UnSupervised Learning Algorithm -Creating Customer Segments from Spending Habits

Understanding the wholesale customer dataset and the segmentation problem.

The UCI Machine Learning Repository offers the wholesale customer dataset at https://archive.ics.uci.edu/ml/datasets/wholesale+customers (https://archive.ics.uci.edu/ml/datasets/wholesale+customers). The dataset refers to clients of a wholesale distributor. It includes the annual spending in monetary units (m.u.) on diverse product categories. The goal of these projects is to apply clustering techniques to identify segments that are relevant for certain business activities, such as rolling out a marketing campaign.Being able to categorize customers into meaningful groups (customer segmentation) based on spending habits is valuable for businesses. Doing so can give a business insight regarding how to best meet the specific needs of different groups of customers.An unsupervised learning algorithm was used to create create meaningfully different groups of customers. Specifically, Principle Component Analysis was used for dimensionality reduction of the features down to just two.K-Means is used to cluster customers into differnt groups.

In [31]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

Attribute Information:

1) FRESH: annual spending (m.u.) on fresh products (Continuous); 2) MILK: annual spending (m.u.) on milk products (Continuous); 3) GROCERY: annual spending (m.u.)on grocery products (Continuous); 4) FROZEN: annual spending (m.u.)on frozen products (Continuous) 5) DETERGENTS_PAPER: annual spending (m.u.) on detergents and paper products (Continuous) 6) DELICATESSEN: annual spending (m.u.)on and delicatessen products (Continuous); 7) CHANNEL: customersâ€™ Channel - Horeca (Hotel/Restaurant/CafÃ©) or Retail channel (Nominal) 8) REGION: customersâ€™ Region â€“ Lisnon, Oporto or Other (Nominal)

In [32]:
```
data = pd.read_csv("data/Wholesale_Customers_Data.csv")
```

In [33]:
```
print(data.shape)
```

```
(440, 8)
```

In [34]:
```
#Print no of integers, floats and strings
data.dtypes.value_counts()
```

Out[34]:
```
int64    8
dtype: int64
```

In [35]:
```
#Data should be numerical
data.head()
```

Out[35]:

	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

Preprocessing Steps

1. Select Features and apply feature tranformation/scaling.

The Wholesale Customers Dataset from the UCI Machine Learning Repository contains the annual spending habits of 440 clients of a whole sale distributor, covering a diverse set of product categories.

Since the purpose of this project is to discover if meaningful clusters can be discovered from only the spending habits of customers, two variables the 'Channel' and 'Region' can be removed.

In [36]:
```
data = data.drop(['Channel', 'Region'], axis=1)
```

In [37]:
```
# Convert to floats
for col in data.columns:
    data[col] = data[col].astype(np.float)
```

In [38]:
```
#While doing PCA if two features are highly corelated its not adding any ex
#we may want to reduce/remove/combine into two without losing much variance

# The correlation matrix
corr_mat = data.corr()

#Every feature with itself will have correlation of one and we need to remo
for x in range(corr_mat.shape[0]):
    corr_mat.iloc[x,x] = 0.0

corr_mat
```

Out[38]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Fresh	0.000000	0.100510	-0.011854	0.345881	-0.101953	0.244690
Milk	0.100510	0.000000	0.728335	0.123994	0.661816	0.406368
Grocery	-0.011854	0.728335	0.000000	-0.040193	0.924641	0.205497
Frozen	0.345881	0.123994	-0.040193	0.000000	-0.131525	0.390947
Detergents_Paper	-0.101953	0.661816	0.924641	-0.131525	0.000000	0.069291
Delicassen	0.244690	0.406368	0.205497	0.390947	0.069291	0.000000

While doing PCA if two features are highly correlated its not adding any extra information and we can combine orremove those features without loosing much variance

In [39]:
```
# which values are highly correlated?
corr_mat.abs().idxmax()
```

Out[39]:
```
Fresh                        Frozen
Milk                       Grocery
Grocery            Detergents_Paper
Frozen                   Delicassen
Detergents_Paper            Grocery
Delicassen                     Milk
dtype: object
```

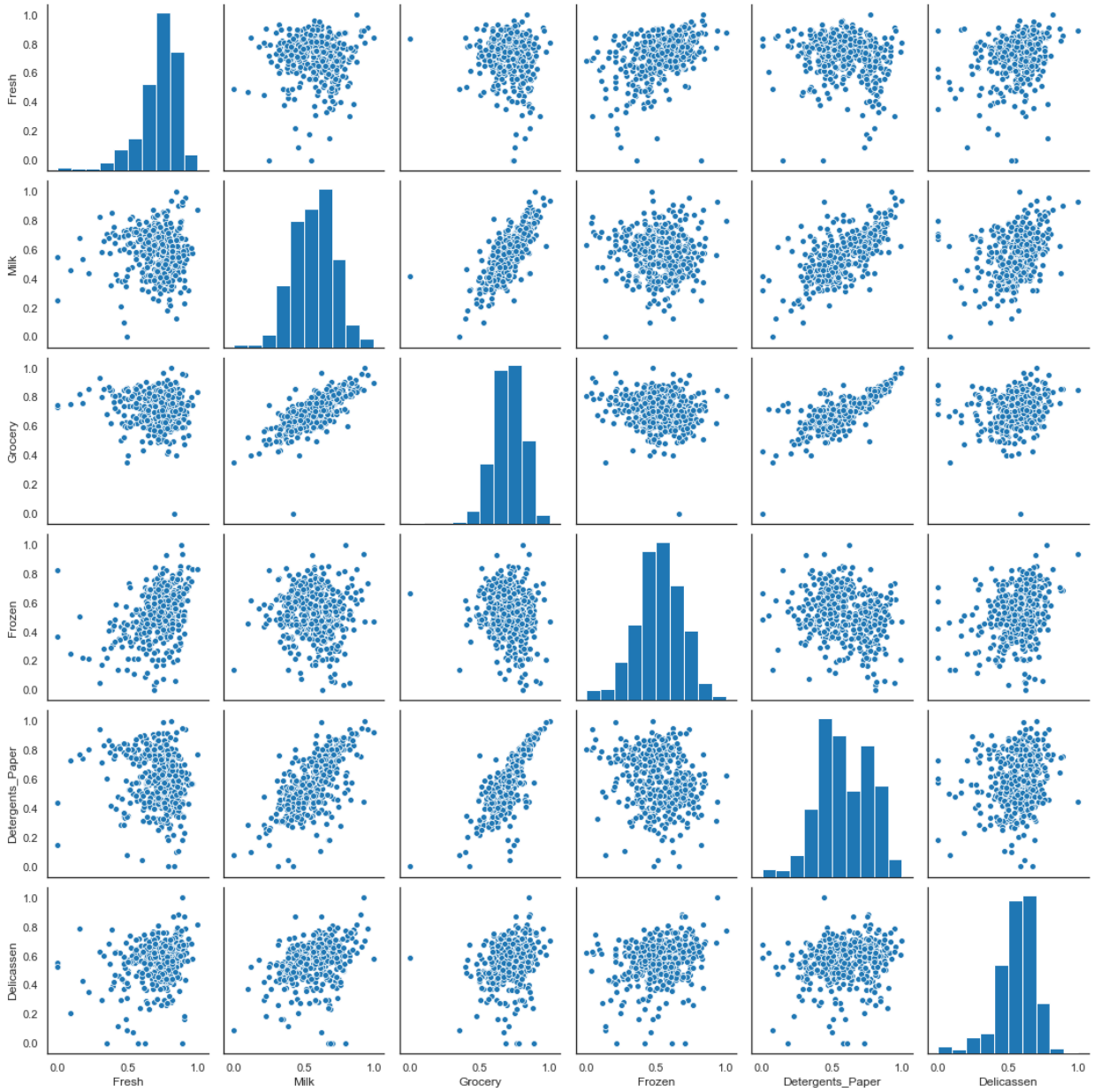
In [40]:
```
#Calculate Skew Vlaues
#0- no skew
#+ve - right skew
#-Ve - left skew
skew_columns = (data.skew().sort_values(ascending=False))
skew_columns
```

Out[40]:
```
Delicassen          11.151586
Frozen               5.907986
Milk                 4.053755
Detergents_Paper     3.631851
Grocery              3.587429
Fresh                2.561323
dtype: float64
```

In [41]:
```
#Getting Skewed Columns and log tranforming it.
skew_columns = skew_columns.loc[skew_columns > 0.75]
# Perform log transform on skewed columns
for col in skew_columns.index.tolist():
    data[col] = np.log1p(data[col])
```

In [42]:
```
#Ensure all values are in same scale.
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
for col in data.columns:
    data[col] = mms.fit_transform(data[[col]]).squeeze()
```

In [43]:
```
#Visualization which ones are highly corealted .
import seaborn as sns
sns.set_context('notebook')
sns.set_style('white')
sns.pairplot(data);
```



Dimensionality Reduction

In order to make the clustering more interpretable, and easy to visualize, we can perform a transformation on the data to reduce the most important information to just two dimensions. We will use Principal Component Analysis (PCA) to do this. PCA will essentially transform the data into a new coordinate system. The dimensions of this new coordinate system are ordered in terms of how much variance they are capable of explaining. This allows us to select the first dimensions, which will contain the most predictive power. PCA will also report the explained variance ratio of each dimension — that is, how much variance within the data is explained by that dimension alone.

In [62]:
```
from sklearn.decomposition import PCA
PCAmod = PCA(n_components=2)
PCAmod.fit(data)
print("Proportion of the variance explained by each PCA Components ")
PCAmod.explained_variance_ratio_
```

```
Proportion of the variance explained by each PCA Components
```

Out[62]:
```
array([0.44801067, 0.27297911])
```

First Principal component contributes to 44% variance and second one 27% and third one 10% . Usually first will contribute more and then seoncond ...

In [63]:
```
#Principal Componets
PCAmod.components_
```

Out[63]:
```
array([[ 0.12715368, -0.51602135, -0.39903088,  0.20132122, -0.70370418,
        -0.15032783],
       [-0.51077808, -0.20541903, -0.05282381, -0.69475861, -0.01227765,
        -0.45965003]])
```

In [64]:
```
abs_feature_values = np.abs(PCAmod.components_).sum(axis=0)
abs_feature_values
```

Out[64]:
```
array([0.63793175, 0.72144038, 0.45185469, 0.89607983, 0.71598183,
       0.60997787])
```

In [65]:
```
feature_weights = abs_feature_values/abs_feature_values.sum()
feature_weights
```

Out[65]:
```
array([0.15816752, 0.17887249, 0.11203195, 0.22217224, 0.1775191 ,
       0.15123669])
```

In [68]:
```
# Apply a PCA transformation the good data
reduced_data = PCAmod.transform(data)
# Create a DataFrame for the reduced data
reduced_data = pd.DataFrame(reduced_data, columns = ['PCA 1', 'PCA 2'])
reduced_data
```

Out[68]:

	PCA 1	PCA 2
0	-0.228732	0.077381
1	-0.212276	-0.097680
2	-0.217829	-0.188815
3	0.155115	-0.178031
4	-0.093180	-0.260302
...	...	...
435	0.039309	-0.364515
436	0.338684	-0.207908
437	-0.433506	-0.032555
438	0.174947	-0.020556
439	0.073455	0.474499

440 rows × 2 columns

In [69]:
```
pca_list = list()
feature_weight_list = list()

# Fit a range of PCA models

for n in range(1, 6):

    # Create and fit the model
    PCAmod = PCA(n_components=n)
    PCAmod.fit(data)

    # Store the model and variance
    pca_list.append(pd.Series({'n':n, 'model':PCAmod,
                               'var': PCAmod.explained_variance_ratio_.sum(

    # Calculate and store feature importances
    abs_feature_values = np.abs(PCAmod.components_).sum(axis=0)
    feature_weight_list.append(pd.DataFrame({'n':n,
                                             'features': data.columns,
                                             'values':abs_feature_values/ab

pca_df = pd.concat(pca_list, axis=1).T.set_index('n')
pca_df
```

Out[69]:

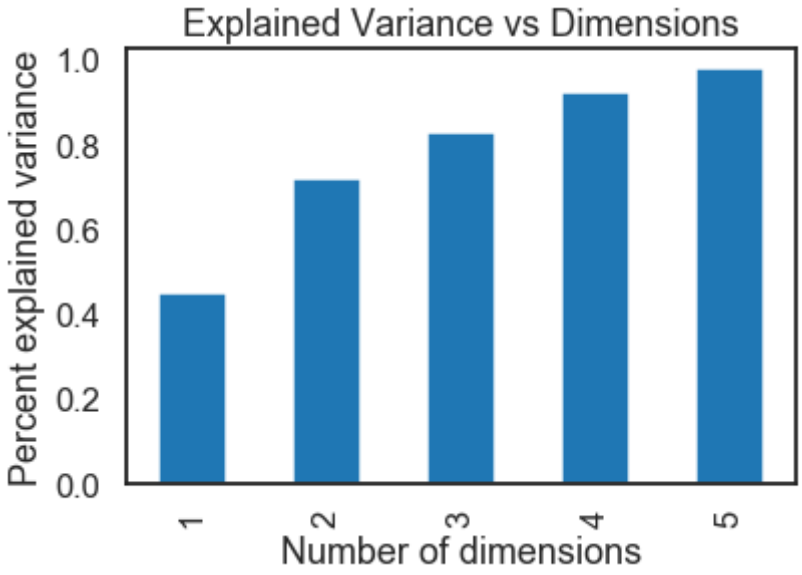
n	model	var
1	PCA(copy=True, iterated_power='auto', n_compon...	0.448011
2	PCA(copy=True, iterated_power='auto', n_compon...	0.72099
3	PCA(copy=True, iterated_power='auto', n_compon...	0.827534
4	PCA(copy=True, iterated_power='auto', n_compon...	0.923045
5	PCA(copy=True, iterated_power='auto', n_compon...	0.979574

In [70]:
```
#how much each features contributed to each PCA.
features_df = (pd.concat(feature_weight_list)
               .pivot(index='n', columns='features', values='values'))

features_df
```

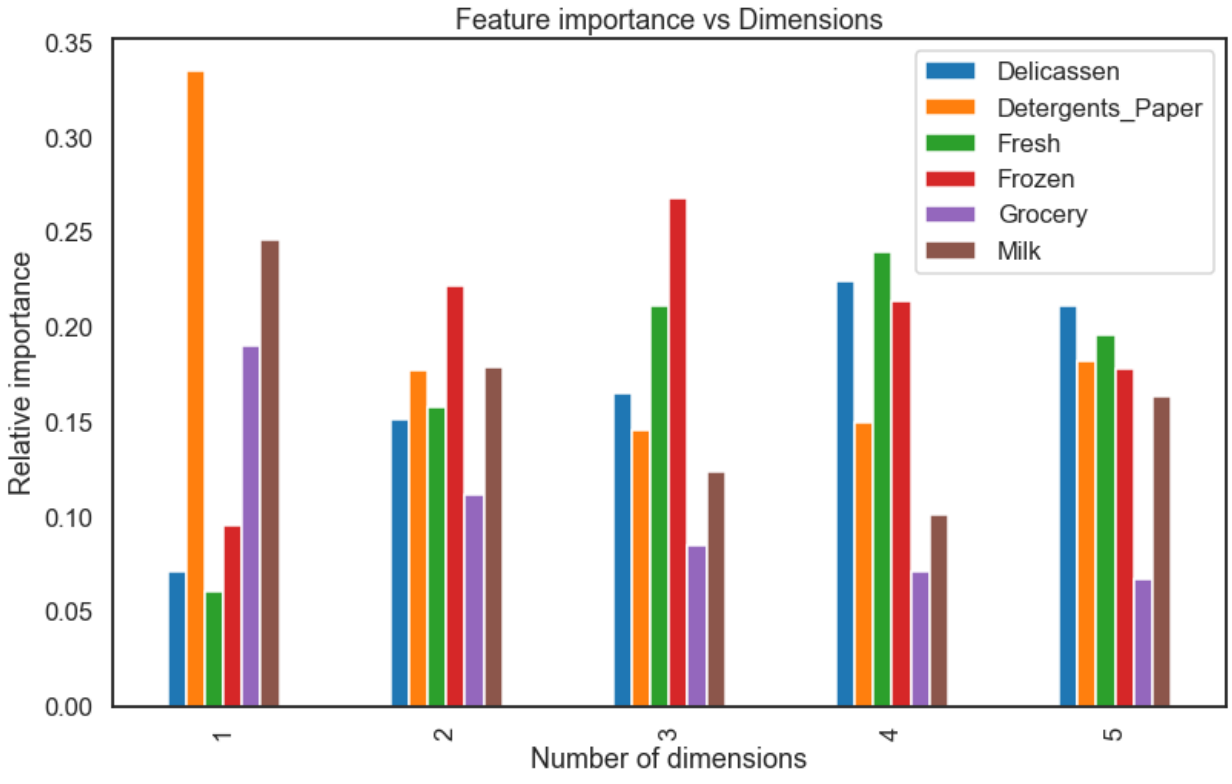
Out[70]:

features	Delicassen	Detergents_Paper	Fresh	Frozen	Grocery	Milk
n						
1	0.071668	0.335487	0.060620	0.095979	0.190236	0.246010
2	0.151237	0.177519	0.158168	0.222172	0.112032	0.178872
3	0.165518	0.145815	0.211434	0.268363	0.084903	0.123967
4	0.224259	0.149981	0.239527	0.214275	0.070971	0.100987
5	0.211840	0.182447	0.196382	0.178104	0.067338	0.163888

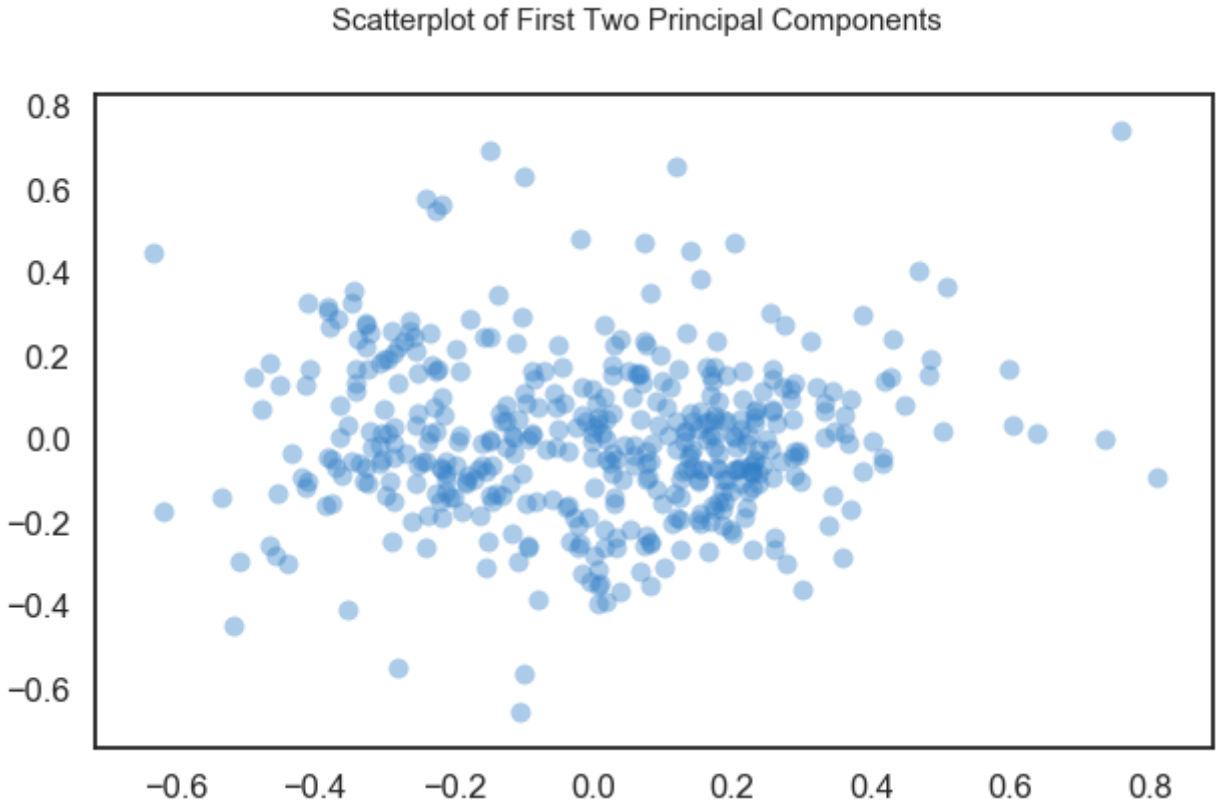
In [71]:
```
sns.set_context('talk')
ax = pca_df['var'].plot(kind='bar')

ax.set(xlabel='Number of dimensions',
       ylabel='Percent explained variance',
       title='Explained Variance vs Dimensions');
```



In [72]:
```
ax = features_df.plot(kind='bar', figsize=(13,8))
ax.legend(loc='upper right')
ax.set(xlabel='Number of dimensions',
       ylabel='Relative importance',
       title='Feature importance vs Dimensions');
```



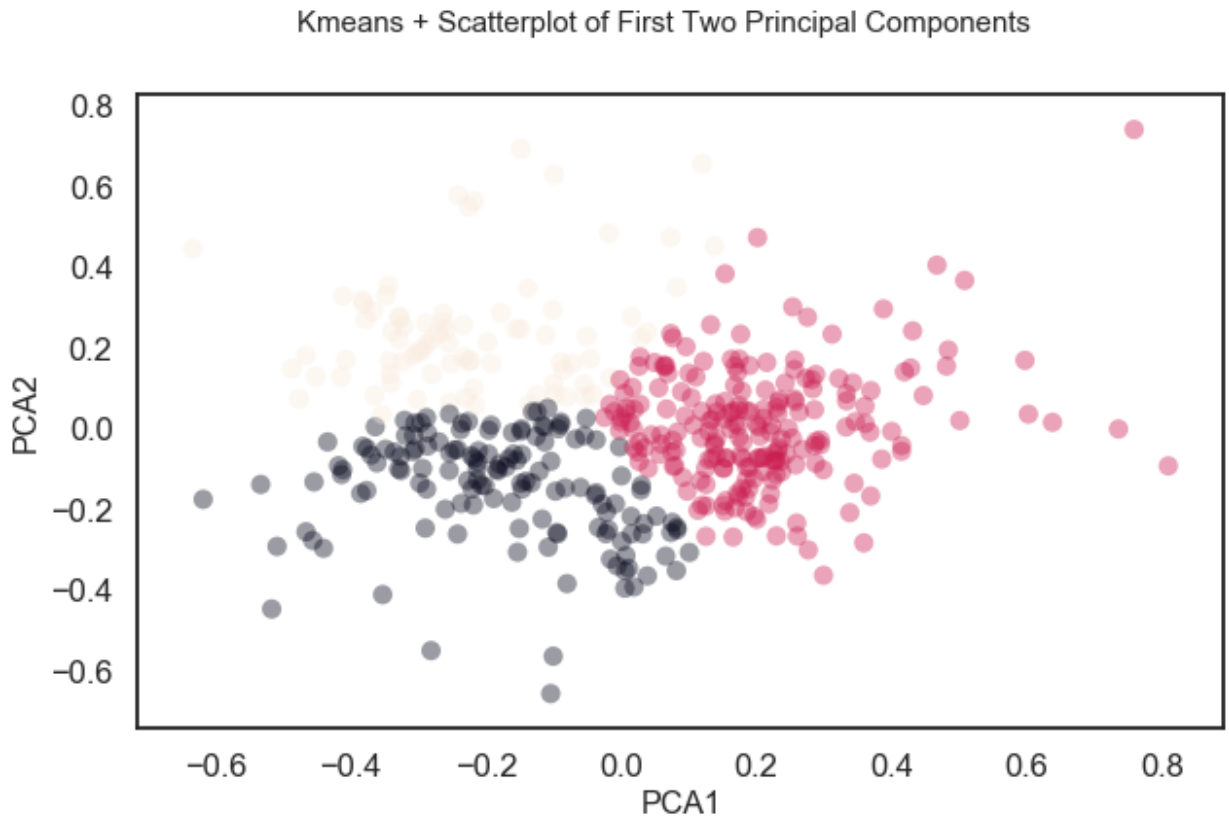
In [73]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 6))
fig.suptitle('Scatterplot of First Two Principal Components', fontsize=15)
img = ax.scatter(reduced_data["PCA 1"], reduced_data["PCA 2"],
                 c="#307EC7", s=100, alpha=0.4, linewidths=0)
```



In [75]:
```
from sklearn.cluster import KMeans
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(reduced_data)
labels = k_means.labels_
print(labels)
```

```
[2 0 0 1 0 0 2 0 2 0 0 1 0 0 0 1 2 0 0 2 0 1 0 0 0 2 1 1 0 1 0 1 1 0 1 2
0
 0 2 1 0 0 2 2 2 0 0 0 0 0 1 2 0 2 1 0 0 2 1 2 2 0 0 0 1 2 2 0 0 1 1 0 1
0
 0 1 1 0 1 2 1 2 0 1 2 0 0 0 1 0 1 1 0 1 2 2 2 1 1 1 0 0 0 0 1 1 2 0 2 2
1
 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 2 1 1 1 1 1 1 1 0 2 1 1 0 1 1 1 2 2 1
1
 1 1 1 1 1 1 1 2 2 1 0 2 2 1 1 0 0 0 0 2 1 1 2 2 2 2 1 2 0 1 1 1 0 0 2 0
2
 1 1 2 0 2 1 1 1 2 1 0 0 0 1 1 0 0 0 2 1 2 1 2 0 0 1 0 1 0 2 0 2 1 2 1 1
2
 1 1 1 1 0 1 1 1 0 0 1 2 1 2 1 1 1 0 0 1 1 0 0 2 1 1 1 1 1 0 1 0 0 1 1 0
0
 0 2 1 1 1 2 0 0 1 0 1 1 1 1 1 1 1 0 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 0 1
2
 1 0 0 1 0 0 2 2 2 2 0 1 1 2 1 1 2 1 1 0 1 2 1 0 1 1 1 1 1 0 1 1 1 1 1 0
1
 2 1 0 1 1 1 1 2 0 2 2 1 2 0 0 1 0 1 0 1 2 0 1 1 2 0 2 1 1 1 2 1 0 1 1 1
1
 0 1 1 0 1 1 0 1 1 2 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 2 0 1
0
 0 0 0 0 1 2 0 1 0 0 2 2 0 2 0 1 0 0 1 0 0 1 1 1 0 1 1 0 0 1 0 1 2]
```

In [79]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 6))
fig.suptitle('Kmeans + Scatterplot of First Two Principal Components', font
ax.scatter(reduced_data["PCA 1"],
                reduced_data["PCA 2"],
                c=labels.astype(np.float), s=100, alpha=0.4, linewidths=0)
plt.xlabel('PCA1', fontsize=16)
plt.ylabel('PCA2', fontsize=16)
plt.show()
```



Summary

We can assign weights to each PCA compoents and ensure that each feature gets different weights in each PCA.

Once dimensions are reduced we apply any clustering algorithm.

In [ ]:
```

```