Project: Creating Customer Segments

The 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.

In [20]:
```
# 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 DataFrames
import warnings
warnings.filterwarnings('ignore')
# 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 each.".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 [21]:
```
# Display a description of the dataset
display(data.describe())
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	De
count	440.000000	440.000000	440.000000	440.000000	440.000000	440
mean	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	152
std	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	282
min	3.000000	55.000000	3.000000	25.000000	3.000000	3.0
25%	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408
50%	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965
75%	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	182
max	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	479

Implementation: Selecting Samples

In [22]:
```
# Select three indices of your choice you wish to sample from the datase
t
indices = [18,199,351]

# 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	18601	6327	10099	2205	2767	3181
1	9670	2280	2112	520	402	347
2	1210	10044	22294	1741	12638	3137

Observation 1

- #1: Local grocery store. For my first sample, the fresh and grocery purchases are above the average, where local Deli will purchase and sell a lot of fresh and grocery goods.
- #2: student. This customer made purchases on goods that are way below average level, average grocery is 7951, but this sample customer only spent 2112, which means that they do not need to do grocery very often, and student could be a good fit for this category.
- #3: Restaurant: The last sample customer consumes huge amount of grocery and detergent_papers, indicating that they could be a possible restaurant that need to purchase a lot of food and detergent papers for cleaning purpose.

Implementation: Feature Relevance

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 train 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.

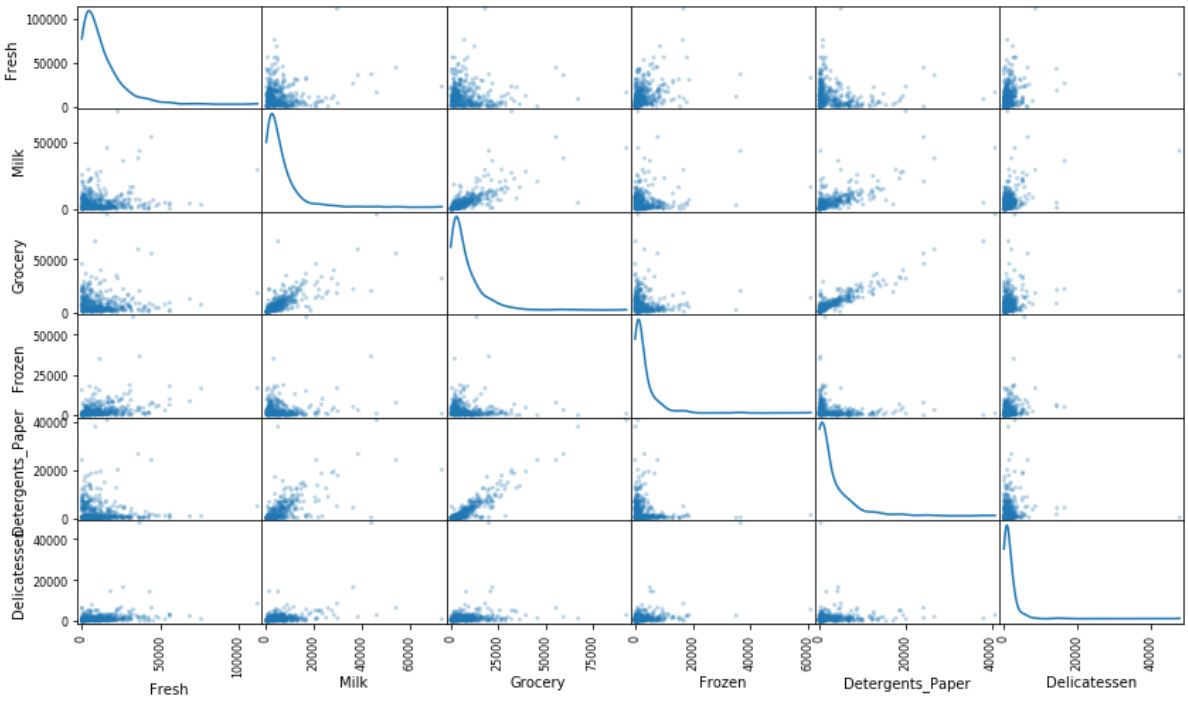
- Assign `new_data` a copy of the data by removing a feature of your choice using the `DataFrame.drop` function.
- Use `sklearn.cross_validation.train_test_split` to split the dataset into training and testing sets.
  - Use the removed feature as your target label. Set a `test_size` of `0.25` and set 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 [23]:
```
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeRegressor
new_data = data.drop(axis=1, columns=['Milk']).copy()
y = data['Milk']

# Set a random state.
X_train, X_test, y_train, y_test = train_test_split(new_data,
                                                    y,
                                                    test_size = 0.25,
                                                    random_state = 0)


regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
score = regressor.score(X_test, y_test)
print(score)
```

```
0.08894146449630225
```

Visualize Feature Distributions

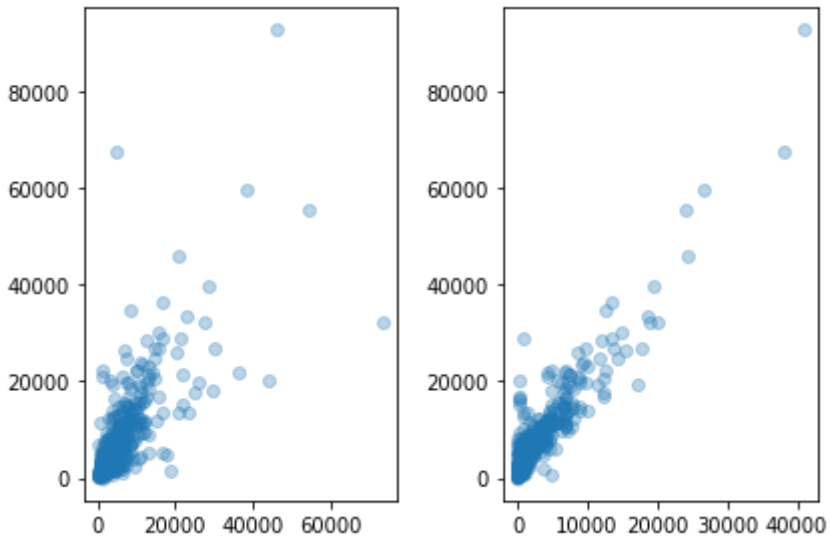
In [24]:
```
# Produce a scatter matrix for each pair of features in the data
pd.scatter_matrix(data, alpha = 0.3, figsize = (14,8), diagonal = 'kde'
);
```



Observation 3

From the correlation matrix, all the features are highly skewed to the right, with long tail on the right, and they are not normally distributed. Other than Grocery and Detergents_Paper features, all other features are not highly correlated with each other. Especially Milk and Grocery, Detergents_paper and Grocery, these two pairs have strong correlations with each other. The strong correlation between Milk and Grocery also support my suspicion that Milk will be a feature that is relevant in Customers' choices.

In [25]:
```
import matplotlib.pyplot as plt
plt.subplot(1,2,1)
plt.scatter(data['Milk'], data['Grocery'],alpha= .3)
plt.subplot(1,2,2)
plt.scatter(data['Detergents_Paper'], data['Grocery'], alpha= .3)
plt.tight_layout()
plt.show()
```

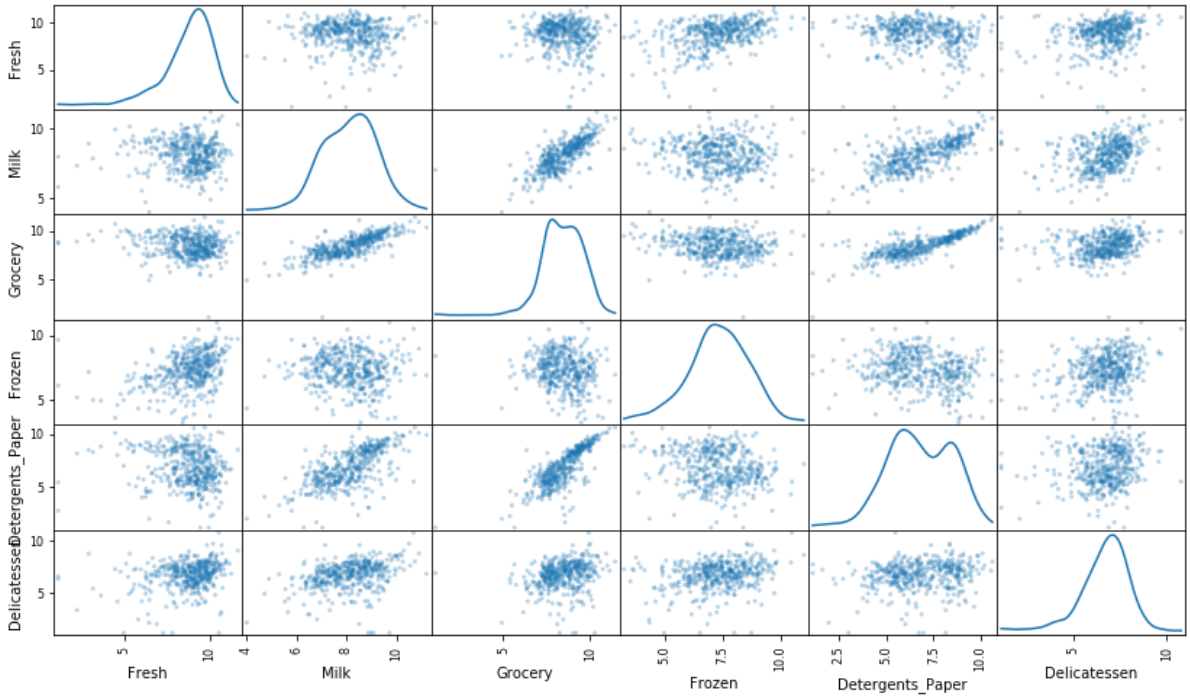


Data Preprocessing

Implementation: Feature Scaling

- 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 [26]:
```
# Scale the data using the natural logarithm
log_data = np.log(data)

# Scale the sample data using the natural logarithm
log_samples = pd.DataFrame(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 = 'k
de');
```



Observation

In [27]:
```
# Display the log-transformed sample data
display(log_samples)
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	9.830971	8.752581	9.220192	7.698483	7.925519	8.064951
1	9.176784	7.731931	7.655391	6.253829	5.996452	5.849325
2	7.098376	9.214731	10.012073	7.462215	9.444463	8.051022

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 (http://datapigtechnologies.com/blog/index.php/highlighting-outliers-in-your-data-with-the-tukey-method/): 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.

- 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.

In [28]:
```
# For each feature find the data points with extreme high or low values

index_list = []
for feature in log_data.keys():

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

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

    # Use the interquartile range to calculate an outlier step (1.5 time
s the interquartile range)
    step = (Q3 - Q1) * 1.5

    # Display the outliers
    print("Data points considered outliers for the feature '{}':".format
(feature))
    index_list +=(log_data[~((log_data[feature] >= Q1 - step) & (log_dat
a[feature] <= Q3 + step))].index.tolist())
    display(log_data[~((log_data[feature] >= Q1 - step) & (log_data[feat
ure] <= Q3 + step))])

print(index_list)

print ("The below data points are outliers in multiple features:")
a = sorted({i for i in index_list if index_list.count(i) > 1})
print (a)

# remove the extras in idx
for i in a:
    index_list.remove(i)

# OPTIONAL: Select the indices for data points you wish to remove
outliers  = index_list

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

print('Follwing are the outlier list: ',outliers)

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

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

```
[65, 66, 81, 95, 96, 128, 171, 193, 218, 304, 305, 338, 353, 355, 357,
412, 86, 98, 154, 356, 75, 154, 38, 57, 65, 145, 175, 264, 325, 420, 42
9, 439, 75, 161, 66, 109, 128, 137, 142, 154, 183, 184, 187, 203, 233,
285, 289, 343]
The below data points are outliers in multiple features:
[65, 66, 75, 128, 154]
Follwing are the outlier list:  [81, 95, 96, 171, 193, 218, 304, 305, 3
38, 353, 355, 357, 412, 86, 98, 356, 154, 38, 57, 65, 145, 175, 264, 32
5, 420, 429, 439, 75, 161, 66, 109, 128, 137, 142, 154, 183, 184, 187,
203, 233, 285, 289, 343]
```

In [29]:
```
data.iloc[[65,66,128,154],]
```

Out[29]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
65	85	20959	45828	36	24231	1423
66	9	1534	7417	175	3468	27
128	140	8847	3823	142	1062	3
154	622	55	137	75	7	8

Observation 4

From Outlier Detection process, we found that there are 11 records were being considered more than one feature. These outliers should be removed from the dataset, as they will affect the accuracy of the clustering result. For index number is 65, this customer seems to purchase excessive amount of milk, grocery and detergent paper, which seems hard to cluster this customer with other customers, while most of the rest of outlier records, they seem to have way below average purchases for each feature, therefore, they will be removed to help the algorithm to focus on the majority of the customers.

Feature Transformation

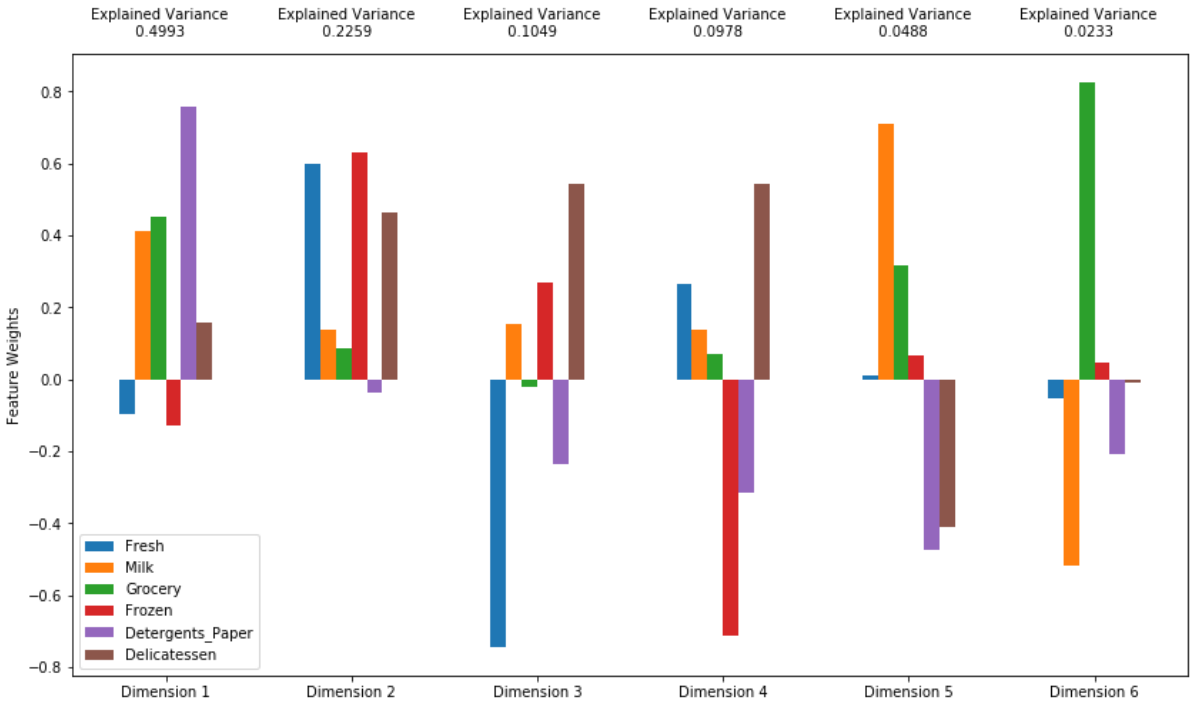
Implementation: PCA

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.

- 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 [30]:
```
# Apply PCA by fitting the good data with the same number of dimensions
 as features
from sklearn.decomposition import PCA
num_feature = len(good_data.columns)

pca = PCA(n_components=num_feature).fit(good_data)
# 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)

# pca.explained_variance_ratio_

fir_var_exp = pca.explained_variance_ratio_[0]
sec_var_exp = pca.explained_variance_ratio_[1]

first_two_var_explained = sum(pca.explained_variance_ratio_[:2])
first_four_var_explained = sum(pca.explained_variance_ratio_[:4])

print('Variance explained by the first principal component: {}'.format(f
ir_var_exp))
print('Variance explained by the second principal component: {}'.format(
sec_var_exp))
print('Total variance explained by the first two principal components:
{}'.format(first_two_var_explained))
print('Total variance explained by the first four principal components:
{}'.format(first_four_var_explained))
```

```
Variance explained by the first principal component: 0.4993048380748743
Variance explained by the second principal component: 0.225948066791485
66
Total variance explained by the first two principal components: 0.72525
290486636
Total variance explained by the first four principal components: 0.9279
536052307858
```



Observation 5

In total, the variance explained by the first two principal components is 0.7253, and first four principal components explained 0.9279 of the total variance.

- Dimension 1 has a high positive weight for Milk, Grocery, and Detergents_Paper features. This dimension could indicate Hotels, where the guests consume a lot of milk, grocery and detergent papers.
- Dimension 2 has a high positive weight for Fresh, Frozen, and Delicatessen. This dimension might represent ‘restaurants’, where cooks need fresh or frozen ingredients for cooking dishes.
- Dimension 3 has a high positive weight for Deli and Frozen features, and a low positive weight for Milk, but has negative weights for Fresh and Detergent Paper. This dimension might represent Fast Food restaurants, where they do not use a lot fresh ingredients.
- Dimension 4 has positive weights for Frozen, Detergents_Paper and Groceries, while having high negative for Fresh and Deli. This dimension might represent Delis, where they do not consume a lot frozen ingredients.

Observation

In [31]:
```
# 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	1.5627	1.4065	-0.0979	0.5199	-0.3294	0.0511
1	-1.1286	-1.1240	-0.8700	0.5270	0.1726	-0.3244
2	3.5582	-0.3169	1.5663	-0.4077	-0.5108	0.2878

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.

- 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 [32]:
```
# Apply PCA by fitting the good data with only two dimensions
pca = PCA(n_components=2).fit(good_data)

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

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

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

Observation

In [33]:
```
# 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']))
```

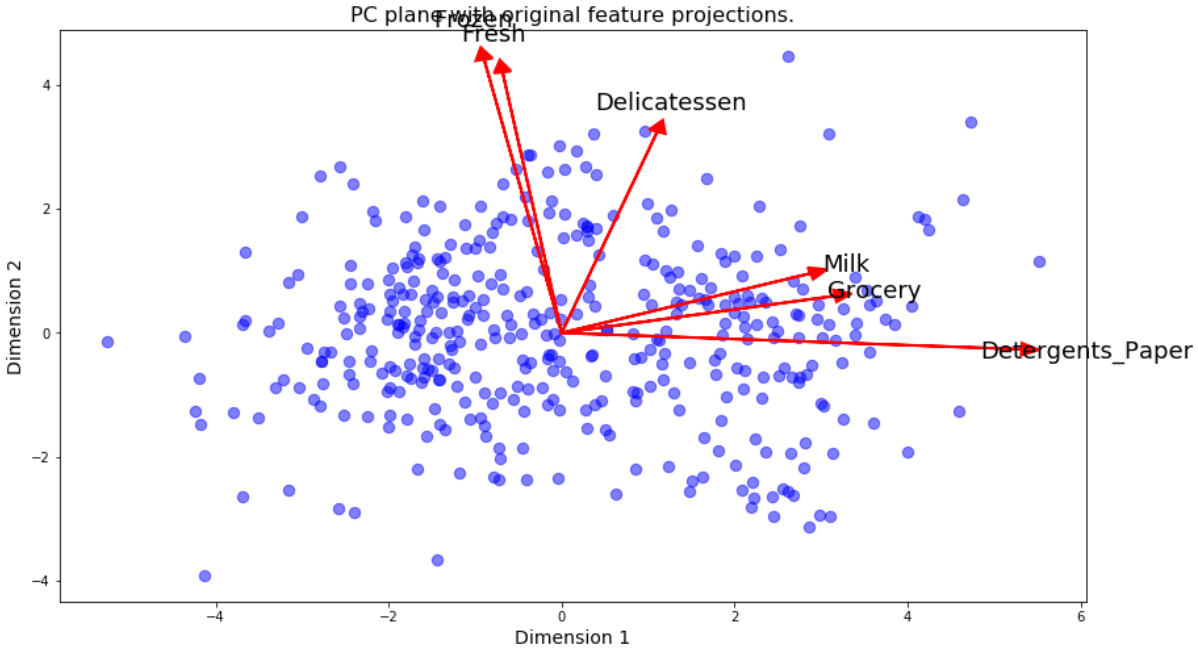
	Dimension 1	Dimension 2
0	1.5627	1.4065
1	-1.1286	-1.1240
2	3.5582	-0.3169

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.

In [34]:
```
# Create a biplot
vs.biplot(good_data, reduced_data, pca)
```

Out[34]: `<matplotlib.axes._subplots.AxesSubplot at 0x1a2092def0>`



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.

Clustering

Observation 6

Model selection

K-Means Clustering Algorithm Advantage:

1. It is easy to implement
2. Fast without using a lot computations.
3. It produce tighter clusters than hierarchical clustering.

Gaussian Mixture Model clustering algorithm advantage:

1. Do not assume cluster and work well with non-linear distribution.
2. Do not have the bias on the cluster size
3. Work well for soft assignment, the uncertainty of points belong to which cluster.

Given current understanding of the data, I would choose Gaussian Mixture Model. The main reason is that GMM works better for soft assignments. As for our customer data, it is not quite clear that we have certain clusters among the data points, and for the densely packed areas, for example, milk, grocery and detergent paper seems to have strong correlation with each other, while clustering them, could incur a lot of uncertainties, and in this case GMM will do a better job in creating clusters.

Implementation: Creating Clusters

- 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 [35]:
```
from sklearn.mixture import GMM
from sklearn.metrics import silhouette_score

cluster_num = [2,3,4,5,6]

def gmm_cluster(cluster_num):
    for i in cluster_num:
        clusterer = GMM(n_components=i).fit(reduced_data)
        preds = clusterer.predict(reduced_data)
        centers = clusterer.means_
        sample_preds = clusterer.predict(pca_samples)
        score =  silhouette_score(reduced_data,preds)
        print('For cluster of {} the score is: {}'.format(i,score))


gmm_cluster(cluster_num)
```

```
For cluster of 2 the score is: 0.4436014740151462
For cluster of 3 the score is: 0.35505321951057794
For cluster of 4 the score is: 0.29164636837674707
For cluster of 5 the score is: 0.2683982070363696
For cluster of 6 the score is: 0.3071228664473279
```

Observation 7

- For cluster of 2 the score is: 0.4436014740151462
- For cluster of 3 the score is: 0.3572945142491256
- For cluster of 4 the score is: 0.29232710283686186
- For cluster of 5 the score is: 0.2631362576236144
- For cluster of 6 the score is: 0.311041640059507

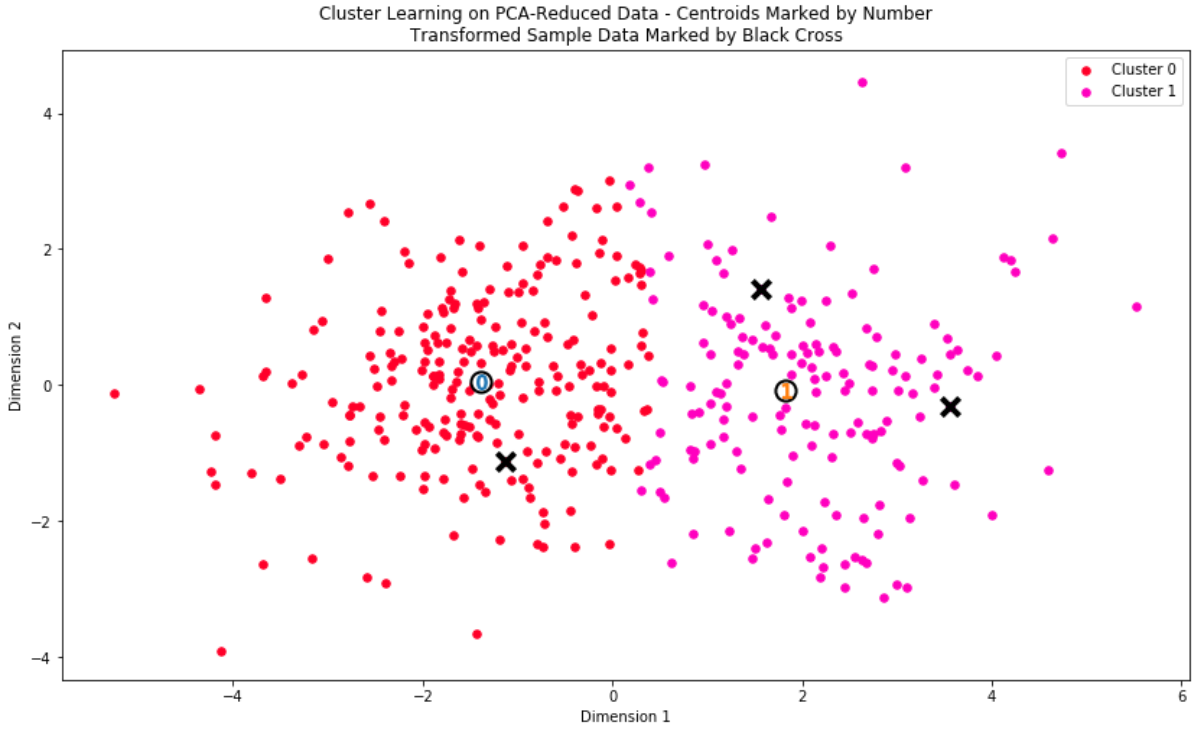
Based on the result, using 2 clusters, we can acheive the best result

Cluster Visualization

In [36]:
```
# Apply your clustering algorithm of choice to the reduced data
clusterer = GMM(n_components = 2)
clusterer.fit(reduced_data)

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

# Find the cluster centers
centers = clusterer.means_

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

# Calculate the mean silhouette coefficient for the number of clusters c
hosen
from sklearn.metrics import silhouette_score
score = silhouette_score(reduced_data, preds)
print(score)
```

```
0.4436014740151462
```

In [37]:
```
# 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.

- 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 [38]:
```
# Inverse transform the centers
log_centers = pca.inverse_transform(centers)


# 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	8967.0	1920.0	2437.0	2081.0	309.0	741.0
Segment 1	6079.0	7042.0	10241.0	1275.0	3546.0	1159.0

In [40]:
```
data.describe()
```

Out[40]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	De
count	440.000000	440.000000	440.000000	440.000000	440.000000	440
mean	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	152
std	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	282
min	3.000000	55.000000	3.000000	25.000000	3.000000	3.0
25%	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408
50%	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965
75%	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	182
max	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	479

Observation 8

For segment 0, milk and grocery is significantly higher than average, which indicates that such segment could be a local restaurant which serves coffee inside, with high consumption of milk.

For segment 1, which have higher fresh purchase among other categories, which could indicate a local convenience store which focusing on local fresh produces.

Observation 9

In [39]:
```
# 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 1
```

Sample point 0 best represented by cluster 1, which is a convenience store

Sample point 1 best represented by cluster 0, which is a restaurant

Sample point 2 best represented by cluster 1, which is a convenience store

Conclusion

Final Conclusion

Observation 10

When applying A/B testing, different customers will react differently towards the changes. In this case, the wholesale distributor is considering to change the delivery frequency from 5 days a week to 3 days a week, basically deliver goods less frequent than before. Most importantly, the wholesale distributor will only make changes to customers react positively, in other words, less sensitive to negative delivery frequency change, who do not require frequent delivery services.

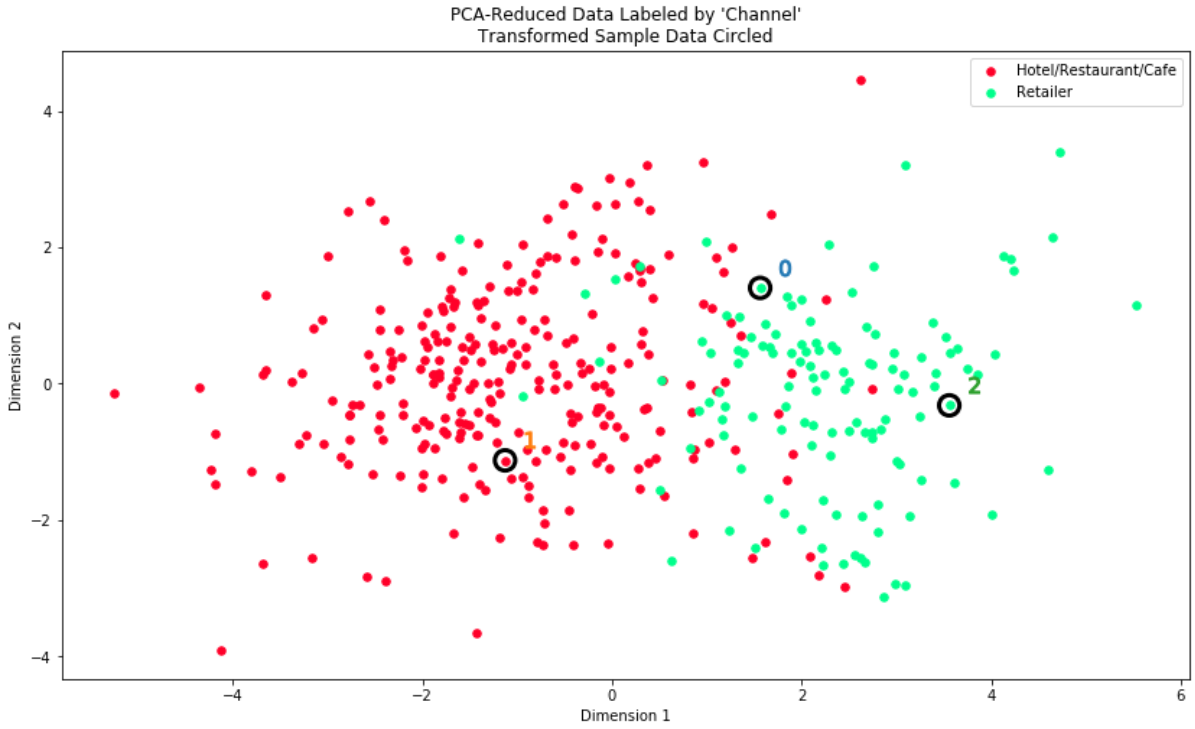
From our example, we observed that for the segment 0, which potentially is a restaurant, and requires a lot of milk and grocery. High milk and grocery consumption will need frequent delivery, and by applying the change in delivery frequency from 5 days to 3 days per week will incur negative effect to the restaurant. Therefore, based on clustering result, it is not recommended to the wholesale distributor to apply such change to segment 0.

Comparing to segment 0, segment 1 consume much less than segment 0, and all purchase quantities were below average. In addition, segment 1 purchased a lot frozen goods comparing to the mean, while other categories' purchase were way below average, thus, segment 1 customer does not seem to require frequent delivery service. It is recommended that wholesale distributor to apply the new delivery change to segment 1 customer to have positive feedback.

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.

In [40]:
```
# Display the clustering results based on 'Channel' data
vs.channel_results(reduced_data, outliers, pca_samples)
```



Sample point 0 best represented by cluster 1, which is a convenience store

Sample point 1 best represented by cluster 0, which is a restaurant

Sample point 2 best represented by cluster 1, which is a convenience store

Observation 12

The clustering did a great job in clustering the channel type, all three sample prediction were correct. Even for sample point 0, which is really close to cluster 0, the clustering algorithm is able to get the correct type.

Based on the result, regarding there are few overlapping data points, in general, we can say that it is clear there are two clusters here, and a lot of the data points would classify purely as ‘Retailers’ or ‘Hotels/Restaurants/Cafes’ by this distribution.

Overall, I believe these classifications are consistent with my previous definition of the customer segments.