Machine Learning Engineer Nanodegree

Unsupervised Learning

Project: Creating Customer Segments

Welcome to the third project of the Machine Learning Engineer Nanodegree! In this notebook, some template code has already been provided for you, and it will be your job to implement the additional functionality necessary to successfully complete this project. Sections that begin with **'Implementation'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section and the specifics of the implementation are marked in the code block with a `'TODO'` statement. Please be sure to read the instructions carefully!

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

Getting Started ¶

In this project, you will analyze a dataset containing data on various customers' annual spending amounts (reported in *monetary units*) of diverse product categories for internal structure. One goal of this project is to best describe the variation in the different types of customers that a wholesale distributor interacts with. Doing so would equip the distributor with insight into how to best structure their delivery service to meet the needs of each customer.

The dataset for this project can be found on the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Wholesale+customers). For the purposes of this project, the features `'Channel'` and `'Region'` will be excluded in the analysis — with focus instead on the six product categories recorded for customers.

Run the code block below to load the wholesale customers dataset, along with a few of the necessary Python libraries required for this project. You will know the dataset loaded successfully if the size of the dataset is reported.

In [1]:
```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
from IPython.display import display # Allows the use of display() for Da
taFrames

# Import supplementary visualizations code visuals.py
import visuals as vs

# Pretty display for notebooks
%matplotlib inline

# Load the wholesale customers dataset
try:
    data = pd.read_csv("customers.csv")
    data.drop(['Region', 'Channel'], axis = 1, inplace = True)
    print("Wholesale customers dataset has {} samples with {} features e
ach.".format(*data.shape))
except:
    print("Dataset could not be loaded. Is the dataset missing?")
```

```
Wholesale customers dataset has 440 samples with 6 features each.
```

Data Exploration

In this section, you will begin exploring the data through visualizations and code to understand how each feature is related to the others. You will observe a statistical description of the dataset, consider the relevance of each feature, and select a few sample data points from the dataset which you will track through the course of this project.

Run the code block below to observe a statistical description of the dataset. Note that the dataset is composed of six important product categories: **'Fresh'**, **'Milk'**, **'Grocery'**, **'Frozen'**, **'Detergents_Paper'**, and **'Delicatessen'**. Consider what each category represents in terms of products you could purchase.

In [2]:
```
# Display a description of the dataset
display(data.describe())
```

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

Implementation: Selecting Samples

To get a better understanding of the customers and how their data will transform through the analysis, it would be best to select a few sample data points and explore them in more detail. In the code block below, add **three** indices of your choice to the `indices` list which will represent the customers to track. It is suggested to try different sets of samples until you obtain customers that vary significantly from one another.

In [3]:
```
# TODO: Select three indices of your choice you wish to sample from the
 dataset
indices = [ 3, 44, 61 ]

# Create a DataFrame of the chosen samples
samples = pd.DataFrame(data.loc[indices], columns = data.keys()).reset_i
ndex(drop = True)
print("Chosen samples of wholesale customers dataset:")
display(samples)
```

```
Chosen samples of wholesale customers dataset:
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	13265	1196	4221	6404	507	1788
1	9670	7027	10471	541	4618	65
2	35942	38369	59598	3254	26701	2017

Question 1

Consider the total purchase cost of each product category and the statistical description of the dataset above for your sample customers.

- What kind of establishment (customer) could each of the three samples you've chosen represent?

Hint: Examples of establishments include places like markets, cafes, delis, wholesale retailers, among many others. Avoid using names for establishments, such as saying "*McDonalds*" when describing a sample customer as a restaurant. You can use the mean values for reference to compare your samples with. The mean values are as follows:

- Fresh: 12000.2977
- Milk: 5796.2
- Grocery: 3071.9
- Detergents_paper: 2881.4
- Delicatessen: 1524.8

Knowing this, how do your samples compare? Does that help in driving your insight into what kind of establishments they might be?

Answer:

Customer #3 can be taken as a restaurant or deli that specializes in Fresh foods or Frozen foods since the numbers for them are way higher than mean valuesof respective categories. Also the other indication is that Detergents & Paper purchase is low indicating that they must be serving in house to their customers.

Customer #44 can be cosidered a groccery store which also serves fresh food since both the categories purchases are way higher than average.

Customer #61 can be considered as a market or wholesale retailer since their purchase numbers are significantly higher than the mean values for most of the categories. The close numbers are for Frozen and Delicatessen as compared to the others.

Implementation: Feature Relevance

One interesting thought to consider is if one (or more) of the six product categories is actually relevant for understanding customer purchasing. That is to say, is it possible to determine whether customers purchasing some amount of one category of products will necessarily purchase some proportional amount of another category of products? We can make this determination quite easily by training a supervised regression learner on a subset of the data with one feature removed, and then score how well that model can predict the removed feature.

In the code block below, you will need to implement the following:

- Assign `new_data` a copy of the data by removing a feature of your choice using the `DataFrame.drop` function.
- Use `sklearn.cross_validation.train_test_split` to split the dataset into training and testing sets.
  - Use the removed feature as your target label. Set a `test_size` of `0.25` and set a `random_state`.
- Import a decision tree regressor, set a `random_state`, and fit the learner to the training data.
- Report the prediction score of the testing set using the regressor's `score` function.

In [4]:
```
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeRegressor

# TODO: Make a copy of the DataFrame, using the 'drop' function to drop
 the given feature
new_data = data.drop('Delicatessen', axis=1)

# TODO: Split the data into training and testing sets(0.25) using the gi
ven feature as the target
# Set a random state.
X_train, X_test, y_train, y_test = train_test_split(new_data, data['Deli
catessen'], test_size = 0.25, random_state=42)

# TODO: Create a decision tree regressor and fit it to the training set
regressor =  DecisionTreeRegressor(random_state=42).fit(X_train, y_train
)

# TODO: Report the score of the prediction using the testing set
score = regressor.score(X_test, y_test)

print(score)
```

```
-2.2547115372

/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor
of the model_selection module into which all the refactored classes and
functions are moved. Also note that the interface of the new CV iterato
rs are different from that of this module. This module will be removed
in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Question 2

- Which feature did you attempt to predict?
- What was the reported prediction score?
- Is this feature necessary for identifying customers' spending habits?

Hint: The coefficient of determination, `R^2` , is scored between 0 and 1, with 1 being a perfect fit. A negative `R^2` implies the model fails to fit the data. If you get a low score for a particular feature, that lends us to beleive that that feature point is hard to predict using the other features, thereby making it an important feature to consider when considering relevance.

Answer: I have tried to predict the 'Delicatessen' feature. The score for the regressor is -2.2547115372

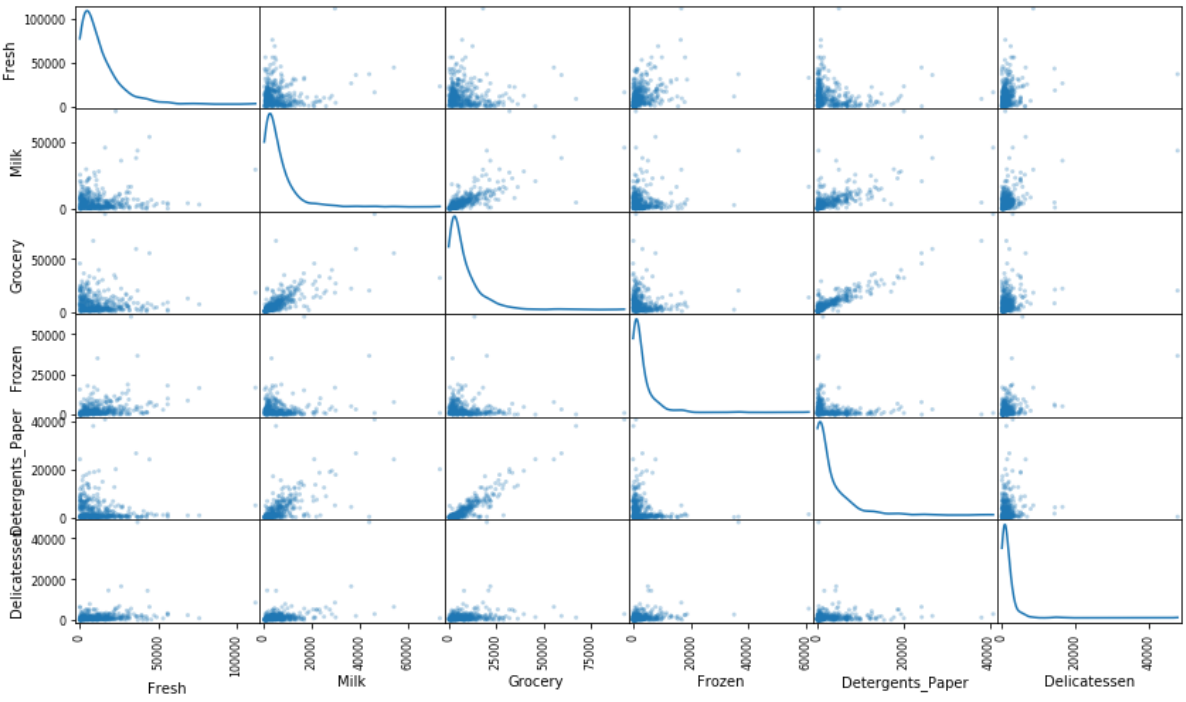
I think that this feature is really important or necessary for predcting the spending habits of customers since the model when used without that particular feature, fails to fit the data. The model's performance is very bad

Visualize Feature Distributions

To get a better understanding of the dataset, we can construct a scatter matrix of each of the six product features present in the data. If you found that the feature you attempted to predict above is relevant for identifying a specific customer, then the scatter matrix below may not show any correlation between that feature and the others. Conversely, if you believe that feature is not relevant for identifying a specific customer, the scatter matrix might show a correlation between that feature and another feature in the data. Run the code block below to produce a scatter matrix.

In [5]:
```
# Produce a scatter matrix for each pair of features in the data
pd.scatter_matrix(data, alpha = 0.3, figsize = (14,8), diagonal = 'kde'
);
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:2: FutureW
arning: pandas.scatter_matrix is deprecated, use pandas.plotting.scatte
r_matrix instead
```



Question 3

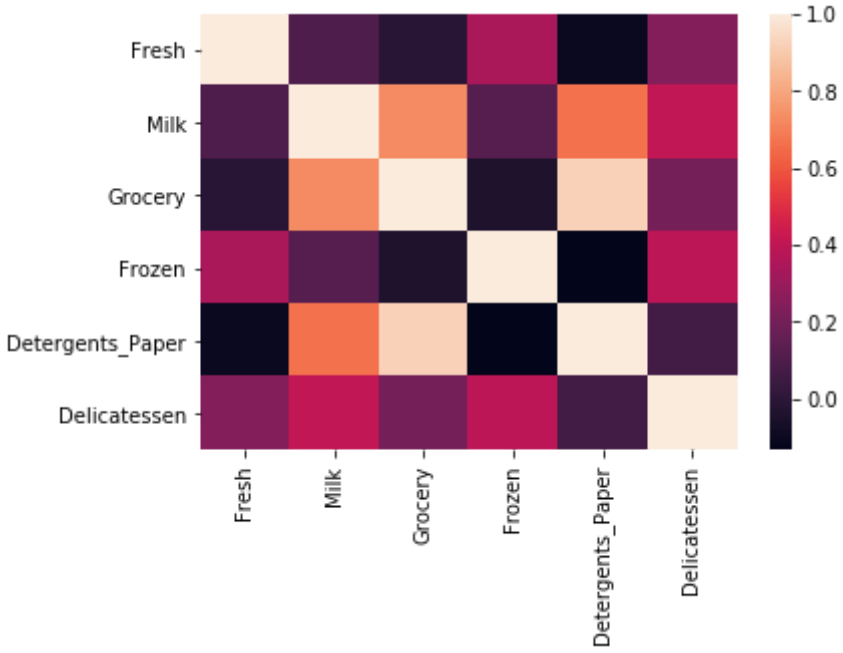
- Using the scatter matrix as a reference, discuss the distribution of the dataset, specifically talk about the normality, outliers, large number of data points near 0 among others. If you need to sepearate out some of the plots individually to further accentuate your point, you may do so as well.
- Are there any pairs of features which exhibit some degree of correlation?
- Does this confirm or deny your suspicions about the relevance of the feature you attempted to predict?
- How is the data for those features distributed?

Hint: Is the data normally distributed? Where do most of the data points lie? You can use corr() (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.corr.html) to get the feature correlations and then visualize them using a heatmap (http://seaborn.pydata.org/generated/seaborn.heatmap.html) (the data that would be fed into the heatmap would be the correlation values, for eg: `data.corr()`) to gain further insight.

In [6]:
```
from seaborn import heatmap

heatmap(data.corr())
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff8583d6160>



Answer:

- The majority of the data points in the above given scatterplot, are spread out in the 3rd quadrant (i.e. the left-bottom corner) and we can say that at only some of the points have occupied more than 75% area of each of the plots. Most of the points are scattered around the 0's on both axes.

- From the heat map we can infer that Grocery and Detergents/Paper seem to have a strong positive correlation of around 0.8-0.9. Similarly we can also pinpoint the black-coloured boxes and can infer that Grocery or Detergent/Paper seem to share almost no correlation with Fresh and Frozen, apart from that, Fresh and Frozen are also seen to not share any considerable correlations. Grocery and Milk also share a decent correlation of around 0.6-0.7.

- These results deny my suspicions about the importance of Delicatessen. The correlations are out of sync and that proves our notion of Delicatessen to be important while predicting the other 5 features.

- The data for the Delicatessen (X-axis) when put against the other 5 features (Y-axis), appears to be distributed along the vertical axis, indicating that the spending for Delicatessen is high or low, there doesn't seem to be a correlation with the Y-axis and this can be a result of the Standard Deviation of the data.

Data Preprocessing

In this section, you will preprocess the data to create a better representation of customers by performing a scaling on the data and detecting (and optionally removing) outliers. Preprocessing data is often times a critical step in assuring that results you obtain from your analysis are significant and meaningful.

Implementation: Feature Scaling

If data is not normally distributed, especially if the mean and median vary significantly (indicating a large skew), it is most [often appropriate (http://econbrowser.com/archives/2014/02/use-of-logarithms-in-economics)](http://econbrowser.com/archives/2014/02/use-of-logarithms-in-economics) to apply a non-linear scaling — particularly for financial data. One way to achieve this scaling is by using a [Box-Cox test (http://scipy.github.io/devdocs/generated/scipy.stats.boxcox.html)](http://scipy.github.io/devdocs/generated/scipy.stats.boxcox.html), which calculates the best power transformation of the data that reduces skewness. A simpler approach which can work in most cases would be applying the natural logarithm.

In the code block below, you will need to implement the following:

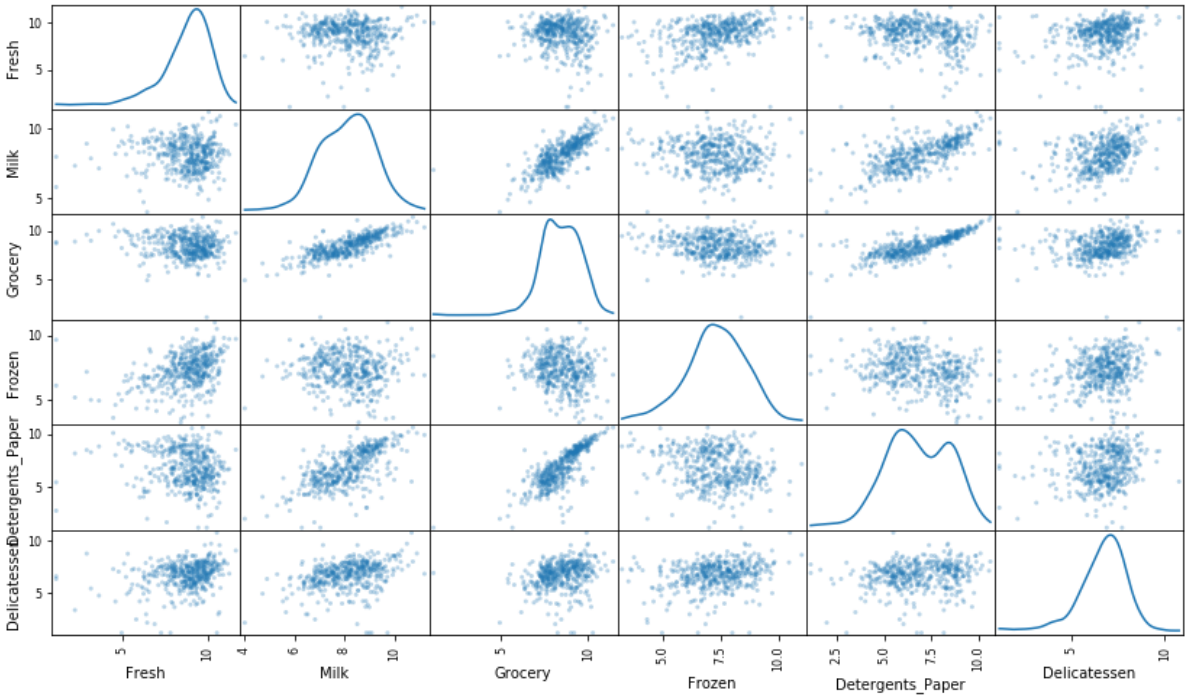
- Assign a copy of the data to `log_data` after applying logarithmic scaling. Use the `np.log` function for this.
- Assign a copy of the sample data to `log_samples` after applying logarithmic scaling. Again, use `np.log`.

In [7]:
```
# TODO: Scale the data using the natural logarithm
log_data = np.log(data)

# TODO: Scale the sample data using the natural logarithm
log_samples = np.log(samples)

# Produce a scatter matrix for each pair of newly-transformed features
pd.scatter_matrix(log_data, alpha = 0.3, figsize = (14,8), diagonal = 'kde');
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:8: FutureWarning: pandas.scatter_matrix is deprecated, use pandas.plotting.scatter_matrix instead
```



Observation

After applying a natural logarithm scaling to the data, the distribution of each feature should appear much more normal. For any pairs of features you may have identified earlier as being correlated, observe here whether that correlation is still present (and whether it is now stronger or weaker than before).

Run the code below to see how the sample data has changed after having the natural logarithm applied to it.

In [8]:
```
# Display the log-transformed sample data
display(log_samples)
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	9.492884	7.086738	8.347827	8.764678	6.228511	7.488853
1	9.176784	8.857515	9.256365	6.293419	8.437717	4.174387
2	10.489662	10.555005	10.995377	8.087640	10.192456	7.609367

Implementation: Outlier Detection

Detecting outliers in the data is extremely important in the data preprocessing step of any analysis. The presence of outliers can often skew results which take into consideration these data points. There are many "rules of thumb" for what constitutes an outlier in a dataset. Here, we will use [Tukey's Method for identfying outliers (https://www.kdnuggets.com/2017/01/3-methods-deal-outliers.html)](): An *outlier step* is calculated as 1.5 times the interquartile range (IQR). A data point with a feature that is beyond an outlier step outside of the IQR for that feature is considered abnormal.

In the code block below, you will need to implement the following:

- Assign the value of the 25th percentile for the given feature to `Q1`. Use `np.percentile` for this.
- Assign the value of the 75th percentile for the given feature to `Q3`. Again, use `np.percentile`.
- Assign the calculation of an outlier step for the given feature to `step`.
- Optionally remove data points from the dataset by adding indices to the `outliers` list.

NOTE: If you choose to remove any outliers, ensure that the sample data does not contain any of these points! Once you have performed this implementation, the dataset will be stored in the variable `good_data`.

In [47]:
```
outliers = []
repeat_outliers = []
# For each feature find the data points with extreme high or low values
for feature in log_data.keys():

    # TODO: Calculate Q1 (25th percentile of the data) for the given fea
ture
    Q1 = np.percentile(log_data[feature], 25)

    # TODO: Calculate Q3 (75th percentile of the data) for the given fea
ture
    Q3 = np.percentile(log_data[feature], 75)

    # TODO: Use the interquartile range to calculate an outlier step (1.
5 times the interquartile range)
    step = (Q3 - Q1)*1.5

    # Display the outliers
    print("Data points considered outliers for the feature '{}':".format
(feature))
    results =log_data[~((log_data[feature] >= Q1 - step) & (log_data[fea
ture] <= Q3 + step))]
    display(results)

    for i in results.index:
        if i not in outliers:
            outliers.append(i)
        elif i not in repeat_outliers:
            repeat_outliers.append(i)

print("Number of outliers: ", len(outliers))
print("Outliers: ", outliers)
print(" ")
print("Number of repeat outliers: ", len(repeat_outliers))
print("Repeat outliers: ", repeat_outliers)



# Remove the outliers, if any were specified
good_data = log_data.drop(log_data.index[outliers]).reset_index(drop = T
rue)


print(log_data.shape , good_data.shape)
```

Data points considered outliers for the feature 'Fresh':

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
65	4.442651	9.950323	10.732651	3.583519	10.095388	7.260523
66	2.197225	7.335634	8.911530	5.164786	8.151333	3.295837
81	5.389072	9.163249	9.575192	5.645447	8.964184	5.049856
95	1.098612	7.979339	8.740657	6.086775	5.407172	6.563856
96	3.135494	7.869402	9.001839	4.976734	8.262043	5.379897
128	4.941642	9.087834	8.248791	4.955827	6.967909	1.098612
171	5.298317	10.160530	9.894245	6.478510	9.079434	8.740337
193	5.192957	8.156223	9.917982	6.865891	8.633731	6.501290
218	2.890372	8.923191	9.629380	7.158514	8.475746	8.759669
304	5.081404	8.917311	10.117510	6.424869	9.374413	7.787382
305	5.493061	9.468001	9.088399	6.683361	8.271037	5.351858
338	1.098612	5.808142	8.856661	9.655090	2.708050	6.309918
353	4.762174	8.742574	9.961898	5.429346	9.069007	7.013016
355	5.247024	6.588926	7.606885	5.501258	5.214936	4.844187
357	3.610918	7.150701	10.011086	4.919981	8.816853	4.700480
412	4.574711	8.190077	9.425452	4.584967	7.996317	4.127134

Data points considered outliers for the feature 'Milk':

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
86	10.039983	11.205013	10.377047	6.894670	9.906981	6.805723
98	6.220590	4.718499	6.656727	6.796824	4.025352	4.882802
154	6.432940	4.007333	4.919981	4.317488	1.945910	2.079442
356	10.029503	4.897840	5.384495	8.057377	2.197225	6.306275

Data points considered outliers for the feature 'Grocery':

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
75	9.923192	7.036148	1.098612	8.390949	1.098612	6.882437
154	6.432940	4.007333	4.919981	4.317488	1.945910	2.079442

Data points considered outliers for the feature 'Frozen':

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
38	8.431853	9.663261	9.723703	3.496508	8.847360	6.070738
57	8.597297	9.203618	9.257892	3.637586	8.932213	7.156177
65	4.442651	9.950323	10.732651	3.583519	10.095388	7.260523
145	10.000569	9.034080	10.457143	3.737670	9.440738	8.396155
175	7.759187	8.967632	9.382106	3.951244	8.341887	7.436617
264	6.978214	9.177714	9.645041	4.110874	8.696176	7.142827
325	10.395650	9.728181	9.519735	11.016479	7.148346	8.632128
420	8.402007	8.569026	9.490015	3.218876	8.827321	7.239215
429	9.060331	7.467371	8.183118	3.850148	4.430817	7.824446
439	7.932721	7.437206	7.828038	4.174387	6.167516	3.951244

Data points considered outliers for the feature 'Detergents_Paper':

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
75	9.923192	7.036148	1.098612	8.390949	1.098612	6.882437
161	9.428190	6.291569	5.645447	6.995766	1.098612	7.711101

Data points considered outliers for the feature 'Delicatessen':

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
66	2.197225	7.335634	8.911530	5.164786	8.151333	3.295837
109	7.248504	9.724899	10.274568	6.511745	6.728629	1.098612
128	4.941642	9.087834	8.248791	4.955827	6.967909	1.098612
137	8.034955	8.997147	9.021840	6.493754	6.580639	3.583519
142	10.519646	8.875147	9.018332	8.004700	2.995732	1.098612
154	6.432940	4.007333	4.919981	4.317488	1.945910	2.079442
183	10.514529	10.690808	9.911952	10.505999	5.476464	10.777768
184	5.789960	6.822197	8.457443	4.304065	5.811141	2.397895
187	7.798933	8.987447	9.192075	8.743372	8.148735	1.098612
203	6.368187	6.529419	7.703459	6.150603	6.860664	2.890372
233	6.871091	8.513988	8.106515	6.842683	6.013715	1.945910
285	10.602965	6.461468	8.188689	6.948897	6.077642	2.890372
289	10.663966	5.655992	6.154858	7.235619	3.465736	3.091042
343	7.431892	8.848509	10.177932	7.283448	9.646593	3.610918

```
Number of outliers:  42
Outliers:  [65, 66, 81, 95, 96, 128, 171, 193, 218, 304, 305, 338, 353,
355, 357, 412, 86, 98, 154, 356, 75, 38, 57, 145, 175, 264, 325, 420, 4
29, 439, 161, 109, 137, 142, 183, 184, 187, 203, 233, 285, 289, 343]

Number of repeat outliers:  5
Repeat outliers:  [154, 65, 75, 66, 128]
(440, 6) (398, 6)
```

Question 4

- Are there any data points considered outliers for more than one feature based on the definition above?
- Should these data points be removed from the dataset?
- If any data points were added to the `outliers` list to be removed, explain why.

Hint: If you have datapoints that are outliers in multiple categories think about why that may be and if they warrant removal. Also note how k-means is affected by outliers and whether or not this plays a factor in your analysis of whether or not to remove them.

Answer:

- Yes, there are 5 data points that are outliers for more tha one feature and are present in the list repeat_outliers
- Yes, all the outliers should be eliminated to prevent any outsized skewness effect on the analysis of our data.

Feature Transformation

In this section you will use principal component analysis (PCA) to draw conclusions about the underlying structure of the wholesale customer data. Since using PCA on a dataset calculates the dimensions which best maximize variance, we will find which compound combinations of features best describe customers.

Implementation: PCA

Now that the data has been scaled to a more normal distribution and has had any necessary outliers removed, we can now apply PCA to the `good_data` to discover which dimensions about the data best maximize the variance of features involved. In addition to finding these dimensions, PCA will also report the *explained variance ratio* of each dimension — how much variance within the data is explained by that dimension alone. Note that a component (dimension) from PCA can be considered a new "feature" of the space, however it is a composition of the original features present in the data.

In the code block below, you will need to implement the following:

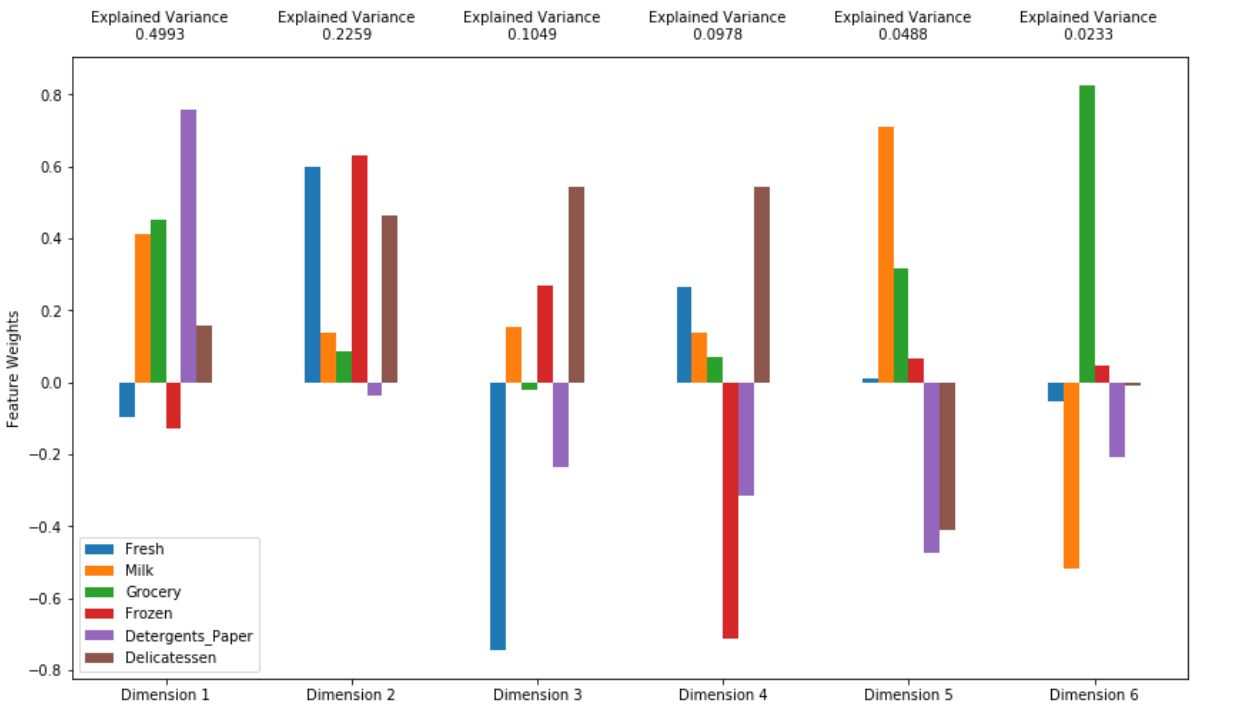
- Import `sklearn.decomposition.PCA` and assign the results of fitting PCA in six dimensions with `good_data` to `pca`.
- Apply a PCA transformation of `log_samples` using `pca.transform`, and assign the results to `pca_samples`.

In [48]:
```
from sklearn.decomposition import PCA


# TODO: Apply PCA by fitting the good data with the same number of dimen
sions as features
pca = PCA().fit(good_data)

# TODO: Transform log_samples using the PCA fit above
pca_samples = pca.transform(log_samples)

# Generate PCA results plot
pca_results = vs.pca_results(good_data, pca)
```



Question 5

- How much variance in the data is explained ***in total*** by the first and second principal component?
- How much variance in the data is explained by the first four principal components?
- Using the visualization provided above, talk about each dimension and the cumulative variance explained by each, stressing upon which features are well represented by each dimension(both in terms of positive and negative variance explained). Discuss what the first four dimensions best represent in terms of customer spending.

Hint: A positive increase in a specific dimension corresponds with an *increase* of the *positive-weighted* features and a *decrease* of the *negative-weighted* features. The rate of increase or decrease is based on the individual feature weights.

In [49]:
```
# DataFrame of results
display(pca_results)

# DataFrame
display(type(pca_results))

# Cumulative explained variance should add to 1
display(pca_results['Explained Variance'].cumsum())
```

	Explained Variance	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
Dimension 1	0.4993	-0.0976	0.4109	0.4511	-0.1280	0.7595	0.1579
Dimension 2	0.2259	0.6008	0.1370	0.0852	0.6300	-0.0376	0.4634
Dimension 3	0.1049	-0.7452	0.1544	-0.0204	0.2670	-0.2349	0.5422
Dimension 4	0.0978	0.2667	0.1375	0.0710	-0.7133	-0.3157	0.5445
Dimension 5	0.0488	0.0114	0.7083	0.3168	0.0671	-0.4729	-0.4120
Dimension 6	0.0233	-0.0543	-0.5177	0.8267	0.0471	-0.2080	-0.0094

```
pandas.core.frame.DataFrame

Dimension 1    0.4993
Dimension 2    0.7252
Dimension 3    0.8301
Dimension 4    0.9279
Dimension 5    0.9767
Dimension 6    1.0000
Name: Explained Variance, dtype: float64
```

Answer:

- The total variance explained by the first and second component is 72.52%
- The total variance explained by first four principal components is 92.79%
- The first principal component largely represents variation in spending on "Detergents_Paper", and to a lesser extent, "Grocery" and "Milk". The positive direction on the first component indicates higher spending on these product categories.
- The second principal component represents variation in spending on "Fresh", "Frozen", and "Delicatessen". The positive direction on the second component indicates higher spending on these product categories.
- The third principal component mostly represents variation in "Fresh" and "Delicatessen". The positive direction on the third component indicates higher spending on "Fresh" and lower spending on "Delicatessen".
- The fourth principal component mostly represents variation in "Frozen" and "Delicatessen", and to a lesser extent "Detergents_Paper". The positive direction on the fourth component indicates higher spending on "Frozen" and "Detergents_Paper", and lower spending on "Delicatessen".

Observation

Run the code below to see how the log-transformed sample data has changed after having a PCA transformation applied to it in six dimensions. Observe the numerical value for the first four dimensions of the sample points. Consider if this is consistent with your initial interpretation of the sample points.

In [50]:
```
# Display sample log-data after having a PCA transformation applied
display(pd.DataFrame(np.round(pca_samples, 4), columns = pca_results.ind
ex.values))
```

	Dimension 1	Dimension 2	Dimension 3	Dimension 4	Dimension 5	Dimension 6
0	-0.9986	1.3694	0.2854	-0.3997	-0.6781	0.6194
1	1.6408	-1.6766	-2.1998	-0.9155	1.0152	-0.0740
2	4.6398	2.1490	-1.0221	-0.1719	0.6589	0.1007

Implementation: Dimensionality Reduction

When using principal component analysis, one of the main goals is to reduce the dimensionality of the data — in effect, reducing the complexity of the problem. Dimensionality reduction comes at a cost: Fewer dimensions used implies less of the total variance in the data is being explained. Because of this, the *cumulative explained variance ratio* is extremely important for knowing how many dimensions are necessary for the problem. Additionally, if a signifiant amount of variance is explained by only two or three dimensions, the reduced data can be visualized afterwards.

In the code block below, you will need to implement the following:

- Assign the results of fitting PCA in two dimensions with `good_data` to `pca`.
- Apply a PCA transformation of `good_data` using `pca.transform`, and assign the results to `reduced_data`.
- Apply a PCA transformation of `log_samples` using `pca.transform`, and assign the results to `pca_samples`.

In [51]:
```
from sklearn.decomposition import PCA

# TODONE: Apply PCA by fitting the good data with only two dimensions
pca = PCA(n_components=2).fit(good_data)

# TODONE: Transform the good data using the PCA fit above
reduced_data = pca.transform(good_data)

# TODONE: Transform log_samples using the PCA fit above
pca_samples = PCA(n_components=2).fit_transform(log_samples)

# Create a DataFrame for the reduced data
reduced_data = pd.DataFrame(reduced_data, columns = ['Dimension 1', 'Dim
ension 2'])
```

Observation

Run the code below to see how the log-transformed sample data has changed after having a PCA transformation applied to it using only two dimensions. Observe how the values for the first two dimensions remains unchanged when compared to a PCA transformation in six dimensions.

In [52]:
```
# Display sample log-data after applying PCA transformation in two dimen
sions
display(pd.DataFrame(np.round(pca_samples, 4), columns = ['Dimension 1',
'Dimension 2']))
```

	Dimension 1	Dimension 2
0	3.0313	-1.3366
1	-0.0448	2.7343
2	-2.9865	-1.3977

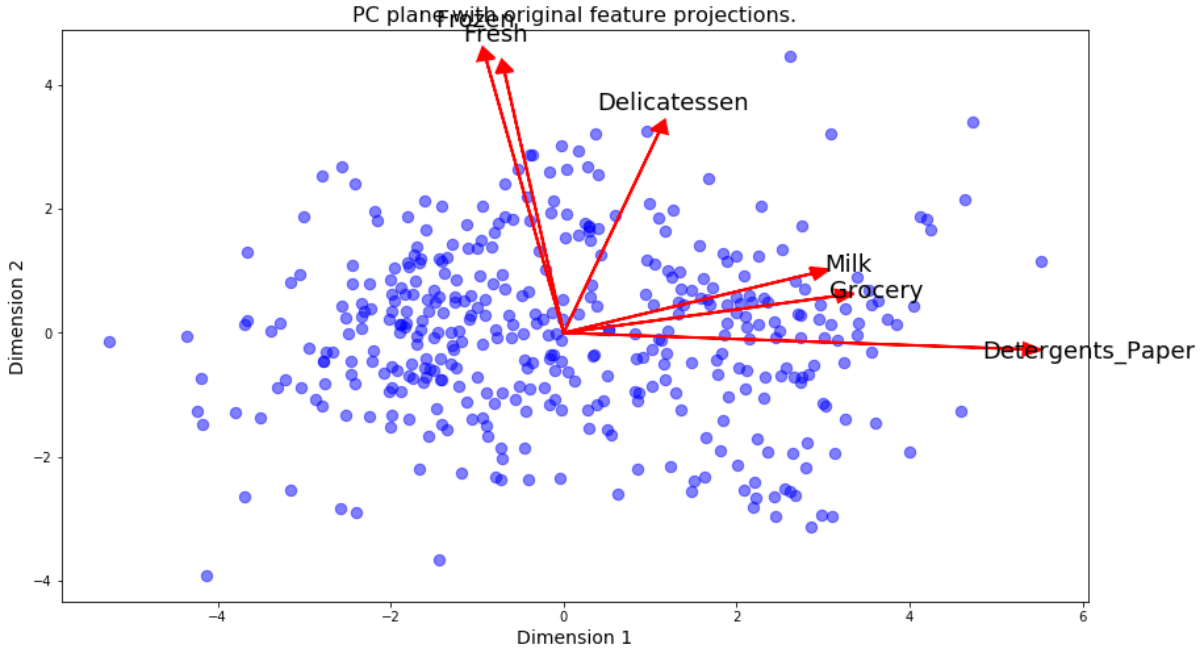
Visualizing a Biplot

A biplot is a scatterplot where each data point is represented by its scores along the principal components. The axes are the principal components (in this case `Dimension 1` and `Dimension 2`). In addition, the biplot shows the projection of the original features along the components. A biplot can help us interpret the reduced dimensions of the data, and discover relationships between the principal components and original features.

Run the code cell below to produce a biplot of the reduced-dimension data.

In [53]:
```
# Create a biplot
vs.biplot(good_data, reduced_data, pca)
```

Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff85b56e940>



Observation

Once we have the original feature projections (in red), it is easier to interpret the relative position of each data point in the scatterplot. For instance, a point the lower right corner of the figure will likely correspond to a customer that spends a lot on `'Milk'`, `'Grocery'` and `'Detergents_Paper'`, but not so much on the other product categories.

From the biplot, which of the original features are most strongly correlated with the first component? What about those that are associated with the second component? Do these observations agree with the pca_results plot you obtained earlier?

Clustering

In this section, you will choose to use either a K-Means clustering algorithm or a Gaussian Mixture Model clustering algorithm to identify the various customer segments hidden in the data. You will then recover specific data points from the clusters to understand their significance by transforming them back into their original dimension and scale.

Question 6

- What are the advantages to using a K-Means clustering algorithm?
- What are the advantages to using a Gaussian Mixture Model clustering algorithm?
- Given your observations about the wholesale customer data so far, which of the two algorithms will you use and why?

Hint: Think about the differences between hard clustering and soft clustering and which would be appropriate for our dataset.

Answer:

- It is a fast and efficient algorthm that is easily implemented, even on high dimensional data. It largely depends upon the starting points (centroids) chosen. It can be said to be doing hard-clustering i.e. it will exactly say in which cluster the data point belongs. And it doesn't pay any attention to the density of clustering.
- Gaussian Mixture Modeling (GMM) is good with clusters of any geometry and is not biased towards circular or near-circular clusters. GMM does soft clustering controrary to the Kmeans. It's performance is better when the density of data points is not uniform.
- The clusters wont be circular as we have reduced the dimensionality of the data, and therefore for this particular situation GMM would be more preferred algorithm for this situation over the Kmeans. The data should itself indicate soft clustering since every business is different from another but they may share upto certain extent of similarity.

Implementation: Creating Clusters

Depending on the problem, the number of clusters that you expect to be in the data may already be known. When the number of clusters is not known *a priori*, there is no guarantee that a given number of clusters best segments the data, since it is unclear what structure exists in the data — if any. However, we can quantify the "goodness" of a clustering by calculating each data point's *silhouette coefficient*. The [silhouette coefficient (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html)](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html) for a data point measures how similar it is to its assigned cluster from -1 (dissimilar) to 1 (similar). Calculating the *mean* silhouette coefficient provides for a simple scoring method of a given clustering.

In the code block below, you will need to implement the following:

- Fit a clustering algorithm to the `reduced_data` and assign it to `clusterer`.
- Predict the cluster for each data point in `reduced_data` using `clusterer.predict` and assign them to `preds`.
- Find the cluster centers using the algorithm's respective attribute and assign them to `centers`.
- Predict the cluster for each sample data point in `pca_samples` and assign them `sample_preds`.
- Import `sklearn.metrics.silhouette_score` and calculate the silhouette score of `reduced_data` against `preds`.
  - Assign the silhouette score to `score` and print the result.

In [54]:
```
from sklearn.metrics import silhouette_score
from sklearn.mixture import GaussianMixture

def run_clusterer(n_components):
    '''
    Functionalized it so that I could easily run it with different n_com
ponents.
    '''

    # TODONE: Apply your clustering algorithm of choice to the reduced d
ata
    clusterer = GaussianMixture(n_components=n_components, random_state=
42).fit(reduced_data)

    # TODONE: Predict the cluster for each data point
    preds = clusterer.predict(reduced_data)

    # TODONE: Find the cluster centers
    centers = clusterer.means_

    # TODONE: Predict the cluster for each transformed sample data point
    sample_preds = clusterer.predict(pca_samples)

    # TODONE: Calculate the mean silhouette coefficient for the number o
f clusters chosen
    score = silhouette_score(reduced_data, preds)

    return clusterer, preds, centers, sample_preds, score
```

Question 7

- Report the silhouette score for several cluster numbers you tried.
- Of these, which number of clusters has the best silhouette score?

In [55]:
```
best_score = 0
best_n = 0
for i in range(2, 26):
    clusterer, preds, centers, sample_preds, score = run_clusterer(i)
    print("The score with {} clusters is: {}".format(i, score))

    if score > best_score:
        best_score = score
        best_n = i

clusterer, preds, centers, sample_preds, score = run_clusterer(best_n)
print("\nThe best n is: {}, with a score of: {}".format(best_n, score))
```

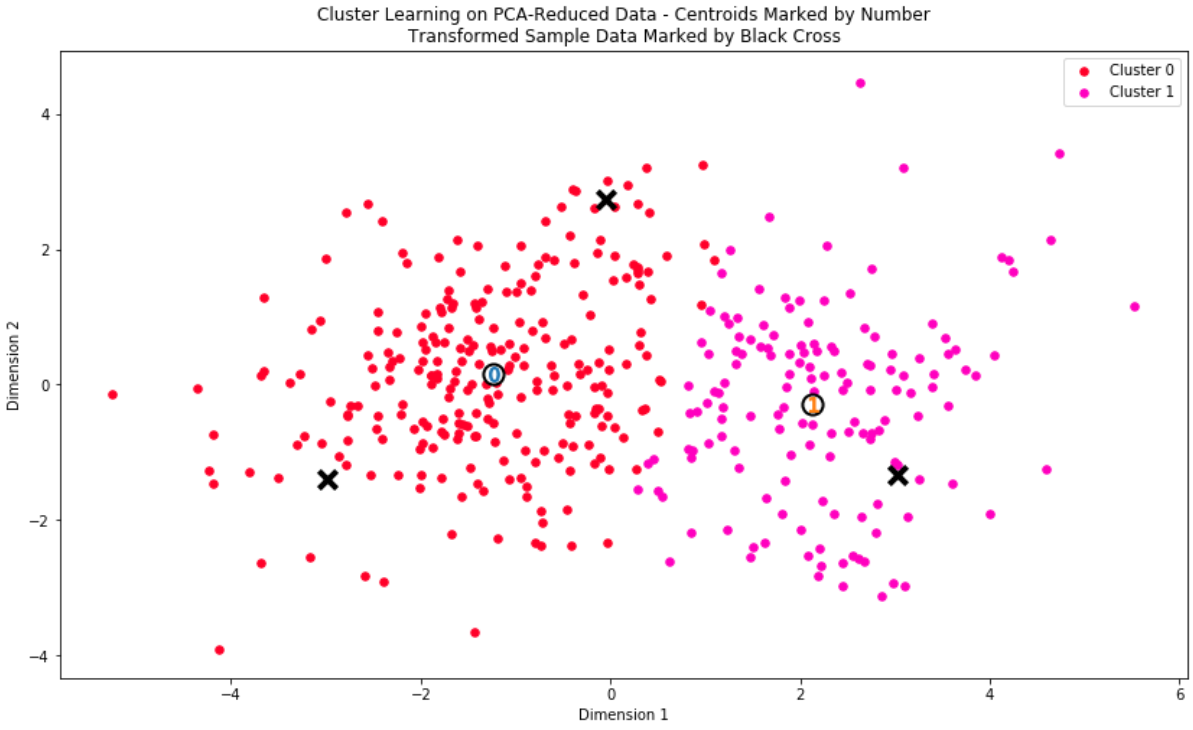
```
The score with 2 clusters is: 0.4474119955709115
The score with 3 clusters is: 0.3611936250386726
The score with 4 clusters is: 0.31825345740294686
The score with 5 clusters is: 0.31305656517669966
The score with 6 clusters is: 0.3406037163816709
The score with 7 clusters is: 0.3296607819491736
The score with 8 clusters is: 0.3291220677952353
The score with 9 clusters is: 0.2626846949782737
The score with 10 clusters is: 0.29957956719713585
The score with 11 clusters is: 0.30398351033213067
The score with 12 clusters is: 0.3108069469535372
The score with 13 clusters is: 0.2693777104624079
The score with 14 clusters is: 0.33737161783192743
The score with 15 clusters is: 0.3362896197611604
The score with 16 clusters is: 0.3140055748722796
The score with 17 clusters is: 0.29587258818872975
The score with 18 clusters is: 0.3184236216847963
The score with 19 clusters is: 0.32113332488161517
The score with 20 clusters is: 0.32362483005004833
The score with 21 clusters is: 0.2798756736382088
The score with 22 clusters is: 0.2777492049972552
The score with 23 clusters is: 0.27539860277577727
The score with 24 clusters is: 0.28147172064062437
The score with 25 clusters is: 0.2840306459010754

The best n is: 2, with a score of: 0.4474119955709115
```

Cluster Visualization

Once you've chosen the optimal number of clusters for your clustering algorithm using the scoring metric above, you can now visualize the results by executing the code block below. Note that, for experimentation purposes, you are welcome to adjust the number of clusters for your clustering algorithm to see various visualizations. The final visualization provided should, however, correspond with the optimal number of clusters.

In [56]:
```
# Display the results of the clustering from implementation
vs.cluster_results(reduced_data, preds, centers, pca_samples)
```



Implementation: Data Recovery

Each cluster present in the visualization above has a central point. These centers (or means) are not specifically data points from the data, but rather the *averages* of all the data points predicted in the respective clusters. For the problem of creating customer segments, a cluster's center point corresponds to *the average customer of that segment*. Since the data is currently reduced in dimension and scaled by a logarithm, we can recover the representative customer spending from these data points by applying the inverse transformations.

In the code block below, you will need to implement the following:

- Apply the inverse transform to `centers` using `pca.inverse_transform` and assign the new centers to `log_centers`.
- Apply the inverse function of `np.log` to `log_centers` using `np.exp` and assign the true centers to `true_centers`.

In [57]:
```
# TODONE: Inverse transform the centers
log_centers = pca.inverse_transform(centers)

# TODONE: Exponentiate the centers
true_centers = np.exp(log_centers)

# Display the true centers
segments = ['Segment {}'.format(i) for i in range(0,len(centers))]
true_centers = pd.DataFrame(np.round(true_centers), columns = data.keys
())
true_centers.index = segments
display(true_centers)
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
Segment 0	9468.0	2067.0	2624.0	2196.0	343.0	799.0
Segment 1	5174.0	7776.0	11581.0	1068.0	4536.0	1101.0

Question 8

- Consider the total purchase cost of each product category for the representative data points above, and reference the statistical description of the dataset at the beginning of this project(specifically looking at the mean values for the various feature points). What set of establishments could each of the customer segments represent?

Hint: A customer who is assigned to `'Cluster X'` should best identify with the establishments represented by the feature set of `'Segment X'`. Think about what each segment represents in terms their values for the feature points chosen. Reference these values with the mean values to get some perspective into what kind of establishment they represent.

In [58]:
```
# Rerun stats with outliers removed and normalization reversed.
clean_data = np.exp(good_data)
display(clean_data.describe())
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.00000
mean	12430.630653	5486.314070	7504.907035	3028.809045	2725.376884	1454.71608
std	12552.698266	6410.878177	9263.803670	3712.563636	4644.023066	1746.45365
min	255.000000	201.000000	223.000000	91.000000	5.000000	46.00000
25%	4043.500000	1597.250000	2125.000000	830.000000	263.250000	448.25000
50%	9108.000000	3611.500000	4573.000000	1729.500000	788.000000	997.50000
75%	16969.000000	6802.500000	9762.250000	3745.000000	3660.500000	1830.00000
max	112151.000000	54259.000000	92780.000000	35009.000000	40827.000000	16523.00000

Answer

A customer in Segment 1 or 'Cluster 1' could be a dine-in restaurant or deli or cafe. This explains the higher Fresh, Milk, Frozen, and Grocery numbers.

A customer in Segment 0 or 'Cluster 0' could be a grocery shop, human beings, or supermarkets or some other business that doesn't hand out a lot of disposable items like paper cups or fast food wrappers. This would also explain the higher Fresh number.

Question 9

- For each sample point, which customer segment from ***Question 8*** best represents it?
- Are the predictions for each sample point consistent with this?*

Run the code block below to find which cluster each sample point is predicted to be.

In [60]:
```
# Display the predictions
for i, pred in enumerate(sample_preds):
    print("Sample point", i, "predicted to be in Cluster", pred)
```

```
Sample point 0 predicted to be in Cluster 1
Sample point 1 predicted to be in Cluster 0
Sample point 2 predicted to be in Cluster 0
```

Answers

- This model correctly predicts the segments which I had assumed initially.
  - Sample 0: assumed= Restaurant/Deli; predicted= Segment 1(Restaurant/Deli)
  - Sample 1: assumed= Grocery store; predicted= Segment 0(grocery store/supermarket/individuals)
  - Sample 2: assumed= market store; predicted= Segment 0(grocery store/supermarket/individuals)

Conclusion

In this final section, you will investigate ways that you can make use of the clustered data. First, you will consider how the different groups of customers, the ***customer segments***, may be affected differently by a specific delivery scheme. Next, you will consider how giving a label to each customer (which *segment* that customer belongs to) can provide for additional features about the customer data. Finally, you will compare the ***customer segments*** to a hidden variable present in the data, to see whether the clustering identified certain relationships.

Question 10

Companies will often run A/B tests (https://en.wikipedia.org/wiki/A/B_testing) when making small changes to their products or services to determine whether making that change will affect its customers positively or negatively. The wholesale distributor is considering changing its delivery service from currently 5 days a week to 3 days a week. However, the distributor will only make this change in delivery service for customers that react positively.

- How can the wholesale distributor use the customer segments to determine which customers, if any, would react positively to the change in delivery service?*

Hint: Can we assume the change affects all customers equally? How can we determine which group of customers it affects the most?

Answer:

We can take a subset of data points close to the cluster centers to act as representative of their respective clusters, change the delivery frequency on half of the points in the subsets for each cluster, and see how those customers react in comparison to the other customers in their cluster who still have the old delivery frequency. Based on this, we can predict how all customers in each cluster would react to the change in delivery frequency.

Question 11

Additional structure is derived from originally unlabeled data when using clustering techniques. Since each customer has a ***customer segment*** it best identifies with (depending on the clustering algorithm applied), we can consider '*customer segment*' as an **engineered feature** for the data. Assume the wholesale distributor recently acquired ten new customers and each provided estimates for anticipated annual spending of each product category. Knowing these estimates, the wholesale distributor wants to classify each new customer to a ***customer segment*** to determine the most appropriate delivery service.

- How can the wholesale distributor label the new customers using only their estimated product spending and the **customer segment** data?

Hint: A supervised learner could be used to train on the original customers. What would be the target variable?

Answer:

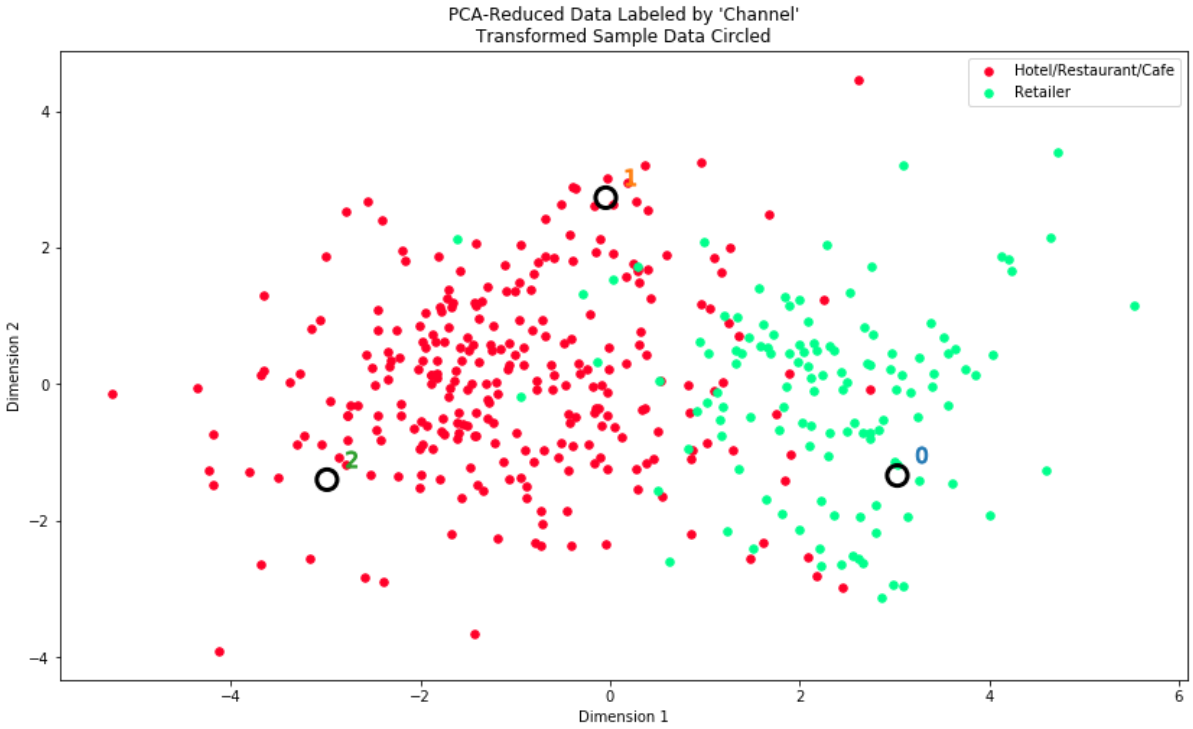
The wholesale distributor can train supervised learning algorithms like SVM or Decision tree classifier where the initial dataset of customer spending can be used to train the algorithm with customer spending as inputs and customer segments as the target variable.

Visualizing Underlying Distributions

At the beginning of this project, it was discussed that the `'Channel'` and `'Region'` features would be excluded from the dataset so that the customer product categories were emphasized in the analysis. By reintroducing the `'Channel'` feature to the dataset, an interesting structure emerges when considering the same PCA dimensionality reduction applied earlier to the original dataset.

Run the code block below to see how each data point is labeled either `'HoReCa'` (Hotel/Restaurant/Cafe) or `'Retail'` the reduced space. In addition, you will find the sample points are circled in the plot, which will identify their labeling.

In [61]:
```
# Display the clustering results based on 'Channel' data
vs.channel_results(reduced_data, outliers, pca_samples)
```



Question 12

- How well does the clustering algorithm and number of clusters you've chosen compare to this underlying distribution of Hotel/Restaurant/Cafe customers to Retailer customers?
- Are there customer segments that would be classified as purely 'Retailers' or 'Hotels/Restaurants/Cafes' by this distribution?
- Would you consider these classifications as consistent with your previous definition of the customer segments?

Answer:

The GMM clustering algorithm was able to depict the relationships very nicely. For most of the samples, it matched the initial assumptions.

There are segments which are pure and present on extreme left and right of the plot. They can be classified purely.

> **Note**: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to **File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

In [ ]: