In [141]:

```
#Libraries for the project
import numpy as np
import pandas as pd
from IPython.display import display # Allows the use of display() for DataFrames
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns
from plotnine import *

from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import  silhouette_score

import warnings
warnings.filterwarnings('ignore', category = FutureWarning)
warnings.filterwarnings('ignore', category = UserWarning)
```

In [69]:

```
#The dataset contains different product categories which are represented by columns and the rows r
epresent the annual
#spending amounts of the customers on diverese product categories. The aim of this project is to b
e able to identify
#variation in different types of customers so that the distributors can get an insight into how to
best structure their
#delivery service to meet the needs of each customer
```

In [70]:

```
# Load the wholesale customers dataset
try:
    cust_data = pd.read_csv("Wholesale customers data.csv")
    print("Wholesale customers dataset has {} samples with {} features each.".format(*data.shape))
except:
    print("Dataset could not be loaded. Is the dataset missing?")
```

```
Dataset could not be loaded. Is the dataset missing?
```

In [71]:

```
#Data Exploration
print(cust_data.head())

#Reomving columns "region" and "channel" as for this analysis this data wouldn't be helpful. To un
derstand different
#purchase bhevaiour of the customer we can focus on the categories of the food items.

cust_data.drop(['Region', 'Channel'], axis = 1, inplace = True)
```

```
   Channel  Region  Fresh  Milk  Grocery  Frozen  Detergents_Paper  Delicassen
0        2       3  12669  9656     7561     214              2674        1338
1        2       3   7057  9810     9568    1762              3293        1776
2        2       3   6353  8808     7684    2405              3516        7844
3        1       3  13265  1196     4221    6404               507        1788
4        2       3  22615  5410     7198    3915              1777        5185
```

In [72]:

```
cust_data.describe()
#If we pick out random rows from the data, by the help of the mean values we can identify what typ
e of customers could they be.
#From this we can get an understading of the distribution of the values.
```

Out[72]:

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

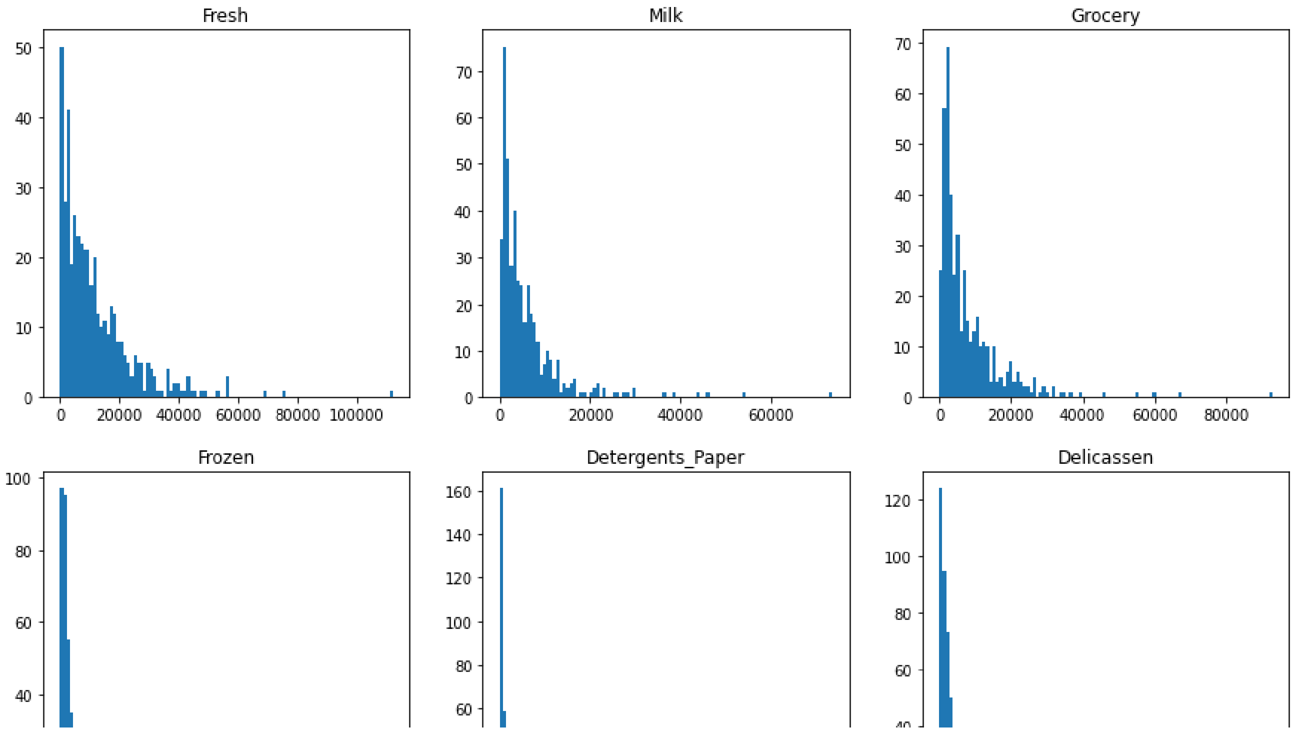
In [73]:

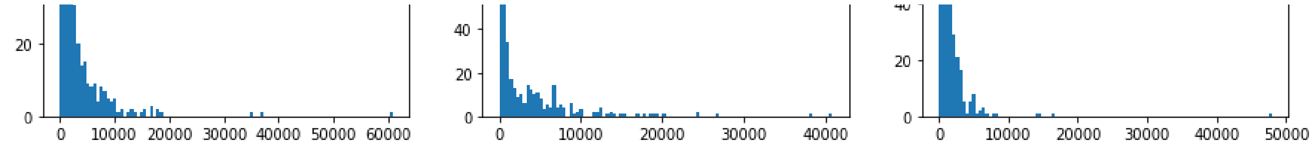
```
#Checking for null values and the datatype of the data
cust_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 6 columns):
Fresh               440 non-null int64
Milk                440 non-null int64
Grocery             440 non-null int64
Frozen              440 non-null int64
Detergents_Paper    440 non-null int64
Delicassen          440 non-null int64
dtypes: int64(6)
memory usage: 20.7 KB
```

In [74]:

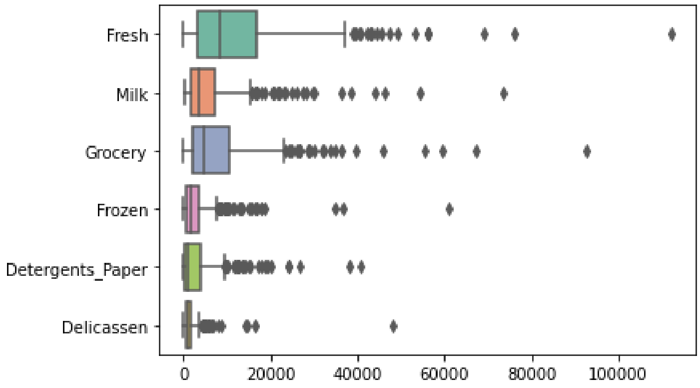
```
#Displaying distributions of various categories
features = cust_data.columns.values

fig = plt.figure(figsize=(15,10))
for i in range(len(features)):
    ax = fig.add_subplot(2,3,i+1)
    ax.set_title(features[i])
    ax.hist(cust_data[features[i]], bins = 100)
plt.show()
```





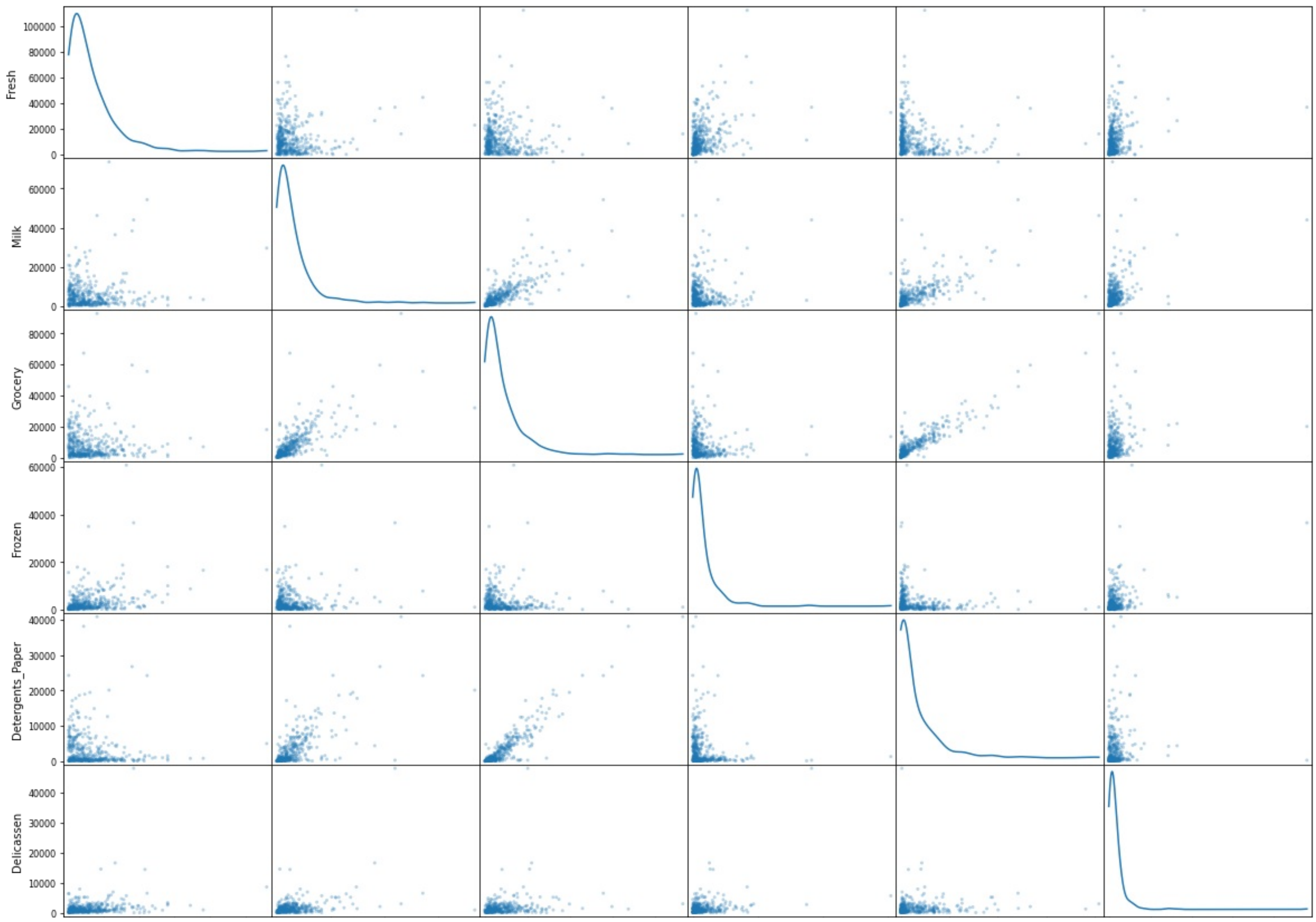
In [77]:

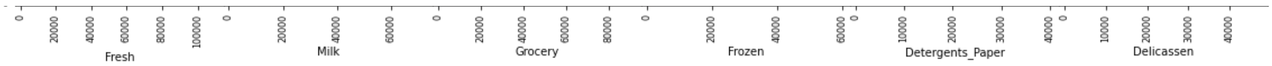
```
#Boxplots are helfpul for us to understand and identify any outliers in the data.
#From the below plots we can see that there are many outliers within different categories. The out
liers also point towards the
#data not being highly correlated
ax = sns.boxplot(data=cust_data, orient="h", palette="Set2")
```



In [78]:

```
#Plotting a scatterplot
#The scatter matrix might show a correlation between features within the data.
scatter_matrix(cust_data, alpha = 0.3, figsize =(20,15), diagonal = 'kde');

#From the plot we can see that the data is not normally distributed and it indicates towards the d
ata being largely skewed
#which we can also observe from the boxplot.
```



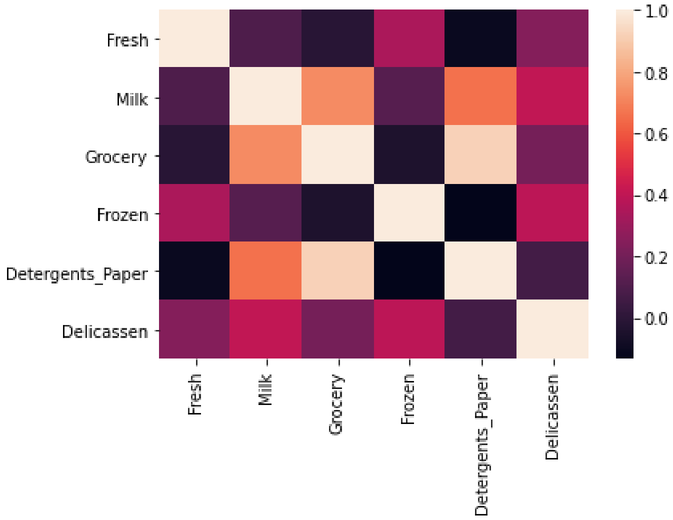


In [79]:

```
cust_data.corr()
sns.heatmap(cust_data.corr())

#From the correlation matrix below we can identify which features are highly correlated
```

Out[79]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x260c5a34dd8>
```



In [80]:

```
#This is the progress so far in terms of visualisations. As the data was numerical, making histogr
ams and seeing
#distributions was a better way to visualise the data.
```

Implementing Feature Relevance

It is important to understand if a customer purchase is relevant to the categories, i.e a customer is purchasing some amount of a category is highly likely to purchase some proportional amount of another category.

To understand this we can build a suprvised model on a subset of the dataset. We can remove one category from it and set it as a target variable to see if we are able to predict it using other categories.

I am trying to predict category - "Detergents_Paper" using other categories

In [81]:

```
# TODO: Make a copy of the DataFrame, using the 'drop' function to drop the given feature
features = cust_data[['Grocery', 'Fresh', 'Milk', 'Frozen', 'Delicassen']]
labels = cust_data[['Detergents_Paper']]

#Splitting the dataset into train and test data
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size = .25, random_state
= 0)

#Building a Decision Tree Regressor Model
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(X_train,y_train)

#Calculating the prediction score
score = regressor.score(X_test, y_test)
print(score)
```

```
0.7230214638403756
```

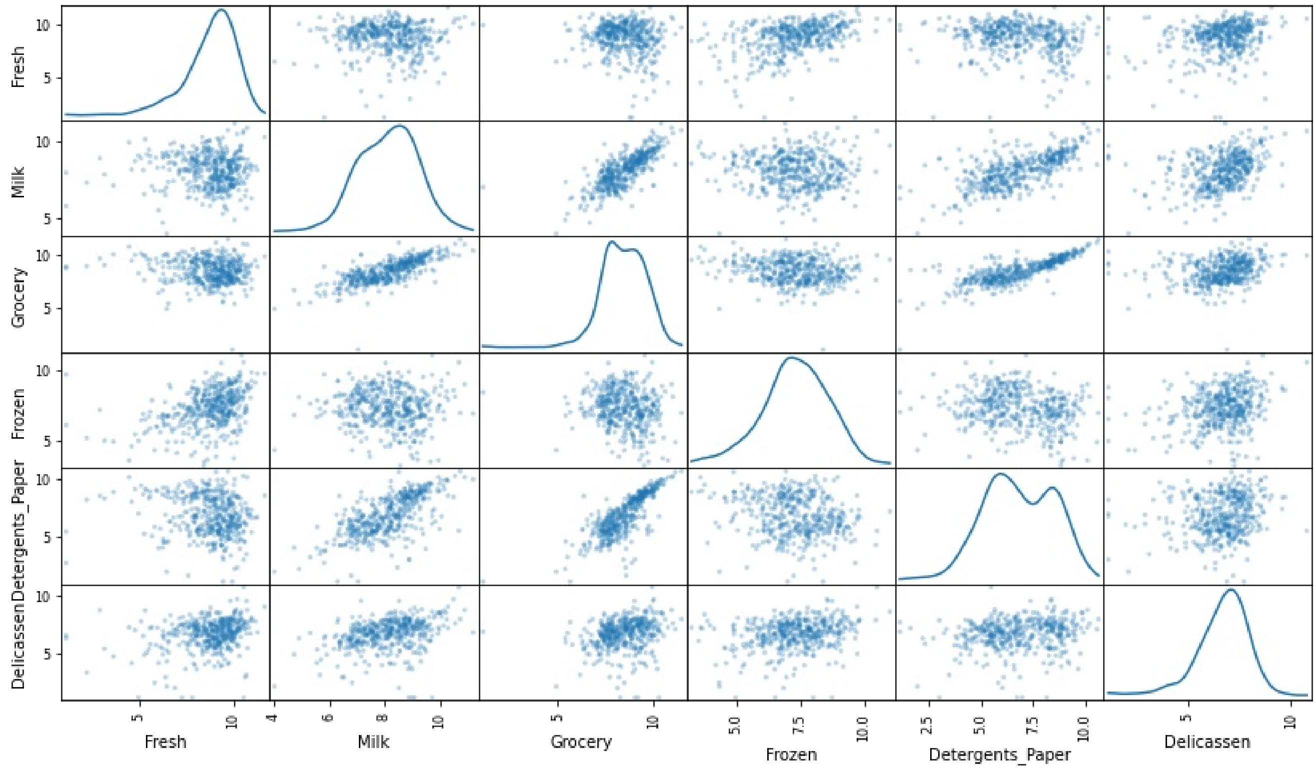
Since we are able to predict "Detergents_Paper" using other categories, it seems that it's not realy relevant for the prediction of customers' spending habits.

Implementing Feature Scaling

As seen from the visualisations above the data is not normally distributed, we can improve this by using a log function to scale the data set

In [82]:

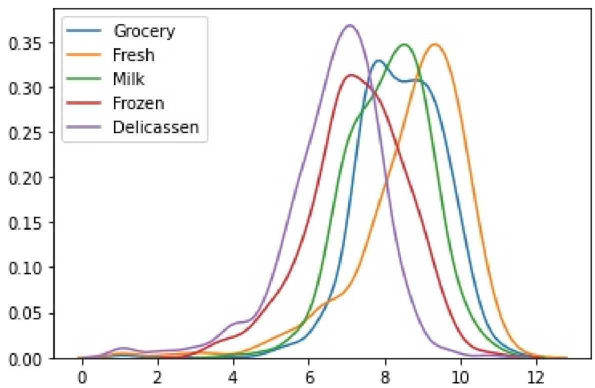
```
# Scaling the data using using natural logarithmn
log_data = np.log(cust_data)

# Scatter matrix for each pair of the transformed features
pd.plotting.scatter_matrix(log_data, alpha = 0.3, figsize = (14,8), diagonal = 'kde');
```



After natural log scaling we can see a little bit of normal distribution in the visualisation amongst the variables

In [83]:

```
for i in features:
    sns.kdeplot(log_data[i])
```



It is important to remove outliers in the dataset if any. There are various methods to do so but for my calculation I am going to use "Tukey's Method for identfying outliers"

In [91]:

```
outliers = pd.DataFrame()

#Finding extreme high and low points for all the features
for feature in log_data.keys():
    #Calculating 25th percentile of the data
    Q1 = np.percentile(log_data[feature], 25)
    #Calculate 75th percentile of the data
    Q3 = np.percentile(log_data[feature], 75)
    #Using the interquartile range to calculate an outlier step (1.5 times the interquartile 
range)
    step = 1.5* (Q3 - Q1)
    #Displaying the outliers

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

duplicates = outliers.groupby(level=0).filter(lambda x: len(x) > 1)

# Removing the outliers which are present in more than one category
Updated_cust_data = log_data.drop(log_data.index[duplicates.index]).reset_index(drop = True)
```

```
Data points considered outliers for the feature 'Fresh':


65     4.442651
66     2.197225
81     5.389072
95     1.098612
96     3.135494
128    4.941642
171    5.298317
193    5.192957
218    2.890372
304    5.081404
305    5.493061
338    1.098612
353    4.762174
355    5.247024
357    3.610918
412    4.574711
Name: Fresh, dtype: float64


Data points considered outliers for the feature 'Milk':


86     11.205013
98      4.718499
154     4.007333
356     4.897840
Name: Milk, dtype: float64


Data points considered outliers for the feature 'Grocery':


75     1.098612
154    4.919981
Name: Grocery, dtype: float64


Data points considered outliers for the feature 'Frozen':


38      3.496508
57      3.637586
65      3.583519
145     3.737670
175     3.951244
264     4.110874
325    11.016479
```

```
420      3.218876
429      3.850148
439      4.174387
Name: Frozen, dtype: float64


Data points considered outliers for the feature 'Detergents_Paper':


75      1.098612
161     1.098612
Name: Detergents_Paper, dtype: float64


Data points considered outliers for the feature 'Delicassen':


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
Name: Delicassen, dtype: float64
```

Using Feature Transformation

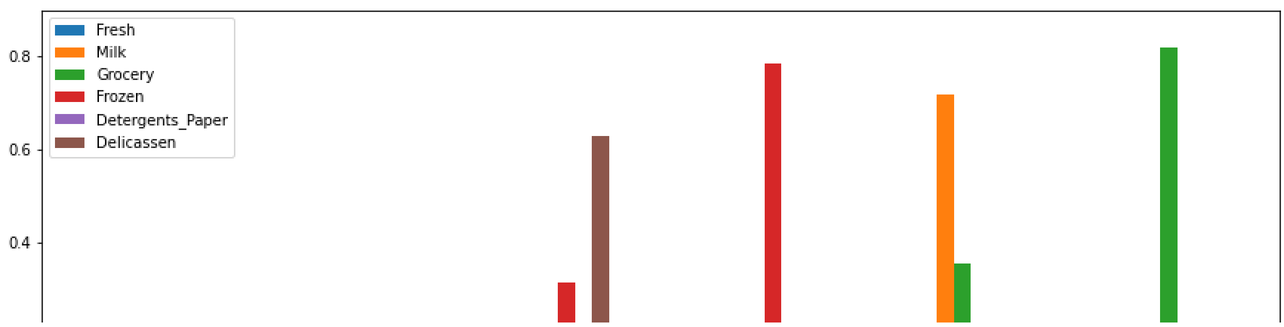
We will use Principal component analysis (PCA) to draw conclusions about the underlying structure of the wholesale customer data. Principal Component Analysis (PCA) is used to explain the variance-covariance structure of a set of variables through linear combinations. Hence by using PCA we will find which compound combinations of features which best describe customers.

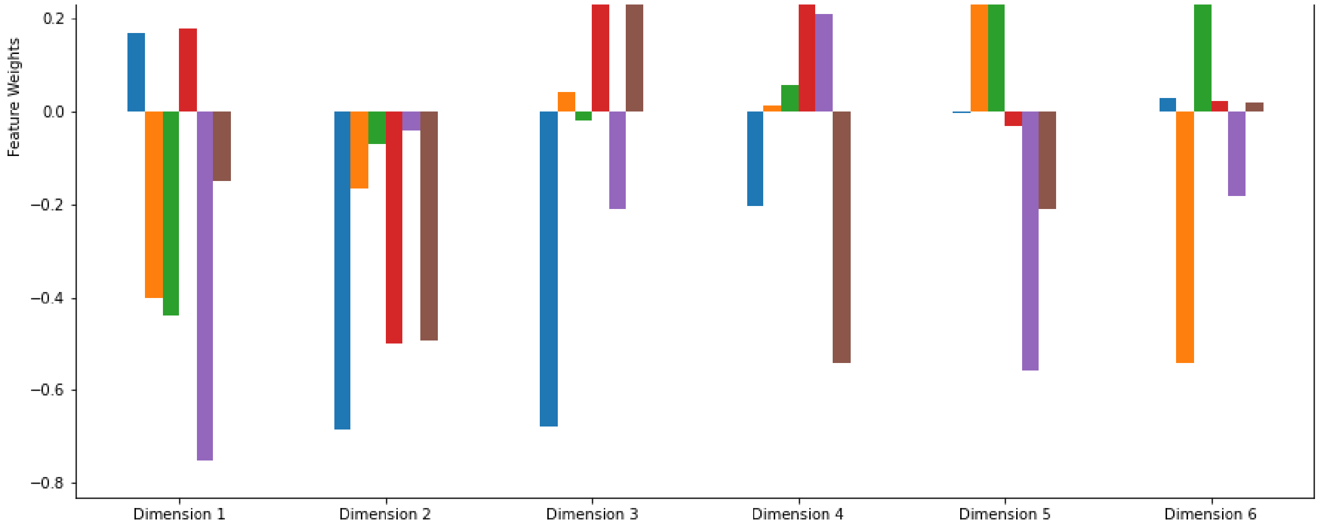
In [149]:

```python
pca = PCA(n_components = 6)
pca.fit(Updated_cust_data)

dimensions = dimensions = ['Dimension {}'.format(i) for i in range(1,len(pca.components_)+1)]
components = pd.DataFrame(np.round(pca.components_, 4), columns = list(Updated_cust_data.keys()))
components.index = dimensions

fig, ax = plt.subplots(figsize = (15,10))

components.plot(ax = ax, kind = 'bar');
ax.set_ylabel("Feature Weights")
ax.set_xticklabels(dimensions, rotation=0)


pca.explained_variance_ratio_
```

Out[149]:

```
array([0.44302505, 0.26379218, 0.1230638 , 0.10120908, 0.04850196,
       0.02040793])
```





Dimension 1 and Dimension 2 covers almost 0.7 of the data's variance.

From the plot, if we take a look at the magnitude of weights in Dimension 1 and Dimension 2, we can identify two different set of features which we can use to separate customer segments: one defined by Detergent_paper, Grocery, and Milk (PC 1) and another defined by Fresh, Milk, Frozen (PC 2)

Dimensionality reduction

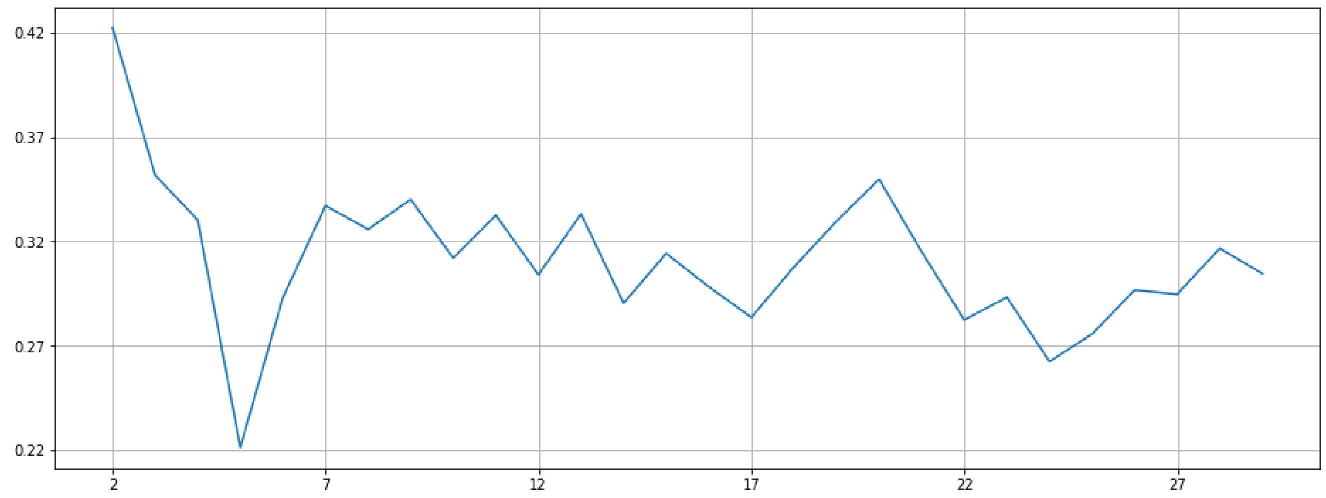
The main idea of principal component analysis is to reduce the dimensionality of a data set consisting of many variables correlated with each other while retaining the variation present in the dataset. It helps to take a dataset with a high number of dimensions and compresses it to a dataset with fewer dimensions,while capturing most variance within the original data.

In [150]:

```
n_component = 2
pca = PCA(n_components=n_component).fit(Updated_cust_data)

# Transform the good data using the PCA fit above
reduced_cust_data =  pca.transform(Updated_cust_data)

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

In [151]:

```
#Using Gaussian Mixture to identify the number of clusters
#Note : Code referenced from various websites

def clustering_errors(k, data):
    gmm = GaussianMixture(n_components=k, init_params = 'kmeans').fit(data)
    predictions = gmm.predict(data)
    silhouette_avg = silhouette_score(data, predictions)
    return silhouette_avg

X = reduced_cust_data
max_clusters = 30
possible_k_values = range(2, max_clusters)
errors_per_k = [clustering_errors(k, X) for k in possible_k_values]
fig, ax = plt.subplots(figsize=(16, 6))
plt.plot(possible_k_values, errors_per_k)

# Ticks and grid
xticks = np.arange(min(possible_k_values), max(possible_k_values)+1, 5.0)
ax.set_xticks(xticks, minor=False)
ax.set_xticks(xticks, minor=True)
ax.xaxis.grid(True, which='both')
yticks = np.arange(round(min(errors_per_k), 2), max(errors_per_k), .05)
ax.set_yticks(yticks, minor=False)
ax.set_yticks(yticks, minor=True)
ax.yaxis.grid(True, which='both')
```



From the visualisation above we can see that the optimal number of clusters is 2

Performing K means Clustering

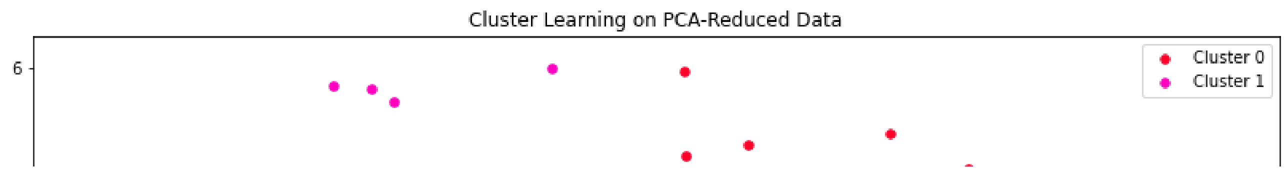
In [132]:

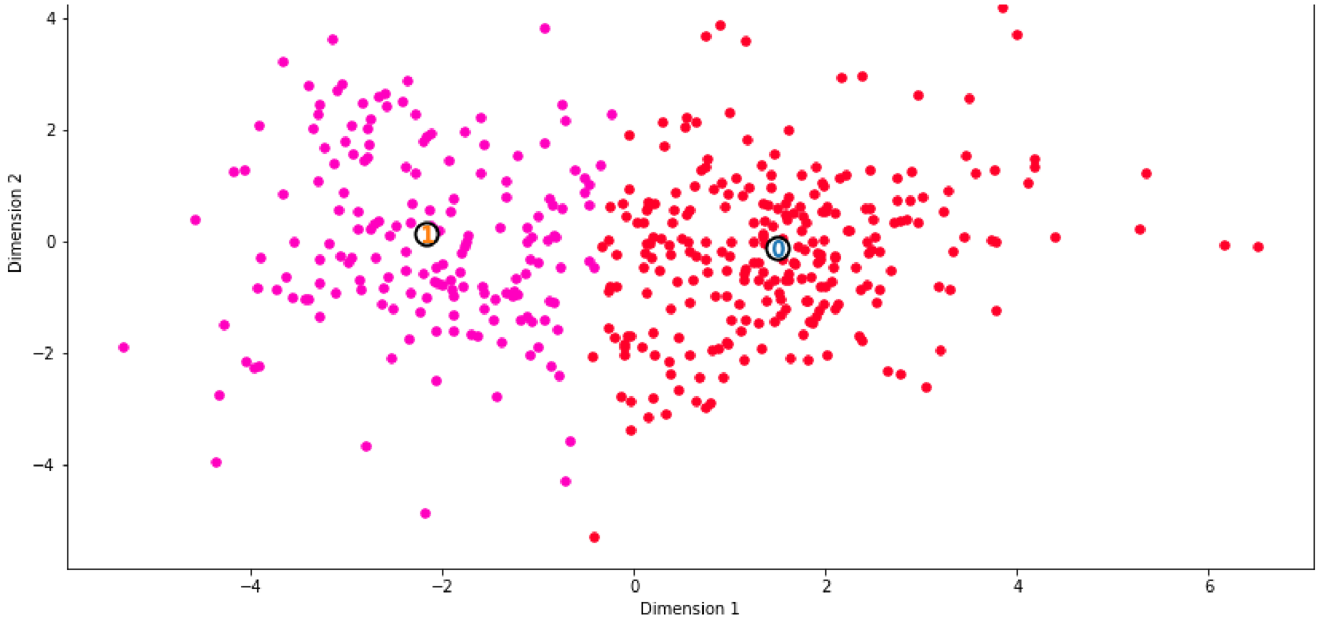
```
clusterer = KMeans(n_clusters=2, random_state=0).fit(reduced_data)

#Predicting cluster for each data point
preds = clusterer.predict(reduced_cust_data)

#Identifying cluster centers
centers = clusterer.cluster_centers_

#Calculating mean silhouette coefficient for the number of clusters chosen
score = silhouette_score(reduced_cust_data, preds)
print(score)
```

```
0.42628101546910835
```

In [148]:

```
predictions = pd.DataFrame(preds, columns = ['Cluster'])
plot_data = pd.concat([predictions, reduced_cust_data], axis = 1)

# Generate the cluster plot
fig, ax = plt.subplots(figsize = (14,8))

# Color map
cmap = cm.get_cmap('gist_rainbow')

# Color the points based on assigned cluster
for i, cluster in plot_data.groupby('Cluster'):
    cluster.plot(ax = ax, kind = 'scatter', x = 'Dimension 1', y = 'Dimension 2', \
                 color = cmap((i)*1.0/(len(centers)-1)), label = 'Cluster %i'%(i), s=30);

# Plot centers with indicators
for i, c in enumerate(centers):
    ax.scatter(x = c[0], y = c[1], color = 'white', edgecolors = 'black', \
               alpha = 1, linewidth = 2, marker = 'o', s=200);
    ax.scatter(x = c[0], y = c[1], marker='$%d$'%(i), alpha = 1, s=100);

# Set plot title
ax.set_title("Cluster Learning on PCA-Reduced Data");
```





The two different color in the above visualisation represents individual clusters and they both have a cetrol point. The centre point is the average of all data points predicted in the respctive clusters. The cluster's center point relates to the average customer of that segment. We can get the customer spending from these data points by applying inverse transformations.

In [ ]: