customer_segments

October 24, 2017

1 Machine Learning Engineer Nanodegree

1.1 Unsupervised Learning

1.2 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.

1.3 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. 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 DataFrames

# 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(*da
except:
    print "Dataset could not be loaded. Is the dataset missing?"
```

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

1.4 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  \
count      440.000000    440.000000    440.000000    440.000000
mean     12000.297727   5796.265909   7951.277273   3071.931818
std      12647.328865   7380.377175   9503.162829   4854.673333
min          3.000000     55.000000      3.000000     25.000000
25%       3127.750000   1533.000000   2153.000000    742.250000
50%       8504.000000   3627.000000   4755.500000   1526.000000
75%      16933.750000   7190.250000  10655.750000   3554.250000
max     112151.000000  73498.000000  92780.000000  60869.000000

       Detergents_Paper  Delicatessen
count        440.000000    440.000000
mean        2881.493182   1524.870455
```

```
std          4767.854448   2820.105937
min             3.000000      3.000000
25%           256.750000    408.250000
50%           816.500000    965.500000
75%          3922.000000   1820.250000
max         40827.000000  47943.000000
```

1.4.1 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.

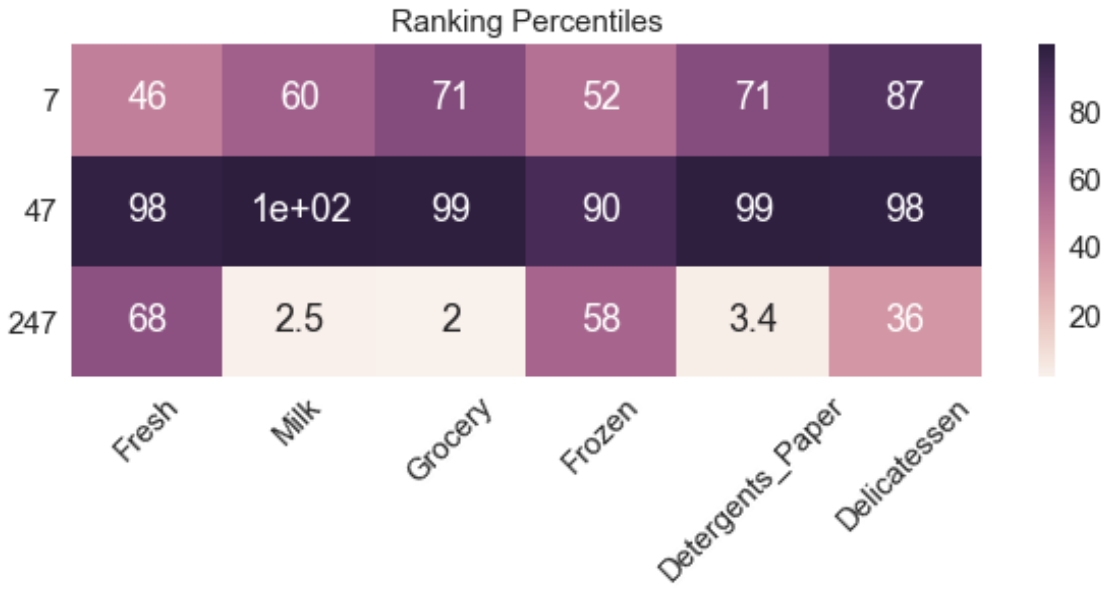
In [3]:
```
# TODO: Select three indices of your choice you wish to sample from the dataset
indices = [7, 47, 247]

# Create a DataFrame of the chosen samples
samples = pd.DataFrame(data.loc[indices], columns = data.keys()).reset_index(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	7579	4956	9426	1669	3321	2566
1	44466	54259	55571	7782	24171	6465
2	13569	346	489	2077	44	659

In [4]:
```
import matplotlib.pyplot as plt
import seaborn as sns

ranking_percentiles = data.rank(axis=0, pct=True).iloc[indices]*100

plt.figure(figsize=(10,3))
sns.set(font_scale=1.5) # for label/font scale size
sns.heatmap(ranking_percentiles, annot=True)
plt.yticks(rotation='horizontal', fontsize=15)
plt.xticks(rotation=45, ha='center', fontsize=15)
plt.title('Ranking Percentiles', fontsize=15)
```

Out[4]:
```
<matplotlib.text.Text at 0x1190c6350>
```



1.4.2 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:

From the samples, I observed that: + Customer 1: * It consumes a large proportional amount on all categories. This amount is moderate, compared to customer 2. So, it could be a **grocery store/retailer** that can consume this amount in every category.

- Customer 2:

  - It consumes a very large proportional amound on all categories, only **supermarket** can consume such a very large amount.

- Customer 3:
  - It consumes particularly on Fresh and Frozen, so it could be a **healthy food restaurant**.

1.4.3 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 [5]: from sklearn.cross_validation import train_test_split
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.metrics import accuracy_score

        import matplotlib.pyplot as plt

        # TODO: Make a copy of the DataFrame, using the 'drop' function to drop the given featur
        features = ['Fresh','Milk','Grocery','Frozen','Detergents_Paper','Delicatessen']

        for i in features:
            new_data = data.drop(i, axis=1)

            # TODO: Split the data into training and testing sets(0.25) using the given feature
            # Set a random state.
            X_train, X_test, y_train, y_test = train_test_split(new_data, data[i], test_size=0.2

            tem = 0
            tem1 =""
            tem2 = 0
            tem4 = 0
            # TODO: Create a decision tree regressor and fit it to the training set
            for j in range(3,12):
                regressor = DecisionTreeRegressor(max_depth=j, random_state=100)
                regressor.fit(X_train, y_train)
                # TODO: Report the score of the prediction using the testing set
                score = regressor.score(X_test,y_test)
                print "The prediction score is {} for {} at max depth {}".format(score, i, j);
```

```
                if tem < score:
                    tem = score
                    tem1 = i
                    tem2 = j
                elif score < 0:
                    tem4 = score
                    tem1 = i
                    tem2 = j
            if tem !=0:
                print "The optimal decision tree model at max depth {} for {} scores {}".format(
```

```
The prediction score is 0.184191659243 for Fresh at max depth 3
The prediction score is 0.0897494084495 for Fresh at max depth 4
The prediction score is 0.105466433425 for Fresh at max depth 5
The prediction score is 0.228553973771 for Fresh at max depth 6
The prediction score is 0.0348549265844 for Fresh at max depth 7
The prediction score is 0.0501216863088 for Fresh at max depth 8
The prediction score is 0.0688860118778 for Fresh at max depth 9
The prediction score is -0.190602190117 for Fresh at max depth 10
The prediction score is -0.139481478116 for Fresh at max depth 11
The optimal decision tree model at max depth 11 for Fresh scores 0.228553973771
The prediction score is -0.0516785644896 for Milk at max depth 3
The prediction score is 0.18281347396 for Milk at max depth 4
The prediction score is -0.513749798155 for Milk at max depth 5
The prediction score is 0.225422032383 for Milk at max depth 6
The prediction score is 0.223272629223 for Milk at max depth 7
The prediction score is -0.269793888287 for Milk at max depth 8
The prediction score is -0.454964102893 for Milk at max depth 9
The prediction score is -0.400883210559 for Milk at max depth 10
The prediction score is -0.0221745100985 for Milk at max depth 11
The optimal decision tree model at max depth 11 for Milk scores 0.225422032383
The prediction score is 0.661129426721 for Grocery at max depth 3
The prediction score is 0.699354501209 for Grocery at max depth 4
The prediction score is 0.67309918032 for Grocery at max depth 5
The prediction score is 0.728059704549 for Grocery at max depth 6
The prediction score is 0.68160518348 for Grocery at max depth 7
The prediction score is 0.616061035865 for Grocery at max depth 8
The prediction score is 0.592473019184 for Grocery at max depth 9
The prediction score is 0.597405296218 for Grocery at max depth 10
The prediction score is 0.62177968831 for Grocery at max depth 11
The optimal decision tree model at max depth 6 for Grocery scores 0.728059704549
The prediction score is 0.12143597108 for Frozen at max depth 3
The prediction score is 0.0541412294285 for Frozen at max depth 4
The prediction score is -0.156355787841 for Frozen at max depth 5
The prediction score is 0.165474505574 for Frozen at max depth 6
The prediction score is 0.331933876374 for Frozen at max depth 7
The prediction score is 0.0914983172673 for Frozen at max depth 8
```

```
The prediction score is 0.0660188791561 for Frozen at max depth 9
The prediction score is 0.339600910157 for Frozen at max depth 10
The prediction score is 0.112287340954 for Frozen at max depth 11
The optimal decision tree model at max depth 10 for Frozen scores 0.339600910157
The prediction score is 0.657504501306 for Detergents_Paper at max depth 3
The prediction score is 0.743874041318 for Detergents_Paper at max depth 4
The prediction score is 0.73328363829 for Detergents_Paper at max depth 5
The prediction score is 0.771320551296 for Detergents_Paper at max depth 6
The prediction score is 0.751478896299 for Detergents_Paper at max depth 7
The prediction score is 0.779771328181 for Detergents_Paper at max depth 8
The prediction score is 0.709225416332 for Detergents_Paper at max depth 9
The prediction score is 0.739700110203 for Detergents_Paper at max depth 10
The prediction score is 0.749033254188 for Detergents_Paper at max depth 11
The optimal decision tree model at max depth 8 for Detergents_Paper scores 0.779771328181
The prediction score is -4.8562415894 for Delicatessen at max depth 3
The prediction score is -5.0119477854 for Delicatessen at max depth 4
The prediction score is -4.75701660602 for Delicatessen at max depth 5
The prediction score is -5.35884267857 for Delicatessen at max depth 6
The prediction score is -4.80356949618 for Delicatessen at max depth 7
The prediction score is -5.2296935662 for Delicatessen at max depth 8
The prediction score is -5.58933389388 for Delicatessen at max depth 9
The prediction score is -5.38065506066 for Delicatessen at max depth 10
The prediction score is -5.68145383653 for Delicatessen at max depth 11


/Users/tungthanhle/anaconda2/lib/python2.7/site-packages/sklearn/cross_validation.py:44: Depreca
  "This module will be removed in 0.20.", DeprecationWarning)
```

1.4.4 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:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
score	0.2285	0.2254	**0.7280**	0.3396	**0.7797**	-4.7570
max depth	11	11	6	10	8	5

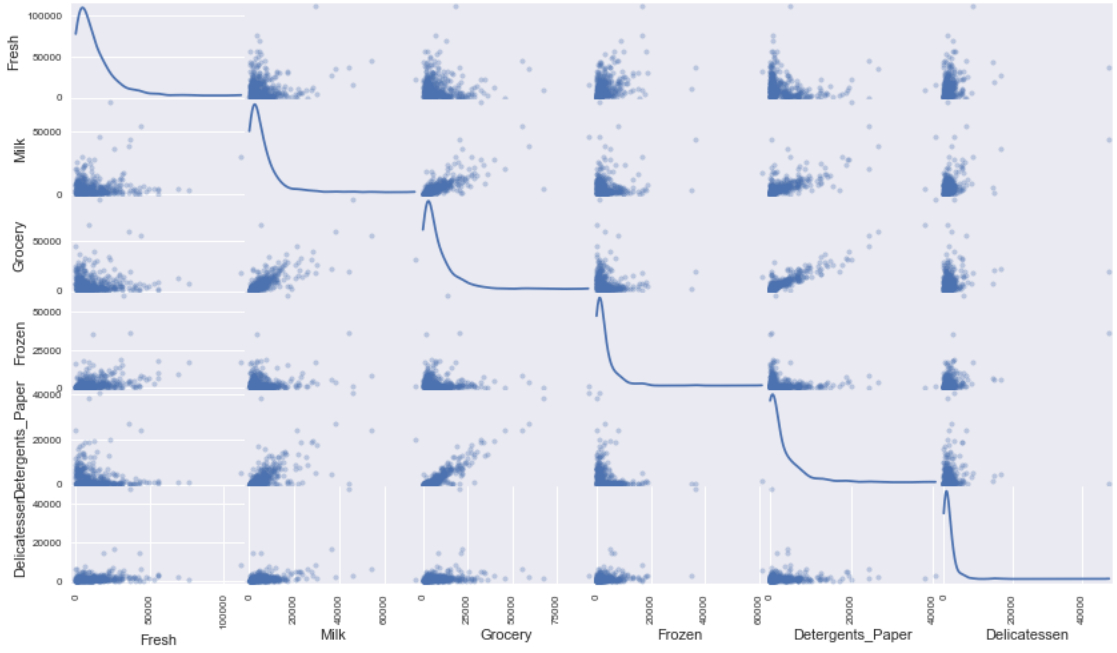
I observe that, `Delicatessen` failed to fit the decision tree learning model at any max depth. `Grocery` and `Detergents_Paper` seem likely to be strongly predictable by `0.72` and `0.77` corresponding to the max depth of `8` and `6`, respectively, but they are not necessary to be the features

for identifying customers' spending habits. `Frozen`, `Fresh`, and `Milk` have scores of `0.33`, `0.22`, and `0.22`, corresponding to the max depth of 10, 11, and 11, respectively.

1.4.5 Visualize Feature Distributions

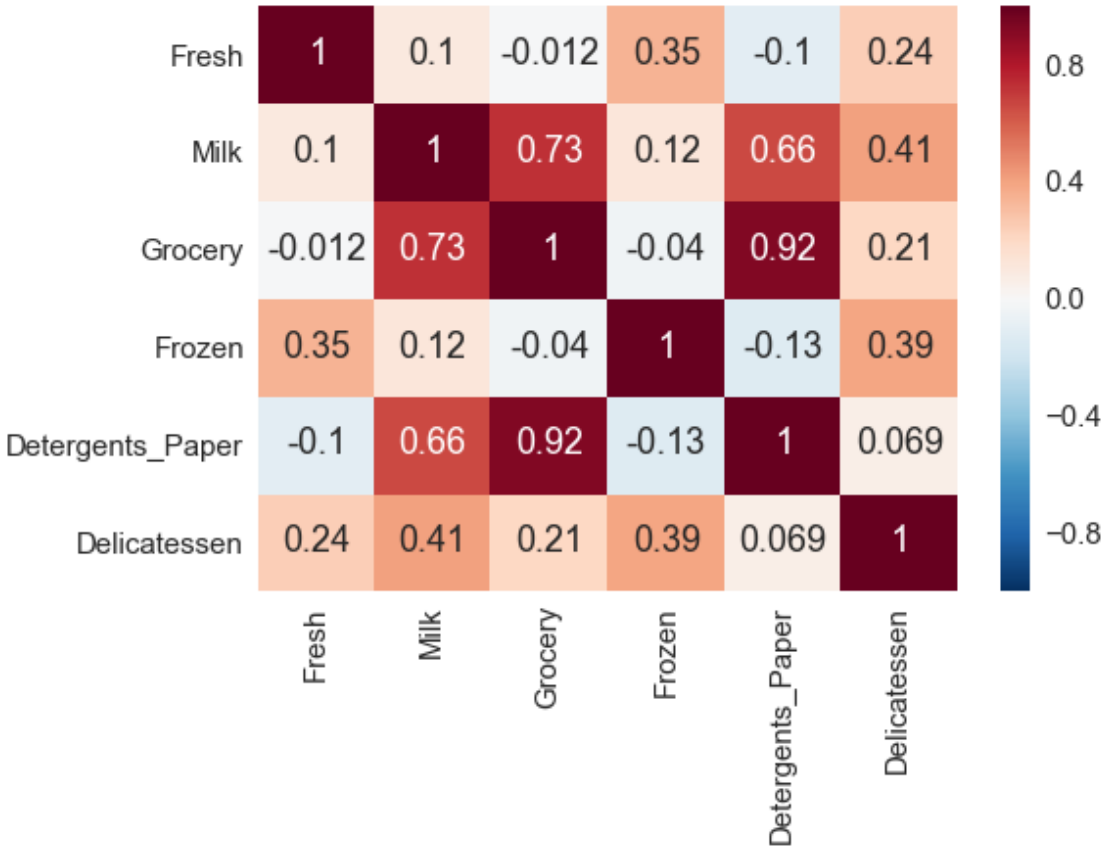
To get a better understanding of the dataset, we can construct a scatter matrix of each of the six product features present in the data. If you found that the feature you attempted to predict above is relevant for identifying a specific customer, then the scatter matrix below may not show any correlation 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 [6]:
```
# Produce a scatter matrix for each pair of features in the data
sns.set(font_scale=1.) # for label/font scale size
pd.scatter_matrix(data, alpha = 0.3, figsize = (14,8), diagonal = 'kde');
```

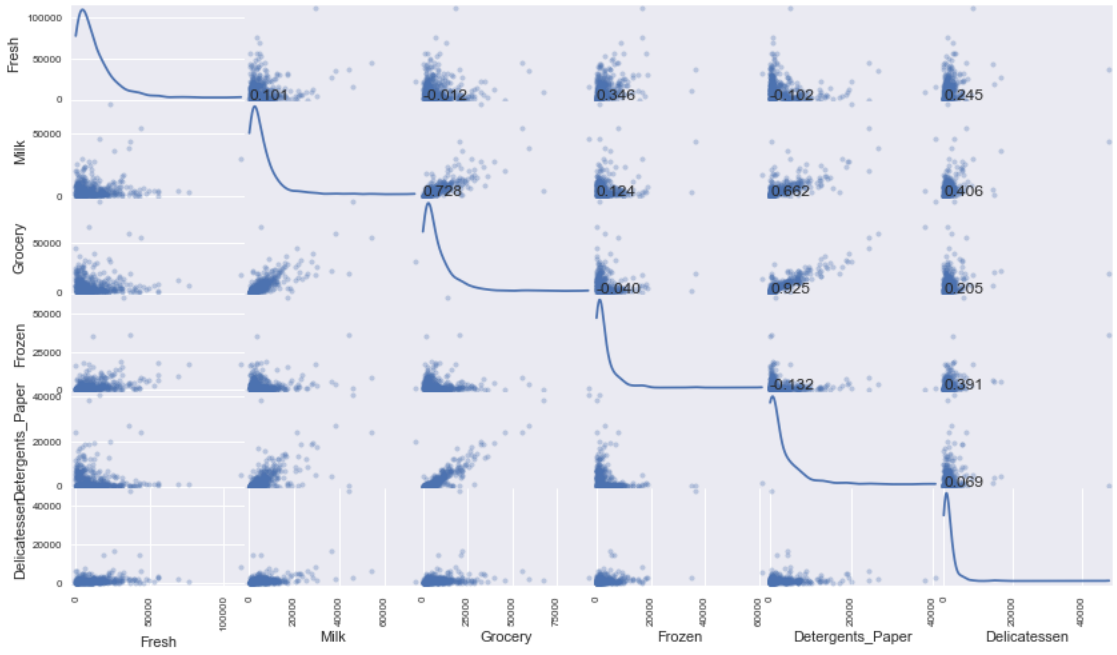


In [7]:
```
import seaborn as sns
sns.set(font_scale=1.5) # for label/font scale size
sns.heatmap(data.corr(), annot=True)
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11b073ed0>
```




```
In [8]: sns.set(font_scale=1.) # for label/font scale size
        scatter_matrix = pd.scatter_matrix(data, alpha = 0.3, figsize = (14,8), diagonal = 'kde'
        correlation = data.corr().as_matrix()
        for i,j in zip(*np.triu_indices_from(scatter_matrix, k=1)):
            scatter_matrix[i,j].annotate("%.3f" %correlation[i,j], (0.8,0.8))
```



1.4.6 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() to get the feature correlations and then visualize them using a heatmap(the data that would be fed into the heatmap would be the correlation values, for eg: `data.corr()`) to gain further insight.

Answer:

- Yes, three pairs of features, that have high relevance, are (`Detergents_Paper`,`Grocery`), (`Grocery`,`Milk`), and (`Detergents_Paper`, `Milk`) with the correlations of `0.925`, `0.728`, and `0.662`, respectively.
- From the heatmap results, it reassures my suspicions where Grocery and Detergents_Papers have strongly correlations.
- From the scatter matrix plot, the features that have high correlation tend to have the distribution of outliers linearly, while the low relevant pairs of features do not.

1.5 Data Preprocessing

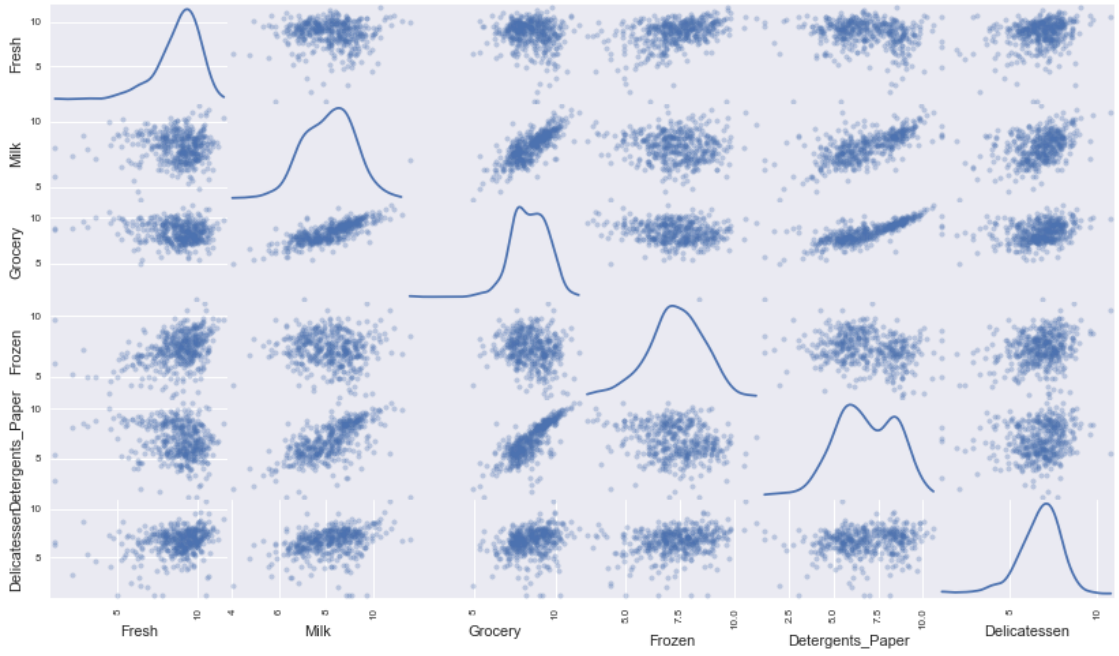
In this section, you will preprocess the data to create a better representation of customers by performing a scaling on the data and detecting (and optionally removing) outliers. Preprocessing data is often times a critical step in assuring that results you obtain from your analysis are significant and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

1.5.1 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 to apply a non-linear scaling — particularly for financial data. One way to achieve this scaling is by using a Box-Cox test, 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 [9]:
```
# TODO: Scale the data using the natural logarithm
log_data = np.log(data)

sns.set(font_scale=1.) # for label/font scale size
# 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');
```



1.5.2 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 [10]: # Display the log-transformed sample data
         display(log_samples)
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	8.933137	8.508354	9.151227	7.419980	8.108021	7.850104
1	10.702480	10.901524	10.925417	8.959569	10.092909	8.774158
2	9.515543	5.846439	6.192362	7.638680	3.784190	6.490724

1.5.3 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: 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`. * Interquartile Range (IQR): we subtract the 3rd Quartile from the 1st Quartile.

```
In [11]: from itertools import combinations
         lists = []

         # 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 feature
             Q1 = np.percentile(log_data[feature], 25)

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

             # TODO: Use the interquartile range to calculate an outlier step (1.5 times the int
```

```
            step = (Q3 - Q1)*1.5 # IQR

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

        # OPTIONAL: Select the indices for data points you wish to remove
        outliers  = []
        for x,y in list(combinations(range(0,len(lists)),2)):
            intersect = list((set(lists[x]).union(set(lists[y])))^( set(lists[x])^set(lists[y]))
            outliers = list(set(outliers).union(set(intersect)))

        outliers.sort()

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

```
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	\
66	2.197225	7.335634	8.911530	5.164786	8.151333	
109	7.248504	9.724899	10.274568	6.511745	6.728629	
128	4.941642	9.087834	8.248791	4.955827	6.967909	
137	8.034955	8.997147	9.021840	6.493754	6.580639	
142	10.519646	8.875147	9.018332	8.004700	2.995732	
154	6.432940	4.007333	4.919981	4.317488	1.945910	
183	10.514529	10.690808	9.911952	10.505999	5.476464	
184	5.789960	6.822197	8.457443	4.304065	5.811141	
187	7.798933	8.987447	9.192075	8.743372	8.148735	
203	6.368187	6.529419	7.703459	6.150603	6.860664	
233	6.871091	8.513988	8.106515	6.842683	6.013715	
285	10.602965	6.461468	8.188689	6.948897	6.077642	
289	10.663966	5.655992	6.154858	7.235619	3.465736	

```
343   7.431892   8.848509  10.177932   7.283448          9.646593
```

```
     Delicatessen
66       3.295837
109      1.098612
128      1.098612
137      3.583519
142      1.098612
154      2.079442
183     10.777768
184      2.397895
187      1.098612
203      2.890372
233      1.945910
285      2.890372
289      3.091042
343      3.610918
```

In [12]:
```
# Heatmap using percentiles to display outliers
import matplotlib.pyplot as plt
import seaborn as sns

ranking_percentiles = data.rank(axis=0, pct=True).iloc[outliers]*100
sns.set(font_scale=1.5) # for label/font scale size
sns.heatmap(ranking_percentiles, annot=True)
plt.yticks(rotation='horizontal', fontsize=15)
plt.xticks(rotation=45, ha='center', fontsize=15)
plt.title('Outliers Ranking Percentiles', fontsize=15)
```

Out[12]:
```
<matplotlib.text.Text at 0x11ca7d750>
```

1.5.4 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: * From the heatmap, we have 5 datapoints considered outliers for more than one feature. * Outlier 65 in 2 features: Fresh, Frozen. * Outlier 66 in 2 features: Fresh, Delicatessen. * Outlier 75 in 2 features: Grocery, Detergents_Paper. * Outlier 128 in 2 features: Fresh, Delicatessen. * Outlier 154 in 3 features: Milk, Grocery, Detergents_Paper.

- The outliers datapoints should be removed from the dataset since they are outside of the expected range and skewing the dataset.
- The added datapoints that fall into the outliers range should be removed from the dataset, since it would allow us to clusterize easily the dataset features.

1.6 Feature Transformation

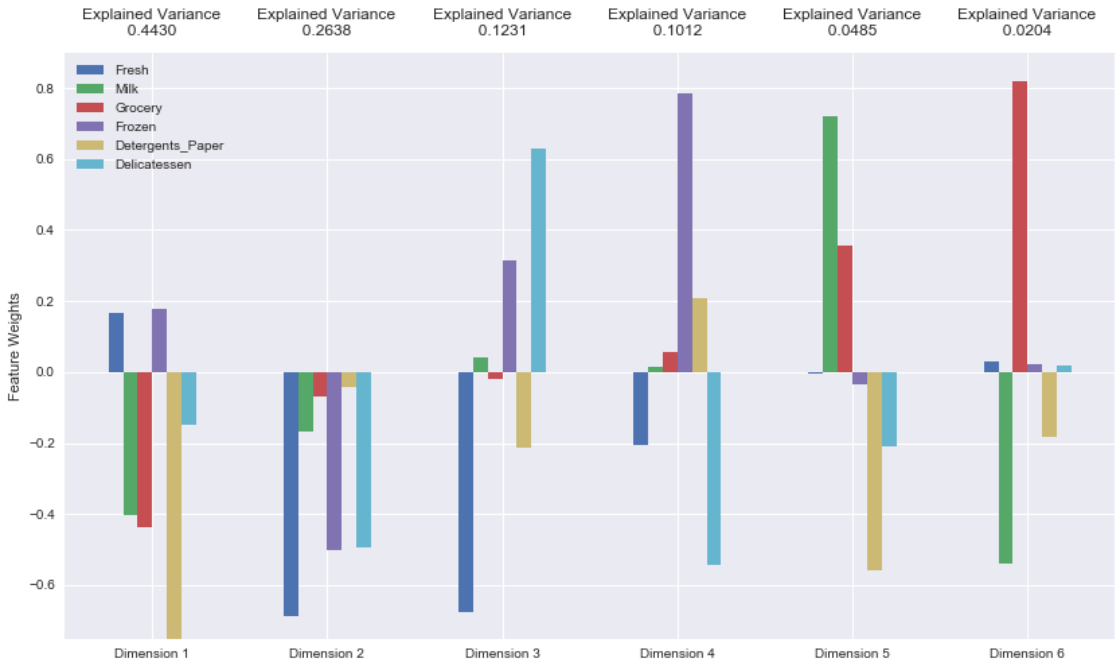
In this section you will use principal component analysis (PCA) to draw conclusions about the underlying structure of the wholesale customer data. Since using PCA on a dataset calculates the dimensions which best maximize variance, we will find which compound combinations of features best describe customers.

1.6.1 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 [13]: from sklearn.decomposition import PCA

         sns.set(font_scale=1.) # for label/font scale size
         # TODO: Apply PCA by fitting the good data with the same number of dimensions as featur
         pca = PCA(n_components=6)

         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)

         print "PCA dimensions in the first and second cumulative sums/principle components: ",
         print "PCA dimensions in the first, second, third, and fourth cumulative sums/principle

PCA dimensions in the first and second cumulative sums/principle components:  0.7068
PCA dimensions in the first, second, third, and fourth cumulative sums/principle components:  0.
```



1.6.2 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.

Answer: * The total variance in the dataset by 1st and 2nd principal components is `70.68%`. With the third and fourth principal components are added, the total variance in the dataset is `93.11%`.

- The first dimension represents some customers who purchased a large proportional amount of Detergents_Paper, Milk, and Grocery. This dimension could likely be represented by retailers' spending.
- The second dimension represents customers who bought a large amount of Fresh, Frozen, and Delicatessen. This dimension could be big restaurants' spending.
- The third dimension represents customers who purchased a large amount of Fresh and Delicatessen. This could represent grocery stores' spending.

- The fourth dimension represents customers who bought a lot of Frozen and Delicatessen. This could be food stores' spending.
- Based on the analysis, the top important features are Fresh, Delicatessen, Frozen. Other features like Milk, Detergents_Paper, and Grocery are easy to predict relatively based on other features. Thus, first four dimensions can be a good representation in terms of customer spending.

1.6.3 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 [14]: # Display sample log-data after having a PCA transformation applied
         display(pd.DataFrame(np.round(pca_samples, 4), columns = pca_results.index.values))
```

```
   Dimension 1  Dimension 2  Dimension 3  Dimension 4  Dimension 5  \
0      -1.5672      -0.9010       0.3684      -0.2682      -0.4571
1      -4.3646      -3.9519      -0.1229       0.6240       0.5379
2       4.3870       0.0997       0.0545      -0.5833      -0.7342

   Dimension 6
0       0.1526
1       0.0551
2      -0.0541
```

1.6.4 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 [15]: # TODO: Apply PCA by fitting the good data with only two dimensions
         pca = PCA(n_components=2)
         pca.fit(good_data)

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

```
        # TODO: 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', 'Dimension 2'])
```

1.6.5 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 [16]: # Display sample log-data after applying PCA transformation in two dimensions
         display(pd.DataFrame(np.round(pca_samples, 4), columns = ['Dimension 1', 'Dimension 2']
```

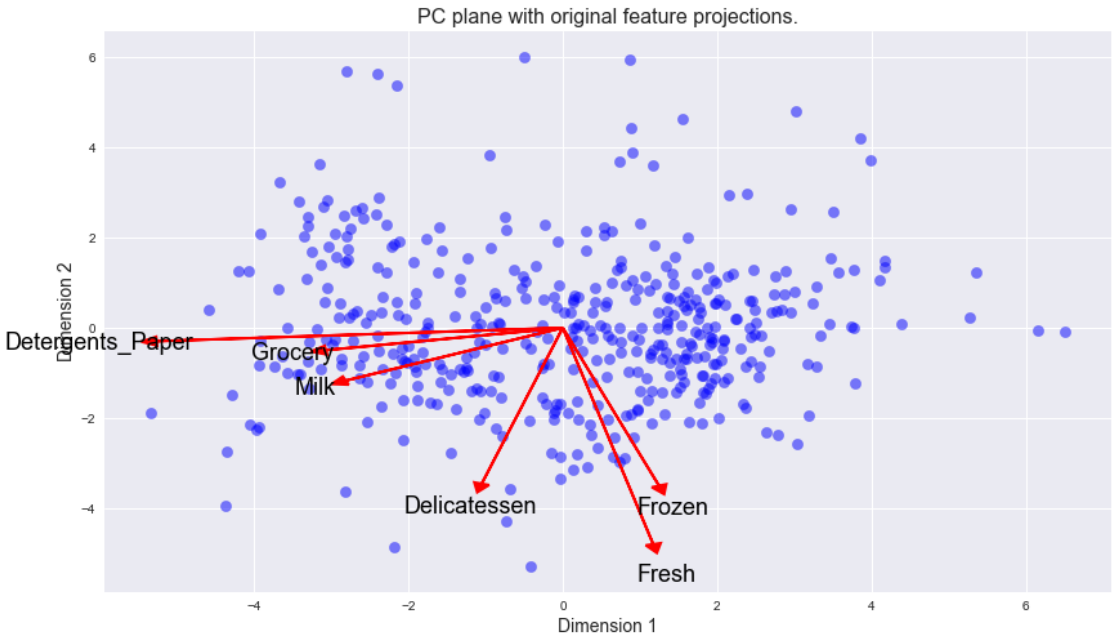
```
   Dimension 1  Dimension 2
0      -1.5672      -0.9010
1      -4.3646      -3.9519
2       4.3870       0.0997
```

1.7 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 [17]: # Create a biplot
         vs.biplot(good_data, reduced_data, pca)

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x11d789910>
```



1.7.1 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?

1.8 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.

1.8.1 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: * The advantages of K-Means: * Fast and efficiency in terms of computational complexity. * Easy to interpret the clustering results.

- The advantages of GMM:
  - Maximize likelihood.
  - Not bias the clustering sizes to specific structures.
- **K-Means algorithm** would be used since the dataset could be easily split into clusters.

1.8.2 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 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 [18]: from sklearn.cluster import KMeans
         from sklearn.metrics import silhouette_score
         # TODO: Apply your clustering algorithm of choice to the reduced data

         cluster = KMeans(n_clusters=2, random_state=100)
         cluster.fit(reduced_data)

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

         # TODO: Find the cluster centers
         centers = cluster.cluster_centers_

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

         # TODO: Calculate the mean silhouette coefficient for the number of clusters chosen
         score = silhouette_score(reduced_data, preds)
         print score
```

```
0.426281015469
```

1.8.3 Question 7

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

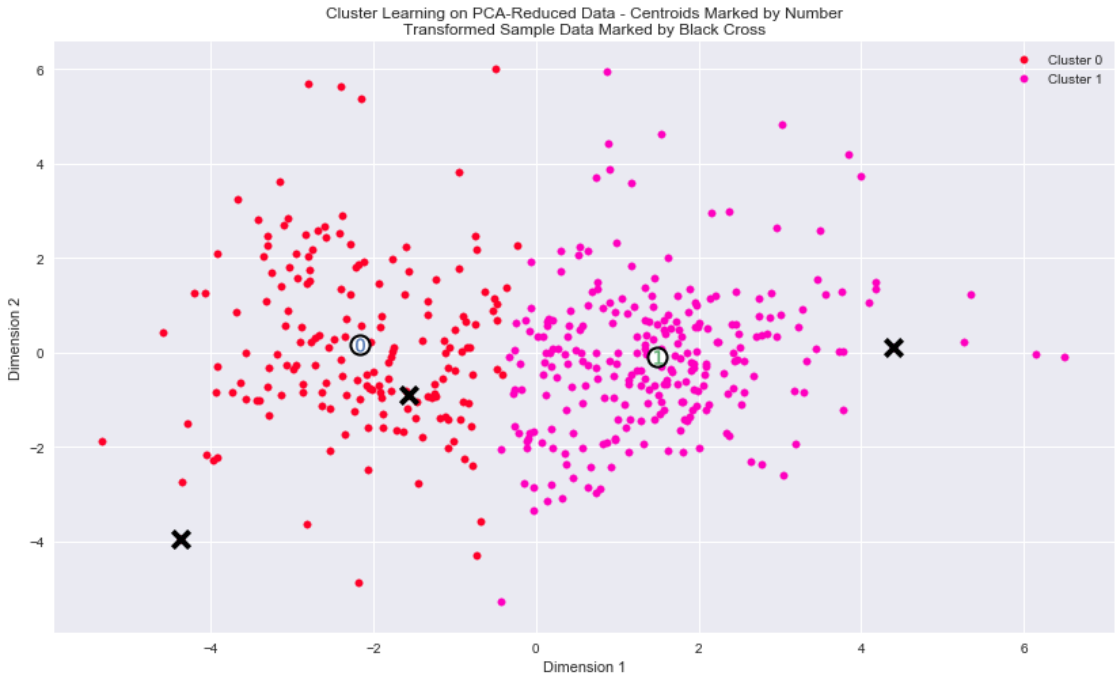
Answer: * I have tested various numbers of n_clusters: * n_clusters=2 has score of **0.42628**. * n_clusters=3 has score of 0.39742. * n_clusters=4 has score of 0.33075. * n_clusters=5 has score of 0.35220.

- The number of clusters (=2) has the best silhouette score is 0.42628.

1.8.4 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 [19]:
```
# Display the results of the clustering from implementation
vs.cluster_results(reduced_data, preds, centers, pca_samples)
```



1.8.5 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.

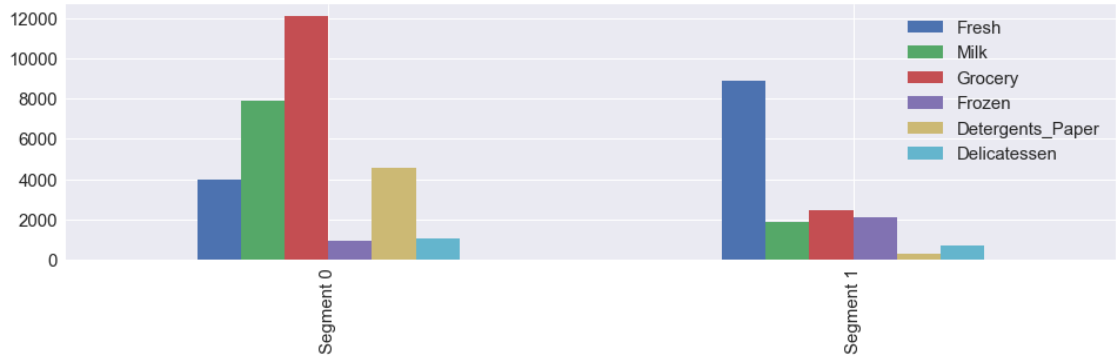
In the code block below, you will need to implement the following: - Apply the inverse transform to `centers` using `pca.inverse_transform` and assign the new centers to `log_centers`. - Apply the inverse function of `np.log` to `log_centers` using `np.exp` and assign the true centers to `true_centers`.

```
In [20]: # TODO: Inverse transform the centers
         log_centers = pca.inverse_transform(centers)

         # TODO: 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	4005.0	7900.0	12104.0	952.0	4561.0	1036.0
Segment 1	8867.0	1897.0	2477.0	2088.0	294.0	681.0

```
In [24]: sns.set(font_scale=1.5)
         true_centers.plot(kind = 'bar', figsize = (16, 4))
```

```
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x11c828c90>
```



1.8.6 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.

Answer: * Segment 0 could probably represent grocery stores' segment since it consumes a lot of Milk, Grocery.

- Segment 1 could probably be retailers' segment since it consumes a large proportional amount of Fresh, Grocery, and Frozen.

1.8.7 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 [25]: # 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 0
Sample point 1 predicted to be in Cluster 0
Sample point 2 predicted to be in Cluster 1
```

Answer:

- Sample point 0 is predicted in cluster 0, which may represent a coffee shop. It has the high costs on Grocery and Milk. This fits to segment 0.
- Sample point 1 is predicted in cluster 0, which may represent a grocery store. It has the high costs on Milk, Delicatessen. This fits to segment 0.
- Sample point 2 is predicted in cluster/segment 1, which may represent a retailer. It has the high costs on Frozen. This fits to segment 1.

1.9 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.

1.9.1 Question 10

Companies will often run A/B tests 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:

- To determine how the changes in delivery schedule can affect the distributor's customers, A/B tests can be run on customer segments separately. By randomly select several sample points on each segment to change their delivery schedule service from five to three days a week. The reaction of the customer segments can be observed to determine whether the changes are applicable for the rest of customers.
- My guess is the `Fresh` in segment 1 could likely be affected by this change.
  - In the first A/B test, the control group is the customers in segment 1 and the original delivery schedule service. The test group is the customers in segment 1 and the changed delivery schedule service.
  - In the second A/B test, the control group is the customers in segment 0 and the original delivery schedule service. The test group is the customers in segment 0 and the changed delivery schedule service.
- My expectation of the tests is, the first test should be the different, and the second test should be the same. From the results, I can determine whether my guess is correct or not.

1.9.2 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?

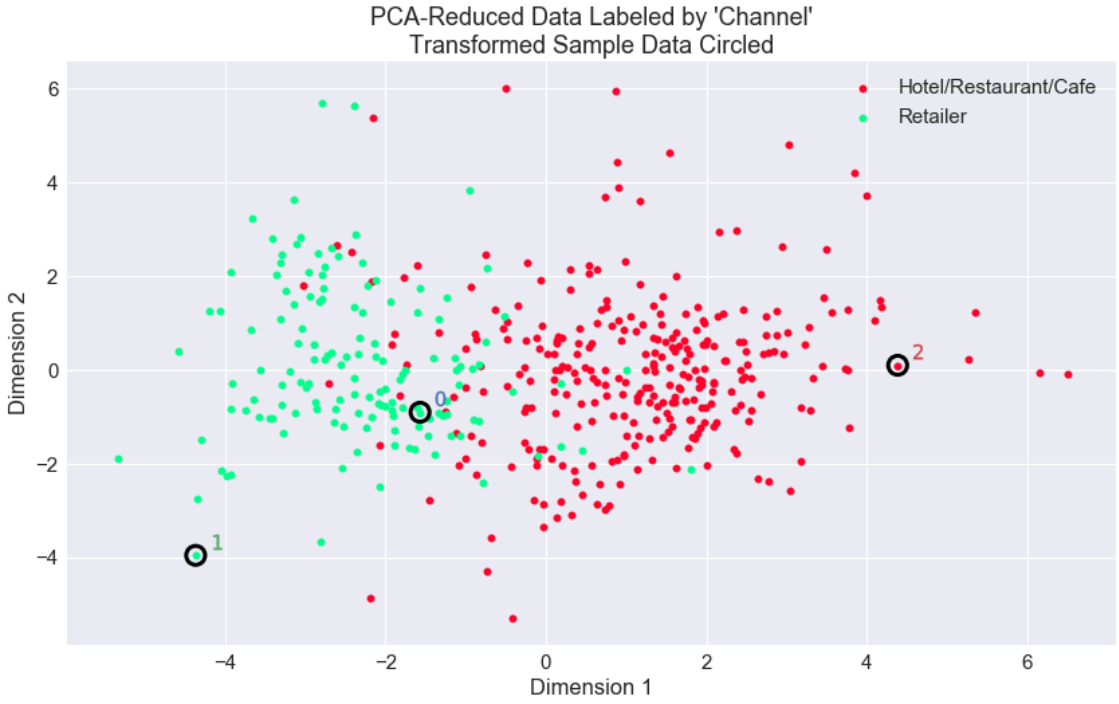
Answer: * The segments information can be labeled as segmentation features, and we can use the supervised learning algorithms such as SVM or K-nearest neighbors (KNN) for classification. Thereby, the classifiers can classify the incoming data easily and faster.

1.9.3 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 [26]: # Display the clustering results based on 'Channel' data
         vs.channel_results(reduced_data, outliers, pca_samples)
```



1.9.4 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 results from K-Means clustering are consistent to the retailers on sample point 2, grocery and coffee shop on sample points 1 and 0, respectively.

- The customer segments are overlapped.

- Yes. My definition on segment 0 is groceries/coffee shops and segment 1 is retailers, which are consistent to this classification.

  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.