forward=True)
plt.show()



print "-"*100
```

```
print "Histogram of spending on specific product types."
print "-"*100


f, ((axis1, axis2), (axis3, axis4), (axis5, axis6)) = plt.subplots(3, 2)

### Visualize data spread.
colormap = np.array(['r', 'g', 'b','c','m','y'])
f.axes[0].hist(data.iloc[:,0],bins=20,color=colormap[0])
f.axes[0].set_xlabel(data.columns.values[0]);

f.axes[1].hist(data.iloc[:,1],bins=20,color=colormap[1])
f.axes[1].set_xlabel(data.columns.values[1]);


f.axes[2].hist(data.iloc[:,2],bins=20,color=colormap[2])
f.axes[2].set_xlabel(data.columns.values[2]);

f.axes[3].hist(data.iloc[:,3],bins=20,color=colormap[3])
f.axes[3].set_xlabel(data.columns.values[3]);

f.axes[4].hist(data.iloc[:,4],bins=20,color=colormap[4])
f.axes[4].set_xlabel(data.columns.values[4]);

f.axes[5].hist(data.iloc[:,5],bins=20,color=colormap[5])
f.axes[5].set_xlabel(data.columns.values[5]);

fig = plt.gcf()
fig.set_size_inches(12, 12)
fig.set_size_inches(18.5, 10.5, forward=True)
plt.show()
```



Histogram of spending on specific product types.

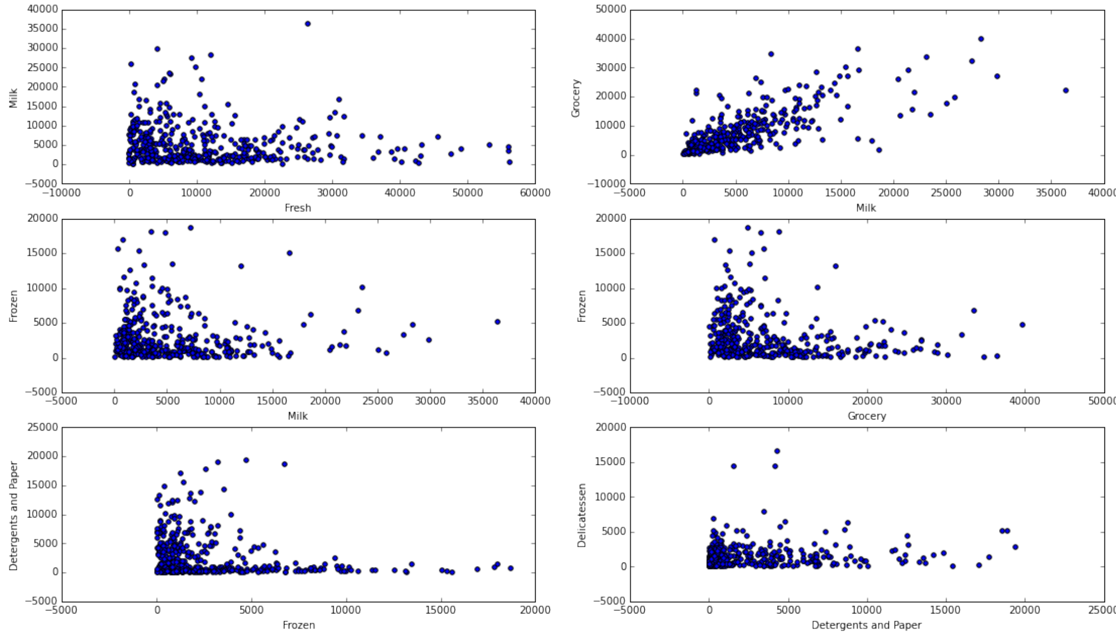


In [87]:
```
## Cleaning outliers could be useful, since it would remove noise which is more prevelent in l


cleaned_data=data.copy(deep=True)

cleaned_data=cleaned_data[cleaned_data['Fresh']<60000]
cleaned_data=cleaned_data[cleaned_data['Milk']<50000]
cleaned_data=cleaned_data[cleaned_data['Grocery']<50000]
cleaned_data=cleaned_data[cleaned_data['Frozen']<30000]
cleaned_data=cleaned_data[cleaned_data['Detergents_Paper']<20000]
cleaned_data=cleaned_data[cleaned_data['Delicatessen']<20000]

# Removed scaling since units are same, and feature-wise expenses are part of same expense, i.
#from sklearn import preprocessing
#cleaned_data[['Fresh','Milk','Grocery','Frozen','Detergents_Paper','Delicatessen']] = cleaned

print '-'*100
print " Cleaned, centered, and normalized data."
print '-'*100
f, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots(3, 2)
ax1.scatter(cleaned_data.iloc[:,0],cleaned_data.iloc[:,1])
ax1.set_xlabel('Fresh')
ax1.set_ylabel('Milk')

ax2.scatter(cleaned_data.iloc[:,1],cleaned_data.iloc[:,2])
ax2.set_xlabel('Milk')
ax2.set_ylabel('Grocery')

ax3.scatter(cleaned_data.iloc[:,1],cleaned_data.iloc[:,3])
ax3.set_xlabel('Milk')
```

```
ax3.set_ylabel('Frozen')

ax4.scatter(cleaned_data.iloc[:,2],cleaned_data.iloc[:,3])
ax4.set_xlabel('Grocery')
ax4.set_ylabel('Frozen')


ax5.scatter(cleaned_data.iloc[:,3],cleaned_data.iloc[:,4])
ax5.set_xlabel('Frozen')
ax5.set_ylabel('Detergents and Paper')


ax6.scatter(cleaned_data.iloc[:,4],cleaned_data.iloc[:,5])
ax6.set_xlabel('Detergents and Paper')
ax6.set_ylabel('Delicatessen')

fig = plt.gcf()
fig.set_size_inches(12, 12)
fig.set_size_inches(18.5, 10.5, forward=True)
plt.show()
print " -> Fresh vs Milk, Milk vs Frozen, Grocery vs Frozen, and Detergent_paper vs Frozen all


print "-"*100
print "Histogram of spending on specific product types."
print "-"*100


f, ((axis1, axis2), (axis3, axis4), (axis5, axis6)) = plt.subplots(3, 2)

### Visualize data spread.
colormap = np.array(['r', 'g', 'b','c','m','y'])
f.axes[0].hist(cleaned_data.iloc[:,0],bins=20,color=colormap[0])
f.axes[0].set_xlabel(cleaned_data.columns.values[0]);

f.axes[1].hist(cleaned_data.iloc[:,1],bins=20,color=colormap[1])
f.axes[1].set_xlabel(cleaned_data.columns.values[1]);


f.axes[2].hist(cleaned_data.iloc[:,2],bins=20,color=colormap[2])
f.axes[2].set_xlabel(cleaned_data.columns.values[2]);

f.axes[3].hist(cleaned_data.iloc[:,3],bins=20,color=colormap[3])
f.axes[3].set_xlabel(cleaned_data.columns.values[3]);

f.axes[4].hist(cleaned_data.iloc[:,4],bins=20,color=colormap[4])
f.axes[4].set_xlabel(cleaned_data.columns.values[4]);

f.axes[5].hist(cleaned_data.iloc[:,5],bins=20,color=colormap[5])
f.axes[5].set_xlabel(cleaned_data.columns.values[5]);

fig = plt.gcf()
fig.set_size_inches(12, 12)
fig.set_size_inches(18.5, 10.5, forward=True)
```

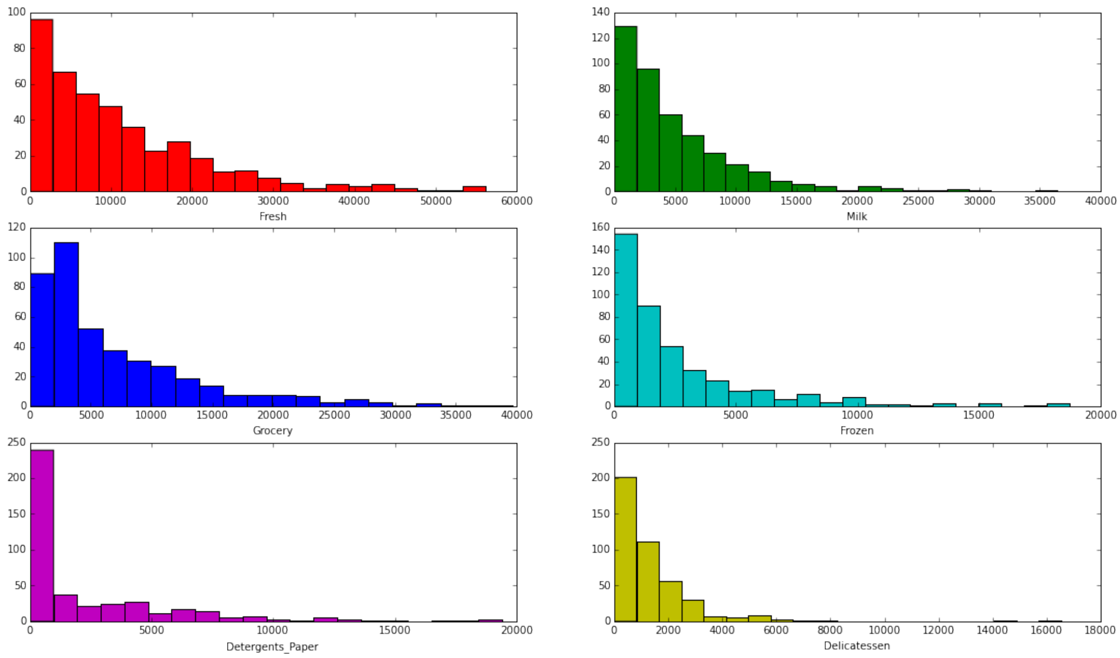
```
plt.show()

print "-> Plots of Fresh, Milk, Grocery, and Frozen seems to have some similarity in shape and
```

Cleaned, centered, and normalized data.



-> Fresh vs Milk, Milk vs Frozen, Grocery vs Frozen, and Detergent_paper vs Frozen all seem to have inve

----
Histogram of spending on specific product types.



-> Plots of Fresh, Milk, Grocery, and Frozen seems to have some similarity in shape and scale.

1.1 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:

Idea 1. Based on data spread, first PCA would be either fresh, or it could be combination of milk and groceries. Second PCA could include Frozen and Detergent_Paper, and Third PCA could be delicatessen.

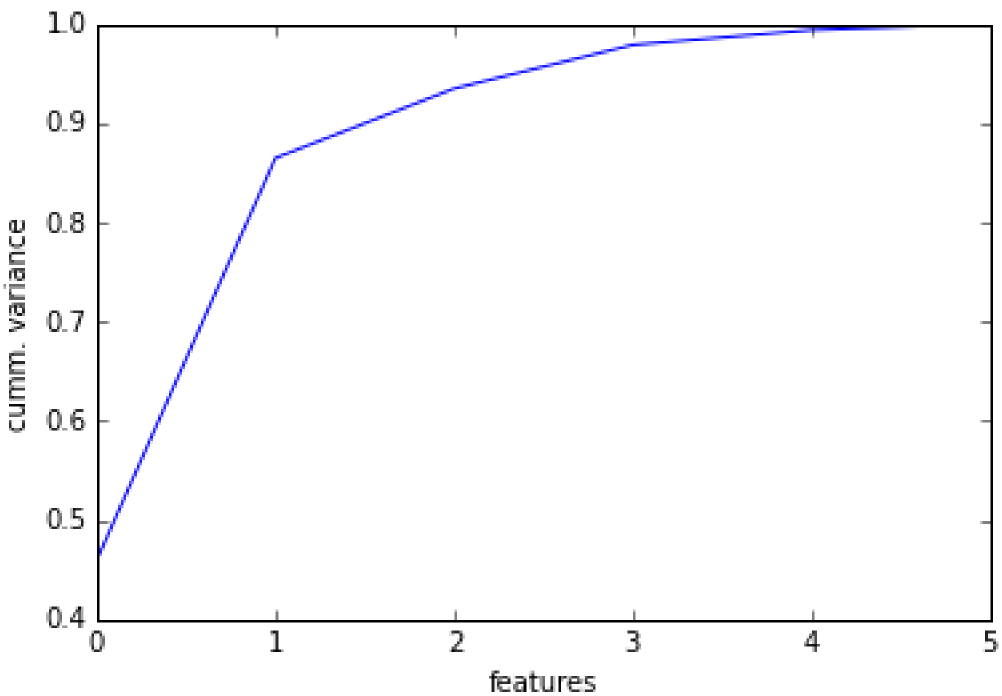
Idea 2. ICA could identify perishability as the differentiator in consumables / non-Delicatessen.

1.1.1 PCA

In [88]:
```
# TODO: Apply PCA with the same number of dimensions as variables in the dataset
# Using original data
from sklearn.decomposition import PCA
pca = PCA(n_components=num_features,whiten=True)
pca.fit(data)

# Print the components and the amount of variance in the data contained in each dimension
print data.columns.values
print pca.components_
print pca.explained_variance_ratio_


print "\n",'*'*5,"PCA on original data.",'*'*5,"\n"
pc_df=pd.DataFrame({"pca":pca.explained_variance_ratio_})
pc_cmf_df=np.cumsum(pc_df)
print '*'*5," Cumm variance:",'*'*5,"\n",pc_cmf_df
```

```
plt.plot(pc_cmf_df)
plt.ylabel('cumm. variance')
plt.xlabel('features')
plt.show()
```

```
['Fresh' 'Milk' 'Grocery' 'Frozen' 'Detergents_Paper' 'Delicatessen']
[[-0.97653685 -0.12118407 -0.06154039 -0.15236462  0.00705417 -0.06810471]
 [-0.11061386  0.51580216  0.76460638 -0.01872345  0.36535076  0.05707921]
 [-0.17855726  0.50988675 -0.27578088  0.71420037 -0.20440987  0.28321747]
 [-0.04187648 -0.64564047  0.37546049  0.64629232  0.14938013 -0.02039579]
 [ 0.015986    0.20323566 -0.1602915   0.22018612  0.20793016 -0.91707659]
 [-0.01576316  0.03349187  0.41093894 -0.01328898 -0.87128428 -0.26541687]]
[ 0.45961362  0.40517227  0.07003008  0.04402344  0.01502212  0.00613848]

***** PCA on original data. *****

*****  Cumm variance: *****
      pca
0 0.45961
1 0.86479
2 0.93482
3 0.97884
4 0.99386
5 1.00000
```

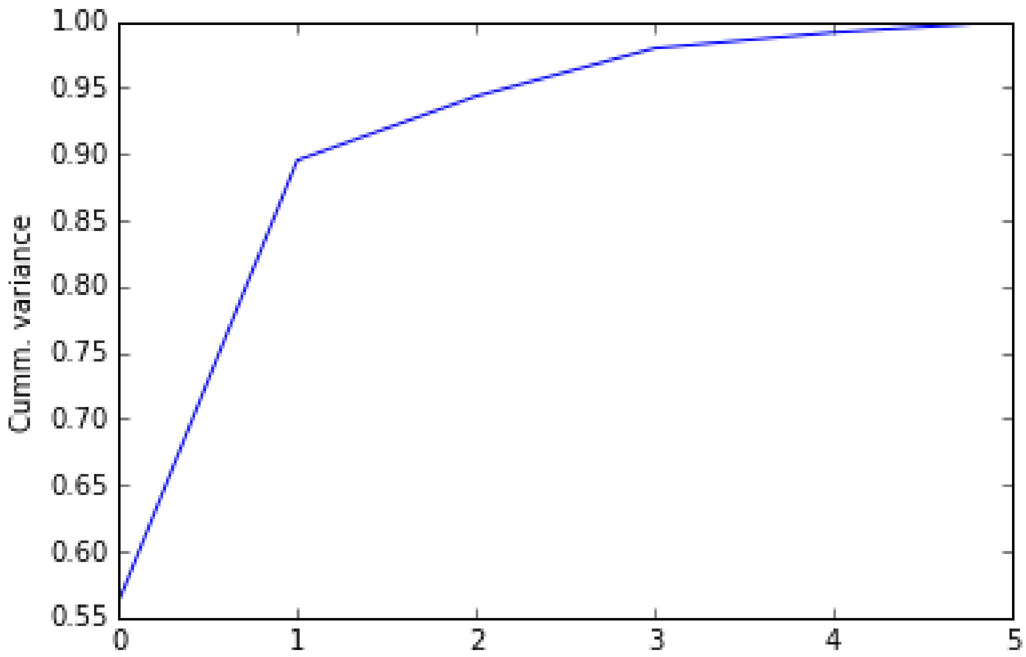



```
In [89]: # TODO: Apply PCA with the same number of dimensions as variables in the dataset
         # Using cleaned up data
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=num_features,whiten=True)
pca.fit(cleaned_data)

# Print the components and the amount of variance in the data contained in each dimension
print cleaned_data.columns.values
print pca.components_
print pca.explained_variance_ratio_


print "\n",'*'*5,"PCA on cleaned data.",'*'*5,"\n"
pc_df=pd.DataFrame({"pca":pca.explained_variance_ratio_})
pc_cmf_df=np.cumsum(pc_df)
print '*'*5," Cumm variance:",'*'*5,"\n",pc_cmf_df
plt.plot(pc_cmf_df)
plt.ylabel('Cumm. variance')
```

```
['Fresh' 'Milk' 'Grocery' 'Frozen' 'Detergents_Paper' 'Delicatessen']
[[-0.93484845  0.15350373  0.26484215 -0.09783119  0.14979567 -0.01854519]
 [ 0.33727454  0.49357837  0.72998125  0.00529907  0.32169172  0.0789937 ]
 [-0.09132697  0.61832533 -0.30822162  0.6719085  -0.2013167   0.14947626]
 [ 0.048662    0.57071797 -0.34542213 -0.72864576 -0.14113319  0.041873  ]
 [ 0.03907126  0.12678591 -0.31604449  0.08770818  0.64625958 -0.67614383]
 [-0.00831154 -0.09325834 -0.28772361 -0.01796047  0.62926724  0.71564593]]
[ 0.56148278  0.33367526  0.04836925  0.03623325  0.01188166  0.0083578 ]

***** PCA on cleaned data. *****

*****  Cumm variance: *****
      pca
0 0.56148
1 0.89516
2 0.94353
3 0.97976
4 0.99164
5 1.00000
```

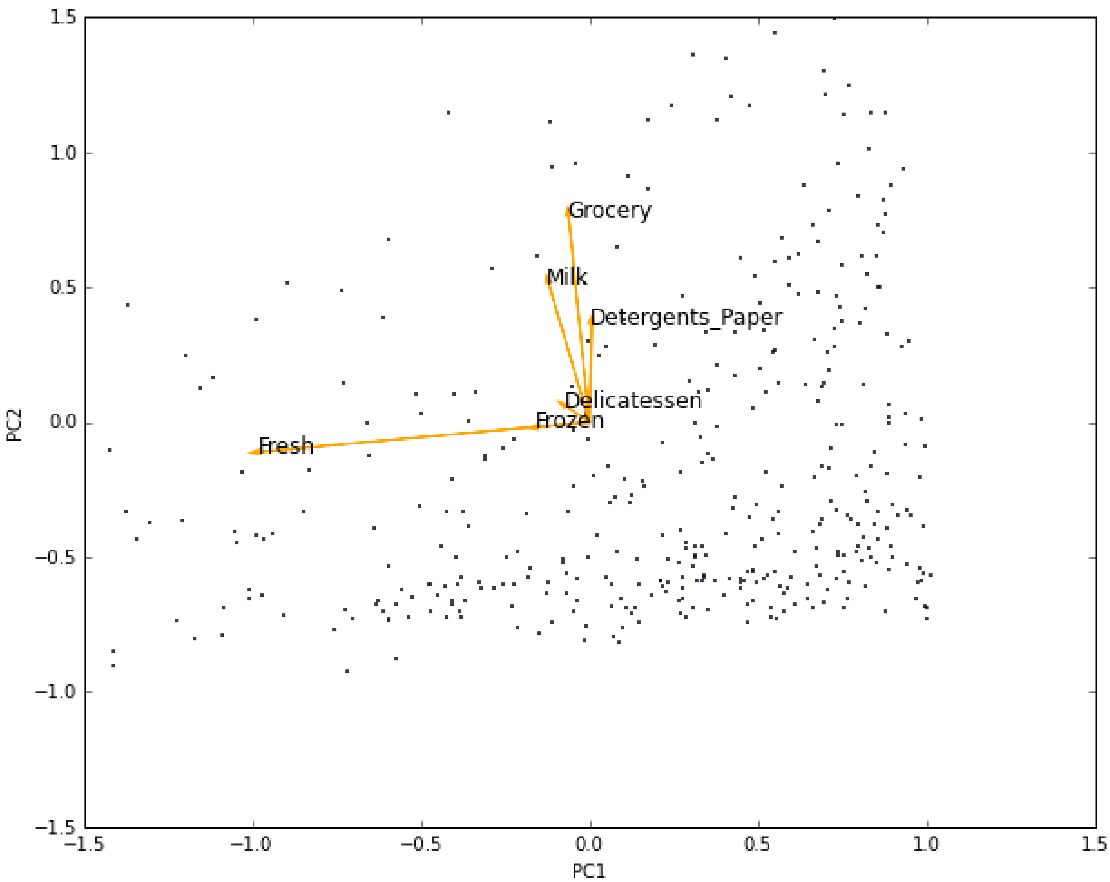
```
Out[89]: <matplotlib.text.Text at 0x106f72a50>
```




```
In [90]: ''' Following function has been taken from Udacity Forum:
         https://discussions.udacity.com/t/
         having-trouble-with-pca-and-ica-specifically-with-explaining-what-the-dimensions-mean/41890/12
         '''

         def biplot12(df):
             # Fit on 2 components
             pca = PCA(n_components=2, whiten=True).fit(df)

             # Plot transformed/projected data
             ax = pd.DataFrame(
                 pca.transform(df),
                 columns=['PC1', 'PC2']
             ).plot(kind='scatter', x='PC1', y='PC2', figsize=(10, 8), s=0.8)

             # Plot arrows and labels
             for i, (pc1, pc2) in enumerate(zip(pca.components_[0], pca.components_[1])):
                 ax.arrow(0, 0, pc1, pc2, width=0.001, fc='orange', ec='orange')
                 ax.annotate(df.columns[i], (pc1, pc2), size=12)

             return ax

         print '-'*100
         print "PC1 / PC2: Bi-plot of original data"
         print '-'*100
         ax = biplot12(data)
         # Play around with the ranges for scaling the plot
         ax.set_xlim([-1.5, 1.5])
         ax.set_ylim([-1.5, 1.5])
```

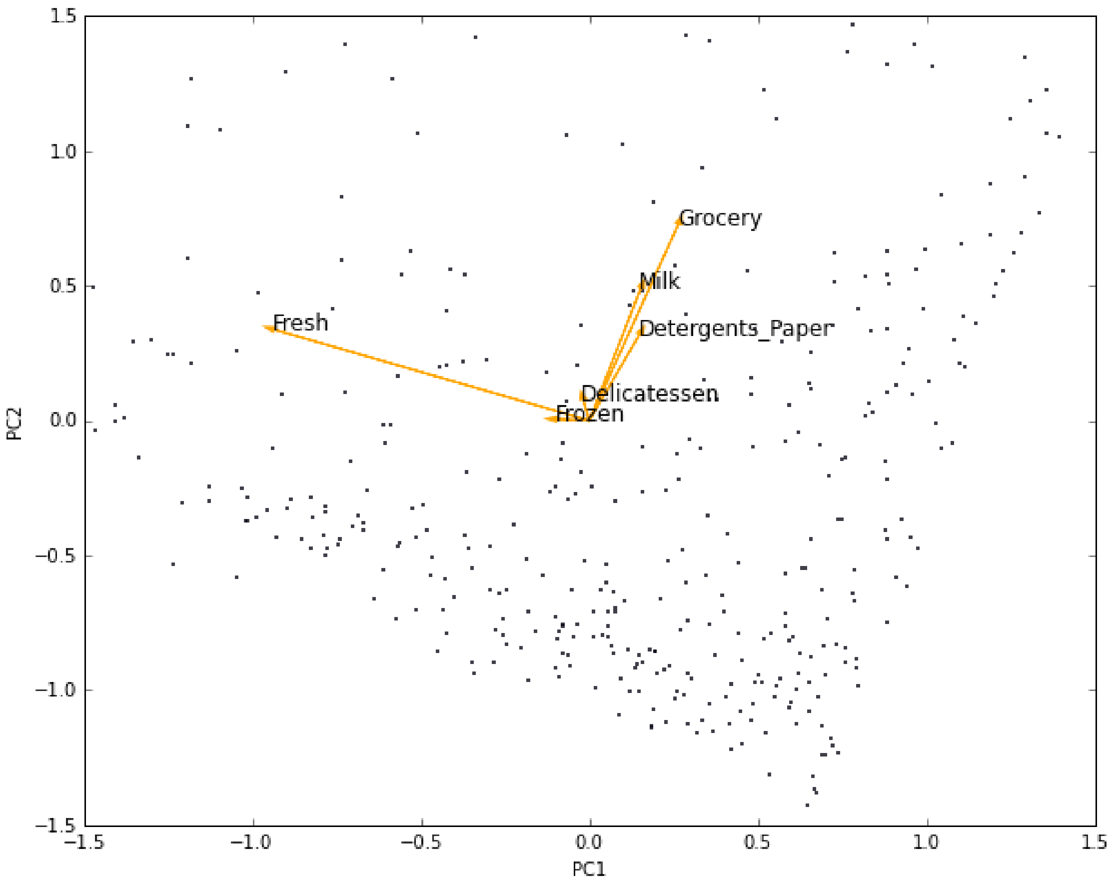
```
plt.show()


print '-'*100
print "PC1 / PC2: Bi-plot of cleaned data."
print '-'*100

ax = biplot12(cleaned_data)
# Play around with the ranges for scaling the plot
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
```

```
----------------------------------------------------------------------------------------------------
PC1 / PC2: Bi-plot of original data
----------------------------------------------------------------------------------------------------
```




```
----------------------------------------------------------------------------------------------------
PC1 / PC2: Bi-plot of cleaned data.
----------------------------------------------------------------------------------------------------
```

Out[90]: (-1.5, 1.5)

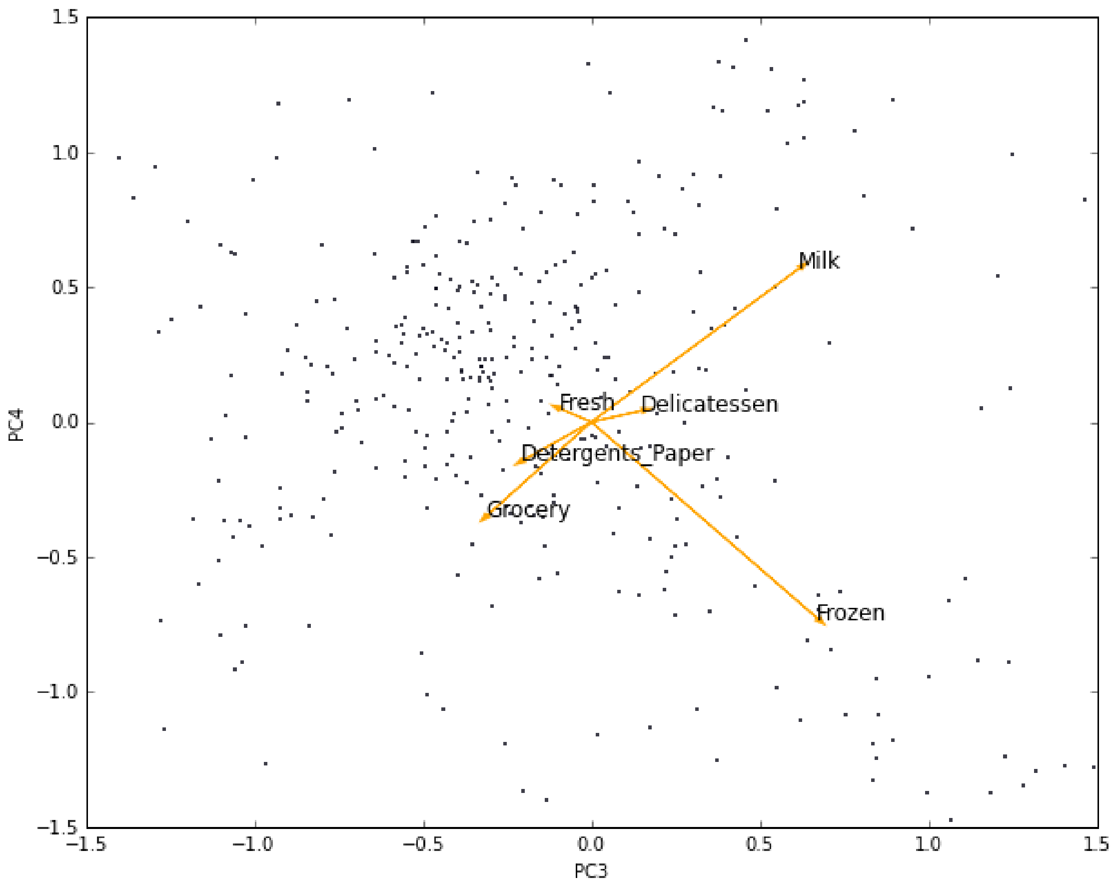



```
In [91]: # TODO: draw a 3-D plot ( or triplot :))
         def biplot34(df):
             # Fit on 2 components
             pca = PCA(n_components=4, whiten=True).fit(df)

             # Plot transformed/projected data
             ax = pd.DataFrame(
                 pca.transform(df),
                 columns=['PC1', 'PC2','PC3','PC4']
             ).plot(kind='scatter', x='PC3', y='PC4', figsize=(10, 8), s=0.8)

             # Plot arrows and labels
             for i, (pc3, pc4) in enumerate(zip(pca.components_[2], pca.components_[3])):
                 ax.arrow(0, 0, pc3, pc4, width=0.001, fc='orange', ec='orange')
                 ax.annotate(df.columns[i], (pc3, pc4), size=12)

             return ax

         print '-'*100
         print "PC3 / PC4: biplot of cleaned data"
         print '-'*100

         ax = biplot34(cleaned_data)
```

```
# Play around with the ranges for scaling the plot
ax.set_xlim([-1.5, 1.5])
ax.set_ylim([-1.5, 1.5])
```

```
--------------------------------------------------------------------------------
PC3 / PC4: biplot of cleaned data
--------------------------------------------------------------------------------
```

Out[91]: (-1.5, 1.5)



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 fast for first 2 dimentions, but then reduces slowly for remaining dimentions. Given the PCA variance graphs above, elbow is formed at 2nd PCA component, both for original data and scaled data. But since there are data points that have a some variance along multiple PCAs.

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

Answer: PCA here can be used in 2 ways here: 1.) to identify similar customers. 2.)To find similar features. But target here is to find similar customers, and first 2 primary components seem to cover a most of variance.

Then, first PCA dimention corresponds to a segment that spends mostly on Fresh and Frozen products.

Second PCA corresponds that spend mostly on Grocery, and significantly on Milk and Detergent_Paper in that order.

We can use this information in many ways: 1.) To transform the data along these 2 PCA, and then find cluster of users using transformed data. But this may not be good approach, since PCA-transformed data might loose some information which could impact be useful for un-biased clustering.

2.) To do clustering independently, and then compare the results with those from PCA, to see if both these results are convergent of divergent.

3.) We can use the results of PCA further components for supervised learning analysis - regression or classification.

4.) We could also use K=2 and K=3 for k-mens clustering. Although value of K could depend on elbow in sum-of-square vs k plot.

1.1.2 ICA

In [92]:
```
# TODO: Fit an ICA model to the data
# Note: Adjust the data to have center at the origin first!
from sklearn.decomposition import FastICA
from sklearn import preprocessing

scaled_data=data.copy(deep=True)

#from sklearn import preprocessing
scaled_data[['Fresh','Milk','Grocery','Frozen','Detergents_Paper','Delicatessen']] = scaled_da

ica = FastICA(whiten=True,random_state=0)
transformed_data=ica.fit_transform(scaled_data)
#
# Print the independent components

print "\n"
print scaled_data.columns.values
print ica.components_

print "\n"
print preprocessing.StandardScaler().fit_transform(ica.components_)
#print "\n"
#print ica.mixing_
```

```
['Fresh' 'Milk' 'Grocery' 'Frozen' 'Detergents_Paper' 'Delicatessen']
[[ 0.00259749 -0.01304261  0.06424104  0.00176503 -0.00789576 -0.00472804]
 [ 0.0036662  -0.01675528 -0.11301178  0.00711535  0.13424464  0.01592772]
 [-0.00189529 -0.07279239  0.05444162  0.00183269 -0.01463357  0.01719393]
 [-0.05024607  0.00639506  0.00647498  0.00325086 -0.0104146   0.00291214]
 [-0.00485887 -0.00161266 -0.00552872 -0.00242502  0.0023066   0.05090388]
 [ 0.01091921  0.00104603 -0.00729797 -0.05405923  0.00256987  0.01686439]]


[[ 0.45910575  0.11611315  1.11498351  0.41769164 -0.48743545 -1.21961489]
 [ 0.51224275 -0.02365303 -1.95603618  0.67015972  2.21983183 -0.03356863]
 [ 0.23572202 -2.13321102  0.9452022   0.42088438 -0.6157666   0.03913673]
 [-2.16830789  0.84785851  0.11414921  0.48780439 -0.53541034 -0.78091823]
 [ 0.08837092  0.54640212 -0.09382264  0.21997389 -0.29311692  1.97475038]
 [ 0.87286647  0.64649027 -0.12447609 -2.21651401 -0.28810251  0.02021463]]
```

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:

['Fresh' 'Milk' 'Grocery' 'Frozen' 'Detergents_Paper' 'Delicatessen']

A. [0.45910575 0.11611315 1.11498351 0.41769164 -0.48743545 -1.21961489] – Delicacy, Grocery, small qty of everything.

B. [0.51224275 -0.02365303 -1.95603618 0.67015972 2.21983183 -0.03356863] – Detergents_Paper, Grocery,

C. [0.23572202 -2.13321102 0.9452022 0.42088438 -0.6157666 0.03913673] – Milk and Grocery, small qty of everything except delicacy.

D. [-2.16830789 0.84785851 0.11414921 0.48780439 -0.53541034 -0.78091823] – Fresh, Milk, Delicacy

E. [0.08837092 0.54640212 -0.09382264 0.21997389 -0.29311692 1.97475038] – Delicacy, Milk

F. [0.87286647 0.64649027 -0.12447609 -2.21651401 -0.28810251 0.02021463] – Frozen, fresh, some milk

-> Store of type A and C buy primarily Groceries, but also little bit of everything. -> Store B buys Detergent, Paper, and Groceries. -> D,E, and F dont buy much grocery, and seem to be specialized store, like bakery or chocolatier. -> So this could mean the purchasing habbits of consumers. For example, consumers purchase milk from different sources. -> There seems to be 3 types of stores. Those buying Grocery and Detergent, those buying milk, fresh, delicacies, and those buy everything.

1.2 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.

1.2.1 Choose a Cluster Type

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

1. k-means is intuitive, and fast.
2. k-means can be computed and stored, for later application. This would allow quickly finding similarity.
3. But k-means is strict.
4. GMM is more soft, and allows more realistic / fuzzy interpretation of distributions.
5. GMM is fast, but uses all available features. SO it is important to reduce features before applying GMM.

6) Below is some starter code to help you visualize some cluster data. The visualization is based on this demo from the sklearn documentation.

In [93]:
```
# Import clustering modules
from sklearn.cluster import KMeans
from sklearn.mixture import GMM
```

In [94]:
```
# TODO: First we reduce the data to two dimensions using PCA to capture variation

pca = PCA(n_components=2, whiten=True)

reduced_data = pca.fit_transform(cleaned_data)
print reduced_data[:10]  # print upto 10 elements
```

```
[[-0.02449642  0.35143511]
 [ 0.49258938  0.33859559]
 [ 0.48026585  0.15697005]
 [-0.35621581 -0.47218347]
 [-0.98151052  0.47176439]
 [ 0.15642939 -0.09731991]
 [-0.07681722 -0.0824213 ]
 [ 0.37789019  0.07512968]
 [ 0.41110717 -0.41820212]
 [ 0.88192697  1.32444758]]
```

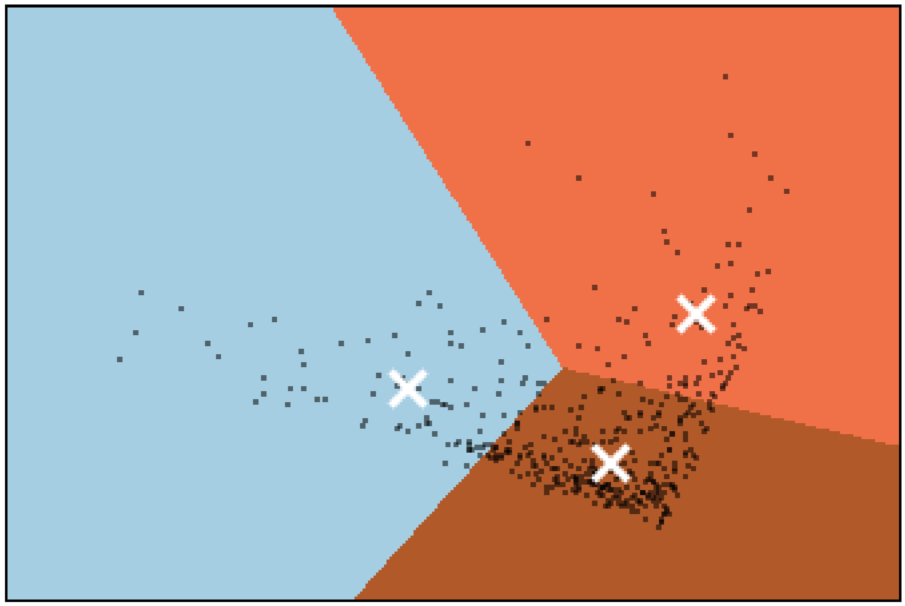
```
In [95]: # TODO: Implement your clustering algorithm here, and fit it to the reduced data for visualiza
         # The visualizer below assumes your clustering object is named 'clusters'
         from scipy.spatial.distance import cdist

         clusters = KMeans(init='k-means++', n_clusters=3, n_init=5).fit(reduced_data)
         print clusters

         # Plot the decision boundary by building a mesh grid to populate a graph.
         x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
         y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
         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, y_max, hy))

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

```
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=3, n_init=5,
    n_jobs=1, precompute_distances='auto', random_state=None, tol=0.0001,
    verbose=0)
```

```
In [96]: # TODO: Find the centroids for KMeans or the cluster means for GMM

         centroids = clusters.cluster_centers_
         print centroids
```

```
[[-1.56091044  0.46290062]
 [ 0.96914205  1.50025771]
 [ 0.21062931 -0.55903733]]
```

```
In [97]: # Put the result into a color plot
         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=2)
         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-reduced data)\n'
                   'Centroids are marked with white cross')
         plt.xlim(x_min, x_max)
         plt.ylim(y_min, y_max)
         plt.xticks(())
         plt.yticks(())
         plt.show()

         print cleaned_data.columns.values
         print centroids
         print pca.inverse_transform(centroids)
```



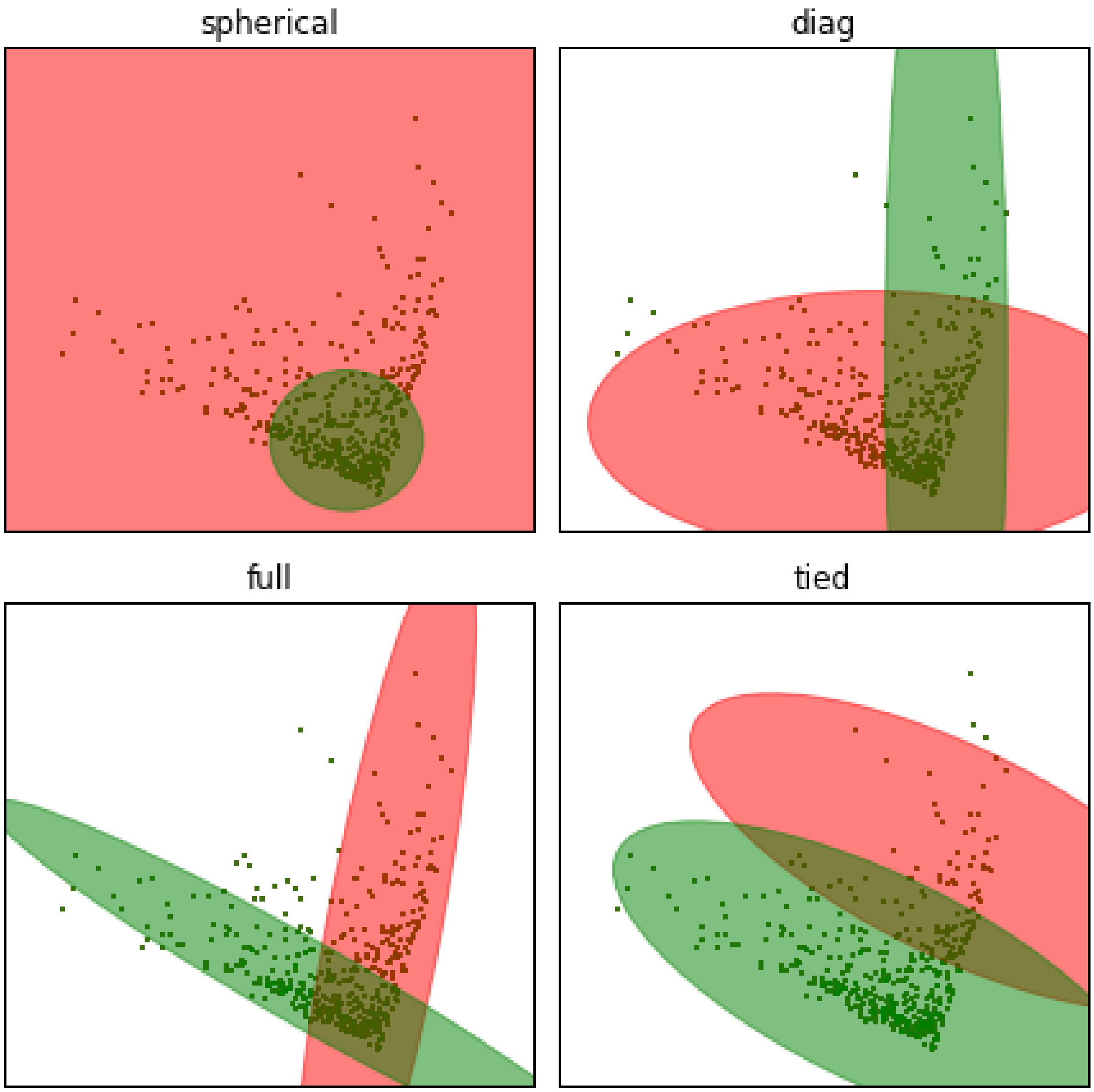

```
['Fresh' 'Milk' 'Grocery' 'Frozen' 'Detergents_Paper' 'Delicatessen']
[[-1.56091044  0.46290062]
 [ 0.96914205  1.50025771]
 [ 0.21062931 -0.55903733]]
[[ 28765.34363818   4461.37504269   5480.99960792   4434.28007521
    1216.65602684   2007.14909155]
 [  5525.33801205  13146.09913836  19386.44803806   1736.19490052
    8272.88602236   2187.37115107]
 [  7451.44491529   3164.84605956   4306.47936921   2465.87828229
    1349.06332628    952.61403359]]
```

```
In [98]: from matplotlib.colors import LogNorm
         import matplotlib as mpl

         n_classes=2
         def make_ellipses(gmm, ax):
             for n, color in enumerate('rg'):
                 v, w = np.linalg.eigh(gmm._get_covars()[n][:2, :2])
                 u = w[0] / np.linalg.norm(w[0])
                 angle = np.arctan2(u[1], u[0])
                 angle = 180 * angle / np.pi  # convert to degrees
                 v *= 9
                 ell = mpl.patches.Ellipse(gmm.means_[n, :2], v[0], v[1],
                                           180 + angle, color=color)
                 ell.set_clip_box(ax.bbox)
                 ell.set_alpha(0.5)
                 ax.add_artist(ell)
```

```
classifiers = dict((covar_type, GMM(n_components=2,
                    covariance_type=covar_type, init_params='wc', n_iter=20))
                   for covar_type in ['spherical', 'diag', 'tied', 'full'])
#clf=GMM(n_components=3, covariance_type='full').fit(reduced_data)

n_classifiers = len(classifiers)

plt.figure(figsize=(3 * n_classifiers / 2, 6))
plt.subplots_adjust(bottom=.01, top=0.95, hspace=.15, wspace=.05,
                    left=.01, right=.99)

for index, (name, classifier) in enumerate(classifiers.items()):
    # Since we have class labels for the training data, we can
    # initialize the GMM parameters in a supervised manner.
    # classifier.means_ = np.array([X_train[y_train == i].mean(axis=0)
    #                               for i in xrange(n_classes)])

    # Train the other parameters using the EM algorithm.
    classifier.fit(reduced_data)

    h = plt.subplot(2, n_classifiers / 2, index + 1)
    make_ellipses(classifier, h)

    for n, color in enumerate('rg'):
        data = reduced_data
        plt.scatter(data[:, 0], data[:, 1], 0.8, color=color)




    plt.xticks(())
    plt.yticks(())
    plt.title(name)
    print cleaned_data.columns.values
    print classifier.sample
    print  pca.inverse_transform(classifier.means_)

plt.legend(loc='lower right', prop=dict(size=12))


plt.show()
```

```
['Fresh' 'Milk' 'Grocery' 'Frozen' 'Detergents_Paper' 'Delicatessen']
<bound method GMM.sample of GMM(covariance_type='spherical', init_params='wc', min_covar=0.001,
  n_components=2, n_init=1, n_iter=20, params='wmc', random_state=None,
  thresh=None, tol=0.001, verbose=0)>
[[ 16649.08654428   8222.28641302  11577.38156102   3069.52928174
    4370.54192259   2000.1008162 ]
 [  7858.39938446   3247.55825763   4416.06139148   2500.29482239
    1386.20390951    980.38215285]]
['Fresh' 'Milk' 'Grocery' 'Frozen' 'Detergents_Paper' 'Delicatessen']
<bound method GMM.sample of GMM(covariance_type='diag', init_params='wc', min_covar=0.001, n_components=2
  n_init=1, n_iter=20, params='wmc', random_state=None, thresh=None,
```

```
  tol=0.001, verbose=0)>
[[ 12992.53507829   3773.77747783   5020.6603329    2962.75900897
    1500.71903364   1262.57444695]
 [  4796.73931178  10308.36558469  15145.53444246   1821.54538804
    6365.46239208   1783.40473597]]
['Fresh' 'Milk' 'Grocery' 'Frozen' 'Detergents_Paper' 'Delicatessen']
<bound method GMM.sample of GMM(covariance_type='full', init_params='wc', min_covar=0.001, n_components=2
  n_init=1, n_iter=20, params='wmc', random_state=None, thresh=None,
  tol=0.001, verbose=0)>
[[  5217.14891445   8890.91236022  12998.66249246   1939.22189258
    5375.17175343   1614.14527774]
 [ 14633.90055509   3072.92806308   3907.0170284    3158.0958139
     942.48785762   1238.32459163]]
['Fresh' 'Milk' 'Grocery' 'Frozen' 'Detergents_Paper' 'Delicatessen']
<bound method GMM.sample of GMM(covariance_type='tied', init_params='wc', min_covar=0.001, n_components=2
  n_init=1, n_iter=20, params='wmc', random_state=None, thresh=None,
  tol=0.001, verbose=0)>
[[  6024.280768    14741.19838442  21767.02792998   1696.77215436
    9340.7777129    2418.14870719]
 [ 11974.84778624   3831.29432371   5144.16935963   2862.24701105
    1588.78015829   1227.95203683]]
```



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

Answer: We will consider the "full" one correctly explains the soft clustering. Based inverse transformming the centroids from k-means: [0.1663451 -0.5683769][0.95776694 1.39994104] [-1.64821881 0.52313298]] [[7883.6470263 3050.04393497 4118.11897994 2513.50309189 1249.80730062 955.41459363][5354.00667651 12703.41122854 18726.96082769 1743.9919777 7978.08628612 2121.9609403] [29844.22811199 4566.91163844 5600.46624755 4531.73759903 1237.26861427 2065.78464029]]

, and also the inverse transform of means values in GMMS, we have following observations:

Centroids in k-means coincide with means of 3 clusters of GMMs.

Cluster 1 (overlap area) has a consumption of everything. Cluster 2 (green)has highest consumption of Milk, Grocery, and Detergents. Cluster 3 (pink) mostly consumes Fresh produce.

1.2.2 Conclusions

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

Answer:

Tried multiple number of clusters for GMM and k-means, and 3 seemed best for k-means and 2 for GMMs. Although both point to 3 types of clusters, including the overlap in GMMs.

All 3 techniques gave different information, and collating all 3 techniques gives confidence in solution. PCA gave more direct info on primary conponents, while ICA and Clustering gave insight into source and unlabeled similarity in data.

But GMMs seem to be best suited for such a problem, due to overlapping of types. Both PCA and k-means showed thst there is no clear separation between the clusters in data.

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

Answer: 1. Company could test and record results using additional feature set, like time of delivery / sales, combine certain features like Fresh and Frozen, or by separating detergent and Paper.

2. Company can also breakup sales for each feature by week / day.

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

Answer: ‘” 1. We can use day / week data to help identify cyclical patterns 2. Given data can be used to predict sames of one type of product for a customer given its sales of other types of products. 3. For each cluster, we can make separate prediction and classification models. ‘”