customer_segments

June 29, 2016

1 Machine Learning Engineer Nanodegree

1.1 Unsupervised Learning

1.2 Project 3: 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 [4]: # Import libraries necessary for this project
        import numpy as np
        import pandas as pd
        import renders as rs
        from IPython.display import display # Allows the use of display() for DataF

        # Show matplotlib plots inline (nicely formatted in the notebook)
        %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
        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 [3]: # 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 [5]: # TODO: Select three indices of your choice you wish to sample from the dat
        indices = [2,220,178]

        # Create a DataFrame of the chosen samples
        samples = pd.DataFrame(data.loc[indices], columns = data.keys()).reset_inde
        print "Chosen samples of wholesale customers dataset:"
        display(samples)
        print "Deviation from mean"
        display(samples - np.round(data.mean()))
        print "Deviation from median"
        display(samples - np.round(data.median()))
```

Chosen samples of wholesale customers dataset:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	6353	8808	7684	2405	3516	7844
1	14755	899	1382	1765	56	749
2	11002	7075	4945	1152	120	395

Deviation from mean

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	-5647.0	3012.0	-267.0	-667.0	635.0	6319.0
1	2755.0	-4897.0	-6569.0	-1307.0	-2825.0	-776.0
2	-998.0	1279.0	-3006.0	-1920.0	-2761.0	-1130.0

Deviation from median

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	-2151.0	5181.0	2928.0	879.0	2700.0	6878.0
1	6251.0	-2728.0	-3374.0	239.0	-760.0	-217.0
2	2498.0	3448.0	189.0	-374.0	-696.0	-571.0

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, and retailers, among many others. Avoid using names for establishments, such as saying *"McDonalds"* when describing a sample customer as a restaurant.

Answer: Customer with index 2 would represent the Delicatessen retailers. Likewise customer 220 would represent grocery stores where fresh items are sold and customer 178 would represent could represent a larger and inclusive market since his/her shopping needs are in the middle in comparison to the other indices.

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 [4]: from sklearn.cross_validation import train_test_split
        from sklearn import tree
        # TODO: Make a copy of the DataFrame, using the 'drop' function to drop the
        new_data = pd.read_csv("customers.csv")
        new_data.drop(['Region', 'Channel'], axis = 1, inplace = True)
        y_data = new_data['Delicatessen']
        new_data.drop(['Delicatessen'], axis = 1, inplace = True)


        # TODO: Split the data into training and testing sets using the given featu
        X_train, X_test, y_train, y_test = train_test_split(new_data, y_data, test_

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

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

        print regressor.score(X_test, y_test)
```

```
-2.2547115372
```

1.4.4 Question 2

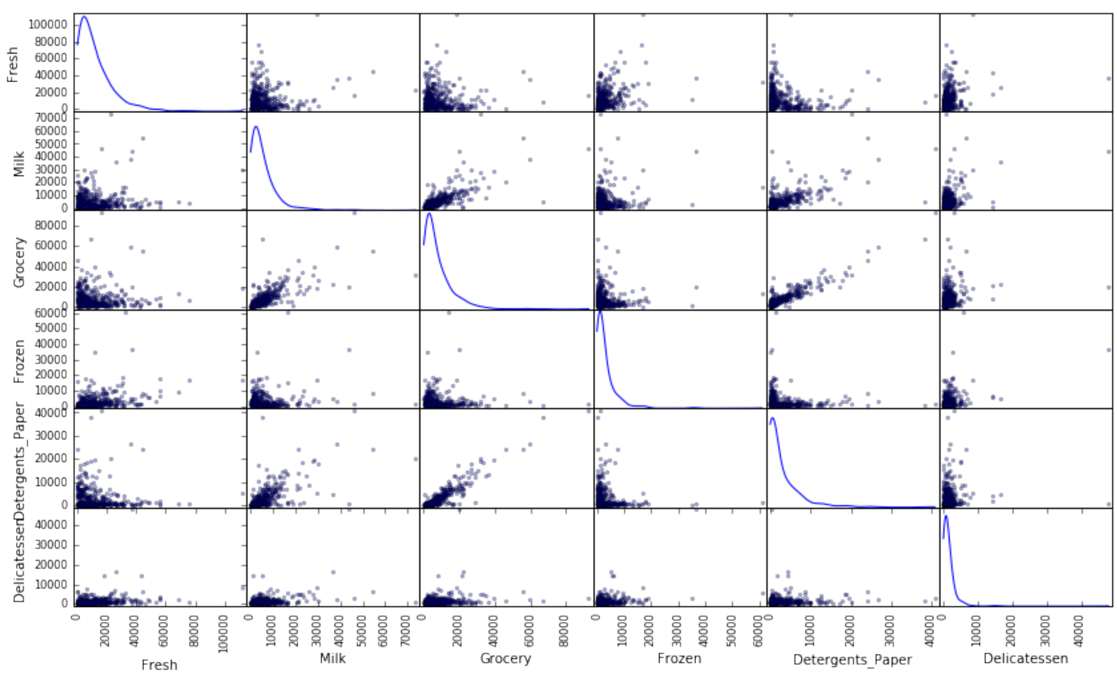
Which feature did you attempt to predict? What was the reported prediction score? Is this feature is necessary for identifying customers' spending habits?
Hint: The coefficient of determination, `R^2`, is scored between 0 and 1, with 1 being a perfect fit. A negative `R^2` implies the model fails to fit the data.

Answer: The feature attempted to predict was Delicatessen, and the regressor score returned was -2.254711. Due to the low score, I would classify this feature unnecessary for identifying customers' spending habits. Since we're dealing with a low R^2 score, the value cannot be determined based on the values of other features. Therefore if the value was not in the dataset, then we wouldn't be able to derive the value.

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
        pd.scatter_matrix(data, alpha = 0.3, figsize = (14,8), diagonal = 'kde');
```



1.4.6 Question 3

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?

Answer: Yes there are a few features that have some degree of correlation. For example detergents_paper and groceries have a strong correlation in comparison with all scatterplots. This confirms my suspicion that some items are statistically coorelated with other items. Although it denies my suspicion that Delicatessen has any statistical correlation with other items. This may be because, this particular item is a specality which is not related to the other more common items. The data for Delicatessen is distributed with the other 5 items around the bottom left corner. The scatterplot looks normally distributed but the curve being shifted to the left, ie a strong skew.

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

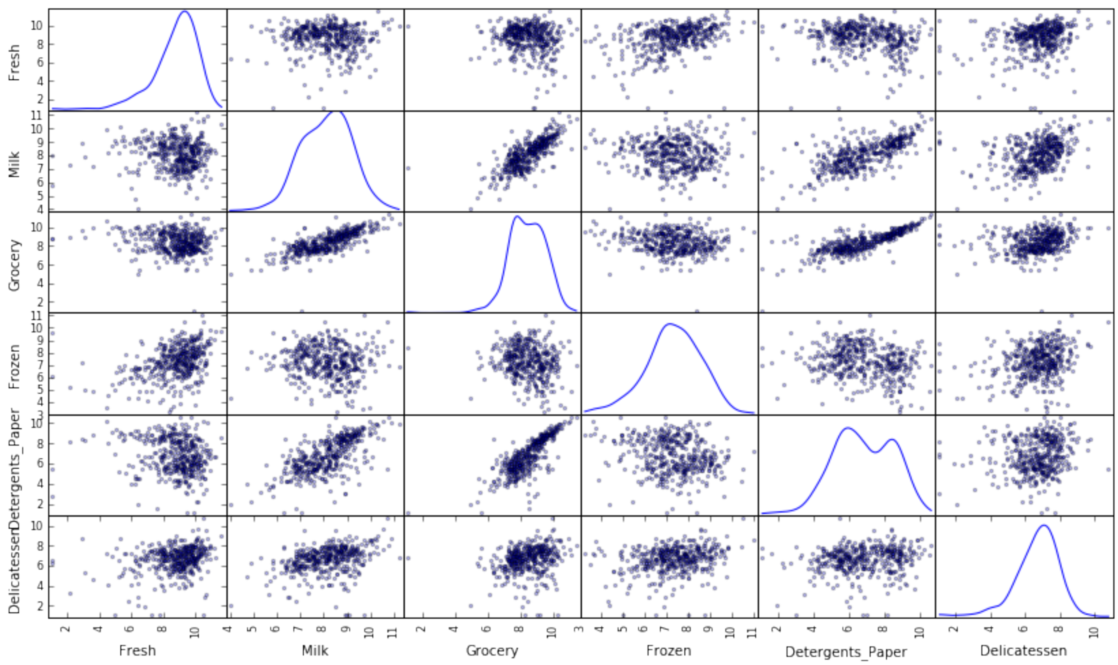
If data is not normally distributed, especially if the mean and median vary significantly (indicating a large skew), it is most often appropriate to apply a non-linear scaling — particularly for financial data. One way to achieve this scaling is by using a Box-Cox test, which calculates the best power 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 a logarithm scaling. Use the `np.log` function for this. - Assign a copy of the sample data to `log_samples` after applying a logrithm 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'
```



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

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	8.756682	9.083416	8.946896	7.785305	8.165079	8.967504
1	9.599337	6.801283	7.231287	7.475906	4.025352	6.618739
2	9.305832	8.864323	8.506132	7.049255	4.787492	5.978886

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`.

In [8]:
```
# OPTIONAL: Select the indices for data points you wish to remove
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 featu
    Q1, Q3 = np.percentile(log_data[feature], [25 ,75])
    iqr = Q3-Q1

    # TODO: Calculate Q3 (75th percentile of the data) for the given featu
    #Q3 = None

    # TODO: Use the interquartile range to calculate an outlier step (1.5 t
    step = 1.5*iqr

    # Display the outliers
    print "Data points considered outliers for the feature '{}':".format(fe
    display(log_data[~((log_data[feature] >= Q1 - step) & (log_data[feature
    outlierVals = log_data[~((log_data[feature] >= Q1 - step) & (log_data[f
    for val in outlierVals:
            outliers.append(val)
print "Here are all the outliers:"
print outliers
print "There are repeats in the outlier list, which were removed:"
completedOutliers=[];
for ii, val in enumerate(outliers):
        if val not in completedOutliers:
                for jj, check in enumerate(outliers):
                        if ii != jj and val == check and val not in complet
                                completedOutliers.append(val)
print completedOutliers
# Remove the outliers, if any were specified
good_data = log_data.drop(log_data.index[completedOutliers]).reset_index(dr
```

```
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
```

```
Here are all the outliers:
[65, 66, 81, 95, 96, 128, 171, 193, 218, 304, 305, 338, 353, 355, 357, 412, 86, 98,
```

```
There are repeats in the outlier list, which were removed:
[65, 66, 128, 154, 75]
```

1.5.4 Question 4

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

Answer: Yes some data points occur as outliers for more than one of its features. These data points should be removed so that the model can form to identify more significant occurances. In fact these data points were removed so that the model formed not representing insignificantly rare occurances.

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.

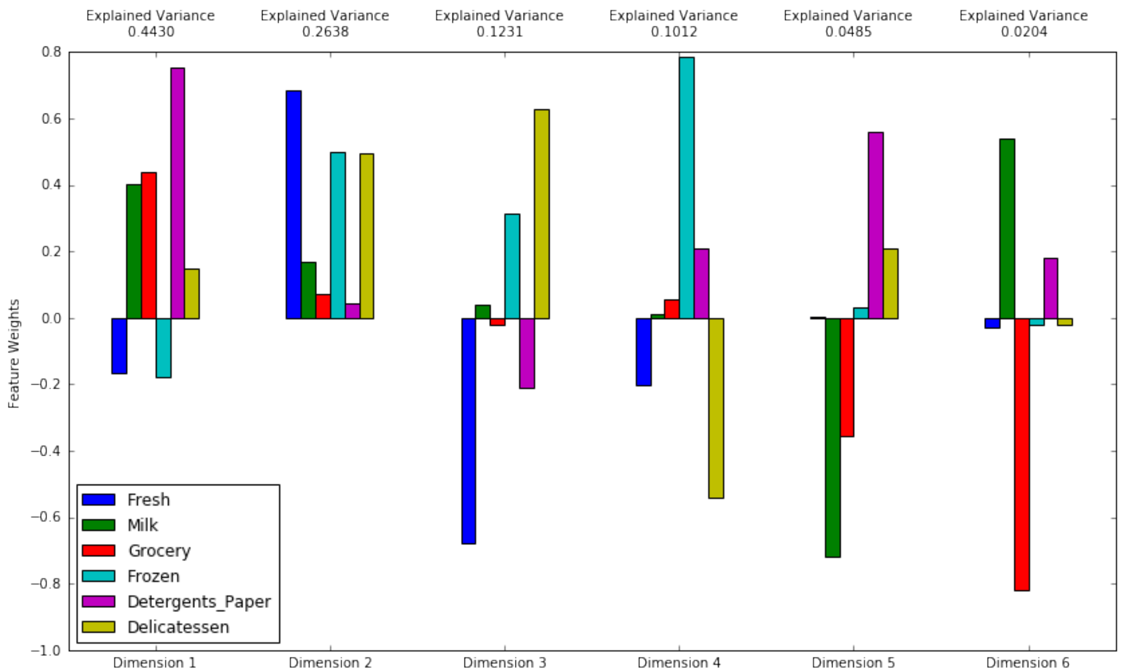
In the code block below, you will need to implement the following: - Import `sklearn.decomposition.PCA` and assign the results of fitting PCA in six dimensions with `good_data` to `pca`. - Apply a PCA transformation of the sample log-data `log_samples` using `pca.transform`, and assign the results to `pca_samples`.

```
In [51]: from sklearn.decomposition import PCA
         # TODO: Apply PCA to the good data with the same number of dimensions as f

         pca = PCA(n_components=6)
         pca = pca.fit(good_data)
         # TODO: Apply a PCA transformation to the sample log-data
         pca_samples = pca.transform(log_samples)

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

         #print '{} \n {}'.format(len(pca_samples),(pca_results))
```



1.6.2 Question 5

How much variance in the data is explained ***in total*** *by the first and second principal component? What about the first four principal components? Using the visualization provided above, 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 indivdual feature weights.

Answer: In total the first and second components hold .7068 of the total variance. The first four components hold .9311 of the total variance. We can see here PCA does a good job of finding components that match total feature variance in the first four components, in which even a larger percentage is held in the first two components. The first component is heavily weighed by Detergents_Paper, Milk, and Groceries. This could be explained as customers tend to buy these three things together, and pca finds this component as a new feature which was combined from the other features. Detergents_Paper provides the highest influest or largest weight. This first component holds .4430 of the explained variance. Next we have the second component which mainly constitues Fresh, Frozen and Delicatessen. The largest weight is Fresh. Next we have the third component which is composed of mainly Delicatessen and Frozen. Finally the fourth component mainly has Frozen and has the least amount of variance compared to the first 3 features. The variance of each component decreases in order with iteration.

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

```
   Dimension 1  Dimension 2  Dimension 3  Dimension 4  Dimension 5  \
0       1.8834       1.5991       1.3204      -0.5432       0.3934
1      -3.3331       0.1830      -0.0061      -0.6771      -0.1655
2      -1.3444      -0.0806      -0.4471      -0.3481      -1.8257

   Dimension 6
0       0.3117
1      -0.2403
2      -0.0029
```

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 reuslts to `reduced_data`. - Apply a PCA transformation of the sample log-data `log_samples` using `pca.transform`, and assign the results to `pca_samples`.

```
In [53]: # TODO: Fit PCA to the good data using only two dimensions
         pca = PCA(n_components = 2)
         pca.fit(good_data)

         # TODO: Apply a PCA transformation the good data
         reduced_data = pca.transform(good_data)

         # TODO: Apply a PCA transformation to the sample log-data
         pca_samples = pca.transform(log_samples)

         # Create a DataFrame for the reduced data
         reduced_data = pd.DataFrame(reduced_data, columns = ['Dimension 1', 'Dimen
         print reduced_data.columns
```

```
Index([u'Dimension 1', u'Dimension 2'], dtype='object')
```

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 [54]: # Display sample log-data after applying PCA transformation in two dimensi
         display(pd.DataFrame(np.round(pca_samples, 4), columns = ['Dimension 1', '
         print len(reduced_data)
```

```
   Dimension 1  Dimension 2
0       1.8834       1.5991
1      -3.3331       0.1830
2      -1.3444      -0.0806


435
```

1.7 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.

1.7.1 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?

Answer: >**General** > >K means algorithm finds m clusters through a series of iterations. Properties and advantages of the k means algorithm is it is sensitive to outliers, only uses numerical features, and proper for compact clusters. In the gaussian mixture model, clusters can overlap, clusters can be non circular, and can provide a generative model of a feature space. I will be using the gaussian mixture model since it seems inclusive due to overlapping flexibility and accesibility to identifying noncircular clusters. > >**Speed and scalability** > > I'm curious about this, please do provide insight in the review. > >**Cluster Assignment** > >K means provides a hard clustering system, while the gaussian mixture model along with the EM algorithm, provides a soft assignment. Gaussian mixture model and EM helps express this uncertainity while k means provides hard classified clusters according to how close the data point (minimize squared Euclidean distance) is to a cluster mean versus other cluster means.

1.7.2 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 [55]: from sklearn.mixture import GMM
         from sklearn.metrics import silhouette_score

         # Apply the selected clustering algorithm to the reduced data
         # Let's fit with 2 clusters and choose a random state to make our results
         clusterer = GMM(n_components=2, random_state=0)
         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 cho
         score = silhouette_score(reduced_data, preds)
         print score
```

```
0.411818864386
```

1.7.3 Question 7

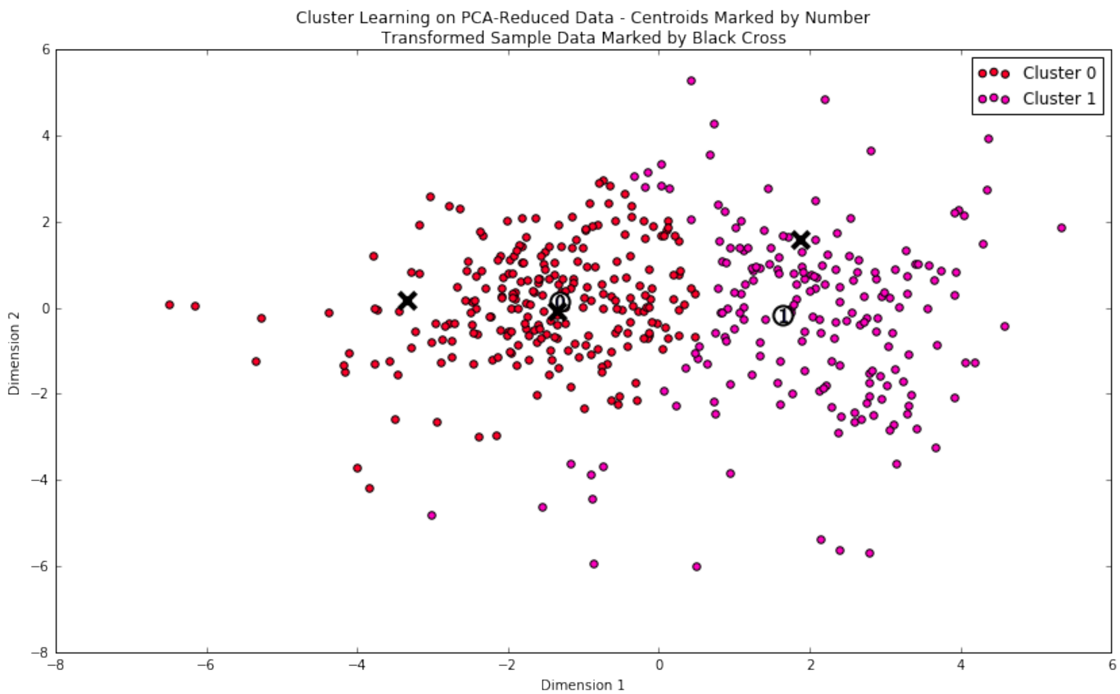
Report the silhouette score for several cluster numbers you tried. Of these, which number of clusters has the best silhouette score?

Answer: The silhoutte score reached for n_components 2 or creating 2 clusters was .4118. This was the best score. I tried other n_components where n_components 3,4,5 had scores 0.373560747175, 0.308243479507 and 0.295441470747 respectively. As the clustering increases, the silhoutte score lowers.

1.7.4 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 [56]: # Display the results of the clustering from implementation
         rs.cluster_results(reduced_data, preds, centers, pca_samples)
```



1.7.5 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 [57]: # 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())

display(true_centers - np.round(data.mean()))
display((true_centers - np.round(data.median())))

true_centers.index = segments
print 'true centers'
display(true_centers)
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	-3188.0	-3744.0	-5262.0	-1014.0	-2544.0	-813.0
1	-7684.0	551.0	1604.0	-2036.0	165.0	-580.0

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	308.0	-1575.0	-2067.0	532.0	-479.0	-254.0
1	-4188.0	2720.0	4799.0	-490.0	2230.0	-21.0

```
true centers
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
Segment 0	8812.0	2052.0	2689.0	2058.0	337.0	712.0
Segment 1	4316.0	6347.0	9555.0	1036.0	3046.0	945.0

1.7.6 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. *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'`.

Answer: A customer that is assigned to a certain cluster, exhibits characteristics of the feature set the cluster represents. The first center has true values 'Fresh':8812, 'Milk':2052, 'Grocery':2689, 'Frozen':2058.0, 'Detergents_Paper':337.0 and 'Delicatessen':712.0. Similarly component 2 has true values 'Fresh':4316.0, 'Milk':6347.0, 'Grocery':9555.0, 'Frozen':1036.0, 'Detergents_Paper':3046.0 and 'Delicatessen':945.0. When comparing the previous means and medians of the data from the beginning we find new center to be much higher values for the first component. The second mean of the cluster has less variance, and resembles less intense high values compared to the previous mean.

1.7.7 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 [58]: # Display the predictions
         for i, pred in enumerate(sample_preds):
             print "Sample point", i, "predicted to be in Cluster", pred
             #print 'sample preds = {}'.format(preds)
```

```
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
```

Answer: The sample points I chose were correctly classified to the respective clusters. sample point 0 predicted to be in cluster 1, sample point 1 predicted to be in cluster 0 and sample point 2 predicted to be in cluster 0. The predictions for each sample point are consistent with it's feature sets and cluster mean feature sets.

1.8 Conclusion

1.8.1 Question 10

Companies often run A/B tests when making small changes to their products or services. If the wholesale distributor wanted to change its delivery service from 5 days a week to 3 days a week, how would you use the structure of the data to help them decide on a group of customers to test?
Hint: Would such a change in the delivery service affect all customers equally? How could the distributor identify who it affects the most?

Answer: The company should orient themselves in delivering according to new features from the combined features. New features from combined features come from PCA analysis. Since PCA combines features, the new axis will contain all the variance and therefore it will give the company a new direction on how to organize their deliveries so that a 3 day delivery week is possible.

1.8.2 Question 11

Assume the wholesale distributor wanted to predict a new feature for each customer based on the purchasing information available. How could the wholesale distributor use the structure of the data to assist a supervised learning analysis?
Hint: What other input feature could the supervised learner use besides the six product features to help make a prediction?

Answer: The wholesale distributor should seek the variance in the data by PCA analysis to create new features of high variance. This will allow the features to be combined effectively to not only lower the data, but to understand customer habits more correctly

1.8.3 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 on 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 [78]: # Display the clustering results based on 'Channel' data
         rs.channel_results(reduced_data, outliers, pca_samples)
```

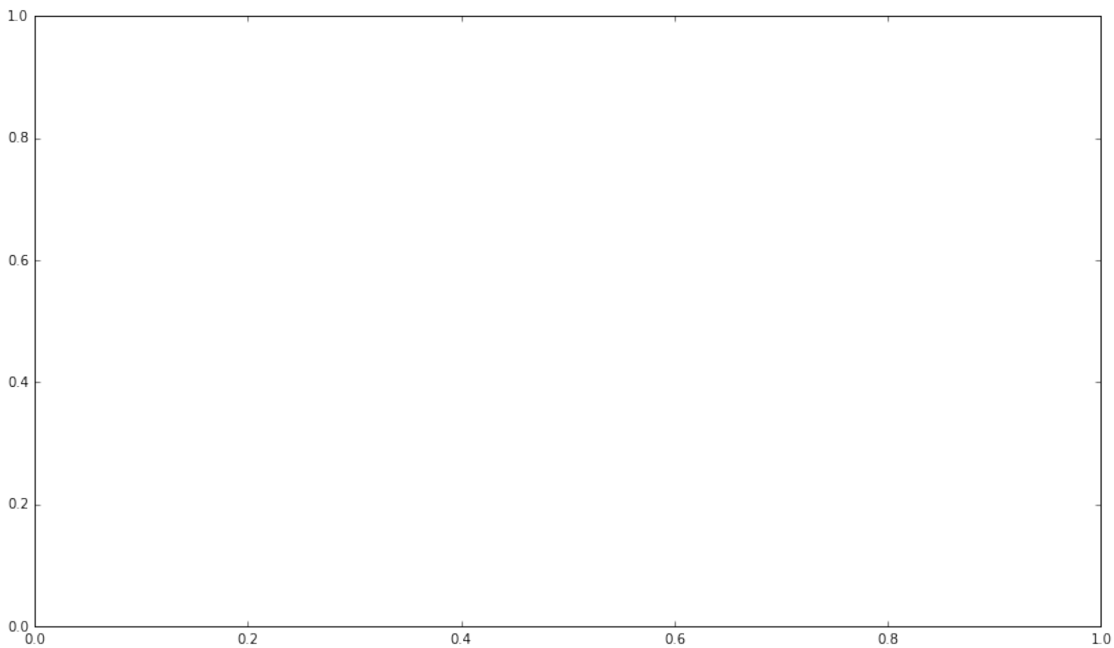
```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-78-461acdb1c7cd> in <module>()
          1 # Display the clustering results based on 'Channel' data
    ----> 2 rs.channel_results(reduced_data, outliers, pca_samples)


        /home/jobin/summer2016/ml-nanodegree/projects/customerSegment/renders.pyc i
        103         for i, channel in grouped:
        104             channel.plot(ax = ax, kind = 'scatter', x = 'Dimension 1',
    --> 105                          color = cmap((i-1)*1.0/2), label = labels[i-1]
        106
        107         # Plot transformed sample points


        TypeError: list indices must be integers, not numpy.float64
```



1.8.4 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 clustering algorithm works well in terms of pca analysis. Although if the variance is low across all features, the clustering system would not work well. Yes there would be customers considered pure retailers and or 'hotels/restaurants/cafes' since the axis built by PCA works in this manner of combining a select group of features with high variance together and low variance with unincluded features.

> **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.